Universidade Federal de Campina Grande
Centro de Ciências e Tecnologia
Departamento de Engenharia Elétrica

RELATÓRIO DO ESTÁGIO

Aluno: Jadsonlee da Silva Sá
Matrícula: 29921229
Orientador: José Sérgio da Rocha Neto

Período: Dezembro/2004 à Fevereiro/2005

Biblioteca Setorial do CDSA. Fevereiro de 2021.

Sumé - PB

Título:

Processador Digital de Sinais TMS320LF2407A

Aluno:

Jadsonlee da Silva Sá

Orientador:

José Sérgio da Rocha Neto

Campina Grande
Fevereiro de 2005

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Os Processadores Digitais de Sinais (DSP) [1] são processadores especializados para o processamento de sinais. Eles possuem uma arquitetura especializada que leva em conta os tipos de aplicações, de dados e operações aritméticas que são mais comumente utilizadas. Para realizar essas operações de forma eficiente estes processadores possuem estruturas mais modernas e eficientes de memória, barramento e instruções, que tornam seu desempenho muito superior aos microprocessadores de uso geral. Os DSPs são utilizados em diversas áreas, tais como biomedicina, telefonia celular, sismologia, processamento de áudio e vídeo e outras áreas [2].

Existem dezenas de famílias de DSPs disponíveis no mercado. As principais famílias de DSPs e algumas de suas características estão listadas na tabela 1.2

Tabela 1.1: Principais famílias de DSP e algumas de suas características.

Fabricante	**Família**	**Aritmética**	**Comprimento dos Dados**	**Taxas (MIPS)**
Analog Device	ADSP-21xx	Fixo	16	33,3
Analog Device	ADSP-210xx	Flutuante	32	40,0
ATT	DSP16xx	Fixo	16	70,0
ATT	DSP32xx	Flutuante	32	20,0
Motorola	DSP5600xx	Fixo	24	40,0
Motorola	DSP561xx	Fixo	16	30,0
Motorola	DSP563xx	Fixo	24	80,0
Motorola	DSP96002	Flutuante	32	20,0
NEC	μPD7701x	Fixo	16	33,3

Tabela 1.2: Continuação da Tabela 1.1.

Fabricante	Família	Aritmética	Comprimento dos Dados	Taxas (MIPS)
Texas Instruments	TMS320C1x	Fixo	16	8,8
Texas Instruments	TMS320C2x	Fixo	16	12,5
Texas Instruments	TMS320C2xx	Fixo	16	40,0
Texas Instruments	TMS320C3x	Flutuante	32	25,0
Texas Instruments	TMS320C4x	Flutuante	32	30,0
Texas Instruments	TMS320C5x	Fixo	16	50,0
Texas Instruments	TMS320C54x	Fixo	16	50,0
Texas Instruments	TMS320C8x	Fixo	8/16	50,0
Zoran	ZR3800x	Fixo	20	33,3

A maioria dos DSPs são projetados para suportarem tarefas repetitivas e numericamente intensivas. As principais características do DSPs são as seguintes [1]:

- Operações de multiplicação e acumulação em um único ciclo de instrução;
- Arquitetura da memória com multiplos acessos;
- Modos especializados de endereçamento;
- Especializado controle do programa;
- Periféricos internos ao *chip* e interfaces de I/O.

Durante o estágio supervisionado realizado no laboratório de pesquisa LIEC (Laboratório de Instrumentação Eletrônica e Controle), localizado no Departamento de Engenharia Elétrica da Universidade Federal de Campina Grande, foram realizadas as seguintes atividades:

- Estudo do Processador Digital de Sinais TMS320LF2407A;
- Realização de atividades práticas utilizando a placa de desenvolvimento eZdsp LF2407A;
- Desenvolvimento de uma apostila sobre o DSP TMS320LF2407A.

O propósito desse trabalho é criar uma apostila que forneça os requisitos para desenvolver aplicações práticas utilizando a placa de desenvolvimento eZdsp LF2407A da *Spectrum Digital*.

eZdsp LF2407A [3] é uma placa que utiliza o processador digital de sinais TMS320LF2407A para o desenvolvimento de aplicações. O dispositivo TMS320LF2407A é um processador digital de sinais da família TMS320C2000 da *Texas Instruments* [4] [5] [6] [7].

Para desenvolver os programas, utilizou-se o *software Code Composer* V4,12 [8] [9]. O *Code Composer* é o ambiente de desenvolvimento de projetos da família de DSPs da *Texas Instruments*. Com esse *software* é possível editar, compilar, carregar o código na memória do processador, executar o código, fazer a monitoração dos registradores e váriaveis do programa, além de outras funções mais específicas.

Nos capítulos seguintes descrevem-se o sistema e os periféricos do DSP TMS320LF2407A, a placa de desenvolvimento eZdsp LF2407A, os conceitos básicos do *Code Composer* e algumas aplicações práticas com códigos comentados e escritos em linguagem C.

Capítulo 2

TMS320LF2407A

2.1 Introdução

O dispositivo TMS320LF2407A é um processador digital de sinais da família TMS320C2000 da Texas Instruments. Esse processador foi projetado para controlar motores digitais e outras aplicações de controle embarcado [6]. Possui um CPU *core* baseado no processador C2xx de 16 *bits*, ponto fixo e de baixo consumo de potência. Utiliza uma arquitetura do tipo *Harvard*. Nesse tipo de arquitetura, o barramento da memória de dados e de programa são separados. Isto permite que os dados e as instruções sejam lidas simultaneamente, proporcionando uma alta velocidade na execução dos programas.

2.2 Características do TMS320LF2407A

O TMS320LF2407A possui as seguintes características:

- Ciclo de instrução de até 25 ns;
- Até 40 MIPS (Milhões de Instruções Por Segundo);
- CPU *core* baseado TMS320C2xx;
- Memória interna ao *chip*:
  - 32 K *Words* x 16 *Bits* de *Flash*/EEPROM;
  - 2,5 K *Words* x 16 *Bits* de RAM (dados/programa): 544 *Words* de DARAM e 2 K *Words* de SARAM;

- Código de segurança para programação da memória *Flash*.

- *Boot* ROM;

- Dois módulos gerenciadores de evento (EVA e EVB). Cada um inclue:
  - Dois *timers* de 16 *bits*;
  - Oito canais PWM;
  - Seis unidades de comparação;
  - Três unidades de captura;
- *Interface* com memória externa:
- Módulo *Watchdog Timer*;
- Um conversor A/D de 10 *bits* com:
  - Dezesseis canais;
  - Tempo mínimo para conversão A/D de 500 ns.
- Módulo SCI (*Serial Communications Interface*);
- Módulo SPI (*Serial Peripheral Interface*)
- Módulo CAN (*Controller Area Network*);
- Entradas e saídas digitais (40 pinos);
- Cinco interrupções externas;
- Três modos de baixa potência;
- Emulador JTAG IEEE 1149.1.

Na figura 2.1 está ilustrado o diagrama de blocos do DSP TMS320LF2407A.

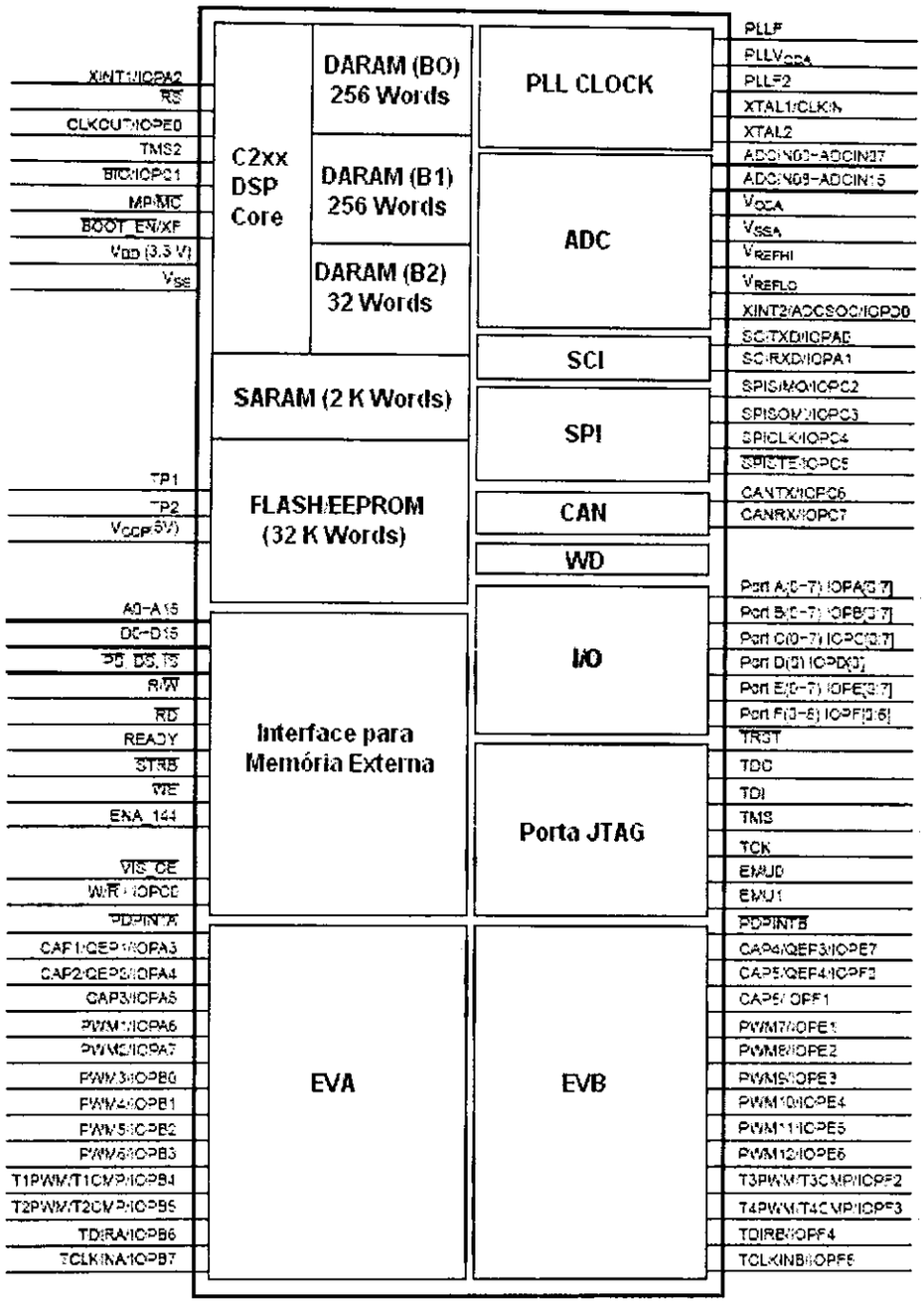


Figura 2.1: Diagrama de blocos do TMS320LF2407A.

Capítulo 3

Configuração do Sistema e Interrupções

3.1 Arquitetura do Dispositivo

A *interface* entre os periféricos e a memória interna é realizada através da *interface* PBUS (Peripheral BUS). Os periféricos internos ao *chip* são acessados (leitura e escrita) em um único ciclo e com um estado de espera nulo. Todos os periféricos excluindo o *watchdog timer*, utilizam o *clock* da CPU. Na figura 3.1 está ilustrada a arquitetura do dispositivo.

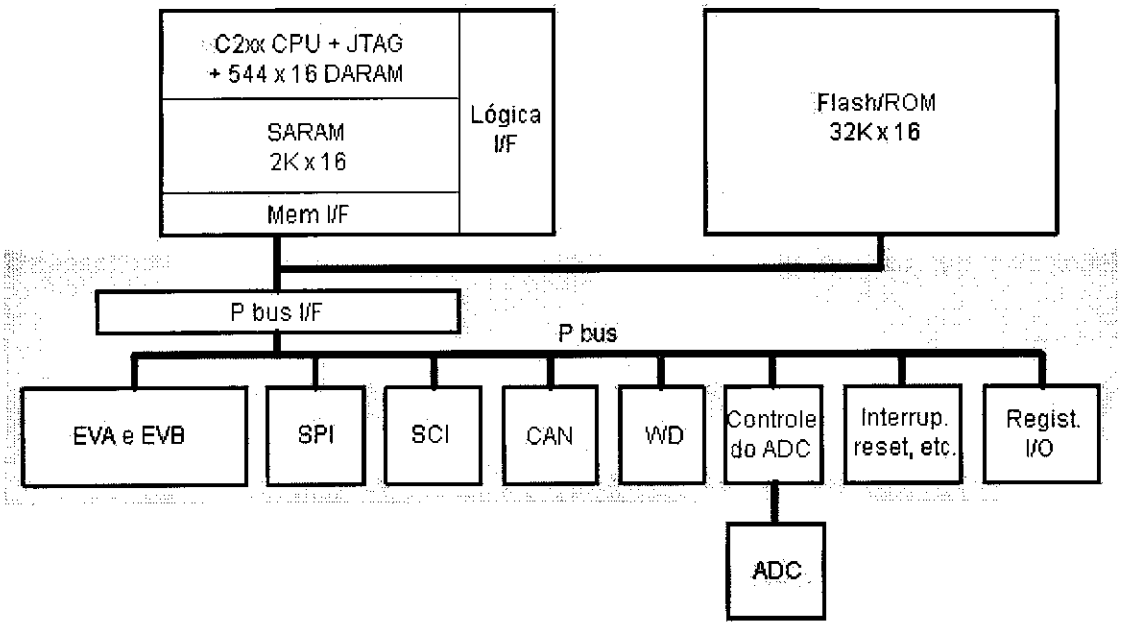


Figura 3.1: Diagrama de Blocos da Arquitetura do TMS320LF2407A.

3.2 Registradores de Configuração

Existem dois registradores que são responsáveis pela configuração do sistema, os registradores de controle de *status* **SCSR1** e **SCSR2**.

3.2.1 Registrador SCSR1

Na figura 3.2 está ilustrado um esquema com as posições dos *bits* no registrador **SCSR1**.

15	14	13	12	11	10	9	8
Reservado	CLKSRC	LPM1	LPM0	CLK PS2	CLK PS1	CLK PS0	Reservado
R-0	RW-0	RW-0	RW-0	RW-1	RW-1	RW-1	R-0
7	6	5	4	3	2	1	0
ADC CLKEN	SCI CLKEN	SPI CLKEN	CAN CLKEN	EVB CLKEN	EVA CLKEN	Reservado	ILLADR
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	R-0	RC-0

Figura 3.2: Esquema com as posições dos *bits* no registrador SCSR1.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **C** = pode ser zerado. **-n** = valor após o *reset*.

- **Bit 14 - CLKOUT** - Seleciona a fonte do *clock* para o pino CLKOUT.

  0 - Pino CLKOUT terá o *clock* da CPU como saída.

  1 - Pino CLKOUT terá o *clock* do *watchdog timer* como saída.

- **Bits 13-12 - LPM(1:0)** - Seleciona o modo de baixo consumo de energia, quando a CPU executa a instrução IDLE.

  00 - IDL1 (LPM0).

  01 - IDL2 (LPM1).

  1x - HALT (LPM2).

- **Bits 11-9 - CLKPS(2:0)** - Seleciona o fator de multiplicação do PLL para o *clock* de entrada.

  000 - 4Fin.

  001 - 2Fin.

  010 - 1,33Fin.

  011 - 1Fin.

100 - 0,8Fin.

101 - 0,66Fin.

110 - 0,57Fin.

111 - 0,5Fin.

Onde **Fin** é a frequência do *clock* de entrada. .

- **Bits 7-2 - ADC, SCI, SPI, CAN, EVB, EVA** - Habilitam o *clock* dos respectivos módulos.

  0 - Desabilita o *clock* do módulo.

  1 - Habilita o *clock* do módulo.

  OBS: Para ler ou escrever em algum registrador de um periférico deve-se habilitar o *clock* do respectivo periférico.

- **Bit 0 - ILLADR** - Esse *bit* será setado quando um endereço ilegal for acessado.

3.2.2 Registrador SCSR2

Na figura 3.3 está ilustrado um esquema com as posições dos *bits* no registrador **SCSR2**.

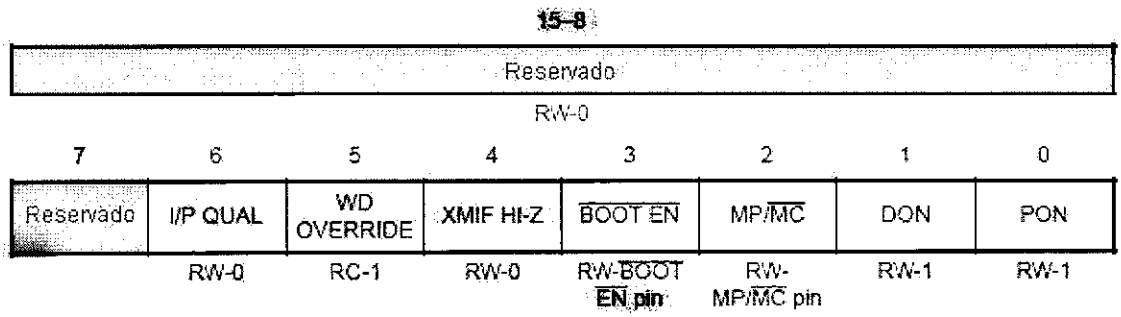


Figura 3.3: Esquema com as posições dos *bits* no registrador SCSR2.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **C** = pode ser zerado. **-n** = valor após o *reset*.

- **Bit 6 - I/P QUAL** - Um circuito qualifica o sinal de entrada dos pinos CAP1-6, XINT1/2, ADCSOC e /PDPINTA-B. O estado do sinal de entrada interno mudará somente após o sinal no pino estiver mantido em nível alto/baixo por 6 ou 12 bordas de *clock* da CPU.

  0 - O circuito bloqueia *glitches* de até 5 bordas de *clock*.

  1 - O circuito bloqueia *glitches* de até 11 bordas de *clock*.

- **Bit 5 - WD OVERRIDE** - Após o *reset*, esse *bit* dar ao usuário a autoridade de desabilitar a função *watchdog* por *software* (setando o *bit* WDDIS em 1 no registrador **WDCR**). Esse *bit* pode apenas ser zerado, e após o *reset* seu valor será 1. Para zerar esse *bit*, deve-se escrever um 1 nele.

  0 - Protege o *watchdog* de ser desabilitado por *software*.

  1 - Permite o usuário desabilitar o *watchdog* por *software*.

- **Bit 4 - XMIF HI-Z** - Este *bit* controla o estado dos sinais de *interface* de memória externa.

  0 - Os sinais XMIF são guiados no modo normal.

  1 - Os sinais XMIF são forçados ao estado HI-Z (alta impedância).

- **Bit 3 - /BOOT EN** - Este *bit* reflete o estado do pino /BOOT EN/XF após o *reset*. Este *bit* pode ser alterado por *software* para re-habilitar a memória *flash*, ou retornar ativar o *boot* ROM.

  0 - Habilita o *boot* ROM - espaço de endereço 0000 - 00FF. A memória *flash* é totalmente desabilitada neste modo.

  1 - Desabilita o *boot* ROM. O espaço de programa 0000 - 7FFF é mapeado para memória *flash*.

- **Bit 2 - MP/MC** - Este *bit* reflete o estado do pino MP/MC após o *reset*. Após o *reset*, esse *bit* pode ser alterado por *software* para permitir um mapeamente dinâmico da memória dentro e fora do *chip*.

  0 - Seta-o no modo microcontrolador. A faixa do espaço de programa de 0000 - 7FFF é mapeada internamente.

  1 - Seta-o no modo microprocessador. A faixa do espaço de programa de 0000 - 7FFF é mapeada em uma memória externa ao dispositivo.

- **Bits 1-0 - DON-PON** - Esses *bits* indicam o *status* da SARAM.

  00 - A SARAM não é mapeada. Esse espaço de endereço é alocado para memória externa.

  01 - A SARAM é mapeada internamente no espaço de programa.

  10 - A SARAM é mapeada internamente no espaço de dados.

  11 - O bloco SARAM é mapeado internamente para ambos os espaços de programa e de dados.

3.3 Vetores e Prioridade das Interrupções

Um esquema de expansão de interrupção é implementado de modo a acomodar o grande número de interrupções dos periféricos com as seis interrupções mascaráveis (INT1 - INT6) disponíveis pela CPU. Na tabela 3.4 está indicado um esquema com os vetores e a prioridade das fontes de interrupção.

Tabela 3.1: Esquema com os Vetores e a Prioridade das Fontes de Interrupção.

Prior.	**Int.**	**Vetor de Int. da CPU**	**Vetor de Int. do Perifér.**	**Masc.?**	**Perifér.**	**Descrição**
1	Reset	RSN 0000h	-	Não	Pino RS, WD	Reset do pino, Timeout WD
2	Reserv.	- 0026h	-	Não	CPU	Emulador TRAP
3	NMI	NMI 0024h	-	Não	Int. não Mascarável	Int. não Mascarável
4	PDPINTA	INT1 0002h	0020h	Sim	EVA	Pino de Int. Driver de Potência
5	PDPINTB	INT1 0002h	0019h	Sim	EVB	Pino de Int. Driver de Potência
6	ADCINT	INT1 0002h	0004h	Sim	ADC	Int. do ADC Alta Prior.
7	XINT1	INT1 0002h	0001h	Sim	Lógica de Int. Externa	Int. Externa Alta Prior.
8	XINT2	INT1 0002h	0011h	Sim	Lógica de Int. Externa	Int. Externa Alta Prior.

Tabela 3.2: Continuação da Tabela 1.1.

Prior.	Int.	Vetor de Int. da CPU	Vetor de Int. do Perifér.	Masc.?	Perifér.	Descrição
9	SPIINT	INT1 0002h	0005h	Sim	SPI	Int. SPI Alta Prior.
10	RXINT	INT1 0002h	0006h	Sim	SCI	Int. de Recep. SPI Alta Prior.
11	TXINT	INT1 0002h	0007h	Sim	SCI	Int. de Transm. SPI Alta Prior.
12	CANMBINT	INT1 0002h	0040h	Sim	CAN	Int. do Mailbox CAN Alta Prior.
13	CANERINT	INT1 0002h	0041h	Sim	CAN	Int. de Erro CAN Alta Prior.
14	CMP1INT	INT2 0004h	0021h	Sim	EVA	Int. de Comparação 1
15	CMP2INT	INT2 0004h	0022h	Sim	EVA	Int. de Comparação 2
16	CMP3INT	INT2 0004h	0023h	Sim	EVA	Int. de Comparação 3
17	T1PINT	INT2 0004h	0027h	Sim	EVA	Int. de Periodo do Timer 1
18	T1CINT	INT2 0004h	0028h	Sim	EVA	Int. de Comparação do Timer 1
19	T1UFINT	INT2 0004h	0029h	Sim	EVA	Int. de Underflow do Timer 1
20	T1OFINT	INT2 0004h	002Ah	Sim	EVA	Int. de Overflow do Timer 1
21	CMP4INT	INT2 0004h	0024h	Sim	EVB	Int. de Comparação 4
22	CMP5INT	INT2 0004h	0025h	Sim	EVB	Int. de Comparação 5
23	CMP6INT	INT2 0004h	0026h	Sim	EVB	Int. de Comparação 6

Tabela 3.3: Continuação da Tabela 1.1.

Prior.	Int.	Vetor de Int. da CPU	Vetor de Int. do Perifér.	Masc.?	Perifér.	Descrição
24	T1PINT	INT2 0004h	002Fh	Sim	EVB	Int. de Periodo do Timer 3
25	T3CINT	INT2 0004h	0030h	Sim	EVB	Int. de Comparação do Timer 3
26	T3UFINT	INT2 0004h	0031h	Sim	EVB	Int. de Underflow do Timer 3
27	T3OFINT	INT2 0004h	0032h	Sim	EVB	Int. de Overflow do Timer 3
28	T2PINT	INT3 0006h	002Bh	Sim	EVA	Int. de Periodo do Timer 2
29	T2CINT	INT3 0006h	002Ch	Sim	EVA	Int. de Comparação do Timer 2
30	T2UFINT	INT3 0006h	002Dh	Sim	EVA	Int. de Underflow do Timer 2
31	T2OFINT	INT3 0006h	002Eh	Sim	EVA	Int. de Overflow do Timer 2
32	T4PINT	INT3 0006h	0039h	Sim	EVB	Int. de Periodo do Timer 4
33	T4CINT	INT3 0006h	003Ah	Sim	EVB	Int. de Comparação do Timer 4
34	T4UFINT	INT3 0006h	003Bh	Sim	EVB	Int. de Underflow do Timer 4
35	T4OFINT	INT3 0006h	003Ch	Sim	EVB	Int. de Overflow do Timer 4
36	CAP1INT	INT4 0008h	0033h	Sim	EVA	Int. de Captura 1

Tabela 3.4: Continuação da Tabela 1.1.

Prior.	Int.	Vetor de Int. da CPU	Vetor de Int. do Perifér.	Masc.?	Perifér.	Descrição
37	CAP2INT	INT4 0008h	0034h	Sim	EVA	Int. de Captura 2
38	CAP3INT	INT4 0008h	0035h	Sim	EVA	Int. de Captura 3
39	CAP4INT	INT4 0008h	0036h	Sim	EVB	Int. de Captura 4
40	CAP5INT	INT4 0008h	0037h	Sim	EVB	Int. de Captura 5
41	CAP6INT	INT4 0008h	0038h	Sim	EVB	Int. de Captura 6
42	SPIINT	INT5 000Ah	0005h	Sim	SPI	Int. SPI Baixa Prior.
43	RXINT	INT5 000Ah	0006h	Sim	SCI	Int. de Recep. SPI - Baixa Prior.
44	TXINT	INT5 000Ah	0007h	Sim	SCI	Int. de Transm. SPI Baixa Prior.
45	CANMBINT	INT5 000Ah	0040h	Sim	CAN	Int. do Mailbox CAN Baixa Prior.
46	CANERINT	INT5 000Ah	0041h	Sim	CAN	Int. de Erro CAN Baixa Prior.
47	ADCINT	INT6 000Ch	0004h	Sim	ADC	Int. do ADC Baixa Prior.
48	XINT1	INT6 000Ch	0001h	Sim	Lógica de Int. Externa	Int. Externa Baixa Prior.
49	XINT2	INT6 000Ch	0011h	Sim	Lógica de Int. Externa	Int. Externa Baixa Prior.
-	Reserv.	000Eh	-	Sim	CPU	Análise de Int.
-	TRAP	0022h	-	-	CPU	Instr. TRAP
-	Vetor de Int. Phantom	-	0000h	-	CPU	Vetor de Int. Phantom

3.4 Controlador da Expansão de Interrupção do Periférico - PIE

A CPU do 2407A possui uma interrupção não mascarável (NMI) e seis requisições de interrupções mascaráveis com prioridade (INT1 - INT6) à nível da CPU. O dispositivo 2407A possui muitos periféricos e cada um deles é capaz de gerar uma ou mais interrupções em resposta a vários eventos à nível do periférico. Para manipular todas as requisições de interrupção dos periféricos à nível da CPU, um controlador de interrupções (PIE) foi requisitado para arbitrar as requisições de interrupção das várias fontes tais como, periféricos e outras pinos externos. Na figura 3.4 está ilustrado um diagrama de blocos do PIE.

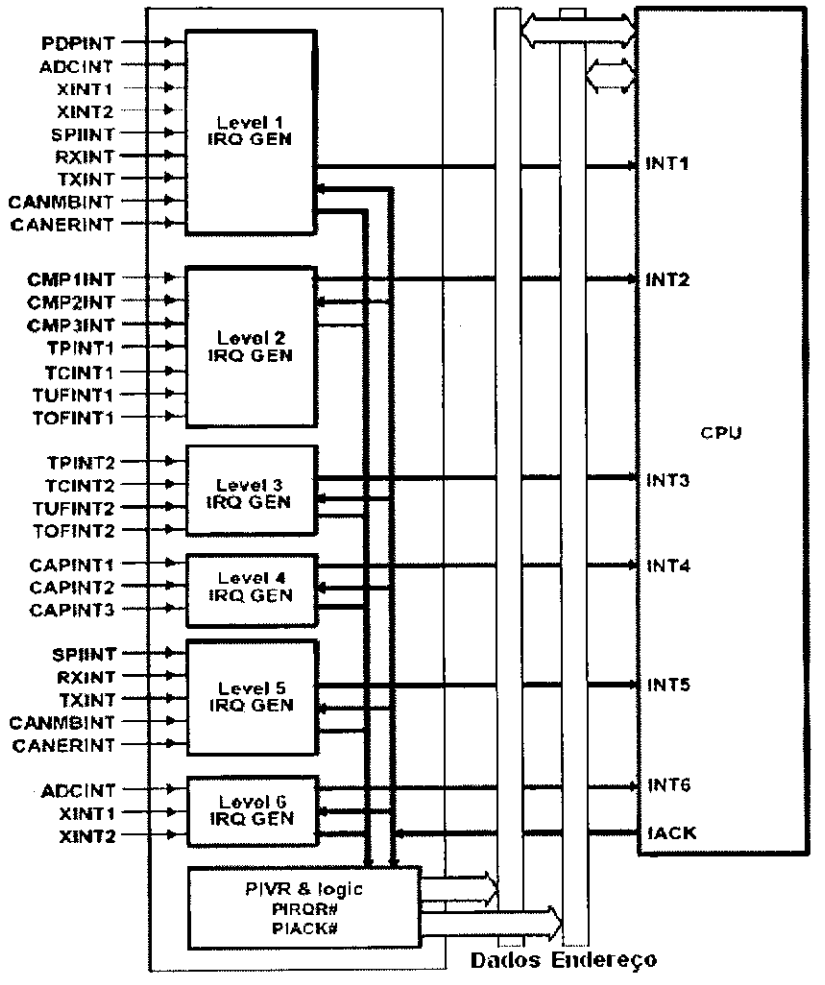


Figura 3.4: Diagrama de Blocos do PIE.

3.4.1 Estrutura de Requisição da Interrupção

As requisições de interrupção dos periféricos (PIRQ) para o controlador de interrupção, são armazenadas para gerar uma requisição de interrupção (INTn) para CPU. Existe um *bit* de sinalização da interrupção e um *bit* que habilita a interrupção localizados nos registradores de configuração dos periféricos para cada evento que possa causar um PIRQ. Se uma interrupção causada por um evento ocorrer em um periférico e o correspondente *bit* que habilita a interrupção estiver setado, a requisição de interrupção do periférico para o controlador de interrupção (PIE) será assegurada. Quando o *bit* de sinalização da interrupção for zerado, a requisição de interrupção será zerada.

Alguns periféricos podem ter a capacidade de requisitar uma interrupção de alta ou baixa prioridade. Se um periférico possui esta capacidade, o valor do *bit* que indica qual o nível de prioridade dessa requisição de interrupção será também transmitido ao controlador de interrupção. A requisição de interrupção do periférico (PIRQ) continua a ser assegurada até que seja automaticamente zerada por um reconhecimento da interrupção ou zerada por *software*.

As interrupções à nível da CPU são automaticamente zeradas após um reconhecimento da interrupção, mas as interrupções à nível do periférico deverão ser resetadas por *software*.

3.4.2 Reconhecimento da Interrupção

O esquema de expansão de interrupção requer um reconhecimento da interrupção para cada requisição de interrupção do periférico (PIRQ) ao controlador de interrupção. Quando a CPU assegura este reconhecimento de interrupção, ele simultaneamente coloca um valor no barramento de endereço de programa, no qual corresponde a interrupção da CPU que está sendo reconhecida. Cada INTn possui um vetor. Por exemplo, INT4 possui o vetor 0008h. Então, o controlador PIE decodifica o valor desse vetor para determinar qual interrupção da CPU está sendo reconhecida. Isto então gera um reconhecimento da interrupção do periférico em resposta ao PIRQ de maior prioridade assegurada no momento associado com esta interrupção da CPU.

3.5 Vetores de Interrupção

Quando a CPU recebe uma requisição de interrupção, ele não sabe qual evento do periférico causou a requisição. Para permitir que a CPU consiga distinguir entre todos os eventos, um único vetor de interrupção do periférico foi gerado para cada evento. Por exemplo, o vetor de interrupção do periférico para o caso de ocorrer um underflow no timer 1 é 0029h. Este vetor será carregado no registrador de vetor de interrupção do periférico (**PIVR**) no controlador PIE. Então, ele poderá ser lido pela CPU e usado para desviar para uma rotina de serviço de interrupção no qual corresponde ao evento que está sendo reconhecido.

3.5.1 Vetor Phantom de Interrupção

Quando o reconhecimento de uma interrupção da CPU for assegurado, mas não existir uma requisição de interrupção do periférico associado, o vetor *phantom* será usado para desviar para uma rotina que deverá manipular essa falha. O vetor phantom será requisitado quando, por exemplo, a CPU executar uma instrução de interrupção por *software* com um argumento correspondendo a uma interrupção de algum periférico. Outro exemplo, é quando um periférico faz uma requisição de interrupção, mas o *flag* INTn do registrador **IFR** foi zerado por *software* antes da CPU reconhecer a requisição. Neste caso, o controlador PIE não sabe qual vetor de interrupção do periférico deverá ser carregado no registrador **PIVR**. Nessas situações, o vetor *phantom* de interrupção será carregado no registrador **PIVR**.

3.5.2 Hierarquia do Software

Existem dois níveis de hierarquia na rotina de serviço de interrupção: a Rotina de Serviço de Interrupção Geral (GISR) e a Rotina de Serviço de Interrupção Específica (SISR). Existe um GISR para cada INTn (INT1-INT6) da CPU, no qual salva todo contexto necessário antes do registrador **PIVR** ser carregado com o vetor de interrupção do periférico. Existe também um SISR para cada requisição de interrupção de um periférico para o controlador de interrupção, e este SISR desempenha as ações requeridas na resposta a requisição de interrupção do periférico.

O GISR deverá ler o vetor de interrupção do periférico no **PIVR** antes das interrupções serem habilitadas novamente (todas interrupções são desabilitadas quando uma interrupção ocorrer). Se o **PIVR** não for lido antes das interrupções serem habilitadas novamente e outra interrupção for assegurada, um novo vetor de interrupção do periférico será carregado no registrador **PIVR**.

Interrupções não - mascaráveis tasi como, RESET e NMI não fazem parte do PIE. O PIE não controla interrupções não - mascaráveis.

3.5.3 Interrupção não - Mascarável - NMI

A interrupção NMI ocorrerá quando algum endereço ilegal for utilizado. Normalmente isso acontece quando é feita uma incorreta inicialização da página de dados.

3.6 Sequência de Operação da Interrupção

Quando um determinado evento ocorrer em um periférico, o correspondente *bit* de sinalização (**IF**) em um registrador desse periférico será setado. Se o correspondente *bit* que habilita essa interrupção (**IE**) estiver setado, o periférico gerará uma requisição de interrupção ao controlador PIE. Se a interrupção não está habilitada, o *bit* de sinalização da interrupção (**IF**) permanecerá setado até que seja zerado por *software*. Se a interrupção for habilitada algum tempo depois, e o bit de sinalização ainda estiver setado, uma requisição de interrupção do periférico (PIRQ) será assegurada.

Se nenhuma requisição de interrupção da CPU não reconhecida do mesmo nível de prioridade tiver sido previamente enviada, o PIRQ provocará o controlador PIE gerar uma requisição de interrupção da CPU (INTn).

A requisição de interrupção da CPU seta o correspondente *bit* de sinalização no registrador **IFR**. Se a interrupção da CPU estiver habilitada, isto é, se o correspondente bit de habilitação da interrupção no registrador **IMR** estiver setado, a CPU pára o que está fazendo, desabilita todas as outras interrupções mascaráveis setando o *bit* **INTM** do registrador **ST0**, salva algum contexto, e inicia a execução do GISR para esta interrupção (INTn). A CPU gerará automaticamente um reconhecimento da interrupção, no qual é enviado por um valor no barramento de endereço de programa (PAB) correspondendo ao nível de prioridade da interrupção que está sendo reconhecida. Por exemplo, se INT3 for assegurado, o vetor 0006h será carregado no PAB.

O controlador PIE decodificará o valor do PAB e gerará um reconhecimento da interrupção do periférico para zerar o *bit* do registrador **PIRQ** associado com a interrupção que está sendo reconhecida. Em seguida, o controlador PIE carrega o registrador **PIVR** com o apropriado vetor de interrupção do periférico (ou o vetor de interrupção *phantom*).

Quando o GISR terminar de salvar algum contexto necessário, ele lerá o **PIVR** e utilizará esse vetor de interrupção para desviar para o SISR da interrupção que ocorreu no periférico.

Na figura 3.5 está ilustrado um fluxograma com a sequência de operação de uma interrupção.

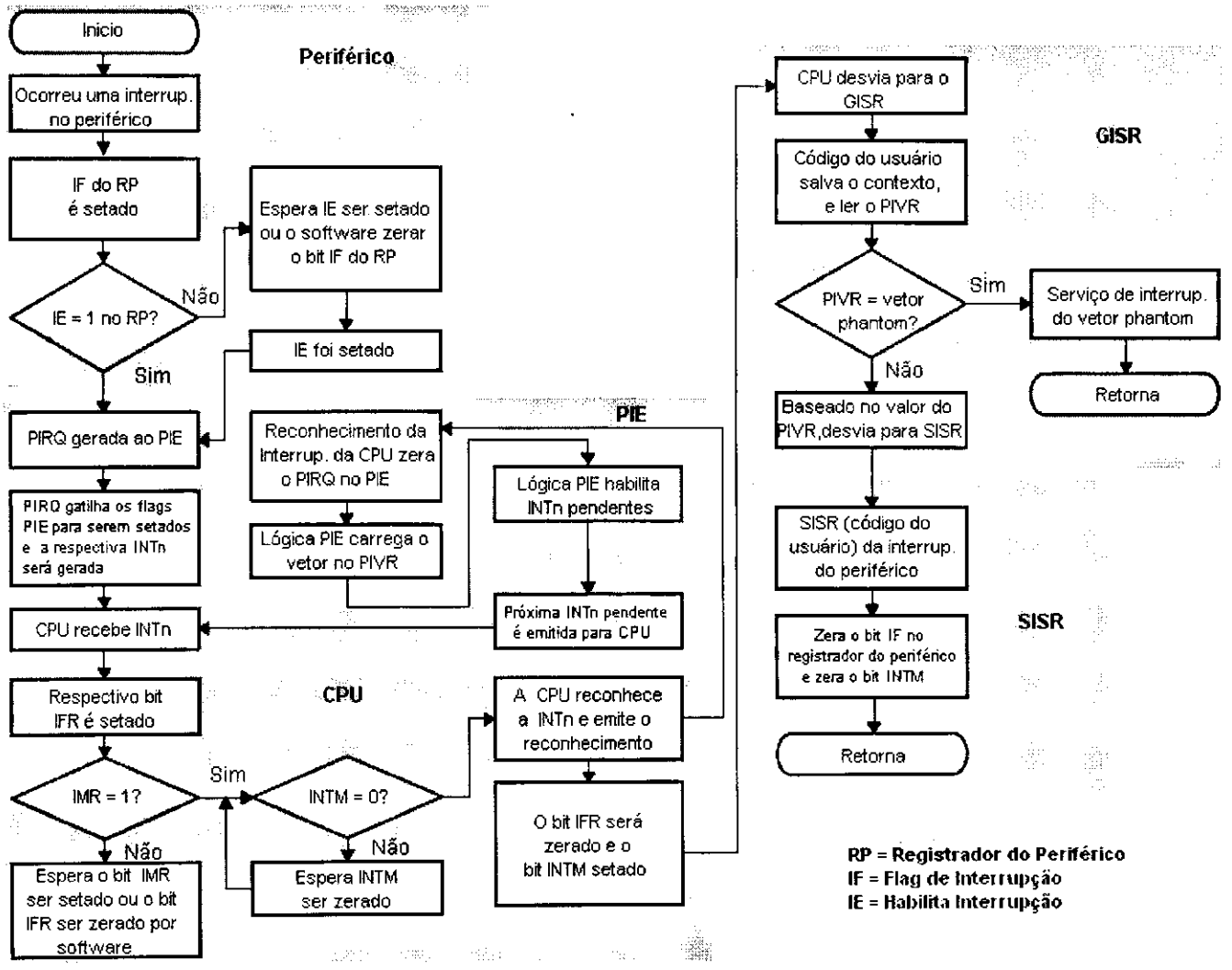


Figura 3.5: Fluxograma com a sequência de operação de uma interrupção.

3.7 Registradores de Interrupção da CPU

Existem dois registradores responsáveis pelas sinalização e habilitação das interrupções a nível da CPU. O registrador de sinalização da interrupção - **IFR** e o registrador de máscara da interrrupção - **IMR**.

3.7.1 Registrador IFR

Este registrador é usado para identificar e zerar interrupções pendentes. O IFR contém bits de sinalização para todas as interrupções mascaráveis (INT1 - INT6).

Quando uma interrupção mascarável for requisitada, o *bit* de sinalização do correspondente registrador de controle do periférico será setado. Se o correspondente *bit* de máscara for também '1', a requisição de interrupção será enviada para CPU, setando o *flag* correspondente no **IFR**. Isto indica que a interrupção está pendente ou esperando por reconhecimento.

Para zerar um *bit* de sinalização do registrador **IFR**, basta escrever '1' no correspondente *bit*. Os seguintes eventos também zeram um *bit* do **IFR**, automaticamente:

- O reconhecimento da interrupção pela CPU;
- Um *reset* do dispositivo.

Quando uma interrupção mascarável é reconhecida, apenas o *bit* de sinalização do registrador **IFR** é zerado automaticamente. O *bit* de sinalização no correspondente registrador de controle do periférico não é zerado, esse *bit* deverá ser zerado por *software*. Na figura 3.6 está ilustrado um esquema com as posições dos *bits* no registrador **IFR**.

15–6	5	4	3	2	1	0
Reservado	INT6 flag	INT5 flag	INT4 flag	INT3 flag	INT2 flag	INT1 flag
0	RW1C-0	RW1C-0	RW1C-0	RW1C-0	RW1C-0	RW1C-0

Figura 3.6: Esquema com as posições dos *bits* no registrador IFR.

OBS: **R** = acesso a leitura, **W1C** = **zera** quando se escreve um '1', **-n** = valor após o *reset*.

- **Bits 5-0 - INTn FLAG** - *Flags* de interrupção.

  0 - Nenhuma interrupção está pendente.

  1 - Interrupção pendente.

3.7.2 Registrador IMR

O **IMR** contém *bits* de máscara para todos os níveis de interrupções mascaráveis (INT1 - INT6). Na figura 3.7 está ilustrado um esquema com as posições dos *bits* no registrador **IMR**.

15–6	5	4	3	2	1	0
Reservado	INT6 mask	INT5 mask	INT4 mask	INT3 mask	INT2 mask	INT1 mask
0	**RW**	RW	RW	RW	RW	RW

Figura 3.7: Esquema com as posições dos *bits* no registrador IMR.

OBS: **R** = acesso a leitura, **W** = acesso a escrita.

- **Bits 5-0 - INTn MASK** - Máscara de interrupção.

  0 - A interrupção de nível **n** está desabilitada (mascarada).

  1 - A interrupção de nível **n** está habilitada (não mascarada).

3.8 Registradores de Interrupção do Periférico

Os seguintes registradores são utilizados no sistema de interrupção dos periféricos:

- Registrador do Vetor de Interrupção do Periférico - **PIVR**;
- Registrador de Requisição de Interrupção do Periférico 0 - **PIRQ0**;
- Registrador de Requisição de Interrupção do Periférico 1 - **PIRQ1**;
- Registrador de Requisição de Interrupção do Periférico 2 - **PIRQ2**;
- Registrador de Reconhecimento da Interrupção do Periférico 0 - **PIACKR0**;
- Registrador de Reconhecimento da Interrupção do Periférico 1 - **PIACKR1**;
- Registrador de Reconhecimento da Interrupção do Periférico 2 - **PIACKR2**;

Os registradores **PIRQ0/1/2** e **PIACKR0/1/2** são registradores de controle interno ao módulo PIE, e são usados para geração das interrupções (INT1 - INT6) da CPU. Esses registradores monitoram as operações internas do PIE, e portanto podem ser ignorados nas aplicações do usuário.

O registrador **PIVR** contém o valor do vetor de interrupção do periférico que causou a interrupção. Esse valor pode ser usado para desviar para o correspondente serviço de rotina de interrupção do periférico. Na figura 3.8 está ilustrado um esquema com as posições dos *bits* no registrador **PIVR**.

15	14	13	12	11	10	9	8
V15	V14	V13	V12	V11	V10	V9	V8
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
7	6	5	4	3	2	1	0
V7	V6	V5	V4	V3	V2	V1	V0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Figura 3.8: Esquema com as posições dos *bits* no registrador PIVR.

OBS: **R** = acesso a leitura, **-n** = valor após o *reset*.

3.9 RESET

Existem duas fontes de *reset* no TMS320LF2407A:

- Pino de *reset* externo;
- *Timeout* do *watchdog timer*.

3.10 Registradores de Controle das Interrupções Externas

Os registradores XINT1CR e XINT2CR controlam e monitoram a atividade nos pinos XINT1 e XINT2. Nas figuras 3.9 e 3.10 estão ilustradas, respectivamente, os esquemas com as posições dos *bits* nos registradores XINT1CR e XINT2CR.

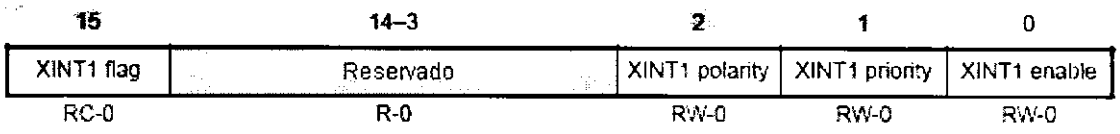


Figura 3.9: Esquema com as posições dos *bits* no registrador XINT1CR.

OBS: **R** = acesso a leitura, **W = acesso a escrita**, **C** = era quando se escreve um '1', **-n** = valor após o *reset*.

- **Bit 15 - XINT1 FLAG** - Este *bit* indica se transição selecionada foi detectada no pino XINT1.

  0 - Transição não detectada.

  1 - Transição detectada.

- **Bit 2 - XINT1 POLARITY** - Este *bit* indica a polaridade.

  0 - Interrupção será gerada na borda de descida.

  1 - Interrupção será gerada na borda de subida.

- **Bit 1 - XINT1 PRIORITY** - Este *bit* indica qual a prioridade da interrupção.

  0 - Alta prioridade.

  1 - baixa prioridade.

- **Bit 0 - XINT1 ENABLE** - Este *bit* habilita/desabilita a interrupção externa no pino XINT1.

  0 - Desabilita a interrupção.

  1 - Habilita a interrupção.

15	14–3	2	1	0
XINT2 flag	Reservado	XINT2 polarity	XINT2 priority	XINT2 enable
RC-0	R-0	RW-0	RW-0	RW-0

Figura 3.10: Esquema com as posições dos *bits* no registrador XINT2CR.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **C** = era quando se escreve um '1', **-n** = valor após o *reset*.

- **Bit 15 - XINT2 FLAG** - Este *bit* indica se transição selecionada foi detectada no pino XINT2.

  0 - Transição não detectada.

  1 - Transição detectada.

- **Bit 2 - XINT2 POLARITY** - Este *bit* indica a polaridade.

  0 - Interrupção será gerada na borda de descida.

  1 - Interrupção será gerada na borda de subida.

- **Bit 1 - XINT2 PRIORITY** - Este *bit* indica qual a prioridade da interrupção.

  0 - Alta prioridade.

  1 - baixa prioridade.

- **Bit 0 - XINT2 ENABLE** - Este *bit* habilita/desabilita a interrupção externa no pino XINT2.

  0 - Desabilita a interrupção.

  1 - Habilita a interrupção.

Capítulo 4

Memória

4.1 Introdução

O TMS320LF2407A possui um barramento de endereço de 16 *bits* que acessa os seguintes espaços, individualmente:

- Espaço de programa - 64 K*words*;
- Espaço de dados - 64 K*words*;
- Espaço de I/O - 64 K*words*;

4.2 Memória RAM Interna ao Chip

Existem interno ao *chip* uma memória com acesso duplo (DARAM) e uma memória de programa/dados com acesso único (SARAM).

4.2.1 DARAM

Existem 544 x 16 *bits* de DARAM interna ao *chip*, o qual pode ser acessada duas vezes por ciclo de máquina. As 544 *words* são divididas em três blocos: B0, B1 e B2. Esta memória é utilizada como memória de dados, mas o bloco B0 pode também ser usada como memória de programa. O bloco B0 pode ser configurado em um dos dois modos, dependendo do valor do *bit* **CNF** do registrador **ST1**. Se o *bit* **CNF** for igual a zero, B0 será mapeado como memória de dados, caso contrário, será mapeado como memória de programa.

4.2.2 SARAM

Existem 2 K*words* x 16 *bits* de SARAM. Pode-se utilizar A SARAM como memória de dados e de programa.

4.3 Memória ROM

A memória ROM é mapeada no espaço de programa.

4.4 Memória Flash

A memória *Flash* é mapeada no espaço da memória de programa. O TMS320LF2407A possui 32 K *words* de memória Flash divididos em quatro setores (4K, 12K, 12K e 4K). Essa memória é utilizada para fornecer um armazenamento permanente do programa. Ela pode ser programada e apagada eletricamente várias vezes para permitir o desenvolvimento do código.

4.5 Visão Geral dos Espaços de Memória e I/O

O dispositivo TMS320LF2407A é baseado na arquitetura *harvard*. Esse dispositivo possue vários espaços de memória acessíveis em três barramentos paralelos:

- Barramento de Endereço de Programa - **PAB**;
- Barramento de Endereço de Leitura de Dados - **DRAB**;
- Barramento de Endereço de Leitura e Escrita - **DWAB**.

Cada um desses três barramentos acessam diferentes espaços de memória de operação do dispositivo. Devido as operações no barramento serem independentes, é possível acessar simultaneamente ambos os espaços de programa e de dados.

O mapa de endereço é organizado em três espaços individuais selecionáveis:

- **Memória de programa** - Possui 64 K*words*. Contém as instruções a serem executadas e dados usados durante a execução do programa.
- **Memória de dados** - Possui 64 K*words*. Armazena os dados usados pela instruções.
- **Espaço I/O** - Possui 64 K*words*. Faz *interface* com periféricos externos e pode conter registradores internos ao *chip*.

Estes espaços fornecem um total de 192 K*words* de espaço. Todo esse espaço de memória é interno ao *chip*. Devido o TMS320LF2407A ter *interface* para memória externa, é possível mapear parte da memória para fora do *chip*. A vantagem de operação com memórias externas, é a possibilidade de acessar um grande espaço de memória. As vantagens de operação com memórias internas ao *chip*, em relação as memórias externas ao *chip* são:

- Melhor desempenho;
- Menor custo;
- Baixo consumo de energia.

4.6 Memória de Programa

Além de armazenar o código do usuário, a memória de programa também armazena operandos imediatos e uma tabela de informações. Um máximo de 64 K*words* pode ser endereçado na memória de programa do TMS320LF2407A. Este número inclue a DARAM e Flash/EEPROM internas ao *chip*.

Sempre que uma locação e memória externa ao *chip* for acessada, os sinais de controle para acesso externo (/PS, /DS, /STRB, etc) serão automaticamente gerados. Na figura 4.1 está ilustrado o mapa da memória de programa.

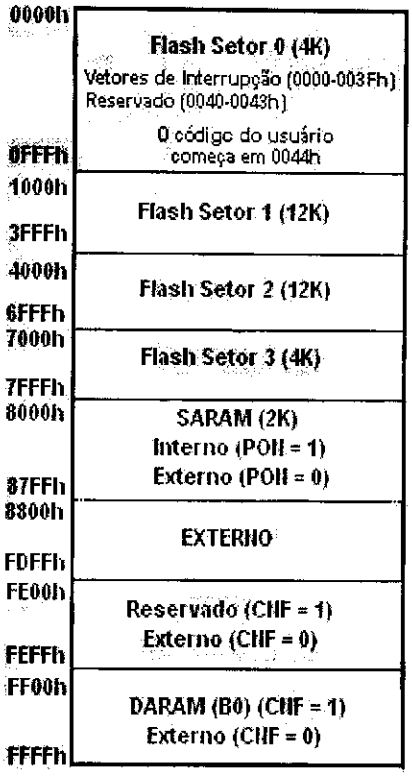


Figura 4.1: Esquema do mapa da memória de programa.

4.6.1 Configuração da Memória de Programa

Dois fatores determinam a configuração da memória de programa:

1. **Bit CNF** - *Bit* do registrador de *Status* **ST1**.
   - **CNF = 0** - As 256 *words* são mapeadas como memória externa.
   - **CNF = 1** - As 256 *words* da DARAM B0 são configuradas para memória de programa.
2. **Pino MP/MC** - O nível lógico desse pino indica se as instruções do programa são lidas da memória *Flash* ou da memória externa.
   - **MP/MC = 0** - O dispositivo está configurado no modo microcontrolador. A memória *Flash*/EEPROM interna ao *chip* está acessível. O acesso aos endereços 0000 - 7FFF da memória de programa é feita na memória interna ao *chip*.

- **MP/MC = 1** - O dispositivo está configurado no modo microprocessador. O acesso aos endereços 0000 - 7FFF da memória de programa é feita na memória externa ao *chip*.

4.7 Memória de Dados

Existe 64 K*words* de memória de dados. A metade desse espaço (0000 - 7FFF) é interna ao *chip*. A memória de dados interna inclui registradores de mapeamento da memória, DARAM e registradores de mapeamento da memória do periférico. Os 32 K*words* restantes de memória (8000 - FFFF) formam parte da memória de dados externa. Na figura 4.2 está ilustrado o mapa da memória de dados do TMS320LF2407A.

Figura 4.2: Esquema do mapa da memória de dados.

4.7.1 Configuração da Memória de Dados

A configuração da memória de dados depende do valor do *bit* **CNF** do registrador ST1:

- **CNF = 0** - B0 será usado no espaço de programa;
- **CNF = 1** - B0 será usado no espaço de dados.

4.8 Espaço I/O

Existem 64 K*words* no espaço I/O. Na figura 4.3 está ilustrado o mapa com os endereços do espaço I/O do TMS320LF2407A.

Figura 4.3: Esquema do mapa com os endereços do espaço de I/O.

4.9 Geração de Estados de Espera

Estados de espera são necessários quando se quer interfacear o TMS320LF2407A com uma lógica externa e com memórias. Adicionando estados de espera, aumenta-se o tempo de espera da CPU para a memória externa ou uma porta I/O externa responder quando a CPU ler de ou escrever para esta memória ou porta. A CPU espera um ciclo extra (um ciclo do CLKOUT) para todo estado de espera.

O TMS320LF2407A dispõe de duas formas para geração de estados de espera:

- **Sinal READY** - Com esse sinal pode-se externamente gerar algum número de estados de espera.
- **Gerador de estados de espera interno ao *chip*** - Com esse gerador pode-se gerar de 0 a 7 estados de espera.

4.9.1 Gerando Estados de Espera com o Sinal READY

Quando o sinal READY está baixo, o TMS320LF2407A espera um ciclo do CLKOUT e verifica o sinal READY novamente. Ele não continuará executando o programa até que o sinal esteja em nível alto. Portanto, se o sinal READY não for usado, ele deve ser colocado em nível alto durante acessos externos. Esse pino não tem efeito em acessos a memória interna.

4.9.2 Gerando Estados de Espera com o Gerador de Estados do TMS320LF2407A

Pode-se programar para gerar de 0 a 7 estados de espera para um determinado espaço de memória (programa, dados e I/O) externo ao *chip*. Esses estados de espera podem ser programados via o registrador **WSGR**, ilustrado na figura 4.4.

15–11	10–9	8–6	5–3	2–0
Reservado	BVIS	ISWS	DSWS	PSWS
0	W-11	W-111	W-111	W-111

Figura 4.4: Esquema com as posições dos *bits* no registrador WSGR.

OBS: **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bits 10-9 - BVIS** - Este *bit* pemite selecionar os vários modos de visibilidade do barramento durante a execução da memória interna de dados e/ou de programa.

  00 - Visibilidade do barramento desligada.

  01 - Visibilidade do barramento desligada.

  10 - Barramento de endereço envia dados para o baramento de endereço externo e o barramento de dados envia dados para o baramento de dados externo.

  11 - Barramento de endereço envia programas para o baramento de endereço externo e o barramento de dados envia programa para o baramento de dados externo.

- **Bits 8-6 - ISWS** - Determina o número de estados de espera (0 a 7) que serão aplicados para ler de e escrever para o espaço I/O externo ao *chip*.

- **Bits 5-3 - DSWS** - Determina o número de estados de espera (0 a 7) que serão aplicados para ler de e escrever para o espaço de dados externo ao *chip*.

- **Bits 2-0 - PSWS** - Determina o número de estados de espera (0 a 7) que serão aplicados para ler de e escrever para o espaço de programa externo ao *chip*.

Enquanto o sinal READY permanece em nível alto, o gerador de estados de espera insere de 0 a 7 estados de espera para um dado espaço de memória, dependendo dos valores de PSWS, DSWS e ISWS. Por exemplo, se **M** é o número de ciclos CLKOUT requisitados para uma particular operação de leitura ou escrita, e **W** é o número de estados de espera adicionais, a operação terá uma duração de **M + W** ciclos.

Capítulo 5

Clocks e Modos de Baixa Potência

5.1 Pinos

Existem três pinos no dispositivo que estão associados com os *clocks*:

- **XTAL1/CLKIN** - Este é o pino que contém o *clock* de entrada do cristal externo para o oscilador interno ao *chip*.
- **XTAL2** - Este é o pino que contém o *clock* de saída do oscilador interno ao *chip* para guiar o cristal externo.
- **CLKOUT/IOPE0** - Este é o pino de saída do *clock*. Ele é multiplexado com o pino 0 da porta E (IOPE0). Este pino pode ser usado como saída do *clock* da CPU ou do *clock* do *watchdog timer*, dependendo do valor do *bit* 14 do registrador **SCSR1**.

5.2 Phase Locked Loop - PLL

O TMS320LF2407A utiliza um circuito PLL embarcado no CPU *core*. O propósito de usar um PLL é multiplicar a frequência externa de referência (10 MHz) para uma frequência maior ou menor, que será usada internamente como *clock* da CPU.

Esse PLL dispõe de fatores de multiplicação que variam de 0,5 a 4 vezes a frequência do *clock* de entrada. Esse fator de multiplicação é configurado nos *bits* 11-9 do registrador **SCSR1**. Como a frequência do *clock* de entrada é de 10 MHz, é possível atingir uma frequência do *clock* da CPU de até 40 MHz, basta configurar o PLL para multiplicar por 4.

5.3 Watchdog timer Clock - WDCLK

Um clock de baixa frequência, **WDCLK**, é usado pelo *clock* do *watchdog timer*. **WDCLK** possui uma frequência nominal de 78125 Hz quando o *clock* da CPU (**CPUCLK ou CLKOUT**) for 40 MHz.

O WDCLK é gerado no periférico *watchdog timer*. A equação abaixo mostra a relação entre o WDCLK e o CLKOUT.

$$WDCLK = \frac{CLKOUT}{512}$$

5.4 Modos de Baixa Potência

O 2407A possui uma instrução **IDLE**, que quando executada, desliga os *clocks* de todos os circuitos da CPU. Entretanto, o *clock* de saída da CPU continua em atividade. Com esta instrução, os *clocks* da CPU são desligados para diminuir o consumo de energia. A CPU sai do estado **IDLE** se for resetado ou se receber uma requisição de interrupção.

Existem dois *bits* de controle que especificam qual modo de baixa potência será utilizado após a instrução **IDLE** ser executada. Esses *bits* de controle estão localizados nos bits 13-12 do registrador **SCSR1**.

Existem três modos de baixa potência: **IDLE1**, **IDLE2** e **HALT**. Quando a CPU entra no modo **IDLE1**, o *clock* da CPU é parado, mas o *clock* do sistema continua em atividade. O modo **IDLE2** é implementado por uma lógica externa. Nesse modo, o *clock* da CPU e do sistema são desligados. No terceiro modo de baixa potência, modo **HALT**, o **clock** de entrada para o PLL é desligado.

Capítulo 6

Entradas e Saídas Digitais - I/O

6.1 Introdução

Existem seis portas no TMS320LF2407A: as portas A, B, C, D, E e F. As portas A, B, C e E possuem oito pinos, a porta D possui apenas um pino, e a porta F possui sete pinos. Cada pino de uma porta pode ser configurado para atuar como um pino de função I/O (função secundária), ou como um pino de função compartilhada (função primária).

Todos os pinos de funções I/O e funções compartilhadas são controlados através de nove registradores de 16 *bits*. Esses registradores são divididos em dois tipos:

- **Registradores de Controle MUX I/O (MCRx)** - Esse registrador é utilizado para escolher entre a função primária ou secundária.
- **Registradores de Controle de Dados e Direção (PxDATDIR)** - Esse registrador é utilizado para controlar os dados e a direção dos dados, quando os pinos estão configurados como uma função I/O (função secundária).

6.2 Registradores de Controle MUX I/O

Existem três registradores de controle MUX I/O: O **MCRA**, **MCRB** e **MCRC**.

6.2.1 Registrador MCRA

Na figura 6.1 está ilustrado um esquema com as posições dos *bits* no registrador **MCRA**.

15	14	13	12	11	10	9	8
MCRA.15	MCRA.14	MCRA.13	MCRA.12	MCRA.11	MCRA.10	MCRA.9	MCRA.8
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7	6	5	4	3	2	1	0
MCRA.7	MCRA.6	MCRA.5	MCRA.4	MCRA.3	MCRA.2	MCRA.1	MCRA.0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Figura 6.1: Esquema com as posições dos *bits* no registrador MCRA.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bits 15-0 - MCRA.n** - Indica qual função será selecionada.

  0 - Função secundária.

  1 - Função primária.

Na tabela 6.1 estão indicados as funções que podem ser selecionados através do registrador **MCRA**.

Tabela 6.1: Funções selecionadas através do registrador MCRA

Bits	MCAn = 1	MCAn = 0
MCRA.0	SCITXD	IOPA0
MCRA.1	SCIRXD	IOPA1
MCRA.2	XINT1	IOPA2
MCRA.3	CAP1/QEP1	IOPA3
MCRA.4	CAP2/QEP2	IOPA4
MCRA.5	CAP3	IOPA5
MCRA.6	PWM1	IOPA6
MCRA.7	PWM2	IOPA7
MCRA.8	PWM3	IOPB0
MCRA.9	PWM4	IOPB1
MCRA.10	PWM5	IOPB2
MCRA.11	PWM6	IOPB3
MCRA.12	T1PWM/T1CMP	IOPB4
MCRA.13	T2PWM/T2CMP	IOPB5
MCRA.14	TDIRA	IOPB6
MCRA.15	TCLKINA	IOPB7

6.2.2 Registrador MCRB

Na figura 6.2 está ilustrado um esquema com as posições dos *bits* no registrador **MCRB**.

15	14	13	12	11	10	9	8
MCRB.15	MCRB.14	MCRB.13	MCRB.12	MCRB.11	MCRB.10	MCRB.9	MCRB.8
RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-0
7	6	5	4	3	2	1	0
MCRB.7	MCRB.6	MCRB.5	MCRB.4	MCRB.3	MCRB.2	MCRB.1	MCRB.0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-1	RW-1

Figura 6.2: Esquema com as posições dos *bits* no registrador MCRB.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bits 15-0 - MCRB.n** - Indica qual função será selecionada.

  0 - Função secundária.

  1 - Função primária.

Na tabela 6.2 estão indicados as funções que podem ser selecionados através do registrador **MCRB**.

Tabela 6.2: Funções selecionadas através do registrador MCRB

Bits	MCAn = 1	MCAn = 0
MCRB.0	W/R	IOPC0
MCRB.1	BIO	IOPC1
MCRB.2	SPISIMO	IOPC2
MCRB.3	SPISOMI	IOPC3
MCRB.4	SPICLK	IOPC4
MCRB.5	SPISTE	IOPC5
MCRB.6	CANTX	IOPC6
MCRB.7	CANRX	IOPC7
MCRB.8	XINT2/ADCSOC	IOPD0
MCRB.9	EMU0	Reservado
MCRB.10	EMU1	Reservado
MCRB.11	TCK	Reservado
MCRB.12	TDI	Reservado
MCRB.13	TDO	Reservado
MCRB.14	TMS	Reservado
MCRB.15	TMS2	Reservado

6.2.3 Registrador MCRC

Na figura 6.3 está ilustrado um esquema com as posições dos *bits* no registrador **MCRC**.

15	14	13	12	11	10	9	8
Reservado	Reservado	MCRC.13	MCRC.12	MCRC.11	MCRC.10	MCRC.9	MCRC.8
		RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
MCRC.7	MCRC.6	MCRC.5	MCRC.4	MCRC.3	MCRC.2	MCRC.1	MCRC.0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-1

Figura 6.3: Esquema com as posições dos *bits* no registrador MCRC.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bits 15-0 - MCRC.n** - Indica qual função será selecionada.

  0 - Função secundária.

  1 - Função primária.

Na tabela 6.3 estão indicados as funções que podem ser selecionados através do registrador **MCRC**.

Tabela 6.3: Funções selecionadas através do registrador MCRC

Bits	MCCn = 1	MCCn = 0
MCRC.0	CLKOUT	IOPE0
MCRC.1	PWM7	IOPE1
MCRC.2	PWM8	IOPE2
MCRC.3	PWM9	IOPE3
MCRC.4	PWM10	IOPE4
MCRC.5	PWM11	IOPE5
MCRC.6	PWM12	IOPE6
MCRC.7	CAP4/QEP3	IOPE7
MCRC.8	CAP5/QEP4	IOPF0
MCRC.9	CAP6	IOPF1
MCRC.10	T3PWM/T3CMP	IOPF2
MCRC.11	T4PWM/T4CMP	IOPF3
MCRC.12	TDIRB	IOPF4
MCRC.13	TCLKINB	IOPF5
MCRC.14	Reservado	IOPF6
MCRC.15	Reservado	Reservado

6.3 Registradores de Controle de Dados e Direção

Existem seis registradores de controle de dados e direção: o PADATDIR, PBDATDIR, PCDATDIR, PDDATDIR, PEDATDIR e o PFDATDIR.

6.3.1 Registrador PADATDIR

Na figura 6.4 está ilustrado um esquema com as posições dos *bits* no registrador **PADATDIR**.

15	14	13	12	11	10	9	8
A7DIR	A6DIR	A5DIR	A4DIR	A3DIR	A2DIR	A1DIR	A0DIR
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7	6	5	4	3	2	1	0
IOPA7	IOPA6	IOPA5	IOPA4	IOPA3	IOPA2	IOPA1	IOPA0
RW-†	RW-†	RW-†	RW-†	RW-†	RW-†	RW-†	RW-†

Figura 6.4: Esquema com as posições dos *bits* no registrador PADATDIR.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-0** = após o *reset* o valor desse *bit* é zero.

-+ = após o *reset* o valor desse *bit* depende do estado dos respectivos pinos.

- **Bits 15-8 - AnDIR** - Configura o respectivo pino como uma entrada ou uma saída.

  0 - Configurado como entrada.

  1 - Configurado como saída.

- **Bits 7-0 - IOPAn** - Indica qual o nível lógico no pino.

  Caso **AnDIR** = 0, temos:

  0 - O valor do pino de entrada é lido como zero.

  1 - O valor do pino de entrada é lido como um.

  Caso **AnDIR** = 1, temos:

  0 - O pino de saída terá nível baixo.

  1 - O pino de saída terá nível alto.

6.3.2 Registrador PBDATDIR

Na figura 6.5 está ilustrado um esquema com as posições dos *bits* no registrador **PBDATDIR**.

15	14	13	12	11	10	9	8
B7DIR	B6DIR	B5DIR	B4DIR	B3DIR	B2DIR	B1DIR	B0DIR
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7	6	5	4	3	2	1	0
IOPB7	IOPB6	IOPB5	IOPB4	IOPB3	IOPB2	IOPB1	IOPB0
RW-†	RW-†	RW-†	RW-†	RW-†	RW-†	RW-†	RW-†

Figura 6.5: Esquema com as posições dos *bits* no registrador PBDATDIR.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-0** = após o *reset* o valor desse *bit* é zero.

-+ = após o *reset* o valor desse *bit* depende do estado dos respectivos pinos.

- **Bits 15-8 - BnDIR -** Configura o respectivo pino como uma entrada ou uma saída.

  0 - Configurado como entrada.

  1 - Configurado como saída.

- **Bits 7-0 - IOPBn** - Indica qual o nível lógico no pino.

  Caso **BnDIR** = 0, temos:

  0 - O valor do pino de entrada é lido como zero.

  1 - O valor do pino de entrada é lido como um.

  Caso **BnDIR** = 1, temos:

  0 - O pino de saída terá nível baixo.

  1 - O pino de saída terá nível alto.

6.3.3 Registrador PCDATDIR

Na figura 6.6 está ilustrado um esquema com as posições dos *bits* no registrador **PCDATDIR**.

15	14	13	12	11	10	9	8
C7DIR	C6DIR	C5DIR	C4DIR	C3DIR	C2DIR	C1DIR	C0DIR
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7	6	5	4	3	2	1	0
IOPC7	IOPC6	IOPC5	IOPC4	IOPC3	IOPC2	IOPC1	IOPC0
RW-†	RW-†	RW-†	RW-†	RW-†	RW-†	RW-†	RW-x

Figura 6.6: Esquema com as posições dos *bits* no registrador PCDATDIR.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-0** = após o *reset* o valor desse *bit* é zero.

-+ = após o *reset* o valor desse *bit* depende do estado dos respectivos pinos.

- **Bits 15-8 - CnDIR** - Configura o respectivo pino como uma entrada ou uma saída.

  0 - Configurado como entrada.

  1 - Configurado como saída.

- **Bits 7-0 - IOPCn** - Indica qual o nível lógico no pino.

  Caso **CnDIR** = 0, temos:

  0 - O valor do pino de entrada é lido como zero.

  1 - O valor do pino de entrada é lido como um.

  Caso **CnDIR** = 1, temos:

  0 - O pino de saída terá nível baixo.

  1 - O pino de saída terá nível alto.

6.3.4 Registrador PDDATDIR

Na figura 6.7 está ilustrado um esquema com as posições dos *bits* no registrador **PDDATDIR**.

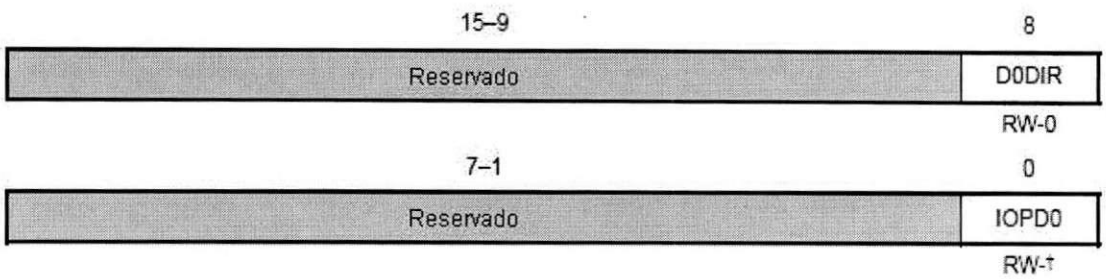


Figura 6.7: Esquema com as posições dos *bits* no registrador PDDATDIR.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-0** = após o *reset* o valor desse *bit* é zero.

-+ = após o *reset* o valor desse *bit* depende do estado dos respectivos pinos.

- **Bits 15-8 - DnDIR** - Configura o respectivo pino como uma entrada ou uma saída.

  0 - Configurado como entrada.

  1 - Configurado como saída.

- **Bits 7-0 - IOPDn** - Indica qual o nível lógico no pino.

  Caso **DnDIR** = 0, temos:

  0 - O valor do pino de entrada é lido como zero.

  1 - O valor do pino de entrada é lido como um.

  Caso **DnDIR** = 1, temos:

  0 - O pino de saída terá nível baixo.

  1 - O pino de saída terá nível alto.

6.3.5 Registrador PEDATDIR

Na figura 6.8 está ilustrado um esquema com as posições dos *bits* no registrador **PEDATDIR**.

15	14	13	12	11	10	9	8
E7DIR	E6DIR	E5DIR	E4DIR	E3DIR	E2DIR	E1DIR	E0DIR
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7	6	5	4	3	2	1	0
IOPE7	IOPE6	IOPE5	IOPE4	IOPE3	IOPE2	IOPE1	IOPE0
RW-†	RW-†	RW-†	RW-†	RW-†	RW-†	RW-†	RW-x

Figura 6.8: Esquema com as posições dos *bits* no registrador PEDATDIR.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-0** = após o *reset* o valor desse *bit* é zero.

-+ = após o *reset* o valor desse *bit* depende do estado dos respectivos pinos.

- **Bits 15-8 - EnDIR** - Configura o respectivo pino como uma entrada ou uma saída.

  0 - Configurado como entrada.

  1 - Configurado como saída.

- **Bits 7-0 - IOPEn** - Indica qual o nível lógico no pino.

  Caso **EnDIR** = 0, temos:

  0 - O valor do pino de entrada é lido como zero.

  1 - O valor do pino de entrada é lido como um.

  Caso **EnDIR** = 1, temos:

  0 - O pino de saída terá nível baixo.

  1 - O pino de saída terá nível alto.

6.3.6 Registrador PFDATDIR

Na figura 6.9 está ilustrado um esquema com as posições dos *bits* no registrador **PFDATDIR**.

15	14	13	12	11	10	9	8
Reservado	F6DIR	F5DIR	F4DIR	F3DIR	F2DIR	F1DIR	F0DIR
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7	6	5	4	3	2	1	0
Reservado	IOPF6	IOPF5	IOPF4	IOPF3	IOPF2	IOPF1	IOPF0
	RW-†	RW-†	RW-†	RW-†	RW-†	RW-†	RW-†

Figura 6.9: Esquema com as posições dos *bits* no registrador PFDATDIR.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-0** = após o *reset* o valor desse *bit* é zero.

-+ = após o *reset* o valor desse *bit* depende do estado dos respectivos pinos.

- **Bits 15-8 - FnDIR** - Configura o respectivo pino como uma entrada ou uma saída.

  0 - Configurado como entrada.

  1 - Configurado como saída.

- **Bits 7-0 - IOPFn** - Indica qual o nível lógico no pino.

  Caso **FnDIR** = 0, temos:

  0 - O valor do pino de entrada é lido como zero.

  1 - O valor do pino de entrada é lido como um.

  Caso **FnDIR** = 1, temos:

  0 - O pino de saída terá nível baixo.

  1 - O pino de saída terá nível alto.

Capítulo 7

Gerenciador de Eventos

7.1 Introdução

O módulo gerenciador de eventos (EV) fornece várias funções que são particularmente utilizadas em aplicações de controle de motores.

Existem dois gerenciadores de eventos: o EVA e o EVB. Eles são idênticos. Cada EV do TMS320LF2407A possui os seguintes blocos funcionais:

- Dois *timers*;
- Três unidades de comparação;
- Circuitos PWM, que incluem os circuitos SVPWM (Space Vector PWM - Modulação Vetorial), unidades de geração de banda morta e lógica de saída;
- Três unidades de captura;
- Circuitos QEP (*Quadrature Encoder Pulse*);
- Lógica de interrupção.

Nas figuras 7.1 e 7.2 estão ilustrados, respectivamente, os diagramas de blocos dos módulos EVA e EVB. Veja que os dois módulos são idênticos.

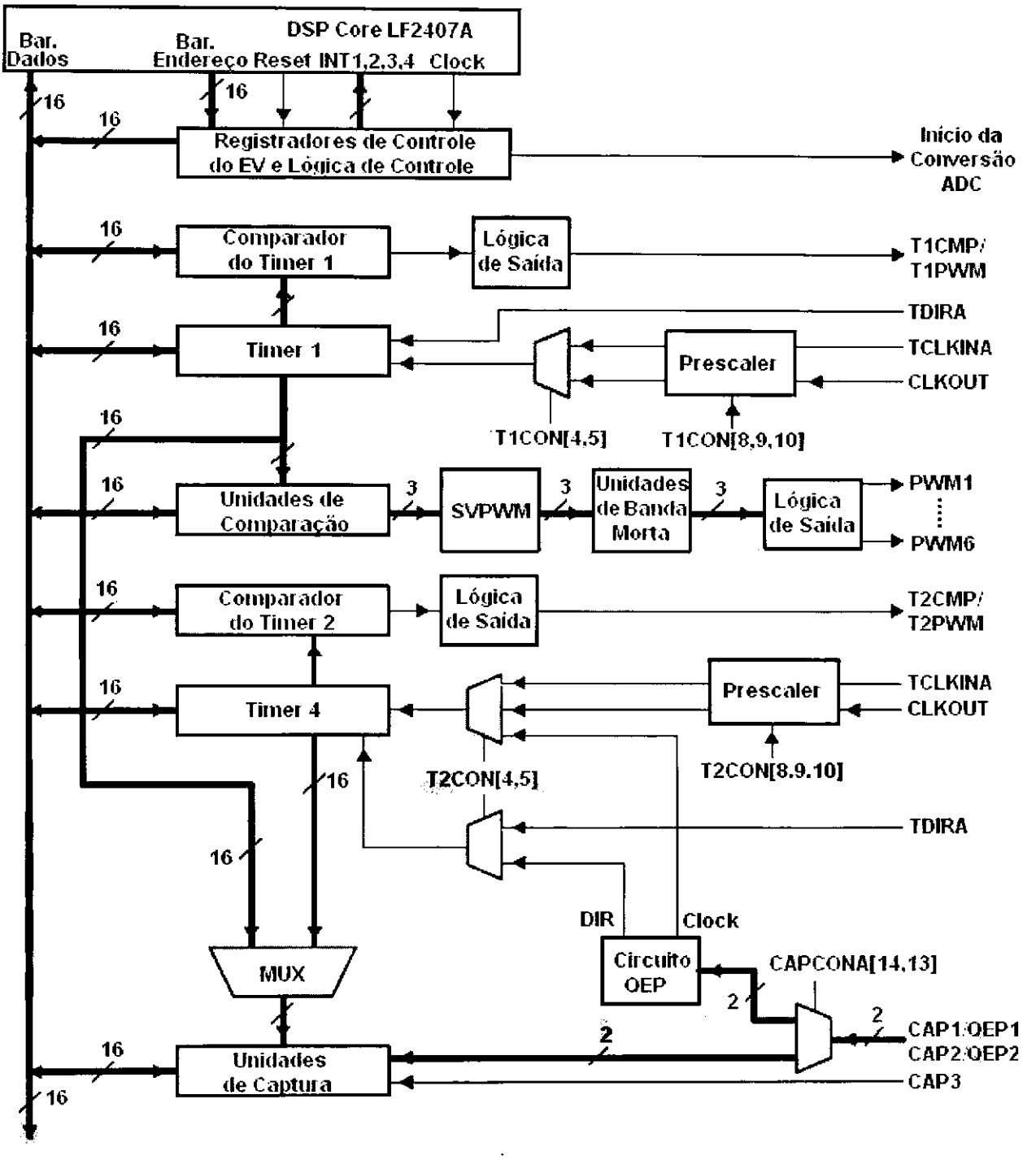


Figura 7.1: Diagrama de blocos do módulo EVA.

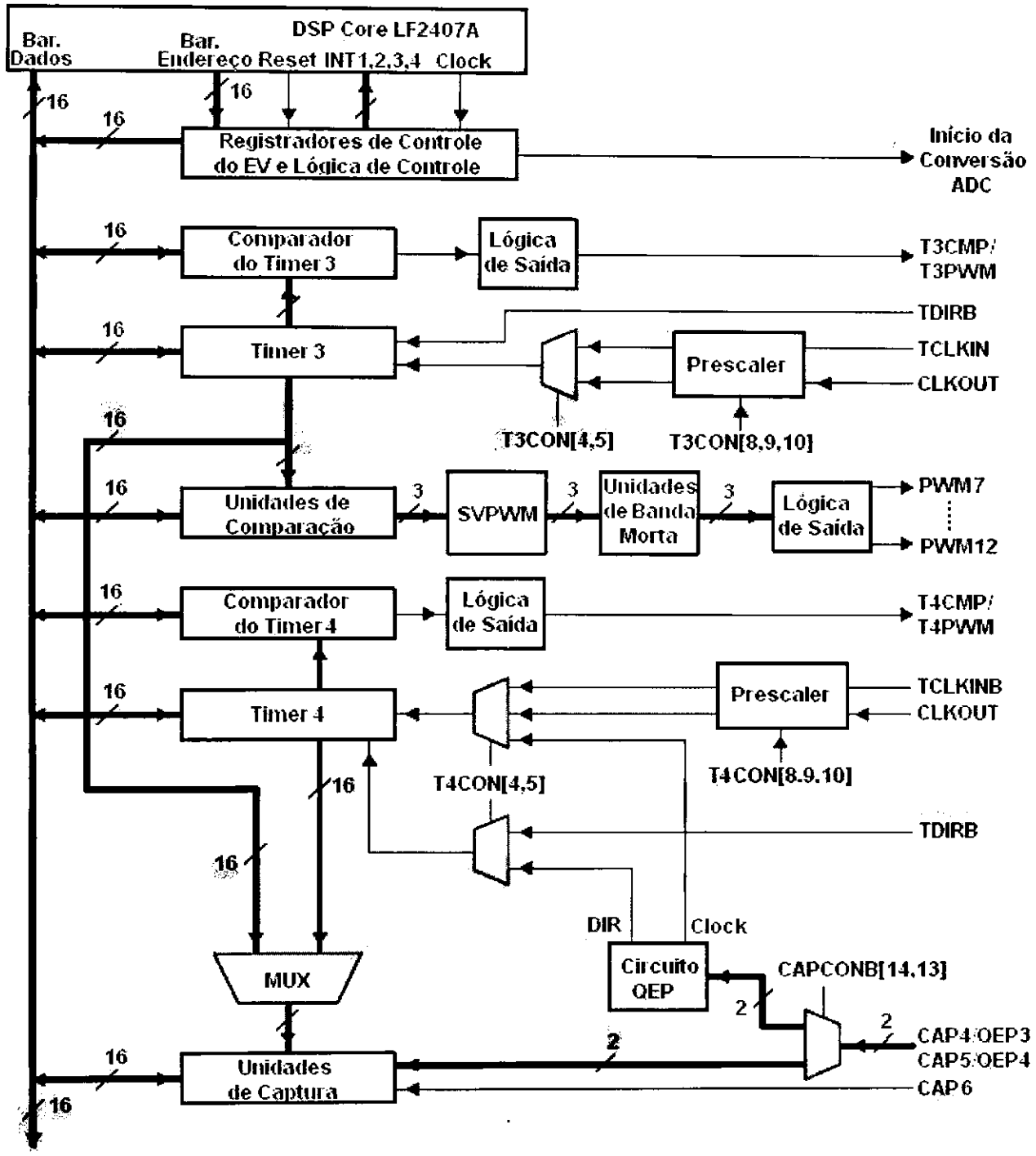


Figura 7.2: Diagrama de blocos do módulo EVB.

7.1.1 Pinos do EV

Cada módulo possui oito pinos disponíveis para as saídas de comparação/PWM, três pinos de entrada para captura e dois pinos de entrada do circuito QEP. Os *timers* podem ser programados para operar baseados em um *clock* externo ou interno ao dispositivo. O pino TCLKINA/B é a entrada do *clock* externo e o pino TDIRA/B é utilizado para especificar a direção de contagem, quando um timer está no modo de contagem crescente/decrescente direcional. Na tabela 7.1 estão indicados os pinos utilizados pelos módulos EVA e EVB e suas respectivas descrições.

Tabela 7.1: Descrição dos pinos utilizados pelos módulos EVA e EVB.

Pino - EVA	**Descrição**	**Pino - EVB**	**Descrição**
CAP1/QEP1	Unidade de Captura 1/ Entrada 1 do circuito QEP	CAP4/QEP3	Unidade de Captura 4/ Entrada 3 do circuito QEP
CAP2/QEP2	Unidade de Captura 2/ Entrada 2 do circuito QEP	CAP5/QEP4	Unidade de Captura 5/ Entrada 4 do circuito QEP
CAP3	Unidade de Captura 1	CAP6	Unidade de Captura 6
PWM1	Saída 1 da Unidade de Comparação 1	PWM7	Saída 1 da Unidade de Comparação 4
PWM2	Saída 2 da Unidade de Comparação 1	PWM8	Saída 2 da Unidade de Comparação 4
PWM3	Saída 1 da Unidade de Comparação 2	PWM9	Saída 1 da Unidade de Comparação 5
PWM4	Saída 2 da Unidade de Comparação 2	PWM10	Saída 2 da Unidade de Comparação 5
PWM5	Saída 1 da Unidade de Comparação 3	PWM11	Saída 1 da Unidade de Comparação 6
PWM6	Saída 2 da Unidade de Comparação 3	PWM12	Saída 2 da Unidade de Comparação 6

Continuação da tabela 1.2.

Pino - EVA	Descrição	Pino - EVB	Descrição
T1CMP/T1PWM	Saída de Comparação/PWM do Timer 1	T3CMP/T3PWM	Saída de Comparação/PWM do Timer 3
T2CMP/T2PWM	Saída de Comparação/PWM do Timer 2	T4CMP/T4PWM	Saída de Comparação/PWM do Timer 4
TCLKINA	Entrada do Clock Externo para os Timers do EVA	TCLKINB	Entrada do Clock Externo para os Timers do EVB
TDIRA	Entrada para Direcionar o Timer Externo no EVA	TDIRB	Entrada para Direcionar o Timer Externo no EVA

7.1.2 Interrupção de Proteção do Drive de Potência - PDPINTn, n = A ou B

O pino PDPTNTn pode ser usado para informar anômalias em um motor tais como: sobretensões, sobrecorrentes e altas temperaturas. Se a interrupção PDPINTn for habilitada, todos os pinos de saída PWM serão colocados em estado de alta impedância imediatamente após o pino PDPINTn ser colocado em nível baixo. Uma interrupção também será gerada.

O *flag* de interrupção associado com o PDPINTn será também setado quando tais eventos ocorrerem. Entretanto, deve-se esperar até que a transição no pino PDPINTn seja qualificada e sincronizada com o *clock* interno. A qualificação e a sincronização causam um atraso de dois dois ciclos de *clock*. O PDPINTn deverá ser mantido em nível baixo por seis ou doze ciclos de *clock* antes de ser reconhecida pela CPU. A sinalização da interrupção não depende apenas da interrupção está habilitada. Depende também da ocorrência de uma transição qualificada no pino PDPINTn. Se a interrupção PDPINTn estiver desabilitada, as saídas do PWM não serão colocados em estado de alta impedância. O valor do pino PDPINTn é refletido no *bit* 8 do registrador **COMCONn**.

7.1.3 Interrupções dos EVs

As interrupções dos EVs são organizadas em três grupos. Cada grupo é sinalizado por uma interrupção da CPU (INT 2, 3 ou 4). Desde que cada grupo possui várias fontes de interrupção, as requisições de interrupção a CPU são processadas pelo módulo PIE. As requisições de interrupção passam pelos seguintes estágios de resposta:

- **Fonte de Interrupção** - Se a condição de interrupção do periférico ocorrer, os respectivos *bits* de sinalização nos registradores **EVnIFRA**, **EVnIFRB** ou **EVnIFRC**, onde n = A ou B, serão setados. Uma vez setados, estes *flags* permanecem nesse estado até que sejam explicitamente zerados por software. Caso eles não sejam zerados, futuras interrupções não serão sinalizadas.

- **Habilitar Interrupção** - As interrupções do EV podem ser habilitadas ou desabilitadas individualmente pelos registradores de máscara de interrupção **EVnIMRA**, **EVnIMRB** ou **EVnIMRC**, onde n = A ou B. Se o respectivo bit for '1' a interrupção será habilitada/desmascarada. Se for '0' a interrupção será desabilitada/mascarada.

- **Requisitar PIE** - Se ambos os *bits* de sinalização e da máscara de interrupção estiverem setados, o periférico emite uma requisição de interrupção para o módulo PIE. O módulo PIE pode receber mais que uma interrupção do periférico. Alógica PIE grava todas as requisições de interrupção e gera a respectiva interrupção da CPU (INT 2, 3 ou 4) baseada na prioridade pré - assinalada das interrupções recebidas.

- **Resposta da CPU** - Na recepção de uma INT 2, 3 ou 4, o respectivo *bit* do registrador **IFR** será setado. Se o correspondente *bit* **IMR** estiver setado e o *bit* INTM do registrador **ST0** estiver zerado, a CPU reconhece a interrupção a interrupção e fornece um reconhecimento ao PIE. Após isto, a CPU finaliza executando a atual instrução de desvio para o vetor de interrupção correspondente a INT 2, 3 ou 4. Neste momento, o respectivo *bit* **IFR** será zerado e o *bit* INTM será setad, desabilitando o reconhecimento de futuras interrupções. O vetor de interrupção contém uma instrução de desvio para a rotina de serviço de interrupção. A partir deste momento, a resposta a interrupção será controlada por *software*.

- **Resposta do PIE** - A lógica PIE utiliza o sinal de reconhecimento da CPU para zerar o *bit* PIRQ que foi emitido pela interrupção da CPU. Ao longo disto, o PIE carrega o registrador **PIVR** com o vetor de interrupção, único ao periférico que foi reconhecido. Após isto, o PIE trabalha em paralelo com a atual interrupção por software para gerar umm interrupção a CPU e outras interrupções pendentes, se houver.

- **Interrupção por *Software*** - Existem dois níveis de resposta:
  - **Nível 1 (GISR)** - O *software* deverá salvar algum contexto e ler o registrador **PIVR** do módulo PIE para decidir qual grupo de interrupção causou a interrupção. Desde que o valor do **PIVR** é único, ele pode ser usado para desviar para uma rotina de serviço de interrupção específica para esta condição de interrupção.
  - **Nível 2 (SISR)** - Este nível é opcional e poderá residir como uma parte do nível 1. Entretanto, neste estágio a interrupção por *software* terá a responsabilidade para evitar respostas a interrupções impróprias. Após executar o código específico da interrupção, a rotina deverá limpar o *flag* de interrupção no **EVnIFRA**, **EVnIFRB** ou **EVnIFRC**, que causou o serviço de interrupção. O código será retornado após habilitar (zerar) o **bit** INTM de interrupção global a CPU.

7.2 Timers

Existem dois *timers* em cada módulo. Os *timers 1 e 2* no EVA, e os *timers 3 e 4* no EVB. Esses *timers* podem ser usados com bases de tempo independentes em aplicações tais como:

- Geração de um período de amostragem em um sistema de controle;
- Fornecer uma base de tempo para a operação do circuito QEP (apenas *timers 2 e 4*) e das unidades de captura;
- Fornecer uma base de tempo para a operação das unidades de comparação e circuitos PWM associados para gerar as saídas PWM;

Cada *timer* possui:

- Um registrador contador crescente/decrescente e crescente de 16 *bits*, **TxCNT** (x=1,2,3,4);
- Um registrador de comparação, **TxCMPR** (x=1,2,3,4);
- Um registrador de período, **TxPR** (x=1,2,3,4);
- Um registrador de controle, **TxCON** (x=1,2,3,4);
- Um pino de saída de comparação, **TxCMP** (x=1,2,3,4);
- Prescaler programável aplicável em ambas as entradas de clock interno e externo;

- Lógica de interrupção e controle;
- Lógica de condicionamento de saída.

Existe um outro registrador de controle, **GPTCONA** e **GPTCONB**, que especificam a ação a ser tomada pelos *timers* nos diferentes eventos que possam ocorrer no *timer* e a direção de contagem do *timer*. Na figura 7.3 está ilustado o diagrama de blocos de um *timer*.
Onde: x = 2 ou 4. Quando x = 2, y = 1 e n = 2. Quando x = 4, y = 3 e n = 4.

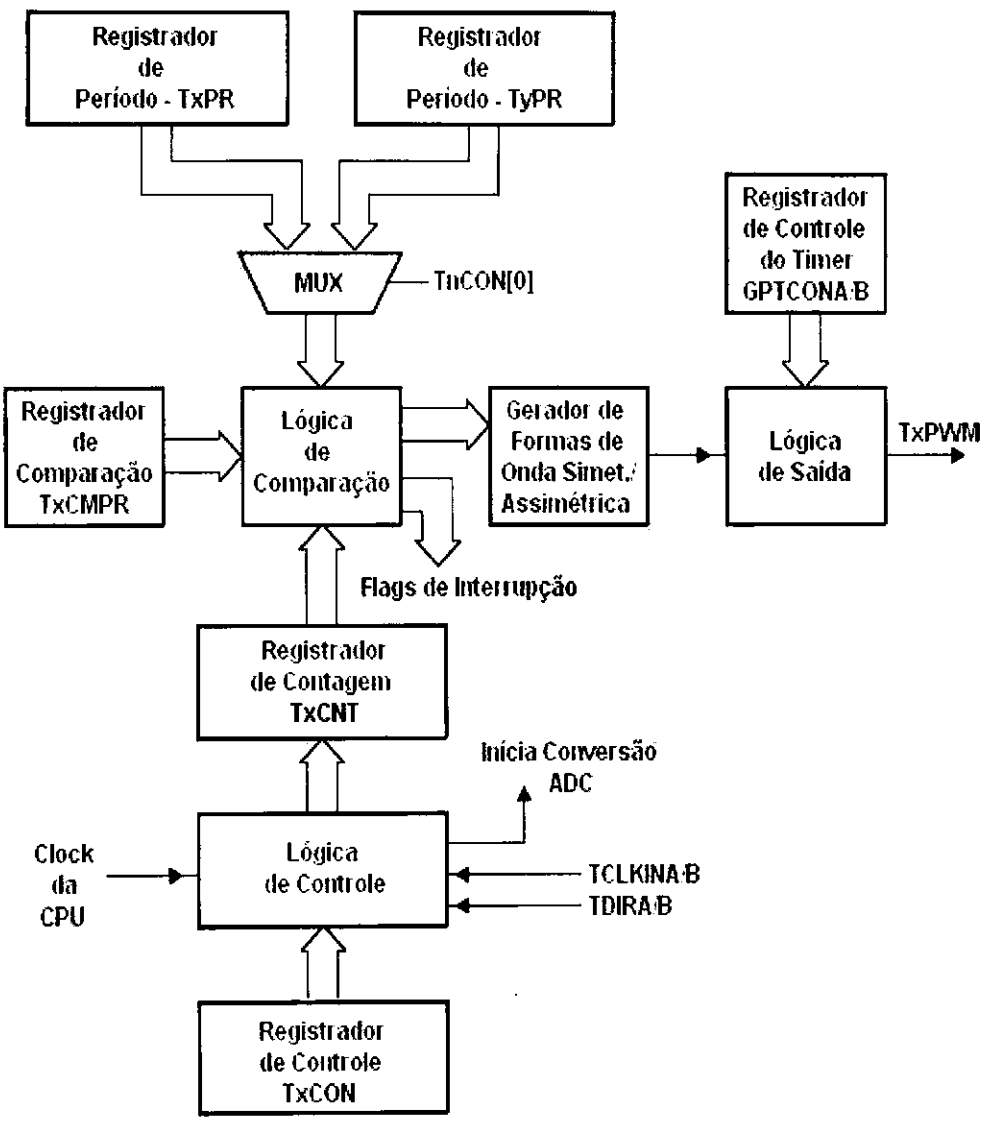


Figura 7.3: Diagrama de blocos de um *timer*.

O *timer 2* pode selecionar o registrador de período do *timer 1* como seu registrador de período. Veja na figura 7.3 que o mux é aplicável somente quando a figura representa o *timer 2*. O mesmo acontece com o *timer 4* e o *timer 3*. O *timer 4* pode selecionar o registrador de período do *timer 3*.

7.2.1 Registrador de Controle - TxCON

O modo de operação de um *timer* é controlado pelo **TxCON**. Os *bits* do **TxCON** determinam:

- Em qual dos quatro modos de contagem o *timer* está operando;
- Se um *clock* interno ou externo está sendo usado pelo *timer*;
- Em qual dos oito fatores do prescaler *clock* interno está sendo usado (variando de 1 à 1/128);
- Em qual condição o registrador de comparação do *timer* está sendo recarregado;
- Se o *timer* está habilitado ou desabilitado;
- Se a operação de comparação do *timer* está habilitado ou desabilitado;
- Qual registrador de período está sendo utilizado pelo *timer 2*, o dele mesmo ou o registrador de período do *timer 1*;
- Qual registrador de período está sendo utilizado pelo *timer 4*, o dele mesmo ou o registrador de período do *timer 3*;

7.2.2 Registradores de Comparação do Timer

O registrador de comparação associado com um *timer* armazena o valor a ser constantemente comparado com o contador do *timer*. Quando o valor a ser comparado for igual ao valor do contador do *timer*, os seguintes eventos podem acontecer:

- Uma transição ocorrerá na associada saída de comparação de acordo com a configuração no registrador **GPTCONA/GBTCONB**;
- O correspondente *flag* de interrupção será setado;
- Uma requisição de interrupção do periférico será gerado se a interrupção estiver habilitada;

7.2.3 Registrador de Período do Timer

O valor do registrador de período de um *timer* determina o período do *timer*. O contador do *timer* será resetado, ou iniciará um contagem decrescente quando ocorrer uma igualdade na comparação entre o registrador de período e o contador do *timer*, dependendo de qual modo de contagem o *timer* for acionado.

7.2.4 Registradores de Duplo Armazenamento

Os registradores **TxCMPR** e **TxPR** de um *timer*, possuem um registrador que armazena o valor que será carregado neles posteriormente, esse registrador é chamado de registrador *shadow*. Para o registrador de comparação, o conteúdo do registrador *shadow* será carregado no registrador **TxCMPR**, somente quando um evento especificado pelo registrador **TxCON** ocorrer. Já o registrador de período será carregado com o valor do seu registrador *shadow*, apenas quando o valor do registrador de contagem **TxCNT** for zero. A condição no qual um registrador de comparação será recarregado, pode ser uma das seguintes:

- Imediatamente após o registrador *shadow* ser escrito;
- Um *underflow*, isto é, quando o valor do contador do *timer* for zero;
- Um *underflow* ou **TxCNT = TxPR**.

A característica de duplo armazenamento dos registradores de período e de comparação, permitem que o código da aplicação faça uma atualização desses registradores em algum momento durante um período de forma a alterar o período do *timer* e a largura do pulso de um sinal PWM no período seguinte. Alterar o valor do período do *timer*, no caso da geração de um sinal PWM, significa alterar a frequência da portadora do PWM.

O registrador de período de um *timer* deverá ser inicializado para um valor diferente de zero antes do contador ser inicializado. Caso contrário, o valor do registrador de período permanecerá inalterado até o próximo *underflow*.

7.2.5 Direção de Contagem do Timer

As direções de contagem do *timer* são refletidos pelos respectivos *bits* do **GPTCONA/GBTCONB** durante todas operações do *timer*, como segue:

- Se for '0', a direção de contagem é decrescente;
- Se for '1', a direção de contagem é crescente.

O pino de entrada TDIRA/B determina a direção de contagem quando um *timer* está no modo de contagem direcional crescente/decrescente. Quando TDIRA/B estiver em nível alto, uma contagem crescente será especificada. Quando TDIRA/B estiver em nível baixo, uma contagem decrescente será especificada.

7.2.6 Clock do Timer

A fonte de *clock* do *timer* pode ser o *clock* interno ao dispositivo ou um *clock* externo (pino TCLK-INA/B). A frequência do *clock* externo deverá ser menor que ou igual a 1/4 do *clock* do dispositivo. O *timer 2* (EVA) e o *timer 4* (EVB) podem ser usados com os circuitos QEP (*Quadrature Encoder Pulse*), no modo direcional de contagem crescente/decrescente. Neste caso, os circuitos QEP fornecem ambas as entradas do *clock* e a direção ao *timer*.

7.2.7 Sincronização do Timer

O *timer 2* pode ser sincronizado com o *timer 1* e o *timer 4* pode ser sincronizado com o *timer 3*, pela configuração de **T2CON** e **T4CON**, respectivamente, nos seguintes modos:

- **EVA/EVB** - Setar o *bit* T2SWT1/T4SWT3 do **T2CON/T4CON** para iniciar a contagem do *timer 2/4* com o *bit* TENABLE do **T1CON/T3CON**. Deste modo, ambos contadores do *timer* iniciam simultaneamente;
- Inicializar os contadores dos *timers 1 e 2/timers 3 e 4* com valores diferentes antes de iniciar a operação de sincronização;
- Especificar que o *timer 2/4* utiliza o registrador de período do *timer 1/3* como registrador de período, setando o *bit* SELT1PR/SELT3PR no registrador **T2CON/T4CON**.

Isto permite uma desejada sincronização entre os eventos do *timer*. Desde que cada *timer* inicia a operação de contagem do valor atual no registrador de contagem, um *timer* pode ser programado para iniciar com um atraso conhecido, após o outro *timer*.

7.2.8 Iniciando o Conversor A/D com um Evento do Timer

Os *bits* do registrador **GPTCONA/GBTCONB** podem especificar que um sinal de inicialização do conversor A/D seja gerado por um evento do *timer*. Esta característica fornece uma sincronização entre o evento do *timer* e o início da conversão A/D sem intervanção da CPU.

7.2.9 Suspensão da Emulação do Timer

Os *bits* dos registradores de controle do *timer* também definem a operação dos *timers* durante a suspensão da emulação. Estes *bits* podem ser setados para permitir que os *timers* continuem funcionando, quando ocorrer uma interrupção da emulação. Eles também podem ser setados para especificar que a operação dos *timers* parem imediatamente, ou após terminar o atual período de contagem, quando ocorrer uma interrupção da emulação.

A suspensão da emulação ocorre quando o *clock* do dispositivo for parado pelo emulador, por exemplo, quando o emulador encontrar um *break point*.

7.2.10 Interrupções do Timer

Existem dezesseis *flags* de interrupção nos registradores **EVAIFRA**, **EVAIFRB**, **EVBIFRA** e **EVBIFRB** para sinalizar eventos nos *timers*. Cada um dos quatro *timers* podem gerar até quatro interrupções nos seguintes eventos:

- ***Overflow***, TxOFINT (x=1,2,3,4);
- ***Underflow***, TxUFINT (x=1,2,3,4);
- **TxCNT = TxCMPR**, TxCINT (x=1,2,3,4);
- **TxCNT = TxPR**, TxPINT (x=1,2,3,4).

Um evento de *overflow* ocorrerá quando o valor do contador da *timer* atingir FFFFh e um evento de *underflow* ocorrerá quando o contador do *timer* atingir o valor 0000h. Veja que as definições de *overflow* e *underflow* são diferentes das definições convencionais.

Na ocorrência de cada um desses eventos, o respectivo *flag* de interrupção será setado após um ciclo de *clock* e uma requisição de interrupção do periférico será assegurada, se a respectiva interrupção estiver habilitada.

7.2.11 Operação de Contagem do Timer

Cada *timer* pode operar em quatro diferentes modos:

- Modo *stop/hold*;
- Modo de contagem contínua crescente;
- Modo de contagem direcional crescente/decrescente;
- Modo de contagem contínua crescente/decrescente.

7.2.12 Modo stop/hold

Neste modo, o *timer* pára e se mantém no atual estado. O contador do *timer*, a saída de comparação e o prescaler permanecem inalterados neste modo.

7.2.13 Modo de Contagem Contínuo Crescente

Neste modo, o o contador do *timer* conta de forma crescente de acordo com o *clock* de entrada escalado, até seu valor atingir o valor do registrador de período. Na próxima borda de subida após a ocorrência desse evento, o *timer* será resetado e a contagem crescente será iniciada novamente. O *flag* de interrupção de período do *timer* será setado um ciclo de *clock* após a ocorrência do evento. Uma requisição de interrupção do periférico será gerada se estiver habilitada. Um sinal para iniciar o ADC será enviado para o módulo ADC no mesmo instante que o *flag* for setado se a interrupção de período deste *timer* tiver sido selecionada pelos apropriados *bits* no registrador **GPTCONA/GBTCONB** para iniciar o ADC.

Um ciclo de clock após o contador do timer atingir o valor zero, o flag de interrupção de underflow do timer será setado. Uma requisição de interrupção será gerada se estiver habilitada. Um sinal para iniciar o ADC será enviado para o módulo ADC no mesmo instante que o *flag* for setado se a interrupção de *underflow* deste *timer* tiver sido selecionada pelos apropriados *bits* no registrador **GPTCONA/GBTCONB** para iniciar o ADC.

O *flag* de interrupção de *overflow* será setado um ciclo de *clock* após o valor de **TxCNT** atingir FFFFh. Uma requisição de interrupção do periférico será gerada se estiver habilitada.

A duração do período do *timer* é **(TxPR + 1)** ciclos do *clock* de entrada escalado, exceto no primeiro período. A duração do período será o mesmo se o contador do *timer* for zero quando a contagem for iniciada.

O valor inicial do *timer* pode ser algum valor entre 0000h e FFFFh. Quando o valor inicial for maior que o valor do registrador de período, o *timer* contará até FFFFh, será resetado e em seguida continuará a operação como se o valor inicial fosse zero. Quando o valor inicial do contador do *timer* for igual ao valor do registrador de período, o *timer* setará o *flag* da interrupção de período (TxPINT), em seguida será resetado e consequentemente o *flag* de interrupção de *underflow* será setado, e por fim a operação continuará novamente como se o valor inicial fosse zero. Se o valor inicial do contador do *timer* for um valor entre zero e o valor do registrador de período, o *timer* contará até o valor do registrador de período e continuará para finalizar o período como se o valor inicial do contador fosse igual ao valor do registrador de período.

O *bit* de indicação da direção da contagem no registrador **GPTCONA/GPTCONB** é sempre '1' nesse modo de operação. Ambos o *clock* interno ou externo ao dispositivo podem ser selecionados como entrada do *clock* do *timer*. A entrada TDIRA/TDIRB é ignorada pelo *timer* nesse modo de contagem. Na figura 7.4 está ilustrado um esquema de funcionamento do modo de contagem contínuo crescente de um *timer*.

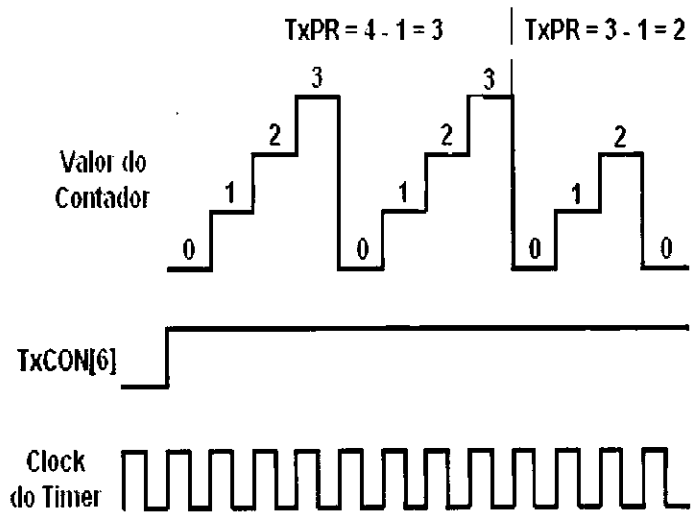


Figura 7.4: Esquema de funcionamento de um *timer* no modo de contagem contínuo crescente.

O modo de contagem contínuo crescente do timer é particularmente usado para a geração de formas de onda PWM assíncronas e amostragem de períodos em muitos sistemas de controle de motores e movimentos.

7.2.14 Modo de Contagem Direcional Crescente/Decrescente

O modo de contagem direcional crescente/decrescente do *timer* conta de forma crescente ou decrescente de acordo com o *clock* escalado e as entradas TDIRA/TDIRB.

O contador do *timer* inicia a contagem crescente até seu valor atingir o valor do registrador de período (ou FFFFh se o valor inicial for maior que o período) quando o pino TDIRA/TDIRB estiver mantido em nível alto. Quando o valor do contador do *timer* for igual ao valor do registrador de período (ou FFFFh) o contador será resetado e ocorrerá novamente uma contagem crescente até o valor do período. Quando TDIRA/TDIRB for mantido em nível baixo, o contador do *timer* contará de forma decrescente até o valor zero. Quando o contador atingir o valor zero, ele será recarregado com o valor do registrador de período e iniciará a contagem decrescente novamente. O valor inicial do contador do *timer* pode ser algum valor entre 0000h e FFFFh. Quando o valor inicial do contador for maior que o valor do registrador de período, o *timer* contará até FFFFh, será resetado e depois iniciará a contagem até o valor do período. Se TDIRA/TDIRB estiver em nível baixo quando o *timer* iniciar a contagem com um valor maior que o valor do registrador de período, ele contará até o valor do registrador de período e continuará a contagem decrescente até atingir o valor zero, neste ponto o contador do *timer* será recarregado com o valor do registrador de período como o normal.

Os *flags* de interrupção de *overflow*, *underflow* e de período, as interrupções e as ações que são geradas nos respectivos eventos são gerenciadas da mesma forma que no modo de contagem contínua crescente.

A latência de uma alteração do TDIRA/TDIRB para mudar a direção de contagem é de um ciclo de *clock* após o fim da atual contagem.

A direção de contagem é indicada pelo *timer* neste modo pelo correspondente *bit* de indicação de direção no registrador **GPTCONA/GPTCONB**: o valor '1' significa que a contagem é cerscente e o valor '0' significa que a contagem é decrescente. Ambos o clock interno ou externo ao dispositivo podem ser utilizados neste modo de contagem. Na figura 7.5 está ilustrado um esquema de funcionamento do modo de contagem direcional crescente/decrescente de um *timer*.

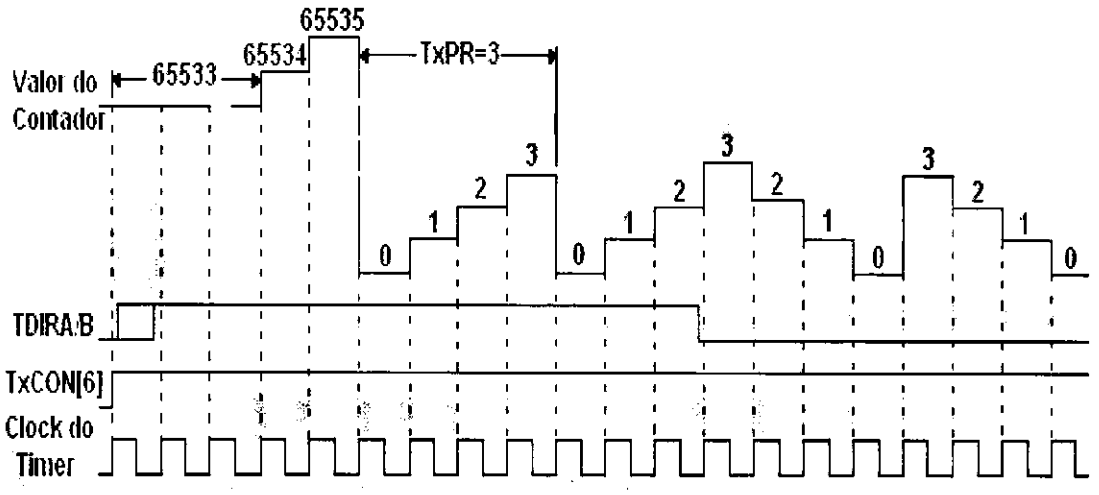


Figura 7.5: Esquema de funcionamento de um *timer* no modo de contagem direcional crescente/decrescente.

Esse modo de contagem no *timer 2/4* pode ser usado com os circuitos QEP do módulo EV. Os circuitos QEP fornecem o *clock* de contagem e a direção do *timer 2/4* neste caso. Este modo de operação pode também ser usado para marcar o tempo de ocorrência de eventos externos em aplicações na eletrônica de potência e controle de motores e de movimento.

7.2.15 Modo de Contagem Contínuo Crescente/Decrescente

Este modo de operação é semelhante ao modo de contagem direcional crescente/decrescente, com a excessão de que o pino TDIRA/TDIRB não tem efeito na direção da contagem. A direção de contagem apenas muda de crescente para decrescente quando o *timer* atingir o valor do período (ou FFFFh se o valor inicial do *timer* for maior que o valor do período). A direção do *timer* muda somente de decrescente para crescente quando o contador atingir o valor zero.

O período do *timer* neste modo é **2*TxPR** ciclos do *clock* de entrada escalado, exceto para o primeiro período. A duração deste primeiro período de contagem somente será igual se o contador do *timer* for zero no inicio da contagem.

O valor inicial do contador do *timer* pode ser um valor entre 0 e FFFFh. Quando o valor inicial for maior que o valor do registrador de período, ele contará até FFFFh, depois será resetado e em seguida continuará a operação como se o valor inicial fosse zero. Quando o valor inicial do contador for igual ao valor do registrador de período, o *timer* contará de forma decrescente até zero e continuará novamente como se o valor inicial fosse zero. Se o valor inicial do contador estiver entre zero e o

conteúdo do registrador de período. o contador contará de forma crescente até o valor do registrador de período e continuará até o fim do período como se o valor inicial fosse igual ao valor do registrador de período.

Os *flags* de interrupção de *overflow*, *underflow* e de período, as interrupções e as ações que são geradas nos respectivos eventos são gerenciadas da mesma forma que no modo de contagem contínua crescente.

O *bit* de indicação de direção de contagem para este *timer* no registrador **GBTCONA/GBTCONB** será um quando o timer contar de forma crescente e zero quando contar de forma decrescente. Ambos o *clock* interno ou externo ao dispositivo podem ser selecionados como entrada do *clock* do *timer*. A entrada TDIRA/TDIRB é ignorada pelo *timer* nesse modo de contagem. Na figura 7.6 está ilustrado um esquema de funcionamento do modo de contagem direcional crescente/decrescente de um *timer*.

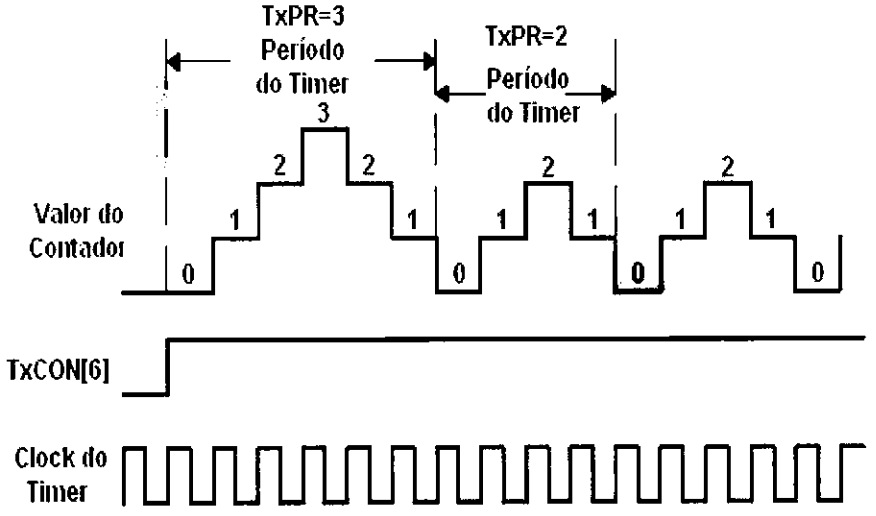


Figura 7.6: Esquema de funcionamento de um *timer* no modo de contagem contínuo crescente/decrescente.

Esse modo de contagem é utilizado na geração de formas de onda PWM simétricas para aplicações na eletrônica de potência.

7.2.16 Operação de Comparação do Timer

Cada *timer* tem um associado registrador de comparação **(TxCMPR)** e um pino de saída PWM **(Tx-PWM)**. O valor do contador do *timer* é constantemente comparado com o valor do associado registrador de comparação até quando ocorrer uma igualdade na comparação, isto é, **TxCNT = TxCMPR**. A operação de comparação é habilitada no *bit* 1 do registrador **TxCON**. Se ele estiver habilitado, e ocorrer uma igualdade na comparação, pode ocorrer o seguinte:

- O *flag* de interrupção de comparação do *timer* será setado um ciclo de *clock* após ocorrer o evento;
- Uma transição ocorrerá na associada saída PWM, de acordo com a configuração do *bit* no registrador **GPTCONA/GPTCONB**, um ciclo de *clock* após ocorrer o evento;
- Se o *flag* de interrupção de comparação tiver sido selecionado para iniciar o conversor A/D, um sinal de início de conversão A/D será gerado no instante que o *flag* de interrupção de comparação for setado.

Uma requisição de interrupção do periférico será gerada se a interrupção estiver habilitada.

7.2.17 Transição do PWM

A transição na saída PWM é controlada por uma gerador de forma de onda simétrico e assimétrico e a associada lógica de saída. O gerador de forma de onda simétrico/assimétrico gera uma forma de onda PWM simétrica ou assimétrica baseado no modo de contagem do *timer* que estiver sendo utilizado.

Uma forma de onda assimétrica é gerada quando timer está no modo de contagem contínuo crescente. Na figura 7.7 está ilustrado o processo de geração de um PWM assimétrica. Uma característica das formas de Onda PWM assimétricas é que uma mudança no valor do registrador de comparação afeta apenas um lado do pulso do PWM. Uma forma de onda simétrica é gerada quando o timer está no modo de contagem contínuo crescente/decrescente. Na figura 7.8 está ilustrado o processo de geração de um PWM simétrica.

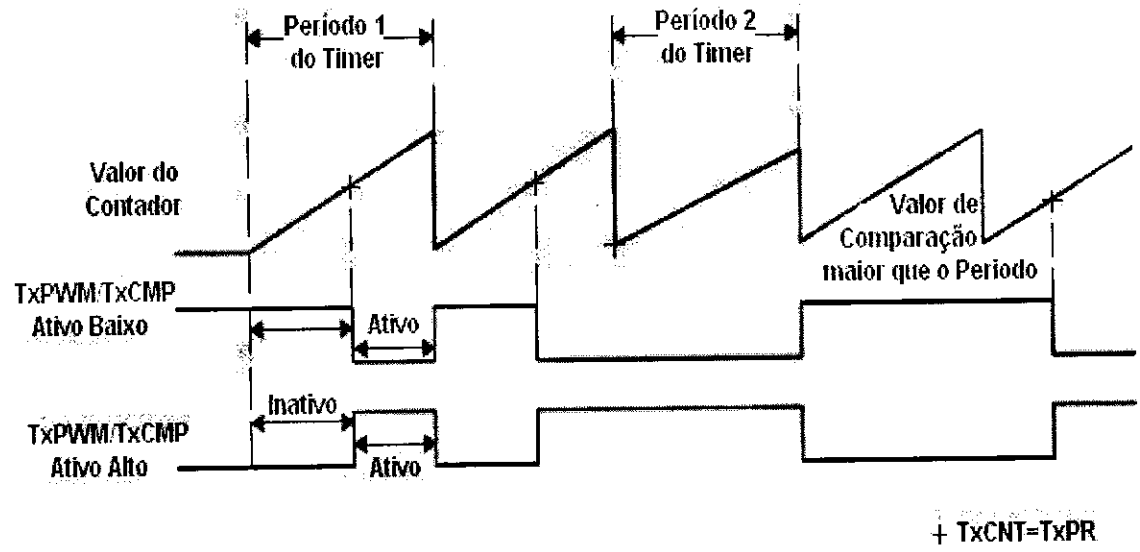


Figura 7.7: Esquema do processo de geração de um PWM assimétrico.

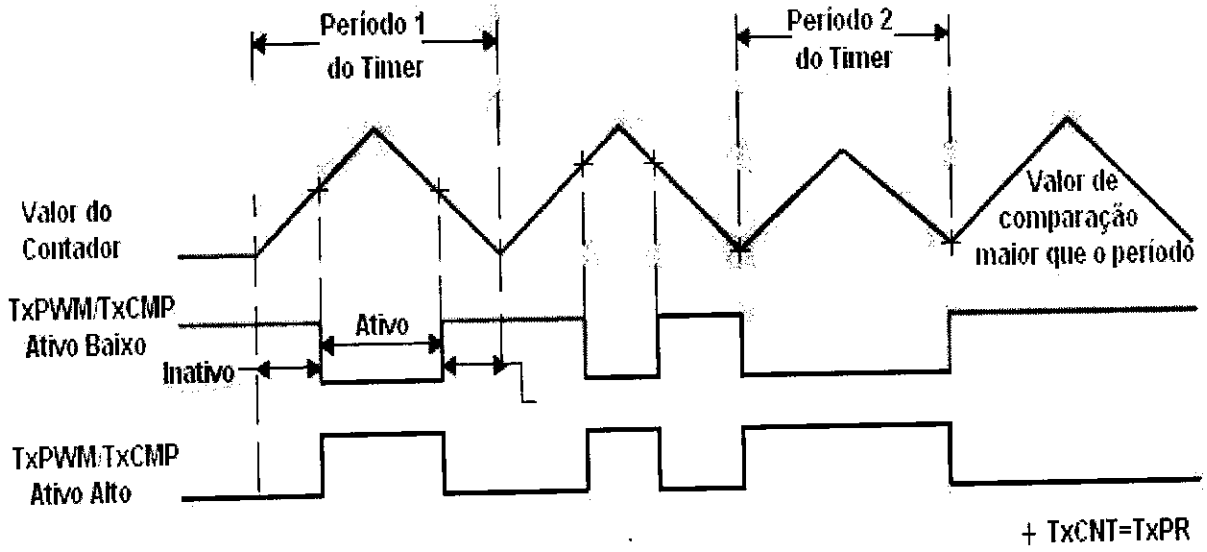


Figura 7.8: Esquema do processo de geração de um PWM simétrico.

7.2.18 Registradores de Controle do Timer (TxCON e GPTCONA/B)

Nas figuras 7.9, 7.10 e 7.11 estão ilustrados, respectivamente, os esquemas com as posições dos *bits* nos registradores **TxCON, GPTCONA** e **GPTCONB**.

15	14	13	12	11	10	9	8
Free	Soft	Reservado	TMODE1	TMODE0	TPS2	TPS1	TPS0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7	6	5	4	3	2	1	0
T2SWT1/ T4SWT3†	TENABLE	TCLKS1	TCLKS0	TCLD1	TCLD0	TECMPR	SELT1PR/ SELT3PR†
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Figura 7.9: Esquema com as posições dos *bits* no registrador TxCON.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bits 15-14 - Free, Soft** - *Bits* de controle da emulação.

  00 - Pára imediatamente na suspensão da emulação.

  01 - Pára após o atual período do timer terminar na suspensão da emulação.

  10 - Operação não é afetada pela suspensão da emulação.

  11 - Operação não é afetada pela suspensão da emulação.

- **Bits 12-11 - TMODE1-TMODE0** - Selecionam o modo de contagem.

  00 - *Stop/Hold.*

  01 - Modo de contagem contínuo crescente/decrescente.

  10 - Modo de contagem contínuo crescente.

  11 - Modo de contagem direcional crescente/decrescente.

- **Bits 10-8 - TPS2-TPS0** - *Prescaler* do *clock* de entrada.

  000 - x/1.

  001 - x/2.

  010 - x/4.

  011 - x/8.

100 - x/16.

101 - x/32.

110 - x/64.

111 - x/128.

x = Frequência do *clock* do dispositivo (CPUCLK ou CLKOUT)

- **Bit 7 - T2SWT1/T4SWT3*** - Este *bit* habilita o *timer 2/timer 4* iniciar a operação com o *timer 1/timer3*. No caso do EVA este *bit* é o T2SWT1, e no caso do EVB este *bit* é o T4SWT3.

  0 - Utiliza o seu próprio *bit* TENABLE para iniciar a operação.

  1 - Utiliza o *bit* TENABLE do **T1CON** ou do **T3CON** para habilitar e desabilitar a operação no *timer*, ignorando o seu próprio *bit* TENABLE.

  * T2SWT1/T4SWT3 é reservado nos registradores **T1CON** e **T3CON**.

- **Bit 6 - TENABLE** - Habilita o *timer*.

  0 - Desabilita a operação do *timer*.

  1 - Habilita a operação do *timer*.

- **Bit 5-4 - TCLKS1-TCLKS0** - Seleciona a fonte do *clock*.

  00 - Fonte do *clock* é interna ao dispositivo.

  01 - Fonte do *clock* é externa ao dispositivo.

  10 - Reservado.

  11 - Fonte do *clock* é gerado pelo circuito QEP no caso de utilizar o *timer 2* ou o *timer 4*. Reservado no caso de utilizar o *timer 1* ou *timer 3*.

  OBS.: O circuito QEP fornecerá a fonte do *clock* somente se o *bit* SELT1PR for zero.

- **Bit 3-2 - TCLD1-TCLD0** - Condição de recarga do registrador de comparação.

  00 - Quando o contador for igual a zero.

  01 - Quando o contador for zero ou igual ao valor do registrador de período.

  10 - Imediatamete.

  11 - Reservado.

- **Bit 1 - TECMPR** - Habilita o processo de comparação no timer.

  0 - Desabilita o processo de comparação.

  1 - Habilita o processo de comparação.

- **Bit 0 - SELT1PR/SELT3PR*** - No caso do EVA, este *bit* é o SELT1PR (seleciona registrador de período). Quando este *bit* for setado no registrador **T2CON**, o registrador de período do *timer 1* será utilizado pelo *timer 2* também, ignorando o registrador de período do *timer 2*. Este *bit* é reservado no **T1CON**. No caso do EVB, este *bit* é o SELT3PR (seleciona registrador de período). Quando este *bit* for setado no registrador **T4CON**, o registrador de período do *timer 3* será utilizado pelo *timer 4* também, ignorando o registrador de período do *timer 4*. Este *bit* é reservado no **T3CON**.

  0 - Utiliza o seu próprio registrador de período.

  1 - Utiliza o registrador de período **T1PR** (no caso do EVA) ou o registrador de período **T3PR** (no caso do EVB), ignorando o seu próprio registrador de período.

  * SELT1PR/SELT3PR é reservado nos registradores **T1CON** e **T3CON**.

15	14	13	12–11	10–9	8–7
Reservado	T2STAT	T1STAT	Reservado	T2TOADC	T1TOADC
RW-0	R-1	R-1	RW-0	RW-0	RW-0

6	5–4	3–2	1–0
TCOMPOE	Reservado	T2PIN	T1PIN
RW-0	RW-0	RW-0	RW-0

Figura 7.10: Esquema com as posições dos *bits* no registrador GPTCONA.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bits 14 - T2STAT** - *Bit* de *status* do *timer 2*.

  0 - Contagem decrescente.

  1 - Contagem crescente.

- **Bits 13 - T1STAT** - *Bit* de *status* do *timer 1*.

  0 - Contagem decrescente.

  1 - Contagem crescente.

- **Bits 10-9 - T2TOADC** - Inicia a conversão A/D com um evento do *timer 2*.

  00 - Nenhum evento inicia a conversão A/D.

  01 - A ocorrência de um *underflow* inicia a conversão A/D.

  10 - Uma igualdade entre os valores do contador do *timer 2* e o seu registrador de período **(T2CNT = T2PR)** inicia a conversão A/D.

  11 - Uma igualdade entre os valores do contador do *timer 2* e o seu registrador de comparação **(T2CNT = T2CMPR)** inicia a conversão A/D.

- **Bits 8-7 - T1TOADC** - Inicia a conversão A/D com um evento do *timer 1*.

  00 - Nenhum evento inicia a conversão A/D.

  01 - A ocorrência de um *underflow* inicia a conversão A/D.

  10 - Uma igualdade entre os valores do contador do *timer 1* e o seu registrador de período **(T1CNT = T1PR)** inicia a conversão A/D.

  11 - Uma igualdade entre os valores do contador do *timer 1* e o seu registrador de comparação **(T1CNT = T1CMPR)** inicia a conversão A/D.

- **Bits 6 - TCOMPOE** - Habilita a saída de comparação. Se PDPINTx for zero, este *bit* será zero também.

  0 - Desabilita todas as saídas de comparação (Todas as saídas de comparação são colocadas em estado de alta impedância).

  1 - Habilita todas as saídas de comparação do *timer*.

- **Bit 3-2 - T2PIN** - Polaridade da saída de comparação do *timer 2*.

  00 - Forçado baixo.

  01 - Ativo baixo.

  10 - Ativo alto.

  11 - Forçado alto.

- **Bit 1-0 - T1PIN** - Polaridade da saída de comparação do *timer 1*.

  00 - Forçado baixo.

  01 - Ativo baixo.

  10 - Ativo alto.

  11 - Forçado alto.

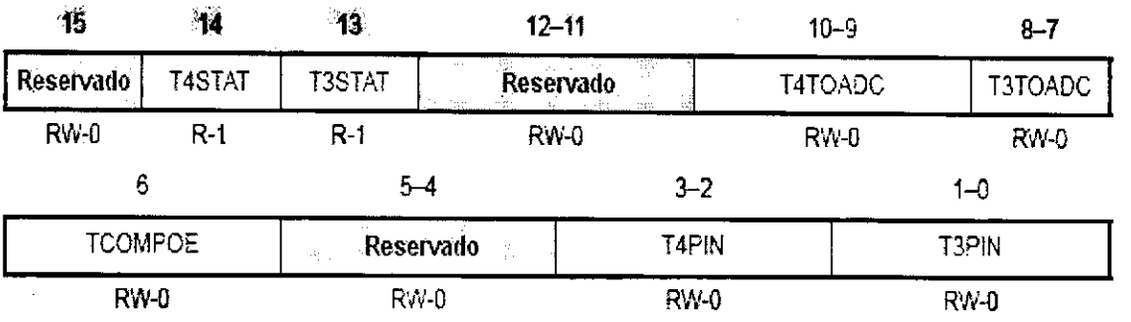


Figura 7.11: Esquema com as posições dos *bits* no registrador GPTCONB.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bits 14 - T4STAT** - *Bit* de *status* do *timer 4*.

  0 - Contagem decrescente.

  1 - Contagem crescente.

- **Bits 13 - T3STAT** - *Bit* de *status* do *timer 3*.

  0 - Contagem decrescente.

  1 - Contagem crescente.

- **Bits 10-9 - T4TOADC** - Inicia a conversão A/D com um evento do *timer 4*.

  00 - Nenhum evento inicia a conversão A/D.

  01 - A ocorrência de um *underflow* inicia a conversão A/D.

  10 - Uma igualdade entre os valores do contador do *timer 4* e o seu registrador de período **(T4CNT = T4PR)** inicia a conversão A/D.

  11 - Uma igualdade entre os valores do contador do *timer 4* e o seu registrador de comparação **(T4CNT = T4CMPR)** inicia a conversão A/D.

- **Bits 8-7 - T3TOADC** - Inicia a conversão A/D com um evento do *timer 3*.

  00 - Nenhum evento inicia a conversão A/D.

  01 - A ocorrência de um *underflow* inicia a conversão A/D.

  10 - Uma igualdade entre os valores do contador do *timer 3* e o seu registrador de período **(T3CNT = T3PR)** inicia a conversão A/D.

  11 - Uma igualdade entre os valores do contador do *timer 3* e o seu registrador de comparação **(T3CNT = T3CMPR)** inicia a conversão A/D.

- **Bits 6 - TCOMPOE** - Habilita a saída de comparação. Se PDPINTx for zero, este *bit* será zero também.

  0 - Desabilita todas as saídas de comparação (Todas as saídas de comparação são colocadas em estado de alta impedância).

  1 - Habilita todas as saídas de comparação do *timer*.

- **Bit 3-2 - T4PIN** - Polaridade da saída de comparação do *timer 4*.

  00 - Forçado baixo.

  01 - Ativo baixo.

  10 - Ativo alto.

  11 - Forçado alto.

- **Bit 1-0 - T3PIN** - Polaridade da saída de comparação do *timer 3*.

  00 - Forçado baixo.

  01 - Ativo baixo.

  10 - Ativo alto.

  11 - Forçado alto.

7.3 Unidades de Comparação

Existem três unidades de comparação em cada módulo EV. Cada unidade de comparação possui duas saídas PWM associadas. A base de tempo para as unidades de comparação são fornecidas pelo *timer 1* (para o EVA) e *timer 2* (para o EVB).

As unidades de comparação de cada módulo EV incluem:

- Três registradores de comparação de 16 *bits* (**CMPR1**, **CMPR2** e **CMPR3** para o EVA, e **CMPR4**, **CMPR5** e **CMPR6** para o EVB);
- Um registrador de controle de comparação (**COMCONA** para o EVA, e **COMCONB** para o EVB);
- Um registrador de controle de ação (**ACTRA** para o EVA, e **ACTRB** para EVB);
- Seis pinos de saída PWM (PWMx, x = 1 à 6 para o EVA, e PWMy, y = 7 à 12 para o EVB);
- Lógica de controle e interrupção.

Na figura 7.12 está ilustrado o diagrama de blocos de uma unidade de comparação.

Para o EVA: x = 1, 2, 3; y = 1, 3, 5; z = 1. Para o EVB: x = 4, 5, 6; y = 7, 9, 11; z = 2.

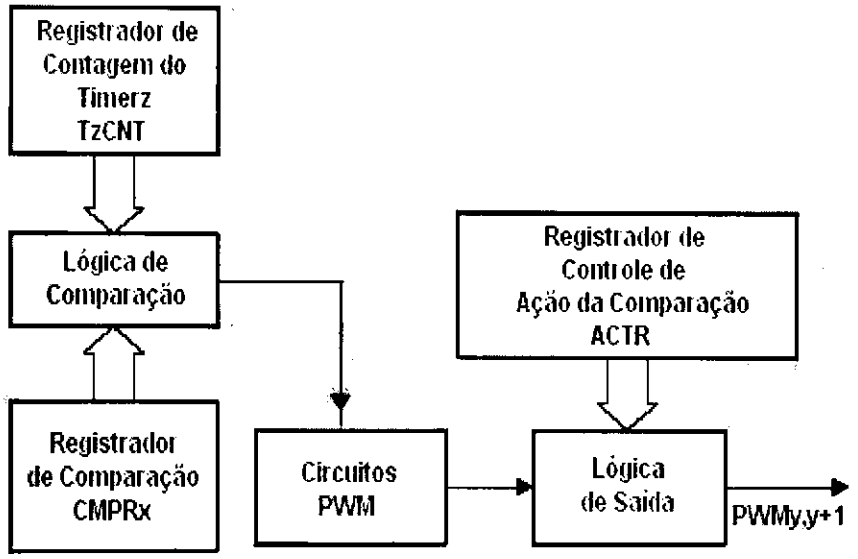


Figura 7.12: Diagrama de blocos de uma unidade de comparação.

7.3.1 Entradas e Saídas de uma Unidade de Comparação

As entradas de uma unidade de comparação incluem:

- Sinais de controle dos registradores de controle;
- Sinais do *timer1/3* (**T1CNT/T3CNT**) e sinais de *underflow* e de igualdade nas comparações com registrador de periodo;
- *RESET*.

A saída de uma unidade de comparação é um sinal gerado pela igualdade em uma comparação (entre o contador e o registrador de periodo). Se a operação de comparação estiver habilitada, este sinal seta o *flag* de interrupção e isto causará transições nos pinos de saída associados com cada unidade de comparação.

7.3.2 Modos de Operação de Comparação

O modo de operação das unidades de comparação é determinado pelos *bits* do registrador **COMCONA/B**. Esses *bits* determinam:

- Se a operação de comparação está habilitada;
- Se as saídas de comparação estão habilitadas;
- A condição na qual os registradores de comparação serão carregados;
- Se o modo SVPWM está habilitado.

7.3.3 Operação de uma Unidade de Comparação

A seguir descreve-se a operação da unidade de comparação do EVA. A operação da unidade de comparação do EVB é idêntica. Para EVB, utiliza-se o *Timer 3* e o registrador **ACTRB**.

O valor do contador do *Timer 1* é continuamente comparado com o valor do registrador de comparação (**CMPRn**). Quando ocorrer uma igualdade na comparação, uma transição ocorrerá nas duas saídas da unidade de comparação de acordo com os *bits* do registrador de controle de ação (**ACTRA**). Os *bits* do **ACTRA** podem individualmente especificar o nível de cada saída (ativo-alto ou ativo-baixo, se não forçado alto ou forçado baixo) em uma igualdade na comparação.

O *flag* de interrupção de comparação associado com uma unidade de comparação será setado quando ocorrer uma igualdade na comparação, se a comparação estiver habilitada. Uma requisição de interrupção do periférico será gerada pelo *flag* se a interrupção estiver habilitada. As saídas das unidades de comparação no modo de comparação estão sujeitas a modificações pelas lógicas de saída, unidades de banda morta e lógic SVPWM (Modulação Vetorial).

Para configurar uma unidade de comparação, a seguinte sequência de operação nos seguintes registradores indicados na tabela 7.2 deverá ser realizada.

Tabela 7.2: Sequência de operação para configurar as unidades de comparação.

Para EVA	**Para o EVB**
Setar **T1PR**	Setar **T3PR**
Setar **ACTRA**	Setar **ACTRB**
Inicializar **CMPRn**	Inicializar **CMPRn**
Setar **COMCONA**	Setar **COMCONB**
Setar **T1CON**	Setar **T3CON**

7.3.4 Registradores de Controle da Comparação - COMCONA/B

A operação das unidades de comparação são controladas pelos registradores **COMCONA** e **COMCONB**. Nas figuras 7.13 e 7.14 estão ilustrados, respectivamente, os esquemas com as posições dos *bits* nos registradores **COMCONA** e **COMCONB**.

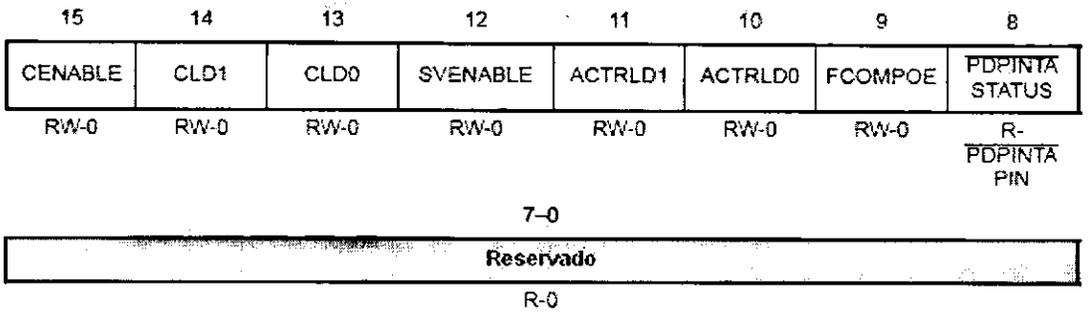


Figura 7.13: Esquema com as posições dos *bits* no registrador COMCONA.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bit 15 - CENABLE** - Habilita a comparação.

  0 - Desabilita a operação de comparação.

  1 - habilita a operação de comparação.

- **Bits 14-13 - CLD1, CLD0** - Condição de recarregamento do registrador de comparação (**CMPRn**).

  00 - Quando **T1CNT** = 0, isto é, quando ocorrer *underflow* em **T1CNT**.

  01 - Quando **T1CNT** = 0 ou **T1CNT** = **T1PR**.

  10 - Imediatamente.

  11 - Reservado; o resultado é imprevisível.

- **Bit 12 - SVENABLE** - Habilita o modo SVPWM.

  0 - Desabilita o modo SVPWM.

  1 - Habilita o modo SVPWM.

- **Bits 11-10 - ACTRLD1, ACTRLD0** - CondiçãO de recarregamento do registrador **ACTRA**.

  00 - Quando **T1CNT** = 0, isto é, quando ocorrer *underflow* em **T1CNT**.

  01 - Quando **T1CNT** = 0 ou **T1CNT** = **T1PR**.

  10 - Imediatamente.

  11 - Reservado; o resultado é imprevisível.

- **Bit 9 - FCOMPOE** - Habilita a saída de comparação. Se PDPINTA for '0', esse *bit* será zerado.

  0 - Os pinos de saída PWM estão em estado de alta impedância, isto é, eles estão desabilitados.

  1 - Os pinos de saída PWM não estão em estado de alta impedância, isto é, eles estão habilitados.

- **Bit 8 - PDPINTA STATUS** - Este *bit* Reflete o *status* do pino PDPINTA.

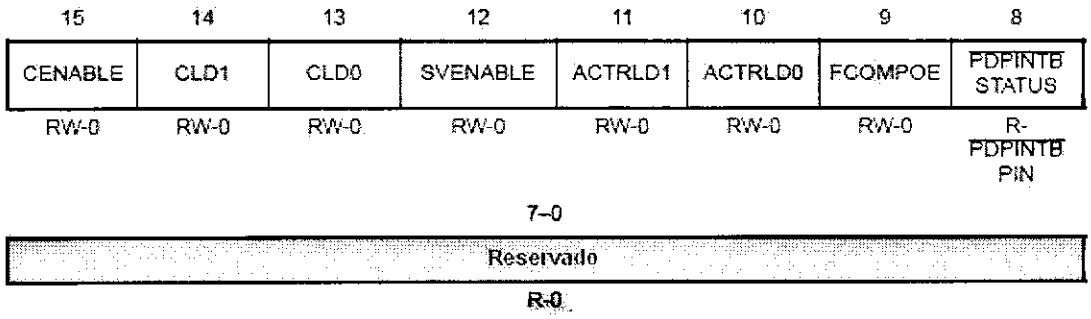


Figura 7.14: Esquema com as posições dos *bits* no registrador COMCONB.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bit 15 - CENABLE** - Habilita a comparação.

  0 - Desabilita a operação de comparação.

  1 - habilita a operação de comparação.

- **Bits 14-13 - CLD1, CLD0** - Condição de recarregamento do registrador de comparação (**CMPRn**).

  00 - Quando **T3CNT** = 0, isto é, quando ocorrer *underflow* em **T3CNT**.

  01 - Quando **T3CNT** = 0 ou **T3CNT** = **T3PR**.

  10 - Imediatamente.

  11 - Reservado; o resultado é imprevisível.

- **Bit 12 - SVENABLE** - Habilita o modo SVPWM.

  0 - Desabilita o modo SVPWM.

  1 - Habilita o modo SVPWM.

- **Bits 11-10 - ACTRLD1, ACTRLD0** - CondiçãO de recarregamento do registrador **ACTRB**.

  00 - Quando **T3CNT** = 0, isto é, quando ocorrer *underflow* em **T3CNT**.

  01 - Quando **T3CNT** = 0 ou **T3CNT** = **T3PR**.

  10 - Imediatamente.

  11 - Reservado; o resultado é imprevisível.

- **Bit 9 - FCOMPOE** - Habilita a saída de comparação. Se PDPINTB for '0', esse *bit* será zerado.

  0 - Os pinos de saída PWM estão em estado de alta impedância, isto é, eles estão desabilitados.

  1 - Os pinos de saída PWM não estão em estado de alta impedância, isto é, eles estão habilitados.

- **Bit 8 - PDPINTB STATUS** - Este *bit* Reflete o *status* do pino PDPINTB.

7.3.5 Registradores de Controle de Ação da Comparação - ACTRA/B

Os registradores **ACTRA** e **ACTRB** controlam a ação a ser tomada em cada um dos seis pinos de saída de comparação (PWMx, onde x = 1-6 para o **ACTRA**, e x = 7-12 para o **ACTRB**) em um evento de comparação, se a operação de comparação foi habilitada pelo *bit* 15 do **CONCONA/B**. Nas figuras 7.15 e 7.16 estão ilustrados, respectivamente, os esquemas com as posições dos *bits* nos registradores **ACTRA** e **ACTRB**.

15	14	13	12	11	10	9	8
SVRDIR	D2	D1	D0	CMP6ACT1	CMP6ACT0	CMP5ACT1	CMP5ACT0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7	6	5	4	3	2	1	0
CMP4ACT1	CMP4ACT0	CMP3ACT1	CMP3ACT0	CMP2ACT1	CMP2ACT0	CMP1ACT1	CMP1ACT0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Figura 7.15: Esquema com as posições dos *bits* no registrador ACTRA.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bit 15 - SVRDIR** - Direção de rotação do SVPWM.

  0 - positiva (CCW).

  1 - Negativa (CW).

- **Bits 14-12 - D2, D1 e D0** - Indicam o atual setor do vetor no SVPWM.

- **Bits 11-10 - CMP6ACT1-0** - Ação a ser tomada no pino 6 saída de comparação, CMP6.

  00 - Forçado baixo.

  01 - Ativo baixo.

  10 - Ativo alto.

  11 - Forçado alto.

- **Bits 9-8 - CMP5ACT1-0** - Ação a ser tomada no pino 5 saída de comparação, CMP5.

  00 - Forçado baixo.

  01 - Ativo baixo.

  10 - Ativo alto.

  11 - Forçado alto.

- **Bits 7-6 - CMP4ACT1-0** - Ação a ser tomada no pino 4 saída de comparação, CMP4.

  00 - Forçado baixo.

  01 - Ativo baixo.

  10 - Ativo alto.

  11 - Forçado alto.

- **Bits 5-4 - CMP3ACT1-0** - Ação a ser tomada no pino 3 saída de comparação, CMP3.

  00 - Forçado baixo.

  01 - Ativo baixo.

  10 - Ativo alto.

  11 - Forçado alto.

- **Bits 3-2 - CMP2ACT1-0** - Ação a ser tomada no pino 2 saída de comparação, CMP2.

  00 - Forçado baixo.

  01 - Ativo baixo.

10 - Ativo alto.

11 - Forçado alto.

- **Bits 1-0 - CMP1ACT1-0** - Ação a ser tomada no pino 1 saída de comparação, CMP1.

00 - Forçado baixo.

01 - Ativo baixo.

10 - Ativo alto.

11 - Forçado alto.

15	14	13	12	11	10	9	8
SVRDIR	D2	D1	D0	CMP12ACT1	CMP12ACT0	CMP11ACT1	CMP11ACT0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7	6	5	4	3	2	1	0
CMP10ACT1	CMP10ACT0	CMP9ACT1	CMP9ACT0	CMP8ACT1	CMP8ACT0	CMP7ACT1	CMP7ACT0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Figura 7.16: Esquema com as posições dos *bits* no registrador ACTRB.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bit 15 - SVRDIR** - Direção de rotação do SVPWM.

0 - positiva (CCW).

1 - Negativa (CW).

- **Bits 14-12 - D2, D1 e D0** - Indicam o atual setor do vetor no SVPWM.

- **Bits 11-10 - CMP12ACT1-0** - Ação a ser tomada no pino 12 saída de comparação, CMP12.

00 - Forçado baixo.

01 - Ativo baixo.

10 - Ativo alto.

11 - Forçado alto.

- **Bits 9-8 - CMP11ACT1-0** - Ação a ser tomada no pino 11 saída de comparação, CMP11.

  00 - Forçado baixo.

  01 - Ativo baixo.

  10 - Ativo alto.

  11 - Forçado alto.

- **Bits 7-6 - CMP10ACT1-0** - Ação a ser tomada no pino 10 saída de comparação, CMP10.

  00 - Forçado baixo.

  01 - Ativo baixo.

  10 - Ativo alto.

  11 - Forçado alto.

- **Bits 5-4 - CMP9ACT1-0** - Ação a ser tomada no pino 9 saída de comparação, CMP9.

  00 - Forçado baixo.

  01 - Ativo baixo.

  10 - Ativo alto.

  11 - Forçado alto.

- **Bits 3-2 - CMP8ACT1-0** - Ação a ser tomada no pino 8 saída de comparação, CMP8.

  00 - Forçado baixo.

  01 - Ativo baixo.

  10 - Ativo alto.

  11 - Forçado alto.

- **Bits 1-0 - CMP7ACT1-0** - Ação a ser tomada no pino 7 saída de comparação, CMP7.

  00 - Forçado baixo.

  01 - Ativo baixo.

  10 - Ativo alto.

  11 - Forçado alto.

7.3.6 Interrupções e Reset das Unidades de Comparação

Existe um *flag* de interrupção mascáravel no EVnIFRA e no EVnIFRC, onde n = A ou B, para cada unidade de comparação. O *flag* de interrupção de uma unidade de comparação é setado um ciclo de *clock* após ocorrer uma igualdade na comparação, se uma operação de comparação estiver habilitada. Uma requisição de interrupção do periférico será gerada, se ela estiver habilitado.

Quando ocorre um *reset*, todos os *bits* dos registradores associados com as unidades de comparação são zerados e todos os pinos de saída de comparação são colocados em estado de alta impedância.

7.4 Circuitos PWM associados com as Unidades de Comparação

Os circuitos PWM associados com as unidades de comparação podem gerar até seis canais de saída PWM por gerenciador de eventos (EV), com banda morta e polaridade de saída programáveis. Na figura 7.17 está ilustrado o diagrama de blocos do circuito PWM do EVA. O diagrama de blocos do circuito PWM do EVB é idêntico ao do EVA, com as correspondentes mudanças nos registradores de configuração.

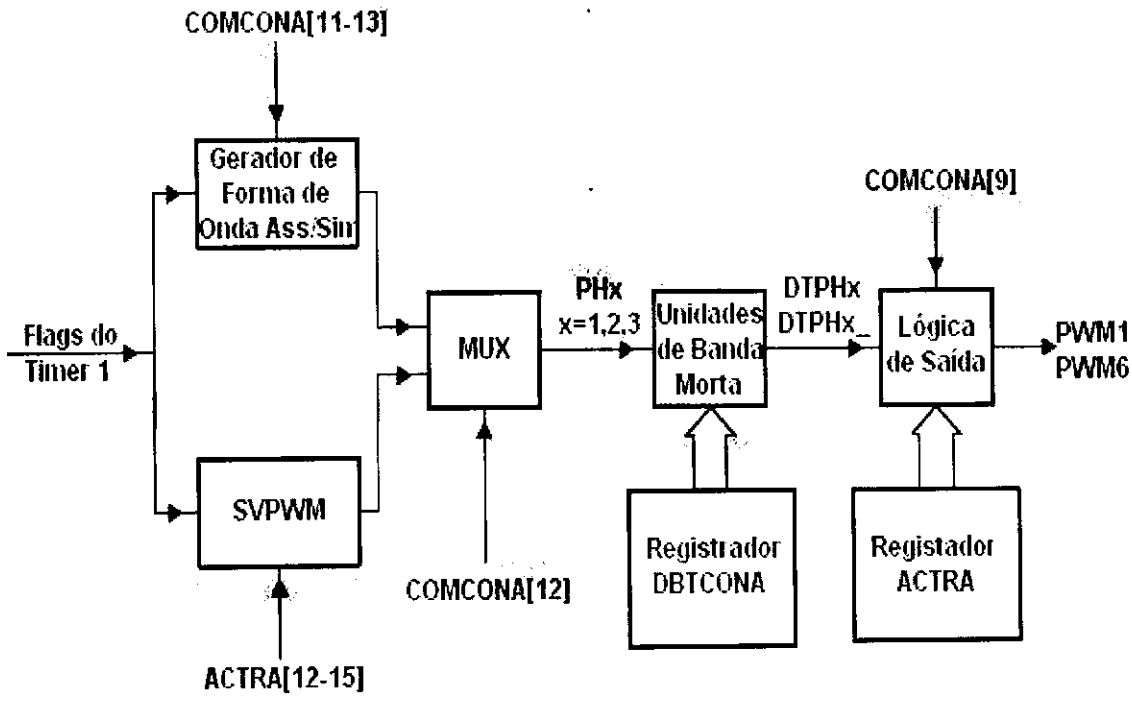


Figura 7.17: Diagrama de blocos do circuito PWM do EVA.

A geração do PWM com as unidades de comparação e circuitos PWM associados são controlados pelos seguintes registradores de controle: **T1CON**, **COMCONA**, **ACTRA** e **DBTCONA** (para o EVA). **T3CON**, **COMCONB**, **ACTRB** e **DBTCONB** (para o EVB).

7.4.1 Capacidade de Geração do PWM dos EVs

A capacidade de geração das formas de onda PWM de cada EV são as seguintes:

- Cinco saídas PWM independentes, três das quais são geradas pelas unidades de comparação e as outras duas pela comparação dos *timers*. Mais três saídas PWM adicionais, dependentes das três unidades de comparação;
- Banda morta programável para os pares de pinos de saída PWM associados com as unidades de comparação;
- Duração mínima da banda morta de um ciclo do *clock* do dispositivo;
- Resolução máxima do PWM é de 16 *bits*;
- Controle de mudança da frequência da portadora do PWM;
- Controle de mudança da largura do pulso do PWM;
- Interrupção de proteção do dispositivo de potência;
- Programação da geração das formas de onda PWM simétricas assimétricase SVPWM;
- Sobrecarga mínima da CPU, devido ao auto - recarregamento dos registradores de comparação e de período.

7.4.2 Programação da Unidade de Banda Morta

O EVA e o EVB possuem os registradores (**DBTCONA** e **DBTCONB**) que controlam o tempo de banda morta. Nas figuras 7.18 e 7.19 estão ilustrados, respectivamente, os esquemas com as posições dos *bits* nos registradores **DBTCONA** e **DBTCONB**.

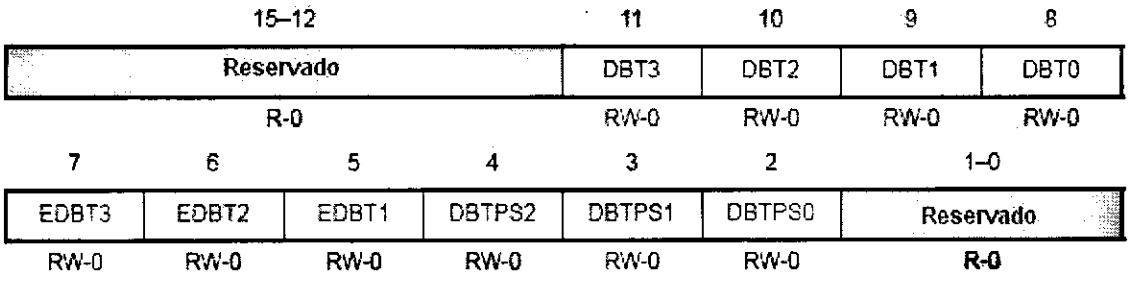


Figura 7.18: Esquema com as posições dos *bits* no registrador DBTCONA.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bits 11-8 - DBT3-DBT0** - Período da banda morta. Esses *bits* definem o valor do periodo (de até 4 *bits*) da banda morta nos três *timers*.

- **Bit7 - EDBT3** - habilita a banda morta no *timer 3*, para os pinos PWM5 e PWM6 da unidade de comparação 3.

  0 - Desabilita.

  1 - habilita.

- **Bit6 - EDBT2** - habilita a banda morta no *timer 2*, para os pinos PWM3 e PWM4 da unidade de comparação 2.

  0 - Desabilita.

  1 - habilita.

- **Bit5 - EDBT1** - habilita a banda morta no *timer 1*, para os pinos PWM1 e PWM2 da unidade de comparação 1.

  0 - Desabilita.

  1 - habilita.

- **Bits 4-2 - DBTPS2-DBTPS0** - Prescaler do tempo de banda morta.

  000 - x/1

  001 - x/2

  010 - x/4

  011 - x/8

  100 - x/16

  101 - x/32

  110 - x/32

  111 - x/32

x = Frequência do *Clock* da CPU (CPUCLK ou CLKOUT).

15–12	11	10	9	8
Reservado	DBT3	DBT2	DBT1	DBT0
R-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1–0
EDBT3	EDBT2	EDBT1	DBTPS2	DBTPS1	DBTPS0	Reservado
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	R-0

Figura 7.19: Esquema com as posições dos *bits* no registrador DBTCONB.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bits 11-8 - DBT3-DBT0** - Período da banda morta. Esses *bits* definem o valor do periodo (de até 4 bits) da banda morta nos três *timers*.

- **Bit7 - EDBT3** - habilita a banda morta no *timer 3*, para os pinos PWM11 e PWM12 da unidade de comparação 6.

  0 - Desabilita.

  1 - habilita.

- **Bit6 - EDBT2** - habilita a banda morta no *timer 2*, para os pinos PWM9 e PWM10 da unidade de comparação 5.

  0 - Desabilita.

  1 - habilita.

- **Bit5 - EDBT1** - habilita a banda morta no *timer 1*, para os pinos PWM7 e PWM8 da unidade de comparação 4.

  0 - Desabilita.

  1 - habilita.

- **Bits 4-2 - DBTPS2-DBTPS0** - *Prescaler* do tempo de banda morta.

  000 - x/1

  001 - x/2

  010 - x/4

  011 - x/8

  100 - x/16

  101 - x/32

  110 - x/32

  111 - x/32

x = Frequência do *Clock* da CPU (CPUCLK ou CLKOUT).

7.4.3 Entradas e Saídas da Unidade de Banda Morta

As entradas para a unidade de banda morta são PH1, PH2 e PH3 das formas de onda assimétrica/simétrica geradas pelas unidades de comparação 1, 2 e 3, respectivamente.

As saídas da unidade de banda morta são DTPH1, DTPH1_, DTPH2, DTPH2_ e DTPH3, DTPH3_, correspondendo a PH1, PH2 e PH3, respectivamente.

7.4.4 Geração da Banda Morta

Para cada sinal de entrada PHx, dois sinais DTPHx e DTPHx_ serão gerados. Quando a unidade de banda morta não for habilitado pelas unidades de comparação, os dois sinais serão exatamente idênticos. Quando a unidade de banda morta for habilitada pela unidade de comparação, a transição das bordas dos sinais serão separadas por um intervalo de tempo chamado de banda morta. Este intervalo de tempo é determinado pelos *bits* do registrador **DBTCONx**. Na figura 7.20 está ilustrado o diagrama de blocos da unidade de banda morta e os sinais nos pinos de entrada e saída.

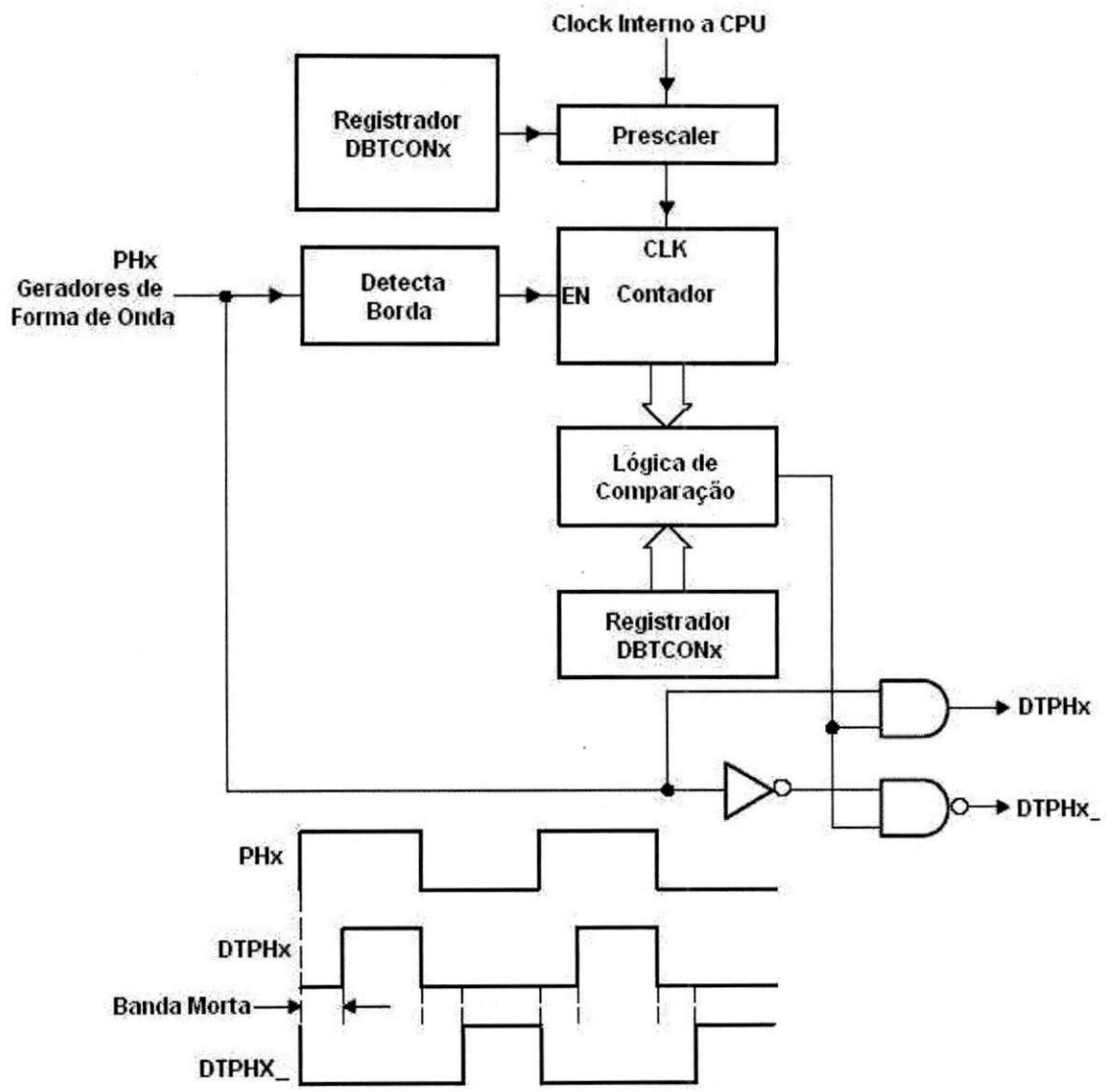


Figura 7.20: Diagrama de blocos da unidade de banda morta e os sinais nos pinos de entrada e saída.

7.4.5 Lógica de Saída

O circuito da lógica de saída determina a polaridade e/ou a ação a ser tomada quando ocorrer uma igualdade na comparação das saídas PWMx, para x=1-12. As saídas associadas com cada unidade de comparação podem ser especificadas como: ativo baixo, ativo alto, forçado baixo e forçado alto. A polaridade e/ou a ação das saídas PWM podem ser programadas pela configuração dos *bits* do registrador **ACTRn**. Os pinos de saída PWM podem todos serem colocados em estado de alta impedância por algum dos seguintes eventos:

- Zerando por *software* o *bit* 9 do registrador **COMCONx**;
- Por *hardware*, colocando PDPINTx em nível baixo, quando a interrupção PDPINTx estiver habilitada;
- Um *reset*.

7.5 Geração de Formas de Onda PWM com as Unidades de Comparaç e Circuitos PWM

Em um sistema de controle de um motor, os sinais PWM são usados para controlar o tempo em que uma chave permanecerá aberta ou fechada em dispositivos de potência que fornecem a corrente e a potência desejadas para as bobinas do motor.

7.5.1 Geração do Sinal PWM

Para gerar um sina PWM, um *timer* será utilizado para repetir um periodo de contagem que é igual ao perido do PWM. Um registrador de comparação será usado para os valores modulantes. O valor do registrador de comparação é constantemente comparado com o valor do contador do *timer*. Quando os valores são iguais, uma transição (de nível alto para baixo, ou de nível baixo para alto) acontecerá na associada saída. Quando ocorrer uma segunda igualdade, ou quando o periodo terminar, outra transição ocorrerá na associada saída. deste modo, um pulso de saída será gerado, e sua duração é proporcional ao valor do registrador de comparação. Este processo se repete a cada período com diferentes valores no registrador de comparação. Como um resultado, um sinal PWM será gerado na associada saída.

7.5.2 Banda Morta

Em muitas aplicações na eletrônica de potência, dois dispositivos de potência são colocados em série com um conversor de potência. de modo a evitar defeitos durante os disparos das chaves, os períodos dos sinais PWM não devem se sobrepôr. Para isso insere-se um tempo entre o desligamento de um transistor e o acionamento de outro transistor. Esse tempo é chamado de tempo de banda morta. Este atraso permite efetuar o desligamento de um transistor antes do acionamento do outro trnsistor. o tempo de atraso requisitado é especificado pelas características de desligamento e acionamento dos transistores de potência e as características da carga em uma aplicação específica.

7.5.3 Geração dos PWM com o EV

Cada uma das três unidades de comparação, junto com o *timer 1* (no caso do EVA), ou *timer 3* (no caso do EVB), a unidade de banda morta e a lógica de saída no módulo EV, podem ser usadas para gerar um par de saídas PWM com banda morta e polaridades programáveis em dois pinos do dispositivo. Existem seis pinos saída PWM associados com as três unidades de comparação de cada módulo EV. Estes seis pinos de saída podem ser usados para controlar motores de indução AC trifásicos ou motores DC.

A flexibilidade de controle do comportamento da saída dada pelo registrador de controle de ação da comparação (**ACTRA**) também torna fácil controlar motores com relutância chaveada e relutância síncrona. Cada unidade de comparação do *timer*, pode se desejado gerar uma saída PWM baseado no *timer*.

7.5.4 Registradores Usados para Geração de PWM

Todos os três tipos de geração de formas de onda PWM com as unidades de comparação e os circuitos associados, requerem a configuração dos mesmos registradores do módulo EV. O processo para geração do PWM incluem os seguintes passos:

- *Setup* e carregamento do registrador **ACTRx**;
- *Setup* e carregamento do **DBTCONx**, se a banda morta for utilizada;
- Inicialização do CMPRx;
- *Setup* e carregamento do **COMCONx**;

- *Setup* e carregamento do **T1CON** (para o EVA) ou **T3CON** (para o EVB) para iniciar a operação;
- Reescrever CMPRx com novos valores determinados.

7.5.5 Geração de Formas de Onda PWM Assimétricas

O sinal PWM assimétrico é caracterizado pelos pulsos modulados que não estão centrados com respeito ao período do PWM. Na figura 7.21 está ilustrado os sinais de um PWM assimétrico.

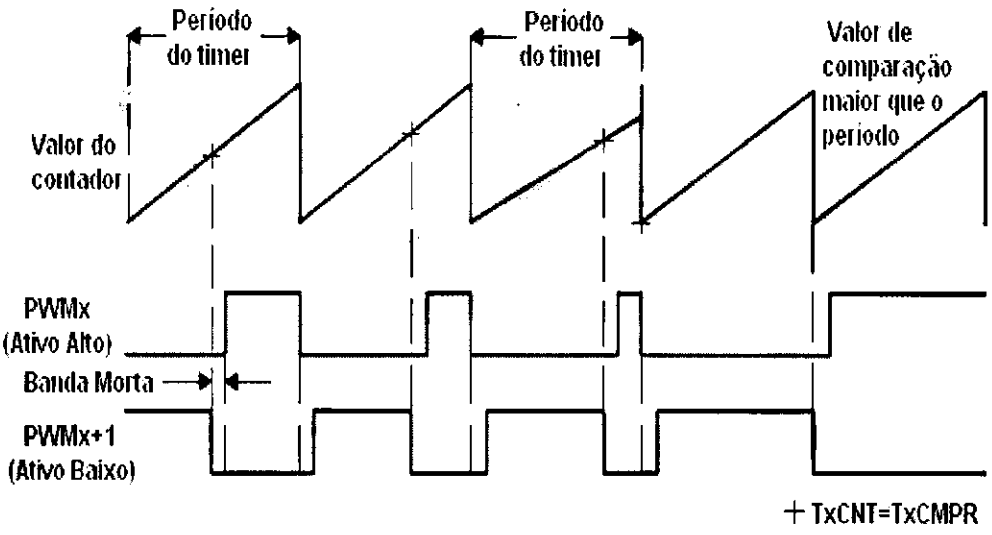


Figura 7.21: Sinais de um PWM assimétrico.

Para gerar um sinal PWM assimétrico, o *timer 1* deverá ser colocado no modo de contagem contínuo crescente e carregar o registrador de período com o valor correspondente do peíodo da portadora do PWM desejado. O registrador **COMCONx** é configurado para habilitar a operação de comparação, ajustar os pinos de saída selecionados para serem as saídas PWM, e habilitar as saídas. Se a banda morta for habilitada, o valor correspondente ao tempo de banda morta requisitado deverá ser escrito por software nos *bits* DBT(3-0) do registrador **DBTCONx**. Este é o período para os *timers* de 4 *bits* utilizados nas unidades de banda morta. Um valor de banda morta será usado para todos os canais de saída PWM.

Pela própria configuração do **ACTRx** por *software*, um sinal PWM normal poderá ser gerado em uma saída associada com uma unidade de comparação enquanto a outra estiver desligada ou ligada, no início, meio ou fim do período do PWM.

Após o *timer 1* ou *timer 3* ser iniciado, os registradores de comparação são sempre reescritos com novos valores de comparação para ajustar a largura do pulso (ciclo de trabalho) das saídas PWM que controlam o tempo de duração do desligamento e acionamento das chaves nos dispositivos de potência. Um novo valor poderá ser escrito nos registradores de comparaçãoem algum momento durante o período do PWM. Novos valores também podem ser escritos nos registradores de ação e de periodo em algum momento durante o período do PWM, para alterar o período do PWM ou forçar a mudança na definição da saída do PWM.

7.5.6 Geração de Formas de Onda PWM Simétricas

Um sinal PWM simétrico é caracterizado pelos pulsos modulados nos quais estão centrados com respeito a cada período do PWM. A vantagem de um PWM simétrico sobre um assimétrico, é que ele tem duas zonas inativas de mesma duração: no início e no fim de cada período do PWM. Essa simétria tem sido usada para causar menos harmônicas nas fases de um motor AC, quando a modulação senoidal é utilizada. Na figura 7.22 está ilustrado os sinais de um PWM simétrico.

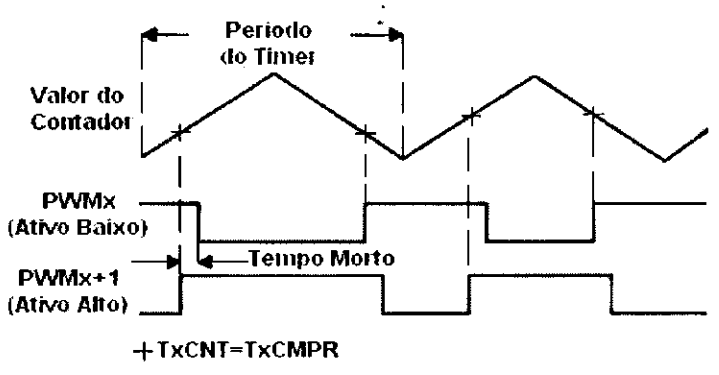


Figura 7.22: Sinais de um PWM simétrico.

A geração de uma forma de PWM simétrica com uma unidade de comparação é similar a geração de um PWM assimétrico. A única diferença é que o *timer 1* ou *timer 3*, dependendo de qual módulo EV está sendo usado, agora é colocado no modo de contagem contínuo crescente/decrescente.

Existem usualmente duas igualdades na comparação em um período do PWM em uma geração de uma forma de onda PWM simétrico. Uma durante a contagem crescente, antes da primeira igualdade na comparação, e outra na contagem decrescente, após a primeira igualdade. Um novo valor de comparação será recarregado quando o valor do contador do *timer* for igual ao dobro do valor do registrador de período.

7.6 Modulação Vetorial - SVPWM

SVPWM é um esquema especial de chaveamento dos seis transistores de potência de um conversor de potência trifásico. Esse esquema de modulação gera gera uma distorção harmônica mínima nas correntes das bobinas do motor AC trifásico. Ele também fornece um uso mais eficiente da fonte de tensão em comparação com o método da modulação senoidal.

7.6.1 Inversor de Potência Trifásico

A estrutura de um típico inversor de potência está ilustrado na figura 7.23. Onde Va, Vb e Vc são as tensões aplicadas nas bobinas do motor. Os seis transistores de potência são controlados pelas entradas DTPHx e DTPHx_ (x = A, B e C). Quando um transistor de cima for fechado (DTPHx = 1), o transistor de baixo será aberto (DTPHx_ = 0). Deste modo, os estados dos transistores de cima (Q1, Q2 e Q3) são suficientes para avaliar a tensão aplicada no motor.

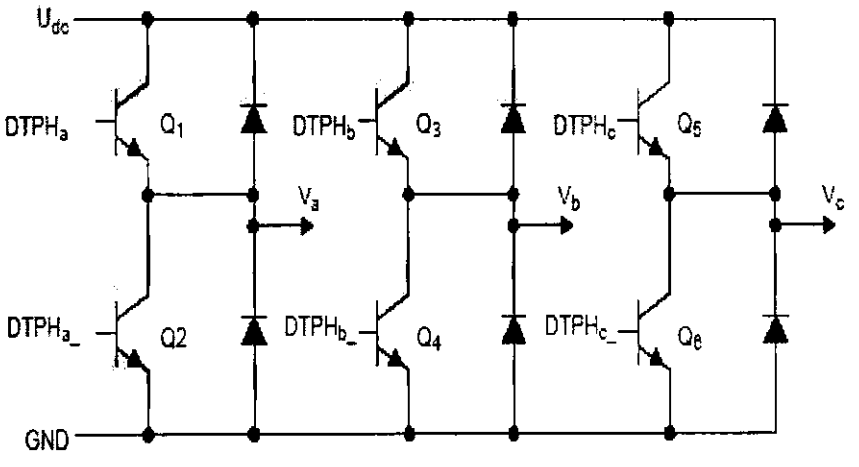


Figura 7.23: Diagrama esquemático de um inversor de potência trifásico.

7.6.2 Chaveamento Padrão de um Inversor de Potência e os Vetores Básicos

Quando um transistor de cima de uma perna está fechado, a tensão Vx aplicada na perna para a correspondente bobina do motor será igual a tensão aplicada Udc. Quando ele for aberto, a tensão aplicada será zero. O chaveamento dos transistores de cima (DTPHx) tem oito possíveis combinações. As oito combinações e os respectivos valores de tensão de linha e de fase em função de Udc são indicados na tabela 7.4.

Tabela 7.3: As oito combinações e os respectivos valores de tensão de linha e de fase em função de Udc.

a	b	c	Va0(Udc)	Vb0(Udc)	Vc0(Udc)	Vab(Udc)	Vbc(Udc)	Vca(Udc)
0	0	0	0	0	0	0	0	0
0	0	1	-1/3	-1/3	2/3	0	-1	1
0	1	0	-1/3	2/3	-1/3	-1	1	0
0	1	1	-2/3	1/3	1/3	-1	0	1
1	0	0	2/3	-1/3	-1/3	1	0	-1
1	0	1	1/3	-2/3	1/3	1	-1	0
1	1	0	1/3	1/3	-2/3	0	1	-1
1	1	1	0	0	0	0	0	0

Mapeando as tensões de fase correspondentes as oito combinações no plano d-q através de uma transformação d-q (no qual é equivalente a uma projeção ortogonal dos três vetores (a b c) para cima dos dois planos dimensionais perpendicular ao vetor (1, 1, 1), plano d-q), resulta em seis vetors não-nulos e dois vetores nulos. Os vetores não-nulos formam um eixo hexagonal. O ângulo entre dois vetores adjacentes é de 60°. Os dois vetores nulos estão na origem. Esses oito vetores são chamados pré-definidos, e são denotados por U0, U60, U120, U180, U240, U300, O000 E O111. A mesma transformação pode ser aplicada ao vetor de tensão Uout a ser aplicado em um motor. Os eixos d e q de um plano d-q correspodem aos eixos geométricos horizontal e vertical, respectivamente, do estator de uma máquina AC. Na figura 7.24 está ilustrado um diagrama com os vetores pré-definidos.

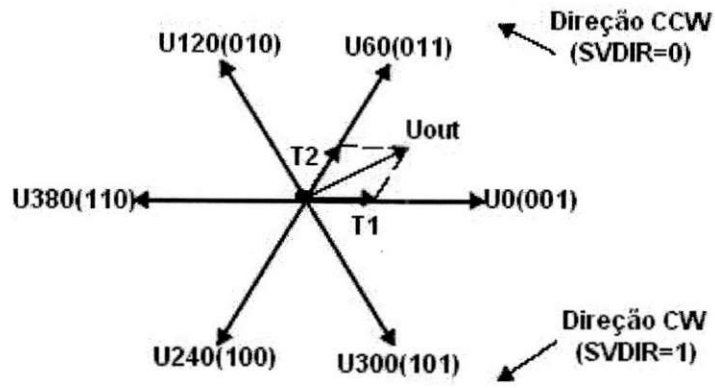


Figura 7.24: Diagrama com os vetores pré-definidos e suas respectivas denotações.

O objetivo do método SVPWM é aproximar o vetor de tensão do motor Uout por uma combinação desses oito possíveis chaveamentos dos seis transistores de potência.

As representações em binário dos dois vetores pré-definidos adjacentes são diferentes em apenas um *bit*, isto é, apenas um dos transistores de cima chaveam quando ocorrer o chaveamento padrão das chaves de Ux à Ux+60 ou de Ux+60 à Ux. Os vetores nulos (O000 e O111) não aplicam tensão ao motor.

O projetado vetor de tensão do motor Uout, em um dado tempo, está localizado em uma das seis seções existentes no eixo hexagonal. Deste modo, para algum período do PWM, ele pode ser aproximado pela soma vetorial das duas componentes do vetor reclinado nos dois vetores adjacentes:

$$Uout = T1 * Ux + T2 * Ux + 60 + T0 * (O000ouO111)$$

Onde T0 é dado por Tp - T1 - T2 e Tp é o período da portadora do PWM. O terceiro termo a direita da equação não afeta a soma do vetor Uout.

7.6.3 Geração do SVPWM com EV

O módulo EV possui no *hardware* uma simplificada forma de geração do SVPWM. Para gerar saídas PWM utilizando a modulação vetorial, o *software* do usuário deverá seguir os seguintes passos:

1. Configurar o registrador **ACTRx** para definir a polaridade dos pinos de saída de comparação;
2. Configurar o registrador **COMCONx** para habilitar a operação de comparação e o modo SVPWM, e ajustar a condição de recarga do registrador **CMPRx** para ser um *underflow*;
3. Colocar o *timer 1* (ou *timer 3*) no modo de contagem contínuo crescente/decrescente para inicia a operação.

O *software* do usuário precisará determinar a tensão Uouta ser aplicada nas fases do motor no plano dimensional d-q, decompondo Uout e desempenhando os seguintes passos para cada período do PWM:

1. Determinar os dois vetores adjacentes, Ux e Ux+60;
2. Determinar os parâmetros T2, T1 e T0;
3. Escrever o tipo de chaveamento correspondente para Ux nos *bits* 14-12 do registrador **ACTRx** e '1' no *bit* 15 do mesmo registrador. Uma outra opção é escrever o tipo de chaveamento correspondente para Ux+60 nos *bits* 14-12 do registrador **ACTRx** e '0' no *bit* 15 do mesmo registrador;
4. Carregar o registrador de comparação **CMPR1** (T1/2) e carregar **CMPR2** (T1/2 + T2/2).

O hardware do SVPWM fará o seguinte no módulo EV para completar um período do SVPWM:

1. No início de cada período, ajusta as saídas PWM para o novo padrão Ux definido pelos *bits* 14-12 do registrador **ACTRx**;
2. Na primeira igualdade da comparação durante a contagem crescente (em T1/2), chavea as saídas PWM para o padrão Ux se o *bit* 15 do registrador **ACTRx** for '1', ou para o padrão Ux+60 se for '0';
3. Na segunda igualdade da comparação durante a contagem crescente (em T1/2 + T2/2), chavea as saídas PWM para o padrão (000) ou (111), no qual difere do segundo padrão por um *bit*;

4. Na primeira igualdade da comparação durante a contagem decrescente (em T1/2 + T2/2), chavea as saídas PWM de volta para o segundo padrão de saída;

5. Na segunda igualdade da comparação durante a contagem decrescente (em T1/2), chavea as saídas PWM de volta para o primeiro padrão de saída;

A formas de onda geradas são simétricas com respeito a metade de cada período do PWM. Na figura 7.25 está ilustrado um esquema com os sinais SVPWM.

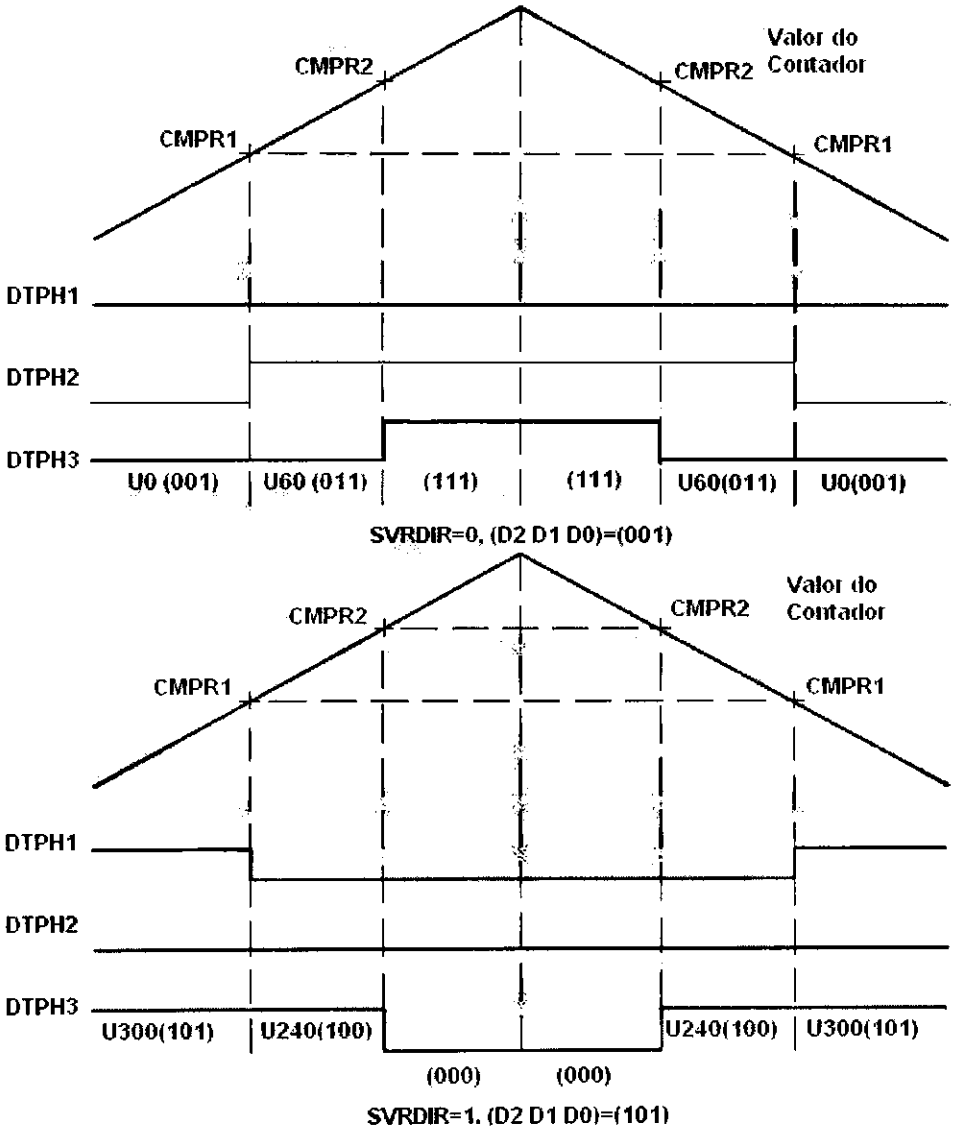


Figura 7.25: Esquema com os sinais SVPWM.

7.6.4 Registradores de Comparação Utilizados no SVPWM

Apenas dois dos três registradores de comparação são utilizados na geração das saídas SVPWM. O terceiro registrador de comparação, é ainda constantemente comparado com o valor do *timer 1*. Quando uma igualdade na comparação ocorrer, o correspondente flag de interrupção de comparação será setado, e uma requisição de interrupção do periférico será gerada se estiver habilitada. Entretanto, o registrador de comparação que não está sendo utilizado na geração da saída SVPWM poderá ser utilizado para marcar eventos que possam acontecer em uma aplicação específica. As transições nas saídas de comparação são atrasadas por um ciclo do *clock* no modo SVPWM.

7.7 Unidades de Captura

As unidades de captura registram as transições nos pinos de entrada de captura. Existem seis unidades de captura, três em cada módulo EV. As unidades de captura 1, 2 e 3 estão associados com o EVA e as unidades de captura 4, 5 e 6 estão associados com o EVB. Cada unidade de captura possui um pino de entrada de captura.

Cada unidade de captura do EVA pode escolher o *timer 1* ou o *timer 2* como sua base de tempo. Entretanto, CAP1 e CAP2 não podem escolher um *timer* diferente como suas base de tempo. As unidade de captura do EVB devem escolher o *timer 3* ou o *timer 4* como suas bases de tempo. Entretanto, CAP3 e CAP4 não podem escolher um *timer* diferente como suas bases de tempo.

O valor do *timer* é capturado e armazenado na correspondente pilha quando uma transição especificada for detectada em um pino de entrada de captura (CAPx). Nas figuras 7.26 e 7.27 estão ilustradas, respectivamente, o diagrama de blocos de uma unidade de captura do EVA e de uma unidade de captura do EVB.

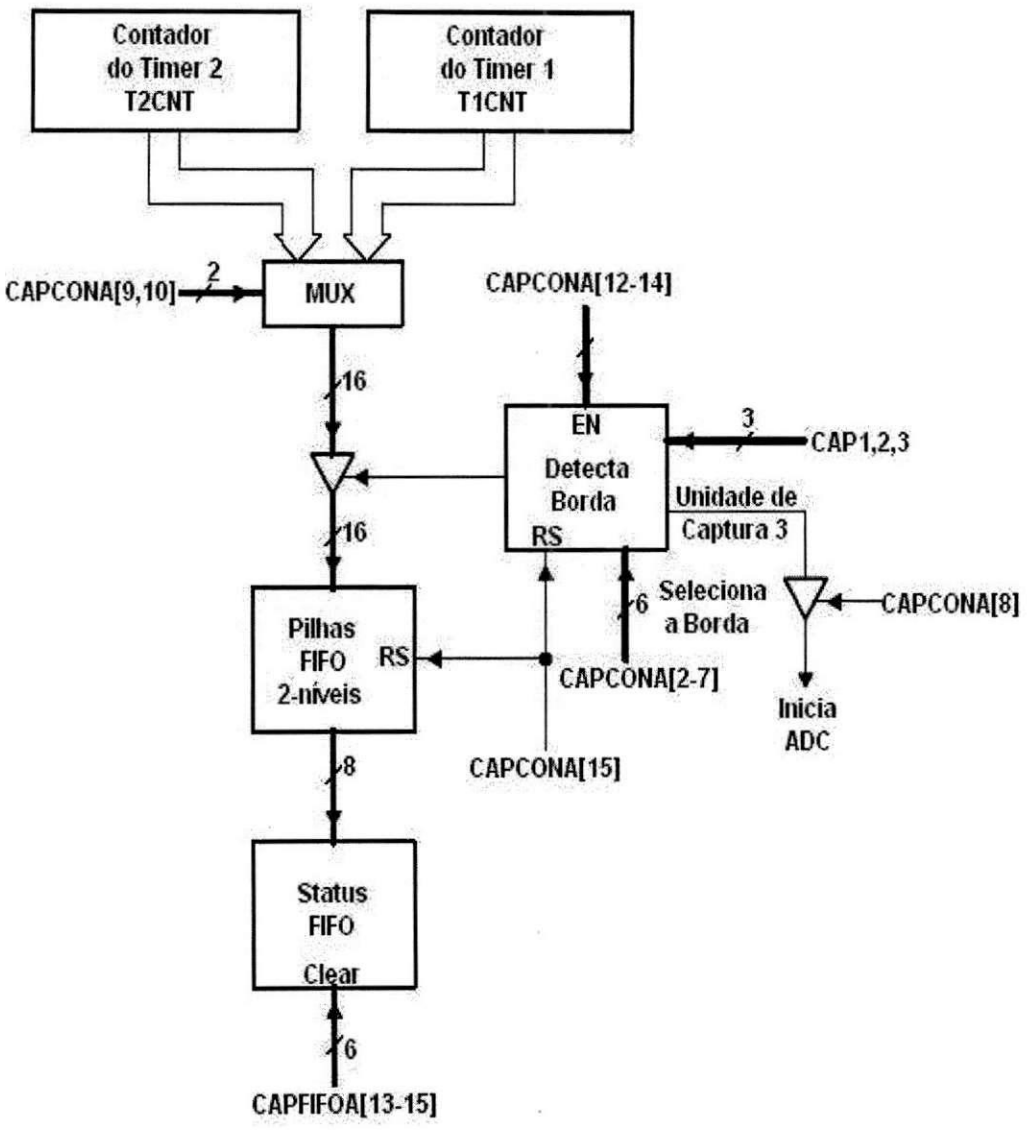


Figura 7.26: Diagrama de blocos de uma unidade de captura do EVA.

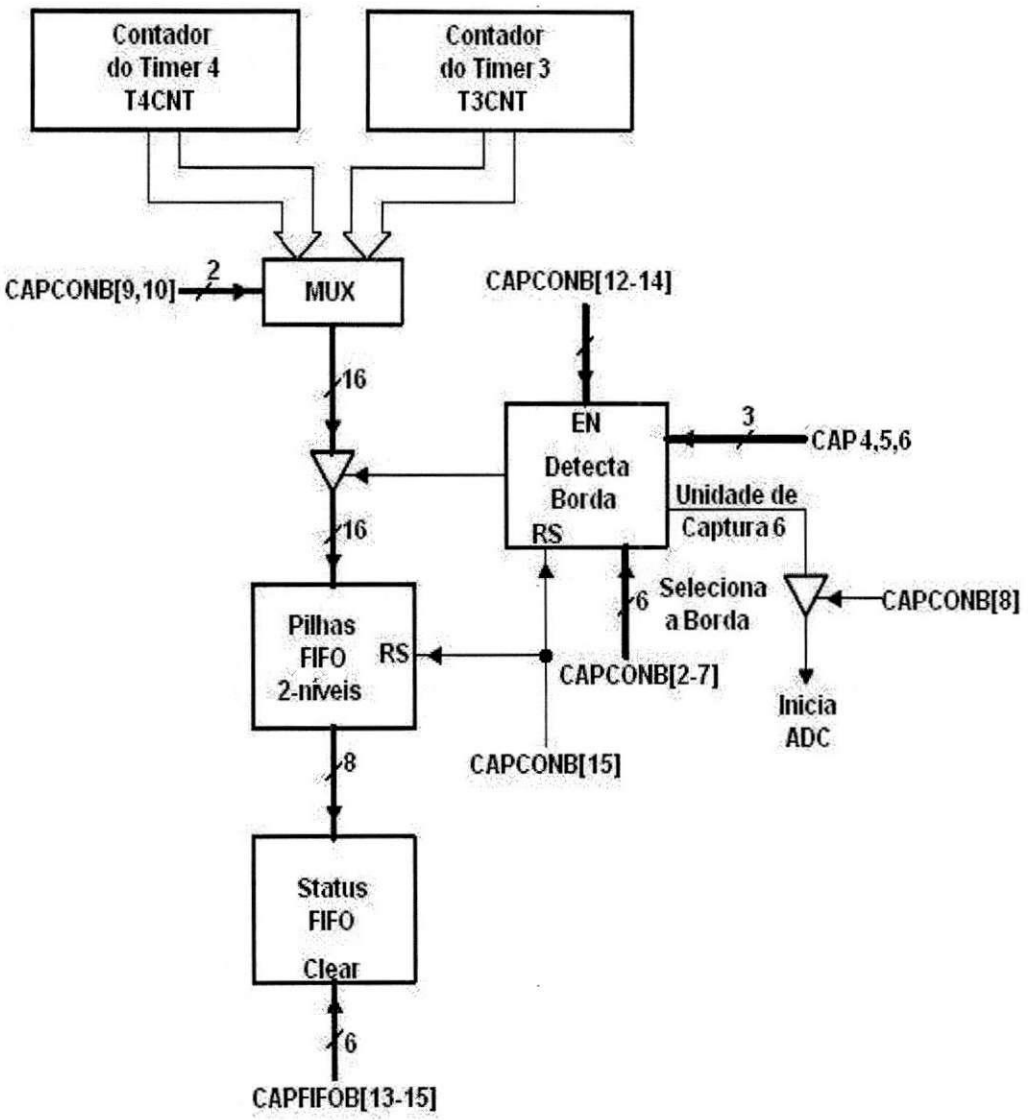


Figura 7.27: Diagrama de blocos de uma unidade de captura do EVB.

7.7.1 Características das Unidades de Captura

Cada unidade de captura possui as seguintes características:

- Um registrador de controle da captura (**CAPCONA** do EVA e **CAPCONB** do EVB);
- Um registrador de *status* da captura FIFO (**CAPFIFOA** e **CAPFIFOB**);
- Seleção do *timer 1* ou o *timer 2* para o EVA e *timer 3* ou *timer 4* para o EVB como base de tempo;
- Três pilhas FIFO, uma para cada unidade de captura;
- Seis pinos de entrada de captura *Shimitt-Triggered*, CAP1 à CAP6, um pino de entrada para cada unidade de captura. Todas as entradas são sincronizadas com o *clock* do dispositivo, de modo que para uma transição ser capturada, a entrada deverá permanecer nesse nível por duas bordas de subida do *clock* do dispositivo. Os pinos de entrada CAP1 e CAP2 (CAP4 e CAP5 no EVB) podem também serem usadas como entradas QEP do circuito QEP;
- O programador especifica a detecção da transição (borda de subida, borda dedescida, ou ambas as bordas);
- Seis *flags* de interrupções mascaráveis, um para cada unidade de captura.

7.7.2 Operação das Unidades de Captura

Após uma unidade de captura ser habilitada, uma transição específica em um associado pino de entrada fará com que o valor do contador do *timer* selecionado seja carregado na correspondente plha FIFO. No mesmo tempo, se existir um ou mais valores de captura armazenados na pilha FIFO (os *bits* CAPxFIFO são diferentes de zero), o correspondente *flag* de interrupção será setado. Se a interrupção estiver habilitada, uma requisição de interrupção do periférico será gerada. Os correspondentes *bits* de *status* no registrador **CAPFIFOx** serão ajustados para refletir o novo *status* da pilha FIFO toda vez que um novo valor do contador for capturado em uma pilha FIFO. Todos os registradores de uma unidade de captura são zerados escrevendo-se um zero ou por um *reset*.

7.7.3 Seleção da Base de Tempo de uma Unidade de Captura

Para o EVA, a unidade de captura 3 possui um *bit* de seleção da base de tempo separado das unidades de captura 1 e 2. Isto permite que os dois *timers* sejam usados no mesmo momento, um para as unidades de captura 1 e 2 e o outro para a unidade de captura 3. No EVB, a unidade de captura 6 possui um *bit* de seleção da base de tempo separado das unidades de captura 4 e 5.

Uma operação de captura não afeta a operação de algum *timer* ou das operações de comparação/PWM associadas com algum *timer*.

7.7.4 Setup de uma Unidade de Captura

Para uma unidade de captura funcionar corretamente, o setup dos seguintes registradores deverão ser realizados:

1. Inicializar o registrador **CAPFIFOx** e zerar os *bits* de *status* apropriados;
2. Setar o *timer* selecionado em um dos modos de operação;
3. Setar o associado registrador de comparação do *timer* ou o registrador de período do *timer*;
4. Setar o registrador **CAPCONA** ou **CAPCONB**.

7.7.5 Registradores das Unidades de Captura

As operações das unidades de captura são controladas por quatro registradores, **CAPCONA/B** e **CAPFIFOA/B**. Os registradores **TxCON** (x=1,2,3 e 4) são também utilizados para controlar a operação das unidades de captura, desde que a base de tempo para os circuitos de captura podem ser fornecidos opr algum desses *timers*. Adicionalmente, os registradores **CAPCONA/B** são também utilizados para controlar a operação dos circuitos QEP. Nas figuras 7.28 e 7.29 estão ilustrados, respectivamente, os esquema com as posições dos *bits* nos registradores **CAPCONA** e **CAPCONB**.

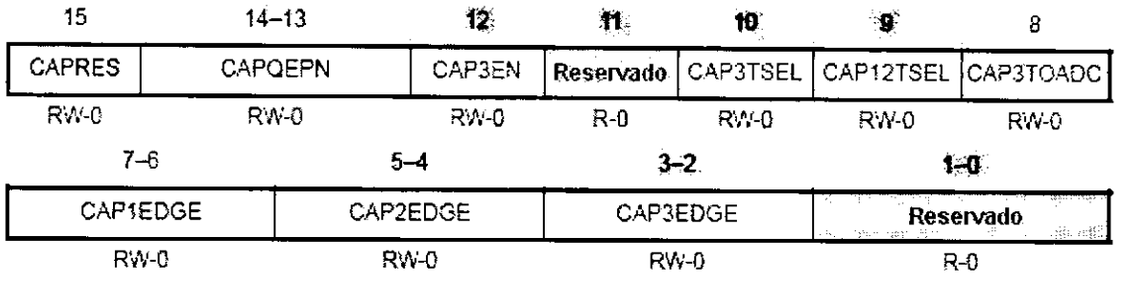


Figura 7.28: Esquema com as posições dos *bits* no registrador CAPCONA.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bit 15 - CAPRES** - Reseta a captura.

  0 - Zera todos os regitradores das unidades de captura e do circuito QEP.

  1 - Sem ação.

- **Bits 14-13 - CAPQEPN** - Controla as unidade de captura 1 e 2.

  00 - Desabilita as unidades de captura 1 e 2. A pilha FIFO retém o seus conteúdos.

  01 - Habilita unidades de captura 1 e 2.

  10 - Reservado.

  11 - Reservado.

- **Bit 12 - CAP3EN** - Controla a unidade de captura 3.

  0 - Desabilita a unidade de captura 3. A pilha FIFO retém o seu conteúdo.

  1 - Habilita unidade de captura 3.

- **Bit 10 - CAP3TSEL** - Seleciona o *timer* para a unidade de captura 3.

  0 - Seleciona o *timer 2*.

  1 - Seleciona o *timer 1*.

- **Bit 9 - CAP12TSEL** - Seleciona o *timer* para as unidades de captura 1 e 2.

  0 - Seleciona o *timer 2*.

  1 - Seleciona o *timer 1*.

- **Bit 8 - CAP3TOADC** - Um evento na unidade de captura 3 inicia o ADC.

  0 - Sem ação.

  1 - Inicia o ADC quando o *flag* CAP3INT for setado.

- **Bits 7-6 - CAP1EDGE** - Controle de detecção da borda para a unidade de captura 1.

  00 - Sem detecção.

  01 - Detecta borda de subida.

  10 - Detecta borda de descida.

  11 - Detecta ambas as bordas.

- **Bits 5-4 - CAP2EDGE** - Controle de detecção da borda para a unidade de captura 2.

  00 - Sem detecção.

  01 - Detecta borda de subida.

  10 - Detecta borda de descida.

  11 - Detecta ambas as bordas.

- **Bits 3-2 - CAP3EDGE** - Controle de detecção da borda para a unidade de captura 3.

  00 - Sem detecção.

  01 - Detecta borda de subida.

  10 - Detecta borda de descida.

  11 - Detecta ambas as bordas.

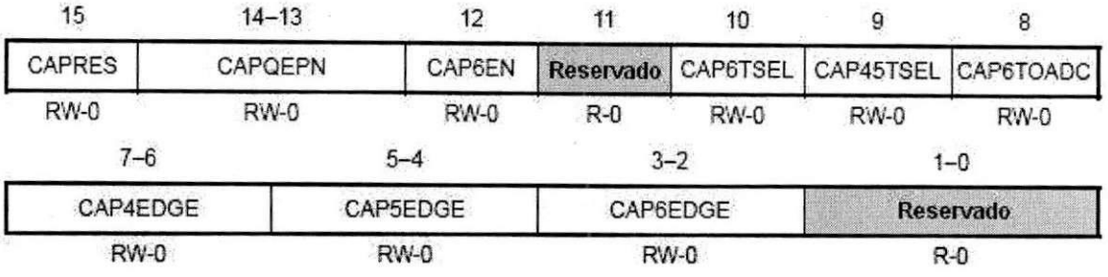


Figura 7.29: Esquema com as posições dos *bits* no registrador CAPCONB.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bit 15 - CAPRES** - Reseta a captura.

  0 - Zera todos os regitradores das unidades de captura e do circuito QEP.

  1 - Sem ação.

- **Bits 14-13 - CAPQEPN** - Controla as unidade de captura 4 e 5.

  00 - Desabilita as unidades de captura 4 e 4. A pilha FIFO retém o seus conteúdos.

  01 - Habilita unidades de captura 4 e 5.

  10 - Reservado.

  11 - Reservado.

- **Bit 12 - CAP6EN** - Controla a unidade de captura 6.

  0 - Desabilita a unidade de captura 6. A pilha FIFO retém o seu conteúdo.

  1 - Habilita unidade de captura 6.

- **Bit 10 - CAP6TSEL** - Seleciona o *timer* para a unidade de captura 6.

  0 - Seleciona o *timer 4*.

  1 - Seleciona o *timer 3*.

- **Bit 9 - CAP45TSEL** - Seleciona o *timer* para as unidades de captura 4 e 5.

  0 - Seleciona o *timer 4*.

  1 - Seleciona o *timer 3*.

- **Bit 8 - CAP6TOADC** - Um evento na unidade de captura 6 inicia o ADC.

  0 - Sem ação.

  1 - Inicia o ADC quando o *flag* CAP6INT for setado.

- **Bits 7-6 - CAP4EDGE** - Controle de detecção da borda para a unidade de captura 4.

  00 - Sem detecção.

  01 - Detecta borda de subida.

  10 - Detecta borda de descida.

  11 - Detecta ambas as bordas.

- **Bits 5-4 - CAP5EDGE** - Controle de detecção da borda para a unidade de captura 5.

  00 - Sem detecção.

  01 - Detecta borda de subida.

  10 - Detecta borda de descida.

  11 - Detecta ambas as bordas.

- **Bits 3-2 - CAP6EDGE** - Controle de detecção da borda para a unidade de captura 6.

  00 - Sem detecção.

  01 - Detecta borda de subida.

  10 - Detecta borda de descida.

  11 - Detecta ambas as bordas.

Os registradores CAPFIFOA/B contém os bits de status de cada uma da três pilhas FIFO da unidades de captura. Nas figuras 7.30 e 7.31 estão ilustrados, respectivamente, os esquemas com as posições dos *bits* nos registradores **CAPFIFOAe CAPFIFOB**.

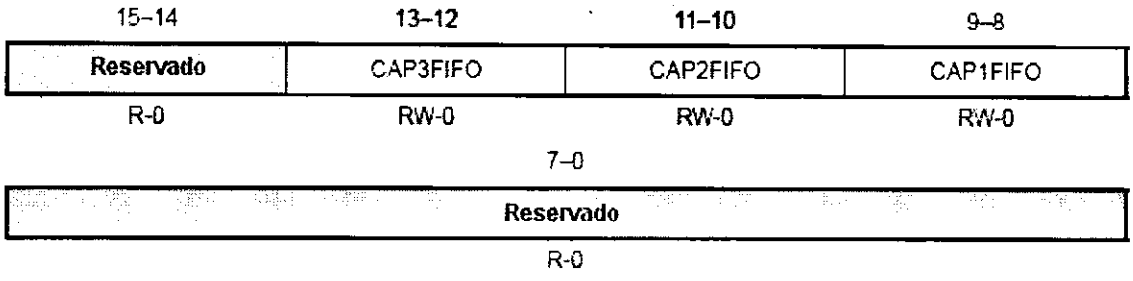


Figura 7.30: Esquema com as posições dos *bits* no registrador CAPFIFOA.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset.*

- **Bits 13-12 - CAP3FIFO** - *Status* do CAP3FIFO.

  00 - Vazio.

  01 - Possui uma entrada.

  10 - Possui duas entradas.

  11 - Possui duas entradas e uma outra capturada. A primeira entrada será perdida.

- **Bits 11-10 - CAP2FIFO** - *Status* do CAP2FIFO.

  00 - Vazio.

  01 - Possui uma entrada.

  10 - Possui duas entradas.

  11 - Possui duas entradas e uma outra capturada. A primeira entrada será perdida.

- **Bits 9-8 - CAP1FIFO** - *Status* do CAP1FIFO.

  00 - Vazio.

  01 - Possui uma entrada.

  10 - Possui duas entradas.

  11 - Possui duas entradas e uma outra capturada. A primeira entrada será perdida.

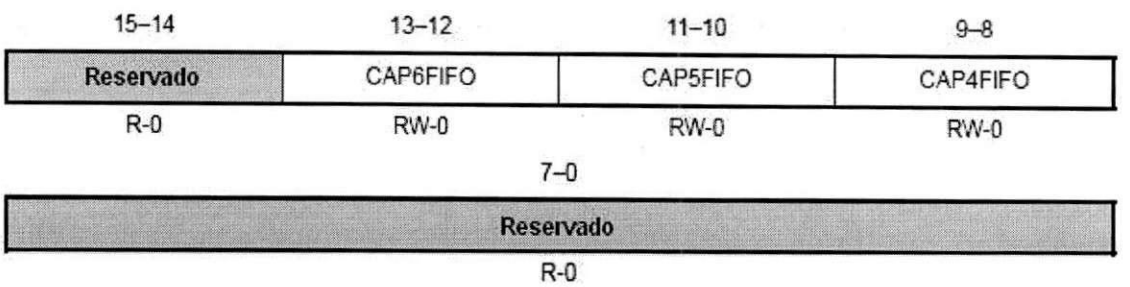


Figura 7.31: Esquema com as posições dos *bits* no registrador CAPFIFOB.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bits 13-12 - CAP6FIFO** - *Status* do CAP6FIFO.

  00 - Vazio.

  01 - Possui uma entrada.

  10 - Possui duas entradas.

  11 - Possui duas entradas e uma outra capturada. A primeira entrada será perdida.

- **Bits 11-10 - CAP5FIFO** - *Status* do CAP5FIFO.

  00 - Vazio.

  01 - Possui uma entrada.

  10 - Possui duas entradas.

  11 - Possui duas entradas e uma outra capturada. A primeira entrada será perdida.

- **Bits 9-8 - CAP4FIFO** - *Status* do CAP4FIFO.

  00 - Vazio.

  01 - Possui uma entrada.

  10 - Possui duas entradas.

  11 - Possui duas entradas e uma outra capturada. A primeira entrada será perdida.

7.7.6 Pilhas FIFO das Unidades de Captura

Cada unidade de captura possui um pilha FIFO. O topo da pilha consiste do CAP1FIFO, CAP2FIFO e CAP3FIFO (para o EVA) ou CAP4FIFO, CAP5F1FO e CAP6FIFO (para o EVB). O fundo da pilha consiste do CAP1FBOT, CAP2FBOT e CAP3FBOT (para o EVA) ou CAP4FBOT, CAP5FBOT e CAP6FBOT(para o EVB). O registrador do nível do topo de alguma das pilhas é um registrador somente de leitura. Ele sempre conterá o valor mais antigo do contador capturado pela correspondente unidade de captura. Portanto, uma leitura a pilha FIFO de uma unidade de captura sempre retornará o valor mais antigo do contador armazenado na pilha. Quando o valor mais antigo da pilha for lido, o valor mais recente do contador que está no fundo da pilha será deslocado para o topo da pilha.

Se desejado, o registrador do fundo da pilha FIFO pode ser lido. Essa leitura causará uma mudança nos *bits* de *status* FIFO para '01' (Possui uma entrada) se eles estiverem previamente em '10 ou '11'. Se os *bits* de *status* FIFO forem previamente '01'quando o registrador do fundo da pilha for lido, eles serão alterados para '00' (vazio).

Na primeira captura, o valor do contador do *timer* selecionado (capturado por uma unidade de captura quando uma transição específica acontecer em um pino de entrada) será escrito no topo da pilha FIFO, se ela estiver vazia. No mesmo instante, o correspondente *bit* de status será setado em '01'. Os *bits* de *status* serão resetados se uma leitura for feita na pilha FIFO antes de outra captura ser feita.

Se outra captura for realizada antes do primeiro valor capturado ser lido, o novo valor capturado será armazzenado no fundo do registrador. Enquanto isso, os correspondentes *bits* de *status* serão setados em '10'. Quando a pilha FIFO for lida antes de outra captura acontecer, o valor mais antigo do contador, que está no topo da pilha será lido fora da pilha, e o novo valor do contador que está no fundo do registrador, será colocado no topo do registrador, e os correspondentes *bits* de *status* serão setados para '01'. O apropriado *flag* de interrupção da captura será setado pela segunda captura. Uma requisição de interrupção do periférico será gerada se a interrupção estiver habilitada.

Se uma terceira captura acontecer qundo já existem dois valores do contador capturados na pilha FIFO, o valor mais antigo, que está no topo da pilha será perdido, e o valor do contador que está no fundo do registrador será colocado no topo do registrador, e a novo valor capturado será armazenado no fundo do registrador. Os *bits* de status serão setados em '11' para indicar que um ou mais valores antigos do contador foram perdidos. O apropriado *flag* de interrupção da captura também será setado pela terceira captura. Uma requisição de interrupção do periférico será gerada se a interrupção estiver habilitada.

7.7.7 Interrupção de Captura

Quando uma captura for feita por uma unidade de captura e já existir um valor válido no FIFO (indicado pelos *bits* CAPxFIFO, que serão diferentes de zero), o correspondente *flag* de interrupção será setado, e uma requisição de interrupção do periférico será gerada se estiver habilitada. Deste modo, um par de valores do contador capturados podem ser lidos por uma rotina de serviço de interrupção, se a interrupção for usada.

7.8 Circuito QEP (Quadrature Encoder Pulse)

Cada módulo EV possui um circuito QEP. O circuito QEP, quando habilitado, decodifica e conta os pulso de entrada codificados dos pinos CAP1/QEP1 e CAP2/QEP2 (para o EVA) ou CAP4/QEP3 e CAP5/QEP4 (para o EVB). O circuito QEP pode ser usado para interfacer com um codificador óptico para dar informações de posição e velocidade de uma rotação de uma máquina. Quando o circuito QEP está habilitado, a função de captura dos pinos CAP1/CAP2 e CAP4/CAP5 serão desabilitados.

7.8.1 Base de Tempo dos Circuitos QEP

A base de tempo para os circuitos QEP é fornecida pelo *timer 2* (EVA) ou pelo *timer 4*(EVB). O *timer* deverá ser colocado no modo de contagem direcional crescente/decrescente com o circuito QEP como fonte do *clock*. Nas figuras 7.32 e 7.33 estão ilustrados, respectivamente, o diagrama de blocos do circuito QEP do EVA e do EVB.

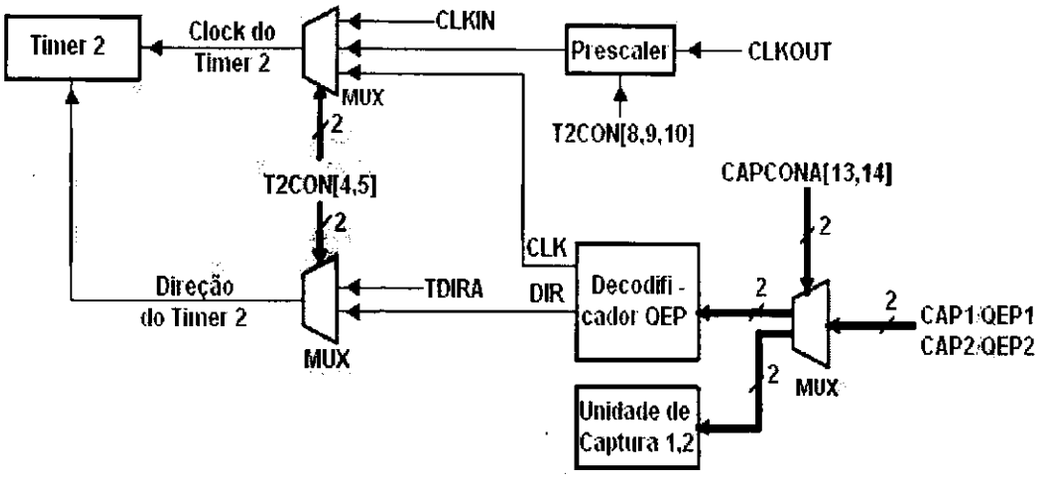


Figura 7.32: Diagrama de blocos do circuito QEP do EVA.

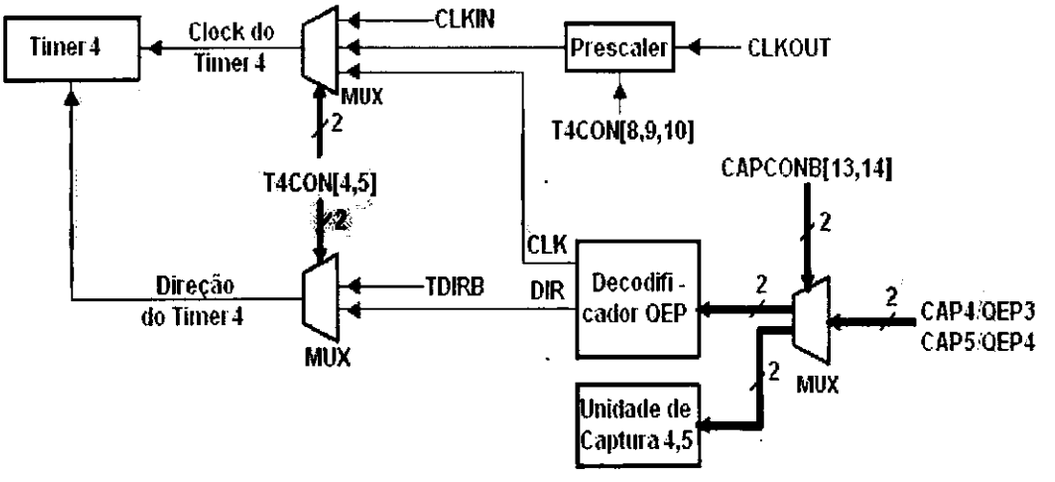


Figura 7.33: Diagrama de blocos do circuito QEP do EVB.

7.8.2 Decodificação

Os pulsos codificados em quadratura são duas sequências de pulsos com uma frequência variável e um deslocamento de fase fixado em 90°. Quando gerados por um codificador óptico em um motor, a direção de rotação do motor pode ser determinado descobrindo qual das duas sequências é a sequência principal. A posição angular e a velocidade podem ser determinadas pela contagem dos pulsos e pela frequência do pulso.

A lógica de detecção de direção do circuito QEP no módulo EV determina qual das sequências é a sequência principal. Ele então gera um sinal de direção como a entrada de direção do *timer 2* (ou *timer 4*). O *timer* contará de forma crescente se a entrada CAP1/QEP1 (CAP4/QEP3 do EVB) for a sequência principal, e conta de forma decrescente se CAP2/QEP2 (CAP5/QEP4 do EVB) for a sequência principal.

Ambas as bordas dos pulsos das entradas codificadas em quadratura são contadas pelo circuito QEP. Entretanto, a frequência do *clock* gerado pela lógica QEP para o *timer 2* (ou *timer 4*) é quatro vezes maior que cada sequência de entrada. Este *clock* em quadratura é conectado ao *clock* de entrada do *timer 2* (ou *timer 4*). NA figura 7.34 está ilustrado um esquema com os pulsos codificados em quadratura, o *clock* derivado e a direção de contagem.

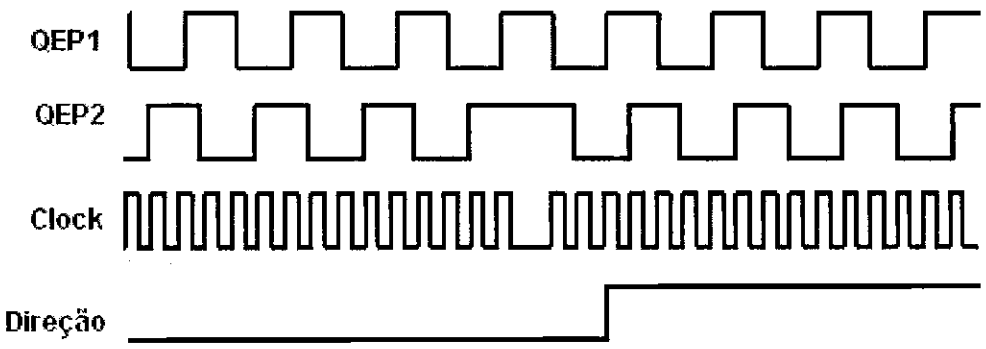


Figura 7.34: Esquema com os pulsos codificados em quadratura, o clock derivado e a direção de contagem.

7.8.3 Contagem do QEP

O *timer 2* (ou *timer 4*) sempre inicia a contagem do seu valor atual. Um valor desejado pode ser carregado no contador do *timer* antes de habilitar o modo QEP. Quando o circuito QEP for selecionado como fonte do *clock*, o *timer* ignorará os pinos de entrada TDIRA/B e TCLKINA/B.

Os *flags* de interrupção de período, *underflow*, *overflow* e comparação de um *timer* com um circuito QEP são gerados quando ocorrerem os seus respectivos eventos. Se as respectivas interrupções estiverem habilitadas, uma requisição de interrupção do periférico será gerada.

7.8.4 Registradores de Setup do Circuito QEP

Para iniciar a operação do circuito QEP no EVA/EVB, deve-se seguir os seguintes passos:

1. Carregar os registradores de contagem, de período e de comparação com valores desejados, se necessário;
2. Configurar o registrador **T2CON/T4CON** para setar o *timer 2/Timer 4* no modo de contagem direcional crescente/decrescente com os circuits QEP como fonte do *clock*, e habilitar o *timer* selecionado;
3. Configurar o registrador **CAPCONA/CAPCONB** paa habilitar o circuito QEP.

7.9 Interrupções do Gerenciador de Eventos

As interrupções do EV são organizadas em três grupos: A, B e C. Cada grupo está associado com um diferente registrador de sinalização da interrupção e de habilitação da interrupção. Na tabela **??** está indicado os registradores responsáveis pelas interrupções nos EVs.

Tabela 7.4: Registradores de sinalização e de habilitação das interrupções.

Registradores do EVA	Registradores do EVB
EVAIFRA	EVAIMRA
EVAIFRB	EVAIMRB
EVAIFRC	EVAIMRC
EVBIFRA	EVBIMRA
EVBIFRB	EVBIMRB
EVBIFRC	EVBIMRC

Os níveis de prioridade e os vetores das interrupções associados com os três grupos de interrupções do EV estão indicados na tabela 3.4.

7.9.1 Registradores de Sinalização da Interrupção do EVA

Nas figuras 7.35, 7.36 e 7.37 estão ilustrados, respectivamente, os esquemas com as posições dos bits nos registradores **EVAIFRA, EVAIFRB** e **EVAIFRC**.

15–11	10	9	8
Reservado	T1OFINT FLAG	T1UFINT FLAG	T1CINT FLAG
R-0	RW1C-0	RW1C-0	RW1C-0

7	6–4	3	2	1	0
T1PINT FLAG	Reservado	CMP3INT FLAG	CMP2INT FLAG	CMP1INT FLAG	PDPINTA FLAG
RW1C-0	R-0	RW1C-0	RW1C-0	RW1C-0	RW1C-0

Figura 7.35: Esquema com as posições dos *bits* no registrador EVAIFRA.

OBS: **R** = acesso a leitura, **W1C** = zerado quando se escreve um '1' nele, **-n** = valor após o *reset*.

- **Bit 10 - T1OFINT FLAG** - Interrupção de *overflow* no *timer 1*.

  0 - não ocorreu um *overflow* no timer 1.

  1 - Ocorreu um *overflow* no timer 1.

- **Bit 9 - T1UFINT FLAG** - Interrupção de *underflow* no *timer 1*.

  0 - não ocorreu um *underflow* no timer 1.

  1 - Ocorreu um *underflow* no timer 1.

- **Bit 8 - T1CINT FLAG** - Interrupção de comparação no *timer 1*.

  0 - não ocorreu uma igualdade na comparação entre os registradores **T1CNT** e o **T1CMPR**.

  1 - Ocorreu uma igualdade na comparação entre os registradores **T1CNT** e o **T1CMPR**.

- **Bit 7 - T1PINT FLAG** - Interrupção de período no *timer 1*.

  0 - não ocorreu uma igualdade na comparação entre os registradores **T1CNT** e o **T1PR**.

  1 - Ocorreu uma igualdade na comparação entre os registradores **T1CNT** e o **T1PR**.

- **Bit 3 - CMP3INT FLAG** - Interrupção no comparador 3.

  0 - não ocorreu uma igualdade na comparação entre o contador do *timer* selecionado e o registrador **CMPR3**.

  1 - Ocorreu uma igualdade na comparação entre o contador do *timer* selecionado e o registrador **CMPR3**.

- **Bit 2 - CMP2INT FLAG** - Interrupção no comparador 2.

  0 - não ocorreu uma igualdade na comparação entre o contador do *timer* selecionado e o registrador **CMPR2**.

  1 - Ocorreu uma igualdade na comparação entre o contador do *timer* selecionado e o registrador **CMPR2**.

- **Bit 1 - CMP1INT FLAG** - Interrupção no comparador 1.

  0 - não ocorreu uma igualdade na comparação entre o contador do *timer* selecionado e o registrador **CMPR1**.

  1 - Ocorreu uma igualdade na comparação entre o contador do *timer* selecionado e o registrador **CMPR1**.

- **Bit 0 - PDPINTA FLAG** - Interrupção de Proteção do Drive de Potência .

  0 - não ocorreu uma Interrupção de Proteção do Drive de Potência.

  1 - Ocorreu uma Interrupção de Proteção do Drive de Potência.

15–4	3	2	1	0
Reservado	T2OFINT FLAG	T2UFINT FLAG	T2CINT FLAG	T2PINT FLAG
R-0	RW1C-0	RW1C-0	RW1C-0	RW1C-0

Figura 7.36: Esquema com as posições dos *bits* no registrador EVAIFRB.

OBS: **R** = acesso a leitura, **W1C** = zerado quando se escreve um '1' nele, **-n** = valor após o *reset*.

- **Bit 3 - T2OFINT FLAG** - Interrupção de *overflow* no *timer 2*.

  0 - não ocorreu um *overflow* no timer 2.

  1 - Ocorreu um *overflow* no timer 2.

- **Bit 2 - T2UFINT FLAG** - Interrupção de *underflow* no *timer 2*.

  0 - não ocorreu um *underflow* no timer 2.

  1 - Ocorreu um *underflow* no timer 2.

- **Bit 1 - T2CINT FLAG** - Interrupção de comparação no *timer 2*.

  0 - não ocorreu uma igualdade na comparação entre os registradores **T2CNT** e o **T2CMPR**.

  1 - Ocorreu uma igualdade na comparação entre os registradores **T2CNT** e o **T2CMPR**.

- **Bit 0 - T2PINT FLAG** - Interrupção de período no *timer 2*.

  0 - não ocorreu uma igualdade na comparação entre os registradores **T2CNT** e o **T2PR**.

  1 - Ocorreu uma igualdade na comparação entre os registradores **T2CNT** e o **T2PR**.

15-3	2	1	0
Reservado	CAP3INT FLAG	CAP2INT FLAG	CAP1INT FLAG
R-0	RW1C-0	RW1C-0	RW1C-0

Figura 7.37: Esquema com as posições dos *bits* no registrador EVAIFRC.

OBS: **R** = acesso a leitura, **W1C** = zerado quando se escreve um '1' nele, **-n** = valor após o *reset*.

- **Bit 2 - CAP3INT FLAG** - Interrupção na unidade de captura 3.

  0 - não ocorreu uma interrupção na unidade de captura 3.

  1 - ocorreu uma interrupção na unidade de captura 3.

- **Bit 1 - CAP2INT FLAG** - Interrupção na unidade de captura 2.

  0 - não ocorreu uma interrupção na unidade de captura 2.

  1 - ocorreu uma interrupção na unidade de captura 2.

- **Bit 0 - CAP1INT FLAG** - Interrupção na unidade de captura 1.

  0 - não ocorreu uma interrupção na unidade de captura 1.

  1 - ocorreu uma interrupção na unidade de captura 1.

7.9.2 Registradores de Habilitação da Interrupção do EVA

Nas figuras 7.38, 7.39 e 7.40 estão ilustrados, respectivamente, os esquemas com as posições dos bits nos registradores **EVAIMRA**, **EVAIMRB** e **EVAIMRC**.

15–11	10	9	8
Reservado	T1OFINT ENABLE	T1UFINT ENABLE	T1CINT ENABLE
R-0	RW-0	RW-0	RW-0

7	6–4	3	2	1	0
T1PINT ENABLE	Reservado	CMP3INT ENABLE	CMP2INT ENABLE	CMP1INT ENABLE	PDPINTA ENABLE
RW-0	R-0	RW-0	RW-0	RW-0	RW-1

Figura 7.38: Esquema com as posições dos *bits* no registrador EVAIMRA.

OBS: **R** = acesso a leitura, **W1C** = zerado quando se escreve um '1' nele, **-n** = valor após o *reset*.

- **Bit 10 - T1OFINT ENABLE** - Habilita/desabilita interrupção de *overflow* no *timer 1*.

  0 - Desabilita.

  1 - Habilita.

- **Bit 9 - T1UFINT ENABLE** - Habilita/desabilita interrupção de *underflow* no *timer 1*.

  0 - Desabilita.

  1 - Habilita.

- **Bit 8 - T1CINT ENABLE** - Habilita/desabilita interrupção de comparação no *timer 1*.

  0 - Desabilita.

  1 - Habilita.

- **Bit 7 - T1PINT ENABLE** - Habilita/desabilita interrupção de período no *timer 1*.

  0 - Desabilita.

  1 - Habilita.

- **Bit 3 - CMP3INT ENABLE** - Habilita/desabilita interrupção no comparador 3.

  0 - Desabilita.

  1 - Habilita.

- **Bit 2 - CMP2INT ENABLE** - Habilita/desabilita interrupção no comparador 2.

  0 - Desabilita.

  1 - Habilita.

- **Bit 1 - CMP1INT ENABLE** - Habilita/desabilita interrupção no comparador 1.

  0 - Desabilita.

  1 - Habilita.

- **Bit 0 - PDPINTA FLAG** - Habilita/desabilita interrupção de Proteção do Drive de Potência.

  0 - Desabilita.

  1 - Habilita.

15–4	3	2	1	0
Reservado	T2OFINT ENABLE	T2UFINT ENABLE	T2CINT ENABLE	T2PINT ENABLE
R-0	RW-0	RW-0	RW-0	RW-0

Figura 7.39: Esquema com as posições dos *bits* no registrador EVAIMRB.

OBS: **R** = acesso a leitura, **W1C** = zerado quando se escreve um '1' nele, **-n** = valor após o *reset*.

- **Bit 3 - T2OFINT FLAG** - Habilita/desabilita interrupção de *overflow* no *timer 2*.

  0 - Desabilita.

  1 - Habilita.

- **Bit 2 - T2UFINT FLAG** - Habilita/desabilita interrupção de *underflow* no *timer 2*.

  0 - Desabilita.

  1 - Habilita.

- **Bit 1 - T2CINT FLAG** - Habilita/desabilita interrupção de comparação no *timer 2*.

  0 - Desabilita.

  1 - Habilita.

- **Bit 0 - T2PINT FLAG** - Habilita/desabilita interrupção de período no *timer 2*.

  0 - Desabilita.

  1 - Habilita.

15–3	2	1	0
Reservado	CAP3INT ENABLE	CAP2INT ENABLE	CAP1INT ENABLE
R-0	RW-0	RW-0	RW-0

Figura 7.40: Esquema com as posições dos *bits* no registrador EVAIMRC.

OBS: **R** = acesso a leitura, **W1C** = zerado quando se escreve um '1' nele, **-n** = valor após o *reset*.

- **Bit 2 - CAP3INT FLAG** - Habilita/desabilita interrupção na unidade de captura 3.

  0 - Desabilita.

  1 - Habilita.

- **Bit 1 - CAP2INT FLAG** - Habilita/desabilita interrupção na unidade de captura 2.

  0 - Desabilita.

  1 - Habilita.

- **Bit 0 - CAP1INT FLAG** - Habilita/desabilita interrupção na unidade de captura 1.

  0 - Desabilita.

  1 - Habilita.

7.9.3 Registradores de Sinalização da Interrupção do EVB

Nas figuras 7.41, 7.42 e 7.43 estão ilustrados, respectivamente, os esquemas com as posições dos bits nos registradores **EVBIFRA**, **EVBIFRB** e **EVBIFRC**.

15–11	10	9	8
Reservado	T3OFINT FLAG	T3UFINT FLAG	T3CINT FLAG
R-0	RW1C-0	RW1C-0	RW1C-0

7	6–4	3	2	1	0
T3PINT FLAG	Reservado	CMP6INT FLAG	CMP5INT FLAG	CMP4INT FLAG	PDPINTB FLAG
RW1C-0	R-0	RW1C-0	RW1C-0	RW1C-0	RW1C-0

Figura 7.41: Esquema com as posições dos *bits* no registrador EVBIFRA.

OBS: **R** = acesso a leitura, **W1C** = zerado quando se escreve um '1' nele, **-n** = valor após o *reset*.

- **Bit 10 - T3OFINT FLAG** - Interrupção de *overflow* no *timer 3*.

  0 - não ocorreu um *overflow* no timer 3.

  1 - Ocorreu um *overflow* no timer 3.

- **Bit 9 - T3UFINT FLAG** - Interrupção de *underflow* no *timer 3*.

  0 - não ocorreu um *underflow* no timer 3.

  1 - Ocorreu um *underflow* no timer 3.

- **Bit 8 - T3CINT FLAG** - Interrupção de comparação no *timer 3*.

  0 - não ocorreu uma igualdade na comparação entre os registradores **T3CNT** e o **T3CMPR**.

  1 - Ocorreu uma igualdade na comparação entre os registradores **T3CNT** e o **T3CMPR**.

- **Bit 7 - T3PINT FLAG** - Interrupção de período no *timer 3*.

  0 - não ocorreu uma igualdade na comparação entre os registradores **T3CNT** e o **T3PR**.

  1 - Ocorreu uma igualdade na comparação entre os registradores **T3CNT** e o **T3PR**.

- **Bit 3 - CMP6INT FLAG** - Interrupção no comparador 6.

  0 - não ocorreu uma igualdade na comparação entre o contador do *timer* selecionado e o registrador **CMPR6**.

  1 - Ocorreu uma igualdade na comparação entre o contador do *timer* selecionado e o registrador **CMPR6**.

- **Bit 2 - CMP5INT FLAG** - Interrupção no comparador 5.

  0 - não ocorreu uma igualdade na comparação entre o contador do *timer* selecionado e o registrador **CMPR5**.

  1 - Ocorreu uma igualdade na comparação entre o contador do *timer* selecionado e o registrador **CMPR5**.

- **Bit 1 - CMP4INT FLAG** - Interrupção no comparador 4.

  0 - não ocorreu uma igualdade na comparação entre o contador do *timer* selecionado e o registrador **CMPR4**.

  1 - Ocorreu uma igualdade na comparação entre o contador do *timer* selecionado e o registrador **CMPR4**.

- **Bit 0 - PDPINTB FLAG** - Interrupção de Proteção do Drive de Potência .

  0 - não ocorreu uma Interrupção de Proteção do Drive de Potência.

  1 - Ocorreu uma Interrupção de Proteção do Drive de Potência.

15-4	3	2	1	0
Reservado	T4OFINT FLAG	T4UFINT FLAG	T4CINT FLAG	T4PINT FLAG
R-0	RW1C-0	RW1C-0	RW1C-0	RW1C-0

Figura 7.42: Esquema com as posições dos *bits* no registrador EVBIFRB.

OBS: **R** = acesso a leitura, **W1C** = zerado quando se escreve um '1' nele, **-n** = valor após o *reset*.

- **Bit 3 - T4OFINT FLAG** - Interrupção de *overflow* no *timer 4*.

  0 - não ocorreu um *overflow* no timer 4.

  1 - Ocorreu um *overflow* no timer 4.

- **Bit 2 - T4UFINT FLAG** - Interrupção de *underflow* no *timer 4*.

  0 - não ocorreu um *underflow* no timer 4.

  1 - Ocorreu um *underflow* no timer 4.

- **Bit 1 - T4CINT FLAG** - Interrupção de comparação no *timer 4*.

  0 - não ocorreu uma igualdade na comparação entre os registradores **T4CNT** e o **T4CMPR**.

  1 - Ocorreu uma igualdade na comparação entre os registradores **T4CNT** e o **T4CMPR**.

- **Bit 0 - T4PINT FLAG** - Interrupção de período no *timer 4*.

  0 - não ocorreu uma igualdade na comparação entre os registradores **T4CNT** e o **T4PR**.

  1 - Ocorreu uma igualdade na comparação entre os registradores **T2CNT** e o **T2PR**.

15–3	2	1	0
Reservado	CAP6INT FLAG	CAP5INT FLAG	CAP4INT FLAG
R-0	RW1C-0	RW1C-0	RW1C-0

Figura 7.43: Esquema com as posições dos *bits* no registrador EVBIFRC.

OBS: **R** = acesso a leitura, **W1C** = zerado quando se escreve um '1' nele, **-n** = valor após o *reset*.

- **Bit 2 - CAP6INT FLAG** - Interrupção na unidade de captura6.

  0 - não ocorreu uma interrupção na unidade de captura 6.

  1 - ocorreu uma interrupção na unidade de captura 6.

- **Bit 1 - CAP5INT FLAG** - Interrupção na unidade de captura 5.

  0 - não ocorreu uma interrupção na unidade de captura 5.

  1 - ocorreu uma interrupção na unidade de captura 5.

- **Bit 0 - CAP4INT FLAG** - Interrupção na unidade de captura 4.

  0 - não ocorreu uma interrupção na unidade de captura 4.

  1 - ocorreu uma interrupção na unidade de captura 4.

7.9.4 Registradores de Habilitação da Interrupção do EVB

Nas figuras 7.44, 7.45 e 7.46 estão ilustrados, respectivamente, os esquemas com as posições dos bits nos registradores **EVBIMRA**, **EVBIMRB** e **EVBIMRC**.

15–11	10	9	8
Reservado	T3OFINT ENABLE	T3UFINT ENABLE	T3CINT ENABLE
R-0	RW-0	RW-0	RW-0

7	6–4	3	2	1	0
T3PINT ENABLE	Reservado	CMP6INT ENABLE	CMP5INT ENABLE	CMP4INT ENABLE	PDPINTB ENABLE
RW-0	R-0	RW-0	RW-0	RW-0	RW-1

Figura 7.44: Esquema com as posições dos *bits* no registrador EVBIMRA.

OBS: **R** = acesso a leitura, **W1C** = zerado quando se escreve um '1' nele, **-n** = valor após o *reset*.

- **Bit 10 - T3OFINT ENABLE** - Habilita/desabilita interrupção de *overflow* no *timer 3*.

  0 - Desabilita.

  1 - Habilita.

- **Bit 9 - T3UFINT ENABLE** - Habilita/desabilita interrupção de *underflow* no *timer 3*.

  0 - Desabilita.

  1 - Habilita.

- **Bit 8 - T3CINT ENABLE** - Habilita/desabilita interrupção de comparação no *timer 3*.

  0 - Desabilita.

  1 - Habilita.

- **Bit 7 - T3PINT ENABLE** - Habilita/desabilita interrupção de período no *timer 3*.

  0 - Desabilita.

  1 - Habilita.

- **Bit 3 - CMP6INT ENABLE** - Habilita/desabilita interrupção no comparador 6.

  0 - Desabilita.

  1 - Habilita.

- **Bit 2 - CMP5INT ENABLE** - Habilita/desabilita interrupção no comparador 5.

  0 - Desabilita.

  1 - Habilita.

- **Bit 1 - CMP4INT ENABLE** - Habilita/desabilita interrupção no comparador 4.

  0 - Desabilita.

  1 - Habilita.

- **Bit 0 - PDPINTB FLAG** - Habilita/desabilita interrupção de Proteção do Drive de Potência.

  0 - Desabilita.

  1 - Habilita.

15–4	3	2	1	0
Reservado	T4OFINT ENABLE	T4UFINT ENABLE	T4CINT ENABLE	T4PINT ENABLE
R-0	RW-0	RW-0	RW-0	RW-0

Figura 7.45: Esquema com as posições dos *bits* no registrador EVBIMRB.

OBS: **R** = acesso a leitura, **W1C** = zerado quando se escreve um '1' nele, **-n** = valor após o *reset*.

- **Bit 3 - T4OFINT FLAG** - Habilita/desabilita interrupção de *overflow* no *timer 4*.

  0 - Desabilita.

  1 - Habilita.

- **Bit 2 - T4UFINT FLAG** - Habilita/desabilita interrupção de *underflow* no *timer 4*.

  0 - Desabilita.

  1 - Habilita.

- **Bit 1 - T4CINT FLAG** - Habilita/desabilita interrupção de comparação no *timer 4*.

  0 - Desabilita.

  1 - Habilita.

- **Bit 0 - T4PINT FLAG** - Habilita/desabilita interrupção de período no *timer 4*.

  0 - Desabilita.

  1 - Habilita.

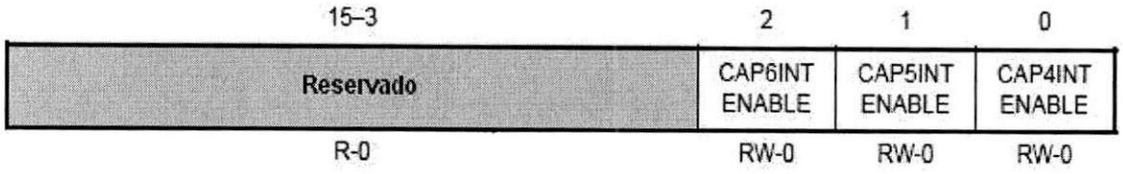


Figura 7.46: Esquema com as posições dos *bits* no registrador EVBIMRC.

OBS: **R** = acesso a leitura, **W1C** = zerado quando se escreve um '1' nele, **-n** = valor após o *reset*.

- **Bit 2 - CAP6INT FLAG** - Habilita/desabilita interrupção na unidade de captura 6.

  0 - Desabilita.

  1 - Habilita.

- **Bit 1 - CAP5INT FLAG** - Habilita/desabilita interrupção na unidade de captura 5.

  0 - Desabilita.

  1 - Habilita.

- **Bit 0 - CAP4INT FLAG** - Habilita/desabiiita interrupção na unidade de captura 4.

  0 - Desabilita.

  1 - Habilita.

Capítulo 8

Conversor Analógico Digital

8.1 Características

O conversor analógico digital do TMS320LF2407A possui as seguintes características:

- Um ADC de 10 *bits* com S/H (*Sample and Hold*);
- Tempo de conversão (S/H + conversão) de 375 ns;
- Dezesseis entradas analógicas multiplexadas (ADCIN0 - ADCIN15);
- Capacidade de autosequenciamento. Até 16 autoconversões em uma única sessão. Cada sessão pode ser programada para selecionar ao menos um dos dezesseis canais de entrada;
- Dois sequenciadores de oito estados independentes (SEQ1 e SEQ2) que podem ser operados individualmente no modo de sequenciamento dual, e um sequenciador de dezesseis estados (SEQ) que opera no modo de sequenciamento cascateado. A palavra "estados"indica o número de autoconversões que podem ser desempenhados com um sequenciador.
- Quatro registradores de controle do sequenciamento (**CHSELSEQn**) que determinam a sequência dos canais analógicos que são usados para conversão em algum modo de sequenciamento;
- Dezesseis registradores de resultados para armazenar os valores convertidos (**RESULT0 - RESULT15**);

- Várias formas de se iniciar uma conversão:
  - Por *software*: usando o *bit* SOC SEQn;
  - EVA: gerenciador de eventos A;
  - EVB: gerenciador de eventos B;
  - Externo: usando o pino ADCSOC.
- Flexível controle de interrupção;
- Sequenciador pode operar no modo *start/stop*, permitindo vários gatilhos de sequenciamento no tempo para sincronizar as conversões;
- EVA e EVB podem independentemente gatilhar, respectivamente, SEQ1 e SEQ2, apenas no modo de sequenciamento dual;
- Controle do *prescaler* para o tempo de aquisição da janela S/H;
- Modo de calibração.

8.2 Visão Geral do ADC

8.2.1 Princípio de Operação do Sequenciador de Autoconversão

O sequenciador do ADC consiste de dois sequenciadores de oito estados (SEQ1 e SEQ2) que podem também serem cscateados em um único sequenciador de dezesseis estados (SEQ). Nas figuras 8.1 e 8.2 estão ilustrados, respectivamente, os diagramas de blocos do autosequenciador ADC nos modos cascateado e dual.

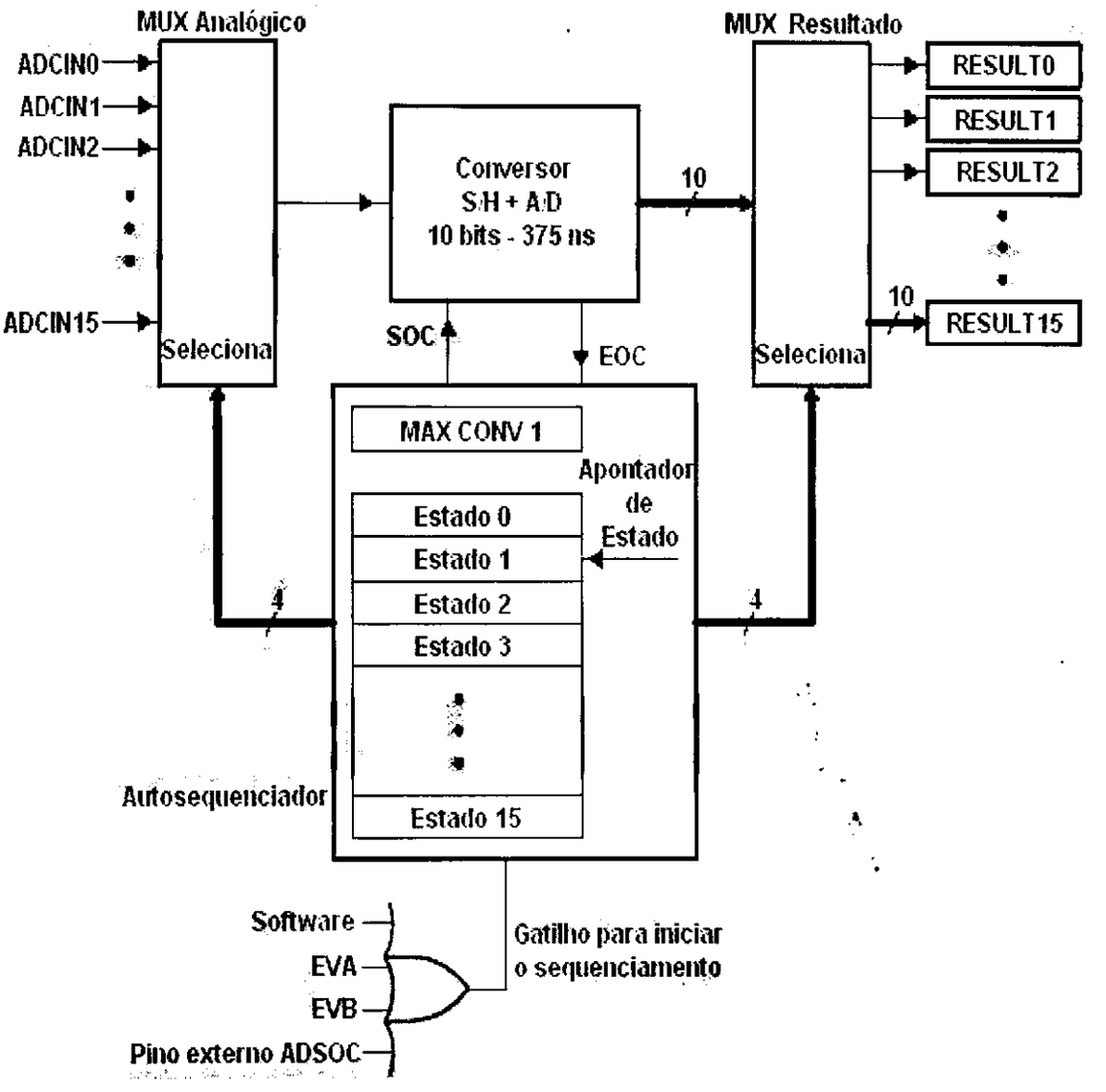


Figura 8.1: Diagrama de blocos do autosequenciador ADC no modo cascateado.

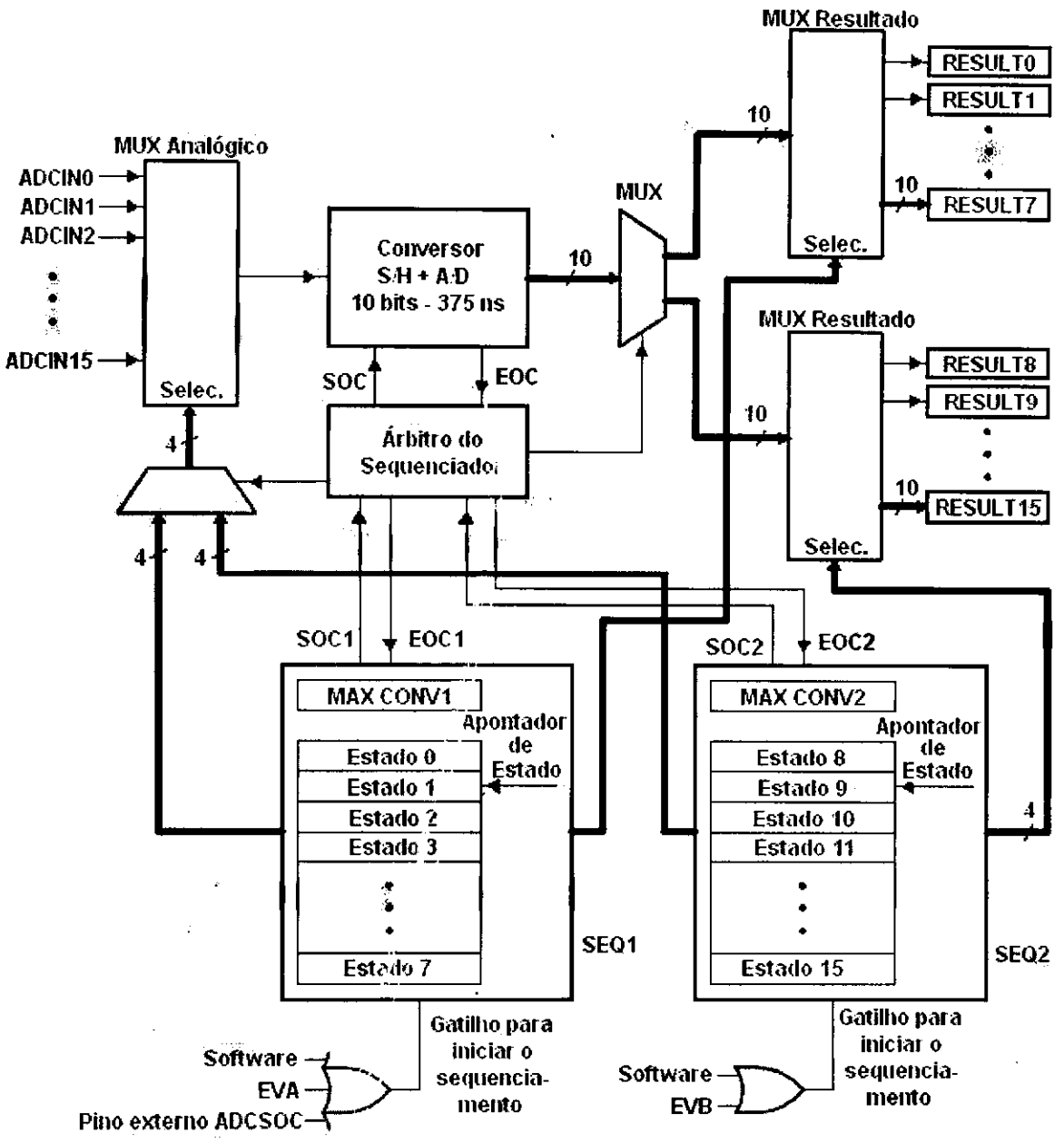


Figura 8.2: Diagrama de blocos do autosequenciador ADC no modo dual.

Em ambos os casos, o ADC tem a habilidade de autosequenciar uma série de conversões. Para toda conversão, ao menos um dos 16 canais disponíveis podem ser selecionados através do MUX analógico. Após a conversão, o valor digital do canal selecionado será armazenado no apropriado registrador de resultados (**RESULTn**). O primeiro resultado é armazenado em **RESULT0**, o segundo em **RESULT1** e assim por diante. No TMS320LF2407A, existe apenas um conversor A/D, e ele é compartilhado pelos dois sequenciadores (SEQ1 e SEQ2) no modo de sequenciamento dual.

O canal de entrada analógico selecionado para cada sequenciamento de conversão é definido nos *bits* CONVnn do registrador **CHSELSEQn**. CONVnn é um campo de quatro *bits* que especifica ao menos um dos dezesseis canais para conversão. Desde que é possível desempenhar dezesseis conversões quando os sequenciadores estão no modo cascateado, dezesseis campos de quatro *bits* (CONV00 - CONV15) são disponíveis e estão distribuídos nos quatro registradores de dezesseis *bits* (**CHSELSEQ1 - CHSELSEQ4**). Os *bits* CONVnn podem ter algum valor entre 0 e 15. Os canais analógicos podem ser escolhidos em uma ordem desejada e o mesmo canal pode ser escolhido várias vezes.

8.2.2 Modo de Autosequenciamento sem Interrupção

A seguinte descrição se aplica aos sequenciadores SEQ1 e SEQ2. Neste modo, SEQ1/SEQ2 podem autosequenciar até oito conversões por canal em uma única sessão de sequenciamento. O resultado de cada conversão será armazenado em um dos oito registradores de resultado (**RESULT0 - RESULT7** para o SEQ1 e **RESULT8 - RESULT15** para o SEQ2). O número de conversões em uma sequência é controlado pelo registrador **MAXCONV** (um campo de três ou quatro *bits* do registrador **MAXCONV**), no qual é automaticamente carregado nos *bits* de *status* do contador de sequenciamento (SEQ CNTR3-0) do registrador **AUTO_SEQ_SR** no início de uma sessão de conversão autosequenciada. O campo MAX CONVn pode ter um valor variando de 0 à 7. Os *bits* SEQ CNTR contam de forma decrescente a partir do valor carregado nele até atingir o valor zero. O número de conversões completadas durante uma sessão de autosequenciamento é igual a (MAX CONVn + 1).

Suponha que são desejadas a realização de sete conversões do sequenciador SEQ1. Os canais a serem usados na conversão em uma sessão de autosequenciamento são: canais 2, 3, 2, 3, 6, 7 e 12. Então, MAX CONV1 deverá ser setado em seis e os registradores **CHSELSEQn** deverão ser setados com os seguintes valores:

CHSELSEQ1 = 0011 0010 0011 0010
CHSELSEQ2 = xxxx 1010 0111 0110
CHSELSEQ3 = xxxx xxxx xxxx xxxx
CHSELSEQ4 = xxxx xxxx xxxx xxxx

A conversão inicia quando o gatilho de início de conversão (SOC-Start Of Conversion) for recebido pelo sequenciador. O gatilho SOC também carrega os *bits* SEQ CNTRn. Aqueles canais que foram especificados nos registradores **CHSELSEQn** são colocados para a conversão na sequência pré-determinada. Os *bits* SEQ CNTRn são decrementados de '1' automaticamente após toda conversão. Quando SEQ CNTRn atingir o valor zero, duas coisas podem acontecer dependendo do *status* do *bit* CONT RUN do registrador **ADCTRL1**:

1. Se CONT RUN for '1', a sequência de conversão inicia novamente, automaticamente. Isto é, SEQ CNTRn é recarregado com o valor original de MAX CONV1 e SEQ1 inicia a conversão de CONV00. O usuário deve garantir que os registradores de resultado foram lidos antes da próxima sequência de conversão iniciar.
2. Se CONT RUN for '0', o sequenciador permanece no último estado (CONV06 neste exemplo) e SEQ CNTRn cntinua com o valor zero.

Desde que o *flag* de interrupção será setado toda vez que SEQ CNTRn atingir o valor '0', o usuário pode, se necessário, resetar manualmente o sequenciador (usando o *bit* RST SEQn do registrador **ADCTRL2**) no serviço de rotina da interrupção, de modo que SEQ CNTRn seja recarregado com o valor original do MAX CONV1 no próximo gatilho SOC e o estado de SEQ1 seja setado para CONV00. Esta característica é utilizada no modo de operação *start/stop* do sequenciador. Este exemplo também é aplicado para SEQ2 com perquenas diferenças.

8.2.3 Modo Sequenciador Start/Stop

Os sequenciadores SEQ1, SEQ2 e SEQ podem ser operados no modo *start/stop*, no qual é sincronizado por vários gatilhos SOC, separados no tempo. Este modo é idêntico ao modo exemplificado anteriomente. Porém o sequenciador pode ser gatilhado novamente sem ser resetado ao estado inicial CONV00 (se esse estado for o primeiro). Então, o sequenciador não é resetado no serviço de rotina da interrupção. Entretanto, quando uma sequência de conversão terminar, o sequenciador permanece no atual estado de conversão. O *bit* CONT RUN do registrador **ADCTRL1** deve ser zerado, isto é, desabilitado neste modo.

Suponha que se deseja desempenhar autoconversões utilizando o modo *start/stop*. As três autoconversões (I1, I2 e I3) são gatihadas quando ocorrer um *underflow* (interrupção de underflow) do *timer* e as três autoconversões (V1, V2 e V3) são gatihadas quando ocorrer uma igualdade entre os valores do contador e do respectivo registrador de período (interrupção de período). Os gatilhos são separados por um tempo de 25 use são fornecidos pelo gerenciador de eventos A (EVA). Veja que apenas o SEQ1 pode ser utilizado nesse caso. Na figura 8.3 está ilustrado um esquema com os pontos em que os gatilhos são acionados para iniciar as autoconversões.

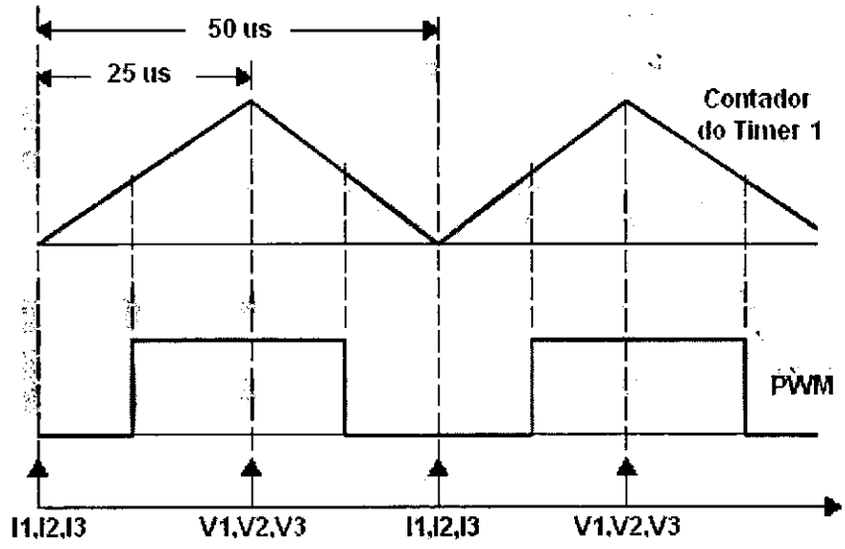


Figura 8.3: Esquema com os pontos em que os gatilhos são acionados para iniciar as autoconversões.

Neste caso, MAX CONV1 será setado em três e os registradores **CHSELSEQn** serão setados com:

CHSELSEQ1 = V1 I3 I2 I1
CHSELSEQ2 = xx xx V3 V2
CHSELSEQ3 = xx xx xx xx
CHSELSEQ4 = xx xx xx xx

Uma vez resetados e inicializados, SEQ1 espera pelo gatilho. No primeiro gatilho, três conversões (CONV00 - I1, CONV01 - I2 e CONV02 - I3) são desempenhadas. Após 25 us, o segundo gatilho ocorrerá e mais três conversões (CONV03 - V1, CONV04 - V2 e CONV05 - V3) serão desempenhadas.

O valor de MAX CONV1 é automaticamente carregado em SEQ CNTRn para ambos os gatilhos. Se um número diferente de conversões forem requisitados no segundo gatilho, o usuário deverá mudar o valor de MAX CONV1 por **software**, caso contrário, o atual valor será utilizado novamente. Isto pode ser feito em um serviço de rotina da interrupção.

No final da última sessão de autoconversão, os registradores de resultado terão os seguintes valores:

RESULT0 = I1, **RESULT4** = V1
RESULT1 = I2, **RESULT5** = V2
RESULT3 = I3, **RESULT6** = V3

Neste momento, SEQ1 sabe atual estado do segundo gatilho. Agora o usuario pode resetar SEQ1 por *software* para o estado CONV00 e repetir os mesmos gatilhos.

8.2.4 Formas de Gatilhamento dos Sequenciadores

Cada sequenciador possui diversas formas de ser gatilhado, para iniciar uma conversão. Na tabela 8.1 está indicado as formas de gatilho dos sequenciadores.

Tabela 8.1: Formas de gatilhamento dos sequenciadores.

SEQ1	SEQ2	SEQ
Por software	Por software	Por software
Pelo EVA	Pelo EVB	Pelo EVA e EVB
Pelo pino externo ADC SOC	-	Pelo pino externo ADC SOC

8.2.5 Operação de Interrupção Durante Conversões Sequenciadas

O sequenciador pode gerar interrupções sob dois modos de operação. Estes modos são determinados pelos *bits* de controle que habilitam o modo de interrupção no registrador **ADCTRL2**. Uma variação do exemplo de autoconversão utilizando o modo *start/stop* explicado anteriormente, pode ser usado para mostrar como o modo 1 de interrupção e o modo 2 de interrupção são utilizados sob diferentes condições de operação. Na figura 8.4 está ilustrado as operações de interrupção durante as conversões sequenciadas.

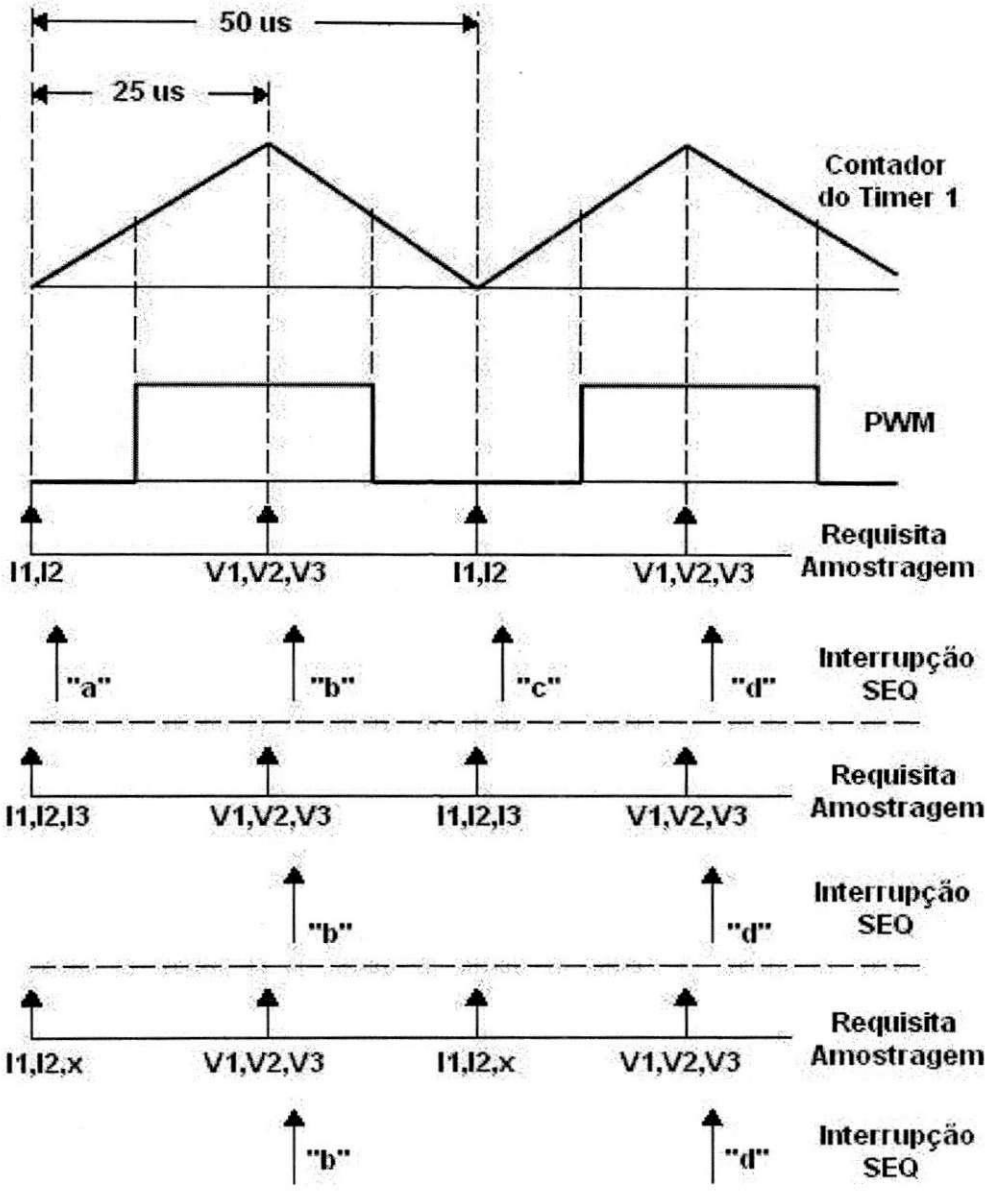


Figura 8.4: Esquema com as operações de interrupção durante as conversões sequenciadas.

CASO 1: O número de amostras na primeira e na segunda sequência são diferentes.

- Modo 1 de operação da interrupção (a requisição de interrupção ocorre sempre no final da sequência):

1. O Sequenciador é inicializado com MAX CONVn = 1 para converter I1 e I2;
2. Na rotina de serviço de interrupção "a", MAX CONVn é alterado para dois (por *software*) para converter V1, V2 e V3;
3. Na rotina de serviço de interrupção "b", os seguintes eventos podem acontecer:
   (a) MAX CONVn é alterado para '1' novamente, para converter I1 e I2;
   (b) Valores I1, I2, V1, V2 e V3 são lidos dos registradores de resultado do ADC;
   (c) O saequenciador é resetado.
4. Os passos 2 e 3 são repetidos. Veja que o flag de interrupção é setado toda vez que SEQ CNTRn atingir o valor zero e ambas as interrupções são reconhecidas.

CASO 2: O número de amostras na primeira e na segunda sequência são iguais.

- Modo 2 de operação da interrupção (a requisição de interrupção ocorre em todo final de sequência):

1. O Sequenciador é inicializado com MAX CONVn = 2 para converter I1, I2 e I3 (ou V1, V2 e V3);
2. Na rotina de serviço de interrupção "b"e "d", os seguintes eventos podem acontecer:
   (a) Valores I1, I2, I3, V1, V2 e V3 são lidos dos registradores de resultado do ADC;
   (b) O saequenciador é resetado.
3. O passo 2 é repetido. Veja que o flag de interrupção é setado toda vez que SEQ CNTRn atingir o valor zero. Isto acontecerá após o ADC finalizar a coversão de I1, I2 e I3, e após converter V1, V2 e V3. Mas, apenas o final do sequenciamento gerado após a conversão de V1, V2 e V3 gatilha a interrupção.

CASO 3: O número de amostras na primeira e na segunda sequência são iguais (com leitura falsa).

- Modo 2 de operação da interrupção (a requisição de interrupção ocorre em todo final de sequência):

1. O Sequenciador é inicializado com MAX CONVn = 2 para converter I1, I2 e x;
2. Na rotina de serviço de interrupção "b" e "d", os seguintes eventos podem acontecer:
   (a) Valores I1, I2, x, V1, V2 e V3 são lidos dos registradores de resultado do ADC;
   (b) O saequenciador é resetado.
3. O passo 2 é repetido. Veja que a terceira amostragem é uma amostra falsa. Porém, para minimizar uma sobrecarga do serviço de rotina da interrupção e a intervenção da CPU, é vantagem escolher de todas as outras, as características de requisição de interrupção do modo 2.

8.2.6 Prescaler do Clock do ADC

O bloco S/H no ADC do TMS320LF2407A foi feito para acomodar variações de impedância na fonte. Isto é realizado pelos *bits* ACQ PS3-ACQ PS0 e o *bit* CPS do registrador **ADCTRL1**. O processo de conversão A/D pode ser dividido em dois segmentos de tempo, como ilustrado na figura 8.5.

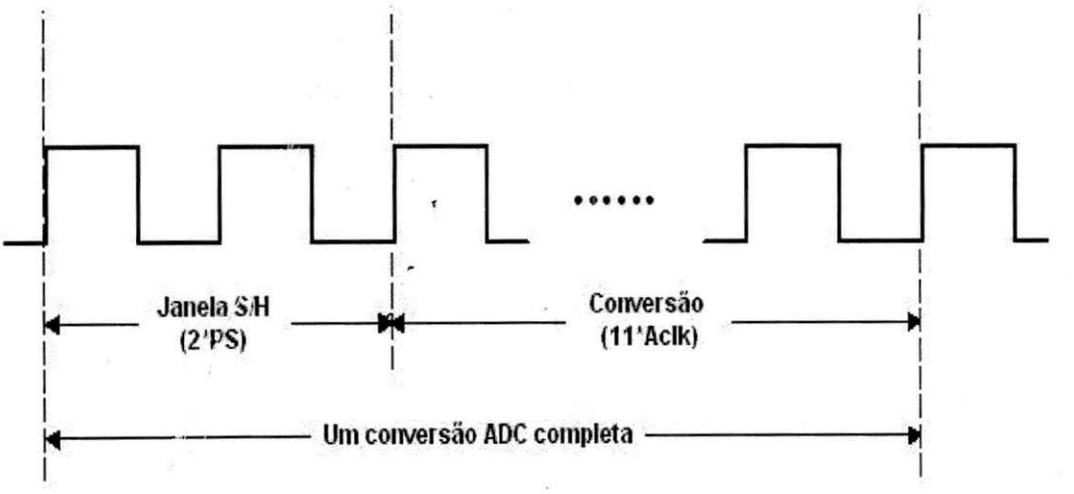


Figura 8.5: Esquema com os segmentos de tempo de uma conversão A/D.

PS será igual ao *clock* da CPU se o *prescaler* for '1' (isto é, os *bits* ACQ PS3-ACQ-PS0 são todos zero) e se CPS for '0'. Para algum outro valor do prescaler, o tamanho de PS será aumentado. Se o *bit* CPS for '1', a janela S/H será duplicada. Na figura 8.6 está ilustrado um diagrama com as possíveis posições indicadas pelos bits do prescaler no ADC.

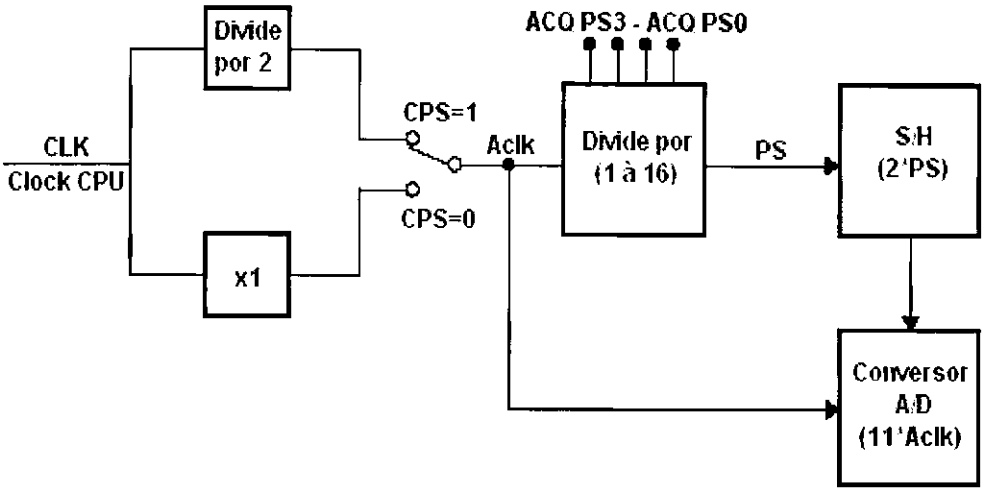


Figura 8.6: diagrama com as possíveis posições indicadas pelos bits do prescaler no ADC.

8.2.7 Calibração

No modo de calibração, os sequenciadores não estão operacionais, e os pinos ADCINn não estão conectados ao conversor A/D. O sinal que será conectado na entrada do A/D é determinado pelos *bits* BRG ENA e HI/LO do registrador **ADCTRL1**. Esses dois sinais conectam ambos Vreflo ou Vrefhi ou a metade deles para a entrada do conversor A/D e uma única conversão será feita. O modo de calibração pode calcular erros de *offset* do ADC. O complemento de 2 do erro de *offset* deverá então ser carregado no registrador **CALIBRATION** (a operação complemento de '2' é aplicável apenas para erros negativos). O *hardware* do ADC adiciona automaticamente o erro de *offset* ao valor convertido. Na figura 8.7 está ilustrado o esquema com as posições dos *bits* no registrador **CALIBRATION**.

O registrador **CALIBRATION** armazena o resultado final da calibração no modo de calibração. No modo normal do ADC, o valor de **CALIBRATION** é automaticamente adicionado a saída do ADC antes do resultado ser armazenado nos registradores **RESULTn**.

15	14	13	12	11	10	9	8
D9	D8	D7	D6	D5	D4	D3	D2

7	6	5	4	3	2	1	0
D1	D0	0	0	0	0	0	0

Figura 8.7: Esquema com as posições dos *bits* no registrador CALIBRATION.

8.2.8 Registrador de Controle do ADC 1 - ADCTRL1

Na figura 8.8 está ilustrado o esquema com as posições dos *bits* no registrador **ADCTRL1**.

15	14	13	12	11	10	9	8
Reservado	RESET	SOFT	FREE	ACQ PS3	ACQ PS2	ACQ PS1	ACQ PS0
	RS-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
CPS	CONT RUN	INT PRI	SEQ CASC	CAL ENA	BRG ENA	HI/LO	STEST ENA
RW-0	RW-0	RW-0	RW-0				

Figura 8.8: Esquema com as posições dos *bits* no registrador ADCTRL1.

OBS: **R** = acesso a leitura, **S** = pode apenas ser setado, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bit 14 - RESET** - Reseta o módulo ADC por *software*. Todos os bits dos registradores e os estados dos sequenciadores são resetados.

  0 - Sem efeito.

  1 - Reseta módulo ADC.

- **Bits 13-12 - SOFT, FREE** - Estes *bits* determinam o que ocorrerá quando ocorrer uma suspensão da emulação. No modo *free*, o periférico pode continuar o que está fazendo. No modo *stop*, o periférico pode parar imediatamente ou parar quando a atual operação for concluída.

  00 - Pára imediatamente na suspensão.

  10 - Continua operando mesmo quando ocorrer uma suspensão.

  01 - Pára quando a atual operação de conversão for concluída.

  11 - Continua operando mesmo quando ocorrer uma suspensão.

- **Bit 11-8 - ACQ PS3-ACQ PS0** - *Bits* do *prescaler* da janela de tempo de aquisição. Nas tabelas 8.2 e 8.3 estão indicados os possíveis valores do *prescale* determinados por esses *bits* para um CLK de 30 MHz e 40 MHz.

Tabela 8.2: Fatores do Prescale determinados pelos bits ACQ PSn para um CLK de 30 MHz

ACQ PS3	ACQ PS2	ACQ PS1	ACQ PS0	Prescaler Dividido por	Janela de Tempo de Aquisição	Imped. CPS=0	Imped. CPS=1
0	0	0	0	1	2*Tclk	67	385
0	0	0	1	2	4*Tclk	385	1020
0	0	1	0	3	6*Tclk	702	1655
0	0	1	1	4	8*Tclk	1020	2290
0	1	0	0	5	10*Tclk	1337	2925
0	1	0	1	6	12*Tclk	1655	3560
0	1	1	0	7	14*Tclk	1972	4194
0	1	1	1	8	16*Tclk	2290	4829
1	0	0	0	9	18*Tclk	2607	5464
1	0	0	1	10	20*Tclk	2925	6099
1	0	1	0	11	22*Tclk	3242	6734
1	0	1	1	12	24*Tclk	3560	7369
1	1	0	0	13	26*Tclk	3877	8004
1	1	0	1	14	28*Tclk	4194	8639
1	1	1	0	15	30*Tclk	4512	9274
1	1	1	1	16	32*Tclk	4829	9909

OBS: O período Tclk depende do bit CPS do **ADCTRL1**. Se CPS for '0', Tclk = 1/CLK. Se CPS for '1', Tclk = 2*(1/CLK)

Tabela 8.3: Fatores do Prescale determinados pelos bits ACQ PSn para um CLK de 40 MHz

ACQ PS3	ACQ PS2	ACQ PS1	ACQ PS0	Prescaler Dividido por	Janela de Tempo de Aquisição	Imped. CPS=0	Imped. CPS=1
0	0	0	0	1	2*Tclk	53	291
0	0	0	1	2	4*Tclk	291	767
0	0	1	0	3	6*Tclk	529	1244
0	0	1	1	4	8*Tclk	767	1720
0	1	0	0	5	10*Tclk	1005	2196
0	1	0	1	6	12*Tclk	1244	2672
0	1	1	0	7	14*Tclk	1482	3148
0	1	1	1	8	16*Tclk	1720	3625
1	0	0	0	9	18*Tclk	1958	4101
1	0	0	1	10	20*Tclk	2196	4577
1	0	1	0	11	22*Tclk	2434	5053
1	0	1	1	12	24*Tclk	2672	5529
1	1	0	0	13	26*Tclk	2910	6005
1	1	0	1	14	28*Tclk	3148	6482
1	1	1	0	15	30*Tclk	3386	6958
1	1	1	1	16	32*Tclk	3625	7434

OBS: O período Tclk depende do bit CPS do **ADCTRL1**. Se CPS for '0', Tclk = 1/CLK. Se CPS for '1', Tclk = 2*(1/CLK)

- **Bit 7 - CPS** - *Prescale* do *clock* de conversão.

  0 - Fclk = CLK.

  1 - Fclk = CLK/2.

- **Bit 6 - CONT RUN** - Este *bit* determina se o sequenciador opera no modo de conversão contínuo ou no modo *start/stop*.

  0 - Modo *start/stop*. O sequenciador pára no final do sequenciamento.

  1 - Modo de conversão contínuo. Após atingir o final do sequenciamento, o sequenciador inicia novamente do estado CONV00 no SEQ1 e SEQ, e para o estado CONV08 no SEQ2.

- **Bit 5 - INT PRI** - Indica a prioridade de requisição da interrupção do ADC.

  0 - Prioridade alta.

  1 - Prioridade baixa.

- **Bit 4 - SEQ CASC** - Indica o modo do sequenciador.

  0 - Modo dual (SEQ1 e SEQ2).

  1 - Modo cascata (SEQ).

- **Bit 3 - CAL ENA** - Habilita o *offset* de calibração. Quando for '1', o canal de entrada multiplexado será desabilitado e a referência de calibração será conectada. A conversão da calibração pode então ser iniciada setando o *bit* STRT CAL do registrador **ADCTRL2**. Vela que o *bit* CAL ENA deverá ser setado antes do *bit* STRT CAL poder ser usado. O *bit* CAL ENA não deverá ser setado em '1', se STEST ENA for igual a '1'.

  0 - Desabilita o modo de calibração.

  1 - Habilita o modo de calibração.

- **Bit 2 - BRG ENA** - Habilita a ponte. Juntamente com o *bit* HI/LO, BRG ENA permite que uma tensão de referência seja convertida no modo de calibração.

  0 - Toda tensão de referência será aplicada na entrada do ADC.

  1 - A metade da tensão de referência será aplicada na entrada do ADC.

- **Bit 1 - HI/LO** - Seleciona Vrefhi/Vreflo. Quando o modo *self - test* for habilitado (STEST ENA = 1), o *bit* HI/LO definirá a polaridade da fonte de referência. No modo normal de operação, o *bit* HI/LO não tem efeito. Na tabela 8.4 está indicado as possíveis tensões de referência definidas pelos *bits* BRG ENA e HI/LO.

  0 - Vreflo é usado na entrada do ADC.

  1 - Vrefhi é usado na entrada do ADC.

Tabela 8.4: Possíveis tensões de referência definidas pelos *bits* BRG ENA e HI/LO.

BRG ENA	HI/LO	**CAL ENA = 1 Referência**	**STEST ENA = 1 Referência**
0	0	Vreflo	Vreflo
0	1	Vrefhi	Vrefhi
1	0	(Vrefhi - Vreflo)/2	Vreflo
1	1	(Vreflo - Vrefhi)/2	Vrefhi

- **Bit 0 - STEST ENA** - Habilita/desabilita a função *self-test*.

  0 - Desabilita o modo *self-test*.

  1 - Habilita o modo *self-test*.

8.2.9 Registrador de Controle do ADC 2 - ADCTRL2

Na figura 8.9 está ilustrado o esquema com as posições dos *bits* no registrador **ADCTRL2**.

15	14	13	12	11	10	9	8
EVB SOC SEQ	RST SEQ1/ STRT CAL	SOC SEQ1	SEQ1 BSY	INT ENA SEQ1 (Mode 1)	INT ENA SEQ1 (Mode 0)	INT FLAG SEQ1	EVA SOC SEQ1
RW-0	RS-0	RW-0	R-0	RW-0	RW-0	RC-0	RW-0
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
EXT SOC SEQ1	RST SEQ2	SOC SEQ2	SEQ2 BSY	INT ENA SEQ2 (Mode 1)	INT ENA SEQ2 (Mode 0)	INT FLAG SEQ2	EVB SOC SEQ2
RW-0	RS-0	RW-0	R-0	RW-0	RW-0	RC-0	RW-0

Figura 8.9: Esquema com as posições dos *bits* no registrador ADCTRL2.

OBS: **R** = acesso a leitura, **S** = pode apenas ser setado, **C** = pode apenas ser zerado, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bit 15 - EVB SOC SEQ** - Habilita o EVB SOC para o sequenciador cascateado. Este *bit* é ativo apenas no modo cascata.

  0 - Sem ação.

  1 - Permite que o sequenciador cascateado seja gatilhado por um sinal do EVB.

- **Bit 14 - RST SEQ1/STRT CAL** - Reseta SEQ1/ inicia calibração.

  0 - Sem ação.

  1 - Se a calibração estiver desabilitada (*bit* 3 do **ADCTRL1** = 0) o sequenciador SEQ será imediatamente resetado para o estado CONV00. Se a calibração estiver habilitada (*bit* 3 do **ADCTRL1** = 1) o processo de calibração será iniciado imediatamente.

- **Bit 13 - SOC SEQ1** - Gatilho do sequenciador SEQ1. Este bit pode ser setado pelos seguintes gatilhos:

  - Por *software*, escrevendo '1' nesse *bit*;
  - Pelo EVA;
  - Pelo EVB;
  - Pelo pino externo ADCSOC.

  Existem três possibilidades, quando ocorrer um gatilhamento:

  - SEQ1 está livre e o *bit* SOC SEQ1 está zerado. Nesse caso, SEQ1 será iniciado imediatamente. Esse *bit* será setado e logo em seguida será zerado.
  - SEQ1 está ocupado e o *bit* SOC SEQ1 está zerado. Nesse caso, o *bit* SOC SEQ1 será setado, indicando que existe uma requisição de gatilhamento pendente. Quando SEQ1 finalmente iniciar a conversão, após ter finalizado a atual conversão, esse *bit* será zerado.
  - SEQ1 está ocupado e o *bit* SOC SEQ1 está setado. Neste caso, a ocorrência de qualquer gatilho será ignorada.

  0 - Nenhum requisição de gatilhamento está pendente.

  1 - Existe uma requisição de gatilhamento pendente.

  OBS: Os *bits* RST SEQ1 e SOC SEQ1 do **ADCTRL2** não deverão ser setados na mesma instrução. Isto resetará o sequenciador, mas não iniciará a sequência. Deve-se primeiro setar o *bit* RST SEQ1 e em seguida o *bit* SOC SEQ1. Isto também se aplica aos *bits* RST SEQ2 e SOC SEQ2.

- **Bit 12 - SEQ1 BSY** - SEQ1 ocupado. Durante o progresso de uma sequência de autoconversão desempenhada pelo SEQ1, esse *bit* será setado. Ele será zerado quando a sequência de conversão for concluída.

  0 - Sequenciador está livre.

  1 - Sequenciador está ocupado.

- **Bits 11-10 - INT ENA SEQ1** - Habilita o modo de interrupção do SEQ1.

  00 - Interrupção está desabilitada.

  01 - O modo 1 de interrupção está habilitado. Uma interrupção será requisitada imediatamente quando o *flag* INT FLAG SEQ1 for setado.

  10 - O modo 2 de interrupção está habilitado. Uma interrupção será requisitada somente se o *flag* INT FLAG SEQ1 já estiver setado. Se estiver zerado, o *flag* INT FLAG SEQ1 será setado e uma requisição de interrupção será omitida.

  11 - Reservado.

- **Bit 9 - INT FLAG SEQ1** - *Bit* de sinalização de interrupção do ADC para o SEQ1. Este *bit* deve ser zerado escrevendo-se um '1' nele.

  0 - Nenhuma interrupção.

  1 - Ocorreu uma interrupção.

- **Bit 8 - EVA SOC SEQ1** - *Bit* de mascára do início de sequenciamento do EVA para o SEQ1.

  0 - SEQ1 não pode ser iniciado pelo EVA.

  1 - SEQ1/SEQ podem ser iniciados por uma gatilho do EVA.

- **Bit 7 - EXT SOC SEQ1** - *Bit* de início de sequenciamento realizado por um sinal externo para o SEQ1.

  0 - Sem ação.

  1 - Permite que uma sequência de autoconversões seja iniciada pelo pino ADCSOC do dispositivo.

- **Bit 6 - RST SEQ2** - Reseta SEQ2.

  0 - Sem ação.

  1 - O sequenciador SEQ2 será imediatamente resetado para o estado CONV08.

- **Bit 5 - SOC SEQ2** - Gatilho do sequenciador SEQ2. Este bit pode ser setado pelos seguintes gatilhos:

  – Por *software*, escrevendo '1' nesse *bit*;

  – Pelo EVB;

  Existem três possibilidades, quando ocorrer um gatilhamento:

- SEQ2 está livre e o *bit* SOC SEQ2 está zerado. Nesse caso, SEQ2 será iniciado imediatamente. Esse *bit* será setado e logo em seguida será zerado.
- SEQ2 está ocupado e o *bit* SOC SEQ2 está zerado. Nesse caso, o *bit* SOC SEQ2 será setado, indicando que existe uma requisição de gatilhamento pendente. Quando SEQ2 finalmente iniciar a conversão, após ter finalizado a atual conversão, esse *bit* será zerado.
- SEQ2 está ocupado e o *bit* SOC SEQ2 está setado. Neste caso, a ocorrência de qualquer gatilho será ignorada.

0 - Nenhum requisição de gatilhamento está pendente.

1 - Existe uma requisição de gatilhamento pendente.

- **Bit 4 - SEQ2 BSY** - SEQ2 ocupado. Durante o progresso de uma sequência de autoconversão desempenhada pelo SEQ2, esse *bit* será setado. Ele será zerado quando a sequência de conversão for concluída.

  0 - Sequenciador está livre.

  1 - Sequenciador está ocupado.

- **Bits 3-2 - INT ENA SEQ2** - Habilita o modo de interrupção do SEQ2.

  00 - Interrupção está desabilitada.

  01 - O modo 1 de interrupção está habilitado. Uma interrupção será requisitada imediatamente quando o *flag* INT FLAG SEQ2 for setado.

  10 - O modo 2 de interrupção está habilitado. Uma interrupção será requisitada somente se o *flag* INT FLAG SEQ2 já estiver setado. Se estiver zerado, o *flag* INT FLAG SEQ2 será setado e uma requisição de interrupção será omitida.

  11 - Reservado.

- **Bit 1 - INT FLAG SEQ2** - *Bit* de sinalização de interrupção do ADC para o SEQ2. Este *bit* deve ser zerado escrevendo-se um '1' nele.

  0 - Nenhuma interrupção.

  1 - Ocorreu uma interrupção.

- **Bit 0 - EVB SOC SEQ2** - *Bit* de mascára do início de sequenciamento do EVB para o SEQ2.

  0 - SEQ2 não pode ser iniciado pelo EVB.

  1 - SEQ2 podem ser iniciados por uma gatilho do EVB.

8.2.10 Registrador MAXCONV

Na figura 8.10 está ilustrado o esquema com as posições dos *bits* no registrador **MAXCONV**.

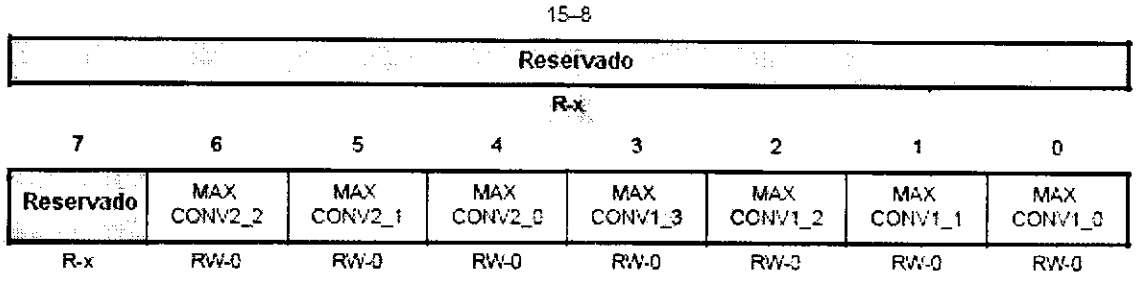


Figura 8.10: Esquema com as posições dos *bits* no registrador MAXCONV.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bit 6-0 - MAX CONVn** - O campo de *bits* MAX CONVn define o número máximo de conversões a serem executadas em uma sessão de autoconversão. O campo de *bits* e as operações variam de acordo com o modo do sequenciador (dual ou cascateado).
  - Para SEQ1, os *bits* MAX CONV1_2 à MAX CONV1_0 são usados.
  - Para SEQ2, os *bits* MAX CONV2_2 à MAX CONV2_0 são usados.
  - Para SEQ, os *bits* MAX CONV1_3 à MAX CONV1_0 são usados.

Uma sessão de autoconversão sempre inicia com o estado inicial e continua sequencialmente até o estado final, se permitido. O *buffer* do resultado é enfileirado em uma ordem sequencial. Algum número de conversões entre 1 e (MAX CONVn + 1) pode ser programado para uma sessão.

Se apenas cinco conversões são requisitadas, o campo MAX CONVn deverá ser setado em '4'. Se o sequenciador SEQ1 ou SEQ forem usados, o sequenciador vai de CONV00 à CONV04, e o resultado das cinco conversões são armazenadas em **RESULT00** à **RESULT04**. Se o sequenciador SEQ2 for usado, o sequenciador vai de CONV08 à CONV12, e o resultado das cinco conversões são armazenadas em **RESULT08** à **RESULT12**. Na tabela 8.5 está indicado os possíveis valores de MAX CONV1.3-0 e seus respectivos números de conversões.

Tabela 8.5: Possíveis valores de MAX CONV1.3-0 e seus respectivos números de conversões.

MAX CONV1.3-0	Número de Conversões
0000	1
0001	2
0010	3
0011	4
0100	5
0101	6
0110	7
0111	8
1000	9
1001	10
1010	11
1011	12
1100	13
1101	14
1110	15
1111	16

8.2.11 Registrador AUTO_SEQ_SR

Na figura 8.11 está ilustrado o esquema com as posições dos *bits* no registrador **AUTO_SEQ_SR**.

15–12				11	10	9	8
Reservado				SEQ CNTR 3	SEQ CNTR 2	SEQ CNTR 1	SEQ CNTR 0
R-x				R-0	R-0	R-0	R-0
7	6	5	4	3	2	1	0
Reserved	SEQ2-State2	SEQ2-State1	SEQ2-State0	SEQ1-State3	SEQ1-State2	SEQ1-State1	SEQ1-State0
R-x	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Figura 8.11: Esquema com as posições dos *bits* no registrador AUTO_SEQ_SR.

OBS: **R** = acesso a leitura, **-n** = valor após o *reset*.

- **Bits 11-8 - SEQ CNTR3 - SEQ CNTR0** - *Bits* de *status* do contador de sequenciamento. Esses *bits* são usados pelo SEQ, SEQ1 e SEQ2. No início de uma sessão de autosequenciamento, SEQ CNTRn é carregado com o valor de MAX CONVn. Os *bits* podem ser lidos em qualquer instante durante o processo de contagem decrescente para verificar o *status* do sequenciador. Este valor, junto com os *bits* SEQ1 BUSY e SEQ2 BUSY, identificam unicamente o progresso ou estado do sequenciador que está ativo.

- **Bits 6-4 - SEQ2-State2 - SEQ3-State0** - Esses *bits* refletem o estado do sequenciador (SEQ2) em qualquer momento. Se for necessário, o usuário pode receber esses *bits* para ler resultados interinos, antes do final da autoconversão. Esses *bits* são irrelevantes no modo cascateado.

- **Bits 3-0 - SEQ1-State3 - SEQ1-State0** - Esses *bits* refletem o estado do sequenciador (SEQ1) em qualquer momento. Se for necessário, o usuário pode receber esses *bits* para ler resultados interinos, antes do final da autoconversão. Esses *bits* são usados também no modo cascateado.

8.2.12 Registradores CHSELSEQn

Na figura 8.12 está ilustrado o esquema com as posições dos *bits* no registrador **CHSELSEQn.**

Bits 15–12	Bits 11–8	Bits 7–4	Bits 3–0	
CONV03	CONV02	CONV01	CONV00	**CHSELSEQ1**
RW-0	RW-0	RW-0	RW-0	
Bits 15–12	**Bits 11–8**	**Bits 7–4**	**Bits 3–0**	
CONV07	CONV06	CONV05	CONV04	**CHSELSEQ2**
RW-0	RW-0	RW-0	RW-0	
Bits 15–12	**Bits 11–8**	**Bits 7–4**	**Bits 3–0**	
CONV11	CONV10	CONV09	CONV08	**CHSELSEQ3**
RW-0	RW-0	RW-0	RW-0	
Bits 15–12	**Bits 11–8**	**Bits 7–4**	**Bits 3–0**	
CONV15	CONV14	CONV13	CONV12	**CHSELSEQ4**
RW-0	RW-0	RW-0	RW-0	

Figura 8.12: Esquema com as posições dos *bits* no registrador CHSELSEQn.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset.*

Cada um dos campos de quatro *bits* (CONVnn) selecionam uma das dezesseis entradas analógicas do ADC para uma sessão de autosequenciamento. Na tabela 8.6 está indicado os possíveis valores dos campos CONVnn com os respectivos canais de entrada selecionados.

Tabela 8.6: Possíveis valores dos campos CONVnn com os respectivos canais de entrada selecionados.

CONVnn	Canal selecionado
0000	Canal 0
0001	Canal 1
0010	Canal 2
0011	Canal 3
0100	Canal 4
0101	Canal 5
0110	Canal 6
0111	Canal 7
1000	Canal 8
1001	Canal 9
1010	Canal 10
1011	Canal 11
1100	Canal 12
1101	Canal 13
1110	Canal 14
1111	Canal 15

8.2.13 Registradores RESULTn

Na figura 8.13 está ilustrado o esquema com as posições dos *bits* no registrador **RESULTn**.

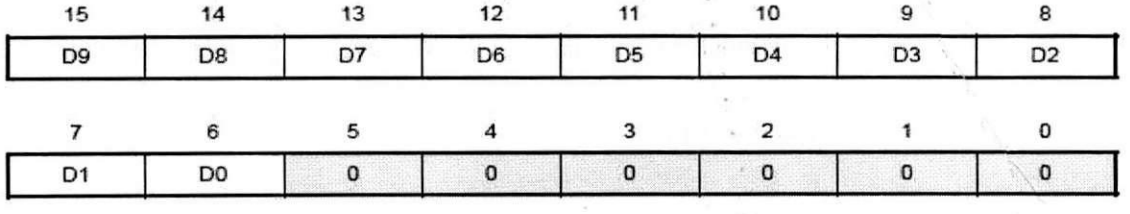


Figura 8.13: Esquema com as posições dos *bits* no registrador RESULTn.

O resultado da conversão de 10 *bits* é armazenado nos *bits* D9 - D0.

8.2.14 Ciclo de Clock do ADC

O tempo de conversão é uma função do número de conversões desempenhadas em uma dada sequência. O ciclo de conversão pode ser dividido em cinco fases:

1. Sincronização do início da sequência **(SOS synch)**;
2. Tempo de aquisição **(ACQ)**;
3. Tempo de conversão **(CONV)**;
4. Ciclo do fim de conversão **(EOC)**;
5. Fim da sequência **(EOS)**.

O SOS synch é aplicado apenas na primeira conversão de uma sequência. O ACQ, CONV e o EOC são aplicados em todas as conversões de uma sequência. O EOS é aplicado apenas na última conversão de uma sequência. Cada fase e seu respectivo número de ciclos do CLKOUT estão listados na tabela 8.7.

Tabela 8.7: Número de ciclos de CLKOUT para cada fase do ciclo de conversão.

Fase	Ciclos de CLKOUT (CPS=0)	Ciclos de CLKOUT (CPS=1)
SOS synch	2	2 ou 3 *
ACQ	2 **	4 **
CONV	10	20
EOC	1	2
EOS	1	1

* Quando o *bit* CPS do registrador **ADCTRL1** for '1', um início de sequência pode ter um ciclo de CLKOUT extra para sincronizar com o *clock* do ADC (ADCCLK), dependendo de qual ciclo o bit SOC SEQn do registrador **ADCTRL2** for setado por *software*.

** O valor de ACQ é dependente dos *bits* ACQ PSn do registrador **ADCTRL1**. Os valores indicados na tabela 8.7 são aplicáveis quando ACQ PS for '0'. Na tabela 8.8 estão indicados valores de ACQ para ACQ PS = 1, 2 e 3. Os valores dessa tabela podem ser extrapolados para todos os possíveis valores de ACQ PS.

Tabela 8.8: Possíveis valores de ACQ para ACQ PS = 1, 2 e 3.

ACQ PS	**(CPS=0)**	**(CPS=1)**
1	ACQ=4	ACQ=8
2	ACQ=6	ACQ=12
3	ACQ=8	ACQ=16

Capítulo 9

Interface de Comunicação Serial - SCI

9.1 Introdução

Este capítulo descreve a arquitetura, funções e programação da *interface* de comunicação serial (SCI). A SCI suporta a comunicação digital serial assíncrona UART (*Universal Asynchronous Receiver/Transmitter*) entre A CPU e outro periféricos assíncronos que usam o formato padrão NRZ (*Non-Return to Zero*).

9.2 Descrição Física da SCI

O módulo SCI possui as seguintes características:

- Dois pinos de I/O: o pino de recepção de dados SCIRXD e o pino de transmissão de dados SCITXD;
- Taxa de transmissão programável;
- Comprimento da palavra de dados programável de um à oito *bits*;
- número de *bits* de stop programável em um ou dois;
- *Clock* gerado internamente;
- Quatro *flags* de detecção de erro;
- Dois modos *wake-up*;
- Operação *half-* ou *full - duplex*;

- Duplo armazenamento nas funções de transmissão e recepção;
- Um transmissor e um receptor podem ser operados por interrupções ou por *polling* do *status* dos *flags*;
- Formato NRZ.

Na figura 9.1 está ilustrado o diagrama de blocos do módulo SCI. Observe que todos os registradores são de oito *bits*.

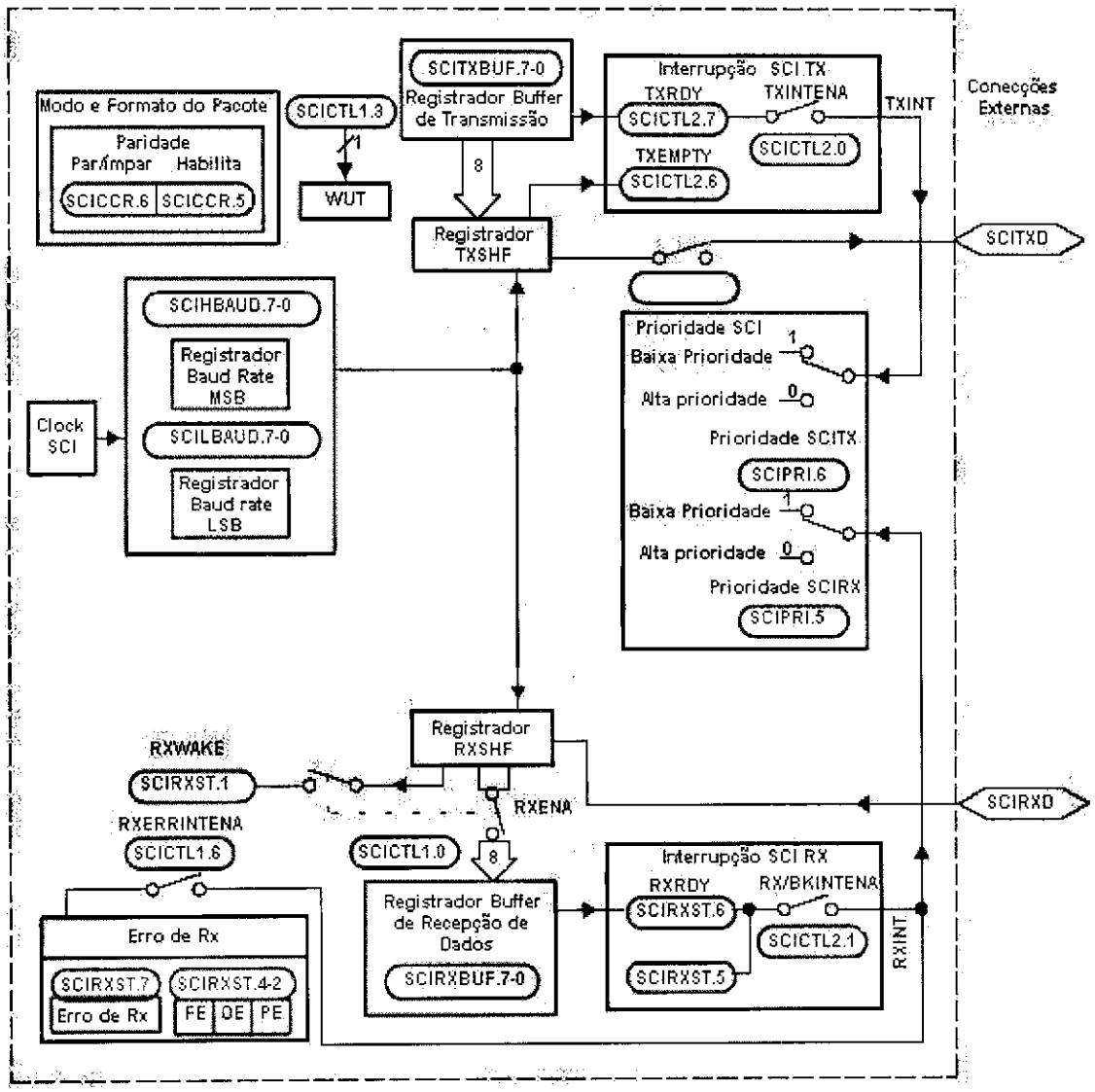


Figura 9.1: Diagrama de blocos do módulo SCI.

9.3 Formato dos Dados na SCI

Os dados na SCI estão no formato NRZ (*Non-Return to Zero*). Um esquema com o formato de dados NRZ está ilustrado na figura 9.2. Esse formato consiste de:

- Um *bit* de *start*;
- Um à oito *bits* de dados;
- Um *bit* de paridade par ou ímpar (opcional);
- Um ou dois *bits* de *stop*;
- Um *bit* extra para distinguir entre endereços de dados (apenas no modo *address-bit*).

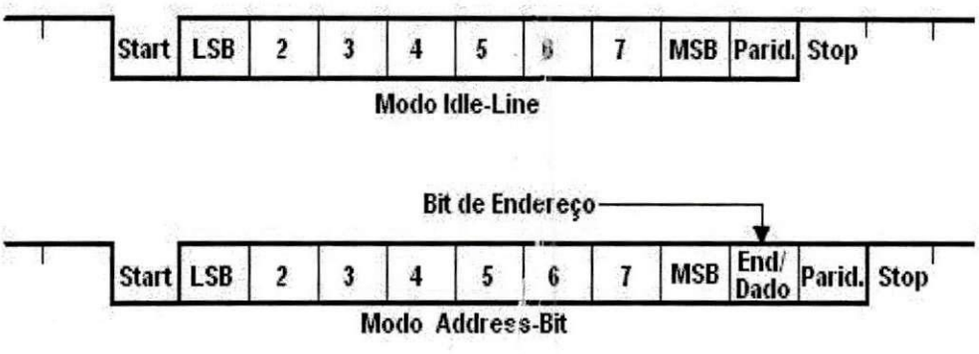


Figura 9.2: Esquema com o formato de dados NRZ.

A unidade básica de dados é chamada de caracter, e possui um comprimento de um à oito *bits*. Cada caracter de dados é formado por um *bit* de *start*, um ou dois *bits* de *stop* e *bits* de paridade e endereço. Um caracter de dados com essas informações é chamado de pacote (mensagem). Para programar o formato do dado, utiliza-se o registrador **SCICCR**.

9.4 Comunicação Multiprocessador da SCI

O formato de comunicação multiprocessador permite que um processador enviar de forma eficiente blocos de dados para outros processadores que estejam no mesma linha serial. Em uma linha serial, poderá ter apenas uma transferência por vez. Existem dois modos multiprocessadores: o modo *idle-line* e o modo *address-bit*.

9.4.1 *Byte* de Endereço

No primeiro *byte* de um bloco de informação que o transmissor envia, contém *byte* de endereço que é lido pelos receptores. Apenas os receptores com o endereço correto podem ser interrompidos pelo *byte* de dados que segue o *byte* de endereço. Os receptores com um endereço incorreto não são interrompidos até o próximo *byte* de endereço.

9.4.2 *Bit Sleep*

Todos os processadores em uma linha serial setam seus *bits SLEEP* no registrador **SCICTL** em '1' de modo que eles são interrompidos apenas quando o endereço de *byte* for detectado. Quando um processador ler um bloco de endereço que corresponde ao endereço da CPU do dispositivo que foi setado por *software*, seu programa deverá zerar o *bit SLEEP* para habilitar a SCI gerar uma interrupção na recepção de cada *byte*.

Embora o receptor permaneça a operar quando o *bit SLEEP* for '1', ele não seta RXRDY, RXINT, ou algum *bit* de *status* de erro de recepção em '1', ao menos que o *byte* de endereço que foi detectado e o *bit* de endereço no pacote recebido é um '1' (aplicável no modo *address-bit*). A SCI não altera o *bit SLEEP*, o *software* do usuário deverá alterar o *bit SLEEP*.

9.4.3 Reconhecimento do *Byte* de Endereço

Um processador reconhece o *byte* de endereço diferentemente, dependendo do modo multiprocessador que estiver usando. No modo *idle-line* não existe um *bit* de endereço/dados extra e ele é mais eficiente que o modo *address-bit* para manipular blocos que contém mais que dez *bytes* de dados. No modo *address-bit* adiciona-se um *bit* extra entre todo *byte*, para distinguir endereços de dados. Este modo é mais eficinte para manipular pequenos blocos de dados.

9.4.4 Sequência em uma Recepção

Nos dois modos de comunicação, a sequência de recepção será:

1. Na recepção de um bloco de endereço, a porta SCI "acorda"e requisita uma interrupção (o *bit* RX/BK INT ENA do registrador **SCICTL** deverá ser habilitado para requisitar uma interrupção). Ele ler o primeiro pacote do bloco, no qual contém o endereço do destinatário;

2. Uma rotina do *software* será executada através da interrupção e verificará o endereço que chegou. Este *byte* de endereço é verificado novamente com o *byte* de endereço do dispositivo armazenado na sua memória;

3. Se o *byte* de endereço for igual ao da CPU do dispositivo, a CPU zera o *bit SLEEP* e ler o resto do bloco. Caso contrário, a rotina do *software* sairá com o *bit SLEEP* setado e não ocorrerá interrupção de recepção até o próximo bloco de dados iniciar.

9.4.5 Modo *idle-line*

Nesse modo os blocos são separados por um tempo livre de 10 ou mais *bits* de nível alto. Esses *bits* indicam o início de um novo bloco. Na figura 9.3 ilústram um esquema com o formato de um pacote no modo *idle-line*.

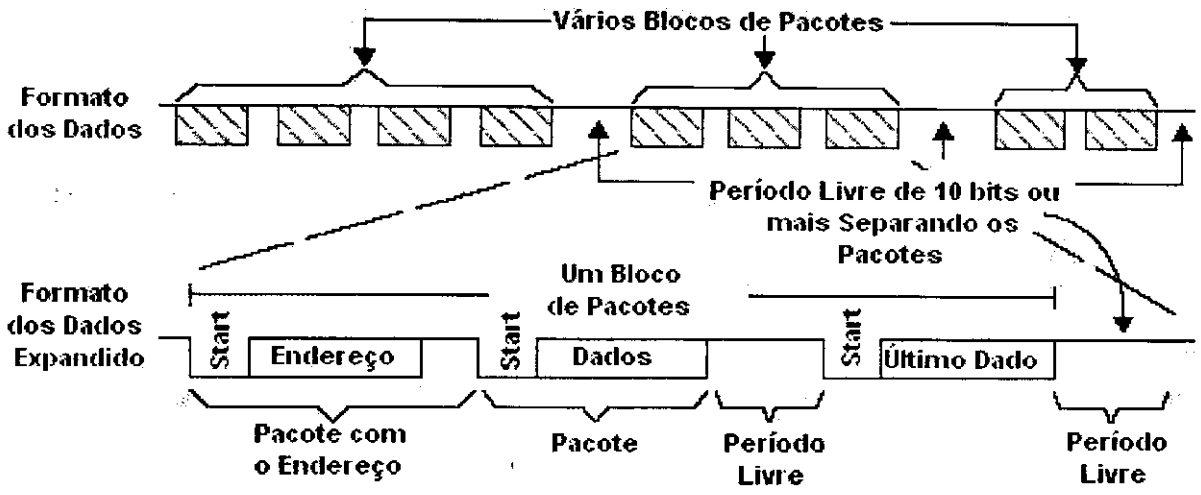


Figura 9.3: Esquema com o formato de um pacote no modo *idle-line*.

Os seguintes passos ocorrem em uma recepção no modo *idle-line*:

1. A SCI "acorda"após receber o sinal de início de bloco;
2. O processador reconhece a próxima interrupão SCI;
3. A rotina de serviço de interrupção compara o endereço recebido;
4. Se for igual ao endereço da CPU, a rotina de serviço de interrupção zera o *bit SLEEP* e recebe o restante do bloco de dados;
5. Se não for igual ao do endereço da CPU, o *bit SLEEP* permanece setado e A CPU continua executado o programa principal.

9.4.6 Sinal de Início de Bloco

Existem dois métodos de enviar um sinal de início de bloco:

- Método 1: Um tempo livre de 10 ou mais *bits* é inserido na linha de comunicação entre o último pacote de dados do bloco anterior e o pacote de endereço do novo bloco;
- Método 2: A porta SCI primeiro seta o *bit* TXWAKE do registrador **SCICTL** em '1' antes de escrever no registrador **SCITXBUF**. Este envia um tempo livre de exatamente 11 *bits*.

Para enviar um sinal de início de bloco, os seguintes passos devem ser desempenhados:

1. Setar em '1' o *bit* TXWAKE;
2. Escrever uma palavra de dados no registrador **SCITXBUF** para enviar o sinal de início do bloco. Quando o registrador **TXSHF** estiver livre novamente, o conteúdo do registrador **SCITXBUF** será deslocado para **TXSHF**, o valor de TXWAKE é deslocado para o *flag* WUT, e então TXWAKE será zerado. Devido TXWAKE ter sido setado em '1', os *bits* de *start*, dados e paridade serão substituídos por um período livre de 11 *bits*, que serão transmitidos após o *bit* de *stop* do pacote anterior.
3. Escrever um novo valor de endereço no registrador **SCITXBUF**.

9.4.7 Modo Address-Bit

Neste modo, os pacotes possuem um *bit* extra chamado de *bit* de endereço que segue imediatamente o último *bit* de dados. O *bit* de endereço é setado em '1' no primeiro pacote do bloco e em '0' nos pacotes restantes. O período de tempo livre é irrelevante.

O valor do *bit* TXWAKE é colocado no *bit* de endereço. Durante a transmissão, quando o registrador **SCITXBUF** e o *bit* TXWAKE são armazenados no registrador **TXSHF** e no *bit* WUT, respectivamente, TXWAKE é resetado e WUT terá o valor do *bit* de endereço do atual pacote. Deste modo, para enviar um endereço, deve-se:

1. Setar o *bit* TXWAKE em '1' e escrever o apropriado valor do endereço no registrador **SCITXBUF**. Quando este valor de endereço for transferido para o registrador **TXSHF** e enviado para a linha de comunicação, seu *bit* de endereço será enviado com o valor '1'. Isto sinaliza aos outros processadores da linha serial para ler o endereço;
2. Escrever em **SCITXBUF** e TXWAKE após **TXSHF** e WUT serem carregados.
3. Deixar o *bit* TXWAKE zerado para transmitir pacotes sem endereço no bloco.

O formato *address-bit* é usado em pacotes de dados de 11 *bytes* ou menos. Este formato adiciona um *bit* ('1' em pacotes de endereço, e '0' em pacotes de dados) para todos os *bytes* de dados transmitidos. O formato *idle-line* é usado em pacotes de dados de 12 *bytes* ou mais. Na figura 9.4 está ilustrado um esquema com o formato de um pacote de dados no modo *address-bit*.

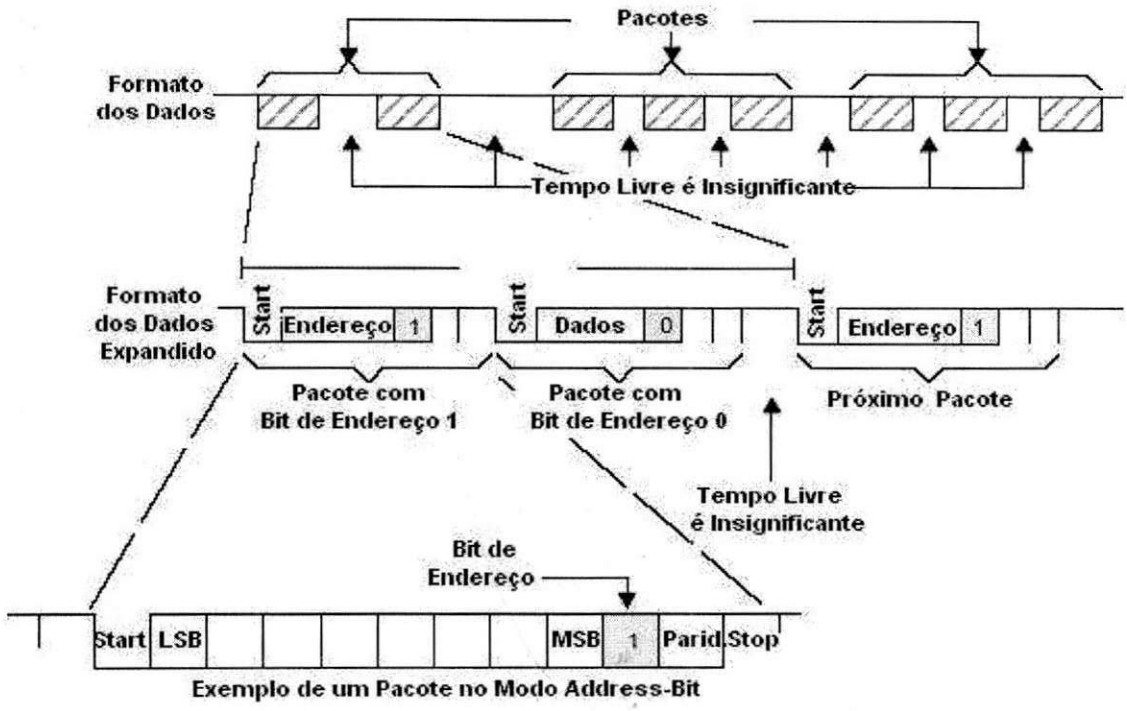


Figura 9.4: Esquema com o formato de um pacote de dados no modo *address-bit*.

9.5 Formato da Comunicação SCI

O formato de comunicação assíncrono SCI utiliza uma única linha ou as duas linhas na comunicação. Neste modo o pacote consiste de um *bit* de *start*, de um à oito *bytes* de dados, um opcional *bit* de paridade par/ímpar, e um ou dois *bits* de *stop*. Existem oito períodos de SCICLK por *bit* de dados. Na figura 9.5 está ilustrado um esquema com o formato de um pacote no formato de comunicação SCI.

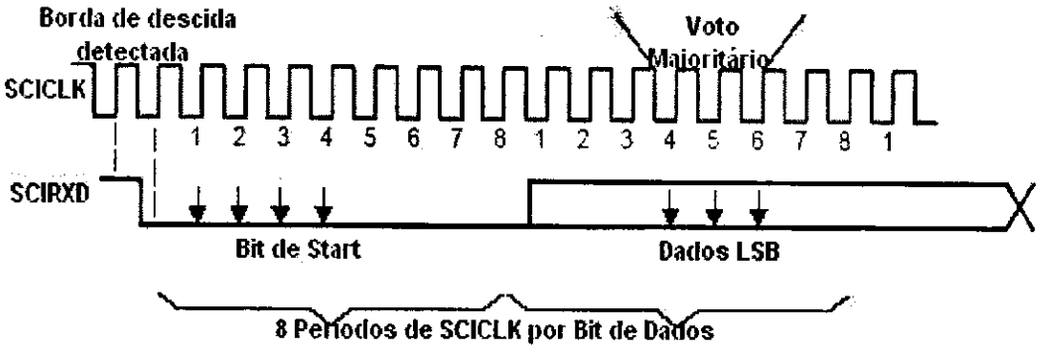


Figura 9.5: Esquema com o formato de um pacote no formato de comunicação SCI.

O receptor inicia a operação de recepção de um *bit* de *start* válido. Um *bit* de *start* válido é identificado por quatro consecutivos períodos SCICLK de *bits* com valor '0', como ilustrado na figura 9.5. Se algum *bit* não for zero, o processador desiste da recepção e começa a observar outros *bits* de *start*.

Para os *bits* que seguem o *bit* de *start*, o processador determina o valor do *bit* fazendo três amostras no meio dos *bits*. Estes amostras ocorrem no quarto, quinto e sexto período do SCICLK, e o valor do *bit* é determinado de forma majoritária (dois dos três) Esse processo está ilustrado na figura 9.5.

Desde que o receptor se sincroniza com os pacotes, os dispositivos transmissores e receptores não usam um *clock* serial sincronizado. O *clock* pode ser gerado localmente.

9.5.1 Sinais do Receptor nos Modos de Comunicação

Na figura 9.6 está ilustrado um esquema com os sinais em uma recepção. Assume-se as seguintes condições:

- Modo *wake-up address-bit*;
- Seis *bits* por caracter.

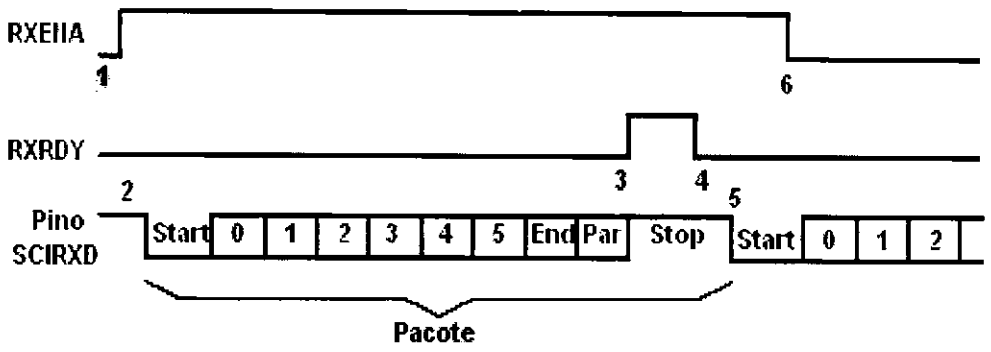


Figura 9.6: Esquema com os sinais em uma recepção SCI.

No **ponto 1**, o *bit* RXENA do registrador **SCICTL1** está em nível alto para habilitar a recepção.

No **ponto 2**, os dados chegam no pino SCIRXD, e o *bit* de *start* é detectado.

No **ponto 3**, os dados são deslocados de **RXSHF** para o *buffer* de recepção **SCIRXBUF**. Uma interrupção será requisitada. O *flag* RXRDY do registrador **SCIRXST** vai para nível alto para indicar que um novo caracter será recebido.

No **ponto 4**, o programa ler **SCIRXBUF** e o *flag* RXRDY é automaticamente zerado.

No **ponto 5**, o próximo *byte* de dados chega no pino SCIRXD e o *bit* de *start* é detectado.

No **ponto 6**, o *bit* RXENA é colocado em nível baixo para desabilitar o receptor. Os dados continuam chegando, mas não são transferidos para o *buffer* de recepção.

9.5.2 Sinais do Transmissor nos Modos de Comunicação

Na figura 9.7 está ilustrado um esquema com os sinais em uma transmissão. Assume-se as seguintes condições:

- Modo *wake-up address-bit*;
- Três *bits* por caracter.

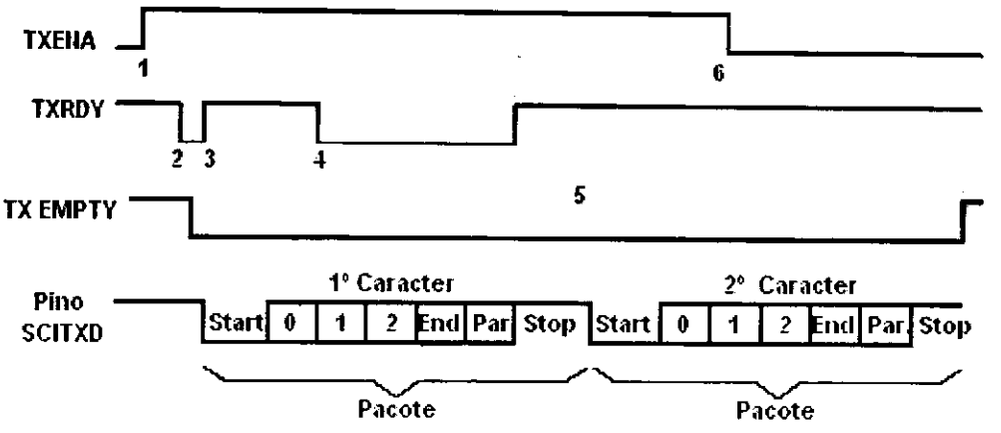


Figura 9.7: Esquema com os sinais em uma transmissão SCI.

No **ponto 1**, o *bit* TXENA do registrador **SCICTL1** está em nível alto para habilitar o transmissor a enviar dados.

No **ponto 2**, **SCITXBUF** armazena os dados a serem enviados. Deste modo, o transmissor não está mais vazio, e TXRDY vai para nível baixo.

No **ponto 3**, os dados são deslocados para **TXSHF**. O transmissor está pronto para um segundo caracter (TXRDY vai para nível alto), e é requisitado uma interrupção (para habilitar uma interrupção o *bit* TX INT ENA do registrador **SCICTL2** deverá ser setado).

No **ponto 4**, o programa escreve um segundo caracter no **SCITXBUF** após TXRDY ir para nível alto (item 3). TXRDY vai para nível baixo novamente após o segundo caracter ser escrito em **SCITXBUF**.

No **ponto 5**, a transmissão do primeiro caracter é concluída. Inicia a transferência do segundo caracter para o registrador de deslocamento **TXSHF**.

No **ponto 6**, o *bit* TXENA é colocado em nível baixo para desabilitar o transmissor. A SCI finaliza a transmissão a transmissão do atual caracter.

No **ponto 7**, a transmissão do segundo caracter é concluída. O transmissor está vazio e pronto para um novo caracter.

9.6 Interrupções da Porta SCI

O receptor e o transmissor podem ser controlados por interrupção. O registrador **SCICTL2** possui um *bit* (TXRDY) que indica condições de interrupção, e o registrador **SCIRXST** possui dois *bits* de sinalização de interrupção (RXRDY e BRKDT), mais o *flag* de interrupção de RX ERROR. O transmissor e o receptor possuem separados *bits* para habilitar as interrupções. Quando não habilitados, as interrupções não são asseguradas. Porém, os *flag* permanecem ativos, refletindo o *status* da transmissão e da recepção.

A SCI possui independentes vetores de interrupção do periférico para a recepção e a transmissão. As interrupções SCI podem ser programadas para assegurar níveis de alta e baixa prioridade através dos *bits* SCIRX PRIORITY e SCITX PRIORITY do registrador **SCIPRI**. Quando ambas requisições de interrupção de recepção e transmissão são programadas para terem o mesmo nível de prioridade, o receptor terá prioridade na ocorrência de uma requisição de transmissão e recepção no mesmo momento.

Se o *bit* RX/BK INT ENA do registrador **SCICTL2** estiver setado, a requisição de interrupção de recepção do periférico será assegurada quando um dos seguintes eventos ocorrer:

- A SCI receber um pacote completo e transferir os dados do registrador **RXSHF** para o registrador **SCIRXBUF**. Esta ação seta o *flag* RXRDY do registrador **SCIRXST** e inicia uma interrupção;
- Uma condição de detecção de *break* ocorrer (o SCIRXD está em nível baixo por 10 períodos de *bit* seguidos de um ausente *bit* de *stop*). Esta ação seta o *bit* BRKDT do registrador **SCIRXST** e inicia uma interrupção.

Se o *bit* TX INT ENA do registrador **SCICTL2** estiver setado, uma interrupção de transmissão será assegurada quando os dados do registrador **SCITXBUF** forem transferidos para o registrador **TXSHF**, indicando que a CPU pode escrever no **SCITXBUF**. Esta ação seta o *bit* TXRDY do registrador **SCICTL2** e inicia uma interrupção.

9.7 Cálculos do Baud Rate da SCI

O *clock* serial gerado internamente é determinado pela frequência do *clock* do dispositivo (CLKOUT) e pelos registradores de seleção do *baud rate*. Na tabela 9.1 está indicado valores de seleção do *baud rate* para as taxas de *bits* mais usadas, para um CLKOUT de 40 MHz.

Tabela 9.1: Valores de seleção do *baud rate* para as taxas de *bits* mais usadas, para um CLKOUT de 40 MHz.

Baud Rate	BRR
2400	2882 (822h)
4800	1040 (411h)
9600	520 (208h)
19200	259 (103h)
38400	129 (81h)

9.8 Registradores da SCI

9.8.1 Registrador SCICCR

O registrador **SCICCR** define o formato do caracter, o protocolo e o modo de comunicação usado pela SCI. Na figura 9.8 está ilustrado um esquema com as posições dos *bits* no registrador **SCICCR**.

7	6	5	4	3	2	1	0
STOP BITS	EVEN/ODD PARITY	PARITY ENABLE	LOOPBACK ENA	ADDR/IDLE MODE	SCICHAR2	SCICHAR1	SCICHAR0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Figura 9.8: Esquema com as posições dos *bits* no registrador SCICCR.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bit 7 - STOP BITS** - Número de *bits* de *stop*. Este *bit* indica o número de *bits* de *stop* que serão transmitidos. O receptor verificará apenas um *bit* de *stop*.

  0 - Um *bit* de *stop*.

  1 - Dois *bits* de *stop*.

- **Bit 6 - PARITY** - Seleciona a paridade par ou ímpar. Se o *bit* PARITY ENABLE estiver setado, o *bit* PARITY selecionará a paridade.

  0 - Paridade ímpar.

  1 - Paridade par.

- **Bit 5 - PARITY ENABLE** - Habilita o *bit* de paridade da SCI. Se a SCI está no modo *address-bit* (*bit* 3 desse registrador), o *bit* de endereço é incluído no cálculo da paridade. Para caracteres menores que 8 *bits*, os *bits* não usados deverão ser mascarados para o cálculo da paridade.

  0 - Paridade é desabilitada. Nenhum *bit* de paridade será gerado durante uma transmissão ou é esperado durante uma recepção.

  1 - Paridade é habilitada.

- **Bit 4 - LOOP BACK ENA** - Habilita o modo de teste *loop back*. No modo *loop back*, o pino Tx é conectado internamente ao pino Rx.

  0 - Modo de teste *loop back* é desabilitado.

  1 - Modo de teste *loop back* é habilitado.

- **Bit 3 - ADDR/IDLE MODE**. *Bit* de controle do modo multiprocessador da SCI. Este *bit* seleciona um dos protocolos multiprocessadores.

  0 - Seleciona o modo *idle-line*.

  1 - Seleciona o modo *address-bit*.

- **Bits 2-0 - SCI CHAR2-0** - *Bits* de controle do comprimento do caracter. Estes *bits* selecionam o comprimento do caracter de 1 à 8 *bits*. Na tabela 9.2 está indicado os possíveis valores de SCI CHAR e os respectivos valores de comprimento do caracter.

Tabela 9.2: Possíveis valores de SCI CHAR e os respectivos valores de comprimento do caracter.

SCI CHAR2-0	Comprimento do Caracter (Bits)
000	1
001	2
010	3
011	4
100	5
101	6
110	7
111	8

9.8.2 Registrador SCICTL1

O registrador SCICTL1 controla o transmissor/receptor, as funções TXWAKE e SLEEP e o reset por software da SCI. Na figura ?? está ilustrado um esquema com as posições dos bits no registrador SCICTL1.

7	6	5	4	3	2	1	0
Reservado	RX ERR INT ENA	SW RESET	Reservado	TXWAKE	SLEEP	TXENA	RXENA
R-0	RW-0	RW-0	R-0	RS-0	RW-0	RW-0	RW-0

Figura 9.9: Esquema com as posições dos *bits* no registrador SCICTL1.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **S** = pode apenas ser setado, **-n** = valor após o *reset*.

- **Bit 6 - RX ERR INT ENA** - Habilita a interrupção de erro na recepção. A setagem deste *bit* habilita uma interrupção se o *bit* RX ERROR do registrador **SCIRXST** for setado devido uma ocorrência de erro.

  0 - Desabilita uma interrupção de erro de recepção.

  1 - Habilita uma interrupção de erro de recepção.

- **Bit 5 - SW RESET** - Reseta a SCI por *software* (ativo baixo). Escrevendo um '0' neste *bit* inicializa o estado de máquina da SCI e a operação de sinalização para a condição de *reset*. Esse *bit* não a configuração dos *bits*. Após um *reset* do sistema, a SCI será habilitada novamente, escrevendo-se um '1' neste *bit*.

  Este *bit* deve ser zerado após um receptor detectar um *break* (*flag* BRKDT do registrador **SCIRXST**). SW RESET afeta a operação dos *flags* da SCI, mas não afeta a configuração dos *bits*. Na figura 9.3 está indicado os *flags* que são afetados pelo *bit* SW RESET.

Tabela 9.3: *Flags* que são afetados pelo *bit* SW RESET.

Flag	Registrador.Bit	Valor Após SW RESET
TXRDY	SCICTL2.7	1
TX EMPTY	SCICTL2.6	1
RXWAKE	SCIRXST.1	0
PE	SCIRXST.2	0
OE	SCIRXST.3	0
FE	SCIRXST.4	0
BRKDT	SCIRXST.5	0
RXRDY	SCIRXST.6	0
RX ERROR	SCIRXST.7	0

- **Bit 4 - TXWAKE** - Seleciona o método *wake-up* de transmissão da SCI. O *bit* TXWAKE controla a seleção das características de transmissão dos dados, dependendo de qual modo de transmissão (*idle-line* ou *address-bit*) foi específicado no *bit* ADDR/IDLE MODE do registrador **SCICCR**.

  0 - A característica de transmissão não é selecionado.

  1 - A característica de transmissão é selecionado, e depende do modo de operação. No modo *idle-line*, escreve-se os dados no registrador **SCITXBUF** para gerar um período livre de 11 *bits* de dados. No modo endereço-bit, escreve-se os dados no registrador SCITXBUF para setar o bit de endereço deste pacote em '1'.

  TXWAKE não será zerado por um *reset* do sistema (*bit* SW RESET) ou pela transferência de TXWAKE para o *flag* WUT.

- **Bit 2 - SLEEP** - Em uma configuração multiprocessador, este *bit* controla a função *sleep* do receptor. Zerando este *bit* faz com que a SCI deixe o modo *sleep*.

  0 - Desabilita o modo *sleep*.

  1 - Habilita o modo *sleep*.

O receptor não opera quando o *bit* SLEEP está setado. Operações de leitura do *buffer* de recepção ou *status* de erro não são efetuadas, ao menos que o *byte* de endereço seja detectado. O *bit* SLEEP não é zerado quando o endereço de *byte* for detectado.

- **Bit 1 - TXENA** - Habilita o transmissor da SCI. Os dados são transmitidos pelo pino SCITXD somente quando TXENA for setado.

  0 - Desabilita transmissor

  1 - Habilita transmissor.

- **Bit 0 - RXENA** - Habilita o receptor da SCI. Os dados são recebidos pelo pino SCIRXD e são enviados para o registrador de deslocamento do receptor e depois para o *buffer* de recepção. Este *bit* habilita ou desabilita a recepção dos dados no *buffer* de recepção.

  0 - Desabilita receptor

  1 - Habilita receptor.

  Se RXENA estiver zerado, os caracteres recebidos não são mais transferidos para os dois *buffers* de recepção (**SCIRXEMU** e **SCIRXBUF**) e não há mais geração de interrupções de recepção. Porém, o registrador de deslocamento de recepção continua verificando os caracteres que chegam.

9.8.3 Registradores SCIHBAUD e SCILBAUD

Os valores dos registradores **SCIHBAUD** e **SCILBAUD** especificam o *baud rate* da SCI. Nas figuras 9.10 e 9.11 estão ilustrados, respectivamente, os esquemas com as posições dos *bits* dos registradores **SCIHBAUD** e **SCILBAUD**.

15	14	13	12	11	10	9	8
BAUD15 (MSB)	BAUD14	BAUD13	BAUD12	BAUD11	BAUD10	BAUD9	BAUD8
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Figura 9.10: Esquema com as posições dos *bits* no registrador SCIHBAUD.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **S** = pode apenas ser setado, **-n** = valor após o *reset*.

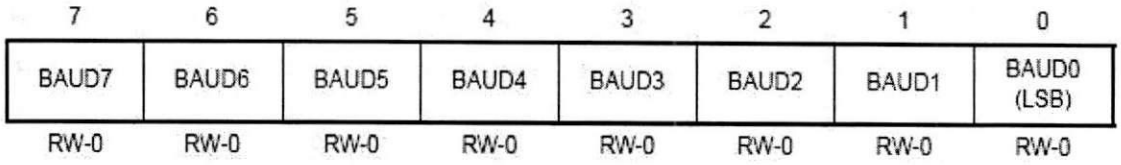


Figura 9.11: Esquema com as posições dos *bits* no registrador SCILBAUD.

- **Bits 15-0 - BAUD15-BAUD0** - Seleciona o *baud rate* da SCI. Os registradores **SCIHBAUD** (*byte* mais significativo) e **SCILBAUD** (*byte* menos significativo) são concatenados para formar um valor de *baud rate* de 16 *bits* (BRR).

  O *baud rate* da SCI é calculado usando a seguinte equação:

$$BAUDRATE = \frac{CLKOUT}{(BRR+1)*8}$$

  alternativamente,

$$BRR = \frac{CLKOUT}{(BAUDRATE*8)-1}$$

  BRR deverá ter um valor entre 1 e 65535. BRR é o valor do registrador concatenado de 16 *bits*.

9.8.4 Registrador SCICTL2

Na figura 9.12 está ilustrado um esquema com as posições dos *bits* no registrador **SCICTL2**.

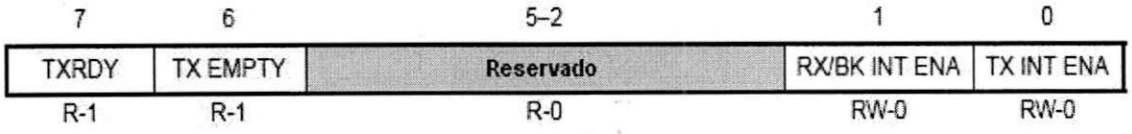


Figura 9.12: Esquema com as posições dos *bits* no registrador SCICTL2.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bit 7 - TXRDY** - *Flag* do *buffer* de transmissão. Quando setado, este *bit* indica que o registrador *buffer* de transmissão de dados **(SCITXBUF)** está pronto para receber outro caracter. Quando se escreve dados no registrador **SCITXBUF**, este *bit* será automaticamente zerado. Quando setado, este *flag* assegura uma requisição de interrupção do transmissor, se o *bit* que habilita a interrupção (TX INT ENA) estiver setado.

  0 - **SCITXBUF** está cheio.

  1 - **SCITXBUF** está pronto para receber um novo caracter.

- **Bit 6 - TX EMPTY** - *Flag* de transmissor vazio.

  0 - O *buffer* de transmissão **(SCITXBUF)** ou o registrador de deslocamento **(TXSHF)** ou ambos estão carregados com dados.

  1 - O *buffer* de transmissão **(SCITXBUF)** e o registrador de deslocamento **(TXSHF)** estão ambos vazios.

- **Bit 1 - TX INT ENA** - Habilita a interrupção do *buffer* de recepção e da detecção de um *break*. Estes *bits* controlam a requisição de interrupção causada pelos *flags* RXRDY e/ou BRKDT.

  0 - Desabilita a interrupção RXRDY/BRKDT.

  1 - Habilita a interrupção RXRDY/BRKDT.

- **Bit 0 - TX INT ENA** - Habilita interrupção do registrador **SCITXBUF**. Este *bit* controla a requisição de interrupção causada pelo *flag* TXRDY.

  0 - Desabilita interrupção TXRDY.

  1 - Habilita interrupção TXRDY.

9.8.5 Registrador SCIRXST

O registrador **SCIRXST** contém sete *bits* que sinalizam o *status* do receptor. Dois do quais podem gerar uma requisição de interrupção. Toda vez que um caracter completo for transferido para os *buffers* de recepção **(SCIRXEMU e SCIRXBUF)**, os *bits* que sinalizam o *status* do receptor são atualizados, e toda vez que os *buffers* são lidos, os *flags* são zerados. Na figura 9.13 está ilustrado um esquema com as posições dos *bits* no registrador **SCIRXST**.

7	6	5	4	3	2	1	0
RX ERROR	RXRDY	BRKDT	FE	OE	PE	RXWAKE	Reservado
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Figura 9.13: Esquema com as posições dos *bits* no registrador SCIRXST.

OBS: **R** = acesso a leitura, **-n** = valor após o *reset*.

- **Bit 7 - RX ERROR** - *Flag* de erro de recepção. Este *bit* indica que um dos *flags* de *status* do receptor (BRKDT, FE, OE e PE) está setado.

  0 - Nenhum *flag* está setado.

  1 - *Flag* de erro está setado.

  Um '1' neste *bit* causará uma interrupção, se o *flag* RX ERR INT ENA do registrador **SCICTL1** estiver setado. Este *flag* de erro não pode ser setado diretamente. Ele é zerado pela escrita de um '0' no *bit* SW RESET ou por um *reset* do sistema.

- **Bit 6 - RXRDY** - *Flag* de recepção. Quando um novo caracter está pronto para ser lido do registrador **SCIRXBUF**, o receptor seta este *bit*, e uma interrupção de recepão poderá ser gerada, se o *bit* RX/BK INT ENA do registrador **SCICTL1** estiver setado. RXRDY é zerado por uma leitura do registrador **SCIRXBUF**, pela escrita de um '0' no *bit* SW RESET ou por um *reset* do sistema,

  0 - Nenhum caracter novo no **SCIRXBUF**.

  1 - Um caracter está pronto para ser lido no **SCIRXBUF**.

- **Bit 5 - BRKDT** - *Flag* de detecção de *break*. A SCI seta este *bit* quando ocorrer uma detecção de *break*. Uma condição de *break* ocorre quando a linha de dados do receptor (SCIRXD) permanecer continuamente em nível baixo nos dez últimos *bits*, iniciando após o primeiro *bit* de *stop*. A ocorrência de um *break* poderá gerar um interrupção do receptor, se o *bit* RX/BK INT ENA estiver setado. Uma interrupção BRKDT pode ocorrer se o *bit* SLEEP do receptor estiver setado. BRKDT é zerado quando se escreve um '0' no *bit* SW RESET ou por um *reset* do sistema. Ele não é zerado por um receptor de um caracter após um *break* ser detectado. De modo a receber mais caracteres, a SCI deverá resetar esse *bit* por intermédio do *bit* SW RESET ou por um *reset* do sistema.

  0 - A condição de um *break* não foi detectada.

  1 - A condição de um *break* foi detectada.

- **Bit 4 - FE** - *Flag* de erro de pacote. Este *bit* será setado quando um *bit* de *stop* não for encontrado. Apenas o primeiro *bit* de parada é verificado. O *flag* FE é resetado quando se escreve um '0' no *bit* SW RESET ou por um *reset* do sistema.

  0 - Nenhum erro de pacote foi detectado.

  1 - Foi detectado um erro de pacote.

- **Bit 3 - OE** - *Flag* de erro de *overrun*. A SCI seta este bit quando um caracter é transferido para os registradores **SCIRXBUF** e **SCIRXEMU** antes do caracter anterior ter sido lido completamente pela CPU. Então, o caracter anterior será sobrescrito e perdido. O *flag* OE é resetado quando se escreve um '0' no *bit* SW RESET ou por um *reset* do sistema.

  0 - Nenhum erro de *overrun* foi detectado.

  1 - Um erro de *overrun* foi detectado.

- **Bit 2 - PE** - *Flag* de erro de paridade. Este *flag* será setado quando um caracter for recebido com um erro de igualdade entre o número de '1s' e seu *bit* de paridade. O *bit* de endereço é incluso no cálculo. Se a geração e a detecção de paridade não estiver habilitada, o *flag* PE será desabilitado e será lido como '0'. O *flag* PE é resetado quando se escreve um '0' no *bit* SW RESET ou por um *reset* do sistema.

  0 - Nenhum erro de paridade ou a paridade está desabilitada.

  1 - Um erro de paridade foi detectado.

- **Bit 1 - RXWAKE** - *Flag* de detecção de wake-up do receptor. Um '1' neste *bit* indica a detecção de uma condição de *wake-up* do receptor. No modo *address-bit*, RXWAKE reflete o valor do *bit* de endereço do caracter contido no **SCIRXBUF**. No modo *idle-line*, RXWAKE é um *flag* apenas de leitura, e é zerado por um dos seguintes eventos:
  - A transferência do primeiro *byte* após o *byte* de endereço para **SCIRXBUF**;
  - A leitura de **SCIRXBUF**;
  - A escrita de um '0' no *bit* SW RESET;
  - Um *reset* do sistema.

9.8.6 Registradores SCIRXEMU e SCIRXBUF

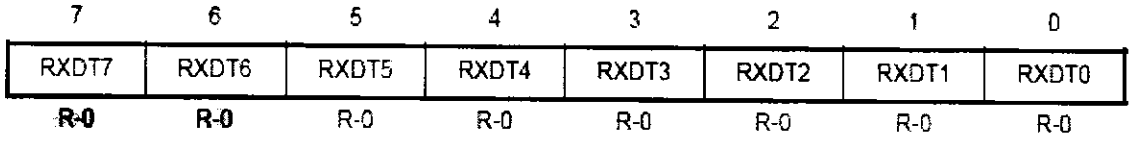
A recepção de dados é transferida de **RXSHF** para **SCIRXEMU** e **SCIRXBUF**. Quando uma transferência for concluída, o *flag* RXRDY será setado, indicando que os dados recebidos estão prontos para serem lidos. Ambos os registradores contém os mesmos dados, eles possuem endereços separados, mas não são separados fisicamente. A única diferença é que a leitura de **SCIRXEMU** não zera o *flag* RXRDY. Porém, a leitura de **SCIRXBUF** zera este *flag*.

O registrador **SCIRXEMU** é usado principalmente pelo emulador, pois ele pode ler continuamente os dados recebidos sem ter que zerar o *flag* RXRDY. Este é o registrador que deverá ser usado em um emulador *watch window* [8] para ver seu conteúdo. **SCIRXEMU** é zerado por um *reset* do sistema. **SCIRXEMU** não é fisicamente implementado, ele é apenas um locação diferente de endereço para acessar o registrador **SCIRXBUF** sem zerar o *flag* RXRDY. Nas figuras 9.14 e 9.15 estão ilustrados, respectivamente, os esquemas com as posições dos *bits* nos registradores **SCIRXEMU** e **SCIRXBUF**.

7	6	5	4	3	2	1	0
ERXDT7	ERXDT6	ERXDT5	ERXDT4	ERXDT3	ERXDT2	ERXDT1	ERXDT0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Figura 9.14: Esquema com as posições dos *bits* no registrador SCIRXEMU.

OBS: **R** = acesso a leitura, **-n** = valor após o *reset*.

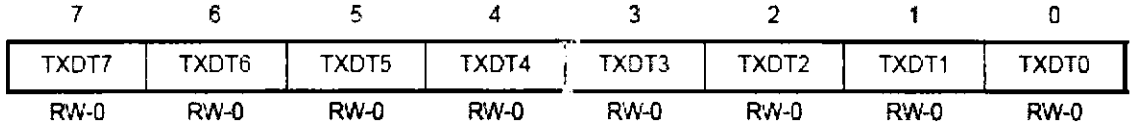


7	6	5	4	3	2	1	0
RXDT7	RXDT6	RXDT5	RXDT4	RXDT3	RXDT2	RXDT1	RXDT0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Figura 9.15: Esquema com as posições dos *bits* no registrador SCIRXBUF.

9.8.7 Registrador SCITXBUF

Os *bits* de dados a serem transmitidos são escritos no registrador **SCITXBUF**. A transferência dos dados desse registrador para o registrador **TXSHF** seta o *flag* TXRDY, indicando que **SCITXBUF** está pronto para receber novos dados. Se o *bit* TX INT ENA estiver setado, esta transferência de dados causará uma interrupção. Na figura 9.16 está ilustrado um esquema com as posições dos *bits* no registrador **SCITXBUF**.



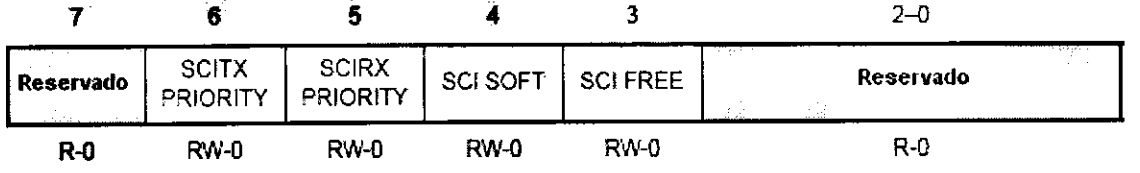
7	6	5	4	3	2	1	0
TXDT7	TXDT6	TXDT5	TXDT4	TXDT3	TXDT2	TXDT1	TXDT0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Figura 9.16: Esquema com as posições dos *bits* no registrador SCITXBUF.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

9.8.8 Registrador SCIPRI

Na figura 9.17 está ilustrado um esquema com as posições dos *bits* no registrador **SCIPRI**.



7	6	5	4	3	2–0
Reservado	SCITX PRIORITY	SCIRX PRIORITY	SCI SOFT	SCI FREE	Reservado
R-0	RW-0	RW-0	RW-0	RW-0	R-0

Figura 9.17: Esquema com as posições dos *bits* no registrador SCIPRI.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bit 6 - SCITX PRIORITY** - Seleciona o nível de prioridade da interrupção do transmissor.

  0 - As requisições de interrupção são de alta prioridade.

  1 - As requisições de interrupção são de baixa prioridade.

- **Bit 5 - SCIRX PRIORITY** - Seleciona o nível de prioridade da interrupção do receptor.

  0 - As requisições de interrupção são de alta prioridade.

  1 - As requisições de interrupção são de baixa prioridade.

- **Bits 4-3 - SCI SOFT, SCI FREE** - Esses *bits* o que ocorrerá quando ocorrer um evento de suspensão da emulação (por exemplo, quando o depurador encontrar um *breakpoint*). No modo *free-run* o periférico pode continuar o que ele está fazendo. Se estiver no modo *soft*, ele pode parar imediatamente ou parar quando a atual operação for concluída.

  00 - Pára imediatamente na suspensão.

  01 - Modo **free-run**. Continua a operação quando ocorrer uma suspensão.

  10 - Termina o que estiver fazendo atualmente antes de parar.

  11 - Modo **free-run**. Continua a operação quando ocorrer uma suspensão.

Capítulo 10

Interface Periférica Serial (SPI)

10.1 Introdução

A *interface* SPI é uma porta de I/O serial síncrona de alta velocidade que permite a transmissão de dados (1 à 16 *bits*) entre um dispositivo mestre e um ou vários dispositivos escravos.

10.1.1 Descrição Física da SPI

O módulo SPI consiste de:

- Quatro pinos I/O: SPISIMO, SPISOMI, SPICLK e SPISTE;
- Modos de operação mestre e escravo;
- Registrador *buffer* de recepção serial **(SPIRXBUF)**;
- Registrador *buffer* de transmissão serial **(SPITXBUF)**;
- Registrador de dados **(SPIDAT)**;
- Controle de polaridade e fase do SPICLK (*clock* da SPI);
- Lógica de controle;
- Registradores de *status* e controle mapeados na memória.

Na figura 10.1 está ilustrado um diagrama de blocos do módulo SPI.

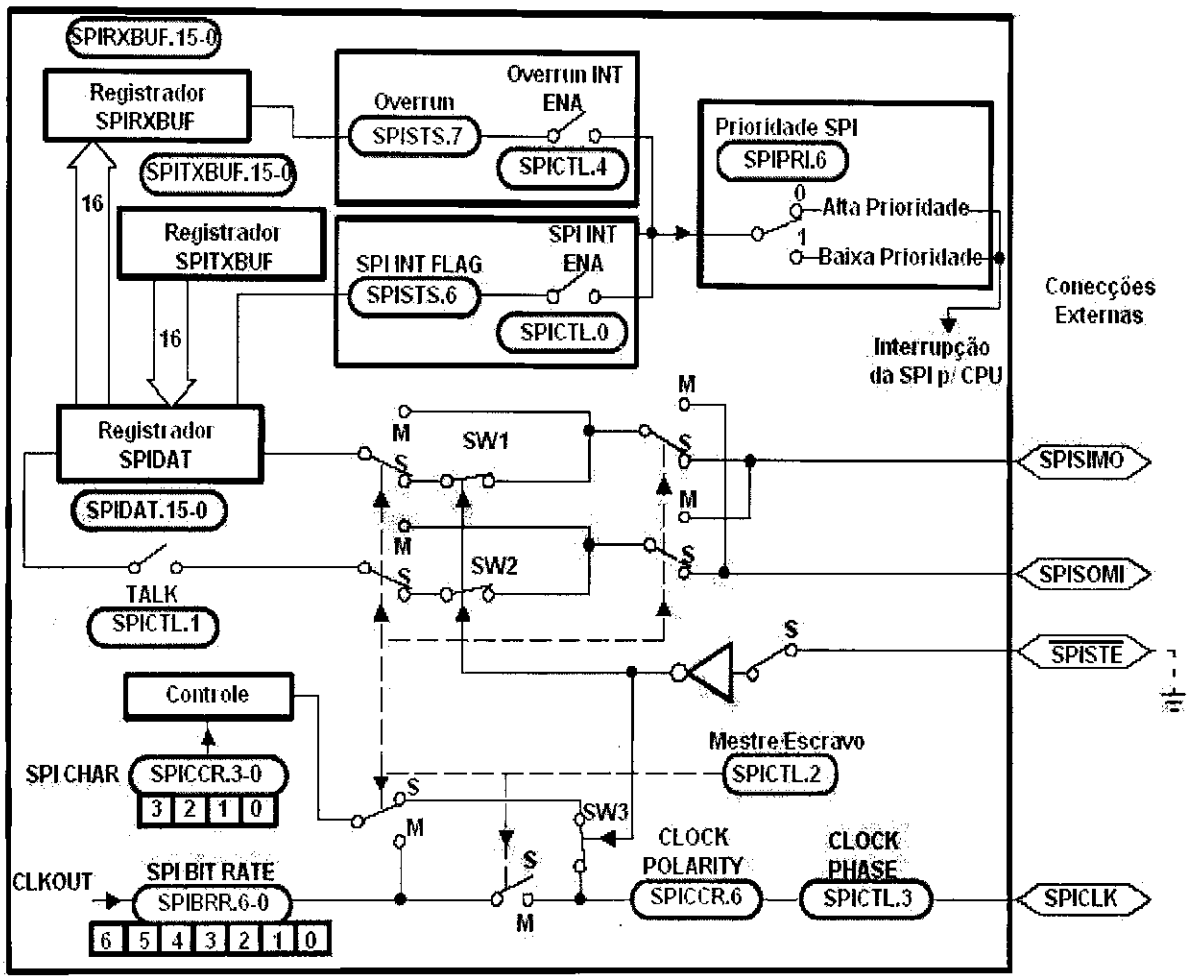


Figura 10.1: Diagrama de blocos do módulo SPI.

10.1.2 Pinos de I/O da SPI

Existem quatro pinos I/O na *interface* SPI: o SPISIMO, SPISOMI, SPICLK e o SPISTE.

O **SPISIMO** (*SPI Slave In Master Out*) é um pino de saída no dispositivo mestre e um pino de entrada no dispositivo escravo. Os dados são transferidos do dispositivo mestre para o dispositivo escravo por esse pino.

O **SPISOMI** (*SPI Slave Out Master In*) é um pino de saída no dispositivo escravo e um pino de entrada no dispositivo mestre. Os dados são transferidos do dispositivo escravo para o dispositivo mestre por esse pino.

O **SPICLK** (*SPI Clock*) é um pino que contém o *clock* da SPI. No dispositivo mestre esse pino é usado como um pino de saída e no dispositivo escravo esse pino é usado como um pino de entrada. Portanto, o dispositivo mestre controla a transferência dos dados. A frequência do *clock* não poderá ser maior que 1/4 do *clock* do dispositivo (CLKOUT).

O **/SPISTE** (*SPI Select Slave*) é um pino de entrada no dispostivo escravo. Esse pino é ativo em nível lógico baixo. A transferência de dados será realizada somente se o nível lógico nesse pino for baixo.

10.2 Operação da SPI

Na figura 10.2 está ilustrado um esquema com as conecções da SPI na comunicação entre dois controladores: um mestre e um escravo.

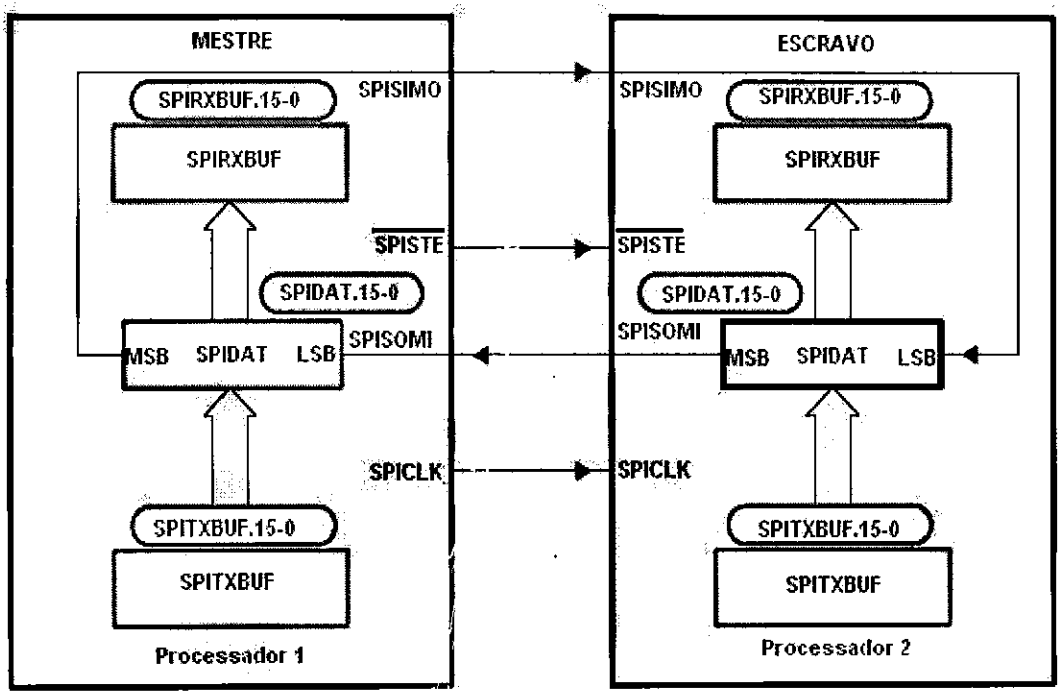


Figura 10.2: Esquema com as conecções dos pinos da SPI na comunicação entre um dispositivo mestre e um escravo.

O mestre inicia a transferência de dados enviando o sinal SPICLK. Para o mestre e o escravo, os dados são deslocados para fora do registrador de deslocamento em uma borda e deslocados para dentro do registrador de deslocamento na outra borda do SPICLK. Portanto, ambos os controladores enviam e recebem dados simultaneamente.

O mestre pode iniciar uma transferência de dados em qualquer momento, pois ele controla o sinal SPICLK.

10.2.1 Modo Mestre

No modo mestre, a SPI fornece o *clock* serial no pino SPICLK para a rede de comunicação serial. Os dados são transmitidos pelo pino SPISIMO e são recebidos no pino SPISOMI.

O registrador **SPIBRR** determina a taxa de transferência da rede. Esse registrador pode selecionar até 126 diferentes valores de taxas de transferência.

Dados escritos no registrador **SPIDAT** ou **SPITXBUF** iniciam a transmissão no pino SPISIMO. Simultaneamente, os dados recebidos (pelo mestre) são deslocados através do pino SPISOMI para o registrador **SPIDAT**. Quando o número de *bits* selecionados tiver sido transmitido, os dados recebidos são transferidos para o **SPIRXBUF** para serem lidos pela CPU.

Quando o número especificado de *bits* de dados tiver sido deslocado através de **SPIDAT**, os seguintes eventos ocorrem:

- O conteúdo de **SPIDAT** é transferido para **SPIRXBUF**;
- O *bit* SPI INT FLAG será setado;
- Se existir dados válidos no *buffer* de transmissão **(SPITXBUF)**, indicado pelo *bit* TXBUF FULL do registrador **SPISTS**, este dado será transferido para **SPIDAT** e em seguida será transmitido. Caso contrário, o SPICLK pára após todos os *bits* terem sido deslocados para fora de **SPIDAT**;
- Se o *bit* SPI INT ENA do registrador **SPICTL** estiver setado, uma interrupção será assegurada.

10.2.2 Modo Escravo

No modo escravo, os dados são transmitidos pelo pino SPISOMI e são recebidos pelo pino SPISIMO. O pino SPICLK é usado como um pino de entrada, no qual recebe o sinal de *clock* do dispositivo mestre.

Dados escritos no registrador **SPIDAT** ou **SPITXBUF** são transmitidos para a rede quando as apropriadas bordas do sinal SPICLK são recebidas do dispositivo mestre. Dados escritos no registrador **SPITXBUF** serão transferidos imediatamente para o registrador SPIDAT quando ele estiver vazio. Para receber dados, a SPI espera o dispositivo mestre enviar o sinal SPICLK e então desloca os dados do pino SPISIMO para o **SPIDAT**.

Quando o *bit* TALK do registrador SPICTL for zerado, a transmissão de dados será desabilitada, e a linha de saída (SPISOMI) será colacada em estado de alta impedância. Se isto ocorrer durante uma transmissão, o caracter que estiver sendo transmitido será transferido completamente e em seguida o pino SPISOMI será colocado em estado de alta impedância.

10.3 Interrupções SPI

Existem cinco *bits* de controle que são utilizados para inicializar as interrupções da SPI, são eles:

- SPI INT ENA do registrador **SPICTL**;
- SPI INT FLAG do registrador **SPISTS**;
- OVERRUN INT ENA do registrador **SPICTL**;
- RECEIVER OVERRUN FLAG do registrador **SPISTS**;
- SPI PRIORITY do registrador **SPIPRI**.

10.4 Formato dos Dados

Existem quatro *bits* do registrador **SPICCR** que especificam o número de *bits* (1 à 16) do caracter de dados. Esta informação é enviada para a lógica de controle, para determinar o fim de uma transmissão ou de uma recepção. Os seguintes cuidados deverão ser aplicados aos caracteres com menos de 16 *bits*:

- Os dados devem ser justificados a esquerda quando forem escritos em **SPIDAT** ou em **SPITXBUF**;
- os dados lidos de volta do **SPIRXBUF** estão justificados a direita;
- **SPIRXBUF** contém os caracteres recebidos mais recentes, justificados à direita, e mais alguns *bits* que restaram da transmissão anterior que foram deslocados à esquerda. Na figura 10.3 está ilustrado um esquema com o comportamento dos registradores **SPIDAT** e **SPIRXBUF** em uma transmissão de um *bit* de dados.

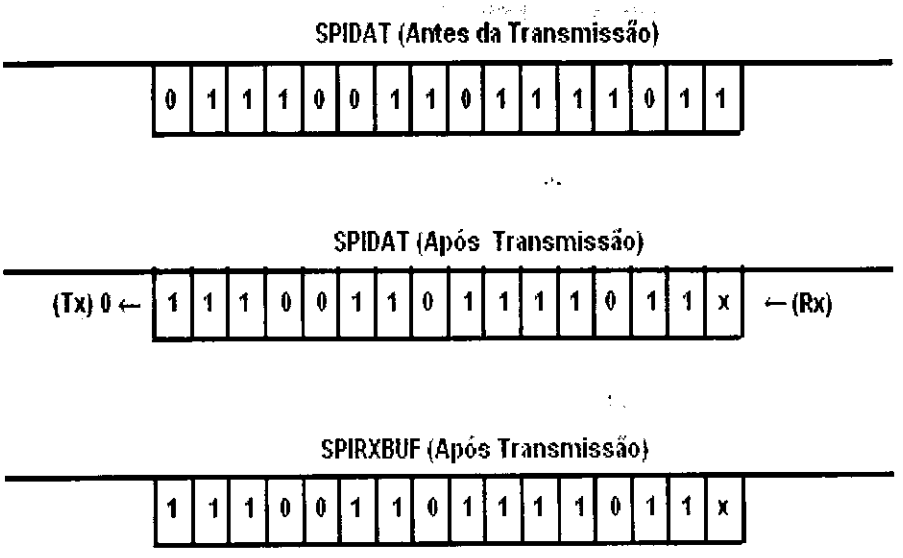


Figura 10.3: Esquema com o comportamento dos registradores SPIDAT e SPIRXBUF em uma transmissão de um *bit* de dados.

10.5 Determinação do Baud Rate

Para determinar o *baud rate* no módulo SPI, utiliza-se as seguintes equações:

Se o valor do registrador **SPIBRR** estiver entre 3 e 127, temos que:

$$BAUDRATE = \frac{CLKOUT}{(SPIBRR+1)}$$

Se o valor do registrador **SPIBRR** for 0, 1 ou 2, temos que:

$$BAUDRATE = \frac{CLKOUT}{4}$$

SPIBRR é o valor do registrador **SPIBRR** no dispositivo mestre.

10.6 Esquema do Clock da SPI

Existem quatro diferentes esquemas do *clock* da SPI. O *bit* CLOCK POLARITY do registrador **SPICCR** e o *bit* CLOCK PHASE do registrador **SPICTL** controlam esses quatro tipos de esquemas. O *bit* CLOCK POLARITY seleciona a borda ativa do *clock* (borda de subida ou de descida) e o *bit* CLOCK PHASE seleciona um atraso do *clock* de meio ciclo. Os quatro esquemas do *clock* são os seguintes:

- Borda de descida sem atraso. A SPI transmite os dados na borda de descida do SPICLK e recebe os dados na borda de subida do SPICLK.
- Borda de descida com atraso. A SPI transmite os dados na borda de descida do SPICLK com um atraso de meio ciclo e recebe os dados na borda de subida do SPICLK também com meio ciclo de atraso.
- Borda de subida sem atraso. A SPI transmite os dados na borda de subida do SPICLK e recebe os dados na borda de descida do SPICLK.
- Borda de subida com atraso. A SPI transmite os dados na borda de subida do SPICLK com um atraso de meio ciclo e recebe os dados na borda de descida do SPICLK também com meio ciclo de atraso.

Na tabela 10.1 está indicado os valores dos *bits* CLOCK POLARITY e CLOCK PHASE para os respectivos esquemas do *clock* da SPI.

Na figura 10.4 está ilustrado os sinais da SPI para os quatro esquemas de *clock*.

Tabela 10.1: Valores dos *bits* CLOCK POLARITY e CLOCK PHASE para os respectivos esquemas do *clock* da SPI.

Esquema do SPICLK	CLOCK POLARITY (Bit 6 do SPICCR)	CLOCK PHASE (Bit 3 do SPICTL)
Borda de subida sem atraso	0	0
Borda de subida com atraso	**0**	**1**
Borda de descida sem atraso	1	0
Borda de descida com atraso	**1**	**1**

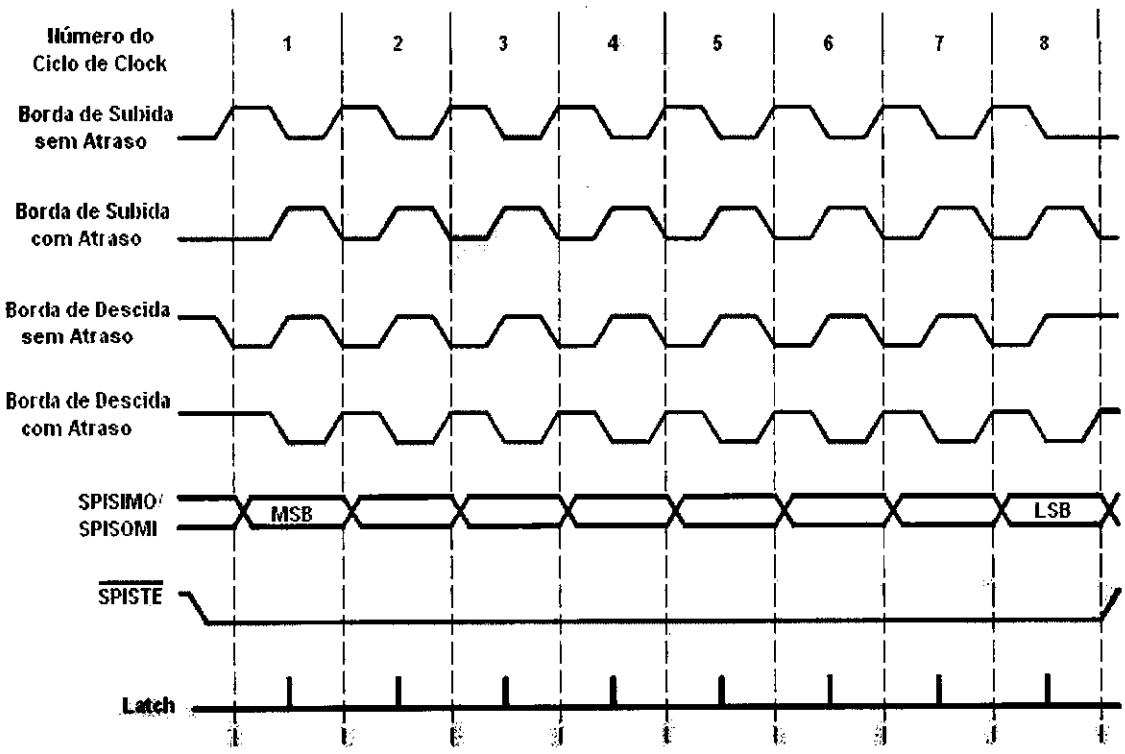


Figura 10.4: Sinais da SPI para os quatro esquemas de *clock*.

A simétria do SPICLK é preservada somente quando o resultado de (SPIBRR + 1) for um valor par. Quando esse valor for ímpar e SPIBRR for maior que três, o SPICLK se torna assimétrico.

10.7 Registradores de Controle da SPI

10.7.1 Registrador de Controle da Configuração - SPICCR

Na figura 10.5 está ilustrado um esquema com a posição dos *bits* no registrador **SPICCR**.

7	6	5–4	3	2	1	0
SPI SW RESET	CLOCK POLARITY	Reservado	SPI CHAR3	SPI CHAR2	SPI CHAR1	SPI CHAR0
RW-0	RW-0	R-0	RW-0	RW-0	RW-0	RW-0

Figura 10.5: Esquema com as posições dos *bits* no registrador SPICCR.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bit 7 - SPI SW RESET** - *Reset* da SPI por *software*. Quando o usuário for alterar a configuração, este *bit* deverá antes ser zerado e logo após a configuração ser concluída esse *bit* deverá ser setado.

  0 - Inicializa os *flags* de operação da SPI para a condição de *reset*. Os *bits* RECEIVER OVERRUN, SPI INT FLAG e TXBUF FULL são zerados. A configuração da SPI permanece inalterada. Se o módulo estiver operando como um mestre, o sinal de saída SPICLK retornará para seu nível inativo.

  1 - A SPI está pronto para transmitir ou receber o próximo caracter. Quando o *bit* SPI SW RESET for '0', um caracter escrito no transmissor não será deslocado para fora quando este *bit* for setado. Um novo caracter deverá ser escrito no registrador de dados.

- **Bit 6 - CLOCK POLARITY** - Indica a polaridade do *clock*.

  0 - Dados são enviados na borda de subida e recebidos na borda de descida. Quando nenhum dado estiver sendo enviado, o SPICLK ficará em nível baixo.

  1 - Dados são enviados na borda de descida e recebidos na borda de subida. Quando nenhum dado estiver sendo enviado, o SPICLK ficará em nível alto.

- **Bits 2-0 - SPI CHAR3 - SPI CHAR0** - Esses *bits* controlam o comprimento do caracter que serão transmitidos e recebidos. Na tabela 10.2 está indicado os possíveis valores desses quatro *bits* e os respectivos comprimentos do caracter.

Tabela 10.2: Possíveis valores dos *bits* SPI CHAR3 - SPI CHAR0 e seus respectivos valores de comprimentos do caracter.

SPI CHAR3-0	Comprimento do Caracter (Bits)
0000	1
0001	2
0010	3
0011	4
0100	5
0101	6
0110	7
0111	8
1000	9
1001	10
1010	11
1011	12
1100	13
1101	14
1110	15
1111	16

10.7.2 Registrador de Controle da Operação da SPI - SPICTL

Na figura 10.6 está ilustrado um esquema com a posição dos *bits* no registrador **SPICTL**.

7–5	4	3	2	1	0
Reservado	OVERRUN INT ENA	CLOCK PHASE	MASTER/ SLAVE	TALK	SPI INT ENA
R-0	RW-0	RW-0	RW-0	RW-0	RW-0

Figura 10.6: Esquema com as posições dos *bits* no registrador SPICTL.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bit 4 - OVERRUN INT ENA** - Habilita operação de *overrun*. Quando o *flag* RECEIVER OVERRUN for setado por *hardware* e este *bit* estiver setado, ma interrupção será gerada.

  0 - Desabilita a interrupção de *overrun* no receptor.

  1 - Habilita a interrupção de *overrun* no receptor.

- **Bit 3 - CLOCK PHASE** - Seleciona a fase do *clock* da SPI.

  0 - Esquema de *clock* normal, dependendo do *bit* CLOCK POLARITY.

  1 - O sinal SPICLK será atrasado por meio ciclo de *clock*. A polaridade é determinada pelo *bit* CLOCK POLARITY.

- **Bit 2 - MASTER/SLAVE** - Indica o modo de operação do módulo SPI.

  0 - SPI configurado como escravo.

  1 - SPI configurado como mestre.

- **Bit 1 - TALK** - Habilita a transmissão do mestre/escravo. Esse *bit* pode desabilitar a transmissão de dados, colocando a saída de dados em estado de alta impedância. Se este *bit* for desabilitado durante uma transmissão, o registrador de deslocamento de transmissão continuará a operar até o caracter ser transferido totalmente. Quando o *bit* TALK for zerado, a SPI continuará a receber os caracteres e a atualizar os *flags* de *status*.

  0 - Desabilita a transmissão. No modo escravo, o pino SPISOMI será colocado em estado de alta impedância. No modo mestre, o pino SPISIMO será colocado em estado de alta impedância.

  1 - Habilita a transmissão

- **Bit 0 - SPI INT ENA** - Habilita a interrupção da SPI. Se esse *bit* estiver setado uma interrupção será gerada em uma recepção/transmissão.

  0 - Desabilita interrupção.

  1 - Habilita interrupção.

10.7.3 Registrador de Status - SPISTS

Na figura 10.7 está ilustrado um esquema com a posição dos *bits* no registrador **SPISTS.**

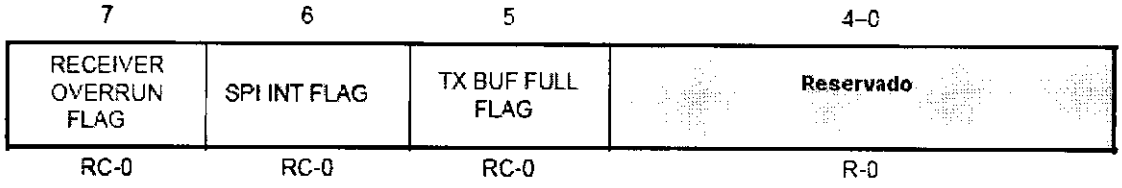


Figura 10.7: Esquema com as posições dos *bits* no registrador SPISTS.

OBS: **R** = acesso a leitura, **C** = pode apenas ser zerado, **-n** = valor após o *reset.*

- **Bit 7 - RECEIVER FLAG OVERRUN** - *Flag* de *overrun* do receptor. O *hardware* da SPI seta este *bit* quando um recepção ou transmissão for concluída antes do caracter anterior ser lido do *buffer*. Este *bit* indica que o último caracter recebido foi sobrescrito, e portanto perdido. Se o *bit* OVERRUN INT ENA estiver setado, uma requisição de interrupção ocorrerá quando este *bit* for setado. O *bit* RECEIVER FLAG OVERRUN pode ser zerado de três formas:

  – Escrevendo um '1' nele;

  – Escrevendo um '0' no *bit* SPI SW RESET;

  – Resetando o sistema.

  Deve-se zerar este *bit* toda vez que ocorrer um *overrun* na rotina de serviço de interrupção. Caso contrário, outras interrupções de *overrun* não serão sinalizadas.

- **Bit 6 - SPI INT FLAG** - *Flag* de interrupção da SPI. Toda vez que o último *bit* de um caracter for transmitido ou recebido, o *hardware* da SPI setará esse *bit*. Se o *bit* SPI INT ENA estiver setado, uma requisição de interrupção será gerada. Este *bit* pode ser zerado de três formas:

  – Lendo o registrador SPIRXBUF;

  – Escrevendo um '0' no *bit* SPI SW RESET;

  – Resetando o sistema.

  Deve-se zerar este *bit* na rotina de serviço de interrupção. Caso contrário, outras interrupções ad SPI não serão sinalizados.

- **Bit 5 - TX BUF FULL FLAG** - Indica que o *buffer* de transmissão está cheio. Quando um caracter for escrito no registrador **SPITXBUF** esse *bit* será setado. Ele será zerado automaticamente quando o caracter for carregado no registrador **SPIDAT**.

10.7.4 Registrador do Baud Rate - SPIBRR

Na figura 10.8 está ilustrado um esquema com a posição dos bits no registrador SPIBRR.

7	6	5	4	3	2	1	0
Reservado	SPI BIT RATE 6	SPI BIT RATE 5	SPI BIT RATE 4	SPI BIT RATE 3	SPI BIT RATE 2	SPI BIT RATE 1	SPI BIT RATE 0
R-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Figura 10.8: Esquema com as posições dos *bits* no registrador SPIBRR.

OBS: **R** = acesso a leitura, **W** = acesso a escrita **-n** = valor após o *reset*.

- **Bits 6-0 - SPI BIT RATE6 - SPI BIT RATE0** - Indicam o *baud rate* da SPI. Esses *bits* indicam a taxa de transferência se o módulo SPI estiver no modo mestre. No modo escravo, esses *bits* não tem efeito. A frequência do *clock* da SPI não poderá ser maior que 1/4 do CLKOUT.

10.7.5 Registrador do Buffer da Emulação - SPIRXEMU

Na figura 10.9 está ilustrado um esquema com a posição dos bits no registrador SPIRXEMU.

15	14	13	12	11	10	9	8
ERXB15	ERXB14	ERXB13	ERXB12	ERXB11	ERXB10	ERXB9	ERXB8
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
7	6	5	4	3	2	1	0
ERXB7	ERXB6	ERXB5	ERXB4	ERXB3	ERXB2	ERXB1	ERXB0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Figura 10.9: Esquema com as posições dos *bits* no registrador SPIRXEMU.

OBS: **R** = acesso a leitura, **-n** = valor após o *reset*.

- **Bits 15-0 - ERXB15 - ERXB0** - Contém os dados recebidos no *buffer* de emulação. **SPIRXEMU** tem a mesma função do **SPIRXBUF**, exceto que a leitura de **SPIRXEMU** não zera o *bit* SPI INT FLAG. Quando o registrador **SPIDAT** receber o caracter completo, esse caracter será transferido para os registradores **SPIRXEMU** e **SPIRXBUF**, onde poderão ser lidos. Esse registrador foi criado para suportar emulação. É recomendável ver o conteúdo de **SPIRXEMU** no modo normal de operação do emulador.

10.7.6 Registrador do Buffer de Recepção - SPIRXBUF

Na figura 10.10 está ilustrado um esquema com a posição dos bits no registrador SPIRXBUF.

15	14	13	12	11	10	9	8
RXB15	RXB14	RXB13	RXB12	RXB11	RXB10	RXB9	RXB8
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
7	6	5	4	3	2	1	0
RXB7	RXB6	RXB5	RXB4	RXB3	RXB2	RXB1	RXB0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Figura 10.10: Esquema com as posições dos *bits* no registrador SPIRXBUF.

OBS: **R** = acesso a leitura, **-n** = valor após o *reset*.

- **Bits 15-0 - RXB15 - RXB0** - Contém os dados recebidos. Quando **SPIDAT** receber o caracter completo, esse caracter será transferido para **SPIRXBUF**, onde poderá ser lido. Esse registrador foi criado para suportar emulação.

10.7.7 Registrador do Buffer de Transmissão - SPITXBUF

Na figura 10.11 está ilustrado um esquema com a posição dos bits no registrador SPITXBUF.

15	14	13	12	11	10	9	8
TXB15	TXB14	TXB13	TXB12	TXB11	TXB10	TXB9	TXB8
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7	6	5	4	3	2	1	0
TXB7	TXB6	TXB5	TXB4	TXB3	TXB2	TXB1	TXB0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Figura 10.11: Esquema com as posições dos *bits* no registrador SPITXBUF.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bits 15-0 - TXB15 - TXB0** - Contém os dados a serem transmitidos. **SPITXBUF** armazena o próximo caracter a ser transmitido. Quando esse registrador for escrito, o *bit* TX BUF FULL será setado. Quando a transmissão do atual caracter for concluída, o conteúdo desse registrador será automaticamente carregado no registrador **SPIDAT** e o *bit* TX BUF FULL será zerado. A escrita nesse registrador deverá ser justificado à esquerda.

10.7.8 Registrador de Dados - SPIDAT

Na figura 10.12 está ilustrado um esquema com a posição dos *bits* no registrador **SPIDAT**.

15	14	13	12	11	10	9	8
SDAT15	SDAT14	SDAT13	SDAT12	SDAT11	SDAT10	SDAT9	SDAT8
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7	6	5	4	3	2	1	0
SDAT7	SDAT6	SDAT5	SDAT4	SDAT3	SDAT2	SDAT1	SDAT0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Figura 10.12: Esquema com as posições dos *bits* no registrador SPITXBUF.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bits 15-0 - SDAT15 - SDAT0** - Contém os dados a serem enviados pela linha serial se o *bit* TALK estiver setado. **SPIDAT** é um registrador de deslocamento na transmissão/recepção. O *bit* mais significativo será deslocado primeiro. Os dados a serem transmitidos deverão ser justificados à esquerda e os dados recebidos são justificados à direita.

10.7.9 Registrador de Controle da Prioridade - SPIPRI

Na figura 10.13 está ilustrado um esquema com a posição dos bits no registrador SPIPRI.

7	6	5	4	3–0
Reservado	SPI PRIORITY	SPI SUSP SOFT	SPI SUSP FREE	Reservado
R-0	RW	RW	RW-0	R-0

Figura 10.13: Esquema com as posições dos *bits* no registrador SPIPRI.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bit 6 - SPI PRIORITY** - Seleciona o nível de prioridade da interrupção SPI.

  0 - As requisições de interrupção são de alta prioridade.

  1 - As requisições de interrupção são de baixa prioridade.

- **Bits 5-4 - SPI SUSP SOFT e SPI SUSP FREE** - Estes *bits* determinam o que ocorrerá quando ocorrer uma suspensão da emulação.

  00 - Pára imediatamente.

  01 - Continua a operação SPI, apesar da suspensão da emulação.

  10 - Completa a atual sequência de recepção/transmissão antes de parar.

  11 - Continua a operação SPI, apesar da suspensão da emulação.

Capítulo 11

Módulo CAN

11.1 Introdução

CAN (*Controller Area Network*) [10] [11] [12] [13] [14] [15] é um protocolo de comunicações serial utilizado de forma eficiente em aplicações onde se deseja controlar sistemas distribuídos em tempo real com um alto nível de confiabilidade das informações. Em geral, utiliza-se a tecnologia CAN em sistemas onde dispositivos inteligentes precisam se comunicar.

O protocolo CAN utiliza uma comunicação serial multi-mestre utilizando apenas dois fios, onde as mensagens possuem prioridades intrínsecas e um comprimento de dados de até oito *bytes* de informação útil, que podem ser transmitidas à uma taxa de até 1 Mbps utilizando um protocolo de arbitragem e excelentes mecanismos de detecção e sinalização de erros que proporcionam a integridade das informações transmitidas.

Existem quatro diferentes tipos de mensagens para comunicação:

- **Mensagem de dados** - Essa mensagem contém os dados do nó transmissor para o nó receptor;
- **Mensagem de requisição remota** - Essa mensagem é transmitida por um nó para requisitar dados de outro nó da rede;
- **Mensagem de erro** - Essa mensagem é transmitida por qualquer nó da rede que detecte um erro. Sua funcão é notificar a ocorrência de falhas;
- **Mensagem de sobrecarga** - Essa mensagem é utilizada para sinalizar a um transmissor que o receptor não está pronto para receber as próximas mensagens.

A versão 2.0B [10] do protocolo CAN define dois formatos de mensagens que diferem no comprimento do campo do identificador, são os formatos padrão e estendido. As mensagens com formato padrão possuem identificadores com 11 *bits* e as mensagens com formato estendido possuem identificadores com 29 *bits*. Para maiores informações sobre o protocolo de comunicações CAN consultar [10] e [11]. Na figura 11.1 está ilustrado um diagrama com os campos de uma mensagem de dados.

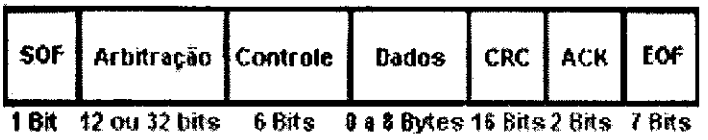


Figura 11.1: Diagrama com os campos de uma mensagem de dados.

O periférico CAN disponível no TMS320LF2407A possui as seguintes características:

- Implementação do protocolo CAN, versão 2.0B;
- Seis *mailbox* (MBOX) com comprimento de dados de 0 à 8 *bytes*;
- Duas máscaras de aceitação local;
- Taxa de *bits* programável;
- Esquema de interrupção programável;
- *Wake-up* programável;
- Resposta automtica a uma requisição remota;
- Re-transmissão automática no caso de erros ou preca de arbitragem;
- Diagnóstico de falhas no barramento;
- Modo *self-test*;

Na figura 11.2 está ilustrado o diagrama de blocos do módulo CAN do TMS320LF2407A.

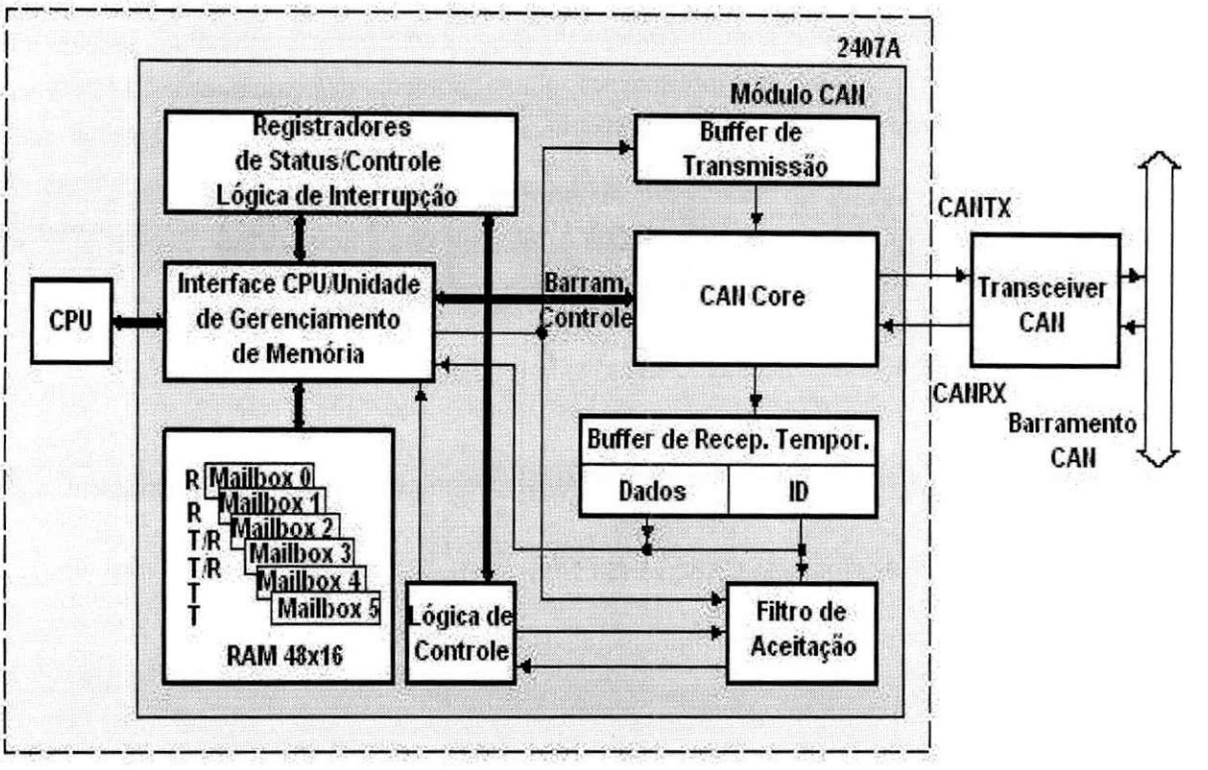


Figura 11.2: Diagrama de blocos do módulo CAN do TMS320LF2407A.

11.2 Mailbox

O *mailbox* (MBOX) é a área onde os pacotes CAN são armazenados antes de serem transmitidos, e onde são recebidos. Existem seis *mailbox* (MBOX0 - MBOX5). Os MBOX 0 e 1 são *mailbox* de recepção, os MBOX 4 e 5 são mailbox de transmissão e os MBOX 2 e 3 podem ser *mailbox* de transmissão ou de recepção. Cada *mailbox* possui quatro registradores de 16 *bits*, no qual podem armazenar um máximo de oito *bytes* (MBXnA, MBXnB, MBXnC e MBXnD). Os *mailboxes* que não estão sendo utilizados para armazenar as mensagens podem ser usados como memória pela CPU.

11.2.1 Identificadores das Mensagens

Cada um dos seis MBOX possui dois registradores que armazenam o identificador da mensagem. Nas figuras 11.3 e 11.4 estão ilustrados, respectivamente, os esquemas com as posições dos *bits* nos registradores **MSGIDnH** e **MSGIDnL**.

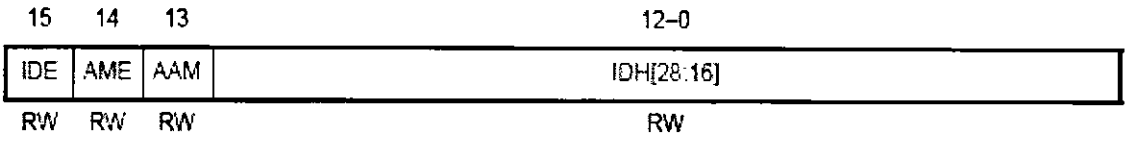


Figura 11.3: Esquemas com as posições dos *bits* no registrador MSGIDnH.

OBS: **R** = acesso a leitura, **W** = acesso a escrita.

- **Bit 15 - IDE** - *Bit* de extensão do identificador.

  0 - A mensagem recebida possui um identificador padrão (MBOX de recepção). A mensagem a ser enviada possui um identificador padrão (MBOX de transmissão).

  1 - A mensagem recebida possui um identificador estendido (MBOX de recepção). A mensagem a ser enviada possui um identificador estendido (MBOX de transmissão).

- **Bit 14 - AME** - *Bit* de habilitação da máscara de aceitação.

  0 - Nenhuma máscara de aceitação será usada. Todos os *bits* do identificador da mensagem recebida e do MBOX de recepção deverão ser iguais para a mensagem ser armazenada.

  1 - A correspondente máscara de aceitação será usada.
  Este *bit* é relevante apenas nos *mailboxes* de recepção. Esse bit não será afetado por uma recepção.

- **Bit 13 - AAM** - *Bit* de resposta automática a uma requisição remota.

  0 - Em um MBOX de transmissão, o MBOX não responderá a requisição remota automaticamente. Se o identificador recebido for igual ao identificador do MBOX, a mensagem não será armazenada.

  1 - Em um MBOX de transmissão, se uma requisição remota for recebida, o periférico CAN responderá enviando o conteúdo do MBOX.

  OBS: Esse *bit* não tem influência em um MBOX de recepção. Ele é usado somente nos MBOX 2 e 3 e não é afetado por uma recepção.

- **Bits 12-0 - IDH[15:0]** - Contém os 13 *bits* mais significativos de um identificador estendido. Para um identificador padrão, os 11 *bits* são armazenados nos *bits* 12 à 2.

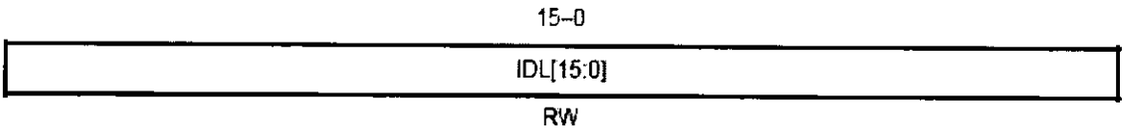


Figura 11.4: Esquemas com as posições dos *bits* no registrador MSGIDnL.

OBS: **R** = acesso a leitura, **W** = acesso a escrita.

- **Bits 15-0 - IDL[15:0]** - Contém a parte menos significativa de um identificador estendido.

11.2.2 Campo de Controle da Mensagem

Cada um dos seis MBOX possui um registrador responsável pelo seu campo de controle. Na figura 11.5 está ilustrado um esquema com as posições dos bits no registrador MSGCTRLn.

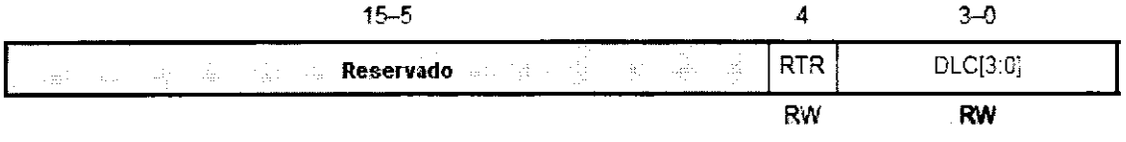


Figura 11.5: Esquemas com as posições dos *bits* no registrador MSGCTRLn.

OBS: **R** = acesso a leitura, **W** = acesso a escrita.

- **Bit 4 - RTR** - *Bit* de requisição remota de transmissão.

  0 - Pacote é de dados.

  1 - Pacote é de requisição remota.

- **Bits 3-0 - DLC** - Indica quantos *bytes* de dados serão transmitidos.

  0001 - Um *byte*

  0010 - Dois *byte*

  0011 - Três *byte*

  0100 - Quatro *byte*

0101 - Cinco *byte*

0110 - Seis *byte*

0111 - Sete *byte*

1000 - Oito *byte*

11.2.3 Acesso de Escrita ao Mailbox

Existem dois diferentes tipos de acesso de escrita ao mailbox:

1. Acesso de escrita ao identificador de um MBOX;
2. Acesso de escrita ao campo de dados ou controle.

O acesso de escrita ao identificador pode somente ser realizado quando o MBOX estiver desabilitado, ou seja, o *bit* MEn do registrador **MDER** estiver zerado.

Durante o acesso ao campo de dados ou ao campo de controle, é importante que os dados não mudem enquanto estiverem sendo lidos. Por isso, um acesso de escrita nesses campos para um MBOX de recepção é desabilitado. Em um MBOX de transmissão, o acesso de escrita será negado se o *bit* TRS ou TRR do registrador **TCR** forem iguais a '1'. Nesses casos, se for realizado um acesso de escrita nesses campos, o *flag* WDIF do registrador **CAN_IFR** será setado, e uma interrupção será gerada se estiver habilitada. Uma forma de acessar os MBOX 2 e 3 é setar o *bit* CDR do registrador **MCR** antes de acessar o MBOX. Após o acesso da CPU ser finalizado, a CPU deverá zerar o *bit* CDR escrevendo um '0' nele. O módulo CAN verifica o *bit* CDR antes e após a leitura do MBOX. Se CDR estiver setado durante a verificação feita pelo módulo CAN, a mensagem não será transmitida, mas o módulo CAN continuará verificando as outras requisições de transmissão.

11.2.4 Mailbox de Transmissão

Os **mailboxes** 4 e 5 são apenas MBOX de transmissão e os MBOX 2 e 3 podem ser configurados como MBOX de transmissão.

A CPU armazena os dados a serem transmitidos em um MBOX de transmissão e em seguida deve-se setar o correspondente *bit* TRS do registrador **TCR** para a mensagem ser enviada. Se mais que um *bit* TRS estiver setado, as mensagens serão enviadas uma após a outra, começando pelo MBOX de maior número.

11.2.5 Mailbox de Recepção

Os *mailboxes* 0 e 1 são apenas MBOX de recepção e os MBOX 2 e 3 podem ser configurados como MBOX de recepção.

O identificador da mensagem a ser recebida é comparado ao identificador dos *mailboxes* de recepção usando a máscara de aceitação. Se a mensagem for aceita, ela será recebida no MBOX que a aceitou. No mesmo instante, o correspondente *bit* RMPn do registrador **RCR** será setado e uma interrupção do MBOX (MIFx) do registrador **CAN_IFR** será gerada se estiver habilitada. O *bit* RMPn deverá ser resetado pela CPU após a leitura do dados. Se uma segunda mensagem for recebida no MBOX e o correspondente *bit* RMPn já estiver setado, o *bit* RMLn do registrador **RCR** será setado. Neste caso, a mensagem armazenada será sobrescrita com o novo dado, se o *bit* OPCn do registrador **RCR** for '0'. Caso o *bit* OPCn for '1', os outros MBOX de recepção serão verificados.

11.3 Manipulação de Pacotes de Requisição

A manipulação de pacotes de requisição remota pode apenas ser feito pelos MBOX 0, 1, 2 e 3.

Se uma requisição remota for recebida, o módulo CAN compara o identificador recebido com o identificador dos MBOX usando as máscaras apropriadas em ordem decrescente (MBOX 3, 2, 1 e 0). Se um MBOX de transmissão receber a mensagem e o *bit* AAM do registrador **MSGIDnH** estiver setado, um pacote de dados será transmitido por esse MBOX automaticamente (TRS=1). Na figura 11.6 está ilustrado um esquema com esse processo.

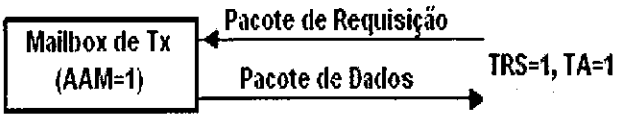


Figura 11.6: Esquema com o processo de recepção de uma requisição remota por um MBOX de transmissão com o bit AAM=1.

Caso o bit AAM for '0', a mensagem não será recebida. Na figura 11.7 está ilustrado um esquema com esse processo.

Figura 11.7: Esquema com o processo de recepção de uma requisição remota por um MBOX de transmissão com o bit AAM=0.

Se um MBOX de recepção receber uma requisição remota, a mensagem será manipulada como se fosse um pacote de dados e o *bit* RMPn será setado. A CPU deverá então decidir como manipular a situação. Na figura 11.8 está ilustrado um esquema com esse processo.

Figura 11.8: Esquema com o processo de recepção de uma requisição remota por um MBOX de recepção.

Se a CPU quiser mudar os dados em uma mensagem que está configurada como um MBOX de pacote de requisição remota e AAM for '1', ele deverá setar primeiro o número do MBOX no *bit* MBNR do registrador **MCR** e setar o *bit* CDR. A CPU poderá então desempenhar o acesso e zerar o *bit* CDR para informar ao módulo CAN que o acesso foi finalizado. Até que o *bit* CDR não seja zerado, a transmissão desse MBOX não será desempenhada.

11.3.1 Enviando um Requisição Remota

Se a CPU de um nó quiser requisitar dados de outro nó, ele deverá configurar o MBOX (apenas os MBOX 2 e 3) como MBOX de recepção e setar o *bit* TRS. Na figura 11.9 está ilustrado um esquema desse processo.

Figura 11.9: Esquema com o processo de envio de uma requisição remota por um MBOX de recepção.

Nesse caso, o módulo enviará um pacote de requisição remota e receberá um pacote de dados no mesmo MBOX que enviou a requisição remota.

11.4 Configuração dos Mailbox

Um MBOX pode ser configurado em quatro diferentes modos:

1. MBOX 2, 3, 4 e 5 podem apenas transmitir mensagens;
2. MBOX 0 e 1 podem apenas receber mensagens;
3. MBOX 2 e 3 configurados como MBOX de recepção podem transmitir um pacote de requisição remota e esperar pelo correspondente pacote de dados se o *bit* TRS for setado;
4. MBOX 2 e 3 configurados como MBOX de transmissão podem transmitir um pacote de dados se um pacote de requisição remota foi recebido por ele, e o *bit* AAM estiver setado.

Após o sucesso da transmissão de um pacote de requisição remota, o *bit* TRS será resetado, mas o *bit* TA não será setado e nem o *flag* de interrupção do MBOX será setado.

11.5 Filtro de Aceitação

O identificador da mensagem a ser recebida é primeiro comparado ao identificador dos *mailboxes* de recepção. Então, a apropriada máscara de aceitação será usada para mascarar os *bits* do identificador que não serão comparados. O *bit* AME é usado para habilitar/desabilitar a máscara de aceitação local (LAM).

Existem duas máscaras de aceitação local: LAM0 e LAM1. A LAM0 é utilizada pelos MBOX 0 e 1, e a LAM1 é utilizada pelos MBOX 2 e 3. Em uma recepção, os MBOX 2 e 3 são verificados antes dos MBOX 0 e 1. Nas figuras 11.10 e 11.11 estão ilustrados, respectivamente, os esquemas com as posições dos *bits* nos registradores **LAMn_H** e **LAMn_L**.

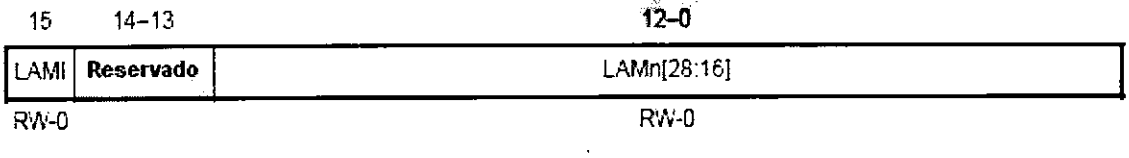


Figura 11.10: Esquema com as posições dos *bits* no registrador LAMn_H.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bit 15 - LAMI** - *Bit* de extensão do identificador da máscara de aceitação local.

  0 - O *bit* IDE do MBOX determina quais mensagens serão recebidas (padrão ou estendida).

  1 - Pacotes no formato padrão e estendido poderão ser recebidos. No caso de um pacote com formato estendido, todos os 29 *bits* do identificador são armazenados no *mailbox* e todos os 29 *bits* do registrador da máscara de aceittação global são usados no filtro. No caso de um pacote no formato padrão, apenas os primeiros 11 *bits* do identificador e da máscara de aceitação local serão usados.

- **Bits 12-0 - LAMn[28:16]** - Contém os 13 *bits* mais significativos da máscara de aceitação local.

  0 - O valor do *bit* do identificador recebido deverá ser igual ao respectivo *bit* do identificador do *mailbox*. Por exemplo, se o *bit* 27 da LAM for zero, então o *bit* 27 do identificador transmitido e o *bit* 27 do *mailbox* de recepção deverão ser iguais.

  1 - Aceita um '0' ou um '1' para o correspondente *bit* do identificador recebido.

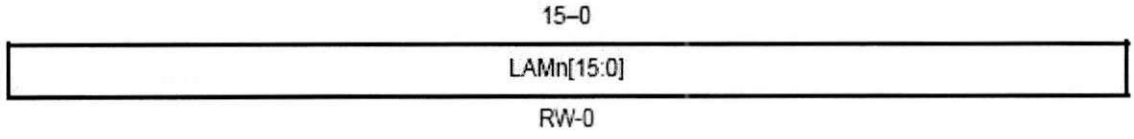


Figura 11.11: Esquema com as posições dos *bits* no registrador LAMn_L.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bits 15-0 - LAMn[15:0]** - *Bits* menos significativos da máscara de aceitação local.

  0 - O valor do *bit* do identificador recebido deverá ser igual ao respectivo *bit* do identificador do *mailbox*.

  1 - Aceita um '0' ou um '1' para o correspondente *bit* do identificador recebido.

11.6 Registradores de Controle CAN

Os registradores de controle permitem que funções do *mailbox* sejam manipuladas. Cada registrador desempenham uma função específica, tais como habilitar ou desabilitar o *mailbox*, controlar as transmissões e recepções e manipular as interrupções.

11.6.1 Registrador de Habilitação e Direção do Mailbox

Na figura 11.12 está ilustrado um esquema com as posições dos *bits* do registrador **MDER**.

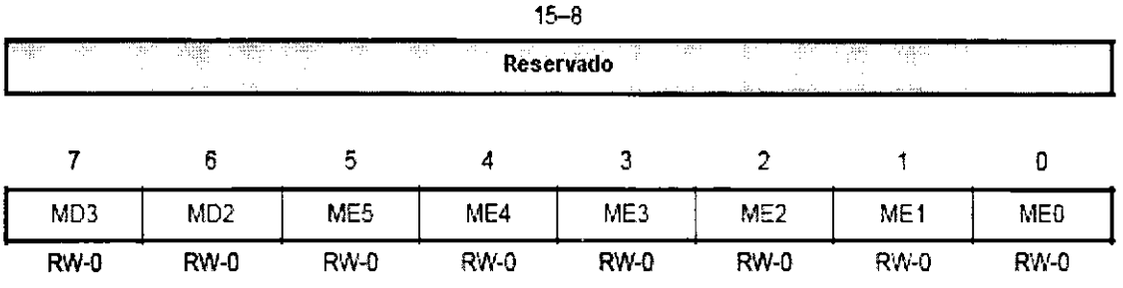


Figura 11.12: Esquema com as posições dos *bits* no registrador MDER.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bits 7-6 - MDn** - Indica a direção do **mailbox** n, onde n = 2 ou 3. Os MBOX 2 e 3 podem ser configurados como um MBOX de transmissão ou de recepção.

  0 - MBOX de transmissão.

  1 - MBOX de recepção.

- **Bits 5-0 - MEn** - Habilita o *mailbox* de número n, onde n pode ser de 0 à 5.

  0 - Desabilita o MBOX.

  1 - Habilita o MBOX.

11.6.2 Registrador de Controle de Transmissão

Na figura 11.13 está ilustrado um esquema com as posições dos bits do registrador TCR.

15	14	13	12	11	10	9	8
TA5	TA4	TA3	TA2	AA5	AA4	AA3	AA2
RC-0	RC-0	RC-0	RC-0	RC-0	RC-0	RC-0	RC-0
7	6	5	4	3	2	1	0
TRS5	TRS4	TRS3	TRS2	TRR5	TRR4	TRR3	TRR2
RS-0	RS-0	RS-0	RS-0	RS-0	RS-0	RS-0	**RS-0**

Figura 11.13: Esquema com as posições dos *bits* no registrador TCR.

OBS: **R** = acesso a leitura, **C** = pode ser apenas zerado, **S** = pode ser apenas setado, **-n** = valor após o *reset*.

- **Bits 15-12 - TAn** - *Bits* de reconhecimento da transmissão. Se a mensagem no MBOX n tiver sido enviada com sucesso, o *bit* TAn será setado. Esses *bits* são resetados quando se escreve um '1' neles. Isto também zera a interrupção, se ela for gerada. Esses *bits* setam o *flag* de interrupção do MBOX (MIFx) do registrador **CAN_IFR**. Se o correspondente *bit* de habilitação da interrupção no registrador **CAN_IMR** estiver setado, uma interrupção será gerada.

- **Bits 11-8 - AAn** - *Bits* de aborto de reconhecimento. Se a transmissão de uma mensagem no MBOX n foi abortada, o *bit* AAn será setado e o *bit* AAIF do registrador **CAN_IFR** será setado. O *bit* AAIF gera uma interrupção de erro se ela estiiver habilitada. Esses *bits* são resetados quando se escreve um '1' neles.

- **Bits 7-4 - TRSn** - *Bits* de requisição de transmissão. para iniciar uma transmissão, o correspondente *bit* TRS deverá ser setado. Se TRS estiver setado, o acesso de escrita ao correspondente MBOX será negado. Caso isso ocorra, o *flag* WDIF será setado e uma interrupção será gerada se estiver habilitada. Esses *bits* são resetados automaticamente após um sucesso da transmissão ou por um aborto da transmissão.

- **Bits 3-0 - TRRn** - *Bits* de *reset* da requisição de transmissão. Esses *bits* podem ser setados apenas pela CPU (usuário) e resetados por uma lógica interna. Se TRR for setado, o acesso de escrita ao correspondente MBOX será negado. Nesse caso, uma tentativa de escrita iniciará uma interrupção se ela estiver habilitada. Se TRR for setado e a transmissão na qual foi iniciada por TRS não foi processada corretamente, a correspondente requisição de transmissão será

cancelada. Se a correspondente mensagem foi processada, esse *bit* será resetado na ocorrência dos seguintes eventos:

1. Um sucesso na transmissão;
2. Um aborto da transmissão devido uma perca de arbitragem;
3. Uma condição de erro detectada no barramento CAN.

Em resumo, se a transmissão foi bem sucedida, o *bit* TAn será setado. Se a transmissão foi abortada, o *bit* AAn será setado e no caso de ocorrer uma condição de erro, um *bit* do registrador **ESR** será setado.

11.6.3 Registrador de Controle de Recepção

Na figura 11.14 está ilustrado um esquema com as posições dos *bits* do registrador **RCR**.

15	14	13	12	11	10	9	8
RFP3	RFP2	RFP1	RFP0	RML3	RML2	RML1	RML0
RC-0	RC-0	RC-0	RC-0	R-0	R-0	R-0	R-0
7	6	5	4	3	2	1	0
RMP3	RMP2	RMP1	RMP0	OPC3	OPC2	OPC1	OPC0
RC-0	RC-0	RC-0	RC-0	RW-0	RW-0	RW-0	RW-0

Figura 11.14: Esquema com as posições dos *bits* no registrador RCR.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **C** =pode ser apenas zerado, **-n** = valor após o *reset*.

- **Bits 15-12 - RFPn** - Esse *bits* indicam que um pacote de requisição remota está pendente. Quando um pacote de requisição remota for recebido em um MBOX, o correspondente *bit* RFPn será setado. Ele poderá ser zerado pela CPU se o *bit* TRSn não estiver setado, caso contrário, ele será resetado automaticamente. Se o *bit* AAM não estiver setado, a CPU deverá zerar o *bit* RFPn após manipular o evento. Caso AAM esteja setado, o *bit* TRS será setado e em seguida será resetado e o *bit* RFPn será zerado automaticamente. Se a mensagem foi enviada com sucesso, RFPn será zerado pelo módulo CAN.

- **Bits 11-8 - RMLn** - Esses *bits* indicam a perca de uma mensagem recebida. Se uma mensagem for sobrescrita por uma outra mensagem em um MBOX, o correspondente *bit* RMLn será setado. RMLn não será setado nos MBOX que tiverem o *bit* OPCn setado. Neste caso uma mensagem será perdida sem notificação. Esses *bits* podem ser resetados pela CPU e podem ser setados por uma lógica interna. Eles são zerados escrevendo-se um '1' nos *bits* RMPn. Se um ou mais bits RMLn estão setados, o *flag* RMLIF do registrador **CAN_IFR** será setado, e uma interrupção será gerada se o *bit* RMLIM do registrador **CAN_IMR** estiver setado.

- **Bits 7-4 - RMPn** - Indica que uma mensagem foi recebida. Se uma mensagem foi recebida em um MBOX n, o respectivo *bit* RMPn será setado. Os *bits* RMPn podem somente serem resetados pela CPU, e são setados por uma lógica interna. Os *bits* RMPn e RMLn são zerados quando se escreve um '1' nos *bits* RMPn. Os *bits* RMPn setam o *flag* de interrupção MIFX do registrador **CAN_IFR**, e se o correspondente *bit* de máscara de interrupção estiver setado, uma interrupção será gerada.

- **Bits 3-0 - OPCn** - *Bits* de controle de proteção de sobrescrita. Se existir uma condição de *overflow* no MBOX n, a nova mensagem será armazenada ou ignorada, dependendo do valor do *bit* OPCn. Se o correspondente *bit* OPCn estiver setado, a mensagem armazenada no MBOX será protegida, isto é, não será sobrescrita por uma nova mensagem. Então, os próximos MBOX serão verificados. Se nenhum MBOX aceitar a mensagem, a mensagem será perdida, sem notificação. Se OPCn não estiver setado, a mensagem no MBOX será sobrescrita pela nova mensagem.

11.6.4 Registrador de Controle Mestre - MCR

Na figura 11.15 está ilustrado um esquema com as posições dos *bits* no registrador **MCR**.

15–14	13	12	11	10	9	8
Reservado	SUSP	CCR	PDR	DBO	WUBA	CDR
	RW-0	RW-1	RW-0	RW-0	RW-0	RW-0

7	6	5-2	1-0
ABO	STM	Reservado	MBNR[1:0]
RW-0	RW-0		RW-0

Figura 11.15: Esquema com as posições dos *bits* no registrador MCR.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bit 13 - SUSP** - *Bit* de ação de suspensão do emulador. O valor desse *bit* não tem efeito nos MBOX de recepção.

  0 - Modo *soft*. O periférico será desligado durante a suspensão após a atual transmissão ser concluída.

  1 - Modo *free*. O periférico continua executando na suspensão.

- **Bit 12 - CCR** - *Bit* de requisição de alteração na configuração.

  0 - A CPU requisita o modo normal de operação.

  1 - A CPU requisita acesso de escrita aos registradores **BCRn** (modo de configuração). O *bit* CCE do registrador **GSR** indica se o acesso foi garantido. CCR deverá ser setado quando o *bit timing* for configurado nos registradores **BCRn**. Este *bit* será automaticamente setado, se a condição de *bus-off* ocorrer e o *bit* ABO não estiver setado. Deste modo, ele deverá ser resetado para sair do modo *bus-off*.

- **Bit 11 - PDR** - *Bit* de requisição do modo de baixa potência. Antes da CPU entrar no modo IDLE, ele deverá requisitar ao módulo CAN o modo de baixa potência, escrevendo um '1' no *bit* PDR. A CPU deverá então verificar o *bit* PDA no registrador **GSR** e entrar no modo IDLE somente após PDA ser setado.

  0 - O modo de baixa potência não foi requisitado (operação normal).

  1 - O modo de baixa potência foi requisitado.

- **Bit 10 - DBO** - Ordem dos *bytes* de dados.

  0 - Os dados são transmitidos ou recebidos na seguinte ordem: 3, 2, 1, 0, 7, 6, 5, 4.

  1 - Os dados são transmitidos ou recebidos na seguinte ordem: 0, 1, 2, 3, 4, 5, 6, 7.

- **Bit 9 - WUBA** - *Bit* de *wake-up* com a atividade no barramento.

  0 - O módulo sai do modo de baixa potência somente após o usuário escrever um '0' no *bit* PDR.

  1 - O módulo sai do modo de baixo quando um *bit* de valor dominante for detectado no barramento.

- **Bit 8 - CDR** - *Bit* de requisição de alteração no campo de dados. Este *bit* é aplicável em todos os MBOX, mas no MBOX 2 e 3 ele é aplicável apenas nas seguintes situações: um ou ambos estão configurados como para transmitir e o *bit* AAM está setado.

  0 - A CPU requisita modo normal de operação.

  1 - A CPU requisita acesso de escrita ao campo de dados do MBOX especificado pelo *bit* MBNR (MBOX 2 ou 3) ou dos demais MBOX. O *bit* CDR deverá ser zerado pela CPU após acessar o MBOX. O módulo CAN não transmitirá o conteúdo do MBOX se CDR estiver setado. Este *bit* é verificado pelo módulo CAN antes e após os dados serem armazenados no *buffer* de transmissão.

- **Bit 7 - ABO** - *Bit auto bus on.*

  0 - O módulo sairá do estado *bus-off* somente apósos 128x11 *bits* recessivos consecutivos no barramentoe após ter resetado o *bit* CCR.

  1 - Após o estado *bus-off*, o módulo volta ao estado *bus-on* após 128x11 *bits* recessivos consecutivos.

- **Bit 6 - STM** - *Bit* de requisição do modo *self-test.*

  0 - O módulo está no modo normal

  1 - O módulo está no modo *self-test.* Neste modo, o módulo CAN gera um sinal de reconhecimento. Deste modo, ele habilita uma operação sem o barramento está conectado ao módulo CAN. A mensagem não será enviada, mas será lida de volta e armazenada no MBOX apropriado. A manipulação de pacotes remotos com o *bit* AAM não é implementado nesse modo. O identificador da mensagem recebida não será armazenado no MBOX de recepção neste modo de operação.

- **Bits 1-0 - MBNR** - *Bit* que indica o número do MBOX (2 ou 3) que terá o seu campo de dados acessado para uma escrita. Se o valor desses *bits* for '10' o MBOX 2 será escolhido. Caso seja '11' o MBOX 3 será escolhido.

11.7 Registradores de Configuração do Bit - BCRn

Os registradores de configuração **BCR1** e **BCR2** são usados para configurar o nó CAN com os parâmetros de temporização apropriados. Pode-se escrever nesses registradores somente quando o módulo CAN estiver no modo de configuração.

O comprimento de um *bit* no barramento CAN é determinado pelos segmentos de tempo TSEG1, TSEG2 e pelo valor do *baud rate prescaler*, BRP. Todos os controladores do barramento CAN deverão ter o mesmo *baud rate* e o mesmo comprimento de *bit*. Nas figuras 11.16 e 11.17 estão ilustrados, respectivamente, os esquemas com as posições dos *bits* nos registradores **BCR2** e **BCR1**.

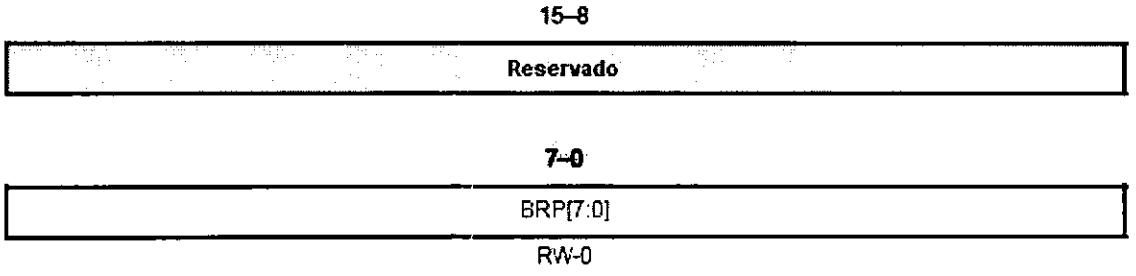


Figura 11.16: Esquema com as posições dos *bits* no registrador BCR2.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bits 7-0 - BRP** - Esses *bits* indicam o *baud rate prescaler*. Eles especificam a duração de um *time quantum* (TQ) do módulo CAN. O comprimento de um TQ é definido pela seguinte equação:

$$TQ = \frac{(BRP+1)}{(CLKOUT)} ns$$

Se BRP = **BCR2** = 0, então 1TQ = 1 ciclo de *clock* da CPU.

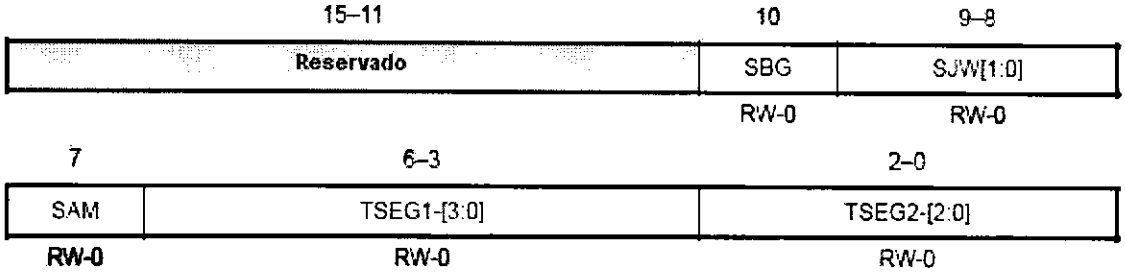


Figura 11.17: Esquema com as posições dos *bits* no registrador BCR1.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bit 10 - SBG** - Esse *bit* indica em qual borda será feita a re-sincronização.

  0 - A re-sincronização será feita apenas na borda de descida.

  1 - A re-sincronização será feita em ambas as bordas.

- **Bits 9-8 - SJW** - Esses *bits* indicam a largura do pulso de sincronização. SJW é programável de 1 à 4 TQ.

- **Bit 7 - SAM** - Esse *bit* indica quantas amostras serão feitas para determinar o nível do barramento.

  0 - O módulo CAN amostra apenas uma vez.

  1 - O módulo CAN amostra três vezes e decide de forma majoritória o nível do barramento.

- **Bits 6-3 - TSEG1** - Esses *bits* indicam o comprimento do segmento de tempo 1. O valor de TSEG1 é programável de 3 à 16 TQ, e deverá ser maior ou igual ao segmento de tempo 2.

$$TSEG1 = PROPSEG + FASESEG1$$

- **Bits 2-0 - TSEG2** - Esses *bits* indicam o comprimento do segmento de tempo 2. Quando a re-sincronização for feita apenas na borda de descida (SBG=0), o valor mínimo de TSEG2 é calculado pela seguinte equação:

$$TSEG2min = 1 + SJW$$

O valor de TSEG2 pode ser programado de 2 à 8 TQ, de acordo com a equação:

$$(SJW + SBG + 1) \leq TSEG2 \leq 8$$

O *baud rate* é calculado pela equação seguinte, em *bits* por segundo:

$$BAUDRATE = \frac{CLKOUT}{(BRP+1)*BitTime} = \frac{1}{(TQ*BitTime)}$$

Onde o *Bit Time* é o número de TQ por *bit*, e é dado pela seguinte equação:

$$BitTime = (TSEG1 + 1) + (TSEG2 + 1) + 1$$

Na tabela 11.1 está indicado alguns valores dos parâmetros de temporização com um CLKOUT de 40 MHz.

Tabela 11.1: Alguns dos valores dos parâmetros de temporização para um CLKOUT de 40 MHz.

TSEG1	TSEG2	Bit Time	BRP	SJW	SBG	Baud Rate
4	3	10	3	1	1	1 Mbps
14	6	23	17	4	1	0,096 Mbps

11.8 Registradores de Status

Existem dois registradores de *status*: o registrador de status de erro (**ESR**) e o registrador de *status* global (**GSR**)

11.8.1 Registrador ESR

Esse registrador é utilizado para indicar erros que ocorrem durante uma operação. Os *bits* desse registrador são zerados quando se escreve um '1' neles, exceto para o *flag* SA1, o qual é zerado quando um *bit* recessivo estiver no barramento. Na figura 11.18 está ilustrado um esquema com as posições dos *bits* no registrador **ESR**.

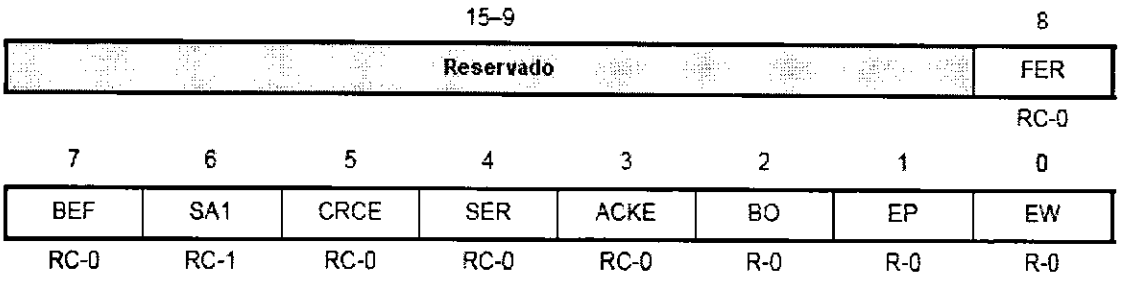


Figura 11.18: Esquema com as posições dos *bits* no registrador ESR.

OBS: **R** = acesso a leitura, **C** = pode apenas ser zerado, **-n** = valor após o *reset*.

- **Bit 8 - FER** - Esse *bit* indica se ocorreu um erro de formato.

  0 - Nenhum erro de formato ocorreu no barramento.

  1 - Ocorreu um erro de formato no barramento. Isto significa que um ou mais *bits* de formato fixo tem um valor diferente do determinado.

- **Bit 7 - BEF** - Esse *bit* indica se ocorreu um erro de *bit*.

  0 - Nenhum erro de *bit* ocorreu no barramento.

  1 - Ocorreu um erro de *bit* no barramento. Durante a transmissão do campo de arbitragem, um *bit* dominante foi enviado, mas um *bit* recessivo foi recebido.

- **Bit 6 - SA1** - Esse *bit* indica se o módulo CAN detectou um *bit* recessivo no barramento.

  0 - O módulo CAN detectou um *bit* recessivo.

  1 - O módulo CAN não detectou um *bit* recessivo.

- **Bit 5 - CRCE** - Esse *bit* indica se ocorreu um erro de CRC.

  0 - O módulo CAN não recebeu um CRC errado.

  1 - O módulo CAN recebeu um CRC errado.

- **Bit 4 - SER** - Esse *bit* indica se ocorreu uma violação da regra de inserção de *bits*.

  0 - Não ocorreu um erro de inserção de *bits*.

  1 - Ocorreu um erro de inserção de *bits*.

- **Bit 3 - ACKE** - Esse *bit* indica se ocorreu um erro de reconhecimento.

  0 - Não ocorreu um erro de reconhecimento.

  1 - Ocorreu um erro de reconhecimento.

- **Bit 2 - BO** - Esse *bit* Indica se o módulo CAN está no estado *bus-off*.

  0 - Operação normal.

  1 - Existe uma taxa anormal de erros ocorrendo no barramento CAN. Essa condição ocorre quando o TEC atingir o limite de 256 erros. Enquanto estiver no estado *bus-off*, nenhuma mensagem poderá ser recebida ou transmitida. Para sair desse estado, deve-se zerar o *bit* CCR ou então setar o *bit* ABO. Após sair do *bus-off* os contadores de erro são zerados.

- **Bit 1 - EP** - Esse *bit* indica se o módulo CAN está no estado passivo ao erro.

  0 - O módulo CAN não está no estado passivo ao erro.

  1 - O módulo CAN está no estado passivo ao erro.

- **Bit 0 - EW** - Esse *bit* indica se o módulo CAN está em estado de advertência.

  0 - O módulo CAN não está em estado de advertência. Os valores de ambos os contadores de erro são menores que 96.

  1 - O módulo CAN está em estado de advertência. Os valores de ambos os contadores de erro atingiram o nível de advertência de 96 erros.

11.8.2 Registrador GSR

Na figura 10-19 está ilustrado um esquema com as posições dos bits no registrador GSR.

15–8						
Reservado						
7–6	5	4	3	2	1	0
Reservado	SMA	CCE	PDA	**Reservado**	RM	TM
	R-0	R-1	R-0		R-0	R-0

Figura 11.19: Esquema com as posições dos *bits* no registrador GSR.

OBS: **R** = acesso a leitura, **-n** = valor após o *reset*.

- **Bit 5 - SMA** - *Bit* de reconhecimento do modo de suspensão.

  0 - O periférico CAN não está no modo de suspensão.

  1 - O periférico CAN entrou no modo de suspensão.

- **Bit 4 - CCE** - Esse *bit* habilita a mudança de configuração.

  0 - O acesso de escrita aos registradores de configuração é negado.

  1 - A CPU tem acesso de escrita aos registradores de configuração **BCR**, enquanto o *bit* CCR estiver setado.

- **Bit 3 - PDA** - *Bit* de reconhecimento do modo de baixa potência.

  0 - Operação normal.

  1 - O periférico CAN entrou no modo de baixa potência.

- **Bit 1 - RM** - Esse *bit* indica se o módulo CAN está recebendo alguma mensagem.

  0 - O módulo CAN não está recebendo uma mensagem.

  1 - O módulo CAN está recebendo uma mensagem.

- **Bit 0 - TM** - Esse *bit* indica se o módulo CAN está transmitindo alguma mensagem.

  0 - O módulo CAN não está transmitindo uma mensagem.

  1 - O módulo CAN está transmitindo uma mensagem.

11.9 Registadores CAN de Contagem de Erro - CEC

O módulo CAN possui dois contadores de erro de oito bits. O contador de erro de transmissão (TEC) e o contador de erros de recepção (REC). Na figura 11.20 está ilustrado um esquema com as posições dos bits no registrador CEC.

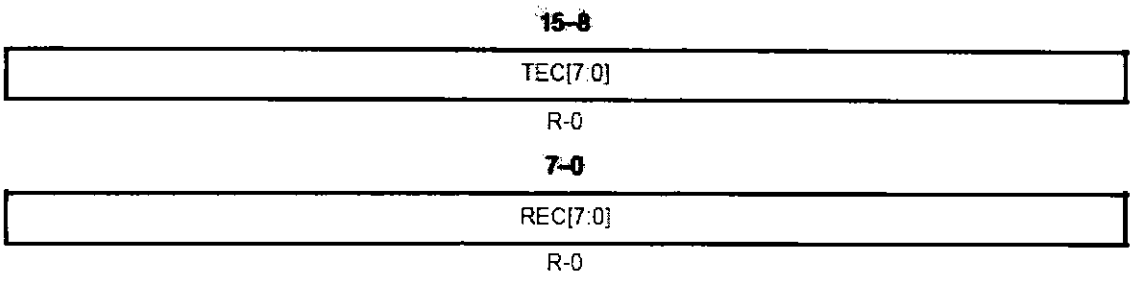


Figura 11.20: Esquema com as posições dos *bits* no registrador CEC.

OBS: **R** = acesso a leitura, **-n** = valor após o *reset*.

Após exceder o limite de erro passivo (128 erros), o REC não será mais incrementado. Quando uma mensagem for recebida corretamente, o contador será setado para um valor entre 119 e 127. Após atingir o estado *bus-off*, o TEC será indefinido, enquanto que REC será zerado e sua função será alterada: ele será incrementado sempre que ocorrer 11 consecutivos *bits* recessivos no barramento. Esses 11 *bits* correspondem a diferença (espaço) entre dois telegramas no barramento CAN. Se o contador de recepção atingir o valor 128, o módulo mudará automaticamente de volta ao estado *bus-on* se o *bit* ABO estiver setado. Caso contrário, ele mudará quando a sequência desejada de 11x128 *bits* for finalizada e o *bit* CCR for resetado pelo DSP. Todos os *flags* internos são resetados e os contadores de erro são zerados. Os contadores de erro são também zerados, quando o módulo CAN entra no modo de configuração.

11.10 Lógica de Interrupção

Existem duas requisições de interrupção do periférico CAN para o controlador PIE: a interrupção do *mailbox* e a interrupção de erro. Existem dois registradores que controlam as interrupções do periférico CAN, os registradores **CAN_IFR** e o **CAN_IMR**.

11.10.1 Registrador CAN_IFR

Esse registrador contém os *flags* de interrupção do periférico CAN. Os flags de interrupção são setados se a correspondente condição de interrupção ocorrer. A apropriada requisição de interrupção do *mailbox* será assegurada somente se a correspondente máscara de interrupção no registrador **CAN_IMR** estiver setado. A requisição de interrupção permanecerá ativa até que o *flag* de interrupção seja zerado pela CPU, escrevendo um '1' no apropriado *bit*. Um reconhecimento da interrupção não zera os *flags* de interrupção. Os *flags* MIFx não podem ser zerados por uma escrita no registrador **CAN_IFR**, eles deverão ser zerados escrevendo um '1' no apropriado *bit* TAn para os MBOX de transmissão, ou no *bit* RMPn para os MBOX de recepção.

De modo a receber futuras interrupções, o *flag* da atual interrupção deverá ser zerado imediatamente após entrar na rotina de serviço de interrupção. Na figura 11.21 está ilustrado um esquema com as posições dos bits no registrador CAN_IFR.

15–14	13	12	11	10	9	8
Reservado	MIF5	MIF4	MIF3	MIF2	MIF1	MIF0
	R-0	R-0	R-0	R-0	R-0	R-0
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reservado	RMLIF	AAIF	WDIF	WUIF	BOIF	EPIF	WLIF
	RC-0	RC-0	RC-0	RC-0	RC-0	RC-0	RC-0

Figura 11.21: Esquema com as posições dos *bits* no registrador CAN_IFR.

OBS: **R** = acesso a leitura, **C** = acesso a leitura, zerado, **-n** = valor após o *reset*.

- **Bits 13-8 - MIFx** - *Flag* de interrupção do *mailbox* (recepção/transmissão).

  0 - Nenhuma mensagem foi transmitida ou recebida.

  1 - O correspondente *mailbox* transmitiu ou recebeu alguma mensagem com sucesso.

  Cada um dois seis *mailbox* podem iniciar uma interrupção. Essas interrupções podem ser uma interrupção de recepção ou de transmissão dependendo da configuração do MBOX. Se um dos MBOX estiver configurado como MBOX de requisição remota (AAM=1) e um pacote de requisição for recebido, uma interrupção de transmissão será setada após esse MBOX enviar o correspondente pacote de dados. Se um pacote de requisição for enviado, uma interrupção de recepção será setada após o desejado pacote de dados ser recebido. Existe uma máscara de interrupção para cada um desses bits. Se o *bit* MIF do registrador **CAN_IFR** for setado e o respectivo *bit* MIM do registrador **CAN_IMR** estiver setado, uma interrupção será gerada.

- **Bit 6 - RMLIF** - Esse *bit* indica que uma mensagem foi perdida.

  0 - Nenhuma mensagem foi perdida.

  1 - Uma condição de *overflow* ocorreu em um dos MBOX. Uma mensagem foi perdida.

- **Bit 5 - AAIF** - Esse *bit* indica um aborto de reconhecimento.

  0 - Nenhuma transmissão foi abortada.

  1 - Uma transmissão foi abortada.

- **Bit 4 - WDIF** - Esse *bit* indica que um acesso de escrita foi negado.

  0 - O acesso de escrita ao MBOX foi feito com sucesso.

  1 - O acesso de escrita ao MBOX não foi permitido.

- **Bit 3 - WUIF** - Esse *bit* indica que ocorreu uma operação de *wake-up*.

  0 - O módulo CAN permanece no modo *sleep* ou no modo normal.

  1 - O módulo CAN saiu do modo *sleep*.

- **Bit 2 - BOIF** - Esse *bit* indica que o módulo CAN está no estado *bus-off*.

  0 - O módulo CAN está no estado *bus-on*.

  1 - O módulo CAN entrou no estado *bus-off*.

- **Bit 1 - EPIF** - Esse *bit* indica que o módulo CAN está no estado passivo ao erro.

  0 - O módulo CAN não está no estado passivo ao erro.

  1 - O módulo CAN está no estado passivo ao erro.

- **Bit 0 - WLIF** - Esse *bit* indica que o módulo CAN atingiu o nível de advertência.

  0 - Nenhum dos contadores de erro atingiu o nível de advertência.

  1 - Um dos contadores de erro atingiu o nível de advertência.

11.10.2 Registrador CAN_IMR

O registrador **CAN_IMR** é idêntico ao **CAN_IFR**, exceto que existem mais dois *bits* que selecionam a prioridade da interrupção. Na figura 11.22 está ilustrado um esquema com as posições dos bits no registrador **CAN_IMR**.

15	14	13	12	11	10	9	8
MIL	Reservado	MIM5	MIM4	MIM3	MIM2	MIM1	MIM0
RW-0		RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7	6	5	4	3	2	1	0
EIL	RMLIM	AAIM	WDIM	WUIM	BOIM	EPIM	WLIM
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Figura 11.22: Esquema com as posições dos *bits* no registrador CAN_IMR.

OBS: **R** = acesso a leitura, **C** = acesso a leitura, zerado, **-n** = valor após o *reset*.

- **Bit 15 - MIL** - Esse *bit* indica o nível de prioridade da interrupção do *mailbox* (MIF5-MIF0).

  0 - O MBOX gera uma requisição de interrupção de alta prioridade.

  1 - O MBOX gera uma requisição de interrupção de baixa prioridade.

- **Bits 13-8 - MIMx** - *Bits* de máscara das interrupções MIFx.

  0 - Desabilita a interrupção.

  1 - Habilita a interrupção.

- **Bit 7 - EIL** - Esse *bit* indica o nível de prioridade das interrupções de erro.

  0 - O MBOX gera uma requisição de interrupção de alta prioridade.

  1 - O MBOX gera uma requisição de interrupção de baixa prioridade.

- **Bits 6-0** - *Bits* de máscara das interrupções de erro.

  0 - Desabilita a interrupção.

  1 - Habilita a interrupção.

11.11 Modo de Configuração

O modo de configuração é o modo no qual o módulo CAN deverá ser inicializado. Para entrar no modo de configuração deve-se setar o *bit* CCR do registrador **MCR**. A inicialização pode ser desempenhada somente quando o *bit* CCE do registrador **GSR** estiver setado. Quando CCE for igual a '1', os *bits* dos registradores **BCR1** e **BCR2** podem ser escritos. O módulo entrará no modo normal novamente, quando CCR for zerado. Na figura 11.23 está ilustrado um fluxograma do processo de requisição do modo de configuração.

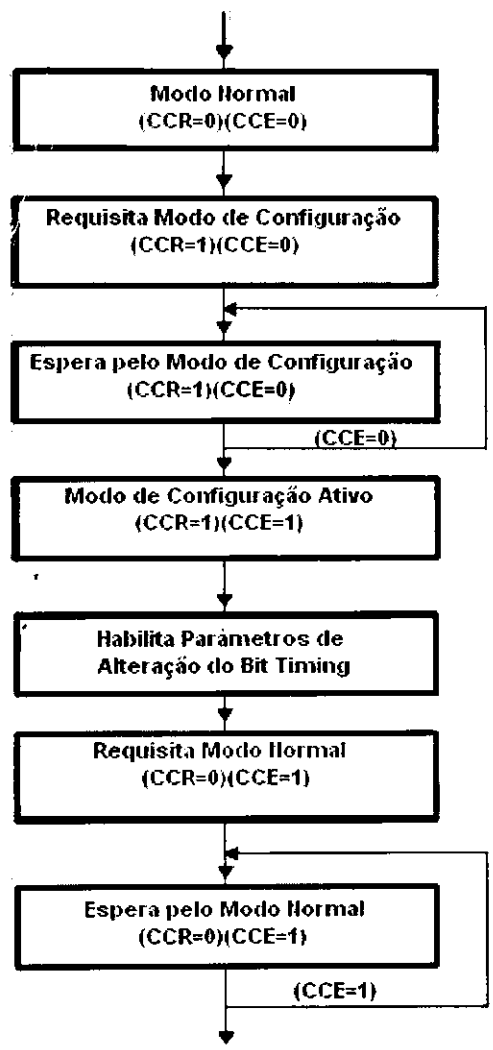


Figura 11.23: Fluxograma do processo de requisição do modo de configuração.

11.12 Modo de Baixa Potência

Antes da CPU entrar no modo IDLE, o modo de baixa potência do dispositivo que potencialmente desliga os *clocks* do dispositivo, deverá ser requisitado pelo periférico CAN, setando o *bit* PDR do registrador **MCR**. Se o módulo estiver transmitindo uma mensagem quando PDR for setado, ele entrará no modo de baixa potência somente quando a transmissão for concluída com sucesso, se houver uma perca de arbitragem ou se ocorrer uma condição de erro no barramento. Quando o módulo estiver pronto para entrar no modo de baixa potência, o *bit* de *status* PDA será setado. O módulo apenas deverá entrar no modo IDLE quando o bit PDA estiver setado.

Para sair do modo de baixa potência, o *bit* PDR deverá ser zerado por *software*, ou automaticamente, se o *bit* WUBA estiver setado e existir atividade no barramento CAN. Quando um sinal dominante for detectado no barramento, o *flag* de interrupção de *wake-up* (WUIF) será setado.

Um *wake-up* automático com a atividade no barramento pode ser habilitado ou desabilitado através do *bit* WUBA. Se houver alguma atividade nas linhas do barramento CAN, o módulo iniciará a sua sequência *power-up*. O módulo espera que 11 *bits* recessivos consecutivos sejam detectados no pino Rx e inicia a atividade no barramento. A primeira mensagem, na qual inicia a atividade no barramento, não poderá ser recebida.

Quando o *bit* WUBA for habilitado, a interrupção de erro WUIF será assegurada automaticamente pelo controlador PIE, no qual manipula uma interrupção de *wake-up* e reinicia os *clocks* do dispositivo se eles estiverem parados.

Após sair do modo *sleep* com um *wake-up*, os *bits* PDR e PDA são zerados. Os contadores de erro CAN permanecem inalterados.

11.13 Modo de Suspensão

O modo de suspensão pode operar em ambos os modos *free* ou *soft*. O modo de suspensão entra em operação quando a CPU ativa o sinal SUSPEND. O *bit* SUSP indica em qual dos modos (*free* ou *soft*) o módulo será operado.

Quando o módulo entra no modo de suspensão *soft*, o *bit* SMA do registrador GSR será setado. Se o módulo estiver transmitindo uma mensagem quando o sinal SUSPEND for ativado, ele entrará no modo de suspensão somente quando a transmissão for concluída com sucesso, se houver uma perca de arbitragem ou se ocorrer uma condição de erro no barramento. Caso contrário, o módulo entrará no modo de suspensão imediatamente após o *bit* SMA ser setado.

Quando no modo de suspensão o módulo não envia e nem recebe mensagens. Os contadores de erro e os demais registradores não são alterados. Se o módulo CAN estiver no estado *bus-off* quando o modo de suspensão for requisitado, ele entrará no modo de suspensão imediatamente. Os contadores de erro são indefinidos nesse estado e os flag *bus-off* e passivo ao erro são setados.

O módulo sairá desse modo quando o sinal SUSPEND for desativado. Ele espera pelos próximos 11 *bits* recessivos no barramento e então volta para o modo normal de operação. Quando no modo *bus-off*, ele deverá esperar pela condição de *bus-on*.

Capítulo 12

Watchdog Timer

12.1 Introdução

O *watchdog* (WD) *timer* monitora as operações do *software* e do *hardware* e implementa funções de *reset* do sistema. Se o *software* entrar em um *loop* impróprio ou se a CPU se "perder"temporariamente, um *overflow* do *watchdog timer* assegurará um *reset* do sistema. O WD *timer* aumenta a confiabilidade da CPU e deste modo assegura a integridade do sistema.

12.2 Características do Watchdog Timer

O módulo *watchdog timer* possui as seguintes características:

- O contador do WD de oito *bits* gera um *reset* do sistema quando ocorrer um *overflow*;
- O Contador *free-running* de seis *bits* que alimenta o contador do WD via o *prescaler* do contador do WD;
- Um registrador chave de *reset* **(WDKEY)** que zera o contador do WD quando a correta combinação de valores for escrita, e gera um *reset* se um incorreto valor for escrito no registrador;
- O WD verifica os *bits* que iniciam um *reset* do sistema se o WD for corrompido;
- Ativação automatica do WD *timer* uma vez que o *reset* do sistema foi liberado;
- Um *prescaler* do WD com seis seleções do contador *free-running* de seis *bits*.

Na figura 12.1 está ilustrado o diagrama de blocos do módulo WD.

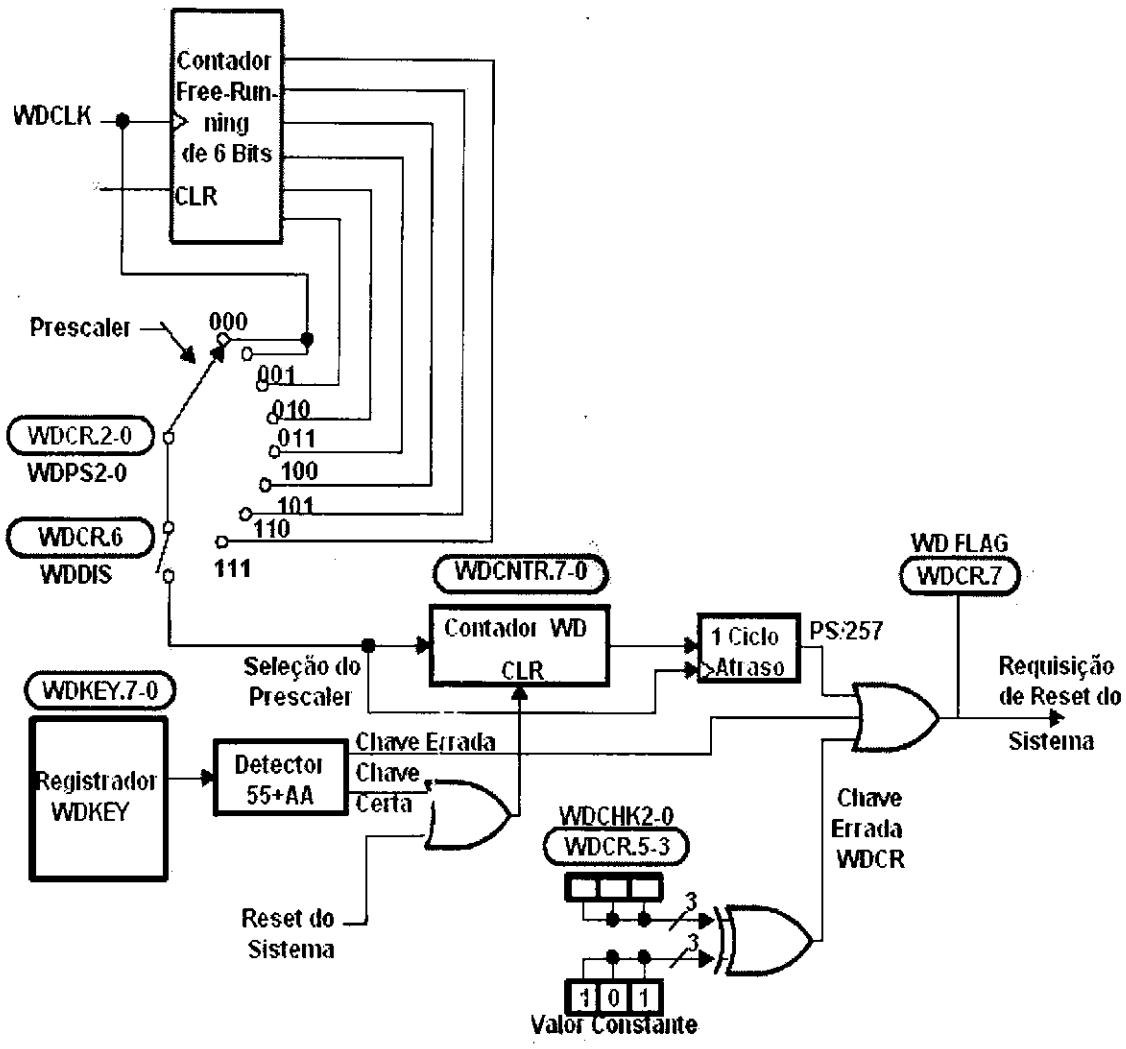


Figura 12.1: Diagrama de blocos do módulo WD.

12.3 Operações do Watchdog Timer

Três registradores controlam as operações do WD *timer*:

- **Registrador WDCNTR** - Este registrador contém o valor do contador do WD *timer* de oito *bits*;
- **Registrador WDKEY** - Este registrador zera o **WDCNTR** quando um valor 55h seguido por um valor AAh for escrito em **WDKEY**.
- **Registrador WDCR** - Este registrador contém os *bits* de controle usados na configuração do WD *timer*.

O *clock* do WD *timer* (WDCLK) é um *clock* de baixa frequência. WDCLK possui uma frequência nominal de 78125 Hz quando o CPUCLK for igual a 40 MHz. WDCLK é derivado do *clock* da CPU. Isto garante que o WD continua a contar quando a CPU está no modo IDLE1 ou IDLE2 (menos no modo HALT). A frequência do WD *timer* é calculado da seguinte equação:

$$WDCLK = \frac{CLKOUT}{512}$$

WDCLK pode ser visto no pino CLKOUT quando o *watchdog* for habilitado. Se o WD for habilitado, o seu contador deverá ser resetado antes de ocorrer um *overflow*. Caso contrário, o DSP será resetado.

O WDCLK será parado quando o sinal de suspensão da CPU estiver ativo. O *clock* do WD não funciona quando o monitor em tempo real estiver sendo executado.

O **WDCNTR** pode ser guiado diretamente do sinal WDCLK ou através de um dos seis *taps* do contador *free-running*. O contador *free-running* de seis *bits* incrementa continuamente por uma taxa fornecida pelo WDCLK. Um dos seis *taps* (ou a entrada direta do WDCLK) pode ser selecionada pelos *bits* de seleção do *prescaler* (*bits* WDPS2-0) como entrada para a base de tempo do **WDCNTR**. Este prescale fornece taxas de *overflow* do WD de 3,28 à 209,7 ms para um WDCLK de 78125 Hz. Enquanto o *chip* estiver no modo normal de operação, o contador *free-running* não poderá ser parado ou resetado, exceto por um *reset* do sistema. O fato de zerar o **WDCNTR** não zera o contador *free-running*.

O **WDCNTR** será resetado quando a sequência apropriada for escrita no registrador **WDKEY** antes de ocorrer um *overflow* do **WDCNTR**. O **WDCNTR** será habilitado para ser resetado quando o valor 55h for escrito em **WDKEY**. Após isso, a escrita do valor AAh em **WDKEY** resetará o **WDCNTR**. Qualquer valor escrito em **WDKEY** diferente de 55h ou AAh causará um *reset* do sistema.

Um *overflow* do **WDCNTR** ou um incorreto valor escrito em **WDKEY**, setam o *flag* WDFLAG. Após um *reset*, o programa deverá ler este *flag* para determinar a fonte do *reset* e em seguida deverá por *software* resetar este *flag* para permitir que a fonte de subsequentes *resets* sejam determinados.

Quando ocorre um *overflow* do **WDCNTR**, o WD *timer* assegura um *reset* do sistema. O *reset* ocorre após um ciclo de *clock* do WDCLK. Para desenvolvimentos de *software* e programação da *flash*, o WD *timer* pode ser desabilitado setando o *bit* WDDIS no registrador **WDCR**.

Os *bits* WDCHK do registrador **WDCR** são continuamente comparados ao valor constante '101' (em binário). Se os valores de WDCHK não forem iguais, um *reset* do sistema será gerado. Isto funciona como uma lógica de verificação no caso de ocorrer escritas impróprias no registrador **WDCR**, ou se um estimulo externo (tais como saltos de tensão, EMI ou outras fontes) corromper o conteúdo de **WDCR**.

12.4 Registradores de Controle do Watchdog

12.4.1 Registrador de Contagem do WD - WDCNTR

O **WDCNTR** é um registrador de oito *bits* que contém o valor do contador do WD. Este registrador é incrementado continuamente por uma taxa selecionada através do registrador de controle do WD. Quando ocorre um *overflow* do **WDCNTR**, um adicional ciclo de atraso será inserido antes do *reset* do sistema ser assegurado. Na figura 12.2 está ilustrado um esquema com a posição dos *bits* no registrador WDCNTR.

7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Figura 12.2: Esquema com a posição dos *bits* no registrador WDCNTR.

OBS: **R** = acesso a leitura, **-n** = valor após o *reset*.

- **Bits 7-0 - D7-D0** - Contém o valor do contador de oito *bits* do WD.

12.4.2 Registrador da Chave do Reset - WDKEY

Na figura 12.3 está ilustrado um esquema com a posição dos bits no registrador WDKEY.

7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Figura 12.3: Esquema com a posição dos *bits* no registrador WDKEY.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **-n** = valor após o *reset*.

- **Bits 7-0 - D7-D0** - Contém o valor da chave do *reset* do WD.

12.4.3 Registrador de Controle do Timer do WD - WDCR

Na figura 12.4 está ilustrado um esquema com a posição dos bits no registrador WDCR.

7	6	5	4	3	2	1	0
WDFLAG	WDDIS	WDCHK2	WDCHK1	WDCHK0	WDPS2	WDPS1	WDPS0
RC-x	RWc-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Figura 12.4: Esquema com a posição dos *bits* no registrador WDCR.

OBS: **R** = acesso a leitura, **W** = acesso a escrita, **C** = zerado pela escrita de um '1', **Wc** = acesso a escrita condicionado ao valor do *bit* WD OVERRIDE que deve ser igual a '1', **-n** = valor após o *reset*, **-x** = valor após o *reset* determinado pela ação do WD *timer*.

- **Bit 7 - WDFLAG** - *Flag* do WD. Este *bit* indica se um *reset* do sistema foi assegurado pelo WD *timer*.

  0 - Indica que o WD *timer* não assegurou um *reset* do sistema.

  1 - Indica que o WD *timer* assegurou um *reset* do sistema.

- **Bit 6 - WDDIS** - Desabilita o watchdog. Este *bit* pode ser escrito apenas quando o *bit* WD OVERRIDE do registrador **SCSR2** for '1'.

  0 - WD está habilitado.

1 - WD está desabilitado.

- **Bit 5 - WDCHK2** - *Bit* 2 da verificação do WD. Deve-se escrever um '1' nesse *bit* quando o registrador **WDCR** for escrito. Caso contrário, um *reset* do sistema será assegurado.

  0 - *Reset* do sistema está assegurado.

  1 - Continua operando normalmente se todos os *bits* de verificação foram escrito corretamente.

- **Bit 4 - WDCHK1** - *Bit* 1 da verificação do WD. Deve-se escrever um '0' nesse *bit* quando o registrador **WDCR** for escrito. Caso contrário, um *reset* do sistema será assegurado.

  0 - Continua operando normalmente se todos os *bits* de verificação foram escrito corretamente.

  1 - *Reset* do sistema está assegurado.

- **Bit 3 - WDCHK0** - *Bit* 0 da verificação do WD. Deve-se escrever um '1' nesse *bit* quando o registrador **WDCR** for escrito. Caso contrário, um *reset* do sistema será assegurado.

  0 - *Reset* do sistema está assegurado.

  1 - Continua operando normalmente se todos os *bits* de verificação foram escrito corretamente.

- **Bits 2-0 - WDPS2-WDPS0** - *Bits* de seleção do *prescaler* do *watchdog timer*. Na tabela 12.1 está indicado os possíveis valores dos *bits* WDPS com os respectivos valores do *prescaler*, para um CLKOUT de 40 MHz. Na mesma tabela estão indicados a frequência do *overflow* e o tempo mínimo de ocorrência do *overflow*.

Tabela 12.1: Possíveis valores dos *bits* WDPS com os respectivos valores do *prescaler*, frequência do *overflow* e do tempo mínimo de ocorrência do *overflow*, para um CLKOUT de 40 MHz.

WDPS2-0	WDCLK dividido por	Frequência de overflow	Overflow mínimo (ms)
00x	1	305,2	3,28
010	2	152,6	6,6
011	4	76,3	13,1
100	8	38,1	26,2
101	16	19,1	52,4
110	32	9,5	104,9
111	64	4,8	209,7

Capítulo 13

Placa de Desenvolvimento - eZdsp LF2407A

13.1 Introdução

O eZdsp LF2407A é uma placa que utiliza o processador digital de sinais TMS320LF2407A para desenvolvimento de aplicações na engenharia. Essa placa possui as seguintes características:

- Taxa de operação de até 40 MIPS (Milhões de Instruções Por Segundo);
- 64 K*words* de memória RAM de programa/dados *on board*;
- 32 K*words* de memória Flash interno ao *chip*;
- Circuito de *clock* interno ao *chip* de 10 MHz;
- Três conectores de expansão (Analógico, I/O e expansão);
- Controlador JTAG IEEE 1149.1 *on board*;
- Alimentação de 5 Volts;
- *Driver* para as ferramentas do TI *Code Composer*;
- Conector para emulação JTAG IEEE 1149.1 *on board.*

Na figura 13.1 está ilustrado o layout do eZdsp LF2407A.

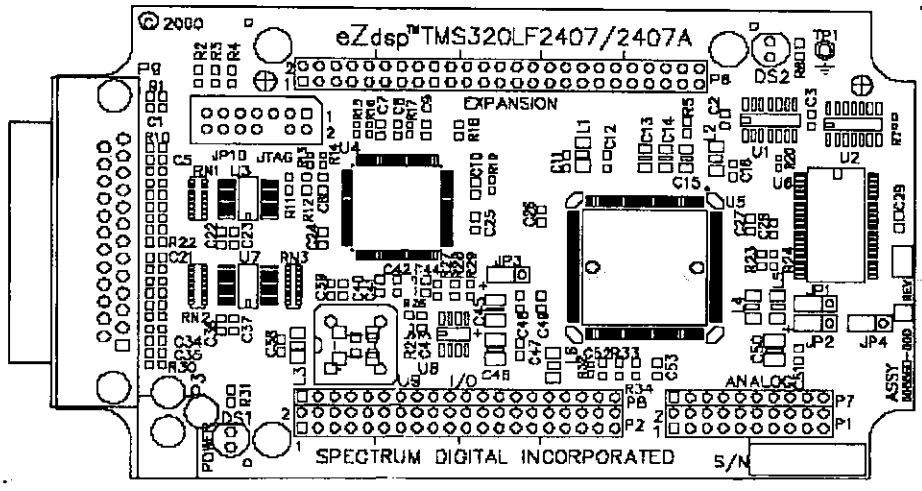


Figura 13.1: Layout do eZdsp LF2407A.

13.2 Conectores do eZdsp LF2407A

Existem oito conectores na placa. A função de cada conector está indicada na tabela 13.1.

Tabela 13.1: Conectores do eZdsp LF2407A.

Conector	Função
P1/P7	*Interface* analógica
P2/P8	*Interface* I/O
P3	Conector da fonte
P6	Conector de expansão I/O
P9	Porta paralela/*Interface* do controlador JTAG
P10	*Interface* JTAG

Na figura 13.2 está ilustrado um *layout* da placa indicando a posição de cada conector.

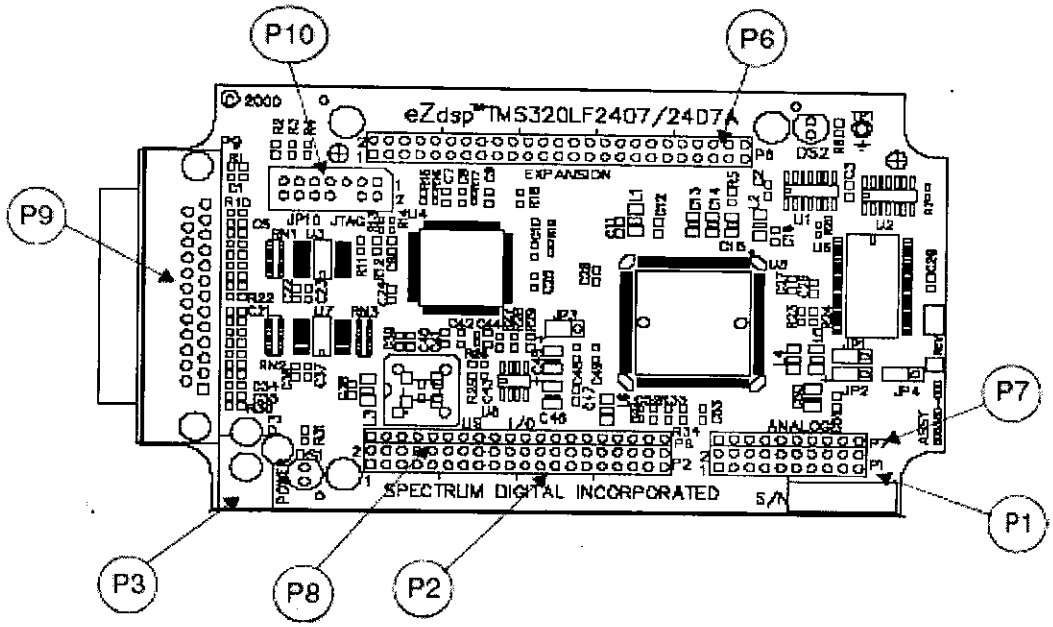


Figura 13.2: Layout do eZdsp LF2407A indicando a posição de cada conector.

13.2.1 Conectores P1/P7 - Interface Analógica

As posições dos 20 pinos do conector P1 e os 10 pinos do conector P7 estão ilustrados na figura 13.3.

ANALOG

P7	1	2	3	4	5	6	7	8	9	10
P1	2	4	6	8	10	12	14	16	18	20
	1	3	5	7	9	11	13	15	17	19

Figura 13.3: Posição dos pinos nos conectores P1 e P7.

O nome de cada pino dos conectores P1 e P7 estão indicados, respectivamente, nas tabelas 13.2 e 13.3.

Tabela 13.2: Conector P1 - *Interface* analógica.

Pino	Sinal	Pino	Sinal
1	GND	2	ADCIN0
3	GND	4	ADCIN1
5	GND	6	ADCIN2
7	GND	8	ADCIN3
9	GND	10	ADCIN4
11	GND	12	ADCIN5
13	GND	14	ADCIN6
15	GND	16	ADCIN7
17	GND	18	VREFLO
19	GND	20	VREFHI

Tabela 13.3: Conector P1 - *Interface* analógica.

Pino	Sinal
1	ADCIN8
2	ADCIN9
3	ADCIN10
4	ADCIN11
5	ADCIN12
6	ADCIN13
7	ADCIN14
8	ADCIN15
9	Reservado
10	Reservado

13.2.2 Conectores P2/P8 - Interface I/O

As posições dos 40 pinos do conector P2 e os 20 pinos do conector P8 estão ilustrados na figura 13.4.

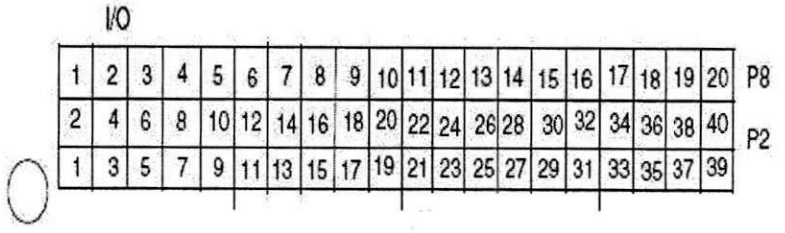


Figura 13.4: Posição dos pinos nos conectores P2 e P8.

O nome de cada pino dos conectores P2 e P8 estão indicados, respectivamente, nas tabelas 13.4 e 13.5.

Tabela 13.4: Conector P2 - *Interface* I/O.

Pino	Sinal	Pino	Sinal
1	+5V	2	+5V
3	SCITXD/IOPA0	4	SCIRXD/IOPA1
5	XINT1-/IOPA2	6	CAP1/QEP1/IOPA3
7	CAP2/QEP2/IOPA4	8	CAP3/IOPA5
9	PWM1/IOPA6	10	PWM2/IOPA7
11	PWM3/IOPB0	12	PWM4/IOPB1
13	PWM5/IOPB2	14	PWM6/IOPB3
15	T1PWM/T1CMP/IOPB4	16	T2PWM/T2CMP/IOPB5
17	TDIRA/IOPB6	18	TCLKINA/IOPB7
19	GND	20	GND
21	WNR/IOPC0	22	BIO-/IOPC1
23	SPISIMO/IOPC2	24	SPISOMI/IOPC3
25	SPICLK/IOPC4	26	SPISTE/IOPC5
27	CANTX/IOPC6	28	CANRX/IOPC7
29	CLKOUT/IOPE0	30	PWM7/IOPE1
31	PWM8/IOPE2	32	PWM9/IOPE3
33	PWM10/IOPE4	34	PWM11/IOPE5
35	PWM12/IOPE6	36	CAP4/QEP3/IOPE7
37	PDPINTA-	38	PDPINTB-
39	GND	40	GND

Tabela 13.5: Conector P8 - *Interface* I/O.

Pino	Sinal
1	+5V
2	XINT2-/ADCSOC/IOPD0
3	EMU0/IOPD1
4	EMU1/IOPD2
5	TDCK/IOPD3
6	TDI/IOPD4
7	TDO/IOPD5
8	TMS/IOPD6
9	TMS2/IOPD7
10	GND
11	CAP5/QEP4/IOPF0
12	CAP6/IOPF1
13	I3PWM/T3CMP/IOPF2
14	I4PWM/T4CMP/IOPF3
15	TDIRB/IOPF4
16	TCLKINB/IOPF5
17	IOPF6
18	Reservado
19	Reservado
20	GND

13.2.3 Conector P3 - Fonte

O eZdsp LF2407A é alimentado por uma fonte de 5 Volts via o conector P3. Na figura 13.5 está ilustrado a vista frontal do conector P3.

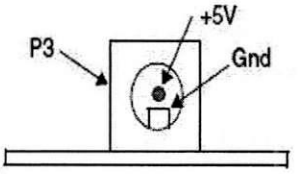


Figura 13.5: Vista frontal do conector P3 - Fonte.

13.2.4 Conector P6 - Conector de Expansão

As posições dos 50 pinos do conector P6 são ilustrados na figura 13.6.

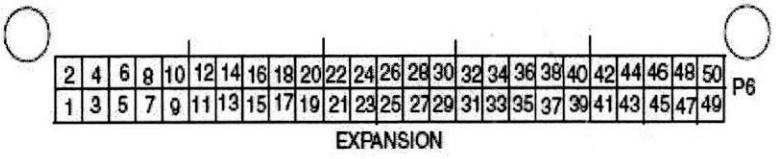


Figura 13.6: Posições dos pinos do conector P6.

O nome de cada pino do conector P6, o qual possui os sinais de *interface* com a memória, são descritos na tabela 13.6. O pino 47 (RS-) é um pino bidirecional, deverá ser guiado com coletor aberto. O pino 45 (3.3V) não pode ser utlizado como fonte de potência.

Tabela 13.6: Conector P6 - *Interface* de Expansão.

Pino	Sinal	Pino	Sinal
1	+5V	2	+5V
3	D0	4	D1
5	D2	6	D3
7	D4	8	D5
9	D6	10	D7
11	D8	12	D9
13	D10	14	D11
15	D12	16	D13
17	D14	18	D15
19	A0	20	A1
21	A2	22	A3
23	A4	24	A5
25	A6	26	A7
27	A8	28	A9
29	A10	30	A11
31	A12	32	A13
33	A14	34	A15
35	GND	36	GND
37	PS-	38	DS-
39	READY	40	IS-
41	R/W-	42	STRB-
43	WE-	44	RD-
45	3.3V	46	Reservado
47	RS-	48	Reservado
49	GND	50	GND

13.2.5 Conector P9 - Porta Paralela/Interface JTAG

O eZdsp LF2407A um dispositivo de interface para a porta paralela e o JTAG. Este dispositivo um padrão de interface da porta paralela que suporta comunicações ECP, EPP e SPP8/bidirecional. O dispositivo tem acesso direto a interface integrada JTAG.

13.2.6 Conector P10 - Interface JTAG

As posições dos 14 pinos do conector P10 estão ilustrados a figura 13.7.

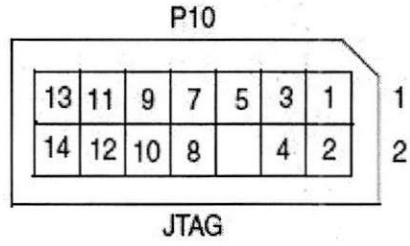


Figura 13.7: Posições dos pinos do conector P10.

O nomes de cada pino do conector P10, no qual possui os sinais JTAG estão indicados na tabela 13.7

Tabela 13.7: Conector P6 - Interface JTAG.

Pino	Sinal	Pino	Sinal
1	TMS	2	TRST-
3	TDI	4	GND
5	PD (+5V)	6	Sem pino
7	TDO	8	GND
9	TCK-RET	10	GND
11	TCK	12	GND
13	EMU0	14	EMU1

13.3 Jumpers

O eZdsp LF2407A tem quatro *jumpers*. Eles determinam as características que serão usadas. Na tabela 13.8 estão listados os *jumpers* e suas funções.

Tabela 13.8: Jumpers do eZdsp LF2407A.

Jumper	Função	Posição Inicial
JP1	Fonte do VREFHI	1-2
JP2	Fonte do VREFLO	1-2
JP3	Seleciona o Vpp	1-2
JP4	Modo MP/MC	1-2

Cada *jumper* do eZdsp LF2407A é um *jumper* 1x3. Cada *jumper* 1x3 deverá ser selecionado na posição 1-2 ou 2-3. Na figura 13.8 está ilustrado o *layout* de um *jumper* 1x3.

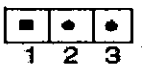


Figura 13.8: Layout de um jumper 1x3.

As posições dos quatro *jumpers* no eZdsp LF2407A estão ilustradas na figura 13.9.

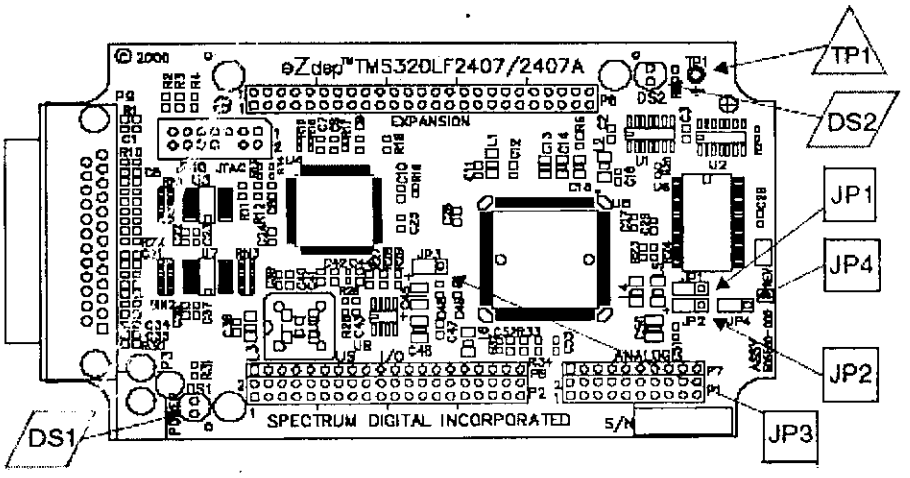


Figura 13.9: Layout do eZdsp LF2407A com as posições dos quatro jumpers.

13.3.1 Jumper 1 - JP1

O *jumper* 1 é utilizado para selecionar a fonte do sinal VREFHI. Se a posição 1-2 foi selecionada o sinal VREFHI será fornecido por uma fonte interna a placa de 3.3 Volts. Se a posição 2-3 for selecionada, o sinal VREFHI será fornecido por uma fonte externa conectada ao pino 20 do conector P1 (*Interface* analógica). Na tabela 13.9 está indicado às possíveis posições do jumper 1 e suas respectivas funções.

Tabela 13.9: Posições do jumper 1.

Posição	Função
1-2	VREFHI interno
2-3	VREFHI externo

13.3.2 Jumper 2 - JP2

O *jumper* 1 é utilizado para selecionar a fonte do sinal VREFLO. Se a posição 1-2 foi selecionada o sinal VREFHI será fornecido por uma fonte interna a placa de 0 Volts. Se a posição 2-3 for selecionada, o sinal VREFLO será fornecido por uma fonte externa conectada ao pino 18 do conector P1 (*Interface* analógica). Na tabela 13.10 está indicado às possíveis posições do jumper 2 e suas respectivas funções.

Tabela 13.10: Posições do jumper 2.

Posição	Função
1-2	VREFLO interno
2-3	VREFLO externo

13.3.3 Jumper 3 - JP3

O *jumper* 3 é utilizado para selecionar a tensão de programação da FLASH. A posição 1-2 remov Vpp do DSP e desabilita a programação. Se a posição 2-3 for selecionada, a tensão de programação da FLASH será habilitada. Na tabela 13.11 está indicado às possíveis posições do jumper 3 e suas respectivas funções.

Tabela 13.11: Posições do jumper 3.

Posição	Função
1-2	Vpp desabilitado
2-3	Vpp habilitado

13.3.4 Jumper 4 - JP4

O *jumper* 4 é utilizado para selecionar o modo de operação do TMS320LF2407A. Se a posição 1-2 foi selecionada o DSP funcionará no modo microprocessador. Se a posição 2-3 for selecionada, o DSP funcionará no modo microcontrolador. Na tabela 13.12 está indicado às possíveis posições do jumper 4 e suas respectivas funções.

Tabela 13.12: Posições do jumper 4.

Posição	Função
1-2	Microprocessador
2-3	Microcontrolador

13.4 LEDs

Existem dois LEDs no eZdsp LF2407A. O LED DS1 indica que a placa está sendo alimentada com +5 Volts. O LED DS2 é controlado por *software* através do pino IOPC0 do DSP. Na tabela 13.13 está indicado os LEDs com os seus respectivos sinais de controle.

Tabela 13.13: Sinais de controle dos LEDs.

LED	Sinal de Controle
DS1	+5 Volts
DS2	IOPC0

13.5 Ponta de teste

O eZdsp LF2407A tem uma ponta de teste (TP1). Essa ponta de teste está ligada ao GND da placa.

Capítulo 14

Code Composer

14.1 Introdução

Neste capítulo apresenta-se os conceitos básicos do *Code Composer*. Maiores informações sobre seus utilitários podem ser adquiridos nas referências [8] e [9].

O *Code Composer* é o ambiente usado para desenvolvimento de projetos com os processadores digitais de sinais da Texas Instruments. Na figura 14.1 apresenta-se a tela do ambiente *Code Composer*.

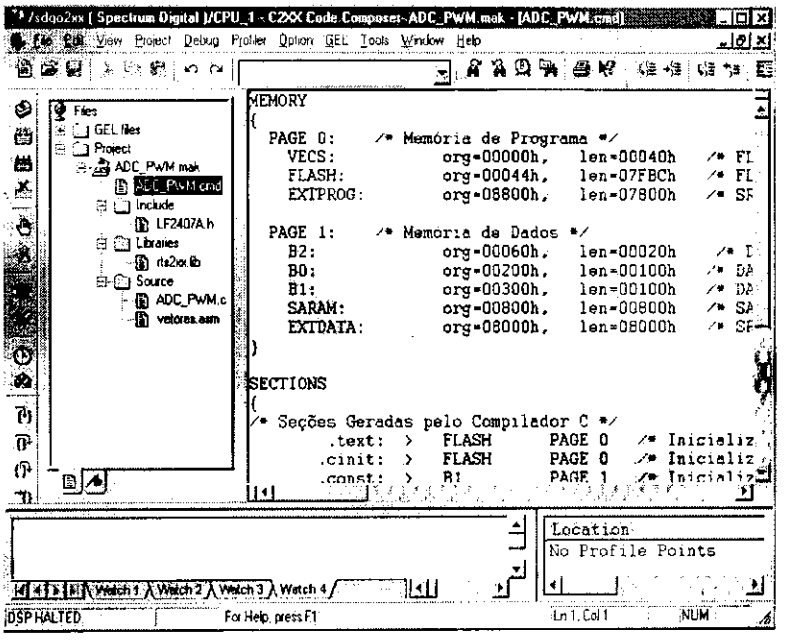


Figura 14.1: Tela do ambiente *Code Composer*.

Esse ambiente oferece todas as ferramentas necessárias para editar programas em linguagem C e *assembly*, compilar, carregar e executar os programas no DSP, monitorar, gerenciar e coletar estatísticas de áreas específicas do programa. Possui uma avançada *interface* para análise de sinais que permite monitorar sinais críticos em aplicações de processamento de imagem, comunicações *wireless* e outras aplicações. O *Code Composer* oferece também uma linguagem de programação similar a linguagem C, a linguagem GEL (*Go DSP Extension Language*), na qual pode-se criar funções para aumentar os utilitários do *Code Composer*.

14.2 Desenvolvimento de Projetos

Para desenvolver projetos utilizando o *Code Composer*, deve-se seguir os seguintes passos:

1. **Criar um projeto:** *Project* ⇒ *New*. Dê um nome ao seu projeto e o salve em uma pasta com seus projetos. Por exemplo, PWM.mak. O arquivo do projeto deverá ter a extensão .mak. Quando o projeto for criado, aparecerá na janela de arquivos várias pastas (*include*, *libraries* e *source*). Na figura 14.2 está ilustrado a janela com os arquivos do projeto.

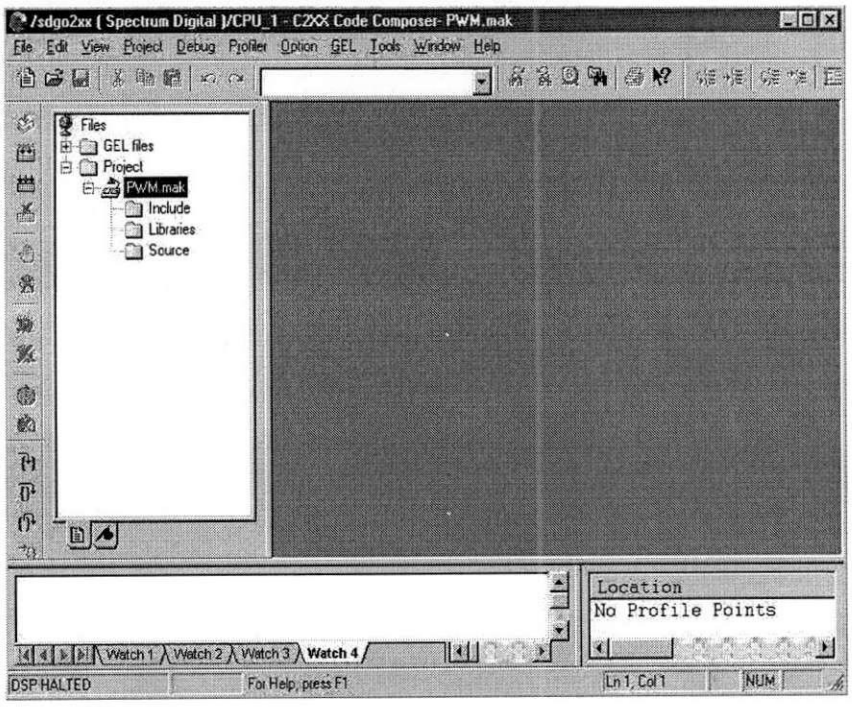


Figura 14.2: Tela do ambiente *Code Composer* com janela dos arquivos do projeto.

2. **Criar o arquivo do programa principal:** *File* ⇒ *New* ⇒ *Source File*. Dê um nome ao arquivo do programa principal e o salve na pasta do projeto. Por exemplo, PWM.c. O arquivo do programa principal, se for escrito em linguagem C, possui a extensão .c. Em seguida adicione-o ao projeto: na janela com os arquivos do projeto, dê um *click* com o botão direito do *mouse* em cima do ícone do projeto PWM.mak e em seguida *click* em *Add Files*. Então, selecione o arquivo PWM.c e o adicione ao projeto. Na figura 14.3 está ilustrado esse processo.

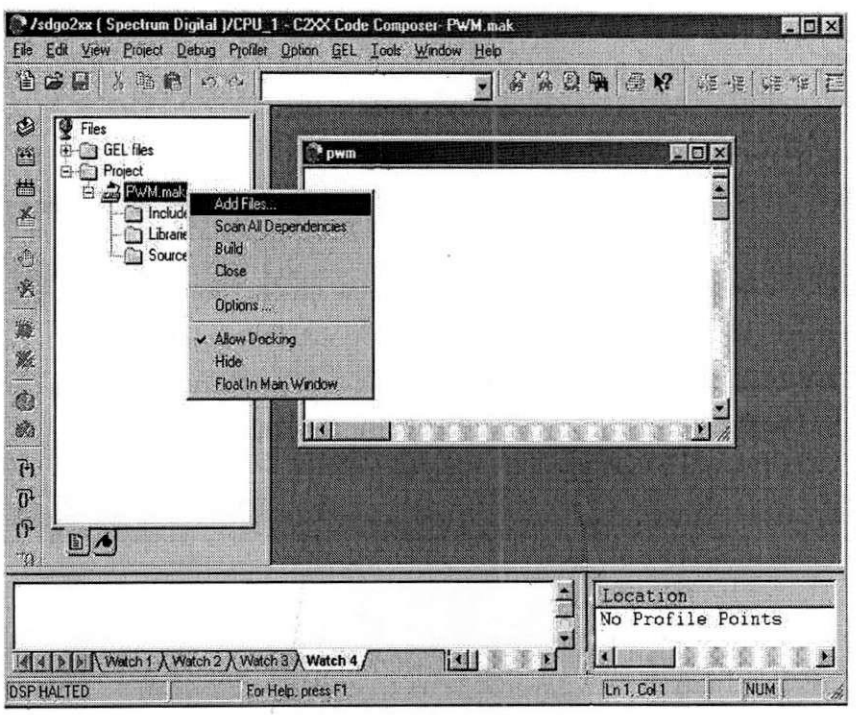


Figura 14.3: Tela do ambiente *Code Composer* com o procedimento para adicionar um arquivo ao projeto.

3. **Criar o arquivo de vetores:** *File* ⇒ *New* ⇒ *Source File*. Dê um nome ao arquivo de vetores e o salve na pasta do projeto. Por exemplo, vetores.asm. O arquivo de vetores deve ser escrito em linguagem *assembly* e portanto terá a extensão .asm. Em seguida adicione-o ao projeto;

4. **Criar o arquivo de ligação:** *File* ⇒ *New* ⇒ *Source File*. Dê um nome ao arquivo de ligação e o salve na pasta do projeto. Por exemplo, PWM.cmd. O arquivo de ligação deverá ter a extensão .cmd. Em seguida adicione-o ao projeto;

5. **Criar o arquivo de cabeçario:** *File* ⇒ *New* ⇒ *Source File*. Dê um nome ao arquivo de cabeçario e o salve na pasta do projeto. Por exemplo, LF2407A.h. Esse arquivo contém a definição dos endereços dos registradores do DSP que serão usados no programa principal. Dê um *click* com o botão direito do mouse em cima do ícone do projeto PWM.mak e em seguida *click* em *Scan All Dependencies*. Na figura 14.4 está ilustrado esse processo.

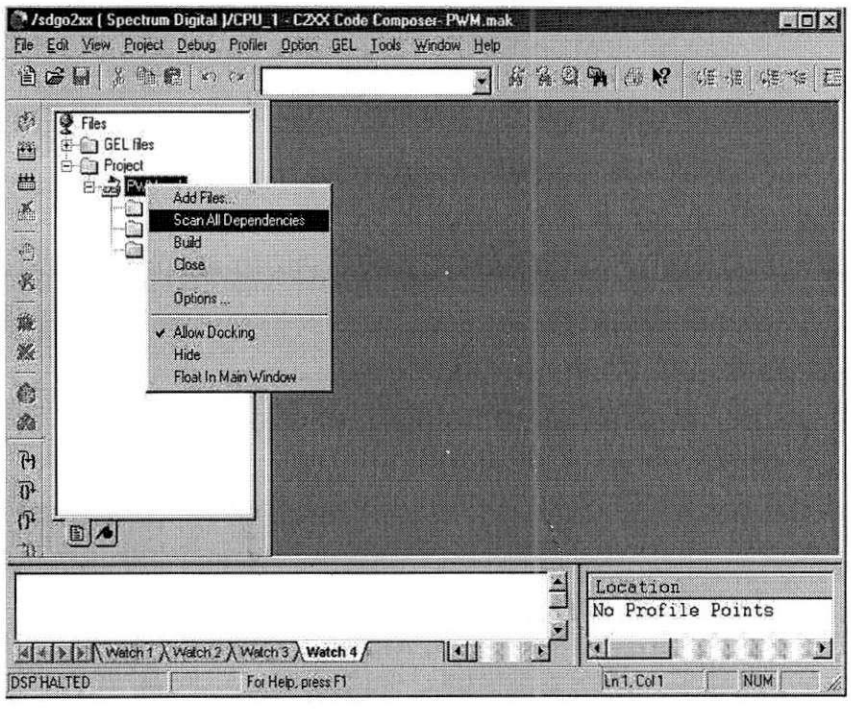


Figura 14.4: Tela do ambiente *Code Composer* com o procedimento para verificar todas dependências.

6. **Adicionar a biblioteca rts2xx.lib**: na janela com os arquivos do projeto, dê um *click* com o botão direito do mouse em cima do ícone do projeto PWM.mak e em seguida *click* em *Add Files*. O arquivo rts2xx.lib está normalmente localizado no seguinte caminho C:\tic2xx\c2000\cgtool \lib.

14.2.1 Executando o Programa

Após compilar os programas, deve-se seguir os seguintes passos para executá-los no DSP:

1. **Construir o programa executável:** *Project* ⇒ *Rebuild All.* Se o programa não conter erros o programa executável com a extensão .out será gerado. No nosso exemplo o arquivo PWM.out será gerado;
2. **Carregar o programa no DSP:** *File* ⇒ *Load Program.* Selecione o programa executável;
3. **Resetar o DSP:** *Debug* ⇒ *ResetDSP*;
4. **Executar o programa:** *Debug* ⇒ *RUN.*

Para parar a execução do programa temporariamente, deve-se fazer:

Debug ⇒ *Halt.*

Para finalizar a execução do programa, deve-se seguir os seguintes passos:

1. *Debug* ⇒ *Halt*;
2. *Debug* ⇒ *Restart*;
3. *Debug* ⇒ *Reset* DSP.

14.2.2 Monitorando Variáveis

A monitoração das variáveis do programa e dos próprios registradores do DSP pode ser realizada através da janela *Watch Window*.

Para monitorar uma variável do programa deve-se seguir os seguintes passos:

1. *View* ⇒ *Watch Window.* A janela *watch window* será aberta;
2. Dê um *click* com o botão direito do *mouse* em cima da janela *watch window* e selecione *Insert New Expression.* Na figura 14.5 está ilustrado a tela do *Code Composer* com a janela Watch Window no momento da realização desse procedimento;

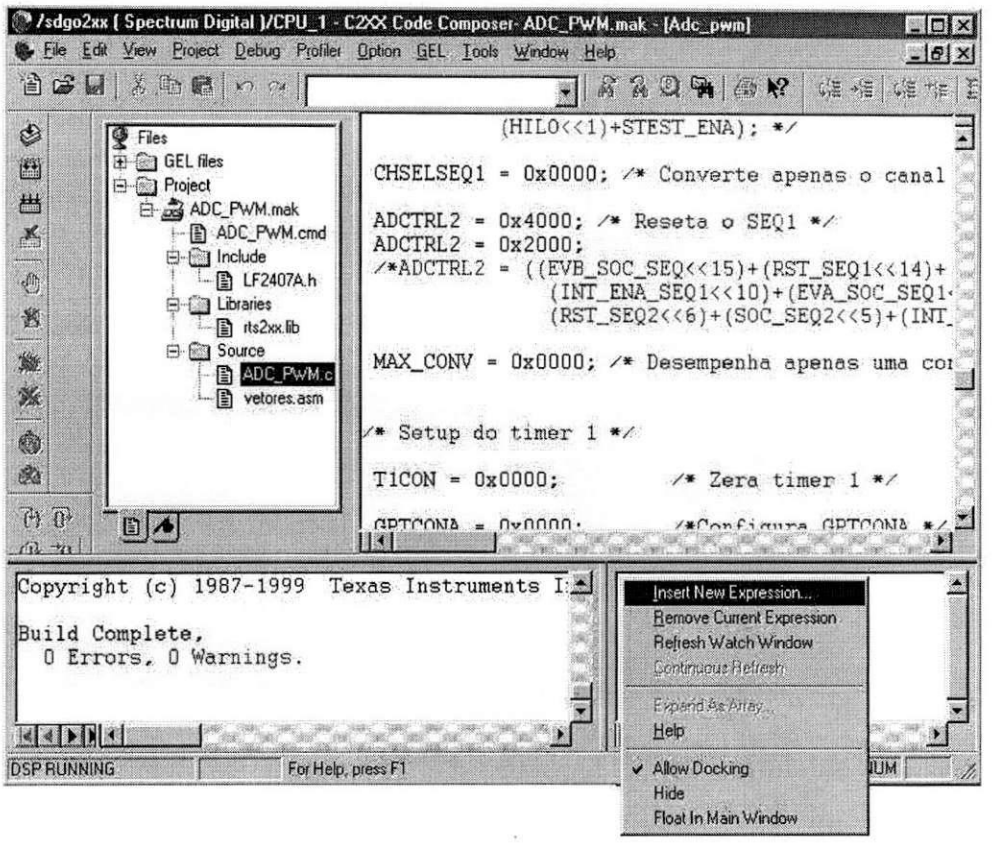


Figura 14.5: Tela do ambiente *Code Composer* com a janela *Watch Window* e o procedimento para monitorar uma variável.

Quando a opção *Insert New Expression* for selecionada, a janela *Watch Add Expression* aparecerá na área de trabalho do *Code Composer*. Na figura 14.6 apresenta-se a tela do *Code Composer* com a janela *Watch Add Expression* no momento da realização desse procedimento;.

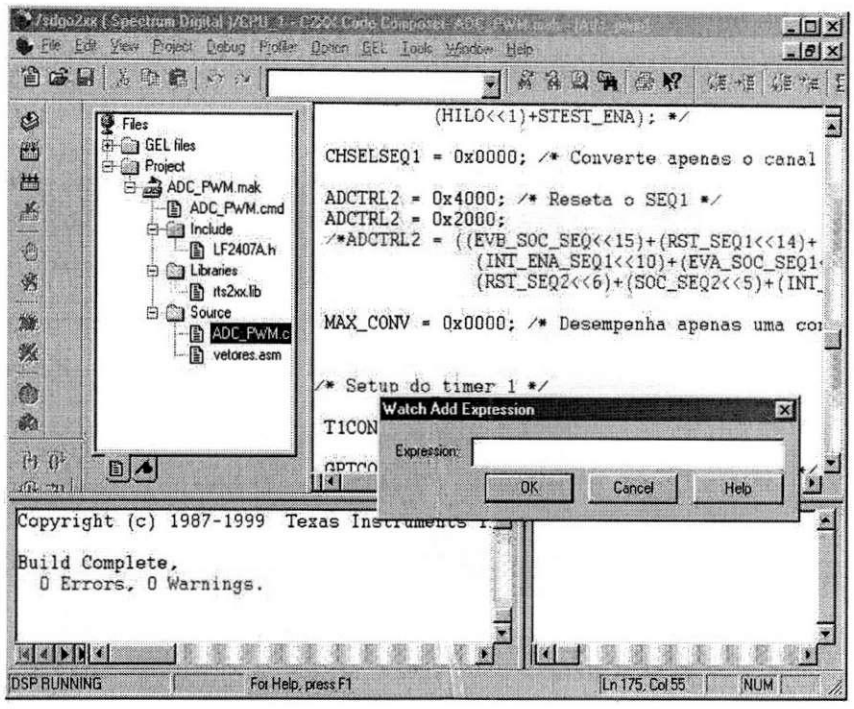


Figura 14.6: Tela do ambiente *Code Composer* com a janela *Watch Add Expression*.

Em seguida digite o nome da variável que será monitorada no campo *Expression*. O valor dessa variável aparecerá na janela *Watch Window*. Para atualizar o valor dessa variável, dê um *click* com o botão direito do *mouse* em cima da váriavel e *click* em *Refresh Watch Window*. Para remover essa variável ou expressão, repita o mesmo procedimento e *click* em *Remove Current Expression*. è possível monitorar mais de uma variável ao mesmo tempo.

O procedimento para monitorar o valor de um ou mais registradores é análogo ao descrito anteriomente. Porém, deve-se digitar o seguinte comando no campo *Expression* para visualizar o valor do registrador em hexadecimal:

***(int*)endereço do registrador@data,x;nome do registrador**

Por exemplo, para monitorar o valor do registrador T1CON, o qual possui o endereço 0x7404, deve-se digtar o seguinte comando:

***(int*)0x7404@data,x;T1CON**

Na figura 14.7 está ilustrado a tela do *Code Composer* com a janela *Watch Window* mostrando o valor do registrador **T1CON**.

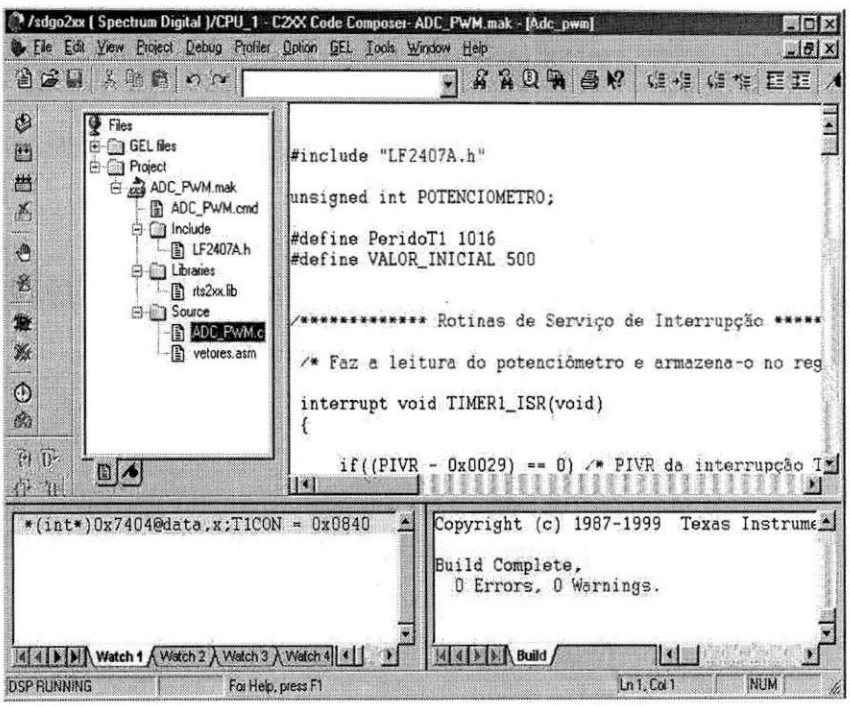


Figura 14.7: Tela do ambiente *Code Composer* com a janela *Watch Window* mostrando o valor do registrador T1CON.

Capítulo 15

Aplicações

15.1 Introdução

Nesta seção apresenta-se um código básico para a inicialização e operação do TMS320LF2407A. O programa exemplo foi escrito em linguagem C e executado na placa de desenvolvimento eZdsp LF2407A, assim como os demais programas exemplos desse capítulo. Os programas foram editados no *software Code Composer* v4.12.

15.1.1 Visão Geral do Programa Exemplo

O programa exemplo desempenha as seguintes tarefas no DSP:

- Configura os dois registradores de *status* e controle do sistema;
- Desabilita *watchdog timer*;
- Configura a *interface* de memória externa;
- Configura os pinos compartilhados do dispositivo;
- Configura o *timer 1* e as unidades de comparação para fornecer um sinal PWM no pino PWM1 com ciclo de trabalho de 25%;
- Configura os registradores de interrupção do gerenciador de eventos e habilita as interrupções globais;

- A rotina de serviço de interrupção do *timer 1* desempenha as seguintes atividades:
  - Salva e re-armazena o contexto usando uma pilha de *software*;
  - A cada 200 ms liga/desliga um LED vermelho rotulado na placa de desenvolvimento como DS2. Esse LED é controlado pelo pino IOPC0 (pino 0 da porta C).

Em adição, o programa exemplo ensina os seguintes conceitos:

- Como trabalhar com vários arquivos;
- Como trabalhar com várias seções de código;
- Como escrever uma rotina de serviço de interrupção usando a linguagem C;
- Como inserir um código *assembly* dentro de um programa escrito em linguagem C;
- Como acessaros registradores dos periféricos usando linguagem C;
- Como usar arquivos *include*;
- Como construir tabelas de vetores de interrupção.

15.1.2 Programa Exemplo

O programa exemplo consiste dos seguintes arquivos:

1. **vetor.asm:** Tabela de vetores de interrupção.
2. **exemplo.c:** Programa principal.
3. **LF2407A.h:** Arquivo contendo as definições dos endereços dos registradores dos periféricos.
4. **exemplo.cmd:** Arquivo de ligação dos comandos.
5. **exemplo.mak:** Arquivo de projeto do *Code Composer* v4.12.
6. **exemplo.out:** Programa executável.
7. **exemplo.map:** Arquivo do mapa de memória criado pelas ferramentas de geração do código.

Apenas os cinco primeiros arquivos listados são necessários para construir o programa. Os dois últimos são criados pelas ferramentas de geração do código. Na figura 15.1 está ilustrado o fluxograma de execução do programa exemplo.

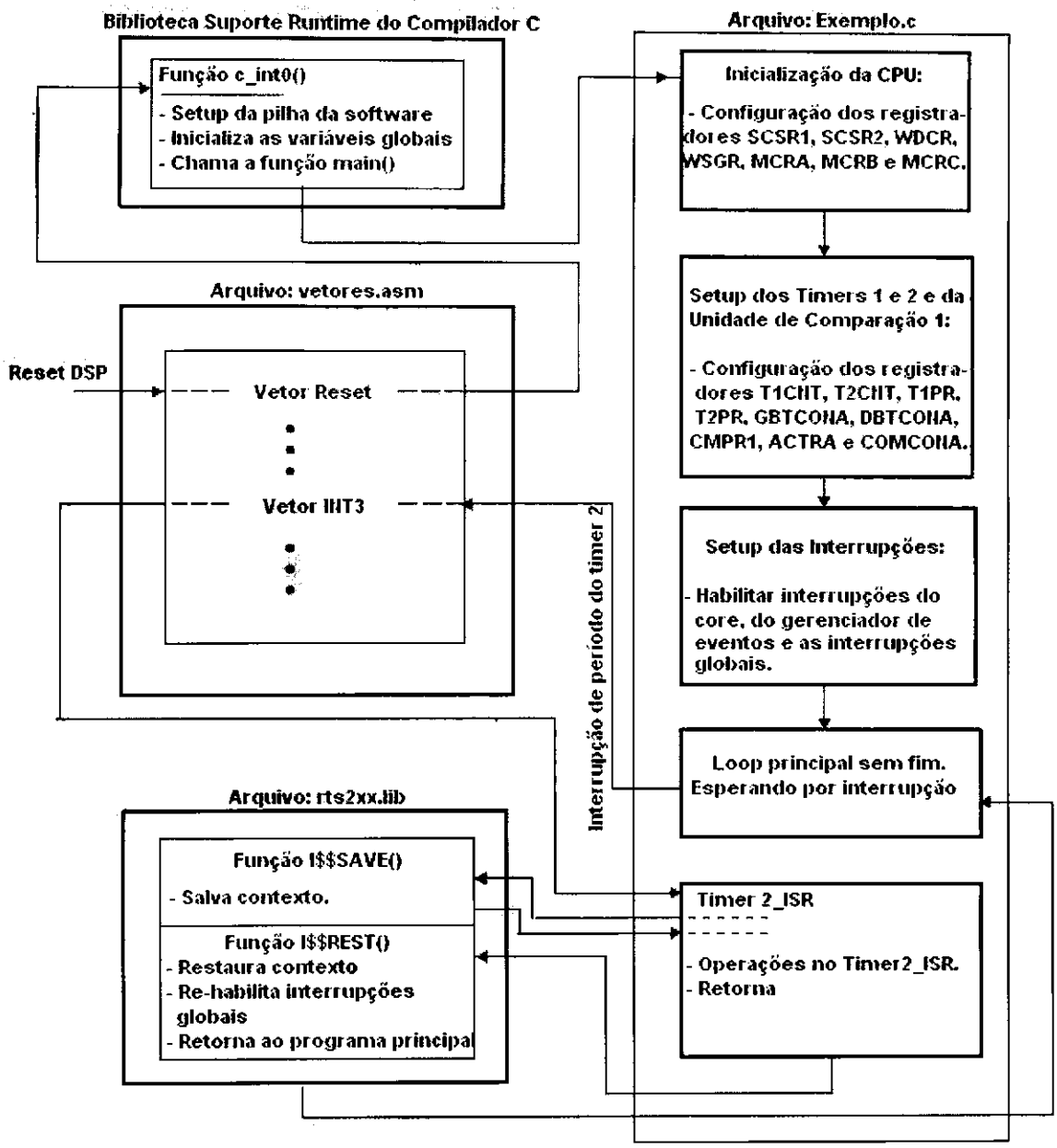


Figura 15.1: Fluxograma de execução do programa exemplo.

A execução do código começa no vetor *reset* do arquivo **vetores.asm** após um *reset* do *hardware*. Este vetor desvia para a função **c_int0** da biblioteca **rts2xx.lib**. Esta biblioteca vêm junto com as ferramentas de geração do código, e deverá ser adicionada em todos os projetos com programas feitos em linguagem C. A função c_int0 ajusta a pilha C, inicializa valores para todas as variáveis globais e estáticas e finalmente chama a função principal. A função c_int0 é documentada na referência [16].

Estando na função principal, o processador será inicializado e em seguida entrará no laço principal infinito. O DSP será interrompido periodicamente e uma rotina de serviço de interrupção (ISR) será executada. o ISR salva e re-armazena o contexto, re-habilita interrupções globais e retorna ao laço principal infinito.

15.1.3 Tabela de Vetores de Interrupção - Vetores.asm

No programa exemplo, o arquivo vetor.asm contém a tabela de vetores de interrupção. Esse arquivo deve ser escrito em linguagem *assembly* em um arquivo separado e deverá ser adicionado ao projeto do *Code Composer*. Esse arquivo será ligado automaticamente pelas ferramentas de geração do código durante a construção do projeto.

Arquivo - vetores.asm

```
.ref _c_int0, _dummy, _timer2_ISR
.sect "vetores"
rset: B _c_int0 ;00h reset
int1: B _dummy ;02h INT1
int2: B _dummy ;04h INT2
int3: B _timer2_ISR ;06h INT3
int4: B _dummy ;08h INT4
int5: B _dummy ;0Ah INT5
int6: B _dummy ;0Ch INT6
int7: B _dummy ;0Eh Reservado
int8: B _dummy ;10h INT8 (software)
int9: B _dummy ;12h INT9 (software)
int10: B _dummy ;14h INT10 (software)
int11: B _dummy ;16h INT11 (software)
```

```
int12: B _dummy ;18h INT12 (software)
int13: B _dummy ;1Ah INT13 (software)
int14: B _dummy ;1Ch INT14 (software)
int15: B _dummy ;1Eh INT15 (software)
int16: B _dummy ;20h INT16 (software)
int17: B _dummy ;22h TRAP
int18: B _dummy ;24h NMI
int19: B _dummy ;26h Reservado
int20: B _dummy ;28h INT20 (software)
int21: B _dummy ;2Ah INT21 (software)
int22: B _dummy ;2Ch INT22 (software)
int23: B _dummy ;2Eh INT23 (software)
int24: B _dummy ;30h INT24 (software)
int25: B _dummy ;34h INT26 (software)
int27: B _dummy ;36h INT27 (software)
int28: B _dummy ;38h INT28 (software)
int29: B _dummy ;3Ah INT29 (software)
int30: B _dummy ;3Ch INT30 (software)
int31: B _dummy ;3Eh INT31 (software)
```

Nesse código, utilizou-se duas diretivas: as diretivas **.ref** e **.sect**.

A diretiva **.ref** é utilizada para referenciar funções definidas externamente por uma arquivo fonte. No programa, a função **c_int0** é declarada na biblioteca **rts2xx.lib** e a função **_timer2_ISR** é declarada no arquivo do programa principal (**Exemplo.c**). Observe que no início das funções existe um *under-score* (**_c_int0()**, **_timer2_ISR()**, **_dummy()**). O *undescore* deve ser utilizado em todos os símbolos que são definidos em linguagem C, mas que são referenciados em linguagem *assembly*. A referência [17] informa mais detalhes sobre essa diretiva.

A diretiva **.sect** é usada para declarar uma seção inicializada. Neste caso, os vetores de interrupção estão sendo colocados na seção chamada vetores, de modo que eles serão ligados para uma faixa específica de endereços. A seção de vetores está listada em *SECTIONS* (ver arquivo **exemplo.cmd**). Observe no programa que estamos utilizando apenas duas interrupções. A interrupção rset que aponta para função **c_int0**, e a interrupção INT3 que aponta para a função **timer2_ISR**. Neste caso, INT3 possui apenas a interrupção do *timer 1*, caso tivesse várias outras, deveria ser feito um tratamento por *software* para descobrir qual interrupção ocorreu. Veja também que as interrupções que não estão sendo utilizadas apontam para a função **dummy**. Essa função não realiza nenhuma atividade. Caso alguma interrupção não esperada seja gerada, a função **dummy** garantirá o *loop* principal infinito.

15.1.4 Descrição do Arquivo - LF2407A.h

Esse arquivo contém a definição dos endereços dos registradores do TMS320LF2407A. O arquivo **LF2407A.h** deve ser criado em um arquivo separado e adicionado ao projeto do *Code Composer*.

Arquivo - LF2407A.h

```
/* Registradores do Core */
#define IMR *(volatile unsigned int *)0x0004
#define GREG *(volatile unsigned int *)0x0005
#define IFR *(volatile unsigned int *)0x0006
/* Registradores de Configuração do Sistema e das Interrupções */
#define PIRQR0 *(volatile unsigned int *)0x7010
#define PIRQR1 *(volatile unsigned int *)0x7011
#define PIRQR2 *(volatile unsigned int *)0x7012
#define PIACKR0 *(volatile unsigned int *)0x7014
#define PIACKR1 *(volatile unsigned int *)0x7015
```

```
#define PIACKR2 *(volatile unsigned int *)0x7016
#define SCSR1 *(volatile unsigned int *)0x7018
#define SCSR2 *(volatile unsigned int *)0x7019
#define DINR *(volatile unsigned int *)0x701C
#define PIVR *(volatile unsigned int *)0x701E
/* Registradores do Watchdog timer (WD) */
#define WDCNTR *(volatile unsigned int *)0x7023
#define WDKEY *(volatile unsigned int *)0x7025
#define WDCR *(volatile unsigned int *)0x7029
/* Registradores da SPI */
#define SPICCR *(volatile unsigned int *)0x7040
#define SPICTL *(volatile unsigned int *)0x7041
#define SPISTS *(volatile unsigned int *)0x7042
#define SPIBRR *(volatile unsigned int *)0x7044
#define SPIRXEMU *(volatile unsigned int *)0x7046
#define SPIRXBUF *(volatile unsigned int *)0x7047
#define SPITXBUF *(volatile unsigned int *)0x7048
#define SPIDAT *(volatile unsigned int *)0x7049
#define SPIPRI *(volatile unsigned int *)0x704F
/* Registradores da SCI */
#define SCICCR *(volatile unsigned int *)0x7050
#define SCICTL1 *(volatile unsigned int *)0x7051
#define SCIHBAUD *(volatile unsigned int *)0x7052
#define SCILBAUD *(volatile unsigned int *)0x7053
#define SCICTL2 *(volatile unsigned int *)0x7054
#define SCIRXST *(volatile unsigned int *)0x7055
#define SCIRXEMU *(volatile unsigned int *)0x7056
#define SCIRXBUF *(volatile unsigned int *)0x7057
```

```
#define SCITXBUF *(volatile unsigned int *)0x7059
#define SCIPRI *(volatile unsigned int *)0x705F
/* Registradores de Configuração das Interrupções Externas */
#define XINT1CR *(volatile unsigned int *)0x7070
#define XINT2CR *(volatile unsigned int *)0x7071
/* Registradores da Portas de I/O */
#define MCRA *(volatile unsigned int *)0x7090
#define MCRB *(volatile unsigned int *)0x7092
#define MCRC *(volatile unsigned int *)0x7094
#define PADATDIR *(volatile unsigned int *)0x7098
#define PBDATDIR *(volatile unsigned int *)0x709A
#define PCDATDIR *(volatile unsigned int *)0x709C
#define PDDATDIR *(volatile unsigned int *)0x709E
#define PEDATDIR *(volatile unsigned int *)0x7095
#define PFDATDIR *(volatile unsigned int *)0x7096
/* Registradores do ADC */
#define ADCTRL1 *(volatile unsigned int *)0x70A0
#define ADCTRL2 *(volatile unsigned int *)0x70A1
#define MAX_CONV *(volatile unsigned int *)0x70A2
#define CHSELSEQ1 *(volatile unsigned int *)0x70A3
#define CHSELSEQ2 *(volatile unsigned int *)0x70A4
#define CHSELSEQ3 *(volatile unsigned int *)0x70A5
#define CHSELSEQ4 *(volatile unsigned int *)0x70A6
#define AUTO_SEQ_SR *(volatile unsigned int *)0x70A7
#define RESULT0 *(volatile unsigned int *)0x70A8
#define RESULT1 *(volatile unsigned int *)0x70A9
#define RESULT2 *(volatile unsigned int *)0x70AA
#define RESULT3 *(volatile unsigned int *)0x70AB
```

```
#define RESULT4 *(volatile unsigned int *)0x70AC
#define RESULT5 *(volatile unsigned int *)0x70AD
#define RESULT6 *(volatile unsigned int *)0x70AE
#define RESULT7 *(volatile unsigned int *)0x70AF
#define RESULT8 *(volatile unsigned int *)0x70B0
#define RESULT9 *(volatile unsigned int *)0x70B1
#define RESULT10 *(volatile unsigned int *)0x70B2
#define RESULT11 *(volatile unsigned int *)0x70B3
#define RESULT12 *(volatile unsigned int *)0x70B4
#define RESULT13 *(volatile unsigned int *)0x70B5
#define RESULT14 *(volatile unsigned int *)0x70B6
#define RESULT15 *(volatile unsigned int *)0x70B7
#define CALIBRATION *(volatile unsigned int *)0x70B8
/* Registradores do CAN */
#define MDER *(volatile unsigned int *)0x7100
#define TCR *(volatile unsigned int *)0x7101
#define RCR *(volatile unsigned int *)0x7102
#define MCR *(volatile unsigned int *)0x7103
#define BCR2 *(volatile unsigned int *)0x7104
#define BCR1 *(volatile unsigned int *)0x7105
#define ESR *(volatile unsigned int *)0x7106
#define GSR *(volatile unsigned int *)0x7107
#define CEC *(volatile unsigned int *)0x7108
#define CAN_IFR *(volatile unsigned int *)0x7109
#define CAN_IMR *(volatile unsigned int *)0x710A
#define LAM0_H *(volatile unsigned int *)0x710B
#define LAM0_L *(volatile unsigned int *)0x710C
#define LAM1_H *(volatile unsigned int *)0x710D
```

```
#define LAM1_L *(volatile unsigned int *)0x710E
#define MSGID0L *(volatile unsigned int *)0x7200
#define MSGID0H *(volatile unsigned int *)0x7201
#define MSGCTRL0 *(volatile unsigned int *)0x7202
#define MBX0A *(volatile unsigned int *)0x7204
#define MBX0B *(volatile unsigned int *)0x7205
#define MBX0C *(volatile unsigned int *)0x7206
#define MBX0D *(volatile unsigned int *)0x7207
#define MSGID1L *(volatile unsigned int *)0x7208
#define MSGID1H *(volatile unsigned int *)0x7209
#define MSGCTRL1 *(volatile unsigned int *)0x720A
#define MBX1A *(volatile unsigned int *)0x720C
#define MBX1B *(volatile unsigned int *)0x720D
#define MBX1C *(volatile unsigned int *)0x720E
#define MBX1D *(volatile unsigned int *)0x720F
#define MSGID2L *(volatile unsigned int *)0x7210
#define MSGID2H *(volatile unsigned int *)0x7211
#define MSGCTRL2 *(volatile unsigned int *)0x7212
#define MBX2A *(volatile unsigned int *)0x7214
#define MBX2B *(volatile unsigned int *)0x7215
#define MBX2C *(volatile unsigned int *)0x7216
#define MBX2D *(volatile unsigned int *)0x7217
#define MSGID3L *(volatile unsigned int *)0x7218
#define MSGID3H *(volatile unsigned int *)0x7219
#define MSGCTRL3 *(volatile unsigned int *)0x721A
#define MBX3A *(volatile unsigned int *)0x721C
#define MBX3B *(volatile unsigned int *)0x721D
#define MBX3C *(volatile unsigned int *)0x721E
```

```
#define MBX3D *(volatile unsigned int *)0x721F
#define MSGID4L *(volatile unsigned int *)0x7220
#define MSGID4H *(volatile unsigned int *)0x7221
#define MSGCTRL4 *(volatile unsigned int *)0x7222
#define MBX4A *(volatile unsigned int *)0x7224
#define MBX4B *(volatile unsigned int *)0x7225
#define MBX4C *(volatile unsigned int *)0x7226
#define MBX4D *(volatile unsigned int *)0x7227
#define MSGID5L *(volatile unsigned int *)0x7228
#define MSGID5H *(volatile unsigned int *)0x7229
#define MSGCTRL5 *(volatile unsigned int *)0x722A
#define MBX5A *(volatile unsigned int *)0x722C
#define MBX5B *(volatile unsigned int *)0x722D
#define MBX5C *(volatile unsigned int *)0x722E
#define MBX5D *(volatile unsigned int *)0x722F
/*Registradores do EVA */
#define GPTCONA *(volatile unsigned int *)0x7400
#define T1CNT *(volatile unsigned int *)0x7401
#define T1CMPR *(volatile unsigned int *)0x7402
#define T1PR *(volatile unsigned int *)0x7403
#define T1CON *(volatile unsigned int *)0x7404
#define T2CNT *(volatile unsigned int *)0x7405
#define T2CMPR *(volatile unsigned int *)0x7406
#define T2PR *(volatile unsigned int *)0x7407
#define T2CON *(volatile unsigned int *)0x7408
#define COMCONA *(volatile unsigned int *)0x7411
#define ACTRA *(volatile unsigned int *)0x7413
#define DBTCONA *(volatile unsigned int *)0x7415
```

```
#define CMPR1 *(volatile unsigned int *)0x7417
#define CMPR2 *(volatile unsigned int *)0x7418
#define CMPR3 *(volatile unsigned int *)0x7419
#define CAPCONA *(volatile unsigned int *)0x7420
#define CAPFIFOA *(volatile unsigned int *)0x7422
#define CAP1FIFO *(volatile unsigned int *)0x7423
#define CAP2FIFO *(volatile unsigned int *)0x7424
#define CAP3FIFO *(volatile unsigned int *)0x7425
#define CAP1FBOT *(volatile unsigned int *)0x7427
#define CAP2FBOT *(volatile unsigned int *)0x7428
#define CAP3FBOT *(volatile unsigned int *)0x7429
#define EVAIMRA *(volatile unsigned int *)0x742C
#define EVAIMRB *(volatile unsigned int *)0x742D
#define EVAIMRC *(volatile unsigned int *)0x742E
#define EVAIFRA *(volatile unsigned int *)0x742F
#define EVAIFRB *(volatile unsigned int *)0x7430
#define EVAIFRC *(volatile unsigned int *)0x7431
/* Registradores do EVB */
#define GPTCONB *(volatile unsigned int *)0x7500
#define T3CNT *(volatile unsigned int *)0x7501
#define T3CMPR *(volatile unsigned int *)0x7502
#define T3PR *(volatile unsigned int *)0x7503
#define T3CON *(volatile unsigned int *)0x7504
#define T4CNT *(volatile unsigned int *)0x7505
#define T4CMPR *(volatile unsigned int *)0x7506
#define T4PR *(volatile unsigned int *)0x7507
#define T4CON *(volatile unsigned int *)0x7508
#define COMCONB *(volatile unsigned int *)0x7511
```

```
#define ACTRB *(volatile unsigned int *)0x7513
#define DBTCONB *(volatile unsigned int *)0x7515
#define CMPR4 *(volatile unsigned int *)0x7517
#define CMPR5 *(volatile unsigned int *)0x7518
#define CMPR6 *(volatile unsigned int *)0x7519
#define CAPCONB *(volatile unsigned int *)0x7520
#define CAPFIFOB *(volatile unsigned int *)0x7522
#define CAP4FIFO *(volatile unsigned int *)0x7523
#define CAP5FIFO *(volatile unsigned int *)0x7524
#define CAP6FIFO *(volatile unsigned int *)0x7525
#define CAP4FBOT *(volatile unsigned int *)0x7527
#define CAP5FBOT *(volatile unsigned int *)0x7528
#define CAP6FBOT *(volatile unsigned int *)0x7529
#define EVBIMRA *(volatile unsigned int *)0x752C
#define EVBIMRB *(volatile unsigned int *)0x752D
#define EVBIMRC *(volatile unsigned int *)0x752E
#define EVBIFRA *(volatile unsigned int *)0x752F
#define EVBIFRB *(volatile unsigned int *)0x7530
#define EVBIFRC *(volatile unsigned int *)0x7531
/* Registradores Mapeados no Espaço I/O */
#define FCMR portFF0F
ioport unsigned int portFF0F;
#define WSGR portFFFF
ioport unsigned int portFFFF;
```

O seguinte código, por exemplo, associa o registrador **IMR** com um apontador do apontador imediato para um inteiro sem sinal localizado no endereço 0x0004.

#define IMR *(volatile unsigned int *)0x0004

Isto permite acessar o registrador definido em programas feitos em linguagem C, da seguinte forma. Por exemplo, para escrever o valor 0003 no registrador **IMR**, basta fazer:

IMR = 0x0003;

Pode-se também definir o registrador da seguinte forma.

#define IMR (volatile unsigned int *)0x0004

Nessa definição, associa-se o registrador **IMR** com um do apontador imediato para um inteiro sem sinal localizado no endereço 0x0004. Nesse caso, para escrever nesse registrador basta fazer:

***IMR = 0x0003;**

A palavra chave **"volatile"** informa ao compilador que essa variável não está apenas sobre seu comando. O valor dessa variável (registrador) poderá ser alterado por *hardware*. Detalhes sobre a palavra chave **"volatile"** são encontrados na referência [16].

Observe no programa que os registradores FCMR e WSGR foram definidos de forma diferente. Eles foram definidos dessa forma, pois eles estão mapeados no espaço I/O.

O compilador fornece a palavra chave **"ioport"** para declarar um endereço no espaço I/O. A palavra **"ioport"** declara um endereço do espaço I/O como sendo de um tipo particular, no caso a seguir, como um unsigned int. O endereço dele mesmo é declarado usando o construtor "portxxxx", onde xxxx é um endereço entre os 64 K *words* do espaço I/O. No arquivo LF2407A.h, temos:

ioport unsigned int portFF0F;

Para se fazer um código em linguagem C mais compreensível, pode-se associar um símbolo à portFF0F, como foi feito no arquivo LF2407A.h:

#define FCMR portFF0F

Pode-se agora acessar o endereço do espaço I/O declarado usando o nome definido. Por exemplo para escrever 0x0001 no espaço de endereço 0xFF0F, basta fazer:

FCMR = 0x0001;

Detalhes sobre a palavra chave **"ioport"** e sobre o mapa de memória do espaço I/O, são encontradas na referência (SPRU024e).

15.1.5 Descrição do Programa Principal - Exemplo.c

No programa principal inclue-se o arquivo LF2407A.h, define-se as variáveis e constantes que serão utilizadas, realiza-se a inicialização e configuração do DSP e dos periféricos que serão utilizados e cria-se as rotinas de serviço de interrupção.

Arquivo Exemplo.c

```
#include "LF2407A.h"
/*Definição das constantes*/
unsigned int periodoT1, periodoT2, ciclo;
#define PWM_T1 750 /* Metade do período do PWM gerado */
#define ciclo 375 /* PWM1 com ciclo de trabalho de 25% */
#define peridoT2 62500 /* A cada 200 ms ocorrerá uma interrupção*/
void main(void)
/* Configura registradores SCSR1 e SCSR2 */
SCSR1 = 0x0005;
SCSR2 = (SCSR2 | 0x000B) & 0x000F;
/* Configura WD TIMER */
WDCR = 0x00E8;
/* Setup da interface de memória externa*/
WSGR = 0x0040;
/* PINOS I/O */
MCRA = 0x0040;
MCRB = 0xFE00;
MCRC = 0x0000;
PCDATDIR= PCDATDIR — 0x0100; /* Configura o pino IOPC0 como saída digital*/
T1CON = 0x0000; /* Desabilita timer 1 */
T2CON = 0x0000;  /* Desabilita timer 2 */
GPTCONA = 0x0000; /*Configura GPTCONA */
```

```
/* Configura timer 1 e PWM1 */
T1CNT = 0x0000;  /* Zera o contador do timer 1 */
T1PR = PWM_T1; /* Seta o período do timer 1 */
DBTCONA = 0x0000; /* Desativa as unidades de banda morta */
CMPR1 = ciclo; /* Seta o ciclo de trabalho do PWM1 */
ACTRA = 0x0002; /* Seta o pino PWM1 em ativo alto */
COMCONA = 0x8200; /* configure COMCON register */
T1CON = 0x0840; /* Configura timer 1 */
/* Configura o timer 2 */
T2CNT = 0x0000;  /* Zera o contador */
T2PR = periodoT2; /* Seta o periodo do timer 2 */
T2CON = 0xD740; /* Configura timer 2 */
/* Setup das interrupções do core */
IMR = 0x0000; /* Zera o registrador IMR */
IFR = 0x003F; /* Zera interrupções do core pendentes */
IMR = 0x0004; /* Habilita a interrupção INT3 */
/* Setup das interrupções do EV */
EVAIFRA = 0xFFFF; /* Zera interrupções do grupo A do EVA */
EVAIFRB = 0xFFFF; /* Zera interrupções do grupo B do EVA */
EVAIFRC = 0xFFFF; /* Zera interrupções do grupo C do EVA */
EVAIMRA = 0x0000; /* Desabilita interrupções desejadas do grupo A do EVA */
EVAIMRB = 0x0001; /* Habilita interrupção T2PINT (timer2) do grupo B do EVA */
EVAIMRC = 0x0000; /* Desabilita interrupções desejadas do grupo C do EVA */
EVBIFRA = 0xFFFF; /* Zera interrupções do grupo A do EVB */
EVBIFRB = 0xFFFF; /* Zera interrupções do grupo B do EVB */
EVBIFRC = 0xFFFF; /* Zera interrupções do grupo C do EVB */
EVBIMRA = 0x0000; /* Desabilita interrupções desejadas do grupo A do EVB */
EVBIMRB = 0x0000; /* Desabilita interrupções desejadas do grupo B do EVB */
```

```
EVBIMRC = 0x0000; /* Desabilita interrupções desejadas do grupo C do EVB */
asm("CLRC INTM ");
while(1); /* Loop sem fim, espera por interrupção */
} /* Fim do main() */
/************* Rotinas de serviço de interrupção *************/
interrupt void timer2_ISR(void)
{
EVAIFRB = EVAIFRB & 0x0001; /* Zera flag T2PINT */
if(PCDATDIR & 0x0101 == 0x0101)  /* LED acesso */
{
PCDATDIR = PCDATDIR & 0x0100; /* Apaga o LED */
}
else
{
PCDATDIR = 0x0101; /* Acende o LED */

}
void dummy(void)
{
while(1);
}
```

No trecho de código **asm("CLRC INTM ")** está escrito está escrito em *assembly*. Esse *bit* não é diretamente acessível de um código escrito em linguagem C. O *bit* de máscara de interrupção global (INTM) do registrador **ST0** está sendo zerado através da instrução *assembly* CLRC. Zerando este *bit*, habilita todas as interrupções mascaráveis que foram habilitadas via o registrador **IMR**. Outros *bits* de registradores podem ser acessados da mesma forma. A palavra chave **"asm"** é melhor definida na referência [16].

15.1.6 Programa de Ligação - Exemplo.cmd

Arquivo Exemplo.cmd

```
MEMORY {
        PAGE 0: /* Memória de Programa */
                VECS: org=00000h, len=00040h /* FLASH Interna */
                FLASH: org=00044h, len=07FBCh /* FLASH Interna */
                EXTPROG: org=08800h, len=07800h /* SRAM Externa */
        PAGE 1: /* Memória de Dados */
                B2: org=00060h, len=00020h /* DARAM Interna */
                B0: org=00200h, len=00100h /* DARAM Interna */
                B1: org=00300h, len=00100h /* DARAM Interna */
                SARAM: org=00800h, len=00800h /* SARAM Interna */
                EXTDATA: org=08000h, len=08000h /* SRAM Externa */
}
/* Seções Geradas pelo Compilador C */
SECTIONS {
        .text: > FLASH PAGE 0 /* Inicializado */
        .cinit: > FLASH PAGE 0 /* Inicializado */
        .const: > B1 PAGE 1 /* Inicializado */
        .switch: > FLASH PAGE 0 /* Inicializado */
        .bss: > B1 PAGE 1  /* Não Inicializado */
        .stack: > SARAM PAGE 1 /* Não Inicializado */
        .sysmem: > B1 PAGE 1 /* Não Inicializado */
                vetores: > VECS PAGE 0
}
```

No arquivo **exemplo.cmd**, a seção MEMORY define toda memória disponível no DSP. O mapa de memória básico pode ser encontrado na referência. As definições de memória são feitas separadamente no espaço de programa (page0) e no espaço de dados (page1).

No espaço de programa, a região de memória chamada de VECS foi especialmente definido de modo que os vetores de *reset* e de interrupção podessem ser ligados para as corretas locações. Veja que a memória VECS é fisicamente parte de ambas a FLASH interna ou da memória externa, dependendo do estado do pino MP/MC no reset. A região de memória chamada de FLASH é definida em uma faixa de endereço da memória de 32Kx16 que corresponde a duas possíveis memórias físicas. Se o pino MP/$\overline{MC}$ for '0' durante o *reset* do DSP, será a memória FLASH interna. Se o pino MP/$\overline{MC}$ for '1' durante o *reset* do DSP, será a memória externa. Finalmente a região de memória chamada de EXTPROG define a memória de programa externa disponível.

No espaço de dados, os blocos de memória DARAM B0, B1 e B2 são definidos, assim como os 2Kx16 do bloco de memória SARAM. A região de memória chamada de EXTDATA define a memória de dados externa.

A seção SECTIONS chama o ligador onde está localizado cada seção usada no código. O compilador C utiliza sete seções específicas (.text, .cinit, .const, .switch, .bss, .stack e .sysmem). A função de cada uma dessas seções é documentada na referência [16]. Em adição as essas seções, uma seção definida pelo usuário chamada de 'vetores' é também usada por este programa, e portanto deverá ser especificada no arquivo de ligação de comandos (exemplo.cmd). A seção 'vetores'está sendo ligada para a região de memória VECS, no qual coloca os vetores de interrupção nos corretos endereços específicos do DSP. A seção SECTIONS é documentada na referência [17].

15.2 Aplicação Utilizando o ADC

Nesta aplicação, utiliza-se o ADC para converter um sinal de tensão que varia de 0 à 3,3 V controlado por um potenciômetro. O sinal será convertido pelo canal 0 do ADC (ADCIN0 - Pino 2 do conector P1) a cada 100 ms. O *timer 1* será utilizado para gerar uma interrupção a cada 100 ms. O conversor A/D está configurado no modo *Start/Stop*.

15.2.1 Registradores Utilizados

Nesse programa, os seguintes registradores são utilizados:

SCSR1 - Registrador de *status* e controle do sistema 1
SCSR2 - Registrador de *status* e controle do sistema 2
WDCR - Registrador de controle do *watchdog timer*
WSGR - Registrador de controle do gerador de estados de espera
MCRA - Registrador A de controle do MUX I/O
MCRB - Registrador B de controle do MUX I/O
MCRC - Registrador C de controle do MUX I/O
GPTCONA - Registrador A de controle do *Timer*
T1CON - Registrador de controle do *Timer 1*
T1CNT - Registrador de contagem do **Timer 1**
T1PR - Registrador de período do *timer 1*
ADCTRL1 - Registrador de controle do ADC 1
ADCTRL2 - Registrador de controle do ADC 2
CHSELSEQ1 - Registrador de controle da seleção do canal de entrada do ADC 1
MAX_CONV - Registrador de seleção do número de conversões por sequenciamento
IMR - Registrador de máscara de interrupção
IFR - Registrador de sinalização de interrupção

15.2.2 Programa Principal - ADC.c

```
#include "LF2407A.h"
    unsigned int ADC; /* Define variável */
    #define PeridoT1 31250 /* A cada 100 ms ocorrerá uma interrupção*/

    void main(void) {

    /* Configuração dos registradores SCSR1, SCSR2, WDCR e WSGR */
    SCSR1 = 0x0085;
    SCSR2 = (SCSR2 | 0x000B) & 0x000F;
    WDCR = 0x00E8;
    WSGR = 0x0000;

    /*** PINOS I/O ***/
    MCRA = 0x0000;
    MCRB = 0xFE00;
    MCRC = 0x0000;

    /****Configuração do Timer 1 ****/
    GPTCONA = 0x0100; /*Configura GPTCONA */
    T1CON = 0x0000; /* Zera timer 1 */
    T1CNT = 0x0000; /* Zera o contador do timer 1 */
    T1PR = PeridoT1; /* Seta o período do timer 1 */

    /* Configuração do ADC */
    ADCTRL1 = 0x4000; /* Reseta o ADC */
    ADCTRL1 = 0x2780;
    CHSELSEQ1 = 0x0000; /* Converte apenas o canal 0 */
    ADCTRL2 = 0x4000; /* Reseta o SEQ1 */
```

```
ADCTRL2 = 0x0500;
MAX_CONV = 0x0000; /* Desempenha apenas uma conversão por vez */

/* Setup das interrupções do core */
IMR = 0x0000; /* Zera o registrador IMR */
IFR = 0x003F; /* Zera interrupções do core pendentes */
IMR = 0x0001; /* Habilita a interrupção INT1 */
asm("CLRC INTM "); /* Habilita interrupção global */
T1CON = 0x1744; /* Configura Timer1 e inicia contagem */
while(1); /* Loop sem fim, espera por interrupção */
} /* Fim do main() */

/******* Rotinas de serviço de interrupção *********/
interrupt void ADC_ISR(void)
{
if((PIVR - 0x0004) == 0) /* PIVR da interrupção do ADC */
{
ADC = RESULT0>>6; /* Armazena o resultado na variável ADC */
ADCTRL2 |= 0x4200; /* Reseta o ADC e Zera flag de interrupção*/
}
}

void dummy(void)
{
while(1);
}
```

15.2.3 Programa com os Vetores de Interrupção ADC - Vetores.asm

Arquivo - vetores.asm

```
.ref _c_int0, _dummy, _ADC_ISR
.sect "vetores"
rset: B _c_int0 ;00h reset
int1: B _ADC_ISR ;02h INT1
int2: B _dummy ;04h INT2
int3: B _dummy ;06h INT3
int4: B _dummy ;08h INT4
int5: B _dummy ;0Ah INT5
int6: B _dummy ;0Ch INT6
int7: B _dummy ;0Eh Reservado
int8: B _dummy ;10h INT8 (software)
int9: B _dummy ;12h INT9 (software)
int10: B _dummy ;14h INT10 (software)
int11: B _dummy ;16h INT11 (software)
int12: B _dummy ;18h INT12 (software)
int13: B _dummy ;1Ah INT13 (software)
int14: B _dummy ;1Ch INT14 (software)
int15: B _dummy ;1Eh INT15 (software)
int16: B _dummy ;20h INT16 (software)
int17: B _dummy ;22h TRAP
int18: B _dummy ;24h NMI
int19: B _dummy ;26h Reservado
int20: B _dummy ;28h INT20 (software)
int21: B _dummy ;2Ah INT21 (software)
int22: B _dummy ;2Ch INT22 (software)
```

```
int23: B _dummy ;2Eh INT23 (software)
int24: B _dummy ;30h INT24 (software)
int25: B _dummy ;34h INT26 (software)
int27: B _dummy ;36h INT27 (software)
int28: B _dummy ;38h INT28 (software)
int29: B _dummy ;3Ah INT29 (software)
int30: B _dummy ;3Ch INT30 (software)
int31: B _dummy ;3Eh INT31 (software)
```

15.3 Aplicação Utilizando o ADC e uma Saída PWM

Nesta aplicação, utiliza-se o ADC para converter um sinal de tensão que varia de 0 à 3,3 V controlado por um potenciômetro para ajustar a largura do ciclo de trabalho de um sinal PWM. O sinal será convertido pelo canal 0 do ADC (ADCIN0 - Pino 2 do conector P1) continuamente. O *timer 1* será utilizado para gerar o sinal PWM com período de 50,8 us.

15.3.1 Registradores Utilizados

Nesse programa, os seguintes registradores são utilizados:

SCSR1 - Registrador de *status* e controle do sistema 1
SCSR2 - Registrador de *status* e controle do sistema 2
WDCR - Registrador de controle do *watchdog timer*
WSGR - Registrador de controle do gerador de estados de espera
MCRA - Registrador A de controle do MUX I/O
MCRB - Registrador B de controle do MUX I/O
MCRC - Registrador C de controle do MUX I/O
GPTCONA - Registrador A de controle do *Timer*
T1CON - Registrador de controle do *Timer 1*
T1CNT - Registrador de contagem do **Timer 1**
T1PR - Registrador de período do *timer 1*
DBTCONA - Registrador de controle do tempo de banda morta
CMPR1 - Registrador da unidade de comparação 1

ACTRA - Registrador de controle de ação da comparação
COMCONA - Registrador A de controle da comparação
ADCTRL1 - Registrador de controle do ADC 1
ADCTRL2 - Registrador de controle do ADC 2
CHSELSEQ1 - Registrador de controle da seleção do canal de entrada do ADC 1
MAX_CONV - Registrador de seleção do número de conversões por sequenciamento
IMR - Registrador de máscara de interrupção
IFR - Registrador de sinalização de interrupção
EVAIMRA - Registrador de máscara de interrupção do grupo A do EVA
EVAIFRA - Registrador de sinalização de interrupção do grupo A do EVA

15.3.2 Programa Principal - ADC_PWM.c

```
#include "LF2407A.h"
    unsigned int POTENCIOMETRO;
    #define PeridoT1 1016
    #define VALOR_INICIAL 500
    /*************** Programa Principal ***************/
    void main(void) {
    /* Configura registradores SCSR1, SCSR2, WDCR e WSGR */
    SCSR1 = 0x0085;
    SCSR2 = (SCSR2 | 0x000B) & 0x000F;
    WDCR = 0x00E8;
    WSGR = 0x0000;

    /*** PINOS I/O ***/
    MCRA = 0x0040;
    MCRB = 0xFE00;
    MCRC = 0x0000;
```

```
/* Configura ADC */
ADCTRL1 = 0x4000; /* Reseta ADC */
ADCTRL1 = 0x27C0;
CHSELSEQ1 = 0x0000; /* Converte apenas o canal 0 */
ADCTRL2 = 0x4000; /* Reseta o SEQ1 */
ADCTRL2 = 0x2000;
MAX_CONV = 0x0000; /* Desempenha apenas uma conversão */

/* Configura timer 1 */
T1CON = 0x0000; /* Zera timer 1 */
GPTCONA = 0x0000; /*Configura o registrador GPTCONA */
T1CNT = 0x0000; /* Zera o contador do timer 1 */
T1PR = PeridoT1;  /* Seta o período do timer 1 */

 /* Configura Unidade de comparação 1*/
DBTCONA = 0x0000; /* Configura o registrador DBTCONA*/
CMPR1 = VALOR_INICIAL; /* Valor de comparação - PWM */
ACTRA = 0x0002; /* Configura o registrador ACTRA*/
COMCONA = 0x8200; /* Configura o registrador COMCONA*/

/* Setup das interrupções do core */
IMR = 0x0000; /* Zera o registrador IMR */
IFR = 0x003F; /* Zera interrupções do core pendentes */
IMR = 0x0002; /* Habilita a interrupção INT2 do timer 1 */
EVAIFRA = 0xFFFF; /* Zera flags de interrupção */
EVAIMRA = 0x0000; /* Zera máscaras de interrupção */
EVAIMRA = 0x0200; /* Habilita interrupção de underflow do timer 1*/
asm("CLRC INTM "); /* Habilita interrupção global */
```

```
T1CON = 0x0840; /* Configura timer 1 */
while(1); /* Loop sem fim, espera por interrupção */
} /* Fim do main() */
/************* Rotinas de Serviço de Interrupção *************/
/* Faz a leitura do potenciômetro e armazena-o no registrador CMPR1 */

interrupt void TIMER1_ISR(void)
{
if((PIVR - 0x0029) == 0) /* PIVR da interrupção T1UFINT */
{
POTENCIOMETRO = RESULT0>>6; /* Ler valor do poteciômetro */
CMPR1 = POTENCIOMETRO; /* Carrega CMPR1 com o valor do potenciômetro */
EVAIFRA = 0x0200; /* Zera flag T1UFINT */
}
}
void dummy(void)
{
while(1);
}
```

15.3.3 Programa com os Vetores de Interrupção ADC_PWM - Vetores.asm

Arquivo - vetores.asm

```
.ref _c_int0, _dummy, _TIMER1_ISR
.sect "vetores"
rset: B _c_int0 ;00h reset
int1: B _dummy ;02h INT1
int2: B _TIMER1_ISR ;04h INT2
int3: B _dummy ;06h INT3
```

```
int4: B _dummy ;08h INT4
int5: B _dummy ;0Ah INT5
int6: B _dummy ;0Ch INT6
int7: B _dummy ;0Eh Reservado
int8: B _dummy ;10h INT8 (software)
int9: B _dummy ;12h INT9 (software)
int10: B _dummy ;14h INT10 (software)
int11: B _dummy ;16h INT11 (software)
int12: B _dummy ;18h INT12 (software)
int13: B _dummy ;1Ah INT13 (software)
int14: B _dummy ;1Ch INT14 (software)
int15: B _dummy ;1Eh INT15 (software)
int16: B _dummy ;20h INT16 (software)
int17: B _dummy ;22h TRAP
int18: B _dummy ;24h NMI
int19: B _dummy ;26h Reservado
int20: B _dummy ;28h INT20 (software)
int21: B _dummy ;2Ah INT21 (software)
int22: B _dummy ;2Ch INT22 (software)
int23: B _dummy ;2Eh INT23 (software)
int24: B _dummy ;30h INT24 (software)
int25: B _dummy ;34h INT26 (software)
int27: B _dummy ;36h INT27 (software)
int28: B _dummy ;38h INT28 (software)
int29: B _dummy ;3Ah INT29 (software)
int30: B _dummy ;3Ch INT30 (software)
int31: B _dummy ;3Eh INT31 (software)
```

Capítulo 16

Conclusões

Os processadores digitais de sinais (DSPs) estão sendo utilizados em diversas áreas da engenharia, devido sua grande capacidade de processamento. Atualmente, existem no mercado diversos DSPs com características específicas para determinadas aplicações.

O DSP TMS320LF2407A da *Texas Instruments*, estudado nesse trabalho, possui periféricos que possibilitam o controle e o acionamento de máquinas, além de ter a capacidade de ser utilizado em outras aplicações. O TMS320LF2407A possui interno ao seu *chip* um controlador CAN, o qual permite que aplicações de controle distribuído com requisitos temporais sejam satisfeitos.

Com este trabalho foi possível estudar a arquitetura e os periféricos do DSP TMS320LF2407A, as caracterísitcas mais comuns entre os DSPs e realizar aplicações práticas utilizando a placa de desenvolvimento eZdsp LF2407A. Além disso, disponibilizar um material com informações básicas para o desenvolvimento de aplicações com o TMS320LF2407A.

Referências Bibliográficas

[1] Lapsley, P.; Bier, J.; Shoham, A. e Lee, Edward A. DSP Processor Fundamentals: Architectures and Features, IEEE Press, 1997.

[2] Wave Report. DSP Tutorial, www.wave-report.com, 2005.

[3] Spectrum Digital. eZdsp LF2407A, Technical Reference, USA, Inc, 2003.

[4] Texas Instruments. TMS320LF2407A DSP Controllers - SPRS145I, USA, Inc, 2003.

[5] Texas Instruments. TMS320F/C24x DSP Controllers CPU and Instruction Set Reference Guide - SPRU160C, USA, Inc, 1999.

[6] Texas Instruments. TMS320LF/LC240x DSP Controllers System and Peripherals Reference Guide - SPRU357B, USA, Inc, 2001.

[7] Texas Instruments. Getting Started in C and Assembly Code With the TMS320LF240x DSP - SPRA755A, USA, Inc, 2002.

[8] Texas Instruments. Code Composer User's Guide - SPRU296, USA, Inc, 1999.

[9] Texas Instruments. Quick Start Guide and Real-Timer Tutorial - SPRU510, USA, Inc, 2001.

[10] Robert Bosch GmbH, CAN Especification, Version 2.0, 1991.

[11] Lawrenz, W. CAN System Engineering: From Theory to Practical Applications, 1997.

[12] Texas Instruments. Understanding the CAN Controller on the TMS320C24x DSP Controller - SPRA500, USA, Inc, 1998.

[13] Texas Instruments. Introduction to the Controller Area Network (CAN) - SLOA101, USA, Inc, 2002.

[14] Texas Instruments. A System Evaluation of CAN Transceivers - SLLA109, USA, Inc, 2002.

[15] Texas Instruments. Industrial Automation using the CAN Bus Platform White Paper, USA, Inc, 2003.

[16] Texas Instruments. TMS320C2x/C2xx/C5x Optimizing C Compiler Users Guide - SPRU024E, USA, Inc, 1999.

[17] Texas Instruments. Fixed-Point DSP Assembly Language Tools Users Guide - SPRU018D, USA, Inc, 1995.