



Sistemas Embebidos para Computação Física e IoT

Volume 2:
Interfaces, Sensores e Atuadores

RUI POLICARPO DUARTE

Projeto de Sistemas Embebidos para Computação Física e IoT

Volume 2: Interfaces, Sensores e Atuadores

Rui Policarpo Duarte, PhD, Eng

24 de outubro de 2023

Copyright © 2023, Rui Polcarpo Duarte

Todos os direitos reservados. Sem a prévia autorização por escrito do autor, esta obra não pode ser reproduzida, no todo ou em parte, por meio de gravação ou por qualquer processo mecânico, fotográfico ou eletrónico, nem ser introduzida numa base de dados, difundida ou de qualquer forma copiada para uso público ou privado, além do uso legal como breve citação em artigos e críticas.

1ª Edição: outubro de 2023

ISBN: 978-989-33-5364-6



Imagem da capa: Foto da placa de desenvolvimento CoolRunner II por Rui Polcarpo Duarte

Conteúdo

Lista de Figuras	xiv
Lista de Tabelas	xv
Lista de Programas Arduino	xviii
Nomenclatura	xix
1 Interfaces de Entrada/Saída	1
1.1 Circuitos Condicionadores de Sinal	2
1.2 Interfaces Analógicas	3
1.2.1 Conversão Impedância-Tensão	4
1.2.2 Conversão Capacidade-Frequência	6
1.2.3 Filtragem RC	7
1.2.4 Ponte de Wheatstone	11
1.2.5 Comparador Analógico	13
1.2.6 Circuito Limitador	14
1.2.7 Dreno/Colector Aberto	15
1.2.8 Push-Pull	17
1.2.9 Ponte-H	17
1.3 Interfaces Digitais	18
1.3.1 Entradas, Saídas e Interrupções	20
1.3.2 Multiplexagem e Charlieplexagem	28
1.3.3 UART	32
1.3.4 SPI	35
1.3.5 I2C	39
1.3.6 CAN	44
1.3.7 1-Wire	46
1.4 Isolamento Galvânico	49
1.4.1 Transformador-Isolador	49
1.4.2 Opto-Isolador	49
1.4.3 Isolador-CMOS	50
1.5 Conversão Analógico-Digital	52
1.5.1 Quantização	52

1.5.2	Ritmo de Amostragem	53
1.5.3	Conversão Analógico-Digital com Arduino	54
1.6	Conversão Digital-Analógico	58
1.7	Processamento Digital de Sinal	60
1.7.1	Filtragem Passa-Baixo	61
1.7.2	Filtragem Passa-Alto	62
1.8	Comunicações	63
1.8.1	RFID	66
1.8.2	LoRa	68
1.8.3	Sigfox	68
1.8.4	Ethernet	69
1.8.5	WiFi	71
1.8.6	Rede Telefónica Pública	72
1.8.7	Bluetooth	74
1.9	Interface Mecânica	75
1.10	Limitações Físicas	76
2	Sensores	79
2.1	Sensores Resistivos	81
2.2	Sensores Capacitivos	82
2.3	Sensores Indutivos	83
2.4	Calibração de um Sensor	83
2.5	Interruptores Momentâneos	85
2.5.1	Botões de Pressão	85
2.5.2	Sensor de Movimento	94
2.5.3	Sensor de Choque	96
2.5.4	Sensor de Inclinação	97
2.5.5	Toque / Sensor Tátil	98
2.5.6	Interruptor Magnético	102
2.5.7	Botão Rotativo	103
2.6	Posição	105
2.6.1	Potenciômetro	105
2.6.2	Manípulo de 2 Eixos (Joystick)	109
2.7	Teclados	110
2.7.1	Teclado Digital	110
2.7.2	Teclado Analógico	115
2.8	Luminosidade	117
2.8.1	Foto-Resistência	118
2.8.2	Fotodíodo e Fototransistor	120
2.8.3	Comparação das Diferentes Tecnologias	121
2.9	Fogo	123
2.10	Presença	126
2.11	Vibração	129

2.12	Campo Magnético	132
2.12.1	Efeito de Hall	132
2.12.2	Bobina	133
2.13	Consumo de Energia Elétrica	135
2.13.1	Resistência em Série	135
2.13.2	Fluxo Magnético	136
2.14	Temperatura	143
2.14.1	Termopar	144
2.14.2	RTD	147
2.14.3	NTC	147
2.14.4	PTC	148
2.14.5	Semicondutor	148
2.14.6	Pilha Termoelétrica	149
2.14.7	Circuito Integrado Digital	149
2.14.8	Comparação das Diferentes Tecnologias	151
2.15	Humidade	152
2.15.1	Higrómetro Resistivo	152
2.15.2	Higrómetro Capacitivo	152
2.16	Índice de Acidez (pH)	155
2.17	Aceleração	157
2.18	Acelerómetro + Magnetómetro + Giroscópio + Altímetro . .	163
2.19	Localização (GNSS/GPS)	164
2.20	Som	169
2.21	Som Subaquático	171
2.22	Pressão Atmosférica	172
2.23	Pressão Hidrostática	173
2.24	Força/Peso	174
2.25	Caudal	180
2.26	Gás	181
2.27	Distância	183
2.27.1	Infra-Vermelhos	183
2.27.2	Ultra-Sons	186
2.27.3	Laser	189
2.28	Radiação Nuclear	193
2.29	Atividade Sísmica	195
2.30	Sinais Fisiológicos	197
2.30.1	Foto-Pletismograma	197
2.30.2	Eletrocardiograma	200
2.30.3	Eletroencefalograma	200
2.30.4	Eletromiografia	201
2.30.5	Respiração	202
2.31	Imagem	203

3	Atuadores	209
3.1	Atuador Básico	210
3.2	Indicadores Visuais / Luminosos	212
3.2.1	Fio Eletroluminescente	213
3.2.2	Lâmpada Fluorescente	214
3.2.3	LEDs Monocromáticos	215
3.2.4	LEDs Dicromáticos	222
3.2.5	LEDs Policromáticos	223
3.2.6	LEDs com Controlador Digital	225
3.3	Mostradores	232
3.3.1	7-21 Segmentos	232
3.3.2	Matriz de Pontos	233
3.3.3	Mostrador Alfanumérico (LCD/VFD)	233
3.3.4	Mostrador Gráfico (LCD/OLED)	240
3.3.5	Válvula de Cátodo Frio (Nixie/Numitron)	243
3.3.6	Sinal Vídeo Analógico	243
3.3.7	Vídeo Composto (PAL)	244
3.3.8	Video Graphics Array (VGA)	245
3.4	Geradores de Som	247
3.4.1	Besouro	247
3.4.2	Piezoelétrico	248
3.4.3	Altifalante	250
3.5	Comutador de Circuitos	254
3.5.1	Relé Eletro-Mecânico	254
3.5.2	Relé Estado Sólido	255
3.6	Temperatura	259
3.6.1	Resistências de Aquecimento	259
3.6.2	Elemento de Peltier	260
3.7	Motores	262
3.7.1	Motor CC	262
3.7.2	Motor de Passo	266
3.7.3	Servo-Motor	269
3.8	Solenóide	274

Lista de Figuras

1.1	Arquitetura base de um sistema embebido genérico com entradas para sensores e saídas para atuadores.	1
1.2	Divisor de tensão.	5
1.3	Circuito divisor de tensão com R1 variável, à esquerda, e R2 variável, à direita.	6
1.4	Tensão à saída do circuito divisor de tensão em função do valor das resistências R1 e R2.	7
1.5	Circuito oscilador realizado com portas NOT do tipo Schmidt-trigger, com a função de conversão da capacidade em frequência. A frequência de oscilação depende do valor da capacidade. . .	8
1.6	Formas do espectro dos diferentes tipos de filtros RC.	9
1.7	Circuitos para implementação de filtros RC passivos.	9
1.8	Filtro RC passa-baixo.	10
1.9	Filtro RC passa-alto.	10
1.10	Circuito amplificador seguidor.	11
1.11	Montagens típicas com AMPOPs.	11
1.12	Ponte de Wheatstone.	12
1.13	Ponte de Wheatstone para compensação das perdas nos fios. .	13
1.14	Aspeto do circuito integrado em encapsulamento DIP8 (esquerda) e diagrama de blocos (direita).	13
1.15	Circuito clipping para limitação da amplitude/tensão máxima do sinal à entrada de microfone do telemóvel.	14
1.16	Colector e Dreno abertos.	16
1.17	Controlo de carga com uma porta open-colector.	17
1.18	Push-Pull.	18
1.19	Push-Pull 2.	18
1.20	Circuito H-bridge.	19
1.21	H-bridge.	19
1.22	Arquitectura de um pino de entrada e saída do MCU num Arduino. Esquema adaptado do manual do componente. . . .	20
1.23	Níveis de tensão para deteção dos valores 0 e 1 na entrada de uma porta do tipo CMOS.	21

1.24	Circuito para ligação de um botão de pressão a um pino de E/S do Arduino, sem pull-up, com pull-up e circuitos equivalentes para o botão premido e libertado.	22
1.25	Sinais de entrada e saída da ligação do botão de pressão sem (topo) e com (baixo) pull-up no pino de entrada.	24
1.26	Multiplexagem de uma matriz de 3x4 LEDs.	29
1.27	Charlieplexagem.	31
1.28	Interface UART.	33
1.29	uart tx rx.	34
1.30	Circuito adaptador de níveis de tensão CMOS/TTL para RS-232.	34
1.31	Interface SPI.	36
1.32	Transmissão e recepção num canal SPI.	36
1.33	Modos de funcionamento da interface SPI.	37
1.34	Transmissão e recepção num canal SPI.	37
1.35	Ligação de 2 Arduinos por canal SPI.	38
1.36	Interface I2C.	40
1.37	Transmissão e recepção num canal I2C.	41
1.38	Esquema de ligação entre um Arduino mestre e vários Arduínos escravos I2C.	42
1.39	Ligação entre dois Arduino via I2C.	43
1.40	Ilustração das ligações numa rede CAN com vários nós.	44
1.41	Comparação entre as normas existentes para o CAN e o modelo OSI.	45
1.42	Variação das tensões CAN+ e CAN- para transmissão de uma sequência de bits num barramento CAN.	46
1.43	Diagrama de olho.	46
1.44	Interface CAN.	47
1.45	Analizador CAN de 2 canais.	47
1.46	Interface 1-Wire.	48
1.47	Transmissão e recepção num canal 1-Wire.	48
1.48	Exemplos de transformadores de isolamento (esquerda) e esquema de aplicação de um transformador de isolamento.	50
1.49	Esquema de um opto-isolador.	51
1.50	Dois opto-isoladores.	51
1.51	Isolador CMOS.	51
1.52	Exemplo de quantização com 3 bits.	53
1.53	Exemplo dos diferentes ritmos de amostragem, e dos sinais digitais resultantes.	54
1.54	adc1.	56
1.55	adc2.	57
1.56	Níveis de tensão para diferentes resoluções.	59
1.57	Exemplos de sinais PWM para diferentes larguras de pulso.	60

1.58	DAC com escada de resistências 2R.	60
1.59	Tabela de Código Morse.	64
1.60	legenda.	66
1.61	legenda.	66
1.62	Diferentes tipos de tags RFID/NFC.	67
1.63	Modulo de comunicação LORA.	68
1.64	Módulo de comunicação Sigfox para Arduino.	69
1.65	Cabo Ethernet e Comutador (Switch) Ethernet.	70
1.66	Ligação de cabos Ethernet directo (à esquerda) e cruzado (à direita).	71
1.67	Router WiFi.	72
1.68	Telefone analógico.	73
1.69	Módulo de comunicações sem fios Bluetooth.	75
1.70	Slip-ring com suporte para 6 canais (esquerda), e detalhe das ligações internas de um slip-ring de 24 canais.	76
2.1	Exemplos de algumas grandezas físicas quantificáveis por sensores.	80
2.2	Condensador a funcionar como sensor de proximidade. Um corpo que altere as linhas de campo irá modificar a capacidade resultante.	83
2.3	Exemplo de uma bobina variável e esquema de possível de construção de uma bobina variável.	83
2.4	Aspeto de vários botões de pressão (esquerda) e representação simplificada (direita).	86
2.5	Modelo de construção de um botão de pressão.	87
2.6	Símbolo elétrico dos botões de pressão.	87
2.7	Circuito electrónico para ligar um botão de pressão ao Arduino.	87
2.8	Realização do circuito do botão de pressão em breadboard	88
2.9	Sinais produzidos pelo botão de pressão quando premido e gerado pelo Arduino. Em cima o sinal observado na saída do botão de pressão, onde é possível ver a presença de transições espúrias, e em baixo o sinal gerado pelo Arduino após a execução das rotinas de <i>debounce</i>	89
2.10	Realização do circuito do botão de pressão em breadboard (esquerda) e <i>breakout</i> (direita).	93
2.11	Sensor de movimento (esquerda) e esquema do modelo de construção simplificado (direita).	94
2.12	Sinal gerado pelo sensor de movimento.	96
2.13	Sensor de choque (esquerda) e esquema do modelo de construção simplificado (direita).	96
2.14	Sinal gerado pelo sensor de choque.	97

2.15	Modelo de construção simplificado (cima), sensor de inclinação com ampola de vidro (baixo à esquerda), e sensor de inclinação com esfera metálica para PCB (baixo à direita).	98
2.16	Sinal gerado pelo sensor de inclinação com bola de mercúrio (topo) e esfera metálica (baixo).	99
2.17	Sensor tátil baseado no transistor KSP13.	100
2.18	Sinal gerado pelo sensor tátil baseado no transistor KSP13. . .	100
2.19	Sensor tátil baseado num circuito oscilador.	101
2.20	Módulo baseado no TTP223 para realizar um sensor tátil. . .	102
2.21	Interruptor magnético com um íman na sua proximidade, e modelo de construção.	103
2.22	Sinal gerado pela ativação do sensor magnético.	103
2.23	Sensor de sentido de rotação e ligações da placa <i>breakout</i> do sensor rotação.	104
2.24	Modelo de construção do sensor do sentido de rotação (esquerda) e sinais gerados pela atividade do sensor (direita). . .	104
2.25	Potenciômetro logarítmico de 10k ohm (esquerda), e o seu símbolo elétrico (direita).	106
2.26	Esquema de ligações do potenciômetro ao Arduino Uno.	107
2.27	Montagem do potenciômetro num Arduino Uno.	108
2.28	Modelo de construção de um Joystick de 2 eixos.	109
2.29	teclado de membrana	110
2.30	Esquema de ligações do teclado 4x4.	111
2.31	Diagramas de duas realizações de teclados analógicos.	116
2.32	Placa de mostrador alfanumérico com teclado analógico. . . .	117
2.33	Optómetro	118
2.34	Aspeto e símbolo elétrico de uma LDR.	119
2.35	LDR soldada numa <i>breadboard</i>	119
2.36	Símbolos elétricos do fotodiodo e do fototransistor.	120
2.37	Vários sensores do tipo fotodiodo e fototransistor.	121
2.38	Placa de demonstração do sensor de chama.	123
2.39	Esquema elétrico do circuito da placa de demonstração do sensor de chama.	124
2.40	Gráfico com a relação entre a distância da chama da vela ao sensor e a tensão analógica.	125
2.41	Modelo equivalente de um sensor PIR.	126
2.42	Placas breakout com sensores PIR, HC-SR505 à esquerda e centro, e Groove PIR à direita.	127
2.43	Esquema elétrico do circuito da placa breakout HC-SR505 para um sensor PIR.	127
2.44	Diagrama simplificado do circuito integrado EG4001 para realiza o condicionamento de sinal de sensores PIR.	128
2.45	modelo do sensor piezoelétrico.	129

2.46	Sensor piezoelétrico.	129
2.47	Ligação de um sensor piezoelétrico através de um ao circuito condicionador de sinal.	130
2.48	Exemplo com batidas de diferentes materiais num sensor pie- zoelétrico a um Arduino.	131
2.49	Ligação de um sensor piezoelétrico a um Arduino.	131
2.50	Modelo de funcionamento e aspeto de um sensor de efeito de Hall.	133
2.51	Modelo de funcionamento e aspeto de um sensor de efeito de Hall.	133
2.52	Pickup de guitarra elétrica (esquerda) e modelo de construção de um <i>pickup</i> (direita).	134
2.53	Resistência de shunt usada para medir correntes em circuitos DC.	136
2.54	Diagrama de blocos funcional do INA190 da Texas Instru- ments. Tirado do manual de referência do componente.	136
2.55	Analogia entre um copo de cerveja e as componentes de potência em CA.	137
2.56	Medidor de potência para a rede elétrica (esquerda) e es- quema de ligações para ligar a carga a medir e a interface série assíncrona para ligação a um Arduino.	138
2.57	Aparato para medida da potência de um computador em fun- cionamento.	142
2.58	Medida da tensão, corrente e potência de um computador a executar diversos programas.	142
2.59	Medida da tensão, corrente e potência de um computador a executar diversos programas.	142
2.60	Escala de temperatura em graus Célsius, graus Fahrenheit e Kelvin.	143
2.61	Modelo de construção e aspeto de um termopar tipo K.	145
2.62	Diagrama de blocos do AD8495 e montagem para ligação de um termopar a uma entrada analógica de um Arduino.	146
2.63	Modelo de construção e aspeto de um termopar tipo K.	146
2.64	Sensor de temperatura digital DS18S20.	149
2.65	sensor líquidos.	152
2.66	Variação do ponto de orvalho em função da temperatura do ar, da humidade relativa e da temperatura da superfície.	153
2.67	sensor humidade.	154
2.68	Escala níveis de pH.	155
2.69	Modelo de construção do sensor de pH.	156
2.70	Modelo do sensor de pHFET.	156
2.71	medidores ph.	157
2.72	medidores ph.	157

2.73	Circuito condicionador de sinal para um sensor de pH.	158
2.74	Acelerómetro analógico ADXL335.	159
2.75	Diagrama temporal dos 3 canais medidos pelo acelerómetro ADXL335.	159
2.76	Módulo GPS NEO-6M.	165
2.77	Esquema de ligações em breadboard do módulo GPS.	166
2.78	Montagem de ensaio do módulo GPS com um Arduino Uno.	166
2.79	Mensagens de localização enviadas pelo módulo GPS.	167
2.80	Localização do Arduino sobre a imagem de satélite no Google Earth.	167
2.81	Mensagens de localização decodificada usando a biblioteca TinyGPS++.	168
2.82	Microfone montado numa placa <i>breakout</i> e circuito para pola- rização do microfone.	169
2.83	Sinal áudio adquirido pelo microfone.	170
2.84	Aspeto de um hidrofone.	171
2.85	pressão conceito.	172
2.86	BMP180 e BME280.	172
2.87	Esquema dos circuitos nas placas BMP180 e BME280.	172
2.88	Extensómetros.	174
2.89	Extensómetro (esquerda) e circuito elétrico equivalente (direita).	174
2.90	Montagem do extensómetro e respetivo circuito de condicio- namento de sinal a um Arduino Uno.	176
2.91	Montagem da balança.	176
2.92	Variação do peso ao longo do ensaio.	179
2.93	Caudalímetro.	180
2.94	Sensores de concentração de gases.	181
2.95	Parte da frente de um sensor de distância (esquerda) e parte traseira do sensor de distância (direita).	184
2.96	Parte da frente de um sensor de distância (esquerda) e parte traseira do sensor de distância (direita).	187
2.97	Ligação de um sensor de distância de ultra-sons a um Arduino Uno.	187
2.98	Sensor de distância baseado em radiação laser.	189
2.99	Ligação de um sensor de distância baseado em radiação laser a um Arduino Uno.	189
2.100	Fotodiodo tipo PIN BPW34.	193
2.101	Detetor Geiger-Müller baseado em tubo de gás.	193
2.102	Detetor de radiação baseado em tecnologia semicondutora (PIN).	194
2.103	Geofone (esquerda) e respetivo suporte para condução das vi- brações do solo ao corpo do sensor (direita).	196
2.104	Medidor de pressão arterial comercial.	197

2.105	Oxímetro de pulso para medição do ritmo cardíaco (esquerda) e esquema de funcionamento do sensor PPG.	198
2.106	Oxímetria de pulso usando um LED e um fotodiodo.	198
2.107	ecg sensor2.	199
2.108	ecg sensor MAX30100.	199
2.109	Circuito para o sensor MAX30100.	200
2.110	ecg sensor MAX30100.	200
2.111	Propagação do sinal elétrico cardíaco ao longo do corpo. . . .	201
2.112	Morfologia do sinal do coração.	201
2.113	Ritmo cardíaco via eletrocardiograma.	202
2.114	Mapa de posicionamento dos eletrodos para obtenção de EEG. .	202
2.115	Sensor EEG.	202
2.116	ecg sensor via eletromiografia.	202
2.117	Câmara OV7670.	203
2.118	Ligação da câmara OV7670 a um Arduíno.	204
2.119	Circuito de demonstração da câmera OV7670.	205
3.1	Escala de tons para diferentes temperaturas de cor.	213
3.2	Decaimento da energia radiada com a distância, de acordo com a lei do inverso do quadrado.	214
3.3	Fio eletroluminescente em funcionamento e respetivo modelo de construção.	215
3.4	Lâmpada fluorescente de ultra-violetas.	215
3.5	Modelo e símbolo elétrico de um LED.	215
3.6	Demonstração da animação com LEDs.	219
3.7	Demonstração da montagem do diodo laser.	221
3.8	Detalhes da construção de um LED dicromático (esquerda) e PCB de demonstração (direita).	228
3.9	Ligação da placa do LED bicolorido a um Arduíno Uno. . . .	229
3.10	Esquema interno de um LED RGB.	229
3.11	Montagem para demonstração do LED RGB.	229
3.12	Detalhe do LED pisca-pisca de 7 cores.	230
3.13	Detalhe de um Neopixel à esquerda e diferentes arranjos de circuitos com LEDs Neopixel.	230
3.14	Detalhe de um Neopixel à esquerda e diferentes arranjos de circuitos com LEDs Neopixel.	230
3.15	Matriz 8x8 de <i>Light Emitting Diodes</i> (LEDs) Neopixel. . . .	231
3.16	Mostrador alfanumérico de 15 segmentos.	232
3.17	Mostrador alfanumérico LCD 16x2.	235
3.18	Mostrador alfanumérico LCD 20x4.	240
3.19	nokia5110 demo.	241
3.20	Ilustração de usos do TFT 240x240.	241
3.21	Sprites para jogo de xadrês [Wikipedia].	242

3.22	Esquema de um monitor CRT.	245
3.23	Esquema de varrimento do ecrã.	246
3.24	Esquema de ligações de um Arduino Uno a um conector VGA.	246
3.25	Beeper.	247
3.26	Dois tipos de besouro.	248
3.27	Emissor de som piezoelétrico.	249
3.28	Montagem piezo.	250
3.29	Esquema de construção de um altifalante.	251
3.30	Circuito amplificador estéreo de 3W.	251
3.31	Símbolos que representam o tempo de cada nota na armadura de clave: semibreve, mínima, colcheia, e semicolcheia.	252
3.32	Partitura da música popular “Atirei o pau ao gato”.	253
3.33	Símbolo de um relé (esquerda) e o interior de um relé 1 circuito com duas posições (direita).	254
3.34	Circuito de demonstração de um relé ligado a um Arduino.	255
3.35	Controlo de uma lâmpada de 220V com um TRIAC.	256
3.36	Relé de estado sólido.	257
3.37	Circuito de demonstração do relé de estado sólido (SSR-2F4805) ligado a um Arduino Uno.	258
3.38	Circuito de demonstração do relé de estado sólido (SSR-2F4805) ligado a iluminador de 220 V, controlado por um Arduino Uno.	258
3.39	Resistência de aquecimento de um ferro de soldar.	259
3.40	Elemento de Peltier (TEC).	260
3.41	Ligação de um Elemento de Peltier a uma Ponte-H.	261
3.42	Pequeno motor DC.	262
3.43	Funcionamento do motor DC.	263
3.44	Especificações técnicas de um motor DC equipado com redução.	264
3.45	Etapas do funcionamento do motor de passo.	267
3.46	Motores de passo.	267
3.47	Circuito de demonstração de um motor de passo.	268
3.48	Servo-motor.	269
3.49	Esquema elétrico e protótipo da demonstração do servo-motor.	270
3.50	Sinais de comando do servo-motor para as posições 0º, 90º e 180º.	273
3.51	Esquema de construção de uma eletro-valvula (solenóide).	274
3.52	Ligação de uma eletro-válvula e um sensor de caudal a um Arduino Uno.	275
3.53	Eletro-valvula ou solenóide.	275

Lista de Tabelas

1.1	Algumas referências de transistores e as principais características.	16
1.2	Combinações dos sinais de E/S para ativar cada LED usando Charlieplexagem.	31
1.3	Pinos dedicados à interface SPI em várias placas Arduino.	37
2.1	Métodos da classe <code>PushButton</code>	90
2.2	Atributos privados da classe <code>PushButton</code> .	91
2.3	Pinos da placa do botão de pressão e sua descrição.	94
2.4	Comparação das vantagens e desvantagens entre fotodiodos e fototransistores.	122
2.5	Resumo das características dos circuitos com sensores PIR.	127
2.6	Resumo dos valores dos comandos e respostas para o medidor de energia elétrica PZEM004.	138
2.7	Conversão entre as diferentes escalas de temperatura.	144
2.8	Caraterísticas dos vários tipos de termopar	145
2.9	Comparação das vantagens e desvantagens das diferentes tecnologias de sensores para medição de temperatura.	151
2.10	Correspondência entre o peso no prato da balança e a medida lida do sensor.	177
2.11	Tipos de sensores de gases.	182
3.1	Comprimento de onda da radiação gerada por diferentes tipos de LEDs e a sua tensão de funcionamento.	216
3.2	Corrente e potência consumida por um LED laser quando ligado a diferentes pinos de um Arduino Uno.	221
3.3	Cor resultante da combinação das várias componentes de cor.	223
3.4	Alguns dos métodos disponibilizados pela classe <code>Adafruit.NeoPixel</code> .	227
3.5	Comandos de configuração do LCD no modo 4 bits	236

Lista de Programas

1.1	ReadAnalogInput.ino	4
1.2	Sketch para simular o funcionamento do circuito 555	14
1.3	Configuração dos pinos de entrada	23
1.4	Configuração de um pino de entrada digital	25
1.5	Exemplo de escrita de todos os bits do porto D	26
1.6	Exemplo de atendimento a interrupções com objetos.	27
1.7	multiplexingDemo.ino	29
1.8	Demonstração da Charlieplexing com LEDs	32
1.9	Demonstração da programação do barramento SPI	39
1.10	i2cDemo.ino	42
1.11	adcDemo.ino	56
1.12	Exemplo de uso de um sinal PWM.	59
1.13	Sketch para realizar filtragem passa-baixo.	61
1.14	Sketch para realizar filtragem passa-baixo.	62
1.15	Mensagens de configuração de um MODEM BT	74
1.16	Mensagens de configuração de um MODEM BT	74
1.17	Mensagens de configuração de um MODEM BT	75
2.1	Listagem da classe basicSensor	81
2.2	Listagem do Sketch com demonstração da utilização do botão de pressão.	89
2.3	Sketch para ensaio do sensor de movimento.	95
2.4	Listagem do Sketch para demonstração do sensor de toque. . .	101
2.5	Listagem do Sketch para o botão rotativo	104
2.6	Listagem para demonstração do uso de um potenciômetro. . .	106
2.7	JoyStick.ino	109
2.8	keypad4x4.ino	112
2.9	keypad.h	113
2.10	keypad.cpp	113
2.11	Demonstração da utilização de um teclado analógico.	117
2.12	Sketch de demonstração da LDR.	120
2.13	IR_Receiver.ino	121
2.14	Demonstração do sensor de chama.	123
2.15	Demonstração do sensor de chama.	130

2.16	Listagem do Sketch para ler as medidas do consumo de energia com um medidor PZEM004	139
2.17	Sketch exemplo <code>serialthermocouple</code>	147
2.18	Exemplo de leitura do valor da temperatura do sensor 1-wire DS18S20	150
2.19	Leitura e processamento dos dados do acelerómetro ADXL335 .	159
2.20	GPS.ino	166
2.21	Exemplo da aplicação de um extensómetro e do HC711 para funcionarem como uma balança de precisão.	175
2.22	Leitura e processamento dos dados do acelerómetro ADXL335 .	183
2.23	Leitura e processamento dos dados do acelerómetro ADXL335 .	186
2.24	Leitura e processamento dos dados do acelerómetro ADXL335 .	189
2.25	hr_sensor.ino	198
2.26	ov7670.ino	204
3.1	Listagem da classe <code>basicAtuator</code>	210
3.2	Kitt #1	217
3.3	Kitt #2	218
3.4	Listagem do Sketch para demonstração da classe Led	220
3.5	Listagem do Sketch LedBiColor.ino	222
3.6	Listagem do Sketch LedRGB.ino para demonstrar o uso do LED RGB	224
3.7	Listagem do Sketch Neopixel.ino para demonstrar o uso do LED NeoPixel	226
3.8	Listagem do ficheiro com a definição da classe Lcd44780	236
3.9	Listagem do Sketch de demonstração do uso da classe Lcd44780	239
3.10	Passive_Buzzer.ino	242
3.11	Passive_Buzzer.ino	248
3.12	Sound_Digital.ino	249
3.13	Sound_Analog.ino	252
3.14	Relay.ino	254
3.15	Sketch MotorDC.ino	265
3.16	Sketch StepMotor.ino	267
3.17	Sketch ServoDemo.ino	270

Nomenclatura

Acronimos e Abreviaturas

ALU	Arithmetic Logic Unit
ASAP	As Soon As Possible
ASIC	Application Specific Integrated Circuit
CAD	Computer-Aided Design
CPU	Central processing Unit
DSP	Digital Signal Processing
FFT	Fast Fourier Transform
FPGA	Field-Programable Gate Array
FSM	Finite State Machine
GPU	Graphics Processing Unit
HDL	Hardware Description Languages
I/O	Input/Output
LUT	Look-Up Table
MAC	Multiply Accumulate
MSb	Most Significant bit
NaN	Not a Number
RAM	Random Access Memory
ROM	Read Only Memory
VHDL	Very-high-speed integrated circuit Hardware Description Language

List of Symbols

$O(n)$ Complexity order n

1

Interfaces de Entrada/Saída

Uma das grandes diferenças entre programar um computador e um sistema embebido consiste na utilização de diferentes periféricos e interfaces. Enquanto que um computador é usado para processar grandes volumes de informação através de periféricos com interfaces bem definidas, tais como o teclado, rede ethernet, e monitor; já um sistema embebido processa menos dados mas tem as suas interfaces customizadas, combinando diferentes tipos de sensores, atuadores e módulos, normalmente usados para monitorizar e controlar um processo físico. A figura 1.1 ilustra um sistema genérico com sensores e atuadores usados para controlar um processo físico.

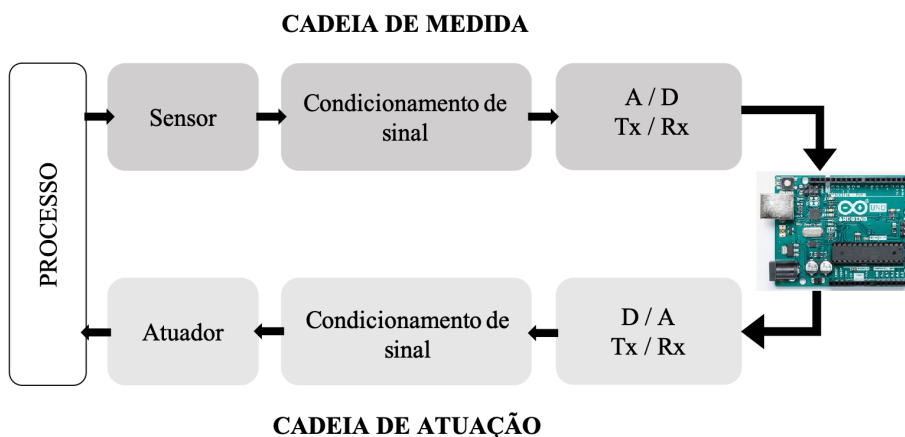


Figura 1.1: Arquitetura base de um sistema embebido genérico com entradas para sensores e saídas para atuadores.

As interfaces de um sistema embebido servem para interagir com componentes, módulos, ou até outros sistemas. Existem dois tipos de interfaces: analógicas - para sensores, e digitais - para ligar módulos ou sub-sistemas com alguma complexidade.

De um modo geral, as interfaces analógicas são usadas para adaptações de sinais, e são uma solução económica para ligação de componentes a um sistema embebido uma vez que a grande maioria dos Microcontrolador - *Micro-Controller Units* (MCUs) incorpora um conversor analógico-digital. Outra motivação é o facto da maioria dos fabricantes de sensores optarem por fabricar apenas o mecanismo responsável pela tradução da grandeza física por uma variação de uma grandeza eléctrica qualquer que o represente sem estarem obrigados a incluir uma interface digital. Por este motivo, habitualmente os sensores analógicos necessitam de circuitos condicionadores de sinal.

Por outro lado, as interfaces digitais realizam tarefas mais complexas que as interfaces analógicas, e são habitualmente usadas para implementar protocolos (mais ou menos sofisticados) para transferência de dados entre elementos. São interfaces mais caras pois necessitam de um controlador dedicado. Os MCUs atuais incorporam alguns controladores de interfaces digitais dedicadas a algumas destas interfaces para facilitar a sua ligação e programação. As interfaces digitais são habitualmente ligadas numa configuração mestre-escravo (master-slave).

1.1 Circuitos Condicionadores de Sinal

Os circuitos condicionadores de sinal servem para fazerem de ponte entre a saída do sensor e o circuito que vai processar o valor medido. O circuito condicionador pode realizar várias funções:

- amplificação de sinais de muito pequena amplitude, na ordem dos μv ou mv ;
- filtragem do sinal para remoção de bandas de frequência não desejadas;
- adaptação de impedâncias para que o sistema embebido não tenha impacto na medida;
- modulação e desmodulação de sinais (por exemplo, converter uma variação de tensão em frequência para aumentar a imunidade ao ruído e gama dinâmica da variação)¹;

¹Um sinal analógico entre 0-5 V digitalizado com um ADC de 10 bits tem um passo de quantização de $(5 - 0)/2^{10} = 5/1024 = 4,883mV$, o que significa que variações menores que esse passo não são observadas. Como com um sistema embebido é possível medir variações de frequência na ordem da dezena de kHz a MHz, a utilização de um circuito conversor tensão-frequência aumenta a resolução da medida do sistema.

- conversão analógico digital e digital-analógico.

O circuito condicionador é escolhido em função do tipo de sensor:

- gerador - o sensor produz uma tensão entre os seus terminais;
- resistivo - o sensor comporta-se como uma resistência variável, que varia em função da grandeza física observada.

Os sensores geradores são caracterizados por:

- produzirem sinais de pequena amplitude que necessitam de amplificação;
- produzirem sinais de muito baixa frequência, o que não permite o desacoplamento DC com um filtro passa-baixo passivo (condensador e resistência) por influenciar a impedância da montagem;
- nalgumas situações o sinal não é fraco, mas o sensor tem impedância elevada (equivalente a uma resistência em série elevada à saída, que cria um circuito equivalente a um divisor de tensão com a impedância do circuito de medida).

Os sensores resistivos são caracterizados por necessitarem de uma tensão de alimentação auxiliar aplicada nos seus terminais, para poder ser lida uma variação de tensão em função da sua resistência. No entanto, como o sensor passa a ser percorrido por uma corrente, pode haver influência na medida devido ao autoaquecimento do sensor. O circuito deve ser dimensionado de tal forma que minimize a corrente que percorre o sensor para ele poder funcionar. Existem dois métodos para a medição de um sensor resistivo:

- deflexão: mede-se a tensão aos terminais da resistência ou a corrente que a atravessa (usando um circuito divisor de tensão);
- comparação: baseiam-se na comparação de valores de resistências conhecidos, também conhecidos como pontes de medida.

1.2 Interfaces Analógicas

As interfaces analógicas são usadas para que um sistema embebido possa receber e produzir sinais para componentes analógicos, por exemplo, medida e controlo de temperatura. As interfaces analógicas são as mais habituais para adquirir sinais de sensores analógicos, uma vez que os processos observados na natureza não apresentam descontinuidades ao longo do tempo.

Muitos dos sensores usados em sistemas embebidos traduzem uma grandeza física numa variação de resistência. Uma resistência pode ser avaliada

através da medição da queda de tensão aos seus terminais, no entanto é necessário adicionar um circuito ao sensor para permitir medir a diferença de potencial nos seus terminais. Nos atuadores, a maior limitação são os níveis máximos de tensão e corrente fornecidos pelos MCUs serem insuficientes para actuar cargas como por exemplo lâmpadas, motores ou resistências de aquecimento. Para estes sistemas é necessário haver um circuito capaz de receber um sinal de comando do MCU e realizar a comutação adequada da carga.

O Arduino Uno possui 6 pinos de entrada exclusivos para tensões analógicas (A0-A5), onde um valor de tensão entre 0-5 V é traduzido num valor numérico entre 0-1023 (10 bits). A programação da interface analógica passa por indicar a instrução para realizar a conversão num determinado pino de entrada. A listagem 1.1 mostra um *Sketch* que “lê” um valor de um pino analógico A0 e mostra o seu valor na consola. Se o traçador gráfico for aberto é possível visualizar a forma de onda criada pela variação do sinal analógico presente na entrada A0. Se não existir nenhum sinal ligado a A0, apenas ruído será mostrado.

Listagem 1.1: ReadAnalogInput.ino

```
// select the input pin
#define analogPin A0

int analogValue = 0; // variable to store the sampled analog value

void setup() {
  Serial.begin(9600); // config serial port 9600 baud
  while (!Serial) ; // wait for serial port to connect.
}

void loop() {
  analogValue = analogRead(analogPin);
  Serial.println(analogValue);
}
```

1.2.1 Conversão Impedância-Tensão

Um sensor resistivo traduz um grandeza física por uma variação de impedância, ou resistência. Para quantificar esta resistência é necessário que ela seja percorrida por uma corrente (malha fechada), de modo a criar uma queda de tensão aos seus terminais. O circuito divisor de tensão é o circuito mais simples que pode ser realizado para avaliar o valor de uma resistência, medindo a queda de tensão causada por um sensor resistivo. Se pensarmos na montagem do divisor de tensão e na expressão da tensão de saída, temos a relação entre uma tensão à entrada e a variação de uma das resistências. O valor da outra resistência deve ser escolhido com cuidado e depende das características do sensor usado. Habitualmente, o valor de 10 $k\Omega$ é razoável para a maioria das aplicações, não devendo ser desprezada a informação fornecida pelo fabricante.

A figura 1.2 mostra o esquema elétrico do divisor de tensão, composto por duas resistências, R1 e R2, sendo uma delas a resistência variável do

sensor, e a outra uma resistência fixa de polarização. A razão de decréscimo depende da relação entre o valor das duas resistências. Em relação à tensão de entrada V_i , a tensão de saída do divisor de tensão é:

$$V_o = V_i \times \frac{R_2}{R_1 + R_2} \quad (1.1)$$

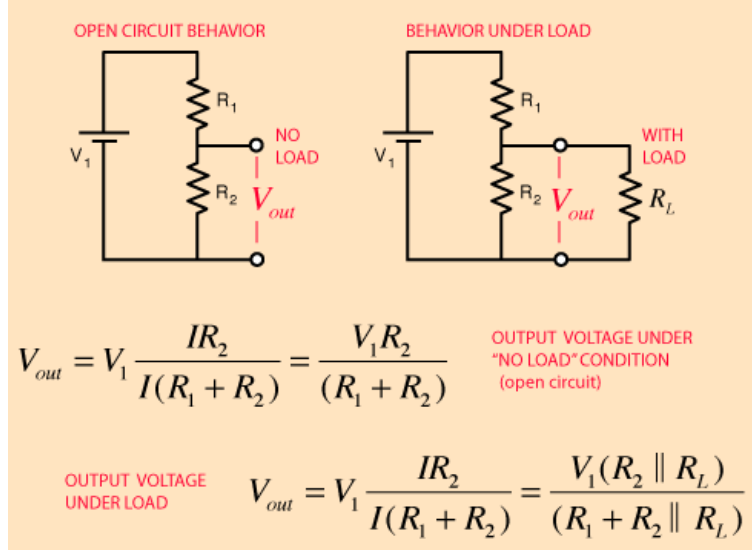


Figura 1.2: Divisor de tensão.

Se R_1 for o sensor resistivo, e R_2 a resistência fixa, a tensão na saída do divisor de tensão aumenta com o decréscimo de R_1 .

Se R_2 for o sensor, o valor de V_o aumenta com o aumento de R_2 . Em ambos os casos, com a variação do valor da resistência do sensor observa-se um andamento não linear. Importa escolher uma resistência de polarização que sirva ao sensor, ou seja, que permita fazer uma leitura o mais linear possível para evitar ter de realizar ajustes aos valores medidos. Para linearizar uma medida pode-se:

- calcular a função inversa correspondente à curva característica do sensor;
- aproximar via função paramétrica ou por troços;
- por correspondência com uma tabela.

Se para valores elevados de R_1 o andamento da curva é quase linear, a escolha de uma resistência de valor elevado tem outras implicações a nível do circuito:

- Consome menos corrente, quando o sensor apresenta baixa resistência;

- Mais sensível ao ruído e interferência eletromagnética;
- Menor gama dinâmica.

A figura 1.3 mostra o circuito com a resistência variável nas duas posições. Em termos práticos, opta-se pela montagem que mais facilmente traduz a grandeza medida. Quando a resistência variável tem um valor próximo de zero, se for ligada a VCC a tensão de saída do circuito é também próxima de VCC.

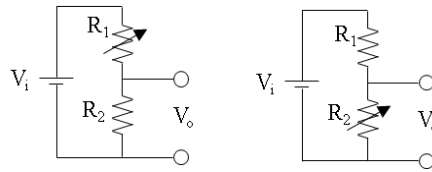


Figura 1.3: Circuito divisor de tensão com R1 variável, à esquerda, e R2 variável, à direita.

A figura 1.4 mostra a evolução da tensão em função da variação dos valores das resistências de um sensor considerando um conjunto de valores de resistências de polarização. Para simular o andamento da tensão para sensores com diferentes variações de resistência, nos gráficos de cima considera-se que os valores da resistência variável variam entre 0 e 1 $kH\Omega$. Nas figuras em baixo, a resistência varia entre 0 e 10 $MH\Omega$.

A maior desvantagem do circuito divisão de tensão é a necessidade de usar uma fonte de alimentação bastante estável e precisa, pois a tensão medida na saída do circuito vai ser influenciada pela alimentação do circuito, por exemplo, 4,95 V em vez de 5 V. Para aumentar a qualidade da leitura deve-se amostrar também o valor da tensão de alimentação do circuito e calcular o valor de saída considerando a tensão medida em vez de assumir um valor ideal e constante.

1.2.2 Conversão Capacidade-Frequência

A forma mais simples de aferir a variação de capacidade é através da realização de um circuito oscilador (por exemplo um *ring-oscillator* realizado com portas lógicas NOT do tipo *Schmidt-trigger*), onde a saída é ligada à entrada através de uma malha RC. A figura 1.5 mostra o esquema do circuito oscilador baseado apenas numa porta NOT, o sensor, e uma resistência.

Quando a entrada da porta lógica NOT é 0, a saída vai ser 1, o que vai contrariar o valor inicial da entrada, e fazer com que a saída passe a 0. A malha RC vai fazer com que o valor da entrada demore a convergir para 0 ou 1, carregando o condensador lentamente. Quando maior for a capacidade

1.2. INTERFACES ANALÓGICAS

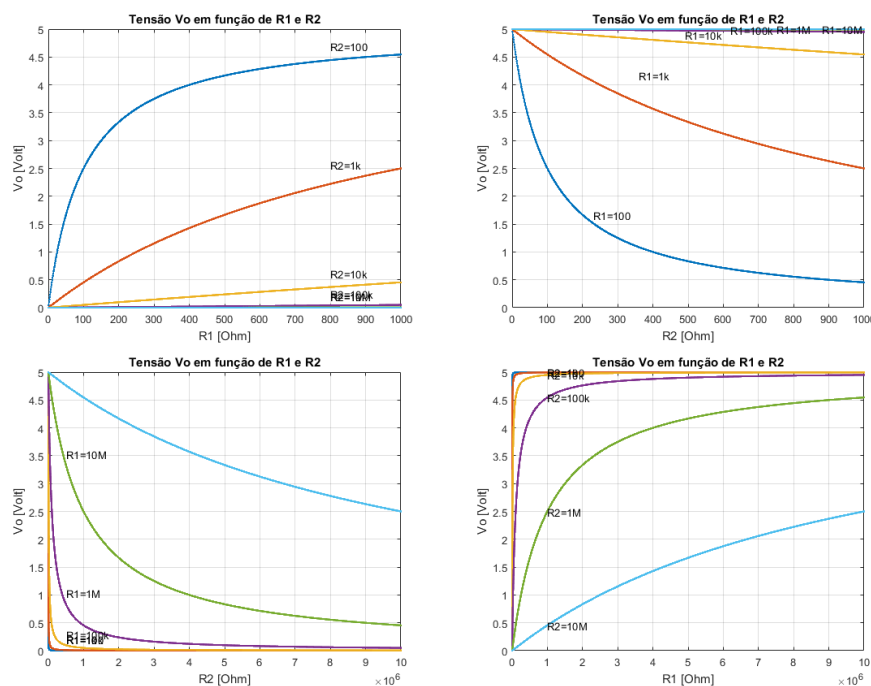


Figura 1.4: Tensão à saída do circuito divisor de tensão em função do valor das resistências R_1 e R_2 .

do condensador, mais lentamente ele carregará, e menor será a frequência de oscilação.

A frequência de oscilação depende da constante de tempo de carga e descarga do condensador imposta pelo produto $\tau = R \times C$. A frequência de oscilação é inversamente proporcional à capacidade. As portas do tipo *Schmidt-trigger* fazem com que apenas o sinal de entrada seja considerado após atingir um limiar de disparo próximo dos extremos (VCC ou GND). Este tipo de portas é muito usado para eliminar transições espúrias causadas por ruído.

A programação necessária num Arduino para aferir a frequência do sinal de saída do oscilador consiste em contar o número de pulsos observados durante um determinado período de tempo. Isto é conseguido com associação de uma rotina de interrupção ao sinal de entrada no sistema embebido, e da programação de um temporizador interno para realizar contagem precisa do tempo.

1.2.3 Filtragem RC

Um sinal elétrico pode ter contribuições em diferentes frequências ou bandas, por exemplo, sinal áudio (música) com tons graves e agudos. Os filtros servem para atenuar um determinado conjunto de frequências de um sinal, e podem

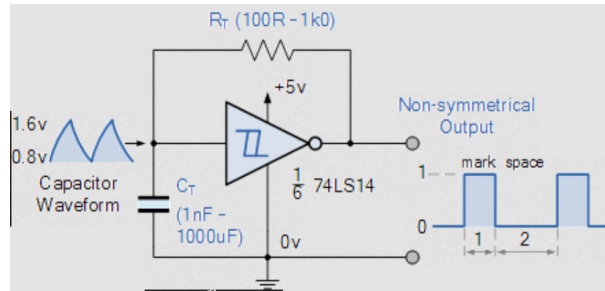


Figura 1.5: Circuito oscilador realizado com portas NOT do tipo Schmitt-trigger, com a função de conversão da capacidade em frequência. A frequência de oscilação depende do valor da capacidade.

ser ativos ou passivos.

Os filtros passivos analógicos são construídos com componentes passivos tais como resistências (R), condensadores (C) e bobines (L), Sendo possível criar filtros com diferentes combinações destes componentes: RC, RL, LC, e RLC. O número de elementos reativos (condensadores e bobines) determinam a ordem do filtro, por exemplo, os filtros RC ou RL são de primeira ordem, e o filtro RLC de segunda ordem. É possível combinar vários filtros RC idênticos para aumentar a ordem e a atenuação causada.

Em termos práticos, cada componente vai ter um comportamento diferente de acordo com a frequência do sinal que o percorre:

- resistência - produz atenuação constante em todo o espectro de frequências.
- condensador:
 - em DC ou muito baixas frequências: o condensador comporta-se como um circuito aberto (pode ser visto como um interruptor que está ligado muito pouco tempo);
 - em altas frequências comporta-se como um curto-circuito;
- bobina:
 - em DC ou muito baixas frequências: comporta-se como um fio, ou curto-circuito;
 - em altas frequências: comporta-se como um circuito aberto;

Um filtro é caracterizado principalmente pela sua frequência de corte e atenuação, ou seja, quando começa a produzir efeito no espectro do sinal e em que quantidade, respectivamente. Existem 4 tipos elementares de filtros analógicos:

- Passa-baixo: atenua as componentes de frequência acima da frequência de corte;

- Passa-alto: atenua as componentes de frequência abaixo da frequência de corte;
- Passa-banda: atenua as componentes de frequência fora de um intervalo de frequências de corte. Ou seja, apenas deixa passar uma determinada gama de frequências entre as duas frequências de corte;
- Rejeita-banda: atenua as componentes de frequência dentro de um intervalo de frequências de corte;

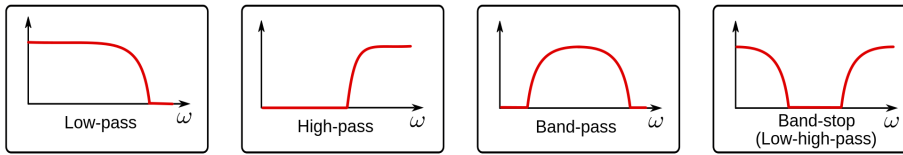


Figura 1.6: Formas do espectro dos diferentes tipos de filtros RC.

O filtro passa-banda é uma composição de um filtro passa-baixo seguido de um passa-alto, em que a frequência de corte do filtro passa-baixo é inferior à do passa-alto. O filtro rejeita-banda é construída mesma forma, mas a frequência de corte do filtro passa-baixo é superior à do passa-alto. Para ambos, as frequências de corte de cada filtro são calculadas da mesma forma, e são determinadas pelos valores de R e C : $f_c = 1/(2\pi.R.C)$.

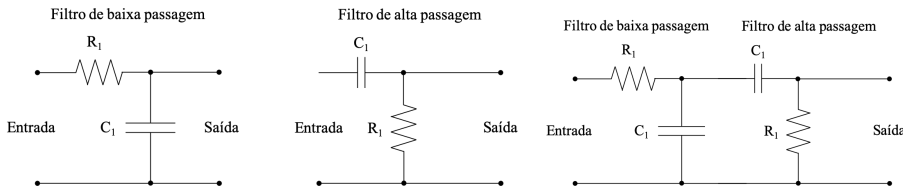


Figura 1.7: Circuitos para implementação de filtros RC passivos.

A figura 1.7 apresenta o circuito de um filtro passa-baixo (esquerda) e passa-alto (direita)

Habitualmente, os filtros de primeira e segunda ordem RC são suficientes para realizar as funções de filtragem num sistema embebido. De seguida apresentam-se as diferentes representações do espectro do filtro, e o respectivo circuito elétrico. Existem outras características no sinal que são alteradas pelos filtros, tal como desvios de fase e atraso de grupo, que para algumas aplicações devem ser observadas.

No filtro passa-baixo, o condensador funciona como um circuito aberto para as baixas frequências, não tendo influência nas baixas frequências do

CAPÍTULO 1. INTERFACES DE ENTRADA/SAÍDA

sinal. No entanto para altas frequências o condensador do passa-baixo funciona como um curto-circuito a GND, atenuando o sinal na saída. No passa-alto acontece o oposto: o condensador em baixas frequências ao comporta-se como um circuito aberto. Para as altas frequências comporta-se como um curto-circuito, logo deixa passar o sinal da entrada para a saída.

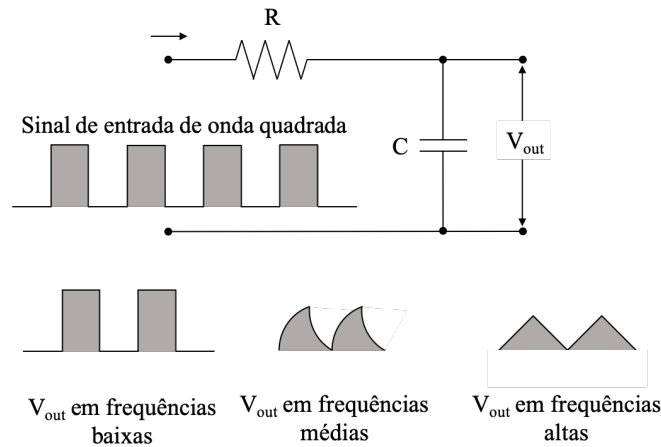


Figura 1.8: Filtro RC passa-baixo.

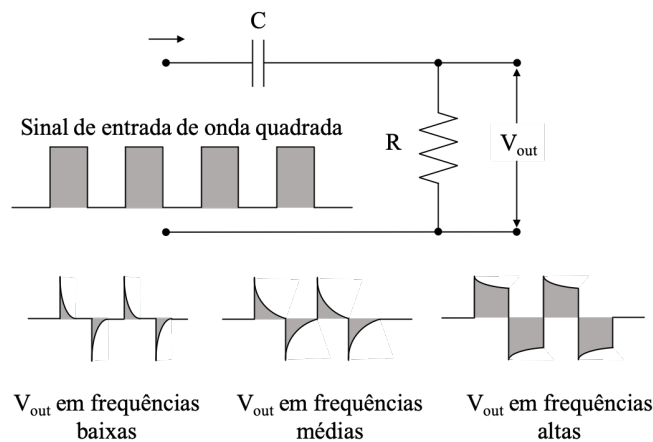


Figura 1.9: Filtro RC passa-alto.

Os filtros ativos são construídos com pelo menos um elemento ativo, normalmente um AMPOP. A figura 1.10 mostra um circuito seguidor, que serve para isolar diferentes etapas de um circuito, com impedâncias diferentes. A figura 1.11 mostra exemplos de diferentes montagens de circuitos com AM-POPs, incluindo filtros. Numa perspectiva bastante redutora e simplista de realizar um filtro ativo passa por colocar um circuito amplificador após o filtro passivo RC. Os detalhes sobre o funcionamento dos AMPOPS e dos

1.2. INTERFACES ANALÓGICAS

filtros ativos são extensos e por isso são remetidos para outras publicações sobre o tema.

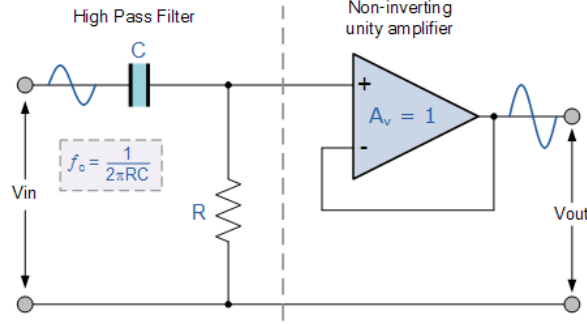


Figura 1.10: Circuito amplificador seguidor.

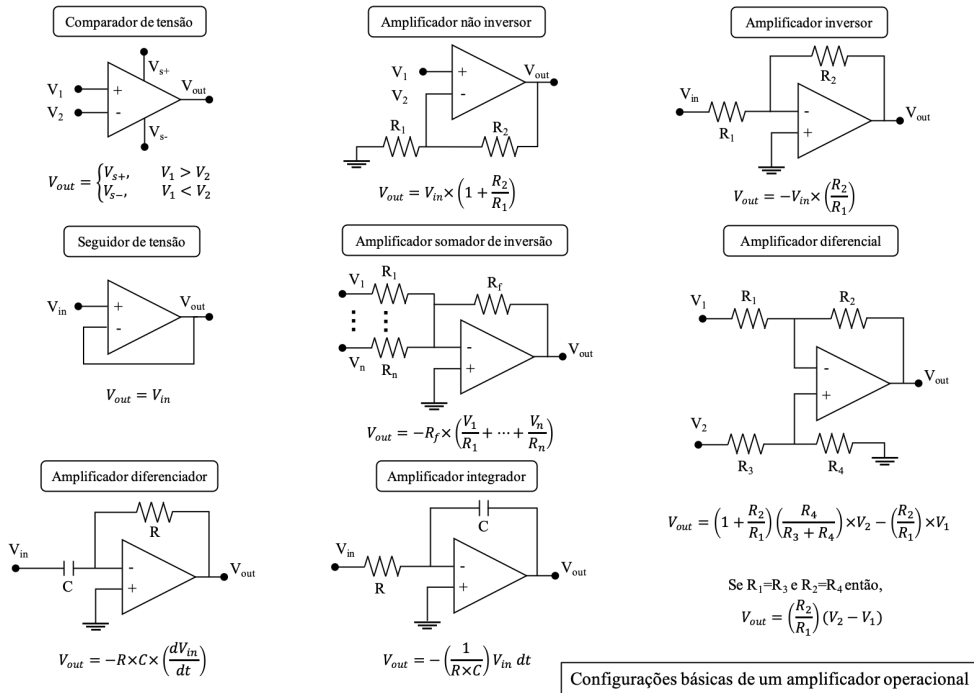


Figura 1.11: Montagens típicas com AMPOPs.

1.2.4 Ponte de Wheatstone

A Ponte de Wheatstone é uma montagem composta por dois divisores de tensão, onde é observada a tensão nos dois pontos centrais. Num dos divisores de tensão é colocado o sensor resistivo a aferir e no outro uma resistência de calibração.

Quando o sensor estiver estável, ou seja, com um valor constante, a resistência de calibração é ajustada de tal forma que a diferença de tensão entre os dois pontos sejam zero volts.

Esta montagem tem a vantagem de pode realizar medidas em sensores resistivos de forma imune às variações da fonte de alimentação. Ou seja, se a tensão subir ou descer o efeito é observado no dois ramos do circuito, importando a diferença de tensão imposta pelo divisor de tensão da resistência de calibração e a tensão imposta pela resistência do sensor.

Outra vantagem deste circuito é compensar a atenuação produzida pelos condutores que ligam ao sensor (resistência em série), pois a resistência de calibração irá compensar essa resistência adicional.

Tendo em conta que as medidas de tensão por um canal analógico de um sistema embebido são feitas entre o sinal e GND, neste caso a tensão diferencial (V_x) deve ser passada a um circuito que realize a função subtração, de modo a obter-se uma tensão em relação a GND (unipolar). Uma alternativa é usar duas entradas analógica e medir as duas tensões, e de preferência no mesmo instante. Se as duas tensões forem observadas em instantes diferentes, podem ocorrer variações numa das tensões, criando erros de medida.

A Ponte de Wheatstone, ilustrada na figura 1.17, pode ser também usada para descobrir qual a resistência de um sensor, por comparação com valores de resistência conhecidos.

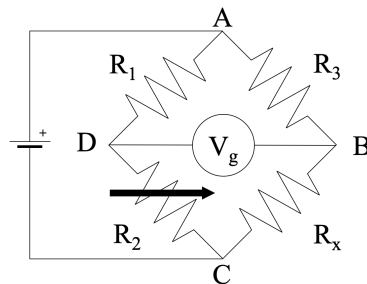


Figura 1.12: Ponte de Wheatstone.

Na figura 1.13 à esquerda, circuito onde a compensação da resistência dos fios de ligação do sensor é feita pela resistência de calibração. À direita, circuito com 3 ligações ao sensor para não ser necessário compensar a resistência dos fios. Na segunda montagem, a resistência dos fios com o mesmo comprimento à adicionada em ambas as resistências do divisor de tensão. O terceiro fio é usado para obter o ponto médio do divisor de tensão. De notar que ao medir a tensão, não existe corrente a passar, e como tal, não queda de tensão devido à resistência do terceiro fio.

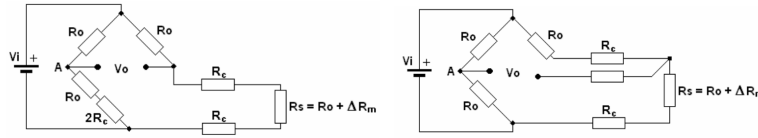


Figura 1.13: Ponte de Wheatstone para compensação das perdas nos fios.

1.2.5 Comparador Analógico

Para comparar o nível de um sinal analógico com um patamar correspondente a um valor limite para indicação de excesso de temperatura, luminosidade, ou uma ocorrência, pode ser necessário desocupar o processador do sistema embebido da tarefa de detetar esse limiar.

O circuito integrado 555 possui dois AMPOPs que funcionam como detetores de níveis de tensão, ligados a um divisor de tensão com três resistências de $5\text{ k}\Omega$, que geram as tensões $1/3\text{ VCC}$ e $2/3\text{ VCC}$ para servirem de referência de comparação. A entrada THRES encontra-se ligada ao positivo de um AMPOP e a entrada TRIG ao negativa do outro. Isto faz com que o AMPOP de cima tenha na sua saída VCC quando a tensão THRES seja superior a $2/3\text{ VCC}$ e GND caso contrário, e o AMPOP de baixo tenha a saída com VCC quando TRIG é inferior a $1/3\text{ VCC}$. Os sinais de saída dos AMPOPs estão ligados a um registo (*flip-flop*) do tipo Set-Reset, que permanece com a saída a VCC sempre que a entrada S seja momentaneamente VCC, ou com a saída a GND se a entrada R for VCC. Este circuito é bastante útil para sistemas com sinais que tendem a oscilar em torno de um valor, por exemplo, depósito de combustível, temperatura do ar, em que não se quer que o circuito controlado ligue e desligue frequentemente. Para este tipo de aplicação as entradas THRES e TRIG são ligadas à saída do sensor analógico. A listagem 1.2 mostra um programa em equivalente à funcionalidade do 555 num Arduino. De salientar que enquanto a tensão variar entre $2/3\text{ VCC}$ e $1/3\text{ VCC}$ não existe alteração do estado de saída (output).

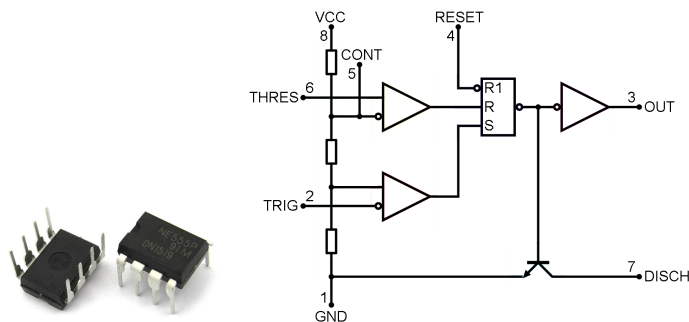


Figura 1.14: Aspeto do circuito integrado em encapsulamento DIP8 (esquerda) e diagrama de blocos (direita).

Listagem 1.2: Sketch para simular o funcionamento do circuito 555

```
int adcVal;
const int minVal = 1024/3, maxVal = 2*1024/3;
bool output = 0;

void loop() {
    adcVal = analogRead(pin);
    if(adcVal > maxVal) output = HIGH;
    if(adcVal < minVal) output = LOW;
}
```

1.2.6 Circuito Limitador

O circuito *clipping* é usado para limitar a tensão de saída a valores entre $-V$ e $+V$, para tensões de entrada fora desse intervalo. Este circuito é feito à custa de dois diodos do tipo zener ligados em série, e colocados em paralelo com a tensão a limitar. O diodo zener tem dois modos de funcionamento:

1. directo - funciona como um diodo normal. Apresenta uma queda de tensão V_d aos seus terminais (entre 0,3V e 0,7V);
2. inverso - começa a conduzir após ultrapassar a tensão de zener (V_z), ou de disrupção.

Existem vários diodos zener, cada um com a sua tensão V_z . Sabendo isto, e pensando num circuito na figura 1.15 à esquerda, apenas com um diodo zener, havendo uma tensão sinusoidal aos seus terminais, o diodo conduz diretamente quando a tensão é negativa, resultando numa tensão de 0,7V. Quando a tensão é positiva, mas inversa ao diodo, ele começa a conduzir após atingir V_z volts. No circuito da figura 1.15 à direita, com dois diodos zener em serie, o que acontece é que quer a tensão seja positiva ou negativa, um deles estará sempre a funcionar em sentido directo e o outro em sentido inverso. A tensão na saída deste circuito é $\pm(V_z + V_d)$.



Figura 1.15: Circuito clipping para limitação da amplitude/tensão máxima do sinal à entrada de microfone do telemóvel.

Em alternativa aos diodos zener, um circuito clipping pode ser construído com 2 series de N diodos convencionais ($V_{clip} = N \times V_d$), em paralelo com o sinal a limitar e com sentidos opostos.

1.2.7 Dreno/Colector Aberto

Os transístores são usados para controlar motores DC, relés, e sinais de controlo de outros sub-sistemas. A saída dos circuitos digitais habitualmente fornece um valor na sua saída, 0 ou 1. No entanto, existem certos circuitos que não fornecem a tensão correspondente ao valor 1. Isto significa que o sinal é 0 ou encontra-se flutuante, e necessita de ser polarizado para ter o valor 1. Por outras palavras, a saída é 1 se não existir um 0 imposto. Outra analogia é usar a saída desse sinal como um interruptor que liga a GND, em que o outro terminal do componente é ligado a VCC.

Um circuito *Open-Drain* é constituído por um transístor (bipolar ou MOSFET) do tipo N numa montagem Emissor/Source-Comum, cuja Base/-Gate se encontra ligada ao terminal de saída do MCU, o Emissor (E) /Source ligado a GND, e o Colector (C) / Dreno (D) ligado ao terminal negativo da carga a controlar. Ao existir uma tensão nula nos terminais do transístor, significa que a junção BE não se encontra polarizada, e por consequência o transístor encontra-se a funcionar na zona de corte. Quando na zona de corte, o transístor, não permite a passagem de corrente entre o colector e o emissor, mantendo a carga em circuito-aberto. Ao existir uma tensão nos terminais BE superior à tensão limite de condução (V_{th}), o transístor activa a sua junção BE e permite a passagem de corrente do colector para o emissor, fechando o circuito da carga a GND, e desta forma activando-a.

Este circuito é habitualmente usado para controlo de cargas onde se observam as seguintes características:

1. com tensão de funcionamento diferente do circuito;
2. com consumo de corrente superior ao máximo fornecido pela saída do circuito de controlo.
3. apenas é necessário forçar um potencial.

Na realização deste circuito importa escolher um transistor/mosfet com características adequadas, nomeadamente:

- Corrente máxima;
- Tensão de funcionamento (V_{th}):

A principal diferença em termos de tecnologia bipolar e MOSFET é o consumo de corrente de Base enquanto o MOSFET não requer corrente na Gate. Como num transistor bipolar a corrente de base é proporcional à corrente de Colector ($I_c = \beta I_b$, onde valores típicos são $\beta = 100$, o valor de corrente máximo que se consegue na carga, usando um transistor, é dada pela corrente máxima que a porta do MCU consegue fornecer, vezes o valor de β).

CAPÍTULO 1. INTERFACES DE ENTRADA/SAÍDA

A escolha do valor de V_{th} é importante pois vai determinar qual a tensão mínima para o transistor entrar em funcionamento. Se este valor for muito baixo ($< 0,3V$) existe o risco do transistor começar a funcionar mesmo com a saída digital com o valor lógico 0. Por outro lado, um V_{th} muito alto, por exemplo, 3V para um sistema que funcione a 3,3 V pode fazer com que o transistor nunca entre em funcionamento devido a quedas de tensão que possam existir na alimentação e no circuito.

Na tabela 1.1 encontram-se exemplos de referências de transistores bipolares e MOSFET do tipo N adequados a realização de um porta open-collector/open-drain.

Referência	Tipo	Ic max	Vce max	f max	Ganho
BC548	BJT NPN	100mA	15 V	150 MHz	110-800
BC558	BJT PNP	100mA	30V	150 MHz	110-800
2N2222	BJT NPN	800mA	40V	300 MHz	100
2N2907	BJT PNP	600mA	60V	200 MHz	100
2N3055	BJT NPN	15A	100V	3MHz	20-70
2N2955	BJT PNP	15A	100V	3MHz	20-70
Referência	Tipo	Ic max	Vce max	f max	VGS on
2N7000	MOSFET N	200mA	60V	150 MHz	0,8-3V
2N7002	MOSFET P	100mA	60V	150 MHz	0,8-3V
IRF540	Power MOSFET N	30A	100V		
IRF610	Power MOSFET N	3.3A	200V		
IRF5305	Power MOSFET P	31A	55 V		
IRF9520	Power MOSFET P	6.8A	100V		
2N5460	JFET P	5mA			
2N5459	JFET N	0.1A			

Tabela 1.1: Algumas referências de transistores e as principais caraterísticas.

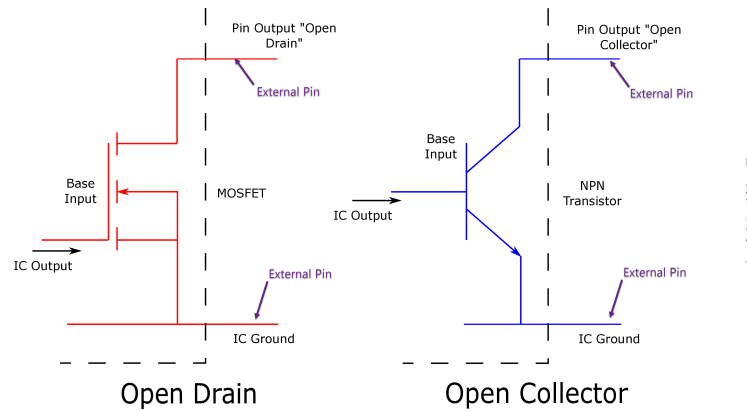


Figura 1.16: Coletor e Dreno abertos.

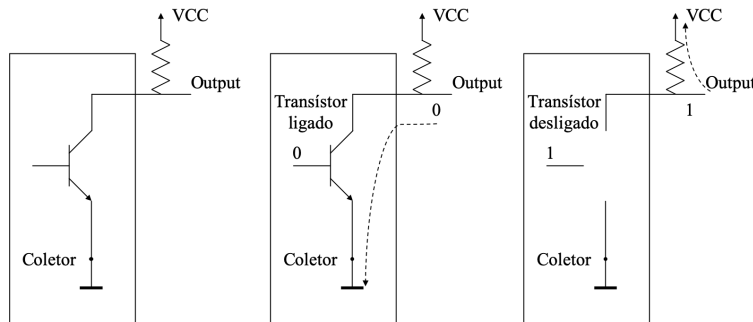


Figura 1.17: Controlo de carga com uma porta open-colector.

1.2.8 Push-Pull

O circuito *Push-Pull* é uma versão mais completa do circuito *open-drain*, ao ativar o circuito para tensões de entrada positivas e negativas. Este tipo de circuitos é usado em circuitos que controlam cargas cuja corrente varia nos dois sentidos, tais como amplificadores áudio. O circuito *Push-Pull*, apresentado na figura 1.18, é construído de forma a que os dois andares nunca estejam ativos em simultâneo, caso contrário ocorreria um curto-circuito de VCC para GND. Em função do sinal de entrada, este circuito comporta-se da seguinte forma:

- quanto a entrada das bases dos transistores é próxima de VCC, o transistor de baixo entra à condução, e o transistor de cima encontra-se ao corte, fazendo a saída ser 0
- quanto a entrada das bases dos transistores é próxima de GND, o transistor de cima encontra-se a conduzir, e o transistor de baixo encontra-se ao corte, fazendo a saída ser 1

Uma característica deste circuito é “puxar” a tensão para cima ou “empurrar” para baixo, em relação à origem (0V). Por este motivo é habitualmente usado em circuitos com tensão negativa (alimentação complementar). No entanto é possível realizar em circuitos com apenas uma tensão de alimentação, desde que seja derivada uma tensão igual $VCC/2$. Neste circuito, para além do transistor do tipo N, é necessário outro do tipo P. O controlo do circuito, do lado do sistema embebido, é feito com uma saída digital.

1.2.9 Ponte-H

O circuito Ponte-H corresponde a dois circuitos *Push-Pull*, com uma saída para cada terminal da carga, onde a carga é ligada ao ponto intermédio de cada *Push-Pull*, ilustrado na figura 1.21. Para além de controlar o nível de cada terminal da carga, este circuito permite controlar o sentido da corrente

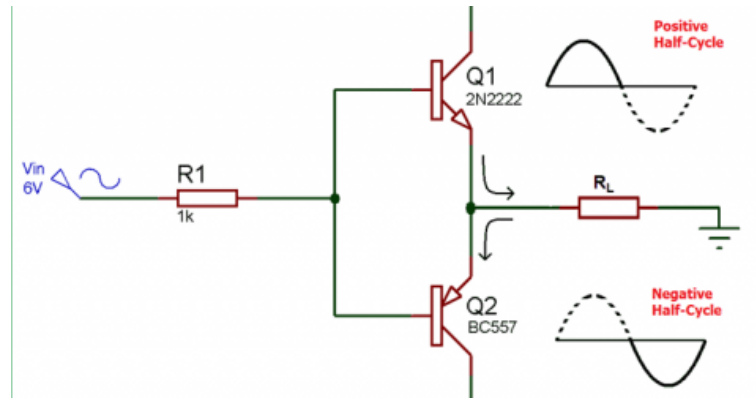


Figura 1.18: Push-Pull.

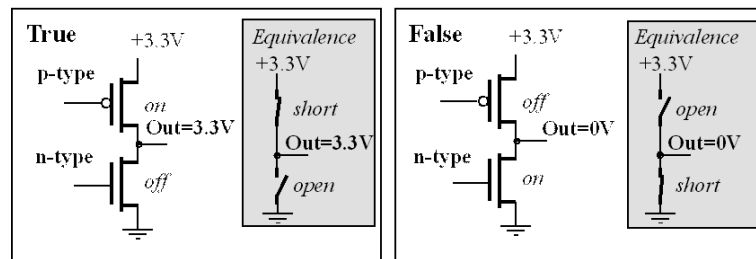


Figura 1.19: Push-Pull 2.

através das várias combinações de saída: H-H, H-L, L-H, e L-L, tal como ilustrado na figura 1.21. Este circuito é bastante usado para controlo de cargas que necessitem de inverter o sentido da corrente que os percorre, nomeadamente:

- motores DC - inverter o sentido de rotação (H-L/L-H), e travar o motor (L-L);
- elementos de Peltier - inverter a temperatura da superfície.

Para controlar este circuito do lado do sistema embebido são necessários duas saídas digitais (dois bits), ou duas tensões analógicas.

1.3 Interfaces Digitais

Nesta secção são mostradas as interfaces digitais mais frequentes em sistemas embebidos para comunicar com outros sistemas ou componentes. As interfaces digitais são compostas por sinais de um ou mais pinos de E/S, e são usadas para transferir dados via conjuntos de bits, apenas num ou nos dois sentidos, de forma organizada (protocolo) e eficiente. Os sinais dos dados não adequados para alimentar componentes ou periféricos, devendo cada componente ter o seu próprio circuito de alimentação.

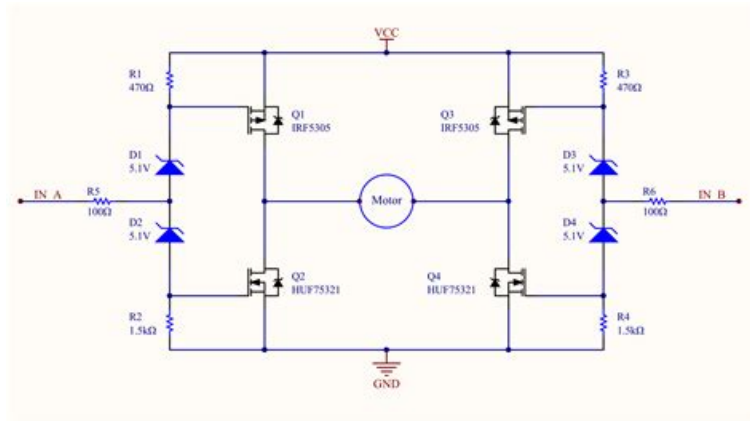


Figura 1.20: Circuito H-bridge.

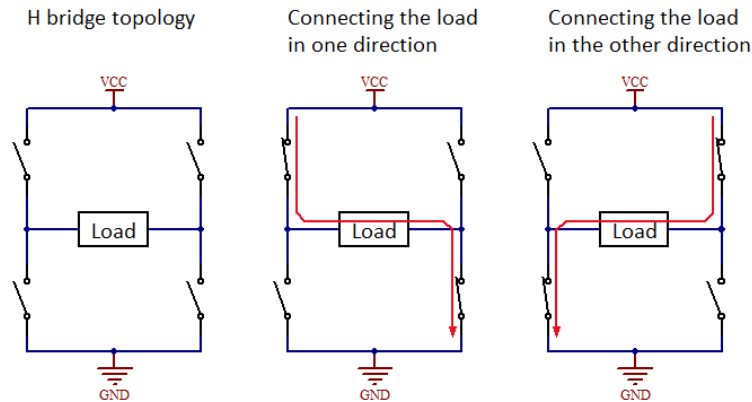


Figura 1.21: H-bridge.

As interfaces podem ser suportadas por uma unidade, dentro do MCU, dedicada a essa funcionalidade ou emulada pelo programa usando pinos de E/S genéricos, sendo o desempenho do sistema penalizado em:

- Ocupação de tempo do CPU;
- Tempo de resposta e velocidade de funcionamento da interface;
- Tempo de processamento consumido com a operação.

Para os protocolos apresentados neste capítulo, a comunicação é feita usando controladores dedicados, habitualmente são circuitos existentes dentro dos MCUs. Esta funcionalidade pode ser emulada em software quando não existe um controlador dedicado disponível. Nas interfaces digitais é importante observar qual a velocidade/frequência máxima suportada.

A depuração e depiste de erros nos canais de comunicação digital é feita com recurso a um analisador de protocolos dedicado ou com um osciloscópio digital que tenha a capacidade de descodificação e análise do protocolo.

1.3.1 Entradas, Saídas e Interrupções

Os pinos de E/S possuem capacidades para várias funcionalidades: receber e enviar informação e fazer o MCU executar uma rotina específica fora da execução normal do programa. A figura 1.22 mostra a arquitetura interna de um porto digital de entrada e saída do MCU, onde o circuito de entrada, saída e pull-up partilham a ligação ao pino no encapsulamento do MCU. Nesta figura são visíveis os circuitos de entrada, saída e de pull-up. Este bloco é replicado em todos os pinos digitais do MCU.

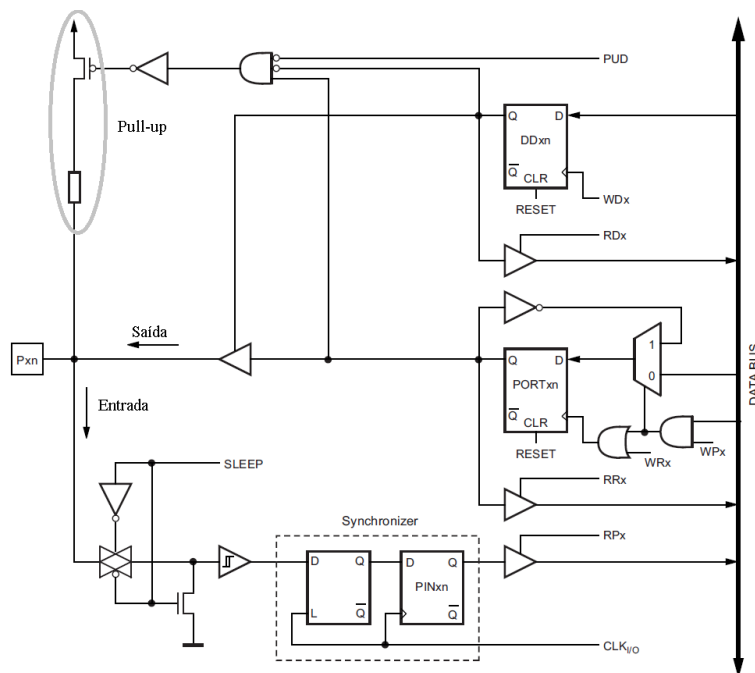


Figura 1.22: Arquitectura de um pino de entrada e saída do MCU num Arduino. Esquema adaptado do manual do componente.

Diferentes tecnologias de fabrico de circuitos digitais fazem com que os dispositivos funcionem de forma diferente. Uma das diferenças é a sua tensão de funcionamento. Os circuitos digitais, tipicamente funcionam com alimentações de 1,8 V, 2,5 V, 3,3 V ou 5 V, e todos os componentes do circuito funcionam tendencialmente com a mesma tensão. Quando é necessário integrar tecnologias que funcionam com tensões diferentes é necessário adicionar circuitos de adaptação sob pena de destruição ou mau funcionamento dos circuitos. A figura 1.23 ilustra os limiares de decisão para 0 e 1, tanto à entrada (ViL e ViH) como à saída (VoL e VoH), para diferentes tecnologias de fabrico de circuitos digitais.

Existem placas Arduino que funcionam com 3,3 V ou 5 V. O sinal IOREF, existente no conector de expansão, disponibiliza a tensão usada nos sinais de E/S para que os Shields possam funcionar com a tensão correta. Se o

circuito a ligar ao Arduino for alimentado com essa tensão é garantida a compatibilidade no seu funcionamento. Em todo o caso, é necessário ter atenção às tensões de funcionamento na escolha de Shields Arduino para que eles não danifiquem, nem sejam danificados pela placa Arduino.

Logic Levels and Noise Margin for CMOS devices

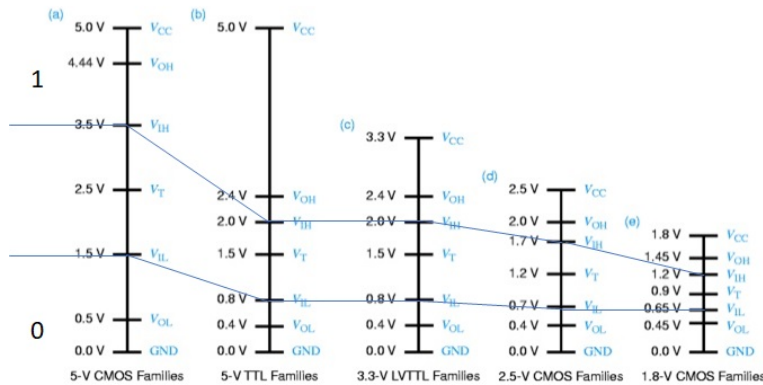


Figura 1.23: Níveis de tensão para detecção dos valores 0 e 1 na entrada de uma porta do tipo CMOS.

Entrada e Saída

Um sinal digital num sistema embebido deve apresentar um dos dois valores “0” ou “1”. Para se pode usar um valor de um pino de entrada digital, é necessário que exista sempre presente um valor conhecido. Quando nenhum sinal se encontra ligado a uma entrada digital, diz-se que o pino de entrada “está no ar”, e é sensível ao ruído electromagnético ambiente, ou seja, o seu valor vai ser 0 ou 1 de forma aleatória.

Interruptores com dois terminais apenas conseguem estabelecer uma ligação a V_{CC} ou a GND . Quando um interruptor está ligado, existe GND , mas quando está desligado está no ar e tanto pode ser lido 0 ou 1. Por causa deste problema é necessário fornecer um valor conhecido ao circuito que possa ser facilmente contrariado pela atuação de outro componente.

Um *pull-up* serve para polarizar um sinal com um determinado nível lógico. Ele coloca no pino um sinal correspondente ao valor por omissão quando não existe outro sinal nele aplicado. A tensão do *pull-up* deve ser *facilmente vencida* por outro sinal colocado no pino de entrada. Na prática o *pull-up* é conseguido com uma resistência de dezenas ou centenas de milhares de Ohm (10-100 k Ω).

Importa lembrar que um pino de E/S do Arduino quando configurado como entrada não tem corrente a percorrê-lo, e que uma resistência quando não é percorrida por uma corrente não provoca queda de tensão nos seus terminais. Deste modo, quando um botão de pressão não se encontra premido, a tensão em ambos os terminais da resistência é VCC, tal como a tensão no pino de entrada do Arduino. Quando o botão de pressão está premido, ele estabelece uma ligação direta a GND, logo *força* a que a tensão no pino de entrada do Arduino seja GND. Neste caso a resistência de *pull-up* tem um terminal ligado a VCC e outro a GND, significando que existe uma corrente a percorrer a resistência e o botão de pressão. Esta corrente não tem qualquer aplicação e é apenas uma consequência do circuito, pelo que deve ter o menor valor possível, especialmente em circuitos ligados a baterias. Por este motivo a resistência de *pull-up* deve ter um valor elevado. No entanto devem-se evitar valores demasiado altos, pois terão um efeito semelhante a não ter tensão aplicada ao pino de entrada. Este efeito acontece porque uma resistência de milhões de Ohm provoca uma queda de tensão elevada que perturba o funcionamento do *pull-up*, e a corrente torna-se residual (pA) tornando a entrada sensível a ruído ambiente.².

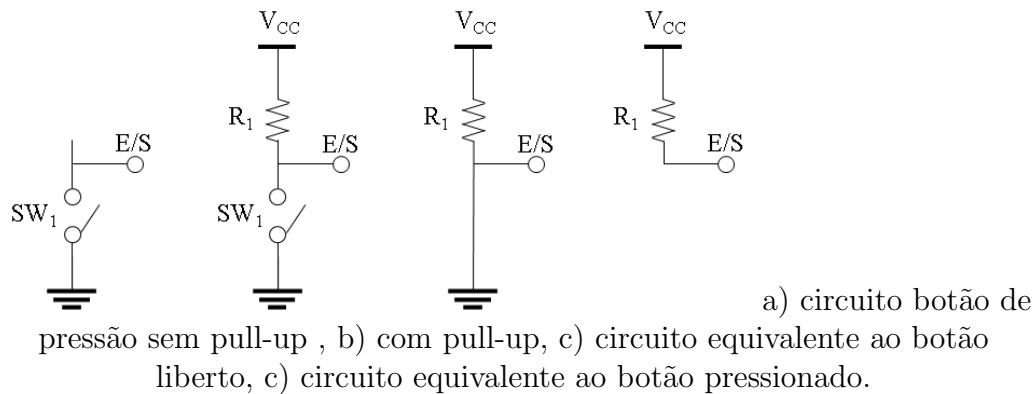


Figura 1.24: Circuito para ligação de um botão de pressão a um pino de E/S do Arduino, sem pull-up, com pull-up e circuitos equivalentes para o botão pressionado e libertado.

Para simplificar o circuito e reduzir o número de componentes necessários, os pinos de entrada do Arduino possuem *pull-ups* internos programáveis. Os pull-ups são programados no *Sketch* Arduino com a função `pinMode()`, ilustrado na listagem 1.3, aquando da definição do pino de entrada:

- INPUT - entrada sem *pull-up*;
- INPUT_PULLUP - entrada com *pull-up*.

²Embora pareça contraditório, é importante ter uma visão prática e compreender os fenómenos envolvidos

Listagem 1.3: Configuração dos pinos de entrada

```
void setup() {  
  pinMode(3, INPUT);  
  pinMode(4, INPUT_PULLUP);  
}
```

Há muito tempo foi convencionado que as ativações correspondem a ligar sinais de entrada a GND, especialmente na indústria automóvel. Por este motivo os MCUs não possuem habitualmente *pull-downs* internos.

A figura 1.25 mostra dois diagramas de forma de onda. No topo, os sinais foram obtidos com a montagem do botão de pressão sem o *pull-up*. Neste caso, o sinal analógico observado é apenas ruído proveniente da interferência existente no espaço envolvente, e o funcionamento do sistema acompanha esse ruído. Nota-se que o sinal de entrada tem um patamar estável em 0V, que corresponde ao momento em que o botão de pressão foi premido. Em baixo, o pino de entrada foi configurado para usar o *pull-up* interno. Neste caso observa-se que, quando o botão não se encontra premido (sinal de entrada está a GND), que ele se encontra a VCC. Neste diagrama observa-se também a existência de transições espúrias, derivadas do movimento das partes mecânicas do botão que muitas vezes não estabelecem o circuito de forma imediata.

A interação mais básica que um sistema embebido pode oferecer é a leitura de um bit de uma entrada digital ou a atuação de um bit numa saída digital. Neste caso o programador é responsável pela implementação da concordância, ou protocolo, de funcionamento entre o MCU e o periférico.

No capítulo anterior foi mostrado como realizar as leituras e escritas de um bit num programa Arduíno, faltando tecer algumas considerações acerca das características elétricas para uma interface deste tipo.

As características de um porto digital que devem ser observadas são:

- Tensão máxima admitida para não destruir o MCU ou o periférico. Por exemplo, é necessário ter cuidado em adaptar os níveis de tensão em componentes que funcionem a 3,3 V e a 5 V;
- Nível de decisão de valor 0 e 1 na leitura do valor de tensão. Existem diferentes tipos de portas que decidem em diferentes valores limite. Por exemplo, as portas do tipo Schmitt-Trigger apenas assumem o valor 0 quando a tensão se encontra abaixo de V_{iL} e o valor 1 acima de V_{iH} , não mudando de valor dentro do intervalo de tensões V_{iL} - V_{iH} . Isto é importante para evitar transições enquanto a tensão do pino se encontra a evoluir de um valor para outro e pode ser sensível a ruído. Esta solução evita dar indicações de atividade espúrias.
- Frequência máxima de funcionamento da porta.

CAPÍTULO 1. INTERFACES DE ENTRADA/SAÍDA

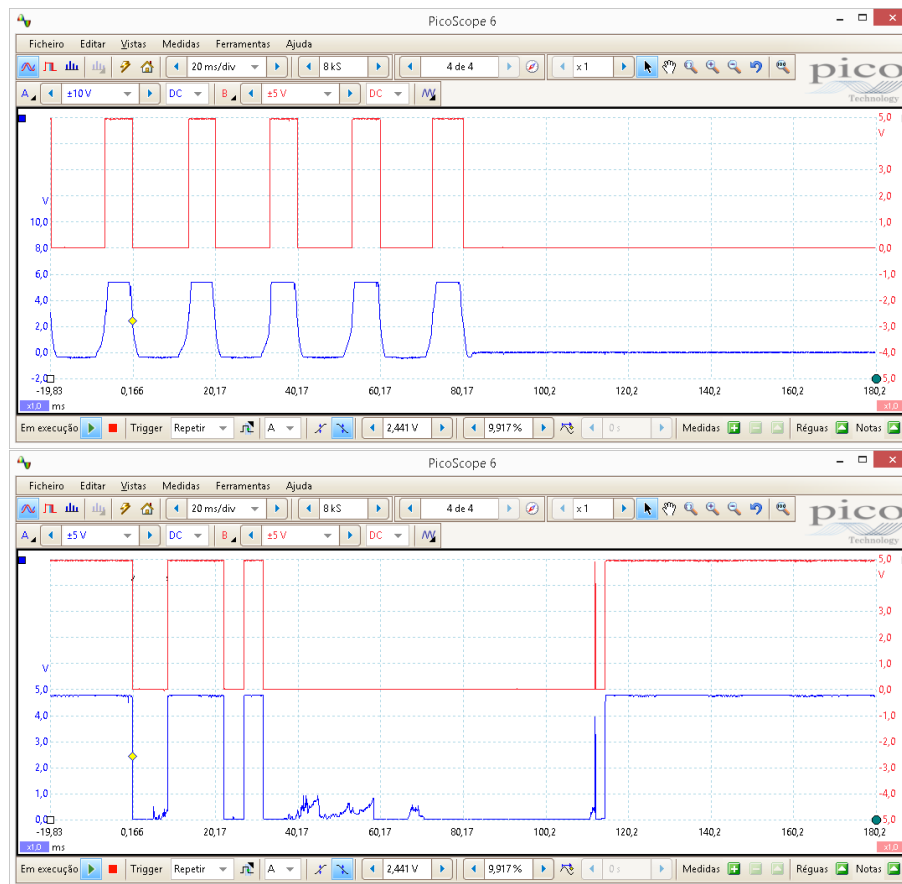


Figura 1.25: Sinais de entrada e saída da ligação do botão de pressão sem (topo) e com (baixo) pull-up no pino de entrada.

Para fazer a adaptação entre circuitos 3,3 V e 5 V que envolvem comunicação bidirecional, é uma boa opção usar buffers bidirecionais, tais como o LTC4315, ou o MOSFET BSS88, descrito no documento AN97055 da Philips. Existem também buffers unidirecionais para elevar de 3,3 V para 5 V, como o 74AHCT244 que é capaz de detectar o nível 1 com 2V nos pinos de entrada.

O circuito divisor de tensão também é usado para reduzir um nível de tensão elevado noutro mais baixo. Por exemplo, um circuito que na sua saída gere um sinal entre 0V e 5 V não pode ser ligado diretamente às entradas de um MCU que funcione a 3,3 V, sob risco de o danificar ou destruir. O divisor de tensão vai permitir reter os 1,7 V em excesso.

Por exemplo, se R1 e R2 forem ambas de 10k Ω , $V_o = V_i / 2$. Se R1 for de 20k e R2 for de 10k Ω , $V_o = V_i * 2 / 3$. No caso prático de $V_i = 5$ V, $V_o = 3,33$ V, o que já é aceitável para servir de interface entre um sistema que funcione a 5 V e outro a 3,3 V. De notar que a direcção é sempre de 5 V para 3,3 V, se for ao contrário, de 3,3 V para 5 V, é necessário usar um

buffer para amplificar o sinal.

A par com este problema alguns fabricantes fornecem pinos de E/S que funcionam a 3,3 V mas que são tolerantes a 5 V, para evitar ter de usar circuitos adicionais para adaptação dos níveis de tensão.

A listagem 1.4 exemplifica a configuração do pino2 como entrada e do pino 13 como saída, correspondente ao LED na placa Arduino Uno. Este sketch realiza a leitura do valor do pino de entrada e copia-o para o pino de saída. O pino de entrada pode ser configurado para ter pull-up através da macro INPUT_PULLUP.

Listagem 1.4: Configuração de um pino de entrada digital

```
void setup() {
  pinMode(2, INPUT); // sem pull-up
  // pinMode(2, INPUT_PULLUP); // com pull-up
  pinMode(13, OUTPUT);
}

void loop() {
  int inputVal = digitalRead(2);
  digitalWrite(13, inputVal);
}
```

Uma alternativa de manipulação individual de pinos/bits, é ler e escrever um conjunto de pinos/bits em simultâneo através de um porto. A vantagem é não ter de efetuar a escrita em vários bits, no entanto também traz desvantagens como gerir leituras e escritas quando vários componentes estão ligados ao mesmo porto³. Um MCU pode ter vários portos de entrada e saída. Num Arduino Uno existem os seguintes portos com a correspondência aos pinos:

- PORTD - D0 a D7;
- PORTB - D8 a D13;
- PORTC - A0 a A5.

Para poder usar os portos é necessário indicar quais os bits que são de entrada e os de saída através da instrução $DDRx = K$, onde x é substituído pela letra do porto (B,C,D), e K o valor inteiro correspondente à configuração de todos os bits do porto. Um bit que seja de entrada tem o valor 0, e se for de saída tem o valor 1. O valor de K é dado pelo peso do bit: $K = \sum_{i=0}^7 b_i \cdot 2^i$. Por exemplo, se D0 a D3 forem saídas e D4 a D7 entradas, $K = 1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3 + 0 \cdot 2^4 + 0 \cdot 2^5 + 0 \cdot 2^6 + 0 \cdot 2^7 = 1 + 2 + 4 + 8 + 0 + 0 + 0 + 0 = 15 = 0x0F$. A alternativa é usar representação binária: $DDRD = B00001111$.

Por exemplo, no sketch 1.5 os pinos D0 a D7 são saídas e alternam o padrão de saídas ativas.

³<https://www.arduino.cc/en/Reference/PortManipulation>

Listagem 1.5: Exemplo de escrita de todos os bits do porto D

```
void setup()
{
  DDRB = 0xFF;
  DDRC = 0x00;
  DDRD = 0xFF;
}

void loop()
{
  PORTB = PORTB + 1;
  PORTD = 0x55;
  delay(500);
  PORTD = 0xAA;
  delay(500);
}
```

Interrupções

Tal como apresentado no capítulo anterior, a interrupção é um mecanismo que permite suspender a execução normal do programa para forçar a execução de uma rotina/função específica para atendimento à interrupção, aquando da alteração do estado de um pino de entrada digital (noutros sistemas embebidos pode ser um sinal de um periférico interno). Depois da rotina terminada, a execução do programa é retomada a partir do ponto onde ficou suspensa.

Nos sistemas embebidos as entradas dedicadas a sinais de interrupção são por vezes ligadas a saídas de periféricos para indicarem que existem dados disponíveis para serem lidos. A rotina de interrupção, do inglês *Interrupt Service Routine* (ISR), não recebe parâmetros nem devolve valores. Esta rotina serve para auxiliar o periférico, e deve ser o mais breve possível, para o programa poder (rapidamente) continuar com a sua execução. Dentro desta rotina não se devem realizar esperas activas nem comunicar com outros periféricos porque vai consumir demasiado tempo. A função `delay()` não funciona dentro das rotinas de interrupção, pois ela funciona com base na interrupção do temporizador interno. É como o CPU ficasse à espera de uma interrupção quando já havia sido interrompido por outra!

No Arduíno as interrupções são programadas usando a função `attachInterrupt(numIntr, ISR, ativação)`, onde `numIntr` corresponde ao número da interrupção a usar para o pino de entrada desejado (atenção que não é o número do pino), `ISR` corresponde ao nome da função a executar. Embora seja possível indicar manualmente a interrupção, é preferível traduzir o número de pino para o número da interrupção com a função `digitalPinToInterrupt(pinoIntr)`. Para usar o pino de entrada diretamente recomenda-se usar: `attachInterrupt(digitalPinToInterrupt(pinoIntr), ISR, ativação)`.

As placas Arduíno variam nas entradas de interrupção:

Placa Arduino	Pinos Intr
Uno, Nano	2, 3
Mega	2, 3, 18, 19, 20, 21
Micro, Leonardo	0, 1, 2, 3, 7
Due	todos os pinos digitais

As ativações de uma interrupção podem ser:

- LOW - quando o sinal passa para o nível 0;
- HIGH - quando o sinal passa para o nível 1 (apenas nos Due e Zero);
- CHANGE - quando o sinal muda de estado de nível 0 para 1 ou 1 para 0;
- RISING - na subida/transição de nível 0 para 1;
- FALLING - na descida/transição de nível 1 para 0;

`detachInterrupt(digitalPinToInterrupt(pin))` desativa a interrupção previamente configurada.

Em aplicações onde um objeto é responsável por interagir com um componente que gera interrupções é necessário ter em atenção que a rotina de atendimento a uma interrupção não pode ser um método de um objeto instanciado. No entanto, pode ser uma rotina privada da classe (tipo `static`). Naturalmente surge a questão: Como pode um objeto manipular os seus atributos se a rotina de interrupção é `static` e não tem acesso a eles? A solução passa por adicionar um ponteiro (`IntrDemo * intrInst_`), que é iniciado no construtor com o ponteiro do objeto instanciado, e invocar outra rotina existente na instância do objeto.

A listagem 1.6 ilustra a utilização de um objeto no atendimento a uma interrupção. Neste exemplo, é mostrado no terminal de texto o valor de um contador a cada segundo. Cada vez que existir uma transição do sinal no pino 3 de 1 (HIGH) para 0 (LOW), o contador é incrementado. Se existirem várias transições durante um segundo, o contador será incrementado esse número de vezes. A nível de circuito é necessário um botão de pressão ligado entre o pino 3 e GND.

A rotina de interrupção `static void IntrDemo::Isr(void)` é associada à interrupção no construtor da classe, aquando da criação de um objeto `IntrDemo`, e o ponteiro `intrInst_` fica com a referência do objeto (`this`). Sempre que houver uma interrupção provocada por um sinal externo, a rotina `Isr()` é invocada, e nela é executado o método privado `doIsrStuff()` através da referência do objeto `intrInst_`.

Listagem 1.6: Exemplo de atendimento a interrupções com objetos.

```
// start of header file intrdemo.h
class IntrDemo {
private:
```

```
    byte intrPin;
    byte counter;
    static void Isr(void);
    void doIsrStuff(void) {
        counter++;
    }
public:
    IntrDemo(byte intrPin, byte type);
    int getCount() { return counter; }
};

// start of source file intrdemo.cpp
static IntrDemo * intrInst_;

static void IntrDemo::Isr(void)
{
    // if(intrInst_) // comment for speedup
    intrInst_>doIsrStuff();
}

IntrDemo::IntrDemo(byte intrPin, byte type) {
    pinMode(intrPin, INPUT_PULLUP);
    intrInst_ = this;
    counter = 0;
    attachInterrupt(digitalPinToInterrupt(intrPin), Isr, type);
}

// start of sketch (use #include "intrdemo.h")
IntrDemo intrObj(3, FALLING);

void setup()
{
    Serial.begin(115200); while(!Serial);
}

void loop()
{
    Serial.println(intrObj.getCount());
    delay(1000);
}
```

1.3.2 Multiplexagem e Charlieplexagem

A multiplexagem consiste na partilha de alguns sinais para controlar componentes diferentes. No exemplo da figura 1.26, uma matriz de 3x4 LEDs, fica com as suas ligações divididas em grupos de 3 linhas e 4 colunas, em vez de ter ligações aos 12 LEDs. Neste caso, os sinais positivos de cada linha são todos ligados em conjunto e apenas a um sinal, comandado por um interruptor S1 a S3 (ex. L5,L6,L7 e L8 em S2). Os negativos de cada LED de uma linha são ligados em colunas independentes, tal como ilustrado na figura (L3,L7,L11 em S6).

Para o circuito dado, S2 controla a linha do meio, e permite que os LEDs 5 a 8 possam ser seleccionados pelos interruptores S4 a S7. Sempre que se feche um circuito com qualquer LED, ele acenderá. É o mesmo que dizer que a ativação de qualquer LED depende da combinação da ativação das linhas com as colunas.

Alternando a selecção das linhas várias vezes por segundo, e ativando os LEDs a acender em cada coluna, é possível dar a ilusão de que vários LEDs acendem em simultâneo e de forma independente. Esta ilusão é possível

através da persistência retiniana, que filtra variações superiores a 30 Hz. O mesmo princípio de multiplexagem pode ser usado para matrizes de teclados, onde as colunas podem ser compartilhadas por todos os botões de pressão de cada coluna, e através da ativação de cada linha num determinado instante de tempo é possível fazer a leitura de cada linha. Ao varrer as várias colunas, é possível dar a ilusão de que as 16 teclas são independentes.

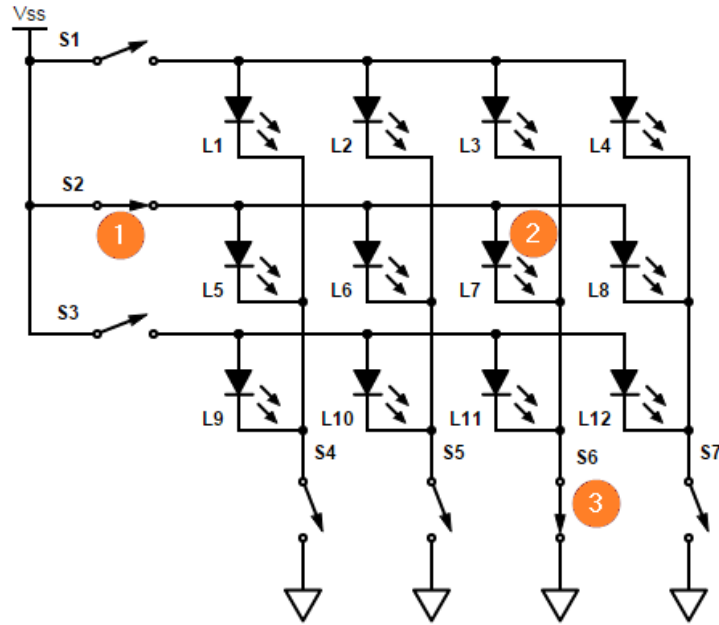


Figura 1.26: Multiplexagem de uma matriz de 3x4 LEDs.

A listagem 1.7 mostra um Sketch de demonstração da multiplexagem da matriz de 3x4 LEDs.

Listagem 1.7: multiplexingDemo.ino

```
#define NUMLINS    3
#define NUMCOLS    4
#define NUMLEDS    (NUMLINS * NUMCOLS)
#define LEDDELAY    333

// I/O pins for lines and columns
byte linPins[] = {2, 3, 4};
byte colPins[] = {5, 6, 7, 8};
// bit activations of lines and coluns for different numbers
byte linVals[] = {0,1,1,1,1,2,2,2,2,4,4,4,0};
// cluns are negated (active = GND)
byte colVals[] = {0xF,0xE,0xD,0xB,0x7,0xE,0xD,0xB,0x7,0xE,0xD,0xB,0x7};

byte i;

void setup(){
  for(i = 0; i < NUMCOLS; i++) {
    pinMode(colPins[i], OUTPUT);
    digitalWrite(colPins[i], LOW);
  }
}
```

```
void setLED(byte ledNum)
{
  byte linVal, colVal;
  // disable cols = clear
  for(i = 0; i < NUMCOLS; i++) {
    digitalWrite(colPins, HIGH);
  }
  if(ledNum > NUMLEDS) return; // if it exceeds, then exit
  linVal = linVals[ledNum];    // get activations for the ledNum
  colVal = colVals[ledNum];
  for(i = 0; i < NUMLINS; i++) { // ativate lines
    digitalWrite(linPins[i], linVal & (1 << i));
  }
  for(i = 0; i < NUMCOLS; i++) { // ativate columns
    digitalWrite(colPins[i], ~(colVal & (1 << i)));
  }
}

void loop(){
  for (i = 0; i <= 12; i++) {
    setLED(i);
    delay(LEDDELAY);
  }
}
```

Neste Sketch cada um dos doze LEDs acende sequencialmente durante um breve instante de tempo. A função `setLED(byte ledNum)` é responsável pela multiplexagem dos LEDs. Esta função começa por desativar todas as colunas, colocando-as a nível H, uma vez que se encontram ligadas ao cátodo dos LEDs. O valor do bit de linha e coluna que vai ser necessário acender para ativar um determinado LED, é realizado com o `switch`. O valor dos bits é codificado em base 2 para ser fácil obter o nível lógico de cada LED à custa de uma máscara com o peso do valor da linha e da coluna. Como as colunas foram desativadas, os valores dos bits das linhas podem ser afetados sem haver alteração momentânea nos LEDs. A ativação das colunas é última parte da função.

A Charlieplexagem foi proposta em 1995 por Charlie Allen da Maxim Integrated, e em vez de considerar apenas 2 estados para os sinais de E/S (1 e 0), usa 3 estados para controlar os pinos de E/S (0, 1 e saída desligada/entrada). A maioria dos pinos de E/S digitais dos MCUs pode ser configurado para leituras ou escritas. No caso de estar configurado como leitura, esse pino deixa de apresentar 1 (H) ou 0 (L), e passa a ficar em alta-impedância (Z), ou seja, não há corrente a circular por esse pino. A Charlieplexagem tira partido desta característica para permitir uma combinação adicional à multiplexagem, definindo as seguintes combinações para dois terminais:

- H-L: LED direto acende, LED inverso apagado;
- H-H: LEDs apagados;
- L-H: LED inverso acende, LED direto apagado;
- L-L: LEDs apagados;
- H-Z: LEDs apagados;

- Z-Z: LEDs apagados;
- Z-H: LEDs apagados;
- Z-L: LEDs apagados;
- L-Z: LEDs apagados;

Com base nestas combinações, é possível ligar várias combinações de LEDs entre o mesmo conjunto de sinais de E/S. A figura 1.27 ilustra um circuito para ligar 6 LEDs usando apenas 3 sinais de E/S. A tabela 1.2 mostra as várias combinações nos 3 pinos de E/S para ativar cada um dos 6 LEDs independentemente.

LED Aceso	X1	X2	X3
1	H	L	Z
2	L	H	Z
3	Z	H	L
4	Z	L	H
5	H	Z	L
6	L	Z	H

Tabela 1.2: Combinações dos sinais de E/S para ativar cada LED usando Charlieplexagem.

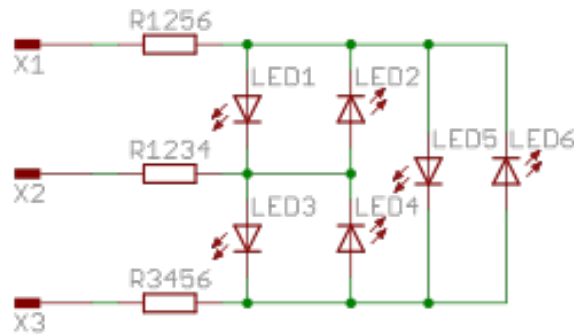


Figura 1.27: Charlieplexagem.

A figura mostra as ligações a realizar para ter 6 LEDs ligados a um Arduino. Um exemplo de programa para demonstração da Charlieplexagem com 6 LEDs, usando 3 sinais de E/S num Arduino encontra-se na listagem 1.8.

Listagem 1.8: Demonstração da Charlieplexing com LEDs

```
// charlieplexing demo with 6 LEDs
#define NUMLEDS 6
#define LEDDELAY 333

// H: set pin as output and HIGH
// L: set pin as output and LOW
// Z: set pin as input
#define setH(pin); { pinMode(pin, OUTPUT); digitalWrite(pin, HIGH); }
#define setL(pin); { pinMode(pin, OUTPUT); digitalWrite(pin, LOW); }
#define setZ(pin); { pinMode(pin, INPUT); }

byte pin[] = { 2, 3, 4};

void setup()
{
    // start all LEDs off
    setZ(pin[0]);
    setZ(pin[1]);
    setZ(pin[2]);
}

// light-up 1 of the LEDs
void setLED(byte ledNum)
{
    switch(ledNum) {
        case 0: setZ(pin[0]); setZ(pin[1]); setZ(pin[2]); break;
        case 1: setH(pin[0]); setL(pin[1]); setZ(pin[2]); break;
        case 2: setL(pin[0]); setH(pin[1]); setZ(pin[2]); break;
        case 3: setZ(pin[0]); setH(pin[1]); setL(pin[2]); break;
        case 4: setZ(pin[0]); setL(pin[1]); setH(pin[2]); break;
        case 5: setH(pin[0]); setZ(pin[1]); setL(pin[2]); break;
        case 6: setL(pin[0]); setZ(pin[1]); setH(pin[2]); break;
        default: break;
    }
}

// light sequence LED 1-6
void loop()
{
    byte i;
    for(i = 0; i <= NUMLEDS; i++) {
        setLED(i);
        delay(LEDDELAY);
    }
    for(i = NUMLEDS - 1; i > 0; i--) {
        setLED(i);
        delay(LEDDELAY);
    }
}
```

Este Sketch acende cada um dos seis LEDs alternadamente, através do método `setLED(byte ledNum)`. Este método é responsável por colocar os três sinais de E/S com o modo e valor para apenas que o LED correspondente esteja aceso. As *macros* `setH(pin)`, `setL(pin)`, e `setZ(pin)` permitem colocar um pino num dos três estados possíveis.

1.3.3 UART

Esta interface digital conhecida por UART, do inglês *Universal Asynchronous Receiver/Transmitter*, permite a transmissão de informação à custa de um sinal para transmissão (TX) e outro para receção (RX), onde em cada instante de tempo é transmitido apenas um bit em cada sinal (TX/RX), ilustrado na figura 1.28.

Esta interface é também conhecida como sendo série assíncrona pois não tem um sinal de sincronismo (ou relógio) a indicar o instante para ler e escrever informação no canal. Por não ter sinal de sincronismo é bastante sensível a desvios de tempo. Isto obriga a que o controlador faça a contagem do tempo precisa para saber qual o instante de cada bit.

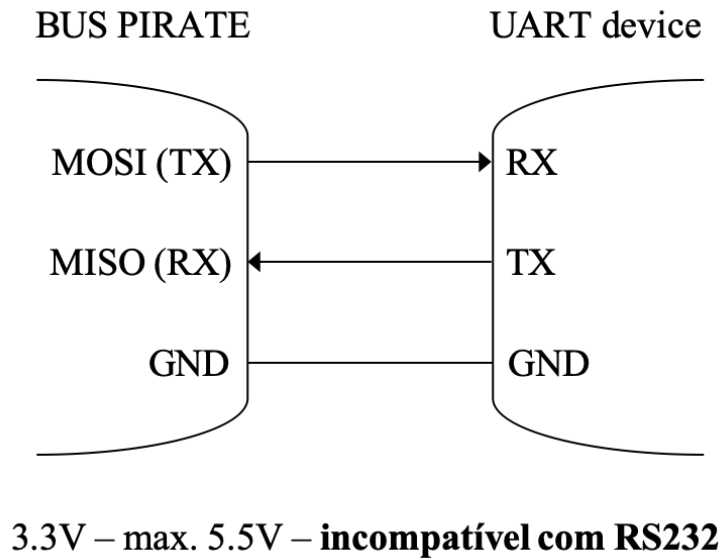


Figura 1.28: Interface UART.

A UART suporta comunicação bidirecional e é a mais simples de todas as interfaces digitais em sistemas embebidos, pois apenas necessita de 3 sinais: TX, RX e GND. A UART existe na maioria dos MCUs atuais e é o controlador responsável pela sequenciação (serialização) e paralelização (des-serialização), ou seja, tr .

A sua implementação mais comum é sobre a forma de portos RS-232. Os portos RS-232 funcionam com níveis de tensão que oscilam entre +12 V e -12 V, e nunca devem ser ligadas diretamente a um MCU sem um conversor de nível, como por exemplo, MAX232, MAX3232, ou MAX3233. Em caso de dúvida, deve-se consultar o *datasheet* dos componentes ou medir a tensão nos pinos TX com um multímetro.

Uma característica do canal série assíncrono é a velocidade de transmissão (*baud rate*), para poder suportar ligações em canais com débitos e periféricos com diferentes condições. Para o *baud rate* de 9600 bps (bits por segundo), cada bit demora 1/9600 segundos a ser transmitido. Quanto maior for o ritmo de transmissão, menor será o tempo de bit e por consequência menor será o despendido na transmissão.

O sinal em repouso encontra-se a VCC. O primeiro bit da transmissão é o um zero, e o bit final é o oposto, para a deixar o canal com tensão presente.

CAPÍTULO 1. INTERFACES DE ENTRADA/SAÍDA

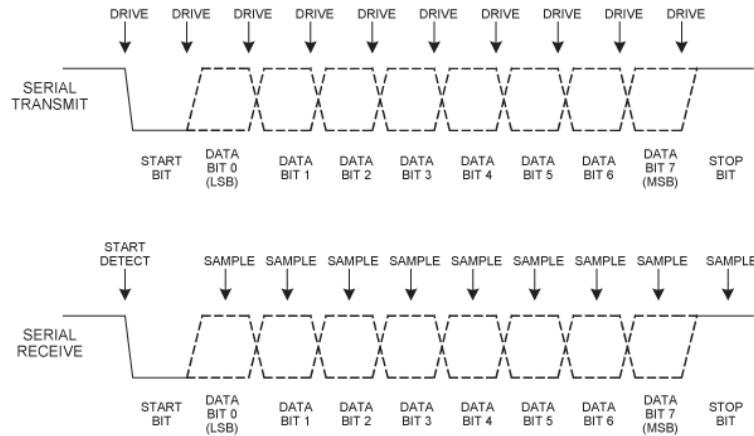


Figura 1.29: uart tx rx.

Em termos de desempenho, o tempo de transmissão corresponde ao tempo dos bits enviados, incluindo os bits de start e stop. Para taxas, ou ritmos, de transmissão elevados, o Arduino ficará menos tempo ocupado com a transmissão. No entanto, nem todos os periféricos conseguem comunicar sem erros a essas velocidades, pelo que terá de ser aferido com o valor máximo para que o sistema funcione corretamente.

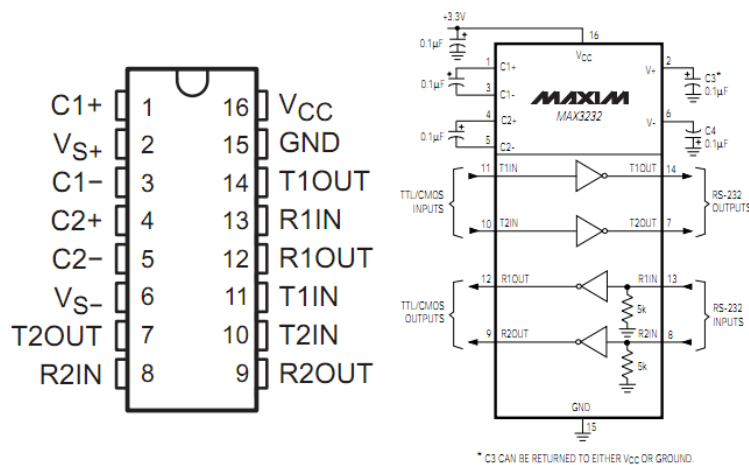


Figura 1.30: Circuito adaptador de níveis de tensão CMOS/TTL para RS-232.

O Sketch ?? apresentado no capítulo 1 demonstra a utilização da UART para receção e envio de caracteres através do terminal Arduino. Neste Sketch o objeto Serial instancia o porto série, e a primeira ação consiste na configuração do seu *baud rate*.

Para aplicações em que o MCU não tenha o número de portos disponíveis para suportar a comunicação dos vários elementos, a alternativa é emular o

porto série em software. Neste caso um par de pinos de E/S digitais podem ser usados para envio e recepção de sequências de bits mantendo o tempo de duração de bit.

No Arduíno a classe **Serial** é responsável pela interação com o controlador UART existente nos MCUs dos Arduínos. Existe também a biblioteca **SoftwareSerial** que permite criar portos de comunicação série assíncrona por software, usada quando não existe um controlador disponível. Esta biblioteca oferece a class **SoftwareSerial** que quando instanciada, recebe os identificadores dos pinos de recepção e transmissão e cria um objeto com os mesmos métodos que a classe **Serial**:

- `begin(baud)` - Configura o controlador UART para transmitir e receber com a velocidade passada como argumento;
- `available()` - Devolve um valor booleano a indicar se existem dados recebidos no controlador UART;
- `read()` - Recebe um carater do controlador UART;
- `write(char)` - Envia um carater para o controlador UART (sem formatação);
- `print/println(char/char[])` - Envia para a UART um carater ou um sequência de caracteres. Esta função permite a especificação do formato dos dados a enviar.

1.3.4 SPI

O canal série síncrono, também conhecido como SPI, derivado da sua designação em Inglês *Serial Peripheral Interface*, permite a comunicação full-duplex entre um mestre (*master*) e um ou mais escravos (*slave*). Dos canais habitualmente usados em sistemas embebidos, o SPI é o que oferece maior desempenho.

Um sistema embebido quando é ligado a um componente SPI funciona como mestre. O mestre é o emissor e envia comandos através uma sequência de bits (ou trama) ao longo do tempo com dados para um escravo SPI, síncrono com o sinal de relógio também por si enviado. Cada trama pode conter 8 ou 16 bits.

O canal SPI é constituído pelo menos por 4 sinais:

- **SCLK** - *Serial Clock*, sinal de sincronismo gerado pelo mestre e enviado para todos os elementos que partilham o canal;
- **SS** - *Slave Select*, sinal de ativação do escravo (ativo = GND). Os escravos num canal partilhado apenas funcionam quando o seu sinal estiver ativo, descartando a informação o resto do tempo;

CAPÍTULO 1. INTERFACES DE ENTRADA/SAÍDA

- MOSI - *Master Output/Slave Input*, saída de dados do mestre/entrada de dados no escravo;
- MISO - *Master Input/Slave Output*, entrada de dados do mestre/saída de dados do escravo.

Para cada escravo existe um sinal SS dedicado para o ativar. O mestre deverá gerar um sinal SS para cada escravo ligado no seu barramento. A figura 1.33 ilustra a ligação de três escravos a um mestre. A figura 1.32 ilustra a transmissão e recepção de bits num canal SPI. Os bits dos sinais MISO e MOSI quando são transmitidos mudam de valor com a transição descendente do sinal SCLK, e na recepção são amostrados na transição ascendente do sinal SCLK. Através da configuração dos modos de funcionamento SPI é possível definir quais as transições do sinal SCLK que são consideradas nas transições e detecções do sinal de relógio. A figura 1.33 ilustra os 4 modos de funcionamento e como eles diferem no funcionamento do canal.

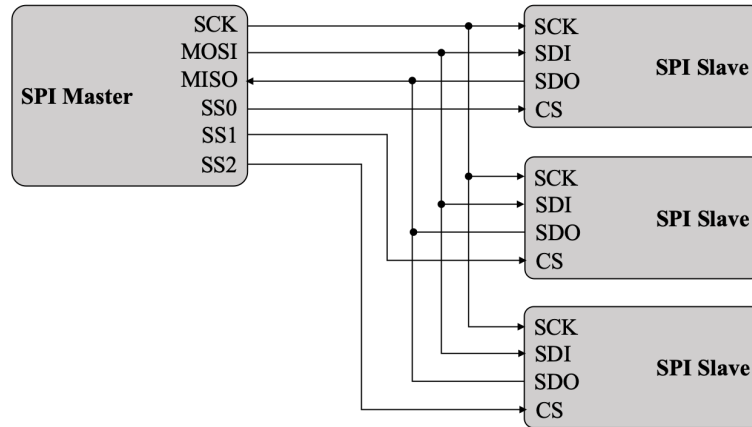


Figura 1.31: Interface SPI.

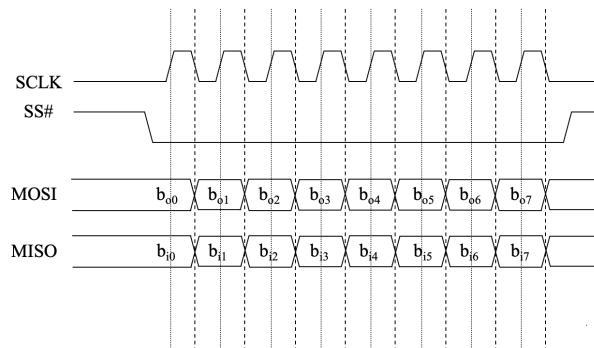


Figura 1.32: Transmissão e recepção num canal SPI.

1.3. INTERFACES DIGITAIS

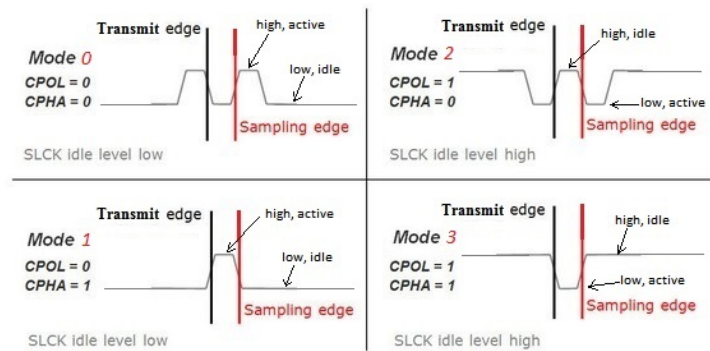


Figura 1.33: Modos de funcionamento da interface SPI.

Num Arduino, para fazer uso da interface SPI é necessário ligar um periférico ao conjunto de pinos dedicados ao SPI, ligados ao circuito controlador interno dentro do MCU. A tabela 1.3 sintetiza a identificação dos pinos em várias placas Arduino e respectivos níveis de tensão máximos. Existe ainda o conector ICSP (*In-Circuit System Programming*) que pode ser usado para ligar os sinais SCLK, MISO e MOSI, ilustrado na figura .

Habitualmente o sinal de relógio é gerado pelo mestre, no entanto podem existir circuitos onde o relógio pode ser gerado por outro componente, devendo o controlador funcionar de acordo com esse sinal.

Arduino	SCLK	MOSI	MISO	SS	Tensão
(todos)	ICSP3	ICSP4	ICSP1	-	5 V
Uno	13	11	12	10	5 V
Due	-	-	-	-	3,3 V
Mega	52	51	50	53	5 V
Leonardo	-	-	-	-	5 V
101	13	11	12	10	3,3 V

Tabela 1.3: Pinos dedicados à interface SPI em várias placas Arduino.



Figura 1.34: Transmissão e recepção num canal SPI.

CAPÍTULO 1. INTERFACES DE ENTRADA/SAÍDA

O barramento SPI tanto pode ser usado para comunicar com periféricos como para estabelecer a ligação entre dois ou mais Arduínos, tal como ilustrado na figura 1.35.

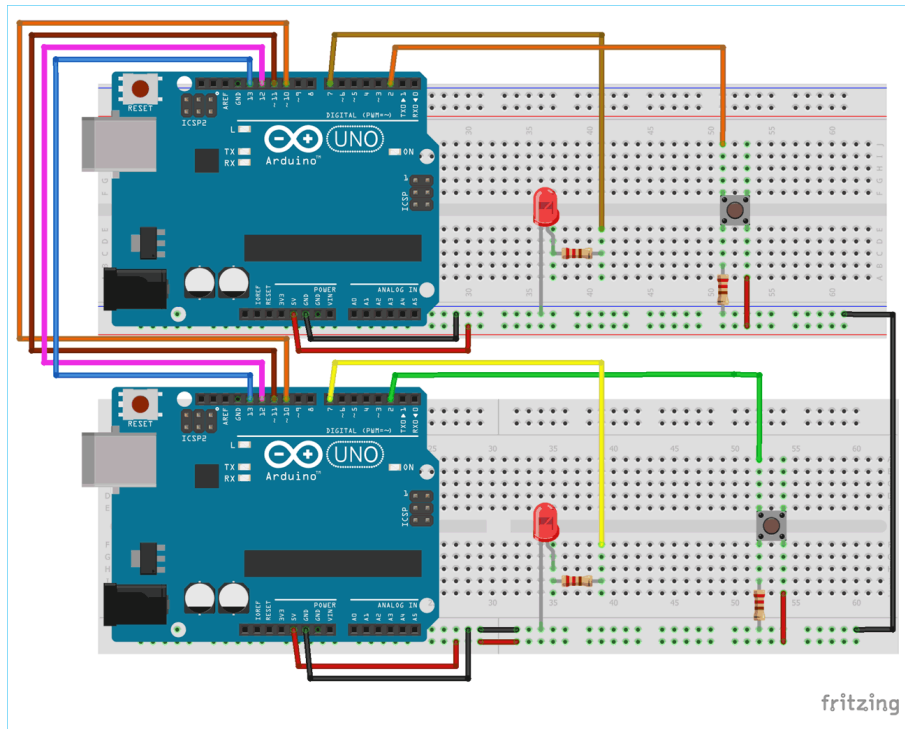


Figura 1.35: Ligação de 2 Arduínos por canal SPI.

A programação do barramento SPI é feita usando o objeto SPI existente na biblioteca SPI.H, disponibilizado através da diretiva `#include <spi.h>`⁴. Esta biblioteca define também a classe `SPISettings` usada para criar objetos com diferentes características de comunicação num barramento SPI, tal como ritmo de transmissão, organização dos dados e polaridade dos sinais. A biblioteca SPI é iniciada através do método `SPI.begin()`, e cada transferência segue a seguinte sequência:

- Indicação do início da transferência - `SPI.beginTransaction(spiSetsA);`
- Seleção do periférico com o qual o Arduino vai comunicar - `digitalWrite(slaveAPin, LOW);`
- Comando, ou pedido, ao periférico - `result = SPI.transfer(op);`
- Desativação do periférico - `digitalWrite(slaveAPin, HIGH);`

⁴É assumido que existe apenas um controlador SPI. Se for necessário usar mais algum será necessário emular em software.

- Término da transação - `SPI.endTransaction()`;

A listagem 1.9 mostra uma demonstração da programação de um Arduino Uno para comunicar com dois periféricos SPI.

Listagem 1.9: Demonstração da programação do barramento SPI

```
// include SPI library
#include <SPI.h>

// two different SPI devices using different SPMODEs
const int spiASelPin = 3;
const int spiBSelPin = 4;

// set up speed, data order and clock/data signals mode
SPISettings spiSetsA(2000000, MSBFIRST, SPMODE1);
SPISettings spiSetsB(16000000, LSBFIRST, SPMODE3);

uint8_t result;

void setup() {
  // set Select Pins as outputs
  pinMode (spiASelPin, OUTPUT);
  pinMode (spiBSelPin, OUTPUT);
  // initialize the SPI bus
  SPI.begin();
}

void loop() {
  SPI.beginTransaction(spiSetsA); // prepare SPI bus for dev A
  digitalWrite (slaveAPin, LOW); // select dev A
  result = SPI.transfer(0); // reading, don't care about argument
  digitalWrite (slaveAPin, HIGH); // deselect dev A
  SPI.endTransaction(); // close SPI bus transaction
  // release SPI bus

  // write on dev B
  SPI.beginTransaction(spiSetsB); // prepare SPI bus for transaction
  digitalWrite (slaveBPin, LOW);
  SPI.transfer(result);
  digitalWrite (slaveBPin, HIGH);
  SPI.endTransaction();
}
```

1.3.5 I2C

O I2C, do inglês *Inter-Integrated Circuit*, é um barramento proprietário da NXP, que permite a comunicação bidirecional através da troca de pacotes de dados a curta distância entre vários mestres e escravos, sendo o cenário mais habitual haver um mestre e vários escravos. Este barramento consegue ritmos de transmissão até 100 kbit/s para o normal e até 400 kbit/s para o modo rápido.

Este barramento faz uso de dois sinais, dados (SDA) e relógio (SCL), e é para ser ligado a portas do tipo open-collector, com resistência de pull-up ligadas a VCC. O sinal de relógio é controlado pelo mestre. Cada componente I2C é caracterizado por ter um endereço de 7 bits, e vem especificado pelo seu fabricante. Deve-se consultar o manual (datasheet) de cada componente para saber qual o valor do seu endereço. Exemplos de sensores I2C:

CAPÍTULO 1. INTERFACES DE ENTRADA/SAÍDA

Sensor	Referência	Fabricante	Endereço I2C
Acelerómetro	ADXL345	Analog Devices	0x53
Giroscópio	L3G4200D	ST Microelectronics	0x69
Magnetómetro	MC5883L	Honeywell	0x1E
Barómetro e Termómetro	BMP085	Bosch	0x77

Uma das grandes vantagens deste barramento ao usar o seu endereço é controlar um conjunto de escravos usando apenas dois sinais de E/S, e não ser necessário o uso de sinais de selecção, ou activação, do escravo com que se pretende comunicar. A figura seguinte mostram a ligação de um Arduino como mestre I2C e vários escravos. Na figura estão também representadas as resistências de polarização (R_p).

As resistências devem ser escolhidas de tal forma que não seja um valor demasiado baixo de tal forma que os componentes não tenham corrente suficiente para impor o nível 0, ou de tal maneira alto que faça com que o sinal demore bastante tempo a variar de 0 para 1, no entanto consome menos corrente. Para um circuito alimentado a 3,3 V, valores aceitáveis de R_p estão entre 1 k Ω e 4,7 k Ω . Em certos cenários podem ser usados valores superiores (20 k Ω), dependendo da velocidade de funcionamento do barramento e do número de escravos ligados ao barramento. Para ter um barramento I2C a funcionar em modo rápido é necessário observar a capacidade parasita entre os sinais SDA e SCL seja inferior a 200 pF, pois quanto maior o seu valor, mais tempo o sinal vai demorar a transitar de valor.

Num Arduino os sinais I2C estão nos pinos A4 (SDA) e A5 (SCL).

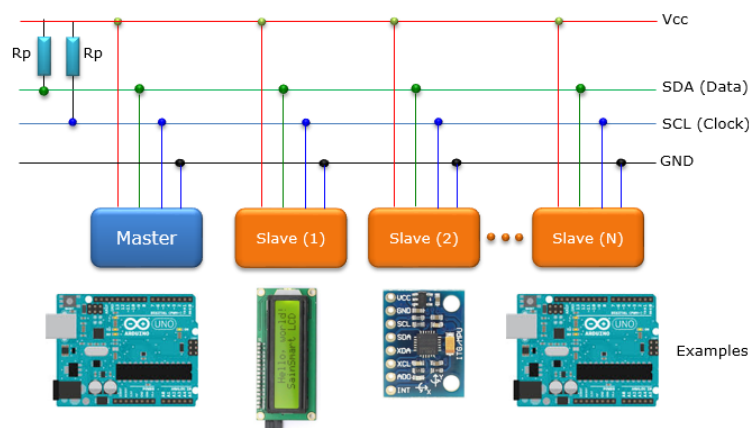


Figura 1.36: Interface I2C.

A operação entre componentes I2C pode ser: transmissão de comandos ou dados pelo mestre, ou transmissão de dados pelo um escravo. Um escravo só escreve depois de ter recebido um pedido de dados por um mestre, e o

mestre ativar o sinal de relógio (SCL). A figura seguinte ilustra o pedido da leitura de temperatura pelo mestre (em cima) e a resposta (em baixo) com o valor da temperatura igual a 100°C.

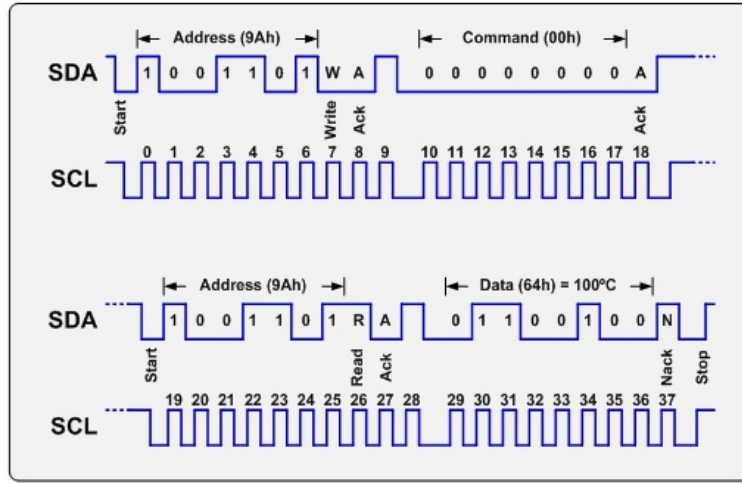


Figura 1.37: Transmissão e recepção num canal I2C.

As funções do I2C estão suportadas na biblioteca Wire, existente na instalação do ambiente Arduino, e acessível através do `#include <Wire.h>`.

Esta biblioteca oferece um conjunto de métodos para suportar as funcionalidades do protocolo I2C:

- `Wire.begin(endereço)`: Função que deve ser chamada para iniciar o controlador do barramento. O argumento endereço (7 bits) é apenas usado para o escravo.
- `Wire.read()`: Função para ler um byte (do mestre ou do escravo). Se for do escravo teve de haver a chamada à função `requestFrom()` por parte do mestre.
- `Wire.write()`: Função para enviar dados do mestre ou escravo.
 - Do mestre: A transmissão deve ser precedida da função `Wire.beginTransmission()` e concluída com a função `Wire.endTransmission()`.
 - Do escravo: O escravo envia dados para o mestre depois deste executar a função `Wire.RequestFrom()` com o pedido de dados.

`Wire.write()` pode ser usada da seguinte forma:

- `Wire.write(valor)`: onde o valor corresponde a um byte.
- `Wire.write(string)`: string: sequência de bytes terminada com o valor 0.

CAPÍTULO 1. INTERFACES DE ENTRADA/SAÍDA

- `Wire.write(dados, dimensão)`: `dados`: ponteiro para uma sequência de bytes, `dimensão`: número de bytes a transmitir.
- `Wire.beginTransmission(endereço)`: função usada para iniciar a comunicação com um escravo I2C (identificado com o endereço de 7 bits).
- `Wire.endTransmission()`: função usada para terminar a comunicação com um escravo `beginTransmission()`.
- `Wire.onRequest()`: função chamada após recepção do pedido de dados `Wire.requestFrom()` do mestre. O escravo pode enviar dados para o mestre via `Wire.write()`.
- `Wire.onReceive()`: função chamada no escravo após recepção de dados do mestre. Pode-se usar `Wire.read()` para ler os dados enviados pelo mestre.
- `Wire.requestFrom(endereço, quantidade)`: função usada pelo mestre para pedir um conjunto de bytes ao escravo. O mestre deve depois usar a função `Wire.read()` para ler os dados enviados pelo escravo. O argumento `endereço` identifica o escravo, e `quantidade` o número de bytes a receber.

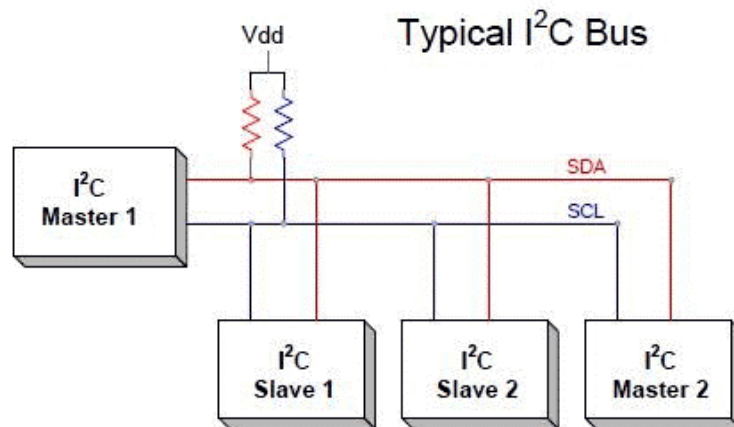


Figura 1.38: Esquema de ligação entre um Arduino mestre e vários Arduínos escravos I2C.

Listagem do Sketch `i2cDemo.ino`:

Listagem 1.10: `i2cDemo.ino`

```
// I2C DEMO - Communication between Two Arduino
// for Arduino Uno use pins A4+A5

#include <Wire.h> // I2C Lib

#define MASTER
```

1.3. INTERFACES DIGITAIS

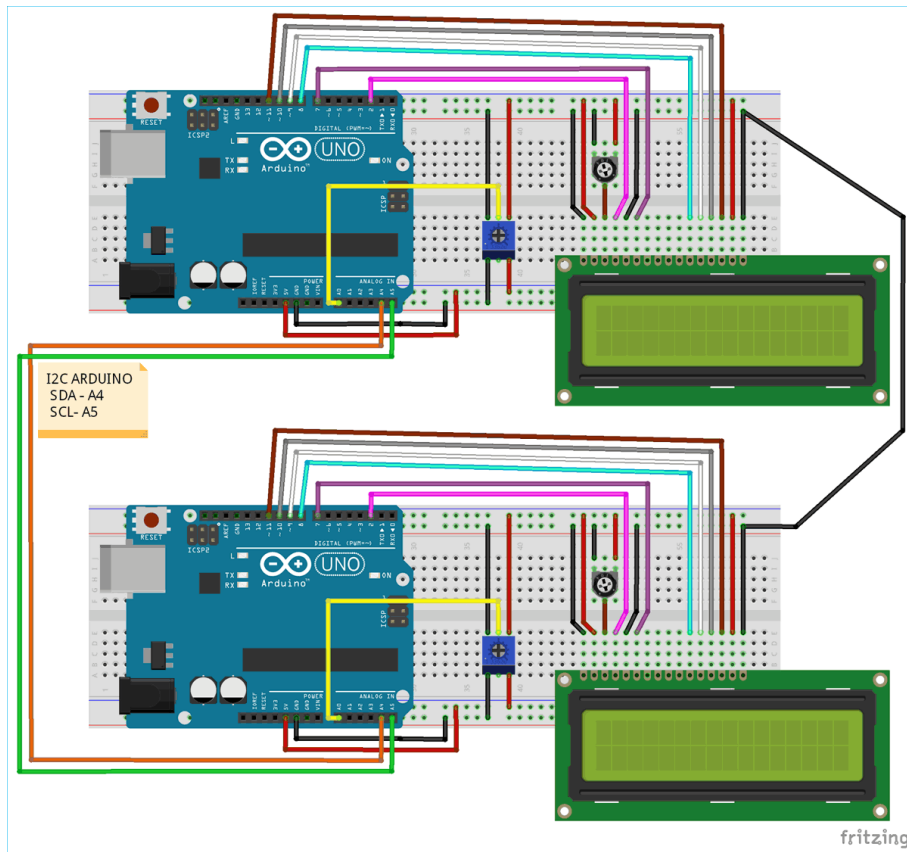


Figura 1.39: Ligação entre dois Arduino via I2C.

```
//#define SLAVE

#if MASTER
void setup()
{
  Serial.begin(115200); while(!Serial);
  Wire.begin(); // start I2C communication at
  while (!Serial) ; // wait for serial port to connect.
}

void loop()
{
  Wire.requestFrom(8,1); // request 1 byte from slave arduino (8)
  byte MasterReceive = Wire.read(); // receive a byte from the slave arduino and store in MasterRec

  Wire.beginTransmission(8); // start transmit to slave arduino (8)
  Wire.write(MasterSend); // sends one byte converted POT value to slave
  Wire.endTransmission(); // stop transmitting
  Serial.println("Master Received From Slave"); //Prints in Serial Monitor
  Serial.println(MasterReceive);
  delay(500);
}
#endif

#if SLAVE
byte dataRx = 0;

void setup()
{
```

CAPÍTULO 1. INTERFACES DE ENTRADA/SAÍDA

```
Serial.begin(115200); while(!Serial);
Serial.println(".:Starting I2C Slave:."); // welcome message
Wire.begin(0x10); // set slave address (7 bits)
Wire.onReceive(rcvFunc); // function called when master sends a value
Wire.onRequest(rqtFunc); // function called when master requests a value
}

void loop(void)
{
  Serial.println(SlaveReceived);
  delay(500);
}

void receiveEvent (int howMany) //This Function is called when Slave receives value from master
{
  SlaveReceived = Wire.read(); //Used to read value received from master and store in variable
}

void requestEvent() //This Function is called when Master wants value from slave
{
  int potvalue = analogRead(A0); // Reads analog value from POT (0-5V)
  byte SlaveSend = map(potvalue,0,1023,0,127); // Convert potvalue digital value (0 to 1023) to (0 to 127)
  Wire.write(SlaveSend); // sends one byte converted POT value to master
}
#endif
```

1.3.6 CAN

O *Controller Area Network*, conhecido como *CANbus*, é um barramento usado para troca de mensagens entre vários componentes de um sistema usando apenas um par de fios (CAN+ e CAN-), e é muito usado em sistemas automóveis, industriais, experiências científicas e espaciais. A comunicação é bidirecional, sendo iniciada por um mestre com comandos ou pedidos de dados para um ou mais escravos. A norma que rege o funcionamento do barramento CAN a nível elétrico é o ISO 11898, e o CANopen a nível de aplicação. O paralelismo com o modelo OSI é ilustrado na figura 1.41.

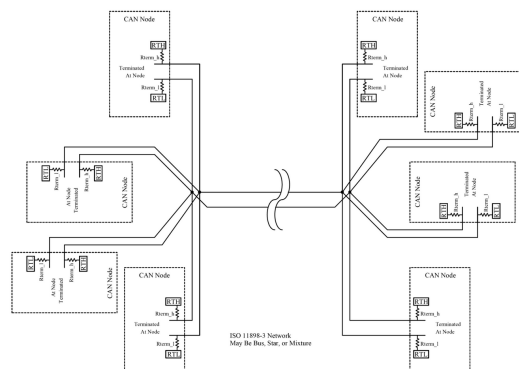


Figura 1.40: Ilustração das ligações numa rede CAN com vários nós.

A nível físico, as formas de onda dos sinais CAN+ e CAN- são simétricas, tal como ilustrado na figura 1.42, para facilitar a discriminação do valor dos bits. A grande vantagem deste barramento é usar um sinal diferencial em

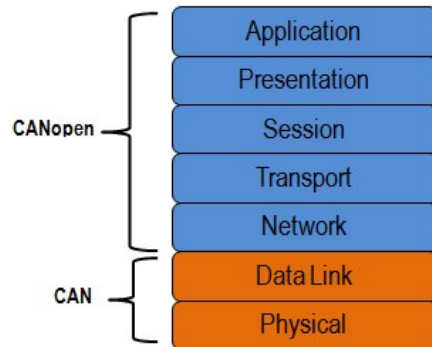


Figura 1.41: Comparação entre as normas existentes para o CAN e o modelo OSI.

vez de sinal em modo comum para transmissão de dados, onde um bit a 1 é dado por uma tensão positiva entre o sinal CAN+ e CAN-, e o bit 0 quando existe uma tensão negativa entre o sinal CAN+ e CAN-. No barramento CAN o bit a 1 é conhecido como estado recessivo (derivado dos pull-ups) e o bit a 0 como estado dominante. Isto significa que quando não existe nenhum nó a transmitir, por omissão o barramento encontra-se a 1, e quando um terminal faz uso do canal, ele força a tensão apenas dos bits a 0. Graças a esta característica é possível usar uma sequência predefinida no início de cada transmissão para determinar se houve outro nó a transmitir em simultâneo.

Em ambiente de laboratório, os sistemas tendem a funcionar de forma controlada e correta. No entanto, em ambientes ruidosos e com interferências, como algumas instalações industriais ou aeroespaciais, os sinais tendem a ser perturbados e a discriminação dificultada. Uma forma de aferir a qualidade do sinal é através do diagrama de olho, ilustrado na figura 1.43. Este diagrama é semelhante à forma de onda produzida por um osciloscópio, no entanto foca apenas a duração de um bit, retendo todas as transições de bits a 0 e 1. Isto faz com que várias curvas sejam sobrepostas. Com os sinais bem discriminados entre 0 e 1 é fácil observar a sua separação, o "olho aparece aberto". Com a presença de ruído os sinais tendem a ter transições cada vez menos perfeitas, chegando a aproximarem-se, e o "olho aparece fechado".

A ligação de vários elementos a um barramento CAN constitui a rede CAN, que é ilustrada na figura 1.44. Nas extremidades da rede é necessário ter resistências de $100 - 120\Omega$ como elementos terminais. Nesta figura cada nó tem uma interface de ligação (transceptor) que faz a tradução dos níveis de sinal diferenciais por sequências de 0s e 1s. Este componente é também responsável pela identificação de colisões na comunicação, quando pelo menos dois componentes comunicam em simultâneo. O controlador é responsável pela sequenciação dos bits enviados e paralelização dos dados recebidos.

Apenas algumas placas da plataforma Arduino suportam o barramento

CAPÍTULO 1. INTERFACES DE ENTRADA/SAÍDA

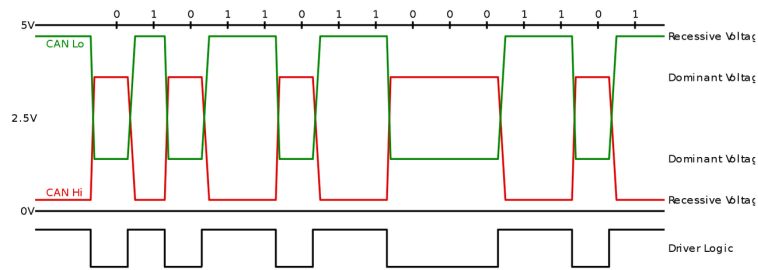


Figura 1.42: Variação das tensões CAN+ e CAN- para transmissão de uma sequência de bits num barramento CAN.

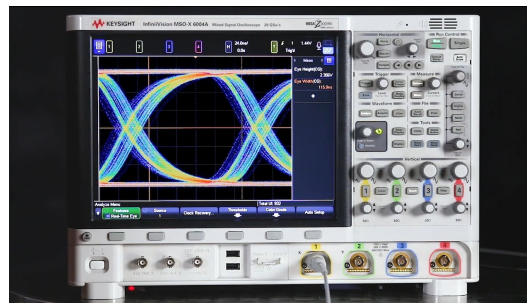


Figura 1.43: Diagrama de olho.

CAN, nomeadamente a Due que tem um controlador presente no seu MCU, sendo necessário a utilização de controladores externos para suportar essa funcionalidade nas restantes placas. A nível da programação, a diferença passa por comunicar diretamente com o controlador embebido ou com um periférico, por exemplo, via SPI.

Quando se implementa uma rede CAN pode ser difícil analisar qual o componente que não está a funcionar corretamente, o mestre ou o escravo. A depuração de uma rede CAN pode ser feita com um analisador CAN, que é colocado em paralelo com os restantes elementos da rede CAN, e serve para registar e analisar todas as transações.

O CANopen, que rege os protocolos entre as camadas de rede e aplicação, especifica o tipo de mensagens e objetos trocados pelos nós, e define um conjunto de requisitos mínimos, a serem implementados em dispositivos com pouca memória e capacidade de processamento, de modo a suportarem um conjunto de funcionalidades que sejam compatíveis entre si. Existem bibliotecas que implementam algumas partes do CANopen nomeadamente

1.3.7 1-Wire

O 1-Wire é um barramento de 1 bit que permite a ligação de dados e alimentação a um componente apenas com um sinal para além de GND, usando no total um par de fios! Os componentes que usam este barramento usando o

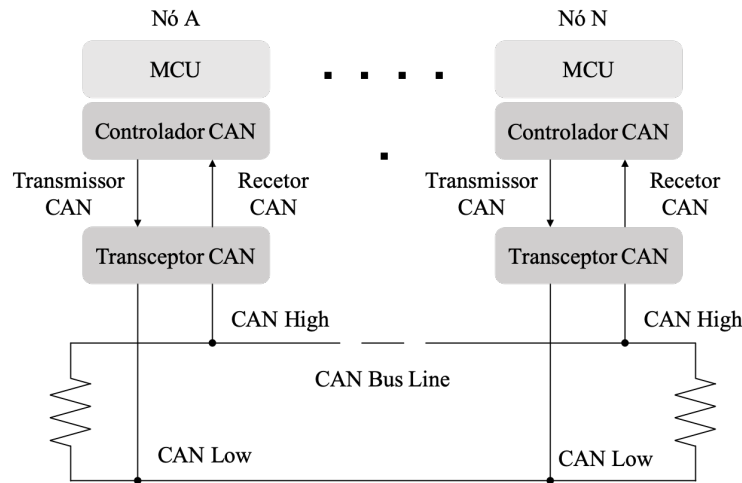


Figura 1.44: Interface CAN.



Figura 1.45: Analisador CAN de 2 canais.

sinal de dados como fonte de alimentação parasita. Por esse motivo, os componentes 1-wire consomem muito pouca energia e são habitualmente usados para autenticação, sensores de temperatura, registadores de dados, memórias não-voláteis e relógios de tempo real. A comunicação é série assíncrona, e é iniciada pelo sistema embebido e destinada a um ou mais componentes 1-wire. Cada componente tem um identificador único com 64 bits.

A nível elétrico, o barramento 1-wire, tem um *pull-up* ($5\text{ k}\Omega$) a VCC, o que permite realizar a alimentação parasita dos componentes, ilustrado na figura 1.46. Quando os componentes 1-wire são soldados na PCB, podem-se ligar os terminais de alimentação. Os componentes têm internamente um pequeno condensador que os permite funcionar enquanto o sinal do barramento é usado para comunicação. Os componentes 1-wire estão sempre com o sinal de dados em alta-impedância à exceção de quando tiverem de enviar dados pedidos pelo sistema embebido. Como existe um *pull-up* no sinal do

CAPÍTULO 1. INTERFACES DE ENTRADA/SAÍDA

barramento, os componentes apenas têm de forçar o sinal a GND quando tiverem de transmitir um bit com valor 0.

A figura 1.47 mostra os tempos necessários para transmitir e receber dados com o protocolo 1-wire. A nível da transferência de informação com o componente 1-wire, como nem todos os sistemas embebidos possuem controladores dedicados para serializar e paralelizar a informação do canal, pode ser usada um porto de comunicação série assíncrono com 9600 e 115200 baud para emular os tempos do protocolo 1-wire. Existem bibliotecas disponíveis para Arduino prontas a usar, nomeadamente a **OneWire**.

Para comunicar com um periférico o sistema embebido é realizada uma sequência de operações semelhante a um protocolo de rede: faz reset ao barramento (início de sessão), seleciona o componente com o qual vai a comunicar (estabelecimento da camada de ligação) e dá o comando/operação a esse componente (camada de transporte e dispositivo).

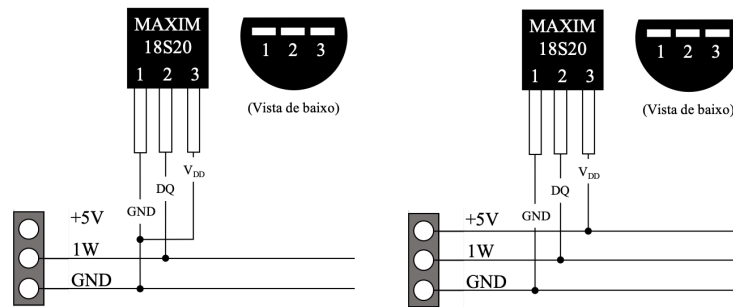


Figura 1.46: Interface 1-Wire.

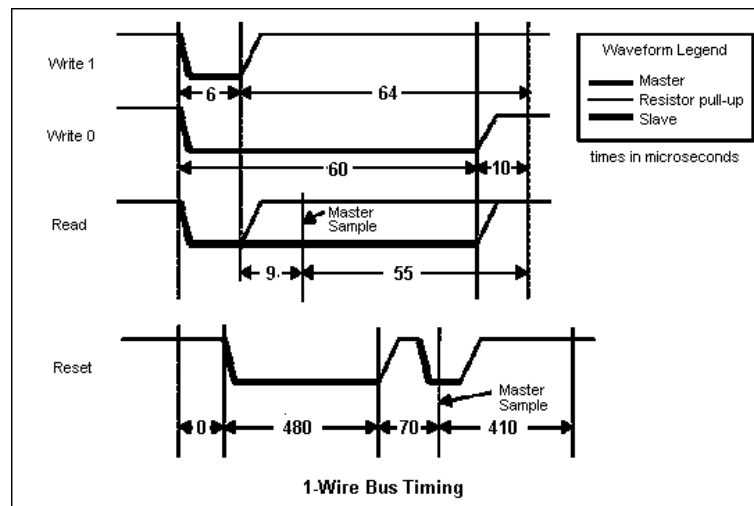


Figura 1.47: Transmissão e recepção num canal 1-Wire.

1.4 Isolamento Galvânico

Em muitos circuitos é necessário garantir o isolamento galvânico entre componentes, sem condução direta, mas permitindo a transferência de energia entre eles. Em termos práticos, quer-se que se passe um sinal elétrico, mas que uma parte do circuito esteja protegida se houver alguma anomalia no resto do circuito. Aplicações habituais deste tipo de componente são: suprimir o ruído de equipamento áudio de alta-fidelidade, isolamento entre o paciente e o equipamento médico para monitorização de sinais fisiológicos, e circuitos para medidas de precisão com elevada impedância de entrada. Os tipos de componentes mais usados para isolar circuitos são: transformadores, opto-isoladores e os isoladores-CMOS.

Os “anéis de massa”, ou *Ground-Loops*, são um problema que pode ocorrer quando diferentes tipos de circuitos estão interligados e existem correntes parasitas a circular entre eles. No equipamento áudio manifestam-se como um zumbido audível em torno dos 50Hz. Para além do isolador entre sinais digitais e analógicos, uma boa prática é referenciar as massas dos circuitos à terra, sempre que seja possível.

1.4.1 Transformador-Isolador

O transformador isolador é usado para isolamento galvânico de dois componentes. É construído de forma semelhante a um transformador de alimentação, mas usa fios de secção menor pois é usado para sinais de baixa corrente. Tal como qualquer outro transformador, este também apenas permite a passagem de sinais variantes no tempo, bloqueando a passagem de sinais DC. Os transformadores de isolamento são aplicados em sistemas onde é necessário que não exista passagem de corrente pela massa dos circuitos, e eventuais contaminações com tensão de modo de comum entre eles. A figura 1.48 mostra alguns transformadores de isolamento e a sua aplicação típica.

1.4.2 Opto-Isolador

Os opto-isoladores, ou opto-acopladores, são componentes semicondutores que integram no mesmo encapsulamento um emissor de luz (LED) e um detector de luz (foto-diodo ou foto-transistor), sem qualquer ligação eléctrica entre eles, sendo ligados a dois circuitos electrónicos separados. Eles são usados para monitorização e controlo de circuitos de altas tensões, e isolamento de sinal GND ou terra. O isolamento é aplicado a sinais unidireccionais e ronda os 10kV.

Este circuito é útil para acoplar circuitos digitais cujos níveis de tensão sejam diferentes, por exemplo: circuito ligado à rede eléctrica (220V) ou automóvel (12V) e um Arduino (3,3 V ou 5 V). Tal como um LED normal, é necessário limitar a corrente no circuito por meio de uma resistência ligada

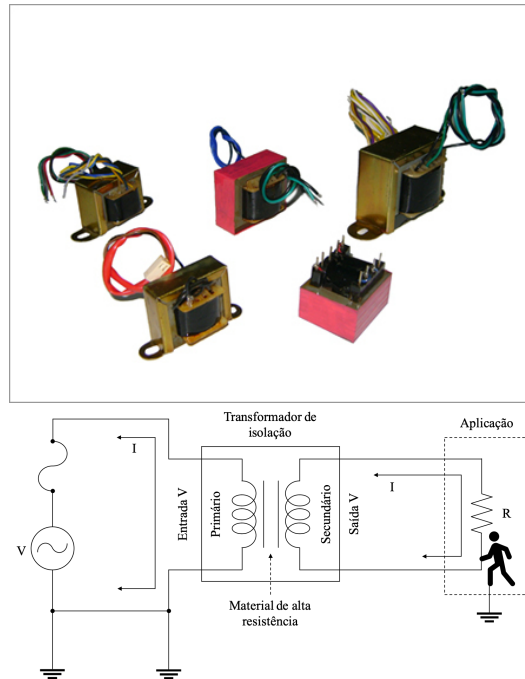


Figura 1.48: Exemplos de transformadores de isolamento (esquerda) e esquema de aplicação de um transformador de isolamento.

em série com a entrada do LED do opto-acoplador para que ele não queime. A saída do opto-acoplador, criada pelo foto-transistor funciona como um circuito de colector-aberto. Um opto-acoplador está limitado a funcionar até uma determinada frequência máxima. Deve ser também observada qual a tensão máxima aos terminais do transistor.

É necessário cuidado na realização de circuitos analógicos devido às não-linearidades dos LEDs e do ganho dos foto-transistores - *Current Transfer Ratio* (CTR). Existem variantes de opto-isoladores que em vez de um transistor têm DIACs, TRIACs, ou tiristores, usados para controlo de cargas de grande potência. Existem também opto-isoladores que são construídos com um espaço entre o emissor e o receptor para serem usados na detecção de obstáculos sem usar contactos metálicos, evitando o risco de choques elétricos e o desgaste do material condutor (por exemplo: rotary encoder).

Algumas referências opto-acopladores: TIL111 (ON Semiconductor), 4N28 (Vishay), BPW34 (OSRAM). A escolha de um opto-isolador deve ter em conta os níveis máximos de tensão e a frequência máxima de funcionamento.

1.4.3 Isolador-CMOS

O isolador CMOS é um circuito integrado complexo, onde os sinais são transferidos da entrada para a saída através de acoplamento magnético, usando um transmissor e um receptor rádio de muito curto alcance, a funcionar em

1.4. ISOLAMENTO GALVÂNICO

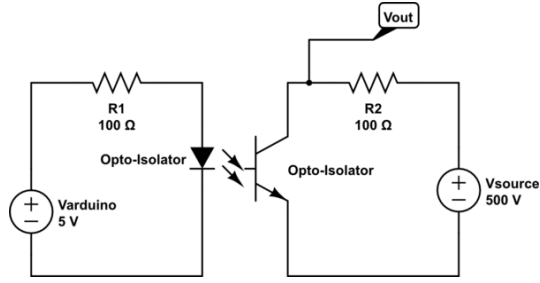


Figura 1.49: Esquema de um opto-isolador.

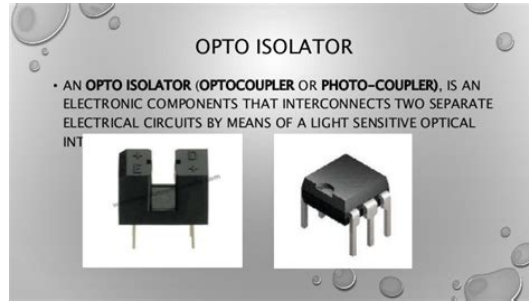


Figura 1.50: Dois opto-isoladores.

frequências entre os 180 e os 300 MHz. Na recepção, o sinal recebido é convertido num sinal digital. Existem soluções da Analog Devices e da Silicon Labs, que oferecem várias configurações de número de entradas e saídas. O esquema de funcionamento desta família de circuitos integrados, juntamente com uma aplicação que faz uso de isolamento bidirecional, são ilustrados na figura seguinte. A figura 1.51 ilustra a construção do dispositivo da Silicon Labs, e um exemplo de aplicação do ADuM2250 para isolar o sistema embebido ao barramento I2C.

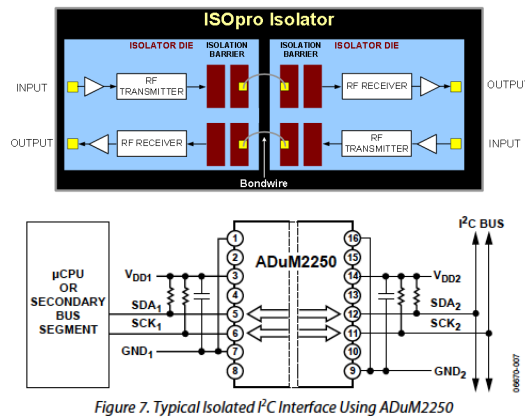


Figure 7. Typical Isolated I²C Interface Using ADuM2250

Figura 1.51: Isolador CMOS.

1.5 Conversão Analógico-Digital

A conversão analógico-digital consiste na observação, ou amostragem, do valor de um sinal analógico num determinado instante e quantificá-lo, ou traduzi-lo, por um valor numérico. É a operação inversa da conversão digital-analógico.

O sinal analógico é contínuo e representa uma quantidade obtida através de sensores, sendo apenas limitado pela precisão do sensor e do ruído introduzido pelo sistema. O sinal digital não sofre qualquer degradação à medida que é processado, no entanto ao ser passado para digital, no entanto, existem perdas na conversão de analógico para digital.

1.5.1 Quantização

Quando um sistema embebido processa um sinal analógico, através da execução de um conjunto de instruções, ele é convertido em digital através de um conversor analógico-digital. Este processo designa-se por quantização ou discretização, uma vez que se deixa de ter um sinal contínuo para ter um sinal discreto. Os sinais discretos são uma representação que apenas existe dentro dos computadores. Para uma determinada gama de tensões analógicas corresponde um conjunto de valores inteiros, tantos quantos os números de bits existentes no *Analog-to-Digital Converter* (ADC).

Como os valores discretos possíveis são espaçados em 2^N intervalos, onde N é o número de bits usados para representar os valores discretos. Quanto maior for N , mais pequenos são os intervalos entre valores, e mais próximo do sinal original será a representação discreta. No entanto, por muito próxima que a representação esteja do sinal original, existirá sempre um erro associado à quantização. No pior dos casos o erro é metade do valor do bit menos significativo, podendo ser positivo ou negativo. Considerando que um sinal produz a mesma quantidade de sinais discretos (uniformemente distribuído) a relação entre o sinal e o ruído de quantização é definido pela expressão:

$$SQNR = 20\log_{10}(2^N)[dB] \quad (1.2)$$

Por exemplo, o sinal áudio de um CD áudio, quantizado com 16 bits, tem no máximo 96 dB.

A figura 1.52 ilustra a quantização de um sinal analógico com 3 bits, ou seja, 8 valores discretos possíveis. É possível observar que o valor discreto fica referenciado ao centro do intervalo discreto, ou seja, $(V_{max} - V_{min}) \times 2^{-N} / 2$.

Um sinal analógico que varie entre 0 V e 5 V, se for convertido com um ADC de 10 bits, significa que vai ter 2^{10} valores possíveis para corresponder a intervalos de tensões analógicos. Assumindo que os intervalos estão distribuídos uniformemente, a diferença entre cada valor inteiro na saída do ADC vai corresponder a uma diferença de $(5 - 0) / (2^{10}) = 4,88 \text{ mV}$

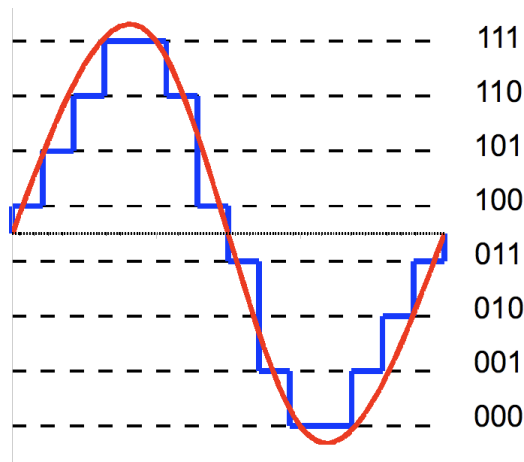


Figura 1.52: Exemplo de quantização com 3 bits.

1.5.2 Ritmo de Amostragem

É intuitivo que a aquisição do sinal deve ser mais rápida do que a variação do sinal ao longo do tempo, caso contrário não vai observar os detalhes do sinal. Neste domínio. O “Teorema da Amostragem de Nyquist-Shannon” determina que se deve amostrar um sinal contínuo (converter um valor analógico numa sequência de valores discretos) a pelo menos ao dobro da frequência do sinal sob pena de não ser possível recuperar o sinal (*sub-sampling*).

Na prática, quanto mais rápido um sinal for amostrado, mais fiel é a reconstrução digital do mesmo, tal como ilustrado na figura 1.53. Seriam necessárias infinitas amostras para poder reconstruir o sinal perfeitamente. Quando um sinal é amostrado a um ritmo inferior ao dobro da sua frequência máxima, são produzidas distorções, ou artefactos, que podem tornar o sinal irreconhecível. Para suprimir este efeito, utilizam-se filtros anti-aliasing para limitar o espectro do sinal antes de ser amostrado. A título ilustrativo um CD áudio tem o sinal amostrado a 44,1 kHz ao passo que os DVDs áudio usam 192 kHz.

No projecto de um sistema embebido é importante observar o ritmo de amostragem para os sinais envolvidos de forma a determinar os seus requisitos para poder realizar o processamento digital de sinais. Neste caso é necessário saber qual a frequência do sinal, quantas instruções são necessárias para implementar um algoritmo, e a quantidade de instruções que o CPU executa por segundo.

Por exemplo, será que um *Central Processing Unit* (CPU) a funcionar a 10 MHz é suficiente para um sistema processar um sinal com uma frequência de 500 Hz, sabendo que é necessário executar 100 instruções, que em média necessitam de dois ciclos de relógio, para processar uma amostra? Neste caso é preciso determinar quanto tempo demora uma amostra a ser processada e depois comparar com o tempo do período da onda:

$$\begin{aligned}
 T_{proc\ amostra} &= T_{instr} \times num\ instr \\
 T_{instr} &= \frac{1}{f_{CPU}} = \frac{1}{10 \times 10^6} = 100\ ns \\
 T_{proc\ amostra} &= (100 \times 10^{-9}) \times 100 = 10 \times 10^{-5} = 0.1\ ms \\
 T_{amostra} &= \frac{1}{2 \times freq\ sinal} = \frac{1}{1000} = 1\ ms \\
 T_{amostra} &> T_{proc\ amostra} \rightarrow OK!
 \end{aligned}$$

Sabendo que temos margem para processar mais amostras do que as duas impostas pelo ritmo de Nyquist, qual é o valor máximo do número de amostras por ciclo que o sistema é capaz de processar? O sistema é capaz de processar 20 amostras.

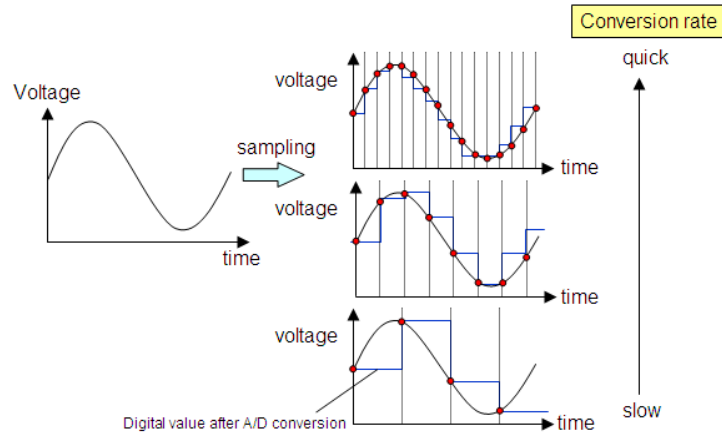


Figura 1.53: Exemplo dos diferentes ritmos de amostragem, e dos sinais digitais resultantes.

1.5.3 Conversão Analógico-Digital com Arduino

Nos sistemas ciber-físicos, onde é necessário realizar cálculos sobre medidas de um fenómeno físico no sistema, ou ambiente, é necessário quantificar um sinal analógico numa medida digital, tal como ilustrado na figura 1.54. Para tal é importante que a conversão analógico-digital cumpra com os requisitos do sistema:

- Resolução - quantidade de valores disponíveis para representar um valor do sinal (também pode ser visto como o intervalo de tensões entre cada valor numérico/binário, deve ser o menor possível);

- Precisão - quantificação do erro associado à medida (é medido em bits menos significativos (Least Significant Bits (LSBs)) e deve ser o menor possível);
- Ritmo de aquisição - o ritmo de amostragem deve ser tal que permita o sistema reagir ao evento. Se for mais baixo que o limiar indicado pelo Ritmo de Nyquist (*undersampling*) perdem-se amostras do sinal. Quanto mais elevada for a amostragem mais fidedigno será o seu processamento;

Um ADC deve ser escolhido de acordo com os seguintes parâmetros: número de bits, ritmo de amostragem, tensão de funcionamento, tipo de canal digital; e pode ter diferentes tipos de realizações:

- comparador paralelo - constituído por vários circuitos comparadores (AMPOPs) ligados em paralelo ao sinal de entrada e ajustados para diferentes tensões, ilustrado na figura 1.57;
- contador - um DAC é usado para gerar a tensão de comparação. A contagem é parada assim que o valor à entrada for atingido.
- aproximações sucessivas - também usa um DAC, mas os valores usados vão sendo refinados em cada aproximação, em vez de realizar a contagem como no caso anterior. Este é o tipo de ADC existente no MCU do Arduino Uno R3;
- outros - existem outras realizações, mas estão fora do âmbito dos sistemas embebidos.

Após as amostras do sinal recolhidas, é necessário formar, ou codificar, o sinal digital. Uma das técnicas mais simples de codificação de valores analógicos em digitais é o *Pulse-Code Modulation*, ou Pulse Coded Modulation (PCM), onde um sinal analógico é amostrado a um ritmo constante, e registado como uma série dos valores digitais das suas amostras. Esta foi a forma usada durante muito tempo pois exigia menos capacidade de processamento, no entanto requer mais capacidade de armazenamento. Hoje em dia, os controladores digitais permitem implementar diferentes técnicas de compressão (exemplo mp3) que produzem ficheiros menores e com qualidade próxima do PCM.

No Arduino Uno R3, o ADC interno tem as seguintes características:

- 10 bits de resolução ($0 : 2^{10} - 1 = 0 : 1023$);
- Não linearidade a 0.5 LSB;
- precisão ± 2 LSB Absolute Accuracy

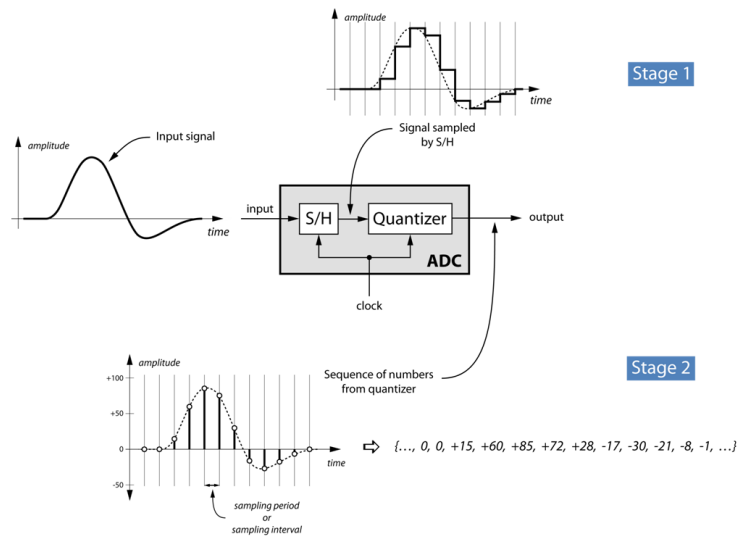


Figura 1.54: adc1.

- Tempo de conversão: 13-260 us
- Até 76.9 kSPS (até 15 kSPS na resolução máxima)
- 6 canais de entrada multiplexados;

A listagem 1.11 ilustra a utilização do ADC do Arduino na leitura de dois valores analógicos, e apresenta a conversão do valor em diferentes formatos.

Listagem 1.11: adcDemo.ino

```
#define ANINPUT_MIN 0
#define ANINPUT_MAX 1023
#define AN_VAL_MIN 0
#define AN_VAL_MAX 5

int adcPin = A0;
int adcVal = 0;
float anVolt = 0.0;
int an_map = 0;
char buf[80], str[10];
String str2;

int mapr(int x, int in_min, int in_max, int out_min, int out_max) {
    float aux = (float)(x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
    return round(aux);
}

float mapf(int x, int in_min, int in_max, int out_min, int out_max) {
    return round(((float)(x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min));
}

void setup()
{
    Serial.begin(115200); // setup serial 115200 baud
    while(!Serial);
    #if 1
        int f, r, m;
        Serial.println("Val,\tmap,\tmapr,\tmapf");
        for(int i = 0; i < 32; i++) {
```

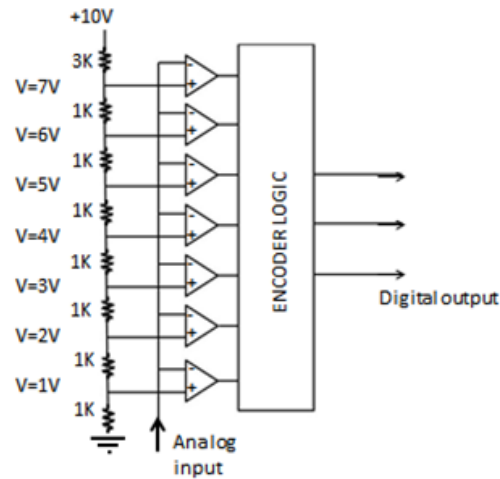


Figura 1.55: adc2.

```

m = map(i, 0, 31, 0, 5);
r = mapr(i, 0, 31, 0, 5);
f = mapf(i, 0, 31, 0, 5);
sprintf(buf, "%d,\t%d,\t%d,\t%d", i, m, r, f);
Serial.println(buf);
}
Serial.println("end of test!");
#endif
}

void loop()
{
  adcVal = analogRead(adcPin);
  an_map = map(adcVal, AN.INPUT.MIN, AN.INPUT.MAX, AN.VAL.MIN, AN.VAL.MAX);
  anVolt = mapf(adcVal, AN.INPUT.MIN, AN.INPUT.MAX, AN.VAL.MIN, AN.VAL.MAX);
  str2 = String(anVolt);
  dtostrf(anVolt, 4, 2, str);
  sprintf(buf, "%d: %s = %s = %d", adcVal, str, str2.c_str(), an_map);
  Serial.println(buf);
  delay(500);
}

```

Na conversão dos valores quantificados pelo ADC do Arduino (0-1023) num valor de tensão correspondente (0-5 V) é importante notar que a função `map()` realiza o ajuste de escala através da determinação da derivada no ponto amostrado: $(x - in_{min}) * (out_{max} - out_{min}) / (in_{max} - in_{min}) + out_{min}$. No entanto a função `map()` opera sobre valores inteiros, o que introduz erro por defeito nos cálculos. Em termos práticos, é como se o valor fosse convertido para o valor inteiro anterior, mesmo que a diferença para o próximo valor seja mínima ($4,99 \rightarrow 4$).

A listagem 1.11 apresenta a função `mapf()`, uma alternativa à função `map()` que realiza os mesmos cálculos mas em virgula flutuante, e a função `mapr()` que realiza o arredondamento ao valor inteiro mais próximo, com a função `round()`, minimizando o erro de quantização. Em ambos os casos os cálculos internos são realizados em virgula flutuante.

Para aplicações onde o erro de quantização para valores inteiros pela

função `map()` introduz comportamento indesejado, a alternativa é considerar os valores de todas as tensões em milivolt ($1\text{ V} = 1000\text{ mV}$) realizando a operação `map(adcVal, 0, 1023, 0, 5000);`.

1.6 Conversão Digital-Analógico

A conversão de um sinal digital noutro analógico é usada para traduzir uma quantidade numérica numa quantidade física. Os sistemas embebidos, especialmente os ciber-físicos que interagem com o meio, necessitam de variar gradualmente uma grandeza física, por exemplo a intensidade luminosa de um LED, resistência de aquecimento ou a velocidade de um motor DC. A conversão digital-analógico pode ser vista como a operação inversa da conversão analógico-digital.

A tradução de um valor digital noutro analógico, normalmente numa tensão, é conseguida através de um conversor digital-analógico (DAC). Convertendo vários valores binários à entrada de um DAC ao longo do tempo, é possível reconstruir um sinal aproximado de um sinal analógico. O sinal diz-se aproximado, porque nunca vai ter as mesmas características de um sinal analógico.

Existem diferentes tipos de DAC, sendo a sua escolha dependente do tipo de aplicação, nomeadamente: áudio, video, controlo de veículos, medicina. Alguns destes tipos de DACs são:

- Modulação por largura de pulso - *Pulse Width Modulator* (PWM) - Converte o valor num sinal digital com a duração do valor 0 ou do 1 quanto menor ou maior for o valor a converter.
- Delta-Sigma - Conversor de sobre-amostragem ou de interpolação - Conversão baseada em densidade de pulsos. Existe apenas um sinal de um bit que varia muito mais rapidamente que o sinal analógico. A quantidade de bits a 1 a dividir pelo número total de bits numa sequência determina o nível de tensão analógico.
- Pesos binários - Este DAC usa resistências (ou fontes de corrente) ligadas a todos os bits da palavra que contém o valor representado. O outro terminal de todas as resistências são ligados num ponto, onde as várias contribuições são somadas. O valor de cada resistência depende do peso de cada bit.
- Escada R-2R - Semelhante ao anterior, mas com uma estrutura que melhora a precisão.

Para além do tipo de DAC existem outros parâmetros a terem conta na escolha de um DAC:

1.6. CONVERSÃO DIGITAL-ANALÓGICO

- Resolução ou número de bits (2^N níveis de sinal possíveis) - ilustrado na figura 1.56,
- Ritmo de amostragem (amostras/segundo),
- Tensão de funcionamento,
- Tipo de canal digital (SPI, I2C).

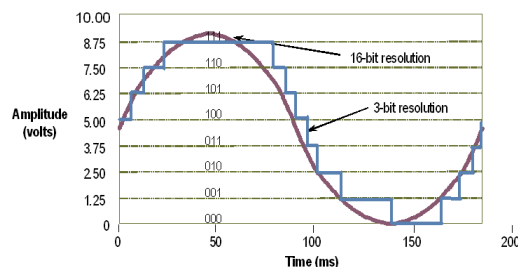


Figura 1.56: Níveis de tensão para diferentes resoluções.

Com a exceção do Arduino Due, nos MCUs dos outros Arduínos não existe um DAC embestado, mas existem saídas que são capazes de gerar sinais PWM. Um sinal PWM é um sinal periódico retangular que tem a largura do seu pulso proporcional ao valor digital. Se o valor digital for máximo, o pulso encontra-se sempre a VCC, se o valor for metade do máximo o sinal fica metade do tempo a VCC e a outra metade a GND. Considerando o valor médio, ou o integral, do sinal de um ciclo completo, e fazendo o sinal retangular variar mais rapidamente do que a carga consegue reagir é possível dar a sensação de controlo da intensidade. A figura ??lustra a forma de onda para sinal PWM com diferentes larguras de pulso.

Num Arduino o sinal PWM é criado num conjunto limitado de pinos de E/S, identificados com o sinal ~. Num Arduino Uno são os pinos digitais 3, 5, 6, 9, 10, e 11. A onda tem uma frequência de 490 Hz nos Arduínos Uno, Mega e Leonardo. A programação do sinal é feita com a função `analogWrite(w)`. Esta função recebe um valor inteiro de 8 bits, com os valores limite em 0 e 255. Esta função num Arduino Due faz uso do DAC interno do MCU. A listagem 1.12 demonstra o uso da função `analogWrite(w)`. Para visualizar o efeito de aumento da intensidade do LED é necessário ligar um LED ao pino D9 e a GND com uma resistência de 470 Ω . O LED volta a ser apagado a cada 5 segundos.

Listagem 1.12: Exemplo de uso de um sinal PWM.

```
int ledPin = 9;
byte val = 0;

void setup() {
  pinMode(ledPin, OUTPUT); // set pin 9 as output
```

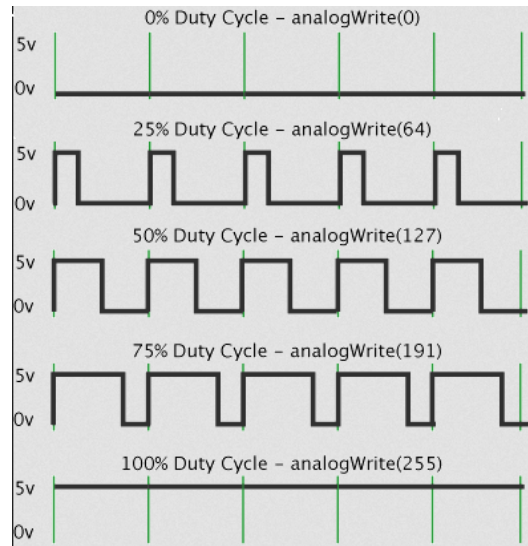


Figura 1.57: Exmeplos de sinais PWM para diferentes larguras de pulso.

```

}
void loop() {
  analogWrite(ledPin, val++); // val will count from 0 to 255 forever
  delay(20);
}

```

Uma alternativa ao uso do PWM e de um DAC é fazer uso de várias saídas digitais e gerar um sinal usando um conjunto de resistências, tal como ilustrado na figura 1.58.

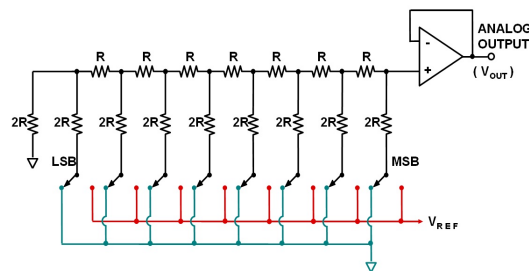


Figura 1.58: DAC com escada de resistências $2R$.

1.7 Processamento Digital de Sinal

O processamento digital de sinal é conhecido como *Digital Signal Processing* (DSP), e é uma disciplina dedicada a métodos e tecnologias implementadas em processadores para discriminar componentes de um sinal digital. Os métodos mais habituais de processamento digital de sinal servem para

filtragem (eliminação de ruído ou separação de bandas de frequência), compressão de dados (exemplo: mp3), reconhecimento de padrões (exemplo: voz e imagem), geração de imagens médicas (exemplo: tomografia computadorizada e ressonância magnética), e extrair informação sobre o terreno de sinais de radar (exemplo: imagens de satélite). O processamento digital de sinal é apresentado em profundidade no livro *The Scientist and Engineer's Guide to Digital Signal Processing*.

No contexto dos sistemas embebidos, onde o poder de processamento é limitado e as aplicações não oferecem muito tempo para processamento, o processamento de sinal mais relevante é a filtragem e a extração de características de sinais, por exemplo detecção de batimento cardíaco, caracterização de movimento e eliminação de ruído.

1.7.1 Filtragem Passa-Baixo

O filtro passa-baixo serve para eliminar as pequenas variações (ruído) nas amostras de um sinal digital e produz outro sinal que tem um valor (mais) estável, mas fiel à medida realizada. Na prática esta filtragem corresponde à média das últimas amostras obtidas. Assumindo que se está a medir a temperatura ambiente, sabe-se que esta grandeza não varia instantaneamente, e o variar entre os valores 24, 25 e 26°C é fácil assumir que a temperatura ambiente é 25°C. A utilização da filtragem passa-baixo é particularmente importante em sistemas ciber-físicos, onde são desencadeadas acções quando são amostrados determinados valores fronteira, para evitar oscilações indesejadas.

Um parâmetro do filtro digital passa-baixo é o número de amostras usadas. Quando maior for o número de amostras mais estável será o valor, mas demorará mais tempo a convergir para o valor real.

A listagem 1.14 exemplifica a implementação de um filtro passa-baixo, que realiza a média das amostras recolhidas.

Listagem 1.13: Sketch para realizar filtragem passa-baixo.

```
/* low-pass filter demo
 * reads A0 and sends value to terminal
 */

#define NUMSAMPLES    16
#define SAMPLEPERIOD  1
uint16_t new_sample, samples[NUMSAMPLES];

enum state_type { IDLE, SAMPLE, FILTER, DISPLAY } state;

void setup() {
  Serial.begin(115200);
  while(!Serial);
  Serial.println("!.:Low-pass Demo:.");
  state = IDLE;
}
```

```
void loop() {
  int i;
  long accum, avg;
  long now, lastSampleT;
  switch(state) {
    case IDLE: if(now - lastSampleT > SAMPLEPERIOD) {
    }
    break;
    case SAMPLE: break;
    new_sample = analogRead(A0);
    case FILTER:
    accum = 0;
    // shift samples
    for(i = 0; i < NUMSAMPLES - 1; i++) {
      samples[i + 1] = samples[i]
      accum += samples[i + 1];
    }
    // get new sample
    accum += samples[0];
    // compute average
    avg = accum / NUMSAMPLES;
    state = DISPLAY;
    break;
    case DISPLAY: break;
    Serial.println(avg);
    state = IDLE;
  }
}
```

1.7.2 Filtragem Passa-Alto

O filtro passa-baixo serve para eliminar as pequenas variações (ruído) nas amostras de um sinal digital e produz outro sinal que tem um valor (mais) estável, mas fiel à medida realizada. Na prática esta filtragem corresponde à média das últimas amostras obtidas. Assumindo que se está a medir a temperatura ambiente, sabe-se que esta grandeza não varia instantâneamente, e o variar entre os valores 24, 25 e 26°C é fácil assumir que a temperatura ambiente é 25°C. A utilização da filtragem passa-baixo é particularmente importante em sistemas ciber-físicos, onde são desencadeadas acções quando são amostrados determinados valores fronteira, para evitar oscilações indesejadas.

Um parâmetro do filtro digital passa-baixo é o número de amostras usadas. Quando maior for o número de amostras mais estável será o valor, mas demorará mais tempo a convergir para o valor real.

A listagem 1.14 exemplifica a implementação de um filtro passa-baixo, que realiza a média das amostras recolhidas.

Listagem 1.14: Sketch para realizar filtragem passa-baixo.

```
/* low-pass filter demo
 * reads A0 and sends value to terminal
 */

#define NUMSAMPLES 16
#define SAMPLEPERIOD 1
uint16_t new_sample, samples[NUMSAMPLES];
```



```

enum state-type { IDLE, SAMPLE, FILTER, DISPLAY } state;

void setup() {
  Serial.begin(115200);
  while(!Serial);
  Serial.println("!.:Low-pass Demo:.");
  state = IDLE;
}

void loop() {
  int i;
  long accum, avg;
  long now, lastSampleT;
  switch(state) {
    case IDLE: if(now - lastSampleT > SAMPLE.PERIOD) {
      }
      break;
    case SAMPLE: break;
      new_sample = analogRead(A0);
    case FILTER:
      accum = 0;
      // shift samples
      for(i = 0; i < NUMSAMPLES - 1; i++) {
        samples[i + 1] = samples[i];
        accum += samples[i + 1];
      }
      // get new sample
      accum += samples[0];
      // compute average
      avg = accum / NUMSAMPLES;
      state = DISPLAY;
      break;
    case DISPLAY: break;
      Serial.println(avg);
      state = IDLE;
  }
}

```

1.8 Comunicações

A transmissão de informação à distância via telégrafo, onde as mensagens eram codificadas em Código Morse, foi inventada em 1844. No Morse existem apenas 2 símbolos para codificar 26 caracteres (letras A-Z e os dígitos 0-9): sinal de duração curta (ponto) e duração longa, e que correspondiam à perfuração feita numa tira de papel, de um ponto ou um traço, respectivamente. Cada símbolo é composto por uma sequência de pontos e traços, sendo os caracteres mais frequentes, por exemplo vogais, codificados com menos símbolos. Com os avanços tecnológicos o Morse foi substituído por outras tecnologias que permitiam a transmissão de voz e dados. Atualmente, os sistemas de comunicações são caracterizados por: tipo de ligação (com ou sem fios), velocidade de transmissão de informação em bits por segundo (bps), codificação da informação ou protocolo, banda de frequências (Hz) e alcance (m).

No âmbito dos sistemas embebidos, ciber-físicos e IoT, existe interesse em

A	• —	U	• • —
B	— • • •	V	• • • —
C	— • — •	W	• — —
D	— • •	X	— • • —
E	•	Y	— • — —
F	• • — •	Z	— — • •
G	— — •		
H	• • • •		
I	• •		
J	• — — —		
K	— • —	1	• — — — —
L	• — • •	2	• • — — —
M	— —	3	• • • — —
N	— •	4	• • • • —
O	— — —	5	• • • • •
P	• — — •	6	— • • • •
Q	— — • —	7	— — • • •
R	• — •	8	— — — • •
S	• • •	9	— — — — •
T	—	0	— — — — —

Figura 1.59: Tabela de Código Morse.

estabelecer ligações para transferências de dados: ponto-a-ponto (entre dois sistemas), ponto-multi ponto (entre vários sistemas), para aplicações onde é necessário difundir comandos a várias unidades remotas (*broadcast*), agregar informações recolhidas por vários sensores, sistemas distribuídos por uma infraestrutura, ou dar comandos para um conjunto de sistemas em simultâneo (*multicast*). Alguns sistemas possuem ainda a capacidade de ligação à Internet.

A comunicação entre sistemas em rede é estabelecida em diferentes patamares ou camadas de protocolos, para resolver o estabelecimento de uma ligação criando caminhos entre vários elementos da rede entre dois elementos. Para que os diferentes sistemas fossem capazes de comunicar foi normalizado o Modelo OSI, do inglês *Open Systems Interconnection*. O Modelo OSI foi proposto nos anos 80, e é composto por 7 camadas, sendo na prática as 3 últimas tratadas atualmente pela aplicação:

7. Aplicação - trata do acesso aos recursos;
6. Apresentação - codificação, cifra e compressão de dados;
5. Sessão - gestão de sessões;
4. Transporte - trata da transmissão de segmentos de dados numa rede;
3. Rede - encarregue do endereçamento, encaminhamento e controlo de tráfego numa rede;

2. Ligação - transmissão fiável de tramas entre dois pontos da rede;
1. Física - transmissão e recepção de dados (bits) num canal físico.

Em termos práticos atualmente as camadas 5 a 7 encontram-se agregadas de tal forma que são tratadas como uma só. Recentemente foi proposto um modelo adequado aos sistemas IoT, e define as seguintes 7 camadas:

7. Colaboração e Processos - processos que envolvem pessoas e empresas;
6. Aplicação - Relatórios, análise, e controlo;
5. Abstração de dados - Agregação e acesso a dados;
4. Acumulação de dados - Armazenamento;
3. Computação na extremidades (*fog/edge*) - pré-processamento e transformação;
2. Ligação - Comunicação;
1. Física - Equipamento e componentes do sistema IoT, p.ex. sensores.

No modelo IoT as sete camadas do Modelo OSI foram substituídas por apenas duas. Nesta secção são apresentadas as tecnologias mais habituais em comunicação com sistemas embebidos, indicando como são caracterizadas e qual o tipo de aplicações:

- *Radio Frequency IDentification* (RFID) e Near Field Communication (NFC) - utilizado para ler dispositivos (*tags*) passivos, ou de muito baixo consumo de energia, normalmente para identificação, realização de pagamentos. Habitualmente, este tipo de sistemas transfere pouca informação, apenas num sentido e com baixa largura de banda.
- LORA e Sigfox - utilizados para interligação de sistemas, pode funcionar em ligações ponto-a-ponto ou em redes como o Bluetooth (BT). Suporta comunicação bidirecional em distâncias na ordem dos km. Necessitam de uma Gateway para aceder à Internet.
- Ethernet - comunicação com fios a redes de computadores, privadas ou públicas. Permite acesso à Internet com alta velocidade (Mbps).
- WiFi - comunicação com fios a rede de computadores, privada ou pública. Permite acesso à Internet com alta velocidade (Mbps). Muito utilizado em sistemas IoT.
- PSTN - rede telefónica usando um par de fios. Também usada para transmissão de dados a baixa velocidade (kbps).

CAPÍTULO 1. INTERFACES DE ENTRADA/SAÍDA

- BT - utilizado para comunicações de curto alcance e baixo débito entre dois terminais (ponto-a-ponto) ou em redes pessoais (kbps).

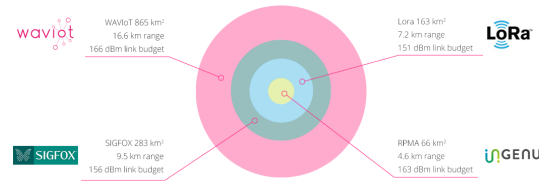


Figura 1.60: legenda.

Technology	WAVIOT NB-I	Link Labs LoRa	Sigfox LRS	Impress RPMA	NeuS Weightless	LTE-M
Frequency	868.8 MHz	868.8 MHz	868.8 MHz	2.4 GHz	868.8 MHz	1.8 - 2.7GHz
Max urban range with 99% reliability, m.	16 600	7 200	9 500	4 600	4 100	640
Maximum link budget, dBm	166	151	156	163	147	147
Node bandwidth	100 Hz	125 kHz	100 Hz	1 MHz	200 Hz	192 kHz
Spectrum efficiency	High	Very Low	Low	Medium	Low	High
Gateway mode	Full Duplex	Half Duplex	Half Duplex	Half Duplex	Uplink Only	Full Duplex
Nodes per gateway	1 350 000	40 000	50 000	500 000+	50 000	20 000
Scalability	High	Very Low	Low	High	Low	High
Simultaneous demodulation capacity	120 000	8	25	4 000	25	64
Security	XTEA 256 bit	32 bit	16 bit	AES 128 bit	No	128 - 256 bit
Minimum node cost, USD	\$1.99	\$29.00	\$1.99	\$5.00	\$19.00	\$5.00
Battery lifetime	10 years	10 years	10 years	10 years	10 years	5 years
Sector antennas	Yes	No	No	No	No	Yes
Commercial Deployment	100+	100+	100+	30	10	N/A until 2020
First commercial project	2011	2014	2010	2010	2013	2020

Figura 1.61: legenda.

Estas tecnologias permitem a realização de sistemas tais como:

- sistemas inteligentes para monitorização de consumo de água, eletricidade, gás, esgotos e resíduos.
- sistemas de acompanhamento de saúde de pessoas idosas ou incapacitadas, condições climáticas, agricultura, criação de gado, espécies animais em perigo, veículos privados e meios de transporte públicos.
- sistemas de alarme inteligentes de desastres: inundações, incêndios, terremotos.

1.8.1 RFID

RFID é uma tecnologia para identificação unívoca de objetos baseada em etiquetas (*tags*) que comunicam através de sinais rádio. A tecnologia RFID está descrita na norma ISO 14443 (*Proximity Contactless Cards*). Os sistemas RFID identificam objetos, através da leitura de um valor das *tags*. Um sistema RFID é composto pelo sistema de leitura, uma antena e o elemento a identificar. Um sistema RFID funciona de forma simples, o leitor envia um sinal para a antena a interrogar as tags na proximidade, e estas respondem

com o seu identificador único. A figura 1.62 ilustra um sistema RFID com os três elementos. Os sistemas RFID encontram-se aplicados em bilhetes eletrónicos nos transportes, acompanhamento de cargas, controlo de acessos e cartões bancários.

Existem 3 tipos de sistemas RFID, e são distintos pela gamas de frequências de funcionamento:

- Baixa frequência - *Low Frequency* (LF), comunica em frequências rádio entre 125 e 134kHz, alcance na ordem dos metros, e apenas lê as *tags*;
- Alta frequência - *High Frequency* (HF), comunica nos 13,56MHz, alcance na ordem dos centímetros, permite ler e escrever nas *tags*;
- Muito alta frequência - *Ultra High Frequency* (UHF), funciona entre os 856MHz e os 960MHz, alcance na ordem dos centímetros, permite ler e escrever nas *tags*.

As *tags* pode ser do tipo ativo, usam alimentação própria, ou passivo, usam a energia radiada pela antena para fazer o seu circuito funcionar.

NFC é uma tecnologia para comunicação em proximidade, correspondendo a um sub-conjunto de sistemas RFID que operam em alta frequência (HF) na banda dos 13,56 MHz. A tecnologia NFC permite a troca de informação de forma segura. Um dispositivo NFC tanto pode identificar outro dispositivo como ser identificado. A tecnologia NFC é aplicada na leitura de cartões e identificadores passivos (sem alimentação), aproveitando a energia radiada pelo emissor e comunicando por radio-frequência (RF). O MIFARE é outra variante de identificadores RFID HF e é proprietária da NXP Semiconductors e é usada para controlo de acessos, porta-moedas electrónico, ou passes de transporte (*Smart Mobility* - bicicletas, trotinetes, parques de estacionamento e transportes públicos) e eventos (jogos e espetáculos). Ambas as tecnologias são usadas para troca de pouca informação, tendo taxas de transferência máximas na ordem das centenas de kbit/s.



Figura 1.62: Diferentes tipos de tags RFID/NFC.

Quando começou a ser comercializado, este tipo de tecnologias vislumbrava uma melhoria dos serviços de venda ao público, ao facilitar a identificação quase instantânea de produtos sem ter de os remover do saco ou do

carrinho de compras. Contudo, e por muito baixo que fosse o custo de fabrico de etiquetas RFID/NFC elas não conseguem competir com a tecnologia de código de barras que é gratuita para o fabricante. Como o código de barras faz parte do rótulo da impressão da embalagem do produto, já está paga à partida.

1.8.2 LoRa

LoRa (*Long Range*) é uma norma para comunicações de longo alcance, e especifica os detalhes físicos e de rede para comunicações sem fios e baixo consumo de energia, cobrindo distâncias por volta dos 10 km. O LoRa não obriga ao licenciamento para operar, estando disponível nas bandas de frequência radio de 433 MHz, 868 MHz na Europa e 915 MHz nos EUA. A forma mais simples de integrar a tecnologia LoRa num sistema embebido é através de módulos que têm controladores com a capacidade de tratar de todos os protocolos até à cama de ligação. A figura 1.63 mostra um módulo LoRa.

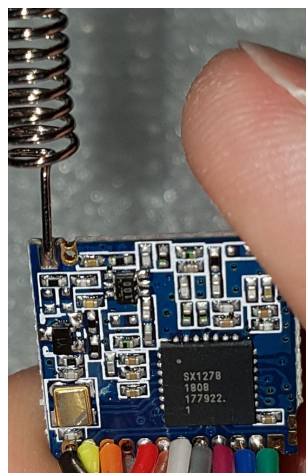


Figura 1.63: Módulo de comunicação LORA.

1.8.3 Sigfox

O Sigfox é uma norma de redes sem fios para sistemas com pouco consumo energético, e que necessitam de estar a transmitir pequenas quantidades de dados (telemetria), e concorrente ao LoRa. O Sigfox funciona na banda dos 868 MHz na Europa. A rede é suportada por um operador que vende o serviço de encaminhar os dados para o destinatário. Tal como no LoRa, os módulos de comunicação, semelhantes ao que se encontra na figura 1.64, são constituídos por um circuito integrado que faz interface com o sistema embebido e trata da gestão de toda interface rádio. O módulo possui alguns

circuitos integrados reguladores de tensão, um cristal ou oscilador para fornecer a base de tempo ao controlador, e também componentes passivos usados no circuito da antena.



Figura 1.64: Módulo de comunicação Sigfox para Arduino.

1.8.4 Ethernet

Ethernet é uma tecnologia que permite a interligação de computadores, e outros equipamentos de comunicação, em redes locais *Local Area Networks* (LAN). Este tipo de redes é bastante usado em redes domésticas e industriais, e foi em parte responsável pela rápida evolução da Internet, por permitir avançar na velocidade de transmissão ao longo do tempo, ao mesmo tempo que permitiu compatibilidade com a maioria dos sistemas disponíveis. Os equipamentos ligados a redes Ethernet organizam a informação a transmitir em segmentos chamados tramas. As tramas têm um formato conhecido, que inclui a identificação do emissor e do destinatário. As tramas Ethernet encapsulam outros protocolos usados para a identificação dos equipamentos (IPv4 ou IPv6), estabelecimento de sessões de comunicação de dados (TCP e UDP), e protocolos de aplicação: *Domain Name System* (DNS), *Dynamic Host Configuration Protocol* (DHCP), *Hypertext Transfer Protocol* (HTTP), *Message Queuing Telemetry Transport* (MQTT), *Extensible Messaging and Presence Protocol* (XMPP).

A nível da tecnologia Ethernet ela define a comunicação entre diferentes equipamento sobre diferentes meios ou tipos de ligações. As velocidades de comunicação 10, 100 e 1000 Mbps, são definidas nas normas 10BASE-T, 100BASE-TX, e 1000BASE-T, respetivamente. Estas normas fazem uso de cabos com 4 pares de fios entrelaçados para suportar a sua ligação.

CAPÍTULO 1. INTERFACES DE ENTRADA/SAÍDA

Para facilitar a interligação, mantendo a compatibilidade, as interface Ethernet suportam auto-negociação da norma a usar na comunicação. Isto permite adaptação dos parâmetros de transmissão às condições do elemento mais lento.

Existem sistemas compatíveis com a especificação *Power Over Ethernet* (PoE) para fornecer alimentação aos equipamentos terminais de rede através da cablagem Ethernet. Isto permite que os equipamento possam funcionar sem ser necessário uma ligação à rede eléctrica, por exemplo telefones IP, câmaras IP, equipamento de controlo industrial, e pontos de acesso para fornecer rede sem fios WiFi. Pelos mesmos motivos, esta funcionalidade é bastante atrativa para sistemas IoT, no entanto a sua implementação necessita que o equipamento de rede esteja preparado para fornecer energia e que o sistema IoT tenha um controlador capaz de negociar o fornecimento. Diferentes normas PoE limitam a transferência máxima de energia para diferentes valores, variando entre 12 e 71 W.



Figura 1.65: Cabo Ethernet e Comutador (Switch) Ethernet.

Numa rede Ethernet, cada equipamento encontra-se ligado a uma porta de um comutador (*Switch*) por um cabo com oito condutores. Um *Switch* é responsável por: a) agregar os dados de todas as portas e envia-los para o próximo elemento de rede; e b) ao receber dados para um determinado equipamento, encaminhar esses dados para a porta correta. Desta forma, as redes Ethernet permitem a transferência comutada de tramas, ou seja, não existem colisões uma vez que o meio não é partilhado. Um Switch não tem noção da topologia da rede. Outro elemento de rede é o encaminhador (*Router*) que é responsável por identificar a topologia da rede e determinar qual o melhor caminho na rede para o envio dos dados. Ele faz uso de um protocolo específico para esse efeito, que é enviado pela rede Ethernet.

De notar que as fichas dos equipamentos terminais (computadores, impressoras) são “simétricas” em relação às fichas do router e do switch. A ligação entre o computador e o switch é feita com um cabo direto ao passo que a ligação entre dois equipamentos idênticos, como por exemplo um sistema embebido e um computador é feita com um cabo cruzado, tal como

ilustrado na figura 1.66. Para ligações com menor débito (10/100 Base-T) podem ser usados cabos com menos condutores, sendo necessário apenas ligar os fios 1,2,3, e 6. Para realizar a derivação de uma tomada para dois equipamentos terminais sem necessidade de equipamento extra basta ligar os pinos 1,2,3,6 entre a ficha de origem e o cabo de uma derivação, e os pinos 5,4,7,8 aos pinos 1,2,3,6 do cabo da segunda derivação.

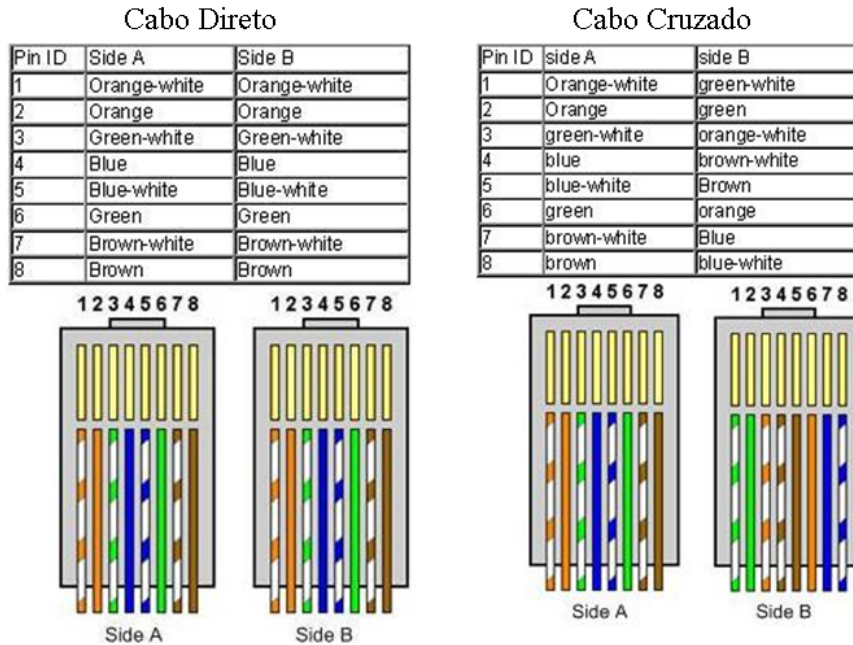


Figura 1.66: Ligação de cabos Ethernet directo (à esquerda) e cruzado (à direita).

1.8.5 WiFi

A tecnologia WiFi define a ligação sem fios (via rádio em 2,4 e 5 GHz) para comunicação dentro de uma rede local, e com capacidade de acesso à Internet. Esta tecnologia está definida na norma IEEE 802.11. Com a evolução da tecnologia e das técnicas de processamento digital de sinal, foram surgindo diferentes versões da norma com suporte para velocidades de transmissão maiores, ao mesmo tempo que permitiam compatibilidade com versões anteriores da norma.

Com a massificação dos dispositivos IoT, o WiFi foi um componente que passou a integrar os sistemas embebidos. As redes sem fios WiFi fornecem uma taxa de transmissão elevada, tendo em conta a capacidade de processamento de um sistema embebido para IoT. A elevada taxa de transmissão faz com que não seja adequado a aplicações com restrições a nível energético, sendo mais adequado para sistemas permanentemente ligados à rede elétrica.

A figura 1.67 mostra um router, com as interfaces Ethernet exposta e as antenas para comunicação via RF.

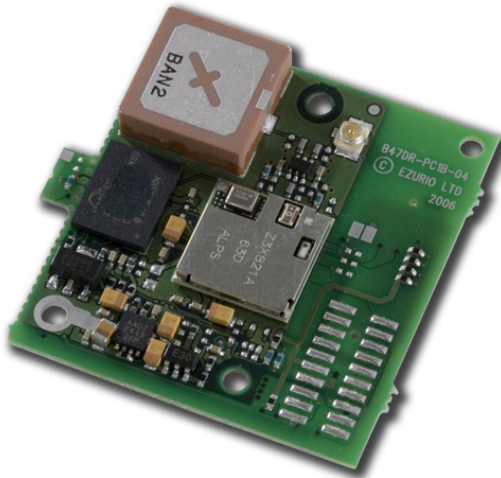


Figura 1.67: Router WiFi.

1.8.6 Rede Telefónica Pública

A rede telefónica pública é das redes de comunicações mais antigas, e que ainda continua em funcionamento. A facilidade de ligação com um par de fios e a simplicidade dos dispositivos permitiu a massificação inicialmente para comunicações de voz analógica e mais tarde para transmissão de documentos (fax) e posteriormente transferências de dados. A rede telefónica inicialmente analógica, que usava aparelhos como o da figura 1.68, com um funcionamento manual, foi tendo atualizações devido aos avanços da tecnologia. Estes aparelhos recebiam o sinal do microfone e modulavam o sinal da linha telefónica partilhada pelos dois equipamentos, ao mesmo tempo que amplificavam o sinal da linha para o auscultador. A marcação do número de telefone era baseado em pulsos, equivalentes ao curto-circuitar os fios da linha telefónica, gerado por um disco que era rodado pelo utilizador com os furos correspondentes ao número a marcar. As redes telefónicas modernas *Public Switched Telephone Network* (PSTN) são suportadas por redes digitais via fibra-ótica e redes sem fios via Rádio Frequência (RF) com várias gerações de protocolos (GSM/GPRS/UMTS), para comunicações a longa distância e móveis, respetivamente.

Antes da existência de Internet móvel e por fibra ótica, a Internet era fornecida através de MODuladores/DEsMODuladores (MODEMs) ligados ao par de fios da rede fixa. Os MODEMs transformam os 0s e 1s a transmitir em tons que são reconhecidos no receptor, que reconstrói a informação digital.



Figura 1.68: Telefone analógico.

Por este motivo os MODEMS funcionam na banda de frequências dos sinais de voz, para a qual a rede telefónica fixa estava preparada. Neste caso, o utilizador marcava o número de telefone do fornecedor de serviço de Internet e iniciava um protocolo para negociação do acesso, onde passava também as credenciais de acesso à Internet. Posteriormente, os dados passaram a ocupar bandas do espectro mais altas, fora da gama do áudio, permitindo ritmos de transmissão superiores e a existência de chamadas de voz em simultâneo com acesso à Internet.

Apesar de estar ultrapassada em tecnologia, uma das grandes vantagens da rede telefónica fixa é a facilidade de utilização e a elevada disponibilidade e fiabilidade, por esse motivo é um canal de comunicação que serve para comunicar pedidos de emergência (112).

A forma de estabelecer uma chamada com um telefone fixo é a seguinte:

1. “agarrar” a linha e esperar pelo tom de convite à marcação. Se o tom de 425 Hz (na Europa) não for audível continuamente significa que existem problemas com a linha;
2. marcar o número do destinatário;
3. se o equipamento no destinatário estiver disponível, ele vai tocar e o chamador ouve a confirmação de que o telefone está a tocar através de um tom 425 Hz durante 1 segundo com intervalos de 2 segundos;
4. se o equipamento no destinatário estiver ocupado com outra chamada ou desligado, o chamador ouve o sinal de chamada interrompida ou ocupada 400 Hz;

No âmbito de sistemas embebidos um exemplo de uma aplicação que faz uso da rede telefónica são os elevadores. Por norma, todos os elevadores devem ter uma linha telefónica para realizar as chamadas de emergência.

A chamada de emergência é iniciada após o utilizador pressionar do botão de alarme durante alguns segundos. Neste caso, o circuito do elevador marca automaticamente o número da central de emergência da empresa responsável pela manutenção do elevador. Um MCU no elevador é responsável por detetar o botão premido e marcar o número através de um Modulator-Desmodulador (MODEM).

Para além do estabelecimento de chamadas de emergência, é possível usar uma linha telefónica em períodos em que os equipamentos e sistemas não estão a ser usados, para enviar estatísticas de utilização, estado dos componentes críticos ou anomalias para a central. Este mecanismo também potencia o desligar do equipamento remotamente caso sejam identificadas anomalias ou caso os equipamentos não tenham manutenção durante longos períodos de tempo para salvaguardar o seu bom funcionamento e evitar acidentes graves.

Os MODEMs tipicamente oferecem uma interface série assíncrona (UART) para poderem ser ligados a um sistema embebido, bastando apenas três fios para o fazer: TX, RX e GND. O canal série serve para envio de comandos de controlo e transferência de dados. A forma de programar um MODEM é através de comandos AT.

A listagem 1.15 exemplifica uma sequência de comandos e respostas AT trocados com um MODEM:

Listagem 1.15: Mensagens de configuração de um MODEM BT

```
>AT
OK
>ATZ
OK
>ATD960123456
```

Os MODEMs para redes sem fios (GSM/UMTS) também usam o protocolo AT e possuem extensões para funcionalidades adicionais relacionadas com a tecnologia da rede sem fios, nomeadamente associação à rede do operador, envio e receção de mensagens de texto, e ligações de dados. As extensões são conhecidas como comandos AT+, apresentados na listagem 1.16.

Listagem 1.16: Mensagens de configuração de um MODEM BT

```
>AT+GMI=?
SIMCOMLtd
OK
```

1.8.7 Bluetooth

O protocolo BT é usado para comunicações de curto alcance (metros), sendo amplamente utilizado em computadores, telemóveis e equipamento áudio como auscultadores, colunas e aparelhagens de som. A grande vantagem deste protocolo é permitir interligação direta de diversos equipamentos sem serem necessários cabos nem adaptadores. A maior desvantagem é a reduzida

velocidade de comunicação. Existe uma versão do protocolo BT, designada por Bluetooth Low Energy (BLE) que é destinado a sistemas de baixo consumo. Para haver troca de informação entre equipamentos BT, deve haver concordância mútua através de um processo de emparelhamento, que consiste na avaliação de uma chave comum.

Para criar um sistema embebido com conectividade BT podem-se usar um MCU já inclui um controlador dedicado restando adicionar a antena (p.ex. Texas Instruments, Nordic). A alternativa para MCUs que não possuem o controlador BT é usar um módulo dedicado à comunicação BT e que já inclui uma antena impressa, incluída no desenho da Placa de Circuito Impresso - *Printed Circuit Board* (PCB). Dada a complexidade da realização do circuito contendo o controlador e o circuito rádio, em sistemas embebidos pode ser preferível optar-se por usar um módulo BT que é responsável pela gestão de toda a camada física e de rede, simplificando o resto do sistema e do programa de controlo. A figura 1.69 mostra um módulo BT que é constituído por uma antena BT do lado esquerdo, controlador dedicado e memória ao centro, e ligação aos pinos de alimentação e comunicação série à direita..



Figura 1.69: Módulo de comunicações sem fios Bluetooth.

Este tipo de módulos oferece uma interface série assíncrona (UART) e suporta uma extensão dos comandos AT destinada a BT. Esta interface é usada para controlo e transferência de dados. A listagem 1.17 exemplifica os comandos para interação com um módulo Bluetooth:

Listagem 1.17: Mensagens de configuração de um MODEM BT

```
>AT
OK
>AT+NAME
+NAME:HC-06
OK
>AT+VERSION
+VERSION:2.1-20120904
OK
>AT+INIT
OK
```

1.9 Interface Mecânica

Em muitas aplicações ciber-físicas, especialmente no domínio da robótica e fabricação industrial, onde existe a necessidade de haver ligação a um sistema

acoplado a um veio em rotação, por exemplo braço mecânico, volante, ou eixo das rodas, é impossível manter uma ligação física feita por fios pois em pouco tempo acabariam por torcer e quebrar. Uma forma de resolver este problema consiste na introdução de um *slip-ring* no veio de rotação. O *slip-ring* é construído com base num pequeno cilindro com anéis espaçados e ligados aos fios pelo lado de dentro do cilindro. Outro cilindro maior encaixa por fora deste, e no local coincidente com os anéis do cilindro mais pequeno tem um conjunto de patilhas metálicas em contacto com os anéis, e ligadas a fios na sua extremidade que fazem contacto com os anéis do cilindro mais pequeno. Esta solução permite que as duas peças rodem em qualquer direção indefinidamente.

O *slip-ring* na figura 1.70, à esquerda, permite ligações de seis fios entre os extremos, e suporta velocidades de rotação até 300 rpm. O detalhe do cilindro interior com os anéis é visível na mesma figura à esquerda.

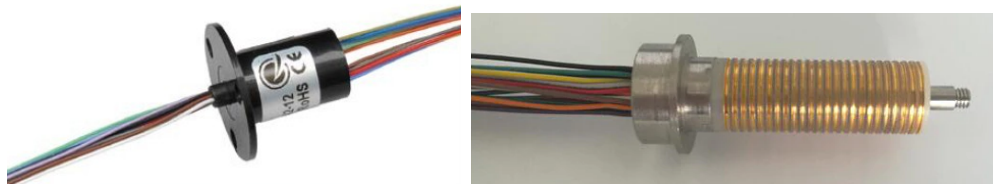


Figura 1.70: Slip-ring com suporte para 6 canais (esquerda), e detalhe das ligações internas de um slip-ring de 24 canais.

1.10 Limitações Físicas

As grandezas habitualmente envolvidas num sistema embebido, tal como enunciado no capítulo 2, estão relacionadas com níveis limite e máximos de tensões de alimentação, nível de sinal de ruído, e frequência e temperatura máxima de funcionamento.

Nos sistemas embebidos existem diversos tipos de sinais (digitais e analógicos) e com amplitudes relativas diferentes, pelo que é necessário ter em atenção alguns aspectos para dimensionar um sistema que funcione com a qualidade pretendida:

- Realizar o circuito analógico o mais próximo do sensor, e ser o mais compacto possível para reduzir os pontos suscetíveis de captarem interferências. Se o sensor ficar afastado do circuito condicionador, usar um par de fios entrelaçados, ou cabos com blindagem, para reduzir o efeito de interferência;
- Usar filtros para reduzir o ruído nas alimentações: colocar uma bobina

10 μH em série com o pino de alimentação (VCC) e um condensador 100 nF entre VCC e GND;

- Ligar as massas analógicas (AGND) e digitais (DGND) num único ponto a GND da alimentação;
- Usar planos de massa para circuitos analógicos. Se possível fechar o circuito numa caixa metálica (gaiola de Faraday);
- Observar as características dos componentes e do circuito, nomeadamente:
 - Impedância - se a impedância não for respeitada o sinal vai sofrer (muita) atenuação;
 - Corrente máxima fornecida - um componente que consuma 100 mA para funcionar ligado a um pino de E/S do MCU apenas fornece 20 mA não vai funcionar corretamente e pode danificar o pino de E/S;
 - Alimentação do circuito “de qualidade” para que o ruído proveniente da fonte não afecte a leitura dos sensores. Recomenda-se usar circuitos de filtragem e isoladores para as fontes de alimentação. Em casos extremos usar um banco de baterias que carregam enquanto o circuito não está em funcionamento;
 - Observar quais as temperaturas de funcionamento dos componentes do circuito. Existem componentes que mudam de comportamento com a variação da temperatura, e a sua variação pode ter de ser compensada;
 - Para ambientes agressivos/corrosivos ou suscetíveis de interferir com o circuito eletrónico, devem-se isolar os componentes/circuitos do ambiente, por exemplo usando caixas estanques e ou cobrir o circuito com resina (por exemplo epoxy).
 - Respeitar a máxima frequência de funcionamento - alguns circuitos e componentes funcionam de forma diferente em altas frequências (normalmente acima de MHz). Deve-se usar técnicas de desenho de placas de circuitos para essas frequências e usar cabos coaxiais para essas ligações, bem como observar as impedâncias de todos os componentes.

2

Sensores

Os sensores são os componentes que permitem traduzir grandezas físicas em elétricas, nomeadamente posição, movimento e aceleração, campos elétricos e magnéticos, níveis de líquidos, humidade e caudais, temperatura, presença de compostos químicos, ruído e vibrações. A figura 2.1 ilustra alguns tipos de sensores possíveis de ligar a um Arduino. Os sensores são conhecidos como transdutores, pois eles traduzem a variação de uma grandeza física numa elétrica. Na realização de um sistema embebido capaz de realizar medidas de uma grandeza é importante ter uma noção dos diferentes tipos de sensores e as suas características para serem aplicados de acordo com os requisitos das aplicações.

Os sensores têm um papel fundamental no âmbito dos sistemas ciberfísicos e Internet das Coisas - *Internet of Things* (IoT) pois a qualidade da sua resposta, ou funcionamento, depende deles. Sem um sensor adequado não existe processamento.

Além das grandezas físicas que medem, os sensores são também caracterizados por:

- ativos (geradores de um sinal elétrico) ou passivos (moduladores de um sinal externo);
- analógicos (variação de medida contínua no tempo) ou digitais (detecção de presença ou ausência, quantidade);
- medida por comparação (relação com um ponto de referência ou equilíbrio) ou deflexão (aferição de uma quantidade);
- sensibilidade - variação física mínima detetada pelo sensor;

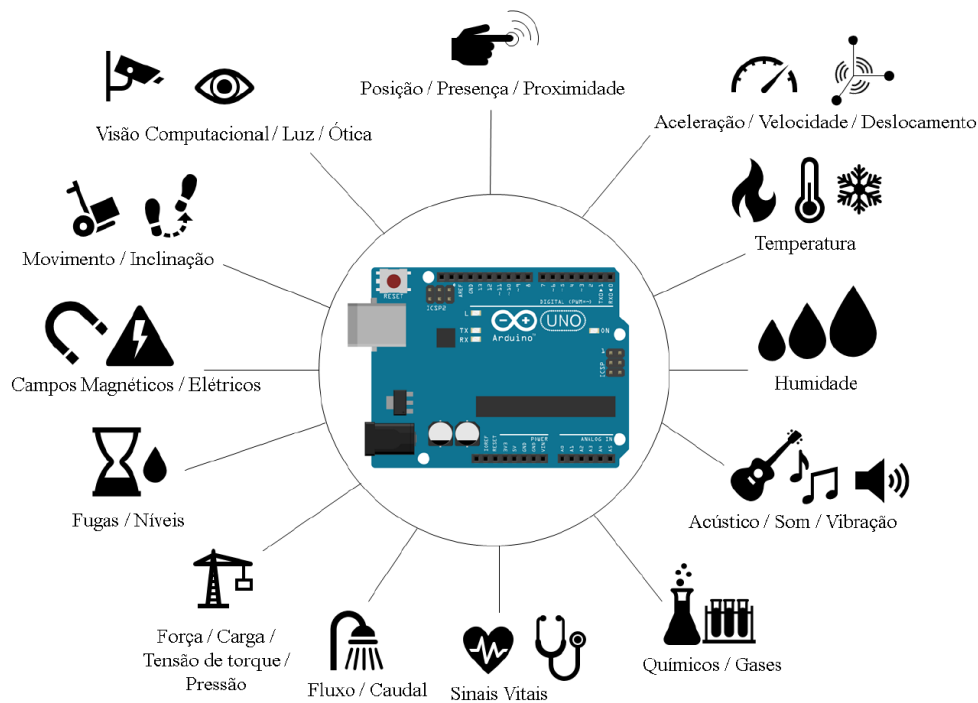


Figura 2.1: Exemplos de algumas grandezas físicas quantificáveis por sensores.

- precisão - diferença entre os valores medidos para a mesma quantidade medida;
- exatidão - diferença entre o valor medido e o valor real;
- resolução - variação mínima no sistema medido para que se observe uma variação na medida;
- tempo de resposta - o tempo necessário pelo sensor para realizar uma medida.

Muitos sensores são sensíveis a perturbações externas, tais como proximidade a sinais agressores (fortes) que afetam o sistema de medida. A sensibilidade ao ruído depende das características elétricas do sensor e do princípio utilizado para medir os sinais de interesse.

O erro associado a uma medida é definido como o desvio em relação ao valor real, e pode ser medido de duas formas:

- erro absoluto = valor medido - valor real (na unidade de medida)
- erro relativo = $100 \cdot (\text{valor medido} - \text{valor real}) / \text{valor real}$ (em %)

Os erros podem ser sistemáticos ou aleatórios. Os erros sistemáticos são aqueles que existem sempre para qualquer valor medido (degradam a exatidão). Conseguem-se identificar erros sistemáticos realizando medidas da mesma quantidade com outros aparelhos de medida. Na maioria das vezes podem ser quantificados por uma aproximação a um valor constante, facilmente compensado no circuito ou no programa. Os erros aleatórios são os que não se mantêm constantes ao longo de várias medidas, e são normalmente associados a ruído (degradam a precisão). Tendo em conta os diferentes tipos de erros, consegue-se ter “uma ideia” do intervalo de valores a que o valor real pertence. Quanto maior o intervalo, maior é a gama de valores possíveis e por consequência é maior a de incerteza da leitura, não oferecendo muita informação. Se o intervalo entre os valores máximo e mínimo for menor, significa que o valor real está num conjunto de valores mais restrito, sendo fácil aferir o valor real.

De uma forma geral as medidas realizadas com os sensores são definidas pela:

- fidelidade: capacidade de produzir o mesmo resultado continuamente, sob as mesmas condições;
- repetibilidade: capacidade de produzir resultados iguais usando as mesmas condições, os mesmos métodos, e equipamento de medida;
- reprodutibilidade: capacidade de produzir resultados próximos usando métodos, equipamento de medida e condições diferentes.

2.1 Sensores Resistivos

Os sensores resistivos são componentes passivos que traduzem uma grandeza física por uma variação proporcional de resistência elétrica. Normalmente eles são baseados em propriedades físicas de materiais, e diferentes grandezas vão ter relações com a resistência diferentes.

Em termos práticos, o sensor resistivo necessita de ser percorrido por uma corrente elétrica para causar uma variação de tensão aos seus terminais e poder ser amostrada por um sistema embebido através de uma das suas entradas analógicas.

A listagem 2.17 mostra o Sketch para demonstração da classe `basicSensor`, usada para realizar medidas de um sensor resistivo.

Listagem 2.1: Listagem da classe `basicSensor`

```
// basic on-off sensor

class basicSensor {
  private:
    byte pin, pinMode;
    byte curState, prevState;
  public:
```

```
    basicSensor(byte pin = 2, bool pinMode = 0) {
        this->pin = pin;
        this->pinMode = pinMode;
    }
    void init() {
        prevState = 0;
        pinMode(pin, pinMode);
    }
    void calibrate() {
        do {
            prevState = curState;
            curState = digitalRead(pin);
            delay(10);
        } while(prevState != curState); // wait for input pin to settle
    }
    bool getState() {
        curState = digitalRead(pin);
        return curState;
    }
    bool check() {
        prevState = curState;
        curState = digitalRead(pin);
        if(prevState != curState) { return 1; } // new pin read differs
        else { return 0; }
    }
}
movSensor(2);

void setup() {
    Serial.begin(115200);
    while(!Serial); // wait for serial port
    movSensor.init();
    movSensor.calibrate();
}

bool a;
void loop() {
    a = movSensor.check();
    if(a) {
        Serial.println("Sensor Active!!");
    }
}
```

2.2 Sensores Capacitivos

Os sensores capacitivos são componentes passivos que traduzem uma variação de uma grandeza física numa variação de capacidade. Neste tipo de sensores, o sensor possui contatos metálicos condutores fixos num determinado material isolante, e o meio envolvente funciona como eletrólito de um condensador, respetivamente. A figura 2.2 ilustra a construção de um sensor capacitivo e das linhas de campo criadas à sua volta, com e sem objetos na sua proximidade. O sensor de humidade é um exemplo de um sensor capacitivo. A presença das gotas de água vai deformar as linhas de campo e por sua vez alterar o valor da capacidade. Outro exemplo é o sensor de proximidade ou de toque, em que a presença de um dedo irá aproximar do sensor uma ligação à terra, causando variação nas linhas de campo do sensor, e por sua vez o valor da sua capacidade.

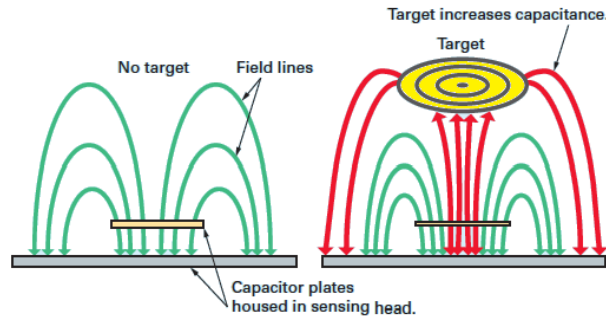


Figura 2.2: Condensador a funcionar como sensor de proximidade. Um corpo que altere as linhas de campo irá modificar a capacidade resultante.

2.3 Sensores Indutivos

Os sensores indutivos são construídos baseados em bobinas e aplicados na deteção da variação de materiais ferromagnéticos no seu núcleo, o que vai causar uma variação no valor da sua indutância. O seu uso é idêntico ao do sensor capacitivo, podendo controlar a frequência de um oscilador tal como os sensores capacitivos.

A figura 2.3 mostra uma bobina variável, que tem um núcleo de ferrite que se desloca ao ser enroscado, e a construção de um sensor indutivo que traduz um deslocamento em variação de indutância. Este tipo de construção é usada em sistemas que não permitem a utilização de outros sensores como a resistência variável. Este sensor também pode ser usado para identificar líquidos com permeabilidades magnéticas diferentes. É necessário cuidado especial na escolha do sensor e no projeto do circuito condicionador de sinal para que o fenómeno físico seja aferido corretamente.

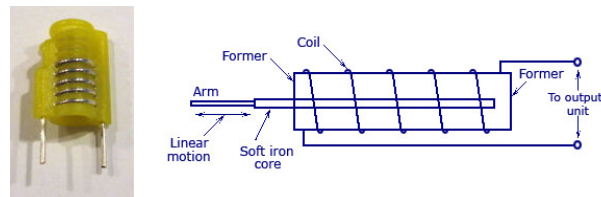


Figura 2.3: Exemplo de uma bobina variável e esquema de possível de construção de uma bobina variável.

2.4 Calibração de um Sensor

A tradução de uma grandeza física noutra elétrica por um sensor segue um comportamento bem conhecido e modelado através uma função de transferência, ou curva característica. Por exemplo um sensor de temperatura

do tipo PT100 varia a sua resistência em função da temperatura medida: $R = R_0(1 + 3,9 * temp/1000)\Omega$, e tem os seus valores da medida obtidos dentro de uma determinada margem de erro, derivada da tolerância de fabrico. A informação sobre a função de transferência e a incerteza do sensor consta do *datasheet* fornecido pelo fabricante.

Outra origem de incerteza numa medida de um sensor, tem a ver com a sua montagem, ou posição inicial. Para além das variações na sua construção e montagem, os restantes componentes eletrónicos do sistema são também eles fabricados com margens de tolerância, e vão por isso introduzir desvios no funcionamento circuito. Como os componentes vão-se degradando ao longo do tempo, as suas propriedades vão-se alterar, introduzindo desvios adicionais ao circuito. Para aumentar a exatidão de uma medida, e obter uma medida mais próxima do valor real, é necessário calibrar o sensor e/ou o circuito onde o sensor está ligado.

É possível calibrar um sistema de medida através da execução dos seguintes passos:

- Instalar o sensor numa posição conhecida ou em repouso, e realizar uma medida sem a presença de qualquer perturbação física;
- Com o sistema em funcionamento, para além da posição de repouso, medir outros pontos de referência, tais como os valores de limite mínimo e máximo, e outros pontos médios bem conhecidos;
- Determinar o desvio da leitura em relação ao valor esperado para a condição de funcionamento;
- No circuito: adicionar componentes variáveis para que o circuito seja ajustado até o desvio da medida ser nulo;
- No programa: modificar a função de transferência para compensar o desvio e produzir o valor correcto.

A vantagem de calibrar o circuito do sensor é poupar no número de instruções executadas e no tempo de execução do programa com este processo. A desvantagem é que requer que o circuito já tenha esta funcionalidade presente, no entanto nem sempre é possível construir um circuito capaz de anular esse desvio, e exige ao utilizador tempo para a calibração manual.

A vantagem da calibração por software é a facilidade em calcular qualquer função matemática, e repetir o processo frequentemente sem ser necessário alterar o circuito, necessitando de intervenção mínima ou nula por parte do utilizador. A desvantagem é o tempo gasto em processamento e a memória ocupada. Em termos práticos, é necessário adicionar uma função de calibração do sensor à classe do sensor, e que deve ser chamada durante a execução da função `setup()` do Arduino.

A implementação da função de calibração consiste na leitura de várias amostras do sensor, em pontos conhecidos, e obter os valores máximo, mínimo, media e moda das últimas. Uma forma simples de correção do valor do sensor é através de uma aproximação linear ($Y = Ax + B$) onde os valores de A e B são escolhidos para minimizarem a diferença entre o valor real e o valor medido, no entanto esta equação não é a mais indicada para sensores com andamento exponencial ($Y = A.B^x$) ou quadrático ($Y = Ax^2 + Bx + C$). Alguns dos vários sensores abordados neste capítulo requerem calibração, nomeadamente: termómetro, acelerómetro e higrómetro.

2.5 Interruptores Momentâneos

Estes componentes interrompem (abrem) ou estabelecem (fecham) um circuito electrónico aquando da existência de uma força mecânica neles aplicada que os obriga a mudar de estado, o circuito controlado terá o mesmo nível de tensão que o existente no outro pino do interruptor. Alguns têm uma mola que os obriga a voltar ao estado anterior quando não existe nenhuma força aplicada sobre eles - botão de pressão. Embora, a nível eléctrico, todos tenham o mesmo princípio de funcionamento, a maior diferença consiste na sua construção.

Estes componentes são caracterizados por:

- número de posições, normalmente 2: posição de repouso e atuado, correspondente a circuito aberto ou fechado;
- número de circuitos em paralelo, normalmente 1;
- posição de repouso: circuito normalmente aberto (NA) ou normalmente fechado (NF);

2.5.1 Botões de Pressão

Os botões de pressão são usados para traduzir uma força nele aplicada no estabelecimento um circuito electrónico, contrariando o estado de repouso. A diferença entre um interruptor e um botão de pressão é o retorno automático à posição de repouso. Num sistema embebido, eles são muito usados para receber comandos do utilizador, por exemplo para dar início ou reiniciar um processo, seleccionar opções, ou incrementar uma quantidade. Para além de receberem comandos de um utilizador, os botões de pressão podem ser associados a qualquer acção mecânica, por exemplo, detecção de fim de curso numa impressora 3D ou detecção de uma janela ou porta aberta ou fechada.

Os botões de pressão mais básicos são constituídos por um veio de plástico, usado para receber o comando do utilizador, e está acoplado a um contacto metálico que faz a ponte entre dois terminais metálicos. O contacto

CAPÍTULO 2. SENSORES

metálico está posicionado para estabelecer o circuito entre os terminais do botão quando ele for premido. Existe uma mola que empurra o contacto metálico no sentido contrário à força nele exercida para que volte ao estado de repouso quando o botão não estiver premido, e por consequência altera o estado do circuito quando foi premido. Os botões de pressão podem ser do tipo Normalmente Aberto (NA), ou seja, liga os terminais quando premido, ou Normalmente Fechado (NF) que desliga os terminais quando premido. Os botões mais básicos contêm apenas dois terminais e são do tipo NA. A figura 2.4 mostra vários tipos de botões de pressão à esquerda e o seu símbolo elétrico à direita, e a figura 2.6 mostra um modelo simplificado de construção de um botão de pressão. Os botões podem ser fabricados para terem mais do que uma posição, estabelecendo a ligação entre diversos pontos e terem mais do que um circuito em paralelo. Alguns exemplos são:

- a) 1 circuito com 1 posição - ao pressionar o botão estabelece um circuito apenas numa posição (pressionado ou solto);
- b) 1 circuito com 2 posições - em qualquer posição o botão tem um circuito estabelecido;
- c) 2 circuitos com 1 posição - ao pressionar o botão, ele vai atuar sobre dois contactos metálicos separados mas solidários, estabelecendo o circuito apenas numa posição;
- d) 2 circuitos com 2 posições - o botão atua sobre um par de contactos metálicos solidários que em qualquer das duas posições têm os dois circuitos estabelecidos.

Estas características são válidas também para interruptores.



Figura 2.4: Aspeto de vários botões de pressão (esquerda) e representação simplificada (direita).

A figura 2.5 mostra um modelo de construção de um botão de pressão, composto por dois contatos

O circuito mais básico construído com um botão de pressão dá indicação da acção ao circuito através da imposição de um nível de tensão GND num terminal de entrada do Arduino. Como o circuito do botão de pressão apenas estabelece um circuito, quando o botão não está premido não existe a tensão

2.5. INTERRUPTORES MOMENTÂNEOS

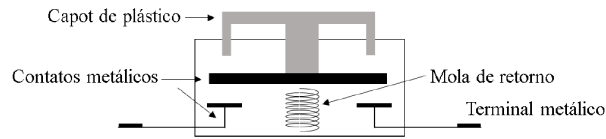


Figura 2.5: Modelo de construção de um botão de pressão.

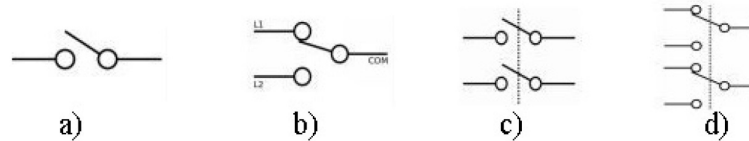


Figura 2.6: Símbolo elétrico dos botões de pressão.

GND aplicada pino de entrada ficando sensível à influência do ruído envolvente. Para mitigar este problema, normalmente são usados *pull-ups* para impor um nível de tensão diferente do usado no botão, abordados na secção 1.3.1. Os pinos de entrada digital dos MCUs do Arduino possuem *pull-ups* internos configuráveis. A figura 2.7 mostra o esquema elétrico da ligação de um botão de pressão a um Arduino, incluindo a resistência de *pull-up*. No entanto, esta resistência só é necessária aquando da utilização de pinos de entrada sem *pull-up* interno.

Para realizar um ensaio com um botão de pressão pode-se realizar o circuito em *breadboard*, tal como ilustrado na figura 2.10.

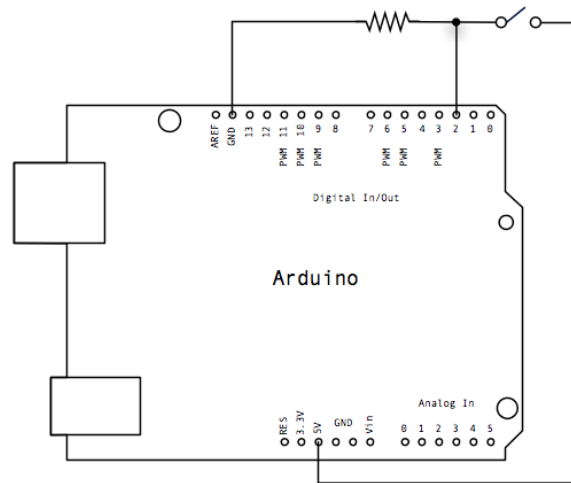


Figura 2.7: Circuito electrónico para ligar um botão de pressão ao Arduino.

Uma característica dos botões de pressão, e outros sensores que partilham o mesmo princípio de funcionamento, é o fato de quando um botão é premido ou libertado, observarem-se transições espúrias do sinal, ou seja, existem várias

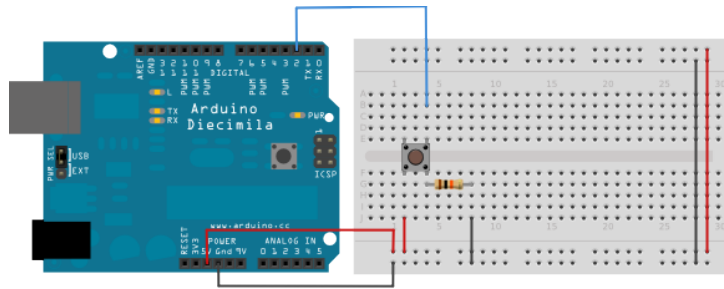


Figura 2.8: Realização do circuito do botão de pressão em breadboard

mudanças de 0 para 1 e de 1 para 0 antes do sinal estabilizar (*bouncing*), que podem originar várias a indicação de atuações indesejadas no circuito. Este comportamento pode ser aferido com um osciloscópio no sinal de entrada, tal como ilustrado na figura 2.9, em cima.

Este problema pode ser resolvido de duas formas: 1) com componentes extra no circuito para filtrar o ruído inicial ou 2) com programação para não considerar as ativações espúrias. Neste caso o circuito de *debounce* é constituído por uma resistência em série com um condensador ligados entre o pino de entrada digital do MCU e GND, equivalente a estarem ligados em paralelo com os terminais do botão de pressão, constituindo um filtro RC passa-baixo de primeira ordem. Valores típicos para a resistência são 47-100 Ω e para o condensador 10-47 μF . A constante de tempo associada a este filtro ($\tau = RC$) ronda o milissegundo. A constante de tempo deve ser escolhida de forma a que elimine as transições espúrias, mas não as atuações legítimas. Dependendo da construção do botão de pressão e da condição física da pessoa, podem ser detectadas cerca de 15 atuações do botão por segundo, o que dá pouco mais de 6 ms entre atuações.

É frequente ver circuitos publicados em que a resistência do filtro RC é desprezada, ligando apenas um condensador nos terminais do botão de pressão. Nessas realizações, cada vez que o botão é premido, embora por breves instantes, o condensador é curto-circuitado pelo botão, ou que significa que ele é descarregado rapidamente pelo botão de pressão. Dependendo da tecnologia e da capacidade do condensador, a corrente que percorre o botão de pressão pode ser significativa ao ponto de poder gerar picos de corrente, o que leva a degradação acelerada do condensador e do botão de pressão.

A solução para *debounce* to botão de pressão em software envolve a avaliação do estado do botão durante um intervalo de tempo aceitável para que seja considerado atuação válida. Assim que é detetada uma alteração no pino de entrada do Arduino é iniciada uma contagem de tempo. Se durante o “período de carência” existir alguma transição no sinal de entrada, o contador é reiniciado e a alteração ignorada. A figura 2.9, em baixo, mostra um

2.5. INTERRUPTORES MOMENTÂNEOS

sinal de saída de um Arduino correspondente ao sinal do botão de pressão depois de filtrado por software. Esta filtragem introduz um atraso de cerca de 70 ms depois do sinal ter estabilizado, o que não é crítico tendo em conta o atraso pela acção do utilizador.

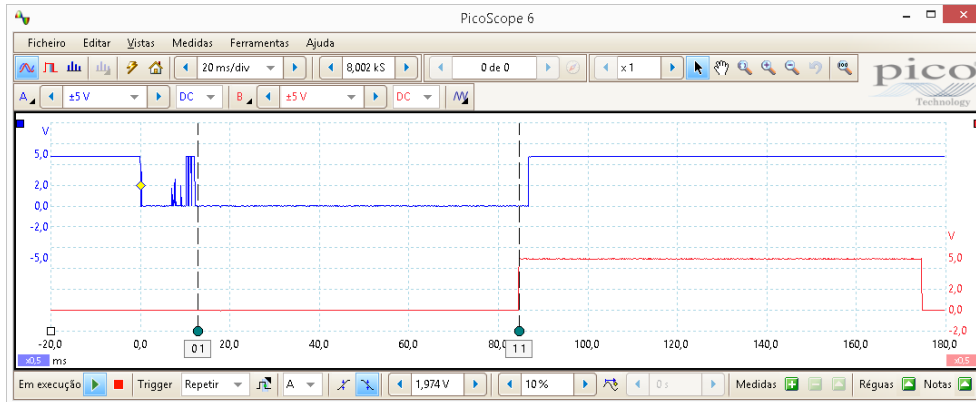


Figura 2.9: Sinais produzidos pelo botão de pressão quando premido e gerado pelo Arduino. Em cima o sinal observado na saída do botão de pressão, onde é possível ver a presença de transições espúrias, e em baixo o sinal gerado pelo Arduino após a execução das rotinas de *debounce*.

A listagem 2.2 apresenta um Sketch Arduino para demonstração da classe `PushButton` com suporte para supressão de transições espúrias do botão de pressão. Esta classe oferece os métodos acessíveis ao nível da aplicação com a funcionalidade descrita na tabela 2.1. Os atributos privados usados na classe `PushButton` sumarizados na tabela 2.2.

Listagem 2.2: Listagem do Sketch com demonstração da utilização do botão de pressão.

```
// demonstração do botão de pressão

const short buttonPin = 2; // pushbutton IO pin number
const int ledPin = 13; // LED IO pin number
char buf[32];

// activeMode: 1-pressed, 2-depressed, 3-both
#define BTN_H2L 1
#define BTN_L2H 2
#define BTN_TDEBOUNCE 75

class PushButton {
private:
    byte pin; // input pin
    byte inputMode, activeMode; // input pin config, (de)press
    bool btnVal; // button current value
    bool btnCh; // button has changed
    unsigned long btnChTime; // time of key changed
    bool btnHitFlag; // there's button activity
    int debounceTime; // settle time
    void (*actionListener)(byte); // function to be executed upon activity
public:
    PushButton() { // default constructor
        this->inputMode = INPUT_PULLUP;
    }
};
```

CAPÍTULO 2. SENSORES

Método	Descrição
<code>PushButton()</code>	construtor por omissão (sem parâmetros) da classe <code>PushButton</code> . Usa o pino D2 sem <i>pull-up</i> , com 75 ms de tempo de estabilização e sensível às transições L-H (largar o botão)
<code>PushButton(const short pin, byte inputMode, byte activeMode, byte debounce)</code>	construtor da classe <code>PushButton</code> com passagem de parâmetros, nomeadamente o pino de entrada, existência de <i>pull-up</i> , transição de ativação e o tempo de estabilização
<code>void init()</code>	serve para configurar do pino do botão de pressão e afetação da variável internas do estado do botão - assume estado de repouso
<code>void scan()</code>	trata de observar alterações no estado do botão de pressão e identificar ativação válida do botão de pressão
<code>bool btnHit()</code>	indica se houve ativação válida do botão de pressão
<code>byte getVal()</code>	devolve o estado do botão
<code>void setActionListener(void (*al)(byte))</code>	configura a função a ser invocada sempre que se verificar ativação válida do botão de pressão
<code>void printInfo()</code>	envia para o terminal os parâmetros do objeto da classe <code>PushButton</code> . Vale a pena usar este método durante o estudo e ensaios preliminares, mas não faz falta durante o funcionamento normal e apenas ocupa espaço de memória (a remover).

Tabela 2.1: Métodos da classe `PushButton`

```

    this→pin = 2;
    this→activeMode = BTNL2H;
    debounceTime = BINTDEBOUNCE;
}
PushButton(const short pin, byte inputMode, byte activeMode, byte debounce) {
    this→inputMode = inputMode;
    this→pin = pin;
    this→activeMode = activeMode;
    debounceTime = debounce;
}
void init(){
    pinMode(pin, inputMode);
    btnVal = HIGH; // idle
}
void scan() {
    byte a, btnOldVal;
    unsigned long now;
    a = digitalRead(pin);
    if(btnVal != a) {
        btnVal = a;
        btnCh = HIGH;
        btnChTime = millis();
    }
    else {

```

2.5. INTERRUPTORES MOMENTÂNEOS

Função	Descrição
byte pin	número do pino de entrada do Arduíno
byte inputMode	configuração do <i>pull-up</i> interno
byte activeMode	transição de ativação do botão: ao pressionar (1/BTN_H2L), ao largar o botão (2/BTN_L2H) ou ambos (3/BTN_H2L—BTN_L2H)
bool btnVal	valor atual do botão (HIGH/LOW)
bool btnCh	indicação de que o estado do botão mudou
unsigned long btnChTime	tempo em que foi detetada última alteração de estado do botão
bool btnHitFlag	indicação de que o botão tem um novo valor válido
int debounceTime	tempo de estabilização do valor do botão de pressão, para obtenção de um valor válido
void (*actionListener)(byte)	função invocada automaticamente pelo método <code>scan()</code> aquando da detecção de um novo valor válido

Tabela 2.2: Atributos privados da classe PushButton.

```

now = millis();
if((btnCh == HIGH) && (now - btnChTime >= debounceTime)) {
    if(((activeMode & 1) && (btnVal == LOW)) || ((activeMode & 2) && (btnVal == HIGH))) {
        btnOldVal = btnVal;
        btnHitFlag = HIGH;
        btnCh = LOW; //
        if(actionListener != NULL) {
            actionListener(btnVal);
        } // if
    } // if
} // else
}

bool btnHit() {
    bool p = btnHitFlag;
    btnHitFlag = LOW; // btn change has been observed, clear it
    return p;
}

byte getVal() {
    return btnVal;
}

void setActionListener(void (*al)(byte)) {
    actionListener = al;
}

#ifdef 1
void printInfo() {
    Serial.println("Button info: ");
    Serial.println(" Pin: " + (String)pin);
    Serial.print(" Active: ");
    switch(activeMode) {
        case 1: Serial.println("Pressed");          break;
        case 2: Serial.println("Released");         break;
        case 3: Serial.println("Pressed+Released"); break;
        default: Serial.println(" Invalid Condition");
    }
    Serial.println(" Input Mode: " + (String) (inputMode == INPUT ? "EXT PULL-UP" : "INT PULL-UP"));
    Serial.println(" Debounce Time: " + (String)debounceTime + "ms");
    sprintf(buf, " Action Listener: %p", actionListener);
    Serial.println(buf);
    Serial.println("-----");
}

```

```
#endif
};

PushButton pb1(buttonPin, INPUT_PULLUP, BTN_L2H | BTN_H2L, BINT_DEBOUNCE);

byte actPerf = 0;
void btnActionPerformed(byte val) {
    if(val == HIGH) {
        actPerf = 1;
    }
    else {
        actPerf = 2;
    }
}

void setup() {
    Serial.begin(9600);
    while(!Serial);
    Serial.println("*** Push-Button Demo ***\n");
    pb1.init();
    pb1.setActionListener(btnActionPerformed);
    pb1.printInfo();
    pinMode(ledPin, OUTPUT); // LED
}

void loop() {
    pb1.scan();
    // Este if controla a execução por polling ou por ação associada.
    #if 0
        if(pb1.btnHit()) {
            //Serial.println(pb1.getVal());
            digitalWrite(ledPin, HIGH);
        } else {
            digitalWrite(ledPin, LOW);
        }
    #else
        if(actPerf != 0) {
            digitalWrite(ledPin, (actPerf == 2) ? HIGH : LOW); // LED
            //Serial.println("btnActPerf: " + (String)(actPerf == 1 ? "HIGH" : "LOW"));
            actPerf = 0;
        }
    #endif
    delay(10);
}
```

O método mais relevante da classe `PushButton` é o `scan()`, onde é feita a detecção de ativação válida do botão de pressão. Este método começa por ler o valor do nível lógico do pino de entrada e compara-o com o valor lido anteriormente no mesmo pino. Se tiver ocorrido alguma alteração no nível lógico do pino de entrada, o valor de `BtnVal` vai ser diferentes, e neste caso é dada a indicação de que existiu uma alteração (`btnCh = HIGH`) e é registado o tempo atual. Caso os valores sejam iguais, significa que o botão permaneceu no mesmo estado (pressionado ou em repouso), e neste caso é preciso avaliar se houve alguma alteração do seu estado, avaliando o `btnCh`. Se `btnCh` estiver a `HIGH` e a diferença entre o valor do tempo atual e o tempo quando houve alteração no botão for superior ao tempo de estabilização (`debounceTime`) então vai ser avaliado o tipo de ação, se foi pressionar ou libertar o botão, de acordo com a configuração do botão. Se o tempo for menor, significa que houve uma alteração ao estado do botão dentro do período de carência para considerar o botão estabilizado.

O botão pode ser configurado para ficar sensível ao pressionar (`HIGH-`

2.5. INTERRUPTORES MOMENTÂNEOS

LOW) ou ao libertar (LOW-HIGH) do botão. Esta configuração é feita através da passagem de um valor constante no construtor: `BTN_H2L` e `BTN_L2H`, correspondente aos valores 1 e 2 respectivamente. Caso se queira o botão sensível às duas transições deve ser usada a combinação dos dois, neste caso `BTN_H2L | BTN_L2H`, o que se traduz na operação lógica $(1 \text{ OR } 2 = 3)$ em decimal, ou $(01 \text{ OR } 10 = 11)$ em binário.

O uso de uma função `ActionListener` tem o propósito de associar ao objecto um método que é executado automaticamente para tratamento do evento pela função `scan()`. O método deve ser o mais breve possível e não deve demorar tempo a interagir com periféricos do Arduino, pois a execução do resto do programa ficará dependente da execução deste método. Habitualmente este tipo de métodos apenas escreve variáveis em memória para serem observadas no corpo do `Sketch`, no exemplo dado `actPerf` codifica a atividade do botão - 0: sem atividade, 1: libertado (L-H), e 2: pressionado (H-L). Caso não se pretenda este comportamento, a alternativa é usar o método `btnHit()` para avaliar se houve uma tecla premida e depois usar o método `getVal()` para obter o valor do botão.

A figura 2.10 mostra a realização do circuito de ensaio em breadboard (à esquerda), e uma *breakout* com um botão de pressão e resistência de *pull-up* soldados. Como os pinos de E/S digitais do Arduino possuem *pull-up* interno não é necessário adicionar uma resistência na *breadboard*. A *breakout* possui três terminais de ligação sendo o primeiro o sinal digital a ligar à entrada e o terceiro a ligação a GND. O pino do meio serve para a ligação de VCC à resistência que se encontra na placa. Esta ligação não é necessário se o *pull-up* interno do Arduino for usado. A tabela 2.3 mostra a correspondência dos pinos e a sua funcionalidade.

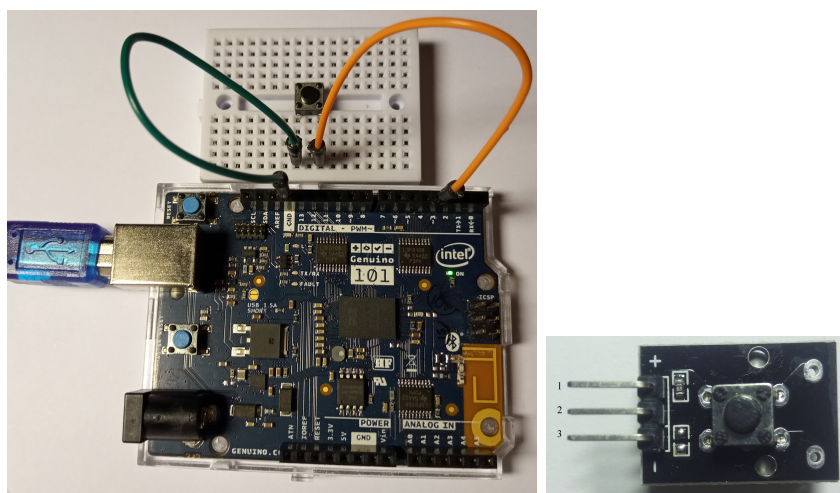


Figura 2.10: Realização do circuito do botão de pressão em breadboard (esquerda) e *breakout* (direita).

Pino	Descrição
1	Sinal do botão
2	VCC (opcional)
3	GND

Tabela 2.3: Pinos da placa do botão de pressão e sua descrição.

2.5.2 Sensor de Movimento

O sensor de movimento é usado para detetar se um determinado corpo abandonou o estado de repouso, por acção de uma força nele exercida. Este sensor é usado por exemplo para alarmes ou para sistemas eletromecânicos onde é necessário detetar que um determinado elemento se encontra numa posição de repouso. Ele é normalmente soldado na placa de circuito impresso, que estando na posição horizontal e solidária com o resto do equipamento, permite detetar se o sistema sofreu algum deslocamento.

O sensor de movimento usa o mesmo circuito do botão de pressão, abre ou fecha uma ligação. Em termos de funcionamento, em vez da pressão do dedo do utilizador, o contacto é feito por uma esfera metálica existente no interior do sensor. Em repouso, a gravidade faz com que ela fique assente nos terminais a fazer contacto eléctrico entre os terminais. Em ação, os estímulos externos fazem a esfera saltar, interrompendo o circuito. Contrariamente ao comportamento do botão de pressão, neste caso não há interesse em suprimir a deteção de transições válidas ou espúrias. Neste caso, o interesse é registar qualquer atividade.

O símbolo eléctrico é o mesmo do que o do botão de pressão, ou podem haver variantes que, no fundo, deixam transparecer como o componente funciona. A figura 2.11 mostra um sensor de vibração montado numa *breakout* board (esquerda) e o modelo de como ele é construído (direita). A *breakout* é igual à do botão de pressão e é ligada da mesma forma.

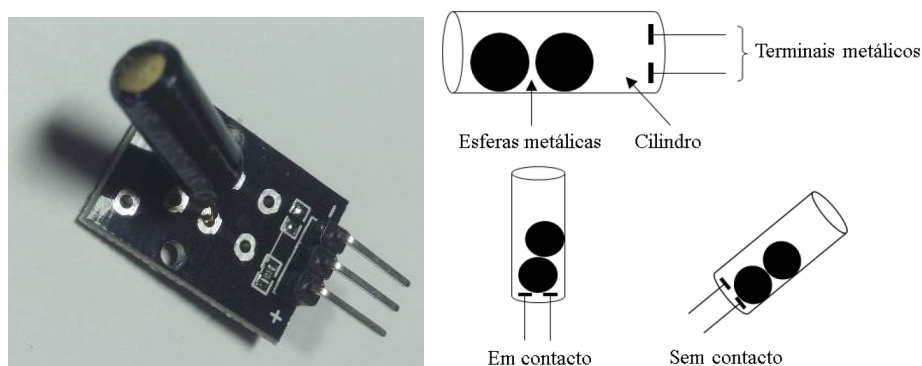


Figura 2.11: Sensor de movimento (esquerda) e esquema do modelo de construção simplificado (direita).

A listagem 2.3 exemplifica o funcionamento da classe `basicSensor`. O construtor recebe como parâmetros o pino onde o sensor fica ligado e a configuração do pino (`INPUT` ou `INPUT_PULLUP`). O método `calibrate()` é executado no arranque do Sketch, dentro método `setup()`, para determinar o estado de repouso do sensor. O método aguarda que o sensor se encontre em regime estacionário, com um sinal constante no pino de entrada. Nesta classe o método `check()` procura identificar alterações de estado no sinal do pino de entrada do sensor. Neste caso é indiferente ao tipo de transição H-L ou L-H e à duração do sinal.

Listagem 2.3: Sketch para ensaio do sensor de movimento.

```
// basic on-off sensor

class basicSensor {
  private:
    byte pin, pinMode;
    byte curState, prevState;
  public:
    basicSensor(byte pin = 2, bool pinMode = 0) {
      this->pin = pin;
      this->pinMode = pinMode;
    }
    void init() {
      prevState = 0;
      pinMode(pin, pinMode);
    }
    void calibrate() {
      do {
        prevState = curState;
        curState = digitalRead(pin);
        delay(10);
      } while (prevState != curState); // wait for input pin to settle
    }
    bool getState() {
      curState = digitalRead(pin);
      return curState;
    }
    bool check() {
      prevState = curState;
      curState = digitalRead(pin);
      if (prevState != curState) { return 1; } // new pin read differs
      else { return 0; }
    }
}

movSensor(2);

void setup() {
  Serial.begin(115200);
  while(!Serial); // wait for serial port
  movSensor.init();
  movSensor.calibrate();
}

bool a;
void loop() {
  a = movSensor.check();
  if(a) {
    Serial.println("Sensor Active!!");
  }
}
```

As ligações deste sensor são as mesmas que as do botão de pressão apresentado anteriormente. A figura 2.12 mostra o diagrama de forma de onda

CAPÍTULO 2. SENSORES

do sinal gerado pelo sensor de movimento, com detalhe do instante em que a placa de circuito onde o sensor está instalado é movido ligeiramente.

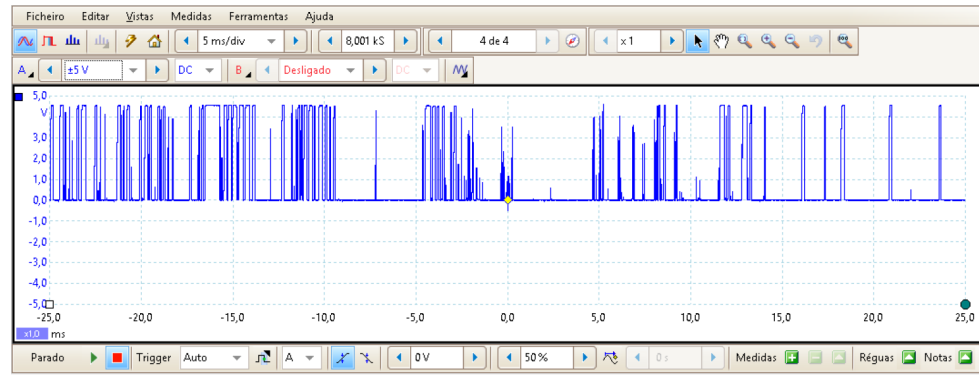


Figura 2.12: Sinal gerado pelo sensor de movimento.

2.5.3 Sensor de Choque

O sensor de choque é semelhante ao sensor de movimento, mas é menos sensível. Ele é apenas sensível a pancadas e choques. O sensor de choque quando acoplado a um objeto pode ser movido sem originar ativação do circuito, mas dará indicação se o objeto embater numa superfície. A figura 2.13 mostra o aspeto do sensor de choque. É possível observar a mola no interior do corpo do sensor.

Este sensor tem dois terminais e é construído por uma mola fixa apenas numa extremidade ligada a um dos terminais, estando a outra extremidade da mola solta e próxima do outro terminal do sensor. Quando o sensor recebe uma pancada, a mola começa a oscilar e toca no contacto da extremidade, fechando momentaneamente o circuito entre os dois terminais do sensor. No estado de repouso como não existe contacto feito pela mola não é necessário calibrar o sensor para determinar o seu estado inicial.

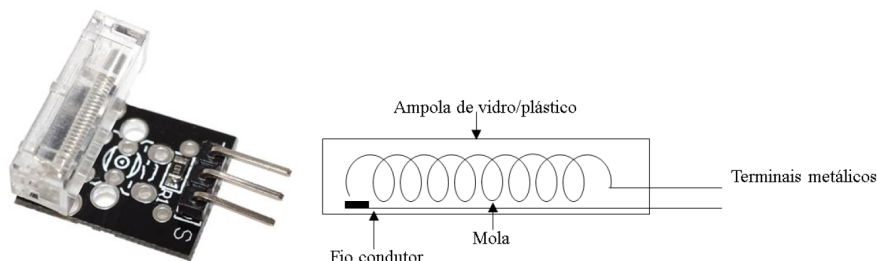


Figura 2.13: Sensor de choque (esquerda) e esquema do modelo de construção simplificado (direita).

2.5. INTERRUPTORES MOMENTÂNEOS

O Sketch usado para testar este sensor é o mesmo que foi usado para o sensor de movimento, na listagem 2.3. A figura 2.14 mostra o diagrama de forma de onda gerada pelo sensor em funcionamento. A placa de demonstração onde o sensor ensaiado foi soldado possui as mesmas ligações que a do botão de pressão.

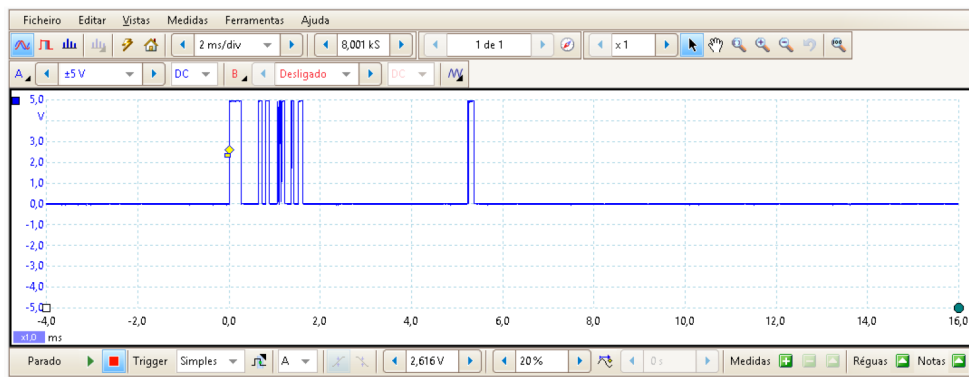


Figura 2.14: Sinal gerado pelo sensor de choque.

2.5.4 Sensor de Inclinação

O sensor de inclinação é também semelhante ao sensor de movimento, contudo ele era construído tradicionalmente com uma bola de mercúrio se deslocava numa ampola de vidro¹. O limiar de inclinação era definido através da moldagem da inclinação do componente dobrando os seus terminais.

Numa das extremidades da ampola encontram-se os terminais do sensor, que ficam em contacto quando a bola de mercúrio toca em ambos. O modelo de construção deste sensor encontra-se na figura 2.15. O sensor é montado ligeiramente inclinado para que em repouso a esfera fique parada numa das extremidades, e quando o corpo a que ele se encontra solidário tiver uma inclinação contrária à do sensor, a esfera desloca-se para a outra extremidade. A figura 2.15 em baixo à esquerda, mostra um sensor de inclinação baseado na bola de mercúrio, e o modelo de como ele é construído. Este sensor pode ser testado com o Sketch usado para testar o sensor de movimento, na listagem 2.3.

Existe uma variante do sensor de inclinação mas que apenas funciona na posição horizontal. Este sensor tem o mesmo princípio de funcionamento mas é construído numa esfera de metal para estabelecer o contacto entre dois terminais, no entanto os seus terminais estão em lados opostos do sensor e

¹Embora o mercúrio tenha a vantagem de ser moldável e não oxidar, oferecendo melhor contacto entre os terminais do que uma esfera de metal, é uma substância venenosa é por isso é necessário cuidado ao manipular um sensor deste tipo para que o vidro não se parta e liberte o mercúrio.

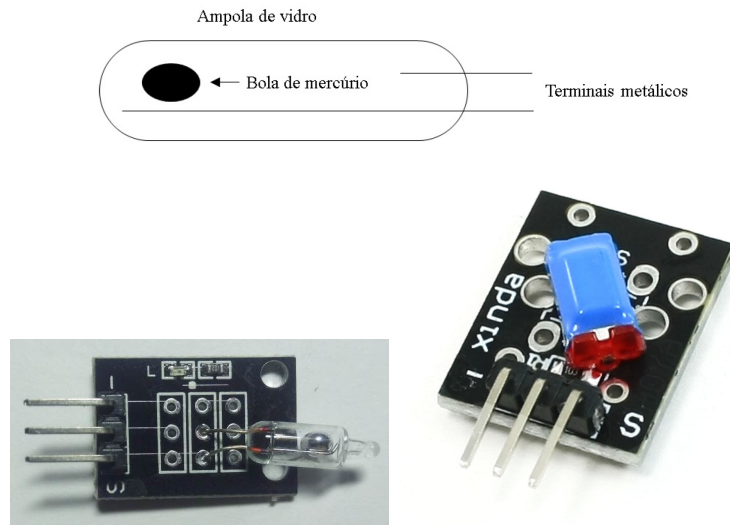


Figura 2.15: Modelo de construção simplificado (cima), sensor de inclinação com ampola de vidro (baixo à esquerda), e sensor de inclinação com esfera metálica para PCB (baixo à direita).

preparados para serem soldados em PCB. A figura 2.15 em baixo à direita, mostra o aspeto desta variante de sensor de inclinação.

Em termos de funcionamento, a bola de mercúrio, sendo maleável, não ressalta quando desliza e toca nos contactos internos do sensor, e estabelece contacto imediatamente, ao passo que a esfera de metal rígida, quando em deslocamento adquire energia cinética que não é imediatamente absorvida pelos contactos do sensor e dá origem a ressaltos e transições espúrias no sinal. A figura 2.16 mostra o diagrama de forma de onda gerada pelos dois tipos de sensores de inclinação em funcionamento. O sensor com bola de mercúrio no topo e esfera metálica em baixo.

Em termos da lógica do circuito e do programa do MCU é indiferente se o sensor estabelece o circuito quando em repouso ou sob influência de uma força.

2.5.5 Toque / Sensor Tátil

O sensor tátil permite dar indicação de atuação com um leve toque no sensor, isto é, sem ter de exercer força sobre ele. Este tipo de sensores não tem partes móveis e funciona com base num circuito que amplifica o sinal proveniente da interferência causada pelo contacto ou proximidade com o corpo humano. O corpo humano está constantemente exposto a radiação ambiente e facilita o contacto em relação à terra.

Um sensor deste tipo pode ser criado com um transistor de elevado ganho tal como o KSP13, montado na placa *breakout* mostrada na figura 2.17. O

2.5. INTERRUPTORES MOMENTÂNEOS

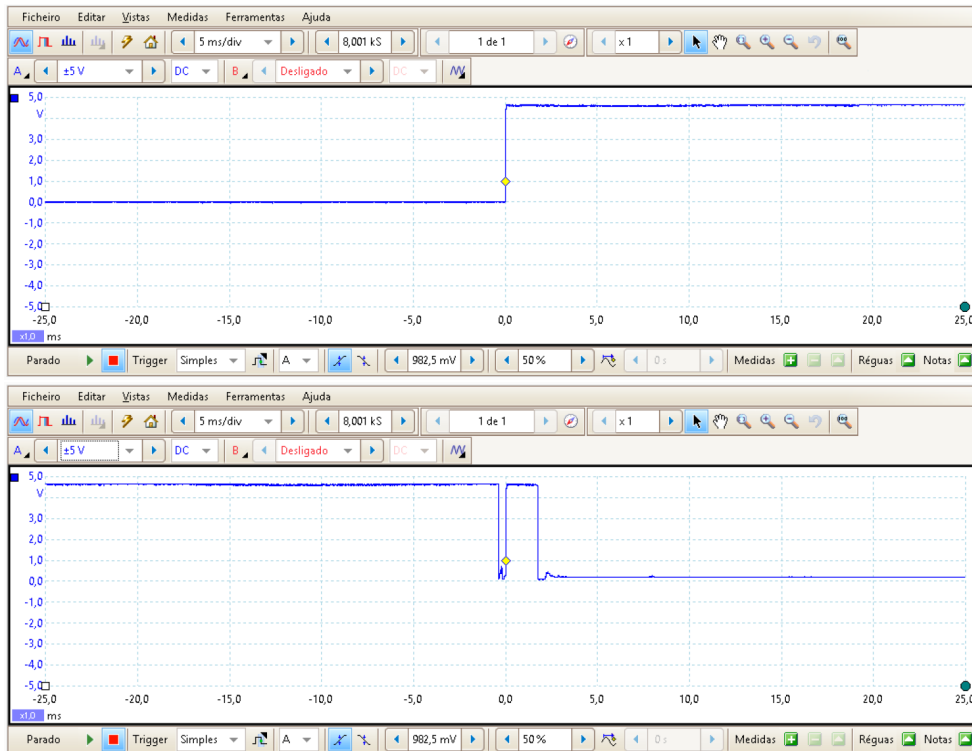


Figura 2.16: Sinal gerado pelo sensor de inclinação com bola de mercúrio (topo) e esfera metálica (baixo).

KSP13 é um transistor (Darlington), composto por um par de transistores ligados em cascata, para aumentar o ganho, e disponibilizados no mesmo encapsulamento dos transistores convencional (TO-92). Nesta montagem, o transistor atua como um amplificador de cerca de 5000x do sinal “captado” pela sua base. Como o sinal interferente mais abundante é o sinal da rede elétrica (220 V, 50 Hz), quando o utilizador aproxima o dedo do sensor este começa a dar indicação de ativação com a frequência da rede elétrica (50 Hz). Por este motivo é necessário realizar o *debounce* do sinal, tal como no botão de pressão mecânico.

A figura 2.18 mostra o sinal produzido pelas saídas analógica (cima) e digital (baixo) da placa *breakout* do KSP13. Verifica-se que quando o utilizador toca no sensor é produzido um sinal retangular com uma frequência de cerca de 50 Hz. A utilização da saída digital permite eliminar pequenas variações, tais como as que existem à esquerda, no instante em que o sensor é tocado.

Uma variante deste tipo de sensor consiste numa montagem baseada num circuito oscilador, onde a frequência de oscilação é determinada por um condensador ou bobine variável. Quando se toca no sensor com um dedo, a massa do dedo altera a proximidade dos eletrodos do condensador fazendo

Para este tipo de circuitos o Arduíno consegue detetar a atuação do botão medindo a frequência de oscilação do circuito, através da contagem do número de pulsos detetados num segundo, e deteta a ativação do sensor quando a frequência de oscilação é diferente da de repouso. Este sensor requer calibração para poder aferir qual o limiar da frequência de oscilação correspondente à tecla tocada. O circuito oscilador e o princípio de funcionamento estão ilustrados na figura 2.19.

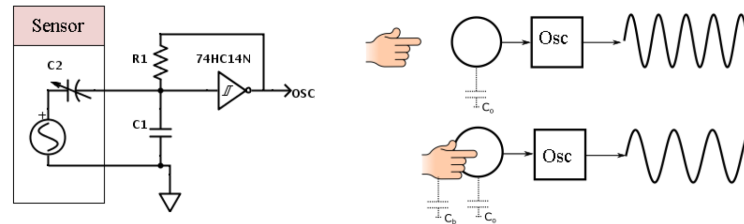


Figura 2.19: Sensor tátil baseado num circuito oscilador.

por iniciar o canal série para apresentar os resultados e configura o pino de entrada 3 para poder ficar sensível às transições do sinal via interrupção, e o pino 2 para gerar um sinal de teste (500 Hz).

O modo de medir a frequência do sinal de entrada consiste na colocação de um contador a zeros, associa a chamada da função `isr()` sempre que houver uma transição ascendente do sinal da interrupção 0 (pino 2). O programa fica suspenso durante um período de tempo para ser feita a medição. Durante este tempo, a função `isr()` apenas incrementa o valor da variável `count`. Findo o período de medição, a interrupção é desativa, e calculada a frequência do sinal.

Listagem 2.4: Listagem do Sketch para demonstração do sensor de toque.

```
// touch sensor
// Pin 2 (INT0): sensor input pin
// Pin 3 (PWM): sensor emulator (apx 500Hz)

float freq;
volatile unsigned long count;

void isr() {
  count++;
}

void setup() {
  Serial.begin(115200);
  pinMode(3, OUTPUT); // dummy PWM signal on pin 3 for verification
  analogWrite(3, 127);
  Serial.println("\n\n** Frequency Counter **\n\n");
}

float getFreq(unsigned int sampleTime) {
  count = 0;
  attachInterrupt(0, isr, RISING); // enable the interrupt
  delay(sampleTime); // isr will count while waiting
  detachInterrupt(0); // disable the interrupt
  return 1000.0 * (float)(count) / sampleTime;
}

void loop() {
  freq = getFreq(100);
  Serial.println(freq);
  delay(1000);
}
```

O TTP223 é um circuito integrado preparado para funcionar com um sensor tátil capacitivo, e encontra-se integrada num encapsulamento SOT-23 com 6 pinos. Com este circuito, o sistema embebido não necessita de estar

ocupado a determinar a frequência de oscilação, uma vez que ele fornece uma saída a 1 ou a 0 consoante o sensor é tocado ou não. A figura 2.20 mostra o diagrama do circuito para a montagem típica deste circuito integrado. O circuito requer ligação a VCC, GND e ao sinal de saída com indicação de atuação.

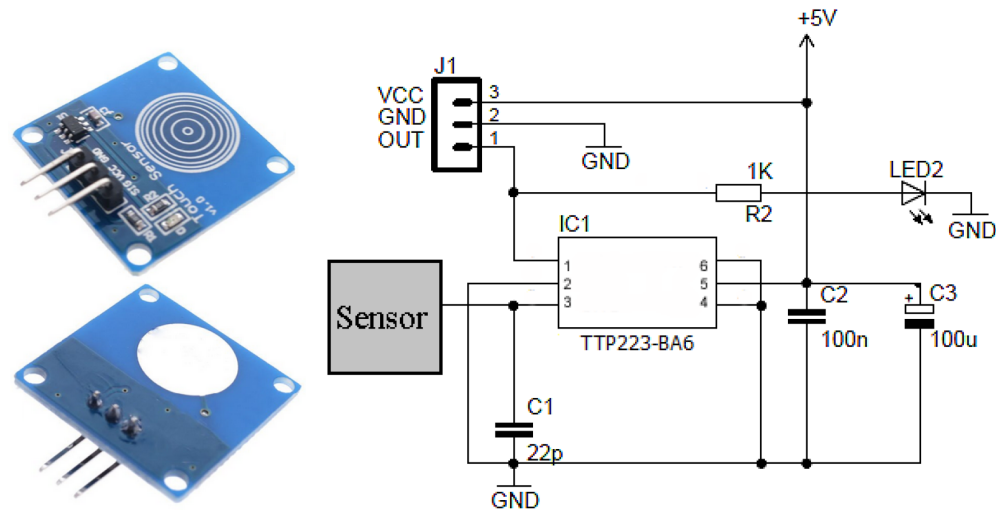


Figura 2.20: Módulo baseado no TTP223 para realizar um sensor tátil.

2.5.6 Interruptor Magnético

O interruptor magnético é conhecido como *Reed Switch* e comporta-se como um botão de pressão que funciona sob influência da presença de um campo magnético na sua proximidade, por exemplo de um íman permanente. Este tipo de sensores é usado para estabelecer ou interromper um circuito elétrico sem haver um contacto mecânico, sendo bastante usados em alarmes de portas e janelas onde a ausência do campo magnético gerado por um íman colado à porta ou janela faz disparar o alarme, determinar se existe pressão no fornecimento de água em máquinas de lavar louça, ou determinar se um computador portátil se encontra fechado para desligar o seu ecrã. Aplicações IoT para este tipo de sensores são deteção de níveis de líquidos com um íman colado num flutuador, ou acompanhar o movimento de uma máquina industrial para saber se está na posição correta para operar.

A figura 2.21 à esquerda mostra um sensor magnético soldado numa placa igual à do botão de pressão, que inclui um *pull-up*. À direita encontra-se o esquema simplificado de construção do sensor magnético. O interruptor magnético é constituído por dois contactos metálicos ferro-magnéticos e flexíveis que se encontram numa ampola de vidro preenchida com gás ródio. A presença do campo magnético faz juntar os dois contactos. Este tipo de

2.5. INTERRUPTORES MOMENTÂNEOS

sensores é lento a mudar de posição aberto/fechado, pelo que não é indicado para aplicações como conta-rotações de motores de alta velocidade. Existem também variantes deste sensor estando normalmente fechado, sendo o circuito elétrico interrompido pela presença do campo magnético.

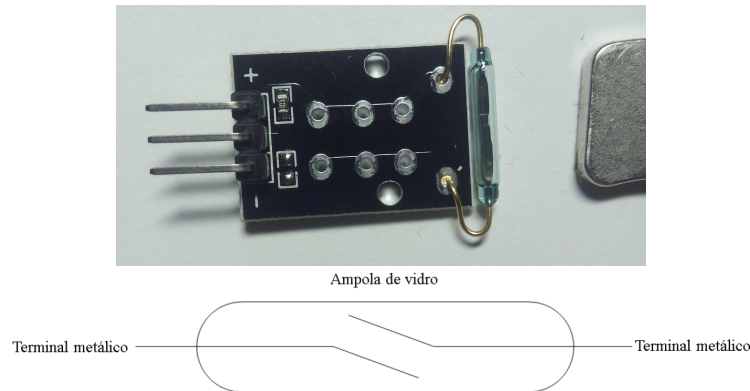


Figura 2.21: Interruptor magnético com um íman na sua proximidade, e modelo de construção.

O Sketch usado para testar este sensor é o mesmo que o sensor de movimento, apresentado na listagem 2.3. A figura 2.22 mostra o diagrama de forma de onda gerada pelo sensor em funcionamento. As transições espúrias são visíveis durante os primeiros $200 \mu s$, pelo que é recomendado a utilização de um programa ou circuito de *debounce* explicado aquando da apresentação do botão de pressão, para evitar múltiplas transições do sinal durante uma única atuação.

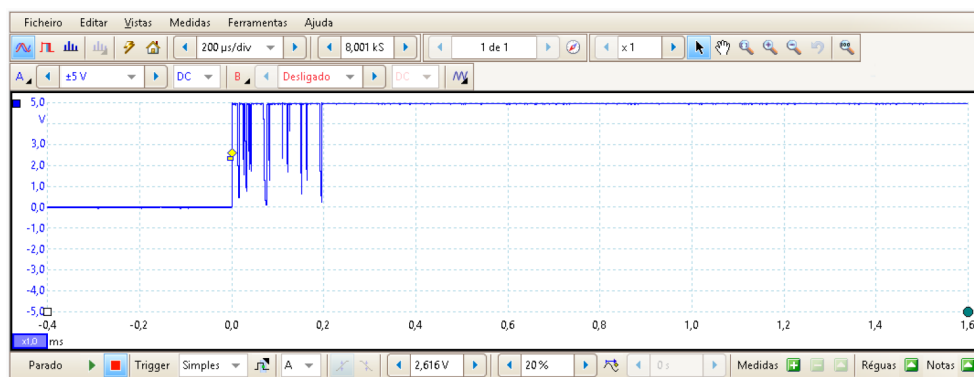


Figura 2.22: Sinal gerado pela ativação do sensor magnético.

2.5.7 Botão Rotativo

O botão rotativo dá indicação do sentido de rotação do eixo de um botão, semelhante ao controlo de volume de um rádio, e roda nos dois sentidos

CAPÍTULO 2. SENSORES

sem limite do número de voltas. Este componente é composto por dois contactos momentâneos que são atuados quando o botão roda, fornecendo dois sinais que codificam a direção de rotação: aumentar/sentido horário e reduzir/sentido anti-horário.

A figura 2.24, à esquerda, mostra o aspeto do botão rotativo instalado numa placa breakout, e o modelo de construção, com base em dois botões de pressão cujo contacto é provocado pelas saliências de um eixo dentado, tal como ilustrado nas figuras seguintes. À direita, a figura mostra as ligações existentes na placa *breakout*, incluindo as resistências de *pull-up*. A figura 2.24 mostra o modelo de construção do sensor do sentido de rotação, à direita, e os sinais A e B gerados pelo rodar de um botão.

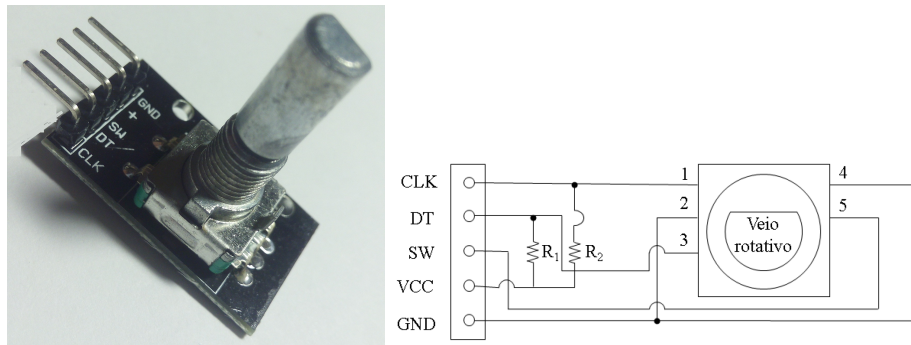


Figura 2.23: Sensor de sentido de rotação e ligações da placa *breakout* do sensor rotação.

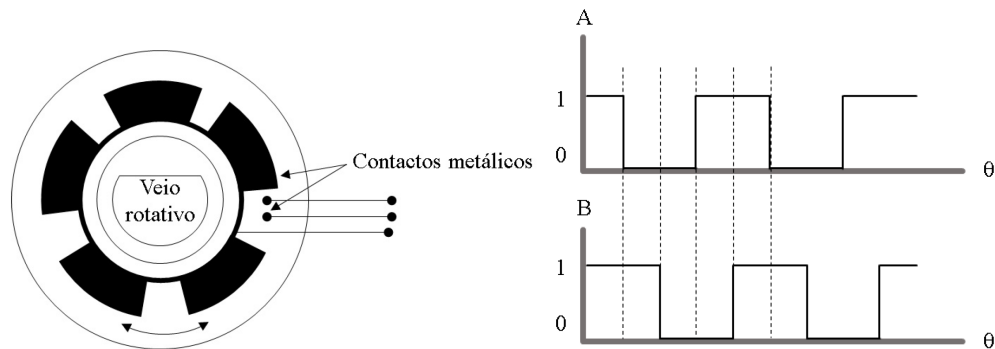


Figura 2.24: Modelo de construção do sensor do sentido de rotação (esquerda) e sinais gerados pela atividade do sensor (direita).

A listagem 2.5 mostra o Sketch para demonstração da classe `RotaryButton`, usada para interpretar os sinais do sensor rotativo.

Listagem 2.5: Listagem do Sketch para o botão rotativo

```
#define outputA 6
#define outputB 7
```

```

int counter = 0;
int aState;
int aLastState;

void setup() {
  pinMode (outputA,INPUT);
  pinMode (outputB,INPUT);
  pinMode(13, OUTPUT);
  Serial.begin (9600);
  // Reads the initial state of the outputA
  aLastState = digitalRead(outputA);
}

void loop() {
  aState = digitalRead(outputA); // Reads the "current" state of the outputA
  // If the previous and the current state of the outputA are different, that means a Pulse has occurred
  if (aState != aLastState){
    digitalWrite(13,HIGH);
    // If the outputB state is different to the outputA state, that means the encoder is rotating clockwise
    if (digitalRead(outputB) != aState) {
      counter ++;
    } else {
      counter --;
    }
    Serial.print("Position: ");
    Serial.println(counter);
  }
  digitalWrite(13,LOW);
  aLastState = aState; // Updates the previous state of the outputA with the current state
}

```

2.6 Posição

Para saber a posição de um alvo é necessário observar uma grandeza que varie com o comprimento ao longo de um, ou mais, eixos. O potenciômetro é um componente que produz uma variação de resistência linear com o deslocamento do seu eixo. Desta forma, se um alvo móvel for acoplado ao eixo de uma resistência variável é possível estabelecer uma relação entre a resistência e a distância.

2.6.1 Potenciômetro

O potenciômetro, ou reóstato, é uma resistência variável com 3 terminais, cujo valor é dependente da posição do seu eixo (deslizante ou radial) e pode ter um andamento linear (tipo A) ou logarítmico (tipo B). O valor da resistência varia entre 0 e um valor máximo definido pelo fabricante, característica do potenciômetro (R_{pot}). O potenciômetro funciona como uma resistência fixa entre os dois terminais das extremidades, mas como resistência variável entre cada um dos terminais dos extremos e o terceiro terminal ligado ao veio que se move ao longo do seu corpo. Os potenciômetros logarítmicos (tipo B) são usados em sistemas áudio de alta-fidelidade e são construídos em duplicado para atuar os dois canais stereo em simultâneo e com dois pontos centrais para controlo do volume e compensação das baixas frequências.

A figura 2.25 mostra um potenciômetro e o seu símbolo elétrico.

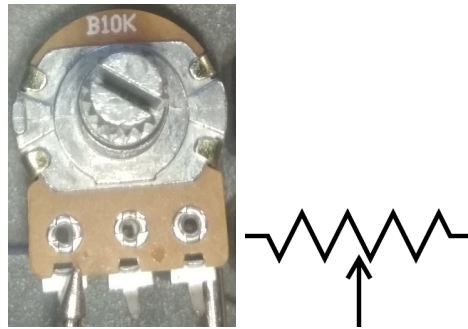


Figura 2.25: Potenciômetro logarítmico de 10k ohm (esquerda), e o seu símbolo elétrico (direita).

Se α for um coeficiente que indique a posição do veio no corpo do potenciômetro em relação a um dos extremos, o seu circuito equivalente corresponde a duas resistências em série com os valores:

$$R_1 = \alpha R_{pot} \quad (2.1)$$

$$R_2 = (1 - \alpha) R_{pot} \quad (2.2)$$

onde a resistência resultante $R_1 + R_2 = R_{pot}$. Esta montagem é idêntica ao divisor de tensão.² Onde a tensão de saída é dada por: $V_o = V_i \frac{\alpha R_p}{\alpha R_p + 1 - \alpha R_p} = \alpha V_i$.

Fazendo as ligações ilustradas na figura 2.26 e programando o Arduino com o Sketch da listagem 2.6 para demonstração do uso do potenciômetro, e realizar algumas experiências básicas como determinar a tensão no seu ponto médio, e qual a posição do veio do potenciômetro. O Sketch usa a instrução `sensorValue = analogRead(A0)`; vai permitir quantificar o valor da tensão na entrada analógica num valor inteiro entre 0 e 1023. O valor real da tensão é dado por: `voltage = sensorValue * (5.0 / 1023.0)`; Sabendo a tensão, a posição do veio do potenciômetro é dada por: $\alpha = V_i/V_o$, onde V_i é a tensão de alimentação no potenciômetro. De salientar que embora se assuma que estão presentes 5V, este valor deve ser medido usando outra entrada analógica.

Listagem 2.6: Listagem para demonstração do uso de um potenciômetro.

```
#define VCC 5
#define ADCMAXVAL 1023

class Pot {
private:
    byte pin;
```

²Nota: para realizar o circuito divisor de tensão deve-se ter atenção ao valor de resistência série do potenciômetro, para não exceder a potência máxima suportada pelo potenciômetro ($P = I^2 R$ ou $P = V^2/R$). Recomendam-se consumos abaixo dos 1/4W ($R_p > 1k\Omega$)

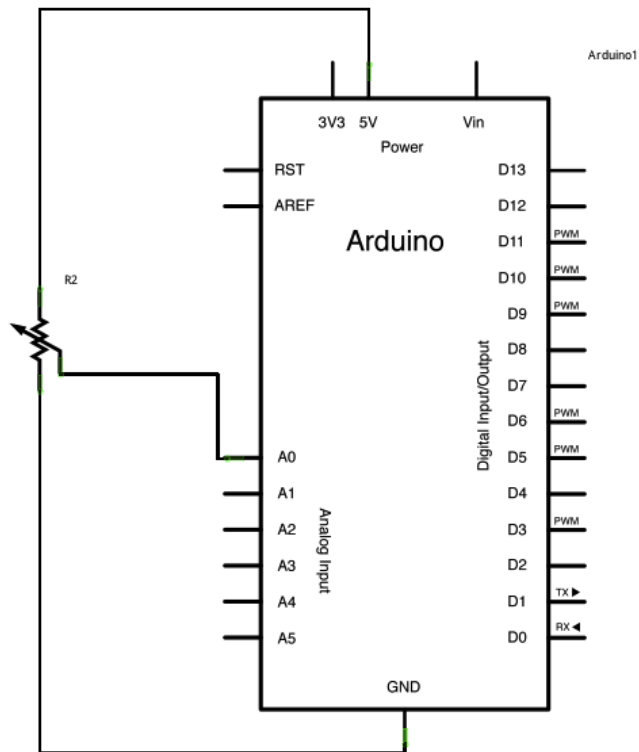


Figura 2.26: Esquema de ligações do potenciômetro ao Arduino Uno.

```

    unsigned int adcValue, maxValue, vcc;
public:
    Pot(byte pin = A0, int vcc = 5) {
        this->pin = pin;
        this->vcc = vcc;
        maxValue = ADCMAXVAL;
    }
    unsigned int getVal() {
        adcValue = analogRead(pin);
        return adcValue;
    }
    float getVolt() {
        //sensorValue = analogRead(pin);
        return (float) adcValue * vcc / maxValue;
    }
    void calibrate() {
        maxValue = 0;
        for(byte i = 0; i < 10; i++) {
            getVal();
            if(adcValue > maxValue) {
                maxValue = adcValue;
            }
            delay(500);
        }
    }
} ;

float voltage, position;
unsigned int potValue;
Pot p1 = Pot(A0, VCC);

```

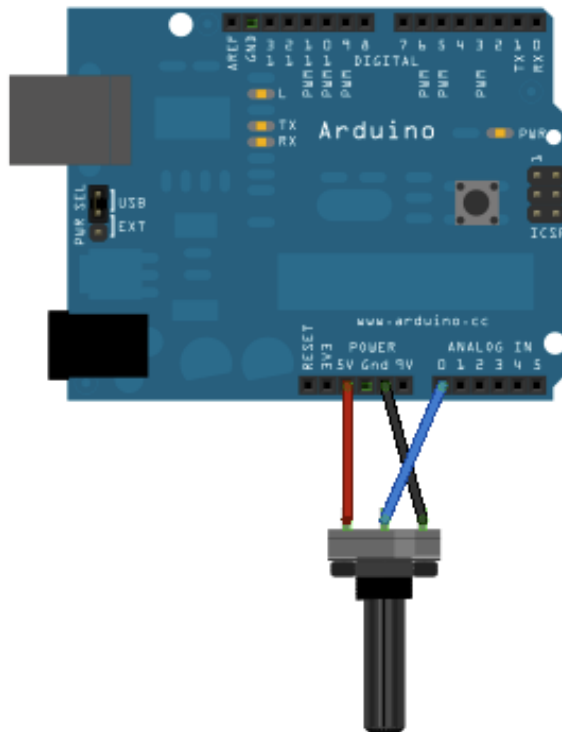


Figura 2.27: Montagem do potenciômetro num Arduino Uno.

```
void setup() {  
  Serial.begin(9600);  
  while(!Serial);  
  p1.calibrate();  
}  
  
// the loop routine runs over and over again forever:  
void loop() {  
  // read the input on analog pin 0:  
  potValue = p1.getVal();  
  // Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 5V):  
  
  Serial.print(p1.getVolt()); Serial.print(", ");  
  position = (float) potValue / ADCMAXVAL;  
  // print out the value you read:  
  Serial.print(100 * position); Serial.println("%");  
}
```

Na listagem 2.6, é apresentada a classe `Pot` para o objecto potenciômetro, que recebe como argumentos no construtor o identificador do pino, e a tensão de alimentação. Esta classe tem como atributos privados o valor lido no canal analógico, e o valor máximo possível. Como atributos públicos, tem os métodos para ler o valor do ADC, através do método `analogRead()`, e o valor da tensão.

2.6.2 Manípulo de 2 Eixos (Joystick)

Os manípulos, mais conhecidos como Joysticks, são normalmente usados para comandar o movimento em consolas de jogos ou robots. Eles são construídos por dois potenciômetros montados em posições ortogonais para capturar o movimento dos eixos horizontal e vertical (X,Y). O Joystick possui ainda molas que o fazem voltar à posição central de repouso. À medida que o manípulo abandona a posição central o valor das resistências vai variando. O potenciômetro possui ainda um botão de pressão no centro que pode ser usado como um botão de pressão apresentado anteriormente. A figura 2.28 à esquerda mostra o esquema de construção de um Joystick de dois eixos analógicos, onde cada existe um potenciômetro para cada eixo X e Y. À direita encontra-se o aspecto de um Joystick montado numa *breakout board*.

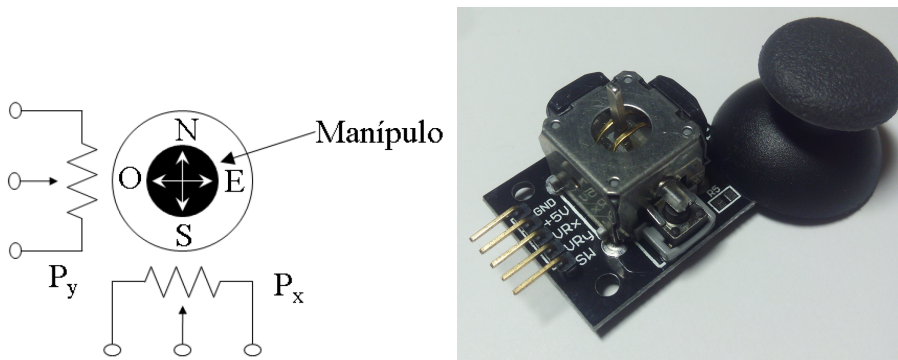


Figura 2.28: Modelo de construção de um Joystick de 2 eixos.

A listagem 2.7 demonstra a utilização do Joystick analógico.

Listagem 2.7: JoyStick.ino

```
// Arduino pin numbers
const int SW_pin = 2; // digital pin connected to switch output
const int X_pin = 0; // analog pin connected to X output
const int Y_pin = 1; // analog pin connected to Y output

void setup() {
  pinMode(SW_pin, INPUT);
  digitalWrite(SW_pin, HIGH);
  Serial.begin(115200);
}

void loop() {
  Serial.print("Switch: ");
  Serial.print(digitalRead(SW_pin));
  Serial.print("\n");
  Serial.print("X-axis: ");
  Serial.print(analogRead(X_pin));
  Serial.print("\n");
  Serial.print("Y-axis: ");
  Serial.println(analogRead(Y_pin));
  Serial.print("\n\n");
  delay(500);
}
```

2.7 Teclados

Os teclados são conjuntos de botões de pressões idênticos organizados numa única construção em linha ou matriz. Eles são usados para receber diferentes entradas de dados do utilizador e podem variar de uma teclado numérico como o de uma calculadora até um alfanumérico de um computador. Com o aumento do número de botões, o número de ligações cresce de tal forma que não existem sinais de E/S suficientes para os ligar num sistema embebido. Para minimizar o número de ligações existem técnicas no domínio digital, tal como a multiplexagem, e no domínio analógico o divisor de tensão.

2.7.1 Teclado Digital

O teclado digital funciona com base numa organização matricial (um pino do botão liga numa linha e o outro pino numa coluna) e na multiplexagem das linhas, onde apenas uma se encontra ativa em cada instante, e as colunas são avaliadas. Como o pressionar de um botão é mais lento do que o *varrimiento* do teclado, isto permite que seja usado sem o utilizador notar que cada conjunto de teclas apenas está a ser avaliado momentaneamente. Esta técnica permite reduzir o número de ligações necessárias no circuito para ler 16 botões, de 17 (16 entradas + GND) para 8 ligações (4 entradas + 4 saídas).

A figura 2.29 mostra dois teclados de 16 botões. A diferença entre eles é a sua construção. O teclado da esquerda é do tipo membrana, os botões são construídos com camadas de material isolante e condutor com uma pequena deformação na zona de cada botão. Este teclado não possui resistências integradas. O teclado da esquerda é construído com 16 botões de pressão convencionais soldados numa placa de circuito impresso.

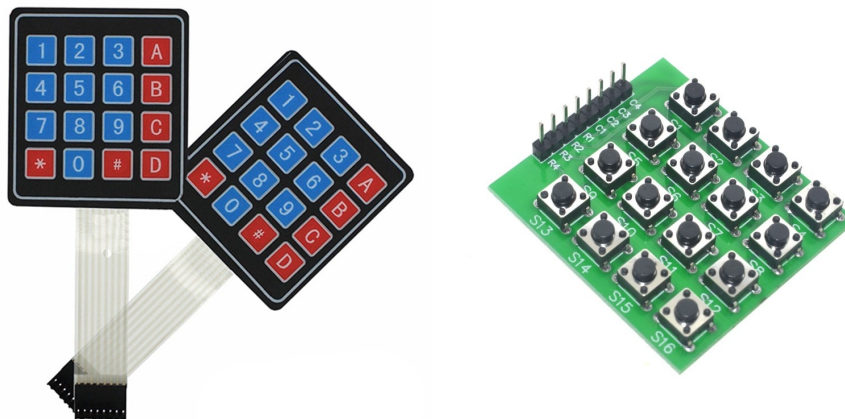


Figura 2.29: teclado de membrana

Quando nenhum botão está a ser avaliado, L1-L4 estão a VCC e os valores de tensão em C1-C4 é também VCC. Para fazer a avaliação da existência de um botão premido na primeira linha, esta é colocada a GND, e depois avaliada a tensão em C1-C4, caso exista algum sinal a GND, ou 0, significa que um dos botões S1-S4 foi premido. Depois desta avaliação L1 volta a VCC, e L2 passa a GND, e os valores de C1-C4 voltam a ser avaliados para determinar se algum dos botões S5-S8 foi premido. Este processo é repetido para todas as linhas L1-L4, e é constantemente repetido para dar a ilusão de que todas os botões estão a ser observados em simultâneo.

A figura 2.30 mostra as ligações de um teclado 4x4. Neste figura é possível observar que cada botão está ligado a uma coluna e a uma linha. Neste circuito as resistências R5 a R8 são usadas como *pull-up*, e dependendo dos pinos E/S usados podem ser desprezadas. No entanto, as resistências R1 a R4 são fundamentais para limitar a corrente que circula nos pinos de E/S. Por exemplo se S1 for pressionado, quando L1 for GND, C1 terá a tensão GND. Mas se S1 e S5 forem premidos em simultâneo, significa que eles estando ambos ligados a C1, estabelecem um curto-circuito entre L1 (GND) e L2 (VCC), e por consequência um curto-circuito dos pinos de E/S do sistema embebido! As resistências R1-R4 devem ter um valor inferior ao da resistência de pull-up para não provocar uma queda de tensão que não permita discriminar um botão premido, mas deve ser grande o suficiente para que a corrente máxima dos pinos de E/S não seja atingida. Recomendam-se valores na ordem da centena de Ohm ($5\text{ V}/(200 + 200\ \Omega) = 12,5\text{ mA}$).

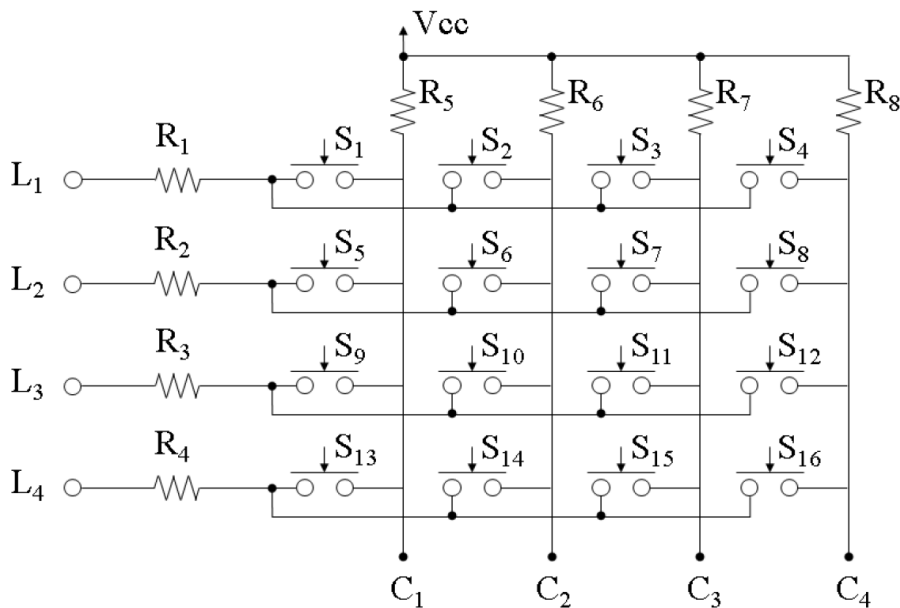


Figura 2.30: Esquema de ligações do teclado 4x4.

As listagens 2.8, 2.9, e 2.10, mostram o conteúdo dos ficheiros do Sketch

(.ino), definições de constantes, classe e protótipos de funções (.h) e implementação dos métodos (.cpp), respectivamente. A classe `Keypad` recebe como parâmetros no construtor: o número de linhas e colunas, um vector com os identificadores dos pinos e colunas, uso de *pull-up* interno nos pinos de entrada, e os níveis de sensibilidade/atuação (pressionar/largar).

No Sketch de demonstração o teclado procura constantemente por alterações no estado dos botões de pressão (`kb.scan()`), e sempre que houver um botão premido (`kb.keyHit()`), é obtido o identificador do botão (`kb.getKey()`). Para teclados onde os botões têm letras ou símbolos, pode-se fazer a conversão do código do botão com uma tabela (`keyLabels[]`).

No coração do funcionamento do teclado está o método `scan()`. Este método compara o estado atual de todos os botões do teclado com um valor anterior, de uma linha de cada vez. Sempre que houver uma alteração o tempo atual é registado juntamente com a indicação da mudança de estado desse botão.

Listagem 2.8: keypad4x4.ino

```
#include <Arduino.h>
#include "QueueArray.h"
#include "keypad.h"

byte rPins[] = {2, 3, 4, 5}; // pins for rows (top->bottom)
byte cPins[] = {6, 7, 8, 9}; // pins for columns (left->right)

QueueArray<byte> queue;

char keyLabels[] = { '1', '2', '3', 'A',
                    '4', '5', '6', 'B',
                    '7', '8', '9', 'C',
                    '*', '0', '#', 'D' };

Keypad kb = Keypad(4, 4, rPins, cPins, INPUT_PULLUP, 3);

void setup() {
  Serial.begin(9600);
  Serial.println("*** 4x4 Keypad Demo ***\n\n");
  kb.init();
  kb.setActionListener(kbActionPerformed);
  kb.printInfo();
}

void kbActionPerformed(byte key) {
  if(key != 15) {
    queue.enqueue(key);
  }
  else {
    while (!queue.isEmpty())
      Serial.print(keyLabels[queue.dequeue()]);
  }
}

void loop() {
  byte k;
  for(;;) {
    kb.scan();
    if(kb.keyHit()) {
      k = kb.getKey();
      if(k < 128) {
        Serial.print("Pressed: ");
      }
      else {

```

```

        Serial.print("Released: ");
        k = k & 127;
    }
    Serial.println(keyLabels[k]);
}
}
}

```

Listagem 2.9: keypad.h

```

#ifndef KEYPAD
#define KEYPAD

#include <arduino.h>

// max num of keys
#define KBMAXMATRIX 16

class Keypad {
private:
    byte rows, cols; // number of rows and columns
    byte *rPins, *cPins; // array with pin IDs
    byte inPinMode, modeKey; // pull-up & level detection
    byte keyValue[KBMAXMATRIX]; // key value
    bool keyChange[KBMAXMATRIX]; // key changed
    unsigned long keyChTime[KBMAXMATRIX]; // time of key changed
    bool keyHitFlag;
    int debounceTime;
    void (*actionListener)(byte); // process each received key
public:
    Keypad(); // default constructor
    Keypad(byte rows, byte cols, byte *rPins, byte *cPins, byte inPinMode, byte modeKey);
    void init();
    void scan();
    bool keyHit();
    byte getKey();
    void setActionListener(void (*al)(byte));
    void printInfo();
};

#endif

```

Listagem 2.10: keypad.cpp

```

#include "keypad.h"

Keypad::Keypad() {    this->rows = 0;    this->cols = 0;    }; // void

Keypad::Keypad(byte rows, byte cols, byte *rPins, byte *cPins, byte inPinMode, byte modeKey) {
    /* rows, cols : number of rows and cols
     * rPins, cPins : pins for rows/cols (top->bottom, left->write)
     * modePins : input or input-pull-up
     * modeKey : depress:1, press:2, both:3
     */
    this->rows = rows;
    this->cols = cols;
    this->rPins = rPins;
    this->cPins = cPins;
    this->inPinMode = inPinMode;
    this->modeKey = modeKey;
    debounceTime = 20;
    actionListener = NULL;
}

void Keypad::init() {
    byte i, j, k = 0;
    for (i = 0; i < rows; i++) {
        pinMode(rPins[i], inPinMode);
    }
    for (i = 0; i < cols; i++) {
        pinMode(cPins[i], OUTPUT);
    }
}

```

CAPÍTULO 2. SENSORES

```
        digitalWrite(cPins[i],HIGH);
        for (j = 0; j < rows; j++) {
            keyValue[k++] = HIGH; // keyb IDLE state
        }
    }
}

void Keypad::scan() {
    byte i, j, k, l;
    unsigned long now;
    for (i = 0; i < cols; i++) {
        digitalWrite(cPins[i], LOW);
        for (j = 0; j < rows; j++) {
            k = i + 4 * j;
            l = digitalRead(rPins[j]);

            if(keyValue[k] != 1) {
                keyValue[k] = 1;
                keyChange[k] = HIGH;
                keyChTime[k] = millis(); // register change time
            }
            else {
                now = millis();
                if((keyChange[k] == HIGH) && (now - keyChTime[k] > debounceTime)) {
                    if(((modeKey & 1) && (keyValue[k] == LOW)) || ((modeKey & 2) && (keyValue[k] == HIGH))) {
                        keyHitFlag = HIGH;
                        if(actionListener != NULL) {
                            actionListener(k);
                        } // if
                    } // if
                } // else
            }
            digitalWrite(cPins[i], HIGH);
        }
    }

    bool Keypad::keyHit() {
        return keyHitFlag;
    }

#ifdef 0
    bool Keypad::keyHit() {
        byte i;
        unsigned long now = millis();
        for (i = 0; i < cols * rows; i++) {
            if((keyChange[i] == HIGH) && (now - keyChTime[i] > debounceTime)) {
                if(((modeKey & 1) && (keyValue[i] == LOW)) || ((modeKey & 2) && (keyValue[i] == HIGH))) {
                    if(actionListener != NULL) {
                        actionListener(i);
                    }
                    return HIGH;
                }
            }
        }
        return LOW;
    }
#endif

    byte Keypad::getKey() {
        byte i;
        unsigned long now = millis();
        for (i = 0; i < cols * rows; i++) {
            if(keyChange[i] == HIGH) {
                if(now - keyChTime[i] > debounceTime) {
                    keyChange[i] = LOW;
                    keyHitFlag = LOW; // clear notification
                    if((modeKey & 1) && (keyValue[i] == LOW)) {
                        return i;
                    }
                }
            }
        }
    }
}
```

```

        if((modeKey & 2) && (keyValue[i] == HIGH)) {
            return i | 128;
        }
    }
}
return -1;
}

void Keypad::setActionListener(void (*al)(byte)) {
    actionListener = al;
}

void Keypad::printInfo() {
#ifdef 1
    byte i;
    Serial.println("Keypad info: ");
    Serial.println(" Num rows: " + String(rows));
    Serial.println(" Num columns: " + String(cols));
    for(i = 0; i < rows; i++) {
        Serial.println(" Pin r" + (String)i + ": " + (String)(*rPins+i));
    }
    for(i = 0; i < rows; i++) {
        Serial.println(" Pin c" + (String)i + ": " + (String)(*cPins+i));
    }
    switch(modeKey) {
        case 1: Serial.println(" Active Pressed");           break;
        case 2: Serial.println(" Active Released");          break;
        case 3: Serial.println(" Active Pressed+Released");   break;
        default: Serial.println(" Invalid Activation");
    }
    Serial.println(" Input Mode: " + (String) (inPinMode == INPUT ? "INPUT" : "INPUT PULL-UP"));
    Serial.println(" Debounce Time: " + (String)debounceTime + "ms");
    Serial.println("-----");
#endif
}

```

2.7.2 Teclado Analógico

O teclado analógico funciona com base no circuito divisor de tensão, onde cada botão de pressão está associado a um valor de resistência, provocando uma tensão à saída dependendo da botão premido. O teclado é ligado ao sistema embebido através de uma entrada analógica e o valor da tensão medido. Cada botão gera uma tensão diferente e criada uma correspondência entre a tensão medida na entrada analógica e o botão usando vários limiares de tensão.

A figura 2.31 mostra o circuito elétrico de duas realizações possíveis de um teclado analógico. O circuito da direita quando não tem um botão premido, tem na sua saída VCC. Quando um dos botões é premido a tensão de saída é: $V_s = VCC \times R_p / (R_p + R_s)$, onde R_p é a resistência de polarização e R_s a resistência associada a cada botão de pressão. Neste circuito cada botão terá um valor de resistência diferente. A escolha dos valores das resistências deve ser tal que maximize a diferença de tensões produzida pelos diferentes botões. Para minimizar a corrente consumida e ter imunidade ao ruído de sinais interferentes, recomendam-se valores de resistências na ordem da dezena de milhar de Ohm ($\times 10k\Omega$).

O circuito da esquerda corresponde a uma escada de resistências e permite usar resistências iguais, o que torna a sua construção mais simples e barata. A tensão na saída do teclado de cinco botões, quando o botão k é premido, é dada pela expressão: $V_s = VCC \times \sum_{i=k}^5 Ri / \sum_1^5 Ri = VCC * (5 - i + 1)/5$. Por exemplo, quando PB3 é pressionado: $V_s = VCC * (R3 + R4 + R5)/(R1 + R2 + R3 + R4 + R5)$. Na prática cada botão irá ter a sua tensão separada por $VCC/5$ V, pelo que o seu limiar de deteção deve ficar a meio, ou seja, $VCC/10$ V em torno de cada valor de tensão. A desvantagem deste circuito, em relação ao da direita, é o fato de existir sempre corrente a circular pelas resistências do divisor de tensão, fazendo aumentar o consumo do circuito.

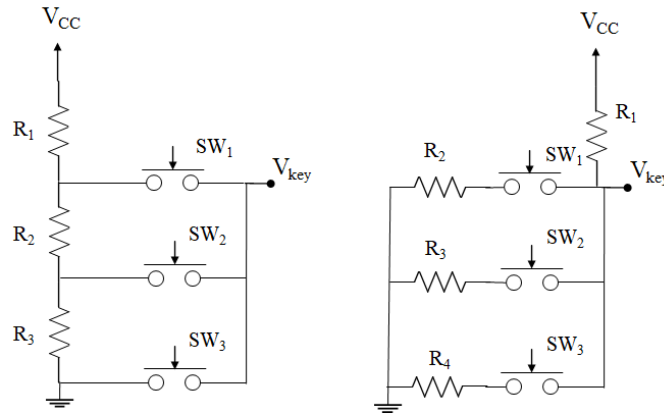


Figura 2.31: Diagramas de duas realizações de teclados analógicos.

A grande vantagem dos teclados analógicos é necessitarem apenas de 3 ligações independentemente do número de botões: VCC, GND e SAÍDA. A desvantagem dos teclados analógicos é serem sensíveis a variações na tensão de alimentação, tolerância dos componentes e degradação das resistências. Para teclados com muitos botões pode ser difícil discriminar as diferentes tensões dos botões corretamente. Com uma alimentação de 3,3 V e apenas 8 botões, vão haver intervalos de 0,41 V entre botões consecutivos. Quanto menor o intervalo de tensão maior a probabilidade do sistema permanecer fiável ao longo do tempo, onde a resistência parasita de contactos e componentes oxidados altera a resistência no circuito divisor de tensão.

A figura 2.32 à esquerda mostra um Shield Arduino de um mostrador alfanumérico que inclui um teclado analógico, e à direita o esquema do circuito do teclado de 5 botões. Neste circuito apresenta uma terceira alternativa de ligação dos botões. A tensão à saída do teclado é dada por: $V_s = VCC * \sum_{i=0}^k Ri / (R6 + Ri)$. Quando PB3 é pressionado a tensão à saída do teclado é igual a $VCC * (R1 + R2)/(R1 + R2 + R6) = 5 * (330 + 620)/(330 + 620 + 2000) = 5 * 950/2950 = 1,61$ V. As tensões para cada botão PB1 a PB5 são: 0, 0,71, 1,61, 2,47, e 3,62 V, com os limiares de tensão a média entre cada dois valores consecutivos: 0,35, 1,16, 2,04, 3,05 V.

Quando nenhum botão está premido a tensão é 5 V. As maiores desvantagens deste circuito são necessitar de resistências de valores diferentes e ter várias resistências a contribuir para a tensão de alguns botões, o que introduzem atenuações e interferências passíveis de alterar o valor de tensão esperado. A listagem ?? mostra um Sketch Arduino para demonstração do teclado analógico. Antes de identificar o botão premido, é calculada a média de duas amostras do valor da tensão analógica para reduzir o ruído.

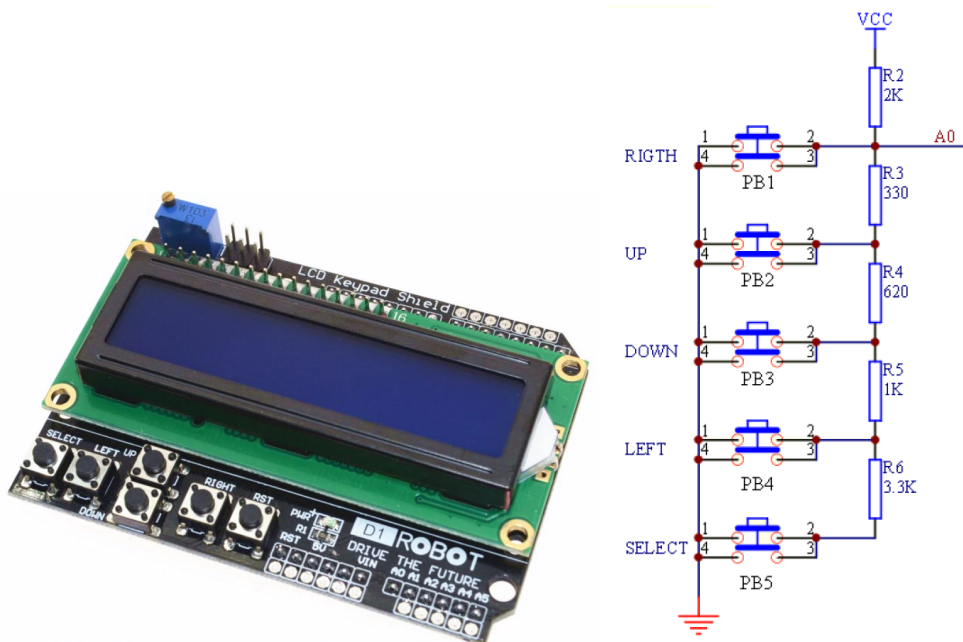


Figura 2.32: Placa de mostrador alfanumérico com teclado analógico.

Listagem 2.11: Demonstração da utilização de um teclado analógico.

```
int sensorPin = A0;    // select the input pin for the potentiometer
int ledPin = 13;      // select the pin for the LED
int sensorValue = 0;  // variable to store the value coming from the sensor

void setup() {
  // declare the ledPin as an OUTPUT:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  sensorValue = analogRead(sensorPin);
}
```

2.8 Luminosidade

Aplicações que necessitam de ter a luminosidade ambiente regulada são por exemplo cultivo e fotografia, ou onde é necessário haver comunicação entre

dois ou mais elementos de um sistema sem haver ligação elétrica entre eles. Existem vários sensores, tais como as foto-resistências, foto-diodos e foto-transistores, que capazes de aferir a luminosidade num determinado espaço e numa determinada gama do espectro luminoso (cor, infravermelhos e ultravioleta). Estes sensores são fabricados com materiais e tecnologias diferentes que lhe conferem características únicas, nomeadamente tempo de resposta, sensibilidade a sinais com frequências (ou comprimentos de onda) diferentes. As características mais importantes são: espectro de resposta (frequências da radiação a que são sensíveis), corrente no escuro, tempo de resposta e a energia mínima necessária para se observar variação de corrente na saída do sensor.

A medida da intensidade luminosa pode ser apresentada em W/m^2 ou W/Sr (em relação a um ângulo sólido). Para realizar uma medida fidedigna, qualquer um destes sensores necessita de calibração, que é feita com um optómetro, apresentado na figura 2.33. As seções seguintes aprofundam cada um destes tipos de sensores e como podem ser usados num sistema embebido.

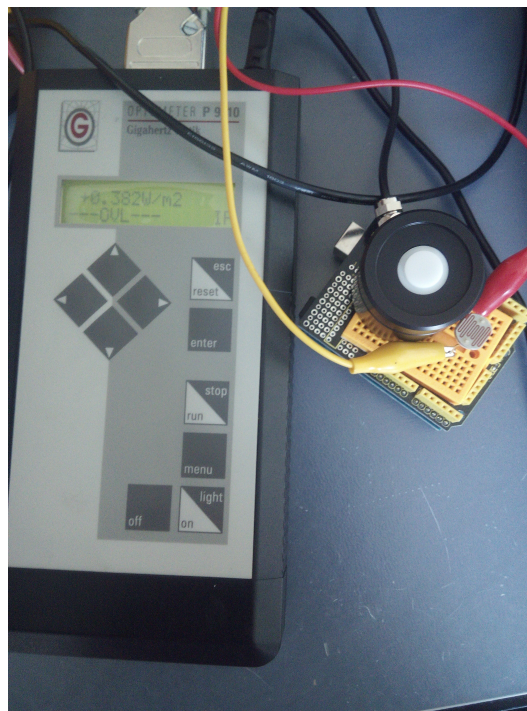


Figura 2.33: Optómetro

2.8.1 Foto-Resistência

A foto-resistência é conhecida como LDR, do inglês *Light Dependent Resistor* e tal como o nome indica é uma resistência que varia o seu valor consoante a

radiação luminosa que nela incide. A relação entre a intensidade da radiação luminosa e a resistência é dada pelo fabricante e pode ser encontrada no seu *datasheet*. Como a frequência máxima de funcionamento de uma LDR ronda os 10 Hz, elas não são recomendadas para transmissão de informação, sendo habitualmente aplicadas na medição da luminosidade de um ambiente que varia lentamente. Uma aplicação típica é o controlo automático da luminosidade num determinado espaço, ou barreiras luminosas para deteção de obstáculos em elevadores ou portões de garagem.

A figura 2.34, à esquerda, mostra o aspeto de uma LDR, e o seu símbolo elétrico à direita. Tal como os sensores resistivos apresentados anteriormente, a LDR necessita de ser integrada numa montagem tal como a do divisor de tensão para que seja produzida uma tensão analógica em função da radiação luminosa: $V_a = VCC * R_{ldr} / (R_p + R_{ldr})$. A figura 2.35 mostra o diagrama de ligações e a montagem em breadboard de uma LDR ligada a um Arduino Uno.

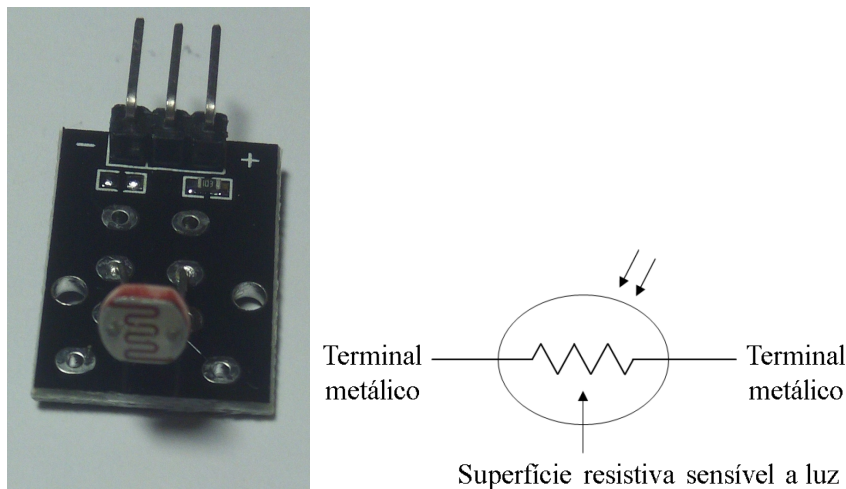


Figura 2.34: Aspeto e símbolo elétrico de uma LDR.

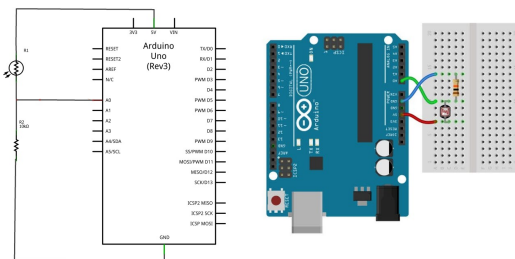


Figura 2.35: LDR soldada numa *breadboard*.

A listagem 2.12 mostra o conteúdo do Sketch de demonstração da medição de luminosidade com uma LDR. A classe LDR é responsável pela amostragem

do valor de tensão analógico, e conversão numa medida de luminosidade.

Listagem 2.12: Sketch de demonstração da LDR.

```
//KY018 Photo resistor module

int sensorPin = A5; // select the input pin for the potentiometer
int ledPin = 13; // select the pin for the LED
int sensorValue = 0; // variable to store the value coming from the sensor
void setup() {
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}
void loop() {
  sensorValue = analogRead(sensorPin);
  digitalWrite(ledPin, HIGH);
  delay(sensorValue);
  digitalWrite(ledPin, LOW);
  delay(sensorValue);
  Serial.println(sensorValue, DEC);
}
```

2.8.2 Fotodíodo e Fototransistor

O fotodiodo e o fototransistor são baseados nas tecnologias de semicondutores e tiram partido das estruturas internas dos diodos e dos transistores para criar componentes sensíveis à radiação luminosa.

Os fotodiodos e os fototransistores suportam frequências de funcionamento mais elevadas do que as das LDRs, tipicamente dezenas de MHz, sendo bastante aplicados em sistemas de comunicações tais como fibra-ótica e transmissão à distância via infra-vermelhos. Em termos de encapsulamento eles são construídos numa caixa plástica ou metálica com uma lente transparente à radiação para a qual eles devem operar, semelhantes aos LEDs. A figura 2.36 mostra os símbolos elétricos usados para os fotodiodos e para os fototransistores. A figura 2.37 mostra vários fotodiodos e fototransistores. Alguns deles têm aparência escura, para filtrar a radiação fora da gama dos infravermelhos.

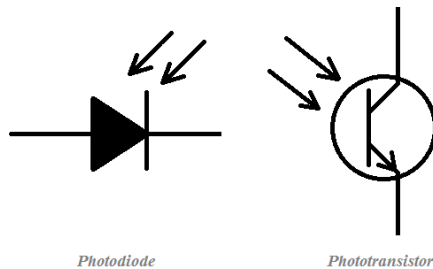


Figura 2.36: Símbolos elétricos do fotodiodo e do fototransistor.

A listagem 2.13 contém a demonstração da utilização de um fotodiodo para recepção de códigos de um comando à distância por infra-vermelhos, e subsequente decodificação.

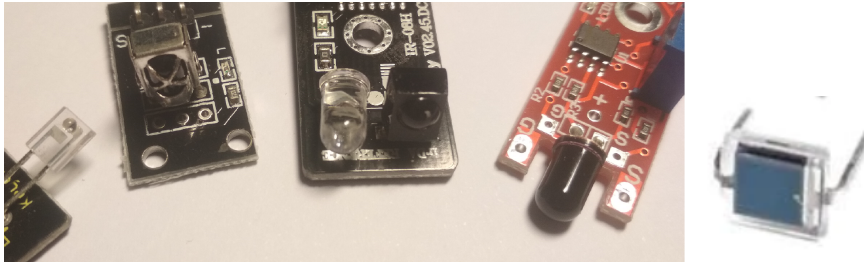


Figura 2.37: Vários sensores do tipo fotodiodo e fototransistor.

Listagem 2.13: IR_Receiver.ino

```

/*
 * IRremote: IRrecvDemo – demonstrates receiving IR codes with IRrecv
 * An IR detector/demodulator must be connected to the input RECV_PIN.
 * Version 0.1 July, 2009
 * Copyright 2009 Ken Shirriff
 * http://arcfn.com
 */

#include <IRremote.h>

int RECV_PIN = 11;

IRrecv irrecv(RECV_PIN);

decode_results results;

void setup()
{
  Serial.begin(9600);
  irrecv.enableIRIn(); // Start the receiver
}

void loop() {
  if (irrecv.decode(&results)) {
    Serial.println(results.value, HEX);
    irrecv.resume(); // Receive the next value
  }
}

```

2.8.3 Comparação das Diferentes Tecnologias

A escolha entre um fotodiodo e um fototransistor incide sobre os requisitos da aplicação. Os fotodiodos consomem menos corrente que os fototransistores mas são menos sensíveis (apx 100x). Os fotodiodos demoram menos tempo a responder à incidência de um fóton na sua estrutura, têm uma resposta linear e maior gama dinâmica que o fototransistor. Os fototransistores são normalmente usados em circuitos digitais, pois atingem a saturação mais rapidamente que os fotodiodos, sendo usados para comutação de circuitos isolados. A tabela 2.4 resume as principais vantagens e desvantagens de cada tecnologia.

Caraterísticas / Tecnologia	Fotodiodo	Fototransistor
Linearidade	✓	×
Frequência de funcionamento	+	-
Tempo de resposta	-	+
Corrente de funcionamento	-	+
Ângulo de abertura	+	-
Sensibilidade	-	+
Sensibilidade a ruído e interferência	-	+
Dependência com a temperatura	+	-
Estabilidade	-	+
Variação no processo de fabrico	-	+
Preço	-	+

Tabela 2.4: Comparação das vantagens e desvantagens entre fotodiodos e fototransistores.

2.9 Fogo

Uma chama pode ser detetada no local onde a combustão ocorre através da medição da temperatura, como nos esquentadores, ou à distância através da medida da radiação infravermelha emitida pela chama, usado em detetores de incêndio. Fontes de calor emitem radiação no espectro dos infravermelhos que podem ser detetadas com fotodiodos ou fototransistores.

Um exemplo de um sensor de chama à distância é o fototransistor YG-1006, que é construído num encapsulamento idêntico ao de um LED de 5 mm. Este sensor é sensível a radiação no espectro próximo dos infra-vermelhos, entre os 760 nm e 1100 nm com máximo nos 940 nm, e tem um ângulo de abertura de 60°. O YG-1006 necessita de circuito de um circuito de polarização com uma resistência de 10 k Ω entre o seu coletor e VCC. Quando na presença de uma chama, o transistor permite a passagem de corrente, entrando em condução e fazendo que o potencial do seu coletor se aproxime de GND. Na ausência de uma chama não existe corrente a percorrer a resistência de polarização e por consequência a tensão no coletor do fototransistor é próxima de VCC.

O YG-1006 foi testado numa placa idêntica à da figura 2.38, cujo esquema elétrico está na figura 2.39, que para além do circuito de polarização inclui também um AMPOP (LM393) que funciona como um circuito de comparação de tensão, para dar indicação do limiar de deteção haver sido ultrapassado através de um sinal digital (DO). O limiar é ajustado através de um potenciômetro. Esta placa suporta tensões de alimentação entre os 3,3 V e os 5 V. O esquema elétrico do circuito nesta placa encontra-se na figura 2.39 à direita. Esta placa possui quatro terminais: VCC, GND, AnalogOutput (AO), e DigitalOutput (DO). Para ligar a placa a um Arduino, para além da alimentação deve-se ligar a saída analógica (AO) a uma entrada analógica, por exemplo A0.

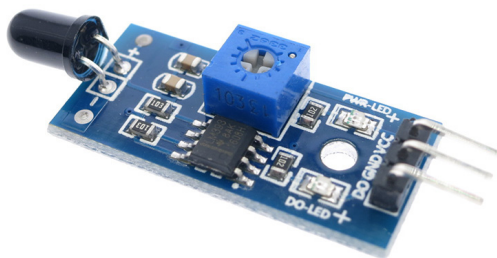


Figura 2.38: Placa de demonstração do sensor de chama.

Para realizar medidas com um Arduino foi usado o Sketch da listagem 2.15.

Listagem 2.14: Demonstração do sensor de chama.

CAPÍTULO 2. SENSORES

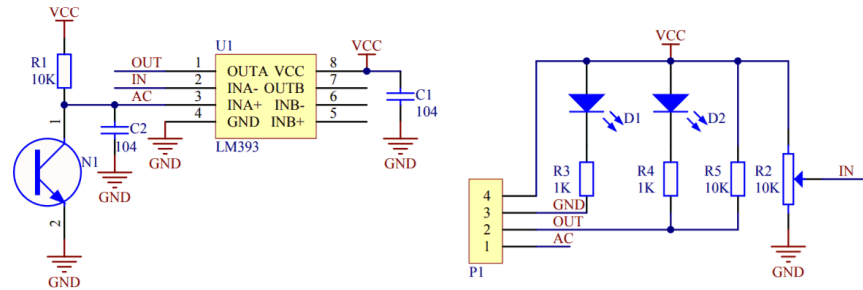


Figura 2.39: Esquema elétrico do circuito da placa de demonstração do sensor de chama.

```
// lowest and highest sensor readings:
const int sensorMin = 0;    // sensor minimum
const int sensorMax = 1024; // sensor maximum

void setup() {
  // initialize serial communication @ 9600 baud:
  Serial.begin(9600);
}

void loop() {
  // read the sensor on analog A0:
  int sensorReading = analogRead(A0);
  // map the sensor range (four options):
  // ex: 'long int map(long int, long int, long int, long int, long int)'
  int range = map(sensorReading, sensorMin, sensorMax, 0, 3);

  // range value:
  switch (range) {
    case 0: // A fire closer than 1.5 feet away.
      Serial.println("** Close Fire **");
      break;
    case 1: // A fire between 1-3 feet away.
      Serial.println("** Distant Fire **");
      break;
    case 2: // No fire detected.
      Serial.println("No Fire");
      break;
  }
  delay(1); // delay between reads
}
```

Foi realizado um ensaio ao sensor com uma vela a arder a distâncias entre os 10 cm e os 300 cm. A figura 2.40 mostra a evolução do nível de tensão em função da distância entre a chama da vela e o sensor. É evidente que o andamento é logarítmico devido à tecnologia semicondutora usada no sensor. As medidas próximas dos três metros foram realizadas próximas de uma parede, o que ajuda a refletir a radiação emitida pela chama da vela e a contribuir para o aumento da radiação detectada, por este motivo as medidas parecem indicar que a chama estava mais próxima do que ela estava posicionada. Cada ponto do gráfico representa o valor médio medido. As barras verticais em cada medida indicam os valores máximos e mínimos medidos derivados da oscilação da chama da vela.

As experiências com chama devem ser feitas em locais sem materiais ou substâncias inflamáveis próximos.

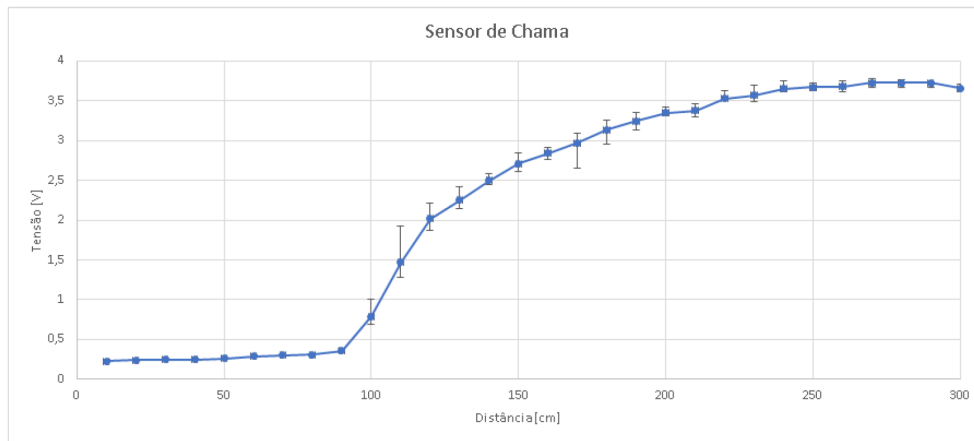


Figura 2.40: Gráfico com a relação entre a distância da chama da vela ao sensor e a tensão analógica.

2.10 Presença

A presença de uma pessoa ou animal a mover-se num determinado espaço pode ser detectada com um sensor passivo de infravermelhos PIR (*Passive Infra Red*). Este tipo de sensores é empregue em edifícios inteligentes, sistemas de alarme, vigilância, iluminação automáticos, e brinquedos. O princípio de funcionamento deste tipo de sensores assenta na deteção da variação da radiação infra-vermelha (calor) emitida por um corpo quente em deslocamento por um sensor de infra-vermelhos de alta sensibilidade ligado a um amplificador de elevado ganho. O sensor de presença é capaz de detetar níveis de radiação que o sensor de chama não consegue.

A figura 2.41 mostra um modelo simplificado de um sensor PIR, correspondente a um agregado de junções PN, como um fotodiodo de grande dimensão, e um transistor do tipo J-FET (elevado ganho e baixo ruído).

As caraterísticas mais importantes deste tipo de sensores são a distância (sensibilidade) e ângulo de deteção, tensão de alimentação, consumo de corrente em repouso e em funcionamento. O consumo de corrente deste tipo de sensores ronda os $60 \mu A$.

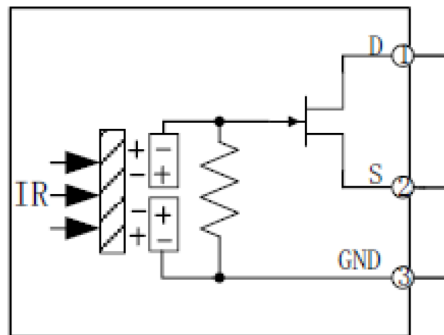


Figura 2.41: Modelo equivalente de um sensor PIR.

Dada as pequenas amplitudes de sinal envolvidas é necessário usar um circuito condicionador de sinal que permita distinguir a presença de um pessoa no sinal gerado. Para exemplificar o uso de um sensor PIR foram usados dois circuitos de demonstração: HC-SR505 e Groove PIR (SeedStudio), ambos ilustrados na figura 2.42. A tabela 2.5 resume as características de cada um destes circuitos.

A figura 2.43 mostra o esquema elétrico do circuito da placa breakout HC-SR505 para um sensor PIR. A figura 2.44 mostra os detalhes do circuito condicionador de sinal para sensores **PIR!** (**PIR!**). Este circuito integrado tem internamente um conjunto de **AMPOP!**s (**AMPOP!**s) usados como comparadores de tensão. Estes circuitos permitem também definir um período de tempo em que o sinal é mantido ativo, durante um período de tempo programável, para poder ser usado diretamente em controladores de

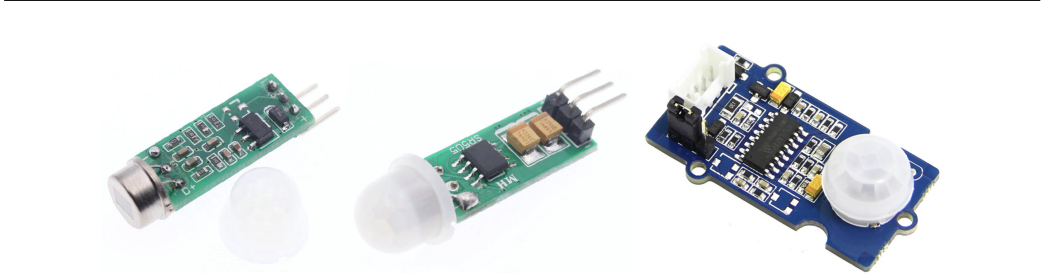


Figura 2.42: Placas breakout com sensores PIR, HC-SR505 à esquerda e centro, e Groove PIR à direita.

Caraterística	HC-SR505	Groove PIR
Tensão de funcionamento	4,5-20 V	3,3-5 V
Ângulo de detecção	100°	120°
Alcance	3 m	6 m
Pinos	1: VCC 2: OUT 3: GND	1: OUT 3: VCC 4: GND

Tabela 2.5: Resumo das caraterísticas dos circuitos com sensores PIR.

iluminação ou alarmes, sem necessidade de estarem ligados a um processador. O ajuste do tempo é feito através da introdução, ou variação, de uma resistência variável no circuito.

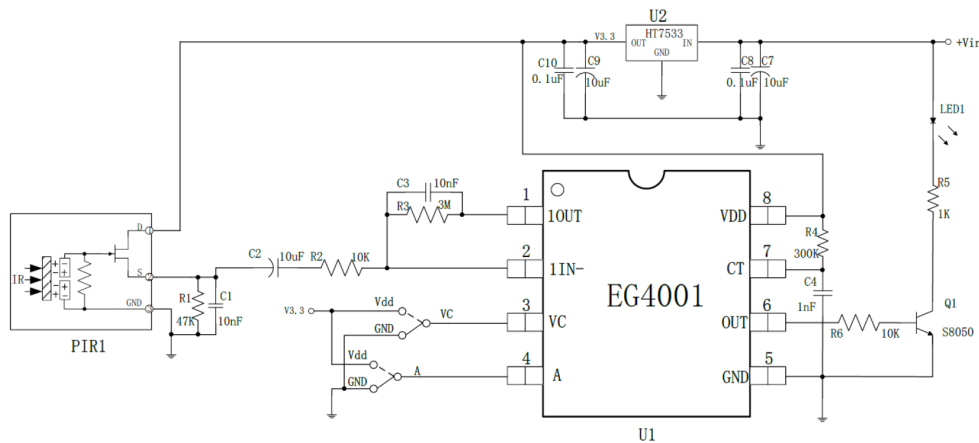


Figura 2.43: Esquema elétrico do circuito da placa breakout HC-SR505 para um sensor PIR.

Como a saída dos circuitos de condicionamento de sinal nas breakouts é binária, em termos de programação de um sistema embebido com um Arduino basta obter a leitura de um sinal de entrada digital. O sinal gerado por estes circuitos não necessita de circuito de *debounce*.

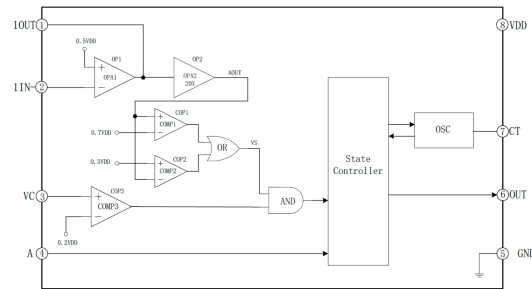


Figura 2.44: Diagrama simplificado do circuito integrado EG4001 para realiza o condicionamento de sinal de sensores PIR.

2.11 Vibração

Os sensores de movimento e choque abordados no início deste capítulo servem dar indicação binária (0/1) de que o corpo a ele acoplado havia abandonado a sua posição de repouso, após uma determinada força mínima ter sido aplicada no corpo. Aplicações tais como monitorização de estruturas, e instrumentos musicais eletrónicos é necessário caracterizar a força aplicada (intensidade e duração), caracterizando a sua amplitude e resposta em frequência.

O sensor piezoelétrico é construído através de material cristalino (p.ex. quartzo) e gera uma tensão aos seus terminais de acordo com a deformação temporária que a estrutura sofreu por acção de um esforço externo. Este efeito é reversível, ou seja, se for aplicada uma diferença de potencial no material este deforma-se, sendo amplamente aplicado como gerador de sons em pequenos aparelhos eletrónicos tais como mini-jogos portáteis e relógios. A figura 2.45 mostra o modelo de construção de um sensor piezoelétrico. A figura 2.46 mostra um sensor piezoelétrico dentro da caixa de proteção.

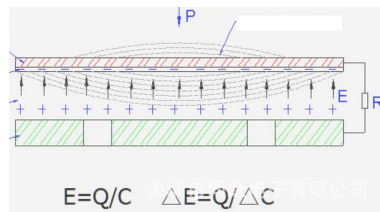


Figura 2.45: modelo do sensor piezoelétrico.



Figura 2.46: Sensor piezoelétrico.

As principais características dos sensores piezoelétricos são:

- elevada sensibilidade - deteta ligeiros toques no seu corpo,
- rigidez mecânica - não é facilmente deformado,
- dimensões reduzidas - servem em relógios de pulso,
- resistência elétrica elevada - faz com que a tensão aos seus terminais tenda para zero em pouco tempo,

CAPÍTULO 2. SENSORES

- sensível à temperatura
- necessita de circuito condicionador de sinal para ser ligado a um sistema embebido.

A figura 2.47 mostra o circuito para ligar um sensor piezoelétrico a um Arduino. Este sensor, embora seja gerador necessita que a tensão por ele gerada não exceda os limites tolerados pelo MCU do Arduino, sob consequência de o destruir. De acordo com o manual do MCU, os portos de entrada do Arduino admitem tensões entre -0,7 e VCC+0,3. Como a corrente gerada pelo sensor é muito reduzida, a limitação é feita apenas com um diodo zener, montado de forma que fique ativo no sentido direto ($V_d = 0,7$ V) quando o sensor gera uma tensão negativa, e no sentido inverso ($V_d = 4,7$ V).

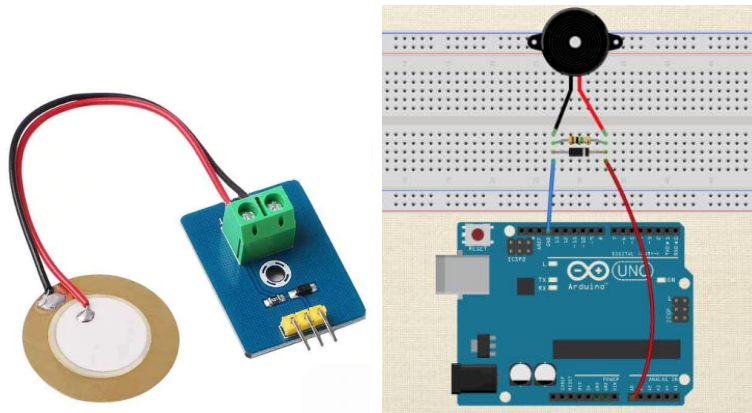


Figura 2.47: Ligação de um sensor piezoelétrico através de um ao circuito condicionador de sinal.

Para realizar medidas com um Arduino foi usado o Sketch da listagem 2.15.

Listagem 2.15: Demonstração do sensor de chama.

```
// lowest and highest sensor readings:
const int sensorMin = 0;    // sensor minimum
const int sensorMax = 1024; // sensor maximum

void setup() {
  // initialize serial communication @ 9600 baud:
  Serial.begin(9600);
}

void loop() {
  // read the sensor on analog A0:
  int sensorReading = analogRead(A0);
  // map the sensor range (four options):
  // ex: 'long int map(long int, long int, long int, long int)'
  int range = map(sensorReading, sensorMin, sensorMax, 0, 3);

  // range value:
  switch (range) {
    case 0: // A fire closer than 1.5 feet away.
      Serial.println("** Close Fire **");
      break;
    case 1: // A fire between 1-3 feet away.
```

```

    Serial.println("** Distant Fire **");
    break;
case 2:    // No fire detected.
    Serial.println("No Fire");
    break;
}
delay(1); // delay between reads
}

```

A figura ??ostra dois exemplos de sinais gerados pelo sensor piezoeletrico quando foi atingido pelos pêlos de um pincel e na outra pelo cabo de madeira do pincel.

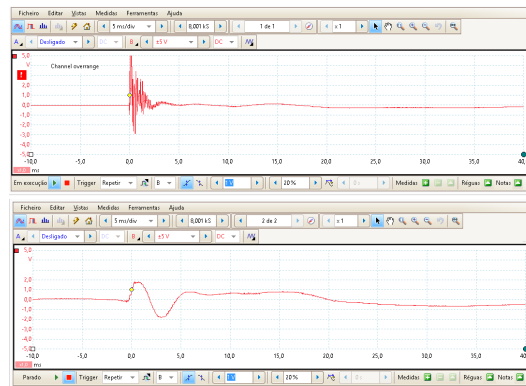


Figura 2.48: Exemplo com batidas de diferentes materiais num sensor piezoelétrico a um Arduino.

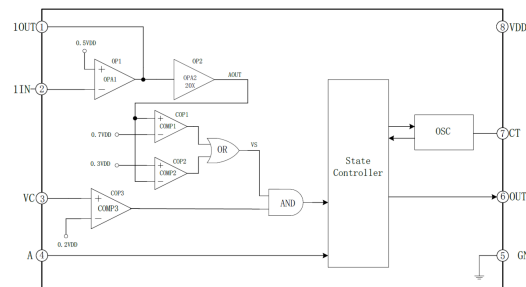


Figura 2.49: Ligação de um sensor piezoelétrico a um Arduino.

aplicação: instrumentos musicais: pickups guitarras acusticas, baterias electronicas.

2.12 Campo Magnético

O sensores de campo magnético servem para detetar a proximidade de um campo magnético na sua proximidade. Este tipo de sensores é útil para aplicações onde não seja possível estabelecer um contacto elétrico exposto, por exemplo em proximidade a ambientes sujos (poeiras, humidade), ou com materiais inflamáveis ou corrosivos, ou aplicados em peças em movimento oscilatório ou rotativo.

O *Reed Switch* apenas atua de forma binária em função da presença de um campo magnético, este tipo de sensores é capaz de detetar variações num campo magnético na sua proximidade. Os sensores de efeito de Hall podem fornecer um sinal de saída proporcional à proximidade do campo magnético ou binária após atingir um limiar. Estes sensores conseguem frequências de funcionamento superiores ao *Reed Switch*. Exemplos de sensores magnéticos são os de Efeito de Hall e as bobinas. Exemplos de aplicação destes sensores são: medição de deslocamentos e posição, detectores de proximidade e medida de espessuras, detecção das rotações de um motor ou rodas de um veículo, deteção de nível de um tanque de água ou combustível, medição de caudal de líquido, vibração das cordas de uma guitarra elétrica, e medição de um campo magnético gerado à volta de um condutor quando percorrido por uma corrente elétrica.

2.12.1 Efeito de Hall

O sensor de efeito de Hall é um componente semicondutor (estado sólido) capaz de gerar uma tensão aos terminais na presença de um campo magnético, com polarização perpendicular ao sensor. A figura 2.50 mostra à esquerda o princípio de funcionamento do sensor de efeito de Hall, é à direita o esquema de aplicação prática.

A figura 2.51, à esquerda, mostra o componente A3141, que inclui um sensor de efeito de Hall, e encontra-se montado numa placa de circuito impresso. À direita, encontra-se o esquema interno do componente. Dentro do seu encapsulamento, este componente tem na saída uma porta lógica do tipo Schmidt-Trigger ligada à base de um transistor bipolar com o emissor ligado a GND e coletor disponível na saída, para dar indicação da proximidade do campo magnético através de um sinal binário (0: campo magnético presente; 1: campo magnético ausente). Como o coletor encontra-se aberto é necessário usar um *pull-up* ou ativar o *pull-up* interno do pino do Arduino.

A integração do sensor magnético depende da aplicação. Para aplicações em que o sensor é usado para indicar uma atividade sem requisitos temporais, este pode ficar ligado a pino de entrada digital. Contudo, para aplicações que exijam um tempo de resposta menor, tais como medir rotação de motores, o sensor deve ser ligado a pino de entrada de interrupção. O Sketch deve ser

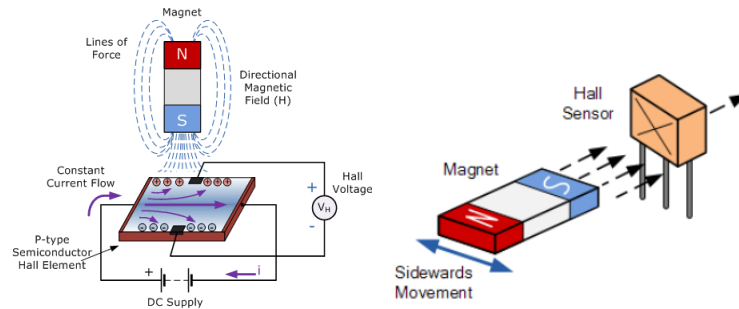


Figura 2.50: Modelo de funcionamento e aspeto de um sensor de efeito de Hall.

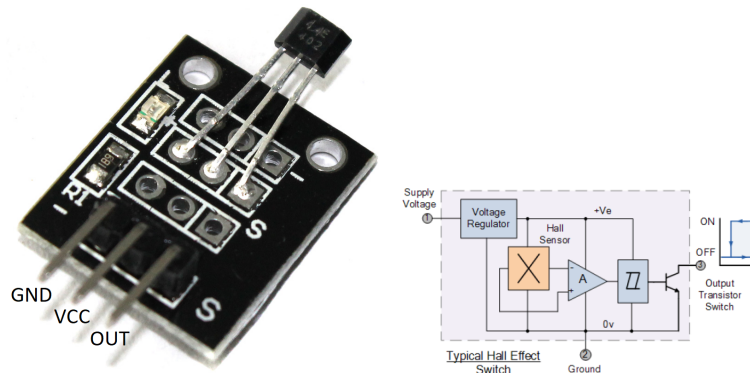


Figura 2.51: Modelo de funcionamento e aspeto de um sensor de efeito de Hall.

escrito de forma adequada. Isto é, ter a rotina de atendimento à interrupção o mais breve possível, de preferência limitando-se a dar indicação da ocorrência através da afetação de uma flag ou incrementando um contador. Ambas as abordagens já foram apresentadas anteriormente e podem ser reutilizadas para ensaiar este sensor.

2.12.2 Bobina

Uma bobina quando é percorrida por uma corrente elétrica, produz um campo magnético à sua volta, sendo a base de construção do electro-íman. No entanto, o processo é reversível, ou seja, a presença de um campo magnético variável gera uma variação de corrente, e por consequência da Lei de Ohm, uma tensão aos seus terminais.

A bobina é construída com um núcleo de material ferro-magnético envolto numa em espiras de fio condutor isolado (verniz). Pode haver um filme fino de material a separar o núcleo das espiras. Em casos onde não é possível associar um íman ao objecto, tal como nas cordas da guitarra elétrica, insere-se um

ímã permanente no núcleo da bobina para induzir um campo magnético nas cordas. Como a bobina apenas é sensível às variações do campo magnético, o ímã não irá interferir na tensão gerada.

As maiores vantagens de se usar uma bobina são: alta sensibilidade mas pouco sensíveis a variações de humidade e temperatura, baixa complexidade, e pouca carga mecânica. No entanto, em casos onde exista interferência de campos magnéticos parasitas é necessário blindar a bobina.

A característica que caracteriza uma bobina é a sua indutância. A indutância define a amplitude do campo magnético induzido por uma corrente elétrica. A indutância de uma bobina é dada pela expressão:

$$L = N^2 \times A \times \mu / l, \quad (2.3)$$

onde N é o número de espiras do enrolamento, A a área do núcleo, μ a permeabilidade magnética do material do núcleo e l o comprimento do núcleo.

A figura 2.52 mostra um conjunto de pickups de guitarra elétrica, à esquerda, e modelo de construção de um pickup genérico, à direita.

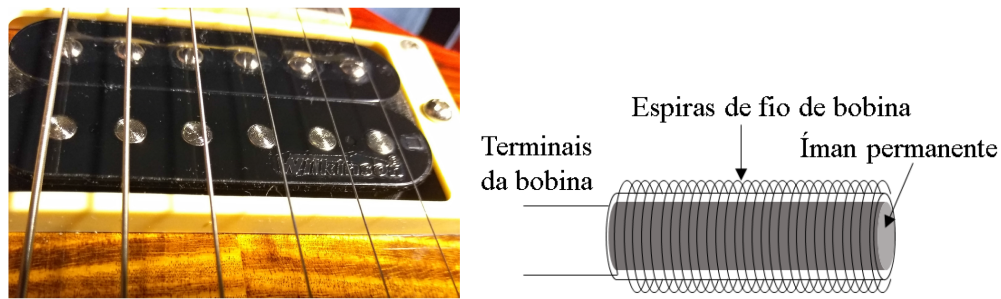


Figura 2.52: Pickup de guitarra elétrica (esquerda) e modelo de construção de um *pickup* (direita).

2.13 Consumo de Energia Elétrica

A maioria das aplicações para IoT são “as coisas inteligentes”, por exemplo edifícios, cidades e transportes inteligentes (*Smart Buildings*, *Smart Cities* e *Smart Transportation*). Uma das medidas que estes sistemas disponibilizam é o consumo energético de equipamentos instalados no local. O consumo energético, é dado pela potência absorvida e pode ser obtida de duas formas: a) medir a queda de tensão numa resistência de baixa impedância em série com o sistema a medir e usar a expressão $P = R.I^2$ onde I corresponde à corrente que percorre a resistência dada queda de tensão medida aos seus terminais $I = \frac{V}{R_{Shunt}}$, e b) medir o fluxo magnético num condutor com um enrolamento semelhante ao núcleo de um transformador. Devido às tolerâncias de fabrico dos componentes e características dos materiais, é necessário calibrar o circuito com o auxílio de um aparelho profissional para garantir a qualidade das medidas realizadas pelo circuito.

Atualmente já existem soluções de reguladores de tensão programáveis capazes de produzirem nas suas saídas várias tensões, e através de um canal de comunicação digital (PMBUS via I2C) dão indicação das tensões e correntes de entrada e saída do circuito alimentado.

2.13.1 Resistência em Série

A Lei de Ohm enuncia que uma corrente que percorre uma resistência provoca uma queda de tensão aos seus terminais. Esta é a motivação para aplicação deste método que se baseia na introdução de uma resistência de muito baixa impedância e alta potência, conhecida como *Shunt Resistor*, colocada em série com a alimentação do circuito, ilustrada na figura 2.53. A impedância de uma resistência de *shunt* é cerca de $m\Omega$. Quando uma resistência de $10\ m\Omega$ é colocada em série com um sistema que consome 1 A, vai haver uma queda de tensão máxima de 10 mV nos terminais da resistência, o que não tem impacto num circuito que funcione a 5 V. Como esta tensão é muito pequena para poder ser convertida em digital, usa-se um amplificador com um ganho de 500x para se poder usufruir dos 5 V de entrada máximos do ADC de um Arduino Uno. Existem componentes com circuitos amplificadores dedicados a sinais de medida de corrente (*current sense amplifier*) como por exemplo o INA170 ou INA190 da Texas Instruments, cujo diagrama de blocos funcional é mostrado na figura 2.54. Este tipo de amplificadores destaca-se por permitirem medidas de tensões na sua entrada superiores à sua tensão de alimentação, e terem uma elevada rejeição à tensão de modo comum, *Common Mode Rejection Ratio* (CMRR).

É necessário ter atenção ao dimensionamento do circuito para que a queda de tensão imposta pela resistência não baixe significativamente a tensão de alimentação do circuito. Por exemplo um circuito alimentado a 5 V com uma

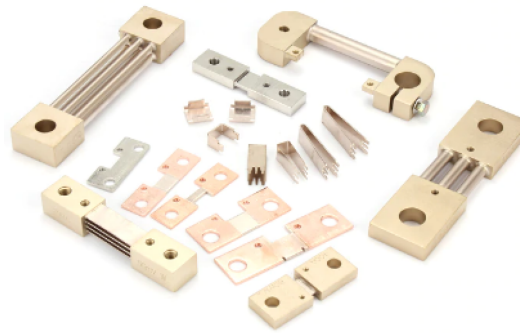


Figura 2.53: Resistência de shunt usada para medir correntes em circuitos DC.

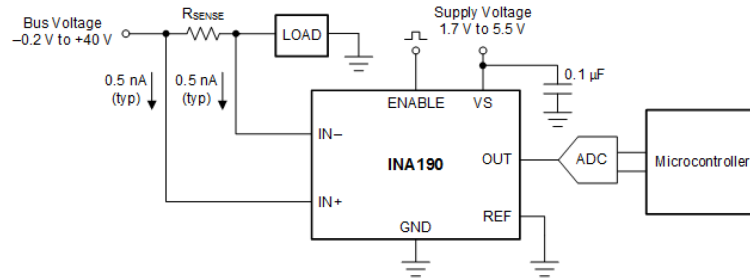


Figura 2.54: Diagrama de blocos funcional do INA190 da Texas Instruments. Tirado do manual de referência do componente.

resistência em série de $1\ \Omega$ percorrida por $1\ \text{A}$ vai produzir $1\ \text{V}$ de queda de tensão, ou seja, o circuito passa a ser alimentado apenas a $4\ \text{V}$. É necessário também ter atenção à potência dissipada no circuito. Outro parâmetro a ter em conta é a potência máxima dissipada pela resistência, que não deve ser excedida e no funcionamento do circuito é igual a $P = R \cdot I^2\ [\text{W}]$. Este método é mais adequado para medições em CC.

2.13.2 Fluxo Magnético

Quando um condutor é percorrido por uma corrente elétrica alternada ele radia à sua volta um campo eletromagnético que varia no tempo. A densidade da corrente (A/m^2) é proporcional ao rotacional do campo magnético (∇B), de acordo com a Lei de Maxwell-Ampère³. As bobinas têm a capacidade de “traduzir” o fluxo magnético numa tensão, usando o mesmo princípio de funcionamento dos transformadores usando um dos condutores que alimenta o circuito. A grande limitação é que apenas funciona em CA. A maior vantagem é ser possível medir a corrente sem intercalar novos componentes no circuito nem fazer derivações.

³Ver as Leis de Maxwell.

2.13. CONSUMO DE ENERGIA ELÉTRICA

A potência total consumida, ou aparente, por um equipamento em CA é dada pela soma da potência útil com a potência reativa. A potência útil e reativa resultam do desfasamento entre a tensão e a corrente introduzido por bobinas e condensadores dos equipamentos, e podem ser comparados a um copo de cerveja, ilustrado na figura 2.55. A potência útil corresponde à potência gasta pelo equipamento na realização de trabalho (útil) em que o equipamento consome o máximo de CA no instante em que a tensão também é máxima. A energia consumida é o integral, ou somatório, da potência consumida ao longo do tempo (kWh).

Para realizar a análise da potência consumida existem circuitos especializados em realizar esses cálculos e fornecem as leituras numa interface possível de ligar a um sistema embebido, por exemplo um porto série assíncrono. A figura 2.56 mostra um medidor de potência para a rede elétrica (esquerda), e o esquema de ligações à carga a medir e ao Arduino. O medidor é ligado à rede elétrica para se alimentar e para poder medir a tensão. Existe também uma ligação para o sensor de campo magnético que deve envolver um dos condutores que alimenta a carga. Este medidor produz uma medida por segundo e comunica com o Arduino a 9600 baud através de uma interface digital série assíncrona com um isolador no interior do medidor (via opto-acoplador) dos sinais TX e RX para evitar que caso aconteçam acidentes com a eletricidade da rede elétrica eles não se propaguem para o sistema embebido.

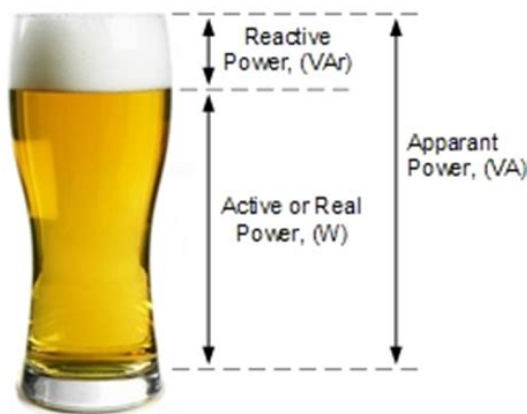


Figura 2.55: Analogia entre um copo de cerveja e as componentes de potência em CA.

O medidor possui um protocolo com mensagens de dimensão fixa (7 bytes) com os seguintes formatos:

Comandos - Tipo de comando (1 byte) + Endereço (4 bytes) + Dados (1 byte) + Controle (1 byte)

Respostas - Tipo de resposta (1 byte) + Dados (5 bytes) + Controle (1 byte)

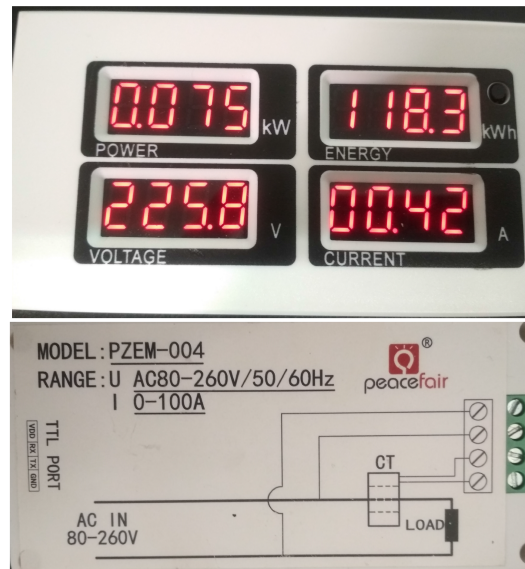


Figura 2.56: Medidor de potência para a rede elétrica (esquerda) e esquema de ligações para ligar a carga a medir e a interface série assíncrona para ligação a um Arduino.

Onde o tipo de comando e resposta corresponde a configurações e às diferentes medidas, resumido na tabela 2.6. Os quatro bytes do endereço têm o valor constante 192.168.1.1 (0xC0, 0xA8, 0x01, 0x01). Apenas o comando que define o alarme para o limite máximo de potência tem um argumento, e que é o valor máximo de potência em kW. O byte de controle serve para verificar a integridade da mensagem, e é obtido somando o valor de dos restantes bytes da mensagem. Se o valor calculado for diferente daquele que consta numa mensagem, a mensagem deve ser descartada.

Comando	Descrição	Resposta	Descrição
0xB0	Medir a tensão	0xA0	Medida da tensão (V)
0xB1	Medir a corrente	0xA1	Medida da corrente (A)
0xB2	Medir a potência	0xA2	Medida da potência (kW)
0xB3	Medir a energia	0xA3	Medida da energia
0xB4	Configurar o endereço	0xA4	Confirmação da configuração
0xB5	Configurar o limiar de potência máxima	0xA5	Confirmação da configuração

Tabela 2.6: Resumo dos valores dos comandos e respostas para o medidor de energia elétrica PZEM004.

Antes de ser possível obter qualquer leitura é necessário configurar o medidor, neste caso envolve configurar um endereço IP (192.168.1.1). O medidor

usa este endereço para se identificar. Comandos enviados para outro endereço IP não são observados. Para obter a leitura da uma medida o Arduino tem de enviar um pedido para cada medida.

Uma limitação deste medidor é realizar uma medida de cada vez, ou seja, ele não é capaz de realizar as 3 medidas em simultâneo. Isto significa que se for pedida a tensão, depois a corrente e depois a potencia, estas medidas não pertencem ao mesmo instantes, o que num equipamento que varia o seu consumo ao longo do tempo, impossibilita de estabelecer a relação entre as medidas.

A listagem 2.16 mostra o Sketch para medir os valores de tensão, corrente e potência do medidor de potência e enviar para o traçador gráfico do Arduino para analisar o consumo. O exemplo da figura 2.59 mostra o consumo de um computador pessoal ao longo do tempo onde várias aplicações vão sendo executadas. Neste gráfico, os pontos com maior consumo correspondem a calculo mais intensivo realizado pelo computador.

Nesta listagem encontra-se declarada a classe PZEM004 que recebe como parâmetros do construtor os pinos da interface série assíncrona a realizar em software, neste caso ligada aos pinos 2 (RX) e 3 (TX). Esta classe tem ainda os métodos públicos: `getVoltage()`, `getCurrent()`, `getPower()`, `getEnergy()`, `setAlarm()`, e `setAddress()` e que são usados para enviar os pedidos ao medidor e obter os valores das medidas. Os métodos privados mais relevantes são o `cmd(...)` e o `resp(...)` pois são eles que tratam da interacção com o medidor.

Listagem 2.16: Listagem do Sketch para ler as medidas do consumo de energia com um medidor PZEM004

```
#include <SoftwareSerial.h>

#define PZEMRESP_VOLTAGE      0xA0
#define PZEMRESP_CURRENT      0xA1
#define PZEMRESP_POWER        0xA2
#define PZEMRESP_ENERGY        0xA3
#define PZEMRESP_SET_ADDRESS   0xA4
#define PZEMRESP_POWERALARM    0xA5

#define PZEMRQST_VOLTAGE      0xB0
#define PZEMRQST_CURRENT      0xB1
#define PZEMRQST_POWER        0xB2
#define PZEMRQST_ENERGY        0xB3

#define PZEMRQST_SET_ADDRESS   0xB4
#define PZEMPOWERALARM        0xB5

#define PZEMFRAME_SIZE        7
#define PZEMRESP_DATA_SIZE    (PZEMFRAME_SIZE - 2)

#define PZEMBAUDRATE 9600
#define PZEMTIMEOUT 1000

// default address: 192.168.1.1
const byte pzemAddr[] = { 0xB4, 0xC0, 0xA8, 0x01, 0x01, 0x00, 0x1E};

// PZEMCommand: byte command, byte addr[4], byte data, byte crc;
```

```
class PZEM004
{
private:
    byte i, txBuf[PZEMFRAME.SIZE], rxBuf[PZEMFRAME.SIZE];
    unsigned long timeout;
    SoftwareSerial* SwSerial;
    void cmd(byte cmd, byte data = 0);
    bool resp(byte resp, byte *data = 0);
    byte crc(byte *buf) {
        byte crc = 0;
        for(i = 0; i < PZEMFRAME.SIZE - 1; i++) {
            crc += buf[i];
        }
        return crc;
    }

public:
    PZEM004(byte rxPin, byte txPin) {
        byte i;
        SwSerial = new SoftwareSerial(rxPin, txPin);
        SwSerial->begin(PZEMBAUDRATE);
        this->timeout = PZEMTIMEOUT;
        // copy address only once, no need to repeat
        for(i = 1; i < 5; i++) {
            txBuf[i] = pzemAddr[i];
        }

        bool setAddress() {
            // set default address: 192.168.1.1
            cmd(PZEMRQST.SET_ADDRESS);
            // expected 0xA4 0x00 0x00 0x00 0x00 0x00 0xA4
            return resp(PZEMRESP.SET_ADDRESS);
        }

        float getVoltage();
        float getCurrent();
        float getPower();
        float getEnergy();
        bool setAlarm(byte threshold);
    } ;

    float PZEM004::getVoltage()
    {
        cmd(PZEMRQST.VOLTAGE);
        if(!resp(PZEMRESP.VOLTAGE, rxBuf))
            return -1;
        return (rxBuf[0] << 8) + rxBuf[1] + (rxBuf[2] / 10.0);
    }

    float PZEM004::getCurrent()
    {
        cmd(PZEMRQST.CURRENT);
        if(!resp(PZEMRESP.CURRENT, rxBuf))
            return -1;
        return (rxBuf[0] << 8) + rxBuf[1] + (rxBuf[2] / 100.0);
    }

    float PZEM004::getPower()
    {
        cmd(PZEMRQST.POWER);
        if(!resp(PZEMRESP.POWER, rxBuf))
            return -1;
        return (rxBuf[0] << 8) + rxBuf[1];
    }

    float PZEM004::getEnergy()
    {
        cmd(PZEMRQST.ENERGY);
        if(!resp(PZEMRESP.ENERGY, rxBuf))
            return -1;
        return ((uint32_t)rxBuf[0] << 16) + ((uint16_t)rxBuf[1] << 8) + rxBuf[2];
    }
};
```

```

}

// parameter in kW
bool PZEM004::setAlarm(byte threshold)
{
    cmd(PZEMPOWERALARM, threshold);
    return resp(PZEMRESP.POWER.ALARM);
}

// send commands to the meter
void PZEM004::cmd(byte cmd, byte data)
{
    txBuf[0] = cmd;
    txBuf[5] = data;
    txBuf[6] = crc(txBuf);
    SwSerial->write(txBuf, PZEMFRAME.SIZE);
}

// get responses from the meter
bool PZEM004::resp(byte resp, byte *data = 0)
{
    byte c;
    byte len = 0, i;

    unsigned long rxTime = millis();
    while((len < PZEMFRAME.SIZE) && (millis() - rxTime < timeout))
    {
        if(SwSerial->available() > 0)
        {
            rxBuf[len++] = SwSerial->read();
        }
    }

    // check for the valid answer
    if( (len != PZEMFRAME.SIZE) ||
        (rxBuf[6] != crc(rxBuf)) ||
        (rxBuf[0] != resp) ) {
        return false;
    }
    return true;
}

PZEM004 pzem(2,3); // sw uart pins (rx, tx)

void setup()
{
    // Arduino-Host Computer
    Serial.begin(9600);
    while (!Serial) ;
    Serial.println(".:Arduino-Power Meter:.");
}

void loop()
{
    float v, i, p, e;

    v = pzem.getVoltage();
    i = pzem.getCurrent();
    p = pzem.getPower();
    e = pzem.getEnergy();

    Serial.print(v); Serial.print(",");
    Serial.print(i); Serial.print(",");
    Serial.print(p); Serial.print(",");
    Serial.print(e); Serial.print("\n");
}

```

A figura 2.57 mostra o aparato realizado para testar o medidor ligando um computador como carga. Nesta figura o é visível o cabo de alimentação com um dos seus condutores a atravessar o sensor de campo magnético. O

CAPÍTULO 2. SENSORES

mesmo cabo serve para alimentar o medidor. Os detalhes das ligações do medidor ao Arduino são ilustradas na figura 2.58.

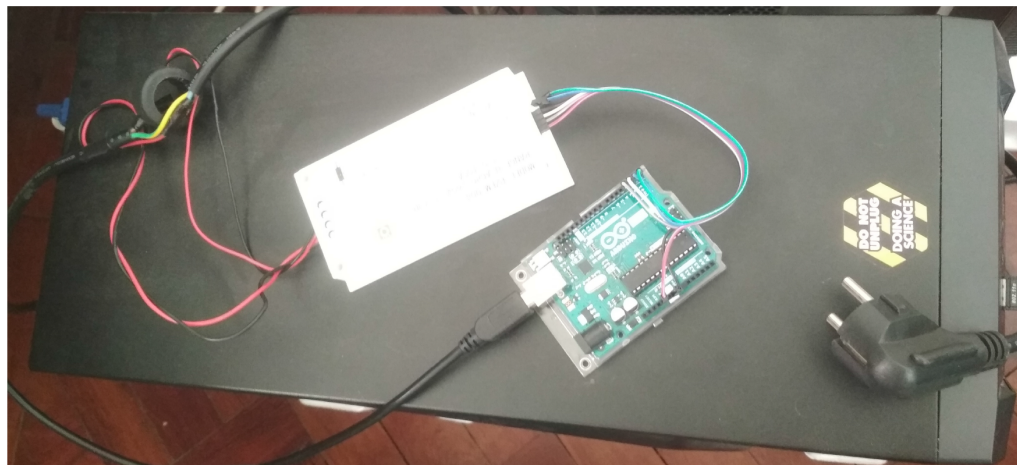


Figura 2.57: Aparato para medida da potência de um computador em funcionamento.

Figura 2.58: Medida da tensão, corrente e potência de um computador a executar diversos programas.

A figura 2.59 mostra o gráfico produzido pelo traçador gráfico para a tensão, corrente e potência de um computador durante a execução de diversos programas, como também os valores no terminal de texto. O traço do topo corresponde à corrente em milésima de Ampére (multiplicado por 1000x), o do meio à tensão em Volts, e o de baixo a potência em Watts.

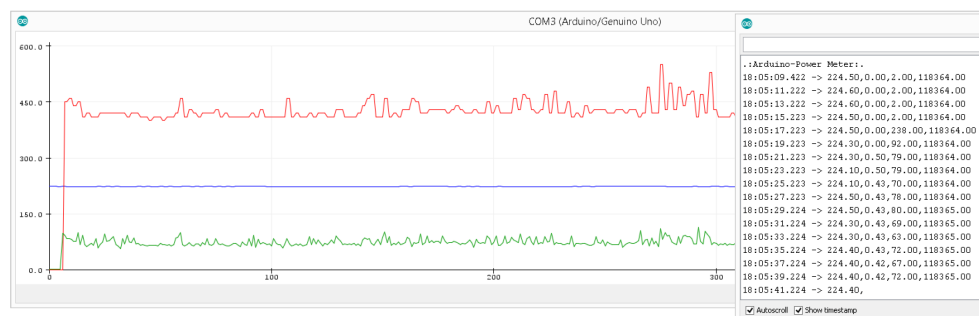


Figura 2.59: Medida da tensão, corrente e potência de um computador a executar diversos programas.

Deve haver cuidado em não tocar em nenhum contacto metálico num circuito ligado à rede elétrica, para evitar acidentes. Se houver necessidade de manusear o equipamento, usar peças em plástico e tocar apenas com uma das mãos.

2.14 Temperatura

Na monitorização de vários processos industriais, estruturas, equipamentos, ambientes, pessoas e animais, há necessidade de registar a sua temperatura. Em contextos diferentes existem três escalas de medida de temperatura: Célsius, Kelvin, Fahrenheit. A figura 2.60 mostra a comparação das três escalas. A escala em graus centígrados é usada em todo o Mundo com a excepção dos Estados Unidos da América, que usa a escala em graus Fahrenheit. A temperatura de 0°C corresponde à temperatura em que água congela e 100°C em que a água entra em ebulição. A escala Kelvin é usada em especificações técnicas, tal como a temperatura da cor em lâmpadas e equipamentos de vídeo, e em trabalho científico uma vez que zero kelvin, o zero absoluto, corresponde à temperatura em que as moléculas de água têm o mínimo de energia.

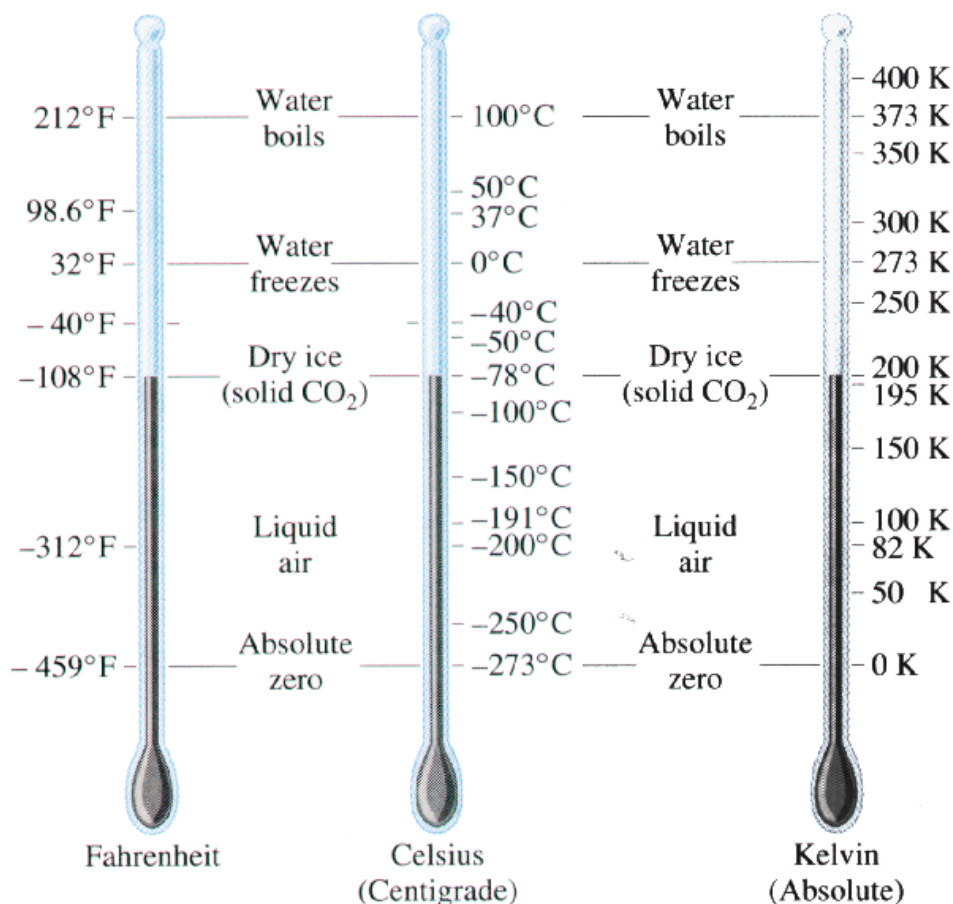


Figura 2.60: Escalas de temperatura em graus Célsius, graus Fahrenheit e Kelvin.

A conversão entre as diferentes escalas é conseguida de acordo com as

Conversão	Expressão matemática
Kelvin $\rightarrow$ Celsius	$C = K - 273.15$
Kelvin $\rightarrow$ Fahrenheit	$F = 1.8(K - 273) + 32$
Celsius $\rightarrow$ Fahrenheit	$F = 1.8(C) + 32$
Celsius $\rightarrow$ Kelvin	$K = C + 273.15$
Fahrenheit $\rightarrow$ Celsius	$C = (F - 32)/1.8$
Fahrenheit $\rightarrow$ Kelvin	$K = 5/9(F - 32) + 273.15$

Tabela 2.7: Conversão entre as diferentes escalas de temperatura.

expressões matemáticas da tabela 2.7.

A medição de temperatura é feita através de sensores que traduzem a variação de temperatura numa grandeza elétrica. Existem vários tipos de sensores de temperatura, adequados a diferentes excursões de temperatura e precisão. Os sensores de temperatura mais comuns são:

- Termopar - Um junção de dois metais diferentes gera uma pequena tensão nos seus terminais, em função da temperatura;
- Detector de temperatura de resistência, ou *Resistance Temperature Detector* (RTD) - Um metal condutor apresenta um valor de resistência que depende da sua temperatura.
- Termistor - Resistência dependente da temperatura. Pode ser do tipo NTC ou PTC:
 - Negative Temperature Coefficient (NTC) - A resistência decresce com o aumento da temperatura;
 - Positive Temperature Coefficient (PTC) - A resistência decresce com o aumento da temperatura
- Semicondutor - Circuito integrado que produz uma tensão em função da sua temperatura.

2.14.1 Termopar

Um termopar é um sensor constituído por dois metais condutores diferentes que quando em contacto geram uma pequena diferença de potencial nos seus terminais que varia em função da temperatura. Este fenómeno é conhecido como Efeito de Seebeck, descoberto em 1822, pelo físico estónio Thomas Johann Seebeck. A figura 2.61 mostra o modelo de um termopar do tipo K, à esquerda, composto por ligas metálicas de cromel e alumel, e à direita a fotografia de um termopar tipo K. O termopar tem duas junções: quente usada para medir a temperatura e fria que funciona como referência de temperatura.

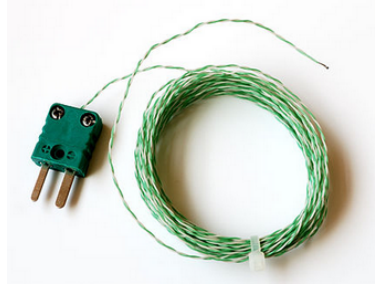
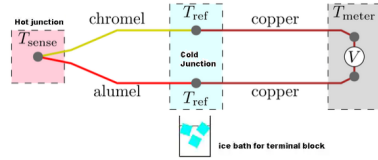


Figura 2.61: Modelo de construção e aspeto de um termopar tipo K.

Tipo	Metais	Gama de Temperaturas ($^{\circ}\text{C}$)	Gama de Tensões (V)
K	Cromel / Alumel	-270 a 1200	-6,458 mV a 48,838 mV
E	Cromel / Constantan	-270 a 1000	-9,835 mV a 76,373 mV
J	Ferro / Constantan	-210 a 760	-8,096 mV a 42,919 mV
N	Nicrosil / Nisil	-270 a 1300	-4,345 mV a 47,513 mV
T	Cobre / Constantan	-270 a 370	-6,258 mV a 20,872 mV
B	Platina / 29,6%Ródio-70,4%Platina	50 a 1820	-6,258 mV a 20,872 mV
R	Platina / 13%Ródio-87%Platina	-50 a 1768	-6,258 mV a 20,872 mV
S	Platina / 10%Ródio-90%Platina	-50 a 1768	-6,258 mV a 20,872 mV

Tabela 2.8: Características dos vários tipos de termopar

Existem diversas combinações de metais, e ligas metálicas, usados em termopares que estão caracterizados e são definidos como diferentes tipos de termopar B/E/J/K/N/R/S/T, sendo os tipos J e K os mais usados. Cada tipo de sensor é construído com quantidades exatas de metais, ou liga metálicas, e é usado para medir gamas de temperaturas e tolerâncias diferentes. A tabela 2.8 sumariza as temperaturas de funcionamento para os diferentes tipos de termopares.

A temperatura máxima de funcionamento de um termopar está limitada pela temperatura de fusão dos metais condutores. É de salientar que os metais condutores são sensíveis a agentes corrosivos, pelo que os metais devem ser isolados caso seja necessário medir a temperatura em ambientes corrosivos.

A ligação ao termopar deve respeitar a polaridade dos terminais, e o circuito deve limitar a corrente que percorre o sensor ao mínimo, caso contrário, a medida será afetada. Isto deve-se ao fato do Efeito de Thompson ser reversível, e à manifestação do efeito de Joule.

Como a tensão gerada pelo termopar ronda as dezenas de milivolt, variando dezenas de $\mu\text{V}/^{\circ}\text{C}$, é necessário introduzir um circuito condicionador de sinal entre o sensor e o sistema embebido de forma a que o sinal seja amplificado antes de ser amostrado pelo ADC. Existem no mercado circuitos condicionadores de sinal, baseados na montagem amplificador de instrumentação,

CAPÍTULO 2. SENSORES

que recebem o sinal do termopar e produzem uma tensão analógica proporcional à temperatura medida, ou o valor da temperatura em formato digital. Existem circuitos integrados analógicos, tal como o AD8494, que para além do circuito amplificador possuem também a compensação da junção fria, de modo a compensar variações na temperatura ambiente. A figura 2.62 mostra o diagrama de blocos do AD8495. Para ler a temperatura do termopar é necessário ligar a saída a uma entrada de um Arduino. Este circuito produz uma variação de tensão de $5 \text{ mV}/^{\circ}\text{C}$.

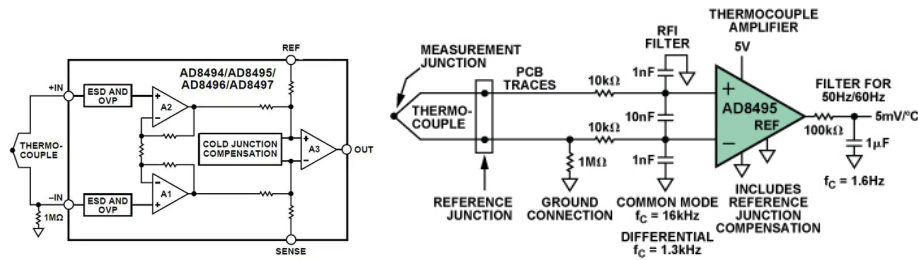


Figura 2.62: Diagrama de blocos do AD8495 e montagem para ligação de um termopar a uma entrada analógica de um Arduino.

Os circuitos integrados MAX6675 e MAX31855 tratam do condicionamento do termopar e da digitalização do valor da temperatura. A figura 2.63 o esquema elétrico do circuito para ligar o MAX6675 e a respetiva placa de circuito impresso com as fichas para ligação de um termopar tipo K e os pinos da interface digital série síncrona para ligar a um Arduino. O MAX6675 não recebe qualquer configuração do Arduino e por isso na sua interface série síncrona apenas tem os sinais: ativação (CS), relógio (CLK) e saída de dados (DO). Na interface digital é passada também a alimentação do circuito integrado (VCC e GND).

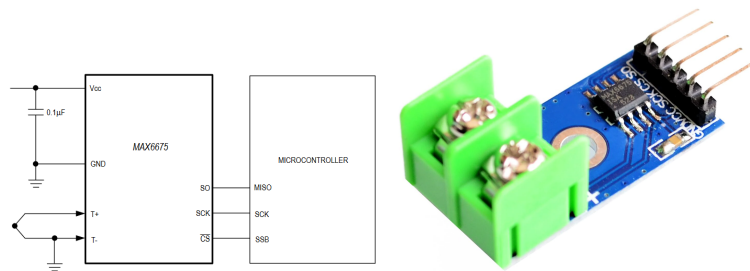


Figura 2.63: Modelo de construção e aspeto de um termopar tipo K.

A programação do Arduino pode ser feita usando a biblioteca disponível através dos gestor de bibliotecas do Arduino, através do nome “Adafruit MAX6675”. Esta biblioteca define a classe `thermocouple` que ao ser instanciada recebe como parâmetros os pinos da interface .

Esta biblioteca inclui dois exemplos, sendo um deles o “serialthermocouple” que permite obter a medida da temperatura do sensor e enviar o valor para o terminal série. Este exemplo, reproduzido na listagem ?? demonstra a leitura dos valores atuais da temperatura nas escalas Celsius e Fahrenheit usando os métodos `readCelsius()` e `readFahrenheit()`, respectivamente.

Listagem 2.17: Sketch exemplo `serialthermocouple`

```
// this example is public domain. enjoy!
// https://learn.adafruit.com/thermocouple/
#include "max6675.h"

int thermoDO = 4;
int thermoCS = 5;
int thermoCLK = 6;

MAX6675 thermocouple(thermoCLK, thermoCS, thermoDO);

void setup() {
  Serial.begin(9600);
  while(!Serial);
  Serial.println("MAX6675 test");
}

void loop() {
  // basic readout test, just print the current temp
  Serial.print("C = ");
  Serial.println(thermocouple.readCelsius());
  Serial.print("F = ");
  Serial.println(thermocouple.readFahrenheit());
  // For the MAX6675 to update, you must delay AT LEAST 250ms between reads!
  delay(1000);
}
```

2.14.2 RTD

As Resistive Temperature Detector (RTD), tal como o nome indica, são resistências que dependem da temperatura. A temperatura é obtida através da medição da resistência, uma vez que tem um valor conhecido consoante a sua temperatura.

Sendo a resistência a

2.14.3 NTC

$$Rt = R0.e^{\beta(\frac{1}{T}-\frac{1}{T0})} \quad (2.4)$$

$R0$ = resistência à temperatura de referência $T0$ = temperatura de referência (25°C) β coeficiente de temperatura característica do material (4000°K)

Aproximação por um modelo de 2 parametros: $Rt = Ae^{\beta/T}$ e para um determinado valor de resistência a temperatura é:

$$T = \frac{\beta}{\ln(\frac{Rt}{A})} \quad (2.5)$$

circuito condicionador de sinal: ref200 + opa4344

2.14.4 PTC

2.14.5 Semicondutor

Este tipo de sensor de temperatura é baseado na função de transferência, ou curva característica, do material semicondutor. Sabendo que um díodo é sensível às variações de temperatura, se a tensão de funcionamento se mantiver constante, é possível medir a temperatura em função da corrente consumida.

Os sensores de temperatura baseados em semicondutor oferecem menos excursão de temperatura, no entanto têm um tamanho bastante reduzido, e são facilmente colocados num circuito integrado. Este tipo de sensores é bastante utilizado para monitorizar a temperatura de processadores e MCUs.

Um sensor de temperatura baseado em material semicondutor pode ser construído com 3 diodos e uma resistência em série. Ao ser aplicada uma tensão nos extremos do circuito, uma corrente irá percorrer os 4 elementos, e provocar uma queda de tensão aos terminais da resistência. Como a corrente que percorre o diodo depende da temperatura, ter 3 diodos em série vai amplificar o efeito da dependência com a temperatura. De acordo com a lei de Ohm, a queda de tensão na resistência é proporcional à temperatura a que o material se encontra.

O LM35 é um sensor de temperatura de precisão capaz de medir temperaturas entre -55°C e 150°C com uma precisão de $0,5^{\circ}\text{C}$. Este sensor tem 2 pinos para alimentação (4-20V) e outro pino de saída que indica o valor da medida de temperatura sob a forma de tensão analógica. Este circuito integrado consome menos de $60\text{ }\mu\text{A}$ de corrente. O LM35 não necessita de qualquer calibração e na configuração mais básica, para medidas entre $2-150^{\circ}\text{C}$, o valor da tensão de saída corresponde a $10\text{mV}/^{\circ}\text{C}$.

2.14.6 Pilha Termoelétrica

A pilha termoelétrica é constituída por vários termopares encadeados em série, por exemplo a MLX90616 da Melexis,

2.14.7 Circuito Integrado Digital

Os sensores de temperatura digitais são construídos num chip de silício com um sensor de temperatura semiconductor e um controlador responsável por interpretar os comandos para fazer a leitura do valor de temperatura e comunicá-lo via a interface digital.

Um dos mais usuais é o Maxim DS18S20, que possui uma interface 1-Wire, e tem a possibilidade de funcionar com alimentação parasita, conseguindo funcionar apenas com a tensão do barramento.

Como a resistência de pull-up interna do ATmega328 é cerca de 20 – 50 $K\Omega$, é necessário usar uma resistência externa de 4,7 $k\Omega$.

A figura 2.64 mostra o aspeto do componente DS18S20, fornecido num encapsulamento TO-92. Os pinos devem ser ligados a GND, dados e VCC, da esquerda para a direita. O VCC poderá não ser usado caso se use a configuração de “alimentação parasita”.

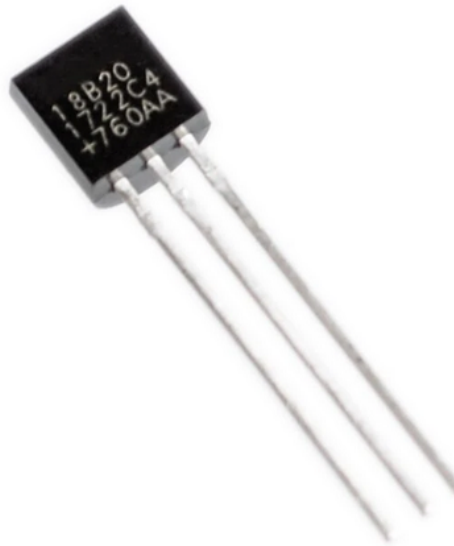


Figura 2.64: Sensor de temperatura digital DS18S20.

A listagem 2.18 exemplifica a programação e leitura dos valores de temperatura com DS18S20.

Listagem 2.18: Exemplo de leitura do valor da temperatura do sensor 1-wire DS18S20

```
#include <OneWire.h>

// DS18S20 Temperature chip i/o
OneWire ds(10); // on pin 10

void setup(void) {
  // initialize inputs/outputs
  // start serial port
```

```
    Serial.begin(9600);
}

void loop(void) {
    byte i;
    byte present = 0;
    byte data[12];
    byte addr[8];

    ds.reset_search();
    if ( !ds.search(addr) ) {
        Serial.print("No more addresses.\n");
        ds.reset_search();
        return;
    }

    Serial.print("R=");
    for( i = 0; i < 8; i++) {
        Serial.print(addr[i], HEX);
        Serial.print(" ");
    }

    if ( OneWire::crc8( addr, 7) != addr[7] ) {
        Serial.print("CRC is not valid!\n");
        return;
    }

    if ( addr[0] == 0x10) {
        Serial.print("Device is a DS18S20 family device.\n");
    }
    else if ( addr[0] == 0x28) {
        Serial.print("Device is a DS18B20 family device.\n");
    }
    else {
        Serial.print("Device family is not recognized: 0x");
        Serial.println(addr[0],HEX);
        return;
    }

    ds.reset();
    ds.select(addr);
    ds.write(0x44,1);          // start conversion, with parasite power on at the end

    delay(1000);              // maybe 750ms is enough, maybe not
    // we might do a ds.depower() here, but the reset will take care of it.

    present = ds.reset();
    ds.select(addr);
    ds.write(0xBE);           // Read Scratchpad

    Serial.print("P=");
    Serial.print(present,HEX);
    Serial.print(" ");
    for ( i = 0; i < 9; i++) {           // we need 9 bytes
        data[i] = ds.read();
        Serial.print(data[i], HEX);
        Serial.print(" ");
    }
    Serial.print(" CRC=");
    Serial.print( OneWire::crc8( data, 8), HEX);
    Serial.println();
}
```

2.14.8 Comparação das Diferentes Tecnologias

A tecnologia mais adequada a usar para medir a temperatura depende dos requisitos impostos pela aplicação do sensor. A tabela 2.9 resume as principais vantagens e desvantagens de cada uma das tecnologias apresentadas

2.14. TEMPERATURA

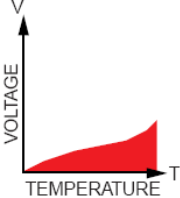
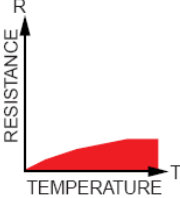
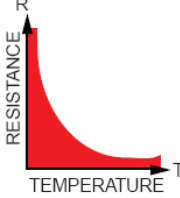
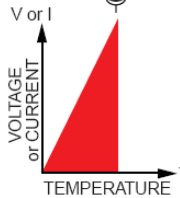
Tecnologia	Termopar 	RTD 	Termistor 	Silício 
				
Vantagens	Simples, barato, temperatura de funcionamento alargada	Estável, preciso, e mais linear que o termopar	Rápido, medição de resistência	Mais linear, fácil de ligar a um sistema embebido, barato
Desvantagens	Não é linear, muito baixa tensão, necessita de referência, pouco estável e sensível	Caro, necessita de uma fonte de corrente, suscetível de auto-aquecimento, valor de resistência baixo	Não é linear, temperatura de funcionamento reduzida, frágil, necessita de fonte de corrente, suscetível de auto-aquecimento	Mede temperaturas a baixo dos 200°C, necessita de alimentação, lento a realizar a medição, suscetível de auto-aquecimento

Tabela 2.9: Comparação das vantagens e desvantagens das diferentes tecnologias de sensores para medição de temperatura.

para medição de temperatura.

2.15 Humidade

2.15.1 Higrómetro Resistivo

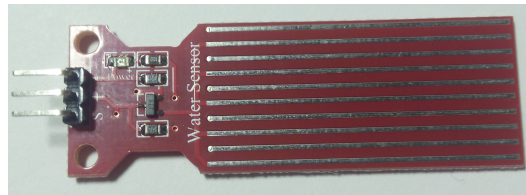


Figura 2.65: sensor líquidos.

2.15.2 Higrómetro Capacitivo

A humidade relativa corresponde à relação entre a quantidade de vapor de água existente no ar e o ponto de saturação, também conhecido como ponto de orvalho. O ponto de saturação depende da temperatura ambiente. O ar frio suporta menos humidade do que o ar quente (vapor).

A figura ilustra a variação do ponto de orvalho em função da temperatura do ar *vs* a temperatura da superfície. O ponto de orvalho ocorre sempre que a temperatura da superfície é igual ou inferior à temperatura do ar. Para valores de humidade relativa baixos é necessária que a superfície esteja a uma temperatura muito mais baixa do que a temperatura do ar.

usar montagem 555 + sensor capacitivo

SENSORES CAPACITIVOS

-a capacidade depende da disposição geométrica entre os condutores e do material dielétrico compreendido entre eles.

$$C = \epsilon \cdot A / d$$

$$\epsilon_0 = 8,85 \text{ pF/m}$$

$$\text{sandwich com 2 dielétricos: } C = A / (d_1 / \epsilon_1 + d_2 / \epsilon_2)$$

Limitações: -expressão válida desde que seja desprezado o efeito de dispersão na zona periférica -isolamento entre as placas alto e constante -capacidade interferências capacitivas -influência dos cabos de ligação

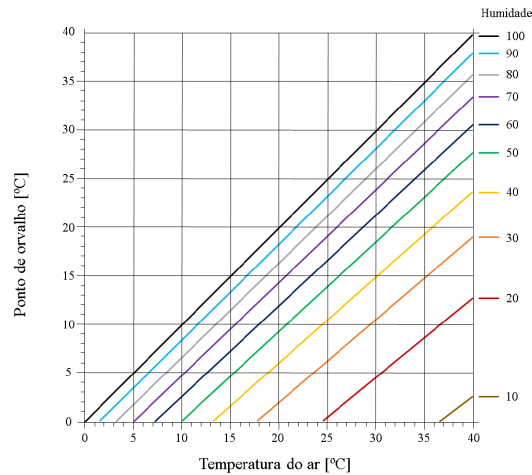


Figura 2.66: Variação do ponto de orvalho em função da temperatura do ar, da humidade relativa e da temperatura da superfície.

linearidade: é em relação a 'e' e a 'A' mas não a 'd'

vantagens: -como sensores de deslocamento: -erro mínimo por efeito de carga -não havendo contacto mecânico directo não há erros devido à fricção nem de histerese -não é necessária muita força para deslocar o elemento móvel -estabilidade e reprodutibilidade -elevada resolução -não produzem campos magnéticos ou eléctricos apreciáveis

-outras medidas: pressão, força, momento ou aceleração -medida de humidade -medida de níveis de líquidos condutores e não

-CONDENSADOR DIFERENCIAL -dois condensadores variáveis dispostos fisicamente de forma a terem uma variação em sentidos diferentes -variação da distância ou da área

CONDICIONADORES DE SINAL PARA SENSORES DE REACTÂNCIA VARIÁVEL

-lei de Ohm -divisor tensão -pontes Wheatstone

-blindagens electrostáticas -guardas activas

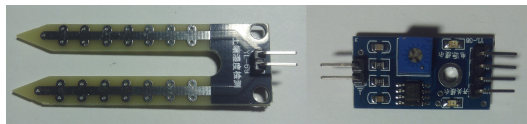


Figura 2.67: sensor humedad.

2.16 Índice de Acidez (pH)

A acidez de uma substância aquosa é medida com a escala de pH (potência de hidrogénio) com valores entre 1 e 10. As substâncias muito ácidas têm um valor de pH próximo de 0 (ácido das baterias de carro), as substâncias muito alcalinas têm um valor de pH próximo de 14 (soda caustica ou hidróxido de sódio). As substâncias neutras têm um valor de pH próximo de 7 (água destilada). A figura 2.68 ilustra o nível de pH de diversas substâncias.

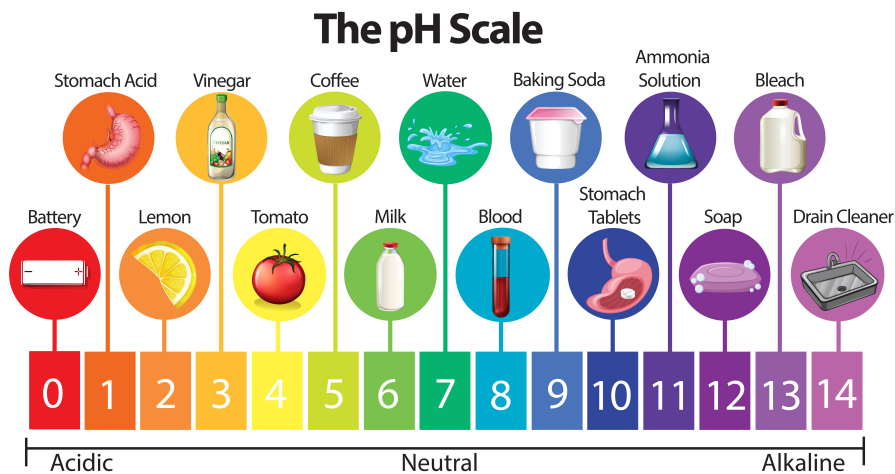


Figura 2.68: Escala níveis de pH.

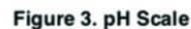
Em diversos processos industriais é necessário controlar o nível de pH para produzir determinados produtos, como também é necessário garantir que os produtos libertados estão neutralizados e não vão causar danificar o meio ambiente. Isso é conseguido através de uma monitorização constante através de sensores de pH.

Os sensores de pH são caracterizados por serem passivos e geradores de uma tensão muito pequena (mV) e uma resistência elevada aos seus terminais. Os sensores mais habituais são construídos por dois eletrodos, um de referência (5) e o de medida que se encontra dentro de uma ampola de vidro (7).

Existe outro tipo de sensores de pH fabricados com base em tecnologia semicondutora MOSFET, designados por pHFETS.

A figura ?? mostra 2 medidores de pH portáteis. A figura 2.73 mostra um sensor e o respectivo circuito condicionador de sinal para o ligar a um Arduino. O circuito condicionador consiste no amplificador de elevado ganho.

Antes de realizar medidas com um novo sensor é necessário proceder à sua calibração. A calibração do circuito deve ser feita com um conjunto de soluções com pH conhecido, ou deve-se usar um medidor calibrado para determinar pH dos líquidos a usar. Não deve ser usada água destilada, mesmo tendo pH 7 ou pH neutro, pois ela está desprovida de iões H^+ para que o



The diagram illustrates a pH sensor system. The sensor structure includes a silicon substrate with an oxide layer, a drain, a source, and a mobile charge region. A pH-responsive membrane is positioned above the source region, with a reference electrode and sample solution above it. The drain is connected to a voltage source V_{ds} . The source is connected to a current source i_{Ds} , which is also connected to a feedback loop. The feedback loop includes a set point and an output voltage V_g .

Figura 2.70: Modelo do sensor de pHFET.

156



Figura 2.71: medidores ph.

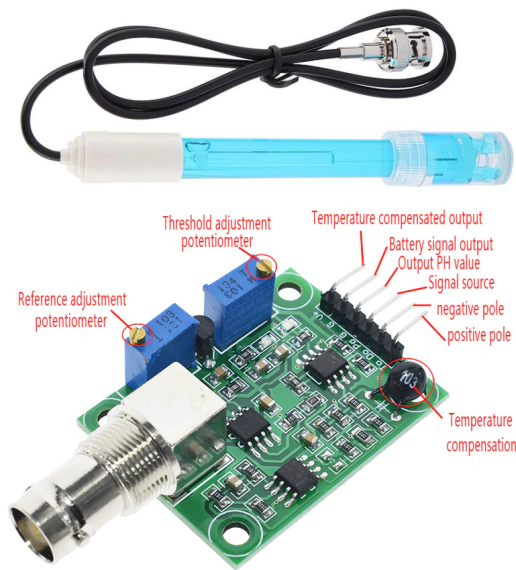


Figura 2.72: medidores ph.

aumentar a concentração de iões.

Em termos práticos os sensores de pH possuem uma resistência interna muito elevada ($M\Omega$) e são lentos a convergir para um valor estável de medida (minutos).

<https://www.14core.com/wiring-the-ph-power-hydrogen-sensor-bnc-electrode-probe-with-microcontroller/>

ph do solo?

A figura

2.17 Aceleração

O acelerómetro mede a aceleração a que o sensor está sujeito. A aceleração medida pode ser causada por movimento do sensor e por acção da força gravítica. Na física, a aceleração é a taxa de variação da velocidade durante

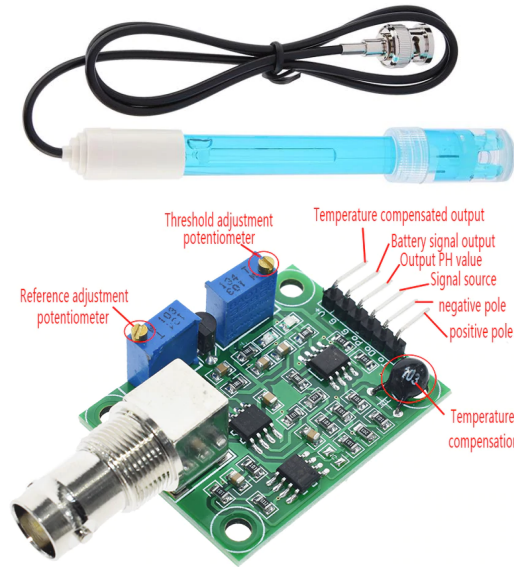


Figura 2.73: Circuito condicionador de sinal para um sensor de pH.

um determinado período de tempo, e é medida em metros por segundo ao quadrado (m/s^2). A velocidade é o deslocamento percorrido durante um determinado período de tempo, e é medida em metros por segundo (m/s). As grandezas também se podem relacionar de outra forma: o deslocamento é o produto da velocidade pelo tempo. Como a velocidade é o produto da aceleração pelo tempo, o deslocamento é o quadrado da aceleração vezes o tempo. Em termos matemáticos, em ordem ao tempo:

- Aceleração (a): é a (primeira) derivada da velocidade $a = dv/dt$, e é a segunda derivada do deslocamento $a = dl/dt^2$
- Velocidade (v): é o integral da aceleração $v = \int a/dt$, e a derivada do deslocamento $v = dl/dt$.
- Deslocamento (l): é o integral duplo da aceleração $l = \int \int a/dt$, e o integral da velocidade $l = \int v/dt$.

Em termos práticos:

- Quando a aceleração é nula a velocidade é constante. Quando é positiva, a velocidade aumenta e quando é negativa a velocidade reduz;
- Quando a velocidade é nula o deslocamento é constante. Quando é positiva, o deslocamento acontece numa direcção, e quando é negativa o deslocamento acontece no sentido oposto.

Um acelerómetro analógico disponível é o ADXL35. Este acelerómetro ...

A listagem 2.24 exemplifica a programação de um Arduino Nano a um ADXL335 para a leitura dos valores de aceleração dos 3 eixos e apresentação dos seus valores. Este Sketch define a classe ADXL335, cujos atributos mais relevantes são os pinos de entrada analógicos, e as variáveis para registo das amostras adquiridas. O método `calibrate()` é responsável pela calibração, que é conseguida através da média das primeiras 256 (2^8) amostras recebidas em cada eixo. O método `getData()` lê uma amostra de cada eixo e coloca numa fila do tipo *First In First Out* (FIFO). Quando o programa inicia (método `setup()`) a UART é iniciada com 115200 baud e o acelerómetro calibrado, para compensar o efeito da gravidade. Existe a possibilidade de escolher a funcionalidade do programa:

Ao deixar o acelerómetro em repouso numa posição diferente da inicial, aquando da calibração, vai originar a leitura de valores de aceleração constantes, pois os eixos sobre os quais a força se fazia sentir mudaram.

A figura 2.74 mostra a montagem do acelerómetro ADXL335 num Arduino nano. A figura 2.75 mostra o conjunto de sinais enviados pelo Arduino para o computador e mostrado no traçador gráfico do ambiente Arduino.

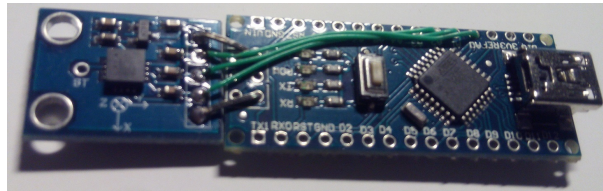


Figura 2.74: Acelerómetro analógico ADXL335.

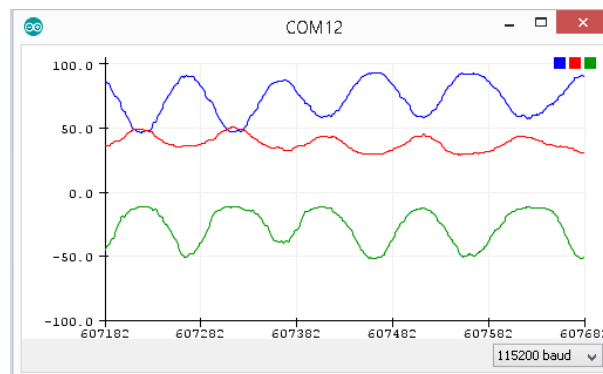


Figura 2.75: Diagrama temporal dos 3 canais medidos pelo acelerómetro ADXL335.

Listagem 2.19: Leitura e processamento dos dados do acelerómetro ADXL335

```
/* accelerometer test: arduino nano + adxl335 */
#define ACCLX_PIN A0 // x-axis of the accelerometer
```

CAPÍTULO 2. SENSORES

```
#define ACCLX_PIN A1 // y-axis
#define ACCLZ_PIN A2 // z-axis (only on 3-axis models)

#define UPDATE_TIME 8
#define ACCL_SAMPLES 32
#define BIAS_SAMPLES 256
#define ACCL_THRESHLD 1

#define DEMO_MODE 1
// modes: 0 raw data, 1 acceleration, 2 velocity, 3 movement

class ADXL35 {
private:
    int xPin, yPin, zPin;
    int xBias, yBias, zBias;
    int xVal[ACCL_SAMPLES], yVal[ACCL_SAMPLES], zVal[ACCL_SAMPLES];
    int xAcc[2], yAcc[2], zAcc[2];
    int xVel[2], yVel[2], zVel[2];
    int xMov[2], yMov[2], zMov[2];
public:
    ADXL35() { xPin = ACCLX_PIN; yPin = ACCLY_PIN; zPin = ACCLZ_PIN; }
    void calibrate() {
        int i;
        long x = 0, y = 0, z = 0;
        for(i = 0; i < BIAS_SAMPLES; i++) {
            x += analogRead(xPin);
            y += analogRead(yPin);
            z += analogRead(zPin);
        }
        xBias = x / BIAS_SAMPLES;
        yBias = y / BIAS_SAMPLES;
        zBias = z / BIAS_SAMPLES;
        for(i = 0; i < ACCL_SAMPLES; i++) {
            xVal[i] = xBias;
            yVal[i] = yBias;
            zVal[i] = zBias;
        }
    }

    void getData() {
        int i;
        // shift data
        for(i = ACCL_SAMPLES - 1; i > 0; i--) {
            xVal[i] = xVal[i-1];
            yVal[i] = yVal[i-1];
            zVal[i] = zVal[i-1];
        }
        // get new sample
        xVal[0] = analogRead(xPin);
        yVal[0] = analogRead(yPin);
        zVal[0] = analogRead(zPin);
    }
    // get last sample from sensor
    int getX() { return xVal[0]; }
    int getY() { return yVal[0]; }
    int getZ() { return zVal[0]; }
    // get instant acceleration measurement
    int getInstAccX() { return xVal[0] - xBias; }
    int getInstAccY() { return yVal[0] - yBias; }
    int getInstAccZ() { return zVal[0] - zBias; }
    // get average acceleration measurement
    void doAvgAcc() {
        int i;
        long xAccum = 0, yAccum = 0, zAccum = 0;
        for(i = 0; i < ACCL_SAMPLES; i++) {
            xAccum += xVal[i] - xBias;
            yAccum += yVal[i] - yBias;
            zAccum += zVal[i] - zBias;
        }
        // save previous value
        xAcc[1] = xAcc[0];
        yAcc[1] = yAcc[0];
        zAcc[1] = zAcc[0];
    }
}
```

```

    // compute new averages
    xAccel[0] = xAcum / ACCLSAMPLES;
    yAccel[0] = yAcum / ACCLSAMPLES;
    zAccel[0] = zAcum / ACCLSAMPLES;
    // min-threshold
    if((xAccel[0] < ACCLTHSHLD) && (xAccel[0] > -ACCLTHSHLD)) {
        xAccel[0] = 0;
    }
    if((yAccel[0] < ACCLTHSHLD) && (yAccel[0] > -ACCLTHSHLD)) {
        yAccel[0] = 0;
    }
    if((zAccel[0] < ACCLTHSHLD) && (zAccel[0] > -ACCLTHSHLD)) {
        zAccel[0] = 0;
    }
}

int getAccelX() { return xAccel[0]; }
int getAccelY() { return yAccel[0]; }
int getAccelZ() { return zAccel[0]; }

// compute velocity
void doVel() {
    // keep last value
    xVel[1] = xVel[0];    yVel[1] = yVel[0];    zVel[1] = zVel[0];
    // compute new value
    xVel[0] += (xAccel[0] + xAccel[1]) ;
    yVel[0] += (yAccel[0] + yAccel[1]) ;
    zVel[0] += (zAccel[0] + zAccel[1]) ;
}

// compute movement
int getVelX() { return xVel[0]; }
int getVelY() { return yVel[0]; }
int getVelZ() { return zVel[0]; }
void doMov() {
    xMov[0] += (xVel[0] + xVel[1]) >> 1;
    yMov[0] += (yVel[0] + yVel[1]) >> 1;
    zMov[0] += (zVel[0] + zVel[1]) >> 1;
}
int getMovX() { return xMov[0]; }
int getMovY() { return yMov[0]; }
int getMovZ() { return zMov[0]; }
} accl;

char buf[80];

void setup() {
    // initialize the serial communications:
    Serial.begin(115200);
    while(!Serial);
    Serial.println("!!! Calibrating accelerometer. Do not move sensor !!!");
    accl.calibrate();
    Serial.println("Calibration finished.");
}

void loop() {
    // read sensor values for 3 axis
    accl.getData();

    #if DEMOMODE == 0
        sprintf(buf, "%d,%d,%d\n", accl.getInstAccelX(), accl.getInstAccelY(), accl.getInstAccelZ());
    #elif DEMOMODE == 1
        accl.doAvgAccel();
        sprintf(buf, "%d,%d,%d\n", accl.getAccelX(), accl.getAccelY(), accl.getAccelZ());
    #elif DEMOMODE == 2
        accl.doAvgAccel();
        accl.doVel();
        sprintf(buf, "%d,%d,%d\n", accl.getVelX(), accl.getVelY(), accl.getVelZ());
    #elif DEMOMODE == 3
        accl.doAvgAccel();
        accl.doVel();
        accl.doMov();
    #endif
}

```

CAPÍTULO 2. sensores

```
    sprintf(buf, "%d,%d,%d\n", accl.getMovX(), accl.getMovY(), accl.getMovZ());
#endif
    Serial.println(buf); // print comma separated values

    // delay before next reading:
    delay(UPDATETIME);
}
```

2.18. ACELERÓMETRO + MAGNETÓMETRO + GIROSCÓPIO +
ALTÍMETRO

**2.18 Acelerómetro + Magnetómetro + Gi-
roscópio + Altímetro**

2.19 Localização (GNSS/GPS)

Os sistemas de posicionamento global, *Global Navigation Satellite System* (GNSS), permitem obter a localização de um terminal em qualquer parte do Mundo, à custa de uma constelação de satélites que emitem um sinal rádio que permite a um recetor calcular a posição dada a distância a cada satélite. A distância a cada satélite em linha de vista é proporcional ao tempo de propagação do sinal. A resolução das coordenadas GPS varia entre os 5 metros e os 30 centímetros, dependendo do recetor.

O sistema de posicionamento global, *Global Positioning System* (GPS), foi inicialmente desenvolvido pelos militares Norte Americanos a partir dos anos 70, tendo ficado completamente operacional em 1995, disponível mas com limitações para uso civil em 1996, e sem limitações desde 2000. Existem outros sistemas em funcionamento em paralelo, com constelações de satélites independentes, nomeadamente o *Global'naya Navigatsionnaya Sputnikovaya Sistema* (GLONASS) (Russo) e o Galileo (Europeu). Existem ainda outros sistemas em desenvolvimento por parte de outros países.

Os recetores GNSS começaram por ser equipamentos que permitiam apenas obter as coordenadas da localização, o que era suficiente para poder comunicar a posição em caso de emergência. Com os avanços tecnológicos e a miniaturização da eletrónica dos recetores, eles passaram a estar disponíveis como pequenos módulos que integram diversos equipamentos tais como veículos automóveis e telemóveis. Atualmente o recetor GNSS consiste num único chip capaz de receber sinais das várias constelações e calcular a posição rapidamente. Para poder receber os sinais RF transmitidos pelos satélites é necessário usar uma antena própria para as frequências dos sistemas GNSS. Se existirem poucos satélites em linha de vista com a antena, o módulo poderá demorar mais tempo a determinar a posição, e com pouca precisão e até alguns desvios, especialmente junto a edifícios altos.

Recetores mais recentes têm a capacidade de auxiliar a descodificação dos sinais recebidos com base na informação da última localização. Os recetores GNSS transmitem a informação de localização e de tempo em diversos formatos, sendo o mais usado o National Marine Electronics Association (NMEA). Este formato é enviado em caracteres ASCII podendo ser facilmente lido num terminal de texto.

As mensagens NMEA começam com o carácter \$ seguido do identificador da mensagem. Os campos das mensagens são separados por “,”, e todas as mensagens terminam com *NN, onde NN é um byte representado em hexadecimal com o resultado da operação XOR de todos os bytes da mensagem. Existem mensagens para diversos fins usadas em navegação, nomeadamente piloto automático e estabelecimentos de rotas. As mensagens com informação de localização GPS são:

- GPRMC - Informação mínima recomendada: posição, velocidade, e

tempo;

- GPVTG - Velocidade no terreno;
- GPGGA - Informação de hora, localização, satélites visíveis;
- GPGSA - Informação sobre os satélites ativos e a precisão das coordenadas obtidas;
- GPGSV - Informação sobre os satélites visíveis e a sua localização.

Para demonstrar como dotar um sistema embebido da informação acerca da sua localização usou-se um módulo recetor GPS Blox NEO-6M, apresentado na figura 2.76. Este módulo vem soldado numa breakout board, com a configuração 9600 baud para a velocidade da porta série e transmite a informação recebida dos satélites uma vez por segundo.

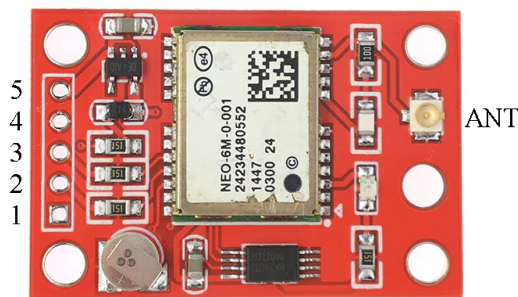


Figura 2.76: Módulo GPS NEO-6M.

GY-NEO6MV2

A *breakout board* deste módulo possui ligações para VCC, GND, TX e RX. Os dois primeiros servem para alimentar o módulo e outros dois para a comunicação série assíncrona. Existe um quinto pino nomeado PPS com um sinal de saída que pulsa a cada segundo. Na extremidade oposta do módulo existe uma ficha do tipo SMA para ligação da antena GPS. É fundamental que a antena não esteja obstruída, e de preferência em linha de vista com os satélites. O esquema elétrico realizado em Fritzing está ilustrado na figura 2.77. Às ligações necessárias para estabelecer a ligação entre o módulo e o Arduino foi adicionado um condensador eletrolítico para estabilizar a tensão no módulo e evitar flutuações de tensão nos instantes em que o módulo consome mais corrente. Este condensador pode ter um valor de capacidade acima de 100 μF . O módulo pode ser alimentado entre 3,3 V e 5 V. Quando alimentado a 5 V, este módulo consome entre 50-80 mA. A figura 2.78 mostra o aspeto da montagem em breadboard do módulo GPS NEO-6M ligado a um Arduino Uno.

O Sketch 2.20 serve para ter um Arduino Uno a fazer a ponte entre o módulo GPS e o computador. Este código começa por configurar os portos

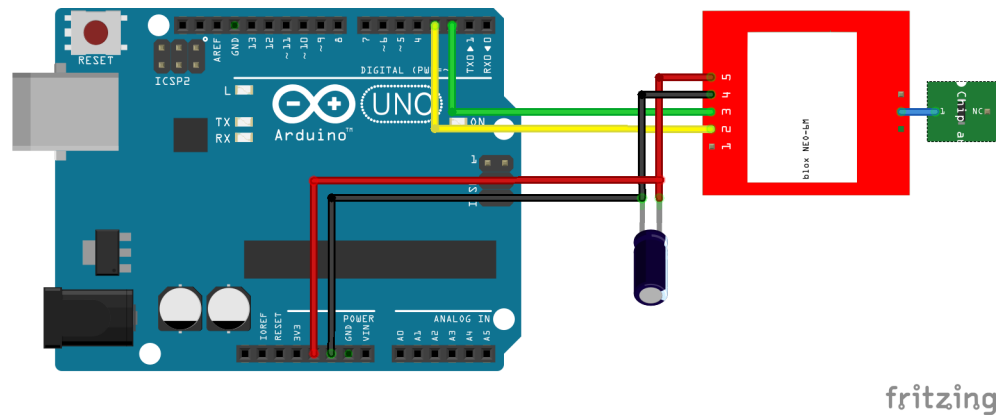


Figura 2.77: Esquema de ligações em breadboard do módulo GPS.

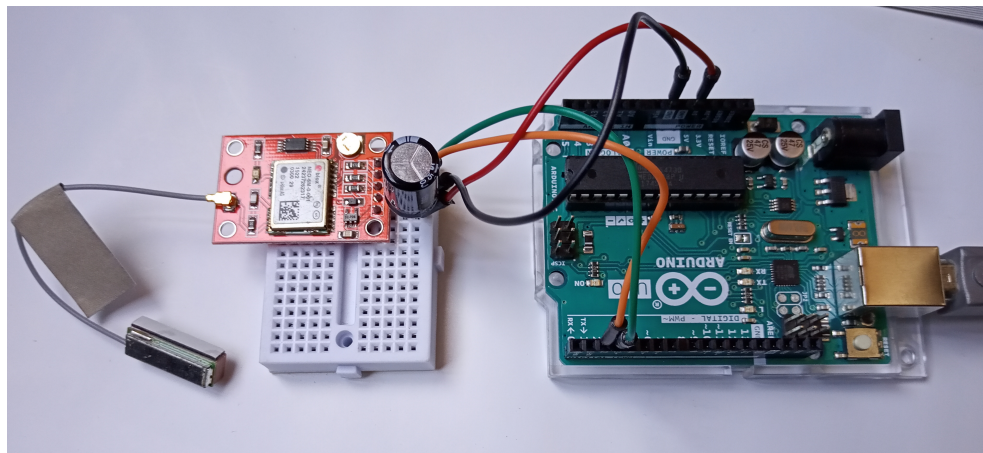


Figura 2.78: Montagem de ensaio do módulo GPS com um Arduino Uno.

série (hardware e software), e tudo o que faz é avaliar se existem caracteres recebidos do porto do módulo GPS, e caso existam, encaminhar para o porto série do terminal de texto todos os caracteres recebidos do módulo GPS. A figura 2.79 ilustra as mensagens enviadas para o terminal de texto do Arduino após o arranque do módulo.

Listagem 2.20: GPS.ino

```
#include <SoftwareSerial.h>

SoftwareSerial gpsSerial(2, 3); // RX, TX

void setup()
{
  // Arduino-Host Computer
  Serial.begin(9600);
  while (!Serial);

  // Arduino-GPS
  gpsSerial.begin(9600);
}
```


2.19. LOCALIZAÇÃO (GNSS/GPS)

```
void loop()  
{  
  if (gpsSerial.available())  
    Serial.write(gpsSerial.read());  
}
```

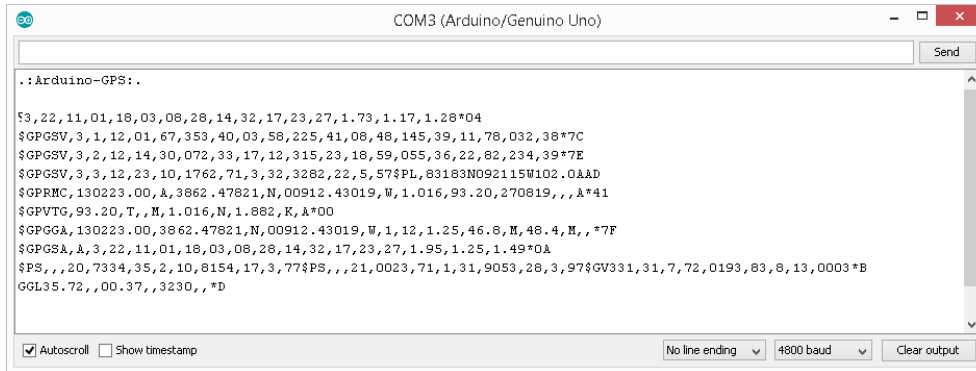


Figura 2.79: Mensagens de localização enviadas pelo módulo GPS.

Fechando o terminal de texto do Arduino é possível usar o Google Earth para mostrar a fotografia do terreno correspondente às coordenadas recebidas. De salientar que a informação enviada pelo módulo vem a 9600 baud e a aplicação Google Earth está à espera de receber os dados em formato NMEA a 4800 baud. Para ativar a receção das coordenadas da posição atual é necessário abrir a ferramenta GPS e ativar o local “Realtime GPS” usando o formato NMEA. A figura 2.80 mostra a localização do Arduino indicada na imagem de satélite sobre Lisboa.

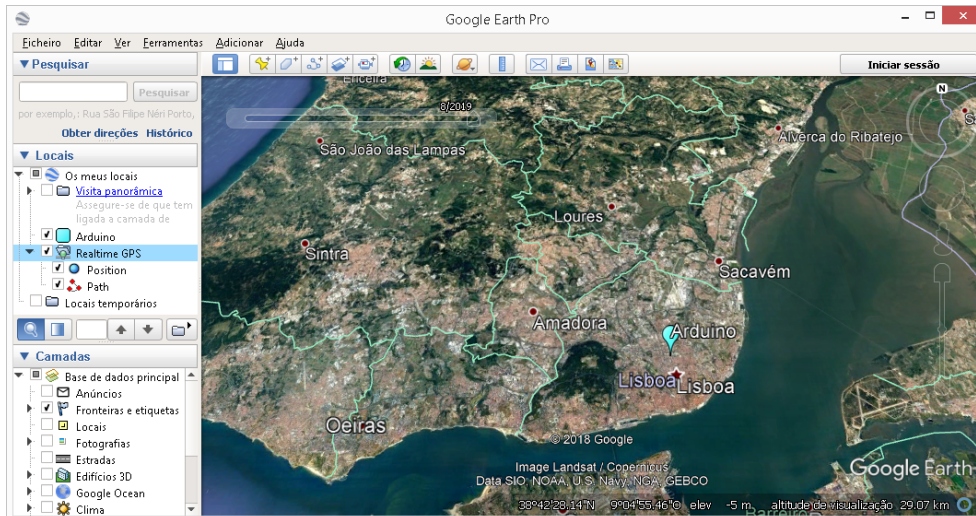
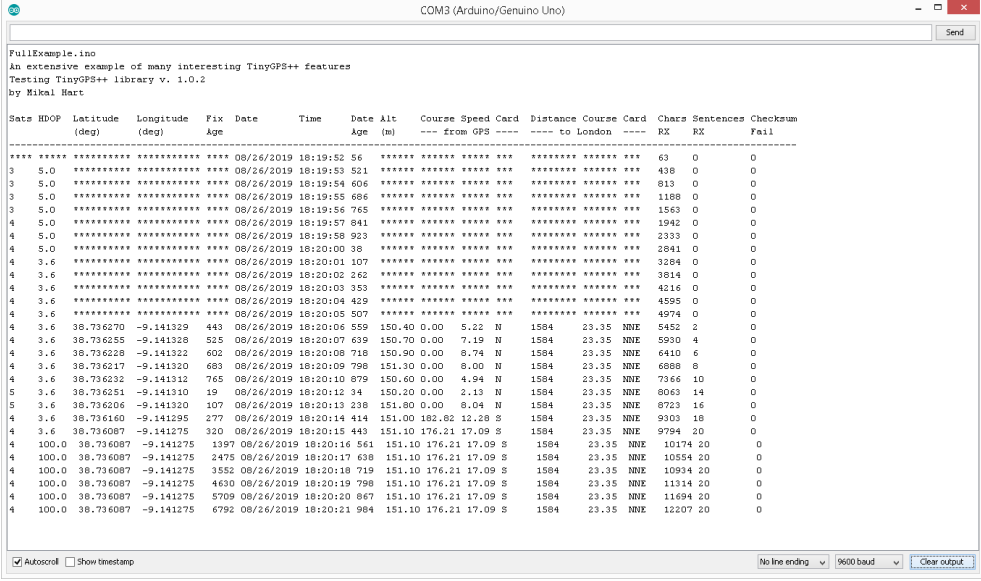


Figura 2.80: Localização do Arduino sobre a imagem de satélite no Google Earth.

CAPÍTULO 2. SENSORES

Para auxiliar a decodificação das coordenadas GPS usou-se a biblioteca Arduino TinyGPS++⁴. O resultado da execução do Sketch de demonstração `FullExample.ino` visualizado no terminal de texto encontra-se na figura 2.81. Este exemplo permite obter: coordenadas longitude e latitude, data e hora, altitude, e velocidade. Este exemplo mostra que foi necessário esperar pela recepção de 14 mensagens NMEA até obter a localização e começar a produzir dados válidos.

Neste Sketch, é usado instanciado o `gps` objeto da classe `TinyGPSPlus`. Os objetos desta classe recebem os dados do porto série assíncrono à medida que vão sendo produzidos pelo módulo GPS através do método `encode(char c)`. Os objetos da classe `TinyGPSPlus` contêm por sua vez como atributos objetos das estruturas `TinyGPSLocation`, `TinyGPSDate`, `TinyGPSTime`, `TinyGPSSpeed`, `TinyGPSCourse`, `TinyGPSAltitude`, `TinyGPSInteger` e `TinyGPSHDOP` que são usadas para obter os dados e realizar os cálculos necessários para obter cada um dos parâmetros do sinal GPS.



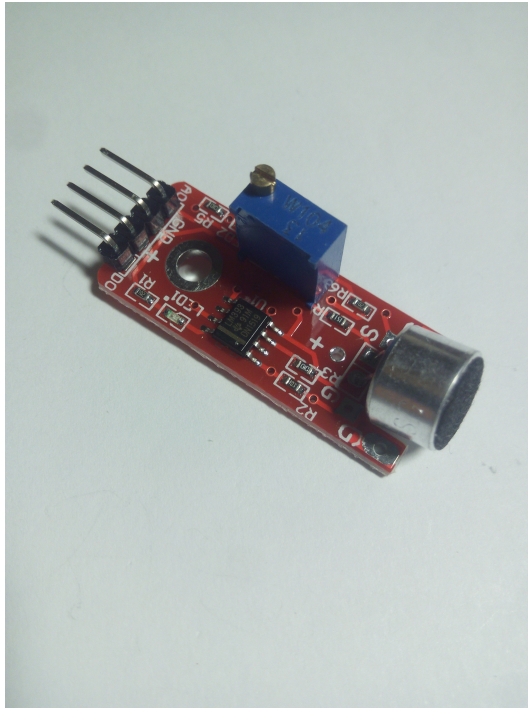
Sats	HDOP	Latitude (deg)	Longitude (deg)	Fix	Date	Time	Date Alt Age (m)	Course Speed Card --- from GPS ---	Distance Course Card ---- to London ----	Chars RX	Sentences RX	Checksum Fail
3	5.0				08/26/2019	18:19:52	56			63	0	0
3	5.0				08/26/2019	18:19:53	521			438	0	0
3	5.0				08/26/2019	18:19:54	606			813	0	0
3	5.0				08/26/2019	18:19:55	686			1188	0	0
3	5.0				08/26/2019	18:19:56	765			1563	0	0
4	5.0				08/26/2019	18:19:57	841			1942	0	0
4	5.0				08/26/2019	18:19:58	923			2333	0	0
4	5.0				08/26/2019	18:20:00	38			2841	0	0
4	3.6				08/26/2019	18:20:01	107			3284	0	0
4	3.6				08/26/2019	18:20:02	262			3814	0	0
4	3.6				08/26/2019	18:20:03	353			4216	0	0
4	3.6				08/26/2019	18:20:04	429			4595	0	0
4	3.6				08/26/2019	18:20:05	507			4974	0	0
4	3.6	38.736270	-9.141329	443	08/26/2019	18:20:06	559	150.40 0.00 5.22 N	1584 23.35 NME	5452	2	0
4	3.6	38.736255	-9.141328	525	08/26/2019	18:20:07	639	150.70 0.00 7.19 N	1584 23.35 NME	5930	4	0
4	3.6	38.736228	-9.141322	602	08/26/2019	18:20:08	718	150.90 0.00 8.74 N	1584 23.35 NME	6410	6	0
4	3.6	38.736217	-9.141320	683	08/26/2019	18:20:09	798	151.30 0.00 8.00 N	1584 23.35 NME	6868	8	0
4	3.6	38.736232	-9.141312	765	08/26/2019	18:20:10	879	150.60 0.00 4.94 N	1584 23.35 NME	7366	10	0
5	3.6	38.736251	-9.141310	19	08/26/2019	18:20:12	94	150.20 0.00 2.13 N	1584 23.35 NME	8063	14	0
5	3.6	38.736206	-9.141320	107	08/26/2019	18:20:13	238	151.80 0.00 8.04 N	1584 23.35 NME	8723	16	0
4	3.6	38.736160	-9.141295	277	08/26/2019	18:20:14	414	151.00 182.82 12.28 S	1584 23.35 NME	9303	18	0
4	3.6	38.736087	-9.141275	320	08/26/2019	18:20:15	443	151.10 176.21 17.09 S	1584 23.35 NME	9794	20	0
4	100.0	38.736087	-9.141275	1397	08/26/2019	18:20:16	561	151.10 176.21 17.09 S	1584 23.35 NME	10174	20	0
4	100.0	38.736087	-9.141275	2475	08/26/2019	18:20:17	638	151.10 176.21 17.09 S	1584 23.35 NME	10554	20	0
4	100.0	38.736087	-9.141275	3552	08/26/2019	18:20:18	719	151.10 176.21 17.09 S	1584 23.35 NME	10934	20	0
4	100.0	38.736087	-9.141275	4630	08/26/2019	18:20:19	798	151.10 176.21 17.09 S	1584 23.35 NME	11314	20	0
4	100.0	38.736087	-9.141275	5709	08/26/2019	18:20:20	867	151.10 176.21 17.09 S	1584 23.35 NME	11694	20	0
4	100.0	38.736087	-9.141275	6792	08/26/2019	18:20:21	984	151.10 176.21 17.09 S	1584 23.35 NME	12207	20	0

Figura 2.81: Mensagens de localização decodificada usando a biblioteca TinyGPS++.

⁴<https://github.com/mikalhart/TinyGPSPlus>

2.20 Som

O som é composto por ondas que se propagam no ar. Para captar estas ondas e transformá-las num sinal elétrico usam-se microfones. Os microfones podem ser construídos com várias tecnologias, sendo o mais vulgar em sistemas embebidos os de eletreto, piezoelétrico e dinâmico.



esquema circuito

Figura 2.82: Microfone montado numa placa *breakout* e circuito para polarização do microfone.

A figura 2.83 mostra o aspeto do sinal áudio captado pelo microfone num espaço onde se ouvia música . A irregularidade deve-se à presença das várias frequências em simultâneo.

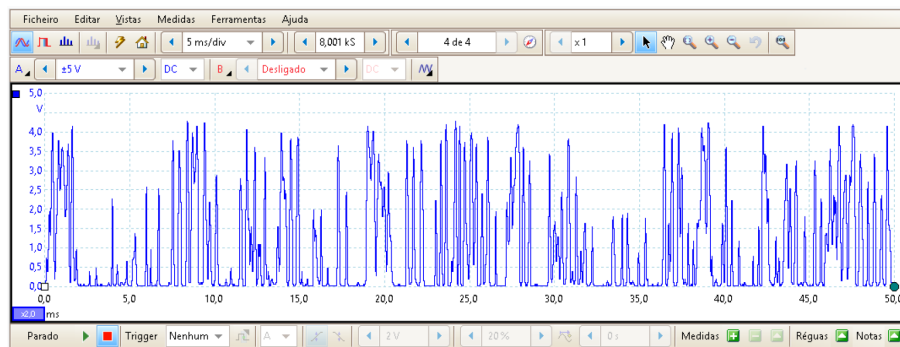


Figura 2.83: Sinal áudio adquirido pelo microfone.

2.21 Som Subaquático

O hidrofone é um microfone para captar sons debaixo de água, nomeadamente vida marinha ou atividade sísmica subaquática. Este tipo de microfone tem uma largura de banda estreita porque os sons não se propagam na água da mesma forma como no ar. Este tipo de sensor é do tipo piezoelétrico e funciona na frequência dos 10 Hz e tem uma impedância de 160 Ω . O hidrofone é construído num invólucro estanque para que a água não entre. A figura 2.84 mostra o aspecto de um hidrofone.

Tal como o geofone, a impedância do hidrofone faz com que os sinais por ele gerados sejam de muito pequena amplitude e seja necessário usar um pré-amplificador para aumentar a amplitude do sinal antes de ser convertido em digital. A gama de frequências do hidrofone é facilmente adquirida por um Arduino.



Figura 2.84: Aspeto de um hidrofone.

2.22 Pressão Atmosférica

Em altitudes mais elevadas as moléculas de ar ficam mais separadas entre si, o que faz com que a pressão seja menor. Em altitudes baixas, as moléculas de ar ficam menos espaçadas, o que faz aumentar a pressão atmosférica. A pressão atmosférica também muda com o clima. As altas pressões devem-se à descida da massa de ar frio, normalmente associadas a dias de chuva, ao passo que as baixas pressões devem-se ao aquecimento da superfície terrestre.

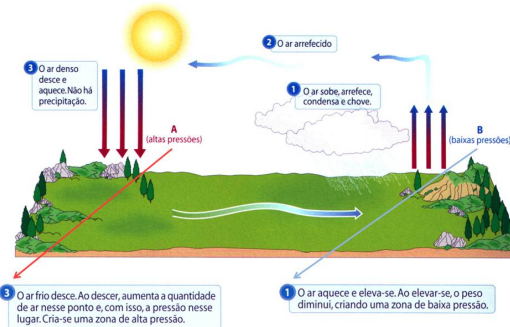


Figura 2.85: pressão conceito.

A pressão é medida em Pascal.

$40 \text{ kPa} = 0,4 \text{ bar} = 0,4078 \text{ kgf/cm}^2$ (kg per sq. cm) = 4078.865 kgf/m^2 (kg per sq. meter)

A figura 2.87 mostra o aspecto dos sensores de pressão atmosférica BMP180 e BME280.

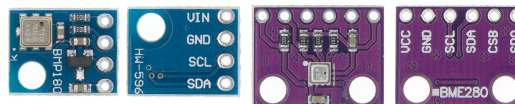


Figura 2.86: BMP180 e BME280.

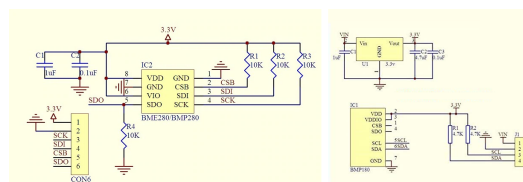


Figura 2.87: Esquema dos circuitos nas placas BMP180 e BME280.

BMP180 e BMP280

Para sistemas pneumáticos ou de verificação de fugas de ar ou líquidos:
MPS20N0040D-D Sphygmomanometer Pressure Sensor 0-40kPa DIP-6 For Arduino Raspb

2.23 Pressão Hidrostática

2.24 Força/Peso

A medição da força exercida num determinado sentido sobre um corpo pode ser aferida com um extensómetro colocado sobre o seu corpo. O extensómetro é uma resistência variável que varia de valor quando é deformada. Estando acoplado, ou colado, a uma peça semi-rígida, a deformação acontece quando é aplicada uma força externa, normalmente esticado ou comprimido. Existem extensómetros com um ou mais eixos, medindo a resistência correspondente ao deslocamento apenas numa direção ou em várias em simultâneo. Para além de serem usados para medir o peso, eles são também usados na monitorização de deformações em estruturas (por exemplo: edifícios, pontes, torres de transmissão e paredes de travagem de terras) e equipamentos ou peças que apresentem pequenos desvios nalguns pontos críticos devido a acção mecânica ou variação das condições ambientes, nomeadamente suscetíveis a dilatação por temperatura elevada ou vibrações de equipamentos em funcionamento.

Figura 2.88: Extensómetros.

Um extensómetro é caracterizado pela sua deformação máxima suportada, que se traduz pela força máxima aplicada, a sua resistência em repouso, variação máxima de resistência, e tensão máxima de funcionamento.

O extensómetro usado nesta demonstração possui uma resistência em repouso de $410\ \Omega$ e tem uma variação máxima de $30\ \Omega$, e está preparado para funcionar com tensões inferiores a 15 V . A figura 2.89 mostra o aspeto de um extensómetro (esquerda) e o circuito elétrico equivalente (direita). O fio vermelho corresponde à entrada de VCC e o preto a GND. As saídas do sensor são o fio azul com o positivo e o branco com o negativo.

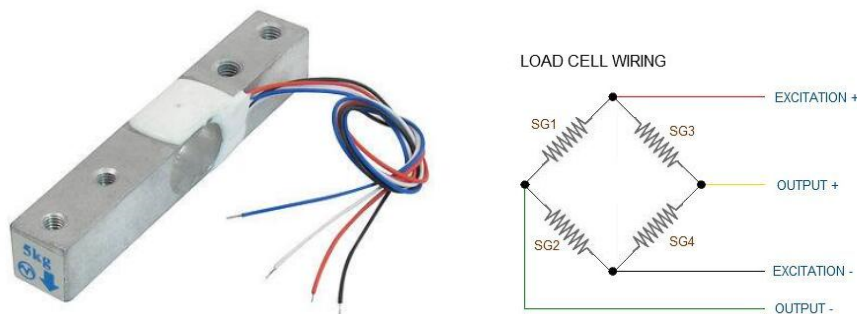


Figura 2.89: Extensómetro (esquerda) e circuito elétrico equivalente (direita).

Devido aos pequenos desvios e às pequenas variações de resistência envolvidas, o método apresentado anteriormente para obter uma medida elétrica proporcional à queda de tensão num sensor resistivo, neste caso não produz

bons resultados pois a variação da resistência é baixa para a resolução oferecida pela montagem. Por outro lado o ruído que afeta os sinais vai influenciar as medidas.

O circuito condicionador de sinal escolhido para o extensômetro é o HX711. Este circuito integrado está preparado para ligar diretamente a um extensômetro, contém um amplificador de ganho programável e um conversor analógico-digital capaz de gerar 80 amostras de 24 bits por segundo. O Hx711 tem uma interface digital série síncrona para comunicar com um sistema embebido. Com conversões de 24 bits, este ADC é capaz de identificar 16.777.216 níveis de tensão diferentes, ou seja, numa gama de 0-5 V, significa que ele é sensível a variações de 0,298 μ V. Em termos práticos, a discriminação mínima oferecida por este ADC vai estar na mesma ordem de grandeza do ruído presente nos fios devido a interferências e flutuações nos sinais de alimentação.

Como o HX711 é fornecido em encapsulamento SMD, é necessário usar uma placa de adaptação, ou *breakout* para fazer as ligações mais facilmente a um Arduino. A figura 2.90 à esquerda mostra o diagrama de blocos do HX711, de acordo com o seu manual técnico, e à direita a ligação de uma placa *breakout* a um Arduino Uno e a um extensômetro. Nesta montagem, o HX711 fornece 10 amostras por segundo.

Para demonstração do sistema foi criado um aparato para funcionar como uma balança, ilustrado na figura 2.91. A listagem 2.21 contém o Sketch de demonstração da utilização do extensômetro e funcionamento como balança, enviando os valores das medidas para o terminal de texto do Arduino. Neste Sketch encontra-se a classe Hx711, que dá suporte à leitura dos valores do sensor. O construtor desta classe recebe como argumentos: O método `readSensor()` é responsável pela interação com o circuito condicionador de sinal e receber as amostras do seu ADC.

Durante o ensaio foram realizadas diversas medidas com pesos conhecidos, nomeadamente 5, 10, 20, 50 e 100 gramas. Para estabelecer a relação entre o valor do ADC e o peso medido cada peso medido, foi calculado o valor médio de 10 amostras. A tabela 2.10 mostra a relação entre os valores medidos pelo HX711 e os pesos no prato da balança. De acordo com os valores recolhidos, em média um grama corresponde a 402 medidos no ADC. A figura 2.92 mostra a medida do peso no prato da balança enquanto vários pesos foram colocados e retirados do prato da balança.

Listagem 2.21: Exemplo da aplicação de um extensômetro e do HC711 para funcionarem como uma balança de precisão.

```
// 1gr = 402
// 5gr = 2010
// 10gr = 4020
// 20gr = 8040
// 40gr = 16080
// 50gr = 20100
// 100gr = 40200
// 200gr = 80400
```

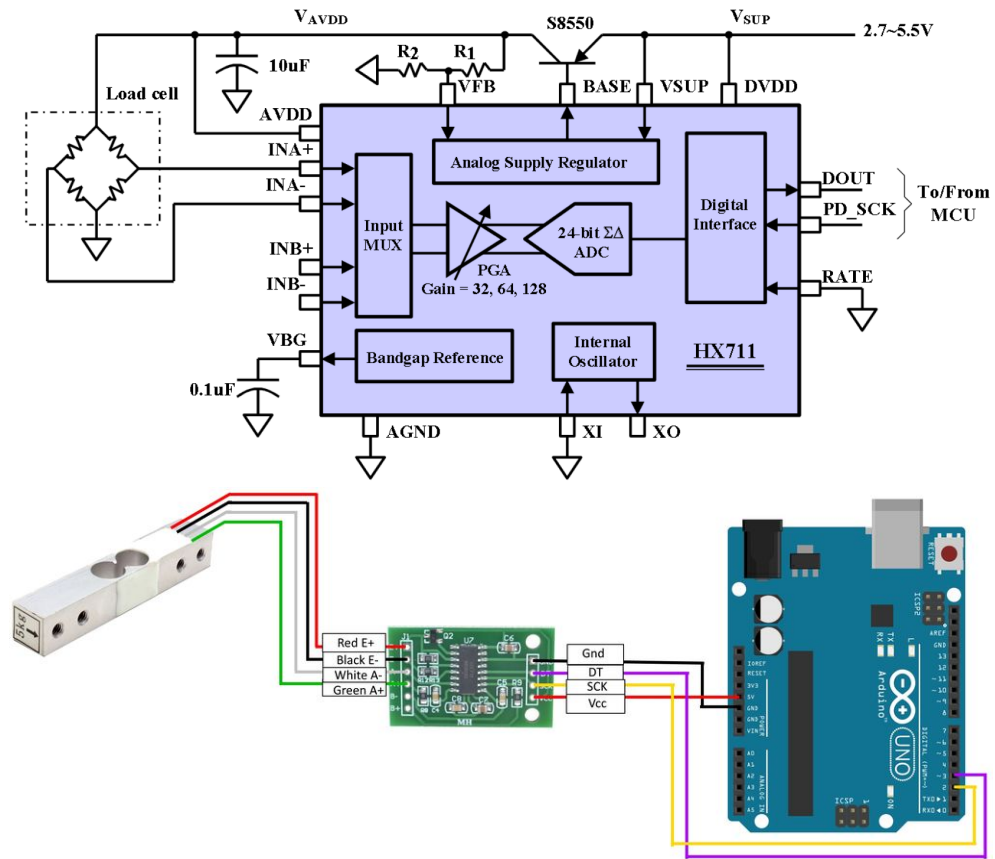


Figura 2.90: Montagem do extensômetro e respetivo circuito de condicionamento de sinal a um Arduino Uno.

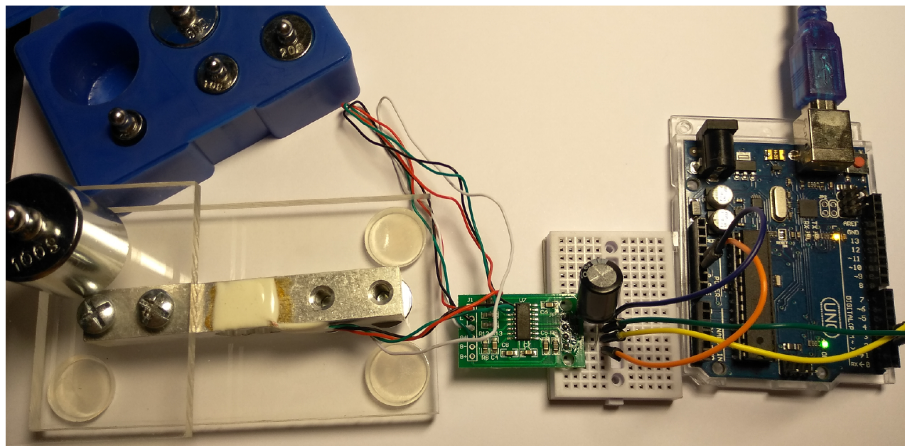


Figura 2.91: Montagem da balança.

```
#define HX711_DATA_PIN 2
#define HX711_SCLK_PIN 3
```

Peso (gramas)	Valor medido
5	2010
10	4020
20	8040
40	16080
50	20100
100	40200
200	80400

Tabela 2.10: Correspondência entre o peso no prato da balança e a medida lida do sensor.

```

union sensorDataType {
    byte part[4];
    long val;
} ;

class Hx711 {
private:
    byte clkPin;
    byte dataPin;
    byte gain;
    long offset, lastSample;
    float scale;
    union sensorDataType sensorData;
public:
    Hx711(byte dataPin, byte clkPin, byte gain = 128) {
        this->dataPin = dataPin;
        this->clkPin = clkPin;
        pinMode(dataPin, INPUT);
        pinMode(clkPin, OUTPUT);
        setGain(gain);
        scale = 1; // from app/user
        offset = 0; // upon init
    }

    // Channel selection is made by passing the appropriate gain:
    // - With a gain factor of 64 or 128, channel A is selected
    // - With a gain factor of 32, channel B is selected
    // The library default is "128" (Channel A).
    void setGain(byte gain = 128) {
        switch (gain) {
            case 128: // channel A, gain factor 128
                gain = 1;
                break;
            case 64: // channel A, gain factor 64
                gain = 3;
                break;
            case 32: // channel B, gain factor 32
                gain = 2;
                break;
        }
        digitalWrite(clkPin, LOW);
        readSensor();
    }

    bool isBusy() { return digitalRead(dataPin) != LOW; }

    // set the gain factor; takes effect only after a call to read()
    // channel A can be set for a 128 or 64 gain; channel B has a fixed 32 gain
    // depending on the parameter, the channel is also set to either A or B

    void setScale(float scale) { this->scale = scale; }
    float getScale() { return scale; }
    void setOffset(long offset) { this->offset = offset; }

```

CAPÍTULO 2. SENSORES

```
long getOffset() { return offset; }

long readSensor() {
    byte i;
    while(isBusy());
    noInterrupts(); // Disable interrupts.

    // read the 24 bits of data from the ADC
    sensorData.part[2] = shiftIn(dataPin, clkPin, MSBFIRST);
    sensorData.part[1] = shiftIn(dataPin, clkPin, MSBFIRST);
    sensorData.part[0] = shiftIn(dataPin, clkPin, MSBFIRST);

    // received 24 bits, but variable has 32 bits
    // 2s complement requires to extend the sign bit
    sensorData.part[3] = (sensorData.part[2] & 0x80) ? 0xFF : 0x00;

    // Set the channel and the gain factor for the next reading using the clock pin.
    for (i = 0; i < gain; i++) {
        digitalWrite(clkPin, HIGH);
        delayMicroseconds(1);
        digitalWrite(clkPin, LOW);
        delayMicroseconds(1);
    }
    interrupts(); // Enable interrupts
    return sensorData.val;
}

long getSample(byte numSamples = 10) {
    for (byte i = 0; i < numSamples; i++) {
        lastSample = 3 * lastSample / 4 + readSensor() / 4;
    }
    return lastSample;
}

double getValue(byte samples = 2) { return getSample(samples) - offset; }
float getWeight(byte samples = 2) { return getValue(samples) / scale; }
void zeros(byte samples = 40) { offset = getValue(samples); }

void suspend() {
    digitalWrite(clkPin, LOW);
    digitalWrite(clkPin, HIGH);
}
void awake() { digitalWrite(clkPin, LOW); }
};

Hx711* miniScale;

void setup() {
    miniScale = new Hx711(HX711.DATA_PIN, HX711.SCLK_PIN, 128);
    miniScale->setScale(402.f); // this value was obtained by calibrating the scale with known weights; see t

    Serial.begin(115200); while(!Serial);
    #if 0
    Serial.println(".:Weight Sensor/HX711.:");
    Serial.println("Before setting up the scale:");
    Serial.print("sample: \t\t");
    Serial.println(miniScale->readSensor()); // print a raw reading from the ADC
    Serial.print("average 10 samples: \t\t");
    Serial.println(miniScale->getValue(10)); // print the average of 20 readings from the ADC
    Serial.print("get weight in gr: \t\t");
    Serial.println(miniScale->getWeight(5)); // print the average of 5 readings from the ADC minus bias wei
    #endif
    miniScale->getValue(10);
    delay(100);
    miniScale->getWeight(5);
    miniScale->zeros(); // reset the scale to 0
    // Serial.print("offset: \t\t");
    // Serial.println(miniScale->getOffset() / miniScale->getScale()); //
```

```
}  
  
void loop() {  
  // miniScale->awake();  
  Serial.print(miniScale->getWeight(1));  
  Serial.print(",");  
  Serial.println(miniScale->getWeight(5));  
  // miniScale->suspend();           // put the ADC in sleep mode  
  delay(300);  
}
```

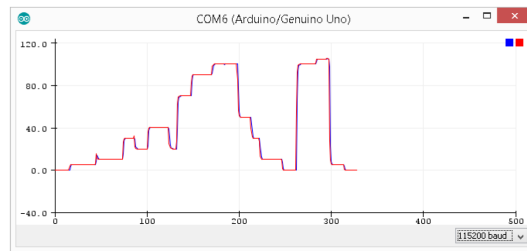


Figura 2.92: Variação do peso ao longo do ensaio.

Se os fios do sensor à placa breakout forem tocados com a mão, o valor lido será influenciado uma vez que os sinais nestas ligações ainda são analógicos e os fios não possuem blindagem. É importante que estas ligações estejam revestidas de material condutor, ligado a GND para reduzir interferências sobre a medida, à semelhança dos fios usados em áudio.

2.25 Caudal

Um caudalímetro é o sensor que permite medir um caudal numa canalização ou tubagem. O caudalímetro produz na sua saída um sinal elétrico proporcional ao fluxo de água que passa por ele.

O princípio de funcionamento do caudalímetro é semelhante ao das ventoinhas e motores, em que um sensor deteta o completar de uma revolução de um eixo através da fixação de um elemento ferromagnético passível de ser detetado por um sensor de efeito de Hall. No caso do caudalímetro, que tem de ser estanque para não permitir fugas de líquido, e é construído com um elemento ferro-magnético solidário com a uma borboleta imersos no tubo que gira proporcionalmente com o fluxo do caudal que se procura medir.

Os caudalímetros para além do sensor de efeito de Hall, possuem também o circuito condicionador de sinal integrado para poderem fornecer um sinal capaz de ser ligado diretamente a um sistema embebido e evitar a necessidade . Nas ligações de um caudalímetro são: VCC (fio vermelho), GND (fio preto) e o sinal que produz um pulso a cada passagem do íman pelo sensor.

O caudalímetro apresentado na figura 2.93 é capaz medir caudais entre 1 e 30 litros por minuto. O número de pulsos contados por unidade de tempo corresponde à frequência do sinal gerado pelo sensor. A frequência correspondente à medida de caudal é dada pela equação 2.6. Nesta equação, f é a frequência dos pulsos gerados e é medida em Hertz (Hz), Q é a quantidade de água por unidade de tempo, ou caudal, expresso em litros por minuto (L/m). Os termos da aproximação linear são indicados pelo fabricante no *datasheet*.

$$Q = \frac{f + 3}{5} [L/m] \quad (2.6)$$



Figura 2.93: Caudalímetro.

2.26 Gás

Em ambientes domésticos, industriais e espaços públicos onde existe o risco de haver contaminação do ar com diversas substâncias há necessidade de monitorizar a concentração dessas substâncias e dar indicação de alarme quando excedem um determinado limite.

Existem diversos tipos de sensores para esses fins, ilustrados na figura 2.94, nomeadamente para fumo, álcool, metano, monóxido e dióxido de carbono, hidrogénio, gás natural, e ozono. Estes sensores são conhecidos pelos prefixo MQ-x, sendo x o número identificador do sensor, resumidos na tabela 2.11. Todos os sensores funcionam com 5 V e consomem cerca de 150 mA, e têm duas saídas uma analógica que produz um sinal proporcional à concentração de gás, e outras digital que é ativa quando a quantidade de gás ultrapassa o valor limite para ser ligada na entrada de sinal de interrupção de um sistema embebido ou para ativar diretamente um extrator de ar via relé.



Figura 2.94: Sensores de concentração de gases.

Identificador	Tipo de sensor
MQ-2	Fumo
MQ-3	Álcool
MQ-4	Metano
MQ-5	Gás natural
MQ-6	Gás automóvel (GPL)
MQ-7	Monóxido de carbono
MQ-8	Hidrogénio
MQ-9	Gás combustível
MQ-135	Qualidade do ar

Tabela 2.11: Tipos de sensores de gases.

2.27 Distância

Os sensores de distância geram um sinal proporcional a um alvo sem que exista uma ligação física. As aplicações mais comuns para este tipo de sensores são os sensores de estacionamento nos automóveis, e linhas de montagem industriais.

A distância pode ser aferida usando métodos diferentes, nomeadamente radiação em infra-vermelhos, ultra-sons, e lasers, sendo cada um deles adequado a medir distâncias diferentes. No entanto todos os métodos baseiam o seu princípio de funcionamento na emissão de um sinal e na observação da sua reflexão produzida ao atingir o alvo, semelhante a um radar.

2.27.1 Infra-Vermelhos

A sua atenuação que um sinal com a distância é inversamente proporcional ao quadrado da distância ($P_{rx} \approx \frac{P_{tx}}{dist^2}$). Sabendo a relação entre a potência de emissão e de receção da radiação infra-vermelha é possível determinar a distância. A distância medida com este sensor é proporcional à quantidade de radiação recebida pelo sensor de infra-vermelhos. Uma desvantagem deste sensor é ser sensível à interferência da luz e temperatura (radiação infra-vermelha) ambientes.

Os infra-vermelhos são mais adequados a medir distâncias reduzidas, na ordem dos centímetros, uma vez que a potência de radiação emitida por um LED infra-vermelho convencional é baixa.

Listagem 2.22: Leitura e processamento dos dados do acelerómetro ADXL335

```
/* accelerometer test: arduino nano + adxl335 */

#define ACCLX_PIN A0           // x-axis of the accelerometer
#define ACCLY_PIN A1           // y-axis
#define ACCLZ_PIN A2           // z-axis (only on 3-axis models)

#define UPDATE_TIME      8
#define ACCL_SAMPLES     32
#define BIAS_SAMPLES     256
#define ACCL_THSHLD      1

#define DEMOMODE          1
// modes: 0 raw data, 1 acceleration, 2 velocity, 3 movement

class ADXL35 {
private:
    int xPin, yPin, zPin;
    int xBias, yBias, zBias;
    int xVal[ACCL_SAMPLES], yVal[ACCL_SAMPLES], zVal[ACCL_SAMPLES];
    int xAcc[2], yAcc[2], zAcc[2];
    int xVel[2], yVel[2], zVel[2];
    int xMov[2], yMov[2], zMov[2];
public:
    ADXL35() { xPin = ACCLX_PIN; yPin = ACCLY_PIN; zPin = ACCLZ_PIN; }
    void calibrate() {
        int i;
        long x = 0, y = 0, z = 0;
        for(i = 0; i < BIAS_SAMPLES; i++) {
            x += analogRead(xPin);
            y += analogRead(yPin);
            z += analogRead(zPin);
        }
    }
};
```

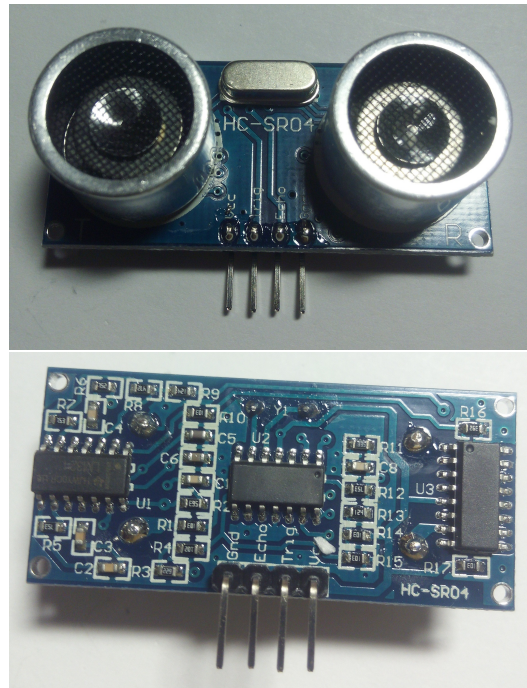


Figura 2.95: Parte da frente de um sensor de distância (esquerda) e parte traseira do sensor de distância (direita).

```

        z += analogRead(zPin);
    }
    xBias = x / BIAS.SAMPLES;
    yBias = y / BIAS.SAMPLES;
    zBias = z / BIAS.SAMPLES;
    for(i = 0; i < ACCLSAMPLES ; i++) {
        xVal[i] = xBias;
        yVal[i] = yBias;
        zVal[i] = zBias;
    }
}

void getData() {
    int i;
    // shift data
    for(i = ACCLSAMPLES - 1; i > 0; i--) {
        xVal[i] = xVal[i-1];
        yVal[i] = yVal[i-1];
        zVal[i] = zVal[i-1];
    }
    // get new sample
    xVal[0] = analogRead(xPin);
    yVal[0] = analogRead(yPin);
    zVal[0] = analogRead(zPin);
}

// get last sample from sensor
int getX() { return xVal[0]; }
int getY() { return yVal[0]; }
int getZ() { return zVal[0]; }

// get instant acceleration measurement
int getInstAccelX() { return xVal[0] - xBias; }
int getInstAccelY() { return yVal[0] - yBias; }
int getInstAccelZ() { return zVal[0] - zBias; }

// get average acceleration measurement
void doAvgAccel() {

```

```

    int i;
    long xAcum = 0, yAcum = 0, zAcum = 0;
    for(i = 0; i < ACCLSAMPLES; i++) {
        xAcum += xVal[i] - xBias;
        yAcum += yVal[i] - yBias;
        zAcum += zVal[i] - zBias;
    }
    // save previous value
    xAccel[1] = xAccel[0];
    yAccel[1] = yAccel[0];
    zAccel[1] = zAccel[0];
    // compute new averages
    xAccel[0] = xAcum / ACCLSAMPLES;
    yAccel[0] = yAcum / ACCLSAMPLES;
    zAccel[0] = zAcum / ACCLSAMPLES;
    // min-threshold
    if((xAccel[0] < ACCLTHSHLD) && (xAccel[0] > -ACCLTHSHLD)) {
        xAccel[0] = 0;
    }
    if((yAccel[0] < ACCLTHSHLD) && (yAccel[0] > -ACCLTHSHLD)) {
        yAccel[0] = 0;
    }
    if((zAccel[0] < ACCLTHSHLD) && (zAccel[0] > -ACCLTHSHLD)) {
        zAccel[0] = 0;
    }
}
int getAccelX() { return xAccel[0]; }
int getAccelY() { return yAccel[0]; }
int getAccelZ() { return zAccel[0]; }

// compute velocity
void doVel() {
    // keep last value
    xVel[1] = xVel[0];    yVel[1] = yVel[0];    zVel[1] = zVel[0];
    // compute new value
    xVel[0] += (xAccel[0] + xAccel[1]) ;
    yVel[0] += (yAccel[0] + yAccel[1]) ;
    zVel[0] += (zAccel[0] + zAccel[1]) ;
}

// compute movement
int getVelX() { return xVel[0]; }
int getVelY() { return yVel[0]; }
int getVelZ() { return zVel[0]; }
void doMov() {
    xMov[0] += (xVel[0] + xVel[1]) >> 1;
    yMov[0] += (yVel[0] + yVel[1]) >> 1;
    zMov[0] += (zVel[0] + zVel[1]) >> 1;
}
int getMovX() { return xMov[0]; }
int getMovY() { return yMov[0]; }
int getMovZ() { return zMov[0]; }
} accl;

char buf[80];

void setup() {
    // initialize the serial communications:
    Serial.begin(115200);
    while(!Serial);
    Serial.println("!!! Calibrating accelerometer. Do not move sensor !!!");
    accl.calibrate();
    Serial.println("Calibration finished.");
}

void loop() {
    // read sensor values for 3 axis
    accl.getData();

#ifdef DEMOMODE == 0
    sprintf(buf, "%d,%d,%d\n", accl.getInstAccelX(), accl.getInstAccelY(), accl.getInstAccelZ());

```

```
#elif DEMOMODE == 1
    accl.doAvgAccl();
    sprintf(buf, "%d,%d,%d\n", accl.getAcclX(), accl.getAcclY(), accl.getAcclZ());
#elif DEMOMODE == 2
    accl.doAvgAccl();
    accl.doVel();
    sprintf(buf, "%d,%d,%d\n", accl.getVelX(), accl.getVelY(), accl.getVelZ());
#elif DEMOMODE == 3
    accl.doAvgAccl();
    accl.doVel();
    accl.doMov();
    sprintf(buf, "%d,%d,%d\n", accl.getMovX(), accl.getMovY(), accl.getMovZ());
#endif
Serial.println(buf); // print comma separated values

// delay before next reading:
delay(UPDATETIME);
}
```

2.27.2 Ultra-Sons

O método de ultra-sons funciona com base num transdutor que emite uma onda sonora acima do limiar audível pelo ouvido humano (ultra-sons) e mede a intensidade do sinal reflectido. Este método permite alcançar distâncias superiores, cerca de metros, e oferece maior imunidade a interferências externas pois não existem muitas fontes de ruído na sua frequência de funcionamento.

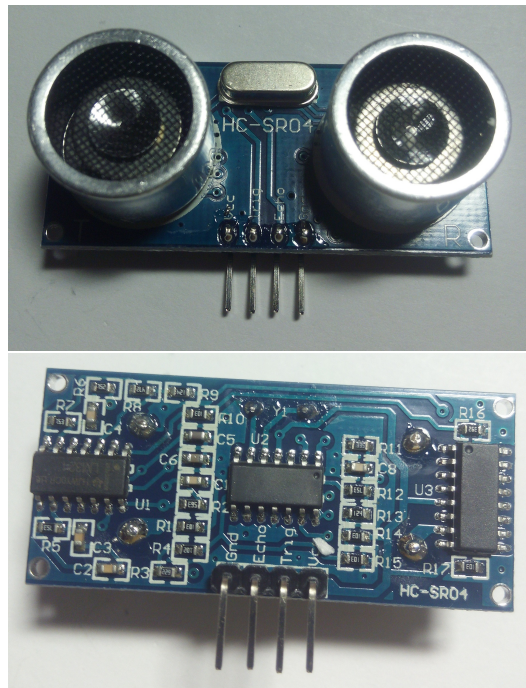


Figura 2.96: Parte da frente de um sensor de distância (esquerda) e parte traseira do sensor de distância (direita).

Figura 2.97: Ligação de um sensor de distância de ultra-sons a um Arduino Uno.

Listagem 2.23: Leitura e processamento dos dados do acelerómetro ADXL335

```

/* accelerometer test:  arduino nano + adxl335 */

#define ACCLX_PIN A0          // x-axis of the accelerometer
#define ACCLY_PIN A1          // y-axis
#define ACCLZ_PIN A2          // z-axis (only on 3-axis models)

#define UPDATE_TIME 8
#define ACCL_SAMPLES 32
#define BIAS_SAMPLES 256
#define ACCL_THRESHOLD 1

#define DEMO_MODE 1
// modes: 0 raw data, 1 acceleration, 2 velocity, 3 movement

class ADXL35 {
private:
    int xPin, yPin, zPin;
    int xBias, yBias, zBias;
    int xVal[ACCL_SAMPLES], yVal[ACCL_SAMPLES], zVal[ACCL_SAMPLES];
    int xAccl[2], yAccl[2], zAccl[2];
    int xVel[2], yVel[2], zVel[2];
    int xMov[2], yMov[2], zMov[2];
public:
    ADXL35() { xPin = ACCLX_PIN; yPin = ACCLY_PIN; zPin = ACCLZ_PIN; }
    void calibrate() {
        int i;
        long x = 0, y = 0, z = 0;
        for(i = 0; i < BIAS_SAMPLES; i++) {
            x += analogRead(xPin);
            y += analogRead(yPin);
            z += analogRead(zPin);
        }
        xBias = x / BIAS_SAMPLES;
        yBias = y / BIAS_SAMPLES;
        zBias = z / BIAS_SAMPLES;
        for(i = 0; i < ACCL_SAMPLES ; i++) {
            xVal[i] = xBias;
            yVal[i] = yBias;
            zVal[i] = zBias;
        }
    }

    void getData() {
        int i;
        // shift data
        for(i = ACCL_SAMPLES - 1; i > 0; i--) {
            xVal[i] = xVal[i-1];
            yVal[i] = yVal[i-1];
            zVal[i] = zVal[i-1];
        }
        // get new sample
        xVal[0] = analogRead(xPin);
        yVal[0] = analogRead(yPin);
        zVal[0] = analogRead(zPin);
    }
    // get last sample from sensor
    int getX() { return xVal[0]; }
    int getY() { return yVal[0]; }
    int getZ() { return zVal[0]; }
    // get instant acceleration measurement
    int getInstAcclX() { return xVal[0] - xBias; }
    int getInstAcclY() { return yVal[0] - yBias; }
    int getInstAcclZ() { return zVal[0] - zBias; }
    // get average acceleration measurement
    void doAvgAccl() {

```

```
int i;
long xAcum = 0, yAcum = 0, zAcum = 0;
for(i = 0; i < ACCLSAMPLES; i++) {
    xAcum += xVal[i] - xBias;
    yAcum += yVal[i] - yBias;
    zAcum += zVal[i] - zBias;
}
// save previous value
xAccl[1] = xAccl[0];
yAccl[1] = yAccl[0];
zAccl[1] = zAccl[0];
// compute new averages
xAccl[0] = xAcum / ACCLSAMPLES;
yAccl[0] = yAcum / ACCLSAMPLES;
zAccl[0] = zAcum / ACCLSAMPLES;
// min-threshold
if((xAccl[0] < ACCLTHSHLD) && (xAccl[0] > -ACCLTHSHLD)) {
    xAccl[0] = 0;
}
if((yAccl[0] < ACCLTHSHLD) && (yAccl[0] > -ACCLTHSHLD)) {
    yAccl[0] = 0;
}
if((zAccl[0] < ACCLTHSHLD) && (zAccl[0] > -ACCLTHSHLD)) {
    zAccl[0] = 0;
}

}
int getAcclX() { return xAccl[0]; }
int getAcclY() { return yAccl[0]; }
int getAcclZ() { return zAccl[0]; }

// compute velocity
void doVel() {
    // keep last value
    xVel[1] = xVel[0];    yVel[1] = yVel[0];    zVel[1] = zVel[0];
    // compute new value
    xVel[0] += (xAccl[0] + xAccl[1]) ;
    yVel[0] += (yAccl[0] + yAccl[1]) ;
    zVel[0] += (zAccl[0] + zAccl[1]) ;
}

// compute movement
int getVelX() { return xVel[0]; }
int getVelY() { return yVel[0]; }
int getVelZ() { return zVel[0]; }
void doMov() {
    xMov[0] += (xVel[0] + xVel[1]) >> 1;
    yMov[0] += (yVel[0] + yVel[1]) >> 1;
    zMov[0] += (zVel[0] + zVel[1]) >> 1;
}
int getMovX() { return xMov[0]; }
int getMovY() { return yMov[0]; }
int getMovZ() { return zMov[0]; }
} accl;

char buf[80];

void setup() {
    // initialize the serial communications:
    Serial.begin(115200);
    while(!Serial);
    Serial.println("!!! Calibrating accelerometer. Do not move sensor !!!");
    accl.calibrate();
    Serial.println("Calibration finished.");
}

void loop() {
    // read sensor values for 3 axis
    accl.getData();

    #if DEMOMODE == 0
        sprintf(buf, "%d,%d,%d\n", accl.getInstAcclX(), accl.getInstAcclY(), accl.getInstAcclZ());
```

```

#elif DEMOMODE == 1
    accl.doAvgAccl();
    sprintf(buf, "%d,%d,%d\n", accl.getAcclX(), accl.getAcclY(), accl.getAcclZ());
#elif DEMOMODE == 2
    accl.doAvgAccl();
    accl.doVel();
    sprintf(buf, "%d,%d,%d\n", accl.getVelX(), accl.getVelY(), accl.getVelZ());
#elif DEMOMODE == 3
    accl.doAvgAccl();
    accl.doVel();
    accl.doMov();
    sprintf(buf, "%d,%d,%d\n", accl.getMovX(), accl.getMovY(), accl.getMovZ());
#endif
Serial.println(buf); // print comma separated values

// delay before next reading:
delay(UPDATETIME);
}

```

2.27.3 Laser

O sensor de distâncias baseado em tecnologia laser também é conhecido como *Laser Range Finder*. O sinal gerado por um LED laser tem toda a sua energia concentrada num ponto, pelo que consegue atingir maiores distâncias que o LED de infra-vermelhos, entre metros e dezenas de metros. No caso deste sensor, em vez da intensidade da energia refletida pelo objeto alvo, o recetor conta o tempo entre o envio de um sinal e a receção da reflexão.

Figura 2.98: Sensor de distância baseado em radiação laser.

Figura 2.99: Ligação de um sensor de distância baseado em radiação laser a um Arduino Uno.

Listagem 2.24: Leitura e processamento dos dados do acelerómetro ADXL335

```

/* accelerometer test:  arduino nano + adxl335 */

#define ACCLX_PIN A0          // x-axis of the accelerometer
#define ACCLY_PIN A1          // y-axis
#define ACCLZ_PIN A2          // z-axis (only on 3-axis models)

#define UPDATE_TIME 8
#define ACCL_SAMPLES 32
#define BIAS_SAMPLES 256
#define ACCL_THRESH 1

#define DEMOMODE 1
// modes: 0 raw data, 1 acceleration, 2 velocity, 3 movement

class ADXL35 {
private:
    int xPin, yPin, zPin;
    int xBias, yBias, zBias;
    int xVal[ACCL_SAMPLES], yVal[ACCL_SAMPLES], zVal[ACCL_SAMPLES];
    int xAccl[2], yAccl[2], zAccl[2];
    int xVel[2], yVel[2], zVel[2];
    int xMov[2], yMov[2], zMov[2];

```

```
public:
ADXL35() { xPin = ACCLX_PIN; yPin = ACCLY_PIN; zPin = ACCLZ_PIN; }
void calibrate() {
    int i;
    long x = 0, y = 0, z = 0;
    for(i = 0; i < BIAS.SAMPLES; i++) {
        x += analogRead(xPin);
        y += analogRead(yPin);
        z += analogRead(zPin);
    }
    xBias = x / BIAS.SAMPLES;
    yBias = y / BIAS.SAMPLES;
    zBias = z / BIAS.SAMPLES;
    for(i = 0; i < ACCLSAMPLES ; i++) {
        xVal[i] = xBias;
        yVal[i] = yBias;
        zVal[i] = zBias;
    }
}

void getData() {
    int i;
    // shift data
    for(i = ACCLSAMPLES - 1; i > 0; i--) {
        xVal[i] = xVal[i-1] ;
        yVal[i] = yVal[i-1] ;
        zVal[i] = zVal[i-1] ;
    }
    // get new sample
    xVal[0] = analogRead(xPin);
    yVal[0] = analogRead(yPin);
    zVal[0] = analogRead(zPin);
}

// get last sample from sensor
int getX() { return xVal[0]; }
int getY() { return yVal[0]; }
int getZ() { return zVal[0]; }
// get instant acceleration measurement
int getInstAcclX() { return xVal[0] - xBias; }
int getInstAcclY() { return yVal[0] - yBias; }
int getInstAcclZ() { return zVal[0] - zBias; }
// get average acceleration measurement
void doAvgAccl() {
    int i;
    long xAcum = 0, yAcum = 0, zAcum = 0;
    for(i = 0; i < ACCLSAMPLES; i++) {
        xAcum += xVal[i] - xBias;
        yAcum += yVal[i] - yBias;
        zAcum += zVal[i] - zBias;
    }
    // save previous value
    xAccl[1] = xAccl[0];
    yAccl[1] = yAccl[0];
    zAccl[1] = zAccl[0];
    // compute new averages
    xAccl[0] = xAcum / ACCLSAMPLES;
    yAccl[0] = yAcum / ACCLSAMPLES;
    zAccl[0] = zAcum / ACCLSAMPLES;
    // min-threshold
    if((xAccl[0] < ACCLTHSHLD) && (xAccl[0] > -ACCLTHSHLD)) {
        xAccl[0] = 0;
    }
    if((yAccl[0] < ACCLTHSHLD) && (yAccl[0] > -ACCLTHSHLD)) {
        yAccl[0] = 0;
    }
    if((zAccl[0] < ACCLTHSHLD) && (zAccl[0] > -ACCLTHSHLD)) {
        zAccl[0] = 0;
    }
}

int getAcclX() { return xAccl[0]; }
int getAcclY() { return yAccl[0]; }
int getAcclZ() { return zAccl[0]; }
```



```

// compute velocity
void doVel() {
    // keep last value
    xVel[1] = xVel[0];      yVel[1] = yVel[0];      zVel[1] = zVel[0];
    // compute new value
    xVel[0] += (xAccl[0] + xAccl[1]) ;
    yVel[0] += (yAccl[0] + yAccl[1]) ;
    zVel[0] += (zAccl[0] + zAccl[1]) ;
}

// compute movement
int getVelX() { return xVel[0]; }
int getVelY() { return yVel[0]; }
int getVelZ() { return zVel[0]; }
void doMov() {
    xMov[0] += (xVel[0] + xVel[1]) >> 1;
    yMov[0] += (yVel[0] + yVel[1]) >> 1;
    zMov[0] += (zVel[0] + zVel[1]) >> 1;
}
int getMovX() { return xMov[0]; }
int getMovY() { return yMov[0]; }
int getMovZ() { return zMov[0]; }
} accl;

char buf[80];

void setup() {
    // initialize the serial communications:
    Serial.begin(115200);
    while(!Serial);
    Serial.println("!!! Calibrating accelerometer. Do not move sensor !!!");
    accl.calibrate();
    Serial.println("Calibration finished.");
}

void loop() {
    // read sensor values for 3 axis
    accl.getData();

    #if DEMOMODE == 0
        sprintf(buf, "%d,%d,%d\n", accl.getInstAcclX(), accl.getInstAcclY(), accl.getInstAcclZ());
    #elif DEMOMODE == 1
        accl.doAvgAccl();
        sprintf(buf, "%d,%d,%d\n", accl.getAcclX(), accl.getAcclY(), accl.getAcclZ());
    #elif DEMOMODE == 2
        accl.doAvgAccl();
        accl.doVel();
        sprintf(buf, "%d,%d,%d\n", accl.getVelX(), accl.getVelY(), accl.getVelZ());
    #elif DEMOMODE == 3
        accl.doAvgAccl();
        accl.doVel();
        accl.doMov();
        sprintf(buf, "%d,%d,%d\n", accl.getMovX(), accl.getMovY(), accl.getMovZ());
    #endif
    Serial.println(buf); // print comma separated values

    // delay before next reading:
    delay(UPDATETIME);
}

```

2.28 Radiação Nuclear

Figura 2.100 fotodiodo como sensor de radiação alfa, beta e gama.

O tudo contém um filamento num espaço preenchido com gás ionizante (Argon). Quando uma partícula ionizante (radiação) atravessa o tudo, faz com um átomo de gás ionizado se desloque do ânodo central para o cátodo que reveste a parede do cilindro, fechando o circuito num breve instante. Este fenómeno repete-se para cada partícula no espaço do sensor. Desta forma, a quantidade de radiação é medida pela quantidade de partículas ionizantes. Para que os eletrões “saltem” é necessário ter o tubo polarizado com uma tensão de cerca de 350 V CC.



Figura 2.100: Fotodiodo tipo PIN BPW34.

an average gamma radiation event produces only around 10 nanoamps for about 50 microseconds, a lot of amplification (100,000x), low noise, and high bandwidth are a must.

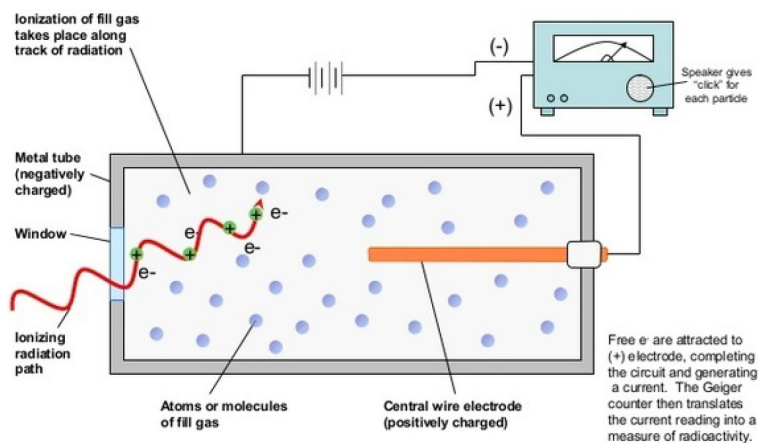


Figura 2.101: Detetor Geiger-Müller baseado em tubo de gás.

Existem vários objetos no quotidiano que emitem radiação que pode ser medida, nomeadamente em fotocopiadoras, detetores de fumo, pedras de granito e bananas⁵.

⁵*Banana Equivalent Dose* corresponde à radiação emitida pelos isotopos de potássio-40 existente nas bananas. Esta radiação corresponde a cerca de $0,1 \mu Sv$ não é nociva, servindo apenas como referência de comparação.

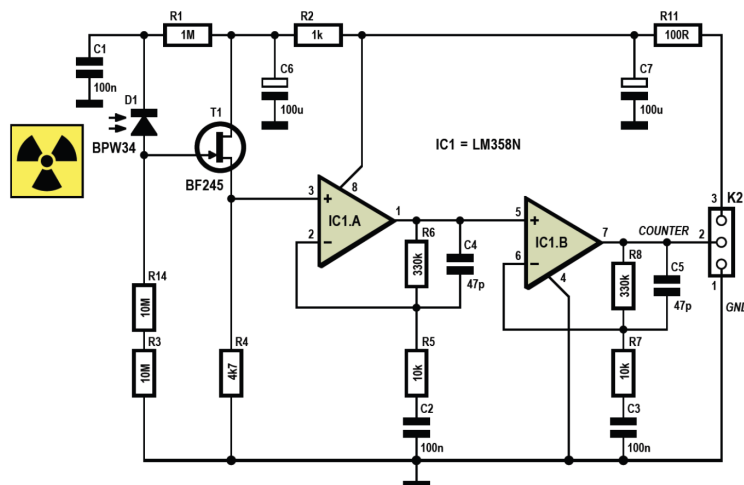


Figura 2.102: Detetor de radiação baseado em tecnologia semicondutora (PIN).

As pedras de granito tem na sua composição vestígios de urânio e tório, que as torna radioativas.

2.29 Atividade Sísmica

O geofone é um sensor que capta ruído de muito baixa frequência (apx 5-100 Hz) no solo. Este sensor é um transdutor de converte vibrações so solo em tensão, e é usado para captar atividade sísmica e monitorização de atividade mineira. O geofone pode ser construído com base numa bobina acoplada a um íman permanente, de forma semelhante a um microfone, ou com base em tecnologia piezoelétrica. Quando o corpo de um geofone é exposto a vibrações, a bobina no seu interior desloca-se ao longo do íman e gera uma tensão aos seus terminais.

Dada a baixa largura de banda do sinal, não é possível extrair muita informação apenas com um sensor, pelo que a aplicação típica são realizadas medidas em simultâneo de vários sensores equidistantes para se determinar qual a direção e a intensidade de propagação do sinal ao longo do tempo. Para além de detetar a atividade sísmica proveniente do subsolo, os geofones são também usados na caraterização do solo, onde uma onda ou batimento é injetado no solo e os geofones são usados para caraterizar a propagação dessa onda. Diferentes tipos de solo vão ter diferentes tipos de resposta.

Para distâncias curtas entre geofones podem-se ligar vários geofones ao mesmo sistema embebido através de cabos. No entanto quando as distâncias estão na ordem dos quilómetros, não é prático estender cabos de ligação e a forma de criar um sistema é ter unidades autónomas que registam a ati-

CAPÍTULO 2. SENSORES

vidade captada pelo sensor independentemente. Para garantir que os dados recolhidos permitem representar a onda propagada no solo é necessário que as amostras das várias unidades sejam adquiridas no mesmo instante de tempo. Para isso podem-se usar recetores GPS que permitem obter a hora atual com grande precisão e sincronizar as várias unidades com exatidão.

Em termos de ligação de um geofone a um sistema embebido, ele trabalha em muito baixas frequências, abaixo dos 100 Hz, e tem uma impedância na ordem dos 100-2k Ω . O geofone gera tensões de muito baixa amplitude, na ordem dos μV , pelo que é necessário adicionar um circuito amplificador de sinal, por exemplo baseado em transistores do tipo J-FET. Como o ritmo de amostragem necessário é baixo, pode-se fazer algum pre-processamento no próprio sistema embebido, nomeadamente o cálculo do espectro do sinal (FFT). A figura 2.103 mostra o aspeto de um geofone e dois tipos de caixa para o acomodar. Uma caixa serve para maximizar a condução das vibrações em solo duro e outro para solo macio.



Figura 2.103: Geofone (esquerda) e respetivo suporte para condução das vibrações do solo ao corpo do sensor (direita).

2.30 Sinais Fisiológicos

O desporto de alta competição, cuidados de saúde e atividades de grande concentração e stress são as áreas onde é necessário quantificar diversos parâmetros do funcionamento do corpo humano, tais como: ritmo cardíaco, pressão arterial, electrocardiograma, electroencefalograma, electromiografia, respiração, movimento e deslocação (acelerometria). Existem aparelhos portáteis e autónomos para aferir algumas destas medidas. A figura 2.104 mostra um aparelho de medição de pressão arterial comercial em utilização.

Neste capítulo são apresentados alguns métodos biométricos, não invasivos, para extração de informação acerca da atividade cardíaca, muscular, respiratória e cerebral de um indivíduo.

Por motivos de segurança, alimente os circuitos descritos nesta seção com pilhas ou baterias, em vez de adaptadores de rede elétrica. Embora existam fontes de alimentação próprias para este tipo de sistemas (fontes de alimentação isoladas/médicas), recomenda-se a ligação a um computador alimentado a bateria.

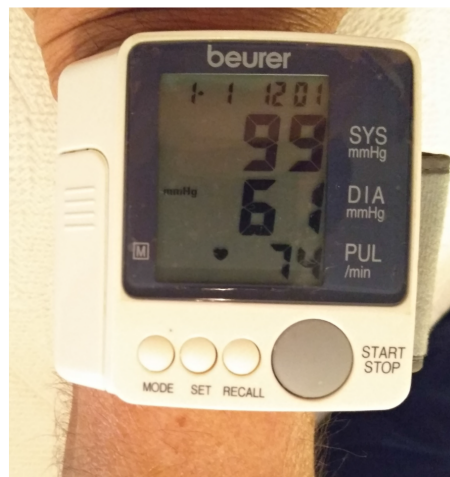


Figura 2.104: Medidor de pressão arterial comercial.

2.30.1 Foto-Pletismograma

A fotoplestismografia é uma técnica que permite aferir o volume de sangue existente nos tecidos.

A figura 2.105 mostra o esquema de funcionamento do sensor de PPG

A figura 2.106

Listagem 2.25: hr_sensor.ino

```
// Pulse Monitor Test Script
int sensorPin = 0;
double alpha = 0.75;
```

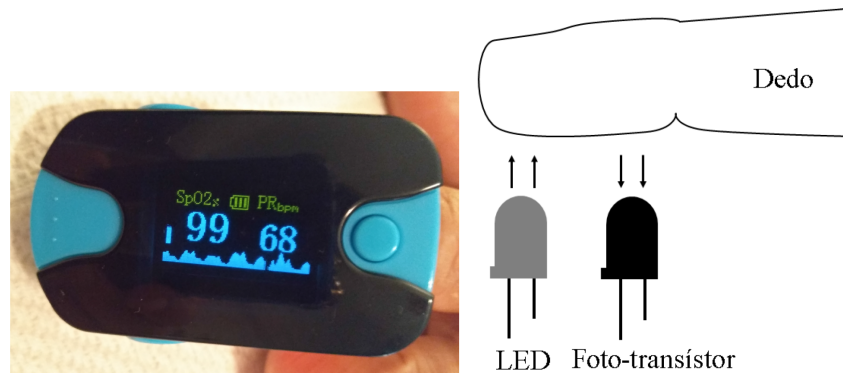


Figura 2.105: Oxímetro de pulso para medição do ritmo cardíaco (esquerda) e esquema de funcionamento do sensor PPG.

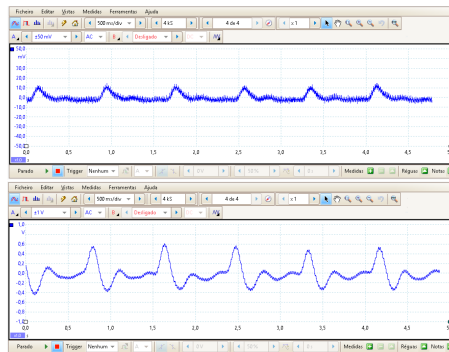


Figura 2.106: Oxímetria de pulso usando um LED e um fotodiodo.

```
int period = 100;
double change = 0.0;
double minval = 0.0;
void setup ()
{
  Serial.begin (9600);
}
void loop ()
{
  static double oldValue = 0;
  static double oldChange = 0;

  int rawValue = analogRead (sensorPin);
  double value = alpha * oldValue + (1 - alpha) * rawValue;

  Serial.print (rawValue);
  Serial.print (",");
  Serial.println (value);
  oldValue = value;

  delay (period);
}
```

A biblioteca existente em <https://github.com/oxullo/Arduino-MAX30100> oferece a definição da classe de objecto MAX30100.

A configuração do modo de funcionamento do sensor é feita na função `setup()` do Sketch Arduino, através da chamada dos métodos da instância do



Figura 2.107: ecg sensor2.

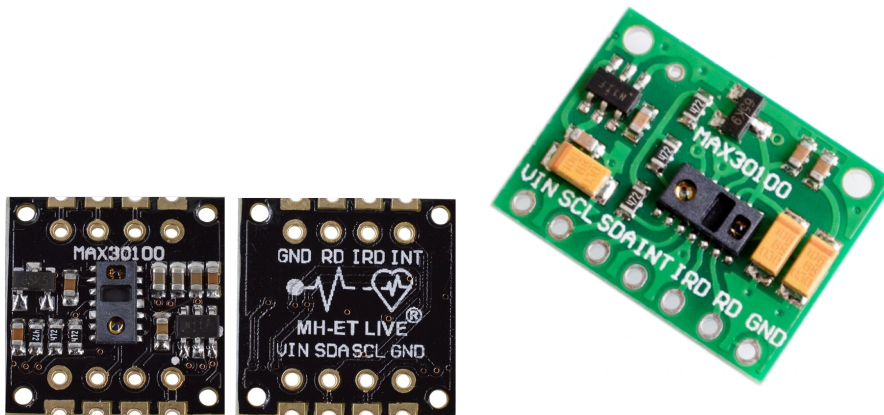


Figura 2.108: ecg sensor MAX30100.

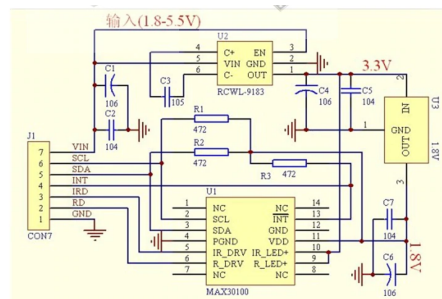


Figura 2.109: Circuito para o sensor MAX30100.

objeto do sensor:

As amostras dos sinais vermelho (RED) e infravermelho (IR) são obtidas através do método `update()`, e registadas nos atributos é

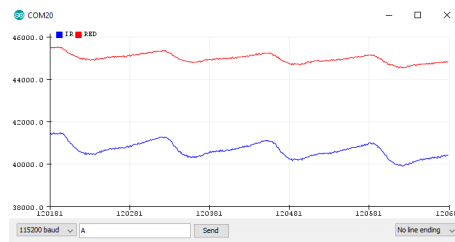


Figura 2.110: ecg sensor MAX30100.

2.30.2 Eletrocardiograma

A ativação dos músculos do coração é feita através de pulsos elétricos, que se propagam pelo corpo. Embora de muito baixa amplitude estes sinais podem ser adquiridos por circuitos eletrônicos e observados. Como em qualquer meio, existem perdas e atrasos na propagação de sinal ao longo do corpo, de modo que o potencial é observado no peito, nos pulsos e nas pernas. A figura 2.111 mostra os pontos usados mais facilmente para obter um eletrocardiograma (ECG).

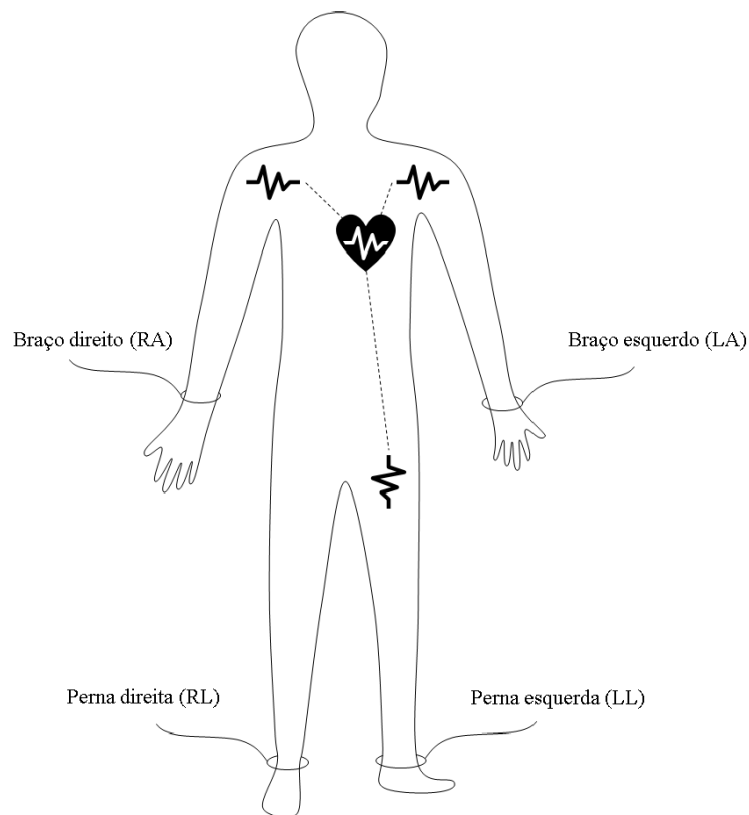


Figura 2.111: Propagação do sinal elétrico cardíaco ao longo do corpo.

A figura 2.112 mostra as diferentes componentes do sinal do coração que

pode ser medido através da diferença de potencial entre os diversos pontos do corpo humano.

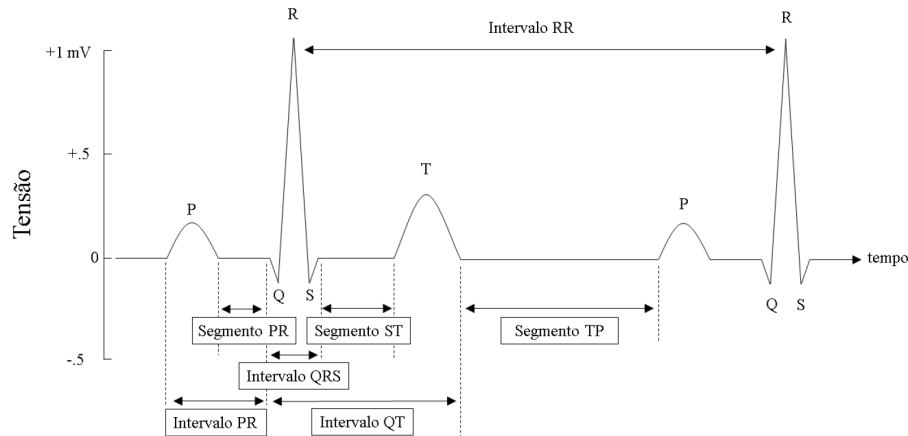


Figura 2.112: Morfologia do sinal do coração.

A figura 2.116 mostra um circuito para aquisição de ECG nos pulsos e na perna direita. Este circuito é baseado num integrado da

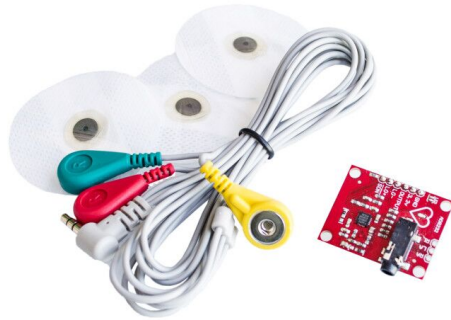


Figura 2.113: Ritmo cardíaco via eletrocardiograma.

2.30.3 Eletroencefalograma

O Eletroencefalograma (EEG) é uma medida da atividade cerebral obtida através do potencial elétrico medido em diversos pontos do crânio. A figura 2.114 mostra o modelo para colocação dos eletrodos no crânio.

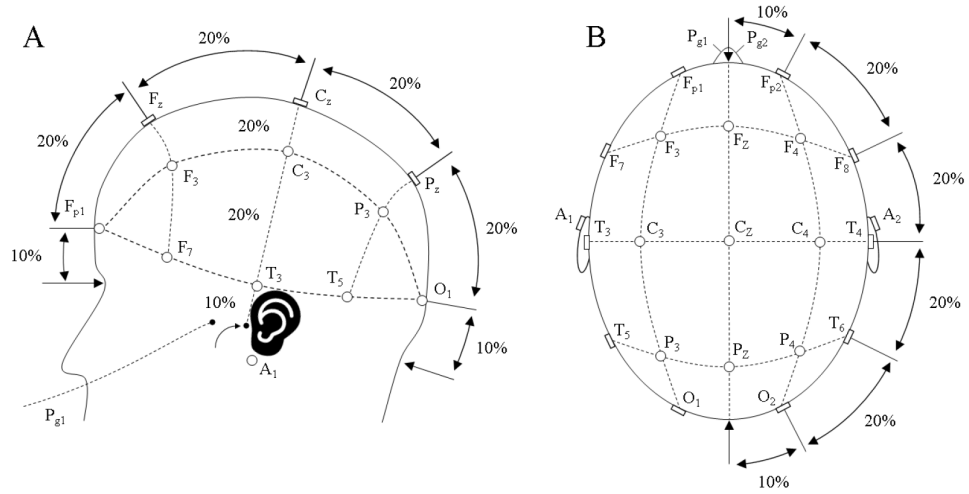


Figura 2.114: Mapa de posicionamento dos eletrodos para obtenção de EEG.

Figura 2.115: Sensor EEG.

2.30.4 Eletromiografia

EMG - medição da dilatação dos vasos sanguíneos.

Figura 2.116: ecg sensor via eletromiografia.

2.30.5 Respiração

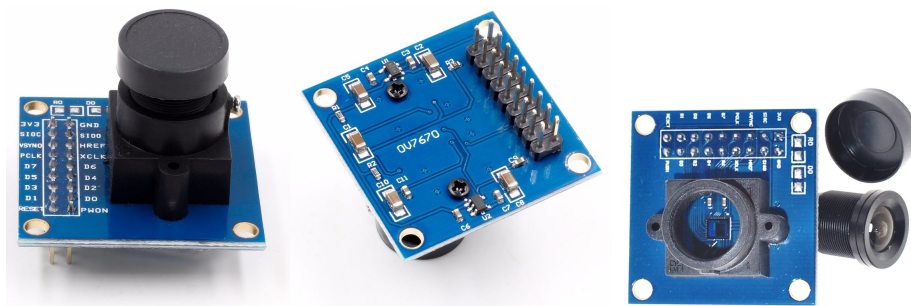


Figura 2.117: Câmera OV7670.

2.31 Imagem

A captação de imagem é um tipo de sensorização que mais informação oferece e que pode ser aplicado em vários âmbitos, desde o simples registo de observações até à extracção automática de informação sobre o estado de crescimento de uma plantação ou reconhecimento de uma pessoa. Um sensor de imagem, também conhecido como *Charge-Coupled Device* ou (CCD), transforma os fotões da luz incidente na sua superfície numa carga eléctrica passível de ser quantificada e transformada numa representação digital sob a forma de uma matriz de pontos vermelhos, verdes e azuis. Cada ponto tem a sua cor codificada como o valor das componentes das três componentes de cor. Quanto mais pontos existirem no sensor mais detalhe terá a imagem gerada.

As máquinas fotográficas convencionais possuem habitualmente uma interface USB e por isso são difíceis de adaptar a um sistema embebido como um Arduino. Existem câmeras mais simples que permitem esse tipo de ligação, nomeadamente a OV7670 da OmniVision, apresentada na figura 2.117. Esta câmara tem 640x480 *pixels* de resolução e possui um módulo processador digital de sinal capaz de manipular diretamente a imagem, aliviando o sistema embebido desse processamento.

Algumas características desta câmara são:

- CCD
- Sensibilidade
- etc...

Esta câmara é ligada através de uma interface paralela de 8 bits. A figura 2.119 mostra como ligar esta câmara a uma Arduino Uno a funcionar a 3,3 V.

A listagem 2.26 mostra o conteúdo do Sketch para adquirir uma imagem da camera. A imagem recebida é enviada pela porta série.

Para demonstrar a programação da câmara, foi realizado-se o circuito da figura 2.119

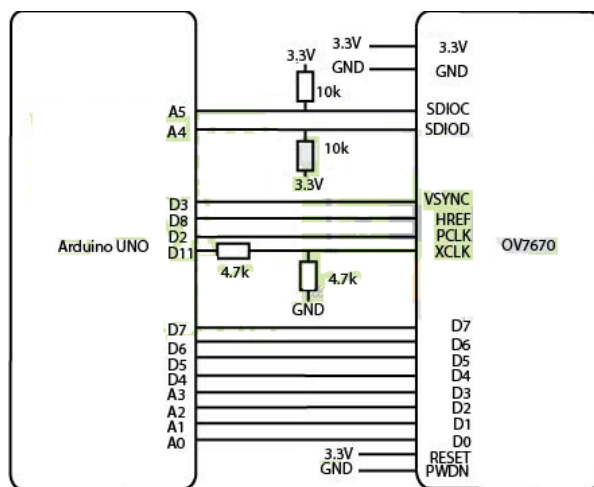


Figura 2.118: Ligação da câmera OV7670 a um Arduino.

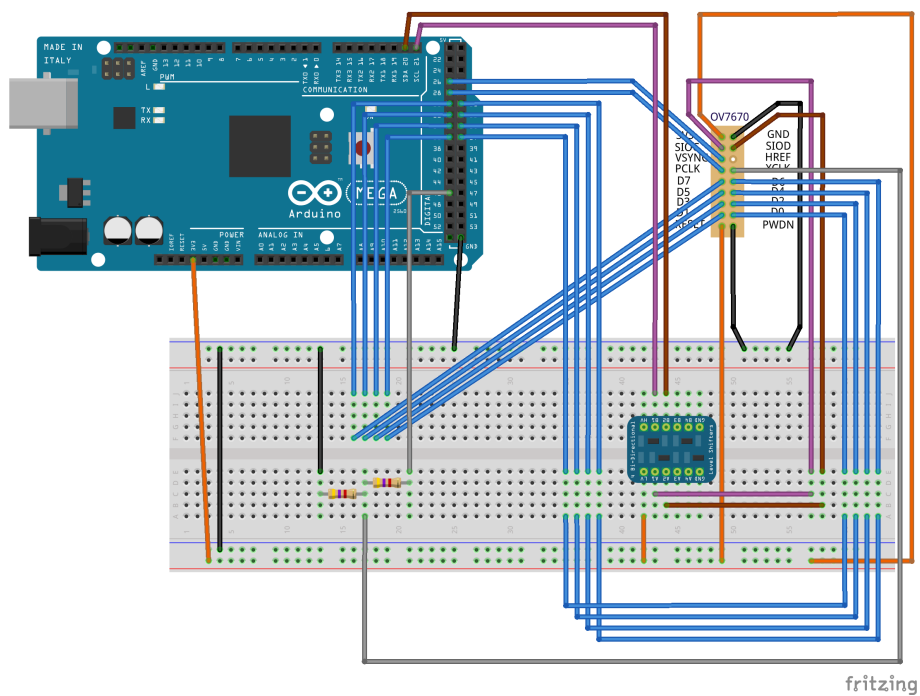


Figura 2.119: Circuito de demonstração da câmera OV7670.

Listagem 2.26: ov7670.ino

```

/* code taken from ComputerNerd/ov7670-no-ram-arduino-uno
   and changed a little

   OV7670 NO FIFO
   Arduino Mega 2560

*/

#include "ov7670.h"

/* Configuration: this lets you easily change between different resolutions
   You must only uncomment one
   no more no less*/
// #define useVga
// #define useQvga
#define useQqvga

inline void serialWrB(uint8_t dat) {
    while (!(UCSR0A & (1 << UDRE0))); // wait for byte to transmit
    UDR0 = dat;
    while (!(UCSR0A & (1 << UDRE0))); // wait for byte to transmit
}

void StringPgm(const char * str) {
    if (!pgm_read_byte_near(str))
        return;
    do {
        serialWrB(pgm_read_byte_near(str));
    } while (pgm_read_byte_near(++str));
}

void captureImg(uint16_t wg, uint16_t hg) {
    uint16_t lg2;
    #ifdef useQvga
        uint8_t buf[640];
    #elif defined(useQqvga)
        uint8_t buf[320];
    #endif
    StringPgm(PSTR("RDY"));
    // wait for vsync
    while (!(PINA & _BV(4))); // wait for high
    while ((PINA & _BV(4))); // wait for low
    #ifdef useVga
        while (hg--) {
            lg2 = wg;
            while (lg2--) {
                while ((PINA & _BV(6))); //wait for low
                UDR0 = PINC;
                while (!(PINA & _BV(6))); //wait for high
            }
        }
    #elif defined(useQvga)
        /*We send half of the line while reading then half later */
        while (hg--) {
            uint8_t*b = buf, *b2 = buf;
            lg2 = wg / 2;
            while (lg2--) {
                while ((PINA & _BV(6))); //wait for low
                *b++ = PINC;
                while (!(PINA & _BV(6))); //wait for high
                while ((PINA & _BV(6))); //wait for low
                *b++ = PINC;
                UDR0 = *b2++;
                while (!(PINA & _BV(6))); //wait for high
            }
        }
        /* Finish sending the remainder during blanking */
        lg2 = wg / 2;
        while (!(UCSR0A & (1 << UDRE0))); //wait for byte to transmit
        while (lg2--) {
            UDR0 = *b2++;
            while (!(UCSR0A & (1 << UDRE0))); //wait for byte to transmit
        }
    }
}

```

CAPÍTULO 2. SENSORES

```

#else
/* This code is very similar to quga sending code except we have even more blanking time to take advantage
while (hg-->0) {
    uint8_t*b = buf, *b2 = buf;
    lg2 = wg / 5;
    while (lg2-->0) {
        while ((PINA & _BV(6))); //wait for low
        *b++ = PINC;
        while (!(PINA & _BV(6))); //wait for high
        while ((PINA & _BV(6))); //wait for low
        *b++ = PINC;
        while (!(PINA & _BV(6))); //wait for high
        while ((PINA & _BV(6))); //wait for low
        *b++ = PINC;
        while (!(PINA & _BV(6))); //wait for high
        while ((PINA & _BV(6))); //wait for low
        *b++ = PINC;
        while (!(PINA & _BV(6))); //wait for high
        while ((PINA & _BV(6))); //wait for low
        *b++ = PINC;
        UDR0 = *b2++;
        while (!(PINA & _BV(6))); //wait for high
    }
    /* Finish sending the remainder during blanking */
    lg2 = 320 - (wg / 5);
    while (!(UCSR0A & (1 << UDRE0))); //wait for byte to transmit
    while (lg2-->0) {
        UDR0 = *b2++;
        while (!(UCSR0A & (1 << UDRE0))); //wait for byte to transmit
    }
}
#endif
}

void setup() {
    cli(); //disable interrupts
    // Setup the 8mhz PWM clock
    // OV7670 XCLK => (through level shift!) => pin 46 = PL3(OC5A), output
    DDRL |= _BV(3);
    TCCR5A = _BV(COM5A0) | _BV(WGM51) | _BV(WGM50);
    TCCR5B = _BV(WGM53) | _BV(WGM52) | _BV(CS50);
    OCR5A = 0; // (F_CPU)/(2*(X+1))
    // OV7670 VSYNC => pin 26 = PA4, input
    DDRA &= ~_BV(4);
    // OV7670 PCLK => pin 28 = PA6, input
    DDRA &= ~_BV(6);
    // OV7670 D7-D0 => pin 30-37 = PC7-PC0, input
    DDRC = 0;
    _delay_ms(3000);
    // set up twi for 100khz
    TWSR &= ~3; // disable prescaler for TWI
    TWBR = 72; // set to 100khz
    // enable serial
    UBRROH = 0;
    UBRROL = 3; // 0 = 2M baud rate. 1 = 1M baud. 3 = 0.5M. 7 = 250k 207 is 9600 baud rate.
    UCSRA |= 2; // double speed async
    UCSRB = (1 << RXEN0) | (1 << TXEN0); // enable receiver and transmitter
    UCSRC = 6; // async USART, 1 stop bit, 8-bit character size, no parity bits
    camInit();
#ifdef useVga
    setResolution(VGA);
    setColorSpace(BAYER_RGB);
    wrReg(REG_CLKRC, 63);
#elif defined(useQvga)
    setResolution(QVGA);
    setColorSpace(YUV422);
    wrReg(REG_CLKRC, 20);
#else
    setResolution(QQVGA);
    setColorSpace(YUV422);
    wrReg(REG_CLKRC, 4);
#endif
}

```

2.31. IMAGEM

```
/* If you are not sure what value to use here for the divider (register 0x11)
   Values I have found to work raw vga 25 qqvga yuv422 12 qvga yuv422 21
   run the commented out test below and pick the smallest value that gets a correct image */
void loop() {
  /* captureImg operates in bytes not pixels in some cases pixels are two bytes per pixel
     So for the width (if you were reading 640x480) you would put 1280 if you are reading yuv422 or rgb565 */
  /*uint8_t x=63;*/Uncomment this block to test divider settings note the other line you need to uncomment
  do{
    wrReg(REG.CLKRC,x);
    _delay_ms(1000);*/
#ifdef useVga
  captureImg(640, 480);
#elif defined(useQvga)
  captureImg(320 * 2, 240);
#else
  captureImg(160 * 2, 120);
#endif
  //}while(--x);*/Uncomment this line to test divider settings
}
```


3

Atuadores

Os atuadores são os componentes que ao receberem um sinal de controlo provocam uma manifestação física. Os atuadores são tão variados quantos os tipos de manifestações físicas existentes, nomeadamente emissão de luz, ondas sonoras, variação de temperatura, ativação de circuitos electrónicos e eléctricos de potência para diversos fins, válvulas hidráulicas e pneumáticas, e motores.

Neste capítulo são abordados diversos tipos de atuadores, o seu princípio de funcionamento, como são ligados a um sistema embebido e programados.

3.1 Atuador Básico

O controlo do tipo liga-desliga, ou ON-OFF, de um atuador é feito através da imposição de um valor num porto de E/S, e no uso de um circuito condicionador de sinal para fazer a adaptação dos sinais entre o sistema embebido e o atuador a controlar. Este tipo de controlo é adequado a diversos atuadores explicados nas seções seguintes deste capítulo são controlados desta forma, e podem ser programados da mesma forma.

A listagem 3.1 mostra o Sketch para demonstração da classe `basicAtuator`, usada para realizar ligar e desligar uma carga genérica ligada num pino de E/S. Este Sketch pode ser usado com vários atuadores apresentados neste capítulo.

Listagem 3.1: Listagem da classe `basicAtuator`

```
// basicAtuator class
// all times are in ms

class basicAtuator {
private:
    uint32_t t0, t1, t2, now;
    byte pin; // output pin
    bool activate; // logic level
    bool state; // current state HIGH/LOW
public:
    // default atuator ⌘ the built-in LED
    basicAtuator(byte pin = 13, bool activate = HIGH) {
        this->pin = pin;
        pinMode(pin, OUTPUT);
    }
    void setON() {
        state = activate;
        digitalWrite(pin, activate);
    }
    void setOFF() {
        digitalWrite(pin, !activate);
        state = !activate;
    }
    bool getState() {
        return this->state;
    }
    void setLevel(byte level) { // PWM where available
        analogWrite(pin, level);
        if(level == 0) {
            state = LOW;
        }
        else {
            state = HIGH;
        }
    }
    void setBlink(int onTime, int totalTime) {
        // if(onTime > totalTime) abort();
        t0 = millis(); // cycle start
        t1 = onTime;
        t2 = totalTime;
        digitalWrite(pin, activate);
        state = activate;
    }
    void task() {
        if(t1 == 0 && t2 == 0) return;
        now = millis();
        if(now >= t0 + t1) { //
            digitalWrite(pin, activate);
            state = activate;
        }
    }
}
```

```

        if(now >= t0 + t2){
            t0 = now;
            digitalWrite(pin, !activate);
            state = !activate;
        }
    }
};

basicAtuator led = basicAtuator(13); // pi D13, active = HIGH

void setup ()
{
    // nothing to do here.
    // Object's construtctor did that already.
}

void loop () {
    // old fashion blink for a while
    for(int i = 0; i < 10; i++) {
        led.setON();
        delay (500);
        led.setOFF();
        delay (1000);
    }

    // now let the object deal with it
    led.setBlink(500, 1000);
    while(1) {
        led.task();
        delay(10);
    }
}

```

A classe `basicAtuator` recebe no seu construtor o pino de saída digital, o nível de atuação (0 ou 1), e oferece os métodos `On()`, `Off()`, e `Pulse()`, que permitem ligar, desligar e ativar o atuador momentaneamente.

Os atributos privados da classe `pin` e `activate` guardam os valores do pino associado à instância do objeto, e estado de ativação, respetivamente. Por omissão de especificação do parâmetro no construtor, o objeto assume que o atuador é ativo quando a nível lógico 1 (HIGH), e desativo quando 0 (LOW). O atributo `state` guarda o estado atual do objeto, devolvido pelo método `getState()`, evitando ter de registar esse dado no conjunto de variáveis da aplicação.

Os métodos `setON()` e `setOFF()` são usados para ligar e desligar a carga, respetivamente. A classe possui ainda métodos para atuadores intermitentes `setBlink(int onTime, int totalTime)`, onde é dado o tempo em que o atuador se encontra ligado, e o tempo total de repetição do ciclo. Ambos os tempos são considerados em milissegundos. Atuadores que possam ser controlados por PWM podem servirem-se do método `setLevel(byte level)`, caso existam pinos disponíveis no Arduino.

3.2 Indicadores Visuais / Luminosos

Os atuadores luminosos servem para dar indicação, ou sinalização visual, quer através de um ponto luminoso, ou símbolo gráfico iluminado, e que tem associado um significado. Os emissores luminosos podem ser de fabricados com tecnologias diferentes, correspondendo a diferentes tipos de geração, sendo os mais habituais:

- Estado sólido / LED / *Organic Light Emitting Diode* (OLED) - Baseados em material semicondutor. Podem também servir para transmissão de informação, por exemplo infra-vermelhos e laser, em comandos remotos de equipamento audio-visual e fibra-ótica.
- Eletroluminescente - Material emite radiação luminosa quando na presença de um campo elétrico elevado. São usados para decoração dada a baixa potência de radiação.
- Fluorescente - Descarga elétrica em vapor de mercúrio dentro de um vidro. Usado em lâmpadas fluorescentes.
- Descarga de gás - Descarga elétrica aplicada num gás ionizado dentro de um vidro. É usado em lâmpadas neon e xenon;
- Incandescentes - cada vez menos frequentes por causa do seu elevado consumo energético, e tempo de vida curto, quando comparados com os LEDs, ainda são usados em situações onde é necessária uma potência de radiação elevada, ou o material semicondutor é susceptível de ser danificado devido a radiação, por exemplo em semáforos rodoviários e aviação.

Em iluminação e em imagem existe o conceito de temperatura de cor, e indicada em graus Kelvin (K), e representada na figura 3.1. Este conceito está associado à cor da radiação emitida por um corpo negro quando submetido a diferentes temperaturas. Na prática ele pode ser observado aquecendo um bloco de grafite. À temperatura de 1000 K o corpo exibe uma cor avermelhada, e à medida que a temperatura continua a subir, o corpo começa a radiar uma cor alaranjada, e por volta dos 6000 K tem a cor branca. Continuando a aumentar a temperatura o corpo começa a radiar a cor azul por volta dos 12000 K. Esta gama de cores é próxima das cores das estrelas no céu. Este conceito é contrário à noção popular de que o frio corresponde ao azul e o quente ao vermelho, tal é apresentado em equipamentos de climatização, e nas torneiras dos lavatórios.

Na especificação de indicadores e radiadores luminosos importa ter em mente a Lei do Inverso do Quadrado da Distância, que estabelece que a

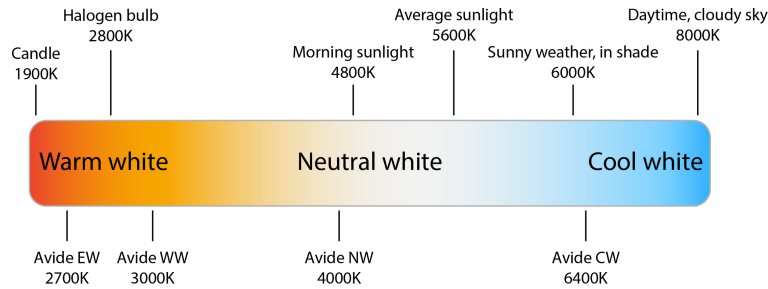


Figura 3.1: Escala de tons para diferentes temperaturas de cor.

intensidade de radiação de uma onda eletromagnética em espaço livre é atenuada proporcionalmente com o inverso do quadrado da distância percorrida pela onda: $A \approx \frac{1}{d^2}$, tal como ilustrado na figura 3.2.

3.2.1 Fio Eletroluminescente

O fio eletroluminescente é constituído por um fio condutor, em cobre, revestido por fósforo, com um segundo condutor muito fino por cima da camada de fósforo, e é capaz de emitir radiação visível quando na presença de um campo elétrico elevado (70-180 V CA). A cor radiada pelo fio é determinada pelo plástico translúcido que reveste os fios de cobre. O par de condutores é equivalente aos eletrodos de um condensador, em que o fósforo no meio funciona como eletrólito. Ao ser estabelecida uma diferença de potencial alternada entre os dois condutores vai provocar a carga e descarga do condensador. Esta carga e descarga vai fazer o fósforo libertar radiação luminosa.

Embora a tensão seja elevada, a corrente é muito pequena ($< 1 \text{ mA}$). A figura 3.4 ilustra a construção e ligação de um fio eletroluminescente à esquerda, e à direita o aspeto de um fio em funcionamento. A radiação emitida num determinado ponto está na mesma ordem de grandeza da de um LED. Por ser fino (3 mm), longo (1-5 m) e flexível é muito usado em instalações artísticas, tais como roupa e decoração de espaços e equipamentos.

A alta tensão alternada necessária para fazer o fio iluminar-se é conseguida com um circuito oscilador (700-2 kHz) de baixa tensão ligado na saída de um pequeno transformador, também conhecido como circuito inversor. O circuito oscilador requer muito pouca corrente e pode ser controlado com um sinal de E/S de um sistema embebido. Não existe polaridade no fio uma vez que é usada tensão alternada. Recomenda-se cuidado a isolar as ligações de alta tensão, sendo habitualmente realizadas fora do sistema embebido e na proximidade do fio eletroluminescente.

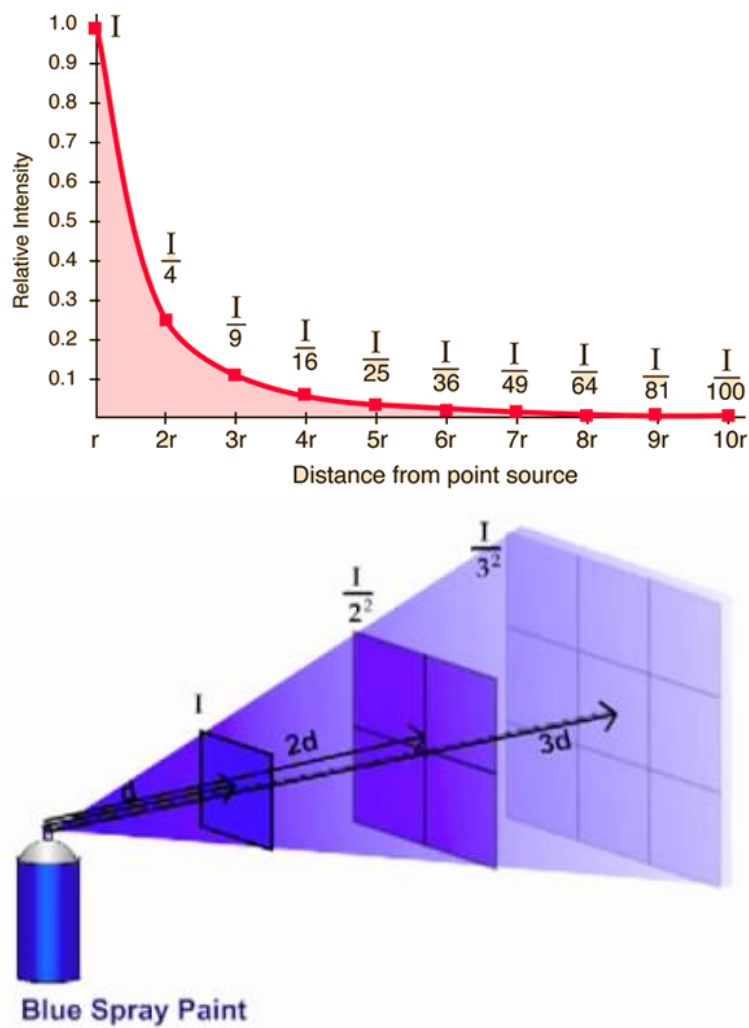


Figura 3.2: Decaimento da energia radiada com a distância, de acordo com a lei do inverso do quadrado.

3.2.2 Lâmpada Fluorescente

As lâmpadas fluorescentes são usadas em iluminação, cultivo, processos fotoquímicos, forenses, e polimerização (construção 3D, colas e restauros dentários). A figura 3.4 mostra uma lâmpada fluorescente de ultra-violetas de baixa potência. Tal como o fio eletroluminescente, esta lâmpada também necessita de alta tensão para funcionar, também conseguida com um pequeno circuito inversor. Por consequência o seu controlo é semelhante. A maior diferença está no consumo de corrente, e que é superior ao do fio eletroluminescente, o que seria de esperar pois a radiação emitida é também superior.

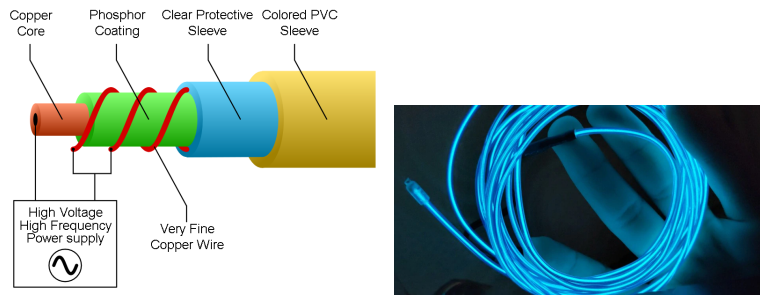


Figura 3.3: Fio eletroluminescente em funcionamento e respetivo modelo de construção.



Figura 3.4: Lâmpada fluorescente de ultra-violetas.

3.2.3 LEDs Monocromáticos

Os indicadores luminosos mais habituais em sistemas embebidos são os LEDs, uma vez que consomem muito menos corrente, funcionam com menos tensão, e têm maior durabilidade do que as tradicionais lâmpadas incandescentes. A figura 3.5 mostra o aspeto de LEDs com encapsulamentos translúcidos de diversos formatos e cores, o seu modelo de construção, e o seu símbolo elétrico.

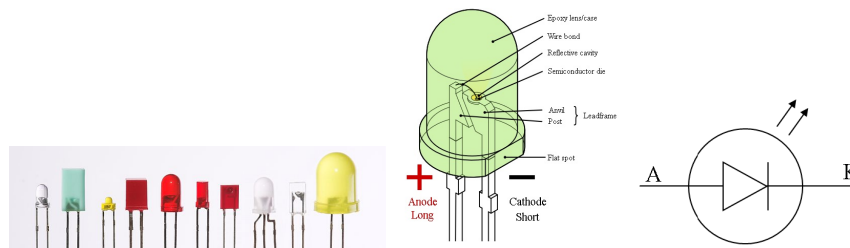


Figura 3.5: Modelo e símbolo elétrico de um LED.

Os LEDs são fabricados para emitir radiação visível numa cor correspondente a uma pequena faixa no espectro, por exemplo: vermelho, verde, azul, amarelo, branco, infravermelho, ultravioleta e laser. Os LEDs coloridos são usados para dar indicações de diferentes tipos, onde é habitual aplicar-se a convenção dos semáforos das estradas: verde - em funcionamento normal, amarelo - atenção, e vermelho - perigo ou alarme. Os LEDs infravermelhos são usados principalmente em comandos de eletrodomésticos tais como televisões e equipamento Hi-Fi. Os LEDs laser são usados para a projecção de pontos com precisão em distâncias de alguns metros.

Cor	Comp. de Onda (nm)	Tensões (V)
Infravermelho	850 a 940	1,4 a 1,7
Vermelho	620 a 660	1,7 a 1,9
Amarelo/Laranja	605 a 620	2,0 a 2,2
Verde	525 a 570	2,1 a 3,0
Azul	430 a 470	3,4 a 3,8
Ultra-violeta	410	3,5 a 4,0
Branco	*	3,3 a 4,0

Tabela 3.1: Comprimento de onda da radiação gerada por diferentes tipos de LEDs e a sua tensão de funcionamento.

Os LEDs possuem polaridade e os mais simples têm dois terminais: positivo (+) e negativo (-), sendo o terminal positivo mais comprido do que o negativo, e o encapsulamento do LED tem um pequeno corte na sua base a indicar o lado do terminal negativo. Para fazer um LED brilhar é necessário ligar o terminal positivo do LED ao da fonte, e o negativo a GND. Se o LED for ligado ao contrário, o LED não brilhará. A tabela 3.1 mostra os valores de comprimento de onda da radiação gerada e a gama de tensões de funcionamento de diferentes tipos de LEDs. As diferenças existem por causa dos diferentes materiais usados na sua construção para produzir as diferentes cores. Os LEDs brancos são criados com materiais que emitem radiação num largo espectro, e são caracterizados pela temperatura da cor, em vez de serem pelo comprimento de onda.

Dentro da gama de tensões de funcionamento de cada LED, um LED adquire mais ou menos brilho consoante o nível de tensão aplicada aos seus terminais. Por exemplo um LED vermelho a funcionar com 1,4 V tem menos brilho que com 1,5 V ou 1,7 V. Importa não exceder o limite máximo definido pelo fabricante sob pena de destruir o LED.

Como a tensão de funcionamento dos LEDs é inferior da tensão de funcionamento dos sistemas embebidos é necessário “retirar alguma tensão”, colocando uma resistência em série, para não os queimar.

Como existem muitos fabricantes de LEDs, e fornecedores de matéria prima para fabricar os LEDs, não há garantia que todos os LEDs sejam produzidos exatamente com as mesmas características, inclusive pela mesma fábrica. A consequência é à produção de componentes ligeiramente diferentes por variação do processo de fabrico. Se para as resistências e os condensadores as tolerâncias de fabrico podem ser compensadas através da associação de outros componentes, numa matriz de LEDs notam-se diferenças no brilho. A única forma de conseguir o mesmo brilho é ajustando a corrente que circula em cada LED individualmente com uma resistência variável ou circuito de controlo para cada um.

Para testar um LED vermelho com $V_d = 1,7\text{ V}$, a uma fonte de ali-

mentação de 5 V, é necessário fazer “cair” cerca de 3,3 V numa resistência intercalada (em série) com um dos terminais do LED e a fonte de alimentação. Segundo a lei de Ohm ($R=V/I$), e assumindo que o LED é percorrido por uma corrente de 10 mA, obtém-se o valor da resistência igual a $3,3/0,01 = 330\Omega$. Se não existir nenhuma resistência limitadora disponível pode-se ligar o ânodo (positivo do LED, terminal mais comprido) a 5 V, e o cátodo (negativo do LED, terminal mais curto) liga-se a 3,3 V, desta forma tem-se uma diferença de potencial de 1,7 V que é suficiente para fazer o LED brilhar.

A programação da atuação de um LED é feita através da escrita num pino de E/S digital de um valor lógico zero ou um, dependendo da montagem do LED. Se o LED tiver o seu ânodo ligado a VCC, significa que o pino de E/S do MCU vai fechar a ligação do cátodo do LED a GND. Desta forma o LED acende sempre que o pino estiver a 0 (GND), e fica apagado quando estiver a 1 (VCC). Com cátodo do LED ligado diretamente a GND, e o seu ânodo ao pino de E/S, ele vai acender quando o pino de E/S estiver a 1 (VCC) e apagado quando estiver a 0 (GND).

Muitas vezes os LEDs são usados não só para dar indicações sobre o estado de um componente ou sistema, mas também para criar animações. Uma animação mais simples e muito conhecida de um série de televisão dos anos 80, “O Justiceiro” (*Knight Rider*), onde o ator principal tinha um Pontiac Firebird Trans Am equipado com um conjunto de luzes na frente do carro que faziam um efeito “vai-e-vem”. Com um Arduino e um conjunto de 8 LEDs é possível recriar esse efeito. As listagens 3.2 e 3.3 mostram Sketches para fazer a programação desse efeito. O primeiro Sketch usa um iterador para ativar o LED dependendo do valor da variável de direção de acendimento. No segundo Sketch foi criada uma matriz que aceita qualquer valor para codificar o acendimento dos LEDs, e o corpo do programa repete essa sequência ciclicamente. Outra diferença é que no primeiro SKetch a espera é bloqueante, e no segundo Sketch a espera não é bloqueante, pelo que podem ser realizadas outras operações noutros estados. Embora a primeira forma seja mais simples, a segunda oferece a possibilidade de fazer animações mais complexas e facilidade em integrar noutras aplicações mais complexas.

Em ambos os casos existe uma máquina de estados definida com três estados: INIT para atribuição dos valores iniciais, MOVE para realizar o deslocamento dos LEDs, e WAIT para esperar pelo momento da nova transição.

Listagem 3.2: Kitt #1

```
// Knight Rider's KITT lights
// see https://www.arduino.cc/en/Tutorial/KnightRider

const int numLEDS = 8;
const int pinLEDS[] = { 0, 1, 2, 3, 4, 5, 6, 7 };
const int waitTime = 50;

enum states { INIT = 0, MOVE, WAIT } state;
char led = 0;
char dir = 1; // -1=<<, 1=>>
```

CAPÍTULO 3. ATUADORES

```
void setup() {
    byte i;
    for (i = 0; i < numLEDS; i++) {
        pinMode(pinLEDS[i], OUTPUT);
    }
}

void loop() {
    switch (state) {
        case INIT:
            led = 0; // first LED
            dir = 1;
            state = MOVE;
            break;
        case MOVE:
            digitalWrite(pinLEDS[led], LOW);
            if ((led > 0) && (led < numLEDS - 1)) {
                digitalWrite(pinLEDS[led - 1], HIGH);
                digitalWrite(pinLEDS[led + 1], HIGH);
            }
            led += dir;
            if (led > numLEDS - 1) {
                dir = -1;
                led = numLEDS - 1;
            }
            if (led < 0) {
                dir = 1;
                led = 0;
            }
            state = WAIT;
            break;
        case WAIT:
            state = MOVE;
            delay(waitTime);
            break;
        default: break;
    }
}
```

Listagem 3.3: Kitt #2

```
// Knight Rider's KITT lights

#define DELAY    50

const int numLEDS = 8;
const int pinLEDS[] = {0, 1, 2, 3, 4, 5, 6, 7 };
const int waitTime = 200;

enum states { INIT = 0, MOVE, WAIT } state;
byte pattern, seqNum;
// active low/negated sequence
byte ledSeq[] = { 0x7F, 0xBF, 0xDF, 0xEF, 0xF7, 0xFB, 0xFD, 0xFE, // vai
                  0xFE, 0xFD, 0xFB, 0xF7, 0xEF, 0xDF, 0xBF, 0x7F }; // vem

byte seqLen = 16;
unsigned long chTime;

void digitalWritePort()
{
    byte i;
    for (i = 0; i < numLEDS; i++) {
        digitalWrite(pinLEDS[i], pattern & (1 << i));
    }
}

void setup() {
    byte i;
    for (i = 0; i < numLEDS; i++) {
        pinMode(pinLEDS[i], OUTPUT);
    }
}
```

```

    }
}

void loop() {
  switch (state) {
    case INIT:
      pattern = ledSeq[0]; // single LED
      seqNum = 0;
      state = MOVE;
      break;
    case MOVE:
      pattern = ledSeq[seqNum];
      seqNum++; // next pattern
      seqNum %= seqLen; // mod 16: 0-15
      digitalWritePort();
      state = WAIT;
      chTime = millis();
      break;
    case WAIT:
      if (millis() - chTime > DELAY) {
        state = MOVE;
      }
      break;
    default: break;
  }
}

```

A figura 3.6 mostra o aspeto do circuito em funcionamento. As ligações para realizar este circuito consistem a ligação a 5 V de um conjunto de 8 resistências que ligam ao ânodo (positivo) de cada LED individualmente e os cátodos dos LEDs aos pinos de E/S.

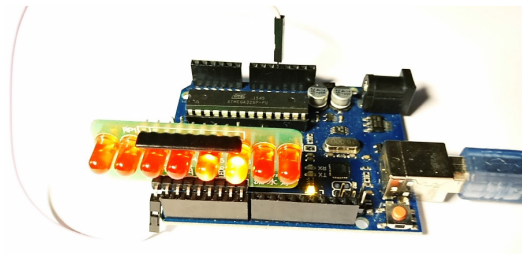


Figura 3.6: Demonstração da animação com LEDs.

Para controlar um LED individualmente, pode-se tirar partido da programação orientada a objetos para ocultar os detalhes da escrita nos pinos de E/S e focar o desenvolvimento no comportamento e nas relações dos objetos. A listagem 3.4 contém a listagem do Sketch de demonstração da utilização da classe `Led`. Esta class tem no construtor o pino de E/S e o nível lógico de ativação HIGH/LOW. como parâmetro de entrada. Esta classe oferece ainda os seguintes métodos públicos: `setON()` para ativar o LED, `setOFF()` para desativar o LED, `getState()` para obter o estado do LED, `setLevel()` para controlar o brilho do LED com um determinado nível (0-255), `setBlink()` para definir o ritmo de intermitência do LED, e `task()` para gerir autonomamente a intermitência do LED sendo chamado no ciclo principal da aplicação.

Listagem 3.4: Listagem do Sketch para demonstração da classe Led

```
class Led {
  private:
    uint32_t t0, t1, t2, now;
    byte pin;
    bool activate;
    bool state;
  public:
    Led(byte pin = 13, bool activate = HIGH) {
      this->pin = pin;
      pinMode (pin, OUTPUT);
    }
    void setON() {
      digitalWrite(pin, activate);
      state = activate;
    }
    void setOFF() {
      digitalWrite(pin, !activate);
      state = !activate;
    }
    bool getState() {
      return this->state;
    }
    void setLevel(byte level) {
      analogWrite(pin, level);
      if(level == 0) {
        state = LOW;
      }
      else {
        state = HIGH;
      }
    }
    void setBlink(int onTime, int totalTime) {
      // if(onTime > totalTime) abort();
      t0 = millis(); // cycle start
      t1 = onTime;
      t2 = totalTime;
      digitalWrite(pin, activate);
      state = activate;
    }
    void task() {
      if(t1 == 0 && t2 == 0) return;
      now = millis();
      if(now >= t0 + t1) {
        digitalWrite(pin, !activate);
        state = !activate;
      }
      if(now >= t0 + t2){
        t0 = now;
        digitalWrite(pin, activate);
        state = activate;
      }
    }
};

Led laser = Led(13);

void setup ()
{
  laser.setBlink(1, 5000);
}

void loop () {
  #if 0
  // old fashion
  laser.setON();
  delay (1000);
  laser.setOFF();
  delay (1000);
  #else
  // let the object deal with it
  laser.task();
  #endif
}
```

3.2. INDICADORES VISUAIS / LUMINOSOS

```
    delay(10);  
#endif  
}
```

Na demonstração da figura 3.7 foi usado um laser vermelho da Keys, o KY-008, que opera na banda dos 650nm. Este laser funciona com tensões entre 3,3 e 5V, e consome cerca de 30mA. O laser tem 3 pinos: VCC, GND e controlo ON/OFF. Neste caso apenas 2 pinos foram disponibilizados na placa de ensaio, estando sempre ativos quando recebem alimentação.

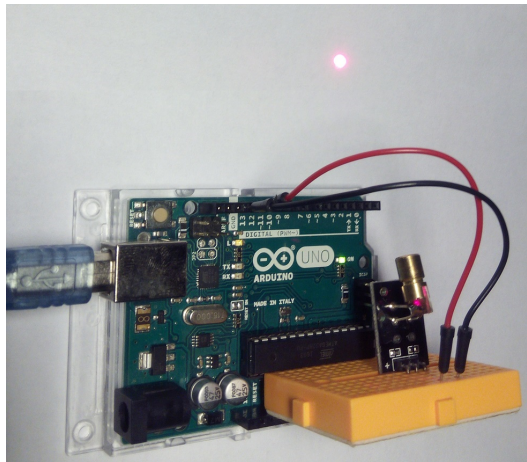


Figura 3.7: Demonstração da montagem do diodo laser.

Foi medida a corrente consumida pelo laser, e foi observado que varia em função da sua alimentação. A tabela 3.2 resume os valores de corrente e potência consumida pelo laser para cada pino. O motivo pelo qual o pino D12 fornece mais corrente ao LED do que o pino D13 deve-se ao facto de parte da corrente máxima fornecida pelo pino E/S estar a ser consumido pelo LED soldado na placa Arduino. Na realização de experiências com LEDs laser devem-se usar de óculos de proteção adequados.

Pino	Corrente [mA]	Potência [mW]
3,3V	11,6	34,8
5V	27	150
D12	22	110
D13	19	95

Tabela 3.2: Corrente e potência consumida por um LED laser quando ligado a diferentes pinos de um Arduino Uno.

Existem LEDs que conseguem produzir mais do que uma cor. Existem LEDs dicromáticos (duplos) e policromáticos (triplos ou RGB). Os LEDs dicromáticos possuem dentro de si dois núcleos LEDs de cores diferentes. A cor emitida por estes LEDs é a soma das cores ativas.

3.2.4 LEDs Dicromáticos

Os LEDs dicromáticos mais comuns são construídos com dois núcleos: um verde e outro vermelho, e possuem três terminais: cátodo (negativo) comum, ânodo do LED verde e ânodo do LED vermelho. Existem também modelos em que o terminal comum é o ânodo (positivo). A figura ?? mostra os detalhes do LED dicromático, das suas ligações internas e de uma PCB de demonstração de um LED dicromático. A ligação de cada LED é realizada como se de LEDs independentes se tratasse, o que para ligar a um sistema embebido requer dois pinos de E/S. Atuando os dois LEDs em simultâneo é possível fazer a mistura das duas cores e produzir uma nova cor, neste caso o amarelo. Variando a intensidade de brilho de cada LED, é possível obter as várias cores do espectro entre o verde e o vermelho.

Para ligar um LED dicromático a um Arduino Uno é necessário colocar resistências em série com a ligação de cada núcleo do LED. A figura 3.9 ilustra a montagem desse circuito em breadboard.

A listagem 3.5 mostra o sketch para programar o LED dicromático. A classe `Led2` define os atributos e comportamento do LED. O construtor da classe recebe como argumentos os dois pinos de E/S necessários para as componentes de cor vermelhas e verdes. A classe define os métodos `setR()`, `setY()`, `setG()`, para tratar das ativações necessárias para que o LED produza as cores vermelho, amarelo e verde, respetivamente. O método `setOFF()` desliga o LED.

Listagem 3.5: Listagem do Sketch `LedBiColor.ino`

```
class Led2 {
    typedef enum Colors { OFF, GREEN, YELLOW, RED } ;
private:
    uint32_t t0, t1, t2, now;
    byte redPin, greenPin;
    Colors state;
public:
    Led2(byte redPin = 11, byte greenPin = 10) {
        this->redPin = redPin;
        this->greenPin = greenPin;
        pinMode (redPin, OUTPUT);
        pinMode (greenPin, OUTPUT);
    }
    void setR() {
        digitalWrite(redPin, HIGH);
        digitalWrite(greenPin, LOW);
        state = RED;
    }
    void setY() {
        digitalWrite(redPin, HIGH);
        digitalWrite(greenPin, HIGH);
        state = YELLOW;
    }
    void setG() {
        digitalWrite(redPin, LOW);
        digitalWrite(greenPin, HIGH);
        state = GREEN;
    }
    void setOFF() {
        digitalWrite(redPin, LOW);
        digitalWrite(greenPin, LOW);
    }
}
```

3.2. INDICADORES VISUAIS / LUMINOSOS

R	G	B	Cor resultante
		x	Azul
	x		Verde
	x	x	Ciano
x			Vermelho
x		x	Roxo
x	x		Amarelo
x	x	x	Branco

Tabela 3.3: Cor resultante da combinação das várias componentes de cor.

```
    state = OFF;
}
bool getState() {
    return this->state;
}
void setLevel(byte rLevel, byte gLevel) {
    analogWrite(redPin, rLevel);
    analogWrite(greenPin, gLevel);
    if(rLevel == 0 && gLevel == 0) {
        state = LOW;
    }
    else {
        state = HIGH;
    }
}
};

Led2 led = Led2(13,12);

void setup () { }

void loop () {
    led.setG();
    delay (1000);
    led.setY();
    delay (1000);
    led.setR();
    delay (1000);
    led.setOFF();
    delay (500);
}
```

3.2.5 LEDs Policromáticos

Os LEDs RGB são semelhantes aos dicromáticos, mas são construídos com três núcleos de semicondutores independentes que emitem as cores: vermelho (R - *red*), verde (G - *green*) e azul (B - *blue*). As combinações das ativações de cada componente de cor, permitem produzir qualquer do espectro visível. As cores resultantes da combinações das três cores corresponde à soma das suas componentes e estão resumidas na tabela 3.3.

Tal como nos LEDs dicromáticos, para reduzir os pinos na construção do LED, em vez de se duplicarem todos os contactos necessários para ligar um LED, agrupam-se os sinais positivos (ânodo comum) ou os negativos (cátodo comum).

CAPÍTULO 3. ATUADORES

A ligações internas de um led RGB estão ilustradas na figura 3.10. A figura 3.11 ilustra as ligações a realizar em breadboard para ligar um LED RGB a um Arduino Uno. Para simplificar a montagem do circuito, todas as resistências são de 220 Ω .

Para controlar o LED RGB, foi criada a classe `LedRGB`. O construtor desta classe recebe como parâmetros os identificadores dos pinos de E/S do LED vermelho, verde e azul, que ficam registados nos atributos privados da classe, `rPin`, `gPin`, `bPin`, respetivamente. Esta classe oferece `setColor(..)` para fazer acender o LED com uma determinada cor. Existem duas implementações desta função com argumentos diferentes:

- `void setColor(byte r, byte g, byte b)` - acende o LED com a codificação de cada componente de cor;
- `void setColor(byte color)` - faz acender o LED com uma cor predefinida no enumerado `colors`: desligado (0), azul, verde, ciano, vermelho, roxo, amarelo, branco (7).

Existe ainda a função `setAnColor(byte r, byte g, byte b)` que permite especificar a intensidade de cada componente de cor individualmente com valores entre 0 e 255. A listagem 3.7 mostra o Sketch para demonstração da classe `LedRGB`.

Listagem 3.6: Listagem do Sketch `LedRGB.ino` para demonstrar o uso do LED RGB

```
enum colors { OFF=0, BLUE, GREEN, CYAN, RED, PURPLE, YELLOW, WHITE };

class LedRGB {
private:
    byte rPin, gPin, bPin;
public:
    LedRGB(byte rPin = 11, byte gPin = 10, byte bPin = 9) {
        this->rPin = rPin;
        this->gPin = gPin;
        this->bPin = bPin;
        pinMode (rPin, OUTPUT);
        pinMode (gPin, OUTPUT);
        pinMode (bPin, OUTPUT);
    }
    void setColor(byte r, byte g, byte b) {
        digitalWrite(rPin, r==0?LOW:HIGH);
        digitalWrite(gPin, g==0?LOW:HIGH);
        digitalWrite(bPin, b==0?LOW:HIGH);
    }
    void setColor(byte color) {
        byte r,g,b;
        switch(color) {
            case 0: r = 0; g = 0; b = 0; break; // off
            case 1: r = 0; g = 0; b = 1; break; // blue
            case 2: r = 0; g = 1; b = 0; break; // green
            case 3: r = 0; g = 1; b = 1; break; // cyan
            case 4: r = 1; g = 0; b = 0; break; // red
            case 5: r = 1; g = 0; b = 1; break; // purple
            case 6: r = 1; g = 1; b = 0; break; // yellow
            case 7: r = 1; g = 1; b = 1; break; // white
            default: r = 0; g = 0; b = 0;
        }
    }
}
```



```

        setColor(r, g, b);
    }
    void setAnColor(byte r, byte g, byte b) {
        analogWrite(rPin, r);
        analogWrite(gPin, g);
        analogWrite(bPin, b);
    }
};

LedRGB led = LedRGB(11, 10, 9);

void setup () {
    led.setColor(WHITE); // test all color components
    delay (1000);
}

void loop () {
    int i, j, k;
    for(i = 0; i < 8; i++) {
        led.setColor(i);
        delay (1000);
    }
    // k - color component combination
    for(k = 0; k < 8; k++) {
        for(i = 0; i < 256; i+=8) {
            led.setAnColor(k&4?i:0, k&2?i:0, k&1?i:0);
            delay (50);
        }
    }
}

```

3.2.6 LEDs com Controlador Digital

Para além de indicação de informação estática, por causa da sua durabilidade, baixo consumo e velocidade de resposta, os LEDs são também usados na construção de animações luminosas. Para facilitar a construção de circuitos para realizar animações luminosas, existem soluções de LEDs que integram controladores dentro do encapsulamento do LED, ou seja, a tensão colocada nos terminais do LED deixa de controlar a cor e a intensidade da luz emitida pelo LED. Neste caso o LED fica com autonomia para criar efeitos luminosos, como por exemplo o efeito pisca-pisca da árvore de natal.

Na figura 3.12 é apresentado o detalhe do interior do LED K-S5MUBS da Kedas. Este LED RGB de 5 mm tem um controlador que gera uma sequência de sete cores que vão alternando ao longo do tempo e com diferentes intensidade e intervalos de tempo. Existem também versões com LEDs de 3 mm e Surface Mount Device (SMD). Este LED é construído com base num LED RGB e num controlador que se encontra na base do LED, sendo necessários apenas os dois terminais da alimentação. Assim que é alimentado o LED começa a piscar com as várias cores. O sistema embebido que estiver ligado a este LED apenas tem capacidade de o ligar ou desligar, não tendo acesso a qualquer outra funcionalidade ou reprogramação da sequência de cores.

Existem ainda LEDs com controlador programável no seu interior, nomeadamente os WS2811/2812/2812B da Integrated Light Source, conhecidos como NeoPixel, apresentados na figura 3.13. Estes LEDs quando alimenta-

dos não acendem e necessitam de ser programados para acender. Os LEDs NeoPixel têm um sinal de entrada de controlo e um sinal de saída. O sinal de saída pode ser ligado à entrada de outro LED NeoPixel em série, ou cascata, o que significa que apenas um sinal de digital pode ser usado para controlar um mostrador com vários LEDs Neopixel, por exemplo a matriz de LEDs 8x8 ou 16x16. Em qualquer dos casos apenas são necessários 3 fios: VCC, GND e DATA.

A programação de um Neopixel é feita com base num protocolo série assíncrono, em que cada bit dura 1,25 μ s. À medida que cada bit é recebido, eles são “deslocados” num shift-register de 24 bits existente dentro de cada LED. O (primeiro) LED ligado ao Arduino após receber a primeira sequência de 24 bits, assim que receber novos bits na sua entrada, vai envia-los pela sua saída para o LED seguinte, e assim sucessivamente até que os primeiros 24 bits cheguem ao último LED.

Cada conjunto de 24 bits é dividido em 3 blocos de 8 bits, um bloco por cada componente verde, vermelho e azul. O valor de 8 bits representa a intensidade da componente de cor, sendo a cor resultante a mistura das três componentes.

Para programar um Arduino de forma a cumprir com o protocolo dos LEDs Neopixel, no menu Sketch-Include Library-Manage Libraries, tal como ilustrado na figura 3.14.

Em alternativa pode ser instalada manualmente a biblioteca disponibilizada pela Adafruit em: https://github.com/adafruit/Adafruit_NeoPixel

Esta biblioteca pode ser incluída num Sketch através de `#include <Adafruit_NeoPixel.h>`, e tem definida a classe `Adafruit_NeoPixel`. Os objetos instanciados desta classe têm como parâmetros o número de LED encadeados (em barra), o pino do Arduino onde estão ligados, e opções de configuração.

As opções de configuração do objeto `Adafruit_NeoPixel` pelo construtor são a velocidade de transmissão dos bits e a ordem das componentes de cor: `NEO_KHZ800` (WS2812), `NEO_KHZ800` (WS2811), `NEO_GRB` (maioria), `NEO_RGB` (NeoPixel v1) e `NEO_RGBW` (NeoPixel RGBW).

Como a programação de um LED requer a alteração dos restantes, a biblioteca guarda em memória a configuração de cada LED, e disponibiliza um método para copiar a programação de todos os LEDs. Isto permite poupar a programação de todos os LEDs por cada um que o Sketch altera.

Um resumo de alguns dos métodos oferecidos por esta classe estão resumidos na table 3.4.

A listagem 3.7 mostra um pequeno Sketch para demonstração da classe NeoPixel.

Listagem 3.7: Listagem do Sketch Neopixel.ino para demonstrar o uso do LED NeoPixel

```
#include <Adafruit_NeoPixel.h>
```

3.2. INDICADORES VISUAIS / LUMINOSOS

Método	Descrição
begin(void)	Inicia o componente
show(void)	Copia a informação em memória para os LEDs
setPixelColor(uint16_t n, uint8_t r, uint8_t g, uint8_t b)	Define a cor do LED n usando as componentes RGB
setPixelColor(uint16_t n, uint32_t c)	Define a cor do LED n com a cor codificada num valor a 32 bits (W-R-G-B)
fill(uint32_t c=0, uint16_t first=0, uint16_t count=0)	Afeta um conjunto de LEDs com a mesma cor (32 bits)
setBrightness(uint8_t)	Define o brilho dos LEDs (0-255)
clear(void)	Apaga todos os LEDs

Tabela 3.4: Alguns dos métodos disponibilizados pela classe `Adafruit_NeoPixel`.

```
#define NEO_PIN 6
#define NUMPIXELS 1

Adafruit_NeoPixel pixels(NUMPIXELS, NEO_PIN, NEO_GRB + NEO_KHZ800);

void setup() {
  pixels.begin(); // initialize NeoPixel object
  pixels.clear(); // all pixel colors to 'off'

  // pixels.Color(R, G, B) from 0,0,0 up to 255,255,255
  pixels.setPixelColor(0, pixels.Color(0, 0, 255));
  pixels.show(); // Send the updated pixel colors to the hardware.
}

void loop() {
}
```

Para além do encadeamento de LEDs NeoPixel em barras, ou anéis, também é possível criar matrizes de LEDs para afixação de texto ou gráficos. Para usar uma matriz de LEDs NeoPixel é necessário instalar as bibliotecas `Adafruit_NeoMatrix` e `Adafruit-GFX-Library` da Sparkfun. Estas bibliotecas consideram o mostrador como uma mapa 2D em que cada posição (x,y) contém a codificação da cor do pixel. A figura 3.15 mostra a demonstração da utilização de um módulo com uma matriz 8x8 de LEDs NeoPixel. É possível agregar vários módulos para criar mostradores maiores.

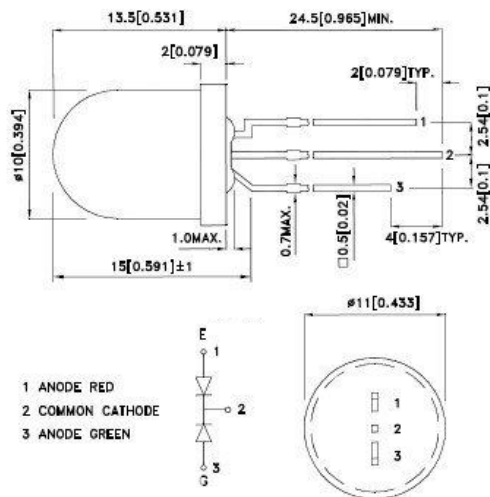


Figura 3.8: Detalhes da construção de um LED dicromático (esquerda) e PCB de demonstração (direita).

3.2. INDICADORES VISUAIS / LUMINOSOS

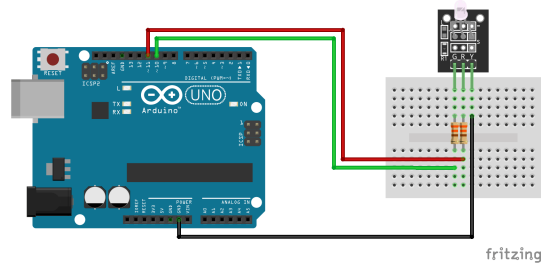


Figura 3.9: Ligação da placa do LED bicolorido a um Arduino Uno.

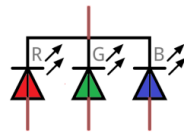


Figura 3.10: Esquema interno de um LED RGB.

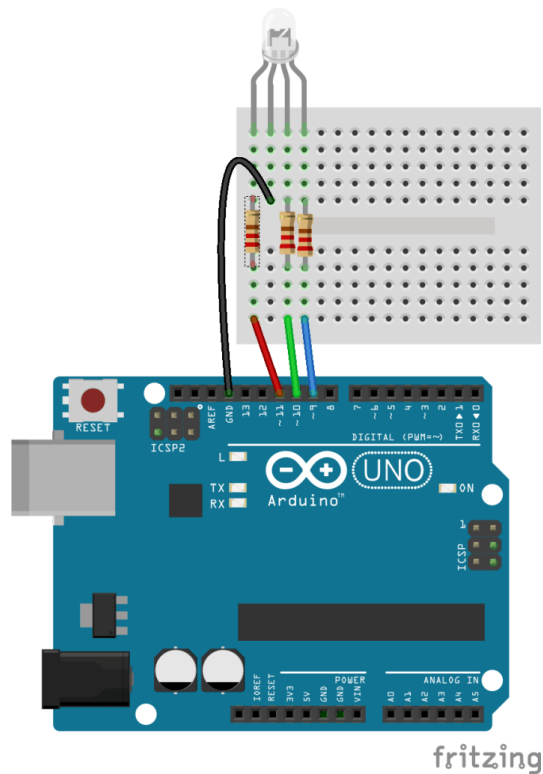


Figura 3.11: Montagem para demonstração do LED RGB.



Figura 3.12: Detalhe do LED pisca-pisca de 7 cores.

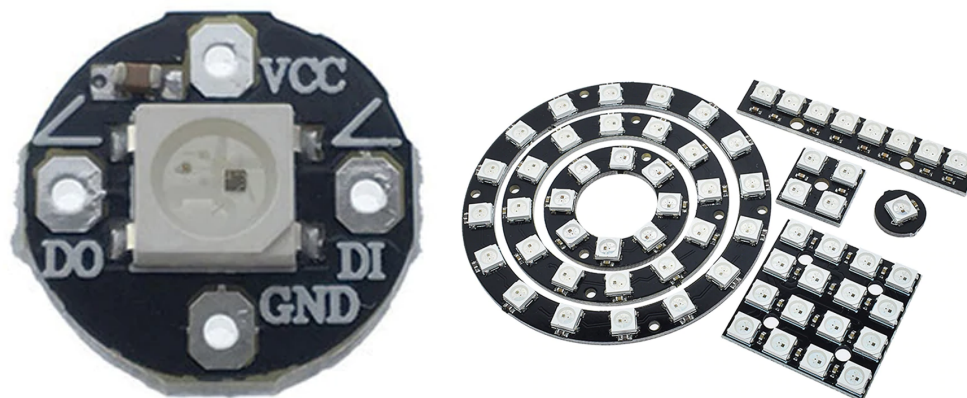


Figura 3.13: Detalhe de um Neopixel à esquerda e diferentes arranjos de circuitos com LEDs Neopixel.

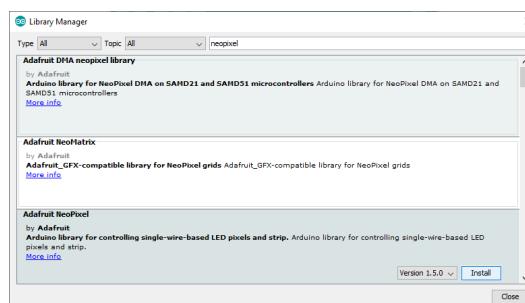


Figura 3.14: Detalhe de um Neopixel à esquerda e diferentes arranjos de circuitos com LEDs Neopixel.

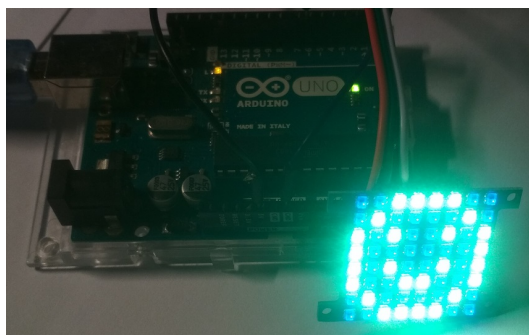


Figura 3.15: Matriz 8x8 de LEDs Neopixel.

3.3 Mostradores

Os mostradores são uma composição de vários indicadores organizados de forma de matriz para mostrar informação mais completa do que um simples indicador colorido.

3.3.1 7-21 Segmentos

A miniaturização da tecnologia LED permitiu criar encapsulamentos moldados de forma a criar conjuntos de 7 ou 21 segmentos, onde cada segmento é iluminado por um LED. Os mostradores de 7 segmentos são habituais na indicação de valores numéricos, por exemplo para controlar a chamada da vez para atendimento (supermercado e serviços), e os de 15 a 21 segmentos onde se pretende dar alguma informação alfanumérica (máquinas de venda automática).

Num mostrador de segmentos, apenas um dos terminais dos LEDs se encontra disponível para ser ligado, ao passo que os outros encontram-se ligados num único ponto. Esta técnica permite reduzir o número de ligações necessárias. Existem versões em que o terminal positivos dos LEDs (ânodo) é comum, ou o terminal negativo dos LEDs é comum (cátodo). Desativando o potencial no terminal comum, todos os segmentos deixam de se iluminar.

Existem versões de mostradores com mais do que um carater no mesmo componente. Neste caso os terminais dos segmentos são partilhados, havendo um pino comum para cada carater, e assim poupar no número de ligações a realizar.

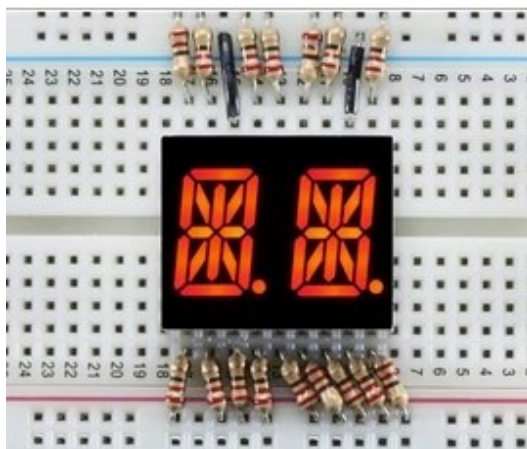


Figura 3.16: Mostrador alfanumérico de 15 segmentos.

3.3.2 Matriz de Pontos

Em algumas aplicações, importa ter mais do que um conjunto de LEDs organizados de forma a compor um mostrador. Os LEDs têm a vantagem de serem baratos, duráveis, simples e têm baixo consumo. Por estes motivos é interessante conseguir compor, e estruturar, a informação a mostrar para facilitar a interpretação pelo utilizador. Um mostrador tem a flexibilidade de conseguir compor mais do que um tipo de mensagens, e até criar animações como texto em movimento.

Um mostrador de matrix de pontos é uma estrutura que contém vários LEDs idênticos organizados em matriz incorporados num único componente. As matrizes de LEDs têm tamanhos de 5x8 ou 8x8 pontos, que é suficiente para criar um carácter de texto ou símbolo. Os mostradores podem ser agrupados para criar mostradores de maiores dimensões. Os LEDs são ligados de forma a que os ânodos são comuns a todas as colunas e os cátodos comuns a todas as linhas.

Por exemplo, um carácter alfanumérico pode ser representado com uma matriz com 5x8 pontos. Esta matriz com 40 LEDs necessitaria de 40 sinais de E/S individuais para controlar cada LED. A través da adopção da multiplexagem em 5 colunas e 8 linhas, é possível reduzir as ligações a 13 sinais de E/S. As linhas e as colunas são partilhadas.

A mutliplexagem, tal como explicada no capítulo dois deste livro, mostra como ativando uma coluna de cada vez é possível dar indicação de quais as linhas que vão ter os LEDs acesos. Através da persistência retiniana, ativando as várias colunas ciclicamente várias vezes por segundo é possível a ilusão de todas as colunas estarem ativas ao mesmo tempo.

3.3.3 Mostrador Alfanumérico (LCD/VFD)

A tecnologia LCD oferece matrizes de pontos de maior densidade, em comparação com a tecnologia LED, logo consegue produzir caracteres com maior detalhe que os LEDs.

Os mostradores de cristais líquidos (LCD) são os mais usados para apresentação de informação, pois permitem escrever texto e símbolos para criar palavras e frases fáceis de interpretar pelos utilizadores. Isto é conseguido graças ao uso do controlador LCD HD44780 da Hitachi, lançado em 1987, foi um dos primeiros Application-Specific Integrated Circuits (ASICs) de baixo custo dedicado à gestão de todos os pontos do mostrador para compor as mensagens afixadas, e que estabelece um interface com o sistema embebido. Uma das grandes vantagens deste mostrador tem uma *Read-Only Memory* (ROM) com caracteres ASCII e japoneses, uma memória RAM para definição de caracteres pelo utilizador. Existem diversos mostradores LCD baseados neste controlador, ou equivalente, que permitem escrever entre 1 a 8 linhas e de 8 a 80 caracteres por linha, variando apenas alguns parâmetros

na sua configuração. A figura 3.17 mostra um mostrador LCD com 2 linhas de 16 caracteres. Cada caráter é composto por uma matriz de pontos com 8 ou 10 linhas e 5 colunas. Os detalhes do controlador podem ser encontrados no *datasheet* do fabricante, no ficheiro HD44780.pdf.

A interface pode funcionar em transferências de 4 ou 8 bits em paralelo, e permite a seguinte funcionalidade:

- Escrever comandos de configuração e controlo, por exemplo presença, posição e forma do cursor;
- Definir caracteres personalizados (por exemplo símbolos e gráficos de baixa resolução);
- Escrever caracteres;
- Obter o estado de atividade (*busy*)

Para suportar estas funcionalidades a interface paralela de 14 ou 16 pinos tem os :

1. VSS/GND - negativo da alimentação;
2. VCC - Positivo da alimentação (5V);
3. Vo - Sinal analógico Vo para controlar o contraste dos pixels;
4. Register Select (RS) - 1: escreve ou lê dados, 0: escreve comandos ou lê o estado (*busy*);
5. Read/Write (R/W) - alternar entre escrita de dados e comandos, e leitura de estado (*busy*);
6. Enable (E) - Na transição 1-0 regista os dados na entrada do barramento;
7. D7-D0 - usado no modo 8 bits. Permite usar os 8 bits para transferências de dados e comandos com um acesso apenas;
8. D7-D4 - usado no modo 4 bits. Cada valor de 8 bits é dividido em duas partes de 4 bits e são enviadas para o LCD em dois acessos, sendo o primeiro para os 4 bit mais significativos e o segundo para os 4 bits menos significativos. Os bits D3-D0 não são usados neste modo.
9. BKL+ e BKL- são o Ânodo e Cátodo, respectivamente, dos LEDs de retro-iluminação do mostrador.

Como vários comandos demoram tempos diferentes a serem processados, o controlador disponibiliza acesso para leitura do bit 7 para aferição do seu estado (*busy?*). Isto permite que o sistema embebido saiba quando o Liquid Crystal Display (LCD) ficou pronto a aceitar o novo comando. No entanto, como os tempos de espera dos comandos do LCD estão entre mili-segundos e micro-segundos, o impacto no funcionamento do LCD é praticamente imperceptível para o utilizador. Desta forma, na maioria das vezes o LCD é tratado como uma máquina de escrever, onde se espera um período de tempo superior ao tempo máximo necessário pelo controlador para interpretação de dados e comandos. O mesmo critério pode ser aplicado à escolha da ligação a 4 ou 8 bits.

Para reduzir o número de ligações e tornar a construção do circuito menos complexa e mais barata, é possível usar a interface a 4 bits. Sabendo que o comando mais lento não necessita de 2ms para ser executado, a escrita de todos os caracteres usando duas transferências de 4 bits, num mostrador 20x4 demoraria $2 \times 2 \times 20 \times 4\text{ms} = 320\text{ms}$. Este tempo é mais que suficiente para operar com o mostrador. A única desvantagem é a ocupação do MCU do sistema embebido a realizar duas transações com o mostrador.



Figura 3.17: Mostrador alfanumérico LCD 16x2.

Para reduzir o número de ligações, uma alternativa a ligar o barramento do controlador LCD ao Arduíno, pode-se fazer uso de um registo I2C. Interface I2C permite ligar o LCD apenas com 4 fios: VCC, GND, SDA e SCL.

O registo I2C é um componente que de um lado tem uma interface escravo I2C e do outro uma saída com 8 bits em paralelo. Isto significa que permite que o sistema embebido escreva um valor com 8 bits num barramento I2C e esse valor é colocada nas entradas do LCD. Os 8 bits são suficientes para operar o LCD com transferências de dados a 4 bits e ativar os restantes sinais de controlo.

CAPÍTULO 3. ATUADORES

Atualmente existem mostradores LCD que oferecem a interface I2C já integrada no controlador do mostrador, mantendo o seu funcionamento compatível para a versão .

Para conseguir afixar mensagens no LCD é necessário seguir os passos definidos no manual do controlador (HD44780.pdf), e são diferentes para a operação a 4 ou 8 bits. No manual encontram-se descritos os passos para todas as configurações. A título de exemplo, segue a sequência de comandos para configuração de um mostrador a 4 bits encontra-se na tabela 3.5.

Ligar a alimentação e esperar 15ms

Enviar para o LCD: RS=0, RW=0, D7-D4=0011, e aguardar 4.1ms
Enviar para o LCD: RS=0, RW=0, D7-D4=0011, e aguardar 100us

Descrição	RS	RW	D7-D4	Observações
Reset	0	0	0 0 1 1	aguardar 4,1ms
-	0	0	0 0 1 1	aguardar 0.1ms
-	0	0	0 0 1 1	aguardar 0.1ms
Interface 4 bits	0	0	0010	testar flag busy
Linhas e fonte	0	0	0 0 1 0	testar flag busy
	0	0	N F x x	
Desligar ecrã	0	0	0 0 0 0	testar flag busy
	0	0	1 0 0 0	
Limpar ecrã	0	0	0 0 0 0	testar flag busy
	0	0	0 0 0 1	
Modo de escrita	0	0	0 0 0 0	testar flag busy
	0	0	0 1 P S	

Tabela 3.5: Comandos de configuração do LCD no modo 4 bits

A cada envio para o LCD é necessário colocar $E = 1$, esperar 230us, e voltar a colocar $E = 0$.

$N = 0$: 1 linha, 1 : 2 linhas (ou 4) $F = 0$: Fonte 5x8, 1 : Fonte 5x10

$P = 0$: decrementa a posição do cursor (move para a esquerda), 1 : incrementa (move para a direita) $S = 0$: deslocar o texto/ecrã para a direita, 1 : move para a esquerda

Se após correr o Sketch de demonstração e não aparecer nada no LCD, pode dar-se o caso do mostrador ter o nível de contraste no mínimo. Para modificar o contraste, deve-se usar uma chave de fendas fina, ou de precisão) e rodar o potenciômetro multivolta que se encontra perto do LCD.

A listagem 3.8 mostra como configurar e controlar o mostradores LCD baseados em controladores compatíveis com o HD44780.

Listagem 3.8: Listagem do ficheiro com a definição da classe Lcd44780

```
// alphanumeric LCD w/ support for I2C and parallel 4-bit interfaces
#include LCD_HD44780.h
```

```

#define LCD_HD44780.h

// I2C support
#include "Wire.h"
#define I2C_ADDR 0x27
// I2C writes: (bit7->bit0): |D3|D2|D1|D0|BL|EN|RW|RS|
#define RS_PIN 0x01
#define RW_PIN 0x02
#define EN_PIN 0x04
#define BL_PIN 0x08

// 4-bit support
// pin 8=RS, 9=EN, 4=D4,..., 7=D7, 10=BL
// no support for RD from LCD (no busy check)
// * compatible w/ LiquidCrystal library, but no BL control
// * requires #include <LiquidCrystal.h>
// * LiquidCrystal lcd(8, 9, 4, 5, 6, 7); */

class Lcd44780
{
private:
    const int T1 = 5, T2 = 5, T3 = 1, T4 = 1;
    int row, col;
    void lcdWr(int data, bool dataMode = false);
    void lcdWr4bit(int data, bool dataMode = false);
    void lcdWrI2C(int data, bool dataMode = false);
    void lcdData(int data);
    void lcdCmd(int data);
    void printChar(char c);
    int i2cAddr;
    int *pins;
    boolean bl;
public:
    Lcd44780(int address) {
        i2cAddr = address;
        bl = true;
    }
    Lcd44780(int pins[7]) {
        int i;
        i2cAddr = 0;
        bl = true;
        this->pins = pins;
        for(i = 0; i < 7; i++) {
            pinMode(pins[i], OUTPUT);
        }
    }
    void init(int row = 2, int col = 16);
    void clear();
    void setCursor(int row, int col);
    void setCursorBlink(bool cursor, bool blink);
    void resetCursor();
    void print(char s[]);
    void backlight(boolean val);
};

// write 4 bits at a time to the LCD
void Lcd44780::lcdWr(int data, bool dataMode = false){
    if(i2cAddr == 0) {
        lcdWr4bit(data, dataMode);
    }
    else {
        lcdWrI2C(data, dataMode);
    }
}

void Lcd44780::lcdWrI2C(int data, bool dataMode = false){
    Wire.beginTransmission(i2cAddr);
    Wire.write((data << 4) | (bl?BL_PIN:0) | (dataMode?RS_PIN:0));
    Wire.endTransmission();
    Wire.beginTransmission(i2cAddr);
    Wire.write((data << 4) | (bl?BL_PIN:0) | (dataMode?RS_PIN:0) | EN_PIN);
    Wire.endTransmission();
    delay(T4); // EN high

```

CAPÍTULO 3. ATUADORES

```
Wire.beginTransmission(i2cAddr);
Wire.write((data << 4) | (bl?BL_PIN:0) | RS_PIN);
Wire.endTransmission();
delay(T4);
}

void Lcd44780::lcdWr4bit(int data, bool dataMode = false){
// PORTD = PORTD & 0xF0; PORTD |= data;
digitalWrite(pins[0], dataMode); // RegisterSelect
digitalWrite(pins[1], 0); // Enable
digitalWrite(pins[2], data&1);
digitalWrite(pins[3], data&2);
digitalWrite(pins[4], data&4);
digitalWrite(pins[5], data&8);
digitalWrite(pins[6], bl); // Backlight
delay(T4); // EN high
digitalWrite(pins[1], 1);
delay(T4);
digitalWrite(pins[1], 0);
delay(T4);
}

void Lcd44780::lcdData(int data){
lcdWr(data >> 4, true);
lcdWr(data, true);
}

void Lcd44780::lcdCmd(int data){
lcdWr(data >> 4, false);
lcdWr(data, false);
}

void Lcd44780::init(int row = 2, int col = 16){
this->row = row;
this->col = col;
Wire.begin();
delay(T3);
lcdWr(0x03);
delay(T4);
lcdWr(0x03);
delay(T3);
lcdCmd(0x32);
delay(T3);
lcdCmd(0x28);
delay(T3);
lcdCmd(0x0F); // display ON, cursor ON e blink ON
delay(T3);
lcdCmd(0x80); // Set Cursor Address
delay(T3);
}

void Lcd44780::clear(){
lcdCmd(0x01); // Clear Display
}

void Lcd44780::setCursor(int row, int col){
int cfg;
// check LCD size (see datasheet for other sizes)
if(this->row == 4 && this->col == 20) {
cfg = 0x80 | ((row & 1) << 6 | (row & 2) << 3 | (row & 2) << 1) + col;
}
else {
cfg = (0x80 | row << 6) | col;
}
lcdCmd(cfg);
}

void Lcd44780::setCursorBlink(bool cursor, bool blink){
lcdCmd(0x0C | (cursor?2:0) | (blink?1:0));
}

void Lcd44780::resetCursor(){
lcdCmd(0x02); // set cursor @ (0, 0)
```

```

}

void Lcd44780::printChar(char c){
  lcdData(c); // Write cursor position
}

void Lcd44780::print(char s[]){
  for (int i = 0; s[i] != 0; ++i)
    lcdData(s[i]);
}

void Lcd44780::backlight(boolean bl){
  this->bl = bl;
  lcdCmd(0); // because of I2C it needs to write something
}
#endif

```

O Sketch de demonstração, na listagem 3.9, inicia o LCD e escreve uma palavra em cada linha do mostrador LCD 20x4.

Listagem 3.9: Listagem do Sketch de demonstração do uso da classe Lcd44780

```

#include "LCD_I2C.h"

#define LCD16x2_4bit
// #define LCD20x4_i2c

#ifdef LCD20x4_i2c
  Lcd44780 lcd(I2C_ADDR);
#endif

#ifdef LCD16x2_4bit
  // https://wiki.dfrobot.com/Arduino_LCD_KeyPad_Shield_SKU_DFR0009_
  int lcd_pins[7] = { 8, 9, 4, 5, 6, 7, 10 };
  Lcd44780 lcd(lcd_pins);
#endif

void setup()
{
  #ifdef LCD20x4_i2c
    lcd.init(4, 20);
  #endif
  #ifdef LCD16x2_4bit
    lcd.init(2, 16);
  #endif
  lcd.clear();
  lcd.setCursor(0, 4);
  lcd.print("Hello");
  lcd.setCursor(1, 5);
  lcd.print("World");
  #ifdef LCD20x4_i2c
    lcd.setCursor(2, 6);
    lcd.print("0f");
    lcd.setCursor(3, 7);
    lcd.print("Arduino!");
  #endif
  lcd.setCursorBlink(true, false);
  lcd.backlight(true);
}

void loop()
{
  // lcd.print("i2c");
}

```

Não contabilizando o espaço em memória necessário pela demonstração, o programa essencial para suportar o LCD consome cerca de 3kB de memória de programa e 240 bytes de memória dinâmica.

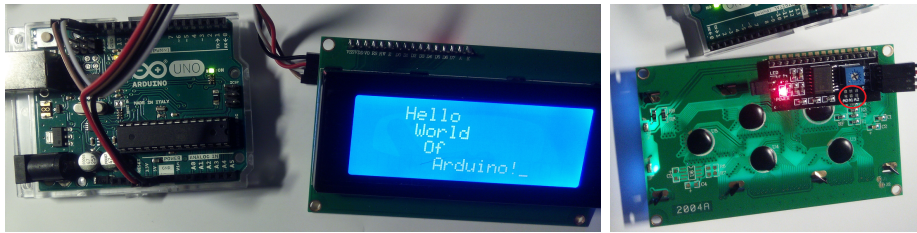


Figura 3.18: Mostrador alfanumérico LCD 20x4.

Embora o mostrador possa parecer limitado por apenas estar preparado para texto, a definição de caracteres personalizados aumenta o potencial. Em <https://www.tinkercad.com/things/6nSdspNbZXI-arduino-lcd-game#/> é apresentado um jogo com um mostrador LCD 16x2 e um botão de pressão ligado a um Arduino Uno. Neste jogo o utilizador controla uma figura que tem de se desviar de obstáculos. A pontuação é dada pelo número de objetos que consegue evitar. E em <https://github.com/davetcc/LcdGameBuilder/blob/master/examples/spaceGame/> é apresentado um jogos para LCD 20x4.

3.3.4 Mostrador Gráfico (LCD/OLED)

Um mostrador gráfico é constituído por centenas a milhões de pontos (pixels) organizados em matriz, e que podem ser coloridos ou monocromáticos. A vantagem do mostrador gráfico em relação ao alfanumérico, apresentado anteriormente, é a possibilidade de mostrar gráficos ou imagens enriquecendo a apresentação e interação com o sistema.

A figura 3.19 mostra um mostrador gráfico monocromático de 48x84 pontos, semelhante ao que equipava os telemóveis 5110 da Nokia. Os pontos deste mostrador possuem apenas dois estados: ativo ou inativo. Este mostrador, tal como o alfanumérico, não expõe as ligações dos seus 4032 pontos, estando ligado diretamente a um controlador, neste caso o PCD8544, com as linhas e colunas multiplexadas. Este controlador funciona como um controlador gráfico muito básico, em que para além de fazer a atuação dos pontos no mostrador, tem internamente uma memória que armazena a imagem apresentada no mostrador, dividida em 6 bancos de 86 bytes ($6 \times 84 \times 8$), endereçada como uma matriz com valores X entre 0 e 83 e Y entre 0 e 5. Enquanto que as colunas são mapeadas diretamente, as linhas estão organizadas em conjuntos de 8 (1 bit do byte guardado por linha). Os seis bytes disponíveis em cada coluna permitem codificar o valor do pixel de cada linha ($6\text{linhas} \times 8\text{bits} = 48\text{linhas}$). Em termos de ligação ao sistema embebida, o controlador PCD8544 é ligado através de uma interface série síncrona.

Os mostradores coloridos possuem em cada ponto dividido em três partes para gerar cada uma das componentes de cor: vermelho, verde e azul. Para além da indicação de componente de cor, é necessário também informação

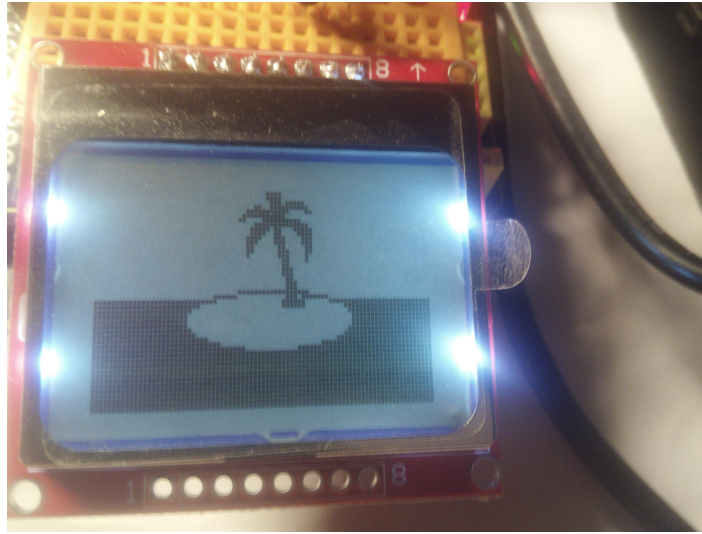


Figura 3.19: nokia5110 demo.

acerca da intensidade de cada uma. As cores que um mostrador colorido consegue gerar depende da combinação do número de níveis de intensidade de cada componente de cor. Um mostrador que tenha 8 bits para cada componente de cor, consegue gerar (2^8) 256 níveis de intensidade diferentes, produzindo $256 \times 3 = 768$ cores. Um mostrador com 16 bits produz $2^{16} \times 3 = 196,6$ milhares de cores.

TFT RGB

A figura

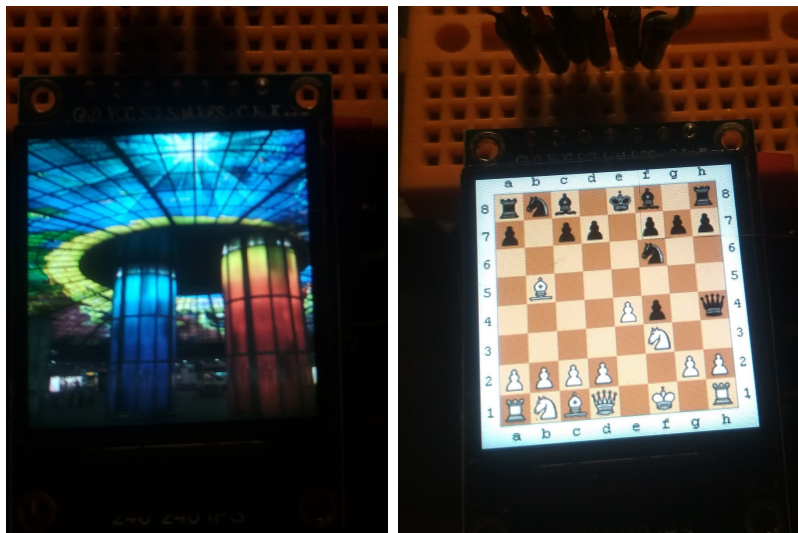


Figura 3.20: Ilustração de usos do TFT 240x240.

A figura 3.21 mostra um conjunto de *sprites* para realização de um jogo

CAPÍTULO 3. ATUADORES

de xadrez.



Figura 3.21: Sprites para jogo de xadrez [Wikipedia].

Listagem 3.10: Passive_Buzzer.ino

```
/*
*****
ST7789 SPI display Demo

Lib by Ananov Ilya.
*****/

#include <Adafruit_GFX.h> // Core graphics library by Adafruit
#include "st7789.h" // Hardware-specific library for ST7789 (with or without CS pin)
#include <SPI.h>

#define TFT_DC 8
#define TFT_RST 9
// #define TFT_CS 10 // only for displays with CS pin
#define TFT_MOSI 11 // for hardware SPI data pin (all of available pins)
#define TFT_SCLK 13 // for hardware SPI sclk pin (all of available pins)

Arduino_ST7789 tft = Arduino_ST7789(TFT_DC, TFT_RST, TFT_MOSI, TFT_SCLK); //for display without CS pin

void setup(void) {
  Serial.begin(115200);
  while(!Serial);
  Serial.println("Hello! ST7789 TFT Test");
  tft.init(240, 240); // initialize a ST7789 chip, 240x240 pixels
  Serial.println("Initialized");
  uint16_t time = millis();
  tft.fillScreen(BLACK);
  time = millis() - time;
}

/* color:
 * int (16 bits) = | red 15-11 | green 10-5 | blue 4-0 |
#define BLUE 0x001F (5 bits - 0000 0000 0001 1111)
#define GREEN 0x07E0 (6 bits - 0000 0111 1110 0000)
#define RED 0xF800 (5 bits - 1111 1000 0000 0000)

https://en.wikipedia.org/wiki/Netpbm_format
Remover as primeiras 3 linhas de valores do ficheiro .ppm:
P3
# Created by GIMP version 2.10.10 PNM plug-in
240 240
255

download apr 45 seg @115200 baud
*/

int readDec() {
  int decVal = 0;
  char c = 0;
  while(c != '\n') {
    while(Serial.available() == 0);
    c = Serial.read();
    if((c >= '0') && (c <= '9')) {
      decVal = (c - '0') + decVal * 10;
    }
  }
  //Serial.println(decVal, DEC);
  return decVal;
}
```

```
void loop() {  
  int x, y, pixel;  
  Serial.println("Waiting for a bitmap (240x240 RGB)");  
  for(y = 0; y < 240; y++) {  
    for(x = 0; x < 240; x++) {  
      pixel = (readDec() & 0xF8) << 8; // R  
      pixel = pixel | ((readDec() & 0xFC) << 3); // G  
      pixel = pixel | ((readDec() ) >> 3); // B  
      tft.drawPixel(x, y, pixel);  
    }  
  }  
}
```

... ..

3.3.5 Válvula de Cátodo Frio (Nixie/Numitron)

Intro: Foto compoennte: Princípio de funcionamento: Datasheet: PS/2

Ligações: Foto montagem e ensaio:

... ..

3.3.6 Sinal Vídeo Analógico

Os sinais de vídeo analógicos foram pensados para funcionar com os primeiros televisores a preto-e-branco, baseadas em tecnologia de tubo de raios catódicos, que era capaz de gerar um fluxo de eletrões a extremidade interior do cinescópio e projectar um ponto na tela (ânodo). A posição do ponto no ecrã depende do campo magnético gerado pelas bobinas de deflexão coloca-

das no cinescópio, no interior do televisor. O movimento rápido do ponto com diferentes níveis de intensidade, de um canto do ecrã até ao outro oposto, dá a ilusão de que uma imagem completa está a ser apresentada no ecrã.

Como os televisores eram baseados em electrónica analógica (não existiam controladores para contar pixels no ecrã), o controlo da posição do ponto era conseguido através do próprio sinal de vídeo. O sinal de vídeo começa um nível de tensão baixo, para sincronizar com o início da imagem, e de seguida a tensão passa a corresponder à intensidade do ponto no ecrã ao longo de uma linha. Quanto maior a tensão mais brilhante é o ponto, e quanto menor a tensão menos brilhante será. No final de cada linha, existe um novo pulso de sincronismo que indica que uma nova linha vai ter início. Isto permite que qualquer cinescópio consiga projectar a imagem, independentemente do seu tamanho.

Mais tarde foram adicionadas as componentes de áudio ao sinal de vídeo, através de uma portadora, e a informação de cor. O “truque” para se poder transmitir a cor e manter a compatibilidade com os equipamentos a preto-e-branco foi criado por XXXX*** em **** quando trabalhava nos laboratórios da RCA (Esta informação foi divulgada recentemente. Como nunca lhe foi dado crédito pela sua invenção, abandonou a engenharia e tornou-se agricultor em ****).

Por motivos de compatibilidade com os sistemas existentes, os ecrãs recentes como os de LCD, são capazes de descodificar os sinais analógicos e apresentar a imagem.

3.3.7 Vídeo Composto (PAL)

A norma PAL, derivado do nome *Phase Alternating Line*, é usada para difusão de sinal de televisão analógico, usado na Europa, Ásia e alguns países de África e América do Sul, alternativa ao NTSC e ao SECAM.

A norma PAL define a imagem com 625 linhas, apresentando em dois quadros de 312,5 linhas alternadamente, ao ritmo de 50 quadros por segundo. Um sinal PAL contém a sinalização em banda, ou seja, os dados da imagem, som e o sincronismo da imagem são enviados no mesmo sinal. Os sinais de sincronismo

A ligação do sinal PAL

Intro: Foto compoennnte: Principio de funcionamento: Datasheet: Ligações: Foto montagem e ensaio:

Os televisores e monitores *Cathode Ray Tube* (CRT) dominaram durante muitas décadas a difusão

A figura 3.24 mostra o esquema de um monitor CRT

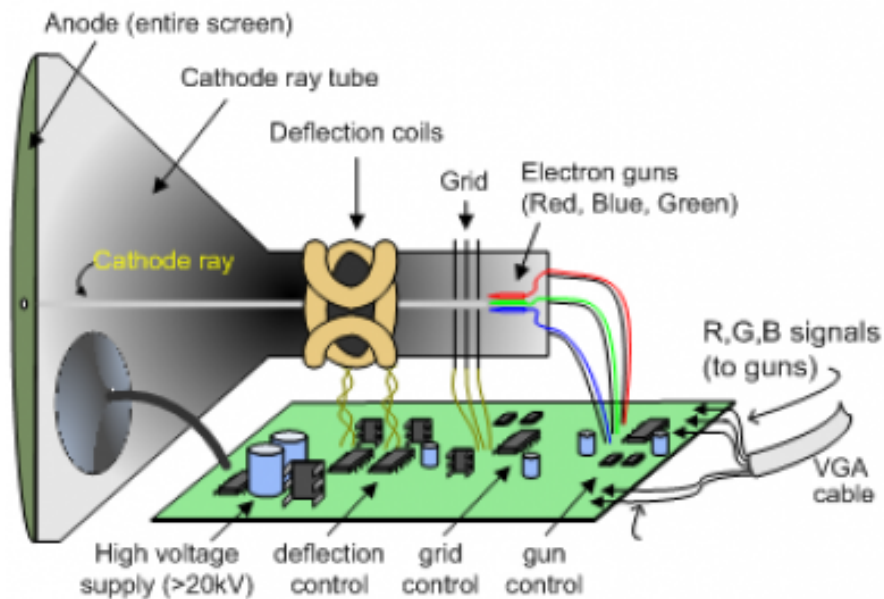


Figura 3.22: Esquema de um monitor CRT.

3.3.8 Video Graphics Array (VGA)

A norma VGA especifica a transmissão de uma imagem em monitores de computador usando sinais analógicos. Embora o VGA tenha sido substituído pela interface digital HDMI, ele ainda é usado e vários monitores recentes continuam a suportar a interface VGA. O VGA possui um conector do tipo D com 15 pinos, onde são passados sinais de sincronismo horizontal e vertical, como também as componentes de cor vermelho, verde e azul, separadamente.

A figura

A listagem ****

A biblioteca VGAX ¹

¹<https://github.com/smaffer/vgax>

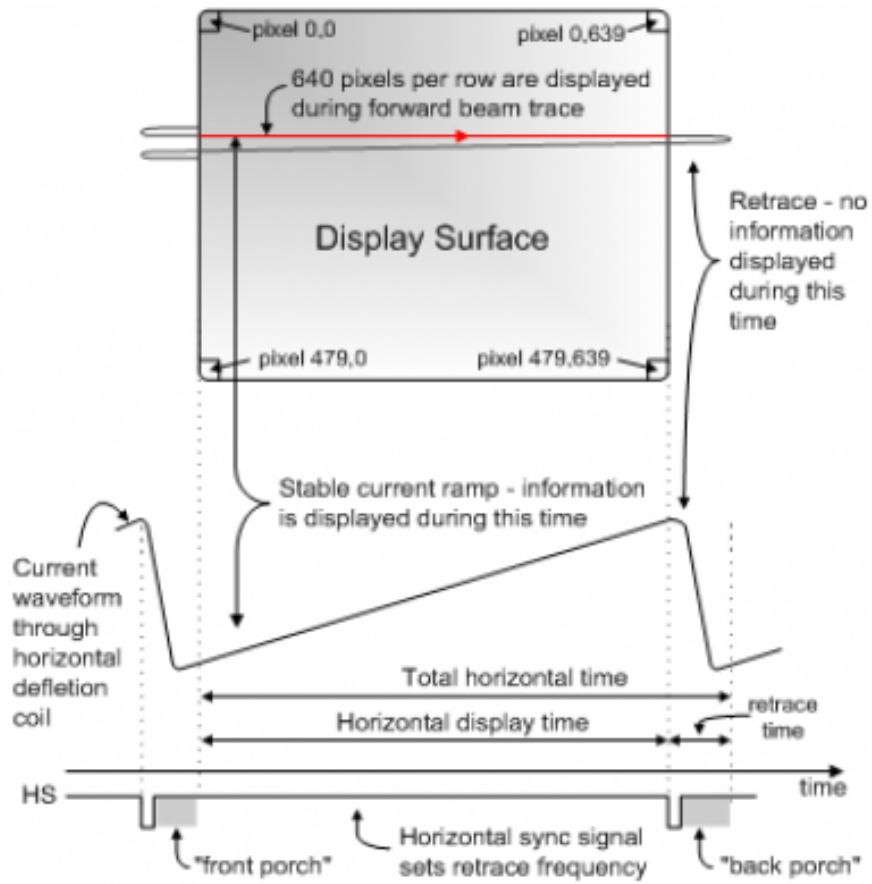


Figura 3.23: Esquema de varrimento do ecrã.

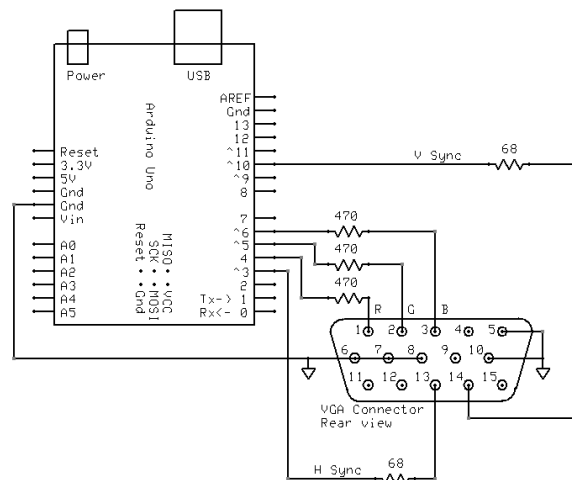


Figura 3.24: Esquema de ligações de um Arduino Uno a um conector VGA.

3.4 Geradores de Som

Os geradores de som servem para dar um sinal sonoro para aviso ou alarme, e são uma alternativa, ou complemento, aos indicadores luminosos nos sistemas embebidos. Os geradores de som são usados desde aplicações básicas, onde apenas emitem um *beep*, até às mais complexas, onde são usados para reproduzir música com elevada fidelidade.

Um gerador de som é definido pelos tons, ou frequências (Hz), que produz, e a sua amplitude ou intensidade de sinal (W ou dB). Nesta secção são apresentados alguns dos atuadores usados em sistemas embebidos para produzir sons, o seu princípio de funcionamento, as suas ligações e a sua programação.

3.4.1 Besouro

O besouro apenas emite um tom quando ativo, e que é determinado pelas dimensões e materiais dos componentes usados na sua construção. O besouro é construído de forma semelhante a um relé, com uma bobina e fulcro metálico móvel, mas em vez de comutar um conjunto de contactos, ele comuta a sua própria alimentação através de uma patilha metálica que oscila. Esta patilha, quando em repouso, permite a passagem de CC por um dos terminais que alimenta a sua bobina. Quando é atuado, o electroímã atrai a patilha para si, o que faz com que seja retirada alimentação ao relé fazendo-o voltar à posição de repouso. Este liga-desliga é o que faz o besouro oscilar constantemente e o toque dos contactos metálicos é o que produz o som metálico caraterístico. A figura 3.25 ilustra o modelo simplificado de construção de um besouro. Este princípio de funcionamento é também usado em algumas buzinas de automóveis. A figura 3.26

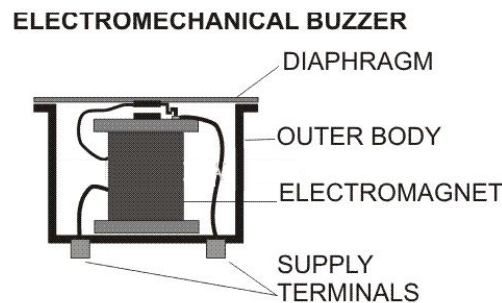


Figura 3.25: Beeper.

Existem besouros que consomem menos corrente do que o máximo disponível num pino de E/S do Arduino, contudo, dado o carácter indutivo da sua carga (bobina), ele deve ser ligado a um circuito condicionador de sinal com MOSFET do tipo *open-drain*, para proteger o porto de E/S do MCU.

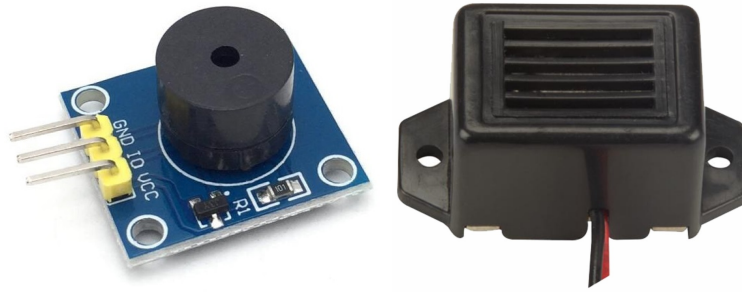


Figura 3.26: Dois tipos de besouro.

A atuação do besouro é do tipo ON-OFF, e pode ser programado com um objeto do tipo `basicAtuator`, descrito no início deste capítulo. A listagem 3.11 exemplifica a programação de um Arduino Uno para controlo de um pequeno besouro ligado ao pino 8 do Arduino Uno. Este Sketch demonstra que se o besouro for ligado e desligado muito rapidamente (ms) ele irá produzir outros tons.

Listagem 3.11: `Passive_Buzzer.ino`

```
//Example for buzzer

int buzzerPin = 8; // setting controls the digital IO foot buzzer

void setup () {
  pinMode (buzzer, OUTPUT) ;// set the digital IO pin mode, OUTPUT out of Wen
}

void loop() {
  unsigned char i, j; // define variables
  while (1) {
    for (i = 0; i < 200; i++) { // delay controls the sound
      digitalWrite(buzzerPin, HIGH); // on
      delay (1); // Delay 1ms
      digitalWrite(buzzerPin, LOW); // off
      delay (1);
    }
    for (i = 0; i < 333; i++) // Wen Qie out another frequency sound
    {
      digitalWrite(buzzerPin, HIGH); // on
      delay(2); // delay 2ms
      digitalWrite (buzzerPin, LOW); // off
      delay(2);
    }
  }
}
```

3.4.2 Piezoelétrico

O gerador de som piezoelétrico funciona com base num fenómeno físico em que ao ter uma tensão aplicada, o material cristalino deforma-se. Ao ser acoplado a uma estrutura semi-rígida, ela própria irá também acompanhar e amplificar o movimento. Ao repetir as deformações dezenas ou milhares de vezes por segundo, o ar no redor da estrutura é movido e o som é propagado.

A vantagem dos geradores de som piezoelétricos são bastante usados em equipamentos portáteis devido ao seu muito baixo consumo e dimensões reduzidas, tal como por exemplo, em relógios de pulso e mini-jogos eletrônicos. Estes geradores de som são capazes de produzir vários tons, mas o seu espectro de frequências é bastante limitado (sons agudos).

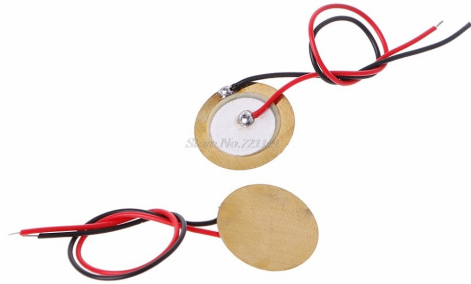


Figura 3.27: Emissor de som piezoelétrico.

A atuação do gerador de som piezoelétrico é feita com base na aplicação de um sinal que oscila na frequência do tom que se quer produzir, e devido ao seu muito baixo consumo e tensão de alimentação, pode ser alimentado diretamente pelo porto de E/S do MCU.

A programação de um tom para ser produzido é feito com a função `tone()` existente no Arduino. Esta função recebe como parâmetros o pino e a frequência de oscilação do sinal. Esta função tem um terceiro parâmetro opcional que é a duração do tom em milissegundos. Esta função não é capaz de gerar tons de frequência inferior a 31 Hz.

A listagem 3.12 corresponde ao Sketch `tone2` de demonstração que acompanha o Arduino, e traduz o valor analógico colocado na entrada A0, dependente da luz incidente na LDR, e gera um som com uma frequência correspondente ao nível de sinal analógico. Quando maior for a tensão do sinal analógico, maior será a frequência do sinal de saída. A figura 3.28 mostra o circuito usado neste exemplo.

Em alternativa, a entrada analógica pode ser ligada a uma potenciômetro de 10k Ω como um divisor de tensão.

Listagem 3.12: Sound_Digital.ino

```
/*
  Pitch follower
  Plays a pitch that changes based on a changing analog input
*/
```

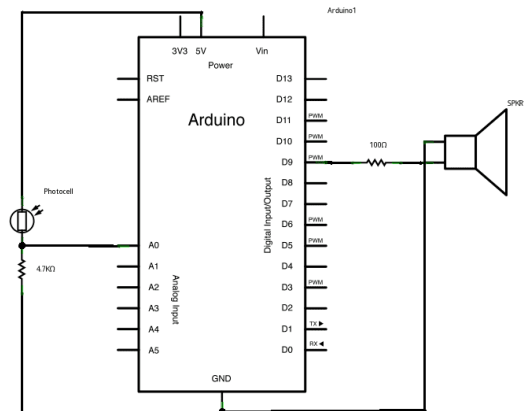


Figura 3.28: Montagem piezo.

```

circuit:
- 8 ohm speaker on digital pin 9
- photoresistor on analog 0 to 5V
- 4.7 kilohm resistor on analog 0 to ground

created 21 Jan 2010
modified 31 May 2012
by Tom Igoe, with suggestion from Michael Flynn

This example code is in the public domain.

http://www.arduino.cc/en/Tutorial/Tone2
*/

void setup() {
  // initialize serial communications (for debugging only):
  Serial.begin(9600);
}

void loop() {
  // read the sensor:
  int sensorReading = analogRead(A0);
  // print the sensor reading so you know its range
  Serial.println(sensorReading);
  // map the analog input range (in this case, 400 - 1000 from the photoresistor)
  // to the output pitch range (120 - 1500Hz)
  // change the minimum and maximum input numbers below depending on the range
  // your sensor's giving:
  int thisPitch = map(sensorReading, 400, 1000, 120, 1500);

  // play the pitch:
  tone(9, thisPitch, 10);
  delay(1);          // delay in between reads for stability
}

```

3.4.3 Altifalante

O altifalante é o mais completo dos geradores de som, a nível da capacidade de gerar a maior gama dos sons, sendo usado em sistemas de som de alta-fidelidade. No entanto, é o mais complexo de todos. A construção de um altifalante está ilustrada na figura 3.29, e tem por base um eletroímã cujo

núcleo de material ferro-magnético se move em função da variação do campo magnético gerado pelas tensões aplicadas nos terminais da bobina. O núcleo por sua vez está colado a uma folha de espuma ou plástico de forma circular. A outra extremidade da folha está colada a um aro de metal. O som é produzido com base na oscilação do ar causada pela vibração do núcleo de metal.

Existem altifalantes de diversos tamanhos, respostas em frequência e potências, sendo os maiores os mais potentes. A ativação de um altifalante é feito normalmente com um amplificador áudio com potência abaixo do limite máximo da coluna. A figura 3.30 mostra um pequeno circuito amplificador de 3 W, suficiente para ser usado em pequenos sistemas embebidos, onde não se espera geração de som de grande qualidade. O amplificador tem a sua entrada ligada à saída de som do sistema embebido, e a sua saída ligada às colunas.

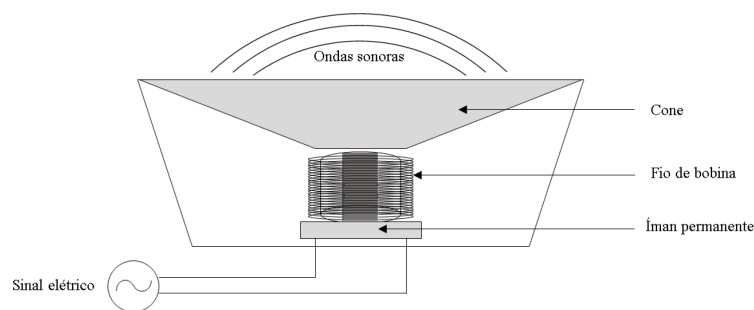


Figura 3.29: Esquema de construção de um altifalante.

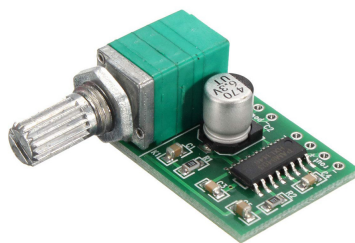


Figura 3.30: Circuito amplificador estéreo de 3W.

A programação de um tom para ser produzido por um altifalante é feito com a função `tone()`, tal como para o gerador de som piezoelétrico.

Para reproduzir música, como um conjunto de notas musicais, para além do tom é necessário a duração da nota. A duração é representada em termos

CAPÍTULO 3. ATUADORES

de divisões do tempo de compasso: mínima (1), semínima (1/2), quarta (1/4) e oitava (1/8).

A listagem 3.13 ilustra a programação de um Arduino Uno ligado a um pequeno altifalante que toca a música popular infantil “Atirei o pau ao gato”, cuja partitura se encontra na figura 3.32. Neste exemplo, cada nota da música é codificada com dois vetores do tipo inteiro **tone** e **duration**, que são sequencialmente lidos e cada tom gerado seguido de um período de pausa dado pela duração da nota.

O exemplo dado é genérico e funciona com 8 oitavas, significando que gera tons desde os 31 aos 4978 Hz. Se forem usados menos tons, por exemplo a 3^a oitava, é possível usar um domínio de representação menor do que os valores inteiros e poupar espaço em memória, guardando o valor do primeiro tom como constante. O mesmo serve para a codificação dos valores da duração das notas.

Listagem 3.13: Sound_Analog.ino

```
int sensorPin = A0; // select the input pin for the potentiometer
int ledPin = 13; // select the pin for the LED
int sensorValue = 0; // variable to store the value coming from the sensor

void setup ()
{
  pinMode (ledPin, OUTPUT);
  Serial.begin (9600);
}

void loop ()
{
  sensorValue = analogRead (sensorPin);
  digitalWrite (ledPin, HIGH);
  delay (sensorValue);
  digitalWrite (ledPin, LOW);
  delay (sensorValue);
  Serial.println (sensorValue, DEC);
}
```



Figura 3.31: Símbolos que representam o tempo de cada nota na armadura de clave: semibreve, mínima, colcheia, e semicolcheia.

Arduíno **Atirei o pau ao gato**

G F E D E F G G G A G F F F G F E E E C C

A - ti-rei o pau ao ga - to - to mas o ga - to - to não mor-reu - eu - eu do-na

5 A A A B A G G G E F G E F G F E D C

chi - ca - ca as-sus-tou-se - se com o ber - ro com o ber-ro que o ga-to deu (Miau)

Figura 3.32: Partitura da música popular “Atirei o pau ao gato”.

3.5 Comutador de Circuitos

Para controlar periféricos ou equipamentos ligados à rede elétrica, o que envolve níveis de tensões e correntes bastante superiores às que estão envolvidas num sistema embebido, não se podem usar os circuitos condicionadores apresentados anteriormente. Para realizar este tipo de controlo os dois componentes mais comuns, e que são apresentados de seguida: o relé, o SCR, e o TRIAC.

3.5.1 Relé Eletro-Mecânico

O relé é um comutador eletro-mecânico, ou seja, é como um interruptor comandado por um electro-íman, que quando energizado faz mover uma haste que controla a posição dos contactos do interruptor de forma semelhante à atuação manual de um interruptor. A figura 3.33 mostra um circuito com relés e o modelo de construção de um relé.

A figura mostra um relé com dois circuitos e duas posições, conhecido como DCDP (dual-circuit, dual-positions). Ao comutar os dois circuitos, ou sinais de alimentação, este tipo de relé permite desligar um periférico por completo do circuito, o que é importante para garantir que um circuito ligado à corrente elétrica não pode ficar com os 220 V no seu circuito.

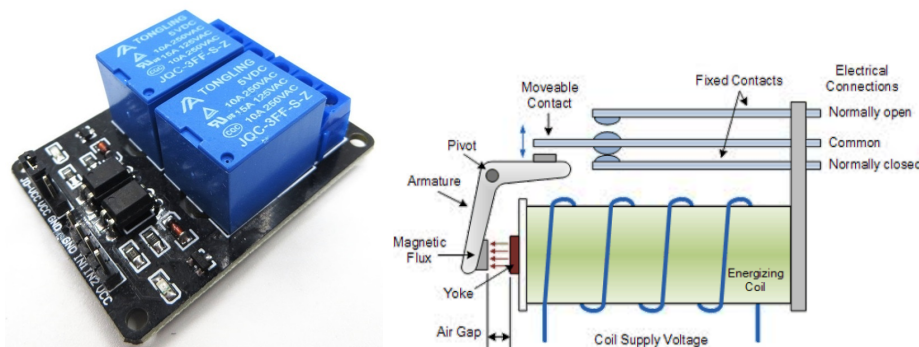


Figura 3.33: Símbolo de um relé (esquerda) e o interior de um relé 1 circuito com duas posições (direita).

Embora o relé permita comutar cargas que têm características bastante diferentes das do Arduino, o eletro-íman no seu interior necessita de tensão e corrente que os pinos de E/S Arduino não conseguem fornecer. Neste caso é necessário usar um BJT ou um MOSFET para realizar o condicionamento de sinal necessário.

A listagem 3.14 ilustra a utilização da classe Rele num Sketch para controlo de um periférico ligado a 220V.

Listagem 3.14: Relay.ino

3.5. COMUTADOR DE CIRCUITOS

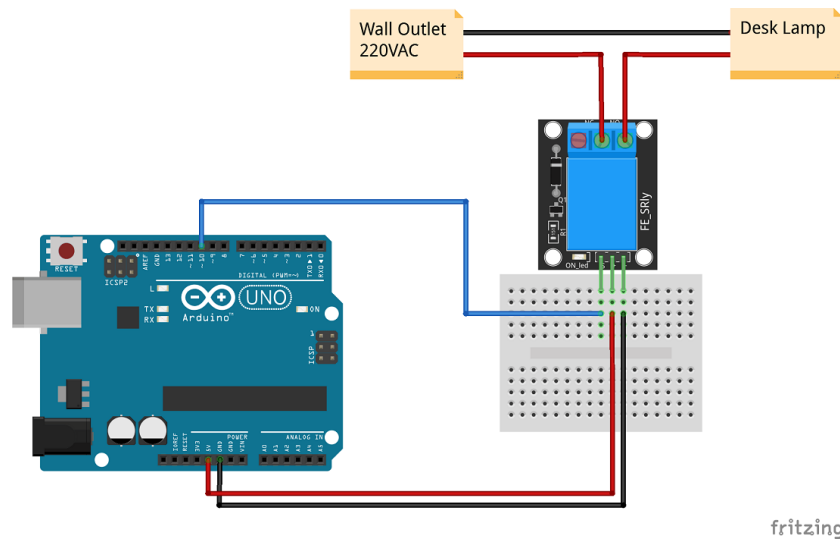


Figura 3.34: Circuito de demonstração de um relé ligado a um Arduino.

```
class Relay {  
  private:  
    int relayPin;  
    bool state;  
  public:  
    Relé(int pin) {  
      relayPin = pin;  
      pinMode (relayPin, OUTPUT); // Define pin as output;  
    }  
    void on() {  
      relayState = HIGH;  
      digitalWrite (relayPin, relayState);  
    }  
    void off() {  
      relayState = LOW;  
      digitalWrite (relayPin, relayState);  
    }  
    void setState(bool state) {  
      relayState = state;  
      digitalWrite (relayPin, relayState);  
    }  
    bool getState() { return relayState; }  
}  
  
void setup ()  
{  
}  
void loop ()  
{  
  relay.setState  
  delay (1000);  
  digitalWrite (relay, LOW); // relay switch is turned off;  
  delay (1000);  
}
```

3.5.2 Relé Estado Sólido

Relé de estado sólido - *Solid State Relay* (SSR) é o nome dado a componentes semicondutores capazes de realizar a mesma funcionalidade de um relé, mas

sem partes móveis. Se no caso de um pequeno sistema a funcionar com tensão CC, é possível ligá-lo com um MOSFET, no caso de uma bomba de água, ou lâmpada de 220 V, o circuito condicionador com um simples MOSFET não pode ser usado por causa das tensões e correntes envolvidas. Os componentes mais comuns capazes de realizar esta funcionalidade são os TRIACs e os SCRs, que recebem um sinal de comando de baixa tensão, e estabelecem a ligação entre dois pontos da carga ligada à rede elétrica, e foram apresentados no capítulo ???. A figura 3.35 ilustra um circuito para controlo de uma lâmpada de 220 V CA com um TRIAC por um Arduino Uno.

As vantagens do uso de TRIAC, em relação aos relés, são a ausência de ruído na comutação do circuito, comutação do circuito mais rápida, e menor consumo de corrente para fazer atuar o circuito. No entanto, os relés são os únicos componentes que ligam e desligam as ligações físicas entre componentes de um circuito.

Enquanto que o TRIAC necessita de mais componentes no circuito, o SSR ao englobar o opto-isolador no seu corpo está preparado para ligar diretamente ao Arduino.

É boa prática isolar o sinal de comando do sistema embebido ao circuito condicionador de componentes ligados à rede elétrica através de isoladores elétricos, tais como os foto-acopladores e transformadores galvânicos.

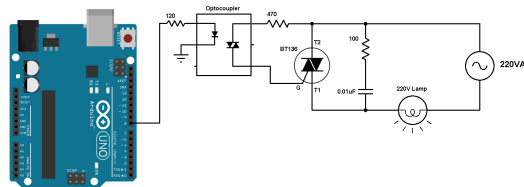


Figura 3.35: Controlo de uma lâmpada de 220V com um TRIAC.

A figura 3.38 mostra a placa de demonstração do relé de estado sólido SSR-2F4805. Este componente é caracterizado por receber um sinal de controlo entre 3 e 32 V, no máximo consome 25 mA, e é capaz de comutar cargas ligadas entre 40 e 480 V CA e consumirem até 5 A. Esta placa tem incluído o circuito de condicionamento do sinal de comando baseado no transistor MMBT5551 (identificado com a marca Q1 num encapsulamento SOT-23) numa montagem em emissor comum², para reduzir a corrente necessária fornecer pelo pino de E/S do Arduino, e com uma resistência de *pull-down* ligada entre a base do transistor e GND para evitar ativações caso o sinal de controlo não esteja ligado. Esta placa tem ainda um LED para dar indicação

²A página do fabricante da placa de demonstração (<https://www.sainsmart.com/products/2-channel-3v-32v-solid-state-relay>) mostra um esquema elétrico mas incorreto, pois a base e o coletor dos transistores estão trocados. Isto pode ser verificado consultando o *datasheet* e o desenho da PCB.

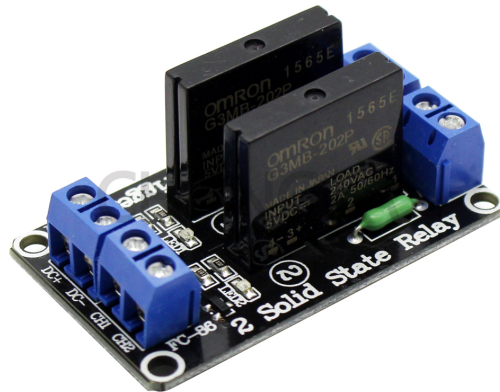


Figura 3.36: Relé de estado sólido.

visual do estado do SSR. O esquema do circuito da placa de desenvolvimento é mostrado na figura 3.37.

A figura 3.38 mostra o aparato para demonstração da utilização de um SSR ligado a iluminador de 220 V ligado a um Arduino Uno

A programação do Arduino pode ser feita com um objeto da classe `basicAtuator`, explicado anteriormente.

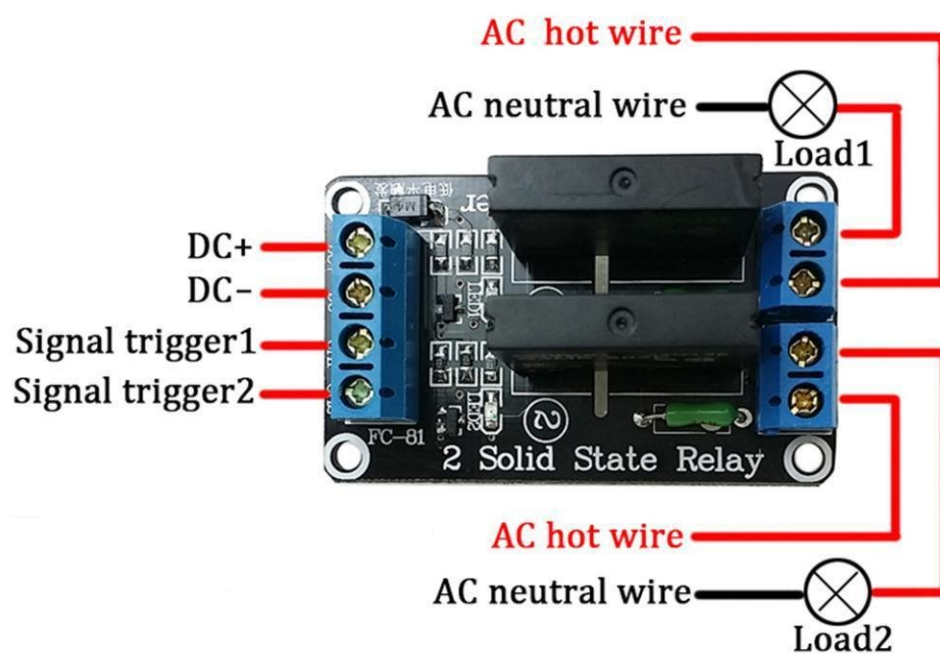


Figura 3.37: Circuito de demonstração do relé de estado sólido (SSR-2F4805) ligado a um Arduino Uno.

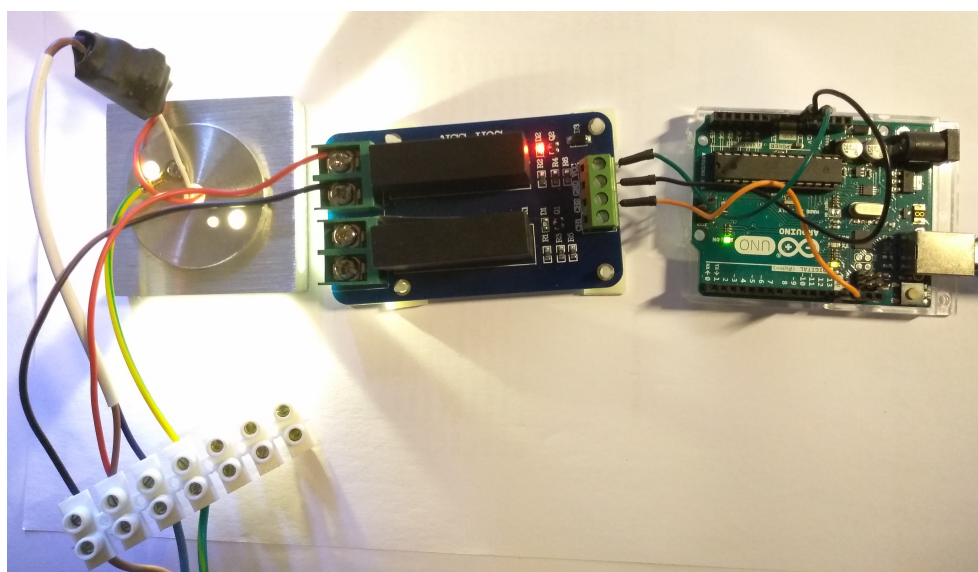


Figura 3.38: Circuito de demonstração do relé de estado sólido (SSR-2F4805) ligado a iluminador de 220 V, controlado por um Arduino Uno.

3.6 Temperatura

A variação de temperatura é usada em sistemas que necessitem de aquecer ou arrefecer um determinado material sólido, líquido ou gasoso, para estar numa determinada gama de temperaturas. As seções seguintes são dedicadas a componentes que permitem variar a temperatura, tais como as resistências de aquecimento e os elementos de Peltier.

3.6.1 Resistências de Aquecimento

Uma resistência quando percorrida por uma corrente dissipa energia sob a forma de calor (processo de Joule). Existem algumas resistências cuja funcionalidade é apenas libertarem calor! As resistências de aquecimento são componentes que quando alimentados aumentam a sua temperatura, e são caracterizadas pela tensão de funcionamento, temperatura máxima [°C] que atingem, ou a potência dissipada [W]. A potência de uma resistência de aquecimento é o produto da sua tensão de funcionamento e corrente consumida ($P = V.I$). Por exemplo, uma resistência de aquecimento de 120 W ligada a uma fonte de tensão de 12 V consome 10 A. As resistências de aquecimento podem funcionar em CC ou CA.

A figura 3.39 mostra o aspeto de uma resistência de aquecimento de um ferro de soldar.

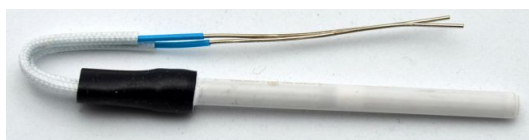


Figura 3.39: Resistência de aquecimento de um ferro de soldar.

Se a temperatura a atingir admitir a variação de vários graus centígrados, pode-se usar um controlador do tipo ON-OFF, que pode ser programado com um objecto do tipo `basicAtuator`. Se for necessário ter um controlo rigoroso da temperatura, com variação abaixo do grau centígrado, deve-se usar um controlador PID. Neste caso o atuador pode usar sinais digitais PWM ou analógicos para fazer variar a quantidade de calor libertada pelo componente.

Para controlo das resistências de aquecimento é necessário usar circuitos acondicionadores de sinal tais como: relé electromecânico (ON-OFF, CA/CC), relé de estado sólido ou TRIAC (ON-OFF, PWM, CA), MOSFET (ON-OFF, PWM, CC). Neste caso não há vantagem em usar uma Ponte-H uma vez que a resistência aquece sempre independentemente da direcção da corrente.

3.6.2 Elemento de Peltier

O elemento de Peltier, também conhecido como *ThermoElectric Cooler* (TEC) é um tipo de dispositivo com duas faces separadas por material isolante, que quando em funcionamento uma delas aquece e a outra arrefece, podendo as variações de temperatura oscilar entre os 0°C e os 85°C ! Os elementos de Peltier são usados para mini frigoríficos, controlo localizado de temperatura (estabilização de geradores de relógio de elevada precisão).

Os elementos de Peltier são componentes semicondutores cujo funcionamento assenta no efeito de Seebeck. O efeito de Seebeck consiste no arrefecimento ou aquecimento da junção dos materiais semicondutores (tal como um díodo) quando percorrida por uma corrente elétrica. O elemento de Peltier é construído de forma que várias junções PN tenham os elementos P e N em faces opostas, tal como um díodo partido ao meio e ligado pela faces dos elementos P e N e com os terminais curto-circuitados. Esta montagem, quando em funcionamento uma das faces aquece e a outra arrefece. Se se inverter a polaridade da tensão aplicada a face fria irá aquecer e a face quente arrefecer. Um elemento de Peltier é caracterizado pela temperatura máxima entre as duas faces, tensão e corrente máximas.

O efeito de Peltier é reversível, o que significa que um elemento de Peltier com as suas faces expostas a temperaturas diferentes irá gerar uma diferença de potencial aos seus terminais, proporcional à diferença da temperatura. Este efeito é reversível, ou seja, mantendo as faces com temperaturas diferentes, irá gerar uma diferença de potencial nos terminais do elemento de Peltier.

A figura 3.40 mostra o aspeto de um elemento de Peltier. Elementos mais pequenos consomem menos corrente e têm menos diferença de temperatura entre as suas faces do que elementos maiores.



Figura 3.40: Elemento de Peltier (TEC).

Na utilização de elementos de Peltier é necessário ter em atenção fenómenos relacionados com termodinâmica, tal como o ponto de orvalho, que determina qual a temperatura de uma superfície que origina a formação de condensação,

dada a temperatura ambiente e a humidade relativa do ar. Na sua instalação é habitual aplicar pasta térmica para facilitar a condução térmica com as superfícies em contacto com as suas faces, contudo é necessário cuidado para não deixar excesso de pasta em contacto com as duas faces, pois dá origem a um “curto-circuito térmico” entre elas.

Quando se liga um elemento de Peltier, ele encontra-se à temperatura ambiente, e a temperatura numa face vai baixar e a outra aumentar em torno dessa temperatura, contudo à medida que a diferença de temperatura aumenta, o ponto médio das temperaturas aumenta e a face quente irá tornar-se muito mais quente do que a face fria em relação à temperatura ambiente.

Se o objectivo da montagem for arrefecer uma superfície, semelhante à construção de um mini-frigorífico, uma base fria para refrigerantes, ou quente para café e chá é necessário “remover” o calor da face quente através de um dissipador com uma ventoinha acoplada ou um sistema de arrefecimento a água. Desta forma é possível ter faces a temperaturas tão díspares como 0°C e 80°C.

Tendo em conta que a diferença de temperatura entre as faces do elemento de Peltier é proporcional à tensão aplicada, o seu controlo pode ser feito com um MOSFET se a intensão for apenas manter fixa a diferença de temperatura nas suas faces, como por exemplo arrefecer uma lata de refrigerante. Se for necessário aquecer e arrefecer com a mesma face, para contrariar o efeito da temperatura ambiente, é necessário usar o circuito de Ponte-H para inverter a polaridade do elemento de Peltier.

Para controlo da temperatura recomenda-se um circuito de controlo da tensão de alimentação do elemento de Peltier em vez do PWM, porque os pulsos VCC-GND produzidos pelo PWM reduzem o tempo de vida útil do componente. Se não houver alternativa, sugere-se que se use o PWM com uma frequência de oscilação muito baixa ($< 100\text{ Hz}$) e que sejam colocados condensadores de elevada capacidade ($> 1000\text{ }\mu\text{F}$) entre os terminais do componente e GND, e duas bobinas em série com os terminais do componente e o MOSFET, tal como ilustrado na figura 3.41.

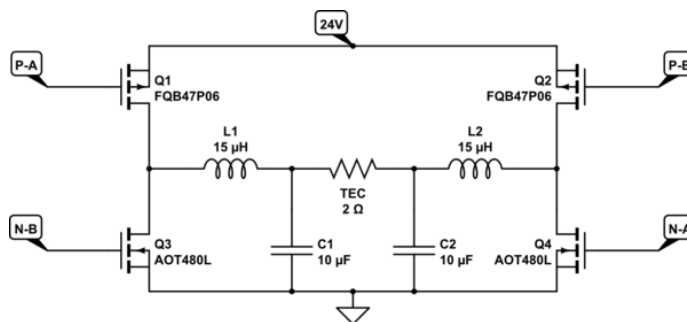


Figura 3.41: Ligação de um Elemento de Peltier a uma Ponte-H.

A título de curiosidade ... ver comentario latex

3.7 Motores

Os motores são usados em sistemas eletrônicos para promover a rotação de um eixo para o deslocamento do próprio sistema ou de outros objetos. Eles são muito usados na construção de robots autónomos, braços robóticos em linhas de montagem, portas automáticas, dispensadores automáticos, e ventiladores/extratores. Existem vários tipos de motores, com funcionamento distintos, usados para diferentes tipos de aplicações. As próximas seções abordam 3 tipos de motores usados em sistemas embebidos: motor CC, motor-de-passo e servo-motor.

3.7.1 Motor CC

O motor CC gira enquanto tiver energia elétrica CC aplicada nos seus terminais. O sentido da rotação é determinado pelo sentido da corrente que o percorre. A velocidade do motor depende da sua construção e das suas condições de alimentação (tensão e corrente). Não é possível controlar a orientação, ou posição, do seu veio tal como é construído sem um aparato electro-mecânico adicional.

O motor CC tem um veio ligado no seu interior que gira quando o motor é alimentado. Dentro do motor, existe um, ou mais, ímanes permanentes colocados na proximidade das bobinas. Ao veio do motor estão acopladas bobinas, que ao serem percorridas por uma corrente elétrica criam um campo eletromagnético que, interagindo com o campo magnético dos ímanes vai provocar a rotação do veio.

A figura 3.42 mostra o aspeto de um pequeno motor CC (esquerda) e os seus componentes (direita). Este motor pode ser encontrado em diversos brinquedos telecomandados, tais como carros e aviões, robots, e ventoinhas de arrefecimento de computadores e circuitos eletrónicos.

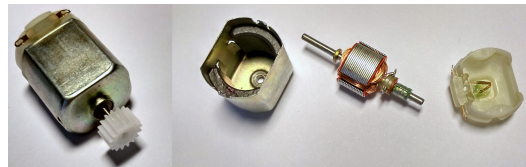


Figura 3.42: Pequeno motor DC.

A figura 3.43 ilustra o funcionamento simplificado do motor DC. Os terminais do exterior motor estão ligados a um comutador por um par de escovas, que faz com que a corrente que circula nas bobinas tenha sempre o mesmo sentido para cada da posição do eixo do motor. A bobina ao ser percorrida por uma corrente elétrica cria um campo magnético com a mesma orientação (polaridade) que o íman adjacente, fazendo com que cada bobina seja repelida por um íman e atraída pelo outro, provocando a rotação do veio.

À medida que o veio gira e os campos magnéticos (de polaridade oposta) estão praticamente alinhados, o comutador alterna a polaridade do campo magnético das bobinas.

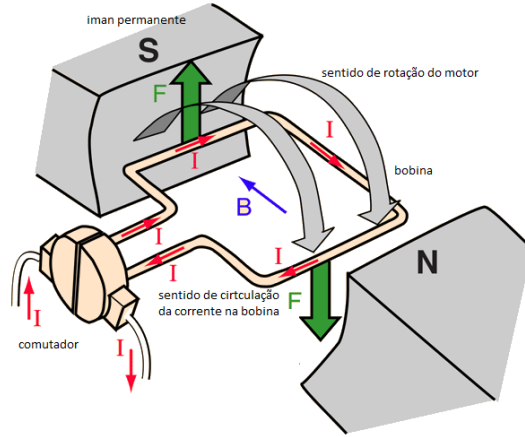


Figura 3.43: Funcionamento do motor DC.

A força de rotação de um motor é designada por torque. O torque é inversamente proporcional à velocidade do motor. Quando não existe nenhuma força inercial que contrarie a rotação do motor, este roda com a sua velocidade máxima. Por este motivo é que um pequeno robot numa subida acentuada anda mais devagar do que quando num plano horizontal.

O torque é medido em Newton/metro (N/m). A potência do motor (W) corresponde ao produto da velocidade angular pelo torque:

$$1 \text{ W} = 1 \text{ N.m/s} \quad (3.1)$$

A velocidade angular é o numero de revoluções completas num segundo, é expressa em radianos por segundo (rad/s) e relaciona-se com a medida de rotações por minuto (RPM) através da expressão:

$$\omega_{rad/s} = \omega_{rpm} \cdot \frac{2\pi}{60} \quad (3.2)$$

Para o exemplo de um pequeno motor DC equipado com engrenagem de redução, ilustrado na figura 3.44, a sua velocidade angular é:

$$\omega = 260 \text{ rpm} = 260 \cdot 2\pi / 60 \text{ rad/s} = 27,2 \text{ rad/s} \quad (3.3)$$

e a sua potência absorvida é:

$$P[\text{Watt}] = V.I[\text{Volt.Ampere}] = 6 \times 0,12 = 0,72 \text{ W} \quad (3.4)$$

De acordo com estas caraterísticas elétricas, o motor tem o torque:

$$T[\text{Watt}] = P/\omega = \frac{0,72}{27,2} = 0,0264 \text{ N.m} \quad (3.5)$$

Dados Técnicos:

Tensão de alimentação dos motores	3 ~ 6VDC
Corrente máxima	120mA
Relação da redução	48:1
Rotação do motor a 6VDC	260 rpm
Velocidade do carro a 6VDC	1 m/s
Dimensões do chassi	22 x 14,7 cm
Dimensões da roda	7 x 7 x 2,6 cm
Perímetro da roda	22cm



Figura 3.44: Especificações técnicas de um motor DC equipado com redução.

Sabendo que a roda tem 22 cm (0,22 m) de perímetro, equivalente a um raio de 3,5 cm, é possível estimar qual a força aplicada pelo motor:

$$F = \frac{T}{r} = 0,75 \text{ N} \quad (3.6)$$

A velocidade do motor é dada pelo quociente da potência do motor pela força que ele exerce, o que dá:

$$v = \frac{P}{F} = \frac{0,72}{0,75} \approx 1 \text{ m/s} \quad (3.7)$$

O que está de acordo com o *datasheet* do motor CC apresentado. Depois da redução de 48:1 o torque do motor sobre as rodas aumenta para 1,26 *N.m*.

O controle da rotação do motor CC é feito através das condições de alimentação do motor. O motor exibido na figura 3.42 funciona com uma tensão entre 3 e 6 V e consome 300 mA corrente quando alimentado a 3 V. Nestas condições, o motor gira com a velocidade de cerca de 1700 *rpm* sem carga mecânica aplicada no eixo. O eixo do motor tem 2 mm de diâmetro, e o corpo do motor 27 × 20 × 15 *mm*.

O controle da velocidade do motor pode ser feito de duas formas, tal como é feito no controle da intensidade luminosa de um LED: variação da alimentação, ou variação da largura do período de funcionamento de um pulso retangular - usando os pinos PWM do Arduino. A segunda opção necessita de menos componentes, no entanto requer instruções extra para controlar a largura do pulso PWM.

Como o motor consome mais corrente do que a que Arduino consegue fornecer, é necessário usar um circuito condicionador que seja capaz de traduzir o sinal de ativação dado pelo Arduino e a ligação à alimentação do motor CC. Como a ativação é do tipo ON/OFF basta usar um transistor para comandar o motor. Dadas as condições de funcionamento do motor CC (3 V, 400 mA) e do sinal de comando do Arduino (LOW: 0-0,8 V, HIGH: 4,1-5 V ou 2,3-3,3 V dependendo do valor de VCC).

Para programar o motor CC com um Arduíno foi criada a classe **MotorDC**. Esta classe no construtor recebe qual o pino usado no controlo do motor. Os atributos privados da classe são o número do pino no Arduíno, o estado do motor e qual a velocidade relativa (0-100%). Esta classe tem métodos públicos para iniciar o motor no estado parado **void init()**, ligar **void on()** e desligar **void off()** o motor, e definir a sua velocidade **speed(byte spd)**. O argumento **spd** é um valor inteiro entre 0 e 100, correspondente à taxa de rotação so motor (0-100%).

A listagem 3.15 ilustra a programação do motor DC.

Listagem 3.15: Sketch MotorDC.ino

```
class MotorDC {
  private:
    bool state; // ON/OFF = HIGH/LOW
    byte speed; // 0-100%
    byte pin; // output pin
  public:
    MotorDC(byte pinMotor, byte senseMotor = 0) {
      state = LOW;
      speed = 0;
      this->pin = pin;
    }
    void init() {
      pinMode(pin, OUTPUT);
    }
    void on() {
      state = HIGH;
    }
    void off() {
      state = LOW;
    }
    void setSpeed(byte spd) {
      speed = 255 * (int)spd / 100;
    }
    byte getState() {
      if(state == LOW) return 0;
      else return speed;
    }
};

class HallSensor{
  private:
    byte pin; // one of the INTx
    const unsigned long measureTime = 1000000; // 1s
    unsigned int rpm; // 64k rpm sounds enough for most applications
    unsigned long lastStart;
  public:
    unsigned int getRPM() {
      return rpm;
    }
    void measureRPM() {
    }
};

#if 0
class FanDC : MotorDC , HallSensor {
  private:
    byte sense; // rpm sensing input pin
  public:
    FanDC(byte motorPin, byte sensorPin) {
      //Motor(motorPin);
    }
};
```

```
};  
#endif  
void setup()  
{  
  Serial.begin(9600);  
}  
  
void loop()  
{  
  if (Serial.available() > 0) {  
    Serial.print(Serial.read());  
  }  
}
```

Para controlar o sentido de rotação do motor, pode-se usar o circuito da ponte-H. Este circuito requer o uso de um sinal de saída adicional no Arduino.

Existe uma variante de motor CC que não possui escovas de ligação ao comutador, e que é construído com um íman permanente acoplado ao eixo que roda e as bobinas encontram-se fixas. Por não terem contacto mecânico permanente com o interior do motor, eles permitem construções mais compactas, são mais duráveis e silenciosos. Este tipo de motor é usado em ventoinhas de arrefecimento de computadores, leitores de discos óticos (CD/DVD), drones e trotinetes elétricas.

3.7.2 Motor de Passo

O motor de passo permite controlar com precisão a rotação do veio do motor. Este tipo de motores é usado para controlar a posição ao longo de um eixo, como por exemplo em impressoras 3D, robots, posicionamento de lasers e espelhos, digitalizadores de documentos, e leitores de CD/DVD. A grande vantagem deste tipo de motores é a sua simplicidade e fiabilidade.

O motor CC possui as bobinas acopladas ao veio, com ímans permanentes à sua volta. O motor de passo possui um íman permanente acoplado ao veio com bobinas à sua volta. Enquanto que o motor CC roda enquanto tiver energia aos seus terminais, no motor de passo a posição do veio avança passo-a-passo, e depende de quais as bobinas energizadas num determinado instante. A rotação do motor envolve a energização das várias bobinas em sequência.

A figura 3.45 ilustra o funcionamento do motor de passo, com os instantes de ativação das bobinas do motor. Um motor de passo ligado a um Arduino requer que os sinais de saída sejam programados para reproduzir a mesma sequência.

A figura 3.47 ilustra o circuito de demonstração de um motor de passo ligado a um Arduino Uno. Como os portos de E/S Arduino não conseguem fornecer corrente suficiente para as bobinas existentes no motor de passo, é

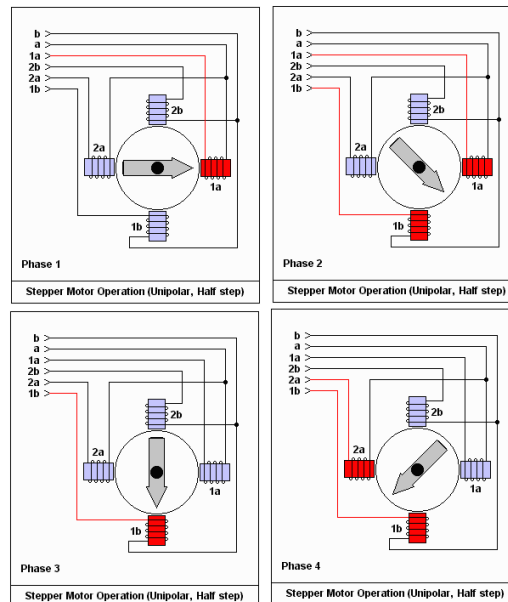


Figura 3.45: Etapas do funcionamento do motor de passo.

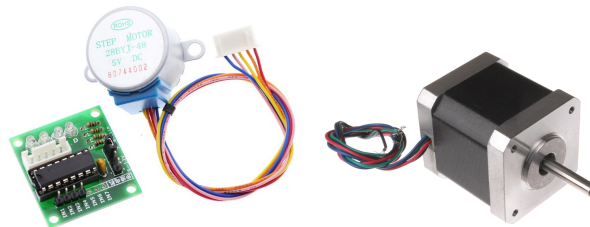


Figura 3.46: Motores de passo.

necessário adicionar o circuito condicionador de sinal que permite a ativação do motor usando os sinais de E/S do Arduino. O circuito condicionador pode ser realizado com um conjunto de transistores discretos (TIP31 ou TIP32), ou em circuito integrado (por exemplo ULN2003 ou ULN2004), que suportem a corrente que percorre cada bobina. Existem também módulos com os conectores próprios para ligar o motor.

A listagem 3.16 ilustra a programação do motor de passo. Este Sketch usa a biblioteca “Stepper” já existente no Arduino para criar o objeto Stepper que recebe como parâmetros o número de passos do motor e os números dos pinos de E/S no Arduino. Esta classe oferece ainda os métodos `setSpeed(num_rpms)` para definir a velocidade do motor em termos de rotações por minuto, e `step(num_passos)` que define o número de passos a dar.

Listagem 3.16: Sketch StepMotor.ino

```
/*
 * MotorKnob
 *
```

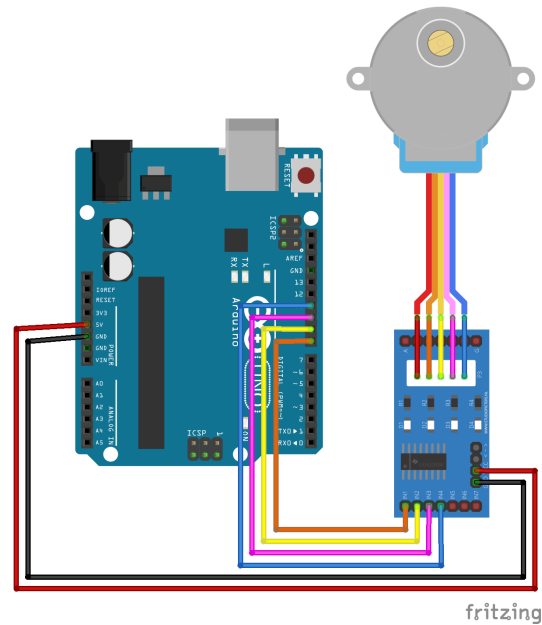


Figura 3.47: Circuito de demonstração de um motor de passo.

```

* A stepper motor follows the turns of a potentiometer
* (or other sensor) on analog input 0.
* 
* http://www.arduino.cc/en/Reference/Stepper
* This example code is in the public domain.
*/

#include <Stepper.h>

// change this to the number of steps on your motor
#define STEPS 100

// create an instance of the stepper class, specifying
// the number of steps of the motor and the pins it's
// attached to
Stepper stepper(STEPS, 8, 9, 10, 11);

// the previous reading from the analog input
int previous = 0;

void setup() {
  // set the speed of the motor to 30 RPMs
  stepper.setSpeed(30);
}

void loop() {
  // get the sensor value
  int val = analogRead(0);

  // move a number of steps equal to the change in the
  // sensor reading
  stepper.step(val - previous);

  // remember the previous value of the sensor
  previous = val;
}

```

3.7.3 Servo-Motor

O servo-motor, permite controlar a posição de um veio em torno de um ponto entre 0° e 180° , ou -90° e 90° . Este componente é usado em modelismo para controlar a direção de carros, barcos e aviões telecomandados - *Radio Controlled* (RC), e é apresentado na figura 3.48 à esquerda. O servo-motor, quando ativo, posicionar-se-á de acordo com o comando recebido. Caso seja contrariado, ele irá compensar exercendo mais força no sentido oposto ao da força aplicada. Quando ele atinge a posição pretendida, ele permanece em repouso.

O servo-motor tem 3 fios de ligação: VCC (vermelho), GND (preto ou castanho) e sinal de controlo (laranja). O sinal de controlo do servo-motor é feito por um sinal periódico, de 50Hz, onde a posição, ou ângulo, é codificada pela largura do pulso - *Pulse Width Modulation* (PWM). Na figura 3.48 à esquerda encontra-se a correspondência entre os pulsos de entrada e o ângulo do servo-motor.

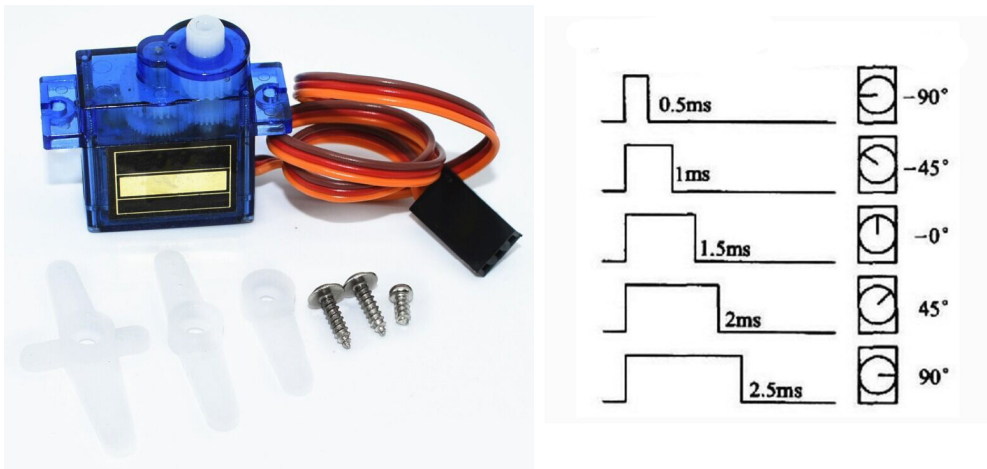


Figura 3.48: Servo-motor.

O ambiente Arduino já vem com uma biblioteca para controlar servos, e pode ser usada incluindo o ficheiro `servo.h` no *sketch*. Este ficheiro fornece a definição da classe `Servo` que permite instanciar um objecto por cada motor a ser controlado. Esta classe fornece os seguintes métodos:

- `attach(int pino)` - associa o número do pino digital ao motor;
- `write(int angulo)` - gera o sinal de comando do motor para se posicionar com o ângulo dado.

Para demonstrar o funcionamento do servo-motor foi usado o SG90. Este modelo funciona com tensões entre 3,2 e 7V, e tem entre 1,2 e 1,6Kg de torque, dependendo da tensão de entrada. Este servo-motor é destinado a modelismo uma vez que apenas pesa 9g. Para

CAPÍTULO 3. ATUADORES

Em termos de ligações, VCC e GND do servo-motor aos 5V e GND do Arduino, respectivamente. O potenciômetro de 10kOhm é usado como um circuito divisor de tensão e o seu ponto médio ligado na entrada analógica A0.

A listagem 3.17 é baseada no exemplo de demonstração que acompanha o ambiente Arduino, e que posiciona o veio do servo em função de um sinal analógico obtido através de um divisor de tensão construído com um potenciômetro de 10 k Ω , tal como ilustrado na figura 3.49. Este Sketch usa a biblioteca `Servo.h` que define a classe `Servo`. Os objetos desta classe possuem os métodos `attach(num_pino)`, que define o numero do pino de E/S onde o motor é ligado, e `write(angulo)` que define o valor do angulo entre 0° e 180°.

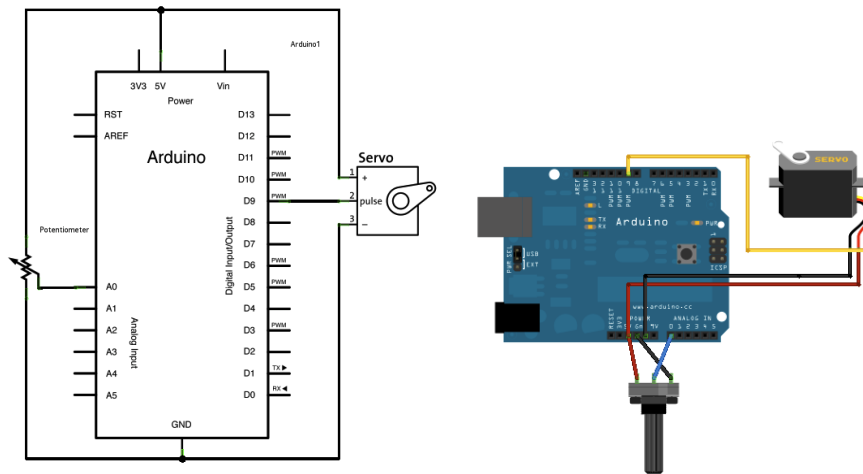


Figura 3.49: Esquema elétrico e protótipo da demonstração do servo-motor.

Em relação à demonstração original no ambiente Arduino, destaca-se o uso de um filtro digital para minimizar o ruído da conversão analógico digital, e na introdução de um condensador eletrolítico de 100 μF /16 V próximo dos terminais do potenciômetro para minimizar as oscilações da tensão na entrada analógica.

Listagem 3.17: Sketch ServoDemo.ino

```
/*  
  Controlling a servo position using a potentiometer (variable resistor)  
  by Michal Rinott <http://people.interaction-ivrea.it/m.rinott>  
  
  modified on 8 Nov 2013  
  by Scott Fitzgerald  
  http://www.arduino.cc/en/Tutorial/Knob  
*/  
  
#include <Servo.h>  
  
Servo myservo;  // create servo object to control a servo  
int potpin = 0;  // analog pin used to connect the potentiometer
```

```

int val;    // variable to read the value from the analog pin
int avg;

void setup() {
  myservo.attach(9); // attaches the servo on pin 9 to the servo object
}

void loop() {
  val = analogRead(potpin);           // reads the value of the potentiometer (value between 0 and 1023)
  avg = (31*avg + val) / 32;
  val = map(avg, 0, 1023, 0, 180);    // scale it to use it with the servo (value between 0 and 180)
  myservo.write(val);                 // sets the servo position according to the scaled value
  delay(15);                           // waits for the servo to get there
}

```

A figura 3.50 mostra as formas de onda do sinal analógico à saída do potenciômetro, que liga à entrada analógica A0, e a tensão de comando do servo.

O leitor é desafiado a realizar este ensaio para observar uma manifestação não desejada. Após colocar o circuito em funcionamento, mesmo não alterando a posição, o motor faz ligeiras oscilações. Tendo colocado um filtro digital no circuito, mas minimizar o ruído na medição, aparentemente não existe motivo para existir variação na posição do motor.

Com este problema em mãos, a forma de o resolver é partir o problema em várias partes e observá-las isoladamente. Neste caso as variáveis do sistema são:

- Tensão analógica indicativa da posição/ângulo - saber se existe tensão na entrada do circuito e se está a codificar a posição do motor. A tensão da posição depende da posição do potenciômetro e da tensão de entrada do circuito;
- Valor digital da posição do motor - observação do valor da variável para perceber se a oscilação tem origem na má manipulação da variável;
- Tensão e corrente consumida pelo motor - observar do motor para verificar se está a ser controlado corretamente;
- Tensão de alimentação - observar se existem variações na tensão de funcionamento do circuito.

Ao iniciar a inspeção de cada um destes sinais com um osciloscópio, foi observado o seguinte:

- A tensão na saída do potenciômetro oscilava com a cadência do movimento do motor;
- O valor da variável após a filtragem digital, quando observado no terminal série ou no traçador gráfico do Arduino, mostra que a variação do valor acompanha a oscilação do motor;

- A corrente consumida pelo servo-motor, em repouso ou fazendo pequenos movimentos é inferior a 10mA, mas quando em movimento, foram medidos cerca de 200mA.
- A tensão de alimentação do circuito (+5V) oscila com a cadência do motor! Isto não é suposto acontecer!

Com estas observações é possível concluir sobre o motivo da oscilação do motor, que não devia acontecer. A conclusão é que o circuito sendo alimentado pelo cabo USB, ligado a um computador, faz com que o cabo USB e as pistas da placa de circuito impresso do Arduino Uno, tendo perdas, fazem com que exista uma queda de tensão significativa face à corrente consumida pelo servo-motor.

A solução passa pela introdução de um condensador para armazenamento de carga que vai compensar as perdas quando o motor se encontra a consumir corrente. Um valor aceitável para o condensador é cerca de 1000 μF /16 V.

3.7. MOTORES



Figura 3.50: Sinais de comando do servo-motor para as posições 0°, 90° e 180°.

3.8 Solenoide

O solenoide é usado para traduzir corrente elétrica em deslocamento físico, e é normalmente usado na construção de fechaduras e trincos elétricos e eletro-válvulas. Ele é construído como uma bobina que têm um núcleo móvel de material ferromagnético, e funciona como electro-ímã que atrai o núcleo para o seu interior quando ligada a uma fonte de alimentação. O seu funcionamento, pode ser comparado ao de um relé, que quando em funcionamento, o eletro-ímã no seu interior provoca o movimento de uma aste, ou êmbolo, com um elemento ferro-magnético. A figura 3.51 ilustra a construção de um solenoide.

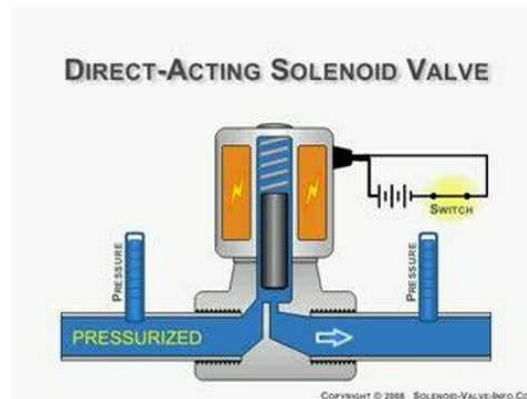


Figura 3.51: Esquema de construção de uma eletro-válvula (solenoide).

Os fatores que importam observar na escolha de um solenoide são:

- Tensão de funcionamento - o componente não funciona com tensão uma inferior à especificada, mas queima com tensão superior;
- Corrente consumida - o circuito de controlo deverá ser capaz de fornecer a corrente necessária para o eletro-ímã mover o êmbolo;
- Força exercida pelo êmbolo - pressão máxima suportada.

Como o consumo de corrente do solenoide e tensão de funcionamento (habitualmente 12 V) são superiores aos fornecidos pelos portos de E/S do Arduino é necessário um circuito condicionador de sinal. A figura 3.52 ilustra as ligações de um circuito para controlar um solenoide. Este circuito tem Como a eletro-válvula funciona a 12 V, o controlo é feito por um MOSFET ligado a um porto de E/S do Arduino. Por ser uma carga indutiva, deve ser usado um MOSFET que inclua um diodo montado inversamente, ou ligado entre o Dreno e a Fonte do MOSFET. Em termos de programação, o solenoide é controlado como outro objeto da classe `basicAtuator`, descrito



As eletro-válvulas funcionam como uma torneira controlada eletricamente, com dois estados: aberta (deixa passar líquido) e fechada (não deixa passar líquido). Elas são usadas para permitir ou bloquear a passagem de líquidos através de um sinal elétrico, e são muito usadas em indústrias química, perfumaria, alimentar, e plantações agrícolas, necessitam de regular ou automatizar o fornecimento de substâncias em estado líquido para manter a produção ininterrupta.

CAPÍTULO 3. ATUADORES

contacto com circuitos elétricos em funcionamento provoca curto-circuitos.