PROGRAMAÇÃO BÁSICA EM C PARA MICROCONTROLADORES PIC

KÁSSIO HENRIQUE RIBEIRO

UFLA – Universidade Federal de Lavras

Lavras – MG

2016

SUMÁRIO

1

INTRODUÇÃO

O conceito de microcontroladores e a sua estrutura interna já foram vistos em outros materiais didáticos disponibilizados pela equipe, portanto, considera-se que o leitor já tenha isso em mente e que trate esse material como uma sequência aos estudos já realizados.

Nessa apostila daremos ao leitor os passos iniciais da programação de microcontroladores da família PIC utilizando a linguagem C, e para isso, trabalharemos com o compilador mikroC que possui uma IDE intuitiva chamada mikroC PRO for PIC®.

Deve-se lembrar que o fato de especificarmos o estudo a uma família de microcontroladores e a um compilador, não restringe nosso conhecimento apenas para os modelos que serão abordados aqui, tendo em vista que grande parte dos microcontroladores funcionará de maneira similar e todos os conceitos aqui adquiridos poderão ser facilmente traduzidos para outros compiladores ou modelos de microcontrolaores de diferentes fabricantes.

Cada tópico terá uma explicação breve, um ou mais exemplos de aplicação onde os comandos serão apresentados e comentados, e no fim, teremos um exercício onde o leitor tentará pôr em prática o conhecimento recém-adquirido. Para testar os códigos pode-se montar o esquemático eletrônico no software ISIS, presente no pacote Proteus®. Desejo um ótimo aprendizado e que o material seja de grande ajuda para inicia-lo no assunto e servir como base para o desenvolvimento de seus próprios projetos.

ENTRADA E SAÍDA DE DADOS DIGITAIS

Os microcontroladores se comunicam com o meio externo através de pinos organizados em barramentos denominados PORTs, tais pinos podem ser configurados via software como entradas ou saídas. Os dados transferidos durante a comunicação podem ser digitais e/ou analógicos (ADC)*. Por isso, antes de começarmos a programar é necessário conferirmos a folha de dados (datasheet) do microcontrolador que estamos utilizando, para sabermos quais pinos estão habilitados para serem entradas ou saídas, digitais ou analógicas. Vale lembrar que devemos ter em mente os requisitos de nosso projeto para sabermos definir o modelo de microcontrolador que se adequará às exigências, evitando erros de dimensionamento que podem encarecer a implementação ou torna-la inviável. Existem modelos, por exemplo, que não possuem portas analógicas e em contrapartida há os que possuem muitas.

Segue abaixo um diagrama de pinos do modelo PIC16F628A, encontrado em sua folha de dados. Na folha**, também pode-se encontrar uma tabela que descreve as funções que cada pino pode assumir de acordo com as siglas do diagrama abaixo:

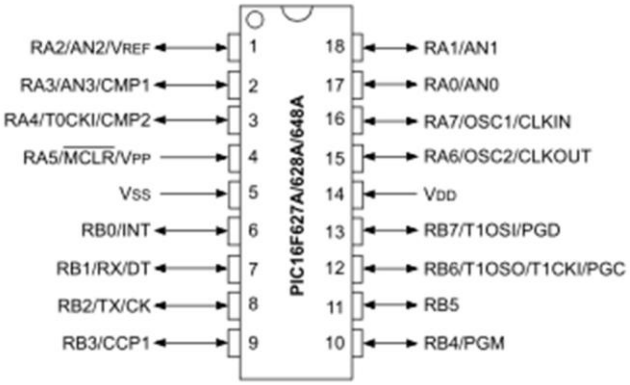

Figura 1 Diagrama de pinos do PIC16F628A

*A conversão de dados analógicos para dados digitais (ADC), já foi abordada em materiais anteriores.

*Não deixem de consultar a folha de dados do modelo que estiverem utilizando, nela consta toda a informação necessária para se trabalhar com o mesmo. Como por exemplo, capacidade das memórias e frequências de clock suportadas.

Através da figura acima, observamos que o modelo PIC16F628A possui seus pinos organizados em dois barramentos, A e B, cada um com 08 pinos enumerados de 0 a 7. Também existem dois pinos Vss e Vdd destinados à alimentação do microcontrolador.

Para começarmos a programar o microcontrolador, iniciaremos um novo projeto no software mikroC PRO for PIC® seguindo os seguintes passos básicos. Esses passos, apesar de não serem reescritos no restante do material, deverão ser repetidos futuramente na criação de outros exemplos e exercícios. Quando houverem alterações, as mesmas serão mencionadas no enunciado, onde caberá ao leitor consultar em qual passo elas ocorrem:

1. Abra o mikroC PRO for PIC®
2. Na aba Project, selecione a opção New Project
3. A janela do Wizard que auxiliará na criação do projeto abrirá, selecione a opção Next
4. Nessa segunda janela é exigido que se escolha o modelo do PIC, coloque o inicialmente o modelo 16F628A e selecione Next
5. Selecione frequência de clock do microcontrolador, coloque inicialmente 4 MHz e Next
6. Selecione onde deseja salvar o novo projeto e o nomeie, em seguida, Next
7. Nessa janela temos a opção de adicionar arquivos ao projeto, no nosso caso, apenas selecionamos Next
8. Aqui o programa pergunta quais as bibliotecas serão adicionadas para serem tomadas como base durante o desenvolvimento de nosso software, vamos marcar a opção “Include All”, que vem como padrão e selecionar Next. Futuramente poderemos selecionar as bibliotecas que precisarmos sem a necessidade de incluir todas.
9. Marcamos a opção “Open Edit Project window to set Configuration bits” e Finish
10. Na janela de Edit Project, que também pode ser acessada pela aba Project, selecionaremos o tipo de oscilador como XT (cristal externo) e desabilitaremos todos os outros itens, incluindo o Master Clear. Em seguida, após conferir o modelo do PIC e a frequência do clock, selecionem a opção OK.

Com a execução dos passos acima visualizaremos a tela principal do mikroC, onde inicia-se a escrita do programa. Note que a rotina principal “main()” já possui sua estrutura pronta. Faremos o primeiro exemplo onde os comandos começarão a ser apresentados com suas respectivas explicações.

Mas antes de darmos continuidade é bom que seja explicado o conceito de registradores, pois toda a configuração do microcontrolador é feita através deles. Sempre existirá um ou mais registradores dedicados à configuração de procedimentos específicos como interrupções, timers, comunicação serial entre outros que veremos mais à frente. Trata-se de um vetor com oito bits onde cada bit habilita ou desabilita uma determinada função dentro procedimento a que se refere o registrador. Na folha de dados podemos consultar todos os registradores existentes em um determinado modelo e todas as funções de seus respectivos bits. Existem diversas maneiras de nos referirmos aos bits dos registradores, seguem abaixo alguns exemplos:

1. REGISTRADOREX.BITDESEJADO=1 – Nesse caso estamos atribuindo o valor 1 ao bit chamado BITDESEJADO dentro do registrador REGISTRADOREX. Se BITDESEJADO não for um bit do registrador REGISTRADOREX, ocorrerá um erro durante a compilação informando que o bit não existe.
2. REGISTRADOREX=0b00001111 – Nesse caso estamos atribuindo o valor 0 aos bits de 7 a 4 e o valor 1 aos bits de 3 a 0
3. REGISTRADOREX=0 – Aqui todos os bits do registrador REGISTRADOREX estão recebendo o valor “0”.
4. BITDESEJADO_bit – Essa é outra maneira de nos referirmos aos bits, mas nesse caso, sem a necessidade de se informar o registrador.

Após essa introdução, é hora de iniciarmos a programação do nosso microcontrolador. A princípio, faremos um software simples que terá como função configurar dois pinos digitais de um determinado barramento, um como saída e outro como entrada. Essa saída estará ligada a um LED que terá que piscar com a frequência de 1 Hz enquanto o pino de entrada receber um sinal de nível lógico alto (pressionando-se um botão configurado como pull-down). No decorrer da apostila, desconsideraremos as ligações externas entre os dispositivos e o microcontrolador, mantendo o enfoque na programação e considerando que todo o sistema foi construído de maneira correta.

Exemplo 1 – Entradas e Saídas Digitais

```
#define BOTAO  portb.rb0
#define LED  portb.rb1

void main() {

TRISB=0b00000001;
PORTB=0;

while(1){
          if(BOTAO==1){
          LED=1;
          delay_ms(500);
          LED=0;
          delay_ms(500);
          }
 }
}
```

Comentários:

- O Comando "define" tem como intuito facilitar a substituição de pinos utilizados e o entendimento do Software. Ele atribui TAGs que podem ser citadas em todo o código para se referir ao pino correspondente.
- A rotina main(), declarada como void, demarca o início da rotina principal onde ocorrem chamadas de funções, laços condicionais, atribuição de valores a variáveis, configurações de registradores, definições de pinos de entrada e saída, estado inicial dos pinos de saída e etc...
- O registrador TRIS tem como função definir quais pinos do barramento especificado pela letra que sucede a palavra TRIS serão entradas digitais e quais serão saídas digitais. O segmento "0b" identifica o vetor de 8 bits do registrador TRIS. Tem-se, portanto, começando-se da esquerda para direita os pinos que vão de 7 a 0 onde o valor 1 atribuído a um bit, configura o respectivo pino como entrada e o valor 0 o configura como saída. No nosso caso definimos o pino RB0 como entrada pois receberá o botão e o restante dos pinos como saída, inclusive RB1 que receberá o LED.
- O registrador PORT, define o nível logico dos pinos do barramento especificado pela letra que sucede a palavra PORT, onde 0 mantem os oito pinos em nível logico baixo (0V) e 1

os mantem em nível logico alto (5V). A definição também poderia ser feita bit a bit (especificando os pinos que estariam em 0V e 5V), ou definindo os oito bits como no caso do TRIS. Lembrando que a atribuição só terá efeito sobre os pinos que foram definidos como saída. No nosso caso estamos dizendo que tudo começará em 0V.

- A condição while(1) significa enquanto verdadeiro, e garante a repetitividade da execução do nosso código sem a necessidade de se refazer a configuração inicial de registradores.
- A condição if(BOTAO==1), inicia nossa lógica, onde dizemos que se o botão for pressionado, o LED ascenderá (LED=1), ficará aceso por 0,5 segundos (delay_ms(500)), apagará (LED=0) e permanecerá apagado também por 0,5 segundos. O ciclo recomeça caso o botão continue pressionado. Além do comando delay_ms, também existe o delay_us. O primeiro atrasa a execução do código em milissegundos e o segundo em microssegundos. Atrasar não é uma boa alternativa para contagem de tempo e mais a frente aprenderemos outro método, os timers.

Exercício Proposto – Ainda no modelo PIC16F628A, os pinos RB7, RB5, RB3 e RB1 serão ligados a LEDs azuis, assim como os pinos RB6, RB4, RB2 e RB0 serão ligados a LEDs amarelos. Faça com que as luzes azuis acedam e apaguem simultaneamente e se alternem com as amarelas em um intervalo de tempo de sua escolha. Adicione um botão em RA0 que fará com que todas as luzes, azuis e amarelas, se acendam por 2 segundos caso seja pressionado (pull-down).

3

CONVERSOR ANALÓGICO DIGITAL

Assim como todo circuito digital, os microcontroladores não são aptos a receberem valores analógicos diretamente em suas entradas digitais, tendo em vista que só conseguem interpretar um conjunto de valores finitos. Como uma forma de se resolver esse problema, em alguns modelos de microcontroladores existem pinos que além de poderem assumir as funções de entradas e saídas digitais, também podem ser configurados como entradas analógicas. Quando habilitamos uma entrada analógica, também estamos habilitando o ADC (Analog-to-Digital Converter), ou conversor analógico-digital, que recebe o sinal analógico após ter passado por um amostrador e por um segurador (sample and hold), e realiza aproximações sucessivas até chegar em um resultado binário próximo ao valor analógico de entrada. Esse valor possui entre 8 e 10 bits de resolução, dependendo do modelo do microcontrolador. O resultado convertido é armazenado em uma variável específica, que poderá ser utilizada no código do programador para a tomada de decisões. Esse procedimento é periódico com período T, chamado período de amostragem, que pode ser alterado em um registrador específico.

Para o nosso exemplo, utilizaremos o PIC18F4550 que possui um conversor ADC com resolução de 10 bits, pois o PIC16F628A utilizado até então, não possui ADC. Nesse exemplo faremos a leitura da tensão de um potenciômetro que estará ligado a uma das portas analógicas do nosso PIC. Associaremos três LEDs (Verde, Amarelo e Vermelho) a três portas digitais, com o intuito de indicar três valores distintos de tensão, ou seja, se a tensão estiver acima de um determinado valor considerado alto acenderemos o LED verde, se estiver em um valor mediano acenderemos o amarelo e caso esteja baixa acenderemos o vermelho.

Exemplo 2 – Conversor Analógico Digital

```
#define verde  portd.rd0
#define amarelo  portd.rd1
#define vermelho  portd.rd2

int conv=0;
```

```
void main() {

TRISD=0;
TRISA=0b00000001;
PORTD=0;

ADCON1.VCFG1=0;
ADCON1.VCFG0=0;
ADCON1.PCFG3=1;
ADCON1.PCFG2=1;
ADCON1.PCFG1=1;
ADCON1.PCFG0=0;
ADC_init();

while(1){

conv=adc_read(0);

if (conv>800) {
vermelho=0;
amarelo=0;
verde=1;
}if (conv<800 && conv>400){
verde=0;
vermelho=0;
amarelo=1;
}if (conv<400){
amarelo=0;
verde=0;
vermelho=1;
}
}
}
```

Comentários:

- Fizemos os devidos 'defines'.
- Foi declarada uma variável do tipo inteiro chamada conv inicialmente em zero, que armazenará o resultado da conversão do valor analógico para digital.
- Em seguida, dentro da rotina principal (main), definimos o pino RA0 como entrada e o restante do barramento A como saída. Definimos todo o barramento D como saída e atribuímos nível lógico baixo ao mesmo, inicialmente.
- Definimos os bits do registrador ADCON1, configurando-o para tomar como referência de tensão da conversão, a alimentação do PIC, e para termos apenas o pino RA0 como

entrada analógica (AN0). A relação dos bits desse registrador e suas respectivas funções estão presentes na folha de dados, não deixe de conferir.

- As funções ADC_Init() e ADC_Read(n°da porta analógica) fazem parte de uma biblioteca presente no mikroC PRO® e facilitam a implementação da conversão, pois configuram outros registradores relevantes. A primeira habilita a conversão e a segunda retorna o resultado da mesma. Caso haja interesse em saber mais sobre essas funções, basta busca-las na ajuda do mikroC dentro da aba índice. Ou caso o leitor queira saber mais sobre os registradores que foram configurados por elas, abra a folha de dados do modelo utilizado e busque por “ANALOG-TO-DIGITAL CONVERTER”, lá estão presentes todos os registradores que devem ter seus bits configurados para a conversão.
- Dentro do laço whille(1), atribuímos o valor resultante da conversão à nossa variável ‘conv’ e em seguida fazemos as devidas associações para que atendamos ao que foi solicitado no enunciado. Você deve estar se perguntando o que significam esses valores nos intervalos dos ‘ifs’. Bom, quando um conversor de 10 bits de resolução realiza a conversão ele retorna um valor dentro de um intervalo de 0 a 1024 se atribuído à uma variável do tipo inteiro, tomando como base as tensões positivas e negativas que foram informadas como referência através dos bits VCFG1 e VCFG0 do registrador ADCON1. Portanto, caberá ao programador correlacionar esse valor obtido, com a tensão que realmente está no pino analógico, fazendo para isso uma simples regra de três. Suponhamos que a tensão de referência positiva Vref+ = 5V e que a tensão de referência negativa Vref- = 0V e que temos uma leitura de 700 informada pelo conversor, o próximo passo é “traduzirmos” 700 dentro do intervalo de 0V a 5V:

  1024 ------- 5V
  700 ------- X = 3,42V

Exercício Proposto – Utilizando o modelo PIC18F4550, faça a leitura de duas entradas analógicas, AN0 e AN1, como se as mesmas estivessem ligadas a dois potenciômetros. Em seguida, compare os dois valores e se o valor da segunda entrada for maior que o da primeira, acenda um LED vermelho, se for igual acenda um LED amarelo e se for menor acenda um LED verde. Use as tensões de alimentação do PIC como referência para a conversão.

4

DISPLAY LCD

Muitas vezes é necessário incluirmos em nosso sistema recursos que possibilitem a interface homem máquina. Em algumas aplicações é necessário que um operador/usuário supervisione o andamento de um processo controlado por um microcontrolador. Para esses casos pode-se incluir buzzer's, LED's, displays de sete segmentos ou até mesmo displays LCD, capazes de representar números e letras de maneira compacta e com fácil implementação.

Nessa sessão aprenderemos através de um exemplo, como utilizar um display LCD LM016L que possui uma matriz de representação de 2x16, ou seja, duas linhas capazes de representar 16 caracteres alfanuméricos.

Esse exemplo se baseará no exemplo 2, onde utilizaremos o display LCD para exibirmos as leituras do conversor (0-1024) e a respectiva tensão associada (0V-5V). Para isso abriremos a ajuda do mikroC PRO® e na aba índice buscaremos por LCD. Abrindo o resultado da busca encontraremos o seguinte trecho onde mostra um exemplo de associação entre variáveis globais necessárias para se utilizar a função lcd_init() e os pinos do microcontrolador:

```
// Lcd pinout settings
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D7 at RB3_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D4 at  RB0_bit;

// Pin direction
sbit  LCD_RS_Direction  at  TRISB4_bit;
sbit  LCD_EN_Direction  at  TRISB5_bit;
sbit  LCD_D7_Direction  at  TRISB3_bit;
sbit  LCD_D6_Direction  at  TRISB2_bit;
sbit  LCD_D5_Direction  at  TRISB1_bit;
sbit LCD_D4_Direction at  TRISB0_bit;
```

A função lcd_init(), que exige a definição acima antes do seu uso, faz parte de uma biblioteca específica presente no mikroC e é responsável por inicializar o display. Também na

mesma janela onde encontramos essa associação, podemos encontrar outros comandos para o LCD, além de exemplos de aplicação.

Exemplo 3 – Display LCD

```
#define verde  portd.rd0
#define amarelo  portd.rd1
#define vermelho  portd.rd2

sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at  RB3_bit;

sbit  LCD_RS_Direction  at  TRISB4_bit;
sbit  LCD_EN_Direction  at  TRISB5_bit;
sbit  LCD_D4_Direction  at  TRISB0_bit;
sbit  LCD_D5_Direction  at  TRISB1_bit;
sbit  LCD_D6_Direction  at  TRISB2_bit;
sbit LCD_D7_Direction at  TRISB3_bit;


float conv=0, tensao=0;
char esc[10],  esc2[10];

void main() {

TRISD=0;
TRISA=1;
TRISB=0;
PORTB=0;
PORTD=0;
ADCON1.VCFG1=0;
ADCON1.VCFG0=0;
ADCON1.PCFG3=1;
ADCON1.PCFG2=1;
ADCON1.PCFG1=1;
ADCON1.PCFG0=0;
ADC_init();
Lcd_init();
lcd_cmd(_lcd_cursor_off);

while(1){

conv=adc_read(0);
tensao=((conv*5)/1024);
floattostr(tensao,esc);
inttostr(conv,esc2);
lcd_out(1,1,"Tensao:");
```

```
lcd_out(1,8,esc);
lcd_out(2,1,"CONVERSOR:");
lcd_out(2,10,esc2);

if (conv>800) {
vermelho=0;
amarelo=0;
verde=1;
}if (conv<800 && conv>400){
verde=0;
vermelho=0;
amarelo=1;
}if (conv<400){
amarelo=0;
verde=0;
vermelho=1;
}}}
```

Comentários:

- Trata-se do mesmo software do exemplo anterior, exceto pela adição das associações entre as variáveis globais da função lcd_init() e os pinos do barramento B do microcontrolador, e de outros ajustes necessários para o funcionamento do LCD.
- Mudamos o tipo da variável 'conv' para float, para que possamos escrever os dígitos decimais no LCD e adicionamos outra varável chamada 'tensao' que receberá o valor convertido em tensão. Também adicionamos duas novas varáveis do tipo vetor char (esc e esc2), pois como o display somente escreve itens que sejam de tal tipo, é necessário fazer a conversação dos valores float para char antes de solicitar a escrita.
- Dentro da rotina principal, configuramos o barramento B como saída para que o mesmo se conecte ao LCD e atribuímos a ele nível lógico baixo, inicialmente.
- Chamamos a função Lcd_Init() para inicializar o LCD e desligamos o cursor do mesmo através do comando lcd_cmd(_lcd_cursor_off).
- Dentro do laço while(1), após 'conv' receber o valor da conversão do ADC, fazemos a regra de três para sabermos qual o valor real da tensão que se encontra na porta analógica AN0.
- Convertemos esse valor float de tensão para uma string que será armazenada no vetor char esc e em seguida fazemos outra conversão, de inteiro para string, de conv para o vetor char esc2.

- Através do comando lcd_out(n,m,string), escrevemos os rótulos e os vetores char nas linhas e colunas do LCD especificadas pelos números contidos no comando.

Exercício Proposto – Escreva os valores de tensão, identificando-os, nas duas entradas analógicas do exercício da sessão anterior, em um display LCD.

5

TIMERS

Até então vimos apenas uma maneira, não muito correta, de mensurarmos o tempo, a instrução delay. Para pequenos intervalos o comando acima é muito útil e não traz grandes problemas em seu uso, mas devemos lembrar que quando o utilizamos, estamos pausando a execução de nosso programa e gerando atrasos nos demais processos que devem ser executados. Dependendo do projeto, esses atrasos podem acarretar em perdas de informação em algum tipo leitura de dados por exemplo, ocasionando o mal funcionamento de todo o sistema.

Nos microcontroladores existe uma ferramenta que funciona exatamente como um temporizador e que além de ser precisa, não interfere no restante da execução do código e pode ser configurada de diversas formas para que se tenha o intervalo de tempo desejado. Trata-se do TIMER, existente em alguns modelos de microcontroladores em até 4 módulos (TIMER0, TIMER1, TIMER2 e TIMER3) que se diferenciam por possuírem características específicas como a resolução (8bits ou 16 bits), a existência de registradores para configurações especiais, entre outras.

Os TIMERs além de temporizadores, também são contadores de pulsos desde que sejam devidamente configurados para tal fim em seus registradores, para saber mais a respeito dessa segunda função, consulte a folha de dados do modelo utilizado.

Nesse material, veremos os módulos TIMER0, TIMER1 e TIMER2 e seus registradores, utilizando o PIC16F628A das sessões iniciais. O TIMER3 possui suas configurações muito parecidas com as configurações do TIMER1, por isso não o veremos aqui.

Todos os módulos possuem um prescaler configurável, que a grosso modo é uma espécie de divisor da frequência de entrada do TIMER, utilizado para aumentar o intervalo de contagem. O módulo TIMER2 possui uma peculiaridade, pois além de possuir o prescaler, também possui um postscaler que recebe o número de vezes em que o temporizador deverá atingir seu valor máximo para que uma contagem seja considerada completa, funcionando como um multiplicador de capacidade.

Seguem abaixo algumas características básicas de cada TIMER e como é feita sua configuração:

TIMER0

- Temporizador de 8 bits
- Prescaler de 8 bits
- Clock interno ou externo
- Interrupção configurável com o estouro
- Modo contador de pulsos ajustável para borda de Subida/Descida

Cálculo do tempo de estouro

$$t_{estouro} = \frac{1}{\frac{f_{osc}}{4}} * (\text{Prescaler}) * (255 - \text{Valor inicial da contagem})$$

Registradores/Bits associados ao TIMER0

TMR0 – Responsável por armazenar o valor da contagem

OPTION

- BIT5 (T0CS) - 1 = Fonte externa de clock (RA4) / 0 = Instrução interna de clock
- BIT4 (T0SE) - 1 = Incrementar em borda de subida / 0 = Em borda de descida
- BIT3 (PSA) - 1 = Associa o prescaler ao WDT / 0 = Associa ao TIMER0
- BIT2 (PS2), BIT1 (PS1) e BIT0 (PS0) – Seleção do prescaler, checar tabela na folha de dados.

INTCON – *Configurado apenas quando se deseja utilizar interrupções.*

- BIT7 (GIE) - 1 = Habilita interrupções globais / 0 = Desabilita todas interrupções
- BIT5 (T0IE) - 1 = Habilita a interrupção por TIMER0 / 0 = Desabilita
- BIT2 (T0IF) - 1 = Sinaliza que TMR0 estourou / 0 = TMR0 não estourou

TIMER1

- Temporizador de 16 bits utilizando dois registradores de 8 bits (TMR1H e TMR1L)
- Prescaler de 3 bits
- Clock interno ou externo
- Interrupção configurável com o estouro
- Modo contador síncrono/assíncrono

Cálculo do tempo de estouro

$$t_{estouro} = \frac{1}{\frac{f_{Osc}}{4}} * (\text{Prescaler}) * (65535 - \text{Valor inicial da contagem})$$

Registradores/Bits associados ao TIMER1

TMR1L – Responsável por armazenar os 8 bits menos significativos da contagem

TMR1H – Responsável por armazenar os 8 bits mais significativos da contagem

T1CON

- BIT7 e BIT6 – Não são usados e serão lidos como "0"
- BIT5 (T1CKPS1) e BIT4 (T1CKPS0) - Seleção do prescaler, deve-se checar a tabela na folha de dados.
- BIT3 (T1OSCEN) - 1 = Oscilador TIMER1 habilitado / 0 = Oscilador desabilitado
- BIT2 (T1SYNC) - 1 = Não sincronização com clock externo / 0 = Sincronização
- BIT1 (TMR1CS) - 1 = Clock externo selecionado / 0 = Clock interno selecionado
- BIT0 (TMR1ON) - 1 = Habilita o TIMER1 / 0 = Desabilita

IINTCON – *Configurado apenas quando se deseja utilizar interrupções.*

- BIT7 (GIE) - 1 = Habilita interrupções globais / 0 = Desabilita todas interrupções
- BIT6 (PEIE) - 1 = Habilita interrupções de periféricos / 0 = Desabilita

PIR1 – *Configurado apenas quando se deseja utilizar interrupções.*

- BIT0 (TMR1IF) – 1 = Sinaliza que TMR1 estourou / 0 = TMR1 não estourou

PIE1 – *Configurado apenas quando se deseja utilizar interrupções.*

- BIT0 (TMR1IE) – 1 = Habilita interrupção por TIMER1 / 0 = Desabilita

TIMER2

- Temporizador de 8 bits
- Prescaler de 1:1, 1:4 e 1:16
- Postscaler de 4 bits
- Usado como base de contagem no modo PWM do módulo CCP

Cálculo do tempo de estouro

$$t_{estouro} = \frac{1}{\frac{f_{osc}}{4}} * (\text{Prescaler}) * (\text{Postscaler}) * (\text{PR2})$$

Registradores/Bits associados ao TIMER2

TMR2 – Responsável por armazenar o valor da contagem

T2CON

- BIT7 – Não utilizado, lido como "0"
- BIT6 (TOUTPS3), BIT5 (TOUTPS2), BIT4 (TOUTPS1), BIT3 (TOUTPS0) - Seleção do postscaler de 4 bits, favor checar a tabela na folha de dados.
- BIT2 (TMR2ON) – 1 = Habilita o TIMER2 / 0 = Desabilita
- BIT1 (T2CKPS1) e BIT0 (T2CKPS0) – Seleção do prescaler, favor checar a tabela na folha de dados.

PR2 – Registrador de 8 bits que seleciona o período do TIMER2

INTCON – *Configurado apenas quando se deseja utilizar interrupções.*

- BIT7 (GIE) - 1 = Habilita interrupções globais / 0 = Desabilita todas interrupções
- BIT6 (PEIE) - 1 = Habilita interrupções de periféricos / 0 = Desabilita

PIR1 – *Configurado apenas quando se deseja utilizar interrupções.*

- BIT1 (TMR2IF) – 1 = Sinaliza que TMR2 estourou / 0 = TMR2 não estourou

PIE1 – *Configurado apenas quando se deseja utilizar interrupções.*

- BIT1 (TMR2IE) – 1 = Habilita interrupção por TIMER2 / 0 = Desabilita

Exemplo 4 – TIMER

TIMER0 – Se tivermos conectado ao nosso PIC um cristal oscilador de 20 MHz e o TIMER0 seja programado como temporizador, com prescaler de 1:4 e contagem inicial em TMR0L=0. Calcule o tempo de estouro do timer para este caso.

Resposta: $t_{estouro} = \frac{1}{\frac{20MHz}{4}} * 4 * (256 - 0) \approx 204us$

TIMER1 – Deseja-se um tempo de estouro de aproximadamente 26ms e para isso pretende-se utilizar o TIMER1 no modo temporizador. Sabe-se que o projeto utiliza um cristal oscilador de 20 MHz. Calcule qual prescaler deverá ser utilizado, considerando que a contagem iniciará em zero.

Resposta: $26ms = 2 * 10^{-7} * (Prescaler) * (65535 - 0)$

$Prescaler = 1{,}98 \approx 2 \therefore 0$ prescaler será de $1:2$

TIMER2 – Se tivermos conectado ao nosso PIC um cristal de 20 MHz e utilizarmos o TIMER2 com um prescaler de 1:1, postscaler de 1:16 e PR2 = 4. Calcule o tempo de estouro.

Resposta: $t_{estouro} = \frac{1}{\frac{20MHz}{4}} * 1 * 16 * 4 = 12{,}8us$

Exercício Proposto – Implemente um cronômetro no mikroC PRO® e exiba a contagem em um display LCD. Não se esqueça de configurar os devidos registradores e de utilizar o PIC16F628A. Escolha o módulo TIMER que achar mais adequado. Lembrando que para esse caso, não é necessário utilizar os registradores referentes às interrupções.

6

INTERRUPÇÕES

Em determinados projetos se faz necessário um tratamento especial à ocorrência de um determinado evento, onde a rotina principal deve ser pausada para que se execute uma rotina específica referente ao ocorrido. Em seguida o programa deve retornar à execução principal no exato trecho onde havia sido pausado antes de ser interrompido pela execução de tal rotina. Para esses casos, existem as chamadas interrupções, que se distinguem pelo tipo de evento que às habilitam, como por exemplo, Interrupções por TIMERs, Interrupção externa, Interrupção da EEPROM, Interrupção por mudança de estado e muitas outras. Os tipos de interrupção variam de modelo para modelo de microcontrolador e atendem necessidades específicas de projeto.

Dois conceitos importantes sobre as interrupções devem ser explicados aqui antes que comecemos o nosso exemplo. Um deles é o conceito de flag (bandeira), que se trata de um bit que sinaliza a ocorrência de uma determinada interrupção. Cada tipo de interrupção possui seu bit de flag, que automaticamente recebe o valor “1” quando a interrupção ocorre e deve ser zerado pelo programador ao fim da rotina caso se deseje que a mesma ocorra novamente. Geralmente, esse bit fica localizado em um dos registradores responsáveis por interrupções. O outro conceito, é o de rotina de interrupção. Nela colocamos toda a lógica que desejamos que seja executada quando a interrupção ocorrer. Para declará-la no código, criamos uma função void com o nome **interrupt** (nome dedicado às interrupções). Isso é suficiente para que o compilador entenda que dentro daquela função estarão os comandos que deverão ser executados com a chamada da respectiva interrupção.

Trataremos apenas das interrupções por TIMERs, como forma de iniciarmos os nossos estudos em interrupções. Que fique a cargo da curiosidade do leitor, estudar o funcionamento e a implementação de outros tipos de interrupção.

No exemplo dessa sessão, faremos com que um LED ligado ao pino RB0 pisque a cada 0,5 segundo, utilizando a interrupção por TIMER1 do PIC16F628A com um oscilador de 4MHz. Primeiramente precisamos realizar o cálculo do prescaler necessário para conseguirmos alcançar

o tempo de estouro solicitado e em seguida configurar os registradores* do TIMER1 e de sua interrupção.

* Todos os registradores a serem configurados estão disponíveis na sessão anterior.

Exemplo 5 – Interrupção por TIMER1

$$0{,}5s = \frac{1}{\frac{4\text{MHz}}{4}} * (\text{Prescaler}) * (65535 - 0)$$

$$\text{Prescaler} \approx 1:8$$

```
#define LED portb.rb0

void interrupt(){
LED=1;
delay_ms(50);
LED=0;
PIR1.TMR1IF=0;
}

void main() {
TRISB=0;
PORTB=0;
TMR1L=0;
TMR1H=0;
T1CON=0b00110001;
INTCON.GIE=1;
INTCON.PEIE=1;
PIR1.TMR1IF=0;
PIE1.TMR1IE=1;
while(1){}
}
```

Comentários:

- Atribuímos a tag LED ao pino RB0
- Em seguida declaramos a rotina dedicada à interrupção e dentro dela colocamos o que desejamos que seja feito quando a interrupção for chamada através do estouro do TIMER1, ou seja, quando a flag presente no bit PIR1.TMR1IF for igual a "1".
- Ao fim da rotina atribuímos à flag (PIR1.TMR1IF) o valor "0", para que a interrupção possa ocorrer novamente, com um novo estouro de temporização.
- Dentro da rotina principal, configuramos os registradores do TIMER1 e de sua interrupção, lembrando que o prescaler, calculado para atender às exigências do projeto, foi de 1:8.

- E ao fim, acrescentamos o laço condicional while(1).
- Nota-se que a aplicação do nosso exemplo não é muito útil em um projeto real, mas serve de base para entendermos o funcionamento das interrupções. No nosso caso, o LED piscará toda vez que a interrupção for chamada com o estouro do TIMER1.

Exercício Proposto – Em uma fábrica, é necessário que um motor inverta o sentido de rotação a cada dois segundos. Seu chefe pediu para que você implemente isso e exigiu o uso do TIMER2 e de sua respectiva interrupção. Você tem disponível um PIC16F628A com um cristal oscilador de 4MHz.

7

PWM

Podemos obter apenas dois níveis de tensão de uma saída digital de um microcontrolador, nível lógico alto (cerca de +5V) ou nível lógico baixo (aproximadamente 0V), seguindo as recomendações de alimentação do nosso CI. Mas o que fazer, caso necessitemos de uma tensão intermediária em nosso projeto? Há um recurso chamado modo PWM, que consiste na modulação da tensão por largura de pulso. Nos PICs que possuem esse recurso, o mesmo se encontra no módulo CCP (Captura, Comparação e PWM) e possui pino(s) específico(s) para sua utilização.

A modulação por largura de pulso funciona como um “liga-desliga” em um período fixo, onde modulamos a porcentagem deste período correspondente ao estado “ligado”. Imagine que você esteja ligando e desligando o interruptor de uma das luzes de sua casa em uma frequência fixa e mantendo metade desse período ligado e metade desligado. Se você fizesse isso muito rápido, teríamos como resultado uma iluminação com apenas metade da capacidade total que a lâmpada fornece. Da mesma forma, se você mantivesse a lâmpada acesa por apenas um terço do período total, teríamos como resultado 33% da iluminação da lâmpada quando totalmente acesa. Claro que esse exemplo é fictício e não esperamos que de fato o leitor teste isso, portanto no nosso exemplo substituiremos a lâmpada por um LED e seu dedo pelo PWM do microcontrolador. A tal porcentagem do período correspondente ao tempo “ligado” se chama duty cycle (ciclo de trabalho) que se trata da razão entre a largura do pulso (estado ativo/ligado) sobre a largura total do período.

No PIC a resolução do nosso PWM é de 8 bits, ou seja, podemos representar tensões de saída entre 0V e 5V, determinando valores de duty cycle dentro de um intervalo de 0 a 255. Por exemplo, se quisermos uma tensão de saída de aproximadamente 2,5V, basta definirmos o duty cycle como 128.

No exemplo, utilizaremos o PIC16F628A, onde ligaremos um motor ao pino RB3 (CCP1 – Pino de PWM) e a cada 2 segundos acrescentaremos 50 ao duty que inicialmente estará em 0,

aumentando gradativamente a velocidade do motor. Para essa temporização utilizaremos interrupção por TIMER1.

Exemplo 6 – PWM

```
#define MOTOR portb.rb3

int i=0, duty=0;

void interrupt(){
i++;
if (i==4){
duty=duty+50;
if (duty>250)  duty=0;
PWM1_SET_DUTY(duty); //Atribui o valor de duty ao PWM
i=0;
}TMR1IF_bit=0;
}

void main() {
TRISB=0;
TMR1L=0;
TMR1H=0;
T1CON=0b00110101;
INTCON.GIE=1;
INTCON.PEIE=1;
TMR1IF_bit=1;
TMR1IE_bit=1;
PWM1_INIT(2000); //Configura o PWM com frequência de 2MHz
PWM1_START();     //Inicia o  PWM
while(1){}
}
```

Comentários:

- O “define” foi utilizado apenas para que saibamos onde está nosso motor.
- Declaramos duas variáveis do tipo inteiro, a primeira será um contador e a segunda receberá o incremento do duty cycle.
- Dentro da nossa rotina de interrupção, esse acréscimo ao “i” foi necessário, pois o estouro do nosso TIMER1 se dará em aproximadamente 0,5 segundos utilizando um prescaler de 1:8 e um oscilador de 4MHz. Com isso para que o duty fosse alterado a cada 2 segundos, seria necessário entrar quatro vezes na rotina de interrupção para que um acréscimo fosse feito. Também limitamos o duty à 250 zerando-o após atingir esse valor. Em

seguida, alteramos o duty para o novo duty calculado, zeramos o “i”, saímos da condição e zeramos o flag da interrupção.

- No restante do código encontram-se os ajustes dos registradores do TIMER1, da Interrupção e os comandos do PWM presentes em uma biblioteca específica do mikroC PRO®.

Exercício Proposto – Utilize um potenciômetro para controlar a velocidade de um motor e mostre o duty cycle atual em um display LCD. Faça com que o motor inverta o sentido de rotação a cada 10 segundos.


TROIA
Equipe de Robótica