Módulo CCP

Roteiro Nº 06

Fundação Universidade Federal de Rondônia, Núcleo de Ciência e Tecnologia, Departamento de Engenharia - DEE
Curso de Bacharelado em Engenharia Elétrica - Disciplina de Sistemas Microprocessados
Elaboração: Ivan S. de Oliveira - Revisão: Prof. M.Sc. Ciro Egoavil
Laboratório de Sistemas Microprocessados

I. Introdução

O Microcontrolador PIC16F877A apresenta dois módulos CCPs, chamados CCP1 e CCP2. O módulo CCP pode funcionar em três modos distintos, que são os seguintes:

- **Captura** - mede o tempo entre dois eventos, ou seja, neste modo, os módulos CCP1 ou CCP2 efetua a medição de um sinal aplicado nas entradas CCP1 ou CCP2, respectivamente.
- **Comparação** - neste modo, o valor do registrador de 16 bits do módulo CCP é comparado com o valor do Timer1, e quando ocorre uma coincidência, pode-se setar os pinos CCP1 ou CCP2, resetar os pinos CPP1 ou CCP2, o TMR1, ou gerar uma interrupção CCPxIF. Uma das utilidades do modo de comparação é a geração de pulsos de largura controlada por software.
- **PWM (modulação por largura de pulso)** - controla a tensão entregue a uma determinada carga, modificando a largura do pulso de sinal, dentro de um período de tempo prefixado.

Cada um desses dois módulos é composto dos registradores CCPxCON, CCPRxH e CCPRxL, onde o 'x' pode ser 1 ou 2, dependendo do módulo.

II. Objetivos

Este trabalho tem por objetivo apresentar os módulos CCPs disponíveis no microcontrolador PIC16F877A em três possíveis aplicações: Periodímetro, controle de servomotor e controle de velocidade de motor DC.

III. Softwares Utilizados

- MPLAB;
- CCS C Compiler;
- PROTEUS Professional.

IV. Modo de Captura

O módulo de captura, como o próprio nome sugere, faz a captura entre bordas na entrada do CCP1 (pino RC2) ou CCP2 (pino RC1). Desta forma, dependendo do módulo CCP a qual se está utilizando, será necessário configurar o bit específico do registrador TRISC como entrada. A captura pode ocorrer de acordo como apresentado abaixo:

- Captura na borda de descida;
- Catura na borda de subida;
- Captura na borda de descida com prescaler 1:4;
- Captura na borda de descida com prescaler 1:16.

O resultado da captura é baseado no Timer1, onde as configurações deste devem ser realizadas no registrador T1CON. O prescaler funciona da seguinte forma: caso seja elecionado o prescaler 1:1, serão necessárias duas bordas de descida ou subida para que seja feita uma captura; caso seja 1:16, serão necessárias 16 bordas para que a captura seja concluída. No final da captura, caso a interrupção deste módulo esteja habilitada, o bit CCPxIF será setado e a interrupção ocorrerá.

Neste modo de operação, o Timer1 não deve estar configurado como contador assíncrono, pois neste estado o módulo pode não funcionar corretamente.

Toda vez que uma captura é realizada, este resultado é armazenado nos registradores de contagem CCPRxH:CCPRxL. Para determinar o tempo de período, precisa-se de duas amostragens, pois a segunda amostragem deve ser subtraída da primeira para que o período seja encontrado.

Com base no conhecimento supracitado, apresenta-se um projeto de um periodímetro simples utilizando um microcontrolador PIC programado em Linguagem C, cujo o esquema elétrico é mostrado na Figura 1.

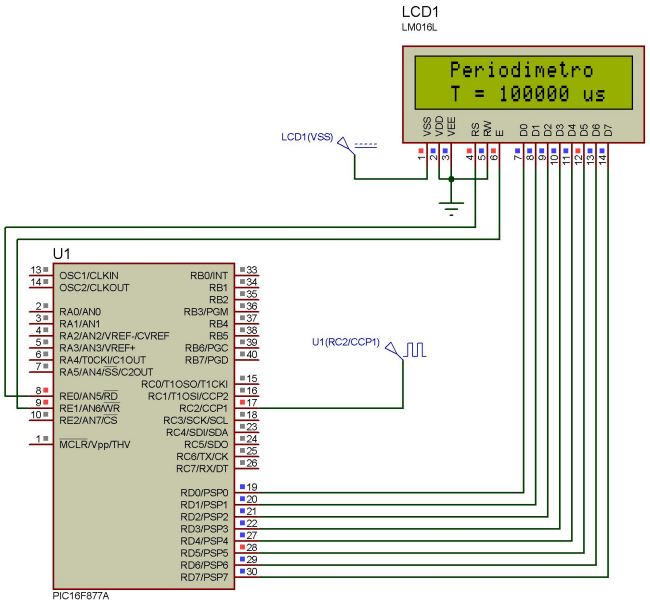

Figura 1. Esquema Elétrico do Circuito - Periodímetro

O periodímetro apresentado é capaz de medir um sinal de clock de 100 ms (10 Hz) à 1250 us (800 Hz) com boa precisão e apresentar o resultado em um display LCD.

O código, escrito em linguagem C, responsável por executar tal tarefa é apresentado abaixo:

```
#include <16f877a.h>
#fuses xt, nowdt, noprotect, put, brownout
#use delay(clock=4000000)

#define   RS        PIN_E0
#define   EN        PIN_E1

void  Inicializa(void);
void  Lcd_Inst(char dado);
void  Lcd_Dado(char dado);

int16  periodo1,periodo2;
int32  periodo;
boolean   flag=0;

void main() {
   Inicializa();
   Lcd_Inst(0x82);
   printf(lcd_dado, "Periodimetro");

   while(TRUE)
   {
      Lcd_Inst(0xC2);
      printf(lcd_dado, "T = %lu us\f",periodo);
   }
}

#int_ccp1
void ccp1_int()
{
   if(flag==0)
   {
      periodo1=CCP_1;
      flag =1;
   }
   else
   {
      periodo2=CCP_1;
      periodo = ((periodo2-periodo1)*2);
      Set_Timer1(0);
      flag = 0;
   }
}

void Inicializa(void)
{
   Setup_Timer_1(T1_INTERNAL|T1_DIV_BY_2);
   Setup_ccp1(CCP_CAPTURE_RE);
   Enable_Interrupts(GLOBAL);
   Enable_Interrupts(INT_CCP1 );

   Lcd_Inst(0x38);
   delay_ms(1);
   Lcd_Inst(0x38);
   Lcd_Inst(0x0C);
   Lcd_Inst(0x06);
   Lcd_Inst(0x01);
   delay_ms(1);

}
void  Lcd_Inst(char dado)
{
   Disable_Interrupts(GLOBAL);
   output_low(RS);
   output_d(dado);
   delay_cycles(2);
   output_high(EN);
   delay_ms(1);
   output_low(EN);
   Enable_Interrupts(GLOBAL);
}

void  Lcd_Dado(char dado)
{
   Disable_Interrupts(GLOBAL);
   output_high(RS);
   output_d(dado);
   delay_cycles(2);
   output_high(EN);
   delay_ms(1);
   output_low(EN);
   Enable_Interrupts(GLOBAL);
}
```

No programa, inicialmente são declaradas três variáveis:

```
int16  periodo1,periodo2;
int32  periodo;
boolean   flag=0;
```

As variáveis $periodo1$ e $periodo2$ são do tipo $int16$, ou seja, inteiras de 16 bits. A variável $periodo$ é do tipo $int32$, ou seja, inteira de 32 bits. Já a variável $flag$ é do tipo $boolean$, ou seja, possui um único bit, onde são representados apenas dois estados.

Na função $main()$ são enviadas as mensagens apresentadas no LCD e também realizada as configurações iniciais de funcionamento do PIC. Nesta, também, encontra-se um laço $while$, onde o resultado da captura é continuamente atualizado no LCD. As configurações iniciais do PIC são realizados na função $inicializa()$, que é reproduzida a seguir:

```
void Inicializa(void)
{
   Setup_Timer_1(T1_INTERNAL|T1_DIV_BY_2);
   Setup_ccp1(CCP_CAPTURE_RE);
   Enable_Interrupts(GLOBAL);
   Enable_Interrupts(INT_CCP1 );

   Lcd_Inst(0x38);
   delay_ms(1);
   Lcd_Inst(0x38);
   Lcd_Inst(0x0C);
   Lcd_Inst(0x06);
   Lcd_Inst(0x01);
   delay_ms(1);
}
```

Nesta função, é realizada a inicialização do display LCD e, também, as seguintes configurações iniciais do PIC:

- $Setup\_Timer\_1(T1\_INTERNAL|T1\_DIV\_BY\_2)$- ajusta o Timer1 para operar com clock interno (Fosc/4) e com prescaler de 1:2;
- $Setup\_ccp1(CCP\_CAPTURE\_RE)$ - configura o módulo CCP1 no modo de captura na borda de subida;
- $Enable\_Interrupts(GLOBAL)$ - habilita as interrupções;

- $Enable\_Interrupts(INT\_CCP1)$ - habilita a interrupção do módulo CCP1.

O resultado da captura é tratado dentro da função de interrupção do modo de captura:

```
#int_ccp1
void ccp1_int()
{
   if(flag==0)
   {
      periodo1=CCP_1;
      flag =1;
   }
   else
   {
      periodo2=CCP_1;
      periodo = ((periodo2-periodo1)*2);
      Set_Timer1(0);
      flag = 0;
   }
}
```

Na função de tratamento da interrupção do módulo CCP1, o realizado uma primeira captura da borda de subida do sinal e o tempo decorrido para esta é armazenado na variável $periodo1$, então a variável $flag$ é setada para sinalizar que a primeira captura foi realizada. Na próxima interrupção que ocorrer, o tempo será armazenado na variável $periodo2$ e, então, na variável $periodo$ é armazenado o valor da diferença entre $periodo1$ e $periodo2$ multiplicado por 2 devido ao prescaler estar em 1:2, e este valor corresponde ao período total do sinal aplicado.

V. Relatório - Parte I

Altere o programa anteriormente apresentado, transformando o periodímetro em um frequencímetro capaz de efetuar a medida das frequências de 10 Hz à 800 Hz.

VI. Modo de Comparação

No modo de comparação, a contagem do Timer1 é constantemente comparada com o valor armazenado no par de registradores CCPRxL e CCPRxH. No caso de coincidência, um dos seguintes eventos é disparado:

- Inverte a saída no evento de comparação;
- Inicializa o pino CCPx em nível baixo e quando a comparação ocorrer, seta o pino CCPx;
- Inicializa o pino CCPx em nível alto e quando a comparação ocorrer, coloca o pino CCPx em nível baixo;
- Seta o bit de interrupção CCPxIF e não altera o pino CCPx;
- Ocorre o evento especial do trigger e o bit CCPxIF é setado.

No evento especial, o Timer1 será resetado e uma conversão A/D será iniciada, se esta estiver habilitada.

Uma das possíveis aplicações do módulo de comparação, como já mencionado, é a geração de pulsos de largura controlada por software. Portanto, nesta parte do roteiro será apresentado o controle de servomotores com o módulo de comparação.

O servomotor é um mecanismo eletromecânico utilizado em diversas aplicações, as mais comuns sendo em antenas parabólicas e aeromodelos, pois a precisão de controle que este mecanismo atinge é considerada boa.

Esta precisão é devida à largura dos pulsos e à duração entre esses pulsos aplicados à sua entrada. Se o pulso de entrada tiver duração de 1,0 ms o rotor do servomotor irá girar para a posição de -90º, se o pulso tiver duração de 1,5 ms o rotor do servomotor irá permanecer na posição de 0º e se o pulso tiver duração de 2,0 ms o rotor do servomotor irá gira para a posição de 90º, conforme Figura 2. Para obter estes ângulos, os pulsos devem ser enviados ao servomotor continuamente em intervalos de cerca de 18 ms.

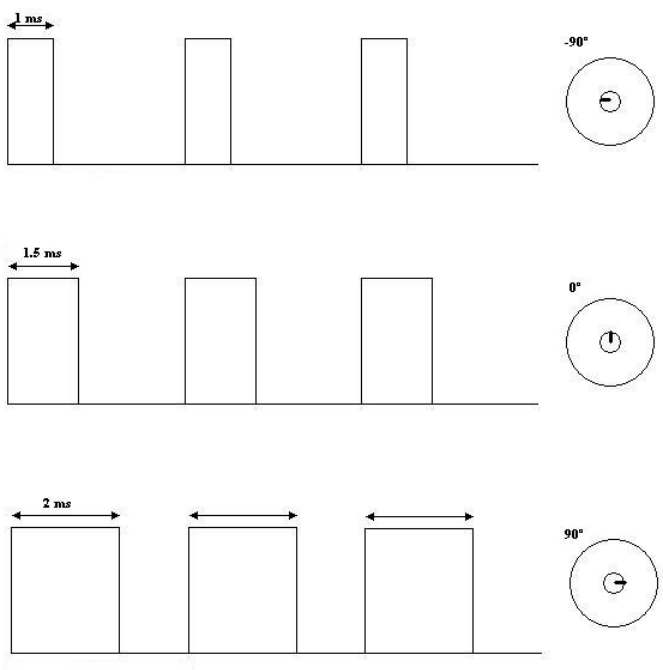

Figura 2. Funcionamento do Servomotor

O esquema elétrico para simulação no PROTEUS é o da Figura 3. Como pode ser observado, um botão é conectado ao PIC (pino RB1), e caso este esteja pressionado, o rotor do servomotor irá girar para a posição de 90º e caso não esteja pressionado, o rotor do servomotor irá para a posição 0º.

O programa escrito em linguagem C que irá executar a operação descrita é apresentado a seguir:

```
#include <16f877a.h>
#fuses xt, nowdt, noprotect, put, brownout
#use delay(clock=4000000)

#define   BOTAO     PIN_B1

void  Inicializa(void);
long Nivel_Baixo = 18000;
long Nivel_Alto;
boolean flag=1;

void main() {
```

```
   Inicializa();
   while(TRUE){
      if(!input(BOTAO))
      {
         Nivel_Alto = 2000;
      }
      else
      {
         Nivel_Alto = 1500;
      }
   }
}

#int_ccp1
void ccp1_int()
{
  if(flag==0)
   {
      output_high(PIN_C1);
      CCP_1 = Nivel_Alto;
      flag =1;
   }
   else
   {
      output_low(PIN_C1);
      CCP_1 = Nivel_Baixo;
      flag = 0;
   }
}

void Inicializa(void)
{
   Setup_Timer_1(T1_INTERNAL);
   Setup_ccp1(CCP_COMPARE_RESET_TIMER);
   Enable_Interrupts(GLOBAL);
   Enable_Interrupts(INT_CCP1);
   Port_b_Pullups(TRUE);
   CCP_1 = 0;
}
```

O código é iniciado com as instruções reproduzidas abaixo:

```
#include <16f877a.h>
#fuses xt, nowdt, noprotect, put, brownout
#use delay(clock=4000000)

#define   BOTAO     PIN_B1

void  Inicializa(void);
long Nivel_Baixo = 18000;
long Nivel_Alto;
boolean flag=1;
```

Neste trecho, é realizada a inclusão do ficheiro com as características do PIC16F877A, microcontrolador utilizado no projeto, é declarada a função $Inicializa(void)$ e o pino RB1 é denominado de $BOTAO$.

A variável $Nivel\_Baixo$ é declara como do tipo $long$, 16 bits, com valor inicial de 18000, que corresponde ao intervalo de 18 ms entre os pulsos enviados ao servomotor. A variável $Nivel\_Alto$ é responsável por armazenar o intervalo de tempo entre 1 ms e 2 ms em que o sinal permanecerá em nível alto. A variável $flag$ tem a função de indicar se o sinal que está sendo enviado ao servomotor naquele momento está em nível baixo ($flag$ = 0) ou em nível alto ($flag$ = 1).

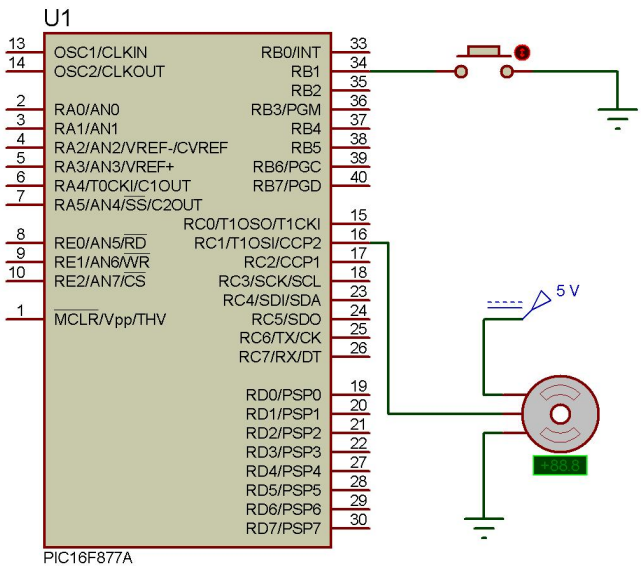

Figura 3. Esquema Elétrico do Circuito - Controle de Servomotor

Na função $main$ é executada a função $Inicializa()$ responsável por realizar a configuração inicial do PIC. Na função $main$, o comando $while(true)$ cria um laço onde é realizado o teste do botão conectado ao pino RB1. Caso o botão não esteja pressionado o valor 1500 (1,5 ms) será armazenado na variável $Nivel\_Alto$ e caso o botão esteja pressionado o valor 2000 (2 ms) será armazenado na variável $Nivel\_Alto$.

```
void main() {
   Inicializa();
   while(TRUE){
      if(!input(BOTAO))
      {
         Nivel_Alto = 2000;
      }
      else
      {
         Nivel_Alto = 1500;
      }
   }
}
```

A execução permanecerá testando o botão até que corra uma interrupção do módulo CCP. Ocorrendo a interrupção a execução será desviada para a função que irá tratá-la.

```
#int_ccp1
void ccp1_int()
{
  if(flag==0)
   {
      output_high(PIN_C1);
      CCP_1 = Nivel_Alto;
      flag =1;
   }
```

```
    else
    {
       output_low(PIN_C1);
       CCP_1 = Nivel_Baixo;
       flag = 0;
    }
}
```

Na função de tratamento da interrupção, $ccp1\_int()$, o valor de $flag$ é testado, caso este esteja zerado o sinal de saída no pino RC1 é posto em nível alto e o valor da variável $Nivel\_Alto$ é enviado a CCP_1 e esta determinará o tempo em que o sinal irá permanecar neste estado. Caso $flag$ esteja setado, o pino RC1 é posto em nível baixo e permanecerá neste estado pelo tempo determinado por $Nivel\_Baixo$, que sempre será de 18 ms.

Na função $Inicializa()$,é realizada a configuração inicial do PIC através das instruções mostradas abaixo:

- $Setup\_Timer(T1\_INTERNAL)$ - ajusta o Timer1 para operar com clock interno (Fosc/4);
- $Setup\_ccp1(CCP\_COMPARE\_RESET\_TIMER)$ - configura o módulo CCP1 no modo de comparação com reset do Timer1.
- $Enable\_Interrupts(GLOBAL)$ - Habilita todas as interrupções;
- $Enable\_Interrupts(INTCCP1)$ - Habilita a interrupção do módulo CCP1;
- $Port\_b\_Pullups(TRUE)$ - Liga os pull-ups da PORTB.
- CCP_1 = 0 - inicializa a variável CCP_1 em zero.

A variável CCP_1 tem 16 bits e executa a mesma função dos registradores CCPRxH e CCPRxL.

```
void Inicializa(void)
{
   Setup_Timer_1(T1_INTERNAL);
   Setup_ccp1(CCP_COMPARE_RESET_TIMER);
   Enable_Interrupts(GLOBAL);
   Enable_Interrupts(INT_CCP1);
   Port_b_Pullups(TRUE);
   CCP_1 = 0;
}
```

VII. Relatório - Parte II

Altere o programa em linguagem C e o circuito apresentado na Figura 3 para que considere quatro botões. Cada botão deve girar o rotor do servomotor para uma posição específica, conforme listado a seguir:

- Botão1 - gira o rotor do servomotor para 90º;
- Botão2 - gira o rotor do servomotor para 45º;
- Botão3 - gira o rotor do servomotor para -45º;
- Botão4 - gira o rotor do servomotor para -90º;

Caso nenhum botão esteja pressionado, o rotor do servomotor deve permanecer na posição 0º.

VIII. Modo PWM

O modo PWM (Pulse Width Modulation) é de fato um dos mais importantes do módulo CCP. O que caracteriza este periférico é principalmente o fato de ele manter uma frequência constante na saída CCPx enquanto o ciclo ativo do sinal é alterado, permitindo desta forma controlar velocidade de motores ou a intensidade de uma lâmpada, por exemplo. A Figura 4 ilustra o funcionamento do módulo PWM.

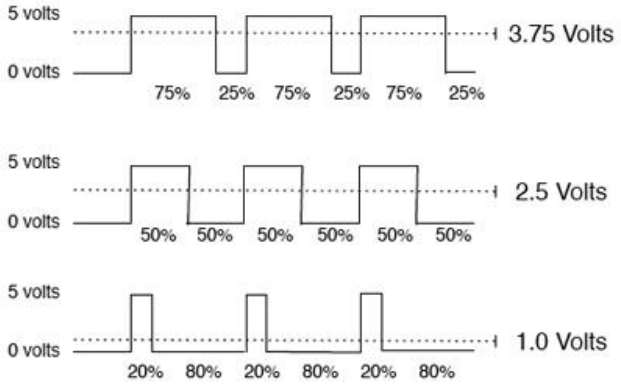

Figura 4. Modulação por Largura de Pulso - PWM

No primeiro sinal, durante o intervalo de um período prefixado no Timer2, 75 % do período fica ativo. Como a tensão máxima DC deste sistema é de 5 V, a tensão média entre a carga é de 3,75 V.

No segundo sinal, 50 % do período fica ativo, então a tensão média é de 2,5 V. E no terceiro, apenas 20 % do período fica ativo e a tensão média é de 1,0 V.

O princípio de funcionamento do módulo PWM no PIC é o seguinte:

A base de tempo do sinal é gerada pelo Timer2. Isto significa que o período (e consequentemente a frequência) do sinal é determinado pela programação do Timer2.

O período ativo é configurado através dos registradores CCPRXL e CCPXCON (bits 5 e 4). Como este PWM é de 10 bis, os 8 bits mais significativos ficam no registrador CCPRxL e os dois bits menos significativos nos bits 5 e 4 do registrador CCPxCON. No início da contagem do Timer2, o pino CCPx será levado a 1 se os registradores CCPRxL e CCPxCON (5:4) forem diferentes de 0. Quando o resultado da contagem do Timer2 for igual aos registradores de ciclo, o pino CCPx será levado a 0 e aguardar-se-á início de uma nova contagem.

Para configurar o período do PWM, é necessário duas informações:

- Frequência de funcionamento do PWM;
- Frequência do microcontrolador (Fosc).

Desta forma, pode-se utilizar a Equação 1 para encontrar o valor do registrador PR2, responsável por determinar o período do sinal gerado.

$$PR2 = \frac{Fosc}{Fpwm * 4 * Prescaler} - 1 \qquad (1)$$

onde

- $Fosc$ = frequência do oscilador do microcontrolador;

- $Fpwm$ = frequência do PWM;
- $Prescaler$ = prescaler do Timer2.

Na Equação 1, o valor de prescaler deve inicializar em 1 e, caso o resultado para PR2 seja maior que 255, o prescaler deve ser aumentado de forma qu obedeça o limite de oito bits do registrador PR2.

Como exemplo, caso deseja-se determinar o valor de PR2 para o PWM operando em 10 KHz para um microcontrolador com Fosc de 4 MHz e prescaler do Timer2 configurado em 1:1, pode-se utilizar a Equação 1 da seguinte forma:

$$PR2 = \frac{4 \times 10^6}{10 \times 10^3 * 4 * 1} - 1 = 99 \quad (2)$$

Apesar de o PWM ser de 10 bits, dependendo da frequência utilizada nem sempre será possível esta resolução. Para determinar a resolução máxima do PWM, utiliza-se a Equação 3

$$Rpwm = \frac{log(\frac{Fosc}{Fpwm})}{log(2)} \quad (3)$$

A resolução do PWM do exemplo apresentado será então:

$$Rpwm = \frac{log(\frac{4 \times 10^6}{10 \times 10^3})}{log(2)} = 8(bits) \quad (4)$$

Neste exemplo, este será o número máximo de resolução em bits e que somente a parte inteira é que tem validade, sendo truncado o resultado da resolução do log.

O período ativo do sinal é dado pela Equação 5:

$$T_{ativo} = \frac{(CCPRXL + CCPXCON(5:4)) * (Prescaler)}{Fosc} \quad (5)$$

O valor (CCPRxL + CCPxCON(5:4)) refere-se ao valor decimal correspondente aos 10 bits resultntes da associação dos 8 bits de CCPRxL mais os bits 5 e 4 do registrador CCPxCON.

A. Controle de Velocidade de Motor DC - Assembly

O controle de velocidade de um motor DC pode ser feito utilizando o CCP, no modo PWM, conectando ao pino de saída deste módulo um CI do tipo ULN2003A, cuja função é fornecer a corrente elétrica adequada ao funcionamento do motor. O esquema elétrico do circuito é apresentado na Figura 5.

Neste circuito, o microcontrolador PIC16F877A possui três botões conectados a PORTB. O primeiro botão, ligado ao pino RB1, quando pressionado, aciona o motor com período ativo de 100 %, ou seja, o motor é acionado a velocidade máxima. O segundo botão, pino RB2, aciona o motor com 60 % do período ativo, ou seja, o motor é acionado a 60 % de sua velocidade máxima. Já botão conectado ao pino RB3 aciona o motor a 20 % de sua velocidade máxima.

O código escrito em *Assembly*, gravado no PIC, com objetivo de executar as operações descritas, é apresentado a seguir:

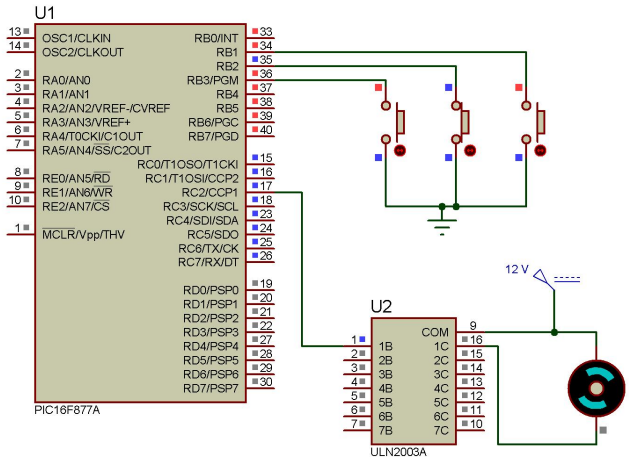

Figura 5. Esquema Elétrico - PWM

```
#include <P16F877A.INC>

 __CONFIG _CP_OFF&_PWRTE_OFF& _WDT_OFF & _XT_OSC

#DEFINE  BANCO0  BCF STATUS,RP0
#DEFINE  BANCO1  BSF STATUS,RP0

 #DEFINE BOTAO1 PORTB,1
 #DEFINE BOTAO2 PORTB,2
 #DEFINE BOTAO3 PORTB,3

  ORG 0X0000
  GOTO INICIO

  ORG 0X0004
  RETFIE


INICIO
  BANCO1
  MOVLW B'11111111'
  MOVWF TRISB

  MOVLW B'00000000'
  MOVWF TRISC

  MOVLW B'11111001'
  MOVWF PR2

  MOVLW B'00000000'
  MOVWF OPTION_REG


  MOVLW B'10000111'
  MOVWF ADCON1

  BANCO0
  MOVLW B'00000000'
  MOVWF INTCON

  MOVLW B'00000000'
  MOVWF T2CON
```

```
  MOVLW B'00001100'
  MOVWF CCP1CON

  MOVLW B'01000000'
  MOVWF ADCON0


  CLRF PORTB
  CLRF PORTC
  BSF T2CON,TMR2ON

LOOP
  BTFSC  BOTAO1
  GOTO  BOTAO_2
  BSF CCP1CON,4
  BSF CCP1CON,5
  MOVLW B'11111111'
  MOVWF CCPR1L
  GOTO LOOP

BOTAO_2
  BTFSC  BOTAO2
  GOTO   BOTAO_3
  BCF CCP1CON,4
  BSF CCP1CON,5
  MOVLW B'10011001'
  MOVWF CCPR1L
  GOTO LOOP

BOTAO_3
  BTFSC  BOTAO3
  GOTO   PARAR
  BSF CCP1CON,4
  BCF CCP1CON,5
  MOVLW B'00110011'
  MOVWF CCPR1L
  GOTO LOOP

PARAR
  BCF CCP1CON,4
  BCF CCP1CON,5
  MOVLW B'00000000'
  MOVWF CCPR1L
  GOTO LOOP

  END
```

Os registradores relevantes na configuração do módulo PWM são o T2CON que configura o Timer2 com prescaler de 1:1, o registrador PR2 que é carregado com o valor 249 para que o PWM funcione a frequência de 4 KHz e o registrador CCP1CON que configura o módulo CCP1 para operar no modo PWM.

O rótulo LOOP é a parte do código responsável pelo teste dos botões. Caso o botão conectado ao pino RB1 esteja pressionado, o valor 1024 (10 bits) é enviado ao registrador CCPR1L e aos bits 4 e 5 do registrador CCP1CON, desta forma o período ativo do sinal de saída será de 100 % e o motor irá operar a velocidade máxima.

Caso o botão conectado ao pino RB2 esteja pressionado, o valor 614 é transferido ao registrador CCPR1L e aos bits 4 e 5 do registrador CCP1CON, o que corresponde a um período ativo do sinal de saída de 60 %, pois 60 % de 1024 é aproximadamente 614, assim, o motor será acionado a 60 % de sua velocidade máxima.

Se o botão conectado ao pino RB3 estiver pressionado, o valor 205 é enviado ao registrador CCPR1L e as bits 4 e 5 do registrador CCP1CON, assim, o motor será acionado a 20 % da velocidade máxima, pois 20 % de 1024 (10 bits) é aproximadamente 205.

Caso nenhum botão esteja pressionado a execução será desviada para o rótulo $PARAR$, onde o valor 0 é enviado ao registrador CCPR1L e as bits 4 e 5 do registrador CCP1CON, e o sinal não terá período ativo, ocasionando o desligamento do motor.

B. Controle de Velocidade de Motor DC - Linguagem C

O controle de velocidade do motor DC, anteriormente realizado na Linguagem *Assembly*, pode ser implementado, também, na Linguagem C. No entanto, esta linguagem apresenta funções prontas que auxiliam a elaboração do programa.

Desta forma, a seguir é apresentado o código em Linguagem C que executa a mesmas operações apresentadas no código anterior, escrito em *Assembly*:

```
#include <16f877a.h>
#fuses xt, nowdt, noprotect, put, brownout
#use delay(clock=4000000)

#define   BOTAO1    PIN_B1
#define   BOTAO2    PIN_B2
#define   BOTAO3    PIN_B3

void  Inicializa(void);

void main() {
   Inicializa();
   while(TRUE){
      if(!input(BOTAO1))
      {
         Set_Pwm1_Duty(255);
      }
       else if(!input(BOTAO2))
      {
         Set_Pwm1_Duty(153);
      }
        else if(!input(BOTAO3))
      {
         Set_Pwm1_Duty(51);
      }
      else
      {
         Set_Pwm1_Duty(0);
      }
   }
}
void Inicializa(void)
{
   Setup_Timer_2(T2_DIV_BY_1,249,1);
   Setup_ccp1(CCP_PWM);
   Disable_Interrupts(GLOBAL);
   Port_b_pullups(TRUE);
}
```

Neste código a função $Inicializa()$ é responsável pelas configurações iniciais do PIC, conforme listado a seguir:

- $Setup\_Timer\_2(T2\_DIV\_BY\_1,\ 249,\ 1)$ - ajusta o Timer2 para operar com prescaler de 1:1, PR2 com valor de 249 (para frequência de operação do PWM de 4 KHz) e postscaler de 1:1;
- $Setup\_ccp1(CCP\_PWM)$ - configura o módulo CCP1 no modo PWM
- $Disable\_Interrupts(GLOBAL)$ - Desabilita as interrupções;
- $Port\_b\_Pullups(TRUE)$ - Liga os pull-ups da PORTB.

Na função $main()$, os botões conectados a PORTB são testados. Caso o botão ligado ao pino RB1 esteja pressionado, o comando $Set\_Pwm1\_Duty(255)$ ajusta o período ativo do sinal de saída para 100 %, isto é obtido através do argumento do comando, que neste caso, é 255. Na Linguagem C do CCS C Compile, a resolução máximo do PWM que pode ser configurada é de 8 bits, por isso o valor para um ciclo ativo de 100 % é de 255 (8 bits), diferentemente da Linguagem *Assembly* onde os 10 bits de resolução máxima do PWM podem ser alterados.

Caso o botão ligado ao pino RB2 esteja pressionado, o comando $Set\_Pwm1\_Duty(153)$ irá ajustar o ciclo ativo do sinal para 60 % do período, e o motor irá operar a 60 % de sua velocidade máxima.

Se o botão conectado ao pino RB3 estiver pressionado, o $Set\_Pwm1\_Duty(51)$ o motor irá operar a 20 % de sua velocidade máxima. Caso nenhum botão esteja pressionado, o motor será desligado.

IX. Relatório - Parte III

Altere os programas em linguagem C e em *Assembly*, para que acione dois motores de acordo com as seguintes condições:

- Botão1 - aciona motor 1 à velocidade máxima e motor 2 a 20 % da velocidade máxima;
- Botão2 - aciona motor 1 a 80 % da velocidade máxima e motor 2 a 60 % da velocidade máxima;
- Botão3 - aciona motor 1 a 60 % da velocidade máxima e motor 2 a 80 % da velocidade máxima;
- Botão4 - aciona motor 1 a 20 % da velocidade máxima e motor 2 à velocidade máxima;

Caso nenhum botão esteja pressionado, os motores deverão ser desligados.

Referências

[1] Souza, Vitor Amadeu, "Projetando com os microcontroladores da família PIC 18: Uma nova percepção", 1ª Ed., São Paulo: Ensino Profissional, 2007.

[2] Souza, David José de, "Desbravando o PIC: ampliado e atualizado para PIC 16F628A", 6ª Ed., São Paulo: Érica, 2003.

[3] Pereira, Fábio, "Micrcontroladores PIC: Técnicas Avançadas", 6ªEd., São Paulo: Érica, 2007.

[4] Apostila de Linguagem C para PIC16F877A com base no CCS - Cerne Tecnologia e Treinamento LTDA.

APÊNDICE

A - REGISTRADORES

- R/W - leitura/escrita
- R - somente leitura
- U - não implementado
- O valor após o traço é o valor inical, após reset.

1 - T2CON

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

Tabela I
REGISTRADOR T2CON

- **TOUTPS3:TOUTPS0** - seleção do postscaler do Timer2;

TOUTPS3	**TOUTPS2**	**TOUTPS1**	**TOUTPS0**	**Divisor**
0	0	0	0	1:1
0	0	0	1	1:2
0	0	1	0	1:3
0	0	1	1	1:4
0	1	0	0	1:5
0	1	0	1	1:6
0	1	1	0	1:7
0	1	1	1	1:8
1	0	0	0	1:9
1	0	0	1	1:10
1	0	1	0	1:11
1	0	1	1	1:12
1	1	0	0	1:13
1	1	0	1	1:14
1	1	1	0	1:15
1	1	1	1	1:16

- **TMR2ON** - controle habilitação/desabilitação Timer2;
  0 - Timer2 desativado, contagem parada;
  1 - Timer2 habilitado, contagem correndo;
- **T2CKPS1:T2CKPS0** - Seleção de prescaler do Timer2;

T2CKPS1	**T2CKPS0**	**Divisor**
0	0	1:1
0	1	1:4
1	0	1:16
1	1	1:16

2 - CCP1CON E CCP2CON

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
-	-	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0
U-0	U-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0

Tabela II
REGISTRADORES CCP1CON E CCP2CON

- **CCPxX e CCPxY** - Bits menos signitificativos do registrador de ciclo ativo quando em modo PWM. Quando o módulo CCP está operando em modo de captura ou comparação, esses bits não têm significado;
- **CCPxM3:CCPxM0** - seleção do modo de operação do módulo CCPx;

CCPxM3:CCPxM0	**Modo**
0000	Módulo CCPx desligado
0100	Captura a cada borda de descida
0101	Captura a cada borda de subida
0110	Captura a cada 4ª borda de subida
0111	Captura a cada 16ª borda de subida
1000	Comparação, seta pino CCPx em caso de coincidência (seta CCPxIF)
1001	Comparação, zera pino CCPx em caso de coincidência (seta CCPxIF)
1010	Comparação, em caso de coincidência gera interrupção (CCPxIF)
1011	Comparação, em caso de coincidência gera interrupção (CCPxIF) e reseta TMR1
11xx	Modo PWM