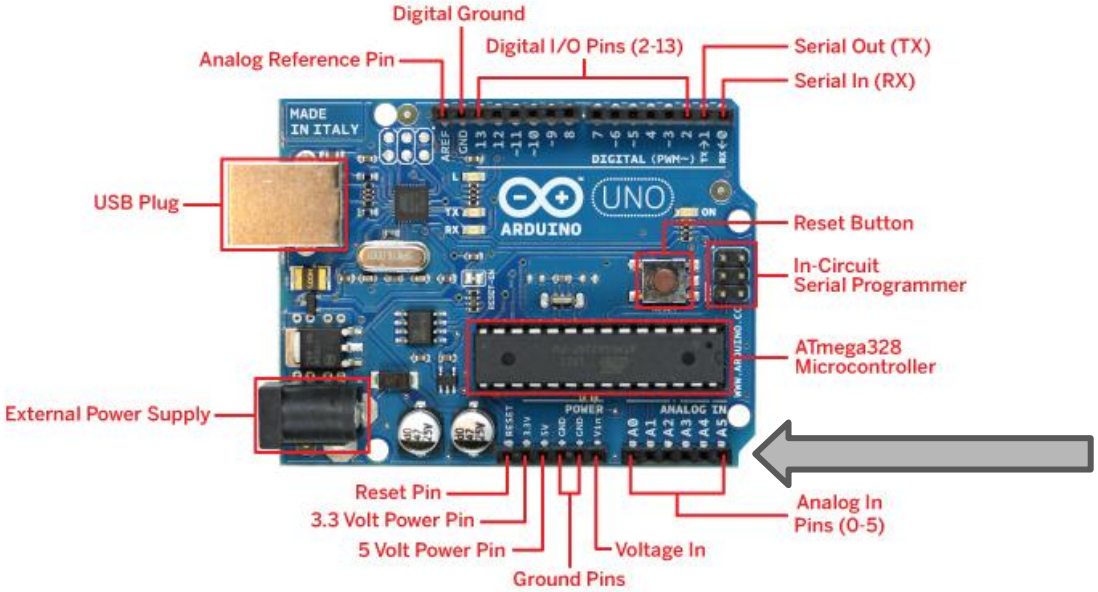
Portas e protocolos de comunicação

Sistemas embarcados

Prof. Allan Rodrigo Leite

Portas analógicas

- O Arduino também tem entradas dedicadas para dados analógicos
  - Além das portas digitais de entrada



Portas analógicas

- A entrada analógica pode receber sinais entre 0v e 5v
  - Esta entrada possui resolução de 10 bits

- Assim, os sinais são convertidos para um valor inteiro entre 0 e 1023
  - Essa conversão é chamada de ADC (*Analogic to Digital Converter*)
  - Exemplo: 2,5v será convertido para 511

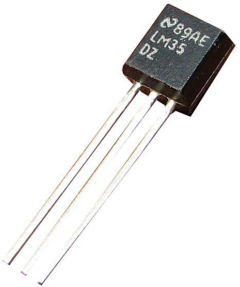
- É aconselhável sempre consultar o datasheet do componente utilizado
  - Alguns fornecem automaticamente a tensão
    - Exemplo: potenciômetros
  - Alguns precisam de resistores para funcionar como um divisor resistivo
    - Exemplo: fotosensores

Portas analógicas

- Exemplo de sensores analógicos

Fotoresistor (LDR)

Potenciômetros



Sensor de temperatura

Portas analógicas

- Exemplo de sensores analógicos (cont.)

Portas analógicas

- A configuração das portas analógicas é similar as portas digitais
  - Também é utilizada a função `pinMode(<porta>,<tipo>)`
    - No entanto, as portas analógicas são somente de entrada
  - Além disso, a identificação das portas analógicas é feita por constantes
    - `A0` representa a porta analógica 0
    - `A1` representa a porta analógica 1
    - ...

```
void setup() {
    pinMode(A0, INPUT);
}
```

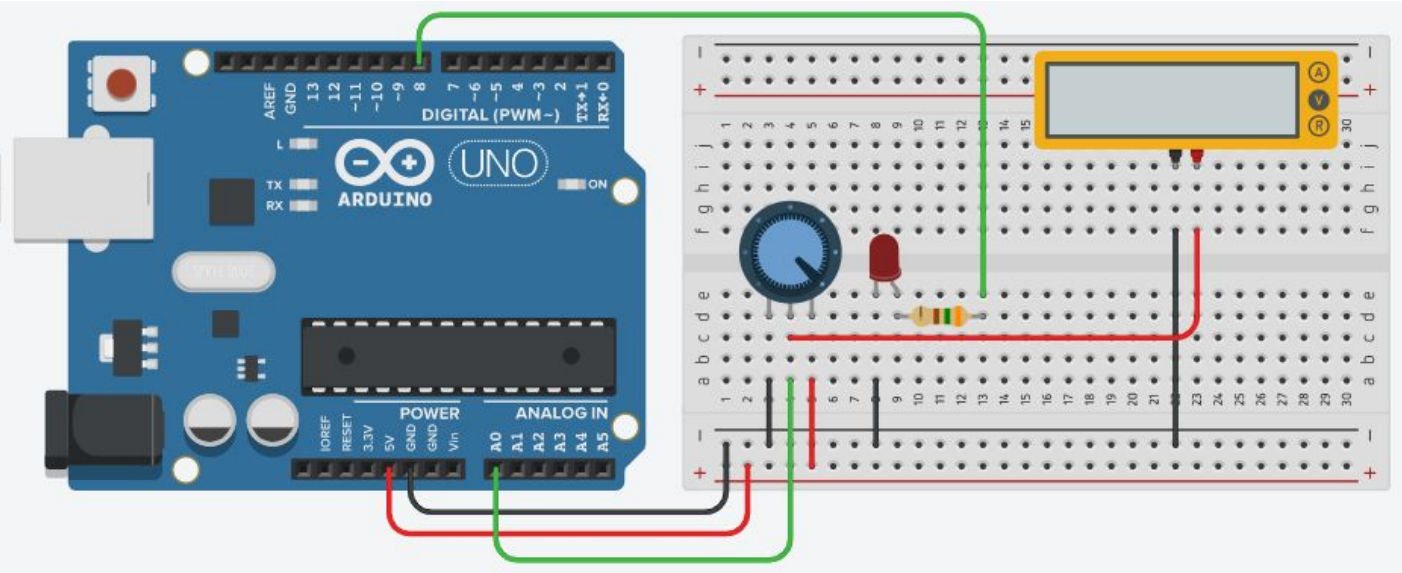
Portas analógicas

- A leitura da porta é realizada pela função `analogRead(<porta>)`
  - Lembrando que o retorno será um valor inteiro entre 0 e 1023

```
void setup() {
    pinMode(A0, INPUT);
}

void loop() {
    int leitura = analogRead(A0); //Retorna um valor inteiro entre 0 e 1023
}
```

Portas analógicas

- Exemplo de uso do potenciômetro para entrada de dados



Portas analógicas

- Exemplo de uso do potenciômetro para entrada de dados (cont.)

```
int pinoLed = 8;
int pinoPotenciometro = A0;

int maximo = 1000; //valor maximo do intervalo de leitura do potenciômetro
int minimo = 200;  //valor mínimo do intervalo de leitura do potenciômetro

void setup() {
    pinMode(pinoLed, OUTPUT);
    pinMode(pinoPotenciometro, INPUT);
}
```

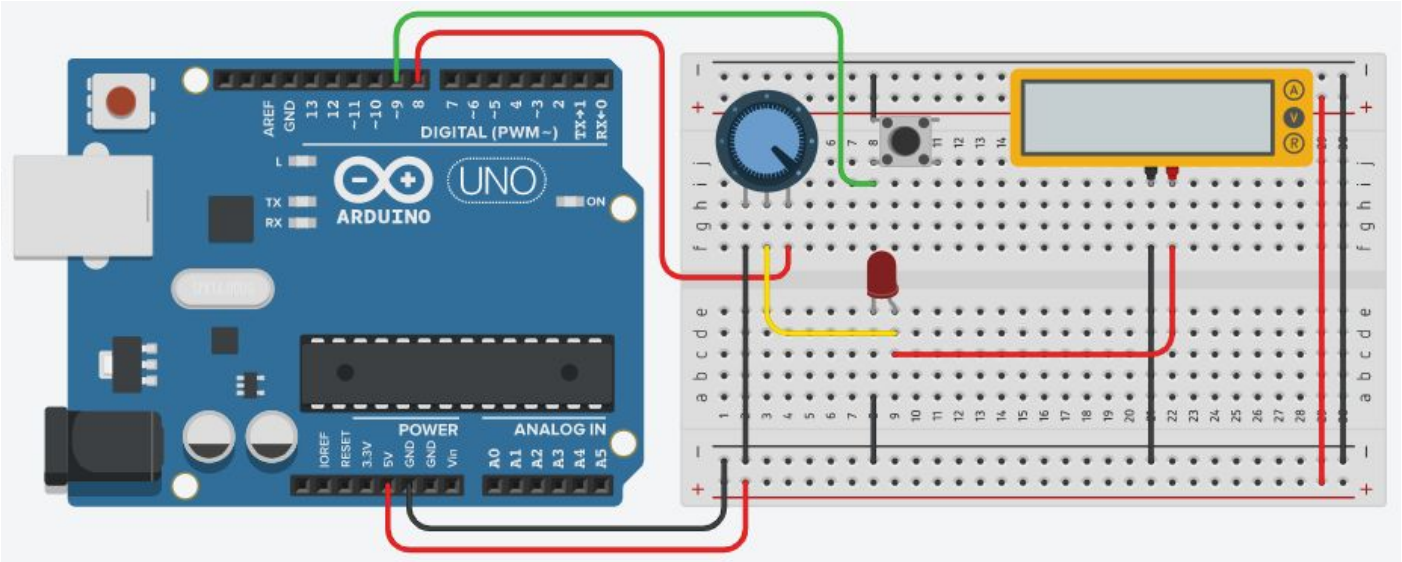
Portas analógicas

- Exemplo de uso do potenciômetro para entrada de dados (cont.)

```
void loop() {
    //realiza a leitura na porta analógica
    float leitura = analogRead(pinoPotenciometro);

    //normaliza a leitura para o valor mínimo e maximo definido
    leitura = maximo - ((float) (maximo - minimo) / 1023) * leitura;

    digitalWrite(pinoLed, HIGH);
    delay(leitura);

    digitalWrite(pinoLed, LOW);
    delay(leitura);
}
```

Portas analógicas

- Exemplo de uso do potenciômetro como resistor para um LED



Portas analógicas

- Exemplo de uso do potenciômetro como resistor para um LED (cont.)

```
int pinoBotao = 9;
int pinoLed = 8;

int ultimoEstado = HIGH;
bool ligado = false;

void setup() {
    pinMode(pinoBotao, INPUT_PULLUP);
    pinMode(pinoLed, OUTPUT);
}
```

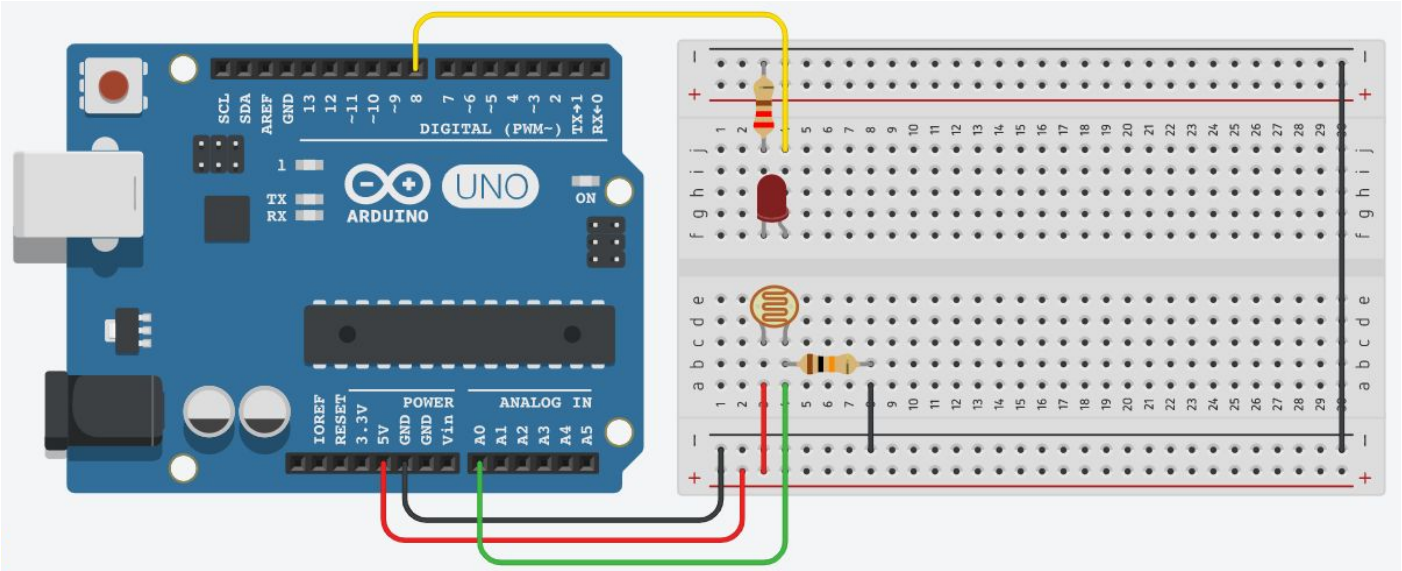
Portas analógicas

- Exemplo de uso do potenciômetro como resistor para um LED (cont.)

```
void loop() {
    int estado = digitalRead(pinoBotao);

    if (ultimoEstado != estado && estado == HIGH) {
        ligado = !ligado;
    }

    ultimoEstado = estado;

    digitalWrite(pinoLed, ligado ? HIGH : LOW);
}
```

Portas analógicas

- Exemplo de uso do fotosensor



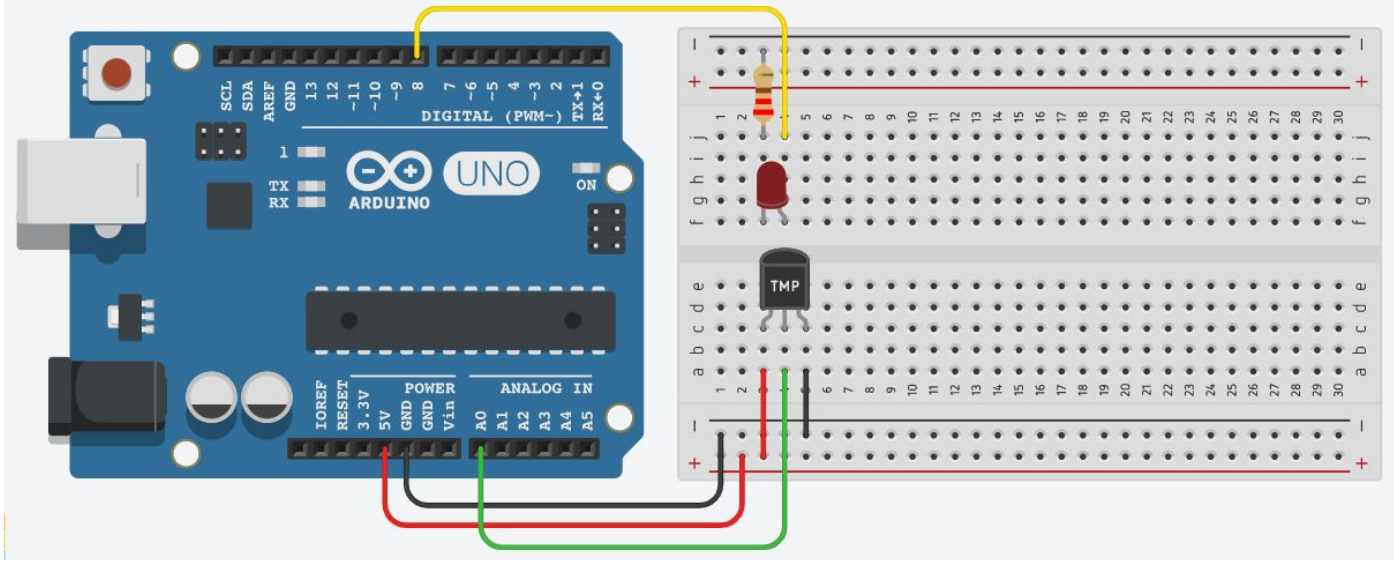
Portas analógicas

- Exemplo de uso do fotosensor (cont.)

```
int pinoLed = 8;
int pinoFotosensor = A0;

void setup() {
    pinMode(pinoLed, OUTPUT);
    pinMode(pinoFotosensor, INPUT);
}

void loop() {
    int intensidade = analogRead(pinoFotosensor);

    digitalWrite(pinoLed, intensidade > 900 ? HIGH : LOW);
    delay(100);
}
```

Portas analógicas

- Exemplo de uso do sensor de temperatura



Portas analógicas

- Exemplo de uso do sensor de temperatura (cont.)

```
int pinoLed = 8;
int pinoTemperatura = A0;

void setup() {
    pinMode(pinoLed, OUTPUT);
    pinMode(pinoTemperatura, INPUT);
}
```

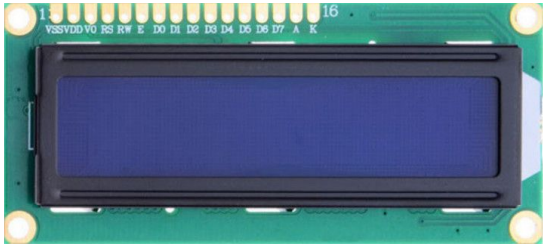
Portas analógicas

- Exemplo de uso do sensor de temperatura (cont.)

```
void loop() {
    int leitura = analogRead(pinoTemperatura);

    //converte o valor da leitura em 10 bit (0 - 1023)
    float bit = (float) (leitura * 5) / 1024;

    //converte o valor em 10 bit para celsius
    float temperatura = ((bit * 1000) - 500) / 10;

    digitalWrite(pinoLed, temperatura > 30 ? HIGH : LOW);
    delay(1000);
}
```

Display LCD

- Os displays LCD padrão possuem
  - 16 colunas e 2 linhas
  - Luz de fundo azul
  - Letras na cor branca



- Para conectá-lo, usa-se 12 pinos para uma conexão básica
  - Considerando conexões de alimentação (pinos 1 e 2)
  - Luz de fundo (pinos 15 e 16)
  - Contraste (pino 3)

Display LCD

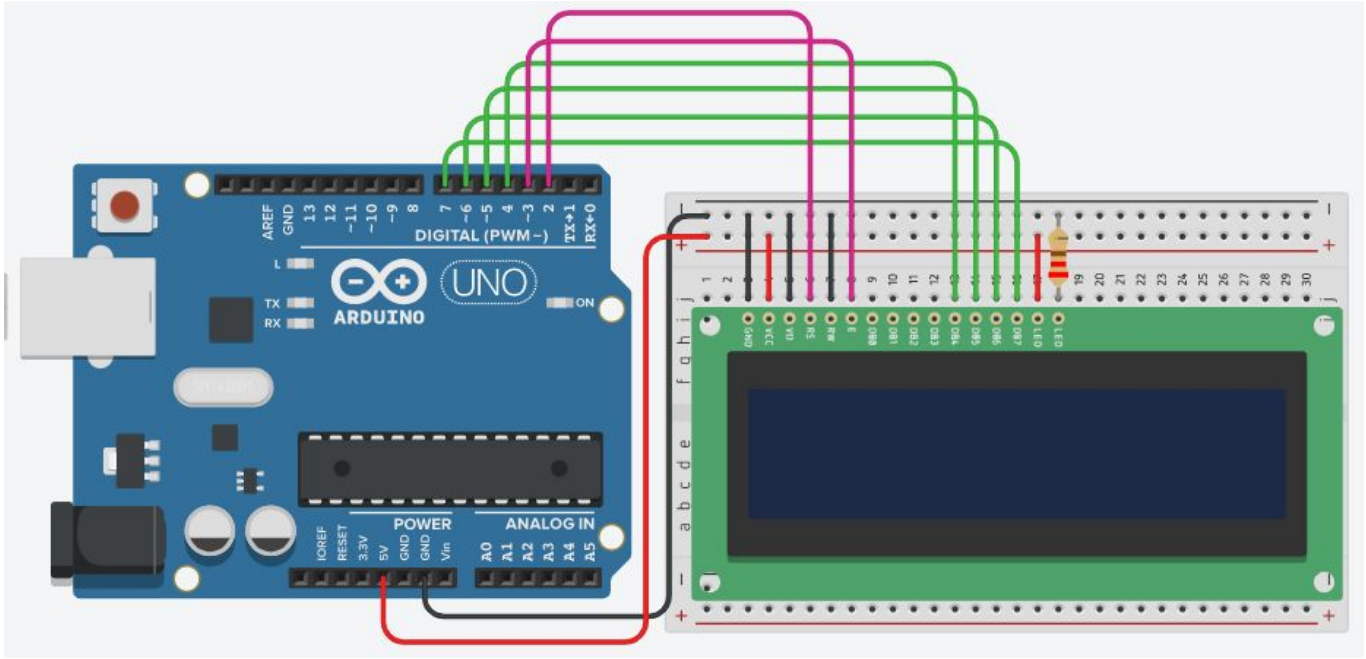
- Recomenda-se manipular display pela biblioteca `LiquidCrystal`
  - Esta biblioteca já consta por padrão na IDE do Arduino

```
LiquidCrystal lcd(<register select>, <pino enable>, <D4>, <D5>, <D6>, <D7>)
```

- As principais funções da biblioteca são
  - `ldc.begin(16,2)`: define o número de linhas e colunas
  - `lcd.clear()`: limpa a tela
  - `lcd.setCursor(<coluna>,<linha>)`: posiciona o cursor na posição
  - `lcd.print(<texto>)`: exibe o texto informado
  - `lcd.scrollDisplayLeft()`: desloca o texto da tela para esquerda
  - `lcd.scrollDisplayRight()`: desloca o texto da tela para direita

Display LCD

- Exemplo de uso do display LCD



Display LCD

- Exemplo de uso do display LCD (cont.)

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(2,3,4,5,6,7);

void setup() {
    lcd.begin(16,2);

    lcd.print("Ola");

    lcd.setCursor(0,1);
    lcd.print("Mundo!");
}
```

Display LCD

- Exemplo de uso do display LCD (cont.)

```
void loop() {
    for (int i = 0; i < 6; i++) {
        lcd.scrollDisplayLeft();
        delay(300);
    }

    for (int i = 0; i < 6; i++) {
        lcd.scrollDisplayRight();
        delay(300);
    }
}
```

Exercícios

1. Crie um *sketch* no Arduino para controlar o tom de um buzzer a partir de um potenciômetro. O projeto deve conter um buzzer, um botão para acioná-lo e um potenciômetro para configurar a frequência do som.

2. Crie um *sketch* no Arduino para configurar uma temperatura utilizando um potenciômetro. Em seguida, um LED deve ser mantido aceso enquanto a temperatura medida pelo sensor de temperatura estiver inferior à temperatura informada. A temperatura capturada pelo sensor deve ser exibida em grau Celsius em um LCD a cada 1s.

3. Crie um *sketch* no Arduino para exibir a intensidade de iluminação e a temperatura (em grau Celsius) em um display LCD. No entanto, as informações devem ser exibidas uma por vez. Para isto, deve ser utilizado um botão para alternar entre as informações exibidas no LCD.

Saídas analógicas

- O Arduino UNO não possui saídas analógicas verdadeiras
  - Apenas o Arduino Due tem essa opção

- Como é possível enviar uma tensão entre 0v e 5v?
  - Para isto é necessário ligar e desligar uma saída digital rapidamente
    - Este processo é chamado de pulso

  - Assim, reproduz-se uma saída analógica ao controlar a largura desse pulso
    - Esta técnica é chamada de modulação de largura de pulso PWM
    - PWM (*Pulse-Width Modulation*)

Modulação de largura de pulso

- Algumas aplicações do PWM
  - Controle de motores
  - Controle de *encoders* e *decoders*
  - Telecomunicações
    - Como uma das formas de modulação de sinal
  - Alimentação de tensão
  - Efeitos de áudio
    - Sintetizadores simples
  - Controlar a intensidade de um LED
  - Controlar o aquecimento de uma resistência

Modulação de largura de pulso

- Exemplo de uso do PWM com motores
  - Um resistor pode ser utilizado para controlar a velocidade de um motor
    - E se for necessário alterar a velocidade durante a execução do programa?
    - Além disto, se o motor for grande, o resistor pode ficar muito quente
    - E calor é desperdício de energia
  - Neste caso, o uso do PWM é uma alternativa eficiente
    - Ligar e desligar a energia em uma frequência controlada é mais efetivo
    - A frequência pode ser controlada por um microcontrolador

Modulação de largura de pulso

- Exemplo de uso do PWM com motores (cont.)
  - A modulação de pulso possui dois ciclos de trabalho
    - Ativo: quando o sinal está ativo, também chamado de *duty cycle*
    - Desligado: quando o sinal está inativo
  - O *duty cycle* é expresso em percentual de tempo

  - Voltando ao exemplo, para controlar um motor de 12v a uma velocidade de 60% da sua capacidade, aplica-se um *duty cycle* de 60%
    - Isto significa que a energia estará ligada 60% do tempo
    - Ou seja, o resultado é igual ao motor ligado com com 7,2v

Modulação de largura de pulso

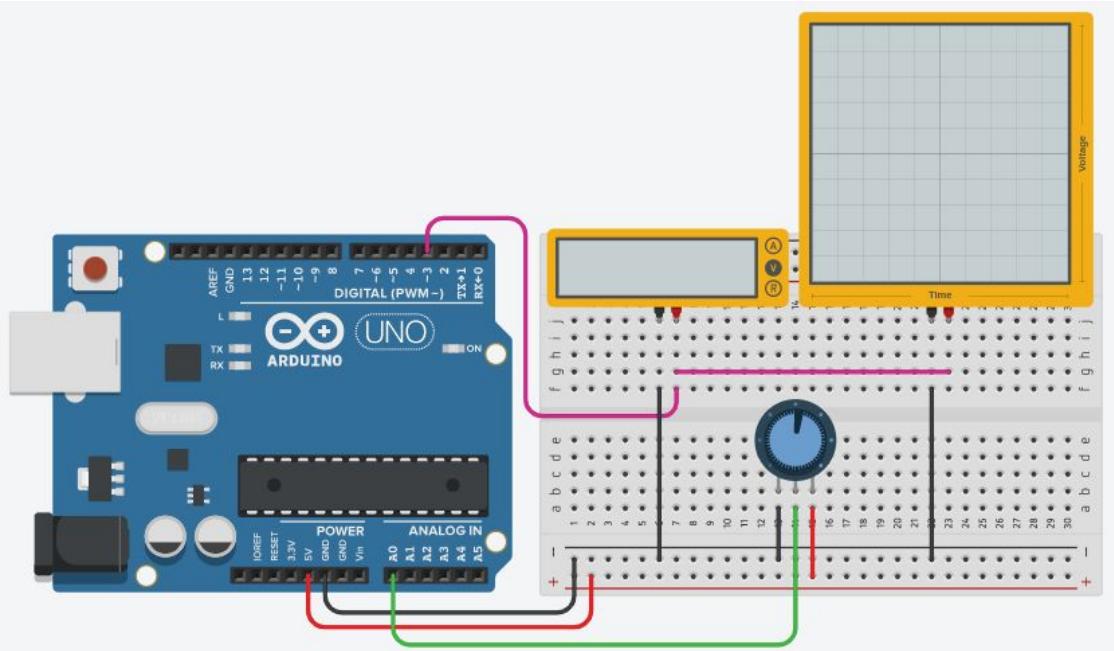
- Função `digitalWrite(<porta>,<valor>)`
  - As portas marcadas com PWM permitem informar valores entre 0 e 255
    - Além dos valores `LOW` e `HIGH`

- Função `map(<valor>, <menor-origem>,<superior-origem>, <inferior-destino>, <superior-destino>)`
  - Facilitar a conversão de valores em escalas diferentes
  - Converão de escala analógica (0 - 1023) para escala digital (0 - 255)

```
int convertido = map(valor, 0, 1023, 0, 255);
```

Modulação de largura de pulso

- Exemplo de uso da modulação de largura de pulso



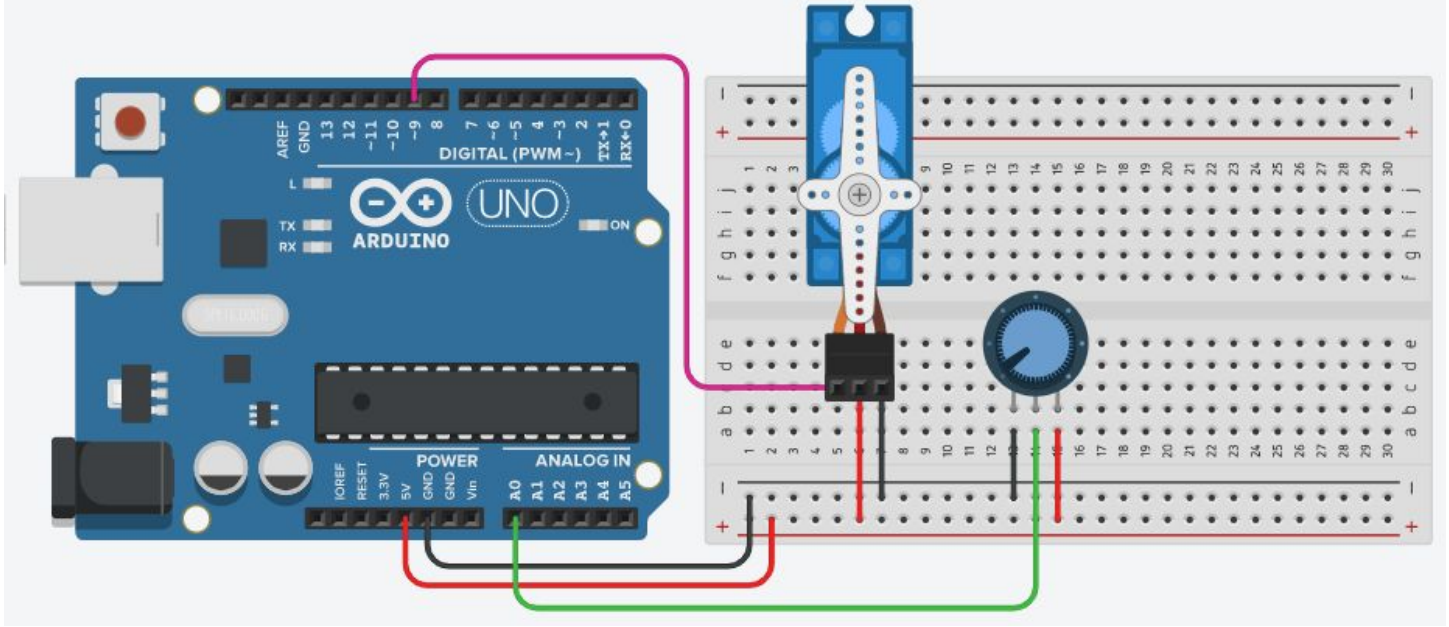
Modulação de largura de pulso

- Exemplo de uso da modulação de largura de pulso (cont.)

```
int pinoPWM = 3;
int pinoPotenciometro = A0;

void setup() {
    pinMode(pinoPotenciometro, INPUT);
    pinMode(pinoPWM, OUTPUT);
}

void loop() {
    int leitura = analogRead(pinoPotenciometro);
    int valor = map(leitura, 0, 1023, 0, 255);

    analogWrite(pinoPWM, valor);
    delay(10);
}
```

Modulação de largura de pulso

- Exemplo de uso do PWD para controle de servo motores



Modulação de largura de pulso

- Exemplo de uso do PWD para controle de servo motores (cont.)

```
#include <Servo.h>

int pinoServo = 9;
int pinoPotenciometro = A0;

Servo servo;

void setup() {
    pinMode(pinoPotenciometro, INPUT);
    servo.attach(pinoServo);
}
```

Modulação de largura de pulso

- Exemplo de uso do PWD para controle de servo motores (cont.)

```
void loop() {
    int leitura = analogRead(pinoPotenciometro);
    int angulo = map(leitura, 0, 1023, 0, 180);

    servo.write(angulo);

    delay(10);
}
```

Exercícios

4. Crie um sketch no Arduino para exibir a distância de um objeto por meio de um sensor de distância. A distância deve ser exibida em um display LCD. O sensor de distância a ser utilizado é o modelo HC-SR04.

5. Crie um *sketch* no Arduino para controlar a garra robótica, permitindo movimentar nos 3 eixos e abrir e fechar a garra usando os joysticks. Além disto, deve ser possível gravar as posições do braço e reproduzi-las ao pressionar um botão.

Interrupções de hardware

- Interrupção é um sinal emitido por um hardware ou software que indica um evento que necessita ser processado imediatamente
  - A requisição de interrupção é chamada de IRQ (*Interruption Request*)

- Ao receber uma interrupção, o processador ou microcontrolador irá
  - Suspender a execução de qualquer código
  - Executa um código específico para tratamento da interrupção
    - Este código é chamado de *interruption handler*
  - Ao encerrar o *handler*, o processador volta a executar o código anterior

Interrupções de hardware

- Exemplos de interrupções de hardware em um computador
  - Pressionar uma tecla no teclado
  - Mover o mouse
  - Placas controladoras de disco
  - Recebimento de pacotes de dados em uma placa de rede

Interrupções de hardware

- No caso de um sistema embarcado, uma interrupção pode detectar a mudança de estado em uma porta

```
int pinoBotao = 11;
int pinoLed = 13;

void loop(){
    if (digitalRead(pinoBotao) == LOW){
        digitalWrite(pinoLed, HIGH);
        delay(5000);

        digitalWrite(pinoLed, LOW);
        delay(5000);
    }
}
```

1) Há algum problema aparente neste código?

2) Quanto tempo leva entre cada verificação de estado do pino 11?

3) O que aconteceria se o botão for pressionado durante este intervalo?

Interrupções de hardware

- A abordagem do exemplo anterior possui algumas ineficiências
  - Requer uma verificação contínua se houve mudança no estado do pino
  - O comportamento da função `loop` é blocante
    - Se a função possuir mais código, não será possível verificar mudanças de estados de outros pinos enquanto o restante do código estiver executando
  - Para sensores com pulsos rápidos, a verificação fica comprometida
    - Ou seja, enquanto o sensor já mudou de estado várias vezes, o código do Arduino pode estar fazendo outra coisa

- Desta forma, as interrupções suspendem a execução do programa e forçam o processador a executar um código específico

Interrupções de hardware

- Tabela de interrupções de portas no Arduino
  - Os números de interrupção mudam de acordo com a versão do Arduino
  - Nem todas as portas dispõem de números de interrupções

Número da Interrupção							
Placa	**0**	**1**	**2**	**3**	**4**	**5**	**Observação**
Uno	D2	D3	-	-	-	-	
Leonardo	D3	D2	D0	D1	-	-	Os números 0 e 1 são invertidos
Mega2560	D2	D3	D21	D20	D19	D18	
Due	Usar o nome dos próprios pinos						

Interrupções de hardware

- Função `attachInterrupt(<número>,<handler>,<tipo>)`
  - Associa um *handler* a uma determinada interrupção pela porta
  - Um *handler* é uma função de *callback*
  - O tipo refere-se à condição de mudança de estado da porta

```
void setup() {
    attachInterrupt(0, trataInterrupcao, CHANGE);
}

void trataInterrupcao() {
    //função invocada sempre que houver uma mudança de estado na porta 0
}
```

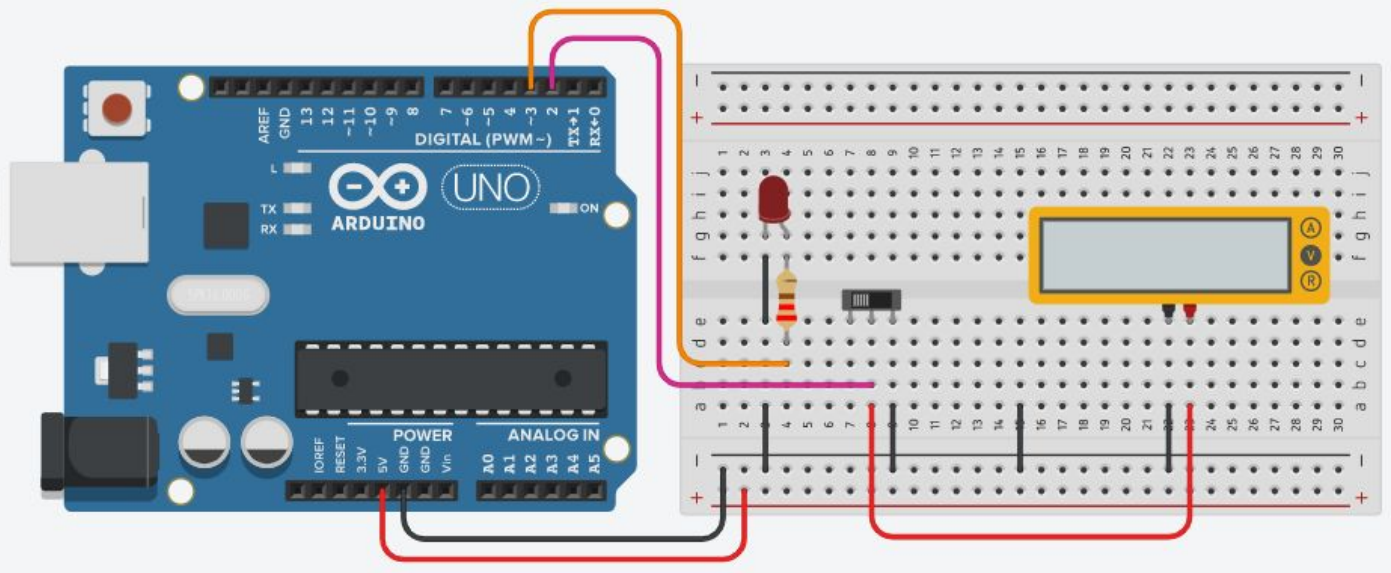
Interrupções de hardware

- Tabela de tipos de interrupções

Modo	Operação	Observações
`LOW`	Dispara a interrupção sempre que estiver em nível baixo (`LOW`)	Executa a interrupção continuamente enquanto o pino estiver `LOW`
`RISING`	Dispara quando o pino passa de baixo (`LOW`) para alto (`HIGH`)	
`FALLING`	Dispara quando o pino passa de alto (`HIGH`) para baixo (`LOW`)	
`CHANGE`	Dispara sempre que o pino mudar de nível em qualquer sentido	
`HIGH`	Dispara a interrupção sempre que estiver em nível alto (`HIGH`)	Disponível apenas no Arduino Due

Interrupções de hardware

- Exemplo de tratamento de interrupções de hardware



Interrupções de hardware

- Exemplo de tratamento de interrupções de hardware (cont.)

```
int pinoLed = 3;
int pinoSwitch = 2;

void setup() {
    pinMode(pinoLed, OUTPUT);
    pinMode(pinoSwitch, INPUT_PULLUP);
    attachInterrupt(0, trataInterrupcao, FALLING);
}

void loop() {
}

void trataInterrupcao() {
    digitalWrite(pinoLed, HIGH);
}
```

Interrupções de hardware

- As funções executadas em uma interrupção são chamadas ISR
  - Uma ISR não pode receber parâmetros e não retorna valor
  - É necessário usar uma variável global volátil para transferir informações entre a ISR e o restante do programa

- Variável do tipo `volatile`
  - Evita que o compilador coloque a variável em um registrador especial
    - Este processo do compilador é uma tentativa de otimizar o desempenho
    - Porém, isto pode resultar em um valor incorreto
  - Em outras palavras, esta diretiva informa ao compilador que o valor desta variável pode mudar a qualquer momento
    - Não necessariamente pelo código a sua volta

Interrupções de hardware

- Exemplo de tratamento de interrupção com a diretiva `volatile`

```
int pinoLed = 3;
int pinoSwitch = 2;
bool volatile ligado = true;

void setup() {
    pinMode(pinoLed, OUTPUT);
    pinMode(pinoSwitch, INPUT_PULLUP);

    attachInterrupt(0, trataInterrupcao, FALLING);
}

void trataInterrupcao() {
    ligado = !ligado;
    digitalWrite(pinoLed, ligado ? HIGH : LOW);
}
```

Interrupções de hardware

- Exemplo de uso de variáveis `volatile` na função `loop`

```
int pinoLed = 3;
int pinoSwitch = 2;
bool volatile rapido = false;

void setup() {
    pinMode(pinoLed, OUTPUT);
    pinMode(pinoSwitch, INPUT_PULLUP);

    attachInterrupt(0, trataInterrupcao, FALLING);
}
```

Interrupções de hardware

- Exemplo de uso de variáveis `volatile` na função `loop` (cont.)

```
void loop() {
    digitalWrite(pinoLed, HIGH);
    delay(rapido ? 200 : 2000);

    digitalWrite(pinoLed, LOW);
    delay(rapido ? 200 : 2000);
}

void trataInterrupcao() {
    rapido = !rapido;
}
```

Interrupções de hardware

- Recomendações e detalhes sobre ISR
  - O controlador não interrompe uma ISR em execução
    - Mesmo se outra interrupção ocorrer enquanto uma ISR estiver em execução
    - Recomenda-se utilizar rotinas pequenas
  - As funções `delay` e `millis` não funcionarão corretamente em uma ISR
    - Estas funções dependem de outras interrupções
    - Recomenda-se o uso do `delayMicroseconds` no lugar de `delay`
  - As comunicações seriais também necessitam de interrupções
    - Portanto, também não funcionarão corretamente em uma ISR

Controle de interrupções de hardware

- Por padrão, as interrupções são habilitadas
  - Exceto dentro de uma ISR

- Porém, há situações que requer desativar interrupções manualmente
  - Exemplo
    - Caminho crítico de programas
    - Interação com o meio físico
    - Gravação de dados
    - Comunicação serial ou pulsos com tempos específicos (sincronizados)

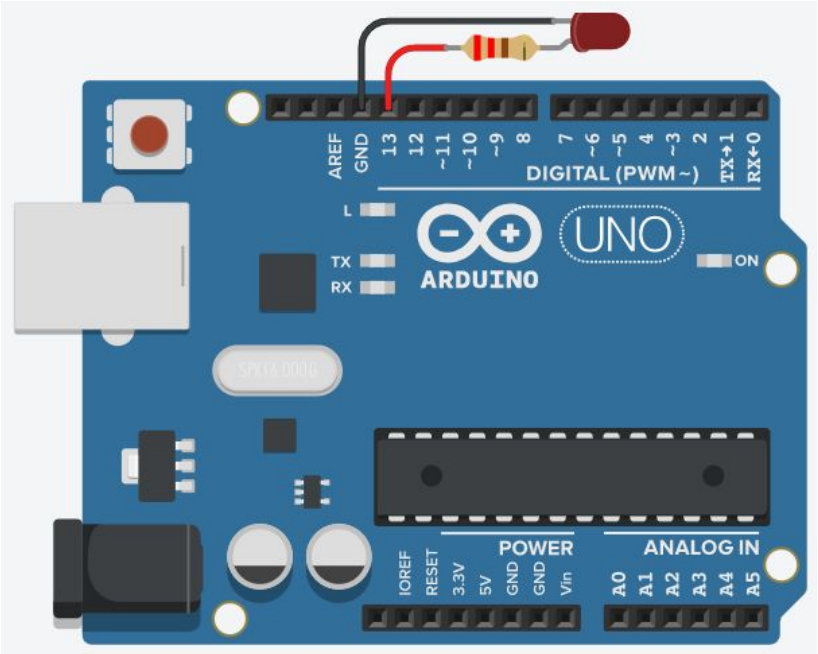
  - Funções para controle de interrupções de hardware
    - `noInterrupts()`: desativa interrupções de hardware
    - `interrupts()`: ativa interrupções de hardware

Interrupções de temporizador

- Também é possível invocar uma ISR de eventos ocorridos no tempo
  - Para isto, utiliza-se bibliotecas adicionais como a `TimerOne`
    - http://playground.arduino.cc/Code/Timer1
  - O tempo da interrupção pode variar de 1 até 8.388.480 microsegundos
    - Aproximadamente 8,4 segundos
- Principais funções de controle
  - `initialize(<intervalo>)`: configura o intervalo de tempo
  - `attachInterrupt(<ISR>)`: configura a ISR para a interrupção

Interrupções de temporizador

- Exemplo de interrupções de temporizador



Interrupções de temporizador

- Exemplo de interrupções de temporizador (cont.)

```
#include <TimerOne.h>

int pinoLed = 13;
volatile int saida = LOW;

void setup() {
    pinMode(pinoLed, OUTPUT);

    Timer1.initialize(500);
    Timer1.attachInterrupt(alteraSaida);
}
```

Interrupções de temporizador

- Exemplo de interrupções de temporizador (cont.)

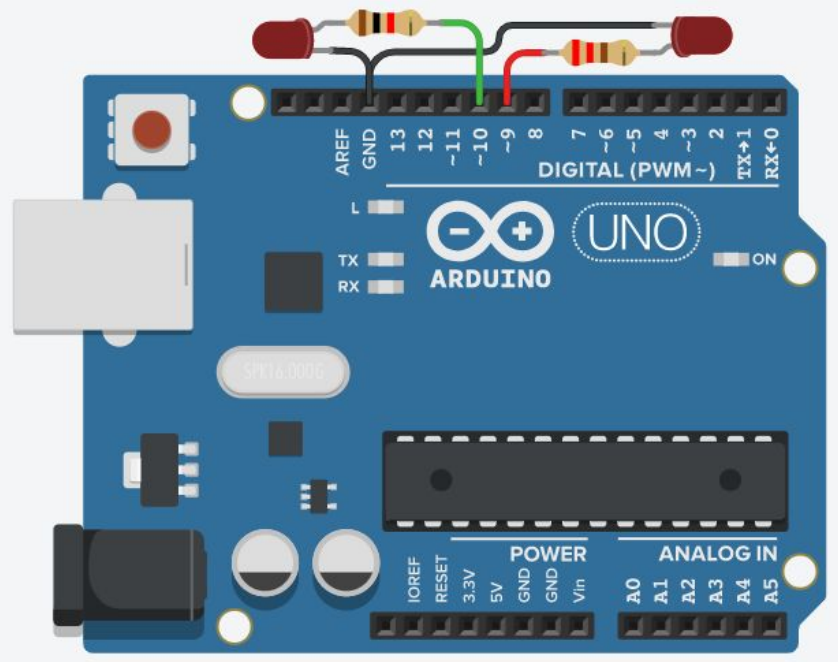
```
void loop() {

}

void alteraSaida(){
    digitalWrite(pinoLed, saida);
    saida = !saida;
}
```

Interrupções de temporizador

- Interrupções de temporizador com PWM
  - Permite gerar um sinal PWM com variação no *duty cycle*
    - Usando apenas o `analogWrite` não é possível alterar a frequência
  - Função `pwm(<pino>, <duty cycle>)`
    - O pino precisa ser o pino 9 ou 10
    - O *duty cycle* é uma variação entre 0 até 1023

Interrupções de temporizador

- Exemplo de interrupções de temporizador com PWM



Interrupções de temporizador

- Exemplo de interrupções de temporizador com PWM (cont.)

```
#include <TimerOne.h>

int pinoLed1 = 9;
int pinoLed2 = 10;

void setup() {
    pinMode(pinoLed1, OUTPUT);
    pinMode(pinoLed2, OUTPUT);

    Timer1.initialize(1000);

    Timer1.pwm(pinoLed1, 512);
    Timer1.pwm(pinoLed2, 255);
}
```

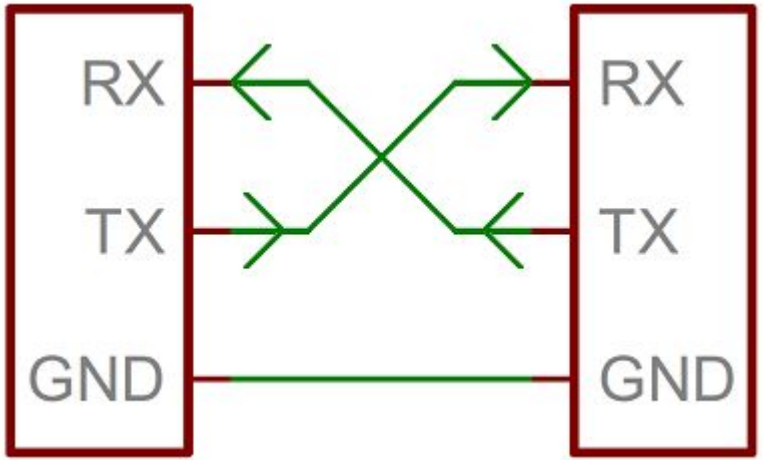
Exercícios

6. Crie um sketch no Arduino para alertar a chegada de uma pessoa em um estabelecimento. Enquanto o sistema não foi acionado (botão pressionado), um LED deve piscar com intervalos de 5 segundos. Ao entrar no estabelecimento, a pessoa deve pressionar um botão. O sistema deve emitir uma melodia com um buzzer e piscar um LED com intervalos de 1 segundo durante 10 segundos. Caso o botão seja pressionado novamente no meio da execução da melodia, esta deve ser reiniciada imediatamente.

7. Crie um sketch no Arduino para reproduzir um semáforo de trânsito contendo 3 LEDs. Porém, utilize apenas temporizadores para controlar a sequência dos LEDs.

Comunicação UART

- Permite ligar dois dispositivos diretamente para troca de informações binárias de forma serial
  - Ambos os dispositivos precisam acordar as seguintes configurações
    - Velocidade de transferência
    - Tamanho de caractere
    - Paridade
    - Bit de parada
  - Note que o RX (receptor) para um dispositivo será o TX (transmissor) para outro



Comunicação UART

- Esta é a forma de comunicação entre o PC e o Arduino
  - O Arduino possui internamente um conversor de USB para UART
  - Outros exemplos de uso de comunicação UART
    - Módulos Bluetooth
    - Módulos de comunicação com rede de celular (GSM/GPRS)

- Algumas tarefas não podem ser realizadas diretamente no Arduino devida a sua baixa capacidade de processamento e memória
  - Processamento de imagens
  - Reconhecimento de linguagem
  - Interfaces gráficas avançadas, com animações ou 3D
  - Algoritmos mais elaborados para robótica e controle

Comunicação UART

- Uma das formas de se comunicar entre dois dispositivos é criar um protocolo de comunicação
  - Um protocolo popular é o FIRMATA
  - https://github.com/firmata/protocol

- Exemplo
  - Computador captura a imagem e identifica a pessoa
  - Computador envia a *string* `"d:1:255"` via UART para o Arduino
  - Ao receber uma *string*, o programa no Arduino interpreta o comando e identifica que deve colocar o pino digital 1 com valor de 255

Comunicação UART

- O protocolo FIRMATA
  - Padrão de comunicação e bibliotecas para diversas linguagens
    - Java
    - C#
    - Python
    - C++
    - …
  - Para usar a biblioteca é necessário
    - Instalar o sketch StandardFirmata no Arduino
    - Baixar uma biblioteca para a linguagem destino
    - Usar os comandos similar aos existentes no Arduino

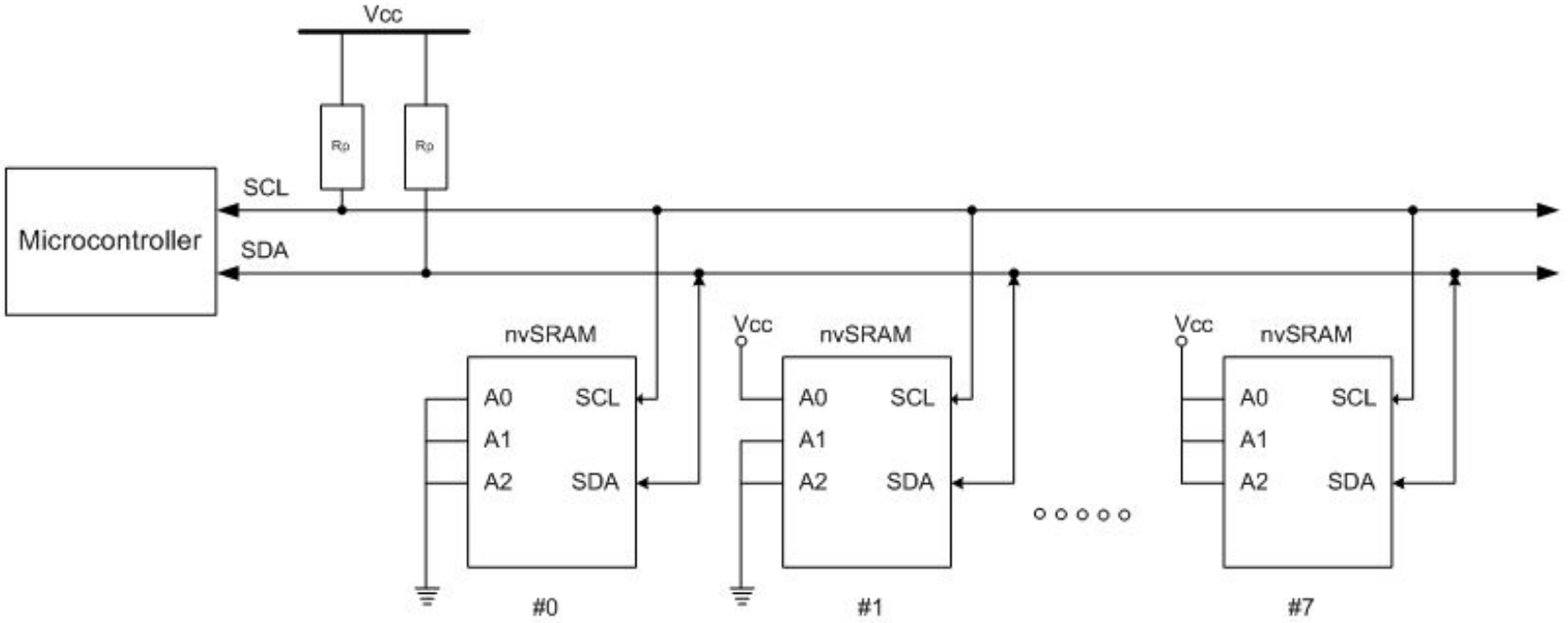
Comunicação I2C

- Padrão de barramento I2C ou TWI (*Two Wire Interface*)
  - Barramento de conexão entre microcontroladores e periféricos
    - É um barramento pois não se limita a ligação de um componente em outro
    - Ou seja, o mesmo barramento pode conectar diversos componentes
    - Para isto usa o conceito mestre-escravos

- O I2C utiliza duas ligações para fazer a comunicação dos dados
  - Ligações SCL (*Serial Clock Line*) e SDA (*Serial Data Line*)
  - O mestre fornece o sinal de *clock* pelo SCL e, quando há dados para transmissão, um dos lados tira a linha SDA do *tri-state*
    - *Tri-state* é o modo flutuante entre o LOW e o HIGH
  - Então é enviado serialmente os dados como níveis lógicos `HIGH` e `LOW`

Comunicação I2C

- Padrão de barramento I2C ou TWI (*Two Wire Interface*)



Comunicação I2C

- O dispositivo mestre se comunica com os escravos através de um endereço definido por cada escravo
  - A comunicação deve informar a quantidade de bytes a serem lidos
  - Normalmente o dispositivo I2C acompanha um *datasheet* mostrando o formato e tamanho das mensagens

- O Arduino possui a biblioteca `Wire` para comunicação I2C
  - As ligações (SCL e SDA) devem ter um resistor de *pullup* de 4.700 ohms

Comunicação I2C

Placa	Pinos	Observações
Uno	A4 (SDA) e A5 (SCL)	As conexões SCL e SDA próximos de AREF também estão conectadas a A4 e A5
Leonardo	D2 (SDA) e D3 (SCL)	As conexões SCL e SDA próximo de AREF também estão conectadas a D2 e D3
Mega2560	D20 (SDA) e D21 (SCL)	
Due	D20 (SDA) e D21 (SCL)	O Due também tem um segundo par de conexões I2C, denominadas SDA1 e SCL2

Comunicação I2C

- O dispositivo mestre se comunica com os escravos através de um endereço definido por cada escravo
  - A comunicação deve informar a quantidade de bytes a serem lidos
  - Normalmente o dispositivo I2C acompanha um *datasheet* mostrando o formato e tamanho das mensagens

- O Arduino possui a biblioteca `Wire` para comunicação I2C
  - As ligações (SCL e SDA) devem ter um resistor de *pullup* de 4.700 ohms

Comunicação I2C

- Exemplo de uso da biblioteca `Wire` para comunicação I2C

```
#include <Wire.h>

void setup() { //código do emissor
    Wire.begin();
    Wire.beginTransmission(4);
    Wire.send(123);
    Wire.send("ABC");
    Wire.endTransmission();
}

void loop() {
}
```

Comunicação I2C

- Exemplo de uso da biblioteca `Wire` para comunicação I2C (cont.)

```
#include <Wire.h>

void setup() { //código do receptor
    Wire.begin();
    Serial.begin(9600);
}

void loop() {
    Wire.requestFrom(4,6); //solicita 6 bytes do escravo no endereço 4
    while (Wire.available()) {
        char c = Wire.read(); //recebe um byte como caractere
        Serial.print(c);
    }
    delay(500);
}
```

Comunicação SPI

- É outro padrão de barramento serial usado para conectar periféricos
  - Porém, não funciona como um barramento verdadeiro
  - Requer um pino adicional para cada periférico adicionado

- Usa 4 ligações e possui uma velocidade maior do que I2C
  - *Full-duplex* de até 10mbps

- As ligações deste padrão são
  - SCLK - *System Clock*
  - MOSI - *Master Out Slave In* (saída de mestre e entrada de escravo)
  - MISO - *Master In Slave Out* (entrada de mestre e saída de escravo)
  - SS - *Slave Select*

Comunicação SPI

- Não há necessidade de resistores de *pull-up*
  - Os dados são enviados e recebidos ao mesmo tempo
  - Enquanto o mestre está enviando um dado pelo pino MOSI, o escravo pode estar enviando um dado pelo pino MISO

- Os dados são transmitidos em bytes inteiros (8 bits)
  - Assim os dispositivos que necessitam de bytes quebrados precisam realizar manipulações
    - Por exemplo, para 12 bits são utilizados 16 bits de transmissão

Comunicação SPI

- Dependendo do fabricante, algumas configurações adicionais são necessárias
  - Frequência do *clock* do relógio
  - Ordem de significância dos bits
  - LSBFIRST (bit menos significativo primeiro)
  - MSBFIRST (bit mais significativo primeiro)
  - Polaridade e fase do sinal de relógio

- O ideal é que o fabricante do dispositivo forneça as bibliotecas necessárias para comunicação

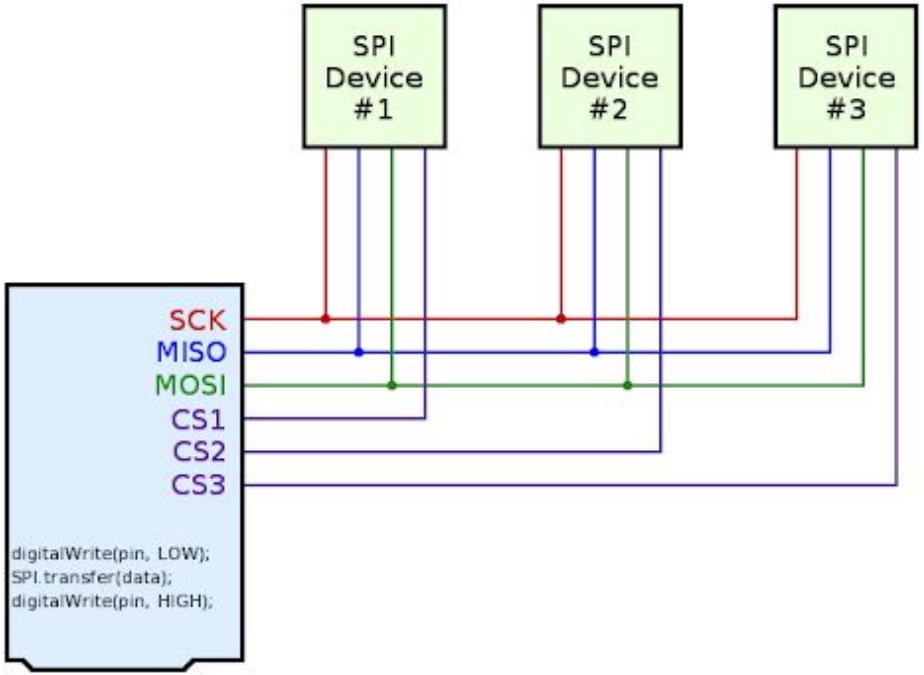
Comunicação SPI

- Exemplos de uso da comunicação SPI
  - Módulos de memória EEPROM e FLASH
  - Controladores de display LCD
  - Controladores USB
  - Módulos de comunicação de rede
  - Demais dispositivos que requerem alta velocidade de comunicação

Placa	**SCLK**	**MOSI**	**MISO**
Uno	13 (ICSP3)	11 (ICSP4)	12 (ICSP1)
Leonardo	ICSP3	ICSP4	ICSP1
Mega2560	52 (ICSP3)	51 (ICSP4)	50 (ICSP1)
Due	ICSP3	ICSP4	ICSP1

Comunicação SPI

- Exemplos de uso da comunicação SPI



Comparativo entre UART, I2C e SPI

Padrão	Tipo	Ligações de sinal	Taxa de transferência	Distância	Hardware	Escala	Exemplo de aplicação
UART	Assíncrono	2	*20kbps*	15m	Médio (Transceiver)	Baixa	Display
I2C	Síncrono	2	*1mbps*	0,5m	Baixo (Resistores)	Alta	Chip to chip
SPI	Síncrono	4+	*25mbps*	0,1m	Baixo (s/ resistor)	Média	Sensores

Portas e protocolos de comunicação

Sistemas embarcados

Prof. Allan Rodrigo Leite