Capítulo 11

Interrupções e timers

As interrupções são poderosas ferramentas que permitem que o microcontrolador realize outras tarefas enquanto espera por um evento específico. Processadores e microcontroladores atuais incluem mecanismos para o tratamento dessas situações especiais. Em uma interrupção, o fluxo normal de instruções é interrompido para que a causa da interrupção seja tratada.

O oscilador do microcontrolador utiliza um cristal de quartzo para a geração do sinal de clock. Embora esta não seja a solução mais simples, existem muitas razões para usá-la. A frequência de osciladores com cristais de quartzo é precisamente definida e muito estável, de modo que gera pulsos sempre da mesma largura, o que os torna ideais para a medição do tempo. Se for necessário medir o tempo entre dois eventos, basta contar a quantidade de pulsos gerados pelo oscilador no intervalo desejado. Isto é o que os timers internos do microcontrolador fazem.

11.1 Introdução

Como o próprio nome sugere, uma interrupção serve para interromper imediatamente o fluxo regular de um programa, tomando atitudes instantâneas quando um determinado evento acontece. Os eventos capazes de gerar uma interrupção são sinalizados para o processador através de pedidos de interrupção (*IRQs – Interrupt Request*). O processamento da interrupção compõe uma troca de contexto para uma rotina de software especificamente escrita para tratar a interrupção. Essa rotina é chamada rotina de serviço de interrupção (*ISR – Interrupt Service Routine*), ou rotina de tratamento de interrupção (*interrupt handler*). Os endereços dessas rotinas na memória de programa são chamados vetores de interrupção (*interrupt vector*) e são armazenados geralmente em uma tabela na memória RAM, permitindo sua modificação caso seja necessário.

As interrupções são ações tratadas diretamente pelo hardware, o que as tornam muito rápidas e disponíveis em qualquer ponto do programa. Assim sendo, quando uma interrupção acontece, o programa é interrompido, uma sequência específica de comandos (definida pelo programador) é executada e depois o programa continua a ser executado do mesmo ponto em que estava. Interrupções de hardware foram introduzidas como forma de evitar o desperdício de tempo computacional em rotinas de software à espera de eventos específicos (chamadas de *polling loops*). Por exemplo, quando você pressiona um botão em um controle remoto, o microcontrolador registrará essa ação e responderá mudando o canal ou aumentando o volume, etc. Se o microcontrolador gastasse a maior parte do tempo verificando se cada um dos botões foi pressionado, isso não seria nada prático. Isto era exatamente o que vínhamos fazendo até agora.

Existem 57 fontes de interrupção, externas e internas, que podem ser habilitadas no microcontrolador Atmega 2560. Apenas para citar algumas, podemos destacar as interrupções dos timers (temporizadores), interrupções de fim de escrita na memória EEPROM, interrupções de fim de conversão no módulo conversor A/D, interrupções por mudança de estado lógico em alguns pinos do microcontrolador, entre outras.

11.2 Interrupções externas

O microcontrolador ATmega 2560 possui 32 pinos que podem ser utilizados como fontes de interrupção externa. Entretanto, apenas alguns desses pinos estão disponíveis na placa do Arduino Mega 2560. Essas interrupções são geradas por um sinal externo ligado aos pinos. Desta maneira, podemos identificar e processar imediatamente um sinal externo.

Podemos usar esse tipo de interrupção para diversas finalidades como, por exemplo, a comunicação entre microcontroladores, garantindo sincronismo, o reconhecimento de um botão ou outro sinal do sistema que necessite de uma ação imediata.

As interrupções externas podem ocorrer quando há uma mudança de nível lógico nos pinos, ou podem ser identificadas pela borda de transição do sinal, ou mesmo o próprio nível do sinal, podendo ser configuradas para reagir a uma mudança de nível, apenas a uma borda de subida (transição do nível baixo para nível alto), apenas a uma borda de descida (transição do nível alto para o nível baixo) ou ao nível lógico baixo.

No Arduino Mega 2560, os pinos digitais 2, 3, 18, 19, 20 e 21 podem ser usados como entrada de sinal para interrupção externa. Para habilitar qualquer uma das interrupções externas, é usada a função:

```
attachInterrupt(digitalPinToInterrupt(pin), ISR, mode),
```

onde pin é um dos pinos que podem ser usados como entrada de sinal, ISR é o nome da função que será chamada e executada quando ocorrer a interrupção, e modo define em qual tipo de variação do sinal a interrupção será disparada, podendo ser uma das 4 opções abaixo:

LOW: Dispara a interrupção quando a tensão no pino está em 0 V;
CHANGE: Dispara sempre que o sinal no pino muda de estado, borda 0 V (0) para 5 V (1) ou vice-versa;
RISING: Dispara somente na borda de subida, 0 V (0) para 5 V (1);
FALLING: Dispara somente na borda de descida, 5 V (1) para 0 V (0).

O exemplo a seguir caracteriza o uso desse tipo de interrupção. O circuito usado é mostrado na figura 1 e o programa é apresentado no quadro seguinte. O sketch faz com que o LED conectado ao pino 13 fique piscando com uma frequência de 5 Hz. Quando a chave conectada ao pino 2 (fonte da interrupção) é pressionada, o LED conectado ao pino 12 muda de estado, enquanto o do pino 13 permanece piscando.

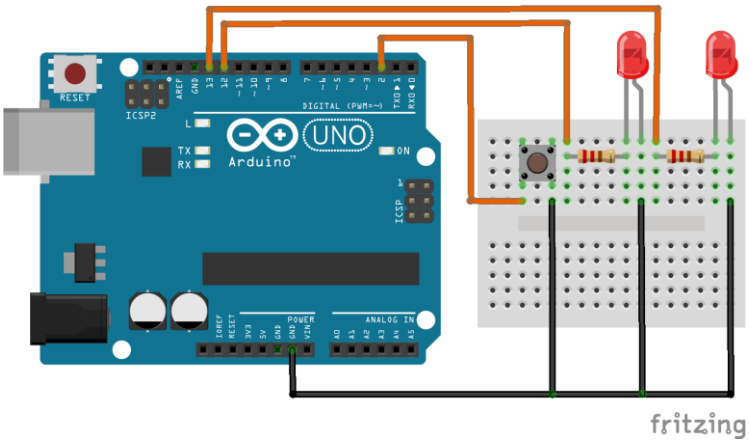

Figura 1 – Circuito para teste do programa com interrupção externa.

```
void interrupcao()
{
  digitalWrite(12, !digitalRead(12));
}

void setup()
{
  pinMode(13, OUTPUT);
  pinMode(13, OUTPUT);
  pinMode(2, INPUT_PULLUP);
  digitalWrite(12, LOW);
  attachInterrupt(digitalPinToInterrupt(2), interrupcao, RISING);
}

void loop()
{
  digitalWrite(13, HIGH);
  delay(100);
  digitalWrite(13, LOW);
  delay(100);
}
```

No programa, a função setup( ) configura os pinos de entrada e saída, leva o pino 12 ao nível lógico baixo e habilita a interrupção externa, gatilhada pela borda de subida (RISING) do sinal conectado ao pino digital 2. Além disso, estabelece que a função que atenderá à interrupção é interrupcao( ), declarada no início do programa. A função loop( ) mantém o LED conectado ao pino 13 piscando numa frequência de 5Hz.

Na borda de subida do sinal conectado ao pino 2, a interrupção é disparada e a função interrupcao( ) é executada imediatamente. Essa função apenas inverte o estado lógico do pino 12, no qual está conectado o outro LED. Perceba que a interrupção só será disparada quando o usuário soltar o botão, visto que é nessa condição que o nível lógico muda de baixo para alto.

Geralmente, uma ISR deve ser tão curta e rápida quanto possível. Se o sketch usa várias ISRs, apenas uma pode ser executada de cada vez, outras interrupções serão executadas depois que a atual terminar em uma ordem que depende da prioridade que eles possuem. A função interna millis( ) depende de interrupções para contar, de modo que nunca irá incrementar dentro de uma ISR. Uma vez que a função delay( ) também exige que as interrupções funcionem, não funcionará se for chamada dentro de uma ISR. A função micros( ) funciona inicialmente, mas começará a se comportar erraticamente após 1-2 ms. A função delayMicroseconds( ) não usa nenhuma interrupção, então funcionará normalmente dentro de uma ISR. Dados seriais recebidos durante a execução de uma ISR podem ser perdidos.

Tipicamente, variáveis globais são usadas para passar dados entre uma ISR e o programa principal. Para garantir que as variáveis compartilhadas entre uma ISR e o programa principal sejam atualizadas corretamente, declare-as como voláteis (volatile).

Para desabilitar uma interrupção externa, use a função:

```
detachInterrupt(digitalPinToInterrupt(pin)),
```

onde pin é o pino que se deseja desabilitar como fonte da interrupção.

Para configurar e habilitar as interrupções de outros periféricos, consulte o *datasheet* do microcontrolador ATmega 2560 para verificar como manipular os registradores de controle de cada fonte de interrupção.

11.3 Timers

Alguns programas usam uma espécie de “relógio em miniatura”. Na prática, esses “relógios”, conhecidos como *timers*, ou ainda *contadores*, são na verdade registradores de 8 ou 16 bits cujo conteúdo é automaticamente incrementado por cada pulso do oscilador do microcontrolador ou um sinal externo. A operação dos timers pode gerar diversos eventos de interrupção.

Se o timer usa um oscilador de quartzo para o seu funcionamento interno, então ele pode ser usado para medir o tempo entre dois eventos. Se o valor do registrador é T1 no momento em que se inicia a medição, e T2, no momento em que termina, então, o tempo decorrido é igual ao resultado da subtração T2-T1.

Se os registradores usam pulsos provenientes de fontes externas conectadas a um pino do microcontrolador, então o timer é transformado em um contador. Obviamente, é o mesmo circuito eletrônico, capaz de operar em dois modos diferentes. Neste segundo caso, os pulsos a serem contados provêm de um pino de entrada do microcontrolador e sua duração (largura) é mais indefinida. É por isso que eles não podem ser usados para medir o tempo, mas para outros fins, tais como contagem de produtos em uma linha de montagem, o número de rotação do eixo de um motor, etc (dependendo do sensor em uso).

Na prática, os impulsos gerados pelo oscilador de quartzo são levados para o circuito que incrementa o número armazenado no registrador associado ao timer. Estes pulsos podem ser levados diretamente, uma vez por cada ciclo de máquina, ou ainda através de um divisor. Por exemplo, se o clico de máquina tem a duração de 1 µs, isto é, quando o oscilador de quartzo funciona a 4 Mhz e o o sinal de clock é dividido por 4, então, o contador será incrementado um milhão de vezes por segundo.

A figura 2 mostra, de forma simplificada, como funciona um timer interno ao microcontrolador.

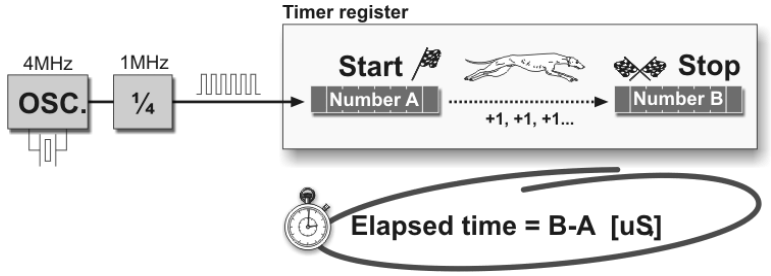

Figura 2 – Operação dos timers internos.

A figura mostra que os pulsos do oscilador de 4 Mhz, usado como exemplo, são divididos por 4. Assim, após essa divisão, temos pulsos a uma frequência de 1 Mhz. Esses pulsos são ligados diretamente ao registrador associado ao timer, e, a cada pulso, o conteúdo desse registrador é incrementado. Se o registrador usado for de 8 bits, é fácil medir intervalos de tempo de até 256 microssegundos, pois é o valor máximo que o contador pode assumir. O intervalo de tempo transcorrido entre dois instantes é calculado como mostra a figura 2.

Essa restrição pode ser facilmente superada de diversas maneiras, como através do uso de um oscilador mais lento, aplicando-se um divisor maior, usando um registrador com mais bits, ou usando interrupções.

Um prescaler (divisor) é um dispositivo eletrônico usado para reduzir a freqüência por um fator predeterminado. A fim de gerar um pulso em sua saída, é necessário levar 2, 4, 8, 16 ou mais pulsos em sua entrada. A maioria dos microcontroladores possui um ou mais prescalers embutidos e sua taxa de divisão pode ser alterada a partir do programa acessando registradores de controle. O divisor é utilizado quando é necessário medir períodos de tempo mais longos. Um prescaler é normalmente compartilhado por vários temporizadores, de modo que não pode ser independentemente configurado para cada um.

11.4 Usando interrupções de timers

Se o registrador do timer é de 8 bits, o maior número que este pode armazenar é 255. Quanto aos registradores de 16 bits, o maior número é 65535. Se esse número for ultrapassado, o temporizador será automaticamente reiniciado e a contagem vai começar do zero novamente. Esta condição é chamada de *estouro* do timer, ou overflow.

O estouro de um timer pode ser configurado para pode gerar uma interrupção, o que dá possibilidades completamente novas ao projetista. Por exemplo, em um relógio digital, o estado dos registradores usados para a contagem de segundos, minutos, horas ou dias pode ser alterado em uma rotina de interrupção gerada precisamente a cada um segundo.

A figura 3 ilustra o uso de uma operação com interrupção em um timer. Se a interrupção por estouro no timer for configurada, o conteúdo do registrador adicional é incrementado. O conteúdo desse registrador adicional é usado para controlar determinadas funções no programa principal. A figura mostra que o clock do oscilador é inicialmente dividido por 4 além de usar um prescaler que faz a divisão da frequência por um fator N, aumentando o tempo de contagem do timer.

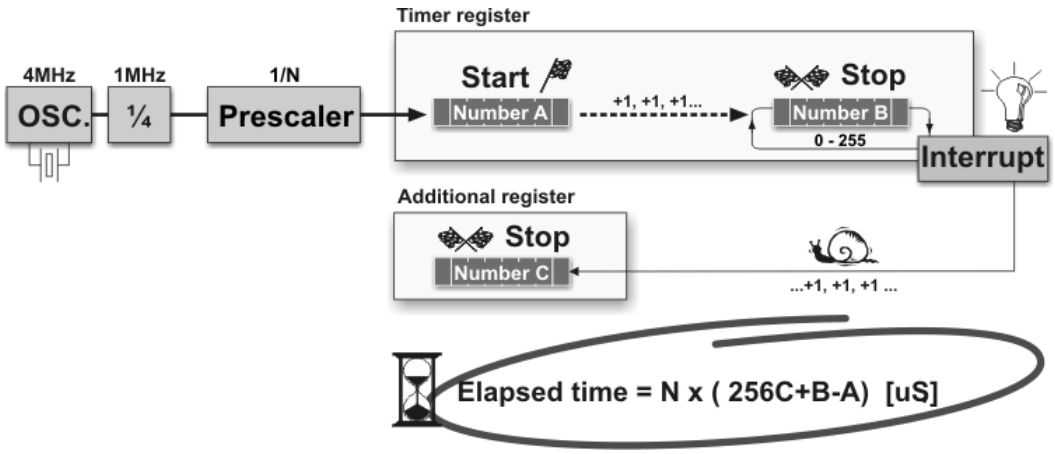

Figura 3 – Uso de interrupção no timer interno.

11.5 Timers no microcontrolador ATmega 2560

O ATmega 2560 possui 6 timers internos, sendo 2 de 8 bits (TIMER0 e TIMER2) e 4 de 16 bits (TIMER1, TIMER3, TIMER4 e TIMER5). Os timers podem operar como timers ou contadores.

Acessar diretamente os registradores de controle dos timers afeta diretamente a operação de algumas funções internas do Arduino. Por exemplo, o TIMER0 é usado pelas funções de geração de atrasos como delay( ) e millis( ). Todos os timers também são usados para controlar a geração de sinais PWM em alguns pinos do microcontrolador (o assunto PWM será abordado com mais detalhes no próximo capítulo). Dessa forma, é importante estar ciente de que o acesso direto aos registradores de controle dos timers pode fazer com que essas aplicações deixem de funcionar com as funções nativas do Arduino.

Como exemplo, analisaremos com mais detalhes o TIMER3. Suas principais características são:

• Operação como timer ou contador de 16 bits;
• Prescaler programável (÷1, ÷8, ÷64, ÷256, ÷1024);
• Fonte de clock selecionável como interna ou externa;
• Interrupção por estouro do contador;
• 3 unidades comparadoras de saída independentes com geração de interrupção;
• 1 unidade de captura de entrada;
• Gerador de frequência;
• Geração de sinais PWM;

Um dos modos de funcionamento desse timer permite que seja gerado um sinal de interrupção quando o conteúdo do timer se iguala ao conteúdo de um outro registrador, chamado de registrador de comparação de saída (OCR – Output Compare Register). O módulo TIMER3 possui 3 registradores de comparação de saída (OCR3A, OCR3B e OCR3C).

O programa do quadro a seguir utiliza a interrupção por comparação com o registrador OCR3A para fazer com que o LED conectado ao pino 13 do Arduino fique piscando numa frequência de 5 Hz.

```
void setup(){
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);

  TCCR3A = TCCR3A & 0b11111100;
  TCCR3B = TCCR3B | 0b00001011;
  TCCR3B = TCCR3B & 0b11101011;
  OCR3A = 25000;
  TIMSK3 = TIMSK3 | 0b00000010;
}

void loop(){
}

ISR(TIMER3_COMPA_vect){
  digitalWrite(13, !digitalRead(13));
}
```

A função setup( ) configura o pino 13 como saída digital e a inicializa com nível lógico baixo. Em seguida, há 5 linhas de código para manipulação do TIMER3, que é configurado para gerar uma interrupção a partir da comparação com o registrador OCR3A. As três primeiras linhas afetam os registradores TCCR3A e TCCR3B (TCCR - *Timer/Counter Control Registers*). Essas linhas ajustam o modo de operação do timer para comparação com o valor do registrador OCR3A. Esse modo zera o conteúdo do timer quando a comparação for efetivada, possibilitando o início imediato de uma nova contagem. As linhas também configuram o prescaler para que a fonte de clock do contador seja o sinal de clock do sistema dividido por 64, o que faz com que o timer seja incrementado a cada 4 µs (para um oscilador de 16 MHz, como é o caso do Arduino Mega 2560). Além disso, o registrador OCR3A é inicializado com o valor 25000, o que significa dizer que o valor do timer será igual ao valor de OCR3A a cada 25000 x 4 µs = 100 ms. Finalmente, a interrupção por comparação com OCR3A é habilitada, setando-se o bit 1 do registrador TIMSK3 (Timer 3 Interrupt Mask)

O Arduino passará a maior parte do tempo esperando o sinal de interrupção dentro da função loop( ), já que ela não tem nenhum comando a executar. Quando a interrupção ocorrer, a função ISR( ) será executada, invertendo o estado lógico do pino 13.

A função ISR( ) não faz parte das funções nativas do Arduino. Na verdade, ela pertence à biblioteca de funções AVR da Atmel e pode ser usada para definir a função de atendimento à qualquer interrupção, inclusive as externas, estudadas anteriormente. O formato geral da declaração é:

```
ISR({vector}_vect){
    // Perform task here
}
```

onde {vector} é o nome do vetor da interrupção, definido no *datasheet* do microcontrolador.

Para obter mais informações sobre os tipos de interrupções e a forma de tratamento de cada uma delas, consulte o datasheet do microcontrolador Atmega 2560.■