

XM118 – Microcontroladores

PIC18

Exsto Tecnologia Ltda.

Av. Coronel Francisco Palma, 95 – Loja 2
Santa Rita do Sapucaí – MG
CEP: 37540-000
+55 35 3471 6898
www.exsto.com.br

Revisão	Principais Autores	Descrição da Versão	Término
A	José Domingos Adriano	Versão inicial.	01/06/2009
B	Raffael De Oliveira Marçano	Alteração do esquema elétrico	12/08/2009
C	Victor Piva Renault Grossi	Alteração no corpo do documento.	21/02/2011

© Copyright 2008 por Exsto Tecnologia Ltda.
Todos os direitos reservados

"Desenvolvido e produzido com orgulho no Brasil"

Exsto Tecnologia Ltda
Av. Cel. Francisco Palma, 95 - Sala 02 - Centro
Santa Rita do Sapucaí - MG
CEP: 37540-000
+55 35 3471 6898
www.exsto.com.br



Sumário

Lista de Figuras	10
Lista de Tabelas	13
Apostila Teórica	22
1 Microcontroladores	25
1.1 Sistema computacional	25
1.1.1 Memórias	26
1.1.2 Dispositivos de entrada e saída	28
1.1.3 Periféricos	30
1.1.4 CPU	31
1.2 Arquitetura Computacional	34
1.2.1 Arquitetura von-Neumann	35
1.2.2 Arquitetura Harvard	36
1.2.3 Microcontrolador, Microprocessador e DSP	37
2 PIC18, microcontroladores de alto desempenho	38
2.1 Microcontroladores PIC	38
2.1.1 A Microchip	39
2.2 Arquitetura	40
2.2.1 Pinagem e hardware básico	42
2.3 Memórias	43
2.3.1 Memória de programa	43
2.3.2 Memória de dados	46
2.4 Processador	49
2.4.1 ALU	50
2.5 Portais de I/O	52
2.6 Conjunto de Instruções	53
2.7 Oscilador	54
2.7.1 Oscilador a cristal (LP,XT e HS)	55
2.7.2 PLL - Phase Locked Loop	56
2.7.3 Oscilador RC	57
2.7.4 Oscilador Externo	58
2.7.5 Oscilador Interno	58
2.7.6 Comparação entre os modos de oscilador	59
2.7.7 Sistema de clock do PIC18F4550	59
2.7.8 Configuração de clock para operar com USB	61
2.7.9 Escolha de modo e troca de oscilador	62
2.8 Características Especiais	63
2.8.1 Modos de baixo consumo	64
2.8.2 Reset	65

2.8.3	POR - Power-On Reset	66
2.8.4	PWRT - Power-up Timer	66
2.8.5	OST - Oscillator Start-up Timer	67
2.8.6	BOR - Brown-out Reset	67
2.8.7	Causas de reset	67
2.8.8	Sequência de inicialização	68
2.8.9	Watch-Dog Timer	69
2.8.10	Proteção de Código (Code Protect)	69
2.8.11	Gravação e depuração	69
3	Ferramentas para o PIC18F: MPLAB IDE e Compilador C18	71
3.1	O MPLAB IDE	71
3.1.1	Criando um novo projeto	72
3.1.2	Construindo um projeto.	76
3.1.3	Gravando o microcontrolador	76
3.2	Simulação em C	76
3.2.1	Simulação	78
3.2.2	Outras funcionalidades	78
3.2.3	Analizador lógico	79
3.2.4	Geração de estímulos	80
3.3	O compilador C18	81
3.3.1	Outros Compiladores C	82
3.4	Depuração em C	82
4	Linguagem C para microcontroladores	84
4.1	Introdução a linguagem C	84
4.1.1	A função printf()	86
4.1.2	"C ou não C, eis a questão".	86
4.1.3	Passado e Futuro da linguagem C	89
4.2	Diretivas	89
4.2.1	#include	90
4.2.2	#define	90
4.2.3	#pragma config	90
4.2.4	#pragma interrupt e #pragma interruptlow	98
4.3	Tipos de dados	98
4.3.1	Declaração de variáveis e constantes	100
4.3.2	Variáveis locais e globais	101
4.3.3	Vetores e matrizes	101
4.3.4	Ponteiros	102
4.3.5	Qualificadores	103
4.4	Comandos Básicos	104
4.4.1	Atribuição	104
4.4.2	Decisão	110
4.4.3	Repetição	113
4.5	Funções	115
4.5.1	Passagem de parâmetros	115
4.6	Interrupções	118
4.6.1	Utilização das Interrupções	121
4.6.2	Interrupções no C18	124
4.7	Funções úteis	126
4.7.1	Operações matemáticas	128
4.8	Técnicas de Otimização de Código	128
4.8.1	Rotinas Matemáticas	129

4.8.2	Usando assembly no C	131
4.8.3	Uso de variáveis locais e globais	131
5	Aplicações	132
5.1	Display de 7 segmentos	132
5.1.1	Displays multiplexados	133
5.1.2	Apresentando valores em display	134
5.2	Buzzer	134
5.3	Teclado Matricial	135
5.3.1	Interrupção por mudança de estado na porta B	137
5.4	Display de cristal líquido	137
5.4.1	A biblioteca XLCD	141
5.5	Memórias EEPROM e FLASH	141
5.5.1	Memória EEPROM	143
5.5.2	Memória Flash	145
6	Periféricos	148
6.1	Como usar a ajuda das bibliotecas do C18	148
6.2	Contadores e Temporizadores	154
6.2.1	Timer 0	154
6.2.2	Timer 1	155
6.2.3	Timer 2	156
6.2.4	Timer 3	157
6.2.5	Funções	158
6.3	Comparadores Analógicos	158
6.3.1	Funcionamento do Comparador	161
6.4	Conversor Analógico-Digital	163
6.4.1	Quantização	163
6.4.2	Taxa de Amostragem	164
6.4.3	Linearidade	165
6.4.4	Desenvolvimento	165
6.4.5	Aplicação	167
6.4.6	Usando o conversor ADC no microcontrolador	167
6.5	CCP - Captura, Comparaçao e PWM	170
6.5.1	Modo captura	170
6.5.2	Modo comparação	172
6.5.3	Modo PWM	173
6.5.4	EPWM	175
6.6	EUSART - Porta Serial Assíncrona	176
6.6.1	Comunicação serial assíncrona	176
6.6.2	Comunicação paralela	176
6.6.3	Comunicação serial	176
6.6.4	EUSART do PIC18F4550	177
6.6.5	EIA-232C	180
6.6.6	Protocolos de comunicação	183
6.7	MSSP - Porta Serial Síncrona	185
6.7.1	SPI	185
6.7.2	I2C	187
6.8	USB	193

7 Anexos	196
7.1 Anexo A - Distribuição de Pinos do 18F4550	196
7.2 Anexo B - Registros de função especial do PIC18F4550	197
7.3 Anexo C - Leituras Recomendadas	198
7.4 Anexo D - Bibliografia:	199
7.5 Anexo E - Conjunto de instruções do PIC18	200
 Caderno de Experiências	 203
8 Aulas Práticas	204
8.1 Aula 1 - Introdução aos microcontroladores	205
8.2 Aula 2 - O PIC18F4550	208
8.3 Aula 3 - Assembly do PIC18	209
8.4 Aula 4 - Ferramentas de desenvolvimento	210
8.5 Aula 5 - Características Especiais	213
8.6 Aula 6 - Iniciando com a linguagem C	215
8.7 Aula 7 - Estruturas de Decisão	217
8.8 Aula 8 - Estruturas de Repetição	221
8.9 Aula 9 - Funções	224
8.10 Aula 10 - Interrupções	225
8.11 Aula 11 - Display de 7 segmentos e buzzer	227
8.12 Aula 12 - Teclado Matricial e LCD	229
8.13 Aula 13 - Contadores e temporizadores	230
8.14 Aula 14 - Conversor A/D	232
8.15 Aula 15 - Módulo CCP	233
8.16 Aula 16 - Comunicação serial assíncrona (RS232 e RS485)	235
8.17 Aula 17 - SPI	238
8.18 Aula 18 - I^2C	240
8.19 Aula 19 - Interface industrial	242
 Manual de Operação e Manutenção.	 244
9 Conteúdo do Kit:	246
9.1 Conteúdo do DVD	247
9.2 Instalações	247
9.2.1 Instalação do Hardware	247
9.2.2 Instalação dos Softwares	248
9.2.3 Configurações após a instalação	258
9.3 Hardware do kit XM118	264
9.3.1 Fonte de alimentação	265
9.3.2 Microcontrolador PIC18F4550	265
9.3.3 Conectores para acesso de I/O (PORTS)	267
9.3.4 ICD e conector ICSP	267
9.3.5 Configuração	269
9.3.6 Dispositivos de entrada	272
9.3.7 Dispositivos de saída	273
9.3.8 Acionamentos	274
9.3.9 Dispositivos analógicos	275
9.3.10 Interfaces seriais	276
9.3.11 Diversos	277
9.4 Resolvendo Problemas	279

Listas de Figuras

1	Características dos PIC18F4550	15
1.1	Diagrama em blocos genérico de um sistema computacional	26
1.2	Portais de entrada e saída	29
1.3	Esquema típico de um pino de I/O	30
1.4	Diagrama em blocos de uma CPU	31
1.5	Arquitetura von-Neumann	35
1.6	Pipeline	36
1.7	Arquitetura Harvard	36
2.1	Diagrama em blocos do PIC18F4550	40
2.2	Pinagem do PIC18F4550	43
2.3	Mapa e memória de programa	44
2.4	Estrutura do PC	45
2.5	Ponteiro da pilha	45
2.6	Mapeamento de memória de dados para PIC18F4550	47
2.7	Registros de funções especiais existentes no PIC18F4550	48
2.8	Ciclos de instrução	50
2.9	ALU	50
2.10	Registro STATUS / Bits de flag	51
2.11	Esquema genérico de um pino	52
2.12	Sistema de oscilação	55
2.13	Circuito do Oscilador a Cristal/Ressonador	55
2.14	Partida do oscilador	56
2.15	Esquema básico de um PLL	56
2.16	Circuito do oscilador no modo RC.	57
2.17	Forma de onda do oscilador RC.	57
2.18	Circuitos osciladores: (a) cristal paralelo e (b) cristal série.	58
2.19	Sistema de geração e seleção de clock do PIC18F4550.	60
2.20	Controle do oscilador (OSCTUNE)	62
2.21	Controle do oscilador (OSCCON)	63
2.22	Diagrama de reset do PIC18F4550.	66
2.23	Situações de Brown-out	67
2.24	RCON e seus bits.	67
3.1	Barra de ferramentas ICD2	76
4.1	Diagrama de interrupções do PIC18F	119
4.2	Registro INTCON	121
4.3	Registro INTCON 2	122
4.4	Registro INTCON 3	123
5.1	- Displays de LED: (a) disposição dos segmentos, (b) catodo comum e (c) anodo comum	132

5.2	Esquema Simplificado do Teclado	136
5.3	Fluxograma / Leitura do teclado.	137
5.4	Esquema Básico do Display	139
5.5	Caracteres do display.	140
5.6	EECON1	142
5.7	Processo de leitura e escrita da EEPROM	143
5.8	Processo de leitura da FLASH	146
5.9	Processo de escrita na FLASH(PIC18F452)	146
6.1	Aspecto da pasta PERIPH-LIB	149
6.2	Conteúdo da ajuda.	149
6.3	Resumo das bibliotecas.	150
6.4	Versão de AD do microcontrolador utilizado.	150
6.5	Apresentação da função ajuda.	151
6.6	Diagrama do Timer 0 no modo 8 bits	155
6.7	Diagrama do Timer 0 no modo 16 bits	155
6.8	Diagrama do Timer 1	156
6.9	Diagrama em blocos do timer 2	157
6.10	Diagrama do timer 3	157
6.11	Registro de configuração do comparador	159
6.12	Configurações possíveis do comparador.	160
6.13	Representação do funcionamento do comparador	161
6.14	Descrição da saída do comparador	162
6.15	Escala de conversão	164
6.16	Grau de Linearidade de Conversão	165
6.17	Diagrama em blocos do conversor A/D	166
6.18	Escala de conversão	166
6.19	Diagrama em blocos do ADC do PIC18F4550	168
6.20	Diagrama em blocos do ADC do PIC18F4550	168
6.21	Diagrama do modo captura	171
6.22	Diagrama do modo comparação	172
6.23	Diagrama em blocos no modo PWM	173
6.24	Sinal PWM	174
6.25	Comunicação Assíncrona	178
6.26	Módulo de Recepção	178
6.27	Módulo de Transmissão	179
6.28	Especificações elétricas do EIA-232C	181
6.29	Esquemas ligação de diversos transceptores a rede	182
6.30	Pinos para operação em SPI	186
6.31	Diagrama em blocos para o modo SPI	186
6.32	Diagrama em blocos para o modo I2C escravo.	188
6.33	Protocolo I2C.	188
6.34	Condições de Start e Stop.	188
6.35	Sinal de <i>no-acknowledge</i> (NACK).	189
6.36	Start e Stop	189
6.37	Operação de leitura.	190
6.38	Operação de escrita	190
6.39	Diagrama em blocos para o modo <i>I²C</i> escravo	190
6.40	Diagrama em blocos para o modo <i>I²C</i> mestre	191
6.41	Tipos de conectores	194
7.1	Convenções do PIC 18F	200
7.2	Conjunto de instruções do Pic	201

7.3	Conjunto de instruções do Pic(segunda parte)	201
7.4	Conjunto de instruções do Pic(terceira parte)	202
9.1	Diagrama de instalação do kit	248
9.2	Tela inicial.	249
9.3	Contrato de licença do MPLAB.	249
9.4	Seleção de versão.	250
9.5	Pasta de instalação do MPLAB.	250
9.6	Contrato de licença do Application Maestro.	251
9.7	Contrato de licença do compilador C32.	251
9.8	Resumo da instalação .	252
9.9	Andamento da instalação.	252
9.10	Instalação do compilador HI-TECH C.	253
9.11	Tela de conclusão da instalação.	253
9.12	Tela inicial de instalação do C18.	254
9.13	Contrato de licença.	255
9.14	Diretório de instalação	255
9.15	Componentes do pacote do compilador.	256
9.16	Campos a serem selecionados.	256
9.17	Campos a serem selecionados 2.	257
9.18	Tela inicio de instalação.	257
9.19	Tela de conclusão.	258
9.20	Assistente para adicionar novo hardware.	258
9.21	Procura avançada de drivers.	259
9.22	Localização do painel de configuração	260
9.23	Selecionando XICD como ferramenta de programação	260
9.24	Tela inicial do wizard para configuração do XICD	261
9.25	Opção de porta de comunicação do XICD	261
9.26	Fonte de alimentação do target.	262
9.27	Configuração da conexão automática	263
9.28	Configuração do download automático do sistema operacional do XICD	263
9.29	Resumo das configurações do XICD	264
9.30	Serigrafia do Kit XM118	264
9.31	Pinagem do PIC18F4550	266
9.32	Esquema de pinos do XICD	268
9.33	Ligaçāo do XICD	268
9.34	Ligaçāo do XICD	269
9.35	Domínios de terra	278
9.36	Ligaçāo de entradas ao módulo XMM01	279
9.37	Ligaçāo de saídas ao módulo XMM01	279

Listas de Tabelas

2.1	Características dos PIC18F4550	41
2.2	Características elétricas de entrada e saída	42
2.3	Ciclos "Q" do microcontrolador	50
2.4	Seleção de capacitores para oscilador a cristal	56
2.5	Comparação entre os modos de oscilador	59
2.6	Frequência / fator de divisão do PLL prescaler	61
2.7	PLL postscaler / Frequência do oscilador principal	62
2.8	Combinações possíveis de modos de baixo consumo e fontes de clock.	65
3.1	Comandos e suas descrições	78
4.1	Palavras reservadas pelo padrão ANSI.	85
4.2	Comparação entre C e Assembly	88
4.3	PLL Prescaler Selection	91
4.4	CPU System Clock Postscaler	91
4.5	USB Clock Selection	91
4.6	Oscillator Selection	92
4.7	Oscillator Selection	92
4.8	Internal/External Oscillator Switchover	92
4.9	Power-up Timer Enable	92
4.10	Brown-out Reset Enable	93
4.11	Brown-out Voltage	93
4.12	USB Voltage Regulator Enable	93
4.13	Watchdog Timer Enable	93
4.14	Watchdog Timer Postscale Select bits	94
4.15	MCLR Pin Enable	94
4.16	Low-Power Timer 1 Oscillator Enable	94
4.17	PORTB A/D Enable	94
4.18	CCP2 MUX	94
4.19	Stack Full/Underflow Reset Enable	95
4.20	Single-Supply ICSP Enable	95
4.21	Dedicated In-Circuit Debug/Programming Port (ICPORT) Enable	95
4.22	Extended Instruction Set Enable	95
4.23	Background Debugger Enable	95
4.24	Code Protection bit Block 0	95
4.25	Code Protection bit Block 1	96
4.26	Code Protection bit Block 2	96
4.27	Code Protection bit Block 3	96
4.28	Boot Block Code Protection bit	96
4.29	Data EEPROM Code Protection bit	96
4.30	Write Protection bit Block 0	96
4.31	Write Protection bit Block 1	96

4.32 Write Protection bit Block 2	96
4.33 Write Protection bit Block 2	97
4.34 Boot Block Write Protection	97
4.35 Configuration Register Write Protection	97
4.36 Data EEPROM Write Protection	97
4.37 table EEPROM Write Protection	97
4.38 Table Read Protection bit Block 1	97
4.39 Table Read Protection bit Block 2	97
4.40 Table Read Protection bit Block 3	98
4.41 Boot Block Table Read Protection	98
4.42 Tipos da dados inteiros	99
4.43 Tipos da dados inteiros	100
4.44 Representação de valores	100
4.45 Operadores matemáticos	105
4.46 Operadores Comparativos	106
4.47 Operadores Comparativos	107
4.48 Operadores Comparativos	107
4.49 operadores de manipulação de memória	108
4.50 Operadores de manipulação de memória	108
4.51 Controle das interrupções	124
4.52 Funções de conversão de tipo	126
4.53 Funções de Reset	126
4.54 Funções do portal B	127
4.55 Funções do portal B	127
4.56 Macros	127
4.57 Macros	128
 5.1 Acionamento de display de sete segmentos.	133
5.2 Dígitos/Linhas/Colunas de um teclado matricial.	135
5.3 Ligação entre o PIC e o teclado matricial.	136
5.4 Códigos hexadecimais de comandos do LCD.	138
5.5 Endereços de posição no display	138
5.6 Descrição dos pinos do display.	139
5.7 Comandos XLCD	141
 6.1 Funções da Biblioteca timers.h	158
6.2 Funções da biblioteca ANCOMP.h	163
6.3 Funções de uso do ADC	169
6.4 Funções de Captura	171
6.5 Funções de comparação	172
6.6 Funções para PWM	175
6.7 Funções da USART	180
6.8 Funções da USART	181
6.9 Funções da USART	185
6.10 Funções da SPI	187
6.11 Pinos para operação em SPI	187
6.12 Funções de <i>I²C</i>	192
6.13 Funções da EEPROM <i>I²C</i>	193
 9.1 Características dos PIC18F4550	266
9.2 Esquema de pinos do XICD	267
9.3 Configuração das chaves.	270
9.4 Configuração do dip switch	271

9.5 Leitura do teclado matricial pelo microcontrolador	272
9.6 Leitura do teclado matricial pelo microcontrolador	273

Introdução

Parabéns! Você acaba de adquirir um produto de alta qualidade e tecnologia de ponta. O Kit Educacional XM118 será de grande auxílio no aprendizado e desenvolvimento de sistemas digitais, na elaboração de cursos e treinamentos que envolvam microcontroladores PIC18.

A Exsto Tecnologia é uma empresa situada em Santa Rita do Sapucaí, Minas Gerais, cidade conhecida como "Vale da Eletrônica" por seu destaque na indústria eletroeletrônica e pela excelência de suas instituições de ensino. Nossa missão é sempre fornecer as melhores ferramentas para o desenvolvimento e aprendizado em eletrônica e desenvolvimento de software. Visite nosso site www.exsto.com.br para conhecer outras soluções e produtos oferecidos.

Apresentação do produto

O Kit Educacional XM118 é um ambiente de desenvolvimento que visa facilitar o aprendizado e o desenvolvimento de aplicações de microcontroladores PIC18.

Seu desenvolvimento foi baseado no PIC18F4550, que dentre outras características importantes possui um periférico USB Device, apesar de suportar outros componentes de 40 pinos (nestes casos nem todas as aplicações da placa são suportadas). A tabela abaixo apresenta algumas das principais características do PIC18F4550.

A tabela a seguir traz as principais características do PIC18F4550.

<i>Característica</i>	<i>PIC18F4550</i>
Freqüência de Operação	DC a 48MHZ
Memória de programa	32768 bytes
Memória de dados RAM	2048 bytes
Memória de dados EEPROM	256 bytes
Fontes de interrupção	20
Terminais de I/O	36
Temporizadores/Contadores	4
CCP	1
ECCP¹	1
Comunicação Serial	MSSP EUSART²
Comunicação USB	Sim
Comunicação Paralela	SPP³
Conversor analógico para digital	10 bits, 13 canais
Detector de tensão programável	1
Conjunto de instruções	75 convencionais + 8 do modo entendido

Notas: ¹ O módulo ECCP é um módulo CCP melhorado, com novas características

² O módulo EUSART é um módulo USART melhorado, com novas características

³ Porta paralela mestre

Figura 1: Características dos PIC18F4550

A escolha da família PIC18 se justifica por ser uma família de dispositivos de 8 bits de alto desempenho, que possui recursos comuns a todos os microcontroladores modernos, permitindo um aprendizado que não se limite apenas a essa família mas que se estenda a outras famílias PIC e mesmo a outros fabricantes. Outra vantagem dessa linha é poder usar um compilador C profissional como o C18. Na atualidade o desenvolvimento em sistema embarcados é praticamente todo feito em C, sendo usado o Assembly apenas em situações específicas.

O kit foi concebido para poder explorar ao máximo os recursos oferecidos pelo microcontrolador. As diversas aplicações estão dispostas de forma didática, com indicação na serigrafia da placa. Estão disponíveis diversos pontos de medida (teste points) que permitem a visualização dos sinais de maior relevância. O gravador/depurador XICD-2 embutido no kit permite, além da gravação a depuração (esse processo será mais explorado ao longo do curso); esse dispositivo é totalmente compatível com o ICD-2 Microchip e conecta-se diretamente ao MPLAB IDE. As principais características do kit XM118 são:

- Baseado no PIC18F4550;
- Fontes de alimentação;
 - +5 e +12V;
 - Fontes chaveadas com seleção automática de tensão e proteções;
- Bastidor robusto em aço;
- Gravador/depurador XICD-2 (compatível com MPLAB) embutido;
- Acesso a todos os pinos do microcontrolador;
- Configurações de hardware através de dip switch;
- 8 chaves dip switch ligadas ao PORTb;
- Teclado matricial de 16 teclas;
- 6 push-buttons em pinos de função especial(reset interrupções,timers);
- 8 LEDs convencionais ligados ao PORTD;
- 2 LEDs bicolores;
- 4 displays de 7 segmentos multiplexados;
- Display LCD Alfanumérico;
- Conector para LCD Gráfico 128x64 (Opcional);
- Buzzer;
- Lâmpada DC acionada por PWM;
- 4 relés;
- Gerador de sinal ajustável;

- Sistema de controle;
 - Resistência para aquecimento;
 - Ventoinha;
 - Sensor de temperatura ;
 - Tacógrafo para medida de velocidade da ventoinha;
- Dispositivos de comunicação serial;
 - Memória EEPROM I2C
 - Potenciômetro digital SPI
 - RTC com bateria própria
- Portas de comunicação serial;
 - RS232;
 - RS485;
 - USB 2.0 Full Speed (Device);
- Periféricos analógicos;
 - DAC a partir do PWM;
 - ADC com canais especiais:
 - * Potenciômetro;
 - * Sensor de temperatura;
 - * Duas entradas com ganho ajustável de 1 a 10;
 - * Entrada 0 a 10VDC;
 - * Entrada 4 a 20mA

Como várias aplicações utilizam os mesmos pinos do microcontrolador o kit possui um série de dip switchs para configuração de hardware. Consulte o manual e o esquema elétrico .

Objetivo deste documento

Este documento é dividido em três unidades, com os seguintes conteúdos:

- Apostila: apresenta a conceituação teórica sobre o assunto do kit
- Caderno de Experiências: traz a orientação para a realização das experiências práticas.
- Manual de Operação e Manutenção: reúne as informações necessárias para instalação e configuração do kit, além de trazer esquemas elétricos e outras informações importantes para manutenção.

Documentos adicionais como guias de software e manuais de componentes estão também contidos no CD ou DVD que acompanha o kit. Esse conteúdo será discutido oportunamente na seção Manual.

Orientação Pedagógica

O material didático (apostila e caderno de experiências) tem como função guiar o aluno durante todo o seu aprendizado em Microcontroladores PIC18 utilizando o kit XM118. A apostila traz os conteúdos teóricos sobre microcontroladores e linguagem C. Ela é organizada em capítulos, de forma que possa ser dividida conforme o plano de aula e carga horária do curso.

Temos o propósito de explorar os conceitos abordados e imediatamente prover a integração do aluno com o prazer da prática, tornado seu aprendizado mais interessante e consistente. Todo o conteúdo aqui é abordado de forma a fomentar a vontade do aluno e aplicar o conhecimento de forma imediata, permitindo que ele possa criar seus próprios circuitos a partir dos conhecimentos adquiridos.

O curso e, em especial as experiências práticas, foram pensados com o objetivo de levar o aluno a adquirir competências para o trabalho com microcontroladores, a saber:

- Compreender, analisar e comparar microcontroladores;
- Compreender e utilizar circuitos com microcontroladores;
- Ser capaz de entender e desenvolver programas em linguagem C para microcontroladores;
- Conhecer os periféricos mais comuns em microcontroladores;
- Desenvolver projetos usando microcontroladores;

O assunto microcontroladores é bastante extenso e não se pode fazer um curso com a pretensão de abordá-lo todo. No caso deste curso foi enfocado o microcontrolador PIC18F4550, um dispositivo de uma família avançada com uma série de recursos. Buscamos explorar a maioria desses recursos, apesar de alguns não terem sido incluídos e outros serem tratados apenas de forma introdutória. Foi dada especial atenção ao estudo da linguagem C para microcontroladores.

A divisão das aulas do curso prevê um esquema de 2 horas de teoria e 1 hora de prática para cada "aula". Abaixo é apresentada uma proposição de divisão de aulas teóricas. Para cada aula destas existe uma aula prática na seção "Caderno de Experiências" (para alguns conteúdos conceituais a aula prática é na verdade um questionário sobre o assunto abordado). Tendo em conta a extensão do assunto e a variedade de cargas horárias das disciplinas, propomos dois "níveis" em que o curso pode ser ministrado: básico e avançado. O nível básico aborda o conteúdo mínimo que entendemos como necessário a um curso de microcontroladores, com uma carga horária limitada em 30 horas (10 aulas). O nível **avançado** foi previsto para uma carga de 60 horas (19 aulas + 1 de projeto) e permite explorar alguns periféricos a mais, além de aumentar a oportunidade de exercitar a linguagem C com mais experiências. Recomendamos que na última

aula do módulo avançado seja proposto um projeto envolvendo diversos conceitos estudados ao longo do curso.

Aula	Níveis	Tópicos Assuntos	Tema
	B A		
		1 Microcontroladores	
1	x x	1.1 Sistema computacional 1.2 Arquitetura Computacional	Base conceitual
		2 PIC18, microcontroladores de alto desempenho	
2	x x	2.1 Microcontroladores PIC 2.2 Arquitetura 2.3 Memórias 2.4 Processador 2.5 Portais de I/O	Hardware do microcontrolador
3		2.6 Conjunto de instruções do PIC18	Assembly do PIC18
		3 Ferramentas para o PIC18F: MPLAB IDE e Compilador C18	
4	x x	3.1 O MPLAB IDE 3.2 Simulação em C 3.3 O compilador C18 3.4 Depuração em C	Ferramentas de desenvolvimento
5		2.7 Oscilador 2.8 Características Especiais 4.2 Diretivas	Características especiais
		4 Linguagem C para microcontroladores	
6	x x	4.1 Introdução a linguagem C 4.3 Tipos de dados 4.4.1 Atribuição	Fundamentos da linguagem
7	x x	4.4.2 Decisão	
8	x	4.4.4 Repetição	
9	x x	4.5 Funções	Funções
10	x x	4.6 Interrupções	Interrupções
		5 Aplicações	
11		5.1 Display de 7 segmentos 5.2 Buzzer	Interface Homem-Máquina
12	x x	5.3 Teclado Matricial 5.4 Display de cristal líquido	
		6 Periféricos	
13	x x	6.2 Contadores e Temporizadores	Temporizados e contadores
14	x	6.4 Conversor Analógico-Digital	
15	x	6.5.1 Modo captura 6.5.2 Modo comparação 6.5.3 Modo PWM	Módulo CCP - Captura, comparação e PWM
16	x	6.6 USART	Portas de comunicação
17	x	6.7.1 SPI	
18	x	6.7.2 I2C	
19	x	Módulo XMM01 e expansor de I/O	Interface Industrial

Algumas observações:

1. Para instituições que possuem cursos de linguagem C - ANSI anterior ao de microcontroladores as aulas 6, 7, 8 e 9 podem ser resumidas em 1 ou 2 aulas, enfocando apenas a parte prática.
2. A aula 1 tem o objetivo de introduzir ou revisar os conceitos fundamentais sobre microcontroladores e processadores. Se houve um estudo prévio de arquitetura de computadores ou outros modelos de microcontroladores, essa aula se torna dispensável.
3. Na apresentação das aulas alguns assuntos foram apresentados em seus sub-itens (como os tópicos 6.4 e 6.5) por uma questão de divisão de aulas. Entenda-se que todo o tópico deve ser estudado para um correto entendimento.

Curso de Microcontroladores PIC18

*A morte do homem começa no instante em que ele desiste de
aprender.*

- Albino Teixeira

A Um Poeta

Olavo Bilac

*Longe do estéril turbilhão da rua,
Beneditino, escreve! No aconchego
Do claustro, na paciência e no sossego,
Trabalha, e teima, e lima, e sofre, e sua!
Mas que na forma se disfarce o emprego
Do esforço; e a trama viva se construa
De tal modo, que a imagem fique nua,
Rica mas sóbria, como um templo grego.*

*Não se mostre na fábrica o suplício
Do mestre. E, natural, o efeito agrade,
Sem lembrar os andaimes do edifício:
Porque a Beleza, gêmea da Verdade,
Arte pura, inimiga do artifício,
É a força e a graça na simplicidade.*

O mundo da tecnologia está em constante e rápida evolução. Acompanhar essa evolução exige dedicação e investimento da parte dos que aceitam o desafio de não só consumir tecnologia, mas também de criá-la.

A linha de microcontroladores PIC18 foi lançada com uma arquitetura renovada e otimizada em relação à linha PIC16, trazendo também várias inovações que permitem o desenvolvimento de aplicações mais avançadas e eficientes. Faz-se necessário, então, uma atualização dos conhecimentos para tirar o máximo proveito das novas características da linha PIC18.

Paralelamente a essa evolução de hardware processa-se uma revolução de software, onde a linguagem C é apresentada como escolha mais adequada para o desenvolvimento de código em microcontroladores, em substituição à linguagem Assembly. Essa revolução é ocasionada não só pelo desenvolvimento de compiladores mais confiáveis e eficientes como também pela otimização do conjunto de instruções e arquitetura dos microcontroladores para a linguagem C. A linha PIC18, objeto de nosso estudo, tem sua arquitetura otimizada para C e a Microchip oferece um eficiente compilador, o C18.

Tendo isso em vista, foi desenvolvido um curso que permitisse abordar as duas inovações: a linha PIC18 e programação de microcontroladores em linguagem C.

No capítulo 1 é apresentado o conceito de microcontroladores. Esse capítulo serve não só para introduzir os que ainda não estão familiarizados com microcontroladores como também como uma revisão dos conceitos fundamentais. Permite, dessa forma, equalização dos conhecimentos para melhor aproveitamento do curso.

O capítulo 2 são apresentadas as características da linha PIC18, tratando especificamente do PIC18F4550. São estudados detalhadamente as memórias de programa e dados, tipos de osciladores, características especiais, ALU. Esse capítulo concentra o estudo do hardware do microcontrolador, com exceção dos periféricos que são tratados no capítulo 6 e das interrupções, abordadas juntamente com a linguagem C no capítulo 4.

No capítulo 3 trata-se das ferramentas de software usadas, o MPLAB IDE e o compilador C18. A instalação, configuração e uso dessas ferramentas são apresentados de forma simples, com

sequencias passo-a-passo de configuração e uso. Estudam-se também as ferramentas de simulação e depuração.

O capítulo 4 trata da linguagem C e seu uso em microcontroladores. Buscou-se apresentar os conceitos de C conforme o padrão ANSI, fazendo comentários sobre detalhes de implementação do C18 quando oportuno. O compilador C18 é acompanhado de diversas bibliotecas de funções, tanto as padronizadas para a linguagem C como as específicas para microcontroladores PIC. As funções dessas bibliotecas são apresentadas detalhadamente no documento "MPLAB® C18 C COMPILER LIBRARIES" (que se encontra no CD do curso) e aos arquivos de ajuda do compilador. Outros capítulos do curso fazem referência a estes documentos.

Diversas aplicações de microcontroladores são apresentadas no capítulo 6. O enfoque principal é na interface homem-máquina, que permite a interação do microcontrolador com o usuário.

No capítulo 6 são estudados vários dos periféricos do PIC18F4550. Cada tópico é iniciado com o funcionamento do periférico, seguindo do estudo das rotinas do compilador C18 que permitem seu uso.

Nos anexos encontram-se diversas informações úteis e para consulta no dia-a-dia. Atenção especial deve ser dada ao anexo D Bibliografia e leituras recomendadas, que comenta diversas fontes de informação sobre PIC, PIC18 e linguagem C.

Capítulo 1

Microcontroladores

O objetivo deste curso é tratar dos microcontroladores da linha PIC18XXXX, mais especificamente do PIC18F4550. Existem diversos textos que também tratam desse assunto. Contudo, existe uma carência de informações sobre o que vem a ser um microcontrolador, seus princípios de funcionamento, as partes comuns a todos os microcontroladores, dentre outras coisas. Mesmos os manuais dos componentes admitem que o leitor já possua um conhecimento prévio do assunto. Para suprir essa carência, a primeira unidade desse material tratará de sistemas computacionais e microcontroladores de forma genérica. É importante este embasamento teórico não somente para garantir um melhor aproveitamento no estudo do PIC 18F4550 como também para tornar o leitor apto a entender o funcionamento de outros microcontroladores, tanto da linha PIC como de outros fabricantes. Esses conhecimentos também são importantes na escolha do microcontrolador a ser utilizado em um projeto, pois permite estabelecer as bases conceituais para a comparação de suas características.

1.1 Sistema computacional

Inicialmente devemos conceituar o que vem a ser um sistema computacional. Como já sabemos, os sistemas digitais podem ser classificados em três tipos: combinacionais, sequenciais e computacionais. Os sistemas computacionais, sendo o tipo mais complexo, são compostos por sistemas combinacionais e sequenciais. O que caracteriza um sistema computacional é a possibilidade de ser programado. Esse conceito não é recente, sendo que já no século XIX foram tentadas implementações dele. Contudo, somente na segunda metade do século passado é que foram desenvolvidas tecnologias (transistor, circuitos integrados) que permitiram a construção dos processadores e computadores como os que conhecemos hoje.

Um sistema computacional é composto por hardware (parte física) e software (programa). O hardware dos sistemas computacionais, ao contrário do que ocorre com sistema combinacionais e sequenciais, não possui uma aplicação específica. Não basta alimentá-lo devidamente para que ele funcione. É necessário que exista um software para ser executado. Em contrapartida, um mesmo hardware pode executar uma infinidade de funções diferentes, simplesmente alterando o seu software. Uma boa analogia é imaginar o hardware como um instrumento musical e o software como uma partitura. Um piano por si só não faz nada, é necessária uma música que possa ser executada a fim de se obter algum resultado. Da mesma forma que a música contida na partitura, um programa será composto por um conjunto limitado de símbolos (no caso da partitura são as

notas musicais e no caso do programa são as instruções) que podem ser organizados de diversas formas diferentes, obtendo-se diferentes resultados.

Como exemplo de sistema computacional, o primeiro que nos vem em mente é o computador pessoal (PC), hoje tão difundido. Contudo, existem vários equipamentos, tão ou mais comuns que os PC's, que são sistemas computacionais. Por exemplo, os vídeos-game, os mini-games, as calculadoras, palm-top's, etc. Além disso, temos os microcontroladores, que são o objetivo desse nosso estudo, que estão presentes nas mais diversas aplicações, de eletrodomésticos a plantas industriais, de alarmes a equipamentos médicos.

A estrutura básica de um sistema computacional é como a apresentada no diagrama em blocos abaixo.

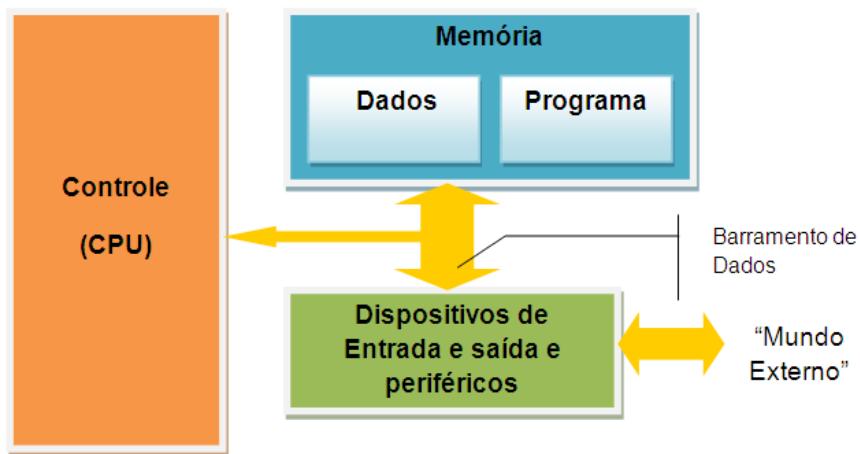


Figura 1.1: Diagrama em blocos genérico de um sistema computacional

A seguir são estudados detalhadamente cada um desses blocos.

1.1.1 Memórias

Memórias são dispositivos que armazenam informações. No caso de um sistema computacional, essa informação, assim como a memória, pode ser dividida em dois tipos: programa e dados.

Programa é a sequencia de instruções que ao CPU deve seguir para desempenhar a função a que se destina a aplicação. Quando se trata de um programa executado por um microcontrolador são usados também os termos firmware e software embarcado (embedded software). Já dados são variáveis que a CPU usa durante a execução do programa.

Existem também Registros (bytes de memória) com funções especiais que trabalham junto ao processador (CPU). Em alguns casos esses registros estão dentro do próprio processador e em outros eles são mapeados juntamente com a memória de dados.

Memória de programa

Sua função é armazenar o software (programa) a ser executado no sistema. Também é utilizada para guardar tabelas de constantes. Dentre os vários endereços de memória de programa existem alguns com finalidade específica, chamados vetores. São endereços para onde o programa desvia quando ocorrem determinados eventos. Assim, sempre existirá um vetor de reset, que é onde o programa começa (ou recomeça) quando o sistema é inicializado (ou resetado). Há também

as interrupções, das quais trataremos mais adiante, e que causam desvios no fluxo do programa quando ocorrem determinados eventos. Os endereços associados às interrupções são chamados de **vetores de interrupção**. Podemos encontrar sistemas onde, independentes da causa da interrupção, há apenas um vetor para onde o programa desvia e sistemas onde cada fonte ou tipo de interrupção possui um vetor.

De uma forma geral, a memória de programa é uma memória não volátil. Isto quer dizer que o programa não é perdido quando se retira a alimentação do sistema; se não fosse assim, o sistema teria que ser programado a cada vez que fosse colocado em funcionamento.

As diferentes tecnologias usadas para implementação da memória de programa dos microcontroladores são:

- ROM: somente de leitura. Geralmente chamada de masked-ROM (masked = mascarado), é gravada pelo fabricante do componente, o que em quantidades muito grandes reduz custos.
- EPROM: apagável através de luz ultra-violeta (memórias "janeladas"). Utilizada no processo de desenvolvimento e algumas vezes no produto acabado, quando há a necessidade de se poder alterar o software.
- OTP: programável somente uma vez, utilizada em produção.
- Flash: é uma memória eletricamente apagável (EEPROM) com tempos de acesso para leitura rápido. Oferece uma grande flexibilidade, pois geralmente é possível reprogramar um equipamento sem retirar o microcontrolador da placa (gravação in-circuit). Além disso, alguns modelos permitem que o microcontrolador altere sua própria memória de programa.

Praticamente todos os microcontroladores já possuem internamente algum desses tipos de memória de programa. De uma forma geral, o tipo de memória não influencia nas características dos componentes. Atualmente quase todos os microcontroladores possuem versões em memória Flash, pois a mesma apresenta muitas vantagens a um custo dos mais baixos. Os lançamentos em microcontroladores tendem a ser todos com memória Flash, mas ainda existem alguns componentes sendo fabricados com os demais tipos de memória.

Alguns microcontroladores apresentam a possibilidade de **expansão de memória**, isto é, permitem acrescentar mais componentes de memória externamente; outros não.

A quantidade de memória interna é uma das principais diferenças entre microcontroladores de uma mesma linha, e tem um impacto significativo no preço dos componentes. Esse é um dos motivos porque se deve otimizar os programas para que tenham o menor tamanho possível, ou seja, ocupem a menor quantidade de memória possível.

Memória de Dados

É a memória onde são armazenados os dados a serem processados pelo computador (as variáveis dos programas e outras informações.). Como esses dados são constantemente alterados, a memória utilizada para armazená-los é do tipo RAM, que pode ser reescrita indefinidamente. Como se trata de uma memória volátil, quando a alimentação é cortada esses dados são perdidos.

Alguns microcontroladores disponibilizam também memórias EEPROM para armazenar dados que não devem ser perdidos com a falta de energia.

Da mesma forma que a memória de programa, os microcontroladores já possuem internamente alguma quantidade de memória RAM de dados. Aqui também alguns permitem expansão de memória e outros não. A quantidade de RAM também varia de componente para componente.

1.1.2 Dispositivos de entrada e saída

São responsáveis por realizar a interface do processador com o "mundo externo". É através deles que um sistema computacional adquire dados externos e aciona processos. São comumente chamados de **dispositivos de I/O** (Input/Output - Entrada/Saída), **portais de I/O** (no inglês, **ports**) ou simplesmente portais. Para entender seu funcionamento antes devemos entender como funcionam os barramentos.

As várias partes de um sistema computacional (CPU, memórias, dispositivos de I/O) são ligadas entre si através de barramentos (também chamados via ou, em inglês, bus), que são ligações físicas de comunicação paralela entre os componentes. Existe um barramento de dados, ao qual todos os dispositivos estão ligados e por onde trocam informações. Todos os dispositivos ligados ao barramento de dados são lidos (colocam dados no barramento) ou escritos (recebem os dados do barramento) pela CPU, que controla todo o processo. Assim temos várias saídas e entradas ligadas aos mesmos pontos. Esse sistema não entra em conflito porque somente um dispositivo é acessado por vez. Enquanto um dispositivo é acessado (lido ou escrito) os demais ficam em alta impedância, se são saída, ou ignoram os dados do barramento, se são entrada. Dessa forma, surge a necessidade de outro barramento, o barramento de endereços, que permite ao processador "endereçar", ou seja, indicar qual dispositivo será acessado. Além desse, há também um barramento de controle, que indica basicamente se o acesso é de leitura ou escrita, pois muitas vezes um mesmo endereço de memória ou dispositivo de I/O pode ser lido ou escrito. O barramento de controle também serve para indicar se está sendo acessado um endereço de memória ou um dispositivo de entrada e saída, já que utilizam o mesmo barramento de endereço.

O tamanho de um barramento, que é seu número de bits, deve ser considerado com atenção. O tamanho do barramento de dados limita o tamanho básico de dados que trafegam pelo sistema e são processados pela CPU. Por exemplo, ao dizer que um sistema é de 8 bits podemos concluir que ele faz operações matemáticas envolvendo operadores de 8 bits. Por isso é comum classificar os sistemas computacionais pelo tamanho de seu barramento. Há hoje microcontroladores de 8, 16 e 32 bits.

Já o tamanho do barramento de endereços nos informa qual a quantidade máxima de endereços (de memória ou dispositivos de I/O) pode ser acessados. A capacidade de memória é de 2^n endereços, sendo n o número de bits do barramento de endereços. Exemplificando, um barramento de endereço de 10 bits é capaz de endereçar $2^{10} = 1024$ endereços.

Conforme exposto acima, os dispositivos de entrada devem permitir colocar sua saída ligada ao barramento de dados em alta impedância. Para tanto, são tradicionalmente utilizados buffers com saída tri-state para essa função.

Para a implementação dos dispositivos de saída são utilizados latches, pois permitem o "carregamento" dos dados somente quando são acionados.

A seleção dos dispositivos de I/O a partir do barramento de endereços é feita por decodificadores.

Exemplo

A figura abaixo mostra um exemplo de utilização de buffers e latches como portais de entrada e saída, respectivamente.

Ambos os portais estão ligados ao barramento de dados. Cada portal possui um decodificador (DEC), que apresenta nível alto na saída quando o valor do barramento de endereços é o endereço do portal. Os sinais RD (read - ler), WR (write - escrever) e IORQ (I/O request - requisição de I/O) constituem o barramento de controle.

Para realizar uma operação de escrita do portal, o processador inicialmente coloca os dados a serem escritos no barramento de dados e o endereço do portal no barramento de endereços. Então são acionados os sinais WR e IORQ. Nesse instante temos '1' nas três entradas da porta AND que aciona o terminal E do latch, que faz com que ele carregue os dados presentes no barramento na saída.

Para a operação de leitura, o endereço do portal é colocado no barramento de endereços. Em seguida, os sinais IORQ e RD são acionados. Assim temos '1' no terminal de habilitação de saída do buffer e o valor em sua entrada é transferido para o barramento de dados, podendo ser armazenado pela CPU.

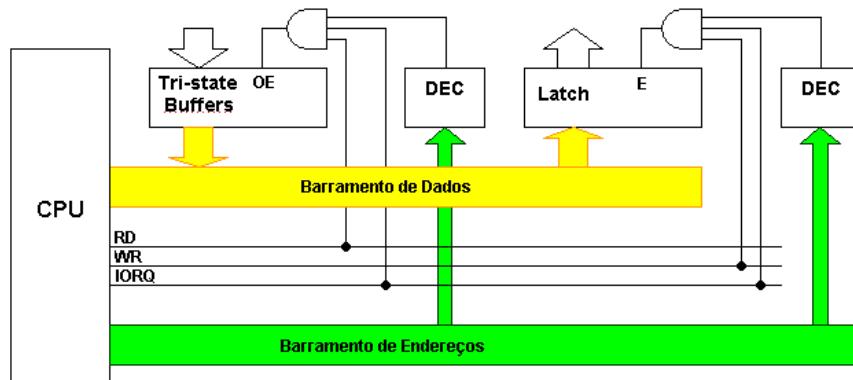
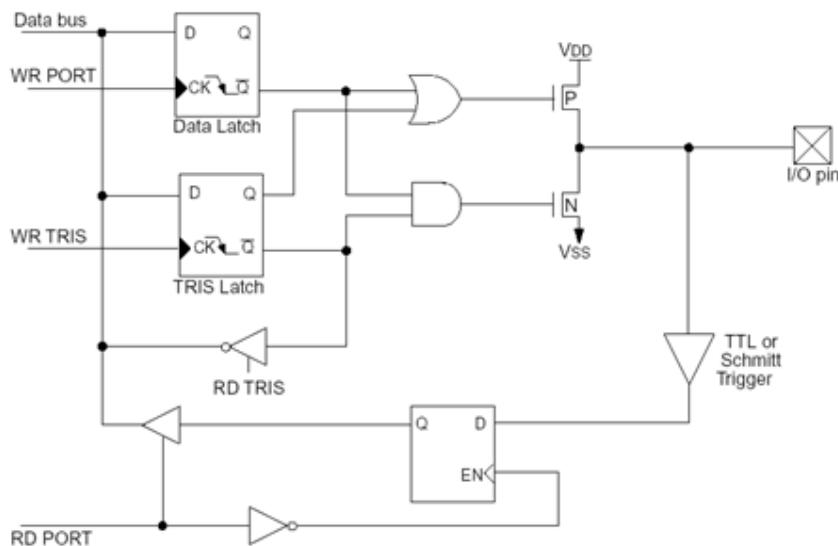


Figura 1.2: Portais de entrada e saída

A função do microcontrolador, como o próprio nome diz, é controlar processos e circuitos. Para otimizar essa função, seus portais de entrada e saída são tratados como registros de funções especiais, ou seja, são lidos e escritos como se fossem simples bytes de memória. Em sistemas como microprocessadores geralmente o tratamento é diferente, havendo instruções específicas para acesso aos portais e outras implicações.

Para maior flexibilidade do sistema, geralmente os microcontroladores utilizam um sistema de portais mais complexo que o apresentado acima, que permite que um mesmo terminal do microcontrolador seja configurado como entrada ou saída. Para que isso seja possível é utilizado um registro de portal e um registro de direção de dados (DDR - Data Direction Register). Esse último permite selecionar a direção de cada bit de um portal de saída, individualmente. A figura abaixo mostra o esquema genérico de um bit de portal. Outros microcontroladores utilizam configurações semelhantes.



Note: I/O pin has protection diodes to VDD and VSS.

Figura 1.3: Esquema típico de um pino de I/O

O latch de saída é chamado "Data Latch". O registro de direção de dados é o "TRIS Latch". Se o valor armazenado no latch TRIS for '1' o pino está configurado como entrada: os transistores de saída estarão cortados independente do valor na saída do latch de dados. Quando for realizada uma operação de leitura do portal (acionando o sinal RD PORT) o nível lógico presente no pino do microcontrolador é armazenado em um latch de entrada e passado ao barramento de dados (Data Bus) através de um buffer tri-state. No restante do tempo a saída desse buffer fica em alta impedância. Se o valor armazenado no latch TRIS for '0', o driver de saída está ativo, forçando o pino ao nível lógico correspondente ao armazenado no latch de dado. Qualquer dado escrito no latch de dados é imediatamente transferido para o terminal do microcontrolador.

Caso seja feita uma operação de escrita em um terminal configurado como entrada, o dado escrito será armazenado no latch de dados, mas não será transferido para o terminal, pois o driver está desativado. Por outro lado, se for realizada uma operação de leitura em um terminal configurado como saída, o valor lido será no presente o terminal, que é o mesmo escrito no latch de dados.

Todo esse processo, contudo, ocorre de forma automática e transparente para o programador, bastando que este defina a direção de cada pino.

1.1.3 Periféricos

Além dos portais de I/O e as memórias, podemos ter muitos outros tipos de dispositivos ligados ao barramento de dados. Esses dispositivos nada mais são do que circuitos destinados a realizar funções especiais, úteis na aplicação a que se destina o sistema computacional. Esses dispositivos periféricos são particularmente importantes nos microcontroladores, que os tem uma grande variedade.

Como periféricos mais comuns podemos citar os temporizadores e contadores (Timers), os módulos de comunicação serial, conversores A/D e D/A, módulos de CCP (Captura, comparação e PWM), drivers de LCD, comparadores analógicos, etc.

O modo de acesso aos periféricos é semelhante ao de acesso aos portais de I/O. Eles muitas vezes possuem vários registros de parâmetros que podem ser configurados, e um ou mais registros de entrada e saída de dados.

1.1.4 CPU

A CPU (Central Processing Unit - Unidade Central de Processamento) ou processador é parte principal de um sistema computacional. É comum se referir à CPU dos microcontroladores como core (núcleo). Sua função é executar as instruções do programa e processar dados. Para tanto ela controla todas as demais partes do sistema e envia ou recebe dados dessas partes. Ela também é capaz de interpretar e colocar em execução as instruções que compõe o programa e realizar operações lógicas e aritméticas.

Genericamente um processador é organizado conforme o diagrama em blocos abaixo.

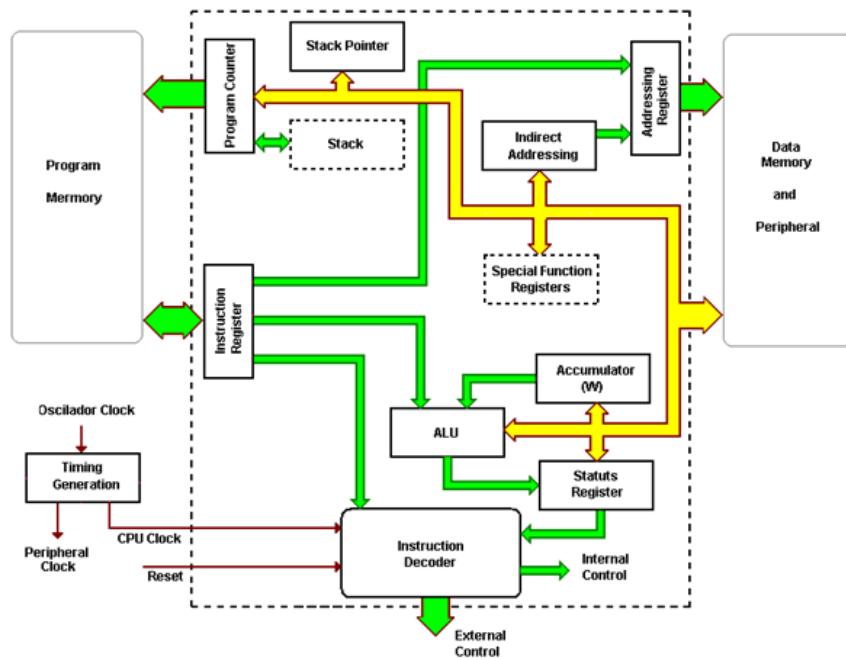


Figura 1.4: Diagrama em blocos de uma CPU

O diagrama é o de um processador de arquitetura Harvard. O que vem a ser essa arquitetura será explicado mais adiante. Por enquanto o importante é saber que nem todos os processadores são organizados dessa forma. Mas seus componentes principais são os mesmos, o que muda é a forma como se comunicam. Optamos por apresentar um processador dessa forma porque é o tipo de CPU utilizado nos microcontroladores PIC.

Os nomes de cada bloco estão em inglês no diagrama para facilitar a identificação desses blocos nos manuais de microcontroladores e microprocessadores, que comumente são escritos nessa língua. Os blocos pontilhados podem ou não estar presentes. As vias em amarelo (cinza claro) representam o barramento de dados interno do processador, que é ligado ao barramento de dados do sistema. As vias em verde (cinza escuro) são outras vias de dados e sinais de controle.

O principal bloco de um processador é o decodificador de instruções (Instruction Decoder). Voltando a analogia entre um sistema computacional e um instrumento musical, o decodificador de instruções pode ser comparado ao músico, que age sobre o instrumento (hardware) executando a

música (software). Esse bloco é composto por um decodificador e um contador. Tal decodificador pode ser visto como um livro de receitas culinárias. No livro, cada página contém uma receita e dentro de cada receita há os passos para sua execução. De modo análogo, cada instrução do processador é como o endereço de uma "página" onde está a sequências de acionamento dos sinais de controles (internos e externos ao processador) que permitem a execução da instrução. O contador é responsável por fazer com que os passos para a execução de uma instrução sejam executados em sequencia. Cada modelo de processador possui um conjunto, ou set, de instruções que pode executar.

O barramento de controle interno (Internal Control) permite ao decodificador de instruções controlar os blocos internos do processador, enquanto o barramento de controle externo (External Control) já foi discutido e tem a função de indicar se o acesso é de leitura ou escrita, se é em memória ou dispositivos de I/O.

Para que o decodificador de instruções possa executar uma instrução é necessário que a mesma seja lida da memória de programa e levada até ele. O programa armazenado na memória é uma sequencia de instruções. Podemos então supor que para endereçar corretamente essas instruções deveria haver um contador. Ele existe e é chamado Contador de Programa ou PC (Program Counter). A cada instrução iniciada o PC é incrementado. Portanto ele aponta a próxima instrução, isto é, contém o endereço da próxima instrução a ser executada. A saída desse contador é ligada a um registro (Program Addressing) que é carregado ao final de cada instrução e sua saída é o barramento de endereços da memória de programa.

Quando a CPU é resetada, o PC é automaticamente carregado com o valor do vetor de reset. Geralmente o vetor de reset é o endereço zero da memória, mas nem todos os sistemas trabalham assim. Durante a execução do programa, um valor pode ser carregado no PC. Isso ocorre para desviar o fluxo do programa. Existem instruções que realizam essa alteração de fluxo, que pode ser de dois tipos: desvio ou chamada.

Quando ocorre um desvio o conteúdo do PC é alterado para que ele passe a executar o programa a partir de outro ponto. No nosso sistema genérico isso é feito simplesmente carregando um valor proveniente do barramento de dados.

Na execução de uma chamada o fluxo do programa também é desviado para um determinado ponto para executar um trecho do programa, mas nesse caso ele deve retornar ao ponto do programa onde ocorreu o desvio (mais precisamente à primeira instrução após a instrução de chamada). Isso permite executar uma sub-rotina, conceito que será discutido quando tratarmos do software.

Fica claro então que em uma chamada o endereço de retorno deve ser armazenado em algum lugar, caso contrário não seria possível retornar ao ponto onde a chamada ocorreu. Esse lugar é chamado de pilha (stack). O nome pilha se deve a seu funcionamento, que é semelhante a uma pilha de pratos: como pode haver várias chamadas consecutivas sem que haja retorno, os endereços de retorno são armazenados "uns sobre os outros". Quando ocorre uma instrução de retorno (que é o que faz o programa retornar da chamada), o programa volta para o último endereço guardado, e assim sucessivamente até que a pilha esteja vazia. A pilha é então uma porção de memória onde podem ser armazenados os endereços de retorno. Em alguns sistemas é possível guardar outros tipos de dados na pilha, o que não ocorre com os microcontroladores PIC. Existem sistemas onde essa porção de memória é implementada no processador e outros

onde é parte da memória de dados. Em ambos os casos surge a necessidade de se conhecer qual é o topo da pilha, ou seja, qual é a próxima posição para se armazenar um endereço de retorno. Para isso existe um registro (na verdade um contador) chamado ponteiro da pilha (Stack Pointer) que aponta o topo. Esse registro é incrementado cada vez que um novo endereço de retorno é armazenado na pilha e decrementado quando ocorre um retorno.

De uma forma geral, quando a pilha é implementada na memória RAM, temos acesso a ela e ao registro stack pointer. Nesses casos geralmente podemos determinar em que porção da memória colocaremos a pilha. Já nos sistemas onde a pilha é implementada separadamente, geralmente não se tem acesso nem ao conteúdo da pilha nem ao stack pointer.

Voltaremos a tratar de chamadas e desvios em outro momento do curso. Todo esse processo estudado tem por objetivo gerar o endereço de memória de programa de onde será lido uma instrução a ser executada. Uma vez determinado o endereço, seu conteúdo é lido e armazenado em um registro chamado registro de instrução (Instruction Register). Desse registro, parte da instrução vai para o decodificador de instruções e parte pode ir para a ALU e/ou parte para o registro de endereçamento de dados (Data Addressing). Para entender o porque disso, devemos ter em mente que uma instrução nada mais é que uma palavra binária. Parte dela, que é efetivamente a instrução, indica ao decodificador de instruções qual a sequência de ações deve ser executada (qual a "página" do decodificador de instruções). O restante constitui os operandos da instrução, ou seja, os dados a serem processados. Esses dados podem ser constantes ou endereços de variáveis na memória RAM. Conforme sua natureza e a instrução a ser executada eles tem um destino ou outro.

A unidade lógico-aritmética ou ALU (Aritmetic and Logic Unit) é o circuito responsável pelos cálculos em um processador. Como próprio nome diz, ela é responsável pela realização de operações lógicas, (E, OU, OU-exclusivo, deslocamentos, rotações, complemento), e aritméticas (incremento, decremento, adição, subtração, multiplicação, e divisão). Os processos de divisão e multiplicação são feitos com a ALU utilizando sequências somas, subtrações e deslocamento, o que efetivamente o que a ALU é capaz de fazer, ou através de circuito multiplicadores especiais. Existe ainda um circuito, chamado MAC (Multiply and Accumulate) que realiza operações de multiplicação por hardware. Esse bloco, contudo não é comum em microcontroladores, sendo mais comum em dispositivos com maior capacidade de processamento, como DSPs.

A ALU trabalha juntamente com dois registros especiais: o Acumulador (Accumulator) e o registro de estado de operações aritméticas (Status). É comum se referir ao acumulador simplesmente como Acc ou, no caso dos PICs, como W (Worker - Trabalhador). O acumulador quase sempre está envolvido nas operações realizadas pela ALU. Ele pode ser um dos operandos, pode ser onde se armazena o resultado ou pode ser as duas coisas. Há também microcontroladores onde qualquer transferência de dados entre dois endereços da RAM passa pelo acumulador.

Quanto ao registro de status, sua função é indicar resultados notáveis das operações matemáticas. Esses resultados são indicados por flags, que são bits desse registro. Através da análise dos flags é possível saber, dentre outras coisas, se uma operação resultou em zero; se houve estouro da capacidade de armazenamento (overflow), que acontece quando um resultado é maior que o máximo valor possível de ser representado pelo sistema; se o resultado de uma operação aritmética é negativo ou positivo.

Existe uma interação do registro status com o decodificador de instruções, pois através da

análise de seus flags é possível realizar instruções de testes. Por exemplo, existem instruções de desvio condicional nas quais é necessário comparar se um valor é igual a outro. Essas comparações são feitas pela ALU e seu resultado é expresso através do registro status. Através da análise dos bits do registro status o processador toma decisões de realizar ou não desvios no fluxo do programa.

Para acessar a memória de dados e os periféricos existe um registro, que em nosso sistema é chamado de endereçamento de dados (Data Addressing) que pode receber valores de duas formas. A primeira é diretamente de parte da instrução. Nesse caso se está fazendo referência a endereços da RAM conhecidos e fixos, pois são carregados valores constantes existentes no programa. Esse modo é chamado endereçamento direto. Contudo, em muitos casos é necessário fazer referências a endereços variáveis. Isso é feito carregando o registro de endereçamento com dados provenientes de um outro registro, o registro de endereçamento indireto (Indirect Addressing). Como qualquer outro registro, ele pode ser carregado com um valor, constante ou proveniente de uma variável, pode ser incrementado, decrementado ou participar de qualquer operação lógico-aritmética. Sua função é semelhante a dos ponteiros em linguagens de alto nível.

Por fim, apesar de não serem considerado parte integrante da CPU, é necessário comentar alguma coisa sobre os sistemas de temporização e reset. Todo o sistema computacional trabalha sincronizado com um mesmo sinal de clock. Devemos lembrar que esse clock é o que faz o decodificador de instruções passar de uma instrução para a outra, e tudo o mais deve estar sincronizado com ele, senão haveria o caos. Para gerar esse sinal de clock é necessário um oscilador. Nos microcontroladores esse oscilador já faz parte do componente e pode ser de vários tipos, como será tratado oportunamente.

Independente da forma como o clock é gerado, esse sinal é aplicado a CPU e aos periféricos. É comum que a frequência do clock dos periféricos seja menor que a da CPU. Para tanto são utilizados divisores de frequência.

A frequência de clock é diretamente proporcional a velocidade com que a CPU executa as instruções. Os microcontroladores geralmente têm uma faixa de frequência de trabalho bem ampla, podendo ir de alguns Hertz a dezenas e até centenas de Mega-Hertz. É importante lembrar que quanto maior a frequência de trabalho, maior é o consumo do sistema. Para casos em que o consumo é crítico, como em equipamentos portáteis alimentados a bateria, existem várias formas de reduzir este consumo, dependendo do microcontrolador utilizado. São os chamados modos de baixo consumo, que podem reduzir o clock da CPU e/ou dos periféricos ou mesmo desligá-los. Cada microcontrolador apresenta diferentes modos de baixo consumo.

Outro ponto importante é o reset. Além do reset que ocorre quando o sistema é ligado, chamado de Power-on reset, os microcontroladores apresentam várias outras fontes de reset. Esses resets são proteções do sistema. Assim, o microcontrolador pode ser resetado automaticamente se houver uma queda de tensão, se o programa travar, se for acessado um endereço inválido de memória de programa, se ocorrer um estouro da pilha, e em outras situações, dependendo do modelo de microcontrolador.

1.2 Arquitetura Computacional

Por arquitetura de computador entende-se a forma como as diversas partes do sistema são organizadas e conectadas entre si.

A arquitetura apresentada acima é chamada arquitetura Harvard. Sua principal característica é ter a memória de programa separada da memória de dados e acessada por um barramento independente. Existe também a arquitetura von-Neumann, onde memória de dados e de programas estão agrupadas em um mesmo bloco, sendo acessadas pelos mesmos barramentos de dados e endereços.

1.2.1 Arquitetura von-Neumann

Na arquitetura von-Neumann, as memórias tanto de dados quanto de programa, são acessadas usando-se o mesmo barramento de dados; os portais de I/O também fazem uso do barramento de dados. Vale lembrar aqui que uma instrução é como uma operação matemática, isto é, composta de operadores, que indicam o que será feito, e operandos, que são os parâmetros envolvidos na operação. Desta forma, o processo de execução de cada instrução é dividido em dois momentos: a leitura da instrução e dos operandos (fetch) e a execução da instrução propriamente dita. Nota-se que dessa forma o processador está parte do tempo ocupado com a leitura da memória de programa e, consequentemente não fica executando o firmware o tempo todo. Outra característica da arquitetura von-Neumann é que, visto que os operandos das instruções são geralmente do mesmo tamanho do barramento de dados, quanto mais complexa a instrução maior será a quantidade de endereços ocupados por ela na memória. Por outro lado, como a complexidade da instrução não tem limite a não ser o espaço ocupado, podemos ter um set de instruções tão complexo quanto se queira.

Do exposto acima, podemos concluir que arquitetura von-Neumann consome muito tempo de processamento com a leitura da instrução e dos operandos. Conclui-se também que instruções diferentes ocupam quantidades diferentes de memória e são executadas em tempos diferentes.

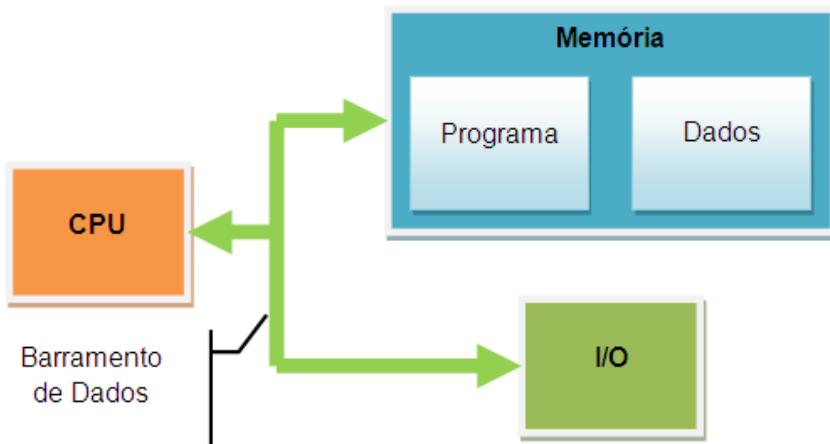


Figura 1.5: Arquitetura von-Neumann

Devido à inexistência de limitações a complexidade das instruções, os sistemas de arquitetura von-Neumann geralmente têm conjuntos de instruções complexos, o que equivale a dizer que possuem um grande número de instruções e cada instrução realiza uma grande sequencia de ações (instruções complexas). Processadores desse tipo são chamados CISC (Complex Instruction Set CPU - CPU com Set de Instruções Complexo).

1.2.2 Arquitetura Harvard

Já a arquitetura chamada de arquitetura Harvard, que é utilizada nos microcontroladores PIC, tem como principal característica acessar a memória de dados separadamente da memória de programa.

A principal vantagem dessa arquitetura é a leitura de instruções e de alguns tipos de operandos pode ser feita ao mesmo tempo em que a execução das instruções. Isso significa que o sistema fica todo o tempo executando instruções, o que acarreta um significativo ganho de velocidade. Enquanto uma instrução está sendo executada, a seguinte está sendo lida. Esse processo é conhecido como pipelining (canalização) e é ilustrado pela figura a seguir.

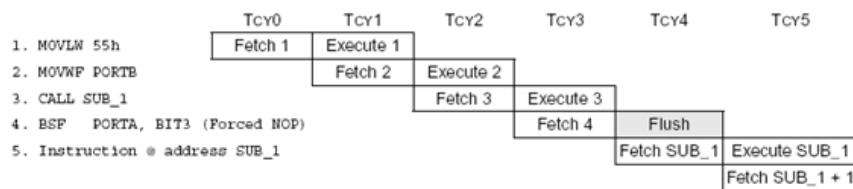


Figura 1.6: Pipeline

Outro fato importante é que o barramento de programa não necessariamente tem o mesmo tamanho do barramento de dados. Dessa forma, em uma única palavra da memória de programa pode conter operando e operadores, o que nos permite carregar toda a instrução em um único ciclo de leitura da memória.

No caso específico dos microcontroladores PIC da família 16Xxxx, o processador foi projetado para que cada instrução ocupe um endereço de memória e seja executada em um ciclo de máquina (que leva 4 períodos de clock para ocorrer), com exceção das instruções de chamada e desvio, que são executadas em dois ciclos. Dessa forma, tem-se um controle do tamanho e, principalmente, do tempo de execução do programa simplesmente contando o número de instruções.

Pode-se notar então que os tempos de execução e de leitura estão atrelados e são os menores possíveis. Isso acarreta em as instruções não poderem executar uma grande sequência de ações, ou seja, não existem instruções complexas. Por isso, os PIC's são considerados processadores RISC (Reduced Instruction Set CPU - CPU com Set de Instruções Reduzido). O número de instruções é reduzido, o que não significa que não se possa executar programas complexos, mas sim que sequências complexas de ações devem ser construídas por sequências de instruções básicas.

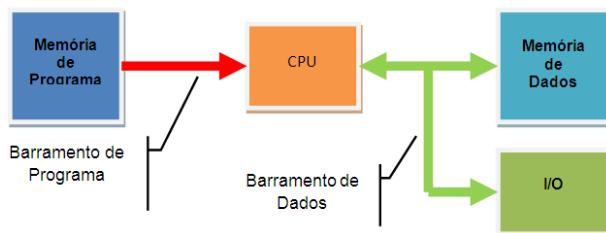


Figura 1.7: Arquitetura Harvard

Para exemplificar, os PIC 18F4550 tem barramento de dados de 8 bits, barramento de instrução de 16 bits e apenas 83 instruções. Essa organização pode ser observada na figura acima.

1.2.3 Microcontrolador, Microprocessador e DSP

Cabem aqui algumas palavras sobre os tipos de sistemas computacionais existentes e suas aplicações.

Os microcontroladores são sistemas computacionais completos em um único chip. Além do processador, das memórias e dispositivos de I/O, os microcontroladores geralmente possuem vários tipos de periféricos. Quando se utiliza microcontroladores é possível construir circuitos de controle com muito poucos componentes, por que o microcontrolador já tem quase todo o necessário internamente. Isso implica em redução de custos e tamanho, menor consumo, facilidade de desenvolvimento, dentre outras vantagens, e é por isso que se utiliza cada vez mais microcontroladores para agregar "inteligência" a diversos produtos. Os microcontroladores, porém, são destinados a aplicações não muito complexas, e tem uma capacidade de processamento limitada. Além disso, já possuem uma estrutura preestabelecida, não sendo possível alterar sua configuração. Atualmente existem microcontroladores de 8, 16 e 32 bits. A maior fatia desse mercado é a dos microcontroladores de 8 bits. Há, porém, um grande crescimento de aplicações em 32 bits. Os microcontroladores de 16 bits estão restritos a aplicações especiais, como o mercado automotivo e tendem a ter seu espaço reduzido pelo fácil acesso à componentes de 32 bits.

Em sistemas construídos com microprocessadores, por sua vez, é necessário desenvolver todo o hardware. Dessa forma existe um maior trabalho de desenvolvimento, porém pode-se construir qualquer configuração que se queira. São sistemas com custo maior, porém com muito mais memória, consequentemente podem executar programas mais complexos. Pode-se dispor de muito mais portais de I/O e de quaisquer periféricos necessários. Além disso, os processadores podem operar com frequências de clock mais altas, portanto são sistemas mais velozes. Sistemas microprocessados são tipicamente de 32 ou 64 bits, sendo processadores de 8 e 16 bits considerados obsoletos hoje. Tem crescido também o uso de processadores dualcore ou multcore, isto é, componentes com dois ou mais processadores operando de forma paralela.

Existem ainda os DSP (Digital Signal Processors - Processadores Digitais de Sinal). São processadores otimizados para realizar cálculos, principalmente somatórios de produtos, que são as operações empregadas em processamento de sinais. Trabalham com frequências de clock elevadas, pois devem realizar grande quantidade de operações em pouco tempo. Geralmente não são os processadores principais do sistema, sendo então controlados por um processador ou microcontrolador. Contudo vêm sendo lançados modelos híbridos de DPP e microcontroladores, que agregam grande capacidade de processamento a sistemas completos com grande número de periféricos em um único chip. Hoje são usados DSPs de 24 e 32 bits, além de controladores híbridos de 16 bits.

Qual desses sistemas utilizar em uma aplicação é uma pergunta que tem como resposta: depende! Um microcontrolador não é capaz de gerenciar um grande banco de dados como um PC; em contrapartida um identificador de chamadas construído com microprocessador teria um preço que inviabilizaria sua venda. Um DSP é tão inadequado para fazer uma central de alarmes como um microcontrolador é para fazer um filtro adaptativo.

Para cada necessidade teremos uma solução diferente. No entanto é importante frisar que a gama de aplicações para microcontroladores é muito grande e cresce a cada dia; eles podem ser encarados como a porta de entrada a esse universo do software embarcado.

Capítulo 2

PIC18, microcontroladores de alto desempenho

2.1 Microcontroladores PIC

Os microcontroladores PICs são divididos em famílias. Cada família, ou linha, tem vários componentes, com tamanhos e recursos diferentes; no entanto o código desenvolvido para um componente de uma determinada família é compatível com os demais componentes da mesma família, exceto por umas poucas alterações, que se referem principalmente aos periféricos. Cada família tem seu próprio set (conjunto) de instruções. Dessa forma, ao se estudar um componente específico de uma família, se está adquirindo conhecimento para trabalhar com microcontroladores de toda a família. Para um mesmo fabricante, as famílias guardam uma semelhança entre si, apesar de não serem exatamente iguais.

As famílias de microcontroladores PIC são:

- PIC10
- PIC12
- PIC14
- PIC16
- PIC17
- PIC18
- PIC24F/PIC24H
- dsPIC30/dsPIC33
- PIC32

Dessas, as mais usadas e com maior número de componentes são as PIC12, PIC16 e PIC18. As famílias PIC14 e PIC17 foram famílias intermediárias na evolução da linha. A família PIC24F/PIC24H é composta por componentes de 16 bits e é a mais recente. Já a família dsPIC30/dsPIC33 trás os controladores híbridos de 16 bits. No final de 2007 foi lançada também a família PIC32, composta

por componentes de 32 bits. Quanto ao tipo de memória, os componentes mais antigos (que ainda continuam sendo fabricados) têm memória do tipo OTP e EPROM. Já os mais novos utilizam somente flash. A tendência do mercado é trabalhar somente com flash, principalmente nas etapas de ensino e desenvolvimento. O tipo de memória flash pode ser identificado por uma letra "F" no nome do componente (por exemplo PIC12F679, PIC16F628, PIC18F4550).

Dentro da família PIC18 há várias sub-famílias, cada uma composta de vários componentes. Os componentes diferem entre si em:

- Quantidade de memória RAM
- Quantidade de memória EEPROM de dados (alguns não tem nada)
- Quantidade de memória Flash de programa
- Número de pinos (18,28,40, ...)
- Frequência máxima de clock
- Periféricos

O quesito "periféricos" é uma das principais características dos PIC. Dentro os disponíveis nas várias famílias, podemos citar: timers, conversores A/D, comparadores analógicos, módulos USART (comunicação serial RS232), Módulos MSSP (I2C e SPI master), módulos CCP (Captura, comparação e PWM), etc...

A linha PIC18 é composta por microcontroladores com core de 8 bits de alto desempenho. Buscou-se manter certa compatibilidade com a linha PIC16, o que pode ser observado no funcionamento de diversos periféricos e no conjunto de instruções. Contudo, muitas mudanças se fizeram necessárias, principalmente na arquitetura e no mapeamento de memória, para superar certas limitações apresentadas pela família PIC16.

A família PIC16 ainda é amplamente utilizada. Contudo, ao estudar uma família mais complexa, como a PIC18, acredita-se que o aluno se tornará apto também a trabalhar com microcontroladores mais simples, como o PIC16 ou PIC12.

Nosso enfoque nesse curso será no PIC18F4550 e na família PIC18. Os assuntos relativos ao hardware serão apresentados de forma generalista para a família PIC18 e depois tratados no que é específico para o PIC18F4550. Dessa maneira pretende-se fazer um estudo não de um componente apenas, mas de toda a família.

2.1.1 A Microchip

O fabricante dos microcontroladores PIC é a empresa americana Microchip. Atualmente ela é uma das maiores fabricantes mundiais de microcontroladores de 8 bits. Além de microcontroladores, a Microchip também fabrica memórias e outros componentes digitais, além de uma vasta linha de componentes analógicos.

A Microchip trabalha com uma política de suporte ao cliente muito eficiente, que provavelmente é uma das causas de seu sucesso. Em seu site (www.microchip.com) existe uma grande quantidade de informação disponível. Além dos manuais dos componentes, existem muitas notas de aplicação (Application Notes) e projetos de referência, que são de grande ajuda para a formação da base de conhecimentos do estudante de microcontroladores PIC.

No Brasil, a Microchip é representada pela Aplicações Eletrônicas Artimar Ltda (www.artimar.com.br) e conta com uma grande rede de revendedores.

2.2 Arquitetura

A figura abaixo apresenta o diagrama em blocos do PIC18F4550. Os demais componentes da família PIC18 apresentam um diagrama muito semelhante, havendo diferença apenas nos número de portais e nos periféricos.

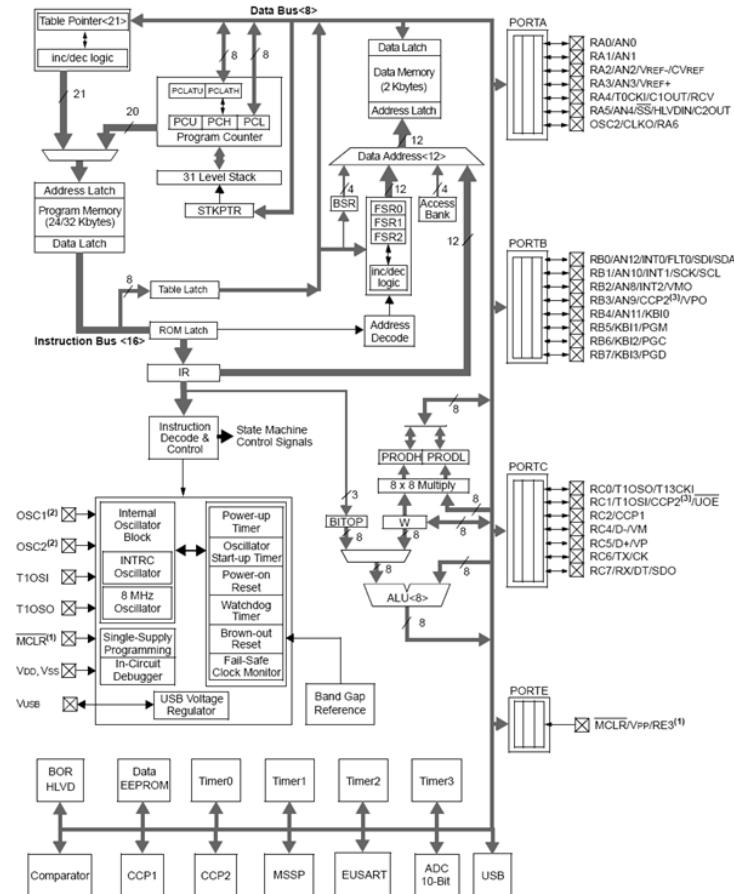


Figura 2.1: Diagrama em blocos do PIC18F4550

Trata-se de um microcontrolador de 8 bits de arquitetura Harvard. Seu barramento de programa é de 16 bits.

Como pode ser observada na figura, a estrutura do PIC18F4550 é semelhante ao sistema genérico estudado. Podem ser observados os barramentos de dados (data bus) e de programa (program bus). Além disso, nota-se como operandos da ALU e de endereçamento da memória de dados vem diretamente da memória de programa.

Também são dignos de nota os periféricos, localizado na parte inferior do diagrama, que estão ligados ao barramento de dados. A principal característica da arquitetura Harvard é ter as memórias de programa e dados separadas, acessadas por diferentes barramentos. Dessa maneira, uma instrução pode ser lida da memória de programa ao mesmo tempo em que a instrução anterior é executada utilizando o barramento de dados, o que é chamado de paralelismo ou pipeline.

O conjunto de instruções do PIC18 foi desenvolvido de forma que a grande maioria das instruções tenha 16 bits de comprimento. Isso garante o uso mais otimizado da memória disponível. O fato da memória de programa ser acessada separada permite que as instruções tenham uma "largura" (número de bits em cada endereço) maior que a memória de dados. No caso do PIC18, a memória de dados (e também o barramento de dados) é de 8 bits enquanto as instruções armazenadas na memória de programa (e também o barramento de instruções) são de 16 bits. O tamanho das instruções está diretamente relacionado com o número de instruções possíveis e com os tipos de operadores suportados. Fazendo uma comparação, a linha PIC16 tinha instruções de 14 bits e um conjunto de 35 instruções; com o aumento para 16 bits na linha PIC18 o conjunto passa a ter um conjunto padrão de 75 instruções e alguns componentes possuem mais 8 instruções (modo estendido), como é o caso do PIC18F4550. Algumas instruções, porém, ocupam 32 bits. Essas instruções, que serão detalhadas no momento oportuno, são importantes principalmente para eliminar a necessidade de paginação de memória, uma inconveniente característica dos PIC16 além de otimizar o modo de endereçamento indireto, o que ajuda o trabalho dos compiladores C. As instruções de 16 bits (que chamaremos de curtas) são executadas em um ciclo de instrução (4 períodos de clock). As instruções de 32 bits (que chamaremos de longas) são executadas em 2 ciclos de instrução. Instruções que causam alteração do fluxo do programa gastam um ciclo de instrução adicional.

O PIC18F4550 foi desenvolvido utilizando a tecnologia nanoWatt da Microchip, tendo disponível diversas funcionalidades que permitem menor consumo de energia.

A tabela a seguir trás as principais características do PIC18F4550.

Característica	PIC18F4550
Frequência de operação	DC a 48MHz
Memória de programa	32768 bytes
Memória de dados RAM	2048 bytes
Memória de dados EEPROM	256 bytes
Fontes de interrupção	20
Terminais de I/O	36
Temporizadores/Contadores	4
CCP	1
ECCP ¹	1
Comunicação Serial	MSSP e EUSART ²
Comunicação USB	Sim
Comunicação Paralela	SPP ³
Conversor analógico para digital	10 bits, 13 canais
Detector de tensão programável	1
Conjunto de instruções	75 convencionais + 8 do modo entendido

Tabela 2.1: Características dos PIC18F4550

¹ O módulo ECCP é um módulo CCP melhorado, com novas características

² O módulo EUSART é um módulo USART melhorado, com novas características

³ Porta paralela mestre

2.2.1 Pinagem e hardware básico

O PIC18F4550 é um componente de 40 pinos em seu encapsulamento PDIP (Plastic Dual In-line Package - Encapsulamento plástico em linha dupla). Esses pinos podem ser divididos em terminais de alimentação, de reset, de conexão com o oscilador e os terminais de portais e periféricos. Como se trata de um componente com diversas características e um pequeno número de terminais, muitos terminais possuem mais de uma função. Dessa forma, temos terminais de entrada e saída dos portais multiplexados com terminais dos periféricos, terminais do oscilador e terminais de reset. De uma forma geral, quando usamos um determinado periférico, o terminal é associado a ele serve ao periférico e sua função de entrada e saída fica desativada. Quando o periférico não é utilizado, o terminal trabalha como I/O.

Os portais do microcontrolador podem variar de funções de modelo para modelo, mas geralmente algumas funcionalidades se repetem sempre para pinos com o mesmo nome. Consulte o manual do microcontrolador para saber quais as funções associadas a cada pino.

Os drivers de entrada podem ser TTL ou Schmitt Trigger (ST), podendo um mesmo pino ter drivers diferentes para funções diferentes.. As saídas são do tipo CMOS, embora exista um terminal (RA4) com saída com dreno aberto (OD - open drain). Esse terminal, para ser usado como saída, precisa de um resistor de pull-up. Há também os drivers especiais para osciladores (XTAL). Além, disso, alguns periféricos possuem entradas analógicas (AN).

A tabela que segue apresenta as características elétricas desses drivers para o microcontrolador operando alimentado por 5V.

Parâmetro	Tipo de Buffer	Min	Max
Vil	TTL	Vss	0.8
	Schmitt Trigger	Vss	1
Vih	TTL	2,0	Vdd
	Schmitt Trigger	4,0	Vdd
Vol	CMOS	-	0,6
VOH	CMOS	4,3	-

Tabela 2.2: Características elétricas de entrada e saída

Os terminais do oscilador podem ser utilizados como I/O, dependendo do modo de oscilador utilizado, conforme estudado. O terminal de reset (MCLR) pode ser configurado como entrada de um portal no momento da gravação.

Quanto à capacidade de corrente, cada terminal configurado como saída é capaz de fornecer ou drenar 25mA. Contudo, a corrente (fornecida ou drenada) total de todos os terminais juntos não pode ultrapassar 200mA.

A figura a seguir apresenta a pinagem do PIC18F4550.

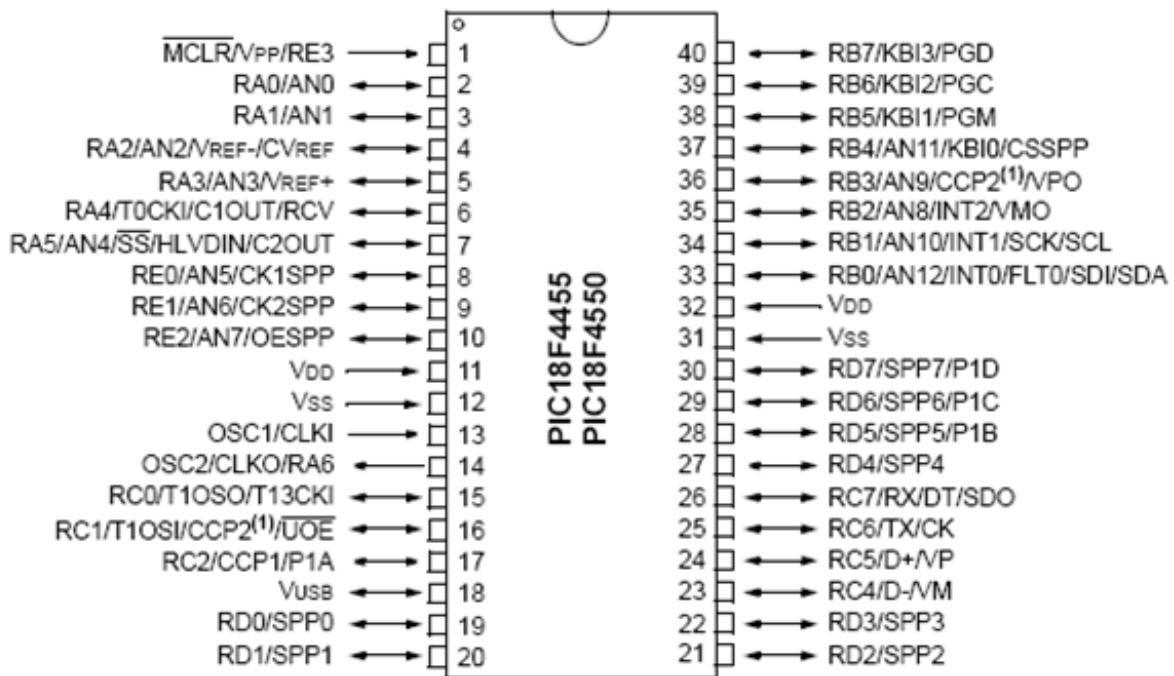


Figura 2.2: Pinagem do PIC18F4550

2.3 Memórias

Por ser um microcontrolador de arquitetura Harvard, o PIC18 tem suas memórias de programa e dados mapeadas separadamente.

2.3.1 Memória de programa

A arquitetura do PIC18 traz um PC (Program Counter - Contador de programa) de 21 bits, portanto é possível endereçar até 2MBytes de endereços de memória de programa. Cada componente da família, porém, tem diferentes quantidades de memória implementadas. No caso do 0 existem 32kB de memória FLASH de programa.

A memória de programa é mapeada de forma que cada endereço tenha 8 bits (1 byte). Isso é feito de forma a facilitar o uso da memória de programa para armazenamento de constantes (de forma a facilitar a transferência para o barramento de dados) e o uso de áreas da flash não ocupadas por programa como se fossem EEPROM de dados. Contudo, as instruções armazenadas na memória de programa têm 16 ou 32 bits. O que acontece na verdade é que cada instrução ocupa dois endereços de memória e o PC, quando executando um programa, incrementa de 2 em 2 e o barramento de instrução permite a leitura simultânea de dois endereços de memória, formando uma instrução de 16 bits (instruções curtas). Em virtude disso o endereço das instruções é sempre um número par. Para a leitura das instruções longas (32 bits) são feitos dois acessos de leitura.

Para o uso da memória de programa para o armazenamento de dados existem instruções especiais que realizam as chamadas operações de tabela. O ponteiro utilizado nessas instruções é o TBLPTR que tem 21 bits. Através dele é possível realizar a leitura e escrita de qualquer endereço da memória de programa.

Resumindo: Para a leitura de instruções o endereçamento é feito com 20 bits e a memória é vista como tendo 16 bits de largura. Para operações de tabela o endereçamento é feito com 21 bits e a memória é vista como tendo 8 bits de largura.

Uma importante novidade no PIC18 é a inexistência de paginação de memória que existia na família PIC16. As instruções CALL e GOTO contêm um campo de endereço de 20 bits (para endereço de instruções o bit menos significativo é sempre 0), portanto capaz de acessar qualquer ponto da memória. Contudo, como se tratam de instruções longas devem ser usadas com parcimônia para uma maior economia de memória. Existem instruções para chamada e desvio relativos (RCALL e BRA, respectivamente) que acessam endereços a uma distância de até 1k do endereço atual e ocupam apenas uma posição de memória.

O vetor de reset para o PIC18F continua sendo o endereço 0000h, como era no PIC16.

Como será tratado no momento oportuno, existem dois vetores de interrupção. O vetor de maior prioridade é o 0008h o de menor é o 0018h.

A figura a seguir apresenta o mapa de memória de programa do PIC18F4550. Leituras realizadas em endereços não implementados de memória retornam 0.

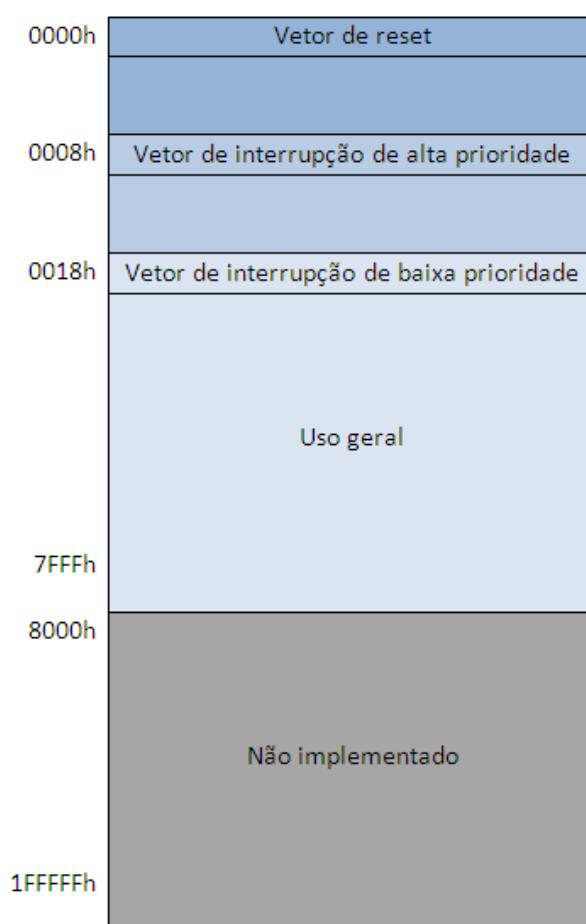


Figura 2.3: Mapa e memória de programa

Contador de programa e Pilha

O PC do PIC18 tem 21 bits. Ele é dividido em três registros de 8 bits: PCL, PCH e PCU e pode ser manipulado a partir da memória de dados através de PCLATU, PCLATH e diretamente através de PCL. A figura apresenta a como esses registros interagem.

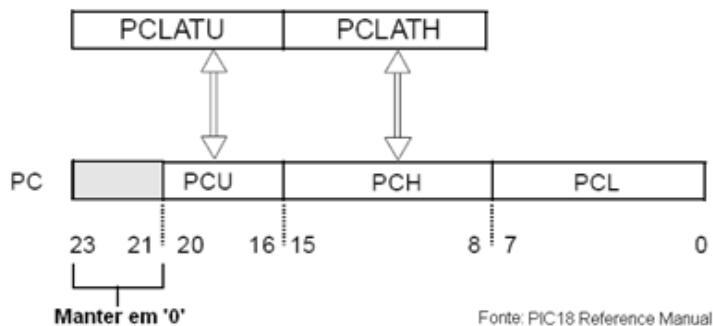


Figura 2.4: Estrutura do PC

De forma similar ao que acontece no PIC16, toda vez que uma operação de escrita é realizada em PCL os valores de PCLATH e PCLATU são carregados em PCH e PCU, respectivamente, causando um desvio imediato. A novidade é que uma leitura do registro PCL faz com que os valores de PCH e PCU sejam transmitidos para o PCLATH e PCLATU, respectivamente. Essa funcionalidade é muito interessante, principalmente na implementação de RTOS (Real Time Operation System - Sistema Operacional de Tempo Real). O bit 0 de PCL não pode ser escrito, ficando sempre em 0.

A pilha do PIC18 é um espaço de memória de 31 posições de 21 bits. Ela não está mapeada nem em RAM nem em FLASH. Trata-se de uma pilha circular, ou seja, o 32º endereço gravado sobrescreve o primeiro, caracterizando um overflow da pilha. Se instruções de retorno forem dadas quando o primeiro nível da pilha estiver livre considera-se um underflow da pilha. Pode-se configurar o microcontrolador para que esses dois eventos causem reset.

O ponteiro da pilha (Stack Pointer) é acessível através do registro STKPTR, o que não acontece na arquitetura PIC16. Nesse registro há também bits que sinaliza o estado da pilha. O ponteiro da pilha contém o endereço do topo da pilha, que recebe o último endereço de retorno guardado.

R/C = 0	R/C = 0	U = 0	R/W = 0				
STKOVF	STKUNF	-	SP4	SP3	SP2	SP1	SP0
Bit 7							Bit 0

Figura 2.5: Ponteiro da pilha

- STKOVF : indicador de overflow
 - 1 = a pilha está cheia ou houve overflow
 - 0 = a pilha não está cheia e não houve overflow.
- STKUNF : indicador de underflow.
 - 1 = Houve underflow da pilha

– 0 = Não houve underflow da pilha

- **SP4:SP0 :** São o ponteiro da pilha propriamente dito.

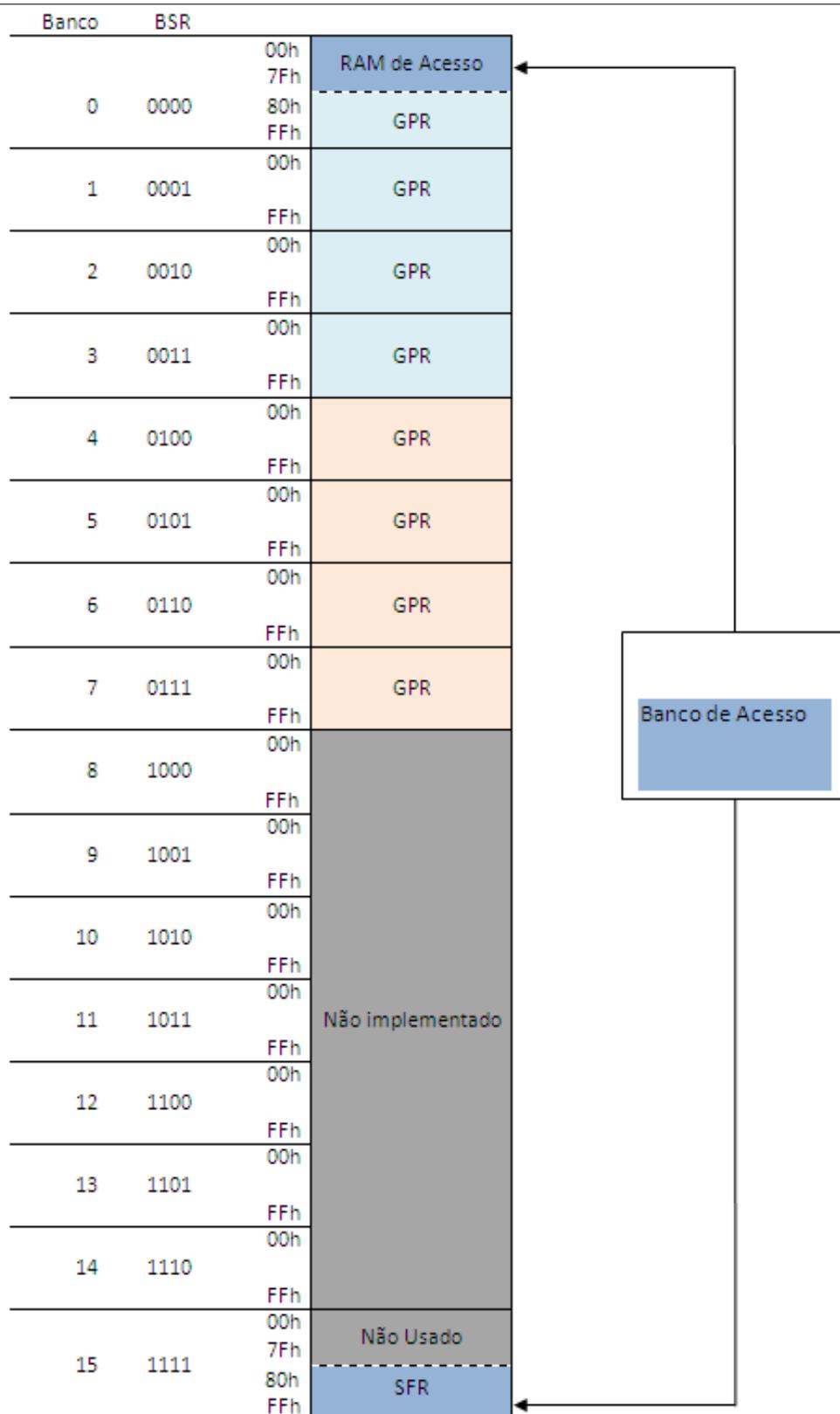
O valor contido no endereço apontado pelo ponteiro, portanto o último endereço salvo na pilha, é chamado de topo da pilha (TOS - Top of Stack). No PIC18 o conteúdo desse endereço é acessível através dos registros de função especial TOSU, TOSH e TOSL. É possível então ler o conteúdo do topo da pilha e alterá-lo. Além disso, existem as instruções PUSH e POP. PUSH incrementa o valor o ponteiro da pilha e carrega o endereço seguinte no TOS. POP decrementa o ponteiro, descartando endereço contido no TOS. Essas funcionalidades são bastante úteis na implementação de um sistema operacional.

2.3.2 Memória de dados

A memória de dados apresenta alterações bastante significativas em relação ao PIC16. A principal diferença está no sistema de paginação. A arquitetura do PIC18 prevê 12 bits para endereçamento de memória de dados, o que permite trabalhar com até 4096 endereços de memória, divididos entre registros de função especial (SFR - Special Function Registers) e registros de uso geral (GPR - General Purpose Registers). As instruções que manipulam dados na memória RAM (com exceção de MOVFF) fornecem 8 bits para compor esse endereço; os outros quatro são fornecidos pelo registro de seleção de banco: BSR.

Esse esquema de acesso parece ser o mesmo usado no PIC16, que apresentava diversas inconveniências em sua aplicação. Há, porém algumas inovações que facilitam o seu uso. Em primeiro lugar, existe a instrução MOVFF é capaz de mover dados entre dois registros em quaisquer bancos; isso é possível porque esta é uma instrução longa (32 bits) e contém os endereços de 12 bits dos dois registros acessados. As demais instruções necessitam da seleção de banco para acesso aos registros. Isso é facilitado pela instrução MOVLB, que move um valor constante diretamente para BSR. Existem ainda instruções especiais para o modo indexado e o banco de acesso, descritos a seguir.

A seguir é apresentado o mapeamento de memória de dados válido para o PIC18F4550:



SFR: Special Function Register – Registros de Função Especial

Figura 2.6: Mapeamento de memória de dados para PIC18F4550

SFR - Registros de função especial

A figura a seguir apresenta os registros de função especial existentes no PIC18F4550. Para quem conhece o PIC16 pode parecer que eles estão distribuídos em quatro bancos, pela semelhança na forma de apresentar, mas na verdade estão todos em um único banco.

Address	Name	Address	Name	Address	Name	Address	Name	Address	Name
FFFh	TOSU	FDFh	INDF2 ⁽¹⁾	FBFh	CCPR1H	F9Fh	IPR1	F7Fh	UEP15
FFEh	TOSH	FDEh	POSTINC2 ⁽¹⁾	FBEh	CCPR1L	F9Eh	PIR1	F7Eh	UEP14
FFDh	TOSL	FDDh	POSTDEC2 ⁽¹⁾	FBDh	CCP1CON	F9Dh	PIE1	F7Dh	UEP13
FFCh	STKPTR	FDCh	PREINC2 ⁽¹⁾	FBCh	CCPR2H	F9Ch	__(2)	F7Ch	UEP12
FFBh	PCLATU	FDBh	PLUSW2 ⁽¹⁾	FBBh	CCPR2L	F9Bh	OSCTUNE	F7Bh	UEP11
FFAh	PCLATH	FDAh	FSR2H	FBAh	CCP2CON	F9Ah	__(2)	F7Ah	UEP10
FF9h	PCL	FD9h	FSR2L	FB9h	__(2)	F99h	__(2)	F79h	UEP9
FF8h	TBLPTRU	FD8h	STATUS	FB8h	BAUDCON	F98h	__(2)	F78h	UEP8
FF7h	TBLPTRH	FD7h	TMR0H	FB7h	ECCP1DEL	F97h	__(2)	F77h	UEP7
FF6h	TBLPTRL	FD6h	TMR0L	FB6h	ECCP1AS	F96h	TRISE ⁽³⁾	F76h	UEP6
FF5h	TABLAT	FD5h	T0CON	FB5h	CVRCON	F95h	TRISD ⁽³⁾	F75h	UEP5
FF4h	PRODH	FD4h	__(2)	FB4h	CMCON	F94h	TRISC	F74h	UEP4
FF3h	PRODL	FD3h	OSCCON	FB3h	TMR3H	F93h	TRISB	F73h	UEP3
FF2h	INTCON	FD2h	HLVDCON	FB2h	TMR3L	F92h	TRISA	F72h	UEP2
FF1h	INTCON2	FD1h	WDTCON	FB1h	T3CON	F91h	__(2)	F71h	UEP1
FF0h	INTCON3	FD0h	RCON	FB0h	SPBRGH	F90h	__(2)	F70h	UEP0
FEFh	INDF0 ⁽¹⁾	FCFh	TMR1H	FAFh	SPBRG	F8Fh	__(2)	F6Fh	UCFG
FEEh	POSTINC0 ⁽¹⁾	FCEh	TMR1L	FAEh	RCREG	F8Eh	__(2)	F6Eh	UADDR
FEDh	POSTDEC0 ⁽¹⁾	FCDh	T1CON	FADh	TXREG	F8Dh	LATE ⁽³⁾	F6Dh	UCON
FECh	PREINC0 ⁽¹⁾	FCCh	TMR2	FACH	TXSTA	F8Ch	LATD ⁽³⁾	F6Ch	USTAT
FEBh	PLUSW0 ⁽¹⁾	FCBh	PR2	FABh	RCSTA	F8Bh	LATC	F6Bh	UEIE
FEAh	FSR0H	FCAh	T2CON	FAAh	__(2)	F8Ah	LATB	F6Ah	UEIR
FE9h	FSR0L	FC9h	SSPBUF	FA9h	EEADR	F89h	LATA	F69h	UIE
FE8h	WREG	FC8h	SSPADD	FA8h	EEDATA	F88h	__(2)	F68h	UIR
FE7h	INDF1 ⁽¹⁾	FC7h	SSPSTAT	FA7h	EECON2 ⁽¹⁾	F87h	__(2)	F67h	UFRMH
FE6h	POSTINC1 ⁽¹⁾	FC6h	SSPCON1	FA6h	EECON1	F86h	__(2)	F66h	UFRML
FE5h	POSTDEC1 ⁽¹⁾	FC5h	SSPCON2	FA5h	__(2)	F85h	__(2)	F65h	SPPCON ⁽³⁾
FE4h	PREINC1 ⁽¹⁾	FC4h	ADRESH	FA4h	__(2)	F84h	PORTE	F64h	SPPEPS ⁽³⁾
FE3h	PLUSW1 ⁽¹⁾	FC3h	ADRESL	FA3h	__(2)	F83h	PORTD ⁽³⁾	F63h	SPPCFG ⁽³⁾
FE2h	FSR1H	FC2h	ADCON0	FA2h	IPR2	F82h	PORTC	F62h	SPPDATA ⁽³⁾
FE1h	FSR1L	FC1h	ADCON1	FA1h	PIR2	F81h	PORTB	F61h	__(2)
FE0h	BSR	FC0h	ADCON2	FA0h	PIE2	F80h	PORTA	F60h	__(2)

Note 1: Not a physical register.

2: Unimplemented registers are read as '0'.

3: These registers are implemented only on 40/44-pin devices.

Figura 2.7: Registros de funções especiais existentes no PIC18F4550

Banco de acesso

Para agilizar operações de manipulação de dados entre variáveis e os registros de uso especial foi implementado um sistema de banco de acesso. Trata-se de uma opção que pode ser informada nas instruções de manipulação de dados que habilita o uso de um sistema alternativo de acesso a memória. Quando uma instrução usa esse sistema o registro BSR é ignorado e "visto" como um único banco de 256 endereços, cujos 128 primeiros endereços são os 128 primeiros do banco

(chamados de RAM de Acesso) enquanto os 128 últimos são os 128 endereços finais do banco 15 onde estão os registros de funções especiais (SFR).

Memória RAM para USB

Quando o periférico USB é usado os bancos 4, 5, 6 e 7 são usados como buffer da comunicação USB. É usando um sistema chamado memória RAM de porta dupla (dual port RAM) onde tanto o periférico USB como o core do PIC tem acesso de leitura e escrita para troca de informações.

Quando não é usada comunicação USB esses bancos podem ser usados como RAM de uso geral.

Modos de endereçamento

A arquitetura do PIC18 possui quatro modos de endereçamento: Inerente, Literal, Direto e Indireto.

O Modo Inerente é aquele das instruções que não necessitam da indicação do operando. O operando é "inerente" à instrução (daí o nome). Exemplos: CLRWDT, DAW, PUSH.

O Modo Literal é aquele onde o operando é um valor constante (literal). Nesse caso o operando é armazenado como parte da instrução na memória flash. Ex.: MOVLB, ADDLW, GOTO.

No modo Direto os operandos da instrução são dados da memória RAM cujos endereços são fixos. Esses endereços fazem parte da própria instrução. Ex.: MOVFF, BSF, ADDWF.

Ao contrário do modo direto, o modo indireto manipula dados de memória RAM de endereços variáveis. O dado é acessado através dos registros INDFx e seu endereço é apontado pelo registro FSRx. Existem três conjuntos de registro FSRx formando três ponteiros de 16 bits (FSR0H:FSR0L, FSR1H:FSR1L e FSR2H:FSR2L). Juntamente com cada par FSRx trabalha um registro INDFx (INDF0, INDF1e INDF2).

2.4 Processador

O processador (ou CPU) do PIC18 executa as instruções e controla todo o hardware do microcontrolador. Ele é frequentemente chamado de core, isto é, o núcleo do microcontrolador.

O processador necessita para trabalhar de um sinal chamado clock de sistema, que é gerado pelo sistema de oscilador. Refere-se ao período do clock de sistema com TSCLK. Cada instrução é executada em 4 ciclos de clock do sistema e esse período do ciclo de instrução é chamado TCY. Portanto, $TCY = 4 \cdot TSCLK$.

Cada instrução executada pelo oscilador gasta 4 ciclos de clock. Esses quatro ciclos, chamados ciclos Q, compõem um ciclo de instrução. Surge então um sinal, cuja frequência é $\frac{1}{4}$ da frequência do oscilador e é chamado de clock de instrução ou clock de periféricos. Ou seja, o tempo de execução de cada instrução é de 4 períodos de clock. Esse é o sinal disponível no terminal RA6/OSC2/CLKOUT quando trabalhamos em um modo de oscilador com saída de clock.

Em cada um dos ciclos Q é realizada uma etapa da instrução, conforme mostrado na tabela.

Ciclo	Ação
Q1	Ciclo de decodificação da instrução
Q2	Ciclo de leitura de dados
Q3	Ciclo de processamento
Q4	Ciclo de escrita de dados

Tabela 2.3: Ciclos "Q" do microcontrolador

A figura abaixo apresenta um diagrama de tempos do sinal de clock e do ciclo de instrução.

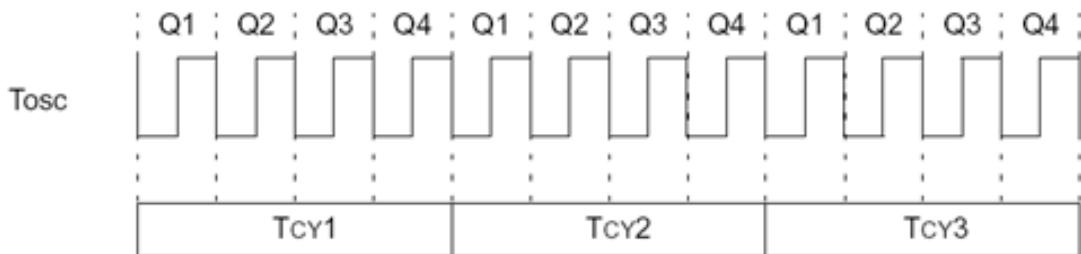


Figura 2.8: Ciclos de instrução

Alguns periféricos e algumas funções do PIC não são sincronizados nem com o clock do oscilador nem com o clock de periféricos, mas com os ciclos Q.

2.4.1 ALU

A ALU (Arithmetic Logical Unit - Unidade Lógico-Aritmética) do PIC18 é capaz de realizar operações de soma, subtração, deslocamento e operações lógicas. Todas as operações têm operandos e resultados de 8 bits. Associada a ALU trabalham dois registros: o registro W, chamado também de WREG está sempre envolvido nas operações matemáticas, podendo ser um dos operandos e/ou o destino do resultado; o registro STATUS informa os resultados notáveis das operações matemáticas através de flags. A figura 2.9 apresenta o diagrama em blocos da ALU. Observe que podem ser operandos valores presentes na memória RAM, constantes vindas das instruções e o próprio WREG.

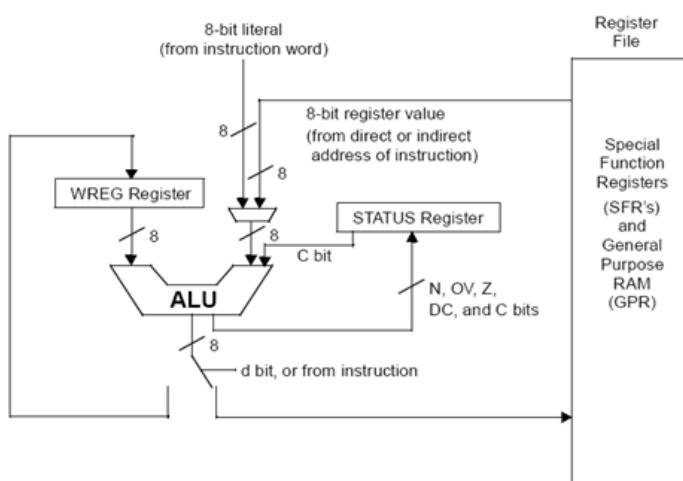


Figura 2.9: ALU

A seguir é apresentado o registro STATUS bem como a interpretação dos bits de flag.

Figura 2.10: Registro STATUS / Bits de flag

- **N** : Negativo, indica quando o resultado de uma operação sinalizada é 0 (cópia do bit 7 do resultado):
 - 1 = resultado negativo;
 - 0 = resultado positivo;
 - **OV** : Overflow, indica a ocorrência de "vai 1" do bit 6 para o bit 7, indicando que houve estouro da capacidade do valor absoluto (7 bits) em operações sinalizadas:
 - 1 = houve overflow para operações sinalizadas;
 - 0 = não houve overflow;
 - **Z**: Bit indicador de zero:
 - 1 = se o resultado de uma operação da ALU é zero;
 - 0 = se o resultado de uma operação da ALU é diferente de zero;
 - **DC**: Carry/Borrow de dígito. Para borrow o bit é inverso:
 - 1 = houve transporte do 4o. para o 5o. bit;
 - 0 = não houve transporte do 4o. para o 5o. bit;
 - **C**: Carry/Borrow . Para borrow o bit é inverso:
 - 1 = houve transporte para 8o. bit;
 - 0 = não houve transporte para 8o. bit;

Multiplicador 8x8 bits

Além das funções básicas da ALU, os PIC18 são equipados com um hardware multiplicador com capacidade de fazer operações com operandos de 8 bits, gerando resultados de 16 bits. O PIC18 possui instruções de multiplicação que, ao contrário do que acontece em alguns processadores, não são implementadas por operações básicas da ALU, mas sim através de um hardware específico. A principal vantagem disso é que as multiplicações (de 8 por 8 bits) são realizadas em um único ciclo de instrução, tornando o microcontrolador muito mais rápido para a execução de qualquer tipo de multiplicação. O multiplicador por hardware, juntamente com a maior velocidade de processamento, permite ao PIC18 implementar algoritmos de processamento de sinal, além de reduzir a ocupação de memória de programa para essas operações.

A instrução MULLW realiza a multiplicação de W por uma constante enquanto MULWF realizada a multiplicação de W por um endereço de memória RAM. Nos dois casos o resultado (de 16 bits) é armazenado no SFRs PROFH e PRODL.

A operação de multiplicação não afeta os flags do registro STATUS.

2.5 Portais de I/O

A configuração dos portais de I/O do PIC18 é muito semelhante a configuração do PIC16, porém algumas importantes diferenças existem.

O processo de definição da direção dos pinos (se entrada ou saída) é feito através do registro TRISx associado a cada portal (TRISA, TRISB, etc), onde '1' em um bit no registro TRISx faz o bit correspondente do portal ser uma entrada enquanto um '0' em um bit no registro TRISx faz o bit correspondente do portal ser uma saída.

Nos processos de escrita e leitura dos portais aparecem as diferenças entre o PIC16 e PIC18. A figura 2.11 traz o diagrama em blocos genérico para um portal e ajuda a compreender esses processos.

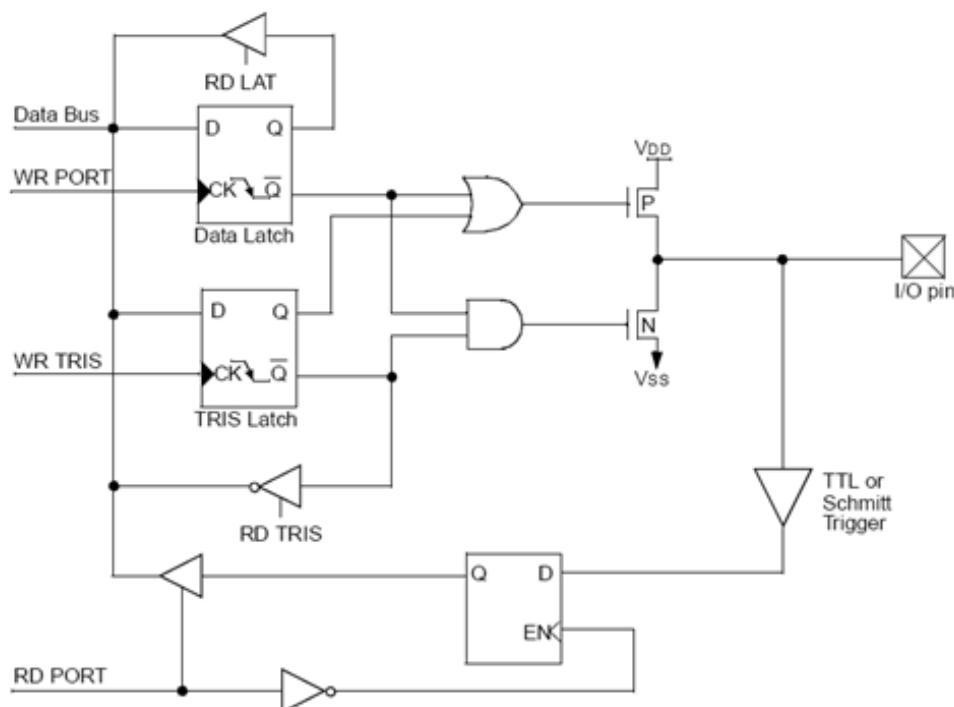


Figura 2.11: Esquema genérico de um pino

Além dos registros PORTx existem no PIC18 os registros LATx (LATA, LATB, LATC, etc). Os registros LATx corresponde ao Data Latch da figura 2.11.

Em um pino configurado como saída uma operação de escrita no registro LATx gera o mesmo resultado de uma escrita no registro PORTx, isto é, o dado armazenado no Data Latch é transferido para o pino do microcontrolador. Se o pino está configurado como entrada a escrita tanto no registro LATx como no PORTx também tem o mesmo resultado, o valor é armazenado no Data Latch mas o driver de saída permanece desabilitado.

Na leitura, porém, os resultados divergem. Primeiro, admite-se que um dado pino está configurado como entrada. Se for realizada uma leitura do registro PORTx o valor de retorno é o estado real do pino enquanto a leitura do registro LAT retorna o valor anteriormente escrito no Data Latch . Admitindo agora que o pinos esteja configurado com saída, a leitura do registro LATx traz o valor que deveria estar no pino enquanto a leitura de PORTx traz o valor que realmente está no pino. Dessa forma é possível detectar a existência de curtos circuitos ou outros defeitos de hardware.

Assim como nos PIC16, cada pino dos PIC18 é capaz de drenar ou fornecer até 25mA, sendo respeitadas as limitações de correntes para cada portal, apresentadas pelo manual de cada componente.

2.6 Conjunto de Instruções

O conjunto de instruções do PIC18 é um de seus pontos de maior destaque por suas diversas características:

- Semelhança com o conjunto de instruções do PIC16, tanto no mnemônico das instruções quanto na sintaxe, o que garante rápida a adaptação do desenvolvedor e facilita a tradução de código entre as famílias PIC16 e PIC18;
- 75 instruções no modo normal e mais 8 instruções no modo estendido.
- Instruções otimizadas para uso de ponteiros (facilita o uso de compiladores C)
- Instruções longas (32 bits) que solucionam problemas de paginação de memória de programa e memória de dados
- Instruções de desvio condicional

O modo estendido é habilitado via bits de configuração e permite que 8 instruções a mais sejam usadas. Essas instruções permitem operações mais avançadas nos modos de acesso indireto e operações de tabela. Nem todos os microcontroladores da linha PIC18 têm o conjunto de instruções estendido.

Como sabemos, instruções são compostas por uma parte que diz o que deve ser feito (que chamaremos operador) e uma parte que diz com quem deve ser feito (que chamaremos operandos). Para compreender o conjunto de instruções do PIC18 é preciso saber que muitas das operações realizadas vão ter como um de seus operandos um registro de função especial chamado W. Ele também é geralmente o destino do resultado de muitas operações, portanto sua função se confunde com a do chamado acumulador existente em outras arquiteturas. Portanto, ao somar dois valores, geralmente somamos um valor (constante ou variável) a W, podendo o resultado armazenado no próprio W.

No Anexo E desta apostila está um quadro resumo do conjunto de instruções do PIC18. Informações mais detalhadas podem ser obtidas no manual do PIC18F4550.

Neste curso não será dado maior enfoque a linguagem Assembly, já que nos concentraremos na linguagem C. Contudo, é importante saber que é possível (e algumas vezes necessário) introduzir trechos de código assembly em programas escritos em C.

2.7 Oscilador

Como o funcionamento de um sistema computacional é baseado numa sequencia de ações, essas ações devem ser executadas numa certa cadênciia. Para obter essa cadênciia é utilizado um sinal de sincronismo ou clock.

Os microcontroladores PIC possuem um circuito de oscilação interna, isto é, são capazes de gerar seu próprio clock com acréscimo de poucos (ou nenhum) componentes externos. Existem diversos tipos diferentes de oscilador, e cada componente da família utiliza alguns deles.

A frequência máxima do PIC18F4550 é de 48MHz. Em muitos outros componentes da família PIC18 essa frequência está limitada a 40 MHz. Por fim, a família PIC16 aceita frequência máxima de 20MHz.

O PIC18 apresenta os seguintes tipos modos de oscilador (nem todos eles presente em todos os componentes da linha):

- Modos a cristal/ressonador cerâmico:
 - LP: Cristal, baixas frequências (Low Frequency (Power) Crystal);
 - XT: Cristal, frequências intermediárias (Crystal/Resonator);
 - HS: Cristal, altas frequências(High Speed Crystal/Resonator);
 - HSPLL: Cristal de alta frequência com PLL (High Speed with Phase Locked Loop)
- Oscilador RC Externo (External Resistor/Capacitor);
 - Com ou sem saída de clock
 - Necessita de um resistor (opcionalmente um capacitor) externo
- Oscilador RC Interno (Internal Resistor/Capacitor);
 - Com ou sem saída de clock
 - Oscilador interno do microcontrolador, sem a necessidade de componentes adicionais.
- Entrada de clock externo (External Clock in)
 - Com ou sem saída de clock;
 - Clock proveniente de outras partes do circuito..

A família PIC18 prevê 3 fontes possíveis de clock: primário, secundário e interno. A fonte primária pode ser um dos modos de oscilador a cristal, clock externo ou mesmo o oscilador interno. A fonte secundária é um segundo oscilador do circuito, ligado ao Timer 1 (trataremos com mais detalhes desse oscilador quando o Timer 1 for estudado). A fonte interna é a opção de se usar o oscilador interno. É possível mudar de fonte de clock a qualquer momento durante a execução do clock. Isso permite uma série de possibilidades relacionadas com economia de energia, como será tratado oportunamente.

O PIC18F4550 possui um circuito oscilador especial para atender a uma demanda de frequência para a comunicação USB. Para o correto funcionamento da porta USB é necessário um sinal de 48 MHz (para USB de alta velocidade) ou 6MHz (para USB de baixa velocidade). Isso implica em um sistema de geração e seleção de clock mais complexo do que o encontrado em outros componentes da família PIC18. O diagrama em blocos abaixo mostra esse sistema de forma simplificada.

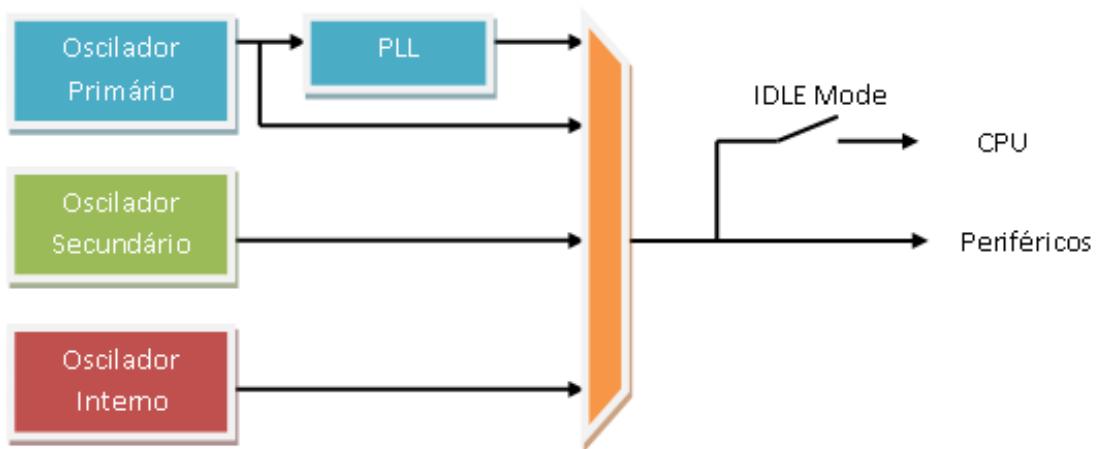


Figura 2.12: Sistema de oscilação

A qualquer momento é possível selecionar qual dos osciladores está sendo usado como fonte de clock para a CPU e periféricos. Existe ainda um PLL associado ao oscilador primário, que será detalhado mais a frente, mas por hora pode ser entendido como um multiplicador de frequências. Deve ser notado o detalhe da chave que permite interromper o clock para CPU; essa chave é a atuação do modo IDLE que será tratado quando forem estudados os modos de baixo consumo.

2.7.1 Oscilador a cristal (LP,XT e HS)

Os modos LP, XT e HS utilizam um cristal ou ressonador cerâmico para estabilizar o clock. A diferença está na faixa de frequência de cada modo.

O modo LP refere-se a cristal/ressonador cerâmico de baixa frequência. Esse modo trabalha com frequências de oscilação de até 200 kHz e nele conseguimos o menor consumo dos três modos em questão.

O modo XT abrange as frequências de 200 kHz até 4 MHz e apresenta um consumo médio.

O modo HS é para frequências de 4 a 25 MHz e tem o mais alto consumo. Cada faixa de frequência possui um ganho do circuito oscilador, que é ajustado de forma diferente para cada modo. É recomendado que se use o modo adequado para cada frequência, caso contrário pode ocorrer mau funcionamento do oscilador. Além do cristal ou do ressonador, nos modos LP, XT e HS, devem existir capacitores ligados dos terminais do cristal terra, conforme mostra a figura.

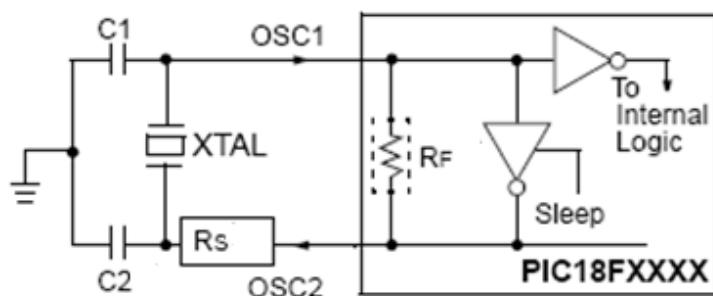


Figura 2.13: Circuito do Oscilador a Cristal/Ressonador

O PIC18F4550 não possui o modo LP, porém outros componentes da família sim. Os valores desses capacitores são os indicados nas tabelas Capacitor Selection for Crystal Oscillator que se

encontra nos manuais de cada PIC na seção que trata de configurações do oscilador. Para o PIC18F4550 os valores são os apresentados na tabela abaixo.

Atenção: esses valores podem mudar conforme o microcontrolador.

Tipo de oscilador	Frequência do cristal	Faixa de capacitor C1	Faixa de capacitor C2
XT	4MHz	27pF	27pF
HS	4MHz	27pF	27pF
	8MHz	22pF	22pF
	20MHz	15pF	15pF

Tabela 2.4: Seleção de capacitores para oscilador a cristal

Capacitores maiores aumentam a estabilidade do oscilador, mas também aumenta o tempo de partida. O tempo de partida do oscilador é o tempo necessário para que o oscilador se estabilize, pois a oscilação não ocorre de forma imediata assim que o circuito é alimentado. Esse tempo depende de uma série de fatores, além dos capacitores, com temperatura, tensão de alimentação e até capacidades parasitas do lay-out da placa.

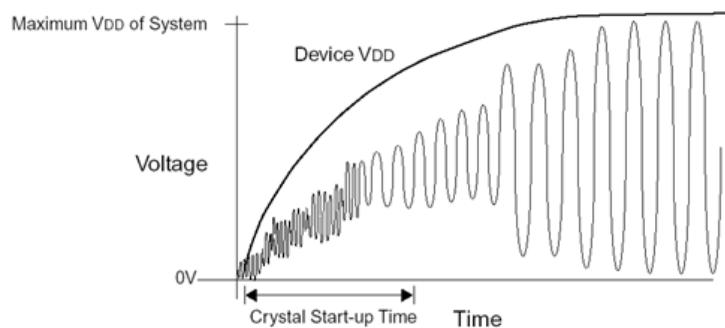


Figura 2.14: Partida do oscilador

2.7.2 PLL - Phase Locked Loop

O modo HSPLL permite utilizar o sinal de clock do modo HS, INTIO1 e INTIO2 como sincronismo para um circuito de PLL (Phase Locked Loop) que gera em sua saída um sinal com quatro vezes a frequência do sinal de referência.

PLL (Phase Locked Loop) é um circuito que permite gerar um sinal de alta frequência baseado em um sinal preciso de baixa frequência. Para entendê-lo será estudado o sistema de PLL comumente encontrado em microcontroladores PIC18 na figura a seguir.

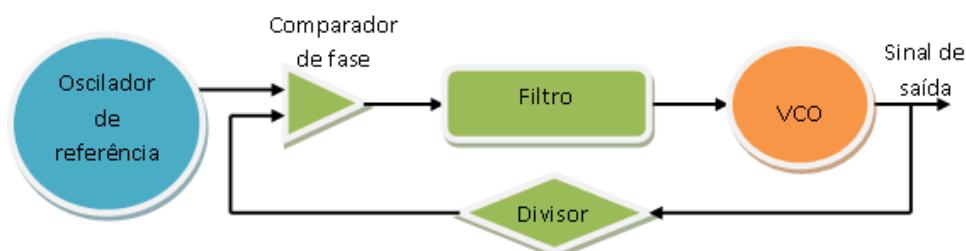


Figura 2.15: Esquema básico de um PLL

Existe um oscilador o cristal que é o oscilador de referência e garante a precisão ao sistema. O sinal desse oscilador é comparado com uma amostra do sinal de saída através de um comparador de fase. Caso existe alguma diferença entre essa amostra e o oscilador de referência um sinal de erro é gerado e filtrado e aplicado ao VCO (Voltage Controlled Oscillator - Oscilador controlado por tensão). O VCO gera então um sinal de saída conforme comandando pelo sinal de erro. Esse sinal passa por um divisor de frequência para gerar a amostra a ser comparada com o oscilador de referência. Como a precisão do sinal de saída é dada pela referência, o sinal é tão preciso quanto for o cristal. Por exemplo, utilizando um PLL é possível fazer com que o microcontrolador opere a 40 MHz utilizando um cristal de 10MHz. Neste caso o divisor de frequências deve dividir o sinal de saída por 4. Essa funcionalidade pode ser útil, não só para se obter uma frequência de trabalho mais alta como também para evitar problemas com interferência eletromagnética, que geralmente se manifestam de forma mais acentuada quando utilizamos cristais de valores mais altos.

2.7.3 Oscilador RC

Também é possível trabalhar com osciladores RC (Resistor/Capacitor). Para isso é utilizada a montagem abaixo e selecionado o modo de oscilador como RC.

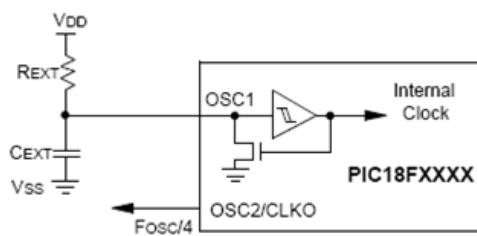


Figura 2.16: Circuito do oscilador no modo RC.

O oscilador RC funciona de forma muito semelhante ao clássico oscilador construído com um CI 555: a carga do capacitor faz a tensão subir até atingir um nível que seja entendido como '1' na entrada do buffer schmitt trigger (VIH); isso faz o transistor conduzir e descarregar o capacitor até um nível de tensão entendido como '0' pelo buffer (VIL), o que faz o transistor cortar e o reinicia o processo de carga do capacitor. Como o tempo de carga é dado em função dos valores do resistor e do capacitor é possível controlar a frequência de oscilação através da escolha destes dois componentes. A figura a seguir mostra as formas de onda na entrada e na saída do buffer.

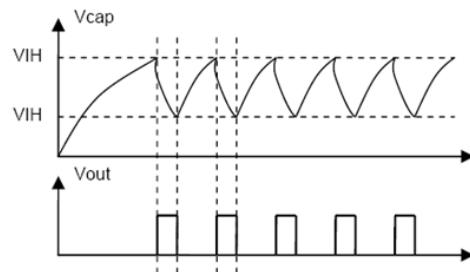


Figura 2.17: Forma de onda do oscilador RC.

O modo RC apresenta algumas limitações: a frequência de trabalho é dependente de vários fatores (valores de resistor e de capacitor, tensão de alimentação, temperatura) e apresenta pouca

estabilidade (tolerância dos resistores e capacitores, variação na tensão de alimentação, variações de temperatura). Por exemplo, em uma produção em quantidade de produtos utilizando PIC's no modo RC, a frequência de cada produto pode variar em função da tolerância dos resistores utilizados. Não é possível garantir precisão em nada que se baseie em tempo no programa. Assim esse modo só é recomendado para aplicações nas quais a frequência não é um fator crítico. Essa situação é que justifica a utilização do modo, pois um oscilador RC tem custo menor que um a cristal.

O PIC18F4550 não possui esse tipo de oscilador.

2.7.4 Oscilador Externo

O modo EC permite que seja aplicado um sinal de clock proveniente de outra parte do circuito. Isso pode ser útil quando já existe algum oscilador ou base de tempo já implementada, ajudando a reduzir custos. Nesses casos o sinal é aplicado ao terminal RA7/OSC1/CLKIN.

A figura ?? apresenta dois circuitos osciladores que podem ser utilizados. Um (a) faz uso de um cristal de ressonância paralela enquanto o outro (b) usa cristal de ressonância série. Na prática, é mais comum encontrar cristais de ressonância paralela.

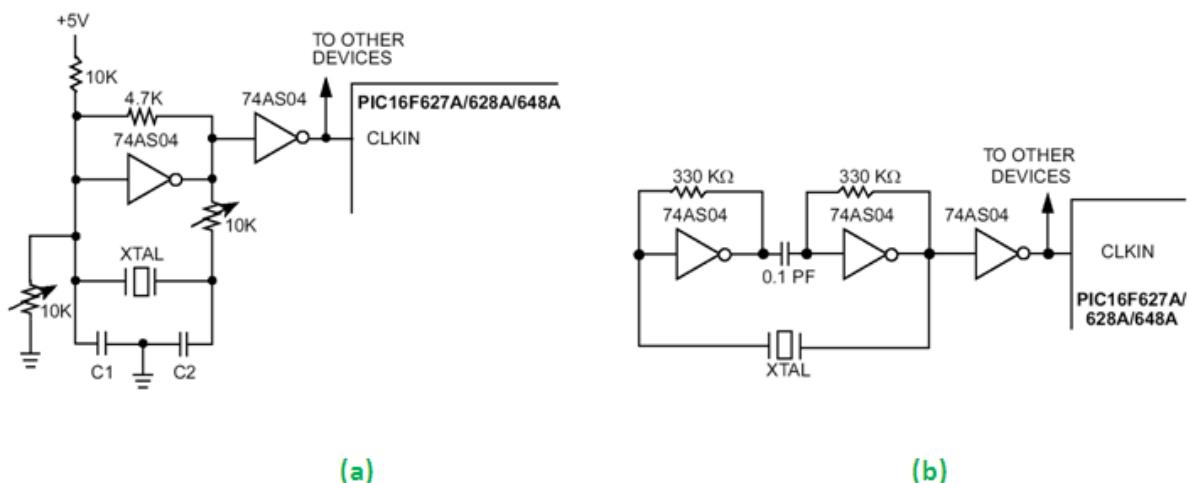


Figura 2.18: Circuitos osciladores: (a) cristal paralelo e (b) cristal série.

2.7.5 Oscilador Interno

O PIC18F4550 possui um oscilador interno (INTOSC) que pode ser usado em substituição a qualquer um dos modos de oscilador interno. O oscilador propriamente dito tem frequência de 8 MHz. Ele possui, porém um pós-escala (divisor de frequência) que permite escolher entre as frequências 8MHz, 4 MHz, 2 MHz, 1MHz, 500 kHz, 250 kHz e 125kHz. Além dessas frequências derivadas do oscilador INTOSC existe outro oscilador chamado de INTRC que gera frequência de 31kHz. Este oscilador gera clock para algumas temporizações do sistema, como o watchdog time, mas também pode ser escolhido como fonte de clock. O processo para determinar qual a frequência usada será apresentado mais a frente.

O oscilador de 8MHz é calibrado em fábrica. Mesmo assim é possível fazer um ajuste fino de sua frequência através do registro OSCTUNE. A precisão típica do oscilador interno é de 1%.

2.7.6 Comparação entre os modos de oscilador

Um modo de oscilador por si só não é melhor que outro. A questão é qual se adapta melhor a necessidade de cada projeto. Os critérios para escolher o tipo de oscilador a ser usado são:

- Precisão: quanto próximo os osciladores estão da frequência nominal em um lote de equipamentos montados.
- Estabilidade: capacidade do oscilador se manter na frequência quando ocorrem mudanças externas (temperatura, tensão de alimentação).
- Variedade de frequências: facilidade de se obter uma frequência desejada qualquer.
- Custo: preço dos componentes externos que compõem o oscilador.

A tabela a seguir apresenta de forma resumida e jovial uma comparação entre os modos de oscilador apresentados (com exceção do modo EC).

Critério	Osciladores a cristal (Com ou sem PLL)	Oscilador RC	Oscilador Interno Internamente
Precisão	Bom	Ruim	Médio
Estabilidade	Bom	Ruim	Médio
Variedade de frequências	Médio	Bom	Médio
Custo	Ruim	Médio	Bom

Tabela 2.5: Comparação entre os modos de oscilador

2.7.7 Sistema de clock do PIC18F4550

Conforme foi dito, o PIC18F4550 usa um sistema de clock mais elaborado que o encontrado em outros componentes da família PIC18. Isso porque ele precisa gerar sinais específicos para o módulo USB. O oscilador primário é feito especificamente com a finalidade de gerar o sinal de clock para a USB. Pode-se optar por usar esse mesmo sinal como clock do sistema ou escolher a fonte secundária ou interna. A figura a seguir representa o sistema de geração e seleção de clock do PIC18F4550.

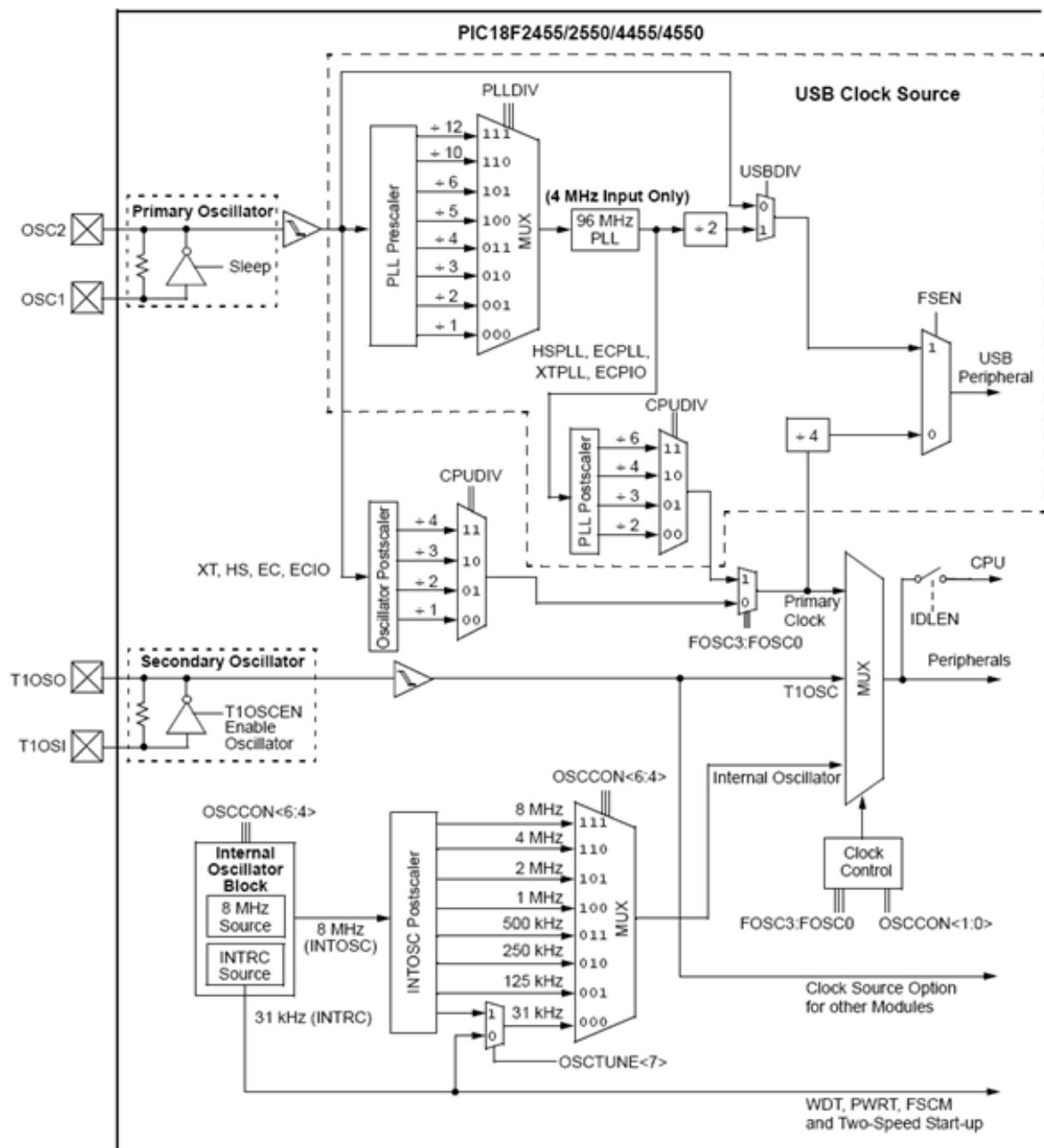


Figura 2.19: Sistema de geração e seleção de clock do PIC18F4550.

Ao escolher o modo de oscilador principal podemos optar por um modo com PLL ou sem PLL. Ao escolher um modo com PLL temos que ter em mente que esse PLL foi concebido para atender as necessidades do módulo USB, portanto o clock gerado para o processador e periférico torna-se uma consequência do clock para USB. Acompanhe na figura acima como cada modo opera.

Ao selecionando um modo sem PLL (XT, HS, EC, ECIO) o sinal do oscilador principal passa por um divisor de frequências, chamado Oscillator Prescaler que permite dividir esse sinal por 2, 3 ou 4, ou não dividir. Podemos assim obter diferentes frequências de operação, mesmo atendendo a necessidade do oscilador para USB. Se não existe interesse em usar a USB, basta escolher esse modo de trabalho e escolher a opção 1:1.

Quando optamos por usar um modo com PLL (HSPLL, XTPLL, ECPLL E ECPIO) o sinal

do oscilador principal é aplicado a um divisor de frequência (PLL Prescaler). O objetivo desse divisor é obter um sinal de 4 MHz em sua saída, permitindo que o usuário escolha cristais de frequências múltiplas de 4MHz. Fazendo-se a seleção correta do fator de divisão obtemos um sinal de 4 MHz aplicado ao PLL, cujo divisor é por 12, ou seja, aplicando uma entrada de 4 MHz obtemos uma saída de 48MHz. Esse sinal é dividido por 2 para se obter a frequência de 48MHz necessária a comunicação USB de alta velocidade. Esse mesmo sinal é aplicado a outro divisor (PLL Postscale) que permite obter uma fração desse sinal para ser usado como sinal de clock pelo processador e periféricos.

Outra fonte de clock disponível é o oscilador secundário. Ele é baseado no oscilador do timer 1, que será tratado em outro lugar. Por hora basta saber que temos mais essa opção de clock.

Por fim, o bloco oscilador interno também está disponível, funcionando conforme discutido acima.

A configuração do oscilador será feita através dos chamados bits de configuração do microcontrolador. Por hora o importante é entender o funcionamento do clock.

2.7.8 Configuração de clock para operar com USB

Para operar corretamente o módulo USB precisa receber sinais de frequências pré-estabelecidas. O módulo USB pode operar no modo baixa velocidade ou alta velocidade.

Em baixa velocidade a USB precisa de um sinal de clock de 6MHz. Para obtê-lo a única opção é usar um cristal de 24MHz com Oscillator Postscale de 1:1. Ao ser selecionado o modo de baixa velocidade o sinal do oscilador primário será dividido por 4 para obter o sinal necessário de 6MHz.

Em alta frequência existe mais flexibilidade. Como o existe um divisor de frequências na entrada do PLL, podemos escolher uma série de valores de cristais, conforme mostrado na tabela abaixo.

Frequência do cristal	Fator de divisão do PLL prescaler
4MHz	1:1
8MHz	1:2
12MHz	1:3
16MHz	1:4
20MHz	1:5
24MHz	1:6
40MHz	1:10
48MHz	1:12

Tabela 2.6: Frequência / fator de divisão do PLL prescaler

Com qualquer um destes cristais é possível obter o sinal de 96MHz na saída do PLL, que será dividido por 2 para se obter 48MHz, necessários a USB. Pode-se também usar o sinal de saída do PLL para escolher uma série de frequências para o clock do processador, através do PLL Postscale, conforme apresentado a seguir:

PLL postscaler	Frequência do oscilador principal
1:2	48MHz
1:3	32MHz
1:4	24MHz
1:6	16MHz

Tabela 2.7: PLL postscaler / Frequência do oscilador principal

2.7.9 Escolha de modo e troca de oscilador

Havendo diversos modos de oscilador e várias opções de configuração é necessário apresentar algumas considerações sobre as combinações possíveis e sobre a mudança de modo de oscilador durante a execução do programa.

Durante a operação normal é possível escolher três fontes para gerar o sinal de clock do sistema: Oscilador principal (ou primário), oscilador secundário e oscilador interno. A mudança da fonte de clock pode ser realizada a qualquer momento durante a execução do programa.

O oscilador principal é o circuito de oscilador externo operando em um dos modos apresentados.

O oscilador secundário é o oscilador do timer 1, que tipicamente opera em 32,768 kHz. Para que o oscilador secundário seja usado é necessário ativar por software o oscilador do Timer 1. O funcionamento desse oscilador é apresentado no tópico específico do Timer1.

Qualquer uma das frequências do oscilador interno pode ser escolhida durante a execução do programa.

Para evitar problemas na troca de oscilador, a execução do programa pára durante a um período de transição dos osciladores. Esse período é dado pela soma do tempo correspondente a dois períodos da fonte antiga mais três ou quatro períodos da nova fonte de clock. Durante esse período o clock para os periféricos também fica suspenso.

Os registros de controle do oscilador são OSCTUNE em OSCCON. A descrição dos seus bits é apresentada a seguir.

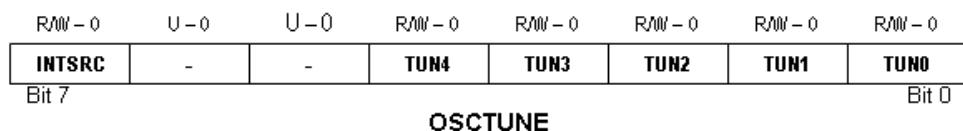


Figura 2.20: Controle do oscilador (OSCTUNE)

- INTRC : Seleção de clock do oscilador interno para 31kHz
 - 1 = 31,25 kHz da pós-escala do oscilador INTOSC
 - 0 = 31 kHz diretamente do oscilador INTRC
- TUN4:TUN0: Ajuste fino de frequência do oscilador INTOSC:
 - O valor 00000 corresponde a calibração de fábrica. Valores entre 00001 e 01111 geram frequências mais altas que a calibrada. Valores entre 11111 e 10000 geram frequências mais baixas que a calibrada.

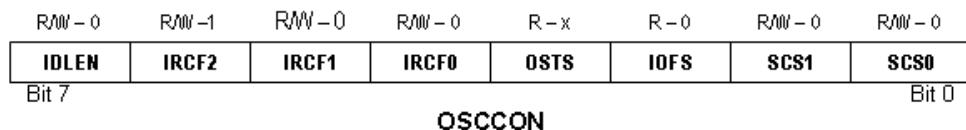


Figura 2.21: Controle do oscilador (OSCCON)

- IDLEN : Habilitação do modo IDLE
 - 1 = Entre em modo IDLE quando a instrução SLEEP for executada
 - 0 = Entre em modo SLEEP quando a instrução SLEEP for executada
- IRCF2:IRC0F : Frequência do oscilador interno.
 - 111 = 8 MHz (sem pos-escala)
 - 110 = 4 MHz
 - 101 = 2 MHz
 - 100 = 1 MHz
 - 011 = 500 kHz
 - 010 = 250 kHz
 - 001 = 125MHz
 - 000 = 31kHz, conforme configurado em INTRC de OSCTUNE
- OSTS: Estado do oscilador principal
 - 1 = Oscilador principal estável, funcionando como fonte de clock
 - 0 = Oscilador principal não estável, ainda não está pronto para fornecer clock
- IOFS: Estado do oscilador interno
 - 1 = Oscilador interno estável
 - 0 = Oscilador interno não estável
- SCS1:SCS0 : Seleção de clock
 - 1x = Oscilador interno
 - 01 = Oscilador secundário (Timer 1)
 - 00 = Oscilador primário.

2.8 Características Especiais

O PIC possui uma série de funcionalidades ligadas à CPU, chamadas de características especiais (Special Features). As principais dentre elas são tratadas abaixo. A maioria dessas características é configurável através dos Bits de Configuração (Configuration Bits) no processo de gravação do microcontrolador através de endereços especiais. A definição dessas configurações também pode ser feita no código em C através da diretiva `#pragma config`, como será tratado quando forem estudadas as diretivas em linguagem C.

2.8.1 Modos de baixo consumo

A arquitetura do PIC18 prevê três tipos de modos de operação: normal (Run), Idle e Sleep. Combinados com as possibilidades de escolha de fonte de clock é possível utilizar 7 modos diferentes de consumo.

No modo normal, a CPU do microcontrolador e os periféricos recebem sinal da fonte de clock selecionada e o programa é executado.

No modo Sleep, os osciladores do microcontrolador são desligados. Nem CPU nem os periféricos recebem sinal de clock (com exceção do TIMER1, caso tenha seu próprio oscilador ativo). Nesse modo o consumo de energia é mínimo, porém deve-se considerar os atrasos antes do retorno a execução normal causados pelo tempo de partida dos osciladores.

No modo Idle os osciladores permanecem em operação e a penas o sinal de clock da CPU é cortado; os periféricos continuam recebendo o sinal da fonte de clock selecionada normalmente. Nesse caso temos uma redução de consumo menor que a proporcionada pelo modo Sleep, porém o processador volta a operar imediatamente ao retornar ao modo normal, já que os osciladores continuam operando (não a tempo de partida). Outro benefício do modo Idle é que os periféricos continuam recebendo sinal de clock, podendo causar o retorno ao modo normal por interrupção.

Qualquer uma das 3 fontes de clock pode ser escolhida durante nos modos normal e Idle. É importante lembrar que quanto menor a frequência de operação, menor o consumo de energia.

Para entrar tanto em modo Idle como em modo Sleep basta executar a instrução SLEEP. O bit INDLEN do registro OSCON controla qual o modo de baixo consumo selecionado. Para um menor consumo de energia ações que reduzam o consumo do circuito coerentes com o hardware devem ser implementadas (como desligar LEDs e cortar a alimentação de algumas partes do circuito.)

Uma vez em modo Sleep ou Idle, três tipos de eventos podem "acordar" o microcontrolador, conforme apresentado a seguir. "Acordar" o microcontrolador significa reativar o oscilador principal e continuar a execução do programa.

- Nível lógico baixo no pino MCLR, o que faz com que o microcontrolador "acorde" e reset
- Estouro do WatchDog, o que não ocasiona RESET, apenas acorda o microcontrolador. Nesse caso o microcontrolador não é resetado.
- A ocorrência de alguma interrupção habilitada.

É importante observar que diversos periféricos podem estar utilizando o sinal do oscilador principal e, portanto pararão de operar durante o modo Sleep. Interrupções associadas a esses periféricos não irão ocorrer, portanto não podem acordar o microcontrolador.

Para uma interrupção causar a saída de um modo de baixo consumo ela precisa apenas estar habilitada, não sendo necessário a habilitação global também. Se a habilitação global não estiver habilitada, a ocorrência da interrupção acorda o microcontrolador e o programa segue a partir da instrução seguinte a instrução SLEEP. Já se estiver habilitada o microcontrolador acorda e o programa desvia imediatamente para o vetor de interrupção apropriado.

A tabela abaixo apresenta um resumo das combinações possíveis de modos de baixo consumo e fontes de clock.

Modo	OSCCON		Sinal de clock		Fonte de clock
	IDLEN	SCS1:SCS0	CPU	Periféricos	
PRI_RUN	X	00	Sim	Sim	Principal
SEC_RUN	X	01	Sim	Sim	Secundário
RC_RUN	X	1X	Sim	Sim	Oscilador interno
PRI_IDLE	1	00	Não	Sim	Principal
SEC_IDLE	1	01	Não	Sim	Secundário
RCI_DLE	1	1X	Não	Sim	Oscilador interno
Sleep	0	X	Não	Sim	Todas desligadas

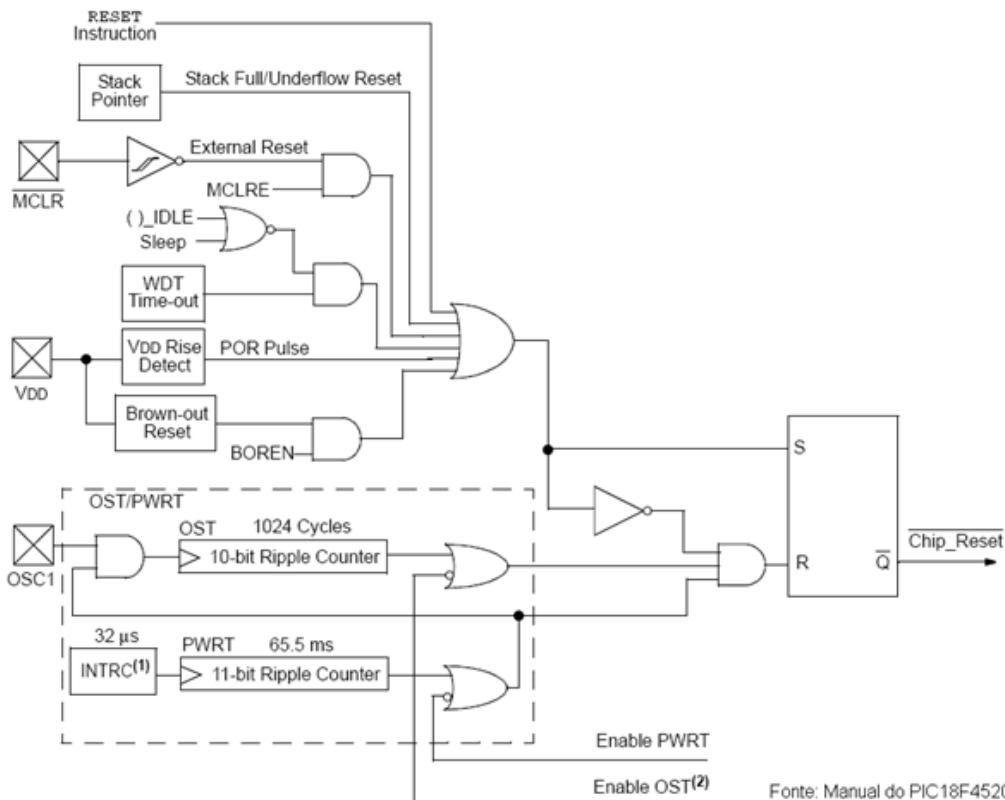
Tabela 2.8: Combinações possíveis de modos de baixo consumo e fontes de clock.

2.8.2 Reset

Os PIC18 possuem diversas fontes de reset e funcionalidades associadas que visam principalmente manter a confiabilidade e segurança das aplicações. As fontes de reset possíveis são:

- POR (Power-On Reset), causado pela deteção da alimentação quando o circuito é ligado;
- Nível lógico baixo no pino MCLR durante a execução normal;
- Nível lógico baixo no pino MCLR com o microcontrolador em modo SLEEP;
- Estouro do WatchDog timer durante a operação normal;
- BOR (Brown-out Reset) reset por queda de tensão;
- Instrução RESET;
- Estouro da pilha (overflow e underflow);

A figura a seguir apresenta um diagrama simplificado do sistema de reset do PIC18F4550.



Fonte: Manual do PIC18F4520

Figura 2.22: Diagrama de reset do PIC18F4550.

A forma mais simples de reset é através do pino MCLR: o microcontrolador reseta quanto um nível lógico é aplicado a esse pino. No PIC18F4550 pode-se desabilitar a função de reset do pino MCLR. Nesse caso ele passa a operar como um pino de entrada do portal E. Isso é feito via bit de configuração.

2.8.3 POR - Power-On Reset

Essa funcionalidade detecta quando o circuito é ligado, percebendo a subida da tensão de alimentação, e realiza o reset do microcontrolador. Dessa forma podemos dispensar os resistor e capacitor tradicionalmente ligados ao terminal de reset para gerar um reset quando o sistema é ligado. O terminal MCLR pode ser ligado diretamente ao nível lógico alto sendo recomendado utilizar um resistor de 10 k ligando-o à VDD. Dessa forma também o pino MCLR pode estar disponível para gravação in-circuit ou para ser usado como entrada se configurado com I/O.

Associado ao POR existem duas funcionalidades interessantes: PWRT e OST.

2.8.4 PWRT - Power-up Timer

É útil, pois permite ao sistema ter tempo de se estabilizar (tensão, reset dos demais componentes, etc.) antes do programa começar a ser executado.

O PWRT (Power-up Timer) é um timer que mantém o microcontrolador em estado de reset por aproximadamente 65,5ms no PIC18F4550 após POR, permitindo que a alimentação do circuito se estabilize. Essa funcionalidade pode ser habilitada ou não.

2.8.5 OST - Oscillator Start-up Timer

Já o OST (Oscillator Start-up Timer) mantém o microcontrolador em estado de reset durante os primeiros 1024 períodos do oscilador, de forma que o programa seja executado somente quando o oscilador já estiver perfeitamente estabilizado. Essa funcionalidade é sempre ativa nos modos de oscilador a cristal. A figura 2.3 apresenta o comportamento de partida do oscilador.

2.8.6 BOR - Brown-out Reset

Essa funcionalidade visa resetar o sistema se houver uma queda na tensão de alimentação. Quando a tensão cai abaixo dos limites tolerados para o circuito os resultados são imprevisíveis. Há situações nas quais é interessante que quando ocorra uma queda dessas o sistema seja resetado.

Nos PIC18 a tensão de limiar para o BOR é configurável. Para o PIC18F4550 essa tensão pode ser 2,05V, 2,79V, 4,33V ou 4,59V. Para outros componentes, consulte o manual específico. O BOR pode ser habilitado ou desabilitado. Uma condição de queda de tensão só é caracterizada se a tensão cair abaixo do limiar escolhido por mais de 100 μ s.

No PIC18F4550 existe ainda a possibilidade de ter o BOR controlador por software, através do bit SBOREN do registro RCON; nesse caso a faixa de tensão é a escolhida na configuração. Quando o BOR é usado recomenda-se que também o PWRT esteja ativo.

Na figura a seguir é apresentada a forma de operação do BOR em conjunto com o PWRT.

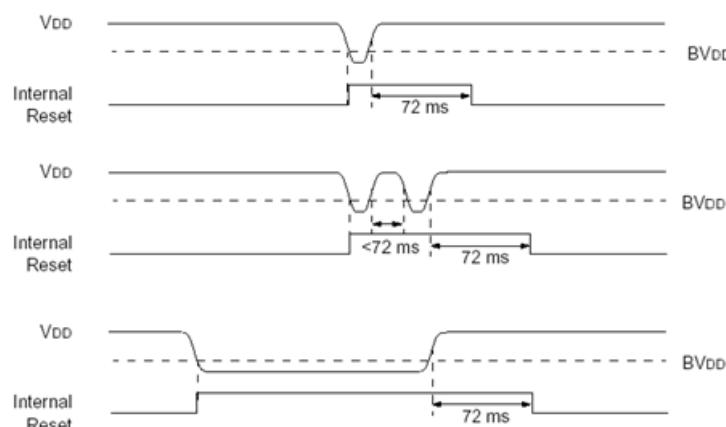


Figura 2.23: Situações de Brown-out

2.8.7 Causas de reset

As causas de reset podem ser identificadas através da de bits do registros RCON e STKPTR. O registro RCON e seus bits são apresentados abaixo.

R/W - 0	R/W - 1	U - 0	R/W - 1	R - 1	R - 1	R/W - 0	R/W - 0
IPEN	SBOREN	-	RI	TO	PD	POR	BOR
Bit 7							Bit 0

RCON

Figura 2.24: RCON e seus bits.

- IPEN : habilitação de prioridade de interrupção:
 - 1 = Prioridade Habilitada;
 - 0 = Prioridade desabilitada (compatível com o PIC16);
- SBOREN : Controle de BOR por software (depende dos bits de configuração). Somente no PIC18F4550.
 - 1 = BOR habilitado;
 - 0 = BOR desabilitado;
- RI : Indicador de execução de instrução RESET (deve ser setado pelo software após a ocorrência de um BOR)
 - 1 = a instrução RESET não foi executada;
 - 0 = a instrução RESET foi executada;
- TO: Indicador de estouro do WacthDog;
 - 1 = setado por POR, e pelas instruções CLRWDT e SLEEP;
 - 0 = zerado quando ocorre um estouro do WatchDog;
- PD: Indicador de modo de baixo consumo (SLEEP)
 - 1 = setado por POR e pela instrução CLRWDT;
 - 0 = zerado pela instrução SLEEP;
- POR: Indicador de POR (deve ser setado por software após ocorrência de POR).
 - 1 = não ocorreu POR;
 - 0 = ocorreu POR;
- BOR: Indicador de BOR (deve ser setado por software após ocorrência de BOR).
 - 1 = não ocorreu BOR;
 - 0 = ocorreu BOR;

2.8.8 Sequência de inicialização

Como exposto, existem vários tempos que podem ser aguardados antes que o microcontrolador saia da condição de reset, isto é, comece a executar o programa. Supondo todas as funcionalidades acima ativas a sequencia é a seguinte:

1. o circuito é ligado: a tensão de alimentação começa a subir;
2. é detectada a subida da tensão: POR;
3. o sistema aguarda o tempo de PWRT (aprox. 65 ms);
4. o oscilador começa a funcionar: 1024 ciclos são contados (OST);
5. o programa começa a rodar.

2.8.9 Watch-Dog Timer

Esse timer de 8 bits, que é baseado em um oscilador INTRC, independente do oscilador principal, gera um reset quando "estoura". Ele é importante em situações em que, por qualquer motivo, o microcontrolador "trava". Quando habilitado ele deve ser zerado a intervalos regulares menores que seu tempo máximo pela instrução CLRWDT. Se o programa "para" e o Watch-Dog não é zerado, tendo sido habilitada essa função na gravação, ocorre o reset. O WatchDog do PIC18 pode ser habilitado ou não na gravação do microcontrolador.

Da mesma maneira é possível definir através dos bits de configuração qual o valor da pós-escala do WatchDog. A pós-escala nada mais é que um contador de 16 bits que conta o número de estouros do Watchdog, uma forma de aumentar o tempo até o reset. Tipicamente o WatchDog estoura a cada 4 ms no PIC18F4550. Se for adotada uma pós-escala de 1:4 significa que um reset ocorrerá a cada 4 estouros do WatchDog, ou seja, a cada $4 \times 4\text{ms} = 16\text{ms}$, se a instrução CLRWDT não for usada. A execução da instrução CLRWDT não só zera o WatchDog como também a pós-escala, reiniciando a contagem.

Ao contrário do que ocorre com o PIC16, no PIC18 não existe qualquer relação entre o WatchDog e o Timer0.

Caso o WatchDog não seja habilitado através de bits de configuração mesmo assim é possível ativá-lo escrevendo '1' do bit SWDTEN do registro WDTCON. Também é possível desabilitar o WatchDog somente escrevendo '0' no bit SWDTEN (desde que o WatchDog não tenha sido habilitado nos bits de configuração).

Se o microcontrolador for colado em modo Sleep ou Idle o WatchDog continua a incrementar, uma vez que sua base de tempo é o oscilador INTRC. Contudo, seu estouro (considerando a pós-escala) não irá resetar o microcontrolador, mas sim tirá-lo do modo Sleep ou Idle. A execução da instrução SLEEP zera o WatchDog e a pós-escala assim como a instrução CLRWDT.

2.8.10 Proteção de Código (Code Protect)

O programa gravado em um PIC pode ser protegido, isto é, pode ser impedida a sua leitura. Essa funcionalidade é muito importante, sobretudo quando se trata de produção industrial, visto que assim se podem preservar os direitos autorais do autor do firmware e dificultar a cópia de produtos.

Quando um componente está protegido e se realiza sua leitura, o valor lido em qualquer endereço é 0000h. Por isso um componente protegido não pode ser confundido com um componente apagado, pois nesse último caso o valor lido em qualquer endereço é 3FFFh.

2.8.11 Gravação e depuração

O PIC é gravado através de um processo chamado ICSP (in-circuit serial programming) que realiza a gravação usando apenas 2 terminais de I/O e o pino de reset. Dessa maneira é possível gravar o microcontrolador mesmo este estando já montado na placa de aplicação. Essa funcionalidade é de grande valia quando se usa componentes SMD ou quando o firmware precisa ser atualizado em campo.

Os pinos usados são chamados de PGD (dados) PGC (clock) e VPP (alta tensão) que no caso do 18F4550 correspondem ao pinos RB7, RB6 e MCLR. Como para gravar é necessário também

que o microcontrolador esteja alimentado, a interface ICSP normalmente tem ainda os pinos de GND e VDD.

Além da gravação a interface ICSP permite também a depuração, que é o processo de executar o programa de forma sincronizada com o MPLAB IDE. Isto permite colocar breakpoints (pontos de paradas em posições do código), ler e alterar o conteúdo da memória, executar o código passo-a-passo. Tudo isso com o programa gravado e sendo executado no microcontrolador. Mais detalhes sobre a depuração são apresentados na sessão que trata especificamente do MPLAB.

Capítulo 3

Ferramentas para o PIC18F: MPLAB IDE e Compilador C18

Como ferramentas para o desenvolvimento de aplicações para o PIC18F serão abordados aqui o ambiente de desenvolvimento MPLAB e o compilador C18 (versão estudantil), ambos fornecidos gratuitamente pela Microchip. Esses softwares se encontram no CD que acompanha a apostila e atualizações podem ser baixadas em www.microchip.com.

O MPLAB IDE apresentado é a versão 8.2 enquanto o C18 é a versão estudantil 3.20. São essas as versões mais recentes disponíveis no momento em que esse texto é escrito. Não há qualquer limitação quanto ao uso do MPLAB IDE. Já o C18 apresenta todos os recursos por um período de 60 dias, perdendo a otimização de código e deixando de usar o conjunto de instruções estendido após esse período; isso implica na geração de código menos eficiente em termos de utilização de memória, mas nenhuma funcionalidade é perdida.

Serão feitos também alguns comentários sobre outros compilador C disponíveis para microcontroladores PIC.

3.1 O MPLAB IDE

Para desenvolver projetos para microcontroladores PIC a Microchip disponibiliza uma IDE (Integrated Development Environment - Ambiente de Desenvolvimento Integrado) bastante versátil e simples de usar chamada MPLAB IDE. As principais características do MPLAB são a simplicidade e a facilidade de utilização.

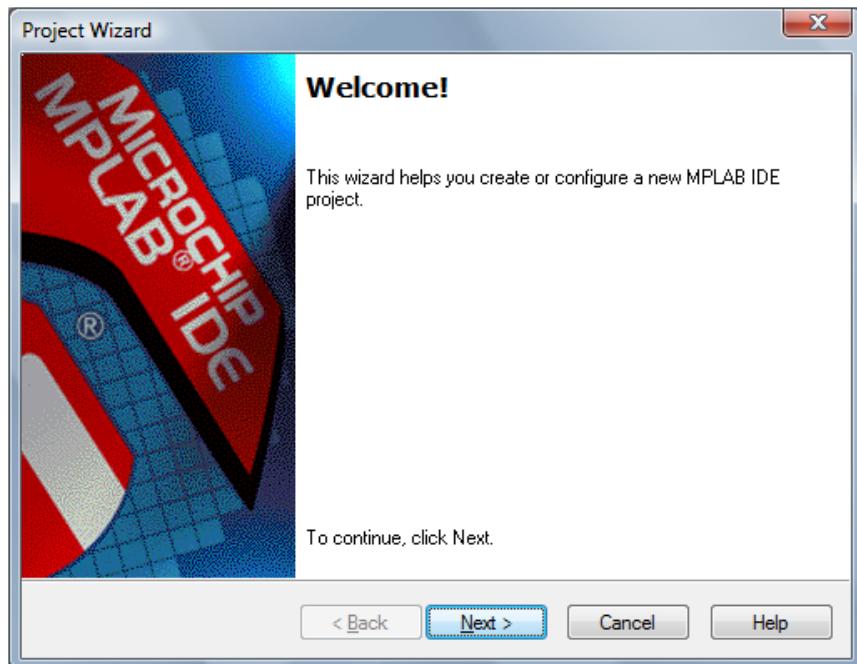
Uma IDE, como próprio nome diz, é um ambiente de desenvolvimento que integra diversos programas em uma interface amigável. Associado ao MPLAB existem diversos programas que são executados a medida que o processo de desenvolvimento ocorre. Esse programas são compiladores, linkers, simuladores, etc. Mais adiante trataremos especificamente do compilador C18 e do simulador MPLAB SIM.

A instalação do MPLAB não apresenta nenhum detalhe especial e pode ser realizada sem maiores detalhes. Basta inserir o CD do curso no drive, clicar no link para instalação e seguir os passos indicados.

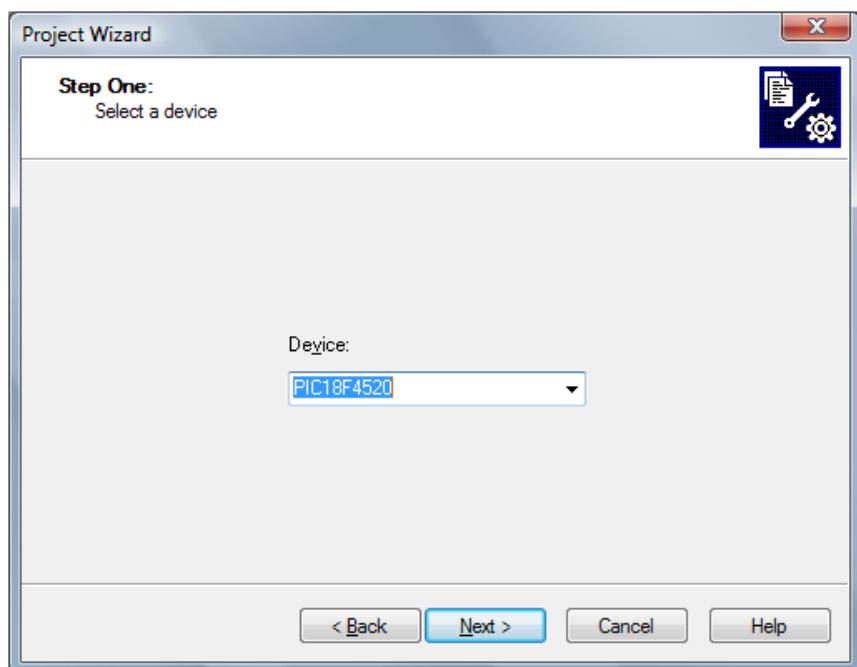
Uma vez instalado o MPLAB IDE , para abri-lo execute **Iniciar ⇒ Programas ⇒ Microchip ⇒ MPLAB IDE v7.31 ⇒ MPLAB IDE**

3.1.1 Criando um novo projeto

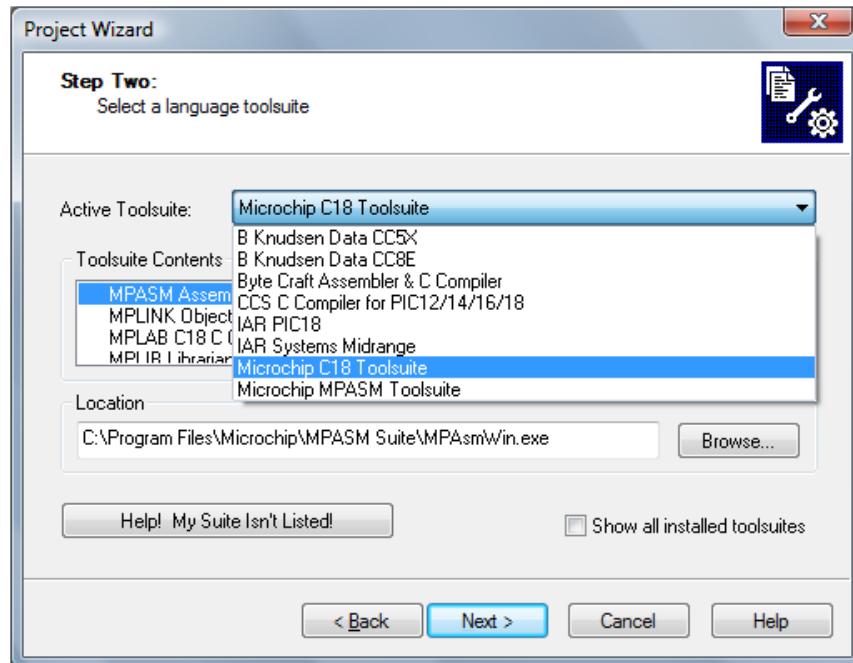
O MPLAB trabalha com o conceito de Projeto, isto é, cada novo aplicação desenvolvida (projeto) é composto por diversos arquivos gerenciados pelo MPLAB. Para criar um novo projeto existe um assistente, que é executado em **Project ⇒ Project Wizard...**



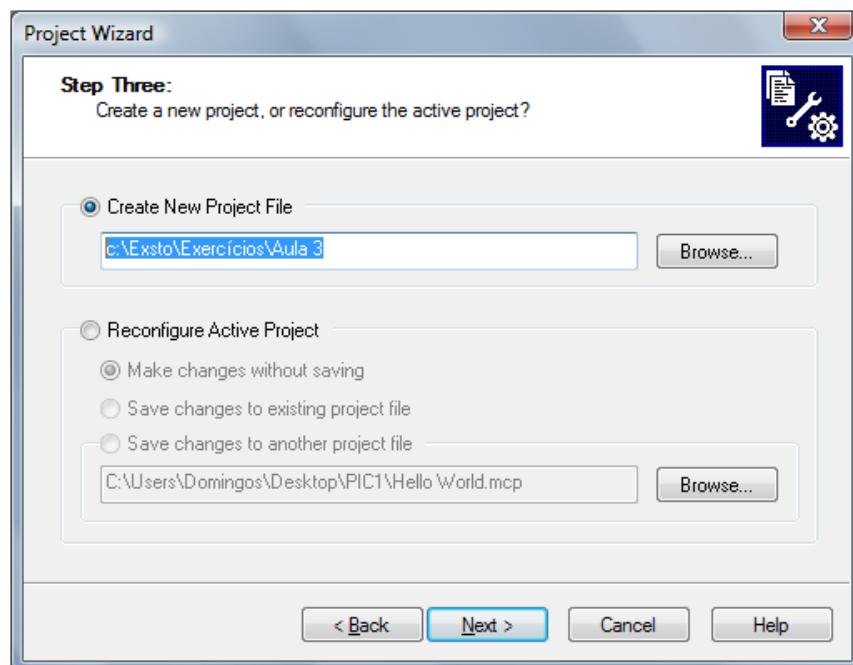
Aparecerá a janela acima Clique em Avançar.



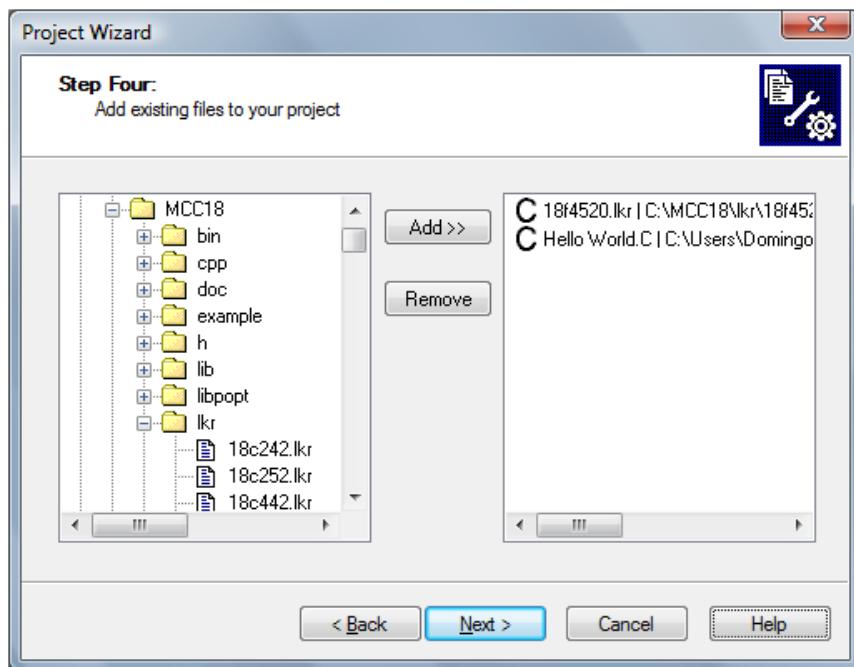
Na janela seguinte é feita a escolha do microcontrolador usado no projeto. Selecione o PIC18F4550 na lista ou simplesmente digite o nome do microcontrolador e clique em Avançar.



Nessa janela é configurada a linguagem de programação a ser utilizada no projeto, através das opções da lista **"Active Toolsuite"**. Para desenvolver um projeto em Assembly escolha a opção **"Microchip MPASM Toolsuite"**. Para desenvolver em C utilizando o compilador C18 escolhar **"Microchip C18 Toolsuite"**. As demais ferramentas da lista podem não estar instaladas. Escolhida a linguagem, clique em Avançar.



Neste passo pode-se optar por criar um novo projeto ou reconfigurar um projeto já existente. Como estamos criando um novo projeto escolheremos esta opção e indicaremos a pasta onde o projeto deve ser criado. Para melhorar a organização, cada projeto é alojado em uma pasta. No campo abaixo de **Create a New Project** preencha o caminho e nome do projeto ou a pasta onde o projeto será guardado, utilizando a opção **Browse...** para explorar o computador. Clique em **Avançar**. Caso o diretório escolhido ainda não exista aparecerá uma mensagem perguntando se ele deve ser criado. Neste caso, clique em **OK**.



A próxima janela permite adicionar um arquivo já existente ao projeto. Clicando em Add o arquivo será adicionado a lista dos arquivos pertencentes ao projeto. O mesmo processo pode ser utilizado para incluir bibliotecas de funções.

É importante incluir também um arquivo com orientações para o linker (linker script). Para isso encontre a pasta "lkr" dentro da pasta de trabalho do C18 (se nenhuma alteração foi feita na instalação essa pasta deve ser C:\MCC18\lkr) e adicione o arquivo referente ao microcontrolador usado, no caso 18F4550.lkr.

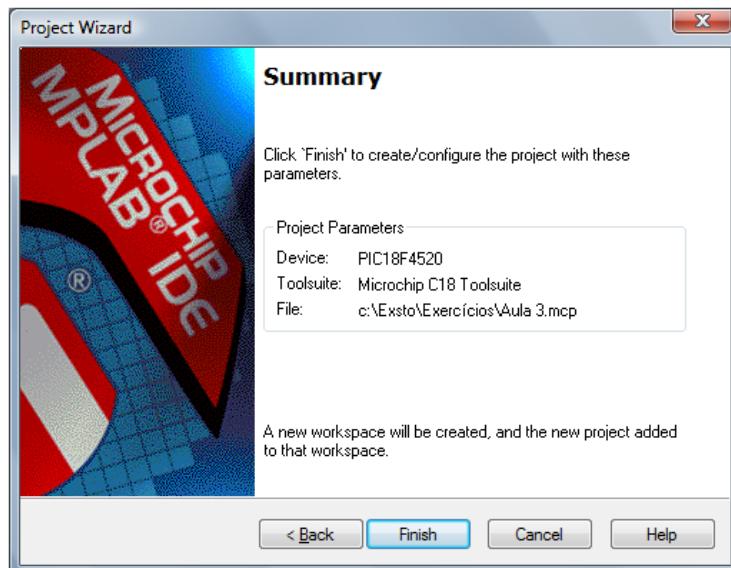
A letra a lado direito do arquivo adicionado indica como o processador irá acessar esse arquivo. Isso impacta principalmente quando o projeto é copiado para outro computador, pois define se o caminho para acessar o arquivo é absoluto (exatamente a mesma pasta em que arquivo se encontra) ou relativo (um caminho relativo a pasta de projeto, geralmente usado quando o arquivo em questão encontra-se na pasta do projeto em alguma subpasta sua). As opções são:

- **A (Auto)**: O MPLAB define o tipo de caminho automaticamente. Se o arquivo estiver na pasta de projeto ele define o caminho como relativo, caso contrário define como absoluto
- **U (Usuário)**: define o caminho como relativo. Recomendado para arquivos criados especificamente para o projeto.
- **S (Sistema)**: define o caminho como absoluto. Normalmente usado para arquivos usados em vários projetos com um local predefinido.

- **C (Copiar):** Copia o arquivo para a pasta de projeto.

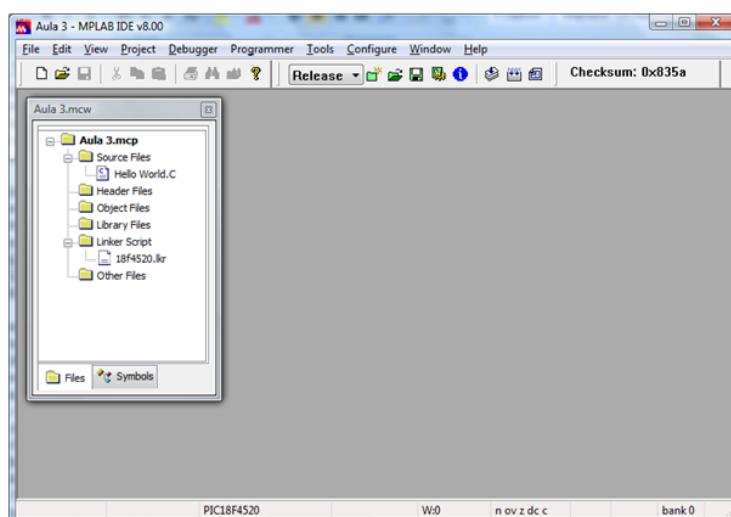
Recomendamos usar sempre a opção C (Copiar o arquivo para pasta de projeto) para evitar que ao levar o projeto para outra maquina o MPLAB não consiga encontrar o caminho do projeto.

Marque os arquivos selecionados com a opção C e em seguida clique em **Avançar**.



A última tela apresenta um resumo do projeto criado. Confira se todas as informações estão corretas e clique em **Concluir**.

Na tela do MPALB aparecerá a janela de gerência do projeto, que contem todos os arquivos pertencentes ao projeto. Para editar um arquivo dê um clique duplo sobre ele.



Para criar um novo arquivo, vá em **File ⇒ New**.

Para criar um novo arquivo e adicioná-lo ao projeto vá em **File ⇒ Add new file to project...**

Para salvar um novo arquivo criado, vá em **File ⇒ Save As...**

Atenção: Não basta apenas criar um arquivo para que ele faça parte de um projeto. É preciso adicioná-lo a lista de arquivos do projeto. Para isso, na janela de gerência de projetos. Clique com o botão direito sobre Source Files (ou sobre Header Files, caso se trate de uma biblioteca, como veremos mais adiante) e selecione a opção Add Files para selecionar e incluir um arquivo.

Por fim, para abrir um projeto já existente, vá em Project ⇒ Open... e escolha um arquivo de projeto (que tem terminação .MCP).

3.1.2 Construindo um projeto.

Uma vez criado o projeto, o mesmo deve ser "construído" para que possa ser gravado no microcontrolador ou simulado.

Cabe aqui dizer que os arquivos de código fonte Assembly possuem a terminação ".ASM" e os arquivos de biblioteca para Assembly têm terminação ".INC". Já os arquivos de código fonte de C têm terminação ".C"; os arquivos de cabeçalho e bibliotecas em C tem a extensão ".H". Além desses, há também o roteiro de ligação (linker script) que traz informações sobre como o programa será alojado na memória do microcontrolador e tem extensão ".LKR"; O montador (MPASM Assembler) ou o compilador (C18) interpretam os arquivos criados pelo usuário ou cria diversos outros. Os de maior importância são os arquivos de erro (terminação .ERR) que permitem visualizar mensagens de erro e os arquivos binários (terminação .HEX) que são os arquivos gravados no microcontrolador.

Para construir um projeto use a tecla de atalho **Ctrl+F10** ou vá em Project ⇒ Build All.

Feito o isso aparecerá a tela **Output** apresentando o resultados de compilação, bem como mensagens, avisos ou erros encontrados. Se ocorrerem erros no processo de compilação os mesmos são apresentados em lista e um clique duplo sobre eles mostra no código fonte a linha onde se encontra o erro. Não havendo erros a última linha da janela Output apresentará a mensagem **BUILD SUCCEEDED** seguida da data.

3.1.3 Gravando o microcontrolador

Para gravar o microcontrolador usa-se o ICD-2 uma ferramenta que permite, além da gravação, a depuração, como veremos mais a frente. Uma vez que ele estiver devidamente conectado ao kit e ao PC, vá em **Programmer** ⇒ **Select Programmer** ⇒ **2 MPLAB ICD 2**. Surgirá uma barra de ferramentas cuja função é apresentada na figura abaixo.



Figura 3.1: Barra de ferramentas ICD2

3.2 Simulação em C

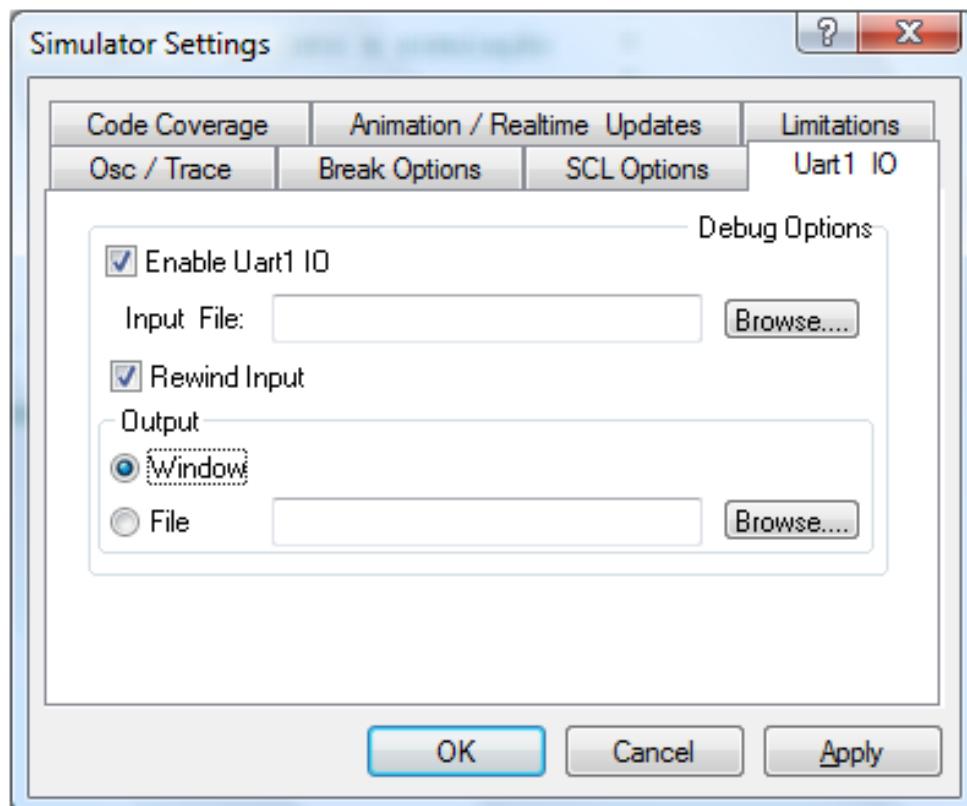
Uma poderosa ferramenta de desenvolvimento é o simulador integrado ao MPLAB IDE. Esse simulador permite a simulação de códigos escritos tanto em C (seja C18 ou outros compiladores suportados) como em Assembly.

O uso do simulador é particularmente interessante durante a etapa de estudo dos microcontrolador e para a verificação de do funcionamento de funções e trechos de código isolados. Já para

programas inteiros a simulação pode não apresentar resultados muitos conclusivos. Nesses casos torna-se mais interessante o uso de uma ferramenta de depuração, conforme tratado a seguir.

A seguir são apresentados os passos básicos para a realização de simulação no MPLAB.

Uma vez criado um projeto, o primeiro passo é habilitar o simulador como ferramenta de depuração. Para isso escolha Debugger 'Select Tool' MPLAB SIM. Em seguida configure a simulação através de Debugger 'Settings...'. Dentre as opções existentes, recomendamos configurar a porta serial virtual conforme apresentado na figura abaixo. Isso permite que as informações enviadas pela porta serial durante a simulação sejam exibidas em um janela do MPLAB.



Através do menu View é possível abrir diversas janelas de visualização da simulação:

- Disassembly Listing: listagem do código. Essa tela eh muito interessante pois permite verificar o código assembly gerado para cada linha escrita em C.
- Hardware Stack: Conteúdo da pilha.
- Program Memory: Conteúdo da memória de programa do microcontrolador.
- File Registers: Conteúdo de todos os endereços da memória de dados.
- EEPROM: Conteúdo da memória de dado EEPROM.
- Special Function Register: conteúdo dos registros de função especial.
- Watch: permite a escolha dos endereços de memória RAM (SFR ou de uso geral) e de bits para visualização. Permite:
 - Adicionar SFR:

- * Selecione o SFR na lista da esquerda e clique Add SFR
 - Adicionar Símbolo (variáveis criadas)
 - * Selecione o símbolo na lista da direita e clique Add Symbol
- Memory Usage Gauge: permite visualizar o consumo de memória de dados e de programa.
- Simulator Logic Analyzer: um analisador lógico para simulação (será visto em detalhes mais a frente).

3.2.1 Simulação

- Acompanhar o andamento do programa pela janela no programa ou pela janela Disassembly Listing.

Comandos (os comando podem ser dados através do Menu Debugger, de teclas de atalho e da barra de ferramentas):

Comando	Atalho	Ícone	Descrição
Run	F9		Executa o programa em tempo real
Halt	F5		Para a execução do programa onde estiver
Animate			Executa continuamente Step Into
Step into	F7		Executa passo a passo; entra em sub rotinas.
Step Over	F8		Execução passo-a-passos; salta em sub-rotinas
Step out			Sai da sub rotina atual
Reset	F6		Reseta o programa

Tabela 3.1: Comandos e suas descrições

3.2.2 Outras funcionalidades

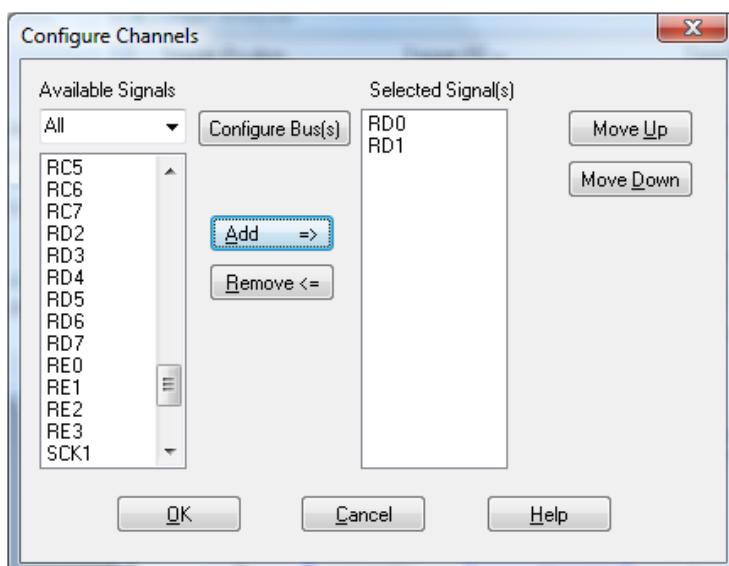
- Breakpoints: param a execução do programa quando a instrução marcada é atingida
 - Clique o botão direito do mouse na área cinza ao lado da instrução;
 - Selecione "Set Breakpoint".
 - Para retirar, clique com o botão direito do mouse sobre o breakpoint e selecione "Remove Breakpoint".
- Desviar o programa para uma determinada instrução.
 - Clique na instrução desejada com o botão direito do mouse.
 - Selecione "Set PC to Cursor"
- Alterar o valor de um registro(ou variável):
 - Na janela Watch:
 - * Clique duplo no campo "Value" e entre com o novo valor.
 - Na janela File Registers:

- * Clique duplo no endereço do registro e entre com o novo valor
- Compilação: antes de simular um programa, ele deve ser compilador (montado)
 - Clique na janela do programa.
 - Menu Project ⇒ Quickbuild <nome_do_arquivo> ou Alt+F10
- Formato do dado mostrado na janela Watch
 - Selecione a linha do registro
 - Clique com o botão direito do mouse e selecione "Properties..."
 - Selecione o formato desejado no campo "Format"
 - * Quando for selecionado "Decimal" aparece o campo "Signed" que se marcado apresenta o valor como um número sinalizado.

3.2.3 Analisador lógico

O analisador lógico do MPLAB SIM se comporta como um analisador lógico real, isto é, ele plota o estado de sinais (pinos do microcontrolador) em função do tempo. Isto permite monitorar o estado dos pinos e de alguns sinais internos como se tivéssemos um osciloscópio conectado a esses pinos. Para ativar o analisador lógico de simulação va em View ' Simulator Logic Analyzer

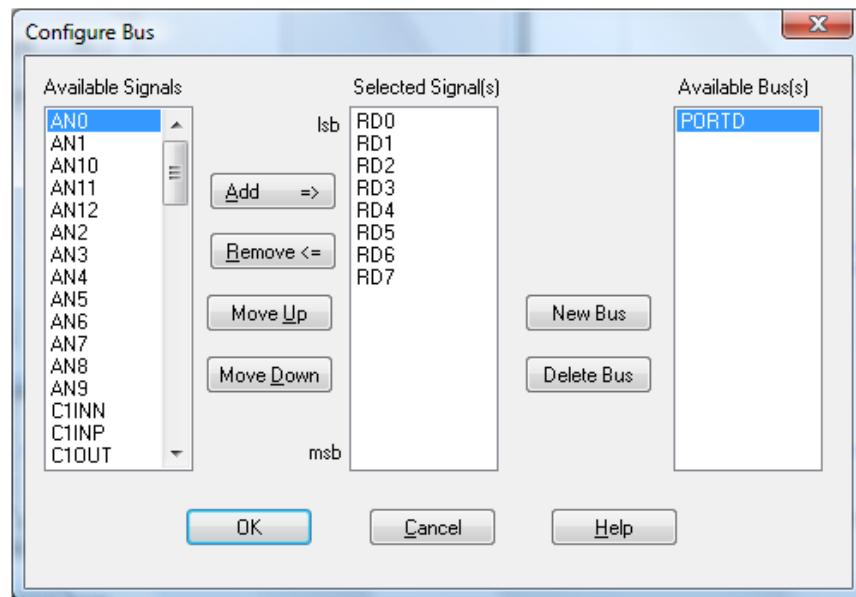
A primeira coisa a fazer para trabalhar com o analisador é selecionar os canais (sinais) que devem ser monitorados. Para isso clique no botão Channels. Surgirá a janela abaixo.



Basta escolher os sinais a serem mostrados na lista a esquerda e clicar em Add para que esses sejam mostrados no analisador.

Uma funcionalidade interessante para monitorar o funcionamento de diversos sinais conjuntamente e agrupá-los em um barramento (Bus). Isso permite que o conjunto de sinais seja apresentado como um único valor hexadecimal. Para isso clique em Configure Bus(s). Será mostrada a tela abaixo.

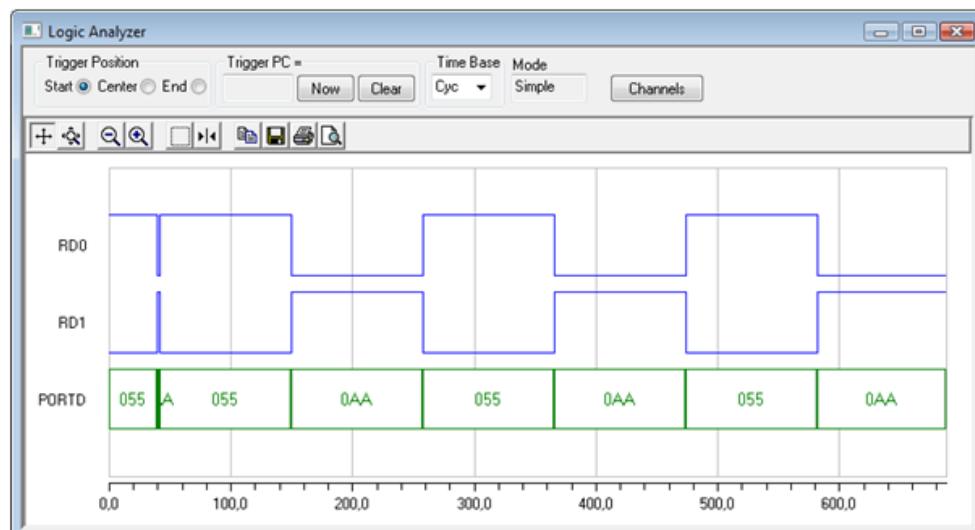
Para criar um barramento clique em New Bus, preencha o nome do barramento e clique em ok na janela de dialogo. Ao fazer isso o barramento criado aparecerá na lista Available Bus(s). Em



seguida selecione o barramento na lista e defina os sinais que comporão esse barramento na lista Available Signals e clique em Add. O barramento está pronto, basta clicar em OK para retornar a tela de seleção de canal, onde o barramento estará na lista Available signals.

Escolhidos os sinais a serem apresentados, clique em ok para retornar ao analisador lógico.

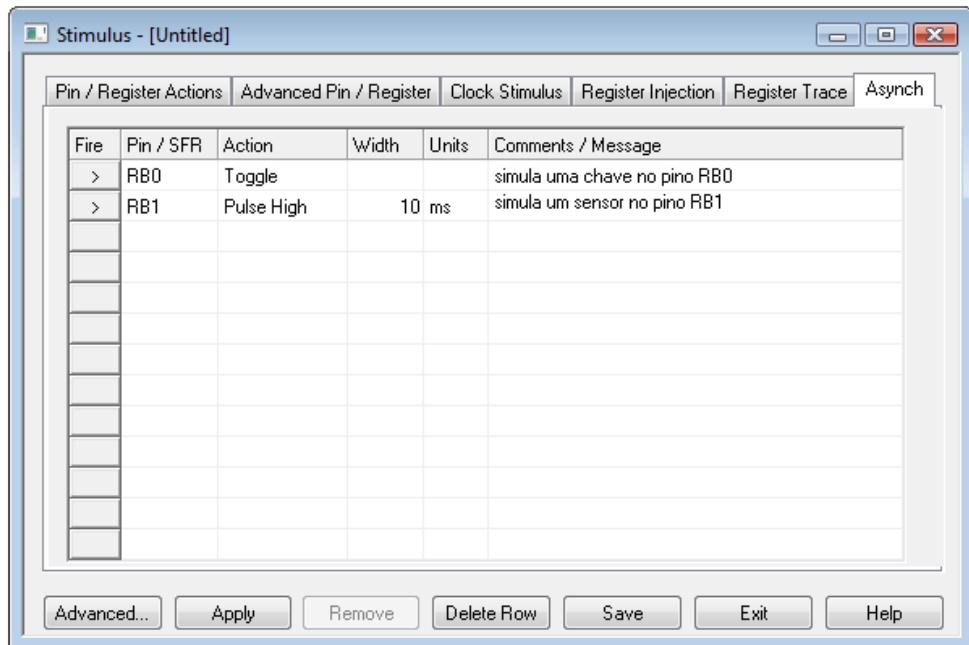
Ao simular o programa os sinais apresentados serão apresentados na tela do analisador. A figura abaixo apresenta um exemplo disso.



3.2.4 Geração de estímulos

Outra funcionalidade interessante do MPLAB SIM é a geração de estímulos que permite simular sinais aplicados a pinos ou sinais do microcontrolador. Para aplicar estímulos a simulação vá em Debugger ' Stimulus ' New Workbook.

Existe uma série de tipos de estímulos que podem ser gerados. Neste curso nos concentraremos nos estímulos assíncronos, que equivalem a chaves ligadas aos terminais do microcontrolador. Para configurar estes, selecione a aba Asynch da janela de stimulus, conforme a figura abaixo.



A coluna Fire contém botões que, quando pressionados no decorrer da simulação, irão realizar a ação descrita na coluna Action no sinal descrito na coluna Pin/SFR.

A coluna Pin/SFR mostra o sinal controlado em cada linha. Clicando em qualquer célula desta coluna se abrirá uma lista para a escolha do sinal desejado.

Na coluna Action escolhe-se a ação a ser realizada quando Fire é pressionado:

- Set High e Set Low: forcão o pino nível lógico alto e baixo, respectivamente;
 - Toggle: inverte o estado do pino;
 - Pulse High e Pulse Low: gera um pulso alto e baixo, respectivamente.

A coluna **Width** define a "largura", ou seja, a duração do pulso quando a coluna **Action** tem as opções **Pulse High** ou **Pulse Low**. A unidade de tempo do pulso, nesse caso, eh escolhida na coluna Units e pode ser uma unidade de tempo ou ciclos de máquina do microcontrolador (cyc).

Por fim, a coluna **Comments/Message** permite inserir um texto comentando a utilidade do estímulos.

Para salvar um conjunto de estímulos clique no botão Save. Para abrir um conjunto de estímulos salvos anteriormente vá em **Debugger** ⇒ **Stimulus** ⇒ **Open Workbook**.

O C18 permite gerar código objeto para microcontroladores PIC da família PIC18 a partir de códigos fonte escrito em C padrão ANSI. Ele é chamado de cross-compiler porque gera código para ser executado em outro processador (no caso, o PIC18) que não seja o que executa a compilação (que é feita no PC).

3.3.1 Outros Compiladores C

Existem diversos compiladores C para microcontroladores das linhas PIC12, PIC16 e PIC18 fornecidos por outras empresas que não a microchip. A maioria destes compiladores são pagos mas existem versões de demonstração que podem ser usadas com algumas limitações. O principal ponto a se observar ao usar um compilador C fornecido por terceiros é se o mesmo segue o padrão C-ANSI ou se apresenta alguma divergência com relação a este padrão. Essas divergências podem impedir uma migração direta os compiladores.

Outra observação importante é que as bibliotecas de funções apresentadas neste curso (com exceção daquelas padronizadas para o C-ANSI) só são válidas para o compilador C18. Outros compiladores podem até ter funções com a mesma finalidade mas provavelmente seu nome e sintaxe serão diferentes.

CCS

Um compilador muito popular é o CCS. Ao contrário do C18, o CCS também é capaz de compilar programas para PICs das famílias PIC12 e PIC16.

A apesar de ser se um compilador C, o CCS não é coerente com o padrão ANSI. Isso prejudica bastante a portabilidade de código entre o CCS e outros compiladores. As principais diferenças estão em nas diretivas específicas para microcontrolador e nos tipos de dados (principalmente o tipo int).

Em contra partida, o CCS permite portar código C entre as linhas PIC18 e PIC16 com bastante facilidade. Além disso, é muito rico em bibliotecas de funções para periféricos do microcontrolador e para acesso a outros dispositivos.

Mais informações sobre o compilador podem ser encontradas em [.](#)

Uma ótima referência sobre o compilador CCS é o livro "PIC - Programação em C", do autor Fábio Pereira.

PICC Lite (Hitech)

Juntamente com as últimas versões do MPLAB IDE vem sendo enviado o compilador PICC Lite da Hitech. Trata-se de um compilador ANSI, portanto permite uma fácil migração de e para o C18. Este compilador aceita componentes das linhas PIC10F, PIC12F e PIC16F, com algumas limitações. Um ponto fraco do compilador é que o mesmo não possui bibliotecas de funções específicas do microcontrolador, apenas aquelas constantes no padrão ANSI.

3.4 Depuração em C

O processo de depuração (debugging) consiste em realizar as mesmas ações possíveis na simulação, porém com o programa sendo realmente executado no microcontrolador. Dessa maneira é possível verificar o valor de variáveis a medida que elas são alteradas no circuito, inserir breakpoints e acompanhar a execução do programa sendo executado no hardware da aplicação.

Se a simulação permite testar a lógica de rotinas e pequenos programas a depuração é uma ferramenta mais poderosa quando se trata de programa de maior complexidade e principalmente na localização e correção de bugs no software.

Para realizar a depuração é necessário um hardware especial chamado XICD-2, embutido no kit XM118. Uma vez instalado e configurado (esse processo é apresentado no manual do kit) os mesmos comandos e janelas disponíveis para simulação ficam disponíveis, só que agora atuando no mundo real.

Capítulo 4

Linguagem C para microcontroladores

4.1 Introdução a linguagem C

Antes de tratarmos da linguagem C devemos apresentar algumas convenções que tornarão mais claro a explanação. Essa considerações referem-se a padronização ANSI (American National Standard for Information Systems).

Em primeiro lugar, comentários em C são indicados pelos símbolos ”/*”, ”*//” e ”//”. Comentários pertencem ao código fonte mas são ignorados pelo compilador quando da compilação. Os símbolos ”/*” e ”*//” informam que o que está entre é um comentário. O símbolo ”//” indica que toda linha a direita trata-se de um comentário. Na verdade, ”//” é um símbolo definido para a linguagem C++, mas a maioria dos compiladores C o aceita. Muitos programadores usam a convenção de se uma ”//” para comentários explicativos e ”/*” e ”*//” para manter ignoradas parte do código. A seguir temos alguns exemplos:

```
/* Isto é um comentário */
// Isto também é um comentário
/*Desta maneira podemos fazer comentários que, por algum motivo qualquer, ocupem diversas
linhas. Do mesmo modo podemos transformar trechos do programa em comentários */
```

Todo comando em C termina, obrigatoriamente, com um ”;”.

Os nomes de funções, variáveis, constantes, etc, são chamados de identificadores e não devem conter caracteres acentuados, espaços, ”ç” e devem começar com letras (o símbolo ”_” é tratado como letra). Letras maiúsculas e minúsculas são consideradas diferentes, isto é, ”Maria” é considerado um identificador diferente de ”maria”.

Existem palavras reservadas que não podem ser usadas como identificadores. As definidas pelo padrão ANSI são apresentas a seguir, mas pode haver outras próprias do compilador.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Tabela 4.1: Palavras reservadas pelo padrão ANSI.

Os programas em C são organizados em funções. A função **main**, como o nome em inglês indica, é a função principal do programa; por ela que o programa começa a ser executado. Todo programa deve ter pelo menos uma função (no caso de apenas uma, ela deve ser obrigatoriamente chamada main). Juntamente com a função main, criada pelo programador, o compilador C cria um código a ser executado antes da função main ser chamada. Sua função é realizar as configurações necessárias à execução do programa desenvolvido. Este código é chamado de startup code.

As chaves ({ e }) delimitam um bloco de instruções. Um bloco de instruções é visto e tratado como uma única instrução.

O operador de atribuição em C é o símbolo “=”. Os demais operadores serão tratados oportunamente.

A seguir temos um exemplo de um programa básico em C, onde podem ser observados diversos temas tratados. Observe que a tabulação, isto é, a distância entre o início dos comandos e o canto do da tela, é muito importante para a organização e compreensão do código.

```

/* Exsto Tecnologia */
/* Curso PIC Avançado */
/* José Domingos Adriano */
/* domingos@exsto.com.br */
/* (c)Exsto Tecnologia 2006 */
/* www.exsto.com.br */

#include<P18F4550.h> // definições para o PIC18F4550
#include<stdio.h> // Funções de saída de caracter
void main (void) // Função principal
{
    // Declaração de variáveis
    unsigned char x,z;
    int y;
    printf("\n Hello World!!!");
    printf("\n Exsto Tecnologia");
    x = 150;
    y = -12000;
}

```

```
z = 0;  
for(;;)  
y = y + 1;  
x = x + 1;  
if(x == 165){  
z = z + 1;  
x = 150;  
}  
}  
}
```

4.1.1 A função printf()

Uma função bastante usada em C é a função printf(). Essa função é definida pelo padrão ANSI e está contida na biblioteca stdio.h. A função printf() escreve dados na porta de saída padrão. Em computadores, a porta padrão é o monitor. No C18 porta de saída padrão é a porta serial.

Printf() é útil principalmente na depuração de programas, pois apresenta dados de diversas maneiras em forma de caracter, permitindo assim conhecer o conteúdo de variáveis. Como será mostrado em momento oportuno, é possível configurar o simulador do MPLAB para apresentar os dados enviados pela serial do PIC. Contudo, na prática o seu uso deve ser evitado, pois essa função consome muita memória e na maioria das vezes pode ser substituída por funções mais simples de envio de caracter ou string simples.

4.1.2 ”C ou não C, eis a questão”.

A escolha da linguagem a ser usada no desenvolvimento de um projeto com microcontroladores é um fator de extrema importância, pois impacta em diversos aspectos: tempo (e, portanto custo) de desenvolvimento, custo de ferramentas de desenvolvimento, portabilidade de código, facilidade de depuração, futuras alterações e atualizações tecnológicas, dentre outros.

No universo dos computadores pessoais há diversas opções de linguagens, podendo atender a necessidade específicas de cada desenvolvedor ou projeto. No universo dos microcontroladores existe também a possibilidade de se programar em diversas linguagens de programação de médio e alto nível, tais como C/C++, Pascal, Basic e Java. Em oposição a essas linguagens existe sempre a opção de se programar na linguagem Assembly nativa de cada microcontrolador. É muito discutida entre os profissionais da área qual a melhor linguagem de programação para microcontroladores. Essa discussão concentra-se principalmente na questão das vantagens e desvantagens de uma linguagem de médio ou alto nível em relação ao Assembly nativo do microcontrolador.

Obviamente deve-se considerar, em primeiro lugar, a linguagem mais adequada para cada aplicação em especial. Além disso, é importante considerar também o quanto o desenvolvedor está familiarizado com uma linguagem e o fator subjetivo de qual linguagem lhe agrada mais. Apesar de ter sempre em mente essas premissas, vamos expor aqui porque, na maioria das vezes e para grande parte dos desenvolvedores, a linguagem C é a mais adequada.

Há quem defende ferrenhamente que o desenvolvedor profissional deva trabalhar em Assembly. Os principais argumentos a favor desse modo de pensar são a eficiência (em termos de ocupação

de memória) do código gerado por um programador experiente, o menor tempo de execução do programa e o controle total, já que nada é feito sem que o programador saiba e queira. A eficiência de código é importante na medida em que a memória disponível nos componentes em questão é restrita e tem grande impacto no custo do componente. Também se perde o controle do código gerado, já que toda vez que um compilador é usado o código Assembly produzido sai do controle do desenvolvedor.

No passado, compiladores pouco confiáveis apresentavam erros de implementação difíceis de serem depurados. E ainda, se o conjunto de instruções não possuir certas características o uso de compiladores gera um código significativamente maior e mais lento que um de mesmas funções em Assembly.

Por outro lado, é mais difícil de se desenvolver algoritmos complexos em Assembly, tornando o trabalho de desenvolvimento mais lento e mais caro. Realizar operações matemáticas complexas (por exemplo, com variáveis do tipo ponto flutuante) torna-se bastante confuso. Como a linguagem nativa e os modelos de memória de cada arquitetura de microcontroladores diferem, é exigido um tempo muito longo de estudo para se adaptar a um novo conjunto de instruções. Instruções de mesma função podem ter mnemônicos totalmente diferentes em diferentes conjuntos de instrução. A grande quantidade de linhas de código e a pouca inteligibilidade tornam os processos de depuração demorados e difíceis. Sendo o Assembly uma linguagem de programação não estruturada os custos de desenvolvimento, depuração e manutenção tendem a ser maiores. Por fim, o reaproveitamento de código e a portabilidade de programas entre uma arquitetura e outra é quase impossível, resultando na reescrita de várias centenas ou milhares de linhas de programa.

No passado era possível a um desenvolvedor especializar-se nessa ou naquela arquitetura, tornando-se cada vez mais experiente no uso de um set de instruções e realizando todos seus projetos com alguns poucos microcontroladores compatíveis. Hoje, porém, a variedade de microcontroladores é muito grande. Praticamente todos os fabricantes de circuitos integrados digitais têm suas famílias de microcontroladores. A concorrência de preço entre esses fabricantes e a diversidade de configurações dos microcontroladores exige que as empresas optem por usar diferentes arquiteturas em diferentes projetos, ou mesmo substituir componentes em produtos já existentes. É necessário que o desenvolvedor de sistemas microcontrolados seja capaz de se adaptar aos diferentes modelos de memória e conjunto de instruções, com pouco tempo para o estudo detalhado de cada caso.

A solução então é utilizar uma linguagem de mais alto nível, que por princípio são independentes do hardware. O desenvolvedor concentra seu trabalho na criação de algoritmos e funcionalidades no projeto e deixa o trabalho de implementação para o compilador. Surge então outro problema: qual linguagem usar?

Para muitas pessoas a resposta é: o C! Vejamos por que.

Em primeiro lugar, o C é considerado uma linguagem de médio nível, isto é, mais amigável para o ser humano, mas mantendo recursos de baixo nível. Em virtude disso, o C permite gerar códigos mais eficientes quando comparado com outros compiladores. Há também a vantagem da independência do hardware que permite que um projeto seja portado de uma arquitetura para outra com poucas alterações no código. Essas características do C não são por acaso. A linguagem foi desenvolvida no início dos anos 70 por Dennis Ritchie especificamente para o desenvolvimento do sistema operacional UNIX. Na época o dilema era quase o mesmo: construir sistemas opera-

cionais em assembly era demorado e dispendioso e todo o trabalho tinha que ser refeito a cada novo processador criado. Os computadores da época tinham pouca memória e os modelos usados por Ritchie não eram capazes de implementar operações de ponto flutuante diretamente. Ritchie desenvolveu uma linguagem simples, fácil de ser aprendida, mas com recursos que permitem o desenvolvimento de programas complexos, como sistemas operacionais, que permite portabilidade de código sem prejudicar as funcionalidades de baixo nível, gerando um código bastante "enxuto", isto é, eficiente no uso da memória.

Por ser uma linguagem estruturada, o C exige que o programador siga boas práticas de programação, evitando assim erros que, em Assembly, poderiam passar despercebidos.

A portabilidade de código fica garantida, pois existe uma norma que rege a linguagem C. Todo compilador que segue o padrão ANSI deve conseguir reconhecer o mesmo código gerado, independente da arquitetura alvo. Os compiladores C profissionais são bastante confiáveis, geram código eficiente e livre de bugs de implementação.

Uma vantagem adicional do C são as bibliotecas de funções que acompanham os compiladores. Além das bibliotecas convencionais do padrão ANSI a maioria dos compiladores possui bibliotecas para uso das funcionalizadas específicas de cada microcontrolador. Além do ganho de tempo em se usar as bibliotecas já prontas existe também um ganho de eficiência de código, já que essas funções são normalmente implementadas em Assembly para otimização de memória e tempo.

Como o C é utilizado também em computadores, trata-se de uma linguagem muito difundida nas escolas e universidade. Também os profissionais que tiveram uma formação acadêmica voltada para o PASCAL terão facilidade em desenvolver em C.

Na tabela a seguir se pode observar uma comparação entre as linguagens C e Assembly.

Característica	Assembly	C
Eficiência de código	Ótima	Muito Boa
Velocidade de execução	Ótima	Muito Boa
Controle do sistema	Ótimo	Muito Bom
Portabilidade	Péssimo	Ótimo
Depuração	Muito difícil	Fácil
Programação	Não-estruturada	Estruturada
Bibliotecas de funções	Não Há	Padronizadas e específicas

Tabela 4.2: Comparação entre C e Assembly

Contudo, não se deve entender que o estudo da linguagem Assembly deve ser negligenciado. Assim como o bom piloto de corrida entende tanto da mecânica de seu carro quanto os engenheiros que o projetaram, o bom desenvolvedor deve ter sólidos conhecimentos da arquitetura, modelo de memória e conjunto de instruções Assembly do microcontrolador com o qual trabalha. É imprescindível conhecer bem o assembly, até para poder conseguir os melhores resultados em C. Sempre é possível analisar o código Assembly gerado pelo compilador, buscando formas de otimizá-lo. Além disso, em situações críticas, é possível inserir linhas de código Assembly dentro de um programa escrito em C.

Por essas e por outras razões os fabricantes de microcontroladores têm, já há alguns anos, desenvolvido arquiteturas de microcontroladores otimizadas para o uso da linguagem C. Alguns tipos de instruções e modos de endereçamento são incluídos para tornar mais fácil a tradução do código C para o código de máquina. Isso acontece com o PIC18 no set de instruções convencional

e mais pronunciadamente no set estendido. Como adaptações do processador ao C podem citar as instruções voltadas para endereçamento indireto e para leitura de constantes na memória de programa, além da presença de diversos ponteiros para endereçamento indireto de variáveis (facilitam a implementação de vetores, matrizes e variáveis do tipo ponteiro).

Esperamos ter demonstrado que a linguagem C tem sido cada vez mais usada no desenvolvimento com microcontroladores, trazendo diversas vantagens que justificam seu uso em aplicações profissionais.

Uma última observação deve ser feita quanto ao uso de linguagens orientadas a objeto. Apesar da orientação a objeto ser uma tendência na programação e trazer inegáveis benefícios ao desenvolvimento de software, hoje ainda ela não usada em larga escala. Uma provável causa é o maior consumo de memória que a técnica acarreta. Contudo já existem compiladores que permitem trabalhar de forma orientada a objetos (em C++, Java, etc). No futuro otimizações na arquitetura dos microcontroladores serão feitas para tornar o código orientado a objeto mais leve aos microcontroladores.

4.1.3 Passado e Futuro da linguagem C

Desde seu surgimento no início dos anos 70, a linguagem C sofreu alterações, ampliações, tornou-se um padrão estável e gerou descendentes.

C foi desenvolvido inicialmente para a implementação do sistema operacional Unix. Como o Unix se popularizou e foi portado para várias plataformas diferentes, foi necessário implementar compiladores C para diferentes arquiteturas. Foi então que a características de portabilidade do C se tornou mais evidente.

Em meados dos anos 80 a ANSI (American National Standard for Information Systems) criou uma norma padronizando o C, criando uma língua comum a ser reconhecida por todos os compiladores C.

Em virtude da evolução da computação, o C gerou várias linhas de evolução. Para atender aos conceitos da programação orientada a objeto foi desenvolvido o C++. Seguindo essa linha, surgiu a linguagem Java. Para aplicações para Internet a linguagem PHP é descendente direto do C. Para aplicações para celulares foi criado o C#. Além disso, várias ferramentas usam C/C++ para desenvolver código com interface gráfica amigável, como o Visual C++ ou o Borland C++ Builder.

4.2 Diretivas

Diretivas podem ser descritas como comandos dados ao compilador, indicado a ações que devem ser feitas no momento da compilação. Elas podem ou não gerar código que efetivamente vai ser executado pelo microcontrolador, já que sua função é orientar a compilação do programa.

O padrão ANSI C prevê uma grande variedade de diretivas. Além disso, o padrão prevê uma diretiva para criação de novas diretivas, chamada `#pragma`, de forma que diferentes compiladores possam oferecer a seus usuários opções que não estão previstas no padrão ANSI. Das diversas diretivas disponíveis enfocaremos apenas e quem tem maior uso neste curso, apresentadas abaixo.

4.2.1 #include

A diretiva `#include` permite incluir arquivos de cabeçalho e bibliotecas de funções.

Exemplo:

```
#include <P18F4550.h> // inclui um arquivo de definição do PIC
#include <spi.h> // inclui a biblioteca de funções SPI.c
```

É como se, ao se chegar a essa diretiva, o compilador "colasse" (incluisse) o conteúdo do arquivo apontado.

4.2.2 #define

Esta diretiva é muito útil pois permite "apelidar" um trecho de texto por um nome. `#define` define que um nome representa o texto imediatamente a seguir, de forma que quando da compilação toda ocorrência desse nome seja substituído pelo texto. Vejamos os exemplos a seguir:

Exemplo:

```
#define VALOR_MAXIMO 125
#define LED POTRDbits.RD0
#define ACENDE_LED LED = 1;
```

No primeiro exemplo, o valor 125 foi chamado de `VALOR_MAXIMO`. Esse tipo de construção é útil quando temos um limite ou constante que queremos usar em diferentes partes do código, mas que pode mudar seu valor devido a alguma necessidade do projeto. Chamando de `VALOR_MAXIMO` o valor 125, se for necessário mudá-lo para, por exemplo, 203 basta alterar na definição, e não em todos os pontos do código.

No segundo caso, usamos `#define` para atribuir um nome a um pino do microcontrolador. Dessa forma, não precisamos fazer sempre referência ao pino, chamamo-lo por seu apelido `LED`. Isto facilita o entendimento do código e possíveis alterações na configuração do hardware.

No último caso usamos `#define` para apelidar um trecho de código, criando o que é conhecido como **macro**.

4.2.3 #pragma config

Esta é uma diretiva específica do compilador C18, criada com o uso da diretiva `#pragma`. Ela permite definir as configurações de características especiais e oscilador do PIC, estudadas anteriormente. A diretiva admite uma série de parâmetros. Esses parâmetros mudam para cada PIC, conforme suas próprias características. Para obter a informação detalhada para cada PIC, consulte o arquivo `hlpC18ConfigSet` que se encontra na pasta `C:\MCC18\doc`. Para o PIC18F4550, temos as opções de configuração descritas abaixo, sendo que estão em negrito as configurações **default**, isto é, aquelas reconhecidas caso nenhuma escolha tenha sido feita. Deve-se estar atento à quais escolher, pois uma configuração errada ocasionará um mal funcionamento do programa.

PLL Prescaler Selection	
PLLDIV = 1	Sem pré-escala (4 MHz)
PLLDIV = 2	Divide por 2 (8 MHz)
PLLDIV = 3	Divide por 3 (12 MHz)
PLLDIV = 4	Divide por 4 (16 MHz)
PLLDIV = 5	Divide por 5 (20 MHz)
PLLDIV = 6	Divide por 6 (24 MHz)
PLLDIV = 10	Divide por 10 (40 MHz) - somente no modo EC
PLLDIV = 12	Divide por 12 (48 MHz) - somente no modo EC

Tabela 4.3: PLL Prescaler Selection

Essa configuração permite usar diferentes valores de cristais como sinal de entrada do PLL para o módulo USB.

CPU System Clock Postscaler		
CPUDIV = OSC1_PLL2	Sem PLL $F_{XTAL} / 1$	Com PLL $F_{PLL} / 2$ (48 MHz)
CPUDIV = OSC2_PLL3	$F_{XTAL} / 2$	$F_{PLL} / 3$ (32 MHz)
CPUDIV = OSC3_PLL4	$F_{XTAL} / 3$	$F_{PLL} / 4$ (24 MHz)
CPUDIV = OSC4_PLL6	$F_{XTAL} / 4$	$F_{PLL} / 6$ (16 MHz)

Tabela 4.4: CPU System Clock Postscaler

Essa configuração permite selecionar o fator de divisão do clock do sistema. Quando usado como modos de oscilador sem PLL ele divide o frequência do cristal (Fxtal) por 1, 2, 3 ou 4. Quando usado em modos com oscilador, divide a frequência da saída do PLL (96MHz) por 2, 3, 4 e 6, obtendo-se as frequências indicadas na tabela.

USB Clock Selection		
USBDIV = 1	Clock de USB vem direto do oscilador principal, sem pré-escala (baixa velocidade)	
USBDIV = 2		Clock de USB vem do PLL (alta velocidade)

Tabela 4.5: USB Clock Selection

Esta opção permite escolher entre o modo de baixa o alta velocidade, para USB. No modo de baixa velocidade a única frequência de entrada aceita é 6MHz.

Oscillator Selection		
	Oscilador de CPU	Oscilador de USB
FOSC = XT_XT	XT	XT
FOSC = XTPPLL_XT	XT com PLL	XT com PLL
FOSC = ECIO_EC	EC (clock externo), RA6 usado como I/O	EC (clock externo)
FOSC = EC_EC	EC (clock externo), RA6 saída de clock	EC (clock externo)
FOSC = ECPLLIO_EC	EC (clock externo) com PLL, RA6 usado como I/O	EC (clock externo) com PLL
FOSC = ECPLL_EC	EC (clock externo) com PLL, RA6 saída de clock	EC (clock externo) com PLL
FOSC = INTOSCIO_EC	Oscilador interno, RA6 usado como I/O	EC (clock externo)
FOSC = INTOSC_EC	Oscilador interno, RA6 saída de clock	EC (clock externo)
FOSC = INTOSC_XT	Oscilador interno	XT
FOSC = INTOSC_HS	Oscilador interno	HS
FOSC = HS	HS	HS
FOSC = HSPLL_HS	HS com PLL	HS com PLL

Tabela 4.6: Oscillator Selection

Seleciona o tipo de oscilador principal. Observe que a seleção de modo define qual será o oscilador da CPU e qual será de USB.

Fail-Safe Clock Monitor Enable	
FCMEN = OFF	Fail-Safe Clock Monitor desabilitado
FCMEN = ON	Fail-Safe Clock Monitor habilitado

Tabela 4.7: Oscillator Selection

Se o Fail-Safe Clock Monitor estiver habilitado é gerada uma interrupção caso o oscilador principal pare de operar e o oscilador interno assume como clock de sistema.

Internal/External Oscillator Switchover	
IESO = OFF	Não permite mudança de oscilador externo para interno
IESO = ON	Permite mudança de oscilador externo para interno

Tabela 4.8: Internal/External Oscillator Switchover

A configuração acima permite ou não que se troque entre as fontes de clock interno e externo.

Power-up Timer Enable	
PWRT = ON	Power-up timer habilitado
PWRT = OFF	Power-up timer desabilitado

Tabela 4.9: Power-up Timer Enable

Brown-out Reset Enable	
BOR = OFF	Brown-out Reset desabilitado em hardware e software
BOR = SOFT	Brown-out Reset habilitado e controlado por software(SBOREN está desativado)
BOR = ON_ACTIVE	Brown-out Reset habilitado somente em hardware e desabilitado no mode Sleep (SBOREN está desativado)
BOR = ON	Brown-out Reset habilitado somente no hardware(SBOREN está desativado)

Tabela 4.10: Brown-out Reset Enable

Brown-out Voltage	
BORV = 0	4,5 V
BORV = 1	4,2 V
BORV = 2	2,7 V
BORV = 3	2,0 V

Tabela 4.11: Brown-out Voltage

USB Voltage Regulator Enable	
VREGEN = OFF	Regulador de USB ativo
VREGEN = ON	Regulador de USB inativo

Tabela 4.12: USB Voltage Regulator Enable

Quando o periférico USB estiver ativo, deve existir uma tensão de 3,3V no pino VUSB. Essa tensão pode ser externa ou gerada internamente por um regulador, controlado pela opção acima.

Watchdog Timer Enable	
WDT = OFF	HW Disabled - SW Controlled
WDT = ON	HW Enabled - SW Disabled

Tabela 4.13: Watchdog Timer Enable

Ao usar o watchdog lembre-se que é necessário zerar periodicamente esse contador no software, senão o microcontrolador reseta.

Watchdog Timer Postscale Select bits	
WDTPS = 1	1:1
WDTPS = 2	1:2
WDTPS = 4	1:4
WDTPS = 8	1:8
WDTPS = 16	1:16
WDTPS = 32	1:32
WDTPS = 64	1:64
WDTPS = 128	1:128
WDTPS = 256	1:256
WDTPS = 512	1:512
WDTPS = 1024	1:1024
WDTPS = 2048	1:2048
WDTPS = 4096	1:4096
WDTPS = 8192	1:8192
WDTPS = 16384	1:16384
WDTPS = 32768	1:32768

Tabela 4.14: Watchdog Timer Postscale Select bits

Para determinar o tempo total entre estouro do watchdog, multiplique o valor da pré-escala por 4ms.

MCLR Pin Enable	
MCLRE = OFF	MCLR (Reset) desabilitado; RE3 habilitado como entrada
MCLRE = ON	MCLR (Reset) habilitado; RE3 desabilitado

Tabela 4.15: MCLR Pin Enable

Low-Power Timer 1 Oscillator Enable	
LPT1OSC = OFF	Oscilador do Timer1 em modo de maior consumo e maior precisão
LPT1OSC = ON	Oscilador do Timer1 em modo de menor consumo e menor precisão

Tabela 4.16: Low-Power Timer 1 Oscillator Enable

PORTB A/D Enable	
PBADEN = OFF	Pinos de PORTB [4:0] configurados como entrada digital após Reset
PBADEN = ON	Pinos de PORTB [4:0] configurados como entrada analógica após Reset

Tabela 4.17: PORTB A/D Enable

Para usar os pinos do PORTB como entrada ou saídas digitais, mesmo com a função de interrupção, é necessário desativar as entradas do A/D ou configurar PBADEN = OFF.

CCP2 MUX	
CCP2MX = OFF	Pino CCP2 conectado a RB3
CCP2MX = ON	Pino CCP2 conectado a RC1

Tabela 4.18: CCP2 MUX

Stack Full/Underflow Reset Enable	
STVREN = OFF	Desabilita reset se ocorrer overflow/underflow da pilha
STVREN = ON	Habilita reset se ocorrer overflow/underflow da pilha

Tabela 4.19: Stack Full/Underflow Reset Enable

Single-Supply ICSP Enable	
LVP = OFF	Gravação em modo de baixa tensão desabilitada
LVP = ON	Gravação em modo de baixa tensão habilitada

Tabela 4.20: Single-Supply ICSP Enable

O kit XM118 suporta apenas gravação em modo de alta tensão, portanto essa opção deve estar desabilitada sempre.

Dedicated In-Circuit Debug/Programming Port (ICPORT) Enable	
ICPRT = OFF	ICPORT desabilitado
ICPRT = ON	ICPORT habilitado

Tabela 4.21: Dedicated In-Circuit Debug/Programming Port (ICPORT) Enable

Essa configuração permite usar pinos não conectados do encapsulamento de 44 pinos (SMD) como pinos de gravação e depuração. Portanto, essa opção só é válida nesses encapsulamentos, quando for usado encapsulamentos DIP ele não tem efeito.

Extended Instruction Set Enable	
XINST = OFF	Modo de instruções estendido desabilitado.
XINST = ON	Modo de instruções estendido habilitado.

Tabela 4.22: Extended Instruction Set Enable

Background Debugger Enable	
DEBUG = ON	Depurador habilitado, RB6 and RB7 são dedicados a depuração in-circuit.
DEBUG = OFF	Depurador desabilitado, RB6 and RB7 pode ser usados como I/O.

Tabela 4.23: Background Debugger Enable

A opção DEBUG=ON deve ser escolhida quando o kit for usado como depurador.

As opções a seguir referem-se a propriedade de proteção de código pra diversos blocos da memória. Para os blocos em que essa propriedade estiver ativa não é possível fazer uma leitura externa da memória.

Code Protection bit Block 0	
CP0 = ON	Bloco 0 (000800-001FFFh) protegido.
CP0 = OFF	Bloco 0 (000800-001FFFh) não protegido.

Tabela 4.24: Code Protection bit Block 0

Code Protection bit Block 1	
CP1 = ON	Bloco 1 (002000-003FFFh) protegido.
CP1 = OFF	Bloco 1 (002000-003FFFh) não protegido.

Tabela 4.25: Code Protection bit Block 1

Code Protection bit Block 2	
CP2 = ON	Bloco 2 (004000-005FFFh) protegido.
CP2 = OFF	Bloco 2 (004000-005FFFh) não protegido.

Tabela 4.26: Code Protection bit Block 2

Code Protection bit Block 3	
CP3 = ON	Bloco 3 (006000-007FFFh) protegido.
CP3 = OFF	Bloco 3 (006000-007FFFh) não protegido.

Tabela 4.27: Code Protection bit Block 3

Boot Block Code Protection bit	
CPB = ON	Bloco de Boot (000000-0007FFh) protegido.
CPB = OFF	Bloco de Boot (000000-0007FFh) não protegido.

Tabela 4.28: Boot Block Code Protection bit

A proteção de código para EEPROM impede que ela seja lida externamente.

Data EEPROM Code Protection bit	
CPD = ON	Data EEPROM code-protected.
CPD = OFF	Data EEPROM not code-protected.

Tabela 4.29: Data EEPROM Code Protection bit

A proteção de escrita de memória FLASH impede que o bloco protegido possa ser escrito pelo próprio microcontrolador.

Write Protection bit Block 0	
WRT0 = ON	Bloco 0 (000800-001FFFh) protegido.
WRT0 = OFF	Bloco 0 (000800-001FFFh) não protegido.

Tabela 4.30: Write Protection bit Block 0

Write Protection bit Block 1	
WRT1 = ON	Block 1 (002000-003FFFh) protegido.
WRT1 = OFF	Block 1 (002000-003FFFh) não protegido.

Tabela 4.31: Write Protection bit Block 1

Write Protection bit Block 2	
WRT2 = ON	Block 2 (004000-005FFFh) protegido.
WRT2 = OFF	Block 2 (004000-005FFFh) não protegido.

Tabela 4.32: Write Protection bit Block 2

Write Protection bit Block 3	
WRT3 = ON	Block 3 (006000-007FFFh) protegido.
WRT3 = OFF	Block 3 (006000-007FFFh) não protegido.

Tabela 4.33: Write Protection bit Block 2

Boot Block Write Protection	
WRTB = ON	Boot block (000000-0007FFh) protegido.
WRTB = OFF	Boot block (000000-0007FFh) não protegido.

Tabela 4.34: Boot Block Write Protection

Configuration Register Write Protection	
WRTC = ON	Bits de configuração (300000-3000FFh) protegido.
WRTC = OFF	Bits de configuração (300000-3000FFh) não protegido.

Tabela 4.35: Configuration Register Write Protection

A proteção de escrita de memória EEPROM impede que a mesma possa ser escrita pelo próprio microcontrolador.

Configuration Register Write Protection	
WRTD = ON	EEPROM protegido.
WRTD = OFF	EEPROM não protegido.

Tabela 4.36: Data EEPROM Write Protection

Os blocos de memória protegidos contra leitura não poderão ser lidos pelo próprio microcontrolador.

Table Read Protection bit Block 0	
EBTR0 = ON	Bloco 0 (000800-001FFFh) protegido.
EBTR0 = OFF	Bloco 0 (000800-001FFFh) não protegido.

Tabela 4.37: table EEPROM Write Protection

Table Read Protection bit Block 1	
EBTR1 = ON	Bloco 1 (002000-003FFFh) protegido.
EBTR1 = OFF	Bloco 1 (002000-003FFFh) não protegido

Tabela 4.38: Table Read Protection bit Block 1

Table Read Protection bit Block 2	
EBTR2 = ON	Bloco 2 (004000-005FFFh) protegido.
EBTR2 = OFF	Bloco 2 (004000-005FFFh) não protegido

Tabela 4.39: Table Read Protection bit Block 2

Table Read Protection bit Block 3	
EBTR3 = ON	Bloco 3 (006000-007FFFh) protegido.

Tabela 4.40: Table Read Protection bit Block 3

Boot Block Table Read Protection	
EBTRB = ON	Bloco de Boot (000000-0007FFh) protegido.
EBTRB = OFF	Bloco de Boot (000000-0007FFh) não protegido

Tabela 4.41: Boot Block Table Read Protection

Abaixo é apresentado um exemplo de código fazendo a configuração do microcontrolador conforme os comentários.

Exemplo:

```
#pragma config PLLDIV = 5      // PLL para 20MHz
#pragma config CPUDIV = OSC1_PLL2  // Oscilador principal a 48MHz
#pragma config FOSC = HS      // Oscilador HS para CPU e USB
#pragma config WDT = OFF     // Watchdog desativado
#pragma config PBADEN = OFF   // Pinos do PORTB começam como digitais
#pragma config LVP = OFF      // Desabilita gravação em baixa tensão
#pragma config DEBUG = ON     // habilita debug
```

4.2.4 #pragma interrupt e #pragma interruptlow

Estas diretivas permitem definir que uma função é uma rotina de tratamento de interrupção (RTI) de baixa ou de alta prioridade. Ao fazer isso o compilador se encarrega de descrever o código específico para chamada, salvamento e retorno de contexto da interrupção. Além disso, a diretiva permite passar uma lista de variáveis (variáveis declaradas pelo usuário e registros do microcontrolador) que devem ter seu contexto salvo quando ocorrer interrupção. Sua sintaxe é apresentada a seguir.

```
#pragma interrupt nome_da_RTI_alta_prioridade [save = lista_de_variaveis]
#pragma interruptlow nome_da_RTI_baixa_prioridade [save = lista_de_variaveis]
```

Mais detalhe e exemplos do uso dessa na seção que trata de interrupções em C.

4.3 Tipos de dados

Ao contrário do que ocorre em Assembly, em C existem diversos tipos de dados, de tamanhos e faixas de valores diferentes. Basta que uma variável seja declarada como sendo de um determinado tipo e o compilador se encarrega de gerar o código necessário para sua manipulação.

O padrão ANSI define diversos tipos de dados para os compiladores C: char, int, float, double e void.

O tipo void serve para indicar que o tipo parâmetro é nulo nas funções, ou seja, que o parâmetro não existe. Mais detalhes sobre passagem de parâmetros para funções serão apresentados no tópico que trata de funções. O tipo char é capaz de armazenar um caracter simples. O tipo int armazena um dado do tipo inteiro, tipicamente do tamanho de barramento de dados do processador, mas não deve ser menor que 16 bits. Float armazena dados do tipo ponto flutuante (fracionários) de precisão simples e double armazena dados de ponto flutuante de dupla precisão.

Além dos tipos de dados padrão existem também modificadores que permitem alterar as características de cada tipo de dado. São eles:

- **signed:** informa que o dado é do tipo sinalizado (valores positivos e negativos);
- **unsigned:** informa que o dado é do tipo não sinalizado (somente valores positivos);
- **long:** informa que o inteiro é uma variação maior que o padrão;
- **short:** informa que o inteiro é uma variação menor que o padrão.

Como os modificadores short e long atuam somente sobre dados do tipo int podemos escrever apenas o modificador, o int fica subentendido.

Quando um dado é do tipo sinalizado, um de seus bits é utilizado para indicação do sinal, o que reduz pela metade o valor absoluto que esse dado pode conter. Os tipos de dados inteiros que ocupam mais de um endereço de memória têm sua parte menos significativa nos endereços mais baixos e a parte mais significativa nos endereços mais altos.

A tabela a seguir apresenta os tipos de dados tratados pelo compilador C18, bem como seu tamanho e faixa de valores.

Tipo	Tamanho	Faixa de valores	
		Mínimo	Máximo
char	8 bits	-128	127
signed char	8 bits	-128	127
unsigned char	8 bits	0	255
int	16 bits	-32.768	32.767
unsigned int	16 bits	0	65.536
Short	16 bits	-32.768	32.767
Unsigned short	16 bits	0	65.536
Short long	24 bits	-8.388.608	8.388.607
Unsigned shot long	24 bits	0	16.777.215
Long	32 bits	-2.147.483.648	2.147.483.647
Unsigned long	32 bits	0	4.294.967.295

Tabela 4.42: Tipos da dados inteiros

Os tipos de dados float e double para variáveis com parte fracionárias (chamados de floating points - ponto decimal flutuante) definidos para C18 como variáveis de 32 bits, que comportam-se de forma idêntica. A tabela a seguir traz os tipos de dados suportados em C bem como as faixas de valores:

Tipo	Tamanho	Faixa de valores	
		Mínimo	Máximo
float	32 bits	2^{-126} (aprox. 1.17549435e - 38)	$2^{128} * (2-2-15) \approx 6.80564693e + 38$
double	32 bits	2^{-126} (aprox. 1.17549435e - 38)	$2^{128} * (2-2-15) \approx 6.80564693e + 38$

Tabela 4.43: Tipos da dados inteiros

4.3.1 Declaração de variáveis e constantes

A declaração de variáveis é feita conforme o exemplo a seguir:

```
char x,y;
shor long medida;
unsigned int dado = 0x35;
float pi = 3.1459;
```

Coloca-se primeiro o identificador de tipo, acompanhado ou não do modificar, seguido pela lista de variáveis desse tipo, separadas por ",". Ao final da declaração é necessário acrescentar o sinal ";" . Além disso, é possível definir um valor inicial para a variável, conforme é apresentado na terceira e quarta linhas do exemplo.

A declaração de constantes é feita de forma bastantes semelhante, bastando acrescentar o identificador "const" antes da declaração. Observe que é necessário informar o tipo de dado da constante, conforme o exemplo a seguir.

Exemplo:

```
const unsigned int SetPoint = 60000;
const char Letra = 'A';
```

A representação de valores nas diferentes bases é feita conforme a tabela a seguir, onde "y" representa o valor a ser expresso. A símbolo usado para separar as partes inteiras e fracionária de um número é o ponto (.). A representação de caracteres é feita entre apóstrofos (').

Tipo	Notação	Exemplos
Decimal	yyy	31, 1025, -3, 50000
Hexadecimal	0xffff	0x31, 0xABCD, 0x0008
Binário	0bffff	0b10101010, 0b1111000011110000
Octal	0o777	0o31, 0o123, 0o1543

Tabela 4.44: Representação de valores

4.3.2 Variáveis locais e globais

Uma variável pode ser declarada de forma global ou local. Variáveis locais são declaradas dentro de funções ou blocos de instruções (delimitados por chaves) e são reconhecidas somente dentro da função ou do bloco. Variáveis globais são declaradas fora das funções e são reconhecidas por todas as funções do programa.

O uso de variáveis globais é necessário em algumas situações quando determinada variável deve ser vista e alterada por diversas funções, como, por exemplo, na troca de dados entre funções de tratamento de interrupção e outras funções. Contudo, seu uso deve ser restrito, pois ocupam uma posição de memória constantemente.

Já as variáveis locais são criadas quando a função é executada e "destruídas" quando a função retorna. Isso significa que o mesmo endereço de memória pode ser usado por diversas variáveis no decorrer do programa, o garantindo um melhor uso de memória.

Caso variáveis locais tenham o mesmo nome de variáveis globais, as globais serão ignoradas naquela função (ou bloco).

4.3.3 Vetores e matrizes

É possível criar vetores e matrizes de dados dos tipos básicos. Vetores e matrizes são conjuntos de diversas variáveis de mesmo tipo que são referenciadas por um mesmo identificador, sendo discriminadas por um índice. Vetores são unidimensionais, enquanto matrizes podem ter duas dimensões ou mais. Todo vetor ou matriz é terminado por um caractere de valor 0.

Para atribuir valores à constantes ou valores iniciais à variáveis enumera-se os valores de cada elemento entre chaves, separados por vírgulas. Um caso particular de vetores são os vetores de caracteres, chamados de strings, que são úteis para armazenar palavras e frases. Nesses casos, o conjunto de caracteres pode ser atribuído a um vetor constante simplesmente usando aspas para marcar início e fim. Esse processo é apresentado no exemplo abaixo. Uma string é sempre terminada com um caractere igual a zero. Este caractere serve para marcar o final de uma string, uma vez que nenhum caractere é representado por zero.

Na criação de um vetor ou matriz é necessário informar o número de elementos que essa contém. Isso é feito colocando o número de elementos entre colchetes na declaração. No caso de um vetor de caracteres constantes não é preciso colocar o número de elementos, pois esse é determinado pelo número de elementos atribuídos.

Para fazer referência a um elemento em particular dentro de uma matriz ou vetor usa-se a posição desse elemento (índice). Índices podem ser valores fixos ou variáveis. Atenção: em C as posições começam em 0, portanto em um vetor de 10 elementos o índice poderá assumir valores de 0 a 9.

A seguir são apresentados alguns exemplos de declaração de variáveis.

Exemplo:

```
int Medidas[16];
const char nome[] = "Domingos";
char Empresa[] = "Exsto Tecnologia";
unsigned int InitLCD[6] = 0x33,0x32,0x28,0x06,0x0C,0x01;
```

O acesso a dados em vetores e constantes é feitas através do uso de índice, conforme mostrado no exemplo abaixo.

```
unsigned char i;
int x;
int Medidas[10];
const int Valores[5] = 0x25, 0x31, 0x33, 0x02, 0x55;
...
Medida[i] = 15;
x = Valores[3];
```

4.3.4 Ponteiros

Um tipo especial de variáveis é o tipo ponteiros. Ponteiros são variáveis que contém o endereço de outras variáveis. Sua função é ”apontar” onde a variável está na memória, daí seu nome. Ponteiros são muito usados em C e um dos motivos é que permitem a implementação do código de forma bastante otimizada. Uma arquitetura de microcontrolador otimizado para a linguagem C deve ter um eficiente sistema de ponteiros.

Para declarar um ponteiro usamos uma sintaxe parecida com a declaração de variável, porém com significado diferente. Deve ser colocado o tipo de dado que o ponteiro aponta e o nome do ponteiro, precedido por ”*”. Por exemplo, na declaração a seguir x é uma variável do tipo inteiro enquanto ip é um ponteiro que aponta para variáveis do tipo inteiro.

Exemplo:

```
int x; // x é um inteiro
int *ip'; // ip é um ponteiro que aponta inteiros
```

Para atribuir a um ponteiro o endereço de uma variável usamos o operador ”&”. Para manipular o conteúdo da variável apontada pelo ponteiro usamos o operador ”*”. O exemplo a seguir ilustra alguns usos de ponteiros.

```
int x = 1, y = 2, z[10]; // x e y são inteiros,
// z é um vetor de inteiros
int *ip, *iq; // ip e iq são ponteiros que
// apontam inteiros

ip = &x; // ip aponta para x
iq = &y; // iq aponta para y
y = ip; // y agora vale 1
```

```
ip = 0; // x agora vale 0  
ip = &z[0]; // ip agora aponta para z[0]  
iq = ip; // iq agora aponta para Z[0] também
```

4.3.5 Qualificadores

Existem alguns qualificadores que indicam como os dados devem ser alocadas na memória.

São naturais da linguagem C os qualificadores static e auto. O qualificador auto indica que a variável é valida somente naquele bloco de instruções (bloco delimitado por chaves ou função), sendo alocada quando o bloco começa e desalocada quando terminar. Desalocar, ou destruir, significa que aquele endereço de memória pode ser usado por outras variáveis do programa. O qualificador static informa que a variável não pode ser desalocada, portanto seu endereço de memória não será ocupado por outras variáveis e seu conteúdo não será alterado após a finalização do bloco. Seu comportamento é semelhante ao de variáveis globais, porém só são reconhecidas dentro do bloco ao qual pertencem.

Quando não especificado, as variáveis são consideradas como auto.

O compilador C18 introduz 4 novos qualificadores: ram e rom, near e far.

Os qualificadores ram e rom definem se o dado (variável ou constante) deve ser alocado na memória de dados (RAM) ou de programa (ROM). A princípio isso parece não fazer sentido, pois imagina-se que variáveis devam ser alocadas na RAM e constantes na memória de programa. Contudo, para aumentar a velocidade da manipulação de dados, constantes podem ser carregadas na memória de dados quando o programa é iniciado. Em outras situações, pode ser desejável que garantir que grandes tabelas de constantes estejam armazenadas na FLASH para reduzir o consumo de memória RAM. As variáveis declaradas em memória de programa devem, obrigatoriamente, ser do tipo static.

Quando não especificado, o C18 assume que os dados devem ser alocados na RAM. Isso é válido tanto para variáveis com para constantes.

Os qualificadores near e far (perto e longe em inglês, respectivamente) informam em que região da memória os dados devem ser alocados. Para dados na memória de programa, near (perto) significa nos primeiros 64 kbytes de memória. Para memória de dados, near significa na RAM de acesso. Em ambos os casos, far significa em qualquer região da memória, inclusive nas regiões abrangidas por near.

Quando não especificado, o C18 trata os dados como qualificados com far.

Seguem abaixo alguns exemplos de usos dos modificadores.

Exemplo:

```
rom static char Senha[4];// variável alocada na memória flash  
const char Nome[] = "João"; // constante alocada na RAM  
rom char *ptrSenha; // ponteiro para variável na FLASH  
near int Resultado; // resultado é alocado na RAM de Acesso  
static int cont; // Cont é do tipo estático.
```

4.4 Comandos Básicos

O C é uma linguagem de programação estruturada. Isso significa que segue seus comando básicos tem o objetivo de implementar os três tipos básicos de estruturas: atribuição, decisão e repetição. Além disso, possui diversos operadores que permitem realizar operações lógicas e matemáticas.

Todos os exemplos dados a seguir podem ser simulados para sua melhor compreensão. Para tanto, os exemplos devem ser copiados para dentro do arquivo SIM_C.ASM, do projeto SIM_C.MCP, que se encontra no CD que acompanha a apostila.

4.4.1 Atribuição

Comando de atribuição é realizado simplesmente com o uso do operador “=”. O identificador a esquerda recebe o valor da expressão à direita. Alguns exemplos são apresentados a seguir.

Exemplo:

```
x = 2;  
y = 3;  
x = 2 + y;  
y = x;
```

Nesses casos devemos evitar o uso do termo “igual” para evitar equívocos. Dizemos, por exemplo, “x recebe dois” e não “x igual a dois”.

O C permite substituir a expressão “x = x + 5” por “x += 5”, como o mesmo efeito.

Acesso aos SFR

No arquivo de cabeçalho que existe para cada microcontrolador PIC (ex.: P18F4550.h) estão definidos todos os registros de uso especial. Dessa maneira, é possível manipular diretamente os registros como se fossem variáveis e envolvê-los em operações com os operadores estudados. Os exemplos abaixo mostram com isso pode ser feito:

Exemplo:

```
#include<P18F4550.h>  
void main(void){  
    TMR0 = 0x25; // carrega valor no timer 0  
    T1CON = 0b0001001; // configura o timer 1  
    TRISA = 0xAA; // configura direção das portas  
    INTCON = 0xC0; // habilita interrupções  
}
```

Da mesma maneira, estão definidos os bits de cada registro, na forma de estruturas, conforme os exemplos a seguir. Basta colocar o nome registro seguido de "bits." e o nome do bit em questão (muita atenção com as letras, pois C diferencia maiúsculo de minúsculo.)

Exemplo:

```
#include<P18F4550.h>

void main(void)
RCONbits.IPEN = 1; // ativa prioridade de int.
INTCONbits.GIEH = 1; // habilita interrupções
ADCON0bits.ADON = 0; // desliga o ADC
TXSTAbits.TXEN = 1; // Ativa recepção serial
}
```

Entrada e Saída em C18

O compilador C18, assim como qualquer compilador C, não define comandos ou funções especiais para realizar operações de entrada e saída de dados. Mas como no PIC18 os portais de I/O são registros de funções especiais e, portanto, definidos no arquivo de cabeçalho de cada microcontrolador, podemos manipulá-los como se fossem variáveis.

Os registros que controlam a direção dos pinos dos portais, isto é, o registro TRISx, podem ser acessados por dois nomes: TRISx (TRISA, TRISB, TRISC, etc...) e DDRx (DDRA, DDRB, DDRC, etc.). DDRx vem de Direction Data Register - Registro de Direção de Dados. Pode-se configurar o portal todo de uma vez ou os registros individualmente. O exemplo a seguir trata do portal B, mas o mesmo raciocínio pode ser estendido aos demais portais.

Operadores em C

Para a execução de expressões e condições o C dispõe de diversos operadores, além do operador '=' (atribuição) já tratado.

Os operadores matemáticos são apresentados na tabela abaixo.

Operador	Operação
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da Divisão
++	Incremento
--	Decremento

Tabela 4.45: Operadores matemáticos

A seguir estão alguns exemplos de uso de operadores matemáticos.

Exemplo:

```
int x, y = 3, z = 2;
x = y + z; // o valor de x será 5
x = y - z; // o valor de x será 1
x = y * z; // o valor de x será 6
x = y / z; // o valor de x será 1 (parte inteira)
x = y % z; // o valor de x será 1 (resto da divisão)
```

Há também os operadores de incremento (++) e decremento (--). Numa atribuição, se usarmos os operadores a **direita** do dado, seu valor vai ser alterado (incrementado ou decrementado) **antes** da atribuição; se os usarmos a esquerda, seu valor será alterado depois. Observemos o exemplo a seguir.

Exemplo:

```
y = 5; // y recebe 5
x = ++y; // y contém 6 e o valor de x é 6
y = 5; // y recebe 5
x = y++; // y contém 6 MAS x contém 5
```

Existem também os operadores relacionais utilizados para comparar dados.

Operador	Operação
>	Maior que
>=	Menor ou igual que
<	Menor que
<=	Menor ou igual que
==	Igual a
!=	Diferente de

Tabela 4.46: Operadores Comparativos

Uma operação relacional resulta em "verdadeiro" ou "falso". Os exemplos a seguir mostram o resultado de algumas operações relacionais.

Exemplo:

Para $x = 10$ e $y = 8$:

- $x < y$, Verdadeiro
- $x \neq y$, Verdadeiro
- $x \mid y$, Falso

x != y,Falso
 x == y,Falso
 x != y ,Verdadeiro

Os operadores lógicos são divididos em dois tipos. O primeiro tipo realiza operações lógicas de comparando o estado de cada operando, se 'falso' (igual a 0) ou 'verdadeiro' (diferente de 0). Esses operadores permitem criar condições com expressões lógicas, envolvendo inclusive operadores relacionais. Mais exemplos de sua aplicação serão apresentados quando se tratar de comandos de decisão. Tais operadores são apresentados na tabela a seguir.

Operador	Operação
&&	E (AND)
	OU (OR)
!	COMPLEMENTO (NOT)

Tabela 4.47: Operadores Comparativos

O segundo tipo de operador lógico realiza operações lógicas bit a bit em seus operandos. Os operadores pertencentes a esse segundo tipo estão na tabela a seguir.

Operador	Operação
&	E (AND)
	OU (OR)
^	OU-EXCLUSIVO (XOR)
~	COMPLEMENTO (NOT)
>>	Deslocamento a direita
<<	Deslocamento a esquerda

Tabela 4.48: Operadores Comparativos

Os exemplos a seguir apresentam operações lógicas bit-a-bit envolvendo dados de 8 bits.

Exemplo:

```

char a,b,s;
a = 0b11110000;
b = 0b10101010;
s = a & b; // o valor de x será 0b10100000
s = a | b; // o valor de x será 0b11110000
s = a ^ b; // o valor de x será 0b01011010
s = ~b; // o valor de x será 0b01010101
    
```

No deslocamento informa-se quantos bits serão deslocados e as posições vagas são preenchidas com zero. No exemplo abaixo é feito um deslocamento de 3 bits a direita e depois 2 bits a esquerda.

Exemplo:

```
char a,b,s;
```

```
a = 0b11110000;
s = a >> 3; // o valor de x será 0b00011110
s = a << 2; // o valor de x será 0b11000000
```

Por fim, existem operadores de manipulação de memória:

Operador	Operação
&	Endereço do identificador
*	Conteúdo do endereço apontado pelo operando

Tabela 4.49: operadores de manipulação de memória

O operador '&' deve ser usado com atenção. Se ele for usado com apenas um operando, tem a função de retornar o endereço de memória desse operando. Mas caso seja usado com dois operandos, sua função é a operação lógica E. Essas duas formas de aplicação são apresentadas nos exemplos abaixo.

Exemplo:

X = &Y; // X recebe o endereço de Y

X = Y & 0x0F // X recebe o resultado da operação E bit-a-bit de Y com 0x0F.

Precedência de operadores

Numa expressão com vários operadores as operações serão efetuadas seguindo uma ordem predefinida relacionada com tipo de operados. O programador pode alterar essa ordem conforme a sua vontade utilizando parênteses, que funcionaram como em uma operação matemática. A seguir é a ordem de precedência dos operadores.

Ordem de execução	Operador
1	() []
2	! ++ - * &(endereço)
3	/ %
4	+ -
5	<<>>
6	<<=>>=
7	== !=
8	& (operação E)
9	^
10	
11	&&
12	
13	?
14	= += -= *= /=

Tabela 4.50: Operadores de manipulação de memória

Exemplo:

```
// da na mesma fazer
TRISB = 0b00110101;
// ou
DDRB = 0b00110101;
// ou
TRISBbits.TRISB0 = 1; // 1 = Entrada
TRISBbits.TRISB1 = 0; // 0 = Saída
TRISBbits.TRISB2 = 1;
TRISBbits.TRISB3 = 0;
TRISBbits.TRISB4 = 1;
TRISBbits.TRISB5 = 1;
TRISBbits.TRISB6 = 0;
TRISBbits.TRISB7 = 0;
// ou ainda
DDRBbits.RB0 = 1;
DDRBbits.RB1 = 0;
DDRBbits.RB2 = 1;
DDRBbits.RB3 = 0;
DDRBbits.RB4 = 1;
DDRBbits.RB5 = 1;
DDRBbits.RB6 = 0;
DDRBbits.RB7 = 0;
```

Para acesso aos portais existem dois registros, LATx e PORTx. O acesso a esse registros pode ser feito conforme apresentado no exemplo a seguir.

Exemplo:

```
LATA = 0x00;
x = PORTB;
PORTBbits.RB0 = 1;
LATBbits.LATB1 = PORTAbits.RA5;
```

Para tornar o código mais comprehensível e facilitar alterações na configuração de hardware podemos usar a diretiva #define para atribuir nomes aos pinos dos portais, conforme apresentado a seguir.

Exemplo:

```
// Esse programa acende o LED se a chaves estiver pressionada
#include<P18F4550.h>
```

```
#define LED0 PORTDbits.RD0
#define Chave PORTBbits.RB1
void main (void)
{
    DDRDbits.RD0 = 0; // saída para o LED
    for(;;)
        if (!Chave)LED = 1;
        else LED = 0;
}
```

4.4.2 Decisão

As estruturas de decisão permitem ao programa tomar decisões baseadas na avaliação de uma condição. É avaliado se essa condição é verdadeira ou falsa.

Em C temos as seguintes estruturas de decisão.

If

A estrutura if (se) executa um bloco de comandos se a condição for verdadeira. Caso contrário segue a execução do programa. Seu aspecto geral é apresentado abaixo, assim como um exemplo de utilização.

Exemplo:

```
if (<condição>)
    <Bloco verdadeiro>
}
```

Exemplo:

```
if(x > 5)
    printf("\n x maior que 5");
}
```

Uma condição é considerada falsa se é igual a 0. Caso contrário é considerada verdadeira. Isso se aplica não só ao resultado de operadores relacionais como também na avaliação de variáveis e bits, conforme mostrado a seguir.

Exemplo:

```
if (TMR0){ // verdadeiro se TMR0 diferente de 0
    printf("TMR0 é diferente de 0")
}
if(!PORTBbits.RB0){ //verdadeiro se RB0 for '0' (note o '!')
    LigaMotor();
}
```

```
if(PIR1bits.TMR1IF){ RTITimer1(); // chama a função se TMR1IF for '1'  
}
```

If-else

A estrutura if-else (se-senão) executa um bloco de comandos se a condição for verdadeira ou outro bloco de comando se a condição for falsa. Temos a seguir forma geral dessa estrutura e um exemplo de utilização.

Comando exemplo:

```
if (jcondição){  
<Bloco verdadeiro>  
}  
else{  
<Bloco falso>  
}
```

Exemplo:

```
if(x > 5){  
printf("\n x maior que 5");  
}  
else{  
printf("\n x menor que 5");  
}
```

Operador ternário condicional

Para situações onde uma decisão implica no valor a ser atribuído a uma variável existe uma forma alternativa de se realiza a estrutura se-senão. Isso é feito através do operador ternário condicional. Sua sintaxe é:

Variável = Condição ? Expressão1 : Expressão2;

Sendo **Condição** uma expressão cujo resultado vai ser avaliado em verdadeiro ou falso e **Expressão1** e **Expressão2** expressões quaisquer cujo valor pode ser atribuído à variável. Se **Condição** for verdadeiro, **Variável** recebe o resultado de **Expressão1**; senão, **Variável** recebe o resultado de **Expressão2**. A seguir o uso do operador ternário condicional é exemplificado.

Exemplo:

```
x = (y >= 10) ? y * 2 : y * 3;  
equivale a  
if( y >= 10) x = y * 2;  
else x = y * 3;  
PORTDbits.RD0 = (valor <= 128) ? 1 : 0;  
equivale a  
if( valor <= 128) PORTDbits.RD0 = 1;  
else PORTDbits.RD0 = 1;
```

Switch

O comando switch permite a implementação de uma estrutura caso. A seguir temos a forma genérica desse comando.

Exemplo:

```
switch(<variável>){  
    case <valor1>:  
        <comandos>  
        break;  
    case <valor1>:  
        <comandos>  
        break;  
    case <valor1>:  
        <comandos>  
        break;  
    default:  
        <comandos>  
        break;
```

O que o comando **switch** faz é comparar <variável> com os valores que ela pode assumir (valor1, valor2, etc). Caso a variável possua um desses valores o bloco associado ao valor é executado. O último bloco, chamado default, é opcional e só é executado se <variável> não é igual a nenhum dos valores.

A seguir, como exemplo, temos uma aplicação típica do comando switch:

Exemplo:

```
switch(x){  
    case 1:  
        printf("\n x igual a 1");  
        break;  
    case 2:  
        printf("\n x igual a 2");  
        break;  
    case 3:  
        printf("\n x igual a 3");  
        break;  
    default:  
        printf("\n x diferente de 3, 2 e 1");  
        break;  
}
```

4.4.3 Repetição

Em C existem três comandos que implementam estruturas de repetição:

- Do-While;
- While;
- For.

Do-While

Esse comando implementa uma estrutura Faça-Enquanto. Sua função executar um blocos de instruções enquanto uma condição for satisfeita. Sua principal característica é que o bloco é executado antes do teste ser realizado. Ou seja, o bloco é executado pelo menos uma vez, independente da condição ser verdadeira ou falsa. Esse funcionamento é particularmente útil quando a condição a ser testada é afetada pelo bloco. A seguir temos sua forma padrão e um exemplo de utilização.

Exemplo da estrutura do comando:

```
do{  
    <bloco de instruções>  
}while(<condição>);
```

Exemplo:

```
// fica imprimindo a mensagem enquanto n < 5; do{  
    printf("\n Bloco do do-while!");  
}
```

```
}while(n < 5);
```

While

Da mesma forma de Do-While, o comando **While** repete um bloco enquanto uma determinada condição é verdadeira. A principal diferença entre os dois comandos é que com While o **teste é feito antes** de se executar o bloco. Em outras palavras, o bloco só vai ser executado se em algum momento a condição for satisfeita.

Exemplo da estrutura do comando:

```
while(<condição>){  
    <bloco de instruções>  
}
```

Exemplo:

```
// fica imprimindo a mensagem enquanto n diferente de x  
while(n != n){  
    printf("\n Bloco do While!!");  
}
```

For

O comando For é utilizado para repetir um bloco por um número pré-determinado de vezes. Sua estrutura básica é apresentada a seguir.

Exemplo da estrutura do comando:

```
for(<Contador> = <ValorInicial>; <Condição>; <Passo>){  
    <Bloco>  
}
```

Obrigatoriamente deve existir uma variável contadora (<contador>). Essa variável é carregada com um valor inicial quando a estrutura for começa a ser executada. A cada repetição do bloco de instruções é dado um "passo" em <contador>, isto é, seu valor é alterado de alguma maneira. Tipicamente o passo é um incremento ou um decremento. O bloco de instruções é repetido enquanto a condição que relaciona <contador> com um valor final é satisfeita.

No exemplo a seguir for é usado para apresentar valores de uma multiplicação. A repetição foi implementada com um passo de decremento, o que quer dizer que é subtraído 1 do contador (cujo valor inicial é 10) até que a condição (contador > 0) não seja mais satisfeita.

Exemplo:

```
for(i = 10; i > 0; i- - ){  
    n = 5*i;  
    printf("\n 5 vezes %d igual a %d",i,z);  
}
```

Comando Break

Quando executado dentro de um laço de repetição, o comando break causa a imediata interrupção, fazendo o programa saltar para fora do laço.

4.5 Funções

A linguagem C permite a criação de funções (equivalentes a sub-rotinas em Assembly). Além disso, várias funções já prontas são fornecidas com o compilador C18, sejam funções definidas pelo padrão ANSI sejam funções para uso específico no PIC18.

Uma função é um trecho de código que deve executar uma função específica e bem definida. Seu objetivo é permitir que um programa seja modular, isto é, composto de blocos cujas responsabilidades dentro do programa são bem claras. Modularidade é um princípio da programação estruturada e C é uma linguagem estruturada por natureza.

Funções bem construídas facilitam a depuração, manutenção e entendimento do código. E ainda, podem ser transportadas para outros programas onde se façam necessárias.

Uma coleção de funções correlacionadas é normalmente agrupada em uma biblioteca. Por exemplo, a biblioteca "math.h" é uma biblioteca de funções matemáticas avançadas.

Na pasta c:\MCC18\doc existe um arquivo hlpC18UG. Nele são apresentadas a sintaxe (a forma como as funções devem ser chamadas) tanto das bibliotecas padronizadas pela ANSI como as bibliotecas específicas do C18.

4.5.1 Passagem de parâmetros

Assim como uma função matemática (daí seu nome) uma função em linguagem C tem operadores, chamados de parâmetros.

Há parâmetros de entrada (que passam informações para a função), também chamados de argumentos, e parâmetros de saída (trazem informação gerada na execução da função). Assim como para variáveis e constantes, é necessário informar o tipo de dado dos parâmetros, sendo válidas as mesmas convenções usadas para declaração de variáveis. Há casos onde não há parâmetros de entrada e/ou de saída. Nesse caso eles são definidos como sendo do tipo void.

A seguir temos a sintaxe geral da definição de uma função.

Exemplo de sintaxe:

```
<tipo> nome_da_funcao(<tipo1> <arg1>, <tipo2> <arg2>, ... ) {
```

```
/* geralmente aqui temos uma declaração de variáveis */  
/* os comando da função vêm em seguida */  
return(<var>); /* retorno o parâmetro de saída */  
}
```

Para exemplificar, suponhamos que se deseja construir uma função que some o valor de dois inteiros e retorne o resultado dessa soma, também em formato inteiro. Essa função poderia ser desenvolvida da seguinte maneira:

Exemplo:

```
int SomaDoisNumeros(int x, int y)  
// Função que soma dois números  
// Entrada: x e y contendo os valores  
// Saída: a soma de x e y  
{  
    int z; // Declaração de variáveis  
    z = x + y; // comandos da função  
    return(z); // retorna o resultado  
}
```

O nome da função é SomaDoisNumeros e obedece as mesmas regras para nomes de variáveis. Seu parâmetro de saída é do tipo int. Diz-se que "SomaDoisNumeros retorna um inteiro". Seus argumentos são dois dados do tipo inteiro. Quaisquer dois dados do tipo dos argumentos podem ser passados para a função, sejam eles constantes, variáveis ou valores imediatos. Esse valores são copiados para as variáveis internas x e y.

As variáveis declaradas em uma função, assim como seus argumentos só existem dentro da função. Isso significa que podemos ter outras variáveis chamadas x em outras funções e estas não têm qualquer relação com a variável x que existe em SomaDoisNumeros.

O exemplo a seguir ilustra como a função pode ser chamada na função principal.

Exemplo:

```
void main(void)  
{  
    int x,z;  
    z = 5;  
    x = SomaDoisNumeros(8,z);  
}
```

Em C a posição onde a função se encontra no código fonte é importante. Uma função (incluindo a função main) só pode chamar outra que esteja escrita acima (antes) de si mesma. Isso pode ser feito organizando a ordem em que as funções aparecem no código ou através da definição do "protótipo" de todas as funções no início do arquivo. Por exemplo, o protótipo da função SomaDoisNumeros é:

```
int SomaDoisNumeros(int x, int y);
```

Funções não "enxergam" variáveis declaradas fora de si mesma (com exceção de variáveis globais) assim como não é possível "enxergar" as variáveis de fora de uma função. Em outras palavras, a chamada de uma função não pode alterar o valor de variáveis passadas como argumentos. O método para realizar isso, que também pode ser usado para funções que retornam vetores ou mais de um parâmetro, é chamado de passagem de parâmetro por referência e é feito com o uso de ponteiros.

Por exemplo, admitindo a necessidade de construir uma função capaz de somar 5 a três variáveis, pode-se escrever o seguinte função:

Exemplo:

```
void Soma5em3Variaveis(int *ptrA, int *ptrB, int *ptrC)
// Função que soma 5 aos argumentos
// Entrada: os endereços de três variáveis do tipo int
// Saída: as variáveis referenciadas
{
    ptrA = *ptrA + 5;
    ptrB = *ptrB + 5;
    ptrC = *ptrC + 5;
}
```

Os argumentos da função são na verdade os endereços das variáveis. A função manipula o conteúdo das variáveis apontadas por esses endereços, portanto toda alteração feita dentro da função é feita com variáveis que foram declaradas fora da função.

O mesmo processo pode ser usado para argumentos que sejam string ou vetor. Usando o nome da string ou do vetor como argumento, o ponteiro é associado ao primeiro elemento. O exemplo abaixo mostra como poderia ser usada uma função que imprime os valores contidos em uma string.

Exemplo:

```
void ImprimeString(char *ptrA)
{
    while(*ptrA != '\0'){ // repete até o fim da string
        printf("\n %c", *ptrA);
        ptrA++; // incrementa o endereço
    }
}
```

```
}
```

```
void main (void)
```

```
{
```

```
const char Seq[] = "Exsto Tecnologia";
```

```
ImprimeString(Seq);
```

```
}
```

4.6 Interrupções

Muitas vezes um projeto exige que providências sejam tomadas imediatamente no momento em que um determinado evento ocorre, porém não se pode ficar o tempo todo verificando as entradas associadas a esse evento. Por exemplo, supondo que um sistema que deve fazer várias medidas continuamente e mostrar uma medida selecionada em um display. Essa medida é selecionada pelo pressionamento de um botão. Vamos admitir que o processo de fazer todas as medidas e mostrá-las no display demore um determinado tempo, alguns segundos. Então só poderemos verificar o estado do botão rapidamente a intervalos de alguns segundos. Se o botão não estiver pressionado no exato momento da leitura o microcontrolador não tomará conhecimento desse fato. O ideal é que o microcontrolador pudesse fazer as medidas o tempo todo e somente no momento em que o botão fosse pressionado ele executaria a rotina relacionada a esse evento, voltando depois às medidas.

Uma interrupção faz exatamente isso: interrompe a execução normal do programa para executar tarefas de maior prioridade no momento. Ela pode ser chamada por determinados eventos, parando a execução normal no momento em que ocorre o evento, executando o código armazenado em uma posição de memória vinculada a interrupção e depois retornam a execução normal do programa. As interrupções podem ser consideradas chamadas de sub-rotinas realizadas pelo hardware.

Quando ocorre um pedido de interrupção para a CPU, o programa é desviado para um endereço pré-estabelecido, o vetor de reset. No PIC18 existem dois vetores de reset, conforme será tratado mais adiante. Desse endereço em diante é feito o chamado tratamento de interrupção, isto é, são executadas as rotinas que se deseja associar ao evento causador da interrupção.

O PIC18F4550 possui 20 fontes de interrupções. A cada interrupção são associados três bits:

- Bit de habilitação: Indica se a interrupção está ativa (0 = interrupção desativada, 1 = interrupção ativa);
- Bit de flag: indica se ocorreu o evento associado à interrupção (0 = não ocorreu, 1 = ocorreu);
- Bit de prioridade: indica a prioridade da interrupção (0 = baixa prioridade, 1 = alta prioridade).

Existem vários "eventos" causadores de interrupção, como por exemplo, uma transição em alguns pinos, o estouro de um timer, a recepção de um dado pela porta serial, etc. a ocorrência

de um evento destes faz com que um bit de flag vai para 1, caso contrário ele permanecerá em 0. Contudo, para que haja um pedido de interrupção para a CPU é necessário que a interrupção associada a esse evento esteja ativa, isto é, seu bit de habilitação seja 1. Em outras palavras, é possível ativar quais interrupções se queira e manter as demais inativas.

Existem dois níveis de prioridade de interrupções, alto e baixo. Uma interrupção de baixa prioridade interrompe a execução do programa e causa o desvio para o vetor de interrupção de baixa prioridade. Já a interrupção de alta prioridade interrompe não só a execução normal do programa como também interrompe a rotina de tratamento de interrupção de baixa prioridade (se estiver sendo executada). Cada fonte de interrupção pode ser configurada para atuar com baixa ou alta prioridade, sendo que para isso basta definir se o bit de prioridade é 0 (baixa) ou 1 (alta).

O vetor de interrupção de alta prioridade é o 0008h enquanto o vetor de interrupção de baixa prioridade é o 0018h.

Para habilitar o sistema de prioridade de interrupção o bit IPEN deve estar em '1'. Caso contrário (IPEN = '0') o sistema comporta-se como se não houvesse prioridade, como nos PIC16. Neste caso, todas as interrupções convergem para o endereço 0008h. Estando habilitada a prioridade existem duas habilitações globais, uma para prioridade alta (GIEH) e outra para baixa (GIEL). Caso contrário, existe uma habilitação global (GIE) e uma habilitação de periféricos (PEIE).

A interrupção INT0 não tem ajuste de prioridade, ela é sempre de prioridade alta.

A figura a seguir apresenta o diagrama interno de interrupções.

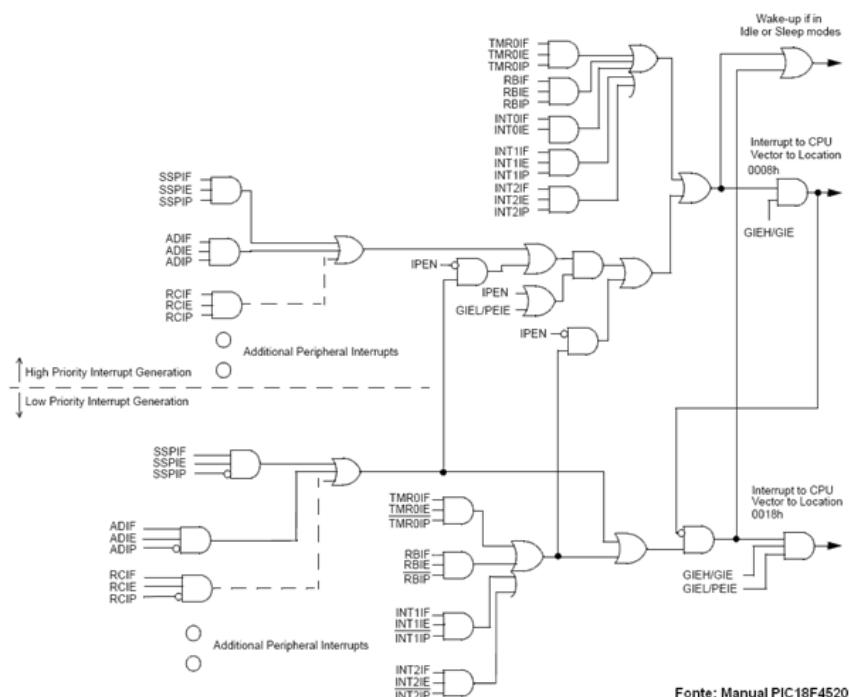


Figura 4.1: Diagrama de interrupções do PIC18F

Dois fatos importantes sobre os bits de flag devem ser enfatizados. O bit de flag vai para 1 quando ocorre o evento mesmo que a interrupção associada a ele não esteja habilitada. Além disso o hardware força o bit de flag para 1 mas o mesmo deve ser zerado pelo software para indicar que a interrupção em questão foi tratada.

As interrupções externas, isto é, que são causadas diretamente por eventos externos ao microcontrolador, é a interrupção INT0 (pino RB0), INT1 (pino RB1), INT2 (pino RB2) e a interrupção por mudança de estado do portal B. As demais interrupções são internas, causadas pelos periféricos para indicar determinados eventos (chegada de dado pelo portal serial, "estouro" de timer, etc).

Quando ocorre uma interrupção o fluxo do programa desvia para um vetor de reset (conforme a prioridade da interrupção). Consequentemente, a rotina de tratamento de interrupção deve obrigatoriamente começar nesse endereço. Para determinar qual interrupção ocorreu testa-se os bits de flags de todas as interrupções ativas com a mesma prioridade.

Pode acontecer de uma rotina de tratamento de interrupção fazer uso de uma variável ou registro de função especial também usada no programa principal. Neste caso, a execução da rotina de tratamento de interrupção iria corromper o valor desta variável, causando um erro intermitente e difícil de detectar. Para entender esse problema basta pensar que se em determinado ponto do programa movemos o valor 10 para a variável X e ocorre uma interrupção neste exato momento, o valor de W será alterado na RTI e o valor que finalmente aparecerá em X não será 10. Ao contrário de alguns microcontroladores, quando ocorre uma interrupção nenhum registro é automaticamente salvo. Assim é importante salvar os registros W e STATUS antes que seja realizado qualquer tratamento de interrupção, bem como qualquer outro SFR que for usado dentro e fora das rotinas de interrupção. Esse processo é chamado salvamento de contexto. Também é recomendável que o registro STATUS seja zerado. É necessário então que existam variáveis criadas para salvar esses registros. Outra prática recomendada é de nunca usar dentro de uma rotina de tratamento de interrupção registros que são utilizados em outras partes do programa.

Um outro cuidado que deve ser tomado quando se trabalha com interrupções é não executar rotinas muito grandes. Isso porque o tratamento de interrupção deve tomar medidas emergências. Nada que não seja estritamente necessário deve ser trabalhado na interrupção, e isso principalmente porque podem ocorrer várias interrupções em seguida, e só se pode tratar uma por vez. Quando uma interrupção é chamada, a habilitação das demais fica desativada, e só é reativada após a execução do RETFIE. Se outra interrupção ocorrer nesse intervalo de tempo, ela é tratada após RETFIE. Deve-se notar que o que ativa a chamada de interrupção é a existência de um flag de uma interrupção habilitada setado. Assim, depois de tratar uma determinada interrupção deve-se zerar o flag associado a ela. Caso contrário, o programa fica travado, pois cada vez que há o retorno de interrupção, a interrupção é chamada novamente.

Finalmente, é necessário considerar que na realidade o tratamento de interrupção não ocorre no exato momento em que a interrupção acontece. Primeiramente, a checagem dos eventos que causam a interrupção obedece a uma varredura interna. E depois de detectado o evento, algumas ações devem ser realizadas, como o desvio para o vetor de reset, por exemplo. Assim, após a real ocorrência de uma interrupção ela só vai começar a ser tratada 3 ciclos de máquina após sua ocorrência, se for uma interrupção interna, e de 3 a 4 ciclos de máquina se for uma interrupção externa. Esses tempos devem ser considerados em aplicações onde a precisão de tempo é importante.

4.6.1 Utilização das Interrupções

Para trabalhar com uma interrupção deve-se primeiramente configurar o periférico associado a ela, o que é feito normalmente uma única vez, na inicialização programa. Nada impede, porém que um periférico seja configurado, ou re-configurado, em qualquer ponto programa.

O próximo passo é habilitar as interrupções a serem utilizadas, e isso pode ser feito em dois momentos. Pode-se determinar que uma interrupção esteja ativa durante toda a execução do programa, e nesse caso sua habilitação deve ser feita na inicialização. Por outro lado, em algumas situações deseja-se que a interrupção seja ativada somente a partir de um dado momento ou ainda que seja ativada e posteriormente desativada. Nessa situação basta habilitar, ou desabilitar, a interrupção no momento oportuno.

É importante lembrar que as interrupções habilitadas somente causam chamadas de interrupção se o bit de habilitação global (GIEH e GIEL quando a prioridade estiver ativa, GIE caso contrário) estiver habilitado.

Finalmente, deve-se fazer o "tratamento" de interrupção, isto é, construir as rotinas que serão chamadas quando ocorrer uma determinada interrupção. Essas rotinas deverão seguir as recomendações dadas acima e deve-se salvar os registros de interesse. As rotinas de tratamento de interrupção serão apresentadas no item da unidade de software que trata de interrupções.

Os registros que controlam o sistema de interrupção são apresentados a seguir.

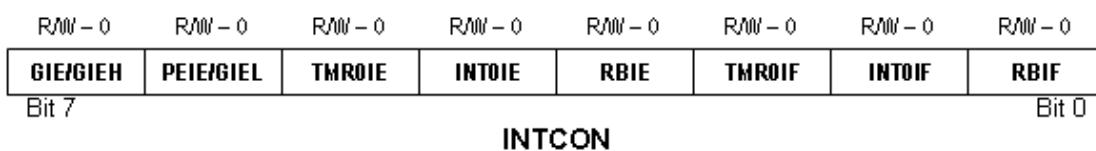


Figura 4.2: Registro INTCON

- GIE/GIEH : Habilitação Global;
 - Se IPEN = 0;
 - * 1 = Habilita as interrupções;
 - * 0 = Desabilita as interrupções;
 - Se IPEN = 1;
 - * 1 = Habilita as interrupções de alta prioridade;
 - * 0 = Desabilita interrupções de alta prioridade;
- PEIE: Habilitação de interrupções de periféricos
 - Se IPEN = 0
 - * 1 = Habilita as interrupções de periféricos
 - * 0 = Desabilita as interrupções de periféricos
 - Se IPEN = 1
 - * 1 = Habilita as interrupções de baixa prioridade
 - * 0 = Desabilita interrupções de baixa prioridade

- Habilitação de interrupção do Timer 0
 - 1 = interrupção habilitada
 - 0 = interrupção desabilitada
- INT0IE: Habilitação de interrupção Externa 0 (RB0)
 - 1 = interrupção habilitada
 - 0 = interrupção desabilitada
- RBIE: Habilitação de interrupção por mudança de estado no PORTB
 - 1 = interrupção habilitada
 - 0 = interrupção desabilitada
- TMR0IF: Flag da interrupção do Timer 0
 - 1 = Houve overflow do timer 0
 - 0 = Não houve overflow do timer 0
- INT0IF: Flag da interrupção Externa 0 (RB0)
 - 1 = Houve transição no pino RB0
 - 0 = Não houve transição no pino RB0
- RBIF: Flag da interrupção por mudança de estado no PORTB
 - 1 = houve alteração em um dos bits de RB7:RB4
 - 0 = não houve alteração em um dos bits de RB7:RB4

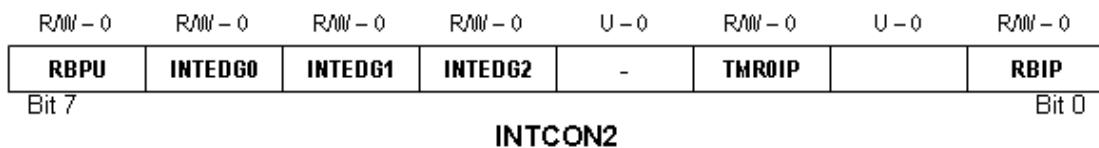


Figura 4.3: Registro INTCON 2

- RBPU : Habilitação dos Pull-ups de PORTB
 - 1 = desabilita Pull-ups
 - 0 = Habilita Pull-ups
- INTEDG0: Seleção da transição da interrupção INT0
 - 1 = Transição de subida
 - 0 = Transição de descida
- INTEDG1: Seleção da transição da interrupção INT1

- 1 = Transição de subida
 - 0 = Transição de descida
 - INTEDG2: Seleção da transição da interrupção INT2
 - 1 = Transição de subida
 - 0 = Transição de descida
 - TMR0IP: Prioridade da Interrupção do Timer 0
 - 1 = Alta prioridade
 - 0 = Baixa prioridade
 - RBIP: Prioridade da Interrupção de PORTB
 - 1 = Alta prioridade
 - 0 = Baixa prioridade

Figura 4.4: Registro INTCON 3

- INT2IP: Prioridade da Interrupção INT2
 - 1 = Alta prioridade
 - 0 = Baixa prioridade
 - INT1IP: Prioridade da Interrupção INT1
 - 1 = Alta prioridade
 - 0 = Baixa prioridade
 - INT2IE: Habilitação de interrupção Externa 2 (RB2)
 - 1 = interrupção habilitada
 - 0 = interrupção desabilitada
 - INT1IE: Habilitação de interrupção Externa 1 (RB1)
 - 1 = interrupção habilitada
 - 0 = interrupção desabilitada
 - INT2IF: Flag da interrupção Externa 2 (RB2)
 - 1 = Houve transição no pino RB2
 - 0 = Não houve transição no pino RB2

- INT1IF: Flag da interrupção Externa 1 (RB1)
 - 1 = Houve transição no pino RB1
 - 0 = Não houve transição no pino RB1

As demais interrupções tem seus bits de flag, habilitação prioridade distribuídos nos registros PIRx, PIEx e IPRx, respectivamente, como mostrado na tabela a seguir:

Evento de Interrupção	Flag	Habilitação	Prioridade
	1 = ocorreu 0 = não ocorreu	1 = Habilitada 0 = Desabilitada	1 = Alta 0 = Baixa
	INTCON	INTCON	INTCON2
Interrupção externa INT0	INT0IF	INT0IE	Sempre alta
Estouro do timer 0	TMR0IF	TMR0IE	TMR0IP
Mudança de estado na porta B	RBIF	RBIE	RBIP
	INTCON3	INTCON3	INTCON3
Interrupção externa INT1	INT1IF	INT1IE	INT1IP
Interrupção externa INT2	INT2IF	INT2IE	INT2IP
	PIR1	PIE1	IPR1
Leitura ou escrita da porta paralela escrava	PSPIF	PSPIE	PSPIP
Conversão do AD completa	ADIF	ADIE	ADIP
Recepção serial USART	RCIF	RCIE	RCIP
Transmissão serial	TXIF	TXIE	TXIP
Transmissão/recepção completa pelo MSSP	SSPIF	SSPIE	SSPIP
Interrupção do módulo CCP1	CCP1IF	CCP1IE	CCP1IP
Estouro do timer 2	TMR2IF	TMR2IE	TMR2IP
Estouro do timer 1	TMR1IF	TMR1IE	TMR1IP
	PIR2	PIE2	IPR2
Falha no oscilador	OSCFIF	OSCFIE	OSCFIP
Alteração no estado do comparador	CMIF	CMIE	CMIP
Término de escrita na EEPROM	EEIF	EEIE	EEIP
Colisão no barramento SPI	BCLIF	BCLIE	BCLIP
Detecção de baixa/alta tensão	HLVDIF	HLVDIE	HLVDIP
Estouro do timer 3	TMR3IF	TMR3IE	TMR3IP
Interrupção do módulo CCP2	CCP2IF	CCP2IE	CCP2IP

Tabela 4.51: Controle das interrupções

4.6.2 Interrupções no C18

O compilador C18 permite o trabalho com interrupções através da diretiva `#pragma`. Através dessa diretiva é possível informar ao compilador que uma determinada função é uma RTI (Rotina de Tratamento de Interrupção) e associá-la a interrupção de alta ou baixa prioridade.

Uma rotina de tratamento de interrupção não pode ter argumentos nem retornar valores. Para realizar a troca de valores entre as RTI e o programa principal (ou mesmo entre as RTI's de alta e baixa prioridade) deve-se declarar uma variável global com o qualificador `volatile`.

A diretiva `#pragma` só informa qual função é uma RTI. O processo de identificar da interrupção e zerar o flag relativo a cada uma deve ser implementado por código. Exemplo abaixo apresenta a forma como a diretiva deve ser usada.

Exemplo:

```
#include<P18F4550.h>
#include<timers.h>
// *** Protótipos
void RTITimer1(void);
void RTITimer0(void);
// *** Configuração das RTI
#pragma code low_vector=0x18
void interrupt_at_low_vector(void) {
    .asm GOTO RTITimer0 _endasm
}
#pragma code
#pragma code high_vector=0x08
void interrupt_at_high_vector(void)
{
    .asm GOTO RTITimer1 _endasm
}
#pragma code
#pragma interruptlow RTITimer0
#pragma interrupt RTITimer1
// *** Funções
void RTITimer0 (void)
{
    INTCONbits.TMR0IF = 0;
    LATB = LATB^0x01; // inverte o bit 0
}
void RTITimer1 (void)
{
    PIR1bits.TMR1IF = 0;
    LATB = LATB^0x02; // inverte o bit 1
}
void main (void)
{
    TRISB = 0;
    PORTB = 0;
    // Configura os timers
    OpenTimer0 (TIMER_INT_ON & T0_SOURCE_INT & T0_16BIT);
    OpenTimer1 (TIMER_INT_ON & T1_SOURCE_INT);
    //Configura a interrupção
    RCONbits.IPEN = 1; // habilita prioridade
    INTCON2bits.TMR0IP = 0; // interrupção do timer 0 é baixa
```

```

IPR1bits.TMR1IP = 1; // interrupção do timer 1 é alta
INTCONbits.GIEH = 1; // habilita interrupções de alta prioridade
INTCONbits.GIEL = 1; // habilita interrupções de baixa prioridade
    
```

```

while (1); // fica em loop sem fazer nada
}
    
```

4.7 Funções úteis

A seguir são apresentadas algumas funções (padronizadas do C ou específicas do C18) que podem ser úteis em diversas situações.

A tabela a seguir apresenta algumas funções que convertem valores numéricos em string's e vice-versa. Essas funções estão presentes na biblioteca **stdlib.h**.

Funções	Descrição
atob	Converte em um valor de 8 bits sinalizado
atof	Converte em um valor fracionário
atoi	Converte em um valor de 16 bits sinalizado
atol	Converte em um valor inteiro longo
btoa	Converte um valor de 8 bits sinalizado em uma string
itoa	Converte um valor de 16 bits sinalizado em uma string
ltoa	Converte um valor inteiro longo sinalizado em uma string

Tabela 4.52: Funções de conversão de tipo

Há funções que permitem identificar o evento que ocasionou último reset. Essas funções pertencem a biblioteca **reset.h**.

Funções	Descrição
isBOR	Retorna 1 se o reset foi causado por BOR
isLVD	Retorna 1 se o reset foi causado por LVD
isMCLR	Retorna 1 se o reset foi causado através do pino MCLR
isPOR	Retorna 1 quando o sistema acaba de ser ligado
isWDTTO	Retorna 1 se ocorreu estouro do WatchDog
isWDTWU	Retorna 1 se saiu do modo SLEEP por estouro do WatchDog
isWU	Retorna 1 se saiu do modo SLEEP por estouro do MCLR ou interrupção.
StatusReset	Seta os pinos POR e BOR. Isto deve ser feito após um POR

Tabela 4.53: Funções de Reset

No portal B do PIC18 existem algumas funcionalidades especiais que podem ser configuradas fazendo uso das funções apresentadas na tabela 2.8. Estas funções pertencem a biblioteca **portb.h**.

Funções	Descrição
ClosePORTB	Desativa a interrupção por mudança de estado e resistores de pull-up do portal B.
CloseRB1INT CloseRB2INT CloseRB3INT	Desabilita as interrupções INT0, INT1 e INT2.
DisablePullups	Desativa os resistores de pull-up internos do portal B
EnablePullups	Ativa os resistores de pull-up internos do portal B
OpenPORTB	Configura interrupções e pull-up do portal B.
OpenRB1INT OpenRB2INT OpenRB3INT	Habilita as interrupções INT0, INT1 e INT2.

Tabela 4.54: Funções do portal B

Há ainda algumas macros que podem ser usadas em C para a execução direta de instruções do Assembly do PIC18. Essas funções estão definidas na biblioteca de cabeçalho de cada microcontrolador. Essas macros são apresentadas na tabela a seguir:

Funções	Ação
ClosePORTB	Desativa a interrupção por mudança de estado e resistores de pull-up do portal B.
CloseRB1INT CloseRB2INT CloseRB3INT	Desabilita as interrupções INT0, INT1 e INT2.
DisablePullups	Desativa os resistores de pull-up internos do portal B
EnablePullups	Ativa os resistores de pull-up internos do portal B
OpenPORTB	Configura interrupções e pull-up do portal B.
OpenRB1INT OpenRB2INT OpenRB3INT	Habilita as interrupções INT0, INT1 e INT2.

Tabela 4.55: Funções do portal B

Há ainda algumas macros que podem ser usadas em C para a execução direta de instruções do Assembly do PIC18. Essas funções estão definidas na biblioteca de cabeçalho de cada microcontrolador. Essas macros são apresentadas na tabela a seguir:

Funções	Ação
Nop()	Executa a instrução NOP
ClrWdt()	Executa a instrução CLRWD - zera o timer do WatchDog
Sleep()	Executa a instrução SLEEP
Reset()	Executa a instrução RESET - causa o reset do microcontrolador
Rlcf(var, dest, access)	Rotaciona var a esquerda através do bit C
Rlncf(var, dest, access)	Rotaciona var a esquerda (sem passar pelo bit C)
Rrcf(var, dest, access)	Rotaciona var a direita através do bit C
Rrnccf(var, dest, access)	Rotaciona var a direita (sem passar pelo bit C)
Swapf(var, dest, access)	Executa a instrução SWAPF

Tabela 4.56: Macros

4.7.1 Operações matemáticas

Um dos grandes benefícios de se usar uma linguagem de programação como C em lugar do Assembly nativo do microcontrolador é a facilidade em se realizar operações matemáticas. Além das operações realizadas com o uso dos operadores já apresentados, existem funções que implementam operações mais complexas. Essas funções estão presentes na biblioteca padrão “**math.h**”.

Funções	Descrição
acos	Cosseno inverso
asin	Seno inverso
atan	Tangente inversa
atan2	Tangente inversa de uma razão
ceil	Retorna o inteiro imediatamente superior
cos	Cosseno
cosh	Cosseno hiperbólico
exp	Exponencial ex
fabs	Valor absoluto de um número fracionário
floor	Retorna o inteiro imediatamente inferior
fmod	Resto da divisão de números fracionários
frexp	Decompõe um fracionário em fração e exponencial
ieeetomchp	Converte um valor fracionário no formato IEEE-754 para o formato Microchip.
ldexp	Multiplica um valor por 2 ^x
log	Calcula o logaritmo na base natural
log10	Calcula o logaritmo na base 10
mchptoeee	Converte um valor fracionário no formato Microchip para o formato IEEE-754.
modf	Módulo de um fracionário
pow	Exponencial xy
sin	Seno
sinh	Seno hiperbólico
sqrt	Raiz quadrada
tan	Tangente
tanh	Tangente hiperbólica

Tabela 4.57: Macros

4.8 Técnicas de Otimização de Código

Um dos principais aspectos da programação em alto nível é a otimização e eficiência do código gerado pelo compilador. Quando programarmos dispositivos tão limitados como os microcontroladores, a otimização pode ser fundamental para atender a requisitos de velocidade e tamanho de código. Estes fatores são determinantes neste tipo de projeto, pois um software tem que ser eficiente.

Por isso, vamos ver a seguir algumas dicas que permitem aumentar a velocidade de execução do código e evitar alguns problemas durante a confecção dos programas.

4.8.1 Rotinas Matemáticas

Devido às limitadas capacidades matemáticas da ULA dos PICs, a realização de operações aritméticas diferentes da soma ou subtração de oito bits pode demandar um grande tempo de execução, além do espaço na memória de programa. Por isso, devemos evitar a realização de operações matemáticas complexas e especialmente utilizando tipos de dados float.

Quando tais operações forem inevitáveis, aqui estão algumas sugestões para acelerar a realização de cálculos matemáticos com PIC's:

- Evite o uso de variáveis muito grandes como de 32 e 16 bits;
- Utilize, sempre que possível, divisões e multiplicações inteiras e se possível, utilizando múltiplos de potências de dois (já que essas operações podem ser facilmente traduzidas em rotações a direita ou esquerda, conforme o caso);
- Evite o uso de valores fracionários; prefira o uso de valores inteiros. Assim, para valores entre 0,00 e 5,00, poderíamos utilizar uma variável de 16 bits com valores entre 0 e 500.

Contudo, alguns cuidados devem ser tomados para aumentar a eficiência do código gerado e evitar problemas de difícil solução.

Em primeiro lugar, a escolha do tipo de dado das variáveis deve ser bem criteriosa. O código mais enxuto é gerado para operações com variáveis com mesmo comprimento do barramento de dados do microcontrolador, ou seja, do tipo char. Quando variáveis de tamanhos diferentes são usados uma quantidade significativa de instruções é necessária para implementar mesmo operações simples como soma e subtração. Ou seja, de uma forma mais geral, quantos mais bytes uma variável tem, maior e mais lento é o código gerado para tratá-la.

O uso de variáveis do tipo ponto flutuante deve ser evitado ao máximo. Operações com esse tipo de dado consomem muita memória e demoram muito tempo para serem executadas. Uma boa técnica para se evitar o uso de ponto flutuante é trabalhar com dados em escala, convertendo-os para ponto flutuante somente quando realmente necessário. Por exemplo, um programa que faz uma medida de tensão que expressa um valor entre 0,000 e 5,000 volts, compara esse valor com outros e o mostra em um display ocupa muita memória por processar dados do tipo ponto flutuante. O mesmo programa pode ocupar menos memória se a faixa de valores for de 0 a 5000, ou seja, "pensar" os valores todos multiplicados por 1000. Dessa maneira podem ser usados dados do tipo int. Na apresentação dos valores em um LCD basta inserir uma vírgula no local adequado.

Outro cuidado a ser tomado é com o tipo dos dados envolvidos em uma operação. Por exemplo, uma expressão onde dois inteiros são somados e atribuídos a um caractere gerará um resultado errado.

Por fim, deve-se estar atento para que valores intermediários de uma expressão não extrapolam a faixa de valores dos operandos da expressão. Por exemplo, o código a seguir aparentemente não contém erros.

Exemplo:

```
void main(void) /* essa função está ERRADA! */  
{
```

```
int x,y,z;  
y = 15;  
z = 500;  
x = y * 20000 / z; // <== o erro está aqui  
}
```

Se o cálculo for feito com uma calculadora teremos como resultado 600. Porém o cálculo realizado pelo microcontrolador resulta em -55! Vejamos porque isso ocorre.

O compilador implementa rotinas em Assembly para realizar a operação, já que todos os operandos envolvidos são inteiros. Mas a operação é feita pela ordem com que os dados aparecem na expressão. Na verdade a operação é realizada em partes, com o uso de variáveis internas de rascunho. O trecho abaixo ilustra o processo, sendo "rasc" é uma variável interna de rascunho, do tipo inteiro.

Exemplo:

```
rasc = y * 20000  
x = rasc / z
```

Ora, para y valendo 15 o valor armazenado em "rasc" é 300.000, grande demais para ser armazenado em um inteiro. Internamente durante a execução do cálculo o valor é truncado, ou seja, partes dele são desprezadas para caber em 16 bits. O valor de x é calculado com base em um valor incorreto e toda a operação fica prejudicada.

Problemas como esse tornam-se difíceis de serem depurados por para alguns valores a operação da certo. Por exemplo, se valesse 3 o resultado em x estaria correto. Uma forma de corrigir o problema é usar parênteses para indicar a precedência das operações. Serão realizadas primeiras as operações dentro dos parênteses mais esternos. Para o código a seguir:

Exemplo:

```
void main(void) /* essa função está CORRETA! */  
{  
    int x,y,z;  
  
    y = 15;  
    z = 500;  
    x = y * (20000 / z);  
}
```

O resultado em x será correto, pois a sequência de cálculos será:

```
rasc = 20000/z  
x = y * rasc
```

4.8.2 Usando assembly no C.

Por mais eficiente que sejam uma linguagem de programação e um compilador, dificilmente eles geram um código tão eficiente quanto um bom programador assembly. Em aplicações críticas, nas quais cada microsegundo conta, podemos utilizar código assembly dentro do programa em C, de forma a acelerar e/ou aperfeiçoar a sua execução.

De maneira geral, podemos dizer que os principais alvos deste tipo de otimização são: Rotinas de tratamento de interrupção (nos casos da ocorrência de um elevado número de interrupções num curto espaço de tempo), rotinas de manipulação de dados como conversões binário/decimal/hexadecimal, dentre várias outras.

A linguagem assembly pode ser inserida no código C em qualquer ponto, utilizando as diretivas `asm` e `_endasm`. As variáveis C conservam seus nomes na utilização da linguagem assembly.

4.8.3 Uso de variáveis locais e globais

Quando estamos programando para microcontroladores, é necessário dentro de um programa em C, criar várias funções que serão chamadas pelo `main()` de forma organizada, fazendo a execução do programa como um todo. Conforme visto anteriormente, quase todos os valores que podem ser manipulados por uma função é definido através de variáveis.

Estas variáveis podem ter inúmeras formas de atribuição, contudo, aqui estamos falando especificamente das variáveis locais e globais. As variáveis locais são locais de memória que armazenam informações que são manipuladas dentro de cada função, sendo que quando a função termina sua execução, esta área de memória pode ser reaproveitada pelo microcontrolador para alocação de outra variável local. A variável global, de outra forma, é uma variável declarada fora da função `main()` que tem "duração" por todo o programa, inclusive dentro de todas as funções em execução. Assim, o endereço onde esta variável guarda o seu valor permanece em uso por todo o programa não podendo ser reaproveitada pelo microcontrolador.

O ponto importante que dentro do microcontrolador, esses dois tipos de variáveis ocupam áreas de memória diferentes. Ainda, cada área de memória em específico possui um tamanho reservado, sendo que para as variáveis globais esta área é bem menor que a área destinada as variáveis locais. Então prefira usar variáveis locais para ter melhor utilização da memória.

Outro ponto sobre variáveis que devemos observar é quando utilizamos variáveis globais e locais com o mesmo nome. Isto causa uma confusão no compilador que podem gerar problemas na execução do programa. Por exemplo, se for criada duas variáveis, uma local e outra global com o mesmo nome, o compilador vai dar preferência para a última definida. Como sempre a ultima a ser definida será a local, a variável global ficará inacessível e começarão a acontecer problemas quando tentar se utilizar a variável global, pois ela será identificada pelo sistema como local.

Capítulo 5

Aplicações

5.1 Display de 7 segmentos

Um dos modos mais comuns de apresentar informação para o usuário em um sistema eletrônico é através de displays de LEDs. Esse tipo de display nada mais é que um conjunto de LEDs numa disposição que permite formar diferentes caracteres conforme são acionados. Eles podem ser de dois tipos: anodo comum ou catodo comum e podem ser vistos na figura a seguir em (b) e (c). A figura a mostra a nomenclatura comumente utilizada para os segmentos que compõe os dígitos e o ponto.

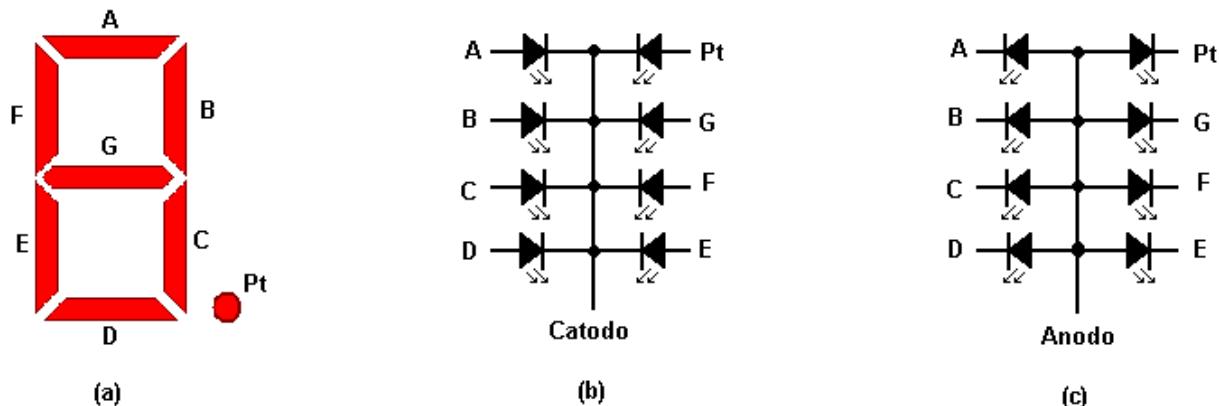


Figura 5.1: - Displays de LED: (a) disposição dos segmentos, (b) catodo comum e (c) anodo comum

Como pode ser observado na figura anterior, os displays de anodo comum tem seus segmentos acionados quando aplicamos '0', ficando o anodo (comum) na alimentação. Já os de catodo comum são acionado com '1' e o catodo (comum) é ligado a terra. No kit Pratic 628 contamos com displays de catodo comum.

Para escrevermos um determinado valor no display é necessário converter, ou decodificar, o valor originalmente em binário para combinações que acionem o display corretamente. Isso pode ser feito através de circuitos integrados feitos para esse fim, mas por motivo de economia de componentes é mais interessante que essa conversão seja feita por software e o acionamento seja feito diretamente pelo microcontrolador. A conversão deve ser feita obedecendo a tabela a seguir. Ela corresponde a kit NEO 201 onde o display é catodo comum e os pinos RD0, RD1, ..., RD6, RD7 estão ligados aos segmentos A, B, ..., G, DP (DP é o ponto decimal do display.)

Valor (Hexadecimal)	Catodo comum							Anodo comum						
	g	f	e	d	c	b	a	g	f	e	d	c	b	a
0	0	1	1	1	1	1	1	1	0	0	0	0	0	0
1	0	0	0	0	1	1	0	1	1	1	1	0	0	1
2	1	0	1	1	0	1	1	0	1	0	0	1	0	0
3	1	0	0	1	1	1	1	0	1	1	0	0	0	0
4	1	1	0	0	1	1	0	0	0	0	1	1	0	0
5	1	1	0	1	1	0	1	0	0	0	1	0	0	1
6	1	1	1	1	0	1	1	0	0	0	0	1	0	0
7	0	0	0	0	1	1	1	1	1	1	1	0	0	0
8	1	1	1	1	1	1	1	0	0	0	0	0	0	0
9	1	1	0	1	1	1	1	0	0	0	1	0	0	0
A	1	1	1	0	1	1	1	0	0	0	0	1	0	0
B	0	0	1	1	1	1	1	1	0	0	0	0	0	0
C	1	0	0	1	1	1	0	0	1	1	0	0	0	1
D	0	1	1	1	1	0	1	1	0	0	0	0	1	0
E	1	0	0	1	1	1	1	0	1	1	0	0	0	0
F	1	0	0	0	1	1	1	0	1	1	1	0	0	0

Tabela 5.1: Acionamento de display de sete segmentos.

Tabelas como as apresentadas são facilmente implementadas em C por vetores, uma vez que o índice do vetor representa a coluna de valor e os valores das posições correspondem aos acionamentos do LED. Abaixo é apresentada uma declaração de vetor de constantes que correspondem a tabela acima para catodo comum.

```
const char catodo[]={00111111, // 0
00000110, //1
01011011, //2
01001111, //3
01100110, //4
01101101, //5
01111101, //6
00000111, //7
01111111, //8
01100111}; //9
```

5.1.1 Displays multiplexados

Uma técnica utilizada para reduzir custos de hardware é a de multiplexar displays. Essa técnica se aproveita de uma característica da visão humana chamada persistência retiniana: a retina "mantém" uma imagem projetada presente por algum tempo depois que essa desaparece do campo de visão. Graças a essa propriedade que podemos enxergar imagens se movendo na TV, que na verdade são sequências de imagens paradas. Fazendo uso dessa propriedade também podemos acionar sequencialmente diversos displays e dar a impressão a quem olha que todos estão acesos ao mesmo tempo.

Basta acionar os diversos displays em sequência de forma que todos sejam acionados a uma taxa maior que 20 vezes por segundo. O circuito para fazer isso consiste em ligar os segmentos

respectivos de cada display em paralelo e acionar o terminal comum (anodo ou catodo) um por vez. Dessa maneira com 7 pinos mais um pino por display eh possível assinar diversos a displays.

5.1.2 Apresentando valores em display

Como vimos, um display pode apresentar valores de 0 a 9 (ou até F se quisermos apresentar valores em hexadecimal) e vários displays podem ser usados conjuntamente usando a técnicas da multiplexação. Porem, como apresentar valores com 123 ou 5000? Para isso é necessário decompor uma variável em unidades, dezenas, centenas, etc... e apresentar individualmente cada parte em um digito. Os trechos de código abaixo decompõem variáveis do tipo char e int (não sinalizados) para apresentação em displays. A partir delas é possível desenvolver rotinas para outros tipos de dados.

Exemplo:

```
// *** Decomposição de unsigned char
//unid, dez, cent são caracteres não sinalizado
cent = valor / 100;
valor = valor % 100;
dez = valor / 10;
unid = valor % 10;

// *** Decomposição de unsigned int
//unid, dez, cent, mil e dezmil são caracteres não sinalizado
dezmil = valor / 10000;
valor = valor % 10000;
mil = valor / 1000;
valor = valor % 1000;
cent = valor / 100;
valor = valor % 100;
dez = valor / 10;
unid = valor % 10;
```

5.2 Buzzer

O buzzer (buzina) nada mais é que um tipo de alto-falante simples, dimensionado para ter uma melhor resposta em um faixa de frequência específica. Através deles conseguimos gerar apitos (beeps) para sinalizações de diversos tipos.

Existem buzzers que são auto oscilantes, aos quais basta aplicar uma tensão de alimentação que geram um tom. Outros precisam que se envie um sinal chaveado para produzir seu som.

5.3 Teclado Matricial

Uma forma muito comum de usuário de um sistema microcontrolado passar informações ao sistema é através de teclas. Em muitas aplicações o número de teclas existentes pode ser bastante grande. Nesses casos, a leitura dessas teclas simplesmente conectando-as aos terminais do microcontrolador incorre na utilização de muitos terminais. Nessa aplicação é apresentado um sistema de varredura que permite fazer um uso otimizado dos terminais do microcontrolador de forma a reduzir o número de terminais utilizados.

Esse processo permite realizar a leitura de um número de teclas N utilizando menos de N entradas. Isso é importante quando existe uma grande quantidade de teclas a serem lidas e não se dispõe tanta entrada.

Para o caso da placa de teclado existem 16 teclas e seus estados são lidos com apenas 4 saídas e 4 entradas, num total de 8 terminais. A varredura funciona da seguinte maneira: o teclado é organizado de forma a ter 4 colunas e 4 linhas, conforme a figura abaixo. Cada linha é ligada a uma entrada, e cada coluna a uma saída. Existem resistores de pull-up nas entradas de forma que enquanto uma tecla não for pressionada, a entrada ficando em aberto tem nível lógico alto. Começando a varredura, é forçado "0" na coluna 0 (C0), "1" na coluna 1 (C1), 2 (C2) e 3 (C3) e são lidas as quatro linhas (L0, L1, L2 e L3). Em seguida coloca-se "1" em C0 e C2 e "0" em C1 e leem-se as linhas, e assim sucessivamente.. Nos momentos em que é feita a leitura das linhas podemos determinar se uma tecla está pressionada e qual é essa tecla da seguinte maneira: admitindo que a tecla está ligada a coluna que está em zero no momento a linha ligada a ela estará em também. As linhas que contém teclas que não estão pressionadas estarão em "1", devido aos resistores de pull-up. Seguindo esse raciocínio podemos gerar a seguinte tabela:

C3	C2	C1	C0	L3	L2	L1	L0	Tecla
1	1	1	0	1	1	1	0	1
1	1	0	1	1	1	1	0	2
1	0	1	1	1	1	1	0	3
0	1	1	1	1	1	1	0	A
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	B
1	1	1	0	1	0	1	1	7
1	1	0	1	1	0	1	1	8
1	0	1	1	1	0	1	1	9
0	1	1	1	1	0	1	1	C
1	1	1	0	0	1	1	1	*/F
1	1	0	1	0	1	1	1	0
1	0	1	1	0	1	1	1	#/E
0	1	1	1	0	1	1	1	D

Tabela 5.2: Dígitos/Linhas/Colunas de um teclado matricial.

O circuito do teclado é apresentado na figura abaixo.

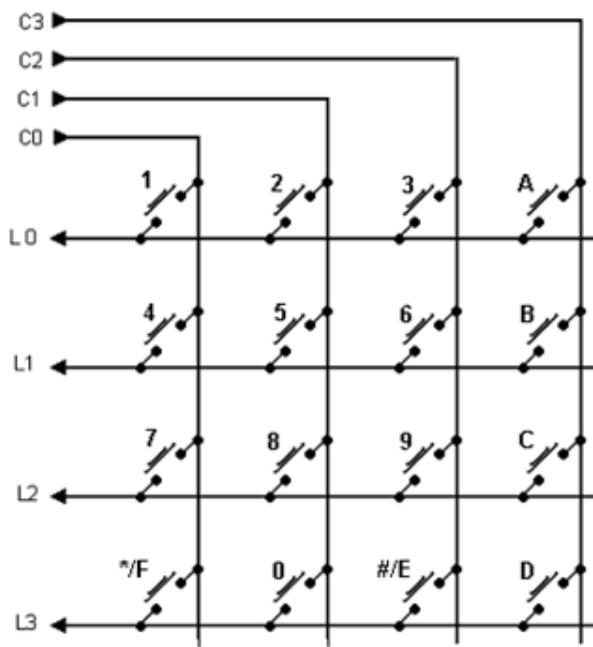


Figura 5.2: Esquema Simplificado do Teclado

No programa que utilizar leitura de teclado deve-se ativar os resistores de pull-up do portal
B. Caso contrário, o teclado não funcionará corretamente.

No Neo201 os teclado está ligado ao PIC seguindo a tabela baixo:

Teclado	PIC	I/O
C3	PORTB,0	Saída
C2	PORTB,1	Saída
C1	PORTB,2	Saída
C0	PORTB,3	Saída
L0	PORTB,4	Entrada
L1	PORTB,5	Entrada
L2	PORTB,6	Entrada
L3	PORTB,7	Entrada

Tabela 5.3: Ligação entre o PIC e o teclado matricial.

O fluxograma abaixo apresenta o processo de leitura para a primeira linha da função de leitura de teclado. A partir dela pode-se desenvolver a função como um todo.

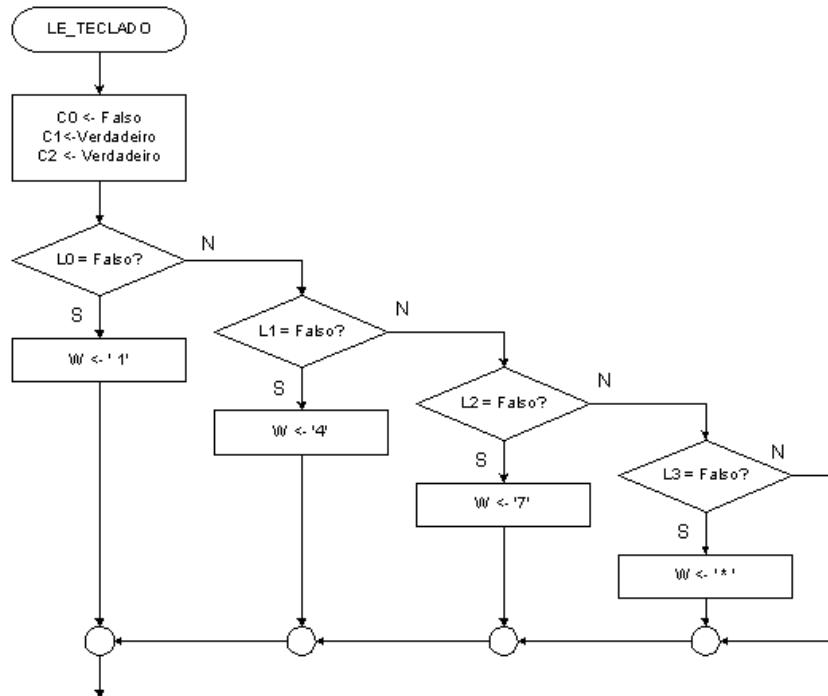


Figura 5.3: Fluxograma / Leitura do teclado.

5.3.1 Interrupção por mudança de estado na porta B

Como é muito comum a utilização de teclados por varredura, os microcontroladores PIC já tem uma funcionalidade pensada para permitir associar uma interrupção ao teclado. Essa funcionalidade é a interrupção por mudança de estado na porta B. O evento que sinaliza essa interrupção eh a mudança do estado dos pinos RB7, RB6, RB5 ou RB4. Observe que esses pinos estão ligados a linhas do teclado, portanto são entradas. Assim, qualquer tecla pressionada causará o pedido de interrupção. Ao se fazer o tratamento da interrupção faz-se a varredura para determinar qual a tecla lida. Mas atenção, para que esse procedimento funcione as colunas devem ser mantidas sempre em 0 quando fora do tratamento de interrupção.

5.4 Display de cristal líquido

Para que informações do sistema sejam passadas ao usuário pode-se utilizar uma serie de recursos, porém muitos deles não são nada maleáveis e alguns pouco amigáveis. Por exemplo, um modo bastante simples de se ter informações do sistema é através de um painel com LED's, cada um indicando uma determinada situação. Embora em muitas aplicações só isso seja suficiente, em muitas outras, principalmente quando a quantidade e variedade de informações são grande, esse método se torna insuficiente. Outro método também muito utilizado e que, porém, aceita uma grande variedade e quantidade de informações é o Display de Cristal Líquido (LCD - Liquid Cristal Display).

Existe uma grande variedade de LCDs no mercado. Existem displays gráficos e displays que aceitam somente caracteres, esses últimos chamados displays alfanuméricicos. Esses podem ter diferentes quantidades de linha e colunas. LCD's alfanuméricicos tem uma determinada "in-

teligência”, isto é, possuem circuitos que controlam o display propriamente dito, e fazer com que algo seja escrito no LCD é somente o trabalho de comunica-se com esses circuitos.

Para a comunicação com o display são necessários 8 bits como via de dados (podendo também ser configurado para trabalhar com 4 bits), um bit EN (Enable - Habilitação) e um bit RS (seleção entre dados e comandos). O display reconhece dois tipos de informação na via de dados: comandos e dados. Os comandos, que são reconhecidos quando RS = 0, são instruções para o display (limpar a tela, ir para a segunda linha, ir para a décima coluna, etc...); os dados são caracteres a serem escritos no display, e são indicados por RS = 1. As 4 bits da via de dados são ligados aos bits 4 a 7 do LCD.

Abaixo é apresentada uma tabela resumida de códigos hexadecimais de comandos do LCD.

Descrição do Comando	Modo	RS	R/W	Código do Comando (Hexadecimal)
Controle do display	Ativo (sem cursor)	0	0	0C
	Inativo	0	0	0A, 08
Limpeza do Display com retorno do cursor		0	0	01
Retorno do cursor à 1a linha e da mensagem à sua posição inicial		0	0	02
Controle do Cursor	Ativo (ligado, fixo)	0	0	0E
	Inativo	0	0	0C
	Alternado	0	0	0F
	Desloc. à esquerda	0	0	10
	Desloc. à direita	0	0	14
	Retorno	0	0	02
	Piscante	0	0	0D
Sentido de deslocamento do cursor na entrada de um novo caractere	Para esquerda	0	0	04
	Para direita	0	0	06
Deslocamento da mensagem com a entrada de um novo caractere	Para esquerda	0	0	07
	Para direita	0	0	05
Deslocamento da mensagem sem entrada de novos caracteres	Para esquerda	0	0	18
	Para direita	0	0	1C
Endereço da primeira posição (à esquerda)	1 ^a Linha	0	0	80
	2 ^a Linha	0	0	C0

Tabela 5.4: Códigos hexadecimais de comandos do LCD.

Os endereços de cada posição no display são dados pela tabela abaixo. Para que um caractere seja escrito em uma determinada posição, envia-se o valor dessa posição como comando e em seguida envia-se o caractere a ser escrito.

80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF

Tabela 5.5: Endereços de posição no display

A figura abaixo apresenta o circuito básico LCD.

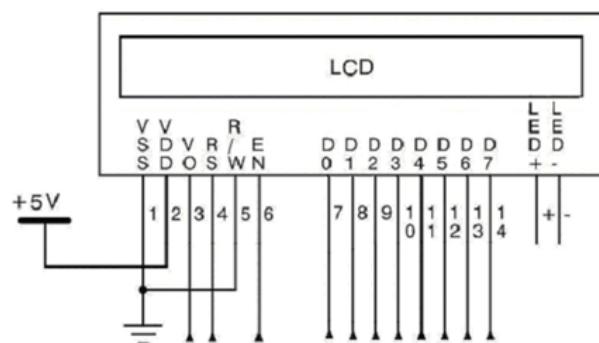


Figura 5.4: Esquema Básico do Display

A tabela seguinte apresenta a função de cada um dos pinos do display e em como eles estão ligados ao PIC.

Pino	Símbolo	Função	Ligaçāo com o PIC
1	Vss	GND	-
2	Vdd	+5V	-
3	Vo	Controle de contraste	-
4	RS	Seleção de modo	PORTE, 0
5	R/W	Leitura/Escrita	PORTE, 2
6	EN	Habilitação	PORTE, 1
7	D0		PORTD, 0
8	D1		PORTD, 1
9	D2		PORTD, 2
10	D3	Via de Dados	PORTD, 3
11	D4		PORTD, 4
12	D5		PORTD, 5
13	D6		PORTD, 6
14	D7		PORTD, 7

Tabela 5.6: Descrição dos pinos do display.

A figura abaixo apresenta uma tabela de caracteres do display.

Upper bit Lower bit	0000 (0)	0001 (1)	0010 (2)	0011 (3)	0100 (4)	0101 (5)	0110 (6)	0111 (7)	1000 (8)	1001 (9)	1010 (A)	1011 (B)	1100 (C)	1101 (D)	1110 (E)	1111 (F)
0000 (0)	CG RAM (1)				0	0P	1	p	0	0E	2	0	1	0	0P	0
0001 (1)		(2)		!	1	AQ	a	o	0	E	1	t	L	S	0	0
0010 (2)		(3)		W	2	B	B	0	0	2	1	0	0	0	0	0
0011 (3)		(4)		W	3	C	C	0	0	2	1	T	0	0	0	0
0100 (4)		(5)		0	4	D	T	d	0	0	1	4	0	0	0	0
0101 (5)		(6)		X	5	E	U	u	0	0	1	0	0	0	0	0
0110 (6)		(7)		0	6	F	U	f	0	0	0	0	0	0	0	0
0111 (7)		(8)		7	7	G	U	u	0	0	0	0	0	0	0	0
1000 (8)		(9)		0	8	H	X	x	0	0	0	0	0	0	0	0
1001 (9)		(2)		0	9	I	Y	1	0	0	2	0	0	0	0	0
1010 (A)		(3)		0	J	Z	0	2	0	0	0	0	0	0	0	0
1011 (B)		(4)		+	0	K	C	0	0	0	0	0	0	0	0	0
1100 (C)		(5)		0	0	L	P	0	0	0	0	0	0	0	0	0
1101 (D)		(6)		0	0	M	0	0	0	0	0	0	0	0	0	0
1110 (E)		(7)		0	0	N	0	0	0	0	0	0	0	0	0	0
1111 (F)		(8)		0	0	O	0	0	0	0	0	0	0	0	0	0

Fonte: DISPLAYTEC - www.displaytec.com.br

Figura 5.5: Caracteres do display.

5.4.1 A biblioteca XLCD

O compilador C18 vem com uma biblioteca genérica pra displays LCD chamada XLCD. Essa biblioteca possui as funções listadas abaixo, quer permitem inicializar o display e escrever simples caracteres ou strings. Note que existem diferentes tipos de função dependendo se a informação a ser escrita está na memória de dados (RAM) ou na memória de programa (FLASH).

Função	Protótipos	Descrição
BusyXLCD	unsigned char BusyXLCD(void);	Retorna 1 se o LCD estiver realizando alguma operação e 0 se estiver livre pra receber novos comandos.
OpenXLCD	void OpenXLCD(unsigned char lcdtype);	Inicializa o LCD.
putcXLCD	void OpenXLCD(unsigned char lcdtype);	Escreve um caractere no LCD
putsXLCD	void putsXLCD(char *buffer);	Escreve uma string (da memória RAM) no LCD
putrsXLCD	void putrsXLCD(const rom char *buffer);	Escreve uma string (da memória FLASH) no LCD
ReadAddrXLCD	unsigned char ReadAddrXLCD(void);	Lê o byte de endereço da memória do controlador de LCD.
ReadDataXLCD	char ReadDataXLCD(void);	Lê um byte do controlador de LCD.
SetCGRamAddr	void SetCGRamAddr(unsigned char addr);	Aponta o endereço do gerador de caracteres
SetDDRamAddr	void SetDDRamAddr(unsigned char addr);	Aponta para um endereço de dados do LCD.
WriteCmdXLCD	void WriteCmdXLCD(unsigned char cmd);	Escreve um comando no LCD
WriteDataXLCD	void WriteDataXLCD (char data);	Escreve um byte no LCD

Tabela 5.7: Comandos XLCD

Para usar a biblioteca XLCD é necessário fazer algumas alterações nas definições da biblioteca para o hardware que estiver sendo usado e recompilar os arquivos. Para facilitar, os arquivos que fazem uso dessa biblioteca no curso já sofreram essas alterações. Baseie-se neles para novos projetos.

5.5 Memórias EEPROM e FLASH

O PIC18F452 possui regiões de memória não-volátil(que não perdem seu conteúdo quando o componente perde alimentação): a memória FLASH de programa e a memória EEPROM de dados.

A memória EEPROM é destinada a guardar dados necessários a execução do programa que não se deseja perder com a falta de alimentação. Essa região de memória tem endereçamento próprio e se encontra separa da memória de programa (FLASH) e de dados (RAM). Para efeitos práticos, essa memória se comporta como um periférico, isto é, seu acesso é feito através de registros de funções especiais.

A memória de programa tem a função principal de armazenar o programa a ser executado

pelo Microcontrolador. É possível, no entanto, ler e escrever dados desta memória de forma muito semelhante ao que se faz com a memória EEPROM. Desta maneira, regiões da FLASH não usadas pelo programa podem ser usadas como um espaço adicional de memória não volátil para dados. Pode-se ainda atualizar tabelas de constantes usadas no programa. E ainda, uma das possibilidades mais interessantes, é criar atualizar o código do microcontrolador estando o mesmo em operação, usando alguma interface de comunicação (RS-232, RS-485, Ethernet, Modem, RF, etc...) permitindo assim o rápido upgrade de novas versões de firmware.

Tanto a EEPROM quanto a FLASH tem seu funcionamento controlado pelo registro EECON1. Antes de analisar as particularidades de cada memória, os bits deste registro são apresentados:

R/W - x	R/W - x	U - 0	R/W - 0	R/W - x	R/W - 0	R/W - 0	R/W - 0
EEPGD	CFG8	-	FREE	WRERR	WREN	WR	RD
Bit 7							

EECON1

Figura 5.6: EECON1

- **EEPGD** : Seleção de memória EEPROM ou FLASH:

- 1 = Acesso a memória FLAHS de programa;
- 0 = Acesso a memória EEPROM de dados;

- **CFG8** : Acesso aos bits de configuração:

- 1 = Acesso aos bits de configuração;
- 0 = Acesso a memória EEPROM ou FLASH (conforme EEPGD);

- **FREE** : Habilitação do apagamento de coluna (quadro) da FLASH:

- 1 = Apaga a coluna (quadro) da memória de programa apontada por TBLPTR no próximo comando de leitura (o bit é zerado quando a operação terminar);
- 0 = Executa uma operação de escrita simples.;

- **WRERR** : Indicador de erro no processo de escrita da FLASH ou EEPROM:

- 1 = Uma operação de leitura foi interrompida antes do fim (por qualquer reset);
- 0 = Operação de escrita concluída com sucesso;

- **WREN**: Habilitação de escrita na FLASH/EEPROM:

- 1 = Permite escrita;
- 0 = Impede escrita;

- **WR**: Controle de escrita:

- 1 = Inicia uma escrita de EEPROM ou FLASH ou o apagamento de FLASH. Esses processos são temporizados internamente e o bit é zerado automaticamente quando o processo termina. Não é possível zerar esse bit por software.;
- Ciclo de escrita de EEPROM ou FLASH ou apagamento de FLASH completo.;

- **RD:** Controle de leitura:

- 1 = Inicia o processo de leitura da EEPROM. Esse bit pode apenas ser setado em software. Se EEPGD = 1 (acesso a FLASH) esse pino não pode ser setado;
- 0 = Não inicia o processo de leitura.

5.5.1 Memória EEPROM

Os PIC18F452 e 18F4550 possuem 256 bytes de memória EEPROM para dados. A EEPROM serve para armazenar dados que podem ser alterados com a execução do programa, mas que não se pretende perder caso a tensão de alimentação seja retirada. Essa memória é mapeada separadamente das memórias RAM e FLASH, tendo endereços de 00h a FFh. A forma de acesso a memória EEPROM do PIC18 é muito semelhante a da linha PIC16. A memória EEPROM pode ser acessada byte a byte, tanto para leitura como para escrita. Na verdade, quando uma operação de escrita é realizada, automaticamente é feito um ciclo de apagamento e escrita do endereço em questão.

A memória EEPROM opera para leitura ou escrita em toda a faixa de tensão. É garantido um tempo mínimo de retenção de dados de 40 anos. A memória suporta no mínimo 100.000 ciclos de escrita, sendo o valor típico para condições normais de operação de 1.000.000 de ciclos. Cada operação de escrita demora 4ms para ser executada.

O acesso a memória EEPROM é feita de forma indireta, através de registros de controle e passagem de parâmetros. Por isso, podemos considerar que seu comportamento é mais semelhante ao de um periférico do que as demais memórias disponíveis no microcontrolador. Os registros de controle são EECON1 (descrito anteriormente) e EECON2 (que na verdade não é um registro real, mas participa na sequência de segurança de gravação, como será visto adiante). O registro EEADR indica o endereço acessado. Já o registro EEDATA permite manipular os dados, isto é, ele contém o dado a ser gravado em uma operação de escrita e o dado lido após uma operação de leitura. A figura abaixo exemplifica esse processo.

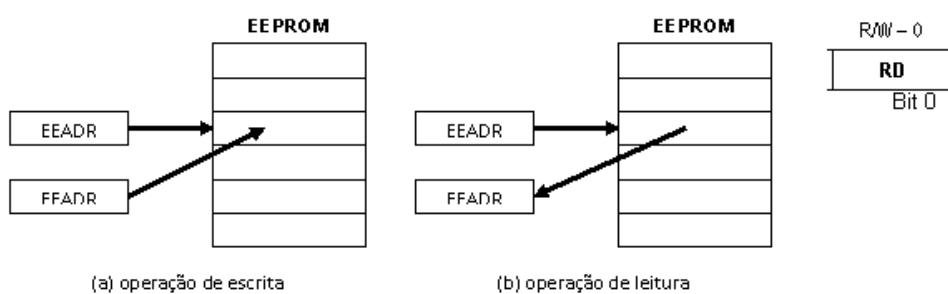


Figura 5.7: Processo de leitura e escrita da EEPROM

A memória EEPROM possui dois níveis de segurança, controlados pelos bits de configuração CPD e WRTD. CPD bloqueia leituras e escritas externas da EEPROM, liberando-as somente se o microcontrolador for totalmente apagado. Já WRTD bloqueia escritas internas e externas. Em todos os casos as operações de leitura interna são permitidas.

Interrupção

O bit EEIF é setado toda vez que uma operação de escrita é terminada. Isso permite iniciar o processo de gravação da EEPROM e voltar à execução normal do programa, já que o tempo de gravação de 4 ms pode ser crítico dependendo da aplicação. Contudo, é recomendado que se desabilite as demais interrupções enquanto é feita uma escrita da EEPROM.

Funções

O acesso à memória é feito seguindo uma sequência obrigatória de comandos. O manual do microcontrolador apresenta rotinas Assembly para esse fim. Contudo, o compilador C18 não provê rotinas para acesso a EEPROM. Segue abaixo os códigos fonte de rotinas que realizam leitura e escrita da EEPROM interna do PIC.

Exemplo:

```
unsigned char ReadEEPROM (unsigned char address)
//*****
// ROTINA PARA LEITURA DA EEPROM
// ENTRADA: address -> endereço de leitura da EEPROM
// SAÍDA: Retorno -> Retorna o valor contido na EEPROM
//*****
{
EEADR = address; // EEADR recebe o endereço
EECON1bits.CFGS = 0; // seleciona acesso a memórias
EECON1bits.EEPGD = 0; // seleciona acesso a EEPROM
EECON1bits.RD = 1; // inicia o processo de escrita
return(EEDATA); // retorna o dado lido da EEPROM
}
void
WriteEEPROM (unsigned char address, unsigned char data)
//*****
// rotina p escrita na eeprom
// entrada: address -> indica posição da memória eeprom
//           data -> dados que serão escritos
// saída:   nulo
//*****
{
EEADR = address; // EEADR recebe o endereço
EEDATA = data; // EEDATA recebe o dado
EECON1bits.CFGS = 0; // seleciona memórias
EECON1bits.EEPGD = 0; // seleciona memória EEPROM
EECON1bits.WREN = 1; // desliga proteção de escrita
INTCONbits.GIE = 0; // desabilita interrupções
```

```
EECON2 = 0x55; // sequência
EECON2 = 0xAA; // obrigatória
EECON1bits.WR = 1; // inicia processo de escrita
while(!PIR2bits.EEIF); // aguarda completar a escrita
INTCONbits.GIE = 1; // reativa as interrupções
PIR2bits.EEIF = 0; // zera o flag de escrita
EECON1bits.WREN = 0; // liga a proteção de escrita
}
```

Em diversas situações é desejável incluir no código fonte os dados a serem gravados na EEPROM no momento em que o microcontrolador é gravado. O processo para se fazer isso é apresentado no trecho de código abaixo. O valor 0xF00000 indica para o gravador que esse dado deve ser armazena do na EEPROM.

Exemplo:

```
// Esta é a diretiva que inicia a região de EEPROM
#pragma romdata dataEEPROM = 0xF00000

// Várias formas de como se gravar um dado
rom unsigned char FirstByte = 0x55;
rom unsigned char SecondByte;
rom unsigned char FirstArray[] = {0x00, 0x01, 0x02};
rom unsigned char SecondArray[3];
```

5.5.2 Memória Flash

Ao contrário da memória EEPROM, a memória FLASH do PIC18 se comporta de forma bastante diferente da memória do PIC16. Inclusive, existem algumas diferenças entre a forma de operação do PIC18F452 e o PIC18F4550.

A memória EEPROM pode ser lida byte a byte, contudo ela só pode ser escrita em blocos de memória (8 bytes para o PIC18F452 e 64 bytes para o PIC28F4550). E ainda, é necessário apagar um bloco de memória inteiro (de 64 bytes) para se poder escrever, mesmo que em um único endereço. Em contra partida ao aumento da complexidade do processo temos um ganho na velocidade de gravação, pois vários endereços são gravados a cada comando de gravação. Esse processo demora tipicamente 2 ms, além de mais 2 ms do processo de apagamento, isto é, o mesmo tempo de se gravar um único byte na EEPROM. Durante o tempo de gravação o programa não é executado.

O acesso de leitura de EEPROM é feito de forma indireta. Existe um ponteiros que aponta qualquer posição da memória de programa chamado TBLPTR, que é composto por três registros: TBLPTRU: TBLPTRL: TBLPTRL, sendo que 22 bits são válidos para endereçamento. A se executar a instrução TBLRD de leitura de tabela (Table Read) o conteúdo do endereço apontado é transferido para o registro TABLAT. Nessa operação a memória é lida de 8 em 8 bits, enquanto no fetch de instruções a leitura é de 16 em 16 bits.

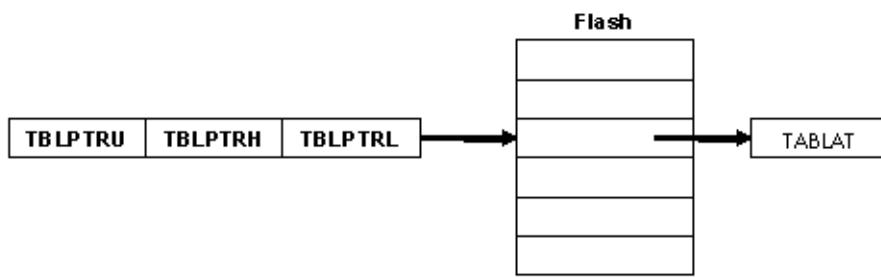


Figura 5.8: Processo de leitura da FLASH

O processo de escrita exige que antes o ”quadro” de 64 bytes seja apagado. Para isso o mesmo ponteiro TBLPTR é carregado para apontar o quadro. É como se a memória fosse dividida em quadros de 64 bytes e um endereço de 16 bits (os 16 bits válidos mais significativos de TBLPTR) indicassem o quadro, enquanto os 6 bits menos significativos são ignorados na operação. Em seguida é executado o processo de apagamento, fazendo o bit FREE de EECON1 igual a ’1’ e executando um comando de escrita.

Para gravação dos dados no PIC18F452 devem ser carregados 8 bytes através da instrução TBLWT de escrita de tabela (table write). Os dados não são gravados diretamente na memória FLASH, mas armazenados em um conjunto de registros de carga (holding registers), para posteriormente serem gravados de uma só vez. Nessa operação os 19 bits válidos mais significativos de TBLPTR apontam o bloco de 8 bytes a ser gravado enquanto os 3 bits menos significativos indicam o registro de carga que recebe os dados. Em seguida é realizada uma operação de gravação (com o bit FREE em ’0’). Um cuidado que se deve tomar é que TBLPTR deve apontar para o bloco de 8 bits no momento da gravação.

Para a gravação do PIC18F4550 o raciocínio é o mesmo, porém os blocos são de 64 bytes.

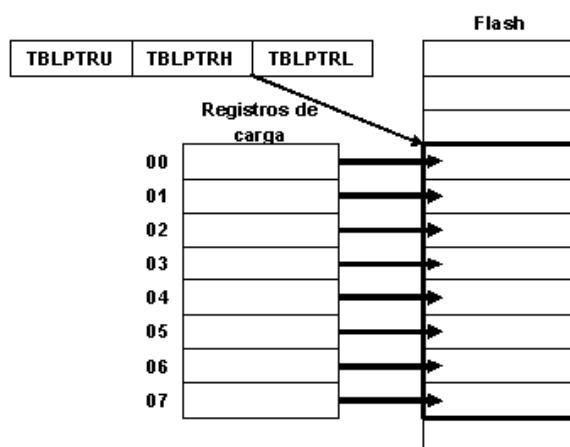


Figura 5.9: Processo de escrita na FLASH(PIC18F452)

Eventualmente pode-se desejar alterar um único endereço da FLASH, preservando os demais conteúdos do quadro de 64 bytes no qual se encontra. Nesta situação é necessário copiar todo o quadro para memória RAM, apagá-lo na FLASH, e re-gravar com o novo dado.

Existem três níveis de proteção de código da memória flash, configurados através dos bits de gravação:

- Code-Protect (bit CPn): impede a leitura e escrita externa.
- Write-Protect (bit WRTn): impede a escrita interna.
- External Block Table Read (bit EBTRn): permite apenas leituras internas e o código que executa a leitura estiver no mesmo setor de memória do dado lido.

Uma observação importante é que, apesar dos bits de configuração poderem ser alterados por instruções de escrita na FLASH (fazendo o bit CFGS igual a 1) os bits de Code-Protect só são zerados se houver uma instrução para apagar a memória inteira; esse comando só pode ser dado externamente por um gravador.

Interrupção

Do mesmo modo que para a EEPROM, o bit EEIF é setado toda a vez que o processo de gravação é finalizado. No entanto, ao contrário do que ocorre com EEPROM, o programa para de ser executado enquanto dados são gravados na FLASH, isto é, obrigatoriamente o tempo de gravação será um "tempo morto" para o programa. Continua existindo a recomendação de se desativar as interrupções durante a gravação, apesar de que, pelos motivos apresentados acima, se ocorresse uma interrupção esta não poderia ser tratada.

Capítulo 6

Periféricos

Uma das características que explica o sucesso dos microcontroladores é a presença de diversos periféricos. Esses componentes agregados ao core permitem realizar diversas tarefas, expandindo as capacidades do componente. Como trabalham de forma independente do processador e possuem interrupções, eles realmente trabalham em paralelo com o processador na execução das tarefas dos programas. O PIC18F4550 é particularmente rico em periféricos e possui um periférico USB, que é um dispositivo especial dentro da família de microcontroladores.

Nessa seção apresentaremos os principais periféricos do PIC18F4550. Além das descrições do funcionamento dos periféricos são apresentadas as bibliotecas de funções de periféricos que acompanham o compilador C18. Como o foco do curso é a linguagem C, não detalharemos o funcionamento dos bits de controle de periféricos, uma vez que isso é feito pelas funções das bibliotecas de cada periférico. Focaremos principalmente no funcionamento e configuração dos periféricos e no uso das funções em C. Os eventos causadores de interrupção são apresentados; os bits associados a cada interrupção são apresentados no capítulo **Interrupções**.

6.1 Como usar a ajuda das bibliotecas do C18

Os detalhes de uso das bibliotecas de função são descritas nos arquivos de ajuda do compilador, que se encontram em **C:\MCC18\DOC\PERIPH-LIB**. Esses arquivo trás a funcionamento das funções relativas a versão do compilador a que se refere. Portanto, podem haver alterações nas funções se usados versões mais novas do compilador. Deve-se estar atendo a isso e levar sempre os arquivos de ajuda como referência.

Para ter acesso a esses arquivos vá a até a pasta **C:\MCC18\DOC\PERIPH-LIB**. O aspecto dessa pasta é mostrado na figura a seguir.

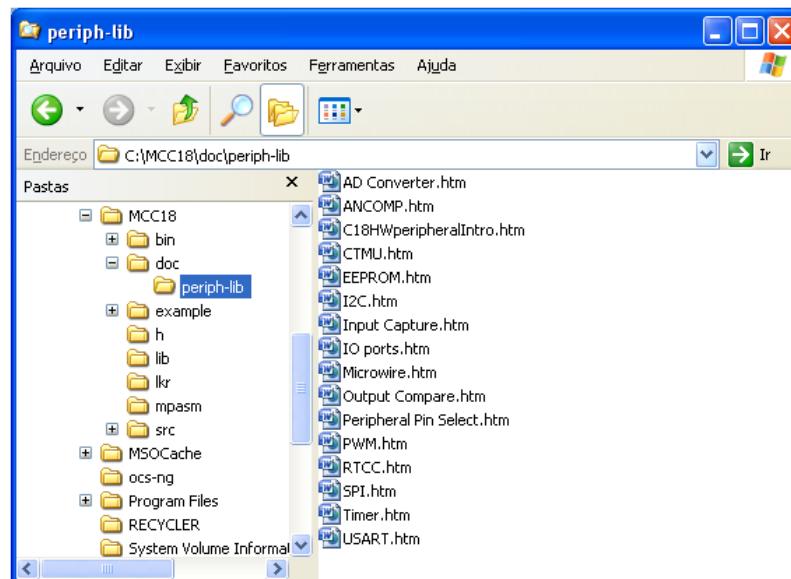


Figura 6.1: Aspecto da pasta PERIPH-LIB

Observe que existe um arquivo de ajuda para cada biblioteca de periférico, que pode ser acessado diretamente ou através do arquivo principal que é **C18HWeriferalIntro.htm**. Os arquivos de ajuda estão escritos em inglês, obviamente, sendo necessária um conhecimento mínimo dessa linguagem para os compreender, assim como é para entender o manual de qualquer componente eletrônico.

Para exemplificar o uso vamos explorar a biblioteca do conversor AD (AD Converter.HTM). Os demais arquivos sempre seguiram esse padrão. Temos a tela inicial abaixo, que apresenta um link para o conteúdo da ajuda.

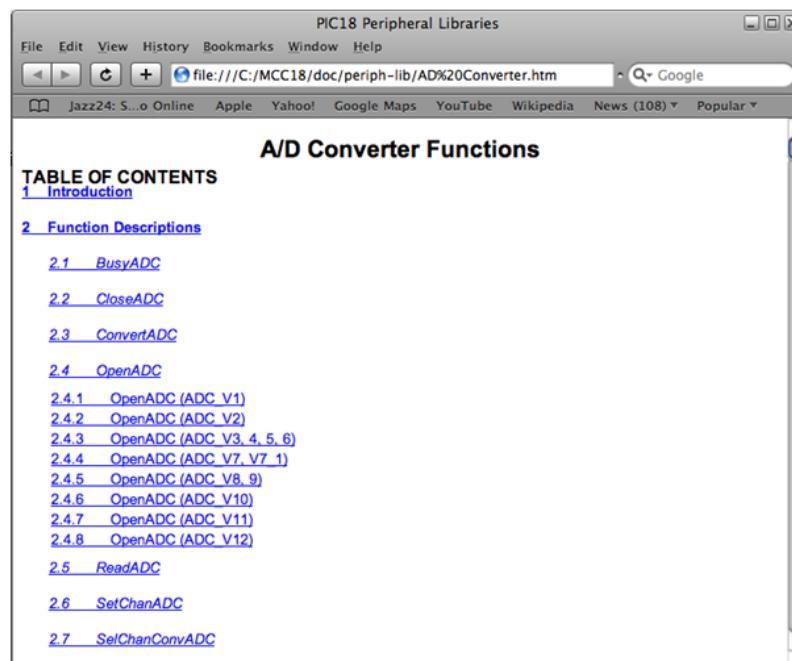


Figura 6.2: Conteúdo da ajuda.

Na introdução temos um resumo das funções da biblioteca.

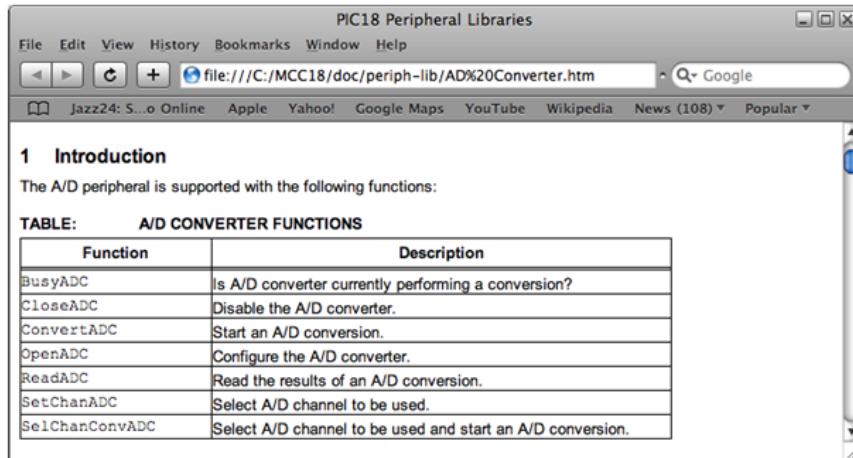


Figura 6.3: Resumo das bibliotecas.

Um pouco além é apresentada uma tabela onde é possível identificar qual a versão de AD do microcontrolador utilizado. Dependendo do periférico podem existir diferenças nas funções conforme o microcontrolador utilizado. Estará indicado para cada função a que versão de periférico se destina e a tabela deve ser sempre consultada para definir a versão do periférico a ser usado. Por exemplo, na figura abaixo podemos identificar que o PIC18F4550 possui um ADC de versão ADC_V5.

The screenshot shows a web browser window titled "PIC18 Peripheral Libraries". The URL in the address bar is "file:///C:/MCC18/doc/periph-lib". A search bar at the top right contains the text "18f4550". The page content includes a table titled "TABLE: VERSION vs. DEVICES". The table lists the following data:

Version name	Device number
ADC_V1	18C242, 18C252, 18C442, 18C452, 18F242, 18F252, 18F442, 18F452, 18F248, 18F258, 18F448, 18F458, 18F2439, 18F2539, 18F4439, 18F4539
ADC_V2	18C601, 18C801, 18C658, 18C858, 18F6620, 18F6720, 18F8620, 18F8720, 18F6520, 18F8520
ADC_V3	18F1220, 18F1320
ADC_V4	18F1230, 18F1330
ADC_V5	18F2220, 18F2320, 18F4220, 18F4320, 18F2420, 18F2520, 18F4420, 18F4520, 18F2423, 18F2523, 18F4423, 18F4523, 18F2455, 18F2550, 18F4455, 18F4550, 18F2410, 18F2510, 18F2515, 18F2610, 18F4410, 18F4510, 18F4515, 18F4610, 18F2525, 18F2620, 18F4525, 18F4620, 18F6310, 18F6410, 18F8310, 18F8410, 18F6390, 18F6490, 18F8390, 18F8490, 18F6527, 18F6622, 18F6627, 18F6722, 18F8527, 18F8622, 18F8627, 18F8722, 18F6585, 18F6680, 18F8585, 18F8680, 18F6525, 18F6621, 18F8525, 18F8621, 18F2450, 18F4450, 18F2480, 18F2580, 18F4480, 18F4580, 18F2585, 18F2680, 18F4585, 18F4680, 18F2682, 18F2685, 18F4682, 18F4685, 18F2221, 18F2321, 18F4221, 18F4321

Figura 6.4: Versão de AD do microcontrolador utilizado.

Explorando um pouco mais a função OpenADC veremos como é a configuração do ADC. Para cada periférico existirá uma função Open... que o inicializa através de uma série de parâmetros. Essa função possuirá um ou mais parâmetros de entrada, que serão uma operação E entre diversas constantes definidas, que representam as configurações possíveis do periférico. Em C o trabalho de configurar o periférico resume-se a definir esses parâmetros de configuração, ficando a cargo do compilador, através da função, de configurar bits e registros. Os diferentes parâmetros são agrupados através do operador & que, como sabemos, realiza a operação E bit-a-bit. A figura abaixo mostra como a função se apresenta na ajuda (note que essa função é a que se aplica a versão ADC_V5, portanto pode ser usada para o PIC18F4550).

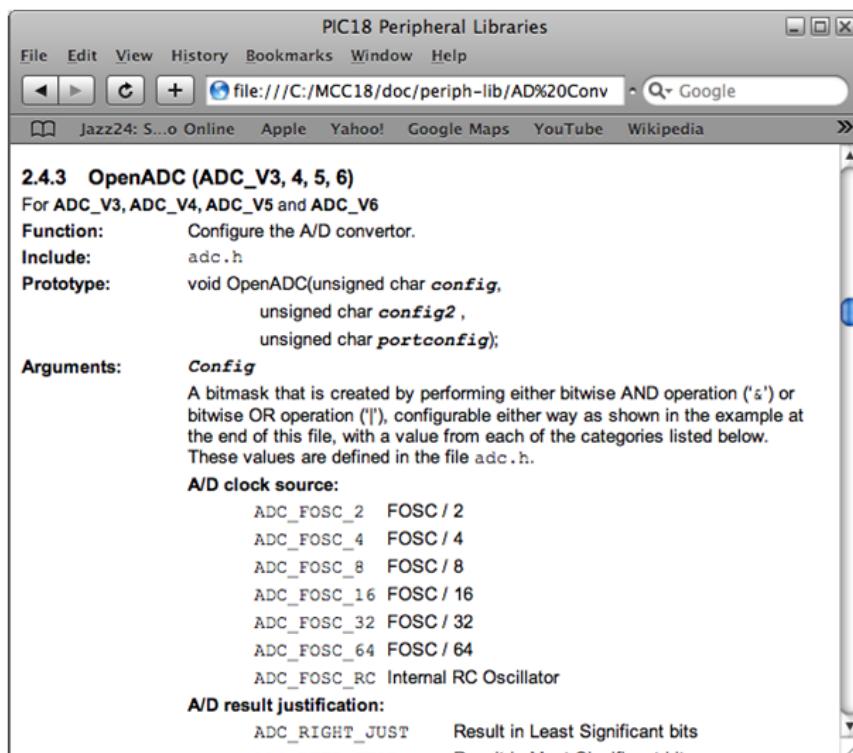


Figura 6.5: Apresentação da função ajuda.

Transcrevemos a seguir a ajuda dessa função na íntegra, para se perceber seus aspectos principais. Note que, ao final existe um exemplo do uso da função, muitas vezes útil.

```

2.4.3          OpenADC (ADC_V3, 4, 5, 6)
For ADC_V3, ADC_V4, ADC_V5 and ADC_V6
Function: Configure the A/D convertor.
Include: adc.h
Prototype: void OpenADC(unsigned char config,
                         unsigned char config2 ,
                         unsigned char portconfig);
Arguments: Config
A bitmask that is created by performing either bitwise AND operation
('&') or bitwise OR operation ('|'), configurable either way as shown
in the example at the end of this file, with a value from each of the
categories listed below. These values are defined in the file adc.h.

```

A/D clock source:

ADC_FOSC_2	FOSC / 2
ADC_FOSC_4	FOSC / 4
ADC_FOSC_8	FOSC / 8
ADC_FOSC_16	FOSC / 16
ADC_FOSC_32	FOSC / 32
ADC_FOSC_64	FOSC / 64
ADC_FOSC_RC	Internal RC Oscillator

A/D result justification:

ADC_RIGHT JUST	Result in Least Significant bits
ADC_LEFT JUST	Result in Most Significant bits

A/D acquisition time select:

ADC_0_TAD	0 Tad
ADC_2_TAD	2 Tad
ADC_4_TAD	4 Tad
ADC_6_TAD	6 Tad
ADC_8_TAD	8 Tad
ADC_12_TAD	12 Tad
ADC_16_TAD	16 Tad
ADC_20_TAD	20 Tad

config2

A bitmask that is created by performing either bitwise AND operation ('&') or bitwise OR operation ('|'), configurable either way as shown in the example at the end of this file, with a value from each of the categories listed below.

These values are defined in the file adc.h.

Channel:

For ADC_V3:

ADC_CHO	Channel 0
ADC_CH1	Channel 1
ADC_CH2	Channel 2
ADC_CH3	Channel 3
ADC_CH4	Channel 4
ADC_CH5	Channel 5
ADC_CH6	Channel 6

For ADC_V4:

ADC_CHO	Channel 0
ADC_CH1	Channel 1
ADC_CH2	Channel 2
ADC_CH3	Channel 3

For ADC_V5 and ADC_V6:

ADC_CHO	Channel 0
---------	-----------

ADC_CH1	Channel 1
ADC_CH2	Channel 2
ADC_CH3	Channel 3
ADC_CH4	Channel 4
ADC_CH5	Channel 5
ADC_CH6	Channel 6
ADC_CH7	Channel 7
ADC_CH8	Channel 8
ADC_CH9	Channel 9
ADC_CH10	Channel 10
ADC_CH11	Channel 11
ADC_CH12	Channel 12
ADC_CH13	Channel 13
ADC_CH14	Channel 14
ADC_CH15	Channel 15

A/D Interrupts:

ADC_INT_ON	Interrupts enabled
ADC_INT_OFF	Interrupts disabled

A/D Vref+ and Vref- configuration:

ADC_REF_VDD_VREFMINUS	VREF+ = VDD & VREF- = Ext.
ADC_REF_VREFPLUS_VREFMINUS	VREF+ = Ext. & VREF- = Ext.
ADC_REF_VREFPLUS_VSS	VREF+ = Ext. & VREF- = VSS
ADC_REF_VDD_VSS	VREF+ = VDD & VREF- = VSS

Portconfig

For ADC_V3:

The value of portconfig can be any value from 0 to 127, few are defined below

ADC_0ANA	All digital
ADC_1ANA	analog:AN0
ADC_2ANA	analog:AN0-AN1
ADC_3ANA	analog:AN0-AN2
ADC_4ANA	analog:AN0-AN3
ADC_5ANA	analog:AN0-AN4
ADC_6ANA	analog:AN0-AN5
ADC_7ANA	analog:AN0-AN6

For ADC_V4:

The value of portconfig can be any value from 0 to 15, few are defined below

ADC_0ANA	All digital
ADC_1ANA	analog:AN0
ADC_2ANA	analog:AN0-AN1
ADC_3ANA	analog:AN0-AN2
ADC_4ANA	analog:AN0-AN3

For ADC_V5 and ADC_V6:

ADC_OANA	All digital	
ADC_1ANA	analog:AN0	digital:AN1-AN15
ADC_2ANA	analog:AN0-AN1	digital:AN2-AN15
ADC_3ANA	analog:AN0-AN2	digital:AN3-AN15
ADC_4ANA	analog:AN0-AN3	digital:AN4-AN15
ADC_5ANA	analog:AN0-AN4	digital:AN5-AN15
ADC_6ANA	analog:AN0-AN5	digital:AN6-AN15
ADC_7ANA	analog:AN0-AN6	digital:AN7-AN15
ADC_8ANA	analog:AN0-AN7	digital:AN8-AN15
ADC_9ANA	analog:AN0-AN8	digital:AN9-AN15
ADC_10ANA	analog:AN0-AN9	digital:AN10-AN15
ADC_11ANA	analog:AN0-AN10	digital:AN11-AN15
ADC_12ANA	analog:AN0-AN11	digital:AN12-AN15
ADC_13ANA	analog:AN0-AN12	digital:AN13-AN15
ADC_14ANA	analog:AN0-AN13	digital:AN14-AN15
ADC_15ANA	All analog	

Remarks: This function resets the A/D-related registers to the POR state and

then configures the clock, result format, voltage reference, port and channel.

File Name: adcopen.c

Code Example: With AND mask:

```
OpenADC( ADC_FOSC_32      &
          ADC_RIGHT JUST   &
          ADC_12_TAD,
          ADC_CHO          &
          ADC_REF_VDD_VSS  &
          ADC_INT_OFF, 12 );
```

With OR mask:

```
OpenADC( ADC_FOSC_32      |
          ADC_RIGHT JUST   |
          ADC_12_TAD,
          ADC_CHO          |
          ADC_REF_VDD_VSS  |
          ADC_INT_OFF, 12 );
```

6.2 Contadores e Temporizadores

O PIC18F4550 possuem 4 timers que trabalham de forma bastante semelhante: Timer 0, Timer 1, Timer 2 e Timer 3.

6.2.1 Timer 0

O Timer 0 do PIC18 é um timer de 16 bits, mas pode operar em modo 8 bits para manter compatibilidade com o Timer 0 do PIC16.

O registro de controle do Timer 0 é T0CON.

A figura a seguir apresenta o diagrama em blocos do Timer 0 operando no modo 8 bits.

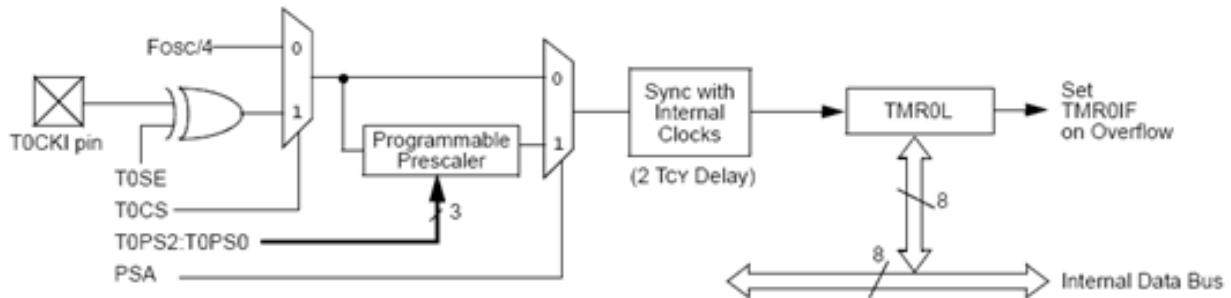


Figura 6.6: Diagrama do Timer 0 no modo 8 bits

Já a figura abaixo mostra o diagrama do Timer 0 operando no modo 16 bits.

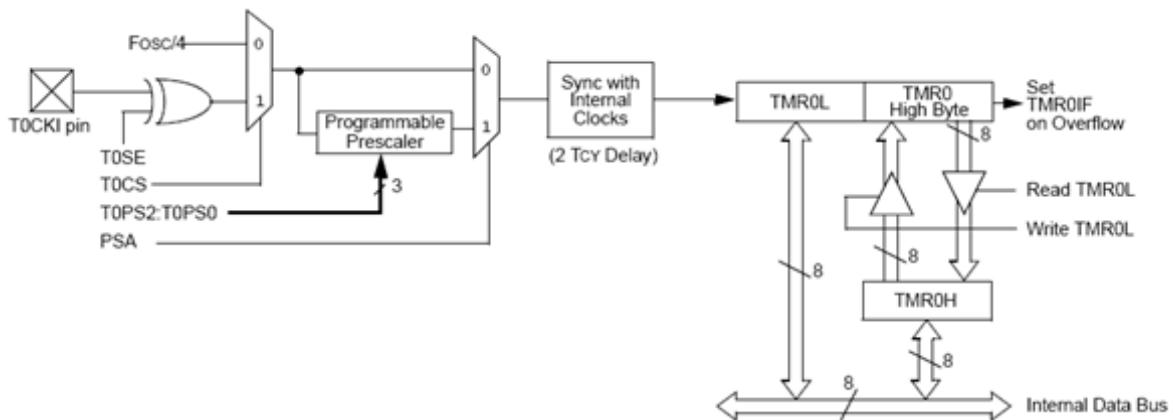


Figura 6.7: Diagrama do Timer 0 no modo 16 bits

As diferenças entre a operação no modo 8 bits e no modo 16 bits são que (1) o flag de interrupção seta no estouro de 8 bits e no estouro de 16 bits, dependendo do modo, e (2) no modo 16 bits existe um sistema para gravação e leitura simultânea dos dois registros.

O sinal de clock do timer 0 pode vir tanto do sinal de clock interno (FOSC/4) quanto do pino RA4/T0CKI. É possível configurar se o timer incrementará na transição de subida ou de descida.

A pré-escala do timer 0 permite multiplicar o período do sinal de entrada por 1, 2, 4, 8, 16, 32, 64, 128, e 256.

Interrupção

O evento que ocasiona a interrupção é o overflow do timer 0, que depende da configuração do timer, se para 8 ou 16 bits.

6.2.2 Timer 1

O timer 1 é um timer de 16 bits. Ele pode operar em dois modos: leitura e escrita em 8 bits (compatível com PIC16) e leitura e escrita em 16 bits. No modo leitura e escrita em 8 bits os dois registros do timer são alterados separadamente, conforme os valores são carregados. No modo de 16 bits existe um sistema que carrega o valor de/para um registro auxiliar quando é feita

uma operação de como byte menos significativo; dessa maneira as operações acontecem de forma simultânea nos dois registros.

A figura abaixo apresenta o diagrama em blocos do timer 1 para o modo de acesso em 16 bits.

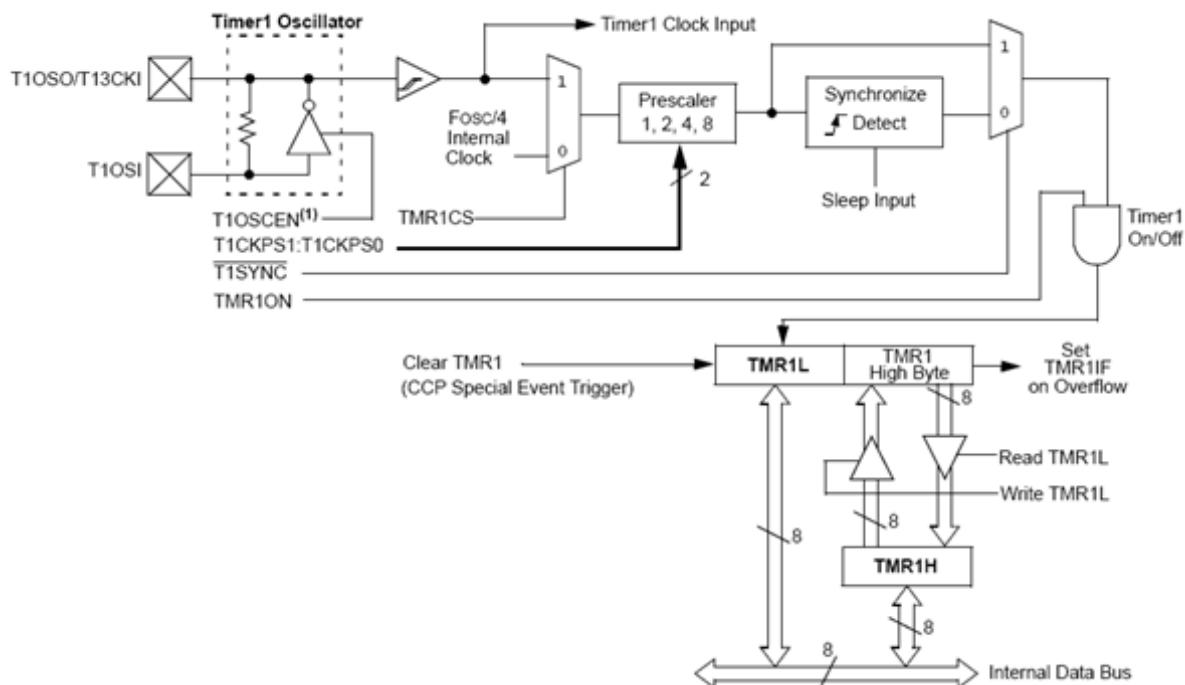


Figura 6.8: Diagrama do Timer 1

Podemos escolher entre três fontes de clock do timer 1: sinal externo, sinal de clock interno ($F_{OSC}/4$) e oscilador do Timer 1.

O oscilador do Timer 1 pode ser selecionado como uma fonte de clock para o processador. Para isso é necessário que o oscilador do timer 1 esteja habilitado. Tipicamente no oscilador do timer 1 é usado um cristal de 32.768 Hz; de cada terminal desse cristal deve ser ligado um capacitor de 27pF para GND. O timer 1 possui uma pré-escala que permite multiplicar o período do sinal de entrada por 1, 2, 4 e 8. É possível ainda sincronizar o sinal de clock, se proveniente de fonte externa, com os ciclos de instrução do microcontrolador. O timer 1 pode servir de base de tempo para o módulo CCP, dependendo da configuração deste.

Por fim, o timer pode ser ligado e desligado, isto é, pode-se suspender o sinal aplicado ao contador.

Interrupção

O evento associado a interrupção do timer 1 é o overflow da contagem de 16 bits.

6.2.3 Timer 2

O timer 2 é um temporizador de 8 bits. Ele apresenta algumas características que diferem dos timers vistos anteriormente. Na figura a seguir pode-se observar o diagrama em blocos desse timer.

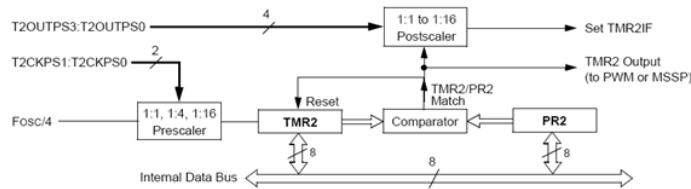


Figura 6.9: Diagrama em blocos do timer 2

Em primeiro lugar, o timer 2 só pode operar com fonte de clock interna, portanto apenas no modo timer. Ele possui uma pré-escala com fatores de multiplicação de 1, 4 e 16. O sinal de saída da pré-escala é aplicado ao contador propriamente dito, que é acessível através do registro TMR2. O valor desse contador é comparado constantemente com o valor do registro PR2. Quando os dois valores são iguais é gerado nível lógico alto no sinal de coincidência (TMR2 Output), que é usado como base de tempo para outros módulos, e o contador é zerado. O sinal de coincidência é aplicado a uma pós-escala que pode acumular de 1 a 16 ocorrências, ativando o flag de interrupção quando a contagem chegar ao valor estabelecido. Não há uma sinalização de overflow, que pode ser simulado fazendo PR2 igual a 0.

O valor de PR2 não é configurado por nenhuma função específica. Basta atribuir ao valor desejado à variável PR2, já definida no arquivo de cabeçalho.

Interrupção

O evento que dispara a interrupção do timer 2 é a contagem de n eventos de coincidência entre o timer 2 e PR2, sendo que n é configurável.

6.2.4 Timer 3

O timer 3 funciona da mesma maneira que o timer 1, conforme pode ser observado na figura a seguir. No modo contador o sinal de clock é aplicado ao mesmo pino do timer (RC0). Além disso, se o oscilador do timer 1 estiver ativo, também pode fornecer sinal de clock para o timer 3.

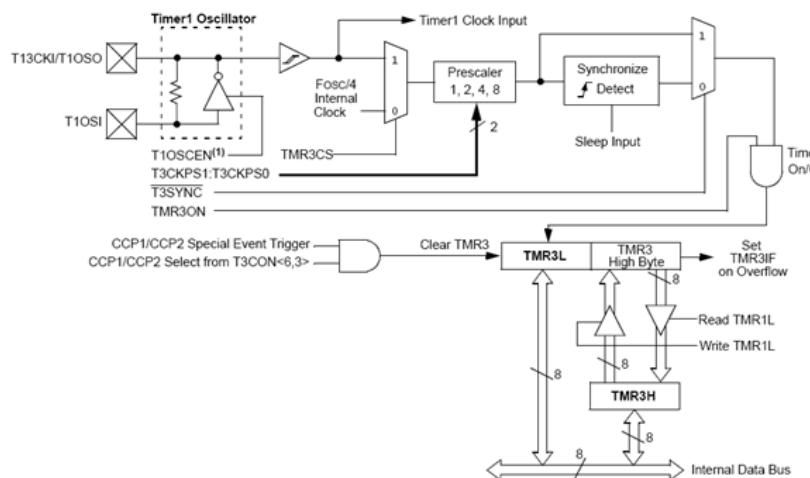


Figura 6.10: Diagrama do timer 3

Interrupção

O evento associado à interrupção do timer é o overflow da contagem de 16 bits.

6.2.5 Funções

As funções do C18 para uso dos timers estão contidas na biblioteca timers.h. A seguir é apresentado um resumo das principais funções.

Função	Protótipos	Descrição
OpenTimerx	void OpenTimer0(unsigned char config); void OpenTimer1(unsigned char config); void OpenTimer2(unsigned char config); void OpenTimer3(unsigned char config);	Configura e ativa o timer x
CloseTimerx	void CloseTimer0(void); void CloseTimer1(void); void CloseTimer2(void); void CloseTimer3(void);	Desativa o timer x
ReadTimerx	unsigned int ReadTimer0(void); unsigned int ReadTimer1(void); unsigned char ReadTimer2(void); unsigned int ReadTimer3(void);	Lê o timer x
WriteTimerx	void WriteTimer0(unsigned int timer); void WriteTimer1(unsigned int timer); void WriteTimer2(unsigned char timer); void WriteTimer3(unsigned int timer);	Escreve no timer x

Tabela 6.1: Funções da Biblioteca timers.h

6.3 Comparadores Analógicos

Quando trabalhamos com microcontroladores, podemos ter situações onde desejamos uma comparação rápida entre dois ou mais valores analógicos para que seja tomada alguma ação sem uma intervenção do programa principal. Isso poderia ser vantajoso se fosse feito de forma automática pelo microcontrolador ao invés de se ter módulos A/D para fazer este tipo de tarefa.

No microcontrolador 18F4550 temos um periférico que possui essa característica. O módulo de comparação analógica contém dois comparadores que podem ser configurados de várias formas diferentes. As entradas podem ser pinos multiplexados de entrada do portal A (RA0 até RA5), bem como podem ser referências de tensões obtidas dentro do microcontrolador. As saídas digitais podem ser obtidas com valores normais ou inversos, estando disponíveis na saída do módulo comparador ou ainda podem ser lidas através do registro de controle.

Abaixo temos o registro que serve para configuração da entrada e da saída do módulo comparador dentro do microcontrolador.

Registro CMCON:

Aqui temos a descrição de cada parte do registro:

- C2OUT: Armazena o valor de saída do comparador 2, onde:

R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-1	R/W-1
C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0
bit 7							bit 0

Figura 6.11: Registro de configuração do comparador

- Quando C2INV for zero, temos o valor um quando o valor de entrada Vin+ do comparador 2 for maior que Vin- e zero quando o valor de entrada de Vin+ for menor que Vin-;
- Quando C2INV for um temos exatamente a situação contrária da anterior.
- C1OUT: Armazena o valor de saída do comparador 1, onde:
 - Quando C1INV for zero, temos o valor um quando o valor de entrada Vin+ do comparador 1 for maior que Vin- e zero quando o valor de entrada de Vin+ for menor que Vin-;
 - Quando C1INV for um temos exatamente a situação contrária da anterior.
- C2INV e C1INV: Como já pôde ser notado, quando possuem valor um inverte o valor de saída do comparador C2 e C1 respectivamente e quando em zero não inverte;
- CM2:CM1:CM0: Estes três registros são responsáveis pela seleção do modo de funcionamento dos dois comparadores, onde sua aplicação pode ser vista abaixo:

Comparadores resetados CM2:CM0 = 000 <p> </p>	Comparadores Desligados CM2:CM0 = 111 <p> </p>
Dois Comparadores Independentes CM2:CM0 = 010 <p> </p>	Dois Comparadores Independentes com saída externa CM2:CM0 = 011 <p> </p>
Dois Comparadores com uma referencia comum CM2:CM0 = 100 <p> </p>	Dois Comparadores com uma referencia comum com saída externa CM2:CM0 = 101 <p> </p>
Um comparador independente com saída externa CM2:CM0 = 001 <p> </p>	Quatro entradas multiplexadas para dois comparadores CM2:CM0 = 110 <p> </p>
A = Entrada Analógica D = Entrada Digital (Usado como portal)	

Figura 6.12: Configurações possíveis do comparador.

- CIS: Este registro serve para selecionar a que pinos estarão conectados os Vin- dos comparadores C1 e C2. Assim, quando usamos CM2:CM1:CM0 = 110 temos:
 - Quando CIS for igual a um, C1 Vin- estará conectado a RA3/AN3/Vref+ e C2 conectado a RA2/AN2/Vref-/CVref;
 - Quando CIS for igual a zero, C1 Vin- estará conectado a RA0/AN0 e C2 conectado a RA1/AN1.

6.3.1 Funcionamento do Comparador

Valores de entrada e referência do comparador

Abaixo podemos ver a figura de um dos comparadores, bem como a relação entre os valores analógicos colocados na entrada e o sinal digital. Quando a entrada analógica Vin+ é menor que a entrada analógica Vin-, a saída do comparador também terá um valor de saída de nível lógico baixo. Quando a entrada analógica em Vin+ é maior que a entrada analógica Vin-, a saída do comparador possui sua saída com um nível lógico alto. As áreas em negrito da saída do comparador representam a incerteza do valor de saída devido aos offsets e o tempo de resposta das entradas.

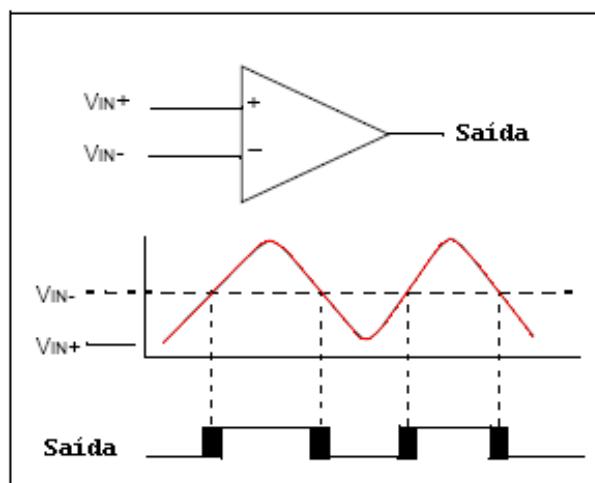


Figura 6.13: Representação do funcionamento do comparador

Dependendo ainda do modo do comparador, como visto anteriormente, um valor de tensão de referência interna ou externa pode ser usado para fazer as comparações de valores externos. O sinal analógico presente em Vin- da figura anterior é comparado com o sinal em Vin+ e a saída digital do comparador é ajustado de acordo com a variação dos mesmos.

Quando tensões de referências externas são usadas, o módulo comparador pode ser configurado para ter os dois comparadores utilizando a mesma fonte de referência externa ou ainda isto pode ser independente para cada um. Entretanto, o sinal de referência deve estar entre VSS (Geralmente GND) e VDD (Geralmente +5V) que podem ser aplicados a ambos os pinos do comparador.

O comparador também pode usar as tensões internas do microcontrolador como valor de referência no módulo comparador, sendo que a referência interna só está disponível para o modo onde as quatro entradas são multiplicadas para os dois comparadores. Neste modo, a tensão interna de referência será aplicada ao Vin+ de ambos os comparadores.

Tempo de resposta do comparador

O tempo de resposta de um comparador é determinado pelo tempo mínimo, depois de selecionado uma nova tensão de referência ou fonte de entrada, antes da saída do comparador ter uma saída válida. Se a tensão de referência é modificada, o atraso máximo para a mudança interna da referência deve ser considerado quando estamos utilizando os valores de saída do comparador. De outra forma, o atraso máximo dos comparadores seria usado.

Saída do comparador

As saídas dos comparadores são lidas através do registro CMCON como pode ser visto acima. Estes bits são somente para leitura. A saída do comparador pode também estar ligada diretamente aos pinos de entrada e saída RA4 e RA5. Quando habilitados, os multiplexadores da saída de cada um destes pinos são utilizados para a saída do comparador.

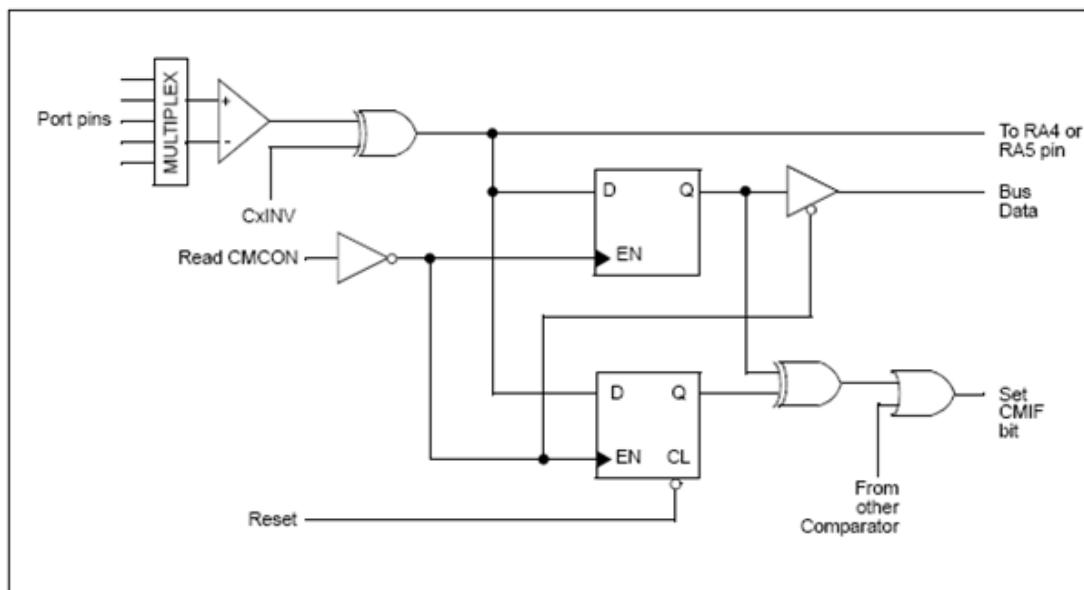


Figura 6.14: Descrição da saída do comparador

Interrupção

A interrupção acontece quando há uma mudança no estado da saída de ambos os comparadores independentemente. Onde os registros modificados são comentados no inicio deste capítulo. A interrupção pode ser verificada pelo o registro CMIF que representa a interrupção dos comparadores.

Funções

Para o uso do comparador o compilador C18 trás as funções descritas abaixo, pertencentes a biblioteca **ANCOMP.h**

Para ter acessos as saídas dos comparadores acessamo-los diretamente através dos bits CMCONbits.C1OUT e CMCONbits.C2OUT.

Função	Protótipos	Descrição
Open_ancomp	void OpenTimer0(unsigned char config);	Configura e ativa o comparador analógico.
Close_ancomp	void Close_ancomp(void);	Desativa comparador analógico e sua interrupção.

Tabela 6.2: Funções da biblioteca ANCOMP.h

6.4 Conversor Analógico-Digital

A maioria dos dados obtidos de sensores comuns, tais como sensores de temperatura, intensidade luminosa, posição, tensão, corrente e etc. fornecem sinais analógicos, ou seja, uma tensão que é proporcional à grandeza medida e que varia de forma contínua numa faixa de valores.

No entanto, a maioria dos equipamentos modernos que fazem a aquisição de dados destes sensores, trabalha com técnicas digitais. Isso significa que o dado analógico, preciso ser convertido para a forma digital. Para fazer esta conversão são utilizados circuitos denominados conversores analógico-digital, ou simplesmente A/D, como seu próprio nome indica, realiza a conversão de sinais, cuja amplitude varia continuamente em sinais digitais correspondentes à amplitude do sinal original.

Para converter se faz o uso de um comparador de tensão ou corrente - variando de acordo com a aplicação - que irá comparar o sinal analógico com o valor de referência.

Desta forma os circuitos A/D devem preencher certos requisitos importantes quanto ao seu desempenho que são:

- Quantização;
- Taxa de Amostragem e;
- Linearidade.

6.4.1 Quantização

Entre os dois valores extremos da escala de valores analógicos que devem ser convertidos para a forma digital existem infinitos valores intermediários, o que justamente caracteriza uma grandeza que varia de forma análoga ou analógica. Entretanto, não podemos simplesmente representar o valor analógico através de bits, pois para infinitos valores deveríamos ter infinitos bits para representar todas as variações possíveis. Para realizar a conversão de um sinal analógico para um valor digital deve ser definido o número de bits em que o valor será representado no universo digital e, a partir disso, definir em quantas faixas de valores digitais a faixa de valores analógicos será dividida.

Assim, por exemplo, se utilizarmos na conversão 4 bits, teremos a possibilidade de representar apenas 16 valores na escala total de valores analógicos, e se usarmos 8 bits poderemos representar 256 valores, conforme indica a seguir.

Se tivermos uma escala de 0 a 8 V, por exemplo, e usarmos 4 bits para a conversão, os "degraus" da escada de conversão terão 0,5 V de altura, o que significa que este conversor terá uma resolução de 0,5 V. Se usarmos um conversor A/D de 8 bits (256 "degraus" de resolução)

para fazer um voltímetro de 0 a 10 V, por exemplo, a resolução deste voltímetro será de 10/256 ou pouco menos de 0,04 V.



Figura 6.15: Escala de conversão

Este comportamento "digital" pode ser observado em muitos instrumentos comuns, tais como os multímetros digitais em que, se a grandeza medida estiver num valor intermediário entre dois degraus da resolução do conversor A/D, o valor apresentado no display oscilará entre eles.

Evidentemente, tanto maior é a precisão na conversão quanto mais bits são utilizados pelo conversor.

Tipos com 8 a 16 bits são comuns nas aplicações industriais e em medidas, dependendo da quantidade de "passos" desejados na conversão ou a resolução. Em aplicações de alta fidelidade pode-se trabalhar com até 24 bits.

Um fator importante que deve ser escolhido não se especificar o número de bits de um conversor é o ruído inerente ao circuito. Por exemplo, supondo uma aplicação onde valores de tensão entre 0 e 5V devem ser convertidos e onde se sabe que existe um ruído da ordem de 20mV no sinal a ser convertido. Ora, se usarmos um conversor A/D de 12 bits, por exemplo, teremos passos de quantização de $5V/(2^{12})$, isto é, da ordem de 1,2 mV. Portanto teremos uma precisão de conversão que não representa uma precisão de medida, pois o sinal de interesse possui uma imprecisão gerada pelo ruído. Aplicações com conversores A/D de 16 ou 24 bits exigem circuito extremamente imunes a ruído e interferência para poder funcionar de forma correta.

6.4.2 Taxa de Amostragem

Muitos processos de aquisição de dados de sensores, de processos ou de outras aplicações precisam ser rápidos. Uma placa de aquisição de dados de um instrumento de medida que projete uma forma de onda, desenhe um gráfico na tela de um PC representando um processo dinâmico ou mesmo um instrumento digital simples como um multímetro, deve estar constantemente convertendo sinais.

Um osciloscópio digital, por exemplo, deve medir as tensões instantâneas de um sinal em diversos pontos ao longo de um ciclo para poder "desenhar" esta forma de onda com precisão na tela. Se a frequência do sinal for alta, isso implica a necessidade de se fazer amostragens num tempo extremamente curto.

Os conversores A/D podem ser encontrados em tipos que têm frequências de amostragem numa ampla escala de valores. Os tipos mais rápidos têm suas velocidades especificadas em

MSPS (Mega Samples Per Second ou Milhões Amostragens Por Segundo).

Uma máquina industrial ou um instrumento de uso geral como um multímetro pode usar conversores A/D relativamente lentos com taxas ou velocidades de amostragens de até algumas unidades por segundo. Um multímetro digital comum, por exemplo, faz de 1 a 10 amostragens por segundo apenas, dependendo do tipo. Todavia, um osciloscópio digital ou virtual que precise observar uma forma de onda de 10 MHz, deve, para ter uma definição razoável, realizar pelo menos 100 milhões de amostragens por segundo (10 pontos por ciclo).

O conceito de taxa de amostragem está ligado também ao Teorema da Amostragem, a determinação da frequência máxima na entrada de um conversor A/D e ao dimensionamento dos chamados filtros anti-aliasing. Contudo não entraremos em detalhes sobre esses assuntos, sendo essa discussão mais adequada para um curso de processamento de sinais.

6.4.3 Linearidade

A curva de conversão da grandeza analógica para a forma digital deve ser linear para um bom conversor. Isso significa que não existem desvios na correspondência entre o valor analógico e a saída digital ao longo da escala de valores em que o conversor deve trabalhar. Em outras palavras, em um gráfico onde um eixo representa os valores analógicos de entrada e o outro os valores digitais de saída, a função de transferência deve ser idealmente uma reta. No entanto, na prática podem ocorrer pequenos desvios, de acordo com o que mostra a figura abaixo.

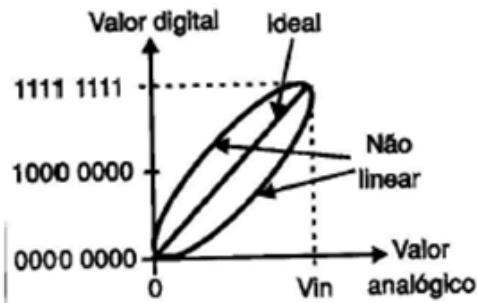


Figura 6.16: Grau de Linearidade de Conversão

Portanto, quanto mais linear um conversor A/D, melhor sua qualidade.

6.4.4 Desenvolvimento

Para fazer uma conversão de sinais analógicos para a forma digital existem diversas técnicas que são empregadas nos circuitos comerciais, muitas delas encontradas em circuitos integrados que são "embutidos" em aplicações mais complexas, os quais fazem o controle de máquinas e equipamentos. Analisamos as tecnologias mais empregadas para esta finalidade começando com o bloco comum a todos os conversores, que é o circuito de amostragem e manutenção (sample and hold).

O valor dos sinais analógicos que devem ser convertidos para a forma digital corresponde a um determinado instante, cuja duração, em alguns casos, não vai além de alguns milionésimos de segundo.

Assim, um primeiro bloco importante do conversor é um circuito que lê o valor do sinal a ser convertido num determinado instante e o armazena de modo que, mesmo que o sinal varie depois, os circuitos que fazem a conversão têm numa memória seu valor. Este circuito é ilustrado em blocos na figura abaixo. O sinal a ser amostrado é amplificado por um buffer de entrada cuja finalidade é não carregar o circuito externo, e ao mesmo tempo proporcionar isolamento do circuito de conversão.

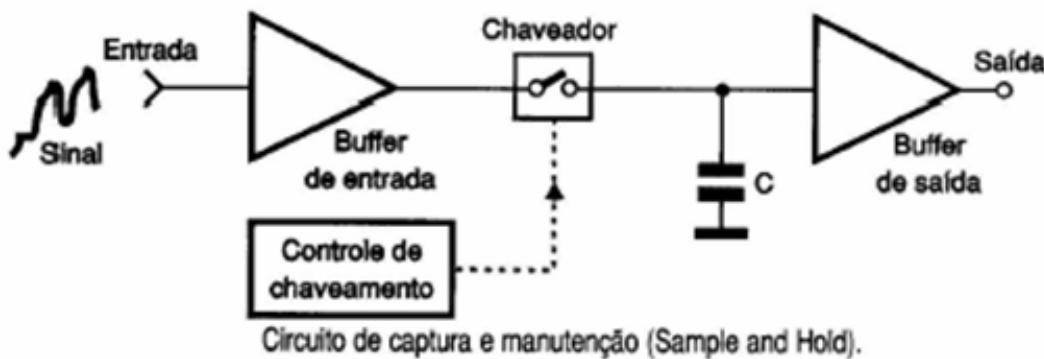


Figura 6.17: Diagrama em blocos do conversor A/D

Na saída deste circuito temos uma chave eletrônica ou chaveador, que determina o instante exato em que a leitura do sinal deve ser feita. A chave fecha então por uma fração de segundo (numa frequência que depende da taxa de amostragem) permitindo que o sinal carregue o capacitor C.

Assim, quando a chave abre, esperando a leitura seguinte, o capacitor tem armazenado o valor da grandeza analógica a ser convertida. Esta tensão no capacitor é mantida no circuito conversor através de um buffer de saída durante o tempo que ele necessita para isso.

Na figura abaixo temos um gráfico que indica de que modo à tensão de entrada varia e o circuito de amostragem e retenção mantém a saída constante durante os intervalos de conversão (que correspondem aos "degraus").

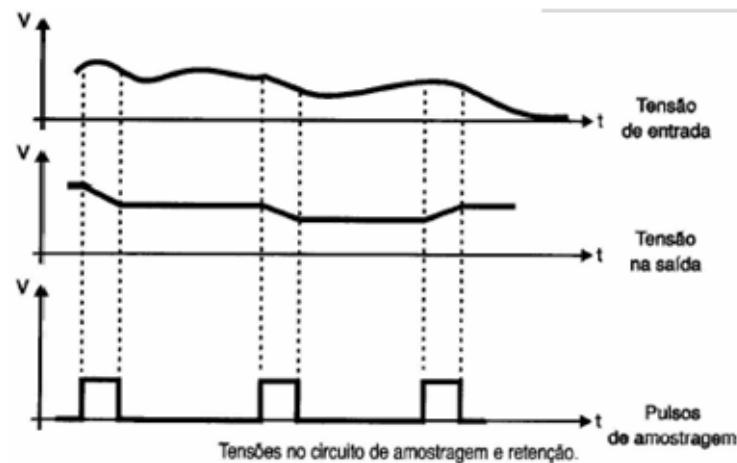


Figura 6.18: Escala de conversão

6.4.5 Aplicação

Existem várias formas de se construir conversores A/D, sendo que cada um tem a sua característica de funcionamento que deve ser levada em conta, na hora de se construir e/ou escolher para sua aplicação. Temos uma relação de possíveis combinações:

- Conversor A/D com comparador em paralelo;
- Conversor A/D com rampa em escada;
- Conversor A/D de aproximações sucessivas;
- Conversor A/D de rampa única;
- Conversor A/D de rampa dupla e;
- Sigma-Delta.

O Sigma-Delta é uma das importantes técnicas de conversão A/D, utilizada em aplicações que se deseja uma altíssima velocidade de conversão, como nos DSPs (Digital Signal Processing).

Portanto, vimos que a conversão do sinal analógico para o digital sempre existe uma perda de informação seja ela de amplitude - característica da quantidade de bits utilizados - ou de fase do sinal - característica da taxa de amostragem empregada.

Vimos que o erro máximo que pode ocorrer na quantização é de metade do valor de nível da quantização assim sendo quanto maior for o número de bits do conversor menor será o seu erro.

O erro de "Aliasing" é facilmente evitado utilizando o teorema da amostragem que "Para que uma determinada frequência f_1 do sinal analógico seja ou possa ser completamente reconstituída a taxa amostral, no processo de digitalização, deve ser no mínimo igual a $2*f_1$ "

Conhecidas as imperfeições da conversão podemos então saber quais os fatores que influem na escolha de um conversor A/D e assim prever melhor os ajustes que sistema deverá sofrer, pois já é conhecida as suas fraquezas.

6.4.6 Usando o conversor ADC no microcontrolador

O ADC (Analog-to-Digital Converter) converte a tensão em palavras de 10 bits. O PIC18F4550 possui um ADC de 10 bits com 13 canais. Isso significa que existem várias entradas para o conversor AD, porém somente uma entrada pode ser convertida por vez. As tensões de referência para a conversão, tanto superior (V_{ref+}) como inferior (V_{ref-}), podem ser selecionadas por software entre as tensões de alimentação e tensões presentes em determinados terminais do microcontrolador. Esses valores de tensão estipulam a faixa de valor a ser convertida. Por exemplo, sendo $V_{ref+} = +5V$ e $V_{ref-} = GND$ temos uma faixa de 5V, correspondendo o valor 0 a 0V e 1024 a 5V. Para determinar qual o valor correspondente dentro dessa faixa, basta aplicar uma simples regra de três.

A figura abaixo mostra o diagrama em blocos do ADC.

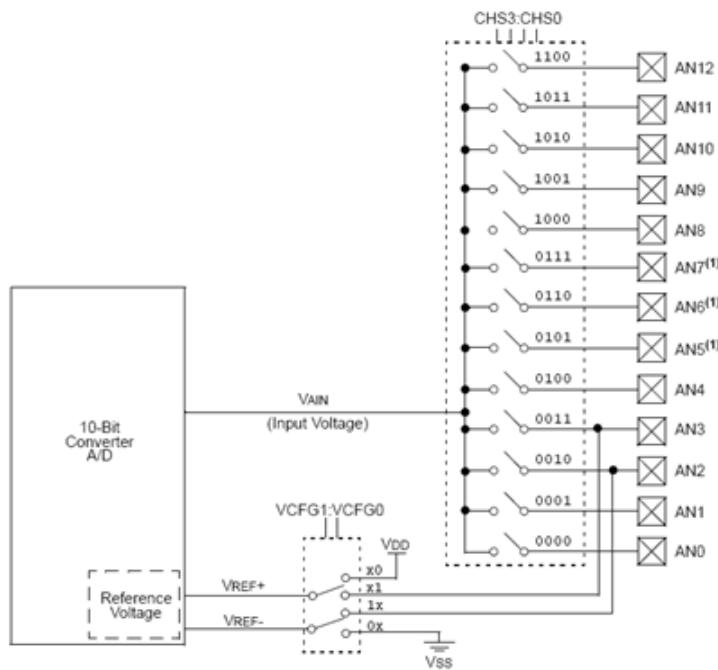


Figura 6.19: Diagrama em blocos do ADC do PIC18F4550

O módulo ADC realiza todo o processo de "Sample and Hold"(Amostragem e retenção). Esse processo é realizado quando uma determinada entrada é selecionada e inicia-se a carga de um capacitor interno CHOLD. Após o tempo de carga do capacitor (THOLD) a entrada é desconectada e inicia-se o processo de conversão da tensão armazenada no capacitor, que é feito pelo método de aproximações sucessivas.

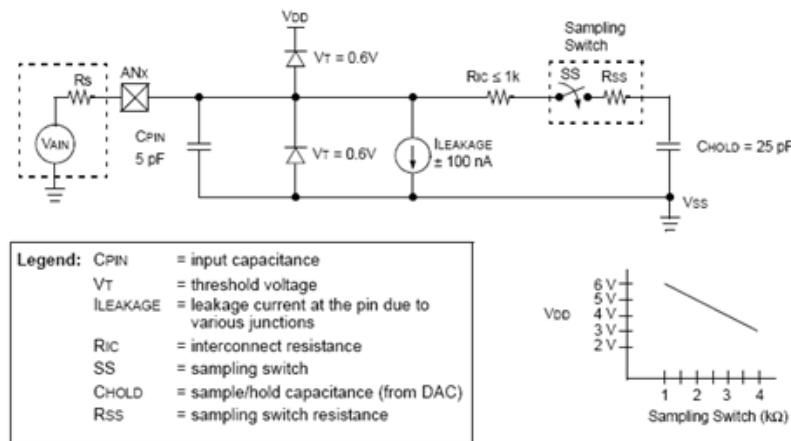


Figura 6.20: Diagrama em blocos do ADC do PIC18F4550

A resistência da fonte da tensão a ser convertida (R_s no circuito acima) não deve ser superior a $10\text{k}\Omega$, caso contrário o capacitor CHOLD pode não ser completamente carregado quando se iniciar a conversão resultando em uma medida incorreta.

Além disso, o processo de conversão só poderá ser iniciado depois de decorrido o tempo de carga do capacitor (THOLD). A ativação do processo de conversão será explicada mais adiante.

O tempo total de amostragem, que vai do instante em que o canal a ser amostrado é selecionado ao momento em que o resultado da conversão é armazenado nos registros de resultado do AD, é a

soma do tempo de aquisição com o tempo de conversão. O tempo de aquisição varia em função da temperatura, da tensão de alimentação e da resistência da fonte do sinal a ser amostrado. Para temperaturas inferiores a 25°C , alimentação de 5V o tempo de aquisição encontra-se no intervalo:

$$11\mu\text{s} < T_{AQ} < 20\mu\text{s};$$

Onde o menor tempo é conseguido para uma $\text{Rs} = 0\Omega$ e o maior para $\text{Rs} = 10\text{k}\Omega$.

Já o tempo de conversão depende do clock de conversão. Esse clock pode ser selecionado como Fosc/2, Fosc/8, Fosc/32 ou baseado em um oscilador RC interno.

Quando utilizando a fonte de clock proveniente do oscilador RC o processo de conversão pode ocorrer estando o microcontrolador em modo SLEEP.

Finalmente, após uma conversão ser concluída é necessário aguardar 2 períodos do clock de conversa antes de se reiniciar o processo.

Para a utilização do ADC são utilizados 4 registros especiais

1. ADRESH : registro de resultado do A/D (MSbits);
2. ADRESL : registro de resultado do A/D (LSbits);
3. ADCON0 : registro de controle 0;
4. ADCON1 : registro de controle 1.

Interrupção

O evento de interrupção do ADC é término de uma conversão completa. Para a implementação de sistemas de amostragem, onde o intervalo entre as amostras é fixo, pode-se configurar um timer com interrupção que inicie o processo de conversão do ADC e habilitar a interrupção do ADC para tratar as amostras quando a conversão terminar.

Funções

O As funções para uso do ADC estão na biblioteca adc.h.

Função	Protótipos	Descrição
BusyADC	7 char BusyADC(void);	Retorna 1 se o ADC estiver realizando uma conversão.
CloseADC	void CloseADC(void);	Desabilita o modo ADC.
ConvertADC	void ConvertADC(void);	Inicia uma conversão
OpenADC	void OpenADC(unsigned char config, unsigned char config2);	Configura o ADC
SetChanADC	void SetChanADC(unsigned char channel);	Seleciona um canal do ADC
ReadADC	int ReadADC(void);	Lê o resultado da conversão

Tabela 6.3: Funções de uso do ADC

Um cuidado especial deve existir quando se deseja desativar as entradas analógicas para fazer uso dos pinos a elas relacionados como I/O digitais. A função **CloseADC()** não desativa as entradas analógicas, sendo necessário utilizar **OpenADC(ADC_1ANA_0REF, ADC_INT_OFF)**.

6.5 CCP - Captura, Comparaçao e PWM

O módulo CCP (Captura, Comparaçao e PWM) permite realizar uma série de funções por hardware. Para exemplificar a versatilidade desse periférico, podemos citar como exemplos de sua utilização: geração de sinais de PWM, geração de sinais analógicos, medida de frequência, medida de largura de pulso, dentre várias. Existem três modos de operação:

- Captura;
- Comparaçao;
- PWM (Modulação por largura de pulso).

Esse periférico é controlado por dois registros que operam conjuntamente como um parâmetro de 16 bits (CCP1H e CCP1L) e um registro de controle (CCP1CON). Além disso, dependendo do modo selecionado, o módulo CCP interagem com os timers. Por fim, existe uma interrupção associada ao módulo CCP, que é disparada em situações diferentes para cada modo.

6.5.1 Modo captura

No modo captura, o valor do timer 1(TMR1H:TMR1L) é armazenado nos registros CCP1H:CCP1L quando ocorre um "evento". Além disso, o flag CCP1IF é setado, permitindo acionar a interrupção.

Esse evento pode ser a ocorrência de:

- 1 transição de descida no terminal RB3/CCP1;
- 1 transição de subida no terminal RB3/CCP1;
- 4 transições de subida no terminal RB3/CCP1;
- 16 transições de subida no terminal RB3/CCP1.

Para operação nesse modo, o terminal RB3/CCP1 deve estar configurado como entrada. O timer deve estar no modo temporizador ou no modo contador síncrono.

O modo captura pode ser utilizado para determinar a diferença de tempo entre dois eventos. Isso pode ser feito de duas formas. No primeiro evento, o timer é reiniciado e no segundo, o valor capturado do timer 1 multiplicado pelo período do seu clock corresponde ao tempo transcorrido entre os dois eventos. Outra forma é deixar o timer 1 incrementado livremente e capturar seu valor nos dois eventos. Fazendo uma subtração entre os dois valores capturados, temos o tempo entre os eventos. Esse procedimento pode ser utilizado para medir velocidade e frequências (nesse caso temos o período e sabemos que $f = 1/T$).

Os modos onde o evento de captura é a ocorrência de 4 ou 16 transições podem ser vistos como tendo uma pré-escala na entrada. Essa pré-escala pode ser útil para trabalhar com sinais de frequências altas. Sua contagem só pode ser zerada com a mudança de modo do CCP.

A figura abaixo apresenta o diagrama em blocos para o modo de captura.

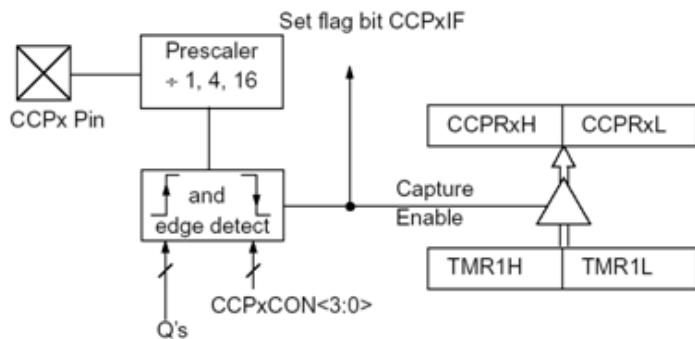


Figura 6.21: Diagrama do modo captura

Durante o modo SLEEP, se ocorrer um evento, o flag é setado, mas o valor do Timer 1 não é capturado, pois o timer 1 (como temporizador ou contador síncrono) não opera durante o SLEEP.

Interrupções

O bit CCPxIF quando ocorre um evento de captura.

Funções

As funções associadas ao modo de captura estão na biblioteca capture.h e são apresentadas a seguir lembrando que o PIC18F4550 possui um módulo CCP2 convencional e um módulog ECCP1 (Enhanced CCP).

Função	Protótipos	Descrição
CloseCapture2	void CloseCapture2(void);	Desativa o modo captura (CCP)
OpenCapture2	void OpenCapture2(unsigned char config);	Configura o módulo de captura (CCP)
ReadCapture2	unsigned int ReadCapture2(void);	Lê o valor capturado
CloseECapture1	void CloseECapture1(void);	Desativa o modo captura (ECCP)
OpenECapture1	void OpenECapture1(unsigned char config);	Configura o módulo de captura (ECCP)
ReadECapture1	unsigned int ReadECapture1(void);	Lê o valor capturado (ECCP)

Tabela 6.4: Funções de Captura

6.5.2 Modo comparação

Nesse modo, o par de registros do módulo CCP (CCP1H:CCP1L) é constantemente comparado com o par de registros do timer 1 (TMR1h:TMR1L). Quando eles coincidem, é setado o flag CCP1IF. Além disso, podemos zerar o timer 1 ou forçar um estado no pino RB3/CCP1. Quando optamos por zerar o timer 1, o funcionamento é muito parecido com o do timer 2 em relação ao registro PR2. Quando se seleciona os modos que forçam estados ('1' ou '0') no pino RB3/CCP1, o pino fica no estado oposto ao desejado até que ocorra a coincidência. Nesse caso, o pino deve ser configurado como saída.

A figura a seguir apresenta o diagrama do modo comparação.

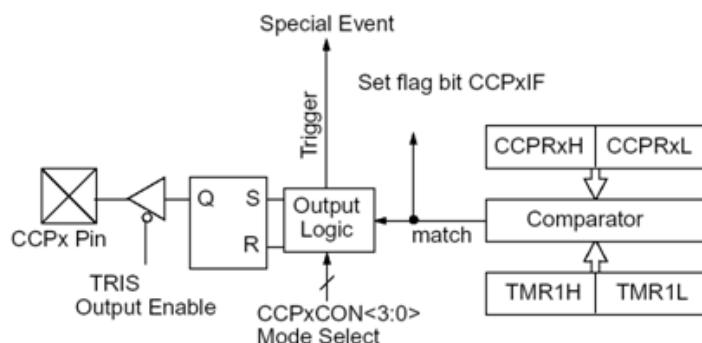


Figura 6.22: Diagrama do modo comparação

Interrupção

O bit CCPxIF é setado quando o Timer 1 e o valor de CCPRxH:CCPRxL coincidem.

Funções

A funções de comparação, presentas na biblioteca compare.h são apresentadas a seguir, lembrando que o PIC18F4550 possui um módulo CCP2 convencional e um módulog ECCP1 (Enhanced CCP):

Função	Protótipos	Descrição
CloseCompare2	void CloseCompare1(void);	Desativa o modo captura (CCP)
OpenCompare2	void OpenCapture1(unsigned char config);	Configura o módulo CCP C em modo de captura
CloseECompare1	void CloseCompare1(void);	Desativa o modo captura (CCP)
OpenCompare1	void OpenCapture1(unsigned char config);	Configura o módulo CCP em modo de captura

Tabela 6.5: Funções de comparação

6.5.3 Modo PWM

No modo PWM o módulo CCP permite utilizar sinais modulados em largura de pulso (PWM - Pulse Width Modulation), que consiste em representar um valor pelo duty-cicle (isto é, tempo em alto) de um trem de pulsos de frequência fixa. Por exemplo, admitindo-se que trabalhando com o PWM do PIC, sua resolução máxima é de 10 bits, ou seja, 1023 correspondem a 100% de duty-cicle. Usando uma regra de três simples, podemos determinar a quanto que corresponde 30%, 25%, 99%, etc. Este processo é chamado de modulação porque permite carregar uma informação (expressa no duty-cicle) em uma portadora (trem de pulsos).

A maior parte das aplicações de PWM para microcontroladores se aproveita da propriedade da energia de um sinal retangular ser proporcional ao seu duty-cicle (a energia de um sinal está relacionada com a área entre o sinal e o eixo do tempo). Vamos imaginar que um sinal PWM é aplicado a uma lâmpada DC. Um duty-cicle de 100% (sinal sempre em '1') fará a lâmpada acender em sua potência máxima; já um sinal com 70% de tempo em alto entrega a lâmpada 70% da potência máxima, e assim por diante. Essa propriedade é utilizada no acionamento de cargas DC, controle de motores, etc.

Outra característica importante do PWM é que, se o sinal for filtrado, podemos obter níveis analógicos, também proporcionais ao duty-cicle. Isso permite que geremos desde níveis analógicos fixos até sinais mais complexos, como tons DTMF (de telefonia).

O PWM precisa de uma base de tempo que dará a frequência do sinal. O módulo CCP utiliza o Timer 2 para conseguir essa base. Isso pode ser observado no diagrama de blocos do PWM.

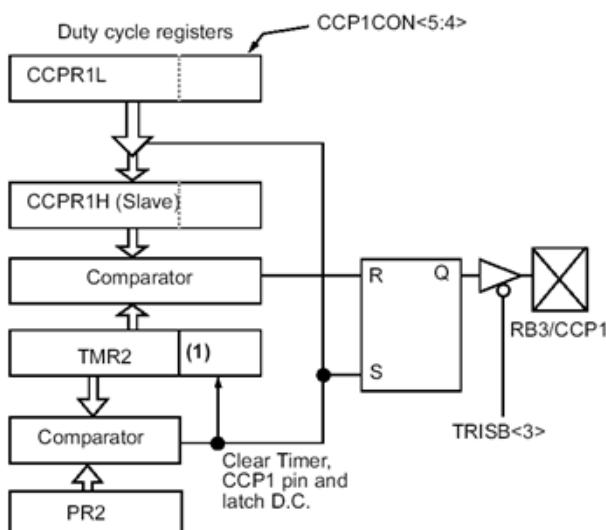


Figura 6.23: Diagrama em blocos no modo PWM

A geração do sinal PWM se dá da seguinte maneira. Cada vez que TMR2 coincide com PR2, o pino RB3/CCP1 é setado e TMR2 é reiniciado. Isso nos dá a frequência do sinal. O duty-cicle é conseguido comparando o CCPR1H concatenado com dois bits de latch com TMR2 concatenado com mais dois bits, da pré-escala ou gerados pelos ciclos Q; Quando há a coincidência, o pino RB3/CCP1 é zerado. As concatenações nos dão 10 bits de resolução. Esse processo pode ser observado na figura a seguir.

O registro CCPR1H não é acessível para leitura no modo PWM. O *duty-cicle* é configurado

através de CCP1L e dos bits 4 e 5 de CCP1CON. Esse valor é atualizado em CCPR1H e nos bits de latch a cada período.

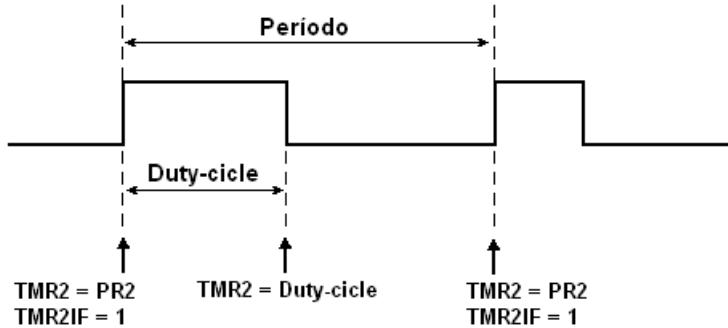


Figura 6.24: Sinal PWM

Configurar o PWM é estabelecer sua frequência e seu duty-cicle. De posse desses dois parâmetros, podemos calcular os valores em cada registro.

O período é o inverso da frequência e é configurado através de PR2 e da pré-escala do TMR2 e pode ser calculado através da equação a seguir:

$$T_{pwm} = (PR2 + 1) \times 4 \times T_{osc} \times (TMR2PS) \quad (6.1)$$

Onde:

- TPWM : período do PWM
- TOSC: período do oscilador
- TMR2PS: fator de pré-escala do timer 2

Para encontrar o valor de PR2 a partir de um dado valor de período pode ser usada a equação abaixo. Valor de PR2 deve ser inteiro e menor que 256. Logo, em alguns casos será necessário arredondar esse valor, o que gerará um pequeno erro entre a frequência desejada e a real. Com valores diferentes de pré-escala podemos chegar a valores menores que 256 e a aproximações que levem a um erro menor.

$$PR2 = \frac{T_{pwm}}{4 \times T_{osc} \times TMR2PS} - 1 \quad (6.2)$$

O duty-cicle por sua vez é configurado através de CCP1L e dos bits 4 e 5 de CCP1CON. Geralmente se especifica o duty-cicle em porcentagem do tempo total. Assim, dado um duty-cicle em porcentagem (DC%), o tempo correspondente a ele é encontrado pela equação abaixo.

$$T_{DC} = T_{PWM} \times \frac{DC\%}{100} \quad (6.3)$$

ou pela equação:

$$T_{DC} = DC[9 : 0] \times T_{osc} \times TMR2PS \quad (6.4)$$

Onde:

- TPWM : período do PWM;
- TOSC: período do oscilador;
- TMR2PS: fator de pré-escala do timer 2;
- DC[9:0] : é um valor de 10 bits obtido acrescentando os bit 5 e 4 a esquerda de CCP1RL.

Dispondo do tempo do duty-cicle, o valor de DC[9:0] é dado pela equação abaixo. Esse valor deve ser inteiro e menor que 1023. O valor da pré-escala deve ser o mesmo utilizado para a determinação do período.

$$DC[9 : 0] = \frac{T_{DC}}{T_{osc} \times TMR2PS} \quad (6.5)$$

O período (e consequentemente a frequência) é dado pelo registro PR2. Para frequências altas (períodos pequenos) existe uma perda da resolução, isto é, na verdade trabalhamos com menos de dez bits. Por exemplo, se o valor usado para PR2 for 63, os dois bits mais significativos são "perdidos" e 100% de duty-cicle corresponderá à 255. Teremos um PWM de 8 bits. A resolução, dada em bits, para uma dada frequência de PWM é dada pela equação abaixo:

$$RESOLUCAO = \frac{\log(\frac{F_{osc}}{F_{pwm}})}{\log(2)} \quad (6.6)$$

6.5.4 EPWM

O EPWM (Enhanced PWM, PWM avançado) em um modo de operação que o PWM assume algumas características especiais para acionamento de pontes de transistores ou FET, com a função de chaveamento usadas em UPS e controle de motores. Esse modo não será tratado neste curso.

Interrupção

Não há interrupção associada ao modo PWM.

Funções

Função	Protótipos	Descrição
ClosePWM2	void ClosePWM2(void);	Desativa o canal PWM (CCP)
OpenPWM2	void OpenPWM2(char period);	Configura o canal PWM (CCP)
SetDCPWM2	void SetDCPWM2(unsigned int dutycycle);	Escreve o Duty-cycle (CCP)
CloseEPWM1	void CloseEPWM1(void);	Desativa o canal PWM (ECCP)
OpenEPWM1	void OpenEPWM1(char period);	Configura o canal PWM (ECCP)
SetDCEPWM1	void SetDCEPWM1(unsigned	Escreve o Duty-cycle (ECCP)
SetOutputEPWM1	void SetOutputEPWM1 (unsigned char outputconfig, unsigned char outputmode);	Configura o modo de saída do ECCP

Tabela 6.6: Funções para PWM

6.6 EUSART - Porta Serial Assíncrona

6.6.1 Comunicação serial assíncrona

A comunicação entre circuitos eletrônicos digitais pode ser feita basicamente de duas maneiras: de forma paralela ou serial.

6.6.2 Comunicação paralela

No envio de dados de forma paralela todos os bits que compõe uma palavra são transmitidos ao mesmo tempo e por diferentes condutores. O exemplo clássico são os barramentos dos PC's, facilmente reconhecíveis pelas diversas trilhas que correm paralelas na placa de circuito impresso.

Esse tipo de comunicação exige pouco ou nenhum preparo para o envio, visto que os dados são tratados dentro do processador de forma paralela. Outra característica dessa forma de comunicação é a velocidade. Como a informação toda é enviada de uma só vez, a transferência de informações ocorre em tempo mínimo.

As limitações dessa técnica surgem quando as distâncias entre os pontos que se deseja comunicar aumentam. Em primeiro lugar para interconexão de equipamentos em distâncias grandes o uso de comunicação paralela exige conectores de muitos pinos e cabos de várias vias. Como pode ser intuído, o custo dos cabos e conectores é proporcional ao número de vias do cabo.

Além do custo dos cabos outro impedimento para a comunicação paralela são os efeitos reativos (capacitivos e indutivos) dos meios de comunicação paralela (sejam eles cabos ou trilhas numa placa). Em altas taxas de transmissão devemos considerar os condutores a luz da teoria de guias de ondas, que de forma resumida nos diz que em altas frequências um fio ou uma trilha não devem ser encarados como um curto-circuito, mas como uma associação de resistências, capacitâncias e indutâncias. Em virtude disso os sinais podem se degradar por interferência mútua e também, pelas próprias características do meio. Outro ponto importante é o ruído (interferência) inserido nos condutores e proveniente do onde eles estão.

Novamente o maior número de condutores agrava a situação. Esses fatos são inerentes a qualquer linha de transmissão de sinais elétricos, mas o agravamento dos problemas na comunicação paralela é que cada linha de transmissão é um meio diferente das demais, ou seja, cada bit é degradado de forma diferente. Isso pode acarretar problemas na comunicação de dados a distâncias longas (o critério para dizer se uma distância é "grande" é sua comparação com o comprimento de onda do sinal que se transmite). Assim limita-se a comunicação para pequenas distâncias em alta velocidade (por exemplo, barramentos do PC) ou a distâncias maiores com baixa taxa de transmissão (por exemplo, comunicação entre PC e impressora).

6.6.3 Comunicação serial

A forma encontrada para solucionar os problemas apresentados pela comunicação paralela foi transmitir os bits de forma serial, ou seja, um bit de cada vez. Quanto ao quesito custo essa forma de transmissão é bastante eficiente, porque em sua forma mais simples pode ser implementada com apenas dois condutores, um para envio de dados e outro de referência.

A comunicação serial também traz vantagens no aumento das taxas e das distâncias de comunicação. Técnicas para a redução de ruído e degradação do sinal podem ser aplicadas mais

facilmente quando a informação transita por um único caminho. Além disso, não há problemas em os bits serem afetados de forma diferente, pois são tratados separadamente.

Em contrapartida, a comunicação serial demanda um circuito mais complexo, uma vez que a informação que é tratada de forma paralela pelo processador deve ser convertida para o formato paralelo. Pelo mesmo motivo, a transmissão é mais lenta que a paralela (observando apenas o tempo do envio de uma informação). O principal problema a ser resolvido na comunicação serial, porém, é saber quando um bit termina e quando começa o próximo. E ainda, os cabos não deixam de serem guias de ondas, apresentando os problemas discutidos acima e mantendo sempre uma relação de compromisso entre o comprimento dos cabos e a taxa transmissão máxima.

Por todas essas razões a quase totalidade das conexões entre equipamentos é feita por comunicação serial. Entre as formas de comunicação serial mais difundida podemos citar o EIA-232, EIA-485, o USB e mesmo as LAN's Ethernet.

A comunicação serial pode ser feita de duas maneiras principais, a síncrona e assíncrona, que diferem na forma de localizar cada bit em uma "rajada".

Como o próprio nome diz, na comunicação serial síncrona existe um sincronismo, nesse caso entre o sinal transmitido e um clock enviado juntamente. O clock permite determinar o exato momento em que o bit do sinal deve ser lido, evitando assim erros na recepção e a correta montagem do dado na forma paralela. Entre as formas de comunicação serial síncrona podemos citar o I2C e o SPI, que não serão tratados aqui.

Já na comunicação assíncrona nenhuma referência de onde o bit deve ser lido é enviado com o sinal. A solução nesse caso é "adivinhar" o momento certo de ler o bit. Para isso é necessário que o receptor saiba a taxa de transmissão e portanto a duração de cada bit. Ainda que o transmissor indique de alguma forma onde começa e onde termina a transmissão. Dessa maneira, o receptor aguarda a chegada da indicação de início, chamado start bit ou bit de início e quando esse é lido ele sabe que a cada intervalo de tempo, chamado tempo de bit (T_b) um bit novo está presente na via de comunicação. Para minimizar o risco de erro a leitura é feita na metade da duração do bit ou são feitas algumas amostras durante esse intervalo de tempo.

Aparentemente, se procedermos dessa forma, bastaria enviar um start bit e para sincronizar receptor e transmissor e depois poderia vir uma sequência infinita de bits. Na verdade, porém, isso não ocorre, pois sempre existiria uma pequena diferença entre as bases de tempo de TX e RX. Como uma amostra ocorre T_b após a anterior, se esse tempo estiver errado, o erro vai se acumulando até que se perca o sincronismo. Por exemplo, se o relógio de RX for 5% mais lento ou mais rápido que o de TX a cada 20 bits recebido o sincronismo será perdido. Na prática os dados são enviados em "pacotes" de alguns bits, iniciados por um start bit e terminados por um stop bit. O stop bit tem a função de marcar o fim do pacote, para que um bit de informação não seja confundido com um novo start bit.

O formato genérico utilizado em um "pacote" de comunicação serial é apresentado na figura abaixo.

6.6.4 EUSART do PIC18F4550

O módulo EUSART do PIC18F4550 permite gerar sinais de comunicação assíncrona segundo o exposto acima. Ela é capaz de operar em modo full-duplex (transmissão e recepção simultâneas) e gerar uma taxa de transmissão/sincronismo de FOSC/4 bits por segundo.

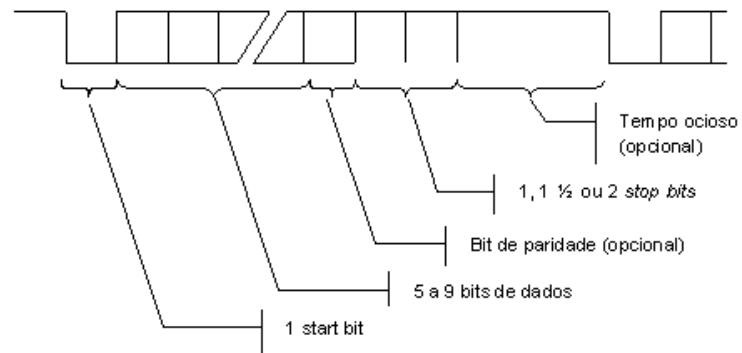


Figura 6.25: Comunicação Assíncrona

A EUSART do PIC é controlada pelos registros:

- TXSTA: Controle de transmissão;
- RCSTA: Controle de recepção;
- BAUDCON: Controle da taxa de transmissão;
- SPBRGH: SPBRG: Gerador de taxa de transmissão.

Operando em modo assíncrono, a EUSART é composta de um módulo de recepção e um de transmissão que trabalham de forma independente, apenas compartilhando o mesmo gerador de taxa de recepção/transmissão. As figuras abaixo apresentam os diagramas em blocos do sistema de recepção e o diagrama em blocos do módulo de transmissão.

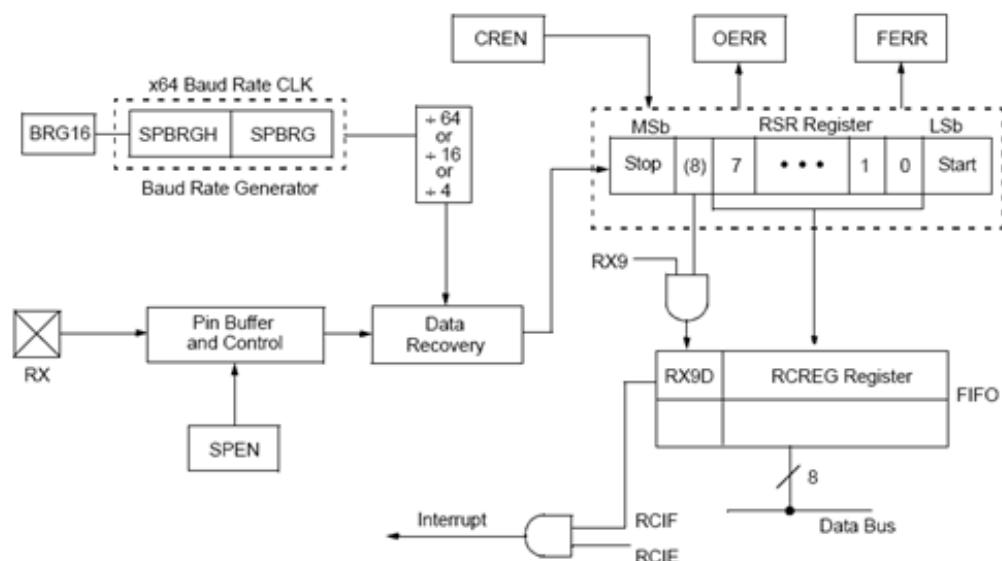


Figura 6.26: Módulo de Recepção

Gerador de taxa de transmissão

Para gerar a temporização necessária para a transmissão e a recepção, existe um gerador de taxa de transmissão (baud rate). Esse gerador deve ser configurado para a taxa de transmissão desejada levando-se em consideração a frequência de clock do oscilador principal.

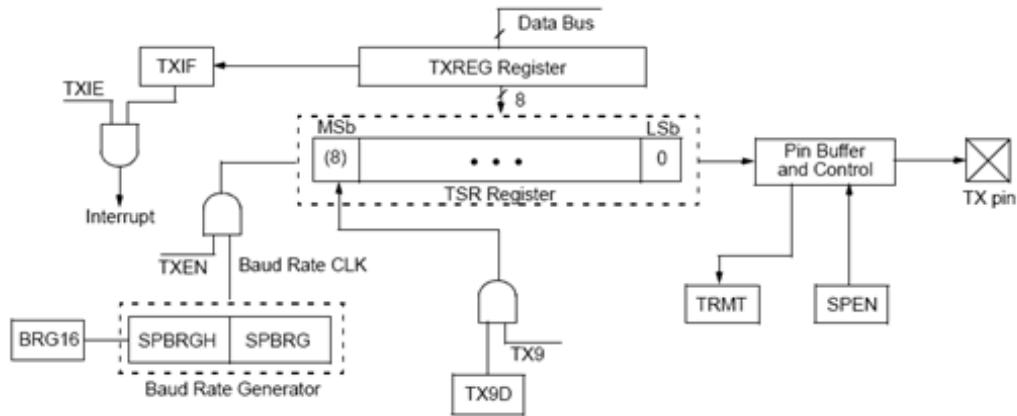


Figura 6.27: Módulo de Transmissão

O gerador de baud rate pode operar em dois modos, taxas altas e taxas baixas. Conforme a seleção do modo é usada uma das seguintes equações:

- Taxas altas (USART_BRGH_HIGH)

$$Br = \frac{F_{osc}}{(16 \times spbrg + 1)} \quad (6.7)$$

$$spbrg = \frac{F_{osc}}{16 \times BR} - 1 \quad (6.8)$$

- Taxas baixas (USART_BRGH_LOW)

$$Br = \frac{F_{osc}}{(64 \times spbrg + 1)} \quad (6.9)$$

$$spbrg = \frac{F_{osc}}{64 \times BR} - 1 \quad (6.10)$$

Sendo spbrg o valor a ser carregado no gerador de baud rate e BR a taxa de transmissão desejada

Deve-se estar atento, pois valor calculado pode não ser um inteiro. Neste caso, ao fazer a aproximação para um inteiro mais próximo o valor de taxa de transmissão pode ser muito diferente do valor nominal. Sempre que arredondar o valor obtido para spbrg calcule a taxa real obtida com o valor de spbrg e obtenha o erro através da equação a seguir.

$$e\% = \frac{|BR_{desejada} - BR_{calculada}|}{BR_{desejada}} \times 100 \quad (6.11)$$

Na prática, podem ocorrer problemas de comunicação para se a diferença entre o valor real e o nominal forem maiores que 5%.

Interrupção

O flag de interrupção de recepção (RCIF) é setado (colocado em nível um) quando um novo byte é recebido com sucesso. Uma aplicação útil é usar a RTI para receber um buffer de vários bytes e tratar apenas quando todos os dados chegarem.

Por sua vez, o flag de interrupção de recepção (TXIF) é setado quando um byte é enviado pela serial (buffer de transmissão vazio). Isto permite construir RTI que transmitam sequencialmente um buffer de diversos bytes sem que o programa principal tenha que aguardar a transmissão de cada byte, seja qual for a taxa de transmissão.

Funções

O C18 define diversas funções de uso da USART. Uma opção é trabalhar com o conjunto de funções definidos pelo C18 par USART na biblioteca USART.H.

Função	Protótipos	Descrição
BusyUSART	char BusyUSART(void);	Retorna 1 se a USART está transmitindo
CloseUSART	void CloseUSART(void);	Desabilita USART
DataRdyUSART	char DataRdyUSART(void);	Retorna 1 se um byte foi recebido
getcUSART		Lê um byte da USART
getsUSART	void getsUSART (char *buffer, unsigned char len);	Lê uma string da USART
OpenUSART	void OpenUSART(unsigned char config, unsigned int spbrg);	Configura USART
putcUSART	char getcUSART(void);	Envia um byte pela USART
putsUSART	void putsUSART(char *data);	Envia um string alocada na RAM pela USART
putrsUSART	void putrsUSART(const rom char *data);	Envia um string alocada na FLASH pela USART
ReadUSART	char ReadUSART(void);	Lê um byte da USART
WriteUSART	void WriteUSART(char data);	Envia um byte pela USART
baudUSART	void putcUSART(char data);	Configura a taxa de transmissão,

Tabela 6.7: Funções da USART

Outra opção é trabalhar com as funções padronizadas do C ANSI para saída de caracteres, presentes na biblioteca STDIO.H. Essa biblioteca define funções para entrada e saída de dados por uma porta do sistema, que no caso do C18 é a EUSART. Estas funções permitem apenas a escrita de dados, e não sua leitura. No decorrer do curso muitas vezes será usada a função printf(). Contudo, seu uso deve ser evitado na prática, devido ao tamanho do código gerado por ela.

Existem ainda funções que permitem implementar a comunicação serial através de pinos de I/O, das quais não trataremos. Essas funções estão na biblioteca SW_UART.

6.6.5 EIA-232C

Popularmente conhecido como RS-232, esse protocolo foi inicialmente desenvolvido para permitir a comunicação entre computadores e modems, para transmissão de dados a longa distância. A norma que rege o protocolo é a TIA/EIA-232, cuja revisão C é a mais recente (1969).

Devido a sua aplicação inicial como protocolo de comunicação entre um terminal de computador e um equipamento de comunicação, o protocolo estabelece os conceitos de DTE (Data Terminal Equipment - Equipamento Terminal de Dados) e DCE (Data Communication Equipment - Equipamento de comunicação de dados). O sentido dos pinos de comunicação é dado do ponto de vista do DTE. Como tipicamente o EIA-232 é utilizado para comunicação entre

um computador e o microcontrolador vamos sempre admitir aqui que computador é o DTE e o microcontrolador DCE.

A norma especifica vários pinos mas na prática os mais utilizados são os apresentados na tabela a seguir. As funções e direções dos pinos são dadas em função do DTE

Pino	Função	Direção
TxD	Transmissão do DTE para o DCE	Saída
RxD	Transmissão do DCE para o DTE	Entrada
RTS	Sinaliza que o DTE está pronto para receber dados do DCE	Saída
CTS	Sinaliza que o DCE está pronto para receber dados do DTE	Entrada
DSR	DCE pronto para operação	Entrada
DTR	DTE pronto para operação	Saída

Tabela 6.8: Funções da USART

Os pinos DSR e DTR servem para indicar que os equipamentos estão conectados e prontos para comunicação. DTS e CTS servem para fazer controle de fluxo: Quando o DTE tem dados para transmitir ele informa o DCE através de RTS; se o DCE pode receber esses dados CTS é ativado.

A sinalização são ativadas em '0' e desativadas em '1'. Em uma grande maioria dos casos são utilizados apenas os terminais de transmissão e recepção de dados, podendo ser usado um cabo com apenas 3 fios (TxD, RxD e Terra).

Segundo a norma, a comunicação serial pode suportar taxas de bit (geralmente chamadas baud rate) de até 20 kbps. É possível realizar comunicação full-duplex, ou seja, nos dois sentidos ao mesmo tempo.

A distância máxima do cabo de comunicação deve ser inferior a 15 metros (50 pés). Umas das principais causas da limitação de taxa e distância do EIA-232C é limitação nos tempos de subida e descida do sinal, que deve ser menor que 4% do tempo tempo de bits.

As especificações elétricas do EIA-232C podem ser observadas na figura abaixo.

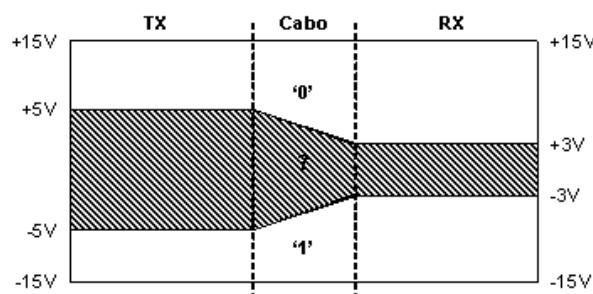


Figura 6.28: Especificações elétricas do EIA-232C

No transmissor o nível lógico '0' é representado por uma tensão entre +5 e +15 Volts e o nível lógico '1' é representado por tensões entre -5 e -15 Volts. No receptor tensões entre +3 e +15 Volts são interpretadas como '0' e entre -3 e -15 Volts interpretadas como '1'; tensões entre +3 e -3 Volts levam a estados indeterminados. A diferença entre os limites de tensão no transmissor (-5V e +5V) e os limites do receptor (-3V e +3V) constitui a margem de segurança, dentro da qual ruídos e eventuais perdas no cabo não degradam os dados.

As tensões máxima mínima que podem ser aplicadas ao receptor sem que haja dano são, respectivamente, +25V e -25V. Além disso, a norma especifica que qualquer curto circuito entre quaisquer pinos, inclusive o pino de terra, não devem causar dano ao circuito.

Como na maioria das aplicações trabalha com circuitos alimentados com +5V são necessários conversores de nível para criar uma interface elétrica com o EIA-232, dentre os quais os mais conhecidos são o MAX232 e seus equivalentes.

EIA-485

Outro padrão muito difundido, principalmente no meio industrial, é o EIA-485, também conhecido como RS-485. As principais vantagens desse padrão sobre o EIA-232C são as maiores distâncias alcançadas e a possibilidade de operação em rede com múltiplos usuários.

No EIA-485 a transmissão é feita por dois fios de forma balanceada. Isso significa que por um condutor é enviado um sinal A e pelo outro condutor uma cópia invertida B ($B = -A$) desse sinal. Na recepção há um circuito diferenciador que faz diferença entre o sinal A e B, portanto na saída desse circuito temos:

$$S = A - B, \text{ mas } B = -A, \text{ portanto, } S = A - (-A) = 2A \quad (6.12)$$

Como o ruído age de forma aditiva, somando-se ao sinal dos dois condutores, na recepção o efeito do ruído é cancelado pela ação do diferenciador, conforme pode ser observado a seguir, onde R representa o sinal de ruído.

$$S = (A + R) - (B + R) = (A + R) - (-A + R) \Rightarrow S = A + R + A - R = 2A \quad (6.13)$$

Além disso, a rede é ”casada” com resistores de terminação, o que reduz a degradação por reflexão do sinal.

Podemos ligar diversos usuários em rede, sendo tipicamente usada a topologia de barramento para tanto. Em um barramento EIA-485 somente um dos terminais pode transmitir por vez, ficando os outros com o driver de saída desligado. Portanto só é possível a comunicação no modo half-duplex. A figura abaixo mostra os esquemas ligação de diversos transceptores a rede. Os resistores RT são resistores de terminação e devem ser colocados nas extremidades do barramento. Seu valor típico é $120\ \Omega$, portanto um valor equivalente de 60Ω .

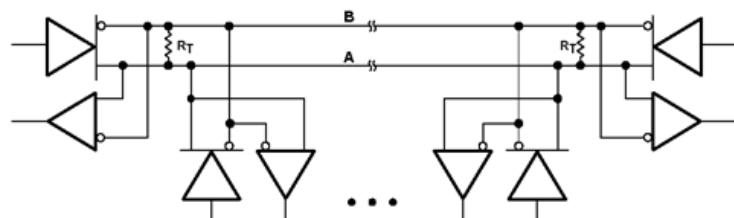


Figura 6.29: Esquemas ligação de diversos transceptores a rede

O número máximo de pontos na rede é limitado pela carga que cada terminal receptor representa para o barramento, que deve ser de 1mA para 12V, e pelas características do buffer de

saída. Seguindo padrão EIA-485 a quantidade de pontos fica limitada a 32 pontos, mas atualmente existem tranceivers com características alteradas que permitem até 128 pontos na rede.

A distância máxima possível na rede é de aproximadamente 1200m (4000 pés). A taxa máxima de transmissão é de 10Mbps. A taxa de transmissão é dada em função dos tempos de subida (t_R) e descida do sinal (t_F), sendo que no EIA-485 os TR e TF podem corresponder a 30% do tempo total do bit. Assim:

$$R_{max} = \frac{1}{t_b} = \frac{0.3}{t_r} \quad (6.14)$$

Sendo:

- RMAX: Taxa máxima de bits;
- tB: Tempo de bit;
- tR: Tempo de subida.

O tempo de subida depende do tipo e comprimento do cabo.

Os sinais elétricos no barramento devem estar entre -7 e +12 Volts em relação a referência. A diferença de tensão entre os sinais A e B também deve estar entre -7 e +12Volts. Podem ser detectados sinais entre A e B com diferença a partir de que 200mV e a diferença mínima no transmissor é de 1,5V, o que resulta em uma margem de segurança de 1,3 Volts. Temos nível lógico '1' quando VA > VB e nível lógico '0' quando VA < VB.

Dos trancetores mais utilizados no mercado podemos citar o MAX485 e seus equivalentes e o SN75176. Esses dois modelos são compatíveis entre si e possuem habilitação individual dos drivers de recepção e transmissão.

6.6.6 Protocolos de comunicação

Em aplicações de comunicação muitas vezes não basta apenas transmitir os dados que se deseja de um receptor para um transmissor. Por exemplo, devemos implementar meios de detectar eventuais erros na transmissão ou delinear o início se fim da mensagem. Além disso, em redes com múltiplos usuários é necessário identificar qual o usuário a que se destina o dado e criar métodos que evitem a colisão de dados(dois usuários transmitindo ao mesmo tempo na rede). Essas são funções da camada 2 do modelo OSI, a camada de enlace, e podem ser implementadas através de rotinas que obedeçam um protocolo de comunicação. A função dessas rotinas é prover o transporte dos dados através de diversos mecanismos e entregar os dados para a aplicação.

Comunicação ponto-a-ponto (para EIA-232)

Nesse caso típico de aplicação para o padrão EIA-232 temos comunicação bidirecional full-duplex, portanto não há risco de colisão e qualquer um dos equipamentos (no caso, o PC e o microcontrolador) podem iniciar a comunicação. Abaixo é apresentado o formato do frame desse protocolo, que é a forma como os dados serão "encapsulados" para transmissão.

1	1	0~254	1
Stx	NB	Dados	CKS

- STX: byte 0x02; sinaliza o início do frame; 1 byte;
- NB: Número de bytes que vêm a seguir; 1 byte;
- Dados: informação útil; de 0 a 254 bytes;
- CKS: checksum, calculado a partir de todos os demais campos.

STX serve para identificar o início do frame. Como NB informa quantos bytes devem ser recebidos, é possível determinar onde termina o frame. O último campo do frame é o checksum, que é uma soma em módulo dois (operação ou-exclusiva) de todos os bytes do frame. O checksum é calculado no transmissor e novamente no receptor. Se o checksum calculado no receptor não for o mesmo do enviado, significa que houve algum erro na transmissão.

Se for implementado um sistema de retransmissão caso ocorram erros o transmissor deve iniciar um timer quando um byte for recebido. Esse timer deve estar configurado para estourar no tempo máximo de espera aceitável para uma resposta. Caso um frame seja recebido com sucesso, um frame com uma confirmação positiva é enviado de volta. Se houver erro, é enviada uma confirmação negativa. Por fim, se o timer do transmissor estourar (time-out) e não for recebido nenhuma confirmação, o frame deve ser retransmitido.

Comunicação ponto-multiponto (para EIA-485)

Em redes 485 podemos ter vários terminais e a comunicação é half-duplex, portanto existirá uma disputa pelo uso do meio e caso dois terminais da rede transmitam ao mesmo tempo, a informação se perderá. Além disso é necessário identificar no frame o endereço do terminal ao qual ele se destina, portanto cada terminal deve ter um endereço individual.

Se todos os terminais puderem iniciar a comunicação há uma grande probabilidade de ocorrência de colisão. A solução mais simples nesse caso é determinar que um dos terminais (mestre) tenha iniciativa de iniciar a comunicação, sendo que os demais terminais (escravos) limitam-se a responder. Esse tipo de comunicação Mestre-Escravo (Master-Slave) é bastante comum em aplicações onde o terminal mestre é um computador que comanda uma rede de escravos microcontrolados. Abaixo temos o aspecto do frame para a comunicação mestre ⇒ escravo (comando).

1	1	1	0~254	1
Stx	NB	End	Dados	CKS

- STX: byte 0x02; sinaliza o início do frame; 1 byte;
- NB: Número de bytes que vêm a seguir; 1 byte;
- End: indica o endereço a que se destina o frame, se for 0xFF se destina a todos;
- Dados: informação útil; de 0 a 253 bytes;
- CKS: checksum, calculado a partir de todos os demais campos.

A seguir temos o aspecto do frame para a comunicação escravo → mestre (resposta).

1	1	1	0~254	1
Stx	NB	End	Dados	CKS

- STX: byte 0x02; sinaliza o início do frame; 1 byte.
- NB: Número de bytes que vêm a seguir; 1 byte;
- End: indica o endereço a que se destina o frame, se for 0xFF se destina a todos;
- Dados: informação útil; de 0 a 253 bytes;
- CKS: checksum, calculado a partir de todos os demais campos.

Nesse caso o mecanismo de detecção de erro e repetição é o mesmo do caso ponto-a-ponto.

6.7 MSSP - Porta Serial Síncrona

O PIC18F4550 possui um módulo MSSP (Master Synchronous Serial Port - Porta de Comunicação Serial Síncrona Mestre). Esse módulo permite a implementação de dois protocolos muito usados em comunicação entre circuitos integrados, o SPI (Serial Peripheral Interface) e o I2C (Inter-Integrated Circuits). Os modos possíveis de operação são:

- Modo SPI;
 - Modo mestre;
 - Modo escravo.
- Modo I2C.
 - Modo mestre;
 - Modo multi-mestre;
 - Modo escravo.

6.7.1 SPI

A comunicação SPI especifica 3 pinos de comunicação de dados: Clock (SCK), Entrada de Dados (SDI) e Saída de Dados (SDO). Além disso, cada periférico SPI deve ter um sinal de habilitação baixo ativo (normalmente chamado CE - Chip Enable ou CS - Chip Select).

A tabela abaixo apresenta os sinais, suas funções e os pinos correspondentes nos PIC18F452 e PIC18F4550.

Sinal	Pino	Função
SDO	RC7	Saída de dados
SDI	RB0	Entrada de dados
SCK	RB1	Modo mestre: saída de clock Modo escravo: entrada de clock
SS	RA5	Habilitação no modo escravo.

Tabela 6.9: Funções da USART

A conexão entre dispositivo SPI deve ser feita conforme o diagrama abaixo. Observe que a saída do mestre (SDO) é ligada a entrada dos escravos (SDI) e a saída dos escravos (SDO) é ligada a entrada do mestre (SDI). Alguns fabricantes utilizam também a nomenclatura MISO (Master Input, Slave Output) e MOSI (Master Output, Slave Input) para simbolizar os pinos de dados.

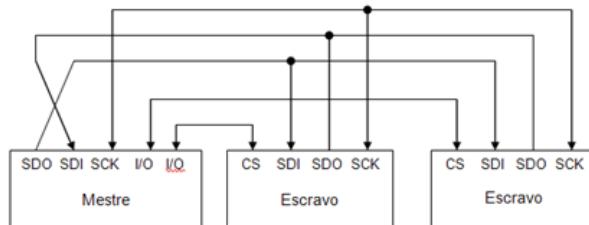


Figura 6.30: Pinos para operação em SPI

O protocolo SPI permite a comunicação do microcontrolador com diversos outros componentes, formando uma rede.

Operando em modo mestre, o microcontrolador gera o sinal de clock e deve ter um pino de I/O para a habilitação de cada periférico SPI. O periférico (escravo) habilitado se comunica com o microcontrolador (mestre) enquanto os demais, que não estão habilitados, ignoram a comunicação.

Operando em modo escravo, o microcontrolador comporta-se como um componente da rede, recebendo o sinal de clock e sendo habilitado pelo pino SS.

O diagrama em blocos do módulo MSSP configurado para o modo SPI é apresentado na figura abaixo.

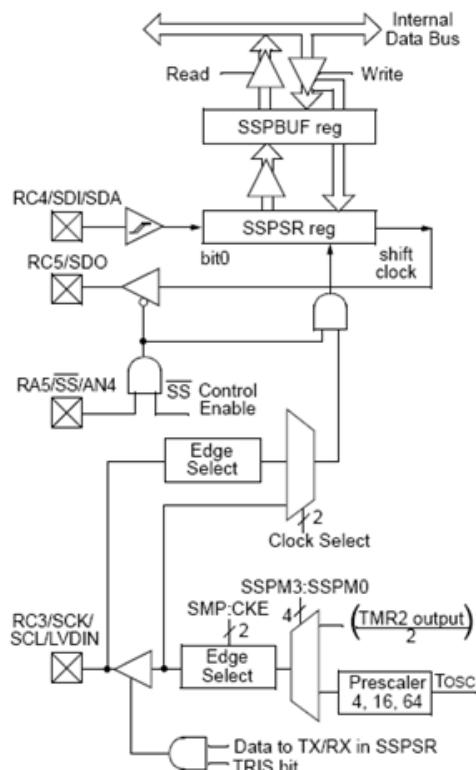


Figura 6.31: Diagrama em blocos para o modo SPI

A taxa de transmissão do módulo é selecionada através da escolha da fonte de clock que pode

ser FOSC/4, FOSC/16, FOSC/64 ou o sinal de coincidência do timer 2 (nesse caso, o período de clock é o dobro do tempo de coincidência do timer 2).

Interrupção

A interrupção no modo SPI é acionada toda vez que um dado é recebido ou enviado.

Funções

O compilador C18 possui funções para comunicação SPI na biblioteca SPI.H descritas abaixo.

Função	Protótipo	Descrição
CloseSPI	void CloseSPI(void);	Desabilita o módulo MSSP
DataRdySPI	unsigned char DataRdySPI(void);	Informa se recebeu dados
getcSPI	unsigned char getcSPI(void);	Lê um byte
getsSPI	void getsSPI(unsigned char rdptr, Unsigned char length);	Lê uma string
OpenSPI	void OpenSPI(unsigned char sync_mode, unsigned char bus_mode, unsigned char smp_phase);	Configura e habilita o MSSP em modo SPI
putcSPI	unsigned char putcSPI(unsigned char data_out);	Escreve um byte
putsSPI	void putsSPI(unsigned char *wrptr);	Escreve uma stringa
ReadSPI	unsigned char ReadSPI(void);	Lê um byte
WriteSPI	unsigned char WriteSPI(unsigned char data_out);	Escreve um byte

Tabela 6.10: Funções da SPI

6.7.2 I2C

O protocolo I2C especifica 2 sinais de comunicação, um com o sinal de clock (gerado pelo mestre) e outro de dados, bidirecional. A tabela 3.5 apresenta os pinos do PIC18F452/PIC18F4550 usados para I2C.

Sinal	Pino	Função
SDA	RB0	Dados seriais
SCL	RB1	Clock (provido pelo mestre)

Tabela 6.11: Pinos para operação em SPI

A conexão entre dispositivo I2C deve ser feita conforme o diagrama abaixo. Os resistores de pull-up são fundamentais, já que os drivers de saída de todos os dispositivos I2C são open-drain. Ao enviar dados um dispositivo I2C (mestre ou escravo) o que na verdade acontece é que nos bits '0' o dispositivo satura um transistor aterrando o barramento e nos bits '1' ele corta esse transistor, ficando o nível '1' garantido pelo resistor de pull-up. Dessa forma vários dispositivos podem trocar informação sem risco de dano.

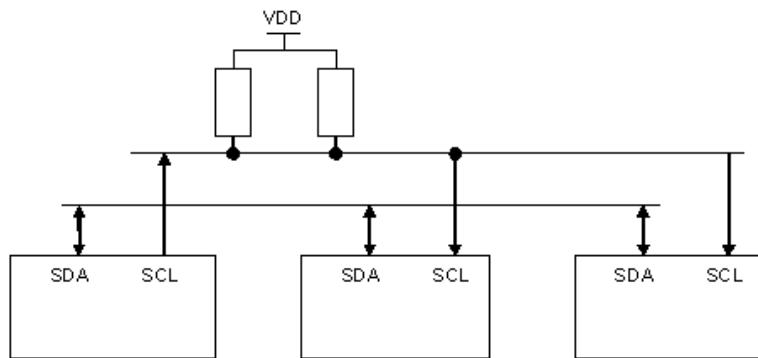


Figura 6.32: Diagrama em blocos para o modo I2C escravo.

No protocolo I2C os diversos dispositivos ligados a um mesmo barramento possuem endereços individuais. Todos os dispositivos da rede recebem os comandos enviados pelo mestre, mas somente aquele endereçado pelo mestre responde.

O protocolo I2C prevê uma série de comandos para controle de fluxo, informando início e fim da comunicação e sinalizando se a mensagem foi recebida corretamente. O protocolo prevê que a mudança de estado dos bits se dê no intervalo de tempo em que o sinal de clock permanece em '0' e que enquanto SCK permanece em '1' o sinal permaneça estável, e é quanto é feita a leitura. A figura abaixo ilustra essa definição.

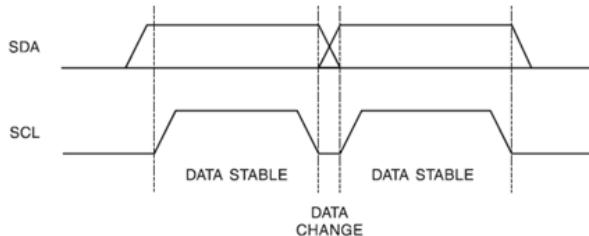


Figura 6.33: Protocolo I2C.

Para sinalizar início e fim de comunicação são usadas as condições de Start e Stop, que são quebras da regra acima. Quando SCK está em '1' e SDA transita de '1' para '0' caracteriza-se uma condição de Start. Quando SCK está em '1' e SDC transita de '0' para '1' caracteriza-se uma condição de Stop. Essas sinalizações são utilizadas para definir o início e fim da comunicação, de forma que os escravos sejam sincronizados com o mestre. Abaixo temos a ilustração dessas condições.

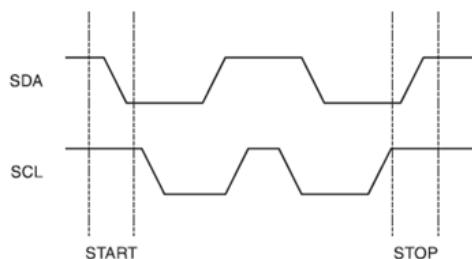


Figura 6.34: Condições de Start e Stop.

Além disso, a cada byte transmitido deve ser gerado pelo escravo que o recebe uma resposta

de confirmação (ACK, do inglês acknowledge). Essa condição é obtida quando o escravo força '0' em SDA no nono pulso de clock. Dependendo do protocolo do dispositivo pode ser gerando também um sinal de *no-acknowledge* (NACK), que é caracterizado pela permanência de SDA em '1' durante o nono bit. A figura abaixo representa essa sinalização, mas esteja atento ao fato que não existem os sinais DATA IN e DATA OUT, são apenas representações da origem do sinal em SDA (DATA IN é gerado pelo mestre e DATA OUT pelo escravo).

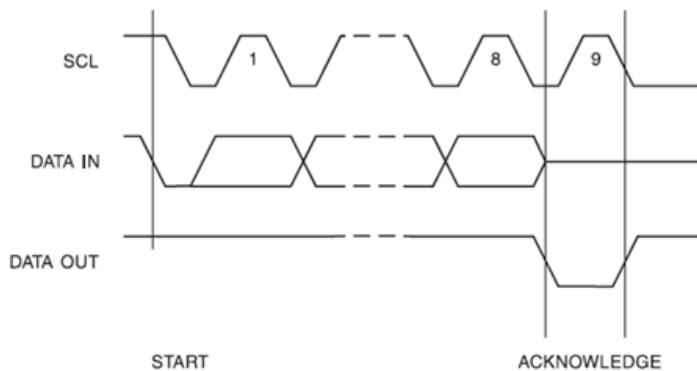


Figura 6.35: Sinal de *no-acknowledge* (NACK).

Uma típica operação de escrita (do mestre para o escravo) é apresentada na figura a seguir (observe quais bits são gerados pelo mestre e quais são pelo escravo) onde é apresentado apenas a sinalização de dados, ficando subentendido o sinal de clock. A comunicação inicia-se com um Start gerado pelo mestre. Em seguida o primeiro byte define o endereço do dispositivo a ser acessado. O oitavo bit define se é uma instrução de leitura ('1') ou escrita ('0'). No nono bit o escravo responde um ACK. Em seguida é enviado um byte definindo um endereço de memória dentro do dispositivo, que também é confirmado por um ACK. A partir daí cada byte enviado é armazenado em um endereço a partir do endereço inicial, até o mestre encerrar a comunicação através de um Stop.

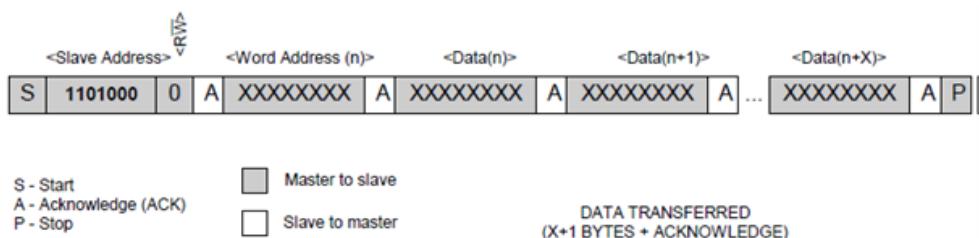


Figura 6.36: Start e Stop

A seguir é apresentado uma operação de leitura. O comando inicia-se com um Start, seguido do endereço do dispositivo (7 bits) e o oitavo bit em 1, indicando leitura. O escravo responde com ACK e passa a enviar seus dados sequencialmente. Note que sempre é o mestre que gera o sinal de clock, o escravo apenas envia os dados para o barramento. A cada byte enviado pelo escravo o mestre responde com um ACK. No último byte o mestre responde com um NACK, sinalizando o fim do envio de dados e finaliza a comunicação com um Stop.

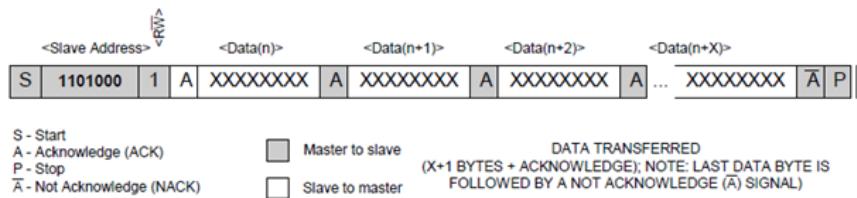


Figura 6.37: Operação de leitura.

Como deve ter sido observado, não foi especificado a partir de qual endereço o escravo deveria enviar seus dados. Isso porque, uma vez dado um comando de leitura o escravo passa a enviar seus dados para o mestre. Esses dados são enviados a partir do endereço atual, isto é, do ultimo endereço gravado. Para realizar a leitura a partir de um endereço específico o que se faz é iniciar um processo de escrita e reiniciar a comunicação, forçando um no Start, seguindo de um comando de leitura, conforme apresentado a seguir.

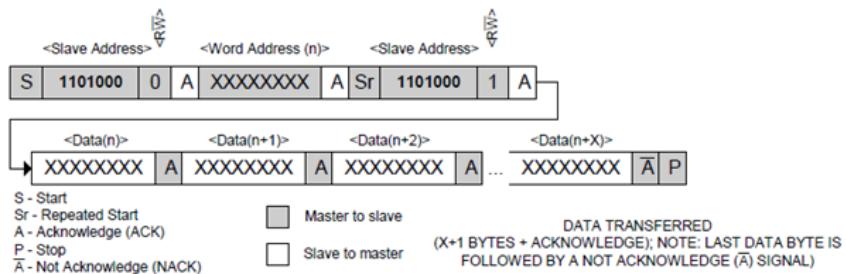


Figura 6.38: Operação de escrita

Operando em modo escravo, o PIC18 comporta-se como um dos elementos da rede, tratando apenas os comandos destinados ao seu endereço. A figura a seguir apresenta o diagrama em blocos do módulo MSSP operando em modo I²C escravo.

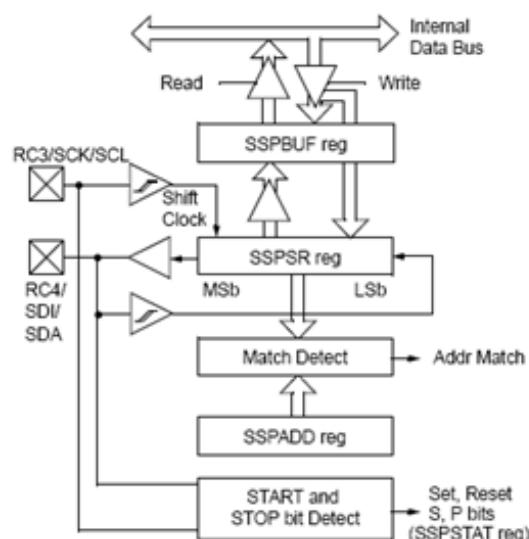


Figura 6.39: Diagrama em blocos para o modo I²C escravo

Operando em modo mestre, o microcontrolador deve gerar o clock para todos os dispositivos

escravos da rede. Além disso, ele deve gerar as condições de bit de partida (start bit) e bit de parada (stop bit) definidas pelo protocolo e verificar a bit de ACK (confirmação gerada pelo escravo). O diagrama em blocos do módulo MSSP operando em modo I²C mestre é mostrado na figura abaixo

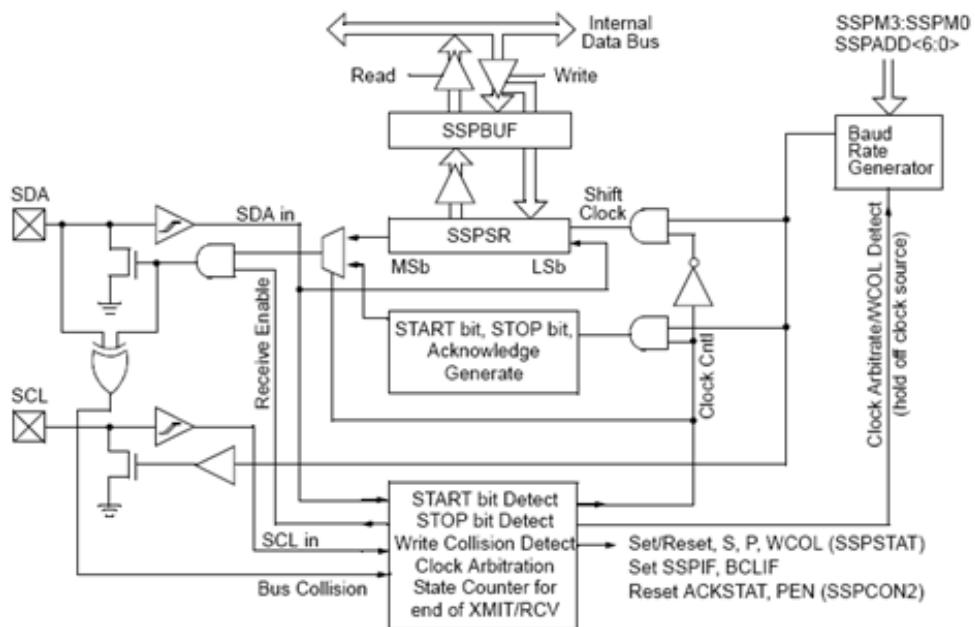


Figura 6.40: Diagrama em blocos para o modo I^2C mestre

No modo mestre o sinal de clock que controla a taxa de transmissão é configurado através dos sete bits menos significativos do registro SSPADD. O valor carregado nesse registro é decrementado duas vezes por ciclo de instrução. Quando chega a zero o estado do sinal de clock é invertido e o valor é recarregado para reiniciar a contagem de tempo.

Interrupção

No modo I²C escravo o módulo MSSP irá gerar uma interrupção toda vez que um comando for enviado para o endereço do microcontrolador. Além disso, pode-se configurar para que a interrupção ocorra toda vez que acontecer um bit de partida de um bit de parada.

No modo I²C mestre o flag de interrupção será acionado por qualquer um dos eventos abaixo:

- Bit de partida;
- Bit de parada;
- Transmissão ou recepção completa;
- Transmissão de ACK;
- Repetição de bit de partida.

Funções

Para uso do módulo MSSP para comunicação I²C existe a biblioteca I2C.H.

Função	Protótipo	Descrição
AckI2C	void AckI2C(void);	Gera condição de ACK
CloseI2C	void CloseI2C(void);	Desativa o módulo SSP
DataRdyI2C	unsigned char DataRdyI2C(void);	Verifica se existem novos dados no buffer de recepção.
getcI2C	unsigned char getcI2C (void);	Lê um byte
getsI2C	unsigned char getsI2C(unsigned char * rdptr, unsigned char length);	Lê uma sequencia de bytes (modo mestre apenas)
IdleI2C	void IdleI2C(void);	Fica em loop enquanto o barramento I2C está inativo.
NotAckI2C	void NotAckI2C(void);	Gera uma condição e NACK
OpenI2C	void OpenI2C(unsigned char sync_mode, unsigned char slew);	Configura o módulo MSSP para operação em modo I2C
putcI2C	unsigned char putcI2C(unsigned char data_out);	Escreve um byte
putsI2C	unsigned char putsI2C(unsigned char *wrptr);	Escreve uma sequência de bytes (modos mestre e escravo)
ReadI2C	unsigned char ReadI2C (void);	Lê um byte.
RestartI2C	void RestartI2C(void);	Gera uma condição de Restart
StartI2C	void StartI2C(void);	Gera uma condição de Start
StopI2C	void StopI2C(void);	Gera uma condição de Stop
WriteI2C	unsigned char WriteI2C(unsigned char data_out);	Escreve um dado

Tabela 6.12: Funções de I^2C

É muito comum o uso de memórias EEPROM seriais I2C. Em virtude disso, o C18 provê a biblioteca I2C.H funções específicas para comunicação com essas memórias. Essas funções são apresentadas na tabela a seguir.

Função	Protótipo	Descrição
EEAckPolling	unsigned char EEAckPolling; (unsigned char control)	Gera uma varredura de ACK
EEByteWritex	unsigned char EEByteWrite(unsigned char control, unsigned char address, unsigned char data);	Escreve um byte
EECurrentAddReadx	unsigned int EECurrentAddRead(unsigned char control);	Lê um dado do próximo endereço
EEPageWritex	unsigned char EEPAGEWrite(unsigned char control, unsigned char address, unsigned char * wrptr);	Escreve uma sequência de dados
EERandomRead	unsigned int EERandomRead(unsigned char control, unsigned char address);	Lê um dado de um endereço qualquer
EESequentialRead	unsigned char EESequentialRead(unsigned char control, unsigned char address, unsigned char * rdptr, unsigned char length);	Lê uma sequência de dados

 Tabela 6.13: Funções da EEPROM I^2C

6.8 USB

O protocolo USB foi inicialmente desenvolvido para ser uma forma prática de conexão de periféricos a computadores (como mouse, teclado e impressoras), em substituições a portas seriais e paralelas. Esse padrão evoluiu e hoje é o padrão de comunicação pra PCs, periféricos e dispositivos pessoais (iPOD, câmeras digitais, etc..).

O protocolo USB apresenta diversas vantagens, das quais podemos citar:

- Altas taxas de transmissão
- Conexão Plug&Play
- Alimentação dos dispositivos através do cabo de comunicação.
- Conexão de diversos periféricos ao mesmo tempo
- Baixo custo

Atualmente estão em uso dois padrões, o USB 1.0 e o USB 2.0. O padrão USB 1.0 suporta as velocidades de 1,5 Mbps (USB-LS - Low Speed) e 12Mbps (USB-FS - Full Speed). O padrão USB 2.0 suporta ainda a velocidade de 480 Mbps (USB-HS - High Speed), apesar de um dispositivo não ter que suportar HS para ser considerado compatível com USB 2.0.

A comunicação USB é uma comunicação Mestre-Escravo, isto é, um dispositivo principal (Mestre ou host) comanda um rede de periféricos (escravos ou devices). Portanto é impossível a comunicação entre 2 devices (por exemplo, um mouse não pode se comunicar com um pen-drive).

Existe ainda dispositivos OTG, que podem se comportar como Host e como escravo. Abaixo uma lista dos tipos de dispositivos USB.

- USB Host: Gerencia o barramento, iniciando todas as comunicações e alimentando os dispositivos. Tipicamente requer uma alta capacidade de processamento e deve possuir drivers para as classes de dispositivos aos quais pode se conectar;
- USB Device: Responde aos comandos do Host e é alimentado por ele;
- USB OTG (On-The-Go): é ao mesmo tempo mestre e escravo. Suporta apenas uma conector Mini-AB para conexão com Host (Mini A) e Device (Mini B);
- Embedded Host: opera como Host com algumas funcionalidades limitadas;
- Hub USB: permitem a expansão de portas USB. Permitem até 127 elementos na rede, sendo que pode-se usar no máximo 5 hubs em cascata.

A figura abaixo ilustra os tipos de conectores que definem ao tipo de elemento que se deve conectar.

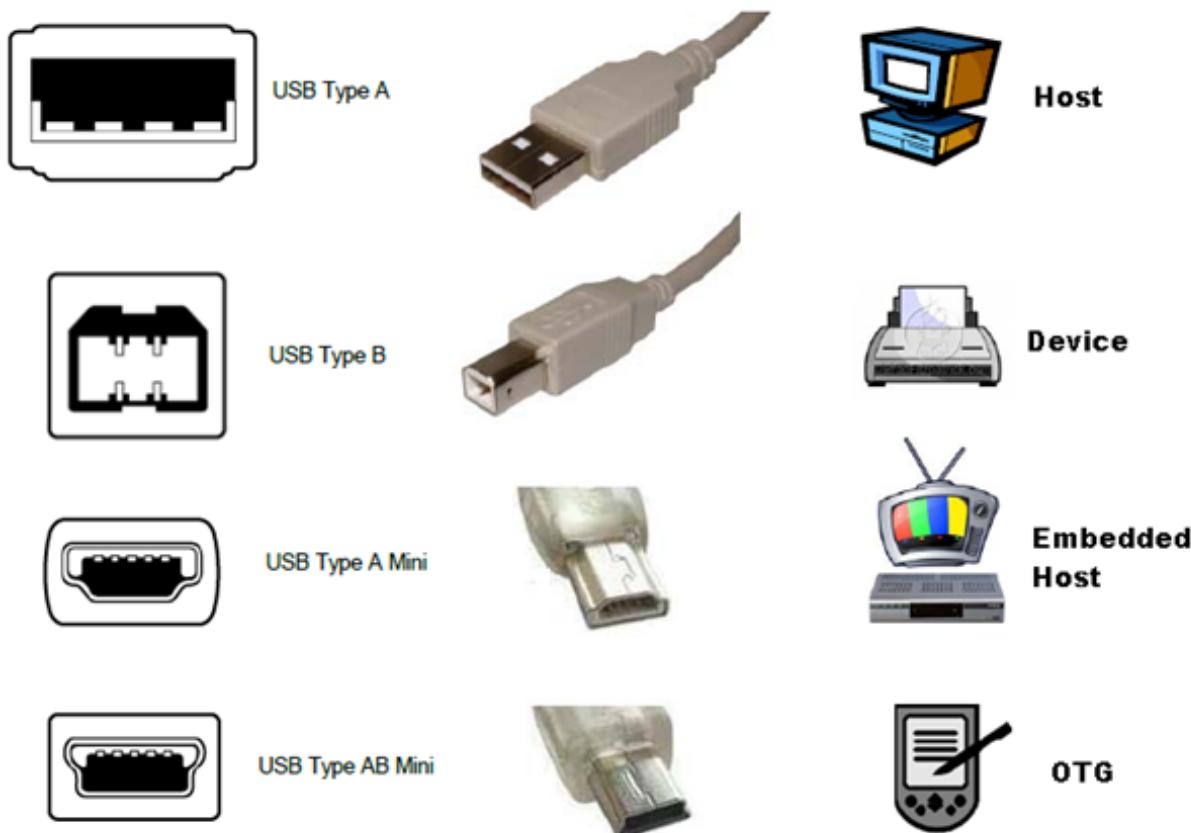


Figura 6.41: Tipos de conectores

O padrão USB prevê diversas classes de dispositivos. Dessa forma, dispositivos de funcionamento semelhante são tratados de forma padronizada. Por exemplo, pen-drives de diversos fabricantes e capacidades são tratados da mesma forma, pois pertencem à classe Mass Storage. A seguir são listadas as classes de dispositivos suportadas:

- Áudio
- Chip/Smart Card Interface
- Communication Device Class(CDC)
- Content security
- Device Firmware update
- Humam Interface (HID)
- IrDA Bridge
- Mass Storage (MSD)
- Printer
- Still Image Capture
- Test and measurement
- Vídeo

Quando um dispositivo não se enquadra nessas classes ele é considerado de uma classe customizada (custom) o que significa escrever um driver sem pertencer a qualquer padrão definido.

O PIC18F4550 possui um periférico de USB Device compatível com a norma USB 2.0. Ele pode operar em full-speed (12MHz) ou low-speed (1,5MHz). Internamente ao PIC existe um regulador de 3,3V, responsável por fornecer a esse componente a tensão necessária para o driver de saída. Existe suporte a resistores de pull-up (que indicam ao host se o dispositivo é full-speed ou low-speed) externo, assim como possibilidade de uso de um transceptor externo. Contudo, esses dispositivos estão disponíveis internamente, não necessitando de nenhum componente externo além do conector B.

O protocolo USB, apesar da praticidade, não é algo trivial. Sua implementação não é simples como a de um periférico como timer ou A/D, exigindo o uso de uma biblioteca de funções complexas. Nesse sentido, a forma mais fácil pra se trabalhar com USB é a partir de um exemplo pronto para a classe de dispositivo que se pretende usar, na qual serão feitas as alterações para nossa aplicação.

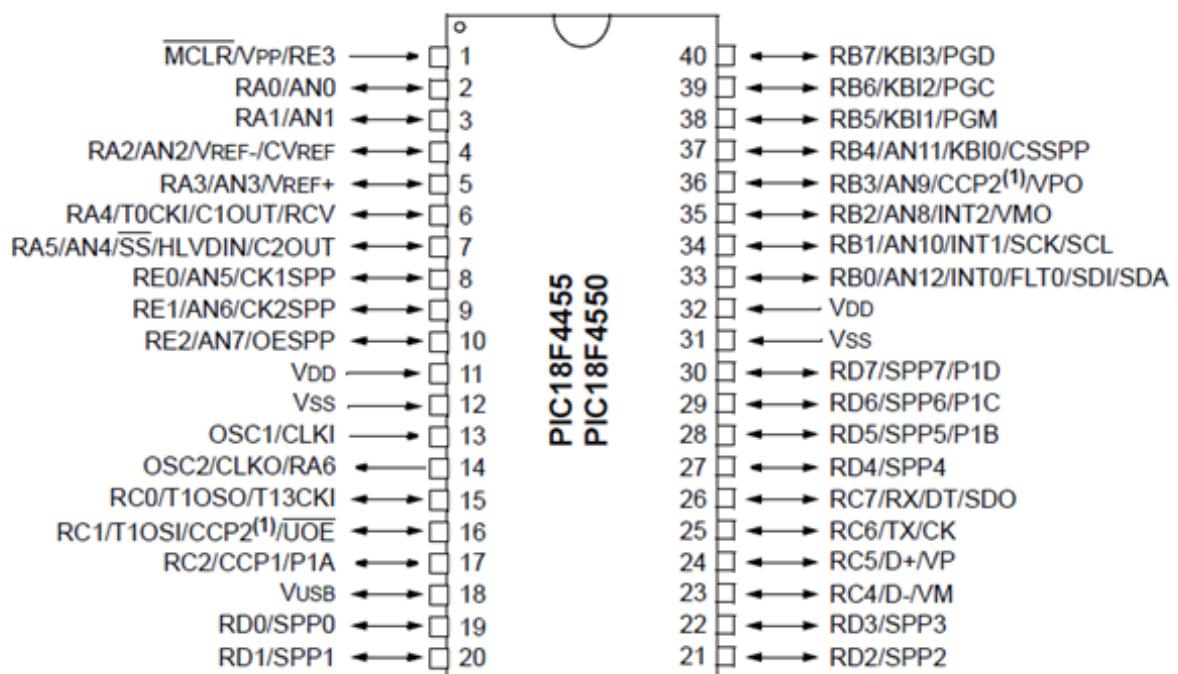
A Microchip provê essas bibliotecas, bem como drivers para os tipos básicos de dispositivos e extensa documentação. Todo esse material está disponível no CD que acompanha o kit em www.microchip.com/usb.

Mais ainda, o kit XM118 é compatível com a placa de demonstração PICDEM FS-USB (com exceção do sensor de temperatura I2C, não presente no kit), portanto diversos exemplos e aplicativos desenvolvidos para essa placa da Microchip serão executados no kit.

Capítulo 7

Anexos

7.1 Anexo A - Distribuição de Pinos do 18F4550



7.2 Anexo B - Registros de função especial do PIC18F4550

Address	Name	Address	Name	Address	Name	Address	Name	Address	Name
FFFh	TOSU	FDFh	INDF2 ⁽¹⁾	FBFh	CCPR1H	F9Fh	IPR1	F7Fh	UEP15
FFEh	TOSH	FDEh	POSTINC2 ⁽¹⁾	FBEh	CCPR1L	F9Eh	PIR1	F7Eh	UEP14
FFDh	TOSL	FDDh	POSTDEC2 ⁽¹⁾	FBDh	CCP1CON	F9Dh	PIE1	F7Dh	UEP13
FFCh	STKPTR	FDCh	PREINC2 ⁽¹⁾	FBCh	CCPR2H	F9Ch	— ⁽²⁾	F7Ch	UEP12
FFBh	PCLATU	FDBh	PLUSW2 ⁽¹⁾	FBBh	CCPR2L	F9Bh	OSCTUNE	F7Bh	UEP11
FFAh	PCLATH	FDAh	FSR2H	FBAh	CCP2CON	F9Ah	— ⁽²⁾	F7Ah	UEP10
FF9h	PCL	FD9h	FSR2L	FB9h	— ⁽²⁾	F99h	— ⁽²⁾	F79h	UEP9
FF8h	TBLPTRU	FD8h	STATUS	FB8h	BAUDCON	F98h	— ⁽²⁾	F78h	UEP8
FF7h	TBLPTRH	FD7h	TMR0H	FB7h	ECCP1DEL	F97h	— ⁽²⁾	F77h	UEP7
FF6h	TBLPTRL	FD6h	TMR0L	FB6h	ECCP1AS	F96h	TRISE ⁽³⁾	F76h	UEP6
FF5h	TABLAT	FD5h	T0CON	FB5h	CVRCON	F95h	TRISD ⁽³⁾	F75h	UEP5
FF4h	PRODH	FD4h	— ⁽²⁾	FB4h	CMCON	F94h	TRISC	F74h	UEP4
FF3h	PRODL	FD3h	OSCCON	FB3h	TMR3H	F93h	TRISB	F73h	UEP3
FF2h	INTCON	FD2h	HLVDCON	FB2h	TMR3L	F92h	TRISA	F72h	UEP2
FF1h	INTCON2	FD1h	WDTCON	FB1h	T3CON	F91h	— ⁽²⁾	F71h	UEP1
FF0h	INTCON3	FD0h	RCON	FB0h	SPBRGH	F90h	— ⁽²⁾	F70h	UEP0
FEFh	INDF0 ⁽¹⁾	FCFh	TMR1H	FAFh	SPBRG	F8Fh	— ⁽²⁾	F6Fh	UCFG
FEEh	POSTINC0 ⁽¹⁾	FCEh	TMR1L	FAEH	RCREG	F8Eh	— ⁽²⁾	F6Eh	UADDR
FEDh	POSTDEC0 ⁽¹⁾	FCDh	T1CON	FADh	TXREG	F8Dh	LATE ⁽³⁾	F6Dh	UCON
FECh	PREINC0 ⁽¹⁾	FCCh	TMR2	FACH	TXSTA	F8Ch	LATD ⁽³⁾	F6Ch	USTAT
FEBh	PLUSW0 ⁽¹⁾	FCBh	PR2	FABh	RCSTA	F8Bh	LATC	F6Bh	UEIE
FEAh	FSR0H	FCAh	T2CON	FAAH	— ⁽²⁾	F8Ah	LATB	F6Ah	UEIR
FE9h	FSR0L	FC9h	SSPBUF	FA9h	EEADR	F89h	LATA	F69h	UIE
FE8h	WREG	FC8h	SSPADD	FA8h	EEDATA	F88h	— ⁽²⁾	F68h	UIR
FE7h	INDF1 ⁽¹⁾	FC7h	SSPSTAT	FA7h	EECON2 ⁽¹⁾	F87h	— ⁽²⁾	F67h	UFRMH
FE6h	POSTINC1 ⁽¹⁾	FC6h	SSPCON1	FA6h	EECON1	F86h	— ⁽²⁾	F66h	UFRML
FE5h	POSTDEC1 ⁽¹⁾	FC5h	SSPCON2	FA5h	— ⁽²⁾	F85h	— ⁽²⁾	F65h	SPPCON ⁽³⁾
FE4h	PREINC1 ⁽¹⁾	FC4h	ADRESH	FA4h	— ⁽²⁾	F84h	PORTE	F64h	SPPEPS ⁽³⁾
FE3h	PLUSW1 ⁽¹⁾	FC3h	ADRESL	FA3h	— ⁽²⁾	F83h	PORTD ⁽³⁾	F63h	SPPCFG ⁽³⁾
FE2h	FSR1H	FC2h	ADCON0	FA2h	IPR2	F82h	PORTC	F62h	SPPDATA ⁽³⁾
FE1h	FSR1L	FC1h	ADCON1	FA1h	PIR2	F81h	PORTB	F61h	— ⁽²⁾
FE0h	BSR	FC0h	ADCON2	FA0h	PIE2	F80h	PORTA	F60h	— ⁽²⁾

Note 1: Not a physical register.

2: Unimplemented registers are read as '0'.

3: These registers are implemented only on 40/44-pin devices.

7.3 Anexo C - Leituras Recomendadas

Os assuntos abordados nessa apostila são bastante abrangentes e existem diversos documentos e livros que podem complementar o assunto.

O CD que acompanha a apostila trás diversos manuais que tratam do PIC18, como do compilador C18. Esses documentos foram tirados do site www.microchip.com. Nestes documentos, parte-se do princípio de que o leitor tem sólidos conhecimentos de microcontroladores e programação em C e os assuntos são tratados de forma bastante detalhada. São recomendados como material avançado para quem pretende se aprofundar mais nos conhecimentos apresentados no curso.

Bons livros existentes no mercado tratam da linguagem C. A princípio, qualquer livro que trate de C ANSI "puro", isto é, que não trate de C++ ou implementações visuais como Visual C/C++ pode ser usado para um maior estudo da linguagem, seja para uma melhor compreensão de suas características, seja para um estudo mais aprofundado. Uma leitura recomendada para quem tem pouca experiência com a linguagem C é:

- Mizrahi, Victorine Viviane. Treinamento em Linguagem C - Módulo 1. Makron Books, São Paulo.

Para programadores de nível intermediário e/ou avançado, é recomendada a leitura da referência [5], da qual um dos autores é ninguém menos que o próprio criador do C, Dennis Ritchie.

Por fim, mais informações sobre microcontroladores de uma forma geral e microcontroladores da família PIC16 podem ser obtidas na referência [1].

7.4 Anexo D - Bibliografia:

- [1] Adriano, J. D. (2005). Curso de microcontroladores PIC16Fxxx - Módulo Básico. Exsto Tecnologia. Santa Rita do Sapucaí
- [2] Kernighan, Brian W. Ritchie, Dennis M. (1988). The C programming language. Editora Prentice-Hall PTR
- [3] Shultz, Thomas W. (1998). C and 8051: hardware, modular programming and multitasking. Vol. I e II. 2a. edição. Ed. Prentice Hall PTR.
- [4] Pereira, Fábio. (2003) Microcontroladores PIC - Programação em C. Editora Érica, São Paulo.
- [5] PIC18F2420/2520/4420/4550 DataSheet
- [6] PIC18F2455/2550/4455/4550 DataSheet
- [7] MPLAB® C18 C COMPILER USER'S GUIDE
- [8] MPLAB® C18 C COMPILER LIBRARIES

7.5 Anexo E - Conjunto de instruções do PIC18

Convenções:

Field	Description
a	RAM access bit a = 0: RAM location in Access RAM (BSR register is ignored) a = 1: RAM bank is specified by BSR register
bbb	Bit address within an 8-bit file register (0 to 7).
BSR	Bank Select Register. Used to select the current RAM bank.
C, DC, Z, OV, N	ALU Status bits: Carry, Digit Carry, Zero, Overflow, Negative.
d	Destination select bit d = 0: store result in WREG d = 1: store result in file register f
dest	Destination: either the WREG register or the specified register file location.
f	8-bit Register file address (00h to FFh) or 2-bit FSR designator (0h to 3h).
f _s	12-bit Register file address (000h to FFFh). This is the source address.
f _d	12-bit Register file address (000h to FFFh). This is the destination address.
GIE	Global Interrupt Enable bit.
k	Literal field, constant data or label (may be either an 8-bit, 12-bit or a 20-bit value).
label	Label name.
mm	The mode of the TBLPTR register for the table read and table write instructions. Only used with table read and table write instructions: * No change to register (such as TBLPTR with table reads and writes) *+ Post-Increment register (such as TBLPTR with table reads and writes) *- Post-Decrement register (such as TBLPTR with table reads and writes) +* Pre-Increment register (such as TBLPTR with table reads and writes)
n	The relative address (2's complement number) for relative branch instructions or the direct address for Call/Branch and Return instructions.
PC	Program Counter.
PCL	Program Counter Low Byte.
PCH	Program Counter High Byte.
PCLATH	Program Counter High Byte Latch.
PCLATU	Program Counter Upper Byte Latch.
PD	Power-down bit.
PRODH	Product of Multiply High Byte.
PRODL	Product of Multiply Low Byte.
s	Fast Call/Return mode select bit s = 0: do not update into/from shadow registers s = 1: certain registers loaded into/from shadow registers (Fast mode)
TBLPTR	21-bit Table Pointer (points to a Program Memory location).
TABLAT	8-bit Table Latch.
TO	Time-out bit.
TOS	Top-of-Stack.
u	Unused or unchanged.
WDT	Watchdog Timer.
WREG	Working register (accumulator).
x	Don't care ('0' or '1'). The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all Microchip software tools.
z _s	7-bit offset value for indirect addressing of register files (source).
z _d	7-bit offset value for indirect addressing of register files (destination).
{ }	Optional argument.
[text]	Indicates an indexed address.
(text)	The contents of text.
[expr]<n>	Specifies bit n of the register indicated by the pointer expr.
→	Assigned to.
< >	Register bit field.
∈	In the set of.
italics	User defined term (font is Courier).

Figura 7.1: Convenções do PIC 18F

Conjunto de instruções:

Mnemonic, Operands	Description	Cycles	16-Bit Instruction Word				Status Affected	Notes	
			MSb	Lsb					
BYTE-ORIENTED OPERATIONS									
ADDWF f, d, a	Add WREG and f	1	0010 01da0	ffff	ffff	C, DC, Z, OV, N		1, 2	
ADDWFC f, d, a	Add WREG and CARRY bit to f	1	0010 0da	ffff	ffff	C, DC, Z, OV, N		1, 2	
ANDWF f, d, a	AND WREG with f	1	0001 01da	ffff	ffff	Z, N		1, 2	
CLRF f, a	Clear f	1	0110 101a	ffff	ffff	Z		2	
COMF f, d, a	Complement f	1	0001 11da	ffff	ffff	Z, N		1, 2	
CPFSEQ f, a	Compare f with WREG, skip =	1 (2 or 3)	0110 001a	ffff	ffff	None		4	
CPFSGT f, a	Compare f with WREG, skip >	1 (2 or 3)	0110 010a	ffff	ffff	None		4	
CPFSLT f, a	Compare f with WREG, skip <	1 (2 or 3)	0110 000a	ffff	ffff	None		1, 2	
DECWF f, d, a	Decrement f	1	0000 01da	ffff	ffff	C, DC, Z, OV, N		1, 2, 3, 4	
DECFSZ f, d, a	Decrement f, Skip if 0	1 (2 or 3)	0010 11da	ffff	ffff	None		1, 2, 3, 4	
DCFSNZ f, d, a	Decrement f, Skip if Not 0	1 (2 or 3)	0100 11da	ffff	ffff	None		1, 2	
INCF f, d, a	Increment f	1	0010 10da	ffff	ffff	C, DC, Z, OV, N		1, 2, 3, 4	
INCFSZ f, d, a	Increment f, Skip if 0	1 (2 or 3)	0011 11da	ffff	ffff	None		4	
INFSNZ f, d, a	Increment f, Skip if Not 0	1 (2 or 3)	0100 10da	ffff	ffff	None		1, 2	
IORWF f, d, a	Inclusive OR WREG with f	1	0001 00da	ffff	ffff	Z, N		1, 2	
MOVWF f, d, a	Move f	1	0101 00da	ffff	ffff	Z, N		1	
MOVFF f _s , f _d	Move f _s (source) to 1st word f _d (destination) 2nd word	2	1100 ffff	ffff	ffff	None			
			1111 ffff	ffff	ffff				
MOVWF f, a	Move WREG to f	1	0110 111a	ffff	ffff	None			
MULWF f, a	Multiply WREG with f	1	0000 001a	ffff	ffff	None		1, 2	
NEGF f, a	Negate f	1	0110 110a	ffff	ffff	C, DC, Z, OV, N			
RLCF f, d, a	Rotate Left f through Carry	1	0011 01da	ffff	ffff	C, Z, N		1, 2	
RLNCF f, d, a	Rotate Left f (No Carry)	1	0100 01da	ffff	ffff	Z, N			
RRCF f, d, a	Rotate Right f through Carry	1	0011 00da	ffff	ffff	C, Z, N			
RRNCF f, d, a	Rotate Right f (No Carry)	1	0100 00da	ffff	ffff	Z, N			
SETF f, a	Set f	1	0110 100a	ffff	ffff	None		1, 2	
SUBFWB f, d, a	Subtract f from WREG with borrow	1	0101 01da	ffff	ffff	C, DC, Z, OV, N			
SUBWF f, d, a	Subtract WREG from f	1	0101 11da	ffff	ffff	C, DC, Z, OV, N		1, 2	
SUBWFB f, d, a	Subtract WREG from f with borrow	1	0101 10da	ffff	ffff	C, DC, Z, OV, N			
SWAPF f, d, a	Swap nibbles in f	1	0011 10da	ffff	ffff	None		4	
TSTFSZ f, a	Test f, skip if 0	1 (2 or 3)	0110 011a	ffff	ffff	None		1, 2	
XORWF f, d, a	Exclusive OR WREG with f	1	0001 10da	ffff	ffff	Z, N			

Figura 7.2: Conjunto de instruções do Pic

Mnemonic, Operands	Description	Cycles	16-Bit Instruction Word				Status Affected	Notes	
			MSb	Lsb					
LITERAL OPERATIONS									
ADDLW k	Add literal and WREG	1	0000 1111	kkkk	kkkk	C, DC, Z, OV, N			
ANDLW k	AND literal with WREG	1	0000 1011	kkkk	kkkk	Z, N			
IORLW k	Inclusive OR literal with WREG	1	0000 1001	kkkk	kkkk	Z, N			
LFSR f, k	Move literal (12-bit) 2nd word to FSR(f) 1st word	2	1110 1110	00ff	kkkk	None			
MOVLB k	Move literal to BSR<3:0>	1	1111 0000	kkkk	kkkk	None			
MOVLW k	Move literal to WREG	1	0000 0001	0000	kkkk	None			
MULLW k	Multiply literal with WREG	1	0000 1101	kkkk	kkkk	None			
RETLW k	Return with literal in WREG	2	0000 1100	kkkk	kkkk	None			
SUBLW k	Subtract WREG from literal	1	0000 1000	kkkk	kkkk	C, DC, Z, OV, N			
XORLW k	Exclusive OR literal with WREG	1	0000 1010	kkkk	kkkk	Z, N			
DATA MEMORY ↔ PROGRAM MEMORY OPERATIONS									
TBLRD*	Table Read	2	0000 0000	0000	1000	None			
TBLRD*+	Table Read with post-increment		0000 0000	0000	1001	None			
TBLRD*-	Table Read with post-decrement		0000 0000	0000	1010	None			
TBLRD**	Table Read with pre-increment		0000 0000	0000	1011	None			
TBLWT*	Table Write	2	0000 0000	0000	1100	None			
TBLWT*+	Table Write with post-increment		0000 0000	0000	1101	None			
TBLWT*-	Table Write with post-decrement		0000 0000	0000	1110	None			
TBLWT**	Table Write with pre-increment		0000 0000	0000	1111	None			

Figura 7.3: Conjunto de instruções do Pic(segunda parte)

Mnemonic, Operands	Description	Cycles	16-Bit Instruction Word				Status Affected	Notes	
			MSb		Lsb				
BIT-ORIENTED OPERATIONS									
BCF	f, b, a	Bit Clear f	1	1001	bbba	ffff	ffff	None	1, 2
BSF	f, b, a	Bit Set f	1	1000	bbba	ffff	ffff	None	1, 2
BTFS	f, b, a	Bit Test f, Skip if Clear	1 (2 or 3)	1011	bbba	ffff	ffff	None	3, 4
BTFS	f, b, a	Bit Test f, Skip if Set	1 (2 or 3)	1010	bbba	ffff	ffff	None	3, 4
BTG	f, d, a	Bit Toggle f	1	0111	bbba	ffff	ffff	None	1, 2
CONTROL OPERATIONS									
BC	n	Branch if Carry	1 (2)	1110	0010	nnnn	nnnn	None	
BN	n	Branch if Negative	1 (2)	1110	0110	nnnn	nnnn	None	
BNC	n	Branch if Not Carry	1 (2)	1110	0011	nnnn	nnnn	None	
BNN	n	Branch if Not Negative	1 (2)	1110	0111	nnnn	nnnn	None	
BNOV	n	Branch if Not Overflow	1 (2)	1110	0101	nnnn	nnnn	None	
BNZ	n	Branch if Not Zero	1 (2)	1110	0001	nnnn	nnnn	None	
BOV	n	Branch if Overflow	1 (2)	1110	0100	nnnn	nnnn	None	
BRA	n	Branch Unconditionally	2	1101	0nnn	nnnn	nnnn	None	
BZ	n	Branch if Zero	1 (2)	1110	0000	nnnn	nnnn	None	
CALL	n, s	Call subroutine 1st word	2	1110	110s	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
CLRWDT	—	Clear Watchdog Timer	1	0000	0000	0000	0100	TO, PD	
DAW	—	Decimal Adjust WREG	1	0000	0000	0000	0111	C	
GOTO	n	Go to address 1st word	2	1110	1111	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
NOP	—	No Operation	1	0000	0000	0000	0000	None	
NOP	—	No Operation	1	1111	xxxx	xxxx	xxxx	None	
POP	—	Pop top of return stack (TOS)	1	0000	0000	0000	0110	None	
PUSH	—	Push top of return stack (TOS)	1	0000	0000	0000	0101	None	
RCALL	n	Relative Call	2	1101	1nnn	nnnn	nnnn	None	
RESET		Software device Reset	1	0000	0000	1111	1111	All	
RETFIE	s	Return from interrupt enable	2	0000	0000	0001	000s	GIE/GIEH, PEIE/GIEL	
RETLW	k	Return with literal in WREG	2	0000	1100	kkkk	kkkk	None	
RETURN	s	Return from Subroutine	2	0000	0000	0001	001s	None	
SLEEP	—	Go into Standby mode	1	0000	0000	0000	0011	TO, PD	

Figura 7.4: Conjunto de instruções do Pic(terceira parte)

Notas:

- Quando uma operação que altera o valor de registro PORT tem como destino o próprio registro PORT o valor considerado é aquele lido diretamente dos terminais do microcontrolador;
- Quando essa instrução é executada no registro TMR0 (e d = 1, se aplicável) a pré-escala é zerada;
- Quando o PC é alterado, gerando um desvio no fluxo do programa, ou um teste condicional resulta verdadeiro, são gastos dois ciclos de máquina.
- Algumas instruções ocupam 2 endereços de memória. Elas são executadas em dois ciclos de instrução e o segundo ciclo tem o efeito de um NOP.

CADERNO DE EXPERIÊNCIAS

*Eu escuto e esqueço. Vejo e lembro. Faço e entendo...
-Confúcio*

Capítulo 8

Aulas Práticas

Este caderno de experiências trás exercícios teóricos e práticos sobre microcontroladores PIC18. As aulas estão distribuídas conforme o plano de aulas apresentada na Orientação Pedagógica, na introdução desse documento. Lá são indicados os conteúdos teóricos referentes a cada aula. Portanto, é necessário verificar qual a sequência de aulas conforme o objetivo e a carga horária do curso.

As três primeiras aulas são puramente teóricas, trazendo questões sobre o conceito de microcontroladores e do PIC18F4550. A partir da aula 4 todas apresentam algum exercício prático.

Todas as aulas são organizadas seguindo a estrutura didática abaixo, quando aplicável cada um dos itens:

- Objetivos: quais os objetivos didáticos a serem atingidos na aula;
- Introdução: quando necessário, uma breve introdução teórica ao assunto da aula;
- Questionário: algumas questões teóricas fundamentais para entender a experiência;
- Exercícios: as experiências propriamente ditas. Muitas vezes são apresentados exercícios resolvidos para análise e propondo alterações. Fica a cargo do instrutor usá-los como exemplos ou propor que sejam desenvolvidos pelos alunos.
- Exercícios propostas: exercícios que complementam a prática realizada. Os exercícios propostos podem ser realizados na própria aula, havendo tempo hábil para isso ou utilizado como lição de casa, fazendo-se uso das ferramentas de simulação.

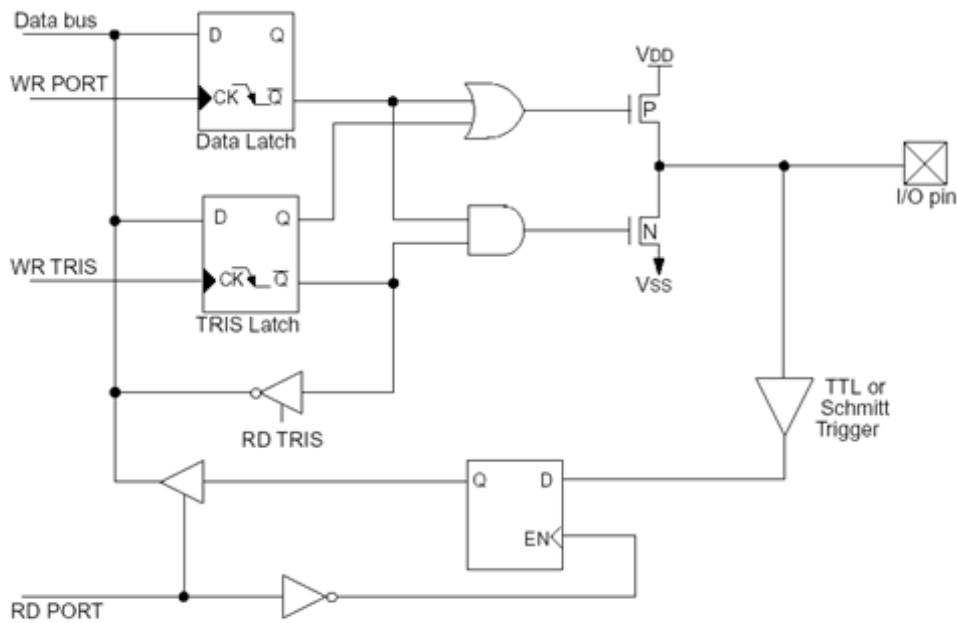
8.1 Aula 1 - Introdução aos microcontroladores

Objetivos

- Estudar os conceitos básicos de microcontroladores.

Questionário

1. Qual a principal característica dos sistemas computacionais, que os diferencia dos demais circuitos digitais?
2. Qual a diferença entre memória de programa e memória de dados?
3. O que são registros?
4. O que são vetores na memória de programa?
5. Descreva os vários tipos de memória de programa.
6. Qual a diferença entre memórias volátil e não-volátil?
7. Para que servem os portais?
8. Por que são utilizados barramentos para interligar as várias partes de um sistema computacional?
9. Descreva a função dos barramentos de:
 - (a) Dados;
 - (b) endereços;
 - (c) controle.
10. O que depende do tamanho (número de bits) do barramento de endereços?
11. Qual a quantidade máxima de memória pode ser acessada por barramentos de endereços de:
 - (a) 7 bits;
 - (b) 12 bits;
 - (c) 16 bits;
 - (d) 21 bits;
12. O que são periféricos?
13. Descreva, de forma rápida, o funcionamento do buffer tri-state e dos latch.
14. Em um pino de I/O (como o apresentado a seguir) o que ocorre se for realizada uma operação de escrita em um pino configurado como saída? Explique porque tendo em vista o circuito.



Note: I/O pin has protection diodes to VDD and Vss.

15. Explique a função dos seguintes blocos componentes da CPU:

- Decodificador de instruções (Instruction Decoder)
- Registro Status
- PC (Program Counter)
- Pilha (Stack)
- Unidade lógico-aritmética (ALU)

16. Qual a principal característica da arquitetura Harvard? O que se ganha com isso?

17. O que significam as siglas CISC e RISC?

18. Explique a diferença entre um microcontrolador e um microprocessador.

Exercícios propostos

- Propomos aqui uma pesquisa por manuais de componentes de diversos tipos de microcontrolador. O objetivo é familiarizar o aluno com os manuais destes componentes e exercitá-lo na atividade de busca de informações.

1. Pesquise na internet e faça download do manual dos seguintes componentes:

- PIC16F628A (Microchip)
- PIC18F4550 (Microchip)
- PIC24FJ128GA108 (Microchip)
- AT89S52 (Atmel)
- MC68HC908QY4 (Freescale)
- MSP430F2011(Texas)

2. Procure o diagrama em blocos de cada componente destes, procurando identificar:

- (a) Core (CPU)
- (b) Barramentos
- (c) Memórias
- (d) Portais de I/O
- (e) Periféricos

3. Baseado nas informações desses manuais, preencha a tabela de características abaixo:

Microcontrolador	Memória Flash [kB]	Memória RAM [B]	Freqüência máxima [MHz]	Número de pinos	Barramentos de dados [bits]
PIC16F628A					
PIC18F4550					
PIC24FJ128AG108					
AT89S52					
MC68HC908QY4					
MSP430F2011					

4. Quais dos componentes acima são CISC e quais são RISC?

8.2 Aula 2 - O PIC18F4550

Objetivos

- Estudar o hardware do microcontrolador PIC18F4550
- Conhecer os modelos de memória para o PIC18

Questionário

1. Quantos bits têm o barramento de instruções do PIC18?
2. Quantos bits têm o barramento de endereço para memória de programa do PIC18? Quantos endereços podem ser acessados então? Quantos bytes estão realmente implementados no PIC18F4550?
3. Qual a diferença entre as instruções RCALL e CALL? E entre BRA e GOTO?
4. Quantos bits têm o barramento de endereço para memória de dados do PIC18? Quantos endereços podem ser acessados então? Quantos bytes estão realmente implementados no PIC18F4550?
5. Explique o que são SFR e GPR.
6. O que é o banco de acesso?
7. Explique, resumidamente, como interagem os registro TRISB, LATB e PORTB.
8. Em quantos ciclos de clock é realizado cada instrução do PIC18? Quais são esses ciclos e qual ação acontece em cada um deles?
9. Descreva o que ocorre com os registros PCLATU e PCLATH quando o registro PCL é lido. E quando se escreve em PCL, o que ocorre?
10. O que fazem a instrução MULLW e MULWF?
11. Quantos níveis têm a pilha do PIC18? Quais eventos associados a pilha causam reset?
12. Com base no funcionamento do processador, sabendo quantos ciclos de clock cada instrução leva para ser executada, calcule quanto é o tempo de execução de uma instrução para as frequências de clock a seguir:
 - (a) 4MHz
 - (b) 20MHz
 - (c) 48MHz
 - (d) 3,58MHz
13. Explique a função dos bits abaixo, pertencentes ao registro status.

8.3 Aula 3 - Assembly do PIC18

Objetivos

- Estudar o conjunto de instruções do PIC18.

Questionário

1. Quantas instruções têm o PIC18?
2. Explique, de forma resumida, o que fazem as instruções abaixo. Para isso, consulte o manual do PIC18 (capítulo 26 - Instruction Set Summary).
 - (a) MOVLW
 - (b) MOVWF
 - (c) MOVFF
 - (d) ADDWF
 - (e) SUBLW
 - (f) ANDWF
 - (g) XORLW
 - (h) INCF
 - (i) RLCF
 - (j) RLNCF
 - (k) BSF
 - (l) BTG
 - (m) BZ
 - (n) BNZ
 - (o) BRA
 - (p) CALL
 - (q) GOTO
 - (r) RETURN
 - (s) SLEEP

8.4 Aula 4 - Ferramentas de desenvolvimento

Objetivos

- Aprender a criar um projeto usando MPLAB e C18
- Compilar, simular, depurar e gravar o projeto criado.

Introdução

As ferramentas de software utilizadas para o desenvolvimento com microcontroladores são de grande ajuda na criação e depuração de projetos. Ter boas ferramentas de desenvolvimento é quase tão importante com ter um bom microcontrolador para o sucesso de qualquer projeto. O objetivo dessa aula prática é passar por todo o processo de desenvolvimento, criando um novo projeto, compilando, simulando e, por fim, gravando e depurando no kit.

Questionário

1. O que é um IDE?
2. Qual a função da Janela Watch?
3. Para que serve o Simulator Logic Analyzer?
4. O que faz um estímulo com configurado como Toggle?

textcolor[rgb]0.16,0.67,0.84Exercícios ⇒ Verifique se o MPLAB e C18 estão instalados em seu computador. Caso contrário, realize a instalação conforme o manual.

1. Criando um novo projeto:

- (a) Crie um novo projeto usando Project Wizard com as seguintes características:
 - i. Microcontrolador: PIC18F4550
 - ii. Compilador: C18
 - iii. Diretório do projeto: c:\Exsto\PIC18\Aula 4 MPLAB
 - iv. Nome do projeto: Aula4E1
 - v. Inclua o arquivo de linker script: c:\MCC18\lkr\18F4550.lkr
- (b) Crie um novo arquivo e escreva nela o treco de código abaixo:

```
#include<P18F4550.h>
#include<delays.h>

//// Configurações
#pragma config PLLDIV = 5 // PLL para 20MHz
#pragma config CPUDIV = OSC1_PLL2 // PLL desligado
#pragma config FOSC = HS // Fosc = 20MHz -> Tcy = 200ns
#pragma config WDT = OFF // Watchdog desativado
#pragma config PBADEN = OFF // PORTB começa como digital
```

```
#pragma config LVP = OFF // Sem gravação em baixa tensão
#pragma config DEBUG = ON // habilita debug

void main(void){
    DDRD = 0x00;
    for(;;){
        PORTD = 0x55;
        // Delay10KTCYx(200); // manter comentado na simulação e depuração
        PORTD = 0xFF;
        // Delay10KTCYx(200); // manter comentado na simulação e depuração
    }
}
```

- (c) Salve o arquivo como Exemplo5.c
 - (d) Inclua Exemplo5.c no projeto
 - (e) Compile o projeto
2. Simulando o projeto:
- (a) Ative o simulador MPLAB SIM
 - (b) Abra a janela do Analisador lógico
 - (c) Inclua no analisador lógico o PORTD
 - (d) Execute o programa por um tempo e depois pare, analisando o resultado na janela do analisador lógico. O que o programa faz?
 - (e) Execute o programa passo-a-passo para entender melhor o funcionamento, observando o analisador lógico e o valor de PORTD na janela Watch.
3. Gravando o projeto:
- (a) Retire o comentário das rotinas de delay no programa e recompile-o.
 - (b) Selecione o ICD-2 como programador: Programmer ⇒ Select Programmer ⇒ MPLAB ICD-2
 - (c) Grave o programa no kit.
 - (d) Execute o programa e observe o resultado
4. Depurando o projeto:
- (a) Comente as rotinas de delay no programa e recompile-o b. Selecione o ICD-2 como depurador: Debugger⇒ Select Tool ⇒ MPLAB ICD-2
Obs: Deve aparecer uma janela informando que ao selecionar ICD-2 como depurador ele não mais será usado como gravador. Clique em ok.
 - (b) Execute o programa passo a passo e observe o resultado no hardware e na janela watch.
 - (c) Retire novamente o comentário das rotinas de delay, recompile

- (d) Insira um break-point na linha "PORTD = 0xFF;"
- (e) Rode o programa usando RUN e observe o que acontece.

textcolor[rgb]0.16,0.67,0.84Exercícios propostos

1. Crie um novo projeto com as mesmas configurações do anterior chamado de Aula4P1.
2. Adicione um arquivo C com o código abaixo.

```
#include<P18F4550.h>
#include<delays.h>

//// Configurações
#pragma config PLLDIV = 5 // PLL para 20MHz
#pragma config CPUDIV = OSC1_PLL2 // PLL desligado
#pragma config FOSC = HS // Fosc = 20MHz -> Tcy = 200ns
#pragma config WDT = OFF // Watchdog desativado
#pragma config PBADEN = OFF // PORTB começa como digital
#pragma config LVP = OFF // Sem gravação em baixa tensão
#pragma config DEBUG = ON // habilita debug

void main(void){
    int i;
    DDRD = 0x00;

    for(;;){
        PORTD = 0xFE;
        for (i=0;i<8;i++){
            LATD = LATD << 1;
            Delay10KTCYx(200); // comentado na simulação e depuração
        }
    }
}
```

3. Repita todos os passos do exercício: simulação, gravação e depuração.
4. O que o programa faz?

8.5 Aula 5 - Características Especiais

Objetivos

- Entender as características especiais do PIC18 e quando usá-las;
- Aprender a usar a diretiva #pragma config;
- Compreender os diversos tipos de oscilador do PIC18;

Questionário

1. Quais eventos provocam a saída dos modos Sleep e Idle?
2. Qual a utilidade do Power-up Timer?
3. Para que serve o watchdog?
4. Qual a finalidade do PLL presente no PIC18?
5. Quais as frequências são geradas pelo oscilador interno?
6. Quantas fontes de sinal de clock têm o PIC18? Podemos mudar de fonte de clock durante a execução do programa?
7. Através de quais bits de qual registro é feita a escolha da fonte de clock?
8. Descreva resumidamente os três modos de consumo do PIC18?
9. A tensão do BOR é configurável?
10. Faça uma tabela comparando os modos de oscilador a cristal (XT, HS e LP), oscilador interno e oscilador RC externo com relação a estabilidade, faixa de frequência e custo.

Exercícios

1. O exercício resolvido Aula5E1 apresenta um exemplo que permite verifica o funcionamento do oscilador do PIC18. Através da leitura da chave DIP ligada ao portal B é possível acionar os bits do registro OSCCON. O programa principal aciona sequencialmente LEDs do portal D conforme a frequência de clock selecionada.

RB5 (SCS1)	RB4 (SCS0)	RB3 (IRCF2)	RB2 (IRCF1)	RB1 (IRCF0)	Oscilador	Freqüência	Tempo por LED
Oscilador Freqüência do oscilador interno							
0	0	X	X	X	Primário (HS+PLL)	48 MHz	16,6ms
0	1	X	X	X	Secundário	Não há oscilador secundário na placa	
1	X	0	0	0		31 kHz	12,8s
1	X	0	0	1		125 kHz	6,4s
1	X	0	1	0		250 kHz	3,2s
1	X	0	1	1		500 kHz	1,6s
1	X	1	0	0		1 MHz	800ms
1	X	1	0	1		2 MHz	400ms
1	X	1	1	0		4 MHz	200ms
1	X	1	1	1		8 MHz	100ms

8.6 Aula 6 - Iniciando com a linguagem C

Objetivos

- Conhecer as ferramentas de desenvolvimento utilizadas: MPLAB IDE e C18
- Entender as diferenças entre as linguagens C e Assembly.
- Entender as principais características da linguagem C.
- Estudar os tipos de dados básicos do C
- Atribuição como realizar atribuição em C
- Entender como usar pinos de I/O do microcontrolador em C

Questionário

1. Faça uma comparação de três características das linguagens C e Assembly que mostrem que é vantajoso trabalhar em C.
2. Porque C é considerada uma linguagem de médio nível?
3. O que é o C ANSI?

Exercícios

1. Crie um novo projeto chamado Aula6E1 e escreva um programa que realize as operações apresentadas abaixo. O objetivo do programa é apenas a simulação, portanto basta realizar as operações. Declare as variáveis no menor tamanho que atenda as operações.
 - (a) $x = y + 775$; (y é um unsigned char)
 - (b) $z = w * k$; (w e k unsigned char)
 - (c) $p = (m * 100) / n$; (w e k inteiros sinalizados)
2. Crie um projeto que configure o PORTD (onde estão ligados os LEDs do kit) como saída. No Loop principal leia o valor das chaves dip switch (PORTB) e transfira esse valor para os LEDs.

Obs.: Para ler as chaves é necessário ativar os resistores de pull-up no PORTB. Isso feito colocando o seguinte comando na inicialização do microcontrolador: INTCON2bits.RBPU = 0;

3. Crie um projeto baseado no exercício anterior onde cada chave aciona um LED, conforme a tabela a seguir. Para isso será necessário acessar não o portal todo, mas cada bit individualmente.

Chaves	LEDs
RB0	RD7
RB1	RD6
RB2	RD5
RB3	RD4
RB4	RD1
RB5	RD3
RB6	RD0
RB7	RD2

Exercícios Propostos

1. Crie um novo projeto com as seguintes características
 - (a) Microcontrolador PIC18F4520
 - (b) Ferramentas da linguagem C18
 - (c) Inclua o arquivo linker scrip para esse microcontrolador
 - (d) Crie um novo arquivo chamado "Proposto.C" e inclua-o no projeto.
 - (e) Escreva o código abaixo no arquivo "Proposto.C" e monte o projeto. Em seguida execute a simulação e observe o que o programa faz.

```

#include<P18F4520.h>
#include<stdio.h>

void main (void)
{
    int i=0;

    printf("\n\nExercício Proposto: ");

    for(;;){
        printf("%d ",i);
        i++;
        if(i == 15) Sleep();
    }
}

```

2. Faça o fluxograma do programa "Proposto.C" e descreva sua operação.

8.7 Aula 7 - Estruturas de Decisão

Objetivos

- Conhecer os comandos básicos do C
- Manipular os pinos de I/O do PIC em C18
- Entender o que são funções e como utilizá-las
- Conhecer as funções de temporização do C18

Introdução

Além dos comandos básicos a linguagem C permite o uso de funções que executam tarefas definidas. O padrão ANSI determina diversas bibliotecas de funções que devem acompanhar o compilador. Além destas, o compilador C18 trás diversas bibliotecas de funções para uso específico. Umas das mais úteis é a biblioteca delays.h, cujas funções são apresentadas na tabela abaixo e são detalhadas no anexo F. Essas funções permitem gerar atrasos no programa e cujo tempo é calculado a partir do tempo de execução de uma instrução T_{CY} ($T_{CY} = 4.T_{OSC}$).

Função	Protótipo	Descrição
Delay1TCY	void Delay1TCY(void)	Atraso de 1 ciclo.
Delay10TCYx	void Delay10TCYx(unsigned char unit)	Atraso de múltiplos de 10 TCY
Delay100TCYx	void Delay100TCYx(unsigned char unit)	Atraso de múltiplos de 100 TCY
Delay1KTCYx	void Delay1KTCYx(unsigned char unit)	Atraso de múltiplos de 1000 TCY
Delay10KTCYx	void Delay10KTCYx(unsigned char unit)	Atraso de múltiplos de 10000 TCY

Questionário

1. Como configurar o PORTD do PIC18F4520 para que seus bits sejam todos saídas?
2. Escreva um trecho de código no qual o pino RD2 vá para '1' se o pino RB0 estiver em '0' e vice-versa (use o comando if-else).
3. Para obter um atraso de 100ms, supondo o sinal de clock de 8MHz, qual função de atraso deve ser usada e qual seu argumento?
4. Preencha a tabela abaixo:

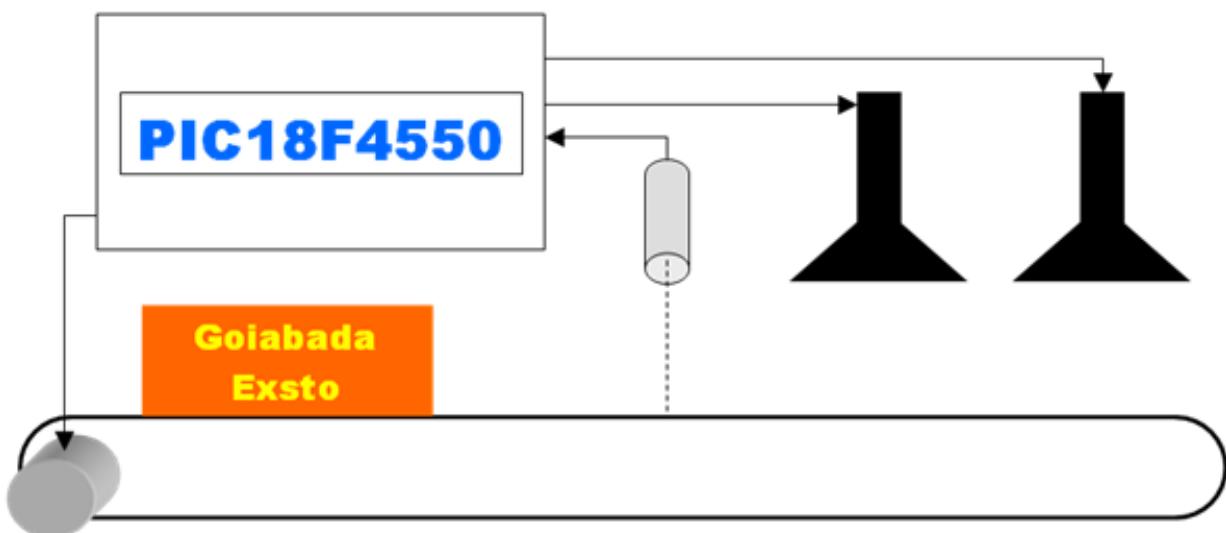
Tipo de Dado	Tamanho [bits]	Faixa de valores
char		
unsigned char		
int		
unsigned int		
Short long		
float		

5. É necessário fazer uma contagem de latas de goiabadas em uma linha de produção. O valor máximo de contagem é 200. Qual o tipo de dado mais adequando para essa variável?
6. O conversor ADC do PIC converte sinais de tensão em valores não sinalizados de 10 bits. Qual o tipo de dado mais adequado para representar essa informação.

Exercícios

1. Exercício resolvido (Aula7E1): Elaborar um programa para controle de uma linha de produção de goiabada, na etapa de impressão de validade e lote. O programa deve cumprir as seguintes tarefas:
 - (a) Manter uma esteira acionada até que passem 5 latas
 - (b) Desligar a esteira
 - (c) Acione o carimbo de "Prazo de validade" por 2s
 - (d) Esperar 1 segundo
 - (e) Acione o carimbo de "lote" por 2s
 - (f) Repetir o processo a partir de 1

O sensor que conta as latas de goiabada é baixo ativo. Os carimbos são alto ativos, ficando acionados enquanto recebem '1' e recolhendo automaticamente quando seu sinal volta a 0.



O hardware está conectado ao microcontrolador conforme a tabela a seguir:

Pino do microcontrolador	Direção	Função
RB0	Entrada	Sensor (baixo ativo)
RD0	Saída	Esteira (alto ativo)
RD1	Saída	Carimbo Lote (alto ativo)
RD2	Saída	Carimbo Validade (alto ativo)

- (g) Simule o projeto e observe o resultado.
 - (h) Grave no kit o projeto e veja seu funcionamento.
2. Levante o fluxograma do exemplo 2 anterior
3. Para aumentar a produtividade, devido ao alto consumo de goiabada, foram instaladas novos carimbos. Agora é possível carimbar 12 latas de goiabada por vez e os carimbos de validade e lote podem ser acionados simultaneamente, por apenas 1s.

⇒Refaça o fluxograma e o código anterior para atender as novas necessidades.

Exercícios Propostos

1. Escrever um programa que controle que pisque um LED com uma frequência de 10 Hz se a chave INT0 estiver pressionada e com frequência de 2 Hz se não estiver. Para fazer isso atribua diferentes valores a uma variável que será o argumento de uma função de delay. Não se esqueça de elaborar o fluxograma
2. Crie um programa (bem como o fluxograma) capaz de implementar a seguinte tabela:

Obedeça a seguinte configuração de hardware.

Chave 1	Chave 2	LED 1	LED 2
0	0	0	0
0	1	0	1
1	0	1	0
1	1	0	0

Pino do microcontrolador	Direção	Função
RB0	Entrada	Chave 1 (INT0)
RB1	Entrada	Chave 2 (INT1)
RD0	Saída	LED 1
RD1	Saída	LED 2

8.8 Aula 8 - Estruturas de Repetição

Objetivos

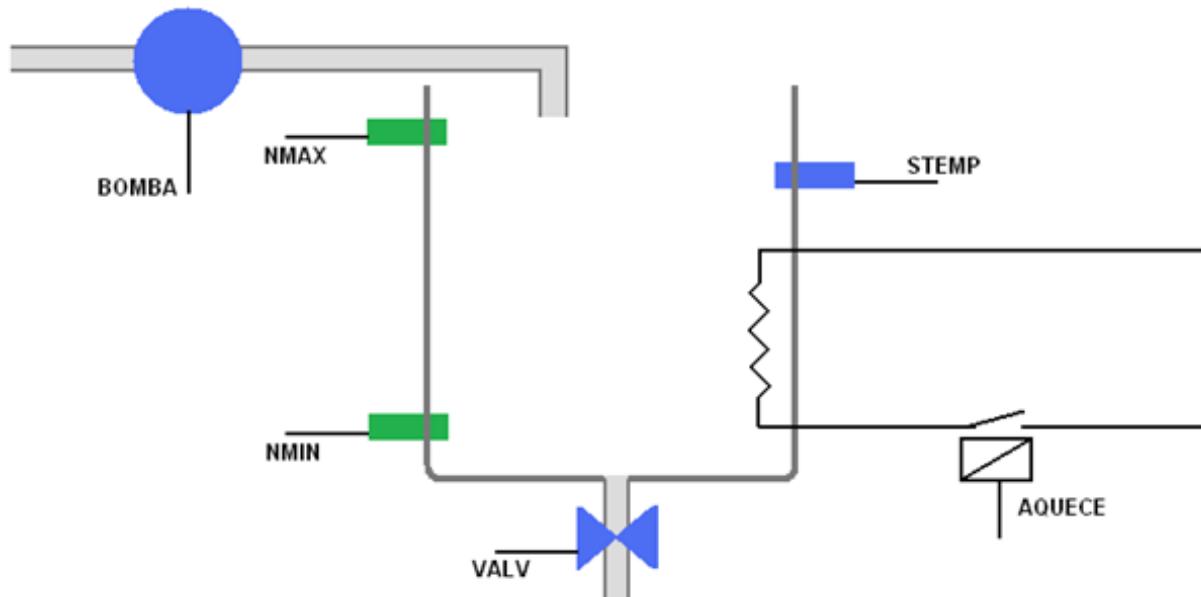
- Estudar as estruturas de repetição
- Compreender as diferenças entre do-while, while e for
- Utilizar o comando break;

Questionário

1. Qual a diferença entre do-while e while?
2. O que difere o for de do-while e while?
3. Para que serve o comando break?
4. Por que programas para microcontroladores geralmente tem um "loop infinito"?

Exercício

1. Exercício resolvido (Aula8E1): Desenvolver o projeto de controle de um sistema de aquecimento de água representado pela figura abaixo.



Funcionamento:

- O tanque deve encher acionando-se a bomba (BOMBA) até que o líquido atinja o nível máximo do tanque, o que será indicado pelo sensor NMAX.
- Estando o tanque cheio, aciona-se o aquecimento através de uma resistência controlada pelo relé AQUECE. Essa Resistência deve ficar acionada por 3 segundos, e depois permanecer acionada até que a temperatura atinja o valor pré-estabelecido, quando então o sensor STEMPS irá atuar.

- Atingida a temperatura máxima a válvula VALV deve ser aberta. Ela deve permanecer aberta até que o nível de água chegue ao mínimo, quando NMIN deixe de indicar presença de líquidos.
- Quando o tanque estiver vazio, a válvula deve ser fechada e o processo se reinicia.

Condições de projeto:

- Condição inicial: todas as entradas desativadas.
- Todos os sensores são alto-ativos e serão simulados por chaves dip switch
- Todas as saídas são baixo-ativas e serão simulados por LED's
- Configuração de hardware:

Sinal	Pino	Direção	Tipo
NMIN	RB0	Entrada	Alto-ativo
NMAX	RB1	Entrada	Alto-ativo
STEMP	RB2	Entrada	Alto-ativo
BOMB	RD0	Saída	Baixo-ativo
VALV	RD1	Saída	Baixo-ativo
AQUECE	RD2	Saída	Baixo-ativo

- (a) Simule o projeto, observando seu funcionamento.
- (b) Grave no kit e teste o projeto.
- (c) Levante o fluxograma do projeto.

2. Altere o projeto para:

- (a) Ao iniciar o programa, deve-se aguardar até que a chave de partida (RB4) esteja em 1 para iniciar o processo.
- (b) Durante o esvaziamento do tanque, a resistência deve alternar entre ligada e desligada a cada 500ms, de forma que a manter a temperatura do líquido.
- (c) Se um comando de emergência for ativado (chave RB3 em '1') o projeto deve parar o que está fazendo, desligar todas as saídas e aguardar novamente o acionamento de RB4. Dica: lembre-se do comando break.

Exercício Propostos

1. Faça um programa que, toda vez que for pressionado o a chave em RB0, o LED em RD0 pisque 3 vezes, usando obrigatoriamente while.
2. Faça um programa que, toda vez que for pressionado o a chave em RB0, o LED em RD0 pisque 3 vezes, usando obrigatoriamente for.
3. Desenvolva um programa que realize as funções descritas na tabela abaixo (lembrando que as chaves são baixo ativas);

Rb0	Rb1	Ação
RB0	RB1	Ação
1	1	Nenhuma ação
0	1	LED0 piscando a 1Hz
1	0	LED0 piscando a 8Hz
0	0	LED0 aceso constantemente

8.9 Aula 9 - Funções

Objetivos

- Estudar funções o uso de funções e da pilha;
- Estudar as diferentes formas de passagem de parâmetro;
- Entender o conceito de biblioteca de funções.

Questionário

1. Quantos níveis têm a pilha do PIC18?
2. O que é uma variável local?
3. O que é uma variável global?
4. Como se comporta uma variável declarada com modificador static?
5. O que é protótipo de uma função?

Exercícios

1. Exercício Resolvido (Aula9E1): Simule o exemplo Funções.c, observado o funcionamento da pilha, o comportamento dos diferentes tipos de variáveis (global, local e static) e a passagem de parâmetros.
2. Crie uma função que realize a decomposição de inteiro para 5 dígitos, chamada IntToBCD. O parâmetro de entrada (inteiro) deve ser o parâmetro de entrada da função. Os 4 dígitos devem ser variáveis globais.
3. Altere o exercício anterior para que os dígitos sejam parâmetros passados por referência (ponteiros).
4. Crie uma biblioteca de funções (composta por IntToBCD.h e IntToBCD.c) para conter função desenvolvida no exemplo anterior.

8.10 Aula 10 - Interrupções

Objetivos

- Estudar o sistema de interrupções do PIC18
- Entender a prioridade de interrupções
- Construir rotinas de tratamento de interrupção.

Questionário

1. Quantos níveis de prioridade de interrupção há no PIC18?
2. Quando o programa está sendo executado e ocorre uma interrupção de baixa prioridade, o que ocorre?
3. Se uma interrupção de alta prioridade ocorre durante a execução da RTI de baixa prioridade, o que acontece.
4. Quais os bits de habilitação global das interrupções de alta e baixa prioridade?
5. Como identificar qual a causa da interrupção?
6. Que providências devem ser tomadas sempre ao se tratar uma interrupção?

Exercício

1. Exercício resolvido (Aula10E1): Para demonstrar o funcionamento das interrupções vamos desenvolver um programa que acione os LED's de forma diferente no programa principal e nas interrupções de baixa e alta prioridade. A interrupção de baixa prioridade será a interrupção INT2 e a interrupção de alta prioridade será INT1.
 - No programa principal os leds ficam todos piscando ao mesmo tempo, em uma frequência de 2Hz.
 - Quando ocorrer uma interrupção de baixa prioridade será feito um "deslocamento" de LED acionado da esquerda para a direita e da direita para esquerda, 3 vezes. Cada LED permanece acesso por 200ms.
 - Quando ocorrer a interrupção de alta prioridade, os LED's serão acionados intercalados (10101010), se terão seus estados invertidos a cada 500ms, por 3 vezes.

Execute o programa. Teste inicialmente as interrupções separadamente. Em seguida tente a acionar uma enquanto as outras estão sendo executadas. Observe e anote os resultados.

⇒ Atenção: Esse exemplo é para fins didáticos somente. É extremamente recomendado que não se use rotinas de atraso dentro de uma RTI.

2. Altere o exemplo anterior de forma que não haja mais prioridade de interrupção (IPEN = 0). Nesse caso, só estará ativa a interrupção de alta prioridade e as RTI devem ser reescritas. Execute o programa e acione uma chave por vez. Em seguida tente a acionar uma enquanto a outra está sendo executada. Observe e anote os resultados.

Exercícios propostos

1. Escreva um programa que inverta o estado de RD0 a cada 500ms em sua rotina principal. O pino RD1 deve ter seu estado invertido conforme o valor de uma variável global Tempo. Quanto ocorrer interrupção INT1, que deve ser de alta prioridade, RD2 irá para '1' e Tempo recebe um valor que faz o RD1 inverter de estado a cada 200ms. Quando ocorrer a interrupção INT2, que deve ser de baixa prioridade, RD2 irá para '0' e RD1 passa a inverter a cada 1s.

8.11 Aula 11 - Display de 7 segmentos e buzzer

Objetivos

- Estudar os displays de 7 segmentos
- Utilizar vetores como tabelas de conversão
- Multiplexar displays de 7 segmentos

Introdução

Em muitas aplicações com microcontroladores é necessário passar informações para pessoas. Nestes casos, diversos recursos podem ser utilizados. Dois muito comuns são o display de sete segmentos e o buzzer, que serão tratados nesta experiência.

O display de 7 segmentos é um arranjo de LEDs que permite formar números e algumas letras conforme a combinação de segmentos acionados. Esse tipo de display pode ser multiplexado, isto é, vários ligados em conjunto acionando um só por vez, o que o torna um interface bastante versátil.

O buzzer nada mais é que um alto-falante muito simples, feito para emitir sons com maior intensidade em determinadas frequências. Esse componente se torna muito útil na medida em que pode chamar a atenção do usuário para um equipamento, sem que aquele esteja olhando diretamente para este. Basicamente existem buzzers auto-oscilantes, que basta alimentar para que emita som, e buzzers que precisam receber um sinal quadrado para serem acionados. O kit é equipado com um buzzer desse último tipo.

Questionário

1. Como é o circuito interno de um display de 7 segmentos catodo-comum?
2. Como declarar um vetor de constantes
3. Desenhe um circuito capaz de multiplexar 4 displays de 7 segmentos?
4. Para que o circuito da questão acima possa operar corretamente, qual tempo cada segmento deve permanecer acionado.
5. Admitindo que um buzzer tenha seu melhor desempenho na frequência de 4,2kHz, qual o período do seu sinal?
6. Para o buzzer do exemplo acima, escreva um trecho de código capaz de acionar ininterruptamente o buzzer usando rotinas de delay e supondo um oscilador de 20 MHz. Dica: o pino do microcontrolador que aciona o buzzer deve permanecer meio período em '1' e meio período em '0'.

Exercícios

1. Desenvolver um programa que apresente em um display de 7 segmentos um valor de 0 a F (hexadecimal). Esse valor deve ser incrementado por RB0 e decrementado por RB1.

2. Desenvolver um programa capaz de apresentar o valor de uma variável com valor máximo de 9999 nos displays de sete segmentos multiplexados. Esse valor deve ser incrementado por RB0 e decrementado por RB1. Dica: cada segmento deve permanecer ativo por 5 ms.
3. Escreva um programa que acione o buzzer quando a chave INT1 for acionada e só o desligue quando INT2 for pressionada.

Exercício Propostos

1. Altere o exercício 2 para que ele seja possível programar um valor de contagem usando RB0 e RB1 e, ao pressionar RB2 esse valor seja decrementado a cada 1 segundo, até zerar.
2. Altere o exercício 3 para que, ao pressionar o botão, a frequência do buzzer alterne entre 4,2 kHz e 2,4kHz, permanecendo (aprox.) 1s cada um. Dica: para definir o tempo em que cada frequência é usada conte os períodos do sinal gerado.

8.12 Aula 12 - Teclado Matricial e LCD

Objetivos

- Estudar o uso de display de cristal líquido alfanumérico
- Estudar leitura de teclado matricial por interrupção

Introdução

Para construir uma IHM (interface homem-máquina) eficiente podemos usar display de cristal líquido alfanuméricico e teclas matriciais. Além disso, o portal B possui uma interrupção especialmente útil para a escrita de teclado. Essas facilidades podem ser usadas em C através de bibliotecas de funções.

Questionário

1. Quais os sinais necessários para controlar um display de cristal líquido alfanuméricico?
2. Qual interrupção do PIC18 pode ser usada em conjunto com um teclado matricial?

Exercícios

1. Exemplo resolvido (Aula12E1): Construir um programa que faça a leitura do teclado e escreva a tecla lida no centro da segunda linha do LCD. Na primeira linha deve aparecer a mensagem "Exsto Tecnologia". Execute o programa e observe os resultados
2. Altere o programa anterior para escrever seu nome em lugar da mensagem "Exsto Tecnologia".
3. Altere o programa para que sejam lidos 4 teclas. Se a sequência dessas teclas for 1, 2, 3 e 4, deve aparecer a mensagem "Senha correta no LCD"; caso contrário, deve parecer "Senha Errada". As teclas lidas devem ser escritas uma ao lado da outra no LCD.

Exercícios Propostos

1. Desenvolva uma rotina capaz de escrever no LCD o valor de variáveis do tipo unsigned char, chamada PrintChar.
2. Desenvolva uma rotina capaz de escrever no LCD o valor de variáveis do tipo unsigned int, chamada PrintInt.

8.13 Aula 13 - Contadores e temporizadores

Objetivos

- Entender o funcionamento do comparador analógico;
- Desenvolver rotinas de temporização utilizando temporizadores e interrupção.

Introdução

Como o próprio nome indica, os timers são periféricos muito úteis para gerar temporização. Para isso, determinamos um valor inicial que deve ser carregado no timer para que ele gere uma interrupção após um tempo pré-determinado. A equação (1) mostra como calcular o tempo (T) até o estouro do timer após a carga de um valor inicial (T_0), usando uma pré-escala de valor PS. O sinal aplicado ao timer tem período TCY (quando a fonte de clock é interna). TOV é o valor de contagem que causa overflow (65536 para 16 bits e 256 para 8 bits). Essa equação é válida para os timers 0, 1 e 3.

$$T = (TOV - T_0) \times (PS \times T_{cy}) \quad (8.1)$$

A equação a seguir mostra como determinar T_0 para obter um tempo T desejado. Nesse caso, o valor da pré-escala deve ser arbitrado.

$$T_0 = 1 - \frac{1}{PS} \times \frac{T}{T_{cy}} \quad (8.2)$$

Questionário

1. Quais os timers do PIC184550? De quantos bits é cada timer?
2. Qual a função da pré-escala? E da pós-escala do timer 2?
3. Se o timer 1 está configurado com pré-escala de 1:4 e usando a fonte de clock interna com oscilador de 8MHz quanto tempo entre o momento que o timer é zerado e seu overflow?
4. Se o timer 2 é configurado com pré-escala de 1:16, pós-escala de 1:3, usando a fonte de clock interna com oscilador de 4MHz e o registro PR2 contém o valor 200, a cada quantos milisegundos o flag de interrupção é setado?
5. Qual o argumento para a função OpenTimer1 para configurar o timer 1 conforme a questão 3?
6. Qual o argumento para a função OpenTimer2 para configurar o timer 1 conforme a questão 4?

Exercícios

1. Desenvolver uma aplicação que gere um sinal de 1 Hz no pino RD1 e um sinal de 10 Hz no pino RD0, gerados pelo timer 1 e timer 3, respectivamente. Para isso, utilize interrupção sem prioridade e oscilador externo de 20MHz. O programa principal deve inicializar o microcontrolador e entrar em loop infinito, sem nenhuma ação.
2. Esboce o fluxograma das RTI's do exemplo acima.

3. Altere o exemplo para usar apenas o timer 3 para gerar as duas frequências.

Exercícios Propostos

1. Determinar as equações para determinação de T_0 e T aplicáveis ao timer 2.
2. Crie um "relógio", com contagem de hora, minutos e segundos usando o timer 1

8.14 Aula 14 - Conversor A/D'

Objetivos

- Entender como funcionam os conversores analógico/digital
- Estudar o funcionamento do conversor A/D do PIC18F4550
- Fazer a leitura de um nível de tensão e processar essa informação no microcontrolador.

Questionário

1. Por que conversores A/D são necessários?
2. Quais os pinos de entrada dos comparadores no PIC18F4550? Após o reset, esses pinos estão configurados como entradas analógicas ou digitais.
3. Qual o evento que gera a interrupção do comparador?
4. Como são acessadas as saídas dos comparadores em C?

Exercícios

1. Exercício Resolvido (Aula14E1): Desenvolver um programa que faça a leitura do canal 0 do conversor A/D e acione os LEDs do PORTD progressivamente conforme essa tensão aumenta (bargraph). Configure o kit de forma que o canal 0 leia a o sinal do potenciômetro (POT).
2. O kit possui um sensor de temperatura LM35. Este sensor apresenta em sua saída um sinal linear de 10mV/oC. Faça a leitura desse sensor e mostre o valor no LCD; para tanto será necessário fazer a conversão do valor lido de binário para oC e posteriormente "quebrar" este valor em dígitos para apresentá-lo no display.

⇒ Dica: evite o uso de variáveis fracionárias (ponto flutuante).

Exercícios Propostos

1. Faça um programa de controle de temperatura, que mantenha a temperatura lida pelo sensor entre 32 e 40oC. Para isso faça a leitura do sensor e acione a resistência de aquecimento (RC1) para aquecer até atingir 40oC e a ventoinha (RC2) para resfriar até atingir 32 oC. Apresente no LCD a temperatura e se está aquecendo ou esfriando.

8.15 Aula 15 - Módulo CCP

Objetivos

- Estudar os três modos de operação do módulo CCP
 - Captura
 - Comparaçāo
 - PWM
- Compreender o sistema de geraçāo de clock do PWM
- Realiza aplicações práticas dos modos comparaçāo e PWM.

Questionário

1. O que fazem cada um dos modos de operação do CCP
2. Quais os eventos (configuráveis) que geram a captura do timer no modo captura?
3. No modo comparaçāo, qual registro é comparado com o timer?
4. Quais as ações podem ser realizadas quando o timer coincide com o registro, no modo captura?
5. Explique de forma sucinta o funcionamento do modo PWM.
6. Porque esse processo PWM é chamado de "modulação"?

Exercícios

1. Exercício Resolvido (Aula15E1): Configure o timer 1 e ECCP1 para que o módulo CCP gere uma interrupção a cada 1 segundo. No tratamento de interrupção do ECCP1 implemente rotinas de um relógio de hora, minutos e segundos. No loop principal apresente no LCD a hora.
2. Exercício Resolvido (Aula15E2): Desenvolver um programa que configure a frequência do PWM do CCP2 em 2 kHz e permita a variação do Duty-cicle com incrementos e decrementos de 10
3. Combine os dois exemplos anteriores, de forma que tenhamos o relógio e o controle da luminosidade da lâmpada ao mesmo tempo

Exercícios Propostos

1. Altere o exercício proposto da aula 15 para que a conversão do AD seja disparada pelo módulo CCP2 configurado em modo comparaçāo, gerando disparos a cada 100ms. Para isso deve-se ativar a interrupção do ADC (e não do CCP2) para fazer a leitura dos valores convertidos. O controle a apresentação da temperatura devem continuar sendo realizados no loop principal.

2. Configure o ECCP1 em modo PWM gerando um sinal de 2kHz. Faça o valor do Duty-cycle variar de 0 ao valor máximo seqüencialmente. Configure o kit para que RC1 esteja ligado a entrada do DAC e meça com um osciloscópio o sinal de saída.

⇒Dica: a variação do valor do PWM deve ser sincronizada com os períodos do PWM. Para gerar esse sincronismo utilize o flag de interrupção do timer 2, que estoura a cada período do PWM quando a pós-escala do timer está em 1:1.

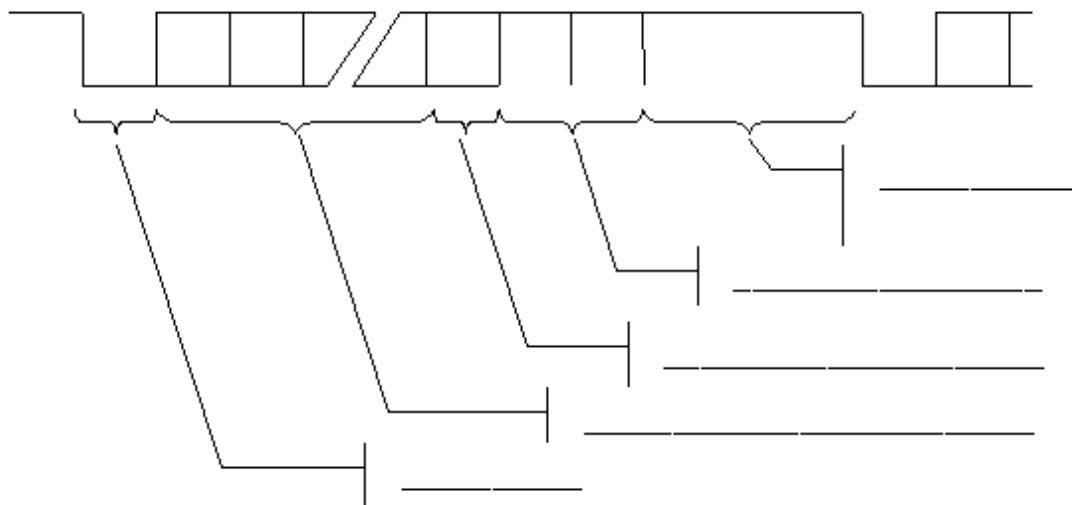
8.16 Aula 16 - Comunicação serial assíncrona (RS232 e RS485)

Objetivos

- Compreender o princípio de funcionamento das comunicações seriais
- Estudar os fundamentos dos padrões RS232 e RS485
- Entender como a USART do PIC184550 possibilita realizar comunicação serial baseada no padrão RS-232

Questionário

1. Sobre comunicação paralela, cite:
 - (a) Suas vantagens:
 - (b) Suas desvantagens:
2. A comunicação de forma serial elimina vários problemas presentes na comunicação paralela, mas apresenta algumas desvantagens. Quais são elas.
3. Cite alguns sistemas de comunicação serial.
4. O que é comunicação síncrona?
5. O que é comunicação assíncrona?
6. Explique o processo pelo qual é possível criar sincronismos na comunicação assíncrona sem a presença do sinal de clock.
7. Por que não é possível enviar infinitos bits usando apenas um star bit inicial?
8. Preencha a figura abaixo com os elementos que compõem a comunicação serial e descreva a função de cada um deles.



9. O que são DTE e DCE?
10. Descreva a função dos principais sinais da comunicação RS232.
11. Segundo a norma, qual a taxa máxima suportada pela RS232? E qual a distância máxima?
12. Quais níveis de tensão representam os estados lógicos '0' e '1' na transmissão do padrão RS232?
13. Cite um componente responsável por fazer a conversão de sinais em nível TTL pra RS232.
14. O que é transmissão balanceada?
15. Como é possível eliminar o ruído em uma transmissão balanceada?
16. Qual a taxa máxima especificada no padrão RS485? E a distância máxima?
17. Supondo que em uma ligação de rede RS485 de 800 metros foi medido um tempo de subida (R) de 6 s, qual a taxa máxima pode ser alcançada?
18. Um equipamento comunica-se usando RS485 a uma taxa de 250 kbps. Qual o tempo de subida máximo aceito para que a comunicação funcione corretamente?
19. Calcule o valor do registro SPBRG e o erro percentual para as taxas de transmissão abaixo, considerando a frequência do oscilador de 20MHz. Arbitre entre a opção de taxas altas e taxas baixas para obter o menor erro possível

Taxa de transmissão (BR)	Spbrg	Taxas altas ou baixas?	Taxa real	e%
2400 bps				
9600 bps				
19200 bps				
57600 bps				
250000 bps				

Exercícios

1. 1. Exercício resolvido (Aula16E1): Desenvolver um programa capaz de transmitir via RS232 para um computador as teclas pressionadas no teclado e imprimir no LCD os caracteres recebidos do computador. Utilize taxa de transmissão de 9600 bps. ⇒Dica:
 - (a) Utilize no computador o Hiper Terminal ou outro software de terminal disponível.
 - (b) Para ganhar tempo, tome como base o arquivo Aula12E1 (LCD e teclado)
2. Altere o programa anterior para realizar a comunicação entre dois kits utilizando RS485. Da mesma maneira que o exercício acima, o valor pressionado no teclado deve ser enviado e o caracteres recebidos impressos no display, de forma que a tecla pressionada em um kit aparece no display do outro, e vice-versa.
 ⇒ Dica: para usar o RS485, além das funções de transmissão é importante lembrar que se faz necessário ativar o transmissor MAX485, o que é feito colocando '1' no pino de habilitação desse componente (ligado a RA4). O pino deve permanecer em '1' pelo tempo da transmissão, que não é o tempo da função, já que a função simplesmente inicia a transmissão.

Exercícios Propostos

1. Desenvolver um programa que fique constantemente lendo o sensor de temperatura e envie essa informação (em graus Celsius) pela porta serial RS232 para um computador.
2. Desenvolver um sistema onde 3 kits trocam informações entre si, sendo um mestre e dois escravos. Cada escravo deve ter um endereço diferente, de forma que o mestre consiga se comunicar hora com um, hora com outro. O mestre deve realizar um comando de leitura que tenha como resposta o valor Lido no potenciômetro POT de cada kit, e mostre simultaneamente esses valores no LCD.

8.17 Aula 17 - SPI

Objetivos

- Entender o funcionamento do protocolo de comunicação SPI
- Aprender a configurar o módulo MSSP do PIC18 para operar em modo SPI mestre
- Conhecer o funcionamento de potenciômetros digitais.
- Implementar um comunicação serial entre o PIC18 e o potenciômetro digital do kit

Introdução

Existem diversos componentes que utilizam o protocolo de SPI para se comunicar com outros circuitos. De fato, como um modo eficiente e simples de comunicação, os mais diversos componentes que podem ser comunicar com microcontroladores possuem essa porta de comunicação, como memórias, relógios, expansores de I/O, dispositivos de comunicação (Zigbee, ethernet, rádios), CODEC, AD, DA, amplificadores de ganho programável, sensores de temperatura, acelerômetros, para citar alguns exemplos.

Como aplicação para o protocolo SPI no kit XM118 existe um potenciômetro digital. Um potenciômetro digital é um componente que se comporta como um potenciômetro convencional, isto é, possui terminais e um cursor, e a resistência entre o cursor e os terminais varia conforme o ajuste. Porém, ao contrário de um potenciômetro convencional, o ajuste do potenciômetro digital é feito digitalmente por um microcontrolador. No caso do componente em questão no kit, que é o MCP41010, esse ajuste é feito através da interface SPI.

No kit o potenciômetro digital está ligado numa montagem divisor de tensão, de maneira que os terminais estejam ligados a VDD e a GND e que possamos medir a tensão no cursor variando entre esses dois extremos. Esse sinal pode ser aplicado a um canal de AD.

Para mais detalhes do funcionamento do MCP41010 e para entender como ele se comunica com o microcontrolador, consulte seu manual, que se encontra no CD que acompanha o kit.

Questionário

1. Em quais modos pode operar a SPI do PIC18F4550?
2. Descreva a função de todos os pinos da porta SPI do PIC18F4550.
3. Quais fontes de clock podem gerar a taxa de transmissão do módulo MSSP em modo SPI?
4. Consulte o manual do potenciômetro digital MCP41010 e responda as questões a seguir
 - (a) Qual a resistência do potenciômetro interno do MCP41010?
 - (b) Qual a faixa de tensão nos terminais do potenciômetro.
 - (c) O número de tapes representa quantos valores de resistências podem ser ajustadas no MCP41010. Qual é esse valor?
5. Encontre no manual o diagrama de tempo que resume o protocolo de comunicação com o potenciômetro digital.

6. Encontre no manual a tabela que apresenta o formato dos comandos enviados ao potenciômetro digital.

Exercícios

1. Exercício Resolvido (Aula16E1): Implemente um programa que envia valores para o potenciômetro digital a cada 50ms. Esses valores devem variar seqüencialmente entre 0 e 255 e depois de 255 a 0. Tomando como base o primeiro exercício sobre AD, meça o valor da saída do potenciômetro e mostre-o através de um bargraph de LEDs. Meça com um osciloscópio esse sinal.
2. Desenvolva um programa que leia a tensão do potenciômetro do kit através do canal AN3 e ajuste o potenciômetro digital com essa mesma tensão (ou o mais próximo que for possível). No display LCD devem ser mostradas as tensões lidas do potenciômetro convencional (AN3) e do potenciômetro digital (AN0).

Exercícios Propostos

1. Altere o exercício 2 para que o valor do potenciômetro digital só se ajuste ao potenciômetro convencional enquanto a tecla em RB0 estiver pressionada, mantendo seu valor caso contrário.

8.18 Aula 18 - I^2C

Objetivos

1. Estudar o protocolo I^2C
2. Entender o funcionamento de uma memória serial I^2C
3. Realizar uma aplicação prática envolvendo memória externa e I^2C .

Questionário

1. Em quais modos pode operar o PIC em modo I²C?
2. Explique como é possível realizar a comunicação bidirecional entre dois dispositivos no barramento I²C usando apenas 1 terminal de dados (SDA). Porque é importante a presença do resistor de pull-up ligado a esse pino?
3. O que caracteriza as condições de Star e Stop no protocolo I²C?
4. O que é o ACK? Quando e como ele é gerado?
5. Qual o procedimento para gerar uma leitura de um endereço qualquer de um dispositivo I²C?

Exercícios

1. Exercício Resolvido (Aula17E1): Desenvolver um programa que leia o teclado e apresente na primeira linha do display valor lido. Se for pressionada a tecla 'A' deve-se gravar a última tecla válida na EEPROM, no endereço 0. Na segunda linha deve-se apresentar o valor lido no endereço 0 da EEPROM. Após grava algum valor desligue o kit e observe que a EEPROM não perde os dados.

⇒ Cuidado! Garanta que o programa só realize uma gravação cada vez que a tecla 'A' for pressionada. Caso o programa entre em loop e realize múltiplas gravações a memória pode ser danificada.

2. Altere o projeto desenvolvido no exercício 3 da aula 12, onde são lidas 4 teclas e comparadas com o "1234", para que a comparação seja feita com os quatro primeiros endereços da EEPROM externa. Além disso, se for pressionada a tecla A peça a senha antiga e em seguida uma nova senha e grave-a na memória.

⇒ Dica: é necessário garantir que a senha original ("1234") esteja gravada na memória. Escreve um trecho de código para fazer isso uma vez e em seguida comente essa parte do programa para que a última senha gravada seja preservada.

Exercícios Propostos

1. Desenvolva um programa que grave na memória 10 valores lidos do AD (apenas os 8 bits mais significativos), sendo que a cada vez que tecla INT for pressionada após o reset um valor seja gravado. Após grava os 10 valores o passe a enviar esses valores repetidamente

para o potenciômetro digital, a intervalos de 100ms. Use um osciloscópio para observar o resultado.

⇒ Como potenciômetro existe apenas um módulo MSSP realizando funções de SPI e I2C é necessário desativar um modo antes de entrar em outro. Para isso use as funções Openxxx() e Closexxx().

8.19 Aula 19 - Interface industrial

Objetivos

- Estudar circuitos para interfacear o microcontrolador com sistemas de controle industrial
- Aprender a usar o expansor de I/O MCP23016
- Implementar aplicações de controle usando a interface industrial

⇒ Atenção: esta experiência faz uso do módulo XMM01 - Industrial Interface, que é um item opcional pra o kit XM118. Verifique se esse produto faz parte do pacote adquirido.

Introdução

Expansores de I/O

Apesar de estarem disponíveis microcontroladores com diferentes quantidades de pinos, as vezes as condições do projeto levam a uma situação onde não há pinos suficientes para a aplicação desejada. Quando isso acontece deve-se pensar em algum processo de expansão de I/O. O método mais simples é através do uso de buffers (por exemplo, 74HC244) e Latchs (por exemplo, 74HC573), criando um "barramento de dados" externo compartilhado por esses dispositivos. O inconveniente desse processo é que, mesmo com a expansão, o número de pinos demandada é grande. Outro método possível são resgitradores de deslocamento, fazendo conversão serial/paralelo ou paralelo/serial, de forma que o microcontrolador utilize poucos pinos em uma interface serial. Por fim, mais modernamente estão disponíveis componentes chamados de expansores de I/O. Esses componentes têm as mesmas características dos portais do microcontrolador, podendo-se configurar individualmente seus pinos como entrada ou saída. Eles possuem interface SPI ou I2C, facilitando muito o desenvolvimento de aplicações.

O módulo XMM01 utiliza um expansor de I/O MCP23016 para obter mais 16 pinos de I/O (8 entradas e 8 saídas). Interface industrial

Muitas vezes o microcontrolador é utilizado em aplicações industriais, substituindo PLCs ou em automação de máquinas. Nesses casos alguns cuidados deve ser tomados. O circuito do microcontrolador, que vamos chamar circuito de controle, opera em baixas tensões (3,3VDC ou 5VDC) e aceita um certo nível de ruído. Já as cargas e dispositivos entrada de aplicações industriais costumam operar em tensões mais altas (10 VDC, 24 VDC e 48 VDC) ou mesmo tensão alternada (110VAC e 220VAC). Além disso, o nível de ruído presente é maior, proveniente de chaveamento de circuitos de potência (contatores, tiristores, etc...) ou proveniente de dispositivos como motores, bombas, etc... Portanto, na prática é o desejável o isolamento galvânico entre as partes de controle e potência do circuito (dois circuitos são isolados galvanicamente quando não existe contato elétrico entre eles). Nessas situações usa-se foto-acopladores (também conhecidos como opto-isoladores) para fazer garantir o isolamento dos sinais. Essa precaução não tem sentido se os circuitos ficarem em contato próximo na placa, portanto também deve existir um cuidado de layout para garantir a separação dos diferentes domínios elétricos.

O módulo XMM118 possui entradas e saídas isoladas (foto-acopladas) que permitem a conexão a sinais de 10 a 24VDC. Através do uso desse módulo é possível realizar aplicações de controle em aplicações reais industriais, como: leitura de sensores digitais e chaves, acionamento de relés, válvulas e outros dispositivos. Adicionalmente o próprio kit XM118 possui entradas (0 a 10V, 4

a 20 mA) e saídas (a saída do DAC pode ser ajustada para até 0 a 10V) que podem ser usadas em aplicações tipicamente industriais (nesse caso não existe isolamento galvânico).

Além disso, observe que os domínios de elétricos estão separados, o que pode ser evidenciado pelos diferentes tipos de malha de terra empregados na placa.

Questionário

⇒ Consulte o manual do MCP23016 e responda as perguntas abaixo.

1. Quantos pinos de I/O o dispositivo possui?
2. Qual a interface de comunicação desse dispositivo?
3. O MCP23016 possui 3 pinos para definição de seu endereço no barramento de comunicação. Quais são esses pinos? Quantos dispositivos iguais poderíamos ter em um mesmo barramento então? Quantos pinos de I/O estariam disponíveis nessa condição?
4. Quais os registros responsáveis pela definição de direção dos pinos?

Exercícios

1. Exercício Resolvido (Aula19E1): Desenvolva uma biblioteca de funções que permita (1) configuração do MCP23016 no que se refere a direção dos pinos de I/O, (2) Leitura de portal de entrada e (3) escrita no portal de saída. Para testar essa biblioteca a informação lida da entrada deve ser apresentada nos LEDs do kit e as saídas devem ser acionadas seqüencialmente a intervalos de 100ms.
⇒ O protocolo de comunicação do MCP23016 é similar ao da memória EEPROM, portanto use as rotinas de leitura e escrita da EEPROM I2C presentes na biblioteca I2C.h para fazer as novas funções.
2. Alterar o programa 3 da aula 7 para operar usando o XMM01.
3. Alterar o programa 2 da aula 8 para operar usando o XMM01.

MANUAL DE OPERAÇÃO E MANUTENÇÃO

O manual de operação e manutenção descreve os circuitos do kit didático, detalhando seu funcionamento. São também apresentados os esquemas elétricos desses circuitos e valores de componentes, de forma a permitir a manutenção do equipamento.

Este manual tem como objetivos principais:

- Apresentar o conteúdo do kit (equipamentos e documentação)
- Apresentar um guia rápido de instalação e testes
- Apresentar detalhes de instalação de software e hardware
- Descrever os circuitos do kit, de forma a permitir identificação de

Capítulo 9

Conteúdo do Kit:

Depois de retirar o seu kit de desenvolvimento da caixa, verifique se o mesmo possui os seguintes itens:

- 01 Kit educacional em bastidor metálico, com fonte de alimentação e gravador/depurador X-ICD2 embutido;
- 01 Cabo de alimentação de três pinos(2P+T);
- 01 Cabo de comunicação serial;
- 02 Cabos de comunicação USB (um para gravação/depuração e um para aplicação)
- 01 header (placa para gravação de outros microcontroladores)
- 01 cabo RJ12/RJ12, para conexão do kit ao header.
- CD

Caso ocorra a falta de algum destes itens ou defeito, consulte a Exsto Tecnologia para esclarecimentos.

9.1 Conteúdo do DVD

O CD traz toda a informação e programas necessários para o uso do kit, tais como:

- Esquemas elétricos do kit.
- Exemplos e exercícios resolvidos.
- Documentação do produto.
- Manual dos componentes e outros documentos relevantes
- Instalação do Quartus II e outros softwares úteis.

⇒ Ao inserir o CD no drive ele deve automaticamente iniciar um aplicativo que permite navegar por seu conteúdo.

9.2 Instalações

9.2.1 Instalação do Hardware

Ao retirar o kit da embalagem confirme a presença de todos os acessórios e equipamentos, conforme descrito no item 1 deste manual.

Para alimentar o kit ligue o cabo tripolar ao kit e a uma tomada de alimentação. O kit é equipado com fontes chaveadas que operam, de forma automática, com 110 ou 220V, em 50 ou 60Hz. Para ligar o kit, use a chave que se encontra junto ao conector tripolar, na parte traseira do kit.

Ligue o kit e verifique se os LED's indicadores de alimentação e o logotipo da Exsto na parte frontal se acenderam, indicando que está tudo funcionando corretamente.



Atenção: o kit possui proteções contra descargas na entrada de alimentação e seu bastidor é aterrada para maior segurança dos usuários. Porém, esses recursos de proteção só são eficazes se o terceiro pino do cabo tripolar estiver conectado a um aterramento de qualidade. A Exsto tecnologia não se responsabiliza por danos ou acidentes ocasionados por baixa qualidade ou inexistência de aterramento.

Na parte traseira do bastidor existe um conector USB. Esse conector conecta o computador ao gravador/depurador X-ICD2.



Atenção: não conecte o cabo USB da X-ICD2 antes de ter instalado o MPLAB, caso contrário não será reconhecido o driver do dispositivo, podendo ocasionar falhas de funcionamento.

Para ligar o seu X-ICD2 siga os passos abaixo:

1. Verifique se o cabo USB que acompanha o produto está conectado a uma porta USB do seu computador;
2. Conecte o X-ICD2 a outra ponta do cabo USB. Depois disto, um led verde com a identificação "USB" deve acender.

3. Na hora da primeira colocação do X-ICD2, será solicitada a instalação dos drivers do novo dispositivo pelo sistema operacional;
4. Depois de indicado a localização dos drivers, o sistema operacional os instalará e o X-ICD2 estará pronto para uso.

É importante lembrar que o equipamento, estará pronto para o uso assim que os seus drivers estiverem instalados. Na sequência do manual, serão descritos os procedimentos de instalação dos drivers e da ferramenta MPLAB da Microchip.

Ainda, devemos lembrar que o uso do equipamento é feito principalmente através do software MPLAB e por isso seria ideal se o mesmo já estivesse instalado.

Depois de alimentada e detectado os drivers da placa, você estará pronto para utilizar o X-ICD2 . A conexão/desconexão do X-ICD2 deve ser feita com a ferramenta de software MPLAB fechada. Caso adicione ou retire o X-ICD2 da conexão USB com o MPLAB aberto, o mesmo não funcionará corretamente no aplicativo.

Além das conexões de alimentação e do X-ICD2 existe ainda a conexão USB de aplicação e serial (RS232). Essas conexões serão usadas nas experiências com o kit.

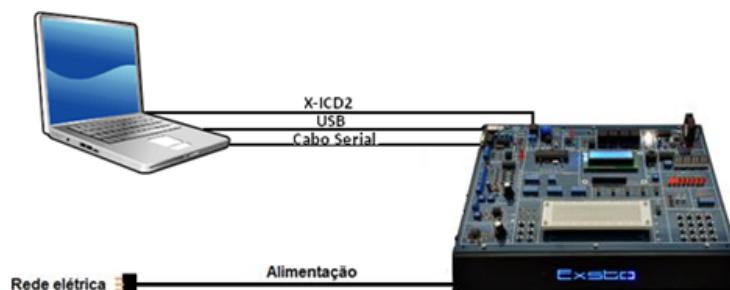


Figura 9.1: Diagrama de instalação do kit

9.2.2 Instalação dos Softwares

Instalação do MPLAB 8.20



A seguir é apresentado o procedimento de instalação do MPLAB 8.20. No momento em que esse documento foi escrito esta era a versão mais recente desse software. Contudo, recomendamos usar sempre a versão mais atual, que pode ser obtida gratuitamente em www.microchip.com

Execute o arquivo **Install_MPLAB_v820.exe**. Para isso clique em "Instalar MPLAB 8.20" no aplicativo executado quando o CD é inserido.



Figura 9.2: Tela inicial.

A primeira tela é apenas uma tela de boas vindas. Nela são recomendadas as seguintes ações antes de instalar o MPLAB:

- Fechar qualquer outro aplicativo
- Desinstalar qualquer versão do MPLAB já existente (em especial as 6.x)
- Desabilitar o anti-vírus.

Tomadas essas precauções, clique em **Next**.

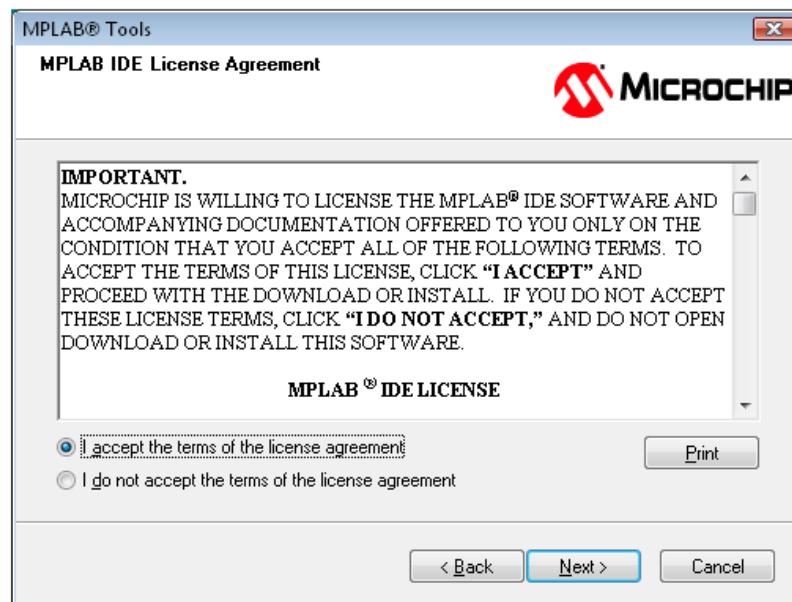


Figura 9.3: Contrato de licença do MPLAB.

Esta tela apresenta o contrato de licença do MPLAB. Leia-o atentamente e, estando de acordo, marque a opção "**I accept the terms of the license agreement**" e clique em **Next**.

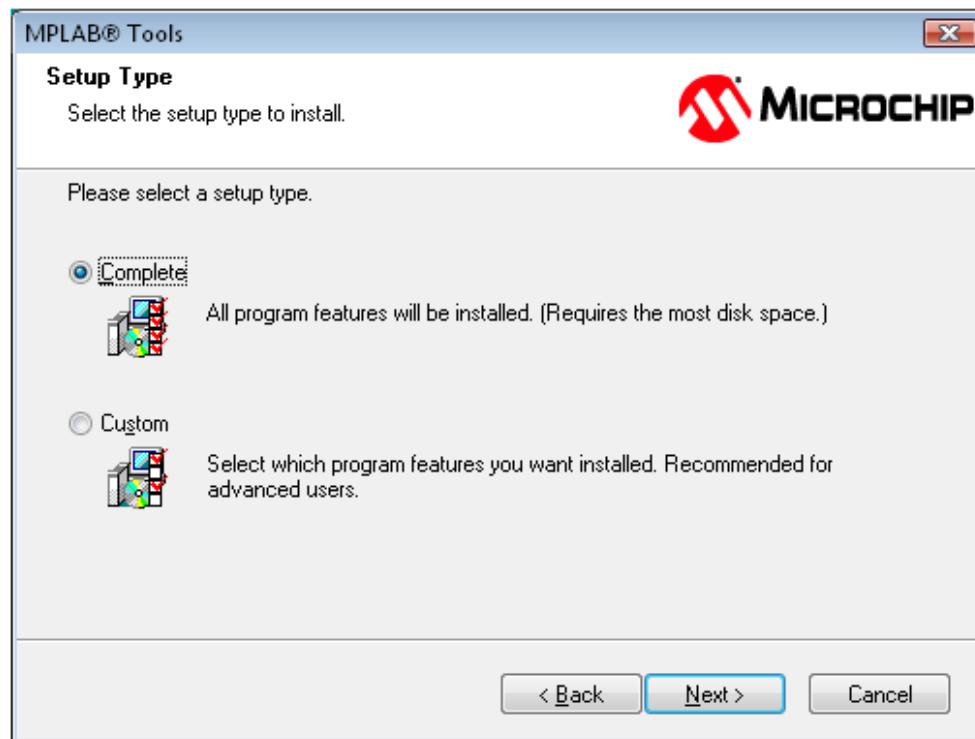


Figura 9.4: Seleção de versão.

Nesta tela é possível escolher entre a versão completa e personalizada. Recomendamos optar por Completa e clicar em Next.

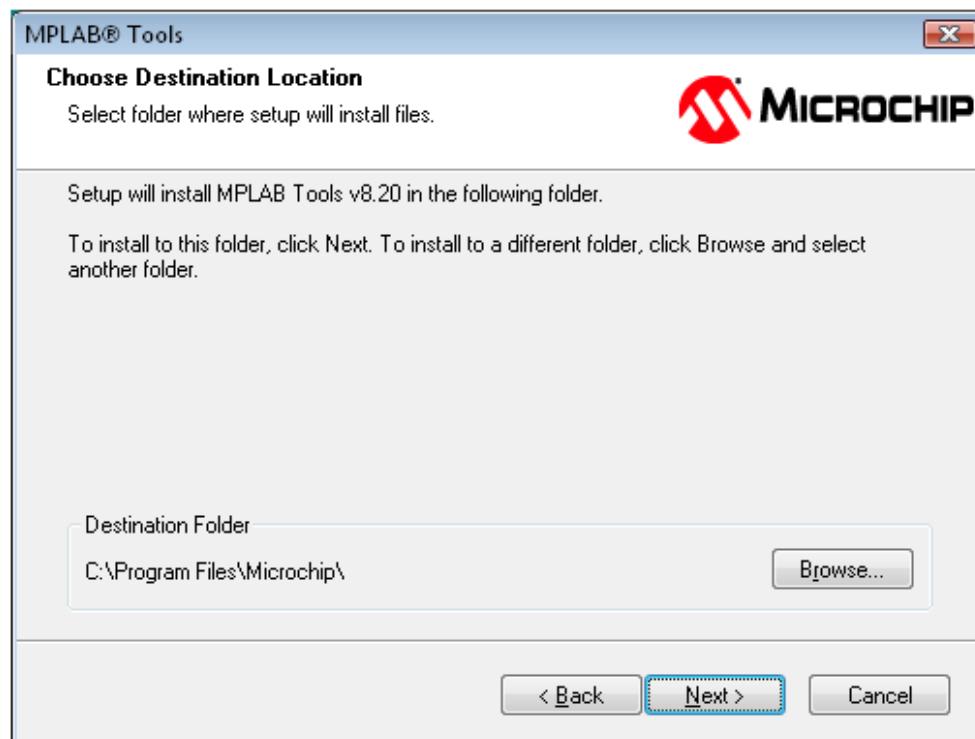


Figura 9.5: Pasta de instalação do MPLAB.

Nesta tela é possível escolher a pasta de instalação do MPLAB, que não recomendamos. Clique em Next.

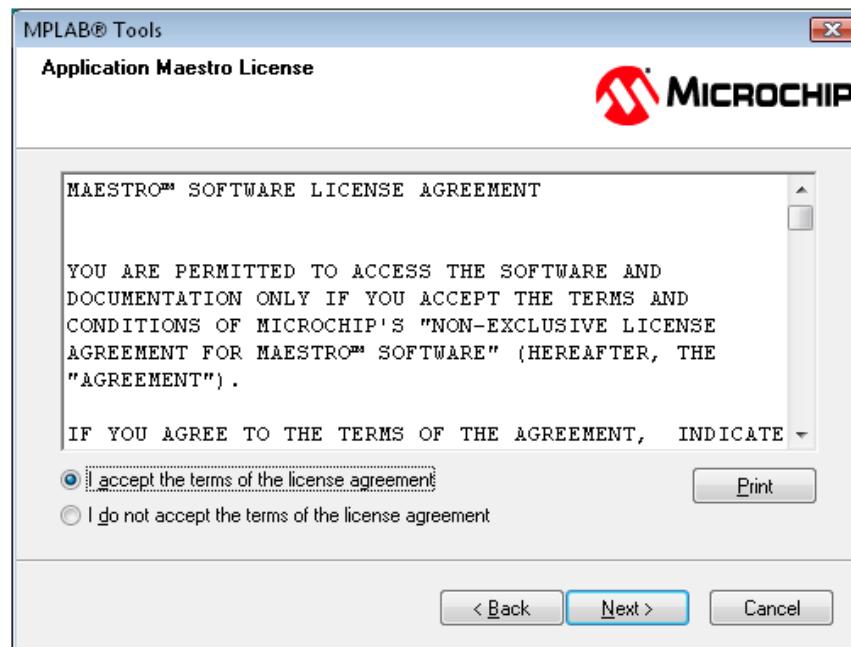


Figura 9.6: Contrato de licença do Application Maestro.

Esta tela apresenta o contrato de licença do Application Maestro (um aplicativo que é instalado juntamente como o MPLAB). Leia-o atentamente e, estando de acordo, marque a opção **"I accept the terms of the license agreement"** e clique em Next.

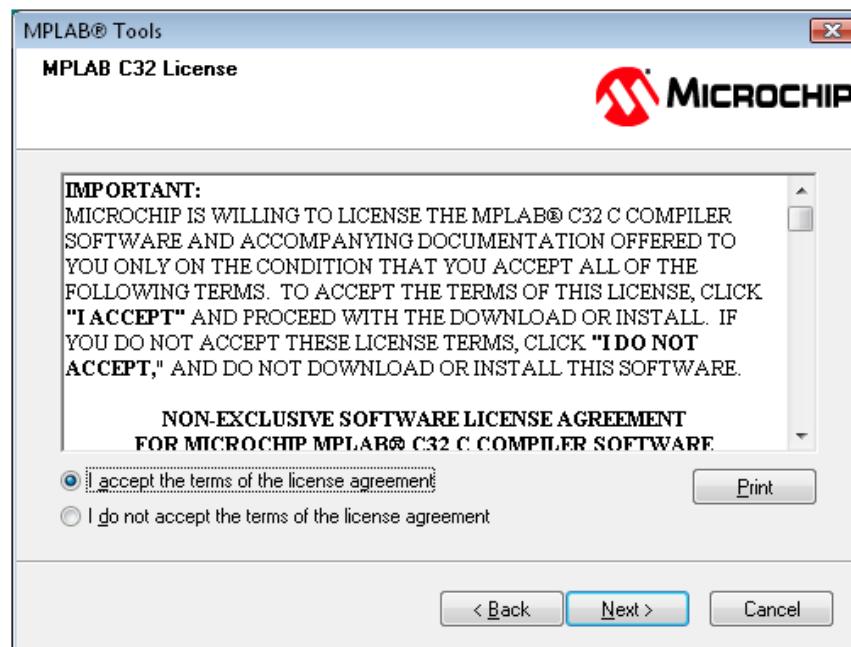


Figura 9.7: Contrato de licença do compilador C32.

Esta tela apresenta o contrato de licença do compilador C32. Leia-o atentamente e, estando de acordo, marque a opção **"I accept the terms of the license agreement"** e clique em Next.

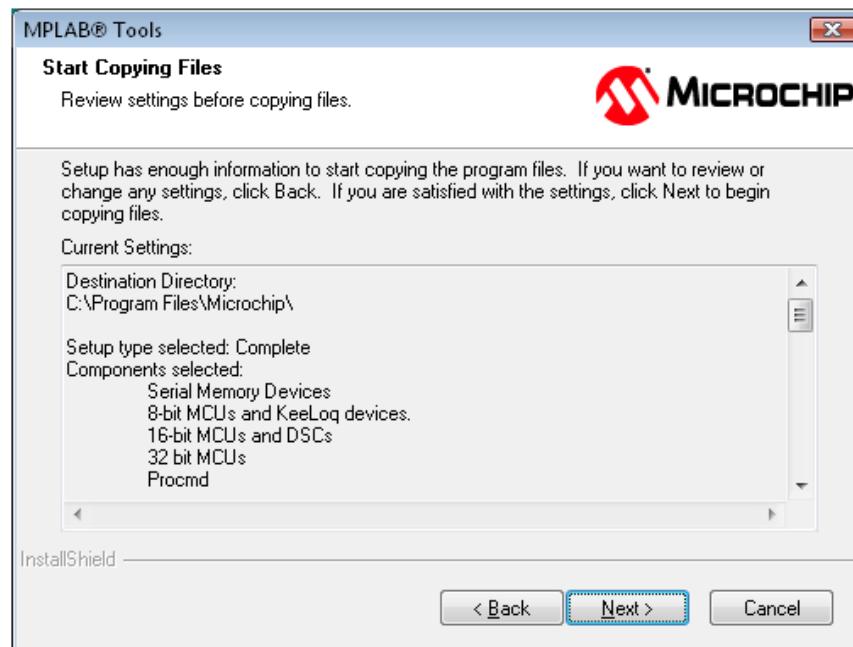


Figura 9.8: Resumo da instalação .

Esta tela apresenta um resumo da instalação antes que essa seja iniciada. Estando tudo correto, clique em **Next** para iniciar a instalação.

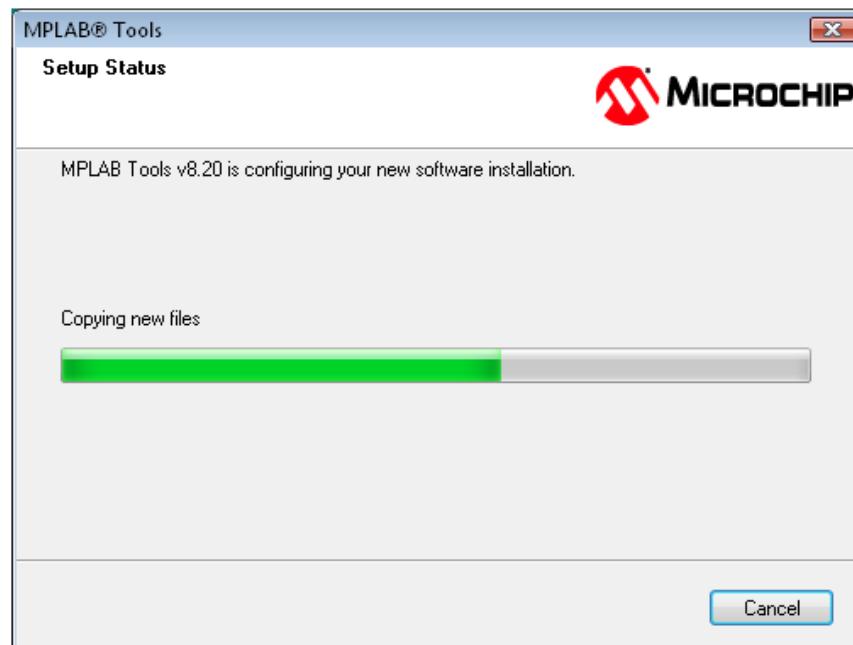


Figura 9.9: Andamento da instalação.

Nesta tela é apresentado o andamento da instalação. Esse processo pode demorar alguns minutos.

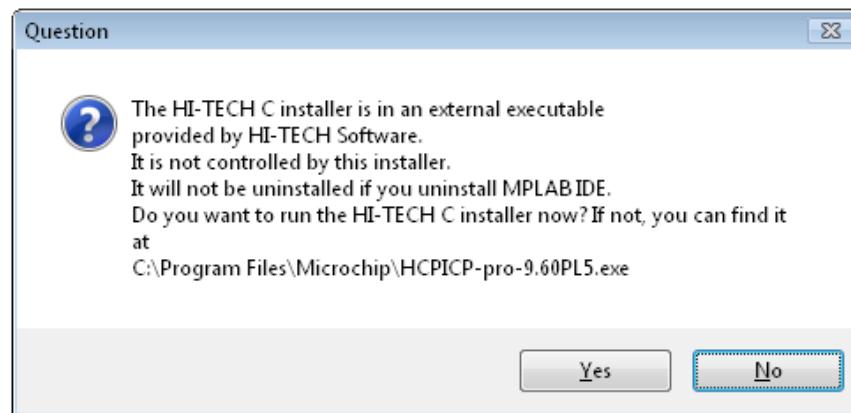


Figura 9.10: Instalação do compilador HI-TECH C.

Essa tela aparece após a copia dos arquivos e questiona sobre a instalação do compilador HI-TECH C, que é um compilador (versão demo) associado ao MPLAB. Clique **Yes** caso queira instalá-lo e **No** caso contrário.

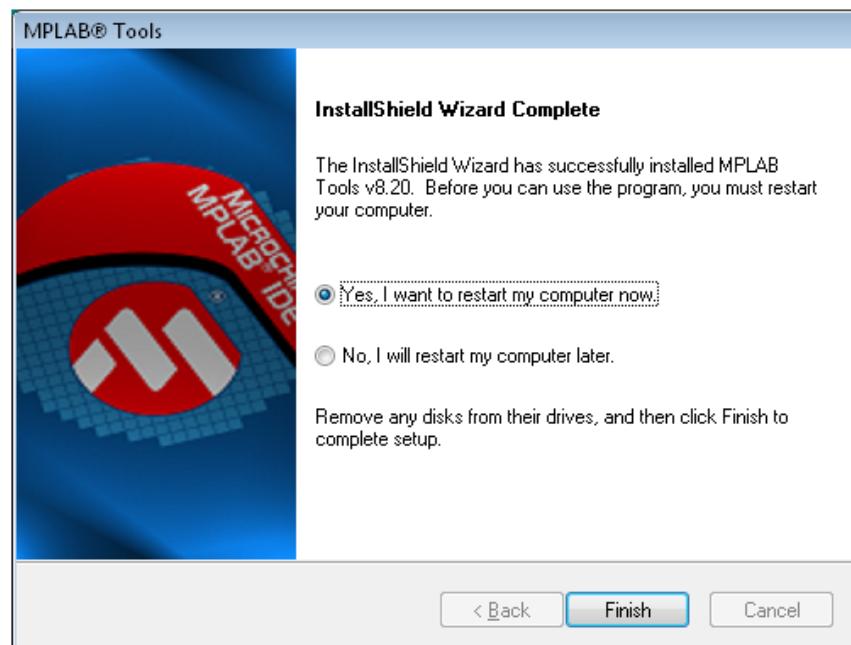
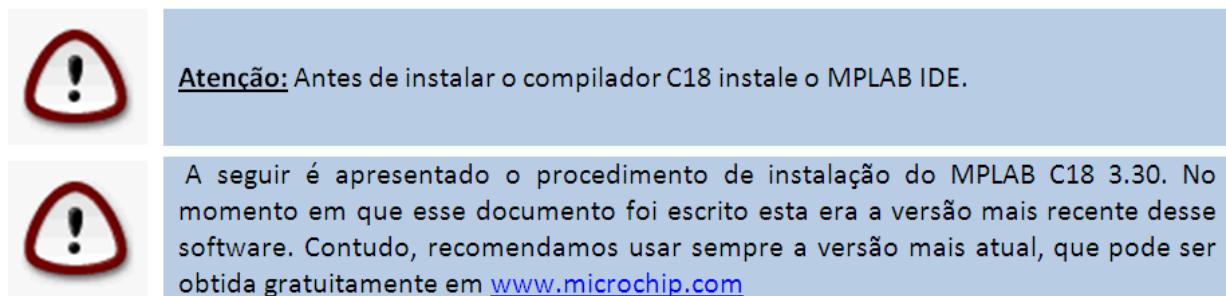


Figura 9.11: Tela de conclusão da instalação.

Ao final da instalação aparecerá a tela acima. É necessário reiniciar o computador para que o MPLAB possa ser executado normalmente. Para isso, marque a opção "**Yes, I want to restart my computer now**" e clique em **Finish**.

Instalando o Compilador C18



Executando o programa de instalação do C18 a primeira tela que aparece é apresentada a seguir.

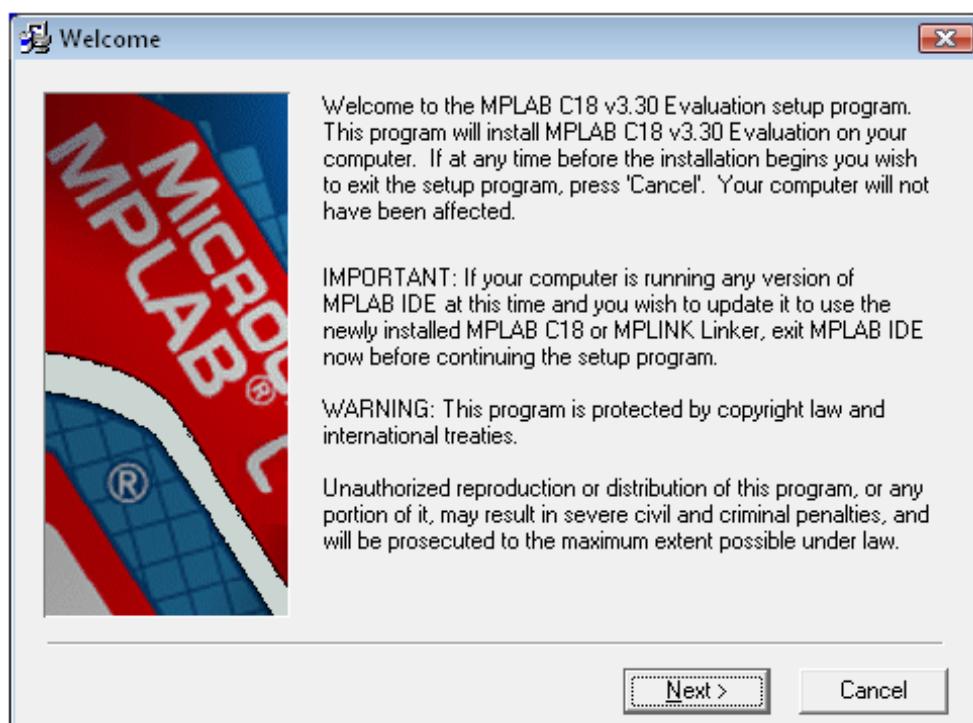


Figura 9.12: Tela inicial de instalação do C18.

Conforme a orientação apresentada, o MPLAB IDE deve ser fechado antes da instalação do C18. Clique em **Next >** para a próxima tela.

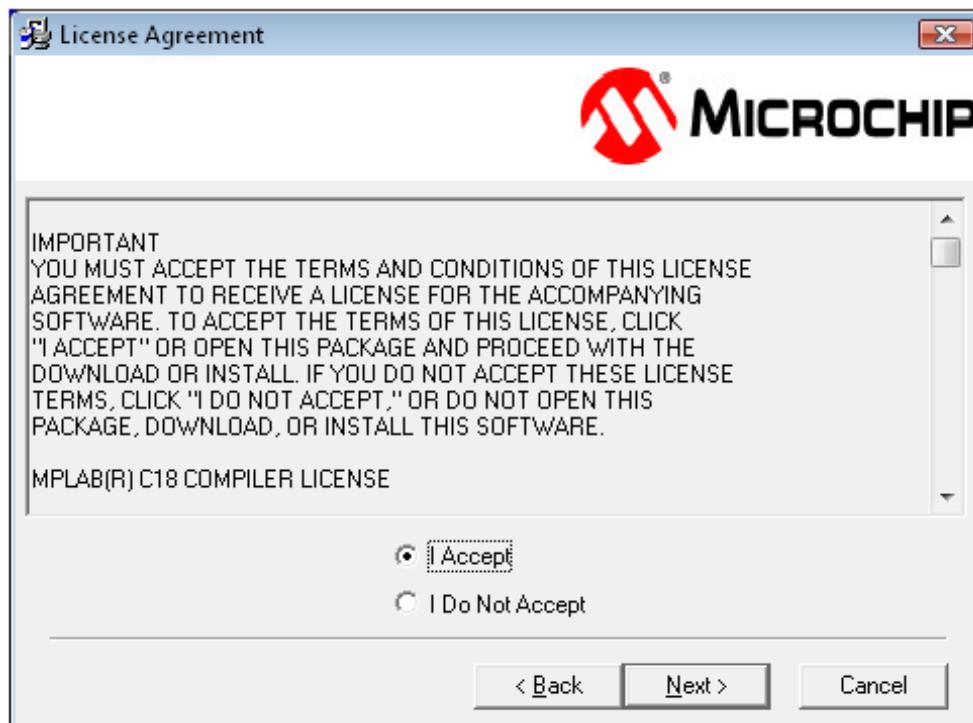


Figura 9.13: Contrato de licença.

Uma vez lido o contrato de licença e aceito seus termos marque a opção ”**I Accept**” e clique em **Next**.

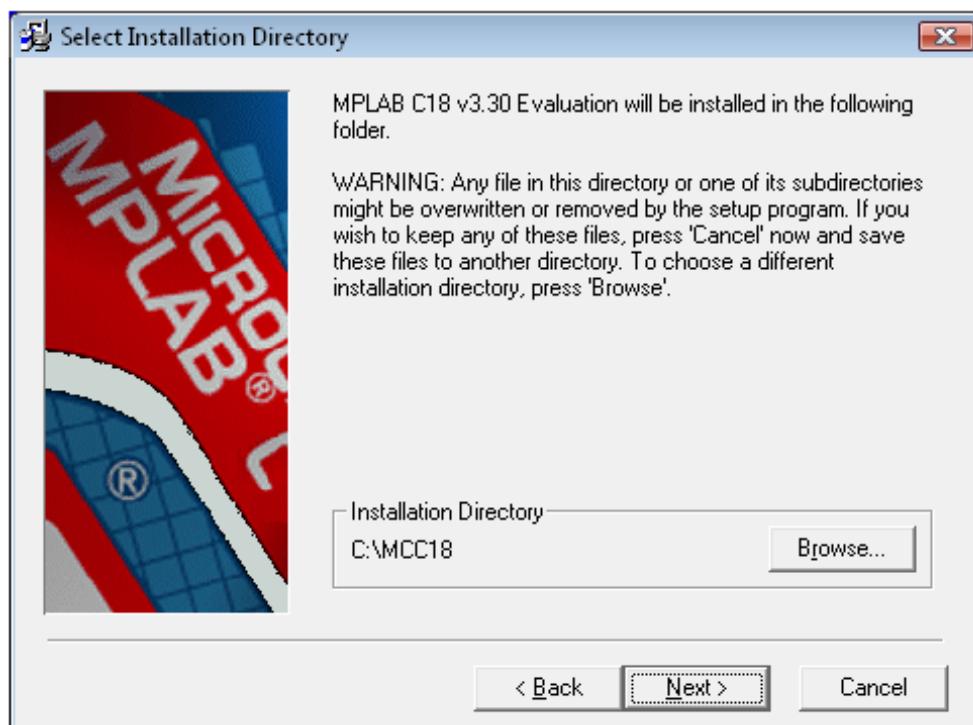


Figura 9.14: Diretório de instalação

Na tela acima é possível alterar o diretório de instalação do C18 clicando no botão **Browse...**, apesar de não recomendarmos que isso seja feito. Clique em **Next**.

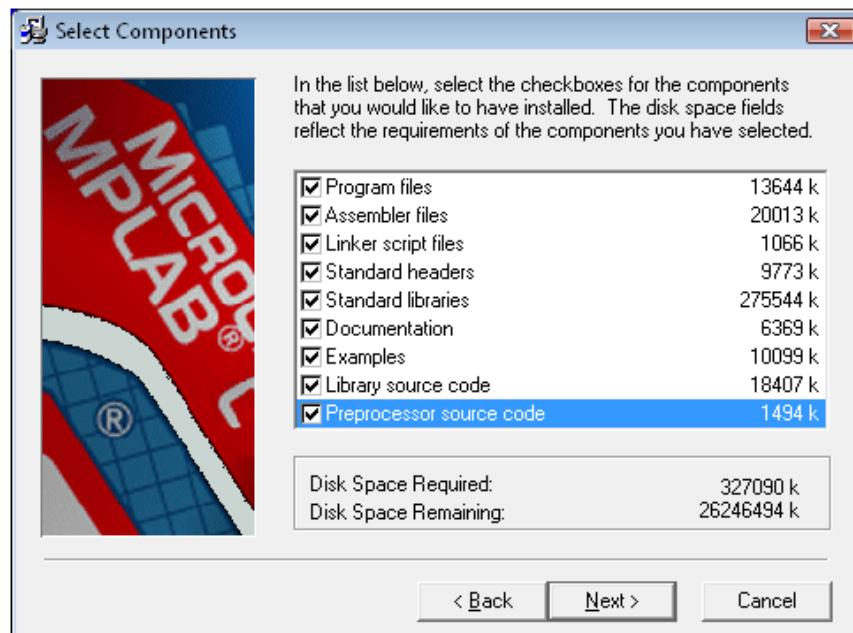


Figura 9.15: Componentes do pacote do compilador.

A tela acima permite escolher os componentes do pacote do compilador a serem instalados. Recomendamos que todos os campos sejam marcados. Feito isso, clique em Next.

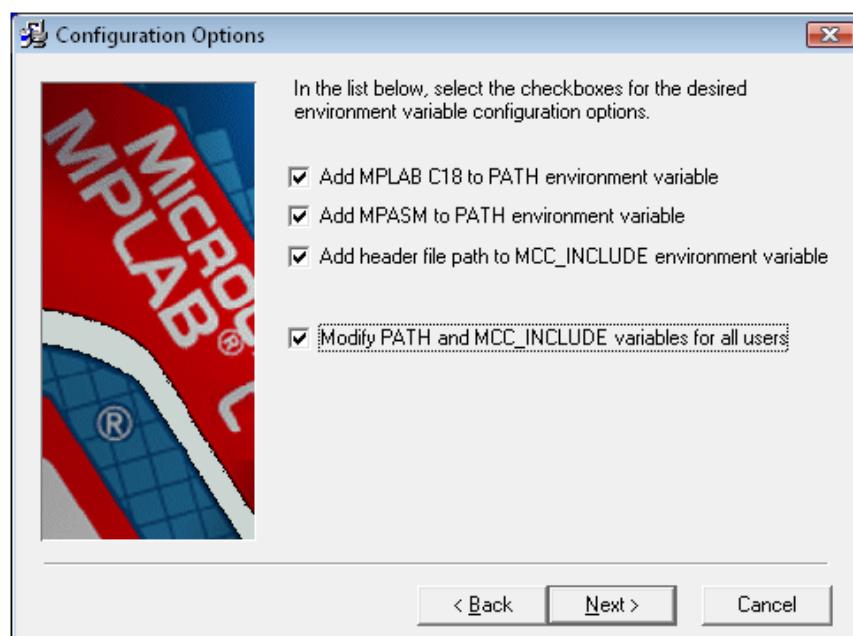


Figura 9.16: Campos a serem selecionados.

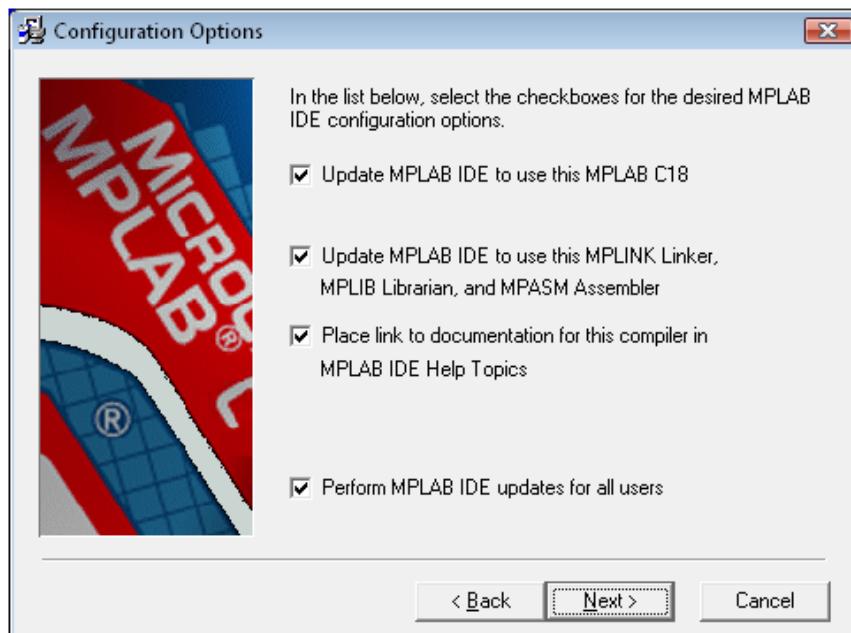


Figura 9.17: Campos a serem selecionados 2.

É importante marcar **todos** os campos nas telas acima para que o MPLAB IDE seja atualizado para trabalhar com o compilador C18. Após marcar todos os campos, clique em **Next**.

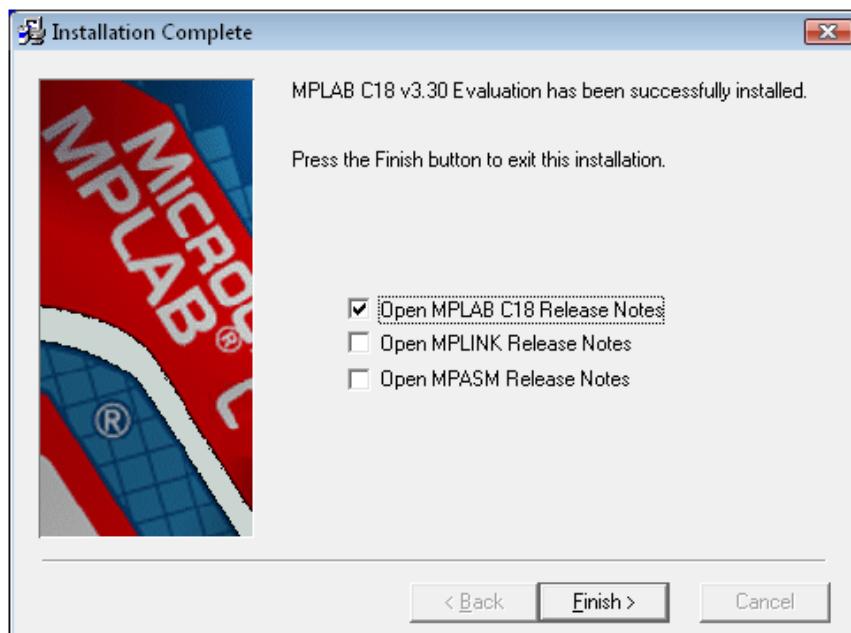


Figura 9.18: Tela inicio de instalação.

Clicando em Next na tela acima a instalação tem início. Esse processo pode levar algum tempo. Uma vez terminado esse processo a tela abaixo é apresentada. Clique em Finish para terminar o processo.

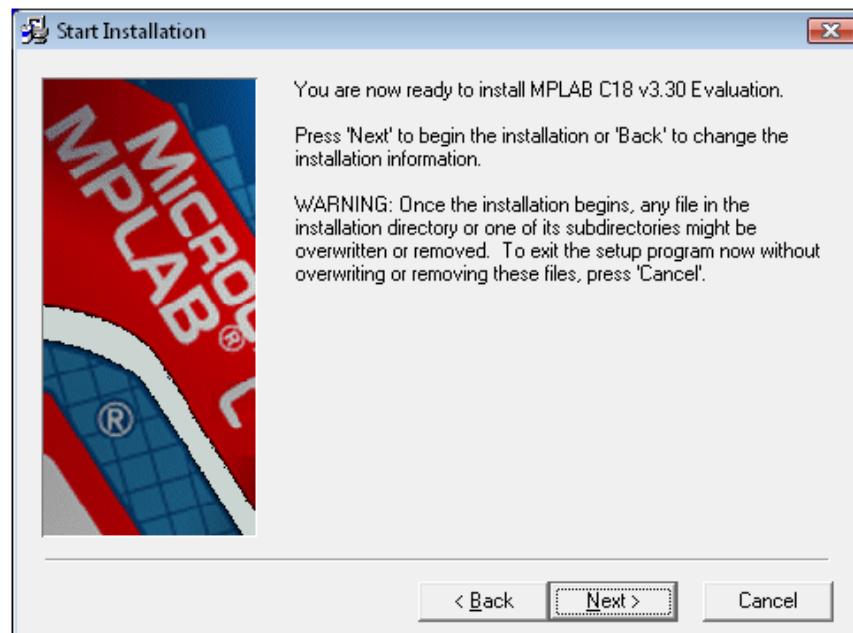


Figura 9.19: Tela de conclusão.

9.2.3 Configurações após a instalação

Configuração dos drivers do XICD

O XM118 é acompanhado de um CD contendo os programas utilizados para o desenvolvimento e ainda os drivers necessários para a instalação do hardware do XICD.

Este item do manual visa continuar os esclarecimentos quanto a instalação dos drivers no primeiro uso do sistema.

Então, quando o equipamento é conectado ao computador através do cabo USB, isto é detectado pelo computador mostrando a seguinte tela de identificação do dispositivo:

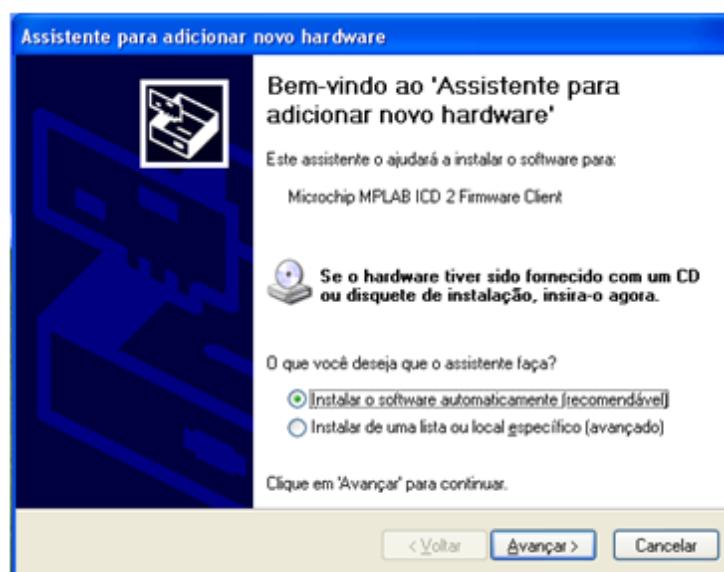


Figura 9.20: Assistente para adicionar novo hardware.

Para a instalação dos drivers do XICD, podemos proceder de pelo menos duas formas: A

primeira é fazer com que o sistema operacional procure automaticamente os drivers do novo dispositivo, deixando a opção selecionada como acima. Isso funciona bem quando há uma conexão internet, pois ele procurará os drivers automaticamente.

A segunda forma é procurar localmente os drivers. Para isso, temos duas localizações diferentes para os drivers. Para adicionar localmente devemos selecionar a opção "Instalar de uma lista ou local específico" e colocar o caminho onde serão encontrados os drivers.

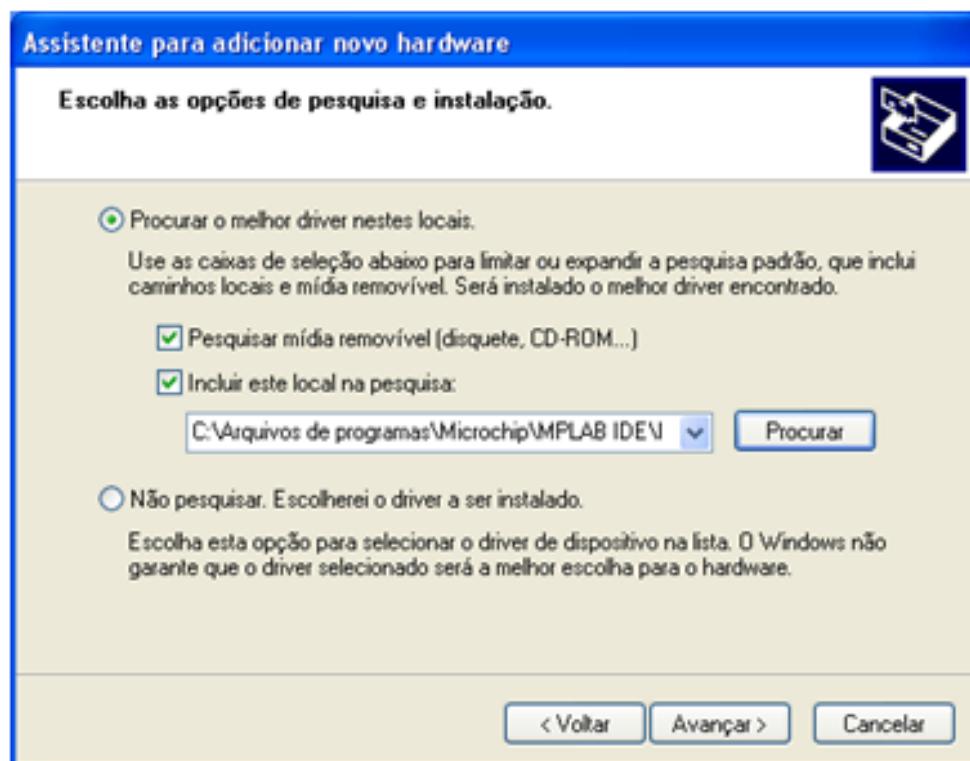


Figura 9.21: Procura avançada de drivers.

Conforme já dito, podemos encontrar os drivers em alguns locais distintos. Um destes lugares seria o diretório onde foi instalado o próprio MPLAB que pode ser visto como exemplo na figura anterior.

O caminho padrão para a procura dos drivers do XICD no MPLAB é: "C:\Arquivos de programas\Microchip\MPLAB IDE\ICD2\Drivers". Neste local teremos os drivers do XICD se você instalou o MPLAB na pasta padrão usando o sistema operacional em português. Outro local onde podemos localizar os drivers seria no CD que acompanha o produto. A localização dele dentro do CD seria na pasta programas e na subpasta drivers.

Depois de instalado o mesmo está pronto para uso e a seguinte janela aparecerá.

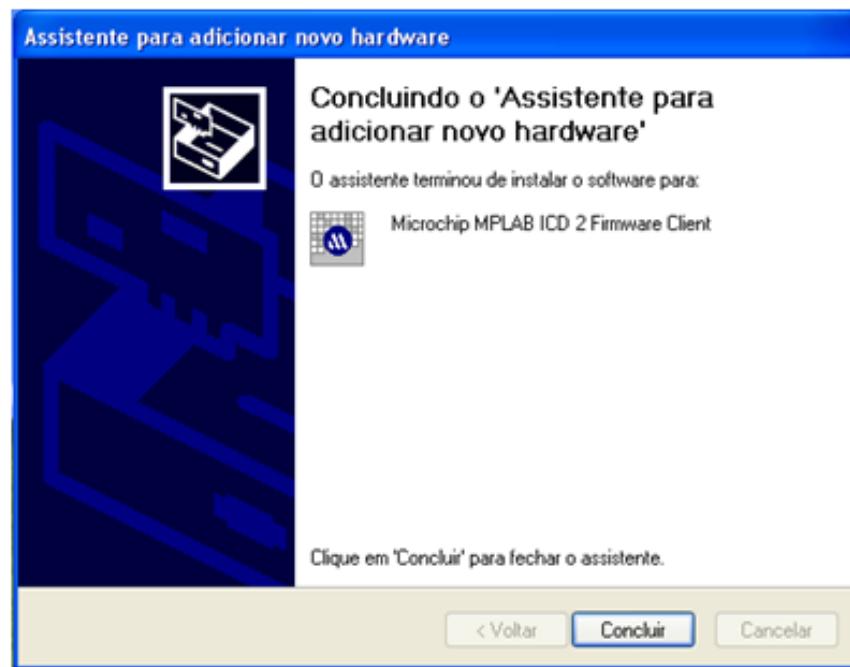


Figura 9.22: Localização do painel de configuração

Configuração do IDE MPLAB para uso do XICD

Como já foi dito anteriormente, o XICD pode ser usado como ferramenta de programação de microcontroladores e também de depuração em tempo real. Para ambas as operações são necessárias o preparo do MPLAB para usar o XICD para cada uma destas funções. Essas funções são chamadas de modos de programação (Programming mode) e modo de depuração (Debugger mode) que são descritos aqui.

Para o uso do XICD para ambos os modos de operação é necessária a sua configuração através de um assistente de configuração do XICD, que define alguns parâmetros para o seu funcionamento. Abaixo temos um passo a passo de como proceder na instalação deste modelo de XICD.

Tanto em modo de depuração quanto em modo de programação é necessária a configuração de qual equipamento será usado pelo MPLAB para acesso ao microcontrolador, que é feita através dos menus "Debugger" e "Programmer".

Depois que o usuário definir que tipo de operação fará, é só selecionar o ICD2 no menu indicado, no nosso caso aqui estamos usando o modo de programação:

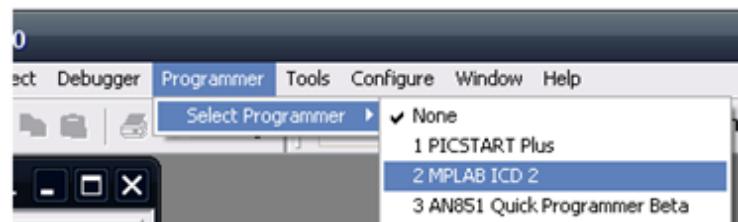


Figura 9.23: Selezionando XICD como ferramenta de programação

Depois de selecionado o ICD2 como ferramenta de programação vem finalmente à configuração do mesmo, onde temos um wizard para facilitar esta configuração.



Figura 9.24: Tela inicial do wizard para configuração do XICD

Pressionando a tecla avançar, temos a próxima janela que permite escolher em que porta externa está o dispositivo. Sendo o nosso XICD com comunicação USB, devemos escolher a opção USB na opção "Com Port".



Figura 9.25: Opção de porta de comunicação do XICD

Em seguida, devemos especificar se o target (placa onde está o microcontrolador a ser gravado) será alimentado pelo XICD ou terá alimentação própria. É recomendado que somente circuitos com baixo consumo sejam colocados para serem alimentados pelo XICD já que sua alimentação vem da porta USB e consequentemente o target será alimentado pela mesma fonte.

Este tipo de alimentação então deve ser usado com prudência para evitar danos a porta USB. Em uma porta USB padrão temos disponível por porta somente 500mA aproximadamente. Assim, o consumo do XICD mais o consumo do target não podem superar o valor máximo da porta USB.

No XICD, no termo desta configuração, possui uma forma de sinalizar se o hardware do XICD está fornecendo a energia ou se é o target.

Abaixo temos a janela de configuração que permite configurar o XICD para fornecer alimentação para o target ou se o mesmo tem sua própria fonte de alimentação.

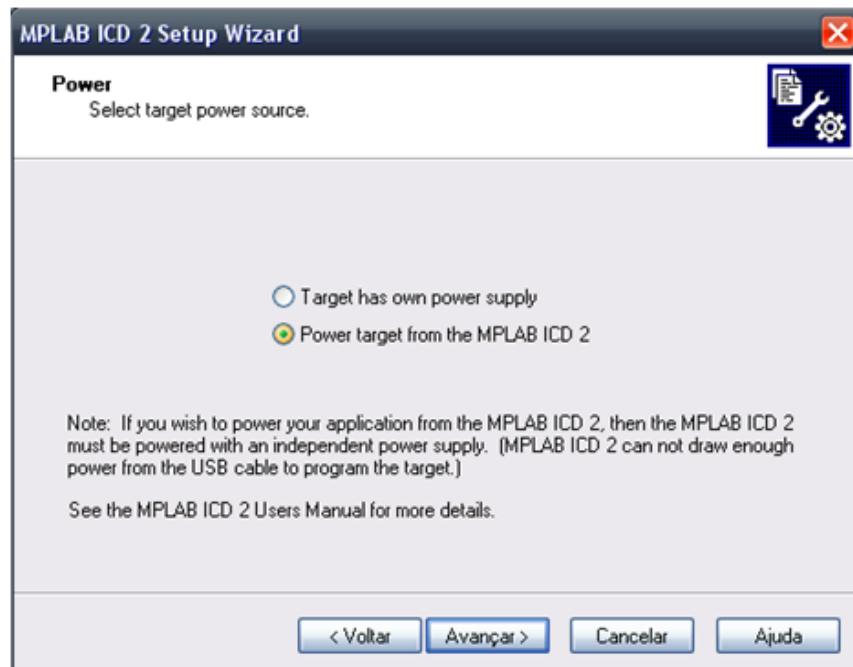


Figura 9.26: Fonte de alimentação do target.



Atenção: No caso do uso com o kit XM118 a opção deve sempre ser utilizar a alimentação do target (*Target has own power supply*).

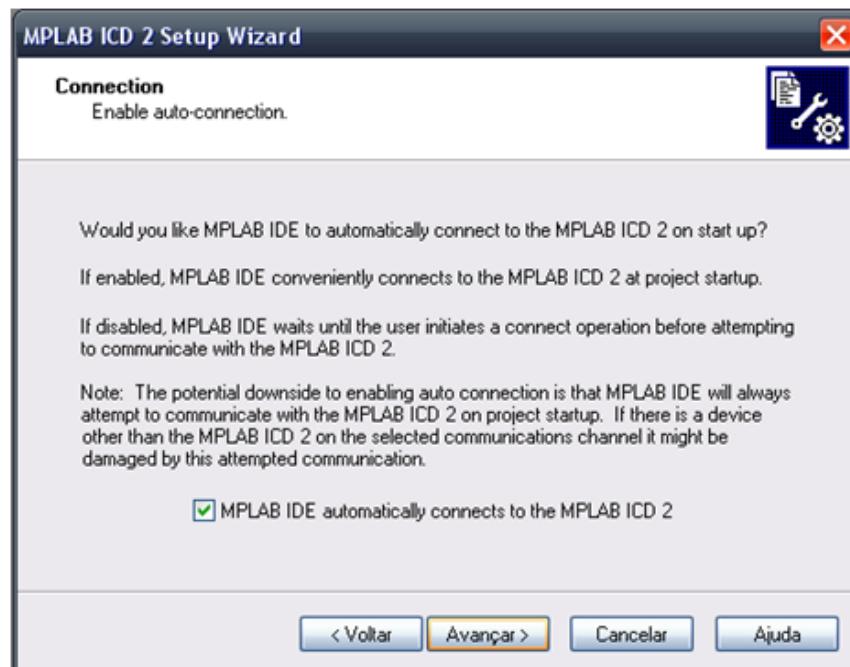


Figura 9.27: Configuração da conexão automática

A janela acima define se o XICD tentará se conectar ao target de forma automática ao ser selecionado no programmer ou debugger e a próxima definirá se será de forma automática o carregamento do sistema operacional para o XICD. Onde cada sistema operacional é diferente para a família de microcontrolador utilizada e necessária para a programação do target.

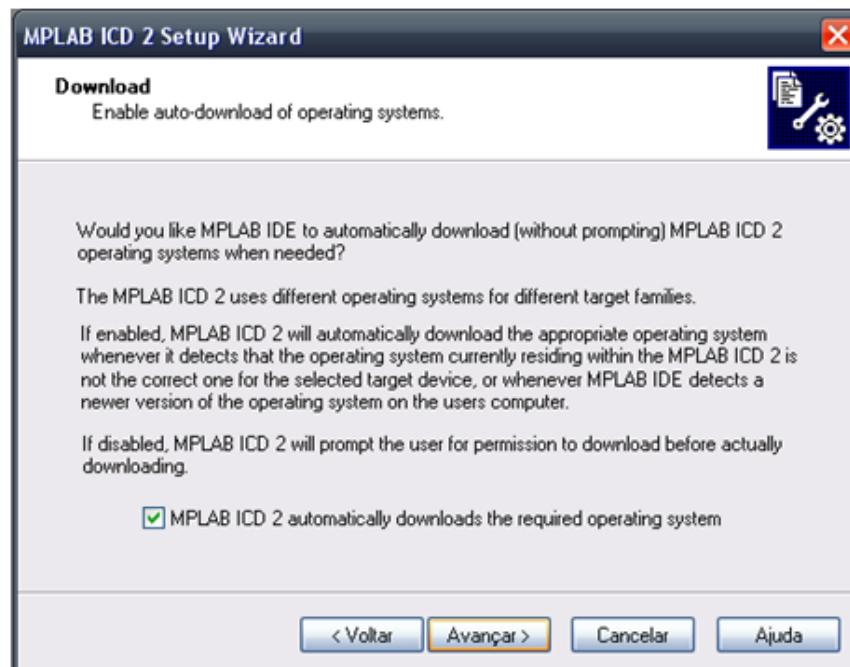


Figura 9.28: Configuração do download automático do sistema operacional do XICD



Figura 9.29: Resumo das configurações do XICD

9.3 Hardware do kit XM118

A Figura abaixo apresenta a serigrafia da placa do Kit Educacional XM118.

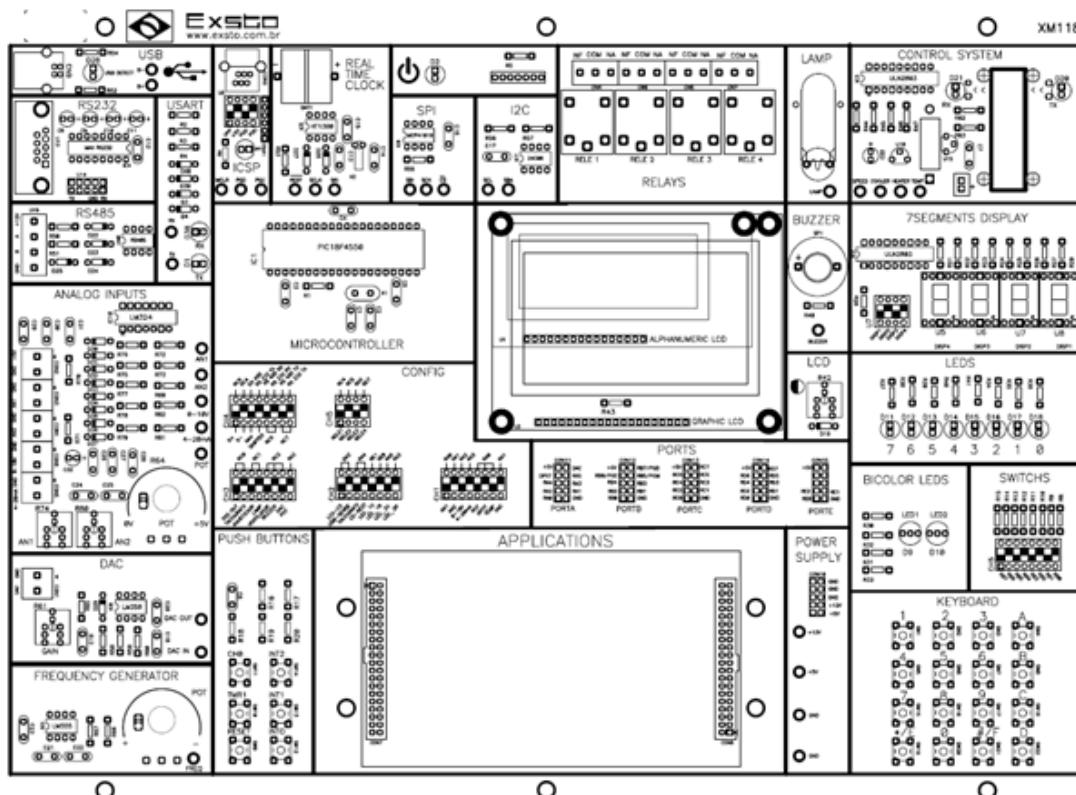


Figura 9.30: Serigrafia do Kit XM118

O hardware do Kit Educacional XM118 foi concebido para permitir a maior flexibilidade possível nas ligações. Para os pinos com diversas funções foram ligados a chaves dip switch, de forma a permitir a conexão com diferentes circuitos de aplicação. Para configurar corretamente o hardware para suas necessidades leia as tabelas indicativas das funções das chaves.

Muitos dos circuitos de aplicação possuem pontos de testes (test points) que facilitam a medida de seus principais sinais. A seguir são apresentadas detalhadamente as partes do circuito. Elas fazem referência ao esquema elétrico que se encontra no anexo A deste manual e em forma PDF no CD.

9.3.1 Fonte de alimentação

O kit Educacional CPLD é equipado com duas fontes chaveadas independentes de +5VDC/3A e +12VDC/1A. A tensão de entrada dessas fontes é de 90 a 240 VAC (fullrange), em 50 ou 60 Hz.

Estas fontes têm proteção de saída contra curto-circuito e sobrecarga que desativa as saídas, retornando a operação normal assim que o problema deixa de existir. Adicionalmente, existe uma proteção entrada contra surtos de tensão. O bastidor metálico é aterrado, para maior proteção dos usuários.



Atenção: os recursos de proteção só são eficazes se o terceiro pino do cabo tripolar estiver conectado a um aterramento de qualidade. A Exsto tecnologia não se responsabiliza por danos ou acidentes ocasionados por baixa qualidade ou inexistência de aterramento.

As fontes de alimentação estão disponíveis para o usuário em conectores apropriados. Seu uso, contudo, deve respeitar a corrente máxima disponível.

9.3.2 Microcontrolador PIC18F4550

O kit XM118 foi baseado no PIC18F4550. Trata-se de um microcontrolador de 8 bits de alto desempenho, com uma série de periféricos e recursos que o tornam bastante completo, permitindo através de seu estudo explorar vários recursos comumente encontrados em microcontroladores. Em especial, este possui um periférico USB onde sua comunicação operando como device , permite sua conexão a um computador ou outro dispositivo host.

A tabela a seguir traz as principais características do PIC18F4550.

Característica	PIC18F4520
Frequência de Operação	DC a 48MHz
Memória de programa	32768 bytes
Memória de dados RAM	2048 bytes
Memória de dados EEPROM	256 bytes
Fontes de interrupção	20
Terminais de I/O	36
Temporizadores/Contadores	4
CCP	1
ECCP	1
Comunicação Serial	MSSP EUSART
Comunicação USB	Sim
Comunicação Paralela	SPP
Conversor analógico para digital	10 bits, 13 canais
Detector de tensão programável	1
Conjunto de instruções	75 convencionais + 8 do modo entendido

Tabela 9.1: Características dos PIC18F4550

A pinagem desse componente é apresentada na figura abaixo.

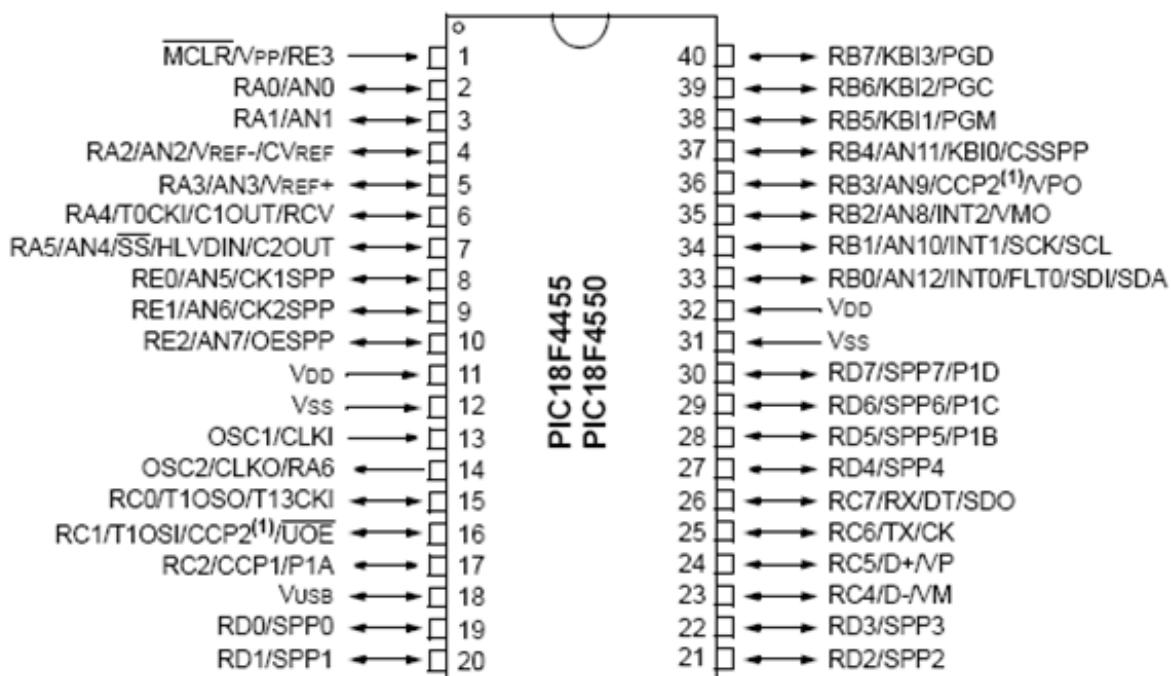


Figura 9.31: Pinagem do PIC18F4550

Para mais dados técnicos, consulte o manual do componente que se encontra no CD do produto.

Além do PIC18F4550 outros microcontroladores PIC de 40 pinos podem ser usados na placa, desde que sejam compatíveis pino a pino (o que é verdade para a maioria dos pinos de componentes com o mesmo encapsulamento). Alguns modelos muito populares suportados com algumas poucas restrições são o PIC18F4520 e o PIC16F877A.

9.3.3 Conectores para acesso de I/O (PORTS)

Imediatamente acima do conector de expansão de aplicação existem conectores que dão acesso aos portais do microcontrolador. A serigrafia da placa indica a disposição dos pinos.



Atenção: quando usar um pino do microcontrolador para aplicações externas verifique o circuito que se encontra associado a este pino no esquema elétrico. Quando possível, desligue as chaves de configuração associadas ao pino em questão.

9.3.4 ICD e conector ICSP

O kit XM118 é equipado com um gravador/depurador X-ICD2, que fica embutido dentro do gabinete do kit. Esse equipamento é totalmente compatível com o MPLAB e com o gravador ICD-2 da Microchip, sendo reconhecido como um ICD-2 dentro do MPLAB.

O XICD-2 está conectado ao XM118 e consequentemente ao microcontrolador nele instalado através da chave U8, essa chave permite desconectar os pinos do microcontrolador do XICD quando necessário, já que os pinos de gravação são pinos de I/O usados em algumas aplicações e o pino de reset.



Atenção: para realizar corretamente o processo de gravação certifique-se de que todas as chaves de U8 estão na posição ON.

Além disso, o conector CN9 permite gravar/depurar outros componentes externos, seja através do header (placa para gravação de componente) que acompanha o kit, seja em placas desenvolvidas pelo próprio usuário. Para isso, um cabo RJ12-RJ12 acompanha o produto.



Atenção: quando for gravar ou depurar um microcontrolador externo todas as chaves de U8 devem estar em OFF, impedindo assim conflito com o microcontrolador instalado no XM118.

Como o hardware do XICD não precisa de nenhuma intervenção do usuário ele não será abordado aqui. Contudo é necessário verificar algumas características para que o produto possa funcionar corretamente em qualquer tipo de circuito alvo. Algumas dessas características podem ser citadas como a sequência de pinos do conector e os elementos que estão ligados nas vias de gravação/depuração do microcontrolador.

No XICD, temos quatro leds indicadores que permitem analisar o atual funcionamento do XICD, cuja função é descrita pela figura abaixo (esses LEDs são do XICD e portanto estão dentro do bastidor, não estando acessíveis para o usuário, servindo apenas para manutenção):

FUNÇÃO/ LED's	
Target/Verde	Indica se a placa alvo está sendo alimentada pelo XICD
Busy/Amarelo	Indica que o XICD está executando alguma tarefa
Error/Vermelho	Indica se aconteceu algum erro na comunicação com o XICD
USB/Verde	Indica que o XICD está alimentado e pronto para uso

Tabela 9.2: Esquema de pinos do XICD

Além desses LEDs internos, existe na placa um LED AZUL que indica quando o microcontrolador está sendo acessado. Observando esse LED podemos detectar se existem problemas na gravação.

O primeiro ponto a ser abordado é a sequência de pinos do XICD. O cabo que acompanha o produto se limita a fazer a ligação entre o XICD e a placa reader. Entretanto, pode ser necessário fazer a ligação do XICD a outro tipo de placa onde o microcontrolador não pode ser colocado no reader ou ainda por ser necessária a depuração o microcontrolador tem que estar funcionando na placa de desenvolvimento.

Como a sequência dos pinos deve ser respeitada para que o funcionamento seja feito, abaixo temos uma figura que mostra a forma com que cada pino está ligado no conector do XICD.

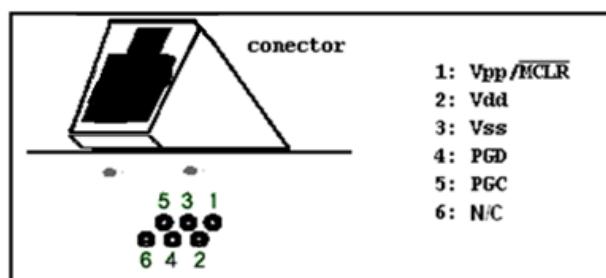


Figura 9.32: Esquema de pinos do XICD



Observação: A inversão da seqüência de pinos pode causar danos ao XICD.
Certifique-se que os pinos estejam devidamente conectados caso esteja construindo a placa alvo e esteja atenta a serigrafia das placas para correta conexão.

Na próxima figura é apresentada a conexão entre o XICD e a placa de desenvolvimento com uso do cabo telefônico adaptado. O diagrama também mostra a conexão entre o conector e o PIC na placa de desenvolvimento. Recomenda-se também que se use um resistor de pull-up, se necessário, entre VPP e VDD de modo que zerando a alimentação possa-se reiniciar o PIC alvo. O valor deste resistor é usualmente de 10KΩ.

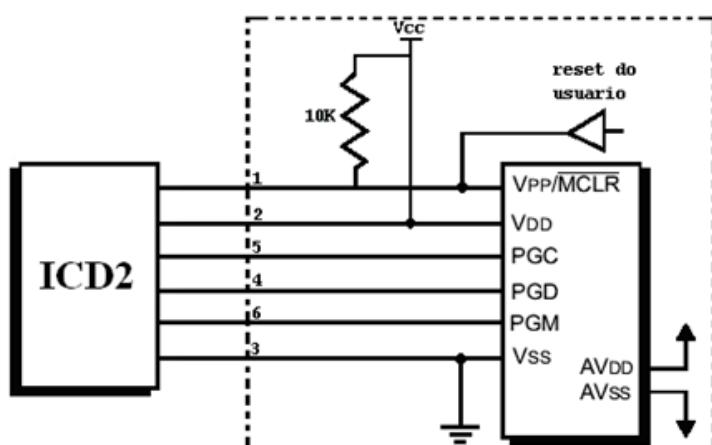


Figura 9.33: Ligação do XICD

Embora o pino 2 (VDD) possa suprir uma quantidade limitada de corrente para o alvo sobre

certas condições, nas operações seguintes os pinos 2 e 3 serão omitidos. Estes pinos são apresentados no diagrama, mas apenas três ligações são relevantes para operação do XICD: VPP, PGC e PGD.

Nos diagramas seguintes os pinos de VDD e VSS (GND) serão ignorados. Mas esteja ciente de que o VDD da placa de desenvolvimento é também usado para alimentar os drivers de saída do XICD. É importante lembrar que nem todos PIC's tem entradas para alimentações distintas, mas se elas existirem no PIC que se deseja operar, todas devem estar conectadas corretamente para que o XICD opere corretamente. A interconexão é muito simples, qualquer problema comum é frequentemente causado por outras conexões ou componentes nas vias de uso do XICD que interferem com seu funcionamento, como os apresentados no capítulo abaixo.

Circuitos que impedirão o funcionamento correto do XICD

Na figura abaixo são apresentados alguns procedimentos que impedirão o funcionamento correto do XICD.

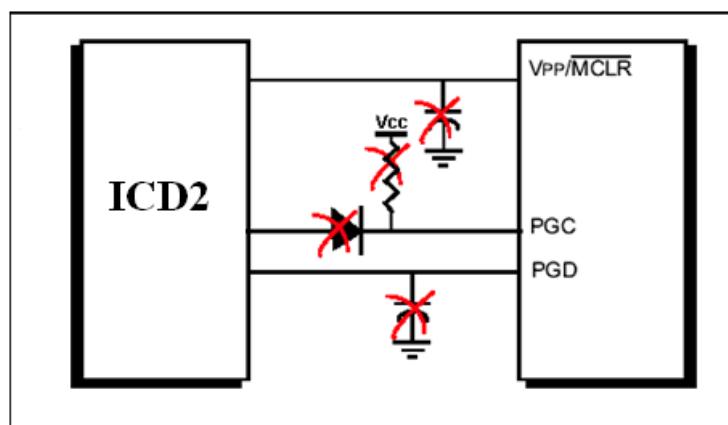


Figura 9.34: Ligação do XICD

Para o projeto de ligação do XICD em uma placa de desenvolvimento siga corretamente as recomendações abaixo:

- Não use resistores de pull-up nos pinos PGC e PGD: eles irão impedir a transição rápida, tanto de dados como do sinal de clock nas operações de gravação e depuração.
- Não use capacitores no pino VPP: eles irão impedir a transição rápida de VPP. Geralmente o resistor de pull-up é suficiente.
- Não use diodos nos pinos PGC e PGD: eles irão impedir a comunicação bidirecional entre a placa e o XICD.

9.3.5 Configuração

O microcontrolador tem um número limitado de pinos para poder interagir com todos os circuitos presentes no kit. Além disso, vários pinos têm mais de uma função, podendo trabalhar de forma diferente com cada circuito a ele associado. Para maximizar o número de aplicações que podem ser realizadas ,se faz necessário atribuir mais de um circuito a cada terminal do componentes e selecionar o circuito a ser usado em cada caso.

Alguns circuitos estão simplesmente ligados em paralelos, como por exemplo, LEDs e LCD, teclado e chaves. Outros são selecionados através de chaves de configuração. A tabela abaixo apresenta as configurações possíveis, indicando o portal do microcontrolador, o sinal utilizado, sua função e direção (em relação o microcontrolador).



Atenção: Nos pinos do microcontrolador que tem mais de uma função associada apenas uma deve ser selecionada por vez.

Portal do microcontrolador	Sinal	Direção	Função
RC0	555_OUT TACÓGRAFO	Entrada Entrada	Sinal do gerador de sinais Sinal do tacógrafo (conta-giros) da ventoinha
RC1	AQUECEDOR LAMP	Saída Saída	Controle da resistência para aquecimento Controle da lâmpada DC
RC2	VENTILADOR BUZZER DAC	Saída Saída Saída	Controle da ventoinha Controle do buzzer Entrada do filtro do DAC do PWM
RA3	POT	Entrada	Sinal analógico do potenciômetro
RA2	LCD_CS USB_SENSE	Saída Entrada	Chip Select do display gráfico Entrada indicadora de conexão com USB
RA4	TACÓGRAFO CS_REST RS_EN	Entrada Saída Saída	Sinal do tacógrafo (conta-giros) da ventoinha Habilitação do MCP41010 e do HT1380 Habilitação de transmissão pela porta RS485
RE1	LED_1G	Saída	Aciona cor verde do LED bicolor 1
RE0	LED_1R	Saída	Aciona cor vermelha do LED bicolor 1
RA5	LED_2G	Saída	Aciona cor verde do LED bicolor 2
RA3	LED_2R	Saída	Aciona cor vermelha do LED bicolor 2
RA0	ANALOG1 POT DPOT TEMP	Entrada Entrada Entrada Entrada	Entrada Analógica AN1 Potenciômetro Potenciômetro digital Sensor de temperatura
RA1	ANALOG2 DAC_OUT	Entrada Entrada	Entrada Analógica AN1 Feedback do conversor D/A
RA2	0-10V	Entrada	Entrada Analógica de 0 a 10 V
RA3	4-20mV	Entrada	Entrada Analógica de 4 a 20 mA
RC4 RC5	D- D+	Bidir Bidir	Comunicação USB
RC6	RS_485_TX RS_232_TX	Saída Saída	Transmissão serial para RS485 Transmissão serial para RS232
RC7	RS_485_RX RS_232_RX	Saída Saída	Recepção serial para RS485 Recepção serial para RS232
RC6	RELÉ_1	Saída	Aciona Relé 1
RC7	RELÉ_2	Saída	Aciona Relé 2
RD6	RELÉ_3	Saída	Aciona Relé 3
RD7	RELÉ_4	Saída	Aciona Relé 4

Tabela 9.3: Configuração das chaves.

A tabela abaixo mostra a configuração dos dip switch para cada circuito.

Portal do microcontrolador	Sinal	Chave	Dip Switch
PWM e Timers			
RC0	555_OUT	1	CH3
	TACOGRAFO	2	
RC1	AQUECEDOR	3	
	LAMP	4	
RC2	VENTILADOR	5	
	BUZZER	6	
RA3	DAC	7	
	POT	8	
Periféricos Diversos			
RA2	LCD_CS	1	CH2
	USB_SENSE	2	
RA4	TACOGRAFO	3	
	CS_REST	4	
RE1	LED_1G	5	
	LED_1R	6	
RE0	LED_2G	7	
	LED_2R	8	
Periféricos Analógicos			
RA0	ANALOG1	1	CH1
	ANALOG2	2	
RA2	0-10V	3	
	4-20mA	4	
RA3	POT	5	
	DPOT	6	
RA0	TEMP	7	
	DAC_OUT	8	
Portas Seriais			
RC4	D	1	CH4
	D	2	
RA4	RS_EN	3	
	Snifer	4	
-	RS_485_TX	5	
	RS_232_TX	6	
RC6	RS_485_RX	7	
	RS_232_RX	8	
Relés			
RC6	RELÉ_1	1	CH5
	RELÉ_2	2	
RD6	RELÉ_3	3	
	RELÉ_4	4	

Tabela 9.4: Configuração do dip switch

9.3.6 Dispositivos de entrada



Atenção: Como os pinos de gravação e depuração são RB7(PGC), RB6(PGD) e VPP/MCLR/RE3 (VPP) caso a aplicação use esses pinos, os mesmos devem ser desconectados do XICD através das chaves de U8. Pelos mesmos motivos, estes pinos não estarão disponíveis para uso se estiver sendo usado o depurador.



Atenção Quando usar um dispositivo de entrada certifique-se que outro não esteja em condição que atrapalhe o uso. Por exemplo, se as chaves estiverem açãoadas o teclado não funcionará corretamente.

Teclado

Um teclado matricial está disponível. Trata-se de um teclado que opera por varredura, isto é, são aplicados diferentes valores nas linhas e pelo valor lido nas colunas é possível identificar a tecla pressionada, com uma clara economia de pinos (8 pinos para ler 16 teclas). O teclado está ligado ao microcontrolador conforme a tabela abaixo.

	RB3	Rb2	RB1	RB0
RB4	1	2	3	
RB5	4	5	6	B
RB6	7	8	9	C
RB7	*/E	0	#/F	D

Tabela 9.5: Leitura do teclado matricial pelo microcontrolador



Atenção: Para o correto funcionamento do teclado é necessário ativar os resistores de pull-up internos do microcontrolador por software.

Chaves

Estão disponíveis 8 chaves dip switch ligadas ao portal B do microcontrolador. Observe que essas chaves foram projetadas para ser baixo ativas, portanto ao serem colocadas na posição ON elas aterraram os pinos do microcontrolador. Estando aberto o nível lógico alto nas entradas do microcontrolador é garantido pelos resistores de pull-up internos.



Atenção: Para o correto funcionamento das chaves é necessário ativar os resistores de pull-up internos do microcontrolador por software.

Push-Buttons

Existem 5 chaves pulsativas (push-buttons) ligados a pinos do microcontrolador. Alguns desses pinos possuem funções especiais, como interrupções e contadores. Essas chaves são baixo ativas, portanto ao serem pressionados forcam nível lógico baixo nos pinos do microcontrolador e quando não estão pressionadas tem nível lógico alto garantido por resistores de pull-up. A tabela abaixo mostra a ligação das chaves PUSH BUTTON ao microcontrolador.

Chave	Nome	Ponto ligado ao microcontrolador
SW1	RESET	MCLR/VPP/RE3
SW4	INT0	RB0/INT
SW5	INT1	RB1
SW6	INT2	RB2
SW2	TMR1	RC0
SW3	CH0	RC1

Tabela 9.6: Leitura do teclado matricial pelo microcontrolador



Atenção: Para o correto funcionamento dos pushbuttons é necessário ativar os resistores de pull-up internos do microcontrolador por software.

9.3.7 Dispositivos de saída

Display LCD

O kit possui um display LCD alfanumérico de 2 linhas por 16 colunas, ligado ao conector P1. O ajuste de contraste desse LCD é feito pelo trimpot R37. O kit suporta também displays LCD gráficos (não inclusos) através do conector P2. O ajuste de contraste desse LCD é feito pelo trimpot R37. Para habilitar o display tem que ligar a chave 1 do dip switch CH2. Para mais informações sobre o uso do LCD consulte o capítulo dedicado a esse assunto na apostila.

Displays de 7 segmentos

O kit XM118 possui 4 displays de sete segmentos. Estes displays trabalham de forma multiplexada, isto é, seus segmentos estão todos ligados em paralelo e os comuns dos displays são acionados por um processo de varredura, dando a impressão de estarem simultaneamente ativos. Para habilitar o display consulte a tabela de configurações.

LEDs

No portal D estão ligados 8 leds convencionais baixo ativos (acendem com nível lógico baixo). Há ainda 2 LEDs bicolores ligados a outros pinos do microcontrolador, conforme o esquema elétrico. Para saber como habilitar os leds bicolores consulte a tabela de configurações.

9.3.8 Acionamentos

Há diversos circuitos de aplicação que podem ser acionados no XM118. A maioria desses circuitos estão associados aos pinos com função de PWM, mas nada impede de usar esse pinos simplesmente para ligar e desligar os circuitos em questão. Todos estes circuitos usam como drive o integrado ULN2803, que possui saída open-coletor com capacidade de até 500mA.



Atenção: Consulte as tabelas de configuração do microcontrolador e faça os ajustes necessários nas dip switch antes de usar os circuitos aqui descritos.

1 Buzzer

Um buzzer piezelétrico permite a geração de sons no kit. Para esse buzzer operar é necessário aplicar um sinal variável na frequência que se deseja ouvir. Ele não produzirá som algum se for simplesmente alimentado.

Relés

Quatro relés estão disponíveis para aplicações do usuário. É possível ter acesso aos terminais Com (comum), NA (normalmente aberto) e NF (normalmente fechado). Os relés tem capacidade de acionar cargas de até 10A, com tensão máxima de até 250V.

Lâmpada DC

Diversas aplicações interessantes podem ser feitas com a lâmpada DC presente no kit. Em especial, aplicações de PWM podem variar a intensidade de brilho. Essa lâmpada é alimentada com 12VDC.

Resistência para aquecimento

Para aplicações envolvendo malhas de controle existe um conjunto de dispositivos formados pela resistência para aquecimento, sensor de temperatura, ventoinha e sensor de rotação. Um sensor de temperatura (mais detalhes na seção que trata de entradas analógicas) é montado junto ao resistor para realização de medidas. Esse resistor de potência está ligado a 12VDC e pode também ser controlado pelo módulo PWM.

Ventoinha

A ventoinha presente na placa é um motor DC que pode ser ligado ou desligado ou ter sua velocidade controlada por PWM. Essa ventoinha está ligada ao lado da resistência, podendo ser usada para refrigerá-la.

Sensor de rotação

Associada a ventoinha foi colocado um sensor de rotação (tacógrafo). Este é composto por um emissor infravermelho constantemente ativo e um receptor infravermelho, cada um de um lado da

ventoinha. Ao girar, as pás da ventoinha interrompem periodicamente o fluxo, fazendo com que o circuito do receptor gere um sinal quadrado de frequência proporcional a velocidade de giro da ventoinha.

9.3.9 Dispositivos analógicos

Diversos circuitos analógicos estão ligados aos canais do ADC do PIC.

Para usar o ADC do PIC consulte o capítulo específico na apostila.



Atenção: Verifique as configurações necessárias para usar cada periférico analógico na tabela de configurações.

Potenciômetro

Trata-se de um potenciômetro linear ligado entre +5VDC e GND, de forma que a excursão de seu cursor gera valores entre estes limites.

Sensor de temperatura

Este sensor de temperatura LM35 fica junto à resistência de aquecimento. Ele fornece uma tensão de 10 mV por grau Celsius, sendo que essa variação é linear em toda a faixa.

Condicionador de sinais analógicos

Para a leitura de sinais analógicos externos foram inclusas no kit 4 entradas analógicas com circuitos condicionadores de sinal. São elas:

- An1 e An2: entradas amplificadas com ganhos ajustáveis de 1 a 10 vezes. O ganho desses amplificadores é ajustado pelos respectivos trim pots.
- 0 a 10 V: esta entrada divide o sinal por 2. Ela foi pensada para permitir a interação com sistemas industriais que usam a escala de 0 a 10 V para medidas.
- 4 a 20 mA: esta entrada é na verdade uma entrada de corrente. Um circuito conversor de corrente para tensão, converte o sinal de entrada de forma que 20 mA corresponderiam a 5 V no terminal do PIC.



Atenção: As saídas dos condicionadores de sinal têm proteção contra sobretensão. Apesar disso, não se deve aplicar sinais superiores a 12 V nas entradas.

Conversor D/A

Uma possibilidade de uso do PWM é para geração de níveis analógicos de tensão. O princípio de funcionamento é que ao passar o sinal do PWM por um filtro passa baixa, a saída apresente um nível de tensão proporcional ao duty-cycle do sinal de entrada. O filtro passa baixa necessário já está presente no kit. A saída desse filtro é amplificada e com ajuste de ganho variável permitindo gerar sinais com até 10 V de amplitude para o duty-cycle de 100%. Existem muitos dispositivos industriais controlados por sinais de tensão de 0 a 10 V, como por exemplo o inversor de frequência.

Existe ainda um feedback do sinal gerado para o microcontrolador, para averiguar o correto funcionamento.

9.3.10 Interfaces seriais

Interface RS-232

O microcontrolador PIC possui uma USART, isto é, uma interface de comunicação serial. Apesar desse módulo do microcontrolador realizar toda a temporização e tratamento lógico da comunicação, para que se possa conectá-lo a um computador é necessário um conversor de nível que adapte os sinais de saída do microcontrolador para o padrão RS-232 (EIA-232C). Isto é feito pelo CI MAX232 e componentes ligados a ele. Existe também um conector DB-9 (CN4) para a ligação do kit ao PC.

Consulte a tabela de configurações para verificar as configurações necessárias. Para monitorar a atividade da USART foram incluídos dois LEDs ligados aos pinos TX e RX do microcontrolador.

Interface RS-485

A USART do microcontrolador também pode ser usada para se comunicar no padrão RS-485 (EIA-485). Para isso é necessário a adequação dos sinais do microcontrolador a esse padrão, função realizada pelo CI MAX485 (CN8). Nesse caso o conector U19 permite a conexão com os sinais de comunicação (A e B) e sinais de alimentação para um circuito remoto (+12V e GND).

Consulte a tabela de configurações para verificar as configurações necessárias. Consulte a tabela de configurações para verificar as configurações necessárias. Para monitorar a atividade da USART foram incluídos dois LEDs ligados aos pinos TX e RX do microcontrolador.

USB

O PIC184550 possui uma interface USB Device. O padrão USB prevê duas classes de dispositivos: device e host. Dispositivos device devem se conectar a um dispositivo host (geralmente um computador), não sendo possível a comunicação entre dois devices.

No kit o conector CN9 é um conector B (device) que permite a conexão do kit a um host USB. O LED USB DETECT (D28) sinaliza quando o kit está conectado a um host.

Devido as características de comunicação USB, que opera em altas taxas, os sinais de D+ e D- estão ligados diretamente aos pinos do microcontrolador, havendo chaves que permitem desconectar esses pinos do restante do circuito. Consulte a tabela de configurações para verificar quais as chaves utilizadas.

9.3.11 Diversos

Essa seção descreve alguns circuitos que se comunicam com o microcontrolador através dos protocolos I2C e SPI. A presença desses componentes é, além do estudo deles em si, o estudo desses protocolos. Para maiores informações sobre esses componentes e seus protocolos comunicação consulte os manuais dos mesmos no CD.

Memória EEPROM

O componente 24C08 (U15) é uma memória EEPROM que se comunica através do protocolo I2C. Ela tem capacidade de 8kbit (1 kbyte).

Potenciômetro digital SPI

Mais uma inovação do kit. O potenciômetro digital utiliza a comunicação serial SPI (Serial Peripheral Interface), que é utilizada para a comunicação entre dois componentes em curta distância. Assim como o potenciômetro convencional, o potenciômetro digital está ligado de tal maneira que é possível ler o valor de seu cursor, tensão esta que varia de 0 a 5V.

Relógio de tempo real (RTC)

O HT1380 (U12) é um relógio de tempo real (RTC - Real Time Clock) e calendário com comunicação serial. Ele possui um cristal próprio para geração da base de tempo (X1) e é mantido por uma bateria recarregável(BAT1), mesmo se o kit for desligado da energia.

O protocolo de comunicação desse componente assemelha-se ao protocolo SPI, porém apresenta algumas divergências do padrão, consulte seu manual para mais detalhes de funcionamento.

Gerador de Frequência

Muitas aplicações do microcontrolador precisam de um sinal periódico aplicado aos seus pinos. Para atender essa necessidade o kit conta com um gerador de frequências capaz de gerar sinais quadrados de 60 Hz a 4 kHz (aproximadamente). Esse circuito tem como núcleo o CI LM555 (U26). A frequência pode ser ajustada através do potenciômetro R36.

Hardware do módulo XMM01 - Interface Industrial

O módulo XMM01 é para ser conectado aos conectores de expansão do Kit XM118. Seu objetivo é ser uma interface entre o microcontrolador e circuitos tipicamente usados na indústria. Para tanto ele é composto de um expansor de I/O, entradas fotoacopladas e saídas fotoacopladas com seus respectivos drives protegidos. A seguir é apresentada uma explicação dos circuitos do módulo. No apêndice desse manual estão os esquemas elétricos (assim como no CD que acompanha o kit) que devem ser considerados para melhor entendimento.

São duas as funções dos circuitos de entrada e saída: (1)adequar o nível dos sinais industriais para o microcontrolador e vice-versa e (2) isolar galvanicamente os circuitos. Os circuitos de entrada foram construídos de forma que são aceitas tensões de entrada entre 12Vdc e 24Vdc. Da mesma forma, os drivers de saída são capazes de acionar sinais de 12Vdc a 24Vdc, sendo protegidos com PTC (fusível rearmável) limitando a corrente de saída em 300mA. O isolamento

galvânico pode ser observado pela divisão de "terrás" na placa, que são propositalmente feitos em formatos diferentes para evidenciar esse fato. Para que o isolamento elétrico seja efetivo, não deve haver conexão entre os terras de entrada e/ou saída com o terra de controle. A fonte que fornece tensão para as saídas deve ser uma fonte externa, isolada da fonte do kit.

A figura abaixo apresenta a placa delimitando os diferentes domínios de terra.

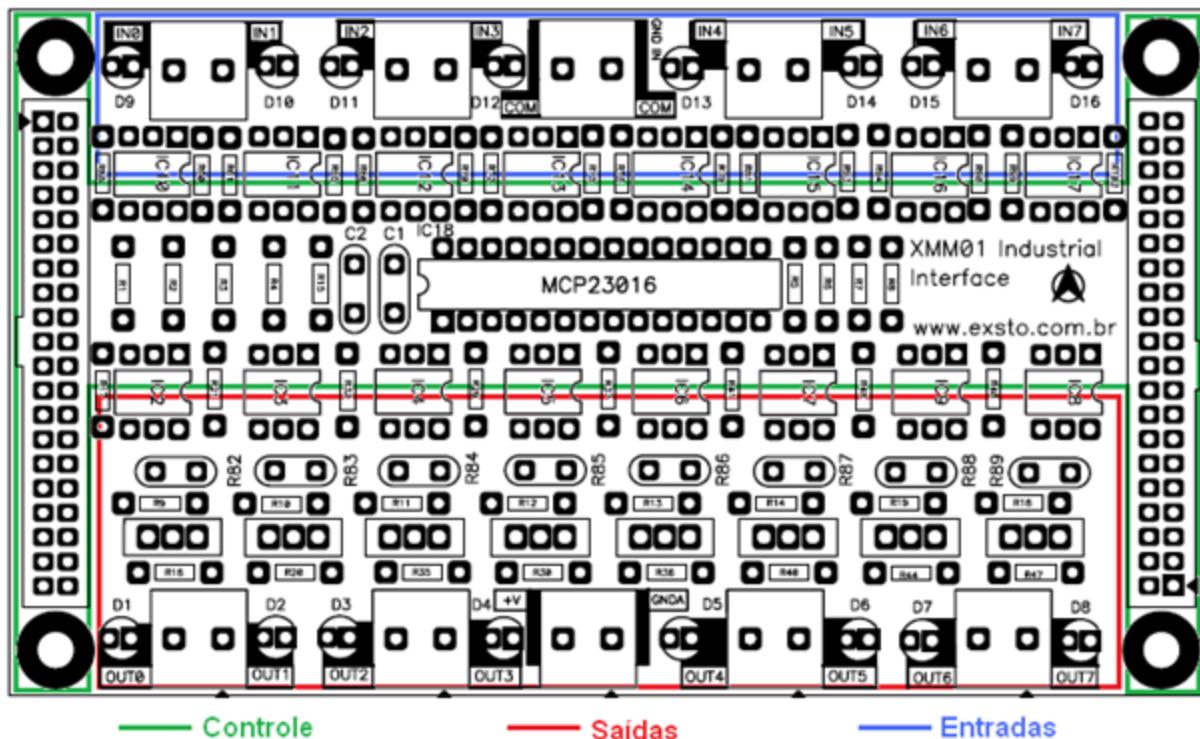


Figura 9.35: Domínios de terra

Tanto entradas como saídas possuem LEDs indicadores de seus estados lógicos.

Expansor de I/O

O expansor de I/O (I/O Expander) é um componente que, como o próprio nome diz, permite expandir a quantidade de entradas e saídas do microcontrolador. O componente usado, o MCP23016, possui uma interface serial I2C e dois portais de oito bits. Esses portais são praticamente iguais aos portais do microcontrolador. Esses portais podem ser configurados como entrada ou saída e, ainda, é possível configurar interrupções nos pinos de entradas. Para mais características técnicas e informações sobre o uso, consulte o manual desse componente.

Entradas

As entradas foto acopladas identificam como '1' sinais entre 12Vdc e 24Vdc, tornando-as aptas a serem conectadas a sensores industriais e controles que operam nessa faixa de tensão. As entradas são protegidas quanto a inversão de polaridade. Tensões acima de 30Vdc podem danificar o foto acoplador.

Todas as entradas são referenciadas a um ponto comum (COM). A figura abaixo mostra como devem ser feitas as ligações.

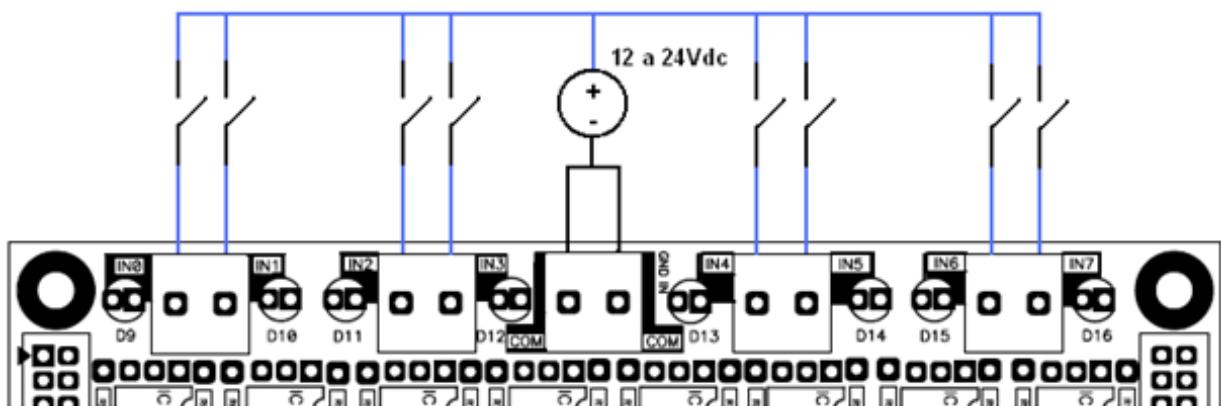


Figura 9.36: Ligação de entradas ao módulo XMM01

Saídas

As saídas também são fotoacopladas e possuem um driver que aciona tensões de 12 a 24VDC, dependendo da alimentação aplicada. Isso permite acionar diretamente algumas cargas, como solenoides e válvulas, ou acionar relés para ativar cargas de mais alta potência. Todas são protegidas contra curto por um PTC (fusível rearmável) que limita a corrente máxima em 300mA por saída.

A figura abaixo apresenta a forma como fonte e cargas devem ser ligadas às saídas.

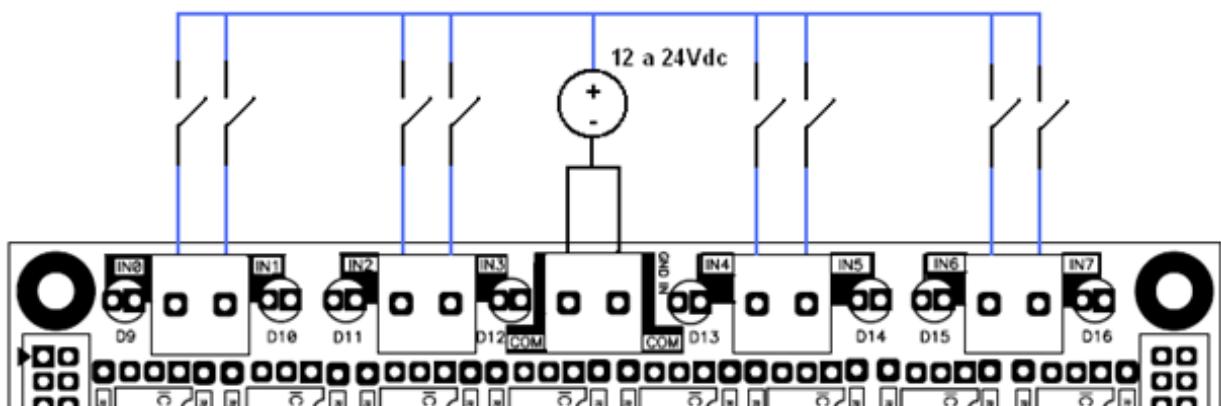


Figura 9.37: Ligação de saídas ao módulo XMM01

9.4 Resolvendo Problemas

9.4.1 Suporte Técnico

A Exsto Tecnologia oferece suporte técnico gratuito para questões de utilização de seus produtos através do e-mail suportexsto.com.br ou do telefone (35) 3471-6898.