



TRABALHO DE GRADUAÇÃO

Proposta de Atualização Curso de Algoritmos e Programação  
de Computadores Utilizando Placa Intel<sup>®</sup> Galileo

Luiz Fernando de Andrade Gadêlha

Brasília, Junho de 2016

**UNIVERSIDADE DE BRASÍLIA**

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA  
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

**Proposta de Atualização Curso de Algoritmos e Programação  
de Computadores Utilizando Placa Intel<sup>®</sup> Galileo**

**Luiz Fernando de Andrade Gadêlha**

*Relatório submetido ao Departamento de Engenharia  
Elétrica como requisito parcial para obtenção  
do grau de Engenheiro Mecatrônico*

Banca Examinadora

Prof. Alexandre Zaghetto, CIC/UnB  
*Orientador*

\_\_\_\_\_

Carla Castanho, CIC/UnB

\_\_\_\_\_

Prof. Flávio Vidal, CIC/UnB

\_\_\_\_\_

## **Dedicatória**

*Dedico este trabalho em primeiro lugar a Deus, por todas bençãos que me fizeram continuar e todas dificuldades que me fizeram crescer. Dedico este trabalho também à minha família, que esteve comigo em todos momentos da minha formação.*

*Luiz Fernando de Andrade Gadêlha*

## Agradecimentos

*Agradeço a meus colegas de curso, projetos e ao meu professor orientador por este trabalho*

*Luiz Fernando de Andrade Gadêlha*

---

## RESUMO

Este trabalho tem como objetivo propor um curso prático em Algoritmos e Programação de Computadores Utilizando a placa Intel<sup>®</sup> Galileo voltada para alunos de graduação dos curso de engenharia mecatrônica, elétrica e de computação. Tal proposta se fundamenta em teorias pedagógicas modernas de aprendizagem ativa e masterização de habilidades. Neste documento são descritas tais teorias e são propostas 8 práticas laboratoriais com a placaIntel<sup>®</sup> Galileo com nível de complexidade crescente em termos de conhecimentos em programação e eletrônica.

---

## ABSTRACT

This paper aims to propose a practical course in Algorithms and Computer Programming Using Intel<sup>®</sup> plate Galileo toward undergraduate students of course mechatronics, electrical engineering and computing. This proposal is based on modern pedagogical theories of active learning skills and mastering. In this document are described such theories are proposed and 8 laboratory practices with the Intel<sup>®</sup> board Galileo with increasing level of complexity in terms of knowledge in programming and electronics.

# SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b>	<b>1</b>
1.1	OBJETIVO	2
1.2	APRESENTAÇÃO DO MANUSCRITO	3
<b>2</b>	<b>REVISÃO BIBLIOGRÁFICA</b>	<b>4</b>
2.1	TEORIAS PEDAGÓGICAS	4
2.1.1	APRENDIZAGEM ATIVA	4
2.1.2	APRENDIZAGEM POR MASTERIZAÇÃO	6
2.1.2.1	INTRODUÇÃO	6
2.1.2.2	TAXONOMIA DOS OBJETIVOS EDUCACIONAIS - TAXONOMIA DE BLOOM	9
2.1.2.3	PROGRAMA DESENVOLVIDO EM QT PARA AUXÍLIO NA DEFINIÇÃO DE OBJETIVOS EDUCACIONAIS	10
2.2	METODOLOGIAS PARA DESENVOLVIMENTO DE PROJETOS	15
2.2.1	METODOLOGIA CASCATA	15
2.2.2	METODOLOGIA SCRUM	16
2.3	ALGORITMOS E PROGRAMAÇÃO DE COMPUTADORES	19
2.3.1	INTRODUÇÃO	19
2.3.2	CURRÍCULOS	20
2.4	PLACA INTEL <sup>®</sup> GALILEO	22
2.4.1	PINAGEM DA PLACA INTEL <sup>®</sup> GALILEO	22
2.4.2	CARACTERÍSTICAS ELÉTRICAS E ELETRÔNICAS DA PLACA INTEL <sup>®</sup> GALILEO	23
2.4.2.1	SINAL PWM	25
2.4.2.2	CONVERSÃO ANALÓGICO-DIGITAL	27
2.4.2.3	BARRAMENTO SERIAL I2C	28
2.4.2.4	COMUNICAÇÃO SERIAL SPI	30
2.4.2.5	PORTA SERIAL UART	31
2.4.2.6	MEMÓRIA CACHE	33
2.4.2.7	MEMÓRIA SRAM	34
2.4.2.8	MEMÓRIA DRAM	36
2.4.2.9	MEMÓRIA EEPROM	37
2.4.2.10	CLOCK DE TEMPO REAL - RTC	37
2.4.2.11	BARRAMENTO MINI PCI-EXPRESS	37
2.4.2.12	USB	38

2.4.2.13	JTAG .....	40
<b>3</b>	<b>PROPOSTA DE ATUALIZAÇÃO DO CURSO DE APC .....</b>	<b>42</b>
3.1	APRENDIZAGEM POR MASTERIZAÇÃO E SCRUM ADAPTADOS AO CURSO DE ALGORITMOS E PROGRAMAÇÃO DE COMPUTADORES .....	44
3.1.1	DEFINIÇÃO DE OBJETIVOS EDUCACIONAIS NA FORMA DA METODOLOGIA SCRUM	47
3.1.2	DEFINIÇÃO DE OBJETIVOS EDUCACIONAIS PARA A DISCIPLINA <i>Algoritmos e Programação de Computadores</i> .....	51
3.1.3	DEFINIÇÃO DE OBJETIVOS EDUCACIONAIS PARA A DISCIPLINA <i>Algoritmos e Programação de Computadores</i> COM RELAÇÃO ÀS ATIVIDADES PRÁTICAS.....	54
3.1.4	PROPOSTA DE USO DE PLACAS DE PROTOTIPAGEM ELETRÔNICA .....	56
3.1.5	FORMA DE AVALIAÇÃO .....	57
<b>4</b>	<b>ELEMENTOS DE CIRCUITOS E PROGRAMAÇÃO .....</b>	<b>58</b>
4.1	INTRODUÇÃO .....	58
4.2	CIRCUITOS ELETRÔNICOS.....	58
4.2.1	RESISTOR .....	58
4.2.2	CIRCUITO - FÓRMULAS E TOPOLOGIAS BÁSICAS .....	60
4.2.2.1	PRIMEIRA LEI DE KIRCHOFF - LEIS DAS NÓS .....	61
4.2.2.2	SEGUNDA LEI DE KIRCHOFF - LEIS DAS MALHAS .....	61
4.2.2.3	LEI DE OHM .....	62
4.2.2.4	LIGAÇÃO EM SÉRIE.....	63
4.2.2.5	LIGAÇÃO EM PARALELO .....	64
4.2.2.6	RESUMO - LIGAÇÃO EM SÉRIE X PARALELO .....	64
4.2.3	DIODO-LED .....	64
4.2.4	PROTOBOARD.....	67
4.2.5	DIVISOR DE TENSÃO.....	68
4.2.6	POTENCIÔMETRO .....	70
4.2.7	LDR .....	73
4.2.8	INTERRUPTORES .....	75
4.2.9	REGISTRADOR DE DESLOCAMENTO (SHIFT REGISTER) .....	78
4.2.10	SENSOR DE TEMPERATURA - LM35.....	80
4.2.11	CAPACITORES.....	82
4.2.12	FILTRO RC .....	83
4.2.13	INVERSOR SCHMITT TRIGGER .....	86
4.2.14	DEBOUNCING - ESTABILIZAÇÃO DE SINAIS DE INTERRUPTORES .....	90
4.2.15	MATRIZ DE LEDS .....	92
4.2.16	DRIVER DE MATRIZ DE LED .....	93
4.3	SOFTWARE .....	95
4.3.1	PROGRAMAÇÃO ESTRUTURADA.....	95
4.3.2	PROGRAMAÇÃO PARA ARDUINO.....	96
4.3.2.1	PADRÃO DE PROGRAMAÇÃO EM ARDUÍNO.....	96

4.3.3	USO DE PORTAS DIGITAIS .....	97
4.3.4	PROCESSO DE COMPILAÇÃO .....	98
4.3.4.1	PRÉ-PROCESSAMENTO.....	99
4.3.4.2	ANÁLISE LÉXICA .....	100
4.3.4.3	ANÁLISE SINTÁTICA .....	100
4.3.4.4	ANÁLISE SEMÂNTICA.....	100
4.3.5	DIRETIVAS DE COMPILAÇÃO .....	101
4.3.6	TIPOS BÁSICOS DE VARIÁVEIS.....	102
4.3.7	CONVERSÃO ENTRE TIPOS DE VARIÁVEIS .....	104
4.3.8	LEITURA ANALÓGICA - CONVERSÃO ANALÓGICO/DIGITAL .....	105
4.3.9	ESTRUTURAS CONDICIONAIS.....	105
4.3.9.1	OPERADORES RELACIONAIS.....	105
4.3.9.2	OPERADORES LÓGICOS .....	106
4.3.9.3	CONTROLE DE FLUXO DE EXECUÇÃO DE UM PROGRAMA (CONDICIONAIS) .....	108
4.3.10	LAÇOS DE REPETIÇÃO.....	114
4.3.10.1	LAÇO FOR.....	115
4.3.10.2	LAÇO WHILE .....	118
4.3.10.3	LAÇO DO-WHILE .....	119
4.3.11	VETORES E MATRIZES .....	121
4.3.11.1	VETORES .....	121
4.3.11.2	MATRIZES .....	123
4.3.12	TIPOS AVANÇADOS DE VARIÁVEIS.....	124
4.3.12.1	BOOLEAN .....	125
4.3.12.2	PONTEIROS.....	126
4.3.12.3	STRING E ARRAYS .....	128
4.3.12.4	REGISTROS - STRUCT .....	131
4.3.13	CONVERSÃO CÓDIGO BINÁRIO PARA DECIMAL E VICE-VERSA.....	135
4.3.14	SUBALGORITMOS (FUNÇÕES) .....	137
4.3.15	INTERRUPÇÃO .....	140
4.3.15.1	INTERRUPÇÃO DE HARDWARE .....	141
4.3.15.2	INTERRUPÇÃO DE SOFTWARE - TIMER .....	144
4.3.16	NUMERO RANDÔMICO .....	146
4.3.17	FUNÇÕES E BIBLIOTECAS DIVERSAS - ARDUINO .....	147
4.3.17.1	FUNÇÃO SHIFTOUT.....	147
4.3.17.2	BIBLIOTECA LEDCONTROL.H .....	148
<b>5</b>	<b>LABORATÓRIOS PROPOSTOS .....</b>	<b>153</b>
5.1	INTRODUÇÃO .....	153
5.2	PRÁTICA 1: COMEÇANDO A USAR O <i>Galileo</i> .....	153
5.3	PRÁTICA 2: INTRODUÇÃO A LEITURA DE SENSORES E TIPOS DE VARIÁVEIS .....	155
5.4	PRÁTICA 3: USO DE CHAVES/BOTÕES E CONTROLADORES DE FLUXO(CONDICIONAIS) .....	157
5.5	PRÁTICA 4: USO DE LAÇOS DE REPETIÇÃO .....	159

5.6	PRÁTICA 5: USO DE VETORES, SHIFT REGISTER E TIPOS VARIADOS DE DADOS	161
5.7	PRÁTICA 6: USO DE FUNÇÕES E SENSOR DE TEMPERATURA .....	163
5.8	PRÁTICA 7: USO DE STRUCTS, INTERRUPÇÃO DE HARDWARE E CIRCUITO DE DEBOUNCE DE SINAIS.....	165
5.9	PRÁTICA 8: PRÁTICA FINAL .....	167
<b>6</b>	<b>CONCLUSÕES .....</b>	<b>169</b>
	<b>REFERÊNCIAS BIBLIOGRÁFICAS .....</b>	<b>170</b>
	<b>APÊNDICES.....</b>	<b>173</b>
<b>I</b>	<b>APÊNDICES.....</b>	<b>174</b>
I.1	PRÁTICA 1: COMEÇANDO A USAR O <i>Galileo</i> .....	174
I.1.1	PROCEDIMENTOS.....	174
I.1.2	ESQUEMA DE MONTAGEM.....	175
I.1.3	CÓDIGO FONTE.....	175
I.1.4	COMENTÁRIOS .....	176
I.2	PRÁTICA 2: INTRODUÇÃO A LEITURA DE SENSORES E TIPOS DE VARIÁVEIS ....	177
I.2.1	PROCEDIMENTOS.....	177
I.2.2	ESQUEMA DE MONTAGEM.....	177
I.2.3	CÓDIGO FONTE.....	178
I.2.4	COMENTÁRIOS .....	179
I.3	PRÁTICA 3: USO DE CHAVES/BOTÕES E CONTROLADORES DE FLUXO(CONDICIONAIS)	180
I.3.1	PROCEDIMENTOS.....	180
I.3.2	ESQUEMA DE MONTAGEM.....	180
I.3.3	CÓDIGO FONTE.....	181
I.3.4	COMENTÁRIOS .....	182
I.4	PRÁTICA 4: USO DE LAÇOS DE REPETIÇÃO .....	183
I.4.1	PROCEDIMENTOS.....	183
I.4.2	ESQUEMA DE MONTAGEM.....	183
I.4.3	CÓDIGO FONTE.....	184
I.4.4	COMENTÁRIOS .....	187
I.5	PRÁTICA 5: USO DE VETORES, SHIFT REGISTER E TIPOS VARIADOS DE DADOS	188
I.5.1	PROCEDIMENTOS.....	188
I.5.2	ESQUEMA DE MONTAGEM.....	188
I.5.3	CÓDIGO FONTE.....	190
I.5.4	COMENTÁRIOS .....	192
I.6	PRÁTICA 6: USO DE FUNÇÕES E SENSOR DE TEMPERATURA .....	193
I.6.1	PROCEDIMENTOS.....	193
I.6.2	ESQUEMA DE MONTAGEM.....	193
I.6.3	CÓDIGO FONTE.....	194
I.6.4	COMENTÁRIOS .....	196

I.7	PRÁTICA 7: USO DE STRUCTS, INTERRUPÇÃO DE HARDWARE E CIRCUITO DE DEBOUNCE DE SINAIS.....	197
I.7.1	PROCEDIMENTOS.....	197
I.7.2	ESQUEMA DE MONTAGEM.....	197
I.7.3	CÓDIGO FONTE.....	198
I.7.4	COMENTÁRIOS.....	202
I.8	PRÁTICA 8: PRÁTICA FINAL.....	203
I.8.1	PROCEDIMENTOS.....	203
I.8.2	ESQUEMA DE MONTAGEM.....	203
I.8.3	CÓDIGO FONTE.....	204
I.8.4	COMENTÁRIOS.....	210
<b>II</b>	<b>DESCRIÇÃO DO CONTEÚDO DO CD.....</b>	<b>211</b>

# LISTA DE FIGURAS

2.1	Níveis de aprendizagem no domínio cognitivo [1].....	9
2.2	Verbos de descrição de objetivos educacionais a serem usados em cada nível cognitivo.	9
2.3	Tela inicial do aplicativo desenvolvido em QT para definir os objetivos educacionais.	10
2.4	Tela inicial do aplicativo desenvolvido em QT para definir os objetivos educacionais, retângulo vermelho marcando a região onde o objetivo educacional definido é mostrado.	11
2.5	Definindo uma unidade de ensino.....	11
2.6	Definindo um objetivo de estudo para a unidade de estudo. ....	12
2.7	Retângulo amarelo destacando os níveis cognitivos. ....	12
2.8	Selecionando um verbo de um nível cognitivo. ....	13
2.9	Selecionando a amplitude do estudo a ser realizado. ....	13
2.10	Selecionando o recursos a ser utilizado para alcançar o objetivo educacional especificado.....	14
2.11	Selecionando um produto a ser desenvolvido como atividade de masterização.....	14
2.12	Selecionando a quantidade de pessoas por grupo. ....	15
2.13	Fluxo da metodologia de desenvolvimento de projeto <i>Cascata</i> .....	16
2.14	Porcentagem dos conteúdos tratados na UnB e <i>California Institute of Technology</i> que podem ser ensinados utilizando a placa Galileo.....	22
2.16	Descrição dos pinos da placa Galileo - parte traseira[2] .....	23
2.15	Descrição dos pinos da placa Galileo - parte frontal[2].....	23
2.17	Sinal PWM.....	25
2.18	Figura esquemática do processo de conversão analógico para digital.....	27
2.19	Figura esquemática do barramento serial I2C.....	29
2.20	I2C - <i>Clock Stretching</i> .....	29
2.21	Barramento SPI - Um <i>dispositivo mestre</i> para <i>dispositivos escravos</i> .....	30
2.22	Modelo simplificado de um dispositivo UART .....	32
2.23	Frame UART para transmissão de 1 byte .....	32
2.24	Modelo completo de um dispositivo UART .....	33
2.25	Organização da memória num sistema computacional. ....	34
2.26	Estrutura da célula de memória SRAM.....	35
2.27	Estrutura da célula SRAM com dois inversores.....	35
2.28	Estrutura da célula de memória DRAM. ....	36
2.29	Estrutura da célula de memória DRAM. ....	38
2.30	Exemplo: Topologia Estrela USB .....	39

2.31	Pinos USB .....	39
2.32	JTAG monitorando a conexão de um CPU com uma FPGA .....	40
2.33	Máquina de estados - JTAG .....	40
2.34	Fluxo de dados num debug JTAG.....	41
3.1	Plataforma scrumdo: Formato dos quadros Scrum planejados para a disciplina <i>Algoritmos e Programação de Computadores</i> .....	49
3.2	Plataforma scrumdo: Quadros Scrum criados e objetivos educacionais, segundo a Taxonomia de Bloom escritos <i>Algoritmos e Programação de Computadores</i> .....	50
3.3	Plataforma scrumdo: Definição de sub-objetivos de um objetivo listado no <i>Product Backlog Algoritmos e Programação de Computadores</i> .....	50
4.1	Exemplo de resistor utilizado em circuitos eletrônicos. ....	59
4.2	Símbolos de resistor utilizado em circuitos eletrônicos. ....	59
4.3	Como ler o código de cores de um resistor. ....	60
4.4	Circuito Simples.....	61
4.5	Primeira Lei de Kirchoff. ....	62
4.6	Segunda Lei Kirchhoff. ....	62
4.7	Ligação em série de n resistores. ....	63
4.8	Mesma corrente percorrendo resistores em série. ....	63
4.9	Componentes conectados em paralelo. ....	65
4.10	Junção N-P em um diodo.....	65
4.11	Símbolo Diodo.....	66
4.12	Relação tensão, corrente num diodo e suas regiões de operação .....	66
4.13	Modelo simplificado de um diodo em polarização direta.....	67
4.14	Protoboard. ....	68
4.15	Esquemático de um circuito e sua construção numa <i>protoboard</i> . ....	68
4.16	Modelo de um circuito divisor de tensão .....	69
4.17	Símbolo de um potenciômetro. ....	70
4.18	Estrutura de um potenciômetro angular.....	70
4.19	Potenciômetro linear. ....	71
4.20	Formas de variação da resistência em um potenciômetro.....	71
4.21	Formas de uso de um potenciômetro angular. ....	72
4.22	Circuito integrado de um joystick que utiliza dois potenciômetros para tradução de sua posição em tensão. ....	72
4.23	Circuito esquemático de um circuito utilizando um joystick similar ao mostrado na Figura 4.22 com a placa Galileo.....	73
4.24	LDR.....	73
4.25	Símbolo do LDR .....	74
4.26	Gráfico LDR : Resistência X Luminância .....	74
4.27	Possível circuito a ser utilizado num sensor de luz .....	75
4.28	Circuito aberto. ....	76
4.29	Circuito fechado.....	76

4.30	Símbolo de um interruptor. ....	76
4.31	Mini botões de pressão, de 2 e 4 pinos. ....	77
4.32	Circuito simples utilizando um botão de 4 pinos de contato ....	78
4.33	Esquema resumido: Série para Paralelo com um registrador de deslocamento.....	78
4.34	Cascata de <i>flip-flops</i> num registrador de deslocamento. ....	79
4.35	Pinagem Registrador de Deslocamento 74HC595.....	79
4.36	Sensor de Temperatura LM35. ....	81
4.37	Pinagem LM35. ....	81
4.38	Circuito sensor de temperatura feito com LM35 e com a placa Galileo.....	82
4.39	Estrutura básica de um capacitor ....	82
4.40	Circuito com um capacitor ....	83
4.41	Filtro RC.....	84
4.42	Evolução da tensão no capacitor e no resistor no tempo, num circuito RC.....	85
4.43	Diagrama de bode de um filtro RC. ....	86
4.44	Sinal com histerese. ....	87
4.45	Implementação de um inversor com histerese com amplificador operacional e diodos zener. ....	87
4.46	Inversor Schmitt trigger 74HC14 [3] - pinos. ....	89
4.47	Inversor Schmitt trigger 74HC14 [3] - diagramas lógicos. ....	89
4.48	Inversor Schmitt trigger 74HC14 [3] - parâmetros eletrônicos e curva de histerese. ....	90
4.49	Fenômeno de <i>bouncing</i> de um sinal de um interruptor.....	90
4.50	Interrupções de rampa de subida causadas pelo fenômeno do <i>bouncing</i> . ....	91
4.51	Circuito para <i>debouncing</i> de um sinal de um interruptor.....	91
4.52	Resultado do uso do circuito de <i>debouncing</i> de um sinal de um interruptor.....	92
4.53	Estrutura esquemática de uma matriz de leds 8x8. ....	93
4.54	Pinos do circuito integrado MAX7219.....	94
4.55	Conexão dos pinos do MAX7219 à placa Galileo. ....	94
4.56	Placa matriz 8x8 + MAX7219. ....	95
4.57	Fluxograma de um programa escrito para Arduino.....	97
4.58	Fluxograma de um programa escrito para Arduino.....	99
4.59	Exemplo de substituição de macro no pré-processamento.....	100
4.60	Resumo do controle de fluxo de um programa ....	109
4.61	Sintaxe de uso do loop for. ....	116
4.62	Exemplo de uso do loop for. ....	116
4.63	Sintaxe comum de uso do loop while. ....	118
4.64	Sintaxe comum de uso do loop do-while. ....	120
4.65	Alocação de memória para um vetor declarado para N espaços de memória. ....	121
4.66	Endereço de memória e conteúdo de um vetor <i>v</i> de 8 espaços. ....	127
4.67	Após a atribuição do endereço do vetor ao ponteiro ptrInt. ....	127
4.68	Exemplo de aplicação da técnica de divisões sucessivas. ....	136
4.69	Fluxo normal de um programa com uma interrupção. ....	140
4.70	Interrupções de rampa de subida e descida. ....	141

4.71	Circuito associado à interrupção de rampa de subida. ....	142
4.72	Circuito associado à interrupção de rampa de descida. ....	142
4.73	Website - <a href="http://playground.arduino.cc/Code/Timer1">http://playground.arduino.cc/Code/Timer1</a> . ....	144
4.74	Download e instruções para instalação da biblioteca Timer1.....	145
4.75	GUI do Arduino aberta. ....	148
4.76	Caminho para a aba <i>Manage Libraries</i> .....	149
4.77	Janela <i>Manage Libraries</i> . ....	149
4.78	Resultado da busca por <i>LedControl</i> . ....	150
4.79	Clique no <i>Install</i> . ....	150
4.80	Biblioteca LedControl.h instalada. ....	151
I.1	Circuito da prática 1.....	175
I.2	Esquemático do circuito da prática 2.....	177
I.3	Circuito da prática 2 construído numa protoboard. ....	178
I.4	Esquemático do circuito da prática 3.....	180
I.5	Circuito da prática 3 construído numa protoboard. ....	181
I.6	Esquemático do circuito da prática 4.....	183
I.7	Circuito da prática 4 construído numa protoboard. ....	184
I.8	Esquemático do circuito da prática 5.....	189
I.9	Circuito da prática 5 construído numa protoboard. ....	190
I.10	Esquemático do circuito da prática 6.....	193
I.11	Circuito da prática 5 construído numa protoboard. ....	194
I.12	Esquemático do circuito da prática 7.....	197
I.13	Circuito da prática 7 construído numa protoboard. ....	198
I.14	Esquemático do circuito da prática 8.....	203
I.15	Circuito da prática 8 construído numa protoboard. ....	204

# LISTA DE TABELAS

2.1	Pontos positivos e negativos da metodologia de desenvolvimento de projetos <i>Cascata</i> [4].	17
2.2	Pontos positivos e negativos da metodologia de desenvolvimento de projetos <i>Scrum</i> .	19
2.3	Tabela com scores de avaliação das 5 melhores universidades do mundo e das 5 melhores universidades brasileiras.	20
2.4	Tabela listando as disciplinas na UnB e na <i>California Insitute of Tecnology</i> que possuem currículo similar a disciplina <i>Algoritmos e Programação de Computadores</i> .	21
2.5	Tabela listando parte dos conteúdos tratados pelas disciplinas na <i>California Insitute of Tecnology</i> e na UnB e levantando a questão da metodologia de ensino ser uma <i>metodologia ativa</i> e se tal conteúdo pode ser ensinado usando a placa Galileo.	21
2.6	Pinos Galileo[2].	24
3.1	Proposta de ementa	43
3.2	Scrum aplicado à <i>Algoritmos e Programação de Computadores</i>	44
4.1	Tabela de leitura de valor e tolerância de um resistor.	60
4.2	Resumo - Ligação em série X Ligação em paralelo.	65
4.3	Tamanho e faixa de uso para os tipos básicos variáveis.	103
4.4	Todos os Tipos de dados definidos pelo Padrão ANSI C, seus tamanhos em bytes e suas faixa de valores.	104
4.5	Operadores relacionais.	106
4.6	Operadores <b>E</b> , <b>OU</b> e <b>Negação</b>	107
4.7	Tabela verdade do operador <code>  </code> .	107
4.8	Tabela verdade do operador <code>&amp;&amp;</code>	108
4.9	Tabela verdade do operador <code>!</code>	108
4.10	Tipos básicos e avançados ( destacados em vermelho) de variáveis.	125
5.1	Prática 1 - Tabela de descrição.	154
5.2	Prática 2 - Tabela de descrição.	156
5.3	Prática 3 - Tabela de descrição.	158
5.4	Prática 4 - Tabela de Descrição	160
5.5	Prática 5 - Tabela de Descrição	162
5.6	Prática 6 - Tabela de Descrição	164
5.7	Prática 7 - Tabela de Descrição	166

5.8 Prática 8 - Tabela de Descrição.....	168
--	-----

# Capítulo 1

## Introdução

Este trabalho tem por objetivo trazer uma proposta de metodologia de curso para a disciplina *Algoritmos e Programação de Computadores* expandindo o conteúdo ensinado para, além de computação básica, também eletrônica básica. A proposta de expansão para eletrônica básica é feita baseando-se na placa de desenvolvimento de circuitos Intel<sup>®</sup> Galileu. A placa Galileo é escolhida por causa da sua facilidade de programação na linguagem de programação C e facilidade de prototipagem de circuitos.

Como embasamento de tal proposta, a seguinte análise dos paradigmas econômicos e educacionais se faz útil: A economia mundial está passando por uma grande revolução neste século. As bases econômicas de muitos países, outrora baseadas em *extração de matérias-primas e indústrias de transformação* de tais matérias primas, são agora baseadas em *conhecimento e transmissão de informação* [5]. O desenvolvimento tecnológico da Ciência da Computação é a principal responsável por tal revolução e todas Engenharias e Ciências Exatas são, direta ou indiretamente, influenciados por ela. Neste contexto, são fundamentais os conhecimentos e habilidades relacionados a Ciência da Computação para o desenvolvimento de todas as Engenharias. Em especial, é importante a revisão da sua forma de ensino e aprendizagem para a realidade na qual vivemos atualmente [6].

A educação como a conhecemos atualmente foi idealizada na Prússia, no final do século XVII. Tal modelo educacional é chamada de *aprendizagem centrada no professor* ou *aprendizagem passiva*. No modelo de *aprendizagem passiva*, os estudantes são meros receptores do conhecimento oriundo do professor. Tal modelo se adequou bem às necessidades da época, as quais se resumiam em treinar trabalhadores em habilidades ligadas à pura repetição e obediência [7].

Hoje em dia, principalmente por causa da revolução iniciada pela computação supracitada, boa parte das universidades no mundo já começaram a modificar seus paradigmas educacionais realizando uma transição da *aprendizagem passiva* para o modelo de *aprendizagem ativa* ou *aprendizagem centrada no aluno*. Nesse modelo, o estudante é o principal responsável por sua aprendizagem e o professor é o orientador das experiências de ensino. Sob o modelo da educação ativa, tem-se conseguido obter altos índices de paradigmas ligados a criatividade, liderança, trabalho em equipe, gerenciamento e auto-gerenciamento além de um aumento substancial na motivação dos estudantes, visto que eles podem se apropriar verdadeiramente de seu processo de aprendizagem além

terem se mostrado próprios para a aprendizagem e ensino de conceitos ligados a computação[8].

A realidade da educação brasileira, no entanto, não tem acompanhado tais tendências. Técnicas eficientes de ensino de conhecimentos relacionados ao ensino de Ciências Exatas e Engenharia são parte estratégica para o desenvolvimento de qualquer país. Entretanto, as mudanças nos cursos de engenharia, no Brasil, em geral têm sido relacionadas a simples adição ou supressão de conteúdos, mas não uma revisão profunda das bases de ensino, levando em consideração as transformações atuais [9].

Segundo o jornal *A Gazeta do Povo*, a taxa de evasão nos cursos de engenharia no Brasil é de aproximadamente 57% [10] e em geral a evasão ocorre nas partes iniciais dos cursos, onde os alunos têm seu primeiro contato com computação . Pode-se afirmar, tendo em vista essa estatística, que o paradigma educacional atual é parte dos inúmeros e problemas responsáveis pelo grande déficit de engenheiros qualificados no Brasil.

Com relação ao ensino de habilidades e conceitos de computação básica, não apenas no Brasil, mas no mundo se observa que os estudantes usualmente têm grandes dificuldades de aprendizagem, que o conhecimento e aprendizagem dos alunos tende a se estagnar nos níveis mais rasos de entendimento, de forma que os conhecimentos não são interconectados, mas são apenas específicos ao contexto estudado [11]. Além disso, os alunos muitas vezes sentem desmotivados devido a fragmentação do conhecimento nas disciplinas [9, 8]. Tais problemas de aprendizagem também se mostram presentes nos profissionais que saem das faculdades. Boa parte dos profissionais, no Brasil, possuem formação deficiente. Não tem capacidade plenamente desenvolvida para serem *aprendizes-estudantes autônomos*. Essa habilidade é essencial para terem sucesso na economia mundial atual, que é centrada em conhecimento e informação [6].

Para realizar a transição entre o modelo *aprendizagem-passiva* para o modelo *aprendizagem-ativa*, muitas universidades já se utilizam de placas eletrônicas com microcontroladores como Arduino e similares [12]. Nas referências citadas, o ensino de computação básica aliada a projetos práticos tem alcançado grande aumento nos índices acadêmicos dos alunos e diminuição nas taxas de evasão.

Este trabalho tem como objetivo primário a proposta de um curso prático para a disciplina *Algoritmos e Programação de Computadores* utilizando a placa de desenvolvimento *Intel<sup>®</sup> Galileo* com dinâmicas pedagógicas próprias do paradigma de *aprendizagem-ativa* de formar a atacar os problemas elencados anteriormente com o objetivo de propor uma disciplina factível a realidade da Universidade de Brasília (UnB).

## 1.1 Objetivo

O objetivo deste trabalho é propor para a disciplina de *Algoritmos e Programação de Computadores* um modelo de curso prático de programação expandindo o conteúdo tratado no curso incluindo, além de computação, eletrônica básica.

Busca-se, por meio desta proposta de curso, oferecer um modelo pedagógico mais amplo para

o ensino de programação na linguagem C e o básico de circuitos embarcados utilizando a placa *Intel<sup>®</sup> Galileo*. Tal proposta tem como motivação tratar do problema da usual baixa profundidade na aprendizagem de programação e o problema da desmotivação dos alunos.

Neste trabalho são propostas 8 aulas práticas que contêm uma explicação aprofundada de todos conceitos tratados no laboratório - tanto os conceitos diretamente ligados à programação na linguagem C, quanto os conceitos ligados a circuitos eletrônicos. Em todas descrições do plano de laboratório são também sugeridas formas de organização pedagógicas para ampliar e aprofundar a aprendizagem dos alunos.

## 1.2 Apresentação do manuscrito

Neste manuscrito, é feito um estudo da teorias pedagógicas modernas, metodologias de desenvolvimento de projetos, currículos similares à disciplina *Algoritmos e Programação de Computadores* e as características eletrônicas da placa Intel<sup>®</sup> Galileo.

A organização deste documento é a seguinte: no capítulo 2 são apresentados as teorias pedagógicas de aprendizagem ativa (seção 2.1.1) e aprendizagem por masterização (Seção 2.1.2). Na Seção 2.1.2, é apresentada uma ferramenta desenvolvida, na plataforma de programação QT<sup>®</sup>, para auxílio na definição de objetivos educacionais. São apresentados também as metodologias de desenvolvimento e projetos em cascata (Seção 2.2.1) e Scrum (Seção 2.2.2).

Após essas seções, é feito um estudo de currículos similares à *Algoritmos e Programação de Computadores* nas 5 melhores do mundo e das 5 melhores universidades no Brasil verificando a possibilidade de revisão metodológica com a inclusão de uma placa de prototipagem similar a placa Intel<sup>®</sup> Galileo (Seção 2.3).

Após os estudos dos currículos similares à *Algoritmos e Programação de Computadores*, é apresentada a estrutura eletrônica da placa Intel<sup>®</sup> Galileo (seção 2.4) em detalhes.

Finalmente, é apresentada a proposta de curso (Seção 3) tendo como fundamentação as teorias de aprendizagem ativa e aprendizagem por masterização sob a ótica de planejamento de projeto Scrum.

O capítulo 4 apresenta toda a fundamentação teórica, reativas a eletrônica e computação, das práticas propostas neste trabalho.

O capítulo 5 apresentada 8 práticas propostas com a placa Galileo, partindo de conhecimentos iniciais até conhecimentos avançados de programação em C.

O capítulo 6 apresenta as conclusões deste trabalho propondo trabalhos futuros como a aplicação da disciplina proposta para verificação da efetividade de tais propostas e planejamento de novas práticas.

## Capítulo 2

# Revisão Bibliográfica

O objetivo deste capítulo é explicitar todos conceitos relevantes deste trabalho relativos a ensino e aprendizagem de *Algoritmos e Programação de Computadores* (APC) e a estrutura detalhada da placa Intel<sup>®</sup> Galileu. Esses estudos servirão de base para a proposição um modelo de aula de APC para estudantes de graduação levando em conta os paradigmas de educação mais eficientes dentre os elencados.

### 2.1 Teorias Pedagógicas

Este trabalho tem como proposta a atualização do curso de *Algoritmos e Programação de Computadores* seguindo as transformações atuais que apontam para métodos de ensino focados em *educação ativa*. Nesta seção, são apresentados:

- Estudo da teoria pedagógica de aprendizagem ativa focando na teoria de *aprendizagem por masterização*;
- Estudo da metodologia SCRUM para desenvolvimento ágil de projetos;
- Estudo de currículos relacionadas a *Algoritmos e Programação de Computadores*.

#### 2.1.1 Aprendizagem Ativa

A aprendizagem ativa é geralmente definida como qualquer método de instrução que envolve os alunos diretamente no processo de aprendizagem. Sob o método de aprendizagem ativa, é demandado dos alunos que eles façam atividades de aprendizagem significativas e constantemente reflitam sobre o que estão aprendendo, como estão aprendendo, dificuldades no processo e aplicações de tais conhecimentos [13].

Apesar dessa definição poder incluir atividades tradicionais como a lição de casa, na prática, aprendizagem ativa se refere a atividades que são introduzidos na sala de aula. O principal elemento da aprendizagem ativa são a atividade dos alunos e seu envolvimento no processo de aprendizagem.

A literatura indica que esse modo de instrução possui efetividade similar à tradicional com relação a retenção de muitas informações factuais a serem lembradas em curto espaço de tempo. Por outro lado, se o objetivo é facilitar a retenção de longo prazo, ajudar os estudantes a desenvolverem habilidades de raciocínio e resolução de problemas ou estimulá-los a aprender um conteúdo em específico a aprendizagem ativa é mais efetiva [14].

A aprendizagem ativa pode ser engrenada de diversas formas, mas a mais tradicional é na forma de prática de laboratório. Segundo [15], a implementação de práticas laboratórias é importante pelos seguintes quesitos:

- Com práticas laboratórias, o estudante aprende a ser um investigador/experimentador;
- Por meio do laboratório, o estudante pode solidificar, ampliar e até aprender conceitos além dos propostos;
- O laboratório ajuda o estudante a ter discernimento da aplicação dos conceitos aprendidos no mundo real.

Os paradigmas mais utilizados na promoção da aprendizagem ativa são os seguintes 2.1.1:

### **1. Aprendizagem Baseada em Problema**

A Aprendizagem Baseada em Problemas é um método de aprendizagem no qual, inicialmente, o professor apresenta aos alunos, sem aula expositiva anterior, um problema o qual é sucedido por uma investigação em um processo de aprendizagem centrada no estudante. Junto com um ou mais professores facilitadores (tutores) para identificar e definir problemas decorrentes, desenvolver hipóteses para explicar o(s) problema(s), e explorar os conhecimentos preexistentes relevantes aos assuntos. Os estudantes estabelecem e exploram o que já conhecem e o que necessitam aprender de forma a progredir no entendimento do(s) problema(s). Os elementos chave do Aprendizagem Baseada em Problema são a formulação de questões que podem ser exploradas e respondidas através da investigação sistemática e auto-dirigida e o teste e a revisão das hipóteses, pela aplicação dos conhecimentos recentemente adquiridos. Essenciais ao processo são a discussão ativas, a análise dos problemas, das hipóteses, dos mecanismos e dos tópicos de aprendizagem, os quais capacitam os estudantes a adquirir e aplicar conhecimentos e a colocar em prática as habilidades de comunicação individual e do grupo, críticas para o ensino/aprendizagem.

### **2. Aprendizagem Baseada em Projetos**

Aprendizagem Baseada em Projeto ou Aprendizagem por Projeto é uma abordagem pedagógica de caráter ativo que enfatiza as atividades de projeto e tem foco no desenvolvimento de competências e habilidades. Assenta-se sobre a aprendizagem colaborativa e a interdisciplinaridade.

Pode-se dizer, pela natureza do que é um projeto, que a Aprendizagem Baseada em Projeto é um conjunto de aprendizagens interligadas realizadas por meio da Aprendizagem Baseada em

Problema, citada anteriormente. Além de todas características da Aprendizagem Baseada em Problemas, a Aprendizagem Baseada em Projetos também oferece ao aluno aprendizagem mais profunda nos seguintes quesitos:

- **Desenvolvimento de habilidades para o século 21;**
- **Desenvolvimento espírito de exploração;**
- **Organizar-se em torno de questões abertas;**
- **Incluir processos de revisão e reflexão;**
- **Apresentar-se para um público.**

Os resultados de se utilizar as metodologias de aprendizagem ativa são excelentes [16]. Entretanto, a aplicação de tais metodologias é complexa. Uns dos principal desafio ao se utilizar o paradigma de aprendizagem ativa é conseguir envolver todos estudantes em atividades produtivas sem sacrificar tempo e recursos importantes no ensino dos conteúdos da disciplina em questão.

## **2.1.2 Aprendizagem por masterização**

### **2.1.2.1 Introdução**

Esta seção trata do paradigma de aprendizagem por masterização e da Taxonomia de definição de objetivos educacionais propostas por Benjamim S. Bloom. Além disso, o é descrito um aplicativo, desenvolvido do ambiente de programação QT, cujo objetivo é auxiliar a definição de objetivos educacionais de forma mais clara e precisa.

*Aprendizagem por Masterização* é uma metodologia de ensino proposta por Benjamim S. Bloom em 1968 juntamente com uma Taxonomia própria. Essa metodologia se baseia nas pesquisa de Bloom relacionadas aos *níveis cognitivos de aprendizagem*.

Como mostrado na seção 2.1.2.2, os níveis cognitivos de aprendizagem são crescente do mais simples ao mais complexo. Isso significa que, para adquirir uma nova habilidade pertencente ao próximo nível, o aluno deve ter dominado e adquirido a habilidade do nível anterior. Ao ensinar um turma sob o paradigma da masterização, a maioria dos alunos ( 80 a 90%) deve ter masterizado a habilidade proposta para que outro conteúdo possa ser tratado.

Só após conhecer um determinado assunto, alguém poderá compreendê-lo e aplicá-lo. Nesse sentido, a taxonomia proposta não é apenas um esquema para classificação, mas uma possibilidade de organização hierárquica dos processos cognitivos de acordo com níveis de complexidade e objetivos do desenvolvimento cognitivo desejado e planejado.

Os processos categorizados pela Taxonomia dos Objetivos Cognitivos de Bloom, além de representarem resultados de aprendizagem esperados, são cumulativos, o que caracteriza uma relação de dependência entre os níveis e são organizados em termos de complexidades dos processos mentais.

Encerrando um modo de utilização bastante prático, uma vez que permite, a partir da utilização de uma tabela Domínio Cognitivo perceber qual o verbo a utilizar / aplicar, em função do

comportamento esperado, organizando os objetivos de aprendizagem em seis níveis, os quais são, por ordem crescente de complexidade os seguin.

A metodologia de aprendizagem sob masterização se baseia sob 2 crenças fundamentais:

1. Virtualmente todos estudantes podem aprender todos conteúdos acadêmicos importantes ao nível de excelência;
2. A função primária das escolas é definir objetivos de aprendizagem e ajudar todos estudante a alcançá-los.

Para que um curso baseado em masterização tenha sucesso, são seguintes características são mandatórias:

- Indicar claramente objetivos e metas instrucionais;
- Garantir clara ligação entre objetivos, ensino e testes;
- Comunicar grandes expectativas de sucesso aos alunos;
- Unidades de ensino pequenas e sequenciadas;
- Manter o ciclo básico de ensino-teste-correção-teste;
- Predefinir os padrões de maestria de conhecimento;
- Manter registros claros e atualizados de progresso dos alunos de forma compreensível para os estudantes para fornecer *feedback* imediato;
- Comprometimento ao desenvolvimento da equipe nas práticas de *masterização*.

A chave para a eficácia do método de aprendizagem por masterização reside na disponibilização sistemática e uso de testes e instruções corretivas.

Instrução corretiva, por sua própria natureza, deve ser direcionada a alunos pessoalmente e, portanto, a problemas e dificuldades pessoais. Este processo será flexível em termos de:

- Tempo: O tempo necessário, nas unidades iniciais, para utilização de instruções corretivas irá reduzir tal necessidade em unidades futuras e mais complexas, permitindo que a turma, possa, em sua maioria, atingir o nível de masterização.
- Estratégias de Grupo:

As estratégias de agrupamento devem incluir a instrução grupo inteiro, aulas em grupo pequeno, tutoria entre pares, e tutoria individual. As decisões serão tomadas à luz dos testes de formação e recursos disponíveis. O agrupamento de alunos para corretivos será depende unicamente no *feedback* de teste de formação com permissão de movimento de maestria para subgrupos de não-mestria, caso seja identificada a necessidade. A associação a grupos não deve permanecer constante.

- Variedade: Os professores e instrutores devem possuir uma gama diversificada de teste e instruções corretivas. Estudantes que masterizaram o conteúdo no primeiro teste formativo, devem ser envolvidos em atividades de ampliação de conhecimento no mesmo conteúdo masterizado, preferencialmente, e em atividades de tutoria em par para estudantes que ainda não masterizarão o conteúdo.

O fluxo da metodologia de masterização é o seguinte:

1. **Definição de objetivos educacionais:** Neste ponto, os objetivos educacionais são determinados pelo professor usando um currículo especificado ou não. Tanto professor quanto alunos devem compreender e focar nos objetivos especificados.
2. **Ensino:** Inicialmente, uma turma seguindo a metodologia de masterização se assemelhará com a metodologia tradicional. O material poderá ser apresentado via aulas, apresentações, demonstrações, discussões ou qualquer forma que o professor considerar mais conveniente. As duas características mais importantes de diferenciação em sala de aula da teoria de masterização em relação a teoria tradicional rapidamente devem ser evidenciados. A primeira característica é a seguinte: **O objetivo de cada aula deve ser claramente exposto.** A segunda característica é a seguinte: O professor deve explicitar aos estudantes que **ele acredita que todos os estudantes podem aprender bem o material e espera que eles assim o façam.**
3. **Primeiro teste formativo:**

Depois que todo material foi passado (o que pode levar um dia ou várias semanas), o professor realiza o teste formativo para verificar a aprendizagem dos alunos. Esse teste não conta como nota para graduação final. Ele serve para que tanto professor quanto alunos saibam onde mais trabalho é necessário. Esse passo é necessário para a existência de *feedback* contínuo e identificação e erros na dinâmica de grupo.
4. **Alternativas de aprendizagem:**

Após o primeiro teste formativo, é fornecido aos estudantes alternativas de aprendizagem. Aqueles que tiveram problemas no teste formativo serão reensinados de formas diferentes para corrigir os erros. Aqueles que já masterizaram o material participarão de atividades de enriquecimento e/ou ajudarão os estudantes que não masterizaram o conteúdo.
5. **Segundo teste formativo (ou reteste):**

Depois que as alternativas de aprendizagem foram concluídas, o professor deve realizar um segundo teste relacionado ao mesmo material. Assumindo que após isso, a maioria dos estudantes pode masterizar o material, então, a turma está pronta para o processo conteúdo e o fluxo volta para o primeiro ponto desta enumeração.
6. **Teste geral:**

Esse teste deve ser realizado após a masterização de um número predeterminado de unidades. Esse teste informará ao professor e estudantes o que foi aprendido até então

### 2.1.2.2 Taxonomia dos objetivos educacionais - Taxonomia de Bloom

A taxonomia dos objetivos educacionais (Taxonomia de Bloom) é um modelo de estruturação de objetivos educacionais segundo níveis hierárquicos e crescentes de níveis de aprendizagem. A taxonomia de Bloom foi planejada para três domínios distintos de aplicação [1]. São eles:

- Domínio cognitivo, cuja abrangência é a aprendizagem intelectual.
- Domínio afetivo, cuja abrangência está ligado aos aspectos de sensibilização, socialização e gradação de valores.
- Domínio psicomotor, cuja abrangência está ligada a habilidades de execução de tarefas que envolvem o aparelho motor.

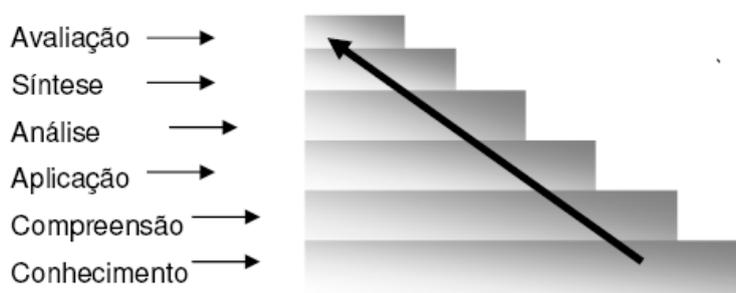


Figura 2.1: Níveis de aprendizagem no domínio cognitivo [1].

Cada um destes domínios tem diversos níveis de profundidade de aprendizado. Por isso a classificação de Bloom é denominada hierarquia: cada nível é mais complexo e mais específico que o anterior como mostrado na Figura 2.1.

Os níveis cognitivos de Bloom são usados para definir *objetivos educacionais* de forma precisa. A Figura 2.2 mostra os verbos a serem usados para se definir objetivos educacionais para cada dos níveis citados de acordo com a Taxonomia de Bloom.

CONHECIMENTO	COMPREENSÃO	APLICAÇÃO	ANÁLISE	SÍNTESE	AVALIAÇÃO
Apontar	Descrever	Aplicar	Analisar	Armar	Ajuizar
Arrolar	Discutir	Demonstrar	Calcular	Articular	Apreciar
Definir	Esclarecer	Dramatizar	Classificar	Compôr	Avaliar
Enunciar	Examinar	Empregar	Comparar	Constituir	Eliminar
Inscrever	Explicar	Ilustrar	Contrastar	Coordenar	Escolher
Marcar	Expressar	Interpretar	Criticar	Criar	Estimar
Recordar	Identificar	Inventariar	Debater	Dirigir	Julgar
Registrar	Localizar	Manipular	Diferenciar	Reunir	Ordenar
Relatar	Narrar	Praticar	Distinguir	Formular	Preferir
Repetir	Reafirmar	Traçar	Examinar	Organizar	Selecionar
Sublinhar	Traduzir	Usar	Provar	Planejar	Taxar
Nomear	Transcrever		Investigar	Prestar	Validar
			Experimental	Propor	Valorizar
				Esquematar	

Figura 2.2: Verbos de descrição de objetivos educacionais a serem usados em cada nível cognitivo.

Um objetivo educacional definido segundo a taxonomia de Bloom segue o seguinte padrão:

"O(s) aluno(s) será(ão) capaz(es) de (**Verbo próprio do nível cognitivo considerado**) (**Conteúdo estudado**).

Um exemplo de aplicação para o caso de primeiro nível cognitivo, no estudado de equações diferenciais:

"O aluno será capaz de **definir** o que são **equações diferenciais**."

### 2.1.2.3 Programa desenvolvido em QT para auxílio na definição de objetivos educacionais

Para este trabalho, foi criado um aplicativo na plataforma de programação QT<sup>®</sup> para facilitar a definição de objetivos educacionais de acordo com a Taxonomia de Bloom.

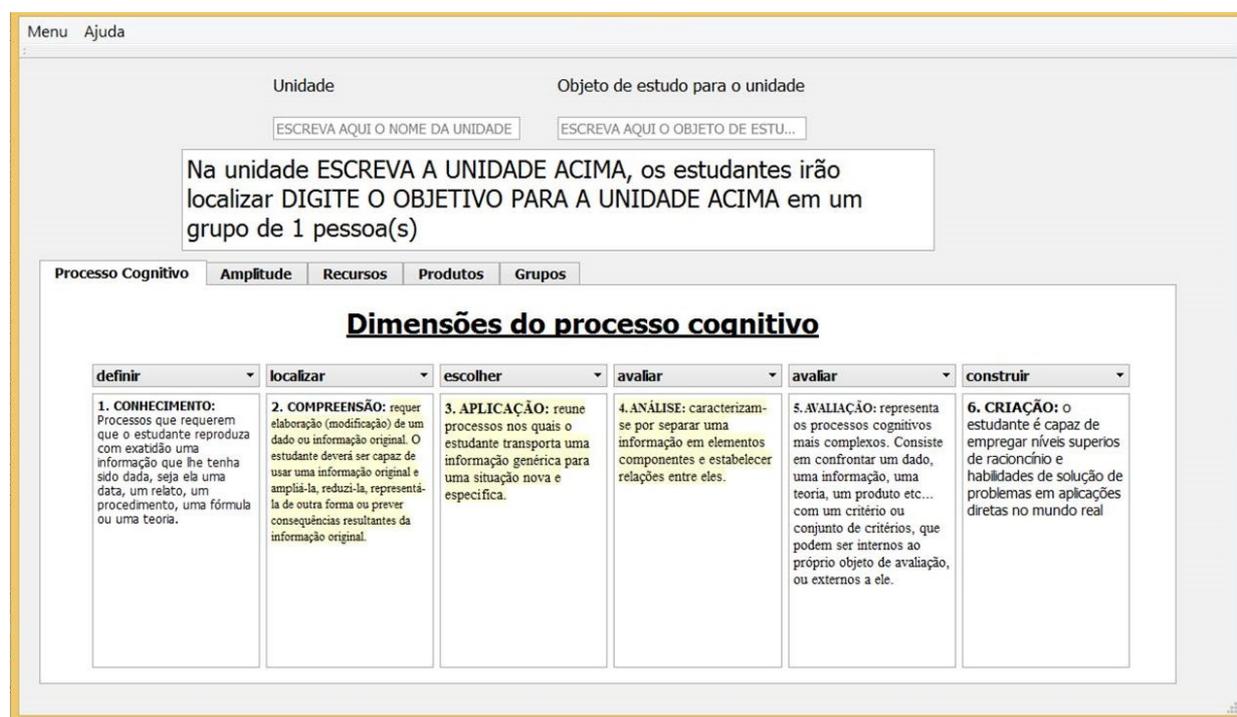


Figura 2.3: Tela inicial do aplicativo desenvolvido em QT para definir os objetivos educacionais.

A Figura 2.3 mostra a tela inicial do aplicativo desenvolvido. Na região de inserção de texto Descrita por **Unidade**, deve-se escrever de qual unidade de ensino se está tratando. Na região textual central, marcada na Figura 2.4, é mostrado resultado do objetivo educacional definido.

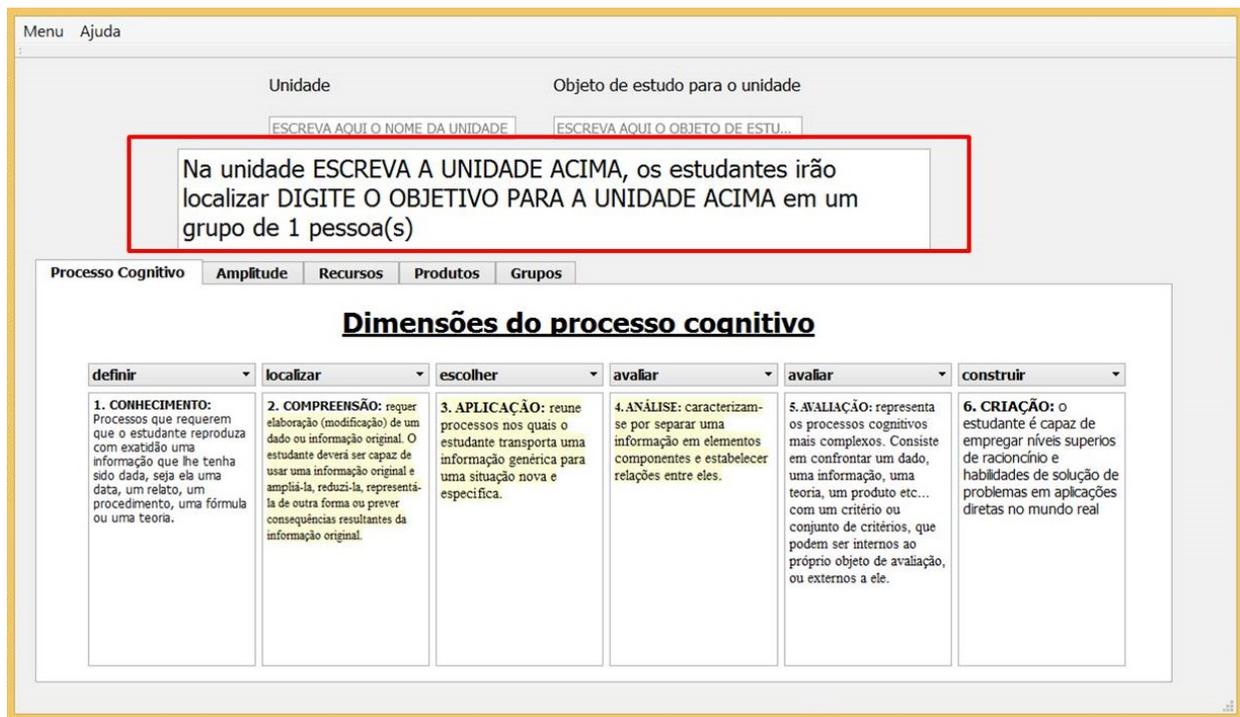


Figura 2.4: Tela inicial do aplicativo desenvolvido em QT para definir os objetivos educacionais, retângulo vermelho marcando a região onde o objetivo educacional definido é mostrado.

A Figura 2.5 mostra o resultado, na região textual central , da definição do **Unidade**.

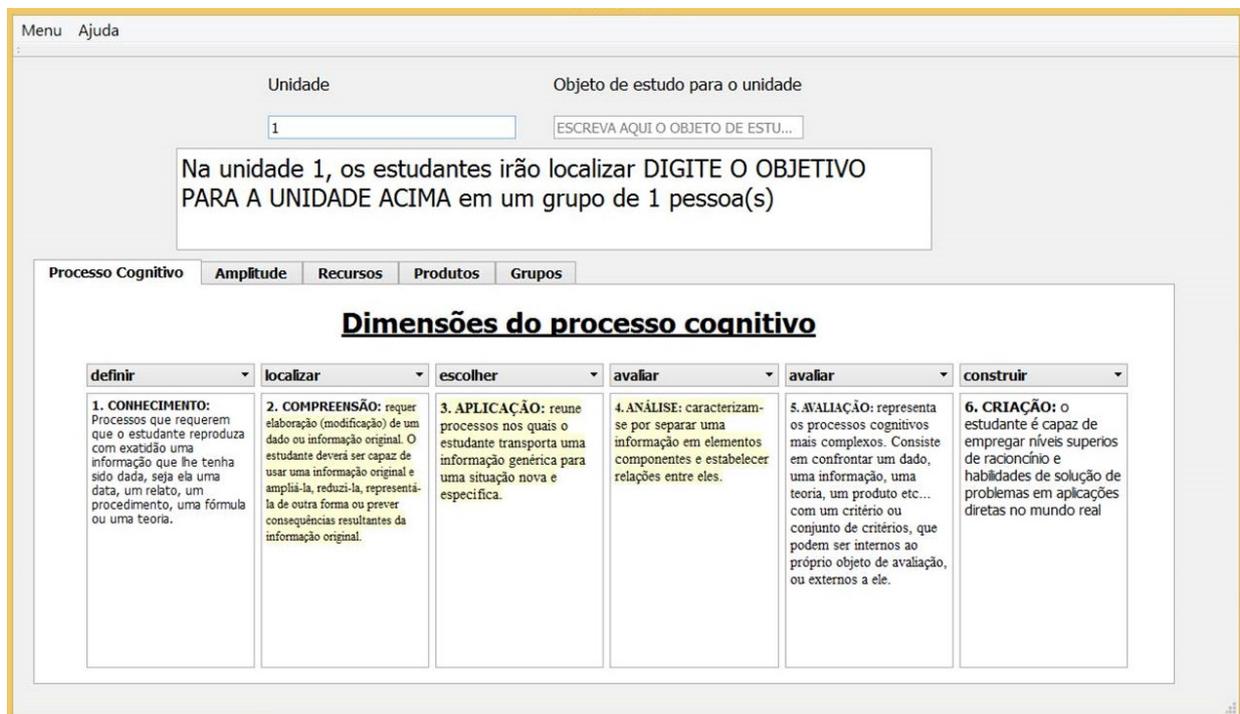


Figura 2.5: Definindo uma unidade de ensino.

A Figura 2.6 mostra o resultado, na região textual central, da definição do **Objeto de estudo** específico para a **Unidade** definida.

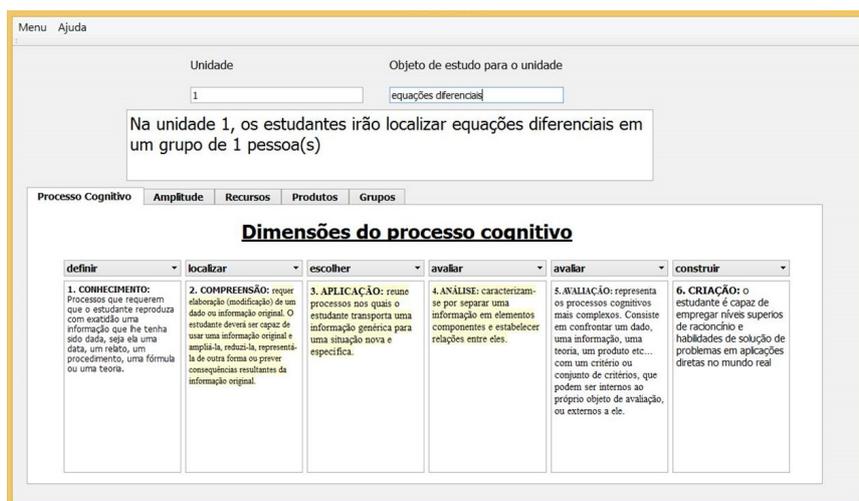


Figura 2.6: Definindo um objetivo de estudo para a unidade de estudo.

A Figura 2.7 mostra os níveis do processos cognitivos da Taxonomia de Bloom.

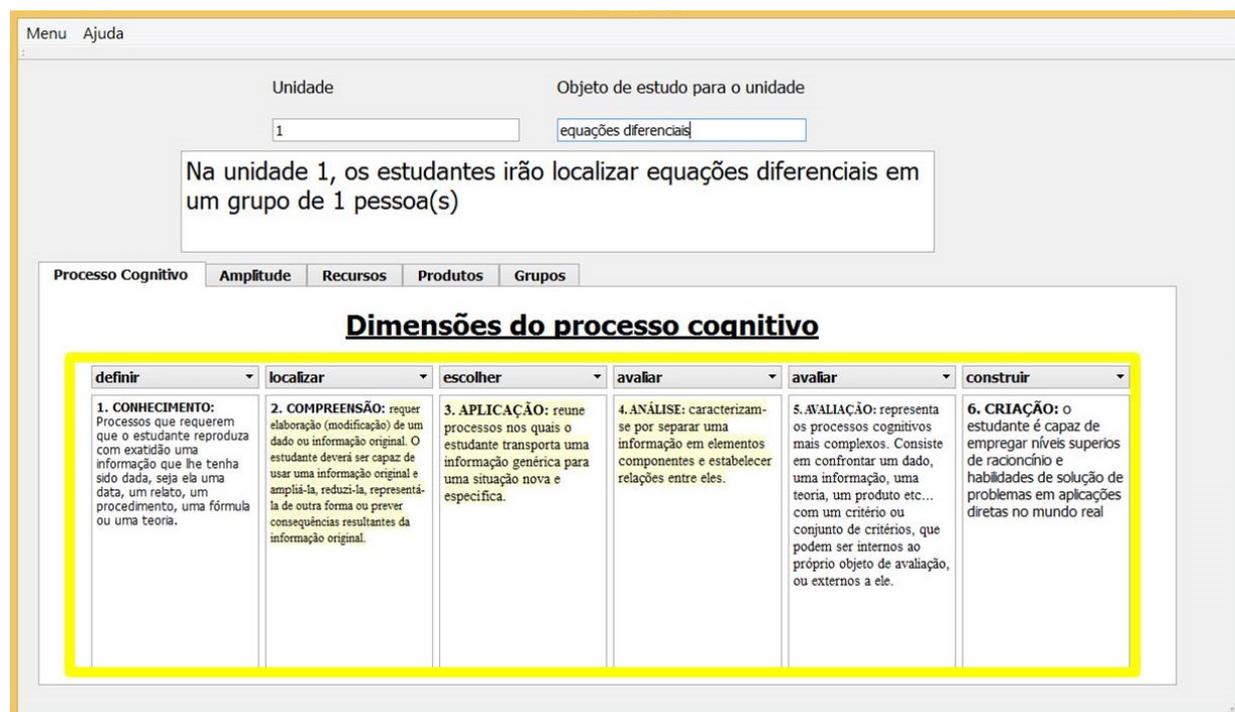


Figura 2.7: Retângulo amarelo destacando os níveis cognitivos.

Cada *Combo-box* mostrado possui os verbos associados a cada nível cognitivo. Quando um verbo é selecionado, o objetivo educacional exposto na região retangular é alterado, como exposto na Figura 2.8.

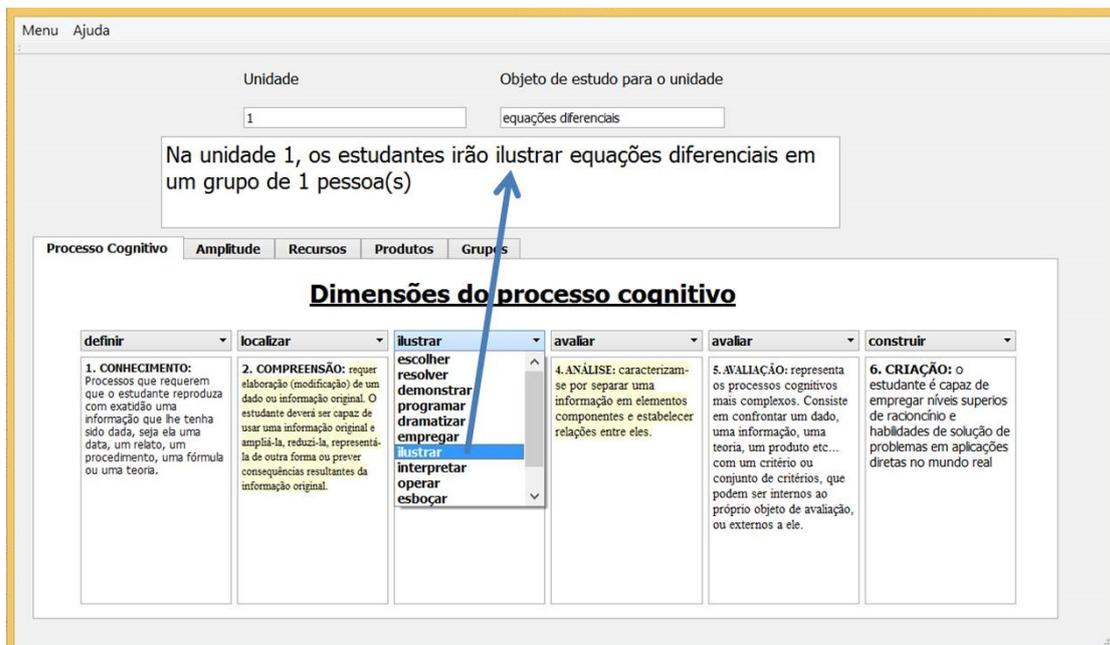


Figura 2.8: Selecionando um verbo de um nível cognitivo.

Na aba *Amplitude*, são selecionados modificadores para especificação do objetivo educacional. A Figura 2.9 mostra o uso do modificador para alterar o objetivo educacional adicionando o modificador "o objetivo geral".

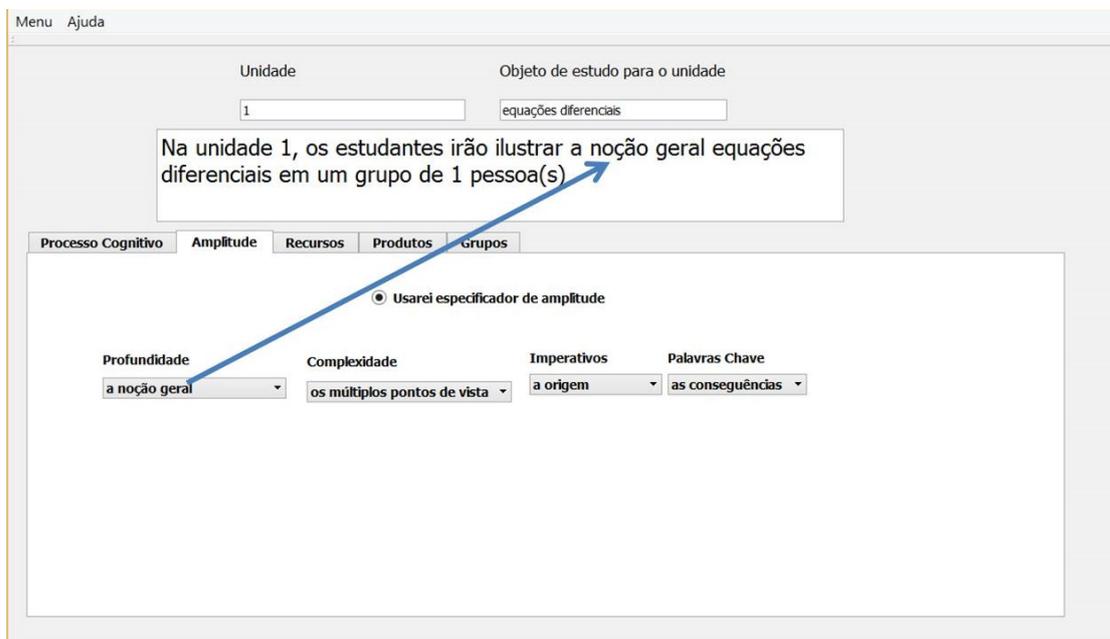


Figura 2.9: Selecionando a amplitude do estudo a ser realizado.

Na aba *Recursos*, são especificados recursos com os quais os estudantes realizarão atividades em direção à masterização do conteúdo. A Figura 2.10 mostra o uso do especificador para alterar o objetivo educacional adicionando o recurso "livro".

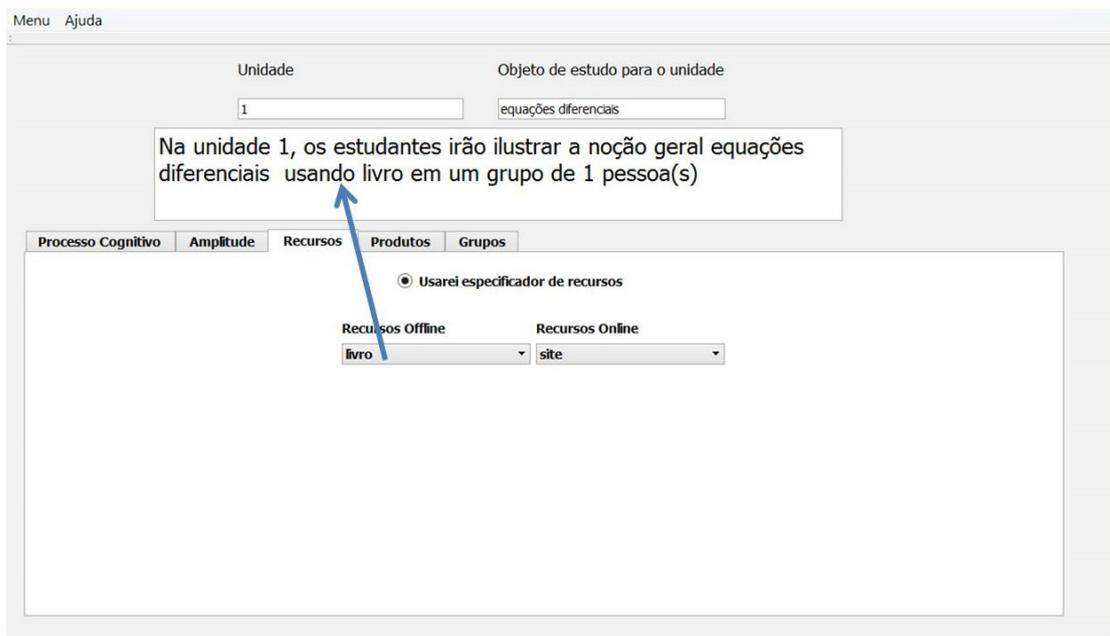


Figura 2.10: Selecionando o recursos a ser utilizado para alcançar o objetivo educacional especificado.

Na aba *Produtos*, é especificado o que os alunos desenvolverão para atingir o objetivo educacional especificado. A Figura 2.11 mostra o uso do especificador de produto a ser desenvolvido para alterar o objetivo educacional adicionando o objetivo de produzir um "artigo".

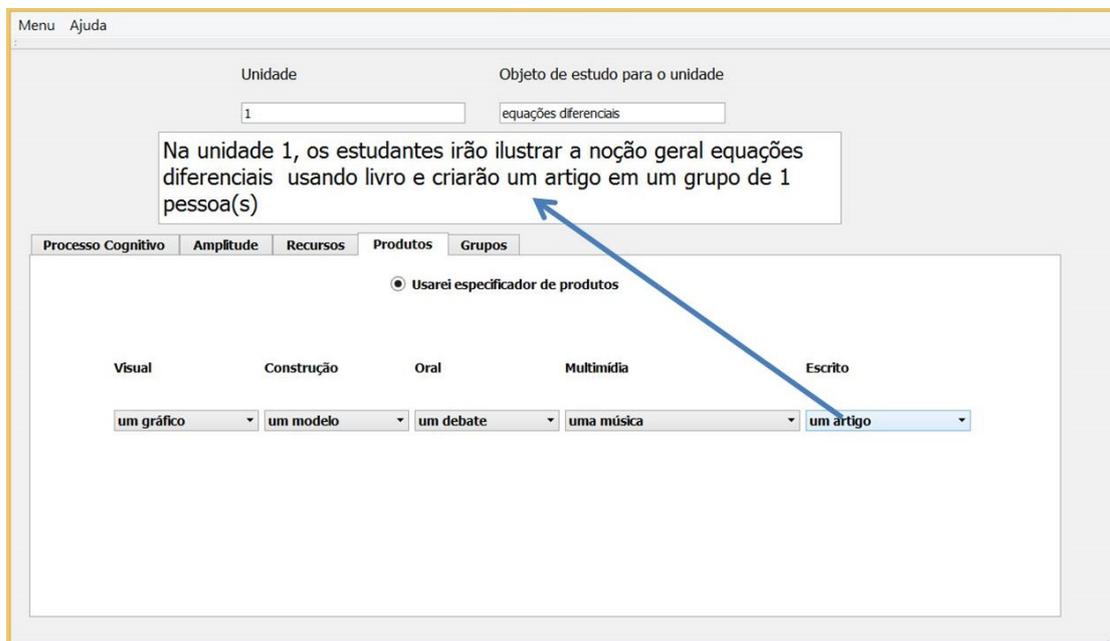


Figura 2.11: Selecionando um produto a ser desenvolvido como atividade de masterização.

Finalmente, na aba *Grupos*, é especificado a quantidade de alunos por grupo que realizarão, conjuntamente as atividades de masterização. A Figura 2.12 mostra a alteração da quantidade de

pessoas por grupo para, por exemplo, 4 pessoas.

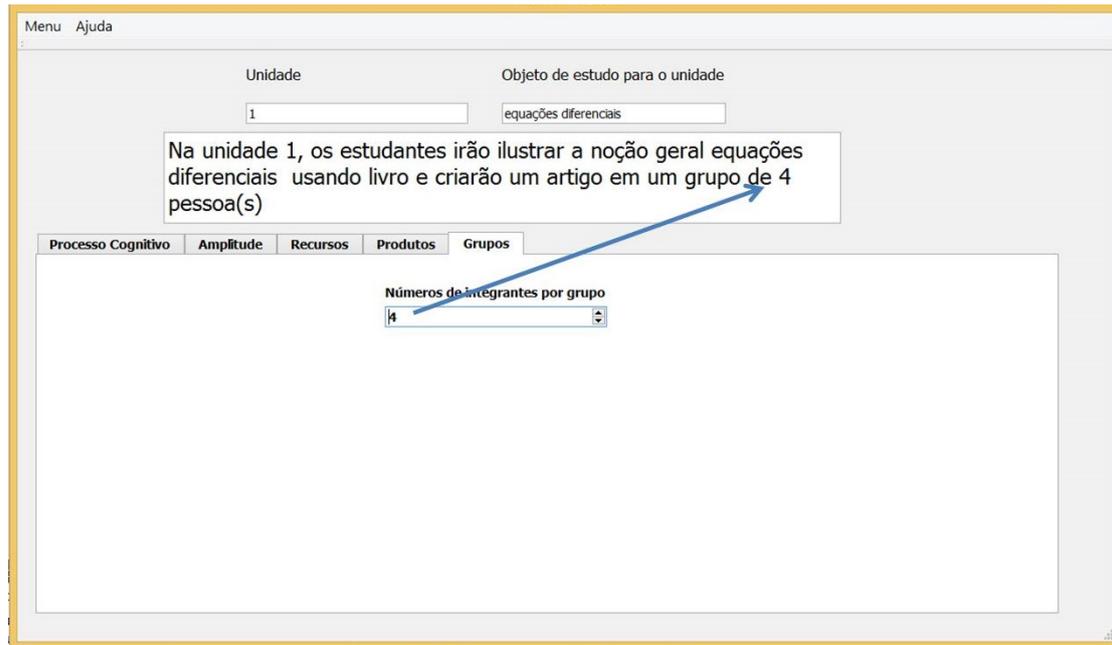


Figura 2.12: Selecionando a quantidade de pessoas por grupo.

## 2.2 Metodologias para desenvolvimento de projetos

Scrum é um metodologia ágil de desenvolvimento de projeto que surgiu no contexto de engenharia de software. O termo ágil se aplica nesse contexto para ser um contraste a metodologia em *Cascata* (metodologia clássica) de desenvolvimento de projetos de software. Enquanto na metodologia *Cascata*, foca-se principalmente em *processos* e na obediência a eles, na metodologia Scrum, foca-se na *melhora contínua* e *entrega constante de valor* [17].

Apesar de suas origens, tanto a metodologia em *Cascata* com o *Scrum* são amplamente utilizados no mais diversos campos.

Nesta seção, descreve-se rapidamente a metodologia de desenvolvimento de projetos *Cascata*, para depois apresentar uma descrição detalhada da metodologia de desenvolvimento de projetos *Scrum*, a qual é a metodologia a ser aplicada no curso *Algoritmos e Programação de Computadores* proposto.

### 2.2.1 Metodologia Cascata

A Figura 2.13 mostra a um breve resumo das fases que devem ser obedecidas na metodologia de desenvolvimento de projetos *Cascata*. São elas:



Figura 2.13: Fluxo da metodologia de desenvolvimento de projeto *Cascata*

Fonte: <https://brainstormdeti.wordpress.com/2010/05/25/uma-comparacao-entremodelo-agil-e-cascata/>

1. Análise e descoberta de requisitos de negócio;
2. Planejamento de produto (Design);
3. Implementação (codificação);
4. Testes;
5. Entrega do produto ao cliente (instalação);
6. Manutenção.

Todas etapas mostrada nessa figura são sequenciais planejadas extensivamente pela equipe de projeto, produzindo, nesse intervalo entre etapas, um conjunto amplo de documentos.

Muitos argumentam que a metodologia em *Cascata* torna o desenvolvimento do projeto muito engessado [18]. Os pontos positivos e negativos da metodologia de desenvolvimento de projeto *Cascata* são mostrados na Tabela 2.1.

O principal problema, pontuado por pesquisadores [4], quanto à metodologia *Cascata*, é, como mostrado na Tabela 2.1 a falta de dinamicidade, especialmente quanto a definição de pre-requisitos de projeto. Uma vez que a primeira fase do projeto, colhimento de requisitos, é concluída, não mais os requisitos são revistos. Isso causa, muitos problemas de retrabalho, em especial, nas fases finais e atrasos grandes.

### 2.2.2 Metodologia Scrum

A metodologia *Scrum* de gerenciamento de projeto surgiu como contraste à metodologia *Cascata* tratada na Seção 2.2.1 sendo uma metodologia ágil de desenvolvimento de projetos.

A metodologia Scrum é uma metodologia a ser desenvolvida em pequenas equipes, chamadas *Time ou Equipe Scrum* e umas de suas principais características são a revisão frequente dos **requisitos de projeto**, a melhora contínua e o **acréscimo de valor direto** acima da obediência a cronogramas e processos (Próprios de metodologias similares a metodologia *Cascata*).

Tabela 2.1: Pontos positivos e negativos da metodologia de desenvolvimento de projetos *Cascata* [4].

Pontos positivos	Pontos negativos
Documentação detalhada	Começo de projeto lento
Requerimentos de projeto totalmente colhidos e acordados no início do projeto	Requerimentos fixos e dificilmente modificáveis
O projeto pode ser desenvolvido por pessoas ainda inexperientes.	Sem acompanhamento do projeto pelo cliente e outros <i>stakeholders</i> até o projeto estar concluído
Número de defeitos reduzido por meio de planejamento e desenvolvimento mais rigoroso	Pouca flexibilidade para mudar a direção e modus-operandi do projeto.
Início e fim de cada estágio plenamente definidos, permitindo acompanhamento mais eficiente do progresso.	Clientes e <i>Stakeholders</i> possivelmente podem alterar os requisitos de projeto após a fase de início de projeto

Os papéis, na metodologia Scrum, são os seguintes:

- **Scrum Master:** O *Scrum Master* é o responsável por manter a *Equipe Scrum* nas diretrizes do processo Scrum. Ele deve treinar e orientar os demais participantes. Além disso, ele é responsável por proteger a equipe Scrum de perturbações externas alheias ao projeto incentivando tal equipe na tomada de decisões para torna-lá progressivamente mais autogerenciável. Com relação a todos envolvidos no projeto, além da equipe Scrum, o *Scrum Master* deve zelar pela visibilidade do progresso do projeto para que a verificação e gerência sejam processos compartilhados.
- **Product Owner:** O *Product Owner* é representa a "voz do cliente". O *Product Owner* atua como "ponte" entre a equipe desenvolvimento e os *Stakeholders* (*Pessoa com interesse direto no produto*). Ele deve entender as necessidades e prioridades dos *Stakeholders* e passar claramente os **critérios de aceitação** para a equipe de desenvolvimento
- **Equipe Scrum:** As equipes Scrum são auto-organizadas e multidisciplinares. Como tal, ela deve escolher a melhor forma de desenvolver seus trabalhos ao invés de serem comandadas por outros de fora da equipe. Equipes multidisciplinares possuem todas as competências necessárias para desenvolverem seus trabalhos sem dependerem de outros que não fazem parte da Equipe Scrum. O modelo da Equipe Scrum é desenvolvido para otimizar a flexibilidade, criatividade e produtividade.

Equipes Scrum entregam produtos iterativamente e incrementalmente maximizando a oportunidade de *feedback*. Entregas incrementais de produto "pronto" garantem que uma versão

do produto potencialmente utilizável está sempre disponível para uso.

O fluxo de um projeto seguindo a metodologia Scrum é o seguinte:

- **Fase 1 - Início:** Para a fase inicial de um projeto Scrum, devem se desenvolver as seguintes etapas:

**1.1) Criar Visão de Projeto:** Descrição da necessidade ou desejo dos clientes e as características do produto que resolvem as necessidades. Sendo elas necessidades relacionadas a:

**Características funcionais:** Características que descrevem uma funcionalidade direta do produto.

**Características não funcionais** Características ligadas a eficiência e eficácia do produto ao realizar uma de suas características funcionais.

1.2) Para descrever as necessidades ou desejo dos clientes, devem ser criadas *User Stories* no formato: "Como um (**Tipo de Usuário**), eu quero, poder fazer (**Característica Funcional**)".

- **Fase 2 - Plano de Projeto e Estimações:**

2.1) Critérios de aceitação são criados (a ser feito pelo Product Owner juntamente com o Scrum Team).

2.2) Scrum Master e time estimam o esforço necessário para desenvolver as User Stories.

2.3) Scrum master e time se comprometem a desenvolver o produto de acordo com as Epics (Conjunto de User Stories) e critérios de aceitação.

2.4) User Stories são divididas em subtarefas para criar uma Task List.

2.5) Time Scrum se reúne para decidir quais tarefas serão realizadas na Sprint (Sprint Backlog).

- **Fase 3 - Implementação:**

Na fase de implementação, o trabalho é dividido em ciclos curtos de trabalho, de 1 a 4 semanas de duração, como mostrado na Figura ???. Esses ciclos são chamados de *Sprints*. Cada Sprint tem sua lista de tarefas (*Task List*) própria.

As responsabilidades da Equipe Scrum na Fase 3, são as seguintes:

3.1) Criar os "entregáveis" do SPRINT.

3.2) Realizar reuniões diárias rápidas de no máximo 15 minutos para discutir problemas e progresso.

3.3) Atualizar o quadro Scrum de ( A FAZER , FAZENDO, FEITO) a cada dia de trabalho.

- **Fase 4 - Revisão e Retrospecto :**

Tabela 2.2: Pontos positivos e negativos da metodologia de desenvolvimento de projetos *Scrum*

Pontos positivos	Pontos negativos
Início rápido, entrega de produto realizada incrementalmente com revisão de clientes e <i>feedback</i> frequentes	Muitas vezes, o planejamento Scrum pode ser interpretado como mau-planejado e indisciplinado
Verificação frequente da evolução dos requerimentos do cliente	Necessita de um time altamente qualificado e pronto a interagir com clientes diretamente
Resposta rápida a mudanças	É necessário um alto nível de envolvimento dos clientes no projeto.
Menos retrabalho a ser feito, por causa do envolvimento do cliente, testes contínuos e <i>feedback</i> frequente.	Falta de planejamento de longo prazo detalhado
Comunicação de tempo real entre time de desenvolvimento e clientes	Pouca documentação produzida

A fase de *Revisão e Retrospecto* é realizada ao final de cada Sprint. As responsabilidades da equipe Scrum na Fase 4 são"

- 4.1) Os entregáveis do Sprint são mostrados para os Stakeholders.
- 4.2) Essas reuniões são feitas para garantir a aprovação do produto desenvolvido e fazer os ajustes necessários o mais rápido possível.
- 4.3) O Scrum Master e o time Scrum se reúnem para discutir o que foi aprendido no Sprint.
- 4.4) Informação é documentada para futuros Sprints.

Após a conclusão dessas 4 fases, o projeto Scrum chega a sua finalização. As informações colhidas a cada Sprint são guardadas para projetos Scrum futuros.

A Tabela 2.2 faz um resumo dos pontos positivos e negativos próprios de metodologias ágeis de desenvolvimento de projeto, como a metodologia Scrum.

## 2.3 Algoritmos e Programação de Computadores

### 2.3.1 Introdução

Este capítulo é destinado ao estudo da importância da computação no contexto do ensino e aprendizagem de ciências exatas. É apresentado neste capítulo um breve estudo de currículos de disciplinas similares a *Algoritmos e Programação de Computadores* das melhores universidades mundiais e as melhores universidades brasileiras.

Tomando o que foi apresentado nas seções anteriores a este capítulo sobre os efeitos positivos da aprendizagem ativa e suas técnicas de implementação é urgente uma revisão das disciplinas, currículos, métodos de avaliação de forma a incluir formas mais dinâmicas de ensino e aprendizagem.

### 2.3.2 Currículos

Fez-se, para os estudos de currículo, uma breve pesquisa das 5 melhores universidade mundiais e as cinco melhores universidades no Brasil. Os resultados de tal estudo são mostrados na Tabela 2.3. A universidade de Brasília (UnB) está, atualmente, entre as 800 melhores universidades do mundo e é a quinta melhor universidade brasileira, segundo a instituição *Times High Education* [19].

Os índices utilizados pela instituição de pesquisa *Times High Education* [19] são os seguintes:

1. Score Ensino;
2. Score Panorama Internacional;
3. Score Impacto na Indústria;
4. Score Pesquisa;
5. Score Citação.

Como dito em [20], pode-se concluir, sem perda de exatidão, que tais índices são totalmente dependentes e proporcionais à qualidade de ensino nas universidade em questão.

UNIVERSIDADE	PAÍS	POSIÇÃO	SCORE MÉDIO	SCORE ENSINO	SCORE PANORAMA INTERNACIONAL	SCORE IMPACTO NA INDÚSTRIA	SCORE PESQUISA	SCORE CITAÇÃO
Universidade de Brasília	Brasil	800	20	21.9	24.1	20	7.4	10.4
Universidade Pontifícia Católica do Rio de Janeiro	Brasil	600	20	24.5	31.3	100	24.1	23.1
Universidade Federal do Rio de Janeiro	Brasil	600	20	32.4	25.1	42.9	19.7	18.3
Universidade de Campinas	Brasil	400	20	44.6	21.1	49.4	42.3	22.6
Universidade de São Paulo	Brasil	300	20	59.1	25.3	30.5	57.1	20.4
University of Chicago	EUA	10	87.9	85.7	65	36.6	88.9	99.2
ETH Zurich – Swiss Federal Institute of Technology Zurich	Suíça	9	88.3	77	97.9	80	95	91.1
Imperial College London	Reino Unido	8	89.1	83.3	96	53.7	88.5	96.7
Princeton University	EUA	7	90.1	85.1	78.5	52.1	91.9	99.3
Harvard University	EUA	6	91.6	83.6	77.2	45.2	99	99.8
Massachusetts Institute of Technology	EUA	5	92	89.4	84	95.4	88.6	99.7
University of Cambridge	Reino Unido	4	92.8	88.2	91.5	55	96.7	97
Stanford University	EUA	3	93.9	92.5	76.3	63.3	96.2	99.9
University of Oxford	Reino Unido	2	94.2	86.5	94.4	73.1	98.9	98.8
California Institute of Technology	EUA	1	95.2	95.6	64	97.8	97.6	99.8

Tabela 2.3: Tabela com scores de avaliação das 5 melhores universidades do mundo e das 5 melhores universidades brasileiras.

Pesquisou-se o currículo da UnB e o currículo da universidade *California Institute of Technology* para disciplinas com ementa similar ao de *Algoritmos e Programação de Computadores*. O resultado de tal pesquisa é mostrado na Tabela 2.4.

Fez-se, também, uma pesquisa mais aprofundada do conteúdo programático de tais disciplinas verificando a possibilidade de uso da placa Galileo e similares numa possível dinâmica de aprendizagem ativa. O resultado de tal pesquisa é mostrado na Tabela 2.5.

Universidade - Curso	Disciplina
CalTech - Applied + Computacional Mathematics	Introduction to Matlab and Mathematica
CalTech - Computer Science	Introduction to Computer Programming
UnB - Engenharia Mecatrônica	Algoritmos e Programação de Computadores
UnB - Engenharia de Computação	Algoritmos e Programação de Computadores
UnB - Engenharia de Produção	Algoritmos e Programação de Computadores
UnB - Ciência Da Computação	Algoritmos e Programação de Computadores
UnB - Engenharia Eletrônica	Algoritmos e Programação de Computadores
UnB - Engenharia de Software	Algoritmos e Programação de Computadores
UnB - Engenharia AeroEspacial	Algoritmos e Programação de Computadores
UnB - Engenharia Automotiva	Algoritmos e Programação de Computadores
UnB - Engenharia de Energia	Algoritmos e Programação de Computadores
UnB - Engenharia Engenharia de Redes	Computação para Engenharia

Tabela 2.4: Tabela listando as disciplinas na UnB e na *California Insitute of Tecnology* que possuem currículo similar a disciplina *Algoritmos e Programação de Computadores*.

CONTEÚDOS CALTECH	JÁ ENSINADO COM APRENDIZAGEM ATIVA	PODE SER ENSINADO COM GALILEU?	CONTEÚDOS UNB	JÁ ENSINADO COM APRENDIZAGEM ATIVA	PODE SER ENSINADO COM GALILEU?
Desenvolvimento de programas em Python	No	Yes	Linguagem C	No	Yes
Resolução de equações diferenciais não-lineares, Transformada rápida de Fourier, EDO	No	No	Introdução Hardware. Software.	No	Yes
vetorização, scripts e funções, arquivos de entrada e saída, arrays, estruturas, strings	No	Yes	Iteração e recursão	No	Yes
Desenvolvimento de programas em Matlab	No	Yes	Entrada e saída de dados	No	Yes
Estruturas de controle: condicional e repetição	No	Yes	Estruturas de controle: condicional e repetição	No	Yes
Noções de programação estruturada e orientada a objeto	No	Yes	Noções de programação estruturada	No	Yes
Testes e depuração	No	Yes	Testes e depuração	No	Yes

Tabela 2.5: Tabela listando parte dos conteúdos tratados pelas disciplinas na *California Insitute of Tecnology* e na UnB e levantando a questão da metodologia de ensino ser uma *metodologia ativa* e se tal conteúdo pode ser ensinado usando a placa Galileo.

Para cada conteúdo identificado, procurou-se saber também se ele era tratado de seguindo algum paradigma moderno de *Educação Ativa* e, se pela natureza do conteúdo, seria possível utilizar a placa Galileo no processo de *ensino-aprendizagem*.

A proporção de conteúdos que poderiam ser ensinados com a placa Galileo é mostrada na Figura 2.14.

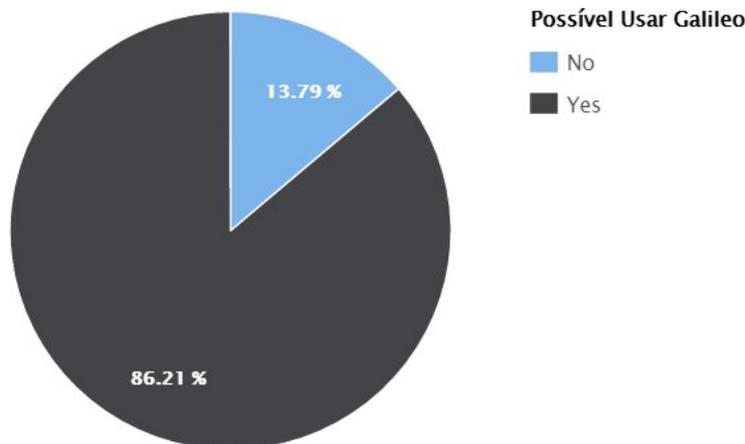


Figura 2.14: Percentagem dos conteúdos tratados na UnB e *California Institute of Technology* que podem ser ensinados utilizando a placa Galileo.

Verificando os conteúdos programáticos, pode-se observar que a inclusão de práticas com a placa Galileo e similares seria possível. Tal mudança curricular em direção à aprendizagem ativa, além de trazer grandes benefícios à qualidade do ensino [21][22], também pode ajudar a formação de novos profissionais alinhados com as competências mais exigidas no século XXI, como proatividade e independência [23].

## 2.4 Placa Intel<sup>®</sup> Galileo

A placa Intel<sup>®</sup> Galileo é uma *placa de desenvolvimento* com microcontrolador baseado processador Intel<sup>®</sup> Quark SoC X1000[24]. A placa Galileo possui software e hardware compatível com a placa *Arduino* com relação aos pinos digitais e analógicos. Um programa escrito para Arduino pode ser usado no Galileu por causa dessa compatibilidade. As Figura 2.15 e 2.16 mostram a placa Intel<sup>®</sup> em suas visão frontal e traseira.

Nesta seção são apresentadas, enumeradas e explicadas todas características da placa Intel<sup>®</sup> Galileo.

Primeiramente são apresentados os pinos da placa Galileo juntamente com uma breve descrição de seu uso. Após isso são descritas as enumeradas e explicadas todas características eletroeletrônicas da placa. Para cada tecnologia na placa é reservada uma pequena sub-seção neste capítulo para sua devida elucidação.

### 2.4.1 Pinagem da placa Intel<sup>®</sup> Galileo

Os pinos da placa Galileo nas partes frontal e traseira são mostrados nas Figuras 2.15 e 2.16



Figura 2.16: Descrição dos pinos da placa Galileo - parte traseira[2]

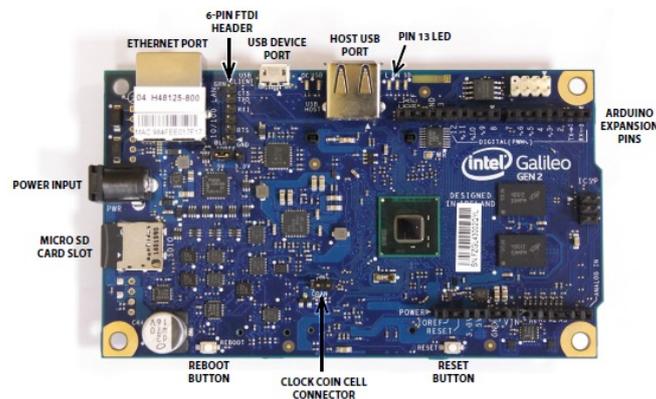


Figura 2.15: Descrição dos pinos da placa Galileo - parte frontal[2]

A descrição de cada um desses pinos é a descrita na tabela 2.6:

#### 2.4.2 Características Elétricas e Eletrônicas da placa Intel<sup>®</sup> Galileo

As características elétricas e eletrônicas da placa são enumeradas a seguir. As características que têm uma sub-seção para explicação mais aprofundada estão marcadas com *itálico* e **negrito**:

- Clock de 400 MHz;
- Arquitetura Intel<sup>®</sup> 32 bits;
- 14 pinos digitais para entrada e saída, 6 das quais podem ser usadas para saída *PWM*;
- 6 pinos para entrada analógica utilizando o *conversor analógico-digital AD7298*;
- Barramento Serial *I2C*;
- Comunicação serial com periféricos *SPI*;
- Porta Serial *UART*;

Tabela 2.6: Pinos Galileo[2]

<b>Pino</b>	<b>Descrição</b>
Micro SD Card Slot	Pino no qual se pode um SD Card para permitir ao Galileo a execução de uma versão de Linux com mais recursos.
Arduino Expansions Pins	Pinos de entrada e saída da placa Galileo. Esses pinos são compatíveis com os pinos do Arduino e Shields relacionadas.
USB Device Port	Pino para conectar um cabo USB do Galileo ao computador para carregar o Galileo com um programa Arduino.
Host USB Port	Pino para conectar um dispositivo periférico
( como webcam, caixa de som, etc).	
6-Pin FTDI Header - Linux instalado no Galileo.	Adaptador para comunicação serial computador
Power Input	Conexão para bateria de 12V. ATENÇÃO, a bateria sempre deve ser conectada ao Galileo antes de conectar um cabo USB do Galileo ao computador para evitar danos a placa.
Ethernet Port	Pino para conectar o Galileo à Internet pelo cabo Ethernet
Mini PCI Express Slot	Pino para conectar um cartão WiFi
Clock Batery Power	Conexão para uma bateria de relógio de 3V de forma a fazer com Galileo guarde informações de data e hora.
Reboot Button	Botão para realizar a placa, inclusive o sistema operacional.
Reset Button	Botão para resetar o código que foi carregado no Galileo.

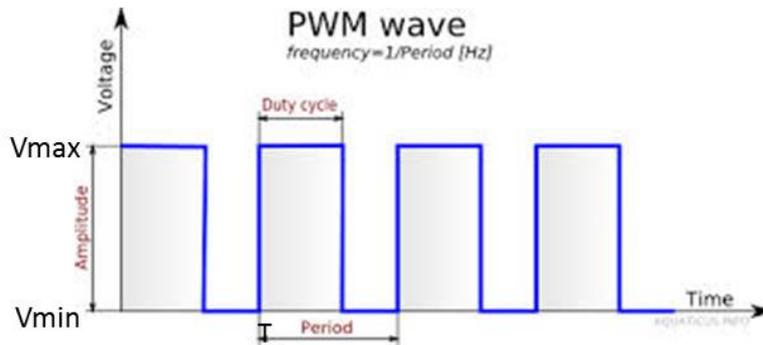


Figura 2.17: Sinal PWM

Fonte: Figura adaptada de <http://www.zembedded.com/avr-introduction-to-pwm-part-i/>

- 16KBytes de memória **L1 Cache**;
- 512KBytes de memória **SRAM**;
- Clock de tempo real integrado (**RTC**);
- Barramento **PCI Express**;
- Conexão para **USB Host e USB Client**;
- 10 pinos padrões **JTAG** para debug;
- 256 MBytes de memória **DRAM**;
- 11 KBytes de memória **EEPROM**.

#### 2.4.2.1 Sinal PWM

*Pulse Width Modulation* (PWM) ou Modulação por Largura de Pulso é uma técnica que modulação de impulso utilizada principalmente para codificar uma mensagem num sinal pulsante [25]

Para o caso do Intel Galileo, as aplicações do sinal PWM são principalmente relacionadas ao controle da tensão DC fornecida a um circuito.

O sinal PWM é gerado com ondas quadradas, de período  $T$  de ciclo. Durante parte do período, o sinal terá amplitude  $V_{max}$ . O intervalo de tempo no qual no sinal tem amplitude  $V_{max}$  é chamado *Duty-Cycle* como mostra a Figura 2.17. O valor DC de um sinal periódico é calculado como a média aritmética da amplitude do sinal no período. O valor da tensão DC fornecida ao circuito pelo Sinal PWM é calculado com a equação 2.1:

$$V_{dc} = \frac{1}{T} \left( \int_0^{DutyCycle} V_{max} dt + \int_{DutyCycle}^T V_{min} dt \right) \quad (2.1)$$

$$V_{dc} = \frac{1}{T}(DutyCicle * V_{max} + T * V_{min} - DutyCicle * V_{min}) \quad (2.2)$$

Se a tensão mínima ( $V_{min}$ ) for igual a zero, o valor DC do sinal PWM é dado por:

$$V_{dc} = \frac{DutyCicle * V_{max}}{T} \quad (2.3)$$

A equação 2.3 mostra que quanto maior for o tempo que o sinal permanecer no seu valor máximo ( $V_{max}$ ), mais próximo de  $V_{max}$  será o valor DC fornecido ao circuito.

O sinal PWM é gerado na placa Galileo utilizando o clock interno máximo de 400 MHz e registradores de Timer específicos para contagem de pulsos do clock.

Como exemplo para a geração do sinal PWM, digamos que o clock da placa foi setado para a frequência 1kHz. Isso significa que a cada 1ms, o clock gerará um pulso, como indicado na equação 2.4.

$$f = 1Khz \rightarrow T = 1ms \rightarrow 1000 \text{ pulsos de clock por segundo} \quad (2.4)$$

Como a tensão de operação da placa Galileo é 5 V, então:

$$V_{max} = 5V \quad (2.5)$$

Caso se queria gerar um sinal PWM cujo componente DC seja 2.5, é necessário então que durante metade do ciclo do sinal PWM, a amplitude do sinal seja 5 V e durante a outra metade do ciclo, a amplitude seja 0V. Para criar tal sinal, o microcontrolador realiza contagem de pulsos de clock.

Para gerar 2.5 V, o microcontrolador( para a frequência exemplo de 1kHz) realiza a contagem de 500 pulsos de clock no intervalo de *DutyCicle* e realiza, após isso, a contagem de 500 pulsos no período no qual a amplitude será de 0 V. Dessa forma, é gerado digitalmente o sinal PWM na placa Galileo.

Como dito no início desta seção, placa Galileo é compatível com a placa Arduino, tanto a nível de hardware quanto a nível de software. Daí, para executar a criação de um sinal PWM na placa galileo deve-se chamar a função *analogWrite(int porta, int valor)*.

A função *analogWrite(int porta, int valor)* recebe como parâmetros dois inteiros. O inteiro *porta* indica quais dos pinos digitais, habilitados para saída PWM, foi selecionado. O inteiro *valor* deve ser um inteiro entre 0 e 255.

```
1 //Comando para setar na porta digital 5 o valor 5*(127/255) = 2.5 Volts
2 analogWrite(5, 127);
```

A tensão DC que estará presente no pino digital segue a formula: 2.6

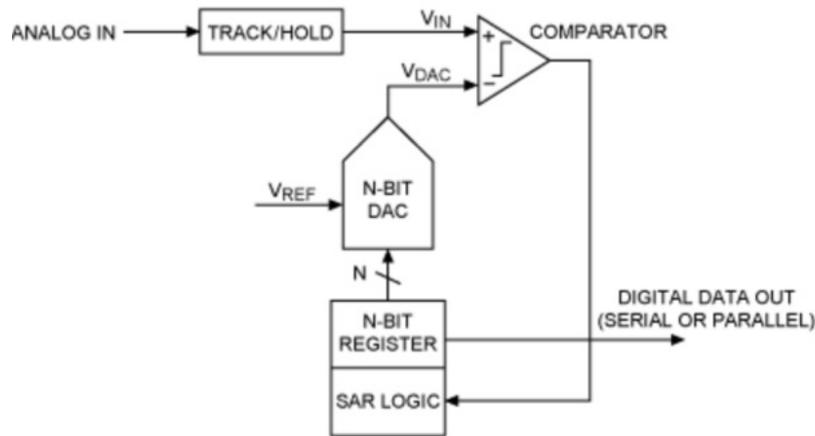


Figura 2.18: Figura esquemática do processo de conversão analógico para digital

Fonte: <https://www.maximintegrated.com/en/app-notes/index.mvp/id/1080>

$$V_{dc} = \frac{5 * valor}{255} \quad (2.6)$$

#### 2.4.2.2 Conversão analógico-digital

A placa Galileo utiliza para a conversão analógico-digital o circuito integrado *AD7298* [26]. O conversor analógico-digital *AD7298* é um conversor de 12 bits e usa para a conversão a técnica de *aproximações sucessivas*.

A Figura 2.18 mostra uma imagem esquemática para o processo de conversão analógico-digital utilizando a técnica de *aproximações sucessivas* e os termos chave para essa técnica são os seguintes:

- Registrador de aproximação sucessiva (SAR)
- Circuito de amostragem e retenção ( Track and Hold)
- Tensão de entrada  $V_{IN}$
- Tensão de referência  $V_{REF}$
- Registrador de N bits (N-BIT REGISTER)
- Conversor digital para analógico de N bits(N-BIT DAC)
- Circuito Comparador

Num primeiro instante, o bit mais significativo do conversor D/A é setado para 1, enquanto os outros N-1 bits são setados para 0. Essa configuração inicial dos N bits do conversor D/A força com que na saída exista 1/2 da tensão de referência  $V_{REF}$ , ou seja  $V_{DAC} = 1/2 V_{REF}$ .

Caso a tensão  $V_{DAC}$  seja maior que a tensão de entrada  $V_{IN}$ , o bit mais significativo será mantido, caso o contrário, esse bit será setado no valor 0. Depois disso, registrador SAR grava o resultado obtido no comparador no bit avaliado. O processo se repete para os N bits, sempre comparando a tensão de entrada  $V_{IN}$  com a tensão de conversão  $V_{DAC}$ , fazendo com que a saída da conversão se aproxime progressivamente a cada iteração [27].

A eficiência do processo de conversão analógico-digital está intimamente ligada ao processo interno de conversão digital-analógico. Há diversos processos de conversão digital-analógico, entretanto, para todos, a quantidade de bits a serem convertidos influencia diretamente na linearidade do processo. Por isso se escolhe, em geral, um conversor digital-analógico de 12 bits.

O conversor A/D utilizado na placa Galileo, segundo seu respectivo datasheet [26], possui características implementadas que, entre outras incluem:

- Sensor de temperatura integrado para devidos ajustes às variações de parâmetros causados pela variação de temperatura;
- Taxa de saída de conversões completadas superior a 1 MSPS ( Million Samples Per Second).

### 2.4.2.3 Barramento Serial I2C

I2C(Inter-Integrated-Circuit) é um protocolo de comunicação serial desenvolvida originalmente pela *NXP Semiconductor*. Ela permite a comunicação direta entre diversos componentes utilizando apenas três barramentos: um barramento para transmissão de bits dados - *Serial Data Line(SDA)* - um barramento para o sinal de clock - *Serial Clock Line(SCL)* e um barramento para o uso de um resistor de *pull-up* ligado diretamente uma tensão  $V_{dd}$  de 5V ou 3.3V. O endereçamento no protocolo I2C pode ser de 7 ou 10 bits. A velocidade de transmissão de dados variam de 10kbits/s - para o modo *low speed*- 400 kbits/s - para o modo *Fast mode* - e 3.4Mbit/s para o *modo Fast mode plus* [28].

O resistor *pull-up* serve para ter como valor alto de tensão( lógico 1) tanto o barramento de clock como o barramento de dados, Figura 2.19. Para trocar o valor lógico enviado nos barramentos, os dispositivos devem chavear suas respectivas conexões com os barramentos.

No protocolo I2C, sempre existem os dispositivos que agem como *dispositivos mestres*(Masters) e os dispositivos que agem como *dispositivos escravos*(Slaves).

Um *dispositivo mestre* pode escolher com qual dos *dispositivos escravos* ele deseja se comunicar realizando a "mensagem de início".Após isso mandando os bits de endereço do *dispositivo escravo* são enviados no barramento de dados. É enviada, juntamente com uma mensagem do endereço, uma indicação, por parte do *dispositivo mestre* mostrando se ele deseja escrever ou ler do *dispositivo escravo*. Após isso, o *dispositivo escravo* deve enviar uma mensagem ACK para completar o estabelecimento da comunicação.Para enviar uma mensagem ACK, o *dispositivo escravo* seta o barramento de dados para o valor 0.

Tendo sido estabelecida a comunicação entre *dispositivo mestre* e *dispositivo escravo*, é incu-

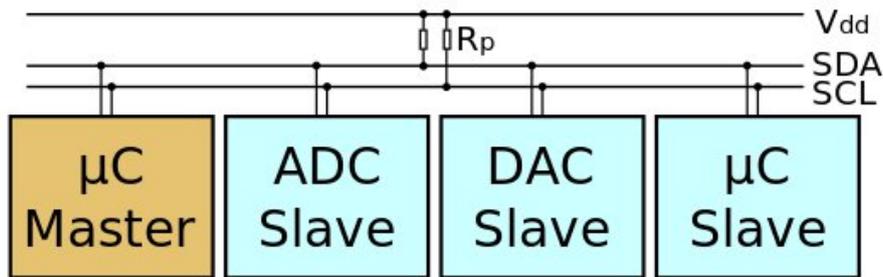


Figura 2.19: Figura esquemática do barramento serial I2C

Fonte: <http://docplayer.com.br/7275040-Monografia-de-graduacao.html>

bência do *dispositivo escravo* enviar, a cada 8 bits recebidos, uma mensagem ACK.

Os *dispositivos mestres* sempre retem o controle do barramento de clock. Quando um *dispositivo mestre* faz com que o barramento de clock tenha o valor lógico 0, é indicado para os *dispositivos escravos* que eles devem setar o barramento de dados com um bit 0 ou 1.

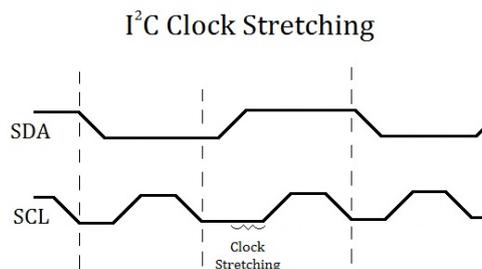


Figura 2.20: I2C - *Clock Stretching*

Fonte: <https://blog.digilentinc.com/index.php/i2c-how-does-it-work/>

Para assegurar o recebimento dos dados, os *dispositivo escravos* podem, possivelmente, realizar o chamado *Clock Stretching*, Figura 2.20, o qual consiste em manter o barramento de clock no nível 0, mesmo que o mestre o tenha setado para o nível. Isso é feito para, ampliar o tempo do processo de recebimento dos dados, por parte dos *dispositivo escravos*, para assegurar o sucesso de tal processo.

I2C oferece um bom suporte para a comunicação entre dispositivos eletrônicos que são acessados de forma ocasional. A vantagem competitiva da I2C sobre outros protocolos de comunicação de curta distância de baixa velocidade é que seu custo e complexidade não aumenta com o número de dispositivos no barramento.

Por outro lado, a complexidade dos componentes de software I2C de suporte pode ser significativamente mais elevada do que a de vários protocolos concorrentes (SPI e MicroWire, por exemplo)

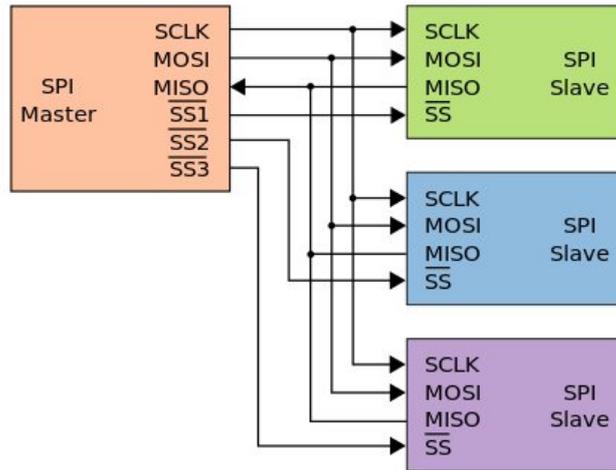


Figura 2.21: Barramento SPI - Um *dispositivo mestre* para *dispositivos escravos*

Fonte: <http://docplayer.com.br/3514866-Pratica-8-comunicacao-spi-8-1-introducao-e-objetivos-8-2-principios-basicos-do-protocolo-spi.html>

com uma configuração muito simples. Entretanto, seu modelo de endereçamento próprio, aliado com a forma de transferência simples de bytes para necessidades de comunicação simples.

O protocolo I2C é muito utilizado em projetos no placas Galileo utilizando a biblioteca: *Wire.h*

#### 2.4.2.4 Comunicação serial SPI

Assim como o protocolo I2C, o protocolo de Interface Serial com Periféricos - Serial Peripheral Interface (SPI) - tem como utilidade a comunicação de curta distância entre dispositivos eletrônicos [29].

No protocolo SPI, há apenas um *dispositivo mestre* para vários *dispositivo escravos*. A comunicação entre os *dispositivo mestre* e os *dispositivos escravos* é *full-duplex*, ou seja, os dispositivos citados podem se comunicar entre si em ambas direções. Os pinos *dispositivo mestre* e nos *dispositivos escravos*, como mostrados na Figura 2.21, são os seguintes:

- SCLK: Barramento Serial para o sinal de Clock originado no *dispositivo mestre*;
- MOSI: *Master Output Slave Input*; sinal originado no *dispositivo mestre*;
- MISO: *Master Input Slave Output*; sinal originado em um dos *dispositivos escravos*;
- SS: *Slave Select* sinal originado no *dispositivo mestre*.

O pino SS é utilizado pelo *dispositivo mestre* para selecionar com qual dos *dispositivos escravos* ele se comunicará (seja para receber mensagens ou enviar). Usualmente, quando um dos *dispo-*

*ativos escravos* é selecionado, todos outros, pela lógica *tri-state* das entradas SS, assumem altas impedância de entrada - o que significa que virtualmente tais escravos estão desconectados do circuito com o *dispositivo mestre*.

Primeiramente, no começo da comunicação com o dispositivo selecionado, é configurado no *dispositivo mestre* a frequência do clock que sai da pino SCLK.

Após isso, começa a ocorrer a troca de bits entre os dispositivos. Para cada bit que o pino MOSI recebe, é também recebido um bit no pino MISO.

Comparado a outros protocolos de inter-comunicação, o protocolo SPI oferece uma das maiores taxas de saída de bits. Isso se deve, dentre outros fatores, a não limitação do tamanho da palavra binária transmitida. As taxas de transmissão são, em geral, da ordem de MHz, entretanto tal taxa é intimamente ligada a velocidade do clock no *dispositivo mestre*, podendo portanto ser livremente aumentada. Além disso, os *dispositivos escravos* não necessitam de um endereço único como no protocolo I2C, daí, todas fase de reconhecimento e estabelecimento de comunicação é facilitada. Entretanto, tais facilidades tornam o protocolo com difícil depuração de erros e não há controle de fluxo nem nos *dispositivos escravos*.

SPI é utilizado em muitas aplicações. Dentre elas, por exemplo:

- Aplicações com sensores:
  - Comunicação com sensores de temperatura;
  - Comunicação com sensores de pressão;
  - Comunicação com sensores de toque.
- aplicações com tipos específicos de memória
  - Flash;
  - EEPROM.

Para fazer projetos com SPI na placa Galileo deve ser utilizada a biblioteca *SPI.h*

#### 2.4.2.5 Porta Serial UART

UART significa *Universal Asynchronous Receiver/Transmitter* (Receptor/Transmissor Universal Assíncrono). Um dispositivo UART é um microchip que tem como responsabilidade controlar a comunicação de um computador ou microcontrolador conectados serialmente. Essencialmente, um dispositivo UART é a dispositivo intermediário entre interfaces seriais e paralelas[30].

A Figura 2.22 mostra um modelo simplificado do que consiste um dispositivo UART. Na parte esquerda da figura, são mostrados os pinos de comunicação paralela pelo barramento de dados (Data Bus). O pinos R/W é utilizado para setar entre modos de leitura e escrita (Read/Write). O pino CLK é o pino do sinal de clock. O pino INT é o pino usado para interrupção de software para avisar o sistema que há dados para serem lidos/escritos no dispositivo UART.

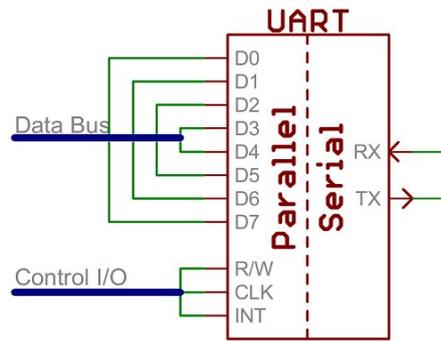


Figura 2.22: Modelo simplificado de um dispositivo UART

Fonte: <https://learn.sparkfun.com/tutorials/serial-communication/uarts>

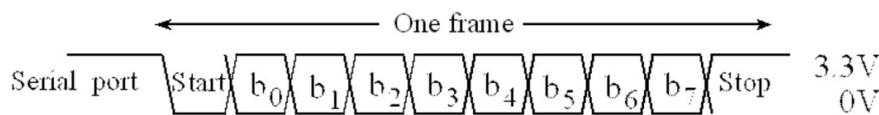


Figura 2.23: Frame UART para transmissão de 1 byte

Fonte: <http://www.mathworks.com/help/hdlverifier/examples/generate-fifo-interface-dpi-component-for-uart-receiver.html?requestedDomain=www.mathworks.com>

A Figura 2.23 mostra o chamado *frame* de dados da placa UART. O *frame* é composto de 10 bits. O primeiro bit é o bit de *start* utilizado para indicar o início do envio ou recebimento de um byte (8 bits) de dados. O bit *stop* indica o fim do frame.

Já a Figura 2.24 mostra em detalhes o processo que ocorre num dispositivo UART. Na figura, UART\_DR\_D é o *registrador de dados (Data Register)*, o qual é preenchido pela dispositivo que deseja realizar a comunicação utilizando o dispositivo UART. FIFO é a fila de recebimento(RX) ou transmissão de dados(TX). Ambas as filas tem 16 bits de tamanho. No caso da fila de recebimento de dados, 4 dos 16 bits são bits de flags para indicar erros na transmissão.

RXFE é uma flag que indica se que a fila de recebimento está vazia e RXFF é outra flag que indica que a fila de recebimento está vazia. Quanto as filas de transmissão, TXEF indica que a fila está vazia e TXFF indica que a fila está cheia. UOTX e UORX são *shift register* são os responsáveis pela transformação da comunicação em série para paralela e vice-versa.

O processo de transmissão de dados é o seguinte:

1. Dados armazenados no registrador de dados são enviados para a fila;
2. Caso a fila esteja vazia( flag TX), a fila recebe os bits do registrador de dados;
3. Os bits são enviados para o shift register UOTX, começando no b0 e sendo "shiftados" até o bit b7;

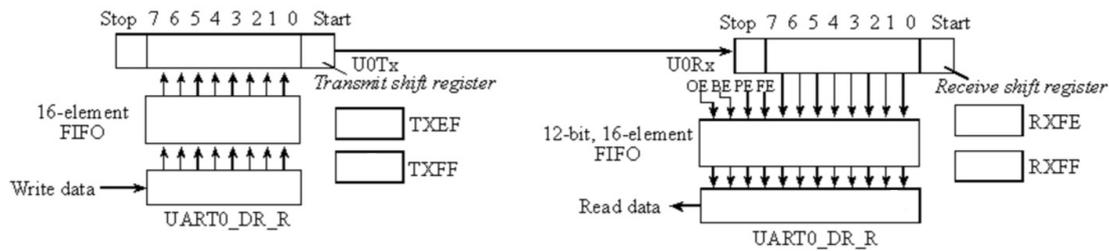


Figura 2.24: Modelo completo de um dispositivo UART

Fonte: [http://users.ece.utexas.edu/valvano/Volume1/E-Book/C11\\_SerialInterface](http://users.ece.utexas.edu/valvano/Volume1/E-Book/C11_SerialInterface)

4. Os bits armazenados no U0Tx são enviados de forma serial para o shift register receptor U0Rx;
5. Caso a fila de recepção esteja vazia (flag RX), os dados são colocados na pilha e lá permanecem até serem lidos.

UART é muito utilizado para projetos que requerem comunicação serial com Galileo ou projeto de Múltiplas Entradas e Saída Única (MISO) ou projeto com Entrada Única e Saída Múltipla (SIMO). Para trabalhar com UART, deve usar a biblioteca *SoftwareSerial.h*

#### 2.4.2.6 Memória Cache

Dentre as operações num sistema computacional, a operação mais demorada é o acesso à memória. Para evitar tais operações, é usada a chamada memória cache [31].

A memória cache faz parte da organização da memória de um sistema computacional. A memória num sistema computacional é organizada da seguinte forma, Figura 2.25:

- Memória de armazenamento (Storage Device - Memória ROM): Este nível de memória é o que possui o maior espaço, entretanto é a memória que demanda mais tempo para ser modificada, por isso, em geral, nesse nível ficam armazenados sistemas operacionais, arquivos de BOOT do sistema, firmwares, etc. A memória nesse nível não-volátil, o que significa que ela não é perdida ao se desligar o sistema;
- RAM (Random Access Memory): Este nível de memória é utilizado como memória principal. A memória RAM é de leitura e escrita. Essa memória é utilizada pelo CPU para armazenar e ler dados, arquivos e programas que estão sendo utilizados no momento. A memória RAM é uma memória volátil, o que significa que o conteúdo armazenado nela é perdido após o desligamento do sistema;
- A memória cache é a parte da memória utilizada pela unidade de processamento central (CPU) de um computador para reduzir o tempo médio necessário para ler ou escrever aos

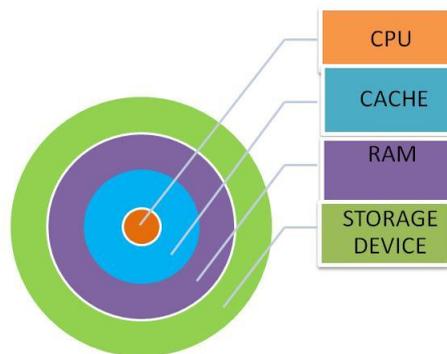


Figura 2.25: Organização da memória num sistema computacional.

dados a partir da memória principal. A memória cache é uma memória menor, mais rápida que armazena cópias dos dados de localizações de memória principais utilizados com frequência para evitar a repetição de acessos lentos. A maioria dos processadores têm diferentes caches independentes, incluindo instruções e dados caches, onde o cache de dados é normalmente organizadas como uma hierarquia de níveis mais cache (L1, L2, etc);

- CPU: Na CPU está armazenada toda arquitetura de instruções do sistema computacional. A CPU é responsável pela gerência de todos processos que ocorrem no computador e ela utiliza a memória cache para realizar a maior parte de suas operações.

Para toda operação que a CPU executa a qual necessita de certo dado da memória, é sempre verificado, primeiramente, se o dado já se encontra na memória cache. Caso o dado não se encontre na cache, é solicitado dos níveis mais baixos da memória o dado em questão. Caso o dado já se encontre na cache, ele é lido e processado rapidamente pela CPU.

Microcontroladores simples, em geral, não possuem a memória cache, visto que toda sua estrutura é simplificada. No caso da placa Galileo e placas similares, a memória cache é necessária, visto que tais sistemas podem, inclusive, executar sistemas operacionais e tem grande quantidade de memória de armazenamento.

Atualmente, vem-se dividindo a memória cache em níveis: cache L1, cache L2, cache L3, etc. Tal divisão é feita para amplificar o efeito de manter na cache os dados de memória usualmente acessados. A cache L1 contém os dados acessados mais frequentemente, a cache L2 contém os dados acessados frequentemente, mas não tanto quanto os dados na cache L1, etc.

No caso da placa Galileo, há apenas um nível de cache: a cache L1 com 16 KBytes, como mostrado na Seção 2.4.2.

#### 2.4.2.7 Memória SRAM

Memória SRAM (Static Random Access Memory) é o tipo de memória de acesso aleatório geralmente utilizado no nível de memória cache. Ser de de acesso aleatório significa que qualquer

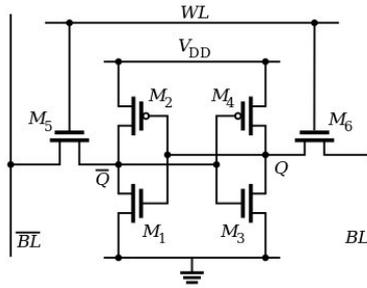


Figura 2.26: Estrutura da célula de memória SRAM

Fonte: <http://web.sfc.keio.ac.jp/~rdv/keio/sfc/teaching/architecture/architecture-2009/lec08-cache.html>

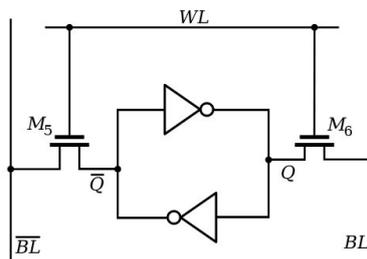


Figura 2.27: Estrutura da célula SRAM com dois inversores.

Fonte: <http://web.sfc.keio.ac.jp/~rdv/keio/sfc/teaching/architecture/architecture-2009/lec08-cache.html>

porção da memória é acessada num tempo igual. SRAM é uma memória estática, o que significa que o dados se manterá armazenado durante um largo intervalo de tempo depois do desligamento do sistema.

A memória SRAM é construída utilizando com flip-flops com transistores MOSFETs. A Figura 2.26 mostra a estrutura básica de armazenamento de uma célula de um bit da memória SRAM.

Resumidamente, os transistores M1, M2, M3 e M4 são os responsáveis por guardar o bit[32]. A estrutura do circuito formada por M1, M2, M3 e M4 realiza a realimentação do bit, sendo responsável pelo qualidade de memória estática que a SRAM possui. A Figura 2.27 mostra como os transistores M1, M2, M3 e M4 podem ser vistos como um par de emissores. Quando um Q, na Figura 2.27, é igual a 1, seu oposto,  $\bar{Q} = 0$ , é criado na saída do inverso e o sinal  $Q = 1$  é realimentado pelo segundo inversor.

Dessa maneira, a estrutura da célula de um bit de memória SRAM torna desnecessário *re-carregamento* do dado armazenado. Os transistores M5 e M6 são usados para ler ou escrever da célula de memória por meio das linhas de bit BL e  $\bar{BL}$ . Tal processo de leitura ou escrita pode ser realizado, em média, em 2ns, velocidade a qual é bastante alta para sistemas computacionais.

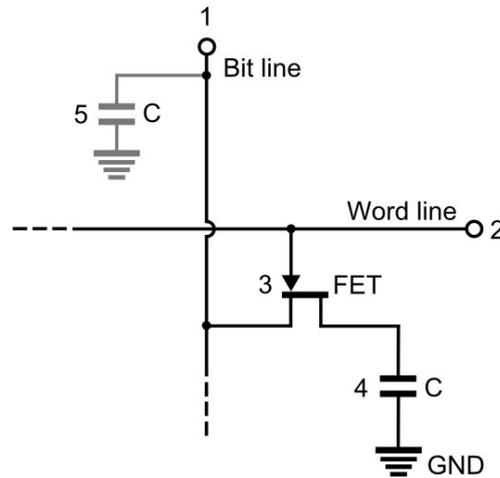


Figura 2.28: Estrutura da célula de memória DRAM.

Fonte: <http://users.ece.gatech.edu/sudha/academic/class/ece2030/Lectures/memory/>

A memória SRAM é utilizada nos mais variados ambientes como: computadores pessoais, microcontroladores, FPGAs, etc. Na placa Galileo, existem 512 Kbytes de SRAM integrados, tornando a placa Galileo altamente eficiente no tocante ao acesso e atualização da memória.

#### 2.4.2.8 Memória DRAM

Dynamic Random Access Memory (DRAM) é uma memória de acesso aleatório como a SRAM. Ao contrário da memória SRAM, a memória DRAM é uma memória *dinâmica*, o que significa que os dados armazenados precisam ser periodicamente recarregados.

Os bits, na memória DRAM, são armazenados numa estrutura de um capacitor juntamente com um transistor. O capacitor estar carregado significa o bit 1, e o capacitor está descarregado significa o bit 0. Na Figura 2.29 é mostrada a estrutura de uma célula de memória DRAM. O capacitor marcado pelo número 4 é onde o bit é armazenado.

O processo de escrita no bit é da seguinte forma:

- 1) Na linha marcada pelo número 1 (Bit Line) é escrito um bit lógico 0 ou 1 (0 ou +Vcc Volts);
- 2) A linha marcada pelo número 2 ativa o transistor conectando a Bit line com o capacitor C (marcado pelo número 4).

O processo de leitura é feita da mesma forma que o processo de escrita, entretanto, a Bit line possui capacitância parasita apreciável. Na figura, essa capacitância é marcada pelo número 5. Tal capacitância parasita diminui a velocidade do processo de leitura por tomar parte da carga armazenada no capacitor marcado pelo número 4 para si.

O descarregamento natural dos capacitores, ainda que em circuito aberto, e a existência de capacitores parasitas na célula de memória trazem a necessidade de circuitos responsáveis por recarregar, a cada leitura, as células DRAM.

O tempo médio de leitura na memória DRAM é de 64 ns. Pelo tempo de leitura e pela necessidade de recarregamento, em geral, a memória DRAM é usada para memórias menos acessadas.

A placa Galileo possui 256 MByte de memória DRAM gerenciados pelo sistema operacional.

#### 2.4.2.9 Memória EEPROM

*Electrically Erasable Programmable Read-Only Memory* ou EEPROM é uma memória não volátil, o que significa que os dados não são apagados após o desligamento do sistema. A memória EEPROM é similar à memória FLASH. Assim como ela, a memória EEPROM escrita aproximadamente 100.000 vezes. A principal vantagem que a memória EEPROM apresenta em relação a memória FLASH é que ela deve escrever em bytes individualmente, enquanto na memória FLASH é necessário escrever um setor inteiro para alterar bytes individuais. Tal característica torna a memória FLASH mais rápida e com vida-útil menor que a memória EEPROM.

Na placa Galileo há 11 Kbytes memória EEPROM. A EEPROM pode ser programada na placa Galileo com a biblioteca *EEPROM.h*.

#### 2.4.2.10 Clock de tempo real - RTC

Um clock de tempo (RTC) é um clock comum de um sistema computacional com a funcionalidade de ter armazenado nele tempo atual, mesmo que o sistema esteja desligado. Quase todos equipamentos eletrônicos atuais, como computadores, celulares, etc, possuem um clock de tempo real integrado. O tempo atual pode ser adquirido com outros equipamentos além do RTC, entretanto, o RTC têm as seguintes vantagens:

- Baixo consumo de energia;
- O fato de ser um sistema independente do sistema central, faz com este tenha seu processamento livre para outras tarefas.

#### 2.4.2.11 Barramento Mini PCI-Express

Mini PCI-Express é um barramento de alta velocidade de transmissão de dados com 52 pinos. Por meio desses 52 pinos, existem as seguintes conexões:

- Conexões para o barramento PCI Express x1;
- Conexões para USB 2.0;
- Conexões para SMBus;

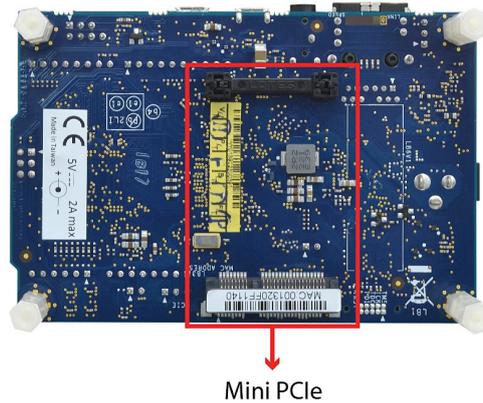


Figura 2.29: Estrutura da célula de memória DRAM.

- Conexões para LEDs de diagnóstico de conexões wireless;
- Conexões para SIM Card;
- Conexões para outras extensões PCI;
- Saída de 1.5V e 3.3 V.

#### 2.4.2.12 USB

USB ou *Universal Serial Bus* é um padrão cabos, conectores e protocolos. O propósito do USB é padronizar a comunicação com equipamentos periféricos como teclados, câmeras, impressoras, telefones, etc. USB já passou por três padronizações:

- USB 1.0 com velocidade máxima de transmissão de dados de 12 Mbits/s;
- USB 2.0 com velocidade máxima de transmissão de dados 480 Mbits/s;
- USB 3.1 com velocidade máxima de transmissão de 10 dados Gbits/s.

Para comunicação com periféricos, USB já tem conseguido substituir com sucesso a comunicação serial e paralela.

A topologia USB é assimétrica em formato de estrela com um dispositivo central (Host), como mostrado no exemplo da Figura 2.30.

Quando um novo equipamento é conectado, o sistema operacional do dispositivo central, a placa Galileo por exemplo, detecta a nova conexão e solicita o driver do equipamento para possibilitar a comunicação. Como mostrado na Figura 2.31, os cabos e conexões USB obedecem os padrões de duas classes: a classe A e a classe B.

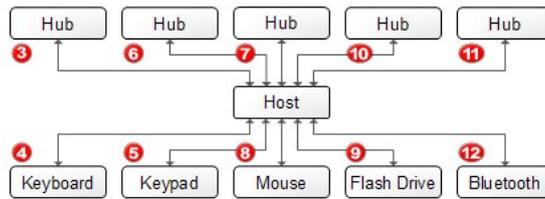


Figura 2.30: Exemplo: Topologia Estrela USB

Fonte: <http://www.usblyzer.com/usb-topology.htm>

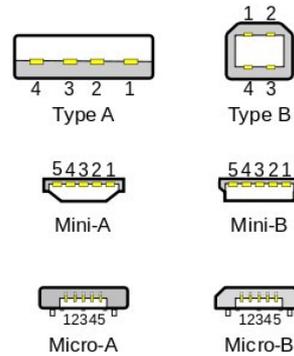


Figura 2.31: Pinos USB

Fonte: [http://www.robotizando.com.br/pinagem\\_usb.php](http://www.robotizando.com.br/pinagem_usb.php)

Quando o dispositivo central é ligado, é definido para cada dispositivo conectado um endereço. Tal processo inicial é chamado de *enumeração*. Durante a *enumeração*, é também solicitado a cada equipamento o tipo de transferência de dados a ser realizado com ele:

- Transferência de dados por meio de interrupção: Transferência de dados pouco frequente e de baixa quantidade, como transferência com teclado e mouse. Nesse caso, vale a pena interromper o sistema operacional.
- Transferência de dados por meio de pacotes: Transferência de dados pouco frequentes e de grande quantidade de dados, como, por exemplo, a transferência realizada para impressoras. Nesse caso, um bloco de dados é transferido de uma vez só pela porta USB.
- Transferência de dados isócrona (tempo real): Transferência de dados frequente e contínua, como as necessárias num alto falante.

Para cada uma das formas de transferência de dados supracitadas, é reservado pelo USB a largura de banda necessária em frames de largura de banda.

### 2.4.2.13 JTAG

JTAG(Joint Test Action Group) é a padronização IEEE-1149.1 usada para testes de circuitos impressos. JTAG foi criada para ajudar no problema da crescente dificuldade de testar circuitos associada com a crescente diminuição dos tamanhos do circuitos. Como mostrado na Figura 2.32, a implementação mais simples de JTAG requer 4 fios para sinalização:

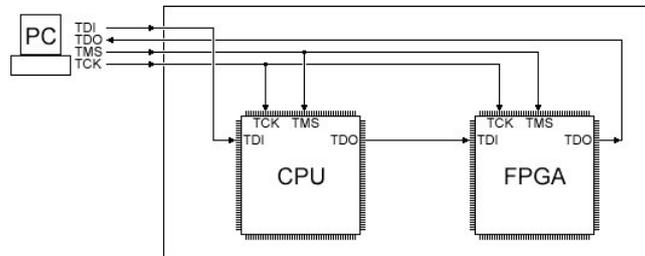


Figura 2.32: JTAG monitorando a conexão de um CPU com uma FPGA

Fonte: [https://courses.cit.cornell.edu/ee476/FinalProjects/s2009/jgs33\\_rrw32/Final20Paper/](https://courses.cit.cornell.edu/ee476/FinalProjects/s2009/jgs33_rrw32/Final20Paper/)

- TDI: Pino para sinal de entrada para a query de teste.
- TDO: Pino para sinal de saída para a query de teste.
- TCK: Sinal do relógio de sincronização do JTAG. Todos outros sinais(TDI, TDO, TMS) são síncronos a esse sinal.
- TMS: Sinal para controlar o estado da máquina de estados interna ao JTAG, a qual tem 16 estados distintos, como mostrado na Figura 2.33.

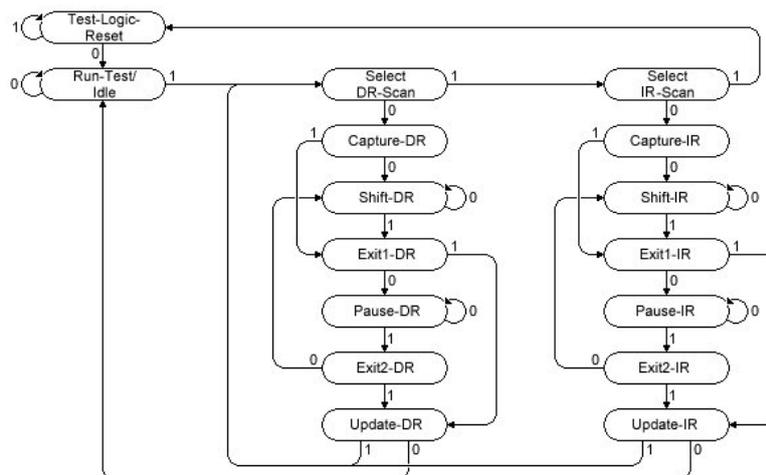


Figura 2.33: Máquina de estados - JTAG

Fonte: [https://courses.cit.cornell.edu/ee476/FinalProjects/s2009/jgs33\\_rrw32/Final20Paper/](https://courses.cit.cornell.edu/ee476/FinalProjects/s2009/jgs33_rrw32/Final20Paper/)

Na máquina de estados mostrada na Figura 2.33, geralmente a JTAG é levada para os estados *Shift-DR*, em primeira instância, e, após isso, levada para o estado *Shift-IR*, onde o dado é coletado. *Shift-DR* e *Shift-IR* tem o mesmo tamanho  $N$  de bits. Por exemplo, se *Shift-DR* e *Shift-IR* tiverem 6 bits de tamanho, após 6 clock realizados no TCK, o dado que chegou no *Shift-IR* chega no *Shift-DR*.

A Figura 2.34 mostra fluxo de dados de ida e volta num debug JTAG: JTAG -> CPU -> FPGA. O dado sai pelo pino TDI, percorre a CPU e a FPGA e volta no pino TDO sendo tudo isso controlado pelo pino TMS tendo todos esses pinos sincronizados pelo pino TCK.

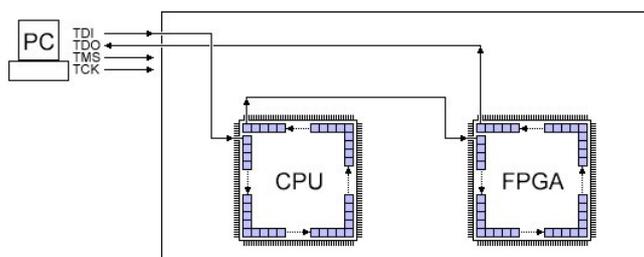


Figura 2.34: Fluxo de dados num debug JTAG

Fonte: <http://www.fpga4fun.com/JTAG.html>

A verificação de entrada e saída, incluindo o tempo de tais eventos, com testes JTAG tornou possível testes complexos em circuitos integrados.

A placa Galileo, como grande partes dos circuitos integrados atuais, possui 9 pinos próprio para debug com JTAG.

## Capítulo 3

# Proposta de atualização do curso de APC

A disciplina proposta seguirá a ementa proposta na Tabela 3.1. Na ementa antiga, na qual está se baseia, a proporção entre horas de aulas teóricas e práticas é de 4 para 2 horas. Foi proposta a troca para 4 horas de atividades práticas e 2 horas de atividades teóricas.

As atividades práticas não devem ser apenas laboratórios, mas devem contemplar, também, os seguintes aspectos relativos a gerência de projetos seguindo a metodologia Scrum:

1. Reuniões das equipes Scrum formadas pelos alunos para:

- Atualização do quadro de atividades Scrum (*Scrum Backlog*);

- Reuniões de revisão de *Sprint*;

- Aprendizagem colaborativa;

- Desenvolvimento de projetos propostos;

- Reuniões com o professor que atuará como *Scrum Master*.

Tabela 3.1: Proposta de ementa

<b>CIC - Computação básica e tópicos em aprendizagem ativa e eletrônica</b>	
<b>Créditos</b>	6 (Teoria:2, Prática:4, Estudo em casa:4), 1 <sup>o</sup> período
<b>Ementa</b>	<p>Estudo dos seguintes tópicos de computação:</p> <ol style="list-style-type: none"> <li>1) Histórico da computação. Sistemas de numeração.</li> <li>2) Arquitetura de Computadores. Linguagens formais.</li> <li>3) Algoritmos e programas. Programação estruturada.</li> <li>4) Identificadores e tipos.</li> <li>5) Operadores e expressões.</li> <li>6) Estruturas de controle: condicional e repetição.</li> <li>7) Entrada e saída de dados.</li> <li>8) Estruturas de dados estáticas:</li> <li>9) Agregados homogêneos e heterogêneos.</li> <li>10) Iteração e recursão.</li> <li>11) Subalgoritmos.</li> <li>12) Implementação de soluções utilizando o fragmento estruturado de uma linguagem de programação</li> </ol> <p>Aplicando os seguintes conceitos de eletrônica básica:</p> <ol style="list-style-type: none"> <li>1) Circuitos</li> <li>2) Resistores e Capacitores</li> <li>3) Lei de Ohm</li> <li>4) Leis de Kirschoof</li> <li>5) Sensores</li> <li>6) Interruptores</li> <li>7) Programação de microcontroladores baseados em Arduino</li> </ol>
<b>Bibliografia</b>	<p>Basic Electronics. New York: Dover Publications, 1973. ISBN 978-0486210766.[33]</p> <p>KERNIGHAN, B. The C programming language. Englewood Cliffs, N.J: Prentice Hall, 1988. ISBN 978-0131103627.</p>
<b>Pré-requisitos</b>	Sem pré-requisitos

Para se construir um projeto Scrum, como exposto na Seção 2.2, para o contexto educacional da teoria de masterização de habilidades proposta por Bloom, os seguintes passos são necessários:

1. Realizar adaptação dos princípios e fluxo da metodologia Scrum, em conjunto com os princípios da metodologia de *masterização de habilidades* para o contexto educacional desejado.
2. Definir os objetivos educacionais na forma de *características de produto* utilizando a taxonomia de Bloom para criar as *User Stories* e *Epics* e definir os critérios de aprovação para cada objetivo de masterização.

3. Criar atividades com a placa Galileo de acordo com o conteúdo programático estabelecido

Cada uma das atividades listadas acima será tratada, neste trabalho, em uma seção em separado.

### 3.1 Aprendizagem por masterização e Scrum adaptados ao curso de Algoritmos e Programação de Computadores

Nesta seção é apresentada uma proposta de adaptação da metodologia Scrum para aplicação na disciplina *Algoritmos e Programação de Computadores*.

A Tabela 3.2 mostra um resumo de todo fluxo mostrado na Figura ?? e, ao lado, a proposta de adaptação:

Tabela 3.2: Scrum aplicado à *Algoritmos e Programação de Computadores*

	Scrum	Scrum + Masterização de habilidades
<b>Por que utilizar Scrum?</b>	<ul style="list-style-type: none"><li>-Uso de ciclo curtos de desenvolvimento e revisão(Sprints)</li><li>-Revisão frequente dos requisitos de projeto a cada Sprint</li><li>- Melhora contínua</li><li>-Valor agregado acima de obediência a cronograma e processos</li><li>- Entrega contínua de Valor</li></ul>	<ol style="list-style-type: none"><li>1)Verificação,de níveis de aprendizagem frequentes</li><li>2)Aumento, a cada Sprint, da responsabilização pessoal dos alunos com relação a seus estudos</li></ol>

**Scrum****Scrum + Masterização de habilidades**

	<b>Scrum</b>	<b>Scrum + Masterização de habilidades</b>
<b>Papéis</b>	<p>- Product Owner Representante da voz do cliente</p> <p>-Scrum Master Gerente do time Scrum responsável por certificar-se que os princípios Scrum estão sendo seguidos pela equipe e também responsável por coletar os requisitos de projeto junto ao Product Owner</p> <p>-Time Scrum -&gt;Equipe responsável por desenvolver os requisitos de projeto descritos, inclusive, por meio de User Stories</p>	<p>Product Owner - Professor</p> <p>Scrum Master - Professor (enquanto os alunos não estiverem familiarizados com a metodologia Scrum e prontos para auto-gerência de seus estudos)</p> <p>Time Equipe Scrum - Equipe de até 4 estudantes responsáveis( sem contar o professor) por concluir, conjuntamente, os objetivos de aprendizagem descritos por meio da taxonomia de Bloom.</p> <p>A equipe Scrum deve eleger um representante novo a cada Sprint para se comunicar diretamente com o professor (Scrum Master) e planejar as atividades e meios de verificação conjuntamente</p>

	<b>Scrum</b>	<b>Scrum + Masterização de habilidades</b>
<b>Fase 1 - Início Definição de projeto</b>	<ol style="list-style-type: none"> <li>1) Criar visão de projeto</li> <li>2) Identificar Scrum Master e Stakeholders</li> <li>3) Formar Time Scrum</li> <li>4) Desenvolver Epics (Conjunto de user stories)</li> <li>5) Criar conjunto prioritário de características do produto</li> <li>6) Criar um plano de desenvolvimento do produto em Sprints sequenciais</li> </ol>	<ol style="list-style-type: none"> <li>1), Criar objetivos educacionais com a taxonomia de Bloom</li> <li>2) Scrum Master = Professor</li> <li>Stakeholders =, alunos, famílias, governo, etc</li> <li>3) Formar grupos de 3 ou 4 alunos, preferencialmente não amigos e com características complementares</li> <li>4) Dividir os objetivos educacionais em módulos</li> <li>5) Criar plano de ascensão dos alunos nos níveis cognitivos de Bloom para os conteúdos especificados</li> <li>6) Criar especificação de Sprint com os objetivos educacionais</li> </ol>
<b>Fase 2 - Plano de Projeto e Estimação</b>	<ol style="list-style-type: none"> <li>1) Critérios de aceitação são criados (a ser feito pelo Product Owner juntamente com o Scrum Team)</li> <li>2) Scrum Master e time estimam o esforço necessário para desenvolver as User Stories</li> <li>3) Scrum Master e time se comprometem a desenvolver o produto de acordo com as Epics (Conjunto de User Stories) e critérios de aceitação</li> <li>4) User Stories são divididas em subtarefas para criar uma Task List</li> <li>5) Time Scrum se reúne para decidir quais tarefas serão realizadas na Sprint (Sprint Backlog)</li> </ol>	<ol style="list-style-type: none"> <li>1) O professor, a cada Sprint, deve deixar claro quais serão os critérios de masterização de habilidades preferencialmente utilizando a taxonomia de Bloom</li> <li>2) Professor é responsável, a cada Sprint, por estimar o esforço necessário por parte da turma para as masterizações desejadas.</li> </ol> <p>A participação dos representantes de equipes com relação à definição de planos Sprint é crescente ao longo dos Sprints.</p>

	<b>Scrum</b>	<b>Scrum + Masterização de habilidades</b>
<b>Fase 3 - Implementação</b>	1) Criar os "entregáveis" do Sprint 2) Realizar reuniões diárias rápidas de no máximo 15 minutos para discutir problemas e progresso 3) Atualizar o quadro Scrum antes de iniciar o dia de trabalho	1) Estudar o conteúdo especificado em grupo com a ajuda do Scrum Master (Professor) 2) Realizar reuniões diárias rápidas de no máximo 15 minutos para discutir problemas e progresso 3) A equipe Scrum decide os meios de aprendizagem que usarão com auxílio do professor. A cada Sprint, a equipe Scrum deve ficar mais livre para decidir os meios mais eficientes de aprendizagem 4) Atualizar o quadro Scrum de ( A FAZER , FAZENDO, FEITO)
<b>Fase 4 - Fim de Sprint; Revisão e Retrospecto</b>	1) Os entregáveis são mostrados aos Stakeholders 2) Essas reuniões são feitas para garantir a aprovação do produto desenvolvido e fazer os ajustes necessários o mais rápido possível 3) O Scrum Master e o time Scrum se reúnem para discutir o que foi aprendido no Sprint 4) Informação é documentada para futuros Sprints	1) A equipe Scrum desenvolve testes rápidos ( para serem resolvidos em no máximo 20 minutos) para que as outras equipes resolvam do conteúdo estudado no Sprint. O gabarito deve ser entregue ao professor antes da aplicação dos testes para que este verifique-os e decida quais testes serão aplicados 2) A equipe resolve, no tempo estipulado, os testes compartilhados 3) Após os testes de unidade, o professor e os alunos tem uma aula para revisar o que foi aprendido de acordo com os objetivos educacionais traçados

### 3.1.1 Definição de objetivos educacionais na forma da metodologia Scrum

Para realizar a definição dos objetivos educacionais, foi utilizada a plataforma web scrumdo<sup>®</sup> sendo cadastrados os seguintes objetivos educacionais próprios da disciplina *Algoritmos e Programação de Computadores*:

1. O histórico da computação;
2. Organização básica de um computador;
3. Introdução ao conceito de algoritmo;
4. Pseudocódigo e Fluxograma;
5. Tipos de variáveis de memória;
6. Operadores e expressões;
7. Algoritmos sequenciais;
8. Algoritmos com alternativas (simples, compostas, aninhadas e de múltipla escolha);
9. Algoritmos com repetição (com teste no início, com teste no fim e com variável de controle);
10. Algoritmos com vetores e matrizes;
11. Subalgoritmos, passagem de parâmetros;
12. Ponteiros;
13. Recursividade;
14. Registros;
15. Arquivos;
16. Ordenação e busca.

Aos objetivos listados, devem ser incluídos os seguintes objetivos educacionais relativos a placa Intel® Galileo e eletrônica básica:

1. Leis básicas de eletrônica( Leis de Kirschoff);
2. Sistema da placa Galileo;
3. Ferramentas básicas de prototipação eletrônica;
4. Fundamentos de programação em Galileo;
5. Introdução a sensores;
6. Interação com sensores;
7. Uso de displays;
8. Motores;

9. Tópico mais avançado em eletrônica a ser escolhido com a Turma; (Comunicação Ethernet ou Internet, Comunicação sem fio, Uso de placa SD de armazenamento de dados, uso de interrupções, Circuitos integrados periféricos, controle de altas cargas, etc).

A Figura 3.1 mostra o plataforma web scrumdo<sup>®</sup> para cadastramento de quadros scrum pertinentes.

**Setup Board** *Choose an initial configuration for your project.*

**Board Wizard Step 3 of 3**

What rows should your board have? You can have just one, or multiple rows are often used to differentiate between classes or services, teams, or types of work.

Dificuldades

Disciplina Corrente

Trabalhos com Intel Galileo

Back Done

Need inspiration? Try a preset: Presets...

**Board Preview**

**Dificuldades**

A fazer	Fazendo	Revisando	Feito

**Disciplina Corrente**

A fazer	Fazendo	Revisando	Feito

**Trabalhos com Intel Galileo**

A fazer	Fazendo	Revisando	Feito

Figura 3.1: Plataforma scrumdo: Formato dos quadros Scrum planejados para a disciplina *Algoritmos e Programação de Computadores*

A Figura 3.2 mostra os quadros *Dificuldades Algoritmos e Programação de Computadores*, *Aplicação com Galileo*. A disposição dos 3 quadros deve refletir a prioridade da turma. As reuniões de Sprint devem servir para resolver questões pendentes posta no quadro *Dificuldades*.

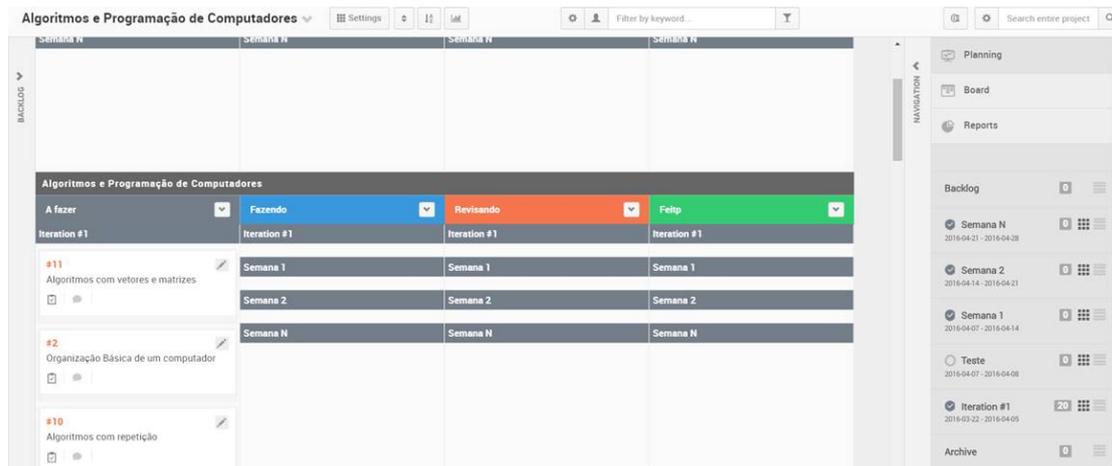


Figura 3.2: Plataforma scrumdo: Quadros Scrum criados e objetivos educacionais, segundo a Taxonomia de Bloom escritos *Algoritmos e Programação de Computadores*

Pode-se, por meio da plataforma, selecionar sub-objetivos. No caso deste trabalho, para cada tópico listado para as práticas com a placa Galileo, foram criados objetivos educacionais para os três primeiros níveis cognitivos identificados por Bloom (Nível de Conhecimento, Compreensão a Aplicação).

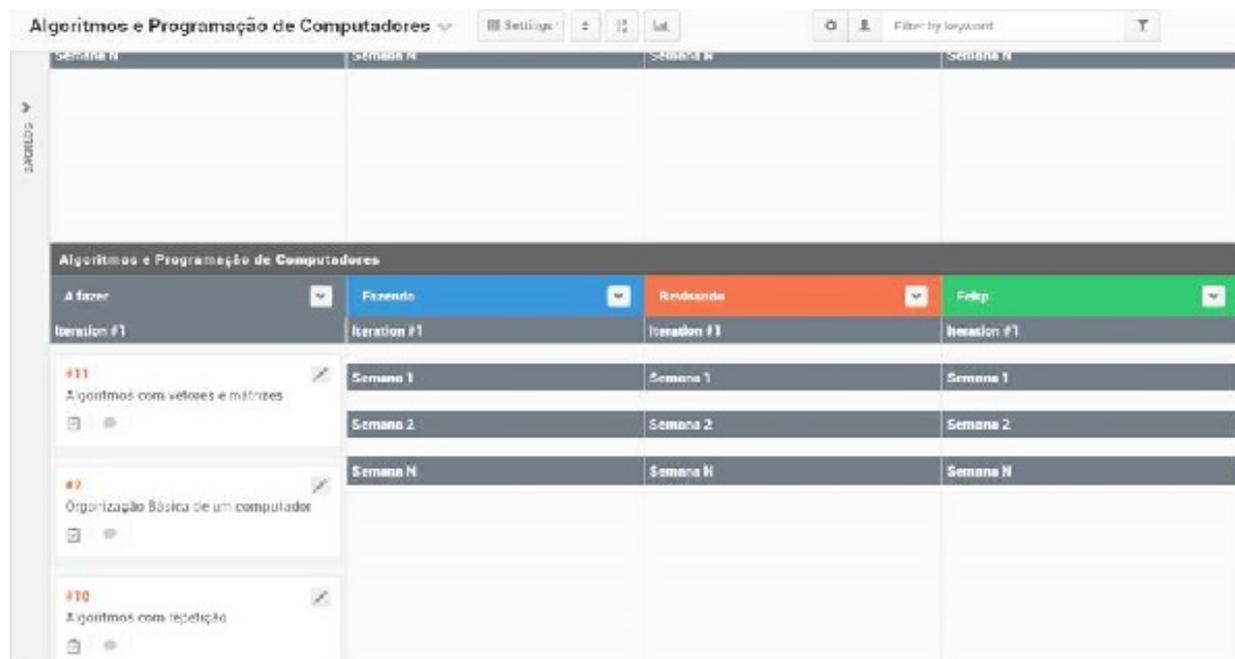


Figura 3.3: Plataforma scrumdo: Definição de sub-objetivos de um objetivo listado no *Product Backlog Algoritmos e Programação de Computadores*

### 3.1.2 Definição de objetivos educacionais para a Disciplina *Algoritmos e Programação de Computadores*

Para especificar cada objetivo educacional, foi utilizado o aplicativo desenvolvido em QT, como descrito anteriormente e como mostrado na Figura ??.

A seguinte enumeração mostra uma possível especificação de cada um dos módulos de aprendizagem definidos anteriormente seguindo a *Taxonomia de Bloom*:

#### 1. O histórico da computação

**Nível 1 Conhecimento:** Na unidade **histórico da computação**, os estudantes irão listar os principais acontecimentos e cientistas que criaram a computação tal qual a conhecemos hoje em um grupo de 4 pessoas.

**Nível 2 Compreensão:** Na unidade histórico da computação, os estudantes irão explicar a relevância da computação no mundo moderno em um grupo de 4 pessoas.

**Nível 3 Aplicação:** Na unidade histórico da computação, os estudantes irão escolher, dentre certos sistemas computacionais exposto, qual seria aquele(s) que mais se adequariam ao contexto exposto em um grupo de 4 pessoas.

#### 2. Organização básica de um computador

**Nível 1 Conhecimento:** Na unidade organização básica de um computador, os estudantes irão enumerar e explicar o funcionamento de todos os sistemas que compõem um computador em um grupo de 4 pessoas.

**Nível 2 Compreensão:** Na unidade organização básica de um computador, os estudantes irão classificar computadores quanto a parâmetros de desempenho entre outros em um grupo de 4 pessoas.

**Nível 3 Aplicação:** Na unidade organização básica de um computador, os estudantes irão ilustrar a dinâmica de funcionamento de um computador e criarão um diagrama em um grupo de 4 pessoas.

#### 3. Introdução ao conceito de algoritmo

**Nível 1 Conhecimento:** Na unidade introdução ao conceito de algoritmo, os estudantes irão definir conceitualmente o que são algoritmos em um grupo de 4 pessoas.

**Nível 2 Compreensão:** Na unidade introdução ao conceito de algoritmo, os estudantes irão identificar vários tipos diferentes algoritmos em um grupo de 4 pessoas.

**Nível 3 Aplicação:** Na unidade introdução ao conceito de algoritmo, os estudantes irão empregar um determinado tipo de algoritmo para cada situação dada em um grupo de 4 pessoas.

#### 4. Pseudocódigo e Fluxograma

**Nível 1 Conhecimento:** Na unidade pseudocódigo e fluxograma, os estudantes irão duplicar por meio das ferramentas citadas, situações do cotidiano em um grupo de 4 pessoas.

**Nível 2 Compreensão:** Na unidade pseudocódigo e fluxograma, os estudantes irão selecionar por meio dos pseudocódigos e fluxogramas já construídos quais são os mais eficientes para cada situação dada em um grupo de 4 pessoas.

**Nível 3 Aplicação:** Na unidade pseudocódigo e fluxograma, os estudantes irão programar os primeiros algoritmos em um grupo de 4 pessoas.

5. Tipos de variáveis de memória

**Nível 1 Conhecimento:** Na unidade tipos de variáveis de memória, os estudantes irão listar todos os tipos de variáveis em um grupo de 4 pessoas.

**Nível 2 Compreensão:** Na unidade tipos de variáveis de memória, os estudantes irão explicar as diferenças de usabilidade entre todos os tipos de variáveis em um grupo de 4 pessoas.

**Nível 3 Aplicação:** Na unidade tipos de variáveis de memória, os estudantes irão escolher dentre todos os tipos de variável de memória, aquelas que melhor se aplicam às situações propostas e explicar o motivo em um grupo de 4 pessoas.

6. Operadores e expressões

**Nível 1 Conhecimento:** Na unidade operadores e expressões, os estudantes irão enumerar os vários tipos de operadores e expressões na linguagem C e demonstrar seu uso em um grupo de 4 pessoas.

**Nível 2 Compreensão:** Na unidade operadores e expressões, os estudantes irão descrever o resultado de várias expressões escritas, explicitando os passos desenvolvidos em um grupo de 4 pessoas.

**Nível 3 Aplicação:** Na unidade operadores e expressões, os estudantes irão desenvolver programas para solução de diversas situações propostas em um grupo de 4 pessoas.

7. Algoritmos sequenciais

**Nível 1 Conhecimento:** Na unidade algoritmos sequenciais, os estudantes irão definir o conceito, todas as expressões e operadores utilizados em um grupo de 4 pessoas.

**Nível 2 Compreensão:** Na unidade algoritmos sequenciais, os estudantes irão definir o resultado esperado na utilização de um algoritmo sequencial em um grupo de 4 pessoas.

**Nível 3 Aplicação:** Na unidade algoritmos sequenciais, os estudantes irão empregar algoritmos sequenciais na solução de situações diversas em um grupo de 4 pessoas.

8. Algoritmos com alternativas (simples, compostas, aninhadas e de múltipla escolha)

**Nível 1 Conhecimento:** Na unidade algoritmos com alternativas, os estudantes irão definir os vários operadores condicionais utilizados em C quanto a sua dinâmica e utilização em um grupo de 4 pessoas.

**Nível 2 Compreensão:** Na unidade algoritmos com alternativas, os estudantes irão identificar o resultado de uma série de condicionais aplicados num algoritmo sequencial em um grupo de 4 pessoas.

**Nível 3 Aplicação:** Na unidade algoritmos com alternativas, os estudantes irão construir programas complexos com alternativa condicionais em um grupo de 4 pessoas.

9. Algoritmos com repetição (com teste no início, com teste no fim e com variável de controle)

**Nível 1 Conhecimento:** Na unidade algoritmos com repetição, os estudantes irão lembrar a sintaxe correta de todas formas de laços de repetição em C em um grupo de 4 pessoas.

**Nível 2 Compreensão:** Na unidade algoritmos com repetição, os estudantes irão explicar a diferença de usabilidade das formas de laços de repetição em C em um grupo de 4 pessoas.

**Nível 3 Aplicação:** Na unidade algoritmos com repetição, os estudantes irão construir programas complexos com laços de repetição em um grupo de 4 pessoas.

10. Algoritmos com vetores e matrizes

**Nível 1 Conhecimento:** Na unidade algoritmos com vetores e matrizes, os estudantes irão compreender o que são vetores e matrizes.

**Nível 2 Compreensão:** Na unidade algoritmos com vetores e matrizes, os estudantes irão compreender o utilidade de vetores e matrizes e seus respectivos usos em programas complexos.

**Nível 3 Aplicação:** Na unidade algoritmos com vetores e matrizes, os estudantes irão construir programas complexos com utilizando tais conceitos em um grupo de 4 pessoas.

11. Subalgoritmos e passagem de parâmetros

**Nível 1 Conhecimento:** Na unidade subalgoritmos e passagem de parâmetros, os estudantes irão expor por meio de exemplo e aplicações, a forma sintaticamente correta de escrita de subalgoritmos e passagem de parâmetros em um grupo de 4 pessoas.

**Nível 2 Compreensão:** Na unidade subalgoritmos e passagem de parâmetros, os estudantes irão identificar todos parâmetros e requisitos de para correta escrita de tais rotinas computacionais em um grupo de 4 pessoas.

**Nível 3 Aplicação:** Na unidade subalgoritmos e passagem de parâmetros, os estudantes irão construir programas complexos, construídos sob diversos subalgoritmos, em um grupo de 4 pessoas.

12. Ponteiros

**Nível 1 Conhecimento:** Na unidade ponteiros, os estudantes irão listar todas características que definem por completo um ponteiro em um grupo de 4 pessoas.

**Nível 2 Compreensão:** Na unidade ponteiros, os estudantes irão identificar os resultados de diversas operações utilizando ponteiros em um grupo de 4 pessoas.

**Nível 3 Aplicação:** Na unidade ponteiros, os estudantes irão construir programas complexos utilizando ponteiros em um grupo de 4 pessoas.

### 13. Recursividade

**Nível 1 Conhecimento:** Na unidade recursividade, os estudantes irão denominar o conceito associado, a usabilidade e aplicações de recursividade em um grupo de 4 pessoas.

**Nível 2 Compreensão:** Na unidade recursividade, os estudantes irão reconhecer o resultado final da aplicação da recursividade em diversas situações propostas em um grupo de 4 pessoas.

**Nível 3 Aplicação:** Na unidade recursividade, os estudantes irão construir programas complexos utilizando tal conceito em um grupo de 4 pessoas.

### 14. Registros

**Nível 1 Conhecimento:** Na unidade registros, os estudantes irão definir todos conceitos e aplicações relacionadas a registros em um grupo de 4 pessoas.

**Nível 2 Compreensão:** Na unidade registros, os estudantes irão selecionar formas de registros mais adequadas para cada situação dada em um grupo de 4 pessoas.

**Nível 3 Aplicação:** Na unidade registros, os estudantes irão construir programas complexos com laços de repetição em um grupo de 4 pessoas.

### 15. Arquivos

**Nível 1 Conhecimento:** Na unidade registros, os estudantes irão definir todos conceitos e aplicações relacionadas a registros em um grupo de 4 pessoas.

**Nível 2 Compreensão:** Na unidade registros, os estudantes irão selecionar formas de registros mais adequadas para cada situação dada em um grupo de 4 pessoas.

**Nível 3 Aplicação:** Na unidade registros, os estudantes irão construir programas complexos com laços de repetição em um grupo de 4 pessoas.

### 16. Ordenação e busca

**Nível 1 Conhecimento:** Na unidade ordenação e busca, os estudantes irão enumerar e definir diversas algoritmos de enumeração e busca em um grupo de 4 pessoas.

**Nível 2 Compreensão:** Na unidade ordenação e busca, os estudantes irão descrever todos passos dos algoritmos de ordenação e busca estudados anteriormente em um grupo de 4 pessoas.

**Nível 3 Aplicação:** Na unidade algoritmos com repetição, os estudantes irão construir programas complexos com laços de repetição em um grupo de 4 pessoas.

### 3.1.3 Definição de objetivos educacionais para a Disciplina *Algoritmos e Programação de Computadores* com relação às atividades práticas

Nesta seção, define-se um possível planejamento de módulos de aprendizagem relacionadas a eletrônica básica utilizando a placa Galileo em paralelo à aprendizagem de programação estruturada na linguagem C.

#### 1. Leis básicas de eletrônica (Leis de Kirschhoff)

**Nível 1 Conhecimento:** Na unidade leis básicas de eletrônica( Leis de Kirschhoff), os estudantes irão enumerá-las todas e com dados exemplos e situações, escreve-lás propriamente em um grupo de 4 pessoas.

**Nível 2 Compreensão:** Na unidade leis básicas de eletrônica( Leis de Kirschhoff), os estudantes irão reconhecer todos parâmetros das leis de Kirschhoff, nos circuitos elétricos exemplificados, em um grupo de 4 pessoas.

**Nível 3 Aplicação:** Na unidade leis básicas de eletrônica( Leis de Kirschhoff), os estudantes irão empregar diferentes dispositivos eletrônicos em circuitos desenvolvidos em simulações e observar as relações das lesi de Kirschhoff estudadas anteriormente em um grupo de 4 pessoas.

#### 2. Sistema da placa Galileo

**Nível 1 Conhecimento:** Na unidade sistemas da placa Galileo, os estudantes irão descrever propriamente as características básicas da placa Galileo grupo de 4 pessoas.

**Nível 2 Compreensão:** Na unidade sistemas da placa Galileo, os estudantes irão explicar a utilidade dos pinos analógicos e digitais e suas relações com o programa carregado na placa em um grupo de 4 pessoas.

**Nível 3 Aplicação:** Na unidade sistemas da placa Galileo, os estudantes irão, sabendo da estrutura da placa Galileo, criar programas utilizando a linguagem de programação C em um grupo de 4 pessoas.

#### 3. Fundamentos de programação em Galileo

**Nível 1 Conhecimento:** Na unidade fundamentos de programação em Galileo, os estudantes irão repetir a estrutura básica de programas Arduino criando simples projetos como *Blink* LED em um grupo de 4 pessoas.

**Nível 2 Compreensão:** Na unidade fundamentos de programação em Galileo, os estudantes serão capazes de, utilizando a estrutura básica de um programa em Arduino, desenvolver seus próprios algoritmos, para resolução de uma situação problema dada em um grupo de 4 pessoas.

**Nível 3 Aplicação:** Na unidade fundamentos de programação em Galileo, os estudantes serão capazes de unir conceitos de programação básica com os conceitos de eletrônica de forma desenvolver soluções eficazes e eficientes em um grupo de 4 pessoas.

#### 4. Introdução a sensores

**Nível 1 Conhecimento:** Na unidade introdução a sensores, os estudantes definir e expor o que são sensores em um grupo de 4 pessoas

**Nível 2 Compreensão:** Na unidade introdução a sensores, os estudantes selecionar sensores apropriados como partes de circuitos complexos de forma a solucionar um problema dado em um grupo de 4 pessoas.

**Nível 3 Aplicação:** Na unidade introdução a sensores, os estudantes desenvolver projetos com a placa Galileo e sensores de forma a resolver situações problemas dadas de forma eficaz em um grupo de 4 pessoas.

#### 5. Uso de Circuitos Integrados

**Nível 1 Conhecimento:** Na unidade uso de circuitos integrados, os estudantes serão capazes de definir o que são circuitos integrados e expor sua importância para sistemas computacionais em um grupo de 4 pessoas.

**Nível 2 Compreensão:** Na unidade uso de circuitos integrados, os estudantes serão capazes de, consultando *Datsheets* e tutoriais diversos, identificar os pinos de tais circuitos e seus respectivos usos para o funcionamento de tais circuitos integrados em um grupo de 4 pessoas.

**Nível 3 Aplicação:** Na unidade uso de circuitos integrados, os estudantes serão capazes de criar projetos complexos com a placa Galileo utilizando circuitos integrados.

### 3.1.4 Proposta de uso de placas de prototipagem eletrônica

O uso da metodologia de aprendizagem por masterização, seja ela pelo viés do uso da metodologia clássica de ensino, pelo uso da metodologia de Aprendizagem Baseada em Problemas ou pelo viés da Aprendizagem Baseada em Projetos citadas na Seção 2.1.1, tem sido feito, progressivamente mais, usando placas de prototipagem eletrônica.

No caso citado em [33], os alunos estavam em início de curso e passaram por uma revisão de modelo de curso. Nessa revisão, foi incluído o uso da placa Arduino para aprendizagem de programação básica. Os alunos que estudaram sob o novo formato, comparados a turmas anteriores, tiveram níveis mais altos de aprendizagem e retenção do conhecimento.

Já em [34], foi proposto, para alunos de ensino médio, um projeto de ensino de robótica baseado em aprendizagem ativa pelo viés da Aprendizagem Baseada em Projetos. Nesse caso citado, foram utilizadas placas Arduino, por causa da sua facilidade de programação e prototipação, o que a tem feito progressivamente mais popular.

Assim como em [33] e [34], as práticas proposta melhoram consideravelmente os índices acadêmicos dos estudantes, assim como sua motivação a aprender e retenção do conhecimento exposto.

No caso desta proposta para a disciplina *Algoritmos e Programação de Computadores*, escolheu-se o uso da Placa Intel<sup>®</sup> Galileo. A placa Intel<sup>®</sup> Galileo, além de possuir todas funcionalidades da placa Arduino supracitada, tem funcionalidades mais diversas as quais permitem a ampliação dos conteúdos tratados não só para cursos de programação básica, mas para as mais diversas disciplinas de computação.

### 3.1.5 Forma de avaliação

Como a metodologia de ensino proposta é distinta da metodologia clássica, centrada no professor, deve ser também distinta a forma de avaliação dos alunos.

Levando em consideração as teorias expostas nas Seções 2.1.2, 2.1.1 e 2.2 deve ser parte majoritária das avaliações os seguintes aspectos:

- Masterização dos conteúdos propostos a cada prática proposta;
- Masterização de habilidades de trabalho em grupo por meio de:
  - Auto-avaliação
  - Peer-review (Avaliação dos outros integrantes do grupo);
- Quantidade e Qualidade dos *Entregáveis* de cada Sprint;
- Qualidade dos trabalhos solicitados, não apenas com relação ao produto final, mas com relação também a:
  - Apresentação
  - Documentação apresentada;

Além disso, é totalmente de acordo com o proposto na teoria de masterização se houver alguma forma de avaliação, pontuação e premiação aos alunos mais engajados em processos de aprendizagem colaborativa.

As percentagens de cada dos critérios sugerido na nota final dos alunos deve ser decidida a critério do professor, o qual está agindo no papel de orientador e *Scrum Master*.

## Capítulo 4

# Elementos de Circuitos e Programação

### 4.1 Introdução

Este capítulo é destinado a explicação detalhada dos conceitos teóricos que embasam as práticas propostas no Capítulo 5 deste trabalho.

### 4.2 Circuitos eletrônicos

Nesta seção são tratados todos conceitos relativos a circuitos eletrônicos e hardwares que embasam as práticas propostas na Seção 5.

#### 4.2.1 Resistor

Num circuito elétrico, chama-se por resistor o elemento que oferece *resistência* a passagem de corrente elétrica entre seus terminais. Resistores são elementos *passivos* num circuito eletrônico. Isso significa que eles são apenas consumidores de energia.

Com o uso de resistores e suas ligações (Subseções 4.2.2.4 e 4.2.2.5), é possível controlar as correntes e tensões num circuito de forma a : limitar seus valores, ajustar nível de sinais, polarizar elementos ativos, entre outras diversas aplicações. Tais funcionalidades do resistor estão sempre associadas à *Lei de Ohm*, descrita com mais detalhes na seção 4.2.2.3. A Figura 4.1 mostra um exemplo de resistor utilizado em circuitos eletrônicos.

Há vários tipos de resistores fabricados de diversas maneiras e, como dito, para as mais diversas aplicações. O resistor mostrado na Figura 4.1 é chamado de resistor de valor fixo e geralmente é utilizado para prototipagem de circuitos eletrônicos. Resistores de valor fixo são geralmente fabricados utilizando carbono, metal, ou películas de óxidos metálicos.

A simbologia para resistores num circuito elétrico é também extensa. A Figura 4.2 mostra todos os tipos de simbologia utilizado para várias aplicações.



Figura 4.1: Exemplo de resistor utilizado em circuitos eletrônicos.

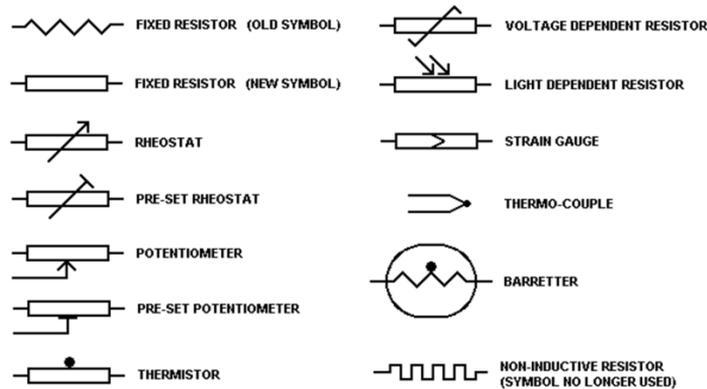


Figura 4.2: Símbolos de resistor utilizado em circuitos eletrônicos.

Fonte: [www.vegyelgep.bme.hu](http://www.vegyelgep.bme.hu)

Resistores de valor fixo, como o mostrado na Figura 4.3, tem como resistência a seguinte expressão:

$$R = (Digito_1 + 10 * Digito_2) * Multiplicador \pm Tolerancia\% \quad (4.1)$$

Os valores  $Digito_1$ ,  $Digito_2$ ,  $Multiplicador$  e  $Tolerância$  são mostrados na Tabela 4.1 para cada cor utilizada no código.

Cada resistor tem um limite que corrente que pode circular entre seus terminais. Geralmente esse limite é dado em relação à potência dissipada. Para saber qual potência será dissipada num resistor com resistência de valor  $R$  Ohms deve-se aplicar a seguinte fórmula:

$$P = IU \text{ (Potência dissipada num resistor de valor R)} \quad (4.2)$$

Onde  $P$  é a potência dissipada em *Watts*,  $I$  é a corrente elétrica que passa no resistor em *Ampéres* e  $U$  é a tensão elétrica entre os terminais do resistor em *Volts*.

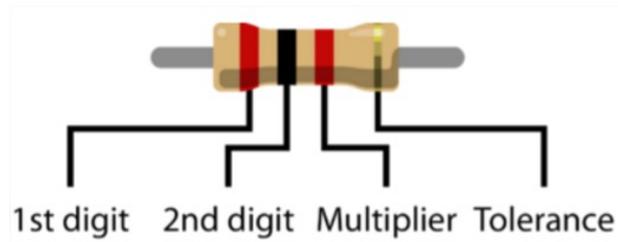


Figura 4.3: Como ler o código de cores de um resistor.

Fonte: <http://education.rec.ri.cmu.edu/content/electronics/common/resistors/1.html>

Tabela 4.1: Tabela de leitura de valor e tolerância de um resistor

Cor	Multiplicador	Dígito 1	Dígito 2	Tolerância
Preto	$10^0$	0	0	
Marrom	$10^1$	1	1	$\pm 1\%$
Vermelho	$10^2$	2	2	$\pm 2\%$
Laranja	$10^3$	3	3	
Amarelo	$10^4$	4	4	
Verde	$10^5$	5	5	
Azul	$10^6$	6	6	
Violeta	$10^7$	7	7	
Cinza	$10^8$	8	8	
Branco	$10^9$	9	9	
Ouro				$\pm 5\%$
Prata				$\pm 10\%$

#### 4.2.2 Circuito - fórmulas e topologias básicas

Nomea-se por circuito elétrico, um caminho fechado entre dois ou mais pontos formado por componentes eletrônicos no qual corrente elétrica pode circular. A Figura 4.4 mostra um circuito elétrico simples formado por uma bateria que fornece uma diferença de potencial (Tensão) entre seus terminais de valor  $V$  Volts e um resistor com valor de resistência  $R$  Ohms.

Os pontos A e B da figura mostram, respectivamente, o ponto de maior potencial elétrico (+ da bateria) e o ponto de menor potencial elétrico (- da bateria).

Para descrever o comportamento a dinâmica de tensões, correntes existem as seguintes leis:

- Primeira lei de Kirchoff - Leis das Malhas;
- Segunda lei de Kirchoff - Leis dos Nós;
- Lei de Ohm.

E para descrever a topologia dos circuitos, existem duas ligações básicas entre componentes a

serem tratadas neste documento:

- Ligação Série;
- Ligação Paralelo.

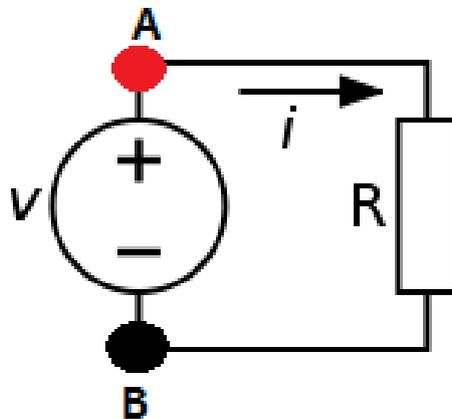


Figura 4.4: Circuito Simples.

Fonte: Adaptado de [www.sparknotes.com](http://www.sparknotes.com)

#### 4.2.2.1 Primeira lei de Kirchoff - Leis das Nós

A primeira lei de Kirchoff diz que a soma das correntes elétricas que entram num nó  $I_{entram}$  é igual a soma das correntes que saem,  $I_{saem}$ , ou seja:

$$\sum I_{entram} = \sum I_{saem} \quad (4.3)$$

Usando a Figura 4.5 como exemplo, a primeira, nesse caso, é escrita da seguinte forma:

$$I_1 + I_2 = I_3 + I_4 + I_5 \quad (4.4)$$

#### 4.2.2.2 Segunda lei de Kirchoff - Leis das Malhas

A segunda lei de Kirchoff diz que a soma das tensões  $u_i$  num caminho fechado, num circuito elétrico é igual a zero, ou seja:

$$\sum u_i = 0 \quad (4.5)$$

Usando a Figura 4.6 como exemplo, onde as tensões em cada componentes são descritas pela letra  $u$ , a primeira lei Kirchoff é escrita na seguinte forma

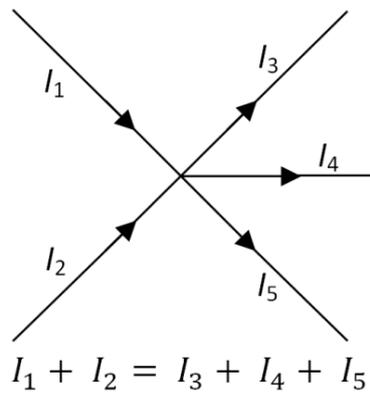


Figura 4.5: Primeira Lei de Kirchoff.

Fonte: [www.crkautomacao.com.br](http://www.crkautomacao.com.br)

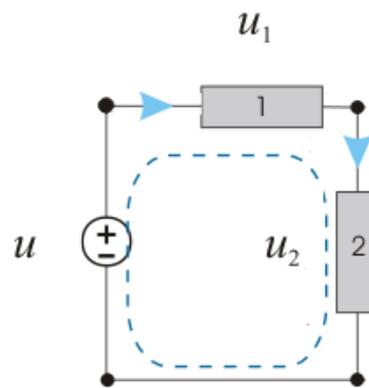


Figura 4.6: Segunda Lei Kirchoff.

Fonte: [www.crkautomacao.com.br](http://www.crkautomacao.com.br)

$$\begin{aligned}
 u - u_1 - u_2 &= 0 \\
 u &= u_1 + u_2
 \end{aligned}
 \tag{4.6}$$

#### 4.2.2.3 Lei de Ohm

A lei de Ohm[25] diz que num circuito fechado, sob um componente eletrônico que com valor resistência  $R$  que esteja submetido a uma diferença de potencial entre seus terminais de valor  $V$ , circulará uma corrente elétrica de  $I$  segundo a seguinte fórmula:

$$I = \frac{V}{R} \tag{4.7}$$

Essa expressão não depende da natureza de tal condutor: ela é válida para todos os condutores. Para um dispositivo condutor que obedeça à lei de Ohm, a diferença de potencial aplicada é proporcional à corrente elétrica, isto é, a resistência é independente da diferença de potencial e

da corrente. Um dispositivo muito utilizado em aparelhos eletrônicos, como rádios, televisores e amplificadores, que obedece à essa lei é o resistor (componente descrito mais detalhadamente na Seção 4.2.1) cuja função é controlar a intensidade de corrente elétrica que passa pelo aparelho.

Entretanto, para alguns materiais, por exemplo os semicondutores, a resistência elétrica não é constante, mesmo que a temperatura seja, ela depende da diferença de potencial  $V$ . Estes são denominados condutores não ôhmicos. Um exemplo de componente eletrônico que não obedece à lei de Ohm é o diodo (componente descrito mais detalhadamente na Seção 4.2.3).

#### 4.2.2.4 Ligação em Série

Quando dois ou mais componentes estão conectados num circuito, um após ao outro, diz que tais componentes estão conectados em série, como mostrado na Figura 4.7.

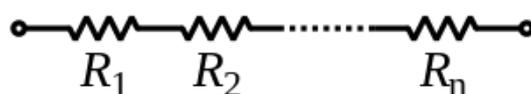


Figura 4.7: Ligação em série de n resistores.

Fonte: [www.crkautomacao.com.br](http://www.crkautomacao.com.br)

Para elementos que se encontram em série, a mesma corrente  $I$  os percorre, como mostrado na Figura 4.8.

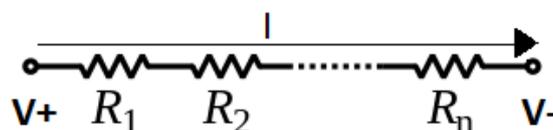


Figura 4.8: Mesma corrente percorrendo resistores em série.

Fonte: [www.crkautomacao.com.br](http://www.crkautomacao.com.br)

Aplicando a segunda lei de Kirchoff, apresentada na Seção 4.2.2.2, e a lei de Ohm, apresentada na Seção 4.2.2.3, tem-se a seguinte sequência de expressões para a demonstração da resistência equivalente de resistores em série:

$$V_+ - V_- = V_{R1} + V_{R2} + \dots V_{Rn} \quad (\text{Aplicando a lei de Kirchoff das malhas}) \quad (4.8)$$

$$I * R_{eq} = I * (R1 + R2 + \dots Rn) \quad (\text{Aplicando a Lei de Ohm}) \quad (4.9)$$

$$R_{eq} = (R_1 + R_2 + \dots + R_n) \quad (\text{Fórmula de resistor equivalente - ligação série}) \quad (4.10)$$

A resistência equivalente de uma ligação em série de resistores pode ser vista como a "resistência enxergada" pela fonte de diferença de potencial entre os pontos V+ e V- mostrados na Figura 4.8.

#### 4.2.2.5 Ligação em Paralelo

Numa conexão em paralelo, os componentes se encontram conectados com seus terminais conectados em comum e possuem sobre eles a mesma tensão elétrica, como mostrado na Figura 4.9.

A primeira lei de Kirchoff, como mostrado na Seção 4.2.2.1, mostra que a soma das correntes que entram num nó é igual a soma das correntes que por ele saem.

Na Figura 4.9, os resistores  $R_1, R_2, \dots, R_n$  encontram em paralelo. A soma das correntes que entram no nó A é igual a soma das correntes que dele saem, ou seja:

$$I = I_1 + I_2 + \dots + I_n \quad (\text{Primeira Lei de Kirchoff}) \quad (4.11)$$

Utilizando a Lei de Ohm e sabendo que, numa ligação em paralelo, todos elementos encontram-se sob a mesma Tensão elétrica, tem-se que:

$$\frac{V_{in}}{R_{eq}} = \frac{V_{in}}{R_1} + \frac{V_{in}}{R_2} + \dots + \frac{V_{in}}{R_n} \quad (\text{Lei de Ohm aplicada à ligação em paralelo}) \quad (4.12)$$

Disso, tem-se que:

$$\frac{1}{R_{eq}} = \frac{1}{R_1} + \frac{1}{R_2} + \dots + \frac{1}{R_n} \quad (\text{Fórmula de resistor equivalente - ligação paralelo}) \quad (4.13)$$

#### 4.2.2.6 Resumo - Ligação em série X paralelo

A Tabela 4.2 resume as características da ligação em série e da ligação em paralelo entre resistores:

### 4.2.3 Diodo-LED

Um **Diodo Emissor de Luz** (LED) é um dispositivo semicondutor o qual, ao passar de corrente elétrica do seu *Ánodo* para seu *Cátodo* emite luz. A Figura 4.11 mostra o símbolo de um LED.

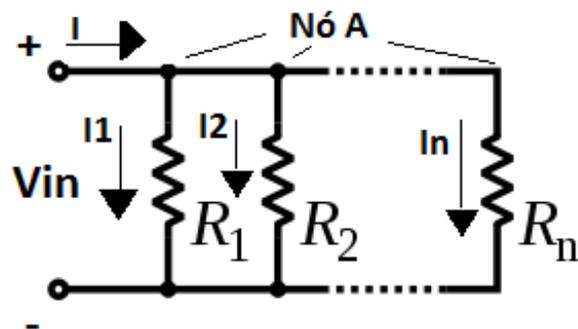


Figura 4.9: Componentes conectados em paralelo.

Tabela 4.2: Resumo - Ligação em série X Ligação em paralelo

	Tensão elétrica nos elementos	Corrente elétrica pelos elementos	Resistência Equivalente entre N elementos conectados
<b>Ligação em Série</b>	Diferente	Igual	$R_{eq} = R_1 + R_2 + \dots + R_n$
<b>Ligação em Paralelo</b>	Igual	Diferente	$\frac{1}{R_{eq}} = \frac{1}{R_1} + \frac{1}{R_2} + \dots + \frac{1}{R_n}$

Como todo diodo, o LED bloqueia a passagem de corrente elétrica na direção do seu cátodo ao seu ânodo e permite a passagem de corrente na direção contrária.

Como mostrado na Figura 4.10, diodo são construídos com duas junções de semicondutores dopados: uma junção N e uma junção P. A junção P possui falta de portadores negativos, ou seja, está positivamente dopado. A junção N possui excesso de portadores negativos, ou seja, está negativamente dopada.

A dinâmica dos portadores entre as junções N - P criem o comportamento de bloqueio e liberação da passagem de corrente no diodo.

Para um diodo com tensão positiva entre ânodo e cátodo, existe uma região chamada região de depleção a qual deve ser superada para que exista fluxo de corrente, ainda que o diodo esteja positivamente polarizado, como mostrado também no gráfico da Figura 4.12.

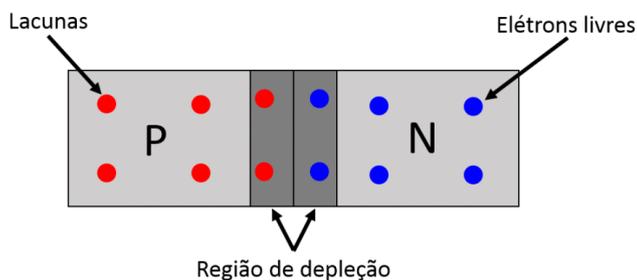


Figura 4.10: Junção N-P em um diodo.

Fonte: [www.electrotechservices.com](http://www.electrotechservices.com)

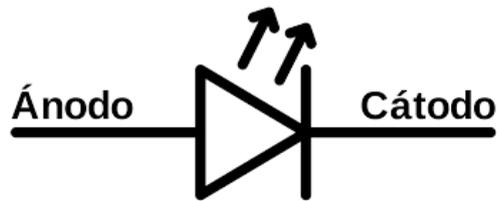


Figura 4.11: Símbolo Diodo

Fonte: [www.electrotechservices.com](http://www.electrotechservices.com)

A Figura 4.12 mostra a relação entre tensão aplicada entre o ânodo e o cátodo e a corrente que atravessa um diodo. Quando a diferença de tensão entre ânodo e cátodo é positiva, o diodo se encontra sob polarização direta. Nessa polarização, o diodo permite passagem de corrente entre seus terminais. A fórmula que expressa essa relação é a seguinte:

$$i_D = I_s(e^{v_d/nkT} - 1) \quad (4.14)$$

Onde  $i_D$  é a corrente se passar pelo diodo,  $v_D$  é a tensão entre ânodo e cátodo,  $T$  é a temperatura,  $k$  é a constante de Boltzmann,  $N$  é o coeficiente de não linearidade e  $I_s$  é a corrente de saturação.

Como mostrado na Figura 4.12, quando em polarização reversa, o diodo não permite passagem de corrente com intensidade apreciável.

Caso a diferença de tensão entre cátodo e ânodo seja maior que a tensão de quebra  $V_{br}$ , o diodo entrará na região de quebra e não mais bloqueará a passagem de corrente.

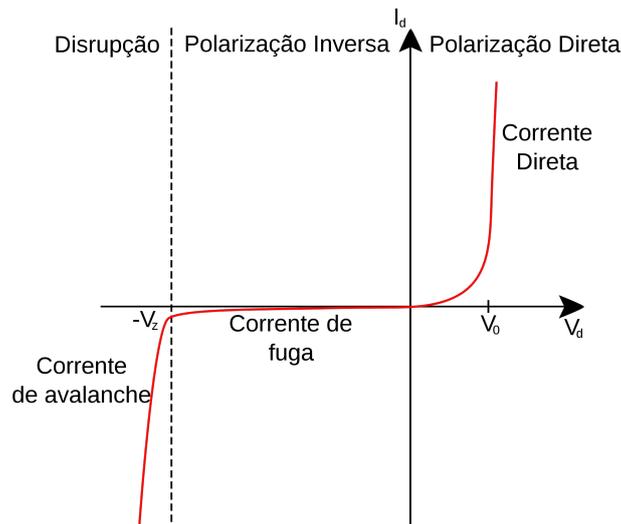


Figura 4.12: Relação tensão, corrente num diodo e suas regiões de operação

Fonte: [http://www2.feg.unesp.br/Home/PaginasPessoais/ProfMarceloWendling/2 --- diodo - semicondutor.pdf](http://www2.feg.unesp.br/Home/PaginasPessoais/ProfMarceloWendling/2---diodo-semicondutor.pdf)

Para aplicações simples em eletrônica, usualmente se adota um modelo simplificado para a operação do diodo mostrado na Figura 4.13.

Nesse modelo, para uma polarização direta, o diodo é substituído por um diodo ideal (sem queda de tensão) com uma bateria em série com valor de tensão de 0.7 a 0.6 de tensão significando a queda de tensão entre os terminais do diodo.

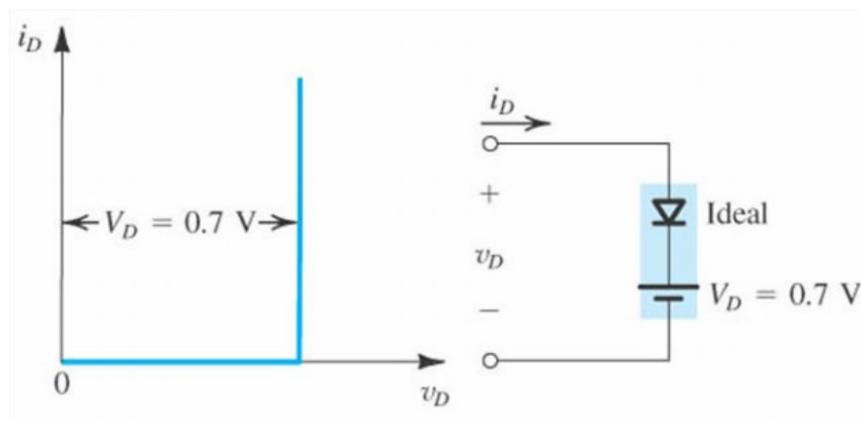


Figura 4.13: Modelo simplificado de um diodo em polarização direta.

Fonte: <https://digitalsignal.files.wordpress.com/2008/09/aula05.pdf>

Diodo são utilizados em várias aplicações em eletrônica. Alguns exemplo de aplicações seriam:

- Retificação de tensão (Transformação de um sinal AC (Corrente Alternada) para DC (Corrente Contínua));
- Isolamento de sinais de fonte de potência;
- Referência de voltagem;
- Controle e limitação da amplitude de sinais;
- Detecção de sinais.

#### 4.2.4 Protoboard

Protoboard ou *BreadBoard* é um componente essencial na prototipação de circuitos eletrônicos. Com uma protoboard, é possível criar circuitos temporários, sem necessidade de realizar soldagem.

As protoboards foram criadas para simplificar a prototipação e testes de circuitos eletrônicos anteriormente a sua efetiva produção por meio em máquinas de circuitos impress. Em geral, as protoboards apresentam a estrutura mostrada na Figura 4.14. As trilhas verticais + e - marcadas na Figura são trilhas contínuas de alumínio. Usualmente, mas não obrigatoriamente, as trilhas + e - são usadas como para os polos positivo e negativo da bateria. A região central, com as trilhas na horizontal, geralmente é utilizada para a construção do circuito desejado.

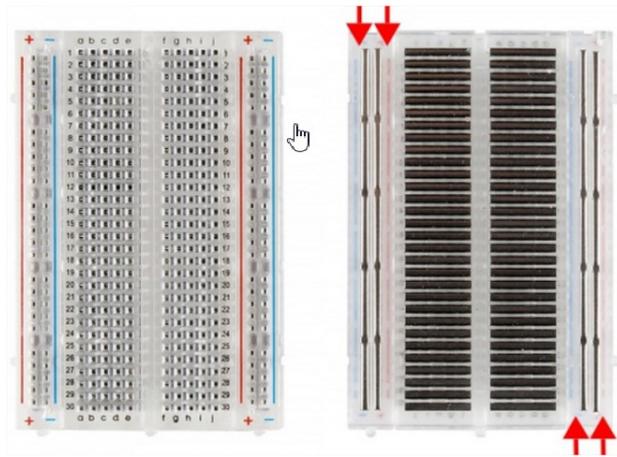


Figura 4.14: Protoboard.

Fonte: <https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard>

A Figura 4.15 mostra o esquemático de circuito simples formado por uma fonte de 9V, um resistor de 1k ohm e um LED e sua construção numa protoboard.

Na parte direita da Figura 4.15, é mostrado os polos positivo e negativos da bateria de 9V conectado à trilha contínua e a região central foi usada para conectar tais polos ao LED e resistor em série. Além disso, a parte direita da Figura mostra o sentido convencional da corrente elétrica.

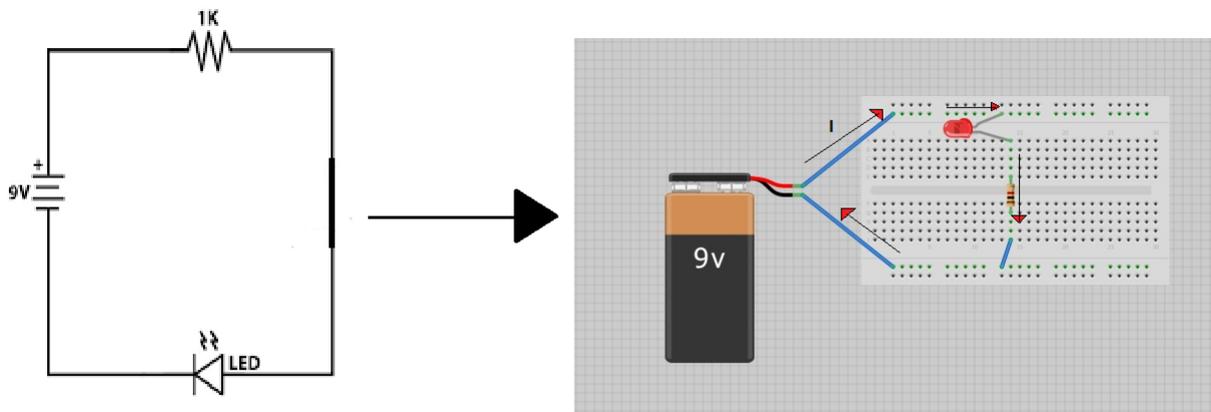


Figura 4.15: Esquemático de um circuito e sua construção numa *protoboard*.

Fonte: <https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard>

#### 4.2.5 Divisor de tensão

Em eletrônica, chama-se divisor de tensão um circuito com estrutura similar ao mostrada na Figura 4.16.

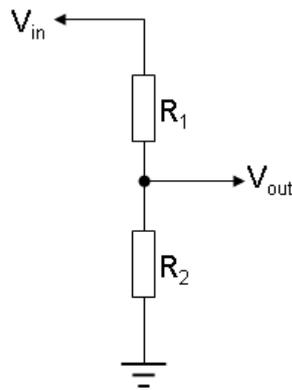


Figura 4.16: Modelo de um circuito divisor de tensão

Pela segunda lei de Kirchoff (seção 4.2.2.2) tem-se a seguinte expressão:

$$V_{in} - V_{R1} - V_{R2} = 0 \quad (4.15)$$

A corrente que circulará nesse circuito é igual a:

$$I = \frac{V_{in}}{(R1 + R2)} \quad (4.16)$$

Calculando o valor da tensão de saída, tem-se:

$$V_{out} = V_{in} - I * R1 \quad (4.17)$$

Portanto, a tensão de saída será igual a:

$$V_{out} = \frac{R2}{R1 + R2} * V_{in} \quad (4.18)$$

O fator  $\frac{R2}{R1+R2}$  é o fator da divisão de tensão, no circuito exemplificado pela Figura 4.16.

Divisores de tensão tem várias aplicações em eletrônica, dentre elas, destacam-se as seguintes:

- **Medição de sensores:** Um divisor de tensão pode ser usado como forma de medição para sensores resistivos ( Sensores que alteram sua resistência de acordo com o fator de sensibilidade).
- **Medição de altas tensões:** Um divisor de tensão pode ser usado para utilizar voltímetros para tensões além de sua respectivas escala de medição.
- **Mudança de escala de sinais:** Um divisor de tensão pode ser usado para adequar a escala de um sinal para sinais que podem ser lidos e processados por um microprocessador em suas faixa operação própria.

## 4.2.6 Potenciômetro

Um potenciômetro é um componente de circuitos elétricos que possui resistência elétrica ajustável mecanicamente. Num esquemático de um circuito, o símbolo usado para potenciômetro é mostrado na Figura 4.17.



Figura 4.17: Símbolo de um potenciômetro.

Potenciômetros são muito usados para controlar / alterar as características de entrada / saída de aparelhos eletrônicos, como volume, balanço, graves, brilho, contraste, cor, tempo de funcionamento (em tv's, dvd's, monitores, relógios, ... ). São também conhecidos como resistores variáveis, ou ainda, reostatos.

Para criar tal característica de variação de resistência, os potenciômetros possuem internamente uma trilha resistiva (de níquel-cromo ou de carbono), sobre a qual desliza um cursor , que altera a resistência elétrica entre seu conector central e um dos dois laterais (normalmente são três conectores).

Os potenciômetros podem ser classificados em duas categorias: quanto a forma de deslocamento do curso e quanto a forma de variação da resistências entre seus terminais.

Em relação a forma de deslizamento do curso, os potenciômetros podem ser angulares (Figura 4.18) e lineares (Figura 4.19).

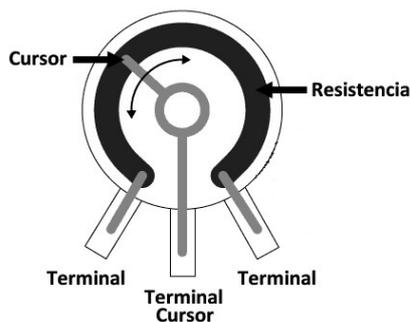


Figura 4.18: Estrutura de um potenciômetro angular.

Fonte: [baudaeletronica.com](http://baudaeletronica.com)



Figura 4.19: Potenciômetro linear.

Fonte: baudaeletronica.com

Em relação a a forma de variação da resistência entre seus terminais, os potenciômetros podem ter variação linear ou variação logarítmica. A Figura 4.20 mostra tais formas de variação.

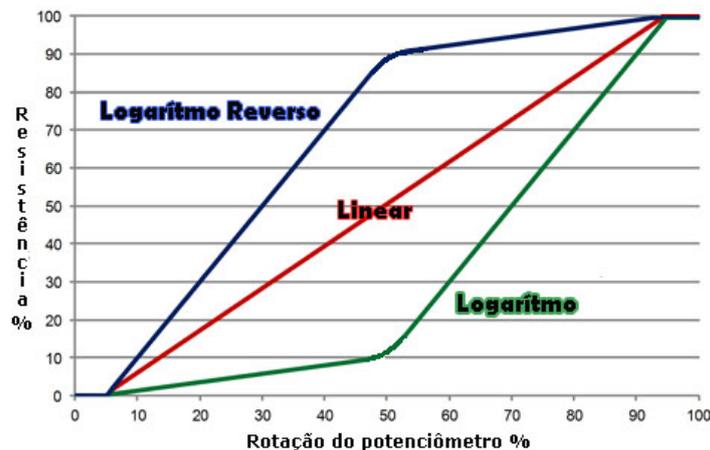


Figura 4.20: Formas de variação da resistência em um potenciômetro.

Fonte: baudaeletronica.com

Assumindo que a resistência entre os terminais externos do potenciômetro da Figura 4.21 é fixa e igual a 10 k ohms tem-se as seguintes formas de uso dos terminais:

- **Potenciômetro 1:** está com os terminais 1 e 2 ligados, neste caso ele varia sua resistência entre 0 ohm e 10 k ohms, nessa ligação quando o eixo é girado para a esquerda ele diminui a sua resistência e quando ele é girado para a direita aumenta a sua resistência.
- **Potenciômetro 2:** Nesse caso, os terminais 2 e 3 encontram-se ligados, ele varia sua resistência entre 0 ohm e 10 k ohms, nessa ligação quando o eixo é girado para a esquerda ele

aumenta a sua resistência e quando é girado para a direita diminui a sua resistência.

- **Potenciômetro 3:** a resistência é fixa, no caso 10 k ohms. Girar o curso não alterará a resistência observada.

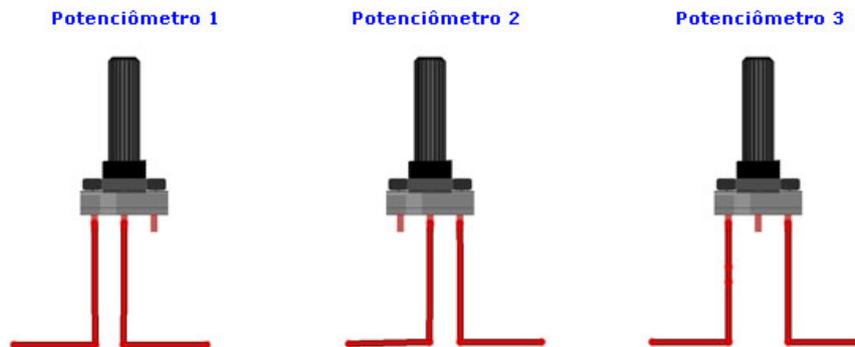


Figura 4.21: Formas de uso de um potenciômetro angular.

Fonte: [www.arduex.com](http://www.arduex.com)

Um exemplo de utilização de potenciômetros são os *joysticks* utilizados em jogos eletrônicos. A Figura 4.22 mostra um circuito integrado de um joystick.

O joystick mostrado é implementado utilizando dois potenciômetros. Um dos potenciômetros é traduz em tensão a posição X em relação à posição inicial e o outro traduz a posição Y.



Figura 4.22: Circuito integrado de um joystick que utiliza dois potenciômetros para tradução de sua posição em tensão.

Fonte: [www.potentiometers.com](http://www.potentiometers.com)

A Figura 4.23 mostra um circuito esquemático utilizando um joystick construído com 2 potenciômetros.

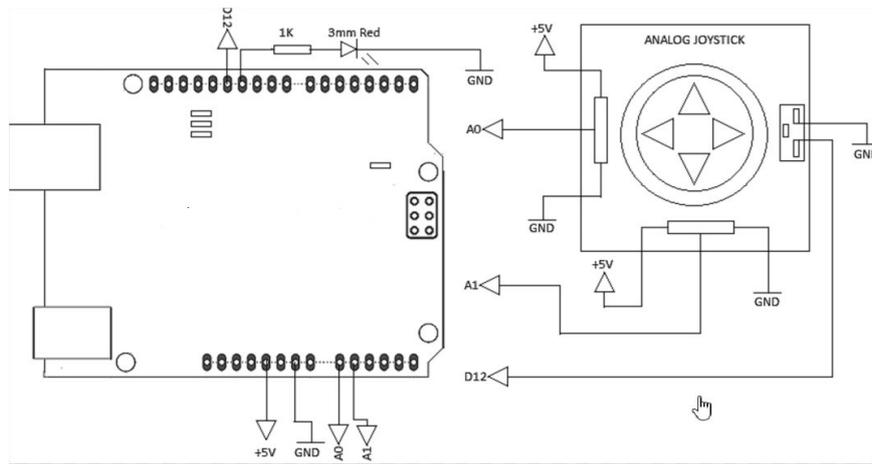


Figura 4.23: Circuito esquemático de um circuito utilizando um joystick similar ao mostrado na Figura 4.22 com a placa Galileo.

Para o circuito mostrado na Figura 4.23, os potenciômetros estão sob 5V de tensão. Na posição inicial, ambas portas analógicas 0 e 1, que leem as posições X e Y, lerão 2.5 V. Ao se movimentar o joystick, as portas analógicas lerão algum valor entre 0 e 5V.

#### 4.2.7 LDR

Um foto-resistor ou LDR (Light Dependent Resistor) é um resistor cuja resistência é dependente da luz que incide sobre ele. A Figura 4.24 mostra um LDR simples, muito utilizado em circuitos eletrônicos e a Figura 4.25 mostra a símbolo para o LDR usado em esquemáticos.

Resistores dependentes de luz podem ser de diferentes tipos. Eles variam em material sensível à luz usada. Um resistor dependente da luz no espectro visível é feita através de sulfureto de cádmio (CdS) ou seleneto de cádmio (CdSe). Este material é sensível ao comprimento de onda de 400 nm - 850 nm. Por perto faixa do infravermelho (1  $\mu\text{m}$  - 3  $\mu\text{m}$ ), existem PBS ou materiais PBSE usado.

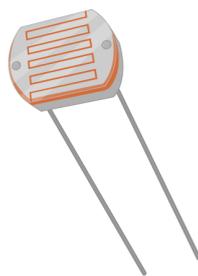


Figura 4.24: LDR

Fonte: [www.electrical4u.com](http://www.electrical4u.com)



Figura 4.25: Símbolo do LDR

Fonte: [www.electrical4u.com](http://www.electrical4u.com)

Um LDR pode variar de dezenas de Megaohms (Escuro) para algumas centenas de ohms (claro) dependendo da intensidade da luz incidente, como mostrado no gráfico da Figura 4.26.

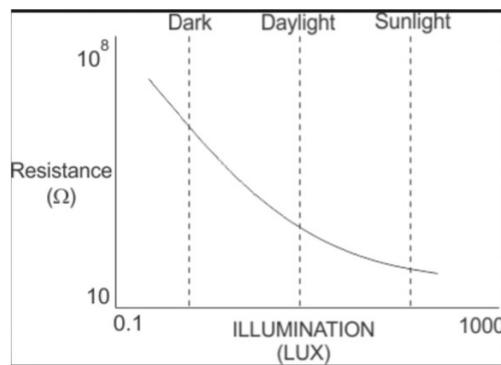


Figura 4.26: Gráfico LDR : Resistência X Luminância

Fonte: [www.electrical4u.com](http://www.electrical4u.com)

Como dito na seção 4.2.5, um divisor de tensão pode ser usado em conjunto com sensores resistivos como o LDR. A Figura 4.27 mostra um sensor de luz implementado com um circuito formado por uma fonte DC, um resistor fixo de 10k ohms e um LDR.

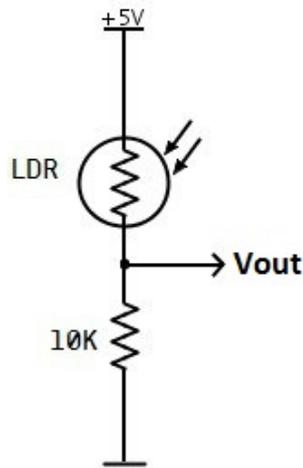


Figura 4.27: Possível circuito a ser utilizado num sensor de luz

Fonte: [www.electrical4u.com](http://www.electrical4u.com)

A tensão de saída  $V_{out}$  é dada segundo a seguinte expressão:

$$V_{out} = \frac{10k}{10k + R_{LDR}} * 5 \quad (4.19)$$

Onde  $R_{LDR}$  obedece ao gráfico mostrado na Figura 4.26.

#### 4.2.8 Interruptores

Ao se desenvolver circuitos eletrônicos, existem duas expressões muito comuns: circuito aberto e circuito fechado. Um circuito aberto é mostrado na Figura 4.28. Nesse circuito, corrente alguma sai da fonte e passa pela lâmpada.

Pode-se dizer que a região entre A e B é uma resistor com resistência que tende ao infinito. Usando essa suposição, ao se calcular a corrente que circula numa região de circuito aberto tem-se a seguinte expressão:

$$I = \lim_{R_{AB} \rightarrow \infty} \frac{V_{bateria}}{R_{lampada} + R_{AB}} = 0A \quad (4.20)$$

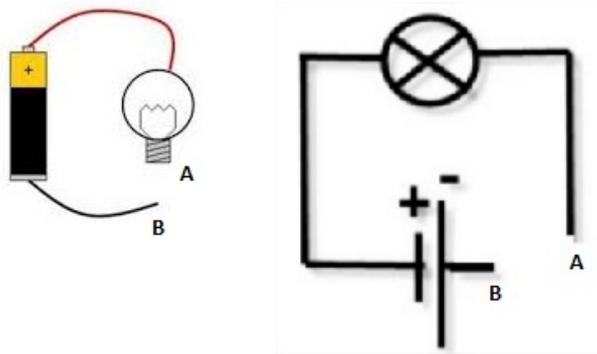


Figura 4.28: Circuito aberto.

Fonte: Adaptado de [www.sciencebuddies.org](http://www.sciencebuddies.org)

Um circuito fechado, é um circuito que se encontra caminho efetivo, para passagem de corrente elétrica. Na Figura 4.29 é mostrado o mesmo circuito da Figura 4.28, porém fechado.

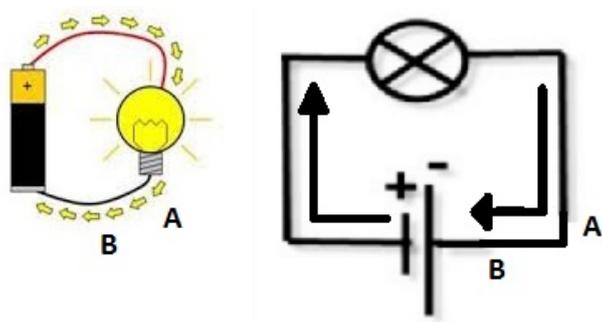


Figura 4.29: Circuito fechado.

Fonte: Adaptado de [www.sciencebuddies.org](http://www.sciencebuddies.org)

Abrir ou fechar uma região de um circuito ou ele por completo pode-ser feito utilizando interruptores ou chaves.

Interruptores ou chaves tem por propósito abrir ou fechar um circuito ou certa região de um circuito de acordo com a vontade de um agente externo. A Figura 4.30 mostra o símbolo de um interruptor num circuito elétrico.

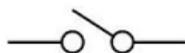


Figura 4.30: Símbolo de um interruptor.

Um tipo especial de interruptor é o botão. A Figura 4.31 mostra a imagem e esquemático de

mini botões.

Os Mini Push Buttons são também chamados de interruptores tendenciosos ou momentâneos, porque após pressionados, eles retornam ao estado de origem (aberto ou fechado).

Existem 2 tipos de Mini Botões de Pressão quanto ao seu estado:

- NO (abreviação de Normally Open), esse interruptor momentâneo fica normalmente aberto (desligado), mas quando pressionado e segurado o botão, o interruptor fecha (liga). Ao soltar o botão, o interruptor abre novamente. Utilizado em teclados de computadores, calculadoras, etc.
- NC (abreviação de Normally Closed), esse interruptor momentâneo fica normalmente fechado (ligado), mas quando apertado e segurado o botão, o interruptor abre (desliga). Ao soltar o botão, o interruptor fecha novamente. Utilizado na iluminação interna das geladeiras, veículos, etc. (ao abrir a porta, o interruptor é acionado, fechando o circuito).

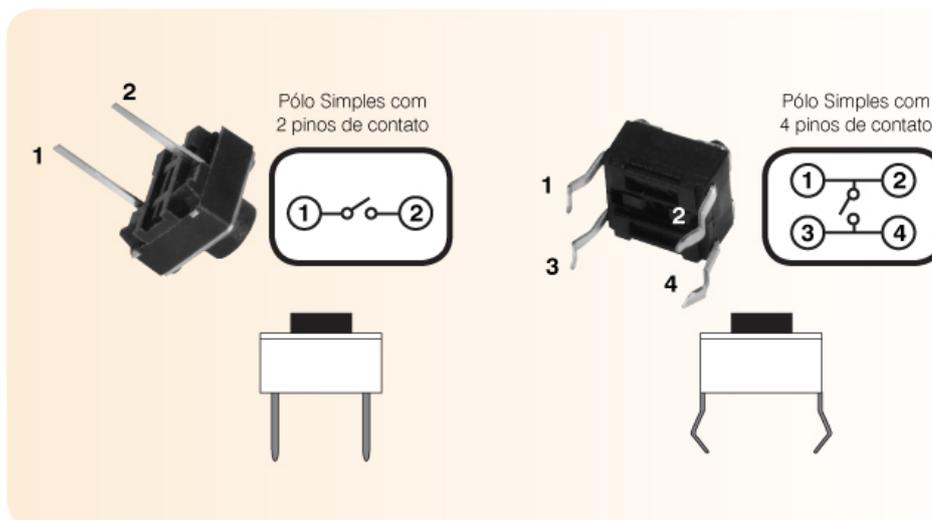


Figura 4.31: Mini botões de pressão, de 2 e 4 pinos.

Fonte: [http : //www.dreaminc.com.br/sala\\_de\\_aula/9b - interruptores - mini - botao - de - pressao/](http://www.dreaminc.com.br/sala_de_aula/9b-interruptores-mini-botao-de-pressao/)

O mini botão de 4 pinos de contato é muitas vezes utilizado para oferecer tensão alta (HIGH) e baixa (LOW) para uma porta de uma placa como a placa Galileo.

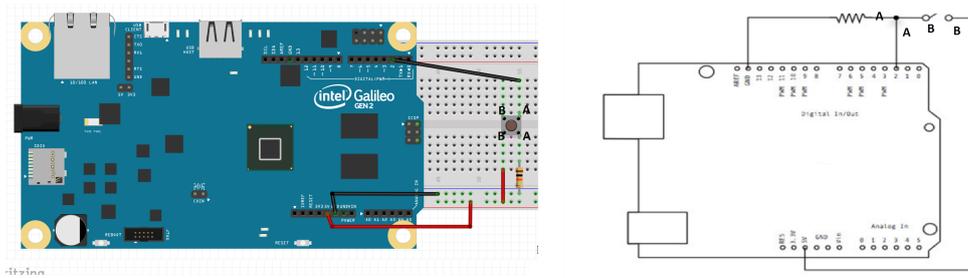


Figura 4.32: Circuito simples utilizando um botão de 4 pinos de contato .

A Figura 4.32 mostra um exemplo de um circuito construído em conjunto com a placa Galileo. Nesse circuito, a porta digital 2 pode ler um nível de tensão alta (HIGH) ou baixa (LOW) dependendo do botão estar pressionado ou não. Como mostrado no esquemático, quando o botão é pressionado, os ponto A e B se encostam e na porta digital passa a existir a tensão alta.

As aplicações para botões em eletrônica são diversas. Neste trabalho, muitas práticas usarão botões no seu desenvolvimento.

#### 4.2.9 Registrador de deslocamento (Shift Register)

Registradores de deslocamento são muito utilizados na conversão entre interfaces seriais para interfaces paralelas. A Figura 4.33 mostra um resumo do funcionamento de um registrador de deslocamento.

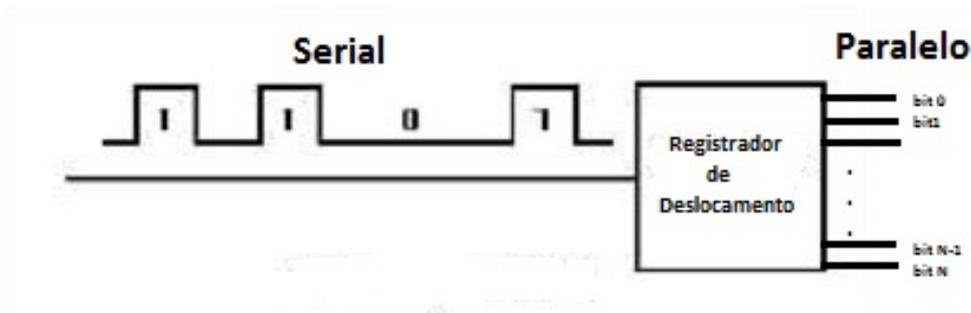


Figura 4.33: Esquema resumido: Série para Paralelo com um registrador de deslocamento.

Fonte: Adaptado de <http://electriciantraining.tpub.com/14185/css/Serial-And-Parallel-Transfers-And-Conversion-Continued-151.htm>

Cada bit que chega no registrador de deslocamento é escrito nas portas paralelas sequencialmente. Tal comportamento é realizado por meio de uma cascata de *flip-flops*. Um flip-flop é um dispositivo que armazena ou reseta um bit de acordo com sinais nas suas portas.

A Figura 4.34 mostra a estrutura interna de um registrador de deslocamento com mais detalhes. Cada vez que um salto alto é posto no pino de Clock, um novo bit que entra no pino *Data in* e os que estavam armazenados nos *flip-flops* são levados para os flip-flops da frente.

Os pinos Q1, Q2, Q3,...,QN são os pinos a serem utilizados para leitura paralela.

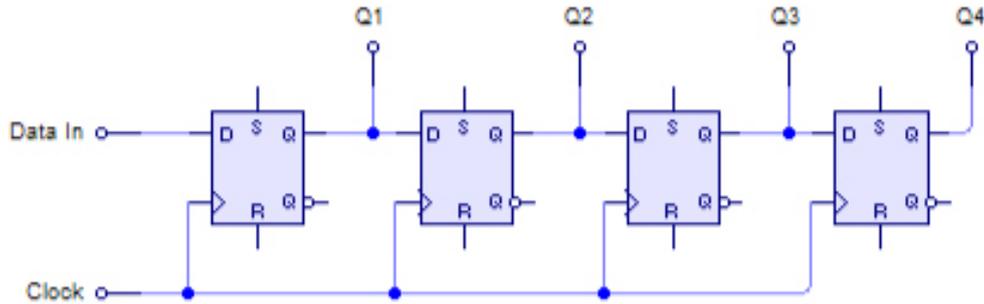


Figura 4.34: Cascata de *flip-flops* num registrador de deslocamento.

Fonte: [wiki.foz.ifpr.edu.br](http://wiki.foz.ifpr.edu.br)

Uma arquitetura similar pode ser usada para realizar a conversão entre comunicação paralela para serial.

Registradores de deslocamento são muito utilizada para controlar diversas entradas e saídas para além das entradas e saídas disponibilizadas por um microcontrolador. Geralmente, os registradores de deslocamento tratam apenas de valores digitais ( $V_{cc}$  ou  $GND$ ) em seus pinos.

Para os propósitos deste trabalho, será usado o registrador de deslocamento 74HC595 [35]. A Figura 4.35 mostra a pinagem desse circuito integrado:

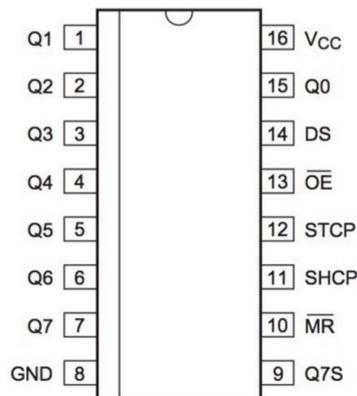


Figura 4.35: Pinagem Registrador de Deslocamento 74HC595.

A descrição dos pinos é a seguinte:

- **Pino 14:** Pino para entrada de dados seriais;
- **Pino 16:** Pino de alimentação VCC;
- **Pino 8:** Pino de referência GND;
- **Pinos 1 a 7 e 15 (Q1 a Q7 e Q0);**

- **Pino 10:** O pino 10 é chamado de *Master Reset (MR)*. Quando um valor baixo de tensão é identificado nesse pino, o registrador de deslocamento é resetado e todos dados que ainda estiverem nele são perdidos;
- **Pino 11:** O pino 11 é destinado para o sinal do clock do sistema, que determinará qual será a frequência com que os bits são deslocados pelo registrador;
- **Pino 12:** O pino 12 é o *Latch Pin*. Quando este pino está no estado de LOW, o registrador de deslocamento está pronto para receber dados na sua pino de entrada. Quando este pino está no estado de HIGH, o registrador de deslocamento é configurado para escrever os dados lidos no pinos Q1 a Q7 e Q0;
- **Pino 13:** O pino 13 é o pino responsável por permitir ou bloquear a transmissão dos dados para as portas de saída. Quando o pino está no estado LOW (GND), a transmissão está permitida, quando está em estado HIGH, a transmissão está bloqueada.

Os bits que entram no pino 14 de forma serial e saem, como dito, sequencialmente nos pinos Q0, Q1, Q2, Q3, Q4, Q5, Q6 e Q7.

#### 4.2.10 Sensor de Temperatura - LM35

Sensores de temperatura são dispositivos os quais possuem alguma propriedade mensurável alterada pela alteração de temperatura.

Sensores são extremamente importantes para sistemas de automação, em qualquer escala ou ambiente.

Em indústrias, onde há processos de produção automatizados, temos muitos tipos de sensores medindo as mais diversas variáveis do processo: temperatura, pressão, peso, pH, dentre muitos outros. Devido a importância da leitura dessas variáveis, existe uma área responsável por instrumentos de medição, a Instrumentação Industrial.

Em ambientes comerciais, sensores de temperatura são muito utilizados para controle de ar-condicionados e verificação de incêndios. Sensores de presença são importantes para segurança.

Com relação a sensores de temperatura, um dos mais usados e baratos é o sensor LM35. O sensor LM35 possui as seguintes características:

- **Entrada temperatura, saída Tensão elétrica:** Alguns sensores de temperatura são sensores resistivos. Se esse fosse o caso, seria necessário a construção de um tipo de divisor de tensão (seção - 4.2.5) para tratar com os dados do sensor na forma de tensão de forma a ter uma variável processável pela placa Galileo.
- **Sem necessidade de calibração da escala Kelvin para Celsius:** Muitas vezes, os sensores possuem respostas calibradas para a escala Kelvin, mas o LM35 já é calibrado para Celsius. A sensibilidade do LM35 é  $10\text{mV}/^{\circ}\text{C}$

- O LM35 consome apenas 60  $\mu\text{A}$  na faixa de 4 a 20 V de alimentação, portanto, pouca influência na leitura ocorre por auto-aquecimento[36].

A Figura 4.36 mostra o LM35 e a Figura 4.37 mostra sua pinagem.

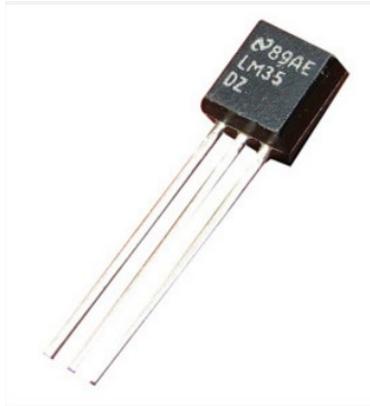


Figura 4.36: Sensor de Temperatura LM35.

Fonte: <http://www.ti.com/lit/ds/symlink/lm35.pdf>

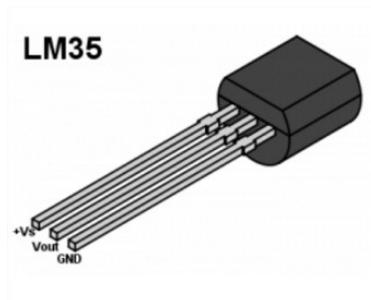


Figura 4.37: Pinagem LM35.

Fonte: <http://www.ti.com/lit/ds/symlink/lm35.pdf>

Na Figura 4.37, os pinos indicam o seguinte:

- **+Vs**: Pino de alimentação. Para aplicações com a placa Galileo, a alimentação 5V já é adequada.
- **GND**: Pino para se conectado ao terra.
- **Vout**: Pino de saída da leitura realizada.

A Figura 4.38 mostra a forma adequada de se conectar o LM35 à placa Galileo.

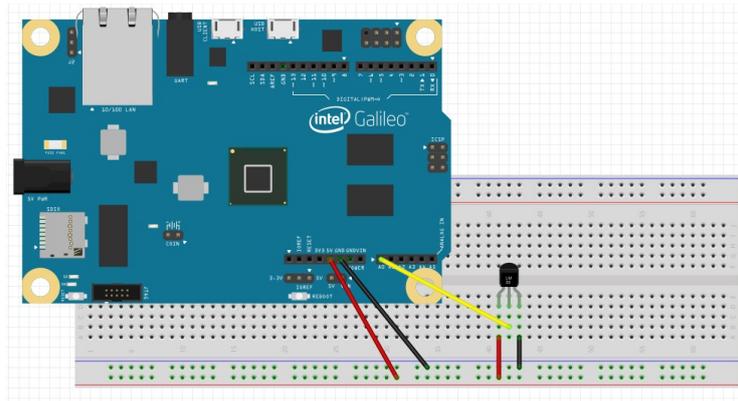


Figura 4.38: Circuito sensor de temperatura feito com LM35 e com a placa Galileo.

Nesse circuito, a temperatura é lida no pino analógico A0.

#### 4.2.11 Capacitores

Capacitores, ao contrário dos resistores que apenas dissipam energia, são dispositivos ativos num circuito elétrico, ou seja, eles armazenam energia. Tal armazenagem é realizada pelo mantimento de cargas elétricas num campo elétrico.

Tipicamente, os capacitores consistem em dois eletrodos ou placas que armazenam cargas de polaridades opostas e um dielétrico isolando tais placas. A Figura 4.39 mostra a estrutura de um capacitor formados por duas placas.

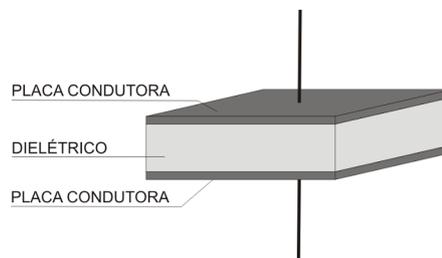


Figura 4.39: Estrutura básica de um capacitor

Fonte: [www.capacitor.com.hk](http://www.capacitor.com.hk)

Para calcular a capacitância de um capacitor semelhante ao da Figura 4.39, usa-se a seguinte expressão:

$$C = \epsilon_o \epsilon_r \frac{A}{d} \quad (4.21)$$

Onde  $C$  é a capacitância dada em *Farad*,  $\epsilon_o$  é a permissividade eletrostática do meio e  $\epsilon_r$  é a

constante dielétrica do isolante.

A Figura 4.40 mostra um circuito com uma fonte de tensão, um resistor e um capacitor. A placa do capacitor mais próxima do potencial elétrico mais elevado, será carregada com cargas positivas e a outra placa será carregada com cargas negativas.

A tensão entre os terminais do capacitor é dada pela seguinte expressão:

$$V = \frac{q}{C} \quad (4.22)$$

Onde  $q$  é a carga acumulada entre os terminais e  $C$  é a capacitância do capacitor.

A energia elétrica acumulada num capacitor é dada pela seguinte expressão:

$$U = \frac{1}{2}CV^2 \quad (4.23)$$

Onde  $U$  é a energia acumulada,  $C$  é a capacitância e  $V$  é a tensão entre os terminais do capacitor.

Capacitores são comumente usados em fontes de energia onde elas suavizam a saída de uma onda retificada completa ou meia onda. As aplicação dos capacitores para transmissão de energia elétrica são comumente relacionadas a correção de fator de potência. Fator de potência é um indicador do quando da energia produzida é efetivamente transmitida. Também podem ser usados em circuitos como filtro passa-baixa, passa-alta ou passa-banda, dependendo da configuração.

Neste trabalho, será explicado o filtro passa-baixa o qual tem aplicações diretas de processamento de sinais para circuitos como os utilizados em micro-controladores como a placa Galileo.

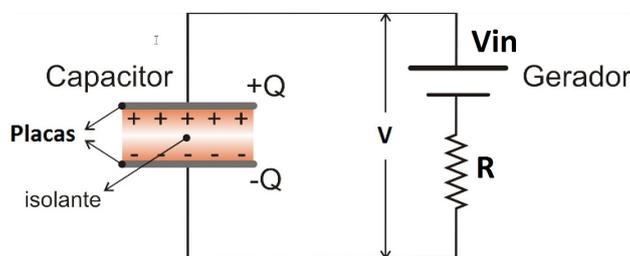


Figura 4.40: Circuito com um capacitor

Fonte: [www.capacitor.com.hk](http://www.capacitor.com.hk)

#### 4.2.12 Filtro RC

Um circuito formado com um resistor e um capacitor (filtro RC) é um dos mais simples filtros de sinais elétricos.

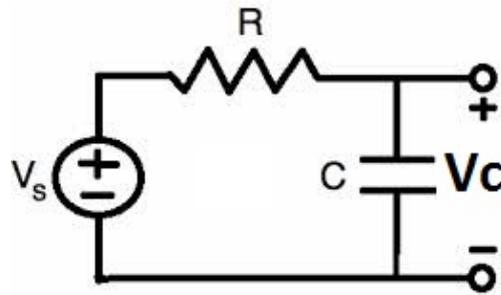


Figura 4.41: Filtro RC.

Fonte: hyperphysics.phy-astr.gsu.edu

Utilizando a fórmula da tensão entre os terminais de um capacitor (Equação 4.22) e a lei das tensões numa malha fechada (Equação 4.5) para o circuito mostrado da Figura 4.41, tem-se a seguinte expressão:

$$V_s - V_R - V_C = 0 \quad (4.24)$$

$$V_s = \frac{q}{C} + I * R \quad (4.25)$$

Muitas vezes, a fórmula da tensão elétrica apresentada na equação essa mesma expressão é tomada em função da corrente elétrica. Daí, tem-se a seguinte expressão

$$V_s = \frac{\int I dt}{C} + I * R \quad (4.26)$$

Tomando-se a derivada da equação 4.26, assumindo que  $V_s$  é uma tensão constante, tem-se:

$$0 = \frac{I}{C} + \frac{dI}{dt} * R \quad (4.27)$$

A solução dessa equação para a tensão sobre o capacitor, assumindo condições iniciais de tensão e corrente nulas, é a seguinte:

$$V_c = V_s(1 - e^{-\frac{t}{RC}}) \quad (4.28)$$

O gráfico mostrado na Figura 4.42 mostra a evolução da tensão sobre o capacitor e sobre o resistor num circuito RC. A constante  $\tau$  é igual a RC.

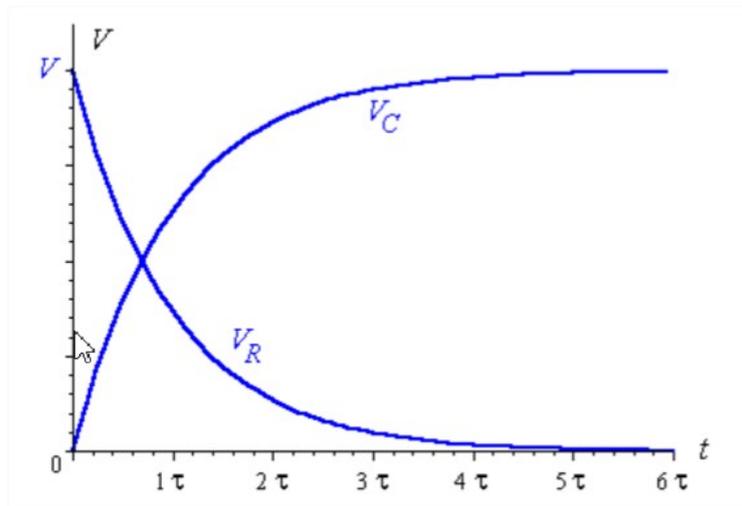


Figura 4.42: Evolução da tensão no capacitor e no resistor no tempo, num circuito RC.

Fonte: [hyperphysics.phy-astr.gsu.edu](http://hyperphysics.phy-astr.gsu.edu)

O termo  $RC$  da Equação 4.28 indica quanto tempo o capacitor leva para carregar 69 por cento do seu valor final. Quando o tempo tende para o infinito, o capacitor num circuito RC tende a se comportar como um circuito aberto. A tensão sobre o capacitor tende a tensão de alimentação e a corrente tende a zero, ou seja,

$$\lim_{t \rightarrow \infty} V_C = V_s \quad (4.29)$$

Com relação a sinais AC, circuitos RC são muito utilizados como filtros passa-baixa, como mostrado no diagrama de bode da Figura 4.43. Para frequências acima da frequência de corte  $\frac{1}{2\pi RC}$  a amplitude do sinal é atenuada.

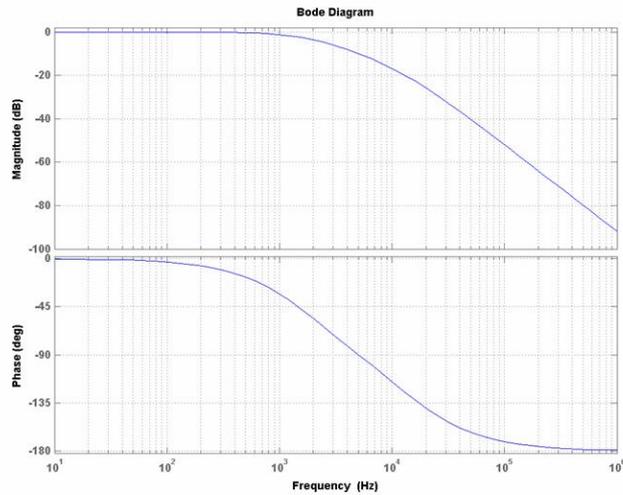


Figura 4.43: Diagrama de bode de um filtro RC.

Fonte: [hyperphysics.phy-astr.gsu.edu](http://hyperphysics.phy-astr.gsu.edu)

Para os propósitos deste trabalho, o filtro RC será utilizado para atenuar ruídos de alta frequência e realização a estabilização de sinais enviados à placa Galileo por meio de interruptores.

#### 4.2.13 Inversor Schmitt trigger

Um inversor é um circuito comparador com histerese com feedback na entrada positiva de um comparador ou um amplificado operacional. Define-se histerese como um comportamento relativo a variável de saída que é distinto nos caminho de aumento da variável de entrada e subsequente diminuição desta.

A Figura 4.44 mostra um sinal com histerese. Tendo a variável de entrada *in* partindo do valor  $-T$  a até o valor  $T$ , a variável de saída *out* terá valor  $-M$ . Após isso, caso a variável de entrada *in* volte para o valor  $-T$ , a variável de saída terá como valor  $+M$ .

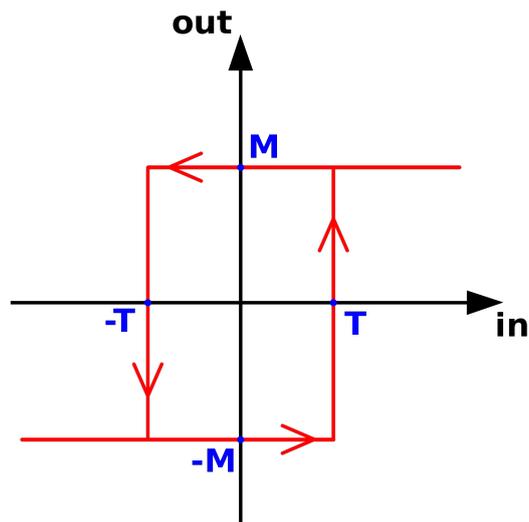


Figura 4.44: Sinal com histerese.

Um inversor *Schmitt trigger* pode ser implementado utilizando, como dito, feedback na entrada positiva de um amplificador operacional como mostrado na Figura 4.45.

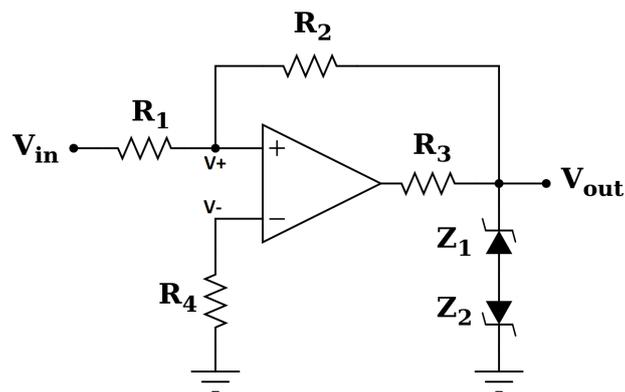


Figura 4.45: Implementação de um inversor com histerese com amplificador operacional e diodos zener.

Fonte: Adaptado de Alessio Damato - Own work, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=1178935>

O circuito mostrado na Figura 4.45 é um comparador implementado com um amplificador operacional. A saída do amplificador  $V_{out}$  obedece às seguintes expressões

$$V_{out} = +V_z, \text{ quando } V_+ > V_- \quad (4.30)$$

e

$$V_{out} = -V_z, \text{ quando } V_+ < V_- \quad (4.31)$$

$V_z$  é a tensão de quebra dos diodos zeners, conectados na saída do amplificador operacional.

As tensões de ida e volta de histerese para esse circuito são encontradas seguindo a seguinte expressão:

$$V_+ = V_{in} \cdot \frac{R2}{R1 + R2} + V_{out} \frac{R1}{R1 + R2} \quad (4.32)$$

A expressão da Equação 4.32 é obtida utilizando a segunda lei de Kirchoff (Equação 4.5) e o teorema da superposição.

Os pontos de quebra  $+M$  e  $-M$  são obtidas da equação 4.32 quando  $V_+$  é igual a  $V_-$ , ou seja:

$$+T = V_{in} = V_- \cdot \left(1 + \frac{R1}{R2}\right) - V_{out} \cdot \frac{R1}{R2} \quad (4.33)$$

e

$$-T = -V_{in} = -V_- \cdot \left(1 + \frac{R1}{R2}\right) + V_{out} \cdot \frac{R1}{R2} \quad (4.34)$$

Caso a tensão  $V_-$  seja igual a zero, tem-se:

$$T = V_{in} \cdot \frac{R1}{R2} \quad (4.35)$$

e

$$-T = V_{in} \cdot \frac{R1}{R2} \quad (4.36)$$

As Expressões 4.32, 4.33 e 4.34 são validas para o caso de  $V_- = 0$ . Caso  $V_-$  seja diferente de zero, o gráfico mostrado na Figura 4.44 será deslocado para direita ou esquerda a depender do valor de  $V_-$  e resistores  $R1$  e  $R2$  como indicado pelas Equações 4.32 e 4.33.

Em resumo, a saída de tensão do Schmitt trigger mostrado na Figura 4.45 se resumirá apenas aos valores  $+V_z$  e  $-V_z$  independentemente de ruídos na entrada.

O inversor Schmitt trigger a ser usado nas práticas propostas neste trabalho é o 74HC14 [3].

A Figura 4.46 mostra os pinos do chip 74HC14. No 74HC14, 6 inputs distintos podem ser invertidos usando os pinos de 1 a 12.

## 6. Pinning information

### 6.1 Pinning

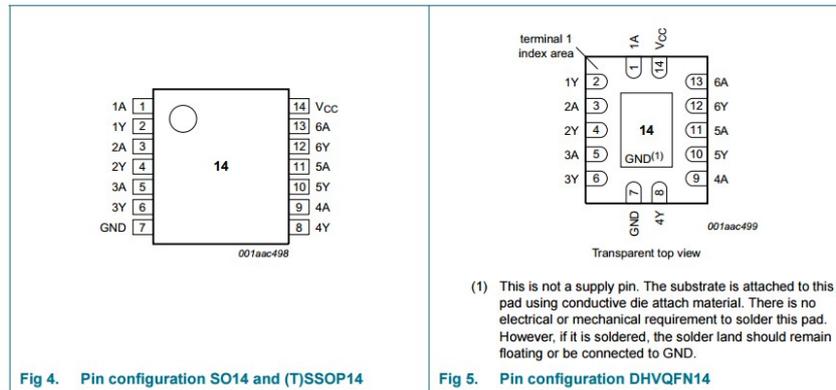


Figura 4.46: Inversor Schmitt trigger 74HC14 [3] - pinos.

A Figura 4.47 mostra diagramas lógicos do chip 74HC14.

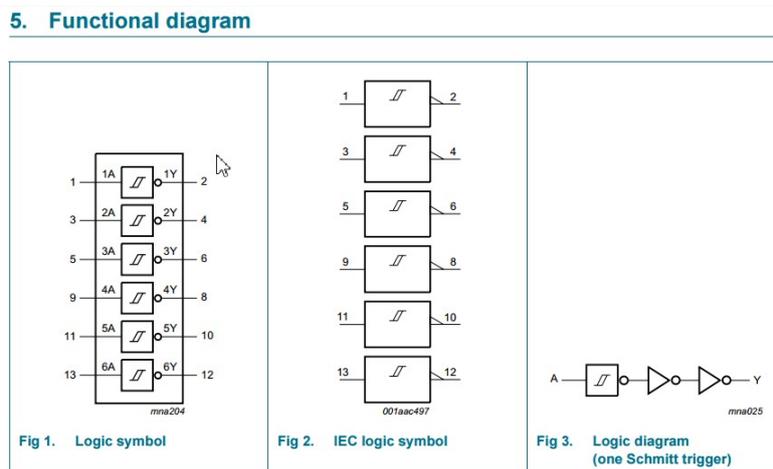


Figura 4.47: Inversor Schmitt trigger 74HC14 [3] - diagramas lógicos.

A Figura 4.48 mostra parâmetros eletrônicos e a curva de histerese do chip 74HC14.

Symbol	Parameter	Conditions	T <sub>amb</sub> = 25 °C			T <sub>amb</sub> = -40 °C to +85 °C		T <sub>amb</sub> = -40 °C to +125 °C		Unit
			Min	Typ	Max	Min	Max	Min	Max	
<b>74HC14</b>										
V <sub>T+</sub>	positive-going threshold voltage	V <sub>CC</sub> = 2.0 V	0.7	1.18	1.5	0.7	1.5	0.7	1.5	V
		V <sub>CC</sub> = 4.5 V	1.7	2.38	3.15	1.7	3.15	1.7	3.15	V
		V <sub>CC</sub> = 6.0 V	2.1	3.14	4.2	2.1	4.2	2.1	4.2	V
V <sub>T-</sub>	negative-going threshold voltage	V <sub>CC</sub> = 2.0 V	0.3	0.52	0.9	0.3	0.9	0.3	0.9	V
		V <sub>CC</sub> = 4.5 V	0.9	1.4	2.0	0.9	2.0	0.9	2.0	V
		V <sub>CC</sub> = 6.0 V	1.2	1.89	2.6	1.2	2.6	1.2	2.6	V
V <sub>H</sub>	hysteresis voltage	V <sub>CC</sub> = 2.0 V	0.2	0.66	1.0	0.2	1.0	0.2	1.0	V
		V <sub>CC</sub> = 4.5 V	0.4	0.98	1.4	0.4	1.4	0.4	1.4	V
		V <sub>CC</sub> = 6.0 V	0.6	1.25	1.6	0.6	1.6	0.6	1.6	V

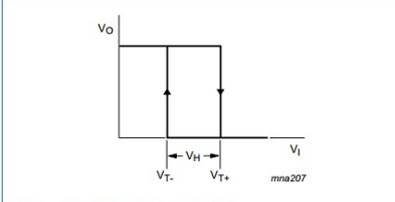


Fig Transfer characteristics

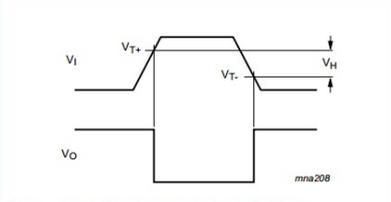


Fig Transfer characteristics definitions

Figura 4.48: Inversor Schmitt trigger 74HC14 [3] - parâmetros eletrônicos e curva de histerese.

#### 4.2.14 Debouncing - estabilização de sinais de interruptores

Um dos grandes problemas ao se trabalhar com interruptores ligados a interrupções de hardware é o fenômeno do *bouncing*. *Bouncing* é uma oscilação num sinal de um botão no momento que a tecla é apertada ou solta.

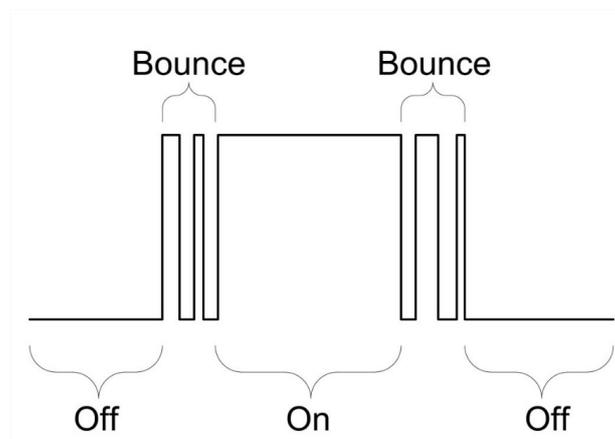


Figura 4.49: Fenômeno de *bouncing* de um sinal de um interruptor.

A Figura 4.49 mostra um exemplo do fenômeno do *bouncing*. Quando uma pessoa fecha um interruptor, o sinal deveria, quase instantaneamente, ir para o valor alto, entretanto, o sinal vai e volta aos nível baixo e alto até se estabilizar.

O mesmo fenômeno se repete quando o interruptor é aberto. O sinal vai e volta aos níveis alto e baixo até eventualmente se estabilizar.

O fenômeno do *bouncing* associado a interruptores causa problemas nas leituras de interrupções de hardware. Tomando a Figura 4.50 como exemplo, não apenas uma interrupção de rampa de

subida (Seção 4.3.15.1) será identificada pelo micro-controlador, mas 5.

Ao se identificar mais de uma rampa de subida, a rotina associada à tal interrupção será executada mais de uma vez.

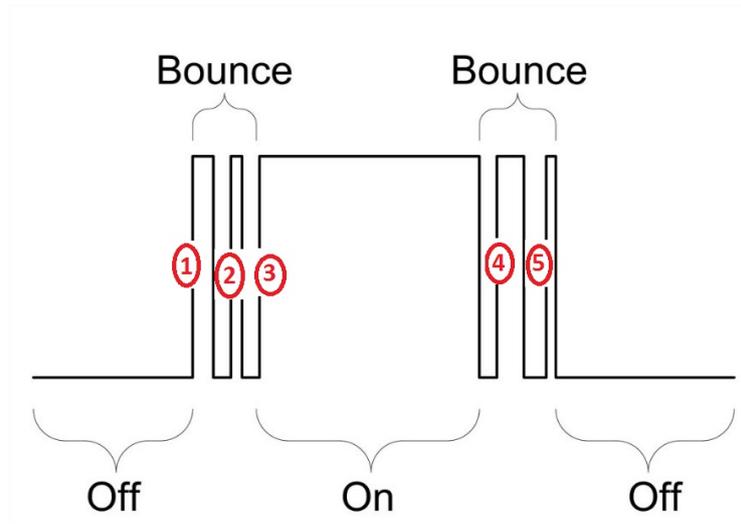


Figura 4.50: Interrupções de rampa de subida causadas pelo fenômeno do *bouncing*.

Para evitar o fenômeno do *bouncing*, deve-se utilizar um filtro RC (seção 4.2.12) associado a um inversor Schmitt trigger (seção 4.2.13).

O esquemático do circuito de debouncing é mostrado na Figura 4.51.

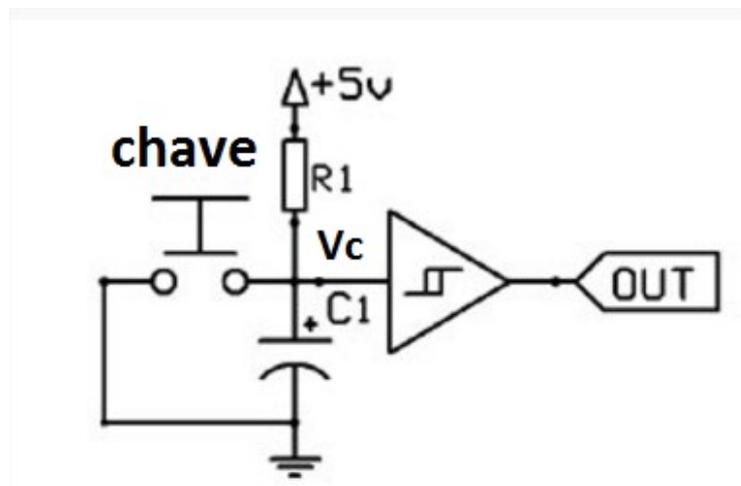


Figura 4.51: Circuito para *debouncing* de um sinal de um interruptor.

A tensão de saída é o inversor da tensão lida no capacitor  $V_c$ , obedecendo a curva de histerese do Schmitt trigger mostrada na Figura 4.48. Com o botão aberto, a tensão sobre o capacitor será 5 V, e a tensão de saída  $V_{out}$  será 0V ou 0 lógico

Quando a chave é fechada, o capacitor começa a descarregar. No final desse processo, a tensão sobre o capacitor será próxima a 0V e a tensão de saída será 5V ou lógico 1.

A Figura 4.52 mostra o resultado do uso do circuito de debounce. Aparte qualquer *bouncing* que ocorrer no sinal do interruptor, o uso do capacitor suavizará tais efeitos e o uso do inversor adequará os níveis lógicos do sinal de saída para a leitura de interrupções com apenas uma rampa de subida ou descida como sinal do interruptor.

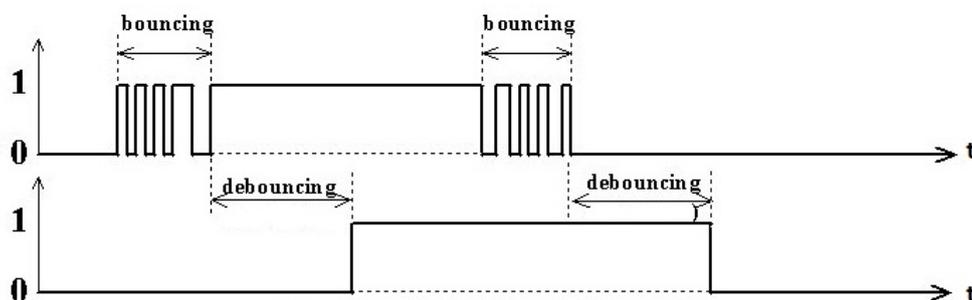


Figura 4.52: Resultado do uso do circuito de *debouncing* de um sinal de um interruptor.

Fonte: <http://www.carlitoscontraptions.com/2007/03/switch-debouncer/>

Uma escolha adequada para os valores do capacitor e resistor são respectivamente 10kohm e 10 uF. Com essa escolha a constante RC tem como valor 0.1 segundos para. Tem valor é mais que adequada ao tempo de resposta humana.

#### 4.2.15 Matriz de Leds

Muitas vezes, se deseja trabalhar com um conjunto de leds dispostos na formato de uma matriz como mostrado na Figura 4.53.

Para acender um LED em específico na estrutura da matriz de leds, deve-se setar uma das linhas com valor alto de tensão (valor lógico 1) e uma das colunas com valor baixo de tensão (valor lógico 0).

Caso se queria, por exemplo, acender o led localizado na linha 7 e coluna 2, deve-se colocar na trilha da linha 7 5 Volts (valor lógico 1).

Ao se setar linha ou coluna ao certo estado (0 ou 1 lógico), toda a linha ou coluna fica comprometida a esse estado, daí, não será possível acender uma linha e uma coluna ao mesmo tempo.

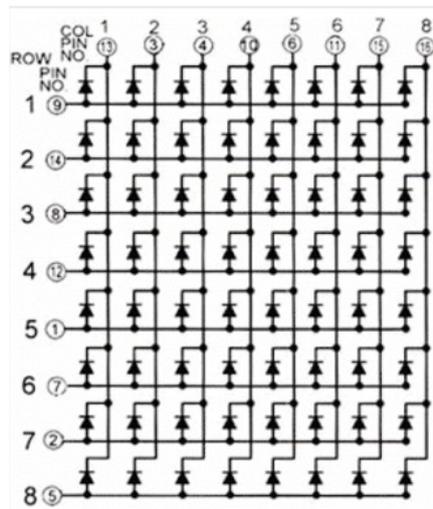


Figura 4.53: Estrutura esquemática de uma matriz de leds 8x8.

Fonte: [electronica.yoreparo.com](http://electronica.yoreparo.com)

Para poder acender uma linha e uma coluna, deve-se multiplexar tais estados, ou seja, acender e apagar, alternadamente, linhas e colunas rapidamente, de forma que a visão humana seja incapaz de perceber tal alternância de estados.

#### 4.2.16 Driver de matriz de led

Uma matriz de leds  $N \times N$  (Seção 4.2.15) pode ser controlado utilizando  $2N$  portas digitais. Por exemplo, se for usada uma matriz de led  $8 \times 8$  serão necessárias 16 portas digitais para controlar tal matriz.

Caso fosse usada uma matriz  $16 \times 16$ , seriam necessários 32 portas digitais para controle de tal matriz. Não há placa que seja capaz de oferecer tal quantidade de portas.

Para possibilitar o controle de tantos led sem utilizar tantas portas digitais, é necessário utilizar um circuito integrado que, como o chip 74HC595 (Seção 4.2.9), transforme comunicação serial em comunicação paralela.

Para realizar o propósito de controlar uma matriz de leds usando poucas portas digitais, deve-se usar um *driver de display*.

Neste trabalho, será usado o circuito integrado MAX7219 como *driver de display*. A Figura 4.54 mostra os pinos do MAX7219.

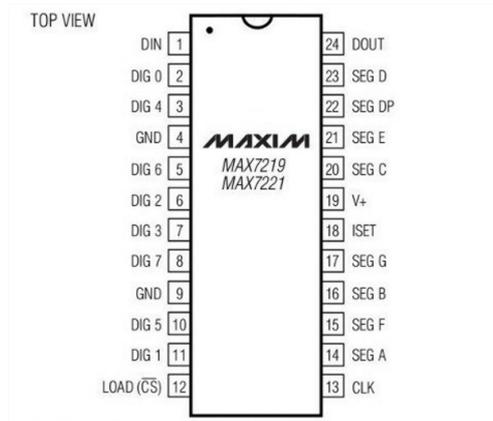


Figura 4.54: Pinos do circuito integrado MAX7219.

A conexão dos pinos é a seguinte:

- Pinos 2, 11, 6, 7, 3, 10, 5 e 8 : DIG0, DIG1, DIG2, DIG3, DIG4, DIG5, DIG6 respectivamente.
- Pinos 14, 16, 20, 23, 21 e 15: SEG A, SEG B, SEG C, SEG D, SEG E e SEG F respectivamente.
- Pino 1 (Din): Pino digital 12 da placa Galileo.
- Pino 13 (Clk): Pino digital 11 da placa Galileo.
- Pino 12 (LOAD(cs)): Pino digital 10 da placa Galileo.
- Pinos 4 e 9: Pino GND
- Pinos 18 e 19: Pino 5V

A Figura 4.55 mostra um esquemático de tais conexões realizadas.

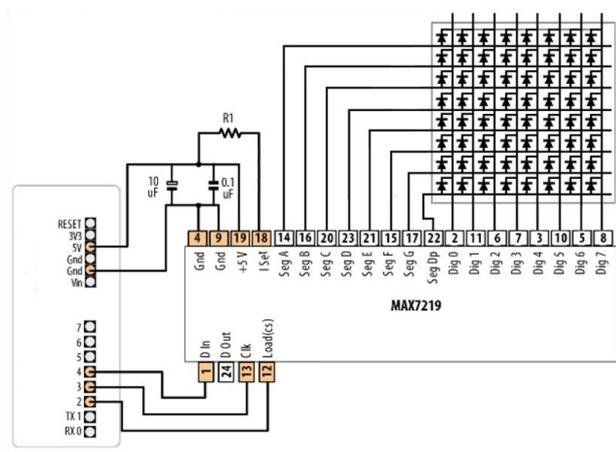


Figura 4.55: Conexão dos pinos do MAX7219 à placa Galileo.

O circuito da Figura 4.56 mostra um circuito integrado com a maioria das conexões mostradas na Figura 4.55 já feitas exceto os pinos Vcc, GND, Din, CS, e CLK.

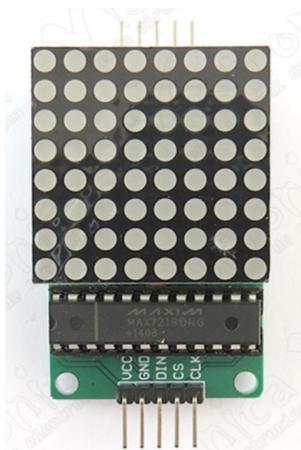


Figura 4.56: Placa matriz 8x8 + MAX7219.

## 4.3 Software

Nesta Seção são tratados todos conceitos relativos a programação que embasam as práticas propostas na seção 5.

### 4.3.1 Programação estruturada

Programação estruturada é um paradigma de programação cujo objetivo é clareza do código, qualidade e tempo de desenvolvimento de software e tempo de execução de algoritmo. Para alcançar tais objetivos, linguagens de programação que se baseiam no paradigma de programação estruturada possuem as seguintes características:

- **Sequência** bem definida de passos a serem seguidos utilizando:
  - Condicionais;
  - Loops;
  - Chamadas a funções iterativas e recursivas.
- **Modularização** do código, em:
  - Funções;
  - Estruturas de dados;
  - Uso de bibliotecas.

Para se resolver um determinado problema sob o paradigma de programação estruturada, deve-se subdividir tal problema em problemas menores.

A solução final é a junção sequencial e lógica dos problemas menores.

A subdivisão proposta pelo paradigma estruturado oferece as seguintes vantagens:

- Cada parte menor tem um código mais simples;
- Facilidade de entendimento do código, uma vez que os subprogramas podem ser analisados como partes independentes (legibilidade);
- Códigos menores são mais facilmente modificáveis para satisfazer novos requisitos do usuário e para correção de erros (manutenibilidade);
- Simplificação da documentação de sistemas;
- Desenvolvimento de software por equipes de programadores;
- Reutilização de subprogramas através de bibliotecas de subprogramas, na linguagem C, sob a forma dos arquivos de cabeçalhos (.h).

### 4.3.2 Programação para Arduino

Nesta sub-seção são explicados os conceitos relevantes para as práticas propostas na Seção 5.

#### 4.3.2.1 Padrão de programação em Arduino

Todo programa escrito para Arduino deve ter, obrigatoriamente, duas funções: função **setup** e a função **loop**.

A função **setup** é a primeira a ser executada pela plataforma programada em Arduino. Em geral, essa função serve para realizar configurações e definições iniciais.

A função **loop** é executada após a **setup** ser executada. A função **loop** será repetida indefinidamente, até a placa Galileo ser desligada ou o botão *reset* ser apertado. Caso o botão *reset* seja apertado, a execução do programa volta a seu início.

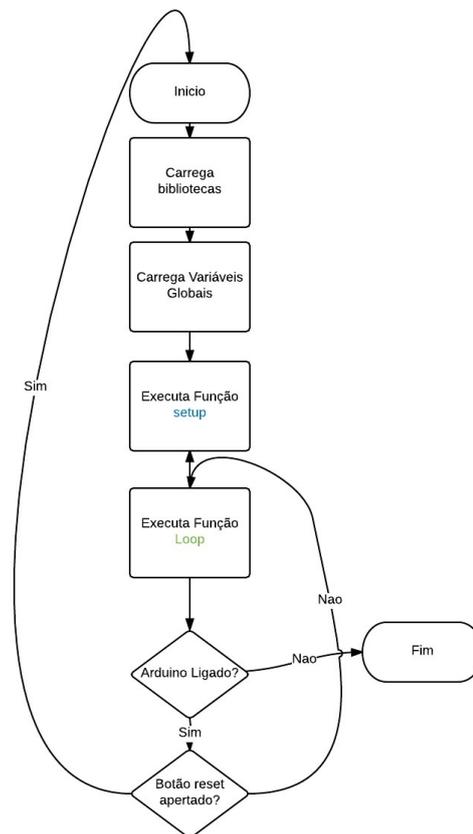


Figura 4.57: Fluxograma de um programa escrito para Arduino.

O fluxo comum de um programa escrito para Arduino é mostra na Figura 4.57. Quanto ao botão reset, a Figura 4.57 é simplificada. O botão reset pode ser apertado a qualquer momento. Ao ser apertado, o botão, é ativado uma *interrupção de hardware* a qual faz com que a execução do programa volte ao início.

### 4.3.3 Uso de portas digitais

Diz-se por porta digital, um pino, de entrada ou saída de tensão, que por ele podem ser lidos ou escritos apenas 0 ou 1 lógicos.

Para uma placa Galileo, 0 lógico é identificado como 0 Volts(GND) e 1 lógico é identificado como 3.3 ou 5 Volts entrando ou saindo da porta digital selecionada.

Seleciona-se 3.3 ou 5 Volts como valor de referência de tensão com o pino **IOREF**.

Deve-se tomar cuidado ao utilizar uma porta digital como entrada(*input*). Caso a tensão aplicada à porta for maior que a tensão selecionada em IOREF, pode-se inutilizar a porta.

Para se utilizar uma porta digital, deve-se seleciona-lá na função setup. No código mostrado em 4.1, mostra-se a seleção da porta digital 13 como saída (OUTPUT) de tensão.

```

1
2 void setup() {

```

```
3 // Seleciona a porta digital 13 como saída de tensao
4 pinMode(13, OUTPUT);
5
6 }
```

Code 4.1: Selecionando a porta digital 13 para saída de tensão

Para selecionar uma porta digital como porta de entrada de tensão, deve-se usar o comando *pinMode(NumeroPorta, INPUT);*.

O exemplo a seguir, Código 4.2, mostra a porta digital 13 sendo selecionada para entrada de tensão:

```
1
2 void setup() {
3 // Seleciona a porta digital 13 como entrada de tensao
4 pinMode(13, INPUT);
5
6 }
```

Code 4.2: Selecionando a porta digital 13 para entrada de tensão

Com relação a porta digital 13, deve-ser apontado que nela já está conectado, automaticamente, um resistor de 13kOhms. Dessa forma, por nela pode ser diretamente ligado um LED, sem correr o risco de queimá-lo.

#### 4.3.4 Processo de Compilação

Todo programa de computador, escrito em passa de uma tradução de uma linguagem compreensível por um ser humano para uma linguagem compreensível para um sistema computacional, ou seja, uma série de bits.

A Figura 4.58 mostra os passos realizados nesse processo. Para fins deste trabalho e desta seção será focado apenas nos passos pré-compilação e nos passos de análise léxica, sintática e semântica de um código.

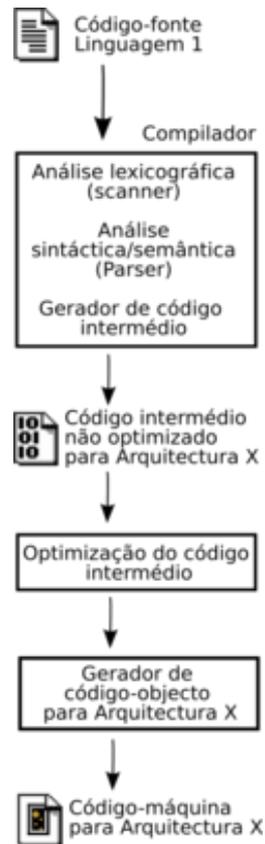


Figura 4.58: Fluxograma de um programa escrito para Arduino.

As etapas a serem tratadas nesta seção são, portanto, os seguintes:

1. Pré-processamento;
2. Análise Léxica;
3. Análise Sintática;
4. Análise Semântica.

#### 4.3.4.1 Pré-processamento

Na etapa de pré-processamento, as diretivas de compilação são resolvidas. Tudo aquilo que foi definido usando a diretiva **#define** é substituído, no texto do código fonte, por aquilo que definido no **#define**.

Além disso, no pré-processamento são removidos comentários e também substituídos no código fonte as macros definidas, como mostrado na Figura 4.59.

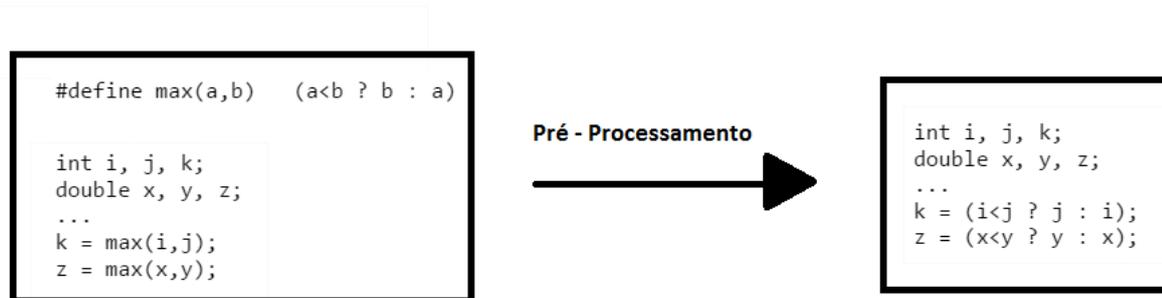


Figura 4.59: Exemplo de substituição de macro no pré-processamento.

#### 4.3.4.2 Análise Léxica

Após a fase de pré-processamento, o compilador realiza a análise léxica do código fonte. A análise léxica é feita por módulo chamado *analisador léxico* ou scanner. Nessa fase, o programa é lido da esquerda para a direita agrupando os caracteres de maneira a forma *tokens*. Os tokens são seqüências de caracteres que em conjunto possuem um significado, como palavras numa linguagem humana.

Na análise léxica, cada *token* identificado é analisado de acordo com as regras da linguagem de programação. Na linguagem de programação C, não é permitido a nenhum token ser iniciado com um número e tão pouco podem os tokens possuir caracteres especiais como \$ , %, #, etc.

#### 4.3.4.3 Análise Sintática

Após a análise léxica, onde os tokens identificados são validados, é realizada a análise sintática do código fonte.

A análise sintática resume-se na análise das expressões construídas com os tokens. Um exemplo de erro sintático é mostrado no Código 4.3. Nesse exemplo é mostrado o uso do operador + feito incorretamente.

```

1 int numero;
2
3 numero = 1 + ; //erro sintatico, o operador + espera dois termos, mas o segundo
4               // nao explicitado

```

Code 4.3: Erro sintático no uso do operador +

#### 4.3.4.4 Análise Semântica

Após a análise léxica, é feita a análise semântica. A análise semântica pode ser resumida como a "análise do relacionamento entre todas as partes do código". O exemplo mostra erros semânticos de referência indefinidas no Código 4.4.

```

1 void setup()
2 {
3 int a = funcao1(); // erro semantico, referencia indefinida a funcao1()
4
5 }
6
7 void loop()
8 {
9 int num;
10
11 num = num + k; //erro semantico, referencia indefinida a k
12 }

```

Code 4.4: Erro referência indefinida

### 4.3.5 Diretivas de Compilação

Como mencionado na Seção 4.3.4, o processo de compilação é iniciado pelo pré-processamento. No pré-processamento, as diretivas de compilação são resolvidas.

As diretivas de compilação mais comuns são `#include`, `#define`, `#ifdef`, `#else` e `#endif`.

- **#include**: A diretiva `#include` é utilizada para adicionar códigos fontes externos ao código fonte no qual se está trabalhando. As formas de uso da diretiva são as seguintes:

`#include <bibliotecaDoSistema.h>` : Nessa forma se adiciona uma biblioteca já nativa do sistema, como por exemplo a `stdlib.h`. A inclusão de tal biblioteca se daria pelo seguinte comando `#include <stdlib.h>`

`#include "bibliotecaDoDesenvolvedor.h"`: Nessa forma se adiciona uma biblioteca já desenvolvida pelo programador ou equipe relacionada. Caso o arquivo não se encontre na mesma pasta que o programa que o está adicionando, deve ser fornecido o caminho para ele no `include`. Por exemplo: `#include "/pasta1/pasta2/bibliotecaDoDesenvolvedor.h"`

- **#define**: A diretiva `#define` é usada para informar ao compilador quais tokens serão substituídos por determinadas expressões no processo de pré-processamento, como mostrado na Seção 4.3.4.1 ou para criar tokens com o status de **defined** que influenciaram a definição de diretivas posteriores.
- **#ifdef**, **#else** e **#endif**: A diretivas `#ifdef`, `#else` e `#endif` são usadas em conjunto para realizar definições de diretivas e macros condicionalmente. Um exemplo disso é mostrado no código 4.5, onde o valor da diretiva *constante* é definida de acordo a com definição da diretiva *teste*.

Um uso muito comum do padrão de diretivas mostrados no Código 4.5, é mostrado no Código 4.6. Na primeira vez que o compilador ver o arquivo de cabeçalho **bibliotecaExemplo.h**, a diretiva **BIBLIOTECAEXEMPLO** será definida e as funções definidas nesse arquivo passaram a ser

reconhecidas pelo compilador. Após essa definição e reconhecimento de funções, não haverá mais erros de redefinição de funções (erro semântico), pois as diretivas `# ifdef BIBLIOTECAEXEMPLO` `# define BIBLIOTECAEXEMPLO` impedirão isso.

```
1 #define teste // define a token teste
2
3 #ifdef teste // executa a acao dfinida dentro deste #ifdef, pois teste estah //
   definido
4 #define constante 3 // define diretiva constante com o valor 3
5 #else // nao entra nesta secao pois teste estah definifo
6 #define constante 4 // // define diretiva constante com o valor 4
7 #endif
```

Code 4.5: Uso de diretivas condicionais

```
1
2 #ifdef BIBLIOTECAEXEMPLO
3 #define BIBLIOTECAEXEMPLO // define a token teste
4
5 void funcao1()
6 {
7 }
8
9 void funcao2()
10 {
11 }
12
13 .
14 .
15 .
16 void funcaoN()
17 {
18 }
19
20 #endif
```

Code 4.6: Uso de diretivas para evitar erro de compilação de repetição de inclusão de arquivos

### 4.3.6 Tipos básicos de variáveis

Variáveis são os elementos básicos que um programa manipula. Uma variável é um espaço reservado na memória do computador para armazenar um tipo de dado determinado. Variáveis devem receber nomes para poderem ser referenciadas e modificadas quando necessário. Muitas linguagens de programação exigem que os programas contêm declarações que especifiquem de que tipo são as variáveis que ele utilizará e as vezes um valor inicial. Tipos podem ser por exemplo: inteiros, reais, caracteres, etc. As expressões combinam variáveis e constantes para calcular novos valores.

Os tipos básicos de variáveis são os seguintes:

- **char**: Caracter: O valor armazenado é um caractere. Caracateres geralmente são armazenados em códigos (usualmente o código ASCII).
- **int**: Número inteiro é o tipo padrão e o tamanho do conjunto que pode ser representado normalmente depende da máquina em que o programa está rodando.
- **float**: Número em ponto flutuante de precisão simples. São conhecidos normalmente como números reais.
- **double**: Número em ponto flutuante de precisão dupla
- **void**: Este tipo serve para indicar que um resultado não tem um tipo definido. Uma das aplicações deste tipo em C é criar um tipo vazio que pode posteriormente ser modificado para um dos tipos anteriores.

Cada um desses tipos é codificado diferentemente para cada arquitetura computacional para a qual o programa será compilado. A Tabela 4.3 mostra o uso de espaço de memória normalmente empregado nas arquiteturas computacionais.

Tabela 4.3: Tamanho e faixa de uso para os tipos básicos váriaveis

<b>Tipo</b>	<b>Tamanho em Bytes</b>	<b>Faixa Mínima</b>
<b>char</b>	1	-127 a 127
<b>int</b>	4	-2.147.483.648 a 2.147.483.647
<b>float</b>	4	Seis dígitos de precisão
<b>double</b>	8	Dez dígitos de precisão

Muitas vezes, as faixas para os tipos de variáveis mostradas na tabela 4.3 não são suficientes ou adequadas para o algoritmo em desenvolvimento. Para resolver tais problemas, podem ser utilizados *modificadores* para ampliar, reduzir ou adequar a faixa de valores que a variável pode guardar no endereço de memória reservado.

Os modificadores de variáveis e seus respectivos efeitos sobre a faixa de valores de cada tipo são mostrados na tabela

Tabela 4.4: Todos os Tipos de dados definidos pelo Padrão ANSI C, seus tamanhos em bytes e suas faixa de valores.

Tipo	Tamanho em Bytes	Faixa Mínima
char	1	-127 a 127
unsigned char	1	0 a 255
signed char	1	-127 a 127
int	4	-2.147.483.648 a 2.147.483.647
unsigned int	4	0 a 4.294.967.295
signed int	4	-2.147.483.648 a 2.147.483.647
short int	2	-32.768 a 32.767
unsigned short int	2	0 a 65.535
signed short int	2	-32.768 a 32.767
long int	4	-2.147.483.648 a 2.147.483.647
signed long int	4	-2.147.483.648 a 2.147.483.647
unsigned long int	4	0 a 4.294.967.295
float	4	Seis dígitos de precisão
double	8	Dez dígitos de precisão
long double	10	Dez dígitos de precisão

### 4.3.7 Conversão entre tipos de variáveis

Conforme explicado na seções anteriores, variáveis vêm em diferentes tipos. O tipo determina o tipo de dados que uma variável pode conter. Uma variável Integer pode conter somente dados numéricos sem pontos decimais. Uma variável char pode conter somente uma letra ou símbolo contido na tabela ASCII.

Muitas vezes, faz-se necessário ou mais conveniente realizar a conversão de um tipo de variável para outro mais conveniente. Para realizar a conversão, deve-se utilizar o comando escrever, ao lado da variável que será convertida a seguinte expressão (NOVOTIPO) variável. O Código 4.7 mostra duas situações onde uma variável inteira é convertida para um char.

Na primeira, não se perde informação, pois o conteúdo da variável *num* pode ser expresso em 1 byte. Na segunda situação, no entanto, se perde informação, pois 10000 (conteúdo da variável *num2*) não pode ser expresso em 1 byte.

```

1 int num = 1; //variavel ocupando 32 bits(4 bytes)
2
3 char a = (char)num; // conversao para variavel contendo 8 bits (1 byte)
4                // conversao sem perda de informacao
5
6 int num2 = 10000;

```

```

7
8 char b = (char) num2; // conversao com perda de informacao
9                          // uma variavel char so pode guardar ateh 127

```

Code 4.7: Convertendo int para char

### 4.3.8 Leitura Analógica - Conversão Analógico/Digital

Para ler o sinal de tensão analógico num dos 6 pinos destinados a conversão Analógico/Digital deve ser utilizada a função *analogRead()*. O código 4.8 mostra um exemplo da utilização da função *analogRead()*, conversão de int para float, e ajuste de escala.

A função *analogRead* tem como argumento de entrada o número do pino que será lido. O tempo de leitura é aproximadamente 100 µs. O retorno dessa função estará entre 0 e 1023, devido ao tamanho do registrador de conversão A/D mencionado na Sub-seção 2.4.2.2.

Para calcular o valor da tensão no pino de entrada na faixa de 0 a 5V é necessário realizar a conversão de variável citada na Seção 4.3.7.

```

1 void setup ()
2 {
3 }
4
5 int sensor;
6 float valorReal
7 void loop ()
8 {
9 sensor = analogRead(A0); // 0<= sensor <=1023
10
11 ValorReal = (float)(5*sensor)/1023 // conversão de tipo de variável e conversão //
//de escala usando uma regra de 3
12
13 }

```

Code 4.8: Uso da funcao analogRead

### 4.3.9 Estruturas Condicionais

Chama-se de estrutura condicional as instruções para testar se uma condição é verdadeira ou falsa.

Para expressar uma condição, deve-se utilizar operadores relacionais e, se necessário operadores lógicos.

#### 4.3.9.1 Operadores Relacionais

A Tabela 4.5 mostra todos operadores relacionais possíveis. É mostrado, na coluna exemplo, que as expressões entre os operadores podem ser quaisquer, desde que não incorreram em erros

léxicos, sintáticos ou semânticos.

Tabela 4.5: Operadores relacionais

Operador relacional	Exemplo de aplicação	Explicação do exemplo
$==$	$a == b$	verifica se a é igual a b
$!=$	$a != b * 3$	verifica se a é diferente de b multiplicado por 3
$>$	$a > (b + a)$	verifica se a é maior do que b + a
$>=$	$(a - b + 5) >= (b / 2)$	verifica se (a - b + 5) é maior ou igual a b/2
$<$	$a < b$	verifica se a é menor do que b
$<=$	$a <= b$	verifica se a é menor ou igual a b

Toda vez que uma operação lógica é aplicada, é retornado TRUE ou FALSE. Os sistemas computacionais identificam FALSE com o valor zero e TRUE como qualquer valor diferente de zero (inclusive valores negativos).

#### 4.3.9.2 Operadores Lógicos

Operadores lógicos são usados para combinar 2 ou mais operações relacionais. A tabela 4.6 mostra os operadores lógicos, exemplo e explicação de tais exemplos.

Tabela 4.6: Operadores **E**, **OU** e **Negação**

Operador lógico	Exemplo de aplicação	Explicação do exemplo
(Operador E) &&	((a == b)&&( a >0))	verifica se a é igual b E verifica se a é maior que 0 se as duas expressões forem verdadeiras, retorna-se TRUE nessa operação lógica do contrário, retorna FALSE
(Operador OU) 	(a >b )   ( b >3)	verifica se a é maior do que b OU se b é maior que 3 se qualquer uma das expressões for verdadeira ,retorna-se TRUE Se a duas expressões forem FALSE, retorna FALSE
(Operador negação) !	!(a >b)	verifica se a é maior do que b. Caso isso seja verdadeiro, retorna FALSE Caso a expressão lógica seja falsa, retorna TRUE

As tabelas verdades das Tabelas 4.7, 4.8 e 4.9 mostram as mesmas relações exemplificadas na Tabela 4.6.

Tabela 4.7: Tabela verdade do operador ||.

A	B	A    B
0(FALSE)	0(FALSE)	0(FALSE)
0(FALSE)	1(TRUE)	1(TRUE)
1(TRUE)	0(FALSE)	1(TRUE)
1(TRUE)	1(TRUE)	1(TRUE)

Tabela 4.8: Tabela verdade do operador && .

<b>A</b>	<b>B</b>	<b>A &amp;&amp; B</b>
0(FALSE)	0(FALSE)	0(FALSE)
0(FALSE)	1(TRUE)	0(FALSE)
1(TRUE)	0(FALSE)	0(FALSE)
1(TRUE)	1(TRUE)	1(TRUE)

Tabela 4.9: Tabela verdade do operador ! .

<b>A</b>	<b>!A</b>
0(FALSE)	1(TRUE)
1(TRUE)	0(FALSE)

#### 4.3.9.3 Controle de fluxo de execução de um programa (Condicionais)

Os operações lógicas construídas juntamente a operadores relacionais são a base para o controle do fluxo com os seguintes operadores:

- if
- if - else
- if - (else if) - else (if-else aninhados)
- switch - case
- operador ? (ternário)

A Figura 4.60 mostra o resumo do que acontece ao se utilizar qualquer dos operadores de controle de fluxo supracitado. Se certa condição lógica é verdadeira (TRUE) se executa uma série

de instruções específicas, senão se executa as intruções especificadas para o caso adverso e então se contínua o fluxo normal do programa.

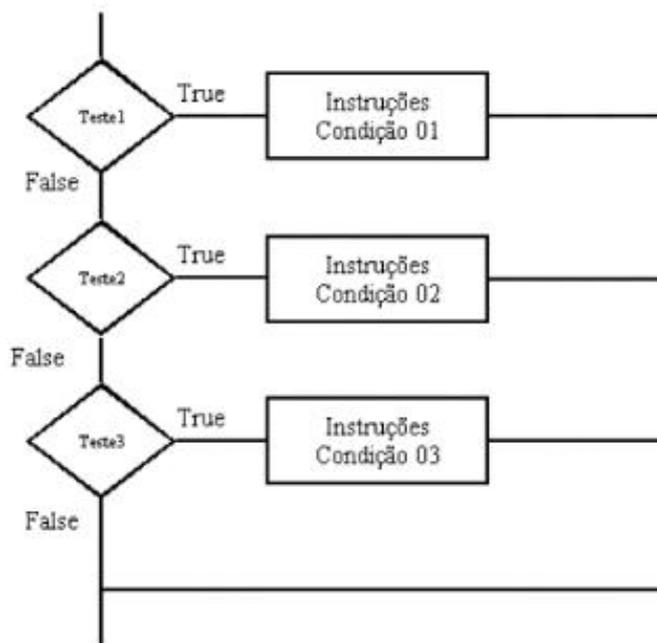


Figura 4.60: Resumo do controle de fluxo de um programa

O operador **if** possui a sintaxe mostrada no Código 4.9. Caso a condição lógica especificada por *condicao1* seja verdadeira, ou seja, caso tal condição retorne verdadeiro (TRUE), os procedimentos dentro das chaves ( { } ) devem ser executados.

```
1 if(condicao1)//verifica se as condições lógicas especificadas em
2 { //condicao1 sao
3     // verdadeira(TRUE). Caso sejam , executa os
4     // procedimentos dentro da chave
5     condicao1Procedimento1;
6     condicao1Procedimento2;
7     .
8     .
9     .
10    condicao1ProcedimentoN;
11 }.
```

Code 4.9: Sintaxe do if

O Código 4.10 mostra um exemplo de uso do if. Supondo que exista uma certa variável chamada *sensor*, caso ela seja maior que 300, a placa Galileo deve escrever na porta digital 2 um valor alto (HIGH) de tensão.

```
1 if(sensor > 300)
2 {
3     digitalWrite(2,HIGH);
```

```
4 }
```

Code 4.10: Exemplo de uso do if

O operador if - else segue o mesmo padrão mostrado nos Códigos 4.9 e 4.10. A diferença, nesse caso, é a adição do else (senão) ao bloco. Caso a condição especificada no if não seja verdadeira, o programa deve executar a série de instruções especificadas dentro do bloco do else. O Código 4.13 mostra a sintaxe de uso do bloco if - else e o Código 4.14 mostra um exemplo de aplicação do bloco if - else.

```
1 if(condicao1)
2 {
3   condicao1Procedimento1;
4   condicao1Procedimento2;
5   .
6   .
7   .
8   condicao1ProcedimentoN;
9 }.
10 else // caso a condição especificada por condicao1 seja falsa (FALSE), deve se
11 { // executar a série de instruções especificadas dentro do else
12   elseCondicao1Procedimento1;
13   elseCondicao1Procedimento2;
14   .
15   .
16   .
17   elseCondicao1ProcedimentoM;
18 }
```

Code 4.11: Sintaxe do bloco if - else

```
1 if(sensor > 300)
2 {
3   digitalWrite(2,HIGH);
4 }
5 else
6 {
7   digitalWrite(2,HIGH);
8   contador = contador + 1;
9 }
10 }
```

Code 4.12: Exemplo de uso do bloco if - else

O bloco if-else pode ser estendido indefinidamente com condições intermediárias entre o if e o else final. Como mostrado no Código 4.13, o programa continuará verificando as condições especificadas dentro dos if's enquanto não for encontrada uma condição verdadeira. Se nenhuma condição verdadeira for encontrada, executar-se-ão os procedimentos dentro do else, caso este tenham sido especificado (pode-se escrever uma série de else -if's sem um else final).

```
1 if(condicao1)
2 {
```

```

3   procedimentos1;
4 }
5 else if(condicao2)// verifica a condicao2 se a condicao1 eh falsa
6 {
7   procedimentoElse1;
8 }
9 else if(condicao3) // verifica a condicao3 se as condicoes 1 e 2 sao falsas
10 {
11   procedimentosElse2;
12 }
13 .
14 .
15 .
16 else if(condicaoN)//verifica a condicaoN se as condicoes 1,2...N-1 sao falsas
17 {
18   procedimentosElseN
19 }
20 else // executa os procedimento especificados se todos condicoes 1, 2, ... N
21 {   // sao falsas
22
23   procedimentosElse;
24 }

```

Code 4.13: Sintaxe do bloco if - (else if) - else (if-else aninhados)

O Código 4.14 mostra um exemplo de uso do bloco if - (else if) - else (if-else aninhados).

```

1 if(sensor > 300)
2 {
3   digitalWrite(2,HIGH);
4 }
5 else if(sensor2 > 300)
6 {
7   digitalWrite(3,HIGH);
8   contador2 = contador + 1;
9 }
10 else
11 {
12   digitalWrite(2,LOW);
13   digitalWrite(3,LOW);
14   contador = 0;
15   contador2 =0;
16 }

```

Code 4.14: Exemplo de uso do bloco if - (else if) - else

Pode-se usar blocos condicionais dentro de blocos condicionais. O Código 4.15 mostra um exemplo disso:

```

1 if(sensor > 300)
2 {
3   digitalWrite(2,HIGH);
4   if(sensor2 > 100)

```

```

5  {
6    if(sensor3 > 100)//executa essa verificacao apenas se a verificacao
7    {      // (sensor2 > 100) for verdadeira
8      digitalWrite(3,HIGH);
9    }
10   else
11   {
12     digitalWrite(3,LOW);
13   }
14 }
15 }
16 else if(sensor2 > 300)// executa essa verificacao se a condicao (sensor > 300)
17 {      // for falsa
18 digitalWrite(3,HIGH);
19 contador2 = contador + 1;
20 }

```

Code 4.15: Exemplo de uso do bloco if - (else if) - else

O operador switch - case opera segundo a sintaxe mostrada no Código 4.16. O bloco switch - case realiza a verificação da igualdade da variável especificada dentro dos parênteses do switch com relação aos valores especificados nos case's.

Quando uma das verificações propostas nos case's se mostra verdadeira, os procedimentos especificados. Após a execução de tais procedimentos, encontra-se o procedimento **break;**. O procedimento break faz com que o programa pule para fora do bloco switch, ignorando todos os outros case's não verificados.

Caso nenhum dos case's seja verdadeiro, executa-se os procedimentos especificados no bloco *default*..

```

1 switch (var) { // seleciona a variavel var para ser comparada em todos os
2     // case's até achar um que seja verdadeiro, caso nenhum seja
3     // achado, executa-se os procedimentos default
4 case Valor1 :// if (var == Valor1)
5
6 procedimentos1;
7
8 break;
9
10 case Valor2 : // if (var == Valor2)
11
12 procedimentos2;
13
14 break;
15
16 .
17 .
18 .
19
20 case ValorN :// if (var == ValorN)
21

```

```

22 procedimentosN;
23
24 break;
25
26 default:
27
28 procedimentosDefault;
29 }

```

Code 4.16: Sintaxe do switch - case

O Código 4.17 mostra um exemplo de aplicação do bloco switch - case.

```

1
2 int botao = digitalRead(4); // ler o pino digital 3
3 switch (botao) { // seleciona a variavel botao para ser comparada em todos os
4 // case's até achar um que seja verdadeiro, caso nenhum seja
5 // achado, executa-se os procedimentos default
6 case HIGH:
7
8 digitalWrite(2, HIGH);
9 digitalWrite(3, LOW);
10
11 break;
12
13 case LOW:
14
15 digitalWrite(2, LOW);
16 digitalWrite(3, HIGH);
17
18 break;
19
20 default:
21
22 digitalWrite(2, LOW);
23 digitalWrite(3, LOW);
24 }

```

Code 4.17: Exemplo de uso do bloco switch - case

O operador `?` ou *ternário* possui a sintaxe mostrada no Código 4.18. Caso a condição especificada for verdadeira, o valor ou variável especificada à esquerda dos dois pontos (`:`) é escolhida para ser atribuída a variável *var*. Caso a condição especificada for falsa, o valor ou variável especificada à direita dos dois pontos (`:`) é escolhida para ser atribuída a variável *var*.

```

1 var = (condicao) ? selecionaSeCondicaoVerdadeira : selecionaSeCondicaoFalsa;
2
3 //O ternário acima é equivalente a
4
5 if(condicao)
6 {
7   var = selecionaSeCondicaoVerdadeira;
8 }

```

```

9 else
10 {
11     var = selecionaSeCondicaoFalsa;
12 }

```

Code 4.18: Sintaxe de utilização do operador ?

Um exemplo da aplicação do operador `:` é mostrado no Código 4.19.

```

1 int i = 2;
2 int var;
3 var = ( i > 1 ) ? 100 : 200;
4
5 //O ternário acima é equivalente a
6
7 if( i > 1)
8 {
9     var = 100;
10 }
11 else
12 {
13     var = 200;
14 }

```

Code 4.19: Exemplo de uso do operador ?

### 4.3.10 Laços de repetição

No codificação de muitos algoritmos, é usual existir a necessidade de repetir certos comandos um determinado número de vezes ou até que certa condição seja atingida.

Uma solução rápida para conseguir realizar certos comandos um número determinado de vezes seria repetir o código o número de vezes desejado, simplesmente copiado as linhas de código desejadas, como mostrado no Código 4.20.

```

1
2 // REPETINDOS OS COMANDO1 até COMANDON 3 VEZES
3 // COPIANDO E COLANDO AS LINHAS DESEJADAS
4
5 comando1;
6 comando2;
7 .
8 .
9 .
10 comandoN;
11
12 comando1;
13 comando2;
14 .
15 .
16 .
17 comandoN;

```

```
18
19 comando1 ;
20 comando2 ;
21 .
22 .
23 .
24 comandoN ;
25
26
```

Code 4.20: Uma possível solução para repetir um comando um número determinado de vezes

Isso tipo de solução para o problema da necessidade de repetição de código, além de poder trazer diversas erros na produção do código, deixa o código grande e limitado.

Para facilitar a codificação de trechos de códigos que devem ser repetidos certo números de vezes ou até que certa condição seja atingida, existem os *laços de repetição*.

Na linguagem de programação C, existem os seguintes estruturas para criar laços:

1. Laço for
2. Laço while
3. Laço do-while

Nesta seção, essas 3 estruturas serão apresentadas.

#### 4.3.10.1 Laço for

O laço *for* serve, primariamente, para repetir certos comando um determinado número de vezes. A Figura 4.61 mostra a sintaxe para o uso do operador for. Primeiramente, deve-se escrever a palavra **loop** e abrir parênteses.

A região na Figura 4.61 indicada por **Inicialização de contador** será usada, como dito, para atribuir à variável responsável por contar quantas vezes o loop já foi executado com alguma valor.

A região na Figura 4.61 indicada por **atualização de contador** será usada, como dito, para , de alguma maneira desejada, atualizar a variável responsável por contar quantas vezes o loop já foi executado. Usualmente se executa  $contador = contador + 1$  nessa região a cada vez que o loop é executado.

A região na Figura 4.61 indicada por **teste de condição de fim de loop** será usada, como dito, para verificar se o loop deve continuar a ser executado ou não. Caso a verificação lógica executada nessa região resulte em FALSE, o loop será encerrado e o programa continuará a executar as instruções abaixo do loop .

```

INSTRUÇÕES A SEREM EXECUTADAS ANTES DO LOOP
//Início do loop
for( inicialização de contador ; teste de condição de fim de loop; atualização de contador )
{
    procedimento1;
    procedimento2;
    .
    .
    .
    procedimentoN;
}
//fim do loop
INSTRUÇÕES A SEREM EXECUTADAS APÓS O LOOP

```

Figura 4.61: Sintaxe de uso do loop for.

Um exemplo de uso de um loop for é mostrado na Figura 4.62. Nesse exemplo, uma inteira variável chamada *cont*, foi inicializada com o valor 0, na região indicada por **inicialização de contador** na Figura 4.61.

Num primeiro instante, a verificação lógica da região descrita por **teste de condição de fim de loop**  $cont < 10$  será executada. Ou seja, a verificação  $0 < 10$  será verificada. Tal verificação lógica resultará em TRUE, portanto o loop continuará.

Dentro do loop, existe a instrução *digitalWrite(cont, HIGH);*. Como o valor atual de *cont* é 0, a instrução é equivalente, nesse instante a *digitalWrite(0, HIGH);*, ou seja, a porta digital 0 apresentará um valor alto de tensão após isso.

Após todas instruções internas ao loop serem executadas, será executada a instrução relativa a região **atualização contador**. No exemplo mostrado, a atualização do contador é a seguinte  $cont = cont + 1$ , ou seja,  $cont = 0(\text{antigovalordecont}) + 1 \rightarrow cont = 1$ .

Esse mesmo padrão se repetirá, fazendo com que as porta 0 até 9 sejam setadas com valor alto de tensão.

Quando a variável *cont* chegar ao valor 10, a verificação lógica  $cont < 10$  não mais será verdadeira, e o laço será finalizado.

```

int cont;
for( cont = 0 ; cont < 10; cont = cont + 1 )
{
    digitalWrite(cont, HIGH); // aciona a porta digital numerada por cont
                             // para valor alto de tensão
}

```

Figura 4.62: Exemplo de uso do loop for.

Um aspecto importante a ser destacado para todas estruturas de repetição na linguagem C

é que todas variáveis usadas nas regiões que definem a estrutura do loop devem ser declaradas anteriormente ao loop.

As variáveis declarada interiormente ao loop não poderão fazer parte das regiões que definem o loop.

O Código 4.21 mostra alguns erros comuns na utilização do loop for

```
1
2 for(i = 0; i < 10; i = i -1)
3 {
4     int soma;
5     soma = soma + i;
6
7     //ERRO 1: variavel i não declarada antes do loop
8     //ERRO 2: Esse erro possivelmente não será detectado pelo
9     // pelo compilador. Ao executar esse loop, a condição de fim
10    // de loop nunca será falsa, devido a forma de atualização
11    // contador
12    //ERRO 3: A variavel definida no interior do loop é somada
13    // consigo mesma e com o contador, entretanto, a variável
14    // soma não possui valor inicial, causando um possível erro
15    // em tempo de execução
16 }
17
18 soma = soma + 10;
19 // ERRO 4: A variável soma é definida apenas interiormente ao //loop referenciá-la
    // fora do loop não é possível
20
21
22
```

Code 4.21: Alguns erros comuns a utilizar loops em C

Já o código 4.22 mostra o mesmo código mostrado em 4.21 corrigido.

```
1
2 //DEFININDO CONTADOR E VARIÁVEL A SER USADA FORA DO
3 //LOOP ANTES DO LOOP
4 int soma = 0;
5 int i;
6
7 for(i = 0; i < 10; i = i +1)// USANDO UMA APROPRIADA DE ATUALIZA
8 {
9     //--CAO DE CONTADOR
10    soma = soma + i;
11 }
12 soma = soma + 10;
```

Code 4.22: Código sem erros

### 4.3.10.2 Laço while

O laço *while* serve, primariamente, para repetir certos comando enquanto certa condição não for atingida. A Figura 4.63 mostra a sintaxe comum de utilização do laço *while*.

```
Inicialização e definição de contador  
ou variável para verificação de fim loop;  
while( teste de condição de fim de loop)  
{  
    comando1;  
    comando2;  
    .  
    .  
    .  
    comandoN;  
    Atualização de contador ou variável de verificação;  
}
```

Figura 4.63: Sintaxe comum de uso do loop while.

A região na Figura 4.64 indicada por **Inicialização de contador ou variável para verificação de quebra de fim loop** será usada, como dito, para atribuir à variável responsável por contar quantas vezes o loop já foi executado com alguma valor ou inicializar a variável que a cada loop irá servir para verificar se a condição de fim de loop já foi atingida.

A região na Figura 4.64 indicada por **teste de condição de fim de loop** será usada, como dito, para verificar se o loop deve continuar a ser executado ou não. Caso a verificação lógica executada nessa região resulte em FALSE, o loop será encerrado e o programa continuará a executar as instruções abaixo do loop .

A região na Figura 4.64 indicada por **atualização de contador** será usada, como dito, para , de alguma maneira desejada, atualizar a variável responsável por indicar que o loop chegou ao fim.

O Código 4.23 mostra um exemplo de uso do while.

```
1 //Inicialização de contador ou variável para verificação de //quebra de fim loop  
2 int soma = 0;  
3 int i = 0;  
4  
5 while (i < 100) //teste de condição de fim de loop  
6 {  
7     soma = soma + 1 // procedimentol  
8  
9     i = i + 1; //Atualizacao de contador ou variavel de  
10     // verificacao  
11 }
```

Code 4.23: Exemplo de uso do laço while

Ao iniciar o loop *while*, a primeira coisa que acontece é a verificação da condição de fim de loop.

No exemplo mostrado no Código 4.23, a variável usada como contador é a variável **i**. Ela é inicializada com o valor 0. O teste de fim de loop, no primeiro instante, é, portanto, o seguinte: ( $0 < 100$ ). Essa verificação lógica terá como resultado TRUE, daí, o loop será executado.

A cada iteração do loop while citado, a variável **i** terá seu valor incrementado por uma unidade.

Quando **i** for igual a 100, o loop while será finalizado.

Um dos usos muito comuns para o loop while, é a verificação do estado de certa variável de software ou de hardware. O Código 4.25 mostra essa aplicação para o loop while num programa feito para a placa Galileo.

```
1
2 #define botao1 2
3 #define led 3
4 void setup()
5 {
6   pinMode(botao1, INPUT);
7   pinMode(led, OUTPUT);
8 }
9
10 void loop()
11 {
12   delay(100);
13   digitalWrite(led, LOW);
14
15   //Enquanto o botao conectado a porta digital 2
16   //nao for apertado
17   //repete o loop while sem fazer coisa alguma
18   //alem de verificar o estado do botao
19   while(digitalRead(botao1) != HIGH)
20   {
21   }
22
23
24   //botao apertado, pisca LED
25   digitalWrite(led, HIGH);
26
27 }
```

Code 4.24: Exemplo de uso do laço while para a placa Galileo

### 4.3.10.3 Laço do-while

O laço *do-while* possui o mesmo comportamento do laço while. A única diferença é que, ao usar o *do-while* é garantido que as instruções internas ao loop serão executadas pelo menos uma vez. Isso deve ao fato de que a verificação de fim de loop só ocorre ao final do loop.

A sintaxe do *do-while* é como mostrado na Figura 4.64

```
Inicialização e definição de contador  
ou variável para verificação de fim loop;  
do {  
    comando1;  
    comando2;  
    .  
    .  
    comandoN;  
    Atualização de contador ou variável de verificação;  
} while( teste de condição de fim de loop);
```

Figura 4.64: Sintaxe comum de uso do loop do-while.

Um exemplo de uso do *do-while* é mostrado no Código 4.25:

```
1  
2 #define botao1 2  
3 #define led 3  
4 void setup()  
5 {  
6 pinMode(botao1, INPUT);  
7 pinMode(led, OUTPUT);  
8 }  
9  
10 int flag = 0;  
11 void loop()  
12 {  
13     if(flag == 0)  
14     {  
15         do{  
16  
17             delay(100);  
18             digitalWrite(led, LOW);  
19             delay(100);  
20             digitalWrite(led, HIGH);  
21  
22         }while(digitalRead(botao1) != HIGH);  
23     }  
24  
25     flag = 1;  
26 }
```

Code 4.25: Exemplo de uso do laço while para a placa Galileo

Esse exemplo possui o comportamento contrário ao comportamento do Código 4.25. Nesse exemplo, o LED piscará continuamente até que o botão seja apertado. Quando o botão for apertado, o programa sairá do loop e a variável *flag* será setada com o valor 1. Ao fazer isso, não mais o programa executará o laço do-while, por causa da verificação *if( flag == 0)*.

### 4.3.11 Vetores e Matrizes

Vetores, também chamados de arrays, e matrizes são conjunto de dados de mesmo tipo. São usadas para tratar grandes quantidades de dados sem a necessidade de declaração de muitas variáveis.

Nesta seção, serão tratados os conceitos de vetores e matrizes alocadas estaticamente na memória (antes da execução do programa) de forma detalhada. Na Seção 4.3.12 serão tratadas, além de outros tipos mais avançados de variáveis, vetores e matrizes alocadas dinamicamente, ou seja, durante o tempo de execução do programa.

#### 4.3.11.1 Vetores

Vetores são conjuntos de dados para serem tratados em apenas uma dimensão de indexação. A Figura 4.65 mostra a alocação para *N* espaços de memória para um vetor.



Figura 4.65: Alocação de memória para um vetor declarado para *N* espaços de memória.

Fonte: Adaptado de <http://www.tiexpert.net/programacao/c/vetores.php>

O Código 4.26 mostra a sintaxe de declaração de um vetor em C.

```
1 tipo nome_da_variavel [quantidade_de_elementos];
```

Code 4.26: Sintaxe de declaração de um vetor em C

Exemplo de declaração de vetores de vários tipos de dados são mostrados no Código 4.27.

```
1
2 //DECLARACAO DE VETORES DE VARIOS TIPOS DE VARIAVEIS
3
4 int v_int [100]; //declara um vetor de 100 inteiros
5 char v_char [100]; //declara um vetor de 100 caracteres
```

```

6
7 float v_float[100]; //declara um vetor de 100 numero racionais

```

Code 4.27: Exemplos de declaração de um vetor em C

Nos exemplos de declaração do Código 4.28, são declarados conjuntos de dados sequenciais.

Para acessar, atribuir ou modificar os elementos de um vetor, a sintaxe do Código 4.28 é usada.

```

1 tipo nome_da_variavel [quantidade_de_elementos];
2
3 nome_da_variavel[indice] = valor;

```

Code 4.28: Sintaxe de atribuição de um valor a certo índice num vetor

Exemplos de atribuição de valor a certos índices em vetores são mostrados no Código 4.29.

```

1
2 int v_int [100]; //declara um vetor de 100 inteiros
3 int vetor_digitos [10]; //declara um vetor de 10 inteiros
4 char v_char[100]; //declara um vetor de 100 caracteres
5 float v_float[100]; //declara um vetor de 100 numero racionais
6
7 v_int[0] = 1; // atribuicao do valor 1 ao primeiro elemento do
8     // do vetor v_int
9
10 vetor_contagem = {0,1,2,3,4,5,6,7,8,9}; // atribuicao de um
11     // bloco completo de
12     // dados ao vetor
13
14 v_char[0] = 'd'; // atribuicao do char 'd' no primeiro espaco de
15     // memoria reservado para a variavel v_char
16
17 v_int{9} = 234567; // atribuicao do valor 234567 ao espaco de
18     // memoria 10 da variavel v_int
19
20 v_float[49] = 0.52 // atribuicao do valor 0.52 ao espaco 50
21     // do vetor v_float

```

Code 4.29: Exemplos de declaração de um vetor em C

Um fato extremamente importante a ser pontuado com relação à indexação de vetores e matrizes é que a contagem sempre começa pelo valor 0. O código acima mostra vários exemplos nos quais tal fato foi pontuado nos comentários.

Vetores e matrizes são muitas vezes utilizados em conjuntos com laços de repetição.

O exemplo do Código 4.30 mostra a utilização de um laço for em conjunto com o um vetor de 10 números inteiros.

```

1
2 //SUPONDO QUE EXISTE UM VETOR De INTEIROS int vetor[10]
3 //JAH COM VALORES SETADOS EM TODOS INDICES
4
5 int i;

```

```

6 int soma = 0;
7 for(i = 0; i < 10; i++)
8 {
9     soma += vetor[i];
10 }
11
12 //A VARIÁVEL soma TEM AGORA A SOMA DE TODOS FATORES

```

Code 4.30: Exemplos de uso de um for com um vetor

O exemplo acima também mostra o acesso de cada índice do vetor.

### 4.3.11.2 Matrizes

Pode-se dizer que matrizes são vetores bidimensionais. Há 2 ou mais dimensões de indexação numa matriz.

```

1 tipo nome_da_variavel [quantidade_de_elementos_dimensao1][
    quantidade_de_elementos_dimensao2]...[quantidade_de_elementos_dimensaoN];

```

Code 4.31: Sintaxe de declaração de uma matriz em C

O Código 4.32 mostra um exemplo de declaração e iniciação de uma matriz 5x3. Nessa matriz há 15 números inteiros organizados em 5 linhas e 3 colunas.

```

1 int alunos_notas [5][3] = { 9,7,9;
2     10,8,6;
3     5,4,5;
4     6,6,6;
5     7,9,8;
6     };

```

Code 4.32: Exemplo de declaração e iniciação de uma matriz de inteiros em C

A primeira atribuição do Código exemplo 4.32 (número 9) é destinada ao elemento (0,0), ou seja,  $alunos\_notas[0][0] = 9$ . Cada vírgula simboliza atribuição à próxima coluna, na linha atual, ou seja,  $alunos\_notas[0][1] = 7$ . Todo ponto e vírgula simboliza "pulo de linha". Após o primeiro ponto e vírgula, tem-se a atribuição do número 10, ou seja,  $alunos\_notas[1][0] = 10$ .

O exemplo 4.33 usa a mesma matriz declarada e iniciada no código 4.32, para calcular a média de 5 alunos em 3 provas e atribuir tal média a um vetor de floats chamado *medias*.

```

1
2 int alunos_notas [5][3] = { 9,7,9;
3 10,8,6;
4 5,4,5;
5 6,6,6;
6 7,9,8;
7 }; //Declaração e atribuição de valores de notas de 3 provas a 5 //alunos
8
9 float medias [5]; //media dos 5 alunos a serem calculadas
10 int i,j; // variáveis a serem usadas como contadores de laços

```

```

11 // e indices de matriz
12
13 for(i = 0; i < 5; i++)
14 {
15     for(j = 0; < 3; j++)
16     {
17         // Calculando a media do aluno i
18         // notar a conversao de variavel
19         medias[i] = (float)alunos_notas[i][j]/3;
20     }
21 }

```

Code 4.33: Exemplo de usod e matriz bidimensional

No exemplo, é executado o comando  $medias[i] = (float)alunos\_notas[i][j]/3$ . A conversão para float, como explicado na seção 4.3.7, é necessária. A primeira razão é que a variável *medias* é uma matriz de floats. Em geral, os compiladores já fazem essa compilação automaticamente. A segunda, e mais relevante para este exemplo, é que a operação  $alunos\_notas[i][j]/3$  resultaria, sem a conversão, num número inteiro, não num número racional, causando erros no cálculo.

### 4.3.12 Tipos avançados de variáveis

Além dos tipos básicos de variáveis (Seção 4.3.6), existem tipos mais complexos e com usabilidade muitas vezes mais úteis na codificação de programas.

Para a plataforma de programação utilizada com Arduino e Galileo são utilizados os tipos mostrados na Tabela 4.10. Os tipos destacados em vermelho são os tipos mais avançados a serem tratados nesta seção.

Datatype	RAM usage
void keyword	N/A
<b>boolean</b>	<b>1 byte</b>
char	1 byte
unsigned char	1 byte
int	2 byte
unsigned int	2 byte
word	2 byte
long	4 byte
unsigned long	4 byte
float	4 byte
double	4 byte
<b>string</b>	<b>1 byte + x</b>
<b>array</b>	<b>1 byte + x</b>
enum	N/A
struct	N/A
pointer	N/A

Tabela 4.10: Tipos básicos e avançados ( destacados em vermelho) de variáveis.

Fonte: Adaptado de <http://playground.arduino.cc/Code/DatatypePractices>

#### 4.3.12.1 boolean

O tipo boolean é um tipo muito simples de dado utilizado para programação em Arduino. Muitas vezes, se deseja saber do estado atual de algo e para isso se pode manipular variáveis do tipo *boolean*. Variáveis do tipo boolean podem armazenar apenas dois valores: *true* ou *false*.

Como a unidade mínima de endereços de variáveis nas arquiteturas atuais é o byte, o qual

possui 8 bits, portanto, o boolean, como mostrado na Tabela 4.10 ocupa 1 byte de memória e não apenas 1 bit, como esperado.

O Código 4.34 mostra um exemplo de uso de uma variável boolean.

```
1 #define pinoBotao 2
2 boolean botaoApertado = false;
3
4 if(digitalRead(pinoBotao) == HIGH)
5 {
6   botaoApertado = true;
7 }
```

Code 4.34: Exemplo de uso de uma variável boolean em Arduino

### 4.3.12.2 Ponteiros

Ponteios, são, como qualquer variável em C, variáveis que armazenam no espaço de memória a ela reservado uma série de bits com algum significado determinado.

A diferença de um ponteiro para uma variável normal é que ele tem como conteúdo, não um número qualquer ou caractere, mas um **endereço de memória**.

A sintaxe para declaração de um ponteiro é mostrado no Código 4.35.

```
1 tipo *nome_do_ponteiro;
```

Code 4.35: Sintaxe de declaração de um ponteiro em C

Exemplos de ponteiros para vários tipos de memória são mostrados no Código 4.36.

```
1 int * ptrInt; // ponteiro para armazenar o endereço de um int
2 char * ptrChar; // ponteiro para armazenar o endereço de um char
3 float *ptrFloat; // ponteiro para armazenar o endereço de um float
4 double *ptrDouble; // ponteiro para armazenar o endereço de um
5 // double
```

Code 4.36: Exemplos de declaração de um ponteiro em C

```
1 int * ptrInt; // ponteiro para armazenar o endereço de um int
2 char * ptrChar; // ponteiro para armazenar o endereço de um char
3 float *ptrFloat; // ponteiro para armazenar o endereço de um float
4 double *ptrDouble; // ponteiro para armazenar o endereço de um
5 // double
6
7
8 char c = 'f';
9 ptrChar = &c; // no ponteiro ptrChar tem agora o endereço da
10 // variavel c
11
12 int v[10] = {5,10,15,3,10,76,5,13,33,45};
13 ptrInt = &v; //ptrInt APONTA para o endereço
14 // inicial do vetor v, cujo
```

```
// conteudo eh igual a 5
```

Code 4.37: Exemplos de atribuição de endereços de variáveis a ponteiros

No Código 4.67, o ponteiro *ptrChar* recebe o endereço de memória da variável *c* e o ponteiro *ptrInt* recebe o endereço do primeiro inteiro do vetor de inteiros *v*.

A Figura 4.66 a alocação de memória para o vetor *v* supondo que o endereço de memória atribuído para *v* se deu a partir de 108.

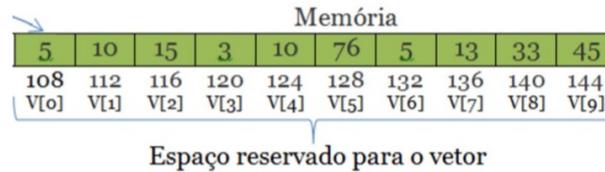


Figura 4.66: Endereço de memória e conteúdo de um vetor *v* de 8 espaços.

Após o comando *ptrInt = &v* se executado, o ponteiro *ptrInt* passou a apontar para o endereço da primeira variável do vetor *v* como mostrado na Figura 4.67.

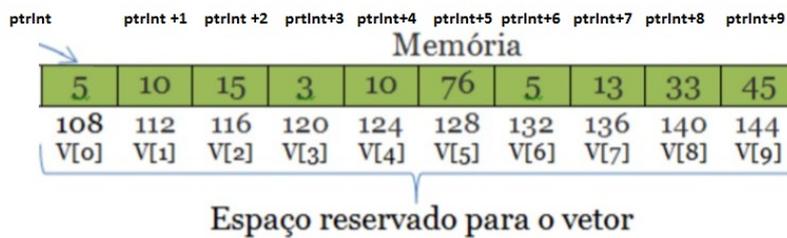


Figura 4.67: Após a atribuição do endereço do vetor ao ponteiro *ptrInt*.

Ponteiros permitem que você se referem ao mesmo espaço na memória de vários locais no programa. Isso significa que se pode atualizar a memória em um local e a mudança pode ser vista em outros escopos.

Ponteiros também podem ser interessantes para economizar espaço por possibilitar o compartilhamento de componentes em estruturas de dados.

Ponteiros também podem ser usados para navegar em matrizes e vetores. Usando o mesmo exemplo do Código 4.36 no Código 4.38, ao ser executar o laço mostrado, o conteúdo do vetor seria 7, 12, 17, 5, 12, 78, 7, 15, 35, 47. O operado *\*ptrInt* refere-se ao conteúdo armazenado no endereço de inteiro apontado por *ptrInt*, assim como *\*(ptrInt + 1)*, se refere do próximo endereço de ponteiro apontado por *ptrInt*.

```
1 int * ptrInt; // ponteiro para armazenar o endereço de um int
2 char * ptrChar; // ponteiro para armazenar o endereço de um char
3 float *ptrFloat; // ponteiro para armazenar o endereço de um float
4 double *ptrDouble; // ponteiro para armazenar o endereço de um
5     // double
6
```

```

7 char c = 'f';
8 prtChar = &c; // no ponteiro prtChar tem agora o endereço da
9 // variável c
10
11 int v[10] = {5,10,15,3,10,76,5,13,33,45};
12 prtInt = &v; //prtInt APONTA para o endereço
13 // inicial do vetor v, cujo
14 // conteúdo eh igual a 5
15
16 int i;
17
18 for(i = 0; i < 10; i++)
19 {
20     *(prtInt + i) = *(prtInt + i) + 2; //adiciona 2 ao i-esimo
21                                     // conteúdo do vetor v
22
23     //comando com efeito igual ao comando anterior
24     //prtInt[i] += 2;
25
26 }

```

Code 4.38: Exemplos de atribuição de endereços de variáveis a ponteiros

### 4.3.12.3 String e Arrays

String e array são nada mais do que vetores de caracteres ou vetores de qualquer tipo de variável (visto que array se define para qualquer tipo de variável).

Esta subseção está escrita após a subseção relativa a ponteiros (Subseção 4.3.12.2) devido ao fato de que o comando relativo a alocação dinâmica de memória ( alocação quando o programa já passou a ser executado) necessitar de conceito de atribuição de endereço de blocos de memória.

String em C são tratadas com o tipo *char\**, ou seja, um ponteiro de char.

O Código 4.39 mostra uma alocação estática de uma string. O conjunto de caracteres "MINHASTRING" é alocado em certo espaço do heap de memória pelo compilador e o ponteiro string aponta para o primeiro caractere do conjunto de caracteres.

```

1
2 char * string = "MINHASTRING";
3
4 //      +-----+
5 //string -> |M|I|N|H|A|S|T|R|I|N|G|  ponteiro string aponta para
6 //      +-----+ esse bloco de memoria
7 //      // no heap de memoria

```

Code 4.39: Exemplos de uma alocação estática de uma string em C

O Código 4.40 mostra uma alocação dinâmica de uma string. Para se alocar dinamicamente um bloco de memória deve-se utilizar o comando *malloc(NUMBYTES)*. O comando malloc reserva,

em tempo de execução, a quantidade de bytes especificados como argumento de entrada e atribui o endereço inicial do bloco ao ponteiro especificado.

```

1
2 char * string = (char *)malloc(11);
3
4 //      +-----+
5 //string -> |X|X|X|X|X|X|X|X|X|X|X|  ponteiro string aponta para
6 //      +-----+ esse bloco de memoria
7 //      // reservado para 11 bytes
8
9 //      +-----+
10 //      |X|X|X|X|X|X|X|X|X|X|X|
11 //      +-----+
12 //
13 //      +-----+
14 //string -> |M|I|N|H|A|S|T|R|I|N|G|  ponteiro string aponta para
15 //      +-----+ esse bloco de memoria, nao
16 //      // o reservado acima

```

Code 4.40: Exemplos de uma alocação dinâmica de uma string em C

O conjunto de caracteres "MINHASTRING", caso atribuído à variável string (*string* = "MINHASTRING" não será atribuído aos 11 bytes alocados, mas para outro bloco, não reservado. É possível que ao tentar acessar o conteúdo da string após o comando *string* = "MINHASTRING" cause um erro em tempo de execução.

A forma correta de atribuição uma string para um ponteiro de char é mostrado no Código 4.41.

```

1
2 char * string = (char *)malloc(11);
3
4 //      +-----+
5 //string -> |X|X|X|X|X|X|X|X|X|X|X|  ponteiro string aponta para
6 //      +-----+ esse bloco de memoria
7 // reservado para 11 bytes
8
9 strcpy(string, "MINHASTRING");
10 //      +-----+
11 //string -> |M|I|N|H|A|S|T|R|I|N|G|
12 //      +-----+

```

Code 4.41: Atribuição de string a um ponteiro de char alocado dinamicamente

Como dito, arrays alocados dinamicamente podem ser criados, utilizando o comando malloc, para qualquer tipo de variável.

O Código 4.43 mostra a alocação dinâmica de um array de 11 floats. O comando sizeof(TIPO) retorna a quantidade de bytes que o tipo de variável especificado por TIPO possui. No caso do exemplo, uma variável float ocupa 4 bytes, portanto, serão reservados 44 bytes de memória e o endereço de tal bloco atribuído ao ponteiro arrayFloats.

```

1 int numFloats = 11;

```

```

2 float * arrayFloats = (float *)malloc(11*sizeof(float));
3
4 //          ++++++
5 //arrayFloats -> |X|X|X|X|X|X|X|X|X|X|X|
6 //          ++++++ esse bloco de memoria
7 // reservado para 11 espaçõs ocupados por floats(4*11 bytes)

```

Code 4.42: Alocação dinâmica de um array de 11 floats

Para se acessar os elementos de um array, pode-se utilizar qualquer uma das sintaxes mostrada no Exemplo 4.44. Caso a operação `arrayFloats++`; seja executada, o ponteiro `arrayFloats` passará a apontar não mais para o primeiro endereço alocado, mas para o segundo. A letra N no comentário mostrado após o comando `arrayFloats++`; simboliza um espaço de memória não alocado com o comando `malloc`.

Abaixo do comando `arrayFloats++`; é mostrado uma série de comandos que podem causar erros em tempo de execução por causa da tentativa de acesso a espaço de memória não reservados.

```

1 int numFloats = 11;
2 float * arrayFloats = (float *)malloc(numFloats*sizeof(float));
3
4 arrayFloats[2] = 7.8;
5 *(arrayFloats + 1) = 0.5;
6
7 //          ++++++
8 //arrayFloats -> |X|0.5|7.8|X|X|X|X|X|X|X|X|X|
9 //          ++++++ esse bloco de //
//                               //memoria
10 // reservado para 11 espaçõs ocupados por floats(4*11 bytes)
11
12 arrayFloats++;
13
14 //          ++++++
15 //arrayFloats -> |0.5|7.8|X|X|X|X|X|X|X|X|N| // arrayFloats
16 //          ++++++//passa a apontar
//                               // para o endereço do
//                               //0.5
17
18
19
20 if(arrayFloats[10] == 0.0)//possível erro em tempo de execucao
21     arrayFloats[10] = 1.0; //

```

Code 4.43: Alocação dinâmica de um array de 11 floats

Para alocar dinamicamente uma matriz, deve-se usar ponteiro de ponteiros e laços de repetição, como mostrado no Código 4.44. Pode-se ver um `int **` (ou um ponteiro de ponteiro de qualquer tipo de variável) como um array de ponteiros de `int`. Cada um desses ponteiros nesse array deve-se ter seu espaço de memória alocado, como mostrado no código.

```

1
2 int **matriz = (int **)malloc(2*sizeof(int *));
3 int i, j;
4

```

```

5 for(i = 0; i <2;i++)
6 {
7   matriz[i] = (int *)malloc(2*sizeof(int));
8 }
9
10 //      +---+
11 //matriz ->|X|X|
12 //      +---+
13 //      |X|X|
14 //      +---+
15
16 for(i = 0; i <2;i++)
17 {
18   for(j = 0; j <2;j++)
19   {
20     matriz[i][j] = i + j;
21   }
22 }
23
24 //      +---+
25 //matriz ->|0|1|
26 //      +---+
27 //      |1|2|
28 //      +---+

```

Code 4.44: Alocação dinâmica de uma matriz 2X2 de inteiros

#### 4.3.12.4 Registros - Struct

Em C, registros são nomeados por *structs*. Chama-se por registro um conjunto de variáveis de tipos quaisquer. Cada variável em um registro é chamado de *campo*.

O exemplo no Código 4.45 mostra a sintaxe de uma struct em C.

```

1 //Definicao de uma struct
2 typedef struct {
3   TIPO1 VAR1;
4   TIPO2 VAR2;
5   .
6   .
7   .
8   TIPON VARN;
9 } NOME_STRUCT;
10
11 //variavel do tipo NOME_STRUCT
12 NOME_STRUCT struct1;
13
14 void setup()
15 {
16   struct1.VAR1 = ...;
17   struct1.VAR2 = ...;

```

```

18 .
19 .
20 .
21 struct1.VARN = ...;
22 }
23
24 void loop()
25 {
26 struct1.VAR1 = ....
27 if(struct1.VAR2....)
28 .
29 .
30 .
31 //Outros procedimentos
32 }

```

Code 4.45: Sintaxe de definição de uma struct

O uso dos símbolos *typedef* não é obrigatório para definição e uso de struct, entretanto, ao utilizá-lo, pode-se declarar uma variável do tipo da estrutura como um novo tipo de dados semelhantes aos tipos básicos (int, char, etc).

O Código 4.46 mostra um exemplo de uma struct que acumula variáveis relevantes para um programa que lide com dados de estudantes, por exemplo.

```

1 //Definicao de uma struct com dados relevantes a um estudante
2 typedef struct {
3 char * nome;
4 int turma;
5 float notaTrabalho1;
6 float notaProva1;
7 float media;
8
9 } estudante;
10
11 //variavel do tipo NOME_STRUCT
12 estudante alunol;
13
14 float calculaMedia(estudante est)
15 {
16 return (est.notaTrabalho1 + est.notaProva1)/2;
17 }
18
19 void setup()
20 {
21 alunol.nome = "Luiz"; // setando a variavel nome (cuidado com //possivel erro de
    memoria nessa setagem de ponteiro de char )
22
23 alunol.turma = 25;
24
25 alunol.notaTrabalho1 = 10;
26 alunol.notaProva1 = 0;

```

```

27 aluno1.media = calculaMedia(aluno1); //chama funcao calculaMedia
28           // para setar o campo media
29
30 Serial.begin(9600);
31 }
32
33 void loop()
34 {
35 Serial.print("Nome Estudante : " );
36 Serial.println(aluno1.nome);
37 Serial.print("Nota Trabalho 1 : " );
38 Serial.println(aluno1.notaTrabalho1);
39 Serial.print("Nota Prova 1 : " );
40 Serial.println(aluno1.notaProva1);
41 Serial.print("Media : " );
42 Serial.println(aluno1.media);
43
44 if(aluno1.media >= 5.0)
45 {
46   Serial.println("Aprovado");
47 }
48 else
49 {
50   Serial.println("Reprovado");
51 }
52
53
54 }

```

Code 4.46: Exemplo de definição e uso de uma struct

Nesse, foi definida uma struct para guardar os dados de um estudante. Os dados são os seguintes :

- Nome
- Turma
- Nota do trabalho 1
- Nota da prova 1

Os campos da struct são acessado, como mostrado, usando um ponto, após o nome da variável.

Como os campos da struct são do tipos básicos de variáveis (int, char, etc) eles podem ser tratadas normalmente como tal.

Struct também podem ser argumentos de entrada ou saída de funções como mostrado no exemplo da função *calculaMedia* do Código 4.46.

Usando a mesma *struct* do Código 4.46, o Código 4.47 mostra como alocar espaço de memória dinamicamente para um ponteiro de struct.

```

1 //Definicao de uma struct com dados relevantes a um estudante
2 typedef struct {
3 char * nome;
4 int turma;
5 float notaTrabalho1;
6 float notaProva1;
7 float media;
8
9 } estudante;
10
11 //variavel do tipo NOME_STRUCT
12 estudante * alunos;
13
14 float calculaMedia(estudante * est, int numEstudante)
15 {
16 return (est[numEstudante]->notaTrabalho1 + est[numEstudante]->notaProva1)/2;
17 }
18
19 void setup()
20 {
21
22 alunos = (estudante *) malloc(2*sizeof(estudante));
23
24 aluno[0]->nome = "Luiz"; // setando a variavel nome (cuidado com //possivel erro de
      memoria nessa setagem de ponteiro de char )
25
26 aluno[0]->turma = 25;
27
28 aluno[0]->notaTrabalho1 = 10;
29 aluno[0]->notaProva1 = 0;
30 aluno[0]->media = calculaMedia(aluno[0]); //chama funcao calculaMedia
31 // para setar o campo media
32
33 Serial.begin(9600);
34 }
35
36 void loop()
37 {
38 Serial.print("Nome Estudante : ");
39 Serial.println(aluno[0]->nome);
40 Serial.print("Nota Trabalho 1 : ");
41 Serial.println(aluno[0]->notaTrabalho1);
42 Serial.print("Nota Prova 1 : ");
43 Serial.println(aluno[0]->notaProva1);
44 Serial.print("Media : ");
45 Serial.println(aluno[0]->media);
46
47 if(aluno[0]->media >= 5.0)
48 {
49 Serial.println("Aprovado");
50 }
51 else

```

```

52 {
53 Serial.println("Reprovado");
54 }
55
56
57 }

```

Code 4.47: Exemplo de definição e uso de uma struct

No Código 4.47, é reservado, para o ponteiro da struct, um espaço para duas variáveis do tipo *estudante* por meio do comando *malloc*.

Para acessar os membros da struct, nesse caso, não se usa ponto, mas o conjunto de caracteres *->*.

### 4.3.13 Conversão Código Binário para Decimal e vice-versa

Todo número segue uma base numérica para representação de números. Toda base numérica é um conjunto de símbolos (ou algarismo).

Representando por  $C$ , o número de símbolos da base  $B$  (por exemplo, na base hexadecimal, o número é 16) e por  $alg_n$  o  $n$ ésimo algarismo de um número representado na base  $B$ , tem-se que todo número, em qualquer base, pode ser convertido da base  $B$  para a base decimal seguindo a seguinte fórmula, sendo  $n$  o número de expoentes necessários para representar a parte inteira do número e  $k$  o número de expoentes necessários para representar a parte fracionária do número:

$$Numero_{baseBparabasedecimal} = alg_n * (C)^n + alg_{n-1} * (C)^{n-1} + \dots + alg_0 * (C)^0 + \dots + alg_{-k} * (C)^{-k} \quad (4.37)$$

A base numérica corriqueiramente usada é a base decimal. A base decimal é formada pelo símbolos 0,1,2,3,4,5,6,7,8,9 e a partir deles podem ser representados todos números. O número de símbolos dessa base é 10, portanto, qualquer número, na base decimal, também segue a mesma lógica apresentada na Equação 4.37.

$$Numero_{basedecimalparabasedecimal} = alg_n * 10^n + alg_{n-1} * 10^{n-1} + \dots + alg_0 * 10^0 + \dots + alg_{-k} * 10^{-k} \quad (4.38)$$

Por exemplo, o número 35,67 na base decimal é igual a :

$$35,67 = 3 * 10^1 + 5 * 10^0 + 6 * 10^{-1} + 7 * 10^{-2} \quad (4.39)$$

A base binária é formada por dois símbolos: 0 e 1. Uma forma usual de se representar um número sem sinal na base binária segue a mesma lógica das expressões mostradas anteriormente:

$$Numero_{basebinriaparabasedecimal} = alg_n * 2^n + alg_{n-1} * 2^{n-1} + \dots + alg_0 * 2^0 + \dots + alg_{-k} * 2^{-k} \quad (4.40)$$

Em sistemas digitais, a base binária é sempre utilizada para representar qualquer número de uma forma compreensível para o sistema.

Para converter um número sem sinal representado na base binária para a base decimal (a qual, usualmente, é mais facilmente compreendida por um ser humano) para a base decimal, basta aplicar a expressão mostrada em 4.40.

Um exemplo seria converter um byte (8 bits), cujo conteúdo é igual B11000010 para decimal:

$$Numero_{B11000010parabasedecimal} = 1 * 2^7 + 1 * 2^6 + 1 * 2^1 = 193 \quad (4.41)$$

Para converter um número na base decimal para a base binária, é necessário realizar a técnica das divisões sucessivas por 2. Nessa técnica, os dígitos do número binário são obtidos realizando sucessivas divisões por 2 ao número na base decimal. A cada vez que a divisão é realizada, deve-se registrar qual foi o resto da divisão. O processo de divisão por 2 e registro do resto continuará até que a parte inteira da divisão seja igual a 0.

O número binário resultante da conversão será igual ao restos da divisão tomados na ordem inversa, ou seja, o último resto é igual ao dígito mais significativo e o primeiro será igual ao dígito menos significativo.

A Figura 4.68 mostra um de conversão da base decimal para a base binária. Nesse exemplo, o número 156 é convertido para binário. O resultado da conversão é o número 10011100<sub>2</sub>.

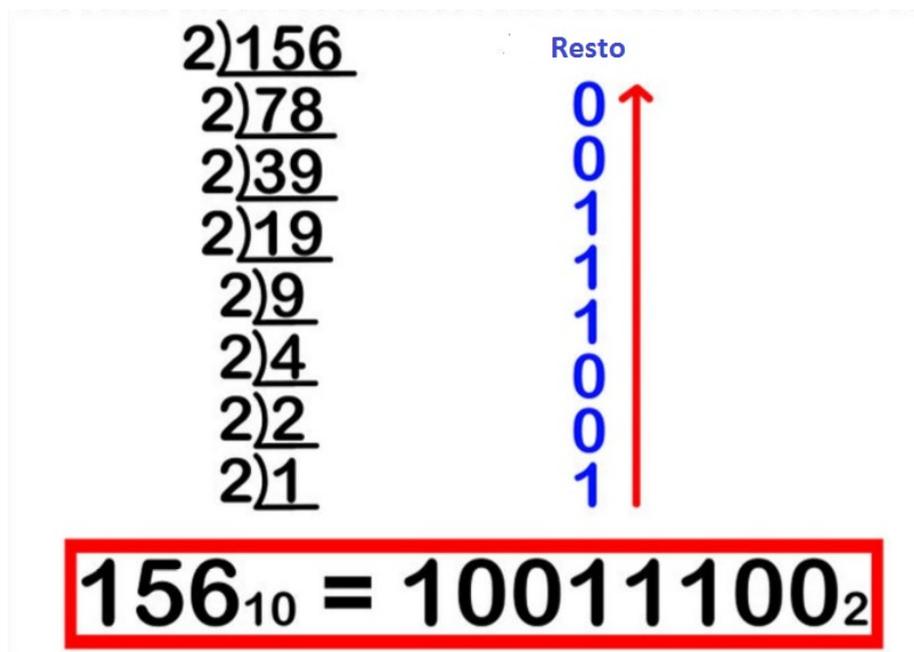


Figura 4.68: Exemplo de aplicação da técnica de divisões sucessivas.

### 4.3.14 Subalgoritmos (Funções)

Funções são um conjunto de instruções definidas por um nome, argumentos de entrada e argumentos de saída as quais podem ser chamadas e executadas num código apenas escrevendo seu nome junto com seus respectivos argumentos de entrada.

A sintaxe para definição de uma função é mostrada no Código 4.48:

```
1
2 TIPO_SAIDA NOME_FUNCAO(TIPOARG1 NOMEARG1, ... , TIPOARGN NOMEARGN)
3 {
4     procedimento1;
5     .
6     .
7     .
8     procediemntoN;
9
10 return varivel //( Do tipo TIPO_SAIDA)
11
12 }
```

Code 4.48: Sintaxe de definição de uma função

Funções são criadas e usadas, principalmente, pelos seguintes motivos:

- Para permitir o reaproveitamento de código já construído.
- Para evitar que um trecho de código que seja repetido várias vezes dentro de um mesmo programa.
- Para permitir a alteração de um trecho de código de uma forma mais rápida. Com o uso de uma função é preciso alterar apenas dentro dela o que se deseja.
- Para que os blocos do programa não fiquem grandes demais e, por consequência, mais difíceis de entender.
- Para facilitar a leitura do programa-fonte de uma forma mais fácil.
- E, principalmente, no paradigma de programação estruturada: para separar o programa em partes(blocos) que possam ser logicamente compreendidos de forma isolada.

Um primeiro exemplo de definição de uma função é mostrado no Código 4.49. Nesse exemplo é definida a função *fatorial*, cujo argumento de entrada é um número inteiro maior ou igual a 0 e saída é o fatorial do argumento de entrada.

```
1
2 /*
3 funcao fatorial
4 Objetivo: Calcular o fatorial de um numero inteiro n
5 Argumentos de entrada - int n
6 Argumento de saida - fatorial do numero n
```

```

7 */
8
9 int fatorial(int n)
10 { int i;
11   int fatorial = 1;
12
13   if(n < 0 )
14   { Serial.println("Argumento de entrada inválido")
15     return -1;
16   }
17   else if(n == 0)
18   {
19     return 1;
20   }
21   for(i = 1; i<=n;i++)
22   {
23     fatorial = fatorial * i;
24   }
25
26   return fatorial;
27 }

```

Code 4.49: Exemplo de definição de uma função

A qualquer momento, durante a execução de uma função, que o comando *return* for encontrado, a função será finalizada e o argumento usado juntamente com o *return* será entregue no contexto que a função foi chamada.

O exemplo mostrado no Código 4.49 mostra 3 situações no qual um comando *return* pode vir a ser executado: No primeiro caso, se entrada *n* for menor que 0, o fatorial não pode ser calculado, então, antes de sair da função é feito um print para avisar no terminal que tal coisa ocorreu. O segundo caso é a possibilidade de a entrada ser igual a 0. Com essa entrada, não se pode fazer o cálculo normal de um fatorial, entretanto a entrada é válida e o resultado é 1.

Caso a entrada seja positiva e maior que 0, o cálculo normal do fatorial é executado e retornado para onde tal função foi executada.

Um primeiro exemplo de definição de uma função a ser aplicada na codificação de uma placa como a placa Galileo é mostrado no Código 4.50. Nesse exemplo 5 aspectos devem ser pontuados:

1. **Protótipos de funções:** A rigor, toda função deve ser declarada antes de poder ser usada, da mesma forma que qualquer variável. Em C, é possível declarar uma função, na forma de protótipo e só definir o seu corpo em um segmento posterior do código. No exemplo 4.50, a função *lerSensor* é prototipada, definindo seu nome, argumentos de entrada e tipo de argumento de saída. Após isso, na função *loop*, ele é chamada e só no final do código ela é definida.
2. **Definição de protótipos de funções:** Toda função prototipada, deve ter seu corpo definido em alguma parte do código. No exemplo 4.50, a função *lerSensor* é definida no final do código.

3. **Escopo de uma variável:** Por escopo de uma variável entende-se o bloco de código onde esta variável é válida. Fora do bloco de código onde a variável foi definida, ela não pode ser acessada. Em C, defini-se um escopo por abrir e fechar chaves ( `{` ).
4. **Variáveis locais:** Variáveis globais são variáveis visíveis apenas no escopo onde ela foi definida. No exemplo do código 4.50, a variável *intsensor* definida dentro da função `lerSensor` é visível apenas dentro dessa função.
5. **Variáveis globais:** Variáveis globais são visíveis por todas funções definidas num código. A variável *intsensor* definida na linha 2 do código 4.50 é, ao contrário da variável *intsensor* definida dentro da função `lerSensor`, uma variável global, ou seja, ela é visível em todos escopos do programa.

O uso de variáveis globais em C deve ser evitado por que isso quebra a modularização do código, trazendo a possibilidade do funcionamento de uma função interferir no funcionamento de outra. É necessário que o programador tenha exata noção do que está fazendo caso escolha usar variáveis globais.

No caso de programação para micro-controladores como a placa Galileo, o uso de variáveis globais não é tão desaconselhável como para uma programa normal em C. Isso se deve ao fato da existência da função `loop`.

Um programa escrito para um micro-controlador é feito para atividade contínua. Caso, por exemplo, a variável *int sensor* fosse definida na linha 11 ( dentro da função `loop`), a cada ciclo do `loop`, a variável `sensor` seria descartada e redefinida, gastando um tempo considerável para aplicações com grandes restrições de tempo de execução.

```
1
2 int lerSensor(int porta); //Prototipo da funcao lerSensor
3 int sensor //Variavel global: pode ser vista, acessada e editada
4     // em qualquer contexto
5
6 void setup{
7
8 }
9
10 void loop{
11     //executada a funcao lerSensor com o argumento int porta = 0
12
13     sensor = lerSensor(0); // atribui a saida dessa funcao
14         // a variavel global sensor
15 }
16
17 //definicao da funcao lerSensor
18 int lerSensor(int porta)
19 {   int sensor; //variavel local, visivel apenas no contexto
20     // da funcao lerSensor
21
22     sensor = analogRead(porta);
```

```
23 return sensor;  
24 }
```

Code 4.50: Exemplo de definição de uma função

### 4.3.15 Interrupção

O fluxo normal de um programa feito para uma plataforma de um microcontrolador é mostrado na Figura 4.57. Como pode ser visto em tal Figura, o programa é iniciado, realiza-se o pré-processamento substituindo as diretivas de compilação, após isso, executa-se a função setup e então a função loop indefinidamente.

Muitas vezes, o fluxo comum de execução de um programa não é capaz de responder a eventos assíncronos externos, causados pelos usuários, ou internos, causados pelo funcionamento do software, rápido o suficiente.

Para resolver parte desses problemas, existem em computação o que se chama de *interrupção de software* e *interrupção de hardware*. As interrupções são sinal, internos ou externos ao processador, que indicam que o processo corrente deve ser posto em espera e outra tarefa (rotina de interrupção), deve ser executada o mais rápido possível.

A Figura 4.69 resume o processo relacionado a uma interrupção.

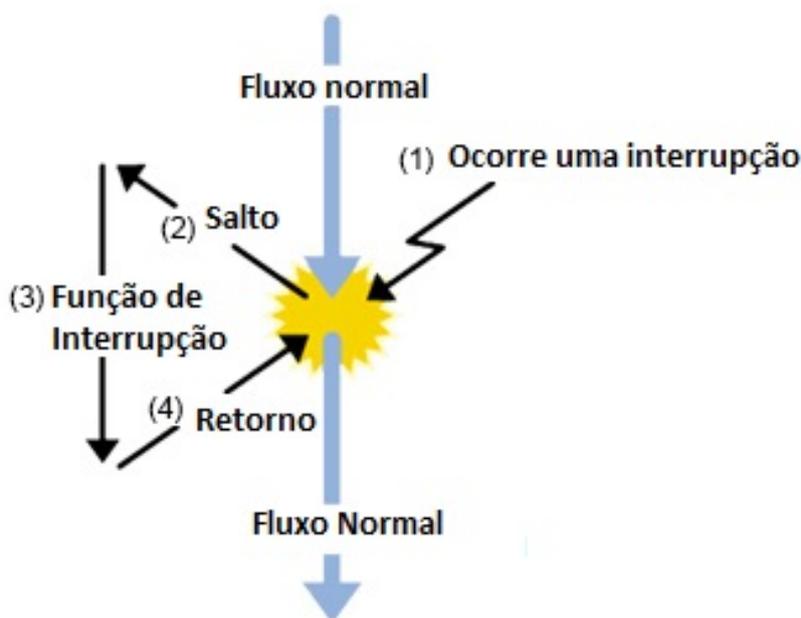


Figura 4.69: Fluxo normal de um programa com uma interrupção.

Fonte: Adaptado de <https://www.contec.com/products/daq/analog/basic.php>

A parte relacionada a **Salto** é tarefa do sistema operacional ou do micro-controlador. Tendo sido identificada a ocorrência de uma interrupção, o endereço de retorno ao fluxo normal é salvo

na pilha de execução e as próximas instruções a serem executadas serão as especificadas na rotina de interrupção.

Após a rotina de interrupção ter sido executada, o endereço de retorno é recolhido da pilha e a execução do programa volta ao fluxo normal.

#### 4.3.15.1 Interrupção de Hardware

Uma interrupção de hardware ocorre por um evento assíncrono. Geralmente, tal interrupção é na forma de algum tipo de uma mudança numa entrada analógica ou digital.

As principais formas de interrupção de hardware são as seguintes:

- **Interrupção de rampa de subida:** A interrupção de rampa de subida é identificada pela mudança do estado lido num pino de 0 Volts (GND ou zero lógico) para 5 Volts (Vcc ou 1 lógico). A Figura 4.71 mostra um circuito capaz de oferecer uma rampa de subida à um pino. Quando a chave é fechada, a tensão lida no pino passa de 0 para 5 V.
- **Interrupção de rampa de descida:** A interrupção de rampa de descida é identificada pela mudança do estado lido num pino de 5 Volts (Vcc ou 1 lógico) para 0 Volts (GND ou zero lógico). A Figura 4.72 mostra um circuito capaz de oferecer uma rampa de descida à um pino. Quando a chave é fechada, a tensão lida no pino passa de 5 para 0 V.

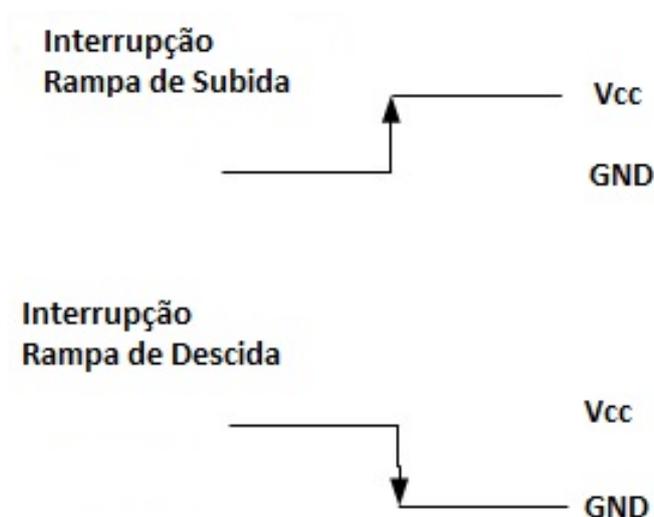


Figura 4.70: Interrupções de rampa de subida e descida.

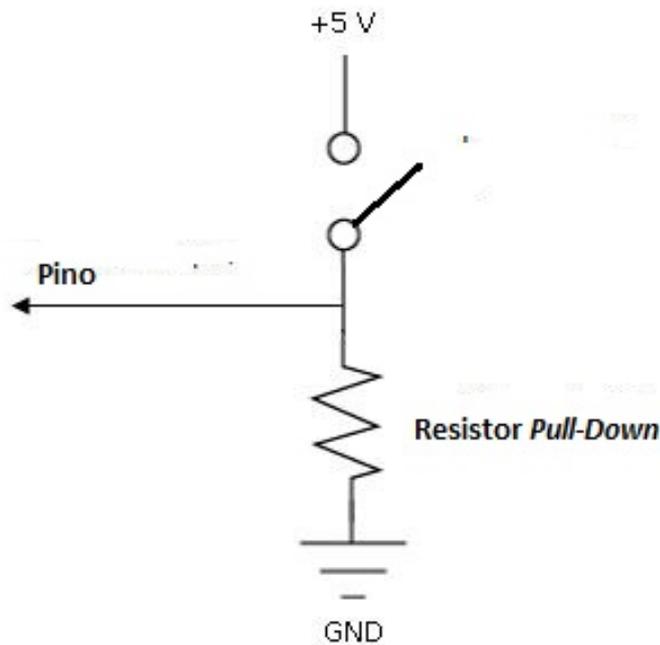


Figura 4.71: Circuito associado à interrupção de rampa de subida.

Fonte: <http://playground.arduino.cc/CommonTopics/PullUpDownResistor>

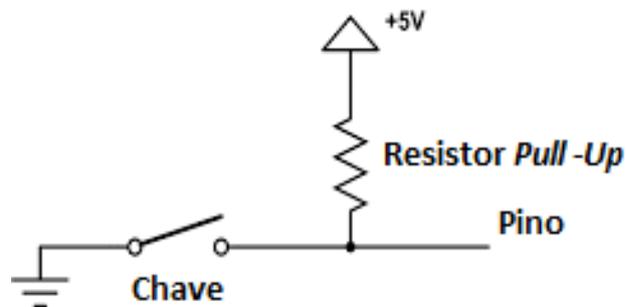


Figura 4.72: Circuito associado à interrupção de rampa de descida.

Além das interrupções de rampa de descida e subida, a plataforma Arduino oferece a identificação de interrupções de estado de qualquer mudança na leitura (CHANGE).

Para a placa Galileo Gen2, apenas as porta digital 2 e 3 podem ser habilitadas para interrupções.

O Código 4.51 mostra, na plataforma Arduino, a sintaxe de definição de uma interrupção de hardware.

1  
2

```

3 void setup()
4 {
5   attachInterrupt(NUM_PINO, ROTINA_INT, TIPO_INT);
6
7 }
8
9
10 TIPO_RETORNO ROTINA_INT (argumentos de entrada)
11 {
12   //instruções da rotina de interrupção;
13 }

```

Code 4.51: Sintaxe para interrupção de hardware

Como mostrado no exemplo do Código 4.51, devem ser fornecidas as seguintes informações para definição da interrupção:

- **NUM\_PINO:** Número do pino digital onde a interrupção é lida  
 Caso seja a porta digital 2, escrever `attachInterrupt(0, ROTINA_INT, TIPO_INT);`  
 Caso seja a porta digital 3, escrever `attachInterrupt(1, ROTINA_INT, TIPO_INT);`
- **ROTINA\_INT:** Função a ser executada quando a interrupção for identificada.
- **TIPO\_INT:**  
 RAMPA DE SUBIDA : escrever "RISING".  
 RAMPA DE DESCIDA : escrever "FALLING".  
 Qualquer mudança: escrever "CHANGE".

O código 4.52 mostra, na plataforma Arduino, um exemplo de uma interrupção de hardware identificada na porta digital 2:

```

1
2 int pino = 13;
3 int estado = LOW;
4 int estado1 = LOW;
5 void setup()
6 {
7   pinMode(pino, OUTPUT);
8   attachInterrupt(0, piscaLed, FALLING);
9 }
10
11 void loop()
12 {
13   digitalWrite(pino, estado);
14
15 }
16
17 //FUNCAO A SER EXECUTADA AO SER IDENTIFICADA UMA RAMPA DE //DESCIDA NA PORTA
    DIGITAL 2(2 -> 0)

```

```

18 void piscaLed ()
19 {
20   if (estado==estado1)
21   {
22     estado = !estado1;
23     //DELAY NECESSARIO PARA EVITAR RUIDOS NA PORTA DIGITAL 2
24     delay (10);
25   }
26   else
27   {
28     //DELAY NECESSARIO PARA EVITAR RUIDOS NA PORTA DIGITAL 2
29     delay (10);
30     estado= estado1;
31   }
32 }

```

Code 4.52: Sintaxe para interrupção de hardware

#### 4.3.15.2 Interrupção de Software - Timer

Interrupção de Software são interrupções disparadas por eventos que ocorrem no software em execução.

Um Timer obedece o mesmo padrão mostrado na Figura 4.69, entretanto, o evento que disparará a interrupção não será um evento de hardware, mas a contagem de pulsos de clock.

*Timers* são contadores de tempo. A cada intervalo prefixado, os *Timers* tem seu contador incrementado, indicando, desde o início da contagem, quantos pulsos de clock foram lidos desde então. Quando o número de pulsos de clock limite for atingido, a rotina de interrupção (*Interruption Service Routine - ISR*) deve ser executada e o contador do Timer será reinicializado.

Para uso de interrupções na plataforma Arduino, é necessário realizar o download da biblioteca *Timer1.h*

Para realizar o download, visite o website <http://playground.arduino.cc/Code/Timer1> como mostrado na Figura 4.73.

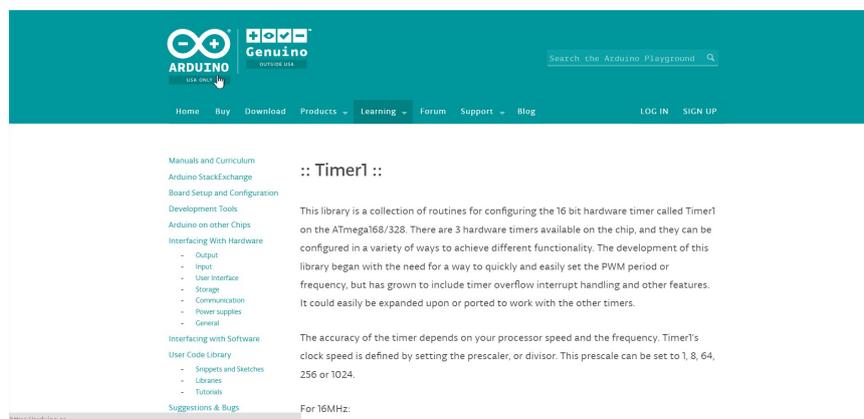


Figura 4.73: Website - <http://playground.arduino.cc/Code/Timer1> .

Clique no link para downloads mostrado na Figura 4.74 e siga as seguintes instruções:

- Descompacte os arquivos do arquivo.zip
- Crie a pasta `Arduino/hardware/libraries/Timer1`
- Copie os arquivos descompactados para a pasta criada

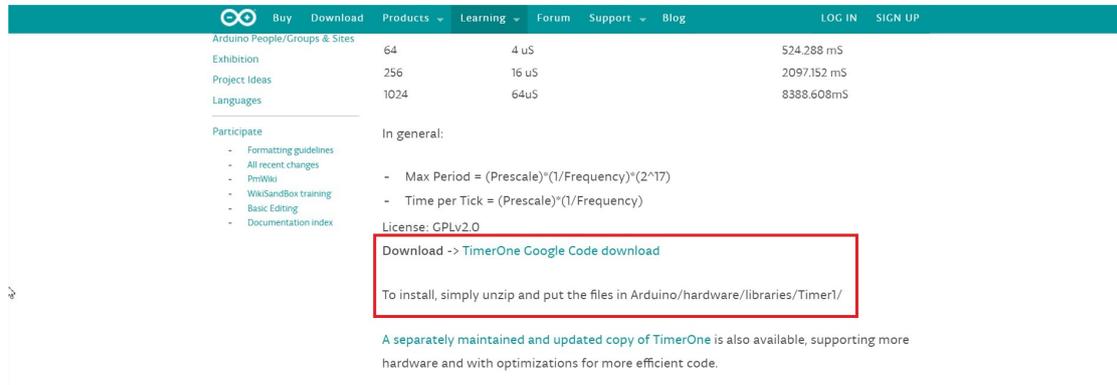


Figura 4.74: Download e instruções para instalação da biblioteca `Timer1`.

Tendo instalado propriamente a biblioteca `Timer1.h`. Para uso de interrupção de timers em Arduino, os seguintes passos devem ser seguidos:

1. Definição do período no qual o timer disparará uma interrupção, executando uma ISR -> função `initialize`
2. Definição da ISR que será executada a cada período especificado
3. Vinculação da ISR ao Timer -> função `attachInterrupt`

O Código 4.53 mostra um exemplo de como utilizar a biblioteca `Timer1`

```
1
2 #include <TimerOne.h> //Inclusao da biblioteca
3
4 void setup()
5 {
6   pinMode(13, OUTPUT); //Inicializando o pino 13 como saida
7
8   Timer1.initialize(100000); // seta o timer para o periodo
9     // 100000 microsegundos (0.1 s - // //10Hz) => a ISR serah executada
10  //vezes por segundo
11
12   Timer1.attachInterrupt( ISR_EXEMPLO, 100000 ); //Vincula a ISR
13     // ISR_EXEMPLO ao timer
14     // de 0.1 segundos a ser executada //todas vez que o timer for
15     //completo (caso fosse escrito //200000 a ISR executada na //segunda vez que o
16     //timer for //completo)
```

```

14 }
15
16 void loop()
17 {
18 //Fluxo normal do programa( sem ser a interrupcao de timer)
19 }
20
21 void ISR_EXEMPLO()
22 {
23 // Pisca LED
24 // o operado ^ eh o XOR, a cada execucao desta ISR, o estado do // LED (acesso ou
    apagado) eh invertifo
25 digitalWrite( 13, digitalRead( 13 ) ^ 1 );
26 }

```

Code 4.53: Sintaxe para interrupção de hardware

As funções da biblioteca *Timer1.h* tomam os parâmetros de tempo em microsegundos, como mostrado no exemplo 4.53.

Nesse exemplo, a cada 100000 microsegundos, ou 0.1 segundos, a função `ISR_EXEMPLO`, que foi vinculada ao timer por meio da função `attachInterrupt()`, é executada, independentemente do que estiver ocorrendo no fluxo normal da função `loop`.

#### 4.3.16 Numero randômico

Muitas vezes, num sistema computacional, faz-se necessário a geração de números aleatórios para definição de parâmetros de execução de um programa.

Número aleatórios têm aplicações em: Jogos de apostas, simulações computacionais, criptografia e outras áreas onde um resultado não predizível se faz necessário.

O principal problema para geração de números aleatórios num sistema computacional é que tais sistemas são, por suas naturezas, completamente determinísticos. Um sistema determinístico não pode, realmente, gerar números aleatórios, pois, para fazê-lo, o sistema seguirá sempre os mesmos passos determinados em seus algoritmos. Dessa dificuldade nasce o termo *número pseudo-aleatório*, o qual se dá a números gerados por sistemas computacional.

Uma maneira, em Arduino, para gerar um número pseudo-aleatório é utilizando o comando `random()`. O Código 4.54 mostra os modos de uso da função `random()`

```

1 int random1 = random(1000); //gera um numero pseudo-aleatoria entre 0 e 999
2 int random2 = random(10,30); //gera um numero pseudo aleatorio entre 10 e 29

```

Code 4.54: Exemplo de geração de um número aleatório usando o comando `random()`

Uma das maneiras de contornar a característica determinística de um algoritmo de geração de número pseudo-aleatório é adicionar um parâmetro determinado por uma característica física ao algoritmo como tempo ou uma leitura analógica de uma variável.

Em Arduino, uma das maneiras de se adicionar ao algoritmo de geração de números aleatórios um parâmetro físico é utilizando o comando `randomSeed()`. O comando `randomSeed()` inicializa o gerador de números aleatórios a depender do valor passado como argumento. O Código 4.55 mostra uma entrada de uma porta analógica em aberto como argumento para o comando `randomSeed()`. Ao se ler uma porta analógica desconectada de um circuito tem-se um valor flutuante determinado apenas por ruídos aleatórios.

```
1 void setup()
2 {
3   Serial.begin(9600);
4   int pinoAnalogicoNaoUtilizado = 3; //Numero de um pino analogico nao utilizado (
      pino em aberto)
5   randomSeed(analogRead(pinoAnalogicoNaoUtilizado)); // geracao de um sequencia de
      numeros a depender de uma variavel aleatoria do pino analogico em //aberto
6 }
7
8 void loop()
9 {
10  int random1 = random(1000); //gera um numero pseudo-aleatoria entre 0 e 999
      utilizando o randomSeed() para inicializar a sequencia de numero aleatorios
11  int random2 = random(10,30); //gera um numero pseudo aleatorio entre 10 e 29
      utilizando o randomSeed() para inicializar a sequencia de numero aleatorios
12
13  delay(20);
14 }
```

Code 4.55: Exemplo de geração de um número aleatório usando o comando `random()` e `randomSeed()`

## 4.3.17 Funções e Bibliotecas Diversas - Arduino

### 4.3.17.1 Função `shiftOut`

A função `shiftOut` tem a seguinte sintaxe:

```
shiftOut(dataPin, clockPin, bitOrder, value);
```

Ela é destinada a transmitir uma série de bits (variável **value**), numa porta selecionada (variável **dataPin**), um bit de cada vez seguindo os pulsos de clock (variável **clockPin**).

A transmissão ocorre em bytes, bit a bit. Tal transmissão pode ocorrer a partir do bit mais significativo ou a partir do bit menos significativo. Para escolher qual das duas opções será escolhida deve-se usar **MSBFIRST** ou **LSBFIRST**, no argumento `bitOrder`, respectivamente.

O código 4.56 mostra um exemplo de uso da função `shiftOut`.

```
1 #define pinoClock 7
2 #define pinoLatch 6
3 #define pinoData 5
4
5 byte numeroOut = 154; // byte = 8 bits
```

```

6
7
8 void setup() {
9   pinMode(pinoLatch, OUTPUT);
10  pinMode(pinoClock, OUTPUT);
11  pinMode(pinoData, OUTPUT);
12 }
13
14 void loop() {
15
16  digitalWrite(pinoLatch, LOW);
17
18  // envia dados nos pinos de saida a partir do bit menos significa
19  // tivo da variavel numeroOut no pinoData
20
21  shiftOut(pinoData, pinoClock, LSBFIRST, numeroOut);
22
23
24  digitalWrite(pinoLatch, HIGH);
25 }

```

Code 4.56: Exemplo de uso da função shiftOut

#### 4.3.17.2 Biblioteca ledcontrol.h

A biblioteca ledcontrol.h deve ser usada em conjunto com os chips MAX7221 e MAX7219 que controlam displays de leds.

Para baixar e instalar a biblioteca ledcontrol.h, os seguintes passos devem ser seguidos:

1. Abra a *General User Interface* (GUI) do Arduino (Figura 4.75).

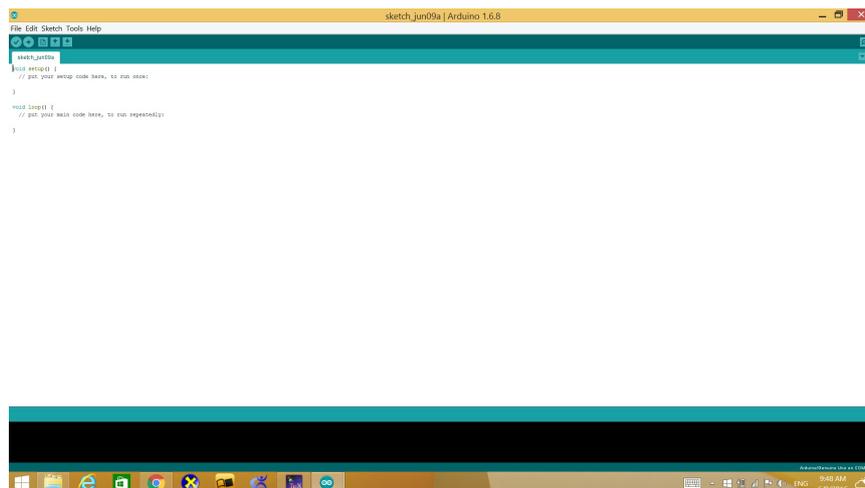


Figura 4.75: GUI do Arduino aberta.

2. Tendo aberto a GUI do Arduino, clique na aba *Manage libraries* que pode ser encontrada

seguindo o seguinte caminho: Sketch/ Include Libraries/Manage Libraries... (Figura 4.76).

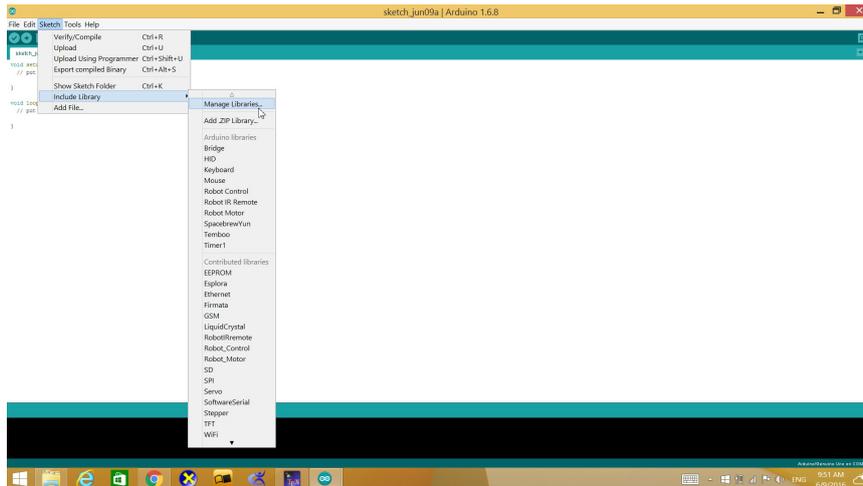


Figura 4.76: Caminho para a aba *Manage Libraries*.

3. Será mostrada a janela *Library Manager* (Figura 4.77).

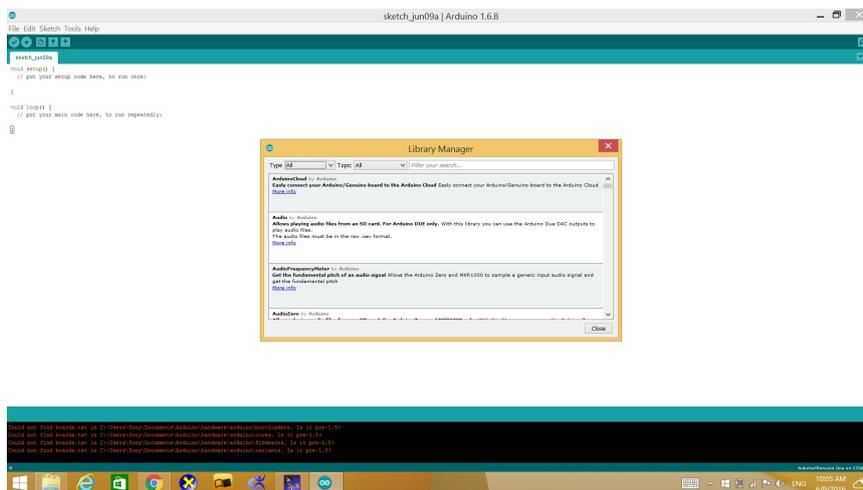


Figura 4.77: Janela *Manage Libraries*.

4. Digite *LedControl* no campo de texto localizado na parte superior (Figura 4.78).

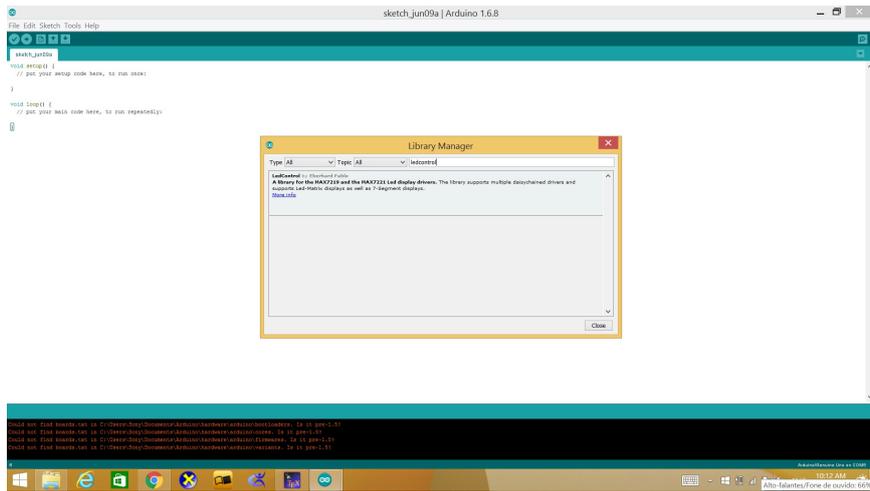


Figura 4.78: Resultado da busca por *LedControl*.

5. Clique no resultado da busca e, após isso, clique em *Install* (Figura 4.79).

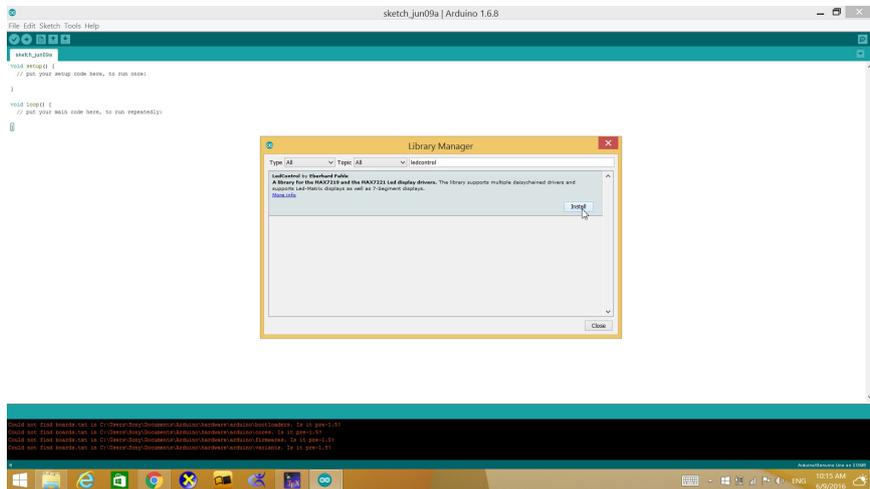


Figura 4.79: Clique no *Install*.

6. Após tais passos, a biblioteca *LedControl.h* estará instalada (Figura 4.80).

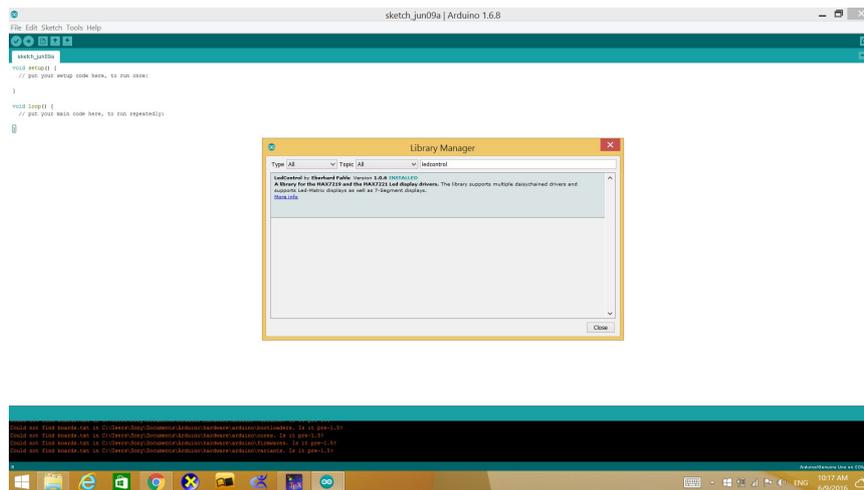


Figura 4.80: Biblioteca LedControl.h instalada.

Para biblioteca LedControl servirá para facilitar controle de uma matriz de LEDs em conjunto com o circuito integrado MAX7219.

O Código 4.57 mostra os passos para o uso da biblioteca. Nesse código, todos led de uma matriz 8x8 são acessos e apagados iterativamente.

```

1
2 #include "LedControl.h" // need the library
3 LedControl lc=LedControl(12,11,10,1); //
4
5 // pino 12 conectado ao pino 1 do MAX7219
6 // pino 11 conectado ao pino 13 CLK
7 // pino 10 conectado ao pino 12 LOAD
8 // O número 1 indica que 1 MAX7219 será usado
9
10 void setup()
11 {
12 // O número 0 como o primeiro argumento de muitas funções
13 // da biblioteca LedControl.h indica o uso de 1 chip MAX7219
14
15 lc.shutdown(0,false); // Desliga o modo de economia de energia, //habilita o
    display
16
17 lc.setIntensity(0,8); // seta a claridade (input entre 0 e 15)
18 lc.clearDisplay(0); // limpa o display
19 }
20 void loop()
21 {
22 for (int linha=0; linha<8; linha++)
23 {
24   for (int coluna=0; coluna<8; coluna++)
25   {
26     lc.setLed(0,coluna ,linha ,true); // liga o LED na //posicao(linha , coluna)
27     delay(25);
28   }
29 }
  
```

```

29 }
30
31 for (int row=0; row<8; row++)
32 {
33     for (int col=0; col<8; col++)
34     {
35         lc.setLed(0,col,row,false); // desliga o LED na
36         //posicao(linha , coluna)
37         delay(25);
38     }
39 }
40 }

```

Code 4.57: Exemplo de uso da biblioteca LedControl.h

# Capítulo 5

## Laboratórios Propostos

### 5.1 Introdução

Neste capítulo, são apresentadas as propostas de prática com a placa de desenvolvimento Intel<sup>®</sup> Galileo. As práticas serão propostas seguindo o modelo apresentado na Seção 2.1.2 de acordo com a proposta de curso apresentada no Capítulo 3 levando em consideração os módulo de masterização para habilidades de programação e eletrônica.

Os circuitos e códigos de solução das práticas propostas encontram-se no Apêndice I.

### 5.2 Prática 1: Começando a usar o *Galileo*

Esta é a primeira prática com a placa Intel<sup>®</sup> Galileo. Nela são descritas os passos iniciais para o desenvolvimento e implementação de soluções.

Os objetivos desta prática são:

- Realizar com sucesso a instalação do driver e da interface de programação para Arduino;
- Criar um programa para a placa Galileo capaz de fazer um LED piscar.

A descrição desta prática é mostrada na Tabela I.1.

Tabela 5.1: Prática 1 - Tabela de descrição.

<b>Nome da prática</b>	Prática 1: Começando a usar a placa Intel Galileo
<b>Objetivos</b>	1) Instalar do driver para uso da placa placa Galileo 2) Instalar IDE do Arduino 3) Realizar a primeira prática utilizando a placa Galileo
<b>Pré-requisitos/ Habilidades masterizadas necessárias</b>	Nenhum
<b>Revisão Teórica - Hardware</b>	Circuito - seção 4.2.2 Resistor - seção 4.2.1 LED - seção 4.2.3
<b>Revisão Teórica - Software</b>	Programação Arduino - seção 4.3.2.1
<b>Material necessário</b>	- 1 Placa Galileo - 1 Transformador 220/120 V ->12 V - 1 cabo USB-Micro-USB - 1 LED
<b>Bibliografia</b>	- Manual de instalação Galileo - Livro ou Website com guia para programação em Arduino (sugestão Getting Started With Galileo)
<b>Habilidades a serem masterizadas com essa prática</b>	1) O aluno é capaz de instalar a placa Galileo no sistema operacional Windows 8 2) O aluno é capaz de escrever um programa simples para Galileo

## 5.3 Prática 2: Introdução a leitura de sensores e tipos de variáveis

Esta prática é destinada, com relação aos tópicos de programação a introduzir os conceitos de diretivas de compilação e tipos de dados.

Os objetivos desta prática são

- Aprender a utilizar uma protoboard;
- Aprender a utilizar um potenciômetro;
- Aprender a utilizar um sensor de luz LDR;
- Criar um circuito de um sensor de luz utilizando LDR, potenciômetro, protoboard e a placa Galileo;
- Aprender os tipos básicos de variáveis de forma a implementar um sensor de luz.

Tabela 5.2: Prática 2 - Tabela de descrição.

<b>Nome da prática</b>	Prática 2: Introdução a leitura de sensores e tipos de variáveis
<b>Objetivos</b>	<p>1) Aprender o básico sobre diretivas de programação</p> <p>2) Aprender os diferentes tipos de variáveis e suas respectivas utilidades</p> <p>3) Aprender a ler o valor numa porta analógica na placa Galileo</p> <p>4) Aprender a utilizar um divisor de tensão para regular um sensor</p>
<b>Pré-requisitos/ Habilidades masterizadas necessárias</b>	Prática 1 - seção 5.2
<b>Revisão Teórica - Hardware</b>	<p>Protoboard - seção 4.2.4</p> <p>Divisor de tensão - seção 4.2.5</p> <p>Potenciômetro - seção 4.2.6</p> <p>LDR - seção 4.2.7</p>
<b>Revisão Teórica - Software</b>	<p>Processo de compilação - seção 4.3.4</p> <p>Diretivas de compilação - seção 4.3.5</p> <p>Tipos básico de variáveis - seção 4.3.6</p> <p>Conversão entre tipos de variáveis - seção 4.3.7</p> <p>Leitura Analógica -</p> <p>Conversão Analógico Digital -seção 4.3.8</p>
<b>Material necessário</b>	<ul style="list-style-type: none"> <li>- 1 Placa Galileo</li> <li>- 1 Transformador 220/120 V -&gt;12 V</li> <li>- 1 cabo USB-Micro-USB</li> <li>- 1 LED</li> <li>- 1 Potenciômetro de 10k ohm</li> <li>- 1 LDR</li> <li>- 1 Protoboard</li> <li>- 5 fios jumper</li> </ul>
<b>Bibliografia</b>	<ul style="list-style-type: none"> <li>- Eletrônica básica [37]</li> <li>- A linguagem de programação C [38]</li> </ul>
<b>Habilidades a serem masterizadas com essa prática</b>	<p>1) O aluno é capaz de construir um circuito simples na protoboard</p> <p>2) O aluno sabe como utilizar um potenciômetro</p> <p>3) O aluno entende o funcionamento de sensores resistivos</p> <p>4) O aluno sabe identificar e utilizar aos tipo variáveis básicas em programação para a linguagem C</p>

## 5.4 Prática 3: Uso de chaves/botões e controladores de fluxo(Condicionais)

Esta prática é destinada, com relação aos tópicos de programação, a introduzir condicionais.

Os objetivos desta prática são:

- Aprender a utilizar botões para enviar sinais digitais para a placa Galileo.
- Aprender o uso de condicionais para controlar um programa.
- Criar um circuito juntamente com a placa Galileo capaz de acender 4 LED's utilizando um botão para enviar um sinal para a placa de forma que o seguinte padrão seja seguido:
  - No começo da execução do código, os 4 LEDs estarão piscando conjuntamente.
  - Ao se apertar o botão, apenas o primeiro LED piscará, enquanto os outros estarão apagados.
  - Após isso, caso o botão seja apertado, apenas o segundo LED piscará, enquanto os outros estarão apagados.
  - O mesmo padrão seguirá até apenas o quarto LED piscar.
  - Caso, enquanto o último LED estiver piscando, o botão seja apertado, todos os LED voltam a piscar conjuntamente.
  - Caso o botão seja apertado, o padrão supracitado se repetirá.

Tabela 5.3: Prática 3 - Tabela de descrição.

<b>Nome da prática</b>	Prática 3: Uso de chaves/botões e controladores de fluxo(Condicionais)
<b>Objetivos</b>	<p>1) Aprender a utilizar botões para enviar sinais digitais para a placa Galileo.</p> <p>2) Aprender o uso de condicionais para controlar um programa.</p> <p>3) Construir um circuito juntamente com um programa que realize o padrão citado no início da descrição desta prática</p>
<b>Pré-requisitos/ Habilidades masterizadas necessárias</b>	Práticas 1 seção 5.2 e 2 - seção 5.3
<b>Revisão Téorica - Hardware</b>	Interruptores - seção 4.2.8
<b>Revisão Téorica - Software</b>	Estruturas Condicionais - seção 4.3.9
<b>Material necessário</b>	<ul style="list-style-type: none"> <li>- 1 Placa Galileo</li> <li>- 1 Transformador 220/120 V -&gt;12 V</li> <li>- 1 cabo USB-Micro-USB</li> <li>- 4 LEDs</li> <li>- 4 resistores 220 ohm</li> <li>- 1 resistor 10k</li> <li>- 1 botao de pressão 4 pinos</li> <li>- 14 fios jumper</li> </ul>
<b>Bibliografia</b>	<ul style="list-style-type: none"> <li>- Eletrônica básica [37]</li> <li>- A linguagem de programação C [38]</li> </ul>
<b>Habilidades a serem masterizadas com essa prática</b>	<p>1) O aluno é capaz de utilizar botões para enviar sinais digitais para a placa Galileo</p> <p>2) O aluno é capaz de escrever trabalhar com condicioanais</p>

## 5.5 Prática 4: Uso de laços de repetição

Esta prática é destinada, com relação aos tópicos de programação, ao uso de laços de repetição.

Os objetivos desta prática são:

- Criar um circuito juntamente com a placa Galileo com 4 LED's um botão e um potenciômetro;
- Utilizando os componentes supracitados, devem-se ser criados 3 comportamentos distintos no acender e apagar dos LEDs;
- Usar o potenciômetro como divisor de tensão( conectar o 5V da placa na pino da direita do potenciômetro, GND na pino da esquerda e a pino centra na entrada analógica A0);
- Utilizar o botão para selecionar o comportamento a ser executado;
- **Comportamento 1:** Piscar 1 e apenas um dos 4 LEDs ao depender do valor de tensão lido na porta A0. Os outros 3 LEDs devem permanecer apagado;
- **Comportamento 2:** Piscar os LEDs sequencialmente( pisca LED 1, outros apagados; pisca LED 2, outros apagados; pisca LED3, outros apagados), com velocidade de piscar dependente do valor lido no potenciômetro;
- **Comportamento 3:** Comportamento a ser escolhido pelo grupo. Esse comportamento deve, obrigatoriamente, ser distinto dos dois primeiros comportamentos citados e diferente dos comportamentos criados por outros grupos.

Tabela 5.4: Prática 4 - Tabela de Descrição

<b>Nome da prática</b>	Prática 4: Uso de laços de repetição
<b>Objetivos</b>	1) Aprender a utilizar os diferentes tipos de laço de repetição juntamente com estruturas condicionais apropriada
<b>Pré-requisitos/ Habilidades masterizadas necessárias</b>	Prática 3 - seção ref
<b>Revisão Teórica - Hardware</b>	Potenciômetro - seção ref Interruptores - seção 4.2.8
<b>Revisão Teórica - Software</b>	Laços de repetição - seção 4.3.10
<b>Material necessário</b>	- 1 Placa Galileo - 1 Transformador 220/120 V ->12 V - 1 cabo USB-Micro-USB - 4 LEDs - 4 resistores 220 ohm - 1 resistor 10k - 1 potenciometro - 1 botão de pressão 4 pinos - 17 fios jumper
<b>Bibliografia</b>	-Eletrônica básica [37] -A linguagem de programação C [38]
<b>Habilidades a serem masterizadas com essa prática</b>	1) O aluno é capaz de usar laços de repetição  2) O aluno é capaz de construir algoritmos apropriados para a execução de tarefas determinadas utilizando condicionais e laços de repetição

## 5.6 Prática 5: Uso de vetores, shift register e tipos variados de dados

Esta prática é destinada, com relação aos tópicos de programação, ao uso de vetores.

Os objetivos desta prática são:

### Objetivo 1:

- Criar um circuito juntamente com a placa Galileo com 8 LED's, 8 resistores e um registrador de deslocamento (Shift Register).
- Utilizando apenas 3 pinos digitais, o circuito deve ser capaz de acender os 8 LED's.
- Para tanto, deve ser utilizado um registrador de deslocamento com pinos propriamente conectados.
- Deve ser utilizado um vetor de 8 inteiros para simbolizar um byte que será transmitido pelo registrador de deslocamento. O número simbolizado pelo vetor representa, na forma binária, quais LED's estarão acessos e quais estarão apagados. O número binário resultante do vetor deve ser convertido num número decimal para se propriamente transmitido pela função shiftOut.
- Quais LEDs estarão acessos ou apagados será uma escolha da equipe, entretanto não será permitido a escolha de todos acessos ou todos apagados.

### Objetivo 2:

- Utilizando o mesmo circuito criado para o objetivo 1, faça um código com o qual os LEDs realizem uma contagem binária. No começo da contagem, todos LEDs estarão apagados, e, no final da contagem que se dará no número 255 ( $255 = 2^8 - 1$ ), todos os LEDs estarão acessos.
- Todos os números da contagem devem ser mostrado no monitor serial, na base decimal e na base binária.

Tabela 5.5: Prática 5 - Tabela de Descrição

<b>Nome da prática</b>	Prática 5: Uso de vetores, loops, e registradores de deslocamento (Conversão série-paralelo)
<b>Objetivos</b>	1) Aprender a utilizar registradores de deslocamento 2) Aprender a manipular vetores 3) Aprender a converter um número binário sem sinal para decimal
<b>Pré-requisitos/ Habilidades masterizadas necessárias</b>	Prática 4 - seção 5.5
<b>Revisão Teórica - Hardware</b>	Registrador de deslocamento - seção 4.2.9
<b>Revisão Teórica - Software</b>	Laços de repetição - seção 4.3.10 Vetores - seção 4.3.11 Função Especial - shiftOut - seção 4.3.17.1
<b>Material necessário</b>	- 1 Placa Galileo - 1 Transformador 220/120 V ->12 V - 1 cabo USB-Micro-USB - 8 LEDs - 8 resistores 220 ohm - 1 Registrador de Deslocamento 74HC595 - 25 Jumpers
<b>Bibliografia</b>	-Eletrônica básica [37] -A linguagem de programação C [38]
<b>Habilidades a serem masterizadas com essa prática</b>	1) O aluno é capaz de usar vetores em conjunto com laços de repetição de forma apropriada  2) O aluno é capaz de construir circuitos mais complexos utilizando registradores de deslocamento para conversão serial - paralelo.

## 5.7 Prática 6: Uso de funções e sensor de temperatura

Esta prática é destinada, com relação aos tópicos de programação, ao uso de funções.

Os objetivos desta prática são:

- Criar um circuito juntamente com a placa Galileo com 3 LED's e 1 sensor de temperatura LM35.
- Deverão ser definidas 3 regiões de temperatura: Temperatura Normal, Temperatura Alta e Temperatura Baixa.
- Em cada uma dessas regiões, um LED em específico deve estar acesso.
- A temperatura lida deve ser mostrada continuamente no monitor serial.
- Nas funções setup e loop, deve haver apenas chamadas a funções que cumprem todos esses requisitos.

Tabela 5.6: Prática 6 - Tabela de Descrição

<b>Nome da prática</b>	Prática 5: Uso de funções e sensor de temperatura
<b>Objetivos</b>	<ol style="list-style-type: none"> <li>1) Aprender a modularizar uma situação problema</li> <li>2) Criar funções que responsáveis por resolver os problemas identificados</li> <li>3) Calcular propriamente a temperatura utilizando o sensor LM35</li> </ol>
<b>Pré-requisitos/ Habilidades masterizadas necessárias</b>	Prática 5 - seção 5.6
<b>Revisão Teórica - Hardware</b>	Sensor de Temperatura LM35 - seção 4.2.10
<b>Revisão Teórica - Software</b>	Subalgoritmos (Funções) - seção 4.3.14
<b>Material necessário</b>	<ul style="list-style-type: none"> <li>- 1 Placa Galileo</li> <li>- 1 Transformador 220/120 V -&gt;12 V</li> <li>- 1 cabo USB-Micro-USB</li> <li>- 3 LEDs</li> <li>- 3 resistores 220 ohm</li> <li>- 1 LM35</li> <li>- 8 Jumpers</li> </ul>
<b>Bibliografia</b>	<ul style="list-style-type: none"> <li>-Eletrônica básica [37]</li> <li>-A linguagem de programação C [38]</li> </ul>
<b>Habilidades a serem masterizadas com essa prática</b>	<ol style="list-style-type: none"> <li>1) O aluno é capaz de identificar um situação problema e dividi-lá em módulos.</li> <li>2) O aluno é capaz de criar funções</li> <li>3) O aluno é capaz de construir um circuito sensor de temperatura, processando apropriadamente o sinal de tensão enviado pelo sensor</li> </ol>

## 5.8 Prática 7: Uso de structs, interrupção de hardware e circuito de debounce de sinais

Esta prática é destinada, com relação aos tópicos de programação, ao uso de de structs e interrupções de hardware.

Os objetivos desta prática são:

- Criar um circuito para implementar um pequeno jogo;
- O jogo consistirá numa disputa entre dois jogadores para descobrir qual dos dois é mais rápido em apertar botão;
- Para ler os botões, devem ser implementadas rotinas de interrupção de hardware (Seção 4.3.15.1);
- Os sinais dos botões deve passar pelo processo de *debouncing* (seção 4.2.14 ) para uma contagem válida da quantidade de rampas de subida;
- Deve ser criada uma *struct* para armazenar: O nome do jogador, quantidade de vezes que ele apertou o botão e a velocidade de apertar o botão e uma frase a ser imprimida no monitor serial em caso de vitória;
- O tempo de disputa é de 5000 ms. Nesse tempo, a placa Galileo deve contar quantas vezes os jogadores apertaram seus respectivos botões;
- Ao final dos 5000 ms, a placa Galileo deve imprimir no monitor serial: as informações dos dois jogadores, a quantidade de vezes que cada um apertou seu respectivo botão e sua velocidade;
- Após imprimir as informações dos jogadores, deve ser impresso o nome do vencedor e sua respectiva frase de vitória;
- Caso ocorra um empate, devem ser impressos os dois nomes e as duas frases de vitória;
- Para cada um dos jogadores, deve existir um LED pronto para piscar. Após imprimir a frase de vitória, um LED deve piscar exatamente a quantidade de vezes que o vencedor apertou o botão;
- Caso ocorra um empate, os dois LED devem piscar conjuntamente.

A Tabela 5.7 mostra a descrição da prática 7.

Tabela 5.7: Prática 7 - Tabela de Descrição

<b>Nome da prática</b>	Prática 7: Uso de structs, interrupção de hardware e circuito de debounce de sinais
<b>Objetivos</b>	1) Aprender a utilizar interrupções de hardware 2) Aprender a utilizar um filtro RC para realizar debouncing de um sinal 3) Aprender a criar e utilizar struct
<b>Pré-requisitos/ Habilidades masterizadas necessárias</b>	- pratica 6 seção 5.7
<b>Revisão Teórica - Hardware</b>	- Capacitores seção 4.2.11 - Filtro RC seção 4.2.12 - Inversor Schmitt trigger seção 4.2.13 - Debouncing seção 4.2.14
<b>Revisão Teórica - Software</b>	-Interrupção seção 4.3.15
<b>Material necessário</b>	- 1 Placa Galileo - 1 Transformador 220/120 V ->12 V - 1 cabo USB-Micro-USB - 1 Protoboard - 2 LED - 2 Resistores 220 Ohm - 2 Resistores 10k Ohm - 2 Capacitores 10u F - 1 chip Schmitt trigger SN74HC14
<b>Bibliografia</b>	-Eletrônica Básica [37] -A linguagem de programação C [38]
<b>Habilidades a serem masterizadas com essa prática</b>	1) O aluno é capaz de criar filtros simples de sinais de interruptores para evitar o bouncing de sinais de origem mecânica 2) O aluno é capaz de escrever rotinas de interrupção (ISR) de acordo com interrupções de hardware 3) O aluno é capaz de transcrever regras de negócios de jogos em algoritmos de programação utilizando estruturas condicionais e laços de repetição

## 5.9 Prática 8: Prática final

Esta prática é prática final do curso. Todos conhecimento de programação visto neste curso podem fazer parte da solução do problema proposto.

Os objetivos desta prática são:

- Criar um circuito para implementar um jogo *Snake Game*
- O jogo deve ser implementado sobre uma matriz de led 8x8
- Para controlar a cobra, deve ser utilizado um joystick ou 4 botões
- Não é necessária a implementação de rotinas de interrupção para esta prática
- A cobra deve inicialmente ocupar 3 LEDs
- A posição da presa deve ser aleatória e não deve ocupar nenhuma parte do corpo da cobra
- Toda vez que a presa for alcançada pela cobra, o corpo da cobra deve aumentar em 1 LED.

A Tabela 5.8 mostra a descrição da prática 8.

Tabela 5.8: Prática 8 - Tabela de Descrição

<b>Nome da prática</b>	Prática 8: Prática final
<b>Objetivos</b>	1) Utilizar todos conhecimentos aprendidos para criar um jogo Snake Game utilizando uma matriz de led 8x8 e um joystick ou 4 botões
<b>Pré-requisitos/ Habilidades masterizadas necessárias</b>	- pratica 7 seção 5.9
<b>Revisão Teórica - Hardware</b>	- Matriz de Led seção 4.2.16 - Driver de matriz de led seção 4.2.16 - Potenciômetro (joystick) seção 4.2.6 - Interruptores seção 4.2.8
<b>Revisão Teórica - Software</b>	- Todas seções
<b>Material necessário</b>	- 1 Placa Galileo - 1 Transformador 220/120 V ->12 V - 1 cabo USB-Micro-USB - 1 Protoboard - 1 Joystick ou 4 botões - 4 Resistores 10k Ohm (se 4 botões forem utilizados) - 1 matriz de led 8x8 - 1 74HC595
<b>Bibliografia</b>	-Eletrônica Básica [37] -A linguagem de programação C [38]
<b>Habilidades a serem masterizadas com essa prática</b>	1) O aluno de unir conhecimentos pontuais de programação e eletrônica para construir projetos razoavelmente complexos.

## Capítulo 6

# Conclusões

Este trabalho teve por objetivo trazer uma proposta de metodologia de curso para a disciplina *Algoritmos e Programação de Computadores* expandindo o conteúdo ensinado para, além de computação básica, também eletrônica básica. A proposta de expansão para eletrônica básica foi feita baseando-se na placa de desenvolvimento de circuitos Intel<sup>®</sup> Galileu. A placa Galileo foi escolhida por causa da sua facilidade de programação na linguagem de programação C e facilidade de prototipagem de circuitos.

Além da revisão do conteúdo programático, foi proposta também uma mudança do paradigma pedagógico do curso. Foi proposto para o curso, a aplicação das metodologias educacionais de aprendizagem por masterização. Já com relação ao ensino, foi proposto o uso da metodologia de gerenciamento de projetos *Scrum* adaptado ao contexto de sala de aula. Tais propostas se basearam nas pesquisas relatadas sobre a qualidade de metodologias de aprendizagem ativa e sua relevância para os paradigmas do século XXI.

Para os objetivos de masterização identificados para a disciplina, foram planejadas 8 práticas com a placa Galileo com nível de complexidade crescente. Todas práticas apresentam uma lista de: materiais necessários, habilidades masterizadas relativas a eletrônica e relativas a programação necessárias, conteúdos de programação e eletrônica novos abordados pela prática, objetivos de masterização e fontes bibliográficas.

Como não foi possível verificar a efetividade do curso planejado neste trabalho, portanto, pode-se, como trabalho futuro, aplicar o plano de curso desenvolvido neste trabalho. Além disso, pode-se ter como trabalho futuro a expansão dos conteúdos ensinados com a placa Intel Intel<sup>®</sup> Galileu para disciplinas como *Estruturas de dados* sob a metodologia de masterização de habilidades.

# REFERÊNCIAS BIBLIOGRÁFICAS

- [1] HUITT, W. Bloom et al.'s taxonomy of the cognitive domain. *Educational psychology interactive*, v. 22, 2004.
- [2] S. BURFOOT J., G. D. e. H. C. B. *A Teacher's Guide to Intel Galileo*. Buildind C5B, Macquarie University, North Ryde, NSW, 2109, 2015.
- [3] NXP. *DATASHEET 74HC14*. [S.l.], 2015.
- [4] MELNIK, G. *Agile 2008 August 4-8, 2008, Toronto, Ontario, Canada*. Los Alamitos, Calif: IEEE.Computer Society, 2008. ISBN 978-0-7695-3321-6.
- [5] SOVIC, A.; JAGUST, T.; SERSIC, D. How to teach basic university-level programming concepts to first graders? In: *Integrated STEM Education Conference (ISEC), 2014 IEEE*. [S.l.: s.n.], 2014. p. 1–6.
- [6] COTO, M.; MORA, S.; ALFARO, G. Giving more autonomy to computer engineering students: Are we ready? In: *IEEE Global Engineering Education Conference, EDUCON 2013, Berlin, Germany, March 13-15, 2013*. [s.n.], 2013. p. 618–626. Disponível em: <<http://dx.doi.org/10.1109/EduCon.2013.6530170>>.
- [7] CELETI, F. R. Origem da educação obrigatória: Um olhar sobre a prússia. *Revista Saber Acadêmico*, v. 1, n. 1, p. 29–33, June 2012.
- [8] OLIVER, J.; TOLEDO, R. On the use of robots in a pbl in the first year of computer science / computer engineering studies. In: *Global Engineering Education Conference (EDUCON), 2012 IEEE*. [S.l.: s.n.], 2012. p. 1–6. ISSN 2165-9559.
- [9] F. de O. V. Crescimento, evolução e o futuro dos cursos de engenharia. *Revista de Ensino de Engenharia*, v. 24, n. 2, p. 3–12, December 2005.
- [10] R., W. *Alta taxa de desistência na universidade causa déficit de engenheiros*. Setembro 2013. [Online; posted 4-Setembro-2013].
- [11] LAHTINEN, E.; ALA-MUTKA, K.; JÄRVINEN, H.-M. A study of the difficulties of novice programmers. *SIGCSE Bull.*, ACM, New York, NY, USA, v. 37, n. 3, p. 14–18, jun. 2005. ISSN 0097-8418. Disponível em: <<http://doi.acm.org/10.1145/1151954.1067453>>.

- [12] ESCUDERO, M. R.; HIERRO, C. M.; PABLO, A. Pérez de Madrid y. Using arduino to enhance computer programming courses in science and engineering. In: *EDULEARN13 Proceedings*. [S.l.]: IATED, 2013. (5th International Conference on Education and New Learning Technologies), p. 5127–5133. ISBN 978-84-616-3822-2. ISSN 2340-1117.
- [13] PRINCE, M. Does active learning work? a review of the research. *Journal of Engineering Education*, Blackwell Publishing Ltd, v. 93, n. 3, p. 223–231, 2004. ISSN 2168-9830. Disponível em: <<http://dx.doi.org/10.1002/j.2168-9830.2004.tb00809.x>>.
- [14] M.FELDER DONALD R.WOODS, J. E. A. R. R. The future of engineering education ii.teaching methods that work. *Chem. Engr Education*, v. 1, n. 34, p. 26–39, September 2000.
- [15] FEISEL, L. D.; ROSA, A. J. The role of the laboratory in undergraduate engineering education. *Journal of Engineering Education*, v. 94, p. 121–130, 2005.
- [16] LYE, S. Y.; KOH, J. H. L. Review on teaching and learning of computational thinking through programming: What is next for k-12? *Computers in Human Behavior*, v. 41, p. 51 – 61, 2014. ISSN 0747-5632. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0747563214004634>>.
- [17] PHAM, A. *Scrum em Acao: Gerenciamento e Desenvolvimento agil de Projetos de Software*. [S.l.]: NOVATEC, 2011. ISBN 8575222856.
- [18] KALISH, S.; MAHAJAN, V.; MULLER, E. Waterfall and sprinkler new-product strategies in competitive global markets. *international Journal of research in Marketing*, Elsevier, v. 12, n. 2, p. 105–119, 1995.
- [19] EDUCATION, T. H. [https://www.timeshighereducation.com/world-university-rankings\\_year=2016\\_url=https://www.timeshighereducation.com/news/ranking-methodology-2016\\_urldate=2016-04-02](https://www.timeshighereducation.com/world-university-rankings_year=2016_url=https://www.timeshighereducation.com/news/ranking-methodology-2016_urldate=2016-04-02).
- [20] TUCKER, P. *Linking teacher evaluation and student learning*. Alexandria, VA: Association for Supervision and Curriculum Development, 2005. ISBN 1-4166-0032-9.
- [21] C., D. Defining a 21st century education. *The Center for Public Education*, v. 1, n. 1, p. 1–79, July 2009.
- [22] GOEL, S. et al. Collaborative teaching in large classes of computer science courses. In: *Eighth International Conference on Contemporary Computing, IC3 2015, Noida, India, August 20-22, 2015*. [s.n.], 2015. p. 397–403. Disponível em: <<http://dx.doi.org/10.1109/IC3.2015.7346714>>.
- [23] K.R, S. Competências do século 21. *Revista Pesquisa e Debate em Educação*, v. 4, n. 2, p. 15–30, August 2014.
- [24] INTEL. *DataSheet Intel Galileo Gen 2 Development Board*. [S.l.], 2014.
- [25] SEDRA., A. S.; SMITH, K. C. *Microeletronics Cicuits*. [S.l.]: Oxford University Press, 2004.

- [26] DEVICES, A. *DATASHEET AD7298*. One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A., 2011.
- [27] BAKER, R. J. *CMOS Circuit Design, Layout, and Simulation, 3rd Edition (IEEE Press Series on Microelectronic Systems)*. [S.l.]: Wiley-IEEE Press, 2010.
- [28] HIMPE, V. *Mastering the I<sup>2</sup>C bus*. Susteren: Elektor International Media, 2011. ISBN 978-0-905705-98-9.
- [29] RUSSELL, R. C. J. *Serial peripheral interface bus*. Place of publication not identified: Book On Demand Ltd, 2012. ISBN 5513504936.
- [30] OSBORNE, A. *An introduction to microcomputers*. Berkeley, Calif: Osborne/McGraw-Hill, 1980. ISBN 0-931988-34-9.
- [31] HENNESSY, J. *Computer architecture : a quantitative approach*. Waltham, MA: Morgan Kaufmann, 2012. ISBN 978-0-12-383872-8.
- [32] ISHIBASHI, K. *Low power and reliable SRAM memory cell and array design*. Berlin New York: Springer, 2011. ISBN 978-3-642-19567-9.
- [33] RECKTENWALD, G. W.; HALL, D. E. Using arduino as a platform for programming, design and measurement in a freshman engineering course. In: *2011 Annual Conference & Exposition*. Vancouver, BC: ASEE Conferences, 2011. <https://peer.asee.org/18720>.
- [34] SOUZA, M. A. M.; DUARTE, J. R. R. Low-cost educational robotics applied to physics teaching in brazil. *Physics Education*, v. 50, n. 4, p. 482, 2015. Disponível em: <<http://stacks.iop.org/0031-9120/50/i=4/a=482>>.
- [35] DEVICES, A. *DATASHEET 74HC595*. One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A., 2011.
- [36] DEVICES, A. *DATASHEET LM35*. One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A., 2009.
- [37] BASIC electronics. New York: Dover Publications, 1973. ISBN 978-0486210766.
- [38] KERNIGHAN, B. *The C programming language*. Englewood Cliffs, N.J: Prentice Hall, 1988. ISBN 978-0131103627.

# APÊNDICES

# I. APÊNDICES

Estes apêndices, como explicado no capítulo 5, são destinados a mostrar circuitos e códigos soluções das práticas propostas.

## I.1 Prática 1: Começando a usar o *Galileo*

### I.1.1 Procedimentos

Para a primeira prática, é feito um programa em Arduino para fazer um LED piscar. Como citado na Seção 4.2.3, para se acender um LED, é necessário submeter o seu ânodo ao nível alto de tensão e seu cátodo à um nível baixo de tensão.

Nessa primeira prática, o circuito mostrado na Figura 4.11 é construído como mostrado na Figura ??.

Os passos a serem seguidos são os seguintes:

1. Conecte o ânodo do LED (perna grande do LED) na porta 13 do galileo.
2. Conecte o cátodo do LED (perna pequena do LED) na porta GND.
3. Escreva o código na IDE do Arduino para piscar LED .

## I.1.2 Esquema de montagem

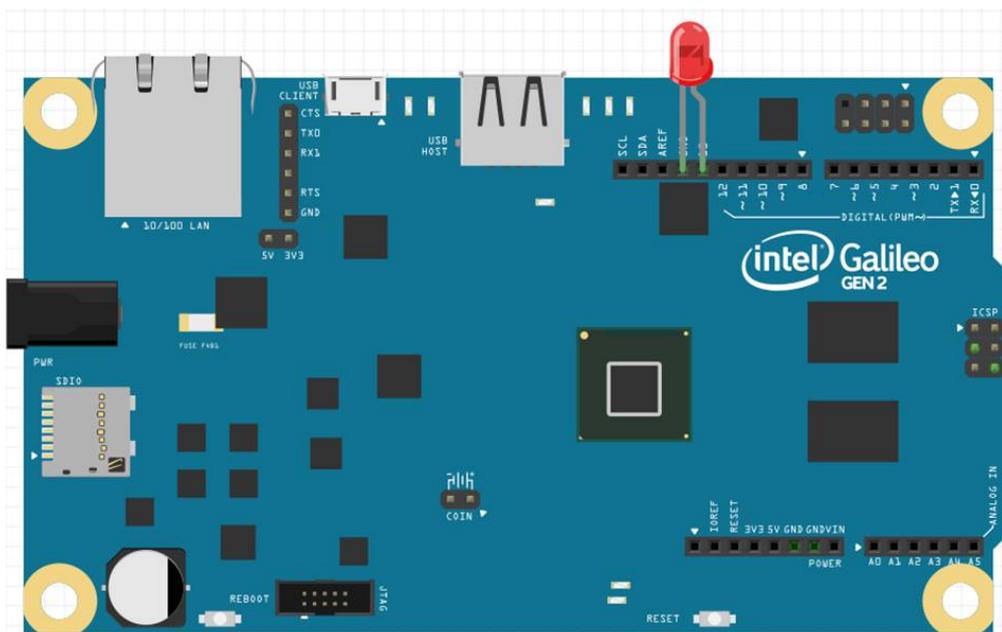


Figura I.1: Circuito da prática 1.

## I.1.3 Código fonte

```
1
2 /*
3  * Funcao Setup:
4  *
5  * Primeira funcao executada pelo sistema.
6  * Essa funcao eh utilizada para fazer as configuracoes iniciais
7  */
8 void setup() {
9     // Seleciona a porta digital 13 como saida de tensao
10    pinMode(13, OUTPUT);
11 }
12
13 /*
14  * Funcao Loop:
15  *
16  * Funcao a ser executada continuamente pelo sistema
17  * apos a execucao da funcao setup()
18  *
19  */
20
21 void loop() {
22
23     digitalWrite(13, HIGH); // Coloca nivel alto de tensao (5 V) na porta digital 13
24     delay(1000); // Faz nada por 1000 ms ( 1 segundo)
```

```
25     digitalWrite(13, LOW); // Coloca nivel baixo de tensao (0 V) na porta digital 13
26     delay(1000); //Faz nada por 1000 ms ( 1 segundo)
27 }
```

Code I.1: Código Prática 1

#### I.1.4 Comentários

Com relação a esta prática, os procedimentos de instalação do driver da placa Intel<sup>®</sup> Galileo podem ser feitos anteriormente à execução da prática laboratorial, de forma a economizar tempo para outros tópicos.

## I.2 Prática 2: Introdução a leitura de sensores e tipos de variáveis

### I.2.1 Procedimentos

Os passos a serem seguidos nessa prática são os seguintes:

- Construção do circuito de um sensor de luz;
- Codificação para leitura de valores de tensão em tal circuito.

### I.2.2 Esquema de montagem

Para esta prática, um circuito de acordo com o esquemático mostrado na Figura I.2.

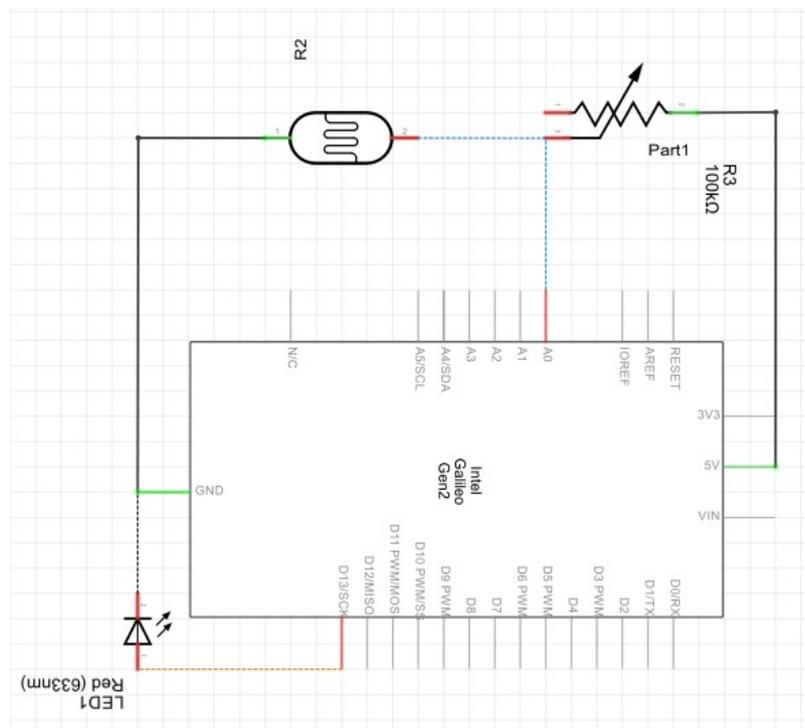


Figura I.2: Esquemático do circuito da prática 2.

A construção do circuito na protoboard é mostrada na Figura ??.

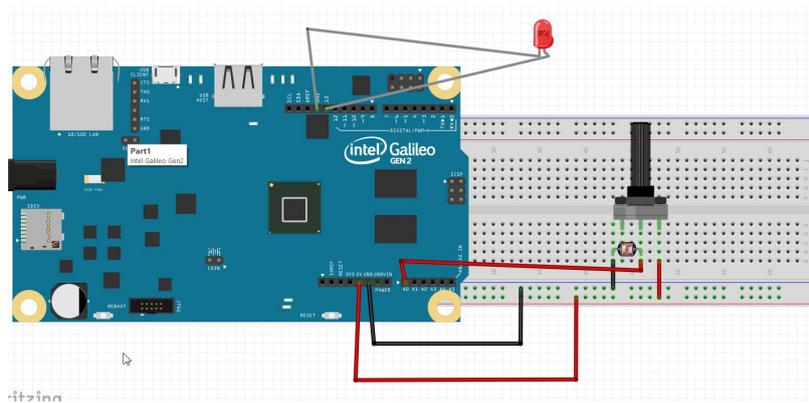


Figura I.3: Circuito da prática 2 construído numa protoboard.

### I.2.3 Código fonte

```

1 #define pinoLed 13 // Pino no qual o LED estah conectado
2 int leituraAD = 0; // leitura do sensor LDR a ser feita com o conversor AD por meio
  da funcao analogRead().
3         // conversao 5 V -> 4095 (12 bits) 11111 11111
4         //           0 V -> 0           00000 00000
5 float sensorLuz; // Valor analogico aproximado da leitura do sensor
6
7 void setup()
8 {
9
10 Serial.begin(9600); // inicia a comunicacao serial Galileo -> Computador
11                   //                                     <-
12 pinMode(pinoLed, OUTPUT); // seleciona a porta digital pinoLed para saida
13
14 }
15
16
17 char letra = 'a';
18 char * texto = "isso eh um texto";
19 void loop()
20 {
21 leituraAD = analogRead(A0); // ler o sinal anal?gico na porta digital A0
22
23 sensorLuz = (float) ( leituraAD * 5)/4095; //regra de 3 escala de 0 a 4095 digital,
  para 0 a 5 anal?gico
24
25 Serial.print("Leitura AD = ");
26 Serial.print(leituraAD);
27
28 Serial.print(" Conversao para float = ");
29 Serial.println(sensorLuz);
30
31
32 delay(500); // faz nada por 500 ms
33

```

## I.2.4 Comentários

No código fonte desta prática, são utilizados diretivas de programação, e todos tipos básicos de variável. É utilizado também uma variável de tipo char \*. Tal tipo será estudado numa prática de vetores e alocação dinâmica de memória a ser mostrada posteriormente.



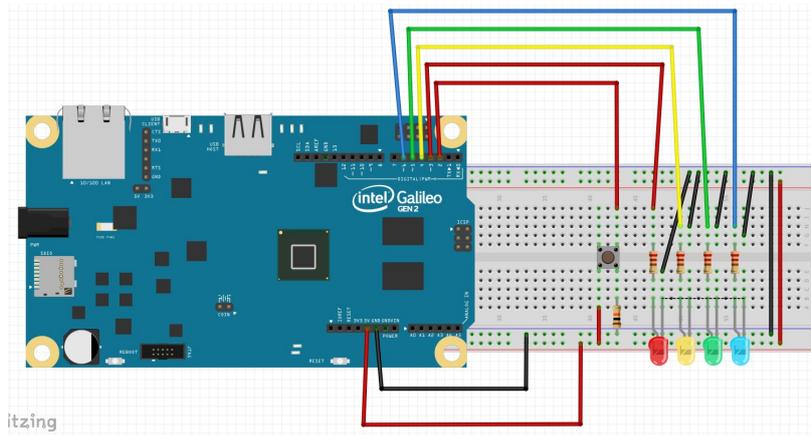


Figura I.5: Circuito da prática 3 construído numa protoboard.

### I.3.3 Código fonte

```

1 #define pinoBotao 2
2 #define pinoLedVermelho 3
3 #define pinoLedAmarelo 4
4 #define pinoLedVerde 5
5 #define pinoLedAzul 6
6 #define tempoDelay 300
7
8 void setup()
9 {
10  pinMode(pinoBotao, INPUT); // Define o pino 2 como input de tensao
11  pinMode(pinoLedVermelho, OUTPUT); // Define o pino 3 como output de tensao
12  pinMode(pinoLedAmarelo, OUTPUT); // Define o pino 4 como output de tensao
13  pinMode(pinoLedVerde, OUTPUT); // Define o pino 5 como output de tensao
14  pinMode(pinoLedAzul, OUTPUT); // Define o pino 6 como output de tensao
15
16 }
17
18 int cont = 1;
19
20 void loop()
21 {
22
23  if(digitalRead(pinoBotao) == HIGH)
24  {
25    cont = cont + 1;
26
27    if(cont > 6)
28    {
29      cont = 1;
30    }
31  }
32
33  if(cont == 1)
34  {

```

```

35     digitalWrite(pinoLedVermelho , HIGH);
36     digitalWrite(pinoLedAmarelo , HIGH);
37     digitalWrite(pinoLedVerde , HIGH);
38     digitalWrite(pinoLedAzul , HIGH);
39
40     delay(tempoDelay);
41
42     digitalWrite(pinoLedVermelho , LOW);
43     digitalWrite(pinoLedAmarelo , LOW);
44     digitalWrite(pinoLedVerde , LOW);
45     digitalWrite(pinoLedAzul , LOW);
46
47     delay(tempoDelay);
48
49 }
50 else
51 {
52     digitalWrite(cont , HIGH);
53
54     delay(tempoDelay);
55
56     digitalWrite(cont , LOW);
57
58     delay(tempoDelay);
59 }
60 }

```

Code I.3: Código Prática 3

### I.3.4 Comentários

No código fonte desta prática, é utilizado uma variável inteira como contador de forma a controlar os LED por meio do botão.



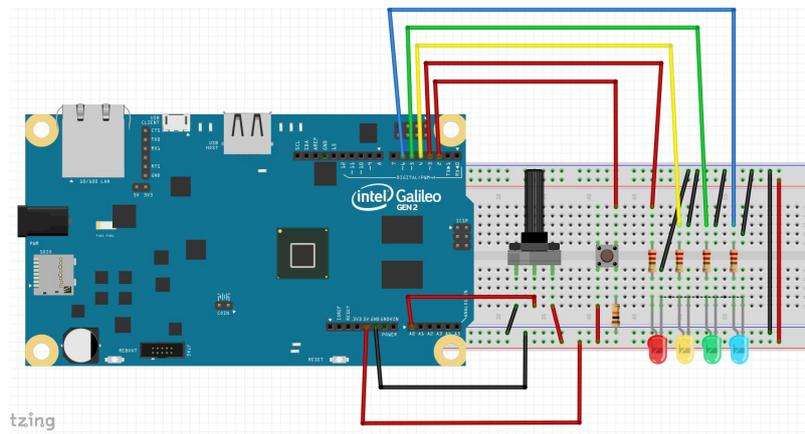


Figura I.7: Circuito da prática 4 construído numa protoboard.

### I.4.3 Código fonte

```

1 #define pinoBotao 2
2 #define tempoDelayPadrao 200
3 #define ledAzul 3
4 #define ledverde 4
5 #define ledAmarelo 5
6 #define ledvermelho 6
7
8 void setup()
9 {
10     int i;
11     for( i = 3; i <= 6; i++)//usando o laço for para definir as portas 3 a 6 como
12         // portas de saida
13     {
14         pinMode(i, OUTPUT); // Define o pino 3 como output de tensao
15     }
16     pinMode(pinoBotao, INPUT); // Define o pino 2 como input de tensao
17 }
18
19
20 int comportamento = 0;
21 int i;
22 int sensorA0;
23 void loop()
24 {
25     i = 0;
26
27     for( i = ledAzul ; i <= ledVermelho; i++) //Apagando todos LEDs
28     {
29         digitalWrite(i, LOW);
30     }
31     i= 0;
32
33     sensorA0 = analogRead(A0); // 0 < sensorA0 <= 1023

```

```

34
35  if(digitalRead(pinoBotao) == HIGH)//Altera o comportamento a ser obedecido ao
    aperta o botao
36  {
37      comportamento += 1;
38      if(comportamento > 3)
39          comportamento = 0;
40  }
41
42  switch (comportamento)
43  {
44
45      case 1:// Comportamento 1: Piscar apenas um LED a depender do valor de tensao
        lido no potenciometro
46          if(sensorA0 < 250)
47          {
48              while( i < 7)
49              {
50                  if( i == ledAzul)
51                  {
52                      digitalWrite(i, LOW);
53                      delay(sensorA0);
54                      digitalWrite(i, HIGH);
55                      delay(sensorA0);
56                  }
57                  else
58                  {
59                      digitalWrite(i, LOW);
60
61                  }
62                  i++;
63              }
64              i = 0;
65          }
66          else if(sensorA0 >= 250 && sensor < 500)
67          {
68              while( i < 7)
69              {
70                  if( i == ledVerde)
71                  {
72                      digitalWrite(i, LOW);
73                      delay(sensorA0);
74                      digitalWrite(i, HIGH);
75                      delay(sensorA0);
76                  }
77                  else
78                  {
79                      digitalWrite(i, LOW);
80
81                  }
82                  i++;
83              }

```

```

84     i = 0;
85 }
86 else if(sensor >= 500 && sensor < 750)
87 {
88     while( i < 7)
89     {
90         if( i == ledAmarelo)
91         {
92             digitalWrite(i, LOW);
93             delay(sensorA0);
94             digitalWrite(i, HIGH);
95             delay(sensorA0);
96         }
97         else
98         {
99             digitalWrite(i, LOW);
100
101         }
102         i++;
103     }
104     i = 0;
105 }
106 else if(sensor >= 750 && sensor <= 1023)
107 {
108     while( i < 7)
109     {
110         if( i == ledVermelho)
111         {
112             digitalWrite(i, LOW);
113             delay(sensorA0);
114             digitalWrite(i, HIGH);
115             delay(sensorA0);
116         }
117         else
118         {
119             digitalWrite(i, LOW);
120
121         }
122         i++;
123     }
124
125     i = 0;
126 }
127 break;
128
129 case 2://Comportamento 2: piscar os leds, do azul ao vermelho com velocidade
determinada pelo potenciometro
130     for( i = ledAzul ; i <= ledVermelho; i++)
131     {
132
133         digitalWrite(i, HIGH);
134         delay(sensorA0);

```

```

135         digitalWrite(i, LOW);
136         delay(sensorA0);
137     }
138
139     break;
140     case 3://Comportamento 3: COMPORTAMENTO A SER ESCOLHIDO PELOS ALUNOS
141         for( i = ledVermelho ; i >= ledAzul; i--)// NESSE EXEMPLO, FOI ESCOLHIDO
REALIZAR O COMPORTAMENTO INVERSOR AO COMPORTAMENTO 2
142         {
143
144             digitalWrite(i, HIGH);
145             delay(sensorA0);
146             digitalWrite(i, LOW);
147             delay(sensorA0);
148         }
149     break;
150     default:
151         for( i = ledAzul ; i <= ledVermelho; i++)
152         {
153
154             digitalWrite(i, LOW);
155         }
156     }
157 }
158 }

```

Code I.4: Código Prática 4

#### I.4.4 Comentários

No código fonte desta prática, o comportamento 3 foi escolhido apenas como um exemplo. Talvez seja interessante, dependendo do nível de aprendizagem da turma, fazer uma pequena disputa entre equipes na criação do comportamento mais divertido.

## **I.5 Prática 5: Uso de vetores, shift register e tipos variados de dados**

### **I.5.1 Procedimentos**

Os passos a serem seguidos nessa prática são os seguintes:

- Conectar o registrador de deslocamento 78HC545 às portas 5,6 7 do Galileo.
- Conectar 8 conjuntos LED + resistor às portas de saída do registrador de deslocamento.
- Escrever um código usando loop e a função shiftOut para realizar os objetivos definidos.

### **I.5.2 Esquema de montagem**

Para esta prática, um circuito de acordo com o esquemático mostrado na Figura I.8.



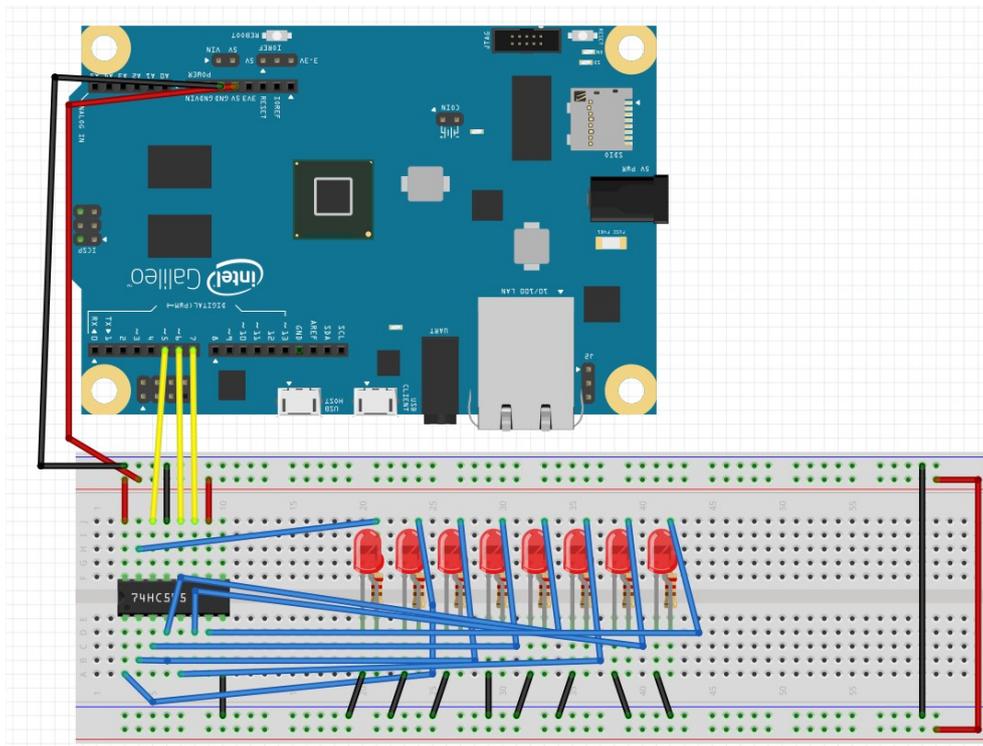


Figura I.9: Circuito da prática 5 construído numa protoboard.

### I.5.3 Código fonte

#### Objetivo 1:

```

1
2 #define pinoClock 7
3 #define pinoLatch 6
4 #define pinoData 5
5
6 byte numeroOut = 0; // byte = 8 bits
7 int bits[8];
8 int i;
9 void setup() {
10 pinMode(pinoLatch, OUTPUT);
11 pinMode(pinoClock, OUTPUT);
12 pinMode(pinoData, OUTPUT);
13 Serial.begin(9600);
14
15 //Setando a variavel bits = B11100101
16 bits[0] = 1; bits[1] = 0; bits[2] = 1; bits[3] = 0; bits[4] = 0; bits[5] = 1; bits[6] =
    1; bits[7] = 1;
17
18 //calculando o numero a ser usado na função shiftOut usando a regra mostrada na
    seção 4.3.13
19 for(i = 0; i < 8; i++)
20 {
21     numeroOut += bits[i]*pow(2,i);

```

```

22 }
23
24 }
25
26 void loop() {
27
28
29 digitalWrite(pinoLatch, LOW);
30
31 // envia dados nos pinos de saida a partir do bit menos significa
32 //tivo da variavel numeroOut no pinoData
33
34 shiftOut(pinoData, pinoClock, LSBFIRST, numeroOut);
35 // utilizar shiftOut(pinoData, pinoClock, LSBFIRST, B11100101);
36 // funcionaria da mesma forma, visto que o compilador sabe lidar com as bases
    decimal, binária, octal e hexadecimal
37 // Entretanto, o objetivo da prática e a utilização de vetores, daí forçar o cálculo
    binario -> decimal
38
39
40 digitalWrite(pinoLatch, HIGH);
41 }

```

Code I.5: Código Prática 5 - Objetivo 1.

## Objetivo 2:

```

1
2 #define pinoClock 7
3 #define pinoLatch 6
4 #define pinoData 5
5
6 int i, j, temp;
7 int digitosResto[8];
8 void setup() {
9 pinMode(pinoLatch, OUTPUT);
10 pinMode(pinoClock, OUTPUT);
11 pinMode(pinoData, OUTPUT);
12 Serial.begin(9600);
13
14
15 }
16
17 void loop()
18 {
19
20 for(i = 0 ; i <= 255; i++)
21 {
22     digitalWrite(pinoLatch, LOW);
23
24 // envia dados nos pinos de saida a partir do bit menos significa
25 //tivo da variavel numeroOut no pinoData
26

```

```

27  shiftOut(pinoData , pinoClock , LSBFIRST, i);
28  Serial.print("Contagem = ");
29  Serial.print(i);
30
31  temp = i; // tem eh a variavel temporaria que passara pelo processo de divisoes
32  sucessivas a cada iteracao
33  j = 7; //indice dos resto inicial (digito menos significativo)
34  Serial.print(" = ");
35  while(j != 0)
36  {
37      digitosResto[j] = temp%2; //registra o resto
38      j--;
39      temp /= 2; //atualizar o dividendo
40  }
41
42  //Imprimindo o numero na base binaria na ordem correta
43  for(j = 0; j < 8; j++)
44  {
45      Serial.print(digitosResto[j]);
46  }
47
48  Serial.println();
49
50
51  digitalWrite(pinoLatch , HIGH);
52  delay(500);
53  }
54  }

```

Code I.6: Código Prática 5 - Objetivo 2.

#### I.5.4 Comentários

Como comentado no código do objetivo 1, está prática não precisa, necessariamente, do uso de vetor para simbolizar os LED's acessos ou apagado. O número B1100101 (base binária) também seria aceito pelo compilador, assim como seria aceito um número na base hexadecimal.

O objetivo 2 demanda saber criar um algoritmo para fazer a conversão decimal-binário de um número no intervalo de 0 a 255 (8 bits).

## I.6 Prática 6: Uso de funções e sensor de temperatura

### I.6.1 Procedimentos

Os passos a serem seguidos nessa prática são os seguintes:

- Conectar 3 conjuntos LED + resistor às portas digitais 2,3 e 4 da placa Galileo.
- Conectar o LM35 à porta analógica A0 apropriadamente, como mostrado na seção 4.2.10.
- Escrever um código para cumprir o algoritmo descrito para essa prática.

### I.6.2 Esquema de montagem

Para esta prática, um circuito de acordo com o esquemático mostrado na Figura I.10.

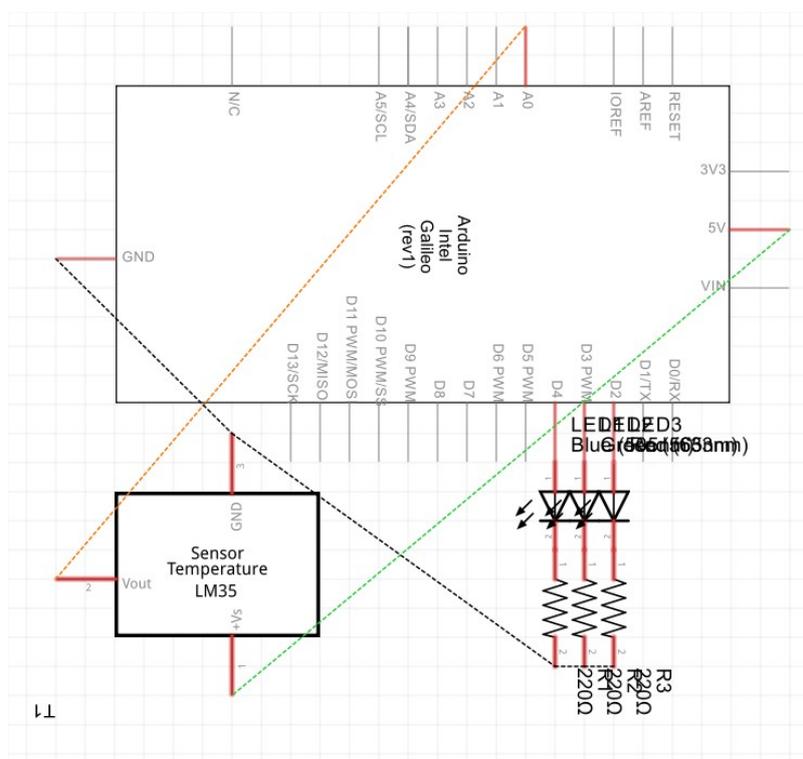


Figura I.10: Esquemático do circuito da prática 6.

A construção do circuito na protoboard é mostrada na Figura I.11.



```

29
30 return valorLido;
31 }
32
33 void escreveTemperaturaTerminal(float temperatura)
34 {
35     Serial.print("Temperatura = ");
36     Serial.print(temperatura);
37 }
38
39 void piscaLed(float temperatura)
40 {
41     if(temperatura > temperaturaAmbiente - 2 && temperatura < temperaturaAmbiente
42         +2 )
43     {
44         digitalWrite(ledQuente, LOW);
45         digitalWrite(ledFrio, LOW);
46         digitalWrite(ledNormal, HIGH);
47     }
48     else if ( temperatura >= temperaturaAmbiente +2 )
49     {
50         digitalWrite(ledQuente, HIGH);
51         digitalWrite(ledFrio, LOW);
52         digitalWrite(ledNormal, LOW);
53     }
54     else if( temperatura <= temperaturaAmbiente - 2)
55     {
56         digitalWrite(ledQuente, LOW);
57         digitalWrite(ledFrio, HIGH);
58         digitalWrite(ledNormal, LOW);
59     }
60 }
61
62
63 void setup()
64 {
65     inicializaPortasDigitais();
66     inicializaSerial();
67 }
68
69 float temperatura;
70 void loop()
71 {
72     temperatura = leTemperatura(0);
73     escreveTemperaturaTerminal(temperatura);
74     piscaLed(temperatura);
75
76     delay(300);
77 }

```

Code I.7: Código Prática 6.

#### **I.6.4 Comentários**

Pode-se dizer que a quantidade de funções definidas para esta prática foi além do necessário, entretanto, é realmente preciso que o aluno saiba identificar e dividir o problema, não apenas como uma habilidade para desenvolvimento de programas, mas como uma habilidade a ser usada em todas áreas da vida.

## I.7 Prática 7: Uso de structs, interrupção de hardware e circuito de debounce de sinais

### I.7.1 Procedimentos

Os passos a serem seguidos nessa prática são os seguintes:

- Montar 2 conjuntos interruptor + circuito debouncing(Sdhmitt trigger e filtro RC).
- Conectar a saída dos conjuntos supracitados às portas digitais 2 e 3, as quais são as que oferecem interrupções de hardware.
- Conectar 2 LED + resistor 220 ohm às portas digitais 5 e 6.
- Escrever um código para cumprir o algoritmo descrito para essa prática.

### I.7.2 Esquema de montagem

Para esta prática, um circuito de acordo com o esquemático mostrado na Figura I.12.

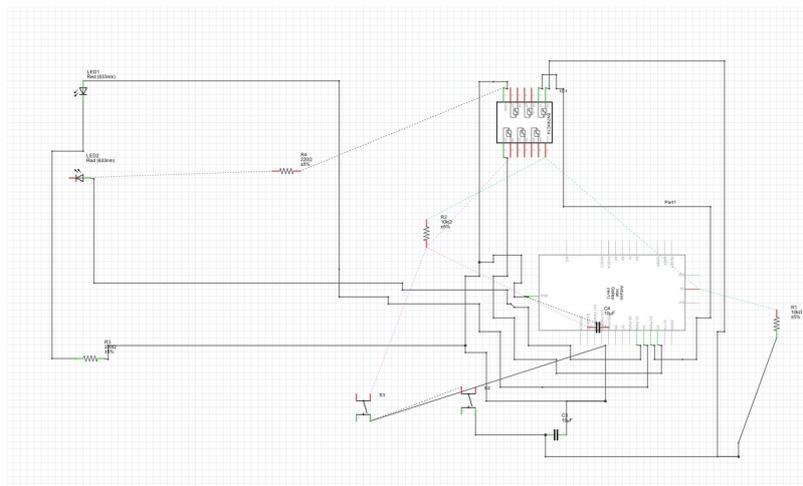


Figura I.12: Esquemático do circuito da prática 7.

A construção do circuito na protoboard é mostrada na Figura I.13.

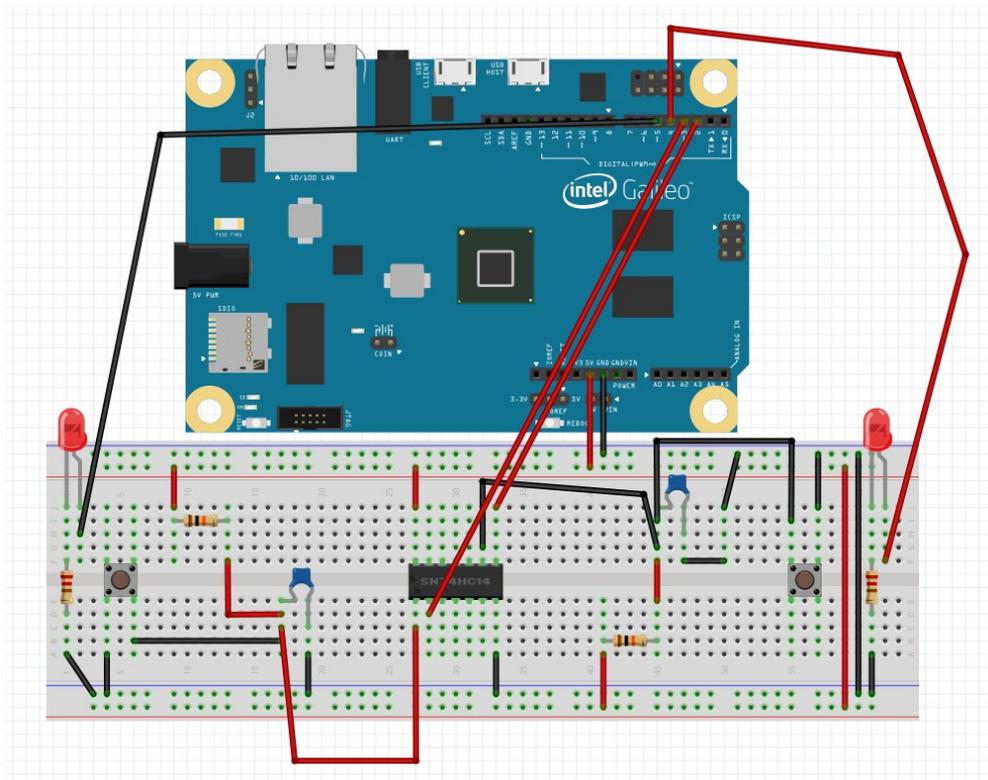


Figura I.13: Circuito da prática 7 construído numa protoboard.

### I.7.3 Código fonte

```

1
2
3 #define int0 0
4 #define int1 1
5 #define intPin0 2
6 #define intPin1 3
7 #define ledJogador1 4;
8 #define ledJogador2 5;
9 #define tempoDisputa 5000 // 5000 ms = 5 segundos
10 //Parametros dos jogadores 1 e 2
11 const char * nomeJogador1 = "Jogador1";
12 const char * nomeJogador2 = "Jogador2";
13 const char * fraseVitoriaJogador1 = "Frase Vitoria1";
14 const char * fraseVitoriaJogador2 = "Frase Vitoria2";
15
16 //Struct para armazenar as informacoes de um jogador
17 typedef struct
18 {
19     int numClicks;
20     char *nome;
21     char *fraseVitoria;
22     float velocidade;
23 }Jogador;
24

```

```

25 //variavel de cada um dos jogadores
26 Jogador jogador1;
27 Jogador jogador2;
28
29 // Rotina de interupcao de hardware na porta digital 2
30 void rotinaInt0()
31 {
32     jogador1.numClicks++;
33 }
34 // Rotina de interupcao de hardware na porta digital 3
35 void rotinaInt1()
36 {
37     jogador2.numClicks++;
38 }
39
40 //funcao para inicializar as struct de cada jogador
41 void inicializaJogador(jogador * jog , int numJogador)
42 {
43     jog[0].numClicks = 0;
44     if(numJogador == 1)
45     {
46         jog[0].nome = nomeJogador1;
47         jog[0].fraseVitoria = fraseVitoriaJogador1;
48         jog[0].velocidade = 0.0;
49     }
50     else if (numJogador == 2)
51     {
52         jog[0].nome = nomeJogador2;
53         jog[0].fraseVitoria = fraseVitoriaJogador2;
54         jog[0].velocidade = 0.0;
55     }
56 }
57
58 //funcao para calcular a velocidade de apertar os botoes de cada jogador
59 float calculaVelocidade( int numClicks , int numMiliSegundos)
60 {
61
62     return (float)(numClicks*1000)/numMiliSegundos;
63 }
64
65 //Funcao para printar as informacoes da disputa
66 // printar a frase do vencedor
67 // e piscar o led dele a quantidade de vezes que ele
68 // pressionou o botao
69 void resultadoFinal(Jogador jogador1 , Jogador jogador2)
70 {
71     int i;
72     //Info jogador 1
73     Serial.println("Resultados Finais:");
74     Serial.println("Jogador 1:");
75     Serial.print("Nome: ");
76     Serial.println(jogador1.nome);

```

```

77 Serial.print("Numero de clicks: ");
78 Serial.println(jogador1.numClicks);
79 Serial.print("Jogador 1 - velocidade: ");
80 Serial.print(jogador1.velocidade);
81 Serial.println(" clicks por segundo");
82 //Info jogador 2
83 Serial.println("Resultados Finais:");
84 Serial.println("Jogador 2:");
85 Serial.print("Nome: ");
86 Serial.println(jogador2.nome);
87 Serial.print("Numero de clicks: ");
88 Serial.println(jogador2.numClicks);
89 Serial.print("Jogador 2 - velocidade: ");
90 Serial.println(jogador2.velocidade);
91 Serial.println(" clicks por segundo");
92
93 if(jogador1.velocidade > jogador2.velocidade)
94 {
95     Serial.print("O vencedor ï£i ");
96     Serial.println(jogador1.nome);
97     Serial.println(jogador1.fraseVitoria);
98     //pisca led do vencedor o numero de vezes que ele apertou o botao
99     for(i = 0; i < jogador1.numClicks; ++i;)
100     {
101         digitalWrite(ledJogador1, HIGH);
102         delay(200);
103         digitalWrite(ledJogador1, LOW);
104         delay(200);
105     }
106 }
107 else if (jogador1.velocidade < jogador2.velocidade)
108 {
109
110     Serial.print("O vencedor ï£i ");
111     Serial.println(jogador2.nome);
112     Serial.println(jogador2.fraseVitoria);
113     for(i = 0; i < jogador2.numClicks; ++i;)
114     {
115         digitalWrite(ledJogador2, HIGH);
116         delay(200);
117         digitalWrite(ledJogador2, LOW);
118         delay(200);
119     }
120
121 }
122 else
123 {
124     Serial.println("EMPATE!");
125     Serial.println(jogador1.fraseVitoria);
126     Serial.println(jogador2.fraseVitoria);
127     for(i = 0; i < jogador2.numClicks; ++i)
128     {

```

```

129     digitalWrite(ledJogador2 , HIGH);
130     digitalWrite(ledJogador1 , HIGH);
131     delay(200);
132     digitalWrite(ledJogador2 , LOW);
133     digitalWrite(ledJogador1 , LOW);
134     delay(200);
135 }
136 }
137
138
139
140 }
141
142 void zeraJogadores ()
143 {
144     jogador1.numClicks = 0;
145     jogador2.numClicks = 0;
146 }
147
148 void setup ()
149 {
150     //Configuarando comunicacao serial
151     Serial.begin(9600);
152     //Configurando pinos digitais
153     pinMode(ledJogador1 , OUIPUT);
154     pinMode(ledJogador2 , OUIPUT);
155     pinMode(intPin0 , INPUT);
156     pinMode(intPin1 , INPUT);
157     //Vinculando rotinas de interrupcao de hardware
158     attachInterrupt(int0 ,rotinaInt0 , RISING);
159     attachInterrupt(int1 ,rotinaInt1 , RISING);
160     //Inicializando os parametros dos dois jogadores
161     inicializaJogador(&jogador1 , 1);
162     inicializaJogador(&jogador2 , 2);
163
164 }
165
166 void loop ()
167 {
168     //No periodo de delay , as interrupcoes continuam a ser lidas normalmente
169     delay(tempoDisputa);
170
171     jogador1.velocidade = calculaVelocidade(jogador1.numClicks , tempoDisputa);
172     jogador2.velocidade = calculaVelocidade(jogador2.numClicks , tempoDisputa);
173     //
174     resultadoFinal(jogador1 ,jogador2);
175
176     zeraJogadores ()
177     delay(10000);
178
179

```

Code I.8: Código Prática 7.

#### I.7.4 Comentários

Esta prática é destinada ao desenvolvimento de habilidades relacionadas a uso de interrupções, inversores, filtros RC e codificação de regras de negócio específicas. O professor pode decidir alterar as regras de negócio propostas nessa prática assim como o resultado do uso do botões com filtros de *debouncing* .

## I.8 Prática 8: Prática final

### I.8.1 Procedimentos

Os passos a serem seguidos nessa prática são os seguintes:

- Montar um circuito com uma matriz de leds 8x8 com uma solução para leitura de movimentos
- Escrever um código para cumprir o algoritmo descrito para essa prática.

### I.8.2 Esquema de montagem

Para esta prática, um circuito de acordo com o esquemático mostrado na Figura I.14.

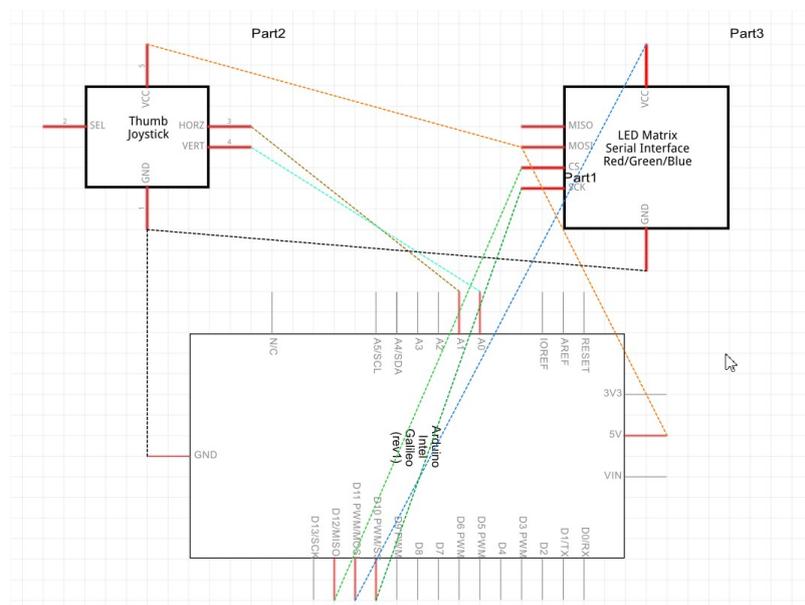


Figura I.14: Esquemático do circuito da prática 8.

A construção do circuito na protoboard é mostrada na Figura I.15.

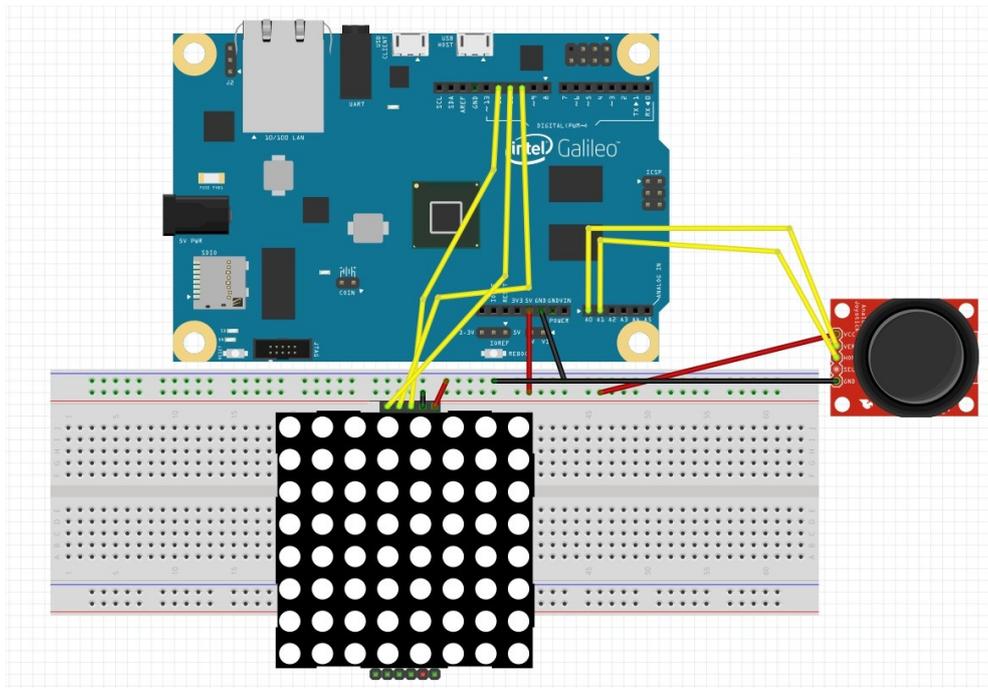


Figura I.15: Circuito da prática 8 construído numa protoboard.

### I.8.3 Código fonte

```

1 #include "LedControl.h"
2 LedControl lc=LedControl(12,11,10,1); //
3 #define dificuldade 1
4 #define tempo1 1000
5 #define tempo2 750
6 #define tempo3 500
7 // pino 12 conectado ao pino 1 do MAX7219
8 // pino 11 conectado ao pino 13 CLK
9 // pino 10 conectado ao pino 12 LOAD
10 // O número 1 indica que 1 MAX7219 será usado
11
12 //*****
13 // Struct definidas para o programa
14 typedef struct
15 {
16     byte x;
17     byte y;
18     byte estado;
19 }
20 }Celula; //Estrutura de uma celula
21
22 typedef struct
23 {
24     Celula corpo [64];
25     int numCelulasOcupadas;
26 }Snake; //Estrutura do corpo da cobra

```

```

27
28 //*****
29
30 //*****
31 //Variaveis globais de funcionamento do programa
32 int tempo;
33 int movimentoX;
34 int movimentoY;
35 byte matriz[8][8];
36 Snake snake;
37 Celula presa;
38
39 int sensorX;
40 int sensorY;
41 //*****
42
43
44 //Funcao para inicializar a matriz, o corpo da cobra e o tempo de transicao entre
    celulas (dificuldades)
45 void inicializa()
46 {
47     int i, j, cont =0;
48
49     //seta dificuldade do jogo
50     switch(dificuldade)
51     {
52     case 1:
53         tempo = tempo1;
54         break;
55     case 2:
56         tempo = tempo2;
57         break;
58     case 3:
59         tempo = tempo3;
60         break;
61     }
62     //inicializa o movimento inicial da cobra
63     movimentoX = 1;
64     movimentoY = 0;
65
66     //Inicializa matriz e corpo da cobra
67     snake.numCelulasOcupadas = 3;
68     for(i = 0; i < 8; i++ )
69     {
70         for(j = 0; j < 8 ; j++ )
71         {
72             matriz[i][j] = 0; //todos LEDs desligados
73
74             // posicao inicial cobra 3 partes (2,0), (1,0) (0,0)
75
76             if(cont == 0)
77             {

```

```

78     snake.corpo[cont].x = 2;
79     snake.corpo[cont].y = 7;
80     snake.corpo[cont].estado = 1;
81 }
82 else if(cont == 1)
83 {
84     snake.corpo[cont].x = 1;
85     snake.corpo[cont].y = 7;
86     snake.corpo[cont].estado = 1;
87 }
88 else if (cont == 2)
89 {
90     snake.corpo[cont].x = 0;
91     snake.corpo[cont].y = 7;
92     snake.corpo[cont].estado = 1;
93 }
94 else
95 {
96     snake.corpo[cont].x = -1;
97     snake.corpo[cont].y = -1;
98     snake.corpo[cont].estado = -1;
99 }
100
101     cont++;
102 }
103
104 }
105 }
106
107
108 //funcao para ler o joystick
109 // movimentoX = 1; direita
110 // movimentoX = -1 esquerda
111 // movimentoX = 0, faz nada
112 // movimentoY = 1; cima
113 // movimentoY = -1 baixo
114 // movimentoY = 0, faz nada
115
116 void leSensores()
117 {
118     int i;
119     int sensorX = 0, sensorY = 0;
120     for(i = 0; i < 10; i++)
121     {
122         sensorX +=analogRead(A0);
123         sensorY +=analogRead(A1);
124     }
125
126     sensorX /= 10;
127     sensorY /= 10;
128
129     if(movimentoY == 0)

```

```

130 {
131     if(sensorY > 900 )
132     {
133         movimentoY = -1;
134         movimentoX = 0;
135         return;
136     }
137     else if(sensorY < 100)
138     {
139         movimentoX = 0;
140         movimentoY = 1;
141         return;
142     }
143 }
144 if(movimentoX == 0)
145 {
146     if(sensorX > 900 )
147     {
148         movimentoY = 0;
149         movimentoX = 1;
150         return;
151     }
152     else if(sensorX < 100)
153     {
154         movimentoX = -1;
155         movimentoY = 0;
156         return;
157     }
158 }
159
160 }
161
162
163 //funcao para verificar se a posicao (x,y) ja esta ocupada pela cobra
164 // 1 - posicao ocupada
165 // 0 - posicao desocupada
166 int verificaPosicoes(int x, int y)
167 {
168     int i;
169     for(i = 0; i < snake.numCelulasOcupadas; ++i)
170     {
171         if(snake.corpo[i].x == x && snake.corpo[i].y == y )
172         {
173             //posicao ocupada pelo corpo da cobra
174             return 1;
175         }
176     }
177
178     //posicao livre
179     return 0;
180 }
181

```

```

182 //
183 //gera uma posicao aleatoria para a presa (x,y) num lugar que nao ha cobra
184 void geraPosicaoPresa ()
185 {
186
187     do{
188         presa.x = random(8); //gera um numero aleatorio entre 0 e 7
189         presa.y = random(8);
190     }while(verificaPosicoes(presa.x, presa.y));
191 }
192
193 int verificaAutoFagia ()
194 {
195     int i;
196     for(i = 1; i < snake.numCelulasOcupadas; ++i)
197     {
198         if(snake.corpo[i].x == snake.corpo[0].x && snake.corpo[i].y == snake.corpo
199         [0].y )
200         {
201             //cobra se atacou
202             return 1;
203         }
204
205
206     }
207     return 0;
208 }
209 //funcao para atualizar as posicoes das celuals do corpo da cobra
210 void atualiza ()
211 {
212     int i;
213
214     Celula temp,temp2;
215
216     temp = snake.corpo[0];
217
218
219     if(snake.corpo[0].x + movimentoX > 7 || snake.corpo[0].x + movimentoX < 0 ||
220     verificaAutoFagia())
221     {//FIM
222         inicializa();
223     }
224
225     if(snake.corpo[0].y + movimentoY > 7 || snake.corpo[0].y + movimentoY < 0)
226     {//FIM
227         inicializa();
228     }
229
230     snake.corpo[0].x += movimentoX;
231     snake.corpo[0].y += movimentoY;

```

```

232
233 //verifica se o alvo foi alcancado
234 if(snake.corpo[0].x == presa.x && snake.corpo[0].y == presa.y)
235 {
236     snake.numCelulasOcupadas++;
237     geraPosicaoPresa();
238 }
239
240
241 for(i = 1; i < snake.numCelulasOcupadas; i++)
242 {
243     temp2 = snake.corpo[i];
244     snake.corpo[i] = temp;
245     temp = temp2;
246 }
247
248 //snake.corpo[1].x += tempX
249 //snake.corpo[1].y += movimentoY;
250
251
252 }
253
254 void printSnake()
255 {
256     int i;
257     lc.clearDisplay(0);
258     for(i = 0; i < snake.numCelulasOcupadas; ++i)
259     {
260         lc.setLed(0, snake.corpo[i].y, snake.corpo[i].x, true); // desliga o LED na
261     }
262
263     lc.setLed(0, presa.y, presa.x, true); // desliga o LED na
264 }
265
266 void printaSerial()
267 {
268
269     Serial.print(0);
270     Serial.print(": ( ");
271     Serial.print(snake.corpo[0].x);
272     Serial.print(" , ");
273     Serial.println(snake.corpo[0].y);
274
275 }
276 void setup()
277 {
278     // O número 0 como o primeiro argumento de muitas funções
279     // da biblioteca LedControl.h indica o uso de 1 chip MAX7219
280     Serial.begin(9600);
281     randomSeed(analogRead(2)); //inicializa uma sequencia de numeros aleatorios de um
        pino analogico aberto
282     lc.shutdown(0, false); // Desliga o modo de economia de energia, //habilita o

```

```

    display
283
284 lc.setIntensity(0,8); // seta a claridade (input entre 0 e 15)
285 lc.clearDisplay(0); // limpa o display
286
287 //inicializa parametros do jogo
288 inicializa();
289 geraPosicaoPresas();
290 printSnake();
291 delay(tempo);
292 }
293
294
295 void loop()
296 {
297   leSensores();
298   atualiza();
299   printSnake();
300   printaSerial();
301   // esse delay define a dificuldade do jogo
302   delay(tempo);
303 }

```

Code I.9: Código Prática 8.

#### I.8.4 Comentários

Como a última prática do curso, o foco desta são as habilidades e conhecimentos de programação em C. A maior forma de resolução deste problema é o uso de *structs* apropriadas e modularização de instruções de forma adequada. Não é necessário o uso de interrupções de hardware ou software para esta prática.

## II. DESCRIÇÃO DO CONTEÚDO DO CD