PMR3406 – Microprocessadores em Automação e Robótica

Jun Okamoto Jr.

26 de janeiro de 2023

Sumário

Parte I

Teoria para Laboratório

Capítulo 1

Introdução

A disciplina de PMR3406 – Microprocessadores em Automação e Robótica tem como objetivo passar para o aluno conhecimentos sobre o projeto de hardware com microprocessadores e microcontroladores e desenvolvimento de software com aplicações voltadas para automação e robótica. Assim, são abordados tópicos presentes nessas áreas tais como acionamento, sensoriamento, comunicação e interface com o usuário. O curso está dividido em aulas expositivas, com 2 créditos semanais e atividades de laboratório com 2 créditos quinzenais.

Nas atividades de laboratório o aluno construirá sua própria placa processadora com um microcontrolador, com a qual executará atividades desenvolvendo o software específico para as áreas de automação e robótica. Com essa placa processadora é possível executar as seguintes tarefas:

- Comunicação através de canal serial RS232C com outro computador ou terminal
- Controle PWM de motor CC com driver externo
- Efetuar leitura de encoder para determinar posição e velocidade do eixo de motor CC
- Mostrar mensagens em display de LCD externo
- Produzir saída digital em LED
- Efetuar leitura digital de chave
- Efetuar leitura digital de sensor de linha
- Efetuar leitura analógica de sensor de proximidade
- Executar programação do microcontrolador através de bootloader

Com essa disciplina o aluno consolidará os conceitos de microprocessadores necessários para a implementação de sistemas de controle para sistemas mecatrônicos. O aluno terá a oportunidade de montar seu próprio hardware e programar com linguagem C os diversos elementos presentes num sistema microcontrolado.

1.1 Sistema de Desenvolvimento

Todo o desenvolvimento de software será feito utilizando-se o MPLAB X [3] e o compilador MPLAB XC8 [4], ambos disponíveis sem custo no site da Microchip (http://www.microchip.com). Nesse curso o mi-

crocontrolador PIC16F886 é programado por meio de um bootloader que carrega o programa desenvolvido pelos alunos na memória do microcontrolador através do canal serial.

Os computadores do laboratório contém o MPLAB X, o compilador C MPLAB XC8 e o bootloader instalados para uso nas aulas de laboratório. Os alunos devem instalar o MPLAB X e o MPLAB XC8 em seus próprios computadores para fazerem o desenvolvimento dos programas antes das aulas de maneira a atender a entrega do código no início da aula. Caso desejem os alunos poderão utilizar os próprios computadores nas atividades de laboratório.

1.2 Estrutura dos Programas

Os programas a serem desenvolvidos no laboratório seguem uma estrutura específica para o controle de máquinas com uma parte inicial de inicialização de periféricos, seguido por um loop infinito único de controle que trata "quase" todos os eventos e produz ações. O "quase" é complementado por eventos especiais gerados por interrupções. Para isso é definida uma rotina para tratamento de interrupções[1] que é chamada por eventos de hardware.

A estrutura básica do programa de controle específica para ser utilizada no laboratório com o PIC é apresentada a seguir. Inicialmente, são necessárias algumas linhas particulares da implementação com o PIC. A seguir, são definidas variáveis globais incluindo variáveis que são atualizadas na função de tratamento de interrupções e usadas no programa principal. Depois vem as funções, função de tratamento de interrupções e por último o programa principal.

O programa principal possui uma fase inicial de declarações de variáveis e inicializações. Depois vem um loop infinito de controle. O que deve ser colocado dentro desse loop e o que deve ser colocado na função de tratamento de interrupções será definido em cada atividade onde ficarão claros os motivos de tal implementação.

Um detalhe importante sobre o loop de controle, é que o mesmo nunca deve parar o processamento aguardando um evento, como por exemplo, esperar um caracter chegar pelo canal serial utilizando `getch()`[2] que não retorna da função enquanto não for recebido um caracter no buffer de recepção. Dentro do loop de controle um evento deve ser testado se ocorreu ou não, no caso de ter ocorrido deve ser tratado e de não ter ocorrido, o processamento deve prosseguir. Ainda no exemplo anterior, dentro do loop de controle, o recebimento de um caracter pelo canal serial deve ser feito com a função `chkchr()` que retorna o caracter recebido ou o número inteiro sem sinal 255 caso nenhum caracter esteja disponível no buffer de recepção. Caso o loop de controle esteja muito rápido pode-se utilizar um atraso por software disponível através das funções `delay_ms()` e `delay_us()` da biblioteca `delay.h`.

```
/*
 * File:     main.c
 * Author:   Jun Okamoto Jr.
 * Date:     dd/mm/yyyy
 * Comments: Estrutura básica de um programa
 * Revision history:
 */
#include <xc.h>     // definições dos microcontroladores
#include <stdio.h>  // I/O básico (printf, sprintf, ...)
#include <stdint.h> // tipos de variáveis padrão (uint8_t, int16_t, ...)
#include <stdlib.h> // funções adicionais (itoa, ltoa, ultoa, ...)
```

[1] Uma única rotina trata todas as interrupções no caso do PIC ou podem ser várias rotinas no caso de outros microprocessadores/microcontroladores.

[2] O comportamento padrão dessa função é não retornar enquanto não chegar um caracter pelo canal serial.

```
...

// Defines

#define ...         // todas as definições usadas no programa


// PIC16F886 Configuration Bit Settings
// CONFIG1
#pragma config FOSC = EC    // Oscillator Selection bits (EC: I/O function on RA6/OSC2...
#pragma config WDTE = OFF   // Watchdog Timer Enable bit (WDT disabled and can be enab...
#pragma config PWRTE = ON   // Power-up Timer Enable bit (PWRT enabled)
#pragma config MCLRE = ON   // RE3/MCLR pin function select bit (RE3/MCLR pin function...
#pragma config CP = OFF     // Code Protection bit (Program memory code protection is ...
#pragma config CPD = OFF    // Data Code Protection bit (Data memory code protection i...
#pragma config BOREN = ON   // Brown Out Reset Selection bits (BOR enabled)
#pragma config IESO = OFF   // Internal External Switchover bit (Internal/External Swi...
#pragma config FCMEN = OFF  // Fail-Safe Clock Monitor Enabled bit (Fail-Safe Clock Mo...
#pragma config LVP = OFF    // Low Voltage Programming Enable bit (RB3 pin has digital...
// CONFIG2
#pragma config BOR4V = BOR40V   // Brown-out Reset Selection bit (Brown-out Reset set ...
#pragma config WRT = OFF        // Flash Program Memory Self Write Enable bits (Write ...


// Project includes

#include "always.h"
#include "delay.h"
#include "lcd8x2.h"
#include "serial.h"
#include "adc.h"

// Variáveis Globais

// Funções

// Função de tratamento de interrupções
void __interrupt() isr(void) {

}

// Programa Principal
void main(void) {
  // -> Declaração de variáveis locais
  // -> Inicialização de variáveis
  // -> Configurações do processador
  // -> inicialização dos dispositivos

  while(1) { // Loop Principal
    ...
    // Nunca deve se parar esse loop esperando um evento (como a chegada de um caracter
    // pelo canal serial, por exemplo). Os dispositivos devem ser lidos e no caso de
    // não conterem informações novas o loop deve continuar. No caso de
    // informações novas é feito o processamento
    .
    .
```

```
    ...
  } // fim do Loop Principal
} // fim do Programa Principal
```

Capítulo 2

Microcontrolador PIC

A família 16 dos microcontroladores PIC da Microchip é composta por uma série de microcontroladores de 8-bits de dados com instruções de 14-bits e recursos básicos de I/O, timers, conversores A/D e canais de comunicação serial.

O PIC16F886 foi adotado nesse curso pois é um microcontrolador simples o suficiente para que o aluno possa entender o seu funcionamento durante o período do curso e sofisticado o suficiente para ter características que permitam sua utilização em automação e robótica. Trata-se de uma versão recente do antigo PIC16F76, cujo custo atual é menor do que o de versões anteriores e encontra-se em produção. Nessa versão atual muitas funções avançadas foram adicionadas em relação às anteriores, mas nesse curso são explorados recursos somente presentes em todas as versões da família, não sendo utilizadas funções avançadas. O que se pretende com isso é a exposição de conceitos que estão presentes nessa família de microcontroladores e que são semelhantes em diversas outras famílias de microcontroladores e não executar funções com alto desempenho como as que as funções avançadas do PIC16F886 permitem. Assim, apesar desse documento conter explicações específicas para o PIC16F886 os conceitos apresentados são encontrados de forma semelhante em muitos microcontroladores além dos da Microchip.

O microcontrolador PIC16F886 é disponível em encapsulamento SPDIP (*Skinny Plastic Dual-In-Line package*) de 28 pinos e possui as seguintes características principais:

- Frequência de operação até 20MHz
- 8 kwords de memória flash para programas
- 368 bytes de memória RAM de dados
- 256 bytes de memória EEPROM
- 3 portas de I/O com 8-bits cada
- 3 Timers: 2 de 8-bits e 1 de 16-bits, até 4 PWM de 10-bits
- 2 canais de comunicação serial:1 síncrono e 1 assíncrono
- 11 canais de conversão A/D de 10-bits
- 13 interrupções
- Programação no circuito (ICSP – *In-Circuit Serial Programming*) através de 2 pinos

Assim como a maioria dos microcontroladores as configurações do PIC16F886 são feitas por meio de SFRs (*Special Function Registers*). Esses registradores estão presentes em endereços específicos no microcontrolador e cada um, ou conjunto de alguns, configuram o funcionamento dos diversos módulos do microcontrolador.

As informações apresentadas a seguir foram retiradas do *data sheet* PIC16F882/883/884/886/887 [2] da Microchip e foram resumidas especificamente para as atividades de laboratório a serem desenvolvidas nesse curso. Assim, para um conhecimento mais aprofundado e utilização geral do hardware e software da família de microcontroladores PIC16F88X deve ser consultado o seu *data sheet*. Foi incluída nessa seção também uma descrição das funções de bibliotecas em XC8 que estão disponíveis para utilização no curso. Algumas dessas bibliotecas foram retiradas da Internet e adaptadas para as características do hardware utilizado no curso e acrescidas com algumas funcionalidades adicionais, outras foram escritas especificamente para o curso. As funções presentes nas bibliotecas implementam em Linguagem C as diversas configurações necessárias para cada módulo do PIC16F886 e funções de acesso à alguns dos módulos.

2.1 Organização da memória

A família de microcontroladores PIC16F88x segue a arquitetura Harvard na qual os espaços das memórias de programa e de dados são separados. A memória de programa do PIC16F886 é composta por 8192 palavras de 14-bits e é do tipo Flash que pode ser apagada e gravada no circuito. O vetor de Reset é implementado no endereço 0000h[1] enquanto que o vetor de interrupção é implementado no endereço 0004h. A memória de dados é particionada em 4 bancos de 128 bytes nos quais os endereços baixos de cada banco são reservados para os *Special Function Registers* e nos endereços acima dos *Special Function Registers* estão localizados os registradores de uso geral (para os dados dos programas do usuário) que no PIC16F886 totalizam 368 bytes. O PIC16F886 possui também um espaço de memória EEPROM (não volátil) de 256 bytes que pode ser gravado e lido por instruções específicas pelo programa do usuário. Normalmente essa memória é utilizada para armazenar parâmetros de configuração que se deseja preservar de maneira mais permanente.

2.2 Programação serial por bootloader

O PIC16F88x possui a capacidade de gravar a própria memória de programa. Esse recurso é explorado nesse laboratório onde um pequeno programa é colocado no final da memória de programa do PIC16F886. Esse programa tem a capacidade de receber o programa do usuário pelo canal serial e colocá-lo na parte inicial da memória de programa do PIC, depois passando a executar o programa do usuário. Esse método utiliza cerca de 100 bytes do final da memória de programa do PIC que não pode ser sobreescrito pelo programa do usuário. Assim recomenda-se que o programa do usuário não ocupe mais do que 95% da memória de programa total do PIC por segurança. A utilização deste método dispensa o uso de gravadores externos como o PICKit 3.

[1] h ou H no final indica que o número é hexadecimal.

2.3 Bits de configuração

Os bits de configuração podem ser programados (com valor "0") ou deixados sem programar (com valor "1") que é o valor default de fábrica. O valor inicial da palavra de configuração é FFFFh. Esses bits estão organizados em duas palavras de 14-bits e estão mapeados nos endereços 2007h (CONFIG1) e 2008h (CONFIG2). Esses endereços estão além do espaço de memória de programa do usuário e pode ser acessado somente durante a programação do microcontrolador.

No compilador XC8 esses bits de configuração podem ser programados através da diretiva e função macro `#pragma config`. Os bits devem ser especificados pelo seu nome constante na documentação e atribuído seu estado (ON ou OFF) com um sinal de igual (=) ou o seu valor especificado na documentação. Os diversos bits podem ser separados por vírgula (,) ou colocados em diretivas independentes.

Os bits de configuração para o PIC16F886 são apresentados abaixo com os respectivos valores que podem ser programados em cada um deles, os valores em negrito são os que devem ser alterados da configuração default para uso nesse curso, os demais podem ser deixados no valor default:

- FOSC – configuração do oscilador (EXTRC_CLKOUT, EXTRC_NOCLKOUT, INTRC_CLKOUT, INTRC_NOCLKOUT, **EC**, HS, XT, LP)
- WDTE – habilita Watchdog Timer(ON ou **OFF**)
- PWRTE – habilita Power-up Timer (**ON** ou OFF)
- MCLRE – seleciona função do pino RE3/MCLR (**ON** ou OFF)
- CP – habilita proteção de código da memória de programa (ON ou **OFF**)
- CPD – habilita proteção dos dados (ON ou **OFF**)
- BOREN – habilita Brown Out Reset: BOREN (**ON** ou OFF)
- IESO – habilita chaveamento entre oscilador interno e externo durante inicialização (ON ou **OFF**)
- FCMEN – habilita monitor de clock seguro contra falhas (ON ou **OFF**)
- LVP – habilita programação ICSP com baixa tensão (ON ou **OFF**)
- BOR4V – seleciona nível de tensão de Brown Out (**BOR40V** ou BOR21V)
- WRT – habilita proteção de escrita na memória flash de programa (256, 1FOURTH, HALF, **OFF**)

Para utilizar a diretiva com a função macro `#pragma config` e as definições de nomes dos bits deve ser incluído o header `xc.h` que carregará as definições para o microcontrolador correto de acordo com o definido no projeto do MPLAB X. Os detalhes sobre essas definições e configurações estão *data sheet* da família PIC16F88x.

O MPLAB X possui uma ferramenta que auxilia na geração de código fonte em Linguagem C para a programação dos bits de configuração. Essa ferramenta pode ser acessada através do menu Windows > PIC Memory Views > Configuration Bits. A Figura 1 mostra a tela e as configurações utilizadas nesse curso. Os valores configurados aparecem em vermelho.

Depois de fazer as alterações necessárias nos bits, clica-se no botão "Generate Source Code to Output" e o código será gerado numa nova janela de saída. A partir daí, pode-se selecionar o texto e copiar & colar na janela do programa `main.c`. A listagem com a estrutura geral do programa apresentada no item 1.2 contém o resultado do código fonte da configuração de bits já colado no programa.

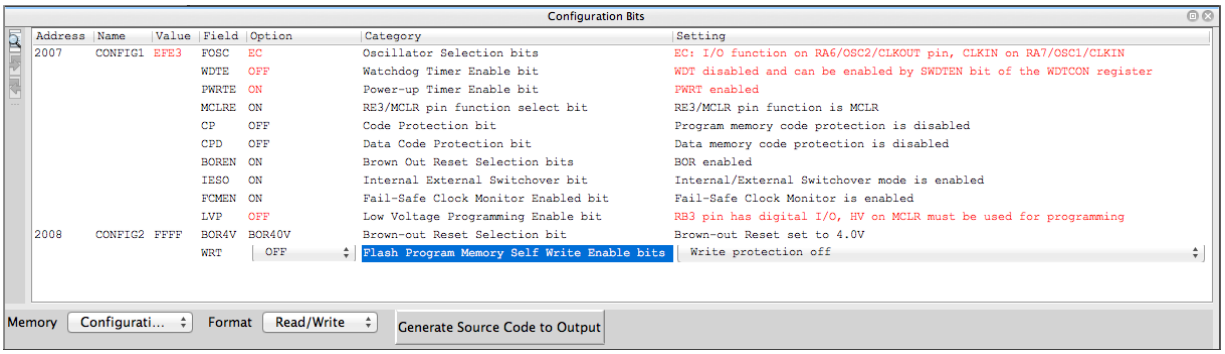


Figura 1: Tela para bits de configuração do MPLAB X

2.4 Portas de entrada e saída

As portas de entrada e saída utilizam pinos do microcontrolador que são multi-funcionais. Qualquer função programada num desses pinos desabilita a função de entrada e saída. Assim, se nenhuma função for programada num desses pinos sua função padrão será de entrada ou saída. Se um determinado pino será usado para entrada ou para saída, isso deve ser configurado num SFR: TRISA, TRISB e TRISC para as portas A, B e C, respectivamente, para o PIC16F886. As Portas A, B e C possuem 8-bits cada. A programação de um bit igual a 0 num SFR TRISx configura o bit da porta para saída, enquanto que a programação de um bit igual a 1 configura o bit da porta para entrada. O default para todos os bits dos SFRs TRISx é "1", ou seja todos os bits de todas as portas estão inicialmente configurados para entrada. Essa é uma condição segura para se evitar curto-circuitos com saídas que esteja conectadas nesses pinos. Na fase de inicialização do programa principal, as saídas devem ser configuradas.

Por exemplo, as linhas de programa abaixo configuram os bits 2, 3 e 4 da Porta C para saída:

```
TRISC = 0xf3; // bits 2 e 3 da Porta C são saída e os demais entrada
              // (0xf3 = 0b11110011), todos os bits são alterados
TRISC4 = 0;   // bit 4 da Porta C é saída e os outros bits não são alterados
```

Outro exemplo é o caso em que se deseja alterar a configuração de alguns bits e manter a configuração de outros inalterada. Para isso pode-se ler o valor do registrador de configuração e colocar os bits em zero com uma operação AND bit a bit:

```
TRISC &= 0b11110001; // faz os bits 2, 3 e 4 da Porta C como saída
                     //  e não altera a  configuração dos demais
```

Para se alterar o valor de um bit de uma porta, ou seja, fazer com que o valor de um pino mude pode-se atribuir o valor ao pino diretamente como se fosse a atribuição de valor à uma variável:

```
RC2 = 1; // bit 2 da Porta C vai para nivel lógico HIGH
```

Da mesma maneira pode-se ler o valor de um bit de uma porta que corresponde ao valor lógico de um pino:

```
bit var;    // variável do tipo bit
...
var = RC5;  // o valor do bit 5, Porta C é atribuido  à variável 'var'
```

Os pinos das Portas A e B são compartilhadas com as entradas de sinal analógico para os canais de conversão A/D e no caso de uso como pinos de I/O além da configuração dos SFRs TRISA e TRISB devem ser configurados os SFRs ANSEL e ANSELH para as Portas A e B, respectivamente. Se o bit da porta

A for usado como I/O digital deve-se escrever um "0" no bit correspondente do SFR ANSEL, lembrando que no PIC16F886 só existem os canais de conversão AN0 até AN4 nessa porta, ou seja, qualquer escrita nos bits 5, 6 e 7 desse registrador é ignorada. Na Porta B, os pinos correspondentes aos bits 0 a 5 são compartilhados com os canais de conversão A/D de 8 a 13 (6 canais de conversão A/D, AN8 a AN13) e deve-se escrever um "0" no bit correspondente do SFR ANSELH se esses bits da porta B forem usados como I/O digital. Da mesma maneira que no caso de ANSEL, a escrita de qualquer valor nos bits 6 e 7 de ANSELH é ignorada.

2.5 Interrupções

A família de microcontroladores PIC16F88x possui múltiplas fontes de interrupção agrupadas num único pedido geral de interrupção para o microcontrolador. A Figura 2 mostra a lógica de pedidos de interrupção.

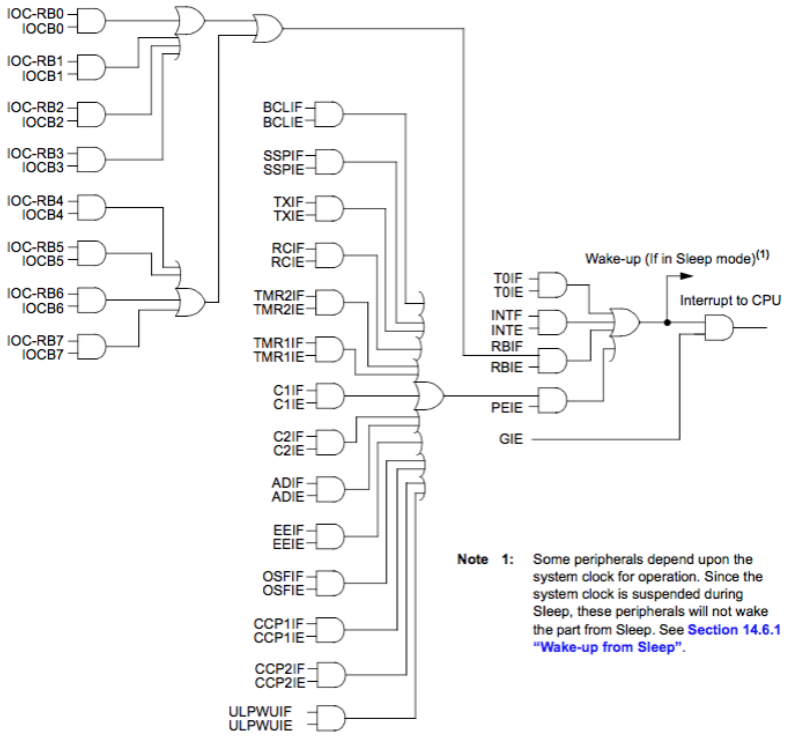


Figura 2: Lógica de pedidos de interrupção do PIC16F88x (extraído de [2])

Cada interrupção pode ser mascarada individualmente, através de um bit específico. Além disso, todas podem ser habilitadas e desabilitadas através do bit GIE (INTCON<7>). Se GIE = 1 todas a interrupções que não estiverem mascaradas poderão gerar um pedido de interrupção para o microcontrolador. Caso contrário, se GIE = 0, todas as interrupções estarão desabilitadas e não serão tratadas pelo microcontrolador, independentemente da máscara.

Na ocorrência de um pedido de interrupção atendido o GIE é automaticamente limpo (GIE = 0), desabilitando assim qualquer pedido de interrupção subsequente. No retorno da função tratamento de interrupções, o GIE é automaticamente re-habilitado (GIE = 1) permitindo que novas interrupções não mascaradas sejam tratadas.

É importante observar que cada interrupção que ocorre levanta[2] um *flag* específico e que esse *flag* é levantado independente das interrupções estarem habilitadas ou não. Por exemplo, no caso da interrupção do Timer 0, seu estouro causará o *flag* T0IF a se tornar "1", mesmo que GIE = 0. Por outro lado, mesmo que GIE = 1, no caso da interrupção do Timer 0, essa só será atendida se TMR0IE = 1, isto é, se a interrupção estiver desmascarada.

Assim, dentro da função de tratamento de interrupções deverá se testar se cada interrupção a ser tratada possui seu *flag* de pedido igual a "1" e se está desmascarada. Por exemplo, no caso de interrupção do Timer 0, deve-se testar se TMR0IF = 1 e se TMR0IE = 1 ao mesmo tempo. Somente nessa condição é que essa interrupção deverá ser tratada. O mesmo dever ser feito para as outras interrupções. O trecho de programa abaixo ilustra esse exemplo.

```
void __interrupt() isr(void) { // única função geral de
                               // tratamento de interrupções,
                               // o nome pode ser qualquer

  if (TMR0IE && TMR0IF) { // se a interupção do Timer 0 ocorreu
    // tratamento da interrupção do Timer 0
  }

  if (TXIE && TXIF) { // se a interrupção de transmissão
                      // serial ocorreu
    // tratamento da interrupção de transmissão serial
  }
  ...
}
```

No compilador XC8 a função de tratamento de interrupções deve obrigatoriamente seguir o seguinte:

- Retornar `void`
- Ter o qualificador de função `__interrupt()`
- Não ter parâmetros
- Só pode existir uma função com o qualificador `__interrupt()`

Isso pode ser visto no exemplo acima.

As variáveis globais utilizadas tanto na função de tratamento de interrupções como no `main` devem ser declaradas `volatile`. A declaração `volatile` significa que não há garantias de que a variável retenha seu valor entre acessos sucessivos, prevenido que o otimizador elimine referências aparentemente redundantes para esses objetos declarados que poderia alterar o comportamento do programa. Isso resolve casos em que os valores dessas variáveis utilizadas na função de tratamento de interrupções sejam lidos incorretamente pelo `main`.

As variáveis locais da função de tratamento de interrupções cujo valor se deseja preservar entre chamadas da função devem ser declaradas como `static`. Isso mantém as variáveis permanentemente na memória de modo que elas não perdem o valor entre acessos e mantém o escopo local à função.

2.5.1 Interrupção na mudança da Porta B

Todos os pinos da Porta B podem ser configurados para gerar pedidos de interrupção ao mudarem de nível (IOC – *Interrupt-On-Change*). São gerados pedidos tanto para mudanças de LOW para HIGH como de

[2] não encontrei uma tradução direta para o português, em inglês é o verbo "to set": *to set a flag*

HIGH para LOW. O software de tratamento de interrupções deve distinguir qual mudança de nível deve ser tratada. Inicialmente a função de interrupção na mudança vem desabilitada após o RESET e deve ser habilitada individualmente para cada pino. Para que uma interrupção num pino possa ser atendida o bit RBIE deve estar habilitado. Ao ocorrer uma mudança de nível o *flag* de interrupção RBIF vai para "1" gerando uma interrupção. A função de tratamento de interrupções deve ler a Porta B para determinar quais bits foram alterados. A interrupção é limpa na leitura da Porta B e ao limpar o *flag* RBIF que deve ser executada nesta ordem.

Para que um bit da Porta B possa ser usado nesta função devem ser configurados os seguintes registradores:

- ANSELH – pino digital para os bits de 0 a 5, correspondendo aos canais analógicos de 8 a 13[3]
- TRISB – entrada digital para o bit desejado
- WPUB – pull-up caso necessário (o controle de pull-up global $\overline{\text{RBPU}}$ do registrador OPTION deve ser habilitado para configuração de cada bit individualmente pelo registrador WPUB)
- IOCB – habilita cada bit individualmente para atender a interrupção na mudança

2.6 Timer 0

O módulo do Timer 0 possui as seguintes características:

- contador/timer de 8-bits (sempre incrementando)
- pode ser escrito e lido
- prescaler de 8-bits programável por software
- seleção de clock interno ou externo
- seleção de borda para o clock externo
- interrupção no transbordo de FFh para 00h

O Timer 0 possui dois modos de operação: contador e timer. No modo contador, o Timer 0 conta pulsos que entram no pino RA4/T0CKI. Já no modo timer, é utilizado o clock interno do microcontrolador (CLKOUT) que corresponde a frequência do oscilador dividido por 4 ($F_{OSC}/4$). O modo timer será utilizado nesse curso e é descrito com mais detalhes a seguir.

O modo timer é selecionado fazendo-se o bit T0CS igual a 0 (OPTION_REG<5>). Nesse modo o CLKOUT passa opcionalmente por um prescaler de 8-bits que pode dividir a frequência desse clock por valores entre 2 e 256 em incrementos de potências de 2 (2, 4, 8, 16, 32, 64, 128, 256). O valor de divisão da frequência é selecionado através de 3 bits, PS2, PS1 e PS0 (OPTION_REG<2:0>).

Esse prescaler é compartilhado com o Watchdog Timer[4] e deve-se selecionar se o prescaler será usado com o Timer 0 ou com o Watchdog Timer. Para tanto programa-se 0 no bit PSA (OPTION_REG<3>), o que configura o prescaler para uso com o Timer 0, que é a configuração usada nas atividades de laboratório. Os demais bits do OPTION_REG não afetam o funcionamento do Timer 0 nesse curso e não precisam

[3] A correspondência entre os pinos da Porta e os canais analógicos é a seguinte: RB0 = AN12, RB1 = AN10, RB2 = AN8, RB3 = AN9, RB4 = AN11, RB5 = AN13.

[4] não utilizado nesse curso

ser definidos. Todas as configurações que podem ser definidas pelo OPTION_REG podem ser vistas na página 79 do *data sheet* do PIC16F88x.

O Timer 0 é incrementado a cada pulso de clock, seja interno ou recebido no pino T0CKI. Sempre que ocorre a mudança de valor da contagem de FFh para 00H (transbordo ou estouro) é gerada uma interrupção sinalizada pelo bit TMR0IF (INTCOM<2>) que vai de 0 para 1. Essa interrupção pode ser mascarada se o bit TMR0IE (INTCON<5>) for limpo (TMR0IE = 0). Se for esse o caso o pedido de interrupção não será gerado, mas mesmo assim o bit TMR0IF irá para 1. No caso de ocorrer a interrupção, o bit TMR0IF deve ser limpo por software pela função de tratamento de interrupções do Timer 0 antes de se re-habilitar essa interrupção. Além disso, deve-se lembrar que para que qualquer interrupção seja atendida, o bit GIE (INTCON<7>) deve ter sido feito = 1.

Num exemplo prático, deseja-se gerar uma interrupção periódica a cada 5 ms aproximadamente usando-se o Timer 0, considerando que a frequência do oscilador do microcontrolador é 20 MHz.

Assim, a frequência de clock interno CLKOUT é 5 MHz (= 20 MHz / 4). Configurando-se o prescaler para divisão por 256, tem-se que:

- Período do clock interno = 200 ns
- Saída do prescaler = 51,20 μs (= 200 ns × 256)
- Valor a ser programado como contagem inicial no Timer 0 = 157 (= 255 − 98) pois 51,20 μs × 98 = 5,0176 ms

O trecho de programa abaixo mostra como configurar o Timer 0 e como habilitar a sua interrupção:

```
void t0_init(void) {  // o Timer 0 é utilizado para interrupção periódica a cada ~5 ms
  OPTION_REGbits.T0CS = 0; // usa clock interno FOSC/4
  OPTION_REGbits.PSA = 0;  // Prescaler é para Timer 0 e não para WDT
  OPTION_REGbits.PS = 7;   // ajusta Prescaler do Timer 0 1:256
  TMR0 = 0xff - 98;        // valor inicial do Timer 0 p/ 5,0176 ms
  TMR0IE = 1;              // habilita interrupção do Timer 0
}
...
void main (void) {
  ...
  t0_init(); // inicializa Timer 0 para interrupcão

  // Controle de Interruções
  GIE = 1;                    // habilita interrupções
  ...
}
```

A função de tratamento de interrupções deve verificar se os bits TMR0IE e TMR0IF são ambos iguais a 1 para, somente nesse caso, executar o tratamento necessário para a interrupção do Timer 0. Nessa função de tratamento do Timer 0 um novo valor de contagem inicial deve ser carregado no Timer 0 para se ajustar o tempo que deverá decorrer até a próxima interrupção.

O trecho de programa abaixo corresponde a função de tratamento de interrupções para esse exemplo:

```
void __interrupt() isr(void) { // função de tratamento de interrupções

  // Timer 0
  if (TMR0IE && TMR0IF) {      // se estiver habilitada e ocorreu a interrupção

    // executa aqui o que for necessário
```

```
    TMR0 = 0xff - 98;    // recarrega a contagem incial
    TMR0IF = 0;          // limpa o flag de interrupção
  } // fim do tratamento do Timer 0

  // tratamento de outras interruções aqui

}
```

Capítulo 3

Debounce de Chave

Todas as chaves ao mudarem de estado apresentam ruído nessa mudança o que faz o sinal na entrada da porta se comportar como se estivessem ocorrendo múltiplos acionamentos, esse fenômeno é chamado de "bounce", vide Figura 3.

Uma das maneiras de se detectar um único acionamento para cada mudança de estado da chave é detectar a primeira transição de HIGH para LOW (ou vice e versa), esperar um tempo e depois confirmar o nível final. Esse procedimento chama-se "debounce" da chave.

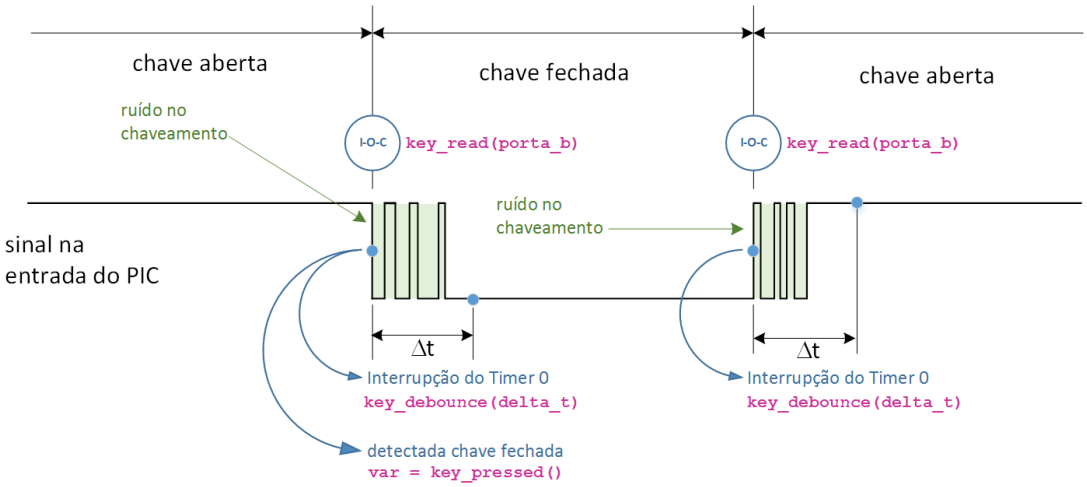


Figura 3: Procedimento de debounce de chave

Uma solução por software é fornecida no laboratório que executa o seguinte procedimento:

1. A primeira transição de nível é detectada por uma interrupção na mudança (Interrupt-On-Change) da porta B que liga dois flags: FLAG1 para sinalizar para a interrupção do Timer 0 que deve ser marcado um período de tempo para o debounce e FLAG2 para sinalizar para o programa principal que a chave foi acionada. Na interrupção na mudança a porta B deve ser lida com a função `key_read(porta_b)` que verifica se o bit da porta que causou a interrupção é o que está ligado na chave e se for o caso liga os dois flags.

2. A interrupção do Timer 0 detecta o FLAG1 e marca um período de tempo[1], entre 5 ms e 10 ms

[1]Esse período de tempo depende da chave. No laboratório, esse tempo foi medido experimentalmente e pode variar dependendo do modelo e fabricante da chave.

(configurável por um parâmetro). Esse tempo deve ser o suficiente para o "bounce" acabar. O tempo é marcado na interrupção do Timer 0 pela função `key_debounce(2)` onde o número passado como parâmetro define a quantidade de milissegundos e corresponde a $n\times$ o período da interrupção do Timer 0. Essa função limpa o FLAG1 de sinalização no final do tempo.

3. O programa principal detecta o FLAG2 e entende que a chave foi acionada prosseguindo com a execução adequada para o caso. A função `key_pressed()` no programa principal retorna TRUE e a chave tiver sido pressionada e retorna FALSE em caso contrário. Essa função também limpa o FLAG2 se a chave foi pressionada[2].

4. O mesmo acontece na outra transição quando a chave é liberada só que nesse caso somente o FLAG1 que sinaliza para o Timer 0 marcar tempo é ligado.

Antes do uso das funções da chave, a porta B deve ser inicializada e configurada para reconhecer a interrupção na mudança. Isso é feito pela função `key_init()`. Deve ser incluído o header `key.h` para que essas funções possam ser utilizadas. Verifique o funcionamento desse procedimento analisando o código fornecido para testes da placa (HW_Test).

[2]Essa função somente detecta a transição de chave aberta para fechada. A transição de chave fechada para aberta é ignorada. Caso se deseje detectar as duas transições, essa função deve ser alterada.

Capítulo 4

Conversão A/D

É através do uso de um conversor A/D que se pode fazer a leitura de sinais contínuos por sistemas digitais. Desta maneira, um valor de tensão analógico poder ser convertido para um valor numérico digital numa variável inteira sem sinal dentro de um programa num microcontrolador. Existem diversos métodos para conversão analógica / digital (A/D). Seja qual for o método, o sinal analógico é amostrado em intervalos discretos e convertido num número digital que representa a amplitude do sinal. O processo de conversão sempre demora um certo tempo para ser completado e sempre ocorrerá um erro de aproximação. Quanto maior o número de bits do conversor, menor será o erro de aproximação.

Um dos mais utilizados atualmente em microcontroladores é o método de aproximações sucessivas [5] que é utilizado pelo PIC 16F886. A Figura 4 mostra a estrutura básica de um conversor A/D por aproximações sucessivas. Neste tipo de conversor é utilizado um conversor D/A para gerar níveis de tensão que são comparados com a tensão analógica de entrada que se deseja converter. Um código binário é gerado para o conversor D/A sempre dividindo no meio a faixa de tensão que se aproxima da tensão de entrada. Esse processo de inicia pelo bit mais significativo e prossegue até o bit menos significativo, quando o processo termina. A Figura 5 mostra a operação desse tipo de conversor com um exemplo de 4 bits.

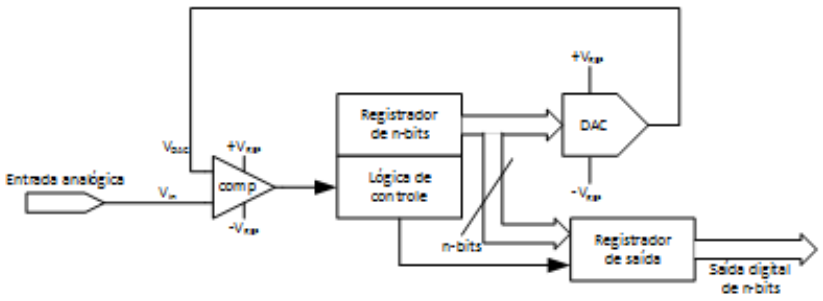


Figura 4: Estrutura básica de um conversor A/D por aproximações sucessivas (adaptado de [5])

4.1 Conversão A/D com o PIC

O módulo de conversão analógica para digital (A/D) do PIC16F886 possui 11 canais compartilhando os pinos das Portas A e B, sendo 5 canais nos pinos da Porta A e 6 canais nos da Porta B. Os canais AN0 a AN4 compartilham os pinos RA0 a RA4 enquanto que os canais AN8 a AN13 compartilham os pinos RB0

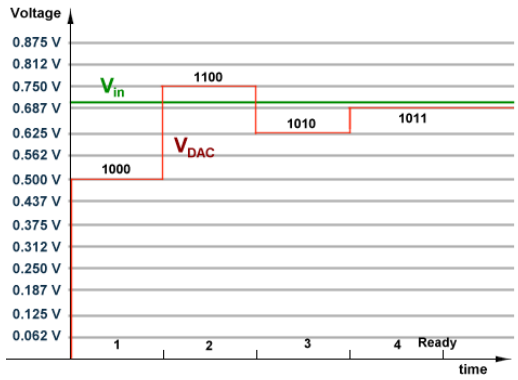


Figura 5: Operação de um conversor A/D por aproximações sucessivas com um exemplo de 4 bits (extraído de [5])

a RB5. Os Canais AN5 a AN7 não existem no PIC16F886. O conversor A/D gera um resultado binário de 10-bits pelo método de aproximações sucessivas descrito anteriormente. O resultado da conversão é armazenado num par de registradores de 8-bits cada. Os 10-bits do resultado podem ser alinhados à direita ou à esquerda nos 16-bits formados por ADRESH e ADRESL. O conversor A/D é controlado pelos registradores ADCON0 e ADCON1.

Em resumo, o módulo de conversão A/D opera com 6 registradores:

- ADCON0 – Registrador de controle e operação
- ADCON1 – Registrador de configuração
- ANSEL – Registrador de configuração dos canais AN0 a AN4 como analógicos ou digitais
- ANSELH – Registrador de configuração dos canais AN8 a AN13 como analógicos ou digitais
- ADRESH – Registrador da parte mais significativa do resultado
- ADRESL – Registrador da parte menos significativa do resultado

Além desses registradores específicos devem ser configurados os registradores TRISA, TRISB. A configuração default desses registradores é para operação como entrada analógica para o conversor A/D para todos os canais.

4.1.1 Seleção do clock de conversão A/D

O tempo de conversão A/D para cada bit é definido como T_{AD}. O processo de conversão A/D requer o tempo mínimo de 11 T_{AD} para a conversão de 10-bits. Para que as conversões possam ser realizadas corretamente a frequência do clock de conversão A/D deve ser definida de maneira a garantir no mínimo um T_{AD} de 1,6 µs.

A frequência do clock de conversão é definida através dos bits ADCS1 e ADCS0 (ADCON0<7:6>). Pode-se definir as frequências de $F_{OSC}/2$, $F_{OSC}/8$, $F_{OSC}/32$ ou FRC (frequência determinada pelo oscilador RC interno do módulo A/D). A Tabela 1 resume as configurações possíveis para a definição da frequência do clock do conversor.

Por exemplo, se o clock do oscilador for F_{OSC} = 20MHz o período será 50 ns, assim o clock de conversão com frequência $F_{OSC}/32$ resultará em T_{AD} = 1,6 µs, que é o valor mínimo possível para T_{AD} . Nesse

Tabela 1: Bits de configuração da frequência do clock de conversão A/D (adaptado da página 105 do *data sheet* do PIC16F88x [2]).

Período do clock do ADC (T_{AD})		Frequência do dispositivo (F_{OSC})			
Frequência do clock de conversão A/D	**ADCON0 ADCS<1:0>**	**20 MHz**	**8 MHz**	**4 MHz**	**1 MHz**
$F_{OSC}/2$	00	100 ns	250 ns	500 ns	2,0 µs
$F_{OSC}/8$	01	400 ns	1,0 µs	2,0 µs	8,0 µs
$F_{OSC}/32$	10	1,6 µs	4,0 µs	8,0 µs	32,0 µs
FRC (frequência determinada pelo oscilador RC interno do módulo A/D)	11	2-6 µs	2-6 µs	2-6 µs	2-6 µs

Legenda: células com fundo cinza estão fora do valor recomendado.

caso os bits ACDS1:ADCS0 devem ser programados com o valor 10. Valores de T_{AD} maiores do que 6 µs também não são considerados adequados pois deixam o processo de conversão muito lento e recomenda-se a escolha de outro fator de divisão para F_{OSC} .

4.1.2 Outras configurações

Justificação do resultado. O resultado de conversão A/D possui de 10-bits de informação que podem ser alinhados à direita ou à esquerda com restante dos bits preenchidos com zero nos registradores ADRESH e ADRESL que possuem juntos 16-bits. Essa configuração é feita no bit ADFM (ADCON1<7>). Se ADFM = 0 o resultado da conversão é justificado à esquerda e se for = 1 é justificado à direita. A Figura 6 mostra como fica a justificação do resultado de acordo com a configuração desse bit.

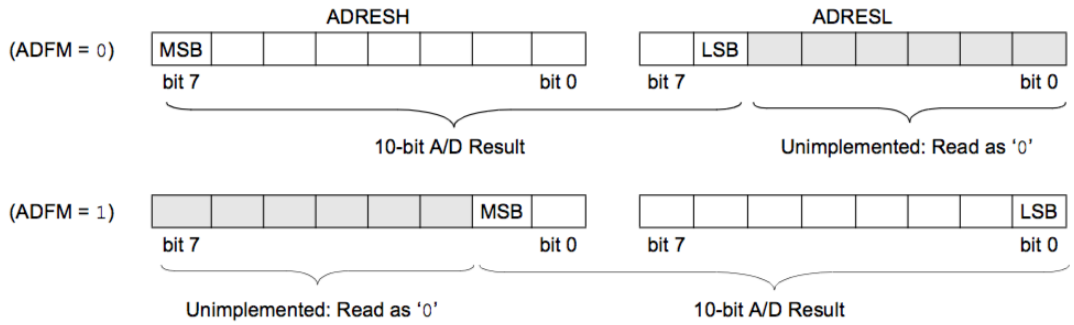


Figura 6: Justificação do resultado da conversão A/D de acordo com ADFM (extraído de [2])

Tensões de referência. As tensões de referência utilizadas para a conversão A/D pode ser configuradas independentemente através dos bits VCFG0 e VCFG1 (ADCON1<5:4>). Para se utilizar como referências as tensões de alimentação (VDD e VSS) do PIC deve-se configurar ambos os bits para "0". Para se configurar a tensão de referência inferior VREF−, escreve-se o valor "1" no bit VCFG1. Nesse caso, o pino RA2/AN2 é utilizado como entrada para esse nível de tensão de referência. Para se configurar a tensão de referência superior VREF+, escreve-se o valor "1" no bit VCFG0. Nesse caso, o pino RA3/AN3 é utilizado como entrada para esse nível de tensão de referência.

4.1.3 Procedimento para realizar uma conversão A/D

Os seguintes passos devem ser seguidos para se fazer uma conversão A/D:

1. Configurar a porta:
   - Desabilitar o driver de saída do canal através da programação de um "1" no bit correspondente do registrador TRISx
   - Configurar o pino como analógico através da programação de um "1"no bit correspondente do registrador ANSEL ou ANSELH
2. Configurar o módulo A/D:
   - Selecionar a frequência do clock de conversão A/D programando-se ADCS1 e ADCS0 no registrador ADCON0
   - Configurar a referência de tensão programando-se VCFG1 e VCFG0 no registrador ADCON1
   - Selecionar o canal de entrada A/D programando-se CHS<3:0> no registrador ADCON0
   - Selecionar o formato do resultado programando-se ADFM no registrador ADCON1
   - Ligar o módulo de conversão A/D através da programação de um "1" no bit ADON do registrador ADCON0
3. Configurar a interrupção A/D (caso desejado):
   - Limpar o bit ADIF
   - Fazer o bit ADIE = 1
   - Fazer o bit PEIE = 1
   - Fazer o bit GIE = 1
4. Esperar o tempo de aquisição necessário.
5. Iniciar o processo de conversão:
   - Fazer o bit GO/$\overline{\text{DONE}}$ = 1 no registrador ADCON0
6. Esperar o processo de conversão ser completado por um de dois métodos:
   - Esperar o bit GO/$\overline{\text{DONE}}$ ser limpo (sem interrupções habilitadas)

     OU
   - Esperar pela interrupção de A/D através do bit ADIF
7. Ler o resultado da conversão A/D no par de registradores ADRESH e ADRESL, limpar o bit ADIF caso necessário.
8. Para a próxima conversão, prosseguir com o passo 2 ou 3 conforme necessário. O tempo de conversão A/D por bit é definido como T_{AD}. Um tempo mínimo de $2T_{AD}$ é necessário antes que uma nova conversão seja iniciada.

4.1.4 Funções de biblioteca

Para configuração e utilização do módulo de conversão analógica para digital (A/D) foi criada uma biblioteca com as seguintes funções:

- `void adc_init(unsigned char ADC_Channel)` – Essa função configura os canais A/D de 0 a 3, onde o número do canal é o parâmetro de entrada `ADC_Channel`. Ela está configurada previamente para o caso do laboratório onde se utiliza somente o canal analógico 0, com tensão de referência dada por VDD e VSS. Qualquer alteração deve ser feita no arquivo `adc.c`. Essa função deve ser chamada uma única vez para cada canal na seção de inicialização dos dispositivos do programa principal.
- `unsigned int adc_read(unsigned char ADC_Channel)` – Recebe o valor do canal A/D passado como parâmetro com 10-bits alinhados à direita. Essa função aguarda o término da conversão antes de retornar. Além disso, essa função configura a frequência de conversão para $F_{OSC}/32$ para a frequência do oscilador de 20MHz e pode ser chamada dentro da função de tratamento de interrupções.

Para a utilização dessas funções deve ser incluído o header `adc.h`.

Capítulo 5

Display LCD

A maioria dos displays de LCD utiliza chips controladores que seguem o mesmo padrão de conexão com um microcontrolador e com comandos muito similares. O display utilizado no laboratório usa um chip HD44780U da Hitachi cuja documentação pode ser baixada da Internet. Neste documento é apresentado um resumo dos pontos mais importantes para se utilizar o display de LCD, caso se deseje modificar o funcionamento do display deve-se consultar a documentação completa do controlador HD44780U. A conexão física do display de LCD com o microcontrolador é vista nas aulas de teoria em detalhes.

5.1 Interface com microcontrolador

A interface do controlador do display LCD com o microcontrolador utiliza 3 sinais de controle e 8 ou 4 bits de dados descritos na Tabela 2.

Tabela 2: Sinais de interface do controlador do display LCD com microcontrolador.

Sinal	C/D	I/O	Descrição
RS	C	I	Seleciona registradores. 0: Registrador de comandos (escrita) ou *Flag* de ocupado e contador de endereços (leitura) 1: Registrador de dados (escrita e leitura)
$\mathrm{R}/\overline{\mathrm{W}}$	C	I	Seleciona leitura ou escrita. 0: escrita 1: leitura
E	C	I	Inicia escrita ou leitura de dados (age como *strobe*)
DB4 a DB7	D	I/O	4 bits mais significativos do barramento de dados tri-state bidirecional. Usado para transferência e recepção de dados entre o microcontrolador e o HD44780U. DB7 pode ser usado como *flag* de ocupado.
DB0 a DB3	D	I/O	4 bits menos significativos do barramento de dados tri-state bidirecional. Usado para transferência e recepção de dados entre o microcontrolador e o HD44780U. Esses sinais não são usados durante a operação em 4-bits.

No circuito montado nesse curso é utilizada uma interface com 4-bits e somente para escrita, isto é, são utilizados os sinais RS, E e DB4 a DB7, o sinal $\mathrm{R}/\overline{\mathrm{W}}$ é mantido em nível lógico LOW o tempo todo e os sinais DB0 a DB3 não são utilizados. Essa abordagem utiliza a menor quantidade de sinais,

economizando desta maneira pinos de I/O do microcontrolador, em contrapartida, o acesso ao display LCD é mais lento. A Figura 7 mostra um exemplo de transferência de dados entre o microcontrolador e o display LCD através de uma interface de 4-bits. Esse exemplo mostra 3 ciclos de transferência de dados:

- Escrita no registrador de instruções
- Leitura do *flag* de ocupado e contador de endereços
- Leitura do registrador de dados

No primeiro e segundo o sinal RS está em nível lógico LOW, o que significa que é escrita ou leitura de um comando. O sinal R/$\overline{\text{W}}$ define se é leitura ou escrita, ou seja, é escrita no primeiro ciclo e leitura no segundo. No terceiro o sinal RS está em nível lógico HIGH e o sinal R/$\overline{\text{W}}$ está em nível lógico HIGH, o que significa que é leitura de dados. Observe que o sinal E pulsa de LOW para HIGH cada vez que um dado está estável no barramento de dados DB4 a DB7. Este é o sinal de *strobe* que comanda a transferência.

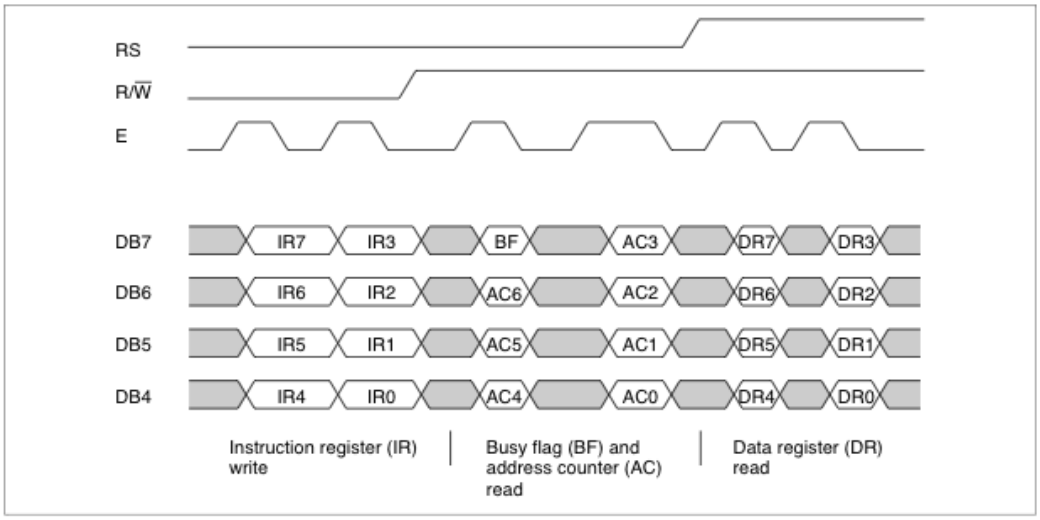


Figura 7: Exemplo de transferência de dados através de interface de 4-bits entre microcontrolador e display LCD (extraído de [1])

Como na implementação desse laboratório o sinal R/$\overline{\text{W}}$está sempre em nível lógico LOW são realizadas somente operações de escrita no display LCD e nunca operações de leitura. A limitação dessa abordagem é que não é possível ler o *flag* de ocupado e os endereços do registrador de dados (se RS = 0) nem o conteúdo do registrador de dados (se RS = 1). Como consequência, entre duas escritas consecutivas é necessário esperar um tempo fixo, cerca de 40 µs, para o controlador do display voltar para um estado desocupado. O não respeito a esse intervalo de tempo causa a perda de dados.

5.2 Instruções e dados

Somente um registrador de instruções (IR) e um registrador de dados (DR) do HD44780U podem ser controlados pelo microcontrolador. Antes de iniciar qualquer operação interna do HD44780U, a informação de controle é temporariamente armazenada nesses registradores o que permite seu interfaceamento com diversos tipos de microcontroladores e me diferentes velocidades.

5.2.1 Instruções

O registrador de instruções (IR) pode conter instruções em 4 categorias:

Especificação de funções do HD44780U, tais como formato do display, comprimento de dados, etc.

- Definição de endereços da RAM interna
- Realização transferência de dados com a RAM interna
- Realização de funções diversas

A Tabela 3 apresenta todas as instruções disponíveis com o seu tempo de execução.

Tabela 3: Instruções do HD44780U (extraído de [1])

Instruction	Code RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{cp} or f_{OSC} is 270 kHz)
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 μs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 μs
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 μs
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 μs
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 μs
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 μs
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 μs

Instruction	Code RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{cp} or f_{OSC} is 270 kHz)
Write data to CG or DDRAM	1	0	Write data								Writes data into DDRAM or CGRAM.	37 μs t_{ADD} = 4 μs*
Read data from CG or DDRAM	1	1	Read data								Reads data from DDRAM or CGRAM.	37 μs t_{ADD} = 4 μs*
	I/D = 1: Increment I/D = 0: Decrement S = 1: Accompanies display shift S/C = 1: Display shift S/C = 0: Cursor move R/L = 1: Shift to the right R/L = 0: Shift to the left DL = 1: 8 bits, DL = 0: 4 bits N = 1: 2 lines, N = 0: 1 line F = 1: 5 × 10 dots, F = 0: 5 × 8 dots BF = 1: Internally operating BF = 0: Instructions acceptable										DDRAM: Display data RAM CGRAM: Character generator RAM ACG: CGRAM address ADD: DDRAM address (corresponds to cursor address) AC: Address counter used for both DD and CGRAM addresses	Execution time changes when frequency changes Example: When f_{cp} or f_{OSC} is 250 kHz, $37\ \mu s \times \frac{270}{250} = 40\ \mu s$

Note: — indicates no effect.

* After execution of the CGRAM/DDRAM data write or read instruction, the RAM address counter is incremented or decremented by 1. The RAM address counter is updated after the busy flag turns off. In Figure 10, t_{ADD} is the time elapsed after the busy flag turns off until the address counter is updated.

Procedimento de inicialização do display LCD

Ao ser aplicada energia no HD44780U, um circuito de RESET interno inicializa o chip que passa por um procedimento de inicialização automático. Durante a inicialização, o *flag* de ocupado está preso no nível lógico HIGH por cerca de 10 ms após a tensão Vcc atingir 4,5V e passa para o nível lógico LOW após a inicialização ser completada.

Apesar a inicialização automática do HD44780U definir uma série de configurações sem a intervenção do usuário, essas configurações não são adequadas para as conexões de hardware feitas na implementação utilizada no laboratório. Assim, após a inicialização automática deve-se realizar um procedimento de inicialização apropriado que deve seguir os seguintes passos:

1. Limpar display
2. Definir as seguintes funções:
3. DL = 0; interface de 4-bits
4. N = 1; display de 2 linhas
5. F = 0; tamanho do caractere em matriz de 5 x 8 pontos
6. Controle de liga/ desliga do display:
7. D = 0; desliga display
8. C = 0; desliga cursor
9. B = 0; sem piscar (blink)
10. Definir modo de entrada:
11. I/D = 1; incrementa de 1
12. S = 0; sem deslocamento

O significado das siglas utilizadas no procedimento acima estão no final da Tabela 3.

5.2.2 Dados

A RAM de dados do display (DDRAM) armazena os dados que são visíveis no display num código de 8-bits por caractere. A quantidade total de caracteres que podem ser armazenados nessa memória é de 80 caracteres. Existe uma relação entre o endereço da DDRAM e a apresentação dos caracteres no display. Num display de 2 linhas a relação dos endereços é apresentada na Figura 8. Observa-se que os 40 primeiros endereços a partir do endereço 00h correspondem à primeira linha do display e os 40 endereços a partir do endereço 40h correspondem à segunda linha. No display LCD utilizado no laboratório uma janela de 8 caracteres por 2 linhas sobre essa organização de endereços da DDRAM sempre estará visível. Ou seja, se a janela estiver sobre os 8 primeiros endereços serão mostrados no display os caracteres armazenados nos endereços 00h a 07h para a primeira linha e nos endereços 40h a 47h para a segunda linha. Essa janela pode ser deslocada através das instruções adequadas. A Figura 9 mostra a posição da janela de 8 caracteres por 2 linhas sobre o início da DDRAM, depois no caso de deslocamento para a esquerda e por último no caso de deslocamento para a direita.

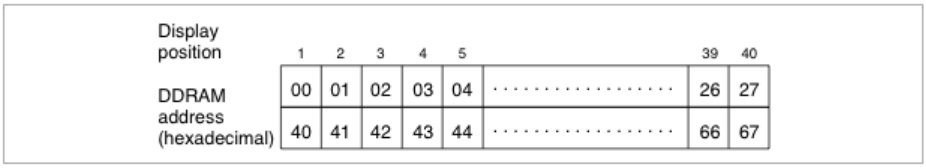


Figura 8: Endereços da DDRAM para apresentação de caracteres no display (extraído de [1])

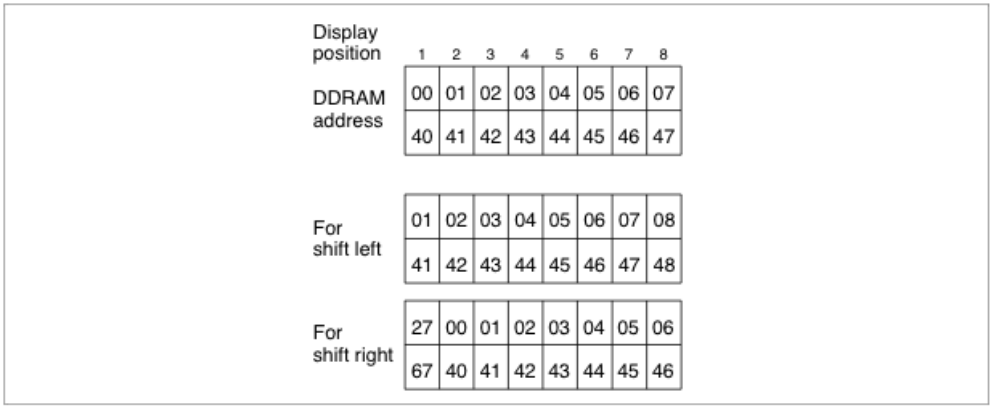


Figura 9: Exemplo de deslocamento da janela de um display de 8 caracteres por 2 linhas (extraído de [1])

Para se escrever caracteres que podem se mostrados no display deve-se primeiramente definir no registrador IR o endereço da DDRAM a partir do qual se deseja armazenar o código do caractere e depois escrever o código no registrador DR. Após essas operações um procedimento interno do HD44780U passa o código para a posição da DDRAM definida. Existe um mecanismo de incremento e decremento automático do endereço da DDRAM de modo que não é necessário definir um novo endereço cada vez que desejar mostrar um caractere no display, é suficiente escrever o primeiro endereço. É importante observar que os endereços da DDRAM possuem somente 7-bits podendo assumir valores entre 0 e 127 (ou 7Fh).

5.3 Caracteres

Os caracteres que podem ser mostrados no display LCD são representados com um código de 8-bits e são armazenados internamente numa memória ROM no HD44780U. Os caracteres com o bit mais significativo igual a 0 e entre os códigos 20h e 7Fh são representados com o código ASCII, o que mantém compatibilidade do display com a maioria dos sistemas. Por outro lado, os códigos que possuem bit mais significativo igual a 1 são particulares de cada fabricante e mostrarão caracteres por ele definidos. Os primeiros 16 códigos podem ser usados para mostrar caracteres definidos pelo usuário e armazenados numa memória apropriada (CGRAM) dentro do HD44780U. A Tabela 4 mostra os caracteres que podem ser mostrados no display por código. As linhas contém os 4-bits menos significativos de cada código enquanto que as colunas contém os 4-bits mais significativos.

Tabela 4: Caracteres que podem ser mostrados no display LCD (extraído de [1])

5.4 Biblioteca

Está disponível uma biblioteca para o XC 8 para acesso ao display LCD no laboratório. Essa biblioteca foi configurada para a implementação de hardware utilizada nesse laboratório. Caso seja alterado algum bit utilizado da Porta B ou C, a biblioteca deverá ser modificada. Para utilizá-la deve incluir o header `lcd8x2.h` no programa. As funções dessa biblioteca são as seguintes:

- `void lcd_init(void)` – Essa função inicializa o bit RB0 como entrada digital para receber o sinal DATA READY do robô, utilizado na comunicação com o display. A interface SPI do PIC deve ser inicializada antes.

- `void lcd_clear(void)` – Essa função limpa o display e coloca o cursor na primeira posição da primeira linha. ATENÇÃO: essa função demora cerca de 2 ms para ser executada e não seve ser utilizada dentro de loops pois causa atrasos na execução do programa e atrapalha a execução de interrupções. Deve-se utilizar essa função somente após a inicialização do display LCD ou em trechos de programa onde não seja repetida periodicamente. Para se retornar o cursor para a primeira posição da primeira linha deve-se utilizar a função `lcd_goto(0)` (vide descrição abaixo) e para se apagar caracteres pode-se escrever espaços (ASCII = 20h) no display.

- `void lcd_puts(const char *s)` – Essa função escreve um *string* na memória DDRAM.
- `void lcd_goto(unsigned char pos)` – Essa função muda o cursor para o endereço especificado no parâmetro. O valor do parâmetro pode estar entre 0 e 39 para a primeira linha e entre 64 e 103 para a segunda linha.
- `void lcd_show_cursor(int on)` – Liga ou desliga o cursor de acordo com o parâmetro de entrada. Se `on` = 1 liga o cursor piscando e se `on` = 0 desliga o cursor.
- `void lcd_putchar(char c)` – Escreve um caracter na posição atual do cursor no LCD. Não é feita nenhuma consistência se o caracter é visível ou não, caso não seja um caracter visível, simplesmente ele não será mostrado no display.

Capítulo 6

Comunicação serial

A comunicação serial entre dois dispositivos pode ser feita de modo síncrono ou assíncrono. No modo síncrono a transmissão e recepção dados é controlada por um sinal de clock, gerado pelo dispositivo mestre. Geralmente, esse modo é utilizado entre dispositivos internos de um computador onde o mestre é o microcontrolador e os dispositivos escravos são periféricos que podem ser conversores A/D e D/A, memórias, sensores, dentre outros. Já no modo assíncrono, a transmissão e recepção de dados é feita através de uma única linha de dados, sem a transmissão do sinal de clock entre os dispositivos. Nesse caso, cada dispositivo gera seu próprio clock internamente. Esse modo é utilizado para comunicação entre computadores independentes ou entre um computador e um periférico do tipo terminal ou console de dados.

Se um módulo de comunicação serial tem a capacidade de receber e transmitir ao mesmo tempo, esse módulo é denominado *full-duplex*. Por outro lado, se só transmite ou recebe dados exclusivamente, é denominado *half-duplex*. Para o modo *full-duplex* são necessárias duas linhas de sinal, uma de entrada e outra de saída para que possam operar simultaneamente. No modo *half-duplex*, a transmissão e recepção pode se dar numa única linha bidirecional. Pode-se ter sistemas de comunicação serial assíncrono e síncrono tanto *full-duplex* como *half-duplex* conceitualmente, mas as implementações são mais restritivas. Por exemplo, no PIC 16F886, a EUSART se configurada em modo assíncrono pode operar em *full-duplex*, mas se configurada em modo síncrono só pode operar em *half-duplex*.

6.1 Comunicação serial assíncrona

No modo assíncrono, o início de transmissão de dados é marcado por um start-bit. Depois é enviada a sequência dos bits de dados, a partir do bit menos significativo. A transmissão é terminada por um stop-bit. Quando nenhum dado está sendo transmitido, a linha de comunicação permanece no mesmo valor do stop-bit cujo nível é o oposto do start-bit. A duração de cada bit e a quantidade de bits de dados devem ser configuradas com os mesmos valores tanto no dispositivo que transmite como no que recebe. A Figura 10 mostra o sinal numa linha de comunicação assíncrona.

Num microcontrolador um módulo de comunicação serial com capacidade de transmitir e receber dados nos modos síncrono e assíncrono, configurável por software, é denominado USART (*Universal Synchronous Asynchronous Receiver Transmitter*).

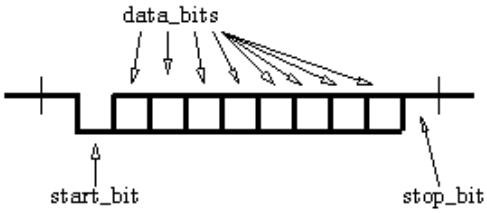


Figura 10: Sinal numa linha de comunicação assíncrona (extraído de http://www.societyofrobots.com/microcontroller_uart.shtml)

6.2 Comunicação serial assíncrona no PIC

No PIC16F886 a USART é denominada EUSART *(Enhanced Universal Synchronous Asynchronous Receiver Transmitter)* pois possui funções adicionais para poder operar em sistemas com barramento LIN (*Local Interconnect Network*). O módulo EUSART contém gerador de clock, registradores de deslocamento, buffers e sistema de interrupção para realizar a transferencia serial de dados para (transmissão) e de (recepção) outro dispositivo, independentemente da execução de um programa. A configuração assíncrona *full-duplex* é utilizada nesse curso para comunicação com outro computador e é descrita em detalhes a seguir.

A transmissão e recepção de dados pelo módulo EUSART utiliza o formato padrão NRZ (*non-return-to-zero*). O NRZ é implementado por dois níveis, um nível MARK que representa um bit de dado igual a "1" e um nível SPACE que representa um bit de dado igual a "0". O termo NRZ se refere ao fato de que dois bits transmitidos consecutivamente com o mesmo valor permanecem no valor sem retornar para um valor neutro entre a transmissão de cada bit. Numa transmissão NRZ o valor de repouso, isto é, quando não estiver sendo transmitido nenhum dado, é o nível MARK. O start-bit sempre tem um nível SPACE e o stop-bit sempre tem um nível MARK. Assim, a transmissão de um caracter consiste de um start-bit, seguido por 8 ou 9 bits de dados e terminado por um ou mais stop-bits. Cada bit transmitido permanece no seu nível pelo tempo de 1/(frequência de transmissão). O formato de dados mais comum é de 8 bits. Os 8 ou 9 bits de dados são transmitidos a partir do bit menos significativo. O transmissor e o receptor do módulo EUSART operam de maneira independente o que permite o seu funcionamento em *full-duplex*. A paridade não é suportada pelo hardware, mas pode ser gerada por software e armazenada e transmitida como o nono bit de dado.

6.2.1 Configuração da taxa de baud

O PIC 16F886 possui um gerador de Taxa de Baud dedicado para suportar tanto a operação em modo síncrono como assíncrono do módulo EUSART. Esse gerador possui um timer de 8 ou 16-bits que pode ser configurado por software. Por default esse timer opera em 8-bits, mas fazendo BRG16 = 1 (BAUDCTL<3>) pode-se alterar o timer para operação em 16-bits. Os registradores SPBRGH e SPBRG determinam a Taxa de Baud. No modo assíncrono o multiplicador do período da Taxa de Baud é determinado tanto

pelo bit BRGH (TXSTA<2>) como pelo BRG16 (BAUDCTL<3>). No modo síncrono o bit BRGH é ignorado. As fórmulas para cálculo da Taxa de Baud são apresentadas na Tabela 5 para todos os valores possíveis dos bits de configuração.

Tabela 5: Fórmulas para cálculo da Taxa de Baud de acordo com os bits de configuração (adaptada da Tabela 12-3, p. 167 do *data sheet* do PIC 16F886)

Bits de configuração			Modo	Timer	Fórmula
SYNC	BRG16	BRGH			
0	0	0	assíncrono	8-bits	$F_{OSC}/[64(n+1)]$
0	0	1	assíncrono	8-bits	$F_{OSC}/[16(n+1)]$
0	1	0	assíncrono	16-bits	
0	1	1	assíncrono	16-bits	$F_{OSC}/[4(n+1)]$
1	0	×	síncrono	8-bits	
1	1	×	síncrono	16-bits	

Por exemplo, o valor que deve ser programado no par de registradores SPBRGH e SPBRG para Taxa de Baud de 9600, em modo assíncrono, com timer de 8-bits para um dispositivo com frequência de oscilador de 16 MHz, pode ser calculado pela seguinte fórmula:

$$Taxa\,de\,Baud\,Desejada = \frac{F_{OSC}}{64([SPBRGH : SPBRG] + 1)}$$

utilizando-se BRGH = 0.

Resolvendo para $SPBRGH : SPBRG$:

$$x = \frac{\frac{F_{OSC}}{Taxa\,de\,Baud\,Desejada}}{64} - 1 = \frac{\frac{16000000}{9600}}{64} - 1 = [25.042] \cong 25$$

Para o valor de $SPBRGH : SPBRG$ calculado e arredondado, pois o valor deve ser inteiro, tem-se:

$$Taxa\,de\,Baud\,Calculada = \frac{16000000}{64(25+1)} = 9615$$

A adoção desse valor produz o seguinte erro no valor da Taxa de Baud:

$$Erro = \frac{Taxa\,de\,Baud\,Calculada - Taxa\,de\,Baud\,Desejada}{Taxa\,de\,Baud\,Desejada} = \frac{(9615 - 9600)}{9600} = 0.16\%$$

Os valores típicos para as Taxas de Baud obtidos com as fórmulas da Tabela 5 para o modo assíncrono e diferentes valores dos bits de configuração e frequências do oscilador estão na Tabela 12-5, p. 168-169, do *data sheet* do PIC 16F886. Analisando-se a Tabela 12-5 percebe-se que o uso de BRGH = 1 ou de BRG16 = 1 pode reduzir o erro da Taxa de Baud. O timer de 16-bits deve ser usado para obter Taxas de Baud mais baixas com frequências de oscilador mais altas.

A escrita de um novo valor no par de registradores SPBRGH e SPBRG causa um RESET imediato no timer do gerador de Taxa de Baud. Isso assegura que o timer não demore até seu próximo estouro para produzir a nova Taxa de Baud.

6.2.2 Configuração e operação do transmissor assíncrono

O transmissor do módulo EUSART trabalha com dois registradores, um buffer de entrada (TXREG) e um registrador de deslocamento (TSR). Somente o TXREG pode ser acessado pelo usuário, o TSR não possui endereço para acesso. Dois *flags* podem ser usados para se determinar o estado do transmissor:

- TXIF – é o *flag* de interrupção e vai para "1" quando TXREG estiver vazio
- TRMT – indica se TSR está vazio e fica em "0" enquanto TSR estiver com algum dado

O diagrama em blocos da Figura 12-1, p. 155, do *data sheet* do PIC 16F886 mostra a estrutura completa do transmissor do módulo EUSART.

A transmissão de dados é iniciada escrevendo-se um caracter no registrador TXREG. Se for o primeiro caracter, ou se o caracter anterior já foi totalmente transmitido, ou seja, se o TSR estiver limpo, o dado do TXREG é transferido imediatamente para o TSR. Por outro lado, se o TSR ainda contiver alguma parte do caracter anterior, o caracter escrito no TXREG permanecerá lá até que o stop-bit do caracter anterior tenha sido transmitido e o TSR esteja limpo. Só então o caracter do TXREG será transferido para o TSR e um novo processo de transmissão é iniciado imediatamente.

O *flag* de interrupção TXIF é setado quando o transmissor do módulo EUSART estiver habilitado e nenhum caracter estiver armazenado em TXREG, ou seja, o TXIF vai de "0" para "1" quanto o transmissor está pronto para enviar outro caracter. Esse *flag* pode ser lido a qualquer momento, mas a sua mudança para "0" após a escrita em TXREG demora um pouco e deve retornar um valor inválido se lido imediatamente após a escrita de um caracter em TXREG. O *flag* TXIF será setado toda vez que TXREG estiver vazio, independente de TXIE. O *flag* TXIF só pode ser lido e não pode ser limpo por software. Para que o *flag* TXIF gere uma interrupção os seguintes bits devem estar habilitados: TXIE, GIE e PEIE (TXIE = 1, GIE = 1 e PEIE = 1).

Para usar interrupções para transmitir dados, TXIE deve ser setado somente quanto existirem mais dados para serem transmitidos. O bit TXIE para habilitação da interrupção de transmissão deve ser limpo quando o último caracter a ser transmitido tiver sido escrito em TXREG.

O bit TRMT indica o estado do registrador TSR. Esse bit somente pode ser lido. Quando o registrador TSR está vazio, o bit TRMT é setado. Esse bit é limpo quando um caracter é transferido do registrador TXREG para o registrador TSR. O bit TRMT fica em "0" até que todos os bits do TSR tenham sido deslocados para serem transmitidos. Esse bit não causa uma interrupção e o usuário deve ler o seu valor para determinar o estado de TSR.

O transmissor pode ser habilitado para operações assíncronas através da configuração dos seguintes bits de controle:

- TXEN = 1 – habilita o circuito de transmissão do módulo EUSART
- SYNC = 0 – configura o módulo EUSART para operação em modo assíncrono
- SPEN = 1 – habilita o módulo EUSART e configura automaticamente o pino de I/O TX/CK como saída. Se o pino TX/CK tiver sido previamente utilizado como entrada analógica, o bit correspondente no registrador ANSEL deve ser limpo.

> **Nota** quando o bit SPEN é programado como "1", o pino de I/O TX/CK é automaticamente configurado como saída, independente do valor programado no TRIS correspondente e independente do receptor do módulo EUSART estar habilitado. O registrador do PORT é desconectado do driver de saída de maneira que não é possível usar o pino TX/CK como saída de uso geral.

Assume-se que todos os outros bits de controle do módulo EUSART estão no seu estado default.

A sequência a seguir resume o procedimento para utilizar o transmissor assíncrono:

1. Inicializar o par de registradores SPBRGH e SPBRG e os bits BRGH e BRG16 de acordo com a Taxa de Baud desejada.
2. Habilitar a comunicação assíncrona fazendo SYNC = 0 e SPEN = 1.
3. Se for desejada transmissão de 9-bits, fazer o bit de controle TX9 = 1.
4. Habilitar a transmissão fazendo TXEN = 1. Isso causará o bit de interrupção TXIF ir para "1".
5. Se for desejado utilizar interrupções, fazer o bit TXIE = 1. Uma interrupção ocorrerá imediatamente se GIE = 1 e PEIE = 1.
6. Se for transmissão de 9-bits, escrever o valor do nono bit em TX9D.
7. Carregar o caracter de 8-bits no registrador TXREG. Isso inicia a transmissão.

6.2.3 Configuração e operação do receptor assíncrono

O receptor do módulo EUSART trabalha com um registrador de deslocamento para receber os dados (RSR) e um buffer FIFO (First-in First-out) de dois caracteres. O registrador RSR e o buffer FIFO não são acessados diretamente. Os dados recebidos são acessados através do registrador RCREG. Três *flags* podem ser usados para se determinar o estado do receptor:

- RCIF – é o *flag* de interrupção e vai para "1" quando um caracter entrar no buffer FIFO
- FERR – é um *flag* de erro e indica que não foi detectado o stop-bit no final do caracter recebido
- OERR – é um *flag* de erro e indica que o buffer FIFO está cheio e não será recebido outro caracter até que o buffer tenha espaço

O diagrama em blocos da Figura 12-2, p. 156, do *data sheet* do PIC 16F886 mostra a estrutura completa do receptor do módulo EUSART.

Assim que um caracter é recebido no registrador de deslocamento RSR, ele é transferido imediatamente para o buffer FIFO e o *flag* RCIF é setado. Se no lugar de um stop-bit, cujo valor é "1", for recebido um "0", é gerado um erro de FRAMING, ou seja, o *flag* FERR é setado. O caracter do topo do buffer FIFO é transferido para fora do buffer na leitura do registrador RCREG. Se o buffer FIFO de dois caracteres estiver cheio é gerado um erro de OVERRUN, ou seja o *flag* OERR é setado, e nenhum caracter adicional será recebido.

O *flag* de interrupção RCIF é setado quando o receptor do módulo EUSART estiver habilitado e quando o buffer FIFO contiver um caracter não lido. Ou seja, o RCIF vai de "0"para "1" quando o receptor tiver um caracter pronto para ser lido. O *flag* RCIF será setado toda vez que um caracter chegar ao buffer FIFO, independente de RCIE. O *flag* RCIF só pode ser lido e não pode ser limpo por software. Para que o *flag* RCIF gere uma interrupção é gerada os seguintes bits devem estar habilitados: RCIE , GIE e PEIE (RCIE = 1, GIE = 1 e PEIE = 1).

Cada caracter no buffer FIFO de recepção possui um bit de erro de FRAMING correspondente. O erro de FRAMING indica que o stop-bit não foi detectado no momento em que era esperado. O estado do erro de FRAMING poder ser acessado através do bit FERR (RCSTA<2>) e representa o estado erro do caracter no topo do buffer FIFO. Portanto, o bit FERR deve ser lido antes de se ler o registrador RCREG senão será sobre-escrito pelo estado de erro de FRAMING do caracter seguinte do buffer FIFO. O bit FERR somente pode ser lido e se aplica somente ao caracter no topo do buffer FIFO. Um erro de

FRAMING não impede que caracteres adicionais sejam recebidos. Não é necessário limpar o bit FERR. A leitura do próximo caracter do buffer FIFO faz avançar a fila para o próximo caracter e consequentemente para o próximo erro de FRAMING correspondente. Pode-se força que o bit FERR seja limpo, limpando-se o bit SPEN. Fazendo-se CREN = 0 não afeta o bit FERR. Um erro de FRAMING não gera interrupção. Se todos os caracteres do buffer FIFO de recepção contiverem erros de FRAMING, a leitura de RCREG não limpará o bit FERR.

O buffer FIFO de recepção pode conter até dois caracteres. É gerado um erro de OVERRUN se for recebido um terceiro caracter antes que a FIFO tenha sido acessada. Se ocorrer essa situação o bit OERR é setado. Os caracteres que já estiverem no buffer FIFO poderão ser lidos, mas nenhum caracter adicional será recebido até que a condição de erro seja cancelada. Isso pode ser feito limpando-se o bit CREN (RCSTA<4>) ou através da reinicialização do módulo EUSART limpando-se o bit SPEN (RCSTA<5>).

O módulo EUSART suporta a recepção de caracteres com 9-bits. Quando o bit RX9 (RCSTA<6>) estiver setado, 9-bits serão deslocados para dentro do registrador RSR para cada caracter recebido. O nono bit, o mais significativo do caracter, poder ser lido no bit RX9D (RCSTA<0>). Esse bit corresponde ao nono bit do caracter do topo do buffer FIFO. Assim, ele deve ser lido antes da leitura dos 8-bits menos significativos que estão no buffer.

O receptor pode ser habilitado para operações assíncronas através da configuração dos seguintes bits de controle:

- CREN = 1 – habilita o circuito de recepção do módulo EUSART
- SYNC = 0 – configura o módulo EUSART para operação em modo assíncrono
- SPEN = 1 – habilita o módulo EUSART e configura automaticamente o pino de I/O RX/DT como entrada. Se o pino RX/DT tiver sido previamente utilizado como entrada analógica, o bit correspondente no registrador ANSEL deve ser limpo.

Nota quando o bit SPEN é programado como "1", o pino de I/O RX/DT é automaticamente configurado como entrada, independente do valor programado no TRIS correspondente e independente do receptor do módulo EUSART estar habilitado. O pino RX/DT pode ser lido normalmente, mas a escrita nesse pino não produzirá nenhum efeito.

Assume-se que todos os outros bits de controle do módulo EUSART estão no seu estado default.

A configuração e utilização da recepção assíncrona deve seguir os seguintes passos:

1. Inicializar o par de registradores SPBRGH e SPBRG e os bits BRGH e BRG16 de acordo com a Taxa de Baud desejada.
2. Habilitar a comunicação assíncrona fazendo SYNC = 0 e SPEN = 1.
3. Se for desejado utilizar interrupções, fazer o bit RCIE = 1, GIE = 1 e PEIE = 1.
4. Se for desejada recepção de 9-bits, fazer o bit RX9 = 1.
5. Habilitar a recepção fazendo o bit CREN = 1.
6. O *flag* RCIF será setado quando um caracter for transferido do RSR para o buffer de recepção e um pedido de interrupção será gerado se RCIE = 1.

7. Ler o registrador RCSTA para determinar se algum erro ocorreu durante a recepção e pegar o nono bit (se estiver habilitado).

8. Ler os 8-bits menos significativos de dados recebidos no buffer de recepção através da leitura do registrador RCREG.

9. Se ocorreu um erro de OVERRUN, limpar o *flag* OERR limpando o bit CREN.

6.2.4 Funções de biblioteca

Para configuração e utilização da EUSART em modo assíncrono no XC8 foi criada uma biblioteca com as seguintes funções:

- `void serial_init(void)` – Essa função configura todos os parâmetros necessários para a operação do canal serial a 19.200 bps, com palavra de dados de 8-bits, considerando a frequência do oscilador do microcontrolador como sendo 20 MHz. Deve ser chamada uma única vez na seção de inicialização dos dispositivos.

- `unsigned char getch(void)` – Recebe um caractere ASCII pelo canal serial. Essa função não retorna enquanto não for recebido um caractere.

- `unsigned char chkchr(void)` – Essa função verifica se existe um caractere no buffer de recepção. Se existir, retorna o caractere, senão retorna o valor 255.

- `void putch(unsigned char c)` – Envia um caractere ASCII pelo canal serial.

- `void putst(register const char * str)` – Envia um *string* pelo canal serial

Para a utilização dessas funções deve ser incluído o header `serial.h` e antes dele o `always.h` que contém definições utilizadas pelo `serial.h`.

6.3 Comunicação síncrona por SPI

A invés de se usar uma interface EUSART em modo síncrono é mais comum se utilizar uma interface SPI (*Serial Peripheral Interface*). Nessa interface serial síncrona, 8-bits de dados podem ser transmitidos e recebidos em modo *full-duplex*, ou seja, simultaneamente. Tipicamente a velocidade de transmissão de dados é de alguns MHz e Ao contrário da interface USART que pode ser usada para comunicação entre dispositvos externos, a interface SPI é usada internamente, entre a CPU e algum periférico, tal como, conversores A/D e D/A, displays, memórias, dentre outros. A operação da interface SPI é do tipo mestre/escravo (*master/slave*) onde o mestre produz o sinal de clock que é enviado para todos os escravos ao mesmo tempo. Cada escravo é selecionado com um sinal de seleção que determina com qual escravo o mestre está se comunicando num determinado momento.

Os sinais utilizados na interface SPI são os seguintes:

- SCK – ou SCLK – sinal de clock produzido pelo mestre e recebido simultaneamente por todos os escravos

- SDO – ou MOSI – saída de dados serial do mestre e entrada nos escravos

- SDI – ou MISO – entrada de dados no mestre e saída nos escravos

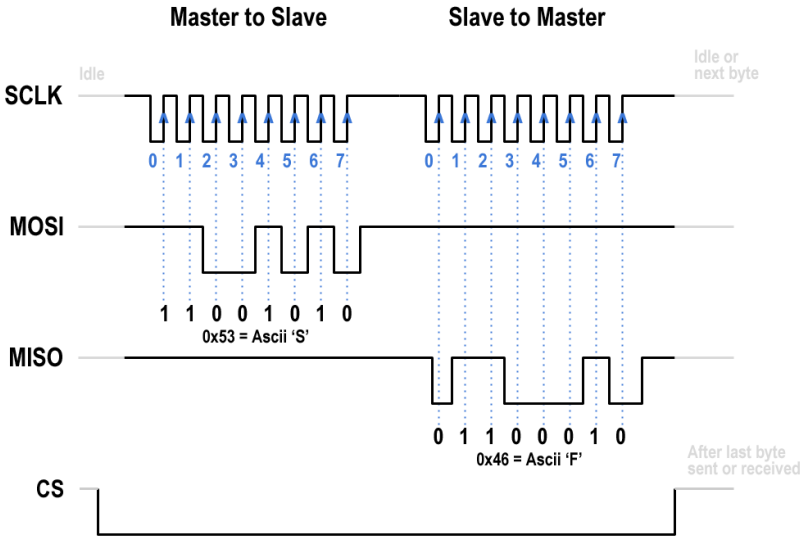


Figura 11: Sinais SPI (extraído de https://electricimp.com/docs/resources/spi/)

- SS – ou CS – sinal de seleção produzido pelo mestre, um para cada escravo

A Figura 12 mostra os sinais de conexão entre um mestre e diversos escravos. Os sinas de seleção de escravos (SSn) são gerados pelo mestre, um para cada escravo. Assim, mesmo usando poucos sinais para comunicação (SCK, SDO e SDI), se o número de escravos for grande, a quantidade total de sinais usados no SPI também é grande.

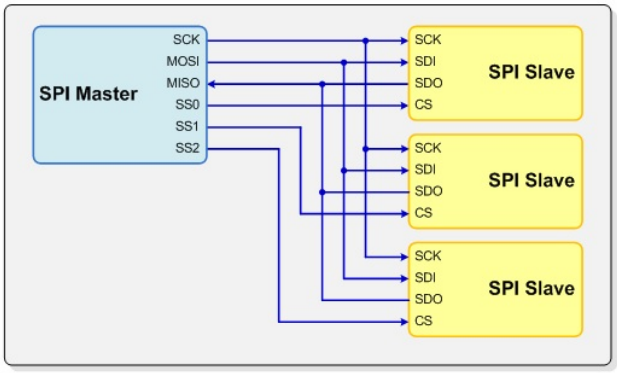


Figura 12: Conexão SPI entre mestre e diversos escravos (extraído de http://www.corelis.com/education/SPI_Tutorial.htm)

No caso de existir somente um escravo para um mestre é possível operar sem o sinal SS (CS). Contudo, caso a comunicação perca o sincronismo somente será possível re-sincronizar a interface com RESET no mestre e no escravo.

Internamente a interface SPI é implementada com um registrador de deslocamento. O registrador de deslocamento do mestre é conectado ao do escravo pelos sinais SDO (MOSI) e SDI (MISO). Assim, enquanto o mestre está transmitindo um byte para o escravo ele também está recebendo um byte do escravo. Por isso o modo de comunicação é *full-duplex*. Agora, se o mestre transmite um byte para o escravo e deve esperar o escravo dar uma resposta depois, para receber a resposta o mestre deve transmitir

alguma coisa, mesmo que sejam somente zeros ou uns e que o escravo não precisa processar. A Figura 13 mostra essa estrutura interna do mestre e do escravo. Assim, na interface SPI não se fala em leitura e escrita de dados e sim em troca (*exchange*) de dados.

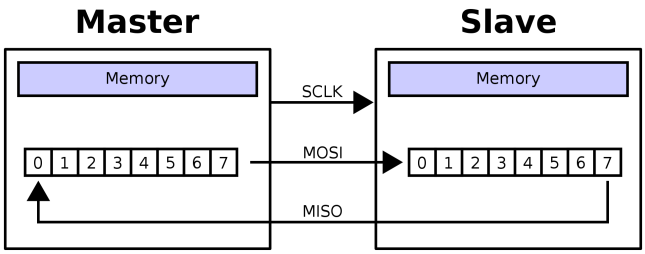


Figura 13: Estrutra interna da interface SPI (extraído de http://electronics.stackexchange.com/questions/132656/spi-reading-from-slave)

Existem 4 modos possíveis de polaridade e fase do clock em relação aos dados nesse tipo de interface serial e o modo correto deve ser configurado tanto no mestre como no escravo. A Figura 14 mostra os 4 modos do clock em relação a uma linha de dados e a Tabela 6 descreve cada um dos modos.

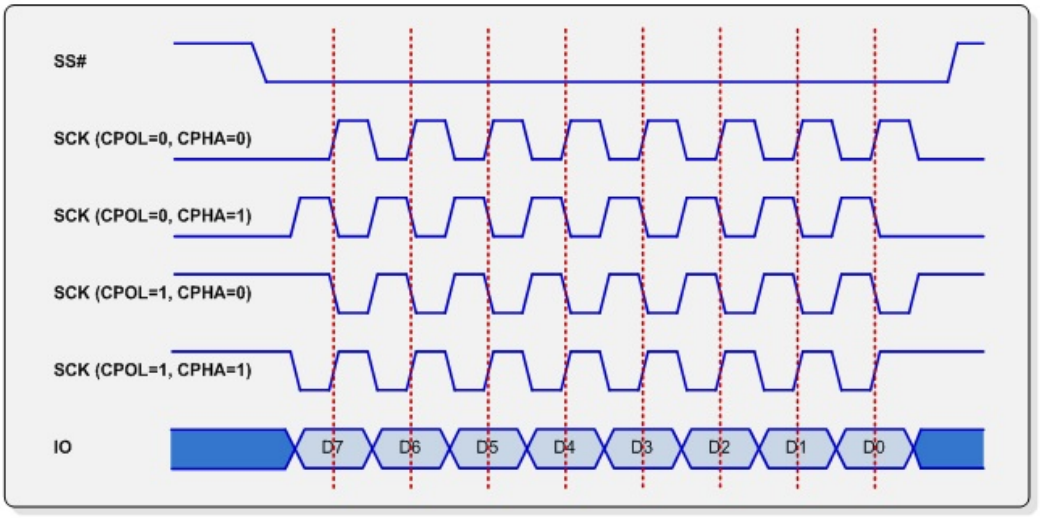


Figura 14: Modos possíveis de polaridade e fase do clock em relação aos dados na interface SPI (extraído de http://www.corelis.com/education/SPI_Tutorial.htm)

Tabela 6: Modos de polaridade e fase do clock em relação aos dados em SPI

Modo	Polaridade (CPOL)	Fase (CPHA)	Relação com dados
0	0	0	clock inativo é nível "0" e dado é amostrado na borda de subida do clock
1	0	1	clock inativo é nível "0" e dado é amostrado na borda de descida do clock
2	1	0	clock inativo é nível "1" e dado é amostrado na borda de descida do clock
3	1	1	clock inativo é nível "1" e dado é amostrado na borda de subida do clock

6.4 SPI no PIC

O PIC16F886 possui um módulo chamado *Master Synchronous Serial Port* (MSSP) que implementa tanto interfaces no modo SPI (mestre ou escravo) como no modo I^2C. Neste curso é utilizada somente a interface no modo SPI. As informações sobre a interface no modo I^2C não estão incluídas nesse documento e podem ser consultadas diretamente no *data sheet* do PIC16F886. No modo SPI são suportados os 4 modos de polaridade e fase do clock e é possível configurar a interface para operar tanto em modo mestre como em modo escravo. São usados os seguintes sinais:

- SCK – clock serial – mapeado no pino RC3, é saída no modo mestre e entrada no modo escravo
- SDO – saída serial de dados – mapeado no pino RC5, é saída tanto no modo mestre como no modo escravo
- SDI – entrada serial de dados – mapeado no pino RC4, é entrada tanto no modo mestre como no modo escravo
- $\overline{SS}$ – seleção de escravo – mapeado no pino RA5, é entrada e é usado somente no modo escravo

A inicialização do modo SPI requer a programação de uma série de bits nos registradores SSPCON<5:0> e SSPSTAT<7:6> para especificar o seguinte:

- Modo mestre (SCK é saída do clock) ou escravo (SCK é entrada de clock)
- Polaridade do clock (valor do clock inativo, se em HIGH ou LOW)
- Borda do clock (se a saída de dados é na borda de subida ou de descuda do clock)
- Amostragem do dado de entrada (se no meio ou no final do período)
- Taxa do clock (somente para modo mestre)
- Modo de seleção do escravo (somente para modo escravo, se usa ou não o sinal $\overline{SS}$)

Os pinos de I/O usados na comunicação SPI devem ser configurados de acordo com o seu uso, para entrada ou para saída:

- SDO dever ter o bit TRISC5 = 0
- SCK deve ter o bit TRISC3 = 0, se estiver em modo mestre
- SCK deve ter o bit TRISC3 = 1, se estiver em modo escravo
- $\overline{SS}$ deve ter o bit TRISA5 = 1, se estiver em modo escravo (não é usado em modo mestre)

Tanto a transmissão de um byte como a recepção de um byte é feita através do registrador SSPBUF. Para se transmitir um byte deve-se escrever no registrador SSPBUF. Ao se receber um byte ele estará no SSPBUF e o *flag* BF do registrador SSPSTAT será levantado junto com o *flag* SSPIF do registrador PIR1. Qualquer escrita em SSPBUF durante a transmissão ou recepção é ignorada e o *flag* de detecção de colisão WCOL do registrador SSPCON será levantado. A leitura de um byte recebido de SSPBUF limpa o *flag* BF. Os *flags* WCOL e SSPIF devem ser limpos pelo software do usuário.

6.4.1 Modo mestre

Somente a interface SPI no modo mestre pode iniciar uma tranferência de dados, seja para transmissão como para recepção. É sempre o mestre que controla o clock (SCK) como saída. A polaridade do clock deve ser definida através da programação do bit CKP do registrador SSPCON enquanto que a borda para dados estáveis no SDO é definida pelo bit CKE de registrador SSPSTAT. A Tabela 7 mostra a correspondência entre CKP e CKE com os modos de polaridade de fase do clock da interface SPI.

Tabela 7: Correspondência entre os modos de polaridade e fase do clock com CKP e CKE

Modo SPI	CKP	CKE
0	0	1
1	0	0
2	1	1
3	1	0

A taxa de transmissão, ou seja, a frequencia do clock, da interface SPI deve ser definida para o modo mestre. As seguintes opções estão dsiponíveis:

- $F_{OSC}/4$
- $F_{OSC}/16$
- $F_{OSC}/64$
- saída do Timer 2 $\div$ 2

Essa configurações permitem taxas de até 5 MHz para clock de 20 MHz do PIC.

No modo mestre os sinais para seleção dos escravos ($\overline{SS}$) devem ser gerados por pinos de I/O genéricos configurados como saída e em nível inativo HIGH. Esses pinos devem ser levados para LOW individualmente antes do início da comunicação SPI com um determinado escravo e retornados para HIGH após o término da comunicação. Devem ser usados tantos pinos de I/O quantos forem os escravos, um para cada, e devem ser acionados um de cada vez. Não devem haver dois ou mais pinos de seleção de escravos em LOW ao mesmo tempo, com risco de curto circuito nas saídas SDO dos escravos.

6.4.2 Modo escravo

No modo escravo a transmissão e recepção de dados é controlada pelo sinal de clock (SCK) vindo do mestre. A polaridade do clock (CKP) e a borda (CKE) devem ser configurados no mesmo modo SPI que o mestre para o correto funcionamento da interface. Ao receber um dado do mestre, o *flag* de interrupção SSPIF do registrador PIR1é levantado. O sinal $\overline{SS}$ é entrada no modo escravo e serve para selecionar o escravo para comunicação com o mestre tirando sua saída SDO do tri-state. No caso de um único escravo para um mestre o sinal $\overline{SS}$ pode ser dispensado. Contudo, o seu uso é obrigatório se CKE = 1, independentemente da quantidade de escravos. Outra função do sinal $\overline{SS}$ é re-sincronizar a comunicação a cada byte transmitido ou recebido caso a comunicação saia do sincronismo. Caso esse sinal não esteja sendo usado a única maneira de re-sincronizar a comunicação é através do RESET do mestre e do escravo.

6.4.3 Funções de biblioteca

Para configuração e utilização da interface SPI no XC8 foi criada uma biblioteca com as seguintes funções:

- `void spi_init(void)` – Essa função configura a interface SPI para operação em modo mestre com polaridade e fase de clock em modo 1, amostragem do dado de entrada no meio do período, saída de SS no pino RB0, entrada de *flag* de dado pronto do escravo pelo pino RB0 e taxa de transmissão em 1,25 MHz.
- `char spi_exchange(char data)` – Realiza uma troca de dado entre o mestre e o escravo. O parâmetro `data` é transmitido enquanto que o retorno é o byte recebido do escravo.

Para a utilização dessas funções deve ser incluído o header `spi.h`.

Capítulo 7

PWM

PWM significa *Pulse Width Modulation* ou modulação por largura de pulso. Um sinal de PWM possui um período constante e um tempo em nível lógico HIGH (o pulso) variável. A cada ciclo o tempo que o sinal permanece no nível lógico HIGH em relação ao período total pode ser descrito por uma porcentagem. Essa porcentagem é chamada de *Duty Cycle*.

$$Duty\,Cycle = \frac{Tempo\,em\,nível\,lógico\,HIGH}{Período\,do\,PWM}$$

Por exemplo, um sinal de PWM com *Duty Cycle* de 50% tem a forma de uma onda quadrada, na metade do período o sinal está em nível lógico HIGH e na outra metade do período o sinal está em nível lógico LOW. A Figura 15 mostra algumas situações para os valores do *Duty Cycle*.

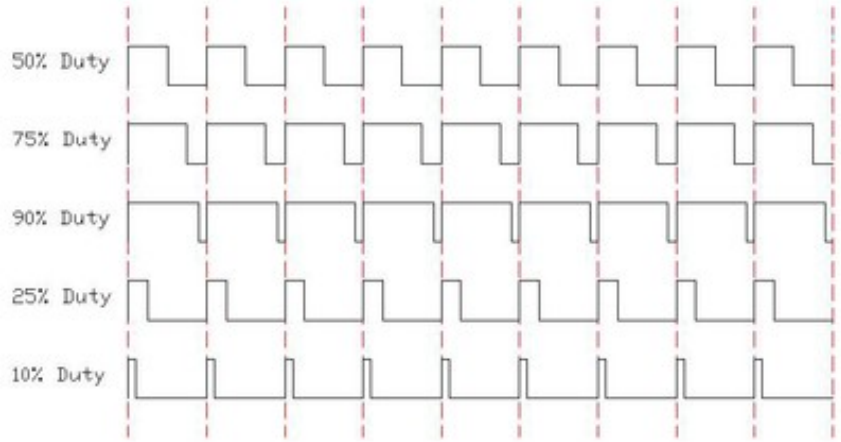


Figura 15: Forma de onda de um sinal PWM com vários valores de *Duty Cycle* (extraído de https://arduino-info.wikispaces.com/Analog-Output)

Um sinal de PWM pode ser utilizado para variar a energia entregue a um determinado dispositivo. Um valor de 0% de *Duty Cycle* significa que nenhuma energia é entregue ao dispositivo. Por outro lado, 50% de *Duty Cycle* significa que 50% de energia é entregue ao dispositivo. Uma das aplicações de um sinal de PWM é o controle a corrente que passa pelo enrolamento de um motor CC e consequentemente variação de sua velocidade. A Figura 16 mostra um arranjo para controle de velocidade em malha fechada de um motor CC com um microcontrolador gerando um sinal de PWM e lendo sinais digitais de um encoder. A

leitura de encoder será explicada próximo capítulo.

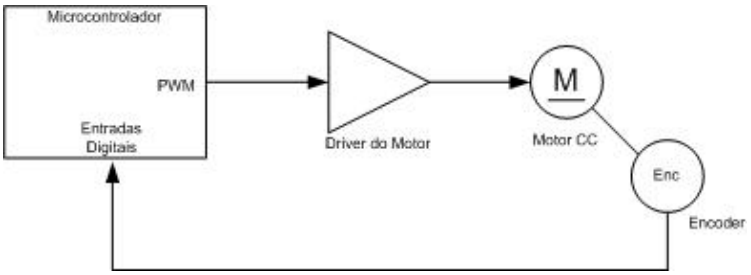


Figura 16: Controle de velocidade de um motor CC com um microcontrolador gerando sinal de PWM.

7.1 PWM com o PIC

O PIC 16F886 possui um módulo Enhanced Capture/Compare/PWM (CCP1) e um módulo Capture/-Compare/PWM (CCP2). Esses módulos são idênticos na operação com exceção das caraterísticas avançadas de PWM disponíveis somente no CCP1. Com isso é possível gerar dois sinais de PWM com esse dispositivo, um através do pino CCP1 e outro através do pino CCP2.

O Timer 2 é utilizado para dar a base de tempo para o PWM. O prescaler do Timer 2 é usado para dar o período do PWM, enquanto que os registradores CCPRxL e CCPRxH e os bits CCPxCON<5:4> são usados para definir a largura do pulso do PWM. O postscaler do Timer 2 não é utilizado no modo PWM. O módulo CCP consegue produzir um sinal PWM com resolução de até 10-bits[1] num dos pinos de saída CCPx. O valor do *Duty Cycle* com a resolução de 10-bits é armazenado no registrador CCPRxL (8-bits mais significativos) e nos bits CCPxCON<5:4> (2-bits menos significativos). O registrador CCPRxH (mais dois bits internos do dispositivo) é usado como escravo do conjunto CCPRxL:CCPxCON<5:4> para que esse possa ser alterado a qualquer momento mantendo estável a largura do pulso sendo produzido até que se complete o ciclo. A Figura 11-3, p. 132, do data sheet do PIC 16F886 mostra um diagrama de blocos simplificado do PWM.

A Figura 17 mostra a saída de um pino CCPx e a sua relação com os valores de PR2 e *Duty Cycle* dado por CCPRxL e CCPxCON<5:4>. O Timer 2 opera numa contagem contínua com uma frequência derivada do oscilador dividida pelo seu prescaler. No início do período do PWM o Timer 2 é zerado, a saída CCPx é colocada em nível lógico HIGH e o valor do conjunto CCPRxL:CCPxCON<5:4> é armazenado no registrador CCPRxH e em dois bits internos. Quando a contagem do Timer 2 atinge o valor composto por CCPRxH e pelos dois bits internos, o valor da saída CCPx é colocado em nível lógico LOW. A contagem do Timer 2 prossegue até que o seu valor atinja o valor de PR2. Nesse momento o Timer 2 é zerado, a saída CCPx é colocada em nível lógico HIGH e o valor do conjunto CCPRxL:CCPxCON<5:4> é novamente carregado no registrador CCPRxH e nos dois bits internos. Esse ciclo se repete indefinidamente enquanto o modo PWM estiver habilitado.

O período do PWM é especificado pelo valor armazenado no registrador PR2 e é calculado pela seguinte fórmula:

$$Período\,do\,PWM = [(PR2) + 1] \times 4 \times T_{OSC} \times (Prescale\,do\,Timer\,2)$$

[1] Os 8-bits do Timer 2 são concatenados com 2-bits internos do clock do sistema (F_{OSC}), ou 2-bits do prescaler, para criar uma base de tempo de 10-bits.

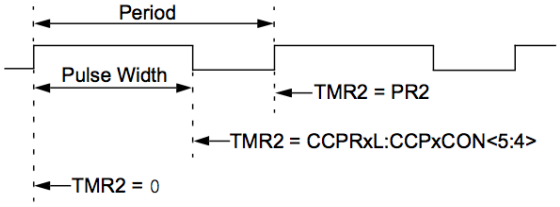


Figura 17: Sinal de saída de PWM num pino CCPx (extraído de [2])

O Prescaler do Timer 2 pode ser definido como 1:1, 1:4 ou 1:16 definindo-se os bits T2CKPS<1:0> (T2CON<1:0>) e T_{OSC} é o período do oscilador.

O *Duty Cycle* do PWM é especificado escrevendo-se um valor de 10-bits, sendo os 8-bits mais significativos escritos no registrador CCPRxL e os 2-bits menos significativos escritos nos bits 5 e 4 do registrador CCPxCON. A seguinte fórmula é utilizada para calcular o a largura do pulso do PWM:

$$Largura\,do\,Pulso\,de\,PWM = (CCPRxL : CCPxCON < 5 : 4 >) \times T_{OSC} \times (Prescale\,do\,Timer\,2)$$

O valor do *Duty Cycle* pode ser dado em porcentagem pela seguinte fórmula:

$$Duty\,Cycle = \frac{(CCPRxL : CCPxCON < 5 : 4 >)}{4(PR2 + 1)}$$

O registrador CCPRxL e os bits 5 e 4 de CCPxCON podem ser escritos a qualquer momento, mas o valor do *Duty Cycle* somente será alterado quando o valor de CCPRxL for armazenado em CCPRxH o que ocorre somente após o valor do Timer 2 atingir o mesmo valor que PR2, ou seja, somente quando o período do PWM estiver completo. No modo PWM o registrador CCPRxH somente pode ser lido.

No PIC 16F886 os bits 5 e 4 do registrador CCPxCOM também podem ser acessados por nomes específicos: DCxB1 e DCxB0, respectivamente.

7.1.1 Procedimento para configuração do PWM

Os seguintes passos devem ser feitos para configurar o módulo CCP para operação em PWM:

1. Desabilitar o pino de PWM (CCPx) setando o bit correspondente do registrador TRIS. Isto é, configurando-o para entrada
2. Definir o período do PWM escrevendo no registrador PR2.
3. Configurar o módulo CCP para modo PWM carregando o registrador CCPxCON com os valores apropriados.
4. Definir o *Duty Cycle* do PWM escrevendo no registrador CCPRxL e nos bits DCxB<1:0> (CCPxCON<5:4>).
5. Configurar e disparar o Timer 2:
   (a) Limpar o bit do *flag* de interrupção TMR2IF (PIR1<1>);

(b) Definir o valor de prescaler do Timer 2 carregando os bits T2CKPS<1:0> (T2CON<1:0>);

(c) Habilitar o Timer 2, fazendo o bit TMR2ON =1 (T2CON<2>).

6. Habilitar a saída PWM depois que um novo ciclo de PWM tiver sido iniciado:

(a) Esperar pelo transbordo do Timer 2 (bit TMR2IF = 1 (PIR1<1>));

(b) Habilitar a saída do pino de PWM (CCPx) limpando o bit correspondente do registrador TRIS. Isto é, configurando-o para saída.

Chapter 8

Encoder

O encoder incremental serve para converter a posição de um eixo em uma quantidade de pulsos. Esses pulsos são gerados por sensores ópticos que detectam alterações de intensidade luminosa e convertem essas alterações para sinais digitais que podem ser processados por um microcontrolador. Tipicamente as alterações de intensidade são obtidas por um disco ligado ao eixo do motor que alternam padrões claros e escuros, conforme mostrado na Figura 18. Para se determinar a direção de rotação desse disco, um par de sensores é colocado num arranjo em quadratura, ou seja defasados em 90°, o que produz as formas de onda mostradas na Figura 18 como saídas dos canais A e B.

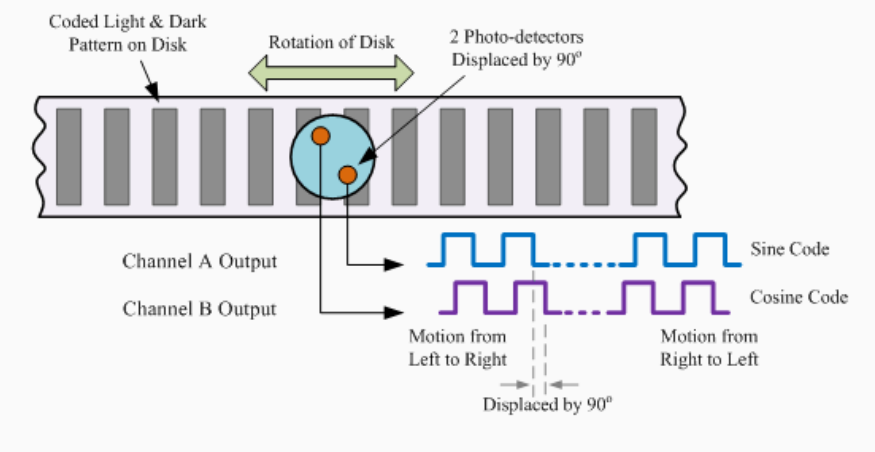


Figura 18: Disco codificado com regiões claras e escuras, sensores ópticos em quadratura, sinais dos canais A e B para rotação no sentido horário e anti-horário (extraído de [6])

Interpretando-se os canais A e B como um código binário de 2 bits, tem-se o seguinte, segundo a Figura 19:

- Se a rotação do disco for no sentido horário, a sequência de códigos produzida será 0, 1, 3, 2, 0, e assim por diante
- Se a rotação do disco for no sentido anti-horário, a sequência de códigos produzida será 0, 2, 3, 1, 0, e assim por diante.

Se os códigos gerados pelos canais A e B forem considerados como estados, pode-se visualizar a máquina de estados mostrada na Figura 20. Se o encoder estiver parado numa posição correspondente ao estado

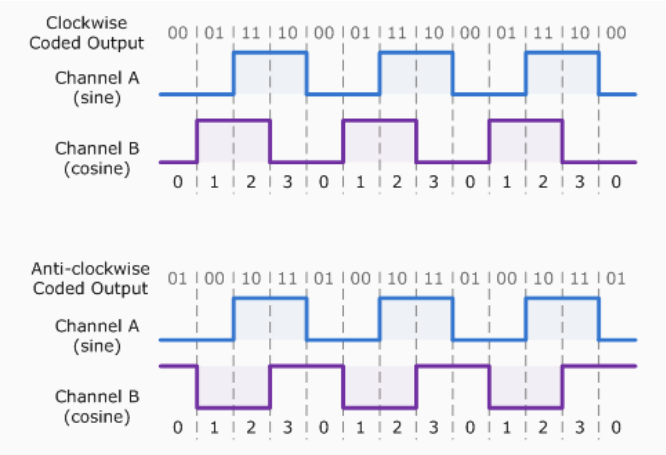


Figura 19: Sinais dos canais A e B mostrados como um par de bits para o sentido horário e anti-horário (extraído de [6])

0, o movimento do eixo do motor e consequentemente do disco do encoder causará uma mudança de estado. A mudança do estado 0 para o estado 1 pode causar o incremento de um contador, enquanto que a mudança para o estado 2 pode causar o decremento deste mesmo contador. Assim contabilizando as mudanças de estado num sentido e no outro pode-se determinar efetivamente o deslocamento angular do eixo do motor, levando-se em consideração inclusive a direção de rotação.

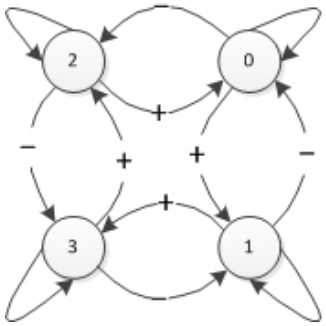


Figura 20: Máquina de estados para algoritmo de leitura de encoder.

Como consequência deste método, tem-se:

- O número de regiões claras ou escuras corresponde a quantidade de pulsos do canal A ou B por volta. Assim, se o disco do encoder tiver 12 regiões claras e 12 regiões escuras é possível produzir 12 pulsos no sinal do canal A e 12 pulsos no sinal do canal B. Ainda, com 12 regiões pode-se produzir um pulso a cada 30° de rotação do eixo do motor.
- Considerando-se que o sinal do canal A combinado com o sinal do canal B produz os 4 estados da máquina de estados, pode-se incrementar ou decrementar a contagem 4 vezes. Dessa maneira numa volta completa do eixo do motor pode-se ter um incremento, ou decremento, de 48. Isso significa que cada incremento ou decremento corresponde a 7,5° de rotação do eixo.

Deve-se observar que a quantidade de pulsos determina o deslocamento angular do eixo do motor, enquanto que a sua frequência determina a velocidade angular do eixo do motor.

8.1 Leitura do encoder com Interrupt-On-Change

Utilizando-se um microcontrolador pode-se conectar os sinais dos canais A e B em um par de bits de uma porta de entrada digital. Um algoritmo que monitora o estado desses 2 bits pode incrementar e decrementar um contador possibilitando, assim, o cálculo do descolamento angular do eixo do motor. A contabilização de mudanças de estados num intervalo de tempo possibilita o cálculo da velocidade angular do eixo do motor.

Se esses 2 bits forem conectados à Porta B do PIC16F886, pode-se usar a Interrupção na mudança para se determinar cada um dos estados desses dois bits. Como ocorre uma interrupção tanto para a borda de subida como para a de descida, todos os estados podem ser detectados. No caso do conjunto motor e encoder do RoboLab 2, na sua velocidade máxima de aproximadamente 550 mm/s serão produzidas interrupções a cada 4,5 ms, que é uma taxa possível de ser tratada pelo PIC. Caso a taxa de interrupções seja muito alta esse método não é aconselhável.

A partir do diagrama da máquina de estados da Figura 20, pode-se criar uma matriz de transição de estados que pode ser usada diretamente para incrementar ou decrementar o contador do encoder:

```
long encoder_counter += transition_matrix[old_state][new_state];
```

Referências Bibliográficas

[1] Hitachi; HD44780U (LCD-II) (Dot Matrix Liquid Crystal Display Controller/Driver); Rev. 0.0; 1999

[2] Microchip Technology Inc.; PIC16F882/883/884/886/887 Data Sheet 28/40/44-Pin Flash-Based, 8-Bot CMOS Microcontrollers; 2012

[3] Microchip Technology Inc.; MPLAB X User's Guide; disponível em http://ww1.microchip.com/downloads/en/DeviceDoc/52027B.pdf; acessado em 15/06/2013; 2012

[4] Microchip Technology Inc.; MPLAB XC8 C Compiler User's Guide; disponível em http://ww1.microchip.com/downloads/en/DeviceDoc/52053B.pdf; acessado em 15/06/2013; 2012

[5] Radartutorial.eu; Radar Basics - Analogue-to-Digital Converter; http://www.radartutorial.eu/17.bauteile/bt45.en.html; acessado em 14/06/2013

[6] Storr, Wayne; Basic Electronics Tutorials: Position Sensors; http://www.electronics-tutorials.ws/io/io_2.html; acessado em 31/01/2011

Parte II

Atividades de Laboratório

Capítulo 9

Orientações gerais

Nesse laboratório os alunos montarão seu próprio hardware de uma placa processadora e desenvolverão o software que lhes permitirá explorar os diversos aspectos do uso de microcontroladores em automação e robótica. Dentre as atividades estão a aquisição de dados digitais e analógicos, apresentação de dados em LED e display de cristal líquido (LCD), comunicação serial, acionamento de motor por PWM e leitura de posição por encoder. As atividades de laboratório seguem em sincronismo com as aulas de teoria onde, dentre outros temas, será abordado um tema que serve como base para a atividade prática antes do início do período da atividade em laboratório.

As atividades práticas tem início antes da primeira aula presencial no laboratório. Durante as 2 primeiras semanas do curso os alunos deverão se preparar para a fabricação de sua própria placa de circuito impresso (PCB) e trazer uma versão do layout da placa a ser fabricada para o dia da primeira aula em laboratório que corresponde a Atividade 2. É adotado nesse curso o software DipTrace para a elaboração do layout do PCB. Nesse período os alunos deverão estudar o software através de um tutorial que acompanha o software e fazer uma versão inicial do seu layout.

A atividade 2 consiste na avaliação pelo professor do layout feito, ajustes pelos alunos e geração de arquivos para a fabricação do PCB. A placa projetada será fabricada numa máquina de prototipagem rápida e cada grupo realizará a montagem e testes da sua placa na atividade 3.

A partir da atividade 4 os programas deverão ser desenvolvidos antes da aula de laboratório correspondente.

ATENÇÃO: No dia da aula, antes do início da atividade de laboratório o grupo de alunos deverá **entregar uma listagem impressa do programa desenvolvido para o professor**.

O professor avaliará o código apresentado e orientará os alunos como devem prosseguir. Durante a aula os alunos deverão executar o programa desenvolvido na própria placa processadora, fazer as correções e ajustes necessários e demonstrar para o professor o funcionamento. Em algumas das atividades deverão ser feitas medidas com o osciloscópio que deverão ser apresentadas e analisadas no relatório. O relatório de cada atividade deverá ser entregue no início da atividade seguinte. Após o término da atividade de laboratório os alunos deverão fazer um backup do software na sua versão final que deverá ser entregue no relatório. Os alunos são responsáveis por apagar do computador do laboratório todos os seus arquivos, deixando o computador limpo para utilização por outros grupos de alunos. Não serão toleradas cópias dos programas entre os grupos de alunos.

Os computadores do laboratório possuem os pacotes de software necessários para o desenvolvimento do software durante o semestre. Mesmo assim, é permitido aos alunos usarem os próprios notebooks no laboratório. Para tanto deverão ser instalados os seguintes pacotes de software:

- DipTrace – pacote de software EDA para esquemáticos e layout de placas de circuito impresso
- MPLAB X – ambiente de desenvolvimento de software para microcontroladores PIC da Microchip baseado no NetBeans
- MAPLAB XC8 – compilador em Linguagem C para microcontroladores PIC de 8-bits da Microchip
- Bootloader AN1310 – utilizado para carregar o programa dos alunos na memória de programa do PIC pelo canal serial. Possui emulador de terminal incorporado
- Driver FTDI (VCP) – usado para interface USB-serial para comunicação entre o Bootloader e o PIC

Todos esses pacotes de software são disponíveis na Internet em versões sem custo que podem ser usados durante o semestre e atendem completamente as necessidades do curso e do PI-7.

9.1 Critérios de avaliação

O software desenvolvido em cada atividade desse curso, a partir da Atividade 4, será avaliado quanto a funcionalidade, eficiência e forma. Os alunos devem procurar escrever códigos conceitualmente corretos, não é suficiente apresentar o programa funcionando. Deve se ter muito cuidado ao desenvolver software para microcontroladores pequenos pois a quantidade de memória disponível tanto para programa como para dados é muito pequena. Assim, não deve ser desperdiçado espaço com variáveis superdimensionadas e mesmo não utilizadas. Esses pontos são levados em consideração na avaliação do código entregue no relatório.

Quanto à forma, no código entregue devem constar obrigatoriamente os seguintes itens:

- Cabeçalho de todas as funções descrevendo o que fazem e os seus parâmetros de entrada e saída
- Comentários explicando a utilização de cada uma das variáveis globais e locais
- Comentários nos trechos de programa que expliquem o seu funcionamento.

> **ATENÇÃO:** Não serão tolerados códigos iguais entre grupos de alunos, resultando em nota zero no relatório para todos os envolvidos.

A entrega do código fonte no início da atividade de laboratório é obrigatória e caso o grupo de alunos não entregue a listagem com o código fonte haverá prejuízo de nota.

Um dos objetivos desse curso é que os alunos aprendam a utilizar bem o osciloscópio como instrumento de medida. Assim, durante o semestre serão coletadas formas de onda com o osciloscópio que deve ser ajustado de maneira a permitir uma boa análise dos tempos e tensões observados. Esse ponto é levado em consideração na avaliação.

Ao término da atividade de laboratório, o professor deve ser chamado para verificar se a implementação atende a especificação da atividade, caso contrário haverá prejuízo de nota.

A cada atividade o aluno deve escrever um relatório que deve ser entregue no início da atividade seguinte. O último relatório deve ser entregue em até uma semana após o término da atividade no escaninho do professor na secretaria do PMR. A não entrega de qualquer relatório na data prevista resulta prejuízo da nota por atraso.

Inicialmente todos os relatórios valem 10,0 pontos, o não atendimento aos critérios de avaliação vai descontando pontos do total. A Tabela 8 apresenta a pontuação a ser descontada para cada critério.

Tabela 8: Critérios de avaliação das atividades

Critérios	Pontuação
Sobre entregas	
Atraso na entrega do conjunto de arquivos GERBER e NC DRILL da atividade 2	– 100% na Atividade 3
Entregas de arquivos GERBER e NC DRILL que não estejam em condições de serem fabricados (Existindo tempo hábil os alunos serão comunicados dos problemas encontrados e poderão corrigir a placa. Não havendo tempo hábil para correção a placa não será fabricada)	– 100% na Atividade 3
Falta de entrega de código no início da aula da atividade 4 em diante	– 50%
Atraso na entrega do relatório	–1 por semana
Sobre avaliação no dia do laboratório	
Não verificação do resultado da atividade pelo professor	–1
Resultado fora das especificações	– 0,5
Sobre o relatório	
Respostas erradas para as perguntas	–1 por erro
Resposta fora de contexto	–1 por resposta
Formas de onda sem indicação de tensões e tempos na figura sem posicionamento claro dos cursores do osciloscópio e sem texto explicativo e análise	–1
Formas de onda coletadas com Auto Set. Essa é uma indicação de que o aluno não sabe ajustar os parâmetros de medida corretamente.	–1
Gráficos sem unidades, sem texto explicativo e análise	–1
Sobre o código fonte no relatório	
Trechos de código fora de contexto da atividade	–1
Trechos sem comentários ou comentários fora de contexto	–1
Código sem a estrutura correta conforme apresentado no item 1.2 e sem as particularidades descritas no enunciado da atividade	–1
Código sem devida indentação	–1
Código ineficiente no uso da memória ou no desempenho	–1
Variáveis globais sem a devida explicação ou desnecessárias	–1

9.2 Material para as atividades

A partir da atividade 4 de laboratório a placa processadora fabricada, montada e testada é utilizada para que o aluno possa desenvolver programas em Linguagem C que permitam o acesso direto a recursos de hardware que interfaceiam com o mundo real. Assim, são desenvolvidos programas para apresentação de informações em display de cristal líquido, comunicação serial, conversão de sinais analógicos em digitais, controle de velocidade de motor de corrente contínua e leitura de posição do eixo do motor. Para tanto

é utilizado um robô móvel no qual é conectado a placa processadora desenvolvida pelos alunos e possui diversos recursos de acionamento de motores, sensores, display LCD, dentre outros. A Figura 21 mostra esse robô móvel com uma placa processadora[1]. Esse robô possui bateria própria para alimentação do próprio robô e da placa desenvolvida, podendo ser recarregada por meio de carregador USB.

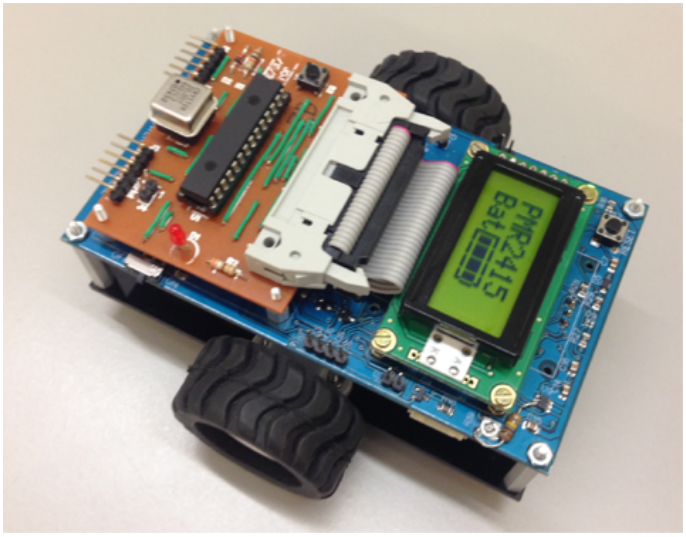


Figura 21: Robô móvel com placa processadora

A comunicação serial entre a placa processadora e um terminal é feita através de um conversor USB/serial, disponível em cada bancada do laboratório. A Figura 22 mostra o conversor utilizado. O lado do USB é conectado no computador, também disponível em cada bancada do laboratório, através de um cabo USB / mini USB. Esse conversor utiliza um chip FT232RL da Future Technologies Devices International Ltd. (FTDI), que disponibiliza o driver para Windows, Mac e Linux no seu site (http://www.ftdichip.com/Drivers/D2XX.htm). Esse driver cria uma porta serial virtual (VCP - Virtual Communication Port) (p. ex: COM3 no Windows) no computador o que permite seu acesso por qualquer programa de comunicação serial. O outro lado possui um conector de 6 pinos com 0,1 pol. de espaçamento entre pinos para ser conectado diretamente na placa processadora. Os sinais presentes nesse conector são padrão serial assíncrono 0V a 5V, onde 0v corresponde ao nível lógico LOW e 5V corresponde ao nível lógico HIGH.

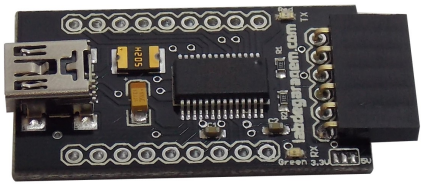


Figura 22: Interface de conversão USB para serial

A programação da placa processadora é feita sem que seja necessário retirar o microcontrolador da

[1]O robô utilizado nesta disciplina foi desenvolvido com apoio financeiro do Fundo Patrimonial Amigos da Poli entre 2015 e 2016.

placa utilizando um bootloader que envia o arquivo, em formato .HEX, gerado pelo compilador MPLAB XC8 através do canal de comunicação serial e grava o programa desenvolvido na memória de programa do microcontrolador PIC.

O kit com o robô móvel e acessórios a ser utilizado a partir da atividade 3 é fornecido numa caixa numerada que os alunos utilizarão durante as atividades no laboratório. Cada grupo de alunos utilizará o mesmo kit durante todo o semestre. No caso de necessidade de manutenção o kit será substituído por outro também numerado e alunos serão informados da troca. Ao ser completada a manutenção o mesmo kit será retornado aos grupos de alunos. Haverá um controle de quais grupos de alunos estarão utilizando os kits numerados. Os alunos deixarão suas placas processadoras dentro das caixas do kit durante o semestre, cada caixa deverá atender até 6 grupos de alunos durante o semestre. Somente os alunos participando do PI-7 poderão retirar suas placas processadoras para uso no PI-7 devendo retorná-las para o professor no final do semestre.

ATENÇÃO: Os itens que compõe o kit numerado estão relacionados abaixo e não podem ser retirados do laboratório:

- Robô móvel
- Base de borracha para apoio do robô sobre a bancada
- Carregador USB do robô com cabo USB / micro USB (branco)
- Conversor USB/serial e cabo USB / mini USB (preto)

Capítulo 10

Atividade 1: Trabalhando com DipTrace

O objetivo desta atividade é aprender a fazer um layout de uma placa de circuito impresso (PCB – Printed Circuit Board) utilizando o software de EDA (Electronic Design Automation) DipTrace. Após a realização de um tutorial do software o aluno deverá fazer o layout da própria placa processadora que será fabricada, montada, testada e utilizada durante o semestre. Para tanto é fornecido esquemático no formato do DipTrace e bibliotecas de componentes específicas para a disciplina PMR3406 que podem ser baixados do site da disciplina.

Essa atividade deve ser feita em casa e não inclui horas presenciais no laboratório. Para realizá-la serão necessárias mais horas do que as disponíveis para uma aula de laboratório que podem ser distribuídas ao longo das duas semanas que antecedem a Atividade 2. Será necessário utilizar o próprio computador e instalar o software para criação de layout de placas de circuitos impresso. O mesmo software está disponível nos computadores do laboratório. O aluno deverá fazer o Tutorial do DipTrace em video disponível no site do DipTrace (http://www.diptrace.com/diptrace-software/guided-tour/).

Para fazer o layout de acordo com as necessidades do curso devem ser configurados alguns ajustes específicos no DipTrace que vão além do que está no tutorial e estão explicados em detalhes nesse texto. Assim, o aluno deve fazer as configurações da seção 10.1.2, caso contrário não será possível fabricar a PCB corretamente.

Essa atividade deve ser feita individualmente e não em grupo. O relatório deverá ser entregue por grupo e deverá conter os resultados de cada um dos alunos do grupo.

10.1 DipTrace

O software utilizado no curso é o DipTrace que pode ser obtido numa versão sem custo com a capacidade de gerar esquemáticos e layouts com até 300 pinos, o que é suficiente para as atividades do curso. Esse software possui 4 módulos:

- Schematic Capture – para desenho de esquemáticos de circuitos eletrônicos
- PCB Layout – para desenho de placas de circuito impresso e geração de arquivos CAM usados para fabricação

- Component Editor – para definir componentes que serão usados no esquemático e manter as bibliotecas .ELI do DipTrace
- Pattern Editor – para definir footprints de componentes que serão utilizados no PCB Layout e manter as bibliotecas .LIB do DipTrace. Esse padrões de footprint contém informações sobre as furações que devem ser realizadas na fabricação dos PCBs.

O esquemático no formato do DipTrace é fornecido através do site da disciplina, bem como as bibliotecas de componentes (componentes e footprints) que contém os componentes específicos utilizados no curso. Destes 4 módulos, somente o PCB Layout será utilizado neste curso. As atividades do PI-7 e de outras disciplinas podem requerer o uso dos outros módulos.

10.1.1 Terminologia no DipTrace

Os elementos utilizados no Layout de placas de circuito impresso recebem nomes específicos no DipTrace e em outros EDAs. A Tabela 9 apresenta alguns termos comuns que são usados nesse documento e devem ficar claros para os alunos trabalharem confortavelmente com o DipTrace.

Tabela 9: Alguns termos comuns usados no DipTrace e nesse documento

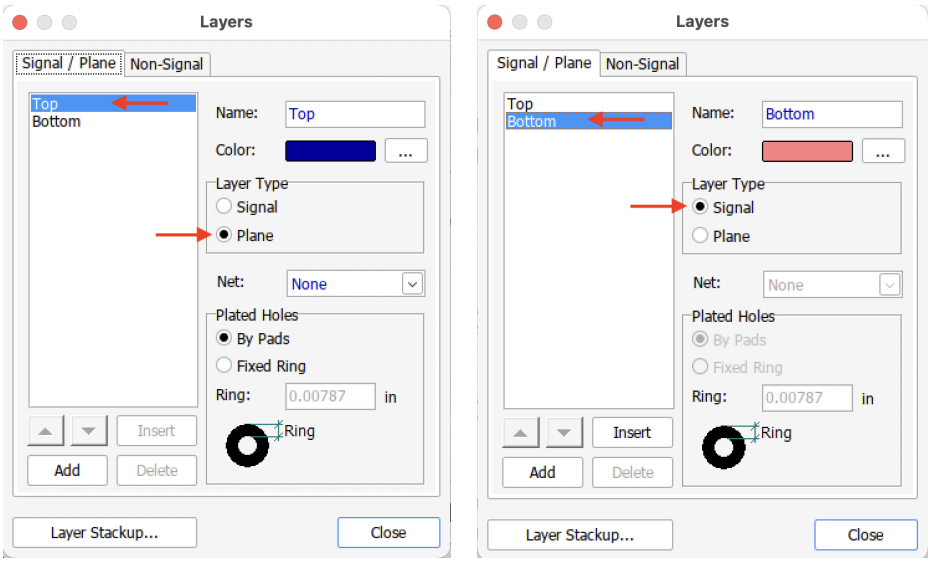
DipTrace	Em Português	Explicação
top	face de componentes	Lado da placa onde são colocados os componentes, face de cima.
bottom	face de solda	Lado da placa onde são soldados os componentes, face de baixo.
trace	trilha	Conexão entre ilhas em cobre na face de solda se for em face simples e nas duas faces se for em face dupla.
pad	ilha	Ilha onde são soldados os terminais dos componentes.
via	ilha	Ilha especial para conexão entre trilhas de uma face à outra no caso de face dupla ou para colocação de fios de jumper no caso de face simples.
copper pour		Preenchimento de uma área com cobre. Essa área pode ser conectada a algum sinal, como por exemplo o terra.
ratline		Mostra a conexão que deve ser feita entre componentes, mas ainda sem trilha roteada
mounting hole	furo de montagem	Furo para fixação da placa
through-hole	furo	Furo por onde passa o terminal de um componente que é posicionado na face de componentes e soldado na face de solda

10.1.2 Configuração para roteamento em face simples

Os PCBs a serem utilizados nesse laboratório possuem cobre somente numa das faces e o roteamento é feito somente nessa face. A face que contém o cobre é chamada de face de solda, ou Bottom no DipTrace, e a outra face é chamada de face de componentes, ou Top no DipTrace. O DipTrace fornece a opção

para dois algoritmos de auto-roteamento, o "Shape Router" e o "Grid Router". Para roteamento em face simples o indicado é o "Grid Router" que pode ser configurado através do menu do DipTrace. Para tanto, realizar as seguintes configurações:

1. Selecionar "Menu > Route > Current Autorouter > Grid Router".
2. Entrar em "Route > Layer Setup..." e configurar o "Top Layer" para "Plane" e o "Bottom Layer" para "Signal" conforme as Figuras 23a e 23b respectivamente.



(a) Configuração do Top Layer para Plane **(b)** Configuração do Bottom Layer para Signal

Figure 23: Configuração de Layer para roteamento

3. Entrar em "Route > Autorouter Setup..." e configurar, conforme Figura 24, o seguinte:
   (a) Desmarcar "Use All Layers"
   (b) Definir "Number of Layers:" como 1
   (c) Marcar "Allow Jumper Wires"
   (d) Marcar "Forbid Jumpers under Patterns"
   (e) Escolher "Jumper Wire Direction" de acordo com a direção de posicionamento do PIC. Ou seja, se o PIC estiver na horizontal, escolher "Jumper Wire Direction: Horizontal" e terminar com OK

Muitas vezes não é possível se completar 100% do roteamento pela face da solda e nesse caso são usados fios pela face de componentes que conectam segmentos de trilhas que não puderam ficar contidos numa única face. Chamamos esses fios de JUMPERs. Cada extremidade do fio de jumper é soldado num tipo de ilha específico para esse fim que recebe o nome de via no DipTrace. Os diâmetros padrão interno e externo das vias não são adequados para que sejam soldados fios de jumper e devem ser alterados. Para tanto deve ser seguindo o seguinte procedimento:

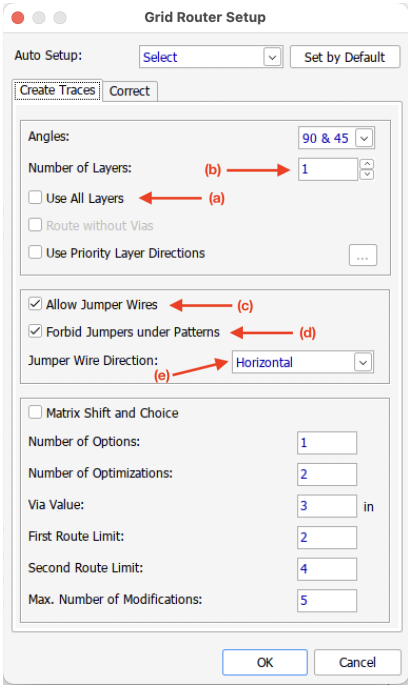


Figura 24: Configuração do Autorouter

1. Entrar em "Menu > Route > Via Styles..." e clicar no botão "Add". Será criado novo estilo de via com nome "ViaStyle1".

2. Clicar sobre o novo estilo de via e alterar os seguintes parâmetros:

   Name: de "ViaStyle1" para "Jumper"

   Outer Diameter: de 0.04 in para 0.065 in

   Hole Diameter: de 0.019 in para 0.026 in

   Terminar com OK. Veja o estado final da janela de diálogo na Figura 25.

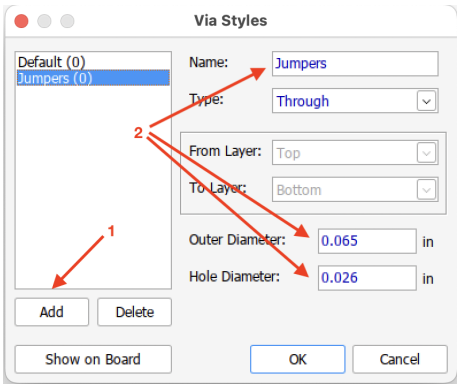


Figura 25: Configuração de estilo de via

3. Entrar em "Menu > Route> NetClasses" e em "Via Styles", desmarcar "Use All Styles"

4. Ainda em "Via Styles", selecionar "Jumper" à direita e passar para a esquerda com o botão "<<". Depois terminar com OK. Veja os estado final da janela de diálogo na Figura 26.

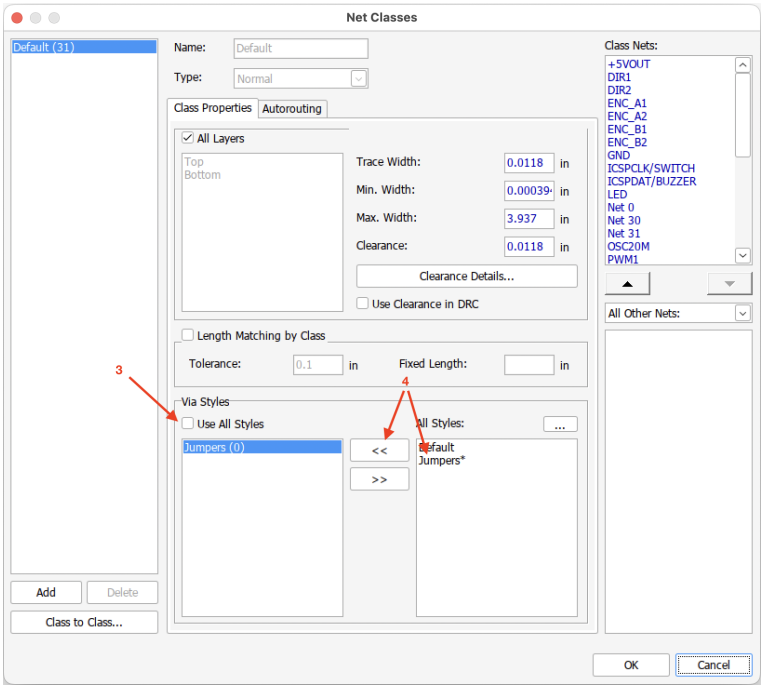


Figura 26: Definir uso do estilo de via para jumpers

Como passar um jumper manualmente

O algoritmo de roteamento "Grid Router" tenta passar os jumpers automaticamente, mas é comum não conseguir resolver todos os casos de maneira que sobram trilhas para serem roteadas manualmente e pode ser que o roteamento manual requeira a colocação de fios de jumper. O procedimento a seguir explica como colocar um jumper manualmente[1]:

1. Inicie o roteamento manual na face de solda através do botão da barra de ferramentas ( ).
2. Clique no pad onde deseja iniciar o roteamento e coloque a trilha deslocando o mouse. Na posição onde quiser iniciar o jumper clique com o mouse (Figura 27a).
3. Pressione a tecla "J" e será colocada imediatamente uma via e a vista mudará para a face de componentes (Figura 27b).
4. Estenda o jumper wire até a posição final desejada movendo o mouse (Figura 27c).
5. Clique com o mouse e será colocada a outra via para o jumper wire e a vista mudará para a face de solda novamente. Continue a trilha na face de solda até terminar a conexão desejada (Figura 27d).

10.1.3 Posicionamento das chaves S1 e S2

As chaves S1 e S2 são ligadas em paralelo e somente uma das duas será montada. O layout é feito com dois modelos de chave pois não é possível especificar exatamente o modelo quando realizamos um processo de compra e deve ser possível montar qualquer um dos modelos nas placas. Assim, as duas chaves devem

[1] As trilhas passam pela face da solda, assim como os pads e vias. Os jumpers são soldados nas vias e são colocados na face de componentes.

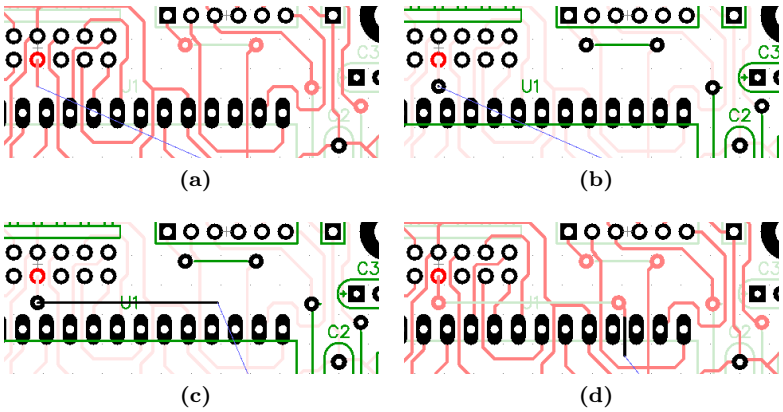


Figura 27: Como passar jumper wires manualmente

ser posicionadas com seus centros aproximadamente coincidentes de acordo com a Figura 28. A grade adotada para posicionamento não permite uma centralização precisa dos centros das chaves. É necessário ajustar as coordenadas X e Y manualmente em milésimos de polegada.

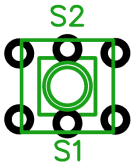
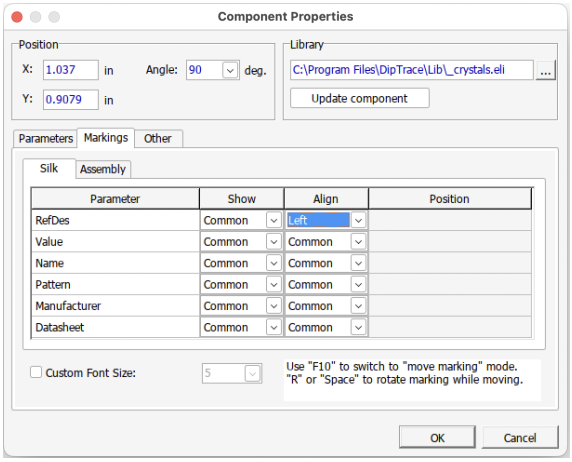


Figura 28: Posicionamento das chaves S1 e S2

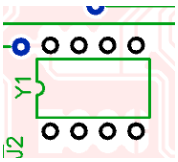
10.1.4 Legendas dos componentes

Todas as legendas dos componentes, tais como U1, Y1, S1, etc. são gravados no lado dos componentes da placa impressa e servem de guia na montagem. Todas as legendas devem estar na camada TOP SILK. No DipTrace, normalmente essas marcações estão em cor verde. Assim, deve-se ter cuidado para que as inscrições nesta camada não se sobreponham e particularmente, as legendas não sejam posicionadas dentro do perímetro dos componentes e nem sobre outras legendas ou marcações. Para ajustar o posicionamento das legendas deve-se clicar com o botão da direita sobre um componente selecionado e escolher "Properties...". Será aberta uma tela para configurações de diversas propriedades do componente. Deverá ser escolhido o segundo tab, "Markings". O default é mostrar o "RefDes" do componente e colocá-lo no topo. No exemplo da Figura 29a, a posição do "RefDes" foi alterada para a esquerda para não sobrepor sobre outras marcações o que pode ser visto na coluna "Align" na qual foi selecionado "Left" para o "RefDes". O resultado é mostrado na Figura 29b.

No caso do componente J2 é necessário mostrar uma legenda adicional para orientação na conexão correta durante a utilização da placa. A função do conector foi definida no componente no campo "Value" com o texto "Serial". Assim, na mesma tela da Figura 29a pode-se definir a visualização do parâmetro "Value" na coluna "Show" e o posicionamento a cima do componente na coluna "Align" com a seleção de "Top", conforme a Figura 30a. Além disso, a Figura 30a mostra o uso do posicionamento fino com o desvio de 0,05 pol. da marcação principal para não ficar muito próxima da borda da placa. A Figura 30b mostra o resultado dessas alterações no reposicionamento das legendas.



(a) Tela para reposicionamento das legendas



(b) Resultado do reposicionamento da legenda

Figura 29: Ajuste da legenda de um componente

10.1.5 Configurações de Grid e DRC

O posicionamento dos elementos no PCB é discretizado por uma grade, ou grid na terminologia do DipTrace, que mantém as distâncias entre os elementos dentro de limites adequados para que não ocorra interferência elétrica e para que a fabricação seja possível. O menor valor permitido nesse curso é de 0.0125 polegadas (0,3175 mm) para X e Y. Deve-se notar que a máquina de prototipagem rápida trabalha com uma fresa de 0,2 mm e que o uso de um grid menor, com 0.00625 polegadas (0,15875 mm) permitiria uma distância mínima menor do que o diâmetro da ferramenta o que causaria problemas na fabricação.

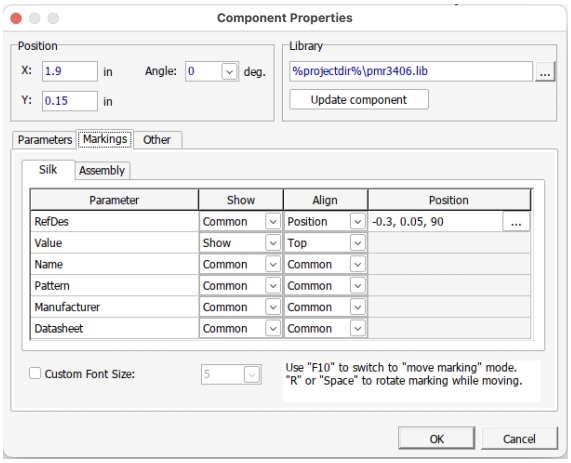
Para ajustar o grid nesse valor, entrar no menu "View > Grid Size" e escolher "0.0125 in". Deve-se verificar também que o mesmo valor é usado para Y através do menu "View > Y Grid Size" onde deve existir um "check mark" em "Identical with X".

Para auxiliar na verificação de que os elementos no PCB seguem as distâncias mínimas para que não ocorra interferência elétrica e para que seja possível a fabricação com a máquina de prototipagem rápida, o programa verifica as regras de projeto através do menu "Verification > Check Design Rules", ou através do botão na barra de ferramentas. Os parâmetros para a verificação podem ser definidos através da janela que é aberta pelo menu "Verification > Design Rules...", ou através do botão na barra de ferramentas. Os valores para os parâmetros devem ser configurados de acordo com a Figura 31. Deve-se observar que o menor valor utilizado como parâmetro é 0.008 polegadas (0,2032 mm) que é superior ao diâmetro mínimo da ferramenta (0,2 mm). A utilização de qualquer valor menor do que esse pode acarretar problemas na fabricação mesmo que o projeto passe pela verificação.

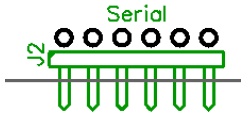
10.2 Procedimentos

Atividades em casa:

- Baixar e instalar o software DipTrace no seu próprio computador
- Fazer o tutorial do DipTrace



(a) Tela para reposicionamento de legendas com ajuste fino



(b) Resultado do reposicionamento das legendas do componente J2

Figura 30: Ajuste da legenda do componente J2

- Iniciar a criação do layout de sua placa impressa utilizando o posicionamento inicial mostrado na Figura 32
- Distribuir os componentes restantes dentro da área da placa **manualmente**
- Fazer o roteamento automático em face simples com jumper wires
- Retirar os jumper wires colocados em excesso pelo roteamento automático
- Completar o roteamento de forma manual
- Utilizar o copper pour conectado ao terra para completar os espaços do lado da solda

Na Figura 32 as dimensões da placa são aproximadamente 50 mm por 63,5 mm que correspondem a 2,0 pol. por 2,5 pol. Os 4 furos para fixação da placa possuem diâmetro interno de 0,094 pol. para parafusos M2 e diâmetro externo de 0,217 pol. para manter a isolação entre porcas e arruelas até as trilhas. O posicionamento dos headers J1 e J2, furos de fixação e dimensões da placa devem ser rigorosamente obedecidos para que a placa encaixe na base do robô móvel.

A placa deve ser identificada com as iniciais dos nomes dos componentes do grupo, turma e grupo da lista de chamada colocados dentro de um retângulo na camada TOP SILK com um tamanho de letras que não seja inferior ao utilizado nas marcações "RefDes". A posição do quadro de identificação não é rigorosa, este pode ser colocado num local onde não existam componentes nem jumpers.

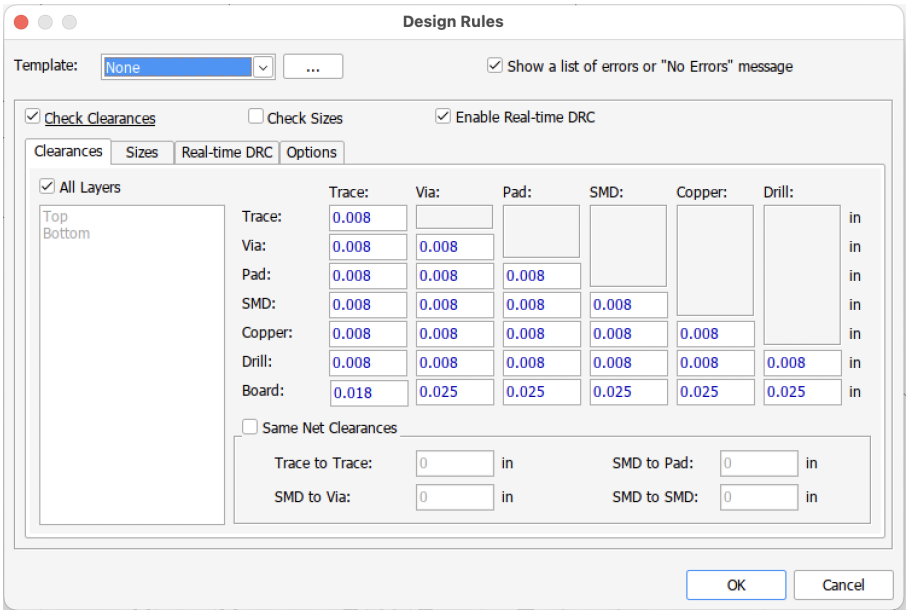


Figura 31: Janela para definição dos valores dos parâmetros para as regras de projeto

ATENÇÃO: A identificação da placa é obrigatória e deve conter as seguintes informações em duas linhas, dentro de um retângulo:

- Primeira linha: Iniciais dos nomes do componentes do grupo (ex. JOFJ ou JO&FJ)
- Segunda linha: Turma 1, 2, 3, 4, 5, 6 (ex. T4) e grupo de acordo com a lista de chamada (ex. G2), separados por hífen (-).

Trazer para a atividade 2 o layout mais completo possível. É aceitável para o início da Atividade 2 que o layout contenha alguns "Ratlines" e não possua "Copper Pour", mas deve conter todos os componentes posicionados, pelo menos numa primeira proposta, e deve ter sido roteada automaticamente com os parâmetros definidos conforme a seção 10.1.2. Recomenda-se que o aluno tenha realizado algum roteamento manual e leve suas dúvidas para a Atividade 2 de modo que o professor possa orientá-lo em como terminar seu layout.

10.3 Relatório

O relatório desta atividade consiste na impressão das faces de componentes e de solda da placa no estado que estiver na véspera da Atividade 2. O relatório deverá ser identificado com o nome de cada aluno associado às imagens impressas.

A impressão pode ser realizada no próprio software de PCB Layout através do menu "File > Preview..." que abrirá uma janela para visualização e controle do que será impresso. Nessa janela, deve-se imprimir os dois lados da placa, seguindo-se o seguinte procedimento:

- Escolher "Layers > Current: Top" e clicar no botão "Save...".

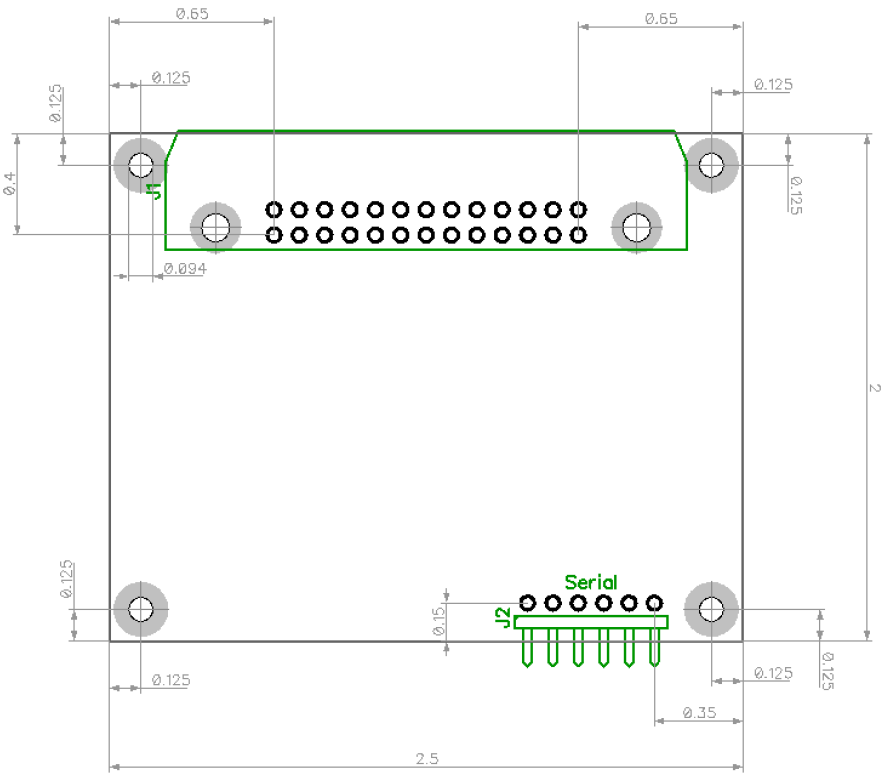


Figura 32: Posicionamento inicial a ser seguido, medidas em polegadas

- Na janela "Save Picture" escolher "Save As: Jpeg Image (*.jpg)", "Area: Sheet", "Resolution: 300 dpi" e clicar em Ok. Escolher o nome do arquivo a ser salvo.
- Repetir o processo para "Layers > Current: Bottom".

Capítulo 11

Atividade 2: Layout e Fabricação

Nesta atividade os alunos concluirão o layout da placa de circuito impresso que será fabricada, montada pelos próprios alunos e utilizada durante o curso e no PI-7. Como resultado desta atividade cada grupo de alunos gerará um conjunto de arquivos que deverão ser enviados por e-mail para um técnico do laboratório até uma data limite.

O técnico do laboratório receberá o conjunto de arquivos de todos os alunos de uma turma e fabricará as placas de circuito impresso que serão entregues para os alunos na próxima atividade para montagem.

Essa atividade tem por objetivos:

- Completar o layout do PCB iniciado na Atividade 1 e que será utilizada durante esse curso e no PI-7
- Gerar os arquivos Gerber e N/C Drill necessários para a fabricação da placa de circuito impresso
- Gerar imagens do layout realizado e visão 3-D da placa com componentes

11.1 Procedimentos

No dia do laboratório:

- Levar o arquivo de layout criado na Atividade 1 para verificação e avaliação pelo professor e para ser completado durante a aula de laboratório

Depois do dia do laboratório:

- Terminar o layout, caso não tenha sido possível durante a aula de laboratório
- Ter certeza de que o layout elaborado passa no teste de regras de projeto (DRC – Design Rules Check)
- Gerar o modelo 3-D da placa colocando os modelos 3-D corretos dos componentes e verificar a interferência entre os componentes, refazer caso necessário
- Gerar o conjunto de arquivos Gerber e NC Drill
- Enviar conjunto de arquivos compactados em .ZIP por email para macdp@usp.br seguindo a programação de entregas comunicada em aula

Os alunos deverão escolher somente um layout a ser fabricado por grupo.

ATENÇÃO: Enviar os arquivos Gerber e NC Drill em arquivo compactado (.ZIP) para macdp@usp.br seguindo a programação que será comunicada durante as aulas. A falta dessa entrega compromete as futuras atividades de laboratório causando prejuízo de nota para os alunos do grupo.

11.1.1 Teste de Regras de Projeto

É muito importante para que a placa projetada possa ser fabricada que o layout passe pelo teste de regras de projeto (DRC – Design Rules Check) com os valores padrão definidos pelo programa "PCB Layout". Passar pelo teste significa também que distâncias mínimas entre redes diferentes estão sendo obedecidas e não apresentam riscos de curto circuito.

Os teste pode ser feito pelo comando "Verification > Check Design Rules", pela tecla F9, ou pelo botão da barra de menu correspondente. Esse teste mostrará uma lista dos pontos que não obedecem as regras e indicará no desenho o ponto com um circulo vermelho. A posição das trilhas e vias deverá ser modificada até que o projeto passe pelo teste sem erros. Os parâmetros para o teste de verificação devem ter sido definidos conforme indicado na Figura 31, conforme descrito anteriormente.

11.1.2 Modelo 3-D do PCB

O DipTrace possui modelos 3-D da maioria dos componentes. Os modelos 3-D de componentes específicos utilizados no projeto estão disponíveis através do site da disciplina, devem ser baixados e usados para fazer o modelo 3-D do PCB. Alguns modelos são disponibilizados pelo DipTrace e foram duplicados no arquivo disponível no Moodle. Outros componentes cujo modelo 3-D não é fornecido pelo DipTrace mas estão no arquivo disponível do Moodle são os seguintes:

- J2 – Header de 6 pinos horizontal – arquivo header1x6h.wrl
- J3 – Header de 2 pinos vertical – arquivo header1x2.wrl
- Y1 – Oscilador de 20 MHz – arquivo osc_dip_8.wrl
- S2 – Chave táctil de 2 pinos[1] – arquivo tact_switch-2pin.wrl

Ao definir um dos modelos 3-D dos componentes fornecidos devem ser ajustados os deslocamentos em x, y e z, as rotações em x, y e z e o fator de escala em x, y e z que é de 0.39 para todos os eixos e para todos os componentes.

Depois de feito o modelo 3-D do PCB com todos os componentes corretos e corretamente posicionados, deve-se avaliar a interferência entre os componentes, isto é, se é possível montar um componente no PCB sem que o mesmo toque no outro. Caso seja detectada alguma interferência entre componentes os mesmos devem ser reposicionados e o layout refeito.

[1] Podem ser montados dois modelos diferentes de chave, uma com 4 pinos (S1) e outra com 2 pinos (S2). Somente uma delas será fornecida e montada na placa.

11.1.3 Arquivos Gerber e NC Drill

Os arquivos Gerber possuem comandos para o controle numérico da máquina de prototipagem rápida de circuitos impressos. Cada arquivo corresponde a uma camada da placa ou contém informações sobre a placa. No programa "PCB Layout" do DipTrace, o comando "File > Export > Gerber..." abre uma janela para controle de exportação dos arquivos Gerber.

Segue a explicação de cada um dos arquivos:

- Top Assembly – máscara de posicionamento de componentes do lado dos componentes[2], usado no caso de montagem de superfície, esse arquivo não conterá informações no nosso caso
- Top Silk – (*) máscara de silk screen do lado dos componentes que contém a identificação e contorno de todos os componentes a serem montados
- Top Mask – máscara de solda do lado dos componentes
- Top Paste – máscara para pasta de solda do lado dos componentes
- Top – trilhas, pads e vias do lado dos componentes
- Bottom – (*) trilhas, pads e vias do lado da solda
- Bottom Paste – máscara para pasta de solda do lado da solda
- Bottom Mask – máscara de solda do lado da solda
- Bottom Silk – máscara de silk screen do lado da solda
- Bottom Assembly – máscara de posicionamento dos componentes do lado da solda, usado no caso de montagem de superfície, esse arquivo não conterá informações no nosso caso
- Board Outline – (*) contorno da placa
- Board – área da placa
- Top Dimension – marcação das dimensões e distâncias marcadas pelo lado dos componentes
- Bottom Dimension– marcação das dimensões e distâncias marcadas pelo lado da solda

Ainda na mesma tela, clicar em "Export All...", após aparecerão 3 opções para exportação, escolher "ZIP Archive: Gerber + NC Drill". Isso criará um arquivo .ZIP num diretório a escolher com todos os arquivos necessários para a fabricação da PCB.

Somente os arquivos marcados com (*) acima, mais um arquivo com nome Through.drl (que contém informação sobre os furos – NC Drill), são utilizados para a fabricação da placa neste curso. Os outros não são utilizados, mas podem ser enviados no arquivo .ZIP. O técnico se encarregará de verificar e utilizar somente os arquivos necessários.

O email a ser enviado deve conter a identificação dos alunos através do nome e número USP e a Turma da qual fazem parte.

[2]Lado dos componentes e lado da solda é uma terminologia usada em placas com componentes do tipo *through hole* (furo passante) pois nesse caso os componentes ficam de um lado e são soldados pelo outro. No caso de montagem do tipo *surface mount* (montagem na superfície) utiliza-se a terminologia lado de cima de lado de baixo pois os componentes podem ser soldados em ambos os lados e a conexão entre os lados é feita somente por vias com furos metalizados.

ATENÇÃO: A identificação dos nomes dos alunos e turma é muito importante pois as placas serão fabricadas em lotes divididos por turma para estarem prontas na data da Atividade da turma correspondente. A falta dessa identificação pode resultar na não fabricação da placa a tempo.

11.1.4 Conferência no recebimento

O técnico conferirá os arquivos recebidos e verificará os pontos constantes da Tabela 10. Caso algum ponto esteja em desacordo, será retornado um e-mail para que o grupo de alunos resolva os pontos anotados pelo técnico e re-submeta os arquivos Gerber e NC Drill. Esse procedimento deve ser feito muito rapidamente pois são pontos que impedem a produção da placa e a sua montagem de maneira segura pelos alunos. Caso o técnico não receba um retorno dos alunos em tempo para a fabricação, a placa será excluída do lote e não será fabricada, sendo que os alunos serão prejudicados nas atividades de laboratório seguintes e haverá prejuízo de nota.

Tabela 10: Check List antes do envio dos arquivos Gerber e NC Drill para o técnico

Item	Ponto a ser verificado pelo técnico	Atendido
1	A placa está roteada somente numa face (face de solda)	☐
2	Os jumpers não se cruzam	☐
3	Os jumpers são somente verticais ou horizontais	☐
4	Os jumpers não se encostam e não se sobrepõe	☐
5	Os jumpers não passam por baixo dos componentes	☐
6	As vias dos jumpers possuem os diâmetros interno e externo conforme a especificação	☐
7	As trilhas são somente horizontais, verticais ou em 45°	☐
8	Existe copper pour do lado da solda	☐
9	Os Furos de montagem possuem os diâmetros interno e externo conforme a especificação	☐
10	Existe identificação dos alunos e turma na máscara de silk screen (Top Silk)	☐

Nota O atendimento de todos os pontos não garante o funcionamento correto da placa, garante somente que a placa possa ser fabricada e montada pelos alunos.

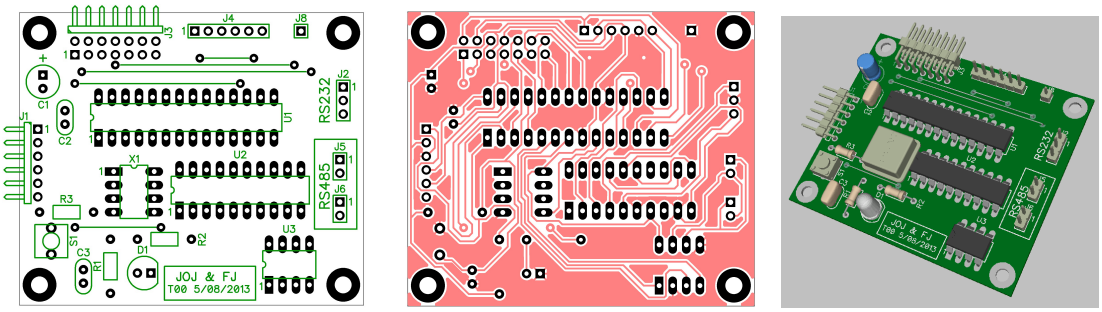
11.2 Relatório

O relatório, para cada aluno, deverá conter 3 figuras geradas a partir da impressão do layout executado, uma para a face dos componentes, uma para a face da solda e uma do modelo 3-D da placa. A figura do lado dos componentes deverá conter as dimensões da placa e indicações do posicionamento dos componentes de acordo com a Figura 32.

A Figura 33 mostra um exemplo de layout e modelo 3-D que corresponde às imagens que devem ser entregues no relatório. Esse exemplo só não mostra as dimensões e medidas de posicionamento dos componentes que devem ir no relatório.

Serão avaliados nesse relatório os seguintes pontos:

- dimensões placa de acordo com o especificado
- posicionamento dos componentes de acordo com a Figura 32
- existência de identificação dos alunos e turma
- roteamento completo, isto é, sem "Ratlines"
- atende ao DRC
- correto posicionamento dos componentes verificado no modelo 3-D



(a) Lado dos componentes, sem indicação de dimensões e medidas de posicionamento

(b) Lado da solda

(c) Modelo 3-D

Figura 33: Exemplo de layout e modelo 3-D

Capítulo 12

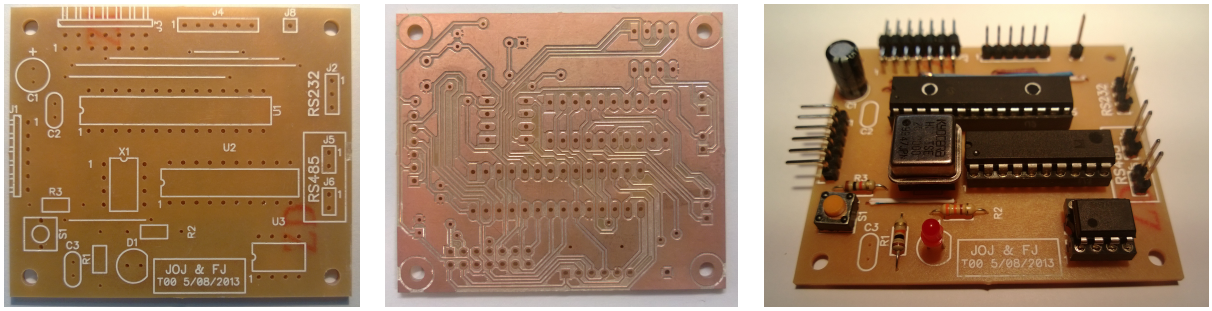
Atividade 3: Montagem e testes

Nessa atividade os alunos receberão a placa fabricada na máquina de prototipagem rápida para ser montada e testada. Será distribuído no início da aula de laboratório um envelope contendo todos os componentes necessários e o esquemático do circuito. Todos os componentes a serem montados possuem uma identificação no lado dos componentes na placa que corresponde a mesma identificação presente no diagrama esquemático do circuito. Com isso os alunos poderão identificar os componentes corretos para a montagem.

Os alunos que não estiverem familiarizados com o processo de soldagem deverão pedir ajuda para o professor ou para o seu assistente para as devidas orientações.

Após a montagem dos componentes na placa o circuito deverá ser testado. Para tanto é fornecido um projeto de MPLAB cuja execução permite o teste funcional dos diversos elementos do circuito. No caso de falha em algum dos testes deverá ser revisada a montagem dos componentes com atenção aos seus valores e as conexões entre a placa processadora, o computador e o robô móvel.

Como resultado dessa atividade os alunos terão a placa processadora completa, funcionando corretamente para ser utilizada nas próximas atividades e no PI-7. A Figura 34 mostra um exemplo de uma placa fabricada, (a) com as marcações no lado dos componentes, (b) as trilhas de cobre no lado da solda e (c) uma placa montada.



(a) Lado dos componentes **(b)** Lado da solda **(c)** Placa montada

Figura 34: Exemplo de placa fabricada e montada

12.1 Procedimentos

Os alunos receberão a placa fabricada no início do laboratório e um envelope com todos os componentes a serem montados na placa. O conteúdo do envelope deverá ser conferido com a lista de componentes no final desta apostila e no caso de falta de algum componente o professor deve ser avisado para que seja providenciado o componente.

A seção "Montagem do circuito" apresenta uma sequência de montagem que deve ser seguida para que a montagem seja feita de maneira mais fácil. Os alunos deverão soldar os componentes na placa de circuito impresso. Caso não possuam experiência anterior em soldagem, o professor prestará um auxílio inicial. Recomenda-se que os dois alunos de um grupo soldem alguns componentes para ganhar experiência prática.

Após a montagem a placa deverá ser testada segundo os detalhes apresentados na seção "Testes do circuito".

12.1.1 Montagem do circuito

A sequência de soldagem dos componentes visa facilitar a montagem da placa. A soldagem dos componentes de menor altura impede que eles se soltem da placa quando esta estiver com a face da solda para cima e apoiada sobre a bancada.

Assim, deve-se soldar os componentes na seguinte sequência:

1. jumpers
2. header horizontal de 6 pinos (J2)[1] [2]
3. resistores (R1, R2, R3, R4)
4. chave (S1 ou S2, dependendo do modelo da chave)
5. capacitor (C2)
6. soquetes[3](U1, Y1[4])
   ATENÇÃO: Devem ser soldados os soquetes e não os componentes na placa e também deve-se prestar atenção na direção do pino 1 dos soquetes para não montar invertidos.
7. header horizontal de 26 pinos com ejetor (J1)
8. LED (D1)
9. header vertical (J3)
10. capacitor (C1)

Depois disso os componentes U1 e Y1 devem ser inseridos nos soquetes correspondentes.

Ao término a placa estará pronta para os testes.

[1] Dependendo do modelo do J2, inverter a sequencia com os resistores, isto é, se J2 for mais alto do que os resistores.
[2] Soldar o lado com pinos mais curtos.
[3] Os componentes U1 e Y1 não são soldados na placa, eles são encaixados nos soquetes fornecidos.
[4] O componente Y1 possui somente 4 pinos e tem o formato de um DIP de 8 pinos, assim é utilizado um soquete DIP de 8 pinos dos quais podem ser soldados somente os pinos 1, 4, 5 e 8 que são utilizados pelo componente Y1.

12.1.2 Testes do circuito

É fornecido um projeto de MPLAB X para testes do circuito montado. Esse projeto está disponível no site da disciplina, com o nome HW_Test, e deve ser baixado e levado no dia da Atividade 3. Cada grupo deverá criar um diretório na Área de Trabalho do computador do laboratório e colocar o projeto baixado nesse diretório. Durante a atividade esse projeto é compilado e gravado na placa processadora montada pelos alunos. Após a realização dos trabalhos a pasta criada com o projeto deve ser apagada do computador.

Esse projeto testa os seguintes itens:

- comunicação SPI entre a placa processadora e o robô
- chave
- LED
- interrupção
- comunicação serial
- conversão A/D para leitura de sensor de proximidade
- leitura digital de sensor de linha
- acionamento PWM dos motores CC do robô

Antes do início dos testes os diversos módulos devem ser conectados segundo o diagrama da Figura 35. Depois disso, o projeto fornecido deve ser aberto no MPLAB X e compilado. Como resultado da compilação é gerado um arquivo .HEX no diretório HW_Test.X/dist/default/production/ que deve ser carregado na memória de programa do PIC utilizando-se um programa chamado bootloader.

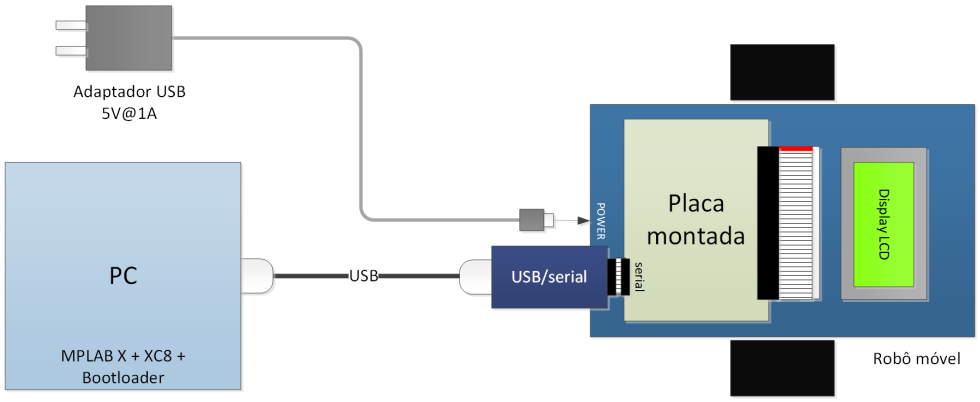


Figura 35: Conexões entre os módulos

No MPLAB X a compilação é feita clicando-se no botão com uma imagem de um martelo, item 1. da Figura 36. A compilação deverá ser bem sucedida e aparecerá uma mensagem em verde "BUILD SUCCESSFUL" na janela de saída do MPLAB X, item 2. da Figura 36. Caso não apareça essa mensagem chamar o professor para verificação do que está ocorrendo.

O bootloader serve para carregar os programas compilados no MPLAB X para o PIC através do canal serial. É composto por duas partes, a primeira é um aplicativo para Windows que corresponde ao executável com nome "Serial Bootloader AN1310", chamado somente de AN1310 daqui em diante. A

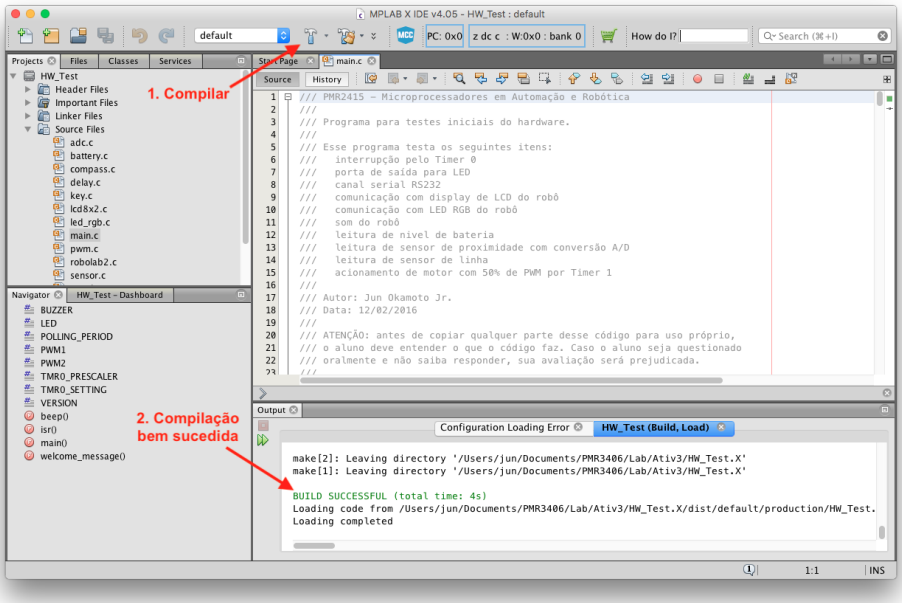


Figura 36: Interface com o usuário do MPLAB X

segunda parte é um pequeno programa residente na memória do PIC, chamado de firmware, previamente gravado, que se comunica com o AN1310 através do canal serial (pelo conversor USB/serial) e tem a função de receber um programa pelo canal serial e gravá-lo na memória de programa do PIC. Em conjunto essas duas partes carregam um programa em formato .HEX na memória de programa do PIC para ser depois executado.

Logo após iniciar o AN1310 deve-se configurar a porta serial como USB Serial no menu Program > Settings... e verificar que a taxa de transferência está em 115.200 bps para o bootloader e 19.200 bps para o console.

Para se entrar no modo de carga de programa, conectando-se o AN1310 com o firmware, deve-se clicar no botão  (Bootloader Mode) na barra de ferramentas do AN1310 para que seja estabelecida a comunicação entre as duas partes e essas possam trocar mensagens. Se a comunicação for estabelecida com sucesso aparecerá a mensagem "Bootloader Firmware v1.05 ..." no canto inferior esquerdo do AN1310. Após essa ação, caso a mensagem apresentada seja "Bootloader not found." deve-se utilizar o osciloscópio e verificar os seguintes sinais: TX, RX, CLOCK e MCLR. Pedir ajuda para o professor para realizar as medidas com o osciloscópio e saber o que observar em cada um dos sinais.

Depois disso deve-se se carregar o programa .HEX gerado pelo MPLAB X no AN1310. Isso é feito abrindo-se o arquivo gerado pelo MPLAB X, presente no diretório especificado anteriormente.

O programa carregado no AN1310 é transferido para o PIC pressionando-se o botão  (Write Device) da barra de ferramentas. No canto inferior esquerdo do AN1310 aparecerá a mensagem de qual endereço do PIC está sendo gravado e no final aparecerá a mensagem "Write Complete...".

Ao ser pressionado o botão  (Run Mode) na barra de ferramentas do AN1310 é liberado o RESET do PIC e o programa de teste será executado no microcontrolador. Nesse momento deve ser apresentada a mensagem "Teste Hw" na primeira linha do display LCD do robô, deve ser emitido um sinal sonoro pelo robô e o LED da placa processadora começará a piscar a uma vez por segundo indicando que a interrupção

está funcionando. O AN1310 entra em modo de terminal recebendo e enviando caracteres para o robô.

Cada vez que a chave da placa processadora for pressionada, a quantidade de vezes que a chave foi pressionada é enviada pelo canal serial, o LED RGB do robô muda de cor e é executado um teste que apresentará informações no display LCD do robô. Depois de passar por todos os testes o programa retorna ao seu estado inicial e os testes podem ser repetidos. Os testes realizados são os seguintes:

1. nível de bateria do robô
2. valor lido pelo conversor A/D do sensor de proximidade
3. valor lido pelos bits de porta digital do sensor de linha
4. valor lido da bússola do robô que deve ser "03d" se a bússola e a comunicação estiverem funcionando
5. acionamento dos motores por PWM para frente
6. acionamento dos motores por PWM para giro do robô para direita
7. acionamento dos motores por PWM para giro do robô para a esquerda

Caso a execução do programa não apresente os resultados descritos acima, a montagem da placa deve ser verificada procurando-se por curto-circuitos na solda, trilhas interrompidas ou em curto-circuito, soldas de baixa qualidade, componentes trocados, dentre outros.

Após a verificação, se os problemas persistirem, deve ser usado o osciloscópio para verificar alguns sinais importantes. O professor deve ser chamado para orientar sobre como fazer os testes.

No final da aula o professor deve ser chamado para verificar o estado final da montagem e testes.

O botão  (BREAK/Reset Mode) do AN1310 envia um caracter de BREAK pelo canal serial o que faz com que a execução de um programa no PIC pare imediatamente. Isso é necessário em algumas situações antes de se clicar no botão  para estabelecer a comunicação entre as duas partes do bootloader.

12.2 Relatório

No relatório devem ser entregues duas figuras contendo uma foto colorida da placa montada do lado dos componentes e outra do lado da solda, na melhor resolução possível. A qualidade da soldagem efetuada será avaliada para efeitos da nota do relatório pelas imagens apresentadas. Assim, caso as imagens não permitam a avaliação da montagem, haverá um desconto na nota devido a qualidade da imagem entregue.

Capítulo 13

Atividade 4: Leitura de sensor analógico

Nessa atividade é explorado o uso do conversor A/D do PIC para leitura de um sinal analógico que vem de um sensor de proximidade que detecta objetos na frente do robô. O robô possui montado na sua parte frontal um sensor Sharp GP2D120 que consegue medir distâncias de 4 cm até 30 cm, fornecendo um sinal analógico como saída. A Figura 37 mostra a relação entre a distância e a tensão de saída do sensor. Como essa é uma relação não linear, o objetivo dessa atividade é definir uma equação que faça a conversão de valor do conversor A/D para distância de forma linear.

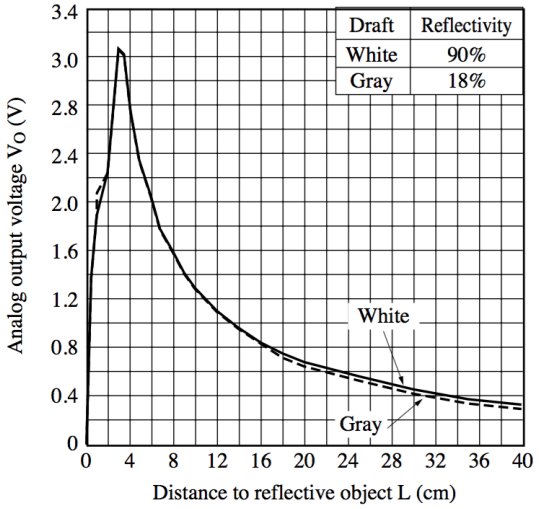


Figura 37: Relação entre tensão de saída e distância para o sensor Sharp GP2D120 (extraído do *data sheet* do componente)

O artigo *Linearizing Sharp Ranger Data*[1] do Blog da Acroname descreve um método de linearização dos dados de sensores Sharp que deve servir como base para determinar uma equação adequada para o sensor usado no laboratório. É importante lembrar 2 pontos:

- Essa equação deve ser implementada com aritmética inteira no programa a ser feito nesta atividade;
- O artigo apresenta a linearização de Volts para distância o que difere do valor numérico que se obtém da conversão A/D no laboratório, por isso os alunos devem obter seu próprio conjunto de medidas

[1] https://acroname.com/blog/linearizing-sharp-ranger-data

experimentais e a equação a ser determinada linearizará a saída do conversor A/D de 10-bits (um valor entre 0 e 1023) para distância.

O aluno deverá aprender a utilizar um canal de conversão A/D do PIC utilizando a biblioteca `adc.c` fornecida. Deverá efetuar processo de calibração de modo a determinar a relação entre a distância real e o valor lido pelo conversor A/D. Utilizar os dados levantados para definir uma equação de conversão e apresentar um valor de distância em mm no display LCD.

Para uso dos sensores é fornecida uma biblioteca `sensor.c` que contém as seguintes funções para uso do sensor de proximidade:

- `void sensor_power(char state)` – liga e desliga a alimentação dos sensores e do buzzer. Como os sensores consomem energia da bateria quando estão ligados, deve-se ligar a alimentação dos sensores antes do seu uso e desligá-la depois. No caso dos programas desenvolvidos nesse laboratório não é necessário desligar os sensores, basta ligá-los uma vez no início do programa. O buzzer do robô necessita que a alimentação dos sensores esteja ligada para funcionar. O sensor de proximidade demora cerca de 55 ms para poder ser usado depois que a alimentação é ligada. Esse tempo é contabilizado na própria função na mudança para o estado ligado. Assim no caso de uso do sensor de proximidade em um loop de medição deve-se ligar a alimentação fora do loop. O parâmetro `state` pode ser 0 – desligado ou 1 – ligado. Podem ser usados `ON` e `OFF`, definidos no arquivo `always.h` como 1 e 0, respectivamente, no lugar de valores numéricos para este parâmetro.

- `void sensor_init(void)` – inicializa o sensor de proximidade, o sensor de linha e o buzzer configurando o bit da porta usada para o sensor de proximidade como entrada de sinal analógico, os bits da porta usada para o sensor de linha e para o buzzer como digital. Esta função chama `adc_init(0)`, da biblioteca `adc.c`, para configurar o canal analógico 0 utilizado para o sensor de proximidade.

- `int sensorNear_read(void)` – retorna uma leitura do sensor de proximidade. Esta função chama `adc_read(0)`, da biblioteca `adc.c`, para ler o canal analógico 0.

O aluno deverá também aprender a criar o seu próprio projeto no MPLAB IDE. Não deve ser usado o mesmo projeto da atividade anterior, a cada nova atividade deve ser criado um novo projeto como novo nome. Segue o procedimento passo a passo de como criar um novo projeto:

1. Abrir o MPLAB_IDE e escolher "File" > "New Project ..."
2. Na tela "1. Choose Project", escolher a Categoria "Microchip Embedded", o Projeto "Standalone Project" e depois "Next > "
3. Na próxima tela, "2. Select Device", escolher a Família "Mid-Range 8-bit MCUs (PIC10/12/16/MCP)", o dispositivo "PIC16F886" e depois "Next > "
4. Na próxima tela, "3. Select Tool", escolher a ferramenta "Simulator" e depois "Next >"
5. Na próxima tela, "6. Select Compiler", escolher o compilador "XC8 (vx.xx)..." e depois "Next > "
6. Na próxima tela, "7. Select Project Name and Folder", escrever um nome para o seu projeto[2], escolher um diretório para sua localização e depois "Finish"

[2]Utilize um nome diferente de HW_Test

7. No Browser de Projetos, clicar com o botão da direita do mouse sobre "Header Files" e escolher "Add Existing Item...". Escolher "Store path as: Relative" e marcar o check box "Copy". Localizar o diretório "RL2" e selecionar todos os arquivos com extensão ".h". Terminar com um click em "Select"

8. Repetir o mesmo para o item "Source Files" para adicionar todos os arquivos ".c" do diretório "RL2" no seu projeto.

9. Clicar com o botão da direita novamente sobre "Source Files" e escolher "New" > "main.c..."

10. Na tela "Name and Location", alterar o nome "newmain" para "main" somente. Manter a Extensão em "c" e o nome do projeto e o local de criação nos mesmos definidos no item 6.

11. No arquivo "main.c" recém criado, você poderá adicionar os trechos de código do "main.c" do projeto "HW_Test" que for utilizar no seu projeto.

Após a criação do novo projeto, é possível compilá-lo clicando-se no botão  ou no botão  . O primeiro compila somente as alterações enquanto que o segundo apaga todos os arquivos temporários e compila tudo desde o início. Assim, a primeira operação é mais rápida que a segunda e é a que é normalmente utilizada. A segunda operação, compilação após limpeza, deve ser utilizada se o projeto foi movido de uma diretório para outro pois os caminhos para os arquivos foram alterado e devem ser refeitos desde o início.

A partir dessa atividade a mensagem inicial do seu programa deve apresentar na 1ª linha o número da atividade e na 2ª linha a identificação do grupo.

> **ATENÇÃO:** A partir dessa atividade a mensagem inicial (`welcome_message()`) do seu programa deve apresentar na 1ª linha o número da atividade e na 2ª linha a identificação do grupo.

13.1 Programas

Essa atividade requer o desenvolvimento de dois programas independentes, um para a calibração e outro para apresentação dos dados.

1. Calibração

   O programa para calibração deve ficar em loop e ao ser pressionada a chave deve fazer 10 leituras do conversor A/D, tirar a média aritmética e apresentar diretamente o valor da média das leituras do conversor A/D no display LCD. Devem ser efetuadas 4 leituras por segundo. Deve ser acionado um beep de 0,3 segundos ao ser apresentado novo valor display LCD.

   Esse programa deve utilizar uma interrupção periódica do Timer 0 a cada 5 ms aproximadamente. Essa mesma interrupção pode ser usada para atender o intervalo de tempo requerido[3] para efetuar as leituras do conversor A/D. A temporização do beep e da apresentação dos dados no display LCD devem ser feitas com funções da biblioteca `delay.c` dentro do próprio loop no programa principal.

[3] Deve ser alterado o valor da contagem final do contador de vezes que a interrupção é chamada.

2. Apresentação de dados

   O programa de apresentação de dados fica em loop contínuo realizando leituras do conversor A/D e atualizando o display LCD com o valor da distância em mm calculada pela equação levantada. O display deve ser atualizado a cada 0.3 segundos aproximadamente. A unidade do valor da distância deve ser apresentada no display LCD.

   A temporização da apresentação dos dados no display LCD deve ser feita com funções da biblioteca `delay.c` dentro do próprio loop no programa principal. Não deve ser usada interrupção para marcar tempo, um `delay_big_ms(300)` é suficiente no final do loop infinito do programa principal.

13.2 Procedimentos

1. Estudar o Capítulo 3 sobre conversão analógico/digital
2. Estudar o código da biblioteca `adc.c`
3. Fazer os programas de calibração e apresentação de dados
4. Utilizando o programa de calibração obter medidas de distância do sensor em cerca de 8 a 10 posições distintas e aproximadamente equiespaçadas entre 40 mm e 400 mm. Veja que cada medida é a média de 10 leituras do conversor A/D.
5. Utilizar os pontos obtidos para determinar uma equação que converta leitura do conversor A/D para milímetros
6. Ajustar o programa de apresentação de dados para utilizar a equação obtida no item anterior
7. Executar o programa de apresentação de dados e conferir a distância apresentada em relação a um objeto com uma régua

13.3 Relatório

No relatório deverão estar identificados os membros do grupo, turma, disciplina e atividade e deverão constar os seguintes itens:

1. Cálculos do Timer 0 para o programa de calibração para realizar 4 medidas por segundo.
2. Explicação sobre como inicializar e utilizar um canal de conversão A/D do PIC com base no estudo do *data sheet* do PIC 16F886 e do código da biblioteca `adc.c`. Mostre e explique todos os bits que devem ser programados para tanto.
3. Gráfico com os pontos levantados experimentalmente com o programa de calibração.
4. Apresentação da equação de conversão de valor do A/D para mm para o sensor proximidade Sharp.
5. Código fonte com comentários explicando o funcionamento do programa.

Capítulo 14

Atividade 5: Comunicação serial

Nesta atividade será implementado um programa de comunicação serial com outro robô e sinais de comunicação serial deverão ser analisados com o osciloscópio.

O programa deverá ter o seguinte comportamento:

1. Ao iniciar o programa deverá ser apresentado no display LCD, na primeira linha, a identificação da Turma e Grupo por 4 segundos e deverá ser emitido um sinal sonoro. Passado o tempo especificado o display deve ser limpo.

2. O cursor do display LCD deve ser posicionado na primeira posição da linha 1 e deverá ser mostrado um caracter que vai mudando ao longo do tempo na sequencia 0 a 9 e A a Z, em loop infinito. A interrupção do Timer 0 serve como base de tempo para o intervalo de tempo que cada caracter é apresentado no display LCD. O ajuste do tempo deve ser feito por um parâmetro no programa que determina quantas interrupções do Timer 0 devem ocorrer antes da troca do caracter.

3. Ao pressionar o botão, o caracter que está sendo apresentado no display é congelado no display e é transmitido pelo canal serial para o outro robô. O cursor do display deve avançar para a próxima posição que passará a apresentar novamente os caracteres mudando na sequência.

4. O caracter recebido pelo canal serial é mostrado na segunda linha do display a partir da primeira posição e assim sucessivamente, cada caracter recebido é apresentado na posição seguinte até o limite da linha.

Para o usuário, todas as operações devem parecer que acontecem em paralelo, ou seja, enquanto os caracteres são apresentados na primeira linha o botão pode ser pressionado e um caracter recebido deve ser imediatamente apresentado na segunda linha. Esse efeito é obtido pela estrutura do programa com um loop principal não blocante, utilizando-se a leitura da chave conforme dado na Atividade 3 e a recepção pelo canal serial com a função `chkchr()` da biblioteca fornecida `serial.c`. Essa função retira do buffer de recepção e retorna um caracter, se existir, ou retorna 255, se o buffer estiver vazio. Assim, a utilização dessa função uma única vez no loop de controle garante que somente uma mudança de estado é detectada.

14.1 Análise dos sinais com osciloscópio

Durante a execução do programa, cada grupo deve conectar o osciloscópio na sua placa e observar 2 pontos do circuito simultaneamente:

- Canal 1 (CH1) do osciloscópio: Sinal de RX do PIC no pino 18 de U1 ou pino 3 do conector J2
- Canal 2 (CH2) do osciloscópio: Sinal de RX do cabo de conexão entre os dois robôs no pino 1 do conector J2 da interface conversora de serial (0–5V) para RS232

As imagens das formas de onda devem ser salvas em *pen drive* para serem anexadas no relatório. Os tempos e tensões medidos devem ser anotados nas figuras de modo a demonstrar como foram feitas as medidas e que os alunos observaram e compreenderam os níveis de tensão e os tempos envolvidos nas medidas. Os alunos deverão ajustar os dois canis verticais do osciloscópio com o CH1 = 2V/div e o CH2 = 5/V/div e devem ajustar a posição vertical dos traços de maneira a permitir visualização clara, sem sobreposição, dos dois sinais ao mesmo tempo na tela. A base de tempo do canal horizontal deve ser ajustada de maneira a permitir que sejam visualizados todos os bits recebidos, incluindo o start-bit e pelo menos 1 stop-bit.

ATENÇÃO: não deve ser usado o AUTO SET do osciloscópio. Chame o professor para auxiliá-lo caso tenha dificuldade no ajuste adequado do osciloscópio.

Com base no tempo de um bit medido com o osciloscópio o sinal deverá ser analisado mostrando a que caracter ASCII a sequência de bits recebida corresponde. Deve ser verificado que o tempo de um bit medido corresponde à taxa de Baud programada no PIC.

14.2 Procedimentos

1. Calcular os parâmetros do Timer 0 para gerar interrupções a cada 50 ms que servirá como base de tempo para apresentação dos caracteres na linha 1 do display.
2. Calcular os parâmetros para operação da USART em 19.200 bps
3. Antes do dia da atividade de laboratório, criar um programa de acordo com a descrição dada, utilizando as bibliotecas `lcd.c`, `key.c` e `serial.c` que foram fornecidas na Atividade 3.
4. Depurar o programa no dia do laboratório utilizando a placa processadora do grupo e os dispositivos do laboratório.
5. A conexão entre os dois robôs e os osciloscópios deve ser feita de acordo com o diagrama apresentado na Figura 38. Pedir as interfaces RS232 e cabos para o professor. O JUMPER amarrado à interface deve ser conectado no J3 da placa processadora para alimentação da interface, conforme mostrado na Figura 39.
6. Tampar o display LCD, pedir para o outro Grupo transmitir um caracter e fazer as medidas dos sinais de RX do serial do PIC (pino 18 do U1) e do cabo de RS232 (pino 1 do J2 da interface, vide Figura 39) com o osciloscópio. Capturar a imagem da tela do osciloscópio e gravar num *pen drive* USB[1] para ser anexado no relatório. A Figura 40 mostra um exemplo de tela capturada. Decodificar os bits observados e dizer a qual caracter ASCII corresponde. Destampar o display e conferir o resultado. Refazer, caso errado.

[1]Caso não possua *pen drive* será aceita foto da tela desde que bem legível.

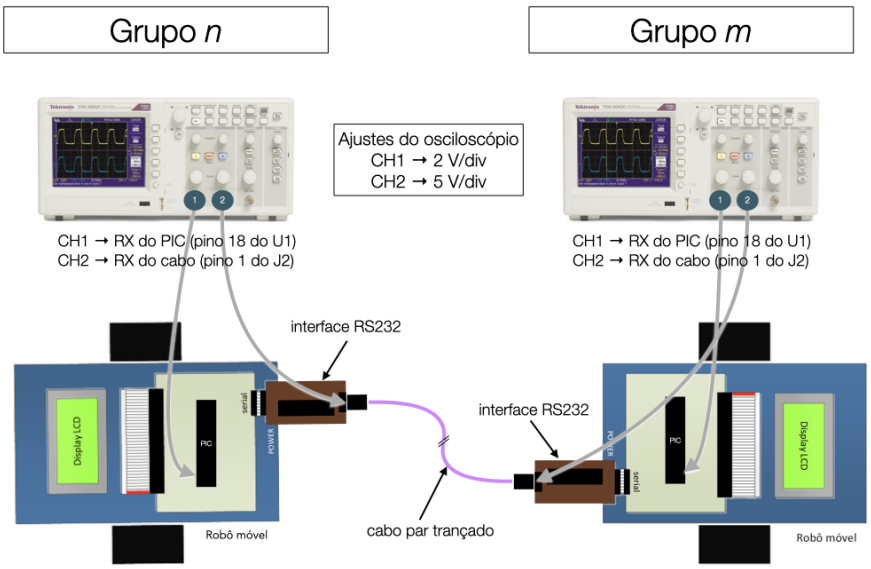


Figura 38: Esquema de conexão entre robôs e osciloscópios.

> **ATENÇÃO:** Para se visualizar corretamente os sinais no osciloscópio o sistema de disparo (*trigger*) deve ser ajustado corretamente com: tipo de disparo por BORDA; disparo pelo CH1; borda de Descida; modo Normal; acoplamento DC; nível de disparo por volta de 1V;

7. Durante a medição com osciloscópio, mostrar para o professor o procedimento de identificação do caracter recebido e o funcionamento do programa.

14.3 Relatório

No relatório deverão estar identificados os membros do grupo, turma, disciplina e atividade e deverão constar os seguintes itens:

1. Cálculos e programação dos bits dos registradores para configurar o Timer 0 para gerar uma interrupção periódica a cada 50 ms.
2. Cálculos e programação dos bits dos registradores do PIC para configurar a USART com taxa de Baud de 19.200 bps com palavra de 8-bits e sem paridade.
3. Imagem capturada durante atividade das formas de ondas dos 2 pontos do circuito indicados, explicando que caracter é que aparece nas formas de onda, tempos e tensões dos sinais. De acordo com as escalas vertical e horizontal do osciloscópio, marcar nas imagens os valores das tensões e os tempos de cada bit de maneira a mostrar que o sinal observado corresponde ao caracter ASCII.
4. Código fonte com comentários explicando o funcionamento do programa.

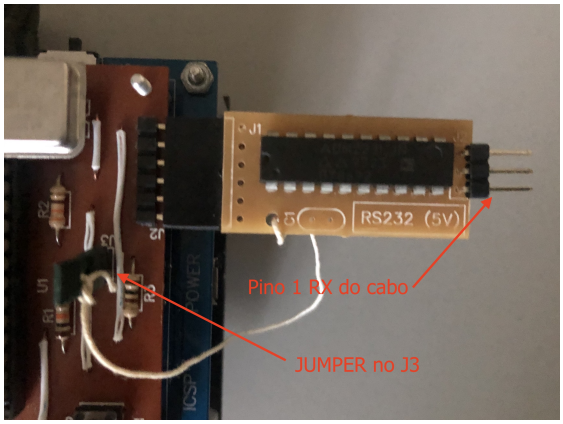


Figura 39: Conexão do JUMPER de alimentação da interface RS232 e identificação do pino 1 do cabo.

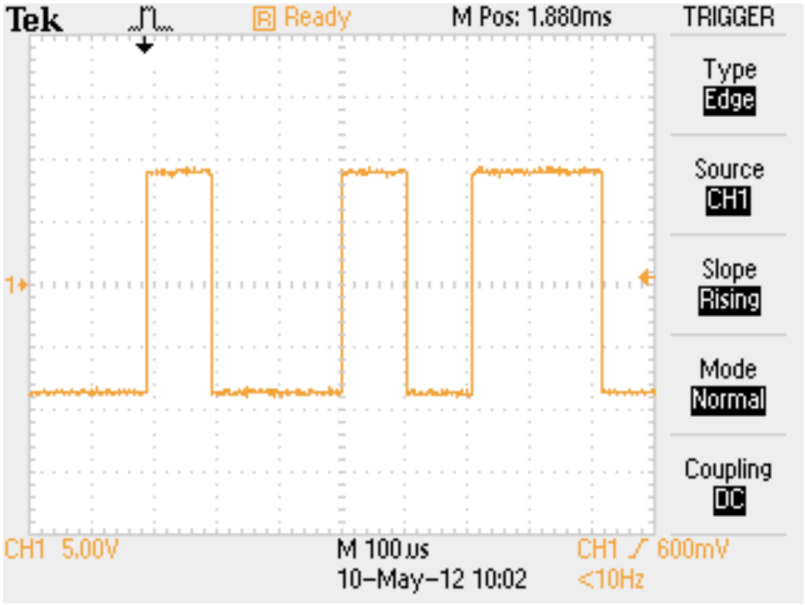


Figura 40: Exemplo de tela de osciloscópio capturada, mostrando sinal do canal 1 (CH1) e menu de disparo (TRIGGER) à direita.

Além da avaliação dos pontos acima, serão levados também em consideração para a nota:

- Estrutura do programa que deve seguir a proposta
- Lógica utilizada na solução do problema
- Relevância e adequação dos comentários inseridos no código do programa
- Trechos de código sem relação com a atividade prejudicarão a nota

Capítulo 15

Atividade 6: Acionamento de motor CC, leitura de encoders e estimação de velocidade

Nesta atividade a velocidade dos motores CC será ajustada de acordo com o valor do sensor de proximidade. Se não for detectado nenhum objeto na frente do robô a velocidade pode ser alta[1]. Se um objeto for detectado na frente do robô a velocidade deve ser reduzida de acordo com a proximidade do objeto até parar completamente se o objeto estiver muito próximo. O display LCD do robô deverá apresentar o valor de distância (em mm) medido com o sensor de proximidade utilizando a equação determinada na Atividade 4 e os valores das velocidades (em mm/s) de cada motor a serem determinados com os encoders nessa atividade.

Os motores são acionados pelos sinais PWMs gerados pela placa processadora e a velocidade de cada motor deve ser estimada pela diferença de leitura do encoder num determinado intervalo de tempo.

São utilizados os dois canais de PWM do PIC 16F886, o canal 1 controla o motor esquerdo e o canal 2 controla o motor direito. Ambos devem ser programados para operar na mesma frequência sendo que a largura do pulso controla a velocidade dos motores. Cada motor possui ainda um bit para controle de direção.

Para o cálculo da velocidade em mm/s, considere o diâmetro da roda como sendo 42 mm. Apresente no display LCD somente valores inteiros em mm/s, não devem ser mostrados décimos de milímetro que está além da precisão de posicionamento do robô móvel.

15.1 Procedimentos

1. Estudar os Capítulos 7 sobre PWM e 8 sobre encoder da apostila.
2. Estudar o item 11.5 do *data sheet* do PIC16F886
3. É fornecida uma biblioteca `pwm.c` que contém as funções para controle do PWM do PIC. No programa de testes foi utilizado o Timer 1 para gerar um sinal de 50% de PWM. Essa parte deve ser

[1]Velocidade alta não significa velocidade máxima. Não é possível partir o robô na velocidade máxima, a bateria não consegue suprir o pico de corrente necessário para tanto e o robô resetará. Utilize no máximo *duty cycle* de 60% do PWM como velocidade máxima.

reescrita nesta atividade, substituindo-se o uso do Timer 1 pelos 2 canais de PWM do PIC. Devem ser re-escritas as funções `pwm_init()` e `pwm_set()` segundo as especificações abaixo:

- `void pwm_init(void)` – inicializa os dois canais de PWM para operarem na frequência de 19,53 kHz[2]. Para tanto deve ser seguido o procedimento indicado no item 6.1.1 desta apostila que é baseado no item 11.5.7 do *data sheet* do PIC16F886.
- `void pwm_set(int channel, int duty_cycle)` – altera a porcentagem do PWM dado por `duty_cycle` para o canal de PWM dado por `channel`. O valor do `duty_cycle` corresponde a porcentagem de PWM multiplicado por 10, por exemplo, para 50% de PWM, o valor de `duty_cycle` é 500. Os valores que `duty_cycle` pode assumir devem ser limitados a valores entre 0 e 1000. Os valores possíveis para o parâmetro `channel` são 1 (motor esquerdo) ou 2 (motor direito).

A porcentagem de PWM para a frequência adotada possui uma resolução de 10-bits, ou seja o valor numérico do `duty_cycle` pode variar de 0 a 1023. Por simplificação pode-se adotar uma correspondência direta entre o valor numérico do parâmetro `duty_cycle` e a porcentagem de PWM multiplicada por 10. Assim, o parâmetro `duty_cycle` igual a 1000 corresponderá a aproximadamente a 100% de PWM. As consequências dessa simplificação são duas:

(a) Os motores atingirão uma velocidade máxima um pouco menor do que a velocidade máxima possível.

(b) Pode-se desprezar a programação dos dois bits menos significativos `CCPxCON<5:4>` (onde `x` = 1 ou 2, dependendo do canal do PWM) que podem ser mantidos em zero, permitindo que sejam programados somente os 8-bits de `CCPRxL` (onde `x` = 1 ou 2, dependendo do canal do PWM).

A função `void pwm_direction(int dir)` altera a direção de giro dos motores, é dada e não precisa ser modificada. O parâmetro `dir` pode assumir valores de 0 a 3, onde 0 significa mover para frente, 1 para trás, 2 girar para a esquerda e 3 girar para a direita.

4. Implementar a leitura dos encoders por Interrupt-On-Change do Port B como uma máquina-de-estados conforme descrito no Capítulo 8 desta apostila. A matriz de transição de estados deverá ser armazenada em EEPROM. Para tanto devem ser usadas as seguintes funções:

   - `__EEPROM_DATA()` – função macro para armazenar dados na EEPROM. Vide seção 5.4.5.2 do manual do XC8 (página 111 da versão 50002737E).
   - `EEPROM_READ()` – função macro para leitura dos dados da EEPROM. Vide seção EEPROM READ (MACRO) do manual do XC8. Alternativamente pode ser utilizada a função de biblioteca `eeprom_read()`.

   Deverá ser feita uma conversão de índice (de 2 dimensões) da matriz de transição de estados para o endereço (1 dimensão) da EEPROM.

   Na EEPROM só são armazenados bytes, 1 byte para cada endereço, sendo que um elemento da matriz de transição de estados deve ocupar somente 1 endereço (ou seja 1 byte) na EEPROM. No programa esse byte deve ser armazenado numa variável do tipo `signed char` (8-bits com sinal) para poder ser usado na matriz de transição de estados.

[2] O motivo para escolha desta frequência está no item 11.5.3 do *data sheet* do PIC16F886, vide Tabela 11-3.

Deve-se ler os bits das fases A e B dos dois encoders junto numa única leitura da porta. Isso garante que o estado correspondente ao valor dos bits reflete um único instante no tempo.

Para a contagem dos pulsos do encoder deverão ser definidas duas variáveis globais de 16-bits, com sinal, que armazenarão a contagem dos pulsos de encoder de cada motor. O movimento para frente de cada motor deverá incrementar a contagem, enquanto que o movimento para trás deverá decrementar a contagem.

Para auxiliar na depuração do programa o valor das contagens armazenadas devem ser mostradas no display LCD dentro do loop infinito no programa principal.
Deve-se confirmar que para cada volta da roda a contagem é de 48 pulsos. Isso pode ser feito deixando os PWMs em zero e girando cada roda com a mão 360°. A contagem mostrada no LCD deve atingir 48 e depois voltar para zero de a roda for girada no sentido contrário em 360°.

5. Implementar a medição de velocidade de cada roda fazendo a diferença de contagem de pulsos do encoder num período de 100 ms. Esse cálculo deverá ser feito no loop infinito do programa principal e os valores devem ser mostrados no display LCD. A base de tempo de 100ms deve ser gerada pela interrupção do Timer 0. Esse tempo não deve ser definido dentro do loop infinito com funções de delay.

   Observar com o osciloscópio a forma de onda de uma das fases A ou B dos encoders e armazenar num *pen drive*[3] para anexar no relatório. Anotar a qual % de PWM e velocidade mostrada no LCD a forma de onda observada corresponde. Determinar a frequência da forma de onda de uma das fases e com base nisso determinar a velocidade da roda. Considere que uma volta da roda são 12 pulsos de uma das fases. Repetir esse procedimento para 5 velocidades diferentes para montar um gráfico no relatório que relacione porcentagem de PWM com as velocidades em mm/s calculada com base na forma de onda observada do encoder e apresentada no display LCD.

6. Implementar a variação de velocidade de acordo com a leitura do sensor de proximidade na frente do robô, onde objetos distantes ou fora da faixa de detecção do sensor permitem velocidades altas. Na medida que o robô se aproxima de um objeto a velocidade vai sendo reduzida até chegar a zero quando for detectada a eminência de uma colisão (cerca de 4 cm). Os valores da leitura do sensor em mm e das velocidades de cada roda em mm/s devem ser mostradas no display LCD. A leitura do sensor de proximidade e variação de velocidade são feitas no loop infinito do programa principal.

ATENÇÃO: Todos os itens desse procedimento devem ser realizados com o robô sobre a base de borracha na bancada. O robô não deverá andar no chão nessa atividade.

15.2 Relatório

No relatório deverão estar identificados os membros do grupo, turma, disciplina e atividade e deverão constar os seguintes itens:

1. Cálculos de programação do Timer 0 para interrupção periódica utilizada na estimação da velocidade.

2. Cálculos do período e da resolução do PWM do PIC para a frequência pedida.

[3] Caso não possua *pen drive* será aceita foto da tela desde que bem legível.

3. Sequencia a ser programada para inicialização do PWM e alteração do *duty cycle* do PWM com explicação sobre os bits programados.

4. Com os dados obtidos no procedimento experimental, fazer um gráfico que relacione porcentagem de PWM com a velocidade do motor em mm/s e achar uma equação que converta % de PWM em mm/s. Comparar o valor da velocidade calculada pela medida do período do sinal do encoder com o valor de velocidade calculado pelo seu programa que é mostrado no display LCD.

5. Código fonte com comentários explicando o funcionamento do programa. Incluir somente os programas que foram escritos para essa atividade.

Capítulo 16

Atividade 7: Tarefa Robótica Autônoma

Nessa atividade são utilizados os sensores digital de linha e analógico de proximidade para a execução de uma tarefa robótica autônoma. Com esses sensores é possível seguir uma linha no chão feita com fita isolante preta e evitar colisão com objetos à frente do robô.

O sensor de linha é composto por três sub-conjuntos de sensores dispostos em linha na frente do robô. Cada sub-conjunto é espaçado de aproximadamente 9,5 mm do outro, o que é o suficiente para detectar uma faixa feita com fita isolante de 19 mm no chão. Esse sub-conjunto é composto por um emissor de luz e um receptor. Um sinal de luz infravermelho é emitido e refletido pelo chão. Se o sinal de luz que retorna possui baixa intensidade, o sensor retorna valor 1, em caso contrário, retorna valor 0. Ou seja, se o sensor estiver sobre a cor preta da fita isolante retornará o valor 1, se estiver fora da fita isolante, retornará o valor 0.

O sensor analógico de proximidade produz um nível de tensão que é lido por um canal de conversão A/D no PIC para detectar a proximidade de um objeto entre 4 cm e 30 cm na frente do robô. Esse sensor foi estudado e ensaiado no Capítulo 13.

A tarefa de seguir uma linha no chão consiste em manter o robô sobre a linha enquanto se desloca pelo chão. A velocidade de cada roda deve ser ajustada com base na leitura dos 3-bits do sensor de linha de modo a manter a linha alinhada com o sensor do centro. O sensor de proximidade deve ser utilizado para detectar outro robô se movendo na frente sobre a mesma linha e a velocidade do robô deve ser ajustada de modo a evitar colisão com o robô da frente. Caso não seja detectada nenhuma linha no chão, o robô deve iniciar um movimento circular até detectar a linha novamente.

Seguem alguns requisitos que devem ser seguidos na implementação da tarefa:

- A chave deve ser pressionada para que o robô inicie e pare a tarefa
- O LED RGB deve mudar de cor de acordo com a direção que o robô está se movendo. Por exemplo: verde, se estiver indo para frente; azul, se estiver girando para a esquerda; magenta, se estiver girando para a direita; etc.
- O LED RGB deverá ser usado também para sinalizar obstáculo encontrado à frente do robô, mostrando a cor vermelha.

Para a utilização do sensor de linha deve-se primeiro inicializar o sensor chamando a função `void sensor_init(void)`,

depois ligar os sensores com a função `void sensor_power(char state)`. A leitura do sensor de linha é feita com a seguinte função:

- `int sensorLine_read(void)` – essa função retorna o valor dos 3-bits do sensor de linha como a posição menos significativa da variável de retorno. Assim, a variável de retorno conterá algum valor entre 0 e 7 dependendo do que for detectado pelo sensor.

Para uso do LED RGB a interface SPI deve estar inicializada. Deve ser incluído o header `led_rgb.h`. A seguinte função deve ser usada para controlar a cor do LED RGB:

- `void led_rgb_set_color(char led_color)` – define a cor do LED RGB, inclusive apagado, com um código entre 0 e 7. Ver o header `led_rgb.h` para as definições de cores.

Em relação a leitura do sensor de proximidade, o valor recebido da função `int sensorNear_read(void)` pode ser usado diretamente para a alteração da velocidade do robô, não sendo necessário a conversão desse valor para milímetros. Essa abordagem evita cálculos em ponto flutuante e melhora o desempenho. Dependendo da implementação essa conversão pode demorar muito tempo e pode comprometer o funcionamento em tempo real. O valor do A/D pode ser utilizado diretamente para o controle de velocidade da seguinte maneira: valor do A/D menor, aumenta a velocidade, valor do A/D maior, reduz a velocidade e se o valor do A/D passar de um limitar superior, pára o robô completamente.

O display LCD pode ser usado para mostrar mensagens que auxiliem no desenvolvimento e testes do algoritmo. Só não devem ser enviadas informações muito longas para o display LCD pois pode atrapalhar o desempenho em tempo real do algoritmo devido ao fato de que escritas no display LCD são demoradas.

16.1 Relatório

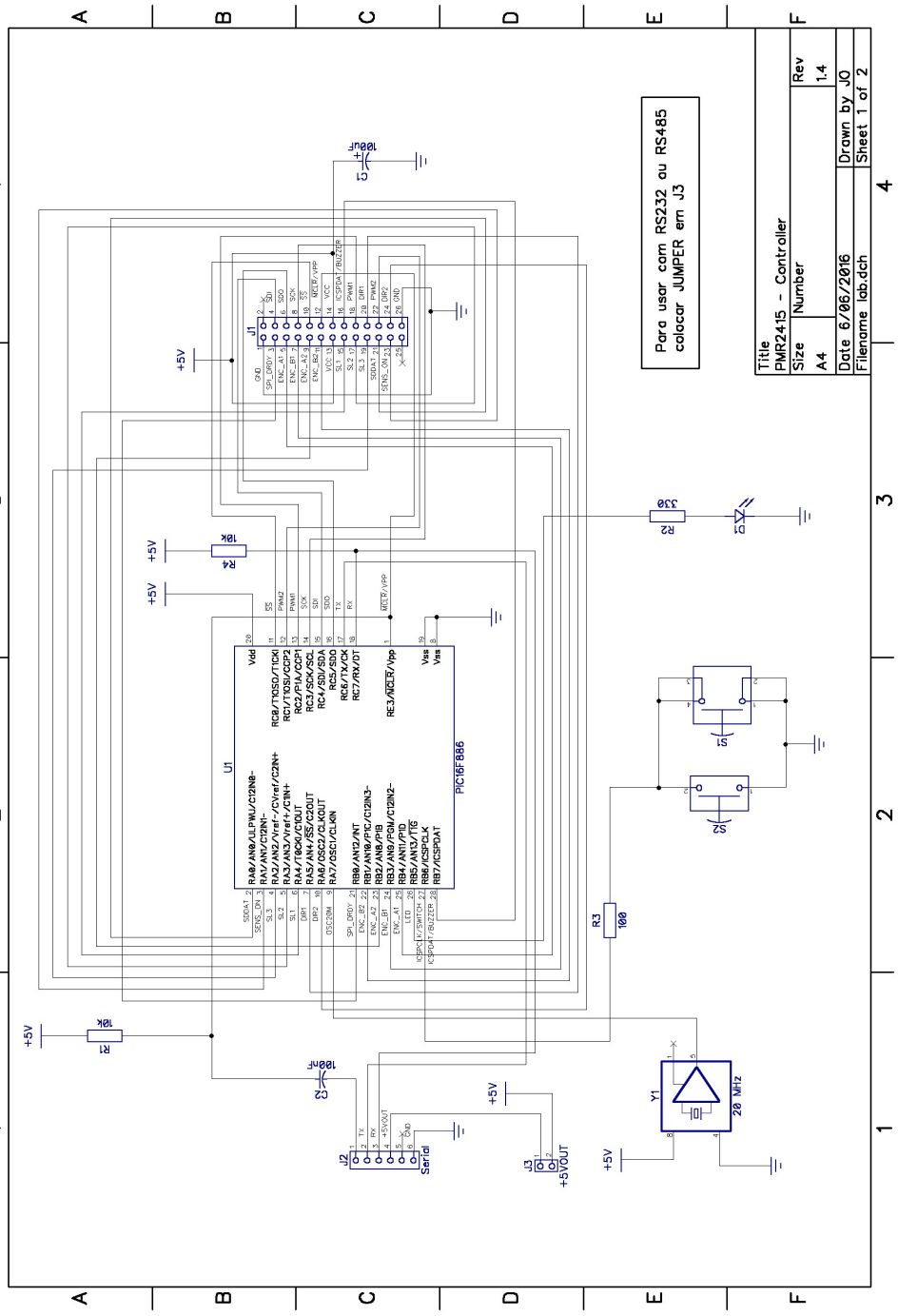
No relatório deverão estar identificados os membros do grupo, turma, disciplina e atividade e deverão constar os seguintes itens para cada tarefa implementada:

1. Texto explicativo sobre como resolver o problema da tarefa proposta.
2. Texto explicativo sobre o funcionamento do programa. Incluir condições nas quais que são sinalizadas no LED RGB.
3. Discutir até que ponto os objetivos foram alcançados.
4. Código fonte com comentários explicando o funcionamento do programa. Incluir somente os programas que foram escritos para essas atividades.

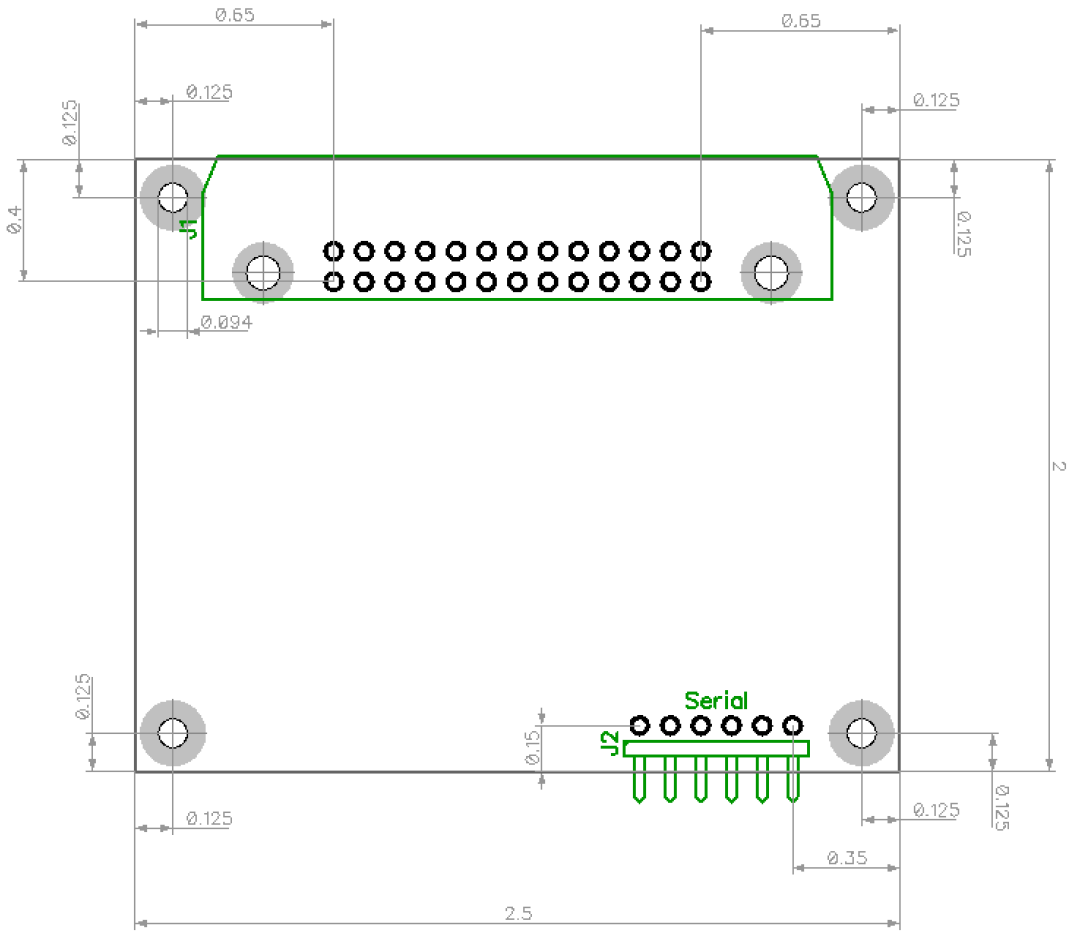
Parte III

Anexos

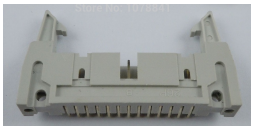
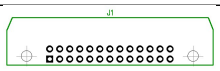
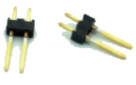
Anexo 1 – Diagrama Lógico

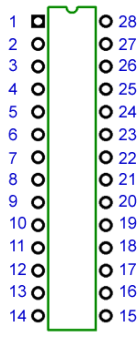


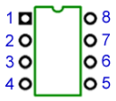
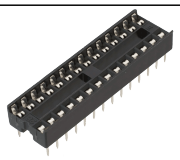
Anexo 2 – Posicionamento inicial de componentes para Layout



Anexo 3 – Lista de Componentes

Ref	Descrição	Foto	Footprint
C1	Capacitor eletrolítico de 100 µF	 OBS: o terminal mais comprido é o positivo	
D1	LED vermelho ou verde de 3 mm ou de 5 mm	 OBS: o terminal mais comprido é o ânodo, o lado chanfrado é o cátodo	 OBS: a ilha quadrada é o cátodo
J1	Header macho de 26 pinos com ejetor (2x13) horizontal		 OBS: a ilha quadrada é o pino 1
J2	Header macho de 6 pinos horizontal	 OBS: o lado mais curto é o da solda	 OBS: a ilha quadrada é o pino 1
J3	Header macho de 2 pinos vertical	 OBS: o lado mais curto é o da solda	 OBS: a ilha quadrada é o pino 1

R1	Resistor de 10 kΩ, 1/8W, 5%	 OBS: as cores são marrom, preto, laranja e dourado	
R2	Resistor de 330Ω, 1/8W, 5%	 OBS: as cores são laranja, laranja, marrom e dourado	
R3	Resistor de 100Ω, 1/8W, 5%	 OBS: as cores são marron, preto, marron e dourado	
S1	Chave Táctil (SPST) normalmente aberta (NO) OBS: somente S1 ou S2 será fornecida, o layout deve ser preparado para aceitar qualquer um dos dois modelos.		
S2	Chave Táctil (SPST) normalmente aberta (NO) OBS: somente S1 ou S2 será fornecida, o layout deve ser preparado para aceitar qualquer um dos dois modelos.		
U1	Microcontrolador PIC16F886 OBS: esse componente não será soldado na placa. No seu lugar deverá ser soldado o soquete correspondente.		 OBS: visto por cima

Y1	Oscilador à cristal de 20 MHz OBS: esse componente não será soldado na placa. No seu lugar deverá ser soldado o soquete correspondente.	 OBS: esse encapsulamento corresponde a um DIP de 8 pinos. Os pinos 2, 3, 6 e 7 não exsitem nesse encapsulamento	 OBS: visto por cima
U1	Soquete para CI 28 pinos DIL (300 mils)		
Y1	Soquete para CI 8 pinos DIL		

Anexo 4 - Headers das Bibliotecas

adc.h

```
/*
 * File:     adc.h
 * Author:   Jun Okamoto Jr.
 * Date:     27/02/2016
 * Comments: configura e faz leitura de canais analógicos
 * Revision history:
 */

#ifndef ADC_H
#define ADC_H

#include <xc.h>  // compilador XC8

extern void adc_init (unsigned char ADC_Channel); ///< inicializa canal analógico

extern unsigned int adc_read(unsigned char ADC_Channel); ///< faz a leitura de um canal analógico

#endif
```

always.h

```
/*
Common header file

Designed by Shane Tolmie of www.microchip.com corporation.  Freely distributable.
Questions and comments to webmaster@microchip.com
Lots of Hi-Tech C FAQ and sample source code at www.microchip.com.

For Microchip PIC16Fx.

Compiled with Hitech-C v7.85
[jo:130816] tested with XC8 and works fine.

Usage: #include in all ".c" files in project along with "pic.h"
       [jo:130816] change "pic.h" by "xc.h" with XC8 compiler

[jo:160206] incluido #define para OFF e ON
[jo:160227] incluido #define para ANALOG e DIGITAL
```

```
*/

//warning about #if statements: if any of the arguments are spelled wrong or
//unrecognised by the compiler, it will not generate a warning, but not include code

#ifndef ALWAYS_H
#define ALWAYS_H

/*
Turning bits on/off according to mask
use ~0 instead of 0xFF, etc, because this ensures machine independence
if int changes from 16-bit to 32-bit
Example C:
x=0b001;
bits_on(x,0b100)  //now x=0b101
*/

#define bits_on(var,mask) var |= mask
#define bits_off(var,mask) var &= ~0 ^ mask

//defines
#define INPUT 1  //port directions, ie: TRISA0=INPUT;
#define OUTPUT 0
#define TRUE 1
#define FALSE 0
#define HIGH 1
#define LOW 0
#define hi 1
#define lo 0
#define OFF 0          // [jo:160207] definição para desligado
#define ON 1           // [jo:160207] definição para ligado
#define ANALOG 1       // [jo:160227] definição para analógico
#define DIGITAL 0      // [jo:160227] definição para digital
#define b asm("nop")  //convenient point for breakpoint (debugging)
#define l while(1)    //loop for ever (debugging)

//see AM576.  If interrupt occurs just when gie gets set to zero, it won't be cleared

#define gie_on    GIE=1
#define gie_off   while(GIE==1) GIE=0

/*
Reading an 8-bit byte in a 16-bit int

With Hi-Tech C, this method is better than using pointers, as using pointers in
different banks needs different #defines

It is just as efficient - the optimizer picks out the correct
byte.  Of course, >>7 requires 7 shifts.

This method cannot be used to alter a hi/lo byte, this needs pointers (as below)

Example C:
unsigned int x;
unsigned char y;
x=0x1234;
```

```
y=hibyte(x);      //now y=0x12 - works for variables in any bank 0 to 3
y=lobyte(x);      //now y=0x34

lobyte(x)=0xaa;   //will not work :( - use pointers
*/

#define hibyte(x) (unsigned char)(x>>8)
#define lobyte(x) (unsigned char)(x & 0xFF)

/*
  given variable of any type (char, uchar, int, uint, long) it modifies
  the unsigned char residing at that memory location
  for ints, byte1 is msb, byte0 is lsb (least significant byte)
  for longs byte3 is msb, byte0 is lsb

  ie: sample C code

  unsigned int myint=0x4321;
  long mylong=0x87654321;

  //for myint    byte1(myint)=0x43; (msb) and byte0(myint)=0x21; (lsb)
  //for mylong   byte3(mylong)=0x87; (msb), byte2(mylong)=0x65;
                 byte1(mylong)=0x43; and byte0(mylong)=0x21; (lsb)

  note: to avoid fixup overflow errors add bankX if the target variable
        resides in banks 1, 2 or 3
*/

#define byte0(x) (unsigned char)(*(((unsigned char *)&x)+0))
#define byte1(x) (unsigned char)(*(((unsigned char *)&x)+1))
#define byte2(x) (unsigned char)(*(((unsigned char *)&x)+2))
#define byte3(x) (unsigned char)(*(((unsigned char *)&x)+3))
#define lobyte_atbank0 byte0  //another way of saying it
#define hibyte_atbank0 byte1

#define byte0_atbank1(x) (unsigned char)(*(((bank1 unsigned char *)&x)+0))
#define byte1_atbank1(x) (unsigned char)(*(((bank1 unsigned char *)&x)+1))
#define byte2_atbank1(x) (unsigned char)(*(((bank1 unsigned char *)&x)+2))
#define byte3_atbank1(x) (unsigned char)(*(((bank1 unsigned char *)&x)+3))

#define lobyte_atbank1 byte0_atbank1  //another way of saying it
#define hibyte_atbank1 byte1_atbank1

#define byte0_atbank2(x) (unsigned char)(*(((bank2 unsigned char *)&x)+0))
#define byte1_atbank2(x) (unsigned char)(*(((bank2 unsigned char *)&x)+1))
#define byte2_atbank2(x) (unsigned char)(*(((bank2 unsigned char *)&x)+2))
#define byte3_atbank2(x) (unsigned char)(*(((bank2 unsigned char *)&x)+3))

#define byte0_atbank3(x) (unsigned char)(*(((bank3 unsigned char *)&x)+0))
#define byte1_atbank3(x) (unsigned char)(*(((bank3 unsigned char *)&x)+1))
#define byte2_atbank3(x) (unsigned char)(*(((bank3 unsigned char *)&x)+2))
#define byte3_atbank3(x) (unsigned char)(*(((bank3 unsigned char *)&x)+3))

/*
  given variable of any type (char, uchar, int, uint, long) it modifies
  the int residing at that memory location
```

```
  ie: sample C code

  unsigned char array[4];
  unsigned int test;

  uint_atbyteaddr(&array[0])=0x4321;   //now array[0->3]={0x21,0x43,0,0};
  uint_atbyteaddr(&array[0+2])=0x8765; //now array[0->3]={0x21,0x43,0x65,0x87};
  test=uint_atbyteaddr(&array[0+2]);   //now test=0x8765

  note: do NOT use &(array[0]+1) to reference the int stored at array[1] as it will
        reference the int after array[0] in pointer arithmetic.  This will
        result with the int at array[2].

        Instead use &array[0+1] to reference the int at uchar array[1]

  note: to avoid fixup overflow errors add bankX if the target variable
        resides in banks 1, 2 or 3
*/

#define uint_atbyteaddr(x)      (unsigned int)(*((unsigned int *)x)))
#define uint_atbank1byteaddr(x) (unsigned int)(*((bank1 unsigned int *)x)))
#define uint_atbank2byteaddr(x) (unsigned int)(*((bank2 unsigned int *)x)))
#define uint_atbank3byteaddr(x) (unsigned int)(*((bank3 unsigned int *)x)))

#define THE_BEER_IS_PLENTIFUL_AND_THE_PARTY_SWINGING TRUE

/*

NOTE: it is not recommended that unions are used to reference hi/lo bytes or
bits of a variable.  Use >>8 or &FF or pointers instead, as above.  It makes
passing variables to a function difficult, as the function must be defined to
accept variables of the same union.  Then, the function will no longer accept
normally defined variables.

these two structures allow access to 2 byte word, high and low bytes of variable
declaration: union wordtype x;
usage:       x.word=0xABCD; x.byte.high=0xAB; x.byte.low=0xCD;
             x.part.bit15=1; (msb), x.part.bit0=1; (lsb)
declaration: union chartype x;
usage:       x.byte=0xAB;
             x.part.bit7=1; (msb), x.part.bit0=1; (lsb)
*/

struct sixteen_bits {
  unsigned char bit0  :1;
  unsigned char bit1  :1;
  unsigned char bit2  :1;
  unsigned char bit3  :1;
  unsigned char bit4  :1;
  unsigned char bit5  :1;
  unsigned char bit6  :1;
  unsigned char bit7  :1;
  unsigned char bit8  :1;
  unsigned char bit9  :1;
  unsigned char bit10 :1;
```

```
  unsigned char bit11 :1;
  unsigned char bit12 :1;
  unsigned char bit13 :1;
  unsigned char bit14 :1;
  unsigned char bit15 :1;
};

struct eight_bits {
  unsigned char bit0 :1;
  unsigned char bit1 :1;
  unsigned char bit2 :1;
  unsigned char bit3 :1;
  unsigned char bit4 :1;
  unsigned char bit5 :1;
  unsigned char bit6 :1;
  unsigned char bit7 :1;
};

struct two_bytes {
  unsigned char low;
  unsigned char high;
};

union wordtype {
  unsigned int word;
  struct two_bytes byte;
  struct sixteen_bits part;
};

union chartype {
  unsigned char byte;
  struct eight_bits part;
};
#endif
```

battery.h

```
/*
 * File:     battery.h
 * Author:   Jun Okamoto Jr.
 * Date:     9/02/2016
 * Comments: Faz leitura no nível da bateria do robô
 * Revision history:
 */

// This is a guard condition so that contents of this file are not included
// more than once.
#ifndef BATTERY_H
#define BATTERY_H

#include <xc.h>      // include processor files - each processor file is guarded.
#include <stdint.h>

void battery_init(); ///< inicializa leitura de bateria
```

```
uint8_t battery_level(); ///< faz a leitura do nível de bateria

#endif  /* BATTEY_H */
```

compass.h

```
/*
 * File:     compass.h
 * Authors:  Thiago Yukio Nagata Alves
 * Date:     28/04/2016
 * Comments: Controle da bússola do robÙ
 * Revision history:
 *           [jo:160608] comentários acertados para 2017 e documentação Doxygem
 */
#ifndef COMPASS_H
#define COMPASS_H
#include <xc.h>
#include <stdint.h>

void compass_init(); ///< inicializa bussola

int16_t compass_get_id(); ///< lê identificador da bussola (0x3d)

uint8_t compass_get_status(); ///< lê status da bussola

int16_t compass_get_X(); ///< lê intensidade do campo no eixo x

int16_t compass_get_Y(); ///< lê intensidade do campo no eixo y

int16_t compass_get_Z(); ///< lê intensidade do campo no eixo z

int8_t compass_get_temperature(); ///< lê temperatura

uint16_t compass_get_heading(); ///< lê o ângulo com o norte

void compass_calibrate(); ///< calibra a bussola, movimenta robô, requer pwm

void compass_enable(); ///< liga o bussola

void compass_disable(); ///< desliga o bussola

void compass_temperature_enable(); ///< habilita o sensor de temperatura

void compass_temperature_disable(); ///< desabilita o sensor de temperatura

void compass_set_operativemode(uint8_t mode); ///< configura o modo de operação
                               //   0 - low power, 1 - medium, 2 - high, 3 - ultra high

void compass_set_conversionmode(uint8_t mode); ///< configura o modo de conversão
                               //   0 - conversão contínua, 1 - conversão única

void compass_set_datarate(uint8_t mode); ///< configura a taxa de dados de saída
                              //   0 - 0.625Hz, 1 - 1.25Hz, 2 - 2.5Hz ..., 7 - 80Hz
```

```
void compass_set_fastdatarate(uint8_t mode); ///< habilita taxas de dados mais rápidas
                              //   0 - 155Hz,  1 - 300Hz,  2 - 560Hz,  3 - 1000Hz

void compass_set_scale(uint8_t scale); ///< configura o fundo de escala
                              //   0 - 4 gauss, 1 - 8, 2 - 12, 3 - 16

void compass_reset(); ///< reseta bussola

uint8_t compass_get_interruptsource(); ///< lê fonte de interrupção (não implementado)

void compass_configure_interrupts(uint8_t config); ///< configura interrupções (não implementado)

#endif
```

delay.h

```
/*

lowlevel delay routines

Designed by Shane Tolmie for www.microchip.com. Freely distributable.
Questions and comments to webmaster@microchip.com.

For Microchip 12C67x, 16C7x, 16F87x and Hi-Tech C
[jo:130813] tested and works for XC8 too!
[jo:160207] function names changed to comply with PMR2415 Coding standards

Example C:

#define PIC_CLK 4000000

#include "delay.h"

unsigned int timeout_int, timeout_char;

DelayUs(40);  //do NOT do DelayUs(N) of N<5 @ 4Mhz or else it executes DelayUs(255) !!!!
DelayUs(255); //max

dly250n;      //delay 250ns
dly1u;        //delay 1us

timeout_char=timeout_char_us(1147);
while(timeout_char-- && (RA1==0));  //wait up to 1147us for port RA1 to go high
                                    //  - this is the max timeout

timeout_int=timeout_int_us(491512);
while(timeout_int-- && (RA1==0));   //wait up to 491512us for port RA1 to go high
                                    //  - this is the max timeout

*/

#define PIC_CLK 20000000 // 20Mhz - Oscilattor frequency in Hz
```

```
#ifndef __DELAY_H
#define __DELAY_H

extern unsigned char delayus_variable;

#if (PIC_CLK == 4000000)
  #define dly125n please remove; for 32Mhz+ only
  #define dly250n please remove; for 16Mhz+ only
  #define dly500n please remove; for 8Mhz+ only
  #define dly1u asm("nop")
  #define dly2u dly1u;dly1u
#elif (PIC_CLK == 8000000)
  #define dly125n please remove; for 32Mhz+ only
  #define dly250n please remove; for 16Mhz+ only
  #define dly500n asm("nop")
  #define dly1u dly500n;dly500n
  #define dly2u dly1u;dly1u
#elif ( (PIC_CLK == 16000000) || (PIC_CLK == 16257000) )
  #define dly125n please remove; for 32Mhz+ only
  #define dly250n asm("nop")
  #define dly500n dly250n;dly250n
  #define dly1u dly500n;dly500n
  #define dly2u dly1u;dly1u
#elif (PIC_CLK == 20000000)
  #define dly200n asm("nop")
  #define dly400n dly250n;dly250n
  #define dly2u dly400n;dly400n;dly400n;dly400n;dly400n
#elif (PIC_CLK == 32000000)
  #define dly125n asm("nop")
  #define dly250n dly125n;dly125n
  #define dly500n dly250n;dly250n
  #define dly1u dly500n;dly500n
  #define dly2u dly1u;dly1u
#else
  #error delay.h - please define pic_clk correctly
#endif

//*****
//delay routine
#if PIC_CLK == 4000000
  #define DelayDivisor 4
  #define WaitFor1Us asm("nop")
  #define Jumpback asm("goto $ -2")
#elif PIC_CLK == 8000000
  #define DelayDivisor 2
  #define WaitFor1Us asm("nop")
  #define Jumpback asm("goto $ -2")
#elif ( (PIC_CLK == 16000000) || (PIC_CLK==16257000) )
  #define DelayDivisor 1
  #define WaitFor1Us asm("nop")
  #define Jumpback asm("goto $ -2")
#elif PIC_CLK == 20000000
  #define DelayDivisor 1
  #define WaitFor1Us asm("nop"); asm("nop")
  #define Jumpback asm("goto $ -3")
#elif PIC_CLK == 32000000
```

```
  #define DelayDivisor 1
  #define WaitFor1Us asm("nop"); asm("nop"); asm("nop"); asm("nop"); asm("nop")
  #define Jumpback asm("goto␣$␣-␣6")
#else
  #error delay.h - please define pic_clk correctly
#endif

#define delay_us(x) { \
  delayus_variable=(unsigned char)(x/DelayDivisor); \
  WaitFor1Us; } \
  asm("decfsz␣_delayus_variable,f"); \
  Jumpback;

/*

timeouts:

C code for testing with ints:

 unsigned int timeout;
  timeout=4000;
  PORT_DIRECTION=OUTPUT;
  while(1) {
    PORT=1;
    timeout=8000;
    while(timeout-- >= 1);  //60ms @ 8Mhz, opt on, 72ms @ 8Mhz, opt off
    PORT=0;
  }

Time taken:  optimisations on:  16cyc/number loop, 8us @ 8Mhz
             optimisations off: 18cyc/number loop, 9us @ 8Mhz
             with extra check ie: && (RB7==1), +3cyc/number loop, +1.5us @ 8Mhz

C code for testing with chars:

  similar to above

Time taken:  optimisations on:  9cyc/number loop, 4.5us @ 8Mhz
             with extra check ie: && (RB7==1), +3cyc/number loop, +1.5us @ 8Mhz

Formula: rough timeout value = (<us desired>/<cycles per loop>) * (PIC_CLK/4.0)

To use:  //for max  timeout of 1147us @ 8Mhz
  #define LOOP_CYCLES_CHAR 9 // how many cycles per loop, optimizations on
  #define timeout_char_us(x) (unsigned char)((x/LOOP_CYCLES_CHAR)*(PIC_CLK/4.0))
  unsigned char timeout;
  timeout=timeout_char_us(1147);  // max timeout allowed @ 8Mhz, 573us @ 16Mhz
  while((timeout-- >= 1) && (<extra condition>)); //wait

To use: //for max 491512us, half sec timeout @ 8Mhz
  #define LOOP_CYCLES_INT 16  // how many cycles per loop, optimizations on
  #define timeout_int_us(x) (unsigned int)((x+/LOOP_CYCLES_INT)*(PIC_CLK/4.0))
  unsigned int timeout;
  timeout=timeout_int_us(491512);  // max timeout allowed @ 8Mhz
  while((timeout-- >= 1) && (<extra condition>));  // wait
*/
```

```
#define LOOP_CYCLES_CHAR   9  // how many cycles per loop, optimizations on
#define timeout_char_us(x)  (long)(((x)/LOOP_CYCLES_CHAR)*(PIC_CLK/1000000/4))

#define LOOP_CYCLES_INT   16  // how many cycles per loop, optimizations on
#define timeout_int_us(x) (long)(((x)/LOOP_CYCLES_INT)*(PIC_CLK/1000000/4))

//if lo byte is zero, faster initialization by 1 instrucion
#define timeout_int_lobyte_zero_us(x) (long)(((x)/LOOP_CYCLES_INT)*(PIC_CLK/4.0)&0xFF00)

// function prototypes
// [jo:160207] orignal names
//void DelayBigUs(unsigned int cnt);
//void DelayMs(unsigned char cnt);
//void DelayMs_interrupt(unsigned char cnt);
//void DelayBigMs(unsigned int cnt);
//void DelayS(unsigned char cnt);

// Function prototypes
// [jo:160207] coding standard compliant names
void delay_big_us(unsigned int cnt);
void delay_ms(unsigned char cnt);
void delay_ms_interrupt(unsigned char cnt);
void delay_big_ms(unsigned int cnt);
void delay_s(unsigned char cnt);

#endif
```

key.h

```
/*
 * File:     key.h
 * Author:   Jun Okamoto Jr.
 * Date:     7/02/2016
 * Comments: Controla leitura de chave
 * Revision history:
 */

#ifndef KEY_H
#define KEY_H

#include <xc.h> // include processor files - each processor file is guarded.
#include <stdint.h>

void key_init(); ///< inicializar chave

void key_debounce(int cycles); ///< faz o debounce da chave, deve ser chamada na interrupção periódica

void key_read(char port); ///< faz a leitura da chave, deve ser chamada no I-O-C do Port B

char key_pressed(); ///< determina de a chave foi pressionada

#endif  /* KEY_H */
```

lcd8x2.h

```
/*
 * File:     lcd8x2.h
 * Authors:  Jun Okamoto Jr.
 *           Bruno Alan Miyamoto
 * Date:     7/02/2016
 * Comments: Controle do display LCD 8x2 do robô
 * Revision history:
 */

#ifndef LCD8X2_H
#define LCD8X2_H

#include <xc.h>
#include <stdint.h>

extern void lcd_init(void); ///< inicializa display, utiliza SPI

extern void lcd_clear(void); ///< limpa e coloca cursor na linha 1 coluna 1

extern void lcd_puts(const char * s); ///< escreve um string de caracteres no LCD

extern void lcd_goto(uint8_t pos); ///< move o cursor para determinada posição

extern void lcd_show_cursor(int on); ///< desliga ou liga cursor

extern void lcd_putchar(char c); ///< escreve um caracter no LCD

extern void lcd_set_display_movement(uint8_t dir); ///< desloca display para direita ou esquerda

extern uint8_t lcd_get_cursor_position(); ///< retorna a posição do cursor

extern void lcd_set_cursor_movement(uint8_t dir); ///< move o cursor para direita ou esquerda

extern void lcd_passthrough_command(uint8_t command); ///< passa comando direto para LCD

extern void lcd_passthrough_data(uint8_t data); ///< passa dado direto para LCD

#endif
```

led_rgb.h

```
/*
 * File:      led_rgb.h
 * Author:    Jun Okamoto Jr.
 * Date:      11/03/2016
 * Comments:  Controla do LED RGB do robô pela interface SPI
 * Revision history:
 */

#ifndef LED_RGB_H
#define LED_RGB_H
```

```
#include <xc.h> // include processor files - each processor file is guarded.

#define BLACK   0
#define BLUE    1
#define GREEN   2
#define CYAN    3
#define RED     4
#define MAGENTA 5
#define YELLOW  6
#define WHITE   7

void led_rgb_init(); ///< inicializa LED RGB, utiliza SPI

void led_rgb_set_color(char led_color); ///< define cor do led, inclusive apagado

void led_rgb_set_red_level(uint8_t level); ///< define a intensidade do canal vermelho

void led_rgb_set_green_level(uint8_t level); ///< define a intensidade do canal verde

void led_rgb_set_blue_level(uint8_t level); ///< define a intensidade do canal azul

#endif  /* LED_RGB_H */
```

pwm.h

```
/*
 * File:      pwm.h
 * Author:    Jun Okamoto Jr.
 * Date:      10/03/2016
 * Comments:  Controle PWM para os motores
 * Revision history:
 * [jo:160311] colocados #defines para direções de movimento
 */

#ifndef LED_PWM_H
#define LED_PWM_H

#define FORWARD 0 // [jo:160311] direção para frente
#define REVERSE 1 // [jo:160311] direção para trás
#define LEFT    2 // [jo:160311] gira para a esquerda
#define RIGHT   3 // [jo:160311] gira para a direita

void pwm_init(); ///< inicializa a funcao de PWM com uma determinada frequencia

void pwm_set(int channel, int duty_cycle); ///< define o duty cycle

void pwm_direction(int dir); ///< altera a direção do movimento

#endif
```

sensor.h

```
/*
 * File:       sensor.h
 * Author:     Jun Okamoto Jr.
 * Date:       11/03/2016
 * Comments:  Dá acesso aos sensores e buzzer do robô
 * Revision history:
 */

#ifndef SENSOR_H
#define SENSOR_H

void sensor_init(); ///< inicializa sensores de linha e proximidade

void sensor_power(char state); ///< liga e desliga alimentação dos sensores e buzzer

int sensorLine_read(); ///< faz as três leituras do sensor de linha

int sensorNear_read(); ///< faz a leitura do sensor de distancia

#endif
```

serial.h

```
///
/// Asynchonous serial channel header file
///

#ifndef SERIAL_H
#define SERIAL_H

/// Initialize an asynchronous serial channel
void serial_init(void);

/// Get an 8-bit character
/// IMPORTANT: it does not return if a character is not received
unsigned char getch(void);

/// Get an 8-bit character
/// It waits a timeout for the chacater to arrive.
/// If no chacarter arrives within the timeout,
/// this function returns 255.
///
unsigned char getch_timeout(void);

/// If there has been a previous timeout error from getch_timeout(),
/// this function returns TRUE
unsigned char usart_timeout(void);

/// Check and return an 8-bit character
/// If no character is ready in the input buffer
/// this function returns 255.
/// IMPORTANT: it always return without waiting
///
unsigned char chkchr(void);
```

```
/// Send an 8-bit character
void putch(unsigned char c);

/// Send a string
void putst(register const char * str);

/// Send a character in decimal form
void putchdec(unsigned char c);

/// Send a character in hexadecimal form
void putchhex(unsigned char c);

/// Send a integer number as hexadecimal form
void putinthex(unsigned int c);

/// Put a line feed
#define putlf putst("\n")

#endif
```

spi.h

```
/*
 * File:     spi.h
 * Author:   Bruno Alan Miyamoto
 * Date:     7/02/2016
 * Comments: Controla interface SPI
 * Revision history:
 */

#ifndef SPI_H
#define SPI_H

#include <xc.h>
#include <stdint.h>

void spi_init(); ///< inicializa o SPI no modo 0,0 master

uint8_t spi_exchange(uint8_t data); ///< transmite e recebe por SPI

uint8_t spi_read(); ///< lê byte pelo SPI

void spi_write(uint8_t data); ///< escreve byte pelo SPI

#endif
```