FIAP GRADUAÇÃO

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Arquiteturas Disruptivas, IoT, Big Data e IA

Prof. Rafael Matsuyama

Roteiro

- ✓ Comunicação no Arduíno
  - ✓ Interrupções
  - ✓ Porta Serial
  - ✓ String
  - ✓ JSON

Comunicação no Arduíno

Lendo o valor dos resistores (Pt. 1)

- Verificar através da tabela de cores o algarismo correspondente à cor do primeiro anel, que será o primeiro dígito do valor da resistência.

- Verificar através da tabela de cores o algarismo correspondente à cor do segundo anel, que será o segundo dígito do valor da resistência.

- Determinar o valor para multiplicar o número formado pelos itens 1 e 2 pela cor do anel multiplicador (3ª ou 4ª faixa).

Lendo o valor dos resistores (Pt. 2)

- Verificar a porcentagem de tolerância do valor nominal da resistência pela cor do último anel.

- Ex: cores marrom (1), preto (0), laranja (x1k) e dourado: 10k ohms de resistência com tolerância de 5%

Lendo o valor dos resistores (Pt. 3)

Cor	1ª Faixa	2ª Faixa	3ª Faixa	Multiplicador	Tolerância
Preto	0	0	0	x 1 Ω	
Marrom	1	1	1	x 10 Ω	+/- 1%
Vermelho	2	2	2	x 100 Ω	+/- 2%
Laranja	3	3	3	x 1K Ω	
Amarelo	4	4	4	x 10K Ω	
Verde	5	5	5	x 100K Ω	+/- .5%
Azul	6	6	6	x 1M Ω	+/- .25%
Violeta	7	7	7	x 10M Ω	+/- .1%
Cinza	8	8	8		+/- .05%
Branco	9	9	9		
Dourado				x .1 Ω	+/- 5%
Prateado				x .01 Ω	+/- 10%

Fonte: http://eletronsdadepressao.blogspot.com.br/2015/01/codigo-de-cores-de-resistores.html

Interrupções no Arduíno (Pt. 1)

- Imagine o seguinte cenário:
  - Uma fábrica precisa monitorar a pressão de 100 tanques de fabricação de amônia.
  - Para ler a pressão de cada tanque o controlador demora 250ms.
  - Se algum tanque estiver com sobrepressão, há um tempo hábil de um segundo para abrir a válvula de escape.

- Se o controlador tivesse que ler a pressão de cada tanque para eventualmente tomar a decisão de abrir a válvula de algum deles, poderíamos ter uma explosão na fábrica, já que ele demoraria 25 segundos para ler todos os tanques

Interrupções no Arduíno (Pt. 2)

- Para essas situações, microcontroladores e microprocessadores possibilitam as chamadas **interrupções**, que são eventos que servem de gatilhos para ações especiais a serem executadas.

- Assim, um sensor especial de sobrepressão poderia estar ligado a uma porta do controlador, que lançaria uma interrupção quando um tanque estivesse nessa situação.

Interrupções no Arduíno (Pt. 3)

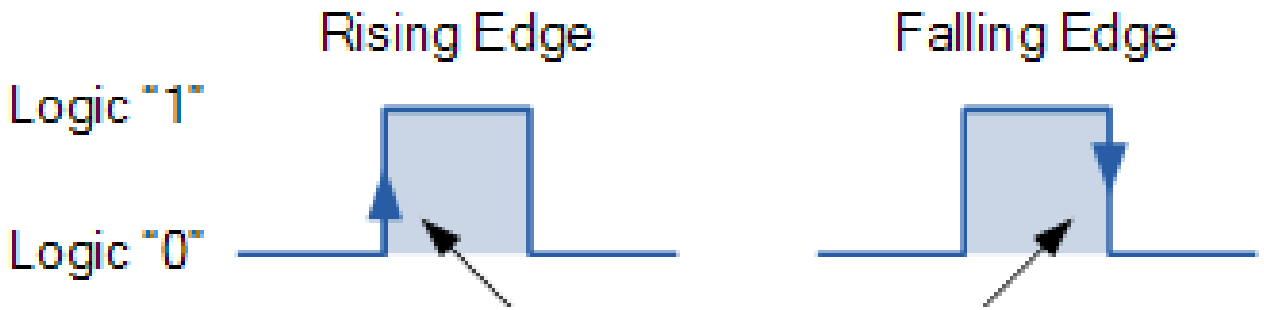
- A ação a ser executada numa interrupção é executada na forma de uma **ISR** (Interrupt Service Routine), uma função com certas limitações que é invocada assim que ocorre a interrupção, interrompendo o processamento sendo executado no momento.

- O gatilho pode ser gerado internamente, através de um temporizador chegando a zero, por exemplo, ou...

- Pode ser um gatilho externo, como o valor de uma porta de entrada sendo escrito.

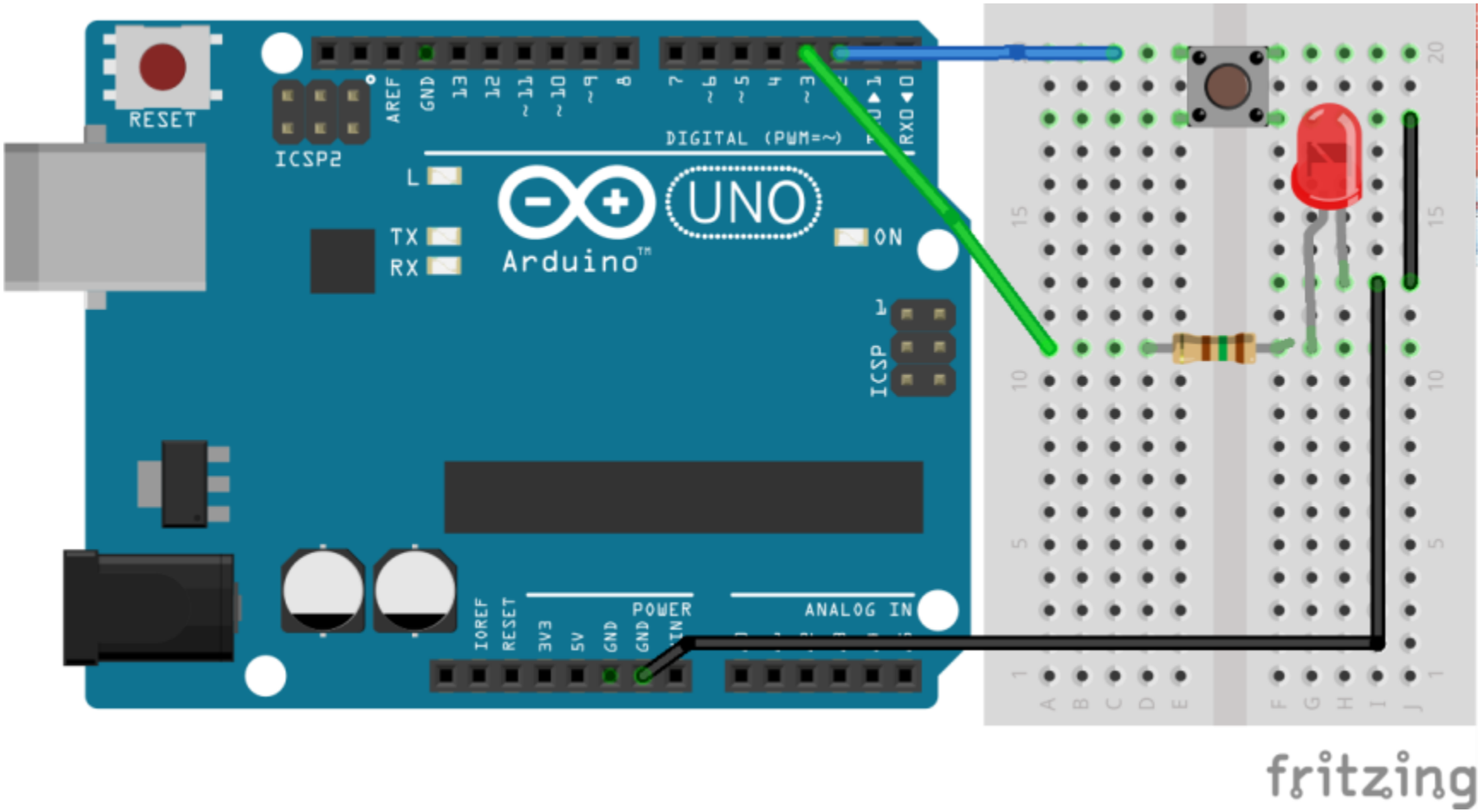
Interrupções no Arduíno (Pt. 4)

- Mais comumente o gatilho externo é de dois tipos:
  - Rising Edge: interrupção com valor indo de LOW para HIGH.
  - Falling Edge: interrupção com valor indo de HIGH para LOW.



A interrupção ocorre nestes momentos

Interrupções no Arduíno (Pt. 5)



Interrupções no Arduíno – Exemplo (Pt. 1)

```
int led = 3; //Porta do LED
//porta da interrupção:
//O Arduino Uno só aceita as portas 2 e 3
int interruptPort = 2;

//Variáveis modificadas por interrupções devem ser
volatile
volatile int state = LOW;

void setup() {
  pinMode(led, OUTPUT);
  pinMode(interruptPort, INPUT_PULLUP);

attachInterrupt(digitalPinToInterrupt(interruptPort),
toggle, CHANGE);
}
```

Interrupções no Arduíno – Exemplo (Pt. 2)

```
void loop() {
   // Qualquer processamento mais longo...
}
void toggle() {
  state = !state;
  digitalWrite(led,state);
}
```

- O botão corta o sinal elétrico para o LED, utilizando-se de uma.

Interrupções no Arduíno – Como Funciona (Pt. 1)

- A definição de uma ISR no Arduino é dada por uma função sem argumentos e sem valor de retorno (void).
- Durante a execução de uma ISR, as interrupções estão desabilitadas:
  - Por isso devem ser de rápida execução.
  - Funções de E/S podem não funcionar durante sua execução.
- Para ativar uma interrupção, invocamos a função:
  - `attachInterrupt(NUMERO, ISR, MODO)`
  - `NUMERO`: identificação da interrupção, a depender da porta usada (ver em http://arduino.cc/en/Reference/attachInterrupt).
    - Para traduzir a porta usada para o número da interrupção, usar a função `digitalPinToInterrupt(pin)`

Interrupções no Arduíno – Como Funciona (Pt. 2)

- Para ativar uma interrupção, invocamos a função:
  - `ISR`: função a ser invocada durante a interrupção
  - `MODO`: o tipo do gatilho, podendo ser
    - `LOW`: dispara a interrupção quando a porta estiver em LOW
    - `CHANGE`: dispara quando há mudança de valor
    - `RISING`: dispara quando ocorre uma Rising Edge na porta
    - `FALLING`: dispara quando ocorre uma Falling Edge na porta

Porta Serial (Pt. 1)

- Da mesma forma como podemos escrever dados na porta serial do Arduino, enviando dados para o computador, podemos também ler os dados que a placa recebe pela mesma porta.

- Podemos receber comandos do computador para executar alguma ação.

- No exemplo a seguir, vamos ler o valor da potência do LED a partir da porta serial (de 0 a 255).

Porta Serial (Pt. 2)

- Protocolo Firmata: controla o Arduino via porta serial através de um programa de computador externo. Assim, podemos mudar a programação sem ter que reprogramar o Arduino (usaremos posteriormente...).

- Para auxiliar na recuperação dos dados da porta serial, usamos a classe String do Arduino.

Classe String (Pt. 1)

- A forma tradicional de representar strings em C é através de arrays de caracteres

```
char string_do_c[256] = "Ola, isto eh uma string";
```

- No entanto, a API do Arduino fornece a classe String, que é bem mais flexível:

```
String string_do_Arduino = "Isto eh uma string do
                                              Arduino";
```

- Criando uma String a partir da concatenação de valores

```
String outraString = String("Valor: ") + 128;
```

Classe String (Pt. 2)

- Anexando valores:

```
outraString += ", outro valor:"; outraString += 256;
```

- Interpretando valores numéricos, retornando zero no caso de erro:

```
int numero = minhaString.toInt();
float numFloat = minhaString.toFloat();
```

- Mais métodos da Classe String em: https://www.arduino.cc/en/Reference/StringObject

Lendo a Porta Serial – Exemplo (Pt. 1)

```
const int LED = 3;
char nextChar = 0, lendo = 0;
String valor;
void setup() {
  Serial.begin(9600);
  pinMode(LED,OUTPUT);
}
```

Lendo a Porta Serial – Exemplo (Pt. 2)

```
void loop() {
  if (Serial.available() > 0) {
    // lê o byte disponível na porta serial
    nextChar = Serial.read();
    if(nextChar == 'B') {
      lendo = 1; //lendo <- true
      valor = "";
    } else if(nextChar == 'E') {
      lendo = 0; //lendo <- false
      analogWrite(LED,valor.toInt());
      Serial.println(String("Potencia do LED: ") + valor);
    } else if(lendo && nextChar >= '0' && nextChar <= '9') {
      valor += nextChar;
    }
  }
}
```

Decodificando Números e Linhas de Texto (Pt. 1)

- A API do Arduino possui funções que consomem os caracteres disponíveis na porta Serial de acordo com alguma regra.
- As funções `Serial.parseInt()` e `Serial.parseFloat()` leem a porta serial em busca de um número inteiro/ponto flutuante:
  - Caso caracteres não numéricos sejam encontrados, são pulados.
  - Essas funções aguardam por um tempo pré-especificado por mais caracteres até retornarem o valor definitivo. Esse tempo é 1000 ms por padrão, e pode ser configurado através de `Serial.setTimeout(int milisseg)`.
  - Caso não seja possível decodificar um número, o resultado é 0 (zero)! Isso pode gerar confusões no programa, então fique atento!

Decodificando Números e Linhas de Texto (Pt. 2)

- A função `Serial.readBytesUntil()` lê caracteres vindo da porta serial, escrevendo em um vetor de caracteres.
  - A função termina quando o caractere de terminação for detectado, o tamanho máximo for lido ou o tempo de espera por mais texto terminou.
  - `Serial.readBytesUntil(char terminador, char buffer[ ] , int tamanho).`

Lendo a Porta Serial – Exemplo 02 (Pt. 1)

```
const int LED = 3;
char nextChar = 0;
String valor;
void setup() {
  Serial.begin(9600);
  pinMode(LED,OUTPUT);
}
```

Lendo a Porta Serial – Exemplo 02 (Pt. 2)

```
void loop() {
  if (Serial.available() > 0) {
    // lê o byte disponível na porta serial
    nextChar = Serial.read();
    if(nextChar == 'B') {
      //Lê o próximo inteiro vindo da serial
      int valor = Serial.parseInt();
      //Atenção: em caso de erro o valor lido será 0
      analogWrite(LED,valor);
      Serial.println(String("Potencia do LED: ") +
valor);
    }
  }
}
```

Escrevendo na Porta Serial

- Podemos escrever na porta serial do Windows de uma forma bem simples através do Serial Monitor do Arduino (barra de texto superior): `B127E`
- O ideal é termos um programa no computador que se comunicasse com o Arduino via porta serial, e que expusesse um serviço para que possa ser controlado remotamente, através de algum protocolo para troca de dados de sensores e comandos para os atuadores.
- Para formatar as mensagens e garantir que uma ampla gama de programas consigam se comunicar com o Arduino, vamos usar o formato JSON.

JSON – JavaScript Object Notation (Pt. 1)

- Do próprio site json.org:
  - JSON é um formato leve de troca de dados (serialização), de fácil leitura e escrita por humanos e máquinas.
  - É parte da especificação de 1999 do JavaScript, que codifica e decodifica JSON nativamente.
  - JSON é um formato de texto completamente independente de linguagem.
- Exemplo de estrutura JSON:
  - {"nome": "João", "idade": 23, "mulher": false, "filhos": ["Pedro", "Artur"] }.
  - A quebra de linha é opcional, porém facilita a visualização humana.

JSON – JavaScript Object Notation (Pt. 2)

- Podemos validar um JSON através do site http://jsonlint.com/

Tipos de Valores em JSON (Pt. 1)

- **String**
  - texto unicode não formatado, escrito sempre entre aspas (como em Java)
  - Exemplos: "José", "Marçäl", "opa123", etc.
- **Número**
  - sequência de dígitos com separador decimal (ponto) e notação científica, como em Java, mas aceita apenas números decimais
  - Exemplos: 12, 15.01, 1.35e-24
- **Objeto (object)**
  - Pares do tipo chave:valor contidos entre chaves ({ }) e separados por vírgula
  - Exemplo: {"idade":23,"peso":53.5}

Tipos de Valores em JSON (Pt. 2)

- **Vetor (array):**
  - Conjunto ordenado de valores contidos entre colchetes ([ ]) e separados por vírgula.
  - Exemplo: [1, 2.5, "três", [4], {"próx": 5}].

- **true, false:** constantes lógicas representando verdadeiro e falso, respectivamente.

- **null:** constante indicando um valor nulo

Objeto JSON

- É a estrutura de dados mais emblemática do JSON.
- É composto por um conjunto de pares do tipo `chave:valor` separados por vírgula.
  - `Chave` deve ser uma string, que serve de rótulo para o valor
  - `Valor` é qualquer valor válido do JSON, incluindo um array ou ainda outro objeto
- Alguns objetos válidos
  - { } : objeto vazio
  - { "chave " : "valor " }
  - { "nome" : "Alberto", "idade" : 54, " pais" : [" José", "Maria"] }

Exemplo de um JSON válido

```
[
   {
      "id": 100,
      "nome": "Astolfo",
      "sobrenome": "Silva",
      "endereco": {
         "rua": "Rua das Orquideas",
         "no": 23
      }
   },
   {
      "id": 101,
      "nome": "Maria",
      "sobrenome": "Teresa",
      "idade": 49
   }
]
```

JSON no Arduíno (Pt. 1)

- Há várias bibliotecas em C++ para a codificação e a decodificação de JSON, porém nem todas são otimizadas para rodar no Arduino.
- A biblioteca que vamos adotar aqui é a **ArduinoJson** (https://github.com/bblanchon/ArduinoJson), que relaciona objetos JSON com a estrutura de dados de dicionário do C++.
- Para usar a API, a primeira providência é importar o seu cabeçalho no código, trazendo na primeira linha do programa:
  - `#include <ArduinoJson.h>`

JSON no Arduíno (Pt. 2)

- Os dicionários do C++ também relacionam uma chave (ou índice) a um rótulo, acrescentando dinamicamente elementos.
  - `meuDic["nome"] = "Pedro Henrique";  //Acrescenta um elemento`
  - `long valor = meuDic["idade"]; // Lê o elemento idade`

Criando um JSON (Pt. 1)

- Primeiramente, devemos reservar memória para a criação do objeto (aqui é Arduino, não esqueçam)

```
StaticJsonBuffer<200> jsonBuffer;  //Reserva 200 bytes
```

- Então devemos alocar a árvore JSON na memória

```
JsonObject& raiz = jsonBuffer.createObject();
```

  - Podemos pensar em um objeto ou array JSON como sendo uma árvore porque cada elemento é considerado um filho, que por sua vez podem ser estruturas contendo outros objetos ou arrays, e assim por diante.

Criando um JSON (Pt. 2)

- Agora podemos criar os elementos, com formatos identificados automaticamente

```
raiz["sensor"] = "gps";
raiz["time"] = 1351824120;
```

- Valores decimais devem ser especificados com o número de casas decimais desejadas, caso contrário são usadas apenas 2 casas. O exemplo abaixo usa 4 casas:

```
raiz["pi"] = double_with_n_digits(3.1415, 4);
```

Criando um JSON (Pt. 3)

- Para acrescentar um array ou outro objeto, é necessário usar um método especial.

```
JsonArray& vetor = raiz.createNestedArray("vetor");
vetor.add("José"); vetor.add(48.756080, 6); //Usa 6 casas
JsonObject& obj = raiz.createNestedObject("obj");
```

- Finalmente, imprimimos a string JSON resultante na porta serial ou em uma string do C.

```
raiz.printTo(Serial); //manda o resultado pela porta serial
```

Lendo um JSON (Pt. 1)

- Para decodificar um JSON é ainda mais fácil. Basta reservar a memória, checar possíveis erros e resgatar os valores desejados
  - ```
    char json[] = "{
    \"sensor\":\"gps\",\"time\":1351824120,
    \"data\":[48.756080,2.302038]}";
    ```
  - ```
    StaticJsonBuffer<200> jsonBuffer;
    ```
  - ```
    JsonObject& raiz = jsonBuffer.parseObject(json);
    ```
  - ```
    if (!raiz.success()) {/* Tratar o erro*/}
    ```

Lendo um JSON (Pt. 2)

- Capturando os valores:
  - `const char* sensor = raiz["sensor"];`
  - `long time = raiz["time"];`
  - `double latitude = raiz["data"][0];`
  - `double longitude = raiz["data"][1];`

- Exemplo: criar um programa que manda a luminosidade, temperatura e umidade no formato JSON pela porta serial, enquanto aceita comandos para o LED usando JSON. Escolha o nome dos campos a serem preenchidos.

Lendo um JSON - Exemplo (Pt. 1)

```
#include <ArduinoJson.h>
const int LED = 3;
const int LUZ = A1;
const int TAMANHO = 200;

void setup() {
  Serial.begin(9600);
  Serial.setTimeout(10); //1000ms é muito tempo
  pinMode(LED,OUTPUT);
}
```

Lendo um JSON - Exemplo (Pt. 2)

```
void loop() {
  if (Serial.available() > 0) {
    //Lê o texto disponível na porta serial:
    char texto[TAMANHO];
    Serial.readBytesUntil('\n', texto, TAMANHO);
    //Grava o texto recebido como JSON
    StaticJsonBuffer<TAMANHO> jsonBuffer;
    JsonObject& json = jsonBuffer.parseObject(texto);
    if(json.success() && json.containsKey("led")) {
      analogWrite(LED, json["led"]);
    }
  }
```

Lendo um JSON - Exemplo (Pt. 3)

```
 StaticJsonBuffer<TAMANHO> jsonBuffer;
  JsonObject& json = jsonBuffer.createObject();
  json["luz"] = analogRead(LUZ);
  json.printTo(Serial); Serial.println();
  delay(1000);
}
```

REFERÊNCIAS

1. http://www.telecom.uff.br/pet/petws/downloads/tutoriais/arduino/Tut_Arduino.pdf
2. http://arduino.cc/en/Reference/HomePage
3. https://www.arduino.cc/en/Reference/StringObject
4. http://json.org

FIAP