Timers e millis()

Timers

Os microcontroladores contam com circuitos chamados ***timers*** integrados no seu CI. Um *timer* nada mais é do que um circuito com um registrador e um incremento que pode ser disparado internamente (ligado ao clock) ou vindo de um pino externo (interessante para fazer contagens com um hardware adequado). Uma vez associado ao clock, é capaz de contar tempo se devidamente configurado. O Arduino com ATmega328 tem 3 *timers*.

delay()

Já usamos a função delay() e vimos que ela serve como forma de temporizar eventos. Essa temporização se faz chamando a função *delay(numero)* e ela demora *numero* milissegundos para retornar. Isto pode ser inconveniente em algumas situações pois caso precisemos continuar lendo sensores/chaves ou fazendo qualquer outro tipo de processamento, teremos que esperar o *delay()* retornar. Isso é mais ou menos equivalente a "paralisar" a CPU. Podemos trabalhar com os *timers* do Arduino de forma direta, manipulando seus registradores ou de forma indireta através da função *millis()*. A forma direta é bem mais trabalhosa e é preciso fazer vários cálculos para se obter o resultado da temporização. Usando a função *millis()*, podemos resolver nosso problema inicial que é não deixar a CPU paralisada enquanto trabalhamos com a temporização de eventos.

millis()

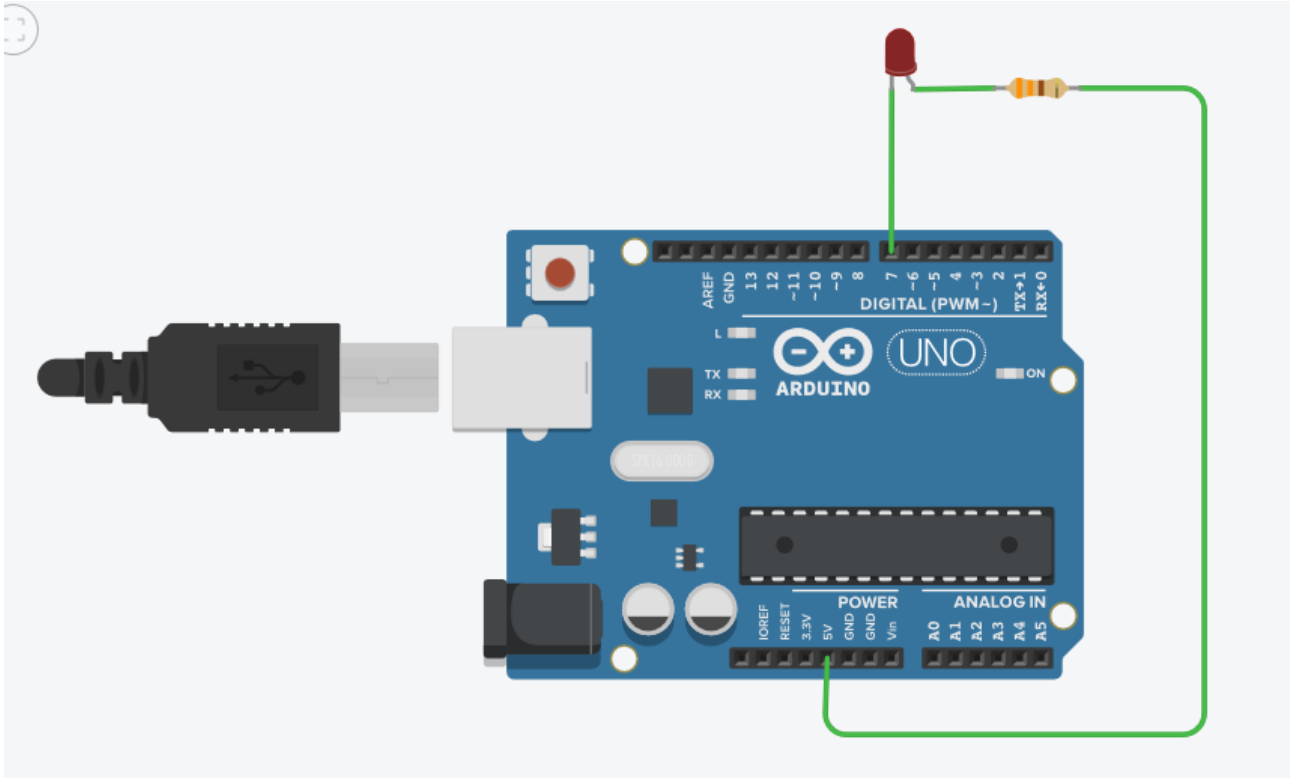
Esta função retorna o tempo em milissegundos desde que o programa começou a ser executado. Este número pode ser bem grande e deve ser atribuído a uma variável *long int*.

Dentro da função *loop()* podemos fazer o teste se o tempo decorrido foi maior que o intervalo que determinamos para a mudança de um evento sem prender a CPU. Toda vez que o teste se repetir já que a função *loop()* é chamada novamente quando ela termina, a função *millis()* retornará um valor maior pois mais tempo terá decorrido, o que em algum momento pode ter chegado naquele intervalo de tempo que a temporização precisa.

Como exemplo, faremos um programa para fazer um LED piscar sem *delay()*. LED deve ser ligado ao pino 7 através de um resistor de 330Ω.

Exemplo 1:

Abaixo temos o circuito:



LED ligado ao Arduino – acende com 1

Programa:

```
#define LED 7

#define INTERVALO 2000 // intervalo de tempo para o LED
                       // piscar
long int millis_atual, millis_anterior = 0;
// millis_anterior é zerado porque queremos estabelecer
// o início do programa como tempo inicial para a
// temporização do LED

byte led_st = LOW;
// o estado do led deve ser armazenado em uma variável

void setup() {
  pinMode(LED, OUTPUT);
}

void loop() {
  millis_atual = millis();
  if ((millis_atual - millis_anterior)>= INTERVALO){
    // so entra aqui se o período de tempo esperado chegou
    if (led_st) led_st = LOW;
    else led_st = HIGH; // troca o estado do LED
    millis_anterior = millis_atual;
    digitalWrite (LED, led_st);
  }
}
```

No início do programa temos a diretiva para definir LED como 7 e INTERVALO como 2000. Este é o intervalo de tempo em milissegundos que usamos para fazer o LED mudar de estado. É equivalente ao argumento de um *delay()* no programa teste *blink*. A variável millis_atual será usada para atribuir o tempo que se passou desde o início do programa até aquele ponto através da função millis(). A variável millis_anterior será usada com o tempo decorrido desde o início do programa ate a última alteração de estado do LED (portanto será menor do que millis_atual) e começa com 0 pois queremos que o LED mude de estado pela primeira vez INTERVALO milissegundos. A variável led_st é o estado do LED. Neste tipo de algoritmo precisamos armazenar o estado do LED para que no momento certo possamos alterá-lo.

Na função setup() temos apenas o sentido da porta do LED que é de saída.

Na função loop(), começamos a atribuir à variável millis_atual o valor de millis() que dá o tempo decorrido desde o início da execução do programa. A seguir, temos:

```
if ((millis_atual - millis_anterior)>= INTERVALO)
```

Para analisar este teste, vamos supor que o millis_anterior está com o valor inicial que é 0. Esta condição diz que se o tempo decorrido desde o início do programa menos o millis_anterior que é 0 for maior ou igual ao intervalo ele deve entrar no if. Digamos que não houve tempo suficiente para que o millis_atual seja >= INTERVALO: a execução não entra no if e a função loop() termina. Como sabemos que a função loop() será chamada novamente ao fim da sua execução, teremos um novo millis_atual (pois se passou um certo tempo para que a função loop fosse chamada novamente) teremos um outro teste e caso este teste seja falso, teremos essa situação se repetindo.

Imagine agora que já se passou tempo suficiente para que millis_atual seja >= INTERVALO (lembrando que no início millis_anterior é zero). A execução neste caso entrará no if e teremos a execução do if-else seguinte que faz apenas a troca de estado da variável led_st (verifique como funciona este if-else). A próxima instrução:

```
millis_anterior = millis_atual;
```

Faz com que o ponto de partida para os novos testes seja o tempo em que decorreu a mudança de estado do LED. Assim agora, millis_anterior terá um valo de INTERVALO ou próximo dele e agora a próxima mudança de estado será próxima de 2xINTERVALO. O último comando diz para colocar no pino do LED o novo estado.