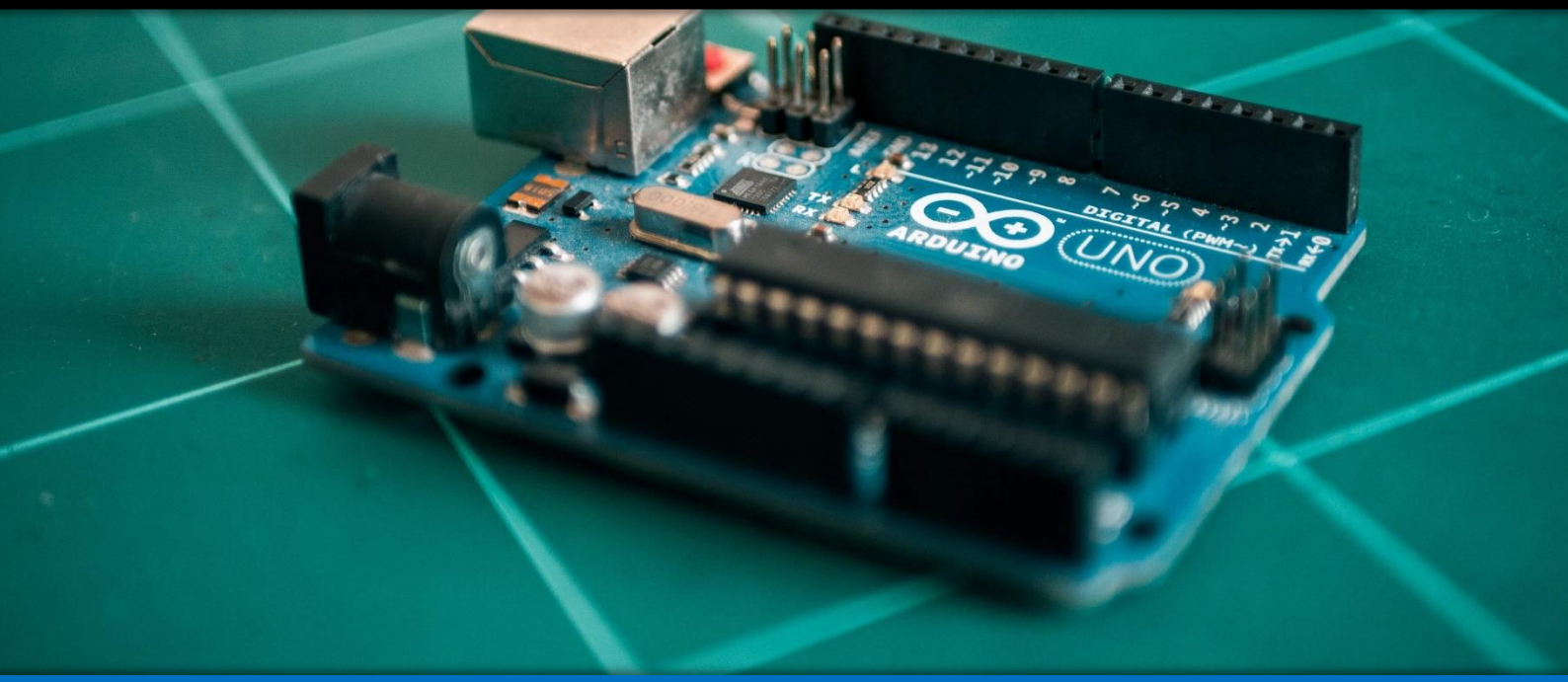
CURSOS

ELETRÔNICA FÁCIL

APRENDER ELETRÔNICA É MAIS FÁCIL DO QUE PARECE!



SEU CURSO ONLINE COMPLETO DE

ELETRÔNICA DIGITAL E ARDUÍNO PARA INICIANTES

- DICAS INÉDITAS
- BÔNUS EXCLUSIVOS
- FORUM DE DEBATE

CURSO COM CERTIFICADO!

➲ **Receba conteúdos GRÁTIS e exclusivos no Telegram:**

CLIQUE PARA ENTRAR NO TELEGAM

➲ **Acessar o site do curso completo de Eletrônica Digital e Arduino para iniciantes:**

CLIQUE AQUI PARA ACESSAR O SITE

CURSOS

ELETRÔNICA FÁCIL

Apresentação

O mundo da Eletroeletrônica é repleto de coisas magníficas, não é mesmo? O avanço tecnológico e contínuo nos processos manufatureiros com a chegada da Indústria 4.0, os novos modelos de automação residencial, as maneiras diversificadas de fazer agricultura e até mesmo a mobilidade e conforto urbano, são alguns dos fatos que comprovam isso. Mas a pergunta é: O quanto você está preparado para ser esse novo modelo de profissional? Já parou para pensar no quanto as formas de produzir vão mudar e que, consequentemente, o modelo de colaborador do futuro será totalmente outro? No entanto, uma coisa é certa, a Eletrônica estará presente em tudo que se pode imaginar.

Com o Curso de Eletrônica Fácil você consegue se destacar buscando o aprendizado de uma forma dinâmica, e aqui levamos o aprendizado é levado a sério, na teoria e na prática! Se você é um entusiasta, técnico, engenheiro ou autodidata, e está em busca do aprendizado "Mão na Massa", certamente aqui é o seu lugar! Esse e-Book foi desenvolvido visando estabelecer um apoio didático, com muitas informações teóricas e listas de exercícios pertinentes à cada módulo de estudo. É um material exclusivo para este curso e totalmente desenvolvido pela equipe Eletrônica Fácil.

Eletrônica Fácil, sua escola online.

Um recado do autor

Caro aluno (a), tenho um recado importante para você sobre boas práticas no curso e utilização desta apostila.

Os módulos aqui contidos podem não acompanhar a sequência numérica dos vídeos da plataforma, no entanto, são posicionados de acordo com a cronologia desenvolvida para o treinamento. Não deixe de fazer os exercícios de fixação presentes na plataforma, e em caso de dúvida, deixe seu comentário no vídeo da aula correspondente.

Viva essa experiência e compartilhe conhecimentos, sempre!

Bons estudos,

Equipe Eletrônica Fácil.

CURSOS

ELETRÔNICA FÁCIL

SEJA MUITO BEM-VINDO (A)!

CARO (A) ALUNO (A), TE DESEJAMOS BOAS-VINDAS E QUE ESTA SEJA UMA EXPIRIÊNCIA INCRÍVEL.

SUA ESCOLA ONLINE

CURSOS ELETRÔNICA FÁCIL

Bons estudos!

SUMÁRIO

INTRODUÇÃO AO FANTÁSTICO MUNDO DA ELETRÔNICA DIGITAL

CONCEITOS BÁSICOS

M1

CURSO DE
ELETRÔNICA FÁCIL

SINAIS ALTERNADOS E SINAIS CONTÍNUOS

Para entender o que sinal alternado ou contínuo, vamos primeiramente entender o que é um sinal.

> *Sinal é uma amostra de um acontecimento, pode ser uma expressão, um nível de tensão, pode ser um gesto, um levantar de mão, um sorriso, enfim, é definitivamente um evento que contém uma informação.*

Quando dizemos que um sinal pode ser contínuo ou alternado estamos automaticamente dizendo que esse sinal pode ou não sofrer mudanças relevantes em um determinado momento, ou seja, ele pode se alterar ou manter-se de uma mesma maneira no decorrer do tempo.

Para efeito didático **vamos considerar momentaneamente um sinal como sendo um nível de tensão** que pode ter potencial **positivo ou negativo**. Quando esse nível de tensão permanece positivo no decorrer do tempo, dizemos que esse **sinal é contínuo**. Quando esse nível de tensão muda sua polaridade no decorrer do tempo, dizemos que esse **sinal é alternado.**

No gráfico podemos observar que no eixo vertical temos a representação da tensão, e no eixo horizontal temos a representação do tempo. Perceba que a linha do tempo está sobre o ponto zero do eixo vertical, por isso a parte superior a linha do tempo irá representar os valores positivos de tensão e a parte inferior os valores negativos de tensão.

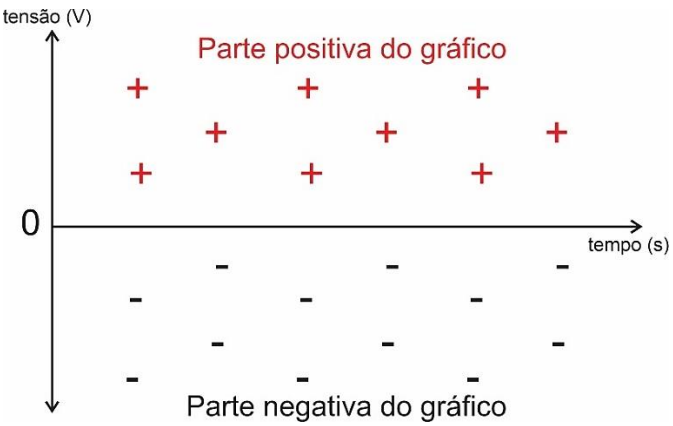


Figura 1 – Polaridades em um gráfico

> ***DEFINIÇÃO: Sinal alternado** é aquele que possui **mais de uma polaridade no decorrer do tempo**.*

Agora veja como um **sinal alternado senoidal** é representado graficamente:

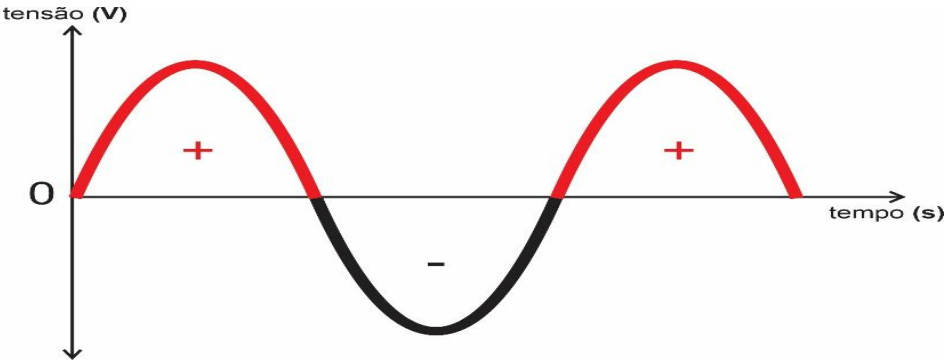


Figura 1 - Sinal senoidal alternado

O sinal acima é a representação de uma tensão que cresce e decresce no decorrer do tempo por meio da *função seno*, ***alternando sua polaridade.*** Portanto, temos um ***sinal senoidal ALTERNADO.***

Figura 3 - https://www.tecnogera.com.br/blog/entenda-o-que-e-uma-subestacao-e-conheca-suas-classificacoes

Figura 2 - Tomada

Você sabe onde encontramos esse sinal? Essa é a forma do sinal elétrico (tensão) que chega das concessionárias de fornecimento de energia elétrica e entra em nossas residências, a famosa tensão alternada. Esse sinal troca sua polaridade em uma frequência de 60Hz.

Mas, o que é um sinal contínuo?

Figura 4 - Sinal Contínuo

DEFINIÇÃO: ***Sinal contínuo*** *é aquele que possui apenas* ***uma polaridade no decorrer do tempo.***

Veja no gráfico que o sinal representado permanece constantemente positivo com o passar do tempo, por isso temos a representação gráfica de um **sinal contínuo**.

Mas onde podemos encontrar esse sinal de tensão contínua?

Figura 6 - Fonte DC 9V

Figura 7 - Fonte ATX - Computador

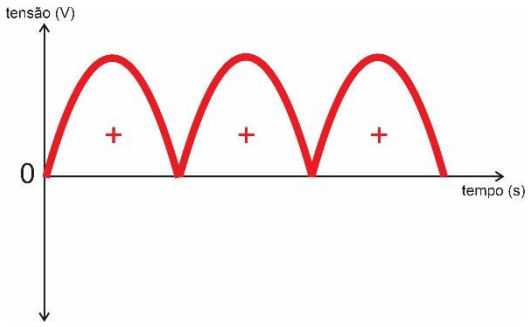
Figura 5 - Pilhas 1,5V

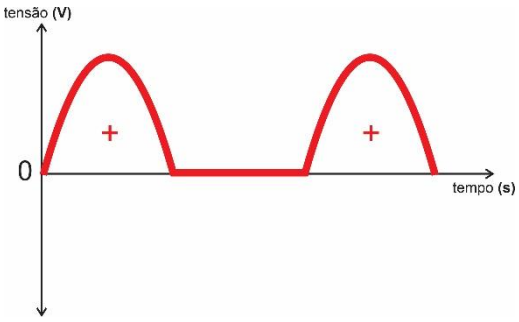
Temos uma variedade de dispositivos que fornecem tensão contínua (DC) para o funcionamento dos aparelhos eletrônicos utilizados no dia a dia, como as pilhas, fontes de alimentação de wifi, fontes de teclados, fontes de notebook, fontes atx, enfim, são diversos os dispositivos.

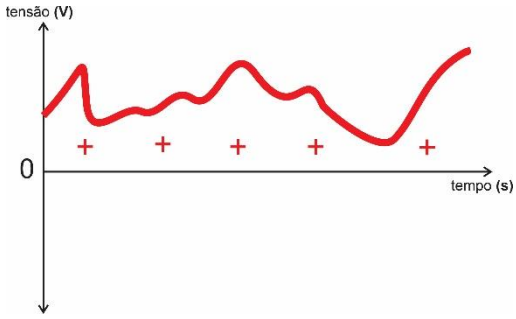
Agora você deve estar pensando, se um sinal contínuo é aquele que permanece em um uma única polaridade no decorrer do tempo, ele tanto pode permanecer positivo quanto negativo, e mesmo assim ser contínuo, correto? Exatamente!

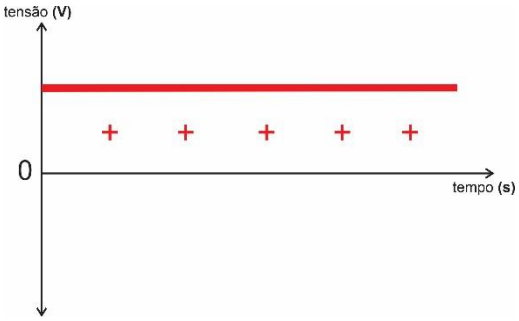
Veja os exemplos abaixo para os sinais contínuos positivos, e sinais contínuos negativos.

Contínuo positivo – possui apenas polaridade positiva no decorrer do tempo

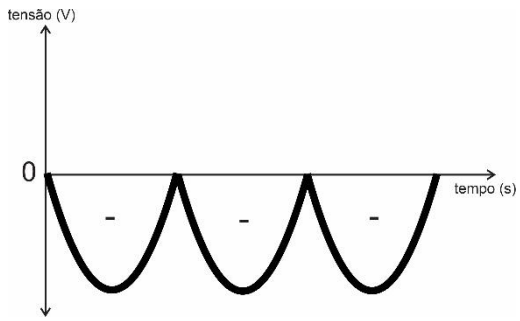


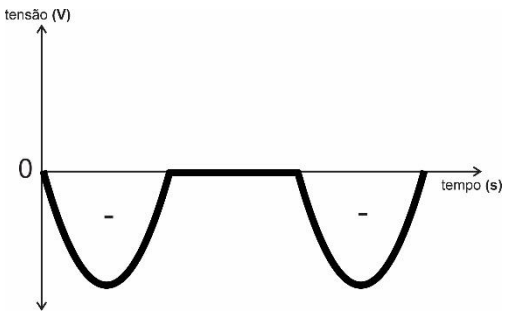


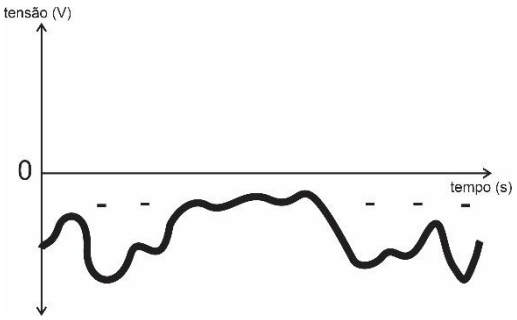


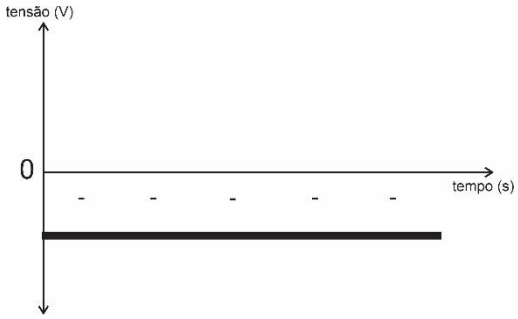


Contínuo negativo – possui apenas polaridade negativa no decorrer do tempo









SITUAÇÕES COTIDIANAS – TIPOS DE SINAIS

SALDO BANCÁRIO

Vamos imaginar que uma pessoa resolveu analisar a movimentação do seu saldo bancário durante alguns meses e para isso colocou todos os dados em um gráfico de saldo em função dos meses. Ao representar graficamente esse saldo bancário durante 10 meses, ficou claro que em um determinado momento o saldo passou de positivo para negativo, e depois voltou a ser positivo novamente. Poderíamos claramente dizer que o **saldo bancário** dessa pessoa é **ALTERNADO**, pois durante os meses, o que era saldo positivo passou a ser negativo, e vice versa.

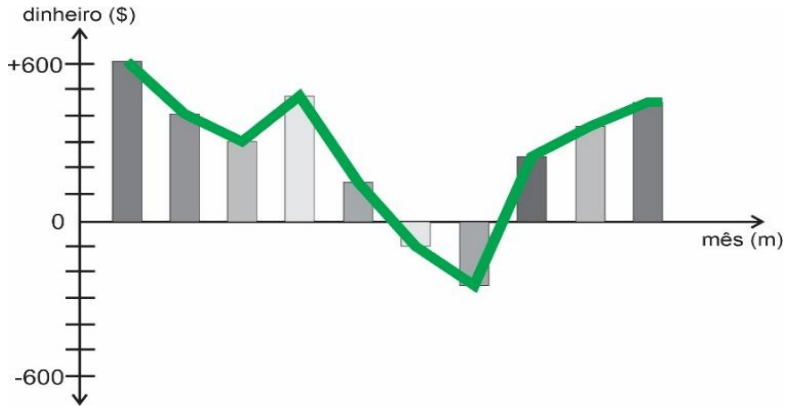


Figura 8 - Saldo bancário alternado

TEMPERATURA DESERTO DO SAARA

Um cientista ao fazer seus estudos de temperatura no deserto no Saara, resolveu representar graficamente as temperaturas aferidas durante 24h. Ao observar o gráfico construído o cientista verificou que durante o dia a temperatura atingia os 50°C e durante a noite essa temperatura caía bruscamente para -5°C. Claramente o cientista concluiu que a temperatura no deserto do Saara possuía um gráfico de **sinal ALTERNADO**.

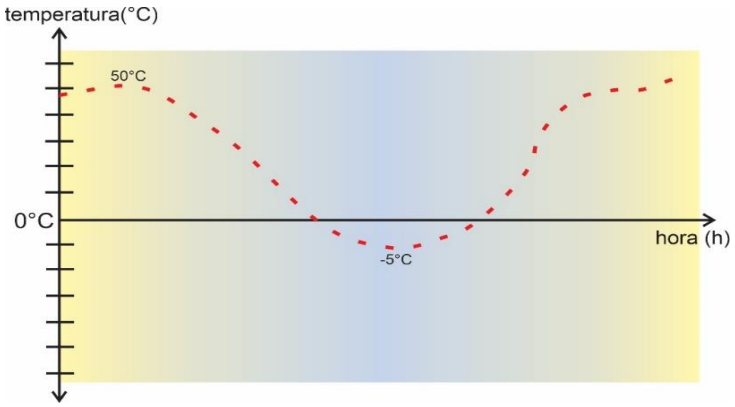


Figura 9 - Temperatura deserto do Saara

TEMPERATURA NO PÓLO SUL DO PLANETA TERRA

No extremo sul do planeta terra a temperatura é constantemente negativa ficando em média nos -30°. Ao observar o gráfico dessa temperatura podemos observar que mesmo com o passar das horas, os valores continuam constantemente negativos, caracterizando um gráfico de **sinal contínuo**, ou seja, continuamente negativo.

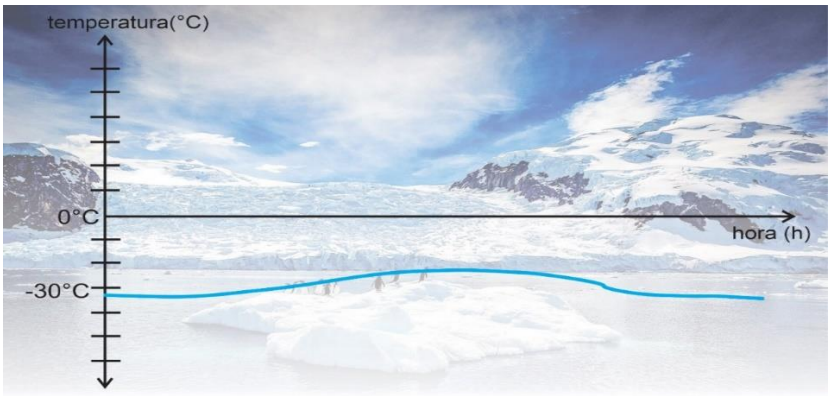


Figura 10 - Temperatura polo Sul

TEMPERATURA DO CORPO HUMANO

Um paciente estava em observação em um consultório médico e sua temperatura foi aferida durante alguns intervalos de tempo e posteriormente inserida em um gráfico. O médico de plantão ao analisar o gráfico concluiu que houve uma queda de temperatura o que indicava uma piora do paciente, pois sua temperatura não estava constante.

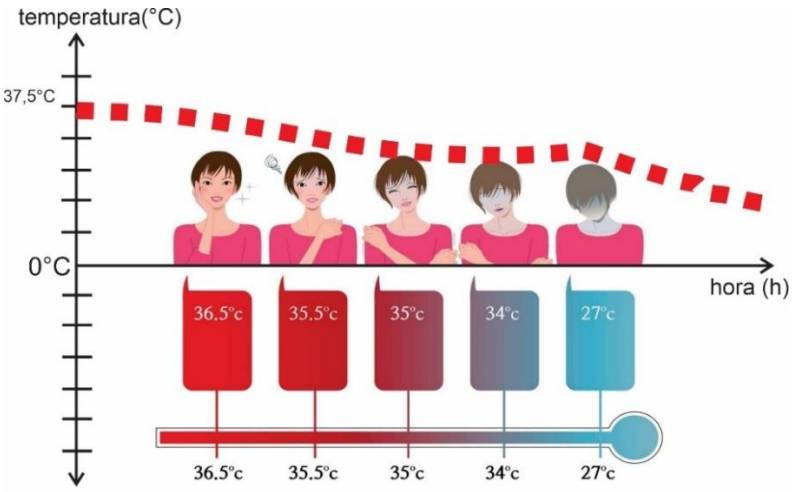


Figura 11 - Temperatura corpo humano

Agora você deve estar pensando: Se a temperatura do paciente não estava constante e houve variação, então ela não é contínua?

Quando falamos em sinal contínuo estamos dizendo que esse sinal ***não trocou sua polaridade no decorrer do tempo****, mas isso não quer dizer que ele não possa ter variado dentro dessa polaridade fundamental. Então,* ***sinais contínuos também podem variar, mas nunca trocar sua polaridade.***

O médico então pode concluir que apesar da variação da temperatura do paciente, o gráfico apontou sempre temperaturas positivas, caracterizando um sinal contínuo positivo.

Mas então, como pode ser conceituada essa variação de informações durante um período de tempo? Existe algo que defina isso?

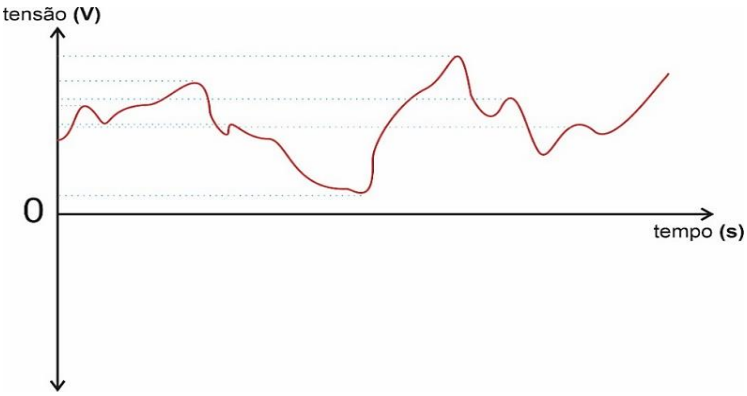
SINAIS ANALÓGICOS E SINAIS DIGITAIS

Já aprendemos que os sinais contínuos e alternados se diferenciam pela variação de polaridade no decorrer do tempo, ou seja, se a polaridade variar o sinal é alternado, se não variar é contínuo.

Agora, para diferenciar um sinal **analógico** de um sinal **digital,** vamos precisar fazer uma análise de **amplitude do sinal.**

> ***Amplitude de um sinal*** *é a intensidade que esse sinal possui, e se pensarmos em sinal como sendo um nível de tensão, a amplitude de um sinal será o nível de tensão que esse sinal terá no decorrer do tempo.*

Dizemos que um sinal é **analógico** quando ele possui diversos valores de amplitude no decorrer do tempo.



> ***DEFINIÇÃO: Sinal analógico*** *é aquele que possui diversos valores de amplitude no decorrer do tempo*

Figura 12 - Sinal analógico

Quando o sinal tem **apenas duas amplitudes** no decorrer do tempo, ou seja, apenas dois valores de tensão no decorrer do tempo, temos a representação de um **sinal digital.**

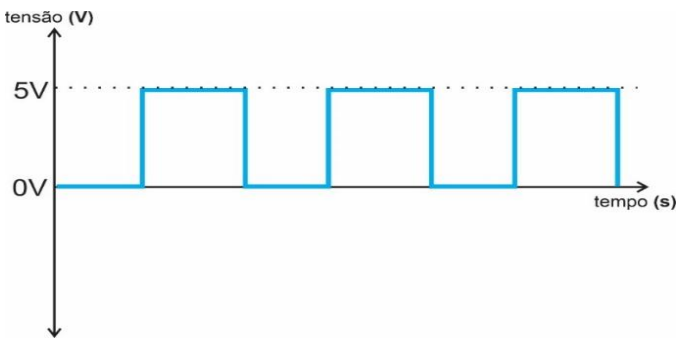


Figura 13 - Sinal Digital

> ***DEFINIÇÃO:*** *Sinal digital é aquele que possui apenas dois valores de amplitude no decorrer do tempo.*

SINAIS ANALÓGICOS E DIGITAIS – situações cotidianas

ANALÓGICO

Os diversos valores de amplitude sonora produzidos pela emissão da voz humana fazem com que ela possua características analógicas de emissão, ou seja, assume diversos valores de intensidade no decorrer do tempo.

DIGITAL

Uma mulher pode estar ou não grávida, dessa forma essa informação só possui dois estados possíveis, sim ou não, portanto, dizemos que a gravidez é um estado que assume valores digitais, 1 ou 0, sim ou não.

ANÁLISE DE SINAIS ANALÓGICOS E SINAIS DIGITAIS – comparações

Ao observar os dois gráficos abaixo é possível perceber que o sinal analógico possui inúmeros valores de amplitude no decorrer do tempo enquanto o sinal digital possui apenas dois valores.

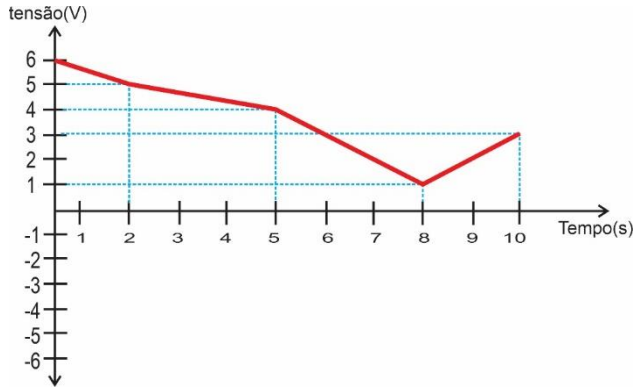


Figura 15 - Sinal Analógico

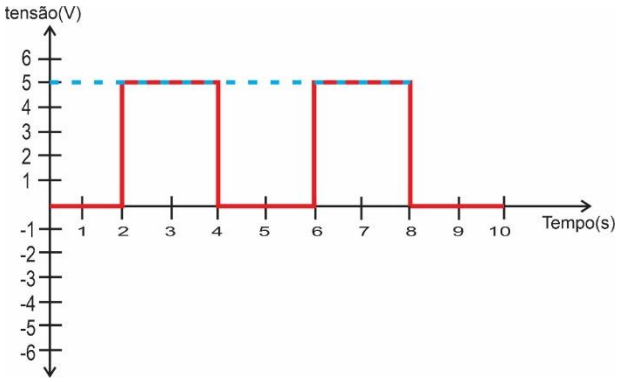


Figura 14 - Sinal Digital

Os sinais digitais e analógicos além de possuírem características de amplitude no decorrer do tempo, também possuem características de polaridade, e como já estudamos anteriormente, a alternância de polaridade definem se um sinal pode ser alternado ou contínuo.

Você então deve estar pensando que, além de classificarmos um sinal como analógico ou digital, podemos também classificar esses mesmos sinais como sendo alternados ou contínuos, acertei? Pois é, vejamos os gráficos a seguir:

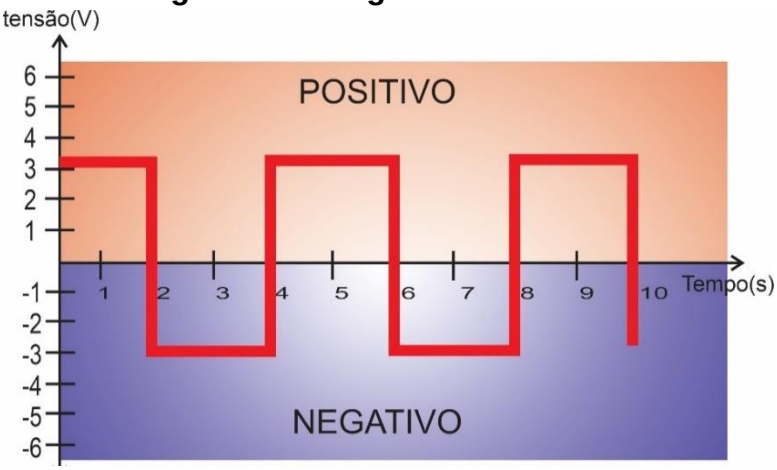


Figura 16 - Sinal digital alternado

Sinal com duas amplitudes no decorrer do tempo, +3V e -3V (digital). Como possui duas polaridades no decorrer do tempo, tempos um ***sinal digital alternado.***

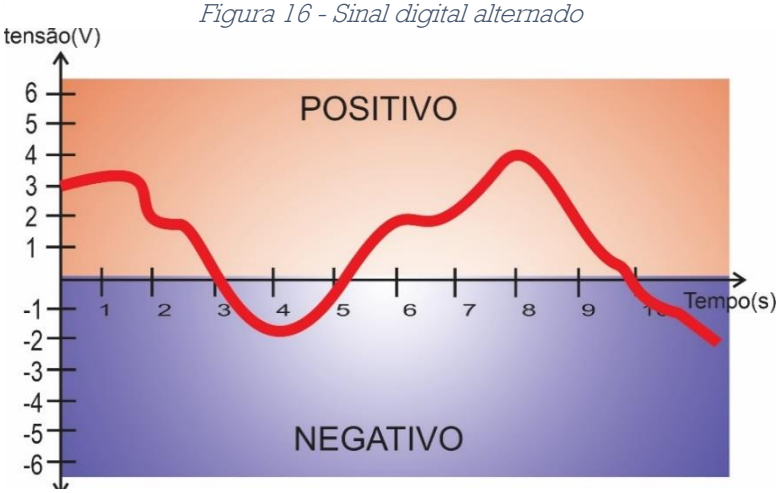


Figura 17 - Sinal Analógico Alternado

Sinal com diversos níveis de amplitude no decorrer do tempo. Como o sinal possui duas polaridades no decorrer do tempo, temos um ***sinal analógico alternado.***

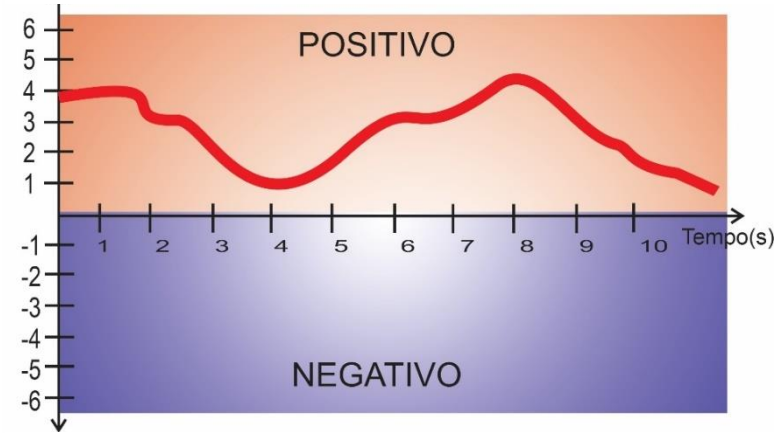


Figura 18 - Sinal Analógico Contínuo

Sinal com diversos níveis de amplitude no decorrer do tempo. Como o sinal possui apenas uma polaridade no decorrer do tempo, tempos um ***sinal analógico contínuo****.*

O QUE É UM BIT?

A palavra BIT (b) é uma sigla que vem de **Bi**nary Digi**t (BIT).**

É a menor unidade de informação que pode ser transmitida ou armazenada. O BIT é uma unidade digital que assume apenas 2 valores, 1 e 0.

bit = 1 ou 0

Quando um bit assume **nível alto (HIGH),** seu valor digital é **1**

Quando um bit assume **nível baixo (LOW),** seu valor digital é **0**

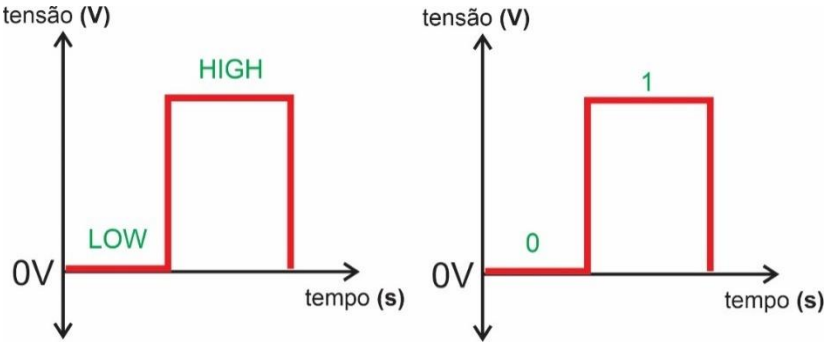


Figura 19 - Estados digitais

Mas o que é uma unidade de informação?

Você deve estar se perguntando, mas na prática, o que é uma informação? O que esses bits 0 e 1 representam? Os bits 0 e 1 na prática simplesmente representam um determinado valor de tensão, onde para nível 1 teremos um valor de tensão, e para nível 0 outro valor de tensão. Esses valores de tensão podem variar de acordo com o projeto, mas em se tratando de tecnologia digital, sempre serão apenas dois valores.

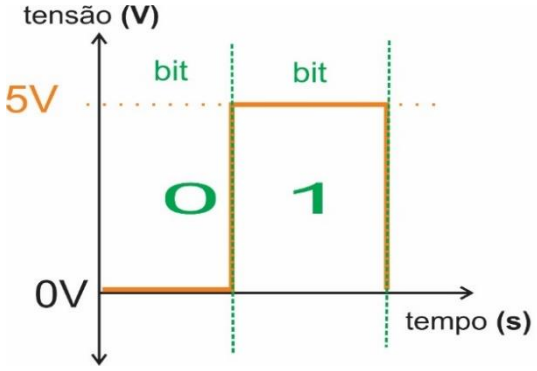


Figura 20 - Níveis de tensão para representar os bits

No gráfico acima podemos observar 2 bits de informação. Cada bit corresponde a um nível de tensão, sendo **nível 0** correspondente à **tensão de 0V** e o **nível 1** correspondente à tensão de **5V.**

Veja então que os bits nada mais são do que níveis de tensão para representar estados digitais altos e baixos.

Observe abaixo outros exemplos de diferentes níveis de tensão para valores digitais.

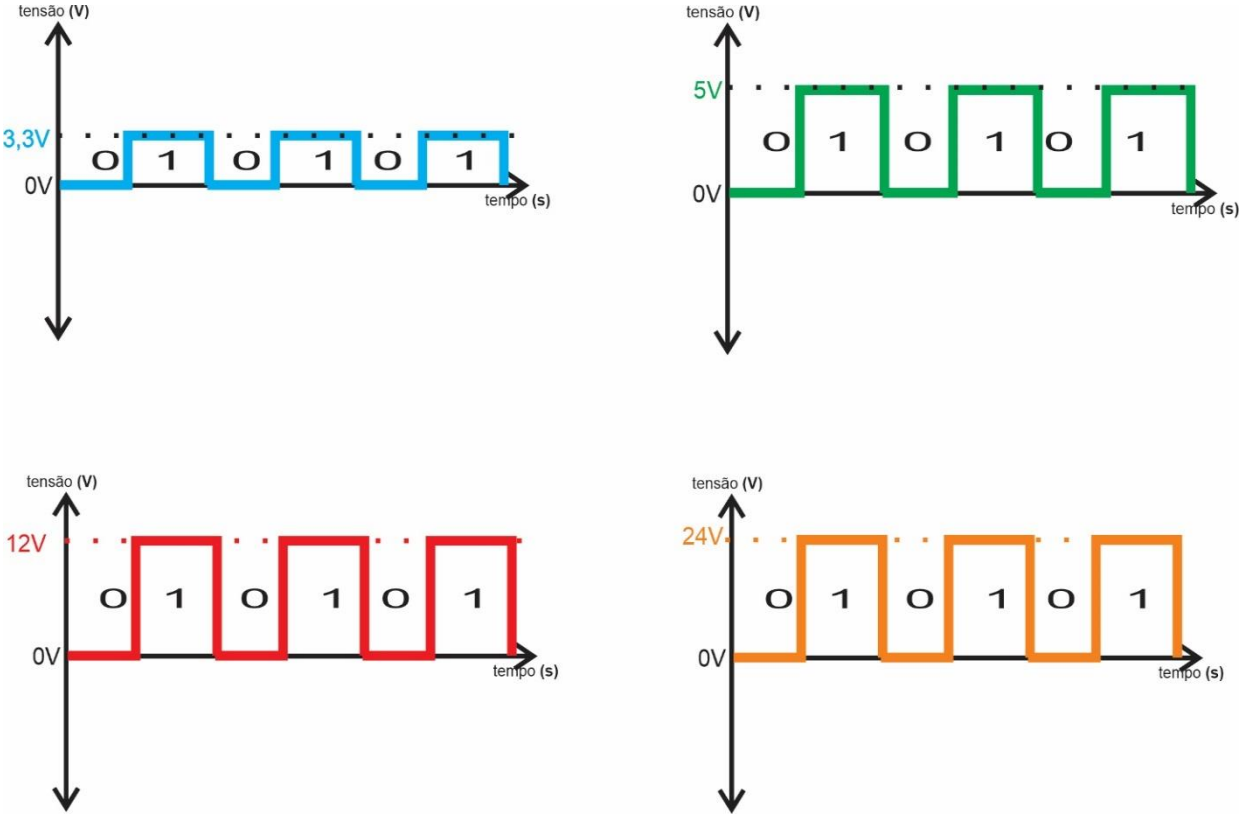


Figura 21 - Diferentes níveis de tensão representando os bits 0 e 1

O QUE É UM BYTE?

A palavra BYTE (B) vem de **B**inar**y** **Te**rm **(BYTE).**

> BYTE é o conjunto de 8 bits. Portanto, a cada **8 bits** de informação teremos **um byte**.

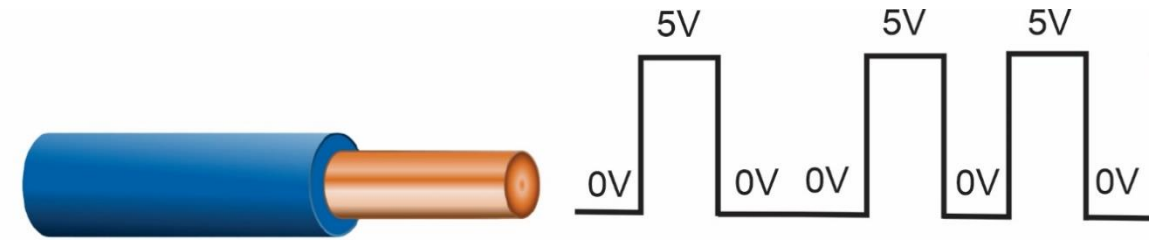


Figura 22 - 8 bits completam 1 byte

Veja no exemplo acima que o fio condutor enviou 8 níveis de tensão, e assim transmitiu 8 bits de informação. Podemos dizer que temos aqui a representação de uma transmissão de 8 bits de informação ou **1 BYTE.**

Já sabemos que um nível de tensão pode representar um valor digital alto ou baixo, mas agora você deve estar se perguntando, como diferenciar os bits transmitidos com valores iguais, como por exemplo 0V e 0V que estão em sequência um do outro, como saber que ali tem dois bits enviados e não apenas um bit com nível baixo?

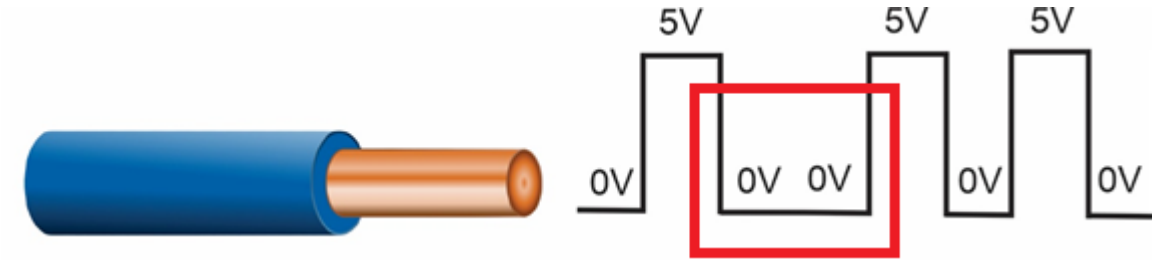


Figura 23 - Bits iguais em sequência

É muito simples. Há um “acordo” de tempo de leitura e envio entre o emissor e o receptor, eles “combinam” uma velocidade de transmissão desses valores de tensão, assim, o tempo de envio e leitura serão os mesmos diferenciando os bits de mesmo valor.

Vamos imaginar que o condutor libera um valor de tensão a cada 1 segundo, logo, a leitura dessa tensão deverá acontecer a cada 1 segundo também, assim iremos fatiar essa transmissão de dados em porções de 1 segundo e conseguiremos identificar separadamente todos os bits enviados.

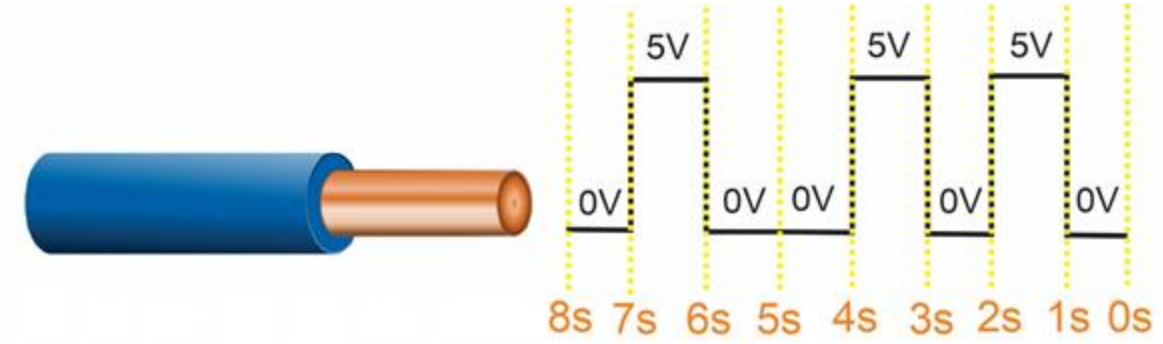


Figura 24 - Envio de bits a cada 1 segundo

Veja no gráfico acima que agora os bits ficaram separados em função do tempo, e mesmo que haja uma sequência de vários bits iguais, a leitura ocorrerá a cada 1 segundo separando cada bit enviado.

No nosso exemplo temos a transmissão de 1 BYTE a cada 8 segundos, ou 1 bit a cada segundo.

O QUE É UM NIBBLE?

Já vimos que o bit é a menor informação possível em eletrônica digital. Também vimos que quando temos um conjunto de 8 bits, temos o famoso BYTE.

Agora, imaginem que eu preciso de apenas um pedaço de informação desse byte, eu quero apenas uma “mordida” de um BYTE.

Esse pequeno pedaço “mordido” chamaremos de **NIBBLE.**

Um nibble é uma “mordida” que possui **4 BITS** de informação.

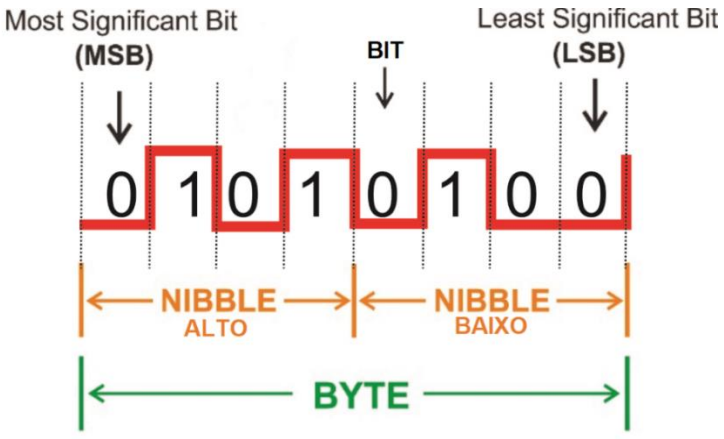


Figura 25 - Divisão de 1 byte em diferentes partes

CHOCOLATE DIGITAL – Veja abaixo como um BYTE pode ser dividido

Figura 26 - Chocolate digital

LSB – bit menos significativo

MSB – bit mais significativo

O bit mais significativo ou menos significativo está relacionado a posição em que ele se encontra na sequência binária.

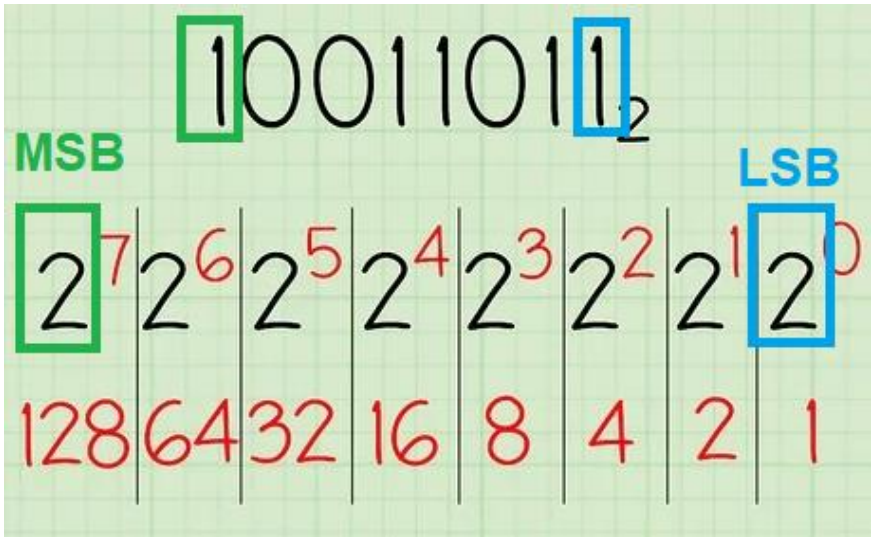


Figura 27 - Representação MSB e LSB de acordo com sua posição da representação binária, entenderemos perfeitamente essa tabela nos estudos futuros sobre conversão binário-decimal.

SINAL DISCRETO E SINAL CONTÍNUO

Sinais discretos possuem dois estados, ligado ou desligado. É comparável ao sinal digital, 0 e 1.

Sinais contínuos são sinais que possuem variação de intensidade, ou diversos valores no decorrer do tempo, e é comparável aos sinais analógicos.

Uma lâmpada pode, por exemplo, ser analisada de forma **discreta e contínua**, pois pode estar ***ligada ou desligada***, 0 ou 1, e também assumir um **nível de intensidade que pode ser fraca, média ou forte.**

SOLDAGEM ELETRÔNICA

Ao realizar qualquer atividade relacionada com eletroeletrônica envolvendo hardwares, a **solda eletrônica** estará sempre presente. Soldar faz parte do dia-a-dia dos curiosos, *makers* e profissionais desta área. Com as dicas deste módulo, juntamente com a sua dedicação e **prática**, facilmente você desenvolverá as habilidades necessárias para realizar um projeto ou até mesmo a manutenção em placas eletrônicas. Fique ligado, pois, esse módulo é essencial para a montagem dos projetos ao decorrer do curso.

Conhecendo a solda Sn/Pb

A solda comum aqui no Brasil é composta por uma liga metálica de **estanho** e **chumbo** *Sn/Pb*. Em certas situações, são adicionados materiais específicos às ligas que têm particularidades em atender desafios característicos de operação, como por exemplo trabalho em altas temperaturas ou alta resistência mecânica.

O chumbo contido na solda comum é prejudicial aos seres vivos e principalmente aos seres humanos. Quando são expostos de forma agressiva por um longo período de tempo, ou seja, trabalham constantemente sem proteção com a substância, podem sofrer danos irreversíveis à saúde. A maioria dos países já não aceita mais produtos eletroeletrônicos que contenham chumbo na composição da solda, como os Estados Unidos e boa parte da Europa. Foi padronizado então a solda ***"lead free"*** (*solda livre de chumbo*), para a produção de equipamentos e dispositivos eletroeletrônicos em quase todo o mundo.

Os formatos da solda e modos de aplicação variam de acordo com cada necessidade. A mais conhecida é a solda em formato de fio, mas exitem também outros tipos de solda, como a solda em esferas e o banho de solda.

A solda que vamos utilizar neste curso é a solda tradicional, usando o fio de estanho, que podem conter ou não fluxo interno. Ah! E o nosso famoso companheiro: **ferro de soldar**.

O que é um ferro de soldar?

O ferro de soldar é a ferramenta responsável por fundir a liga Sn/Pb, ou mais conhecido como estanho. O ferro de soldar tem diversos modelos que atendem cada necessidade. Os modelos mais usados são: O **ferro de solda independente** e a **estação de solda**, essa pode vir com recursos adicionais como, por exemplo, um soprador térmico (figura 10) Veja alguns modelos conhecidos abaixo:

Figura 16 – Ferro de solda - <ncenter.com.br/>

Figura 17 – Estação de solda - <ncenter.com.br/>

Através da aplicação do fio entre a ponta do ferro de soldar, o terminal do componente e a ilha localizada na placa, tem-se então a aplicação correta da solda. Para efeito de **preservação**, é importante manter sempre a ponta do ferro estanhada, para evitar oxidação. Vale lembrar que, se a ponta for danificada, ela pode ser substituída.

Figura 18 Pontas ferro - <eletronicacastro.com.br>

Conhecendo o "sugador de solda"

O sugador de solda serve para remover os excessos de solda ou até mesmo auxiliar no processo de dessoldagem, que é remover um componente já fixado.

Figura 19 – Sugador de solda

Para utilizar o sugador basta apenas pressionar a mola, aquecer o ponto que deseja remover excesso, posicionar a região de sucção e pressionar o disparo.

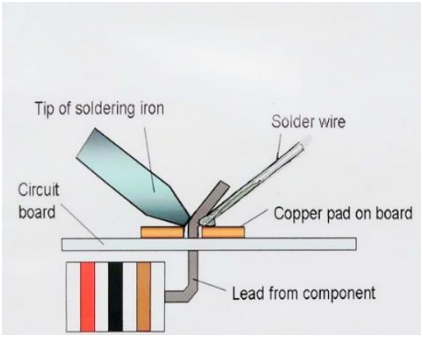


Figura 20 – Sugador <endigital.orgfree.com>

***Quando o sugador não estiver em uso, deixar a mola em estado normal para evitar degradação do equipamento.**

O passo-a-passo de uma solda de sucesso

1º. Antes de tudo é **imprescindível** realizar uma boa **limpeza** no local que for realizar a solda, pois quanto mais limpa estiver a placa (*as ilhas de solda*) maior será a **aderência da solda** e uma solda aderente evita mau contato.

Utiliza-se comumente: **álcool isopropílico**, **escova** e **lenço para limpeza para fazer esta atividade**.

Figura 21 – Álcool <implastec.com.br>

Figura 22 – Lenço Limpeza <implastec.com.br>

Figura 23 – Escova Limpeza <implastec.com.br>

2º. Agora é hora de adicionar o **fluxo de solda**. Este produto elimina a **camada de óxido** *(*camada corrosiva*)* que se forma nas ilhas de cobre, conserva a placa e facilita muito o processo de soldagem. É indicado, principalmente, em situações onde a solda será realizada com estanho em **fio sem fluxo interno**. Não se esqueça de tomar **CUIDADO**, pois o fluxo é inflamável, mantenha-o distante do ferro de soldar e outro objetos que possam causar aquecimento ou faíscas.

Figura 24 – Fluxo Líquido <implastec.com.br/>

Figura 25 – Fluxo Pastoso <baudaeletronica.com.br/>

3º. Depois de ter feito a limpeza da placa e já adicionado o fluxo de solda, é o momento de **realizar a solda**. Confira algumas dicas importantes:

I. É crucial deixar o ferro de soldar **atingir** a **temperatura de fusão**. É importante saber que, a temperatura de aplicação de cada solda varia de acordo com a liga (Sn/Pb), atente-se a isso;

II. Aplique **solda na ponta do ferro de soldar**, isso facilitará no processo. No entanto, evite deixar **excesso de solda na ponta**, visto que pode formar uma **gota** e esta cair sobre o circuito causando curto-circuito e, consequentemente retrabalho;

III. Siga o passo a passo para efetuar uma solda com **SUCESSO**!

Figura 26 – Adaptado de: <blog.adafruit.com/2016/12/26/new-reference-card-soldering-101/>

IV. Sua solda não saiu como esperava? Não se preocupe! Veja quais os tipos de soldas defeituosas e **como corrigir**:

V.

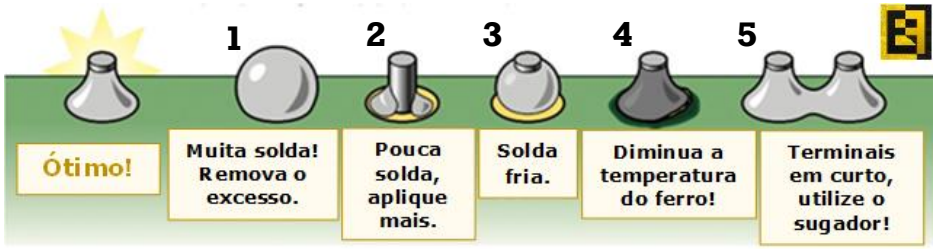


Figura 27 - Adaptado de: <blog.adafruit.com/2016/12/26/new-reference-card-soldering-101/>

1° Caso: Pecou pelo excesso! Quando se utiliza um fio de estanho muito espesso para a proporção dos componentes, terminais e ilhas o axcesso de solda pode acontecer, ou até mesmo quando estamos aprendendo a soldar. Mas não se preocupe, bata utilizar o sugador e remover a solda e fazê-la novamente!

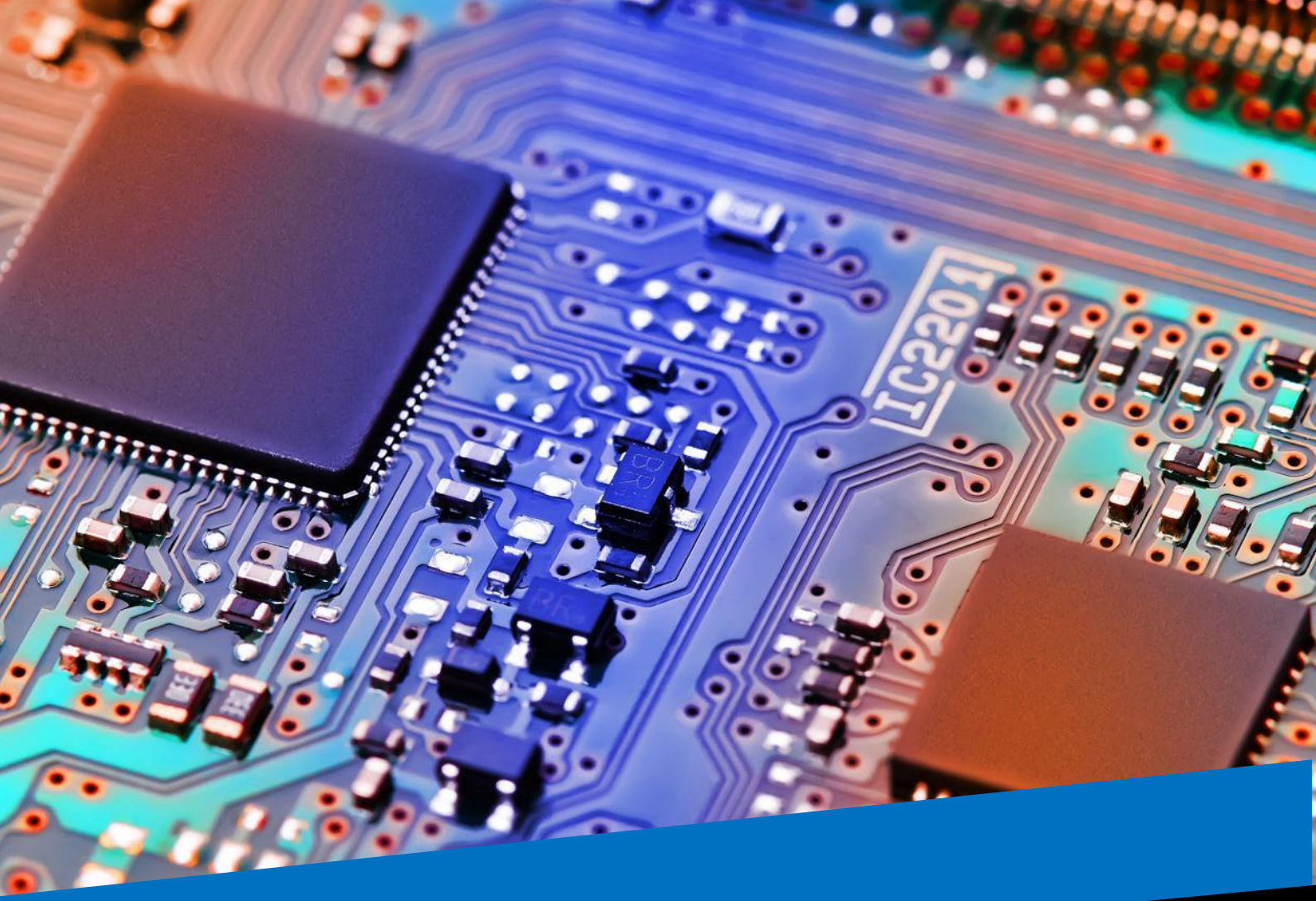
2° Caso: Falta adicionar solda! **O ideal é que a ilha esteja preenchida totalmente**. Essa falta de solda é um dos maiores responsáveis pelos maus contatos.

3° Caso: A solda fria é quando **não existe uma fusão perfeita entre solda, ilha e terminal do componente**. A maior causa da solda fria é a oxidação das ilhas e a oleosidade, por isso não dispense a limpeza da sua placa e utilização do fluxo de solda na hora de soldar.

4° Caso: Uma temperatura elevada do ferro de solda, de 400ºC a 450ºC também pode ser danosa ao processo de soldagem. A temperatura de soldagem varia de acordo com a liga do fio de estanho utilizado, por exemplo, a liga 60/40 ou a 63/37 que são as mais utilizadas atualmente, possuem **ponto de fusão em +/-183 °C**, porém para que a soldagem aconteça com sucesso, três pontos devem ter a mesma temperatura, o ferro de soldar, o terminal do componente e a ilha de solda, ou seja, para realizar esse aquecimento é aconselhável que a temperatura do ferro de soldar esteja entre 280ºC e 330ºC, com o objetivo de equilibrar termicamente os três pontos em quatro segundos aproximadamente, proporcionando uma solda de qualidade.

Para evitar uma temperatura excessiva na ponta do ferro de soldar, verifique se a tensão dele é compatível com a da fonte de energia (*tomada*), ou até mesmo, para aqueles que podem investir, vale a pena adquirir uma *estação de solda* onde a temperatura é regulada facilmente de acordo com a situação em questão.

5° Caso: Os terminais foram unidos causando um curto-circuito, e agora? Nesse tipo de situação **utiliza-se o sugador de solda** e realiza-se a solda novamente. Para evitar esse problema vale a pena adquirir um fio de estanho com uma espessura ideal para a situação.



É Hora de Estudar e Fazer o Seu Resumo de Aula

Aprender = Ler + Ouvir + Ver + Escrever + Fazer e Compartilhar

M1

CURSO DE
ELETRÔNICA FÁCIL

Módulo:

Título da Aula:

Faixa:

Data:

Realizar um resumo da aula anotando abaixo todos os principais pontos aprendidos e relevantes:

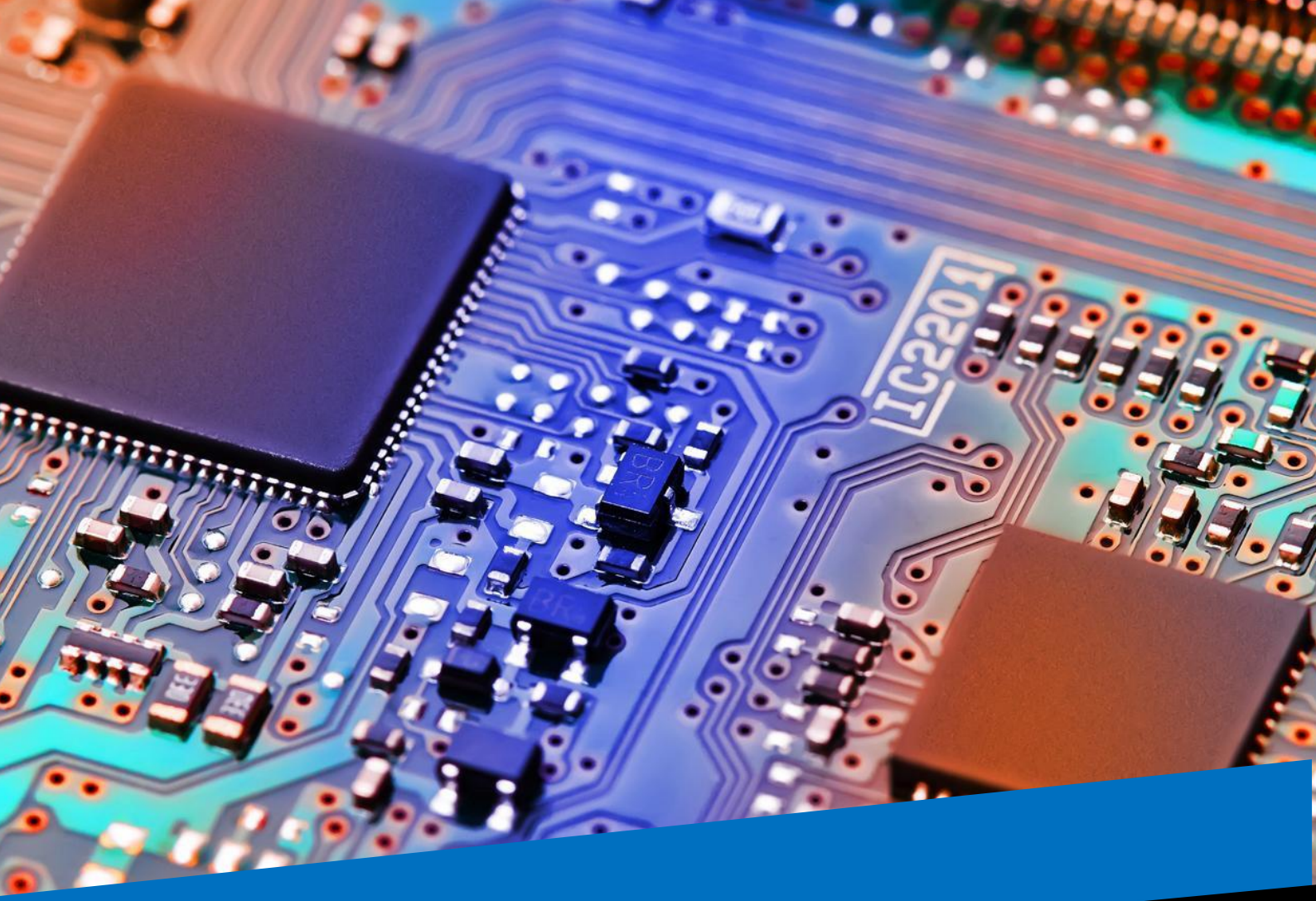
Anote abaixo: O que você aprendeu agora e que nunca tinha visto antes?

Escreva a Conclusão do Módulo em Apenas 1 parágrafo:

Quais foram suas dúvidas neste módulo do curso? Anote abaixo para postar na plataforma do Curso e com isso a equipe possa te responder, gerando mais valor para todos alunos!

Não esqueça de compartilhar todo seu conhecimento aprendido com os demais alunos do curso, pois quanto mais você compartilha, ensina e ajuda o proximo, mais você aprende e evolui no fantástico mundo da eletrônica!

Um Abraço do Professor Rodolpho e de toda a equipe e #tamojunto #familia #familaeletronicafacil #juntossomosmaisfortes #faixabranca



CIRCUITOS COMBINACIONAIS

LÓGICAS E PORTAS LÓGICAS

M2

CURSO DE
ELETRÔNICA FÁCIL

MÓDULO 2 - COMEÇANDO COM O PÉ DIREITO

No módulo 2 iremos dar sequência aos conceitos aprendidos nos módulos 1, por isso não é recomendado que o aluno venha diretamente para o módulo 2. Como sempre alertamos, quem pula é canguru, no incrível mundo da eletrônica pular etapas pode ser fatal para que o entendimento NÃO aconteça de forma show de bola e de maneira integra, por isso, se você já estudou o módulo 1 e está por dentro de tudo que falamos anteriormente, seja muito bem vindo a essa nova etapa do nosso treinamento sobre sistemas combinacionais. #vamospracima

CIRCUITOS COMBINACIONAIS

Circuitos combinacionais *são aqueles em que o valor da saída depende unicamente dos valores combinados em suas entradas.*

Um sistema combinacional possui basicamente dispositivos de entradas, dispositivos de saídas, e um cérebro digital que faz a gestão das decisões a partir de combinações desses sinais.

Figura 28 - Sistema combinacional genérico

Dispositivos de entrada – *input*

Dispositivos de entrada *são dispositivos preparados para receber, captar informações e enviar para o cérebro digital, para que seja feita a gestão de funcionamento de acordo com as informações programadas.*

Para que o entendimento sobre dispositivos de entrada seja bem didático, vamos fazer uma análise do nosso corpo. O corpo humano possui diversos dispositivos de entrada que ajudam o cérebro a tomar decisões. Veja abaixo.

Figura 29- Dispositivos de entrada em um corpo humano

TATO – Envia ao cérebro informações obtidas através do toque, como temperatura, e texturas, e principalmente se alguma ameaça está acontecendo ao corpo humano, como picadas, arranhões, etc.

VISÃO – A visão é responsável por enviar ao cérebro informações obtidas através da luz, como distancia, cores, formas e etc.

PALADAR – O paladar é responsável por enviar ao cérebro informações obtidas através da degustação, caracterizando substancias como salgadas, doces, azedas, picante, etc.

AUDIÇÃO – A audição envia ao cérebro todas as informações obtidas em forma de som, como vozes, pássaros, trovão, etc.

OLFATO – O olfato é responsável por enviar ao cérebro informações obtidas por meio do ar, como cheiros e aromas.

Podemos então observar que o corpo humano possui sensores (dispositivos de entrada) que enviam diferentes informações ao cérebro ajudando na interpretação das diferentes situações cotidianas e principalmente sendo fundamental nas tomadas de decisões.

Mas, e depois das tomadas de decisões, o que seriam os dispositivos de saída no corpo humano?

Dispositivos de saída - *output*

Dispositivos de saída são elementos controlados pelo cérebro a partir das informações obtidas pelos dispositivos de entrada.

Encostar o dedo na panela quente – Ao encostar o dedo indicador em uma panela quente, imediatamente a sensibilidade do tato (dispositivo de entrada) irá enviar uma informação ao cérebro, que irá reconhecer que o material está em uma temperatura perigosa, acionando então os dispositivos de saídas como mãos, braços e pernas, retirando imediatamente o corpo daquele lugar.

Dessa forma podemos concluir que o dispositivo de entrada é o famoso **"dedo duro",** e o dispositivo de saída é o famoso **"pau mandado".** O dedo duro avisa o cérebro do que está acontecendo, e o pau mandado obedece a decisão tomada pelo cérebro depois das informações obtidas.

Dispositivos de INPUTs – "O FAMOSO DEDO DURO".

Os dispositivos de entrada, aqui chamados carinhosamente de "dedo duro", são os dispositivos que vão avisar o cérebro digital do que pode estar acontecendo em uma

determinada etapa do circuito eletrônico, ou até mesmo fazendo uma análise de condições físicas externas ao circuito e enviando ao cérebro os dados obtidos.

A partir de agora, vamos analisar alguns exemplos de dispositivos de entrada utilizados no mundo da eletrônica, ou seja, dispositivos que enviam informações ao cérebro digital para que decisões sejam processadas.

TECLADO – O teclado do computador envia informações de caracteres para que o cérebro digital possa registrar e interpretar códigos enviados pelo usuário.

CÂMERA – A Câmera converte a luz em impulsos elétricos enviando sinais que serão interpretados pelo cérebro digital e convertidos novamente em imagem.

CHAVE ON/OFF – Um botão de dois estágios é um dispositivo de entrada controlado pelo usuário que enviará uma informação a ser interpretada pelo cérebro digital.

SENSOR DE PRESENÇA – Este dispositivo irá identificar a presença de objetos ou pessoas em determinado local, enviando informações dessas identificações para o cérebro digital.

Dispositivos de INPUTs discretos e contínuos.

Os dispositivos de entrada, ou inputs, podem ser divididos em dispositivos discretos (digitais) ou dispositivos contínuos (analógicos). Essa divisão acontecerá a partir do tipo de sinal que está sendo enviado para o cérebro digital fazer a interpretação.

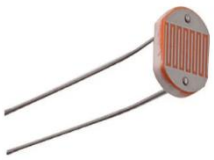
Um **dispositivo de entrada discreta** trabalha com sinais digitais, que podem variar entre os estados 0 e 1, ou seja, os sinais enviados ao cérebro digital a partir desse dispositivo só poderá ter dois estados, ou alto, ou baixo, ou 0 ou 1, ou HIGH ou LOW. Veja alguns exemplos.

CHAVE TÁCTIL – Dispositivo de entrada que pode enviar apenas dois estados de sinal, ou HIGH, ou LOW, sendo dessa forma caracterizado por dispositivo de entrada discreto ou digital.

CHAVE FIM DE CURSO COM ROLETE – Dispositivo de entrada utilizado em portas, gavetas, indicando o final de um percurso mecânico. Esse dispositivo só trabalha com dois sinais possíveis, ou ativado, ou desativado, sendo dessa forma caracterizado por dispositivo de entrada discreto ou digital.

Já os **dispositivos de entrada analógicos ou contínuos**, são dispositivos que podem enviar diferentes valores de amplitude para o cérebro digital e não mais apenas 0 e 1 como os digitais. Veja alguns exemplos.

POTÊNCIOMETRO – Dispositivo que varia sua resistência de acordo com a disposição mecânica do cursor, enviando dessa forma diferentes valores de amplitude para o cérebro digital.



LDR – Dispositivo que varia sua resistência de acordo com a luminosidade, muito utilizado em dispositivos de iluminação noturna. De acordo com a iluminação recebida pelo LDR, diferentes amplitudes de sinais podem ser enviadas ao cérebro digital para a gestão de funcionamento do circuito seja feita com sucesso.

Dispositivos de OUTPUTs – "O FAMOSO PAU MANDADO"

Os Dispositivos de saída, carinhosamente chamados de **"pau mandado"** são os dispositivos que irão receber ordens do cérebro digital para que executem alguma tarefa. Se o cérebro não ordenar nenhuma atividade, ele fica inoperante a espera de algum comando.

Vamos analisar alguns dispositivos de saída utilizados no mundo da eletrônica.

LÂMPADA – A lâmpada é um dispositivo de saída, que somente irá acender se algum sinal for enviado aos seus terminais. Ou seja, se o cérebro digital mandar tensão ela irá acender, caso contrário ficará apagada esperando o nível de tensão necessário para executar o trabalho.

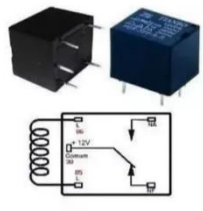
RELÉ – Dispositivo de saída que receberá um nível de tensão em sua bobina, para que dessa forma possa fechar ou abrir um contato, permitindo ou impedindo que a corrente circule.

Dispositivos de OUTPUTs discretos e contínuos.

Os dispositivos de saída, ou outputs, podem ser divididos em dispositivos **discretos (digitais)** ou dispositivos **contínuos (analógicos),** assim como as entradas. Essa divisão acontecerá a partir do tipo de sinal que estará sendo enviado para o dispositivo para que ele possa executar um trabalho. Se o sinal que fará a ativação do dispositivo possuir apenas dois estados, o dispositivo de saída então será chamado de discreto. Se o sinal de ativação do dispositivo tiver variação de amplitude no decorrer do tempo, o dispositivo será chamado de contínuo ou analógico. Veja alguns exemplos.

MOTOR TRIFÁSICO – O motor trifásico pode ter o seu controle de velocidade alterado a partir de diferentes níveis de tensão, o que faz com que esse dispositivo tenha seu funcionamento caracterizado como analógico ou contínuo.

CAIXA DE SOM - A amplitude sonora de uma caixa de som pode variar de acordo com a intensidade do sinal enviado para ela. Sendo assim, a caixa de som é um dispositivo que tem seu funcionamento a partir de sinais analógicos.



RELÉ – O relé é um dispositivo que trabalha com determinado nível de tensão para ser acionado, e a ausência dessa tensão o torna desativa. Ou seja, seu funcionamento possui dois estados, ligado ou desligado. Dessa forma podemos caracterizá-lo como um dispositivo de saída digital.

FECHADURA SOLENÓIDE – A fechadura tem seu princípio de funcionamento em dois estados, ou ativada ou desativada. Dessa forma podemos caracterizá-la como dispositivo de saída digital.

Agora que já entendemos o que são dispositivos de entrada e saída, e como eles podem ser divididos, vamos entender como funcionam os sistemas combinacionais e suas portas lógicas, como é feita a gestão dos sinais e como as decisões são tomadas. #vamospracima

PORTAS LÓGICAS

Para iniciar nossos estudos sobre este tema é preciso antes de mais nada entender o que significa e como é definido esse conceito de portas lógicas.

***Porta lógica** é um dispositivo capaz de receber um ou mais sinais em suas entradas, e, a partir de uma combinação lógica, gerar sinais específicos em suas saídas.*

O valor da saída de uma porta lógica depende exclusivamente dos valores dispostos em suas entradas.

Portas lógicas trabalham somente com níveis lógicos, ou seja, 0 e 1, HIGH e LOW, nível alto e nível baixo.

 SIM (1)

 NÃO (0)

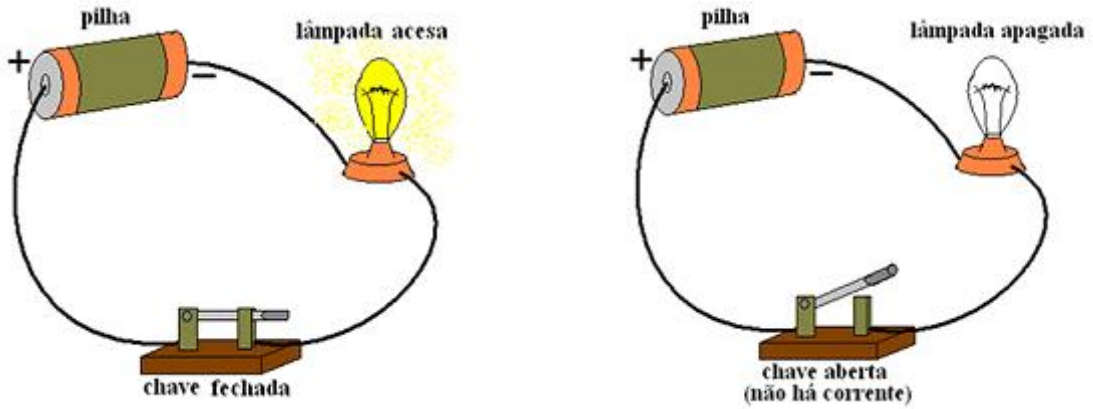
As portas lógicas possuem em seu funcionamento uma LÓGICA. Vamos estudar 4 lógicas principais:

- LÓGICA **SIM**
- LÓGICA **NÃO**
- LÓGICA **E**
- LÓGICA **OU**

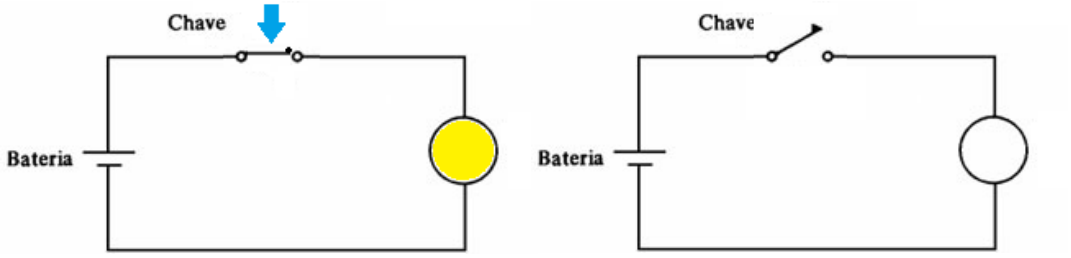
Para iniciarmos nossos estudos das 4 lógicas mencionadas anteriormente, vamos utilizar um pequeno circuito eletrônico utilizando uma lâmpada e um chave de dois estados liga/desliga. A chave será o dispositivo de entrada, o famoso dedo duro, será quem enviará o sinal. A lâmpada será o famoso “pau mandado”, aquele que obedece às ordens do “dedo duro”. Nesse exemplo **ainda não teremos um cérebro digital** fazendo o controle de tudo, resumiremos o circuito para um melhor entendimento.

LÓGICA SIM

A lógica **SIM** coloca em sua saída o mesmo estado lógico de sua entrada. Se a chave for acionada (nível 1) a saída estará em nível 1 também. Se a chave não for acionada (nível 0) a saída também estará em nível 0. A chave que representará a lógica sim é uma chave de contato N.A (normal aberto).



Vamos ver o mesmo circuito em um esquema eletrônico:



Chave acionada (1), lâmpada acesa (1) *Chave desligada (0), lâmpada desligada (0)*

Portanto, na lógica sim, o estado do botão (chave) define o estado da saída.

Veja uma aplicação de lógica SIM com transistor para acender um LED.

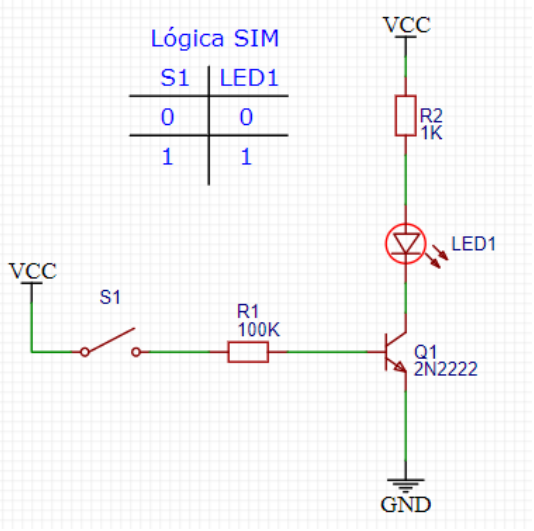
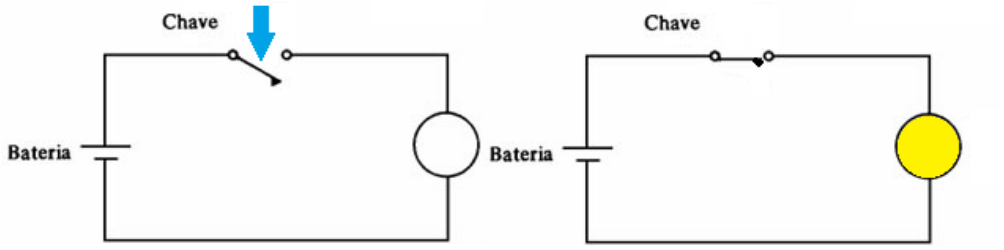


Figura 30 - Se S1 for pressionada, o transistor Q1 será polarizado, passando a funcionar como uma chave fechada e permitindo que a corrente circule pelo LED1, fazendo-o acender.

LÓGICA NÃO

A lógica **NÃO** coloca em sua saída um sinal oposto ao sinal de entrada. Se a chave for acionada (nível 1), a saída terá um sinal oposto (nível 0). Se a chave não for acionada (nível 0), a saída terá sinal em nível alto (nível 1). A chave que representará a lógica NÃO é uma chave de contato N.F (normal fechado)



Chave acionada (0), lâmpada acesa (1) *Chave desligada (1), lâmpada desligada (0)*

Veja uma aplicação de lógica NÃO com transistor para acender um LED.

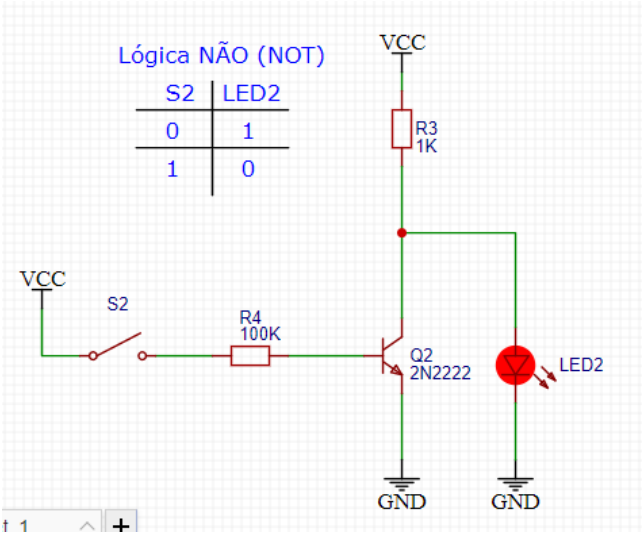
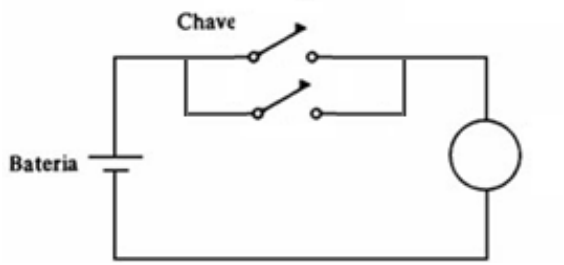


Figura 31 - Se S1 for pressionada, o transistor Q1 será polarizado, passando a funcionar como uma chave fechada curto-circuitando os terminais do LED, fazendo-o apagar. Se a chave não for pressionada, o transistor não será polarizado e o LED se manterá aceso.

Portanto, na lógica NÃO o estado lógico da saída será o inverso do estado lógico da entrada.

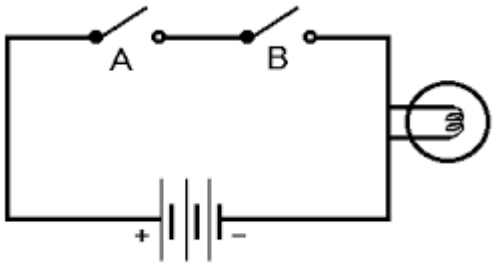
LÓGICA OU

A lógica **OU** pode ser entendida como dois interruptores dispostos paralelamente em um circuito com uma lâmpada. Dessa forma, a lâmpada será acionada se ao menos um interruptor for acionado. Portanto, dizemos que para a lâmpada acender, um **OU** outro interruptor deve ser acionado.



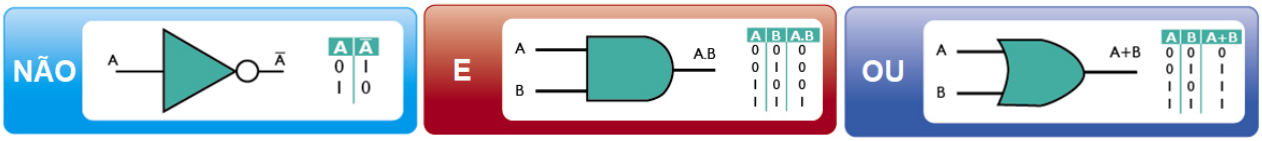
LÓGICA E

A lógica **E** pode ser entendida com dois interruptores dispostos em série. Para que a lâmpada acenda será preciso que os dois interruptores estejam acionados. Por isso, dizemos que para que a lâmpada acenda, os interruptores **a** E **b** devem ser acionados.



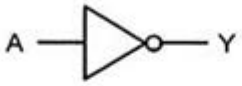
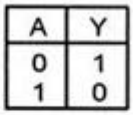
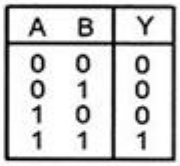
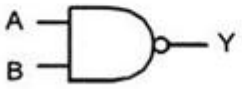
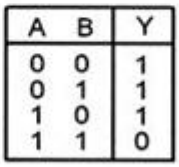
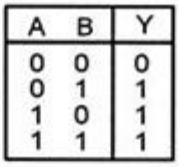
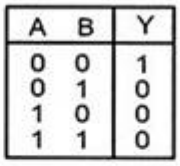
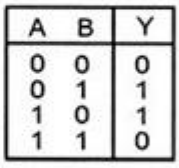
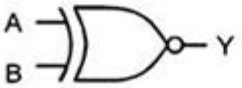
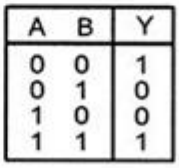
Depois de estudarmos as **lógicas SIM, NÃO, E, e OU**, vamos conhecer algumas simbologias que representam as **portas lógicas** com as funções lógicas descritas anteriormente.

*As l**ógicas** e **portas lógicas OU** e **E** serão estudadas com maior profundidade em momentos futuros, neste momento estamos apenas tomando conhecimento de algumas lógicas existentes para que mais adiante possamos compreender melhor o funcionamento e a forma de aplicação de cada lógica estudada.*



As portas lógicas recebem seu nome em inglês, portanto as portas **NÃO, E, OU**, passarão a ser chamadas de **Not, And** e **Or.**

Apenas a título de conhecimento, já que nesse momento não iremos aprofundar os estudos nesse assunto, veja na tabela abaixo alguns tipos de portas lógicas.

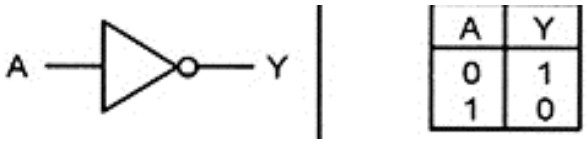
Função lógica	Símbolo lógico	Tabela verdade	Expressão booleana
Porta NOT - Inversora			$Y = \overline{A}$
Porta AND			$Y = A \bullet B$
Porta NAND			$Y = \overline{A \bullet B}$
Porta OR			$Y = A + B$
Porta NOR			$Y = \overline{A + B}$
Porta XOR			$Y = A \oplus B$
Porta XNOR			$Y = \overline{A \oplus B}$

*Agora você deve estar se perguntando, o que é essa tal **tabela da verdade** e **expressão booleana** que apareceu na imagem acima?*

TABELA DA VERDADE

Tabela da verdade é um recurso utilizado para demonstrar como as saídas irão se comportar a partir de diferentes níveis lógicos de entrada. Como sabemos, as saídas lógicas estão diretamente ligadas e dependentes das entradas, dessa forma, a tabela irá mostrar as combinações lógicas possíveis para que se consiga diferentes resultados em suas saídas.

Veja este exemplo:



A	Y
0	1
1	0

Figura 32 - porta lógica NOT

A porta NOT, também apresentada como porta NÃO, tem sua entrada representada pela letra A e sua saída representada pela letra Y;

Veja na tabela que, quando o sinal da entrada A é 0, a saída Y é 1; quando o sinal da entrada A é 1, a saída Y é 0. A tabela conseguiu mostrar todas as combinações possíveis e os diferentes valores obtidos para esta porta lógica.

As letras que indicam as entradas e saídas das portas lógicas podem variar, suas escolhas não possuem uma regra especifica.

É possível calcular o número de possibilidades de saída de uma determinada porta lógica, ou seja, o número de linhas da tabela da verdade?

O número de combinações possíveis, ou o número de linhas que a tabela da verdade irá possuir, poderá ser obtida de maneira bem simples. Como cada entrada da porta terá somente dois estados possíveis, estamos trabalhando com possibilidades binárias, ou seja, base 2 (0 e 1). Basta então elevar 2 ao número de entradas da porta lógica.

$$2^{(n° \text{ entradas})}$$

Por exemplo, uma porta lógica com 3 entradas terá seu número de linhas a partir de: $2^3 = 8$. Portanto, 8 linhas. Veja abaixo:

ENTRADAS			SAIDA
C	B	A	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

A partir da tabela da verdade, surge uma questão: seria possível escrever matematicamente a lógica obtida na tabela da verdade?

Sim, para isso existe a chamada ***EXPRESSÃO BOOLEANA.***

EXPRESSÃO BOOLEANA

Expressão booleana, ou álgebra Booleana, é uma pequena fórmula que surge para representar matematicamente o funcionamento da lógica utilizada em uma porta. É uma forma de indicar como determinada porta lógica trabalha, poderíamos dizer que é um método abreviado de mostrar o que está acontecendo um circuito lógico.

Observando expressão booleana podemos perceber que ela representa exatamente o funcionamento lógico da tabela da verdade:

IN	OUT
A	Y
0	0
1	1

$Y = A$

A SAÍDA É IGUAL A ENTRADA

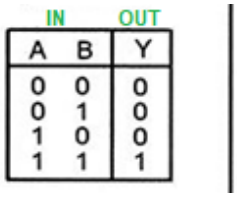
LÓGICA SIM

IN	OUT
A	Y
0	1
1	0

$Y = \overline{A}$

A SAÍDA É IGUAL AO INVERSO DA ENTRADA

LÓGICA NÃO

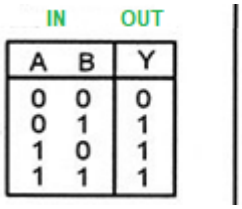


IN		OUT
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Y = A•B

A SAÍDA SERÁ 1 SE A **E** B FOREM 1

LÓGICA E



IN		OUT
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Y = A + B

A SAÍDA SERÁ 1 SE A **OU** B FOREM 1

LÓGICA OU

*Neste ponto dos nossos estudos já entendemos algumas **funções lógicas**, conhecemos as **portas lógicas**, entendemos o que é uma **expressão booleana** e conhecemos o funcionamento da **tabela da verdade**. É importante ressaltar que o estudo aprofundado desses tópicos se encontra no nosso treinamento de eletrônica digital que pode ser acessado pelo link https://ead.cursoseletronicafacil.com.br/curso/curso-de-eletronica-digital-e-arduino-para-iniciantes/*

Neste material tentamos de maneira mais didática possível sintetizar os conteúdos ensinados no treinamento, de maneira a proporcionar uma experiência de aprendizagem leve tanto para o Aluno EF, como para toda a audiência e seguidores da escola que tiverem acesso a esse material.

MÃO NA MASSA – SIMULAÇÃO EM SOFTWARE

Chegou a hora de simular o funcionamento das portas lógicas estudadas, para isso o aluno pode utilizar qualquer software de simulação de circuitos eletrônicos (utilizaremos o proteus), onde o objetivo será evidenciar no campo prático os conceitos aprendidos anteriormente. Vamos nessa?

Vamos iniciar nossos estudos simulando a porta NOT. Já sabemos que a porta NOT coloca em sua saída o oposto do sinal de entrada. Veja abaixo um simples circuito com uma porta NOT para acender um led.

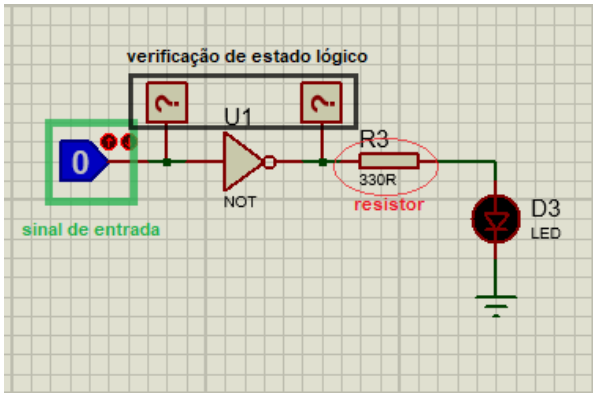


Figura 33 - Elementos da simulação no software proteus

Vamos entender os elementos contidos nesse circuito?

Temos um nível lógico aplicado na entrada da porta lógica. Na entrada e na saída temos um verificador de estado lógico, que fará a confirmação dos níveis lógicos da porta NOT durante a simulação. O circuito possui também um resistor para limitar a corrente que vai para o LED, que será o nosso dispositivo de saída.

SIMULANDO A PORTA NOT

Abaixo temos a simulação da porta NOT ao inserir nível lógico 1 em sua entrada.

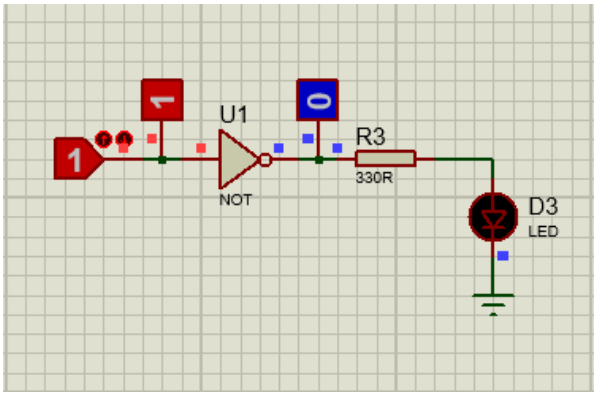


Figura 34 - Sinal de entrada 1, saída 0. Dispositivo de entrada desativado

Simulação ao inserir nível lógico 0 na entrada da porta lógica NOT.

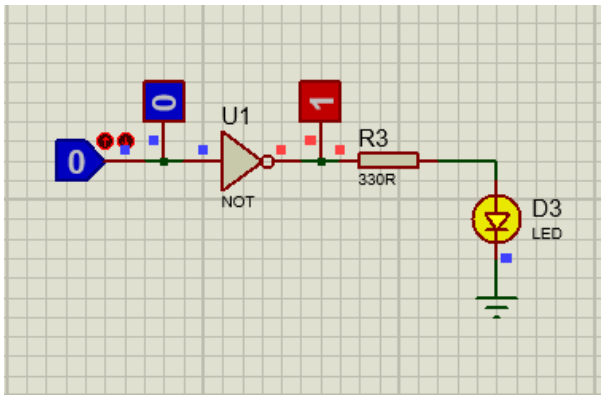


Figura 35 - Sinal lógico de entrada igual a 0, saída em estado lógico 1. Dispositivo de saída ativado.

INDEFINIÇÃO LÓGICA

Perceba que durante as simulações apareceram pequenos quadrados sinalizadores na cor **vermelha** e cor **azul**. Esses quadrados são indicações de estados lógicos nas entradas e saídas dos componentes eletrônicos, vermelho indicando nível lógico 1 e azul indicando nível lógico 0.

Durante a simulação pode aparecer também uma indicação em que não há nem estado lógico 1 definido, nem estado lógico 0, é um estado de indefinição lógica e será indicado pela cor cinza. Mas, por que essa indefinição acontece?

Essa indefinição acontece devido o nível de tensão nesse ponto não estar nem dentro dos limites definidos para nível 0, nem dentro dos limites definidos para nível 1, causando uma interpretação de indefinição lógica e revelando um valor de tensão que se enquadra na chamada **ZONA PROIBIDA**. Estudaremos esses limites mais à frente. Veja a imagem abaixo.

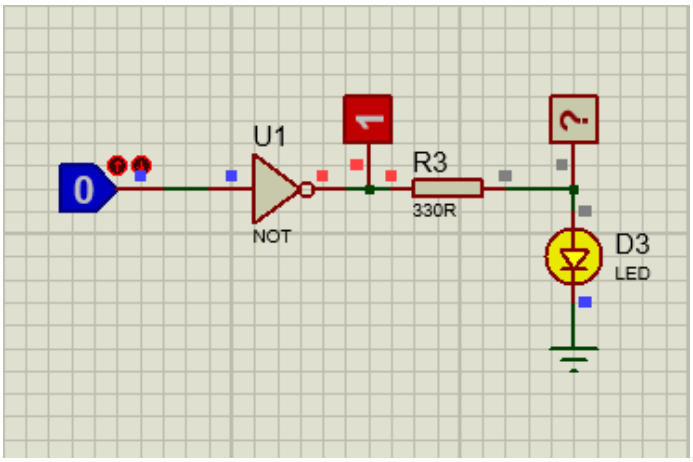


Figura 36 - Ponto de indefinição Lógica

INDICAÇÃO DE CURTO CIRCUITO

Outra indicação muito importante durante as simulações dos circuitos eletrônicos são as indicações de curto circuitos, e são representadas com pequenos sinalizadores amarelos.

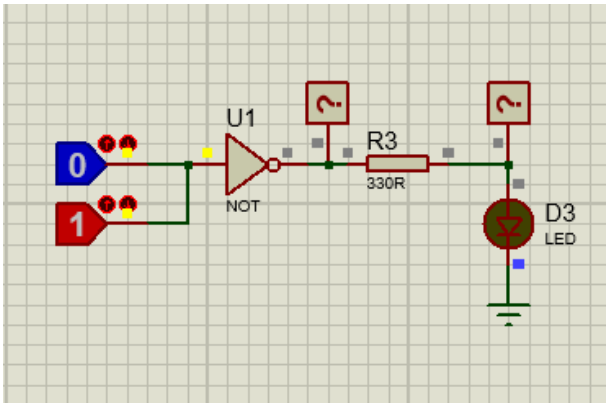


Figura 37 - Curto circuito durante simulação

Veja que foi inserido dois níveis lógicos distintos na entrada da porta NOT, nível 1 e nível 0. Sabendo que cada nível lógico representa um potencial elétrico, tipicamente 0V e 5V no exemplo demonstrado, esse circuito uniu esses dois potenciais e apresentou um curto circuito, sendo indicado pelo simulador com **sinalizadores amarelos.**

O QUE É NÍVEL 1 E NÍVEL 0 PARA O CIRCUITO DIGITAL

Como vimos anteriormente, um circuito digital é muito semelhante ao corpo humano, possui seus dispositivos de entrada, dispositivos de saída e um cérebro digital. Para que tudo isso funcione bem, os circuitos digitais, assim como nós, também precisam se alimentar, precisam receber tensões em seus terminais de alimentação.

Os níveis de tensão de alimentação dos circuitos integrados digitais serão a base para entendermos os níveis de tensão definidos como 1 e como 0. Veja abaixo como definir nível 1 e nível 0 em eletrônica digital.

NIVEL 1 = tensão mais próxima possível da alimentação positiva do CI.

NÍVEL 0 = tensão mais próxima possível da alimentação negativa do CI.

Por exemplo: um circuito integrado digital alimentado com 5V, terá seu nível 1 garantido com tensões bem próximas a esse valor, como 4,5V, 4,7V, quanto mais próximo de VCC melhor.

Todas as informações técnicas sobre um componente eletrônico podem ser consultadas no seu famoso DATASHEET, que nada mais é do que um manual técnico, uma documentação e descrição de todos os parâmetros de funcionamento do componente.

Mas, quais seriam os limites mínimos e máximos aceitáveis para cada nível lógico?

Para entender os limites mínimos e máximos de uma porta lógica vamos analisar o datasheet do circuito integrado 7404. Esse circuito integrado possui 6 portas lógicas inversoras em sua construção. Veja a imagem abaixo:

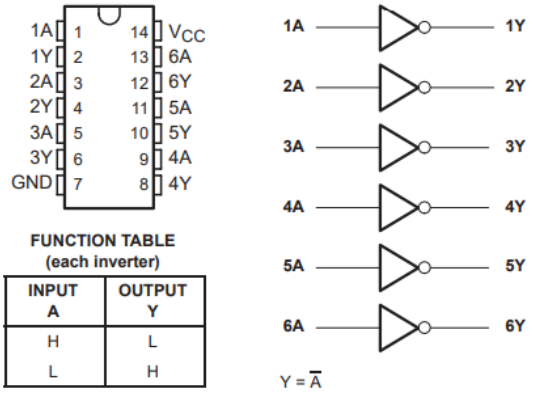


Figura 38 - informações do datasheet do 7404

Na tabela abaixo, encontrada no datasheet do 7404, podemos ver os limites de tensão para que o componente seja alimentado e funcione com sucesso (VCC - Supply Voltage)

recommended operating conditions (see Note 3)

		SN5404			SN7404			UNIT
		MIN	NOM	MAX	MIN	NOM	MAX	
V_{CC}	Supply voltage	4.5	5	5.5	4.75	5	5.25	V
V_{IH}	High-level input voltage	2			2			V
V_{IL}	Low-level input voltage			0.8			0.8	V
I_{OH}	High-level output current			−0.4			−0.4	mA
I_{OL}	Low-level output current			16			16	mA
T_A	Operating free-air temperature	−55		125	0		70	°C

Figura 39 - Figura 37 – Tensão de alimentação

Na tabela, logo abaixo de VCC, o datasheet informa dois parâmetros importantíssimos para entendermos os limites máximos e mínimos de cada nível lógico:

VIH = TENSÃO DE ENTRADA EM NÍVEL ALTO

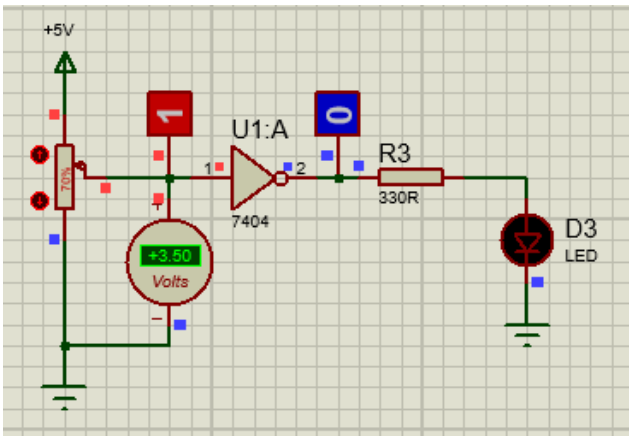
VIL = TENSÃO DE ENTRADA EM NÍVEL BAIXO

recommended operating conditions (see Note 3)

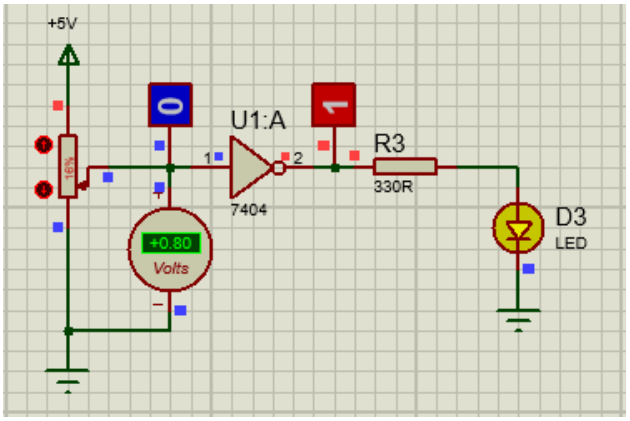
		SN5404			SN7404			UNIT
		MIN	NOM	MAX	MIN	NOM	MAX	
V_{CC}	Supply voltage	4.5	5	5.5	4.75	5	5.25	V
V_{IH}	High-level input voltage	2			2			V
V_{IL}	Low-level input voltage			0.8			0.8	V
I_{OH}	High-level output current			−0.4			−0.4	mA
I_{OL}	Low-level output current			16			16	mA
T_A	Operating free-air temperature	−55		125	0		70	°C

Figura 40 - Tensão de entrada em nível alto e baixo

Para VIH, tensão de entrada em nível alto, o datasheet informa que a partir de 2V o circuito já entenderá como nível 1. Ou seja, qualquer tensão a partir de 2V até a alimentação positiva do circuito integrado será considerada como nível alto pela entrada da porta lógica.



Para VIL, tensão de entrada em nível baixo, o datasheet informa que até 0,8V será considerado como nível 0. Ou seja, qualquer tensão a partir da tensão de alimentação negativa, 0V ou GND, até o nível máximo de 0,8V será considerada como nível baixo pela entrada da porta lógica.



Mas e a região de tensão entre 0,8V até 1,9V? Como serão interpretados esses níveis de tensão?

Essa é a famosa **zona proibida.** Esses níveis de tensão da zona proibida não conseguem ser interpretados e nem definidos pela porta lógica, e quando aplicados em suas entradas podem gerar a chamada **indefinição lógica**, desestabilizando completamente o funcionamento do circuito digital.

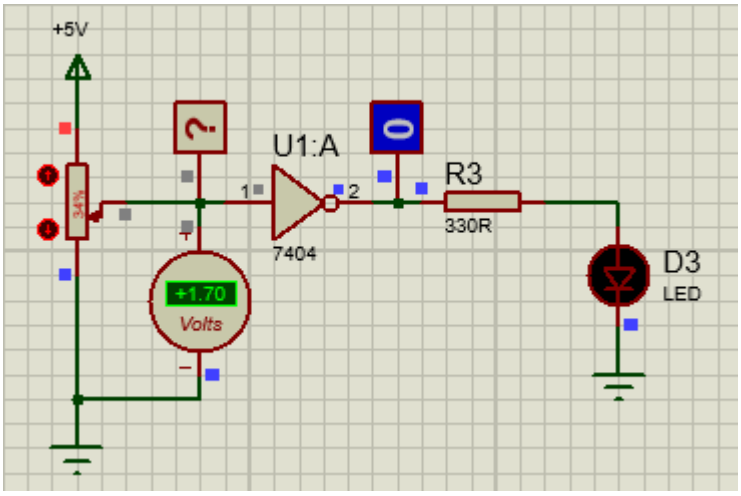


Figura 41- indefinição lógica

Portanto, esses níveis de tensão fora dos limites de reconhecimento da porta lógica **devem ser totalmente evitados em circuitos digitais.**

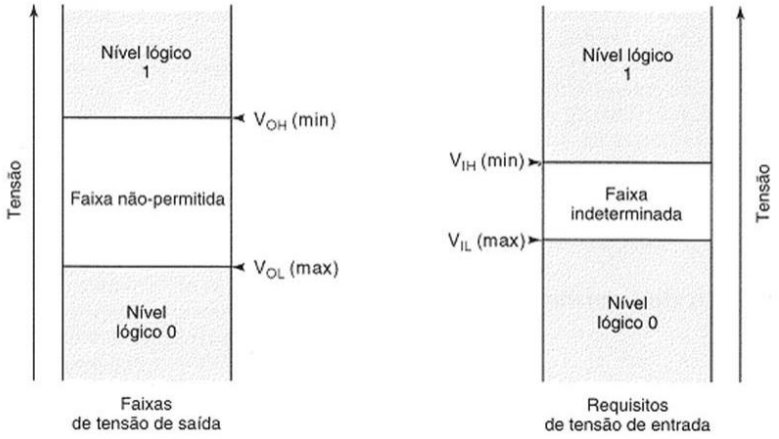


Figura 42 - Demonstração da zona intermediaria, ou zona proibida.

RESISTORES DE PULL DOWN

Imagine a seguinte situação: em um circuito eletrônico genérico um botão ao ser pressionado envia sinal lógico alto para entrada de uma porta lógica inversora, que coloca em sua saída nível lógico 0, desligando um LED. No momento em que este botão não está acionado, qual sinal será o nível enviado para a entrada da porta lógica? Nível 1? Nível 0? Pois é, na verdade nenhum.

Você deve estar se questionado se apenas não enviar sinal nenhum não valeria como 0 volt, afinal se não temos nível 1 (5V por exemplo), temos então nível 0.

NÃO ENVIAR NADA NÃO SIGNIFICA ENVIAR NÍVEL 0.

O grande problema de não enviar nível nenhum, nem 0, é que pode acontecer de pequenas interferências ou mesmo acumulo de cargas no circuito fazerem com que a porta lógica reconheça em sua entrada tanto nível 1 como nível 0, deixando a entrada da porta lógica totalmente vulnerável a instabilidades e comprometendo o funcionamento do circuito. Veja abaixo um caso de instabilidade.

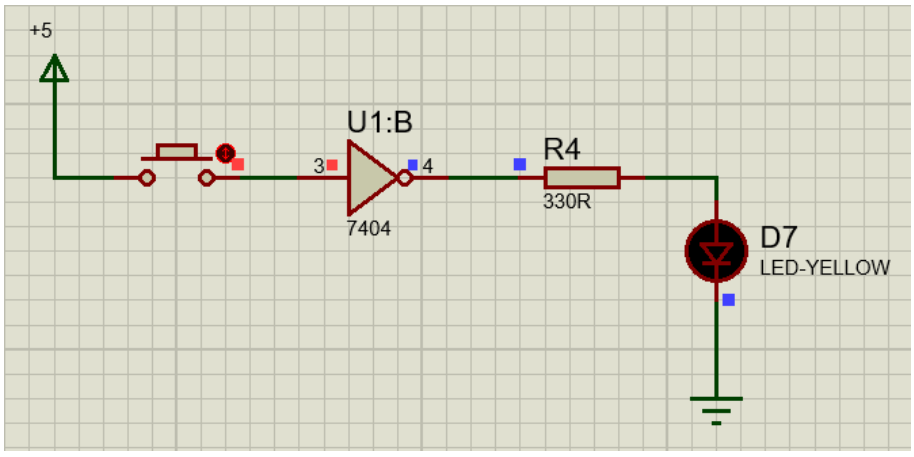


Figura 43- Veja, o botão não está acionado e a porta lógica está interpretando nível 1 em sua entrada.

*A utilização do resistor como **pull down** (puxar para baixo) tem a função de garantir nível baixo em um determinado ponto do circuito eletrônico ou porta lógica quando nenhum sinal é enviado. Sendo assim, o resistor de pull down irá interligar o ponto tido como nível 0 no circuito eletrônico ao ponto que se quer garantir esse estado, "trazendo para baixo" o potencial elétrico desse ponto.*

Como solução para esse problema, surge então o resistor de pull down.

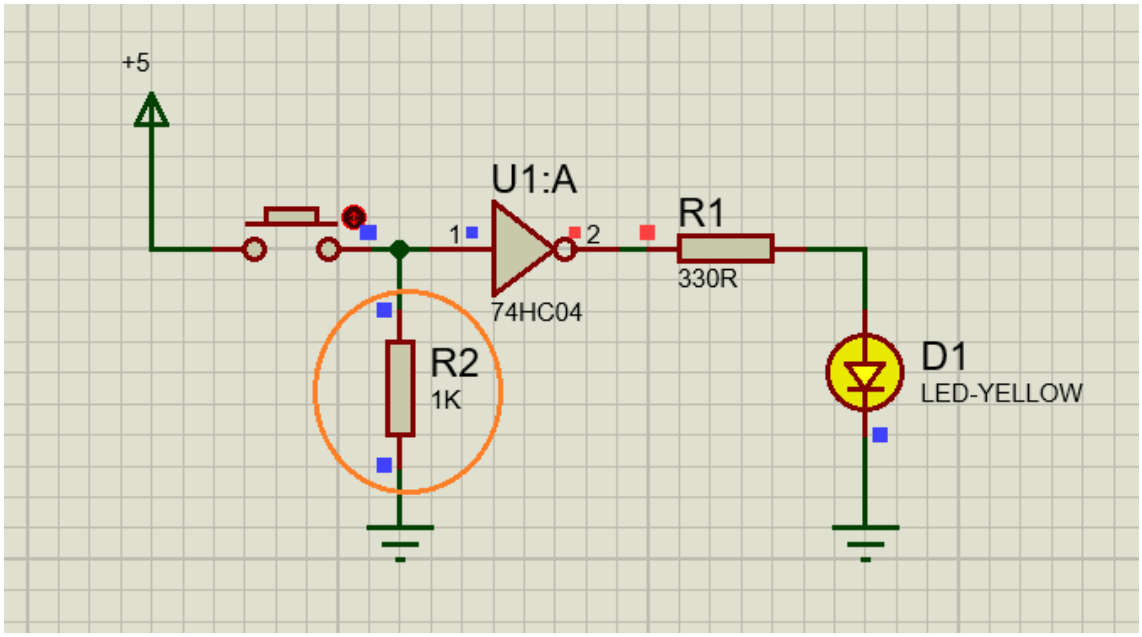


Figura 44 - No momento em que a chave não está acionada, R2 puxa para baixo (pull down) o potencial da entrada da porta lógica, garantindo nível 0 para o circuito digital. Nível 0 na entrada de uma porta inversora (NOT) garante nível em sua saída, fazendo com que o led acenda.

RESISTORES DE PULL UP

Seguindo o mesmo raciocínio para os resistores de PULL DOWN visto anteriormente, os resistores de PULL UP (puxar para cima) vão garantir nível lógico alto em um determinado ponto do circuito ou entrada de uma porta lógica.

Veja o exemplo abaixo:

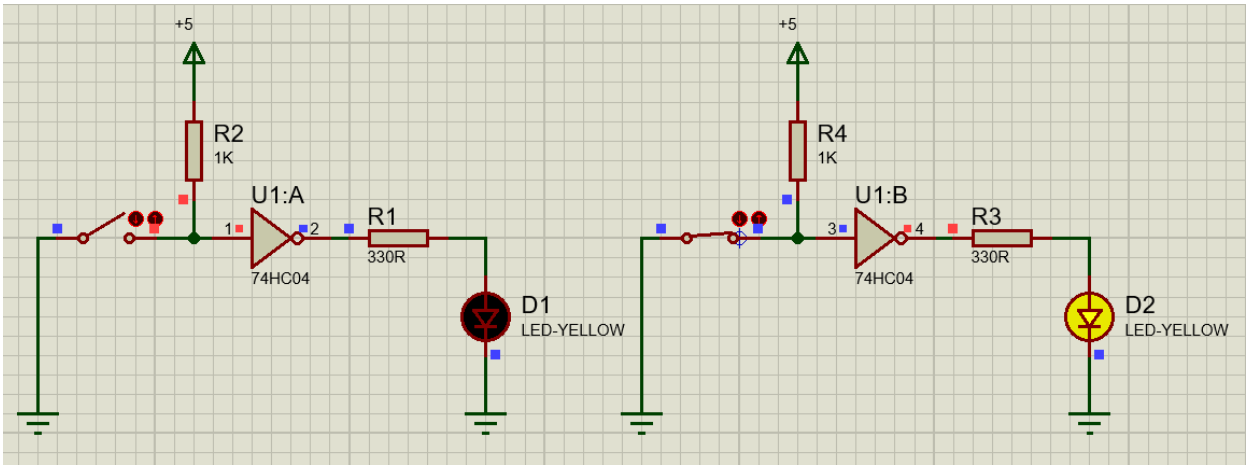


Figura 45 - No momento em que a chave não está acionada, R2 garante nível alto na entrada da porta lógica.

COMO CALCULAR RESISTORES PULL DOWN

Para calcular o resistor de pull down, muito utilizado nas entradas das portas lógicas, é bastante simples, basta aplicar a lei de ohm utilizando alguns dados encontrados no datasheet do componente que possui a porta lógica.

Utilizando como exemplo o circuito integrado 7404, que possui 6 portas lógicas inversoras, veja abaixo alguns parâmetros encontrados no datasheet:

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER	TEST CONDITIONS†			SN54LS04			SN74LS04			UNIT
				MIN	TYP‡	MAX	MIN	TYP‡	MAX	
V_{IK}	V_{CC} = MIN,	I_I = −18 mA				−1.5			−1.5	V
V_{OH}	V_{CC} = MIN,	V_{IL} = MAX,	I_{OH} = −0.4 mA	2.5	3.4		2.7	3.4		V
V_{OL}	V_{CC} = MIN,	V_{IH} = 2 V	I_{OL} = 4 mA		0.25	0.4			0.4	V
			I_{OL} = 8 mA					0.25	0.5	
I_I	V_{CC} = MAX,	V_I = 7 V				0.1			0.1	mA
I_{IH}	V_{CC} = MAX,	V_I = 2.7 V				20			20	µA
I_{IL}	V_{CC} = MAX,	V_I = 0.4 V				−0.4			−0.4	mA
I_{OS}§	V_{CC} = MAX			−20		−100	−20		−100	mA
I_{CCH}	V_{CC} = MAX,	V_I = 0 V			1.2	2.4		1.2	2.4	mA
I_{CCL}	V_{CC} = MAX,	V_I = 4.5 V			3.6	6.6		3.6	6.6	mA

† For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.
‡ All typical values are at V_{CC} = 5 V, T_A = 25°C.
§ Not more than one output should be shorted at a time, and the duration of the short-circuit should not exceed one second.

Observando o parâmetro IIL (i = corrente, i = input, L = Nível baixo) temos a informação de que a condição de teste para nível baixo com uma tensão de input Vi = 0,4V apresentou uma corrente máxima de -0,4mA (*esse sinal negativo apenas indica que a corrente está saindo da porta lógica e não entrando*).

Dessa forma, com esses dados e aplicando a lei de ohm conseguiremos saber qual a resistência elétrica que poderá garantir essa mesma condição na entrada da porta lógica, garantindo nível lógico baixo. Veja:

$$R = \frac{V}{I} \qquad R = \frac{0{,}4}{0{,}0004} \qquad R = 1000\Omega$$

PULL DOWN = 1KΩ

Veja, 1000Ω foi o valor encontrado de resistência, e esse será o valor utilizado para o resistor de pull down. Agora você poderá voltar e ver novamente as imagens dos circuitos anteriores e observar que os valores de resistores de pull down utilizados eram de 1000Ω, ou 1KΩ.

Você sabia que um simples LED pode queimar a saída de uma porta lógica?

Consultando o datasheet podemos verificar as correntes de saída em nível alto e nível baixo das portas lógicas do circuito integrado 7404. Veja:

recommended operating conditions (see Note 3)

		SN5404			SN7404			UNIT
		MIN	NOM	MAX	MIN	NOM	MAX	
V_{CC}	Supply voltage	4.5	5	5.5	4.75	5	5.25	V
V_{IH}	High-level input voltage	2			2			V
V_{IL}	Low-level input voltage			0.8			0.8	V
I_{OH}	High-level output current			−0.4			−0.4	mA
I_{OL}	Low-level output current			16			16	mA
T_A	Operating free-air temperature	−55		125	0		70	°C

NOTE 3: All unused inputs of the device must be held at V_{CC} or GND to ensure proper device operation. Refer to the TI application report, *Implications of Slow or Floating CMOS Inputs*, literature number SCBA004.

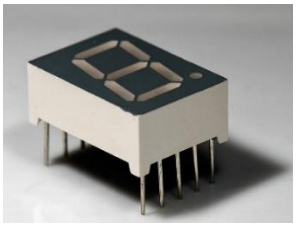
Figura 46- IoH - Corrente de saída em nível alto | IoL=Corrente de saída em nível baixo

Tomando como exemplo um LED que precisa de 20mA de corrente para funcionar com sucesso, estaria correto utilizar a saída da porta lógica inversora em nível alto para fazer o acionamento do led mencionado?

Consultando o parâmetro IOH (corrente de saída em nível alto), podemos verificar que o limite de fornecimento de corrente é de -0,4mA, ou seja, -400uA. Veja que o limite de fornecimento da porá lógica é muito inferior ao necessário para que o led trabalhe com sucesso, isso poderia gerar um estresse muito grande para o circuito integrado, com grandes chances de danificar essa porta.

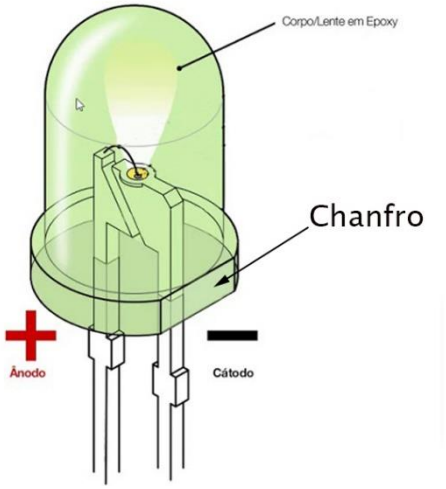
A melhor forma de acionar o LED seria utilizando a saída da porta lógica com nível baixo, onde IOL é de 16mA.

CATODO COMUM E ANODO COMUM



Um dispositivo muito comum utilizado no incrível mundo da eletrônica é o famoso display de 7 seguimentos. Ele é um conjunto de 7 leds dispostos de forma a gerar dígitos ou caracteres dependendo da sua configuração de acionamento.

Um led possui basicamente dois terminais, anodo e catodo. O anodo é alimentado com potencial positivo e o catodo alimentado com potencial negativo.



Sabendo que o display é disposto por 7 seguimentos e cada seguimento corresponde a um led, existem duas configurações básicas para que eles sejam acionados: catodo comum ou anodo comum.

Catodo comum é a configuração onde os catodos de todos os led estão interligados no mesmo ponto (potencial negativo), e o acionamento de cada led será feito individualmente pelo envio de potencial positivo em seus anodos.

Anodo comum é a configuração onde todos os anodos dos leds se encontram interligados no mesmo potencial positivo, onde o acionamento de cada led será feito pelo envio de sinais negativos separadamente em cada segmento (led).

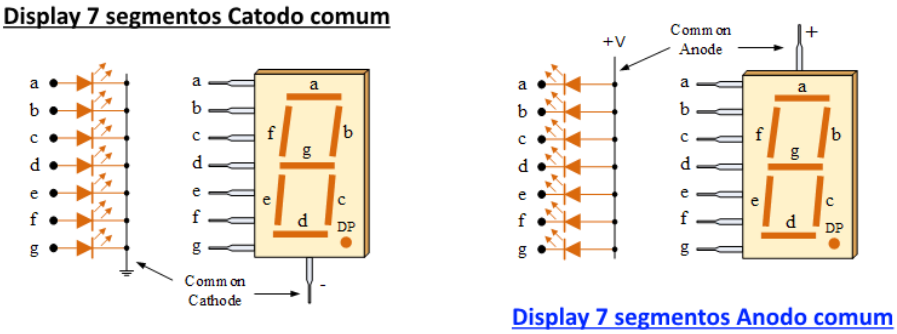


Figura 47- https://www.filipeflop.com/blog/como-construir-um-relogio-com-arduino/

SAÍDA SINK E SOURCE

As portas lógicas podem funcionar fornecendo ou absorvendo corrente em suas saídas. Dessa forma, há duas maneiras de utilizar as saídas das portas lógicas para controlar outros dispositivos: **Saída SINK, e saída SOURCE.**

Utilizar a saída de uma porta lógica como **SINK**, significa dizer que ela fará o acionamento de dispositivos com nível 0, ou seja, absorvendo corrente em sua saída.

Utilizar a saída de uma porta lógica como **SOURCE,** significa dizer que ela fará o acionamento de dispositivos com nível 1, fornecendo corrente em sua saída.

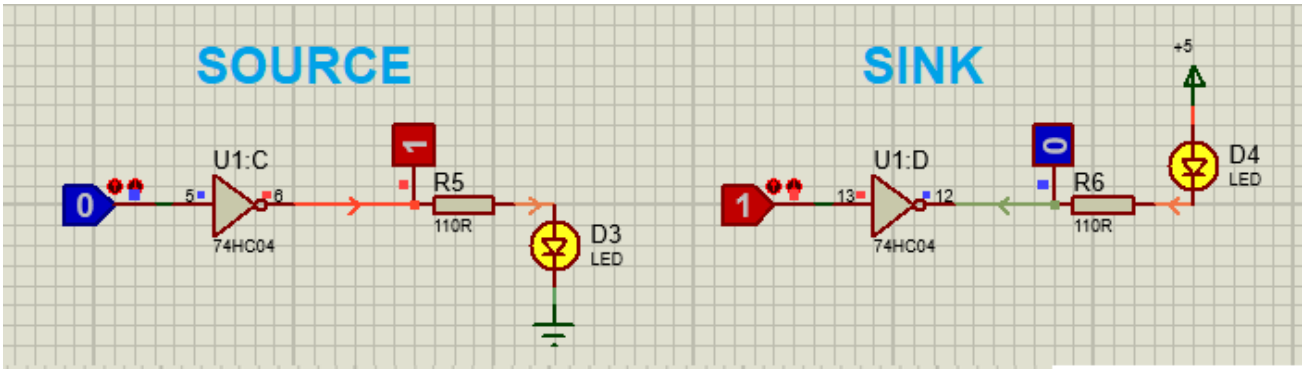


Figura 48 - Observe na saída da porta inversora a seta indicando o sentido de corrente nas duas configurações. SOURCE fornece corrente, SINK absorve corrente.

MANDAMENTOS DA MONTAGEM

EM PROTOBOARD

M1

- SEPARARÁS OS MATERIAIS E FERRAMENTAS NECESSÁRIAS PARA SUA MONTAGEM

M2

- POSICIONARÁS CORRETAMENTE O CI NO PROTOBOARD

M3

- ALIMENTARÁS O CI OBRIGATÓRIAMENTE

M4

- MEDIRÁS CONTINUIDADE ELÉTRICA DAS SUAS MONTAGENS ANTES DE ENERGIZAR O SEU CIRCUITO

M5

- TESTARÁS O SEU CIRCUITO TOTALMENTE E VERÁS QUE A PRÁTICA É UMA CAIXA DE SURPRESAS

M6

- NÃO FARÁS APENAS MONTAGENS NUTELAS! O SIMPLES E PRÁTICO FUNCIONA MUITO BEM

M1 – *Antes de iniciar qualquer montagem de circuitos ou experimentos no protoboard, confirme a lista de materiais necessários para que a atividade seja realizada com sucesso. É muito importante que o procedimento de montagem só se inicie quando todos os materiais estiverem disponíveis e forem adquiridos.*

M2 – *É de suma importância que o aluno tenha total conhecimento dos principais barramentos do protoboard bem como eles são distribuídos na matriz de contato. Diferentemente de componentes com apenas dois ou três terminais, o circuito integrado 7404 possui 6 terminais de cada lado do seu encapsulamento, o que exige muita atenção ao ser utilizado no protoboard para que não aconteça de seus terminais ficarem curto-circuitados pelos barramentos da protoboard*.

M3 – *A alimentação (VCC e GND) do circuito integrado é importantíssima e merece bastante atenção. Qualquer problema nessa etapa acaba comprometendo integralmente todo o projeto.*

M4 – *Após todo o procedimento de montagem no protoboard, fixação de componentes e fios, é imprescindível que todas as continuidades elétricas sejam testadas antes mesmo de iniciar a energização do circuito. Isso irá garantir que pequenos problemas não sejam causadores de frustração durante as atividades práticas. É muito comum fios mau encaixados, terminais sem continuidade com barramentos, e esses testes já irão eliminar esses problemas antes mesmo de testar o circuito.*

M5 – *Ao energizar o sistema, mesmo depois de ter feitos todos os testes de continuidade, pode acontecer de o sistema não funcionar. Diversas podem ser as causas para que isso aconteça, e pra isso o aluno deverá ser paciente para chegar na solução do problema, mas é importante ressaltar que nem sempre um componente novo é garantia de bom funcionamento. Surpresas podem acontecer e isso também é comum.*

M6 – *É importante, além de pensar nas questões elétricas como condutividade, conexões, posição correta dos componentes, encapsulamento, numeração de terminais, pensar também nas questões estéticas do projeto. Esse cuidado não se trata apenas de uma questão de "beleza", mas de organização até mesmo para que possíveis defeitos sejam corrigidos com maior facilidade, ou até mesmo para produzir pequenas alterações no projeto, que com organização e melhor legibilidade ficará mais fácil verificar as malhas e toda a estrutura do circuito montado.*

APRENDER NOVOS IDIOMAS, UMA PORTA PARA O CONHECIMENTO

Aprender uma nova língua é um desafio e tanto, exige dedicação e muita disciplina. A linguagem, de forma geral, é uma grande ferramenta de comunicação e é a base para a transmissão e absorção de conhecimento, das relações e principalmente o alicerce das evoluções tecnológicas.

Imaginem se os pesquisadores de hoje só tivessem acesso aos artigos disponíveis em sua língua, será que estaríamos no mesmo patamar de evolução tecnológica? Certamente que não! As trocas de informações entre todos aqueles que pretendem estudar é sem dúvida a grande porta para o conhecimento e desenvolvimento, porém, para que de fato essas portas se abram não podemos nos limitar apenas à nossa língua materna, devemos expandir e aprender novas línguas, como por exemplo a língua inglesa.

Durante o treinamento você tem visto que ao consultar os datasheets, todas as informações do documento vêm disponível principalmente em língua inglesa. Isso não se limita apenas ao datasheet, e ao mundo da eletrônica, mas em todas as áreas do conhecimento, afinal não é só no Brasil que existem pessoas estudando, pesquisando e produzindo.

Por isso, se você é um aluno que quer expandir grandemente seus conhecimentos, entender documentos importantes e repletos de informações preciosas, comece aceitar a ideia de utilizar materiais em outras línguas, ainda que traduzindo palavra por palavra com a ajuda do google, mas isso será fundamental para sua evolução.

SISTEMAS DE NUMERAÇÃO

Chegou a hora de estudarmos os sistemas de numeração. Basicamente um sistema de numeração é a forma como os algarismos e/ou caracteres se organizam para representar valores. Estudaremos neste treinamento basicamente três sistemas: **decimal, hexadecimal e binário.**

> *Os sistemas de numeração compreendem um conjunto de regras que, em uma determinada BASE, possam representar quantidades e informações*.

Mas, o que é essa tal de base?

Quando dizemos que um sistema de numeração está na base 10 por exemplo, estamos nos referindo aos números de algarismos utilizados no sistema para que, combinados, possam gerar informações. Sendo assim, um sistema na base 10 (decimal) possui dez possibilidades de algarismos, assim como um sistema na base 2 (binário) possui 2 possibilidades de algarismos, e para fechar, na base 16 (hexadecimal) possui dezesseis possibilidades de algarismos.

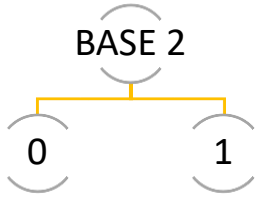


Figura 49 – SISTEMA BINÁRIO

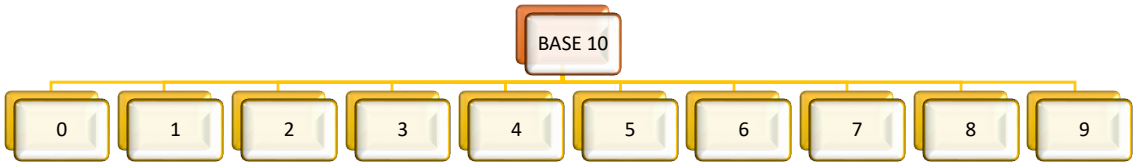


Figura 50 - SISTEMA DECIMAL

Figura 51 - Sistema Hexadecimal

ENTENDENDO A BASE DECIMAL

Os números em base decimal, ou simplesmente sistemas decimais, são chamados assim por possuírem apenas 10 algarismos para serem combinados. Essas combinações podem ser divididas em unidade, dezena, centena, milhar, e assim sucessivamente.

Vamos estudar cada divisão entre unidade, dezena, centena, tomando como referência a base 10, e isso significa dizer que cada divisão possuirá 10 possibilidades, assim que uma atingir seu limite, passamos para outra. Assim que a unidade se completar com 10 algarismos, passamos para dezena.

A numeração decimal possui 10 algarismos possíveis de serem combinados, 0,1,2,3,4,5,6,7,8,9. Quando a utilização desses algarismos chega no ultimo possível que é o número 9, é preciso então combinar dois dígitos para continuar a sequência. [...]7,8,9,**10.** O número 10 é formado por 2 dígitos, 1 e 0. Dizemos então que ele possui 1 dezena e 0 unidades. Veja a tabela abaixo:

milhar 10^3	centena 10^2	dezena 10^1	unidade 10^0
0	0	0	0
0	0	0	1
0	0	0	2
0	0	0	3
0	0	0	4
0	0	0	5
0	0	0	6
0	0	0	7
0	0	0	8
0	0	0	9
0	0	1	0

Vamos entender a tabela:

Base 10 em seu primeiro dígito	$10^0 = 1$	**UNIDADE**
Base 10 em seu segundo dígito	$10^1 = 10$	**DEZENA**
Base 10 em seu terceiro dígito	$10^2 = 100$	**CENTENA**
Base 10 em seu quarto dígito	$10^3 = 1000$	**MILHAR**

Vamos analisar o número 125;

milhar 10^3	centena 10^2	dezena 10^1	unidade 10^0
0	1 $1 \times 10^2 = 100$ CENTO	2 $2 \times 10^1 = 20$ VINTE	5 $5 \times 10^0 = 5$ CINCO

O número 125 possui 1 centena (cento), 2 dezenas (vinte) e 5 unidades (cinco)

PORTANTO, 125 é Cento e vinte e cinco.

Dessa forma fica claro de onde surgem as divisões. É muito importante entender essa etapa para prosseguirmos com nossos estudos, será fundamental para o avanço e entendimento dos novos conteúdos.

CONTAGEM BINÁRIA

Em contagem binaria há apenas dois algarismos disponíveis (1 e 0) para que sejam feitas as combinações gerando valores e informações. Por isso dizemos que a contagem binaria é em base 2.

Assim como no sistema decimal, o sistema binário seguirá a mesma regra de quando chegar no ultimo algarismo disponível para contagem, adiciona-se um digito a esquerda

MSB			LSB	
2^3	2^2	2^1	2^0	DECIMAL
0	0	0	0	*0*
0	0	0	1	*1*
0	0	1	0	*2*
0	0	1	1	*3*
0	1	0	0	*4*
0	1	0	1	*5*
0	1	1	0	*6*
0	1	1	1	*7*
1	0	0	0	*8*
1	0	0	1	*9*
1	0	1	0	*10*

Vamos analisar o número binário 0011.

A regra para utilizar a tabela é muito simples, multiplica-se cada digito binário pelo valor da sua coluna correspondente (2^0 , 2^1 ...) depois soma-se todos os valores encontrados e o resultado será o valor em decimal.

MSB			LSB
2^3	2^2	2^1	2^0
0	**0**	**1**	**1**
$0 \times 2^3 = 0$	$0 \times 2^2 = 0$	$1 \times 2^1 = 2$	$1 \times 2^0 = 1$

0 + 0 + 2 + 1 = 3

$11_2 = 3_{10}$

0011 em binário corresponde a 3 em decimal

Vamos agora utilizar uma tabela facilitada para interpretar os números binários. Essa tabela já converte os valores das potências das colunas. Basta somar os valores das colunas onde existirem dígitos validos 1, e onde o digito for 0 não entra na soma. Veja o exemplo abaixo com o número binário 1011:

MSB			LSB
8	4	2	1
1	**0**	**1**	**1**

8 + 0 + 2 + 1

= 11

1011 em binário = 11 em decimal

Veja outro exemplo: **1101_2**

MSB			LSB
8	4	2	1
1	**1**	**0**	**1**

8 + 4 + 0 + 1

= 13

1101 em binário = 13 em decimal

Veja que na tabela há duas siglas que já estudamos anteriormente, **LSB (Least Significant Bit) e MSB (Most Significant Bit).** Ambos indicam no código binário os bits mais significativos e menos significativos.

Veja, quanto maior for o número, mais ele avança em dígitos para a esquerda, por isso o bit mais significativo está à esquerda e o menos significativo à direita.

CONTAGEM HEXADECIMAL

Seguindo a mesma lógica disposta anteriormente para os números decimais e binários, vamos agora estudar os números hexadecimais, aqueles que possuem 16 algarismos disponíveis para combinações (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F).

Veja que agora temos letras nos algarismos disponíveis, e cada letra disponível no sistema corresponderá a um número decimal, A=10, B=11, C=12, D=13, E=14, e F=15.

Veja o exemplo abaixo com o número $31A_{16}$

16^3	16^2	16^1	16^0
0	3	1	A
$0 \times 16^3 = 0$	$3 \times 16^2 = 768$	$1 \times 16^1 = 16$	$A \times 16^0 = 10$ *OBS.: A = 10*
0 + 768 + 16 + 10 = 794 $31A_{16} = 794_{10}$ **31A em hexa corresponde a 794 em decimal**			

CONVERSÃO - HEXADECIMAL PARA BINÁRIO

Para fazer a conversão de hexadecimal para binário é muito simples, para isso vamos utilizar um recurso de tabela comparativa entre binário e hexadecimal apresentada logo abaixo a direita.

Para converter um número hexadecimal em binário, primeiramente vamos analisar quantos dígitos serão necessários em binário para representar o maior digito em hexadecimal. A letra F é a maior possibilidade de valor no sistema hexadecimal, e de acordo com a tabela, corresponde a 4 dígitos em binário. Dessa forma, para fazer a conversão de hexa para binário, vamos utilizar 4 dígitos binários para cada algarismo hexadecimal que será convertido, veja abaixo.

*Converter **3F**$_{16}$ para sistema Binário*

3	F
0011	1111
00111111_2	

*Converter **FACA**$_{16}$ para sistema Binário*

F	A	C	A
1111	1010	1100	1010
1111101011001010_2			

8	4	2	1	HEXADECIMAL
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	A
1	0	1	1	B
1	1	0	0	C
1	1	0	1	D
1	1	1	0	E
1	1	1	1	F

CONVERSÃO – BINÁRIO PARA HEXADECIMAL

Para converter binário em hexadecimal o procedimento será o inverso do estudado anteriormente. Vimos que cada algarismo em hexadecimal representam 4 dígitos em binário, dessa forma, a cada 4 dígitos em binários utilizaremos um único algarismo em hexadecimal para representa-lo. Para saber esse número utilizaremos a tabela comparativa. Veja:

Separe os grupos de 4 dígitos do valor em binário, começando sempre da direita para a esquerda.

*Converter **11001011**$_2$ para sistema Hexadecimal*

1100	1011
C	B
CB$_{16}$	

*Converter **0010001101111011**$_2$ para hexadecimal*

0010	0011	0111	1011
2	3	7	B
237B$_{16}$			

8	4	2	1	HEXADECIMAL
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	A
1	0	1	1	B
1	1	0	0	C
1	1	0	1	D
1	1	1	0	E
1	1	1	1	F

Se na hora de dividir o número binário em grupos de 4 dígitos, acabar sobrando algarismos, inclua zeros até completar 4 dígitos. Veja um exemplo:

*Converter **10101111110101**$_2$ para sistema hexadecimal*

0010	1011	1111	0101
2	B	F	5
2BF5$_{16}$			

METODO DA DIVISÃO "INFINITA" – DECIMAL PARA BINÁRIO

Vamos estudar uma forma de conversão entre sistemas de numeração por meio da divisão infinita. Não que a divisão seja infinita de fato, mas é uma forma de indicar *carinhosamente* a divisão de um número por outro até o ponto que não seja mais possível continuar a operação.

Essa forma pode ser aplicada para fazer a conversão de sistema decimal para qualquer sistema de numeração. Veja:

Converter número 25 em decimal para binário

Como a conversão será para o sistema binário, base 2, então divide-se por 2 o valor a ser convertido, até que não seja mais possível efetuar a operação.

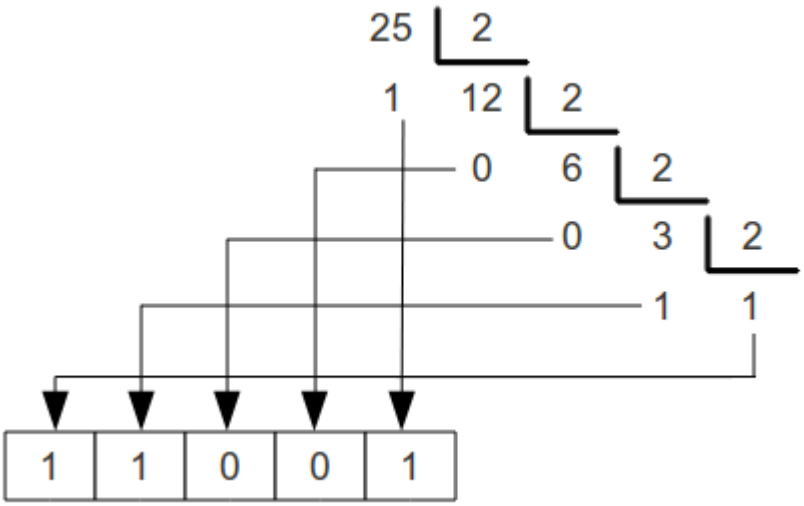


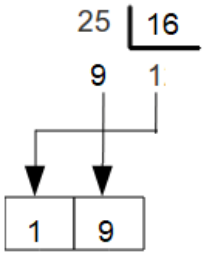
Figura 52 - Decimal para binário

O resultado dessa conversão será obtido a partir dos valores restantes das divisões, **começando pelo 1 da última divisão**. Veja a imagem acima.

Portanto, $25_2 = 11001_2$

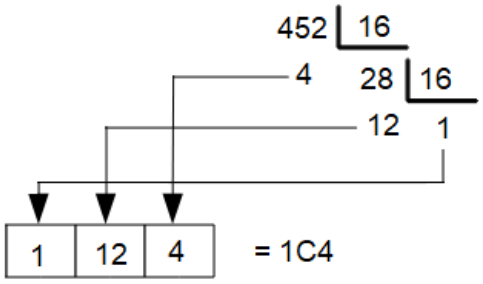
Converter número 25 em decimal para hexadecimal

A maneira utilizada para fazer esta conversão será a mesma apresentada anteriormente, método da divisão “infinita”. Como estaremos fazendo a conversão de decimal para hexadecimal, faremos a divisão por 16.



Portanto, $25_{10} = 19_{16}$

Veja mais um exemplo:



Veja, como na conversão apareceu o número 12, esse valor deverá ser convertido para a base 16, que é representado pela letra C.

Converter QUALQUER BASE para decimal

Para converter qualquer base para decimal, basta aplicar o método da tabela mágica. Essa tabela apresenta um valor em decimal para cada coluna. Multiplica-se cada digito do número a ser convertido pelo valor da coluna, e soma-se todos os resultados. Veja abaixo como é simples:

2^3	2^2	2^1	2^0
8	4	2	1
1	0	1	1
1 x 8 =8	0 x 4 =0	1 x 2 =2	1 x 1 =1

8 + 0 + 2 + 1 = 11

1011 em binário corresponde a 11 em decimal

16^3	16^2	16^1	16^0
4096	256	16	1
1	0	A	1
1 x 4096 =4096	0 x 256 =0	16 x 10 =10	1 x 1 =1

4096 + 0 + 160 + 1 = 4257

10A1 em hexadecimal corresponde a 4257 em decimal

Veja que as duas conversões acima foram feitas utilizando a tabela já estudada nesse treinamento, onde os dígitos do valor a ser convertido multiplica o valor correspondente de sua coluna, depois todos os resultados serão somados formando o valor final em decimal. Lembrando que se o valor a ser convertido for em binário, as colunas possuirão base 2, e se o valor a ser convertido for hexadecimal, as colunas possuirão base 16.

A FAMOSA TABELA ASCII

A tabela ASCII, ou Código Padrão Norte-americano para Intercambio de informação ("*American Standard Code for Information Interchange*"), tem por objetivo padronizar códigos para representar os caracteres alfanuméricos como sinais, letras, números e acentos, utilizados em diversos computadores. Dessa forma, com um código padrão, computadores produzidos por diferentes empresas poderiam entender os mesmos códigos.

ASCII control characters

00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

Figura 53 - tabela ASCII

Agora você pode estar se questionando se o motivo de termos estudado aquelas conversões todas é pelo simples fato de que as máquinas trabalham suas interpretações a partir dessas conversões. Correto? Exatamente!!! Para entender como o mundo digital funciona, é preciso entender como ele se comunica, e por isso estamos aqui apresentando detalhes de como tudo funciona.

*Vamos pra um desafio. Você sabe o que **é CODIFICAR e DECODIFICAR**?*

Codificar é converter um conjunto de informações de alto nível em um código de baixo nível.

***Linguagem de alto nível** é uma linguagem feita para a interpretação dos seres humanos.*

***Linguagem de baixo nível** é uma linguagem de máquina, composta por códigos eletrônicos que serão interpretados pelo sistema digital.*

Decodificar é converter um conjunto de informações de baixo nível em um código de alto nível, ou seja, converter a linguagem de máquina em linguagem pronta para ser interpretada pelos humanos.

VAMOS AO DESAFIO – CODIFICANDO UMA FRASE

O desafio será, a partir do código ASCII, identificar os valores de cada algarismo representado na frase abaixo, e converter esses valores para os três sistemas estudados, decimal, binário e hexadecimal.

Vamos converter uma linguagem de nível alto (frase entendida por humanos) em uma linguagem de nível baixo (linguagem entendida pelas máquinas). A frase que será convertida será a seguinte:

F.E.F

Amo VCS!

Agora, consultando cada caractere dessa frase na tabela ASCII, obtemos um valor em decimal. Vamos converter esse valor decimal para binário e hexadecimal. Veja:

ASCII	F	.	E	.	F
dec	70	46	69	46	70
bin	01000110	00101110	01000101	00101110	01000110
hexa	0x46	0x2E	0x45	0x2E	0x46

ASCII	enter	A	m	o	espaço	V	C	S	!	NULL
dec	13	65	109	111	32	86	67	83	33	0
bin	01000001	00101110	01101101	01101111	00100000	01010110	01000011	01010011	00100001	00000000
hexa	0x0D	0x41	0x6D	0x6F	0x20	0x56	0x43	0x53	0x21	0x00

Perceba que, pontos, maiúscula e minúscula, espaço, pular linha, e até mesmo o fim da frase, representam um valor a ser interpretado e codificado pelo sistema.

O fim da frase sempre virá acompanhado de uma identificação NULL, que significa basicamente NULO, ou seja, a partir desse caractere a string se encerra.

LÓGICA E (AND)

Sabemos que um sistema combinacional é aquele em que a saída depende única e exclusivamente dos valores de suas entradas. Na lógica E, o valor da saída só será verdadeiro (1), quando todos os valores das entradas forem verdadeiros (1).

Vamos imaginar a seguinte situação – **RECEITA DE ABACATE EXTRAORDINÁRIO**

Para que uma vitamina de abacate ganhe a categoria de extraordinária ela **precisa** possuir abacate, mel e limão. Essa combinação caracteriza a receita de abacate extraordinário.

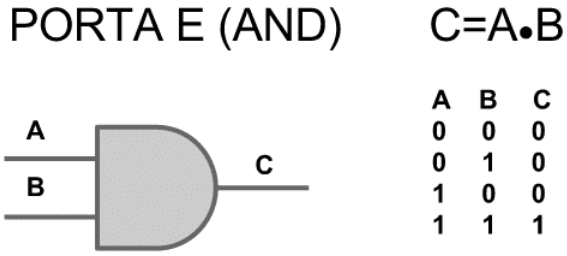
Sendo assim, a receita de abacate extraordinário precisa ter mel **E** limão.

Veja a representação da lógica E na receita de abacate extraordinário.

INGREDIENTE A MEL	INGREDIENTE B LIMÃO	ABACATE EXTRAORDINÁRIO
NÃO	*NÃO*	*NÃO*
SIM	*NÃO*	*NÃO*
NÃO	*SIM*	*NÃO*
SIM	***SIM***	***SIM***

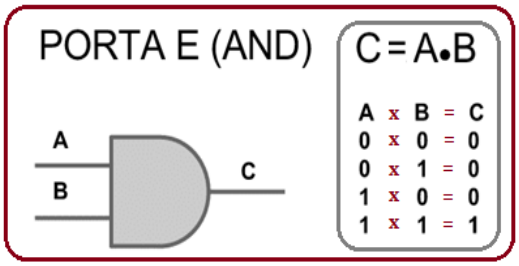
Veja, para que o abacate seja extraordinário, é preciso que a receita tenha necessariamente mel **E** limão, caso contrário ela não será verdadeira. Esse é um exemplo onde o valor final de um produto (sabor) depende exclusivamente dos ingredientes de sua produção, podendo ser comparado com um sistema combinacional. Já os ingredientes e a composição da receita representam muito bem uma lógica E, onde dois ingredientes, um **E** outro, precisam estar presentes para que a receita seja concluída com sucesso.

Veja como uma porta lógica **E** é representada:



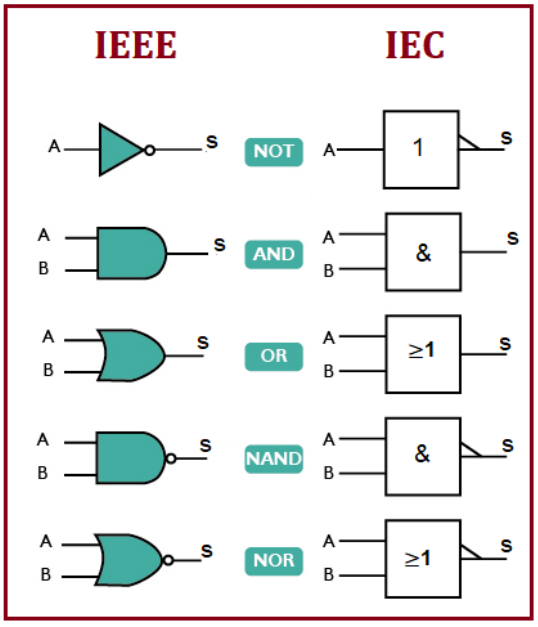
Observando a imagem acima, temos a representação gráfica da porta lógica E, a tabela da verdade mostrando todas as suas combinações, e também a expressão booleana que representa essas combinações.

C = A . B *(Leia: C será verdade (1) se A **E** B forem verdadeiros)*



Essa expressão tem o ponto (Multiplicação) como representação da lógica E, dessa forma, podemos efetuar a multiplicação dos valores digitais das entradas para que se obtenha o valor da saída.

SIMBOLOGIAS NA NORMALIZAÇÃO IEEE E IEC



As portas lógicas podem aparecer graficamente de duas formas, na normalização **americana IEEE** (Institute of Electrical and Electronic Engineers), ou na normalização **europeia IEC** (International Electrotechnical Commission). Veja a diferença gráfica entre as duas normalizações.

SIMULANDO A PORTA LÓGICA E

Vamos agora utilizar um software de simulação de circuitos eletrônicos para comprovar toda a lógica E estudada até aqui. Por meio da simulação podemos botar a mão na massa e materializar o conhecimento adquirido. Veja a comprovação da tabela da verdade na simulação da porta lógica E.

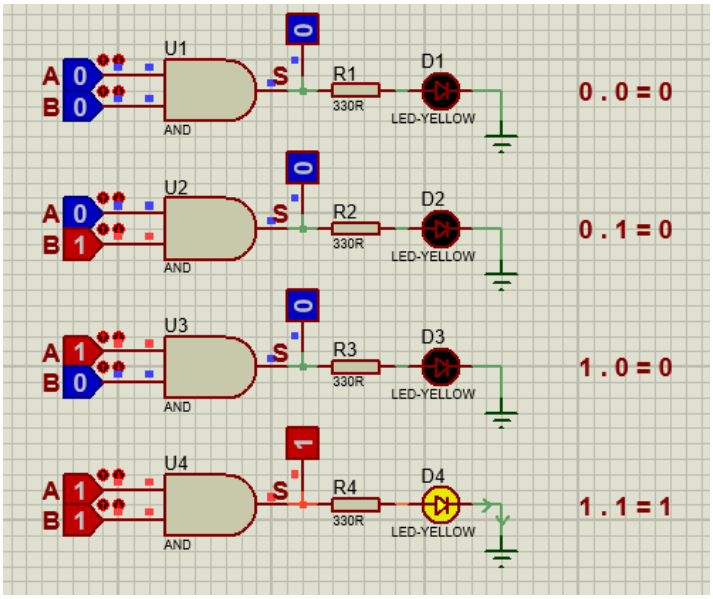


Figura 54 - Porta lógica AND

Podemos observar na simulação que a saída da porta lógica **E** só será verdadeira (1) quando todas as entradas forem verdadeiras.

Uma outra maneira de entender a porta lógica **E** é fazer uma associação em série de chaves NA (normalmente abertas). Veja:

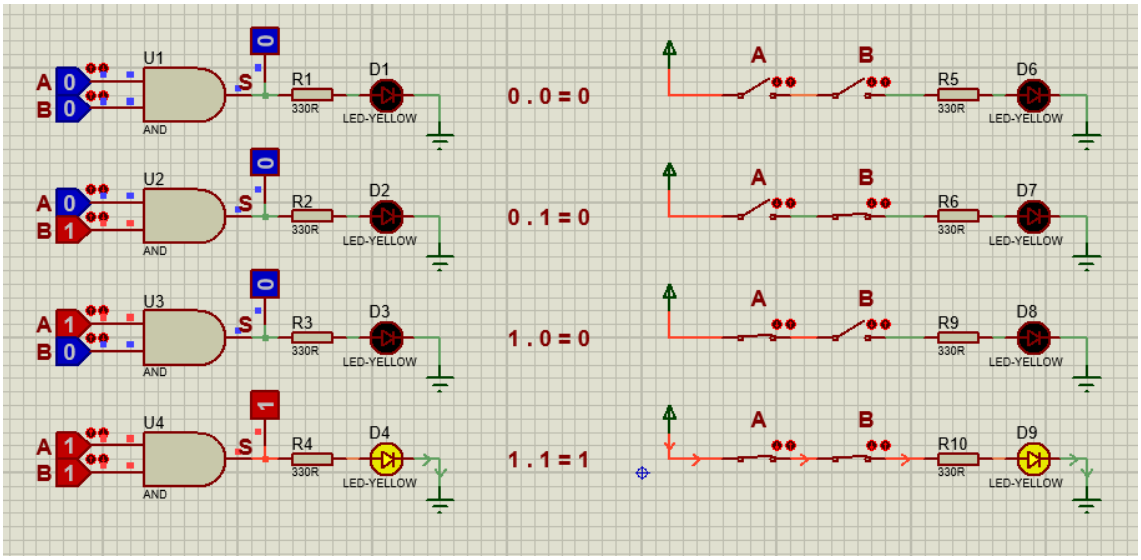


Figura 55- Comparação de portas lógicas E com circuito série de chaves NA. Se uma das chaves estiver aberta, o caminho de circulação de corrente é interrompido e o led não será aceso. Apenas quando as duas chaves estão acionadas é que a saída será verdadeira.

Será que existem portas lógicas com mais entradas, e não somente duas?

A resposta é, SIM!

Veja alguns circuitos integrados de portas lógicas com diferentes números de entradas.

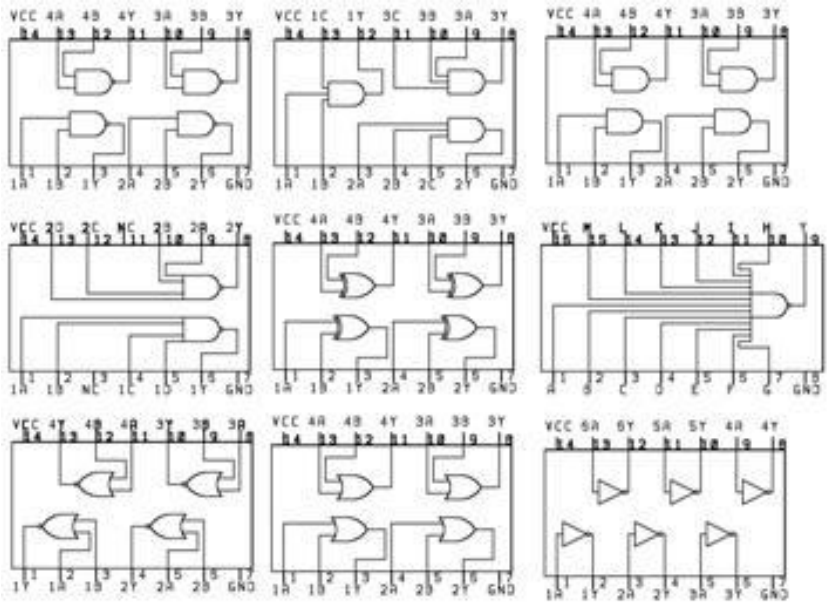


Figura 56 - Portas lógicas com diferentes números de entradas

Agora vamos para a prática simular no protoboard a porta lógica E utilizando o circuito integrado 7408, com dicas incríveis sobre os erros mais comuns que podem acabar destruindo seu circuito integrado. Pegue seus materiais e vamos colocar a mão na massa.

LÓGICA OU (OR)

A lógica **OU** se fundamenta em ter ao menos um valor de entrada verdadeiro para que a saída seja verdadeira.

Vamos utilizar como exemplo um churrasco. Imaginem que a felicidade do sujeito que irá realizar o churrasco é ter carne para poder assar. Sendo assim, o resultado é o sujeito feliz com o churrasco se ao menos tiver disponível um tipo de carne para poder assar, ou um tipo, ou outro, ou quem sabe os dois, melhor ainda.

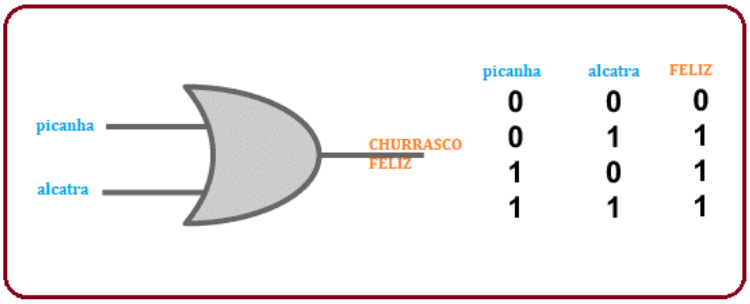
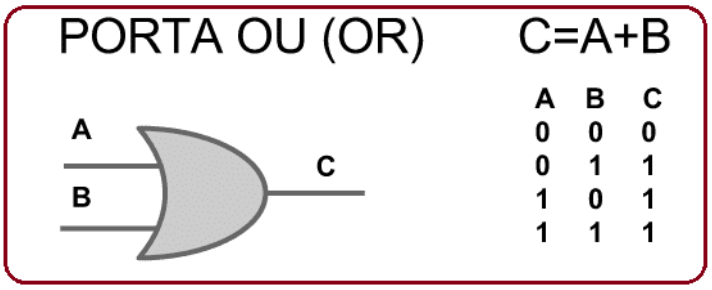


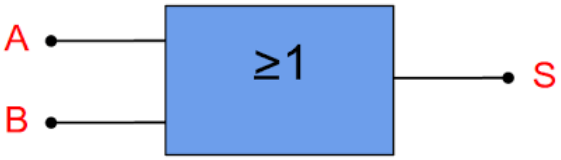
Figura 57 - Se houver picanha OU alcatra, o churrasco feliz acontecerá.

Trazendo a lógica OU para as portas, temos a representação gráfica abaixo:



C = A + B *(Leia: C será verdade (1) se A **OU** B forem verdadeiros)*

Observando a expressão booleana, vimos que a lógica **OU** utiliza o sinal de soma. Uma maneira de entender essa soma é analisar a simbologia da porta lógica na normalização IEC. Veja:



A lógica **OU** na normalização IEC é representada por **≥1.**

Essa representação de ≥1 (maior ou igual a 1) significa dizer que realizando a soma dos valores das entradas, sempre que o resultado for ≥1, a saída será verdadeira. Veja:

A	B	S = A + B	SAÍDA
0	0	0 = 0 + 0	0
0	1	1 = 0 + 1	1
1	0	1 = 1 + 0	1
	1	2 = 1 + 1	1

SIMULANDO A PORTA LÓGICA OU

Ao simular a porta lógica OU, podemos comprovar o funcionamento da lógica estudada anteriormente. Sempre que uma ou mais entradas apresentarem nível lógico 1, a saída será verdadeira (1). Veja a simulação abaixo:

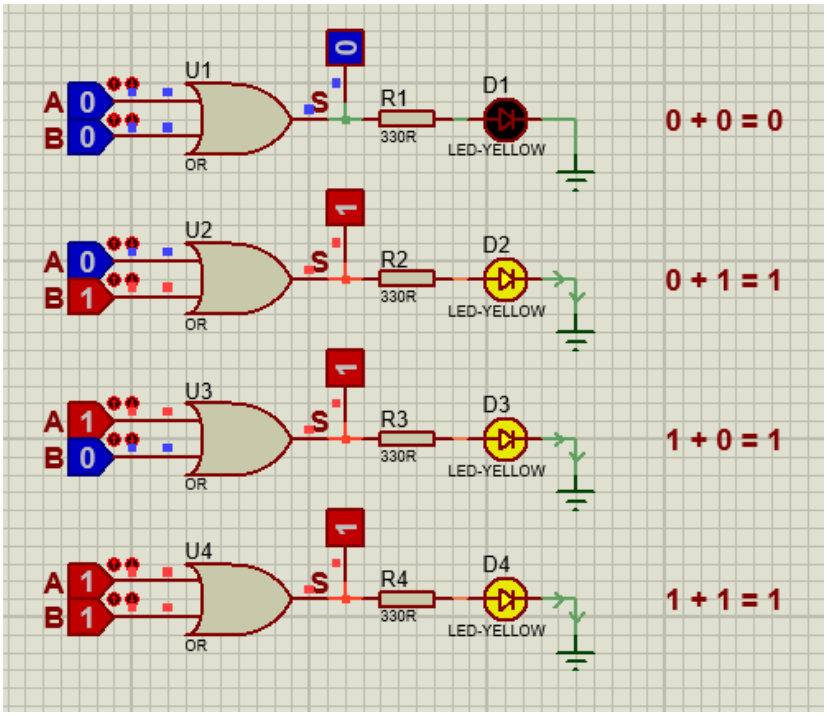


Figura 58 - Portas lógicas OR

Outra forma de entender o funcionamento das portas lógicas **OU**, é fazendo um comparativo com um circuito paralelo de chaves normalmente abertas (NA). Veja:

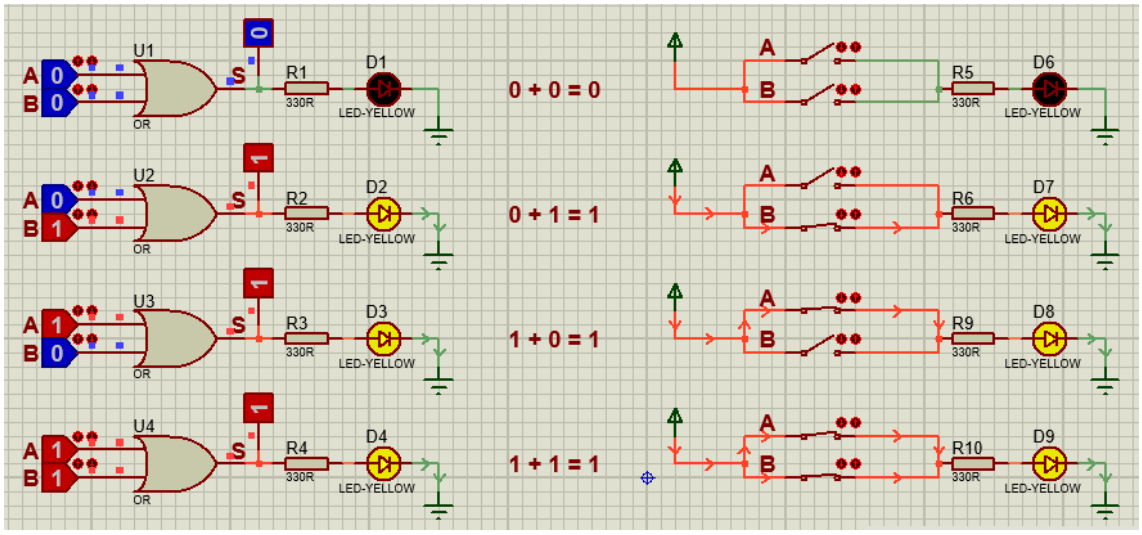


Figura 59 - Portas lógicas OR com demonstração de circuito de chaves

DATASHEET - PARÂMETROS IMPORTANTES

Neste treinamento sempre destacamos a importância de consultar o datasheet, de buscar as características de funcionamento dos componentes para que a aplicação, substituição e entendimento do funcionamento de cada dispositivo seja a melhor e mais segura possível.

Ao estudar as portas lógicas, é importante destacar alguns parâmetros que podem ser cruciais na hora de entender e utilizar uma porta lógica. São eles:

IOH	Corrente de saída em nível alto
IOL	Corrente de saída em nível baixo
VOH	Tensão de saída em nível alto
VOL	Tensão de saída em nível baixo

Vamos destacar nessa etapa do treinamento dois parâmetros, IOH e VOH. Veja o datasheet do circuito integrado de portas lógicas **OR**, *DM74LS32:*

Electrical Characteristics

over recommended operating free air temperature range (unless otherwise noted)

Symbol	Parameter	Conditions	Min	Typ (Note 2)	Max	Units
V_I	Input Clamp Voltage	V_{CC} = Min, I_I = −18 mA			−1.5	V
V_{OH}	HIGH Level Output Voltage	V_{CC} = Min, I_{OH} = Max V_{IH} = Min	2.7	3.4		V
V_{OL}	LOW Level Output Voltage	V_{CC} = Min, I_{OL} = Max V_{IL} = Max		0.35	0.5	V
		I_{OL} = 4 mA, V_{CC} = Min		0.25	0.4	
I_I	Input Current @ Max Input Voltage	V_{CC} = Max, V_I = 7V			0.1	mA
I_{IH}	HIGH Level Input Current	V_{CC} = Max, V_I = 2.7V			20	µA
I_{IL}	LOW Level Input Current	V_{CC} = Max, V_I = 0.4V			−0.36	mA
I_{OS}	Short Circuit Output Current	V_{CC} = Max (Note 3)	−20		−100	mA
I_{CCH}	Supply Current with Outputs HIGH	V_{CC} = Max		3.1	6.2	mA
I_{CCL}	Supply Current with Outputs LOW	V_{CC} = Max		4.9	9.8	mA

Note 2: All typicals are at V_{CC} = 5V, T_A = 25°C.

Note 3: Not more than one output should be shorted at a time, and the duration should not exceed one second.

Figura 60 - datasheet do circuito integrado DM74LS32

Veja que a tensão de saída em nível alto (VOH) está constando como 3,4V, e desde o primeiro momento neste treinamento destacamos a importância de o nível alto estar o mais próximo possível da tensão de alimentação do CI (no caso, 5,5V), portanto, consideramos esse nível lógico alto de 3,5V como péssimo e deve ser evitado ao máximo nos projetos. *Mas, de fato, há algo que possa ocasionar esse péssimo nível alto de saída, impedindo conseguir um valor próximo de VCC em meu projeto?*

As portas lógicas possuem um limite de fornecimento de corrente que é determinado pelos parâmetros IOH e IOL, isso poderá ser o grande vilão na hora de se conseguir nível altos de qualidades nas saídas das portas lógicas. Vamos analisar o datasheet:

Recommended Operating Conditions

Symbol	Parameter	Min	Nom	Max	Units
V_{CC}	Supply Voltage	4.75	5	5.25	V
V_{IH}	HIGH Level Input Voltage	2			V
V_{IL}	LOW Level Input Voltage			0.8	V
I_{OH}	HIGH Level Output Current			–0.4	mA
I_{OL}	LOW Level Output Current			8	mA
T_A	Free Air Operating Temperature	0		70	°C

Figura 61 - Datasheet do circuito integrado DM74LS32

Veja que a IOH é de -0,4mA e que a IOL é de 8mA. Aqui já podemos evidenciar que, a corrente de trabalho dessa porta lógica é muito maior no modo SINK, ou seja, quando temos a saída em nível baixo. Se, por algum motivo, o consumo de corrente em nível alto ultrapassar o limite informado pelo datasheet, ***haverá um proporcional arriamento da tensão de saída***, provocando aquele **péssimo nível lógico alto** de saída destacado anteriormente. Veja na simulação como isso ocorre.

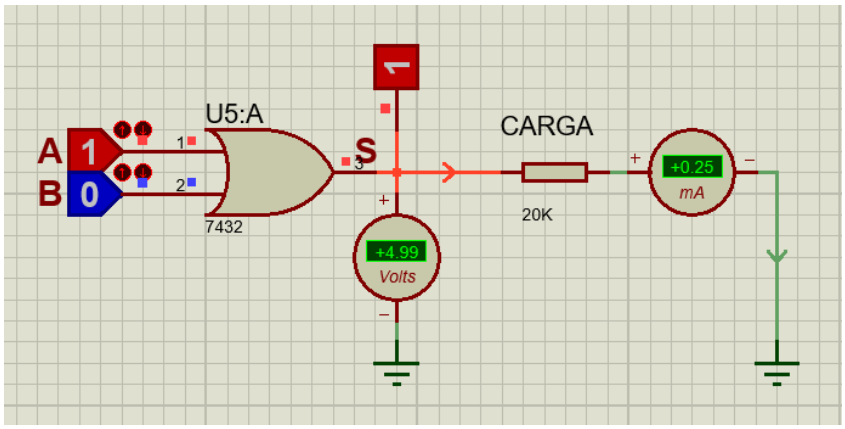


Figura 62 - Corrente de saída no modo Source

Na simulação demonstrada acima, com uma carga de 20K há um consumo de corrente de saída de 0,25mA, **consumo abaixo do limite máximo** de fornecimento oferecido pela porta lógica, que é de **0,4mA**. Por isso a tensão de **saída em nível alto está bem próxima da tensão de VCC (+5V)** do circuito integrado, caracterizando uma **excelente tensão em nível alto.**

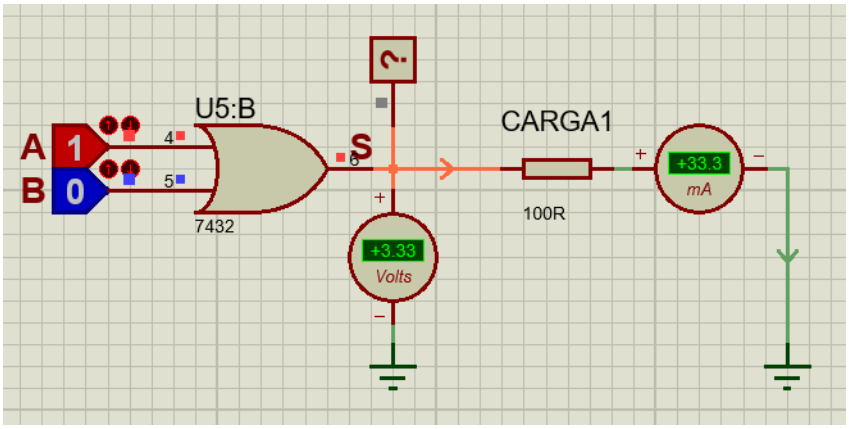


Figura 63 - Corrente de saída acima do limite máximo, provocando queda de tensão no nível lógico de saída.

Já nessa segunda simulação, com uma carga de 100 Ohms, podemos observar um consumo de **corrente de saída de 33,3mA,** muito **acima do limite máximo** de fornecimento informado pelo datasheet **(0,4mA),** situação essa que leva a porta lógica ao extremo trabalho fazendo com que a tensão de saída tenha uma queda, no caso da simulação apresentou 3,3V, caracterizando uma **péssima tensão em nível lógico alto**. Veja que essa tensão de saída é tão ruim logicamente falando que nem é reconhecida pelo identificador de nível lógico colocado na simulação, que permanece com um ponto de interrogação sem conseguir identificar o nível de saída.

Portanto, veja como é importante ter conhecimento dos parâmetros de funcionamento das portas lógicas, uma alteração no dispositivo conectado nas saídas das portas lógicas poderá comprometer totalmente o seu funcionamento. Quanto menor a resistência do dispositivo acionado pela porta lógica, maior o consumo de corrente de saída, consequentemente maior o esforço da porta lógica, podendo comprometer seu funcionamento e até mesmo queimar o circuito integrado.

O mesmo ocorre com a porta lógica trabalhando no modo SINK, ou seja, ativando a carga com nível lógico baixo e absorvendo corrente. Veja esse comparativo mostrando que ultrapassar o limite máximo de corrente IOL pode gerar problemas com a tensão de saída em nível lógico baixo VOL.

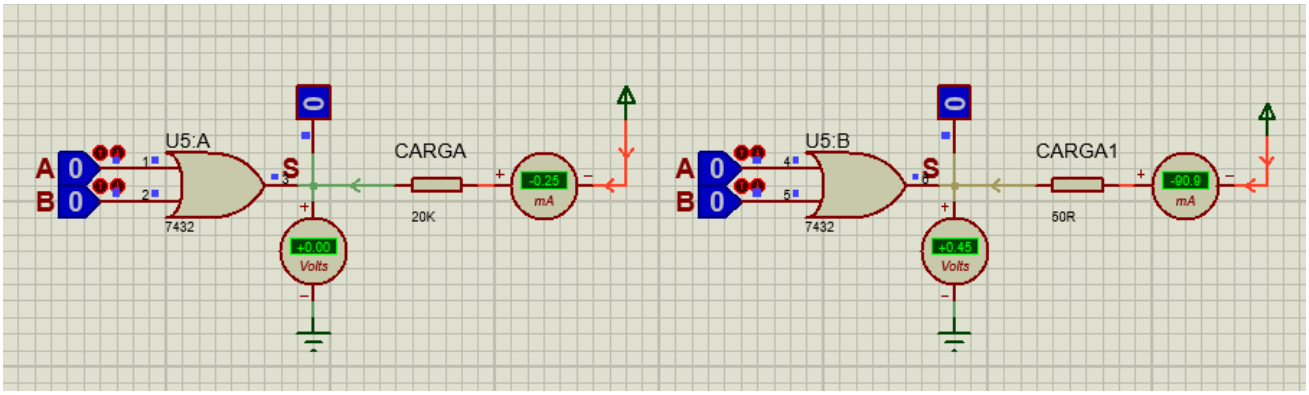


Figura 64 - Comparação dos níveis de corrente de saída dentro e acima dos limites máximos, provocando alteração no nível lógico de saída.

Sabendo que o **limite de corrente em nível baixo informado pelo datasheet é de 8mA,** a simulação acima mostra que quando esse **consumo é menor do que o limite**, a tensão em nível baixo **VOL se aproxima** bastante, chegando a ser a mesma, **do GND**. Essa é a condição ideal para que a porta lógica funcione com sucesso.

No momento em que a corrente IOL ultrapassa o limite máximo de 8mA, a tensão em nível baixo começa perder qualidade e começa subir, se aproximando do limite máximo determinado para o nível 0 que é de 0,4V, oferecendo risco para o funcionamento da porta lógica e para o circuito integrado como um todo.

TEMPO DE PROPAGAÇÃO DE NÍVEL LÓGICO

Uma porta lógica pode colocar dois estados em sua saída, nível alto e nível baixo. No momento em que a porta faz a transição de um estado para o outro há um pequeno tempo de transição para que essa mudança aconteça por completo. Esse tempo de transição pode ser TPLH, ou TPHL.

TPLH	**Tempo de propagação de LOW para HIGH**
TPHL	**Tempo de propagação de HIGH para LOW**

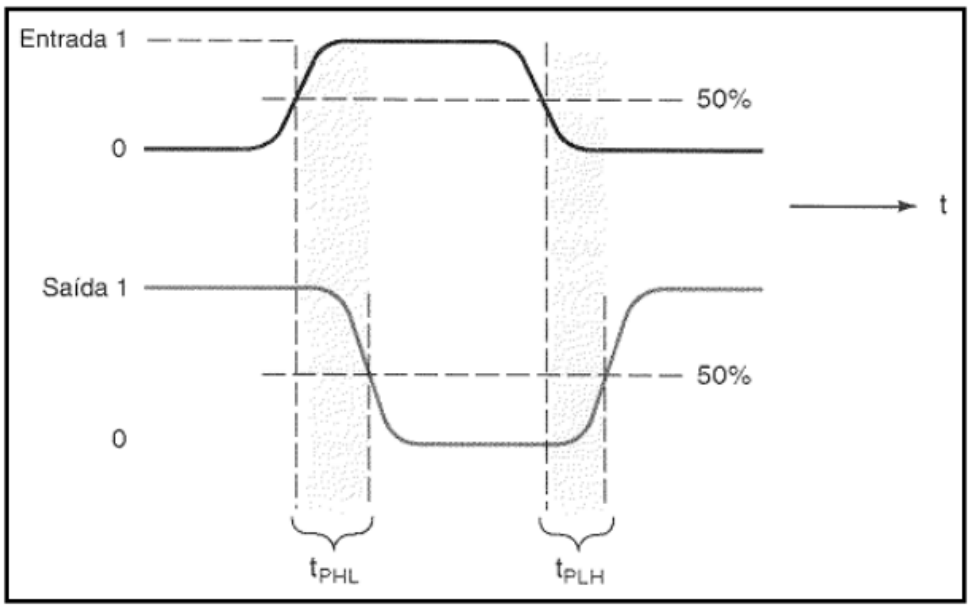
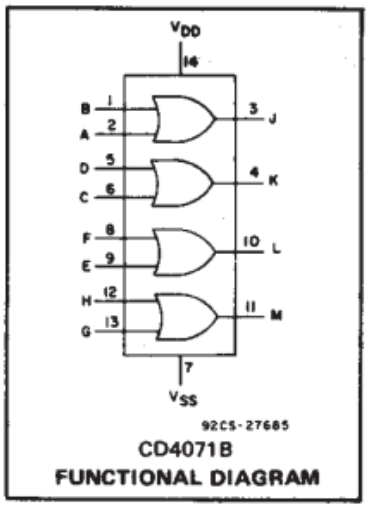


Figura 65 - Veja que a partir do momento em que a entrada interpreta a mudança de estado lógico, há um período de atraso na mudança de estado da saída. Elas não acontecem ao mesmo tempo.

CD4071B, CD4072B, CD4075B Types

Features:

- Medium-Speed Operation-t_{PLH}, t_{PHL} = 60 ns (typ.) at V_{DD} = 10 V
- 100% tested for quiescent current at 20 V
- Maximum input current of 1 μA at 18 V over full package-temperature range; 100 nA at 18 V and 25°C
- Standardized, symmetrical output characteristics
- Noise margin (over full package temperature range)
  1 V at V_{DD} = 5 V
  2 V at V_{DD} = 10 V
  2.5 V at V_{DD} = 15 V
- 5-V, 10-V, and 15-V parametric ratings
- Meets all requirements of JEDEC Tentative Standard No. 13B, "Standard Specifications for Description of 'B' Series CMOS Devices"



CD4071B
FUNCTIONAL DIAGRAM

Figura 66 - TPHL - tempo de propagação high to low

Analisando o datasheet acima podemos ver que o período de transição de estado do circuito integrado 4017 é de 60nS.

Sabendo o tempo de transição de estado, podemos calcular qual a frequência máxima de trabalho dessa porta, ou seja, qual a velocidade máxima de acionamentos possíveis não irá interferir nos valores de saída devido ao atraso característico da porta. Dessa forma, o número máximo de transições que poderão ser feitas em 1 segundo determinará a sua frequência máxima de trabalho.

$$Frequência(Hz) = \frac{1}{tempo\ (s)}$$

Sabendo que o tempo de subida é de 60nS e descida também é de 60nS, temos um ciclo completo com 120nS. Portanto, aplicando a fórmula:

$$frequência = \frac{1}{0{,}000000120} = 8.333.333Hz\ ou\ 8{,}3MHz$$

Portanto, essa porta lógica conseguirá responder a sinais de entrada com a frequência máxima de 8,3MHz.

FAMILIA TTL OU CMOS

Os circuitos integrados da família TTL (Transistor – Transistor Logic) tem uma arquitetura de funcionamento baseada em transistores. Já os circuitos integrados da família CMOS (Complementary Metal-Oxide Semiconductor) possuem uma arquitetura construtiva de transistores MOSFETs.

É importante pontuar que os componentes da família TTL são mais rápidos que os componentes da família CMOS. Isso se dá devido a diferença construtiva entre os mosfets e o os transistores bipolares de junção (TBJ).

Diferentemente dos TBJ, os mosfets possuem uma fina camada de oxido que isola a comporta (gate) que faz o acionamento do componente, do canal onde irá circular a corrente. Dessa forma, esses dois pontos isolados, gate e canal, se comportam como as placas de um capacitor criando certa capacitância. O gate ao receber tensão, terá primeiro que carregar essas “placas” para que, a partir de um determinado nível, o acionamento seja feito com sucesso.

Esse tempo de carregamento e suprimento dessa capacitância é o principal responsável pela baixa velocidade dos circuitos CMOS.

Método ERDINGER

Prática e Mão na Massa

CURSO DE
ELETRÔNICA FÁCIL

MÓDULO 3 – METODOLOGIA ERDINGER

Parabéns para você que chegou nesta etapa do nosso treinamento e que conseguiu assimilar todo o conteúdo disponibilizado anteriormente, sem pular nenhuma etapa. Com certeza você estará cada vez mais preparado para entender com riqueza de detalhes como a eletrônica digital se organiza e seus mecanismos de funcionamento.

Entraremos na fase do método ERDINGER, totalmente elaborado pensando na melhor forma de o aluno aprender os conceitos da eletrônica digital e programação, reconhecendo o motivo da utilização de cada ferramenta, código, e principalmente entendendo que um simples piscar de LED, por mais simples que seja, tem aplicações importantíssimas e que, se não houver conhecimento, seu princípio de funcionamento se torna complexo.

Então vamos nessa, com muita garra, determinação, entender o que é o método ERDINGER, como é estruturado, quais suas etapas, para que ao chegar no final de mais esse módulo possamos olhar para trás e dizer: “como o conhecimento é libertador e construtivo”.

É importante ressaltar que esse material é complemento do curso de Eletrônica Digital e Arduíno para iniciantes, portanto, o máximo aproveitamento dos temas abordados compreende não só a utilização deste ebook, mas também todas explicações, exemplos, dicas e metodologia aplicada nos vídeos disponibilizados pelo professor na plataforma.

Seja muito bem vindo(a) a mais um desafio, conte sempre com a família eletrônica fácil, estaremos juntos nessa caminhada superando e entendo os “mistérios” desse mundo incrível que é a eletrônica digital. Vamos nessa!

METODO ERDINGER – O que é?

A metodologia **ERDINGER (Exemplos Rápidos que Despertam Interesse e Geram Resultados)** tem sua base de aprendizagem na utilização de códigos de programação prontos, onde sua aplicação como ferramenta pedagógica irá gerar um resultado final que irá motivar o aprendizado do aluno, e então esse aluno motivado por ver o código funcionando parte para a investigação do que pode ter gerado esse incrível resultado final, quais comandos, recursos, ferramentas foram utilizadas para que esse exemplo de código funcionasse com sucesso.

Método ERDINGER e a primeira infância

Neste momento, ao ler esse título, você deve estar se questionando se estamos em um método de eletrônica digital ou de pedagogia, mas o fato é que o entendimento da metodologia aplicada neste treinamento é crucial para que o aluno compreenda os caminhos traçados e que, ao pular uma etapa, estará se limitando em aprendizados futuros.

Ao mencionar a primeira infância com a metodologia ERDINGER queremos que você pense, de maneira genérica e imediatista, como acontece o processo de alfabetização de uma criança.

Primeiro ela aprende a escrever e depois a falar, ou primeiro aprende a falar, imitando os adultos, para então aprender a escrever?

Pois é, parece obvio né, mas não seguimos essa regra quando vamos aprender algo do absoluto zero, e isso é um grande problema.

O método ERDINGER irá levar você estudante para o campo das experiências, testando códigos prontos para que, depois, você possa entender o que foi feito e então produzir seu próprio desenvolvimento. Primeiro você vai **"falar a língua de alguém"**, observar os resultados, entender cada parte dessa linguagem para que, a partir disso, possa produzir a sua própria linguagem, código, projeto, enfim, como queira chamar.

Mas professor, copiar não é aprender, isso não limita a criatividade? Ué, as crianças não aprenderam a dar suas primeiras palavras ouvindo as pessoas ao seu redor?

A grande questão não é copiar, mas ter o compromisso de estudar o que se copia para que o despertar da criatividade aconteça de forma leve com um vocabulário rico e adequado, **sem gerar frustração e desistência dos estudantes**.

FASES DA METODOLOGIA ERDINGER

Agora vamos entender como a metodologia se organiza, quais são suas etapas e o que será feito em cada uma delas.

Nesta primeira fase o aluno irá escolher um tema de aprendizado, depois irá buscar um código exemplo pronto dentro do tema escolhido, fará uma observação geral dos comandos que compõe o código escolhido, verificará a necessidade de hardware para que o código funcione com sucesso (se for acender um led, será preciso ter um led para acender), carregar o código no arduino, testar seu funcionamento, observar os resultados, e buscar uma aplicação pratica para o código escolhido.

1ª Fase – PRÁTICA

Veja abaixo como será dividia a primeira fase da metodologia ERDINGER.

SEQUÊNCIA DA FASE 1	EXEMPLOS
1 – TEMA ESCOLHIDO	Controle de entradas ou saídas digitais Leitura de entradas analógicas Escritas em saídas analógicas Envio de mensagem via porta serial
2 – BUSCAR EXEMPLO	Buscar exemplo na IDE do Arduino
3 – VISÃO GERAL	Observar absolutamente tudo que compõe o código do exemplo escolhido, desde *setup, loop, if, while, ; , (), digitalWrite, digitalRead,*
4 – REQUISITO DE HARDWARE	O que o exemplo escolhido necessita de parte física (hardware) para ser executado. A necessidade de hardware do exemplo do código escolhido (software).
5 – CARREGAR CÓDIGO E TESTAR	Utilizar o shield eletrônica fácil juntamente com o Arduino para carregar o código e testar seu funcionamento.
6 – ONDE USAR NO DIA A DIA	Onde, no campo prático, pode ser aplicado exemplo escolhido, por exemplo ligar e desligar uma lâmpada, um sistema de refrigeração, alarmes, etc.

CLIQUE AQUI PARA ACESSAR A AULA

2ª Fase – IMERSÃO

A fase 2 da metodologia ERDINGER consiste no entendimento do que foi utilizado no código exemplo escolhido na fase 1. É nessa etapa da metodologia que o aluno irá focar na busca pela compreensão de cada linha, código, símbolos, utilizados no exemplo, trazendo o entendimento necessário para que ele possa utilizar da maneira que desejar futuramente.

Veja como a fase 2 é organizada.

SEQUÊNCIA DA FASE 2	EXEMPLOS
1 – LISTAR DÚVIDAS	Observando o código escolhido na fase 1 o aluno deverá anotar tudo o que não entender, suas dúvidas deverão ser registradas para que, a partir delas, a busca pelo conhecimento aconteça.
2 – BUSCAR CONTEÚDO	A busca pelo conteúdo explicativo terá como base as dúvidas listadas anteriormente. O aluno irá consultar a plataforma do curso EDI, o site Arduíno CC, Ebooks, Blogs, comunidade Família Eletrônica Fácil, e onde mais o aluno sentir necessidade para que todas as dúvidas sejam eliminadas e tragam para o aluno propriedade sobre a utilização de cada ferramenta estudada.
3 – APLICAR A PIRAMIDE DO CONHECIMENTO	

3ª Fase – DESAFIO INDIVIDUAL

Na fase 3 o aluno irá aplicar os conhecimentos obtidos na fase 1 e fase 2 da metodologia. Veja como será organizada essa fase de estudos e prática.

SEQUÊNCIA DA FASE 3	EXEMPLOS
1 – DESENVOLVER PROJETOS	Desenvolver de 3 a 5 projetos, que serão denominados de F12 (fase 1 e 2 de conhecimentos obtidos). Se o aluno aprendeu a piscar um led, deverá então criar um projeto para piscar 2 leds por exemplo, ligar e desligar um buzzer, uma lâmpada, um motor, enfim, essa lógica estudada na fase anterior deverá fazer parte dos projetos que serão desenvolvidos pelo aluno nesta etapa.
2 – DOCUMENTAR O PROGRESSO	Registrar todo o processo, inclusive com erros e acertos, com fotos e vídeos sempre guardando em pastas organizadas (será mostrado a frente). #projetoF12
3 – ONDE APLICAR	Onde aplicar no nosso cotidiano esses projetos que foram desenvolvidos. Buscar aplicações práticas para o seu projeto.

4ª Fase – DESAFIO DO MESTRE

Na última etapa da metodologia o aluno deverá criar de 2 a 4 projetos proposto em forma de desafios pela equipe eletrônica fácil, incluindo todos os aprendizados obtidos nas fases anteriores dessa metodologia.

Essa fase é a união de todas as fases anteriores da metodologia, logo, o aluno que não pulou nenhuma etapa, desenvolverá essa fase com muita tranquilidade e conhecimento.

É muito importante que, o aluno que já possuir algum conhecimento em linguagem de programação, ao realizar os desafios, não utilize recursos que não tenham sido estudados nas fases anteriores, isso faz parte do desafio, utilizar apenas aquilo que foi estudado e que, a partir dos conhecimentos adquiridos, o aluno consiga demonstrar propriedade na aplicação de recursos e ferramentas para a realização das atividades.

SEQUÊNCIA DA FASE 4	EXEMPLOS
1 – REALIZAR OS DESAFIOS	Desenvolver projetos de 2 a 4 projetos que solucionem os desafios propostos pela equipe eletrônica fácil.
2 – REQUISITO PARA REALIZAÇÃO	Desenvolver o código de programação Fotografar Filmar Compartilhar projeto com amigos do treinamento.

Depois de entendermos cada etapa da metodologia e o que compõe cada fase, chegou a hora de aplicar tudo que foi proposto pela metodologia ERDINGER, para isso vamos escolher o tema norteador dos nossos estudos

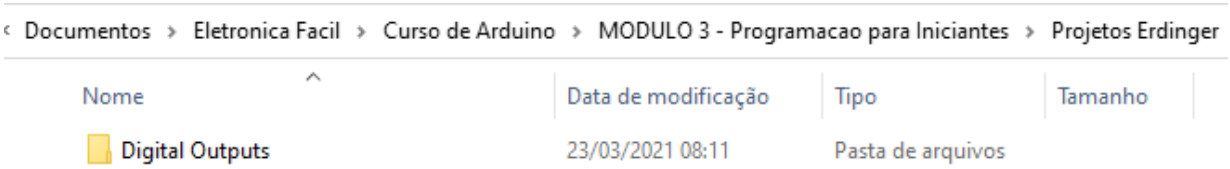
Tema: **CONTROLE DE SAÍDAS DIGITAIS**.

INICIANDO OS ESTUDOS E APLICAÇÃO DO METODO ERDINGER

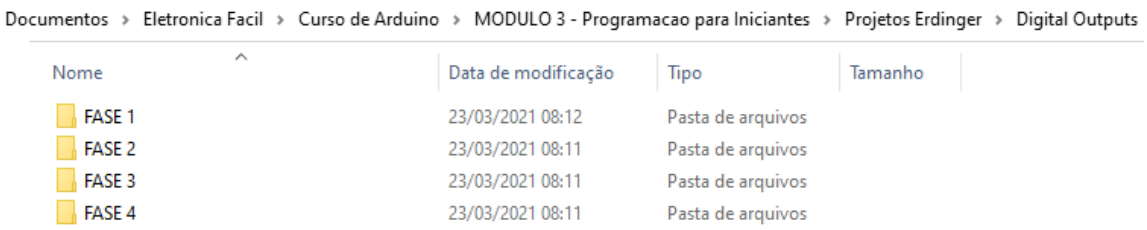
Criação de pastas e organização dos estudos

Atenção: Antes de começar a ler e realizar o passo a passo do método Erdinger, vamos criar a sequência de pastas conforme recomendado nas explicações anteriores, nelas você poderá documentar e organizar todo seu aprendizado e evolução no treinamento e na programação do Arduino.

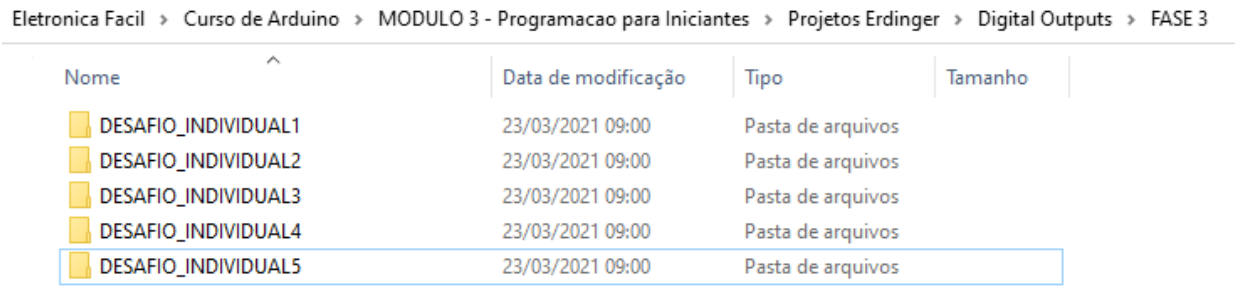
➔ Crie uma pasta chama Digital Outputs (tema escolhido)



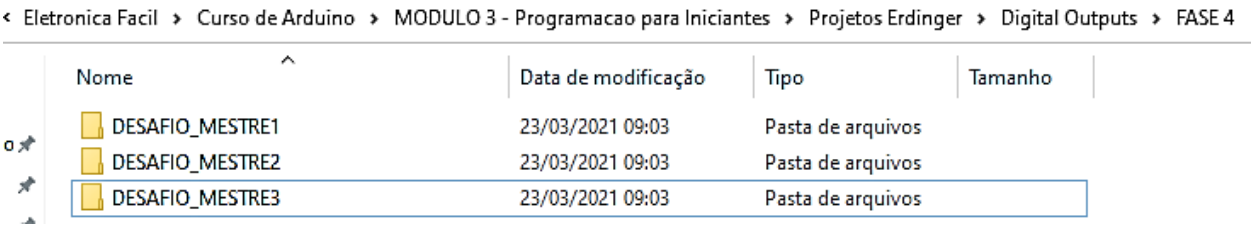
➔ Dentro da pasta Digital Outputs crie uma pasta para cada fase da metodologia



➔ Dentro da pasta fase 3 crie 5 pastas para os desafios individuais



➔ Dentro da pasta FASE 4 crie 3 pastas para os desafios do mestre

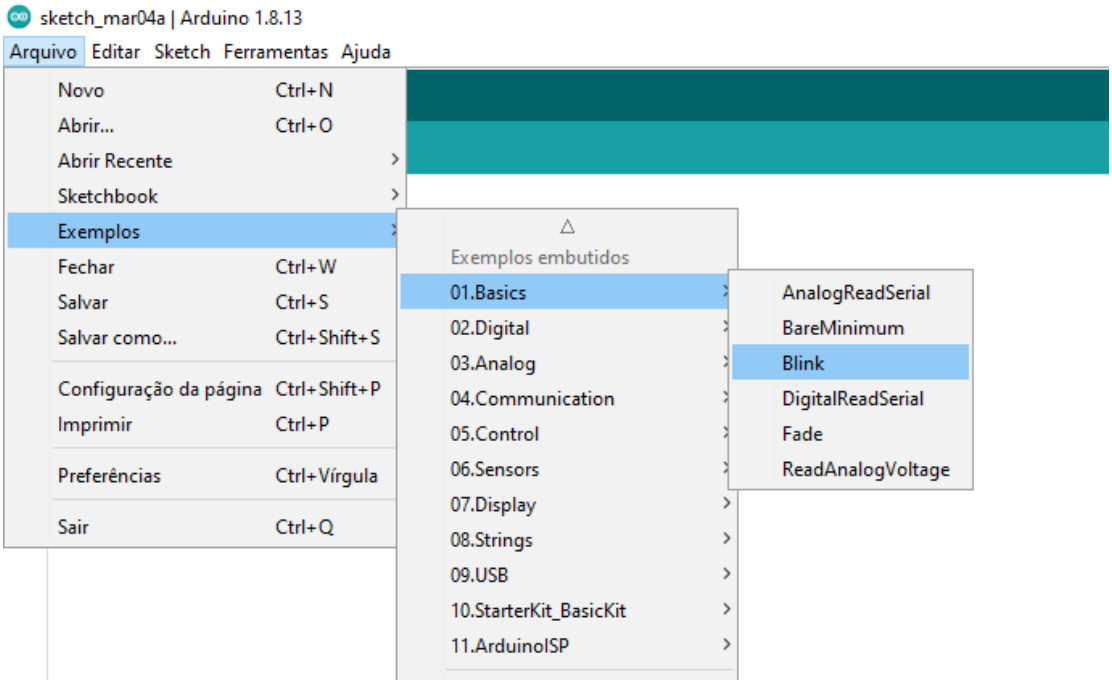


INICIANDO A FASE 1 – Prática e mão na massa

1. **Escolher o Tema e Objetivo:** Aprender a ligar e desligar dispositivos eletrônicos que possam gerar trabalho com a passagem da corrente elétrica.

- Leds, Relés, Buzzer e outros dispositivos de saída digital 0 / 1

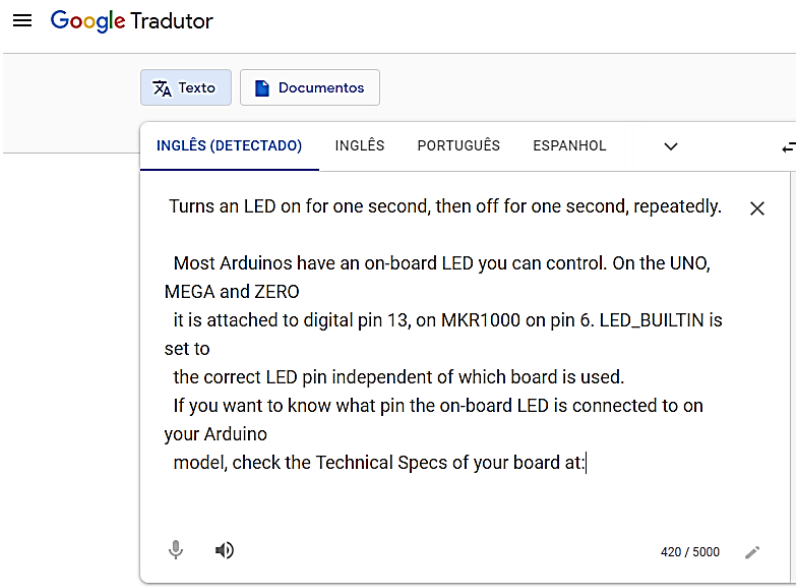
2. **Escolher exemplo:** Basics → Blink



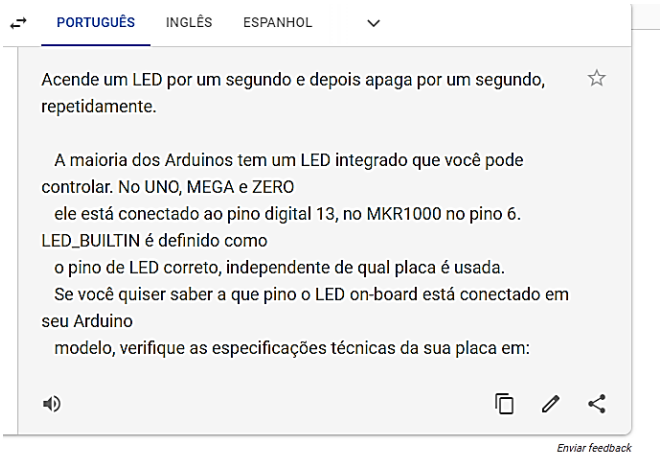
3. Visão Geral do exemplo:

```
Blink
/*
  Blink

  Turns an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
  it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
  the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected to on your Arduino
  model, check the Technical Specs of your board at:
  https://www.arduino.cc/en/Main/Products

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Blink
*/
```

Caso o aluno não tenha domínio da língua inglesa, não há motivos para preocupação, basta traduzir com a utilização do google tradutor.



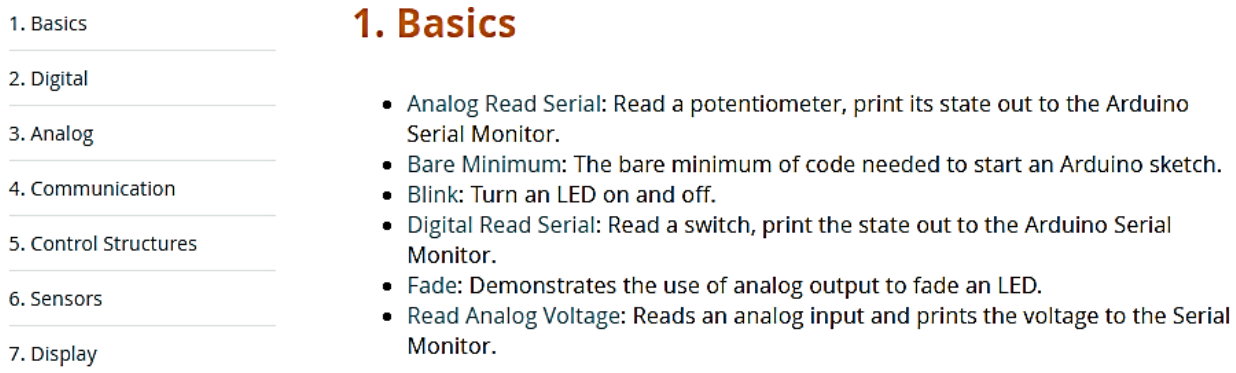
A dificuldade antes encontrada com a língua estrangeira ficará assim:



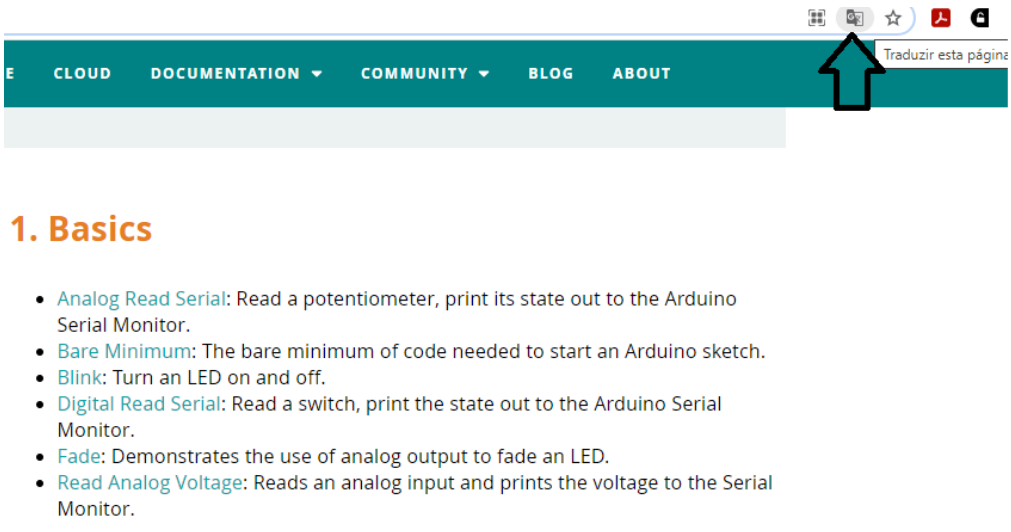
Para saber ainda mais sobre este projeto, você poderá acessar o site oficial do Arduino.cc http://www.arduino.cc/en/Tutorial/Blink e lá terão muitas informações que na Fase 2 de IMERSÃO você poderá usar para aprender e evoluir ainda mais em seu conhecimento sobre Arduino.

4. **Requisitos de Hardware:** Aqui você precisa saber quais pinos do seu Arduino serão usados no exemplo escolhido e também quais componentes serão necessários para testar o Exemplo Rápido que Desperta o Interesse e Gera Resultado.

Acessando: https://www.arduino.cc/en/Tutorial/BuiltInExamples você poderá ver todos os exemplos presentes na IDE Arduino e todos os descritivos de funcionamento e informações sobre o exemplo Blink.



Se você desejar traduzir a página você também pode clicar no ícone de tradução do seu navegador.



Clicando neste Link você poderá ir até a explicação completa do exemplo Blink: https://www.arduino.cc/en/Tutorial/BuiltInExamples/Blink

Hardware Required

- Arduino Board

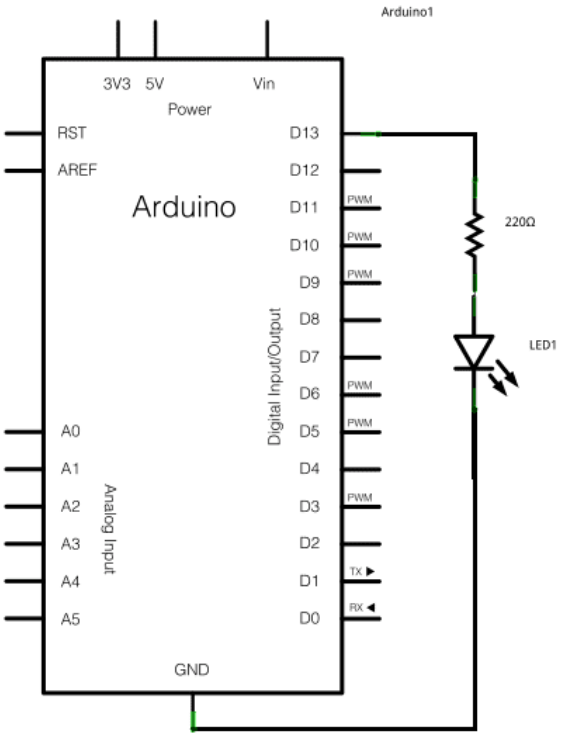
optional

- LED
- 220 ohm resistor

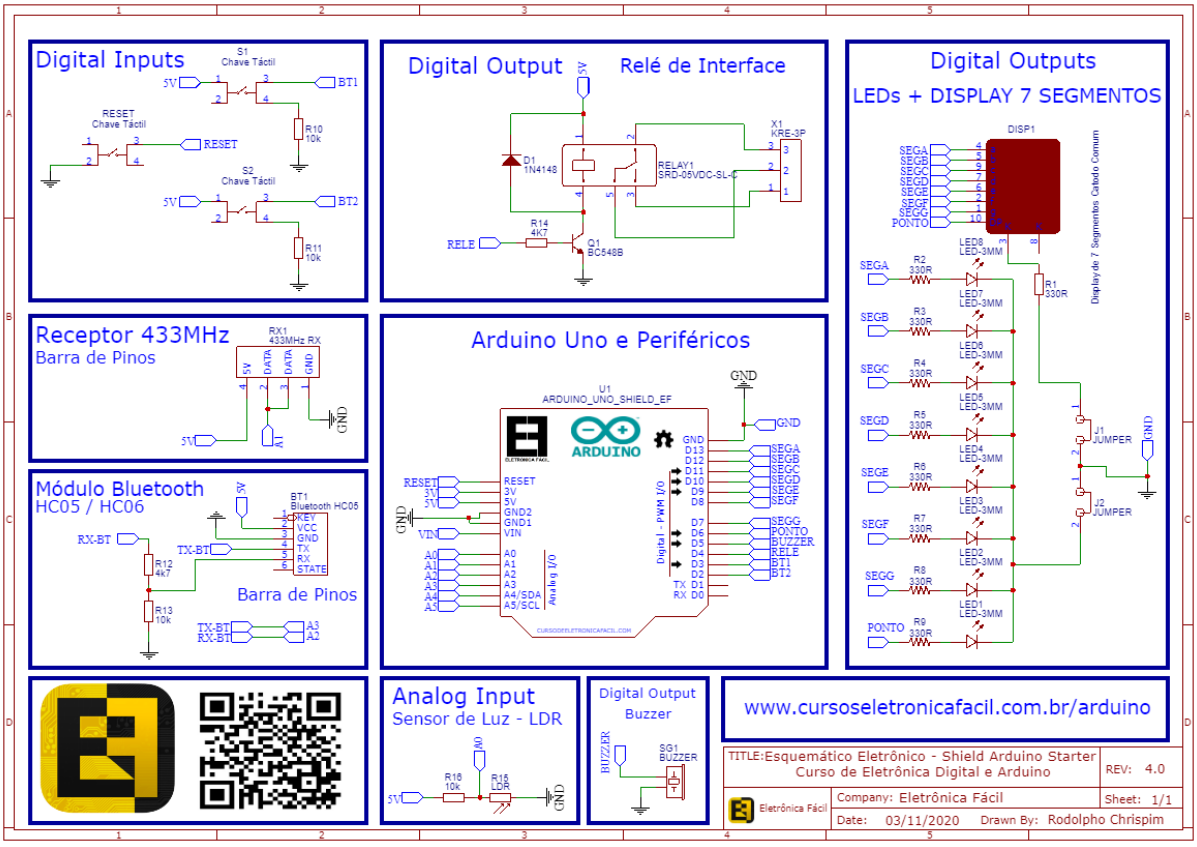
Ao consultar os requisitos na plataforma do Arduino, o mínimo de hardware necessário para testar o projeto escolhido será um LED e um resistor de 220 Ohms. No Arduino Uno o pino que controlará o LED será o D13

Esquema Eletrônico Sugerido pelo site:

Schematic



Os alunos que já montaram o Shield para Arduino Starter **NÃO PRECISARÃO MONTAR ESSE DIAGRAMA NO PROTOBOARD** para executar o código exemplo. O shied já vem com as ligações e requisitos necessários para a execução desse código exemplo, veja o esquema eletrônico abaixo:



Conforme o esquema acima, o pino D13 do Arduino Uno irá controlar tanto o SEG **A** do display de 7 segmentos quanto o LED 7, e você selecionará qual deseja controlar através do jumper J1 e J2 do Shield.

O Shield irá acelerar muito o seu aprendizado de programação por conta de a maioria dos exemplos disponíveis para teste precisarem de um hardware que já se encontra no shield, dessa forma, nossa maior preocupação será apenas aprender a programar.

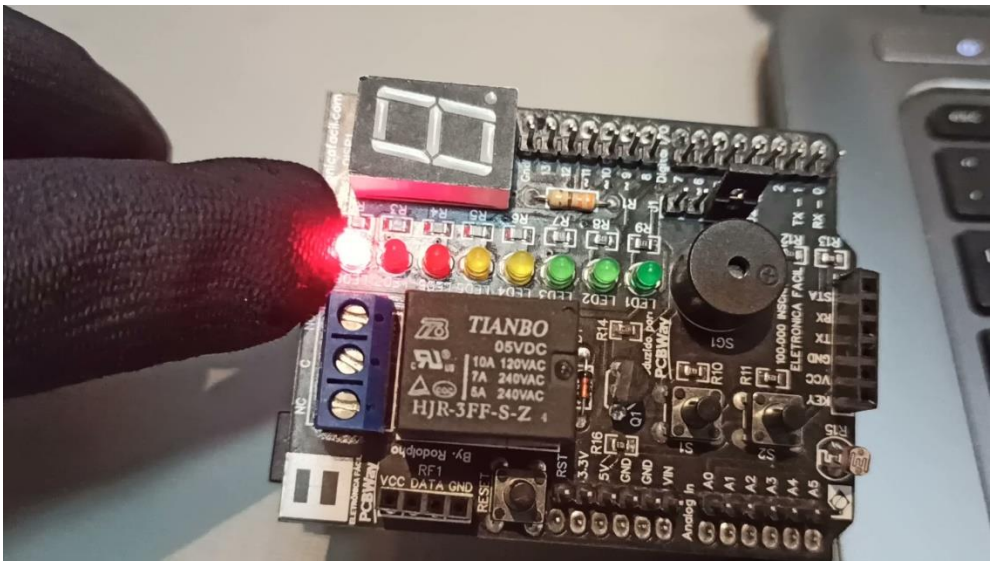
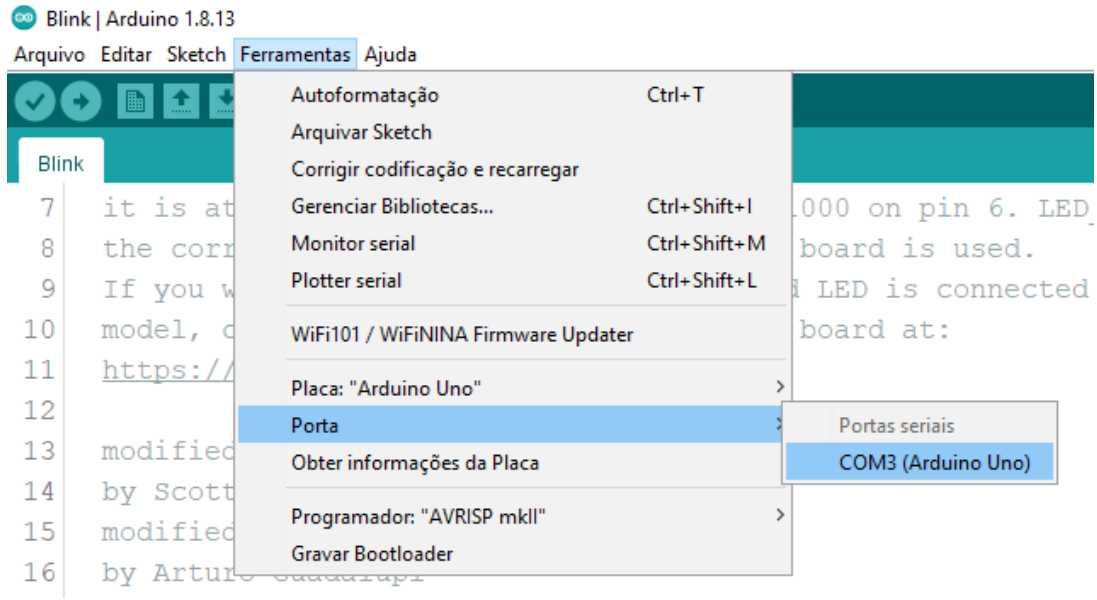


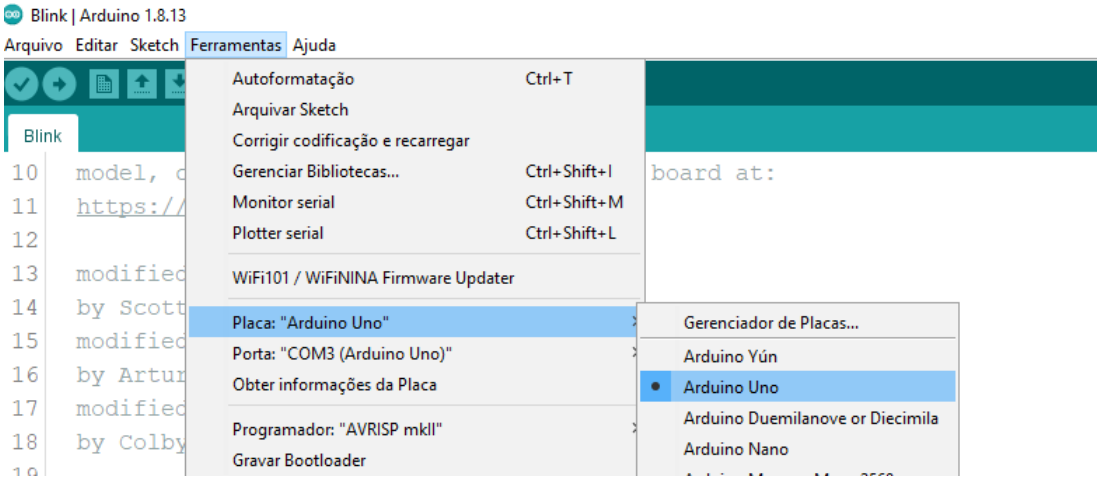
Figura 67 - Shield controlando LED

5. **Carregar e Testar:** Após conectar seu shield no arduino, abrir o código exemplo e plugar na porta USB do seu computador, siga os procedimentos abaixo.

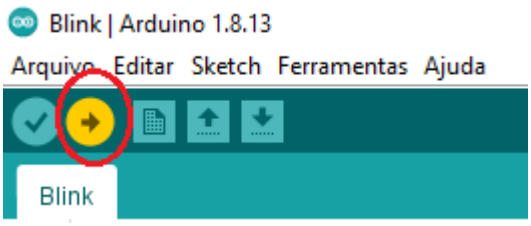
➔ Selecione a porta serial disponível



➔ Selecione a placa arduino que será programada



➔ Clique em carregar código



➔ Verifique a barra de carregamento sendo preenchida até o final e exibindo que o processo de gravação está acontecendo com sucesso.

➔ Quando mostrar a palavra CARREGADO, seu programa exemplo foi gravado com sucesso e já entrará em funcionamento no Arduino Uno.

Assista o vídeo mostrando a atividade prática sendo comprovada e testada com sucesso clicando no link a seguir ou scaneando o QR Code com seu Smartphone.

http://bit.ly/Pratica_ARDUINO_EDG1

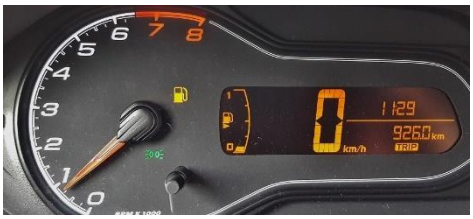
Lembre-se se salvar o exemplo blink na pasta Fase 1

6. **Reflexão:** Onde poderíamos aplicar esse projeto que aprendemos na fase 1? Onde usaríamos isso em equipamentos, processos e produtos? Tem algo que usa este funcionamento do projeto blink na vivência profissional e pessoal? Liste todas as aplicações práticas que lembrar sobre o princípio do blink em seu funcionamento prático.

Exemplo 1: Luz sinalizadora da seta e pisca alerta do automóvel.

Exemplo 2: Buzzer do Micro-ondas sinalizando que o tempo selecionado de aquecimento acabou.

Exemplo 3: Buzzer e LED da geladeira que indicam que a porta ficou aberta por muito tempo.



Exemplo 4: Led sinalizador no painel veicular que pisca se o combustível está acabando.

Exemplo 5 – Led de um soft starter que pisca enquanto o motor está partindo e ainda não chegou na sua tensão nominal de trabalho.

Entre muitos outros exemplos, escreva abaixo quais são os exemplos práticos que você gostaria de citar da aplicação do projeto blink.

Considerações finais da fase 1 deste projeto aplicando o método ERDINGER:
O que você já aprendeu ao executar na prática o seu primeito Blink?

Liste abaixo tudo aquilo que aprendeu com esta fase 1 e que definitivamente você nunca pensou que seria capaz de fazer sozinho:

Neste momento você se sente mais motivado em entrar no mundo da programação de Arduino? O que está sentindo após fazer este projeto?

INICIANDO A FASE 2 – Imersão, o conhecimento é libertador

Chegamos na fase de imersão da metodologia ERDINGER. Vamos literalmente imergir, mergulhar, focar no código utilizado e listar todas as dúvidas que ficaram com relação ao **exemplo Blink escolhido** na fase anterior. Veja alguns exemplos de questionamentos que podemos nos fazer:

- *O que estava presente no programa exemplo da fase 1 que não entendi?*
- *Quais são as linhas de programação que não tenho ideia do que fazem ou como funcionam?*
- *Como o professor encontrou aquele site com o programa exemplo?*
- *Quais foram as configurações realizadas na função setup?*
- *Se eu precisar explicá-las passo a passo e eu conseguiria?*
- *Quais foram as tarefas executadas na função loop?*
- *Você consegue mandar um áudio resumo explicando o que faz cada linha de programação e do seu nível de importância para o projeto?*

Estes são alguns exemplos de perguntas que você pode fazer para testar o quanto você sabe do código exemplo escolhido na fase 1, e esses questionamentos serão os norteadores para que você inicie a busca por conteúdo, explicações, artigos, vídeos, módulos do curso, livros, e-books, revistas e qualquer tipo de fonte de informação confiável e que possa gerar valor e conhecimento, buscando sempre entender como tudo funciona no projeto escolhido.

*Lembre-se sempre, mais **vale entender** de forma faixa preta o **essencial,** a raiz, a base e o alicerce de tudo, do que ficar pulando como canguru para todo lado, somente copiando e colando códigos prontos para fazer as coisas funcionarem, e com isso gerar a falsa ilusão que você é um programador de Arduino.*

Um verdadeiro programador é aquele que "diz" como o sistema deve funcionar, através da sua **linha de raciocínio,** do seu conhecimento, mesmo que usando **simples recursos da linguagem de programação**, mas que com muita frequência, intensidade e luta pelo aprendizado, **se tornará faixa preta** naquilo que sonha, e não apenas um indivíduo faixa branca que copia e cola e se diz desenvolvedor. Portanto, a fase 2 do método ERDINGER é crucial para você seja um programador e desenvolvedor de verdade no futuro, e não um canguru copiador recheado de sucesso e felicidade no presente e que, no futuro, não conseguirá desenvolver nada sozinho.

Depois do sermão da montanha (risos), vamos para um exemplo de como executar a fase 2 do método ERDINGER.

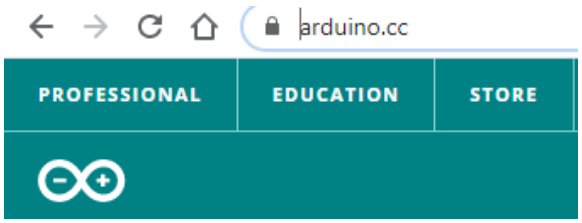
O desafio agora será entender como a programação do projeto Blink funciona, todos os recursos e ferramentas utilizadas, e como a estrutura foi desenvolvida.

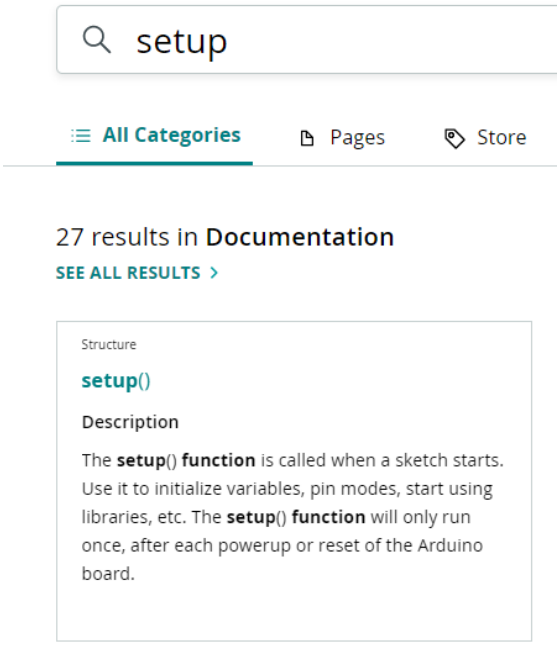
```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

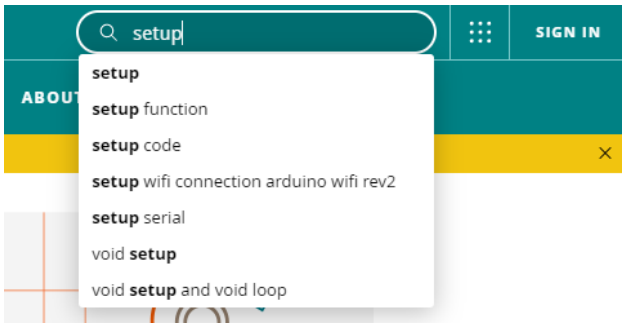
O que é void setup()

Logo de início já é possível questionar o que é o tal **void setup()** neste código utilizado. Para entender o que esse comando faz e o porquê ele está aí, vamos iniciar nossa pesquisa.

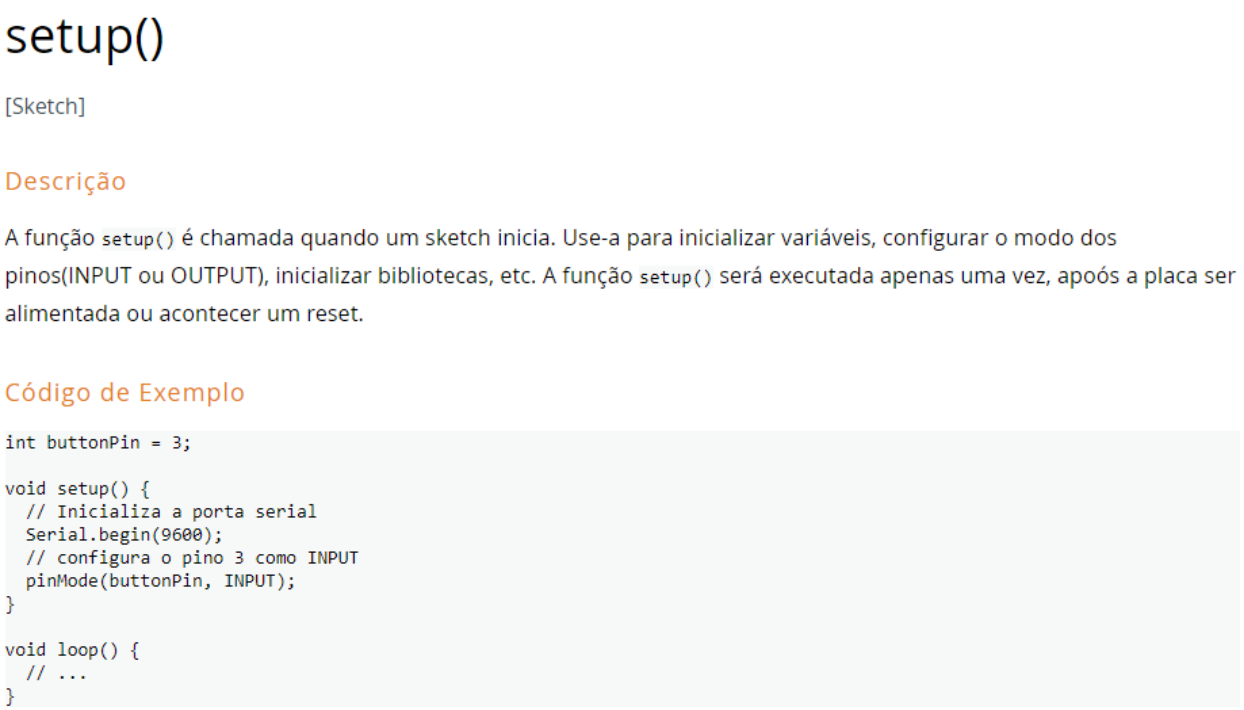
Acessando o site https://www.arduino.cc/ e no campo de pesquisa buscando por **setup** encontraremos informações muito importantes sobre ele.







Ao clicar no resultado apresentado durante a pesquisa você encontrará informações detalhadas sobre o que é a FUNÇÃO setup*(). (Altere o idioma da página para o português para facilitar o entendimento)*



setup()

[Sketch]

Descrição

A função `setup()` é chamada quando um sketch inicia. Use-a para inicializar variáveis, configurar o modo dos pinos(INPUT ou OUTPUT), inicializar bibliotecas, etc. A função `setup()` será executada apenas uma vez, apoós a placa ser alimentada ou acontecer um reset.

Código de Exemplo

```
int buttonPin = 3;

void setup() {
  // Inicializa a porta serial
  Serial.begin(9600);
  // configura o pino 3 como INPUT
  pinMode(buttonPin, INPUT);
}

void loop() {
  // ...
}
```

Ao fazer essa pesquisa encontramos também a “mina de ouro” do conhecimento, ou seja, ela acaba de informar que a **função setup ()** tem o objetivo de realizar as configurações iniciais do projeto (entradas e saídas) e é executada no momento que o sketch (projeto) se inicia, após o microcontrolador ser alimentado ou passar por um reset.

O setup será utilizado para configurar os pinos do arduino com a função de INPUT (entrada) ou como OUTPUT (saída). Precisamos avisar o microcontrolador se um determinado pino irá ler uma informação ou se ele irá ativar um dispositivo. Para isso usaremos a função setup ().

Todos os dispositivos de I/O devem ser configurados na função setup obrigatoriamente.

O que é uma função

Vimos anteriormente que o setup é uma função, e que dentro dessa função iremos configurar as entradas de INPUT e OUTPUT. Mas, o que de fato é uma função?

Função é um conjunto de instruções que, executadas sequencialmente, uma após a outra, irão realizar uma determinada tarefa.

```
acende_led ()
{
        ACENDER LED;
}
```

A função acima executa uma única tarefa que é acender um led. Pronto!

As funções geralmente possuem nomes que já facilitam o entendimento do trabalho que executarão no código. Veja alguns exemplos:

acende_led(); desligaLed(); ler_botão(); setup(); loop(); digitalWrite(); digitalRead();

Para que servem as chaves {}

As chaves serão utilizadas para indicar a abertura e fechamento das instruções que serão executadas. Todas as instruções da função deverão estar dentro das chaves.

```
desliga_tudo ()
{
    desligar led1;
    desligar led2;
    desligar led3;
    desligar led4;
}
```

Para que servem os parênteses ()

Os parênteses nas funções servem para receber, ou não, **parâmetros** (informações, valores, para que a função execute suas tarefas). Os parâmetros podem possuir tipos, como **int, float, char**, mas isso veremos mais à frente. Veja abaixo alguns exemplos de funções recebendo informações por parâmetros:

Essa é uma **forma didática** de representar uma Função que, recebe um valor por parâmetros (dentro dos parênteses), e esse valor determinará a forma de pagamento de uma compra em uma loja:

```
void pagamento (int forma_de_pagamento)
{
        se forma_de_pagamento for1 = pagar com cartão;
        se forma_de_pagamento for2 = pagar com dinheiro;
        se forma_de_pagamento for3 = pagar com cheque;
        se forma_de_pagamento for4 = pagar com pix;
}
```

Veja que se a função receber um valor do tipo inteiro (int), 1 ao 4, uma escolha de um tipo de pagamento será executada.

```
void pagamento (2)
{
        se forma_de_pagamento for1 = pagar com cartão;
        se forma_de_pagamento for2 = pagar com dinheiro;
        se forma_de_pagamento for3 = pagar com cheque;
        se forma_de_pagamento for4 = pagar com pix;
}
```

Essa é uma forma bem genérica de você entender *para que servem os parâmetros, mas eles podem ser utilizados de diversas formas, portanto, não se prenda a esse* ***exemplo didático****, veremos muitos outros no decorrer do treinamento.*

O que é o VOID

Segundo o arduino.cc o termo **void** é usado quando você deseja criar e declarar uma função que não precisa retornar nenhum dado, ou seja, você irá chamá-la, ela será executada e não precisará retornar nenhum valor, resposta, dado e etc. Quando nos referimos a retornar valores estamos dizendo que essa função não irá disponibilizar nenhum dado após executar sua tarefa. Por exemplo, uma função de soma, recebe dois valores e retorna o resultado, que será a soma dos dois números inseridos, portanto essa função não será void, pois ela retorna um resultado.

Como a setup apenas realiza configurações e inicializações, não é necessário retornar nenhum valor, basta executar as configurações necessárias e está tudo certo, por isso ela é do tipo **void.**

```
int soma ( int valor_1 , int valor_2)
{
    int a = valor_1;
    int b = valor_2;
    int soma = a + b;

    return soma;
}
```

Figura 68 - Exemplo de função que retorna um valor do tipo inteiro -> "int"

void

[Data Types]

Descrição

A palavra chave `void` é usada apenas em declarações de funções. Ela indica que é esperado que a função não retorne nenhuma informação para a função da qual foi chamada.

Código de Exemplo

O código mostra como usar a palavra chave `void`.

```
// são realizadas ações nas funções "setup" e "loop"
// porém não é retornada nenhuma informação quando as mesmas são executadas

void setup() {
  // ...
}

void loop() {
  // ...
}
```

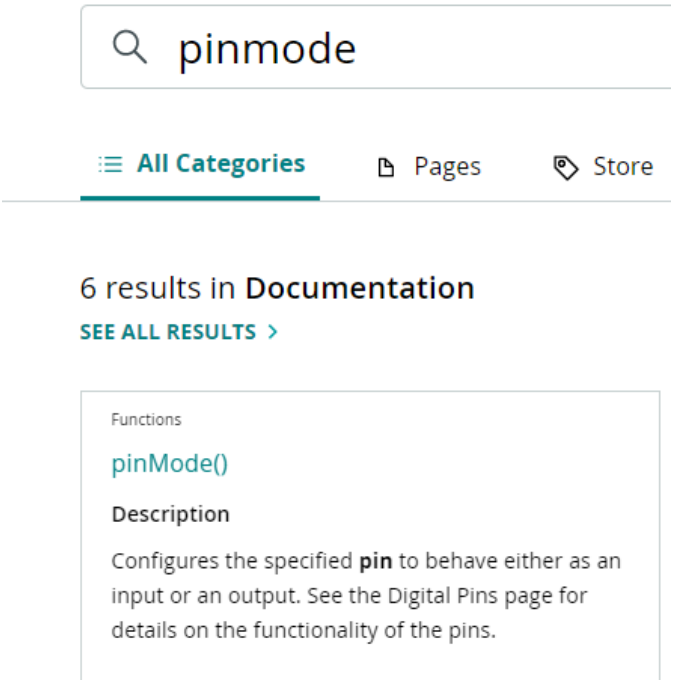
O mesmo acontece com a **função loop()** que recebe o tipo **void**. A função loop precisa apenas ficar executando repetidamente suas instruções, sem retornar nenhum valor, por isso sua definição é como **void.**

Depois te ter estudado todos os assuntos anteriores, veja se seu olhar mudou ao analisar o código blink.

```
25 // the setup function runs once when you press reset or power the board
26 void setup()
27 {
28   // initialize digital pin LED_BUILTIN as an output.
29   pinMode(LED_BUILTIN, OUTPUT);
30 }
31
32 // the loop function runs over and over again forever
33 void loop()
34 {
35   digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
36   delay(1000);                       // wait for a second
37   digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
38   delay(1000);                       // wait for a second
39 }
```

O que é a função pinMode()

Ao fazer nossa consulta na plataforma do arduino.cc obtivemos esse resultado:



Basicamente a função pinMode() irá configurar determinados pinos como entradas ou como saídas.

```
1  void configurações()                  //função de configurações gerais
2  {  // abre conjunto de instruções
3      configurar_Pino (pino 13, saída);  //função para configurar pino como entrada ou saída
4  }  //fecha conjunto de instruções
5
6  void setup ()
7  {
8      pinMode (13, OUTPUT);
9  }
```

Veja no exemplo acima que dentro da função de configurações, foi usado a função pinMode() para configurar o pino 13 como saída.

A função pinMode recebe por parâmetros o pino a ser configurado, e o modo como esse pino deve trabalhar. O modo do pino deve ser escrito em maiúscula, INPUT ou OUTPUT, e o pino deverá ser presentado com um número inteiro, 13 por exemplo.

pinMode (pino, MODO)

O nome da função já revela o que ela faz, veja:

Pin = pino **Mode = Modo**

pinMode () = modo do pino (entrada ou saída)

A função pinMode possui um nome super didático, que por sinal é o objetivo do Arduino. Como visto, pinMode significa o modo que o pino vai operar, e acredite, você a partir de agora vai usar muito esta função, pois na maioria de seus projetos, você terá que configurar os pinos como entrada ou saída.

Além de controlar saídas, você terá que ler dispositivos de entrada como botões, sensores, chaves e etc, pois se você parar para pensar, qual é o projeto ou produto que não tem pelo menos um botão de start e stop? Ou aumenta e diminui? Pois é, então isso só reforça o quanto terá que aprender a usar o pinMode daqui para frente para configurar as entradas e saídas.

O que é o LED_BUILTIN

O LED_BUILTIN nada mais é do que o led que já vem inserido na placa do arduino e que está ligado no pino 13, portanto, essa representação LED_BUILTIN significa o número 13 para o programa. Dessa forma, onde a palavra LED_BUILTIN for colocada representará o valor 13, que é a representação do pino onde o led está conectado.

Inclusive quando você carregar o programa Blink do exemplo, será este LED que irá piscar mostrando que o programa está funcionando com sucesso, e como você viu no vídeo

da Fase 1 do método ERDINGER, ao conectar o Shield Starter do Eletrônica Fácil, o LED que está sendo controlado pelo pino 13 também irá piscar.

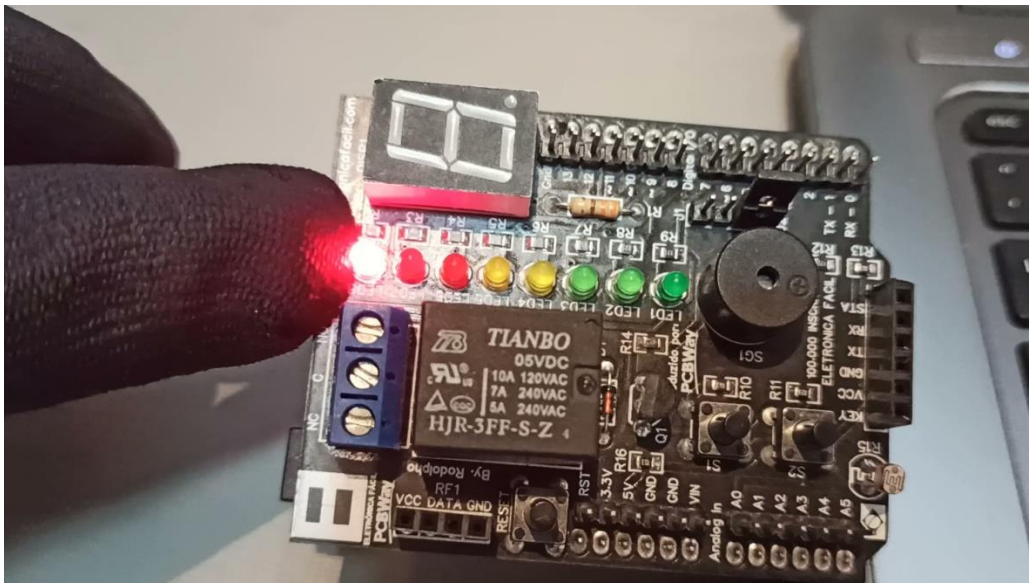


Figura 69 - led controlado pelo pino 13

Onde usar o ponto e vírgula;

O ponto e vírgula indica o final de um comando, um fim de instrução em uma linha, ou seja, é a indicação de encerramento de uma tarefa.

```
void setup ()
{
    pinMode (13, OUTPUT);
}
```

Figura 70 - ponto e vírgula indicando fim da função pinMode.

Erros mais comuns de compilação

Um ponto importante a se observar é que ao escrever corretamente os termos LED_BUILTIN, e OUTPUT, na IDE de programação, eles ficarão com a cor azul claro, mostrando ao programador que a grafia está correta e o compilador reconheceu esses termos.

Outro ponto é considerar que estes nomes e outros tantos, são case sensitive, ou seja, são sensíveis a maiúsculas e minúsculas, e devem ser escritos fielmente conforme o padrão, caso contrário, ao executar o código um erro será apresentado e a danada da cor laranja poderá aparecer. Veja abaixo:

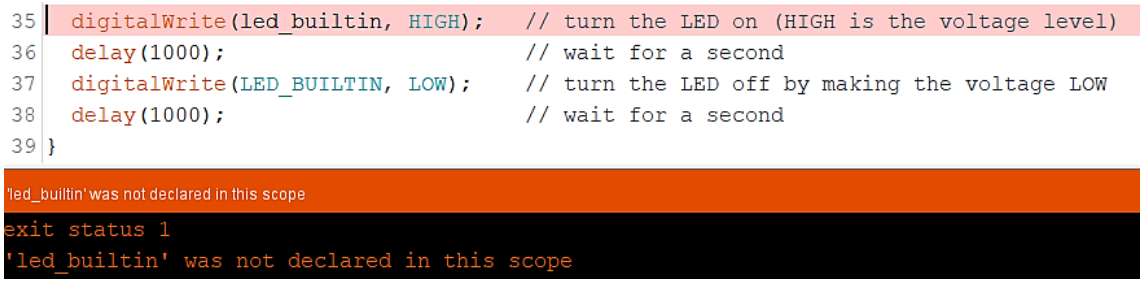
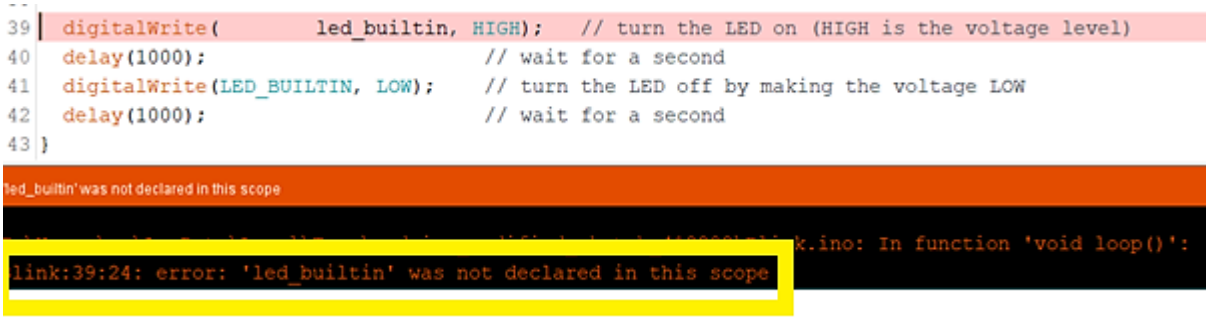


Figura 71 - LED_BUILTIN escrito de forma errada, com letras minúsculas.

Como pode ver acima**, ao escrever led_builtin minúsculo**, o compilador detectou um erro chamado **"was not declared in this scope",** que significa que o tal do nome led_builtin não foi declarado no projeto como uma variável, ou seja, não foi reconhecido, logo, como a palavra é desconhecida o compilador acusa o erro e inclusive aponta para você onde ele está. Veja:



Neste caso o compilador indicou que a linha onde está o erro é a 39 e o caractere da linha seria o 24, ou seja, a partir do caractere 24 começa o erro de sintaxe. Neste caso a escrita em minúsculo conforme citado acima é o erro apresentado.

Assistam o vídeo que fiz com muito carinho para todos vocês scaneando o QR Code ao lado e aprendendo outros erros muito frequentes no mundo da programação de Arduino para quem está começando do absoluto zero, e que tenho a certeza que você poderá cometê-los no decorrer do tempo. Posso contar com vocês?

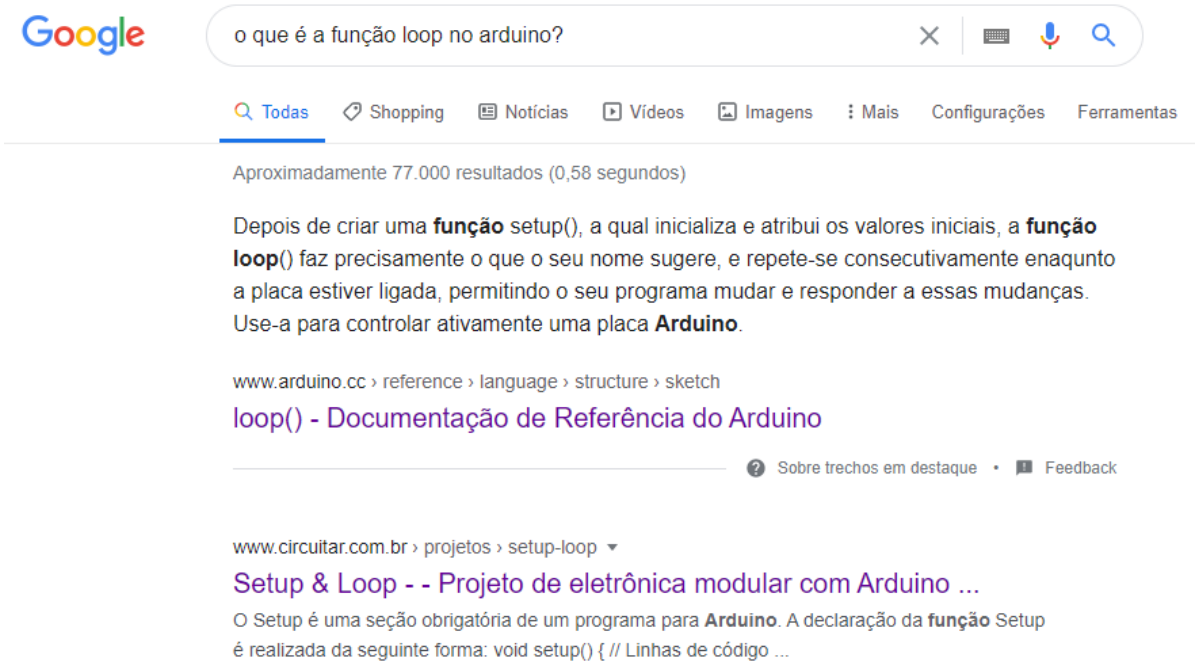
O que é a void loop()

A loop é uma função que executará suas instruções em ordem, uma após a outra, até que a placa seja desligada, ou seja, assim que todas suas instruções tiverem sido executadas ela retorna no início de suas instruções novamente. Ela recebe o tipo void pois não retornará nenhum valor ou dado para outra etapa do programa.

```
void loop()
{
    instruções...;
    ...;
    ...;
}
```

Toda a lógica de programação ficará dentro da função loop.

Digitando no Google a seguinte pergunta: **O que é a função loop no arduino?** Temos inúmeros links possíveis para acessar e aprender, e o primeiro apresentado é o https://www.circuitar.com.br/projetos/setup-loop/index.html, vamos usá-lo como referência agora.



Loop

Assim como a seção Setup, o **Loop** também é obrigatório em um programa para Arduino. A sua declaração é feita da seguinte forma:

```
void loop()
{
// Linhas de código do loop
}
```

A grande maioria do seu código será executado dentro dessa seção. Após a execução do `setup()`, o `loop()` é iniciado. O programa começa logo após a abertura da chave ({), e o processador vai executando as linhas de código até chegar na chave de fechamento (}). Uma vez chegado ao fim, ele pula de volta para a primeira linha do loop e começa tudo de novo.

A função `loop ()` será executada para sempre, ou até que você faça upload de um novo código, reiniciando o processo. Ela também pode ser reiniciada resetando o Arduino (através do botão de reset por exemplo).

Como podemos perceber, a função loop () é aquela que será executada após o setup(). Ao entrar nela, todos os códigos que estiverem dentro das chaves {} serão executados em loop, ou seja, de forma contínua no decorrer do tempo, portanto**, todas as atividades dos projetos e sua lógica de funcionamento estarão do loop ().**

Como você já aprendeu anteriormente, o void significa que o loop não retornará nenhum valor depois de executado, e **os parênteses vazios " () " indicam que esta função não receberá parâmetros para usar dentro de sua atividade.**

O que é a digitalWrite()

A função digitalWrite, como o próprio nome sugere, "escreve" em um determinado pino um nível lógico digital, que pode ser HIGH, ou LOW. Essa função recebe 2 parâmetros, pino e nível lógico, veja:

```
void loop()
{
    digitalWrite (13, HIGH);
}
```

Dessa forma, a função mostrada acima da forma como foi configurada, irá colocar nível logico alto no pino 13.

Aplicando o método de pesquisa que estamos acostumados, através do link https://www.arduino.cc/reference/en/language/functions/digital-io/digitalwrite/ encontramos as seguintes informações:

Reference > Language > Functions > Digital io > Digitalwrite

digitalWrite()

[Digital I/O]

Descrição

Aciona um valor `HIGH` ou `LOW` em um pino digital.

Se o pino for configurado como saída (OUTPUT) com a função `pinMode()`, sua tensão será acionada para o valor correspondente: 5V (ou 3.3V em placas alimentadas com 3.3V como o DUE) para o valor `HIGH`, 0V (ou ground) para `LOW`.

Se o pino for configurado como entrada (INPUT), a função `digitalWrite()` irá ativar (HIGH) ou desativar (LOW) o resistor interno de pull-up no pino de entrada. É recomendado configurar pinMode() com INPUT_PULLUP para ativar o resistor interno de pull-up. Veja o tutorial sobre pinos digitais para mais informações.

Se você não configurar o pino com `pinMode()` e `OUTPUT`, e conectar um LED ao pino, quando chamar `digitalWrite(HIGH)`, o LED pode aparecer um pouco apagado. Sem configurar explicitamente `pinMode()`, `digitalWrite()` irá apenas ativar o resistor de pull-up interno, que age como um grande resistor limitador de corrente.

Como pode ser visto acima, a função digitalWrite (pino , nível digital); terá como função colocar o pino selecionado como nível HIGH (nível 1) nível alto, ou nível LOW (nível 0) nível baixo. O nível HIGH no arduino tem o mesmo valor de VCC, e nível LOW de GND, respectivamente 5V e 0V.

Para o digitalWrite funcionar com sucesso e enviar aos pinos os níveis 0 e 1 corretamente, dentro do setup (), a função pinMode () deverá configurar o pino que está sendo manipulado como saída (OUTPUT), caso contrário você poderá não ter resultados satisfatórios no controle dos dispositivos de saída como LEDs, Buzzers e etc.

O que é o delay()

A função delay força uma parada no processamento do programa, ou seja, durante um período de tempo o programa ficará pausado sem executar nenhuma tarefa, após completar esse tempo o programa segue sua execução normalmente.

A função delay recebe o parâmetro de tempo em milissegundos.

```
delay (tempo em milisegundos);

void loop ()
{
  digitalWrite (13, HIGH);
  delay (1000);
  digitalWrite (13, LOW);
  delay (1000);
}
```

Veja abaixo uma pesquisa no site da própria IDE do arduino sobre a função delay.

https://www.arduino.cc/reference/en/language/functions/time/delay/

Reference > Language > Functions > Time > Delay

delay()

[Time]

Descrição

Pausa o programa por uma quantidade especificada de tempo (em milissegundos). Cada segundo equivale a 1000 milissegundos.

Sintaxe

`delay(ms)`

Parâmetros

`ms`: o número de milissegundos para pausar o programa (`unsigned long`)

Retorna

Nada

A partir da pesquisa é possível concluir que a função delay () tem como objetivo pausar a execução das instruções dentro do loop por um tempo determinado em milissegundos.

delay (1)	// aguarda 1 ms para continuar o tratamento do programa.
delay (150)	// aguarda 150 ms para continuar o tratamento do programa.
delay (1500)	// aguarda 1,5 s para continuar o tratamento do programa.
delay (10000)	// aguarda 10 s para continuar o tratamento do programa.
delay (400)	// aguarda 0,4 s para continuar o tratamento do programa

1000 mili segundos correspondem a 1 segundo

Com todo o conhecimento obtido na imersão, agora você já conseguirá entender 100% de tudo que está acontecendo dentro do programa Blink, linha por linha. Vamos lá:

```
25 // the setup function runs once when you press reset or power the board
26 void setup() {
27   // initialize digital pin LED_BUILTIN as an output.
28   pinMode(LED_BUILTIN, OUTPUT);
29 }
30
31 // the loop function runs over and over again forever
32 void loop() {
33   digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
34   delay(1000);                       // wait for a second
35   digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
36   delay(1000);                       // wait for a second
37 }
```

Linha 26: Abertura da função void setup () {

Linha 27: Apenas barra dupla para que sejam inseridos comentários //

Linha 28: Utilização da função pinMode para configurar o pino 13 como saída

Linha 29: Fechamento de chave e encerramento das instruções da função void setup ()

Linha 30: Apenas espaçamento

Linha 31: Apenas barra dupla para que sejam inseridos comentários //

Linha 32: Abertura da função void loop () {

Linha 33: Envia o nível 1 (HIGH) (5V) para o pino 13 (Ligando o LED do Shield para Arduino)

Linha 34: Pausa a instrução por 1000 ms, que equivalem a 1s

Linha 35: Envia o nível 0 (LOW) (0V) para o pino 13 (Desligando o LED do Shield para Arduino)

Linha 36: Pausa a instrução por 1000 ms, que equivalem a 1s

Linha 37: Fechamento de chave e encerramento as instruções da função void loop ()

Resumindo, o que acontece na prática é o LED ligar durante 1 segundo e desligar por 1 segundo, gerando o efeito de pisca-pisca.

Encerramos aqui nossa fase 2 da metodologia ERDINGER onde, com muita dedicação, disposição, força de vontade, pudemos imergir dentro de diversas pesquisas sobre tudo que encontramos dentro de um simples exemplo de um código para piscar um LED.

Seja sincero, você imagina que esse simples código podia conter tanto conhecimento? Imaginava que poderíamos aprender tanto? Isso é o poder do verdadeiro estudo. Vamos pra cima que vem muito mais conhecimento por aí. #tamojunto

Método ERDINGER

Fase 3: Desafio Individual

CURSO DE

ELETRÔNICA FÁCIL

Meus queridos amigos e alunos do curso de eletrônica digital e arduino para iniciantes, quanta coisa bacana já aprendemos no módulo 1 *Introdução ao fanstástico mundo da eletrônica digital*, no módulo 2 sobre *circuitos digitais combinacionais* e agora no módulo 3, o incrível *método ERDINGER – Primeiros passos com o arduino uno*.

Gradativamente você está evoluindo, crescendo e desbravando o mundo da eletrônica digital e programação de microcontroladores, e neste momento, chegamos ao um ponto **crucial** do método ERDINGER que é o o desafio individual.

O Desafio individual não tem obrigatoriedade de execução, porém se deixar de lado e não o fizer, as chances de frustação crescem muito, tristeza, desmotivação, principalmente quando chegarmos na fase 4 que é o desafio do mestre.

No desafio do mestre, eu (professor Rodolpho) irei dar pelos menos 2 desafios para você provar e testar seus conhecimentos sobre a fase 1 e 2. Poderiamos dizer que a fase 3 é o treino e a fase 4 é a final de campeonato, se você não treinar você pode até jogar a final de campeonato, porém as chances de derrota e de perder o título é gigantesca, afinal, no jogo você aplica o que fez no treinamento, não é verdade?

Então dica do professor Rodolpho Chrispim é que tome muito cuidado com o mosquitinho da auto-confiança que fica no seu ouvido assim: *zzZZzzzZzzZzzZz você já sabe de tudo; ZZzZzz você pode pular esta etapa; zzZZzzZzz você não precisa ficar fazendo estes 3 desafios ou 5 desafios individuais pois não vai agregar em nada; ZZzzzZZzzzZZzzz você é o maior canguru eletrônico do mundo e não precisa ficar escutando este professor doidão; (risos)*

Brincadeiras a parte, não dê ouvidos a este mosquitinho, já deixa a hashtag #mosquitoDoMal #EuSigoErdinger no grupo vip de alunos do Telegram para sabermos que você não vai dar ouvidos ao mosquitinho danado, e sim, fará de tudo para aprender da forma correta sobre o lindo mundo da eletrônica junto com a programação de microcontroladores.

FASE 3 – DESAFIO INDIVIDUAL

Vamos agora iniciar o passo a passo para desenvolver a fase 3 da nossa metodologia ERDINGER – desafio individual.

Passo 1: Acessar o código exemplo utilizado na Fase 1 e ***salvar como*** na pasta **Desafio individual 1** com o nome da atividade que sera desenvolvida, veja um exemplo:

Figura 72 - Com o arquivo exemplo aberto, ir em SALVAR COMO para salvá-lo na pasta desafio individual

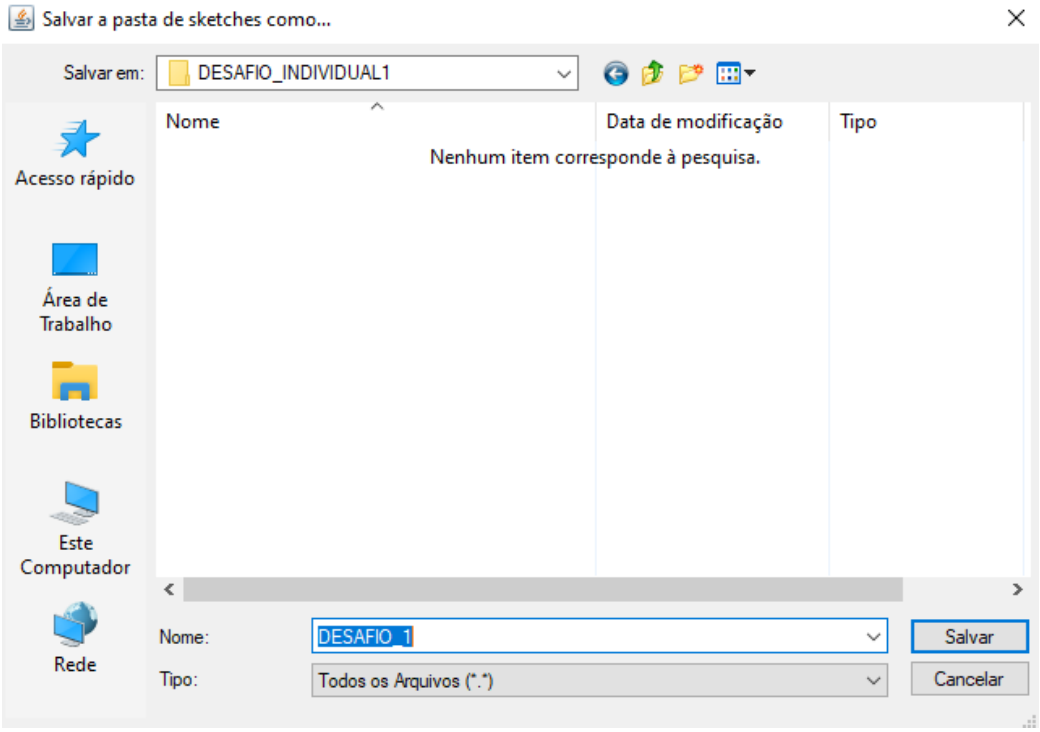
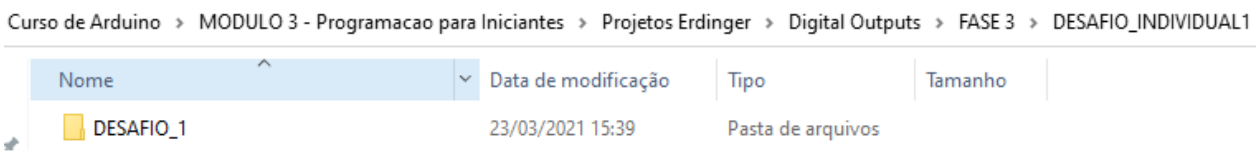
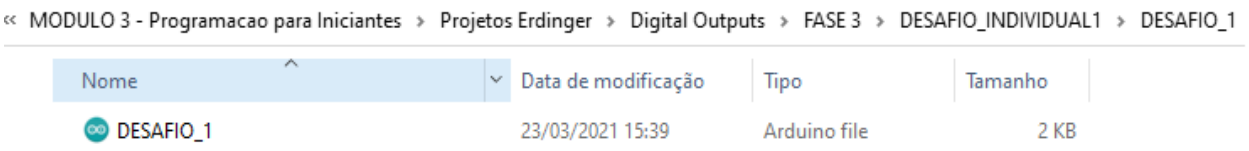


Figura 73 - Nomear como DESAFIO_1

Após salvar o projeto com o nome **DESAFIO_1**, automaticamente o *Arduino IDE* cria uma pasta com o nome do projeto e também um arquivo executavel da IDE Arduino para você abrír quando quiser ver o código. Veja abaixo:





Pronto, tudo salvo e organizado, agora podemos clicar duas vezes sobre o arquivo para abri-lo e começar as alterações necessárias para que o desafio individual seja iniciado. Vamos nessa?

Ao clicar duas vezes sobre o arquivo, irá aparecer um código da seguinte forma:

```
25 // the setup function runs once when you press reset or power the board
26 void setup() {
27   // initialize digital pin LED_BUILTIN as an output.
28   pinMode(LED_BUILTIN, OUTPUT);
29 }
30
31 // the loop function runs over and over again forever
32 void loop() {
33   digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
34   delay(1000);                       // wait for a second
35   digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
36   delay(1000);                       // wait for a second
37 }
```

Perceba que o código exemplo aberto e demonstrado na imagem acima está um pouco poluído de comentários. Vamos limpar esses comentários deixando apenas o código executável, promovendo mais legibilidade nesse primeiro momento de manutenção do código.

Veja como ficará o código depois de limpo:

Figura 74 - Código limpo sem comentários, com chaves alinhadas

Ao limpar o código, pode acontecer de algum algarismo importante para a execução do código ter sido removido junto com a limpeza, e isso poderá gerar erros na hora de fazer a compilação, por isso, após fazer a limpeza dos comentários vamos executar o programa para ver se algum erro é identificado.

Se ao executar o código e a mensagem *COMPILAÇÃO TERMINADA* aparecer, significa que nenhum erro foi encontrado e o código foi executado com sucesso.

Podemos então seguir com nosso trabalho tranquilamente pois tudo está acontecendo como planejado.

Sabendo que essa fase da metodologia compreende o chamado DESAFIO INDIVIDUAL, precisamos antes de mais nada entender que os desafios dependem de um ponto de vista, de uma perspectiva de quem executa tal ação, por isso algumas atividades podem ser grandes desafios para uns, e quase nada para outros, mas em qualquer um dos casos **nunca subestime o simples**, essa é a chave para que um iniciante se torne um excelente profissional.

Vamos considerar que o desafio que será apresentado aqui se aplica ao aluno que está começando do absoluto zero, ou seja, esse é o primeiro contato com o mundo da programação e da eletrônica digital, dessa forma qualquer atividade relacionada com esse assunto será um grande desafio.

Se você já é um iniciado nessa área, parabéns, você poderá compartilhar suas experiências com os demais colegas e principalmente poderá ajuda-los nos desafios iniciais, motivando, compartilhando ideias, e principalmente aplaudindo de pé cada aprendizado conquistado durantes seus estudos. Dito isto, vamos começar?

DESAFIO INDIVIDUAL Nº1

DESAFIO Nº1 – desafio individual

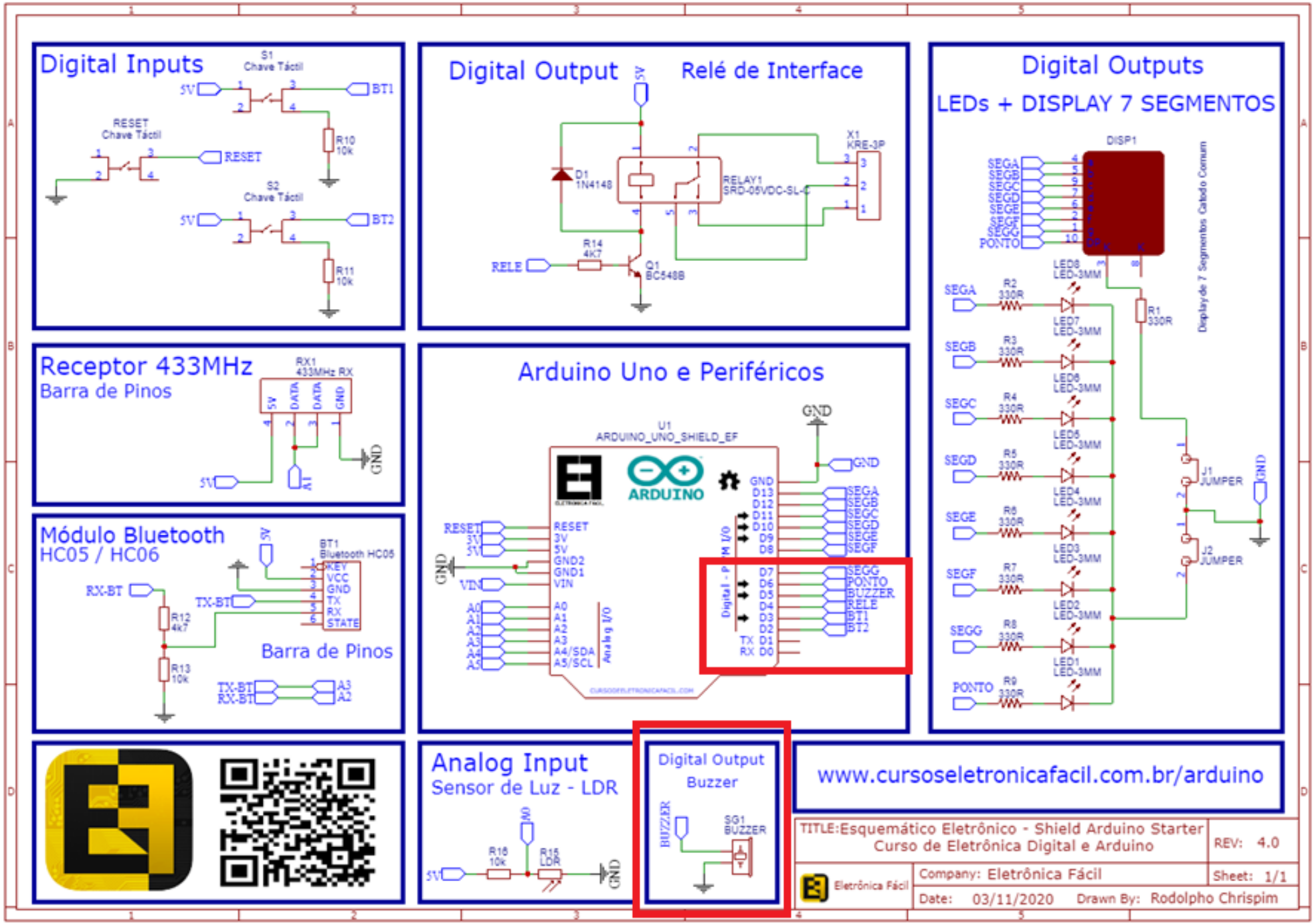
Ao invés de controlar o LED que está sendo manipulado pelo pino 13 do Arduino Uno, controlar o buzzer do Shield para Arduino Starter do Eletrônica Fácil de forma a ele ficar ligado por 0,5 segundos e desligado por 2,5 segundos de forma contínua.

INSTRUÇÕES

- ✓ Controlar o buzzer sonoro (Shield para arduino EF)
- ✓ Tempo que o buzzer ficará ligado: 0,5s
- ✓ Tempo que o buzzer ficará desligado: 2,5s
- ✓ Execução do código: infinita, até que o arduino seja desligado.

APLICAÇÕES

- ✓ Alarme sonoro de tempo esgotado em um forno elétrico
- ✓ Alarme sonoro de micro-ondas indicando fim de processo de aquecimento
- ✓ Indicação de marcha ré em veículos motorizados
- ✓ Alarme antifurto
- ✓ Sinal sonoro indicativo de saída de veículos em garagens.


Digital Inputs
S1
Chave Tátil
5V
BT1
R10
10k
RESET
Chave Tátil
RESET
S2
Chave Tátil
5V
BT2
R11
10k
Digital Output
Relé de Interface
5V
D1
1N4148
RELAY1
SRD-05VDC-SL-C
X1
KRE-3P
R14
4K7
RELE
Q1
BC548B
Digital Outputs
LEDs + DISPLAY 7 SEGMENTOS
DISP1
SEGA
SEGB
SEGC
SEGD
SEGE
SEGF
SEGG
PONTO
Display de 7 Segmentos Catodo Comum
LED8
LED-3MM
R1
330R
R2
330R
LED7
LED-3MM
R3
330R
LED6
LED-3MM
R4
330R
LED5
LED-3MM
R5
330R
LED4
LED-3MM
R6
330R
LED3
LED-3MM
R7
330R
LED2
LED-3MM
R8
330R
LED1
LED-3MM
R9
330R
J1
JUMPER
J2
JUMPER
GND
Receptor 433MHz
Barra de Pinos
RX1
433MHz RX
5V
DATA
DATA
GND
Arduino Uno e Periféricos
U1
ARDUINO_UNO_SHIELD_EF
ARDUINO
RESET
3V
5V
GND2
GND1
VIN
A0
A1
A2
A3
A4/SDA
A5/SCL
Analog I/O
D13
D12
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
TX D1
RX D0
Digital I - P
M I/O
Módulo Bluetooth
HC05 / HC06
BT1
Bluetooth HC05
KEY
VCC
GND
TX
RX
STATE
RX-BT
TX-BT
R12
4k7
R13
10k
Barra de Pinos
TX-BT
RX-BT
A3
A2
Analog Input
Sensor de Luz - LDR
A0
R15
10k
R16
LDR
Digital Output
Buzzer
BUZZER
SG1
BUZZER
www.cursoseletronicafacil.com.br/arduino
TITLE:Esquemático Eletrônico - Shield Arduino Starter
Curso de Eletrônica Digital e Arduino
REV: 4.0
Eletrônica Fácil
Company: Eletrônica Fácil
Sheet: 1/1
Date: 03/11/2020
Drawn By: Rodolpho Chrispim

BLOCO DE NOTAS - ANOTAÇÕES PARA EXECUTAR PROJETO

Pino do Arduino que controla o Buzzer: ***PINO 5***

Configuração do pino como: ***SAÍDA***

Função que faz configuração do pino no código: ***pinMode (PINO, in ou out)***

Função em que a pinMode estará inserida: ***void setup ()***

Função para controlar o tempo de execução do Buzzer: ***delay (TEMPO EM MILI SEG)***

Função para ligar e desligar o PINO 5: ***digitalWrite (PINO, NÍVEL DIGITAL)***

Função em que o código deverá ser escrito: ***void loop ()***

Tudo anotado e pronto para iniciarmos nosso projeto na IDE do arduino.

Uma boa prática é sempre começar um código com um cabeçalho, trazendo informações como autor, data e objetivo do projeto.

Existe duas formas de fazer um comentário em um projeto sem que os caracteres escritos interfiram na compilação.

***/** = Abertura de comentário.*

Todas as linhas que estiverem entre esses dois símbolos não serão interpretadas pelo compilador, servirão apenas como comentários.

****/** = fechamento de comentário*

```
/*
   ...comentário
   ...comentário
   ...comentário
*/
```

// = comentário de uma única linha (não necessita fechar)

```
// comentário de uma única linha
```

Veja então exemplo de cabeçalho para o código do desafio individual e outros projetos.

```
/*
        Desafio nº
        Metodologia de estudo:
        Curso:
        Nome do autor:
        Data:
        Objetivo:
*/
```

Veja na prática a utilização do cabeçalho:

```
/*
 Desafio 1 - Fase 3 - BUZZER
 Metodologia de estudo: ERDINGER
 Curso: Eletrônica Digital e Arduino para Iniciantes
 Nome do autor: Prof. Rodolpho
 Data:03/04/2021
 Objetivo: Testar as saídas digitais do arduino uno
*/
```

Vamos iniciar o código do nosso desafio?

Primeira etapa é configurar o pino 5 como saída, e para isso utilizaremos a função pinMode () dentro da função void setup (), que é a função utilizada para fazer as configurações.

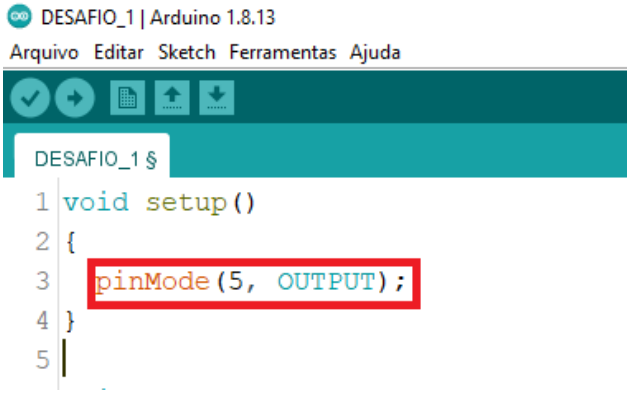
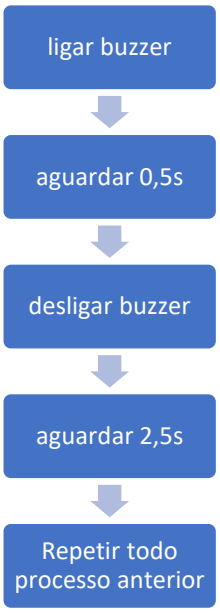


Figura 75 - pino 5 configurado como saída

Agora vamos entender como será a lógica de funcionamento, de forma sequencial, do código que iremos desenvolver.



Para ligar o buzzer será preciso mandar nível alto para o pino 5. Para isso utilizaremos a função que “escreve” níveis lógicos em saídas digitais, digitalWrite ().

```
digitalWrite(5, HIGH);
```

Para manter o buzzer ativado vamos pausar a execução do programa por um determinado tempo (0,5s), para isso utilizaremos a função delay (*tempo em milissegundos*).

```
delay(500);
```

Para desligar o buzzer será preciso mandar nível baixo para o pino 5. Para isso utilizaremos a função que escreve níveis lógicos em saídas digitais, digitalWrite ().

```
digitalWrite(5, LOW);
```

Para manter o buzzer desligado vamos pausar a execução do programa por um determinado tempo (2,5s), para isso utilizaremos a função delay (*tempo em milissegundos*).

```
delay(2500);
```

Para repetir todo o processo novamente, utilizaremos a função void loop (), ou seja, todo esse código será colocado dentro dessa função de repetição para que ele fique se repetindo infinitamente.

```
void loop()
{
  digitalWrite(5, HIGH);
  delay(500);
  digitalWrite(5, LOW);
  delay(2500);
}
```

Prontinho, colocamos tudo em linguagem de programação que poderá ser interpretada pelo compilador e então ser transferida para nosso arduino uno.

Vamos ver como ficou o código todo pronto?

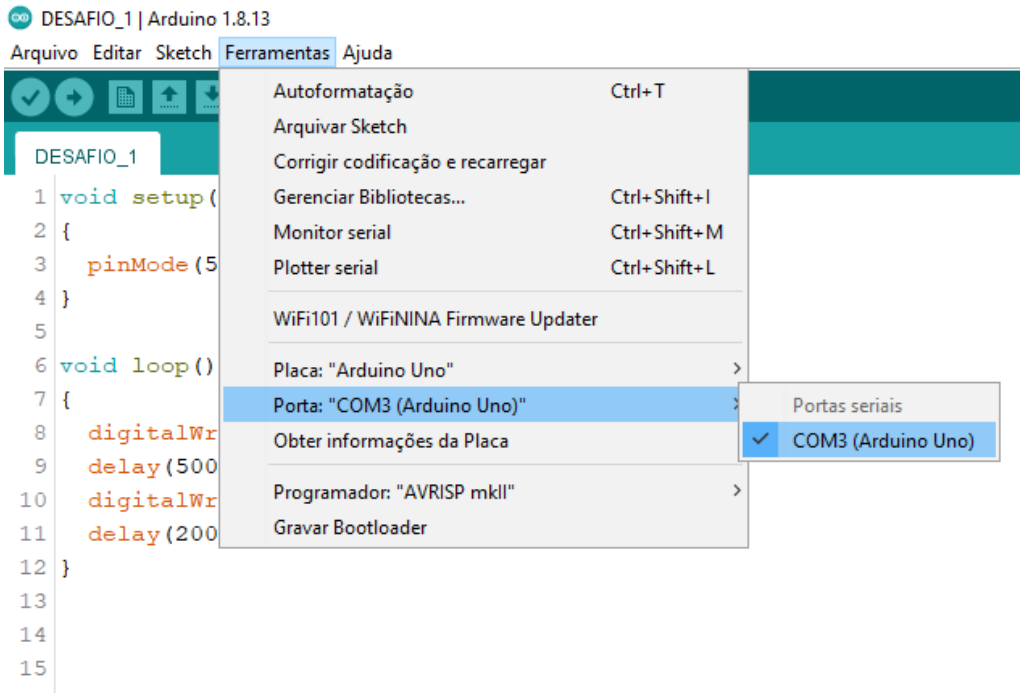
```
/*
 Desafio 1 - Fase 3 - BUZZER
 Metodologia de estudo: ERDINGER
 Curso: Eletrônica Digital e Arduino para Iniciantes
 Nome do autor: Prof. Rodolpho
 Data:03/04/2021
 Objetivo: Testar as saidas digitais do arduino uno
*/

void setup()                    // Chamada da função setup()
{                               // Chave de abertura da função setup()
  pinMode(5, OUTPUT);           // Chama a função pinMode() para configurar pino 5 como saida
}                               // Chave de fechamento da função setup()

void loop()                     // Chamada da função loop()
{                               // Chave de abertura do laço de repetição
  digitalWrite(5, HIGH);        // Enviar nivel lógico alto para o pino 5
  delay(500);                   // Aguardar 500ms (0,5s)
  digitalWrite(5, LOW);         // Enviar nivel lógico baixo para o pino 5
  delay(2500);                  // Aguardar 2500ms (2,5s)
}                               // Chave de fechamento do laço de repetição
```

Agora basta conectar o arduino na porta USB, compilar o programa e transferir o código para a placa.

ATENÇÃO – caso o erro abaixo seja apresentado ao compilar o código, significa que nenhuma porta serial foi selecionada para que a gravação do arduino seja feita com sucesso.

```
Serial port not selected.
O sketch usa 932 bytes (2%) de espaço de armazenamento para programas. O máximo são 32256 bytes.
Variáveis globais usam 9 bytes (0%) de memória dinâmica, deixando 2039 bytes para variáveis locais. O máximo são 2048 bytes.
Serial port not selected.
```

Para selecionar a porta serial basta seguir o caminho mostrado pela imagem abaixo.



Veja o resultado final desse código escaneando o QR code abaixo, ou acessando o link http://bit.ly/DESAFIO1_EDGF3.

DESAFIO INDIVIDUAL Nº2

Vamos agora para o próximo desafio, que terá um pequeno avanço no código feito no desafio anterior, tudo muito tranquilo. Vamos nessa?

DESAFIO Nº2 – desafio individual

Visando combinar o alarme sonoro com um alarme visual, agora você aluno do curso de eletrônica digital e arduino para iniciantes deverá alternar a ativação do alarme sonoro desenvolvido no desafio 1 com um alarme visual, onde ao ligar o buzzer, o led do alarme visual será desligado e ao ligar o led, o buzzer será desligado.

INSTRUÇÕES

- ✓ Controlar o buzzer sonoro e o LED
- ✓ Tempo que o buzzer ficará ligado: 0,5s
- ✓ Tempo que o LED ficará desligado: 0,5s
- ✓ Tempo que o LED ficará ligado: 2s
- ✓ Tempo que o buzzer ficará desligado: 2s
- ✓ Execução do código: infinita, até que o arduino seja desligado.

APLICAÇÕES

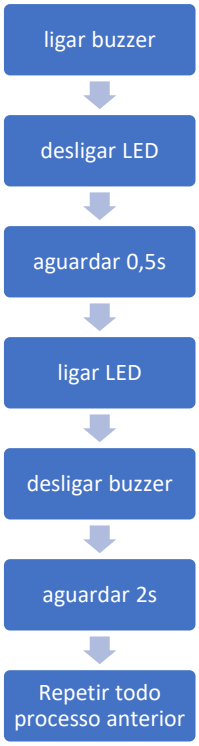
- ✓ Alarme sonoro e visual de indicação de farol ligado ao desligar o veiculo
- ✓ Alarme sonoro e visual indicando porta de geladeira aberta
- ✓ Indicação de movimento em veículos motorizados de carga em supermercados
- ✓ Alarme antifurto
- ✓ Sinal sonoro indicativo de saída de veículos em garagens.

Você estudante poderá escolher qualquer LED do shield para controlar, ou seja, fique a vontade para escolher o número do pino do Arduino que irá ligar o alarme visual e também a cor do LED. Vamos escolher o pino 10 para controlar o led amarelo 5 do shield.

BLOCO DE NOTAS - ANOTAÇÕES PARA EXECUTAR PROJETO

Pino do Arduino que controla o Buzzer: ***PINO 5***
Pino do Arduino que controla o LED escolhido: ***PINO 10 (led amarelo do shied)***
Configuração do pino 5 como: ***SAÍDA***
Configuração do pino 10 como: ***SAÍDA***
Função que faz configuração do pino no código: ***pinMode (PINO, in ou out)***
Função em que a pinMode estará inserida: ***void setup ()***
Função para controlar o tempo de execução do Buzzer e do LED: ***delay (TEMPO EM MILI SEG)***
Função para ativar e desativar o PINO 5 e 10: ***digitalWrite (PINO, NÍVEL DIGITAL)***
Função em que o código deverá ser escrito: ***void loop ()***

Depois de estruturarmos as anotações do que faremos, vamos organizar em diagrama de blocos a lógica do programa que escreveremos.



Agora ficou muito fácil, já construímos a sequência lógica do código, basta converter as instruções para a linguagem de programação.

Primeiro vamos configurar os pinos 5 e 10 como saída dentro da função void setup ().

```
void setup()
{
  pinMode(5, OUTPUT);
  pinMode(10, OUTPUT);
}
```

Agora devemos transcrever o diagrama de blocos para a linguagem de programação dentro do laço de repetição, assim como explicamos no desafio número 1.

```
void loop()
{
  digitalWrite(5, HIGH);
  digitalWrite(10, LOW);
  delay(500);
  digitalWrite(10, HIGH);
  digitalWrite(5, LOW);
  delay(2000);
}
```

Lembrando que todas as partes do nosso código dever comentada, buscando o pleno entendimento e principalmente deixando evidente cada etapa da lógica de programação para que futuras manutenções no código sejam feitas com maior facilidade.

Detalhe: Um bom programador também é reconhecido pela organização do seu código, identação e comentários de tudo que produz , e lembre-se, por enquanto estamos falando de 15 linhas de programação, imagine quando chegarmos a 1500 linhas de programação? Fazendo tudo certinho e documentado, você não está perdendo tempo, mas ganhando em orgnização e qualidade sempre!

Vamos agora ver como ficou nosso código todo comentado e organizado, pronto para ser gravado no arduino.

```
/*
 Desafio 2 - Fase 3 - BUZZER e LED
 Metodologia de estudo: ERDINGER
 Curso: Eletrônica Digital e Arduino para Iniciantes
 Nome do autor: Prof. Rodolpho
 Data:03/04/2021
 Objetivo: Testar as saídas digitais do arduino uno
*/

void setup()                        // Chamada da função setup()
{                                   // Chave de abertura da função setup()
  pinMode(5, OUTPUT);               // Chama a função pinMode() para configurar pino 5 como saída
  pinMode(10, OUTPUT);              // Chama a função pinMode() para configurar pino 10 como saída
}                                   // Chave de fechamento da função setup()

void loop()                         // Chamada da função loop()
{                                   // Chave de abertura do laço de repetição
  digitalWrite(5, HIGH);            // Enviar nível lógico alto para o pino 5
  digitalWrite(10, LOW);            // Enviar nível lógico baixo para o pino 10
  delay(500);                       // Aguardar 500ms (0,5s)
  digitalWrite(10, HIGH);           // Enviar nível lógico alto para o pino 10
  digitalWrite(5, LOW);             // Enviar nível lógico baixo para o pino 5
  delay(2000);                      // Aguardar 2000ms (2s)
}                                   // Chave de fechamento do laço de repetição
```

Agora basta gravar o código construído no seu arduino, verificar o funcionamento e comemorar mais uma vitória.

Veja o funcionamento do desafio nº2 escaneando o QR code abaixo ou pelo link http://bit.ly/EDGF3_DESAFIO2

Vamos lá, chegou a hora de pensarmos um pouco, mas antes disso, já podemos dizer que você é um vitorioso, mande aquela hashtag no grupo vip de alunos, *#Eusobrevivi #ERDINGER #DESAFIO2* para sabermos que chegou até este ponto do nosso E-BOOK, pois isso sempre motiva o professor e os demais alunos a continuarem lutando pela busca da faixa preta.

Pessoal, não sei se o que vou dizer aqui já aconteceu com vocês, mas comigo sim e com muitos alunos iniciantes que tive a oportunidade de trabalhar de forma presencial e a distância também, que é você aprender uma coisa **muito básica** porém super, mega, ultra legal, por exemplo o piscar de um led e/ou ligar e desligar um buzzer, atividades simples que já proporcionam aquela motivação para continuar no aprendizado, e a partir desse resultado você decide mostrar o seu sucesso e realização para alguém, e ao compartilhar sua felicidade você acaba ouvindo: “Nossa, mas só ta piscando aissss luizinhasssssssssssss? Ah, agora da para fazer um pisca pisca de natal né, mas não é mais fácil comprar na loja de 10 Reais? Ou então “Olocooooooooooo, achei que ia ver algo interessante funcionando e você me mostra só um LED ligando e desligando assim, me chame quando tiver algo interessante de verdade”

Na verdade o que eu quero dizer do fundo do coração é muita gente vai dar ducha de agua fria em vocês, e as vezes nem é por maldade, as vezes acabam saindo as palavras de forma automática em forma de brincadeira mas que podem magoar e até mesmo frustrar os estudantes, desmotivar e colocar para baixo, principalmente os que estão começando em sua rotina de aprendizagem.

Portanto, **NÃO DESANIMEM POR NADA NESTE MUNDO**, lembrem-se do que fez vocês iniciarem nesta luta e usem isso como motivação para continuarem lutando pelo objetivo e pelo sonho a ser realizado. Quem está de fora não entende o quão prazeroso é piscar um LED, ligar um Buzzer e desbravar o mundo da eletrônica e da programação, e principalmente o poder que esses simples projetos guardam de conhecimento para desenvolvimento de projetos futuros, por isso vamos juntos nessa que o caminho é longo e o sucesso é certo. Lembrem-se, juntos somos mais fortes e sempre chegamos mais longe.

DESAFIO INDIVIDUAL Nº3

Vamos agora avançar ainda mais nos desafios propostos, vamos iniciar o desafio individual Nº3.

Dessa vez vamos iniciar a atividade apresentando um vídeo do que precisará ser feito, mas atenção, foque apenas no funcionamento prático do dispositivo e não na maneira como foi instalado, técnicas de instalação ou coisas do gênero.

Para visualizar o vídeo, clique no link

http://bit.ly/DESAFIO3_ATIVIDADEhttp://bit.ly/DESAFIO3_ATIVIDADE ou escaneie com seu smartfone o QR code.

Como demonstrado no vídeo, o objetivo desta atividade será programar o Arduino Uno para realizar o mesmo funcionamento do dispositivo apresentado, para isso você deverá realizar um código de programação que controle os 8 Leds do Shield para Arduino para que apresentem o mesmo princípio de funcionamento dos dispositivos apresentados.

DESAFIO Nº3 – desafio individual

O funcionamento do dispositivo apresentado se iniciará com todos os leds desligados. Os leds serão ligados um por vez e permanecerão acesos até que todos tenham sido ativados, então o processo se repete novamente começando com todos os leds desligados. O primeiro led a ser ativado será o led1 verde, após 100ms o segundo led verde será ativado, após mais 100ms o terceiro led verde será ativado, e seguindo essa sequência com intervalos de 100ms todos os leds do shield deverão ser acionados em ordem sequencial, do primeiro verde até o ultimo vermelho, quando todos estiverem acesos o tempo de espera será de 800ms, então todos os leds deverão ser apagados aguardando mais 50ms. Então o processo se repete do inicio.

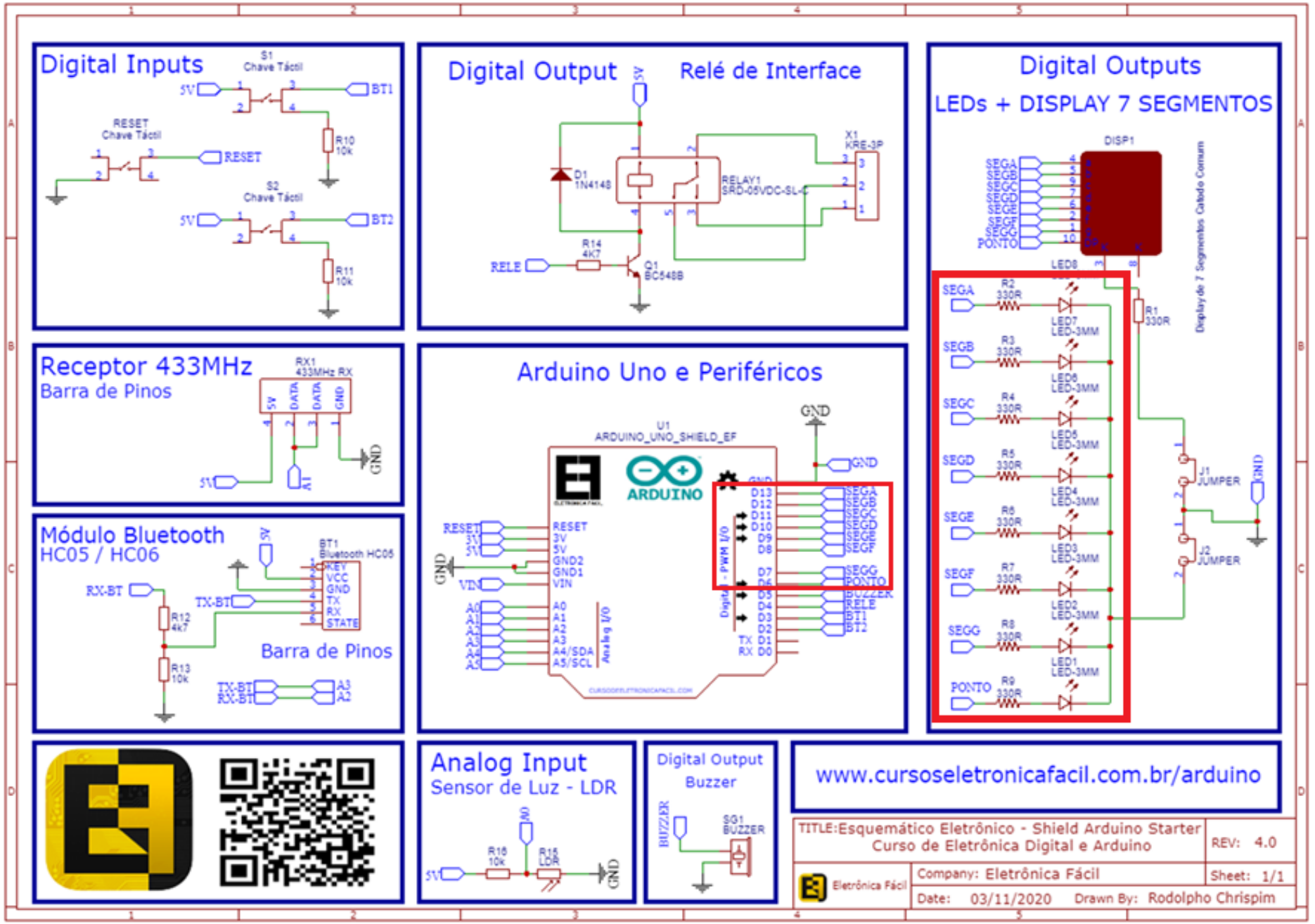
INSTRUÇÕES

- ✓ Controlar todos os oito LEDS do shield
- ✓ Estado inicial dos LEDs: desligados
- ✓ Ligar um led por vez
- ✓ Iniciar acionamento pelos leds verdes 1, 2 e 3, depois leds amarelos 1 e 2, finalizando com os leds vermelhos 1, 2 e 3.
- ✓ Intervalo entre cada acionamento de LED: 100ms
- ✓ Quando todos os leds estiverem acesos, aguardar 800ms.

- ✓ Após todos os leds terem sidos ativados, desligar todos os leds, aguardar 50ms e repetir todo o processo do início.

APLICAÇÕES

- ✓ Seta sinalizadora de conversões para diferentes veículos motorizados


Digital Inputs
S1
Chave Tátil
5V
BT1
RESET
Chave Tátil
RESET
R10
10k
S2
Chave Tátil
5V
BT2
R11
10k
Digital Output
Relé de Interface
5V
D1
1N4148
RELAY1
SRD-05VDC-SL-C
X1
KRE-3P
R14
4K7
RELE
Q1
BC548B
Digital Outputs
LEDs + DISPLAY 7 SEGMENTOS
DISP1
SEGA
SEGB
SEGC
SEGD
SEGE
SEGF
SEGG
PONTO
Display de 7 Segmentos Catodo Comum
LED8
R1
330R
R2
330R
LED7
LED-3MM
R3
330R
LED6
LED-3MM
R4
330R
LED5
LED-3MM
R5
330R
LED4
LED-3MM
R6
330R
LED3
LED-3MM
R7
330R
LED2
LED-3MM
R8
330R
LED1
LED-3MM
R9
330R
J1
JUMPER
J2
JUMPER
GND
Receptor 433MHz
Barra de Pinos
RX1
433MHz RX
5V
DATA
DATA
GND
AT
Arduino Uno e Periféricos
U1
ARDUINO_UNO_SHIELD_EF
ARDUINO
RESET
3V
5V
GND2
GND1
VIN
A0
A1
A2
A3
A4/SDA
A5/SCL
Analog I/O
Digital - PWM I/O
D13
D12
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
TX D1
RX D0
BUZZER
RELE
BT1
BT2
Módulo Bluetooth
HC05 / HC06
BT1
Bluetooth HC05
KEY
VCC
GND
TX
RX
STATE
RX-BT
TX-BT
R12
4k7
R13
10k
Barra de Pinos
TX-BT
RX-BT
A3
A2
Analog Input
Sensor de Luz - LDR
R16
10k
R15
LDR
Digital Output
Buzzer
BUZZER
SG1
BUZZER
www.cursoseletronicafacil.com.br/arduino
TITLE:Esquemático Eletrônico - Shield Arduino Starter
Curso de Eletrônica Digital e Arduino
REV: 4.0
Eletrônica Fácil
Company: Eletrônica Fácil
Sheet: 1/1
Date: 03/11/2020
Drawn By: Rodolpho Chrispim

BLOCO DE NOTAS - ANOTAÇÕES PARA EXECUTAR PROJETO

Pino do Arduino que controla o led1 verde: **6**

Pino do Arduino que controla o led2 verde: **7**

Pino do Arduino que controla o led3 verde: **8**

Pino do Arduino que controla o led1 amarelo: **9**

Pino do Arduino que controla o led2 amarelo: **10**

Pino do Arduino que controla o led1 vermelho: **11**

Pino do Arduino que controla o led2 vermelho: **12**

Pino do Arduino que controla o led3 vermelho: **13**

Configuração dos pinos 6 ao 13 como: ***SAÍDA***

Função que faz configuração dos pinos no código: ***pinMode (PINO, in ou out)***

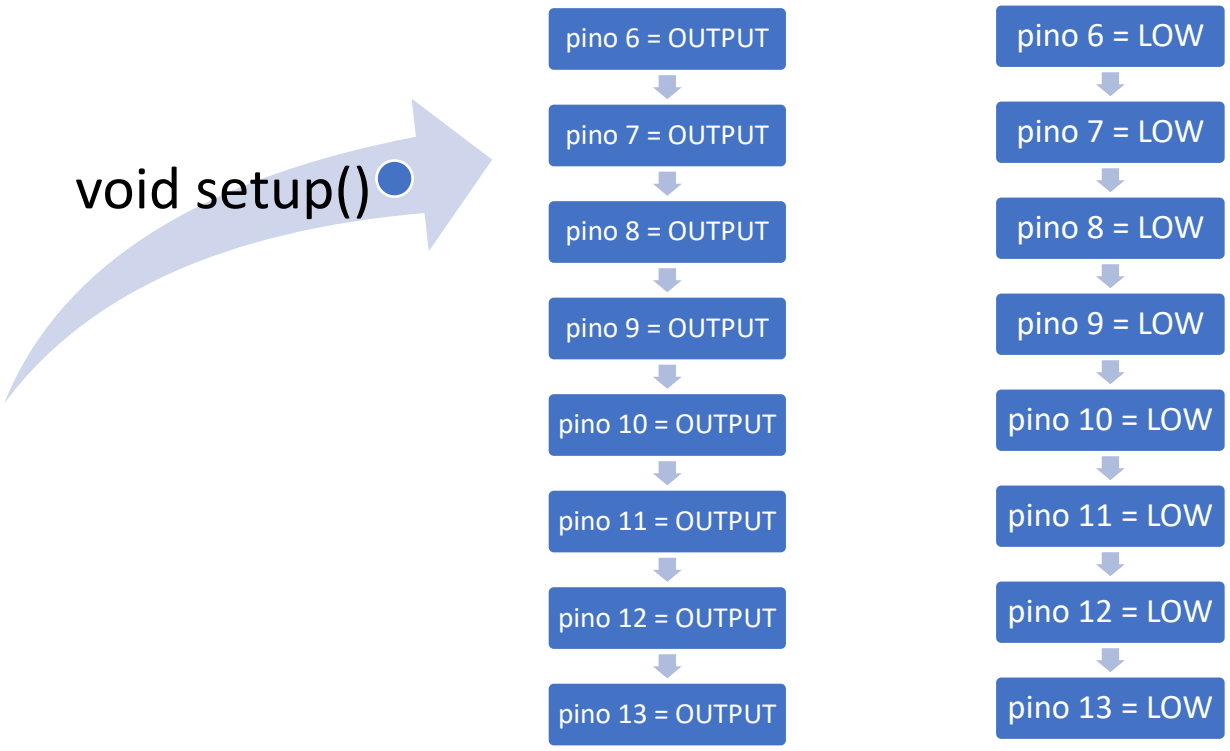
Função para controlar o tempo de execução do LED: ***delay (TEMPO EM MILI SEG)***

Função para ativar e desativar os leds: ***digitalWrite (PINO, NÍVEL DIGITAL)***

Função em que o código deverá ser escrito: ***void loop ()***

Agora vamos organizar em um diagrama de blocos a sequencia lógica do nosso programa.

A primeira etapa será, dentro da função void setup, configurar como output os pinos do arduino que irão controlar os leds, e garantir que todos os leds iniciem suas atividades desligados *(podemos iniciar o estado de um pino dentro da função setup antes de começarmos o código, isso garante que ele comece no estado desejado, high ou low)*

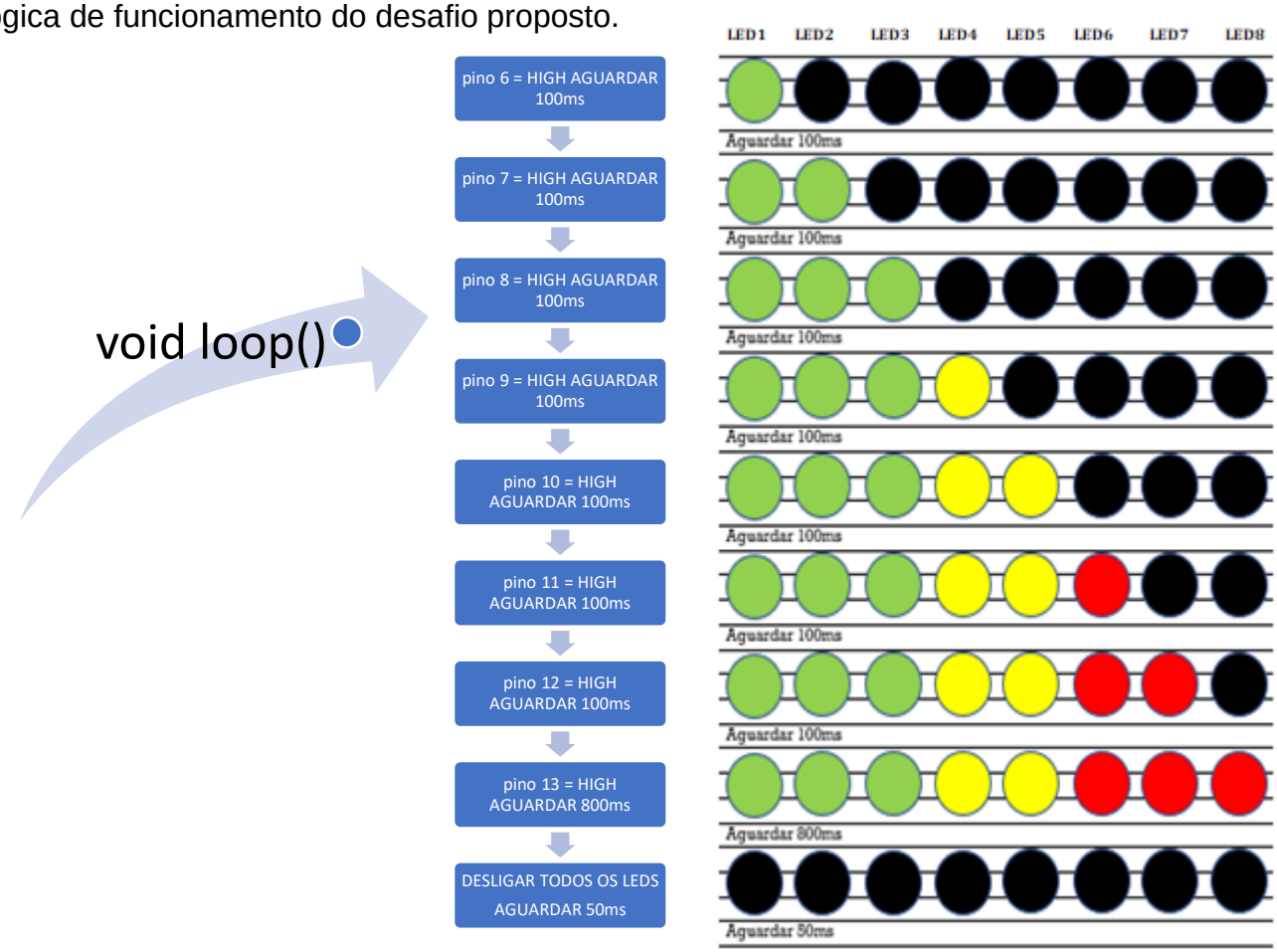


Convertendo para o código de programação a lógica descrita anteriormente.

```
void setup()                        // Chamada da função setup()
{                                   // Chave de abertura da função setup()
  pinMode(6, OUTPUT);               // Chama a função pinMode() para configurar pino 6 como saída
  pinMode(7, OUTPUT);               // Chama a função pinMode() para configurar pino 7 como saída
  pinMode(8, OUTPUT);               // Chama a função pinMode() para configurar pino 8 como saída
  pinMode(9, OUTPUT);               // Chama a função pinMode() para configurar pino 9 como saída
  pinMode(10, OUTPUT);              // Chama a função pinMode() para configurar pino 10 como saída
  pinMode(11, OUTPUT);              // Chama a função pinMode() para configurar pino 11 como saída
  pinMode(12, OUTPUT);              // Chama a função pinMode() para configurar pino 12 como saída
  pinMode(13, OUTPUT);              // Chama a função pinMode() para configurar pino 13 como saída

  digitalWrite(6, LOW);             // Enviar nível lógico baixo para o pino 6
  digitalWrite(7, LOW);             // Enviar nível lógico baixo para o pino 7
  digitalWrite(8, LOW);             // Enviar nível lógico baixo para o pino 8
  digitalWrite(9, LOW);             // Enviar nível lógico baixo para o pino 9
  digitalWrite(10, LOW);            // Enviar nível lógico baixo para o pino 10
  digitalWrite(11, LOW);            // Enviar nível lógico baixo para o pino 11
  digitalWrite(12, LOW);            // Enviar nível lógico baixo para o pino 12
  digitalWrite(13, LOW);            // Enviar nível lógico baixo para o pino 13
}                                   // Chave de fechamento da função setup()
```

Veja que, além de configurar os pinos do arduino como saída para que possam controlar os leds, também já garantimos por meio da função digitalWrite() que esses pinos iniciem seu funcionamento em nível baixo, ou seja, com os leds desligados.

Após configurar os pinos do arduino como saida podemos então inciar a construção da lógica de funcionamento do desafio proposto.



Agora é hora de transformar a lógica apresentada pelo diagrama de blocos em linguagem de programação.

```
void loop()                          // Chamada da função loop()
{                                    // Chave de abertura do laço de repetição
  digitalWrite(6, HIGH);             // Enviar nível lógico alto para o pino 6
  delay(100);                        // Aguardar 100ms (0,1s)
  digitalWrite(7, HIGH);             // Enviar nível lógico alto para o pino 7
  delay(100);                        // Aguardar 100ms (0,1s)
  digitalWrite(8, HIGH);             // Enviar nível lógico alto para o pino 8
  delay(100);                        // Aguardar 100ms (0,1s)
  digitalWrite(9, HIGH);             // Enviar nível lógico alto para o pino 9
  delay(100);                        // Aguardar 100ms (0,1s)
  digitalWrite(10, HIGH);            // Enviar nível lógico alto para o pino 10
  delay(100);                        // Aguardar 100ms (0,1s)
  digitalWrite(11, HIGH);            // Enviar nível lógico alto para o pino 11
  delay(100);                        // Aguardar 100ms (0,1s)
  digitalWrite(12, HIGH);            // Enviar nível lógico alto para o pino 12
  delay(100);                        // Aguardar 100ms (0,1s)
  digitalWrite(13, HIGH);            // Enviar nível lógico alto para o pino 13
  delay(800);                        // Aguardar 100ms (0,8s)

  digitalWrite(6, LOW);              // Enviar nível lógico baixo para o pino 6
  digitalWrite(7, LOW);              // Enviar nível lógico baixo para o pino 7
  digitalWrite(8, LOW);              // Enviar nível lógico baixo para o pino 8
  digitalWrite(9, LOW);              // Enviar nível lógico baixo para o pino 9
  digitalWrite(10, LOW);             // Enviar nível lógico baixo para o pino 10
  digitalWrite(11, LOW);             // Enviar nível lógico baixo para o pino 11
  digitalWrite(12, LOW);             // Enviar nível lógico baixo para o pino 12
  digitalWrite(13, LOW);             // Enviar nível lógico baixo para o pino 13
  delay(50);                         // Aguardar 100ms (0,05s)
}
```

Terminada a etapa de construção docódigo na linguagem de programação, agora é só compilar, fazer a gravação no arduino, e verificar o funcionamento do projeto.

Você pode verificar o funcionamento desse desafio por meio do QR code ao lado ou pelo link http://bit.ly/DESAFIO3_SEQUENCIAL

DESAFIO INDIVIDUAL Nº4

Como você tem percebido, um desafio é continuação do outro, podemos sempre aproveitar o codigo de um desafio para utilizar no outro, e não será diferente neste desafio 4.

Assistindo o video de demonstração da seta sinalizadora, você deve ter reparado que o dispositivo é dividido em três setas amarelas principais que acendem uma por vez.

O desafio agora será acender um conjunto de cores de leds por vez que representarão as setas. Simples assim. Vamos ver o desafio.

DESAFIO Nº4 – desafio individual

O funcionamento do dispositivo apresentado se iniciará com todos os leds desligados. Os leds serão ligados por grupos de cores, um grupo por vez, e permanecerão acesos até que todos tenham sido ativados, então o processo se repete novamente começando com todos os leds desligados. O primeiro grupo de leds a ser ativado será o da cor verde, após 100ms o segundo grupo de leds amarelos será ativado, após mais 100ms o terceiro grupo de leds vermelhos serão ativados, quando todos os grupos estiverem acesos o tempo de espera será de 500ms, então todos os leds deverão ser apagados aguardando mais 50ms. Então todo o processo se repete do inicio.

INSTRUÇÕES

- ✓ Controlar os grupos de cores de LEDS do shield
- ✓ Estado inicial dos LEDs: desligados
- ✓ Ligar um grupo de cores por vez
- ✓ Iniciar acionamento pelo grupo de leds verdes 1, 2 e 3, depois o grupo amarelo de leds 1 e 2, finalizando com o grupo de leds vermelhos 1, 2 e 3.
- ✓ Intervalo entre cada acionamento de LED: 100ms
- ✓ Quando todos os grupos estiverem acesos, aguardar 500ms.
- ✓ Desligar todos os grupos de leds e aguardar 50ms
- ✓ Repetir todo o processo novamente

APLICAÇÕES

- ✓ Seta sinalizadora de conversões para diferentes veículos motorizados

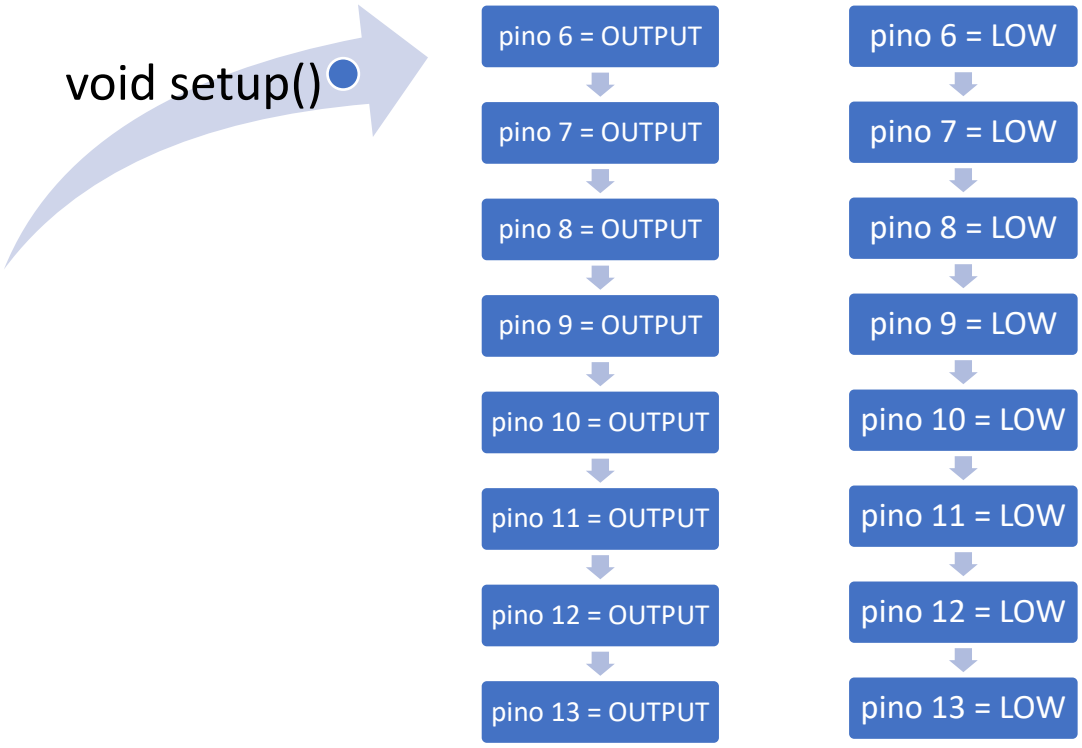
Como de costume vamos montar nosso bloco de anotações com informações que utilizaremos para montar nossa lógica de programação.

Já sabemos, a partir do descritivo, que o código será praticamente o mesmo, poderemos manter as mesmas configurações de saída, mudaremos apenas a lógica de acionamento dos leds, que agora serão por grupos de cores. Utilize o arquivo do desafio 3 para salvar o desafio 4, onde faremos a nova programação e alterações.

BLOCO DE NOTAS - ANOTAÇÕES PARA EXECUTAR PROJETO
Pino do Arduino que controla o led1 verde: **6**
Pino do Arduino que controla o led2 verde: **7**
Pino do Arduino que controla o led3 verde: **8**
Pino do Arduino que controla o led1 amarelo: **9**
Pino do Arduino que controla o led2 amarelo: **10**
Pino do Arduino que controla o led1 vermelho: **11**
Pino do Arduino que controla o led2 vermelho: **12**
Pino do Arduino que controla o led3 vermelho: **13**
Configuração dos pinos 6 ao 13 como: ***SAÍDA***
Função que faz configuração dos pinos no código: ***pinMode (PINO, in ou out)***
Função para controlar o tempo de execução do LED: ***delay (TEMPO EM MILI SEG)***
Função para ativar e desativar os leds: ***digitalWrite (PINO, NÍVEL DIGITAL)***
Função em que o código deverá ser escrito: ***void loop ()***

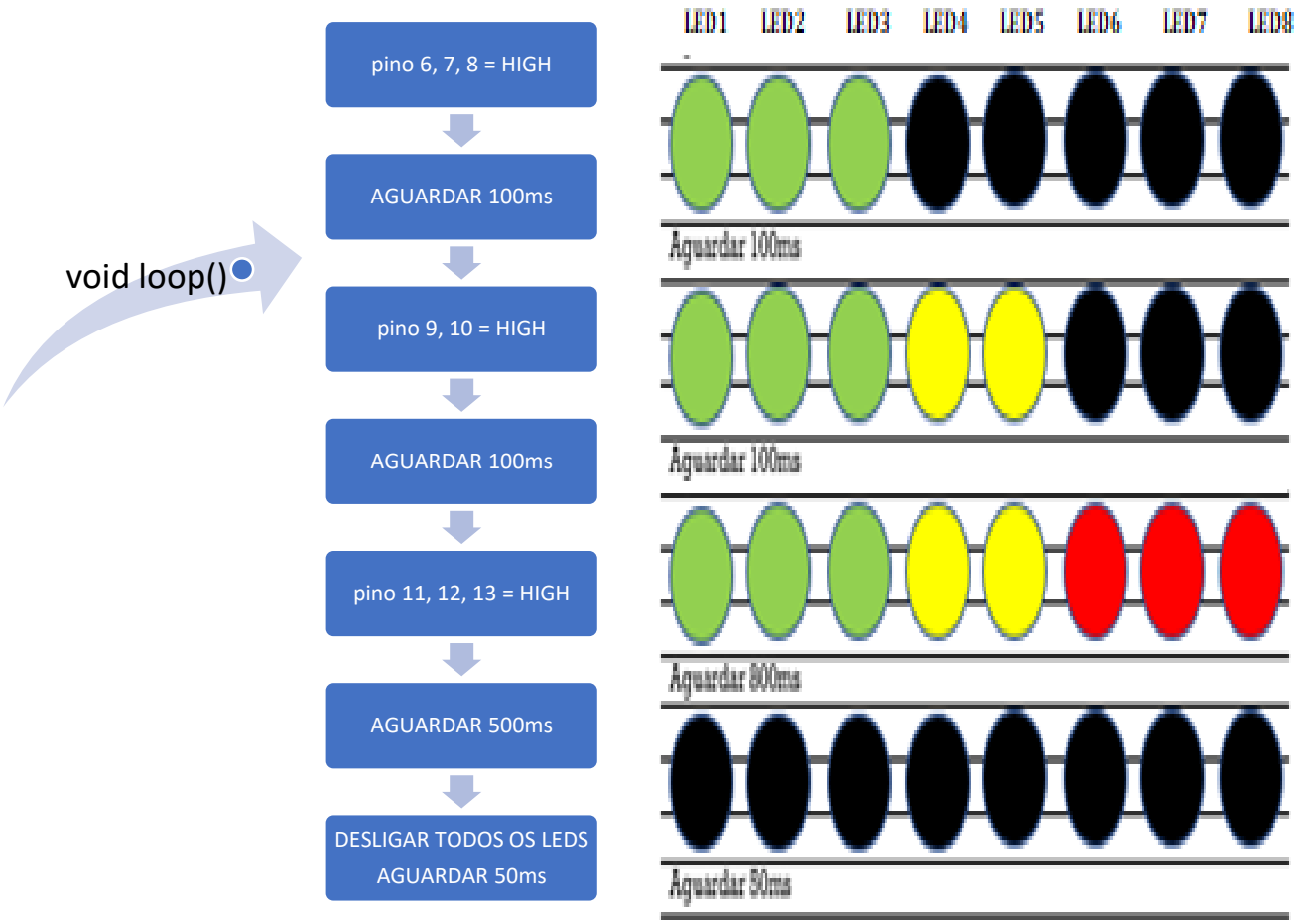
Manteremos as configurações iniciais da função void setup, já apresentadas anteriormente, pois se enquadram totalmente na ideia do desafio número 4, ou seja, todos os pinos que controlam os leds serão configurados como saída e todos os leds estarão sendo inicializados em nível 0, ou seja, desligados.

Convertendo a lógica apresentada acima em linguagem de programação, temos a configuração do setup da seguinte maneira:



```
void setup()                    // Chamada da função setup()
{                               // Chave de abertura da função setup()
  pinMode(6, OUTPUT);           // Chama a função pinMode() para configurar pino 6 como saída
  pinMode(7, OUTPUT);           // Chama a função pinMode() para configurar pino 7 como saída
  pinMode(8, OUTPUT);           // Chama a função pinMode() para configurar pino 8 como saída
  pinMode(9, OUTPUT);           // Chama a função pinMode() para configurar pino 9 como saída
  pinMode(10, OUTPUT);          // Chama a função pinMode() para configurar pino 10 como saída
  pinMode(11, OUTPUT);          // Chama a função pinMode() para configurar pino 11 como saída
  pinMode(12, OUTPUT);          // Chama a função pinMode() para configurar pino 12 como saída
  pinMode(13, OUTPUT);          // Chama a função pinMode() para configurar pino 13 como saída

  digitalWrite(6, LOW);         // Enviar nível lógico baixo para o pino 6
  digitalWrite(7, LOW);         // Enviar nível lógico baixo para o pino 7
  digitalWrite(8, LOW);         // Enviar nível lógico baixo para o pino 8
  digitalWrite(9, LOW);         // Enviar nível lógico baixo para o pino 9
  digitalWrite(10, LOW);        // Enviar nível lógico baixo para o pino 10
  digitalWrite(11, LOW);        // Enviar nível lógico baixo para o pino 11
  digitalWrite(12, LOW);        // Enviar nível lógico baixo para o pino 12
  digitalWrite(13, LOW);        // Enviar nível lógico baixo para o pino 13
}                               // Chave de fechamento da função setup()
```

Agora vamos estruturar o acionamento dos grupos de leds em diagramas de blocos.



Seguindo a nossa estruturação lógica pelo diagrama de blocos, vamos transcrever essa ideia estruturada para a linguagem de programação dentro da função void loop ().

Escrevendo o código de acionamento do primeiro grupo de leds da cor verde, controlados pelos pinos 6, 7,e 8, terminando com a espera de 100ms.

```
13 void loop()                 // Início da função de repetição
14 {                           // chave de abertura da função
15   digitalWrite(6, HIGH);    // Envia Nível Alto para o pino 6    --> LED1
16   digitalWrite(7, HIGH);    // Envia Nível Alto para o pino 7    --> LED2
17   digitalWrite(8, HIGH);    // Envia Nível Alto para o pino 8    --> LED3
18
19   delay(100);               // Aguarda 100ms
```

Acionamento do segundo grupo de leds, agora amarelos, controlados pelos pinos 9 e 10, terminando com a espera de 100ms.

```
21   digitalWrite(9, HIGH);    // Envia Nível Alto para o pino 9    --> LED4
22   digitalWrite(10, HIGH);   // Envia Nível Alto para o pino 10   --> LED5
23
24   delay(100);               // Aguarda 100ms
```

E por último o acionamento do grupo de leds vermelhos, controlados pelos pinos 11, 12, 13, terminando com espera de 500ms.

```
26   digitalWrite(11, HIGH); // Envia Nível Alto para o pino 11   --> LED6
27   digitalWrite(12, HIGH); // Envia Nível Alto para o pino 12   --> LED7
28   digitalWrite(13, HIGH); // Envia Nível Alto para o pino 13   --> LED8
29
30   delay(500);             // Aguarda 500ms
```

Após todos os grupos de leds terem sido ativados com seus respectivos tempos de espera, agora todos os grupos serão desligados, haverá um tempo de espera de 50ms e todo o processo se iniciará novamente. Veja:

```
32   digitalWrite(6, LOW);    // Envia Nível Baixo para o pino 6   --> LED1
33   digitalWrite(7, LOW);    // Envia Nível Baixo para o pino 7   --> LED2
34   digitalWrite(8, LOW);    // Envia Nível Baixo para o pino 8   --> LED3
35   digitalWrite(9, LOW);    // Envia Nível Baixo para o pino 9   --> LED4
36   digitalWrite(10, LOW);   // Envia Nível Baixo para o pino 10  --> LED5
37   digitalWrite(11, LOW);   // Envia Nível Baixo para o pino 11  --> LED6
38   digitalWrite(12, LOW);   // Envia Nível Baixo para o pino 12  --> LED7
39   digitalWrite(13, LOW);   // Envia Nível Baixo para o pino 13  --> LED8
40   delay(50);               // Aguarda 50ms
41 }                          // chave de fechamento da função
```

Veja abaixo o código completo e quanta coisa já aprendemos nessa caminhada. É simplesmente incrível.

```
/*
 Desafio 4 - Fase 3 - GRUPO DE LEDS COMPONDO SETA
 Metodologia de estudo: ERDINGER
 Curso: Eletrônica Digital e Arduino para Iniciantes
 Nome do autor: Prof. Rodolpho
 Data:03/04/2021
 Objetivo: Testar as saídas digitais do arduino uno
*/

void setup()                        // Chamada da função setup()
{                                   // Chave de abertura da função setup()
  pinMode(6, OUTPUT);               // Chama a função pinMode() para configurar pino 6 como saída
  pinMode(7, OUTPUT);               // Chama a função pinMode() para configurar pino 7 como saída
  pinMode(8, OUTPUT);               // Chama a função pinMode() para configurar pino 8 como saída
  pinMode(9, OUTPUT);               // Chama a função pinMode() para configurar pino 9 como saída
  pinMode(10, OUTPUT);              // Chama a função pinMode() para configurar pino 10 como saída
  pinMode(11, OUTPUT);              // Chama a função pinMode() para configurar pino 11 como saída
  pinMode(12, OUTPUT);              // Chama a função pinMode() para configurar pino 12 como saída
  pinMode(13, OUTPUT);              // Chama a função pinMode() para configurar pino 13 como saída

  digitalWrite(6, LOW);             // Enviar nível lógico baixo para o pino 6
  digitalWrite(7, LOW);             // Enviar nível lógico baixo para o pino 7
  digitalWrite(8, LOW);             // Enviar nível lógico baixo para o pino 8
  digitalWrite(9, LOW);             // Enviar nível lógico baixo para o pino 9
  digitalWrite(10, LOW);            // Enviar nível lógico baixo para o pino 10
  digitalWrite(11, LOW);            // Enviar nível lógico baixo para o pino 11
  digitalWrite(12, LOW);            // Enviar nível lógico baixo para o pino 12
  digitalWrite(13, LOW);            // Enviar nível lógico baixo para o pino 13
}                                   // Chave de fechamento da função setup()


void loop()                         // Chamada da função loop()
{                                   // Chave de abertura do laço de repetição
  digitalWrite(6, HIGH);            // Enviar nível lógico alto para o pino 6 - LED VERDE 1
  digitalWrite(7, HIGH);            // Enviar nível lógico alto para o pino 7 - LED VERDE 2
  digitalWrite(8, HIGH);            // Enviar nível lógico alto para o pino 8 - LED VERDE 3
  delay(100);                       // Aguardar 100ms (0,1s)
  digitalWrite(9, HIGH);            // Enviar nível lógico alto para o pino 9  - LED AMARELO 1
  digitalWrite(10, HIGH);           // Enviar nível lógico alto para o pino 10 - LED AMARELO 2
  delay(100);                       // Aguardar 100ms (0,1s)
  digitalWrite(11, HIGH);           // Enviar nível lógico alto para o pino 11 - LED VERMELHO 1
  digitalWrite(12, HIGH);           // Enviar nível lógico alto para o pino 12 - LED VERMELHO 2
  digitalWrite(13, HIGH);           // Enviar nível lógico alto para o pino 13 - LED VERMELHO 3
  delay(500);                       // Aguardar 100ms (0,5s)

  digitalWrite(6, LOW);             // Enviar nível lógico baixo para o pino 6
  digitalWrite(7, LOW);             // Enviar nível lógico baixo para o pino 7
  digitalWrite(8, LOW);             // Enviar nível lógico baixo para o pino 8
  digitalWrite(9, LOW);             // Enviar nível lógico baixo para o pino 9
  digitalWrite(10, LOW);            // Enviar nível lógico baixo para o pino 10
  digitalWrite(11, LOW);            // Enviar nível lógico baixo para o pino 11
  digitalWrite(12, LOW);            // Enviar nível lógico baixo para o pino 12
  digitalWrite(13, LOW);            // Enviar nível lógico baixo para o pino 13
  delay(50);                        // Aguardar 100ms (0,05s)
}
```

Agora basta compilar o seu código e testar o funcionamento, verificar se não ocorreu nenhum erro e comemorar mais uma etapa vencida.

Veja o funcionamento desse desafio pelo QR code ou pelo link http://bit.ly/DESAFIO4_EDG_F3

DESAFIO INDIVIDUAL Nº5

Diversos equipamentos encontrados no nosso dia a dia exibem uma contagem numérica através de um display indicativo de tempo, como por exemplo um forno microondas que ao ser programado e acionado indica o tempo restante para que o processo de aquecimento seja encerrado, muitas vezes funcionando de maneira regressiva, ou seja, do tempo programado até 0 segundos, indicando o fim da atividade.

Esse recurso é utilizado em muitos outros dispositivos, como sistemas de irrigação, bombeamento de agua, temporização da lavagem de roupa, e em muitos outros equipamentos.

Você deve ter observado que no shield para arduino starter do eletrônica fácil existe um display de 7 segmentos. **Seu Objetivo** no desafio 4 será elaborar um código para executar uma **contagem regressiva de 9 a 0,** utilizando o display de 7 segmentos**.**

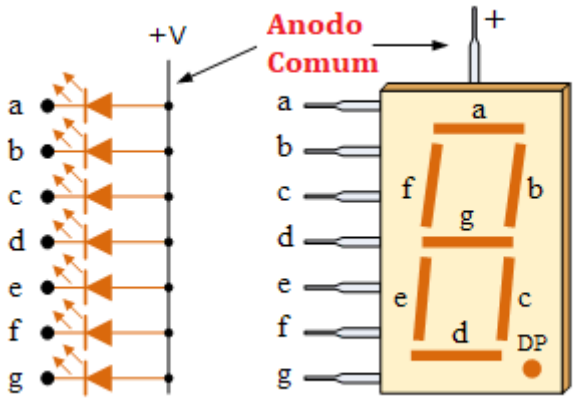
O seu grande desafio será, além de elaborar a lógica de contagem de tempo, fazer acender cada representação de número no display de 7 segmentos.

Para realizar esse desafio precisamos entender como o display de 7 segmentos funciona, e como ele será acionado no shield nosso shield starter.

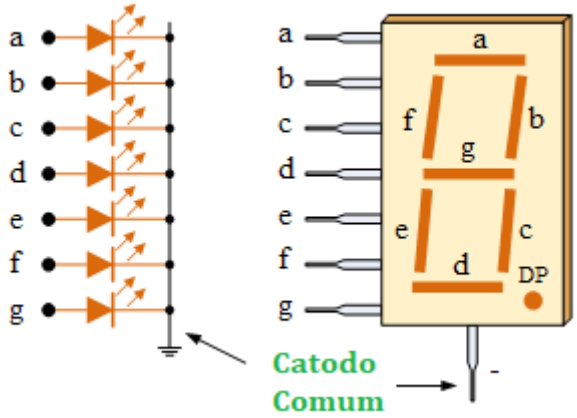
O que é um display de 7 segmentos

O display de sete segmentos recebe esse nome pois é composto por 7 partes que acendem, ou seja, cada segmento corresponde a um led. Sabemos que um led é composto por um anodo (+) e por um catodo (-), e um display de 7 segmentos pode ser de 2 tipos: **anodo comum** ou **catodo comum.**

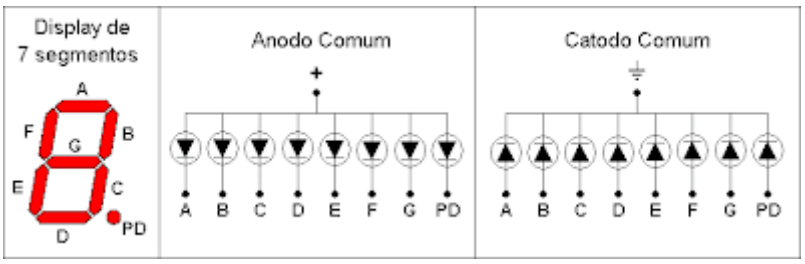
Anodo comum é o display que possui todos os anodos conetados em um único ponto positivo, para que o acionamento de cada (segmento) led seja feito individualmente pelos terminais negativos.



Catodo comum é o display que possui todos os terminais catodos dos segmentos conectados em um único ponto negativo, para que o acionamento dos leds seja feito individualmente pelos seus terminais positivos.

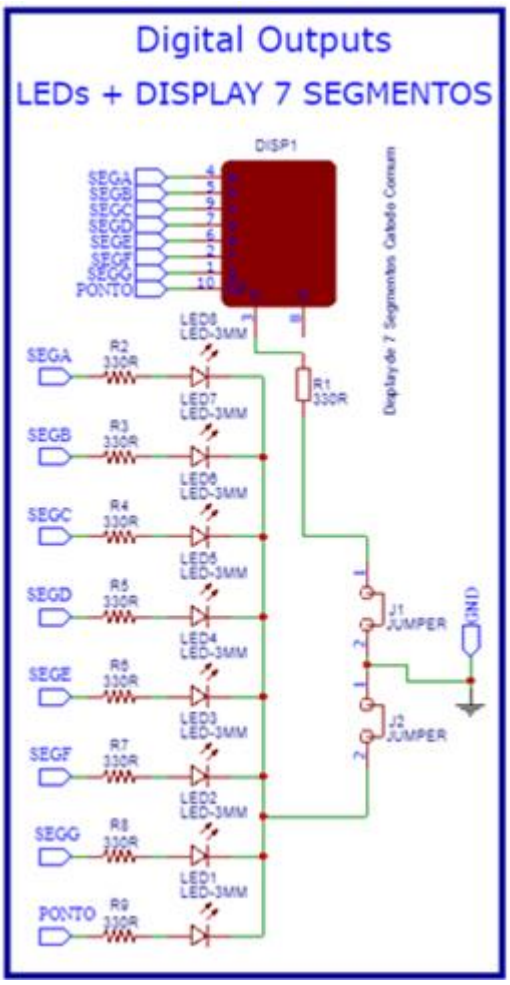


Resumindo, um tipo de display é acionado com tensão negativa e o outro com tensão positiva. Essa informação será muito importante na hora de elaborarmos nosso desafio, pois será fundamental saber se o acionamento do segmento será feito com nivel alto ou baixo.



Agora que já ficou claro que o display de 7 segmentos é apenas um conjuntos de leds, você já deve ter percebido que, de acordo com a configuração de ativação dos segmentos, um número poderá ser desenhado no dispositivo.

Veja pelo esquemático ao lado que os mesmos pinos que ativam os leds estudados anteriormente, também ativam os segmentos do display, porém, para que os segmentos do display sejam acionados o jumper deve estar posicionado na **posição J1**, para que o negativo vá para o catodo comum do display. Sabendo que o display é catodo comum, a ativação de cada segmento será feita com **nível digitais altos**.



Assim**, os segmentos A, B, C, D, E, F, G** e ponto, serão controlados respectivamente pelos pinos **13, 12, 11, 10, 9, 8, 7 e 6.** Veja a tabela:

Pino do Arduino	Segmento do display
13	a
12	b
11	c
10	d
9	e
8	f
7	g

Chegamos ao ponto crucial do nosso estudo para que o desafio nº4 seja realizado, que é entender quais pinos deverão ser ativados para que cada número da contagem regressiva seja desenhado no display.

Para isso, vamos observar a tabela da verdade onde cada número que será desenhado no display será representado por sua combinação de ativação dos segmentos do display. Veja abaixo:

De acordo com a tabela ao lado, o número 1 será representado no display quando apenas os segmentos **b** e **c** estiverem em nível alto e todos os outros segmentos, **a, d, e, f, g,** em nível baixo.

Veja que o número 8 será representado no display quando todos os segmentos do display estiverem em nível alto.

Agora estamos prontos para começarmos a montar nossa lógica de programação do desafio proposto, que será elaborar uma contagem regressiva de 9 a 0 com o display de 7 segmentos. Vamos nessa?

Saídas 7 segmentos							Display
a	b	c	d	e	f	g	
1	1	1	1	1	1	0	0
0	1	1	0	0	0	0	1
1	1	0	1	1	0	1	2
1	1	1	1	0	0	1	3
0	1	1	0	0	1	1	4
1	0	1	1	0	1	1	5
1	0	1	1	1	1	1	6
1	1	1	0	0	0	0	7
1	1	1	1	1	1	1	8
1	1	1	0	0	1	1	9

Lembrando que a tensão elétrica de cada LED do display é de 2V, por isso no esquema eletrônico do shield existe um ***resistor*** *de vai do comum do display para o negativo, para que a tensão aplicada no led seja dividida com ele quando o Arduino Uno enviar 5V, portando 2V irão ficar nos LEDs do Display e os outros 3V no resistor de 330R.* ***NUNCA envie 5V diretamente para qualquer LED pois isso poderá queimá-lo****. O resistor em série com o comum do display do projeto é o braço direito para que cada LED funcione com segurança.*

DESAFIO Nº5 – desafio individual

O desafio nº 5 será elaborar um código para contagem regressiva de 9 a 0, utilizando o display de 7 segmentos do shield para arduíno starter.

INSTRUÇÕES

- ✓ Controlar os segmentos do display do tipo catodo comum
- ✓ Os segmentos do display são controlados pelos pinos 7 ao 13
- ✓ Cada segundo passado de contagem deverá apresentar seu algarismo numérico correspondente a contagem regressiva de 9 a 0.
- ✓ Ao chegar no algarismo 0 repetir todo processo novamente

APLICAÇÕES

- ✓ Indicador de tempos de término de aquecimento em fornos eletrônicos

Antes de iniciarmos o código de programação, vamos apresentar uma maneira de deixar mais legível o código do que é controlado por cada pino no arduino.

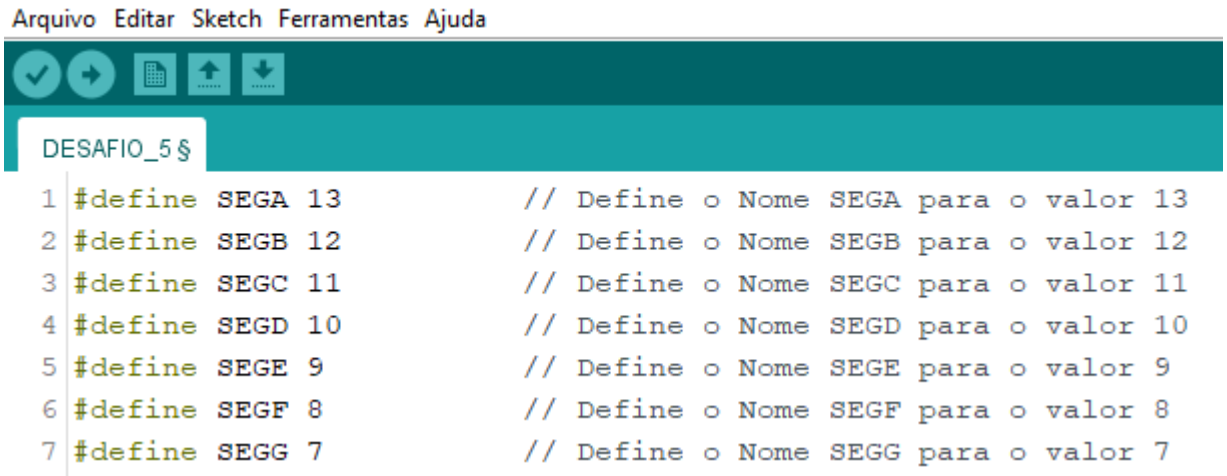
Existe um recurso na programação chamado **#define** e é utilizado para nomear um determinado pino, dessa forma não precisamos decorar o que determinado pino está controlando, já nomeamos esse pino com o nome do que ele estará controlando e dessa forma ele poderá ser identificado imediatamente no código, facilitando a interpretação e desenvolvimento do programa.

Por exemplo, o pino 13 controla o segmento A do display, poderemos nomear o pino 13 como SEGMENTO A, facilitando a interpretação do código.

Essa definição deverá ser feita no início do código fora de todas as funções. Veja como é feito essa definição:

```
1 #define SEGA 13          // Define o Nome SEGA para o valor 13
```

Simples, não é mesmo? Basta usar a **#define**, dar espaço e colocar o **nome que deseja dar ao pino**, dar um espaço e colocar o **pino que será nomeado**.

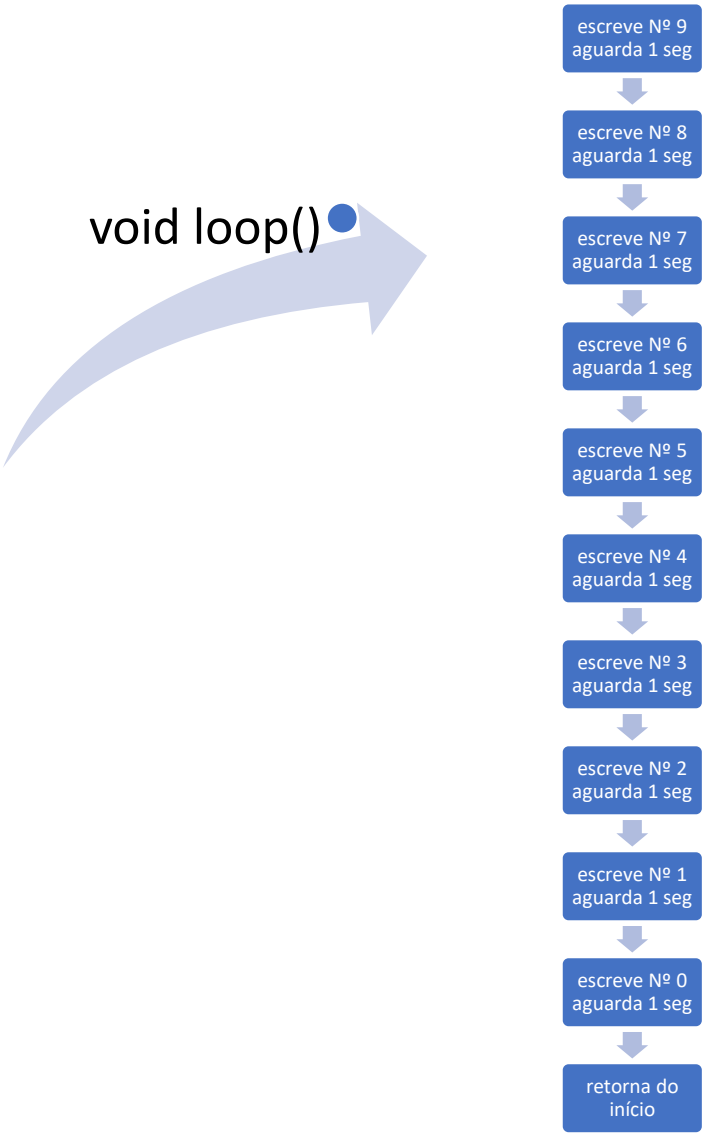


Depois de nomearmos os pinos com seus respectivos pinos de controle, vamos iniciar o código. A primeira etapa será definir cada pino como saída na função void setup ().

Obs. Não colocaremos mais o número do pino e sim o nome dele, afinal nomeamos cada pino.

```
10 void setup()              // Início da função de configuração
11 {                         // chave de abertura da função
12   pinMode(SEGG, OUTPUT);  // Configura o modo do pino 7  como Saída Digital
13   pinMode(SEGF, OUTPUT);  // Configura o modo do pino 8  como Saída Digital
14   pinMode(SEGE, OUTPUT);  // Configura o modo do pino 9  como Saída Digital
15   pinMode(SEGD, OUTPUT);  // Configura o modo do pino 10 como Saída Digital
16   pinMode(SEGC, OUTPUT);  // Configura o modo do pino 11 como Saída Digital
17   pinMode(SEGB, OUTPUT);  // Configura o modo do pino 12 como Saída Digital
18   pinMode(SEGA, OUTPUT);  // Configura o modo do pino 13 como Saída Digital
19 }                         // chave de fechamento da função
```

Prontinho, configurações feitas, vamos prosseguir com nosso código. Como de costume vamos montar nosso diagrama de blocos para facilitar o entendimento da lógica aplicada.



Observe que o código será muito simples, precisamos escrever o número no display e aguardar 1 segundo, só isso!

Para escrever cada número no display, basta consultar a tabela da verdade para ver quais segmentos deverão ser acionados para a formação dos algarismos numéricos.

```
//-----------ESCREVE N-9 NO DISPLAY-------------------------//
 digitalWrite(SEGA, HIGH);
 digitalWrite(SEGB, HIGH);
 digitalWrite(SEGC, HIGH);
 digitalWrite(SEGD, LOW);
 digitalWrite(SEGE, LOW);
 digitalWrite(SEGF, HIGH);
 digitalWrite(SEGG, HIGH);
 delay (1000); // aguarda 1 segundo
//-----------TÉRMINO DA ESCRITA N-9 NO DISPLAY--------------//
```

```
//-----------ESCREVE N-8 NO DISPLAY-------------------------//
digitalWrite(SEGA, HIGH);
digitalWrite(SEGB, HIGH);
digitalWrite(SEGC, HIGH);
digitalWrite(SEGD, HIGH);
digitalWrite(SEGE, HIGH);
digitalWrite(SEGF, HIGH);
digitalWrite(SEGG, HIGH);
delay (1000); // aguarda 1 segundo
//-----------TÉRMINO DA ESCRITA N-8 NO DISPLAY--------------//
```

Queridos amigos e alunos da familia eletrônica fácil, até o momento, mostramos a lógica completa de funcionamento de todos os programas do desafio 1 ao desafio 4, porém chegou a hora de você exercitar seu conhecimentos e desenvolver todo o código de programação do desafio nº5. Siga a sequência abaixo, executando parte a parte da produção do código, que seu projeto será um sucesso:

1. *Dentro do loop( ), escrever o trecho do código que imprime o número 9, aguardar 1 segundo, escrever o número 8, aguardar mais 1 segundo, e feche a função loop() do código para voltar ao início. Compile o código e veja se está funcionando corretamente para seguir adiante;*

2. *Validada a parte anterior, escrever o trecho do código que imprime o número 7, aguardar 1 segundo, escrever o número 6, aguardar mais 1 segundo, e feche a função loop() do código para voltar ao início. Compile o código e veja se está funcionando corretamente para seguir adiante;*

3. *Validada a parte anterior, escrever o trecho do código que imprime o número 5, aguardar 1 segundo, escrever o número 4, aguardar mais 1 segundo, e feche a função loop() do código para voltar ao início. Compile o código e veja se está funcionando corretamente para seguir adiante;*

4. *Validada a parte anterior, escrever o trecho do código que imprime o número 3, aguardar 1 segundo, escrever o número 2, aguardar mais 1 segundo, e feche a função loop() do código para voltar ao início. Compile o código e veja se está funcionando corretamente para seguir adiante;*

5. *Validada a parte anterior, escrever o trecho do código que imprime o número 3, aguardar 1 segundo, escrever o número 2, aguardar mais 1 segundo, e feche a função loop() do código para voltar ao início. Terminada esta tarefa você terá encerrado o seu código da contagem regressiva de 9 a 0 de forma show de bola;*

6. *Enviar o código no Grupo Vip de alunos com a #ERDINGER #DESAFIOF12 #DESAFIO5*

7. *Enviar um vídeo de cada fase desenvolvida no grupo vip de alunos do curso de eletrônica digital e arduino para iniciantes para que possamos comemorar com você o seu sucesso e evolução profissional.*

8. *Aproveite para postar seus vídeos nas redes sociais como Facebook e Instagram e marque o @eletronica_facil_oficial para que possamos curtir seu sucesso e repostá-lo em nossas redes também para marcarmos história juntos.*

Você pode verificar o funcionamento desse desafio por meio do QR code ao lado ou pelo link http://bit.ly/DESAFIO5_Cont_Display

Método ERDINGER

Fase 4: Desafio do mestre

CURSO DE

ELETRÔNICA FÁCIL

FASE 4 – DESAFIO DO MESTRE

Queridos amigos e alunos da família eletrônica fácil, chegamos na última fase da metodologia ERDINGER, o Desafio do Mestre. Neste momento vocês irão testar os conhecimentos obtidos nas fases 1, 2 e 3 deste treinamento.

NÃO PULE ETAPAS, TODOS OS CONHECIMENTOS ANTERIORES SERÃO CRUCIAIS PARA O SUCESSO NOS SEUS ESTUDOS.

Vamos iniciar essa etapa propondo algumas questões. Caso já tenha conhecimento sobre o que estará sendo perguntado, você poderá treinar ainda mais a profundidade dos seus conhecimentos, porém se não souber, faça uma pesquisa utilizando as dicas oferecidas durante o estudo deste e-book. Vamos nessa?

PERGUNTAS BÔNUS PARA ALAVANCAR SEUS CONHECIMENTOS

Pergunta Bônus 1: O que é um sinal eletrônico PWM?

Pergunta Bônus 2: O que significa duty cicle em um sinal PWM?

Pergunta Bônus 3: Como se calcula o duty cicle de um sinal PWM?

Pergunta Bônus 4: O que significa o termo TL para o sinal PWM?

Pergunta Bônus 5: O que significa o termo TH para o sinal PWM?

Pergunta Bônus 6: O que significa o termo TT para o sinal PWM?

Pergunta Bônus 7: O que significa o termo Vm para o sinal PWM?

Pergunta Bônus 8: Como Calcular o Vm de um sinal PWM?

Pergunta Bônus 9: Um sinal PWM pode ser considerado um sinal analógico ou digital?

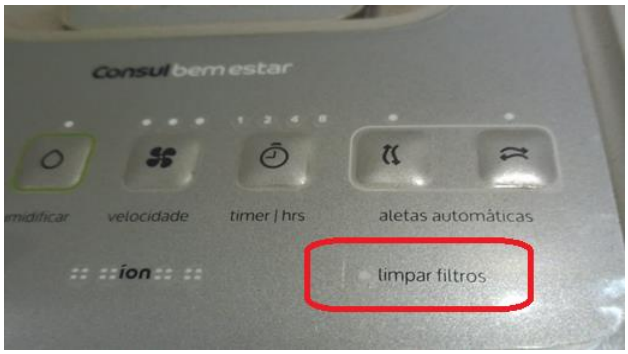
Dica rápida: https://www. youtube.com/watch?v=z4Rm6BrruxU

Todas as questões colocadas como sugestão de pesquisa serão tratadas posteriormente neste treinamento, por isso o objetivo agora é fazer com que a curiosidade seja despertada e cada um possa obter um conhecimento prévio para os próximos estudos.

Chegou a hora, vamos para o grande desafio do mestre.

DESAFIO DO MESTRE N.º 1

Sua missão será fazer o desenvolvimento da programação de um microcontrolador de um climatizador da empresa Consul, e uma das tarefas que ele terá que executar é indicar ao usuário a necessidade de efetuar a limpeza de filtro do climatizador.



Para fazer essa sinalização ao usuário a empresa optou por utilizar um led piscante em frequência de 4Hz, que será acionado sempre que for necessário fazer a limpeza de filtro.

Como esse é o primeiro desafio com esse tipo de problema, vamos dar uma força para você compreender o caminho a seguir. Estamos juntos nessa.

Controlando um LED em função da frequência

Para controlar um dispositivo em função de uma determinada frequência precisamos analisar as etapas de controle desse dispositivo e dividir esses processos dentro do tempo desejado. Para isso, vamos pensar no nosso desafio, que é fazer o led piscar em uma frequência de 4Hz.

Para que o led pisque, será necessário que ele seja ativado, aguarde um tempo, seja desativo e aguarde outro tempo, repetindo este processo pelo tempo desejado. Isso é o funcionamento de um led piscante.

Agora vem a pergunta, qual será o tempo necessário de ativação e desligamento do led para que ele pisque em 4 Hertz?

Piscar em 4 hertz significa dizer que esse led irá acender 4 vezes em 1 segundo.

Dividindo 1 segundo em 4 partes temos:

$$Tempo = \frac{1}{4}$$ ***Portanto, o tempo será de 0,250s, ou 250ms***

Já sabemos que o led piscara a cada 250ms, porém ele também precisará desligar. Então, vamos dividir esse tempo de 250ms em duas partes, uma para ligar e outra para desligar.

$$Tempo = \frac{250}{2}$$ ***Portanto, on e off terão 125ms cada***

Concluímos que, para piscar um led na frequência de 4 Hertz, será necessário manter o led ligado por 125ms e desliga-lo também por 125ms.

Dessa forma, o arduino deverá mandar nível lógico HIGH e LOW nos tempos calculados anteriormente, formando uma onda quadrada, ligando e desligando.

Cada parte dessa onda quadrada irá receber um nome, veja:

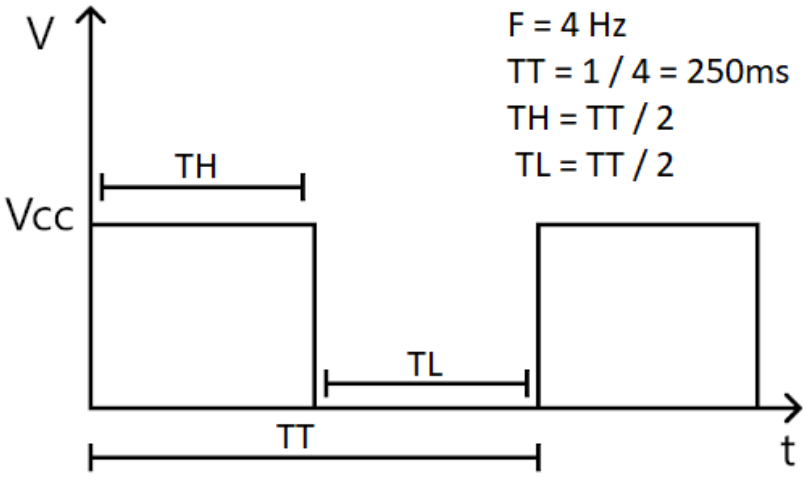


*Figura 76 - **TT** = Tempo Total - **TH** = Tempo HIGH - **TL** = Tempo LOW*

Você deve estar se perguntando, é possível visualizar essa onda? Mas como vou observar essa onda quadrada com o Arduino? Essa demonstração acima foi apenas por conhecimento? Fique tranquilo, iremos entender isso mais a frente, neste momento vamos apenas assimilar o que o desafio pediu e mostrar que existe um mundo por trás desse simples pisca-pisca.

DESAFIO Nº 1 – desafio do mestre

Programar o microcontrolador que irá controlar um led sinalizador de limpeza de filtro para a produção de climatizadores da empresa CONSUL.

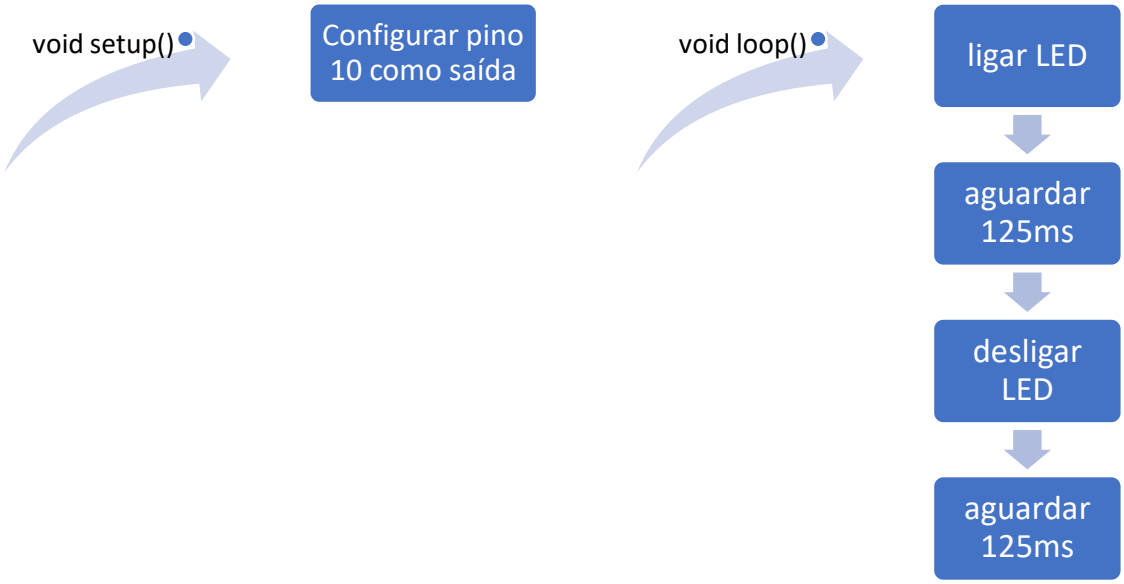
INSTRUÇÕES

- ✓ Utilizar pino 10 para controlar led
- ✓ Led piscará em 4Hz
- ✓ Utilizar função digitalWrite () para ativar e desativar o led
- ✓ Escrever código dentro da função void loop ()

APLICAÇÕES

- ✓ Indicador de necessidade de limpeza de filtro

Agora é a hora de elaborar nosso diagrama de blocos com as necessidades do projeto.

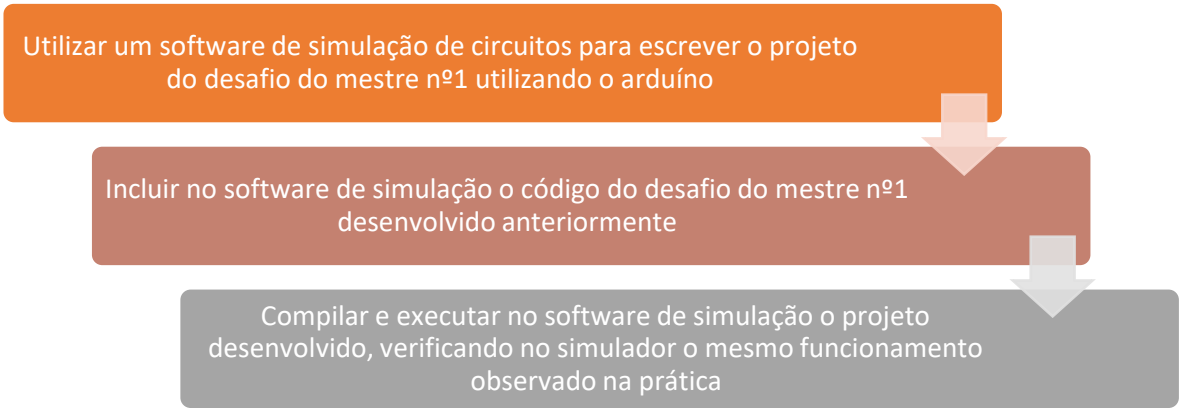


Agora é com você, desenvolva o código, compile e grave no arduino para ver se tudo ocorreu como o desejado, sem apresentar erros e atendendo a necessidade da empresa CONSUL.

DESAFIO BÔNUS – software de simulação

Você deve estar lembrado que anteriormente apresentamos uma onda quadrada gerada pelo código de programação do Arduino para resolução do desafio do mestre n.º 1. Agora você poderá observar essa onda, fazer medições, e comprovar no simulador o perfil quadrático formado pela execução do código desenvolvido para piscar o led.

Caso você tenha, os mesmos testes podem ser feitos com a utilização de um osciloscópio no pino de saída do arduino que controla o led, porém neste momento, levando em consideração que é uma ferramenta com alto valor de investimento e muitos ainda não a possuem, utilizaremos um software de simulação.



Após tudo funcionar correntemente no software de simulação, vamos incluir o osciloscópio para fazer medições.

Adicionar o osciloscópio no software de simulação e verificar a frequência do pino 10 que faz o controle do led.

Utilizar a ferramenta cursor e verificar os valores de TT, TH, TL, e comparar com os calculos feitos anteriormente.

Comparar os valores obtidos na simulação com os resultados obtidos na prática.

Utilizar a ferramenta DEBUG para verificar o passo a passo (step by step) da execução do código de programação utilizado

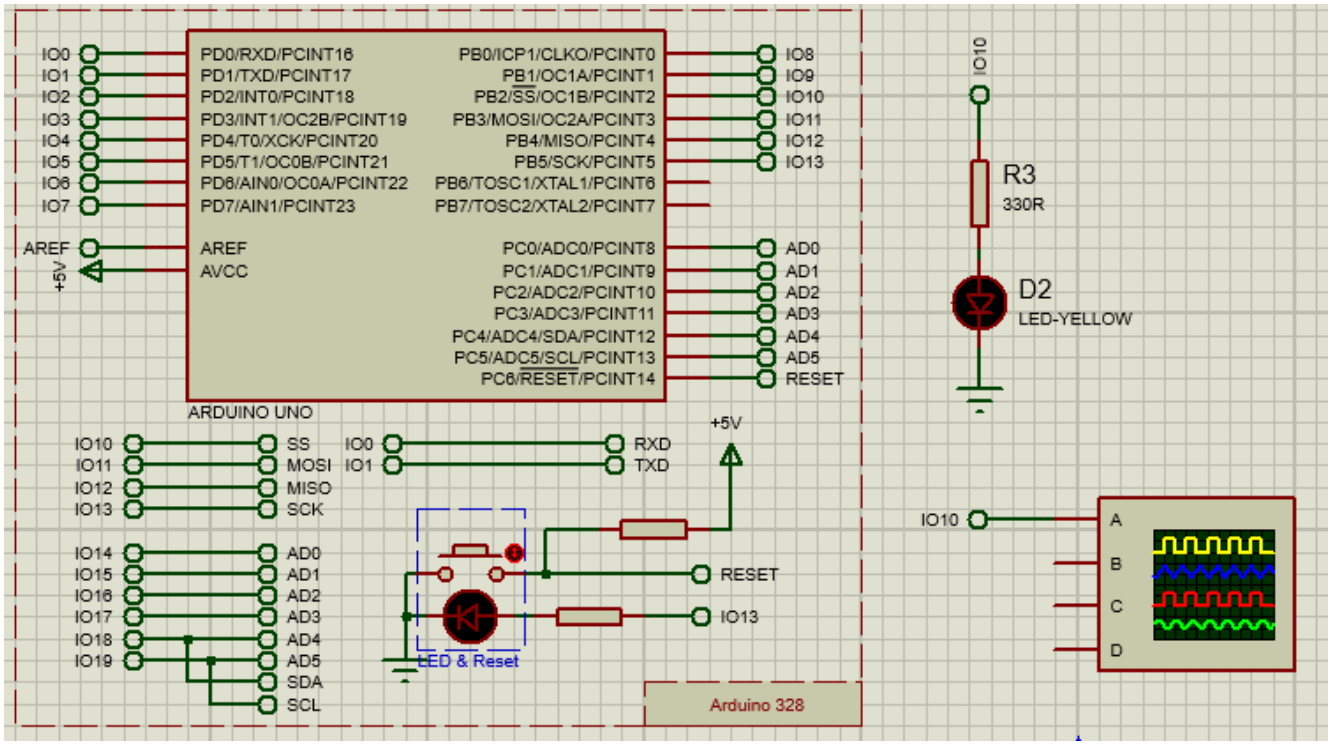


Figura 77 - Circuito eletrônico montado no software proteus para simular o funcionamento do desafio do mestre 1

Para você que é aluno ELETRÔNICA FÁCIL, basta acessar o conteúdo dessa aula contendo todas as explicações de como fazer a simulação, montagem do projeto, medições com o osciloscópio, análise de sequência de execução das linhas do código de programação e muito mais. Vamos pra cima que vem muito mais conteúdos por aí. #tamojunto

DESAFIO DO MESTRE N.º 2

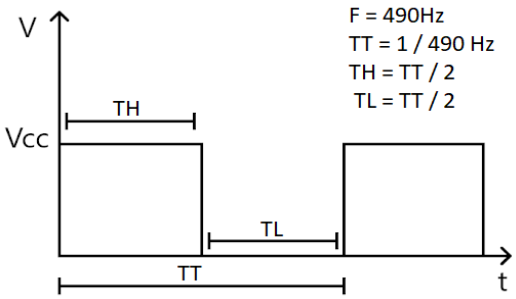
Aproveitando todo o trabalho desenvolvido no desafio 1, iremos modificar a programação para que ela gere uma onda quadrada com a frequência de 490 Hz para o pino 10.

DESAFIO Nº 2 – desafio do mestre

Programar o arduino para que um sinal de 490Hz seja gerado no pino 10, controlando o LED. Ou seja, o perfil de onda quadrada desejada será obtida ligando e desligando o nível de tensão que é enviado para o pino 10 do arduino.

INSTRUÇÕES

- ✓ Utilizar pino 10 para saída de sinal
- ✓ Produzir sinal de 490Hz
- ✓ Escrever código dentro da função void loop ()
- ✓ Fazer os cálculos aprendidos anteriormente para identificar os tempos on e off.
- ✓ Utilizar a função delayMicroseconds ()

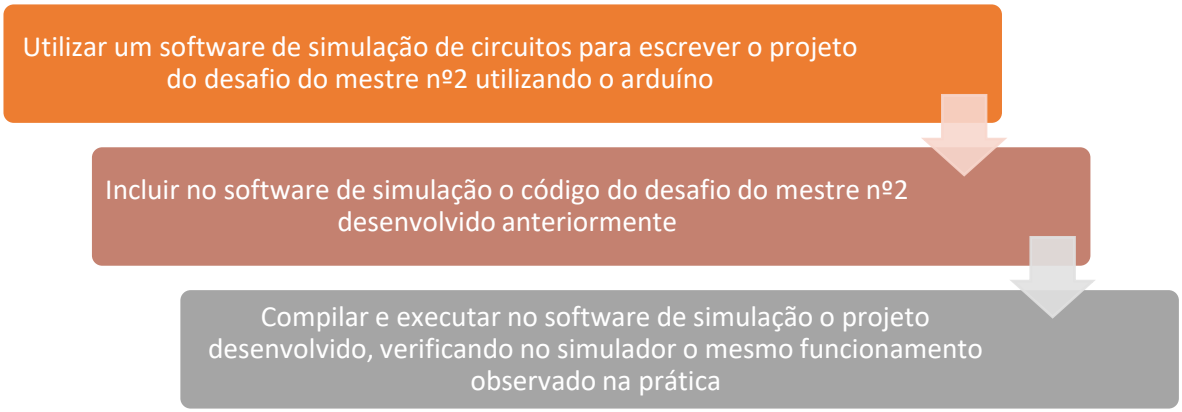


APLICAÇÕES

- ✓ Aparelhos que utilizem diferentes níveis de frequência em seu funcionamento

Ao calcular o TH e TL você notará que o valor obtido em milissegundo será quebrado, e não será possível utilizar a função delay para construir o código, por isso o valor deverá ser convertido para microssegundos e você poderá utilizar a função delayMicroseconds. Caso fique alguma dúvida, faça uma pesquisa utilizando as dicas aprendidas nesta metodologia.

Após compilar seu código e executar no Arduino, confirmando o funcionamento esperado sem apresentar erros, siga para a simulação no software.



Adicionar o osciloscópio no software de simulação e verificar a frequência do pino 10 que faz o controle do led.

Utilizar a ferramenta cursor e verificar os valores de TT, TH, TL, e comparar com os calculos feitos anteriormente.

Comparar os valores obtidos na simulação com os resultados obtidos na prática.

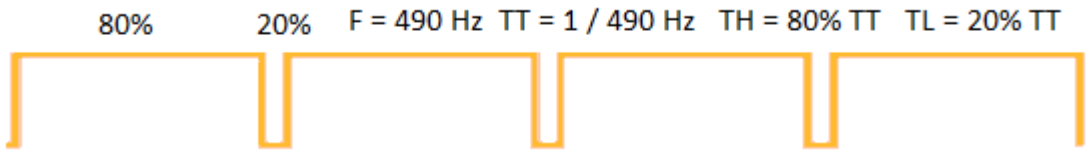
Utilizar a ferramenta DEBUG para verificar o passo a passo (step by step) da execução do código de programação utilizado

DESAFIO DO MESTRE N.º 3

DESAFIO Nº 3 – desafio do mestre

Aproveitando todo o trabalho desenvolvido no desafio 2, modificar a programação para que ela continue gerando uma onda quadrada com a frequência de 490 Hz no pino 10, porém o nível 1 da onda funcionará em 80% do tempo e nível 0 em apenas 20% do tempo do ciclo completo.

INSTRUÇÕES



- ✓ Utilizar pino 10 para saída de sinal
- ✓ Produzir sinal de 490Hz
- ✓ Calcular Tempo Total TT da oscilação
- ✓ Calcular tempo TH em 80%
- ✓ Calcular tempo TL em 20%
- ✓ Construir código dentro da função void loop ()
- ✓ Utilizar a função delayMicroseconds ()

APLICAÇÕES

- ✓ Aparelhos que utilizem diferentes níveis de frequência em seu funcionamento

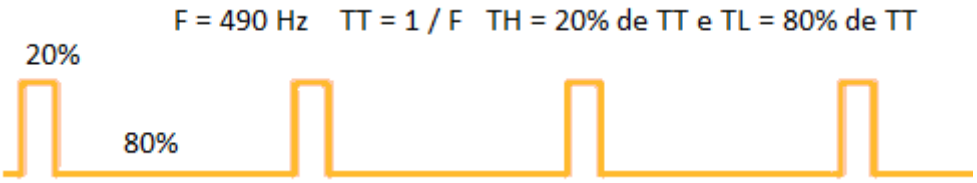
Após compilar seu código e executar no Arduino, confirmando o funcionamento esperado sem apresentar erros, siga para a simulação no software já descrita anteriormente em outros desafios.

DESAFIO DO MESTRE N.º 4

DESAFIO Nº 4 – desafio do mestre

Aproveitando todo o trabalho desenvolvido no desafio 3, modificar a programação para que ela continue gerando uma onda quadrada com a frequência de 490 Hz no pino 10, porém agora manipulando o nível lógico contido no pino 10 do Arduino Uno de forma a enviar nível 1 em 20% do tempo do ciclo e nível 0 em 80% do tempo do ciclo.

INSTRUÇÕES



- ✓ Utilizar pino 10 para saída de sinal
- ✓ Produzir sinal de 490Hz
- ✓ Calcular Tempo Total TT da oscilação
- ✓ Calcular tempo TH em 20%
- ✓ Calcular tempo TL em 80%
- ✓ Construir código dentro da função void loop ()
- ✓ Utilizar a função delayMicroseconds ()

APLICAÇÕES

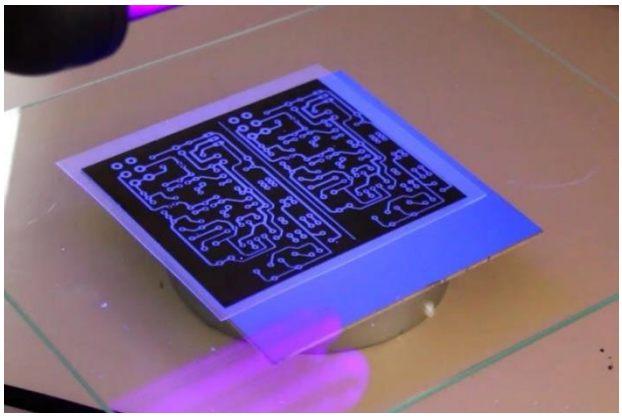
- ✓ Aparelhos que utilizem diferentes níveis de frequência em seu funcionamento

Após compilar seu código e executar no Arduino, confirmando o funcionamento esperado sem apresentar erros, siga para a simulação no software já descrita anteriormente em outros desafios.

DESAFIO DO MESTRE N.º 5

O desafio nº5 tem por objetivo fazer a programação de um microcontrolador que irá controlar um dispositivo de emissão de luz ultravioleta, muito utilizado em produções de placas para projetos eletrônicos.

Basicamente o dispositivo irá fazer o controle de tempo da luz ultravioleta que será enviada para um material fotosensivel, que fará com que as placas sejam desenhadas de acodo com o layout desenvolvido pelo projetista, surgindo dessa forma as famosas trilhas das pcbs.



O usuário desenvolvedor irá colocar uma placa eletrônica com a tinta foto sensível aplicada na camada de cobre, depois o Layout do circuito será adicionado sobre a placa, de forma que as trilhas da pcb NÃO ESTARÃO com tinta preta para que a luz UV penetre e faça o desenho na na PCB. Já a parte que está com tinta preta sairá no processo posterior de revelação.

NOTA DE ADVERTÊNCIA E SEGURANÇA

Atenção, evite olhar por muito tempo para as lâmpadas UV com luz azul, pois isso poderá causar irritação aos olhos, dores de cabeça, entre outros sintomas indesejados. Sempre considere a sua segurança e saúde em primeiro lugar. Note que os diversos equipamentos que usam a luz UV com cor azul possuem proteção para que a luz não seja emitida diretamente aos olhos do usuário, como é o caso do equipamento secador de unhas demonstrado no e-book e também no curso de eletrônica digital e arduino para iniciantes. Cuidado, durante a demonstração do funcionamento do desafio do mestre, o secador de unhas foi desmontado para que ele fosse usado como ferramenta didática, porém é preciso muito cuidado com essa luz exposta diretamente aos olhos.

DESAFIO Nº 5 – desafio do mestre

Ao ligar o equipamento o led de sinalização deverá piscar em 4Hz durante 10s, alertando o usuário que o procedimento será iniciado. Ao terminar a inicialização, o led de sinalização ficará aceso e a lâmpada UV será ativada por 3 minutos. Neste momento, uma contagem de tempo em minutos deverá aparecer no display de 7 segmentos de forma regressiva, ou seja, ao ligar a lâmpada o display apresentará o numero 3 indicando que faltam 3 minutos. Passado 1 minuto o display apresentará o número 2 indicando 2 minutos restantes, em seguida o numero 1 será apresentado e por fim o 0. Ao terminar o tempo de iluminação o buzzer deverá emitir um sinal sonoro na frequencia de 2Hz durante 5s indicando término do processo. Somente após o término da sinalização sonora, a lampada UV deverá ser desligada. A partir deste momento o número 0 ficará piscando em uma frequência de 2Hz indicando finalização total de utilização do equipamento. O equipamento volta a inicialização assim que for desligado e ligado novamente..

INSTRUÇÕES

- ✓ Led de sinalização = pino 6
- ✓ Lâmpada UV = pino 4
- ✓ Buzzer = pino 5
- ✓ Led inicialização piscante em 4Hz durante 10s
- ✓ Led permanece ligado
- ✓ Lâmpada UV permanece 3 minutos ligada
- ✓ Contagem regressiva de minutos no display
- ✓ Contagem encerrada buzzer é ativado em 2Hz durante 5s.
- ✓ Encerrando alarme sonoro lâmpadas e buzzer serão desligados
- ✓ O número 0 passa a piscar em 2 Hz no display

APLICAÇÕES

- ✓ Programação de aparelhos para trabalhos em dispositivos fotossensíveis

Pino do Arduino	Dispositivo controlador
13	Segmento a
12	Segmento b
11	Segmento c
10	Segmento d
9	Segmento e
8	Segmento f
7	Segmento g
6	LED de sinalização (ponto)
5	Buzzer
4	Relé – Controle da Lâmpada UV

Para resolver este desafio, será preciso transformar esse grande problema proposto em problemas menores, veja:

PROBLEMA 1	•Piscar LED de inicialização em 4Hz durante 10s
PROBLEMA 2	•Acionar lâmpada UV e manter LED de sinalização aceso
PROBLEMA 3	•Escrever indicação regressiva em minutos no display de 7 segmentos
PROBLEMA 4	•Sinalização sonora em 2Hz por 5s indicando término do tempo
PROBLEMA 5	•Desligar lâmpadas e sinalização sonora
PROBLEMA 6	•Piscar o número zero continuamente em uma frequência de 2Hz.

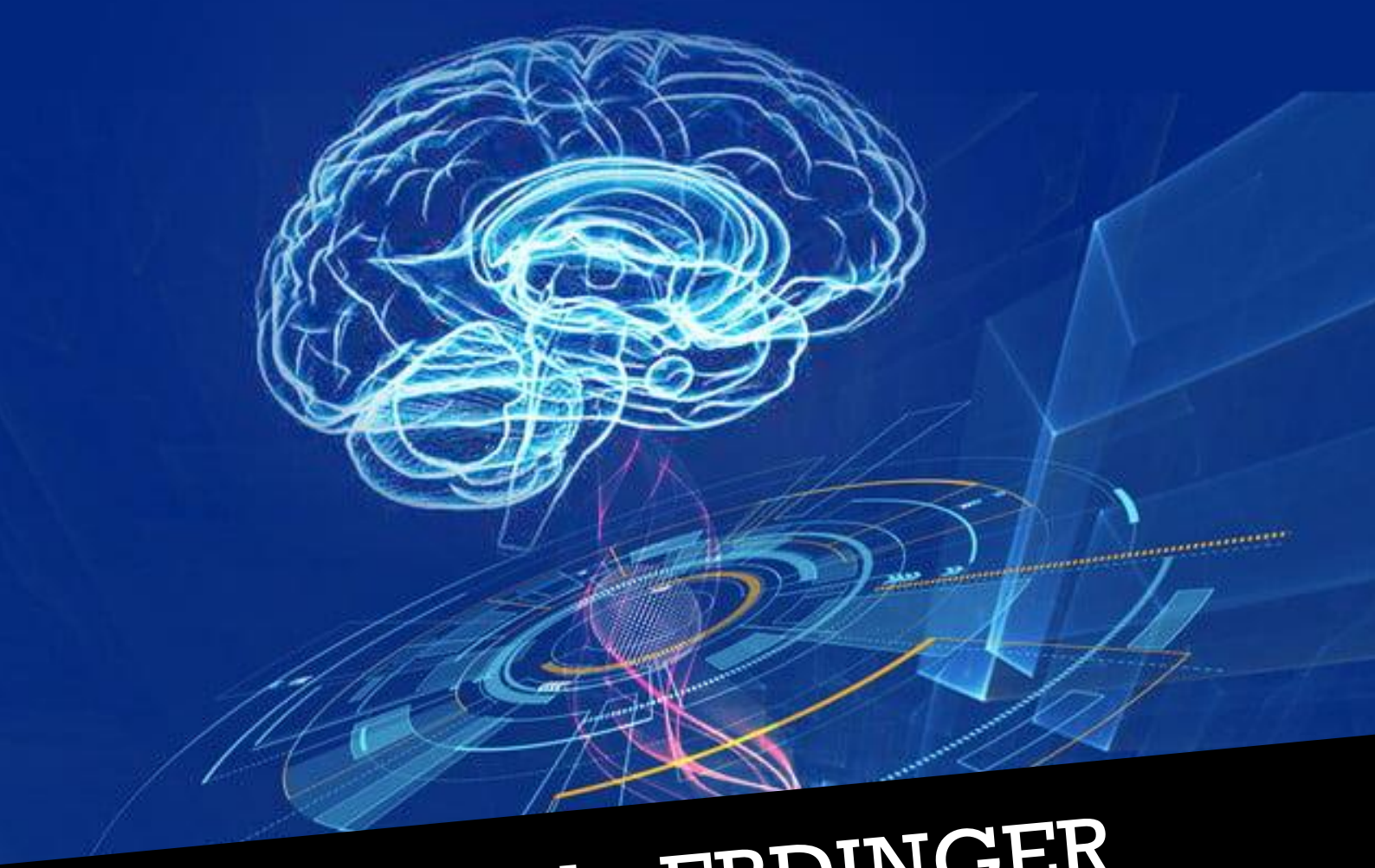
Neste momento encerramos o módulo 3 do Curso de Eletrônica Digital e Arduino para iniciantes, onde explicamos o passo a passo para colocar em prática a metodologia ERDINGER de aprendizagem e fomos em busca de soluções para os desafios apresentados.

Seguiremos, com muita garra e determinação para o modulo 4 deste treinamento, onde vamos desbravar ainda mais a programação Arduino Uno, aprendendo novos recursos, de forma incrível, show de bola e sensacional. Vejo vocês lá e #vamospracima!

Para saber mais informações sobre o curso completo de eletrônica digital e arduino para iniciantes acesse o link abaixo:

https://ead.cursoseletronicafacil.com.br/curso/curso-de-eletronica-digital-e-arduino-para-iniciantes/

ATENÇÃO – em breve uma nova versão deste e-book faixa preta vem por aí, com muito mais conteúdos e atualizações top de linha e sensacionais. Fique atento na página oficial do curso para sempre ter a versão atualizada do ebook. #tamojunto #familiaeletrônicafácil



Método ERDINGER

Leitura de Entradas Digitais

M4

CURSO DE

ELETRÔNICA FÁCIL

MODULO 4

INTRODUÇÃO A UM CONTEÚDO IMPORTANTÍSSIMO

Chegamos em mais um módulo mega importante do nosso treinamento e trataremos de um tema que será fundamental para que sua evolução seja ainda maior e cheia de novas descobertas: leitura de entradas digitais.

No dia a dia encontramos diversas ferramentas que estão constantemente aplicando a lógica do 0 e 1, a famosa lógica binária, interpretando informações e oferecendo maior praticidade na vida cotidiana.

Nessa primeira aula trataremos de uma máquina de self service de refrigerante de uma loja de lanches, onde o cliente ao apertar um simples botão recebe em seu copo refil o refrigerante escolhido. Essa praticidade acontece a partir de um sistema que está baseado na leitura de uma entrada digital, ou seja, um simples botão que pode estar ligado ou desligado, e diante desse estado tomar decisões servindo o produto desejado pelo cliente.

ERDINGER – APLICANDO A METODOLOGIA

Quando nos referimos à dispositivos de entrada, estamos automaticamente falando de um sistema que enviará sinais para uma central de controle, um cérebro digital, que fará a leitura e interpretação desse sinal, tomando as decisões programadas previamente pelo desenvolvedor. Então, dispositivos de entrada são dispositivos que enviam sinais para uma central de controle.

Aplicando a metodologia Erdinger, já temos o nosso **tema**:

LEITURA DE ENTRADAS DIGITAIS

Agora vamos **buscar um exemplo** na IDE do arduino para dar continuidade aos nossos estudos e poder materializar mais esse conhecimento.

Clicando em arquivos, exemplos, digital, e button, teremos um programa exemplo para leitura de entrada digitais. Veja a imagem abaixo:

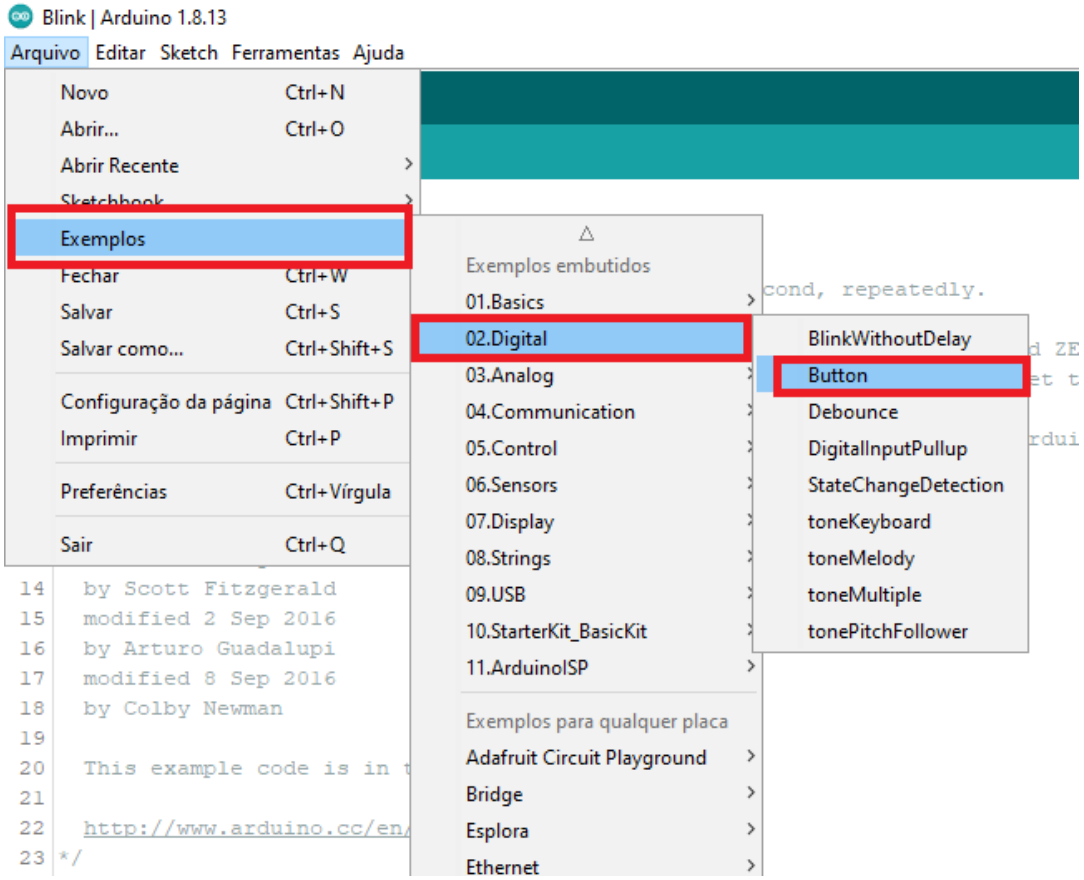


Figura 78 - Escolha do exemplo na IDE do Arduino

Veja abaixo o código exemplo button disponível na IDE do Arduíno.

```
// constants won't change. They're used here to set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed. If it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

O código exemplo já vem com uma descrição de como esse projeto irá funcionar, e caso você não tenha domínio da língua inglesa, você pode utilizar o google tradutor para entender a mensagem de instrução do código.

```
/*
  Button

  Turns on and off a light emitting diode(LED) connected to digital pin 13,
  when pressing a pushbutton attached to pin 2.

  The circuit:
  - LED attached from pin 13 to ground
  - pushbutton attached to pin 2 from +5V
  - 10K resistor attached to pin 2 from ground

  - Note: on most Arduinos there is already an LED on the board
    attached to pin 13.

  created 2005
  by DojoDave <http://www.0j0.org>
  modified 30 Aug 2011
  by Tom Igoe

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Button
*/
```

/*

Botão

Liga e desliga um diodo emissor de luz (LED) conectado ao pino digital 13, ao pressionar um botão conectado ao pino 2.

O circuito:

- LED conectado do pino 13 ao terra
- Botão conectado ao pino 2 de +5V
- Resistor de 10K conectado ao pino 2 do terra

- Nota: na maioria dos Arduino já existe um LED na placa conectado ao pino 13.

criado em 2005 por DojoDave <http://www.0j0.org> modificado em 30 de agosto de 2011 por Tom Igoe

Este código de exemplo é de domínio público.

http://www.arduino.cc/en/Tutorial/Button

*/

A partir da tradução feita acima, já conseguimos identificar os requisitos mínimos de hardware necessário para a execução desse projeto. Como nosso shield já vem projetado para atender a esse projeto, não teremos problemas para rodar e executar esse código.

Agora vamos conectar nosso Arduino no shield EF e carregar o código exemplo para verificar na prática o funcionamento desse código.

Utilize o Jumper na posição indicada na fotografia ao lado, unindo os dois pinos na posição J2.

O led conectado ao pino 13 será o led 8 do shield para arduino, portanto, esse será o led que irá acender nesse código exemplo escolhido.

Agora vamos compilar o programa e verificar se existe algum erro.

Não existindo erros, é hora de transferir o código do programa para nosso Arduino.

Após o código ter sido carregado, é hora de fazer o teste pressionando o botão s2.

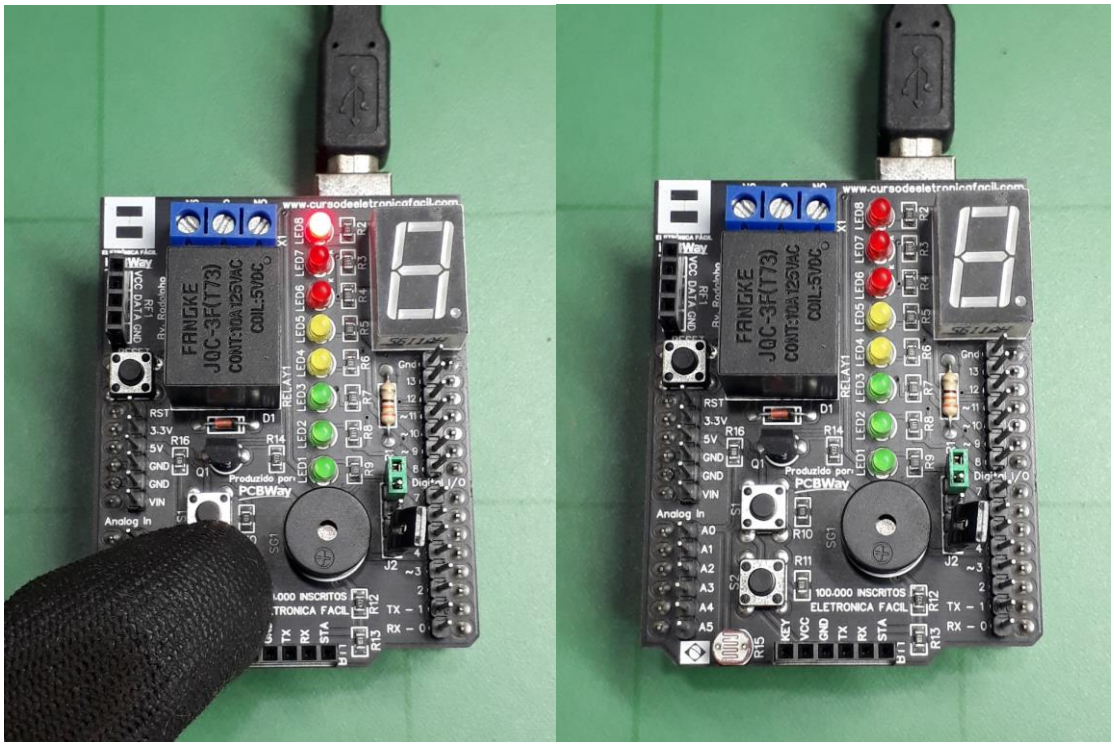


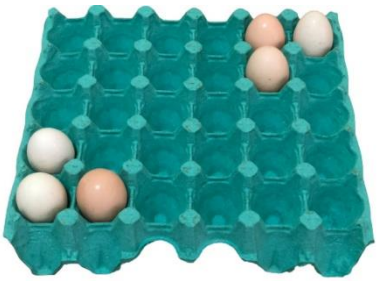
Figura 79 - Teste programa exemplo - entradas digitais

Excelente!!! Ao pressionar o botão s2 o led 8 ficou aceso. Ao soltar o botão o led se apagou. Top demais galera, mais um projeto simples e que pode ser comparado a máquina de refrigerante apresentada no início desse módulo, onde o usuário precisava pressionar um botão de um determinado sabor de refrigerante e então o liquido era liberado para encher seu copo. Veja o poder de um simples botão com informações digitais 1 e 0.

VARIÁVEIS – LOCAL PARA ARMAZENAR VALORES

Não é possível falar em programação sem falar em variáveis, mas o que realmente significa esse termo?

Vamos imaginar um armário de despensa em uma cozinha. Já pensou na variedade de embalagens que existem para guardar diversos tipos de alimentos sólidos, líquidos, pastoso, grãos, enfim, onde cada recipiente foi feito para que o armazenamento desse produto seja o melhor possível, tanto para acomodar os produtos dentro da embalagem, como para guarda-lo na prateleira ou mesmo transportá-lo.



Cada embalagem foi desenvolvida para um determinado produto. Por exemplo, não conseguiremos guardar ovos dentro de uma garrafa de refrigerante, justamente porque os tipos de armazenamentos são diferentes e incompatíveis. Da mesma forma que não conseguiremos armazenar refrigerante dentro de uma bandeja de ovos. Dessa forma fica claro que cada embalagem tem **um tipo especifico de armazenamento**.

Além do tipo de embalagem, podemos considerar que cada embalagem consegue guardar uma determinada **quantidade de produto** para a qual ela foi desenvolvida. Por exemplo, se uma garrafa consegue guardar até 2 litros de água, mais que isso serão necessárias duas garrafas ou uma garrafa de um tipo maior.

Então, além do tipo de embalagem também podemos considerar que ela tem um limite de armazenamento de produtos.

Assim também são as **variáveis** em linguagem de programação, são pequenos **“recipientes”** dedicados a armazenar um determinado tipo de informação. Esses recipientes são espaços da memória que ficam reservados para essa finalidade.

UMA VARIÁVEL É NADA MAIS QUE UM NOME QUE DAMOS A UMA DETERMINADA POSIÇÃO DE MEMÓRIA PARA CONTER UM VALOR DE UM DETERMINADO TIPO

Sempre que precisamos guardar um valor que pode se modificar durante a execução do programa, utilizamos as chamadas **variáveis**. Portanto, para que uma variável seja diferenciada de outra, ou seja, para que existam diversas variáveis em um código de um programa, será preciso fazer a identificação dessas variáveis com um nome e um tipo.

Por exemplo, se eu preciso armazenar a idade de uma pessoa, podemos criar a variável chamada de “idade”. Se precisarmos guardar a temperatura medida por um sensor, podemos nomear a variável como “temperatura”. Veja que o nome deve sempre ser muito intuitivo pois isso irá gerar maior **legibilidade** para o seu código e ficará mais fácil interpretá-lo.

Existem algumas regras na hora de nomear uma variável, veja abaixo:

- *O nome de uma variável pode ser constituído por letras do alfabeto (maiúsculas e minúsculas), dígitos (0 ... 9) e ainda pelo caractere underscore (_).*
- *O primeiro caractere não pode ser um digito. Terá que ser uma letra ou um underscore, porém não é aconselhável a utilização deste último.*
- *Maiúsculas e minúsculas representam caracteres diferentes.*
- *Uma variável não pode ter por nome uma palavra reservada a própria linguagem C. Dessa forma não é permitido criar variáveis com nomes do tipo int, float, char, já que essas palavras são utilizadas pela linguagem na identificação de tipos de dados.*

As variáveis podem assumir tipos de acordo com o tipo de dado que será armazenado:

TIPO	FUNÇÃO	TAMANHO	Menor e Maior valor
INT	Armazenar números inteiros	2 Bytes	- 32.768 a 32.767
FLOAT	Armazenar números reais, valores com partes fracionarias	4 Bytes	-2.147.483.648 a 2.147.483.647
CHAR	Armazenar UM ÚNICO caractere	1 byte	256 valores de caracteres

Na declaração de uma variável do tipo int ainda pode ser utilizados prefixos para melhorar a definição das características da variável. *Isso pode ser interessante quando um programa é utilizado em diferentes maquinas, onde uma variável int pode ser interpretada como tendo 2 bytes ou 4 bytes dependendo do compilador.*

Veja abaixo alguns prefixos:

short	2 bytes
long	4 bytes
signed	Inteiros positivos e negativos
unsigned	Inteiros apenas positivos

Aplicando esses prefixos você garante que a variável sempre tenha a mesma capacidade de armazenamento em diferentes compiladores.

EXEMPLOS DE APLICAÇÕES:

short *int = 2 bytes*

long *int = 4 bytes*

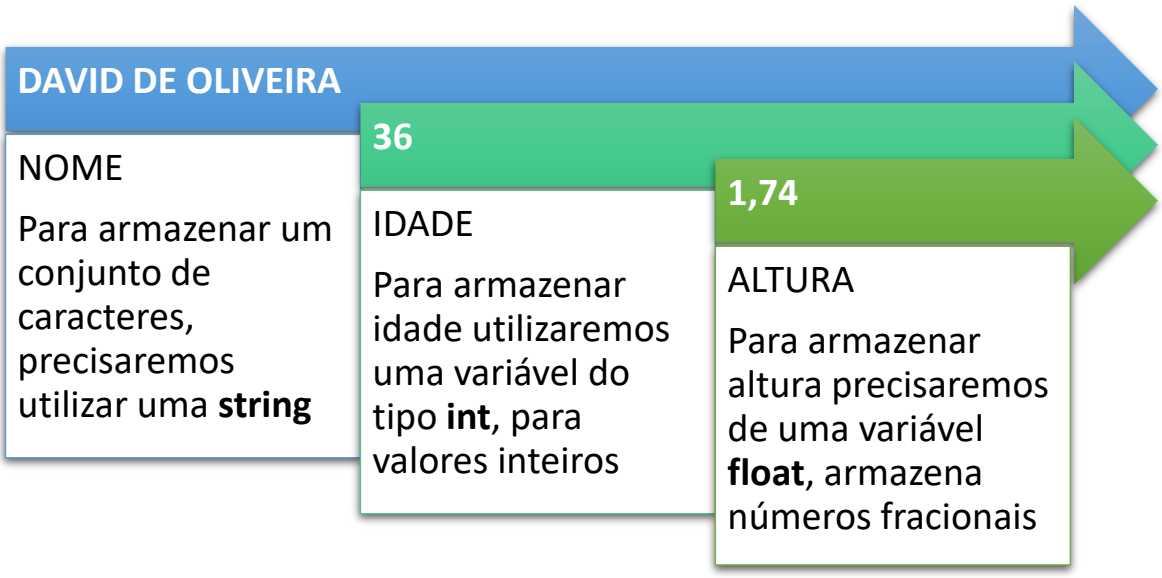
signed *int = números inteiros positivos e negativos, dessa forma a capacidade total de armazenamento se divide entre valores positivos e negativos*

unsigned *int = armazenamento total para inteiros positivos*

As variáveis possuem um ***TIPO*** e uma ***AMPLITUDE***. O **tipo** irá definir que tipo de **informação será armazenada**, se é um número inteiro, real, caractere, enfim. A **amplitude** define **o tamanho dessa informação**.

Nesse ponto é preciso muito cuidado, pois vamos imaginar a seguinte ocasião: Vamos supor que precisamos armazenar apenas informações digitais em uma variável, ou seja, 0 e 1. Não há necessidade de reservar um espaço gigante na memória para guardar essa informação, pois estamos reservando memória desnecessária e que não será totalmente utilizada. Por exemplo, utilizar *long int (4 bytes) para armazenar 0 e 1.* Também não podemos reservar pouco espaço para contagens ou números muito grandes, correndo o risco de uma variável não suportar armazenar todas essas informações. Por isso, é muito importante e necessário conhecer bem os dados que serão armazenados nas variáveis para que a escolha do tipo seja coerente com a necessidade.

Veja o que fazer ao criar três variáveis para armazenar o **nome,** a **idade** e **altura** de uma determinada pessoa.



Veja que, para cada tipo de dado, um TIPO de variável foi escolhido. A escolha incorreta de um tipo de armazenamento pode comprometer o funcionamento do seu programa.

Todos os tipos do ANSI C

Tipo	Num de bits	Intervalo
char	8	-128 a 127
unsigned char	8	0 a 255
signed char	8	-128 a 127
int	16	-32.768 a 32.767
unsigned int	16	0 a 65.535
signed int	16	-32.768 a 32.767
short int	16	-32.768 a 32.767
unsigned short int	16	0 a 65.535
signed short int	16	-32.768 a 32.767
long int	32	-2.147.483.648 a 2.147.483.647
signed long int	32	-2.147.483.648 a 2.147.483.647
unsigned long int	32	0 a 4.294.967.295
float	32	3,4E-38 a 3.4E+38
double	64	1,7E-308 a 1,7E+308
long double	80	3,4E-4932 a 3,4E+4932

Figura 80 - https://slideplayer.com.br/slide/12134839/

CONVERSANDO COM O ARDUINO – serial monitor

Vamos aprofundar nossos estudos e conhecer mais um recurso sensacional oferecido pela IDE do arduino: **o monitor serial**. Seria monitor nada mais é do que uma ferramenta que permite a visualização do valor de uma variável, ou visualização de frases, através de um monitor. Esse recurso nos permite ver em tempo real os valores que estão sendo armazenados em uma determinada variável.

Vamos abrir um exemplo: vá em *Arquivo -> Exemplos -> Basics -> DigitalReadSerial.*

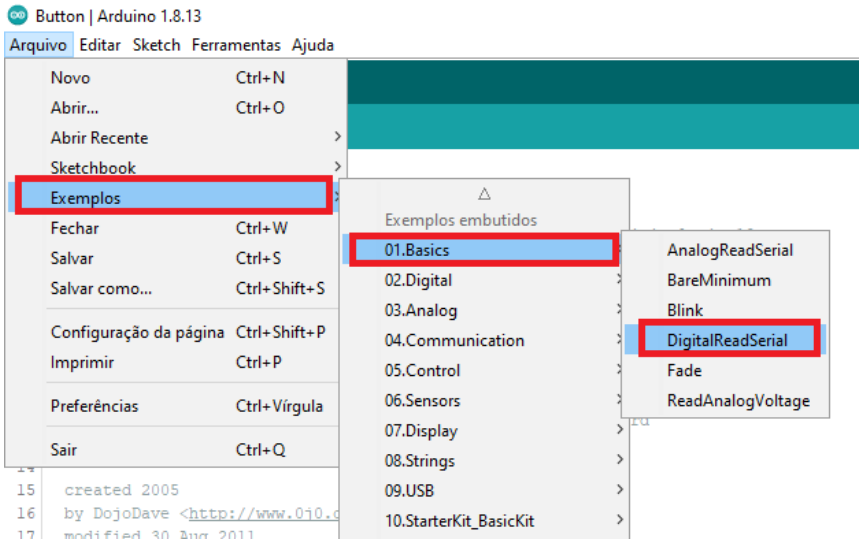


Figura 81 - DigitalReadSerial

```
// digital pin 2 has a pushbutton attached to it. Give it a name:
int pushButton = 2;

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
  // make the pushbutton's pin an input:
  pinMode(pushButton, INPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input pin:
  int buttonState = digitalRead(pushButton);
  // print out the state of the button:
  Serial.println(buttonState);
  delay(1);        // delay in between reads for stability
}
```

Esse código irá ler o botão 2 com a função digitalRead() e irá armazenar o valor lido dentro da variável buttonState. Em seguida o valor armazenado dentro da variável buttonState será mandada para o serial monitor por meio da função Serial.println.

Carregue o código no seu Arduino e clique na lupa no canto direito da tela. Ao ativar ou desativar o botão s2 será possível ver os valores digitais 0 e 1 na janela do serial monitor. Veja:

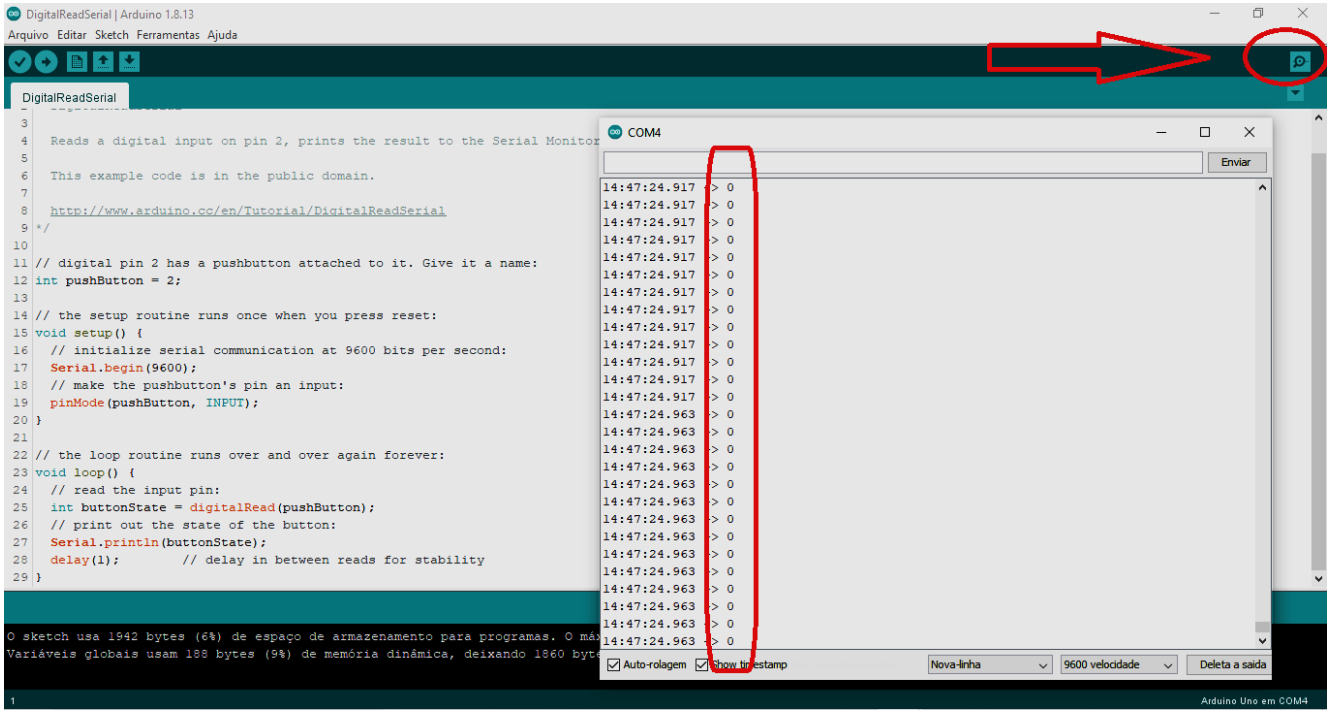


Figura 82 - Serial Monitor

COMO UTILIZAR O MONITOR SERIAL

Para utilizar o serial monitor será preciso fazer a configuração da velocidade de comunicação entre a placa arduino e o seu computador. Para fazer essa configuração utilizaremos a função **Serial.begin();** dentro da função setup(); e utilizaremos a velocidade de 9600bps (bits por segundo)

```
void setup()
{
    //inicializando a comunicação serial em 9600 bits por segundo
    Serial.begin(9600);
}
```

Após configurar a velocidade de comunicação, você deverá utilizar a função **Serial.println();** para fazer alguma impressão no monitor. Dentro dos parênteses da função Serial.println deve ser colocada a *frase (entre aspas duplas)* ou a variável que irá conter o valor a ser impresso.

```
void loop()
{
  int buttonState = digitalRead(pushButton);

  Serial.println(buttonState);

  delay(1);
}
```

Figura 83 - Utilização do serial monitor para visualizar o valor que está sendo alocado dentro da variável buttonState

A velocidade de comunicação entre a placa do arduino e o computador também é conhecida como BAUD RATE

AVISO IMPORTANTE

QUAL A DIFERENÇA ENTRE Serial.print ou Serial.println

Tanto a função **Serial.print** como a **Serial.println** servem para imprimir no serial monitor uma frase ou valor de uma determinada variável, mas a diferença entre uma e outra está no fato de se pular uma linha ou não após uma impressão.

A **função Serial.print();** não pula linhas após imprimir um resultado.

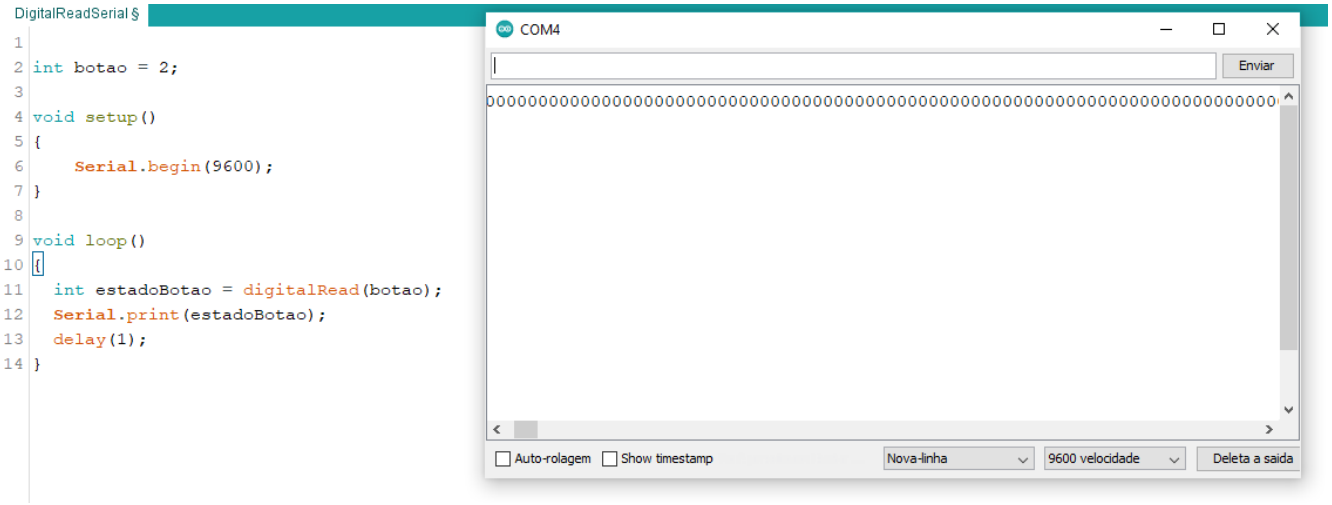


Figura 84 -Resultado do estado do botão sendo apresentado em uma única linha no serial monitor

A função **Serial.println();** fará a cada impressão um salto de linha.

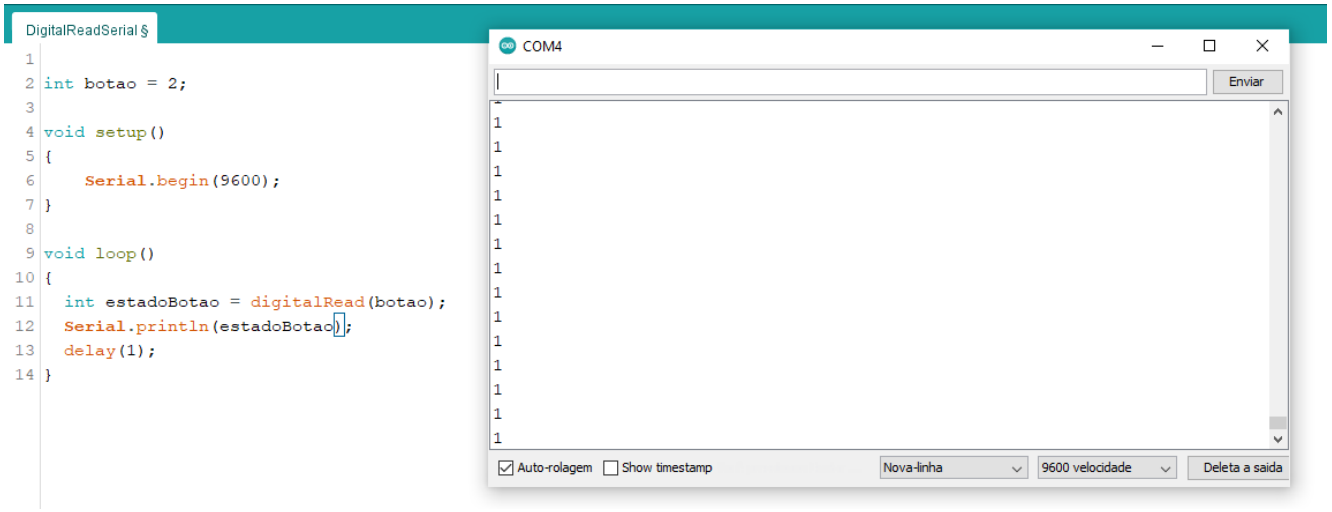


Figura 85 - Função Serial.println(); saltando uma linha a cada impressão

Portanto, a escolha de utilizar a função Serial.print ou Serial.println dependerá se o programador deseja ou não pular uma linha após a impressão.

IMPRIMINDO UMA STRING NO SERIAL MONITOR

Para imprimir uma frase (string) utilizando a função Serial.print(); basta colocar a frase desejada dentro dos parênteses entre aspas duplas. Veja o exemplo abaixo:

```
Serial.print("O estado do botao eh:");
```

Veja a união entre duas funções de impressão, uma sem pular linha e outra pulando, formando uma única frase.

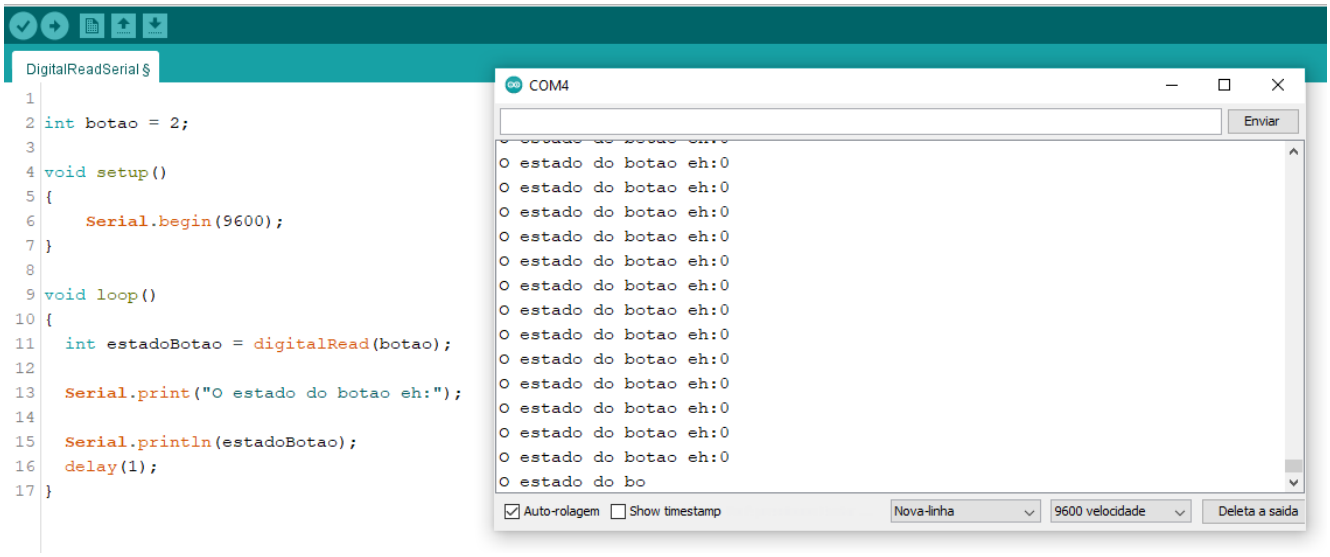
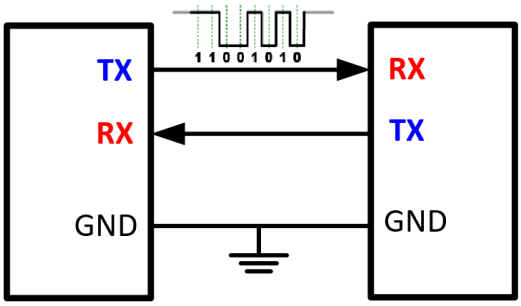


Figura 86 - Primeiro é impresso a frase sem pular linha, na sequência é impresso o valor do botão, e ai sim pula uma linha, formando uma única frase por linha.

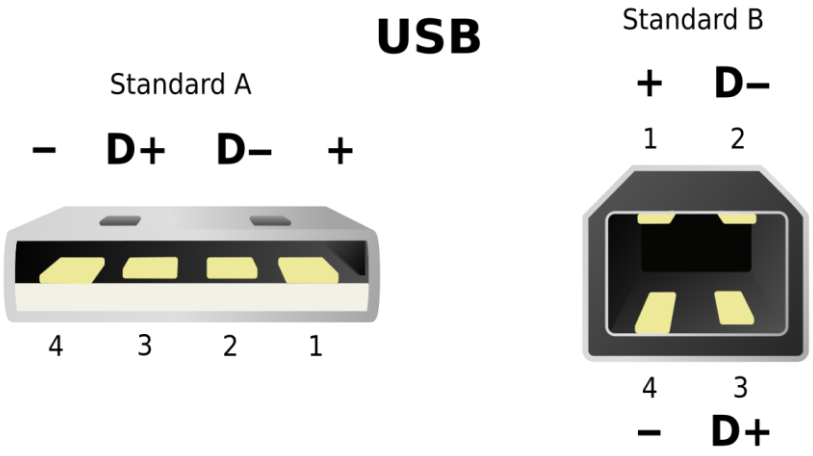
IMERSÃO NA COMUNICAÇÃO SERIAL

Vamos entender como acontece a comunicação serial entre o computador e a placa do Arduino.

A comunicação se chama **serial** pois os valores que trafegam entre os dois dispositivos são enviados um após o outro, em série, formando um pacote de envio. A transmissão de dados é feita pelo pino TX, e o recebimento de dados pelo pino RX.



A comunicação elétrica entre os dois dispositivos, arduino e computador, é feita por meio de um cabo USB, que possui tipicamente 4 vias. Veja abaixo:



+ Alimentação positiva

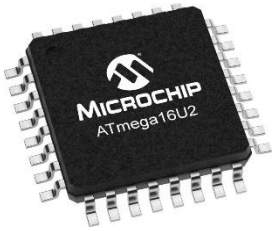
- Alimentação negativa

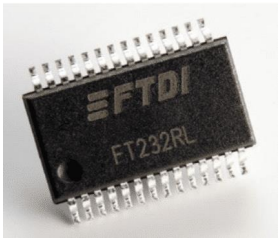
D+ Malha para Envio de dados

D- Malha para Recebimento de dados

Vimos que a comunicação serial acontece de forma seriada, ou seja, um valor é enviado após o outro, e que essa transmissão acontece por meio do cabo USB. Porém, ***o ATMega328p não consegue se comunicar diretamente com a porta USB*** para fazer o envio e recebimento de dados, pois seus protocolos de comunicação são diferentes.

Para que essa comunicação seja estabelecida de forma correta entre o arduino e o computador, *falando* a mesma língua, é necessário a utilização de uma **INTERFACE** de comunicação entre os dois dispositivos. Os dispositivos de conversão USB serial mais utilizados no Arduino são: *ATmega16u2, FTDI232 e CH340G.*





Esses circuitos integrados serão responsáveis por converter o protocolo de comunicação ***USB em UART*** e vice versa, permitindo que o arduino se comunique com o computador.

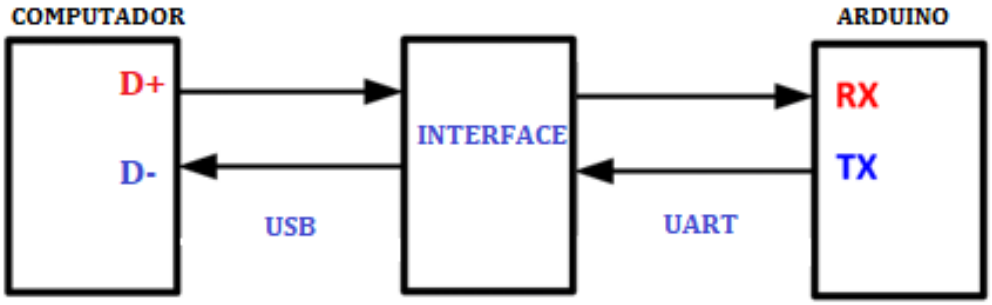


Figura 87 - Comunicação entre arduino e computador por meio de uma interface

DESBRAVANDO VARIÁVEIS DO TIPO INTEIRO – INT

Já vimos anteriormente que as variáveis possuem tipos, ou seja, elas possuem características de armazenamento e essas características estão diretamente ligadas ao tipo de dado que ela armazenará. Uma variável do tipo inteira – **int** – irá armazenar números inteiros.

DEFINIÇÃO

Números inteiros: *Um número inteiro é um número positivo ou negativo que pode ser escrito sem um componente fracional. Por exemplo, 21, 4, 0, e −2048 são números inteiros.*

Pela definição acima já sabemos o tipo de dado que será armazenado, mas qual será o tamanho desse espaço reservado para esse armazenamento de números inteiros?

*No Arduino uno e outras placas baseadas na arquitetura do Atmega, uma variável do tipo inteira irá armazenar **16 bits**, ou seja, **2 Bytes**. Esse armazenamento garante um intervalo de -32.768 até 32.767.*

Para declarar uma variável do tipo inteiro basta adicionar o **prefixo int** antes do nome.

```
int variavel;
```

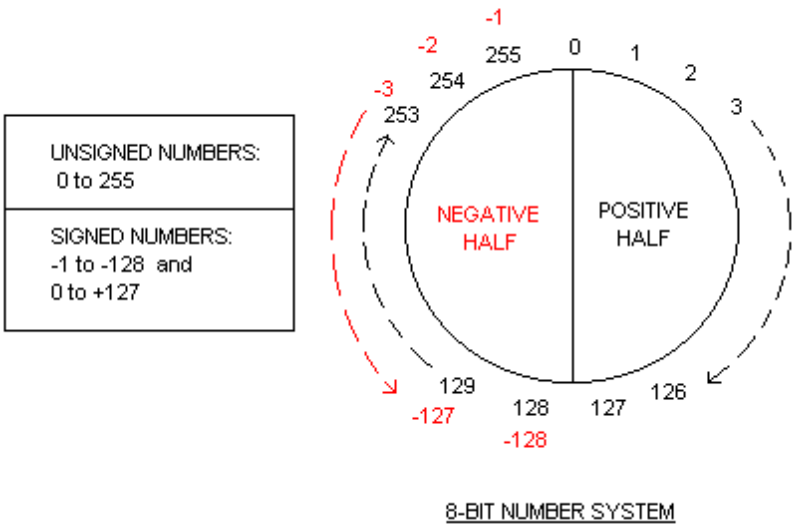
Mas, o que será que acontece quando tentamos armazenar um valor maior que o espaço reservado na memória para um determinado tipo de dado?

Quando há a tentativa de armazenar um valor maior que o espaço reservado na memória, acontece o chamado ***ESTOURO*** *da variável.*

Tomando por base uma variável do tipo int, sabemos que ela pode armazenar valores que vão de ***-32.768 até 32.767***. Ao tentar armazenar um valor de 32.768 positivo, por exemplo, não será possível, ela irá estourar o seu limite de 32.767 e irá passar a contagem de armazenamento de 1 unidade estourada para o início dos números negativos disponíveis, armazenando -32.768 ao invés de 32.767.

Se tentarmos armazenar 32.769, ela irá estourar em duas unidades, armazenando dessa forma o número -32.767. Ou seja, uma variável quando estoura acaba armazenando valores indesejados e isso pode comprometer todo o funcionamento do programa.

Para facilitar o entendimento vamos pensar em uma variável do **tipo char**, que armazena apenas **1 byte,** ou seja **256 (0 ao 255)** opções de valores.

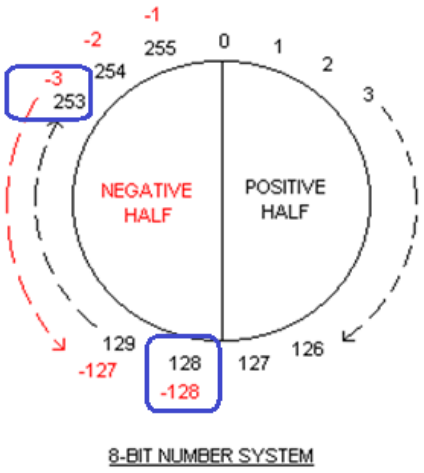


8-BIT NUMBER SYSTEM

As variáveis podem ser configuradas para armazenarem somente números positivos ***(unsigned)*** *ou números positivos e negativos* ***(signed).***

Quando uma variável é ***unsigned*** *ela dedicará* ***toda sua capacidade de armazenamento para números positivos****, no exemplo de uma variável de 1 byte, ela armazenará* ***256 valores*** *positivos.*

*Quando uma variável é **signed**, ela **dividirá seu armazenamento entre números positivos e negativos**, no exemplo anterior de 1 byte, será **127 valores positivos e -128 negativos.***



8-BIT NUMBER SYSTEM

Voltando ao **ESTOURO DA VARIÁVEL**, se observarmos o círculo da imagem ao lado veremos exatamente o momento em que a variável pode estourar e ir para os números negativos. Se tentarmos armazenar o valor **128** em uma **variável signed** de 1 byte (armazena positivos e negativos), iremos estourar do limite máximo de **127** que ela possui, que avançará para o número -128, onde se encerram os negativos. Se tentarmos armazenar o **valor 253**, iremos estourar o limite e o valor armazenado será **valor -3.**

INCREMENTAR UMA VARIAVEL – OPERADORES ++ E --

Quando vamos implementar um código que faça a contagem de algum produto, seja de uma produção de chocolates, ou de latas de refrigerante, ou de palitos de churrasco, enfim, podemos utilizar o recurso de contagem que incrementa um valor a uma variável a cada novo objeto contado.

Quando se quer contar algo basta adicionar UMA UNIDADE na contagem a cada novo item contado. Podemos aplicar essa lógica em uma variável utilizando o seguinte recurso: **incremento de uma variável.**

Para utilizar esse recurso o primeiro passo será declarar uma variável com um determinado tipo. Vamos determinar que iremos contar até 100, ou seja, vamos armazenar números inteiros. Poderemos tranquilamente utilizar o tipo int.

```
int contador;
```

Agora vamos atribuir um valor inicial para a variável, ou seja, o valor que ela terá no início da contagem.

```
int contador = 0;
```

Agora, para testar, podemos utilizar a função loop (que se repete infinitamente) para que cada vez que ela for executada, a variável contadora será incrementada em 1 unidade.

```
void loop()
{
  contador = contador + 1;
}
```

ATENÇÃO – O símbolo de = não significa IGUAL, mas RECEBE (atribuição).

Analisando a lógica do código anterior, temos que a variável *contador* **RECEBE** a soma do valor da variável *contador* mais *uma unidade*. Se *contador* iniciou em 0, *contador* receberá na primeira execução 0 + 1;

Quando iniciar a segunda execução, contador irá valer uma unidade, dessa forma teremos:

Contador RECEBE 1 + 1;

Após essa execução contador valerá 2, e assim por diante.

Podemos ainda simplificar essa atribuição de uma unidade, escrevendo da seguinte forma:

```
void loop()
{
  contador++;
}
```

Da mesma forma que podemos incrementar 1 unidade a um contador, podemos também fazer o **decremento**, ou seja **reduzir** uma unidade. Basta utilizar o símbolo negativo.

```
void loop()
{
  contador = contador - 1;
}
```

Ou então:

```
void loop()
{
  contador--;
}
```

Podemos ainda fazer o incremento de duas unidades por vez:

```
void loop()
{
  contador = contador + 2;
}
```

Ou então:

```
void loop()
{
  contador =+ 2;
}
```

CLIQUE AQUI PARA ACESSAR A AULA AULA

MEMÓRIA RAM E MEMORIA FLASH

Falamos anteriormente que uma variável é uma reserva de memória para que sejam armazenados dados, e esse espaço será reservado dentro de uma memória de dados de acesso aleatório, que é a chamada **memória RAM (Random Acess Memory).**

Memória RAM é uma memória do tipo **VOLÁTIL**, ou seja, todas as vezes que o sistema é desenergizado ela perde os dados armazenados e libera os espaços reservados.

A memória **FLASH** é um tipo de memória EEPROM (Electrically-Erasable Programmable Read-Only Memory), onde será salvo o programa que será desenvolvido. A memória **FLASH é do tipo NÃO VOLATIL**, ou seja, mesmo que o sistema seja desligado, os dados armazenados não serão perdidos.

Portanto, as memórias RAM salvam valores das variáveis enquanto o programa está rodando, e esses dados serão perdidos assim que o sistema for desligado. A memória FLASH irá armazenar a lógica desenvolvida pelo programador, e mesmo que o sistema seja desligado todos os dados continuarão armazenados.

CLIQUE AQUI PARA ACESSAR A AULA

INT, CONST INT E OCUPAÇÃO DE MEMÓRIA RAM

Ao declarar uma variável do tipo inteira estamos reservando um espaço de 2 bytes na memória RAM com um determinado NOME. A essa variável podemos atribuir um valor. Por exemplo:

```
int pushButton = 2;
```

No exemplo anterior reservamos um espaço de 2 bytes na memória **(int)** para armazenar **valores inteiros,** e nomeamos de esse espaço de **pushButton.** Em seguida ATRIBUIMOS (=) o valor 2 a esse espaço de memória chamado de pushButton, ou seja, colocamos dentro desse espaço reservado o valor 2.

Agora a variável pushButton vale 2.

*Abaixo temos um espaço de 16 bits (2 bytes) reservado na memória para a **variável pushButton** armazenando o valor 2.*

0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Todas as vezes que quisermos alterar o valor armazenado nesse espaço de memória utilizamos a variável chamada de pushButton no código.

Agora vamos supor que queremos que um determinado valor armazenado nesse espaço NÃO SEJA ALTERADO durante a execução do programa, QUEREMOS QUE ELE SEJA CONSTANTE, bloqueado, o que fazer?

Para que um valor de uma variável não corra risco de ser, podemos atribuir o prefixo ***const*** antes do tipo da variável. Veja:

```
const int pushButton = 2;
```

Dessa forma, o valor 2 armazenado no espaço de memória chamado de pushButton **NÃO PODERÁ SER ALTERADO** durante a execução do programa. Se tentarmos atribuir qualquer valor a uma variável **const**, o compilador acusará um erro. Veja:

```
17 const int pushButton = 2;
18
19 void loop()
20 {
21   pushButton = 3;
22 }
```

```
assignment of read-only variable 'pushButton'
                  ^
exit status 1
assignment of read-only variable 'pushButton'
```

No exemplo, declaramos ***const int pushButton*** e atribuímos o valor 2. Dentro da função loop tentamos atribuir o valor 3 a essa variável e o compilador acusou um conflito, pois uma variável declarada como ***const*** não permite alterações do seu valor durante a execução do programa.

CLIQUE AQUI PARA ACESSAR A AULA

VARIÁVEIS GLOBAIS E LOCAIS NA PROGRAMAÇÃO

As variáveis podem ser declaradas de duas formas em código de programação: **global e local.**

Quando declaramos uma variável dentro de um escopo de uma função, essa variável passa a ser “visível” apenas dentro desse escopo, não sendo identificada fora dele. Ou seja, ela só poderá ser utilizada na função onde ela foi declarada. **Essa é a variável LOCAL.**

Quando declaramos uma variável antes e fora das funções, essa variável poderá ser identificada por todas as funções utilizadas no código, ou seja, qualquer função poderá utilizar essa variável. **Essa é a variável GLOBAL.**

```
int variavel_global;

void funcao1()
{
  int variavel_local_1;
}

void funcao2()
{
  int variavel_local_2;
}
```

Veja na imagem anterior os exemplos de variáveis locais e globais. A variável global é declarada **FORA das funções**, por isso é chamada de global e pode ser utilizada pelo código todo. As variáveis locais foram declaradas dentro das funções e só podem ser utilizadas naquele escopo, elas só existem dentro das funções que foram declaradas.

Por isso, se tentarmos utilizar a variável, *variável_local_1* dentro da *funcao2()* o programa não identificará essa variável e acusará um erro, pois essa variável só existe dentro da *funcao1().*

Já a *varivel_global*, por ter sido declarada fora das funções, poderá ser utilizada em qualquer uma das funções do código.

*A vantagem de se utilizar a **variável local** é que se pode economizar espaço de memória, uma vez que a variável local só existe quando a função é chamada, após o termino da execução dessa função, o espaço reservado pela variável local é liberado. A variável global armazena um espaço de memória constante e durante todo o funcionamento do código, ou seja, aquele espaço de memória ficará permanentemente alocado para ela.*

CLIQUE AQUI PARA ACESSAR A AULA

DESBRAVANDO O DIGITALREAD(pino)

A função digitalRead() é utilizada para fazer a leitura de valores digitais de um dispositivo de entrada, ou seja, de um pino especifico do Arduino.

ATENÇÃO – o nome da função é a soma de duas palavras, digital (0 e 1) e Read (leitura). A função é escrita com a letra R maiúscula, digitalRead();

Para utilizar a função digitalRead é muito simples. Basta identificar dentro dos parênteses o pino que será feita a leitura do sinal digital.

digitalRead (8); // função lê estado digital do pino 8

Após fazer a leitura do pino, a função irá retornar o valor lido nesse pino, e esse valor deverá ser armazenado em uma variável.

int estado_botao = digitalRead (8);

Veja no exemplo que a variável *estado_botao* foi utilizada para receber o valor lido no pino 8 por digitalRead. Dessa forma, estado_ botão terá o valor lido no pino 8.

CLIQUE AQUI PARA ACESSAR A AULA

INPUTS UTILIZANDO SOFTWARE PROTEUS

Para testar os conhecimentos aprendidos anteriormente, vamos criar um projeto no software proteus e vamos aplicar os conceitos na simulação com a placa arduino UNO. Antes de desenharmos o projeto, vamos no site https://www.arduino.cc/reference/en/language/functions/digital-io/digitalread/ e vamos utilizar o código exemplo disponível.

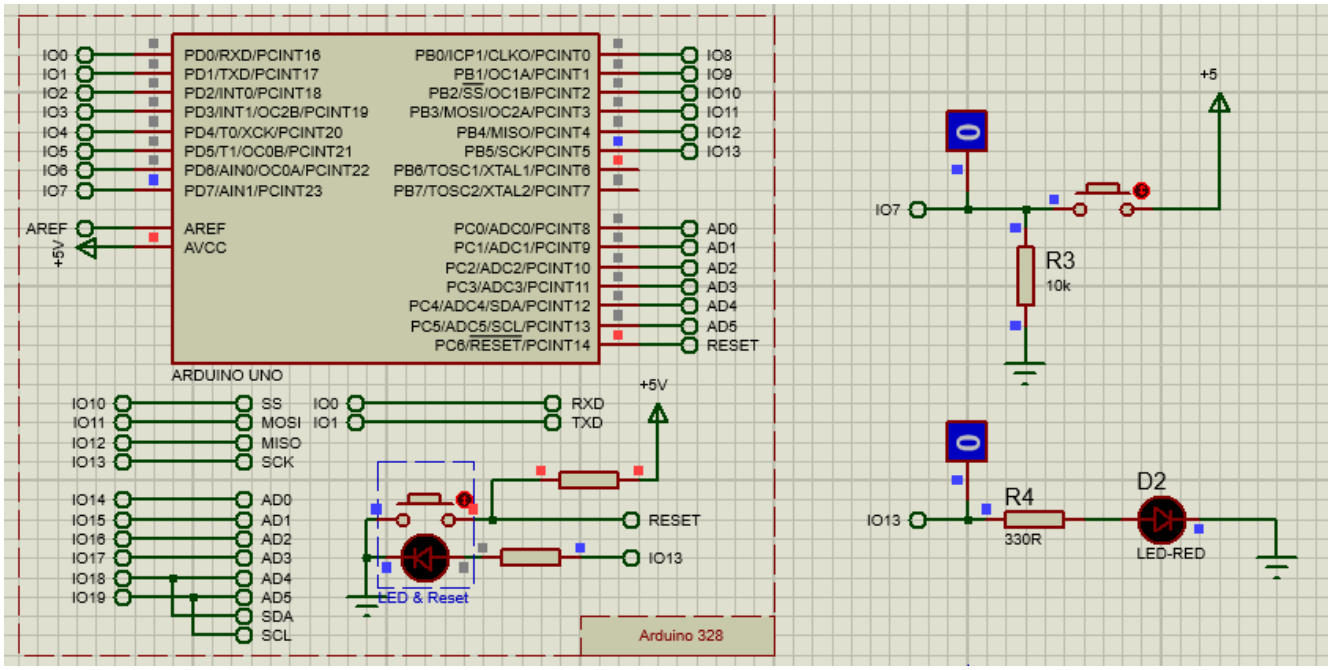
A grande vantagem de simular o projeto com o proteus é que podemos implementar vários periféricos, como resistores, botão, led, sem a necessidade de tê-los disponíveis fisicamente nesta etapa de aprendizado.

Example Code

Sets pin 13 to the same value as pin 7, declared as an input.

```
int ledPin = 13;  // LED connected to digital pin 13
int inPin = 7;    // pushbutton connected to digital pin 7
int val = 0;      // variable to store the read value

void setup() {
  pinMode(ledPin, OUTPUT);  // sets the digital pin 13 as output
  pinMode(inPin, INPUT);    // sets the digital pin 7 as input
}

void loop() {
  val = digitalRead(inPin);   // read the input pin
  digitalWrite(ledPin, val);  // sets the LED to the button's value
}
```

Figura 88 - código exemplo para digitalRead();



Para conhecer cada etapa da montagem do projeto no proteus acesse a aula por meio do seguinte link. https://ead.cursoseletronicafacil.com.br/lesson/detail/3/216/

ESTA REGRA VOCÊ UTILIZARÁ PELO RESTO DA VIDA

Sempre que estivermos trabalhando com leitura de entradas analógicas ou digitais no código de um programa, ***HAVERÁ A NECESSIDADE DE SE CRIAR UMA VARIÁVEL PARA ARMAZENAR CADA VALOR DE ENTRADA LIDO.***

Dessa forma, se o programa tiver que fazer a leitura de 5 entradas digitais, necessitaremos de 5 variáveis para armazenar separadamente cada valor lido nas entradas.

Veja no código abaixo que para cada botão que será feito a leitura foi criado uma variável pra armazenar o seu valor, para o pino 3 nomeado como BOTÃO1 teremos a variável VALOR_B1 para armazenar o seu valor. Para o pino 2 nomeado como BOTAO2, teremos a variável VALOR_B2 para armazenar o seu valor.

```
int BOTAO1 = 3;   //define BOTAO1 como pino 3
int BOTAO2 = 2;   //define BOTAO2 como pino 2
int VALOR_B1 = 0; //variável para armazenar valor do BOTÃO1
int VALOR_B2 = 0; //variável para armazenar valor do BOTAO2

void setup()
{
  Serial.begin(9600);     //inicia comunicação serial em 9600bps
  pinMode(BOTAO1, INPUT); //configura BOTAO1 como entrada
  pinMode(BOTAO2, INPUT); //configura BOTAO2 como entrada
}
void loop()
{
  VALOR_B1 = digitalRead(BOTAO1); //valor lido em BOTAO1 é atribuido à VALOR_B1
  VALOR_B2 = digitalRead(BOTAO2); //valor lido em BOTAO1 é atribuido à VALOR_B1
}
```

Figura 89 - digitalRead atribuindo valor a uma variável

Analisando cada etapa do código acima temos as seguintes partes:

- ✓ Declaração das variáveis

```
int BOTAO1 = 3;   //define BOTAO1 como pino 3
int BOTAO2 = 2;   //define BOTAO2 como pino 2
int VALOR_B1 = 0; //variável para armazenar valor do BOTÃO1
int VALOR_B2 = 0; //variável para armazenar valor do BOTAO2
```

- ✓ Configuração das entradas e inicialização da comunicação serial

```
void setup()
{
  Serial.begin(9600);     //inicia comunicação serial em 9600bps
  pinMode(BOTAO1, INPUT); //configura BOTAO1 como entrada
  pinMode(BOTAO2, INPUT); //configura BOTAO2 como entrada
}
```

- ✓ Atribuição dos valores lidos nos botões para uma variável

```
void loop()
{
  VALOR_B1 = digitalRead(BOTAO1); //valor lido em BOTAO1 é atribuido à VALOR_B1
  VALOR_B2 = digitalRead(BOTAO2); //valor lido em BOTAO1 é atribuido à VALOR_B1
}
```

Agora vamos implementar a visualização desses valores, lidos e armazenados nas variáveis, por meio do serial monitor.

```
void loop()
{
  VALOR_B1 = digitalRead(BOTAO1); //valor lido em BOTAO1 é atribuido à VALOR_B1
  VALOR_B2 = digitalRead(BOTAO2); //valor lido em BOTAO1 é atribuido à VALOR_B1

   Serial.print("O valor lido no BOTÃO 1 eh: ");
   Serial.println(VALOR_B1);
   Serial.print("O valor lido no BOTÃO 2 eh: ");
   Serial.println(VALOR_B2);

   delay (1000);
}
```

Podemos verificar que foram utilizadas as 2 funções de impressão no serial monitor, uma que não pula linha e outra que pula. Isso foi feito para que a frase e o valor fossem impressos na mesma linha. Veja, primeiro é impresso "*O valor lido no BOTÃO 1 eh: "* sem pular linha, assim a próxima impressão do *valor* será feita na sequência da frase anterior, e então, somente após esse valor impresso, a função faz o pulo de uma linha.

A execução desse loop será feita a cada 1000 milésimos, ou 1 segundo, por conta da função delay (1000);

CRIANDO UMA FUNÇÃO PARA LEITURA DE ENTRADAS

Uma forma de deixar o seu código ainda mais legível e bem organizado, é separar em funções cada uma das etapas do código desenvolvido dentro do loop. Então, se o código fará a leitura de entradas digitais por exemplo, podemos criar uma função FORA do loop somente para este fim e chama-la dentro do loop.

Veja, uma função que irá fazer apenas leitura de entradas não precisa retornar nenhum valor, **iremos então coloca-la como void,** e a chamaremos de **LEITURA_ENTRADAS.**

```
void LEITURA_ENTRADAS()
{
  VALOR_B1 = digitalRead(BOTAO1); //valor lido em BOTAO1 é atribuido à VALOR_B1
  VALOR_B2 = digitalRead(BOTAO2); //valor lido em BOTAO1 é atribuido à VALOR_B1
}
```

Agora, dentro do loop, onde estava o código de leitura das entradas, vamos chamar a função LEITURA_ENTRADAS() para que seja feita a leitura.

```
void loop()
{

   LEITURA_ENTRADAS();

   Serial.print("O valor lido no BOTÃO 1 eh: ");
   Serial.println(VALOR_B1);
   Serial.print("O valor lido no BOTÃO 2 eh: ");
   Serial.println(VALOR_B2);

   delay (1000);
}
```

Podemos ainda criar uma outra função somente para impressão dos valores lidos.

```
void IMPRIME_VALORES()
{
   Serial.print("O valor lido no BOTÃO 1 eh: ");
   Serial.println(VALOR_B1);
   Serial.print("O valor lido no BOTÃO 2 eh: ");
   Serial.println(VALOR_B2);
}
```

Agora vamos chamar essa função dentro do loop após a leitura dos valores de entrada.

```
void loop()
{
   LEITURA_ENTRADAS();
   IMPRIME_VALORES();
   delay (1000);
}
```

Veja como a organização do loop ficou muito melhor, de imediato entendemos que o programa faz a leitura de entrada e na sequência imprime seus valores lidos. Então, é uma excelente prática organizar em funções o código do seu projeto.

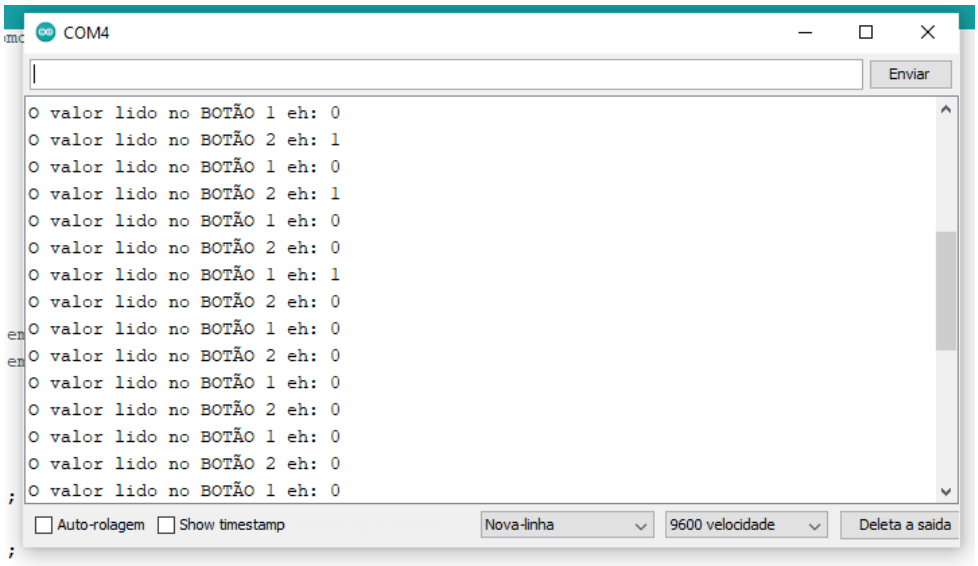
Dessa forma encerramos a imersão de leituras de entradas digitais e, a partir de agora, estamos prontos para desenvolver projetos com esse recurso. Parabéns por ter chego até aqui sem o vício do copiar e colar, mas com a responsabilidade de aprender e entender cada parte do código que está sendo implementado.

Veja como ficou o projeto total:

```
int BOTAO1 = 3;   //define BOTAO1 como pino 3
int BOTAO2 = 2;   //define BOTAO2 como pino 2
int VALOR_B1 = 0; //variável para armazenar valor do BOTÃO1
int VALOR_B2 = 0; //variável para armazenar valor do BOTAO2

void setup()
{
  Serial.begin(9600);     //inicia comunicação serial em 9600bps
  pinMode(BOTAO1, INPUT); //configura BOTAO1 como entrada
  pinMode(BOTAO2, INPUT); //configura BOTAO2 como entrada
}
void loop()
{
   LEITURA_ENTRADAS();
   IMPRIME_VALORES();
   delay (1000);
}

void LEITURA_ENTRADAS()
{
  VALOR_B1 = digitalRead(BOTAO1); //valor lido em BOTAO1 é atribuido à VALOR_B1
  VALOR_B2 = digitalRead(BOTAO2); //valor lido em BOTAO1 é atribuido à VALOR_B1
}
```

```
void IMPRIME_VALORES()
{
   Serial.print("O valor lido no BOTÃO 1 eh: ");
   Serial.println(VALOR_B1);
   Serial.print("O valor lido no BOTÃO 2 eh: ");
   Serial.println(VALOR_B2);
}
```

Vejam a impressão no serial monitor:



VIRTUAL TERMINAL NO PROTEUS

Da mesma forma que conseguimos visualizar os valores lidos no arduino por meio do recurso do **serial monitor** disponível na IDE do Arduino, também conseguiremos visualizar os valores lidos na simulação do arduino no software proteus. Para isso utilizaremos a ferramenta **VIRTUAL TERMINAL**.

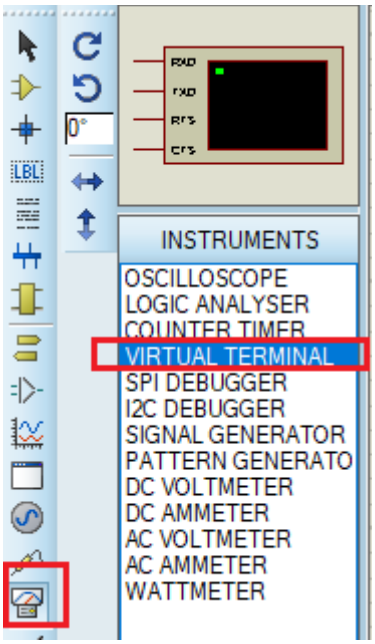
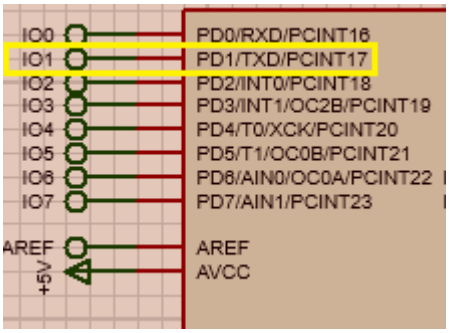
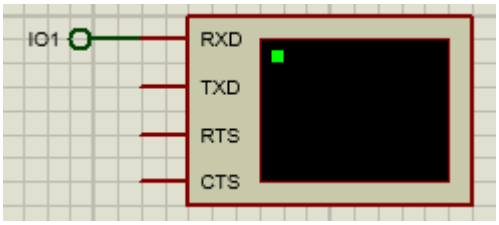


Figura 90 - Virtual Terminal

Adicione o *virtual terminal* ao seu projeto e em seu pino de **recebimento de informações (RX)** deveremos conectar o pino de **transmissão de dados do arduino (TX),** localizado no IO1.





Agora, para que os valores sejam impressos no virtual terminal, será preciso inicializar a comunicação serial com Serial.begin(9600) dentro do setup, e colocar as funções de impressão Serial.print() dentro do loop, exatamente como foi feito na IDE do arduino. Esse será o resultado:

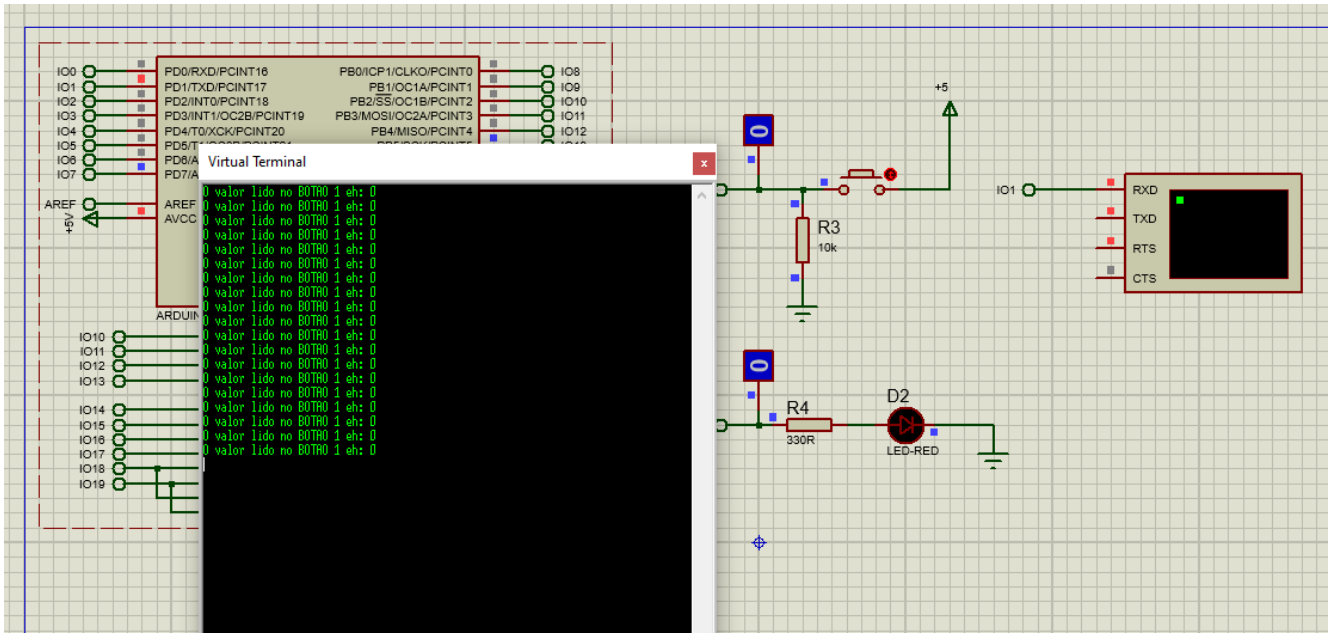


Figura 91 - Virtual Terminal

Podemos também adicionar um osciloscópio e conectá-lo ao pino IO1 para visualizar os pulsos digitais 0 e 1 enviados pelo arduino.

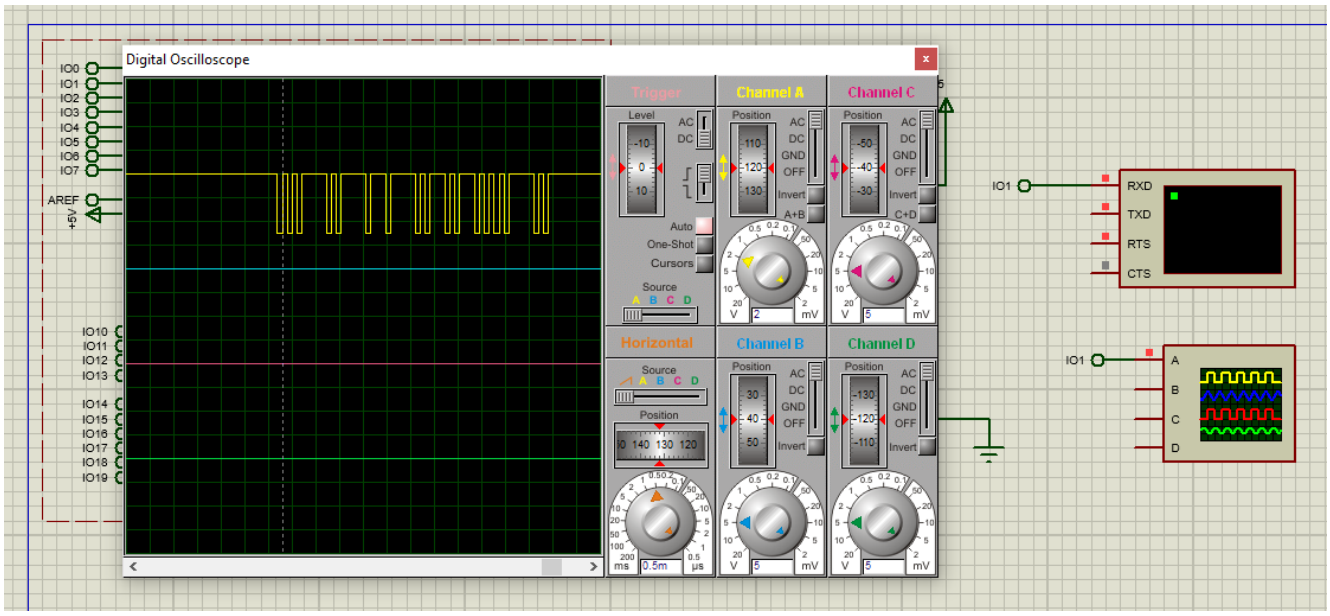


Figura 92 - Osciloscópio lendo pulsos digitais

Agora, depois de todo esse conteúdo espetacular, vamos deixar um grande desafio para você. Queremos com isso treinar seus conhecimentos e principalmente colocar em prática tudo que foi aprendido anteriormente. E aí, vamos pra cima?

DESAFIO NO PROTEUS – Leds indicadores de estado do botão

Programar o arduido para que leia 4 botões, e cada vez que um dessses botões for acionado, um respectivo led deverá ser aceso, ao soltar o botão, o led deverá ser apagado. Será um led para cada botão. Então, os leds indicarão o nível lógicos dos botões, aceso indicará nível lógico 1, apagado nível lógico 0. A escolha dos pinos ficará por sua conta. Você também deverá adicionar o virtual terminal ao seu projeto, que deverá imprimir o nível lógico de cada botão, com a seguinte frase "O VALOR LOGICO DO BOTAO1 EH:...." Isso deverá ser feito para os 4 botões. Grave um vídeo e poste no nosso grupo vip para que possamos ver o seu desenvolvimento e vibramos juntos com sua vitória.

INSTRUÇÕES

- ✓ Utilizar 4 botões
- ✓ Fazer a leitura de 4 entradas digitais
- ✓ Utilizar 4 LEDS
- ✓ Iniciar comunicação serial (9600)
- ✓ Utilizar funções de impressão
- ✓ Imprimir o valor dos botões no virtual terminal

IF E ELSE – VOCÊ VAI USAR O RESTO DA VIDA

Vamos iniciar esse conteúdo imaginando a seguinte ocasião: *suponhamos que você tenha um compromisso hoje à noite e o tempo está muito instável, e então você pensa: se fizer calor vou de blusa regata, senão vou de casaco de frio.*

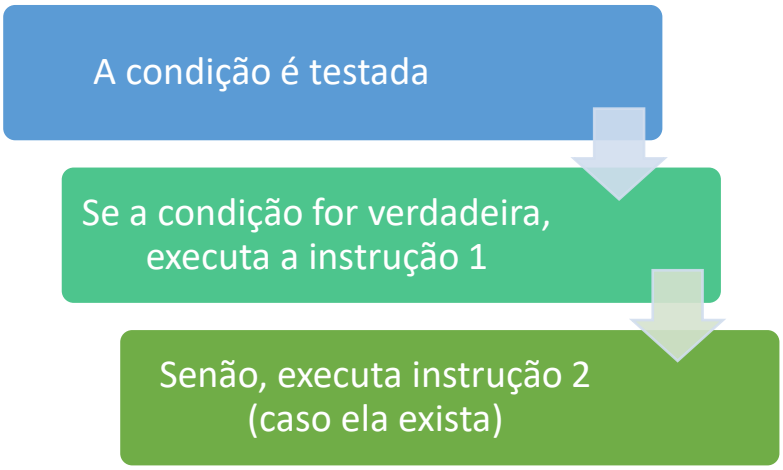
Veja que você colocou uma condição para uma ação futura, **SE** algo acontecer, faço isso, **SENÃO** faço aquilo.

```
se (fizer calor a noite)
{
  usar regata;
}
senão
{
  usar casaco de frio;
}
```

Em programação essa estrutura de controle é chamada de **IF e ELSE**. Colocando em linguagem de programação, teremos:

```
if (condição)
{
  instrução 1;
}
else
  {
    instrução 2;
  }
```

A instrução **if-else** funciona da seguinte maneira:



Após entendermos como funciona a estrutura de controle if-else, vamos partir para o código exemplo disponível na IDE do arduino e fazer algumas alterações. O código se encontra neste link: https://docs.arduino.cc/built-in-examples/digital/Button

É muito importante que nesse ponto você tenha absorvido muito bem os conceitos ensinados até aqui, pois a cada etapa os conteúdos serão somados a outros que estão vindo, formando um aprendizado único, e qualquer lacuna poderá gerar frustrações desnecessárias. Por isso, você que é aluno, se alguma dúvida estiver presente aqui, já deixe na plataforma em forma de comentário para que nossos tutores possam te ajudar, tudo bem? Vamos seguir firme e fortes, e sem dúvidas. Vamos pra cima!

Código exemplo:

```
// constants won't change. They're used here to set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed. If it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

Para colocarmos em prática o que estudamos vamos fazer algumas alterações no código exemplo acima. Vamos alterar os nomes das duas variáveis buttonPin e ledPin. Chamaremos de BOTAO2 e LED2.

```
// constants won't change. They're used here to set pin numbers:
const int BOTAO2 = 2;     // the number of the pushbutton pin
const int LED2 =  13;      // the number of the LED pin
```

Vamos também alterar o pino de LED2 para pino 7.

```
// constants won't change. They're used here to set pin numbers:
const int BOTAO2 = 2;     // the number of the pushbutton pin
const int LED2 =  7;      // the number of the LED pin
```

Também iremos alterar o nome da variável *buttonState* que irá armazenar o valor de leitura de botão. Vamos chama-la de VALOR_B2;

```
// variables will change:
int VALOR_B2 = 0;         // variable for reading the pushbutton status
```

Como nós alteramos os nomes das variáveis, vamos também alterar esses nomes dentro da função setup(), onde seus valores representarão os pinos que serão inicializados como entrada ou saída.

```
void setup() {
  // initialize the LED pin as an output:
  pinMode(LED2, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(BOTAO2, INPUT);
}
```

Agora vamos para o loop(), onde iremos colocar nossa condição de teste do botão utilizando a estrutura de controle if-else.

```
void loop()
{
  VALOR_B2 = digitalRead(BOTAO2);

  if (VALOR_B2 == HIGH)
  {
    digitalWrite(LED2, HIGH);
  }
  else
  {
    digitalWrite(LED2, LOW);
  }
}
```

ATENÇÃO – Em programação, quando queremos saber se alguma variável é IGUAL outra ou algum valor, utilizaremos dois símbolos de igualdade

==

Por exemplo:

if (valor == 10) *Se valor for igual a 10*

Vimos então que, podemos comparar dois valores para que uma decisão seja tomada. No exemplo acima, o valor do botão2 foi testado: se ele for igual a 1, ou estiver pressionado, o led será aceso. Se não estiver pressionado, led será apagado.

CLIQUE AQUI PARA ACESSAR A AULA

SITUAÇÃO PROBLEMA – INDICADOR VERTICAL DE TEMPERATURA COM LEDS

E aí, estão prontos para mais um desafio?

INDICADOR VERTICAL DE TEMPERATURA COM LEDS (BARGRAPH)

Você deverá executar um código de programação para o arduino que indique a temperatura através do acendimento de leds. Para cada grupo de valores de temperatura um led deverá ser aceso. Os leds que deverão ser acessos como indicador de temperatura são os leds disponíveis no nosso shield para arduíno.

INSTRUÇÕES

- ✓ Utilizar os 8 leds disponíveis no shield
- ✓ Configurar os pinos como output
- ✓ Estruturar os if-else para cada temperatura

°C	Led 1	Led 2	Led 3	Led 4	Led 5	Led 6	Led 7	Led 8
0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	1	0	0	0	0	0	0	0
10	1	0	0	0	0	0	0	0
11	1	1	0	0	0	0	0	0
20	1	1	0	0	0	0	0	0
21	1	1	1	0	0	0	0	0
30	1	1	1	0	0	0	0	0
31	1	1	1	1	0	0	0	0
40	1	1	1	1	0	0	0	0
41	1	1	1	1	1	0	0	0
50	1	1	1	1	1	0	0	0
51	1	1	1	1	1	1	0	0
60	1	1	1	1	1	1	0	0
61	1	1	1	1	1	1	1	0
70	1	1	1	1	1	1	1	0
71	1	1	1	1	1	1	1	1

Atenção: nesse projeto não iremos fazer a leitura de temperatura por meio de nenhum sensor, você deverá criar uma variável e atribuir a ela um valor de temperatura, que será testada pelas condições if-else e indicará a temperatura por meio dos leds do shield.

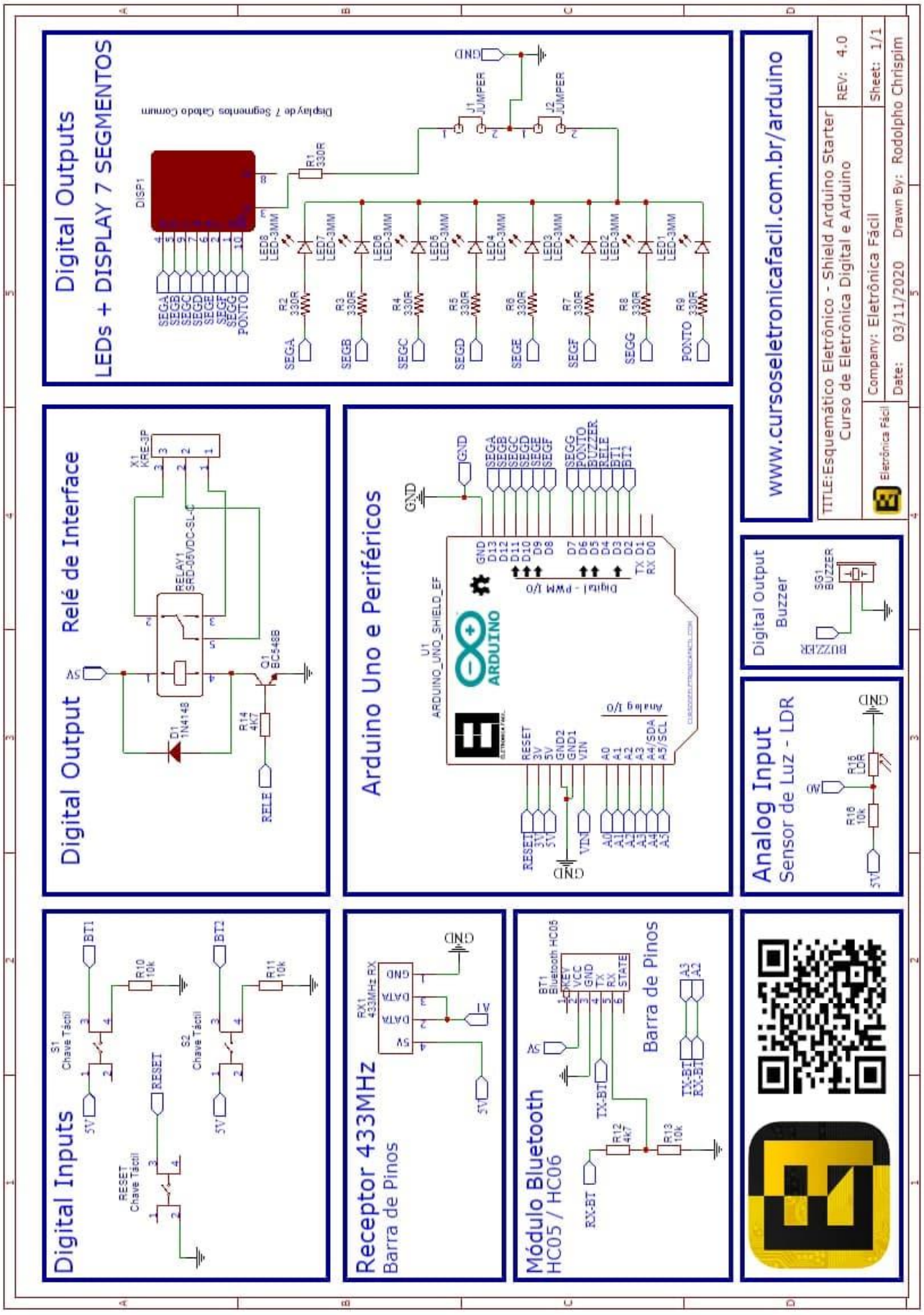


Figura 93 - esquema eletrônico SHIELD PARA ARDUINO ELETRÔNICA FÁCIL

Analisando o esquema eletrônico do shield podemos verificar que os pinos que fazem os acendimentos dos leds vão do **PONTO ao SEGA**. Esses pinos correspondem do **pino 6 ao 13**. Portanto, esses pinos deverão ser configurados como **saída digital** dentro da função setup.

Agora vamos interpretar essa tabela oferecida no desafio.

Existem diversas maneiras de interpretar a tabela com a lógica de acendimento de leds, e isso ficará a critério do programador, mas vamos ajudar você nesse primeiro momento a chegar em uma conclusão de funcionamento.

Podemos verificar que de 0 a 5 graus celsius todos os leds estarão apagados.

°C	*Led 1*	*Led 2*	*Led 3*	*Led 4*	*Led 5*	*Led 6*	*Led 7*	*Led 8*
0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

Podemos então dizer que:

SE temperatura for *menor que* 6, *desligar led1, led2, led3, led4, led5, led6, led7 e led8.*

E assim seguiremos para as outras temperaturas, comparando os limites de temperatura para fazer os acionamentos dos leds.

6	1							
10	1	0	0	0	0	0	0	0

SE temperatura for *maior que* 5 **E** *menor que* 11, *acionar led1*.

11	1	1	0	0	0	0	0	0
20	1	1	0	0	0	0	0	0

SE temperatura for *maior que* 10 **E** *menor que* 21, *acionar led1 e led2*.

21	1	1	1	0	0	0	0	0
30	1	1	1	0	0	0	0	0

SE temperatura for *maior que* 20 **E** *menor que* 31, *acionar led1, led2* e led3.

31	1	1	1	1				
40	1	1	1	1	0	0	0	0

SE temperatura for *maior que* 30 **E** *menor que* 41, *acionar led1, led2, led3 e led4*.

41	1	1	1	1	1			0
50	1	1	1	1	1	0	0	0

SE temperatura for *maior que* 40 **E** *menor que* 51, *acionar led1, led2, led3, led4 e led5*.

51	1	1	1	1	1	1		0
60	1	1	1	1	1	1	0	0

SE temperatura for *maior que* 50 **E** *menor que* 61, *acionar led1, led2, led3, led4, led5 e led6*.

61	1	1	1	1	1	1	1	
70	1	1	1	1	1	1	1	0

SE temperatura for *maior que* 60 **E** *menor que* 71, *acionar led1, led2, led3, led4, led5, led6 e led7*.

71	1	1	1	1	1	1	1	1

SE temperatura for *maior que* 70, *acionar led1, led2, led3, led4, led5, led6, led7 e led8.*

Agora chegou o momento de passar essas interpretações da tabela para a linguagem de programação, mas antes vamos disponibilizar alguns operadores utilizados em linguagem de programação que serão extremamente uteis na hora de transcrever essas interpretações para a lógica de programação.

SÍMBOLO	OPERADOR	APLICAÇÃO	SIGNIFICADO
>	*Maior que*	*x > y*	*X maior que y*
>=	*Maior ou igual*	*x >= y*	*X maior ou igual a y*
<	*Menor*	*x < y*	*X menor que y*
<=	*Menor ou igual*	*x <= y*	*X menor ou igual a y*
==	*igual*	*x == y*	*X igual a y*
!=	*Diferente*	*x != y*	*X diferente de y*
&&	*E*	*x > 0 && x < 2*	*X maior que 0 E x menor que 2*
||	*OU*	*x ==1 || x == 2*	*X igual a 1 OU x == a 2*

Tabela 1 - Operadores aritméticos e lógicos

Agora basta aplicar seus conhecimentos em utilizar as condições de controle if-else juntamente com os operadores disponibilizados na tabela anterior e montar a programação do desafio proposto. Vamos nessa?

Para conferir a resolução desse desafio acesse a aula na plataforma do curso.

CLIQUE AQUI PARA ACESSAR A AULA

CONHECENDO A FUNÇÃO RANDOM();

No código que desenvolvemos acima, como não tínhamos nenhum sensor fazendo a leitura da temperatura para que os leds indicadores acendessem, então criamos uma variável e fomos atribuindo um valor qualquer para que assim pudéssemos testar o código com diversas temperaturas e o funcionamento do indicador funcionasse com sucesso.

Há uma outra maneira bem interessante de se atribuir valores aleatórios à variável para testar o nosso programa, e para isso utilizaremos a função ***random();***

Para conhecer a função random podemos fazer uma pesquisa no próprio site do arduino por meio do link https://www.arduino.cc/reference/pt/language/functions/random-numbers/random/

random()

[Random Numbers]

Descrição

A função `random()` gera números pseudoaleatórios.

Sintaxe

```
random(max) random(min, max)
```

Parâmetros

`min` - menor limite do valor aleatório, inclusivo e opcional (long)

`max` - maior limite do valor aleatório, exclusivo (long)

Analisando os dados fornecidos pelo site, podemos verificar que a função random() recebe por parâmetros dois valores, **min** e **max, que serão os limites dos números gerados por essa função.**

> ***Min*** *= o menor valor que a função irá gerar*
>
> ***Max*** *= o maior valor que a função irá gerar*

Levando em consideração uma temperatura de 0º a 80°, passaremos esses dois valores como parâmetros para a função, que RETORNARÁ um número aleatório dentro desse intervalo. É importante separar esses dois valores por uma VIRGULA.

random (0, 80);

Já estudamos que quando uma função retorna um valor, precisamos de uma variável para armazenar esse valor retornado.

ATENÇÃO - A VARIÁVEL QUE RECEBERÁ ESSE VALOR DEVERÁ SER DO **TIPO LONG**

Vamos então declarar uma variável do tipo long chamada de numero_aleatorio:

```
long numero_aleatorio;
```

Agora vamos colocar a variável para receber o valor aleatório gerado pela função random:

```
numero_aleatorio = random(0, 80);
```

A partir desse momento a variável *numero_aleatorio* receberá um valor diferente cada vez que o loop for executado, no nosso caso, por exemplo, sendo utilizada para testar diferentes valores de temperatura.

Para visualizar na prática essa aplicação veja o vídeo por meio do link da aula ao lado.

CLIQUE AQUI PARA ACESSAR A AULA

ESTRUTURA IF – ELSE IF

O primeiro passo de um projeto é pensar em uma lógica que resolva um determinado problema, ou seja, fazer o código para que tudo funcione como esperado e o objetivo principal seja atingido. Depois do código base implementado, gradativamente podemos ir melhorando nosso projeto, incluindo ferramentas ou recursos que possam dar mais efetividade para o nosso projeto, pensando sempre que esses estudos iniciais um dia se tornarão projetos complexos e que melhorias serão inevitáveis.

Pensando nessa melhoria, vamos apresentar a **estrutura if -else if.**

No nosso projeto nós aprendemos a montar uma estrutura de teste das temperaturas aninhando os **IFs** um após o outro. Veja:

```
if (temperatura < 6) faça isso

if (temperatura > 5 && temperatura < 11) faça isso

if (temperatura > 10 && temperatura < 21) faça isso

if (temperatura > 20 && temperatura < 31) faça isso

....

else faça isso
```

Dessa forma, todos os testes serão executados um após o outro, mas vamos pensar juntos:

Observando o código acima, se a temperatura for 4ºC, a primeira condição de teste será verdadeira e será executada, fará sentido fazer todos os testes seguintes sabendo que eles NÃO SERÃO VERDADEIROS?

Pois é, não é preciso. Para que o programa pule todos os testes seguintes após uma condição verdadeira, podemos utilizar a **estrutura if -else if.**

SE (VERDADEIRO?) *FAÇA ISSO*

SENÃO SE (VERDADEIRO?) *FAÇA ISSO*

SENÃO *FAÇA ISSO*

if (temperatura < 6) *faça isso*

else if (temperatura > 5 && temperatura < 11) *faça isso*

else if (temperatura > 10 && temperatura < 21) *faça isso*

else if (temperatura > 20 && temperatura < 31) *faça isso*

e assim sucessivamente...

else *faça isso*

Dessa forma, assim que uma condição de teste verdadeira for encontrada, automaticamente o programa pula todos os testes seguintes economizando tempo de execução. Simples, não é mesmo?

Vamos supor que a temperatura testada seja 7ºC

if (temperatura < 6) *faça isso*

else if (temperatura > 5 && temperatura < 11) *faça isso*

~~**else if (temperatura > 10 && temperatura < 21)** *faça isso*~~

~~**else if (temperatura > 20 && temperatura < 31)** *faça isso*~~

~~*else faça isso*~~

e assim sucessivamente...

A primeira condição será testada, como 7º não é menor que 6, A CONDIÇÃO É FALSA e o programa passará para a segunda condição de teste.

A segunda condição será VERDADEIRA pois 7º é maior que 5 e menor que 11º, então o programa irá executar a instrução *FAÇA ISSO.*

Ao sair dessa condição testada como verdadeira, todas as outras condições de testes ***else if*** serão ignoradas, economizando tempo e agilizando a execução do nosso código.

CLIQUE AQUI PARA ACESSAR A AULA

SITUAÇÃO DE APRENDIZAGEM – ESTEIRA DE ACADEMIA

Neste momento vamos propor uma situação de aprendizagem. Se objetivo será fazer um programa que simule uma esteira de academia com botões que irão aumentar e diminuir a velocidade de caminhada e um display que irá mostrar a velocidade atual da esteira.

Mas como sempre, vamos dividir esse problema em algumas etapas para que tudo fique mais fácil e simplificado. Lembrem-se, o segredo é sempre dividir um problema grande em problemas menores.

ESTEIRA DE ACADEMIA – SITUAÇÃO DE APRENDIZAGEM 1

Neste primeiro momento você deverá criar um código de programação que irá mostrar inicialmente no display a velocidade de 0, e cada vez que o botão s1 for pressionado essa velocidade deverá subir em uma unidade até no máximo 9.

INSTRUÇÕES

- ✓ Botão S1 – aumentar velocidade
- ✓ Mostrar velocidade de 0 a 9 no display, iniciando 0
- ✓ Salve o desafio como ESTEIRA_PARTE1

ESTEIRA DE ACADEMIA – SITUAÇÃO DE APRENDIZAGEM 2

Agora você deverá incrementar no seu código anterior a função de diminuição de velocidade por meio do botão s2. Ou seja, a velocidade inicial da esteira será de 0, e cada vez que pressionar s1 a velocidade deverá ser incrementada em uma unidade, e cada vez que s2 for pressionado a velocidade deverá diminuir em uma unidade.

INSTRUÇÕES

- ✓ Botão S1 – aumentar velocidade
- ✓ Botão s2 – diminuir velocidade
- ✓ Mostrar velocidade de 0 a 9 no display, iniciando 0
- ✓ Salve o desafio como ESTEIRA_PARTE2

ESTEIRA DE ACADEMIA – SITUAÇÃO DE APRENDIZAGEM 3

Agora você deverá fazer com que seu código dê prioridade para o botão de diminuição de velocidade quando os dois botões, s1 e s2, forem pressionados juntos. Ou seja, sempre que os dois botões foram pressionados juntos, a velocidade deverá diminuir.

INSTRUÇÕES

- ✓ Botão S1 – aumentar velocidade
- ✓ Botão s2 – diminuir velocidade
- ✓ Mostrar velocidade de 0 a 9 no display, iniciando 0
- ✓ Ao pressionar os dois botões ao mesmo tempo, prioridade para diminuição
- ✓ Salve o desafio como ESTEIRA_PARTE3

FOCO NO PLANEJAMENTO – PORTUGUES ESTRUTURADO.

Ao chegar aqui, duas coisas podem ter acontecido com você ao tentar realizar os desafios propostos anteriormente: **ou você conseguir realizar, ou não conseguiu**.

Em qualquer um dos casos você já está de parabéns, pois se você conseguiu realizar os desafios, você está aqui continuando os estudos e empenhado na busca por sua evolução no fantástico mundo da programação e isso é motivo de muito orgulho.

Caso não tenha conseguido realizar os desafios anteriores, você não desanimou e continuou estudando, buscando entender o que te faltou para conseguir concretizar os objetivos esperados. Parabéns!

Neste momento VAMOS PLANEJAR em *português estruturado* a resolução dos desafios propostos anteriormente, com o simples objetivo de organizar tudo que precisa ser feito para que as situações de aprendizagens anteriores sejam resolvidas da melhor forma possível.

> *Português estruturado* é uma forma simplificada de estruturar ideias ou uma sequência lógica utilizando a língua portuguesa.

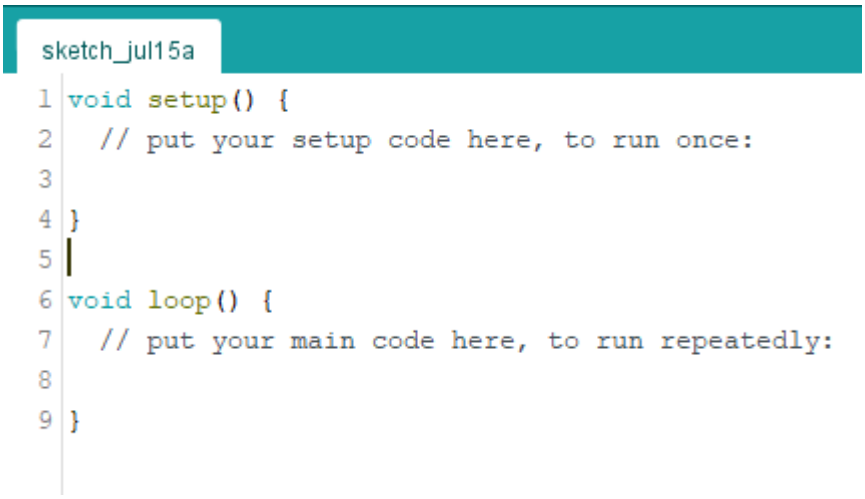
PLANEJAMENTO FASE 1

- ❖ DECLARAR AS VARIÁVEIS QUE SERÃO UTILIZADAS NO PROJETO
- ❖ NOMEAR OS PINOS DO DE ENTRADA E SAÍDA DO ARDUINO

✓ **Nomear pino 13 como** SEGA ✓ **Nomear pino 12 como** SEGB ✓ **Nomear pino 11 como** SEGC ✓ **Nomear pino 10 como** SEGD ✓ **Nomear pino 9 como** SEGE ✓ **Nomear pino 8 como** SEGF ✓ **Nomear pino 7 como** SEGG ✓ **Nomear pino 3 como** BOTAO1 ✓ **Nomear pino 2 como** BOTAO2	✓ Criar variável **TIPO BOOLEANA** para armazenar valor do **BOTÃO1** ✓ Criar variável **TIPO BOOLEANA** para armazenar valor do **BOTÃO2** ✓ Criar variável do **TIPO BYTE** para armazenar a velocidade da esteira.

Depois de estruturarmos as necessidades iniciais do nosso projeto, vamos começar a montar nosso código de programação do absoluto zero.

Abra a IDE do arduino e clique em Arquivo – NOVO para criar um novo projeto.



Nesse novo arquivo teremos as funções setup e loop já disponíveis. Agora precisaremos nomear os pinos do arduino seguindo nosso planejamento anterior. Essa nomeação será feita fora das funções setup e loop.

TRÊS FORMAS DE NOMEAR OS PINOS DO ARDUINO.

Podemos nomear os pinos do arduino basicamente de 3 maneiras, utilizando **#define, const int, e int.** Veja como podemos nomear o pino 13 como SEGA utilizando as três formas.

#define SEGA 13 (sem ponto e vírgula no final) **–** Quando utilizamos a diretiva *#define* não estamos reservando um espaço de memória para SEGA, mas estamos pedindo que o compilador troque a palavra SEGA pelo valor 13 todas as vezes que ela aparecer no código.

const int SEGA = 13; – estamos reservando um espaço de 2 bytes para armazenar um número inteiro constante, que NÃO PODERÁ SER ALTERADO durante a execução do programa.

Int SEGA = 13; - estamos armazenando um espaço de 2 Bytes para armazenar um número inteiro.

VARIÁVEL TIPO BOOLEANA E TIPO BYTE

Variáveis do **tipo booleana** são utilizadas para armazenar valores lógicos digitais, como **VERDADEIRO ou FALSO, 0 ou 1**. Para declarar uma variável do tipo booleana, basta colocar o prefixo bool antes do nome escolhido para a variável.

```
bool variavel;
```

Podemos acessar maiores informações na própria IDE do arduino por meio do link https://www.arduino.cc/reference/pt/language/variables/data-types/bool/

bool

[Data Types]

Descrição

O tipo `bool` pode armazenar dois valores: `true` or `false`. (Cada variável `bool` ocupa um byte na memória.)

Sintaxe

```
bool var = val;
```

Parâmetros

`var`: nome da variável

`val`: valor a ser atribuído à variável

Variáveis do tipo Byte são utilizadas apara armazenar números positivos de 0 a 255.

A disponibilidade de tipos na hora de declarar as variáveis que apresentamos aqui tem por objetivo aumentar o conhecimento do aluno oferecendo ferramentas e recursos para que sua autonomia seja cada vez maior no mundo da programação.

Para declarar uma variável do tipo byte basta adicionar o prefixo byte antes do nome escolhido para a variável.

```
byte variavel;
```

Para saber mais sobre as variáveis do tipo byte, basta acessar o link https://www.arduino.cc/reference/pt/language/variables/data-types/byte/

byte

[Data Types]

Descrição

Uma variável 'byte' armazena valores numéricos de 8-bit sem sinal, de 0 a 255.

Sintaxe

```
byte var = val;
```

Parâmetros

`var`: nome da variável

`val`: valor a ser atribuído à variável

Depois de todas as pesquisas e planejamento inicial nosso código inicialmente ficará assim:

sketch_jul18a §

```
1
2  #define SEGA 13
3  #define SEGB 12
4  #define SEGC 11
5  #define SEGD 10
6  #define SEGE 9
7  #define SEGF 8
8  #define SEGG 7
9
10 #define BOTAO1 3
11 #define BOTAO2 2
12
13 bool VALOR_B1 = false;
14 bool VALOR_B2 = 0;
15
16 byte velocidade = 0;
```

CONFIGURAÇÕES DE SETUP

Agora chegou o momento de configurarmos os pinos como entrada e como saída dentro da função setup. Para isso vamos organizar primeiro a sequência de planejamento fase 2, e em seguida já colocaremos esse planejamento em linguagem de programação na IDE do arduino.

PLANEJAMENTO FASE 2

❖ CONFIGURAÇÕES INICIAIS DO PROJETO

❖ VOID SETUP () {}

- ✓ **Configurar** SEGA como saída Digital
- ✓ **Configurar** SEGB como saída Digital
- ✓ **Configurar** SEGC como saída Digital
- ✓ **Configurar** SEGD como saída Digital
- ✓ **Configurar** SEGE como saída Digital
- ✓ **Configurar** SEGF como saída Digital
- ✓ **Configurar** SEGG como saída Digital
- ✓ **Configurar** BOTAO1 como entrada Digital
- ✓ **Configurar** BOTAO2 como entrada Digital
- ✓ Inicializar a **comunicação serial** em 9600bps

```
18 void setup()
19 {
20   pinMode (SEGA, OUTPUT);
21   pinMode (SEGB, OUTPUT);
22   pinMode (SEGC, OUTPUT);
23   pinMode (SEGD, OUTPUT);
24   pinMode (SEGE, OUTPUT);
25   pinMode (SEGF, OUTPUT);
26   pinMode (SEGG, OUTPUT);
27
28   pinMode (BOTAO1, INPUT);
29   pinMode (BOTAO2, INPUT);
30
31   Serial.begin (9600);
32 }
```

CONFIGURANDO O VOID LOOP

Chegou a hora de montarmos a estrutura de funcionamento do nosso programa dentro da função void loop, e será nessa etapa que estará presente toda a lógica de programação do nosso projeto e que ficará se repetindo por todo o tempo em que a placa estiver em funcionamento.

PLANEJAMENTO FASE 3 – parte 1

- CONFIGURAÇÕES DA LÓGICA DE PROGRAMAÇÃO
- VOID LOOP () {}

- ✓ **Ler** BOTAO1 e armazenar valor lido em VALOR_B1
- ✓ **Testar** se botão 1 foi pressionado. Se sim, **verificar** se velocidade está em intensidade máxima permitida (9).**Se** velocidade for menor que 9 aumentar velocidade, caso velocidade atingir valor máximo **emitir alerta** ao usuário e não aumentar mais o valor.
- ✓ Toda vez que velocidade for aumentada, **imprimir valor atual.**
- ✓ **Se** botão não estiver pressionado, emitir mensagem ao usuário dizendo que o sistema aguarda comandos para aumentar velocidade.

Vamos então converter nosso planejamento em lógica de programação.

- ✓ **Ler** BOTAO1 e armazenar valor lido em VALOR_B1

```
VALOR_B1 = digitalRead (BOTAO1);
```

- ✓ **Testar** se botão 1 foi pressionado. Se sim, **verificar** se velocidade está em intensidade máxima permitida (9).**Se** velocidade for menor que 9 aumentar velocidade, caso velocidade atingir valor máximo **emitir alerta** ao usuário e não aumentar mais o valor.

```
if (VALOR_B1 == 1)
{
  if (velocidade < 9)
  {
    velocidade++;
  }
  else
  {
    Serial.println ("Velocidade Máxima Atingida");
  }
}
```

✓ Toda vez que velocidade for aumentada, **imprimir valor atual.**

```
void loop()
{
  VALOR_B1 = digitalRead (BOTAO1);

  if (VALOR_B1 == 1)
  {
    if (velocidade < 9)
    {
      velocidade++;
      Serial.print ("Velocidade Atual:");
      Serial.println (velocidade);
    }
    else
    {
      Serial.println ("Velocidade Máxima Atingida");
    }
  }
```

✓ **Se** botão não estiver pressionado, emitir mensagem ao usuário dizendo que o sistema aguarda comandos para aumentar velocidade.

```
else
{
  Serial.println ("Aguardando Comandos");
}
```

ATENÇÃO – ao testar esse código você verá que ao clicar uma única vez no botão o serial monitor irá apresentar a mensagem de velocidade máxima atingida. Isso acontecerá por conta da velocidade de execução do loop. Ao clicar uma única vez no botão, o tempo em que ele fica pressionado, ainda que seja muito pequeno, o loop já foi executado muitas vezes, contabilizando diversas vezes esse pouco tempo de botão pressionado e dessa forma ultrapassando o limite máximo de velocidade.

Para corrigir esse problema, vamos adicionar um delay de 500ms fazendo com que o loop seja executado nessa velocidade. Essa velocidade de execução pode ser alterada de acordo com o objetivo do projetista.

Temos então o nosso loop completo com os testes iniciais do nosso planejamento.

```
void loop()
{
  VALOR_B1 = digitalRead (BOTAO1);

  if (VALOR_B1 == 1)
  {
    if (velocidade < 9)
    {
      velocidade++;
      Serial.print ("Velocidade Atual:");
      Serial.println (velocidade);
    }
    else
    {
      Serial.println ("Velocidade Máxima Atingida");
    }
  }
  else
  {
    Serial.println ("Aguardando Comandos");
  }
  delay (500);
}
```

PLANEJAMENTO FASE 3 – parte 2

- CONFIGURAÇÕES DA LÓGICA DE PROGRAMAÇÃO
- VOID LOOP () {}

✓ **Exibir** velocidade atual da esteira no display de 7 segmentos

Agora vamos implementar no nosso código a impressão da velocidade atual da esteira no display de 7 segmentos. O teste básico que podemos fazer é:

```
Se velocidade = 0
{
   escrever 0 no display;
}

Senão, se velocidade = 1
{
   escrever 1 no display;
}

Senão, se velocidade = 2
{
   escrever 2 no display;
}

[....]continua até 8

Senão
{
   escrever 9 no display;
}
```

Veja que a última condição de teste SENÃO já conclui que se a velocidade não foi nenhuma das anteriores, ela será a última permitida que é 9.

Agora vamos passar essa lógica para a linguagem de programação.

***DICA:* FAÇA UMA FUNÇÃO PARA CADA IMPRESSÃO DE NUMERO NO DISPLAY, POR EXEMPLO escreve_0(); escreve_1(); ... e assim sucessivamente. Essas funções devem ser criadas fora da função loop.**

Veja como fica muito mais organizado criar uma função para cada impressão de número no display. Depois, basta chamar essa função quando houver a necessidade de imprimir determinado número.

```
void escreve_0()
{
  digitalWrite(SEGA, HIGH);
  digitalWrite(SEGB, HIGH);
  digitalWrite(SEGC, HIGH);
  digitalWrite(SEGD, HIGH);
  digitalWrite(SEGE, HIGH);
  digitalWrite(SEGF, HIGH);
  digitalWrite(SEGG,  LOW);
}
void escreve_1()
{
  digitalWrite(SEGA,  LOW);
  digitalWrite(SEGB, HIGH);
  digitalWrite(SEGC, HIGH);
  digitalWrite(SEGD,  LOW);
  digitalWrite(SEGE,  LOW);
  digitalWrite(SEGF,  LOW);
  digitalWrite(SEGG,  LOW);
}
```

```
void escreve_2()
{
  digitalWrite(SEGA, HIGH);
  digitalWrite(SEGB, HIGH);
  digitalWrite(SEGC,  LOW);
  digitalWrite(SEGD, HIGH);
  digitalWrite(SEGE, HIGH);
  digitalWrite(SEGF,  LOW);
  digitalWrite(SEGG, HIGH);
}
void escreve_3()
{
  digitalWrite(SEGA, HIGH);
  digitalWrite(SEGB, HIGH);
  digitalWrite(SEGC, HIGH);
  digitalWrite(SEGD, HIGH);
  digitalWrite(SEGE,  LOW);
  digitalWrite(SEGF,  LOW);
  digitalWrite(SEGG, HIGH);
}
```

```
void escreve_4()
{
  digitalWrite(SEGA,  LOW);
  digitalWrite(SEGB, HIGH);
  digitalWrite(SEGC, HIGH);
  digitalWrite(SEGD,  LOW);
  digitalWrite(SEGE,  LOW);
  digitalWrite(SEGF, HIGH);
  digitalWrite(SEGG, HIGH);
}
void escreve_5()
{
  digitalWrite(SEGA, HIGH);
  digitalWrite(SEGB,  LOW);
  digitalWrite(SEGC, HIGH);
  digitalWrite(SEGD, HIGH);
  digitalWrite(SEGE,  LOW);
  digitalWrite(SEGF, HIGH);
  digitalWrite(SEGG, HIGH);
}
```

```
void escreve_6()
{
  digitalWrite(SEGA, HIGH);
  digitalWrite(SEGB,  LOW);
  digitalWrite(SEGC, HIGH);
  digitalWrite(SEGD, HIGH);
  digitalWrite(SEGE, HIGH);
  digitalWrite(SEGF, HIGH);
  digitalWrite(SEGG, HIGH);
}
void escreve_7()
{
  digitalWrite(SEGA, HIGH);
  digitalWrite(SEGB, HIGH);
  digitalWrite(SEGC, HIGH);
  digitalWrite(SEGD,  LOW);
  digitalWrite(SEGE,  LOW);
  digitalWrite(SEGF,  LOW);
  digitalWrite(SEGG,  LOW);
}
```

```
void escreve_8()
{
  digitalWrite(SEGA, HIGH);
  digitalWrite(SEGB, HIGH);
  digitalWrite(SEGC, HIGH);
  digitalWrite(SEGD, HIGH);
  digitalWrite(SEGE, HIGH);
  digitalWrite(SEGF, HIGH);
  digitalWrite(SEGG, HIGH);
}
```

```
void escreve_9()
{
  digitalWrite(SEGA, HIGH);
  digitalWrite(SEGB, HIGH);
  digitalWrite(SEGC, HIGH);
  digitalWrite(SEGD, HIGH);
  digitalWrite(SEGE,  LOW);
  digitalWrite(SEGF, HIGH);
  digitalWrite(SEGG, HIGH);
}
```

Agora vamos testar a velocidade da esteira no nosso código e vamos chamar a função de impressão no display correspondente ao valor atual de velocidade.

```
if (velocidade == 0)
{
  escreve_0();
}
 else if (velocidade == 1)
 {
  escreve_1();
 }
 else if (velocidade == 2)
 {
  escreve_2();
 }
 else if (velocidade == 3)
 {
  escreve_3();
 }
 else if (velocidade == 4)
 {
  escreve_4();
 }

 else if (velocidade == 5)
 {
  escreve_5();
 }
 else if (velocidade == 6)
 {
  escreve_6();
 }
 else if (velocidade == 7)
 {
  escreve_7();
 }
 else if (velocidade == 8)
 {
  escreve_8();
 }
 else
 {
  escreve_9();
 }
```

PLANEJAMENTO FASE 3 – parte 3

❖ CONFIGURAÇÕES DA LÓGICA DE PROGRAMAÇÃO

❖ VOID LOOP () {}

- ✓ **Ler** BOTAO1 e atribuir o valor lido a VALOR_B1
- ✓ **Ler** BOTAO2 e atribuir o valor lido a VALOR_B2
- ✓ **Testar se** BOTAO1 foi pressionado. Se velocidade for menor que 9, aumentar em uma unidade o valor da velocidade.
- ✓ **Testar se** BOTAO2 foi pressionado. Se velocidade for maior que 0, diminuir em uma unidade o valor da velocidade.
- ✓ **Exibir** velocidade atual no display de 7 segmentos

- ✓ **Ler** BOTAO1 e atribuir o valor lido a VALOR_B1
- ✓ **Ler** BOTAO2 e atribuir o valor lido a VALOR_B2

```
VALOR_B1 = digitalRead (BOTAO1);
VALOR_B2 = digitalRead (BOTAO2);
```

- ✓ **Testar se** BOTAO1 foi pressionado. Se velocidade for menor que 9, aumentar em uma unidade o valor da velocidade.

```
if (VALOR_B1 == 1)
{
  if (velocidade < 9)
  {
    velocidade++;
    Serial.print ("Velocidade Atual:");
    Serial.println (velocidade);
  }
}
```

- ✓ **Testar se** BOTAO2 foi pressionado. Se velocidade for maior que 0, diminuir em uma unidade o valor da velocidade.

```
  if (VALOR_B2 == 1)
{
  if (velocidade > 0)
  {
    velocidade--;
    Serial.print ("Velocidade Atual:");
    Serial.println (velocidade);
  }
}
```

✓ **Exibir** velocidade atual no display de 7 segmentos

Esta etapa é exatamente igual a etapa anterior onde implementamos um código para que os números fossem impressos no display.

> *Coloque também um delay de 500ms dentro de cada termino de if, isso ajudará na velocidade de leitura do botão, deixando uma latência para que diversos valores não sejam lidos ao mesmo tempo.*

Dessa forma encerramos a lógica de programação para leitura do botão 1 e botão 2, aumentando e diminuindo a velocidade da esteira de academia. Vamos então para próxima etapa.

```
void loop()
{
  VALOR_B1 = digitalRead (BOTAO1);
  VALOR_B2 = digitalRead (BOTAO2);

  if (VALOR_B1 == 1)
  {
    if (velocidade < 9)
    {
      velocidade++;
      Serial.print ("Velocidade Atual:");
      Serial.println (velocidade);
      delay (500);
    }
  }

    if (VALOR_B2 == 1)
  {
    if (velocidade > 0)
    {
      velocidade--;
      Serial.print ("Velocidade Atual:");
      Serial.println (velocidade);
      delay (500);
    }
  }
```

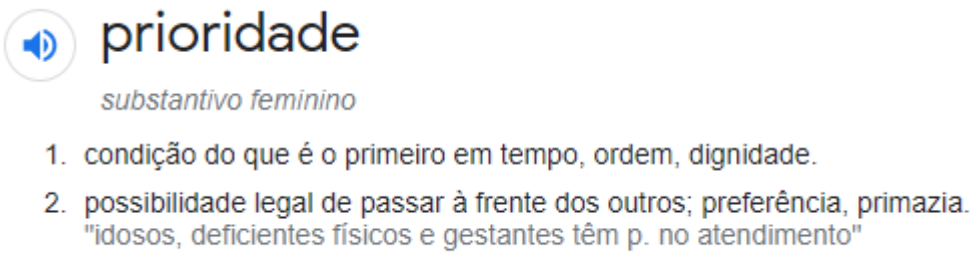
O restante do código é o mesmo do anterior, com os testes da velocidade e impressão do valor atual no display.

DEFININDO A PRIORIDADE DO BOTÃO

Boa parte do nosso objetivo já foi concluído, porém ainda precisamos implementar a prioridade do botão de diminuição de velocidade quando os dois botões forem pressionados juntos. (aumento e diminuição).

Isso deve ser implementado pensando na segurança do atleta, não só porque ele pode apertar os dois ao mesmo tempo sem querer, mas prevenindo um possível problema no botão de aumento que faria com que a velocidade aumentasse sem interrupção. Com isso, dando prioridade para o botão de diminuição, um possível problema de disparo no botão de aumento poderá ser solucionado pressionando o botão de diminuição.

Para falarmos em prioridade de botão, vamos primeiro entender qual a definição de prioridade:



Se prioridade é o que acontece primeiro, é o que tem preferência, podemos concluir que se os dois botões estiverem apertados, somente o de diminuição funcionará, afinal ele será a prioridade.

Para isso, vamos colocar nosso código na seguinte ordem:

1. *Testar botão de diminuição*
2. *Testar botão de aumento*

Agora vamos pensar na lógica que deverá ser aplicada para que a prioridade aconteça.

Sabemos que, se botão de diminuição for pressionado deverá diminuir velocidade da esteira. Sabemos também que, se **SOMENTE** botão de aumento de velocidade for pressionado, a velocidade da esteira deverá subir. Veja que agora importa se ele foi pressionado sozinho ou não.

Dessa forma, a velocidade só poderá aumentar se botão de aumento **ESTIVER PRESSIONADO** e botão de diminuição **NÃO ESTIVER PRESSIONADO. Se os dois estiverem pressionados juntos, o aumento não acontece.** Então, ficaremos assim:

> ***Se*** *BOTAO2 == 1 → Diminua velocidade*
>
> ***Se*** *BOTAO1 == 1* ***E*** *BOTAO2 == 0 → Aumente velocidade*

Para aplicarmos a lógica **E** na condição de teste, utilizaremos o operador **&&.**

De forma resumida o código fica assim:

```
if (VALOR_B2 == 1)
{
  if (velocidade > 0)
  {
    velocidade--;
  }
}
if (VALOR_B1 == 1 && VALOR_B2 == 0)
{
  if (velocidade < 9)
  {
    velocidade++;
  }
}
```

Montando o código completo como estávamos fazendo antes, teremos:

```
if (VALOR_B2 == 1)
{
  if (velocidade > 0)
  {
    velocidade--;
    Serial.print ("Velocidade Atual:");
    Serial.println (velocidade);
    delay (500);
  }
}
if (VALOR_B1 == 1 && VALOR_B2 == 0)
{
  if (velocidade < 9)
  {
    velocidade++;
    Serial.print ("Velocidade Atual:");
    Serial.println (velocidade);
    delay (500);
  }
}
```

CLIQUE AQUI PARA ACESSAR A AULA

PRIORIDADE COM *IF - ELSE IF*

Existem diversas maneiras de se conseguir implementar uma lógica de programação para atingir um mesmo objetivo.

Uma outra forma de conseguirmos dar prioridade ao botão de diminuição da esteira é utilizando a estrutura de controle if – else if vista anteriormente.

Sabemos que a estrutura if -else if tem uma arquitetura que prioriza o teste verdadeiro, deixando de testar as próximas condições seguintes, pois considera que se a condição verdadeira já foi executada, todas as outras automaticamente serão falsas.

Dessa forma, podemos dar prioridade ao botão de diminuição de velocidade da esteira da seguinte maneira:

> **SE** BOTAO2 == 1 → diminuir velocidade.
>
> **SENAO SE** BOTAO1 == 1 → aumentar velocidade

Veja que agora a condição de teste de aumento de velocidade só acontecerá se botão de diminuição não estiver pressionado. Se botão de diminuição for pressionado, o botão de aumento nem será testado, dando total prioridade pra o botão de diminuição.

Vamos converter essa lógica para o código de programação resumido.

```
if (VALOR_B2 == 1)
{
  if (velocidade > 0)
  {
    velocidade--;
  }
}
else if (VALOR_B1 == 1)
{
  if (velocidade < 9)
  {
    velocidade++;
  }
}
```

Agora o código de teste completo com as funções de impressão do serial print e o delay para melhorar a leitura do botão.

```
if (VALOR_B2 == 1)
{
  if (velocidade > 0)
  {
    velocidade--;
    Serial.print ("Velocidade Atual:");
    Serial.println (velocidade);
    delay (500);
  }
}
else if (VALOR_B1 == 1)
{
  if (velocidade < 9)
  {
    velocidade++;
    Serial.print ("Velocidade Atual:");
    Serial.println (velocidade);
    delay (500);
  }
}
```

ORGANIZANDO O CÓDIGO EM FUNÇÕES

Um fator importantíssimo em programação é a legibilidade do código, quanto mais organizado estiver a estrutura, mais fácil será seu entendimento e sua manutenção futura.

Sabemos que a lógica de funcionamento alocada dentro da função loop possui basicamente três etapas: **LEITURA DAS ENTRADAS, CONTROLE DE VELOCIDADE, e ATUALIZAÇÃO DO DISPLAY**.

Para melhorar a legibilidade do nosso código, podemos transformar cada etapa mencionada anteriormente em uma função, bastando chama-las dentro da função loop.

ATENÇÃO – a função é criada fora do loop.

Para criar uma função que não retorna nenhum valor, utilizaremos ***VOID***.

```
void LEITURA_ENTRADAS()
{

}

void CONTROLE_VELOCIDADE()
{

}

void ATUALIZA_DISPLAY()
{

}
```

Agora vamos inserir dentro de cada nova função criada, o código correspondente dentro da função loop.

```
void LEITURA_ENTRADAS()
{
  VALOR_B1 = digitalRead (BOTAO1);
  VALOR_B2 = digitalRead (BOTAO2);
}
```

```
void CONTROLE_VELOCIDADE()
{
  if (VALOR_B2 == 1)
  {
    if (velocidade > 0)
    {
      velocidade--;
      Serial.print ("Velocidade Atual:");
      Serial.println (velocidade);
      delay (500);
    }
  }
  else if (VALOR_B1 == 1)
  {
    if (velocidade < 9)
    {
      velocidade++;
      Serial.print ("Velocidade Atual:");
      Serial.println (velocidade);
      delay (500);
    }
  }
}
```

```
void ATUALIZA_DISPLAY()
{

  if (velocidade == 0)
  {
    escreve_0();
  }
   else if (velocidade == 1)
   {
    escreve_1();
   }
   else if (velocidade == 2)
   {
    escreve_2();
   }
   else if (velocidade == 3)
   {
    escreve_3();
   }
   else if (velocidade == 4)
   {
    escreve_4();
   }
```

```
    else if (velocidade == 5)
    {
     escreve_5();
    }
    else if (velocidade == 6)
    {
     escreve_6();
    }
    else if (velocidade == 7)
    {
     escreve_7();
    }
    else if (velocidade == 8)
    {
     escreve_8();
    }
    else
    {
     escreve_9();
    }
}
```

Agora vamos chamar dentro do loop as funções criadas. Veja como ficará muito mais legível o funcionamento do nosso código E MUITO MAIS ORGANIZADO. Ao olhar essa nova forma de organizar o loop, imediatamente conseguimos identificar as etapas de funcionamento do programa.

```
void loop()
{
  LEITURA_ENTRADAS();
  CONTROLE_VELOCIDADE();
  ATUALIZA_DISPLAY();
}
```

TESTE DE BOTÕES E VELOCIDADE JUNTOS

Agora que nosso código está todo organizado, vamos fazer uma manutenção de melhoria dentro da função CONTROLE_VELOCIDADE.

Observe que há dois ifs dentro da condição de teste dos botões. Primeiro o if testa se o botão foi apertado e depois outro if testa se a velocidade é compatível com a função de aumentar ou diminuir a velocidade.

```
if (VALOR_B2 == 1)
{
  if (velocidade > 0)
  {
    velocidade--;
    Serial.print ("Velocidade Atual:");
    Serial.println (velocidade);
    delay (500);
  }
}
```

Podemos fazer esses dois testes em um único if utilizando a **lógica E**.

SE VALOR_B2 for igual a 1 **E** velocidade for maior que zero

```
if (VALOR_B2 == 1 && velocidade > 0)
{
    velocidade--;
    Serial.print ("Velocidade Atual:");
    Serial.println (velocidade);
    delay (500);
}
```

Aplicamos a mesma lógica ao outro teste de botão.

SENÃO SE VALOR_B1 for igual a 1 **E** velocidade for menor que 9

```
else if (VALOR_B1 == 1 && velocidade < 9)
{
    velocidade++;
    Serial.print ("Velocidade Atual:");
    Serial.println (velocidade);
    delay (500);
}
```

Dessa forma, fechamos de maneira sensacional nosso projeto da esteira, utilizando dois botões do shield, um para aumentar a velocidade e outro para diminuir a velocidade mostrando no display a velocidade atual. Aprendemos também a testar quando um botão é apertado ou não, aprendemos a aplicar a lógica boolena E, e ainda aprendemos sobre prioridades na hora de implementação. Veja quantas informações importantes foram compartilhadas, quanto conhecimento absorvido desenvolvendo passo a passo cada etapa desse projeto.

Vamos que vamos que vem muito mais conhecimento por aí.

INFORMAÇÕES INFINITAS NO ARDUINO.CC

Uma grande qualidade em um técnico, em um programador, ou mesmo em uma pessoa que trabalha em qualquer área, é a CURIOSIDADE.

curiosidade

substantivo feminino

1. característica ou qualidade de curioso.
2. desejo intenso de ver, ouvir, conhecer, experimentar algo ger. novo, original, desconhecido.
   "o programa não satisfez a c. dos espectadores"

Veja que, a partir da definição de curiosidade podemos concluir que o curioso está sempre em busca de conhecer, experimentar, buscar o novo, e isso é muito poderoso quando estamos aprendendo coisas novas. Por isso, desperte sua curiosidade e navegue pela imensidão de conteúdos disponíveis na própria plataforma online do arduino. https://www.arduino.cc/

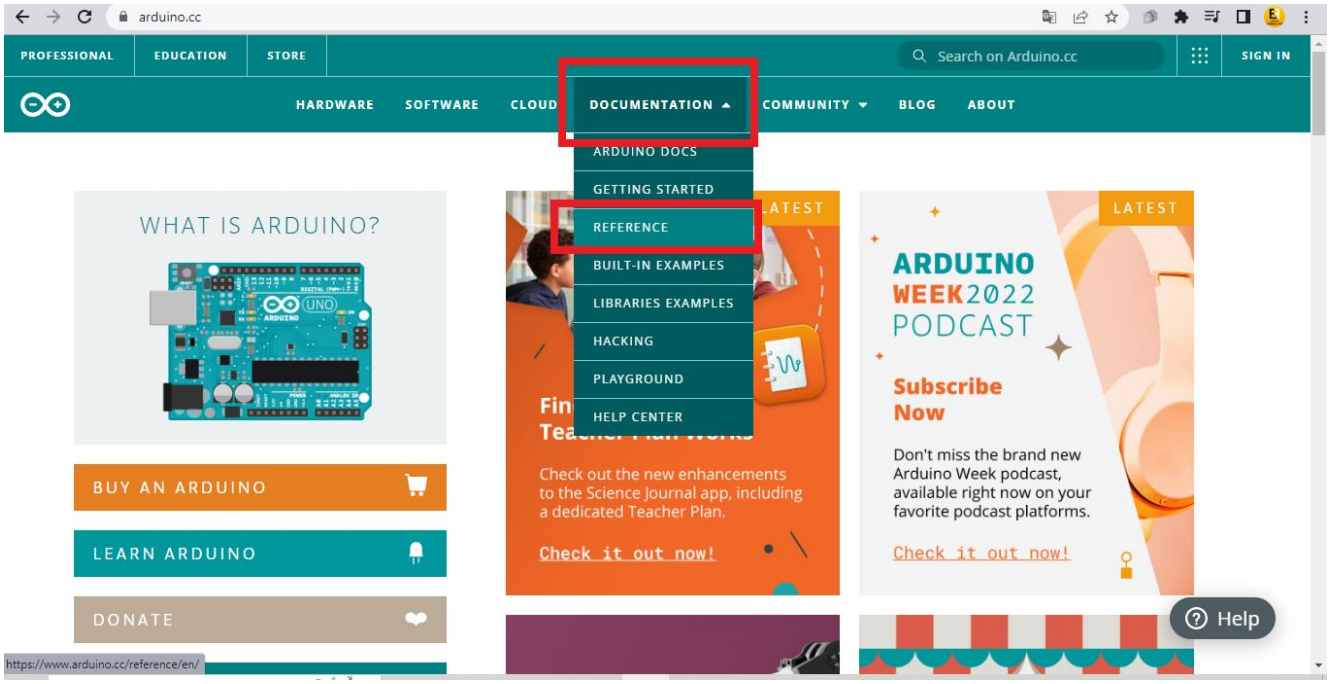


Figura 94 - Referências e documentações

Nessa página você encontrará diversas referências sobre funções, tipos de variáveis, estruturas de controle, operadores, comparadores, enfim, uma infinidade de conteúdos incríveis.

Vamos encontrar nessa página os operadores booleanos, e esse será nosso assunto a partir de agora.

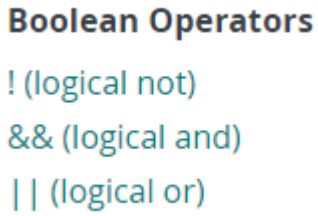


Figura 95 - Operadores Booleanos

Já aprendemos como utilizar os operadores && para implementar a lógica E. Agora vamos utilizar dois caracteres “pipe” || (duas barras verticais) para implementar a lógica OU.

Nos conteúdos anteriores já trabalhamos bastante as lógicas E e OU, por isso vamos considerar que estas lógicas estão bem definidas dentro do conhecimento de vocês. O que vamos aprender aqui é converter essa ideia de lógica **OU** e **NÃO** para a linguagem de programação.

Vamos implementar um código que irá ativar o buzzer quando os dois botões do shield estiverem ativados. Logo, vamos utilizar a lógica **E** já estudada.

```
//---------NOMEAÇÃO DOS PINOS------------
const int BOTAO1 = 3;
const int BOTAO2 = 2;
const int BUZZER = 5;

//-------DECLARAÇÃO DE VARIÁVEIS---------
bool VALOR_B1 = 0;
bool VALOR_B2 = 0;

//---CONFIGURAÇÃO DE ENTRADAS E SAÍDAS---
void setup()
{
  pinMode (BOTAO1, INPUT);
  pinMode (BOTAO2, INPUT);
  pinMode (BUZZER, OUTPUT);
}

//---LAÇO DE REPETIÇÃO DA LÓGICA PROGRAMADA---
void loop()
{
  VALOR_B1 = digitalRead (BOTAO1);
  VALOR_B2 = digitalRead (BOTAO2);

  if (VALOR_B1 == 1 && VALOR_B2 == 1)
  {
    digitalWrite (BUZZER, HIGH);
  }
  else
  {
    digitalWrite (BUZZER, LOW);
  }
}
```

Agora vamos utilizar o mesmo programa para implementar a **lógica OU**, e para isso vamos trocar apenas os operadores **E (&&)** por operadores **OU (||).**

Na lógica E condicionamos que se botão 1 **E** botão 2 fossem pressionados, o buzzer seria ativado.

Na lógica OU condicionaremos que se o botão 1 **OU** o botão 2 for pressionado, o buzzer será ativado.

```
//---LAÇO DE REPETIÇÃO DA LÓGICA PROGRAMADA---
void loop()
{
  VALOR_B1 = digitalRead (BOTAO1);
  VALOR_B2 = digitalRead (BOTAO2);

  if (VALOR_B1 == 1 || VALOR_B2 == 1)
  {
    digitalWrite (BUZZER, HIGH);
  }
  else
  {
    digitalWrite (BUZZER, LOW);
  }
}
```

Agora vamos utilizar o operador de negação **NÃO,** ou operador inversor, e o símbolo utilizado para esse operador é a exclamação (!).

Quando colocamos a exclamação na frente de alguma variável para fazer um teste condicional, estamos invertendo o true pelo false, ou seja, a condição será executada quando for false. Por exemplo:

Aprendemos que quando a condição for verdadeira, ação é executada:

SE (verdadeiro) { executar ação}

Mas se colocarmos ! antes da variável, a ação será executada quando condição for falsa

SE (!verdadeiro) { executar ação}

ou

SE (falso) { executar ação}

```
//---LAÇO DE REPETIÇÃO DA LÓGICA PROGRAMADA---
void loop()
{
  VALOR_B1 = digitalRead (BOTAO1);
  VALOR_B2 = digitalRead (BOTAO2);

  if (!VALOR_B1)
  {
    digitalWrite (BUZZER, HIGH);
  }
  else
  {
    digitalWrite (BUZZER, LOW);
  }
}
```

Veja no código acima que, a exclamação colocada antes de VALOR_B1 irá inverter o valor dele, então, quando ele valer 0 o valor será invertido para 1, e a condição será executada, ou seja, quando VALOR_B1 for **falso**, **a condição será executada**. Quando botão não estiver pressionado o buzzer será ligado, e quando o botão for pressionado o buzzer será desligado.

Aplicando a ideia anterior de teste, podemos aplicar a lógica **SIM,** ou seja, se a condição for verdadeira, o if será executado.

Veja as diversas formas de escrever a lógica SIM e a lógica NÃO em linguagem de programação.

LÓGICA NÃO

```
if (!VALOR_B1){ faça algo; }
if (VALOR_B1 == 0){ faça algo; }
if (VALOR_B1 == LOW){ faça algo; }
if (VALOR_B1 == false){ faça algo; }
```

LÓGICA SIM

```
if (VALOR_B1){ faça algo; }
if (VALOR_B1 == 1){ faça algo; }
if (VALOR_B1 == HIGH){ faça algo; }
if (VALOR_B1 == true){ faça algo; }
```

EXERCÍCIOS PARA PRATICAR

Chegou a hora de colocar os conhecimentos aprendidos em prática por meio de uma série de exercícios dispostos em tabela da verdade. Você deverá utilizar o botão 1 e 2 do shield para Arduino para ativar o buzzer, mas seguindo a disposição lógica apresentada na tabela da verdade. Lembrando, nível 1 = ativado, nível 0 = desativado, aplicado tanto ao buzzer quanto ao botão.

Você deverá criar um código para cada exercício.

EXERCÍCIO 1		
VALOR_B1	**VALOR_B2**	**BUZZER**
0	**0**	**1**
0	**1**	**0**
1	**0**	**0**
1	**1**	**0**

EXERCÍCIO 2		
VALOR_B1	**VALOR_B2**	**BUZZER**
0	**0**	**0**
0	**1**	**1**
1	**0**	**0**
1	**1**	**0**

EXERCÍCIO 3		
VALOR_B1	**VALOR_B2**	**BUZZER**
0	**0**	**0**
0	**1**	**0**
1	**0**	**1**
1	**1**	**0**

EXERCÍCIO 4		
VALOR_B1	VALOR_B2	BUZZER
0	0	0
0	1	1
1	0	1
1	1	0

EXERCÍCIO 5		
VALOR_B1	VALOR_B2	BUZZER
0	0	1
0	1	0
1	0	0
1	1	1

PERGUNTAS BÔNUS

1. *O Que é uma variável?*
2. *Em qual memória do microcontrolador os valores das variáveis são salvos?*
3. *O que é memória volátil?*
4. *O que é memória não volátil?*
5. *O que significa debugar um programa?*
6. *O que é uma variável local?*
7. *O que é uma variável global?*
8. *Qual é a diferença entre uma variável unsigned int para apenas int?*
9. *O que é um estouro de variável?*
10. *Quais são os benefícios de usar a estrutura if e else if ao invés de usar apenas os testes através de múltiplos if em sequência na programação do microcontrolador?*

FASE 3 – DESAFIOS INDIVIDUAIS, SAINDO DA ZONA DE CONFORTO

Chegou a hora de materializar todo o conhecimento adquirido nesse módulo incrível e sensacional, e para isso você será desafiado a criar 3 projetos individuais para resolver alguma situação problema do dia a dia. Você escolherá a situação problema que será resolvida.

É importante tentar aplicar o máximo de conhecimento adquirido durantes seus estudos nessa resolução de problema que será implementada por você. Esse exercício vem justamente para que você consiga fixar ainda mais tudo que foi ensinado, tornando mais simples a visualização da aplicabilidade dos conceitos estudados.

Abaixo teremos um pré-planejamento de como você pode estruturar o projeto, visando dar mais agilidade ao processo de criação e auxiliando nas etapas a serem seguidas e que são fundamentais a todo projeto.

➔ Criação de cabeçalho em forma de comentário.

```
// ..... NOME DO PROJETO:
//.....DESENVOLVIDO POR:
//.....DATA:
```

➔ Nomear botões
➔ Nomear leds
➔ Criar variável para armazenar valores dos botões

➔ Configurar botões como entrada digital
➔ Configurar leds como saídas digitais
➔ Inicializar comunicação seria (9600)bps

➔ Construir a lógica do programa utilizando os conceitos aprendidos neste módulo

DESAFIO DO MESTRE 1 – TORNEIRA AUTOMATIZADA

Você será desafiado a fazer o desenvolvimento da programação de um microcontrolador que irá fazer a gestão de funcionamento de uma torneira com sensor de presença.

Suas Tarefas para resolver este desafio serão:

1. Elaborar a programação na IDE do Arduino e validar o funcionamento na prática utilizando o shield para Arduino Starter do Eletrônica Fácil como hardware de teste rápido;

2. Testar o passo a passo de funcionamento deste projeto utilizando o software Proteus;

3. Postar fotos, vídeos, códigos desenvolvidos com as #ERDINGER #DesafioDoMestre #FamiliaEletronicaFacil #Desafio1 na comunidade E. Fácil para compartilhar o seu sucesso com os demais alunos de turma e comemorarmos juntos a sua vitória.

A torneira terá dois **MODOS DE OPERAÇÃO**

1 – Modo Contínuo: *Ativado pelo sensor de presença localizado na lateral da torneira.*

- ✓ *Ao aproximar a mão pela 1ª vez no sensor lateral, a eletroválvula é ATIVADA liberando água continuamente.*
- ✓ *Ao aproximar a mão pela 2ª vez no sensor lateral, a eletroválvula é DESATIVADA interrompendo a liberação de água.*

2 – Modo Instantâneo: *Ativado pelo sensor de presença localizado na parte de baixo da torneira*

- ✓ *Enquanto a mão estiver próxima ao sensor, a eletroválvula permanece ATIVADA liberando água continuamente*
- ✓ *Ao afastar a mão, a eletroválvula é DESATIVADA interrompendo liberação de água*

Obs. A torneira só funcionará em um modo por vez, portanto se ***modo continuo*** *estiver habilitado,* ***modo instantâneo*** *ficará desabilitado*

Para colocar este projeto em prática iremos usar a seguinte alocação de I/O

Pino do arduino	*Dispositivo controlado*
2	*Sensor de presença lateral*
3	*Sensor de presença parte de baixo*
4	*Relê controlador da eletroválvula*
13	*Led de sinalização*
5	*Buzzer para alarme sonoro*

Observações

- ✓ *O LED de sinalização ligará enquanto a vazão de água estiver ativada.*
- ✓ *O Alarme sonoro será ativado se a vazão de água ficar ativada por mais de 3 minutos, sua finalidade será alertar o usuário que o desperdício de água pode estar acontecendo.*

DESAFIO DO MESTRE 2 – SECADOR DE MÃOS

Você está iniciando o desenvolvimento da programação de um microcontrolador que irá fazer a gestão de funcionamento de um secador de mãos automatizado. Geralmente esses aparelhos são utilizados em banheiros de shoppings, comércios, indústrias, bares e etc.

<u>Suas Tarefas para resolver este desafio serão:</u>

1. *Elaborar a programação na IDE do Arduino e validar o funcionamento na prática utilizando o shield para Arduino Starter do Eletrônica Fácil como hardware de teste rápido.*
2. *Testar o passo a passo de funcionamento deste projeto utilizando o software Proteus.*
3. *Postar fotos, vídeos, códigos desenvolvidos com as #ERDINGER #DesafioDoMestre #FamiliaEletronicaFacil #Desafio2 na comunidade E. Fácil para compartilhar o seu sucesso com os demais alunos de turma e comemorarmos juntos a sua vitória*

O funcionamento do equipamento deverá acontecer da seguinte maneira:

- ✓ Na área onde o usuário coloca suas mãos para secarem, existem dois sensores de presença, um para a mão esquerda (sensor 1) e outro para a mão direita (sensor 2). Ao aproximar as mãos os sensores serão ativados, ou individualmente ou simultâneos. Ao serem ativados, um sinal digital será enviado para o microcontrolador que, imediatamente, ligará o ar quente para iniciar o processo de secagem das mãos do usuário.

✓ No secador de mãos existe um display de 7 segmentos que marcará o tempo, em contagem regressiva de 9 a 0 segundos, que será o tempo de secagem recomendado ao usuário para que o processo aconteça com sucesso. Porém, antes de o usuário colocar suas mãos no secador, o segmento central do display (segmento G) ficará aceso , indicando que a contagem ainda não foi iniciada e que nenhuma mão foi detectada (secador em stand-by).

✓ Se a secagem das mãos estiver ativa, um LED de sinalização irá ligar indicando que o processo está ativo. Caso a secagem seja interrompida, o LED será desligado.

✓ Enquanto o usuário estiver com as mãos no secador, a contagem regressiva de 9 – 0 irá acontecer e o secador ficará ligado. Se acabarem os 9 segundos e o usuário mantiver as mãos no secador, a secagem continua, porém o número 0 ficará piscando em 2 Hz indicando o termino do tempo sugerido.

✓ Caso o usuário fique por mais de **20 segundos** utilizando o secador, um alarme sonoro será ativado.

✓ Se o usuário utilizar o aparelho por mais de **30 segundos**, automaticamente por segurança o secador desligará e só voltará ao funcionamento se o usuário retirar as **DUAS MÃOS** do equipamento, reiniciando o processo de secagem das mãos do início.

✓ Se o usuário retirar as mãos antes do tempo total acabar, o processo voltará ao início. Não é necessário completar o ciclo de 9 segundos para que exista a reinicialização do equipamento.

Para colocar este projeto em prática iremos usar a seguinte alocação de I/O

Pino do arduino	*Dispositivo controlado*
3	*Sensor de presença 1*
2	*Sensor de presença 2*
4	*Relê controlador de secagem*
6	*Led de sinalização*
5	*Buzzer para alarme sonoro*
7 a 13	*Display de 7 segmentos*

DESAFIO DO MESTRE 3 – ENVASADORA DE CHOPP

Você, desenvolvedor(a) de projetos eletrônicos com microcontroladores, ficou encarregado de realizar a programação de um microcontrolador que irá controlar uma envasadora de chopp que será utilizada em cervejarias, choperias, bares, distribuidoras e etc.

Suas Tarefas para resolver este desafio serão:

4. Elaborar a programação na IDE do Arduino e validar o funcionamento na prática utilizando o shield para Arduino Starter do Eletrônica Fácil como hardware de teste rápido.

5. Testar o passo a passo de funcionamento deste projeto utilizando o software Proteus.

6. Postar fotos, vídeos, códigos desenvolvidos com as #ERDINGER #DesafioDoMestre #FamiliaEletronicaFacil #Desafio2 na comunidade E. Fácil para compartilhar o seu sucesso com os demais alunos de turma e comemorarmos juntos a sua vitória

Veja abaixo as opções de volumes de chopps e que será envasado pela máquina que você deverá programar.

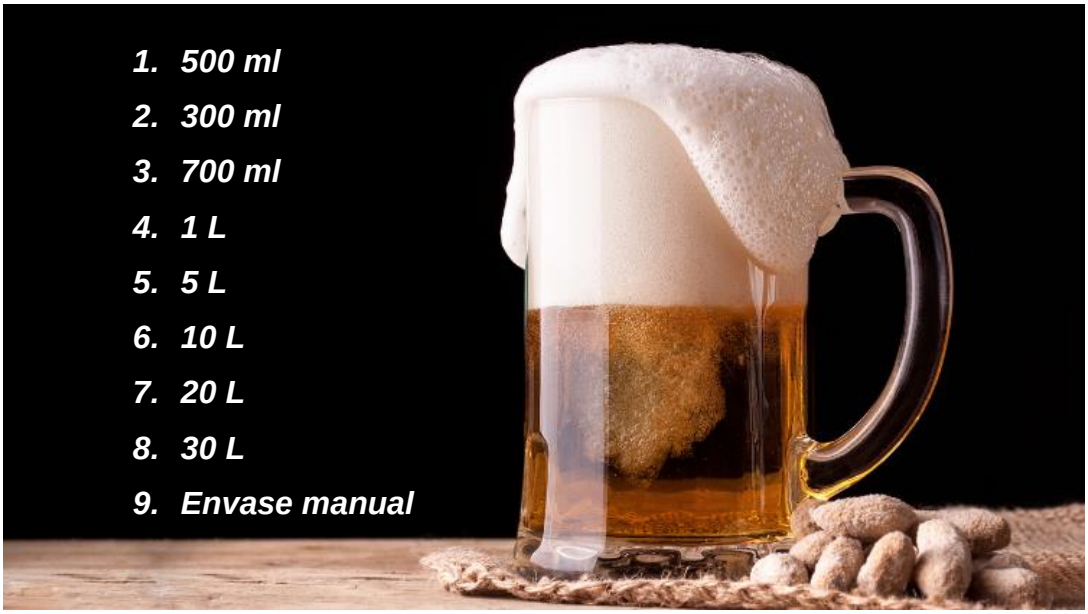


Figura 96 - Opções de envasamento

Na envasadora existirão **dois botões, chamados de volume e start**

O botão de **volume** irá selecionar a opção de 1 a 9 antes de realizar o processamento de envasamento da bebida. A cada toque no botão volume irá incrementar os valores de 1 a 9 para selecionar a opção de volume conforme figura acima. Ao chegar no número 9 e o botão continuar sendo pressionado a seleção volta para o número 1 (0,5 L).

Ao pressionar o botão start, o LED indicador irá piscar em 4 Hz por 3 segundos indicando que o envasamento do chopp está prestes a iniciar. Durante o envasamento do chopp o LED permanecerá ligado. Ao terminar o envasamento do chopp o LED será desligado.

ATENÇÃO - *Se durante o envase do chopp o botão start for pressionado novamente, o sistema de envase deve ser desligado imediatamente por segurança e o processo de seleção voltará ao início, inclusive exibindo o número 1 no display de seleção (0,5 L);*

Para colocar este projeto em prática iremos usar a seguinte alocação de I/O

Pino do arduino	*Dispositivo controlado*
2	*Botão volume*
3	*Botão Start – stop*
4	*Relê controlador de envase*
13 - 7	*Display de 7 segmentos*
6	*Led de sinalização*
5	*Buzzer alarme sonoro*

Observação: Todas as vezes que a máquina parar de envasar o chopp, um o buzzer deverá ligar por 0,5 segundo indicando o termino do processo.

Para cada volume de chopp, deixe o relé ativado pelo seguinte tempo

1. *500 ml*	*2 s*
2. *300 ml*	*1,2 s*
3. *700 ml*	*2,8 s*
4. *1 L*	*4 s*
5. *5 L*	*20 s*
6. *10 L*	*40 s*
7. *20 L*	*80 s*
8. *30 L*	*120 s*

9. *Ligar envase enquanto o botão de start estiver ativado (Limite de segurança 120 segundos).*

Terminado o módulo 4 do Curso de Eletrônica Digital e Arduino para iniciantes onde explicamos a todos o passo a passo para colocar em prática o método ERDINGER de aprendizagem aplicado a leitura de entradas digitais, chegou a hora de irmos ao módulo do 5 do curso onde vamos desbravar ainda mais a programação do nosso querido e amado Arduino Uno de forma incrível, show de bola e sensacional. Vejo vocês lá!

CLIQUE AQUI PARA ACESSAR A AULA