Inatel Instituto Nacional de Telecomunicações	**RELATÓRIO 9**	**Data: / /**	
	Disciplina: E209		
	Prof: Yvo Marcelo Chiaradia Masselli **Monitores: João Lucas/Luan Siqueira/Maria Luiza/ Lucas Lares/Rafaela Papale**		
Conteúdo: Microcontrolador ATMega328p			
Tema: Temporizador e PWM			
Nome: Francisco José Carvalho Junior		**Matrícula: 1628**	**Curso: GEC**

OBJETIVOS:

- Utilizar as ferramentas de simulação para desenvolver programas para o ATMega328p.
- Desenvolver um programa de controle que faça uso do temporizador interno operando como gerador de sinal PWM.
- Utilizar as entradas e saídas do ATMega328p com circuitos de aplicação.

Parte Teórica

Temporizador (Timer) - PWM

O objetivo desse relatório 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.

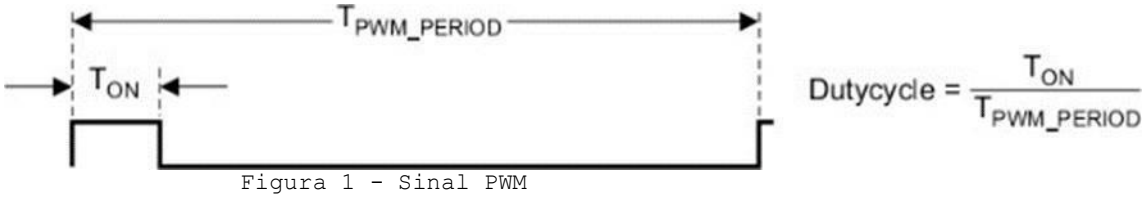
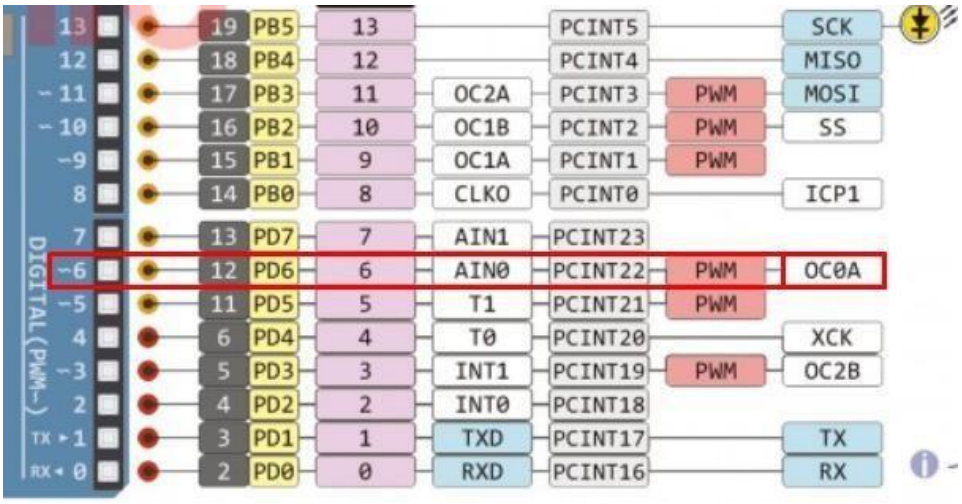


Figura 1 - Sinal PWM

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 |= ( 1 << PD6); //Configurar como função de saída de sinal do OCR0A.
```



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 |= (1 << COM0A1);
```

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).

Logo após, devemos configurar o modo de operação do **TIMER0**. No nosso relatório anterior escolhemos o modo CTC, agora iremos escolher o modo **FAST PWM**.

```
TCCR0A |= (1 << WGM01) | (1 << WGM00);
```

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 = (1 << CS00); // 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) * 255);
   ```

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

Programa 1)

Crie um programa que enquanto o botão esteja pressionado, o led verde é aceso com 50% de intensidade e assim que for solto o led deve apagar. Utilize o **TIMER PWM** para a elaboração do firmware.

```
void setup(){

    //Configs

    //Configurando o PWM
    TCCR0A |= (1 << WGM01) | (1 << WGM00); // Modo fast PWM

    //TCNT0 = 0 -> PD6(OC0A) = 1
    //TCNT0 = OCR0A -> PD6(OC0A) = 0
    TCCR0A |= (1 << COM0A1); //TCNT0 = 0, PD6(OC0A) = 1
    TCCR0B |= (1 << CS00); // preescaler 1

    //Configurando pd6 saida
    DDRD |= (1 << PD6);

    //Pull UP em PD2
    PORTD |= (1 << PD2);

}

void loop(){
    if((PIND & (1 << PD2)) == 0){
        OCR0A = 128;
    }else{
        OCR0A = 0;
    }
}
```

Programa 2)

Modifique o programa anterior para que cada vez que o botão for pressionado o led receba +10% de luminosidade. O led deve começar apagado e quando chegar a 100%, após apertar o botão mais uma vez ele desliga. Utilize interrupção externa e o TIMER configurado com PWM.

```c
int brilho = 0;

ISR(INT0_vect){
    brilho += 25;// + 10% DC

    if(brilho >= 255){
        brilho = 0;
    }
}

void setup(){

    //Configs

    //Configurando o PWM
    TCCR0A |= (1 << WGM01) | (1 << WGM00); // Modo fast PWM

    //TCNT0 = 0 -> PD6(OC0A) = 1
    //TCNT0 = OCR0A -> PD6(OC0A) = 0
    TCCR0A |= (1 << COM0A1); //TCNT0 = 0, PD6(OC0A) = 1
    TCCR0B |= (1 << CS00); // preescaler 1

    //Configurando INT0
    EICRA |= (1 << ISC01);
    EIMSK |= (1 << INT0);

    //Configurando pd6 saida
    DDRD |= (1 << PD6);

    //Pull UP em PD2
    PORTD |= (1 << PD2);

    sei();
}

void loop(){
    OCR0A = brilho;
}
```

ANEXO)

```c
#define pwm_out (1 << PD6)
int brightness = 0;
int main()
{
    DDRD |= pwm_out;   // configura saída para o PWM
    PORTD &= ~pwm_out; // PWM inicia desligado
    // Configura modo FAST PWM e modo do comparador A
    TCCR0A |= (1 << WGM01) | (1 << WGM00) | (1 << COM0A1);
    TCCR0B = 1; // Seleciona opção para frequência
    OCR0A = 0;
    for (;;)
    {
        OCR0A = brightness;
        brightness += 10;
        if (brightness > 255)
            brightness = 0;
        _delay_ms(100);
    }
}
```

```c
#define pwmOut (1 << PD6)
int brightness = 0;
void setup()
{
    DDRD |= pwmOut;   // configura saída para o PWM
    PORTD &= ~pwmOut; // PWM inicia desligado
    // Configura modo FAST PWM e modo do comparador A
    TCCR0A |= (1 << WGM01) | (1 << WGM00) | (1 << COM0A1);
    TCCR0B = (1 << CS00); // Seleciona opção para frequência
}
void loop()
{
    OCR0A = brightness;
    brightness += 10;
    if (brightness > 255)
        brightness = 0;
    _delay_ms(100);
```