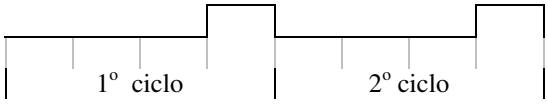
Implementando PWM por soft - um método simples.

Por Renie S. Marquet –reniemarquet.sites.com.br - versão 25.02.2005

O PWM (Pulse Width Modulation – Modulação por largura de pulso) consiste em controlar o período em que um sinal digital permanece em nível alto em relação ao período que o mesmo sinal permanece em nível baixo dentro de um ciclo.

Ex: sinal modulado com 25% de duty cicle



Com PWM é possível controlar a potência de um dispositivo de forma digital.

É importante escolher uma freqüência adequada para o PWM, pois se for utilizada uma freqüência muito abaixo ou muito acima do necessário os resultados não serão satisfatórios. Em dispositivos luminosos(lâmpadas, LEDs etc) o uso de freqüências abaixo do necessário geram um efeito cintilante e no pior dos casos, um pisca-pisca, nos motores tem-se a perda de força. Em freqüências acima do necessário pode-se ter um consumo desnecessário de potência tanto do dispositivo quanto do circuito oscilador, como também a perda do controle desejado devido a inércia dos componentes envolvidos.

Além da freqüência o usuário deve escolher a resolução que deseja/necessita trabalhar. A resolução em circuitos microcontrolados é dada em bits. Quanto maior a resolução, maior será a quantidade de variações da potência entre o mínimo e o máximo (ex: com 4 bits tem-se 16 graduações, com 5 bits 32 graduações, com 8 bits 256 graduações, etc.).

Alguns modelos de microcontroladores já possuem internamente circuito gerador de PWM, o que facilita bastante a vida do usuário , só é necessário configurá-los para obter-se o efeito desejado. Há casos em que esta facilidade não está presente, ou mesmo não existem em quantidade suficiente, restando ao usuário a única opção de gerar a(s) saída(s) PWM por software.

Para gerar PWM por software, alguns pontos devem ser considerados pois além de responsáveis pelos cálculos necessários, também são limitadores. São eles:

- Freqüência de clock
- Ciclos de clock consumidos por cada instrução.
- Freqüência desejada para o PWM.
- Resolução desejada para o PWM.

Como a geração de PWM por soft é totalmente dependente da programação, os valores máximos obtidos (freqüência e/ou resolução) são bem menores que os possíveis por hardware no mesmo microcontrolador pois os comandos consomem tempo (ciclos de clock) para serem executados.

O método apresentado neste documento é bem simples e flexível, podendo ser implementado "in-line" no programa do usuário ou por interrupção (melhor opção por manter a freqüencia do PWM mais constante – aconselhado para programas em que as demais rotinas tenham tempo de execução com variações consideráveis),

A base de controle consiste de:

- Uma constante para definir a resolução (que passaremos a chamar de graduação, pois neste método não está mais intimamente ligado a bits e bytes.
- Uma variável de trabalho para os passos da graduação.
- Uma variável para guardar o valor do duty cicle (uma para cada saída PWM desejada).
- Uma variável de trabalho para o duty cicle (uma para cada saída PWM desejada).
- Uma pequena lógica para ativar/desativar cada saída PWM que poderá ser utilizada no corpo do programa ou em uma rotina de interrupção.

Em seguida será apresentada a lógica de controle através de pseudo-código, de modo que o usuário poderá utilizar a linguagem de programação que melhor lhe convier.

Este exemplo "in-line" é baseado em 2 saídas PWM com 100 graduações cada, que dará a possibilidade de ter na saída de 0 a 100% da potência controlada em passos de 1%:

```
Programa PWM
Definições
  pinoA = saída
  pinoB = saída
Constantes
 graduação = 100
Variáveis
 graduação-temp
 duty-saída1
 duty-saída2
 duty-saída1-temp
 duty-saída2-temp

Início
 graduação-temp = 0
 duty-saída1 =  60
 duty-saída2 =  35
```

```
repetir
controle-PWM:
  se graduação-temp  = 0
    graduação-temp = graduação
    duty-saída1-temp = duty-saída1
    duty-saída2-temp = duty-saída2
  senão
    gastar-tempo1
  fim-se

 se duty-saída1-temp > 0
  colocar pinoA em nível alto
  duty-saída1-temp = duty-saída1-temp – 1
 senão
  colocar pinoA em nível baixo
  gastar-tempo2
 fim-se

 se duty-saída2-temp > 0
  colocar pinoB em nível alto
  duty-saída2-temp = duty-saída2-temp – 1
 senão
  colocar pinoB em nível baixo
  gastar-tempo2
 fim-se

graduação-temp = graduação-temp – 1

voltar para repetir

Fim-Programa-PWM
```

Como pode ser visto no exemplo, o método é bem simples e teoricamente pode ser ampliado para quantas saídas o usuário desejar; isto é,teoricamente porque a quantidade máxima é limitada pelo tempo consumido pelos comandos (que também depende da linguagem utilizada), pela freqüência desejada para o PWM , quantidade de graduações disponíveis desejada e pelo clock do circuito.

Um ponto importante a ser observado é a utilização dos blocos/rotinas gastar-tempo1 e gastar-tempo2, para que a freqüência do PWM seja constante. Estas rotinas devem ser dimensionadas de modo que o mesmo tempo seja consumido quando a condição dos testes forem verdadeiras ou falsas. Por exemplo, no trecho abaixo, considerando os valores indicados ao lados dos comandos como o tempo consumido para a execução dos mesmos (1 T = uma unidade de tempo)

```
 se duty-saída2-temp > 0
   colocar pinoB em nível alto     .....    1 T
   duty-saída2-temp = duty-saída2-temp – 1  .....   2 T
  senão
   colocar pinoB em nível baixo .....    1 T
   gastar-tempo2
  fim-se
```

O bloco/rotina gastar-tempo2 precisa ser implementado de modo a consumir também exatamente 2 T para manter constante a freqüência do PWM, o que geralmente é feito com simples comandos nulos ou laços de repetição.

No exemplo também pode ser visto que duty-saída1 e duty-saída2 foram implementados como variáveis ao invés de constantes, o que não tem sentido pois não sofrem alteração no corpo do programa. Isto foi feito de propósito para o próximo exemplo, pois do jeito que está o programa não é funcional visto que para alterar os duty cicles seria necessário recompila-lo.

O exemplo seguinte é uma evolução do primeiro exemplo, em que através de 4 botões serão alterados os duty cicles de cada saída conforme abaixo:

- Botão 1 = aumenta duty cicle da saída 1
- Botão 2 = diminui duty cicle da saída 1
- Botão 3 = aumenta duty cicle da saída 2
- Botão 4 = diminui duty cicle da saída 2

```
Programa PWM 2
Definições
  pinoA = saída
  pinoB = saída
  pinoC = entrada  (botão 1)
  pinoD = entrada  (botão 2)
  pinoE = entrada  (botão 3)
  pinoF = entrada  (botão 4)
Constantes
 graduação = 100
Variáveis
 graduação-temp
 duty-saída1
 duty-saída2
 duty-saída1-temp
 duty-saída2-temp
 botão            (variável para guardar o botão pressionado)

Início
 botão = 0
 graduação-temp = 0
 duty-saída1 =  0   (começar com as saídas desligadas)
 duty-saída2 =  0

repetir
controle-PWM:
  se graduação-temp  = 0
    graduação-temp = graduação
    duty-saída1-temp = duty-saída1
    duty-saída2-temp = duty-saída2
  senão
    gastar-tempo1
  fim-se

 se duty-saída1-temp > 0
  colocar pinoA em nível alto
  duty-saída1-temp = duty-saída1-temp – 1
```

```
  senão
    colocar pinoA em nível baixo
    gastar-tempo2
  fim-se

  se duty-saída2-temp > 0
    colocar pinoB em nível alto
    duty-saída2-temp = duty-saída2-temp – 1
  senão
    colocar pinoB em nível baixo
    gastar-tempo2
  fim-se

  ler-botões
se botão = 1
     se duty-saída1 < 100
        duty-saída1 = duty-saída1 + 1
     senão
        gastar-tempo3
     fim-se
senão
  se botão = 2
     se duty-saída1 > 0
        duty-saída1 = duty-saída1 - 1
     senão
        gastar-tempo3
     fim-se
  senão
    se botão = 3
       se duty-saída2 < 100
          duty-saída2 = duty-saída2 + 1
       senão
         gastar-tempo3
       fim-se
    senão
       se botão = 4
          se duty-saída2  > 0
            duty-saída2 = duty-saída2  -  1
          senão
            gastar-tempo3
          fim-se
       senão
          gastar-tempo4
       fim-se      (botão = 4)
    fim-se         (botão = 3)
  fim-se           (botão = 2)
fim-se             (botão = 1)

graduação-temp = graduação-temp – 1

voltar para repetir

Fim-Programa-PWM 2
```

Neste segundo exemplo, agora funcional, o usuário pode alterar os duty cicles de cada saída em passos de 1% para mais ou para menos conforme o botão pressionado que é identificado na rotina ler-botões (não descrita aqui pois foge ao escopo deste documento). As alterações nos duty cicle só estarão disponíveis após o término do ciclo atual.

Como pode ser visto no programa, novas rotinas/blocos para gastar tempo (gastar-tempo3 e gastar-tempo4) fazem-se necessárias para garantir a constância da freqüência de saída do PWM.

Para uma aplicação simples onde há apenas o controle do PWM, o artifício de utilizar rotinas para gastar tempo e assim garantir a constância da freqüência de saída como nos exemplos anteriores, é válido.Para aplicações em que seja necessário outras funcionalidades ou mesmo controlar várias saídas PWM, esta prática limita muito a freqüência máxima possível. Além do tempo necessário para executar os outros comandos, mais e mais rotinas de gastar tempo seriam necessárias para garantir a estabilidade da freqüência de saída.

A maneira mais elegante e eficaz para controlar o PWM é colocar o controle do mesmo em uma rotina chamada por interrupção. Deste modo pode-se manter a linearidade da freqüência de saída sem sobrecarregar o programa com rotinas de perda de tempo e possibilitar outras funcionalidades a aplicação.

A seguir, temos o programa do exemplo anterior alterado para que o controle do PWM seja executado por interrupção.

```
Programa PWM 3
Definições
   pinoA = saída
   pinoB = saída
   pinoC = entrada  (botão 1)
   pinoD = entrada  (botão 2)
   pinoE = entrada  (botão 3)
   pinoF = entrada  (botão 4)
Constantes
  graduação = 100
Variáveis
  graduação-temp
  duty-saída1
  duty-saída2
  duty-saída1-temp
  duty-saída2-temp
  botão              (variável para guardar o botão pressionado)

Rotina-de-interrupção
controle-PWM:
   se graduação-temp  = 0
      graduação-temp = graduação
      duty-saída1-temp = duty-saída1
      duty-saída2-temp = duty-saída2
   senão
      gastar-tempo1
   fim-se

  se duty-saída1-temp > 0
```

```
    colocar pinoA em nível alto
    duty-saída1-temp = duty-saída1-temp – 1
  senão
    colocar pinoA em nível baixo
    gastar-tempo2
  fim-se

  se duty-saída2-temp > 0
    colocar pinoB em nível alto
    duty-saída2-temp = duty-saída2-temp – 1
  senão
    colocar pinoB em nível baixo
    gastar-tempo2
  fim-se

Fim-rotina-de-interrupção


Início
  botão = 0
  graduação-temp = 0
  duty-saída1 =  0   (começar com as saídas desligadas)
  duty-saída2 =  0

configurar-interrupção

repetir

ler-botões

se botão = 1
      se duty-saída1 < 100
          duty-saída1 = duty-saída1 + 1
      fim-se
senão
  se botão = 2
      se duty-saída1 > 0
          duty-saída1 = duty-saída1 – 1
      fim-se
  senão
     se botão = 3
        se duty-saída2 < 100
            duty-saída2 = duty-saída2 + 1
        fim-se
     senão
        se botão = 4
            se duty-saída2  > 0
              duty-saída2 = duty-saída2  -  1
            fim-se
        fim-se      (botão = 4)
     fim-se         (botão = 3)
  fim-se            (botão = 2)
fim-se              (botão = 1)

graduação-temp = graduação-temp – 1

(outras funcionalidades)

voltar para repetir
```

Fim-Programa-PWM 3

Com a implementação do controle do PWM por interrupção, o usuário tem mais liberdade para incluir outras funcionalidades no programa sem ter que se preocupar em montar rotinas de perda de tempo para equalizar a freqüência de saída.

Para que o usuário defina o tempo a ser configurado para as ocorrências das interrupções, são necessários alguns cálulos preliminares. Os dados a serem apurados para os cálculos já foram apresentados no princípio deste documento.

Para determinar o tempo (ou ciclos de clock) consumidos pela rotina de controle do PWM quando o programa é desenvolvido em assembly , basta o usuário somar os tempos/ciclos de cada instrução que podem ser obtidos no datasheet do componente utilizado. Quando é utilizada uma linguagem de mais alto nível (C, Pascal etc.), a maioria dos ambientes de desenvolvimento possuem alguma ferramenta onde o usuário pode obter o tempo consumido pelos comandos/rotinas do programa.

Utilizando o último exemplo e alguns dados sobre o componente em que o mesmo seria aplicado, a seguir serão apurados os valores limites que podem ser obtidos para o PWM.

Considerando que o componente a ser utilizado com um clock de 4MHz obtenha uma velocidade de 1 MIPS (milhões de instruções por segundo) e que a rotina de interrupção consuma 40 micro-segundos (40 instruções em assembly), com as 100 graduações desejadas, a freqüência máxima possível seria de 250 Hz.

$$1.000.000 \ / (40 * 100) = 250$$

Apesar do cálculo indicar uma freqüência máxima de 250Hz, na prática deve-se utilizar um valor mais baixo de modo que os comandos do programa fora da rotina de interrupção possam ser executados também.

Se a freqüência for reduzida para 200 Hz, o programa poderá executar 10 instruções entre cada chamada a rotina de interrupção; o qwue é satisfatório para o programa exemplo.

$$1.000.000 / ((40 + 10) * 100) = 200$$

Portanto, o usuário deverá configurar para que ocorra uma interrupção a cada 0,005 segundos.

No caso de ser necessária uma freqüência do PWM maior, o usuário teria como opção aumentar a freqüência de clock do circuito (se o componente suportar) ou diminuir a quantidade de graduações.

Com o mesmo exemplo, elevando-se o clock para 10 MHz (2,5 MIPS) e reduzindo a quantidade de graduações para 50 (passos de 2%), o usuário poderá obter uma freqüência de 1 KHz para o PWM.

2.500.000 / ((40 + 10) * 50) = 1.000