UNIDADE 2: INPUTS/OUTPUTS DIGITAIS E SUSPENSÕES

OBJETIVOS

Nesta unidade não vamos explorar o que são *inputs* nem *outputs* digitais. Assumimos que já os utilizaram e que estão familiarizados com eles. Vamos, sim, clarificar algumas ideias sobre o controlo de equipamentos digitais e explicar o que são resistências *pull-up* e *pull-down*.

Vamos também observar de perto as suspensões. O Arduino é capaz de suspender um programa que esteja em processo e fazer correr outro, ou desempenhar uma outra tarefa. Assim que esta tarefa tenha sido executada, o Arduino volta ao programa original. Tudo isto pode acontecer quando os pins de *input* detetam sinais específicos.

SECÇÃO TEÓRICA

- OUTPUTS DIGITAIS E NÍVEIS LÓGICOS
- INPUTS DIGITAIS; RESISTÊNCIAS *PULL-UP* E *PULL-DOW*
- SUSPENSÕES

SECÇÃO PRÁTICA

- EXEMPLO 1: Iluminação de LEDs
- EXEMPLO 2: Monitorização de *inputs*
- EXEMPLO 3: Monitorização de *inputs* com resistências *pull-up*
- EXEMPLO 4: Monitorização de *inputs* sem suspensões
- EXEMPLO 5: Monitorização de *inputs* com suspensões
- EXEMPLO 6: Controle de duas suspensões

MATERIAL

-Computador portátil ou de secretária

-Ambiente Arduino IDE; deve incluir o material suplementar já instalado e configurado.

-Placa de controlo Arduino UNO

-Cabo USB

ÍNDICE

SECÇÃO TEÓRICA

1. OUTPUTS DIGITAIS E NÍVEIS LÓGICOS

Pode soar um pouco estranho, mas não é realmente muito difícil compreender o que vai ser explorado nesta secção. Assumimos que já está familiarizado/a com as funções relacionadas com a operação de *output* digital. Vamos revê-las:

- **pinMode(n,OUTPUT)**: permite configurar qualquer um dos pins D0:D13 como outlets.
- **digitalWrite(n,LOW)**: envia um nível lógico de "0" de 0 V através do pin *n* indicado.
- **digitalWrite(n,HIGH)**: envia um nível lógico de "1" de +5 V através do pin indicado.

Agora, vamos imaginar que quer controlar a iluminação de um LED que está ligado ao pin do output digital D4. De certeza que irá escrever uma sequência de funções como esta:

```
pinMode(4,OUTPUT);        //Output pin D4

digitalWrite(4,HIGH);     //enviar nível "1" através do pin D4
```

Porque falamos sempre de estabelecer um nível "1" (+5 V) cada vez que se pretende ligar algo? Utilizamos apenas o nível "0" (0 V) para desligar? Será possível fazer de forma inversa? Sim, é possível! Ambos os níveis, "1" e "0", são igualmente representativos. Observe os diagramas na Figura 1. Que tipo de nível lógico teríamos que enviar através do sinal D4 (pin 7) para ligar o LED no circuito da esquerda? E o LED da direita?

Figura 1

Não se esquecendo que o anodo tem de ser positivo comparativamente ao catodo para acender um LED, tem que ser enviado um "1" (+5V) através do pin 7 para o circuito da esquerda. Isto deve-se ao facto de o catodo estar ligado ao GND (0 V) através da resistência de absorção. No entanto, temos de enviar um "0" (0 V) através do D4 para o catodo no circuito da direita, uma vez que o anodo se liga ao +5 V através da resistência. Percebeu tudo?

Estamos a referir-nos a um LED, mas o mesmo aplica-se a outros aparelhos como bobinas de retransmissão, motores, sirenes, etc.. Observe os diagramas na Figura 2. Há apenas um aspeto que gostaríamos de salientar: pode reparar que a rotação direcional do motor muda quando o liga com o nível "1", em vez do nível "0".

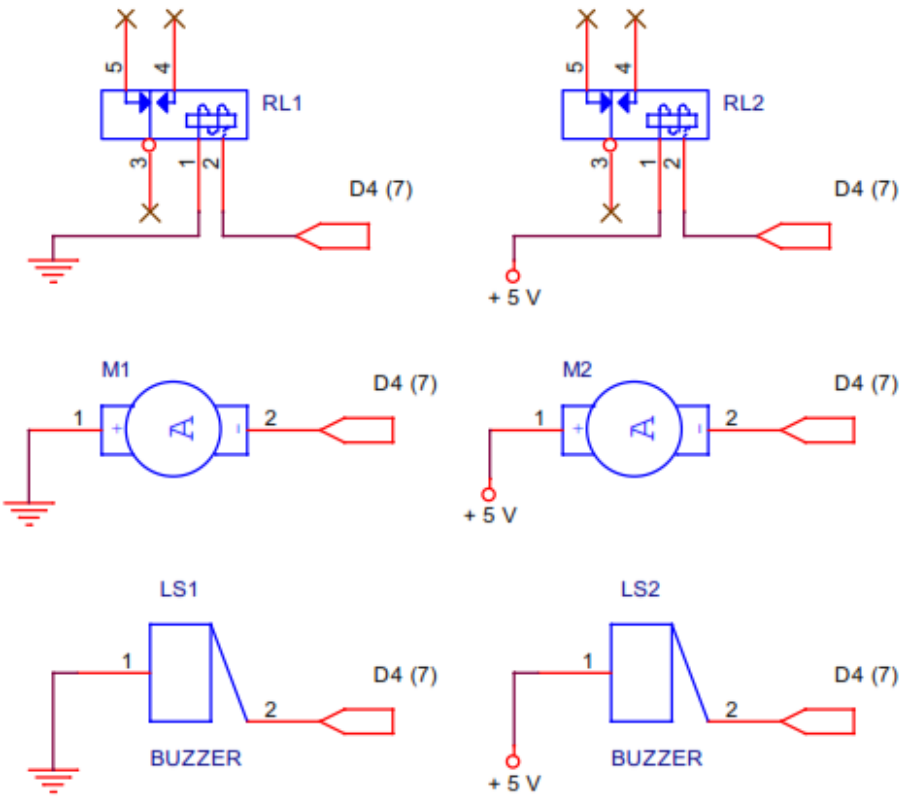

Figura 2

Em suma, existem periféricos de *output* que se ligam quando lhes é aplicado o "1" – chamam-se "sinais ativos altos". Existem também periféricos de output que se ligam quando lhes é aplicado o nível "0". Vai encontrar os dois tipos. Depende basicamente da forma como o periférico é montado e ligado.

2. INPUTS DIGITAIS; RESISTÊNCIAS PULL-UP E PULL-DOWN

Embora possa dominar tudo sobre *inputs* digitais, há sempre aspetos a explora. Vamos analisar um problema com que provavelmente já se deparou, mas sem saber porquê. Para começar, recorde-se que os periféricos de *input* digitais mais básicos e económicos são botões simples de ligação e interruptores, como os apresentados na Figura 3. Encontra todos os tipos de formas e tamanhos. Alguns são desenhados para utilização industrial em máquinas, painéis de controlo, etc..

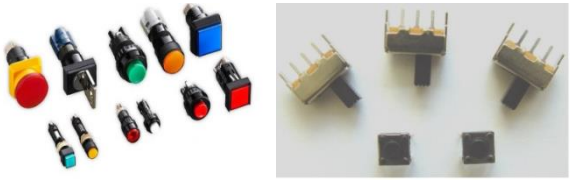

Figura 3

Existem outros muito mais simples (e mais baratos); são amplamente utilizados em casa, em instituições educativas, para ligar objetos em cartões impressos, em pequenos eletrodomésticos, etc.. Vamos utilizar pequenos botões como os dois que pode observar na parte inferior da imagem (à esquerda), na Figura 3.

A forma como funcionam é extremamente simples, por isso, não é relevante para que são utilizados. Quando premidos, uma folha de metal abre e fecha um contacto elétrico entre dois ou mais pins no aparelho. Dizemos que um interruptor ou comutador está "embebido". Quando é ligado, o interruptor mantém-se na posição em que o deixou até o mudar novamente. Os interruptores de iluminação são assim, mas os botões de ligação não: abrem e fecham um ou mais circuitos apenas quando ligados. Depois disso, voltam à sua posição quando deixam de ser premidos – tal como uma campainha de porta de uma casa. Pode observar um diagrama de um comutador simples e de um botão nas imagens da Figura 4. Também pode ver os respetivos símbolos elétricos.

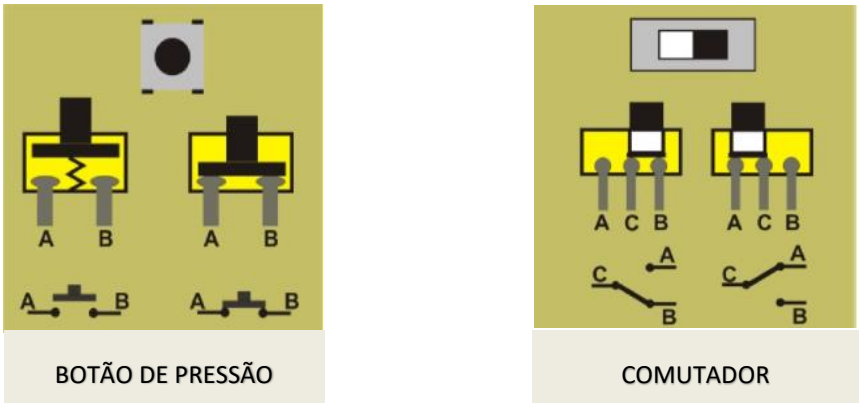

Figura 4

O comutador na imagem tem três pins; um deles é comum aos outros dois pins. Quando o deslizamos para a direita, a folha fecha o circuito entre os pins C e B. O circuito C-A mantém-se aberto, mas não está ligado. Se o deslizarmos para a esquerda, o circuito C-A fecha-se e o C-B mantém-se aberto.

O botão tem dois pins, A e B. Dizemos que, normalmente, na posição de descanso está aberto. Os pins não estão ligados. Quando premido, o circuito fecha-se e ligam-se ambos os pins. Quando libertado, um salto pressiona a folha de metal que retorna à sua posição de descanso. Existem ainda botões de ligação que normalmente se fecham e que só abrem quando premidos (Figura 4).

De certo que já está familiarizado/a com ambas as funções da linguagem Arduino que controlam os inputs digitais:

- **pinMode(n,INPUT)**: permite configurar qualquer um dos pins digitais D0:D13 como inputs. Na verdade, não tem de fazer nada: sempre que o ligar ou premir RESET, sempre que reinicia o sistema, todos os pins passam a ser *input*.
- **digitalRead(n)**: lê o nível lógico de qualquer pin *n* que preferir.

No entanto, reflita sobre os circuitos nos diagramas da Figura 5. Um botão foi ligado a um D2 (pin 5) que, aparentemente, está configurado como *input*.

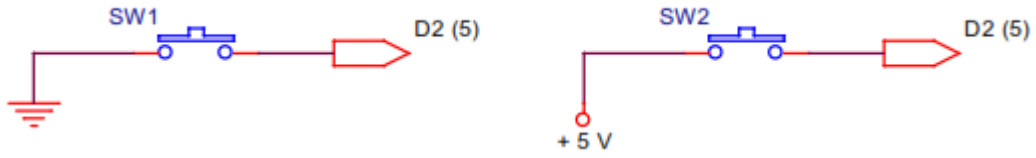

Figura 5

Sempre que premir o botão no circuito da esquerda, o pin D2 liga-se com o GND (0 V), estando assim no nível "0". O da direita funciona ao contrário. Ao premir o botão no circuito da direita, liga-o com o +5 V e, por isso, fica no nível "1".

E agora, a questão essencial: o que acontece aos pins se não premirmos nenhum dos botões nos dois circuitos? Alguns de vocês irão de certo dizer "apenas o oposto do que acontece quando são premidos." Por outras palavras, se não premirmos os botões no circuito da esquerda, o pin D2 permanece no nível "1" e o da direita no nível "0".

Errado! Vamos ver: se o botão não for premido em nenhum dos circuitos, qual é a diferença no pin D2? Absolutamente nenhuma. O D2 mantém-se desligado em ambos os casos. Dizemos que está a "flutuar". Qual vence: o nível "1" ou o nível "0"? Não existe uma resposta definitiva; às vezes o "1" prevalece, outras vezes é o "0". Como se percebe, esta não é uma situação ideal.

No que diz respeito aos circuitos, o melhor a fazer é adicionar uma resistência. Se estiver ligada ao +5 V positivo, chama-se resistência "*PULL-UP*"; se estiver ligada ao GND ou ao 0 V, chama-se "*PULL-DOWN*". Observe os dois circuitos abaixo (Figura 5).

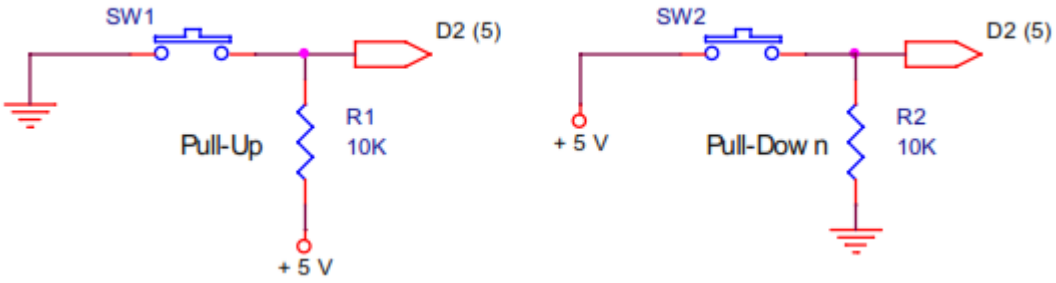

Figura 5

No circuito da Figura 6, à esquerda, a resistência pull-up R1 mantém o sinal D2 (pin 5) no nível "1" (+5 V) quando o botão SW1 não é premido, ou, por outras palavras, está aberto. Quando o pin D2 é premido, passa a nível "0". No circuito da direita, a resistência pull-down R2 detém o pin no nível "0" (GND) quando o botão SW2 não é premido. Quando se prime o pin D2, passa a nível "1".

Mas então que valor devemos atribuir às resistências? Este aspeto não é muito relevante. Lembre-se que cada vez que premir o botão, há uma corrente de eletrões entre o GND e o +5 V, ou que, por outras palavras, a corrente (I) atravessa o circuito. Devemos tentar minimizar esta corrente ou consumo. Por exemplo, se colocar 100 Ω através do R1 ou do R2, o consumo será o seguinte:

$$I = \frac{V}{R} = \frac{5}{100} = 0.05\ A = 50\ mA$$

Se escolher um valor de 10 KΩ, o consumo será este:

$$I = \frac{V}{R} = \frac{5}{10000} = 0.0005\ A = 0,5\ mA$$

Tenha cuidado, não deve sobrecarregá-lo. Pode sentir-se tentado/a a utilizar uma resistência potente para reduzir o consumo ao mínimo. A consequência de utilizar um valor assim alto é isolar o pin D2 de +5 V (no caso da *pull-up*) ou de GND (no caso da *pull-down*) e não permanece. Se esta situação ocorrer, retrocede ao ponto onde iniciou, por isso, é sempre melhor não utilizar nenhuma resistência. Os valores comuns encontram-se entre os 4700 e os 10 KΩ.

Mais uma vez, sublinhamos que o nível ativo – quando premimos o botão – não tem necessariamente de ser "1"; também poderia ser "0". Tudo depende da forma como esse botão for ligado.

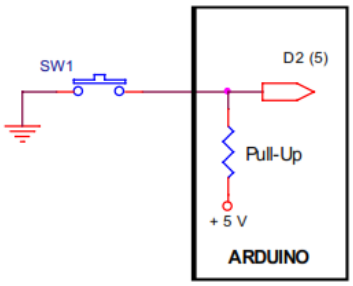

Figura 6

Como é que o Arduino nos ajuda? Fornece a resistência pull-up e isto previne que tenha de instalar uma nos seus circuitos externamente. Repare na Figura 7: a resistência foi integrada no controlador Arduino.

Pode configurar esta opção utilizando a seguinte função:

- **pinMode (n, INPUT_PULLUP)**: quando *n* representa o pin de *input*, significa que pretende ligá-lo à resistência *pull-up* correspondente.

Pode fazer isto com qualquer um dos pins D0:D13 a configurar como *input*. Não tem sentido aplicar este aspeto aos *outputs* e também não tem hipótese de configurar um *input* como resistência *pull-down*.

Talvez considere importante poupar uma resistência *input*. Custam apenas alguns cêntimos de euro. No entanto, de um ponto de vista de negócio, se o seu projeto precisar de um número razoável de resistências *pull-up*, não vai apenas poupar nas resistências, mas também:

- Poupa no espaço, porque não terá de as armazenar nem de as ter em *stock*.
- Poupa no tamanho das impressões das placas de circuitos. Vendem-se por decímetro quadrado e, quanto maior for o número de componentes, maior a placa, por isso, mais terá de pagar.
- Poupa no tempo despendido a desenhar as faixas para as placas de circuitos a imprimir: quanto mais componentes utilizar, mais tempo será necessário para os desenhar. Tempo é dinheiro.
- Poupa tempo na montagem da placa de circuitos. Quanto maior o número de componentes, mais tempo precisará para montar a placa.

Se multiplicar tudo isto pelo número de placas de circuitos ou kits que irá vender, poderá corresponder a uma poupança de centenas ou milhares de euros.

Agora que já sabe como, utilize a função *pull-up*, sempre que considerar necessário.

3. SUSPENSÕES

Vamos analisar outro aspeto: as suspensões. Talvez associe este o termo “suspensões” a parar o controlador, colocá-lo em pausa para que pare de executar o programa. Na verdade, não tem nada a ver com isso.

Imagine uma ferramenta de uma máquina que esteja a produzir peças. De repente, o alarme dispara devido a uma das situações seguintes:

- Uma peça mal posicionada que dá origem a um produto com defeito.
- O motor que dirige a correia de transporte parte, provocando o acumular das peças na linha de montagem.
- Um sensor deteta que um componente da máquina está a sobreaquecer.
- Um trabalhador coloca a mão numa parte indevida da máquina.

O que deve fazer o controlador para parar a anomalia?

a) Nada em especial; continuar a executar o programa normalmente como se nada fosse.

b) Interromper e parar de executar o programa imediatamente.

c) Parar de executar o programa normal e iniciar outro que afira as circunstâncias que motivaram o disparar do alarme, emitindo avisos e executando as tarefas necessárias: remover a peça com defeito, parar a confeção das peças, ligar o sistema de refrigeração, emitir um sinal de aviso ao trabalhador, etc..

A resposta é bastante óbvia, não lhe parece? Uma suspensão não significa necessariamente que o controlador pare por completo; pára a tarefa que estava a executar no momento e inicia outra.

Como provocamos uma interrupção/suspensão? Existem imensas situações que podem despoletar uma suspensão. Depende do modelo de controlador. O mais habitual é os periféricos externos enviarem sinais através de pins do controlador. No caso que consideramos, o Arduino UNO tem duas fontes de suspensão, ou pins, diferentes: D2/INT0 e D3/INT1.

Temos a certeza de que se recorda de ter utilizado estes pins em inúmeras ocasiões e de que nunca causaram qualquer suspensão. São apenas pins de input/output digitais como todos os de D0:D13. É verdade. Acontece que, para os pins D2 e D3 trabalharem como inputs de suspensão INT0 e INT1, têm de ser configurados de forma correta e equipados com uma autorização especial. Provocar uma suspensão não é apenas um detalhe. É exatamente sobre estes aspetos que versa este capítulo.

O que acontece quando um controlador recebe uma suspensão? Observe a Figura 8 atentamente, onde se apresenta como funciona o processo:

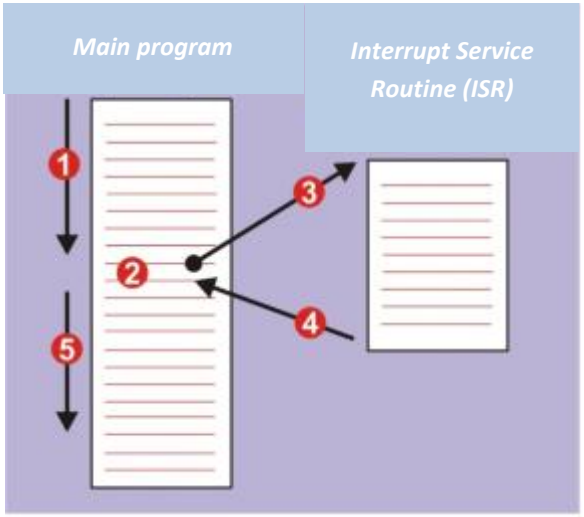

Figura 7

1. O controlador está a executar o seu habitual programa principal.

2. Em determinado momento, um periférico solicita e provoca uma suspensão.

3. O controlador suspende o programa principal e passa a executar um programa que lida com o evento, também conhecido por ISR (*Interrupt Service Routine*).

4. Assim que a execução que lida com a suspensão é concluída, o controlador regressa ao programa principal.

5. A execução retoma a partir do ponto em que foi interrompida.

Para gerir e utilizar as suspensões do Arduino UNO tem quatro funções à sua disposição:

- **Função attachInterrupt()**

  Configura a forma como uma suspensão deve funcionar.

Sintaxe:

attachInterrupt(pin, ISR, mode);

pin: Representa o pin que irá ser configurado como input de suspensão. No caso do Arduino UNO pode ser INT0 (D2) ou também INT1 (D3).

ISR: Identificação da rotina ou função que tem de ser executada cada vez que ocorre uma interrupção/suspensão.

mode: Refere-se ao momento em que uma suspensão deve ser despoletada:

LOW: quando o pin envia através de um nível “0”.

CHANGE: quando é detetada uma mudança de estado no pin.

RISING: quando o pin deteta uma vantagem crescente (“0” -> “1”).

FALLING: quando o pin deteta uma vantagem decrescente (“1”-> “0”).

- **Função detachInterrupt()**

  Desliga uma suspensão.

Sintaxe:

detachInterrupt(pin);

pin: representa o pin de suspensão a ser desligado. No caso do Arduino UNO pode ser INT0 (D2) ou INT1 (D3).

- **Função interrupts()**

Liga as suspensões. Considere-a como uma espécie de autorização geral para as suspensões previamente configuradas através de attachInterrupt().

Sintaxe:

interrupts();

- **Function noInterrupts()**

Desliga todas as suspensões. Considere-a como uma espécie de proibição geral que previne o funcionamento de todas as suspensões.

Syntax:

noInterrupts();

A Figura 9 dá-lhe uma ideia da apresentação de um programa que utiliza suspensões.

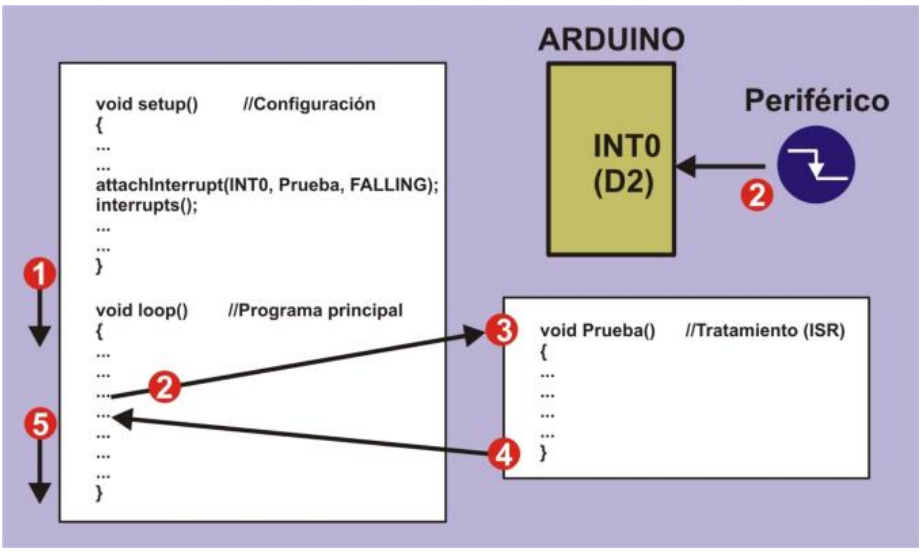

Figura 8

Cada vez que uma vantagem decrescente é detetada, o pin D2 (INT0) configura a suspensão na função de configuração **setup()** utilizando **attachInterrupt()**. Se tal acontece, a função **Test()** que corresponde à ISR é executada. Quase em simultâneo, a função **interrupts()** emite a autorização geral. Aqui está o que acontece:

1. Começa por iniciar o programa **loop()** principal.

2. Num determinado momento, um periférico emite um pedido de suspensão com uma vantagem decrescente através do pin INT0 (D2) pin. O controlador pára de executar o programa.

3. O controlador executa a função **Test()** (*Interrupt Service Routine*).

4. Assim que a função **Test()** tenha sido executada, o controlador regressa ao programa principal.

5. O controlador renova a execução do programa do ponto onde havia parado.

Limitações

Devido à natureza do Arduino, o seu sistema de suspensão é um pouco limitado. Um exemplo é o facto de as bem conhecidas funções delay() e millis() não funcionarem se já estiverem inseridas na rotina do serviço de suspensões (ISR). Isto acontece porque estas funções utilizam o seu próprio sistema de suspensão interno e o Arduino não consegue responder a uma suspensão oriunda de outro sistema.

Não é possível transferir parâmetros para uma rotina de serviço de suspensão (ISR) e o ISR também não consegue devolver nada. Se realmente precisar, terá de utilizar variáveis globais para transferir os dados a serem utilizados pelo programa principal e pela função de rotina do serviço de suspensão.

Lembre-se que uma suspensão é um evento que pode ou não acontecer de tempos a tempos. Estas situações de alarme podem ou não ocorrer. Um periférico pode ou não enviar um sinal de suspensão. É boa ideia executar uma função de ISR o mais rapidamente possível. Lembre-se que durante a execução de uma função ISR, o controlador suspende a execução do programa principal.

Vantagens

Também há vantagens. Geralmente, um programa que utiliza suspensões é muito mais eficiente. Tem, no entanto, de ter cuidado relativamente à forma como o utiliza. Imagine que está a fazer um projeto e que tem de levar a cabo uma determinada tarefa cada vez que um periférico envia um sinal.

Tem duas opções: uma via a função digitalRead(), que lê e espera que o sinal seja despoletado. Lembre-se que, enquanto está à espera do sinal, não pode fazer mais nada.

A outra solução – a melhor – é utilizar uma suspensão. O programa principal pode estar a executar qualquer tarefa, mas quando o sinal o alcança, aí, e apenas aí, o controlador "responde" à suspensão e executa a tarefa correspondente. Isto significa que não existe tempo "morto" porque o controlador está sempre a fazer algo de útil – como se houvesse um par de tarefas a serem executadas em simultâneo.

SECÇÃO PRÁTICA

4. EXEMPLO 1: ILUMINAÇÃO DE LEDS

Com este exemplo, não vai aprender nada de novo sobre programação. Neste momento já deve saber como ligar e desligar LEDs. Nesta secção vai aprender a fazer uma instalação elétrica para verificar que um LED ou outros periféricos podem ser ligados depois de receberem um nível lógico “0” ou um nível lógico “1”. Observe o diagrama na Figura 10.

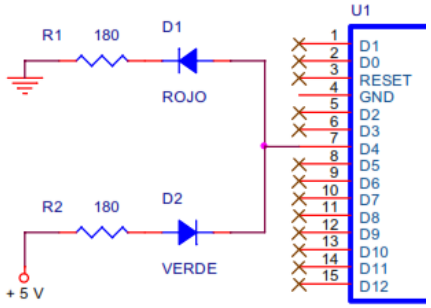

Figura 10

O mesmo pin the output D4 irá atuar em ambos os LEDs. Mas repare que é o anodo do LED vermelho (*rojo*) com o catodo que se direciona ao GND que está ligado. Por isso, este LED acender-se-á quando D4=“1”. Por outro lado, o D4 está ligado ao catodo do LED verde com o anodo direcionado a +5 V; este LED, por isso, acende-se quando D4=“0”.

O programa é muito simples. O pin D4 é configurado como output na função setup().

O programa principal é um loop contínuo. O pin D4 está no nível “1”. O LED vermelho permanece aceso enquanto o verde estiver apagado.

Cronometrado a 0.25 segundos, o pin D4 é programado para o nível “0”. O LED vermelho apaga-se e o verde acende-se. Após os 0.25 segundos, o ciclo repete-se (Figura 11).

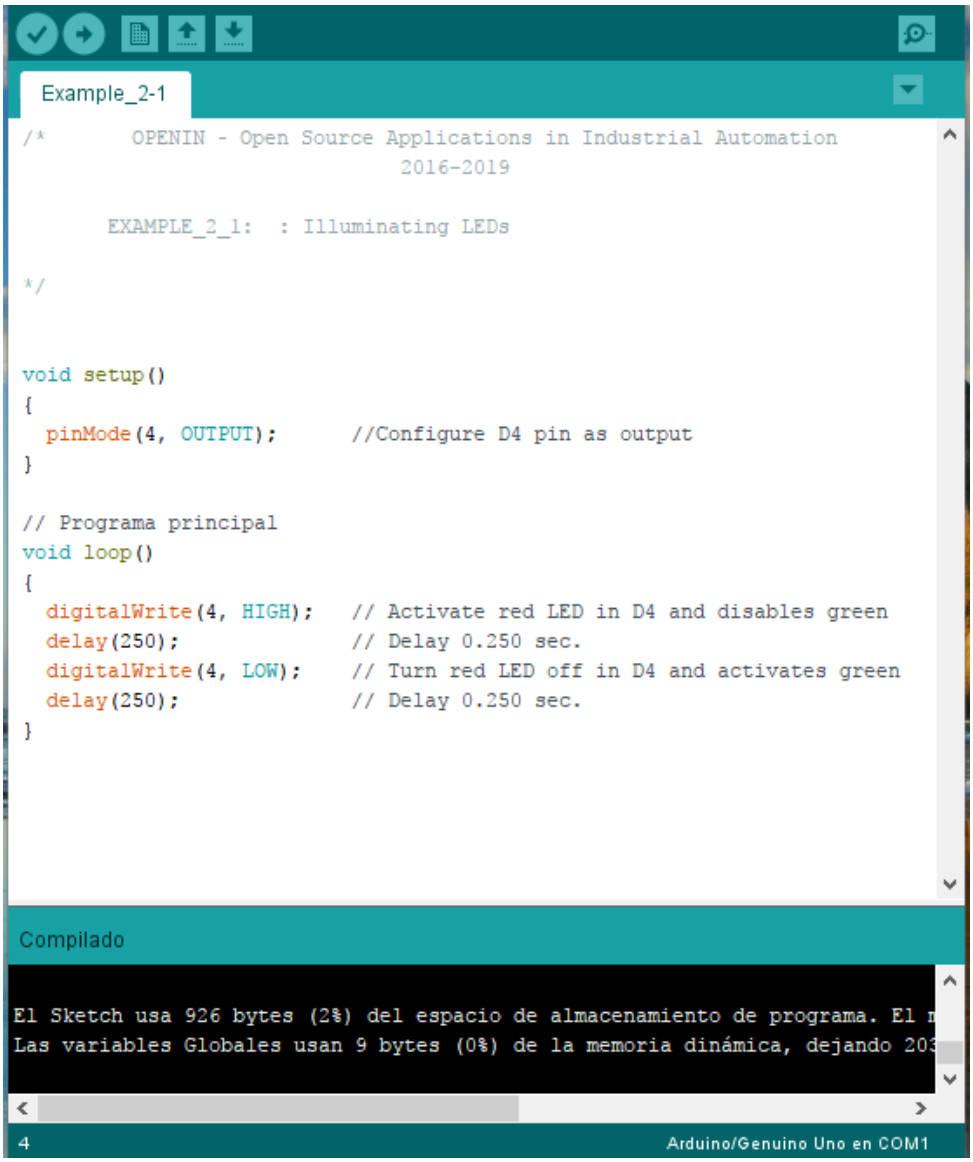

Figura 11

A Figura 12 dá uma ideia de como se apresenta a montagem prática na placa do módulo. Observe o *layout* e posicionamento dos componentes.

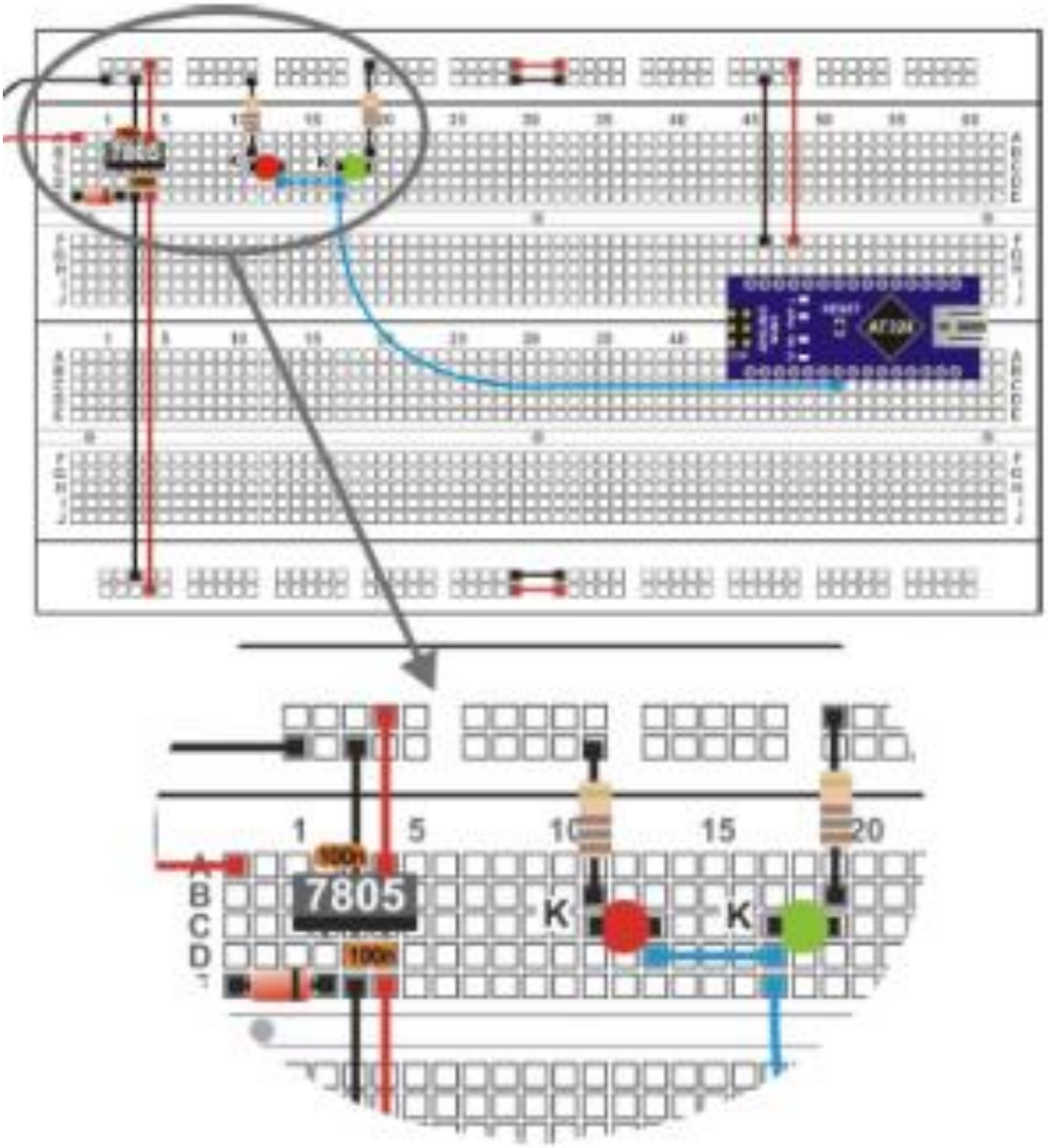

Figura 12

Agora é a sua vez!

O programa é muito simples, por isso não o vai alterar. O que pode fazer é modificar o valor da resistência de absorção R2 do LED verde. O seu valor é de 180 Ω, por isso, altere-o para 10 KΩ.

O que é que acontece?

-

Porquê?

-

Experimente e descubra quanta corrente passa pelo LED verde no momento:

$$I = \frac{V}{R} = \frac{V - V_{AK}}{R} = \frac{5 - 1.5}{10000} = 0.00035\ A = 0,35\ mA$$

Pode observar agora que a corrente (I) que atravessa o LED é de 0.35 mA quando o fabricante sugere que deveria ser aproximadamente 20 mA. Agora já conhece outra forma de regular o brilho de um LED iluminado. Coloque novamente a resistência R2 no seu valor 180 Ω original.

5. EXEMPLO 2: MONITORIZAÇÃO DE INPUTS

Na Figura 13, encontra-se o diagrama. É exatamente o mesmo do exemplo anterior, com exceção de um botão SW1 que ligado a D2.

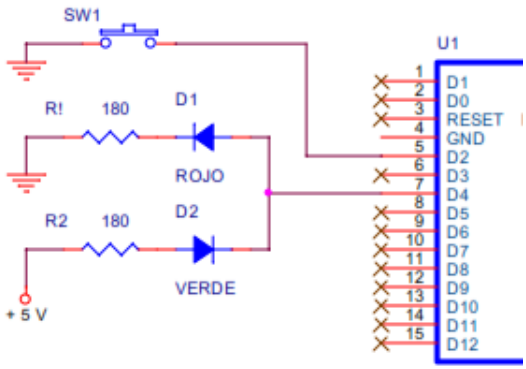

Figura 13

Este exemplo também não nos revela nada de novo em relação a programar. O que tem de fazer é ler o nível lógico do pin de *input* D2 e aplicá-lo aos LEDs verde e vermelho. Se o *input* estiver em “1”, o vermelho acende-se; se estiver em “0”, acende-se o verde. Ambos os LEDs são controlados pelo pin de output D4.

Sugerimos que utilize alicates de ponta plana para apertar os pins nos botões, como apresentado na Figura 14, e que insira os botões adequadamente na placa do módulo. Os pins têm de ficar completamente perpendiculares ao corpo do botão. A Figura 14 exemplifica como deve ser o aspeto da montagem na prática. Deve gradualmente familiarizar-se com a montagem do circuito na placa do módulo.

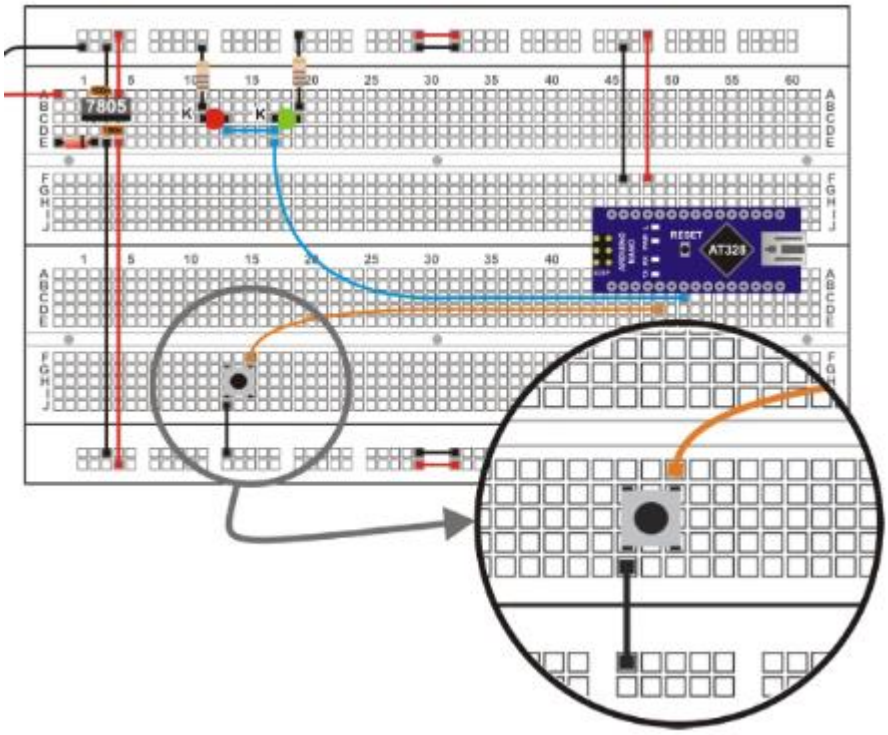

Figura 14

Agora é a sua vez!

Como referido anteriormente, não há nada no programa do exemplo 2 que já não saiba. Tudo o que tem a fazer é gravar no Arduino UNO e certificar-se que funciona corretamente. Pode ter uma surpresa.

É óbvio que quando o botão é premido, se envia um sinal para o nível “0” através do pin D2 e que o LED verde dever-se-ía acender. Mas, caso não o prima, o que é que acontece?

Parece que os LEDs se acendem aleatoriamente. Isto acontece porque quando não se prima o SW1, o input D2 está a “flutuar” e não adquire um nível lógico definitivo. Preste novamente atenção ao ponto 2 na secção teórica.

Altere o circuito elétrico ligeiramente adicionando uma resistência 10 KΩ como se apresenta no diagrama teórico (Figura 15) e no da instalação elétrica (Figura 16). Referimo-nos à resistência pull-up. Esta resistência certifica-se que o pin D2 permanece ancorado ao nível “1” enquanto o botão não for pressionado.

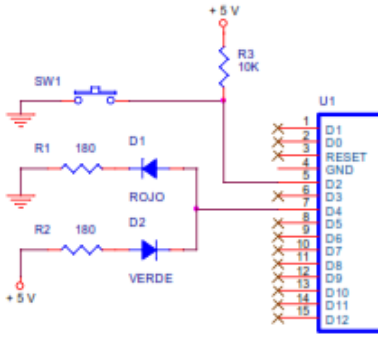

Figura 15

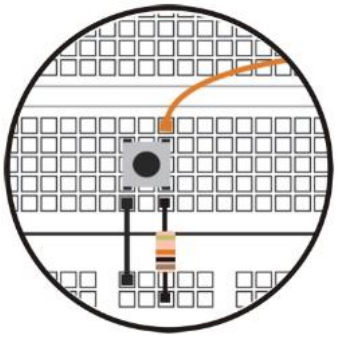

Figura 16

Altere o circuito elétrico ligeiramente adicionando uma resistência 10 KΩ como se apresenta no diagrama teórico e no da instalação elétrica. Referimo-nos à resistência pull-up. Esta resistência certifica-se que o pin D2 permanece ancorado ao nível “1” enquanto o botão não for pressionado.

Certifique-se que o mesmo exemplo 2 funciona adequadamente sem o modificar e que os LEDs acendem complete e normalmente. Vai aperceber-se que os detalhes são importantes.

6. EXEMPLO 3: MONITORIZAÇÃO DE INPUTS COM RESISTÊNCIAS PULL-UP

Este exemplo fornece-nos uma solução definitiva para as resistências pull-up. O pin D2 está configurado como input com uma resistência pull up para que não tenha de adicionar uma externa. E

agora o circuito elétrico apresenta-se exatamente como o primeiro diagrama do exemplo anterior. O único aspeto diferente é o programa.

```
void setup()
{
  pinMode(4, OUTPUT);      //Configure D4 pin as output
  pinMode(2, INPUT_PULLUP);
}
```

O pin D2 está configurado para funcionar como um input com uma resistência pull up, utilizando a função pinMode(2, INPUT_PULLUP) que está no setup().

Antes de gravar o programa, extraia a resistência pull up de 10 K pull-up que havia instalado no exemplo anterior. Não pode haver mais do que uma resistência e, neste caso, o Arduino já tem uma.

Agora grave o programa e assegure-se que tudo está a funcionar corretamente. Os LEDs ligados ao output D4 monitorizam o estado lógico dos botões ligados ao pin D2. O vermelho acende-se quando o botão não foi pressionado (nível "1" devido à resistência pull up interna) e o verde quando é pressionado (nível "0").

Em suma, dos primeiros três exemplos, podemos retirar as seguintes conclusões:

- Qualquer periférico de output digital pode ser ligado usando o nível "1" ou o nível "0"; depende da forma como está ligado.
- Da mesma forma, os periféricos digitais podem gerar um nível "1" ou mesmo um nível "0" quando são premidos. Também depende da forma como são ligados.
- O nível "0" é tão válido quanto o nível "1". São ambos igualmente representativos.
- Alguns periféricos de input podem ou não incluir uma resistência pull-up. Se não incluirem, terá de instalar uma resistência externa ou utilizar o Arduino UNO.

7. EXEMPLO 4: MONITORIZAÇÃO DE INPUTS COM SUSPENSÕES

Mais uma vez, aqui está um exemplo que não lhe irá ensinar nada que já não saiba sobre programação, mas que irá permitir-lhe avaliar se é ou não adequado utilizar suspensões. Foi ligado um LED vermelho ao pin de *output* D4 que irá monitorizar constantemente o estado do pin de *input* ligado ao D2. A ideia é que o outro LED, o verde, ligado ao output D5, acenda e apague cada cinco segundos ao mesmo tempo. Parece simples, não parece?

A Figura 17 mostra o diagrama do circuito. O LED vermelho é controlado a partir do D4; monitoriza o estado do botão ligado ao D2. O LED verde é controlado a partir do D5 e acende e apaga cada cinco segundos. Repare que ambos os LEDs se acendem no nível "1".

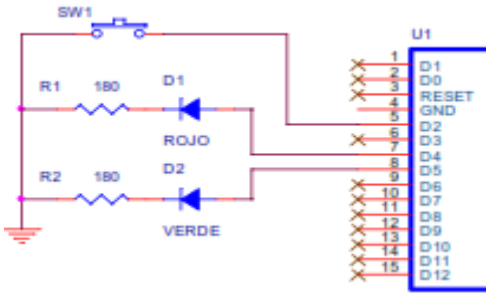

Figura 17

A Figura 18 apresenta a organização prática que tem de efetuar na placa do módulo.

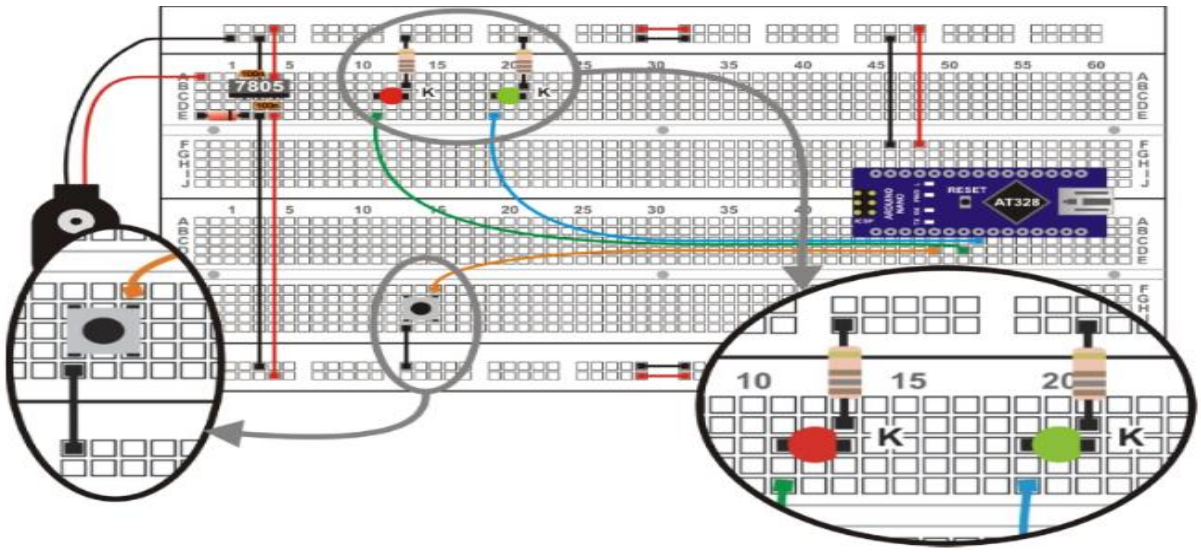

Figura 18

Assim que tiver preparado e gravado o programa, certifique-se que está a funcionar corretamente.

O LED vermelho que monitoriza o estado do botão parece que está em “espera”. E, às vezes, se pressionar o botão mesmo rapidamente, o LED vermelho “nem sequer se apercebe”. Porque é que isto acontece?

Enquanto o controlador está a executar as funções delay(500), não pode executar a função digitalWrite(4, digitalRead(2)) ao mesmo tempo. Por outras palavras, não tem capacidade para acompanhar o que está a acontecer com o botão de input D2 e, por isso, não consegue monitorizar o estado do D4 ao mesmo tempo.

Na verdade, o controlador apenas monitoriza um ciclo de ligar e desligar de cada vez a partir do final do output D5; isto acontece cada segundo. Vamos tentar resolver o problema no exemplo que se segue, utilizando uma suspensão.

8. EXEMPLO 5: MONITORIZAÇÃO UTILIZANDO INPUTS COM SUSPENSÕES

Este exemplo propõe-se a fazer exatamente o mesmo que o anterior, mas de forma adequada. O LED ligado ao output D5 ativa-se por meio segundo e desliga-se durante meio segundo. Ao mesmo tempo, o output D4 monitoriza o estado lógico do input D2.

Utilize o mesmo diagrama de circuito e o mesmo circuito prático. Altere o programa que tem três secções realmente diferentes. Observe a Figura 19:

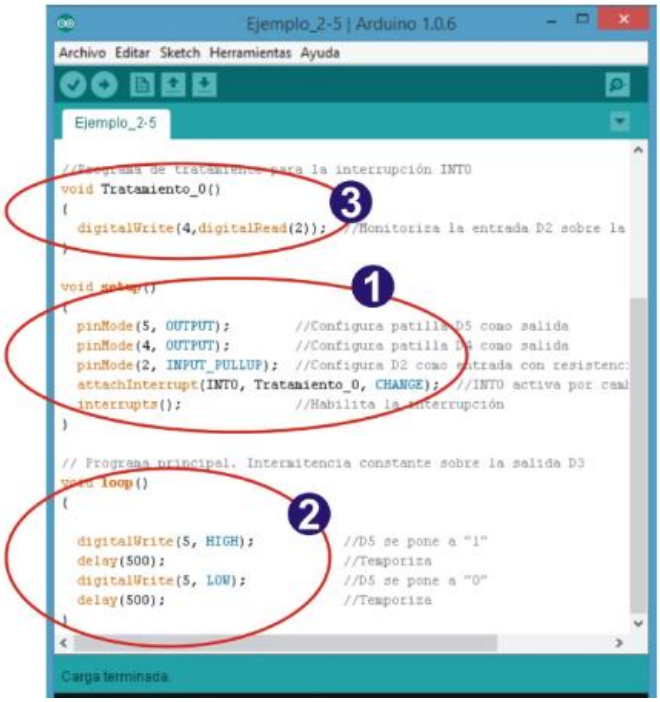

Figura 19

Setup():

Os pins D4 e D5 estão configurados como outputs. D2 está configurado como input com uma resistência pull-up. A suspensão está configurada utilizando a função attachInterrupt(). Isto associa-se ao pin INT0 (D2); acende quando é detetada uma mudança de estado no pin. Sempre que isto ocorre, a função Treatment_0() é executada. A função interrupts() permite e autoriza as suspensões, que neste caso é INT0.

loop():

Este é o programa principal. Provoca intermitência no output D5. O LED liga-se por meio segundo e desliga-se por meio segundo; nada de novo aqui.

Treatment_0():

Este é o programa que lida com a rotina de serviço de suspensão ou ISR. Sempre que é detetada uma alteração de estado no pin INT0 (D2), o controlador para o que está a fazer e executa a função Treatment_0(). Esta função lê o estado do input D2 e monitoriza o output D4. Quando fica concluída, volta ao programa principal e começa onde havia parado.

Agora quando gravar o programa poderá verificar que o controlador monitoriza instantaneamente o LED vermelho e regista o estado do input. Apesar disso, o LED verde continua a piscar como se nada tivesse acontecido. Lembre-se que o programa de tratamento – o que monitoriza o estado de input – é executado em poucos microsegundos e não tem nenhum efeito significativo no programa principal.

9. EXEMPLO 6: CONTROLAR DUAS SUSPENSÕES

Concluímos a Unidade 2 com outro exemplo que utiliza duas suspensões disponíveis no Arduino UNO: INT0 (D2) e INT1 (D3). Imagine a ferramenta de uma máquina que é responsável por controlar os dois outputs, D4 e D5, que seguem uma sequência definitiva.

Dois sensores detetam duas situações de alarme possíveis: sobreaquecimento e uma peça mal posicionada. Os sensores, coordenados pelos botões SW1 e SW2, estão ligados aos pins D2 e D3 que despoletam as suspensões INT0 e INT1 respetivamente. Observe o diagrama do circuito na Figura 20.

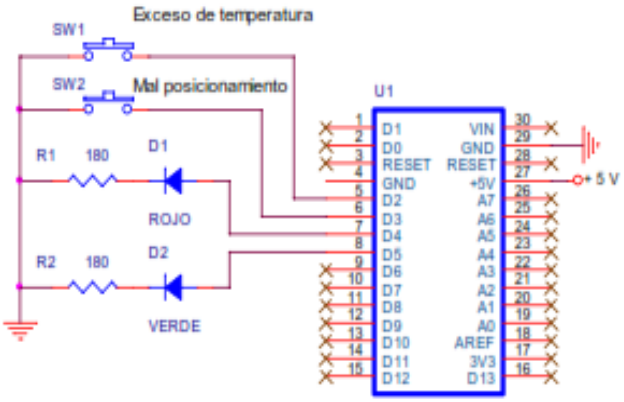

Figura 20

O programa devolve os LEDs vermelho e verde ligados e desligados em sequência. Quando uma das suspensões é detetada, o programa de tratamento adequado transmite uma mensagem de alerta em série, como as que pode observar na janela anexada ao monitor de comunicação de série (Figura 21).

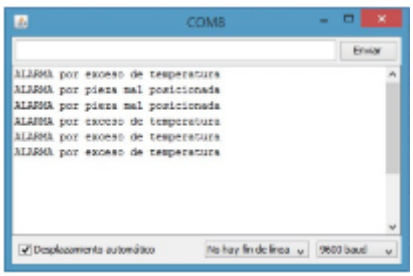

Figura 21

Ambas as suspensões estão configuradas para serem vantagens decrescentes ativas. Isto ocorre cada vez que premir o botão de ligação apropriado. Lembre-se que os pins D2 (INT0) e D3 (INT1) estão configurados como inputs com resistências pull-up. Quando os botões estão em baixo, os pins estão em “1”.

Na Figura 22 abaixo pode observar a prática de montagem. É muito semelhante ao exemplo anterior. O botão SW2 ligado ao D3 (INT1) foi adicionado.

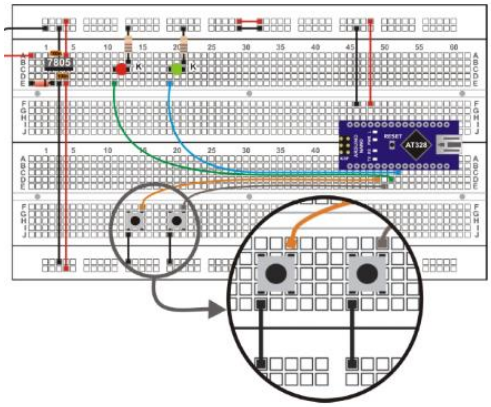

Figura 22

Assim que tiver gravado o programa, certifique-se que está a funcionar corretamente. Abra o monitor de série. O programa principal limita-se a ligar e a desligar os LEDs vermelho e verde sem que seja necessário premir qualquer botão. Se quiser, pode alterar esta opção.

Se algum dos botões for premido, a mensagem adequada irá surgir na janela do monitor de série.

Vimos que, de vez em quando, quando qualquer um das duas suspensões é despoletada, o sistema encerra. Tem de o reiniciar, premindo RESET. Isto acontece devido ao efeito “knock on” dos dois botões de ligação. O que acontece é o seguinte: mesmo que só o prima uma vez, pode ser lido como diversas suspensões consecutivas e o Arduino não tem capacidade para as tratar de forma adequada. Também vimos no laboratório que se a suspensão ou os sinais de alarme vêm de um gerador, não exercem nenhum efeito knock on, o sistema funciona adequadamente e não se fecha. Despoletamos até dez suspensões por segundo, em cinco minutos e o sistema sempre correspondeu.

Pode experimentar com outros periféricos para além de botões de ligação. É altura de pôr mãos à obra!

REFERÊNCIAS

LIVROS

[1]. **EXPLORING ARDUINO**, Jeremy Blum, Ed.Willey
[2]. Practical ARDUINO, Jonathan Oxer & Hugh Blemings, Ed.Technology in action
[3]. Programing Arduino, Next Steps, Simon Monk
[4]. Sensores y actuadores, Aplicaciones con ARDUINO, Leonel G.Corona Ramirez, Griselda S. Abarca Jiménez, Jesús Mares Carreño, Ed.Patria
[5]. ARDUINO: Curso práctico de formación, Oscar Torrente Artero, Ed.RC libros
[6]. 30 ARDUINO PROJECTS FOR THE EVIL GENIUS, Simon Monk, Ed. TAB
[7]. Beginning C for ARDUINO, Jack Purdum, Ph.D., Ed.Technology in action
[8]. ARDUINO programing notebook, Brian W.Evans

PÁGINAS WEB

[1]. https://www.arduino.cc/
[2]. https://www.prometec.net/
[3]. http://blog.bricogeek.com/
[4]. https://aprendiendoarduino.wordpress.com/
[5]. https://www.sparkfun.com