Leitura de sensores de resposta binária com uso de interrupções

Apesar de praticamente todos os sensores digitais de resposta binária permitirem a leitura do seu estado a partir do método digitalRead, de forma simples, temos algumas exceções que necessitam de uma técnica um pouco mais elaborada. Eles são justamente os dois sensores vistos por último na relação anterior: os encoders e o sensor de vazão. Um botão, um sensor de som, um sensor de chama, etc, são ativados poucas vezes por segundo e além disso geralmente não nos interessa quantas vezes isso ocorreu. Portanto, a técnica que utilizamos é dentro do loop, com um tempo de espera pequeno, ler a porta consecutivas vezes até perceber uma mudança de estado e, a partir daí, tomar uma ação. Já nos sensores do tipo encoder e no sensor de vazão, o total de mudanças de estado (chamadas de pulsos), é fundamental para estimar as grandezas que estes sensores medem (frequência, velocidade, rotação, vazão, etc). E é aí que temos um problema: não podemos ler o estado dentro de um loop e a cada vez que HIGH for detectado mudar o estado, pois não sabemos com que frequência isso deverá ser feito (e ela muda de acordo com a velocidade do que está senso medido). E mesmo que a gente saiba, pode acontecer de uma mesma mudança de estado ser lida duas ou mais vezes apenas porque a velocidade de execução está muito alta. Então, perdemos totalmente a precisão da leitura. É aí que entram as interrupções.

O Arduino Uno possui duas portas de interrupção: a interrupção 0 (presente na porta digital 2) e a interrupção 1 (presente na porta digital 3). Estas interrupções, quando ativadas, funcionam mesmo que o código esteja travado em uma linha (em um delay muito longo por exemplo). Além disso, possuem uma característica muito interessante: através de um método próprio, podemos vincular um método criado por nós à interrupção para que ele seja executado toda vez que esta interrupção perceber uma mudança de estado. Este método é o attachInterrupt:

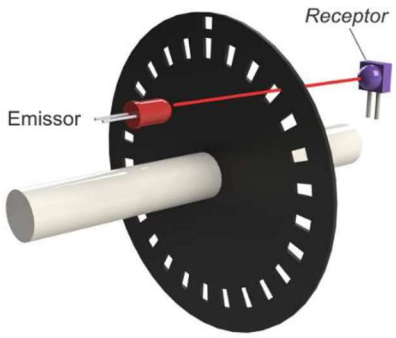
```
attachInterrupt( <interrupcao>, <metodo>, <modo> );
```

```
<interrupcao>
   número da interrupção que queremos monitorar (0 - pino 2 ou 1 - pino 3)

<metodo>
método que será chamado a cada nova chamada da interrupção

<modo>
modo de disparo da interrupção. Pode receber os seguintes valores:

   - LOW: dispara quando o estado da porta for para LOW
   - CHANGE: dispara quando o estado da porta mudar (LOW para HIGH ou HIGH para LOW)
   - RISING: dispara quando o estado do porta mudar de LOW para HIGH apenas
   - FALLING: dispara quando o estado da porta mudar de HIGH para LOW apenas
```

O primeiro parâmetro do attachInterrupt é a interrupção na qual ligamos o sensor que desejamos monitorar (0, caso esteja na digital 2 ou 1 caso esteja na digital 3). O segundo parâmetro é qual é o método que queremos vincular à interrupção de modo a ser disparado sempre que uma mudança de estado for percebida pela porta. O terceiro parâmetro é a definição de como a porta deve perceber uma mudança de estado, disparando a interrupção quando ela for detectada. Aqui, temos uma mudança na lógica de leitura do sensor: ao invés de realizar múltiplas leituras da porta no loop, de forma rápida e consecutiva, até que uma mudança de estado seja percebida (executando algo quando isso ocorrer), criamos uma forma de só executar algo que desejamos quando uma mudança for percebida, sem a necessidade de uma leitura constante dentro do loop.

Além disso, para ativar a interrupção e ela agir nas mudanças de estado, utilizamos o método **sei()** e para interromper a interrupção utilizamos o método **cli()**. Para entender e ver na prática o uso das interrupções, vamos a dois exemplos práticos.

Sensor encoder: exemplo detalhado com uso de interrupções

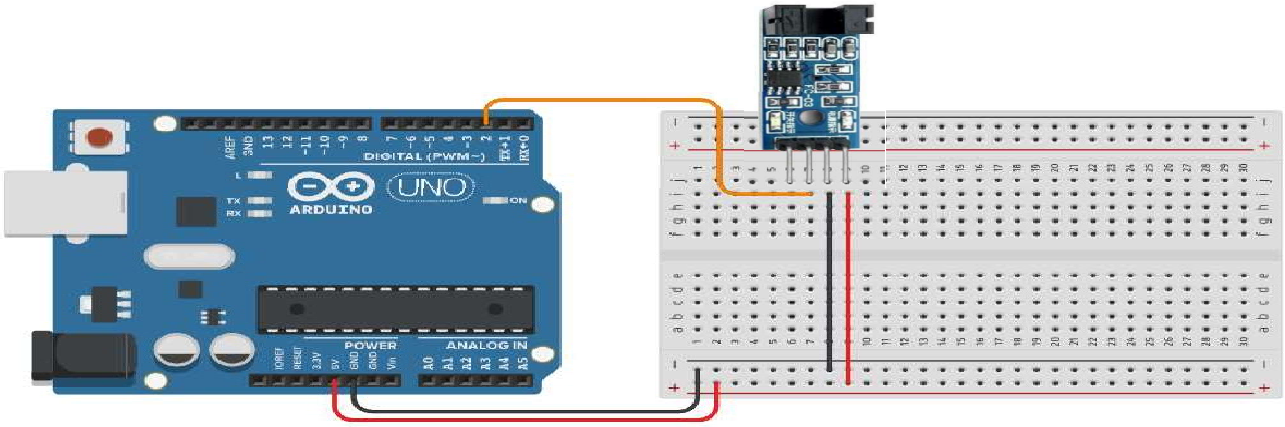


Para o nosso primeiro exemplo, vamos considerar um encoder do tipo óptico. O mesmo possui um emissor de luz infravermelha e um receptor de luz infravermelha paralelo a ele. Quando algo passa em frente a esta luz e a corta, o encoder modifica seu estado de LOW para HIGH. Então, este tipo de sensor geralmente necessita de um elemento preso à parte móvel do objeto que queremos monitorar a frequência de execuções (no caso do encoder óptico, geralmente motores) para que ele fique continuamente interrompendo e liberando a passagem da luz, gerando assim, pulsos. No nosso caso, iremos utilizar um disco preso a um motor. Este disco possui diversos furos e conforme gira faz com que o encoder mude seu estado constantemente. Todos os sensores do tipo encoder possuem no mínimo 3 pinos: um pino de sinal (que deve ser conectado à nossa porta de interrupção), um VCC e um GND. Alguns deles possuem um pino adicional analógico para a geração de ondas caso se deseje visualizar o comportamento da aceleração e desaceleração em um osciloscópio ou gráfico.

No nosso exemplo, utilizaremos um encoder óptico de 4 pinos: o primeiro é o analógico para produção de sinal e não será utilizado, o segundo é o digital de dados que ligaremos à porta 2 (interrupção 0), o terceiro é o GND e o quarto é o VCC. Assim, teremos

o seguinte esquema eletrônico:

Nosso primeiro objetivo no código é durante exatamente um segundo capturar o total de pulsos gerados pelo movimento de nosso disco. Após, utilizando conceitos da física,



podemos converter o número de voltas em uma velocidade em metros por segundo.

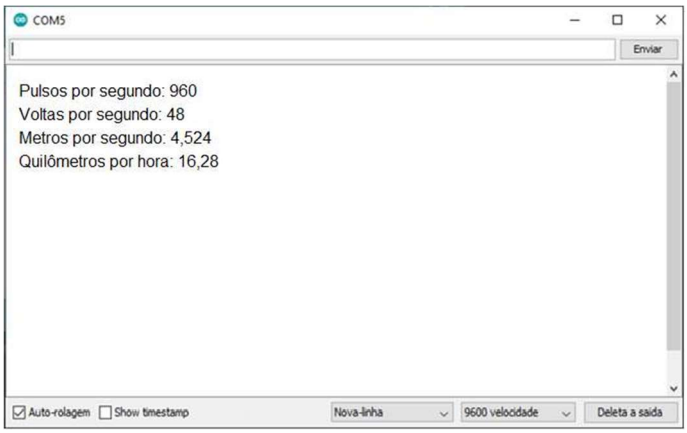
velocidade (m/s) = voltas por segundo . 2 . π . r

O disco do nosso exemplo possui um raio de 1,5 cm (0,015 m) e 20 furos. Como cada furo estimula um pulso, 20 pulsos equivalem a uma volta completa. Deste modo, para descobrir o total de voltas, dividimos o total de pulsos por 20. Para a contagem destes pulsos, precisamos criar um método que cada vez que for chamado aumente em um o valor de uma variável que conte os pulsos. Este método deve ser vinculado à interrupção 0 para cada vez que o estado mudar de LOW para HIGH (ou seja, mudança do tipo RISING), um novo pulso ser contado. Assim, teremos o seguinte código:

```
int pulsos;

void contaPulso(){
  pulsos++;
}

void setup(){
  attachInterrupt(0, contaPulso, RISING);
  Serial.begin(9600);
}

void loop(){
  pulsos = 0;
  sei();
  Serial.println("Pulsos por segundo: " + (String)pulsos);
  Serial.println("Voltas por segundo: " + (String)voltasSegundo);
  Serial.println("Metros por segundo: " + (String)metrosSegundo);
  Serial.println("Quilômetros por hora: " + (String)metrosSegundo);
}
```

Entendendo o código: começamos criando uma variável inteira chamada de *pulsos*, que irá contar o número de pulsos lidos. Essa variável será incrementada pelo método

contaPulso a cada vez que ele for chamado pela interrupção, aumentando seu valor em 1. Dentro do setup, usamos o attachInterrupt para vincular o método contaPulso à interrupção 0 que será acionada sempre que perceber uma mudança de LOW para HIGH (ou seja, do tipo RISING). Além disso no setup inicializamos nossa Serial para que possamos escrever nela os resultados e visualizá-los posteriormente. Após, em cada repetição do loop, zeramos a variável *pulsos*, ativamos a nossa interrupção com o método *sei* (passando a contabilizar os pulsos), paramos a execução do código por exatamente 1 segundo e após desativamos a nossa interrupção com o método *cli* (terminando assim a contagem de pulsos). Como a interrupção funciona independentemente da execução do código principal, mesmo com ele parado no delay, os pulsos seguem sendo contabilizados pelo período estabelecido. Ao final de 1 segundo, podemos finalmente pegar o total de pulsos contabilizados e dividir por 20 para descobrir o total de voltas dadas neste segundo e com estas voltas descobrir a velocidade em metros por segundo e após (multiplicando por 3,6) em quilômetros por hora. Se tudo estiver certo, ao abrir o Monitor Serial devemos ter uma saída parecida com esta:

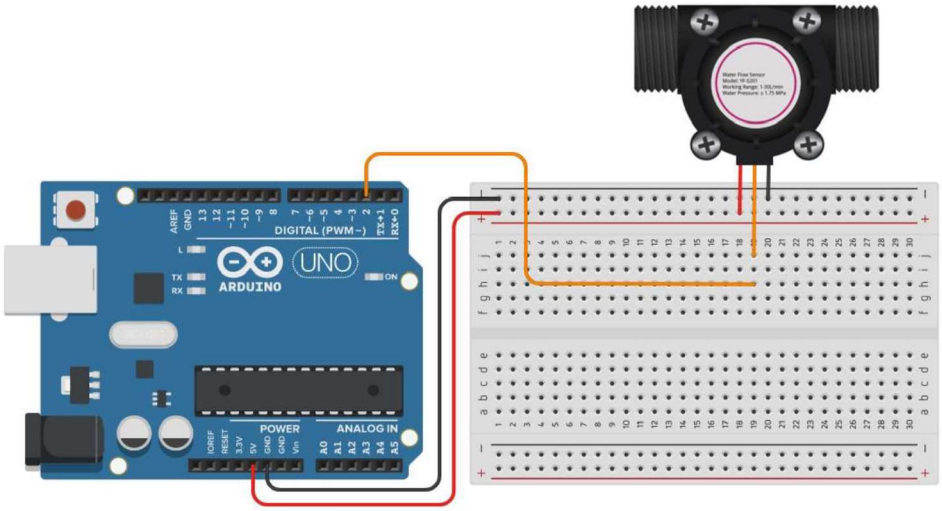


Esse processo com uso de interrupções pode ser utilizado por qualquer tipo de sensor que nos devolva uma mudança de estado que desejamos monitorar a frequência (ocorrências em um determinado tempo). Então, este exemplo pode ser utilizado de base para situações similares ou para sensores de funcionamento similar.

Sensor de vazão: exemplo detalhado com uso de interrupções

De forma similar ao exemplo anterior, um sensor de vazão emite um pulso a cada volta de sua hélice. Para isso, iniciamos ligando o seu pino de sinal (geralmente o do meio, mas isso pode mudar de acordo com o fabricante) à nossa porta de interrupção (no caso deste exemplo, a interrupção 0 localizada na porta digital 2).

Assim, teremos um esquema eletrônico similar ao representado abaixo:



Se contarmos o total de pulsos em um segundo (que no caso equivale diretamente a voltas) e multiplicarmos por 5,5 (fator já calculado como o tamanho do giro é conhecido) conseguimos estimar a vazão de ar ou líquido atravessando o sensor (litros por segundo). Multiplicando esse valor por 60, temos a estimativa de litros por minuto e multiplicando por 60 novamente temos a estimativa de litros por hora. Nosso código será então baseado na realização da contagem dos pulsos (voltas) em 1 segundo, para obter os demais valores desejados. Assim como no exemplo anterior do encoder, a contagem será realizada por um método contaPulso que apenas incrementa a variável inteira *pulsos* (zerada ao final de cada segundo). Assim, temos:

```
int pulsos;

void contaPulso(){
  pulsos++;
}

void setup(){
  attachInterrupt(0, contaPulso, RISING);
  Serial.begin(9600);
}

void loop(){
  pulsos = 0;
  sei();
  delay(1000);
  cli();

  float litrosSegundo = pulsos / 5.5;
  float litrosMinuto = litrosSegundo * 60;
  float litrosHora = litrosMinuto * 60;

  Serial.println("Litros por segundo: " + (String)litrosSegundo);
  Serial.println("Litros por minuto: " + (String)litrosMinuto);
  Serial.println("Litros por hora: " + (String)litrosHora);
}
```

Entendendo o código: o funcionamento é análogo ao funcionamento do código do encoder então a mesma explicação vale para este exemplo. A única diferença é que após a obtenção do total de pulsos em um segundo, utilizamos este total para calcular informações diferentes (litros por segundo, por minuto e por hora). Se tudo funcionou corretamente, devemos ter uma saída similar à essa em nosso Monitor Serial: