9. TEMPORIZADORES/CONTADORES

Uma necessidade importante no trabalho com circuitos microcontrolados é a geração de sinais periódicos e eventos. Essas funcionalidades são realizadas pelos contadores/temporizadores (TCs) e muitos projetos só podem ser executados com o uso deles. Logo, entender o seu funcionamento é indispensável, permitindo a escrita de programas eficientes e projetos bem feitos. Neste capítulo, são apresentados os conceitos essenciais para o trabalho com os TCs do ATmega e um resumo para o trabalho com o seus registradores.

No ATmega328, existem dois temporizadores/contadores de 8 bits (TC0 e TC2) e um de 16 bits (TC1). Todos independentes e com características próprias.

O Temporizador/Contador 0 (TC0) emprega 8 bits (permite contagens de 0 até 255). Suas principais características são:

- Contador simples (baseado no *clock* da CPU).
- Contador de eventos externos.
- Divisor do *clock* para o contador de até 10 bits - *prescaler*.
- Gerador para 2 sinais PWM (pinos OC0A e OC0B).
- Gerador de frequência (onda quadrada).
- 3 fontes independentes de interrupção (por estouro e igualdades de comparação).

O Temporizador/Contador 2 (TC2) também emprega 8 bits e suas características são similares ao TC0. Entretanto, apresenta uma função especial para a contagem precisa de 1 s, permitindo o uso de um cristal externo independente para o seu *clock* (32,768 kHz). Pode gerar dois sinais PWM nos pinos OC2A e OC2B.

O Temporizador/Contador 1 (TC1) possui 16 bits e pode contar até 65535. Permite a execução precisa de temporizações, geração de formas de onda e medição de períodos de tempo. Suas principais características são:

- Contador simples (baseado no *clock* da CPU).
- Contador de eventos externos.
- Divisor do *clock* para o contador de até 10 bits - *prescaler*.
- Gerador para 2 sinais PWM (pinos OC1A e OC1B) com inúmeras possibilidades de configuração.
- Gerador de frequência (onda quadrada).
- 4 fontes independentes de interrupção (por estouro e igualdades de comparação).

9.1 TEMPORIZANDO E CONTANDO

Uma das principais função dos TCs é a contagem de pulsos de *clock*, os quais podem ser externos ou internos ao microcontrolador. Dessa forma, permitem a geração de eventos periódicos para uso do programador ou para a determinação de um número de contagens.

Um estouro do contador ocorre quando ele passa do valor máximo permitido para a contagem para o valor zero. Assim, se o TC for de 8 bits, ele contará de 0 até 255 resultando em 256 contagens; se for de 16 bits, contará de 0 até 65535, resultando em 65536 contagens. Assim, o tempo que um TC leva para estourar é dado por:

$$t_{estouro} = \frac{(TOP+1) \times prescaler}{f_{osc}} \quad (9.1)$$

onde *TOP* é o valor máximo de contagem, f_{osc} é a frequência do *clock* empregado (oscilador) e o *prescaler* é o divisor dessa frequência.

Exemplos:

1. Supondo um TC de 8 bits (conta até o valor TOP de 255), trabalhando com uma frequência de 1 MHz, sem divisor de frequência, o tempo para o seu estouro é de:

$$t_{estouro} = 256 \times \frac{1}{1\ \text{MHz}} \times 1 = 256\ \mu\text{s}$$

2. Supondo um TC de 16 bits (conta até o valor TOP de 65535), trabalhando com uma frequência de 16 MHz, com divisor de frequência de 64, o tempo para o seu estouro é de:

$$t_{estouro} = 65536 \times \frac{1}{16\ \text{MHz}} \times 64 = 0{,}262\ \text{s}$$

Na fig. 9.1, o funcionamento de um TC para o ATmega é ilustrado.

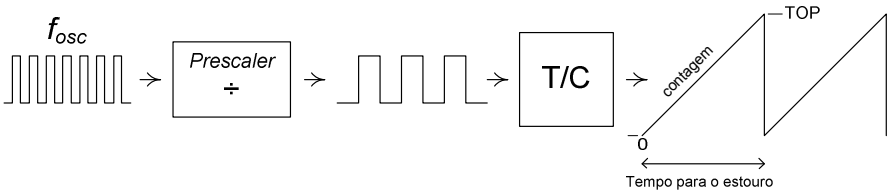

Fig. 9.1 – Funcionamento de um temporizador/contador do ATmega.

9.2 MODULAÇÃO POR LARGURA DE PULSO – PWM

A geração de sinais PWM é outra função importante dos TCs. O uso desses sinais é baseado no conceito de valor médio de uma forma de onda periódica. Baseado no valor médio de um sinal PWM, muitos dispositivos eletrônicos podem ser controlados, como por exemplo: motores, lâmpadas, LEDs, fontes chaveadas e circuitos inversores.

Digitalmente, o valor médio de uma forma de onda é controlado pelo tempo em que o sinal fica em nível lógico alto durante um determinado intervalo de tempo. No PWM, esse tempo é chamado de ciclo ativo (*Duty Cycle*). Na fig. 9.2, são apresentadas formas de onda PWM em que a largura do pulso (*Duty Cycle*) é variável de 0 até 100%, com incrementos de 25%.

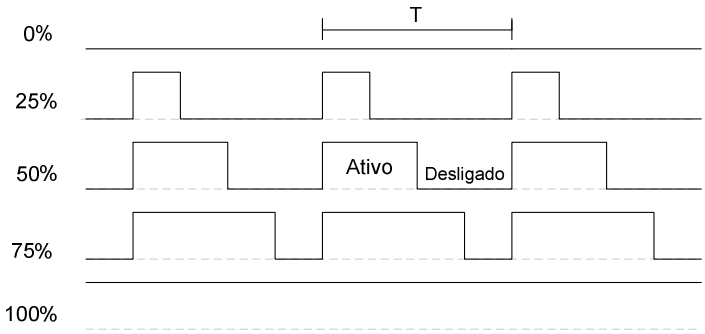

Fig. 9.2 – PWM com período T e ciclo ativo de 0%, 25%, 50%, 75% e 100%.

O cálculo do valor médio de um sinal digital é dado por:

$$V_{médio} = \frac{Amplitude\ Máxima}{Período} \times (Tempo\ Ativo\ no\ Periodo) \qquad (9.2)$$

Assim, se o sinal digital tem variação de 0 a 5 V, um ciclo ativo de 50% corresponde a um valor médio de 2,5 V, enquanto um ciclo ativo de 75% corresponderia a 3,75 V. Logo, ao se alterar o ciclo útil do sinal PWM, altera-se o seu valor médio. Na fig. 9.3, é ilustrada a variação do ciclo ativo de um sinal PWM de 0 a 100% do seu ciclo útil (período) com o passar do tempo.

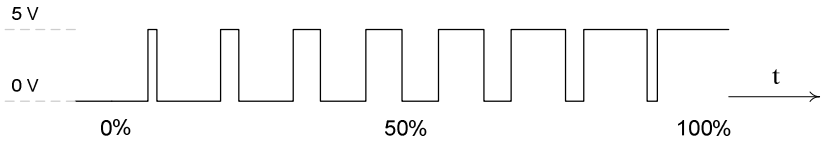

Fig. 9.3 – Variação do ciclo ativo de um sinal PWM em um intervalo de tempo.

É importante notar que o período do sinal PWM não se altera, e sim sua largura de ciclo ativo. Desta forma, quando se gera um sinal PWM, deve-se em primeiro lugar determinar sua frequência de trabalho.

A resolução do PWM em um microcontrolador é dada pelo seu número de bits e indica quantos diferentes ciclos ativos podem ser gerados. Por exemplo, um PWM de 10 bits pode gerar 1024 diferentes níveis, começando com o nível de tensão 0 e terminando com 100 % do ciclo ativo. Como será visto posteriormente, no ATmega o ciclo ativo de um sinal PWM é controlado pela escrita em registradores específicos e sua resolução vai depender do TC empregado.

Além de permitir a geração de sinais PWM, os TCs do ATmega permitem gerar a forma de onda quadrada. Sempre que se desejar gerar tais sinais, os pinos correspondentes do microcontrolador devem ser configurados como pinos de saída, caso contrário o sinal não aparecerá no pino.

9.3 TEMPORIZADOR/CONTADOR 0

O TC0 é um contador de 8 bits que é incrementado com pulsos de *clock,* os quais podem ser obtidos da fonte interna de *clock* do microcontrolador ou de uma fonte externa. Existem vários modos de operação: normal, CTC, PWM rápido e PWM com fase corrigida; permitindo desde simples contagens até a geração de diferentes tipos de sinais PWM.

MODO NORMAL

É o modo mais simples de operação, onde o TC0 conta continuamente de forma crescente. A contagem se dá de 0 até 255 voltando a 0, num ciclo contínuo. Quando a contagem passa de 255 para 0, ocorre o estouro e então, o bit sinalizador de estouro (TOV0) é colocado em 1. Se habilitada, uma interrupção é gerada.

A contagem é feita no registrador TCNT0 e um novo valor de contagem pode ser escrito a qualquer momento, permitindo-se alterar o número de contagens via programação.

MODO CTC

No modo CTC (*Clear Timer on Compare* - limpeza do contador na igualdade de comparação), o registrador OCR0A é usado para manipular a resolução do TC0. Neste modo, o contador é zerado quando o valor do contador (TCNT0) é igual ao valor de OCR0A (o valor TOP da contagem), ou seja, o contador conta de 0 até o valor de OCR0A. Isso permite um controle mais fino da frequência de operação e da resolução do TC0. O modo CTC permite configurar os pinos OC0A e OC0B para gerar ondas quadradas. Uma interrupção pode ser gerada cada vez que o contador atinge o valor de comparação (OCR0A ou OCR0B). Um exemplo de diagrama de tempo para o modo CTC é mostrado na fig. 9.4, onde os pinos OC0A e OC0B foram configurados para trocar de estado quando ocorre a igualdade entre o valor do registrador TCNT0 com OCR0A e OCR0B.

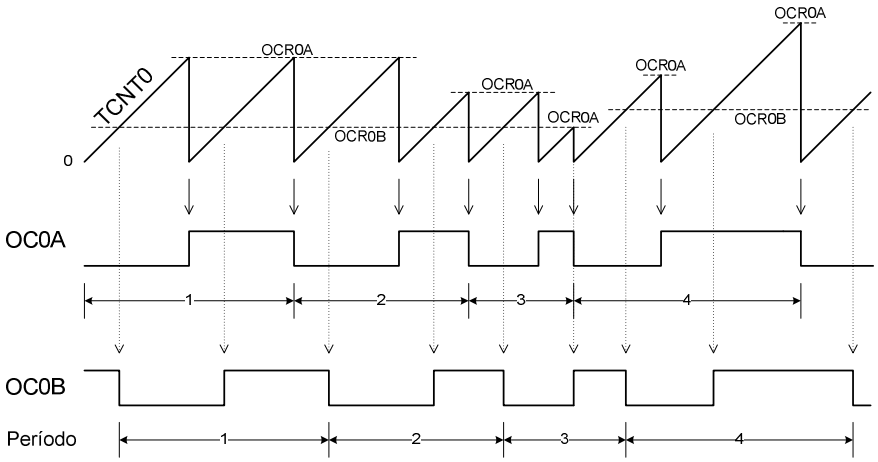

Fig. 9.4 – Modo CTC para a geração de ondas quadradas.

No exemplo, para o pino OC0B, o registrador OCR0B é empregado para deslocar a forma de onda em relação a OC0A e deve ter valor menor ou igual a OCR0A. Para gerar uma onda de saída no modo CTC, os pinos OC0A e/ou OC0B devem ser configurados como saídas e ajustados para

trocar de estado em cada igualdade de comparação através dos bits do modo de comparação.

A frequência no modo CTC é definida por:

$$f_{OC0A} = \frac{f_{osc}}{2N(1+OCR0A)} \quad \text{[Hz]} \tag{9.3}$$

onde f_{osc} é a frequência de operação do microcontrolador, OCR0A assume valores entre 0 e 255 e N é o fator do *prescaler* (1, 8, 64, 256 ou 1024). Para calcular o valor de OCR0A baseado na frequência desejada, basta isolá-lo na equação acima, o que resulta em:

$$\text{OCR0A} = \frac{f_{osc}}{2N.f_{OC0A}} - 1 \tag{9.4}$$

O modo CTC não necessita ser empregado para a geração de formas de onda, pode ser utilizado para temporizações e contagens externas. Ele permite flexibilizar o número de contagens do TC0 e gerar interrupções por igualdade de comparação do TCNT0 com os registradores OCR0A e OCR0B.

MODO PWM RÁPIDO

O modo de modulação por largura de pulso rápido permite a geração de um sinal PWM de alta frequência. O contador conta de zero até o valor máximo e volta a zero. No modo de comparação com saída não-invertida, o pino OC0A é zerado na igualdade entre TCNT0 e OCR0A, e colocado em 1 no valor mínimo do contador. No modo de comparação com saída invertida, o processo é inverso: OC0A é ativo na igualdade e zerado no valor mínimo. É o registrador OCR0A que determina o ciclo ativo do sinal PWM. Também é possível habilitar o pino OC0B para gerar um sinal PWM, onde o ciclo ativo do PWM será controlado pelo valor de OCR0B. A contagem do TC0 pode ser ajustada para 255 ou pelo valor dado por OCR0A; se for utilizado o registrador OCR0A, o pino OC0A não poderá gerar um sinal PWM, apenas uma onda quadrada. Quando se controla o ciclo ativo do sinal

PWM, o valor zero para OCR0A produz um pequeno ruído e o valor máximo (255) vai deixar o sinal PWM em 0 ou 1, conforme foi habilitada a saída, invertida ou não.

A alta frequência do PWM rápido o torna adequado para aplicações de regulação de potência, retificação e outras usando conversores DAs. Na fig. 9.5, é apresentado o exemplo do diagrama temporal para o modo PWM rápido para as saídas não invertidas e contagem máxima do TC de 255.

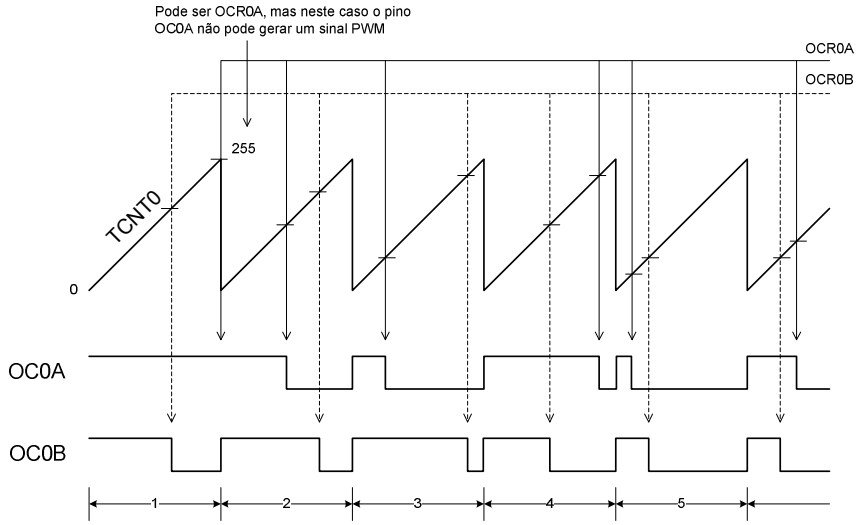

Fig. 9.5 – Modo PWM rápido, saídas não invertidas.

O valor de comparação OCR0A pode ser atualizado na interrupção por estouro do contador cada vez que o contador chegar no valor máximo de contagem, gerando um sinal PWM variável.

A frequência de saída do PWM é calculada como:

$$f_{OC0A\_PWM} = \frac{f_{osc}}{N(1+TOP)} \qquad \text{[Hz]} \qquad (9.5)$$

onde f_{osc} é a frequência de operação do microcontrolador, N é o fator do *prescaler* (1, 8, 64, 256 ou 1024) e TOP assume valores conforme tab. 9.7.

Um sinal com ciclo ativo do PWM de 50% pode ser obtido ajustando-se OC0A para mudar de estado a cada igualdade de comparação. A máxima frequência é obtida quando OCR0A é zero ($f_{OC2\_PWM} = f_{osc}/2$).

MODO PWM COM FASE CORRIGIDA

O modo PWM com fase corrigida permite ajustar a fase do PWM, isto é, o início e fim do ciclo ativo do sinal PWM. É baseado na contagem crescente e decrescente do TCNT0, que conta repetidamente de zero até o valor máximo (TOP) e vice-versa (permanece um ciclo de *clock* no valor TOP), o que torna a saída PWM simétrica dentro de um período. Além disso, o valor de comparação que determina o razão cíclica da saída PWM é armazenado em registradores, sendo atualizado apenas quando o contador atinge o valor TOP, o qual marca o início e o fim de um ciclo PWM. Assim, se o valor TOP não for alterado com o temporizador em operação, o pulso gerado será sempre simétrico em relação ao ponto médio do período (TCNT0 = 0x00), qualquer que seja a razão cíclica.

Por ser um sinal que permite um maior controle do ciclo ativo do sinal PWM, é adequado para o controle de motores, sendo mais lento e preciso que o modo PWM rápido.

Para o sinal PWM não invertido, o pino OC0A é zerado na igualdade entre TCNT0 e OCR0A, quando a contagem é crescente, e colocado em 1 quando a contagem é decrescente. No modo de comparação com saída invertida, o processo é contrário. A comparação para o pino OC0B é feita com o registrador OCR0B. O valor máximo de contagem pode ser definido como 255 ou pelo valor de OCR0A, como no modo PWM rápido; se OCR0A for utilizado para determinar o valor TOP, o pino OC0A não poderá gerar um sinal PWM, somente uma onda quadrada.

Na fig. 9.6, é exemplificado um diagrama de tempo do modo PWM com fase corrigida com a saída OC0A não invertida e a saída OC0B invertida.

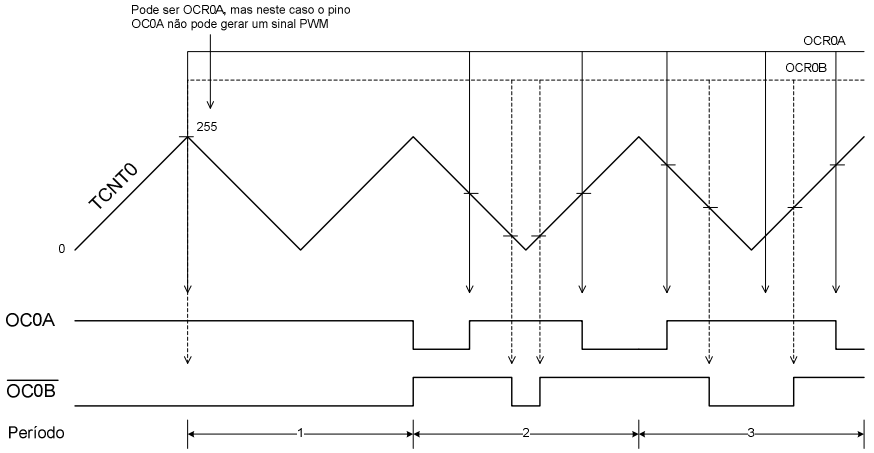

Fig. 9.6 – Modo PWM com correção de fase , OC0A não invertida e OC0B invertida.

A frequência de saída do PWM com fase corrigida é calculada como:

$$f_{OC0A\_PWM} = \frac{f_{osc}}{2N(1+\text{TOP})} \qquad \text{[Hz]} \qquad (9.6)$$

onde f_{osc} é a frequência de operação do microcontrolador, N é o fator do *prescaler* (1, 8, 64, 256 ou 1024) e TOP assume valores conforme tab. 9.7.

9.3.1 REGISTRADORES DO TC0

O controle do modo de operação do TC0 é feito nos registradores TCCR0A e TCCR0B (*Timer/Counter Control* 0 *Register A e B*).

Bit		7	6	5	4	3	2	1	0
	TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00
Lê/Escr.		L/E	L/E	L/E	L/E	L	L	L/E	L/E
Valor Inic.		0	0	0	0	0	0	0	0

Bits 7:6 – COM0A1:0 – *Compare Match Output A Mode*
Estes bits controlam o comportamento do pino OC0A (*Output Compare* 0A). Se um ou ambos os bits forem colocados em 1, a funcionalidade normal do pino é alterada. Entretanto, o bit do registrador DDRx correspondente ao pino OC0A deve estar ajustado para habilitar o *driver* de saída. A funcionalidade dos bits COM0A1:0 depende do ajuste dos bits WGM02:0. Suas possíveis configurações são apresentadas nas tabs. 9.1-3.

Tab. 9.1 – Modo CTC (não PWM).

COM0A1	COM0A0	Descrição
0	0	Operação normal do pino, OC0A desconectado.
0	1	Mudança do estado de OC0A na igualdade de comparação.
1	0	OC0A é limpo na igualdade de comparação.
1	1	OC0A é ativo na igualdade de comparação.

Tab. 9.2 – Modo PWM rápido.

COM0A1	COM0A0	Descrição
0	0	Operação normal do pino, OC0A desconectado.
0	1	WGM02 = 0: operação normal do pino, OC0A desconectado. WGM02 = 1: troca de estado do OC0A na igualdade de comparação.
1	0	OC0A é limpo na igualdade de comparação, OC0A ativo no valor do TC mínimo (modo não invertido).
1	1	OC0A é ativo na igualdade de comparação e limpo no valor do TC mínimo (modo invertido).

Tab. 9.3 – Modo PWM com fase corrigida.

COM0A1	COM0A0	Descrição
0	0	Operação normal do pino, OC0A desconectado.
0	1	WGM02 = 0: operação normal do pino, OC0A desconectado. WGM02 = 1: troca de estado do OC0A na igualdade de comparação.
1	0	OC0A é limpo é na igualdade de comparação quando a contagem é crescente, e ativo na igualdade de comparação quando a contagem é decrescente.
1	1	OC0A é ativo na igualdade de comparação quando a contagem é crescente, e limpo na igualdade de comparação quando a contagem é decrescente.

Bits 5:4 – COM0B1:0 – *Compare Match Output B Mode*
Estes bits controlam o comportamento do pino OC0B (*Output Compare* 0B). Se um ou ambos os bits forem colocados em 1, a funcionalidade normal do pino é alterada. Entretanto, o bit do registrador DDRx correspondente ao pino OC0B deve estar ajustado para habilitar o *driver* de saída. A funcionalidade dos bits COM0B1:0 depende do ajuste dos bits WGM02:0. Suas possíveis configurações são apresentadas nas tabs. 9.4-6.

Tab. 9.4 – Modo CTC (não PWM).

COM0B1	COM0B0	Descrição
0	0	Operação normal do pino, OC0B desconectado.
0	1	Mudança do estado de OC0B na igualdade de comparação.
1	0	OC0B é limpo na igualdade de comparação.
1	1	OC0B é ativo na igualdade de comparação.

Tab. 9.5 – Modo PWM rápido.

COM0B1	COM0B0	Descrição
0	0	Operação normal do pino, OC0B desconectado.
0	1	Reservado.
1	0	OC0B é limpo na igualdade de comparação, OC0B ativo no valor do TC mínimo (modo não invertido).
1	1	OC0B é ativo na igualdade de comparação e limpo no valor do TC mínimo (modo invertido).

Tab. 9.6 – Modo PWM com fase corrigida.

COM0B1	COM0B0	Descrição
0	0	Operação normal do pino, OC0B desconectado.
0	1	Reservado.
1	0	OC0B é limpo é na igualdade de comparação quando a contagem é crescente, e ativo na igualdade de comparação quando a contagem é decrescente.
1	1	OC0B é ativo na igualdade de comparação quando a contagem é crescente, e limpo na igualdade de comparação quando a contagem é decrescente.

Bits 1:0 – WGM01:0 – *Wave Form Generation Mode*
Combinados com o bit WGM02 do registrador TCCR0B, estes bits controlam a sequência de contagem do contador, a fonte do valor máximo para contagem (TOP) e o tipo de forma de onda a ser gerada, conforme tab. 9.7.

Tab. 9.7 – Bits para configurar o modo de operação do TC0.

Modo	WGM02	WGM01	WGM00	Modo de Operação TC	TOP	Atualização de OCR0A no valor:	Sinalização do bit TOV0 no valor:
0	0	0	0	Normal	0xFF	Imediata	0xFF
1	0	0	1	PWM com fase corrigida	0xFF	0xFF	0x00
2	0	1	0	CTC	OCR0A	Imediata	0xFF
3	0	1	1	PWM rápido	0xFF	0x00	0xFF
4	1	0	0	Reservado	-	-	-
5	1	0	1	PWM com fase corrigida	OCR0A	OCR0A	0x00
6	1	1	0	Reservado	-	-	-
7	1	1	1	PWM rápido	OCR0A	0x00	OCR0A

TCCR0B – *Timer/Counter* 0 *Control Register B*

Bit		7	6	5	4	3	2	1	0
	TCCR0B	FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00
Lê/Escr.		E	E	L	L	L/E	L/E	L/E	L/E
Valor Inic.		0	0	0	0	0	0	0	0

Bits 7:6 – FOC0A:B – ***Force Output Compare A e B***
Estes bits são ativos somente para os modos não-PWM. Quando em 1, uma comparação é forçada no módulo gerador de onda. O efeito nas saídas dependerá da configuração dada aos bits COM0A1:0 e COM0B1:0.

Bit 3 – WGM02 – ***Wave Form Generation Mode***
Função descrita na tab. 9.7.

Bits 2:0 – CS02:0 – ***Clock Select***
Bits para seleção da fonte de *clock* para o TC0, conforme tab. 9.8.

Tab. 9.8 – Seleção do *clock* para o TC0.

CS02	CS01	CS00	Descrição
0	0	0	Sem fonte de *clock* (TC0 parado).
0	0	1	*clock*/1 (*prescaler=1*) - sem *prescaler.*
0	1	0	*clock*/8 (*prescaler* = 8).
0	1	1	*clock*/64 (*prescaler* = 64).
1	0	0	*clock*/256 (*prescaler* = 256).
1	0	1	*clock*/1024 (*prescaler* = 1024).
1	1	0	clock externo no pino T0. Contagem na borda de descida.
1	1	1	clock externo no pino T0. Contagem na borda de subida.

TCNT0 – ***Timer/Counter* 0 *Register***
Registrador de 8 bits onde é realizada a contagem do TC0, pode ser lido ou escrito a qualquer tempo.

OCR0A – ***Output Compare* 0 *Register A***
Registrador de comparação A de 8 bits, possui o valor que é continuamente comparado com o valor do contador (TCNT0). A igualdade pode ser utilizada para gerar uma interrupção ou uma forma de onda no pino OC0A.

OCR0B – ***Output Compare* 0 *Register B***
Registrador de comparação B de 8 bits, possui o valor que é continuamente comparado com o valor do contador (TCNT0). A igualdade pode ser utilizada para gerar uma interrupção ou uma forma de onda no pino OC0B.

TIMSK0 – *Timer/Counter 0 Interrupt Mask Register*

Bit		7	6	5	4	3	2	1	0
	TIMSK0	-	-	-	-	-	OCIE0B	OCIE0A	TOIE0
Lê/Escreve		L	L	L	L	L	L/E	L/E	L/E
Valor Inicial		0	0	0	0	0	0	0	0

Bit 2 – OCIE0B – *Timer/Counter 0 Output Compare Match B Interrupt Enable*
A escrita 1 neste bit ativa a interrupção do TC0 na igualdade de comparação com o registrador OCR0B.

Bit 1 – OCIE0A – *Timer/Counter 0 Output Compare Match A Interrupt Enable*
A escrita 1 neste bit ativa a interrupção do TC0 na igualdade de comparação com o registrador OCR0A.

Bit 0 – TOIE0 – *Timer/Counter 0 Overflow Interrupt Enable*
A escrita 1 neste bit ativa a interrupção por estouro do TC0.

As interrupções individuais dependem da habilitação das interrupções globais pelo bit I do SREG.

TIFR0 – *Timer/Counter 0 Interrupt Flag Register*

Bit		7	6	5	4	3	2	1	0
	TIFR0	-	-	-	-	-	OCF0B	OCF0A	TOV0
Lê/Escreve		L	L	L	L	L	L/E	L/E	L/E
Valor Inicial		0	0	0	0	0	0	0	0

Bit 2 – OCF0B – *Timer/Counter 0 Output Compare B Match Flag*
Este bit é colocado em 1 quando o valor da contagem (TCNT0) é igual ao valor do registrador de comparação de saída B (OCR0B) do TC0.

Bit 1 – OCF0A – *Timer/Counter 0 Output Compare A Match Flag*
Este bit é colocado em 1 quando o valor da contagem (TCNT0) é igual ao valor do registrador de comparação de saída A (OCR0A) do TC0.

Bit 0 – TOV0 – *Timer/Counter 0 Overflow Flag*
Este bit é colocado em 1 quando um estouro do TC0 ocorre.

9.3.2 CÓDIGOS EXEMPLO

MODO NORMAL – GERANDO UM PEDIDO DE INTERRUPÇÃO

O código abaixo é utilizado para gerar um evento periódico de interrupção a cada 16,384 ms. Neste caso, um LED ligado ao pino PB5 troca de estado (liga ou desliga) a cada interrupção. Foi empregado o maior divisor de *clock* possível, 1024, e não há possibilidade de gerar um intervalo de tempo maior sem a diminuição da frequência de trabalho do microcontrolador.

TC0_estouro.c

```
//============================================================================== //
//       HABILITANDO A INTERRUPÇÃO POR ESTOURO DO TC0                            //
//============================================================================== //
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>

#define cpl_bit(y,bit) (y^=(1<<bit))//troca o estado lógico do bit x da variável Y
#define LED    PB5

//-------------------------------------------------------------------------------
ISR(TIMER0_OVF_vect) //interrupção do TC0
{
     cpl_bit(PORTB,LED);
}
//-------------------------------------------------------------------------------
int main()
{
     DDRB  = 0b00100000;//somente pino do LED como saída
     PORTB = 0b11011111;//apaga LED e habilita pull-ups nos pinos não utilizados

     TCCR0B = (1<<CS02) | (1<<CS00);/*TC0 com prescaler de 1024, a 16 MHz gera uma
                                                    interrupção a cada 16,384 ms*/
     TIMSK0 = 1<<TOIE0;   //habilita a interrupção do TC0
     sei();               //habilita a chave de interrupção global

     while(1)
     {
          /*Aqui vai o código, a cada estouro do TC0 o programa desvia para
                                                              ISR(TIMER0_OVF_vect)*/
     }
}
//==============================================================================
```

MODO CTC

Neste exemplo, os pinos OC0A e OC0B são configurados para trocar de estado na igualdade de comparação do registrador TCNT0 com o OCR0A e OCR0B, respectivamente.

TC0_PWMs.c

```
#define F_CPU 16000000UL
#include <avr/io.h>

int main(void)
{
     DDRD  = 0b01100000;  //pinos OC0B e OC0A (PD5 e PD6) como saída
     PORTD = 0b10011111;  //zera saídas e habilita pull-ups nos pinos não utilizados

     //MODO CTC
     TCCR0A = 0b01010010; /*habilita OC0A e OC0B para trocar de estado na igualdade de
                                                                          comparação*/
     TCCR0B = 0b00000001; //liga TC0 com prescaler = 1.
     OCR0A  = 200;        //máximo valor de contagem
     OCR0B  = 100;        //deslocamento de OC0B em relação a OC0A

     while(1)
     {
        //O programa principal vai aqui
     }
}
```

MODO PWM RÁPIDO

Abaixo são apresentadas algumas configurações, considerando que os pinos OC0A e OC0B estejam configurados como saída.

```
...
//fast PWM, TOP = 0xFF, OC0A e OC0B habilitados
TCCR0A = 0b10100011; //PWM não invertido nos pinos OC0A e OC0B
TCCR0B = 0b00000011; //liga TC0, prescaler = 64
OCR0A = 200;         //controle do ciclo ativo do PWM 0C0A
OCR0B = 50;          //controle do ciclo ativo do PWM OC0B

...
/*fast PWM, TOP = OCR0A com OC0A gerando uma onda quadrada e OC0B um sinal PWM não
                                                                          Invertido*/
TCCR0A = 0b01100011; //TOP = OCR0A, OC0A e OC0B habilitados
TCCR0B = 0b00001001; //liga TC0, prescaler = 1 e ajusta modo para comparação com OCR0A
OCR0A  = 100;        //controle da período do sinal no pino OC0A
OCR0B  = 30;         //controle do ciclo ativo do PWM 0C0B
```

MODO PWM COM FASE CORRIGIDA

Abaixo são apresentadas algumas configurações, considerando que os pinos OC0A e OC0B estejam configurados como saída.

```
...
//phase correct PWM, TOP = 0xFF
TCCR0A = 0b10100001; //OC0A e OC0B habilitados
TCCR0B = 0b00000001; //liga TC0, prescaler = 1
OCR0A = 100;         //controle da período do sinal no pino OC0A
OCR0B = 50;          //controle do ciclo ativo do PWM 0B

...
//phase correct PWM, TOP = OCR0A
TCCR0A = 0b01100011; //OC0A e OC0B habilitado
TCCR0B = 0b00001001; /*liga TC0, prescaler = 1 e habilita o pino OC0A para gerar um
                                                                    onda quadrada*/
OCR0A = 200;         //controle do período do sinal no pino OC0A
OCR0B = 10;          //controle do ciclo ativo do PWM OC0B
```

Exercícios:

9.1 – Considerando o TC0 trabalhando a 8 MHz com um *prescaler* de 256, quanto tempo leva para o TC0 gerar um estouro de contagem?

9.2 – Considerando-se o PWM rápido configurado para que o valor máximo de contagem do TC0 seja dado pelo valor de OCR0A e que o pino OC0B esteja configurado para gerar o sinal PWM, determine:

- Qual o período do sinal PWM se o ATmega328 estiver rodando a 12 MHz e o valor de OCR0A for 99?
- Supondo que OCR0A é 200, qual deve ser o valor de OCR0B para que o ciclo ativo do sinal PWM (OC0B) seja de 75 %?

9.3 – Verifique a configuração dada ao TC0 para os exemplos acima. Analise os bits dos registradores envolvidos.

9.4 – Com o emprego de um osciloscópio, teste os sinais gerados pelos códigos exemplo dados previamente, gravando-os no Arduino. Altere os valores dos registradores OCR0A e OCR0B para analisar o comportamento do microcontrolador. Para a análise, também pode ser empregado o software de simulação Proteus (ISIS).

9.5 – Elaborar um programa para ler 6 teclas utilizando a interrupção externa INT0 do ATmega328 (ver o capítulo 6). O programa principal não deve ficar responsável pela varredura do teclado (ver o capítulo 8); a interrupção do TC0 deve ser empregada para esse fim. O tratamento do

teclado será transparente ao programa principal, devendo ser feito nas interrupções. Assim, quando uma tecla for pressionada será requisitado um pedido de interrupção e o LED correspondente deve ser ligado. O circuito abaixo ilustra o circuito necessário.

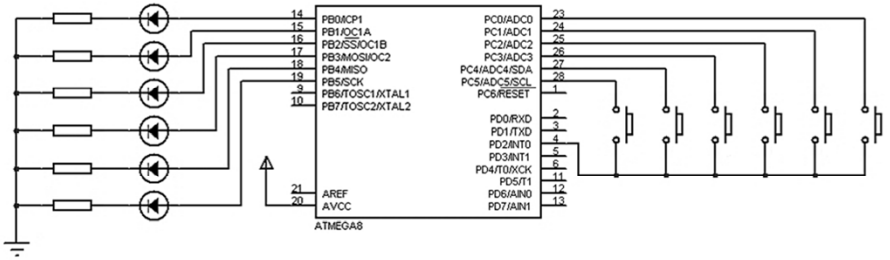

Fig. 9.7 – Seis teclas gerando um pedido de interrupção com o emprego de apenas uma interrupção externa (obs.: a pinagem do ATmega8 é igual a do ATmega328).

9.4 TEMPORIZADOR/CONTADOR 2

O TC2 possui características similares a do TC0, é um contador de 8 bits que permite os modos normal, CTC, PWM rápido e PWM com fase corrigida. Possui dois pinos de saída para os sinais gerados, OC2A (PB3) e OC2B (PD3), ligados aos PORTB e PORTD, respectivamente. Não possui entrada para contagem de sinais de *clock* externos como o TC0. Entretanto, possui mais possibilidades para a divisão do *clock* para a contagem (*prescaler*) e uma característica peculiar: permite o uso de um cristal externo de 32,768 kHz para o seu circuito de *clock*. Essa característica é denominada operação assíncrona.

Para o TC2, os registradores de comparação são o OCR2A e OCR2B e as fórmulas de cálculo para as frequência de trabalho do PWM são similares as do TC0, ver as eqs. 9.3-6. O registrador principal é o OCR2A, que pode ser configurado para determinar o valor máximo de contagem. O *prescaler* pode assumir os seguintes valores: 1, 8, 32, 64, 128, 256 ou 1024.

OPERAÇÃO ASSÍNCRONA

Neste modo, o TC2 utiliza um cristal de relógio de 32,768 kHz para gerar precisamente uma base de tempo de 1 s ou frações dessa. O TC2 possui um circuito oscilador otimizado exclusivamente para o trabalho com esse cristal. Essa característica do TC2 é chamada pela Atmel de RTc – *Real Timer counter*.

Para o ATmega328, os pinos de conexão para o cristal de 32,768 kHz (TOSC1 e TOSC2) são os mesmos pinos de conexão para o uso de um cristal externo ou ressonador cerâmico (pinos XTAL1 e XTAL2) que definem a frequência de trabalho da CPU e, portanto, se for empregado um, o outro não pode ser conectado. Caso o modo assíncrono seja utilizado, é obrigatório o uso do oscilador interno para gerar o sinal de *clock* do microcontrolador. No Arduino não é possível utilizar a operação assíncrona do TC2, pois o microcontrolador trabalha com um cristal de 16 MHz soldado à placa do hardware.

9.4.1 REGISTRADORES DO TC2

O controle do modo de operação do TC2 é feito nos registradores TCCR2A e TCCR2B (*Timer/Counter Control 2 Register A e B*).

Bit		7	6	5	4	3	2	1	0
	TCCR2A	COM2A1	COM2A0	COM2B1	COM2B0	-	-	WGM21	WGM20
Lê/Escr.		L/E	L/E	L/E	L/E	L	L	L/E	L/E
Valor Inic.		0	0	0	0	0	0	0	0

Bits 7:6 – COM2A1:0 – *Compare Match Output A Mode*
Estes bits controlam o comportamento do pino OC2A (*Output Compare* 2A). Se um ou ambos os bits forem colocados em 1, a funcionalidade normal do pino é alterada. Entretanto, o bit do registrador DDRx correspondente ao pino OC2A deve estar ajustado para habilitar o *driver* de saída. A funcionalidade dos bits COM2A1:0 depende do ajuste dos bits WGM22:0. Configurações dadas nas tabs. 9.9-11.

Tab. 9.9 – Modo CTC (não PWM).

COM2A1	COM2A0	Descrição
0	0	Operação normal do pino, OC2A desconectado.
0	1	Mudança do estado de OC2A na igualdade de comparação.
1	0	OC2A é limpo na igualdade de comparação.
1	1	OC2A é ativo na igualdade de comparação.

Tab. 9.10 – Modo PWM rápido.

COM2A1	COM2A0	Descrição
0	0	Operação normal do pino, OC2A desconectado.
0	1	WGM22 = 0: operação normal do pino, OC2A desconectado. WGM22 = 1: troca de estado do OC2A na igualdade de comparação.
1	0	OC2A é limpo na igualdade de comparação, OC2A ativo no valor do TC mínimo (modo não invertido).
1	1	OC2A é ativo na igualdade de comparação e limpo no valor do TC mínimo (modo invertido).

Tab. 9.11 – Modo PWM com fase corrigida.

COM2A1	COM2A0	Descrição
0	0	Operação normal do pino, OC2A desconectado.
0	1	WGM22 = 0: operação normal do pino, OC2A desconectado. WGM22 = 1: troca de estado do OC2A na igualdade de comparação.
1	0	OC2A é limpo é na igualdade de comparação quando a contagem é crescente, e ativo na igualdade de comparação quando a contagem é decrescente.
1	1	OC2A é ativo na igualdade de comparação quando a contagem é crescente, e limpo na igualdade de comparação quando a contagem é decrescente.

Bits 5:4 – COM2B1:0 – *Compare Match Output B Mode*
Estes bits controlam o comportamento do pino OC2B (*Output Compare* 2B). Se um ou ambos os bits forem colocados em 1, a funcionalidade normal do pino é alterada. Entretanto, o bit do registrador DDRx correspondente ao pino OC2B deve estar ajustado para habilitar o *driver* de saída. A funcionalidade dos bits COM2B1:0 depende do ajuste dos bits WGM22:0. Suas possíveis configurações são dadas nas tabs. 9.12-14.

Tab. 9.12 – Modo CTC (não PWM).

COM2B1	COM2B0	Descrição
0	0	Operação normal do pino, OC2B desconectado.
0	1	Mudança do estado de OC2B na igualdade de comparação.
1	0	OC2B é limpo na igualdade de comparação.
1	1	OC2B é ativo na igualdade de comparação.

Tab. 9.13 – Modo PWM rápido.

COM2B1	COM2B0	Descrição
0	0	Operação normal do pino, OC2B desconectado.
0	1	Reservado.
1	0	OC2B é limpo na igualdade de comparação, OC2B ativo no valor do TC mínimo (modo não invertido).
1	1	OC2B é ativo na igualdade de comparação e limpo no valor do TC mínimo (modo invertido).

Tab. 9.14 – Modo PWM com fase corrigida.

COM2B1	COM2B0	Descrição
0	0	Operação normal do pino, OC2B desconectado.
0	1	Reservado
1	0	OC2B é limpo é na igualdade de comparação quando a contagem é crescente, e ativo na igualdade de comparação quando a contagem é decrescente.
1	1	OC2B é ativo na igualdade de comparação quando a contagem é crescente, e limpo na igualdade de comparação quando a contagem é decrescente.

Bits 1:0 – WGM21:0 – *Wave Form Generation Mode*
Combinados com o bit WGM22 do registrador TCCR2B, esses bits controlam a sequência de contagem do contador, a fonte do valor máximo para contagem (TOP) e o tipo de forma de onda a ser gerada, conforme tab. 9.15.

Tab. 9.15 – Bits para configurar o modo de operação do TC2.

Modo	WGM22	WGM21	WGM20	Modo de Operação TC	TOP	Atualização de OCR2A no valor:	Sinalização do bit TOV2 no valor:
0	0	0	0	Normal	0xFF	Imediata	0xFF
1	0	0	1	PWM com fase corrigida	0xFF	0xFF	0x00
2	0	1	0	CTC	OCR2A	Imediata	OCR2A
3	0	1	1	PWM rápido	0xFF	0x00	0xFF
4	1	0	0	Reservado	-	-	-
5	1	0	1	PWM com fase corrigida	OCR2A	OCR2A anterior	0x00
6	1	1	0	Reservado	-	-	-
7	1	1	1	PWM rápido	OCR2A	0x00	OCR2A

TCCR2B – *Timer/Counter 2 Control Register B*

Bit		7	6	5	4	3	2	1	0
	TCCR2B	FOC2A	FOC2B	-	-	WGM22	CS22	CS21	CS20
Lê/Escr.		E	E	L	L	L/E	L/E	L/E	L/E
Valor Inic.		0	0	0	0	0	0	0	0

Bits 7:6 – FOC2A:B – *Force Output Compare A e B*
Estes bits são ativos somente para os modos não-PWM. Quando em 1, obrigam uma comparação no módulo gerador de forma de onda. O efeito nas saídas dependerá da configuração dado aos bits COM2A1:0 e COM2B1:0.

Bit 3 – WGM22 - *Wave Form Generation Mode*
Função descrita na tab. 9.15.

Bits 2:0 – CS22:0 – *Clock Select*
Bits para seleção da fonte de *clock* para o TC2, conforme tab. 9.16.

Tab. 9.16 – Seleção do *clock* para o TC2.

CS22	CS21	CS20	Descrição
0	0	0	Sem fonte de *clock* (TC2 parado)
0	0	1	*clock*/1 (*prescaler=1*) - sem *prescaler*
0	1	0	*clock*/8 (*prescaler* = 8)
0	1	1	*clock*/32 (*prescaler* = 32)
1	0	0	*clock*/64 (*prescaler* = 64)
1	0	1	*clock*/128 (*prescaler* = 128)
1	1	0	*clock*/256 (*prescaler* = 256)
1	1	1	*clock*/1024 (*prescaler* = 1024)

TCNT2 – *Timer/Counter 2 Register*
Registrador de 8 bits onde é realizada a contagem do TC2, pode ser lido ou escrito a qualquer momento.

OCR2A – *Output Compare 2 Register A*
Registrador de comparação A de 8 bits, possui o valor que é continuamente comparado com o valor do contador (TCNT2). A igualdade pode ser utilizada para gerar uma interrupção ou uma forma de onda na pino OC2A.

OCR2B – *Output Compare 2 Register B*
Registrador de comparação B de 8 bits, possui o valor que é continuamente comparado com o valor do contador (TCNT2). A igualdade pode ser utilizada para gerar uma interrupção ou uma forma de onda na pino OC2B.

TIMSK2 – *Timer/Counter 2 Interrupt Mask Register*

Bit	7	6	5	4	3	2	1	0
TIMSK2	-	-	-	-	-	OCIE2B	OCIE2A	TOIE2
Lê/Escreve	L	L	L	L	L	L/E	L/E	L/E
Valor Inicial	0	0	0	0	0	0	0	0

Bit 2 – OCIE2B – *Timer/Counter 2 Output Compare Match B Interrupt Enable*
A escrita de 1 neste bit ativa a interrupção do TC2 na igualdade de comparação com o registrador OCR2B.

Bit 1 – OCIE2A – *Timer/Counter 2 Output Compare Match A Interrupt Enable*
A escrita de 1 neste bit ativa a interrupção do TC2 na igualdade de comparação com o registrador OCR2A.

Bit 0 – TOIE2 – *Timer/Counter 2, Overflow Interrupt Enable*
A escrita de 1 neste bit ativa a interrupção por estouro do TC2.

As interrupções individuais dependem da habilitação das interrupções globais pelo bit I do registrador SREG.

TIFR2 – *Timer/Counter 2 Interrupt Flag Register*

Bit	7	6	5	4	3	2	1	0
TIFR2	-	-	-	-	-	OCF2B	OCF2A	TOV2
Lê/Escreve	L	L	L	L	L	L/E	L/E	L/E
Valor Inicial	0	0	0	0	0	0	0	0

Bit 2 – OCF2B – *Timer/Counter 2, Output Compare B Match Flag*
Este bit é colocado em 1 quando o valor da contagem (TCNT2) é igual ao valor do registrador de comparação de saída B (OCR2B) do TC2.

Bit 1 – OCF2A – *Timer/Counter 2, Output Compare A Match Flag*
Este bit é colocado em 1 quando o valor da contagem (TCNT2) é igual ao valor do registrador de comparação de saída A (OCR2A) do TC2.

Bit 0 – TOV2 – *Timer/Counter 2 Overflow Flag*
Este bit é colocado em 1 quando um estouro do TC2 ocorre.

ASSR – *Asynchronous Status Register*

Bit		7	6	5	4	3	2	1	0
	ASSR	-	EXCLK	AS2	TCN2UB	OCR2AUB	OCR2BUB	TCR2AUB	TCR2BUB
Lê/Escreve		L	L/E	L/E	L	L	L	L	L
Valor Inicial		0	0	0	0	0	0	0	0

Bit 6 – EXCLK – *Enable External Clock Input*
A escrita de 1 neste bit em conjunto com a seleção do *clock* assíncrono permite que um *clock* externo seja a entrada para o pino TOSC1, ao invés do uso de um cristal de 32,768 kHz. O oscilador a cristal só funcionará se este bit estiver em zero.

Bit 5 – AS2 – *Asynchronous Timer/Counter 2*
A escrita de 1 neste bit ativa o uso do cristal externo de 32,768 kHz para o uso exclusivo do TC2, nos pinos TOSC1 e TOSC2.

Bit 4:0 – TCN2UB, OCR2AUB, OCR2BUB, TCR2AUB e TCR2BUB.
São bits sinalizadores da escrita nos registradores TCNT2, OCR2B, OCR2A, TCCR2A e TCCR2B, respectivamente.

9.4.2 CÓDIGO EXEMPLO

CONTANDO 1 s

O código a seguir é utilizado para gerar um evento periódico de interrupção a cada 1 s. Neste caso, um LED ligado ao pino PB5 troca de estado (liga ou desliga) a cada interrupção. Dentro da rotina de interrupção está o código para a geração de um relógio com a contagem de segundos, minutos e horas. É bom lembrar que quando um registrador de configuração não aparece no código, ou seja, não é escrito, assume-se o seu valor *default*, que geralmente é zero.

TC2_relogio.c

```
//=============================================================================== //
//       TC2 COM CRISTAL EXTERNO DE 32,768 kHz                                    //
//       Rotina para implementar um relógio e piscar um LED                      //
//=============================================================================== //
#define F_CPU 1000000UL        //frequência de operação de 1 MHz
#include <avr/io.h>
#include <avr/interrupt.h>

#define cpl_bit(Y,bit_x) (Y^=(1<<bit_x)) //complementa bit
#define LED PB5

volatile unsigned char segundos, minutos, horas;/*variáveis com os valores para o
                    relógio precisam ser tratadas em um código adequado. Variáveis
                    globais que são utilizadas dentro de interrupções e lidas fora
                    delas devem ser declaradas como volatile. Isto é uma exigência
                    para o compilador. Respostas imprevisíveis correrão se o comando
                    volatile não for empregado.*/
//------------------------------------------------------------------------------
ISR(TIMER2_OVF_vect)      //entrada aqui a cada 1 segundo
{                         //rotina para contagem das horas, minutos e segundos
     cpl_bit(PORTB,LED);  //pisca LED

     segundos++;

     if(segundos == 60)
     {
          segundos = 0;
          minutos++;

          if (minutos == 60)
          {
               minutos = 0;
               horas++;

               if (horas == 24)
                    horas = 0;
          }
     }
}
//------------------------------------------------------------------------------
int main()
{
     DDRB = 0b00100000;//cristal ext., não importa a config. dos pinos TOSC1 e TOSC2
     PORTB= 0b11011111;//pull-ups habilitados nos pinos não utilizados

     ASSR  = 1<<AS2;//habilita o cristal externo para o contador de tempo real
     TCCR2B = (1<<CS22)|(1<<CS20);/*prescaler = 128, freq. p/ o contador = 32.768/128,
                              o que resulta em uma freq. de 256 Hz. Como o contador
                              é de 8 bits, ele conta 256 vezes, resultando em um
                              estouro preciso a cada 1 segundo*/
     TIMSK2 = 1<<TOIE2;   //habilita interrupção do TC2

     sei();               //habilita interrupção global

     while(1)
     {}                   //código principal (display, ajuste de hora, minutos, etc..)
}
//==============================================================================
```

Exercícios:

9.6 – Qual o valor do divisor de frequência para que o TC2, trabalhando com um cristal externo de 32,768 kHz, produza um estouro a cada 1/4 de segundo?

9.7 – O uso de um cristal 32,768 kHz em relógios digitais tem um motivo óbvio, qual?

9.8 – Elaborar um programa para que o hardware da fig. 9.8 funcione como um relógio 24 h. O ajuste do horário deve ser feito nos botões específicos.

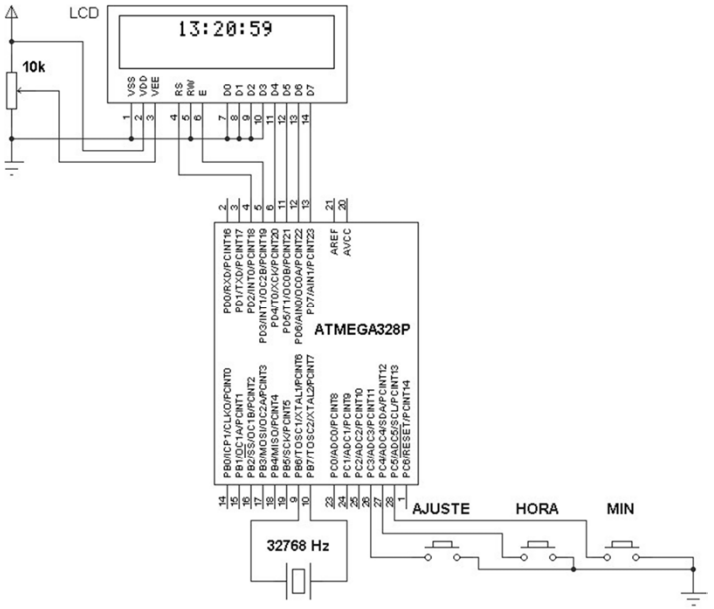

Fig. 9.8 – Relógio com contagem baseado em cristal externo.

9.5 TEMPORIZADOR/CONTADOR 1

O TC1 é um contador de 16 bits que permite grande precisão na geração de formas de onda, temporizações e medição de largura de pulsos. Interrupções podem ser habilitadas para os eventos de estouro ou igualdade de comparação.

A contagem é feita no par de registradores TCNT1H e TCNT1L (contador TCNT1). O TC1 é incrementado via *clock* interno, com ou sem *prescaler*, ou por *clock* externo, ligado ao pino T1. Possuindo dois registradores de controle: TCCR1A e TCCR1B. Os registradores de comparação de saída OCR1A e OCR1B são constantemente comparados com o valor de contagem. O resultado da comparação pode ser usado para gerar sinais PWMs ou com frequência variável nos pinos de saída de comparação (OC1A e OC1B). O valor máximo de contagem (TOP) para o TC1 pode ser definido em alguns modos de operação por OCR1A, ICR1 ou por um conjunto fixo de valores. O registrador de captura de entrada pode capturar o valor do contador quando ocorrer um evento externo no pino ICP1 ou nos pinos do comparador analógico.

MODO NORMAL

É o modo mais simples de operação do contador. Neste modo, a contagem é crescente e contínua, o contador conta de 0 até 65535 (0xFFFF) e volta a zero, sinalizando um estouro e solicitando um pedido de interrupção (se habilitada). O registrador de contagem TCNT1 pode ser escrito ou lido a qualquer momento pelo programa e tem prioridade sobre a contagem ou a limpeza do contador.

MODO DE CAPTURA DE ENTRADA

O modo captura de entrada permite medir o período de um sinal digital externo. Para isso, deve-se ler o valor do TC1 quando ocorrer a interrupção; após duas interrupções, é possível calcular o tempo decorrido entre elas. O

programa deve lidar com os estouros do TC1, caso ocorram. No modo de captura de entrada, escolhe-se qual o tipo de borda do sinal gerará a interrupção (descida ou subida). Para medir o tempo em que um sinal permanece em nível alto ou baixo, após cada captura, é preciso alterar o tipo de borda para a captura.

MODO CTC

No modo CTC (*Clear Timer on Compare*) os registradores OCR1A ou ICR1 são utilizados para mudar a resolução do contador (valor de topo). Neste modo, o contador é limpo (zerado) quando o valor do TCNT1 é igual a OCR1A ou igual a ICR1. Este modo permite grande controle na comparação para estabelecer a frequência de saída e também simplifica a operação de contagem de eventos externos. Um exemplo de diagrama temporal para o modo CTC é apresentado na fig. 9.9; os pinos OC1A e OC1B foram habilitados.

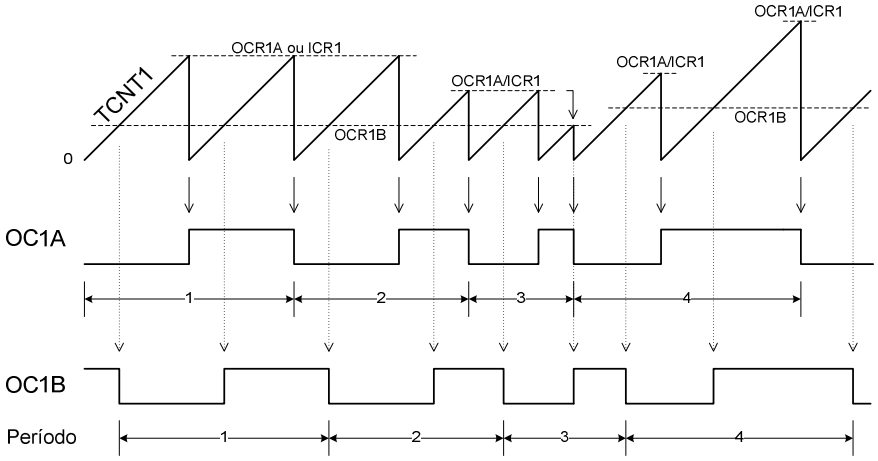

Fig. 9.9 – Diagrama de tempo para o modo CTC.

Para gerar uma onda de saída no modo CTC, os pinos OC1A e OC1B devem ser ajustados para trocar de nível a cada igualdade de comparação. Os valores de OC1A OC1B não serão visíveis nos seus respectivos pinos, a

não ser que estejam configurados como saída. A máxima frequência que a forma de onda pode alcançar é a metade da frequência de trabalho da CPU. Essa frequência é definida por:

$$f_{OC1A} = \frac{f_{osc}}{2N(1+TOP)} \qquad \text{[Hz]} \tag{9.7}$$

onde f_{osc} é a frequência de operação do microcontrolador, TOP tem um valor entre 0 e 65535, e N é o fator do *prescaler* (1, 8, 64, 256 ou 1024). Para calcular o valor TOP em função da frequência desejada, basta isolá-lo na equação acima, o que resulta em:

$$\text{TOP} = \frac{f_{osc}}{2N.f_{OC1A}} - 1 \tag{9.8}$$

MODO PWM RÁPIDO

O modo PWM rápido permite a geração de um sinal PWM de alta frequência. O contador conta de zero até o valor máximo e volta a zero. No modo de comparação com saída não-invertida, o pino OC1A é zerado na igualdade entre TCNT1 e OCR1A e colocado em 1 no valor mínimo do contador. No modo de comparação com saída invertida, o processo é inverso: OC1A é ativo na igualdade e zerado no valor mínimo. A comparação para o pino OC1B é feita com OCR1B. O valor de contagem para o TC1 pode ser definido pelos valores fixos 0x0FF (255), 0x1FF (511) ou 0x3FF (1023), por OCR1A ou ICR1; caso OCR1A seja empregado, o pino OC1A não pode gerar uma sinal PWM, apenas uma onda quadrada.

A resolução para o PWM rápido pode ser fixada em 8, 9, 10 bits (0x0FF, 0x1FF ou 0x3FF) ou definida pelos registradores OCR1A ou ICR1. A resolução mínima é de 2 bits (OCR1A ou ICR1 = 3) e a máxima é de 16 bits (ICR1 ou OCR1A = 0xFFFF). A resolução do PWM é calculada com:

$$R_{PWM\_Rápido} = \frac{\log(TOP+1)}{\log(2)} \qquad \text{[bits]} \tag{9.9}$$

Neste modo, o contador é incrementado até encontrar um dos valores fixos 0x0FF, 0x1FF ou 0x3FF, o valor de OCR1A ou o valor de ICR1. Um diagrama exemplo para o PWM rápido com OC1A e OC1B como saídas não invertidas é apresentado na fig. 9.10.

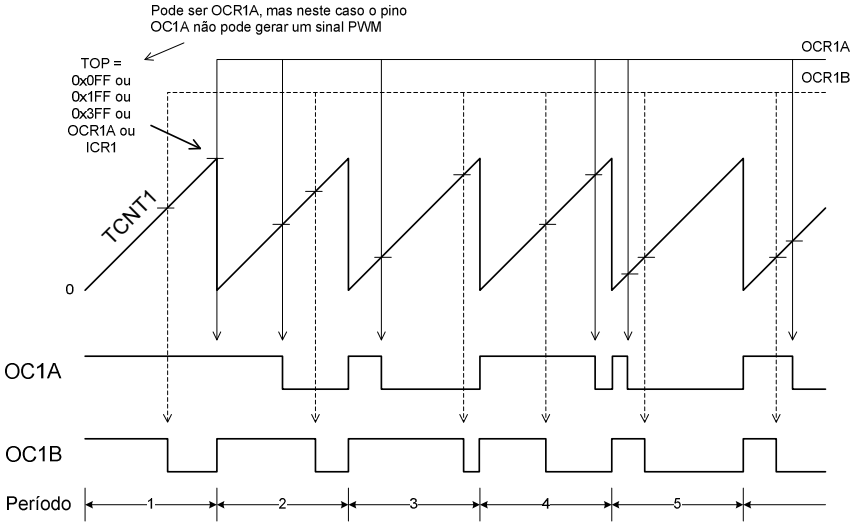

Fig. 9.10 – Diagrama temporal para o modo PWM rápido.

A frequência de saída do PWM rápido é calculada com:

$$f_{OC1x\_PWM} = \frac{f_{osc}}{N.(1+TOP)} \qquad \text{[Hz]} \qquad (9.10)$$

onde f_{osc} é a frequência de operação do microcontrolador e N é o fator do *prescaler* (1, 8, 64, 256 ou 1024). Um sinal com ciclo ativo de 50% pode ser obtido ajustando-se OC1A para mudar de nível lógico a cada igualdade de comparação (onda quadrada). Isso se aplica apenas quando OCR1A é usado para definir o valor TOP. A frequência máxima gerada será a metade da frequência de trabalho da CPU quando OCR1A = 0. Quando se controla o ciclo ativo do sinal PWM, o valor zero para OCR1A produz um pequeno ruído, e o valor máximo (TOP) vai deixar o sinal PWM em 0 ou 1, conforme foi habilitada a saída, invertida ou não.

MODO PWM COM FASE CORRIGIDA

O modo PWM com fase corrigida permite ajustar a fase do PWM, isto é, o início e fim do ciclo ativo do sinal PWM. É baseado na contagem crescente e decrescente do TCNT1, que conta repetidamente de zero até o valor máximo (TOP) e vice-versa (permanece um ciclo de *clock* no valor TOP), o que torna a saída PWM simétrica dentro de um período do PWM. Além disso, o valor de comparação que determina o razão cíclica da saída PWM é armazenado em registradores, sendo atualizado apenas quando o contador atinge o valor TOP, o qual marca o início e o fim de um ciclo PWM. Assim, se o valor TOP não for alterado com o temporizador em operação, o pulso gerado será sempre simétrico em relação ao ponto médio do período (TCNT1 = 0x00), qualquer que seja a razão cíclica.

Para o sinal PWM não invertido, o pino OC1A é zerado na igualdade entre TCNT1 e OCR1A, quando a contagem é crescente, e colocado em 1 quando a contagem é decrescente. No modo de comparação com saída invertida, o processo é o contrário. A comparação para o pino OC1B é feita com o registrador OCR1B. O valor máximo de contagem pode ser definido como 0x0FF, 0x1FF, 0x3FF, pelo valor de OCR1A ou por ICR1, como no modo PWM rápido. Se OCR1A for utilizado para determinar o valor TOP, o pino OC1A não poderá gerar um sinal PWM, somente uma onda quadrada. A resolução do PWM é dada pela mesma equação do modo PWM rápido (eq. 9.9). Um diagrama de tempo exemplo, para o PWM com correção de fase, é mostrado na fig. 9.11 para OC1A como saída invertida e OC1B como saída não-invertida.

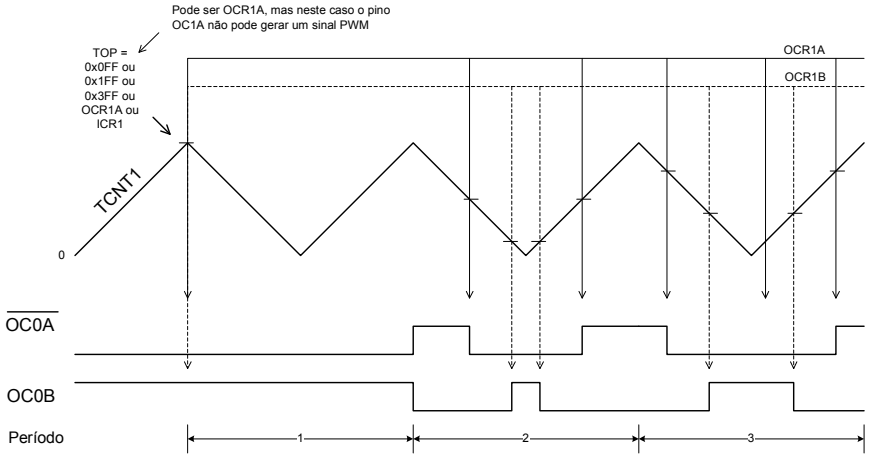

Fig. 9.11 – Diagrama de tempo para o PWM com fase corrigida, saída OC1A invertida e OC1B não invertida.

A frequência de saída do PWM com fase corrigida é calculada com:

$$f_{OC1A\_PWM} = \frac{f_{osc}}{2N.\,TOP} \qquad \text{[Hz]} \qquad (9.11)$$

onde f_{osc} é a frequência de operação do microcontrolador e N é o fator do *prescaler* (1, 8, 64, 256 ou 1024). Se OCR1A definir o valor TOP, a saída OC1A terá um ciclo ativo de 50%.

MODO PWM COM FASE E FREQUÊNCIA CORRIGIDAS

Este modo é igual ao PWM com fase corrigida, a diferença principal é que o pulso gerado será sempre simétrico em relação ao ponto médio do período (neste modo PWM, o TOP), mesmo quando o valor de TOP é alterado com o TC1 em operação. Isso ocorre porque os registradores de comparação são atualizados quando TCNT1 = 0x00, permitindo que o momento da comparação nas inclinações de subida e de descida seja sempre o mesmo (para mais detalhes consultar o manual do ATmega328).

9.5.1 REGISTRADORES DO TC1

O controle do modo de operação do TC1 é feito nos registradores TCCR1A e TCCR1B (*Timer/Counter* 1 *Control Registers*).

Bit		7	6	5	4	3	2	1	0
	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	-	-	WGM11	WGM10
Lê/Escr.		L/E	L/E	L/E	L/E	L	L	L/E	L/E
Valor Inicial		0	0	0	0	0	0	0	0

Bits 7:6 – COM1A1:0 – ***Compare Output Mode for channel A***

Bits 5:4 – COM1B1:0 – ***Compare Output Mode for channel B***

COM1A1:0 e COM1B1:0 controlam o comportamento dos pinos OC1A e OC1B, respectivamente. Se o valor 1 for escrito nesses bits a funcionalidade dos pinos é alterada (os bits no registrador DDRx correspondentes a OC1A e OC1B devem ser colocados em 1 para habilitar cada *driver* de saída). Quando OC1A e OC1B forem empregados, a funcionalidade dos bits COM1x1:0 dependerá do ajuste dos bits WGM13:0. Nas tabs. 9.17-19, são apresentadas as configurações dos bits COM1x1:0 para os diferentes modos de operação do TC1.

Tab. 9.17– Modo não PWM (normal e CTC).

COM1A1/COM1B1	COM1A0/COM1B0	Descrição
0	0	Operação normal dos pinos, OC1A/OC1B desconectados.
0	1	Mudança de OC1A/OC1B na igualdade de comparação.
1	0	Limpeza de OC1A/OC1B na igualdade de comparação (saída em nível lógico baixo).
1	1	OC1A/OC1B ativos na igualdade de comparação (saída em nível lógico alto).

Tab. 9.18 – Modo PWM rápido.

COM1A1/COM1B1	COM1A0/COM1B0	Descrição
0	0	Operação normal dos pinos, OC1A/OC1B desconectados.
0	1	WGM13:0 = 14 ou 15: Mudança de OC1A na igualdade de comparação, OC1B desconectado (operação normal do pino). Para os demais valores de WGM1, OC1A/OC1B estarão desconectados (operação normal dos pinos).
1	0	Limpeza de OC1A/OC1B na igualdade de comparação, ativos no valor mínimo de comparação (modo não invertido).
1	1	OC1A/OC1B ativos na igualdade de comparação, limpos no valor mínimo de comparação (modo invertido).

Tab. 9.19 – Modo PWM com correção de fase e correção de fase e frequência.

COM1A1/COM1B1	COM1A0/COM1B0	Descrição
0	0	Operação normal dos pinos, OC1A/OC1B desconectados.
0	1	WGM13:0 = 9 ou 11: Mudança de OC1A na igualdade de comparação, OC1B desconectado (operação normal do pino). Para os demais valores de WGM1, OC1A/OC1B estarão desconectados (operação normal dos pinos).
1	0	Limpeza de OC1A/OC1B na igualdade de comparação quando a contagem é crescente, ativos no valor mínimo de comparação quando a contagem é decrescente.
1	1	OC1A/OC1B ativos na igualdade de comparação quando a contagem é crescente, limpos no valor mínimo de comparação quando a contagem é decrescente.

Bits 1:0 – WGM11:0 – ***Waveform Generation Mode***
Combinados com os bits WGM13:2 do registrador TCCR1B, esses bits controlam a forma de contagem do contador, a fonte para o valor máximo (TOP) e qual tipo de forma de onda gerada será empregada (tab. 9.20).

Tab. 9.20 – Descrição dos bits para os modos de geração de formas de onda.

Modo	WGM 13	WGM 12	WGM 11	WGM 10	Modo de operação do TC1	Valor TOP	Atualiz. OCR1x no valor	Bit TOV1 ativo no valor:
0	0	0	0	0	Normal	0xFFFF	Imediata	0xFFFF
1	0	0	0	1	PWM com fase corrigida, 8 bits	0x00FF	0x00FF	0
2	0	0	1	0	PWM com fase corrigida, 9 bits	0x01FF	0x01FF	0
3	0	0	1	1	PWM com fase corrigida, 10 bits	0x03FF	0x03FF	0
4	0	1	0	0	CTC	OCR1A	Imediata	0xFFFF
5	0	1	0	1	PWM rápido, 8 bits	0x00FF	0	0x00FF
6	0	1	1	0	PWM rápido, 9 bits	0x01FF	0	0x01FF
7	0	1	1	1	PWM rápido, 10 bits	0x03FF	0	0x03FF
8	1	0	0	0	PWM com fase e freq. corrigidas	ICR1	0	0
9	1	0	0	1	PWM com fase e freq. corrigidas	OCR1A	0	0
10	1	0	1	0	PWM com fase corrigida	ICR1	ICR1	0
11	1	0	1	1	PWM com fase corrigida	OCR1A	OCR1A	0
12	1	1	0	0	CTC	ICR1	Imediata	0xFFFF
13	1	1	0	1	Reservado	-	-	-
14	1	1	1	0	PWM rápido	ICR1	0	ICR1
15	1	1	1	1	PWM rápido	OCR1A	0	OCR1A

TCCR1B - *Timer/Counter* 1 *Control Register B*

Bit		7	6	5	4	3	2	1	0
	TCCR1B	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10
Lê/Escr.		L/E	L/E	L	L/E	L/E	L/E	L/E	L/E
Valor Inicial		0	0	0	0	0	0	0	0

Bit 7 – ICNC1 – *Input Capture Noise Canceler*
Colocando este bit em 1, o filtro de ruído do pino de captura ICP1 é habilitado. Esse filtro requer 4 amostras sucessivas iguais para o ICP1 mudar sua saída. Assim, a captura de entrada é atrasada por 4 ciclos de *clock*.

Bit 6 – ICES1 – *Input Capture Edge Select*
Este bit seleciona qual borda no pino de entrada de captura (ICP1) será usada para disparar o evento de captura (ICES1=0 na transição de 1 para 0, ICES1=1 na transição de 0 para 1). Quando uma captura ocorre, o valor do contador é copiado no registrador ICR1.

Bit 4:3 – WGM13:2 – *Waveform Generation Mode*
Ver a tab. 9.20.

Bit 2:0 – CS12:0 – *Clock Select*
Existem 3 bits para a escolha da fonte de *clock* para o TC1 (tab. 9.21).

Tab. 9.21 – Descrição dos bits para seleção do *clock* para o TC1.

CS12	CS11	CS10	Descrição
0	0	0	Sem fonte de *clock* (TC1 parado).
0	0	1	*clock*/1 (prescaler = 1) sem *prescaler.*
0	1	0	*clock*/8 (*prescaler* = 8) .
0	1	1	*clock*/64 (*prescaler* = 64).
1	0	0	*clock*/256 (*prescaler* = 256).
1	0	1	*clock*/1024(*prescaler* = 1024).
1	1	0	Fonte de *clock* externa no pino T1 (contagem na borda de descida).
1	1	1	Fonte de *clock* externa no pino T1 (contagem na borda de subida).

TCCR1C - *Timer/Counter* 1 *Control Register C*

Bit		7	6	5	4	3	2	1	0
	TCCR1C	FOC1A	FOC1B	-	-	-	-	-	-
Lê/Escr.		L/E	L/E	L	L	L	L	L	L
Valor Inicial		0	0	0	0	0	0	0	0

Bits 7:6 – FOC2A:B – *Force Output Compare A e B*
Estes bits são ativos somente para os modos não-PWM. Quando em 1, obrigam uma comparação no módulo gerador de forma de onda. O efeito nas saídas dependerá da configuração dado aos bits COM1A1:0 e COM1B1:0.

TCNTH e TCNTL (TCNT1) – *Timer/Counter* 1 *Register*
São os dois registrador de 8 bits onde é realizada a contagem do TC1, H (*high*) L (*Low*), podem ser lidos ou escritos a qualquer tempo.

OCR1AH e OCR1AL (OCR1A) – *Output Compare* 1 *Register A*
Registradores de comparação A de 8 bits cada, possui o valor que é continuamente comparado com o valor do contador (TCNT1). A igualdade pode ser utilizada para gerar uma interrupção ou uma forma de onda no pino OC1A.

OCR1BH e OCR1BL (OCR1B) – *Output Compare* 1 *Register B*
Registradores de comparação B de 8 bits cada, possui o valor que é continuamente comparado com o valor do contador (TCNT1). A igualdade pode ser utilizada para gerar uma interrupção ou uma forma de onda na pino OC1B.

ICR1H e ICR1L (ICR1) – *Input Capture Register* 1
Esses registradores são atualizados com o valor do TCNT1 cada vez que um evento ocorre no pino ICP1 (ou opcionalmente nos pinos do comparador analógico). Também são empregados para definir o valor máximo de contagem (TOP).

TIMSK1 – *Timer/Counter* 1 *Interrupt Mask Register*

Bit		7	6	5	4	3	2	1	0
	TIMSK1	-	-	ICIE1	-	-	OCIE1B	OCIE1A	TOIE1
Lê/Escreve		L	L	L/E	L	L	L/E	L/E	L/E
Valor Inicial		0	0	0	0	0	0	0	0

Bit 5 – ICIE1 – *Timer/Counter* 1, *Input Capture Interrupt Enable*
A escrita de 1 neste bit ativa a interrupção por captura da entrada. Quando ocorre uma mudança no pino ICP1 ou nos pinos do comparador analógico, o valor do TCNT1 é salvo no registrador ICR1.

Bit 2 – OCIE1B – *Timer/Counter* 1 *Output Compare Match B Interrupt Enable*
A escrita de 1 neste bit ativa a interrupção do TC1 na igualdade de comparação com o registrador OCR1B.

Bit 1 – OCIE1A – *Timer/Counter* 1 *Output Compare Match A Interrupt Enable*
A escrita de 1 neste bit ativa a interrupção do TC1 na igualdade de comparação com o registrador OCR1A.

Bit 0 – TOIE1 – *Timer/Counter* 1 *Overflow Interrupt Enable*
A escrita de 1 neste bit ativa a interrupção por estouro do TC1.

As interrupções individuais dependem da habilitação das interrupções globais pelo bit I do registrador SREG.

TIFR1 – *Timer/Counter* 1 *Interrupt Flag Register*

Bit	7	6	5	4	3	2	1	0
TIFR2	-	-	ICF1	-	-	OCF1B	OCF1A	TOV1
Lê/Escreve	L	L	L/E	L	L	L/E	L/E	L/E
Valor Inicial	0	0	0	0	0	0	0	0

Bit 5 – ICF1 – *Timer/Counter* 1 *Input Capture Flag*
Este bit é colocado em 1 quando um evento relacionado ao pino ICP1 ocorre.

Bit 2 – OCF1B – *Timer/Counter* 1 *Output Compare B Match Flag*
Este bit é colocado em 1 quando o valor da contagem (TCNT1) é igual ao valor do registrador de comparação de saída B (OCR1B) do TC1.

Bit 1 – OCF1A – *Timer/Counter* 1 *Output Compare A Match Flag*
Este bit é colocado em 1 quando o valor da contagem (TCNT1) é igual ao valor do registrador de comparação de saída A (OCR1A) do TC1.

Bit 0 – TOV1 – *Timer/Counter* 1 *Overflow Flag*
Este bit é colocado em 1 quando um estouro do TC1 ocorre.

9.5.2 CÓDIGOS EXEMPLO

MODO NORMAL – GERANDO UM PEDIDO DE INTERRUPÇÃO

O código abaixo é utilizado para gerar um evento periódico de interrupção a cada 4,19 s. Neste caso, um LED ligado ao pino PB5 troca de estado (liga ou desliga) a cada interrupção. Foi empregado o maior divisor de *clock* possível, 1024. Em relação ao exemplo apresentado na seção 9.3.2 (TC0), percebe-se o aumento considerável no tempo de temporização, pois, desta vez, o TC1 conta 65536 vezes ao invés de apenas 256.

TC1_estouro.c

```
//=============================================================================== //
//       HABILITANDO A INTERRUPÇÃO POR ESTOURO DO TC1                             //
//=============================================================================== //
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#define cpl_bit(y,bit) (y^=(1<<bit))//troca o estado lógico do bit x da variável Y
#define LED     PB5
//-------------------------------------------------------------------------------
ISR(TIMER1_OVF_vect)                //interrupção do TC1
{
     cpl_bit(PORTB,LED);
}
//-------------------------------------------------------------------------------
```

```
int main()
{
     DDRB  = 0b00100000;  //somente pino do LED como saída
     PORTB = 0b11011111;  //apaga LED e habilita pull-ups nos pinos não utilizados

     TCCR1B = (1<<CS12) | (1<<CS10);/*TC1 com prescaler de 1024, a 16 MHz gera uma
                                                              interrupção a cada 4,19 s*/
     TIMSK1 = 1<<TOIE1;   //habilita a interrupção do TC1
     sei()                //habilita interrupções globais

     while(1)
     {
          /*Aqui vai o código, a cada estouro do TC1 o programa desvia para ISR(TIMER1_OVF_vect)*/
     }
}
//============================================================================
```

MODO DE CAPTURA DE ENTRADA

O programa abaixo utiliza um botão para gerar uma interrupção por captura no pino ICP1. Não foi feito o tratamento do ruído do botão e cada vez que o botão é pressionado o pino PB5 troca de estado. O valor do registrador ICR1 é utilizado para ler o valor do TC1.

TC1_captura.c

```
//============================================================================ //
//       HABILITANDO A INTERRUPÇÃO POR CAPTURA                                  //
//============================================================================ //
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#define cpl_bit(y,bit) (y^=(1<<bit))//troca o estado lógico do bit x da variável Y
#define LED          PB5
//----------------------------------------------------------------------------
/*interrupção do TC1, toda vez que ocorrer um evento no pino ICP1 (PB0) ICR1 terá o
valor de contagem do TCNT1*/
//----------------------------------------------------------------------------
ISR(TIMER1_CAPT_vect)
{
     cpl_bit(PORTB,LED); //troca estado do pino PB5
}
//----------------------------------------------------------------------------
int main()
{
     DDRB  = 0b00100000;  //somente pino do LED como saída, botão no PB0
     PORTB = 0b11011111;  //apaga LED e habilita pull-ups nos outros pinos
     TCCR1B = 1<<CS10;    //TC1 com prescaler = 1, captura na borda de descida
     TIMSK1 = 1<<ICIE1;   //habilita a interrupção por captura
     sei();               //habilita interrupções globais

     while(1)
     {    /*Aqui vai o código, a cada evento no pino ICP1 o programa desvia para
                                                                ISR(TIMER1_CAPT_vect)*/
     }
}
//============================================================================
```

MODO CTC

TC1_PWMs.c

```
#define F_CPU 16000000UL
#include <avr/io.h>

int main(void)
{
     DDRB  = 0b00000110;  //pinos OC1B e OC1A (PB2 e PB1) como saída
     PORTB = 0b11111001;  //zera saídas e habilita pull-ups nos pinos não utilizados

     //MODO CTC - TOP = ICR1
     TCCR1A = 0b01010000;/*habilita OC1A e OC1B para trocar de estado na igualdade de
                                                                          comparação*/
     TCCR1B = 0b00011011; //liga TC1 com prescaler = 64.
     ICR1 = 10000;        //valor máximo de contagem

     while(1)
     {}                   //o programa principal vai aqui
 }
```

MODO PWM RÁPIDO

```
...
//fast PWM, TOP = ICR1, OC1A e OC1B habilitados
TCCR1A = 0b10100010;      //PWM não invertido nos pinos OC1A e OC1B
TCCR1B = 0b00011001;      //liga TC1, prescaler = 1
ICR1 = 35000;             //valor máximo para contagem
OCR1A = 2000;             //controle do ciclo ativo do PWM 0C1A
OCR1B = 100;              //controle do ciclo ativo do PWM OC1B

...
```

MODO PWM COM FASE E FREQUÊNCIA CORRIGIDAS

```
...
//phase and frequency correct PWM, TOP = OCR1A
TCCR1A = 0b00100011;      // OC1B habilitado, modo não invertido
TCCR1B = 0b00011001;      //liga TC1, prescaler = 1 e habilita
OCR1A = 300;              //máximo valor para contagem
OCR0B = 100;              //controle do ciclo ativo do PWM OC1B

...
```

MODO CTC – CONTANDO EVENTOS EXTERNOS

Abaixo, é apresentado um programa para gerar uma interrupção a cada 1 s baseado num sinal de 60 Hz externo ao microcontrolador. Esse tipo de sinal, por exemplo, é utilizado em rádios-relógio para gerar a base

de tempo. Na fig. 9.12, é apresentado o circuito para o programa. O sinal de 60 Hz é digital, sendo, usualmente obtido da rede elétrica após um tratamento adequado.

TC1_externo.c

```
//=============================================================================== //
//          TC1 estouro na igualdade de comparação  - sinal externo               //
//                       Pisca LED a cada 1 segundo                               //
//=============================================================================== //
#include <avr/io.h>
#include <avr/interrupt.h>

#define cpl_bit(Y,bit_x) (Y^=(1<<bit_x))
#define LED          PB5
//-------------------------------------------------------------------------------
ISR(TIMER1_COMPA_vect)    //sub-rotina de interrupção por igualdade de comparação
{
     cpl_bit(PORTB,PB5);//troca o estado do LED do pino PB5
}
//-------------------------------------------------------------------------------
int main()
{
     DDRD = 0x00;         //PORTD será a entrada do sinal de clock para o TC1 (PD3)
     DDRB  = 1<<PB5;      //pino PB5 é a saída para o LED     de sinalização

     TIMSK1 = 1<<OCIE1A;  //habilita a interrupção do TC1 por igualdade de comparação
     TCCR1B = (1<<WGM12)|(1<<CS12) | (1<<CS11) |(1<<CS10);/*clock externo  contagem na
                                                          borda de subida - modo CTC*/
     OCR1A = 59;     /*valor para a contagem máxima do TC1 (conta 60 vezes) – valor de
                         comparação. Como o sinal de clock externo é de 60 Hz, é gerada
                                                          uma interrupção a cada 1 s*/
     sei();               //liga a interrupção

    while(1){}
}
//===============================================================================
```

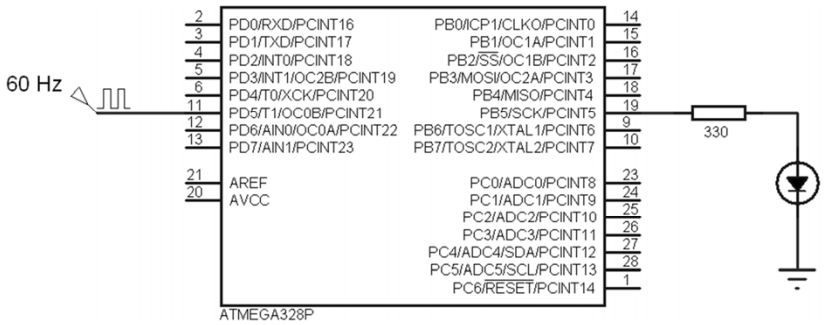

Fig. 9.12 – Sinal de 60 Hz para ligar um LED a cada 1 s.

Exercícios:

9.9 – Considerando o ATmega328 rodando a 16MHz e o TC1 operando no modo normal (65536 contagens), quanto tempo levará para ocorrer um estouro do TC1 para cada *prescaler* possível?

9.10 – Elaborar um programa para que o ATmega328 funcione como frequencímetro digital, apresentando o valor lido em um LCD 16×2. Empregue um circuito adequado (sugestão na fig. 9.13).

Sugestão: utilize um TC de 8 bits para gerar uma base de tempo de 1 s e o TC1, de 16 bits, para contar os eventos externos. A frequência será dada pelo número de contagem do TC de 16 bits (se houver estouro, o programa deve levar esse fato em conta no cômputo da frequência).

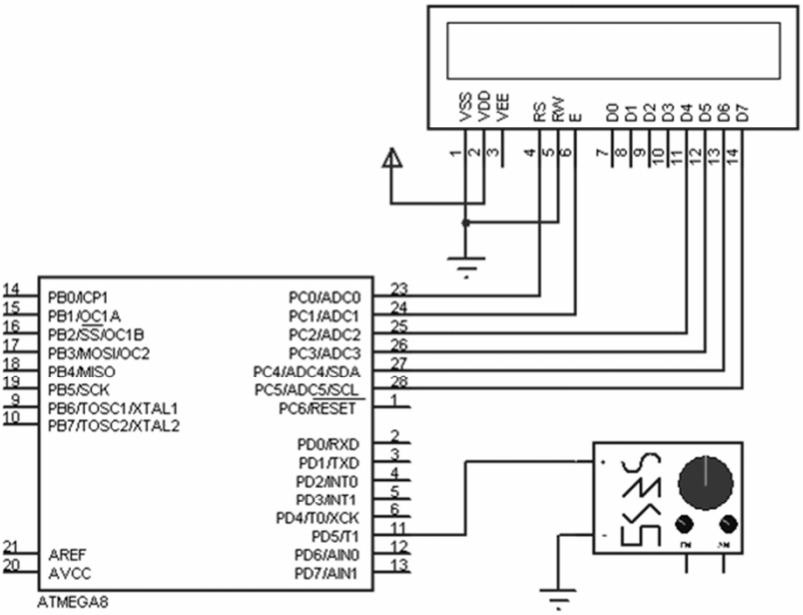

Fig. 9.13 – Circuito para o exercício 9.10 (obs.: a pinagem do ATmega8 é igual a do ATmega328).

9.6 PWM POR SOFTWARE

Algumas vezes pode ser necessário criar sinais PWM unicamente através da programação. Isso ocorre quando o microcontrolador não dispõe do número desejado de canais PWM, ou os mesmos não apresentam a resolução necessária. No ATmega328 estão disponíveis 6 sinais PWMs, dos quais, 4 são de 8 bits e 2 de 16 bits. Porém, os pinos que podem gerar esses sinais são estáticos (não podem ser alterados), não permitindo muita flexibilidade na hora de projetar o hardware.

Se for inviável substituir o microcontrolador por um que satisfaça os requisitos de projeto, o problema deve ser resolvido por programação. A ideia é simples: utilizar um temporizador para gerar a base de tempo para o sinal PWM, cuja resolução ficará dependente das variáveis utilizadas e do número de contagens. O programa a seguir ilustra essa ideia.

PWM_software.c

```
//============================================================================== //
//       PWM borda simples - não invertido, via software                        //
//============================================================================== //
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>

#define   set_bit(y,bit)  (y|=(1<<bit))  //coloca em 1 o bit x da variável Y
#define   clr_bit(y,bit)  (y&=~(1<<bit)) //coloca em 0 o bit x da variável Y

#define PWM1         PB0       //escolha do pino para o sinal PWM1
#define Resol_PWM1   1000      //PWM1 com 1000 passos temporais de resolução

volatile unsigned int Passo_PWM1 = 0;
volatile unsigned int Ciclo_Ativo_PWM1;

//------------------------------------------------------------------------------
ISR(TIMER0_OVF_vect)//o tempo de estouro do TC0  determina a menor resolução temporal
{                   //para o PWM (ciclo ativo)
     Passo_PWM1++; //incremento do passo de tempo

     if(Passo_PWM1==Resol_PWM1)
     {
          Passo_PWM1=0;      //inicializa o contador
          set_bit(PORTB,PWM1);//na igualdade de comparação coloca o pino PWM1 em 1
     }

     if(Passo_PWM1==Ciclo_Ativo_PWM1)
          clr_bit(PORTB,PWM1); /*quando o contador atinge o valor do ciclo ativo do
                                                              PWM1 o pino vai a zero*/
}
//------------------------------------------------------------------------------
```

```
int main()
{
     DDRB  = 0b00000001;//somente o pino do LED como saída
     PORTB = 0b11111110;//apaga o LED e habilita pull-ups nos pinos não utilizados

     TCCR0B = 1<<CS00;//TC0 com prescaler de 1, a 16 MHz gera uma interrupção a cada 16 us
     TIMSK0 = 1<<TOIE0;//habilita a interrupção do TC0
     sei();            //habilita interrupções globais

     Ciclo_Ativo_PWM1 = 500;//determinação do ciclo ativo para o PWM1

     while(1)
     {}              //Aqui vai o código principal

}
//============================================================================
```

O programa acima utilizou uma resolução de 1000 pontos (Resol_PWM1) e o TC0 foi configurado para trabalhar sem divisão de *clock*. Desta forma, o tempo de estouro para um frequência de trabalho de 16 MHz foi de 16 µs (256/16MHz). Isso significa que a cada 16 us é gerada uma interrupção, na qual é executada a rotina para gerar o sinal PWM. Considerando o número de pontos do sinal PWM e a menor resolução temporal possível para o ciclo ativo, resulta em um período de 16 ms para o sinal, ou seja, uma frequência de 62,5 Hz. O ciclo ativo do sinal é controlado por uma variável própria e é escolhido qual pino disponibilizará o sinal PWM.

Para alterar a frequência do sinal PWM é necessário alterar a sua resolução, ou seja, o tempo de estouro do TC0. Lembrando-se que é possível configurar diversos modos de trabalho para os TCs no ATmega.

Exercício

9.11 – Faça um programa para gerar 4 sinais PWM com resolução de 100, 400, 1000 e 5000 pontos. Escolha a forma de trabalho do TC empregado.

Qual é a frequência dos sinais gerados se for empregado uma frequência para o trabalho da CPU de 20 MHz?

9.7 ACIONANDO MOTORES

Nesta seção, são apresentados alguns conceitos básicos de motores comumente utilizados com circuitos eletrônicos de baixa potência, os quais podem ser acionados diretamente por chaves transistorizadas ou *drivers* de corrente de baixa potência. Uma abordagem profunda está longe do escopo deste trabalho. Recomenda-se, assim, que bibliografias especializadas sejam consultadas para o completo entendimento do funcionamento dos referidos motores.

9.7.1 MOTOR DC DE IMÃ PERMANENTE

Os motores DC de imã permanente são muito utilizados em equipamentos eletrônicos, sendo encontrados em aparelhos de CD/DVD, *Blue Ray*, computadores (ventiladores) e em muitos dispositivos que exigem movimento de partes mecânicas sem grande precisão. Para aplicações eletrônicas, geralmente operam com baixas tensões, até uns 12 V, e possuem pequenas dimensões. Alguns tipos de motores DC são apresentados na fig. 9.14.

Fig. 9.14 – Motores DC encontrados em aparelhos de CD/DVD de computadores.

O movimento dos motores DC é baseado na interação entre o campo magnético de seu(s) imã(s) permanente(s) e o campo magnético gerado pela corrente que percorre suas bobinas. Sua rotação é dependente da tensão média aplicada aos seus terminais. Dessa forma, com o emprego de um sinal PWM, é possível controlar a velocidade de um motor DC.

Para a inversão do sentido de rotação do motor é necessário inverter a tensão de alimentação do motor e, logo, sua corrente. Para alcançar tal funcionalidade deve ser empregado um circuito com transistores funcionando como chaves, a chamada ponte H, ilustrada na fig. 9.15. Para o motor girar para a direita devem ser acionados os transistores Q1 e Q2, então a corrente terá o sentido de I_D; para girar para a esquerda, devem ser acionados os transistores Q3 e Q4, gerando a corrente I_E (maiores detalhes sobre chaves transistorizadas podem ser vistos no apêndice D).

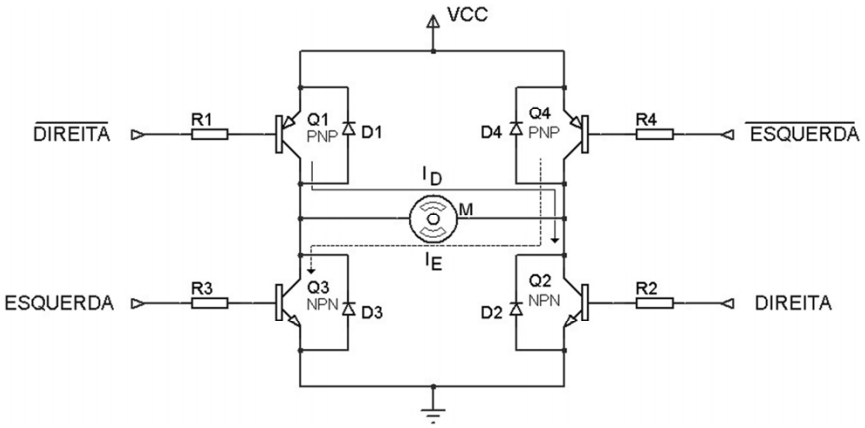

Fig. 9.15 – Ponte H transistorizada para o controle do sentido de rotação de um motor DC.

Um circuito de controle é necessário para gerar adequadamente os sinais para a ponte H, geralmente comandado por um microcontrolador.

A ponte H apresentada na fig. 9.15 é um tanto simplificada, as tensões de controle devem ser adequadas ao valor da tensão de

alimentação do motor. Circuitos adaptadores (*buffers*) podem ser necessários para fazer o casamento entre os sinais digitais de controle e os níveis de tensão da ponte H. Da mesma forma, de acordo com as potências envolvidas, as chaves transistorizadas deverão ser adequadamente projetadas.

Exercícios:

9.12 – Utilizando o circuito da fig. 9.16, elaborar um programa para controlar, através de um sinal PWM, um motor DC com 256 níveis de velocidade. O nível da velocidade selecionado deve ser apresentado no *display* (valor hexadecimal) e armazenado na memória EEPROM para a inicialização do motor. O display de 7 segmentos deve apresentar um número entre 00 e FF.

Obs.: é aconselhado que a alimentação do motor seja independente do circuito de controle, pois podem haver picos de corrente prejudiciais. No caso do microcontrolador, ele pode ser reinicializado. Tudo vai depender das características da fonte de alimentação

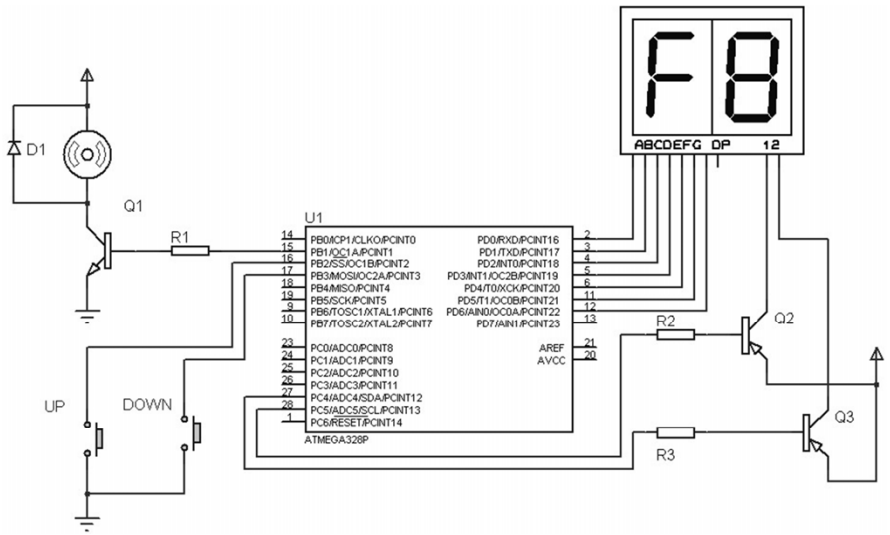

Fig. 9.16 – Controle de um motor DC com um sinal PWM.

9.13 – Existem circuitos integrados específicos para o controle do sentido de rotação de motores DC, tal como o L298. Consulte o manual do fabricante para entender o seu funcionamento. Quais seriam os pinos do L298 onde um sinal PWM poderia ser empregado?

Para o Arduino, existem módulos prontos para o trabalho com motores DC, como o apresentado na fig. 9.17, o qual emprega o L298 e o ULN2803.

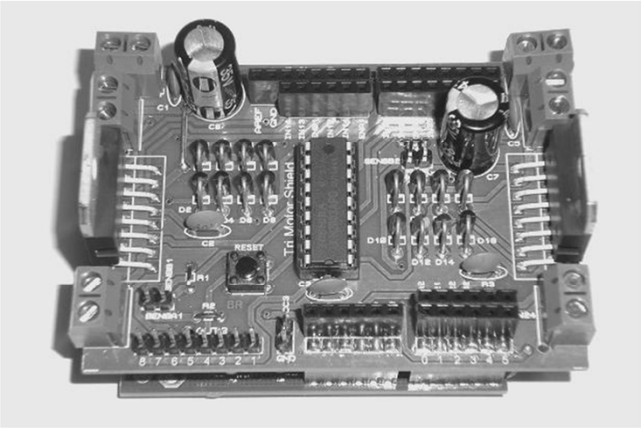

Fig. 9.17 – Módulo para controle de motores para o Arduino[29].

9.7.2 MICRO SERVO MOTOR

Os servo motores apresentam movimentação precisa do seu eixo de acordo com ângulos precisos de movimento, sendo adequados para o posicionamento de partes mecânicas. Em eletrônica, são encontrados em aeromodelismo e outros dispositivos mecânicos de controle remoto, sendo chamados de servo motores, servo para robista, ou micro servo, dado suas pequenas dimensões. São constituídos por um motor DC de imãs permanentes, um circuito de controle interno, incluindo os *drivers* necessários para acionar o motor e , ainda, por engrenagens de redução. Essas características os tornam compactos, robustos e fáceis de usar. Na fig. 9.18, é possível ver um micro servo. À direita nessa figura, ele está

[29] Circuito esquemático, PCB e demais detalhes podem ser encontrados em www.borgescorporation.blogspot.com

aberto, permitindo a visualização de suas partes constituintes: motor DC, circuito de controle e engrenagens.

Fig. 9.18 – Micro servo motor.

Os servo motores possuem 3 fios: dois de alimentação e um de controle. O interessante é que o ângulo de giro do motor, a saída do sistema de engrenagens, é determinado pela largura do ciclo ativo de um sinal PWM aplicado no controle, como exemplificado na fig. 9.19. O período do sinal é de 20 ms (50 Hz) e o ciclo ativo geralmente pode variar de 0,5 ms até 2,5 ms (vai depender das especificações do fabricante).

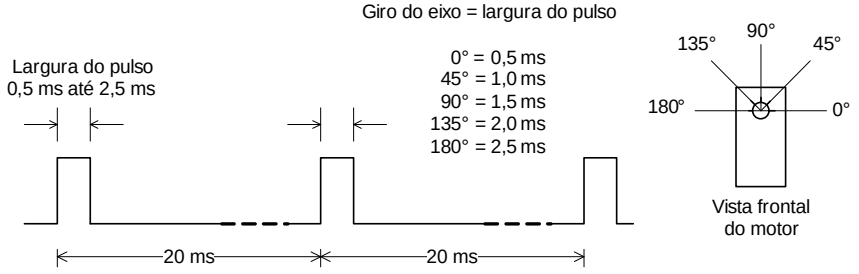

Fig. 9.19 – Sinal de controle para um micro servo motor.

Com base no sinal de controle do servo é possível utilizar o ATmega328 para gerar o sinal PWM adequado, conforme exemplificado pelo programa a seguir, empregando o TC1.

Controle_servo_motor.c

```
/*--------------------------------------------------------------------------------
     EXEMPLO DO CONTROLE DE DOIS MOTORES SERVO COM SINAIS PWM
     Uso do TC1 no modo PWM rápido, não invertido, pinos OC1A e OC1B
     Valor TOP de contagem em ICR1
--------------------------------------------------------------------------------*/
#define F_CPU 16000000UL
#include <avr/io.h>

//Definições de macros
#define   set_bit(adress,bit)  (adress|=(1<<bit))
#define   clr_bit(adress,bit)  (adress&=~(1<<bit))

#define TOP 39999    //valor para a máxima contagem

int main()
{
     DDRB  = 0b00000110;//habilita os pinos OC1A e OC1B (PB1 e PB2) como saídas
     PORTB = 0b11111001;
                    //TOP = (F_CPU/(N*F_PWM))-1, com N = 8 e F_PWM = 50 Hz
     ICR1 = TOP;     //configura o período do PWM (20 ms)

     // Configura o TC1 para o modo PWM rápido via ICR1, prescaler = 8
     TCCR1A = (1 << WGM11);
     TCCR1B = (1 << WGM13) | (1<<WGM12) | (1 << CS11);

     set_bit(TCCR1A,COM1A1);   //ativa o PWM no OC1B, modo de comparação não-invertido
                               //para desabilitar empregar clr_bit(TCCR1A, COM1A1)
     set_bit(TCCR1A,COM1B1);   //ativa o PWM no OC1A, modo de comparação não-invertido
                               //para desabilitar empregar clr_bit(TCCR1A, COM1B1)
     //Pulso de 2 ms em OC1A
     OCR1A = 4000;  /*regra de três para determinar este valor: ICR1(TOP) = 20 ms,
                                                               OCR1A (4000) = 2 ms)*/

     //Pulso de 1 ms em OC1B
     OCR1B = 2000;  /*regra de três para determinar este valor: ICR1(TOP) = 20 ms,
                                                               OCR1B (2000) = 1 ms)*/
     while(1)
     {}//programa principal
}
//--------------------------------------------------------------------------------
```

Dada as características do servo motor: controle e engrenagens, alguns projetistas gostam de empregá-los para substituir os motores DC em algumas aplicações, como por exemplo, em pequenos robôs. Existem motores servo que não possuem limite de giro (ângulo) e podem ser acionados continuamente. Todavia, é possível alterar os servos comuns

para que executem giros contínuos. A inversão de giro é feita com a alteração da largura do pulso de controle.

Exercício:

9.14 – Elaborar um programa para controlar o ângulo de giro de um micro servo motor de acordo com dois botões, fig. 9.20. Começando com um pulso de 0,5 ms até 2,5 ms, com passo de incremento de 0,1 ms. Repita o procedimento para um passo de 0,05 ms. Qual a relação angular de giro com a largura de pulso?

Obs.: é aconselhado que a alimentação do motor seja independente do circuito de controle, pois podem haver picos de corrente prejudiciais, no caso do microcontrolador, ele pode ser inicializado. Tudo vai depender das características da fonte de alimentação.

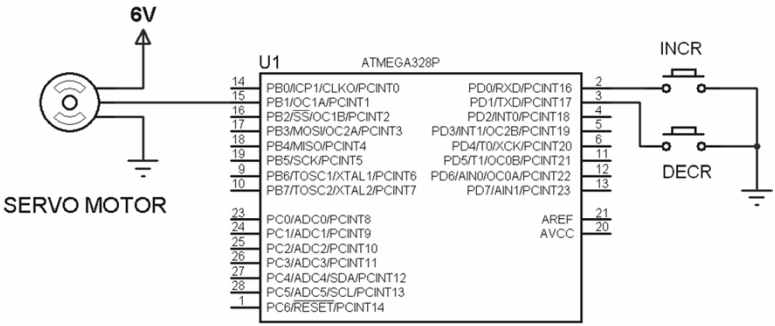

Fig. 9.20 – Circuito para o acionamento de um servo motor.

9.7.3 MOTOR DE PASSO UNIPOLAR

O motor de passo produz uma rotação precisa em seu eixo com passos angulares discretos de acordo com a sequência de energização de suas bobinas. É muito utilizado em sistemas que necessitam de um posicionamento preciso de algum componente mecânico, como por exemplo, impressoras e robôs. A seguir, será descrito o motor de passo do tipo unipolar com rotor de ímã permanente. Na fig.9.21, alguns motores de

passo são apresentados. O motor aberto é de um antigo *driver* de disco de 5 ¼ de um computador e o outro o motor de uma impressora jato de tinta.

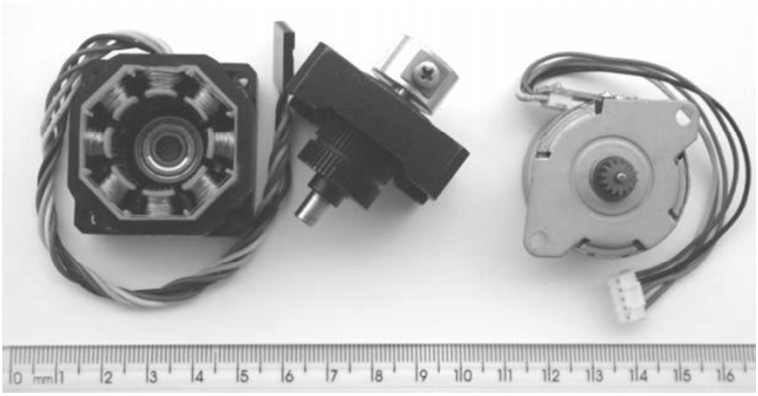

Fig. 9.21 – Motores de passo unipolar.

Na fig. 9.22a, é ilustrada a estrutura de um motor de passo unipolar de 4 fases; na fig. 9.22b, uma forma mais simplificada de representação. O princípio de funcionamento do motor de passo é simples: de acordo com as bobinas energizadas, o rotor irá girar a passos discretos, alinhando seu campo magnético com o campo magnético produzido pelas bobinas.

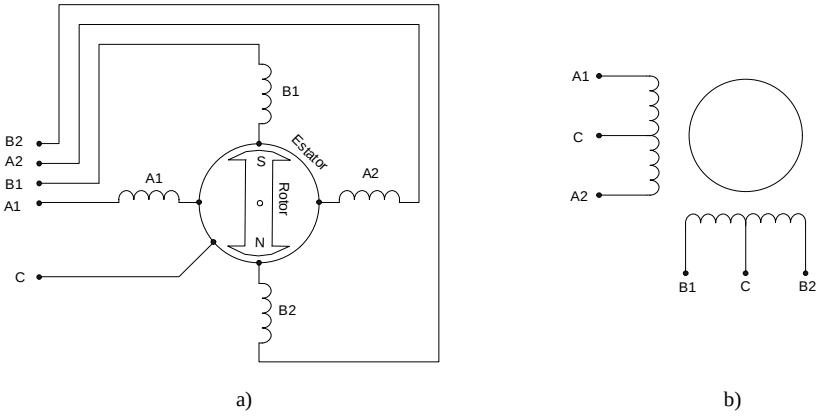

Fig. 9.22 – Controle de um motor de passo unipolar, a) diagrama de um motor de passo de 4 fases, b) diagrama simplificado.

O controle do motor de passo é feito com a sequência correta de energização das bobinas. Podem ser acionadas uma ou mais bobinas ao mesmo tempo. Com o acionamento de duas bobinas simultaneamente, existe a possibilidade de se obter meio passo de giro ou um torque maior. Nas fig. 9.23a e 9.23b, as sequências de energização para um passo por vez são apresentadas, com o emprego de uma e duas bobinas, respectivamente. O acionamento de duas bobinas ao mesmo tempo consome duas vezes mais corrente e, assim, fornece um torque maior. Na fig. 9.24, a sequência para meio passo de rotação é apresentada.

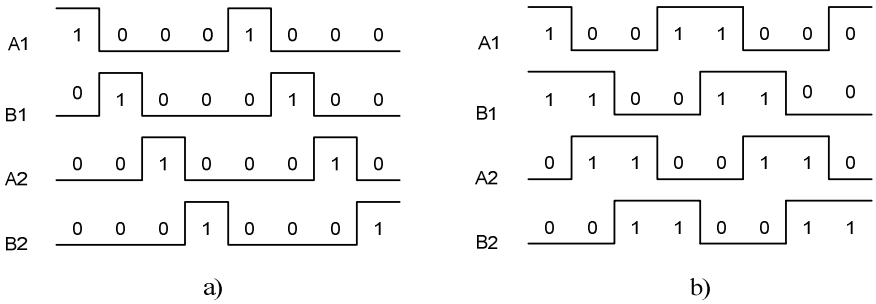

Fig. 9.23 – Sequência de acionamento das bobinas de um motor de passo unipolar para um passo. a) um bobina por vez, b) duas bobinas por vez.

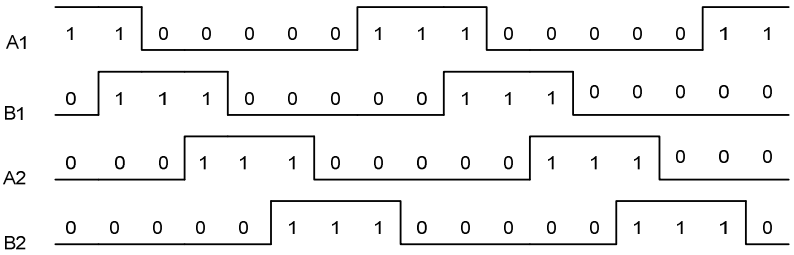

Fig. 9.24 – Sequência de acionamento das bobinas de um motor de passo unipolar para meio passo.

Para o acionamento de um motor de passo, é necessário empregar uma interface (*driver* de corrente) entre o sistema microcontrolado e o

motor para que as correntes exigidas de operação possam ser fornecidas (apêndice D). Geralmente, o ponto comum das bobinas é ligado ao terminal positivo da tensão de alimentação e as bobinas são acionadas quando aterradas.

Outro tipo de motor de passo é o bipolar, que difere do anterior por não conter o ponto comum de ligação entre as bobinas, exigindo, assim, um circuito de controle bem mais complexo.

Exercícios:

9.15 – Elaborar um programa para controlar o motor de passo unipolar da fig. 9.25. Dois botões controlam a direção de rotação e outros dois, a velocidade. Primeiro, deve-se consultar o manual do fabricante do ULN2803 para entender suas características e funcionamento.

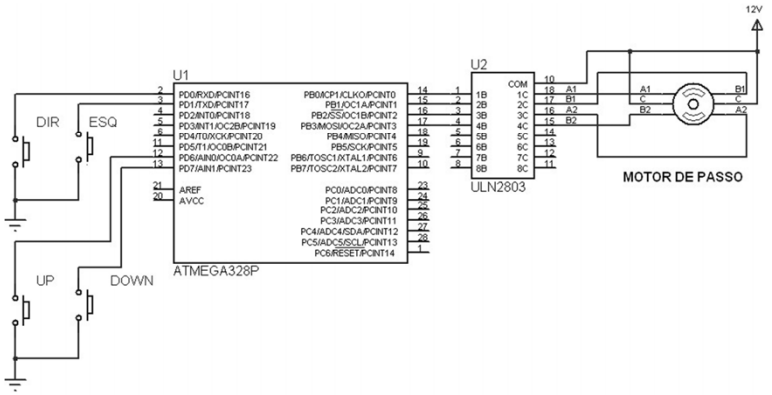

Fig. 9.25 – Controle de um motor de passo unipolar.

9.16 – Como funciona um motor de passos bipolar e como deve ser o seu circuito de controle?

9.8 MÓDULO DE ULTRASSOM - SONAR

Com a crescente popularização da eletrônica, foram disponibilizados módulos sensores prontos para serem utilizados com microcontroladores; um deles, é o de ultrassom. Eles geram um sinal sonoro acima da percepção humana (que é de 20 kHz) e determinam o tempo que esse sinal leva para alcançar e retornar ao encontrar um obstáculo, permitindo dessa forma, determinar a distância do sensor ao obstáculo. Na fig. 9.26, é apresentado um módulo desses, usualmente encontrado no mercado mundial, o HC-SR04. Esse módulo possui toda a eletrônica necessária ao seu funcionamento, bastando ao microcontrolador ordenar a medição da distância e interpretar o sinal de retorno. Possui 4 pinos, dois de alimentação (5 V e GND), um para o acionamento do sensor e outro para a leitura do sinal que indica a distância do objeto. Na fig. 9.27, os sinais de trabalho do módulo são apresentados.

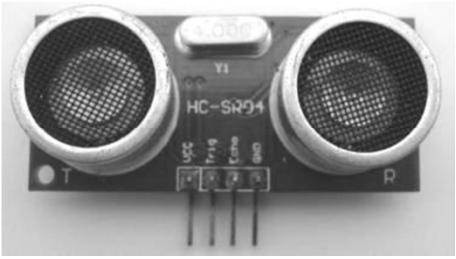

Fig. 9.26 – Módulo de ultrassom HC-SR04.

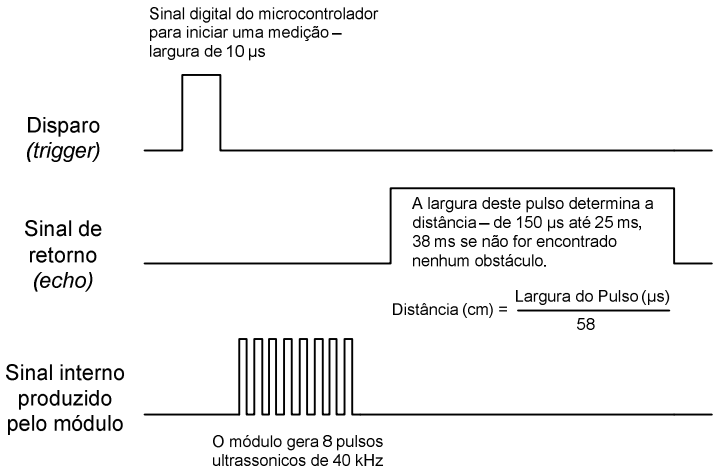

Fig. 9.27 – Sinais de trabalho do módulo HC-SR04.

É necessário enviar um pulso de 10 µs ao módulo para iniciar uma medição. Após isso, o módulo irá gerar 8 pulso ultrassônicos de 40 kHz para avaliar a distância dele em relação ao objeto, produzindo um sinal de retorno, cuja largura indica a distância do sensor ao objeto. Esse valor estará compreendido entre 150 µs e 25 ms, caso nenhum obstáculo seja encontrado a largura do pulso será de 38 ms. O tempo entre as conversões deve ser de no mínimo 50 ms. O fabricante do módulo especifica que a distância medida deve estar compreendida entre 2 cm e 5 m, que a medição apresenta uma resolução de 0,3 cm e que o ângulo ativo de medição é de 15°.

Dadas as características do sinal retornado pelo módulo de ultrassom, para o ATmega328, o TC1 é o mais adequado para medir o período desse sinal, como exemplificado pelo programa a seguir. Na fig. 9.28 é apresentado o circuito empregado.

Sonar.c

```
//=============================================================================== //
//             Programa para teste do módulo Sonar HC-SR04                       //
//=============================================================================== //
#include "def_principais.h"    //inclusão do arquivo com as principais definições
#include "LCD.h"

#define DISPARO PB1

unsigned int Inicio_Sinal, Distancia;
prog_char mensagem1[] = "Distanc =     cm\0";
prog_char mensagem2[] = "xxx\0";
//-------------------------------------------------------------------------------------
ISR(TIMER1_CAPT_vect)          //interrupção por captura do valor do TCNT1
{
     cpl_bit(TCCR1B,ICES1);    //troca a borda de captura do sinal

     if(!tst_bit(TCCR1B,ICES1))//lê o valor de contagem do TC1 na borda de subida do sinal
          Inicio_Sinal = ICR1;//salva a primeira contagem para determinar a largura do pulso
     else                     //lê o valor de contagem do TC1 na borda de descida do sinal
          Distancia = (ICR1 - Inicio_Sinal)/58;/*agora ICR1 tem o valor do TC1 na borda de
                                                 descida do sinal, então calcula a distância */
}
//-------------------------------------------------------------------------------------
int main()
{
     unsigned char digitos[tam_vetor];//declaração da variável para armazenagem dos digitos

     DDRD  = 0xFF;
     DDRB  = 0b00000010;//somente pino de disparo como saída (PB1), captura no PB0 (ICP1)
     PORTB = 0b11111101;

     TCCR1B = (1<<ICES1)|(1<<CS11); //TC1 com prescaler = 8, captura na borda de subida
     TIMSK1 = 1<<ICIE1;        //habilita a interrupção por captura
     sei();                    //habilita a chave de interrupções globais


     inic_LCD_4bits();
     escreve_LCD_Flash(mensagem1);

     while(1)
     {
          //pulso de disparo
          set_bit(PORTB,DISPARO);
          _delay_us(10);
          clr_bit(PORTB,DISPARO);

          cmd_LCD(0x8A,0);
          if(Distancia<431)//se o pulso for menor que 25 ms mostra o valor da distância
          {
               ident_num(Distancia, digitos);
               cmd_LCD(digitos[2],1);
               cmd_LCD(digitos[1],1);
               cmd_LCD(digitos[0],1);
          }
          else //senão escreve xxx no valor
               escreve_LCD_Flash(mensagem2);

          _delay_ms(50);//mínimo tempo para uma nova medida de distância
    }
}
//===============================================================================
```

Os arquivos de inclusão no início do código foram apresentados no capítulo 5. Entretanto, no arquivo **def_principais.h** foi incluída a biblioteca para tratamento das interrupções (`#include <avr/interrupt.h>`).

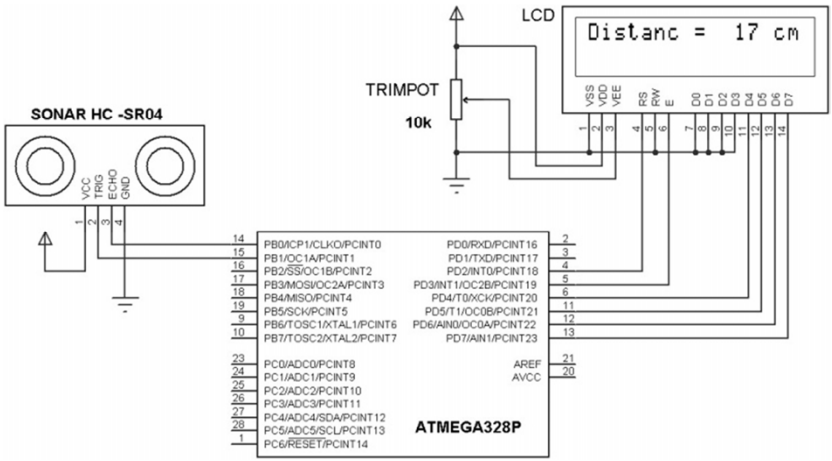

Fig. 9.28 – Circuito para teste do sonar.

Para medir a largura do pulso gerado pelo módulo, utiliza-se a interrupção por captura do TC1. Quando o sinal na entrada do pino ICP1 passa de 0 para 1 (borda de subida), é gerado um pedido de interrupção, onde se armazena o valor de contagem do TC1. Então, altera-se a interrupção por captura para ser gerada na borda de descida do sinal. Quando essa ocorre, o novo valor do TC1 é subtraído do primeiro, que foi coletado na borda de subida. Dessa forma, determina-se quantas contagens foram realizadas no ciclo ativo do sinal e pode-se avaliar a largura do sinal retornado pelo sensor.

Os testes práticos mostraram que o sonar utilizado não é muito preciso, porém apresenta um estimativa razoável da distância.

Exercício:

9.17 – Qual a resolução temporal empregada para o TC1 do programa de trabalho com o módulo sonar HC-SR04? Qual sua relação com a resolução do módulo?
Por que o limite para a apresentação da distância foi de 431 cm?
