Ficha 2

Exercícios de preparação da actividade 2

Sítio:	Elearning UAb
Unidade curricular:	FATAC - Sensores e Actuadores (DMAD 2013-14)
Livro:	Ficha 2
Impresso por:	José Coelho
Data:	Quarta, 4 Junho 2014, 11:04

Índice

1 Arduino UNO, o que é?

2 Como ligar LEDs ao arduino?

3 Como ler informação de um interruptor?

4 Como ler informação de um botão?

5 Como ler informação analógica, de um potenciómetro?

6 Como gerar output analógico?

7 Como controlar um actuador piezoeléctrico?

8 Como ler informação do sensor piezoeléctrico?

9 Como utilizar um display de 7 segmentos?

10 Como utilizar um registo, para poupar o número de pins do Arduíno?

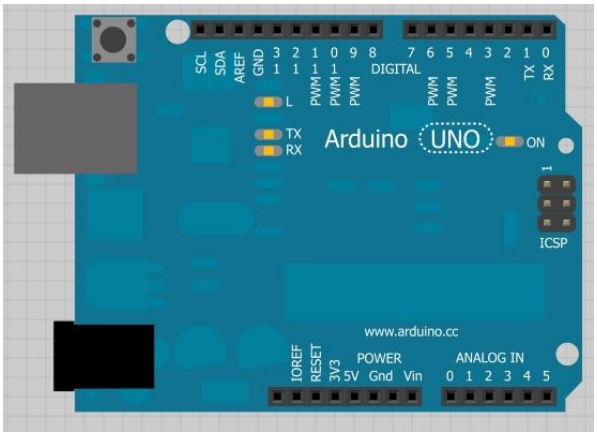
11 Como estabilizar um sinal analógico instável?

12 Como gravar e reproduzir um sinal ao longo do tempo?

1 Arduino UNO, o que é?

O Arduino UNO é uma placa de programação simples, destinada a controlar sensores e actuadores (http://arduino.cc/en/Main/ArduinoBoardUnc).

Tem 14 entradas digitais, e 6 entradas analógicas (precisão de 10 bits, valor entre 0 e 1023), podendo todas as entradas serem saídas digitais, das quais 6 pins podem ter uma saída analógica (precisão de 8 bits, valor entre 0 e 255, pins: 3, 5, 6, 9, 10 e 11). Em baixo encontra-se uma fotografia do Arduino UNO, e respectivo esquema em Fritzing.



A placa pode ser ligada a um computador por USB, podendo a ligação ser utilizada tanto para carregar o programa na placa, como para comunicar em tempo de execução com outros programas a correr, por exemplo feitos em Processing.

A alimentação pode ser feita através do cabo USB, mas pode-se também fornecer alimentação através de um transformador (canto inferior esquerdo), ou de um pin para o efeito (Vin, em baixo), devendo a tensão de entrada ser entre 7 e 12 Volts (valores recomendados). Desta forma é possível montar uma instalação apenas dependente do Arduino, de forma independente de um computador.

Nos pins digitais, a tensão correspondente ao valor 1 é de 5 Volts, e ao valor 0 de 0 Volts, sendo esta também a tensão correspondente à saída/entrada analógica máxima e mínima respectivamente.

A estrutura do programa em Arduíno é muito simples, e idêntica à estrutra de um programa em Processing, mas neste caso existem as funções: **setup**/**loop**. A função **setup** executa uma vez, tal como em Processing, e de seguida a função **loop** será permanentemente executada, tal como a função **draw** em Processing.

As principais operações específicas do Arduíno, são as seguintes:

- digitalWrite - escreve para um pin um determinado valor (LOW ou HIGH);
- digitalRead - lê o valor num determinado pin
- analogWrite - escreve para um pin (apenas os que aceitam saídas analógicas), um valor entre 0 e 255
- analogRead - lê o valor de um determinado pin de entrada analógica, um valor entre 0 e 1023
- pinMode - INPUT / OUTPUT / INPUT_PULLUP - coloca o pin em modo de entrada ou de saída. Se for em modo de entrada, pode-se especificar o INPUT_PULLUP de modo a que quando a entrada estiver em alta impedância (solta), o valor lido seja HIGH.

- delay / millis - aguarda um determinado tempo / retorna o número de milisegundos desde que a placa foi iniciada
- random / randomSeed - primitivas para geração de números aleatórios

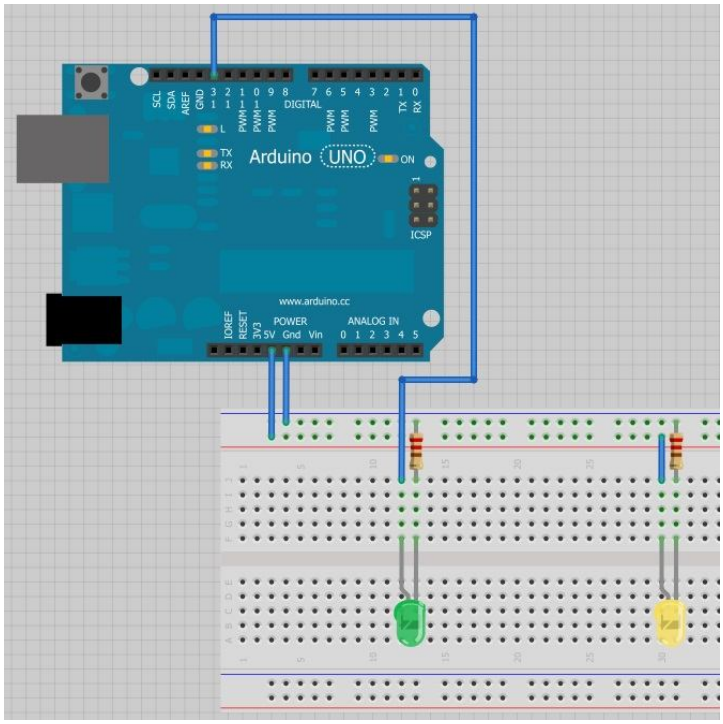
Estas operações serão demonstradas nas páginas seguintes desta ficha, neste momento é importante que fique com uma ideia do que pode fazer com o Arduíno. Siga as instruções em http://arduino.cc/en/Guide/Windows, de instalação do ambiente de desenvolvimento e ligação ao Arduino, até executar o programa Blink:

```
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {

	// initialize the digital pin as an output.
	pinMode(led, OUTPUT);

}

// the loop routine runs over and over again forever:
void loop() {

	digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
	delay(1000); // wait for a second
	digitalWrite(led, LOW); // turn the LED off by making the voltage LOW
	delay(1000); // wait for a second

}
```

Após executar este programa, e ver o LED a piscar, significa que está em condições de prosseguir.

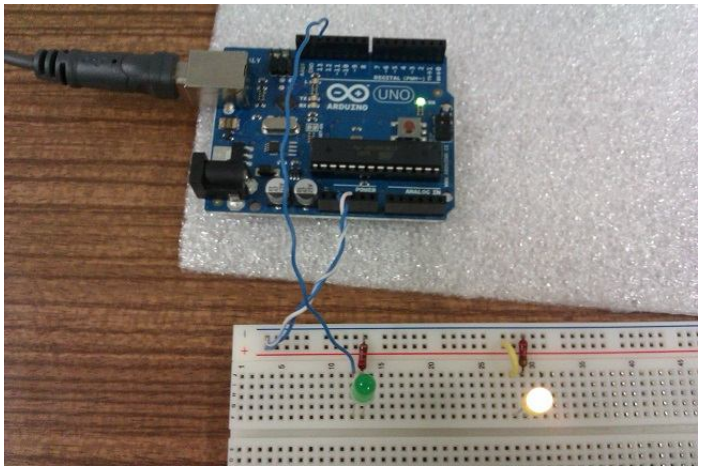
2 Como ligar LEDs ao arduino?

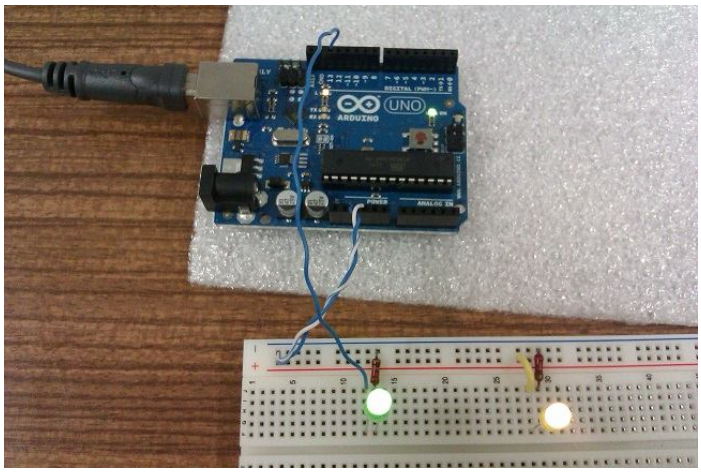
Para ligar um LED que seja controlado pelo Arduino, basta ligá-lo a um pin do Arduino, em modo de saída. Pode-se manter o programa que está na placa, o exemplo Blink, pelo que utiliza-se o pin 13. Para o LED, a fonte de alimentação é o pin, acendendo no caso do valor ficar a 1 (5 Volts), e apagando no caso de estar a 0 (0 Volts). A montagem do LED é igual à actividade anterior, tem que se colocar em série com uma resistência de 220 Ohms. Vamos colocar um outro LED, ligado tal como na ficha 1, sem estar controlado pelo Arduino.



Repare-se que nesta montagem foram alimentadas as linhas horizontais da *breadboard*, tal como na ficha 1. Deste modo facilita-se a alimentação do LED que está sempre ligado, bem como a alimentação de qualquer outro dispositivo que se utilize. Essa alimentação saiu da zona do Power, onde existe dois pins Gnd (terra), e o pin 5V (positivo). Existe ainda um pin Gnd na zona de cima do Arduíno, devendo-se utilizar o pin de 5V para alimentar outros dispositivos.

Pode-se ver a instalação física, nas duas fotografias em baixo:





Vamos agora alterar o código, mantendo a instalação. Pretendemos agora colocar um comportamento aleatório no piscar do LED, mas com alguma continuidade embora que aleatória. Para tal, basta alterar o programa e carregá-lo para o Arduino, fazendo uso de uma variável de espera, e das funções geradoras de valores aleatórios:

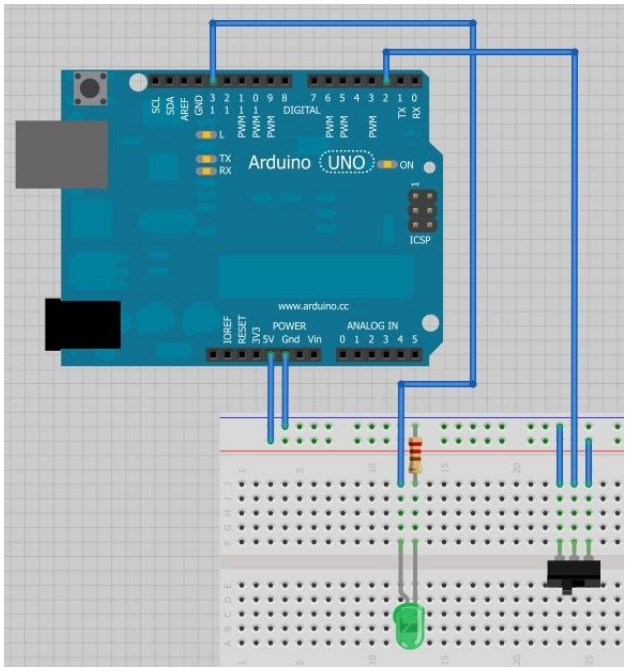
```
int led = 13, espera=1000;

void setup() {

	pinMode(led, OUTPUT);

	// inicializar a semente aleatória, de uma leitura analógica de um pin não ligado (valor aleatório)
	randomSeed(analogRead(0));

}

void loop() {

	espera+=random(-250,250);
	if(espera<200) espera=200;
	if(espera>5000) espera=5000;
	digitalWrite(led, HIGH);
	delay(espera);
	digitalWrite(led, LOW);
	delay(espera);

}
```

Notar na simplicidade em fazer esta alteração, bastou alterar o programa e carregá-lo novamente no Arduino. Sem a placa tinha-se de ter um relógio, e um contador, apenas para fazer o exemplo Blink com a espera em 1 segundo exacto. Se se quiser variar aleatóriamente o tempo, já seria um grande problema.

A estratégia aconselhada para fazer montagens utilizando o Arduino, é ligar todos os sensores/actuadores com o Arduino, e controlar tudo no programa, já que assim os circuitos de hardware ficam simples, uma vez que são entre o Arduino e o sensor/actuador, e pode-se alterar o programa mantendo a instalação, desde que os sensores/actuadores sejam os mesmos. Apenas em casos excepcionais, é que os sensores/actuadores ficam ligados entre si directamente.

3 Como ler informação de um interruptor?

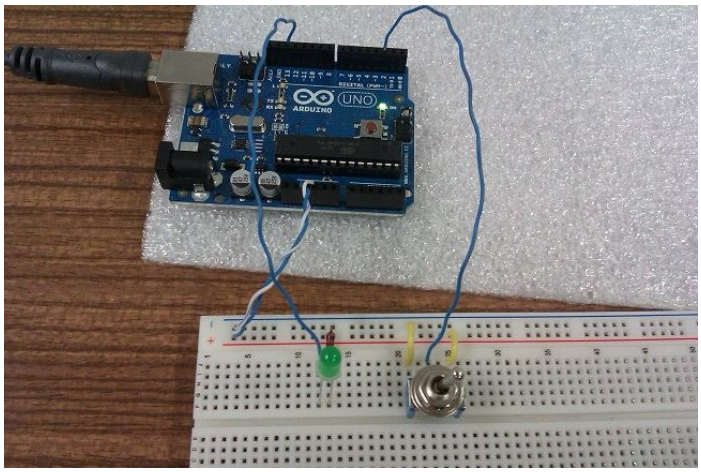
Para ler informação de um interruptor, basta ligá-lo a um pin em modo de entrada. Naturalmente que os pins do interruptor laterais, têm de estar ligados ao positivo e ao negativo tal como na ficha 1, ligando o pin central ao Arduino. Em baixo encontra-se o esquema de ligação, ao qual se mantém o LED do exemplo anterior, ligado ao pin 13. Notar que o interruptor não está ligado ao pin 0 nem ao pin 1. Esses pins não devem ser utilizados, já que são utilizados para comunicação em série com o computador ou outros dispositivos, e a sua utilização para outros fins poderia trazer problemas na comunicação, nomeadamente para carregar o programa seguinte.

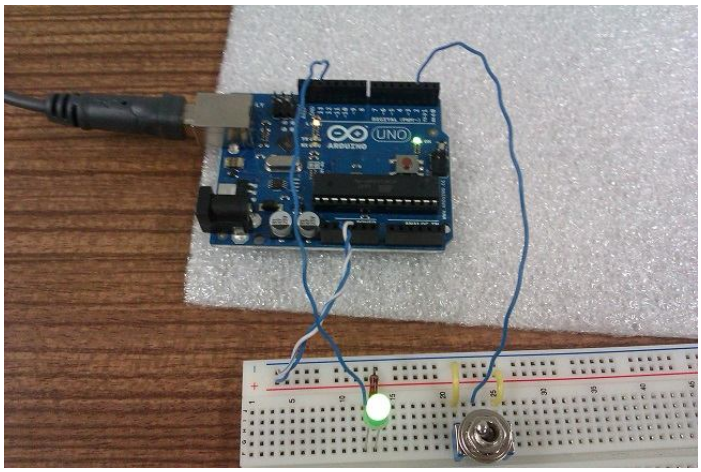


O programa no Arduino tem de ler a informação do pin 2, e colocá-la no pin 13:

```
int ledSaida = 13, interruptor=2;

void setup() {

	pinMode(ledSaida, OUTPUT);
	pinMode(interruptor, INPUT);

}

void loop() {

	int valor=digitalRead(interruptor);
	digitalWrite(ledSaida, valor);

}
```

Na função setup é indicada a utilização dos pins, e na função loop lê-se a informação e coloca-se na saída. A leitura é digital, o que significa que o valor é binário. Em baixo estão as fotografias da instalação física:

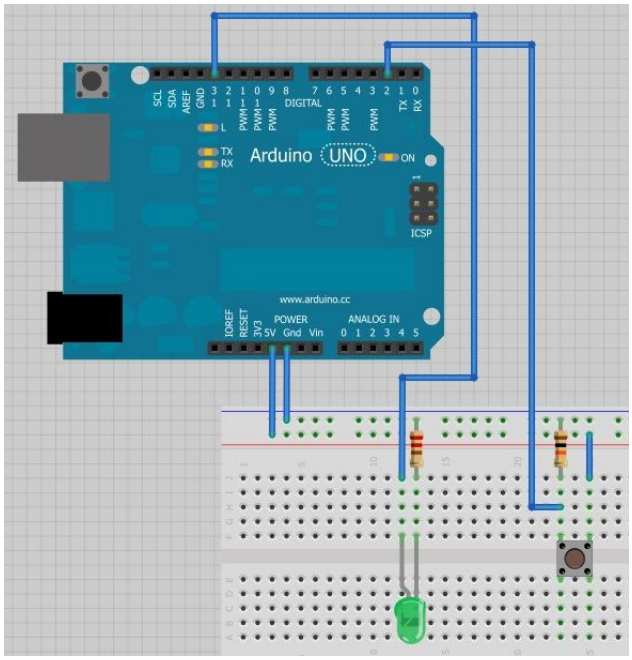




Note-se na vantagem de tanto o LED como o interruptor comunicarem directamente com o Arduíno, e não entre sí. Caso se pretenda, o programa pode fazer o que quiser, por exemplo, levar algum tempo até reflectir a informação do interruptor no LED, ou passado algum tempo, inverter o LED. Para alterar algo numa instalação sem programação, esta flexibilidade não existe. Tem que se alterar a instalação para conseguir outra funcionalidade, e por outro lado, se ligar-mos tudo ao Arduino, a complexidade máxima de todas as instalações fica limitada à ligação de cada sensor/actuador ao Arduino, ou seja, com o Arduino todas as instalações são simples.

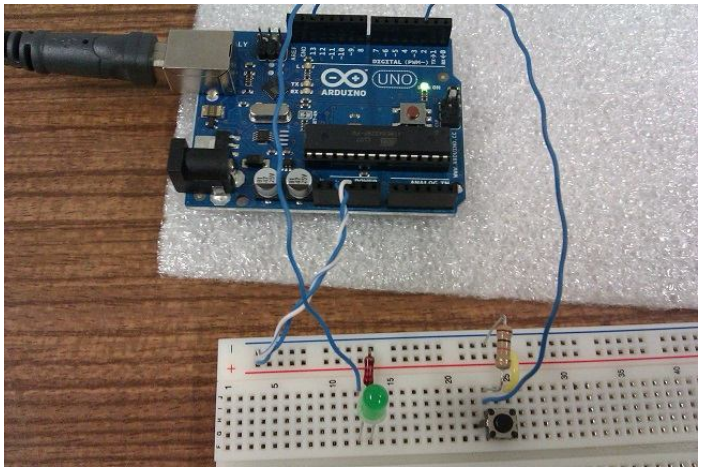
4 Como ler informação de um botão?

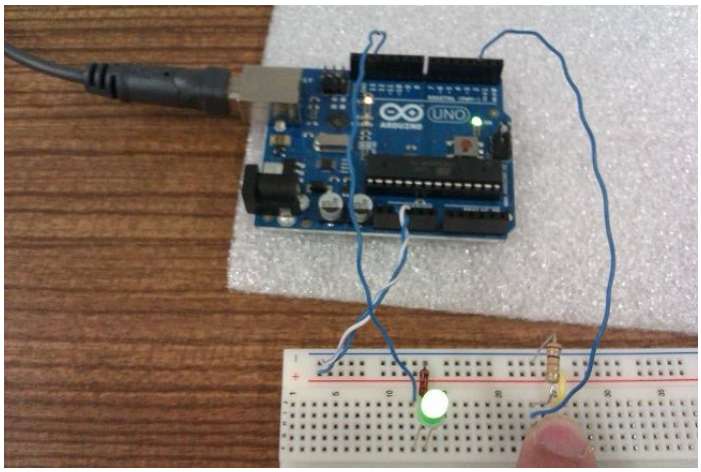
Um botão de pressão liga dois pontos, ficando ambos os pontos desligados quando o botão não está carregado. Caso se faça uma montagem idêntica ao interruptor, o que acontece é que a entrada fica em alta impedância quando o botão não está carregado, o que nesse caso pode gerar uma leitura incorrecta no Arduino. Para evitar esta situação, basta instalar uma resistência pull-down, como na ficha 1, para quando o botão não está carregado a leitura seja 0.



Repare-se no circuito, a resistência de 10 K Ohms pouco ou nada altera quando o botão está carregado, ficando o pin 2 com a tensão de 5 Volts. No entanto, quando o botão não está carregado, o pin 2 fica ligado à terra, embora através de uma resistência elevada, o que é suficiente para que no pin 2 seja lida a tensão de 0 Volts. O resto da instalação e programa é igual ao exemplo do interruptor, já que a entrada no pin 2 é também binária.

Em baixo estão as fotografias da instalação:

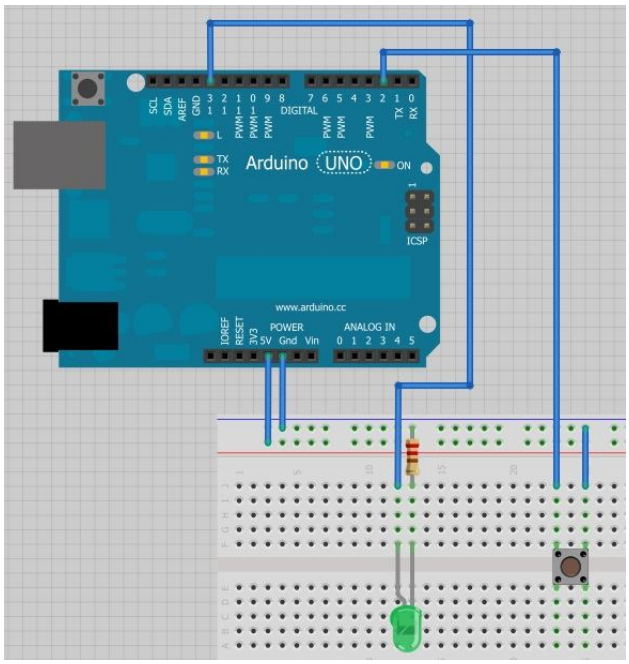




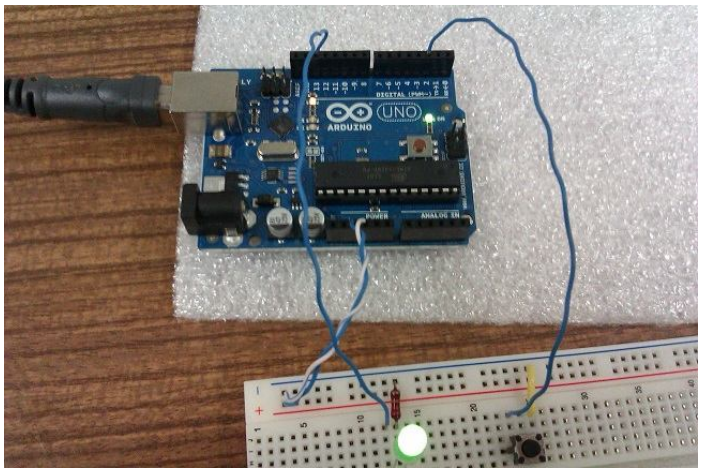
Pode-se no entanto fazer melhor com o Arduino, não sendo necessária a resistência para utilizar um botão. Na verdade, o Arduino tem uma resistência pull-up interno em cada pin, que se pode ligar, basta declarar o pin como estando no modo INPUT_PULLUP, bastando para tal alterar uma linha do programa:

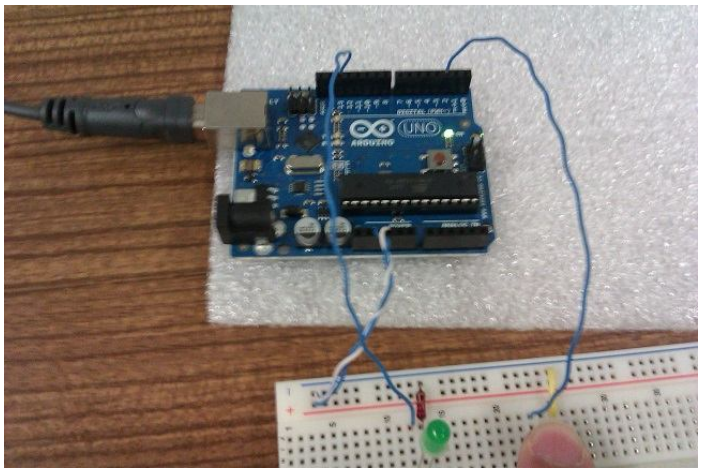
```
pinMode(interruptor, INPUT_PULLUP);
```

Como resultado, se o pin não estiver ligado nem ao positivo, nem à terra, a leitura será de 1. Neste caso, tem que se ligar o botão à terra, para que o valor lido seja alterado ao carregar no botão. Segue o esquema desta implementação:



O valor fica invertido, mas naturalmente é simples de inverter esse valor no programa. Neste caso deixou-se o programa inalterado, de modo a realçar o facto do botão ter ficado invertido:





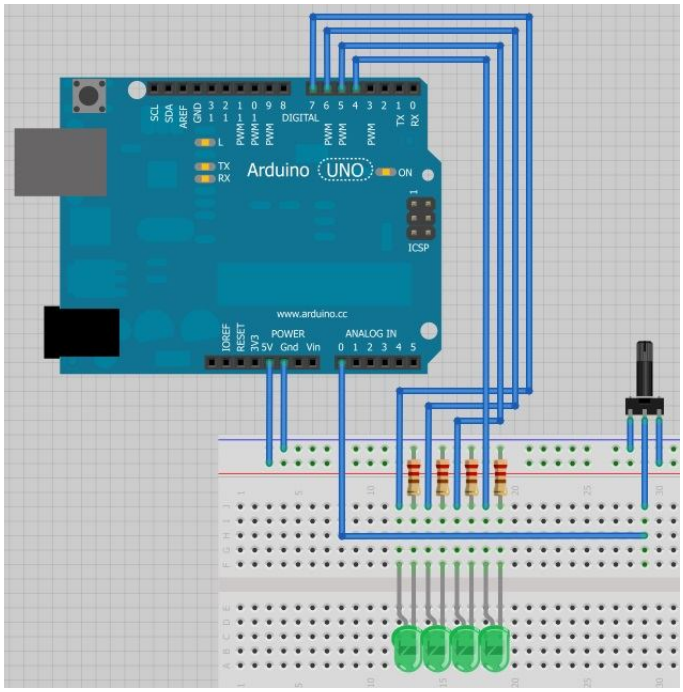
Esta segunda implementação de um botão, é preferível à primeira, já que poupa hardware.

5 Como ler informação analógica, de um potenciómetro?

Passamos agora à leitura de informação analógica. Esse tipo de informação pode vir de um potenciómetro, como neste exemplo, mas também pode ser obtida de qualquer sensor cuja saída seja analógica.

O Arduino tem 6 pins de entrada analógica, sendo a precisão de 10 bits, ou seja, a tensão de entrada entre 0 e 5 Volts, será convertida num valor entre 0 e 1023 ($2^{10} = 1024$). Esses pins são numerados de 14 a 19, mas podem ser referidos no programa pelas constantes A0 a A5.

Nesta experiência, vamos colocar a informação lida num conjunto de 4 LEDs, que vão ligando à medida que o valor do potenciómetro aumenta. Portanto a entrada é analógica, as saídas são digitais. Segue-se o esquema:

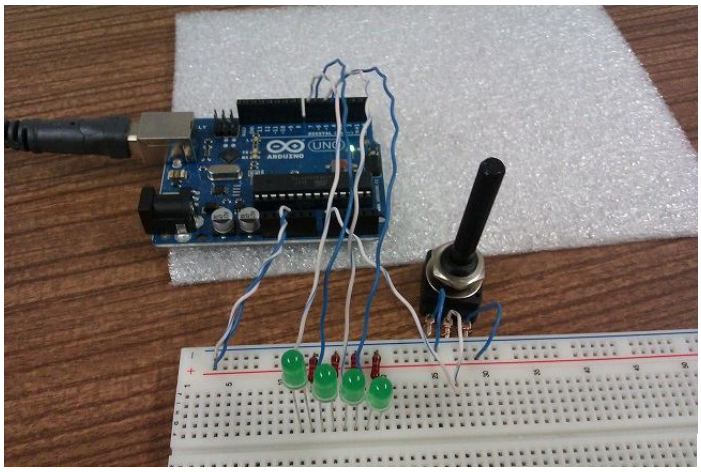


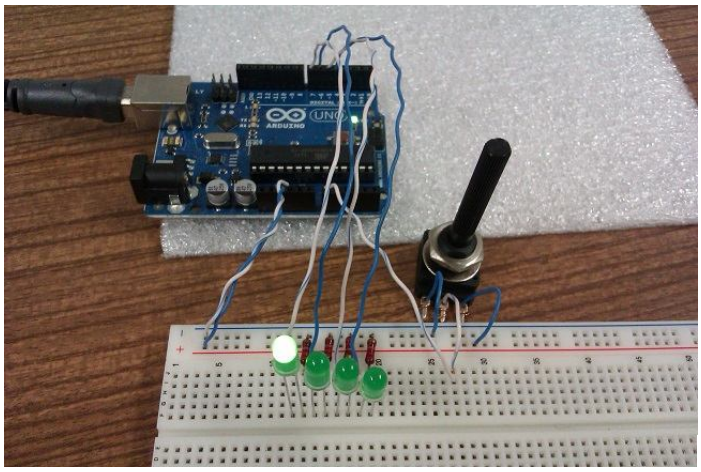
Os LEDs estão todos ligados ao Arduino (pins de 4 a 7). O potenciómetro liga-se ao Arduino tal como o interruptor, ou seja, o pin central liga-se a um pin do Arduino, mas neste caso a uma entrada analógica, e os dois extremos ligam-se ao positivo e à terra. O programa seguinte irá colocar o valor lido do potenciómetro, nos LEDs. Se o valor estiver no primeiro 1º quinto, não acendem LEDs nenhuns, se estiver no 2º quinto, acende um LED e assim sucessivamente até que acendem todos os LEDs, se o potenciómetro estiver no 5º quinto.

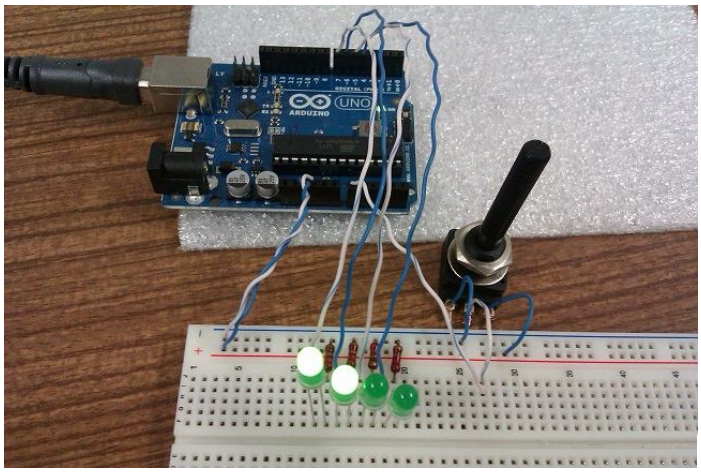
```
int leds [] = {7,6,5,4};

void setup()
{

        for(int i=0;i<4;i++)

                pinMode(leds[i],OUTPUT);

}
```

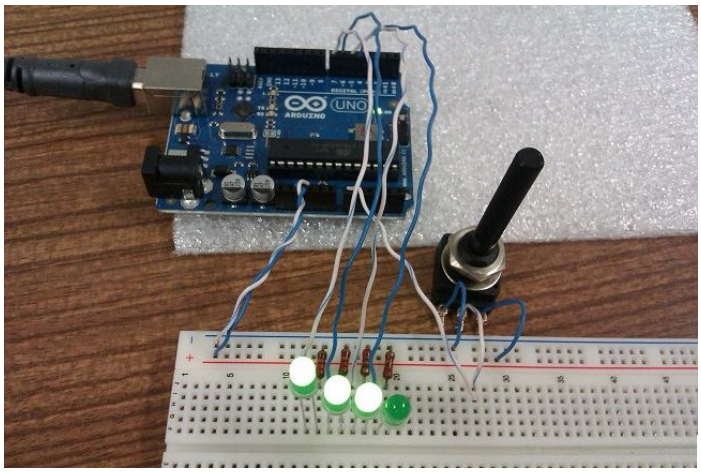
```
void loop()
{

        int valor=analogRead(A0);

        // valor entre 0 e 1023, como há 4 LEDs dividir por 5 (apagado, 1, 2, 3, 4 leds)
        valor/=(1024/5);

        for(int i=0;i<4;i++)

                if(valor>i) digitalWrite(leds[i],HIGH);
                else digitalWrite(leds[i],LOW);

}
```

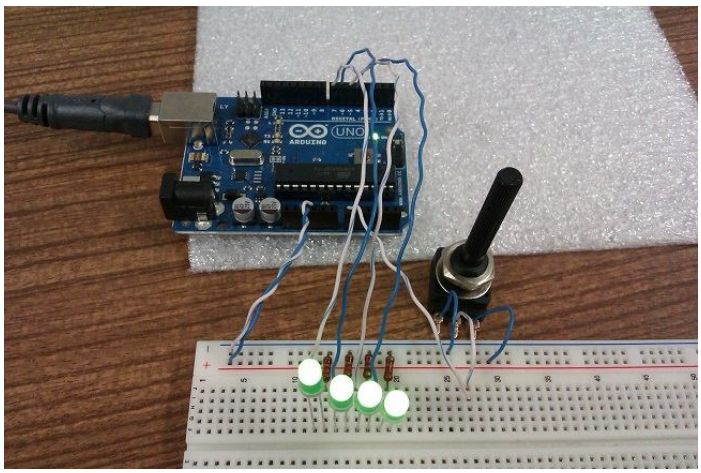
Notar que os números dos pins utilizados nos LEDs, estão num vector. É importante que assim seja, para simplificar por um lado o processamento idêntico para todos os pins, como é o caso dos dois ciclos for, que não seriam possíveis se não existisse esse vector, como também para dar flexibilidade para trocar os pins de entrada, e manter o resto do programa operacional. Seguem-se algumas fotografias da instalação:





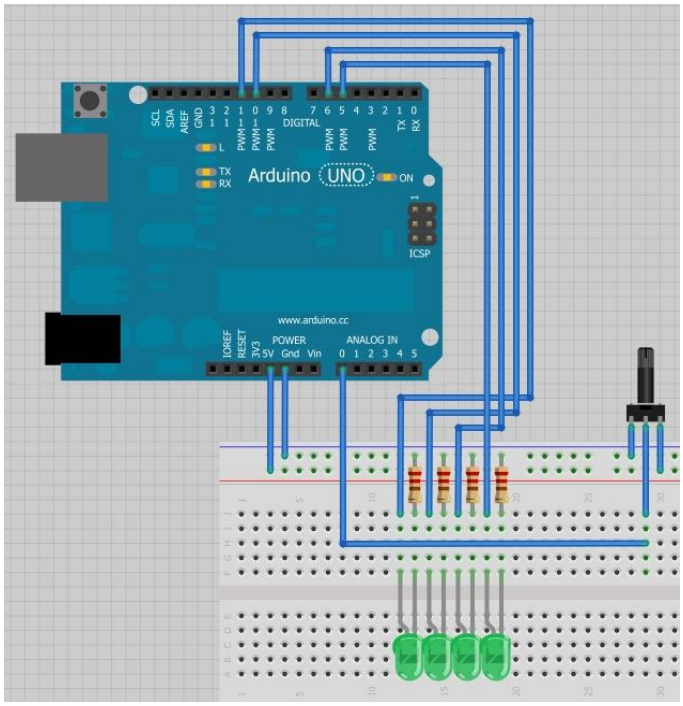






6 Como gerar output analógico?

Vamos agora reproduzir a experiência anterior, mas utilizando um output analógico para os LEDs, de modo a que o último LED ligado tenha uma intensidade dependente do valor do potenciómetro, e assim seja fornecida ao utilizador uma informação mais correcta, relativamente ao valor do potenciómetro. No entanto, apenas 6 pins podem ter uma saída analógica (têm uma marcação com um ~), e estas são de 8 bits, sendo o valor a colocar entre 0 e 255, convertido para 0 a 5 Volts. Temos portanto de alterar ligeiramente a instalação, trocando os pins aos quais os LEDs estão ligados, tudo o resto é feito no programa:

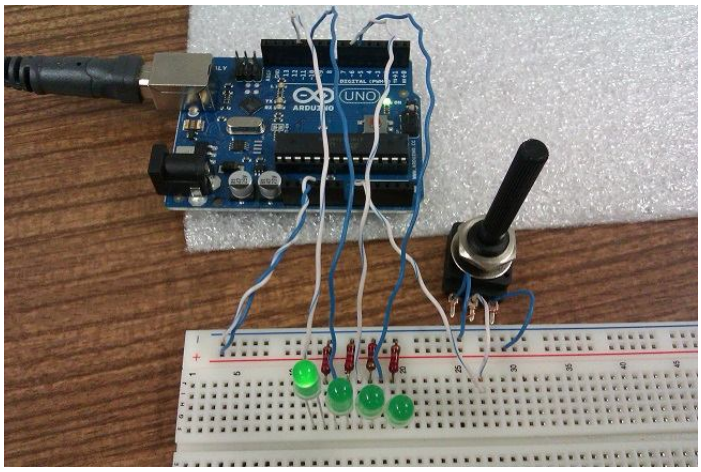


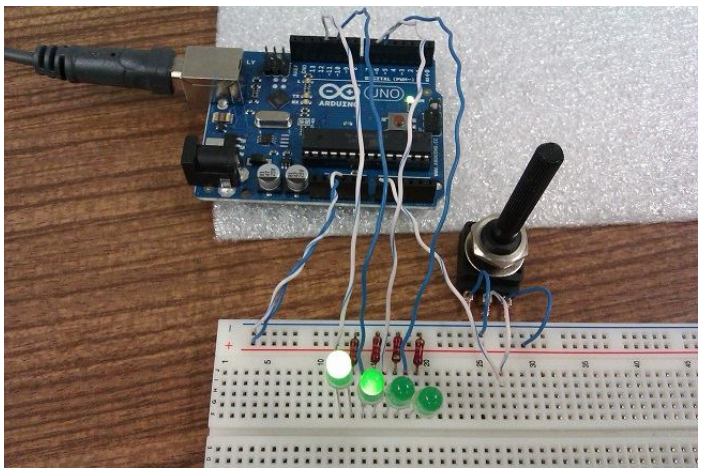
No programa, temos de actualizar os pins aos quais os LEDs estão ligados, e o valor a colocar na saída:

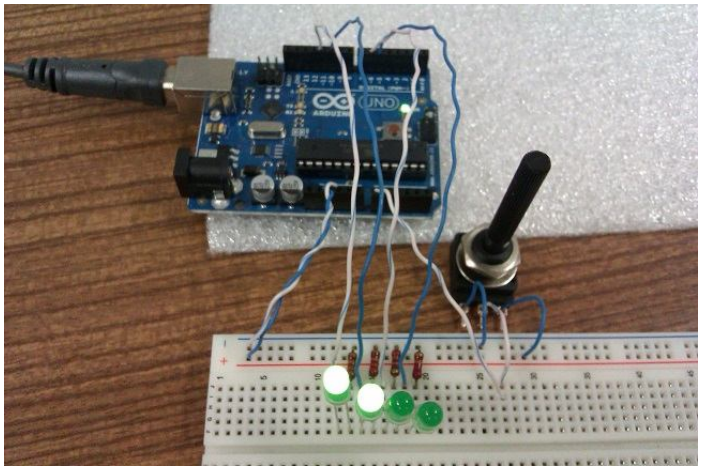
```
int leds [] = {11,10,6,5};

void setup()
{

        for(int i=0;i<4;i++)
        pinMode(leds[i],OUTPUT);

}

void loop()
{

        int valor=analogRead(A0);
        // valor entre 0 e 1023, como há 4 LEDs, saída analógica, dividir por 4
        // há 255 para cada LED, que é precisamente o valor da saída, pelo que
        // nem há contas a fazer.

        int nleds=valor/256;
```

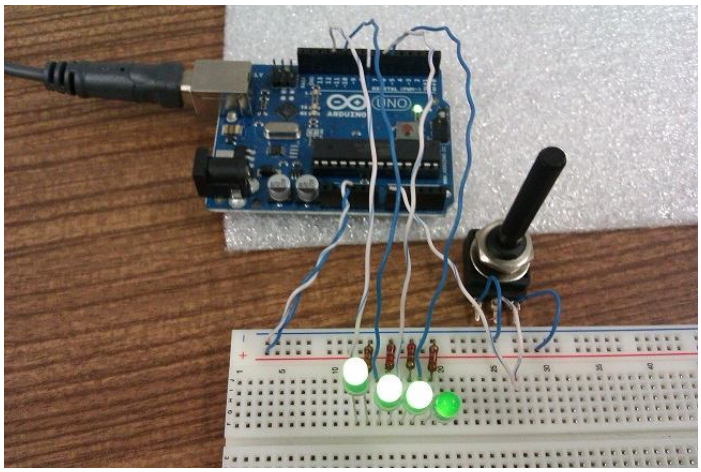
```
    int ultimoLed=valor%256;

    for(int i=0;i<4;i++)

        if(nleds>i) analogWrite(leds[i],255);
        else if(nleds==i) analogWrite(leds[i],ultimoLed);
        else digitalWrite(leds[i],0);

}
```

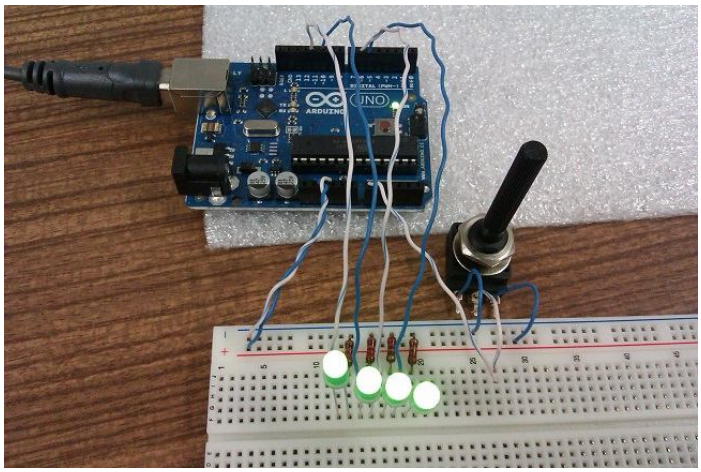
Seguem-se fotografias da instalação:







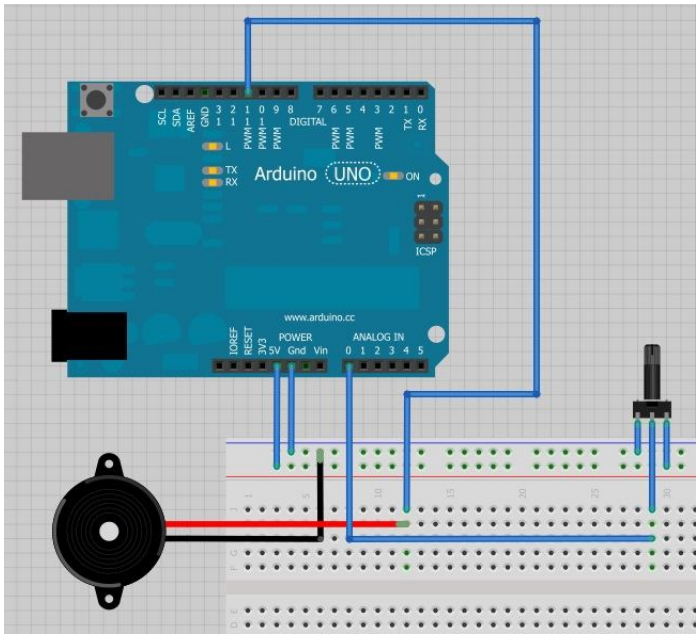




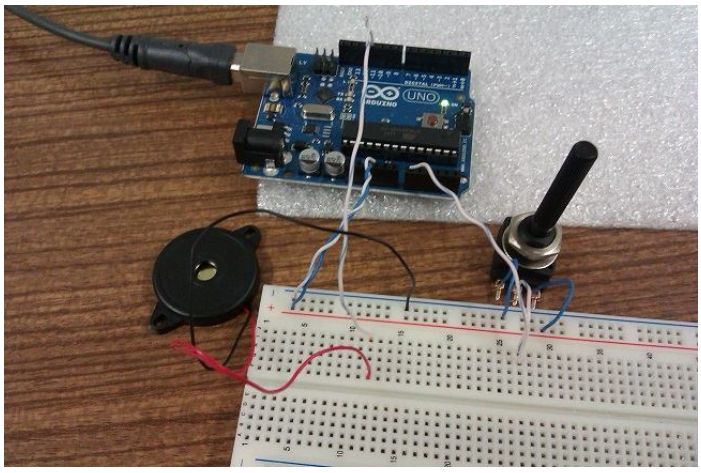
Temos aqui um exemplo de uma instalação que permanece quase igual, e com a alteração do programa fica com uma funcionalidade distinta, neste caso com mais informação. É natural, já que os sensores e actuadores permaneceram iguais, pelo que todas as instalações com os mesmos sensores/actuadores, têm a mesma complexidade na montagem, podem é ter complexidades distintas nos programas.

7 Como controlar um actuador piezoeléctrico?

O sensor/actuador piezoeléctrico, vibra e emite som quando há variação de tensão electrica, e também o inverso. Vamos reproduzir a experiência do potenciómetro, mas em vez de o ligar a LEDs, liga-se ao sensor/actuador piezoeléctrico, como se pode ver no esquema seguinte:



O valor lido no potenciómetro, é colocado no sensor/actuador piezoeléctrico, fazendo um som variável conforme a posição do potenciómetro. De seguida mostra-se uma fotografia da instalação:



Caso se ligue directamente o sensor/actuador ao potenciómetro, o resultado é o silêncio. Isto acontece porque o potenciómetro mantém o valor constante, enquanto que o Arduino vai colocando o valor em pequenos instantes de tempo, mas é o suficiente para que a tensão oscile e gere um som. Para controlar o audio de forma mais correcta, deve-se utilizar as funções tone/noTone, tendo-se que dar a frequência da nota. No entanto, a nota pode ser gerada apenas para um pin de saída, pelo que não se pode utilizar estas funções com vários pins em simultâneo.

Em baixo está o programa utilizado:

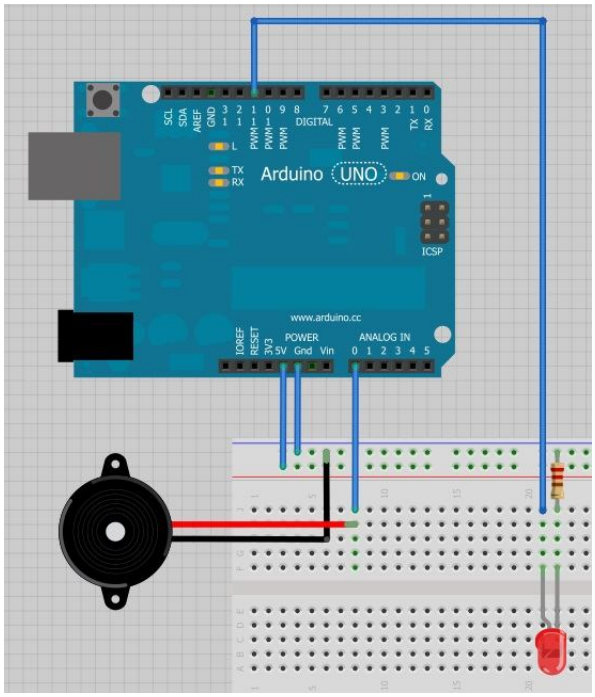
```
int piezo=11;
```

```
void setup()
{

	pinMode(piezo,OUTPUT);

}

void loop()
{

	int valor=analogRead(A0);
	// valor entre 0 e 1023, re-escalar para a saída do sensor (0 a 255)
	analogWrite(piezo,valor/4);

}
```

Caso pretenda construir uma melodia, pode fazer o seguinte tutorial:
http://www.arduino.cc/en/Tutorial/Melody

8 Como ler informação do sensor piezoeléctrico?

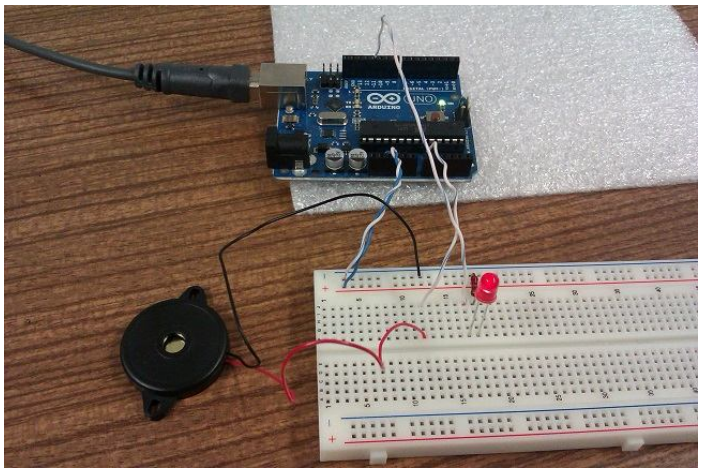
Para ler a informação do sensor piezoeléctrico, o exemplo é idêntico ao potenciómetro. O esquema é o seguinte:

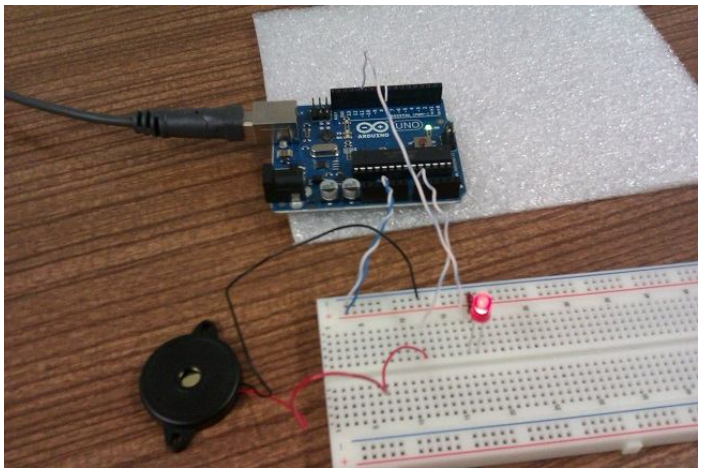


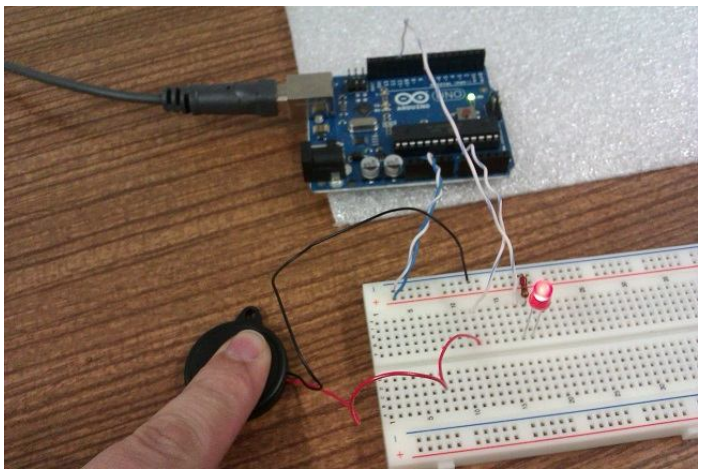
O programa mantém a simplicidade esperada, sendo o sensor tratado como se fosse um potenciómetro. No entanto pode haver necessidade de re-escalar o valor lido, no caso do sensor apenas retornar tensões entre 0 e 2,5 Volts, poder-se-ia duplicar o valor lido, de modo a ter a escala completa:

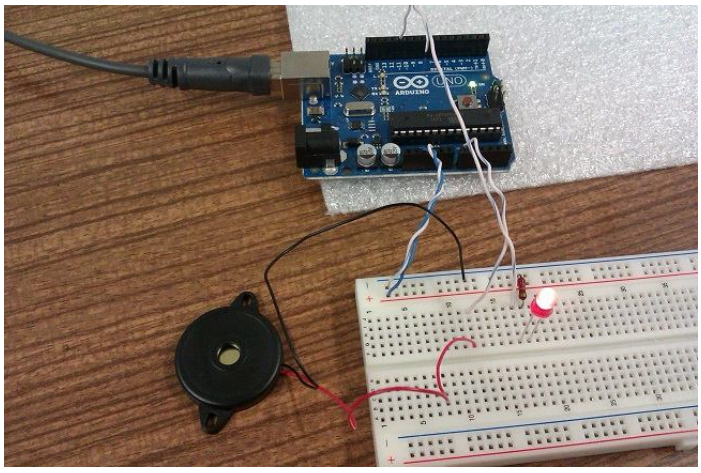
```
int piezo=A0, led=11;

void setup()
{

	pinMode(led,OUTPUT);
	pinMode(piezo,INPUT);

}

void loop()
{

	int valor=analogRead(piezo);
	// valor entre 0 e 1023, re-escalar para a saída do sensor (0 a 255)
	analogWrite(led,valor/4);

}
```

Em baixo estão fotografias da instalação:





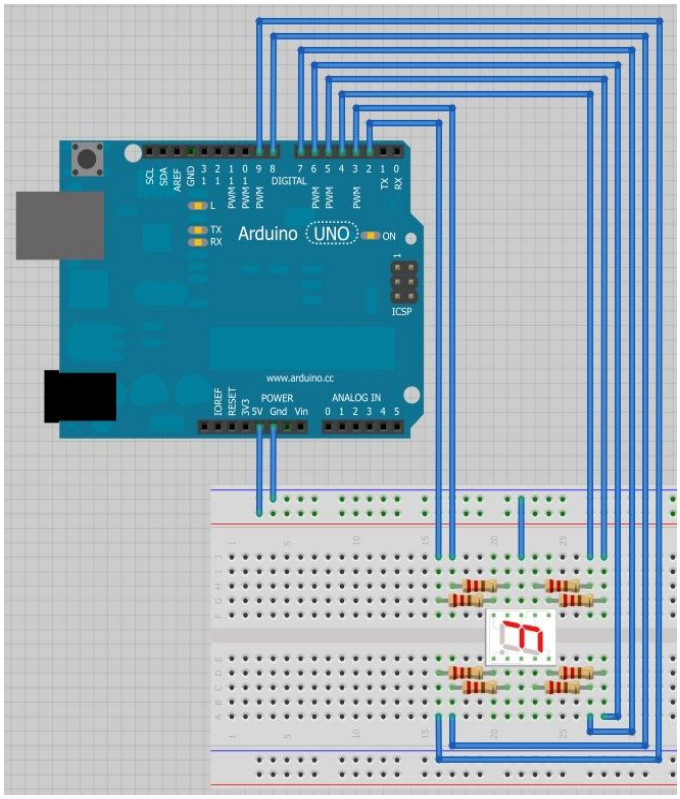




Notar que o sensor mostra tensões distintas de zero, mesmo sem estar pressionado, já que mantém a tensão durante algum tempo, após ter sido pressionado. As leituras do Arduino não retiram muita corrente, caso contrário a tensão baixaria rapidamente para 0 Volts, e o LED apagaria-se rapidamente.

9 Como utilizar um display de 7 segmentos?

Por vezes, há necessidade de controlar actuadores com muitas entradas. Um dos exemplos clássicos é o display de 7 segmentos, que permite por exemplo mostrar um número. O display na verdade é um conjunto de 7 LEDs, com um cátodo ou ânodo comum (ver na referência qual a situação), disposto de forma a que se possa formar números. Ora para controlar 7 LEDs, é necessário utilizar 7 pins do Arduino, e é isso que vamos fazer neste exemplo. Apresenta-se de seguida o esquema utilizado:



Tendo um só actuador, basta ligá-lo ao Arduino, e como tem 7 LEDs, tem que se ligar todos. Temos neste caso um cátodo comum, pelo que liga-se à terra. Na verdade existem 8 LEDs, sendo um dos LEDs o ponto final, que é útil para apresentar números com parte faccionária. Para cada LED tem que se colocar uma resistência de 220 Ohms. Evidentemente que a instalação de um sistema destes, é mais complicada que as anteriores, já que requer um elevado número de ligações. Torna-se naturalmente importante que os fios sigam todos juntos, de modo a minimizar a confusão.

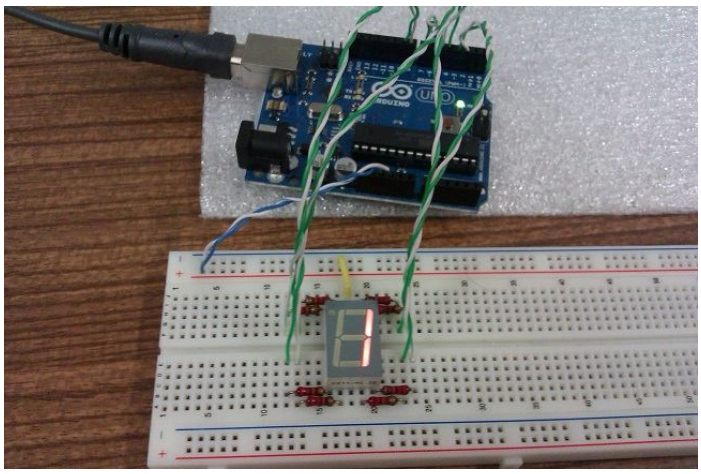
Vamos pôr o programa a colocar alguma informação no display, por exemplo o minuto ou segundo actual:

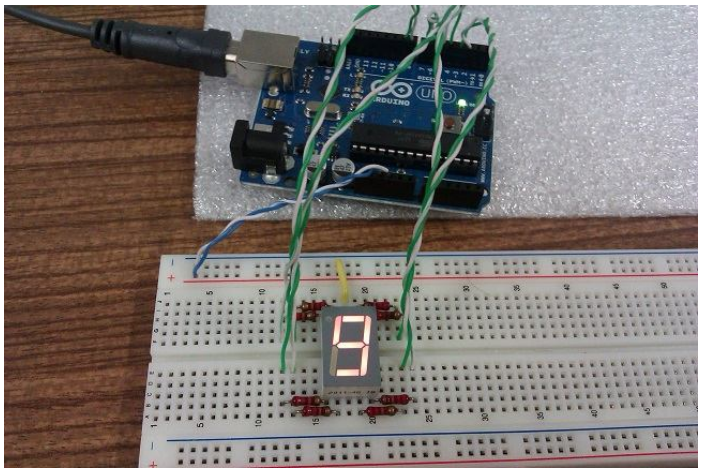
```
int segmentos[]={4,5,3,6,7,9,2,8};

void setup()
{

        for(int i=0;i<8;i++)

                pinMode(segmentos[i],OUTPUT);

}

void OutputDisplay(int numero)
```

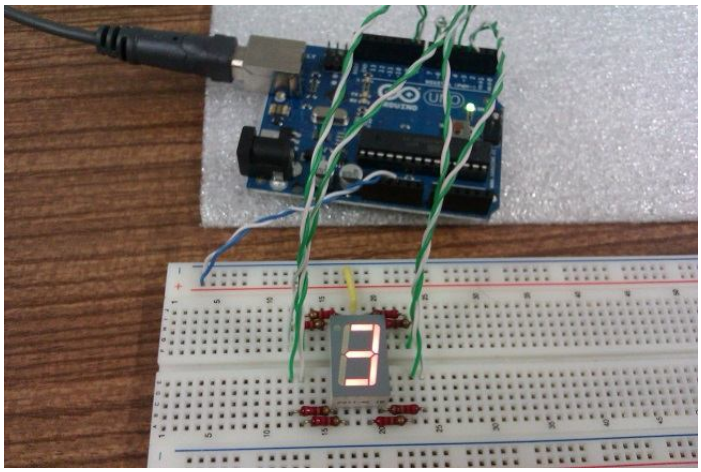
```
{

    int digitos[10][8]={

        {HIGH,HIGH,HIGH,HIGH,HIGH,HIGH,LOW,LOW}, // 0
        {LOW,HIGH,HIGH,LOW,LOW,LOW,LOW,LOW}, // 1
        {HIGH,HIGH,LOW,HIGH,HIGH,LOW,HIGH,LOW}, // 2
        {HIGH,HIGH,HIGH,HIGH,LOW,LOW,HIGH,LOW}, // 3
        {LOW,HIGH,HIGH,LOW,LOW,HIGH,HIGH,LOW}, // 4
        {HIGH,LOW,HIGH,HIGH,LOW,HIGH,HIGH,LOW}, // 5
        {HIGH,LOW,HIGH,HIGH,HIGH,HIGH,HIGH,LOW}, // 6
        {HIGH,HIGH,HIGH,LOW,LOW,LOW,LOW,LOW}, // 7
        {HIGH,HIGH,HIGH,HIGH,HIGH,HIGH,HIGH,LOW}, // 8
        {HIGH,HIGH,HIGH,HIGH,LOW,HIGH,HIGH,LOW}, // 9
        };

    for(int i=0;i<8;i++)

        digitalWrite(segmentos[i],digitos[numero%10][i]);

}


void loop()
{

    OutputDisplay(millis()/1000);
    delay(1000);

}
```

Os segmentos estão num vector, de modo a ser simples trocá-los. Ao montar a instalação, não interessa a que LED corresponde cada pin. É preciso garantir é que todos os pins sejam ligados. Após a montagem estar pronta, tem que se acertar os pins, por troca da ordem dos pins no vector segmentos. A matriz de dígitos tem o valor dos LEDs que devem ser ligados para cada número, e não deve ser alterada, apenas a ordem dos elementos no vector segmentos.

Segue-se em baixo algumas fotografias da instalação:







Repare-se que na montagem houve preocupação de manter os fios juntos, de modo a parecerem poucos, tal como as resistências. Caso faça esta montagem, naturalmente que à primeira não irão aparecer os números correctos. Pode alterar o programa para colocar o valor 8, que tem todos os 7 LEDs ligados excepto o LED do ponto, e caso não esteja um 8, saberá onde está o LED a que corresponde ao ponto final, pelo que deve trocá-lo, e assim sucessivamente. Verifica depois os restantes números, e quanto muito encontrará um ou outro LED trocado. Evidentemente que se não tiver esta flexibilidade do lado do programa, não poderia ligar os LEDs aos pins do Arduino sem prestar atenção a que pin corresponde cada LED, e teria naturalmente uma tarefa muito mais complexa.

10 Como utilizar um registo, para poupar o número de pins do Arduíno?

O exemplo anterior é muito interessante, mas por cada display são gastos 8 ou no mínimo 7 pins. Ora o Arduino tem 14+6 pins, pelo que no máximo poderia ter 2 displays de 7 segmentos. Como ultrapassar esta limitação, e simplificar o número de ligações entre o Arduino e a instalação? Basta que se utilize um circuito integrado com um registo de 8 bits, com saída paralela (para alimentar o display de 7 segmentos), e entrada em série, de modo a evitar ser necessário utilizar-se muitos pins. Nesta página mostra-se como utilizar os integrados 74LS259 e 74LS164.

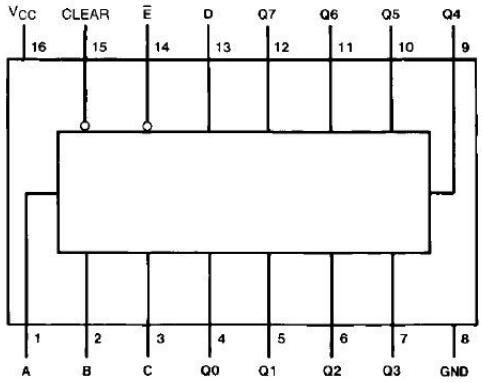
Existem outros integrados que podem ser úteis utilizar-se com o Arduino, como por exemplo memórias com entradas e saídas em série. Não gastam muitos pins e podem ser úteis no caso da memória do Arduino não ser suficiente.

Circuito Integrado 74LS259

Fica aqui um exemplo de utilização do circuito integrado 74LS259, cuja informação essencial se apresenta de seguida:

DM74LS259

Connection Diagram



Function Table

Inputs		Output of Addressed Latch	Each Other Output	Function
Clear	$\overline{E}$			
H	L	D	Q_{i0}	Addressable Latch
H	H	Q_{i0}	Q_{i0}	Memory
L	L	D	L	8-Line Demultiplexer
L	H	L	L	Clear

Latch Selection Table

Select Inputs			Latch Addressed
C	B	A	
L	L	L	0
L	L	H	1
L	H	L	2
L	H	H	3
H	L	L	4
H	L	H	5
H	H	L	6
H	H	H	7

H = HIGH Level
L = LOW Level
D = the Level of the Data Input
Q_{i0} = the Level of Q_i (i = 0, 1,...7, as Appropriate) before the Indicated Steady-State Input Conditions Were Established.

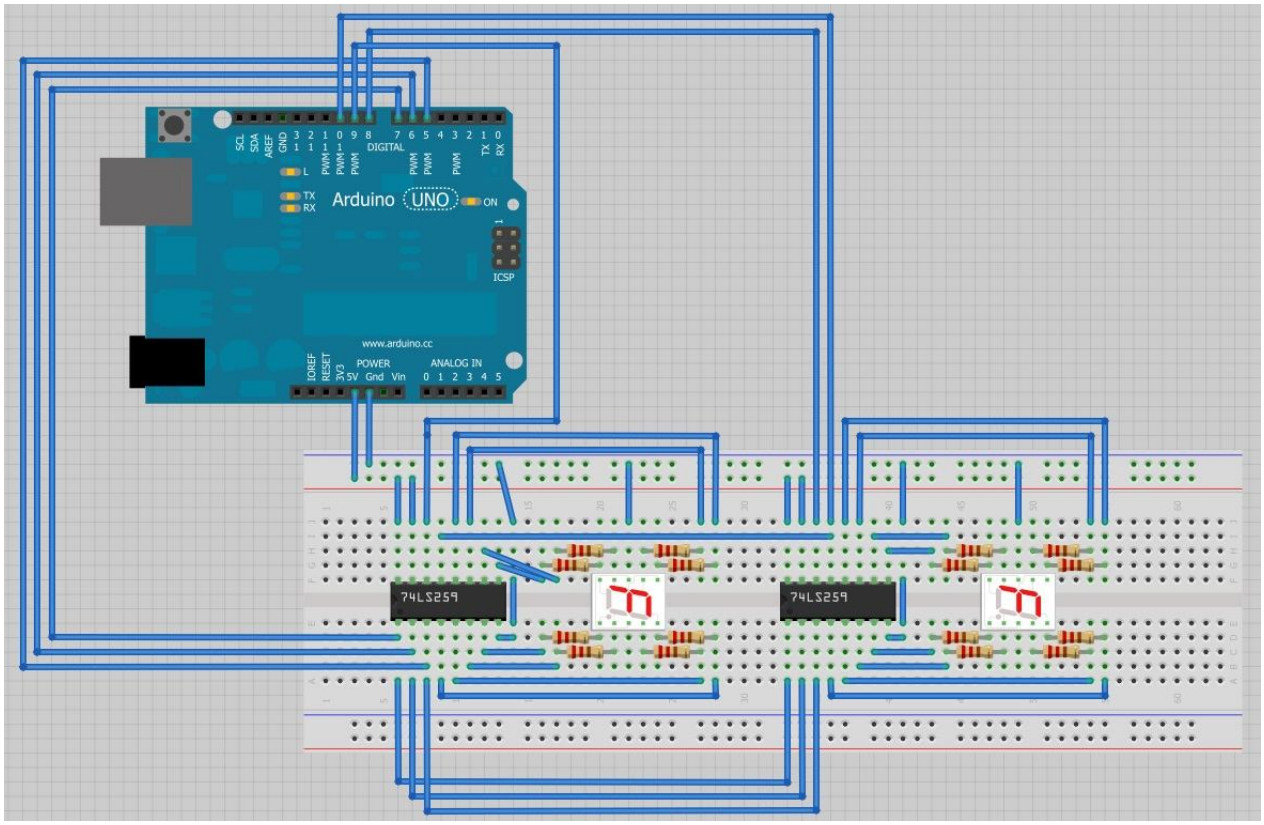
Embora este integrado não seja dos mais simples de utilizar, dado que a entrada não é em série mas sim através do endereço de cada bit (entradas A,B,C indicam o endereço de cada bit), utiliza-se este para exemplificar o caso mais complexo. O integrado tem vários modos de operar, como se pode ver na tabela de função. Com o Clear a L, tanto pode limpar os valores no registo, como ser utilizado como um *desmultiplexor* (indica 1 na saída endereçada, e 0 nas restantes, ou seja, descodifica o número binário colocado nas entradas ABC). Este modo não é o interessante, mas sim com Clear a H, em que tanto pode funcionar como memória, ou seja, repete o valor colocado em cada bit (no caso de E ser H), como pode alterar o valor do bit endereçado com o valor em D.

Temos portanto que ligar o Clear a H fixo, já que não nos interessa o modo alternativo, e ligar 3 bits

(ABC), com o endereço do registo, e um bit D com o valor a colocar no registo, para além de se ligar também o bit com o E, para indicar ao integrado que é altura de gravar D, no bit indicado por ABC. No total utilizam-se 5 pins do Arduino, poupando portanto 2 pins.

Não sendo esta uma grande poupança, com um outro integrado poder-se-ia utilizar muito menos pins, vamos ainda assim fazer valer esta vantagem colocando dois displays de 7 segmentos, em vez de um. Neste caso, em vez dos 5 pins, utilizam-se não 10 pins, mas sim apenas 6 pins. É necessário um pin extra para a entrada E do segundo integrado, todos os restantes pins podem ser partilhados.

Apresenta-se o esquema proposto:



Com dois display de 7 segmentos, há menos pins em utilização que no exemplo anterior. No entanto a que custo? Como é que no programa se coloca um valor em cada display? O custo existe apenas na montagem, no programa pode-se fazer uma função para colocar o valor que se pretender, e assim podemos abstrair-nos da tarefa de comunicar com os circuitos integrados:

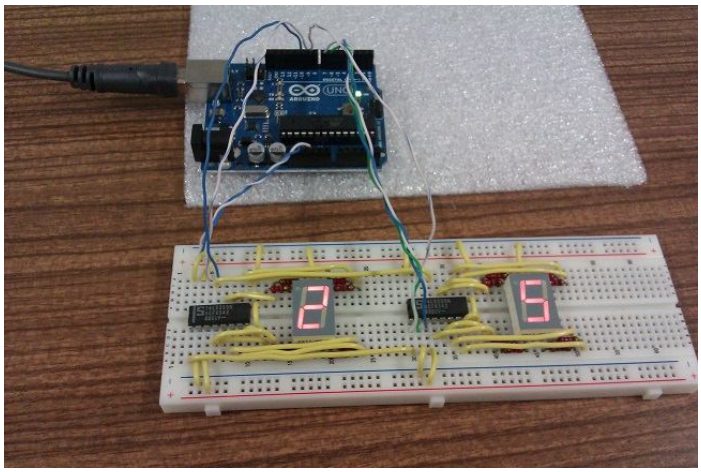
```
int displays[]={8,9};
int entradas[]={7,6,5};
int segmentos[]={3,4,2,6,1,0,7,5};
int led=10;

void setup()
{

        for(int i=0;i<3;i++)

                pinMode(entradas[i],OUTPUT);

        for(int i=0;i<2;i++) {

                pinMode(displays[i],OUTPUT);
```

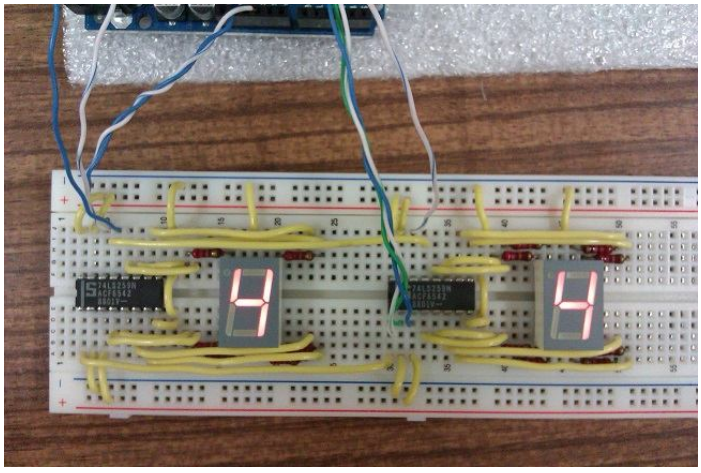
```
            digitalWrite(displays[i],HIGH);

        }
        pinMode(led,OUTPUT);

}

void OutputDisplay(int numero, int local)
{

        int digitos[10][8]={

                {HIGH,HIGH,HIGH,HIGH,HIGH,HIGH,LOW,LOW}, // 0
                {LOW,HIGH,HIGH,LOW,LOW,LOW,LOW,LOW}, // 1
                {HIGH,HIGH,LOW,HIGH,HIGH,LOW,HIGH,LOW}, // 2
                {HIGH,HIGH,HIGH,HIGH,LOW,LOW,HIGH,LOW}, // 3
                {LOW,HIGH,HIGH,LOW,LOW,HIGH,HIGH,LOW}, // 4
                {HIGH,LOW,HIGH,HIGH,LOW,HIGH,HIGH,LOW}, // 5
                {HIGH,LOW,HIGH,HIGH,HIGH,HIGH,HIGH,LOW}, // 6
                {HIGH,HIGH,HIGH,LOW,LOW,LOW,LOW,LOW}, // 7
                {HIGH,HIGH,HIGH,HIGH,HIGH,HIGH,HIGH,LOW}, // 8
                {HIGH,HIGH,HIGH,HIGH,LOW,HIGH,HIGH,LOW}, // 9

        };
        // colocar todos os 8 bits
        for(int i=0;i<8;i++)
        {

                // colocar endereço do bit
                for(int j=0, mascara=1;j<3;j++,mascara*=2)

                        if(i&mascara) digitalWrite(entradas[j],HIGH);
                        else digitalWrite(entradas[j],LOW);

                // colocar valor do led
                digitalWrite(led,digitos[numero%10][segmentos[i]]);
                // gravar
                digitalWrite(displays[local],LOW);
                delay(10);
                digitalWrite(displays[local],HIGH);

        }

}

void loop()
{

        int segundos=millis()/1000;
        OutputDisplay(segundos%10, 0);
        OutputDisplay((segundos/10)%10, 1);
```

}

A diferença está na função de OutputDisplay, que em vez de enviar um bit para um pin, tem de comunicar com o circuito integrado. Para cada um dos 8 bits no registo, começa por colocar nos pins de endereço (ABC) o endereço do bit, e no pin D coloca o valor a colocar no LED. De seguida baixa o pin E, do display em que se está a actualizar, para que o bit fique gravado, e volta a levantá-lo. O outro circuito integrado pode ler também os valores de ABC e de D, mas como o valor de E não foi alterado, mantém o registo intacto. A espera de 10 milissegundos é talvez exagerada, convém verificar na documentação do circuito quanto tempo é necessário para garantir o bom funcionamento.

Em baixo estão fotografias da implementação:

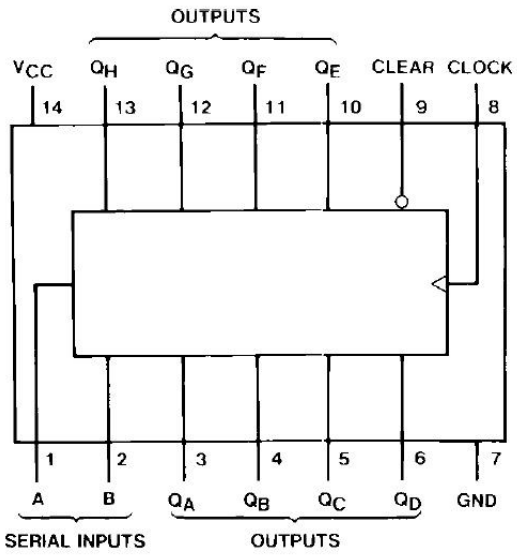




Circuito Integrado 74LS164

A utilização do circuito integrado 74LS164, é mais simples já que contém exactamente o que se pretende do circuito integrado, isto é, entrada em série e saída em paralelo:

Connection Diagram

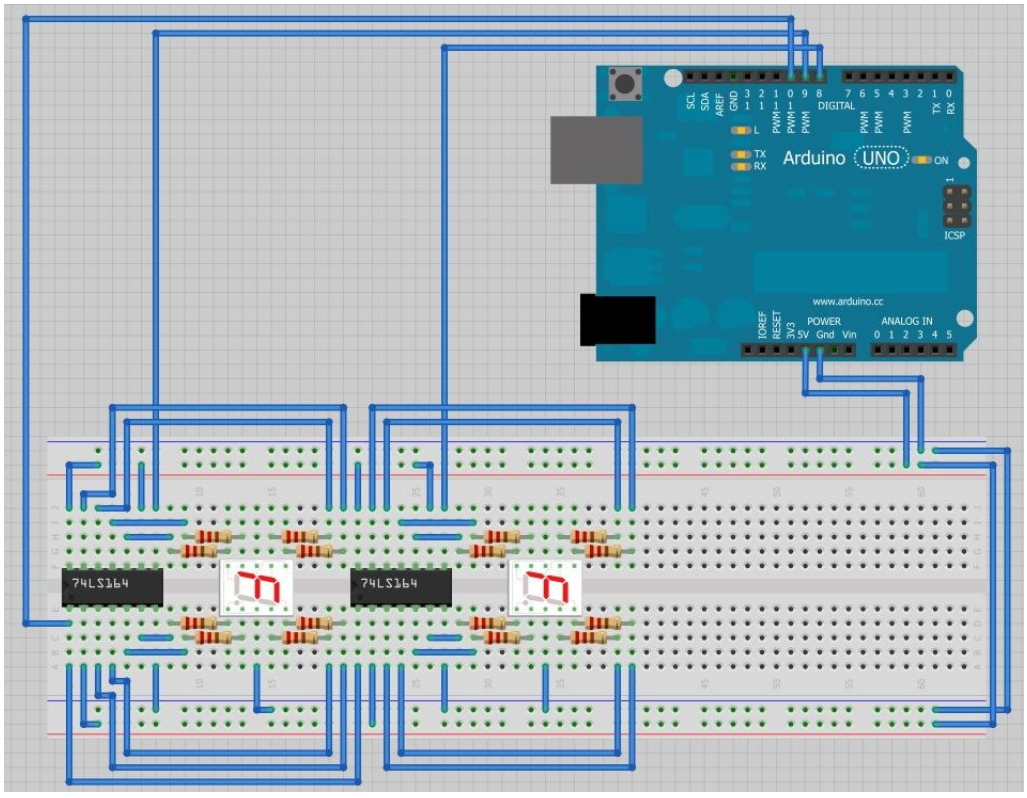


Function Table

Inputs				Outputs			
Clear	Clock	A	B	Q_A	Q_B	...	Q_H
L	X	X	X	L	L	...	L
H	L	X	X	Q_{A0}	Q_{B0}	...	Q_{H0}
H	↑	H	H	H	Q_{An}	...	Q_{Gn}
H	↑	L	X	L	Q_{An}	...	Q_{Gn}
H	↑	X	L	L	Q_{An}	...	Q_{Gn}

H = HIGH Level (steady state)
L = LOW Level (steady state)
X = Don't Care (any input, including transitions)
↑ = Transition from LOW-to-HIGH level
Q_{A0}, Q_{B0}, Q_{H0} = The level of Q_A, Q_B, or Q_H, respectively, before the indicated steady-state input conditions were established.
Q_{An}, Q_{Gn} = The level of Q_A or Q_G before the most recent ↑ transition of the clock; indicates a one-bit shift.

Basicamente existem duas entradas em série, com uma conjunção. Precisamos apenas de um bit, pelo que a outra entrada liga-se ao positivo, e não necessitando do CLEAR, liga-se também ao positivo. Os dados são carregados com a mudança da entrada CLOCK de LOW para HIGH, o que e facilmente executado no programa do Arduino. Apenas estes dois pins necessitam de estar ligados ao Arduino, e a entrada em série pode ser comum com outros circuitos integrados, tal como no exemplo anterior. Assim, o número de pins necessários para dois registos é 3, para quatro registos são necessários 5 pins (no caso de se utilizar muitos registos, para não estar a gastar um pin por cada registo, seria até possível utilizar-se o circuito integrado em cima no modo de descodificador, e assim com 4 registos seriam necessários 1+2 pins (2^2=4), com 8 registos seriam necessários 1+3 pins (2^3=8), e assim sucessivamente.



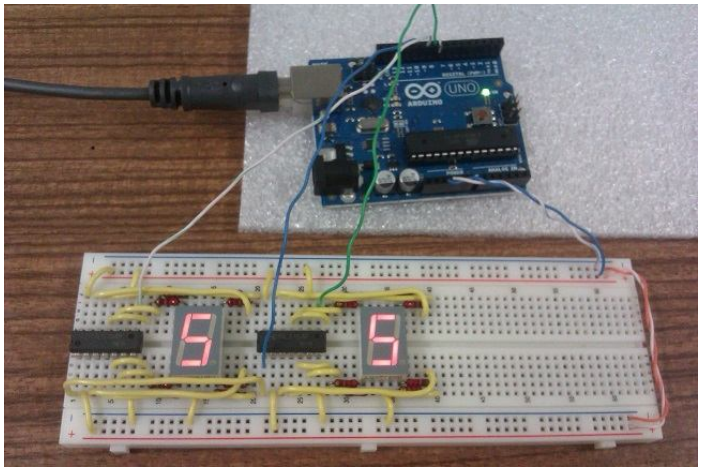
No programa, a função OutputDisplay fica mais simplificada:

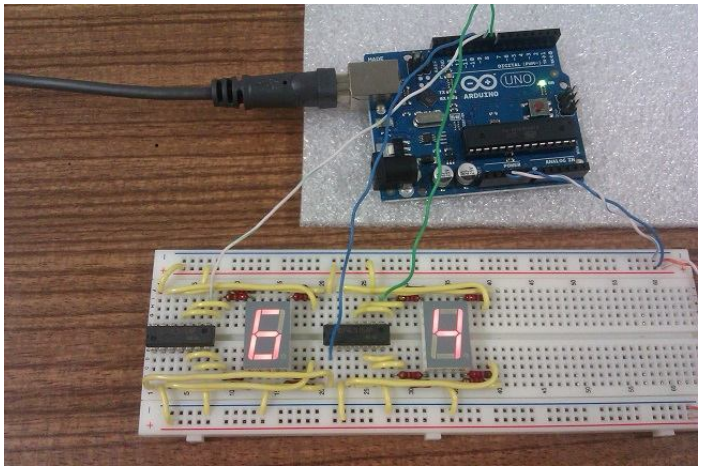
```
int displays[]={8,9};
```

```
int segmentos[]={0,1,6,5,3,4,7,2};
int led=10;

void setup()
{

	for(int i=0;i<2;i++) {

		pinMode(displays[i],OUTPUT);
		digitalWrite(displays[i],HIGH);

	}
	pinMode(led,OUTPUT);

}

void OutputDisplay(int numero, int local)
{

	static int valores[2]={-1,-1};

	int digitos[10][8]={

		{HIGH,HIGH,HIGH,HIGH,HIGH,HIGH,LOW,LOW}, // 0
		{LOW,HIGH,HIGH,LOW,LOW,LOW,LOW,LOW}, // 1
		{HIGH,HIGH,LOW,HIGH,HIGH,LOW,HIGH,LOW}, // 2
		{HIGH,HIGH,HIGH,HIGH,LOW,LOW,HIGH,LOW}, // 3
		{LOW,HIGH,HIGH,LOW,LOW,HIGH,HIGH,LOW}, // 4
		{HIGH,LOW,HIGH,HIGH,LOW,HIGH,HIGH,LOW}, // 5
		{HIGH,LOW,HIGH,HIGH,HIGH,HIGH,HIGH,LOW}, // 6
		{HIGH,HIGH,HIGH,LOW,LOW,LOW,LOW,LOW}, // 7
		{HIGH,HIGH,HIGH,HIGH,HIGH,HIGH,HIGH,LOW}, // 8
		{HIGH,HIGH,HIGH,HIGH,LOW,HIGH,HIGH,LOW}, // 9

	};

	// verificar se o valor no display é distinto

	if(valores[local]!=numero%10) {

		// colocar todos os 8 bits
		for(int i=0;i<8;i++)
		{

			// colocar valor do led
			digitalWrite(led,digitos[numero%10][segmentos[i]]);
			// gravar
			digitalWrite(displays[local],LOW);
			digitalWrite(displays[local],HIGH);

		}
```

```
            valores[local]=numero%10;

        }

}

void loop()
{

        int segundos=millis()/1000;
        OutputDisplay(segundos, 0);
        OutputDisplay(segundos/10, 1);

}
```

A tarefa da função OutputDisplay, é de apenas enviar os bits todos, sequencialmente. Daí baixar e levatar o valor do CLOCK do integrado, para carregar cada um dos 8 bits. Segue-se fotografias da implementação:





Este exemplo pretende também demonstrar a facilidade com que no Arduino se pode utilizar circuitos integrados. O Arduino pode inclusive ser utilizado para teste do bom funcionamento e velocidade do circuitos integrados, se se fizer o mapeando todas as entradas e saídas do circuito integrado para pins do Arduino. Basicamente permite levar a programação à electrónica, facilitando assim a sua utilização, mas não convém deixar de se chamar à atenção que programas mais pesados/complexos, devem correr num computador, e não no Arduino, podendo este comunicar com o computador. Esse é o tema da ficha 3.

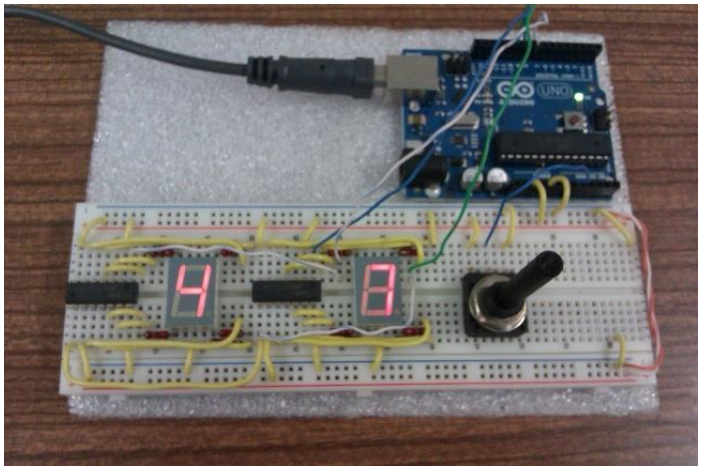
11 Como estabilizar um sinal analógico instável?

Por vezes, a precisão de leitura analógica é superior à precisão do material, gerando oscilações que podem ser complicadas de lidar. Tal ocorre por exemplo com a leitura de um valor de um potenciómetro. No exemplo da leitura com o potenciómetro, acenderam-se apenas 4 LEDs, mas se se colocar o valor do potenciómetro para uma escala de 0 a 99, utilizando por exemplo os displays de 7 segmentos da página anterior, a precisão poderá não ser suficiente para selecionar qualquer valor. Um potenciómetro normalmente tem um ângulo de rotação reduzido, ao qual se adiciona atrito variável do material e incerteza na mão que opera com o potenciómetro. Outros sensores têm outras incertezas associadas, pelo que por vezes torna-se necessário estabilizar o sinal.

Primeiro temos de confirmar o problema, a leitura instável, para depois apresentar a solução. Relativamente ao programa anterior, que mostra o número de segundos em dois displays de 7 segmentos, há que trocar apenas a função loop:

```
void loop()
{

	int valor=analogRead(A0);
	OutputDisplay(map(valor,0,1024,0,100), 0);
	OutputDisplay(map(valor,0,1024,0,100)/10, 1);

}
```

Em vez de se colocar nos displays o número de segundos, coloca-se o valor da leitura A0, após re-escalada para o intervalo de 0 a 99. Após adicionar um potenciómetro, colocando a saída do potenciómetro na entrada A0, podemos obter situações deste tipo:



Repare-se que o 7 está a alternar entre 7 e 8. Para evitar este efeito, tem que se conseguir evitar a instabilidade. Pode-se evitar a instabilidade tanto através de uma média móvel (utilizar o valor médio das últimas K leituras), como através do alisamento exponencial (utilizar uma média ponderada entre o valor atual e o valor anterior). O exemplo seguinte, utiliza o alisamento exponencial:

```
void loop()
{
```

```
    static int anterior=0;
    int valor=analogRead(A0);
    valor=0.5*valor+(1-0.5)*anterior;
    anterior=valor;
    OutputDisplay(map(valor,0,1024,0,100), 0);
    OutputDisplay(map(valor,0,1024,0,100)/10, 1);

}
```

O valor 0,5 é a importância relativa do valor atual, relativamente ao valor anterior. Pode-se agora fazer a experiência, comentando ou descomentado as duas linhas que implementam o alisamento exponencial, ou baixando o valor de 0.5 para 0.25, e tentar reproduzir com o alisamento, situações em que o número visualizado alterna entre dois valores. Será muito mais difícil de reproduzir, confirmando-se que esta técnica dá estabilidade ao valor lido.

Para utilizar a média móvel, tem que se definir o número de elementos a colocar na média. Vamos utilizar 8:

```
void loop()
{

    static int anteriores[8]={0,0,0,0,0,0,0,0};
    int valor=analogRead(A0);
    // discartar o último valor e empurrar os restantes
    for(int i=0;i<7;i++)

        anteriores[i]=anteriores[i+1];

    anteriores[7]=valor;
    // colocar em valor, a média de todos os valores
    valor=0;
    for(int i=0;i<8;i++)

        valor+=anteriores[i];

    valor/=8;
    OutputDisplay(map(valor,0,1024,0,100), 0);
    OutputDisplay(map(valor,0,1024,0,100)/10, 1);

}
```

O código é mais complexo, sendo o efeito idêntico, pelo que aconselha-se a utilizar o alisamento exponencial, ficando no entanto este método como uma alternativa.

Notar que a estabilidade do sinal, faz-se na variável de maior precisão, neste caso a entrada analógica de 10 bits, portanto um valor entre 0 e 1023. Poder-se-ia fazer a estabilização no valor entre 0 e 99, mas assim a informação do(s) valor(es) anterior(es) era menor.

12 Como gravar e reproduzir um sinal ao longo do tempo?

Uma operação que poderá ser vulgar, é a gravação e reprodução de um sinal, digital ou analógico, ao longo do tempo. Caso não exista capacidade de reprodução de um sinal de forma temporizada, a instalação apenas irá reagir a agitação dos sensores, o que poderá ser suficiente para muitas instalações, mas para outras revelar-se insuficiente.

Vamos continuar a utilizar o potenciómetro e os dois displays de 7 segmentos, para ler os valores introduzidos. Após 1 segundo de paragem, a sequência introduzida é reproduzida nos displays. Numa primeira experiência, guardam-se apenas os números inteiros de 0 a 99, e reproduzem-se utilizando 1 segundo para cada número. Numa segunda experiência, iremos guardar toda a informação introduzida, e fazer uma reprodução exata.

```
// variáveis de gravação
int valoresGravados[100], numeroValores=0, tamanho=100;
long ultimoInstante=0, inatividade=1000; // 1 segundo
// gravar o valor lido, se distinto, retornando o último instante de tempo em que foi gravado um valor
void Gravar(int valor)
{

	// verificar se o valor é distinto do anterior, c.c. ignorar
	if(numeroValores==0 || valoresGravados[numeroValores-1]!=valor) {

		// verificar se é altura de se fazer um reset, no caso de ter existido uma parágem longa
		if(millis()-ultimoInstante>inatividade)

			numeroValores=0;

		// carregar até ao limite de memória
		if(numeroValores<tamanho)

			valoresGravados[numeroValores++]=valor;

		else {

			// descartar o valor mais antigo
			for(int i=0;i<tamanho-1;i++)

				valoresGravados[i]=valoresGravados[i+1];

			valoresGravados[tamanho-1]=valor;

		}
		// último instante em que houve alteração
		ultimoInstante=millis();

	}

}
```

```
// reproduzir o valor, que retornará o valor correpondente ao instante pedido
int Reproduzir(long instante)
{

	// uma nota por segundo
	long indice=instante/1000;
	if(indice<numeroValores)

		return valoresGravados[indice];

	// sequência já acabou, retornar 0 ou um código que indique que a sequência já acabou
	return 0;


}

void loop()
{

	static int anterior;
	int valorBruto=analogRead(A0);
	int valor;
	valorBruto=0.5*valorBruto+(1-0.5)*anterior;
	anterior=valorBruto;
	valor=map(valorBruto,0,1024,0,100);
	Gravar(valor);

	if(millis()-ultimoInstante>inatividade) // 1 segundo de inactividade

		valor=Reproduzir(millis()-ultimoInstante-inatividade); // reproduzir, indicando o instante
		de tempo

	// a reprodução é nos displays de 7 segmentos, mas poderia ser num LED
	OutputDisplay(valor, 0);
	OutputDisplay(valor/10, 1);

}
```

Ao código do alisamento exponencial, adicionam-se as duas funções Gravar e Reproduzir, juntamente com as variáveis globais de gravação. Naturalmente que se houver dois sinais a serem gravados, ter-se-ia de adicionar argumentos às funções Gravar e Reproduzir, com o número do sinal. A função gravar, grava o valor no caso deste ser distinto, e atualiza a variável ultimoInstante. No caso de inatividade, momento para iniciar a reprodução, chama-se a função Reproduzir com o instante em que se está na reprodução, sendo devolvido o valor do sinal nessa altura, valor esse que deve ser mostrado, este caso através dos displays de 7 segmentos, mas poderia ser outra qualquer utilização. A gravação vai guardando os valores num vetor, e tem a atenção de descartar os valores mais antigos, quando o tamanho do vetor é atingido. A função reprodução, simplesmente calcula o segundo atual, e devolve o valor no vetor.

Vamos agora guardar a informação completa, isto é, para além o valor na precisão maior, guardamos também o instante de tempo em que o valor foi colocado. Desta forma, a reprodução do sinal

registado, poderá ser completa. A diferença relativamente ao código anterior, não é tão grande como previsto, basicamente tem que se adicionar um vetor com os instantes de tempo de gravação:

```
// variáveis de gravação
int valoresGravados[100], numeroValores=0, tamanho=100;
long instantesGravados[100];
long ultimoInstante=0, inatividade=1000; // 1 segundo
// gravar o valor lido, se distinto, retornando o último instante de tempo em que foi gravado um valor
void Gravar(int valor)
{

	// verificar se o valor é distinto do anterior, c.c. ignorar
	if(numeroValores==0 || valoresGravados[numeroValores-1]!=valor) {

		// verificar se é altura de se fazer um reset, no caso de ter existido uma parágem longa
		if(millis()-ultimoInstante>inatividade)

			numeroValores=0;

		// carregar até ao limite de memória
		if(numeroValores<tamanho) {

			valoresGravados[numeroValores]=valor;
			// gravar também o instante
			instantesGravados[numeroValores]=millis();
			numeroValores++;

		} else {

			// descartar o valor mais antigo
			for(int i=0;i<tamanho-1;i++) {

				valoresGravados[i]=valoresGravados[i+1];
				instantesGravados[i]=instantesGravados[i+1];

			}
			valoresGravados[tamanho-1]=valor;
			instantesGravados[tamanho-1]=millis();

		}
		// último instante em que houve alteração
		ultimoInstante=millis();

	}

}

// reproduzir o valor, que retornará o valor correpondente ao instante pedido
int Reproduzir(long instante)
{

	// obter o índice do valor correspondente ao instante pedido (subtrair pelo primeiro)
```

```
    for(long indice=1;indice<numeroValores;indice++)

        if(instantesGravados[indice]-instantesGravados[0]>instante)

            // devolver o valor de indice-1, já que o instante de indice já é posterior
            return valoresGravados[indice-1];

    // retornar a última nota durante o tempo de inactividade
    if(numeroValores>0 && instantesGravados[numeroValores-1]+inatividade-
    instantesGravados[0]>instante)

        return valoresGravados[numeroValores-1];

    // sequência já acabou, retornar 0 ou um código que indique que a sequência já acabou
    return 0;

}
```

A função loop() permanece sem alterações. Pode-se quanto muito trocar o sinal gravado para o valor antes de ser mapeado de 0 a 99, mas como estamos a ver apenas a esta precisão, mantém-se o código na função loop().

Pode-se ver que ao introduzir uma sequência, esta passa com a mesma velocidade que foi introduzida. Poderiam estar várias sequências a ser reproduzidas sem simultâneo, sem problema, mas se a frequência baixar, devido a utilização da função delay() com valor elevado ou por existir um processamento muito pesado, algumas notas podem ser "saltadas", não sendo reproduzidas.