Iniciando com a programação Arduino

Um projeto simples e interessante para começarmos a programação com o Arduino, é realizar o controle, via programação, de leds conectados em suas portas digitais. Relembrando, as portas digitais do Arduino podem funcionar como entrada (para ler informações externas através de sensores) ou saída. Quando funcionam como saída, podem se comportar como portas digitais convencionais, fornecendo 0V de tensão (equivalente ao bit 0) ou 5V de tensão (equivalente ao bit 1), ou então modulando o sinal de saída digital de modo que seja consumido como uma tensão variável e que podemos ajustar (caso das portas PWM). Faremos 2 exemplos partindo desses princípios: o primeiro apenas controlando a sequencia e tempo de acendimento dos leds e o segundo controlando a sua luminosidade através de uma porta PWM. Mas antes, necessitamos conhecer alguns conceitos iniciais.

Configurando as portas digitais como entrada ou saída

A configuração de portas digitais como entrada ou saída é simples e realizada através de um método que recebe apenas dois parâmetros. Ele dirá se ela lê dados, envia dados ou é uma porta especial com uso de um resistor de pullup interno. Não cabe o aprofundamento sobre pullup, mas nos basta saber que esse tipo de configuração com resistor é utilizada quando queremos ler um estado alto ou baixo (0 ou 1) eliminando pequenas flutuações que podem ser interpretadas de forma errada. Isso é especialmente útil quando usamos botões ou chaves interruptoras que ao serem totalmente pressionados ou totalmente soltos, passam por estados intermediários que não desejamos detectar. Para essa configuração, utilizamos o método abaixo:

```
pinMode( <porta>, <modo> );

<porta>
   número da porta que queremos definir o modo

<modo>
   modo de funcionamento da porta, que pode receber os
   valores. Pode receber os valores:

      - INPUT: porta funciona como entrada (leitura)
      - OUTPUT: porta funciona como saída (controle)
      - INPUT_PULLUP: funciona com porta com pullup (botões)
```

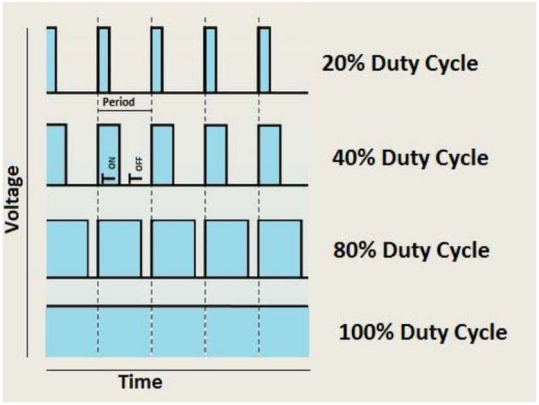
Escrevendo o estado de saída da porta digital convencional

Quando uma porta digital está configurada como porta de saída (OUTPUT), podemos escrever o seu estado, fazendo com que a mesma ligue (estado alto, com tensão de 5V) ou desligue (estado baixo, com tensão de 0V). Para isso, utilizamos o método abaixo:

```
digitalWrite( <porta>, <estado> );

<porta>
   número da porta que queremos definir o estado de saída

<estado>
   Estado lógico da porta. Pode receber os valores:

      - LOW: estado baixo (0V, desligada)
      - HIGH: estado alto (5V, ligada)
```

Escrevendo o estado de saída da porta digital PWM

As portas PWM, assim como as portas digitais convencionais, também só conseguem trabalhar com dois estados e, consequentemente, com tensões absolutas de 0V ou de 5V. Porém, enquanto nas portas digitais convencionais o estado de saída permanece constante após sua escrita, nas portas PWM a tensão é gerada em forma de onda que possui uma determinada frequência, alternando entre os dois estados um determinado número de vezes por segundo. Alterar a frequência da alternância entre os dois estados, acaba criando uma tensão média na saída, mais próxima de 0V se o espaçamento for muito alto ou mais próxima de 5V se o espaçamento for muito curto, perto de uma onda linear. Chamamos isso de Duty Cycle. Vamos a um exemplo, conforme a figura ao lado: perceba que quando o Duty Cycle é de 100%, o tempo em que a tensão permanece em 5V é de 100% e, portanto, a tensão de saída é muito próxima a 5V. Já quando o Duty Cycle está com 20%, isso significa que em somente 20% do tempo a tensão está em 5V (estando nos outros 80% em 0V) e isso faz com que a tensão média da saída seja 20% dos 5V (no caso, 1V). Se o Duty Cycle fosse de 50%, durante apenas metade do tempo a tensão estaria em 5V, gerando então uma tensão de saída de metade dos 5V (no caso, 2,5V). O Arduino permite então em suas portas PWM, que o valor de Duty Cycle seja alterado, alterando a tensão média de saída da porta gerada pela



modulação dos estados, porém ao invés de trabalhar com percentuais, a placa trabalha com níveis que podem ir de 0 (equivalente a 0V) até 255 (valor máximo, equivalente à 5V). Qualquer valor intermediário entre 0 e 255 gera então uma tensão de saída entre 0V e 5V. Para isso, utilizamos o seguinte método:

```
analogWrite( <porta>, <valor_pwm> );

<porta>
   número da porta que queremos definir o estado de saída

<valor_pwm>
   valor da modulação PWM entre 0 e 255 que irá gerar uma
   tensão média de saída entre 0V e 5V
```

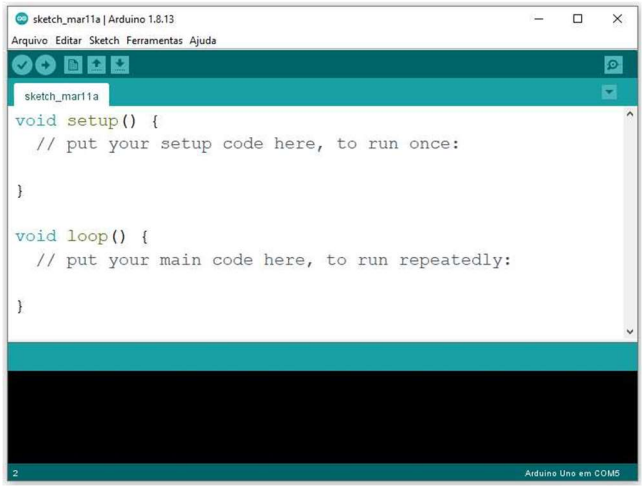
Criando paradas temporizadas

Durante a execução do código, muitas vezes após modificar o estado de uma porta, ler o valor de um led e enviar uma ordem para um atuador, é necessária uma parada temporizada que garanta que aquele estado se mantenha por um determinado tempo mínimo e que a próxima ação só seja disparada após decorrido esse tempo. Para isso, temos no Arduino alguns comandos que criam estas paradas com um tempo definido. Veremos nesse momento o mais simples e utilizado deles, sendo que os demais serão aprofundados e só farão sentido quando falarmos das interrupções de sistema. Então, para criar uma parada temporizada no sistema, utilizamos o seguinte método:

```
delay( <tempo> );

<tempo>
   tempo de parada, informado em milissegundos (ou seja, cada
   1000 equivale a 1 segundo)
```

Escrevendo nosso código e entendendo o setup e o loop

Ao criarmos um novo projeto na Arduino IDE, percebemos que um pequeno trecho de código automaticamente já é preenchido. Nele, existem dois métodos já prontos para podermos utilizar: o setup e o loop. Estes são métodos estruturais do Arduino, previamente programados com finalidades especiais e que controlam a execução dos comandos internos aos seus blocos, em momentos determinados.



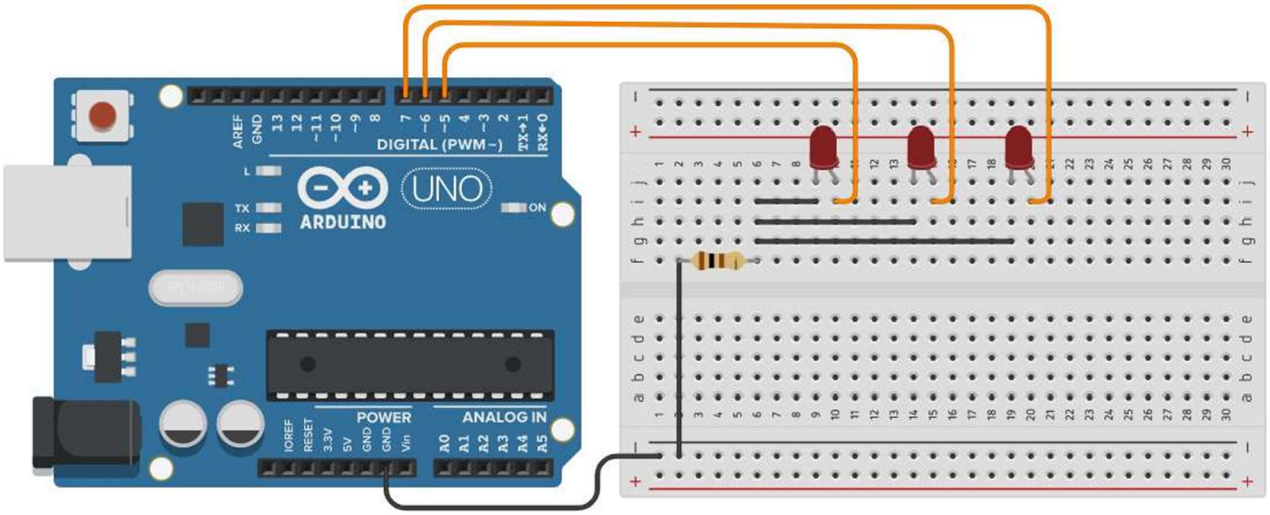
setup: o método setup é o primeiro a ser executado quando a placa Arduino é energizada (ligada) e enquanto todos os comandos escritos dentro do seu bloco não forem executados, garantidamente nada mais é executado. Isso é especialmente pensado para reunirmos em um único ponto os comandos que configuram, inicializam e preparam a placa para a execução do programa principal, trazendo maior segurança, estabilidade e confiabilidade. Geralmente é aqui que inicializamos os componentes que precisam ser previamente configurados ou ajustados, definimos os valores de inicialização de nossas variáveis, iniciamos nossas portas do tipo Serial, etc.

loop: o método loop é o principal do Arduino e o responsável pela execução do nosso código de forma contínua. Trata-se de um laço de repetição infinito que lê cada uma das linhas de comando do seu bloco, de forma sequencial e, ao chegar ao seu final, volta para o início e começa novamente a execução desde a primeira linha novamente. Assim, o Arduino não utiliza o paradigma de orientação a eventos, onde temos listeners especializados (rotinas que monitoram um estado ou comportamento) que disparam funções programadas. Aqui, a cada loop, coletemos e verificamos dados para, com base neles, podermos tomar decisões e/ou executar ações. Isso torna a programação muito mais natural, simples e amigável, porém traz problemas para a execução de tarefas simultâneas, visto que cada comando precisa esperar sua hora de ser executado. Geralmente o loop inicia ou termina com uma interrupção temporizada, controlando assim, sua frequência de execução.

Exemplo 1: ligando e desligando leds

Neste primeiro exemplo, nosso objetivo é realizar o controle de 3 leds com o Arduino, ligando-os e desligando-os em uma sequência pré-determinada e em um tempo pré-determinado. Para isso, vamos ligar cada um dos 3 leds em portas digitais diferentes (no nosso exemplo, nas portas 5, 6 e 7), que possam ter seu estado de saída alterado para alto, garantindo uma tensão de 5V (através do seu pino indicado para isso, geralmente um pino mais comprido ou com um joelho). O outro pino deverá ser ligado ao negativo (no Arduino, à porta GND) para que a corrente que entra no led possa encontrar um caminho de saída e retorno à sua fonte de tensão (a própria placa). Por segurança, como visto nas aulas iniciais de conceitos de eletrônica, colocamos um resistor neste caminho (neste exemplo

de 100Ω), obrigando que a corrente passe por ele, reduzindo-a assim em todos os 3 segmentos, garantindo proteção para que o led não queime. Assim, a montagem do circuito deverá ficar como a abaixo:



Toda vez que uma das portas tiver seu estado de saída alterado para alto, ela fornecerá uma tensão de 5V que fará com que uma corrente atravesse o led (saindo pelo GND) e o iluminando. Para isso, porém, necessitamos que antes as portas digitais sejam configuradas como portas de saída e é justamente por aí que vamos começar a nossa programação. Dentro do setup, duas coisas são necessárias: tornar as 3 portas digitais portas de saída (através do método pinMode) e garantir que elas iniciam em estado baixo (através do método digitalWrite). Então, teremos:

```
void setup(){
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);

  digitalWrite(5, LOW);
  digitalWrite(6, LOW);
  digitalWrite(7, LOW);
}
```

Agora, dentro do loop, podemos alternar as portas entre os estados alto e baixo (com o método digitalWrite) e, assim, acender e apagar os leds. A criação de sequências diferentes consequentemente implica em diferentes padrões de comportamento para o acendimento dos leds. Deste modo, podemos utilizar a programação para gerar efeitos de movimento e sequência. Vamos ver três entre as infinitas possibilidades e o efeito do código no padrão:

```
void loop(){

  digitalWrite(5, HIGH);
  delay(1000);
  digitalWrite(5, LOW);
  delay(1000);

  digitalWrite(6, HIGH);
  delay(1000);
  digitalWrite(6, LOW);
  delay(1000);

  digitalWrite(7, HIGH);
  delay(1000);
  digitalWrite(7, LOW);
  delay(1000);

}
```

Padrão 1: caso nosso código do loop seja igual a este, o padrão de funcionamento dos leds será o seguinte: o primeiro led (da porta 5) liga e após um segundo, desliga. Após mais um segundo, o segundo led (da porta 6) liga e um segundo depois, desliga. Após mais um segundo, o terceiro led (da porta 7) liga e um segundo depois, desliga. Assim temos um acendimento sequencial e um desligamento sequencial, led a led, da esquerda para a direita.

```
void loop(){

  digitalWrite(5, HIGH);
  delay(1000);
  digitalWrite(6, HIGH);
  delay(1000);
  digitalWrite(7, HIGH);
  delay(1000);

  digitalWrite(5, LOW);
  digitalWrite(6, LOW);
  digitalWrite(7, LOW);
  delay(1000);

}
```

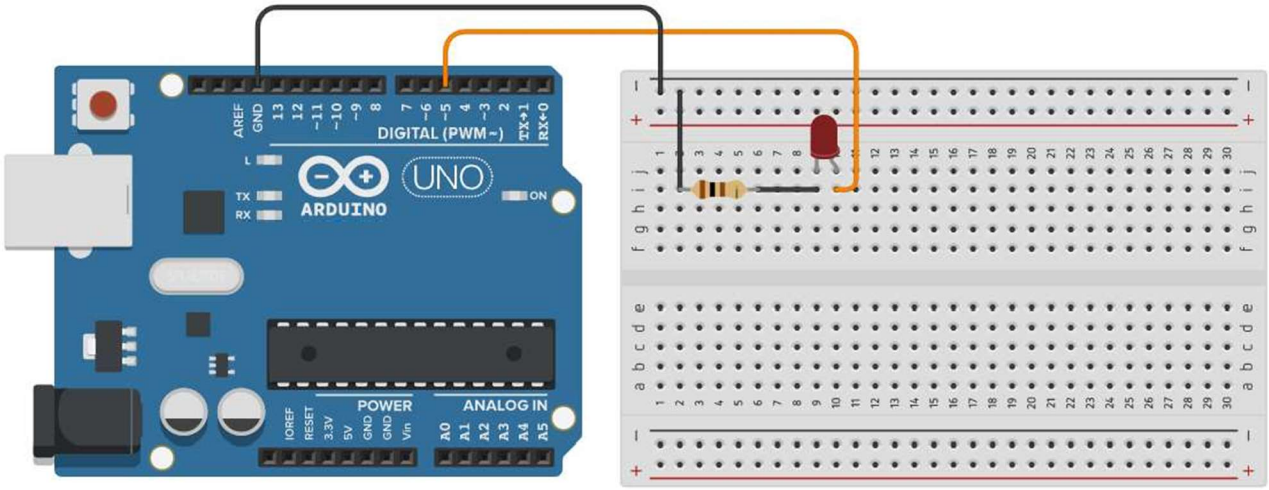
Padrão 2: caso nosso código do loop seja igual a este, o padrão de funcionamento dos leds será o seguinte: o primeiro led (da porta 5) liga, após um segundo o segundo led (da porta 6) também liga e após mais um segundo o terceiro led (da porta 7) também liga. Após mais um segundo, todos os leds desligam ao mesmo tempo. Assim, temos um acendimento sequencial, da esquerda para a direita, permanecendo todos os leds ligados até o momento em que todos desligam juntos.

```
void loop(){

  digitalWrite(7, HIGH);
  delay(1000);
  digitalWrite(6, HIGH);
  delay(1000);
  digitalWrite(5, HIGH);
  delay(1000);

  digitalWrite(5, LOW);
  delay(1000);
  digitalWrite(6, LOW);
  delay(1000);
  digitalWrite(7, LOW);
  delay(1000);

}
```

Padrão 3: caso nosso código do loop seja igual a este, o padrão de funcionamento dos leds será o seguinte: o terceiro led (da porta 7) liga, após um segundo o segundo led (da porta 6) também liga e após mais um segundo o primeiro led (da porta 5) também liga. Após mais um segundo, o último led a ser ligado (da porta 5) desligada, seguido do penúltimo led a ser ligado (da porta 6) após mais um segundo e finalmente seguidos do primeiro led a ser ligado (da porta 7). Assim, temos um acendimento sequencial, da direita para a esquerda, seguido de um desligamento sequencial, da esquerda para a direita.

Exemplo 2: ligando o led e regulando a intensidade com PWM

Neste segundo exemplo, nosso objetivo é realizar o controle de apenas um led com o Arduino, mas não somente o ligando e o desligando, mas controlando a sua luminosidade através da modulação da tensão na saída (variando o Duty Cycle). Para isso, vamos ligar o led (através do pino positivo) em uma porta digital PWM, indicadas com um sinal de til (~) ao lado do número (no nosso exemplo, na porta 5). O outro pino deverá ser ligado ao negativo (no Arduino, a porta GND) passando por um resistor (no nosso caso, de 100Ω) para baixar a corrente e evitar a sua queima. Assim, a montagem do circuito deverá ficar como a abaixo:



Primeiramente, devemos configurar a porta 5 como uma porta de saída, afinal, queremos através dela gerar uma tensão que alimentará o led. Também devemos garantir que quando a placa for iniciada, seu estado inicial seja baixo. Não há diferença nos métodos utilizados para isso, em relação ao exemplo anterior. Assim, em nosso setup, teremos o seguinte código:

```
void setup(){

  pinMode(5, OUTPUT);
  digitalWrite(5, LOW);

}
```

Agora, dentro do loop, vamos fazer com que a cada 1 segundo a intensidade do led aumente até que, ao atingir seu valor máximo, apague novamente. Para isso devemos utilizar o método analogWrite e escrever na porta um valor entre 0 e 255. Quando mais perto de 0, mais próximo de 0V. Quanto mais perto de 255, mais próximo de 5V. No exemplo, iremos

aumentar essa intensidade variando o valor que controla o Duty Cycle de 50 em 50. Deste modo, o led inicia apagado e a cada segundo aumenta um pouco a sua intensidade por conta da maior tensão recebida e, consequentemente, da maior corrente gerada que o atravessa. Caso desejado, conseguimos calcular a tensão correspondente ao valor utilizado no analogWrite aplicando uma regra de 3. Exemplo:

```
void loop(){

  analogWrite(5, 50);
  delay(1000);

  analogWrite(5, 100);
  delay(1000);

  analogWrite(5, 150);
  delay(1000);

  analogWrite(5, 200);
  delay(1000);

  analogWrite(5, 255);
  delay(1000);

  analogWrite(5, 0);
  delay(1000);

}
```

Se 255 é igual a 5V, 50 é igual à quanto?

$$255 = 5$$
$$50 = x$$

Assim temos que: $x = (5 \cdot 50) / 255 \cong \mathbf{1\ V}$

De forma análoga, descobrimos que neste nosso exemplo, um valor de 100 no analogWrite para controle do Duty Cycle, produz uma tensão de saída próxima à 2V, assim como um valor de 150 produz uma tensão de saída próxima à 3V, um valor de 200 produz uma tensão de saída próxima à 4V e um valor de 250 produz uma tensão de saída próxima à 5V. Lógico que caso a intensão seja descobrir esses valores posteriormente à construção do circuito, o uso de um multímetro facilitaria e tornaria desnecessários os cálculos. Mas quando pensamos no dimensionamento prévio, calcular estes mapeamentos é importante.