Inatel Instituto Nacional de Telecomunicações	**RELATÓRIO 3 - Temporizadores**	**Data: / /**
	Disciplina: E209	
	Prof: João Pedro Magalhães de Paula Paiva	
Conteúdo: Microcontroladores AVR		
Tema: Temporizadores e PWM		
Nome:	**Matrícula:**	**Curso:**

OBJETIVOS:

- Utilizar ferramentas de simulação para desenvolver programas para o ATmega328p.
- Desenvolver programas que utilizam os conceitos de temporizadores e geração de PWM.
- Utilizar as entradas e saídas do ATmega328p com circuitos de aplicação.

Parte Teórica

Temporizadores

O timer é um **bloco interno do microcontrolador capaz de contar tempo.** Qualquer temporizador programável é um contador no qual os pulsos de clock possuem um período fixo. Ou seja, se o clock está ajustado para um período de 1 ms e o contador apresenta em um determinado momento o número 100 significa que se passaram 100 ms (admitindo o início em 0).
O ATMega328p possui 3 temporizadores internos, sendo o timer 0 e 2 de 8 bits e o timer 1 de 16 bits, com fonte de clock ajustável. A base de tempo dos TIMERS pode ser ajustada entre algumas opções. No caso que vamos abordar, a fonte de clock do timer é o clock do cristal **(XTAL = 16MHz)**. O valor padrão é 16 MHz, ou seja, o período base é de 62,5ns. Atente-se que esse valor de frequência está ajustado na inicialização do programa pela calibração do oscilador interno, que também possui outros valores calibrados. Além disso, é possível ajustar diferentes fatores de divisão para o sinal de clock dos TIMERS em 8, 64, 256, 1024, ou seja:

```
Se fator de divisão = 8    => Fclk = 16MHz/8    = 2MHz    => Tclk = 500ns.
Se fator de divisão = 64   => Fclk = 16MHz/64   = 250kHz  => Tclk = 4µs.
Se fator de divisão = 256  => Fclk = 16MHz/256  = 62,5kHz => Tclk = 16µs.
Se fator de divisão = 1024 => Fclk = 16MHz/1024 = 15,6kHz => Tclk = 64µs.
```

O timer do ATMega238p tem vários modos: normal, CTC (Clear timer on compare match), PWM, Fast PWM. O nosso interesse é no modo CTC, ou seja, modo de comparação, que é feito pela configuração dos campos **WGM00 = 0, WGM01 = 1** no registrador **TCCR0A**.
A contagem é realizada pelo registro **TCNT0** e ele é comparado com o registro **OCR0A** ou **OCR0B**. Essa contagem está representada nas Figuras 1 e 2. Quando o valor do **TCNT0** atinge o valor de **OCR0A ou OCR0B**, o fim da contagem é indicado por um bit de sinalização chamado de flag de interrupção, nesse caso a flag do canal **OCR0A** que é **OCF0A**. Nesse momento, o valor do **TCNT0** é automaticamente zerado e a contagem é reiniciada e o processo se repete.

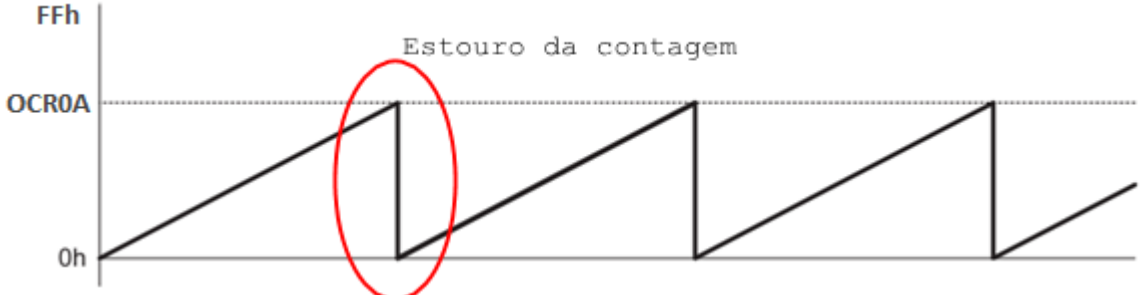


Figura 1 – Contagem crescente do Timer0 até o valor do OCR0A

Os 3 timers internos do ATMega328p são independentes: **Timer0, Timer1 e Timer2.** Como iremos utilizar o **Timer0**, os registros **TCCR0A**, **TCCR0B**, **OCR0A** e **TIMSK0** devem ser configurados para gerar uma interrupção a cada intervalo desejado.
De forma geral, sempre utilizamos a seguinte sequência de configurações para um timer qualquer:

1) Configurar o modo de operação do timer e o divisor de clock: **TCCR0A e TCCR0B.**
2) Configurar o valor máximo de contagem: **OCR0A**
3) Habilitar a interrupção do comparador desejado: **TIMSK0**
4) **Habilitar a interrupção global do microcontrolador: sei();**

Exemplos de como configurar o Timer (linhas de comando no programa):

Para essa etapa você primeiro deve saber exatamente como quer preparar seu timer, ou seja, qual a base de tempo e como vai gerar a interrupção de tratamento.
A primeira coisa é configurar o modo do contador para CTC.

```
TCCR0A = 0b00000010;
```

Após configurar o modo é necessário decidir o divisor do clock base **(F_CPU = 16MHZ)**, os valores oferecidos são: 8, 64, 256, 1024. Se optar por usar a divisão lembre-se que o tempo base de contagem irá mudar, por exemplo: O clock de 16MHZ dividido por 256, será igual à 62,5kHz (16MHZ/256 = 16,5kHz) assim você terá um período base de 16µs.

```
TCCR0B = 0b00000100;
```

Após escolher o clock base e o divisor, você estará apto a escolher o tempo que quer que seu timer conte. Sabendo que o timer conta quantos ciclos de clock ocorreram, se ele contar 10 clocks de 16µs cada, ele terá contado por 160us. Dessa forma se quer contar 100ms com um período de clock de 16µs é necessário contar 6250 pulsos de clock. Lembre-se que o TIMER tem 8 bits, logo pode contar de 0 a 255. O valor a ser contado deve ser:

```
OCR0A = INTERVALO;
```

Agora temos um timer configurado e contando um tempo específico, entretanto, ele ainda não é capaz de avisar ao ATMega328p que o tempo definido foi alcançado. Para isso é necessário ligar a Interrupção do comparador e criar a função de tratamento. Lembre-se de acionar a interrupção global caso ela ainda não esteja acionada.

Linha para ligar interrupção do comparador A do timer0:

```
TIMSK0 = 0b00000010;
```

Linha para ligar interrupção GLOBAL:

```
sei();
```

Função de tratamento da interrupção:

```
ISR(TIMER0_COMPA_vect)
{
    // aqui você colocara o que será realizado toda vez que o timer estourar
}
```

Algo muito comum de acontecer é quando a capacidade de contagem do timer não é o suficiente para alcançar o tempo desejado, para isso podemos dividir o valor total em pequenos valores que cabem entre 0 e 255. Dentro da interrupção do timer colocamos uma variável que contará quantas vezes o timer estourou **(Número de interrupções geradas)**. Usando essa variável podemos saber quanto tempo já se passou desde que o timer iniciou e se o tempo que queremos já foi alcançado.
Por exemplo: Para contar 1 segundo usando um clock de 16MHz e o divisor 1024 (Tclock = 64us) com limite de 200 contagens no OCR0A, ou seja, cada vez que o **TCNT0**

zerar terá se passado(64us x 200 = 12,8ms), sendo assim conseguiremos contar 1 segundo. Então dividimos esse 1 segundo por 12,8ms, ou seja, teremos que entrar na função de interrupção 78 vezes (78 * 12,8ms ≈ 1s).

```
ISR(TIMER0_COMPA_vect) // Vetor de interrupção compa
{
  cont++;
  if(cont >= 78) {
      //aqui você colocará sua lógica
      //toda vez que o timer estourar 78 vezes
      //totalizando 1 segundo
       cont = 0;
  }
}
```

Dica: Ao invés de digitar todas essas linhas em sua função MAIN podemos criar funções extras para diferentes propósitos, como:

```
void ConfigTimer0(void)
{
      TCCR0B = 0b00000101; //Configuração do PreScaler em 1024
      TCCR0A = 0b00000010; //Modo de comparação
      OCR0A = INTERVALO;
      TIMSK0 = 0b00000010; //Interrupção habilitada para o comparador A
}
```

E também:

```
void DisableTimer0(void)
{
      TCCR0B = 0;
}
```

Essas funções extras que criamos tornam o código mais organizado e fácil de identificar erros.

PWM

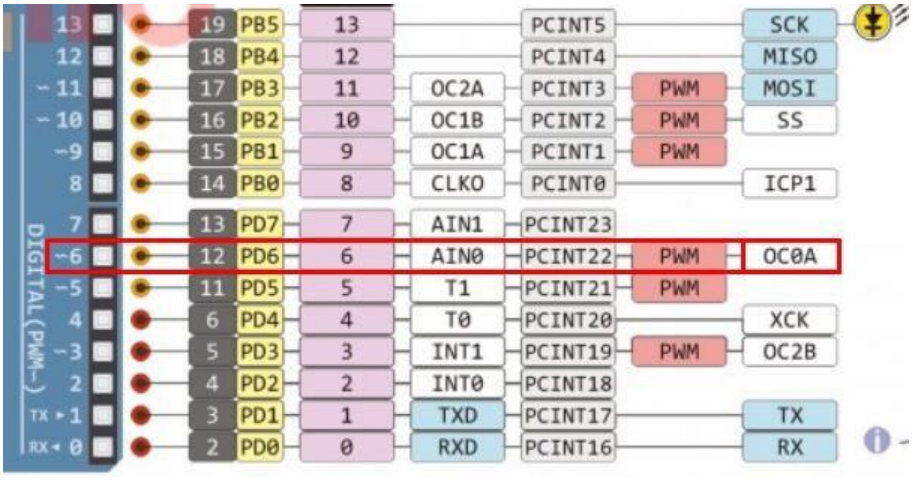
O objetivo dessa seção é estudar o modo comparação para geração de sinal **PWM** (Pulse Width Modulation = modulação por largura de pulso), com o temporizador do ATMega328p. O sinal PWM apresenta um período **(T)** fixo e um ciclo de trabalho **(DC = dutycycle)** variável e, consequentemente, um tempo ligado **(Ton)** também variável. Dessa forma, ao variar o valor do DC tem-se a variação do nível médio do sinal. Essa variação pode ser utilizada para diferentes aplicações, tais como controle de intensidade de rotação de um motor, controle de chaveamento de uma fonte, controle de volume digital, geração de sinais analógicos, entre outras. A Figura 1 ilustra um sinal periódico com período **Tpwm_period** e um tempo em alta **Ton**. O PWM consiste, então, em aumentar ou diminuir o **Ton**, comparado ao período para que forneça uma ideia de ajuste proporcional de uma carga conectada ao microcontrolador.

Figura 1 - Sinal PWM

Como configurar o PWM

O primeiro passo para se configurar um timer PWM é escolher um timer(0, 1 ou 2) e verificar qual sua saída, para isso devemos consultar o manual do ATMega328p. Após a escolha, devemos "indicar" ao microcontrolador que o pino será saída. Para isso devemos utilizar o registro **DDRx**. Vamos considerar o **TIMER0**, a saída **pwm** é dada pelo **OCR0A**(comparador A **TIMER0**):

```
DDRD = 0b01000000; //Configurar como função de saída de sinal do PWM A .
```



Uma vez configurado o pino, passamos para a configuração do **TIMER**, pois é ele que irá gerar os tempos que serão utilizados para ligar e desligar a saída. A primeira configuração é selecionar o modo **SET** e **RESET** que se encontra no registro **TCCR0A**. Esse modo faz com que seja gerado no pino de saída uma transição toda vez que o TIMER alcança o **OCR0A** e o **TCNT0** (contador do TIMER0) reseta.

```
TCCR0A = 0b10000011;
```

Table 14-3. Compare Output Mode, Fast PWM Mode(1)

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	WGM02 = 0: Normal port operation, OC0A disconnected. WGM02 = 1: Toggle OC0A on compare match.
1	0	Clear OC0A on compare match, set OC0A at BOTTOM, (non-inverting mode).
1	1	Set OC0A on compare match, clear OC0A at BOTTOM, (inverting mode).

Table 14-8. Waveform Generation Mode Bit Description

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on(1)(2)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, phase correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, phase correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Notes: 1. MAX = 0xFF
2. BOTTOM = 0x00

Agora devemos então escolher o divisor do clock do nosso TIMER:

```
TCCR0B = 0b00000001; // clock com divisor por 1 ou sem divisor
```

Table 14-9. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(no prescaling)
0	1	0	$clk_{I/O}$/8 (from prescaler)
0	1	1	$clk_{I/O}$/64 (from prescaler)
1	0	0	$clk_{I/O}$/256 (from prescaler)
1	0	1	$clk_{I/O}$/1024 (from prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

E então, devemos definir qual será o dutycycle do sinal. O registro que devem ser modificados é:

```
OCR0A = PERIODO;
```

Esse registro será comparado com o tempo já contado pelo ATMega328p, chamaremos esse tempo de **TCNT0**. Quando **TCNT0** é igual a **OCR0A** o timer irá gerar um **RESET, saída vai para 0,** e quando é igual a **TCNT0** é gerado um **SET, saída vai para 1.**

Modificando o TON:

Para fazermos uso do PWM devemos modificar seu tempo ligado **(TON)**, para isso, podemos fazer de duas formas:

1. Modificando o valor do registro diretamente: Para modificarmos o valor do registro, podemos chamá-lo e atribuirmos seu novo valor:
   ```
   OCR0A = NOVO_VALOR;
   ```

2. Criar a função de Duty Cycle: Para criarmos a função de duty cycle* devemos ter em mente que o valor do **OCR0A** equivale sempre a uma parte do **TCNT0** que vai até seu overflow, como é um registro de 8 bits seu overflow é 255, em porcentagem de 0% a 100% de **255**. Essa função recebe como entrada o valor desejado de DC (0% a 100%) e o converte para um valor de **0** a **255**:

   ```
   if (VALOR_DC >= 0 && VALOR_DC < 100)
       OCR0A = (int)((VALOR_DC / 100.0) * 256);
   ```

Lembre-se que o valor atribuído ao **OCR0A** não pode ser maior que o valor que **TCNT0** suporta.

*duty cycle é a relação entre tempo ligado e tempo total, ou seja, um DC de 25% em um período (T) de 50ms equivale a ficar 12,5ms ligado e 37,5ms desligado.
Essas funções extras que criamos tornam o código mais organizado e fácil de se identificar erros.

Parte prática:

1. (Fácil) Elaborar um firmware para criar um cronômetro HH:MM:SS com amostragem na interface sempre que um segundo for contado.

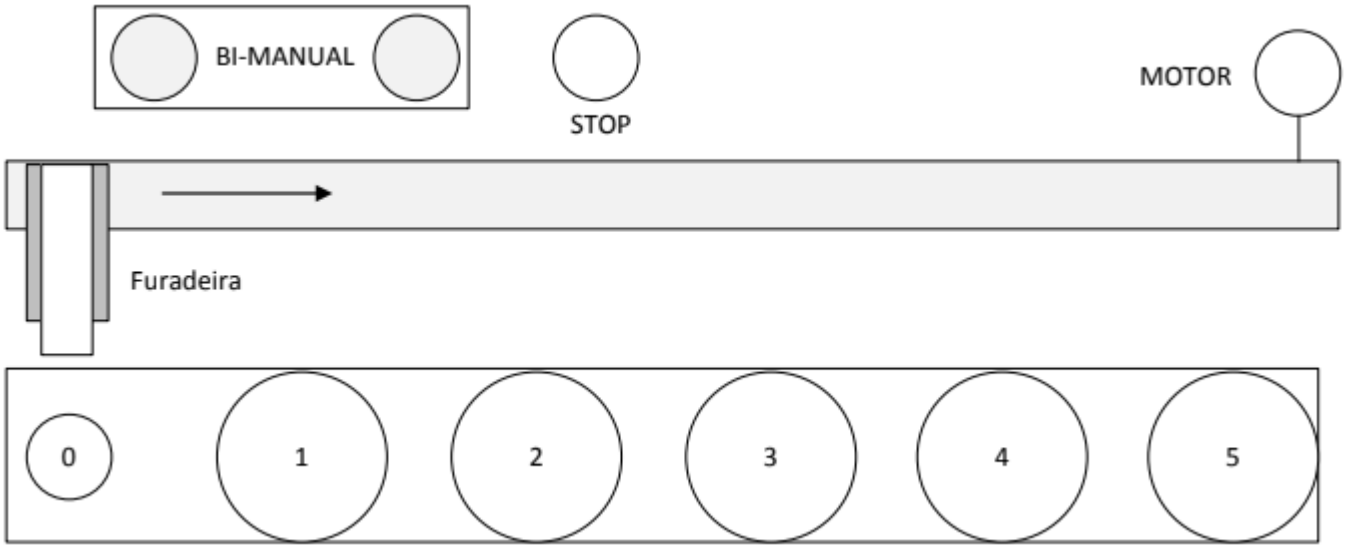
2. (Fácil) Criar um firmware para que cada vez que o botão NA (PB4 - Interrupção) for pressionado, o DC de um PWM (PD6) é aumentado em 10%:

3. (Médio) Elaborar um firmware para que um motor (PD1) seja acionado 10 segundos após o botão LIGA/NA (PB1 – Interrupção) ser mantido pressionado. É preciso prever o desligamento através do botão DESLIGA/NF (PB2 – Interrupção)

4. (Médio) Elaborar um firmware, para manter um motor (PD6) ligado por 8 segundos sempre que o botão LIGA/NA (PB1 – Interrupção) for pressionado. A velocidade desse motor será incrementada em 12,5 % a cada segundo. É preciso prever o desligamento através do botão DESLIGA/NF (PB2 – Interrupção) em qualquer momento da operação.

5. (Médio) Elaborar um firmware para controlar o sentido de giro do motor (PD7) de uma esteira. Ao pressionar o botão LIGA/NA (PC0) o motor irá girar em um sentido até que um sensor S1 (PB0 – Interrupção) seja acionado. Neste instante, o motor para durante 5,5 segundos e volta a girar no sentido oposto até um outro sensor S2 (PB1 – Interrupção) seja acionado. Neste instante, o motor para durante outros 7,5 segundos e volta a girar no sentido inicial, repetindo todo o processo cicliacamente. Acrescentar o botão DESLIGA/NF (PC1) para parar o todo o processo em qualquer instante.

6. (Difícil - Extra) Em um processo de usinagem em uma indústria metalúrgica, o processo de furação em peça, ocorre de forma seriada, ou seja, as peças são furadas automaticamente nos “berços” utilizando uma furadeira que se movimenta em uma guia linear através de um Motor (PB0 - ON/OFF).
   Elaborar um firmware para controlar o sistema de furação seguindo os passos:



   a) Ao pressionar o comando Bi-Manual /NA (PC0 e PC1 acionadas ao mesmo tempo), o Motor é acionado e a furadeira se movimenta para a direita até ficar alinhado com os berços onde estão as peças. O posicionamento da furadeira nos berços é feito por temporizador e segue o diagrama apresentado abaixo.

Movimento	Tempo (segundos)
0 para 1	4
1 para 2	3
2 para 3	2
3 para 4	6
4 para 5	5

b) Após a furadeira se movimentar até o berço, o Motor de Furação (PD6) é acionado por 4 segundos para executar o trabalho, com sua potência sendo variada de acordo com a tabela apresentada abaixo. Assim que a furação em um determinado berço termina, automaticamente a furadeira vai para o próximo berço, seguindo a sequencia 0, 1, 2, 3, 4 e 5. Depois de furada a última peça no berço 5, o sistema todo é desligado, e o operador volta a furadeira manualmente para o berço 0.

Berço	Potência (%)
1	40
2	80
3	35
4	10
5	75

c) Em qualquer momento de processo, se o botão DESLIGA (PC2 - NF) for pressionado, o sistema deve parar e retornar a posição inicial.