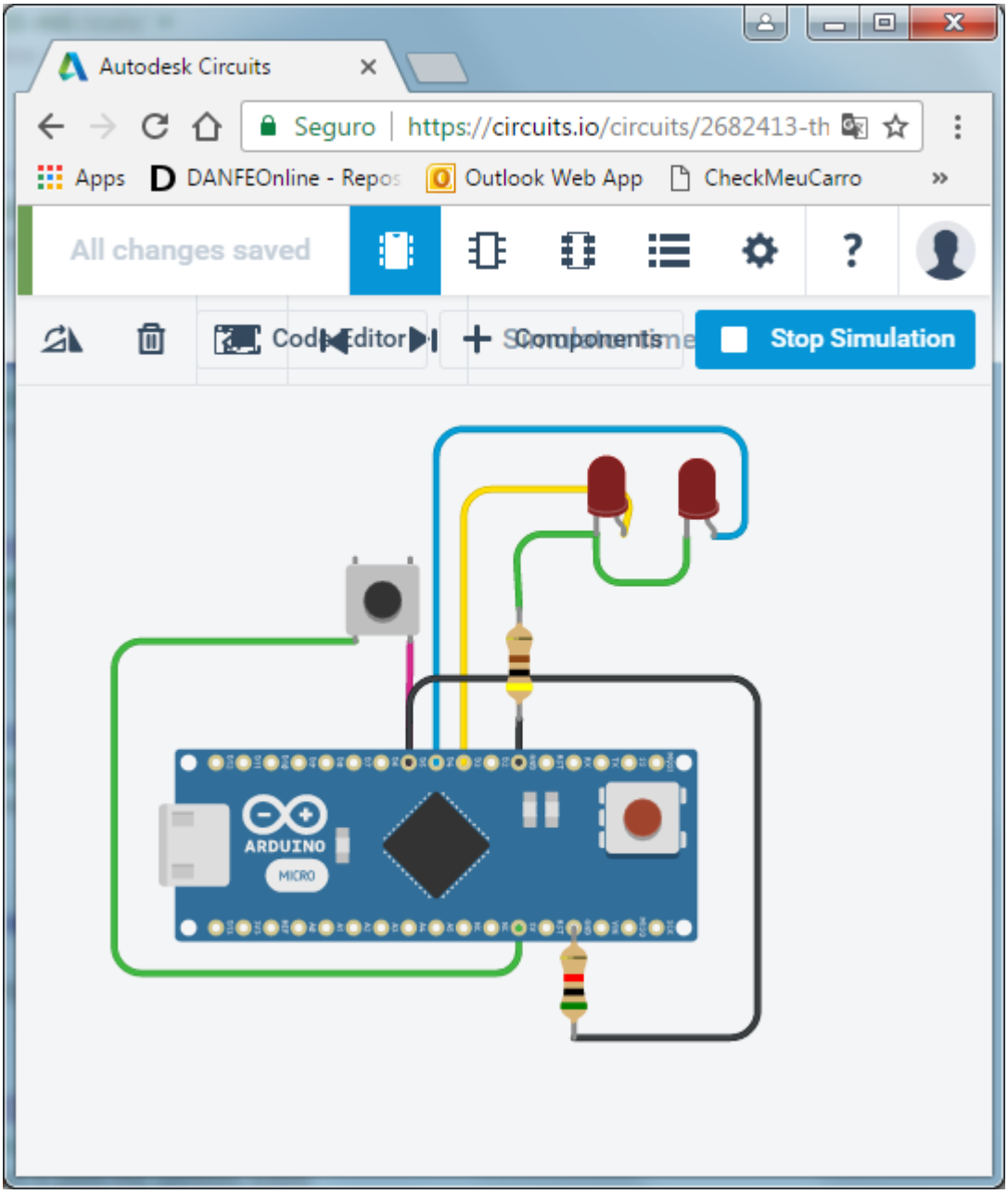
Este codigo foi criado por mim para ser usado em uma automação de duas bombas elevadoras de agua de cisterna para caixa alta, onde operam sempre uma ou outra, nunca simultaneamente, no caso, a cada chamada de solicitação de agua pela boia, sera acionada uma bomba, desta forma elas se revezam e desta forma ambas tem o mesmo desgaste.

LIGAÇÕES




O Objetivo do circuito acima é ligar um led de cada vez, a cada vez que o botão for pressionado. Enquanto o botão se mantém pressionado, um dos leds se mantém ligado. Quando o botão for solto os dois Leds permanecem desligados. Da próxima vez que o botão foi ligado, o outro led se mantém ligado, até o botão for solto. E assim vai alternando qual led é acionado a cada vez.

Os dois leds compartilham o mesmo resistor, pois nesse circuito, nunca os dois leds estarão ligados ao mesmo tempo.

O Botão foi ligado com um resistor de Pulldown.

CÓDIGO

```
byte b     = 5;
byte led1  = 4;
byte led2  = 3;

void setup() {
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(b, INPUT);
}

void loop() {
  static boolean lastValue = false;
  static unsigned long lastDebounceTime = 0;
  static byte lastButtonState = LOW;

  static byte iLed = 0;
  byte r = digitalRead(b);


  if (r != lastButtonState) { lastDebounceTime = millis(); }

  if ((millis() - lastDebounceTime) > 80) {  //80ms time debouncing

    if (r && r != lastValue){    //borda de subida
      lastValue = true;

      iLed++;
      if (iLed > 1) {iLed = 0;}

      if (iLed == 0){
        digitalWrite(led1, HIGH);
      } else {
        digitalWrite(led2, HIGH);
      }

    }

    if (!r && r != lastValue){    //borda de descida
      lastValue = false;

      if (iLed == 0){
        digitalWrite(led1, LOW);
      } else {
        digitalWrite(led2, LOW);
      }
    }
  }
  lastButtonState = r;

  delay(20);
}
```

EXPLICAÇÕES

```
byte b     = 5;
byte led1  = 4;
byte led2  = 3;

void setup() {
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(b, INPUT);
}
```

Primeiramente declaramos os pinos utilizados, pino do botão, pino do led1 e pino do led2 e dentro do loop definimos os pinos dos leds como saída e o botão como entrada.

```
static boolean lastValue = false;
static unsigned long lastDebounceTime = 0;
static byte lastButtonState = LOW;

static byte iLed = 0;
byte r = digitalRead(b);
```

Das 5 variáveis utilizadas pela dentro do laço Loop, 4 delas são static. Quando uma variável é static, significa que ela mantém seu valor entre uma execução e outra do código. Isso significa que o valor de inicialização da variável só recebe aquele valor na primeira vez que ela é executada. Na próxima vez que o programa executar aquela linha do programa, o variável não terá o valor inicializado novamente, mas sim o último valor assumido da vez anterior que recebeu. Na prática, uma variável static tem o comportamento bastante parecido com uma variável global, já que ela mantém seu valor entre uma execução e outra do mesmo trecho de código, coisa que não acontece em uma variável normal. A única diferença, é que uma variável static só é visível dentro do escopo onde foi declarada. Então se uma variável precisa manter o seu valor mesmo depois de terminar a execução do código e ela não precisa ser acessada em diferentes partes do programa, é uma boa ideia definir a variável como static. Porém deve-se ficar atento, pois uma variável static, aloca a memória durante toda a execução do programa, não liberando aquele espaço até o fim da execução do programa.

A cada iteração da função loop, verificamos o estado do botão, se pressionado ou solto. Porém, por uma questão de evitar ruídos na leitura do botão, incluímos uma parte no código responsável por verificar se o botão ficou pressionado ou solto, por pelo menos 80 ms. Essa é a parte de debouncing via Software. Veja que no primeiro if, quando houver mudança de estado da leitura do botão, a variável `lastDebounceTime` assume o valor atual de milis. Se ficar mais de 80ms Entra então no próximo if, garantindo que ruidos não atrapalhem.

```
byte r = digitalRead(b);  //le o estado do botão
if (r != lastButtonState) { lastDebounceTime = millis(); }
if ((millis() - lastDebounceTime) > 80) {  //80ms time debouncing
```

Dentro desse segundo if, temos duas verificações. Se o botão está em uma borda de subida do sinal, ou em uma borda de descida do sinal. Borda, significa aquele momento exato da mudança de estado (mas no nosso caso, considerada apenas após o tratamento de debouncing), ou seja, apertando ou soltando o botão. Para saber isso, precisamos saber qual era o estado anterior da leitura e para isso utilizamos a variável `lastValue` (que é static e consegue guardar seu último valor) conseguimos saber se ela se modificou em relação ao seu último estado.

```
if (r && r != lastValue){     //borda de subida

e

if (!r && r != lastValue){     //borda de descida
```

Em cada um dos eventos (subida ou descida) temos um comportamento diferente do que será feito. Na borda de subida (ao pressionar o botão) iremos ligar um dos dois leds, já na borda de descida desligamos os dois.

Como controlamos um led de cada vez, é na borda de subida que invertemos qual led será controlado, e isso é feito através da variável `iLed:`

```
iLed++;
if (iLed > 1) {iLed = 0;}
```

A cada execução incrementamos iLed, informando que mudamos o led a ser controlado.

E por fim, atualizamos atualizamos o valor de `lastButtonState;` atribunindo a ela, o valor da leitura atual do botão. Pois é através dela, na próxima execução que saberemos se houve mudança de estado e em seguida damos um delay no código. Esse delay só é necessário no simulador, diretamente no arduino pode ser dispensado.

```
  delay(20);
```

Eng.Carlos kwiek