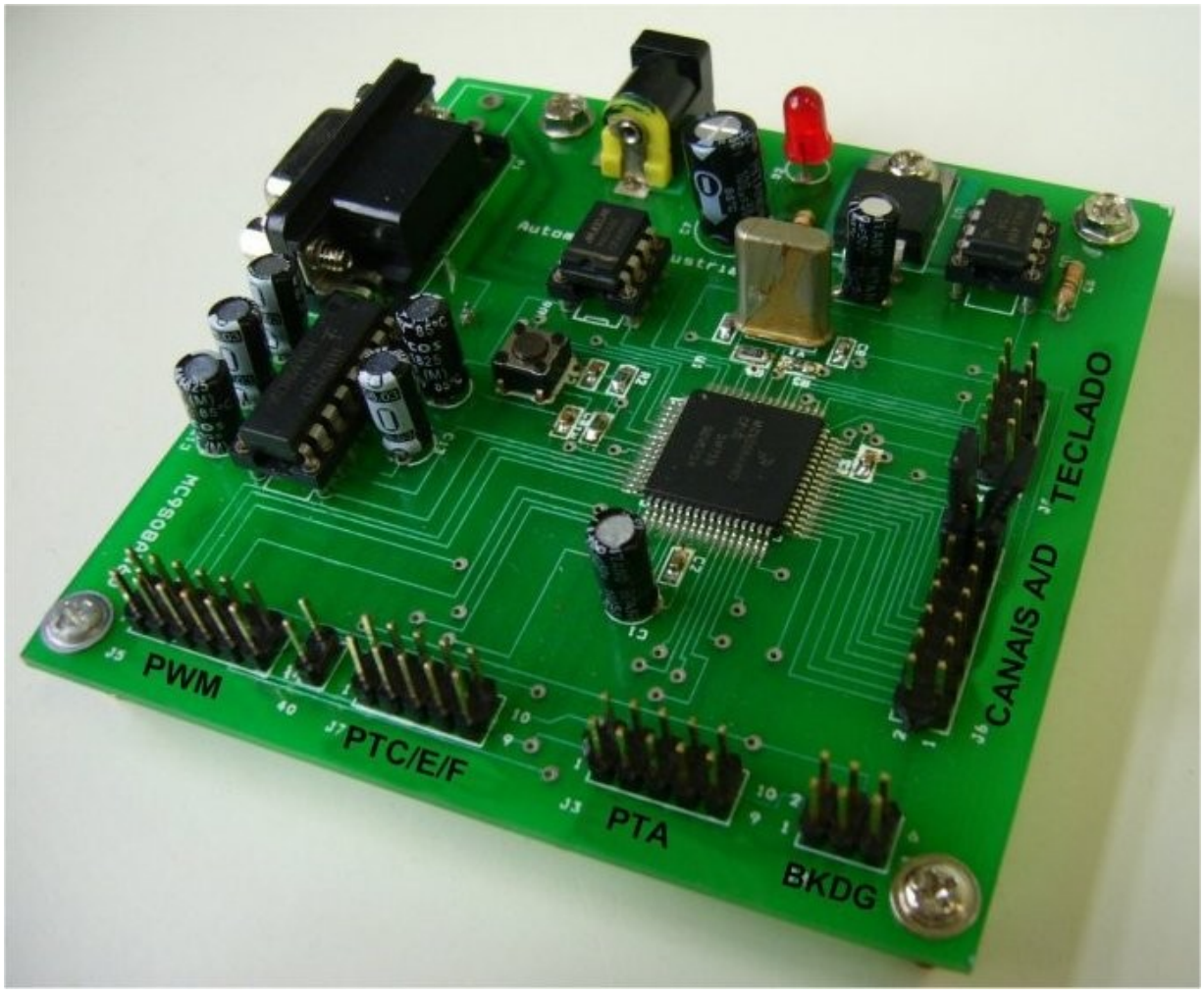
Tutorial para Programação de Microcontroladores

HCS08 Baseado no MC9S08AW60



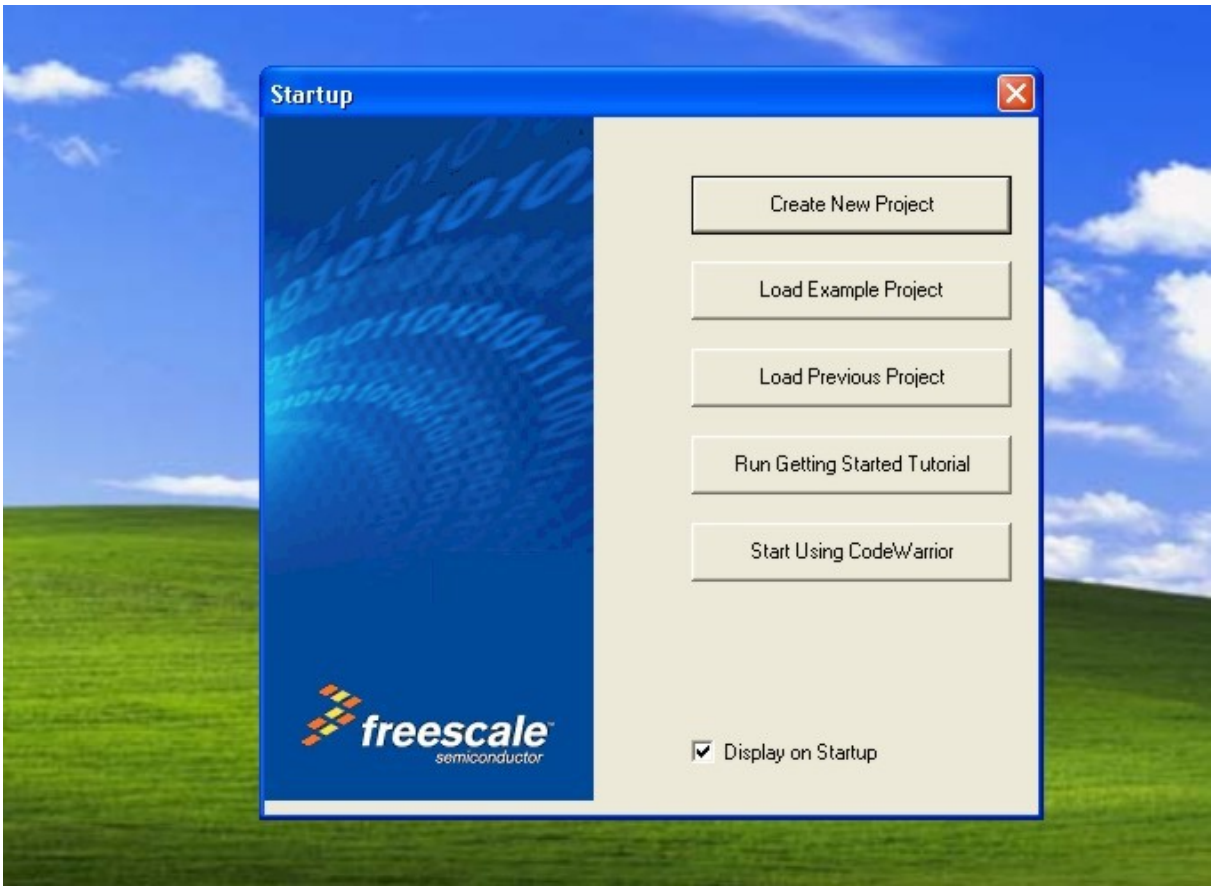
SUMÁRIO

Tutorial para Programação de Microcontroladores HCS08 Baseado no MC9S08AW60

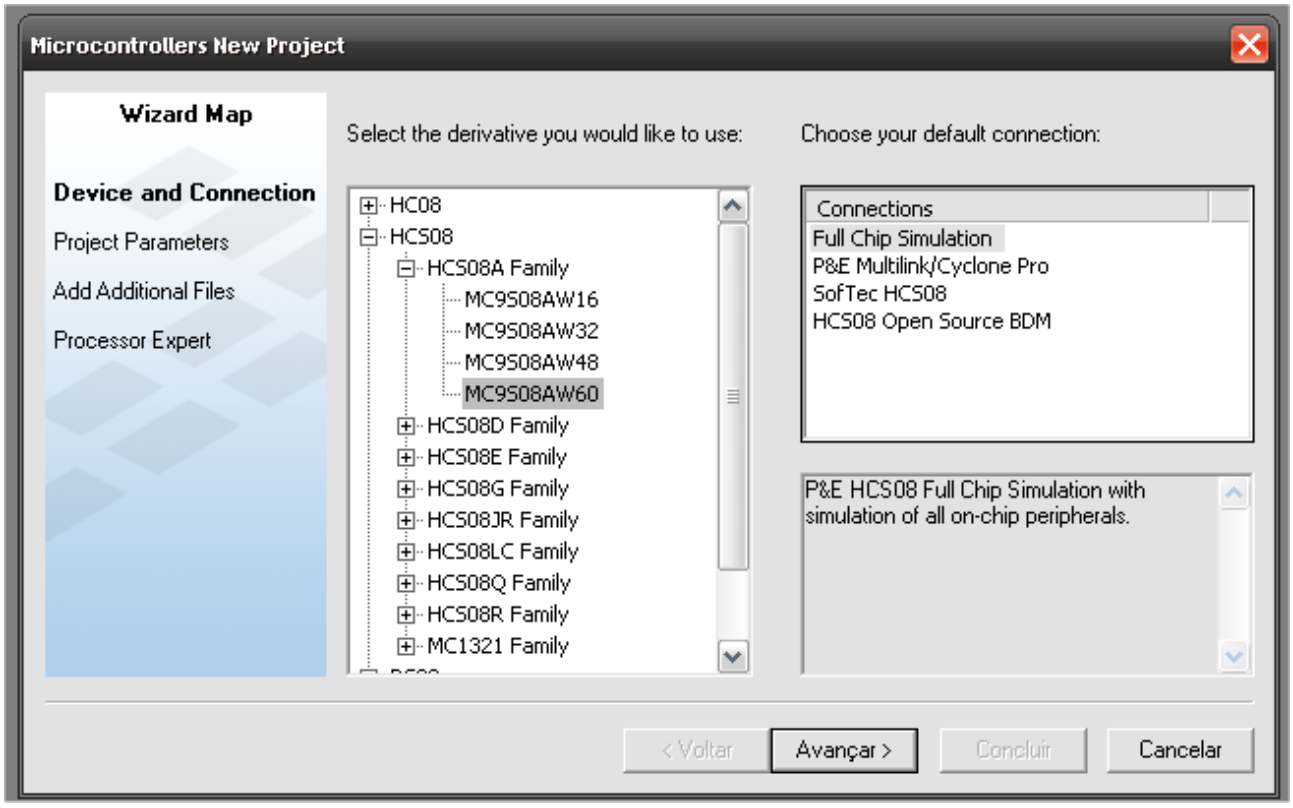
Este documento tem como objetivo explicar de forma detalhada como iniciar um projeto em linguagem “C” e configurar os periféricos de um microcontrolador da família HCS08 da *Freescale*. O microcontrolador utilizado neste tutorial será o MC9S08AW60.

1. Iniciando um Projeto na IDE CodeWarrior

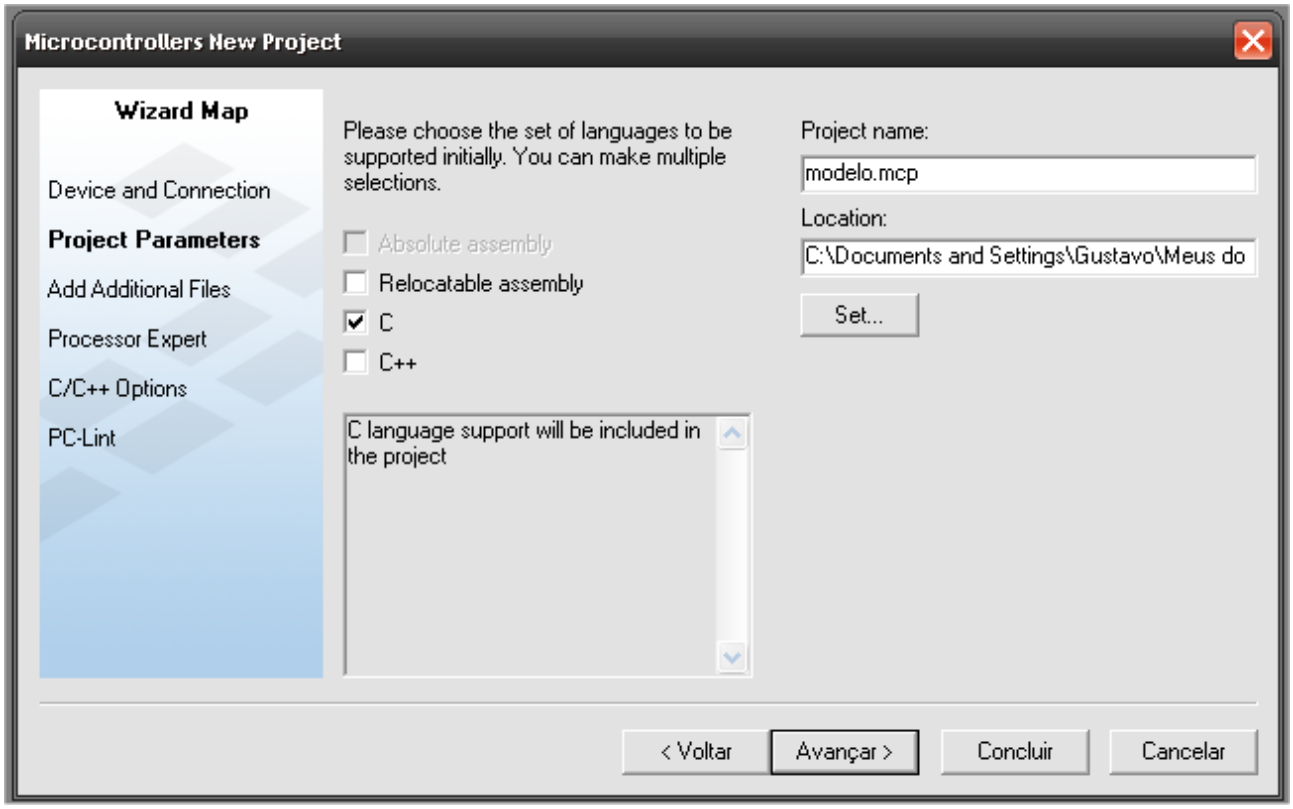
Iremos utilizar o aplicativo *CodeWarrior for Microcontrollers* da *Freescale* para iniciarmos o projeto. Ao executar este aplicativo será apresentada a seguinte tela.



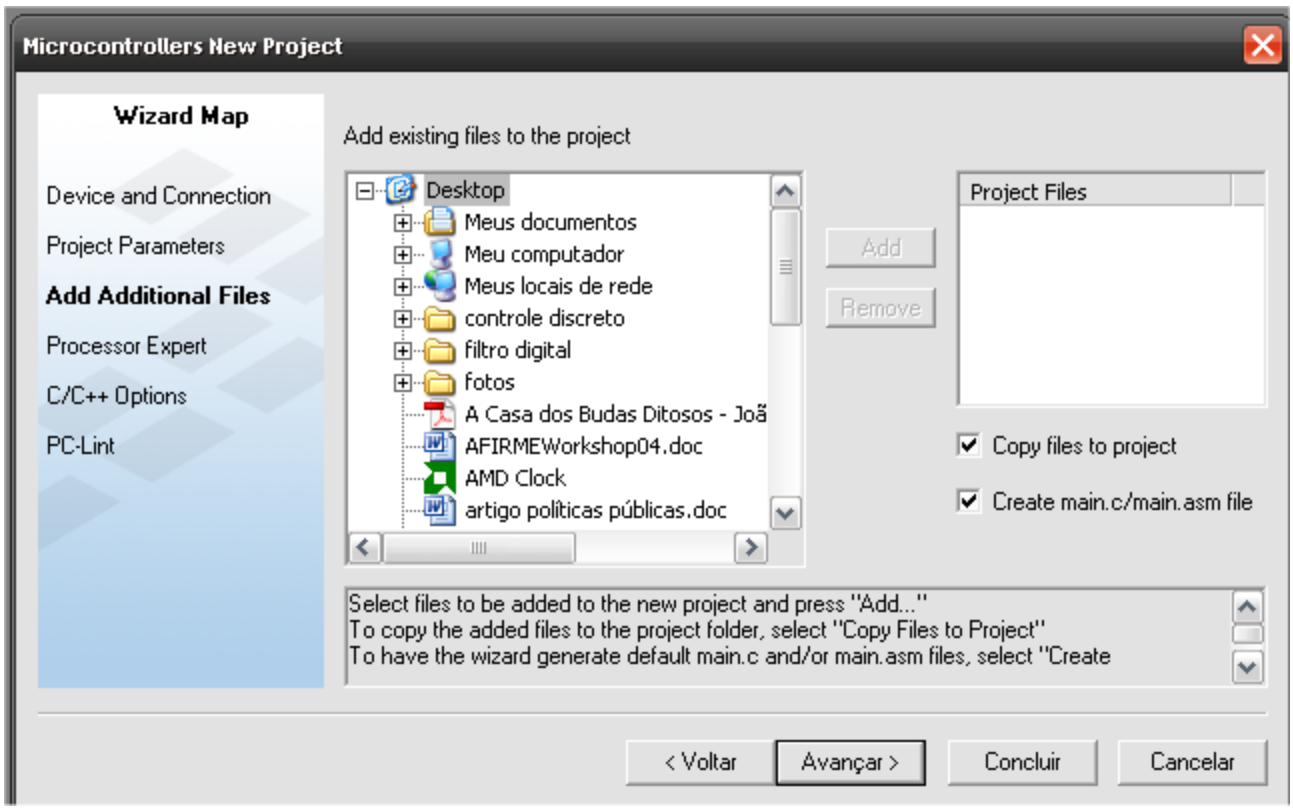
Para iniciar um projeto devemos clicar no botão *Create New Project*. A seguir a seguinte tela será apresentada:



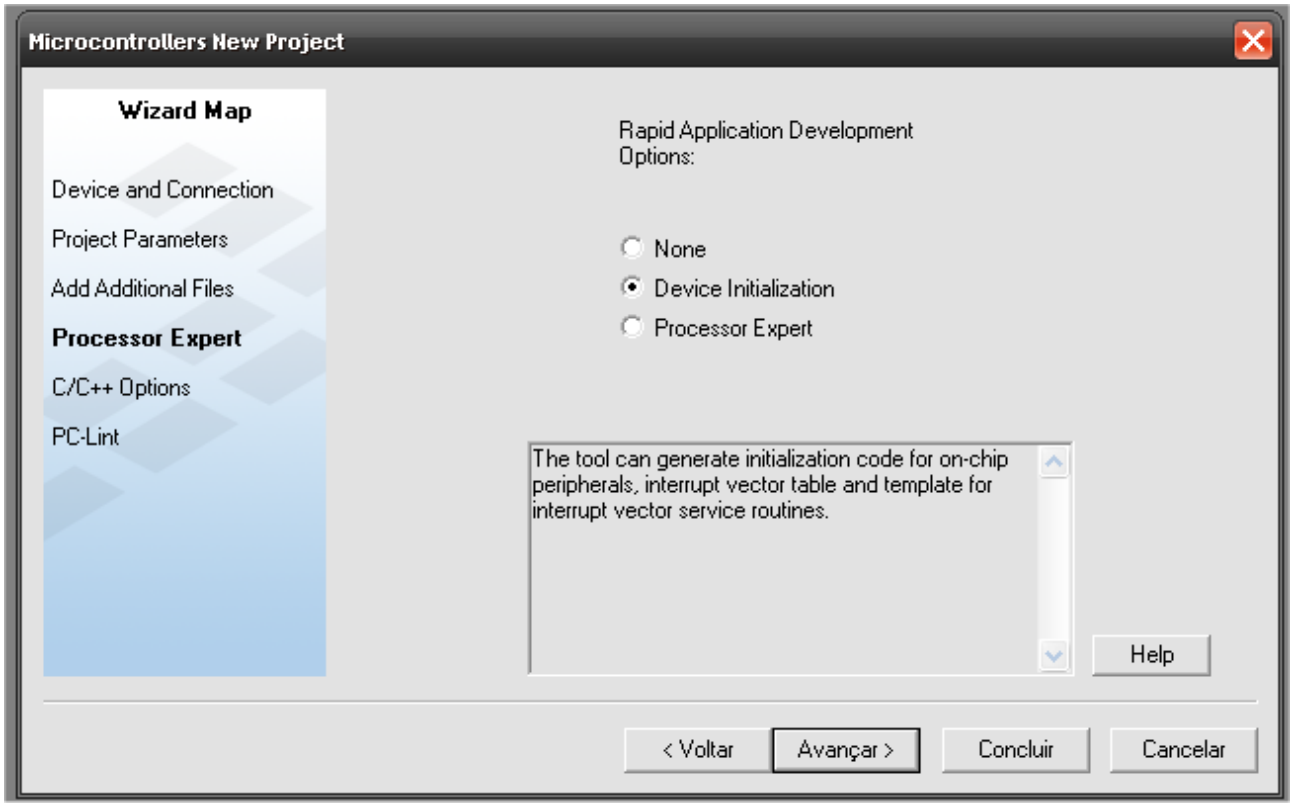
Iremos selecionar o microcontrolador MC9S08AW60 e clicar em avançar. Em seguida será apresentado a seguinte tela:



Nesta tela iremos selecionar a linguagem “C” e escolher o nome do projeto, neste caso, modelo.mcp. Em seguir a seguinte tela será apresentada:

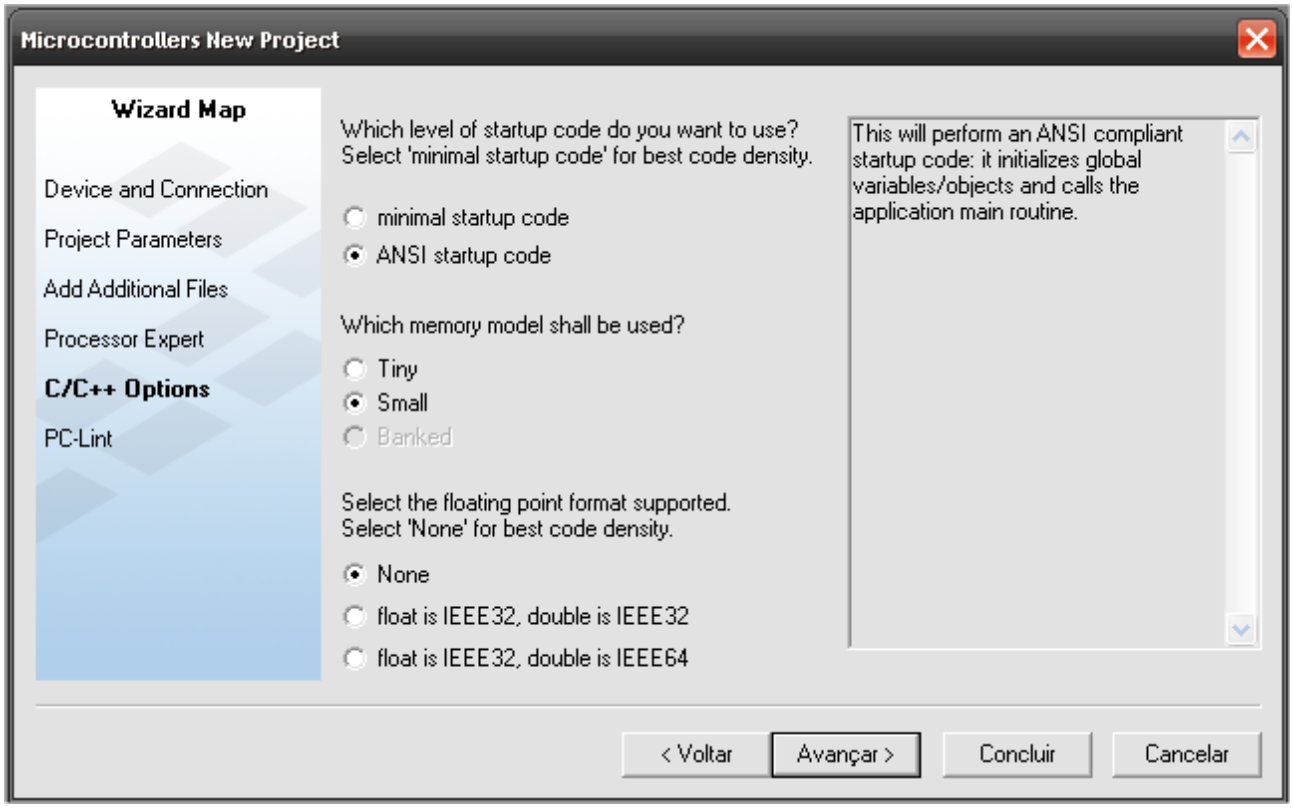


Nesta tela podemos selecionar arquivos com bibliotecas pré-programadas para o projeto. Neste exemplo não iremos incluir nenhuma biblioteca, simplesmente clicando em avançar. Desta forma será apresentada a tela abaixo:



Na tela acima existem três opções. A opção *None* não irá gerar nenhum código automaticamente. A opção *Device Initialization* irá gerar automaticamente um código mínimo de inicialização para o microcontrolador, além dos vetores de interrupção. A opção *Processor Expert* irá ajudar na configuração de periféricos, no entanto reduz o controle do programador sobre o código gerado. Desta forma iremos selecionar a opção *Device Initialization*.

Após selecionar esta opção, o aplicativo irá apresentar a seguinte tela:



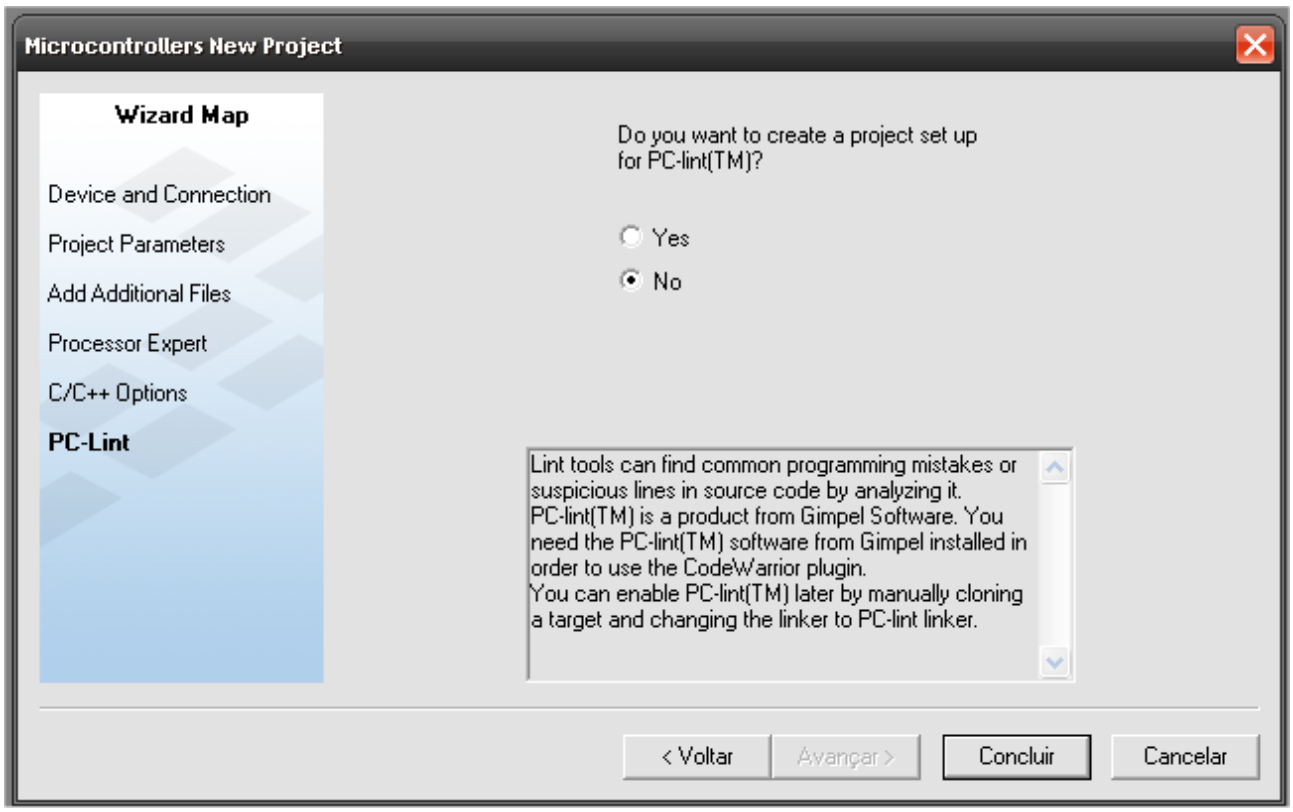
Nesta tela recomenda-se as seguintes configurações:

ANSI startup Code – Inicialização básica da memória RAM

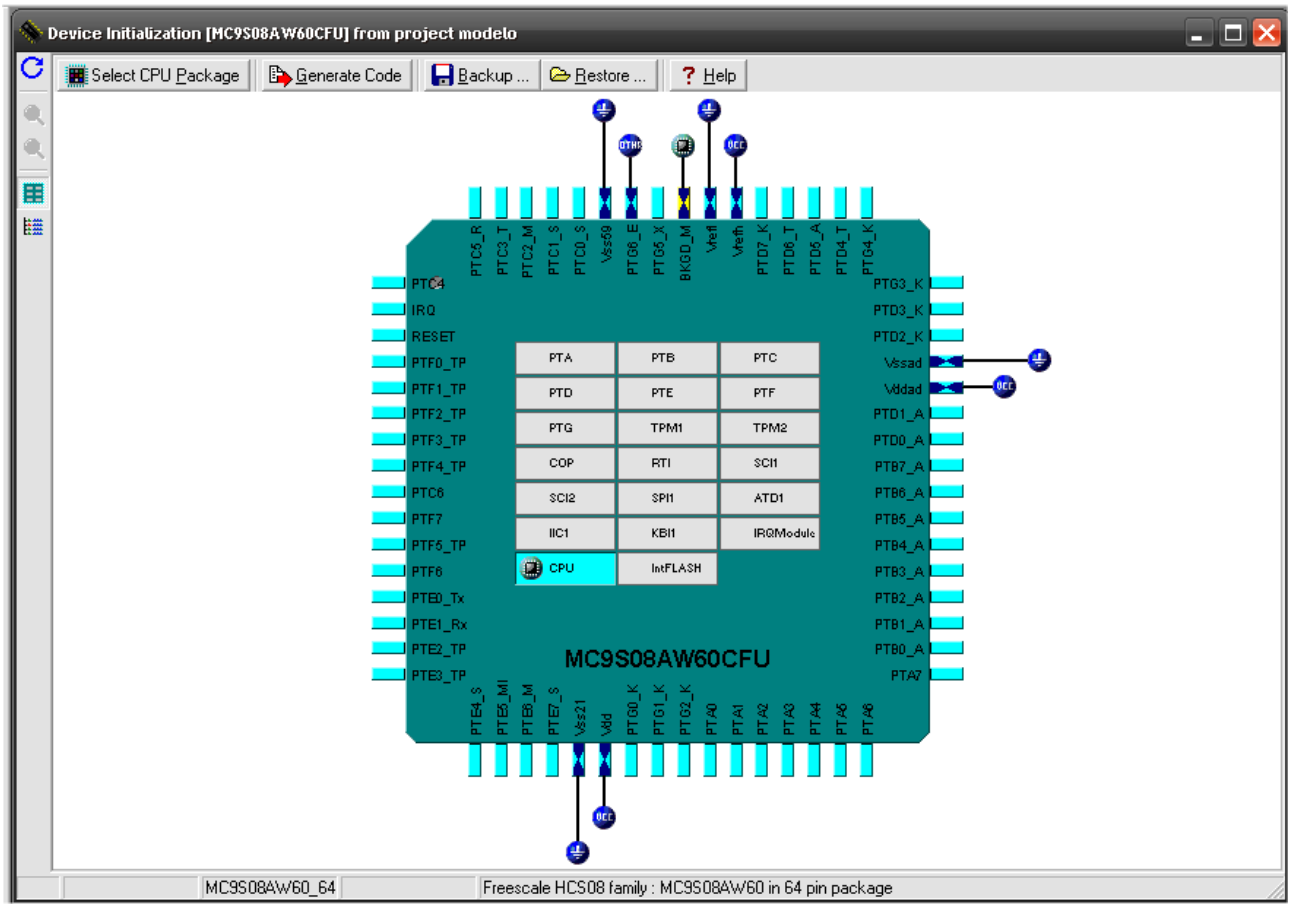
Small – Modo padrão de acesso a memória

None – Sem utilizar bibliotecas de ponto flutuante.

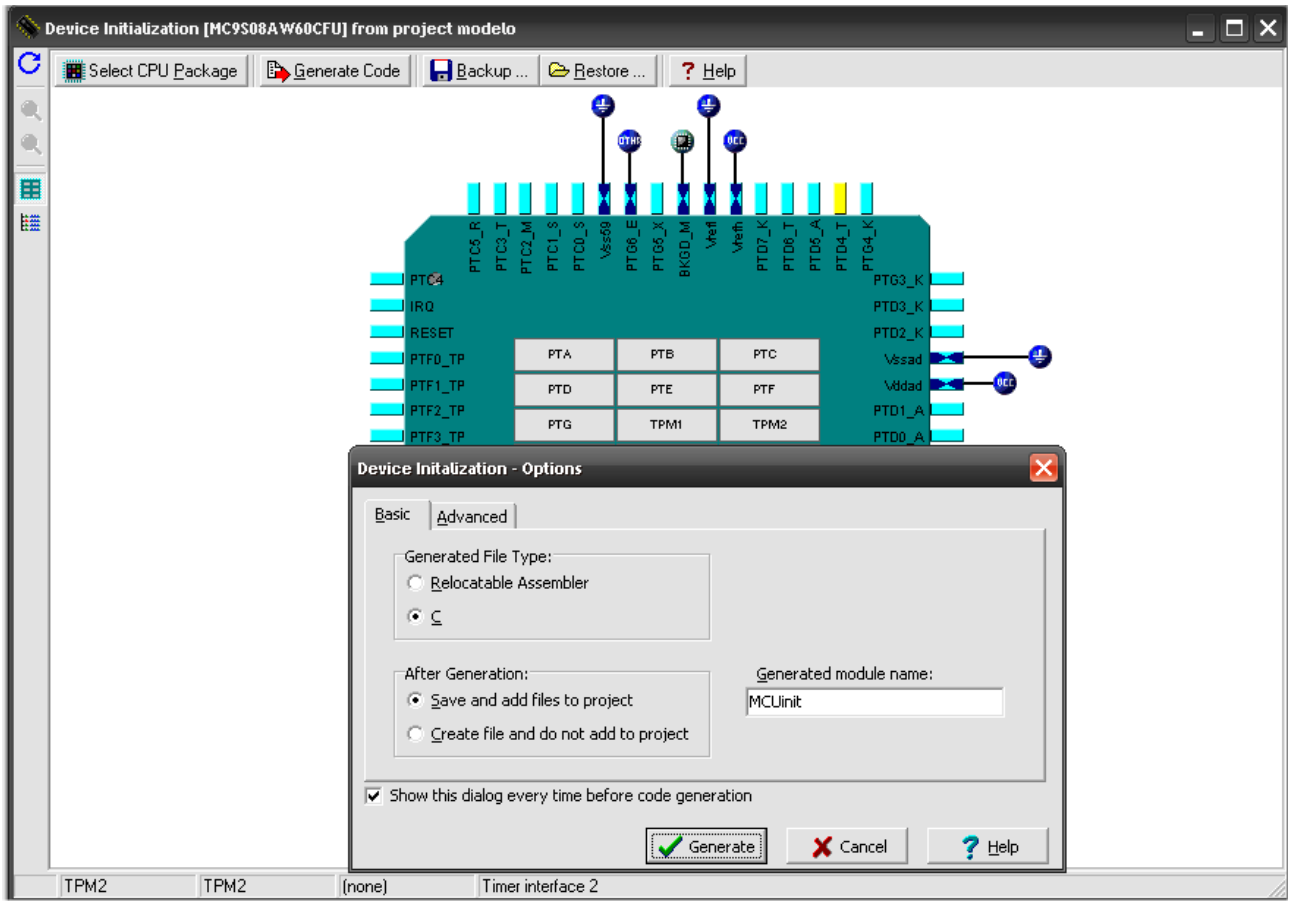
A seguir a seguinte tela será apresentada:



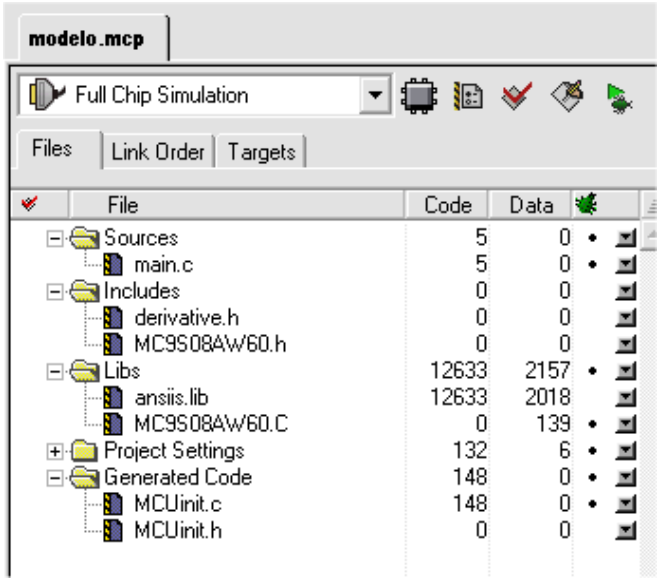
Nesta tela podemos ajustar o projeto para um *hardware* proprietário, conhecido por PC-Lint. Como não iremos utilizar tal *hardware*, iremos selecionar a opção *No* e clicar em Concluir. A tela a seguir será apresentada:



Nesta tela devemos selecionar a CPU MC9S08AW60CFU em *Select CPU Package*, que é o encapsulamento do microcontrolador utilizado na placa de demonstração. Após devemos clicar na opção *Generate Code*, para gerar o código de inicialização do microcontrolador. Em seguida devemos clicar na opção *Generate* da tela abaixo:



Após clicarmos neste botão o projeto será criado. A estrutura do projeto é apresentada na figura abaixo.



A pasta ***Sources*** irá conter o código fonte do projeto criado, já possuindo o arquivo principal do projeto. Na pasta ***Libs*** podemos adicionar bibliotecas de código já implementado, como por exemplo a biblioteca para acessar um display LCD. A Pasta ***Generated Code*** irá possuir a função de inicialização do microcontrolador, chamada *MCU_Init*, além dos vetores de interrupção.

A configuração dos periféricos será realizada dentro da função *MCU_Init,* pois esta função é chamada na inicialização do programa do microcontrolador.

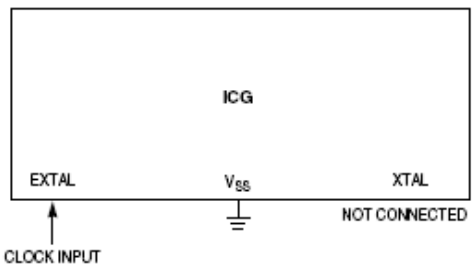
2. Configurações Básicas do Microcontrolador

O módulo interno de geração de *clock* (ICG) é utilizado para gerar o *clock* do sistema para a série de microcontroladores MC9S08AW. O módulo ICG provê multiplas opções para a fonte do sinal de *clock*. Estas opções oferecem ao usuário grande flexibilidade durante o projeto do sistema microcontrolado, permitindo escolhas entre custo, precisão, consumo de corrente e performance.

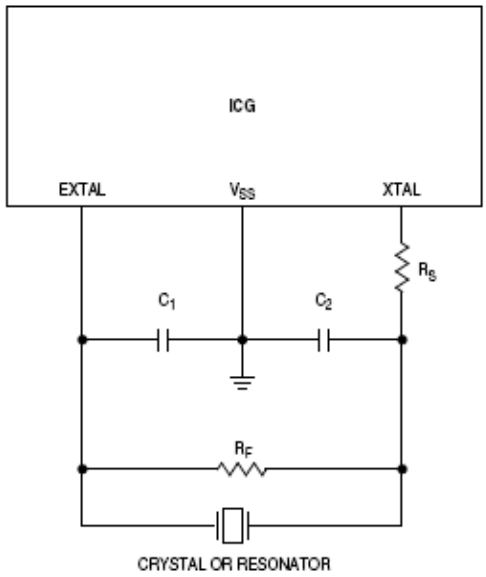
Algumas características do módulo ICG e do sistema de distribuição de *clock*:

- Várias possibilidades de fonte primária para o *clock*:
  - Cristal ou ressonador entre 32 Khz e 100 Khz
  - Cristal ou ressonador entre 1 MHz e 16 MHz
  - *Clock* Externo
  - Gerador de referência interna
- A configuração padrão do módulo ICG utiliza a referência interna de *clock*.
- Possui FLL (Frequeny-locked loop) para gerar *clocks* de barramento entre 4 MHz e 20 MHz
- Interrupção de tempo real com fonte de *clock* separada e auto-alimentada.
- Configuração do oscilador externo para baixo consumo ou alto ganho (para maior imunidade a ruídos).

Se um *clock* externo for utilizado os pinos devem ser utilizados como na figura abaixo:



Se uma referência de freqüência externa a cristal ou ressonador for utilizada, os pinos devem ser conectados como apresentado na figura abaixo:



A tabela de configuração dos resistores R_S e R_F e dos capacitores C_1 e C_2 é apresentada abaixo:

Characteristic	Symbol	Min	Typ[1]	Max	Unit
Load capacitors	C_1 C_2	See Note [2]			
Feedback resistor Low range (32k to 100 kHz) High range (1M – 16 MHz)	R_F		 10 1		 MΩ MΩ
Series resistor Low range Low Gain (HGO = 0) High Gain (HGO = 1) High range Low Gain (HGO = 0) High Gain (HGO = 1) ≥ 8 MHz 4 MHz 1 MHz	R_S	 — — — — — —	 0 100 0 0 10 20	 — — — — — —	 kΩ

[1] Typical values are based on characterization data at V_{DD} = 5.0V, 25°C or is typical recommended value.
[2] See crystal or resonator manufacturer's recommendation.

A placa de desenvolvimento **MC9S08AW60** utiliza um cristal externo de **4 Mhz**. Iremos utilizar esta referência de freqüência junto com o FLL para configurar uma **freqüência de barramento de 20 Mhz**, obtendo desta forma a máxima performance do microcontrolador.

A seguir serão apresentadas as configurações necessárias para a inicialização do *clock* neste modo de operação e a inicialização básica do sistema.

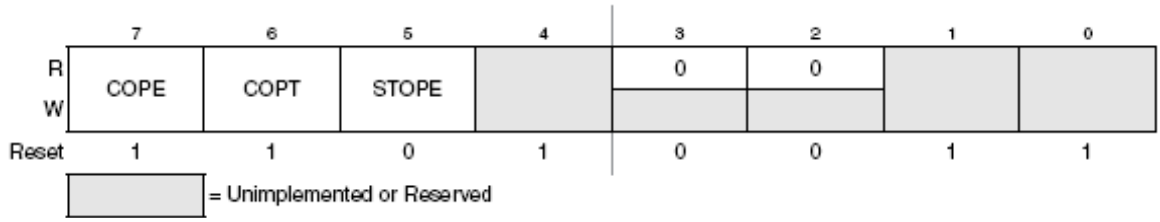
Para configurar os registradores de inicialização do sistema devemos seguir uma ordem. A ordem de configuração dos registradores pode ser obtida na folha de dados do microcontrolador.

Abaixo temos uma sugestão de código de inicialização. Este código deve ser incluído no início da função ***void MCU_init(void)***.

```
  /* SOPT: COPE=0,COPT=1,STOPE=0,??=1,??=0,??=0,??=1,??=1 */
  SOPT = 0x53;

  /* SPMSC1: LVDF=0,LVDACK=0,LVDIE=0,LVDRE=1,LVDSE=1,LVDE=1,??=0,BGBE=0 */
  SPMSC1 = 0x1C;

  /* SPMSC2: LVWF=0,LVWACK=0,LVDV=0,LVWV=0,PPDF=0,PPDACK=0,??=0,PPDC=0 */
  SPMSC2 = 0x00;

  // Saída de Clock Externo do pino PTC2 desabilitado
  /* SMCLK: MPE=0,MCSEL=0 */
  SMCLK = 0;

  /*  Inicialização do sistema de clock */
  /* ICGC1: HGO=1,RANGE=1,REFS=1,CLKS1=1,CLKS0=1,OSCSTEN=0,LOCD=0,??=0 */
  ICGC1 = 0xF8;
  /* ICGC2: LOLRE=0,MFD2=0,MFD1=1,MFD0=1,LOCRE=0,RFD2=0,RFD1=0,RFD0=0 */
  ICGC2 = 0x30;
  while(!ICGS1_LOCK) { /* Espera o módulo ICG estabilizar o clock configurado */
}
```

O primeiro registrador a ser configurado é o SOPT (*System Options Register*). Este registrador pode ser lido a qualquer momento, mas só pode ser escrito uma vez, normalmente na inicialização do programa, após o reset.

O registrador SOPT pode ser visualizado na figura abaixo:



Descrição dos bits de controle:

COPE (*COP Watchdog Enable*)

0 – COP Watchdog Desabilitado

1 – COP Watchdog Habilitado

COPT (*COP Watchdog Timeout*)

0 – Seleção do período curto de estouro (2^{13} ciclos de barramento)

1 – Seleção do período longo de estouro (2^{18} ciclos de barramento)

STOPE (*STOP Mode Enable*)

0 – Modo STOP desabilitado

1 – Modo STOP habilitado

Iremos configurar o registrador SOPT para desligar o COP e o Modo STOP, ou seja, **SOPT = 0x53**. Assim que o programa desenvolvido estiver completo e revisado podemos habilitar o COP, substituindo o valor acima para **SOPT = 0xF3.**

Os próximo registradores a serem configurados são o SPMSC1 (*System Power Management Status and Control 1 Register*) e o SPMSC2 (*System Power Management Status and Control 2 Register*). Com estes registradores podemos configurar os modos de operação do sistema de alimentação do microcontrolador.

	7	6	5	4	3	2	1[1]	0
R	LVDF	0	LVDIE	LVDRE[2]	LVDSE[2]	LVDE[2]		BGBE
W		LVDACK						
Reset	0	0	0	1	1	1	0	0

= Unimplemented or Reserved

[1] Bit 1 is a reserved bit that must always be written to 0.

[2] This bit can be written only one time after reset. Additional writes are ignored.

A descrição dos bits de controle do registrador SPMSC1 é apresentada a seguir:

LVDF (*Low-Voltage Detect Flag*) - Este bit permite somente leitura e indica a detecção de um evento de baixa tensão na alimentação.

LVDACK (*Low-Voltage Detect Acknowledge*) – Este bit permite somente escrita e é utilizado para indicar o reconhecimento de erros causados por baixa tensão na alimentação do microcontrolador.

LVDIE (*Low-Voltage Detect Interrupt Enable*) – Este bit permite habilitar uma requisição de interrupção por *hardware* para a flag LVDF.

0 – Interrupção de *hardware* desabilitada

1 – Gera uma interrupção quando LVDF = 1

LVDRE (*Low-Voltage Detect Reset Enable*) – Permite a geração de um reset associado a LVDF.

0 – LVDF não gera reset

1 – Força um reset do microcontrolador quando LVDF = 1

LVDSE (*Low-Voltage Detect Stop Enable*)

0 – Modo de detecção de baixa tensão desabilitada durante o modo de STOP se LVDE = 1

1 – Modo de detecção de baixa tensão habilitada durante o modo de STOP se LVDE = 1

LVDE (*Low-Voltage Detect Enable*)

0 – Lógicas de detecção de baixa tensão desabilitadas

1 – Lógicas de detecção de baixa tensão habilitadas

BGBE (*Bandgap Buffer Enable*) – Este bit é utilizado para habilitar um buffer para a tensão de referência de bandgap para o módulo conversor Analógico / Digital em um dos canais internos.

0 – *Buffer* de *Bandgap* desabilitado

1 – *Buffer* de *Bandgap* habilitado

	7	6	5	4	3	2	1	0
R	LVWF	0	LVDV	LVWV	PPDF	0		PPDC[1]
W		LVWACK				PPDACK		
Power-on reset:	0[2]	0	0	0	0	0	0	0
LVD reset:	0[2]	0	U	U	0	0	0	0
Any other reset:	0[2]	0	U	U	0	0	0	0

= Unimplemented or Reserved U = Unaffected by reset

1 This bit can be written only one time after reset. Additional writes are ignored.

2 LVWF will be set in the case when V_{Supply} transitions below the trip point or after reset and V_{Supply} is already below V_{LVW}.

Abaixo é apresentada a descrição dos bits de controle do registrador SPMSC2:

LVWF (*Low-Voltage Warning Flag*) – O bit LVWF indica um estado de baixa tensão

0 – Aviso de baixa tensão não presente

1 – Aviso de baixa tensão ativo

LVWACK (*Low-Voltage Warning Acknowledge*) – Este bit permite somente escrita e é utilizado para indicar o reconhecimento de uma condição de baixa tensão.

LVDV (*Low-Voltage Detect Voltage Select*) – Seleciona o nível de tensão que irá ativar o LVDF.

0 – Seleciona Low trip point (VLVD = VLVDL)

1 – Seleciona High trip point (VLVD = VLVDH)

LVWV (*Low-Voltage Warning Voltage Select*) - Seleciona o nível de tensão que irá ativar o LVWF.

0 – Seleciona Low trip point (VLVW = VLVWL)

1 – Seleciona High trip point selected (VLVW = VLVWH)

PPDF (*Partial Power Down Flag*) – Este bit indica quando o microcontrolador sair do modo stop2

0 – não houve recuperação do modo stop2

1 – Recuperação do modo stop2

PPDACK (*Partial Power Down Acknowledge*) – Escrever “1” para este bit limpa o bit PPDF.

PPDC (*Partial Power Down Control*) – Este bit somente de escrita controla quando os modos stop2 ou stop3 são selecionados.

0 – modo stop3 habilitado

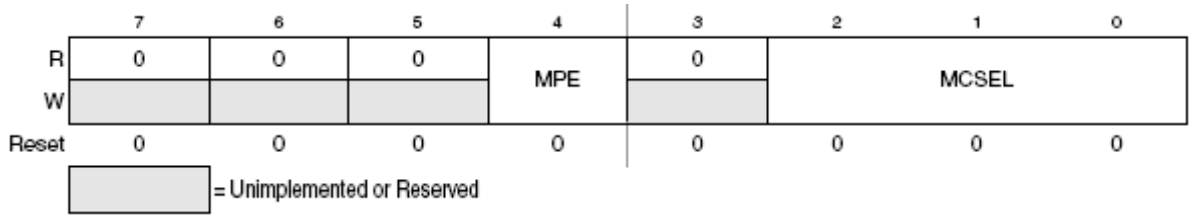
1 – modo stop2 habilitado, *power down* parcial

Utilize a tabela abaixo para definir a configuração dos valores para **LVDV** e **LVWV**.

Low-voltage detection threshold — high range V_{DD} falling V_{DD} rising	V_{LVDH}	4.2 4.3	4.3 4.4	4.4 4.5	V
Low-voltage detection threshold — low range V_{DD} falling V_{DD} rising	V_{LVDL}	2.48 2.54	2.56 2.62	2.64 2.7	V
Low-voltage warning threshold — high range V_{DD} falling V_{DD} rising	V_{LVWH}	4.2 4.3	4.3 4.4	4.4 4.5	V
Low-voltage warning threshold — low range V_{DD} falling V_{DD} rising	V_{LVWL}	2.48 2.54	2.56 2.62	2.64 2.7	V

Iremos configurar o registrador SPMSC1 para habilitar a lógica de detecção de baixa tensão tanto no modo STOP quanto em operação normal. Ainda, quando a condição de baixa tensão for detectada irá gerar um reset no microcontrolador. Desta forma, **SPMSC1 = 0x1C**. O registrador SPMSC2 será configurado no modo padrão, ou seja, **SPMSC2 = 0x00.**

O próximo registrador a ser configurado é o SMCLK. Este registrador permite criarmos um sinal de clock externo, baseado no clock do barramento do microcontrolador. Este clock externo, se habilitado, tem como local de saída o bit 2 da porta C, ou seja, no pino **PTC2**. A configuração dos bits de controle do registrador SMCLK é apresentada abaixo:



Abaixo a descrição dos bits de controle do registrador SMCLK é apresentada:

MPE (*MCLK Pin Enable*) – Este bit é utilizado para habilitar a função MCLK.

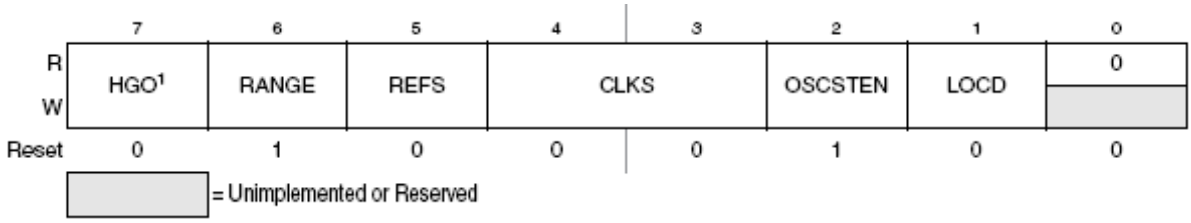
0 – MCLK desabilitado

1 – Saída MCLK habilitada no pino PTC2.

MCSEL (*MCLK Divide Select*) – Estes bits são utilizados para selecionar a taxa de divisão para a saída MCLK, de acordo com a equação abaixo.

$$frequencia\ MCLK = \frac{(frequencia\ de\ clock\ do\ barramento)}{(2 * MCSEL)}$$

A partir de agora iremos configurar os registradores relativos ao clock do barramento. O primeiro registrador de configuração do clock é o ICGC1. Abaixo temos a configuração dos bits de controle deste registrador.



HGO (*High Gain Oscillator Select*) – Este bit é utilizado para selecionar entre operação de baixo consumo ou alto ganho para melhor imunidade a ruídos.

0 – Oscilador configurado para operação de baixo consumo

1 – Oscilador configurado para operação com alto ganho.

RANGE (*Frequency Range Select*) – Este bit é utilizado para controlar a referência de divisão para o oscilador e para configurar o fator de multiplicação (P) do loop FLL. Ele seleciona uma ou duas referências de freqüência para o ICG.

0 – Oscilador configurado para escala de baixa freqüência. Fator de multiplicação do Loop FLL configurado para P = 64.

1 – Oscilador configurado para escala de alta freqüência. Fator de multiplicação do Loop FLL configurado para P = 1.

REFS (*External Reference Select*) – Seletor de referência externa.

0 – Clock externo selecionado como referência externa

1 – Cristal ou ressonador selecionado como referência externa para o oscilador.

CLKS (*Clock Mode Select*) – Seleção de modo de clock

00 – Clock interno selecionado

01 – FLL acoplado, utilizando referência interna

10 – FLL *bypassed*, utilizando referência externa

11 – FLL acoplado, utilizando referência externa

OSC2TEN (*Enable Oscillator in Off Mode*)

0 – Oscilador habilitado quando o módulo ICG é desligado.

1 – Oscilador habilitado quando o módulo ICG é desligado.

LOCD (*Loss of Clock Disable*)

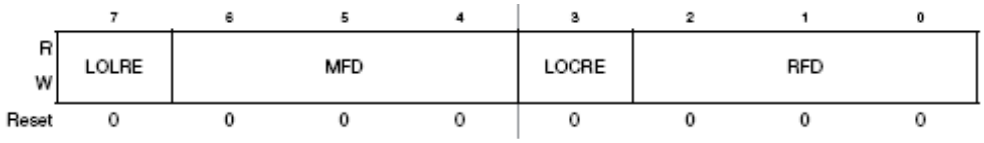
0 – Detecção de perda de clock habilitada

1 – Detecção de perda de clock desligada

O módulo FLL (*Frequeny-locked loop*) é utilizado para multiplicar a freqüência de referência (interna ou externa) de modo a obtermos clocks de barramento entre 4 MHz e 20 MHz. A freqüência de referência interna do microcontrolador MC9S08AW60 é 243 Khz. OBS: O clock gerado através dos registradores ICGC1 e ICGC2 será o clock do barramento multiplicado por 2, ou seja, para obtermos a máxima freqüência de barramento (20 Mhz), devemos configurar os registradores para obtermos uma freqüência do módulo ICG de 40 Mhz.

Devemos configurar o registrador ICGC1 levando em conta o *hardware* utilizado. Como a placa desenvolvida para o MC9S08AW60 é constituída por um cristal de 4 Mhz configurado no modo de alto ganho, iremos configurar o registrador **ICGC1** para **0xF8**. Desta forma estaremos habilitando o módulo FLL acoplado utilizando referência externa (cristal), com oscilador operando em modo de alto ganho e fator de multiplicação do FLL igual a 1 (P = 1). Ainda, oscilador habilitando quando o ICG estiver desligado e detecção de perda de clock ativada.

Dando seqüência a configuração do clock, iremos apresentar as configurações possíveis através do registrador ICGC2.



Os bits de controle do registrador acima indicado são:

LOLRE (*Loss of Lock Reset Enable*) – Configura a ação a ser tomada no caso de instabilidade do clock gerado pelo módulo ICG.

0 – Gera uma interrupção no caso de instabilidade do clock gerado

1 – Gera um reset no caso de instabilidade do clock gerado

MFD (*Multiplication Factor*) – Fator de multiplicação (M) do loop FLL.

000 – Fator de multiplicação = 4

001 – Fator de multiplicação = 6

010 – Fator de multiplicação = 8

011 – Fator de multiplicação = 10

100 – Fator de multiplicação = 12

101 – Fator de multiplicação = 14

110 – Fator de multiplicação = 16

111 – Fator de multiplicação = 18

LOCRE (*Loss of Clock Reset Enable*) - Configura a ação a ser tomada no caso de instabilidade do clock do sistema.

0 – Gera uma interrupção no caso de instabilidade do clock do sistema

1 – Gera um reset no caso de instabilidade do clock do sistema

RFD (*Reduced Frequency Divider*) – Fator de divisão (R) do loop FLL.

000 – Fator de divisão = 1

001 – Fator de divisão = 2

010 – Fator de divisão = 4

011 – Fator de divisão = 8

100 – Fator de divisão = 16

101 – Fator de divisão = 32

110 – Fator de divisão = 64

111 – Fator de divisão = 128

A equação de geração do clock do módulo ICG é apresentada abaixo:

$$Frequencia\ ICG = \frac{(P\ x\ N)}{R}\ x\ Frequencia\ de\ Referencia$$

Iremos configurar o registrador ICGC2 para obtermos a máxima freqüência do módulo ICG (40 Mhz). Desta forma, **ICGC2 = 0x30**. Assim iremos obter **N = 10** e **R = 1**, além de configurarmos o sistema para gerar uma interrupção caso ocorra uma perda de clock.

$$Frequencia\ ICG = \frac{(1\ x\ 10)}{1}\ x\ 4\ Mhz = 40\ Mhz$$

e ainda,

$$Frequencia\ de\ Barramento = \frac{(\ frequencia\ ICG)}{2} = 20\ Mhz$$

Após configurarmos os registradores ICGC1 e ICGC2 devemos esperar que o clock gerado pelo módulo ICG estabilize. Podemos verificar este fato através do bit **LOCK**, do registrador **ICGS1** (Registrador 1 de estado do módulo ICG). No momento em que o clock gerado estiver estável, este bit será levado a condição de nível lógico alto, ou seja, "1". O código abaixo exemplifica esta espera.

```
while(!ICGS1_LOCK) {     /* Espera o módulo ICG sintonizar o clock configurado */
}
```

3. Configuração das portas de entrada e saída

Quanto configuramos as portas de entrada e saída de um microcontrolador devemos levar em consideração diversos aspectos, entre eles, condição inicial da porta, direção dos pinos e acionamento de pull-up.

A seguir iremos apresentar os principais registradores de configuração das portas de entrada e saída do microcontrolador MC9S08AW60, utilizando a porta "A" do microcontrolador como referência.

*Registrador de Dados da Porta "A" (**PTAD**)*

	7	6	5	4	3	2	1	0
R W	PTAD7	PTAD6	PTAD5	PTAD4	PTAD3	PTAD2	PTAD1	PTAD0
Reset	0	0	0	0	0	0	0	0

Quando os pinos forem configurados para entrada, a leitura dos mesmos irá retornar o nível lógico do pino associado. Caso os pinos estejam configurados para saída, a leitura dos pinos irá retornar o último valor escrito para o registrador **PTAD**. Ainda, com os pinos configurados para saída o nível lógico dos mesmos será imposto pelo valor do registrador **PTAD**.

*Registrador de Direção da Porta "A" (**PTADD**)*

	7	6	5	4	3	2	1	0
R W	PTADD7	PTADD6	PTADD5	PTADD4	PTADD3	PTADD2	PTADD1	PTADD0
Reset	0	0	0	0	0	0	0	0

Este registrador de escrita / leitura controla a direção dos pinos da porta A.

0 – pinos configurados como entrada (*driver* de saída desabilitado)

1 – *Driver* de saída habilitado para o bit n da porta A

*Registrador de Pull-Up da Porta "A" (**PTAPE**)*

	7	6	5	4	3	2	1	0
R W	PTAPE7	PTAPE6	PTAPE5	PTAPE4	PTAPE3	PTAPE2	PTAPE1	PTAPE0
Reset	0	0	0	0	0	0	0	0

Cada um dos bits de controle deste registrador determina se o dispositivo interno de *pull-up* está habilitado para o pino da porta A associado a ele. Para os pinos da porta A configurados como saída este registrador não tem efeito e o dispositivo interno de *pull-up* será desabilitado.

0 – Dispositivo de *pull-up* interno desabilitado para a porta A bit n.

1 – Dispositivo de *pull-up* interno habilitado para a porta A bit n.

*Registrador de Controle de Slew Rate da Porta "A" (**PTASE**)*

	7	6	5	4	3	2	1	0
R W	PTASE7	PTASE6	PTASE5	PTASE4	PTASE3	PTASE2	PTASE1	PTASE0
Reset	0	0	0	0	0	0	0	0

Slew Rate é a taxa de variação (dV/dt) da saída quando temos um "degrau". Cada um destes bits de controle determina quando o controle de saída *slew rate* é habilitado para o pino da porta A associado. Para os pinos da porta A configurados como entrada este registrador não tem efeito.

0 – Controle de saída *Slew Rate* desabilitado para o bit n da porta A.

1 – Controle de saída *Slew Rate* habilitado para o bit n da porta A.

A figura abaixo apresenta os tempos de subida e descida de sinal nas portas do microcontrolador, com a taxa de variação (*slew rate*) habilitada e desabilitada.

Port rise and fall time — Low output drive (PTxDS = 0) (load = 50 pF)[5]	t_{Rise}, t_{Fall}	—	40	—	ns
Slew rate control disabled (PTxSE = 0)		—	75	—	
Slew rate control enabled (PTxSE = 1)					
Port rise and fall time — High output drive (PTxDS = 1) (load = 50 pF)	t_{Rise}, t_{Fall}	—	11	—	ns
Slew rate control disabled (PTxSE = 0)		—	35	—	
Slew rate control enabled (PTxSE = 1)					

*Registrador de Seleção da Capacidade de corrente do driver da Porta "A" (**PTADS**)*

	7	6	5	4	3	2	1	0
R W	PTADS7	PTADS6	PTADS5	PTADS4	PTADS3	PTADS2	PTADS1	PTADS0
Reset	0	0	0	0	0	0	0	0

Cada um destes bits de controle permite a seleção entre *driver* de saída com baixa ou alta capacidade de corrente para o pino da porta A associado. No entanto, devemos tomar o cuidado de evitar que a soma das correntes dos pinos não ultrapasse o limite total de corrente do microcontrolador.

0 – Driver de saída com baixa capacidade de corrente habilitado para o bit n da porta A.

1 – Driver de saída com alta capacidade de corrente habilitado para o bit n da porta A.

A tabela a seguir informa os níveis de corrente para o *driver* em alta e baixa capacidade de corrente.

OBS: Pinos que não forem utilizados na aplicação desenvolvida devem ser conectados a um terminador. Esta medida previne o excesso de corrente causado pela flutuação das entradas e melhora a imunidade a ruídos e transientes. As terminações podem ser implementadas através de uma das opções abaixo relacionadas:

- Configurar os pinos não utilizados como saída e força-los para nível lógico alto ou baixo
- Configurar os pinos não utilizados como entrada e utilizar um *pull-up* interno ou externo.

Parameter	Symbol	Min	Typ[1]	Max	Unit
Output high voltage — Low Drive (PTxDSn = 0)	V_{OH}				V
5 V, I_{Load} = –2 mA		V_{DD} – 1.5	—	—	
3 V, I_{Load} = –0.6 mA		V_{DD} – 1.5	—	—	
5 V, I_{Load} = –0.4 mA		V_{DD} – 0.8	—	—	
3 V, I_{Load} = –0.24 mA		V_{DD} – 0.8	—	—	
Output high voltage — High Drive (PTxDSn = 1)					
5 V, I_{Load} = –10 mA		V_{DD} – 1.5	—	—	
3 V, I_{Load} = –3 mA		V_{DD} – 1.5	—	—	
5 V, I_{Load} = –2 mA		V_{DD} – 0.8	—	—	
3 V, I_{Load} = –0.4 mA		V_{DD} – 0.8	—	—	
Output low voltage — Low Drive (PTxDSn = 0)	V_{OL}				V
5 V, I_{Load} = 2 mA		—	—	1.5	
3 V, I_{Load} = 0.6 mA		—	—	1.5	
5 V, I_{Load} = 0.4 mA		—	—	0.8	
3 V, I_{Load} = 0.24 mA		—	—	0.8	
Output low voltage — High Drive (PTxDSn = 1)					
5 V, I_{Load} = 10 mA		—	—	1.5	
3 V, I_{Load} = 3 mA		—	—	1.5	
5 V, I_{Load} = 2 mA		—	—	0.8	
3 V, I_{Load} = 0.4 mA		—	—	0.8	
Output high current — Max total I_{OH} for all ports					mA
5V	I_{OHT}	—	—	100	
3V		—	—	60	
Output low current — Max total I_{OL} for all ports					mA
5V	I_{OLT}	—	—	100	
3V		—	—	60	
Input high voltage; all digital inputs	V_{IH}	0.65 x V_{DD}	—	—	V
Input low voltage; all digital inputs	V_{IL}	—	—	0.35 x V_{DD}	

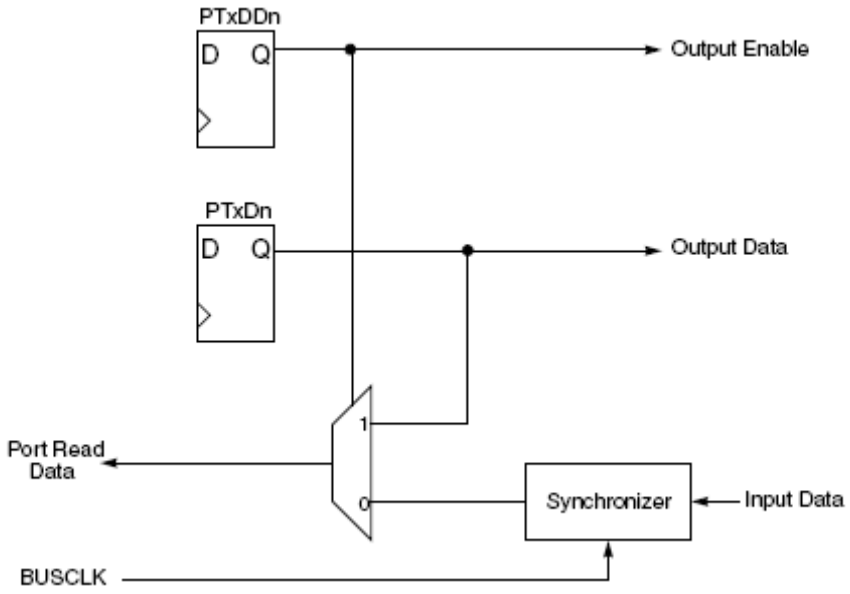


Diagrama de blocos das portas de entrada e saída paralelas

A figura abaixo apresenta os valores limites de tensão na alimentação, bem como a máxima corrente por pino e a máxima corrente total na alimentação do microcontrolador.

Rating	Symbol	Value	Unit
Supply voltage	V_{DD}	–0.3 to + 5.8	V
Input voltage	V_{In}	– 0.3 to V_{DD} + 0.3	V
Instantaneous maximum current Single pin limit (applies to all port pins)[1, 2, 3]	I_D	± 25	mA
Maximum current into V_{DD}	I_{DD}	120	mA

A seguir é apresentado um exemplo de código de inicialização para a porta A com todos os pinos configurados como saída com *driver* em baixa capacidade de corrente e controle de slew rate ativo. Ainda, a porta B configurada com todos os pinos para entrada com *pull-up* ativo.

```
/* Inicialização das Portas A e B do microcontrolador MC9S08AW60 */

// Controle de Slew Rate ativo para a Porta A
/* PTASE: PTASE7=1,PTASE6=1,PTASE5=1,PTASE4=1,PTASE3=1,PTASE2=1,PTASE1=1,PTASE0=1 */
PTASE = 0xFF;
// Controle de Slew Rate desabilitado para a Porta B
/* PTBSE: PTBSE7=1,PTBSE6=1,PTBSE5=1,PTBSE4=1,PTBSE3=1,PTBSE2=1,PTBSE1=1,PTBSE0=1 */
PTBSE = 0x00;

// Driver de corrente da porta A configurado para baixa capacidade de corrente
/* PTADS: PTADS7=0,PTADS6=0,PTADS5=0,PTADS4=0,PTADS3=0,PTADS2=0,PTADS1=0,PTADS0=0 */
PTADS = 0x00;
// Driver de corrente da porta A configurado para baixa capacidade de corrente
/* PTBDS: PTBDS7=0,PTBDS6=0,PTBDS5=0,PTBDS4=0,PTBDS3=0,PTBDS2=0,PTBDS1=0,PTBDS0=0 */
PTBDS = 0x00;

// Registradores de dados das portas A e B inicializados com nível lógico 0
PTAD = 0x00;
PTBD = 0x00;

// Registrador de pull-up da porta B com todos os bits ativados
PTBPE = 0xFF;

// Registrador de direção da porta A com todos os bits configurados para saída
PTADD = 0xFF;
// Registrador de direção da porta B com todos os bits configurados para entrada
PTBDD = 0x00;
```

4. Configuração da Interrupção de Estouro de Tempo

Um microprocessador deve possuir um relógio (*clock*). O relógio pode ser implementado por um cristal oscilador que sincroniza todo o funcionamento do microprocessador, controlando o tempo de cada um dos eventos relacionados aos dispositivos integrados a ele.

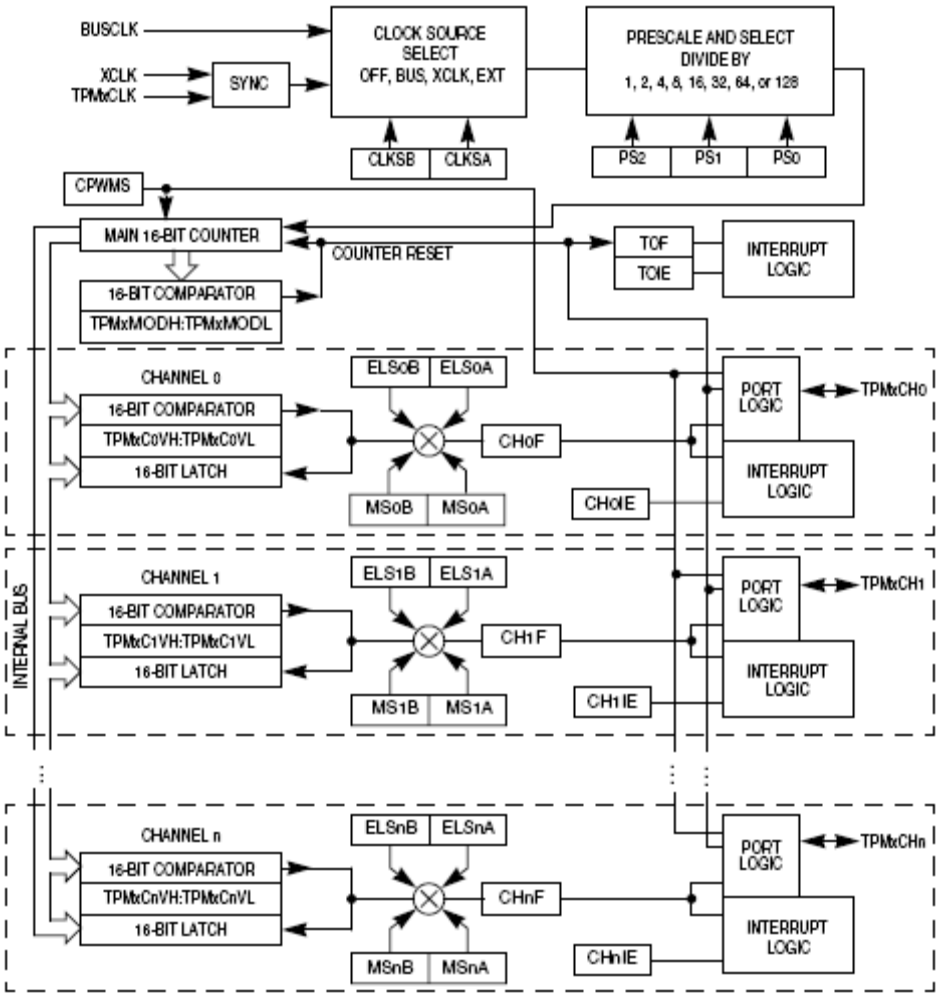
Os temporizadores utilizam a base de tempo do relógio para poder implementar contagens de tempo bem específicas e configuráveis. Estes utilizam contadores, incrementados na mesma base de tempo do relógio. Desta forma é possível descrever tempo em número de ciclos de um relógio. Por exemplo, imagine um microprocessador utilizando um relógio de 20 Mhz. O período relativo a esta freqüência é 50 ns. Podemos representar então um tempo de 1ms através de períodos de 50ns, obtendo o valor de 20000. Ou seja, se incrementarmos um contador a cada ciclo de relógio, no caso 50ns, quando este contador atingir o valor de 20.000, teremos atingido a contagem de 1ms.

Para que este método seja aplicado, existe a necessidade da utilização de pelo menos 2 registradores. O registrador que será incrementado e o registrador que conterá o valor a ser atingido. No entanto, existe um problema relacionado a este método. Imagine que precisemos de um tempo na ordem de vários milissegundos ou segundos, por exemplo, 400ms. Se estivermos utilizando um relógio de 20 Mhz, seria necessário registradores de 24 bits para representar o valor de $8x10^6$. Com o intuito de reduzir o tamanho destes contadores, os temporizadores apresentam a possibilidade de divisão do relógio, normalmente por valores múltiplos de 2. No caso do exemplo acima, poderíamos dividir o relógio por 128, obtendo uma freqüência de 156,250 KHz. Então, para representarmos 400 milissegundos, seria necessário uma contagem de 62500 períodos de 6.4µs, ou seja, um valor que pode ser representado em 16 bits.

Devemos lembrar que a divisão terá influência somente no tempo de incrementação do contador, continuando o barramento interno do microprocessador a operar com o relógio original.

O processo de configuração do temporizador do microcontrolador MC9S08AW60 é apresentado a seguir.

Os microcontroladores da linha HCS08 normalmente possuem 1 ou 2 temporizadores. Os registradores relativos a estes temporizadores apresentam nomes semelhantes, tendo apenas o número no temporizador para diferenciá-los. Os três registradores de configuração do temporizador neste microcontrolador são: TPMxSC, TPMxCNT e TPMxMOD, sendo os dois últimos de 16 bits. Devemos substituir o x do nome dos registradores pelo número do temporizador, no caso do MC9S08AW60, 1 ou 2. Abaixo é apresentado o diagrama de blocos do temporizador destes microcontroladores. É importante ressaltar que alguns dos registradores apresentados na figura são relativos ao módulo PWM e captura de entrada.



*Registrador de Controle e Estado do Temporizador x **(TPMxSC)***

	7	6	5	4	3	2	1	0
R	TOF	TOIE	CPWMS	CLKSB	CLKSA	PS2	PS1	PS0
W								
Reset	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

A descrição dos bits de controle deste registrador é apresentada abaixo:

TOF (*Timer Overflow Flag*) - Este bit de escrita/leitura é levado para 1 quando o registrador contador (TPMxCNT) atinge o valor do registrador de módulo de contagem (TPMxMOD), condição essa que indica o estouro da contagem de tempo. O procedimento correto para limpar esta indicação é ler o registrador TPMxSC e escrever um “0” lógico para o bit TOF.

1 - O módulo temporizador atingiu o valor desejado

0 - O módulo temporizador não atingiu o valor desejado

TOIE (*Timer Overflow Interrupt Enable Bit*): Este bit de escrita/leitura habilita a interrupção do temporizador quando o bit TOF for definido em 1.

1 - Interrupção do temporizador ativa

0 - Interrupção do temporizador desabilitada

CPWMS (*Center-Aligned PWM Select*) – Iremos comentar sobre este bit na seção sobre PWM.

CLKS (*Clock Source Select)* – Estes dois bits são utilizados para desligar o módulo TPM ou selecionar uma das três possíveis fontes de clock para incrementar o contador.

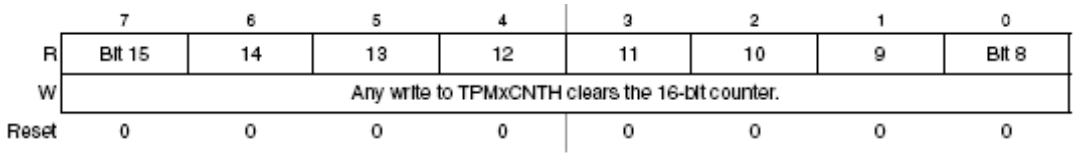
CLKSB	CLKSA	Fonte de clock do módulo TPM
0	0	TPM desabilitado
0	1	Clock do barramento (BUSCLK)
1	0	Clock fixo do sistema (XCLK)
1	1	Fonte externa (TPMxCLK)

PS[2:0] (*Prescaler Setect Bits*) – Bits de pré-escala da base de tempo. Estes bits de escrita/leitura selecionam um dos oito possíveis valores de divisão da base de tempo do relógio para utilização como base de tempo do temporizador.

PS2	PS1	PS0	Base de tempo do temporizador
0	0	0	Clock do módulo TPM / 1
0	0	1	Clock do módulo TPM / 2
0	1	0	Clock do módulo TPM / 4
0	1	1	Clock do módulo TPM / 8
1	0	0	Clock do módulo TPM / 16
1	0	1	Clock do módulo TPM / 32
1	1	0	Clock do módulo TPM / 64
1	1	1	Clock do módulo TPM / 128

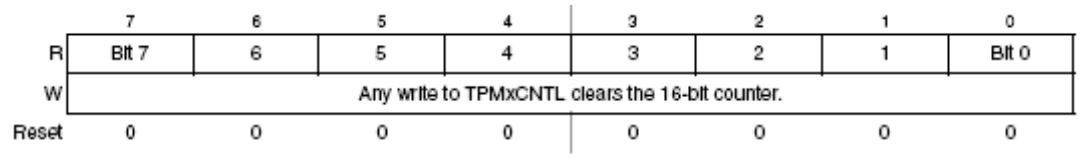
*Registradores de Contagem de Tempo (**TPMxCNT**)*:

Estes registradores são somente de leitura e contém o valor mais significativo (TPMxCNTH) e menos significativo (TPMxCNTL) do contador do temporizador. A leitura do registrador mais significativo deve ser realizada primeiro.



	7	6	5	4	3	2	1	0
R	Bit 15	14	13	12	11	10	9	Bit 8
W	Any write to TPMxCNTH clears the 16-bit counter.							
Reset	0	0	0	0	0	0	0	0

Figure 10-4. Timer x Counter Register High (TPMxCNTH)

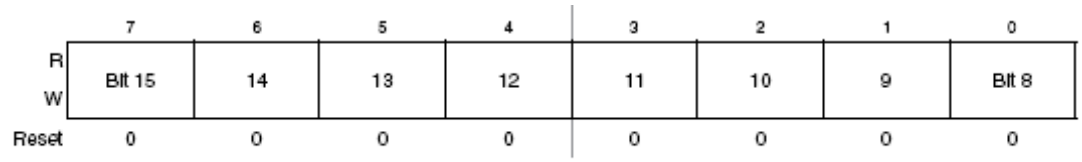


	7	6	5	4	3	2	1	0
R	Bit 7	6	5	4	3	2	1	Bit 0
W	Any write to TPMxCNTL clears the 16-bit counter.							
Reset	0	0	0	0	0	0	0	0

Figure 10-5. Timer x Counter Register Low (TPMxCNTL)

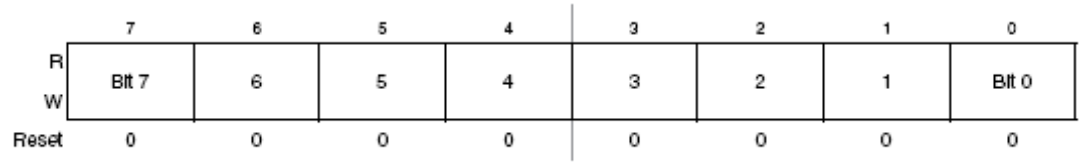
*Registradores de Módulo do Temporizador (**TPMxMOD**)*

Estes registradores de escrita/leitura contém o valor do módulo da contagem do temporizador. Quando os registradores de contagem (TPMxCNT) atingem o valor dos registradores de módulo (TPMxMOD), o bit TOF torna-se nível lógico “1” e os registradores de contagem resumem a contagem para $0000 até o próximo passo de clock. Escrever no registrador TPMxMODH inibe o bit TOF até que o registrador TPMxMODL seja escrito.



	7	6	5	4	3	2	1	0
R W	Bit 15	14	13	12	11	10	9	Bit 8
Reset	0	0	0	0	0	0	0	0

Figure 10-6. Timer x Counter Modulo Register High (TPMxMODH)



	7	6	5	4	3	2	1	0
R W	Bit 7	6	5	4	3	2	1	Bit 0
Reset	0	0	0	0	0	0	0	0

Figure 10-7. Timer x Counter Modulo Register Low (TPMxMODL)

Estando os registradores a serem configurados apresentados pode-se demonstrar um exemplo de utilização. Abaixo será implementada a configuração de um temporizador utilizando um clock de barramento de 20 MHz para obter a base de tempo de 50µs.

Período relativo a base de tempo: $Período = \frac{1}{(20 \times 10^6)} = 50 \times 10^{-9} segundos$

Cálculo do valor de módulo de tempo: $Módulo = \frac{50\times10^{-6}}{(50\times10^{-9})} = 1000$

O valor 1000 deve ser escrito no registrador TPMxMOD. Abaixo o exemplo de configuração dos registradores para este caso é apresentado em linguagem “c”, utilizando o temporizador 1 do microcontrolador MC9S08AW60.

```
// Configuração do timer para 50us
/* TPM1SC: TOF=0,TOIE=0,CPWMS=0,CLKSB=0,CLKSA=0,PS2=0,PS1=0,PS0=0 */
TPM1SC = 0;                  /* Para o contador */
/* TPM1CNTH: BIT15=0,BIT14=0,BIT13=0,BIT12=0,BIT11=0,BIT10=0,BIT9=0,BIT8=0 */
TPM1CNT = 0;                 /* Reseta o contador */
/* Configura o valor correto no registrador de módulo para obtermos 50us */
TPM1MOD = 1000;
/* Configura a pré-escala para 1, habilita a interrupção */
// Utiliza como base o clock do barramento
/* TPM1SC: TOF=0,TOIE=1,CPWMS=0,CLKSB=0,CLKSA=1,PS2=0,PS1=0,PS0=0 */
TPM1SC = 0x48;
```

Para demonstrar um caso onde se faz necessário a divisão da base de tempo, o temporizador será configurado utilizando um relógio de 20 MHz para obter a base de tempo de 100ms.

Período relativo a base de tempo: $Período = \frac{1}{(20\times10^{6})} = 50\times10^{-9}$

Cálculo do valor de módulo de tempo: $Módulo = \frac{100\times10^{-3}}{(50\times10^{-9})} = 2000000$

Para representar 2000000 é necessário mais do que os 16 bits disponíveis. Desta forma se faz necessário a divisão da base de tempo. Dividindo 2000000 pelo maior valor possível em 16 bits, encontramos 30,518. Assumimos então o próximo valor válido de divisão, ou seja, 32, como fator de divisão da base de tempo.

Fator de divisão da base de tempo: $Fator = \frac{2000000}{65535} = 30{,}518$

E, recalculando os valores de configuração:

Período relativo a base de tempo: $$Período = \frac{1}{(\frac{20\times10^{6}}{32})} = 1{,}6\times10^{-6}$$

Cálculo do valor de módulo de tempo: $$Módulo = \frac{100\times10^{-3}}{(1{,}6\times10^{-6})} = 62500$$

Sendo 62500 um valor válido em 16 bits, a configuração do temporizador em linguagem “c” é apresentada abaixo:

```
// Configuração do timer para 50us
/* TPM1SC: TOF=0,TOIE=0,CPWMS=0,CLKSB=0,CLKSA=0,PS2=0,PS1=0,PS0=0 */
TPM1SC = 0;                  /* Para o contador */
/* TPM1CNTH: BIT15=0,BIT14=0,BIT13=0,BIT12=0,BIT11=0,BIT10=0,BIT9=0,BIT8=0 */
TPM1CNT = 0;                 /* Reseta o contador */
/* Configura o valor correto no registrador de módulo para obtermos 100ms */
TPM1MOD = 62500;
/* Configura a pré-escala para 32, habilita a interrupção */
// Utiliza como base o clock do barramento
/* TPM1SC: TOF=0,TOIE=1,CPWMS=0,CLKSB=0,CLKSA=1,PS2=1,PS1=0,PS0=1 */
TPM1SC = 0x4D;
```

Obs: Devemos lembrar que o código contido nas interrupções deve ser o menor possível, com o intuito de evitar que o tempo de execução deste código seja superior ao tempo da próxima interrupção. Por exemplo, se configurarmos a interrupção do temporizador para ocorrer a cada 50µs, o tempo de execução do código contido nesta interrupção não deve ser superior a esta base de tempo.

Exemplo de utilização da interrupção do temporizador (esta interrupção é conhecida como interrupção de estouro de tempo):

```
/* Interrupção de timer com código para incrementar a variável i e alterar o
estado do pino PTAD0 a cada interrupção */
interrupt void timer(void){
  TPM1SC_TOF = 0        // Limpa a flag de condição de estouro do temporizador
  PTAD_PTAD0 = PTAD_PTAD0 ^ 1;
  i++;
}
```

5. Configuração do Conversor Analógico / Digital

De um modo geral, os sinais encontrados no mundo real são contínuos (ou analógicos, pois variam no tempo de forma contínua), como, por exemplo: a intensidade luminosa de um ambiente que se modifica com a distância, a aceleração de um carro de corrida, etc. Os sinais manipulados por computadores e sistemas embarcados são digitais, como por exemplo, uma faixa de áudio lida de um *compact disk*.

A conversão analógico-digital (A/D) é o processo que possibilita a representação de sinais analógicos no mundo digital. Desta forma é possível utilizar os dados extraídos do mundo real para cálculos ou operar seus valores.

Em geral, o conversor A/D está presente internamente nos processadores e controladores de sinais digitais e alguns microcontroladores, mas também existem circuitos integrados dedicados a este fim.

Basicamente é um bloco que apresenta portas de entrada e saída. A entrada recebe sinais elétricos de forma contínua e possui uma faixa de tensão de entrada máxima e mínima. Nos microcontroladores que possuem um conversor A/D e operam na faixa de 5V, geralmente a faixa de tensão aceita sinais elétricos entre -5V e +5V.

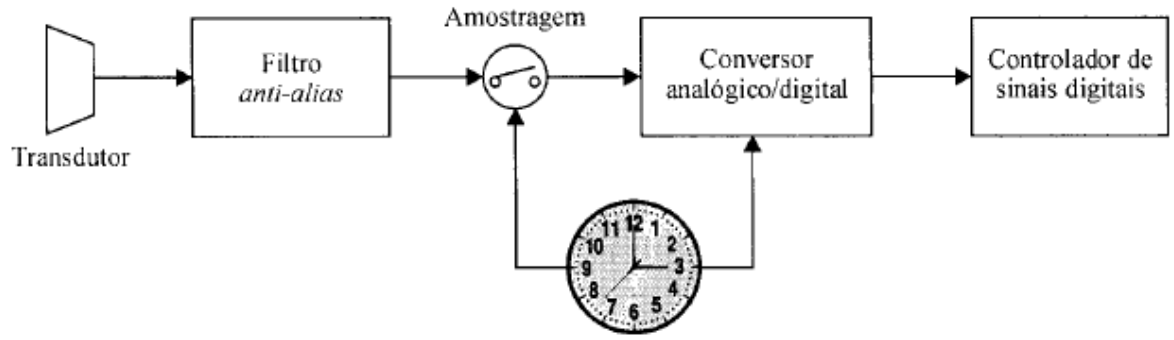
Na saída o sinal é amostrado em um dado intervalo de tempo fixo (determinado pela freqüência de amostragem). Esta amostra disponibiliza um certo valor que representa o sinal original naquele momento (quantização). As características de quantização estão relacionadas à precisão do conversor.

A informação digital é diferente de sua forma original contínua em dois aspectos fundamentais:

- É amostrada porque é baseada em amostragens, ou seja, são realizadas leituras em um intervalo fixo de tempo no sinal contínuo;
- É quantizada porque é atribuído um valor proporcional a cada amostra.

A figura abaixo detalha um pouco mais as três etapas mais importantes do processo: a aquisição, a amostragem e o processamento.

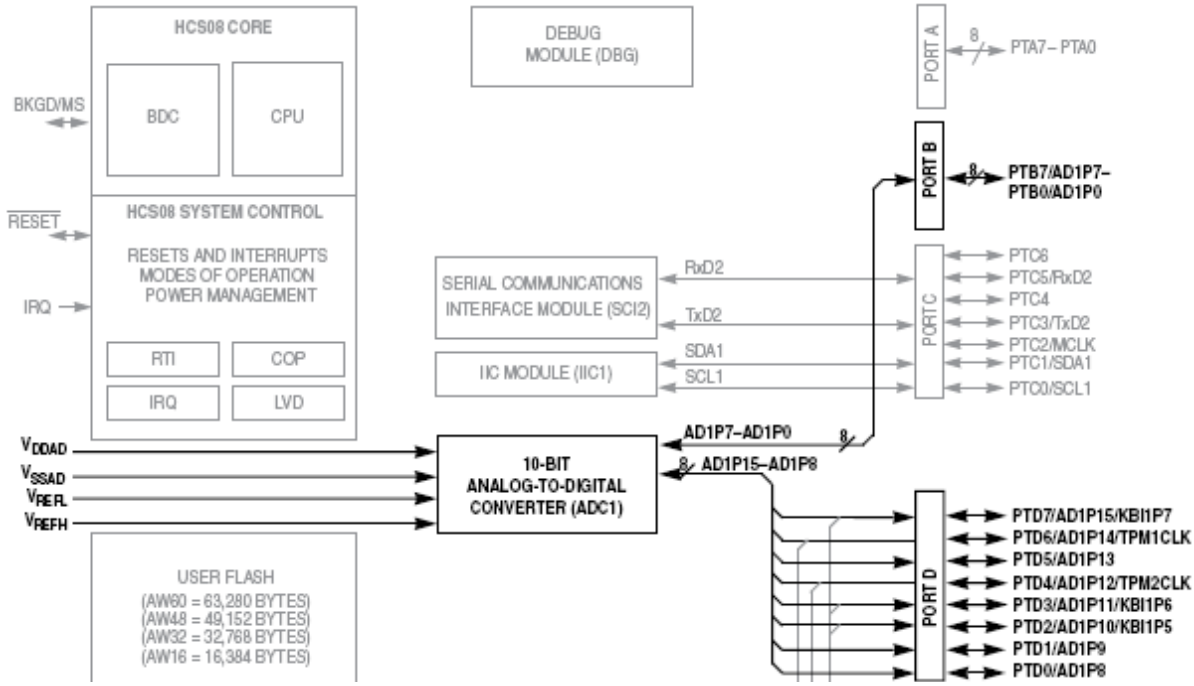
Neste diagrama de blocos, o sinal analógico é capturado pelo transdutor (sensor), em seguida passa por um filtro, denominado de *anti-alias*, a fim de diminuir os ruídos. A chave representa a freqüência de amostragem do conversor A/D, entregando ao processador o sinal digitalizado.



A freqüência de amostragem é o número de amostras capturadas em um segundo. Esta freqüência é dada em *Hertz* (Hz) e é considerada adequada quando se pode reconstruir o sinal analógico razoável a partir de amostras obtidas na conversão.

A taxa de conversão ou freqüência de amostragem é de suma importância para o processamento de sinais reais. Para obter uma taxa de amostragem adequada pode-se utilizar os teoremas de *Nyquist* ou *Shannon*. Estes teoremas indicam que um sinal contínuo x(t) pode ser amostrado adequadamente se tiver banda limitada, ou seja, seu espectro de freqüência não pode conter freqüências acima de um valor máximo (*Fmáx* – freqüência máxima). Ainda, outro ponto importante é que a taxa de amostragem (*Fa* – Freqüência de amostragem) deve ser escolhida para ser no mínimo duas vezes maior que a freqüência máxima (*Fmáx*). Por exemplo, para representar um sinal de áudio com freqüências até 10 KHz, o conversor A/D deve amostrar esses sinais utilizando uma freqüência de amostragem de no mínimo 20 Khz.

Abaixo é apresentada a conexão dos pinos aos canais do conversor A/D do microcontrolador MC9S08AW60.



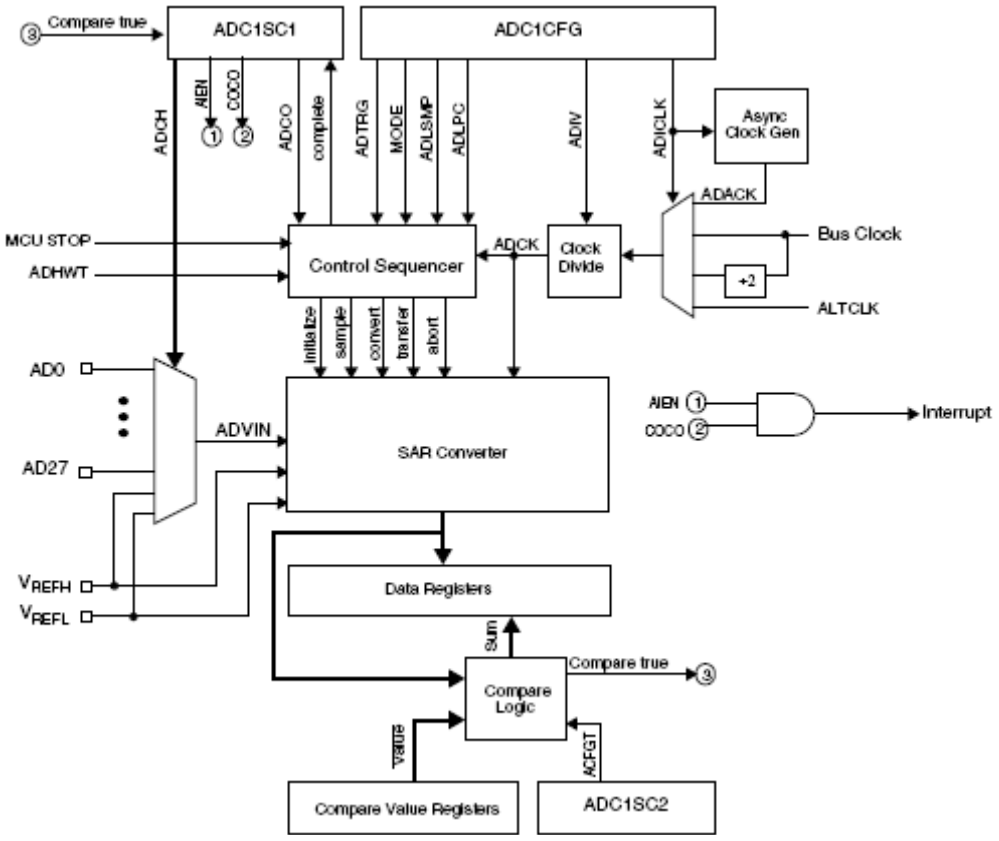
Este microcontrolador possui um conversor analógico/digital de 10 bits operando pela técnica de aproximação sucessiva. Este conversor é projetado para suportar até 28 entradas analógicas separadas (AD0-AD27). Somente 18 (AD0-AD15, AD26 e AD27) das possíveis entradas são implementadas na série de microcontroladores MC9S08AW. Estas entradas, conhecidas por canais de A/D, são selecionadas através dos bits ADCH, contidos no registrador ADC1SC1.

Este módulo A/D é capaz de realizar conversões usando o *clock* do barramento do microcontrolador, o *clock* dividido por dois, o clock assíncrono local (ADACK) dentro do conversor ou através de um *clock* alternativo (ALTCLK).

Características do módulo A/D:

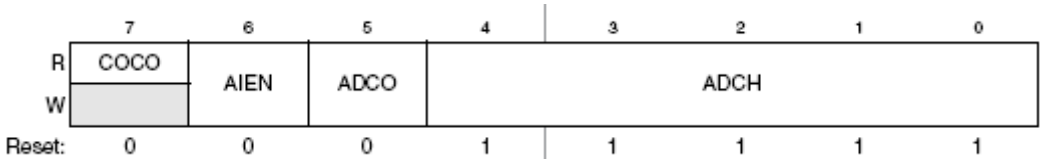
- Algoritmo de aproximação linear sucessiva com resolução de 10 bits
- até 28 entradas analógicas
- Saída formatada em 10 ou 8 bits com formatado justificado a direita
- Conversão simples ou contínua
- Tempo de amostragem configurável
- Configuração para velocidade ou baixo consumo de energia
- Interrupção associada ao final de uma conversão
- Fonte de clock assíncrono para operação com baixo nível de ruído
- Registrador de comparação com interrupção associada

Na figura abaixo o diagrama de blocos do conversor A/D é apresentado:



A seguir os principais registradores do módulo A/D serão apresentados.

Registrador 1 de Controle e Estado do Módulo A/D ***(ADC1SC1)***



COCO (Conversion Complete *Flag*) - Este bit de leitura é levado para 1 sempre que uma conversão é completada quando a função de comparação esta desabilitada (ACFE = 0). Quando a função de comparação esta habilitada (ACFE = 1), a flag COCO é setada ao fim de uma conversão somente se o resultado da comparação for verdadeiro. Este bit é limpo quando o registrador ADC1SC1 é escrito ou quando o registrador ADC1RL é lido.

1 – Conversão completa

0 – Conversão não completa

AIEN (*Interrupt Enable*)

1 – Interrupção de conversão completa habilitada

0 – Interrupção de conversão completa desabilitada

ADCO (*Continuos Conversion Enable*) – Este bit é utilizado para habilitar o modo de conversão contínua.

1 – Conversão contínua inicializada a partir de uma escrita no registrador ADC1SC1 quando operando por gatilho de *software*. Conversão contínua inicializada por um evento de ADHWT quando operando por gatilho de *hardware*.

0 – Ativa uma conversão após uma escrita no registrador ADC1SC1 quando operando por gatilho de *software*. Quando operando por gatilho de *hardware*, ativa uma conversão a cada evento de ADHWT.

ADCH (*Input Channel Select*) – É um campo de 5 bits que é utilizado para selecionar o canal de entrada do conversor A/D. Quanto todos os bits são colocados em 1, o sistema de conversão por aproximações sucessivas é desligado. Abaixo é apresentada a tabela de seleção de canais do conversor.

ADCH	Input Select
00000	AD0
00001	AD1
00010	AD2
00011	AD3
00100	AD4
00101	AD5
00110	AD6
00111	AD7

ADCH	Input Select
10000	AD16
10001	AD17
10010	AD18
10011	AD19
10100	AD20
10101	AD21
10110	AD22
10111	AD23

ADCH	Input Select
01000	AD8
01001	AD9
01010	AD10
01011	AD11
01100	AD12
01101	AD13
01110	AD14
01111	AD15

ADCH	Input Select
11000	AD24
11001	AD25
11010	AD26
11011	AD27
11100	Reserved
11101	V_{REFH}
11110	V_{REFL}
11111	Module disabled

Registrador 2 de Controle e Estado do Módulo A/D ***(ADC1SC2)***

	7	6	5	4	3	2	1	0
R	ADACT	ADTRG	ACFE	ACFGT	0	0	R[1]	R[1]
W								
Reset:	0	0	0	0	0	0	0	0

ADACT (*Conversion Active*) - Este bit indica uma conversão em progresso. Este bit é levado para 1 quando uma conversão é iniciada e limpo quando a conversão é completada ou abortada.

1 – Conversão em progresso

0 – Nenhuma conversão em progresso

ADTRG (*Conversion Trigger Select*) - Este bit é utilizado para selecionar o tipo de gatilho das conversões.

1 – Gatilho por *hardware* selecionado

0 – Gatilho por *software* selecionado

ACFE (*Compare Function Enable*) - Este bit é utilizado para habilitar a função de comparação.

1 – Função de comparação habilitada

0 – Função de comparação desabilitada

ACFGT (*Compare Function Greater Than Enable*) – Este bit configura a função para disparar o gatilho de comparação.

1 – Gatilho de comparação dispara quando a entrada é maior ou igual ao nível de comparação

0 – Gatilho de comparação dispara quando a entrada é menor do que o nível de comparação

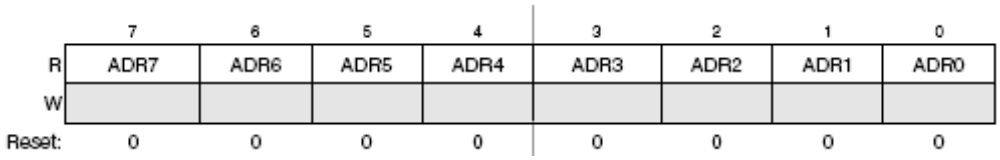
Registrador de Resultado de Dados Alto do Módulo A/D ***(ADC1RH)***

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	ADR9	ADR8
W								
Reset:	0	0	0	0	0	0	0	0

O registrador ADC1RH contém os dois bits mais significativos do resultado de uma conversão de 10 bits. Quando configurado para conversão de 8 bits, ambos ADR8 e ADR9 serão iguais a zero. Este registrador é atualizado sempre que uma conversão é completada.

No modo de conversão em 10 bits, devemos ler o registrador ADC1RH antes do registrador ADC1RL. Ainda neste modo, após a leitura do registrador ADC1RH devemos obrigatoriamente ler o registrador ADC1RL, pois caso esta leitura não seja realizada, o resultado das conversões subseqüentes será perdido.

Registrador de Resultado de Dados Baixo do Módulo A/D ***(ADC1RL)***

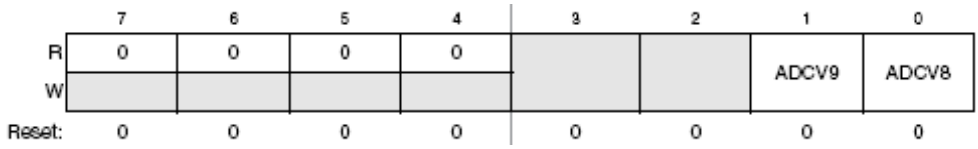


O registrador ADC1RL contém os oito bits menos significativos do resultado de uma conversão de 10 bits e todos os oito bits de uma conversão de 8 bits. Este registrador é atualizado sempre que uma conversão é completada, exceto quando a comparação automática está habilitada e a condição de comparação não é satisfeita.

Registradores de Valor de Comparação do Módulo A/D Alto e Baixo ***(ADC1CVH e ADC1CVL)***

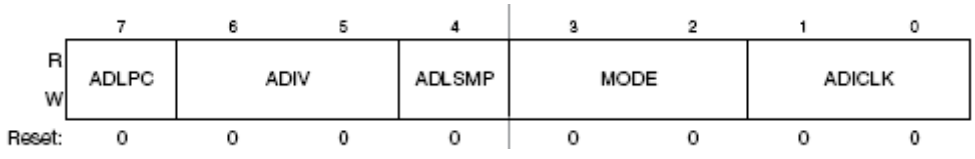
Estes registradores contém a parte alta e baixa do valor de 10 bits utilizado como valor de referência para a função de comparação do módulo A/D.

ADC1CVH



ADC1CVL

Registrador de Configuração do Módulo A/D ***(ADC1CFG)***



Este registrador é utilizado para selecionar o modo de operação , a fonte de *clock*, o fator de divisão do *clock* e a configuração de consumo de energia e tempo de amostragem do módulo A/D.

ADLPC (*Low Power Configuration*) – Este registrador é utilizado para otimizar o consumo de energia do módulo A/D quando altas taxas de amostragem não são requiridas.

1 – Configuração de baixo consumo de energia

0 – Configuração para alta velocidade de conversão.

ADIV (*Clock Divide Select*) – Estes bits selecionam a taxa de divisão utilizada pelo A/D para gerar o clock interno ADCK. A tabela abaixo apresenta as possíveis configurações.

ADIV	Divide Ratio	Clock Rate
00	1	Input clock
01	2	Input clock ÷ 2
10	4	Input clock ÷ 4
11	8	Input clock ÷ 8

ADLSMP (*Long Sample Time Configuration*) – Este bit seleciona entre tempos de amostragem longos ou curtos. Esta configuração ajusta o período de amostragem para permitir que entradas de alta impedância sejam amostradas de forma mais exata ou para maximizar a velocidade de conversão para entradas de baixa impedância.

1 – Tempo de amostragem longo

0 – Tempo de amostragem curto

MODE (*Conversion Mode Selection*) – Estes bits são utilizados para selecionar entre o modo de operação de 8 ou 10 bits, de acordo com a tabela abaixo.

MODE	Mode Description
00	8-bit conversion (N=8)
01	Reserved
10	10-bit conversion (N=10)
11	Reserved

ADICLK (*Input Clock Select*) – Estes bits selecionam a fonte de entrada de *clock* para gerar o *clock* interno do módulo A/D, conforme tabela abaixo:

ADICLK	Selected Clock Source
00	Bus clock
01	Bus clock divided by 2
10	Alternate clock (ALTCLK)
11	Asynchronous clock (ADACK)

Registradores de Controle de Pinos 1, 2 e 3 do Módulo A/D ***(APCTL1, APCTL2 e APCTL3)***

Estes registradores são utilizados para habilitar ou desabilitar o controle de entrada / saída dos pinos do microcontrolador associados aos canais do módulo conversor A/D quando utilizados como entradas analógicas. Quando o bit associado ao controle de um pino é definido em 1, as seguintes condições são forçadas ao pino do microcontrolador:

- O *buffer* de saída é forçado para um estado de alta impedância
- O *buffer* de entrada é desabilitado. Uma leitura da porta irá retornar zero para qualquer pino em que o *buffer* de entrada estiver desabilitado.
- O *pull-up* será desabilitado.

APCTL1

	7	6	5	4	3	2	1	0
R W	ADPC7	ADPC6	ADPC5	ADPC4	ADPC3	ADPC2	ADPC1	ADPC0
Reset:	0	0	0	0	0	0	0	0

APCTL2

	7	6	5	4	3	2	1	0
R W	ADPC15	ADPC14	ADPC13	ADPC12	ADPC11	ADPC10	ADPC9	ADPC8
Reset:	0	0	0	0	0	0	0	0

APCTL3

	7	6	5	4	3	2	1	0
R W	ADPC23	ADPC22	ADPC21	ADPC20	ADPC19	ADPC18	ADPC17	ADPC16
Reset:	0	0	0	0	0	0	0	0

Descrição Funcional do Módulo A/D

O módulo A/D é desabilitado durante o reset ou quando os bits ADCH estiverem todos em nível lógico alto. O módulo entra em estado de espera quando uma conversão é completada e outra conversão não é iniciada. Quando em estado de espera, o módulo está em seu estado de menor consumo de energia.

O A/D pode realizar uma conversão analógica / digital em qualquer um dos canais selecionados por *software*. A tensão elétrica do canal selecionado será convertida por um algoritmo de aproximações sucessivas em um resultado digital de 11 bits. No modo de 8 bits, a a tensão do canal selecionado é convertida pelo algoritmo de aproximações sucessivas em um resultado digital de 9 bits.

Quando a conversão é completada, o resultado é colocado nos registradores de dados (ADC1RH e ADC1RL). No modo de 10 bits o resultado é arredondado para 10 bits e colocado nos registradores ADC1RH e ADC1RL. No modo de 8 bits o resultado é arredondado para 8 bits e colocado no registrador ADC1RL.

A *flag* de conversão completa (COCO) é setada quando a conversão é completada e uma interrupção é gerada se o bit de interrupção por conversão completa estiver habilitado (AIEN = 1).

O módulo A/D tem a capacidade de comparar automaticamente o resultado da conversão com o conteúdo dos registradores de comparação. A função de comparação é habilitada setando o bit ACFE e opera em conjunto com qualquer modo e configuração de conversão.

Tempo total de Conversão

O tempo total de conversão depende do tempo de amostragem (determinado pelo bit ADLSMP), da freqüência de barramento do microcontrolador, do modo de conversão (8 ou 10 bits) e da freqüência do clock de conversão (f_{ADCK}). Depois que o módulo torna-se ativo, a amostragem das entradas começa. O bit ADLSMP é utilizado para selecionar entre tempos de amostragem longos e curtos. Quando a amostragem é completada, o conversor é isolado do canal de entrada e o algoritmo de aproximações sucessivas é executado para determinar o valor digital do sinal analógico.

O resultado da conversão é transferido para os registradores ADC1RH e ADC1RL assim que o algoritmo de conversão é completado. Se a freqüência do barramento for menor que a freqüência f_{ADCK}, tempos de amostragem precisos para conversões contínuas

não podem ser garantidas quando períodos de amostragem curtos estão habilitados (ADLSMP = 0). Se a freqüência de barramento for $1/11$ vezes menor que a freqüência f_{ADCK}, tempos de amostragem precisos para conversões contínuas não podem ser garantidos quando períodos de amostragem longos estão habilitados (ADLSMP = 1).

O tempo de conversão máximo para diferentes condições pode ser visualizado na tabela abaixo:

Conversion Type	ADICLK	ADLSMP	Max Total Conversion Time
Single or first continuous 8-bit	0x, 10	0	20 ADCK cycles + 5 bus clock cycles
Single or first continuous 10-bit	0x, 10	0	23 ADCK cycles + 5 bus clock cycles
Single or first continuous 8-bit	0x, 10	1	40 ADCK cycles + 5 bus clock cycles
Single or first continuous 10-bit	0x, 10	1	43 ADCK cycles + 5 bus clock cycles
Single or first continuous 8-bit	11	0	5 µs + 20 ADCK + 5 bus clock cycles
Single or first continuous 10-bit	11	0	5 µs + 23 ADCK + 5 bus clock cycles
Single or first continuous 8-bit	11	1	5 µs + 40 ADCK + 5 bus clock cycles
Single or first continuous 10-bit	11	1	5 µs + 43 ADCK + 5 bus clock cycles
Subsequent continuous 8-bit; $f_{BUS} \geq f_{ADCK}$	xx	0	17 ADCK cycles
Subsequent continuous 10-bit; $f_{BUS} \geq f_{ADCK}$	xx	0	20 ADCK cycles
Subsequent continuous 8-bit; $f_{BUS} \geq f_{ADCK}/11$	xx	1	37 ADCK cycles
Subsequent continuous 10-bit; $f_{BUS} \geq f_{ADCK}/11$	xx	1	40 ADCK cycles

O tempo total máximo de conversão é determinado pela fonte de *clock* escolhida e pelo fator de divisão selecionado. A fonte de *clock* é selecionada pelos bits ADICLK e o fator de divisão é especificado pelos bits ADIV.

Por exemplo, no modo de conversão em 10 bits, com *clock* de barramento selecionado como fonte de entrada de *clock*, entrada de *clock* dividida por 1 e uma freqüência de barramento de 20 Mhz, o tempo de conversão total para uma conversão simples é:

$$Tempo\,total = (23\,ciclos\ ADCK) * (20\,Mhz/1) + (5\,ciclos\,de\,barramento) * (20\,Mhz/1) = 1.4\,\mu s$$

Número total de ciclo de barramento = 28 ciclos.

OBS.: A freqüência ADCK deve estar entre a f_{ADCK} mínima e a f_{ADCK} máxima para atingir as especificações do módulo A/D.

A configuração do módulo A/D para o exemplo acima é apresentada a seguir.

```
/* ADC1SC2: ADACT=0,ADTRG=0,ACFE=0,ACFGT=0,??=0,??=0,??=0,??=0 */
ADC1SC2 = 0;        /* Desabilita o gatilho por hardware e a auto comparação */

// Seleciona clock do barramento como entrada, fator de divisão por 1
// Configura modo de conversão para 10 bits
/* ADLPC=0,ADIV1=1,ADIV0=0,ADLSMP=0,MODE1=1,MODE0=0,ADICLK1=0,ADICLK0=0 */
ADC1CFG = 0x48;

/* ADC1SC1: COCO=0,AIEN=0,ADCO=0,ADCH4=1,ADCH3=1,ADCH2=1,ADCH1=1,ADCH0=1 */
ADC1SC1 = 0x1F;     /* Desliga o módulo */
```

Abaixo é apresentado um exemplo de função para adquirir o valor digital de um canal específico:

```
word converte(byte canal)
{
 word retorno=0;
 ADC1SC1 = canal;                  // Inicia a conversão do canal
 while (!ADC1SC1_COCO);            // Espera a conversão completar
 retorno = ADC1R;                  // Adquire o valor da conversão
 ADC1SC1 = 0x1F;                   // Desliga o conversor A/D
 return retorno;                   // retorna o valor digital adquirido
}
```

Características do Módulo A/D

Abaixo são apresentadas as principais características do módulo A/D interno da série de microcontroladores MC9S08AW.

Characteristic	Conditions	Symb	Min	Typ[1]	Max	Unit
Supply voltage	Absolute	V_{DDAD}	2.7	—	5.5	V
	Delta to V_{DD} (V_{DD}–V_{DDAD})[2]	ΔV_{DDAD}	–100	0	+100	mV
Ground voltage	Delta to V_{SS} (V_{SS}–V_{SSAD})[2]	ΔV_{SSAD}	–100	0	+100	mV
Ref voltage high		V_{REFH}	2.7	V_{DDAD}	V_{DDAD}	V
Ref voltage low		V_{REFL}	V_{SSAD}	V_{SSAD}	V_{SSAD}	V
Input voltage		V_{ADIN}	V_{REFL}	—	V_{REFH}	V
Input capacitance		C_{ADIN}	—	4.5	5.5	pF
Input resistance		R_{ADIN}	—	3	5	kΩ
Analog source resistance External to MCU	10-bit mode f_{ADCK} > 4MHz f_{ADCK} < 4MHz	R_{AS}	— —	— —	5 10	kΩ
	8-bit mode (all valid f_{ADCK})		—	—	10	
ADC conversion clock frequency	High speed (ADLPC = 0)	f_{ADCK}	0.4	—	8.0	MHz
	Low power (ADLPC = 1)		0.4	—	4.0	

Characteristic	Conditions	C	Symb	Min	Typ[1]	Max	Unit
ADC asynchronous clock source $t_{ADACK} = 1/f_{ADACK}$	High speed (ADLPC = 0)	P	f_{ADACK}	2	3.3	5	MHzS
	Low power (ADLPC = 1)			1.25	2	3.3	
Conversion time (Including sample time)	Short sample (ADLSMP = 0)	P	t_{ADC}	—	20	—	ADCK cycles
	Long sample (ADLSMP = 1)			—	40	—	
Sample time	Short sample (ADLSMP = 0)	P	t_{ADS}	—	3.5	—	ADCK cycles
	Long sample (ADLSMP = 1)			—	23.5	—	
Total unadjusted error Includes quantization	10-bit mode	P	E_{TUE}	—	±1	±2.5	LSB[2]
	8-bit mode			—	±0.5	±1.0	
Differential non-linearity	10-bit mode	P	DNL	—	±0.5	±1.0	LSB[2]
	8-bit mode			—	±0.3	±0.5	

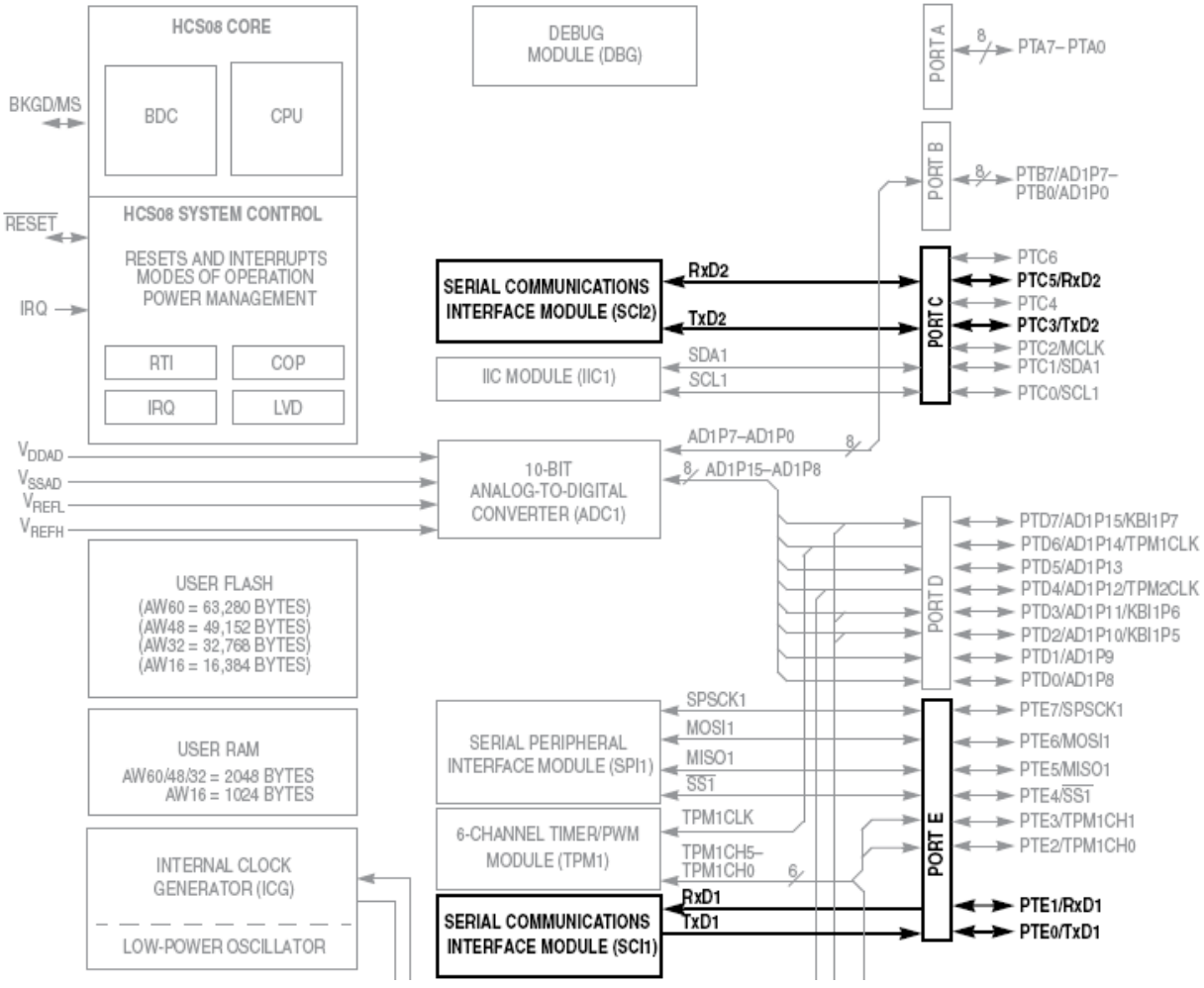
6. Configuração da Porta Serial SCI

A série de microcontroladores MC9S08AW inclui duas interfaces seriais de comunicação (SCI) independentes, que também são conhecidas por Transmissor / Receptor Universal Assíncrono (*universal asynchronous receiver/transmitters* – UARTs). Tipicamente, estes sistemas são utilizados para conectar portas de entrada / saída RS232 de um computador de uso pessoal (PC) ou workstations, mas também podem ser utilizados para comunicar com outros dispositivos embarcados.

Este sistema possui um módulo gerador de *baud rate* flexível de 13 bits, que suporta uma grande faixa de padrões de *baud rates* até a taxa máxima de 115,2 kbaud. A transmissão e a recepção dentro de um mesmo módulo SCI utiliza o mesmo *baud rate* e cada módulo SCI possui um gerador de *baud rate* separado.

O sistema SCI desta série de microcontroladores inclui detecção de ruídos, paridade por *hardware*, duplo *buffer* para transmissão e recepção e saída do modo de espera por interrupção de recepção.

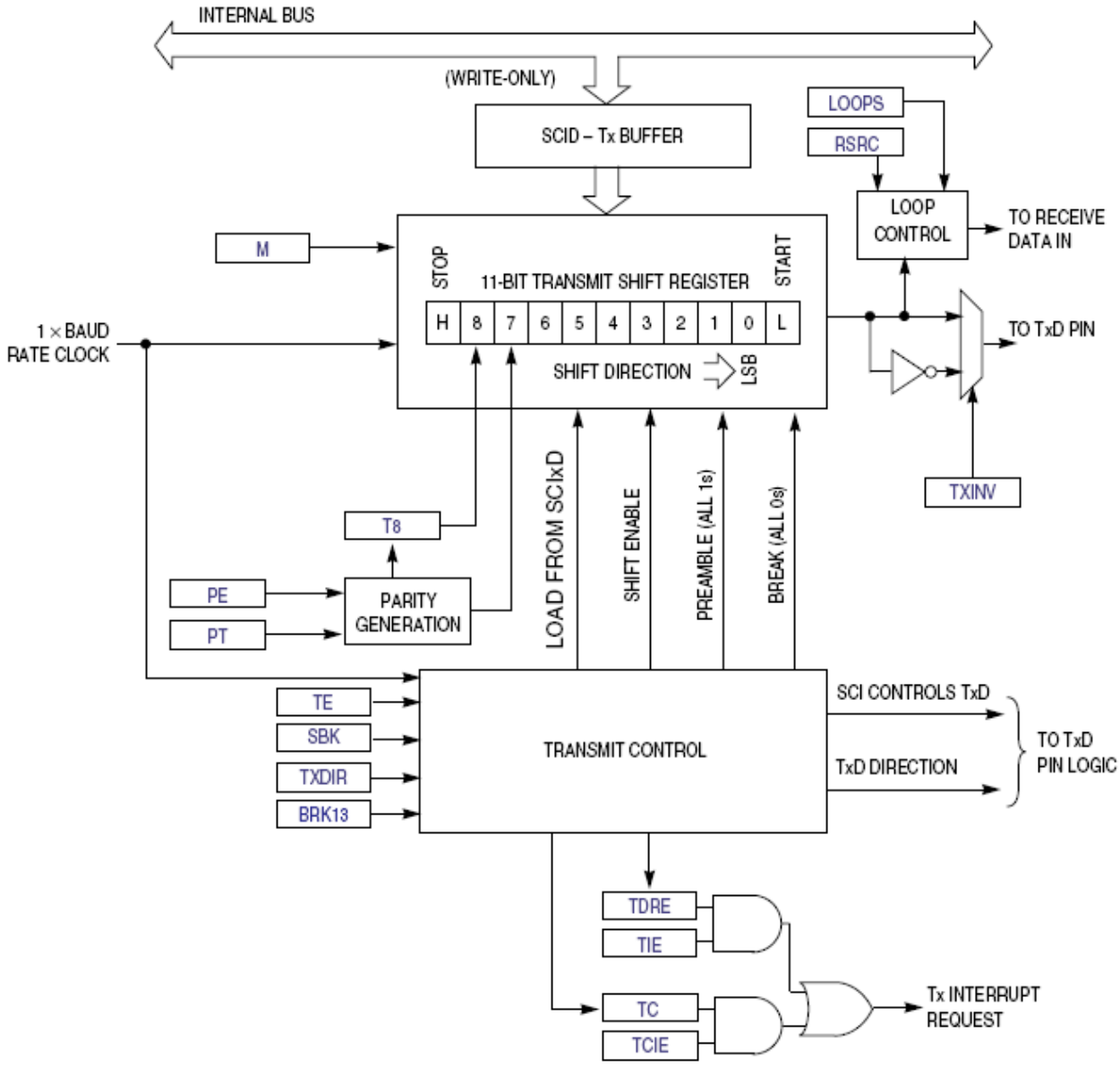
A figura abaixo apresenta o diagrama de blocos do microcontrolador, demonstrando as conexões dos módulos SCI.



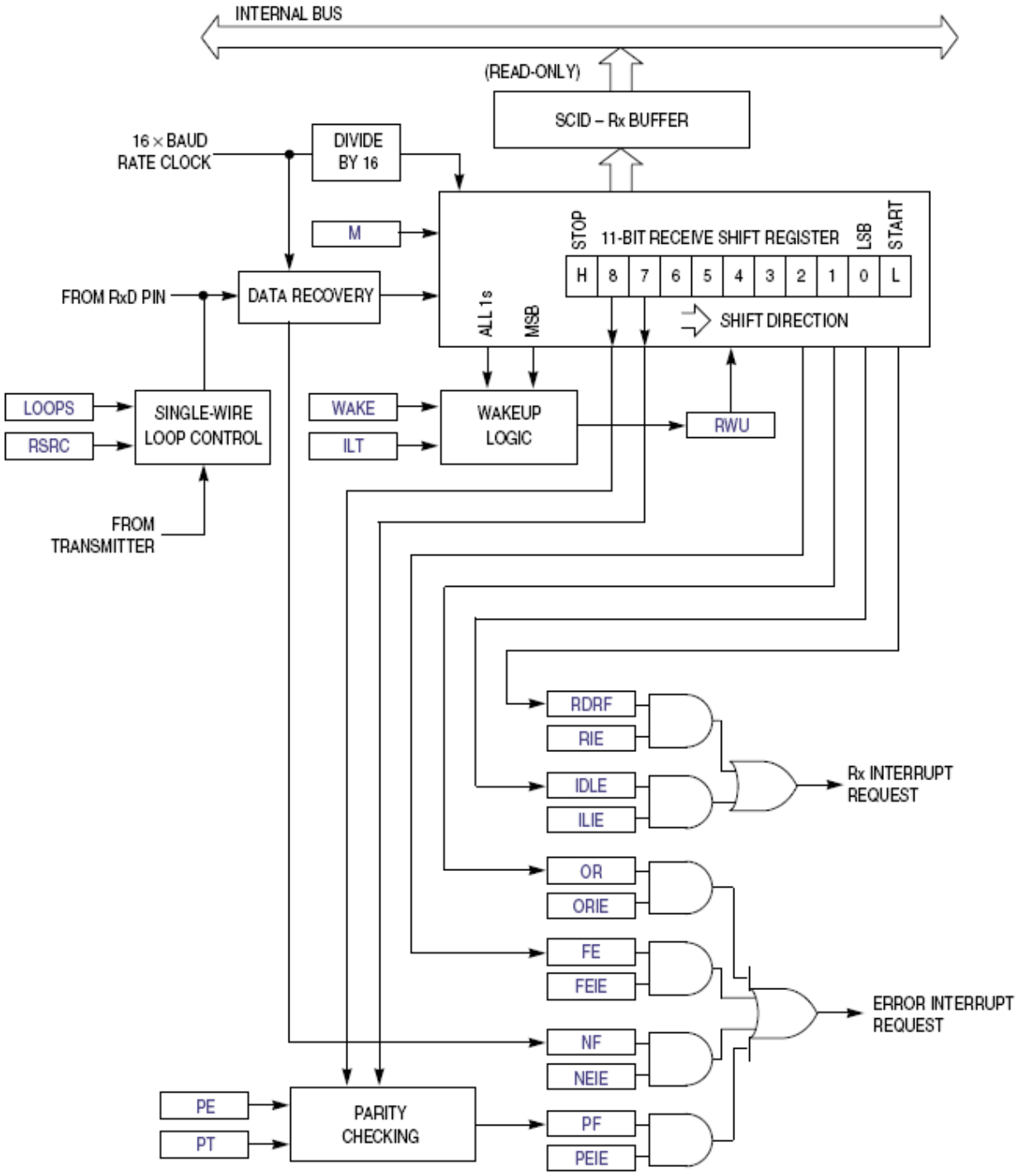
Características dos Módulos SCI:

- Formato padrão sem retorno para zero (NRZ), *full-duplex*
- *Buffer* duplo para transmissão e recepção com *enables* separados.
- Baud Rate programável
- Interrupções para os seguintes eventos:
  - Transmissão completa e Registrador de dados de transmissão vazio
  - Registrador de dados de recepção cheio
  - Erro de paridade, erro de enquadramento, erro por ruído e por recepção excedente
  - Detecção de estado de espera
- Geração e verificação de paridade por *hardware*
- Tamanho de caractere programável em 8 bits ou 9 bits
- Seleção de polaridade de saída da transmissão

A figura abaixo apresenta o diagrama de blocos da transmissão SCI.



E a seguir é apresentado o diagrama de blocos da recepção SCI.



O módulo SCI possui 8 registradores de 8 bits para controlar o *baud rate*, selecionar as opções do módulo, retornar condições de funcionamento e para dados de transmissão e recepção.

A seguir serão apresentados estes registradores com suas descrições e funcionalidades.

Registradores para Seleção de Baud Rate SCI ***(SCIxBDH, SCIxBHL)***

Este par de registradores controla o fator de divisão para a geração do baud rate SCI. Para atualizar a configuração dos 13 bits de *baud rate* (SBR12:SBR0), primeiro devemos escrever no registrador mais significativo (SCIxBDH) e depois no registrador menos significativo (SCIxBDL). O valor do registrador mais significativo não é alterado até que o menos significativo seja alterado. Após um *reset* a geração de *baud rate* é desabilita até que o transmissor ou receptor seja ativado pela primeira vez.

SCIxBDH

	7	6	5	4	3	2	1	0
R	0	0	0	SBR12	SBR11	SBR10	SBR9	SBR8
W								
Reset	0	0	0	0	0	0	0	0

SCIxBDL

	7	6	5	4	3	2	1	0
R	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
W								
Reset	0	0	0	0	0	1	0	0

Estes 13 bits são utilizados como um fator de divisão do módulo de *baud rate*, e serão chamados de BR. O módulo gerador de *baud rate* será desabilitado quando BR = 0 para reduzir o consumo de energia. O valor de BR pode variar de 1 a 8191. O *baud rate* gerado é dado pela equação abaixo:

$$baud\ rate = \frac{(Clock\ de\ Barramento)}{(16\,x\,BR)}$$

ou seja,

$$BR = \frac{(Clock\ de\ Barramento)}{(16\,x\,baud\ rate)}$$

OBS: Dificilmente iremos conseguir *baud rates* exatos. Devemos lembrar que existe uma tolerância de até 3,5% no *baud rate* gerado. Com o intuito de minimizar os erros, é interessante evitar erros maiores que 2%.

*Registrador 1 de Controle SCI **(SCIxC1)***

	7	6	5	4	3	2	1	0
R W	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT
Reset	0	0	0	0	0	0	0	0

LOOPS (*Loop Mode Select*) – Selecionada entre o modo de operação em *loop* ou modo de operação normal, ou seja, modo *full-duplex* com 2 pinos.

0 – Operação normal, RxD e TxD usam pinos separados

1 – No modo *loop* a saída do transmissor é internamente conectada a entrada do receptor

SCISWAI (*SCI Stops in Wait Mode*)

0 – O módulo SCI continua operando durante um estado de espera e pode ser fonte de uma interrupção que acorda o microcontrolador.

1 – O módulo SCI para de operar quando o microcontrolador entra em estado de espera

RSRC (*Receiver Source Select*) – Este bit não tem efeito a menos que o bit LOOPS esteja definido em 1. Quando LOOPS = 1, a entrada do receptor é internamente conectada ao pino TxD e RSRC determina quando esta conexão é também ligada a saída do transmissor.

0 – Se LOOPS = 1, seleciona modo de loop interno e a SCI não utiliza os pinos RxD e TxD

1 – Modo de transmissão a um fio, onde TxD é conectado a saída do transmissor e a entrada do receptor.

M (*9 bits or 8 bits Mode Select*)

0 – Operação Normal – *Start* + 8 bits de dados (menos significativo primeiro) + *stop*

1 – Modo com caractere em 9 bits – *Start* + 8 bits de dados + 9º bit de dados + *stop*

WAKE (*Receiver Wakeup Method Select*)

0 – A condição para retirar o receptor de um estado de espera será a detecção de um estado de repouso na linha entre mensagens.

1 – A condição para retirar o receptor de um estado de espera será um nível lógico 1 no bit de dados mais significativo em um caractere.

ILT (*Idle Line Type Select*) – Configura o modo que uma linha em estado de repouso será detectada.

0 – A contagem dos bits para determinar um estado de repouso inicia depois do bit de *start*. Desta forma o bit de *stop* e qualquer outro bit com nível lógico 1 no final de um caractere conta para o tempo de um caractere completo do repouso.

1 – A contagem não inicia até que o tempo de um bit de *stop* ocorra. Desta forma a detecção do estado de repouso não é afetada pelos dados no último caractere da mensagem anterior.

PE (*Parity Enable*) – Habilita a geração e verificação de paridade por *hardware*. Quando a paridade é habilitada, o bit mais significativo do caractere de dados (oitavo ou nono bit de dados) é tratado como bit de paridade.

0 – Sem geração ou verificação de paridade por *hardware*

1 – Paridade habilitada

PT (*Parity Type*) – Este bit determina se a paridade utilizada será par ou impar. Paridade impar significa que o número total de 1s no caractere de dados, incluindo o bit de paridade, será impar. Paridade par significa que o número total de 1s no caractere de dados, incluindo a paridade, será par.

0 – Paridade Par

1 – Paridade Impar

*Registrador 2 de Controle SCI **(SCIxC2)***

	7	6	5	4	3	2	1	0
R W	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
Reset	0	0	0	0	0	0	0	0

TIE *(Transmit Interrupt Enable – for TDRE)*

0 – Interrupção por *hardware* para a flag TDRE desabilita (utilizar *polling*)

1 – Interrupção por *hardware* requisitada quando a flag TDRE for para nível lógico 1

TCIE *(Transmission Complete Interrupt Enable – for TC)*

0 – Interrupção por *hardware* para a flag TC desabilita (utilizar polling)

1 – Interrupção por *hardware* requisitada quando a flag TC for para nível lógico 1

RIE *(Receiver Interrupt Enable – for RDRF)*

0 – Interrupção por *hardware* para a flag RDRF desabilita (utilizar polling)

1 – Interrupção por *hardware* requisitada quando a flag RDRF for para nível lógico 1

ILIE *(Idle Line Interrupt Enable)*

0 – Interrupção por *hardware* para a flag IDLE desabilita (utilizar polling)

1 – Interrupção por *hardware* requisitada quando a flag IDLE for para nível lógico 1

TE *(Transmitter Enable)* – Quando a transmissão SCI está desligada o pino TxD passa a ser um pino de entrada / saída de propósito geral. Quando o bit TE é escrito para 0, a transmissão continua controlando o pino TxD até que todos os dados agendados para serem transmitidos terminem antes de permitir que o pino se reverta em um pino de entrada / saída de propósito geral.

0 – Transmissão desligada

1 – Transmissão ligada

RE *(Receiver Enable)* – Quando a recepção SCI está desligada o pino RxD passa a ser um pino de entrada / saída de propósito geral.

0 – Recepção Desligada

1 – Recepção ligada

RWU (*Receiver Wakeup Control)* – Este bit pode ser escrito para 1 para colocar a recepção em estado de espera, onde o receptor espera pela detecção automática de *hardware* de uma condição para acordar selecionada. A condição para acordar é definida pelo bit de controle WAKE.

0 – Operação normal de recepção SCI

1 – Receptor SCI em *standby* esperando por uma condição para acordar.

SBK *(Send Break)* - Escrevendo 1 e depois um 0 para SBK agenda um caractere de parada no fluxo de dados da transmissão. Caracteres de parada adicionais são agendados enquanto o bit SBK estiver em nível lógico alto.

0 – Operação de transmissão normal

1 – Agenda um caractere de parada para ser enviado

Registrador 1 de Estado do SCI ***(SCIxS1)***

Este registrador possui 8 bits somente de leitura operando como *flags* de estado. Escrever nestes bits não tem efeito. Seqüências especiais de *software* (que não envolvem escrever neste registrador) são usadas para limpar o estado destas *flags*.

	7	6	5	4	3	2	1	0
R	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
W								
Reset	1	1	0	0	0	0	0	0

TDRE (*Transmit Data Register Empty Flag*) – O bit TDRE é levado para nível lógico 1 no reset e quando um dado é transferido do *buffer* de transmissão de dados para o *transmit shifter*, deixando espaço para um novo caractere no *buffer*. Para limpar o bit TDRE deve-se ler o registrador SCIxS1 com TDRE = 1 e então escrever para o registrador de dados do módulo SCI (SCIxD).

0 – Registrador de dados de transmissão (*buffer*) cheio

1 – Registrador de dados de transmissão (buffer) vazio

TC (*Transmission Complete Flag*) – O bit TC é levado para nível lógico 1 no reset e quando TDRE = 1 sem dados, preâmbulo ou caractere de parada sendo transmitido. O bit TC é levado para nível lógico 0 através da leitura do registrador SCIxS1 com TC = 1 e então realizando uma destas três possibilidades: escrever no registrador SCIxD para transmitir novo dado, agendar um preâmbulo alterando TE de 0 para 1 ou agendando um caractere de parada escrevendo 1 no bit SBK no registrador SCIxC2.

0 – Transmissor ativo (enviando dados, um preâmbulo ou um caractere de parada)

1 – Transmissor em repouso

RDRF (*Receive Data Register Full Flag*) – O bit RDRF é levado para nível lógico 1 quando um caractere é transferido do receive shifter para o registrador de dados de recepção (SCIxD). No modo de 8 bits, para limpar o bit RDRF, devemos ler o registrador SCIxS1 com RDRF = 1 e então ler o registrador de dados (SCIxD). No modo de 9 bits, para limpar o bit RDRF, devemos ler o registrador SCIxS1 com RDRF = 1 e então ler os registradores SCIxD e SCIxC3. Os registradores SCIxD e SCIxC3 podem ser lidos em qualquer ordem, mas a flag só será levada para nível lógico 0 quando ambos os registradores forem lidos.

0 – Registrador de dados de recepção vazio

1 - Registrador de dados de recepção cheio

IDLE (*Idle Line Flag*) – O bit IDLE é levado para nível lógico 1 quando a linha de recepção SCI passa para um estado de repouso por um período equivalente ao tempo de um caractere completo depois do período de atividade. Para levar o bit IDLE para nível lógico 0 devemos ler o registrador SCIxS1 com IDLE = 1 e então ler o registrador de dados (SCIxD). Depois de IDLE tornar-se 0, ele não voltará a ser 1 até que um novo caractere seja recebido e o bit RDRF torne-se 1.

0 – Linha em estado de repouso não detectada

1 – Linha em estado de repouso detectada

OR (*Receiver Underrun Flag*) – O bit OR é levado para nível lógico 1 quando um novo caractere está pronto para ser transferido para o registrador de dados de recepção (*buffer*), mas o caractere recebido anteriormente ainda não foi lido ainda no registrador SCIxD. Neste caso, o novo caractere (e toda informação de erro associada) é perdido porque não há local para mover este para o registrador SCIxD. Para limpar o bit OR devemos ler o registrador SCIxS1 com OR = 1 e então ler o registrador de dados (SCIxD).

0 – Não ocorreu *overrun*

1 – Ocorreu overrun na recepção (o novo dado SCI foi perdido)

NF (*Noise Flag*) – A técnica de amostragem utilizada no receptor adquire sete amostras durante o bit de *start* e três amostras em cada bit de dados e *stop* recebidos. Se qualquer uma destas amostras for diferente do resto das amostras dentro do período de um bit em um quadro, a *flag* NF irá tornar-se 1 ao mesmo tempo que a *flag* RDRF passar para nível lógico 1 para este caractere. Para limpar esta *flag* devemos ler o registrador SCIxS1 e então ler o registrador de dados SCI (SCIxD).

0 – Nenhum ruído detectado

1 – Ruído detectado na recepção de um caractere em SCIxD.

FE (*Framing Error Flag*) – O bit FE é levado para nível lógico 1 ao mesmo tempo em que a *flag* RDRF quando o receptor detectar um nível lógico 0 onde um bit de *stop* for esperado. Isto sugere que o receptor não foi corretamente alinhado ao enquadramento do caractere. Para limpar o bit FE devemos ler o registrador SCIxS1 e então ler o registrador de dados (SCIxD).

0 – Nenhum erro de enquadramento detectado. Isto não garante que o caractere está correto.

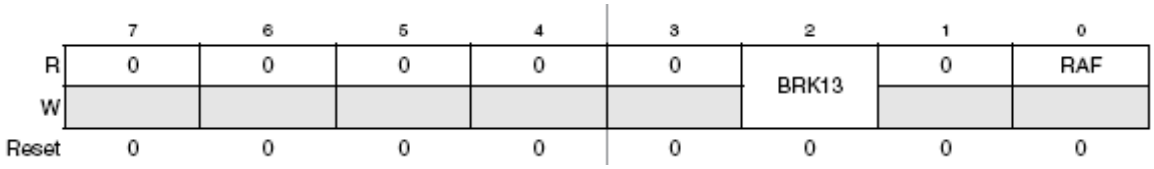
1 – Erro de enquadramento

PF (*Parity Error Flag*) – O bit PF é levado para nível lógico 1 ao mesmo tempo que a *flag* RDRF quando a paridade está habilitada (PE = 1) e o bit de paridade no caractere recebido não estiver de acordo com o valor de paridade esperado. Para limpar esta flag devemos ler o registrador SCIxS1 e então ler o registrador de dados (SCIxD).

0 – Sem erro de paridade

1 – Erro de paridade detectado

*Registrador 2 de Estado do SCI **(SCIxS2)***



BRK13 (*Break Character Lenght*) – O bit BRK13 é utilizado para selecionar um comprimento de caractere de parada mais longo. A detecção de erros de enquadramento não é afetada pelo estado deste bit.

0 – O caractere de parada terá comprimento de 10 bits (11 se M = 1)

1 – O caractere de parada terá comprimento de 13 bits (14 se M = 1)

RAF (*Receiver Active Flag*) – O bit RAF é levado para nível lógico 1 quando o receptor SCI detecta o início de um bit de *start* válido. Este bit é levado para nível lógico 0 automaticamente quando o receptor detecta a linha em estado de repouso. Esta *flag* de estado pode ser utilizada para verificar quando um caractere SCI está sendo recebido antes de colocar o microcontrolador no modo de *stop*.

0 – Receptor SCI em repouso, esperando um bit de *start*

1 – Receptor SCI ativo (entrada RxD não está em repouso)

*Registrador 3 de Controle SCI **(SCIxC3)***

	7	6	5	4	3	2	1	0
R	R8	T8	TXDIR	TXINV	ORIE	NEIE	FEIE	PEIE
W								
Reset	0	0	0	0	0	0	0	0

R8 (*Ninth Data Bit for Receiver*) – Quando o módulo SCI é configurado para 9 bits de dados (M = 1), R8 pode ser visto como o nono bit de dados recebido a esquerda do bit mais significativo do *buffer* de dados no registrador SCIxD. Quando lendo dados de 9 bits, tanto o bit R8 quanto o registrador SCIxD devem ser lidos para completar a seqüência automática de limpeza da flag RDRF.

T8 (*Ninth Data Bit for Transmitter*) - Quando o módulo SCI é configurado para 9 bits de dados (M = 1), T8 pode ser visto como o nono bit de dados a esquerda do bit mais significativo do *buffer* de dados no registrador SCIxD. Quando escrevendo dados de 9 bits, o valor de 9 bits inteiro é transferido para o *shift register* depois que o registrador SCIxD é escrito. Então o bit T8 deve ser escrito (se ele precisa ser alterado de seu último valor) antes do registrador SCIxD ser escrito. Se o bit T8 não precisar ser alterado (por exemplo, quando for utilizado para gerar paridade par ou impar), este não necessitará ser escrito a cada vez que o registrador SCIxD for escrito.

TXDIR (*TxD Pin Direction in Single-Wire Mode*) – Quando o módulo SCI é configurado para operação half-duplex (LOOPS = RSRC = 1), este bit determina a direção dos dados no pino TxD.

0 – O pino TxD é um entrada no modo com 1 fio

1 – O pino TxD é um saída no modo com 1 fio

TXINV (*Transmit Data Inversion*) – Colocando este bit em nível lógico alto reverte a polaridade da saída dos dados transmitidos.

0 – Dados transmitidos não invertidos

1 – Dados transmitidos invertidos

ORIE (*Overrun Interrupt Enable*) – Este bit habilita a *flag* de *overrun* (OR) gerar uma requisição de interrupção de *hardware*.

0 – Interrupção pela *flag* OR desabilitada (utilizar *polling*)

1 – Interrupção de hardware requisitada quando OR = 1

NEIE (*Noise Error Interrupt Enable*) - Este bit habilita a *flag* de ruído (NF) gerar uma requisição de interrupção de *hardware*.

0 – Interrupção pela *flag* NF desabilitada (utilizar *polling*)

1 – Interrupção de hardware requisitada quando NF = 1

FEIE (*Framing Error Interrupt Enable*) - Este bit habilita que a *flag* de erro de quadro (FE) gere uma requisição de interrupção de *hardware*.

0 – Interrupção pela *flag* FE desabilitada (utilizar *polling*)

1 – Interrupção de hardware requisitada quando FE = 1

PEIE (*Parity Error Interrupt Enable*) - Este bit habilita a *flag* de erro de paridade (PF) gerar uma requisição de interrupção de *hardware*.

0 – Interrupção pela *flag* PF desabilitada (utilizar *polling*)

1 – Interrupção de hardware requisitada quando PF = 1

*Registrador de Dados SCI **(SCIxD)***

Este registrador é na verdade separado em dois registradores. A leitura deste registrador retorna o conteúdo do *buffer* (somente de leitura) de recepção de dados. O valor escrito neste registrador vai para o *buffer* (escrita somente) de transmissão de dados. Ler e escrever neste registrador também envolve um mecanismo de limpeza automática das *flags* de estado do módulo SCI.

	7	6	5	4	3	2	1	0
R	R7	R6	R5	R4	R3	R2	R1	R0
W	T7	T6	T5	T4	T3	T2	T1	T0
Reset	0	0	0	0	0	0	0	0

A seguir é apresentado uma sugestão de configuração do módulo SCI 1, considerando os seguintes parâmetros: Clock de barramento igual a 20Mhz, paridade par, 9 bits de tamanho de caractere (devido a paridade), interrupção de recepção e interrupção de erros ativa e *baud rate* 57600.

```
// Configuração da Porta Serial
/* SCI1C1: LOOPS=0,SCISWAI=0,RSRC=0,M=1,WAKE=0,ILT=0,PE=1,PT=0 */
// 9 Bits com paridade par
SCI1C1 = 0x12;               /* Configure the SCI */
```

```
  /* SCI1C3: R8=0,T8=0,TXDIR=0,TXINV=0,ORIE=0,NEIE=0,FEIE=0,PEIE=0 */
  // Desabilita as interrupções de erro
  SCI1C3 = 0x00;

  /* SCI1S2: ??=0,??=0,??=0,??=0,??=0,BRK13=0,??=0,RAF=0 */
  // Inicialização do registrador de estado 2 limpado as flags
  SCI1S2 = 0x00;

  /* SCI1C2: TIE=0,TCIE=0,RIE=0,ILIE=0,TE=0,RE=0,RWU=0,SBK=0 */
  //
  SCI1C2 = 0x00;               /* Desabilita todas as interrupções */

  // 0x41 = 19200 bauds
  // 0x20 = 38400 bauds
  // 0x16 = 57600 bauds
  // Configura o Baud Rate para 57600
  SCI1BDH = 0x00;
  SCI1BDL = 0x16;

  /* SCI1C3: ORIE=1,NEIE=1,FEIE=1,PEIE=1 */
  // Dados de transmissão não invertidos
  // Habilita interrupção de erro de: overrun, ruído, enquadramento e paridade
  SCI1C3 |= 0x0F;                          /* Habilita interrupções de erro */

  // Habilita transmissão e recepção. Habilita interrupção de recepção
 // SCI1C2: TIE=0,TCIE=0,RIE=1,ILIE=0,TE=1,RE=1,RWU=0,SBK=0
  SCI1C2 = 0x2C;
```

Abaixo são sugeridos exemplos básicos de códigos para as interrupções do módulo SCI, bem como exemplos de funções para transferência de dados SCI.

OBS: Estes códigos consideram a utilização de caractere com 9 bits utilizando paridade.

Interrupção de Transmissão do Módulo SCI:

```
interrupt void serial_tx(void){
  /* Leitura do registrador SCI1S1 para analisar o estado da transmissão */
  (void)SCI1S1;

  /* Leitura do registrador SCI1C3 para limpar o bit de paridade */
  (void)SCI1C3;
}
```

Interrupção de Recepção do Módulo SCI:

```
interrupt void serial_rx(void){
  /* Leitura do registrador SCI1S1 para analisar o estado da transmissão */
  (void)SCI1S1;

  /* Leitura do registrador SCI1C3 para limpar o bit de paridade */
  (void)SCI1C3;

  // valor é uma variável do tipo bit declarada como global
  valor = SCI1D;                        /* Leitura dos dados recebidos */
}
```

Interrupção de Erros do Módulo SCI:

```
interrupt void serial_erro(void){
  /* Leitura do registrador SCI1S1 para analisar o estado da transmissão */
  (void)SCI1S1;

  /* Leitura do registrador SCI1C3 para limpar o bit de paridade */
  (void)SCI1C3;

  // A próxima ação depende do tipo de erro
  // Devemos analisar o erro ocorrido e tomar uma das atitudes abaixo

  // Leitura do registrador SCI1D para limpar os indicativos de erro.
  (void)SCI1D;

  // Escrita no registrador SCI1D para limpar os indicativos de erro.
  SCI1D = 0;
}
```

Função para Transmitir um Caractere Via SCI:

```
void Serial_Envia_Caracter(byte caracter)
{
  // Espera o fim da transmissão do caractere anterior para transmitir o próximo
  while (!SCI1S1_TC);

  /* Armazena o caractere a ser transmitido no registrador de transmissão */
  SCI1D = caracter;
}
```

Função para Transmitir uma String Via SCI:

```
void Serial_Envia_Frase(char *string)
{
  while(*string){
  // Espera o fim da transmissão do caractere anterior para transmitir o próximo
    while (!SCI1S1_TC);

  /* Armazena o caractere a ser transmitido no registrador de transmissão */
    SCI1D = *string;

  // Incrementa o ponteiro da string a ser transmitida
    string++;
  }
}
```

7. Configuração do Módulo PWM

O módulo de geração de Modulação por Largura de Pulso (PWM) é um recurso muito utilizado para o controle de motores e conversores CC-CC em geral. A partir dele é possível gerar um sinal analógico, apesar de sua saída ser um sinal digital que assume apenas os níveis lógicos alto (um) e baixo (zero). A saída gerada é uma onda quadrada, com freqüência constante e largura de pulso variável. Estes conceitos estão diretamente relacionados com o período fixo e o ciclo ativo (*duty cycle*) respectivamente.

A freqüência de uma onda pode ser definida como a quantidade de vezes que ela se repete no tempo. E o período é cada pedaço dessa onda que irá se repetir.

O *duty cycle* define o tempo de sinal ativo (nível lógico alto) em um período fixo. Assim, quando temos um *duty cycle* de 100%, temos nível lógico alto por todo o período. Um *duty cycle* de 50% define a metade do período em nível lógico alto e a outra metade em nível lógico baixo. Se uma saída TTL for utilizada, a tensão média de saída em um *duty cycle* de 50% será 2,5V. Estes conceitos são demonstrados na figura abaixo. Devemos lembrar que o PWM nem sempre possui estado inicial positivo, podendo iniciar o período com nível lógico baixo.

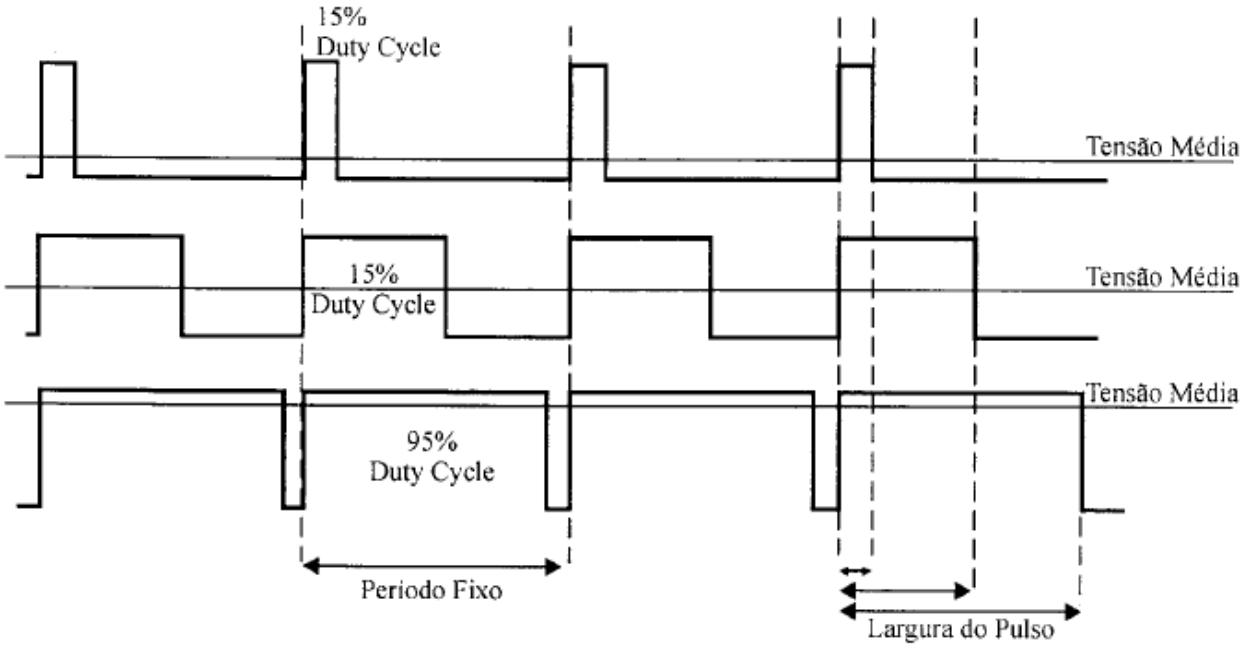
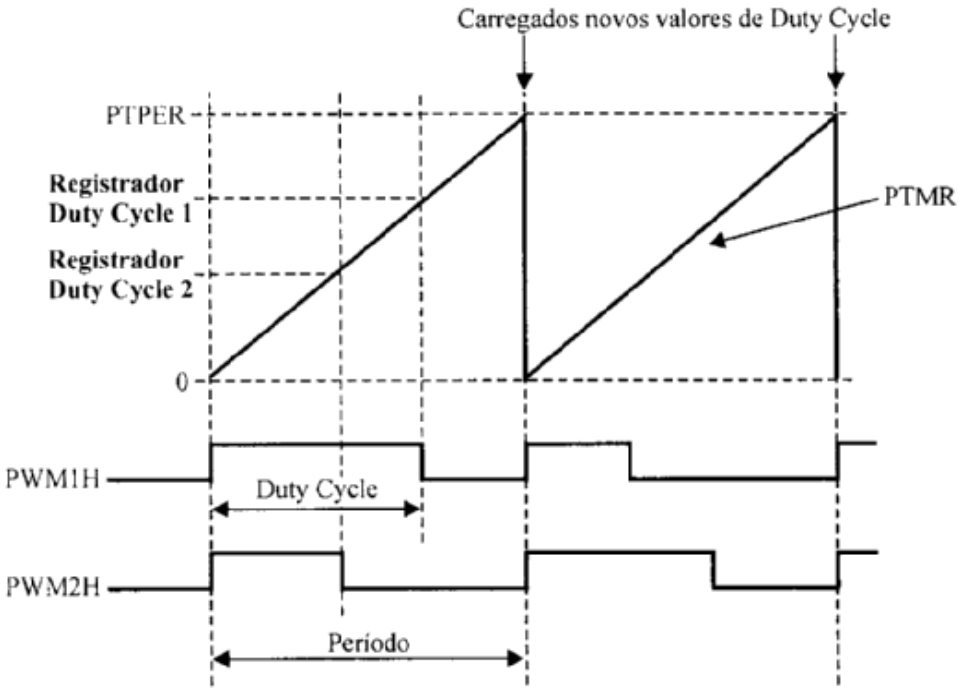


Figura – Sinais modulados por Largura de Pulso

A base de tempo dos módulos PWM normalmente é implementada de duas formas. Uma destas formas é utilizando o próprio módulo temporizador como base de tempo no PWM, ou seja, se o temporizador está configurado para um período de 1ms, a

freqüência do PWM será de 1 KHz. A outra forma é utilizando um temporizador específico para o PWM, que deve ser configurado para a freqüência desejada. Ainda, um temporizador pode ser utilizado como base de tempo de várias saídas PWM, ou seja, vários PWM com a mesma freqüência, mas larguras de pulso diferentes.

A figura a seguir irá exemplificar o funcionamento de um PWM em um microcontrolador onde o registrador PTPER possui o valor referente ao período do PWM e os registradores PWM1H e PWM2H representam dois canais de saída PWM.



Sinais PWM com mesmo período e largura de pulso diferentes.

Para exemplificar a configuração de um módulo PWM será utilizado novamente o microcontrolador MC9S08AW60. Este microcontrolador utiliza o temporizador (interrupção de estouro de tempo) como base de tempo para o período do PWM. Desta forma, para configurar a freqüência do PWM deve-se utilizar a mesma metodologia adotada para o cálculo dos valores dos registradores que controlam a interrupção de estouro de tempo. Neste exemplo será configurado um PWM de 20 KHz com largura de pulso inicial do ciclo ativo de 40% e clock de barramento em 20Mhz.

Período da base de tempo do relógio: $Período = \frac{1}{(20 \times 10^6)} = 50 \times 10^{-9}$

Valor de módulo de tempo para 50µs: $Módulo = \frac{50 \times 10^{-6}}{(50 \times 10^{-9})} = 1000$

Estando o módulo da base de tempo do PWM configurado nos registradores TPMxSC e TPMxMODH:TPMxMODL (conforme seção que trata sobre interrupção de estouro de tempo), deve-se partir para a configuração da largura do pulso, do nível do ciclo inicial e do modo de operação do PWM.

A seguir iremos apresentar algumas características do módulo PWM deste microcontrolador, bem como os registradores de configuração dos canais PWM, de modo a podermos continuar com a configuração do exemplo de PWM apresentado acima.

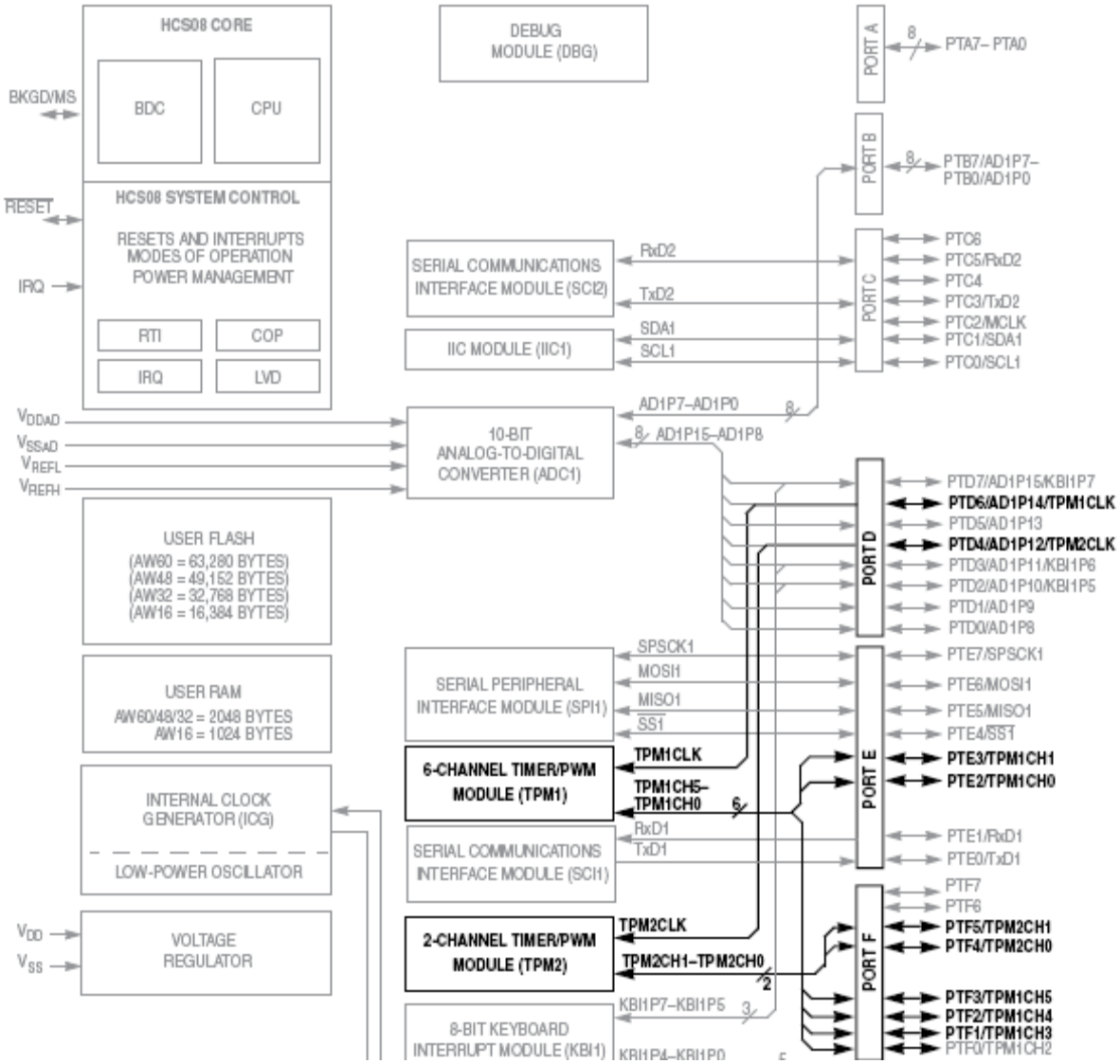
A série de microcontroladores MC9S08AW inclui dois módulos temporizadores / PWM (TPM) que suportam as funções tradicionais de captura de entrada, comparação de saída e PWM alinhados pela borda com *buffer* em cada canal. Um bit de controle no registrador de controle de cada módulo TPM configura todos os canais daquele módulo para operar como PWM alinhado pela borda ou alinhado ao centro. Em cada um destes 2 módulos TPM as funções de tempo são baseadas em um contador de 16 bits com características de pré-escala e módulo para controlar a freqüência e a escala (período entre estouros de tempo) da referência de tempo (já visto na seção sobre interrupção de estouro de tempo). Este sistema de tempo é aplicável a uma grande gama de aplicações de controle e, a capacidade de gerar PWMs alinhados ao centro estende estes campos de aplicação para controle de motores.

Características dos Módulos TPM

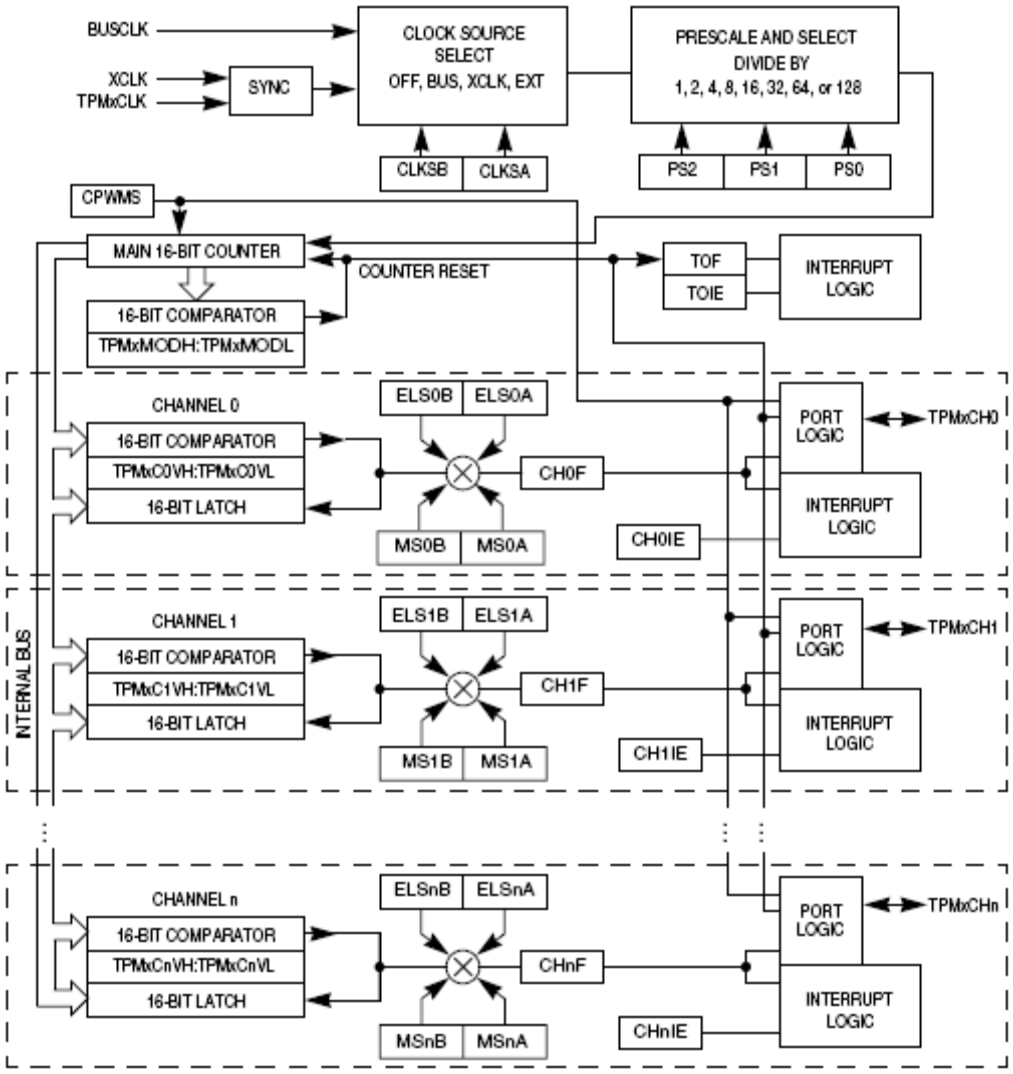
O sistema temporizador na série de microcontroladores MC9S08AW inclui um TPM1 com 6 canais e um TPM2 separado com 2 canais. As principais características do sistema temporizador vinculadas a geração de PWMs são:

- Total de 8 canais
- Cada canal pode ser configurado para captura de entrada, comparação de saída ou PWM com *buffer* alinhado na borda.
- Polaridade das saídas PWM configuráveis
- Cada módulo TPM pode ser configurado para geração de PWMs alinhados ao centro (CPWM) em todos os canais
- Operação de contagem de 16 bits em *free-running* ou *up / down* (CPWM)
- Uma interrupção por canal mais uma interrupção de estouro de tempo por módulo TPM.

A seguir o diagrama de blocos do microcontrolador com os módulos TPM em evidencia é apresentado.



E ainda temos o diagrama de blocos dos módulos TPM.



O componente central do módulo TPM é um contador de 16 bits que pode operar como contador *free-running*, contador de módulo ou contador *up / down* quando o módulo TPM estiver configurado para PWM alinhado ao centro. Todos os canais podem ser programados independentemente.

Devemos lembrar que os registradores TPMxSC, TPMxCNTH: TPMxCNTL e TPMxMODH:TPMxMODL já foram apresentados na seção sobre interrupção de estouro de tempo. Desta forma iremos analisar a seguir os outros registradores de configuração dos módulos TPM.

Registrador de Estado e Controle do Canal n do TPM x ***(TPMxCnSC)***

O registrador TPMxCnSC contém a *flag* de estado da interrupção e os bits de controle que são utilizados para configurar a habilitação da interrupção, configurar o canal e definir a função do pino associado ao canal.

	7	6	5	4	3	2	1	0
R	CHnF	CHnIE	MSnB	MSnA	ELSnB	ELSnA	0	0
W								
Reset	0	0	0	0	0	0	0	0

A seguir serão apresentados os bits de controle e suas funções:

CHnF (Channel n Flag) – Quando o canal n é configurado para PWM alinhado a borda ou comparação de saída o bit CHnF será levado para nível lógico 1 quando o valor no registrador do contador TPM for igual ao contido no registrador de valor do canal n do mesmo TPM. Esta *flag* é raramente utilizada com PWMs alinhados ao centro, porque será levado a nível lógico 1 sempre que o contador atingir o valor do canal, que corresponde a ambas as bordas do período de *duty cycle* ativo.

Uma interrupção é requisitada quando a *flag* CHnF passa para nível lógico 1 e a interrupção do canal estiver habilitada (ChnIE = 1). A *flag* CHnF é limpa através da leitura do registrador TPMxCnSC enquanto CHnF = 1 e então escrevendo 0 para o bit CHnF. Se outra interrupção ocorrer antes de que a seqüência para limpar a *flag* seja completada, a seqüência deverá ser reiniciada. Caso contrário, a *flag* ChnF permanecerá em nível lógico 1 mesmo depois que a seqüência para limpar a *flag* esteja completa. Desta forma nenhuma requisição de interrupção será perdida pela processo para limpar um *flag* anterior.

OBS: Verifique que a única diferença entre o modo de comparação de saída e o modo PWM alinhado a borda é que em modo PWM ocorrerá uma inversão na saída do canal quando ocorrer o estouro de tempo do TPM associado, ou seja, quando TPMxCNT = TPMxMOD.

0 – Nenhum evento de comparação de saída ocorreu no canal (TPMxCNT não atingiu ainda o valor contido em TPMxCnV)

1 – Ocorreu um evento de comparação de saída (TPMxCNT = TPMxCnV)

CHnIE (*Channel n Interrupt Enable*) – Este bit habilita a interrupção associada ao canal n.

0 – Requisição de interrupção do canal n desabilita (utilizar *polling*)

1 – Requisição de interrupção do canal n habilitada

MSnB (*Mode Select B for TPM Channel n*) – Quando CPWMS = 0 e MsnB = 1 o canal n será configurado para PWM alinhado a borda. Veja outros modos na tabela abaixo.

MSnA (*Mode Select A for TPM Channel n*) – Veja tabela abaixo.

ELSn[B:A] *(Edge / Level Select Bits)* – Estes bits selecionam a polaridade da saída do PWM, conforme tabela a seguir.

A seguir é apresentada a tabela com as possíveis configurações para o canal n.

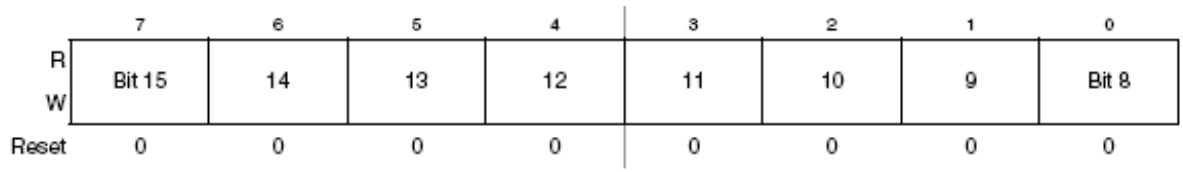
CPWMS	MSnB:MSnA	ELSnB:ELSnA	Modo	Configuração
X	**XX**	**00**	Pino não utilizado como canal TPM. Utilizado como entrada para clock externo do TPM ou revertido como pino de entrada / saída de propósito geral.	
0	**00**	**01**	Captura de Entrada	Captura de entrada somente na borda de subida
		10		Captura de entrada somente na borda de descida
		11		Captura de entrada em ambas as bordas
	01	**00**	Comparação de Saída	Somente comparação por software
		01		Inversão da saída na comparação
		10		Saída para nível lógico 0 na comparação
		11		Saída para nível lógico 1 na comparação
	1X	**10**	PWM alinhado a borda	Pulsos iniciando em nível lógico alto (saída do PWM passa para nível lógico 0 na comparação)
		X1		Pulsos iniciando em nível lógico baixo (saída do PWM passa para nível lógico 1 na comparação)
1	**XX**	**10**	PWM alinhado ao centro	Pulsos iniciando em nível lógico alto (saída do PWM passa para nível lógico 0 na comparação da contagem *up*)
		X1		Pulsos iniciando em nível lógico baixo (saída do PWM passa para nível lógico 1 na comparação da contagem *up*)

Tipicamente um programa deve limpar a *flag* de estado (CHnF) depois de alterar o modo de operação do canal e antes de habilitar a interrupção do canal para evitar qualquer comportamento inesperado.

Registradores de Valor do Canal n do TPM x **(TPMxCnVH: TPMxCnVL)**

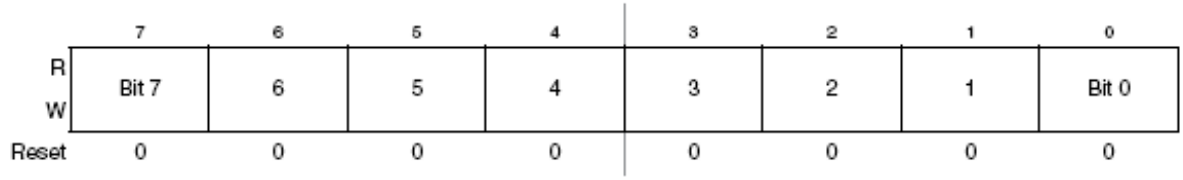
Estes registradores de leitura / escrita contém o valor para a comparação de saída quando operando em modo PWM ou comparação de saída.

Ainda, nos modos de comparação de saída ou PWM, escrever para os bytes TPMxCnVH ou TPMxCnVL trava o valor deste em um buffer. Quando ambos os bytes forem escritos estes serão transferidos como um valor coerente de 16 bits para os registradores de valor do canal.



	7	6	5	4	3	2	1	0
R W	Bit 15	14	13	12	11	10	9	Bit 8
Reset	0	0	0	0	0	0	0	0

Timer x Channel Value Register High (TPMxCnVH)



	7	6	5	4	3	2	1	0
R W	Bit 7	6	5	4	3	2	1	Bit 0
Reset	0	0	0	0	0	0	0	0

Timer Channel Value Register Low (TPMxCnVL)

Descrição Funcional do Módulo PWM

Todas as funções do módulo TPM são associadas a um contador principal de 16 bits que permite uma flexibilidade na seleção de fonte de *clock* e no divisor de pré-escala. Um registrador de módulo de 16 bits também é associado com o contador principal do módulo TPM. Cada canal TPM é opcionalmente associado com um pino do microcontrolador e uma função de interrupção mascarável. O módulo TPM possui capacidades de geração de PWMs alinhas ao centro controladas pelo bit de controle CPWMS do registrador TPMxSC. Quando CPWMS é levado para nível lógico 1, o contador TPMxCNT se altera para modo de contagem *up / down* e todos os canais do módulo TPM associado passam a operar como canais de PWM alinhados ao centro. Quando CPWMS = 0, cada canal pode independentemente ser configurado para operar em modo de captura de entrada, comparação de saída ou PWM com *buffer* alinhado a borda.

Uma *flag* de interrupção e sinal de habilitação (*enable)* são associados com o contador principal de 16 bits. A *flag* de estouro de tempo (TOF) é uma indicação acessível por software de que a contagem chegou ao seu fim. O sinal de habilitação seleciona entre um *polling* por *software* (TOIE = 0) onde não há geração de interrupção por *hardware* ou operação acionada por interrupção (TOIE = 1) onde uma interrupção por *hardware* é automaticamente gerada quando a *flag* TOF for para nível lógico 1.

As condições que levam TOF para nível lógico 1 dependem do modo do contador (contador *up* ou *up / down*). Com o contador no modo up, o contador principal irá contar de 0x0000 até 0xFFFF e retornar para 0x0000 na próxima contagem. O bit TOF passa para nível lógico 1 na transição de 0xFFFF para 0x0000. Quando um valor limite de módulo existe (TPMxMOD diferente de 0x0000), o bit TOF passa para nível lógico 1 na transição do valor de módulo para 0x0000.

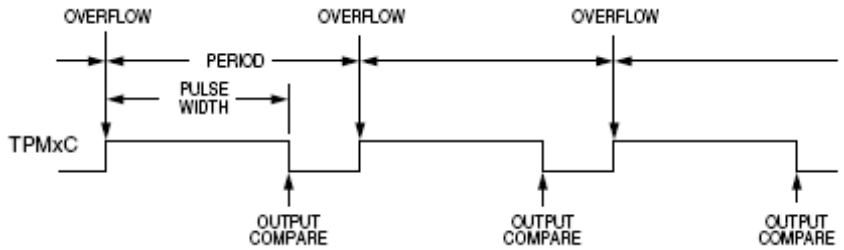
Quando o contador principal está operando em modo *up / down*, a flag TOF passa para nível lógico 1 quando a contagem troca de direção, na transição do valor de módulo para a contagem regressiva. Isto corresponde ao fim do período do PWM (o valor de contagem 0x0000 corresponde ao centro do período).

O contador principal pode ser reiniciado manualmente a qualquer momento pela escrita de qualquer valor valor em um dos registradores do contador (TPMxCNTH ou TPMxCNTL).

PWM Alinhado na Borda

Este tipo de saída PWM utiliza o contador operando em modo progressivo (CPWMS = 0) e pode ser utilizado quando outros canais no mesmo módulo TPM são configurados para captura de entrada ou comparação de saída. O período do sinal PWM é determinado pelo valor dos registradores de módulo (TPMxMODH: TPMxMODL). O ciclo de trabalho (*duty cycle*) é determinado pelo conteúdo dos registradores de valor do canal (TPMxCnVH: TPMxCnVL). A polaridade do sinal PWM é determinada pelo bit de controle ELSnA. Casos de ciclos de trabalho de 0 e 100% são possíveis.

Como é demonstrado na figura abaixo, o valor da comparação de saída no registrador de canal TPM determina a largura de pulso (*duty cycle*) do sinal PWM. O tempo entre o estouro de tempo do módulo (TPMxCNT = TPMxMOD) e a comparação de saída é a largura do pulso. Se ELSnA = 0, o evento de estouro de tempo irá forçar o sinal do PWM para nível lógico alto e a comparação de saída irá forçar o sinal do PWM para nível lógico baixo. Se ELSnA = 1, o evento de estouro de tempo irá forçar o sinal do PWM para nível lógico baixo e a comparação de saída irá forçar o sinal do PWM para nível lógico alto.



Quando o registrador de valor do canal (TPMxCnVH:TPMxCnVL) for escrito para 0x0000, o ciclo ativo passará a ser 0%. Ao configurar o registrador de valor do canal para valores maiores do que o registrador de módulo (TPMxMODH:TPMxMODL) o ciclo ativo passará para 100%.

Devido a arquitetura de 8 bits dos microcontroladores da família HCS08 as configurações dos registradores de valor do canal são protegidas por *buffers* para garantir atualizações coerentes de 16 bits, bem como evitar larguras de pulso inesperadas.

No modo PWM com alinhamento na borda atualizações nos registradores TPMxCnVH e TPMxCnVL só serão passadas para os mesmos após ambos estarem escritos e o valor do contador principal estar em 0x0000. O novo *duty cycle* não terá efeito até que comece um novo período.

PWM Alinhado ao Centro

Este tipo de saída PWM utiliza o contador principal no modo *up / down* (CPWMS=1). O valor de comparação de saída do canal (TPMxCnVH:TPMxCnVL) determina a largura de pulso do sinal PWM e o período é determinado pelo calor contido nos registradores de módulo (TPMxMODH: TPMxMODL). Os registradores de módulo devem ser restringidos a valores entre 0x0001 até 0x7FFF porque valores fora deste faixa pode produzir resultados ambíguos. O bit ELSnA irá determinar a polaridade de saída do PWM alinhado ao centro.

$$Largura\ de\ Pulso = 2 * (TPMxCnVh : TPMxCnVL)$$

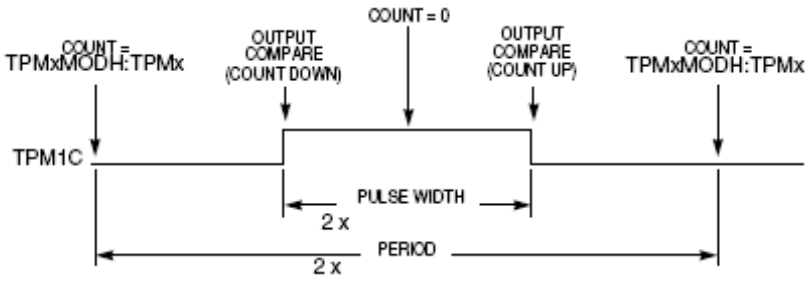
$$Período = 2 * (TPMxMODH : TPMxMODL)\ para\ valores\ de\ TPMxMOD\ entre\ 0\text{x}0000\ e\ 0\text{x}7\text{FFF}$$

Se os registradores de valor do canal (TPMxCnVH:TPMxCnVL) forem configurados para zero ou valores negativos (bit 15 em nível lógico 1), o ciclo ativo do PWM será 0%. Se estes registradores tiverem valor positivo (bit 15 em nível lógico 0) e maior do que o valor configurado de módulo (TPMxMOD), o ciclo ativo do PWM será 100%, porque a comparação de ciclo ativo não ocorrerá. Isto implica em uma faixa útil para o período configurado nos registradores de módulo entre 0x0001 e 0x7FFE (0x7FFF se a geração de ciclos ativos de 100% não é necessária). Esta não é uma limitação significativa, pois o período resultante é muito maior do que os requeridos em aplicações normais.

Os registradores TPMxMODH: TPMxMODL = 0x0000 é um caso especial que não deve ser utilizado com o PWM alinhado ao centro. Se CPWMS = 1 torna-se obrigatório utilizar um valor válido para os registradores de módulo (valores diferentes de 0x0000) para que possa ocorrer a alteração de direção de contagem progressiva para contagem regressiva.

A figura a seguir apresenta o valor de comparação de saída nos registradores de canal do módulo TPM (multiplicado por 2), que determina a largura de pulso do sinal de PWM alinhado ao centro.

O sinal de saída do PWM alinhado ao centro será forçado para nível lógico 0 quando ocorrer um evento de comparação de saída (TPMxCNT = TPMxMOD) se ELSnA=0 e o contador estiver operando em contagem progressiva. Se o contador estiver operando em modo regressivo nestas condições, o sinal de saída do PWM será forçado para nível lógico alto. O contador operá em modo progressivo até que atinja o valor configurado nos registradores de módulo (TPMxMODH:TPMxMODL). Então o contador passa a operar em modo regressivo até que atinja 0x0000. Este comportamento faz com que o período seja igual a duas vezes o valor configurado nos registradores de módulo.



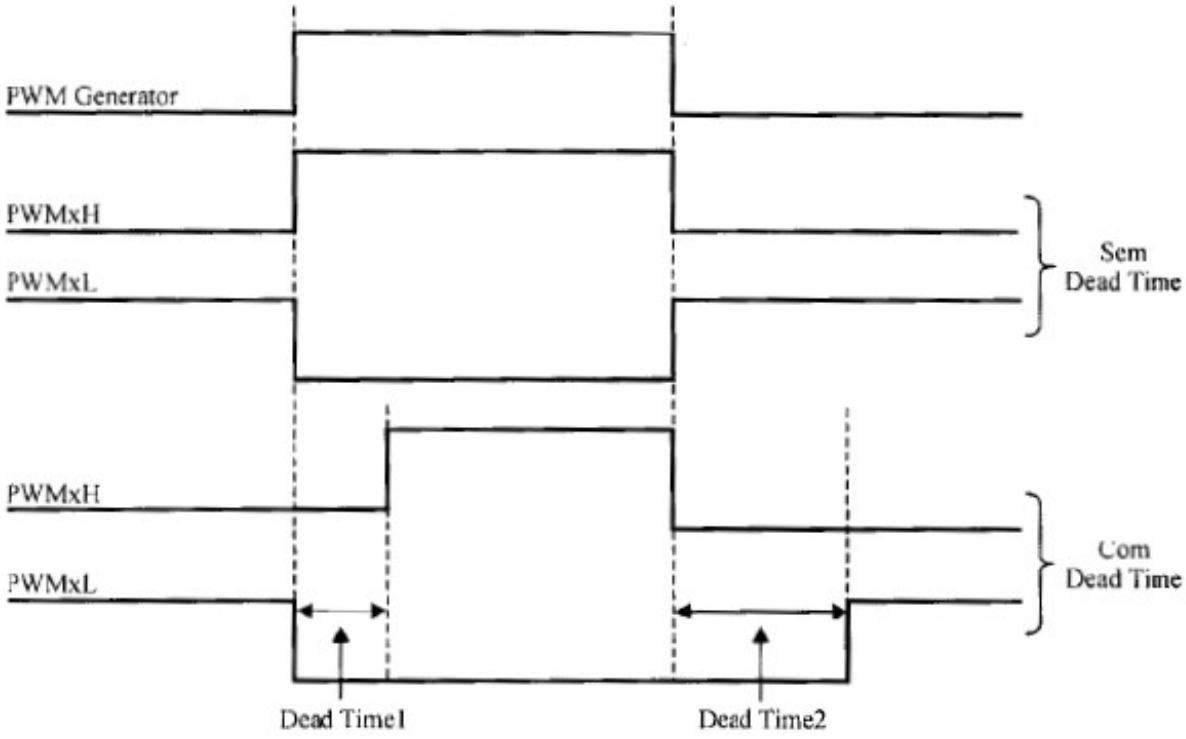
PWM alinhados ao centro tipicamente produzem menos ruído do que PWM alinhados a borda porque ocorrem menos transições nos pinos de saída alinhadas com a mesma borda de clock do sistema. Este tipo de PWM é requerido também em algumas aplicações para acionamento de motores. Quando escrevermos nos registradores TPMxMOD e TPMxCnV estaremos na verdade escrevendo para um buffer. Quando ambos os valores de 8 bits destes registradores forem escritos e o contador reverter sua direção é que seus valores serão atualizados. Este comportamento garante valores coerentes de 16 bits e evita larguras de pulso inesperadas na saída PWM.

Opcionalmente, quando TPMxCNT = TPMxMOD, o módulo TPM pode gerar uma interrupção associada a flag TOF. O usuário pode utilizar este evento para alterar valores na largura de pulso do PWM, fazendo com que os novos valores passem a valer instantaneamente, no início do novo período.

Quando o PWM estiver em operação, escrever para o registrador TPMxSC cancela os valores escritos nos registradores TPMxMODH e TPMxMODL. Ainda, escrever para o registrador TPMxCnSC cancela os valores escritos nos registradores de valor do canal (TPMxCnVH: TPMxCnVL).

Em algumas aplicações é necessário a inserção de um tempo morto (*dead time*) devido ao tempo de comutação de certos tipos de componentes utilizados ou devido as topologias dos sistemas agregados a estas saídas PWM.

Normalmente estes tempos mortos são inseridos em pares complementares de saídas PWM, ou seja, quando uma das saídas comuta para nível lógico alto, a outra saída comuta para nível lógico baixo. Abaixo é apresentado um exemplo de utilização de tempos mortos.



A seguir serão apresentados exemplos de códigos de inicialização para PWMs operando em modo alinhado a borda e alinhado ao centro. Devemos considerar as seguintes condições: Estamos utilizando um *clock* de barramento de 20Mhz e iremos gerar um PWM com freqüência de 20Khz. Ainda, iremos utilizar o módulo 2 TPM, onde o canal 0 será configurado para pulsos iniciando em nível lógico alto e o canal 1 para pulsos iniciando em nível lógico baixo. Os ciclos ativos serão: canal 0 = 40% e canal 1 = 50%.

Relembrando o cálculo do módulo de período visto anteriormente:

Valor de módulo de tempo para 50µs: $Módulo = \frac{50\times10^{-6}}{(50\times10^{-9})} = 1000$

e para o PWM alinhado ao centro:

Valor de módulo de tempo para 50µs: $Módulo = \frac{(\frac{50\times10^{-6}}{(50\times10^{-9})})}{2} = 500$

PWM ALINHADO A BORDA:

```
// Geração de PWM alinhado a borda
/* TPM1SC: TOF=0,TOIE=0,CPWMS=0,CLKSB=0,CLKSA=0,PS2=0,PS1=0,PS0=0 */
TPM2SC = 0x00;               //Para o módulo TPM2

// Canal 0 iniciando com sinal em nível lógico alto
// Interrupção do canal 0 desabilitada
TPM2C0SC = 0x28;
// Largura de pulso do canal zero = 400 x 50ns = 20us (f = 20Mhz)
TPM2C0V = 400;

// Canal 1 iniciando com sinal em nível lógico baixo
// Interrupção do canal 1 desabilita
TPM2C1SC = 0x24;
// Largura de pulso do canal um = 500 x 50ns = 25us (f = 20Mhz)
TPM2C1V = 500;


// PWM alinhado a borda com interrupção de estouro de tempo ativa
/* TPM1SC: TOF=0,TOIE=1,CPWMS=0,CLKSB=0,CLKSA=1,PS2=0,PS1=0,PS0=0 */
TPM2SC = 0x48;

// Configuração do período do PWM para 100us (1000 x 50ns)
TPM2MOD = 1000;
```

PWM ALINHADO AO CENTRO:

```
// Geração de PWM alinhado ao centro
/* TPM1SC: TOF=0,TOIE=0,CPWMS=0,CLKSB=0,CLKSA=0,PS2=0,PS1=0,PS0=0 */
TPM2SC = 0x00;               //Para o módulo TPM2

// Canal 0 iniciando com sinal em nível lógico alto
// Interrupção do canal 0 desabilitada
TPM2C0SC = 0x08;
// Largura de pulso do canal zero = 2 x 200 x 50ns = 20us (f = 20Mhz)
TPM2C0V = 200;

// Canal 1 iniciando com sinal em nível lógico baixo
// Interrupção do canal 1 desabilitada
TPM2C1SC = 0x04;
// Largura de pulso do canal um = 2 x 250 x 50ns = 25us (f = 20Mhz)
TPM2C1V = 250;


// PWM alinhado a borda com interrupção de estouro de tempo ativa
/* TPM1SC: TOF=0,TOIE=1,CPWMS=0,CLKSB=0,CLKSA=1,PS2=0,PS1=0,PS0=0 */
TPM2SC = 0x68;

// Configuração do período do PWM para 100us (2 x 500 x 50ns)
TPM2MOD = 500;
```

OBS: Não é recomendado habilitar as interrupções de canal quando utilizando o modo PWM. O processo de limpeza de *flag* das interrupções pode abortar a atualização dos registradores TPMxCnV, pois o processo envolve a escrita do registrador TPMxCnSC.

As configurações mínimas para a limpeza das *flags* das interrupções de estouro de tempo, do canal 0 e do canal 1 do módulo TPM2 são apresentadas a seguir:

```
// Interrupção de estouro de tempo do módulo TPM2
interrupt void timer2(void){
  (void)TPM2SC;                    // Lê o registrador TPM2SC
  TPM2SC_TOF = 0;                  // Limpa a flag TOF do registrador TPM2SC
}


// Interrupção associada ao canal 0 do módulo TPM2
interrupt void t2canal0(void){
  (void)TPM2C0SC;                  // Lê o registrador TPM2C0SC
  TPM2C0SC_CH0F = 0;               // Limpa a flag CH0F do registrador TPM2C0SC
}


// Interrupção associada ao canal 1 do módulo TPM2
interrupt void t2canal1(void){
  (void)TPM2C1SC;                  // Lê o registrador TPM2C1SC
  TPM2C1SC_CH1F = 0;               // Limpa a flag CH1F do registrador TPM2C1SC
```

Ainda, é apresentado abaixo uma sugestão de função para alterar a largura de pulso nos canais do módulo TPM2, tendo como parâmetros o canal a ser alterado e a largura de pulso desejada em valores percentuais.

```
// Função para mudança de largura de pulso do PWM
// canal 0 ou 1
// percent de 0 a 100%
void largura_pulso(byte canal,byte percent){
  switch(canal){              // Escolha do canal a ser alterado
    case 0:
      TPM2C0V = (TPM2MOD * percent) / 100;
      break;
    case 1:
      TPM2C1V = (TPM2MOD * percent) / 100;
      break;
    default:
      break;
  }
}
```

8. Configuração do Modo de Captura de Entrada

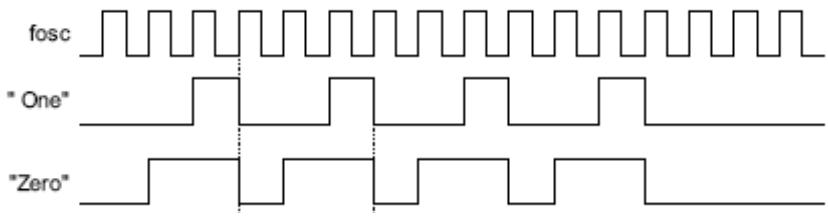
O modo de captura de entrada utiliza assim como no modo PWM o contador principal de 16 bits como seu componente fundamental. Este modo é utilizado em aplicações bem específicas, como captura de informação de *encoders* analógicos, decodificação de sinal serial recebido por módulos RF (normalmente módulos RF de 433Mhz utilizados em controles de garagens, portões, etc), bem como para determinar distância percorrida e velocidade em veículos simples.

Com a função de captura de entrada o módulo TPM pode capturar o tempo no qual um evento externo ocorreu. No momento em que uma transição de sinal ocorrer (borda de subida ou descida) no pino associado ao canal configurado como captura de entrada, o módulo TPM copia o conteúdo do contador (TPMxCNT) para o registrador de valor do canal (TPMxCnV). A borda ativa que dispara o gatilho de um evento de captura de entrada pode ser configurada para borda de subida, borda de descida ou ambas as bordas.

Um evento de captura de entrada leva o bit de *flag* do canal (CHnF) para nível lógico 1. Esta *flag* pode ser opcionalmente associada a uma requisição de interrupção do microcontrolador.

Um exemplo clássico de utilização do modo de captura de entrada é na decodificação de sinais RF. O estado lógico de um sinal transmitido serialmente por um *encoder* padrão para sinais RF é determinado pelo tempo de atividade da freqüência de operação do módulo (por exemplo 433Mhz). Esta modulação é uma variação da modulação AM digital, conhecida por ASK (*amplitude shift keying*).

Pode-se perceber na figura abaixo que o sinal gerado é sincronizado por um sinal de *clock* (f_{osc}). Um estado lógico 0 é representado pelo tempo de 2 períodos deste *clock*. O estado lógico 1 é representado por 1 período deste *clock*.



Desta forma, para decodificarmos este sinal no lado do receptor, devemos contar o tempo de atividade do sinal recebido. Esta contagem deve ser realizada a partir da borda de subida do sinal codificado até ocorrer uma borda de descida. Analisando o caso, fica evidente que o modo de captura de entrada é ideal para a decodificação deste sinal.

Normalmente o *clock* de sincronismo utilizado pelos *encoders* de RF opera em torno de 2 Khz. Podemos verificar então que quando identificarmos um sinal com duração em torno de 500µs, o mesmo estará representando um estado lógico 1. Quando identificarmos um sinal com duração próxima de 1000µs, este estará representando um estado lógico 0.

A seguir é apresentado um exemplo de código de inicialização para o módulo TPM1 com o canal 2 operando em modo de captura de entrada. Iremos configurar o módulo para captura em ambas as bordas. O registrador de módulo deve ser configurado para gerar tempos bem maiores que os tempos de transmissão de um bit, para evitarmos que ocorra mais de um estouro de tempo dentro da decodificação de um bit, fato este que invalidaria o bit amostrado. Desta forma iremos configurar o estouro de tempo para 20ms, utilizando um *clock* de barramento de 20Mhz.

Valor de módulo de tempo para 20ms: $Módulo = \frac{20 \times 10^{-3}}{(50 \times 10^{-9})} = 400000$

Para representar 400000 é necessário mais do que os 16 bits disponíveis. Desta forma se faz necessário a divisão da base de tempo. Dividindo 400000 pelo maior valor possível em 16 bits, encontramos 6,104. Assumimos então o próximo valor válido de divisão, ou seja, 8, como fator de divisão da base de tempo.

Fator de divisão da base de tempo: $Fator = \frac{400000}{65535} = 6{,}104$

E, recalculando os valores de configuração:

Período relativo a base de tempo: $Período = \frac{1}{(\frac{20 \times 10^{6}}{8})} = 400 \times 10^{-9}$

Cálculo do valor de módulo de tempo: $Módulo = \frac{20 \times 10^{-3}}{(400 \times 10^{-9})} = 50000$

Desta forma encontramos um vamos de módulo igual a 50000, que é um valor válido em 16 bits.

A seguir o código que implementa estas condições é demonstrado.

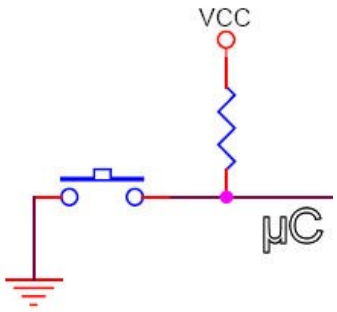
```
//Inicialização do Modo Captura de Entrada
/* TPM1SC: TOF=0,TOIE=1,CPWMS=0,CLKSB=0,CLKSA=1,PS2=0,PS1=1,PS0=1 */
// Fonte de clock = clock de barramento / 8
TPM1SC = 0x4B;
TPM1MOD = 50000;                    // configura o timer para 20ms
TPM1C2SC = 0x4C;                    // Captura de entrada para ambas as bordas
                                    // com interrupção habilitada
```

Abaixo é apresentado uma sugestão de código para determinar o nível lógico do bit recebido em um canal RF, utilizando a interrupção de captura de entrada configurada para o canal 2 do módulo TPM1.

```
// Declaração das variáveis globais necessárias para análise
// do estado lógico do bit recebido
unsigned int tempoh,tempol,tempom;



interrupt void captura_entrada(void){
  (void)TPM1C2SC;             // Limpa flag de interrupção de captura
  TPM1C2SC_CH2F = 0;

  if (PTFD_PTFD0 == 1){       // Verifica se a borda foi de subida ou descida
                              // PTF0 é o pino associado ao canal 2 do TPM1
    tempoh = TPM1C2V;         // Guarda o valor do contador na borda de subida
  } else{
    tempol = TPM1C2V;         // Guarda o valor do contador na borda de descida
  }

  if (tempol > tempoh){           // Verifica se ocorreu um estouro de tempo
                                  // durante o tempo de um bit
    // Caso não tenha ocorrido, tempo do bit = tempol - tempoh
    tempom = tempol - tempoh;
  } else{
    /* caso tenha ocorrido, calcula-se a diferença entre o módulo
     atingido e o tempo na borda de subida e soma-se com o tempo
     na borda de descida */
    tempom = (TPM1MOD - tempoh) + tempol;
  }

  // Iremos considerar uma faixa de segurança de 125us para
  // cima e para baixo do valor esperado pelo estado lógico
  /* Valores equivalentes a tempos:
     375us = 937
     625us = 1562
     875us = 2187
     1125us = 2812
  */

  // Analise do estado lógico do bit recebido
  if ((tempom > 937) && (tempom < 1562)){
    // O valor recebido esta em nível lógico 1
  }

  if ((tempom > 2187) && (tempom < 2812)){
    // O valor recebido esta em nível lógico 0
  }
}
```

9. Configuração do Módulo de Teclado

Nesta capítulo iremos apresentar as formas mais tradicionais de conexão de chaves mecânicas às entradas de microcontroladores. Para que possamos conectar uma chave a um pino do microcontrolador devemos configurar este pino para operar como entrada. As chaves mecânicas normalmente apresentam a configuração demonstrada abaixo ao serem conectadas ao pino de um microcontrolador.



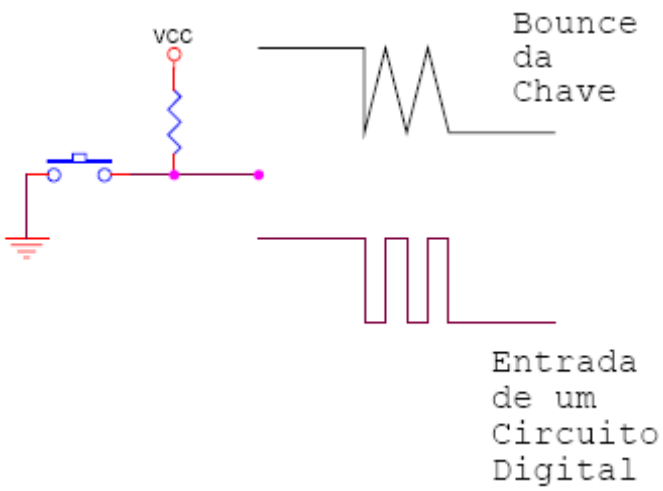
Podemos notar que o lado da chave que será conectado ao microcontrolador utiliza um resistor de *pull-up*. Este resistor irá forçar um nível lógico alto no pino do microcontrolador configurado como entrada. Caso este resistor não seja utilizado, o pino do microcontrolador ficará sujeito a ruídos.

Em alguns microcontroladores os pinos de entrada podem ser conectados internamente a um resistor de *pull-up*, evitando que seja necessário utilizar este resistor externamente.

Os microcontroladores geralmente possuem uma interrupção de *hardware* associada a pinos específicos para teclado. Esta interrupção tem como objetivo evitar que seja necessário o monitoramento por *software* dos pinos conectados a chaves. Ainda, o gatilho de requisição destas interrupções pode ser disparado por eventos de detecção de bordas de descida ou subida, ou ainda por detecção de nível de tensão.

As chaves mecânicas possuem micro ranhuras geradas no seu processo de fabricação. Devido a estas micro ranhuras, ao serem fechadas geram ruídos (causados pela vibração gerada pelas micro ranhuras) que poderão ser detectados pelo *software* desenvolvido como múltiplos cliques durante o processo de leitura. Estas vibrações que causam ruídos, também conhecidas por *contact bounce*, devem ser eliminadas por hardware (capacitor, flip-flop tipo D) ou por software, inserindo um atraso antes da leitura da chave fechada.

A figura abaixo demonstra o nível de tensão na entrada do pino de um microcontrolador conectado a uma chave recém pressionada.



O caso apresentado acima irá causar 3 interrupções se a interrupção associada a este pino estiver configurada para disparo de gatilho por borda de descida e não for implementado nenhuma técnica para evitar este ruído.

Para conectar chaves aos pinos do microcontrolador MC9S08AW60 devemos configurar os registradores associados a uma das portas de entrada / saída do microcontrolador (PTxDDn, PTxPEn, KBIPEn e KBI1SC).

Os registradores PTxDD e PTxPE foram apresentados no capítulo 2. A seguir serão apresentadas as funções dos bits de controle dos registradores KBIPE e KBI1SC.

Registrador de Estado e Controle do Módulo de Teclado KBI ***(KBI1SC)***

	7	6	5	4	3	2	1	0
R	KBEDG7	KBEDG6	KBEDG5	KBEDG4	KBF	0	KBIE	KBIMOD
W						KBACK		
Reset	0	0	0	0	0	0	0	0

KBEDG[7:4] *(Keyboard Edge Select for KBI Port Bits)* – Cada um destes bits de leitura / escrita seleciona a borda e/ou nível que será reconhecido como evento de gatilho no pino do microcontrolador configurado como entrada de interrupção de teclado (KBIPEn = 1).

0 – Borda de descida / Nível lógico 0

1 – Borda de subida / Nível lógico 1

KBF *(Keyboard Interrupt Flag)* – Este bit de *flag* de estado é levado a nível lógico 1 quando um evento associado a borda selecionada é detectado em qualquer um dos pinos configurados como entrada de interrupção de teclado. Esta *flag* é limpa através da escrita de nível lógico 1 no bit de controle KBACK. A *flag* irá permanecer em nível lógico 1 se KBIMOD = 1 e o pino de entrada de teclado permanecer no nível de tensão configurado como evento de gatilho.

Esta *flag* pode ser utilizada para *polling* em software ou poderá gerar uma requisição de uma interrupção de hardware do módulo de teclado (KBIE = 1).

0 – Não existe interrupção de teclado pendente

1 – Interrupção de teclado pendente

KBACK (*Keyboard Interrupt Acknowledge*) – Este bit somente de escrita é utilizado para limpar a *flag* de estado KBF, pela escrita de nível lógico 1. A *flag* KBF irá permanecer em nível lógico 1 se KBIMOD=1 para selecionar o modo de operação sensível a nível e borda e qualquer um dos pinos habilitados para entrada permanecerem no nível configurado como evento de gatilho. Neste cenário escrever o nível lógico 1 para o bit KBACK não irá ter efeito sobre a *flag* KBF.

KBIE *(Keyboard Interrupt Enable)* – Este bit de controle de leitura / escrita determina se interrupções de hardware serão geradas quando a *flag* de estado KBF estiver em nível lógico igual a 1. Quando KBIE = 0, nenhuma interrupção de hardware será gerada pelo módulo de teclado, mas KBF continuará sendo utilizada para *polling* por software.

0 – A flag KBF não irá gerar interrupções de hardware (utilizar *polling*)

1 – Será gerado interrupção de hardware de teclado quando KBF = 1

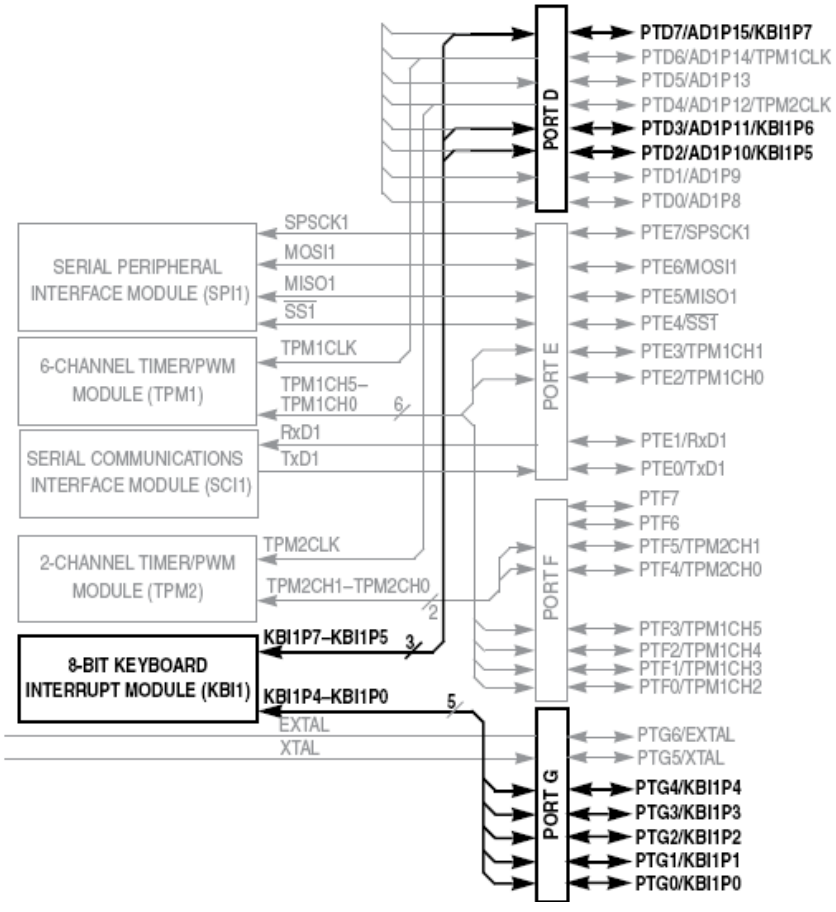
KBIMOD *(Keyboard Detection Mode)* – Este bit de controle de leitura / escrita seleciona entre detecção somente por borda ou detecção por borda e nível. Os bits 3 a 0 da porta de teclado podem detectar somente borda de descida ou borda de descida e nível lógico baixo. Os bits 7 a 4 da porta de teclado podem ser configurados para detecção como apresentado abaixo:

- Somente borda de subida ou borda de subida e nível lógico alto (KBEDGn = 1)
- Somente borda de descida ou borda de descida e nível lógico baixo (KBEDGn = 0)

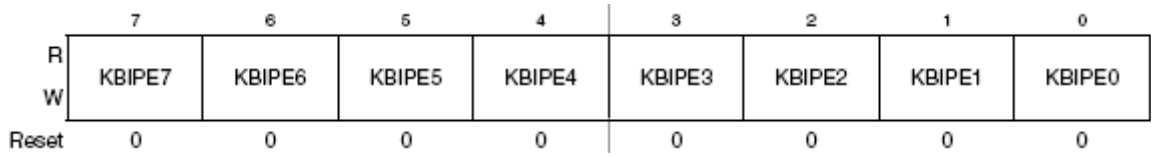
0 – Detecção somente por borda

1 – Detecção por borda e nível

Abaixo é apresentado o diagrama de blocos do microcontrolador com os pinos destinados à teclado realçados.



Registrador para Habilitação do Modo de Teclado ***(KBI1PE)***



	7	6	5	4	3	2	1	0
R W	KBIPE7	KBIPE6	KBIPE5	KBIPE4	KBIPE3	KBIPE2	KBIPE1	KBIPE0
Reset	0	0	0	0	0	0	0	0

Este registrador é utilizado para configurar se os pinos associados ao módulo de teclado serão habilitados ou não.

KBIPE[7:0] *(Keyboard Pin Enable for KBI Port Bits)* – Cada um destes bits de leitura / escrita seleciona se o pino da porta de teclado associado será habilitado como entrada de interrupção de teclado ou como pino de entrada / saída para propósito geral.

0 – Bit n da porta de teclado é um pino de entrada / saída de propósito geral não associada como o módulo de teclado.

1- Bit n da porta de teclado habilitado como entrada para interrupção de teclado.

A tabela abaixo apresenta as possíveis configuração para *pull-up* ou *pull-down* nos pinos de I/O associados ao módulo de teclado.

PTxPEn (Pull Enable)	**PTxDDn (Direção dos dados)**	**KBIPEn (Habilita pino para teclado**	**KBEDGn (Seleção de borda)**	***Pull-Up***	***Pull-Down***
0	0	0	X	desabilitado	desabilitado
1	0	0	X	Habilitado	desabilitado
X	1	0	X	desabilitado	desabilitado
1	X	1	0	Habilitado	desabilitado
1	X	1	1	desabilitado	Habilitado
0	X	1	X	desabilitado	desabilitado

A partir dos registradores de configuração apresentados acima para o microcontrolador MC9S08AW60 pode-se sugerir um exemplo de inicialização para o módulo de teclado. O exemplo a seguir considera os 8 pinos associados ao módulo de teclado (PTG0:PTG4, PTD2:PTD3 e PTD7) configurados como entrada de interrupção com *pull-up* ativo. Ainda, requisição de interrupção ativa somente para borda de descida.

```
// Configuração do módulo de teclado
// Inicia a latch da porta G com todos os bits em zero
PTGD = 0;
// Todos os pinos da porta G como entradas
PTGDD = 0;
// Pinos do módulo de teclado da porta G configurados para pull-up
PTGPE = 0x1F;
// Inicia a latch da porta D com todos os bits em zero
PTDD = 0;
// Todos os pinos da porta D como entradas
PTDDD = 0;
```

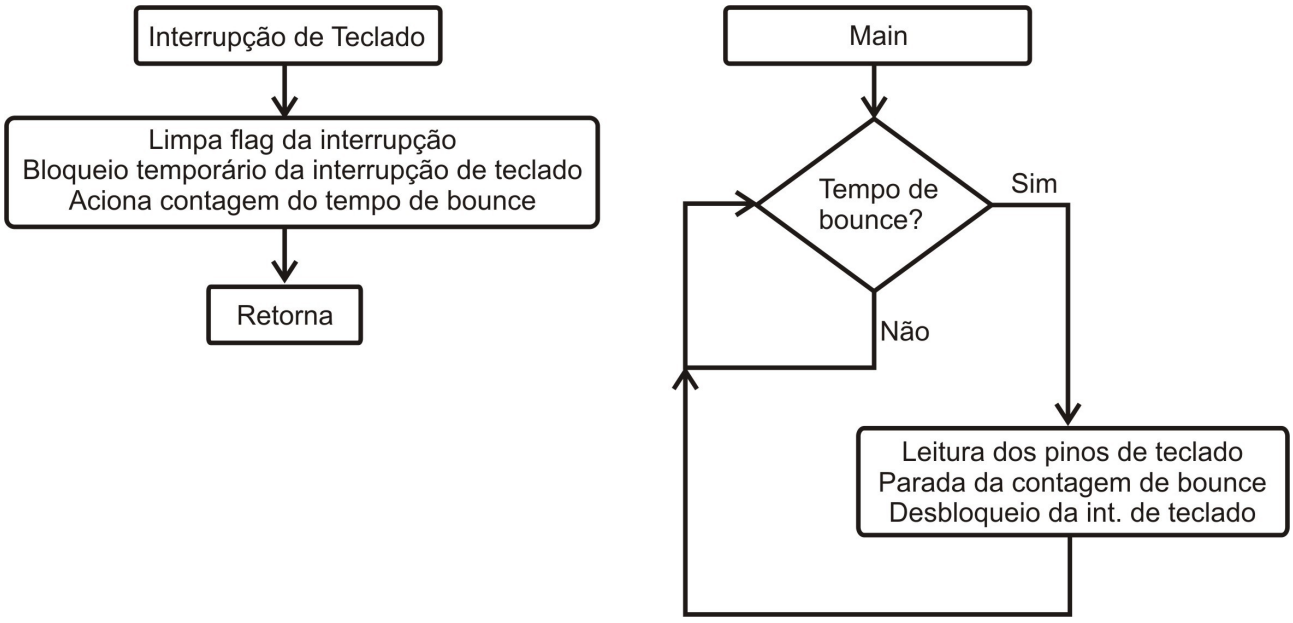
```
// Pinos do módulo de teclado da porta D configurados para pull-up
PTDPE = 0b10001100;

// 8 pinos associados ao módulo KBI configurados para teclado
KBI1PE = 0xFF;

/* KBEDG7=0,KBEDG6=0,KBEDG5=0,KBEDG4=0,KBF=0,KBACK=0,KIE=1,KBIMOD  */
// Módulo de teclado com interrupção ativa e acionada por borda de descida
KBI1SC = 0x02;

KBI1SC_KBACK = 1;     // Limpa flag de teclado
```

A função mínima de interrupção para o módulo de teclado é apresentada abaixo.

```
interrupt void teclado(void){
  KBI1SC_KBACK = 1;                    // Limpa flag de teclado
}
```

Note que a função de interrupção apresentada acima não leva em consideração a existência de vibrações na chave. Como já foi comentado, para evitar os problemas relacionados com estas vibrações podemos utilizar recursos de proteção em software. As possíveis proteções em software são a utilização de um atraso (*delay*) para leitura do teclado (*delay* nunca é uma boa solução, pois o processador ficará parado, sem executar nenhuma instrução útil) ou a utilização do temporizador para agendar uma leitura de teclado após a ocorrência da interrupção. A temporização irá depender do tempo necessário para evitar as vibrações da chave, onde normalmente 1ms é suficiente. Caso os efeitos das vibrações persistam, pode-se aumentar este tempo.

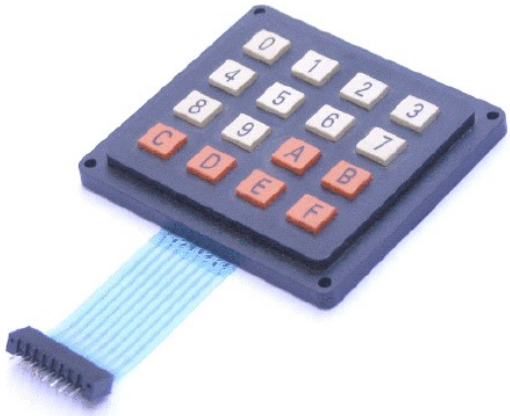
A figura abaixo apresenta uma sugestão de implementação do uso de um temporizador para leitura de teclado evitando os efeitos de vibrações na chave.



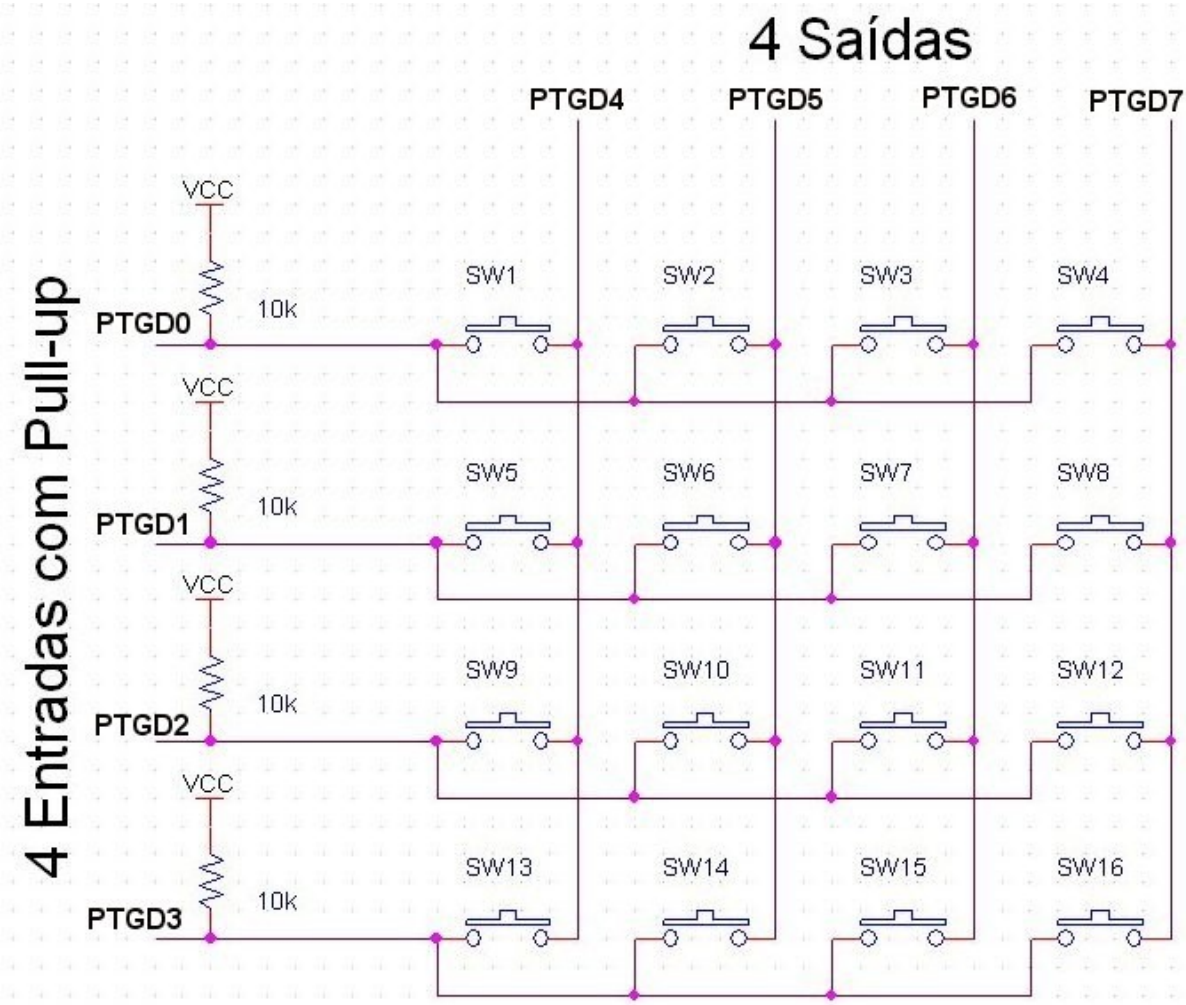
O código abaixo demonstra a implementação deste processo para o microcontrolador MC9S08AW60 utilizando uma variável de *flag* para agendamento de tarefa.

```
/* Interrupção de teclado */
interrupt void teclado(void){
  KBI1SC_KBACK = 1;                    // Limpa flag de teclado
  KBU1SC_KBIE = 0;                     // Desliga interrupção de teclado
  flag_teclado = 1;                    // Aciona a contagem do tempo de bounce
}


/* Interrupção de timer a cada 50us */
interrupt void timer(void){
  TPM1SC_TOF = 0                         // Limpa a flag do temporizador
  if(flag_teclado){
    i++;
  }
}

void main(void) {
  MCU_init();                 // Chama função de inicialização do ucontrolador

  for(;;)
  {
    if(i == 20){              // 20 * 50us = 1ms
      teclado = PTGD;         // Leitura dos pinos de teclado
      flag_teclado = 0;       // Desliga contagem do tempo de bounce
      i = 0;                  // Reinicia contagem do tempo de bounce
      KBI1SC_KBACK = 1;       // Limpa interrupções acionadas por vibrações
      KBI1SC_KBIE = 1;        // Habilita interrupção de teclado novamente
    }
  } /* laço infinito */
/* tenha certeza de que você nunca sairá do main do aplicativo desenvolvido */
}
```

Outra forma bastante comum de implementação de teclado em sistemas embarcados é o teclado matricial. Um teclado matricial de 4 linhas por 4 colunas permite a geração de 16 códigos independentes utilizando-se apenas 8 pinos de uma porta do microcontrolador. A figura abaixo apresenta um teclado matricial comercial.



Um exemplo de conexão interna de um teclado matricial com 16 chaves pode ser observado na figura abaixo. Note que para realizar a interface com o microcontrolador as colunas do teclado matricial são conectadas com pinos configurados como saídas e as linhas do teclado são conectadas a pinos configurados como entradas. Ainda, nos pinos configurados como entradas será utilizado *pull-up*.



Os pinos configurados como saídas são utilizados para realizar a varredura do teclado matricial. Esta varredura terá como tarefa identificar quando uma chave do teclado for pressionada. A varredura irá alternar os valores lógicos nos pinos configurados como saídas. Estes pinos sempre estarão configurados para que somente um deles contenha nível lógico zero em um determinado momento. Desta forma, quando uma chave for pressionada irá gerar um código bem definido, onde somente dois pinos da porta utilizada como teclado estarão em nível lógico zero (um pino de saída e um pino de entrada).

A tabela abaixo apresenta um exemplo de códigos de identificação de chaves em uma interface com um teclado matricial (a porta G do microcontrolador será utilizada para realizar a interface com o teclado).

Considere ainda para a geração desta tabela as figuras apresentadas anteriormente.

Pinos configurados como saída				Pinos configurados como entrada				Tecla
PTGD7	PTGD6	PTGD5	PTGD4	PTGD3	PTGD2	PTGD1	PTGD0	
1	1	1	0	1	1	1	0	**0**
1	1	0	1	1	1	1	0	**1**
1	0	1	1	1	1	1	0	**2**
0	1	1	1	1	1	1	0	**3**
1	1	1	0	1	1	0	1	**4**
1	1	0	1	1	1	0	1	**5**
1	0	1	1	1	1	0	1	**6**
0	1	1	1	1	1	0	1	**7**
1	1	1	0	1	0	1	1	**8**
1	1	0	1	1	0	1	1	**9**
1	0	1	1	1	0	1	1	**A**
0	1	1	1	1	0	1	1	**B**
1	1	1	0	0	1	1	1	**C**
1	1	0	1	0	1	1	1	**D**
1	0	1	1	0	1	1	1	**E**
0	1	1	1	0	1	1	1	**F**

Caso nenhuma chave for pressionada, o nível lógico dos pinos configurados como entrada (linhas) será um, devido ao *pull-up* utilizado. Este *pull-up* pode ser interno ao microcontrolador ou externo.

A partir da tabela apresentada podemos definir dois vetores de constantes: o vetor contendo os valores da varredura e o vetor que irá conter os valores relativos aos códigos de cada tecla.

Os vetores para este caso são apresentados abaixo:

```
  const byte varredura[4] =
    {0xE0,0xD0,0xB0,0x70};

  const byte codigo[16] =
     {0xEE,0xDE,0xBE,0x7E,0xED,0xDD,0xBD,0x7D,0xEB,0xDB,0xBB,0x7B,0xE7,0xD7,0xB7,
0x77};
```

O processo de varredura deverá ser inserido na interrupção do estouro de tempo do temporizador da seguinte forma:

```
interrupt void timer(void){
  (void)TPM1SC;
  TPM1SC_TOF = 0;                // Limpa a flag do estouro de tempo
  PTGD = varredura[ivarre];      // Altera o valor de varredura do teclado
  ivarre++;                      // Incrementa o índice de valores de varredura
  if (ivarre == 4)               // Retorna ao início da varredura
    ivarre = 0;
}
```

O módulo de tempo pode ser configurado para valores entre 20µs e 50ms, dependendo a sensibilidade desejada para o teclado.

Se uma tecla for pressionada e a varredura estiver posicionada com nível lógico 0 na coluna desta chave, irá ocorrer uma interrupção de teclado devido a borda de descida no pino de entrada conectado a chave. Dentro desta interrupção pode-se tomar medidas para análise de qual chave foi pressionada. O código abaixo apresenta uma das possíveis formas de análise do teclado para este caso.

```
interrupt void teclado(void){
  KBI1SC_KBACK = 1;              // Limpa flag de teclado
  tecla = PTGD;
  switch (tecla){                // Analisa qual tecla foi pressionada
    case codigo[0]:              // e escreve o valor na variável tecla
      tecla = 0;
      break;
    case codigo[1]:
      tecla = 1;
      break;
    case codigo[2]:
      tecla = 2;
      break;
    case codigo[3]:
      tecla = 3;
      break;
    case codigo[4]:
      tecla = 4;
      break;
    case codigo[5]:
      tecla = 5;
      break;
    case codigo[6]:
      tecla = 6;
      break;
    case codigo[7]:
      tecla = 7;
      break;
    case codigo[8]:
      tecla = 8;
      break;
    case codigo[9]:
      tecla = 9;
      break;
    case codigo[10]:
```

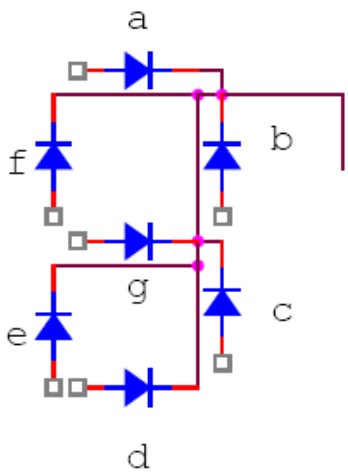
```
      tecla = 'A';
      break;
    case codigo[11]:
      tecla = 'B';
      break;
    case codigo[12]:
      tecla = 'C';
      break;
    case codigo[13]:
      tecla = 'D';
      break;
    case codigo[14]:
      tecla = 'E';
      break;
    case codigo[15]:
      tecla = 'F';
      break;
    default:
      tecla = 255;
      break;
  }
}
```

Devemos lembrar que este código não contempla a proteção em software para as vibrações na chave. Para que esta implementação seja funcional é necessário adicionar esta proteção ou pelo menos uma proteção em hardware.

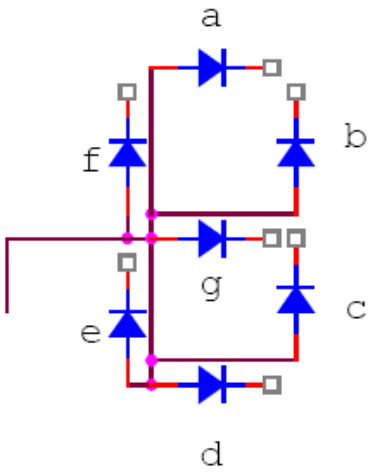
10. Interface com um Display 7 Segmentos

Um display de 7 segmentos é formado por 7 LEDs (a, b, c, d, e, f, g) que são previamente encapsulados e conectados de duas maneiras: anodo comum e catodo comum. A figura abaixo apresenta estas configurações.

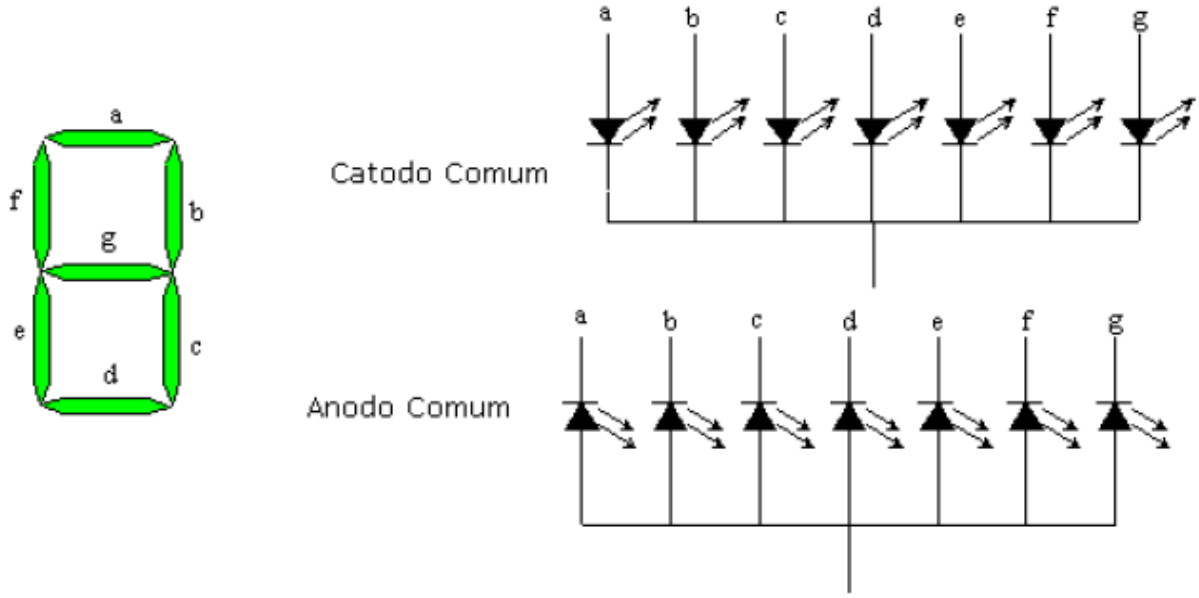
Catodo Comum:



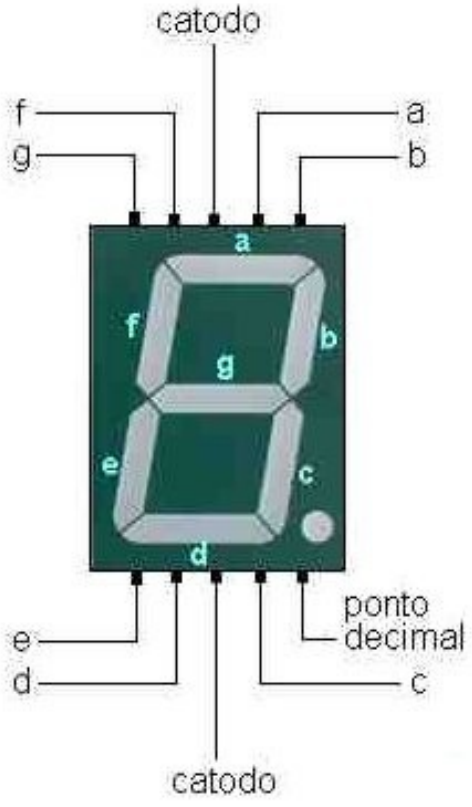
Anodo Comum:



No modo catodo comum acende-se cada LED conectando-se o barramento comum ao GND e aplicando-se o valor lógico 1 em cada segmento que se deseja acender. Já no modo anodo comum acende-se cada LED conectando-se o barramento comum ao VCC e aplicando-se o valor lógico 0 em cada segmento que se deseja acender. A figura abaixo apresenta de forma mais clara estas configurações.



Normalmente estes displays possuem um oitavo LED, que possibilita a representação de um ponto decimal. Esta configuração é apresentada na figura abaixo.



Para realizar a interface de um display de 7 segmentos com um microcontrolador deve-se determinar quais bits de uma porta serão utilizados para acionar os LEDs dos segmentos. Por exemplo, iremos definir a porta A como interface com o display. Desta forma tem-se a seguinte configuração:

Porta do microcontrolador	**PTAD7**	**PTAD6**	**PTAD5**	**PTAD4**	**PTAD3**	**PTAD2**	**PTAD1**	**PTAD0**
Segmentos	Ponto Decimal	a	b	c	d	e	f	g

A partir da definição da conexão com os pinos do microcontrolador pode-se criar a tabela de codificação para um display anodo ou catodo comum.

Tabela de Codificação para um display cadoto comum								
PTAD6	**PTAD5**	**PTAD4**	**PTAD3**	**PTAD2**	**PTAD1**	**PTAD0**	**Dígito**	**Código em Hexa**
1	1	1	1	1	1	0	0	7E
0	1	1	0	0	0	0	1	30
1	1	0	1	1	0	1	2	6D
1	1	1	1	0	0	1	3	79
0	1	1	0	0	1	1	4	33
1	0	1	1	0	1	1	5	5B
1	0	1	1	1	1	1	6	5F
1	1	1	0	0	0	0	7	70
1	1	1	1	1	1	1	8	7F
1	1	1	1	0	1	1	9	7B
1	1	1	0	1	1	1	A	77
0	0	1	1	1	1	1	b	1F
1	0	0	1	1	1	0	C	4E
0	1	1	1	1	0	1	d	3D
1	0	0	1	1	1	1	E	4F
1	0	0	0	1	1	1	F	47

OBS: Para construir uma tabela de codificação para um display anodo comum considerando este cenário devemos inverter os níveis lógicos apresentados na tabela acima. Ainda, o bit mais significativo da porta conectada ao display poderá ser utilizado para representar o ponto decimal. Sendo assim, se PTAD7 = 1 o ponto decimal estará ativo.

A partir do exposto notamos que não existe uma grande diferença entre um display anodo comum para um display catodo comum além da lógica de acionamento dos LEDs. No entanto, quando projeta-se este circuito percebe-se uma diferença importante. Em uma configuração catodo comum, o microcontrolador estará fornecendo corrente para

os LEDs do display, enquanto em uma configuração anodo comum o microcontrolador estará drenando corrente. Este fato é importante porque normalmente os pinos de entrada/saída de um microcontrolador possuem maior capacidade de dreno de corrente do que para fornecer corrente. Ainda, existe um limite máximo de fornecimento de corrente, considerando todos as portas do microcontrolador. No caso do MC9S08AW60 este limite é 100mA quando alimentado em 5V e 60mA quando alimentado em 3V. De posse destas informações pode-se chegar a conclusão que é recomendável utilizar displays do tipo anodo comum para interface com um microcontrolador.

O código que implementa a escrita no display LCD é extremamente simples, pois será simplesmente uma igualdade de uma das constantes apresentadas na tabela acima para a porta onde o display está conectado, como apresentado abaixo.

```
// Declaração das constantes do display
const byte codigo7seg[16] =
   {0x7E,0x30,0x6D,0x79,0x33,0x5B,0x5F,0x70,0x7F,0x7B,0x77,0x1F,0x4E,0x3D,0x4F,
0x47};
```

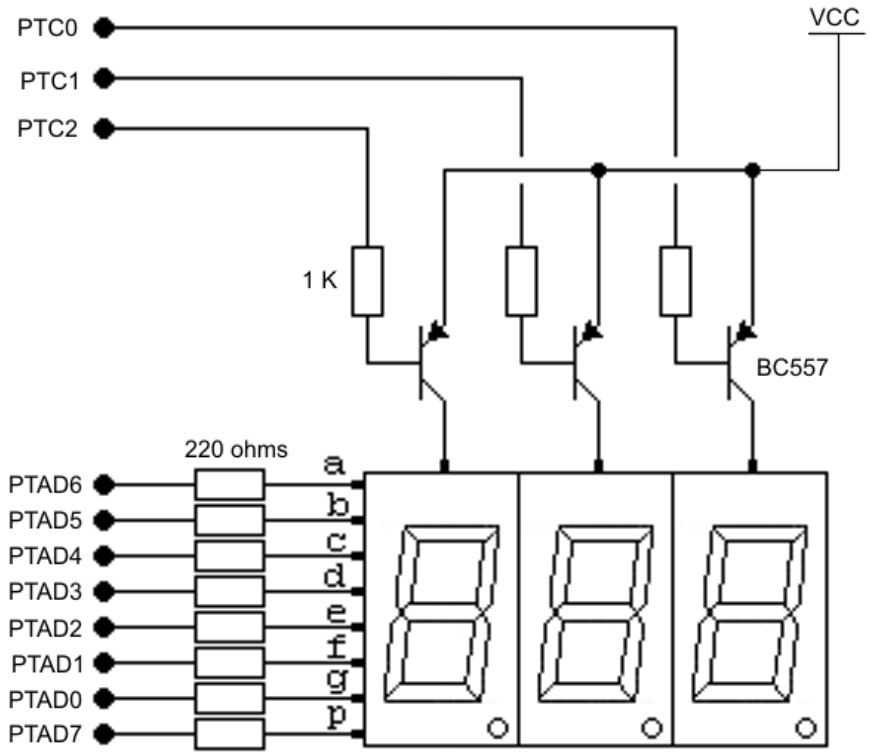
E para escrever no display:

```
// Escreve 0 no display 7 segmentos conectado a porta A
PTAD = codigo7seg[0];
```

A implementação da interface com um display de 7 segmentos para um dígito aparentemente é simples. No entanto, se for necessário conectar 2 ou mais dígitos a abordagem é um pouco diferente. Utilizar 1 porta de entrada/saída completa para cada dígito não é uma boa solução. Desta forma, precisamos encontrar formas mais econômicas em termos de pinos e portas para implementar está interface. Duas soluções bem conhecidas para estes casos podem ser utilizadas: multiplexação por latches ou multiplexação no tempo com transistores.

O problema de utilizar latches é que será necessário ainda utilizar um *driver* de corrente para alimentar os LEDs do display, sendo necessário 1 latch e um *driver* para cada dígito utilizado. A segunda solução é mais eficiente e reduzida em termos de componentes. Com a multiplexação no tempo uma porta irá manter o valor do código hexadecimal correspondente ao dígito a ser aceso e uma outra porta de entrada / saída indicará em qual dos displays será aceso o dígito equivalente. Portanto, deve ser realizado uma varredura do dígito menos significativo para o dígito mais significativo (ou vice-versa), controlada por uma outra porta, alterando-se o valor de cada dígito no tempo, através da porta conectada aos LEDs dos displays.

A seguir é apresentado uma possível interface de 3 dígitos de 7 segmentos anodo comum com o microcontrolador MC9S08AW60.



Na técnica de multiplexação do tempo temos somente um display ativo num dado momento, sendo os displays ativados e desativados em seqüência, um após o outro, em um ciclo que se repete a uma velocidade muito alta, de forma que o olho humano tem a ilusão de que todos estão acesos simultaneamente. Esta sensação de ilusão é obtida quando utilizarmos freqüências de varredura acima de 30Hz.

Exemplo: Define-se que a freqüência de varredura de 3 displays de 7 segmentos conectados à um microcontrolador será 65 Hz. De quanto em quanto tempo devemos alterar o valor do dígito a ser escrito?

$$Período = \frac{1}{65} = 15{,}38\,ms$$

Verificamos que o período total de varredura dos 3 dígitos deverá ser em torno de 15ms. Como temos 3 dígitos, devemos alterar o valor dos dígitos a cada **5ms**. Para que isto seja possível pode-se utilizar a interrupção de estouro de tempo configurada para este tempo. A cada entrada nesta interrupção deve-se alterar qual dígito será aceso e o valor do código hexadecimal que irá escrever o valor desejado no display.

Abaixo é apresentado o código que implementa a escrita de três dígitos numéricos em um conjunto de displays de 7 segmentos anodo comum, onde a porta A será utilizada para especificar os LEDs a serem acesos e a porta C realizará a varredura.

Note que como será utilizada a configuração anodo comum, os códigos apresentados na tabela anterior devem ser invertidos, pois nesta configuração os LEDs acendem com nível lógico 0.

```
// Declaração das constantes do display com ponto apagado
const byte codigo7seg[10] =
  {0x81, 0xCF, 0x92, 0x86, 0xCC, 0xA4, 0xA0, 0x8F, 0x80, 0x84};
```

Imagine agora que o valor a ser escrito foi previamente processado e que as variáveis dig1, dig2 e dig3 contém os valores numéricos a serem escritos nos displays (0 a 9). Devemos definir uma condição inicial no programa, por exemplo, o dígito mais a direita (dig3) aceso. Desta forma temos o código abaixo:

```
PTCD = 0x00;                // Inicialmente todos os displays apagados
PTAD = codigo7seg[dig3];    // Escreve os bits que irão acender os LEDs
PTCD_PTCD0 = 1;             // Acende o terceiro display
```

E na interrupção de estouro de tempo devemos implementar a lógica que irá controlar a varredura do display, como visto abaixo.

```
interrupt void timer(void){
  (void)TPM1SC;
  TPM1SC_TOF = 0;               // Limpa a flag do estouro de tempo
  if (PTCD_PTCD0 == 1){         // Se o terceiro digito estava aceso
    PTCD_PTCD0 = 0;             // apaga terceiro digito
    PTAD = codigo7seg[dig2];    // Copia o código do segundo digito
    PTCD_PTCD1 = 1;             // acende segundo dígito
  }else{                        // senão
    if (PTCD_PTCD1 == 1){       // ...
      PTCD_PTCD1 = 0;
      PTAD = codigo7seg[dig1];
      PTCD_PTCD2 = 1;
    }else{
      if (PTCD_PTCD2 == 1){
        PTCD_PTCD2 = 0;
        PTAD = codigo7seg[dig3];
        PTCD_PTCD0 = 1;
      }
    }
  }
}
```

11. Interface com um Display LCD

Neste capítulo será apresentado as formas de se realizar uma interface com um display LCD, bem como algumas funções em linguagem C para comunicação com displays LCD utilizando controladores KS0066 ou HD44780 (controladores de LCD mais difundidos no mercado). Estes controladores possuem 80 bytes de memória RAM para os dados do display (DDRAM), 64 bytes de RAM para o gerador de caracteres do usuário e 1240 bytes para a ROM do gerador de caracteres.

Os módulos LCD normalmente apresentam distribuição de pinos padronizada, com um conector de 16 pinos, cujas funções encontram-se na tabela abaixo.

Pino	Nome	Função
1	V_{SS}	Referência de terra
2	V_{DD}	Alimentação (+5V)
3	V_O	Tensão de contraste
4	RS	Seleção entre modo de comando (0) ou dados (1)
5	$R/\overline{w}$	Seleção entre escrita (0) ou leitura (1)
6	E	Habilitação – um pulso de no mínimo 450ns irá permitir que o módulo receba comandos ou dados ou ainda enviar dados
7	DB0	Bit 0 do dado / comando. Não utilizado no modo de 4 bits
8	DB1	Bit 1 do dado / comando. Não utilizado no modo de 4 bits
9	DB2	Bit 2 do dado / comando. Não utilizado no modo de 4 bits
10	DB3	Bit 3 do dado / comando. Não utilizado no modo de 4 bits
11	DB4	Bit 4 do dado / comando
12	DB5	Bit 5 do dado / comando
13	DB6	Bit 6 do dado / comando
14	DB7	Bit 7 do dado / comando
15	A	Anodo do LED de *backlight*
16	K	Catodo do LED de *backlight*

A DDRAM é organizada de forma que os endereços de 0x00 a 0x27 armazenem os dados da primeira linha e de 0x40 a 0x67, os dados da segunda linha. É claro que desses 40 bytes de cada linha, apenas 16 em cada uma são visíveis (em um display de 16 caracteres). A figura abaixo apresenta a organização da DDRAM.

Linha	Endereço																																							
1	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27
2	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	61	62	63	64	65	66	67

No modo normal (sem deslocamento da imagem) o primeiro caractere da primeira linha é o endereço 0x00, o segundo o endereço 0x01 e assim por diante.

A CGRAM é uma área de memória dedicada à construção de caracteres pelo próprio usuário. Nos 64 bytes da CGRAM é possível a criação de 8 caracteres de 8x5 ou 4 de 10x5. Esses caracteres do usuário podem ser selecionados pelos códigos da primeira coluna da tabela abaixo.

A ROM do gerador de caracteres contém uma tabela de 208 caracteres 8x5 e 32 caracteres 10x5, conforme pode ser visto abaixo (a tabela a seguir representa o conjunto de caracteres da ROM de código A00).

Para utilizar um módulo LCD é necessário conhecer o seu hardware. As informações descritas neste documento são baseadas na utilização de um módulo LCD de duas linhas por 16 caracteres, com controlador KS0066.

Esses módulos podem trabalhar em dois modos distintos de barramento de comunicação: 4 ou 8 bits. No modo de 8 bits, os dados e comandos são lidos e escritos pelo barramento de 8 bits do módulo (pinos DB0 a DB7). No modo de 4 bits, utilizam-se apenas os pinos DB4 a DB7 e a informação é enviada em duas etapas.

Para utilizar o módulo em modo 4 ou 8 bits é necessário utilizar uma seqüência de inicialização determinada pelo fabricante, conforme será visto mais tarde. No entanto, antes de apresentar os modos de inicialização é necessário conhecer os tipos de comandos disponíveis no módulo e a sua forma de utilização.

A tabela abaixo apresenta os comandos possíveis para os módulos com controladores KS0066 ou HD44780.

Comando	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Descrição	Duração
Apaga Display	0	0	0	0	0	0	0	0	0	1	Apaga o Display e retorna o cursor para a primeira posição da primeira linha.	---
Retorno	0	0	0	0	0	0	0	0	1	X	Retorna o cursor para a primeira posição da primeira linha e retorna a mensagem ao formato original caso estivesse deslocada.	1,52ms
Configura Modo	0	0	0	0	0	0	0	1	I / D	S	I / D – Configura o sentido de deslocamento do display: 1 – movimenta o cursor à direita para cada caractere escrito; 0 – movimenta o cursor à esquerda para cada caractere escrito; S – ativa (1) ou desativa (0) o deslocamento da mensagem a cada leitura da DDRAM.	37µs
Controle Liga / Desliga do Display	0	0	0	0	0	0	1	D	C	B	Liga (D = 1) ou desliga (D = 0) o display. Ativa (C = 1) ou desativa (C = 0) o cursor. Ativa (B = 1) ou desativa (B = 0) o cursor piscante.	37µs
Deslocamento	0	0	0	0	0	1	S / C	R / L	X	X	S / C – seleciona entre deslocamento do cursor (0) ou deslocamento do cursor + mensagem (1) R / L – sentido de deslocamento: direita (1) ou esquerda (0).	37µs
Configuração do Display	0	0	0	0	1	DL	N	F	X	X	DL – número de bits do barramento: 8 bits (1) ou 4 bits (0). N – número de linhas: 2 linhas (1) ou 1 linha (0). F – tamanho dos caracteres: 10x5 (1) ou 8x5 (0).	37µs
Endereça a CGRAM	0	0	0	1	A	A	A	A	A	A	Especifica um endereço (6 bits) de acesso à CGRAM.	37µs
Endereça a DDRAM	0	0	1	A	A	A	A	A	A	A	Especifica um endereço (7 bits) de acesso a DDRAM.	37µs
Lê contador de endereços e Busy Flag	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Lê o valor do contador de endereços (AC) e estado de flag BF: BF = 1, controlador ocupado BF = 0, controlador livre	37µs
Escreve dado na CGRAM ou DDRAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Escreve um dado (D) na posição indicada pelo registrador AC da memória CGRAM ou DDRAM.	37µs
Lê um dado da CGRAM ou DDRAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Lê um dado (D) da posição de memória (CGRAM ou DDRAM) endereçada pelo registrador AC.	37µs

Repare que o envio de um comando deve obedecer aos seguintes passos:

1. O dado / comando é colocado no barramento de dados;
2. a linha RS é configurada para nível “1” caso seja um dado ou “0” caso seja um comando;
3. a linha E gera um pulso de pelo menos 450ns.

A seqüência necessária para inicialização do módulo no modo de comunicação em 8 bits é:

1. Envia-se o comando de configuração do display;
2. envia-se o comando de configuração liga / desliga do display;
3. envia-se o comando de configuração do modo de operação;
4. envia-se o comando para apagar qualquer mensagem no display e posicionar o cursor no início da primeira linha.

Observe que os três últimos passos são opcionais.

A seqüência para inicialização do módulo no modo de comunicação em 4 bits, com as modificações necessárias, é apresentada a seguir.

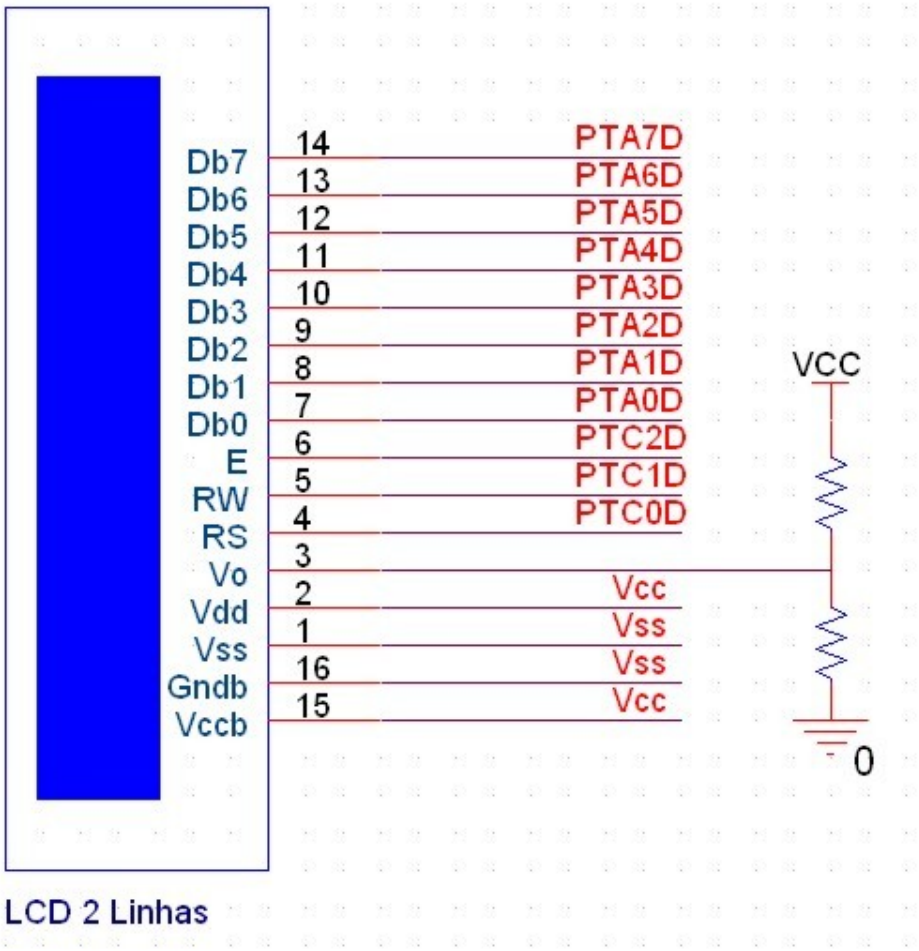
1. Envia-se um comando de quadro bits contendo o valor 3 (0011 binário) por meio das linhas DB4 a DB7;
2. aguarda-se aproximadamente 5ms;
3. envia-se novamente o comando do item 1;
4. aguarda-se aproximadamente 100µs;
5. envia-se novamente o comando do item 1;
6. envia-se o comando 2 (0010 binário). Desse ponto em diante o módulo passa a trabalhar no modo de 4 bits e devemos enviar cada byte em duas metades de 4 bits (conhecidas por *nibble*), iniciando pelos 4 bits mais significativos;
7. envia-se o comando de configuração de display;
8. envia-se o comando de configuração liga / desliga do display;
9. envia-se o comando de configuração do modo de operação;
10. envia-se o comando para apagar qualquer mensagem no display e posicionar o cursor no início da primeira linha.

Configurado o módulo podemos escrever os dados nele a qualquer momento. Basta enviar os dados com a linha RS em nível lógico alto.

A tabela abaixo apresenta alguns dos principais comandos interpretados pelos módulos LCD com controladores KS0066 ou HD44780 em formato hexadecimal.

Configuração do Display	
1 linha com caracteres 5x7 (8 bits)	**0x30**
2 linhas com caracteres 5x7 (8 bits)	**0x38**
1 linha com caracteres 5x10 (8 bits)	**0x34**
1 linha com caracteres 5x7 (4 bits)	**0x20**
2 linhas com caracteres 5x7 (4 bits)	**0x28**
1 linha com caracteres 5x10 (4 bits)	**0x24**
Controle Ativo / Inativo do Display	
Display aceso com cursor fixo	**0x0E**
Display aceso com cursor intermitente	**0x0F**
Display aceso sem o cursor	**0x0C**
Display apagado	**0x08**
Configuração do modo de operação	
Escreve deslocando a mensagem a esquerda	**0x07**
Escreve deslocando a mensagem a direita	**0x05**
Escreve deslocando o cursor a direita	**0x06**
Escreve deslocando o cursor a esquerda	**0x04**
Comandos Úteis	
Limpa display e retorna o cursor	**0x01**
Retorna o cursor (sem alterar a DDRAM)	**0x02**
Desloca somente o cursor a direita	**0x14**
Desloca somente o cursor a esquerda	**0x10**
Desloca cursor + mensagem a direita	**0x1C**
Desloca cursor + mensagem a esquerda	**0x18**
Desloca cursor para primeira linha, primeira posição	**0x80**
Desloca cursor para segunda linha, primeira posição	**0xC0**
Endereço da CGRAM (caracteres especiais)	
Para configurar (endereço inicial)	**0x40**
Para escrever primeiro caractere	**0x00**
Para escrever último caractere	**0x70**

Para realizar a interface com o microcontrolador MC9S08AW60 em 8 bits iremos utilizar a porta A como barramento de dados e a porta C como bits de controle (E, RS e RW). A figura a seguir apresenta a configuração de hardware para interface entre o microcontrolador e um módulo LCD.



A seguir são apresentadas algumas funções em linguagem C para comunicação do microcontrolador MC9S08AW60 com um módulo LCD baseadas no hardware exposto acima.

```
#include "derivative.h" /* include peripheral declarations */


// Definições para escolha dos pinos da porta do microcontrolador
#define RS PTCD_PTCD0
#define RW PTCD_PTCD0
#define E PTCD_PTCD0
#define dados PTAD
#define direcao PTADD
#define busy_flag PTAD_PTAD7


// Declaração do protótipo da função de teste da busy flag
void testa_bit(void);
```

```
// Função para escrita de uma frase no display
void write_lcd(char *string){

while(*string){
  testa_bit();
  RS = 1;
  RW = 0;
  dados = *string;
  E = 1;
  asm{                    // Pulso de 450ns na linha E
    nop
    nop
    nop
  }
  E = 0;
  string++;
}
}


// Função para escrita de um número de 2 digitos no display
void write_numero(unsigned char numero){
unsigned char i = 0;

  testa_bit();
  RS = 1;
  RW = 0;
  i = (numero / 10);
  dados = 48 + i;
  E = 1;
  asm{
    nop
    nop
    nop
  }
  E = 0;

  testa_bit();
  RS = 1;
  RW = 0;
  dados = 48 + (numero - (i * 10));
  E = 1;
  asm{
    nop
    nop
    nop
  }
  E = 0;
}

// Envia um comando para o módulo do display
void instr(char comando){
  testa_bit();
  RS = 0;
  RW = 0;
  dados = comando;
  E = 1;
  asm{
    nop
    nop
    nop
  }
  E = 0;
}
```

```
// Envia um caractere para o módulo do display
void caractere(char dado){
  testa_bit();
  RS = 1;
  RW = 0;
  dados = dado;
  E = 1;
  asm{
    nop
    nop
    nop
  }
  E = 0;
}


// Inicialização do display para 2 linhas com caracteres de 5x7
// Deslocamento do cursor a direita com cursor apagado
void init_lcd(void){
  instr(0x38);
  instr(0x38);
  instr(0x06);
  instr(0x0c);
  instr(0x01);
}


// Função de posicionamento do cursor do display
void posiciona(byte linha, byte coluna){
  testa_bit();
  RS = 0;
  RW = 0;
  switch(linha){
    case 1:
      coluna--;
      dados = 0x80 + coluna;
      break;
    case 2:
      coluna--;
      dados = 0xC0 + coluna;
      break;
    default:
      return;
  }
  E = 1;
  asm{
    nop
    nop
    nop
  }
  E = 0;
}
```

```
// Função para limpar o display e retornar a primeira posição da primeira linha
void limpa_lcd(void){
  testa_bit();
  RS = 0;
  RW = 0;
  dados = 0x01;
  E = 1;
  asm{
    nop
    nop
    nop
  }
  E = 0;
}


// Função de teste da busy flag
void testa_bit(void){
  RS = 0;
  RW = 1;
  direcao = 0x00;
  E = 1;
  while(busy_flag);
  E = 0;
  RW = 0;
  direcao = 0xFF;
}
```

É interessante criar um arquivo .h com os protótipos das funções para utilizar o código como biblioteca através de um *include*, como visto abaixo.

```
void write_lcd(char *string);
void write_numero(unsigned char numero);
void instr(char comando);
void init_lcd(void);
void posiciona(byte linha, byte coluna);
void limpa_lcd(void);
void testa_bit(void);
```

Para utilizarmos as funções criadas incluímos o arquivo .h, por exemplo, lcd.h no programa principal.

```
#include "lcd.h"
```

Desta forma as funções estarão disponível para o programa principal, como mostrado abaixo:

```
  //Inicializa o módulo LCD
  init_lcd();
  // Escreve "Teste de Escrita" na primeira posição da primeira linha
  write_lcd("Teste de Escrita");
  // Posiciona o cursor na primeira posição da segunda linha
  posiciona(2,1);
  // Escreve "MC9S08AW60" na primeira posição da segunda linha
  write_lcd("MC9S08AW60");
  // Escreve o caractere A
  caractere('A');
```

Módulos LCD de 4 Linhas por 20 Caracteres

Em módulos LCD's 4x20 (4 linhas x 20 caracteres) utilizando o controlador KS0066, poucas características de operação se modificam, tanto em hardware como no software.

O hardware permanece quase o mesmo. Os pinos de dados e controle do módulo são iguais aos dos módulos de 2 linhas por 16 caracteres apresentados acima. No entanto, os módulos LCD 4x20 possuem dois controladores, cada um responsável por um par de linhas. O software sofre algumas alterações na inicialização do módulo e nos endereços da DD RAM para posicionamento dos caracteres.

A figura abaixo apresenta a organização da DDRAM dos módulos LCD 4x20 que utilizam o controlador KS0066.

Linha	Endereço																			
1	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
2	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53
3	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27
4	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	61	62	63	64	65	66	67

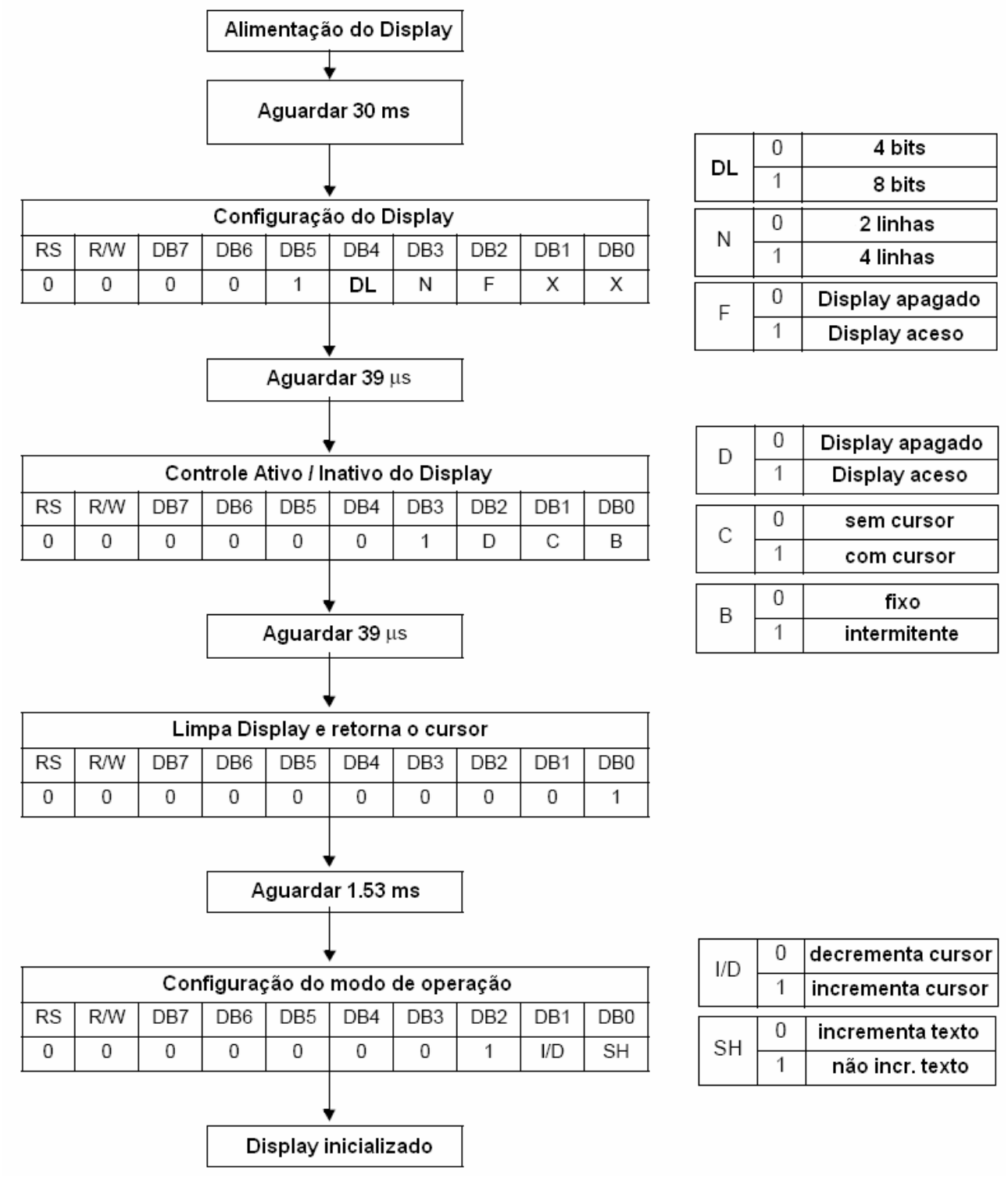
Obs: Verifique que no comando para posicionar o cursor na DD RAM utiliza o bit mais significativo em nível lógico 1, ou seja, o software desenvolvido deve enviar os comandos como indicado abaixo para o correto posicionamento.

LCD	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
linha 1	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F	90	91	92	93
linha 2	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	D0	D1	D2	D3
linha 3	94	95	96	97	98	99	9A	9B	9C	9E	9D	9F	A0	A1	A2	A3	A4	A5	A6	A7
linha 4	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF	E0	E1	E2	E3	E4	E5	E6	E7

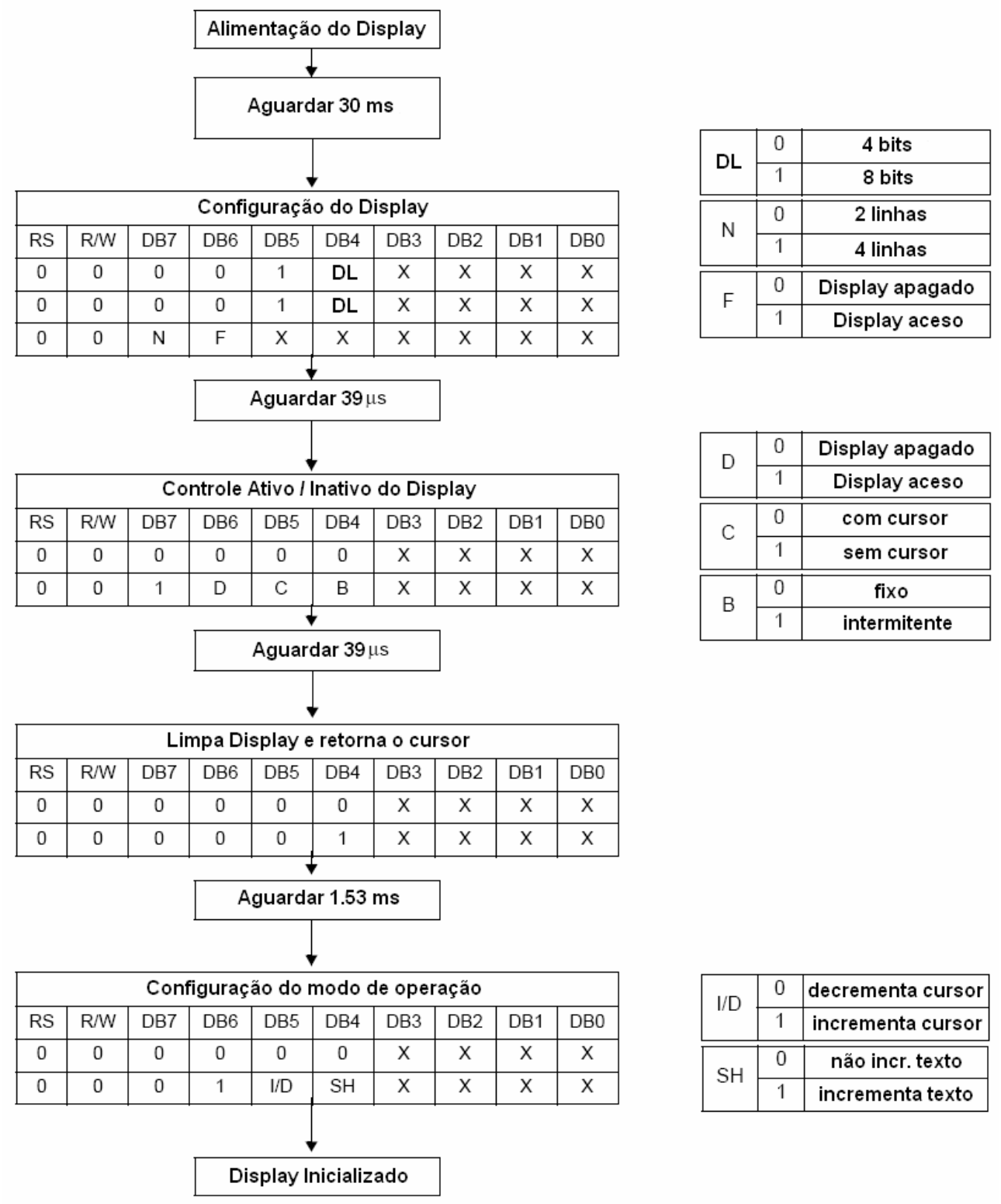
A inicialização dos módulos com controlador KS0066 com comunicação em 8 bits deve seguir a seqüência abaixo:

1. Envia-se o comando de configuração do display;
2. Envia-se o comando de configuração liga/ desliga do display;
3. Envia-se o comando para apagar qualquer mensagem no display e posicionar o cursor no inicio da primeira linha;
4. Envia-se o comando de configuração do modo de operação.

O fluxograma da seqüência apresenta acima com os tempos de processamento relativos a cada comando é apresentado na figura abaixo.



Se o modo de comunicação de 4 bits (alterando valor do bit DL) for utilizado, os passos serão os mesmos, e os dados terão que ser enviados em *nibbles*. como no diagrama de blocos abaixo.



Devido a estas diferenças de operação algumas funções apresentadas anteriormente irão sofrer modificações. Estas são as funções de posicionamento (pois agora possuímos mais caracteres por linha e mais linhas) e de inicialização. A seguir estas modificações são apresentadas.

```
// Inicialização do display para 4 linhas com caracteres de 5x7
// Deslocamento do cursor a direita com cursor apagado
void init_lcd(void){
  instr(0x3C);
  instr(0x0C);
  instr(0x01);
  instr(0x06);
}


// Função de posicionamento do cursor do display 4x20
void posiciona(byte linha, byte coluna){
  testa_bit();
  RS = 0;
  RW = 0;
  switch(linha){
    case 1:
      coluna--;
      dados = 0x80 + coluna;
      break;
    case 2:
      coluna--;
      dados = 0xC0 + coluna;
      break;
    case 2:
      coluna--;
      dados = 0x94 + coluna;
      break;
    case 2:
      coluna--;
      dados = 0xD4 + coluna;
      break;
    default:
      return;
  }
  E = 1;
  asm{
    nop
    nop
    nop
  }
  E = 0;
}
```

12. Acesso a Memória FLASH do Microcontrolador

Os microcontroladores HCS08 possuem um circuito de controle que automatiza as operações de escrita e apagamento da memória FLASH.

O controlador utiliza um circuito elevador de tensão interno, capaz de gerar a tensão necessária à programação e apagamento das células de memória. Inclui ainda um circuito capaz de dividir a freqüência de barramento por um fator programável, de forma que a freqüência resultante (FCLK) esteja situada na faixa entre 150 e 200kHz, que é a região de trabalho do controlador da FLASH.

Quanto maior a freqüência de operação do controlador, mais rápida será a programação e apagamento da memória, entretanto, a operação fora da faixa de freqüências especificada não é garantida pelo fabricante.

A tabela abaixo apresenta os tempos típicos para a execução dos comandos de programação e apagamento da FLASH.

Comando	Ciclos (FCLK)	Tempo para FCLK = 200 kHz
Programação de byte	9	45µs
Programação de múltiplos bytes	4	20µs
Apagamento de página	4000	20ms
Apagamento completo	20000	100ms

A freqüência de operação do controlador (FCLK) pode ser calculada utilizando a seguinte equação:

$$FCLK = \frac{BUSCLK}{(8 * bPRDIV8) * (bDIV + 1)}$$

Sendo: bPRDIV8 – bit de pré-divisão por 8 (registrador FCDIV)

bDIV – fator de divisão do clock (registrador FCDIV)

O controlador de memória opera de forma muito simples: o usuário programa o divisor de clock do controlador (registrador FCDIV), escreve o endereço o qual deseja programar ou apagar, escreve o comando a ser executado no registrador FCMD e em seguida apaga o bit FCBEF (registrador FSTAT), dando início à execução do comando. Para apagar o bit FCBEF deve-se escrever o valor 1 nele. Após a execução do comando, o controlador seta o bit FCCF (registrador FSTAT).

Observe que antes da execução de um comando é necessário verificar o bit FCBEF. Um novo comando só pode ser escrito no registrador FCMD quando o bit FCBEF está em nível lógico 1. Caso contrário, um erro de acesso à memória FLASH é gerado.

OBS: Não é possível programar ou apagar a memória FLASH através de um programa executado na própria FLASH. Estas operações somente podem ser realizadas a partir de um programa sendo executado na memória RAM. Os exemplos apresentados neste capítulo demonstram uma das possíveis formas de se realizar tal operação.

Comandos do Controlador

O controlador reconhece cinco comandos diferentes: verificação de apagamento, programação de um byte, programação de múltiplos bytes, apagamento de página e apagamento completo.

Verificação de Apagamento

Este comando efetua a leitura de todo o conteúdo da memória FLASH e caso todos os bits estejam em nível lógico 1 (não programados), o bit FBLANK (registrador FSTAT) é levado para nível lógico 1. Caso a memória não esteja apagada, este bit é levado a nível lógico 0.

Um outro efeito da execução do comando de verificação de apagamento é que, caso a memória esteja apagada, o controlador automaticamente desabilita a proteção de acesso à memória, escrevendo os valores 1 e 0 nos bits SEC01 e SEC00 respectivamente, ambos localizados no registrador FOPT.

Programação de um Byte

Esse comando pode ser utilizado para programar uma posição qualquer da memória FLASH.

A utilização deste comando deve seguir os passos descritos a seguir:

1. O programa verifica se a *flag* de erro de acesso à memória FLASH (bit FACCERR no registrador FSTAT) está em nível lógico 1. Qualquer operação do controlador da memória é inibida quando este bit está ativado. O apagamento é realizado pela escrita de nível lógico 1 neste bit;

2. programa-se o fator de divisão do *clock* do controlador pelo registrador FCDIV (esta programação somente pode ser realizada uma única vez, após o *reset*;
3. o programa deve verificar se a *flag* FCBEF está em nível lógico 1. Caso ele esteja em nível lógico 0, o programa deve aguardar até que a execução do comando atual termine;
4. escreve-se o dado a ser programado no endereço desejado da memória FLASH. Lembre-se que pode ser necessário apagar a memória antes de escrever nela;
5. o comando de programação de byte (0x20) é escrito no registrador FCMD;
6. a *flag* FCBEF deve ser levada a nível lógico 0, escrevendo 1 neste bit. Este procedimento inicia a execução do comando;
7. após esta seqüência, o programa deve verificar se houve algum tipo de erro: violação de acesso à memória FLASH (FACCERR) ou violação de proteção (FPVIOL). Repare que o erro de violação de proteção ocorre quando se tenta programar uma área de memória que esteja dentro da área de proteção definida pelo registrador FPROT;
8. Aguarda-se o término da execução do comando pelo monitoramento da *flag* FCCF (registrador FSTAT). Ele é automaticamente apagado no início da execução do comando e levado a nível lógico 1 ao término da execução do comando. Lembre-se de que o programa deve aguardar um período mínimo de quatro ciclos de barramento antes de efetuar a leitura da *flag* FCCF ou FCBEF.

OBS: É recomendado que após a memória FLASH ser apagada, somente seja realizada uma única gravação em cada posição de memória de uma página. Novas gravações somente devem ser realizadas mediante novo apagamento da página. A programação de uma posição de memória mais de uma vez após seu apagamento pode corromper o conteúdo da memória FLASH.

Programação de Múltiplos Bytes

Este modo, denominado de rajada (*burst*), caracteriza-se por não desligar o circuito elevador de tensão entre a programação de cada byte. Desta forma, o tempo de programação da memória é reduzido consideravelmente.

A utilização do comando de programação de múltiplos bytes é limitada, pois somente é possível programar bytes situados em uma mesma linha. Uma linha é composta por 64 bytes da memória FLASH em que os bits de endereço A15 até A6 são iguais, conforme exemplificado na figura abaixo.

	Endereço	Bits do barramento de endereços (interno)															
		A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Linha da FLASH	0xFFFF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	a																
	0xFFC0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
Linha da FLASH	0xFFBF	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
	a																
	0xFF80	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
Linha da FLASH	0xFF7F	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
	a																
	0xFF40	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0

Os passos a serem seguidos para efetuar a programação de múltiplos bytes da FLASH são basicamente os mesmos da programação de um único byte, com exceção do comando a ser escrito no registrador FCMD (que neste caso é 0x25). Além disso, após a programação de cada byte, o programa deve retornar ao passo 3 (da seqüência para escrita de um byte) para efetuar a programação de um novo byte, repetindo o processo por toda a linha da FLASH, ou enquanto houver dados a serem gravados.

Após o último byte, o programa deve monitorar a *flag* FCCF, aguardando a conclusão do comando de programação.

Apagamento de Página

Os microcontroladores HCS08 dividem a memória FLASH em segmentos de 512 bytes, chamados de páginas. Uma página é a menor porção de memória que pode ser apagada.

Para efetuar o apagamento de uma página, basta seguir os mesmos procedimentos descritos para o comando de programação de byte. O endereço em que deve ocorrer a escrita seleciona a página a ser apagada. O restante da seqüência é idêntico, com exceção do comando, que neste caso é 0x40.

Após a operação de apagamento, as posições de memória conterão o valor 0xFF (todos os bits em nível lógico 1).

Apagamento Completo

Este comando apagará toda a memória FLASH. A execução do comando é idêntica à do apagamento de página, com exceção do comando a ser carregado no FCMD, que neste caso é 0x41.

Proteção da Memória FLASH

Os microcontroladores HCS08 incluem um mecanismo de proteção contra gravação ou apagamento da memória FLASH. Esse mecanismo é controlado pelo registrador FPROT. Este registrador não pode ser alterado pelo usuário e o seu conteúdo é carregado no *reset*, a partir do registrador NVPROT, localizado na memória FLASH.

A principal aplicação da proteção da memória é impedir que um programa consiga apagar uma parte de si mesmo inadvertidamente. Isto pode acontecer quando se utiliza a FLASH para o armazenamento de dados ou parâmetros reprogramáveis pelo usuário.

Outra aplicação é a proteção do código de um *bootloader*, um pequeno programa que pode ser utilizado para reprogramar a memória FLASH do microcontrolador. A utilização de um *bootloader* permite atualizações ou modificações em um programa principal residente na memória FLASH, utilizando uma simples conexão serial com um computador.

Na família de microcontroladores MC9S08AW, o registrador FPROT possui sete bits responsáveis pela seleção da área de memória protegida. Os bits FPS7 a FPS1 (bits mais significativos do endereço) concatenados com o valor 0x1FF determinam o fim da memória FLASH desprotegida. A tabela abaixo apresenta possível valores de proteção para a memória.

FPS7	FPS6	FPS5	FPS4	FPS3	FPS2	FPS1	Faixa de endereços protegidos	Tamanho total do bloco protegido	Endereço dos vetores redirecionados
0	0	0	1	1	0	0	0x1A00 a 0xFFFF	58880 bytes	0x19C0 a 0x19FD
0	0	0	1	1	0	1	0x1C00 a 0xFFFF	58368 bytes	0x1BC0 a 0x1BFD
0	0	0	1	1	1	0	0x1E00 a 0xFFFF	57856 bytes	0x1DC0 a 0x1DFD
0	0	0	1	1	1	1	0x2000 a 0xFFFF	57344 bytes	0x1FC0 a 0x1FFD
0	0	1	0	0	0	0	0x2200 a 0xFFFF	56832 bytes	0x21C0 a 0x21FD
0	0	1	0	0	0	1	0x2400 a 0xFFFF	56320 bytes	0x23C0 a 0x23FD
0	0	1	0	0	1	0	0x2600 a 0xFFFF	55808 bytes	0x25C0 a 0x25FD
0	0	1	0	0	1	1	0x2800 a 0xFFFF	55296 bytes	0x27C0 a 0x27FD

0	0	1	0	1	0	0	0x2A00 a 0xFFFF	54784 bytes	0x29C0 a 0x29FD
0	0	1	0	1	0	1	0x2C00 a 0xFFFF	54272 bytes	0x2BC0 a 0x2BFD
0	0	1	0	1	1	0	0x2E00 a 0xFFFF	53760 bytes	0x2DC0 a 0x2DFD
0	0	1	0	1	1	1	0x3000 a 0xFFFF	53248 bytes	0x2FC0 a 0x2FFD
0	0	1	1	0	0	0	0x3200 a 0xFFFF	52736 bytes	0x31C0 a 0x31FD
0	0	1	1	0	0	1	0x3400 a 0xFFFF	52224 bytes	0x33C0 a 0x33FD
0	0	1	1	0	1	0	0x3600 a 0xFFFF	51712 bytes	0x35C0 a 0x35FD
0	0	1	1	0	1	1	0x3800 a 0xFFFF	51200 bytes	0x37C0 a 0x37FD
0	0	1	1	1	0	0	0x3A00 a 0xFFFF	50688 bytes	0x39C0 a 0x39FD
0	0	1	1	1	0	1	0x3C00 a 0xFFFF	50176 bytes	0x3BC0 a 0x3BFD
0	0	1	1	1	1	0	0x3E00 a 0xFFFF	49664 bytes	0x3DC0 a 0x3DFD
0	0	1	1	1	1	1	0x4000 a 0xFFFF	49152 bytes	0x3FC0 a 0x3FFD
0	1	0	0	0	0	0	0x4200 a 0xFFFF	48640 bytes	0x41C0 a 0x41FD
0	1	0	0	0	0	1	0x4400 a 0xFFFF	48128 bytes	0x43C0 a 0x43FD
0	1	0	0	0	1	0	0x4600 a 0xFFFF	47616 bytes	0x45C0 a 0x45FD
0	1	0	0	0	1	1	0x4800 a 0xFFFF	47104 bytes	0x47C0 a 0x47FD
0	1	0	0	1	0	0	0x4A00 a 0xFFFF	46592 bytes	0x49C0 a 0x49FD
0	1	0	0	1	0	1	0x4C00 a 0xFFFF	46080 bytes	0x4BC0 a 0x4BFD
0	1	0	0	1	1	0	0x4E00 a 0xFFFF	45568 bytes	0x4DC0 a 0x4DFD
0	1	0	0	1	1	1	0x5000 a 0xFFFF	45056 bytes	0x4FC0 a 0x4FFD
....	....	....	....	....	....	....	....	....	....
....	....	....	....	....	....	....	....	....	....
0	1	1	1	1	1	1	0x8000 a 0xFFFF	32768 bytes	0x7FC0 a 0x7FFD
1	0	1	1	1	1	1	0xC000 a 0xFFFF	16384 bytes	0xBFC0 a 0xBFFD
1	1	0	1	1	1	1	0xE000 a 0xFFFF	8192 bytes	0xDFC0 a 0xDFFD
1	1	1	0	1	1	1	0xF000 a 0xFFFF	4096 bytes	0xEFC0 a 0xEFFD
1	1	1	1	0	1	1	0xF800 a 0xFFFF	2048 bytes	0xF7C0 a 0xF7FD
1	1	1	1	1	0	1	0xFC00 a 0xFFFF	1024 bytes	0xFBC0 a 0xFBFD
1	1	1	1	1	1	0	0xFE00 a 0xFFFF	512 bytes	0xFDC0 a 0xFDFD

O registrador FPROT inclui também um bit de controle:

FPDIS *(FLASH Protection Disable)*

0 – ativa a proteção da memória

1 – desabilita a proteção da memória

O registrador FPROT não pode ser alterado diretamente pelo usuário (a não ser via BDM), o que significa que o valor deve ser programado no registrador NVPROT. Em um *reset*, o conteúdo de NVPROT é copiado para FPROT.

O registrador FPROT através do bit FPDIS pode desabilitar totalmente a proteção contra gravação e apagamento da memória FLASH.

Redirecionamento dos Vetores de Interrupção

Outro mecanismo interessante é o de redirecionamento dos vetores de interrupção. Esse mecanismo é controlado pelo bit FNORED (registrador FOPT) e quando habilitado, redireciona os vetores de interrupção, dos endereços 0xFFC0 a 0xFFFF, para o último bloco de 64 bytes da memória desprotegida. A tabela anterior apresenta os endereços de redirecionamento para as faixas de proteção indicadas.

A faixa de endereços para os quais os vetores são direcionados deve ser calculada manualmente. Repare que o mecanismo de redirecionamento de vetores não inclui o vetor de *reset*, que é mantido fixo no seu endereço (0xFFFE a 0xFFFF). Além disto, o mecanismo de redirecionamento não pode ser utilizado quando a proteção de memória está desativada.

Segurança da Memória

Para prevenir a leitura não autorizada do conteúdo de toda a memória (FLASH e RAM), os microcontroladores HCS08 incluem também um mecanismo de segurança que impede o acesso à memória via interface BDM.

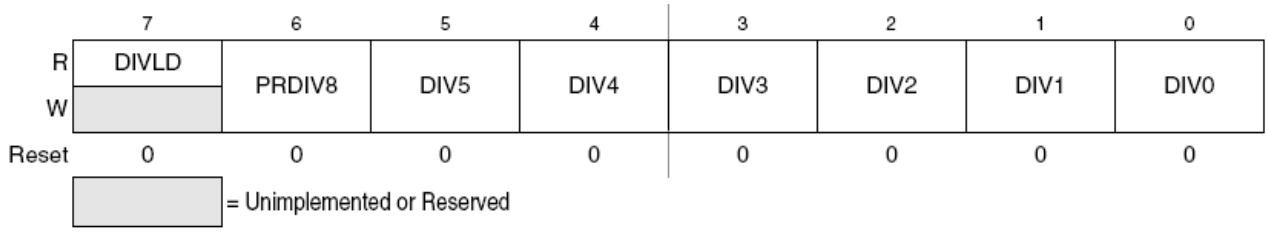
Para ativar esta proteção , basta configurar os bits SEC01 e SEC00 para 00, 01 ou 11. Estes bits estão localizados no registrador FOPT mas não podem ser alterados diretamente pelo usuário. O seu estado somente pode ser indiretamente alterado através do registrador NVOPT.

No *reset*, o registrador FOPT é carregado com o conteúdo do registrador NVOPT, localizado na memória FLASH.

Uma vez protegida, só existem duas formas de desproteger a memória:

1. Pelo apagamento completo da memória seguido de uma verificação de apagamento (o comando de verificação de apagamento desprotege a memória quando detecta que ela está totalmente apagada). Essa operação pode ser realizada com o uso do BDM.
2. Com uma senha de 8 bytes: os HCS08 possuem um mecanismo de senha parecido com o existente no monitor da família HC08. É possível programar uma senha nos endereços 0xFFB0 a 0xFFB7 que será utilizada para permitir o acesso à memória do microcontrolador. Essa senha somente está disponível quando habilitada pelo bit KEYEN (registrador FOPT). Note que a senha não pode ser utilizada com o BDM. Somente um programa executado a partir da memória protegida pode liberar o acesso à memória através da senha.

*Registrador de Divisão do Clock da FLASH **(FCDIV)***



DIVDL (*Divisor Loaded Status Flag*) – indicador de divisor carregado. Esse bit, inicialmente em nível lógico 0 após o reset, é levado a nível lógico 1 após a escrita no registrador FCDIV, de forma a indicar que o fator DIV foi programado. As operações de escrita e apagamento da FLASH são desabilitadas quando este bit está em nível lógico 0.

0 – o registrador FCDIV ainda não foi programado

1 – o registrador FCDIV foi programado e as operações de escrita na FLASH estão liberadas

PRDIV8 (*Prescale FLASH clock by 8*) – seleção do clock do controlador da memória FLASH:

0 – BUSCLK

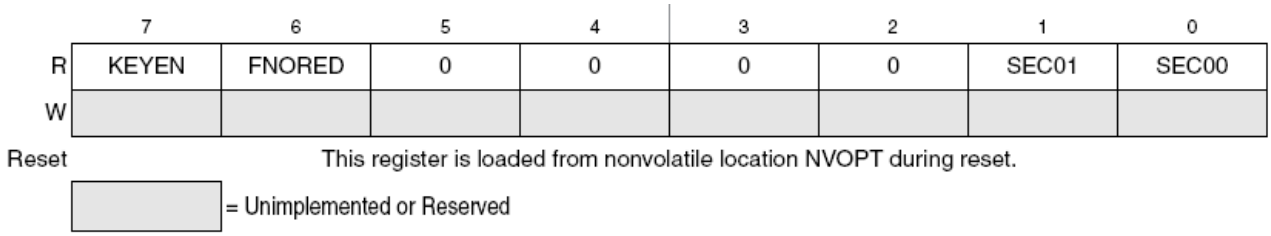
1 – BUSCLK / 8

DIV[5:0] (*Divisor for FLASH clock Divider*) – fator de divisão do clock do controlador da memória FLASH. O clock da memória FLASH é igual ao valor da fonte de clock dividido por DIV mais um. O valor resultante deve estar entre 150 e 200kHz.

A tabela a seguir apresenta a configuração do registrador FCDIV para diferentes clocks de barramento.

f_{bus}	PRDIV8	DIV5:DIV0	f_{FCLK}	Tempo médio para programação
20 MHz	1	12	192.3 kHz	5,2 µs
10 MHz	0	49	200 kHz	5 µs
8 MHz	0	39	200 kHz	5 µs
4 MHz	0	19	200 kHz	5 µs
2 MHz	0	9	200 kHz	5 µs
1 MHz	0	4	200 kHz	5 µs
200 kHz	0	0	200 kHz	5 µs
150 kHz	0	0	150 kHz	6,7 µs

Registrador de Opções da FLASH ***(FOPT e NVOPT)***



	7	6	5	4	3	2	1	0
R	KEYEN	FNORED	0	0	0	0	SEC01	SEC00
W								
Reset	This register is loaded from nonvolatile location NVOPT during reset.							

= Unimplemented or Reserved

KEYEN (*Backdoor Key Mechanism Enable*) – habilitação do mecanismo de senha de acesso.

0 – entrada via senha desabilitada, nenhum acesso é permitido

1 – entrada via senha habilitada

FNORED (*Vector Redirection Disable*) – controle de redirecionamento dos vetores de interrupção.

0 – redirecionamento habilitado

1 – redirecionamento desabilitado

SEC0[1:0] (*Security State Code*) – seleção do modo de proteção da memória. No modo protegido, o conteúdo da memória FLASH e RAM não pode ser acessado pelo BDM. No modo desprotegido, é possível ler o conteúdo da memória pelo BDM.

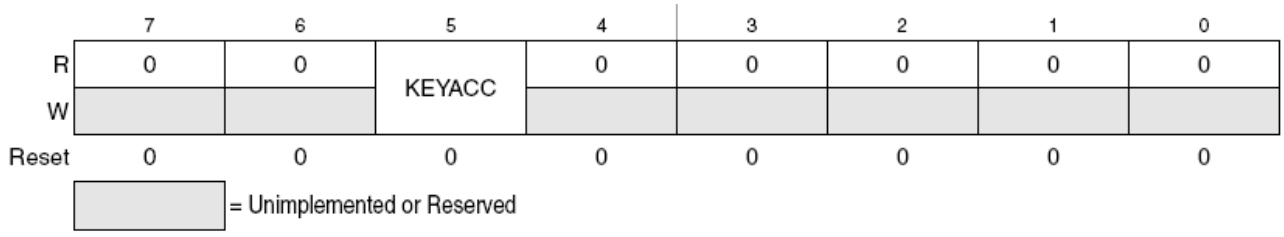
00 – modo protegido

01 – modo protegido

10 – modo desprotegido

11 – modo protegido

Registrador de Configuração da FLASH ***(FCNFG)***



	7	6	5	4	3	2	1	0
R	0	0	KEYACC	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

KEYACC (*Enable Writing of Access Key*) – seleção do modo de entrada da senha de 8 bytes.

0 – a escrita nos endereços 0xFFB0 a 0xFFB7 é interpretada como uma opção normal de escrita ou apagamento de memória.

1 – a escrita nos endereços 0xFFB0 a 0xFFB7 é comparada com a senha armazenada nos mesmos endereços

Registrador de Proteção da FLASH ***(FPROT e NVPROT)***

	7	6	5	4	3	2	1	0
R	FPS7	FPS6	FPS5	FPS4	FPS3	FPS2	FPS1	FPDIS
W	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)
Reset	This register is loaded from nonvolatile location NVPROT during reset.							

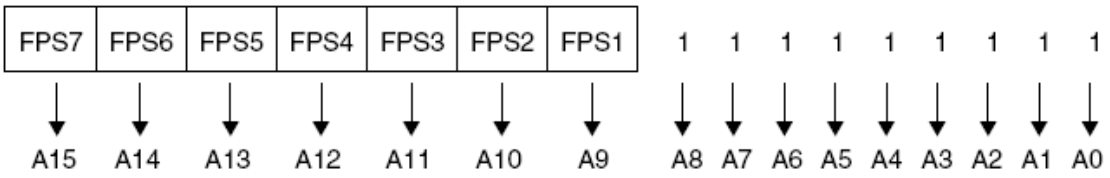
FPS (*Flash Protect Select Bits*) – seleção da área de memória FLASH protegida contra escrita e apagamento. Este registrador contém os 7 bits mais significativos do último endereço da memória que permanecerá desprotegida. O bloco entre o endereço seguinte até o final da memória (0xFFFF) pode ser protegido contra escrita / apagamento caso o bit FPDIS esteja em nível lógico 0.

FPDIS (*Flash Protection Disable*) – desabilitação da proteção da FLASH.

0 – bloco de memória especificado pelos bits FPS protegido

1 – memória desprotegida

A figura abaixo apresenta a formação do endereço a partir do qual a memória será desprotegida.



Registrador de Estado da FLASH ***(FSTAT)***

	7	6	5	4	3	2	1	0
R	FCBEF	FCCF	FPVIOL	FACCERR	0	FBLANK	0	0
W								
Reset	1	1	0	0	0	0	0	0

= Unimplemented or Reserved

FCBEF (*Flash Command Buffer Empty Flag*) – indicador de *buffer* de comandos da FLASH vazio.

0 – *buffer* de comandos cheio

1 – *buffer* de comandos vazio

FCCF (*FLASH command Complete Flag*) – indicador de comando da FLASH completo.

0 – comando em execução

1 – nenhum comando em execução

FPVIOL (*Protection Violation Flag*) – indicador de violação da memória FLASH (tentativa de escrever / apagar um endereço de memória protegida). Esta indicação pode ser apagada escrevendo-se nível lógico 1 no bit em questão.

0 – nenhuma violação

1 – houve uma violação de programação da FLASH

FACCERR (*Access Error Flag*) – Indicador de erro de acesso à FLASH. Esse bit é levado a nível lógico 1 sempre que a seqüência correta de programação da memória não é seguida, ou nos casos descritos no texto. Esta indicação pode ser apagada escrevendo-se nível lógico 1 no bit em questão.

0 – nenhum erro de acesso

1 – erro de acesso

FBLANK (*FLASH Verified as All Blank Flag*) – indicador de verificação da memória FLASH completamente apagada. Este bit é levado para nível lógico 1 caso, após a execução de um comando de verificação de apagamento da memória, esta esteja totalmente apagada. Esta indicação é apagada quando o bit FCBEF é levado a nível lógico 0.

0 – memória FLASH não apagada

1 – memória FLASH apagada

*Registrador de Comandos da FLASH **(FCMD)***

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W	FCMD7	FCMD6	FCMD5	FCMD4	FCMD3	FCMD2	FCMD1	FCMD0
Reset	0	0	0	0	0	0	0	0

Os possíveis comandos da FLASH são apresentados na tabela abaixo. Qualquer outro código gera um erro de acesso.

Comando	FCMD	Equate / define
Verificação de apagamento	0x05	mBlank
Programação de um byte	0x20	mByteProg
Programação de múltiplos bytes	0x25	mBurstProg
Apagamento de página (512 bytes)	0x40	mPageErase
Apagamento geral da memória	0x41	mMassErase

A seguir serão apresentados os procedimentos em linguagem C para realizar acessos de escrita e leitura na memória FLASH.

O primeiro passo para que possamos utilizar a memória FLASH é realocar o espaço utilizado para o programa principal. Por exemplo, o microcontrolador MC9S08AW60 tem como posição inicial de memória de programa (FLASH) o endereço 0x1860. O espaço de memória a ser utilizado como memória de armazenamento deve ser reservado no início da memória FLASH. Assim, para reservar 1952 bytes, devemos reposicionar o início da memória FLASH destinada ao programa desenvolvido para o endereço 0x2000 no segmento ROM do arquivo Project.prm.

```
SEGMENTS /* Here all RAM/ROM areas of the device are listed. Used in PLACEMENT
below. */
    Z_RAM                   =  READ_WRITE   0x0070 TO 0x00FF;
    RAM                     =  READ_WRITE   0x0100 TO 0x086F;
    ROM                     =  READ_ONLY    0x2000 TO 0xFFAF;
    ROM1                    =  READ_ONLY    0x0870 TO 0x17FF;
    ROM2                    =  READ_ONLY    0xFFC0 TO 0xFFCB;
 /* INTVECTS                    =  READ_ONLY    0xFFCC TO 0xFFFF; Reserved for
Interrupt Vectors */
END
```

A seguir deve-se incluir um arquivo com o código abaixo, em Assembly, das principais funções de acesso a memória na pasta *Sources* do projeto no CodeWarrior.

```
;*****************************************************************
;* This stationery is meant to serve as the framework for a      *
;* user application. For a more comprehensive program that       *
;* demonstrates the more advanced functionality of this          *
;* processor, please see the demonstration applications          *
;* located in the examples subdirectory of the                   *
;* Metrowerks Codewarrior for the HC08 Program directory         *
;*****************************************************************
; export symbols
            XDEF DoOnStack
            XDEF FlashErase
            XDEF FlashProg
            ; we use export 'Entry' as symbol. This allows us to
            ; reference 'Entry' either in the linker .prm file
            ; or from C/C++ later on

; include derivative specific macros

            Include 'mc9s08aw60.inc'

; variable/data section
MY_ZEROPAGE: SECTION  SHORT
; Insert here your data definition. For demonstration, temp_byte is used.
; temp_byte ds.b 1
```

```
; code section
MyCode:     SECTION
;***************************************************************
; this assembly routine is called the C/C++ application
DoOnStack:        pshx
                  pshh ;save pointer to flash
                  psha ;save command on stack
                  ldhx #SpSubEnd ;point at last byte to move to stack;
SpMoveLoop:       lda ,x ;read from flash
                  psha ;move onto stack
                  aix #-1 ;next byte to move
                  cphx #SpSub-1 ;past end?
                  bne SpMoveLoop ;loop till whole sub on stack
                  tsx ;point to sub on stack
                  tpa ;move CCR to A for testing
                  and #$08 ;check the I mask
                  bne I_set ;skip if I already set
                  sei ;block interrupts while FLASH busy
                  lda SpSubSize+6,sp ;preload data for command
                  jsr ,x ;execute the sub on the stack
                  cli ;ok to clear I mask now
                  bra I_cont ;continue to stack de-allocation
I_set:            lda SpSubSize+6,sp ;preload data for command
                  jsr ,x ;execute the sub on the stack
I_cont:           ais  #SpSubSize+3  ;deallocate  sub  body  +  H:X  +  command  ;H:X
flash pointer OK from SpSub
                  lsla ;A=00 & Z=1 unless PVIOL or ACCERR
                  rts ;to flash where DoOnStack was called
;***************************************************************
SpSub:            ldhx LOW(SpSubSize+4),sp ;get flash address from stack
                  sta 0,x ;write to flash; latch addr and data
                  lda SpSubSize+3,sp ;get flash command
                  sta FCMD ;write the flash command
                  lda #mFSTAT_FCBEF ;mask to initiate command
                  sta FSTAT ;[pwpp] register command
                  nop ;[p] want min 4~ from w cycle to r
ChkDone:          lda FSTAT ;[prpp] so FCCF is valid
                  lsla ;FCCF now in MSB
                  bpl ChkDone ;loop if FCCF = 0
SpSubEnd:         rts ;back into DoOnStack in flash
SpSubSize:        equ (*-SpSub)
;***************************************************************
FlashErase:       psha ;adjust sp for DoOnStack entry
                  lda #(mFSTAT_FPVIOL+mFSTAT_FACCERR) ;mask
                  sta FSTAT ;abort any command and clear errors
                  lda #mPageErase ;mask pattern for page erase command
                  bsr DoOnStack ;finish command from stack-based sub
                  ais #1 ;deallocate data location from stack
                  rts
;***************************************************************
FlashProg:        psha ;temporarily save entry data
                  lda #(mFSTAT_FPVIOL+mFSTAT_FACCERR) ;mask
                  sta FSTAT ;abort any command and clear errors
                  lda #mByteProg ;mask pattern for byte prog command
                  bsr DoOnStack ;execute prog code from stack RAM
                  ais #1 ;deallocate data location from stack
                  rts
;***************************************************************
```

Ainda, no arquivo principal do programa desenvolvido devemos incluir um arquivo com os protótipos destas funções.

```
#include "doonstack.h"
```

Este arquivo deve conter o seguinte código:

```
/*
- this file API between main.c and doonstack.asm
*/

#ifndef _doonstack
#define _doonstack

#ifdef __cplusplus
      extern "C" { /* our assembly functions have C calling convention */
#endif

/* prototype for DoOnStack routine */
void DoOnStack(void);
/* prototype for FlashErase routine */
/* Page Erase command */
void FlashErase(unsigned char *);
/* prototype for FlashProg routine */
/* Byte Program command */
void FlashProg(unsigned char *, unsigned char);

#ifdef __cplusplus
      }
#endif

#endif /* _doonstack */
```

No programa principal deve-se configurar a freqüência de operação do controlador da memória FLASH e os protótipos das funções que serão utilizadas.

```
#define vFCDIV 76 // divisor para o FCLK = (BUSCLK) / (8 * 12+1)
                  // FCLK = 20Mhz / (112) = 192,3 Khz
                  // O valor máximo não pode ultrapassar 200 Khz

// Protótipos de funções de acesso a memória FLASH
void Serial_Envia_Caracter(byte caracter);
void Serial_Envia_Frase(char *string);
void largura_pulso(byte canal,byte percent);
```

A seguir são apresentados exemplos de funções para leitura e escrita na memória FLASH.

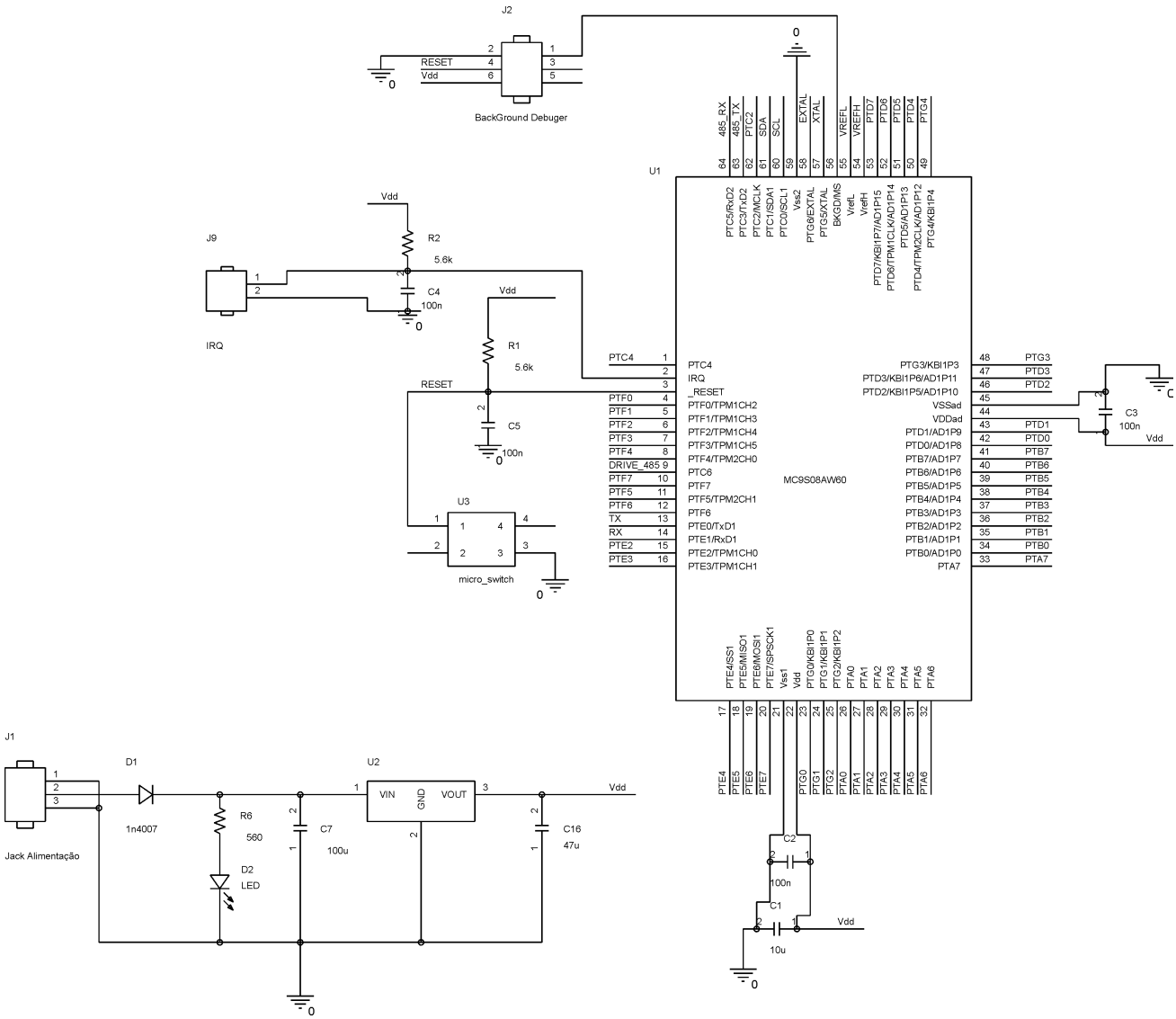
```
 //Lê um endereço de memória
 unsigned char flash_read(unsigned int endereco)
 {
  unsigned char *ponteiro;
  //guarda o endereço da memória em um ponteiro
  ponteiro = (char*) endereco;
  //retorna o valor armazenado no endereço indicado pelo ponteiro
  return (*ponteiro);
 }
```

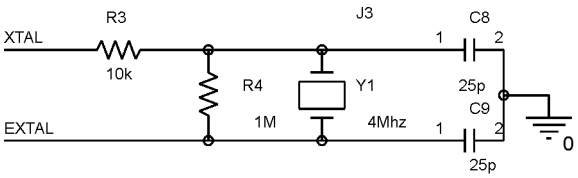
```
//Escreve em um endereço da memória FLASH
unsigned char flash_write(unsigned int endereco, unsigned char dado)
{
 unsigned char *ponteiro;
 //guarda o endereço da memória em um ponteiro
 ponteiro = (char*) endereco;
 //Verifica e apaga flag de erro de acesso a FLASH
 if(FSTAT_FACCERR)
   FSTAT_FACCERR = 1;
 //Chama a função de programação da FLASH
 FlashProg(ponteiro,dado);
 //se houver erro, retorna 1. Caso contrário, retorna 0
 if(FSTAT_FACCERR || FSTAT_FPVIOL)
   return(1);
 else
   return(0);
}


// Apaga uma página na FLASH
unsigned char flash_page_erase(unsigned int endereco)
{
 unsigned char *ponteiro;
 //guarda o endereço da memória em um ponteiro
 ponteiro = (char*) endereco;
 //Verifica e apaga flag de erro de acesso a FLASH
 if(FSTAT_FACCERR)
   FSTAT_FACCERR = 1;
 //Chama a função de programação da FLASH
 FlashErase(ponteiro);
 //se houver erro, retorna 1. Caso contrário, retorna 0
 if(FSTAT_FACCERR || FSTAT_FPVIOL)
   return(1);
 else
   return(0);
}

 // Inicializa o controlador da memória FLASH
 void init_FLASH(void){

   if(FSTAT_FACCERR)
     FSTAT_FACCERR = 1;
   FCDIV = vFCDIV;
   // Apaga a FLASH do endereço 0x1860 até a endereço 0x1FFF
   res = flash_page_erase(0x1860);     // 416 bytes
   res = flash_page_erase(0x1A00);     // 512 bytes
   res = flash_page_erase(0x1C00);     // 512 bytes
   res = flash_page_erase(0x1E00);     // 512 bytes
 }
```

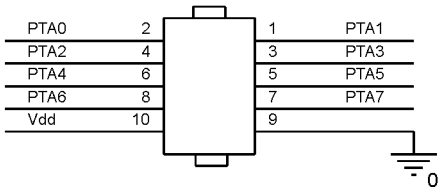
Exemplo de utilização das funções:

```
 init_FLASH();
 res = flash_write(0x1860,'A');
 res = flash_write(0x1A00,'B');
 res = flash_read(0x1A00);
```

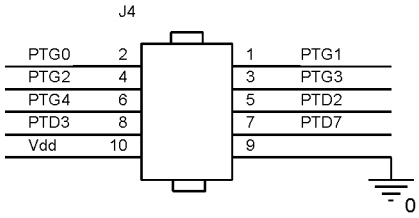
13. Diagrama esquemático da Placa MC9S08AW60



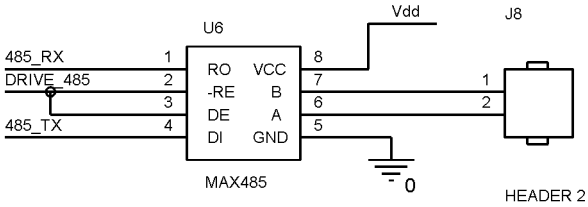




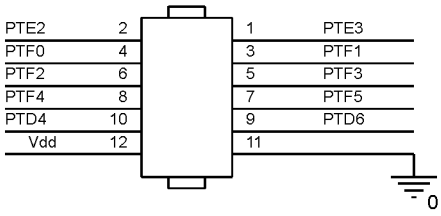
USO GERAL 1



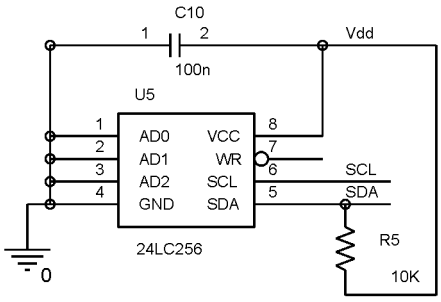
TECLADO



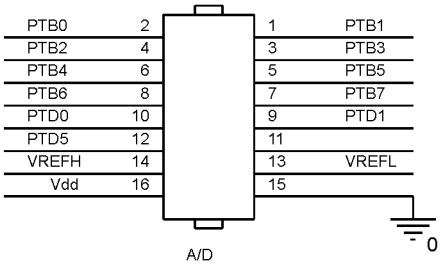
J5



TIMER/PWM

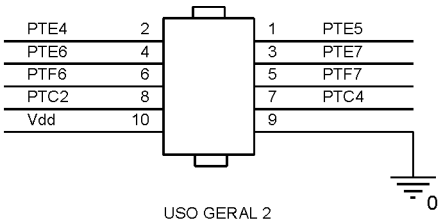


J6



A/D

J7



USO GERAL 2