## **Timers do ATmega328 no Arduino** Temporização é fundamental quando se trabalha com circuitos microcontrolados, seja para um simples delay ou para geração de sinais ou eventos periódicos. Desde o mais simples microcontrolador sempre está disponível pelo menos um periférico temporizador/contator. Esse periférico possui hardware dedicado para contagem de tempo e o seu correto uso, auxilia em uma programação eficiente para realização de diversos projetos.

O Atmega328, utilizado na placa Arduino UNO, possui 3 timers, sendo dois de 8 bits (TIMER0 e TIMER2) e um de 16 bits (TIMER1).A diferença mais importante entre o temporizador de 8 bits e 16 bits é a resolução do temporizador. 8 bits significa 256 valores, onde 16 bits significa 65536 valores para resolução mais alta.  
 Esses temporizadores são importantes para diversas funcionalidades, tais como:

* Temporização;
* Contagem de eventos externos;
* Geração de sinais PWM;
* Interrupções periódicas;
* Medida de intervalos de pulsos.

Cada temporizador possui características próprias e são utilizados conforme os recursos disponíveis. A biblioteca do Arduino abstrai o uso destes temporizadores em muitas de suas funções. Por exemplos, as funções *delay()*, *millis()*, *micros()*, *tone()*, *analogWrite()* utilizam recursos de timers para o funcionamento.  
O TIMER1 é utilizado somente em algumas bibliotecas no Arduino específicas, podendo ser utilizado para outras finalidades sem causar muito impacto no funcionamento do restante das funções. A seguir serão apresentadas suas características.  
  
Documentação - descrições simplificadas do que as rotinas mais importantes fazem

BIBLIOTECA <avr/TimerOne.h>

**initialize(period)**Você deve chamar esse método primeiro para usar qualquer um dos outros métodos. Você pode opcionalmente especificar o período do temporizador aqui (em microssegundos), por padrão, ele é definido como 1 segundo. Observe que isso quebra analogWrite () para pinos digitais 9 e 10 no Arduino.  
**setPeriod (período)**Define o período em microssegundos. O período mínimo ou a frequência mais alta que esta biblioteca suporta é 1 microssegundo ou 1 MHz. O período máximo é de 8388480 microssegundos ou cerca de 8,3 segundos. Observe que definir o período mudará a interrupção associada e as frequências e ciclos de trabalho de ambas as saídas pwm simultaneamente.  
**pwm (pino, dever, período)**Gera uma forma de onda PWM no pino especificado. Os pinos de saída para Timer1 são PORTB pinos 1 e 2, então você tem que escolher entre estes dois, qualquer outra coisa será ignorada. No Arduino, esses são os pinos digitais 9 e 10, portanto, esses aliases também funcionam. Os pinos de saída para Timer3 são da PORTE e correspondem a 2,3 e 5 no Arduino Mega. O ciclo de serviço é especificado como um valor de 10 bits, então qualquer coisa entre 0 e 1023. Observe que você pode definir opcionalmente o período com esta função se incluir um valor em microssegundos como o último parâmetro ao chamá-lo. **attachInterrupt (função, período)**Chama uma função no intervalo especificado em microssegundos. Tenha cuidado ao tentar executar uma interrupção muito complicada em uma frequência muito alta, ou a CPU pode nunca entrar no loop principal e seu programa irá 'travar'. Observe que você pode definir o período opcionalmente com esta função se incluir um valor em microssegundos como o último parâmetro ao chamá-lo.  
**setPwmDuty (pin, duty)**Um atalho rápido para definir o dever pwm para um determinado pino se você já o configurou chamando pwm () anteriormente. Isso evita a sobrecarga de habilitar o modo pwm para o pino, definir o registro de direção de dados, verificar se há ajustes de período opcionais etc. que são obrigatórios quando você chama pwm ().

**detachInterrupt ()**Desativa a interrupção anexada.  
**disablePwm (pin)**Desativa o PWM para o pino especificado para que você possa usar esse pino para outra coisa.  
**read()**Lê o tempo desde o último rollover em microssegundos.

**Exemplos**  
**Modo Normal com interrupção por overflow**

O exemplo a seguir exibe como piscar o LED no pino 13, da placa Arduino UNO, em intervalos de 1 segundo e utilizando interrupção por estouro timer:  
#define ledPin 13  
void setup()  
{ pinMode(ledPin, OUTPUT); // Configuração do timer1   
 TCCR1A = 0; //confira timer para operação normal pinos OC1A e OC1B desconectados  
 TCCR1B = 0; //limpa registrador  
 TCCR1B |= (1<<CS10)|(1 << CS12); // configura prescaler para 1024: CS12 = 1 e CS10 = 1  
 TCNT1 = 0xC2F7;// inicia timer com valor para que estouro ocorra em 1 seg // 65536-(16MHz/1024/1Hz) = 49911 = 0xC2F7  
 TIMSK1 |= (1 << TOIE1); // habilita a interrupção do TIMER1}  
void loop()  
{ //loop principal. a manipulação do led é feita na ISR}  
ISR(TIMER1\_OVF\_vect) //interrupção do TIMER1   
{ TCNT1 = 0xC2F7; // Renicia TIMER  
 digitalWrite(ledPin, digitalRead(ledPin) ^ 1); //inverte estado do led}  
  
Nesse exemplo o timer foi configurado para modo normal, com pinos OC1A e OC1B desconectados (TCCR1A = 0). Foi selecionado o prescaler para 1024 através do registrador TCCR1B. Para que o timer estoure a cada segundo é necessário iniciar seu valor com a diferença entre o seu valor máximo (65536) e o período desejado. O período é calculado levando em consideração a frequência do oscilador e o prescaler selecionado, além da frequência de interrupção desejada. Por fim foi habilitada a interrupção de estouro do TIMER1 através do bit T0IE1 do registrador TIMSK1. A inversão do LED é feita na rotina de interrupção, note que é necessário recarregar o timer para a correta contagem  
  
**MODO CTC interrupção por comparação**O exemplo a seguir exibe como piscar o LED, utilizando o modo CTC e gerando interrupção por comparação:

#define ledPin 13  
void setup()  
{ pinMode(ledPin, OUTPUT);   
 // Configuração do TIMER1  
 TCCR1A = 0;//confira timer para operação normal  
 TCCR1B = 0; //limpa registrador  
 TCNT1 = 0; //zera temporizado  
 OCR1A = 0x3D09; // carrega registrador de comparação: 16MHz/1024/1Hz = 15625 = 0X3D09  
 TCCR1B |= (1 << WGM12)|(1<<CS10)|(1 << CS12); // modo CTC, prescaler de 1024: CS12 = 1 e CS10 = 1   
 TIMSK1 |= (1 << OCIE1A); // habilita interrupção por igualdade de comparação}

void loop()  
{ //loop principal. a manipulação do led é feita na ISR}  
ISR(TIMER1\_COMPA\_vect)// interrupção por igualdade de comparação no TIMER1  
{ digitalWrite(ledPin, digitalRead(ledPin) ^ 1); //inverte estado do LED  
}  
  
Nesse exemplo é utilizado o modo CTC, dessa forma o valor de contagem do timer é constantemente comparado com o registrador OCR1A. Para o funcionamento foi selecionado o modo CTC, bit WGM12 = 1 e configurado o prescaler para 1024. O valor de comparação foi carregado no registrador OCR1A e por último foi habilitada a interrupção de comparação. A inversão do LED é feita na rotina de interrupção, e note que não é necessário reiniciar o timer com um valor, pois o modo CTC zera o timer quando atingido valor de comparação.  
  
  
Configura a saída PWM no pino 9 com um ciclo de trabalho de 50% e conecta uma interrupção que alterna o pino digital 10 a cada meio segundo.  
#include "TimerOne.h"

1. void setup()
2. { pinMode(10, OUTPUT);
3. Timer1.initialize(500000);// initialize timer1, and set a 1/2 second period
4. Timer1.pwm(9, 512); // setup pwm on pin 9, 50% duty cycle
5. Timer1.attachInterrupt(callback);// attaches callback() as a timer overflow interrupt}
6. void callback()
7. { digitalWrite(10, digitalRead(10) ^ 1);}

Watchdog Timer:

O Watchdog Timer (WDT) é um timer que funciona independente do microcontrolador e pode ser usado para causar uma interrupção ou um reset.

Ele geralmente é utilizado para, por exemplo, resetar o microcontrolador caso haja alguma falha e ele esteja travado, executar uma tarefa em um intervalo determinado de tempo, trazer o microcontrolador de volta de um modo de hibernação, etc.

Watchdog Timer - Data sheet págs. 43 a 48

**Link Vídeo Explicação:** [Arduino Watch Dog Timer / AVR](https://www.youtube.com/watch?v=8sZheJVl-R8)

**Link Biblioteca WDT:** <https://www.mysensors.org/apidocs/group__avr__watchdog.html>

**Programando o cronômetro de watchdog do AVR**

Existem duas maneiras de configurar o temporizador de watchdog: programando o fusível WDTON ao piscar o MCU e outra durante o fluxo do programa. Se o fusível WDTON estiver programado, ele não pode ser desabilitado no software e só funciona no modo de reinicialização do sistema (sem interrupção disponível). Isso pode ser usado para garantir a segurança do sistema. A única coisa que podemos mudar aqui é uma pré-chamada.

A programação de watchdog de software oferece mais flexibilidade e funcionalidade. Como mencionamos, você pode usar os modos de reinicialização e interrupção. Para iniciar o cronômetro de watchdog ou alterar seu prescaler, alguma sequência de programação é necessária. Isso ajuda a garantir o fluxo seguro do programa. Esses são:

Desabilite interrupções;

Reinicialize o cronômetro de watchdog com o comando WDR;

Grave a lógica um nos bits WDCE e WDE de WDTCSR simultaneamente;

Então, dentro de quatro ciclos, escreva a lógica um para habilitar (ou zero para desabilitar) o cronômetro de vigilância ou alterar o pré-chamada.

Existem alguns exemplos de código na folha de dados do AVR como você pode fazer isso diretamente. Mas há uma alternativa melhor no AVRGCC avrlibc. Existe uma biblioteca especial wdt.h preparada para manipular o cronômetro do watchdog. Nesse caso, podemos chamar qualquer uma das três funções que cuidarão da sequência de execução adequada:

1 #define wdt\_reset (); // redefine o tempo de watchdog

2 #define wdt\_disable (); // desativa o cronômetro de watchdog

3 #define wdt\_enable (tempo limite); // define um prescaller watchdog

Dependendo do microcontrolador selecionado, os prescalers podem ser definidos para um dos seguintes valores:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | #define WDTO\_15MS 0  #define WDTO\_30MS 1  #define WDTO\_60MS 2  #define WDTO\_120MS 3  #define WDTO\_250MS 4  #define WDTO\_500MS 5  #define WDTO\_1S 6  #define WDTO\_2S 7  #define WDTO\_4S 8  #define WDTO\_8S 9 |

Vamos escrever um programa simples que usará o watchdog timer nos modos de interrupção e reset simultaneamente. Para fazer isso, simplesmente executamos o programa sem redefinir o cronômetro de watchdog. Na rotina principal, piscaremos o LED com aproximadamente 0,5 Hz e quando ocorrer a interrupção do watchdog, ele emitirá um burst de cinco sinais de 0,1 Hz. Após a reinicialização, o LED acenderá por meio segundo. Isso nos dará uma pista dos processos em andamento. Lembre-se de que, após a interrupção do watchdog, o contador deve contar mais uma vez para que ocorra um reset..

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | #include <avr/io.h>  #include <avr/interrupt.h>  #include <avr/wdt.h>  #include <util/delay.h>  // Prepare pino de LED  void PortInit (void)  {PORTD | = (1 << PD2); // LED inicialmente LIGADO  DDRD | = (1 << PD2); // PD2 como saída  }  void WDT\_Init (void) //inicializar watchdog  {cli (); // desativa interrupções  wdt\_reset (); // redefinir watchdog  WDTCSR = (1 << WDCE) | (1 << WDE); // configurar interrupção WDT  WDTCSR = (1 << WDIE) | (1 << WDE) | (1 << WDP3); // Inicia o cronômetro de watchdog com o prescaller 4s  sei (); // Habilita interrupções globais  }  // Tempo limite do Watchdog ISR  ISR (WDT\_vect)  {para (uint8\_t i = 0; i <4; i ++) // Estouro de pulsos de 0,1 Hz  {PORTD | = (1 << PD2); // LED ON  \_delay\_ms (20); // ~ 0,1s de atraso  PORTD & = ~ (1 << PD2); // LED apagado  \_delay\_ms (80);  }  }  int main (vazio)  {PortInit (); // Inicializar porta  WDT\_Init (); // inicializar watchdog  \_delay\_ms (500); // demora para detectar o reset  enquanto (1)  {PORTD | = (1 << PD2); // LED ON  \_delay\_ms (20);  PORTD & = ~ (1 << PD2); // LED apagado  \_delay\_ms (500); // ~ 0,5s de atraso  }  } |

Interrupções

O que é uma interrupção?

O programa em execução em um controlador é normalmente executado em sequência, instrução por instrução. Uma interrupção é um evento externo que interrompe o programa em execução e executa uma rotina especial de serviço de interrupção (ISR). Após o ISR ter sido concluído, o programa em execução continua com a próxima instrução. Instrução significa uma única instrução de máquina, não uma linha de código C ou C ++.

Antes que uma interrupção pendente possa chamar um ISR, as seguintes condições devem ser verdadeiras:

* As interrupções devem ser geralmente habilitadas
* a máscara de interrupção correspondente deve ser habilitada

As interrupções geralmente podem ser ativadas / desativadas com as [interrupções de](http://arduino.cc/en/Reference/Interrupts) função [() 270](http://arduino.cc/en/Reference/Interrupts)/ [noInterrupts () 69](http://arduino.cc/en/Reference/NoInterrupts). Por padrão, as interrupções de firmware do Arduino estão habilitadas. Máscaras de interrupção são habilitadas / desabilitadas definindo / apagando bits no registro de máscara de interrupção (TIMSKx).

Quando ocorre uma interrupção, um sinalizador no registrador de interrupção (TIFRx) é definido. Esta interrupção será apagada automaticamente ao entrar no ISR ou apagando manualmente o bit no registro de flag de interrupção.

As funções do Arduino [attachInterrupt () 328](http://arduino.cc/en/Reference/AttachInterrupt)e [detachInterrupt () 57](http://arduino.cc/en/Reference/DetachInterrupt)só pode ser usado para pinos de interrupção externa. Essas são fontes de interrupção diferentes, não discutidas aqui.

Interrupções do cronômetro

Um cronômetro pode gerar diferentes tipos de interrupções. As definições de registro e bit podem ser encontradas na folha de dados do processador ( [Atmega328](http://www.atmel.com/dyn/resources/prod_documents/doc8271.pdf) ou [Atmega2560](http://www.atmel.com/dyn/resources/prod_documents/doc2549.pdf)) e no arquivo de cabeçalho de definição de E / S (iomx8.h para Arduino, iomxx0\_1.h para Arduino Mega na pasta hardware / tools / avr / include / avr). O sufixo x representa o número do temporizador (0..5), o sufixo y representa o número de saída (A, B, C), por exemplo TIMSK1 (registro de máscara de interrupção do temporizador1) ou OCR2A (registro de comparação de saída do temporizador2 A).

Estouro do cronômetro: O

estouro do cronômetro significa que o cronômetro atingiu seu valor limite. Quando ocorre uma interrupção de estouro de temporizador, o bit de estouro de temporizador TOVx será definido no registrador de sinalizador de interrupção TIFRx. Quando o bit de habilitação de interrupção de estouro de temporizador TOIEx no registro de máscara de interrupção TIMSKx é definido, a rotina de serviço de interrupção de estouro de temporizador ISR (TIMERx\_OVF\_vect) será chamada.

Resultado de comparação de correspondência:

Quando ocorre uma interrupção de correspondência de comparação de saída, o sinalizador OCFxy será definido no registrador de sinalizador de interrupção TIFRx. Quando o bit de habilitação de interrupção de comparação de saída OCIExy no registro de máscara de interrupção TIMSKx é definido, a rotina ISR (TIMERx\_COMPy\_vect) do serviço de interrupção de comparação de saída de saída será chamada.

Captura de entrada de temporizador:

Quando ocorre uma interrupção de captura de entrada de temporizador, o bit de sinalizador de captura de entrada ICFx será definido no registrador de sinalizador de interrupção TIFRx. Quando o bit de habilitação de interrupção de captura de entrada ICIEx no registro de máscara de interrupção TIMSKx é definido, a rotina de serviço de interrupção de captura de entrada do temporizador ISR (TIMERx\_CAPT\_vect) será chamada.

PWM e temporizador

Existe uma relação fixa entre os temporizadores e as saídas compatíveis com PWM. Quando você olha a folha de dados ou a pinagem do processador, esses pinos com capacidade PWM têm nomes como OCRxA, OCRxB ou OCRxC (onde x significa o número do temporizador 0..5). A funcionalidade PWM é freqüentemente compartilhada com outra funcionalidade de pino.

O Arduino possui 3 Timers e 6 pinos de saída PWM. A relação entre os timers e as saídas PWM é:

Pinos 5 e 6: controlados pelo timer0

Pinos 9 e 10: controlados pelo timer1

Pinos 11 e 3: controlados pelo timer2

No Arduino Mega temos 6 timers e 15 saídas PWM:

Pinos 4 e 13 : controlado por timer0

Pinos 11 e 12: controlados por timer1

Pinos 9 e 10: controlados por timer2

Pinos 2, 3 e 5: controlados por timer 3

Pino 6, 7 e 8: controlado pelo temporizador 4

Pino 46, 45 e 44 :: controlado pelo temporizador 5