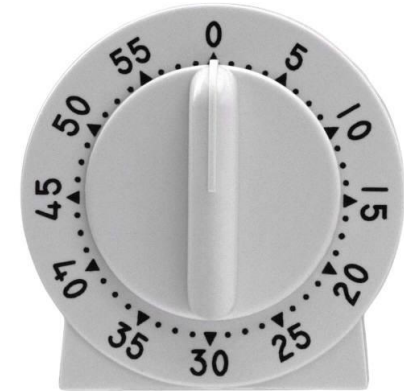


TEMPORIZADORES E CONTADORES



Temporizadores / Contadores

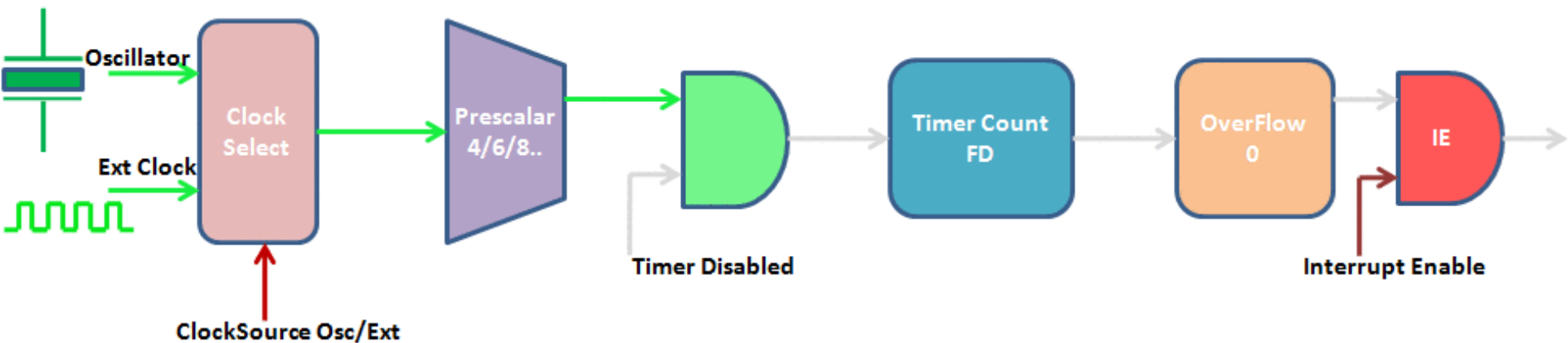
- A maioria dos microcontroladores inclui algum tipo de sistema de temporizador. Isto facilita o monitoramento e controle em tempo real:



1. Medindo o tempo entre duas ocorrências de eventos
 - Velocidade do motor com base no tempo para uma revolução completa
2. Invocar uma ação em intervalos **precisos**
 - Injetar combustível na câmara de combustão de um motor
3. Medir o número de eventos que ocorrem dentro de um intervalo de tempo específico
 - Número de falhas de ignição do motor
4. Gerando uma forma de onda em uma frequência específica

Temporizador / Contador - FUNCIONAMIENTO BÁSICO -

Timer Block Diagram



ExploreEmbedded

Atmega328 - Temporizador / Contador

- Subsistema com 3 unidades de T/C -

- Timer/Contador 0
 - 8-bit timer
- Timer/Contador 1
 - 16-bit timer
- Timer/Contador 2
 - 8-bit timer



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) - 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 de **8 bits** e com características são similares ao TC0.

Entretanto, apresenta uma função especial para a contagem precisa de 1s, permitindo o uso de um cristal

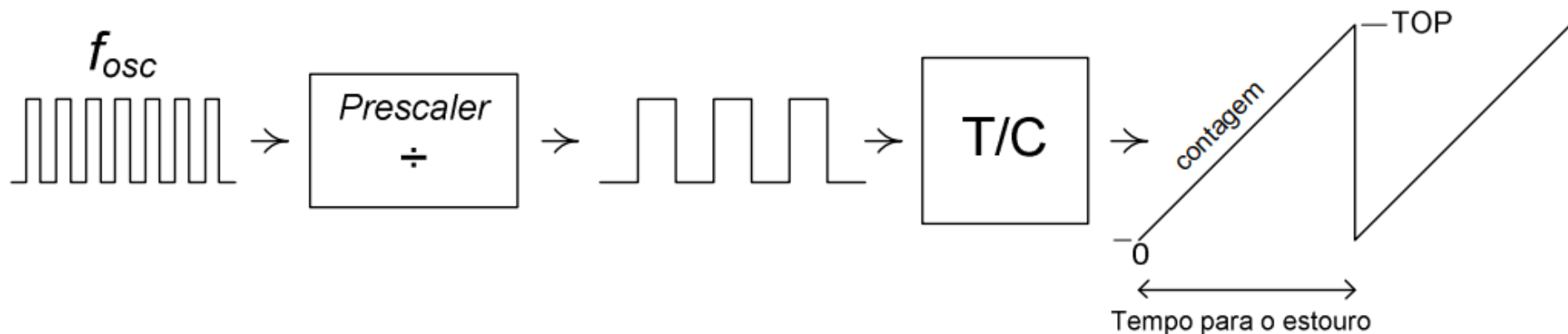
externo independente para o seu *clock* (32,768 kHz).

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)
- Gerador de frequência (onda quadrada).
- 4 fontes independentes de interrupção (por estouro e igualdades de comparação)

O funcionamento de um TC para o ATmega é ilustrado abaixo:



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:

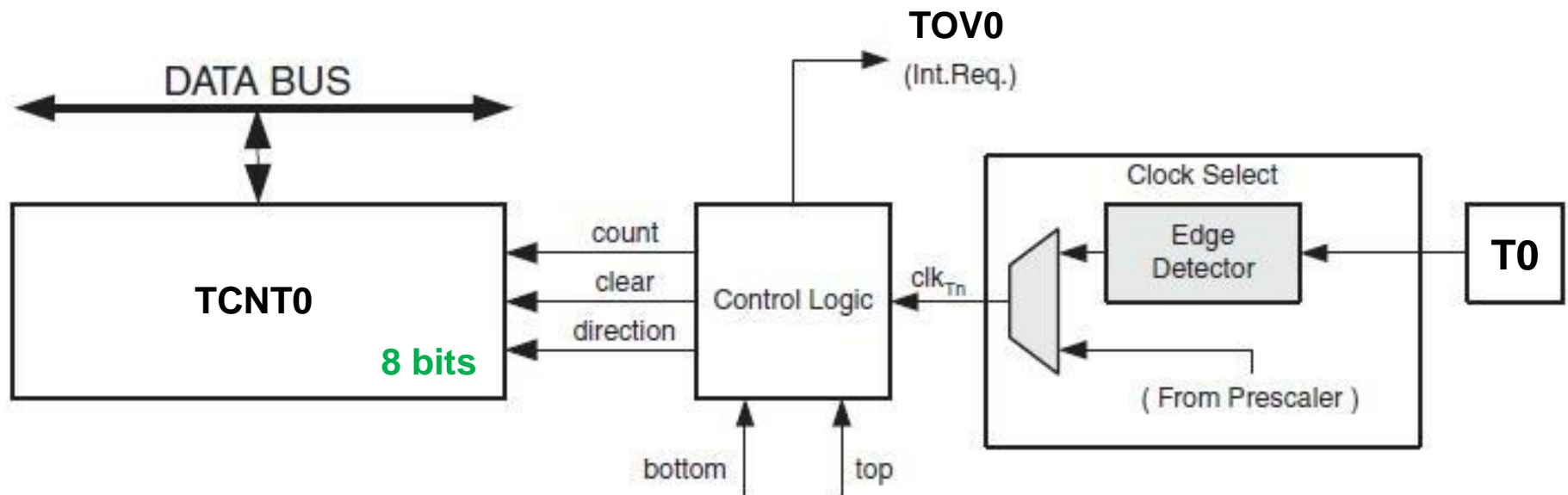
$$t_{estouro} = \frac{(TOP+1) \times prescaler}{f_{osc}}$$

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

Temporizador/Contador 0 (TC0)

O TC0 é um contador de 8 bits, incrementado com pulsos de *clock* (interno ou externo).

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.



MODOS DE OPERAÇÃO

MODO NORMAL

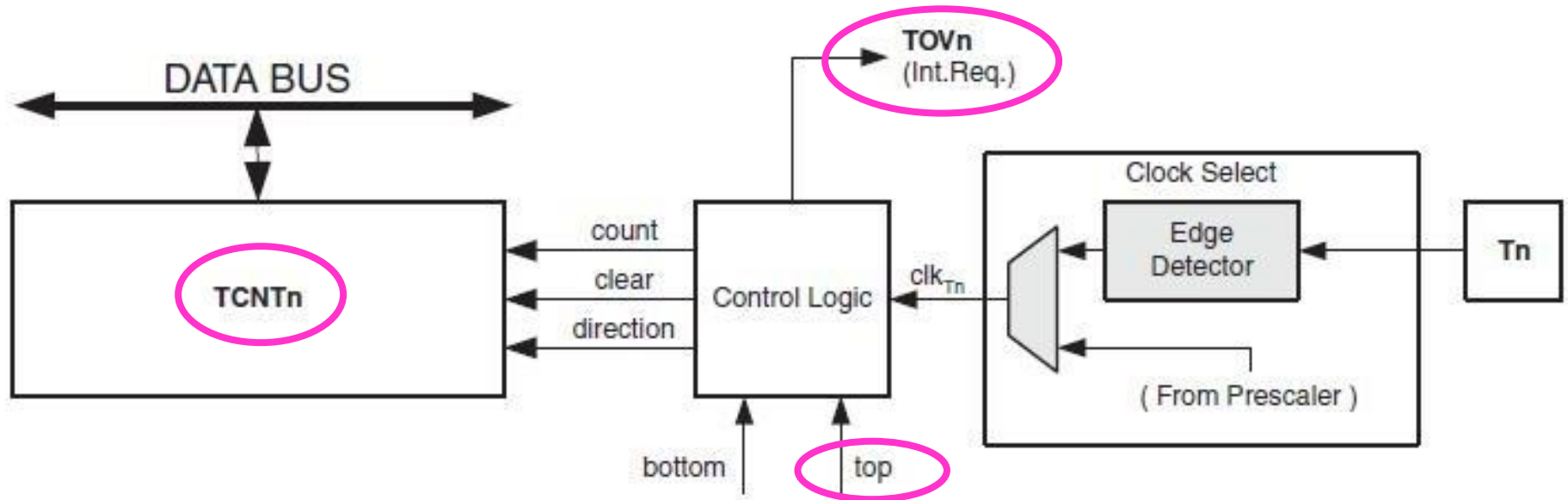
É o modo mais simples de operação, onde o TC0 conta continuamente de forma crescente.

A contagem se dá de 0 até 255 voltando a 0, num ciclo contínuo.

Quando a contagem passa de 255 para 0, ocorre o OVERFLOW (estouro) e então, o bit sinalizador de estouro (TOV0) é setado (colocado em 1). Se habilitado, 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.

Funcionamento geral do Temporizador/Contador

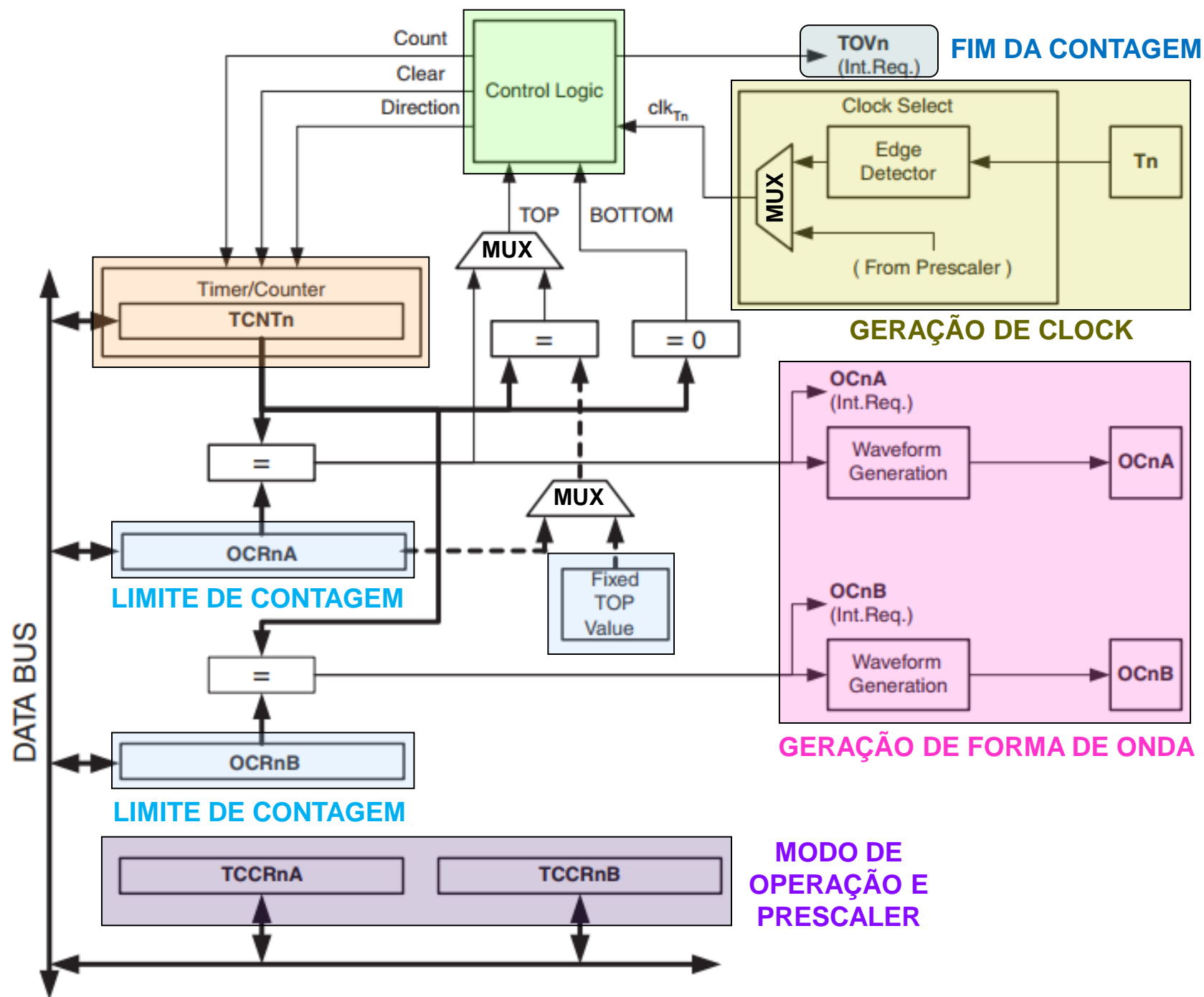


A fonte de *clock* (interna ou externa) envia pulsos ao *prescaler*, que os divide por uma determinada quantidade.

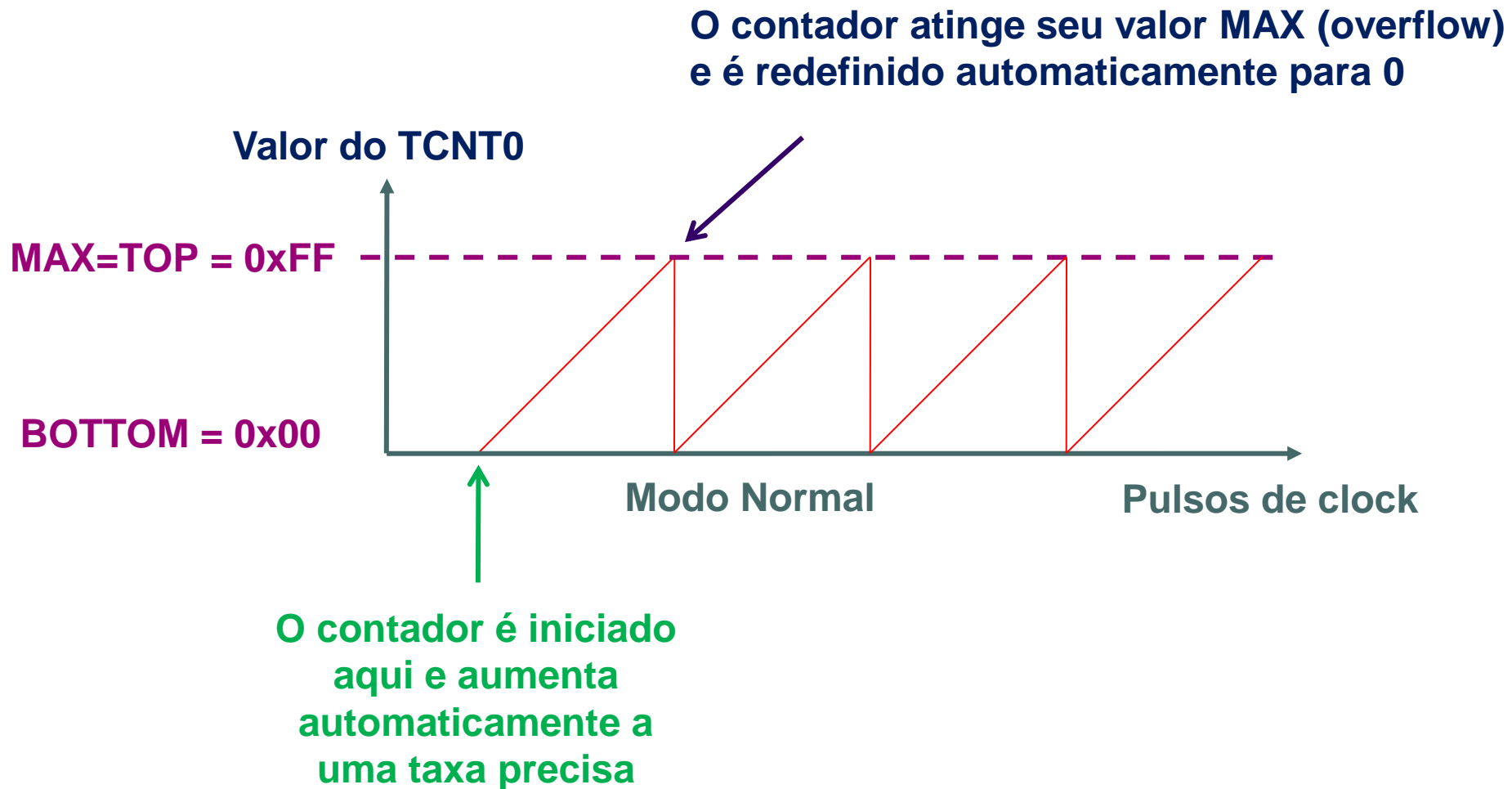
Esta entrada é enviada ao circuito de controle que incrementa o registro **TCNTn**.

Quando o registrador atinge seu valor **TOP**, ele é zerado e envia um sinal **TOVn** (estouro do temporizador) que pode ser usado para disparar uma interrupção.

O valor máximo de contagem para um contador de 8 bits é 255 contador de 16 bits é 65535.

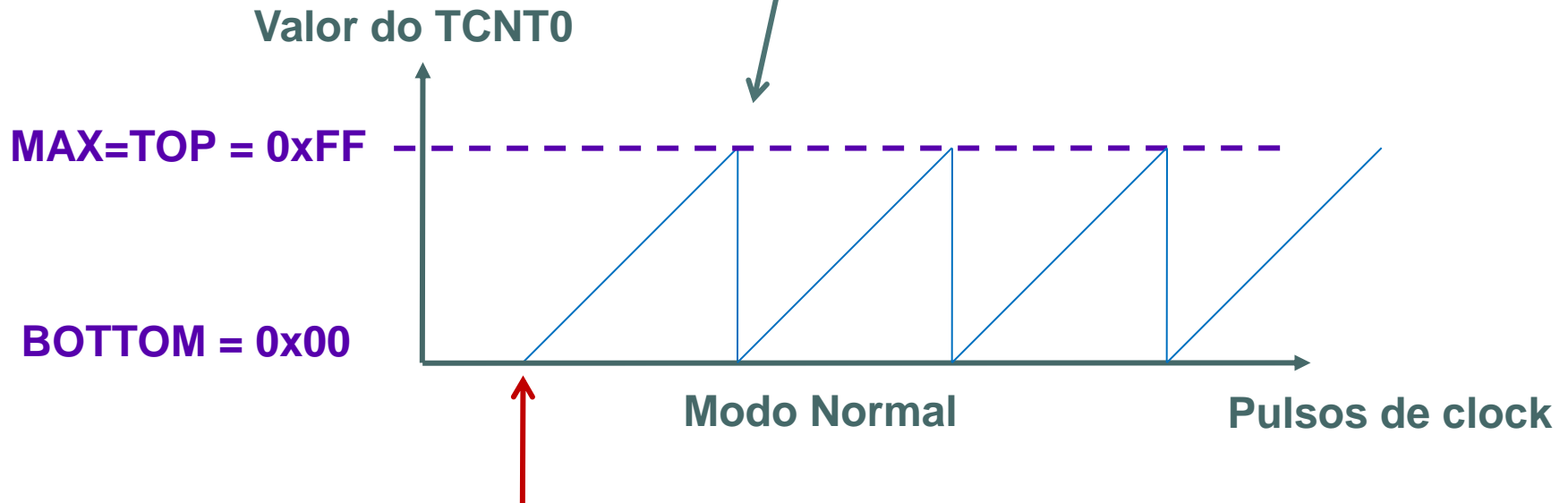


Operação básica do temporizador / contador (8-bits)



Modo Normal (interrupção)

Quando o valor de TCNT0 se iguala ao definido em TOP uma interrupção é gerada. Neste caso TOP=MAX.



O contador é iniciado aqui e aumenta automaticamente a uma taxa precisa

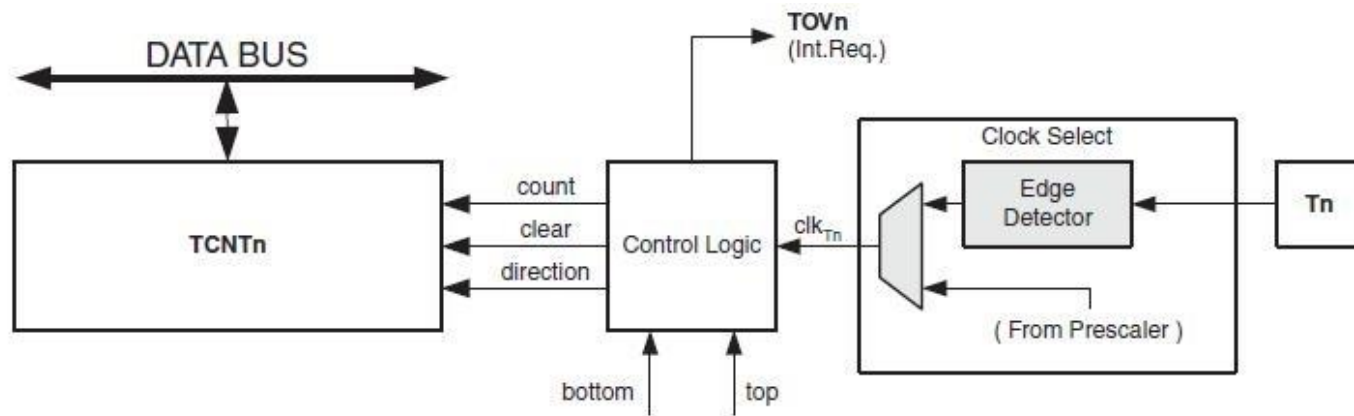
Operação básica do temporizador / contador (8-bits)

MODO DE COMPARAÇÃO (CTC)

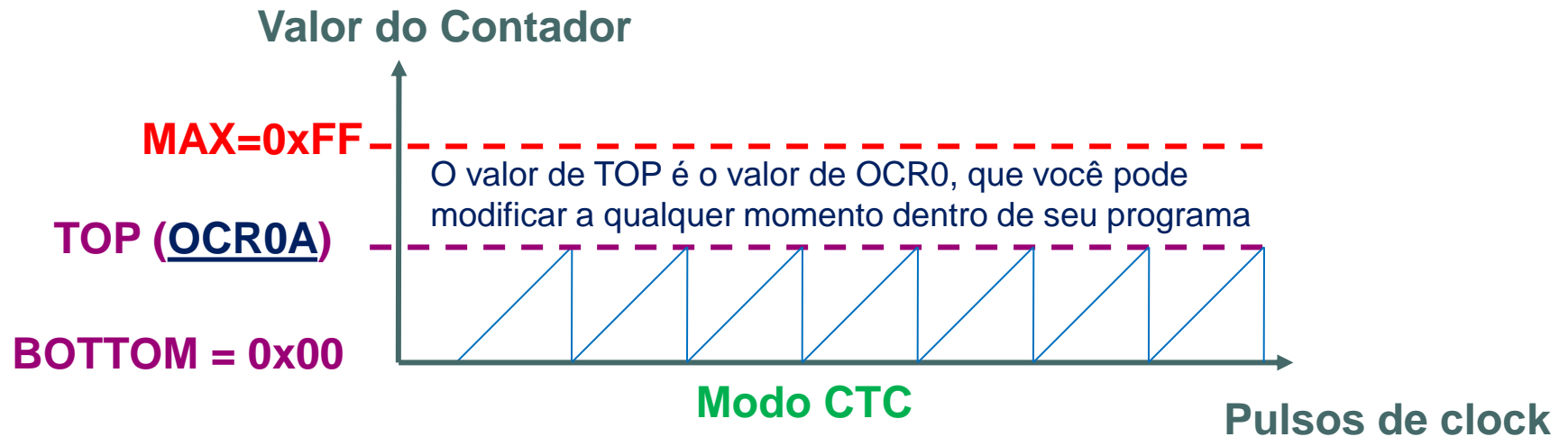
No modo CTC (*Clear Timer on Compare* - limpeza do contador na igualdade de comparação), o registrador OCR0A é usado para manipular a resolução do TC0.

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.

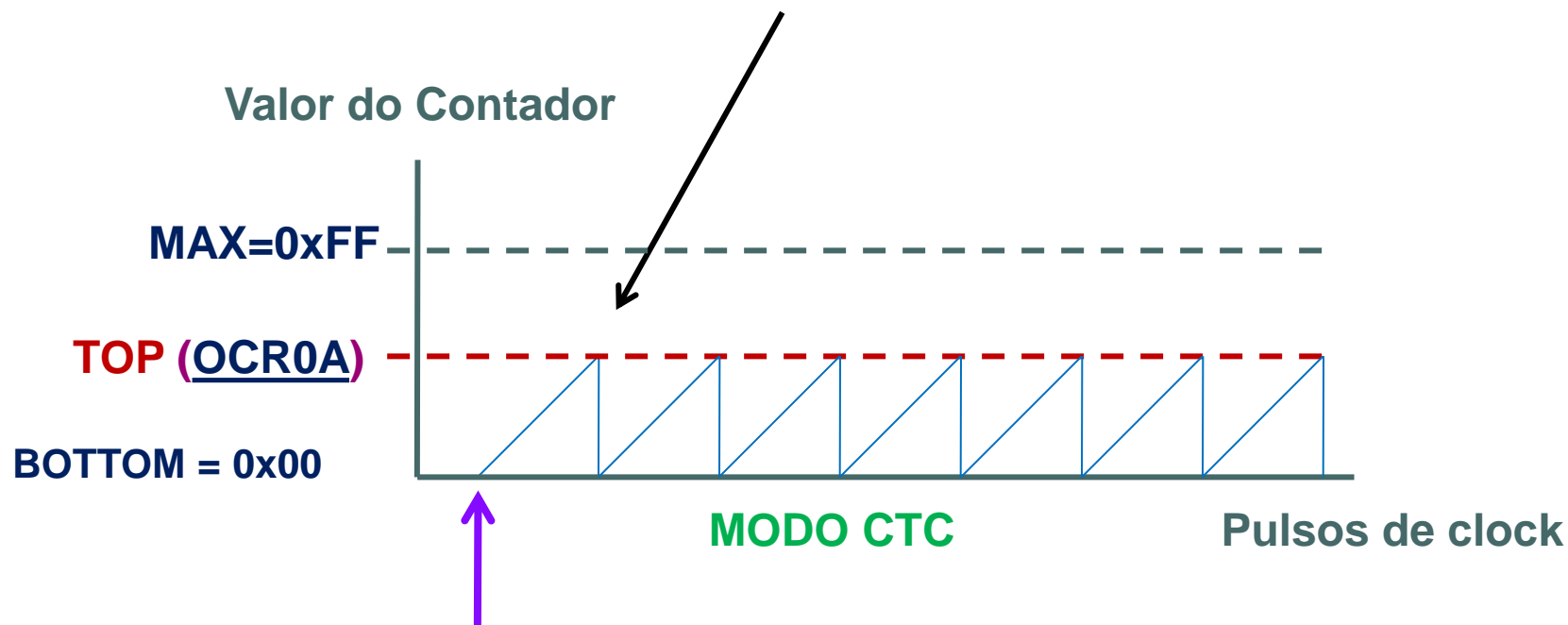


- Quando o *prescaler* recebe um pulso de um ciclo de *clock* e o passa para a Lógica de Controle o registrador **TCNTn** é incrementado de uma unidade.
- O valor de **TCNTn** é então comparado ao de **OCRn**.
 - Quando a comparação é verdadeira, o bit **TOVn** é setado no registro **TIFR**.
 - **TOV0** pode gerar uma interrupção de estouro de temporizador. Para ativar as interrupções do **timer0**, é preciso setar o bit **TOIE0** no registrador **TIMSK**.



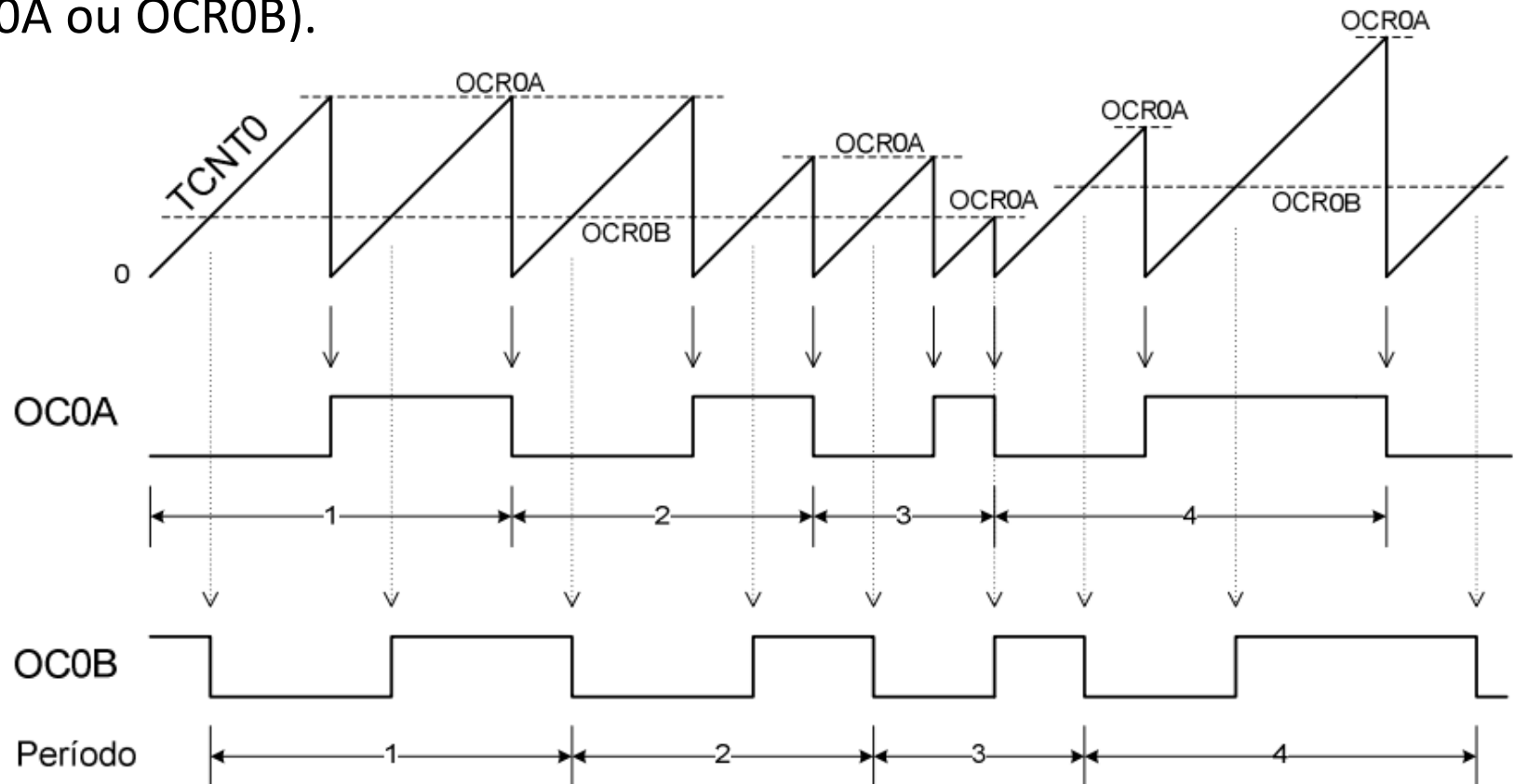
Interrupção do modo CTC

Quando o valor de TCNT0 se iguala ao valor definido em TOP(OCR0A), uma interrupção é gerada.



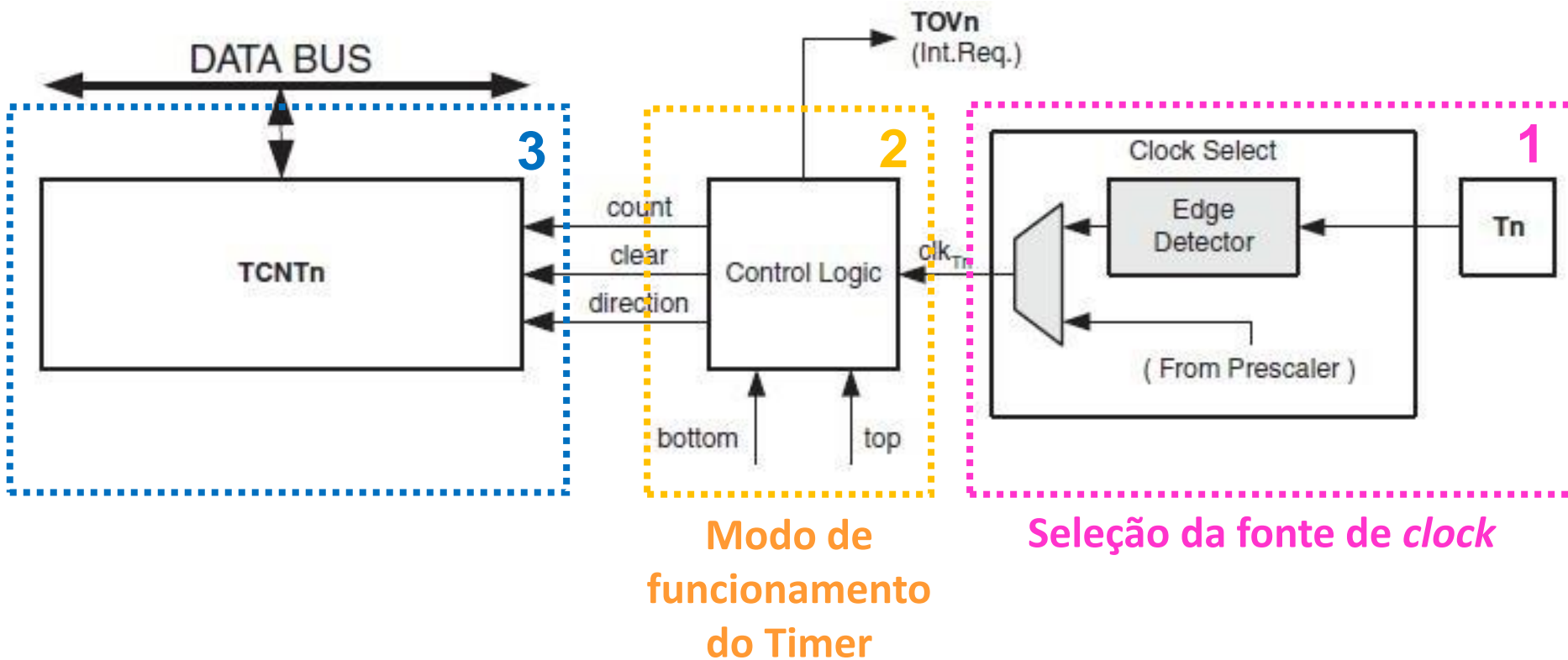
O contador é iniciado aqui e aumenta automaticamente a uma taxa precisa

O modo CTC também permite a geração de ondas quadradas. Basta configurar os pinos OC0A e OC0B para gerar interrupções. Assim uma interrupção é gerada cada vez que o contador atingir o valor de comparação (OCR0A ou OCR0B).



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.

Configuração dos Registradores do TC0



Para a configuração do timer devem ser utilizados até 7 registradores: TCCRnA, TCCRnB, OCRnA, OCRnB, TCNTn, TIMSKn e TIFRn, sendo “n” o número do timer que será utilizado. Inicialmente, será configurado o timer no modo normal e, para isto, serão utilizados apenas 3 registradores. Os demais podem ser consultados no datasheet do componente.


Registadores 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*).

 Modos 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.

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 5:4 – COM0B1:0 – *Compare Match Output B Mode*

Estes bits controlam o comportamento do pino OC0B (*Output Compare 0B*).

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.

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 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

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 tabela anterior.

Bits 2:0 – CS02:0 – Clock Select

Bits para seleção da fonte de *clock* para o TCO, como na tabela a seguir.

Seleção do *clock* para o TC0.

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

TCNT0 – Timer/Counter 0 Register

Registrador de 8 bits no qual 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.

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.

PWM – Pulse Width Modulation

PWM – Pulse Width Modulation

É a técnica que possibilita a geração de uma onda quadrada cuja largura do pulso pode ser controlada.

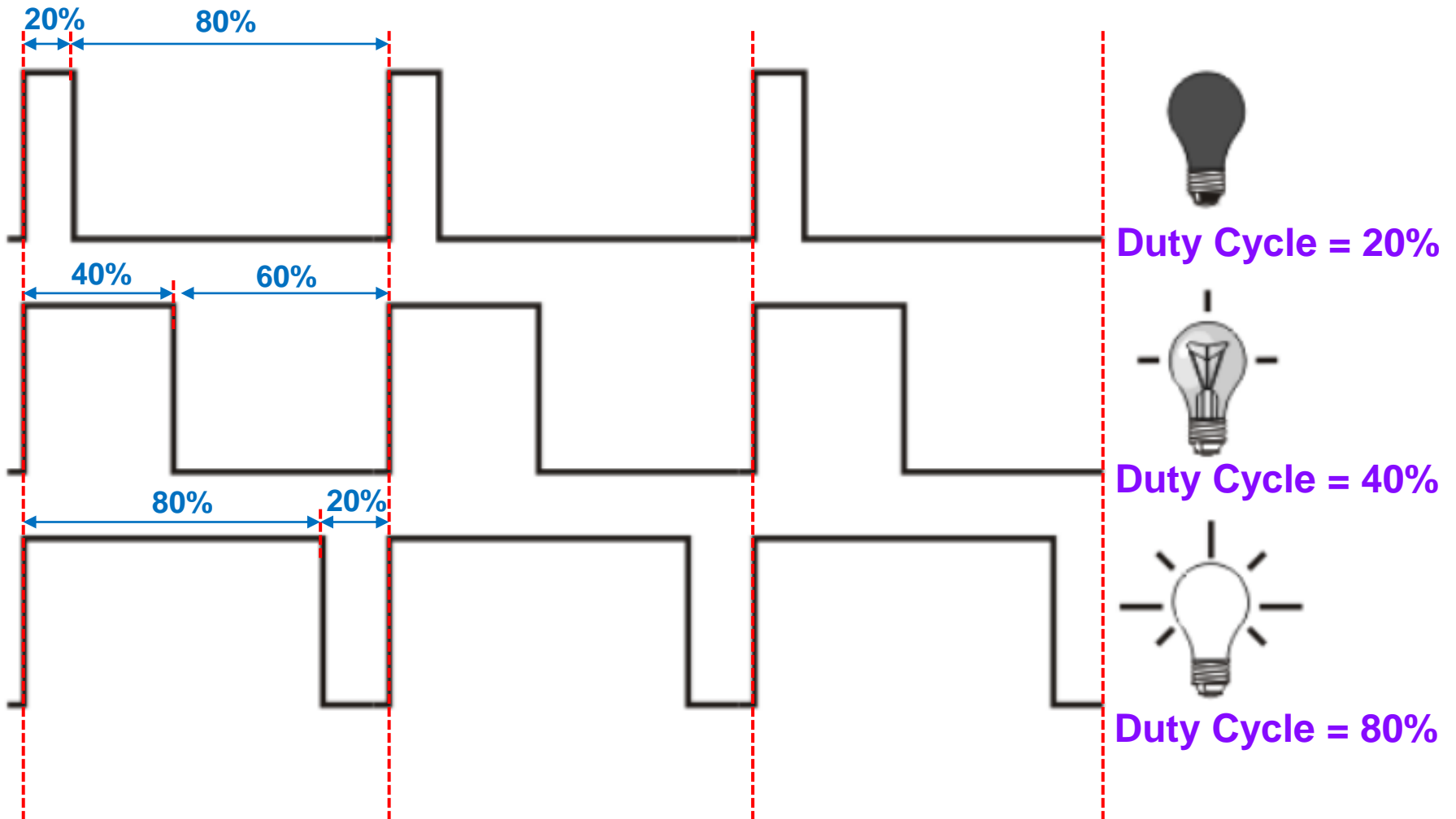
Assim, é possível controlar a **porcentagem do tempo em que a onda permanece em nível lógico alto**.

Esse tempo é chamado de *Duty Cycle* (Ciclo de trabalho) e sua alteração provoca mudança no valor médio da onda, indo desde 0V (0% de *Duty Cycle*) a 5V (100% de *Duty Cycle*), no caso dos microcontroladores.

A alteração do *Duty Cycle* faz com que seja possível a geração de sinais com diferentes valores médios (DC), fazendo com que uma saída PWM opere como uma saída analógica.

PWM – Pulse Width Modulation

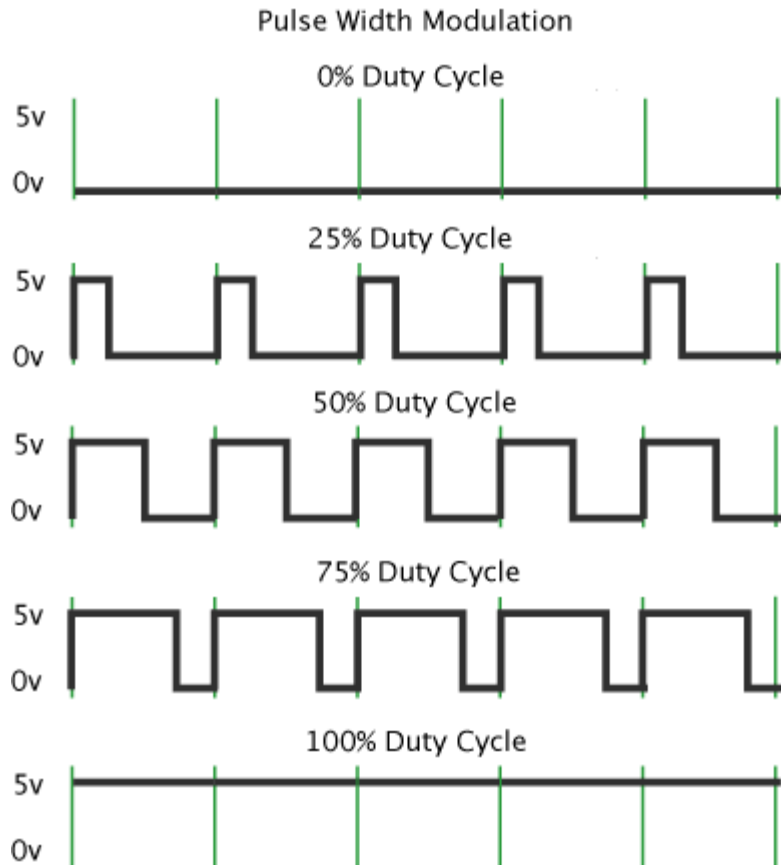
Diferentes configurações de DC (Duty Cycle):



PWM – Pulse Width Modulation

Como a tensão de saída em um pino do microcontrolador varia, normalmente, de 0V a 5V, pode-se calcular o valor média da tensão na saída de um pino PWM:

$$V_{MÉDIA} = V_{MÁX} * \text{Duty Cycle}(\%)$$



Variação da tensão de saída em um pino de acordo com a variação do DC:

→ $V_{MÉDIA} = 5 * 0 = 0V$

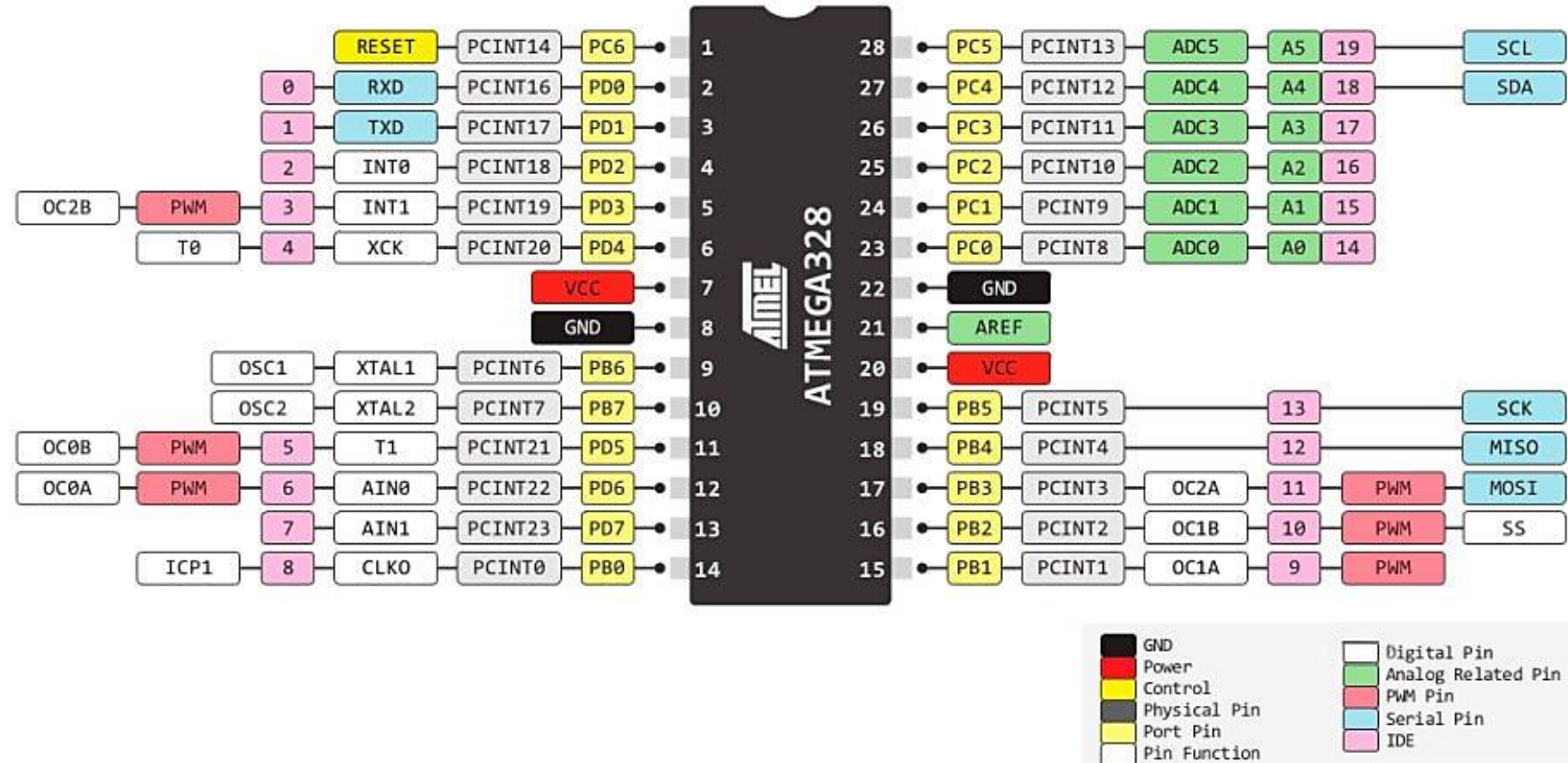
→ $V_{MÉDIA} = 5 * 0,25 = 1,25V$

→ $V_{MÉDIA} = 5 * 0,5 = 2,5V$

→ $V_{MÉDIA} = 5 * 0,75 = 3,75V$

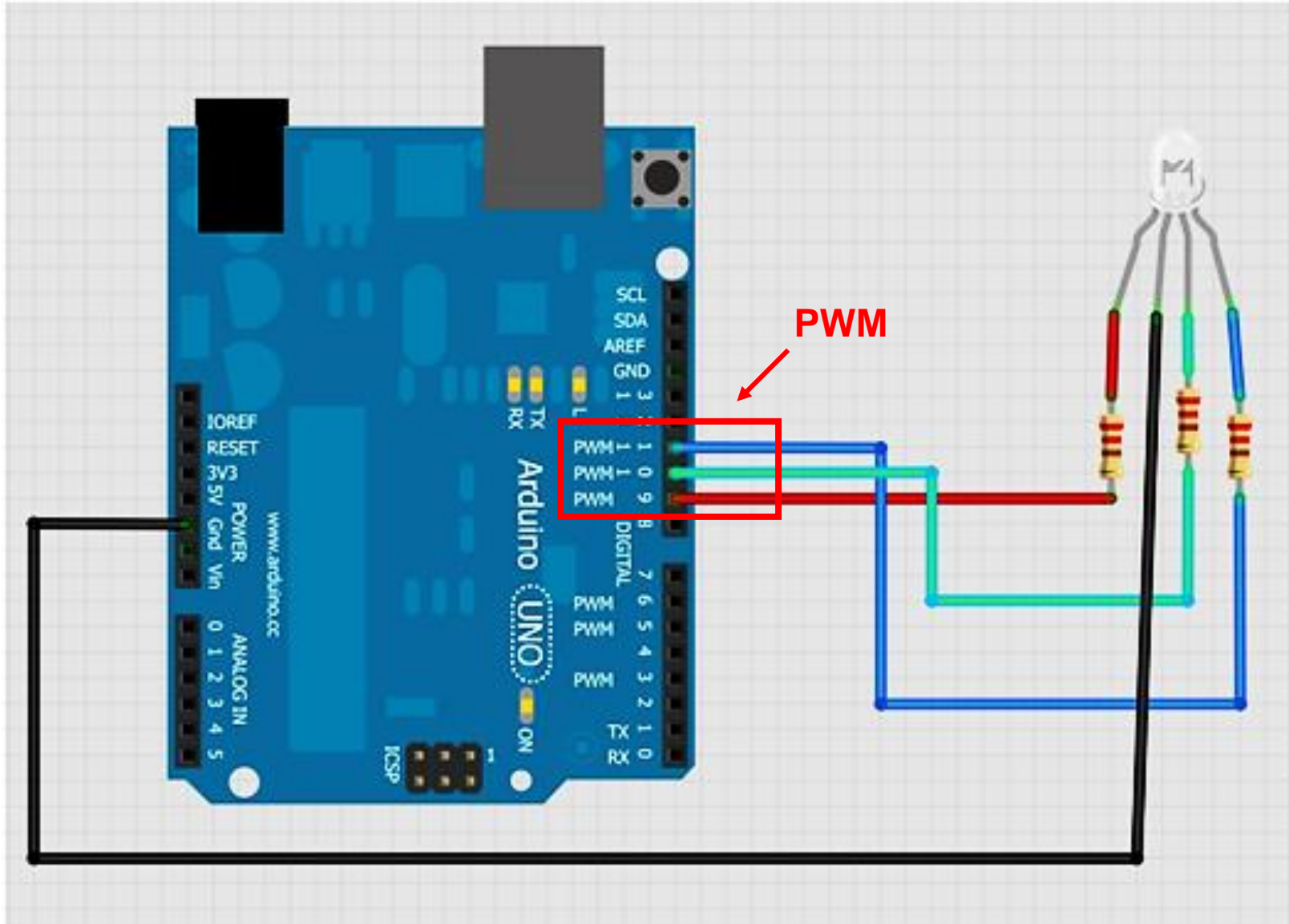
→ $V_{MÉDIA} = 5 * 1 = 5V$

Na figura abaixo é possível identificar os pinos com saída PWM no ATMEGA328



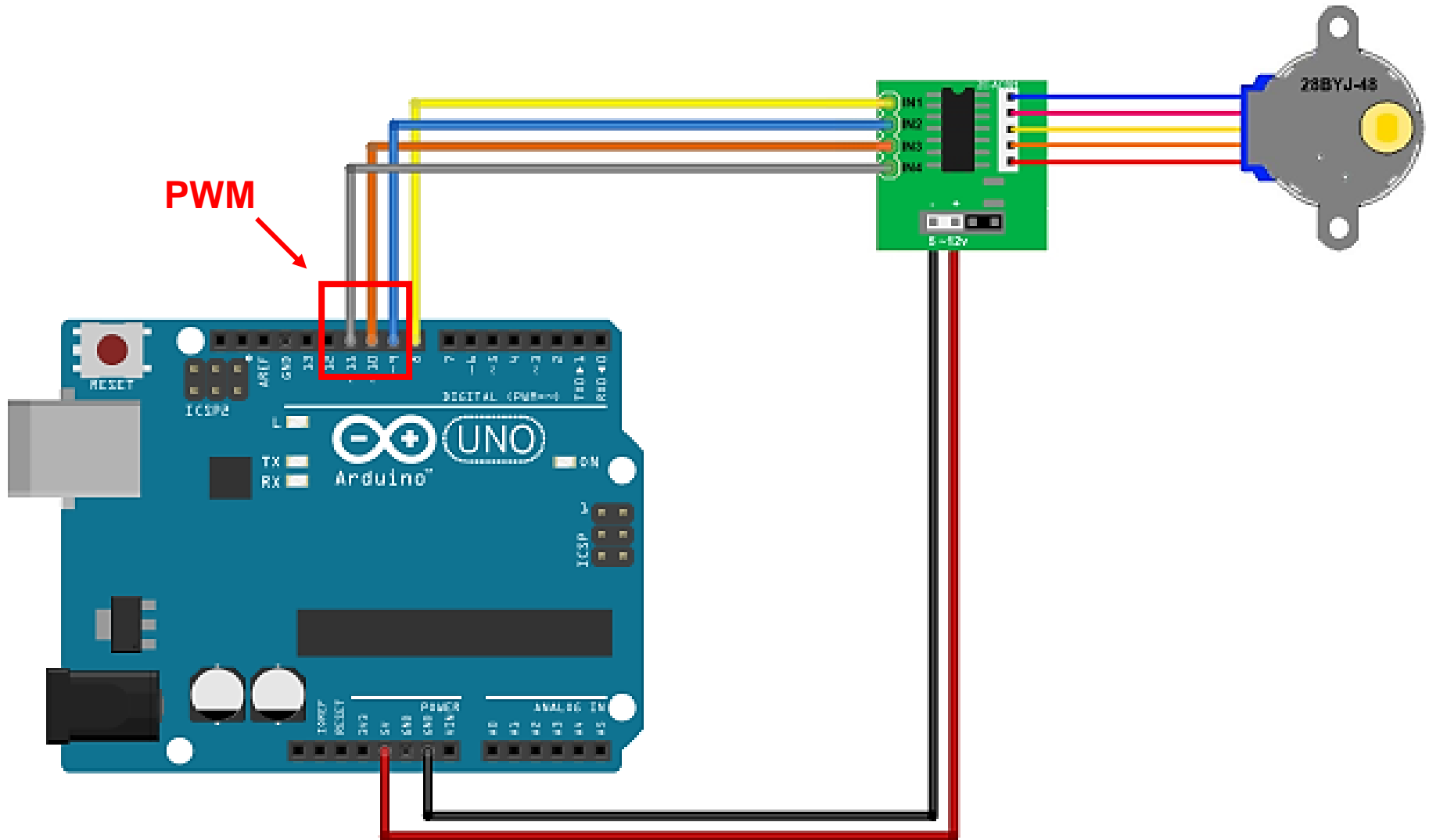
Aplicações do PWM

Controlando a cor de um led RGB:



Aplicações do PWM

Controle de velocidade de motor CC de 5V



A importância do *Prescaler* para o PWM

- Como vimos, o papel do *prescaler* é de retardar as coisas. Isso é bom porque nos permite executar o PWM em frequências diferentes.

- É também importante pois alguns dispositivos são sensíveis às “velocidades” PWM.

Um motor, por exemplo, ficará quente se a forma de onda do PWM for muito “rápida” e ficará instável se o PWM for muito “lento”.

- O ATmega328 tem 6 saídas PWM, duas em cada temporizador/ contador.

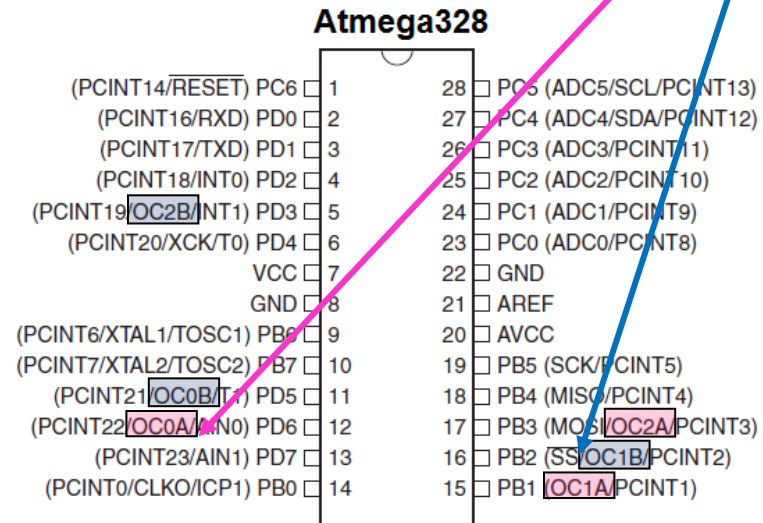
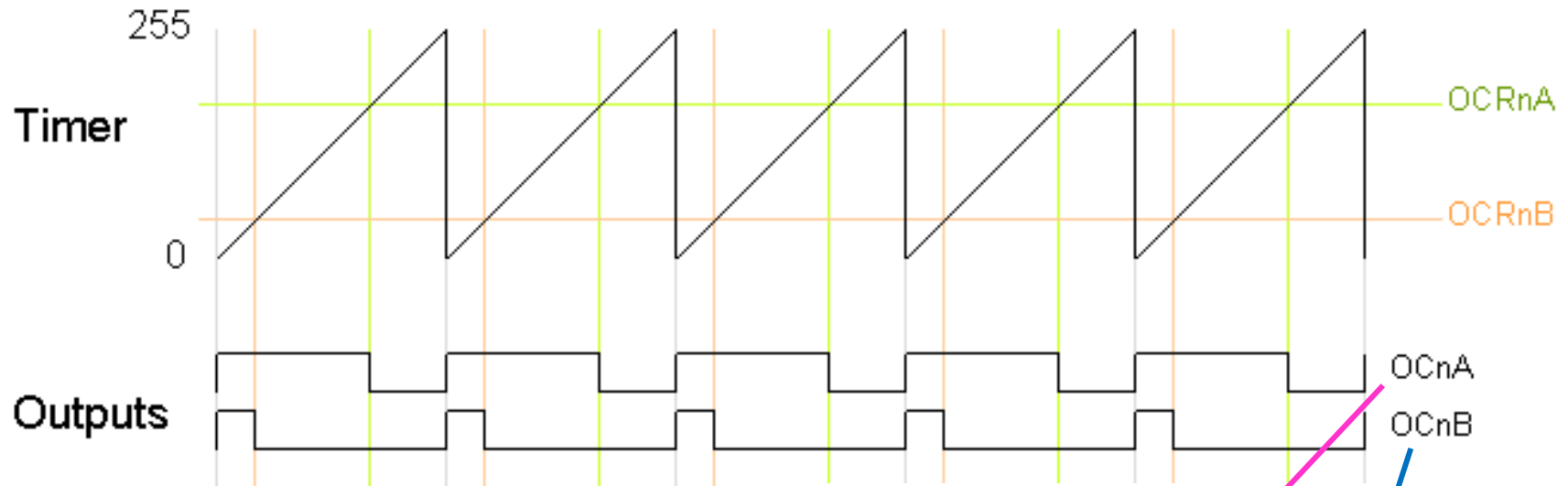
O temporizador PWM do AVR é capaz de funcionar em 3 modos diferentes: **Fast PWM**, **PWM com correção de fase** e **PWM com correção de fase e frequência**.

Modo PWM rápido:

- O PWM rápido funciona da mesma maneira que o contador normal.
- Neste modo (mais simples), o temporizador conta repetidamente de 0 a 255.
- A saída liga quando o temporizador está em 0 e desliga quando o temporizador corresponde ao registrador de comparação de saída (output compare register – OCR).
- Quanto maior for o valor no registro de comparação de saída, maior será o ciclo de trabalho. Este modo é conhecido como **Modo PWM Rápido** .

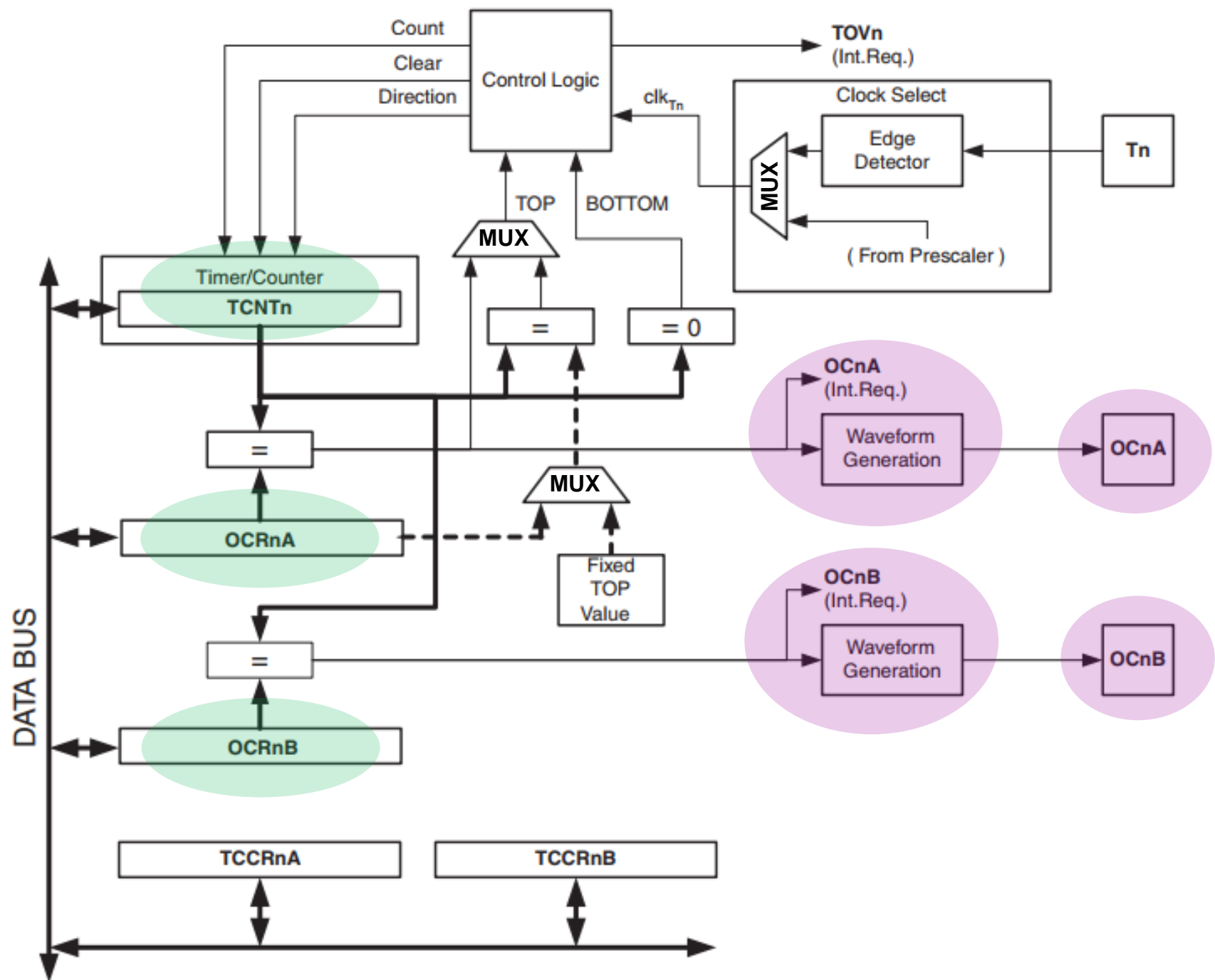
O diagrama abaixo mostra as saídas para dois valores específicos de OCRnA e OCRnB.

Observe que ambas as saídas têm a mesma frequência, combinando com a frequência de um ciclo completo do temporizador.



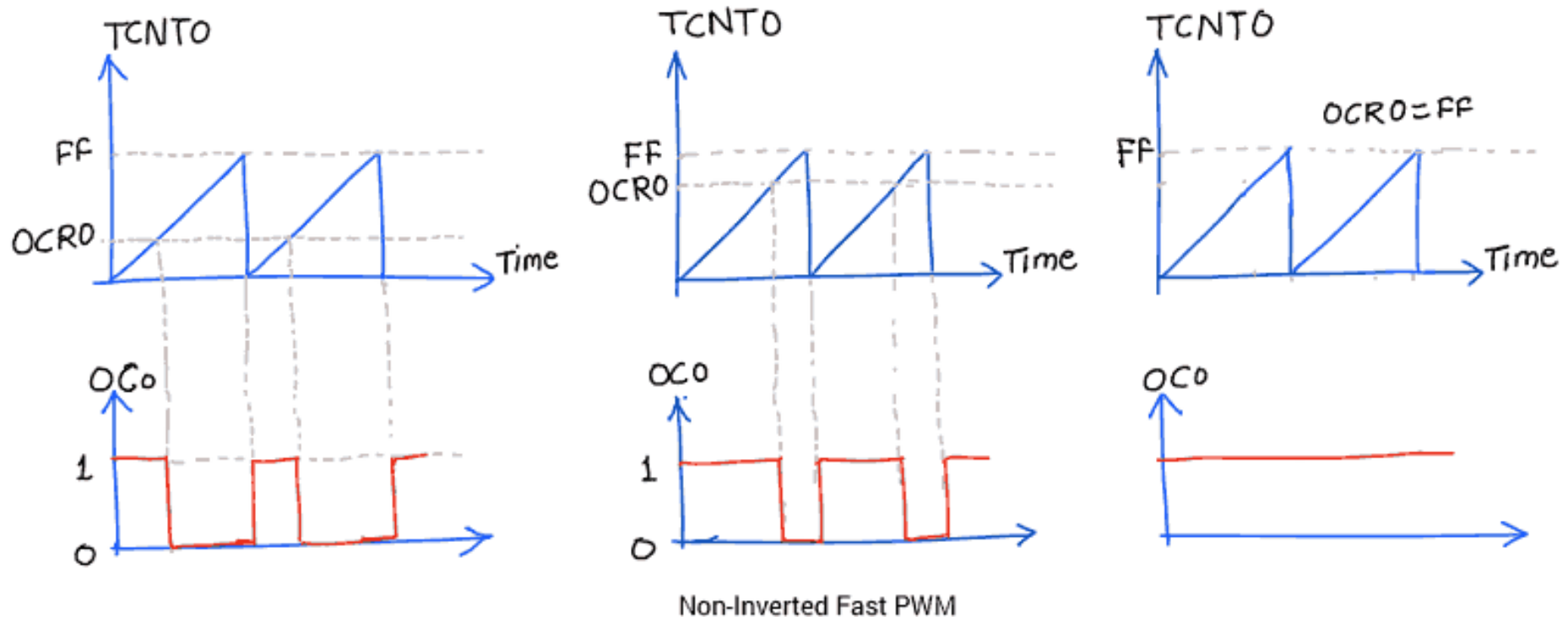
Modo PWM rápido (funcionamento):

- A lógica de controle recebe o sinal e incrementa o registro **TCNTn**.
- Quando uma correspondência é detectada, a flag **OCFnx** é setada e o sinal é enviado para o gerador de forma de onda (*Waveform Generator*).
- O *Waveform Generator* então muda o estado do pino **OCnx** (o estado é determinado pelo modo selecionado).
- Quando o registro TCNTn passa o valor TOP (**0xFF** ou **OCRnA**), ele simplesmente transborda (ou ultrapassa) e volta para 0, ao mesmo tempo que a flag **OCFnx** é setada.



- A flag OCFnx pode ser configurado para acionar uma interrupção.
A mesma pode ser apagado por software, mas como sempre é apagada automaticamente quando uma solicitação de interrupção é acionada.
- Devido à alta frequência deste modo é melhor usado para DAC, controle de brilho de LEDs, retificação e regulação de energia.
- A frequência do PWM rápido pode ser calculada pela seguinte equação.

$$f_{PWM} = f_{CLK} / [\text{Valor_do_Prescaler} * (1 + \text{Valor_de_TOP})]$$



O fragmento de código a seguir configura o PWM rápido nos pinos 3 e 11 (Timer 2).

```
pinMode (3, OUTPUT);  
pinMode (11, OUTPUT);  
TCCR2A = (1<<COM2A0) | (1<<COM2B1) | (1<<WGM20);  
TCCR2B = (1<<WGM22) | (1<<CS22);  
OCR2A = 180;  
OCR2B = 50;
```

- **Frequência de saída A:** $16 \text{ MHz} / 64 / (180 + 1) / 2 = 690,6 \text{ Hz}$
- **Ciclo de trabalho de saída A:** 50%
- **Frequência de saída B:** $6 \text{ MHz} / 64 / (180 + 1) = 1381,2 \text{ Hz}$
- **Ciclo de trabalho de saída B:** $(50 + 1) / (180 + 1) = 28,2\%$

Observe que, neste exemplo, o temporizador vai de 0 a 180, o que leva 181 ciclos de *clock*, então a frequência de saída é dividida por 181.

A “saída A” tem metade da frequência da “saída B” porque o modo *Toggle on Compare Match* alterna a “saída A” à cada ciclo completo de *clock*.

Em resumo são feitas seguintes configurações de registro:

- Definição do **PWM rápido**: bits **WGM** como “0”, “1” e “1”, respectivamente.
- Definição do **PWM não invertido** para as saídas A e B: bits COM2A e COM2B para 1 e 0, respectivamente.
- Definição do **Prescaler**: os bits “CS” para “1”, “0” e “0” (clock/64).
- Definição dos registros de comparação de saída (**Output Compare**)
No exemplo são arbitrariamente definidos como 180 e 50 para controlar o ciclo de trabalho PWM das saídas A e B.
- O modo de OCR2A é definido como "*Toggle on Compare Match*", definindo os bits COM2A como 01.

Passos para se desenvolver um PWM via software:

- 1- Defina a frequência e o período de trabalho desejado, por exemplo 1KHz (1ms);
- 2- Defina a resolução desejada para este PWM, isto é, em quantas partes o ciclo ativo será dividido. Neste exemplo, usaremos 100, assim a resolução ficará entre 0 e 100%;
- 3- Configure um timer para que ele gere uma interrupção respeitando o seguinte cálculo:

$$Período_{estouro} = \frac{Período_{PWM}}{resolução} = \frac{0,001}{100} = 10\mu s$$

- 4- Na interrupção, incremente uma variável de contagem e ative o pino desejado quando ela atingir o valor da resolução e desligue o pino quando a contagem atingir o ciclo ativo desejado.

EXEMPLOS

Exemplo de temporização do LED conectado ao pino PD4

```
int main(void)
{
    DDRD |= (1 << 4);
    TCCR0B|=(1<<CS02) | (1<<CS00); //CONFIG. DO PRESCALER
    TCNT0=0;                        //INICIALIZAÇÃO DO CONTADOR (EM ZERO)
    while(1)
    {
        if (TCNT0 >= 255)          // QUANDO O CONT ATINGIR O MÁXIMO
        {
            PORTD ^= (1 << 4); //INVERTE O LED
            TCNT0 = 0;           //ZERA O CONTADOR
        }
    }
}
```

Exemplo de temporização do LED conectado ao pino PD4, com interrupção

```
ISR(TIMER0_OVF_vect)
{
    PORTD^=(1<<PD7);
}
```

```
int main(void)
{
    DDRD |= 0b11111111;
    PORTD = 0b00000000;
    TCCR0B = 0b00000101; // ou TCCR0B|=(1 << CS02)|(1 << CS00);
    TIMSK0=0b00000001;
    sei();
    while(1) {
        //PORTD^=(1<<4); - Inclusão de um novo led como referência
        //_delay_ms(500);
    }
}
```

```

#include <avr/io.h>
#include <avr/interrupt.h>

ISR (TIMER1_COMPA_vect)
{
    PORTD^=(1<<PD6);
}

int main (void) {
    //inicializar o LED no pino 4 como saída
    DDRD = 0b01000000; // OU DDRD |= (1<<PD6)
    //CONFIGURAÇÃO DO REGISTRADOR DE CONTROLE DO TIMER0 (TCCR0B)
    TCCR1B = 0b00001101;
    /*CONFIGURANDO O CLOCK COM PRESCALER DE 1024 e O MODO DE
    COMPARAÇÃO (CTC) */
    TIMSK1 = 0b00000010; /*O ULTIMO BIT LIGA O SERVIÇO DE INTERRUPÇÃO
    QUANDO O CONTADOR ATINGIR O MÁXIMO O OCR1A */
    OCR1A = 15624;
    sei(); // LIGA O SERVIÇO GLOBAL DE INTERRUPÇÕES
    while (1) {
    }
}

```

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.

```
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)  
  {  
  }  
}
```

Exemplo: Fast PWM sem interrupção

```
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

    //fast PWM, TOP = 0xFF, OC0A e OC0B habilitados
    TCCR0A = 0b10100011; //PWM com saída não invertida - pinos OC0A e
    OC0B
    TCCR0B = 0b00000011; //liga TC0, prescaler = 64
    OCR0A = 200; //controle do ciclo ativo do PWM OC0A
    OCR0B = 50;  //controle do ciclo ativo do PWM OC0B
    while(1) {
    }
}
```


PWM SEM INTERRUPÇÃO (USANDO O **TIMER 0**)

No exemplo a seguir a saída do contador1A é configurada com Duty Cycle de 25% e a saída do contador1B com Duty Cycle de 75% usando ICR1 como TOP (16bits), Fast PWM.

PINO 6 DO ARDUINO

```
#include <avr/io.h>
```

```
int main(void)
```

```
{
```

```
    DDRD |= (1 << 6); // PD6 como saída
```

```
    OCR0A = 128; //config. o PWM com 25% de duty cycle
```

```
    TCCR0A |= (1 << COM0A1); //PWM com saída não invertida
```

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

```
    TCCR0B |= (1 << CS01); //Prescaler=8 e ativação do PWM
```

```
    while (1);
```

```
    {
```

```
    }
```

```
}
```

PWM SEM INTERRUPTÃO (USANDO O **TIMER 1**)

No exemplo a seguir a saída do contador1A é configurada com Duty Cycle de 25% e a saída do contador1B com Duty Cycle de 75% usando ICR1 como TOP (16bits), Fast PWM.

PINO 10 DO ARDUINO

```
int main(void)
{
    DDRB |= (1 << 1)|(1 << 2); // PB1 e PB2 como saídas
    ICR1 = 65535; // configurando o valor TOP para 65535
    OCR1A = 16383; // config. o PWM com 25% de duty cycle
    OCR1B = 49151; // config. o PWM com 75% de duty cycle
    TCCR1A |= (1 << COM1A1)|(1 << COM1B1); // modo de não-inversão
    // Modo Fast PWM com ICR1 como valor de TOP:
    TCCR1A |= (1 << WGM11); TCCR1B |= (1 << WGM12)|(1 << WGM13);
    TCCR1B |= (1 << CS10); // Inicia o timer sem prescaler

    while (1);
    {
    }
}
```

PWM SEM INTERRUPÇÃO (USANDO O **TIMER 2**)

No exemplo a seguir a saída do contador1A é configurada com Duty Cycle de 25% e a saída do contador1B com Duty Cycle de 75% usando ICR1 como TOP (16bits), Fast PWM.

PINO 11 DO ARDUINO

```
#include <avr/io.h>
int main(void)
{
    DDRB |= (1 << 3); // PD3 como saída
    OCR2A = 128; //config. o PWM com 25% de duty cycle
    TCCR2A |= (1 << COM2A1); //PWM com saída não invertida
    TCCR2A |= (1 << WGM21) | (1 << WGM20); // Modo fast PWM
    TCCR2B |= (1 << CS21); //Prescaler=8 e ativação do PWM
    while (1);
    {
    }
}
```

PWM sem interrupção

```
#define F_CPU 8000000UL
```

```
#include "avr/io.h"
```

```
#include <util/delay.h>
```

```
void PWM_init() {
```

```
//PWM com saída não invertida:
```

```
TCCR0 = (1<<WGM00) | (1<<WGM01) | (1<<COM01) | (1<<CS00);
```

```
DDRB |= (1<<PB3); // Pino OC0 como saída
```

```
}
```

```
int main () {
```

```
    unsigned char ciclo_ativo;
```

```
    Inicia_PWM();
```

```
    while (1) {
```

```
        for(ciclo_ativo = 0; ciclo_ativo < 255; duty++) {
```

```
            OCR0 = ciclo_ativo; // aumenta a intensidade do LED
```

```
            _delay_ms(8);
```

```
        }
```

```
        for(ciclo_ativo = 255; ciclo_ativo > 1; ciclo_ativo --) {
```

```
            OCR0 = ciclo_ativo; // diminui a intensidade do LED
```

```
            _delay_ms(8);
```

```
        }
```

```
    }
```

```
}
```

```
void inicia_PWM(void){
```

```
TCCR0A=(1<<WGM00)|(1<<WGM01)|(1<<COM00)|(1<<COM0B0);
```

```
TCCR0B=(1<<CS01);
```

```
DDRD =(1<<PD6);
```

```
}
```

```
int main(void)
```

```
{ unsigned char ciclo_ativo;
```

```
  inicia_PWM();
```

```
    while (1){
```

```
        for
```

```
(ciclo_ativo=0;ciclo_ativo<255;ciclo_ativo++){
```

```
            OCR0A=ciclo_ativo; _delay_ms(100);
```

```
        }
```

```
        for
```

```
(ciclo_ativo=255;ciclo_ativo>1;ciclo_ativo--){
```

```
            OCR0A=ciclo_ativo; _delay_ms(100);
```

```
        }
```

```
    }
```

```
}
```

Fast PWM com interrupção

```
#include <avr/io.h>
#include <avr/interrupt.h>
volatile unsigned int Passo_PWM1 = 0;
volatile unsigned int Ciclo_Ativo_PWM1;

ISR(TIMER0_OVF_vect)
{
    TCNT0 = 96; //Recarrega o timer para garantir a próxima interrupção com 10us
    Passo_PWM1++; //incremento do passo de tempo
    if (Passo_PWM1 == 100) // 100 É A RESOLUÇÃO DO PWM
    {
        Passo_PWM1 = 0; //inicializa o contador
        PORTD|= (1<<PD1); //na igualdade de comparação coloca o pino PWM1 em 1
    }
    else if (Passo_PWM1 == Ciclo_Ativo_PWM1)
        PORTD&=~(1<<PD1); //Apaga o led quando o contador chega ao valor do ciclo ativo do PWM1
}

int main() {
    DDRD = 0b11111111; //PORTD como saída
    PORTD = 0b00000001; //acende apenas o LED0
    TCCR0B = 0b00000001; //TC0 com prescaler de 1
    TIMSK0 = 0b00000001; //habilita a interrupção do TC0
    TCNT0 = 96; //Inicia a contagem em 96 para gerar a interrupção a cada 10us 1/[freq(PWM)*resolução]
    sei(); //habilita as interrupções
    Ciclo_Ativo_PWM1 = 50; //Determina o ciclo ativo para o PWM1 (0 - 100)
    while (1) {
    }
}
```