Alexandre Calerio de Oliveira

Motorino
módulo semi-independente para controle de motores dc baseado na plataforma Arduino

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação da Universidade Federal do Paraná. Área de concentração: *Ciência da Computação.*

Orientador: Eduardo Todt.

Curitiba PR
2016

Resumo

Nos dias atuais, temas como internet das coisas, robótica, automação, drones, impressão 3D são bastante frequentes no meio tecnológico e despertam o interesse de uma comunidade cada vez maior de pessoas que se identificam com essas tecnologias. Muitas destas, mesmo possuindo alguma afinidade com linguagens de programação e ambientes de desenvolvimento ainda apresentam dificuldades técnicas quando precisam trabalhar com projetos que envolvam hardware e controle de atuadores que necessitam do uso de eletrônica de potência.

O presente trabalho visa estudar o tema e desenvolver um módulo de hardware microcontrolado para controle de cargas indutivas, especificamente motores DC, estes amplamente utilizados em sistemas robóticos e de automação. Foi baseado na plataforma de hardware e software livres do Arduino e explora as funcionalidades embarcadas no microcontrolador ATmega328, que possibilitam o controle lógico de um circuito de potência e comunicação com outros dispositivos. Como resultado, este módulo propicia uma camada de abstração nos projetos que envolvam controle de tais motores bem como tem a capacidade de adicionar modularidade e escalabilidade, monitoramento independente e tratamento descentralizado de condições de funcionamento.

O desenvolvimento do trabalho contou com três etapas: O projeto do *hardware*, com o desenvolvimento do diagrama lógico, diagrama esquemático do circuito e a montagem de um protótipo; O *software*, com o desenvolvimento de uma biblioteca básica para controle de motores DC, que além dos métodos de controle, conta com as funcionalidades de uma API que recebe comandos e devolve os resultados das execuções através de strings curtos e estruturados; A *integração*, onde IDE do Arduino é configurada para reconhecer a placa e a biblioteca como elementos integrantes.

Ao final do processo foi obtido um protótipo completamente funcional onde é possível produzir e gravar aplicações da mesma forma como é feito com as placas oficiais do Arduino. Cumpre todos os requisitos necessários no controle de potência e medição de corrente de dois motores DC independentes tendo a capacidade de atuar como módulo independente ou como dispositivo escravo, recebendo comandos remotamente via conexão serial.

Palavras-chave: Motor, Arduino, Robótica, Automação, Microcontrolador, UART, ADC, PWM, ISP, I2C, ATmega328, Ponte H.

Abstract

Nowadays, topics such internet of things, robotics, automation, drones, 3D printing are quite frequent in the technological environment and awake interest of a growing community of people who identify themselves with these technologies. Many of these people, even those who have some skills with programming languages and development environments have yet some technical difficulties when they need to work on projects involving hardware and the control of actuators that require the use of power electronics.

The present work aims to study and develop a microcontrolled hardware module intended to control inductive loads, specifically DC motors, which are widely used in robotic and automation systems. It was based on Arduino hardware and software platform and explores the ATmega328 microcontroller's embedded features usefull to perform the logical control of power electronics and communication among devices. As result this module should provide an abstraction layer to the projects that involve the control of such engines as well the ability to add modularity, scalability, independent monitoring and decentralized treatment of adverse operating conditions.

Work's development has done in three stages: The *hardware design*, done by producing circuit's logic diagram, schematic diagram and assemblying a prototype; The *software* stage, where was developed a basic library intended to control DC motors, on which besides the control methods, integrates the functionalities of an API that receives commands and returns the results through short and structured strings; The *integration*, where the Arduino IDE was configured to recognize the board and library as its integrant elements. After these stages, was obtained a fully functional prototype where is possible to produce and burn applications into it's flash memory in the same way as it is done with official Arduino boards, fulfills the necessary requirements to perform power control and current measurement over two independent DC motors and also has the hability to work as an independent module or a slave device being commanded remotely by serial connection.

Keywords: Motor, Arduino, Robotics, Automation, Microcontroller, UART, ADC, PWM, ISP, I2C, ATmega328, H Bridge.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

CI	Circuito Integrado
DC	Direct Current
IoT	Internet of Things
IDE	Integrated Development Environment
I2C	Inter-Integrated Circuit
I/O	Input/Output
SPI	Serial Peripheral Interface
ICSP	In-Circuit Serial Programming
MOSFET	Metal-Oxide-Semiconductor Field Effect Transistor
MCU	Micro Controller Unit
SoC	System on Chip
TTL	Transistor-Transistor Logic
ROM	Read Only Memory
ADC	Analog to Digital Convertor
DAC	Digital do Analog Convertor
SDA	Serial Data
SCL	Serial Clock
R/W	Read/Write
EEPROM	Electrically-Erasable Programmable Read-Only Memory
USART	Universal Serial Asynchronous Receiver/Transmitter
PWM	Pulse-Width Modulation
SMD	Surface Mounting Device
MSB	Most Significative Bit
LSB	Last Significative Bit
SPS	Samples Per Second
API	Application Programming Interface
DTR	Data Terminal Ready

Capítulo 1

Introdução

Até algum tempo atrás, os processos de projeto, prototipação e construção de dispositivos eletrônicos estavam restritos a serem executados poucas pessoas que dedicaram vários anos de estudos a fim de adquirir conhecimento técnico necessário afim de desenvolver tais atividades. Este processo costumava ser lento e custoso pois os componentes eram adicionados um a um e condicionados ao projeto. Ao se abrir um equipamento eletrônico complexo, como um *Video Cassete Recorder* (VCR) por exemplo, fabricado nos anos 1980, podem encontrar placas de circuito densamente povoadas por circuitos integrados e outros componentes eletrônicos. Ao se abrir as mesmas categorias de equipamentos eletrônicos fabricados nos últimos anos veremos, em contraste, alguns poucos circuitos integrados grandes e um pequeno grupo de outros componentes eletrônicos auxiliares. Ainda na década de 1980, alguns fabricantes começaram incorporar microprocessadores ao seus produtos a fim de diminuir a complexidade do circuito, o que proporcionou economia nos custos de produção, tornando seus produtos mais competitivos no mercado [34].

Os primeiros microprocessadores de 8 bits, como o Intel 8080 [38] ou o Zilog Z80 [39] apareceram na década de 1970 e trouxeram consigo uma revolução na indústria eletrônica. Engenheiros e projetistas perceberam que ao se introduzir um microprocessador em um equipamento eletrônico, além da simplificação do circuito e expandir suas funcionalidades, os custos de manutenção dos equipamentos cairiam significativamente, pois, ao invés de ter que tratar de um circuito complexo com centenas de componentes muitos problemas poderiam ser resolvidos com uma simples atualização de *software*. Na época, isso consistia na troca do chip que tinha a função da ROM (*Read Only Memory*), que continha o *firmware* do equipamento.

Mesmo que o microprocessadores tenham causado uma grande evolução na indústria eletrônica, ainda não eram considerados uma solução completa, pois, para serem realmente úteis, precisavam de uma variedade de CI's externos que davam suporte ao seu funcionamento, como memórias RAM (*Ramdom Access Memory*) e ROM (*Read Only Memory*), multiplexadores, demultiplexadores, buffers, bem como outros para permitir interação com o ambiente externo (conversores analógico-digital e digital-analógico, portas

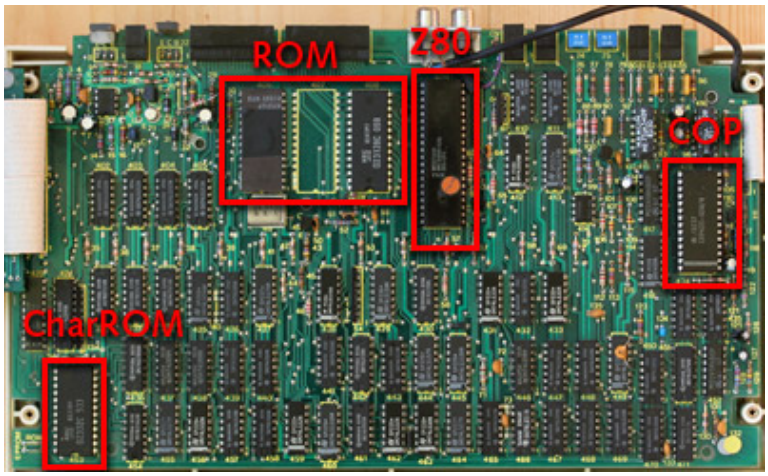


Figura 1.1: Placa principal do computador NewBrain. Fonte:[1]

seriais, timers, relógios de tempo real).

Na figura 1.1 pode ser vista a placa principal do computador *NewBrain* produzido pela empresa inglesa *Grundy Business Systems Ltd* no início da década de 1980 [37]. Pode-se perceber a complexidade do circuito devido ao fato microprocessador Z80 implementar basicamente a CPU, necessitando de uma variedade de hardware externo.

Por volta dos anos de 1990, a evolução dos processos de fabricação e miniaturização dos chips eletrônicos permitiu que fossem introduzidos nos microprocessadores cada vez mais circuitos adicionais, melhorando e ampliando suas funcionalidades, que antes eram exercidas por chips externos. Para diferenciar esses novos microprocessadores aumentados dos seus parentes mais simples, passaram a ser chamados de microcontroladores [8], ou MCU (*Micro Controller Unit*).

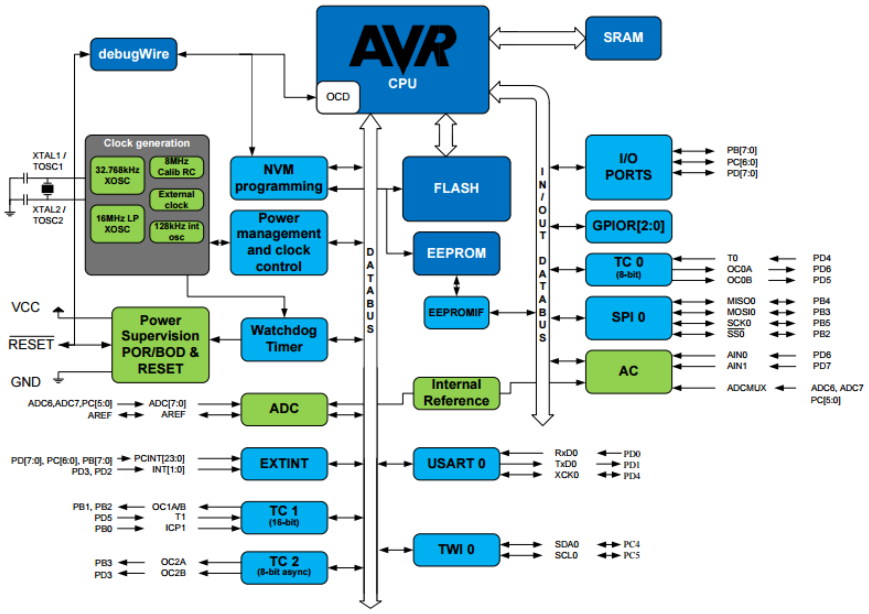


Figura 1.2: Diagrama interno do microcontrolador ATmega328. Fonte:[3]

A Figura 1.2 ilustra o diagrama interno de uma MCU moderna de 8 bits, o ATmega328, onde podem ser vistas suas funcionalidades embarcadas como, memória, gerador de *clock*, supervisor de alimentação, temporizadores e uma variedade de circuitos adicionais que proporcionam independência de diversos periféricos externos.

Mesmo com toda praticidade e flexibilidade que os microcontroladores trazem para o processo de projeto de *hardware*, a utilização dos recursos embarcados ainda demanda um conhecimento técnico em *hardware* e *software* em baixo nível considerável por parte dos projetistas, pois para acessar e tornar disponíveis tais recursos é necessário um estudo detalhado da arquitetura interna da MCU e através de *software* configurar registradores específicos para habilitar e configurar portas de I/O. Ainda, em muitos casos cada pino é capaz de desempenhar mais de uma função. Não existe um padrão universal para o acesso a tais recursos e configurações, estes diferem entre fabricantes e mesmo entre modelos distintos de MCUs. Para ter acesso a tais recursos da MCU e usá-los de forma adequada é preciso um estudo detalhado da documentação fornecida pelo fabricante, onde são descritas em detalhes todas as características do componente.

Em 2003, um estudante de mestrado chamado Hernando Barragán do *Interaction Design Institute Ivrea* (IDII) na Itália começou a desenvolver um trabalho que tinha por objetivo oferecer acesso facilitado à eletrônica para artistas, designers e para o público em geral que não possui conhecimentos avançados em eletrônica e programação. A ideia por trás deste trabalho foi a criação de um hardware base, genérico, baseado em microcontroladores de forma que se adapte às necessidades do projeto através de software e em conjunto uma IDE (*Integrated Development Environment*) que tem como função principal proporcionar abstração do *hardware* do microcontrolador, oferecendo acesso às suas funcionalidades através do uso de funções específicas, tornando o processo de prototipação de dispositivos eletrônicos mais simples e intuitivo. Esse projeto recebeu o nome de Wiring [6].

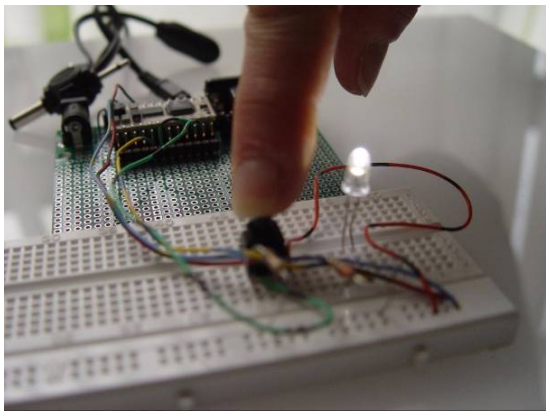


Figura 1.3: Primeiro protótipo de *hardware* do Wiring. Fonte:[6]

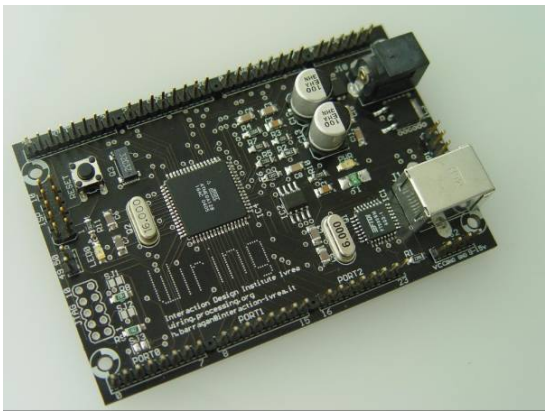


Figura 1.4: Placa do Wiring finalizada, modelo atual. Fonte:[6]

De acordo com o documento de autoria de Hernando Barragán intitulado *The Untold History of Arduino* [6] o Wiring foi um grande sucesso, sendo usado no próprio IDII como base no ensino da disciplina *physical computing*, que consiste na construção de protótipos eletrônicos, que através de sensores e atuadores, interagem com pessoas e ambientes à sua volta [5], sendo este integrante de projetos com os mais variados fins. De acordo com o mesmo documento, em 2005, Massimo Banzi, que havia sido orientador da tese de Barragán, em conjunto com David Mellis, estudante do IDII à época, fizeram um desenvolvimento paralelo da plataforma Wiring adicionando suporte ao microcontrolador ATmega8 e ao qual posteriormente foi dado o nome Arduino.

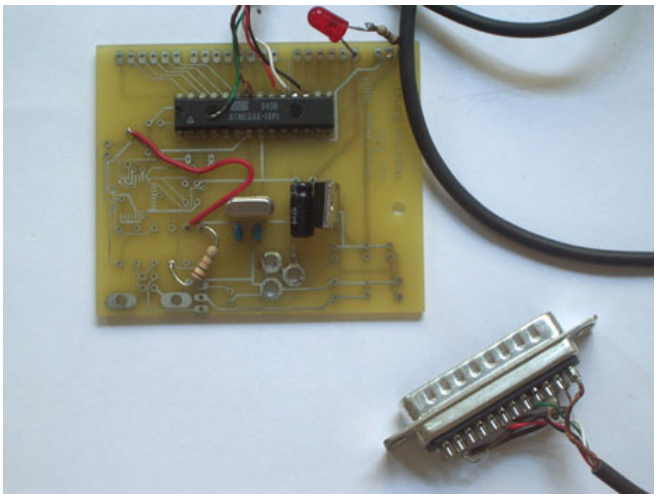


Figura 1.5: Primeiro protótipo de hardware do Arduino. Fonte:[31]

De acordo com uma matéria publicada na revista *IEEE Spectrum* de Outubro de 2011, intitulada *The Making of Arduino* [31], a decisão de criar uma plataforma nos mesmos moldes do Wiring com suporte a microcontroladores de 8 bits da linha ATmega e o desenvolvimento de uma IDE baseada na linguagem *Processing* [21] aconteceu com o intuito de ser uma plataforma mais barata, mais simples e ainda mais fácil de se usar, conforme Banzi. A formatação em produto final do *hardware* do Arduino continuou valorizando a simplicidade e funcionalidade, sendo adicionado suporte à conexão USB para interface direta com o computador, regulador de tensão e pouquíssimos outros componentes, o que contribuiu para manter o custo final baixo. O ponto forte da plataforma baseia-se na IDE multiplataforma (Windows, Linux e MacOS) que permite o acesso às funcionalidades do microcontrolador de forma simples e intuitiva, promovendo uma camada de abstração sobre a complexidade técnica exigida na utilização dos microcontroladores de forma nativa[5].

Devido à facilidade de uso, a plataforma rapidamente fez um enorme sucesso entre estudantes, mesmo entre aqueles com pouco ou nenhum conhecimento em programação e eletrônica [33]. Outro fator importante que contribuiu para o sucesso do Arduino foi o surgimento de uma comunidade bastante ativa, estimulada pela criação do fórum em `www.arduino.cc` onde pessoas do mundo todo ajudam-se a aprender sobre a plataforma e a

Wiki chamada *Playground* (`www.arduino.cc/playground`) onde é possível encontrar documentação de *hardware* e *software*, circuitos, códigos e tutoriais para os mais diversos fins [5].

Hoje, o Arduino conta com dezessete placas e treze *shields* oficiais [2] (placas acopláveis que agregam funcionalidades à placa principal) desenvolvidos para as mais diversas finalidades e uma grande quantidade de placas, *shields*, módulos, sensores, atuadores e bibliotecas não oficiais, pois as licenças de *hardware* e *software* livres do Arduino [5] permitem que os diagramas e circuitos sejam copiados, modificados, adaptados, bem como o *software* pode ser baixado, estudado, modificado, adaptado conforme conveniência, sem que haja necessidade de pagamento de taxas e licenças [15].

Este trabalho teve *hardware* e *software* baseados na plataforma Arduino, visa estudar e desenvolver um módulo microcontrolado de atuação independente com capacidade de controle autônomo ou remoto para atuadores que necessitam de eletrônica de potência, tendo como foco as características elétricas e os métodos de controle de motores DC. Foram realizados testes em condições controladas de carga para a validação do *hardware* e *software* desenvolvidos.

1.1 Motivação

A motivação na realização deste trabalho partiu da percepção de que muitas pessoas, que apesar de terem familiaridade com linguagens e ambientes de programação, mostravam dificuldades técnicas quando precisam trabalhar com hardware. Destes, mesmo aqueles que têm familiaridade com a plataforma Arduino experimentam dificuldades técnicas quando trabalham com projetos que necessitam de controle de motores. Mesmo que essas barreiras de conhecimento técnico não existam, o acionamento, controle e monitoração de motores DC consomem muitos recursos (memória, processamento e pinos de I/O) do microcontrolador, que tem a função de unidade de controle geral do sistema. A proposta desse trabalho baseia-se na premissa de que um módulo de controle de motores DC que possui o driver de controle e microcontrolador próprio com rotinas de controle pré-programadas e software específico possa impactar significativamente na diminuição do tempo de desenvolvimento, confiabilidade dos processos de controle e monitoramento do funcionamento de motores DC, bem como na diminuição da complexidade física e lógica da unidade de comando central. Pode vir a se tornar uma ferramenta muito útil para o público em geral que se disponha a construir sistemas de automação, robóticos, impressoras 3D e outras categorias de dispositivos, aos quais seja necessário o controle de cargas indutivas, em especial, motores DC.

1.2 Objetivos

Esse trabalho visa o desenvolvimento de um módulo microcontrolado para controle de cargas, com foco em motores DC, estes amplamente utilizados nas áreas de robótica e automação. Como resultado, este módulo deve propiciar uma camada de abstração nos projetos que envolvam controle de tais motores bem como tem a capacidade de adicionar modularidade e escalabilidade, monitoramento independente e tratamento descentralizado de condições adversas, como sobrecorrente, avanço sobre fim de curso, dentre outras que o projetista do sistema julgue necessário, melhorando a confiabilidade do processo e poupando recursos ou mesmo em alguns casos podendo eliminar a necessidade de uma unidade controladora central.

1.3 Metodologia

O desenvolvimento foi dividido em três partes:

Hardware: Desenvolvimento do diagrama lógico do circuito, diagrama esquemático e montagem de um protótipo;

Software: Desenvolvimento de uma biblioteca básica para controle de motores DC para testar as funcionalidades do protótipo;

Integração: Inclusão da placa de protótipo e da biblioteca desenvolvida na IDE do Arduino. Com a placa integrada é possível gravar o *bootloader* no microcontrolador ATmega328 usando um gravador AVR-ISP através das funcionalidades da IDE.

Cumpridas as etapas do desenvolvimento é possível produzir e gravar aplicações no protótipo da mesma forma como é feito com as placas oficiais do Arduino porém utilizando-se de um conversor USB-UART/TTL externo para a comunicação com o PC.
Por final são feitos testes com a biblioteca desenvolvida para validação das funcionalidades do hardware do protótipo montado.

1.4 Estrutura do trabalho

O trabalho está dividido em seis capítulos como descritos a seguir:

1. **Introdução:** O presente capítulo;

2. **Trabalhos Relacionados:** Apresentação de alguns trabalhos onde são utilizados sistemas microcontrolados para o controle de cargas indutivas.

3. **Fundamentação Técnica:** Principais componentes de hardware envolvidos e seus princípios de funcionamento: apresentação do AtMega328 e abordagem das funcionalidades de interesse embarcadas (UART, ISP, I2C, ADC, PWM), fundamentos do funcionamento de uma Ponte H e uma discussão acerca das vantagens de se usar um circuito integrado específico para o controle de motores DC.

4. **Desenvolvimento:** Aborda as escolhas de projeto no processo de controle, produção do diagrama esquemático, protótipo, processo gravação do software de base com o uso de um gravador AVR ISP, desenvolvimento de uma biblioteca específica para o hardware montado, integração da placa e da biblioteca na IDE do Arduino e a gravação do código no protótipo utilizando um conversor USB-UART/TTL.

5. **Testes e Resultados:** Experimentos e validação do sistema desenvolvido com a realização de testes e análise dos resultados obtidos.

6. **Conclusão:** Discussão sobre o trabalho, conclusões e demandas para trabalhos futuros.

Capítulo 2

Trabalhos Relacionados

Nos tempos atuais, onde IoT (*Internet of Things*), automação, robótica, drones, impressora 3D são termos de interesse bastante recorrentes no meio tecnológico, fica implícito que existe a necessidade do uso de atuadores para que dispositivos microprocessados e microcontrolados exerçam alguma ação sobre o ambiente. Algumas dessas tecnologias são impraticáveis sem a utilização de motores ou solenóides, que exigem requisitos de corrente e tensão elétricas que vão muito além da capacidades nativas de fornecimento de potência proporcionados por circuitos digitais, necessitando de um circuito complementar que faz parte de uma outra categoria de dispositivos eletrônicos, chamada eletrônica de potência.

Neste capítulo são citados alguns trabalhos onde são exploradas as funcionalidades embarcadas nos microcontroladores para o controle de cargas com o uso de eletrônica de potência para fins científicos ou didáticos. Todos os artigos foram obtidos a partir da plataforma *IEEEXplore* em `http://ieeexplore.ieee.org/Xplore/`.

No artigo *Full building of a sun tracker and control* [16] os autores constroem uma plataforma que tem por função fazer o acompanhamento do movimento do sol, servindo de instrumento didático prático no ensino de engenharia industrial.
Os movimentos verticais e horizontais são produzidos por dois motores DC com sistema de redução por engrenagens. Como sistema lógico de controle, os autores utilizaram uma placa Arduino Uno e para o acionamento dos motores um *driver* de ponte H [1] composto pelo CI L293D. O software utilizado foi produzido na IDE do Arduino, sendo responsável por ler e interpretar os dados dos sensores e controlar o posicionamento da plataforma acionando os motores, que têm suas velocidades controladas utilizando-se do método de modulação por largura de pulso (PWM) [2].

[1] Uma ponte H é um circuito eletrônico com quatro transístores dispostos em forma de H, muito utilizado no controle de cargas que necessitam de inversão de polaridade.
Este circuito é mostrado e discutido em detalhes na Seção 3.2.

[2] A modulação por largura de pulso, ou PWM (*Pulse Width Modulation*), é uma técnica utilizada para o controle de circuitos analógicos através de saídas digitais.
Este método é mostrado e discutido em detalhes na Seção 3.1.5.

No artigo *Obstacle Avoidance and Orientation Determination for a Remote Operated Vehicle* [35] os autores desenvolvem um robô guiado por um sistema de sonar que tem por objetivo desviar de obstáculos. Foi utilizado como base um microcontrolador PIC18F2431 [18] e como ambiente de programação a ferramenta proprietária da Microchip, a MPLab IDE e como drivers dos motores o CI LMD18200 [10].

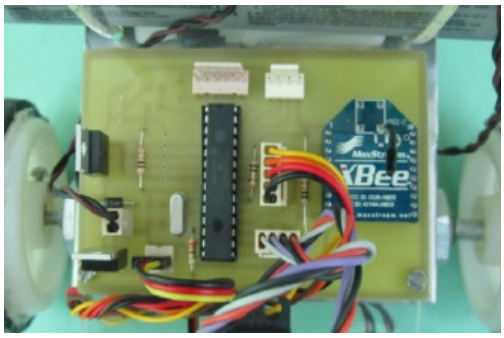


Figura 2.1: Placa controladora. Fonte:[35]

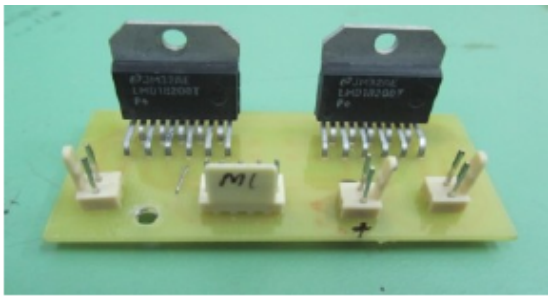


Figura 2.2: Drivers dos motores. Fonte:[35]

No artigo *Reuse of 3D printers for laboratory training System control and automation using microcontrollers and PLCs* [4] os autores usam uma estrutura mecânica de impressora 3D controlada por um PLC (*Programmable Logic Controller*) para promover a aplicação de conceitos teóricos para alunos de engenharia em diferentes ênfases. O sistema mecânico é composto de três motores de passo e como driver de controle os autores sugerem a possibilidade do uso do CI L293 [9], que tem as mesmas funções de controle do CI L298N [19], porém com características elétricas mais modestas. Como alternativa, sugerem o uso da placa PololuA4988 (que agrega o CI A4988 [20]) que é um driver desenvolvido especificamente para motores de passo que incorpora duas Pontes H e lógica de controle.

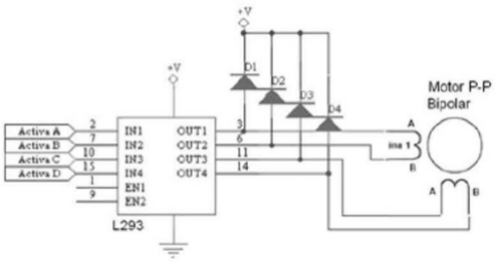


Figura 2.3: Esquema de ligação do CI L293 em motores de passo. Fonte:[4]

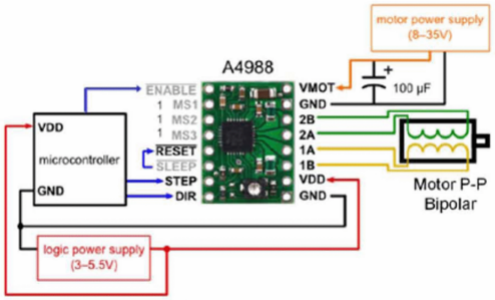


Figura 2.4: Placa Pololua4988 e esquema com conecções básicas. Fonte:[4]

No artigo *Development of Custom-made Engine Control Unit for a Research Engine* [23] os autores propõem um método para diminuir o erro na medida de combustível injetado através da medida da relutância do solenóide do bico injetor em um circuito de loop fechado com o controlador do bico injetor. Para realizar este trabalho os autores utilizaram um Arduino em conjunto com um circuito amplificador para medir a relutância do solenóide e o CI ULN2803A como driver para o acionamento controlado do solenóide.

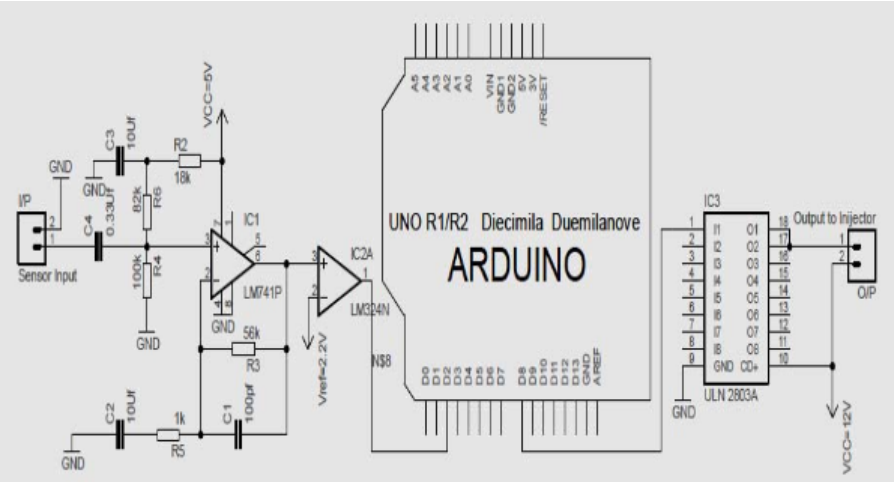


Figura 2.5: Diagrama do hardware da ECU (*Engine Control Unit.*) Fonte:[23]

No artigo *Using the Arduino Uno to teach digital control of power electronics* [13] os autores propõem o uso da plataforma do Arduino Uno como base no ensino dos conceitos que envolvem controle digital para eletrônica de potência. Eles exploram as capacidades do *hardware* embarcado no microcontrolador ATmega328 (interrupções, ADCs, timers geradores de PWM, comparadores analógicos) e modificam suas propriedades através de bibliotecas que manipulam diretamente os registradores internos, assim conseguem controlar as funcionalidades embarcadas de uma forma que não é possível com o uso das bibliotecas padrão do Arduino, melhorando o tempo de resposta dos ADCs, modificando a frequência dos PWMs e tendo acesso aos comparadores analógicos.

2.1 Considerações

Os artigos aqui citados serviram de base para a pesquisa da aplicabilidade e dos métodos de controle de carga por meio de microcontroladores. Arduino e ponte H são utilizados juntos em diversos casos.

No capítulo seguinte são vistos em detalhes os recursos de *hardware* embarcados no microcontrolador ATmega328 que viabilizam os controles autônomo e remoto de dispositivos de eletrônica de potência bem como o método de controle de cargas empregados neste trabalho.

Capítulo 3

Fundamentação Tecnológica

Este capítulo apresenta os principais componentes de *hardware* deste trabalho, explicando seus princípios de funcionamento, protocolos envolvidos e interfaces destacando a relevância e a justificativa de escolha dos mesmos.

3.1 Microcontrolador ATmega328

Hoje os microcontroladores podem ser encontrados em quase qualquer dispositivo eletrônico complexo, de dispositivos de música portáteis a máquinas de lavar roupa e até mesmo em carros. Eles são programáveis, baratos, pequenos, suportam bem situações adversas de temperatura, umidade, vibração e ruído elétrico, consomem pouca eletricidade, exigem poucos ou nenhum componente eletrônico adicional para sua utilização e são fabricados com as mais variadas configurações de hardware embarcado para atender as mais diversas necessidades. Essas características que os tornam tão úteis para a robótica, os permitem serem vistos como pequenos computadores dedicados que podem ser embutidos facilmente e que dão funcionalidades ao robô.

Diante de uma enorme variedade de microcontroladores no mercado, a escolha do ATmega328 aconteceu de forma natural devido às necessidades do projeto, características do componente e facilidades conforme listadas abaixo:

- Baixo custo (aprox. 2USD em lojas de comércio eletrônico internacional);
- Necessita de pouco hardware externo para seu funcionamento;
- Hardware integrado para comunicação com outros dispositivos (USART e I2C);
- Modulação por Largura de Pulso (PWM) implementada em hardware;
- Número suficiente de portas de I/O (conforme Tabela 3.1);
- Integrante de vários modelos de hardware do Arduino proporcionando:

- Possibilidade de integração de soluções de *hardware* e *software* do Arduino ao projeto graças à licença de hardware e software livres;
- Proporciona facilidade de programação e operação graças à popularidade da plataforma Arduino;
- Abstração de *hardware* do microcontrolador por software;
- Integração à IDE do Arduino via *bootloader*[1] específico;
- Farta documentação online de *software* e *hardware*;

A Atmel, fabricante do ATmega328, lista no Capítulo 2 do *datasheet* do microcontrolador [3] os principais recursos embarcados conforme Tabela 3.1, sendo que destes, os mais relevantes para este trabalho são:

USART (*Universal Serial Asynchronous Receiver/Transmitter*);

ADC (*Analog to Digital Convertor*);

I^2C (*Inter-Integrated Circuit*);

SPI (*Serial Peripheral Interface*);

Timer de 16 bits com a funcionalidade PWM (*Pulse-Width Modulation*) habilitada.

Estes recursos são discutidos em detalhes nas seções que seguem neste capítulo.

Tabela 3.1: Principais recursos de hardware do ATmega328 [3]

Recurso	**ATmega328**
Número de pinos	28
Memória Flash	32K Bytes
Memória RAM	2K Bytes
Memória EEPROM	1K Bytes
Pinos de I/O de propósito geral	23
Comunicação SPI	1
Two Wire Interface (I^2C)	1
Porta USART	1
Conversores Analógico-Digital (ADC)	8 (10bits - 15K SPS)
Número de ADC's	6
Contadores/Timers de 8bits	2
Contadores/Timers de 16bits	1

Os recursos embarcados ATmega328 são acessíveis fisicamente através de um conjunto de pinos da MCU. Estes são multiplexados (cada pino pode oferecer mais de uma função) necessitando que haja uma configuração prévia (*preset*) dos registradores internos correspondentes para que os pinos desempenhem as funções desejadas bem como as configurações de registradores específicos do recurso desejado.

[1] O *bootloader* é um pequeno trecho de código que é executado imediatamente após o microcontrolador ser energizado ou sofrer um processo de *reset*, sendo abordado com mais detalhes na Seção 4.4.

Para ilustrar, vamos supor que seja preciso configurar e realizar uma operação de escrita ou leitura em alguns dos pinos de I/O. Para isso será preciso realizar operações de escrita ou leitura em bits específicos de registradores fisicamente implementados na MCU. O ATmega328 possui 3 portas (com 7 ou 8 pinos cada) identificadas na Figura 3.4 por PB, PC e PD onde cada pino pode ser configurado e acessado individualmente. Cada uma dessas portas possui 3 registradores de 8 bits cada chamados DDRx, PINx e PORTx, onde 'x' corresponde a 'B', 'C' ou 'D' de acordo com a porta com a qual estamos querendo acessar (totalizando 9 registradores). O valor de um bit numa determinada posição de cada um desses registradores representa a configuração e a operação que se realiza sobre o pino correspondente.

Os registradores DDRx, PINx e PORTx têm suas funções descritas a seguir e as figuras 3.1, 3.2 e 3.3 ilustram respectivamente os registradores DDRD, PIND e PORTD do microcontrolador ATmega328.

DDRx: Define a direção dos dados, onde 0 define o pino correspondente como *entrada* e 1 como *saída*.

Name: DDRD
Offset: 0x2A
Reset: 0x00
Property: When addressing as I/O Register: address offset is 0x0A

Bit	7	6	5	4	3	2	1	0
	DDRD7	DDRD6	DDRD5	DDRD4	DDRD3	DDRD2	DDRD1	DDRD0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Figura 3.1: Registrador DDRD. Fonte:[3]

PINx: Ao ler este registrador se obtém o valor (digital) no pino correspondente.

Name: PIND
Offset: 0x29
Reset: N/A
Property: When addressing as I/O Register: address offset is 0x09

Bit	7	6	5	4	3	2	1	0
	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

Figura 3.2: Registrador PIND. Fonte:[3]

PORTx: Ao escrever neste registrador define-se o valor (digital) no pino correspondente.

Name: PORTD
Offset: 0x2B
Reset: 0x00
Property: When addressing as I/O Register: address offset is 0x0B

Bit	7	6	5	4	3	2	1	0
	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Figura 3.3: Registrador PORTD. Fonte:[3]

O Capítulo 18 do *datasheet* do ATmega328 [3] descreve com detalhes o *hardware* interno, os registradores, as configurações, operações e modos de funcionamento possíveis para os pinos de I/O e os Capítulos 5 e 6 descrevem com detalhes a funcionalidade de cada pino. Uma perspectiva simplificada dos pinos do microcontrolador e suas respectivas funções é mostrada na Figura 3.4, onde é possível ver as funcionalidades multiplexadas dos pinos por suas descrições individuais e os respectivos grupos funcionais pela legenda de cores.

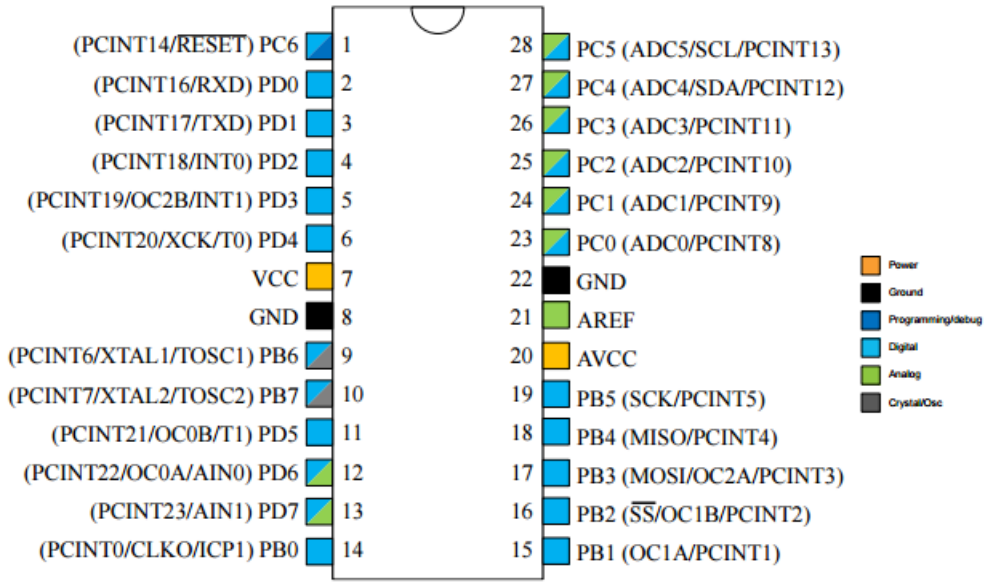


Figura 3.4: Pinagem e respectivas funcionalidades do ATmega328. Fonte:[3]

Uma das grandes vantagens de se usar o ATmega328 no projeto, conforme citado acima, é que a integração e o desenvolvimento do software utilizando-se da IDE do Arduino proporciona abstração sobre todos estes processos de leitura e escrita em hardware para se usufruir das funcionalidades embarcadas na MCU o que impacta de forma bastante significativa na diminuição do tempo do desenvolvimento do projeto.

Nas seções a seguir são apresentados os recursos de *hardware* do ATmega328 mais relevantes para o projeto: USART, ADC, I2C, SPI e PWM.

3.1.1 USART

USART, sigla para *Universal Synchronous/Asynchronous Receiver Transmitter* (receptor e transmissor síncrono/assíncrono universal) é um esquema de comunicação muito simples, implementado em *hardware* na maioria dos microcontroladores modernos e possibilita a comunicação ponto-a-ponto entre entre dispositivo e computador de uma forma simples e confiável. Apesar do ATmega328 suportar a comunicação síncrona, esta é pouco usada, pois traz a necessidade de transmitir não só informação mas também o *clock* de sincronismo, dessa forma cada bit de *clock* informa o receptor quando guardar o bit de informação. O processo de sincronismo aumenta a banda necessária para transmitir a informação ou necessita de um pino extra exclusivo para o *clock* como é o caso dos protocolos síncronos usados para comunicação via SPI e I^2C.
A comunicação no modo assíncrono torna o processo mais simples eliminando-se a necessidade da transmissão do *clock*, porém o sistema é dependente do tempo, dessa forma transmissor e receptor têm de estar em acordo com a velocidade de transmissão, tipo e tamanho do pacote de dados para a perfeita compreensão por parte do receptor como também é adicionado um *start bit* e um *stop bit* no pacote de dados para que o receptor assimile corretamente a informação [22]. A IDE do Arduino estabelece apenas o modo de comunicação assíncrono, portanto aqui será tratado apenas deste protocolo, sendo este referenciado por UART (sem o '*S*' referente ao '*Synchronous*' da sigla USART).

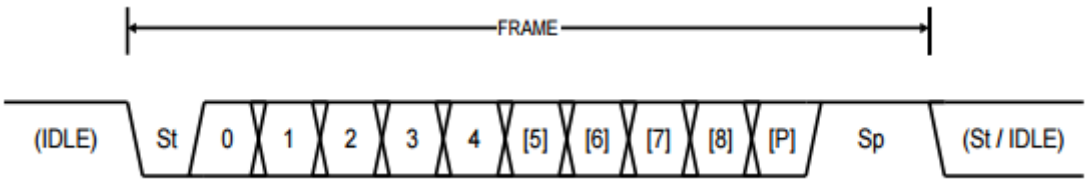


Figura 3.5: Representação de um pacote de dados do protocolo UART. Fonte:[3]

A Figura 3.5 mostra a representação dos bits de uma pacote de dados assíncrono do protocolo UART. A composição e função dos bits representados por *Start (St)*, *Data (0-8)*, *Parity (P)* e *Stop (Sp)* são explicados a seguir.

Parâmetros para a comunicação assíncrona

Para que a comunicação assíncrona ocorra de forma confiável, tanto o transmissor quanto o receptor devem possuir alguns parâmetros configurados de forma correta para evitar erros e perfeita compreensão dos dados pelo receptor, são estes:

Bits de dados *(0-8)*
Informa quantos bits do pacote são referentes aos dados, os demais são usados para controle e teste de integridade. Como pode ser visto na Figura 3.5 o tamanho dos

dados pode variar entre 5 e 9 bits. Esta característica é útil pois pode tornar o protocolo mais eficiente dependendo do tipo de dados que se pretende transmitir, por exemplo, ao se transmitir caracteres ASCII de 7 bits, cada pacote pode ser responsável pelo transporte de um caractere.
Outra configuração importante referente aos bits de dados é a sequência com a qual eles serão transferidos, ou seja, se será transferido primeiro o bit menos significativo ou o mais significativo. Quando não há referência a essa configuração assume-se que o primeito bit refere-se ao menos significativo e o último ao mais significativo.

Bits de sincronismo *(St e Sp)*
São dois ou três bits de controle que marcam o início e o final do pacote, chamados respectivamente de *start bit* e *stop bit*.
O *start bit* é caracterizado pela transição da linha de 0 para 1 quando esta encontra-se em estado de espera (*idle*).
Os *stop bits* podem ser configurados como sendo um ou dois bits (comumente utiliza-se apenas um bit) e são caracterizados pela transição da linha para estado de espera, permanecendo em nível lógico 1.

Bit de paridade *(P)*
O uso do bit de paridade é uma forma muito simples de verificação de erro. É adicionado um bit na sequência dos dados de forma a complementar a paridade pré-estabelecida. Por exemplo, supondo que o sistema esteja configurado com *paridade par* e o transmissor envie o byte *11100101*, que contém cinco bits 1, então o transmissor adiciona 1 como bit de paridade. O receptor conta o número de 1's dos dados mais paridade e se for par entende que a transmissão ocorreu sem erros.

Baud Rate
Especifica a velocidade com a qual os dados são transmitidos, normalmente expresso em *bps* (*bits per second*). O *Baud Rate* determina por quanto tempo o transmissor mantém a linha em níveis lógicos 1 ou 0 ou de quanto em quanto tempo o receptor deve fazer a leitura da linha de transmissão. Por exemplo, se a transferência ocorre a 9600 bps significa que cada bit está sendo transferido e recebido em intervalos de $1/9600 = 104\mu$s. A determinação do valor do *Baud Rate* não segue uma regra rígida, está mais relacionado com a capacidade do hardware e da linha de transmissão, mas existem alguns valores são convencionados como padrão, sendo estes: 1200, 2400, 4800, 9600 19200, 38400, 57600, e 115200 bps.

Conexão física e modos de comunicação

O protocolo UART foi concebido para estabelecer comunicação serial ponto-a-ponto, portanto os dispositivos não necessitam de endereçamento, apenas que seus terminais estejam conectados e referenciados de forma correta. O terminal responsável pela transmissão dos dados é normalmente referenciado por T_x e o responsável pela recepção por R_x, desta forma a referência de transmissão e recepção devem ser conectados de forma invertidas entre os dispositivos como ilustrado na Figura 3.6.

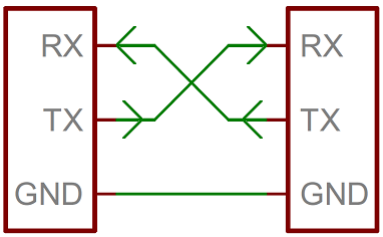


Figura 3.6: Conexão física entre dois dispositivos UART. Fonte:[29]

Além das conexões dos terminais T_x e R_x é importante a conexão interligando o *terra* (GND) dos dispositivos para que haja uma referência elétrica comum para os níveis de tensão que representam os níveis lógicos 1 e 0.

Esta configuração de conexão cruzada e a implementação de *hardware* da UART permitem que os dispositivos se comuniquem de três modos distintos referentes a ordem e sincronia de transmissão e recepção. São estes:

Full-Duplex: Neste modo os dois dispositivos são capazes de transmitir e receber dados simultaneamente.

Half-Duplex: Os dois dispositivos transmitem e recebem dados, no entanto não o fazem ao mesmo tempo, os dispositivos se revezam nestas tarefas.

Simplex: Neste modo apenas um dispositivo transmite dados e outro apenas recebe.

Comunicação serial e o padrão RS-232

Muitas vezes encontramos material técnico referindo-se ao termo *RS-232* como sinônimo de comunicação serial assíncrona, deixando implícito que a *RS-232* determina o protocolo de comunicação, neste ponto há alguma confusão sobre o que a EIA (*Eletronics Industries Alliance*) padronizou no *RS-232*. Este padrão apenas especifica características elétricas dos circuitos e a numeração dos pinos. Outras características como o conector em forma de "D", o uso de código ASCII, formato dos dados e comunicação assíncrona não são parte do *RS-232*, a palavra "padrão", porém, é utilizada geralmente quando todos estas características aparecem juntas, de modo a tornarem-se efetivamente obrigatórias. Quando microcontroladores e outros CIs se comunicam de forma serial normalmente

utilizam níveis de tensão TTL (*Transistor-Transistor Logic*). Os níveis de tensão para representar os níveis lógicos são determinados pela tensão de alimentação, tipicamente 3.3V ou 5V conforme ilustrado na Figura 3.7.

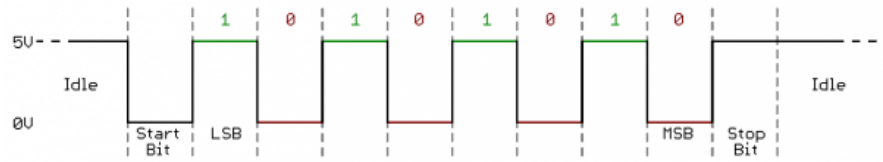


Figura 3.7: Pacote de comunicação serial com níveis de tensão TTL. Fonte:[29]

Computadores que ainda possuem portas de comunicação serial, apesar de utilizarem o mesmo protocolo, implementam o padrão *RS-232* que estabelece níveis de tensão muito superiores aos níveis TTL e inclusive níveis de tensão negativos para representar o nível lógico 0, tipicamente +/-12V conforme ilustrado na Figura 3.8.

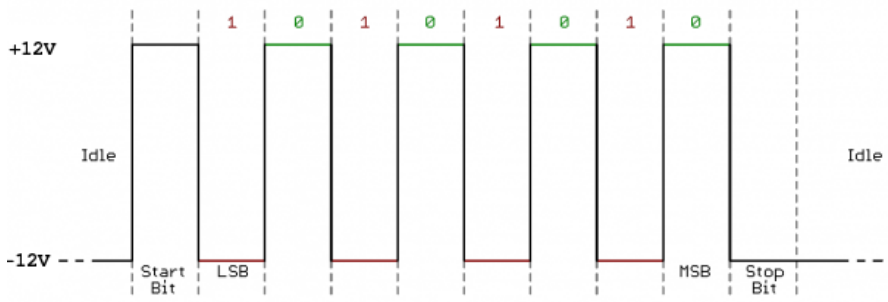


Figura 3.8: Pacote de comunicação serial com níveis de tensão RS-232. Adaptado de:[29]

Essa característica de variação elétrica foi idealizada para estabelecer comunicação serial mantendo a integridade dos dados por cerca de uma dezena de metros, como também é uma maneira simples de detectar falha na linha de transmissão, já que um cabo rompido deve apresentar nível de tensão de 0V no receptor [22].

Devido a grande diferença de níveis de tensão entre TTL e *RS-232*, conectar a UART de um microcontrolador diretamente no porta serial do computador muito provavelmente causará danos irreversíveis ao microcontrolador, devendo ser feita a tradução apropriada dos níveis de tensão com o auxílio de circuitos apropriados, como por exemplo utilizando CIs especializados para essa função, como é o caso do MAX232 [11].

USART no Motorino

A interface USART é de grande importância para o sistema proposto e desenvolvido neste trabalho, pois é através dela que é feita a comunicação com o computador de forma a carregar o código compilado através da IDE do Arduino. Também é possível enviar e receber dados através da função *Serial Monitor* da IDE, bastante útil para enviar dados e principalmente ler dados de processamento de forma a executar uma espécie de *debug* da

execução. Como é cada vez mais raro os computadores modernos possuírem porta serial, a comunicação é feita pela porta USB do computador intermediada por um conversor USB-UART/TTL.
Mais detalhes sobre o conversor USB-UART/TTL utilizado nesse trabalho podem ser encontrados na Seção 4.4. Todos os detalhes sobre a interface USART do ATmega328 podem ser encontradas no Capítulo 24 do *datasheet* [3].

3.1.2 Serial Peripheral Interface - SPI

O SPI (*Serial Peripheral Interface*) é outra forma de comunicação serial embarcada no ATmega328. É um barramento de comunicação síncrona, o que significa que o transmissor e receptor envolvidos na comunicação via SPI devem compartilhar a mesma linha de *clock* para a sincronização da detecção dos bits no receptor. Normalmente o SPI é utilizado para comunicações em distâncias bastante curtas, como por exemplo entre microcontroladores numa mesma placa de circuito ou em placas próximas, diferente da USART que tem por objetivo a comunicação entre circuitos mais distantes, usualmente entre um computador e o microcontrolador. O SPI foi desenvolvido para comunicação a altas velocidades em curtas distâncias com um mínimo de pinos do microcontrolador [22].

A comunicação via SPI envolve dispositivos configurados como *master* e *slave* e ambos enviam e recebem dados simultaneamente porém o *master* é responsável por fornecer o sinal de *clock* para a transferência, deste modo, o *master* controla a velocidade da transferência no barramento [22].

Um dos motivos pelo qual a comunicação via SPI é popular em microcontroladores é a simplicidade da implementação do hardware, baseada na transferência de bits entre *shift registers* (registradores de deslocamento). Para ilustrar, a Figura 3.9 mostra o diagrama lógico do CI 74164 [24] que compõe um *shift-register* de 8 bits.

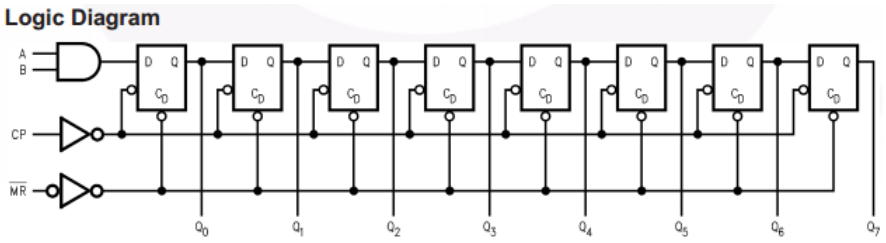


Figura 3.9: Diagrama lógico de um *shift-register* de 8 bits. Fonte:[24]

Na Figura 3.9, as conexões A e B representam a entrada de dados, *Q0-Q7* as saídas dos dados, CP a entrada do *clock* e $\overline{MR}$ faz o *reset* de todas as saídas. Note que a saída de cada um dos *flip-flops* [17] está ligada na entrada do seguinte, isto permite que a cada ciclo do *clock* os bits se desloquem da entrada até *Q7*, desta forma, após o oitavo ciclo de *clock* teremos um byte completo carregado de forma serial em *Q0* a *Q7*.

Interconexão entre dispositivos

Os dispositivos envolvidos na comunicação SPI podem trabalhar dois a dois sendo que um opera em modo *master* e outro em modo *slave*, ou em barramento contendo dois ou mais dispositivos. A Figura 3.10 mostra a interconexão entre dispositivos *master* e *slave*.

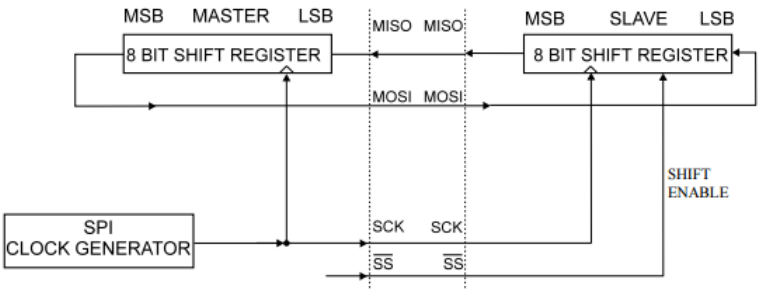


Figura 3.10: Interconexão SPI entre *master* e *slave*. Fonte:[3]

O ATmega328 possui quatro conexões envolvidas na configuração e comunicação SPI, conforme pode ser visto na Figura 3.11.

MISO (*Master-In-Slave-Out*) Se o dispositivo estiver configurado como *master* este pino representa a entrada de dados do seu *shift register* interno, e se configurado como *slave*, a saída de dados de forma serial.

MOSI (*Master-Out-Slave-In*) Se o dispositivo estiver configurado como *master* este pino representa a saída de dados do seu *shift register* interno, e se configurado como *slave*, a entrada de dados de forma serial.

SCK (*Serial Clock*) Se o dispositivo estiver configurado como *master*, fornece o *clock* de sincronismo da transmissão, e se configurado como *slave*, recebe o *clock* de sincronismo.

SS (*Slave Select*) Se o dispositivo estiver configurado como *master* e este pino estiver configurado como saída, não interferirá no funcionamento do SPI e se configurando como pino de entrada, deve ser mantido em nível lógico 1 para que ocorra a transferência. Se o dispositivo estiver configurado como *slave*, este pino deve atuar como entrada e habilita o funcionamento da SPI quando em nível lógico 0.

Barramento SPI

Se conectado em forma de barramento, o SPI admite que um dispositivo *master* se comunique com diversos dispositivos *slaves*, no entanto a comunicação acontece somente aos pares necessitando que somente um dispositivo *slave* receba em seu pino SS o nível lógico 0 para habilitar a recepção dos dados. A Figura 3.11 ilustra a conexão dos dispositivos em barramento.

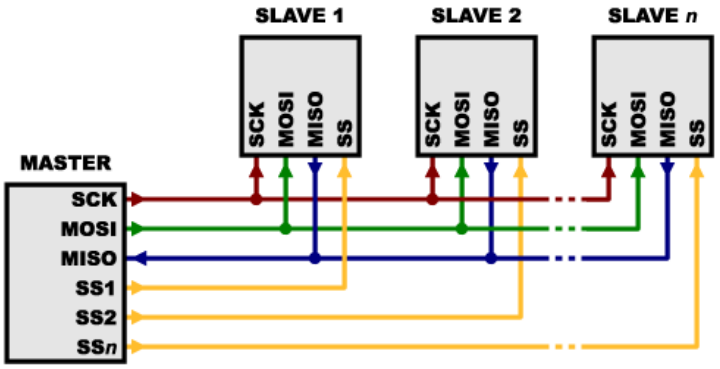


Figura 3.11: Barramento SPI com múltiplos *slaves*. Fonte:[30]

A desvantagem desse tipo de conexão é que necessita de n linhas de habilitação para n dispositivos configurados como *slave* através do pino *SS*.

Dispositivos SPI em cascata

Uma outra forma de conectar os dispositivos é em cascata, também chamado de conexão *daisy chain*. A Figura 3.12 ilustra a ligação feita em modo cascata.

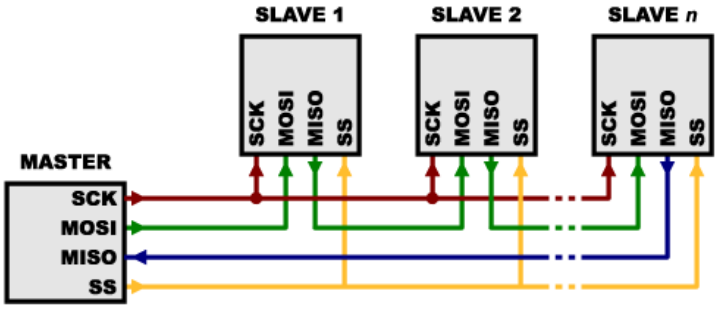


Figura 3.12: Dispositivos SPI em cascata. Fonte:[30]

Note que as conexões MOSI e MISO são conectadas em série. Na prática estamos conectando os *shift-registers* dos dispositivos em série. A vantagem desse tipo de conexão em relação ao barramento é que não é mais necessário uma conexão de habilitação exclusiva para cada dispositivo *slave*, porém tem como desvantagem que será preciso muito mais pulsos de *clock* para que a informação se propague entre os dispositivos. Também não é possível alterar o conteúdo de apenas um dispositivo.

Porta SPI no Motorino

No protótipo do Motorino está previsto um conector para o barramento SPI para que se possa utilizar esta interface conforme descrito acima. O uso do SPI é facilitado pela biblioteca *SPI* desenvolvida para o Arduino (`https://www.arduino.cc/en/Reference/SPI`).
Um outro uso dessa interface é que através dela é possível carregar diretamente um arquivo binário na memória do ATmega328, sem o uso do *bootloader* mas com o auxílio de uma gravador do tipo *AVR Programmer* por um método chamado ICSP (*In-Circuit Serial Programming*), que é um método desenvolvido por fabricantes para que seja possível gravar na memória do microcontrolador mesmo depois de soldado na placa de circuito impresso. Mais detalhes sobre a gravação do *bootloader* do Arduino via SPI utilizando um gravador tipo *AVR Programmer* será mostrado na Seção 4.4. Todos os detalhes sobre a interface SPI no ATmega328 podem ser encontrados no Capítulo 23 do *datasheet* [3].

3.1.3 Two Wire Interface - I2C

O I^2C, sigla para *Inter-Integrated Circuit*, também conhecido como "*two-wire bus*" (barramento de dois fios) é uma interface serial criada pela *Phillips Semiconductors* (hoje, *NXP Semiconductors*) para fazer a comunicação entre seus CIs, placas e dispositivos. Desde 10 de Outubro de 2006 não são cobradas licenças para implementação do protocolo I^2C, no entanto ainda existem licenças para se usar alguns endereços para *slaves* alocados pela NXP [26]. Desde então, vários fabricantes se utilizam deste protocolo para fazer comunicação entre seus dispositivos sendo estes dos mais variados tipos, como EEPROMs (*Electrically-Erasable Programmable Read-Only Memory*), RTCs, sensores de temperatura, displays etc. Alguns microcontroladores, como o ATmega328 implementam esta interface em *hardware* que desempenham as tarefas de baixo nível necessárias para estabelecer a comunicação, embora seja também possível a implementação em *software* em microcontroladores que não possuem hardware específico [36].

Barramento

Num barramento I^2C temos duas categorias de dispositivos:

- *Master*: Nó que gera o clock e inicializa a comunicação com os *slaves*.
- *Slave*: Nó que recebe o clock e responde quando endereçado pelo *master*.

Apesar da definição que o nó com a função de *Master* é responsável por inicializar a comunicação, os nós configurados como *Slaves* também têm a habilidade de inicializar a comunicação com o nó *Master* desde que este esteja configurado para esperar pelas transmissões e receber os dados dos *Slaves*.

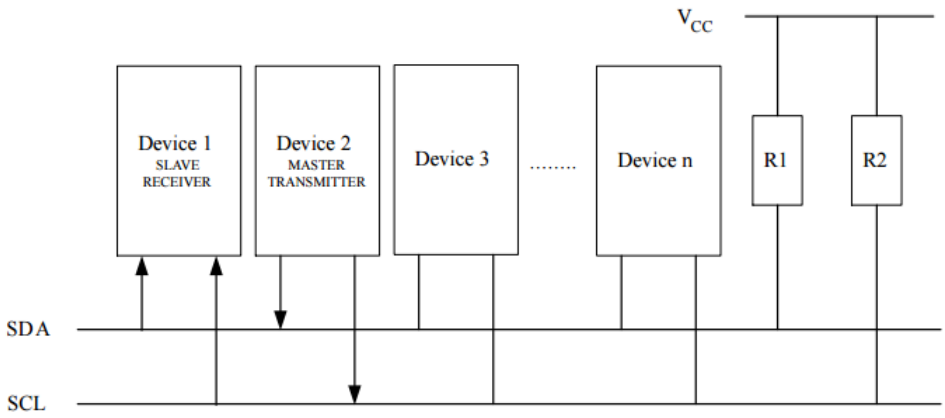


Figura 3.13: Típico barramento I^2C. Fonte:[3]

O barramento I^2C também é chamado de *barramento de dois fios* devido ao fato de que os terminais denominados SDA (*Serial Data*) e SCL (*Serial Clock*) serem responsáveis pela sincronização e transmissão dos dados, no entanto, para que haja referência elétrica aos sinais lógicos é necessário que exista uma interligação com o terra de todos os dispositivos do barramento, e em alguns casos é necessário uma ligação do positivo com os terminais SDA e SCL via resistores de *pull-up*, representados na Figura 3.13 como R_1 e R_2 para que se evite a degeneração do sinal devido a atenuação pela resistência elétrica dos cabos e vias do barramento, ou garantir o funcionamento caso os dispositivos conectados tenham as linhas SCL e SDA com coletor aberto.

Endereçamento

Cada dispositivo no barramento possui um endereço distinto que o identifica. Este endereço é composto por 7 bits, o que proporciona um limite teórico de 128 dispositivos. Alguns fabricantes determinam endereçamento de 8 bits para seus dispositivos, o que gera uma certa confusão, pois o bit menos significativo tem a função de *flag* que determina se a operação que está sendo realizada é de leitura ou escrita no barramento.

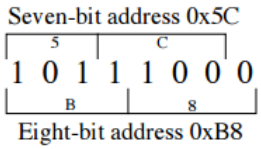


Figura 3.14: Enderaçamento I^2C de 7 e 8 bits. Fonte:[36]

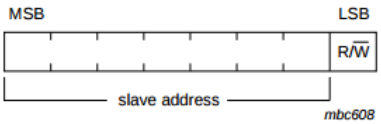


Figura 3.15: Frame do byte de endereçamento. Fonte:[25]

No exemplo da Figura 3.14 o dispositivo tem um endereço de 7 bits de 0x5C mais o *flag* no dígito menos significativo do byte de endereçamento, onde 1 significa uma operação de leitura no barramento e 0 operação de escrita. Se for considerado o endereçamento contendo 8 bits, o mesmo dispositivo terá endereços distintos para as operações de escrita e leitura no barramento.

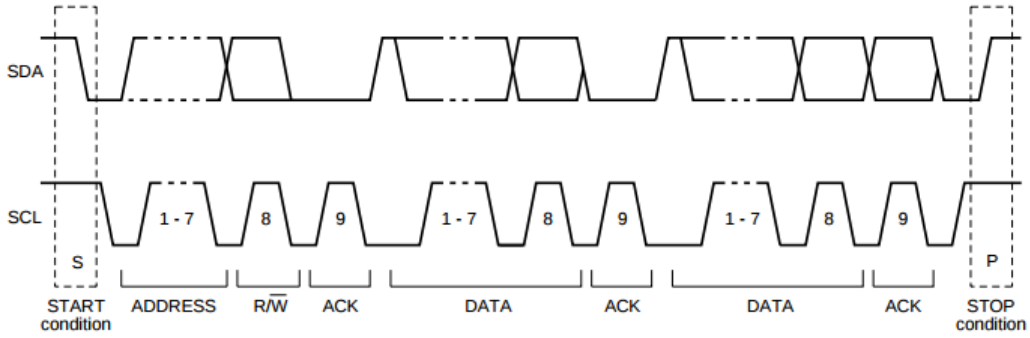


Figura 3.16: Frames de comunicação do protocolo I^2C [25]

As mensagens são divididas em dois tipos de *frames*: O primeiro corresponde ao endereçamento, onde o *master* determina para qual *slave* a mensagem se destina e em sequência, um ou mais frames com a mensagem no formato de pacotes de 8 bits cada. Os dados são transmitidos através da linha SDA depois que a linha SCL é colocada em nível lógico zero e o byte é registrado pelo receptor após a linha SCL ser colocada em nível lógico 1. O tempo de *clock* entre as operações de escrita e leitura é definida pelos dispositivos no barramento e podem variar de acordo com os CIs envolvidos.

Barramento I^2C no Motorino

No protótipo do Motorino está previsto um conector para o barramento I^2C com o propósito de facilitar a conexão e o gerenciamento de vários dispositivos através deste barramento. Essa ação contribui para a diminuição da complexidade das conexões de controle entre dispositivos e proporciona modularidade ao projeto do qual o Motorino é parte integrante. A utilização deste barramento é facilitada através da biblioteca *Wire* desenvolvida para o Arduino (`https://www.arduino.cc/en/Reference/Wire`). Detalhes sobre as conexões utilizadas para o barramento I^2C no Motorino podem ser encontrados na Seção 4.1. Todos os detalhes da implementação de hardware do I^2C no ATmega328, chamado na documentação de TWI (*Two Wire Interface*) podem ser encontrados no Capítulo 26 do *datasheet*.

3.1.4 Conversor Analógico-Digital - ADC

Para poder adquirir valores de grandezas do mundo externo, sistemas microcontrolados utilizam-se de sensores, que são dispositivos que convertem uma grandeza física específica em uma variação determinada de sinais elétricos pré-determinadas, ou seja, traduzem uma grandeza física, por exemplo, luminosidade, intensidade sonora, concentração de gás carbônico, etc, em sinais que possuem uma variação elétrica proporcional à intensidade da grandeza medida. Em princípio, um sensor quando alimentado por 5V pode proporcionar com qualquer valor que varie entre 0V e 5V. Neste caso o microcontrolador

precisa transformar esse valor de tensão elétrica num valor binário, proporcional, de forma que possa tomar alguma decisão. Para isso se utiliza de um *Conversor Analógico-Digital*, do inglês, ADC (*Analog to Digital Converter*).

ADC no ATmega328

As principais características do ADC do ATmega328 podem ser vistas na Tabela 3.2 e todos os detalhes de implementação e funcionamento podem ser encontrados no Capítulo 28 (*ADC - Analog to Digital Converter*) do *Datasheet*.

Tabela 3.2: Principais características dos ADCs do ATmega328 [3]

Número de ADCs	6 acessíveis individualmente
Resolução	10bits
Frequência de amostragem	15 mil amostras por segundo
Método de conversão	Aproximações sucessivas
Tensão de referência externa	0V a Vcc
Tensão de referência interna	1.1V

Resolução de um ADC

A resolução de um ADC é a quantidade de sinal analógico necessária para incrementar o valor digital convertido de uma unidade. Um ADC pode quantificar o sinal analógico em 2^n-1 partes, no caso da resolução em 10 bits do ATmega328 significa que o valor mínimo que pode representar é uma parte em 1023 da tensão usada como referência, (também chamada de tensão de fundo de escala). Por exemplo, usando a tensão de referência interna de 1.1V, conforme a tensão analógica de entrada aumenta em aproximadamente 1.07mV o contador do ADC incrementa de uma unidade até o limite de 1023 [17] [3].

Método de conversão

Existem vários tipos de circuitos que implementam um ADC, e apesar de todos fazerem a mesma função, cada tipo de circuito foi pensado com um propósito, seja diminuir erros de conversão, simplicidade do circuito, diminuir o tempo de conversão, dentre outros [17]. O tipo de ADC implementado no ATmega328 utiliza o método de *aproximações sucessivas* [3], que foi projetado com o propósito de proporcionar um tempo de conversão reduzido se comparado a outros métodos.

No algoritmo de transformação deste método, um gerador binário constrói o valor do resultado da conversão partindo do bit mais significativo, MSB (*Most Significative Bit*) em direção ao bit menos significativo, LSB (*Less Significative Bit*). Este número gerado entra num conversor *digital para analógico* e o nível de tensão analógico gerado é comparado com a entrada. Se o valor da entrada for maior ou igual, o bit testado é mantido como 1,

se for menor este bit se torna 0 e a comparação passa a ser feita com o próximo bit, assim sucessivamente até serem cobertos todos os bits da resolução do ADC. Após a comparação com o último bit, o valor gerado é colocado na saída do conversor [17].
A Figura 3.17 ilustra o processo de conversão de um ADC de 4 bits onde a conversão resulta no valor binário *1001*. Cada bifurcação representa uma decisão, se o valor convertido é maior ou não que o valor de entrada.

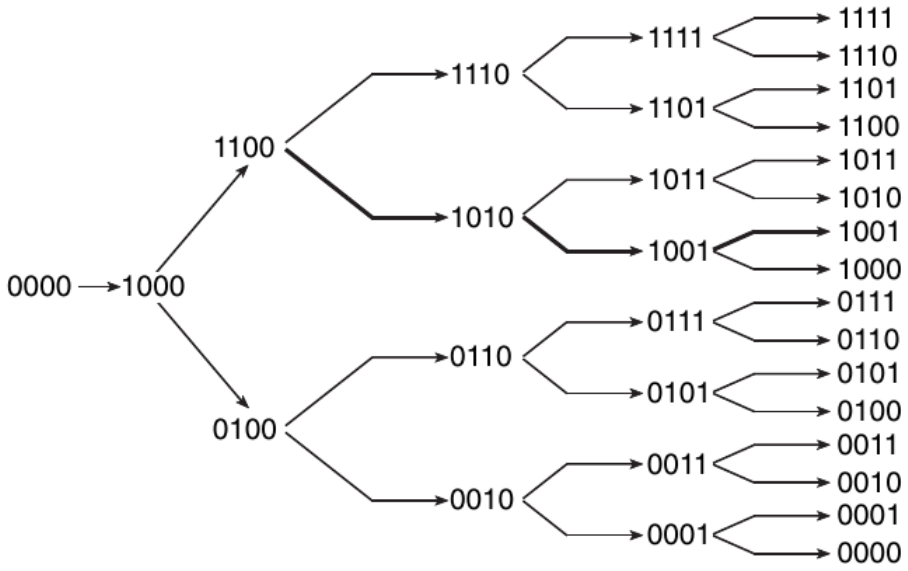


Figura 3.17: Processo de conversão de um ADC por aproximações sucessivas. Fonte:[17]

Frequência de amostragem

A frequência de amostragem de um ADC é o número de conversões realizadas por unidade de tempo. Usualmente é expressado pela sigla *SPS* que significa *Samples per Second* ou amostras por segundo. Em circuitos síncronos esta característica está diretamente relacionada ao número de ciclos de *clock* necessários para se realizar uma conversão. Pelo exemplo da Figura 3.17, é fácil perceber que num ADC por aproximações sucessivas de n bits o tempo de uma conversão é o tempo de n ciclos de *clock*, portanto teria uma frequência de amostragem de $Freq_{clock}(Hz)/Resolução(bits)$ SPS.
O ADC no ATmega328, apesar de usar o método de aproximações sucessivas tem uma implementação mais complexa do que este exemplo ilustrativo, portanto o tempo de conversão é um pouco maior que dez ciclos de *clock*.

Uso do ADC do ATmega328 no Motorino

O Motorino ocupa de forma exclusiva dois dos seis ADCs disponíveis no microcontrolador. Estes serão utilizados para a medição das correntes que circulam pelas duas pontes H do circuito. Os detalhes da utilização destes ADCs são descritos na Seção 4.1. Todos os detalhes da implementação do ADC no ATmega328 podem ser encontrados no Capítulo 28 do *datasheet* [3].

3.1.5 Modulação por Largura de Pulso - PWM

PWM (*Pulse Width Modulation*) ou Modulação por Largura de Pulso é uma técnica utilizada para o controle de circuitos analógicos através de saídas digitais.
O método consiste na variação do *duty cycle*, proporção do ciclo em que uma onda quadrada permanece em nível alto também chamado de *ciclo ativo* da onda, de forma que o valor médio da tensão da onda corresponda a um valor DC desejado. A forma de onda é então filtrada e usada para controlar dispositivos analógicos, criando assim um *DAC* (Conversor Digital para Analógico) [22].
A Figura 3.18 ilustra a determinação do ciclo ativo de uma onda quadrada e a Figura 3.19 a tensão média de uma onda quadrada do método PWM.

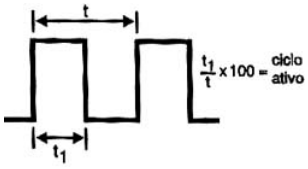


Figura 3.18: Ciclo ativo em uma onda quadrada. Fonte:[7]

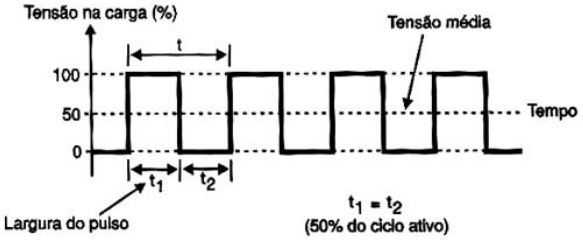


Figura 3.19: Tensão média em uma onda quadrada. Fonte:[7]

O PWM, apesar da possibilidade de ser gerado em *software*, é usualmente implementado nos microcontroladores em seu *hardware* como uma funcionalidade inerente de seus *timers/counters* (temporizadores/contadores) internos. Tal estratégia traz algumas vantagens como por exemplo, a independência do código em execução no microcontrolador e a utilização de poucas linhas de código necessárias para a configuração e habilitação do seu funcionamento.

Temporizadores do ATmega328

O Atmega328 possui três temporizadores (*timers*), sendo dois de 8 bits (TC0 e TC2) e um de 16 bits (TC1). Esses temporizadores são importantes para diversas funcionalidades, tais como [3]:

- Temporização;
- Contagem de eventos externos;
- Geração de sinais PWM;
- Interrupções periódicas;
- Medida de intervalos de pulsos;
- Gerador de frequência.

Os *timers* possuem características próprias de configuração e modos de funcionamento porém todos possuem a habilidade de gerar sinais PWM, sendo que cada um deles pode atuar em dois pinos distintos do microcontrolador perfazendo seis saídas PWM que devidamente configurados trabalham de forma independente através de um conjunto de registradores dedicados. Os registradores TCCRnA e TCCRnB (*Timer/Counter Control Registers*) agregam vários grupos de bits que são responsáveis pelas configurações principais dos *timers*, sendo estes [3]:

- WGM (*Waveform Generation Mode*): Modo de funcionamento do *timer*;
- CS (*Clock Select*): Controlam o *prescaler* (divisor) do sinal de clock;
- COMnA (*Compare Output Match A Mode*): habilita/desabilita/inverte a saída A;
- COMnB (*Compare Output Match B Mode*): habilita/desabilita/inverte a saída B;
- OCRnA e OCRnB (*Output Compare Registers*): valor de referência para o contador do *timer*. Quando o contador do *timer* atinge o valor guardado nestes registradores a saída é alterada conforme o modo de funcionamento selecionado. Os termos 'A' e 'B' referem-se as duas saídas (pinos do microcontrolador) regidos pelo modo PWM do referente *timer*.

Os *timers* do ATmega328 quando configurados como PWM podem trabalhar em dois modos distintos, chamados na documentação de *Fast PWM* (PWM rápido) e *Phase-Correct PWM* (PWM com fase corrigida).

Fast PWM: Neste modo de funcionamento, o *timer* conta repetidamente de 0 a 255. A saída assume nível lógico um ao início da contagem e alterna para o nível lógico zero quando o valor do contador atinge o valor armazenado respectivo registrador OCRnA ou OCRnB. Ao ocorrer o *overflow* no contador a contagem e o ciclo é reiniciado.
Tomando como exemplo a saída A, a frequência do PWM neste modo de operação conforme documentação do microcontrolador, é dada por:

$$f_{OCnA} = \frac{F_{clock}}{N \times (1 + Val_{max})} \tag{3.1}$$

Onde:

f_{OCnA} frequência do PWM para a saída A;
F_{clock} frequência de operação do microcontrolador;
N valor do divisor do *prescaler* (1, 8, 64, 256, ou 1024);
Val_{max} valor máximo atingido pelo contador (255 para 8 bits ou 65535 para 16 bits).

Phase-Correct PWM: Neste modo de funcionamento, o *timer* conta de 0 até atingir 255 e então conta regressivamente a 0. A saída assume nível lógico um durante a contagem progressiva e alterna para o nível lógico zero quando o valor do contador atinge o valor armazenado respectivo registrador OCRnA ou OCRnB fazendo o processo inverso quando em contagem regressiva. A frequência de saída neste modo será aproximadamente metade do valor no modo *Fast PWM* pois o *timer* faz duas vezes a contagem (progressiva e regressiva) por ciclo. Tomando como exemplo a saída A, a frequência do PWM neste modo de operação conforme documentação do microcontrolador, é dada por:

$$f_{OCnA} = \frac{F_{clock}}{2 \times N \times Val_{max}} \tag{3.2}$$

Onde:

f_{OCnA} frequência do PWM para a saída A;
F_{clock} frequência de operação do microcontrolador;
N valor do divisor do *prescaler* (1, 8, 64, 256, ou 1024);
Val_{max} valor máximo atingido pelo contador (255 para 8 bits ou 65535 para 16 bits).

Uso do PWM do ATmega328 no Motorino

O Motorino ocupa de forma exclusiva duas das seis saídas PWM disponíveis no microcontrolador (pinos 15 e 16), ambas controladas pelo *timer* TC1 [3]. Estas são ligadas diretamente nos pinos de habilitação das duas pontes H para que seja possível o controle de velocidade dos motores pela técnica de PWM. Detalhes sobre a interligação do microcontrolador com a ponte H podem ser encontrados na Seção 4.1. A implementação do *timer* TC1 no ATmega328 pode ser encontrada no Capítulo 20 do *datasheet* [3].

3.2 Circuito de Ponte H

Uma ponte H é um circuito eletrônico de potência que permite o controle de direção e velocidade de um motor. Normalmente o controle dos motores é feito com o auxílio de algum circuito lógico, usualmente um microcontrolador que controla o comportamento do motor para que o mesmo execute suas funções mecânicas em conformidade com a função para a qual foi projetado [27]. O microcontrolador provê instruções mas não tem capacidade de fornecer diretamente a potência elétrica necessária para o funcionamento do motor. Um circuito eletrônico em ponte H tem como entrada os sinais elétricos de baixa potência do microcontrolador e os amplifica de forma a fornecer ao motor os requisitos de potência elétrica que ele necessita.

3.2.1 Princípio de funcionamento

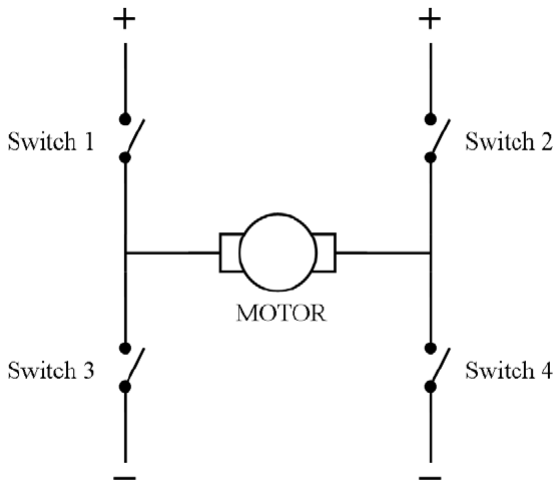


Figura 3.20: Diagrama conceitual de um circuito de controle ponte H. Fonte:[27]

Uma ponte H permite que motores DC de alto consumo trabalhe em ambas as direções acionados por sinais lógicos de baixo nível. O diagrama da Ponte H pode ser visualizado na Figura 3.20.
Os circuitos de acionamento de potência são compostos por transístores MOSFET [28] que apresentam baixa resistência elétrica quando em regime de saturação permitindo que a maior parte da energia drenada da fonte de alimentação seja transmitida ao motor e alta resistência elétrica quando em regime de corte, equivalente a um circuito aberto. Essas características fazem com que tenham um comportamento semelhante a chaves mecânicas controladas eletronicamente. No circuito da Figura 3.20 os transístores são representados pelas chaves numeradas de 1 a 4.

Para ilustrar conceitualmente o uso da ponte H no controle de rotação de motores, vamos supor que o movimento do motor no sentido horário se dê através a aplicação do polo positivo no lado esquerdo do motor e polo negativo no lado direito e o movimento no sentido anti-horário invertendo esta ligação. A Figura 3.21 ilustra as possíveis configurações permitidas paras as chaves 1 a 4.

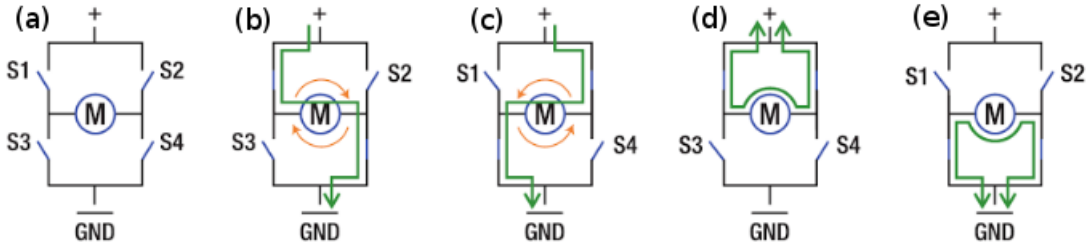


Figura 3.21: Configurações seguras para uma Ponte H. Adaptado de:[12]

Além das configurações onde o motor se encontra desenergizado (ilustração (a) na Figura 3.21), com rotação em sentidos horário e anti-horário (ilustrações (b) e (c) respectivamente na Figura 3.21), existem duas outras configurações onde o motor se encontra curto-circuitado com o polo positivo ou negativo da fonte de alimentação conforme mostram as ilustrações (d) e (e) respectivamente na Figura 3.21. Essas configurações promovem no motor uma condição de *freio elétrico* [12]. Isso acontece porque a maioria dos motores DC também funciona como um gerador elétrico e ao terem seus terminais curto-circuitados a corrente gerada pelo motor é forçada a circular através das bobinas de indução que provoca uma força contra-eletromotriz que se opõe ao movimento do motor. Essa condição é útil quando desejamos, por exemplo, que um robô parado não desça por um declive pela influência da gravidade.

Ao se executar o controle de uma Ponte H deve-se ter cuidado e evitar algumas configurações de acionamento que são prejudiciais ao circuito e em especial à fonte de alimentação. Algumas configurações que ilustram essa situação podem ser visualizadas na Figura 3.22.

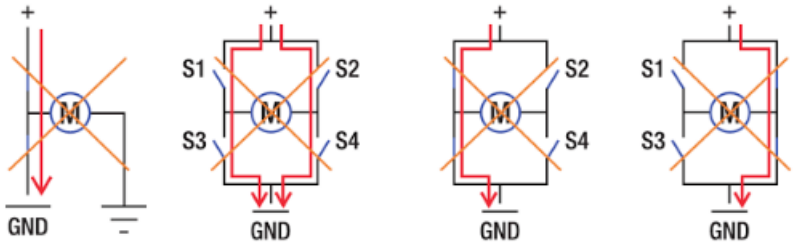


Figura 3.22: Configurações prejudiciais para uma Ponte H. Fonte:[12]

Em todos os casos mostrados na Figura 3.22 o acionamento das chaves provoca um curto-circuito na fonte de alimentação através dos transístores da ponte H. O excesso de corrente pode danificar a própria ponte H, a fonte de alimentação e em alguns casos

provocar acidentes. Mesmo que essas configurações aconteçam por um período curto de tempo, por exemplo, ocasionadas por uma transição lenta do acionamento das chaves, os transístores podem sofrer sobreaquecimento e como também sobrecarregar a fonte de alimentação. É fácil perceber que, através das quatro chaves podemos ter dezesseis configurações distintas, no entanto algumas são irrelevantes.
Um resumo das condições relevantes podem ser visualizadas na Tabela 3.3. A numeração das chaves tem como referência a Figura 3.20 e os conceitos de rotação nos sentidos horário e anti-horário estão em conformidade com o que foi convencionado no texto e ilustrados na Figura 3.21.

Tabela 3.3: Configurações relevantes de uma Ponte H

S1	**S2**	**S3**	**S4**	**Condição**
aberta	aberta	aberta	aberta	motor desenergizado
fechada	aberta	aberta	fechada	rotação sentido horário
aberta	fechada	fechada	aberta	rotação sentido anti-horário
fechada	fechada	aberta	aberta	condição de "*freio elétrico*"
aberta	aberta	fechada	fechada	condição de "*freio elétrico*"
fechada	aberta	fechada	aberta	curto-circuito da fonte de alimentação
aberta	fechada	aberta	fechada	curto-circuito da fonte de alimentação
fechada	fechada	fechada	fechada	curto-circuito da fonte de alimentação

3.2.2 Circuito Integrado L298N

É perfeitamente possível que o projetista construa sua própria ponte H com componentes discretos (transístores, diodos, resistores, capacitores e afins) e a insira no circuito, mas este processo traz uma complexidade muitas vezes desnecessária e algumas desvantagens, como por exemplo o aumento do espaço do circuito necessário para acomodar e interligar todos esses componentes. Um circuito integrado de ponte H consiste na sua implementação em um único chip e que necessita de poucos componentes externos para seu funcionamento. Além da economia de espaço em circuito proporcionada, espera-se que também esteja incorporado proteção contra curto-circuito, sobreaquecimento, sobrecarga e capacidade de operar sob altas frequências [12]. Assim estes circuitos integrados são muito mais resistentes a danos provocados por erros de uso e conexão do que circuitos construídos manualmente.
Atualmente existem uma grande diversidade de CIs com esta função, que incorporam as quatro chaves e métodos que permitem seu controle de forma prática e segura. Um exemplo popular e de baixo custo de CI que exerce tal função é o L298N.

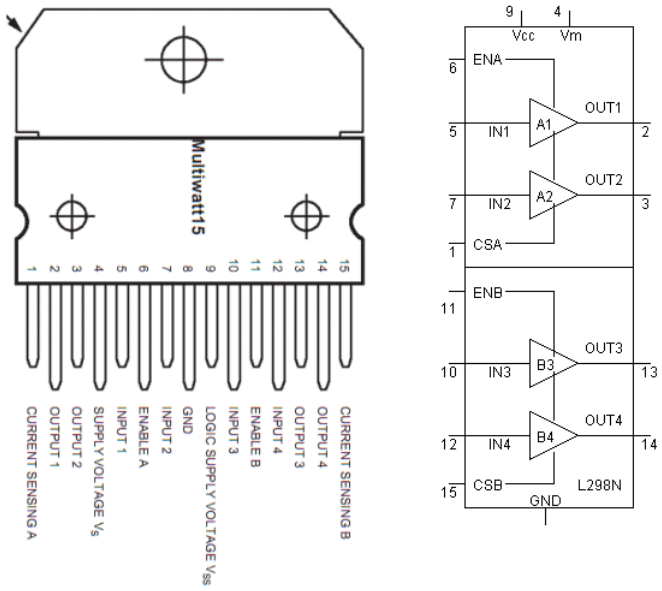


Figura 3.23: Pinagem do CI L298N. Adaptado de:[19]

O CI L298N fabricado pela *STMicroelectronics* é um componente de baixo custo e bastante versátil. Especialmente desenvolvido para controlar cargas indutivas, tem incorporado duas pontes H completas com circuito lógico para chaveamento dos transístores de potência com um controle de habilitação (Enable A e B) independente para cada uma das pontes, alimentação independente para o circuito lógico e para a carga. Com essa configuração é possível controlar até quatro relés ou solenóides (ligação em meia ponte H), dois motores DC ou um motor de passo de duas fases suportando até 2 ampères por ponte em regime de funcionamento contínuo, havendo a possibilidade de se ligar as duas pontes em paralelo caso haja a necessidade de se controlar dispositivos que consumam maior corrente [19]. O diagrama interno do CI L298N pode ser visualizado na Figura 3.24.

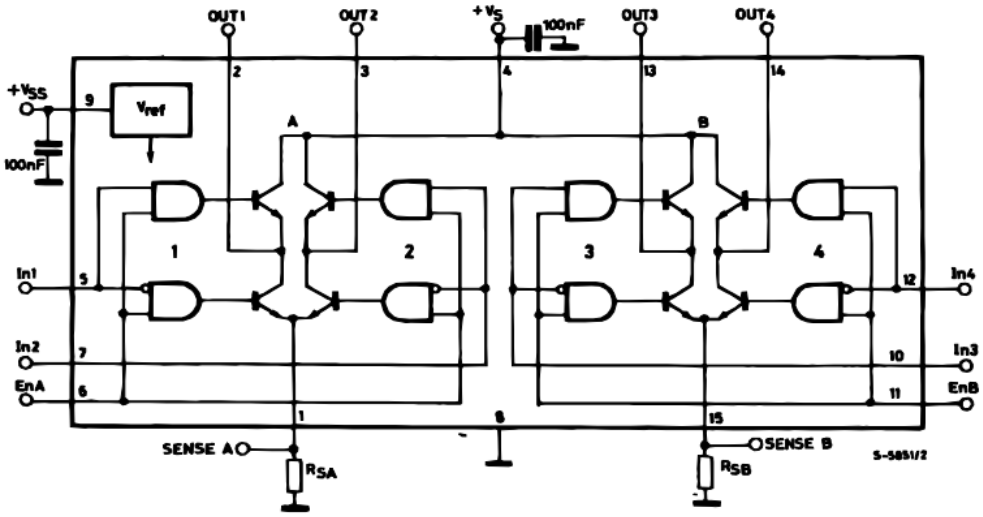


Figura 3.24: Diagrama interno do CI L298N. Fonte:[19]

O CI L298N permite que seja feito a descarga das pontes H independentemente do aterramento geral do CI através dos pinos 1 e 15, o que é útil para se medir as correntes que percorrem os motores com o auxílio de resistores de baixo valor, conectados em série, representados na Figura 3.24 por R_{SA} e R_{SB}. O Motorino utiliza-se de resistores de 0.5Ω para que seja possível a medição de corrente nos motores através de conversores analógico-digital existentes no microcontrolador.

Outra característica explorada pelo Motorino através de facilidades oferecidas por este CI é o controle de velocidade dos motores pelo método de PWM utilizando-se dos pinos de habilitação (*EnableA*, pino 6 e *EnableB*, pino 11). Este método é discutido com mais detalhes nas seções 3.1.5 e 4.2.

3.3 Considerações

Neste capítulo foi feita uma exploração dos principais elementos de hardware escolhidos para este trabalho, suas características, princípios de funcionamento e protocolos envolvidos, justificando assim o uso de tais componentes para se alcançar as funcionalidades desejadas no protótipo de *hardware* construído neste trabalho.
No próximo capítulo é feita a integração lógica e física destes componentes, o desenvolvimento do componente de *software* que dá funcionalidades ao *hardware* como também descreve o processo que torna possível a programação do protótipo de *hardware* desenvolvido com o uso da IDE do Arduino.

Capítulo 4

Desenvolvimento

Este capítulo apresenta a concepção lógica do *hardware*, a evolução da ideia para o circuito propriamente dito com a apresentação do diagrama esquemático, escolhas de projeto e a realização do projeto de *hardware* resultando num protótipo funcional. As etapas seguintes tratam do processo de gravação de do *software* base do microcontrolador, o *bootloader*, o desenvolvimento de uma biblioteca básica a fim de dar funcionalidade ao *hardware* com o intuito de validar o protótipo e a integração desta biblioteca e do protótipo ao ambiente de desenvolvimento integrado da plataforma Arduino.
As ferramentas de software utilizadas são para ambiente Linux X86-64 (Debian 8 com kernel 3.16.36-1+deb8u2) e IDE do Arduino versão 1.6.10.

4.1 Projeto do hardware

A Figura 4.1 mostra a concepção lógica do Motorino com as escolhas dos principais componentes que integram os blocos funcionais e seus sinais de conexão.

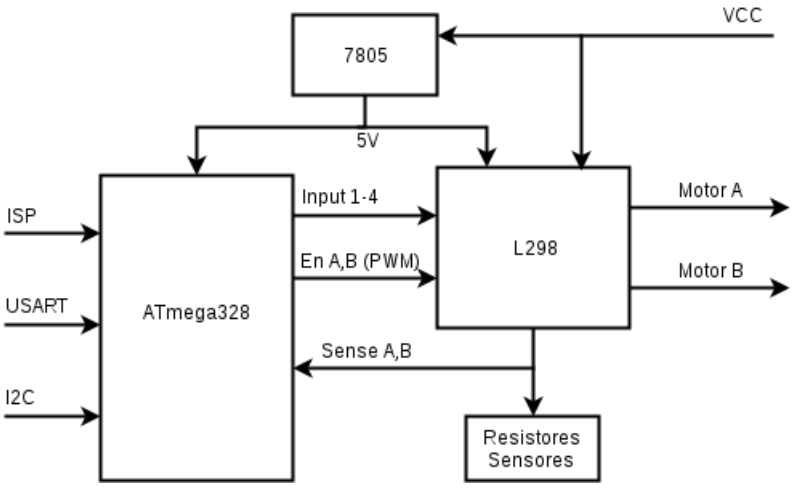


Figura 4.1: Diagrama em blocos do Motorino

Estes blocos são interligados ou fornecem ligações com o ambiente externo à placa através de conexões físicas, que podem ser de *entrada* ou *saída* (conforme direção das setas), por onde percorrem sinais elétricos com funcionalidades bem definidas, sendo estas:

VCC: Alimentação geral do sistema, conexão com fonte ou baterias;

5V: Alimentação de 5V para o ATmega328 e circuito lógico do CI L298N;

Motor A: Conector para ligação com o motor A;

Motor B: Conector para ligação com o motor B;

Sense A,B: Conexão dos *resistores sensores* com os ADCs do ATmega328;

Input 1-4: Controle lógico digital das pontes H do CI L298N;

EnA,B (PWM): Habilitações das pontes H, controladas pelo método PWM;

UART: Conector para comunicação UART (Conforme descrito na Seção 3.1.1);

ISP: Conector para comunicação ISP (Conforme descrito na Seção 3.1.2);

I2C: Conector para comunicação I2C (Conforme descrito na Seção 3.1.3);

As escolhas dos componentes que integram os blocos funcionais do *hardware* foram feitas conforme os critérios descritos a seguir.

Microcontrolador

O primeiro passo no desenvolvimento do *hardware* foi fazer a concepção lógica do projeto e escolher os principais componentes. Como descrito na Seção 3.1, a escolha do ATmega328 aconteceu de forma natural devido ao hardware embarcado, baixo custo e facilidades de integração à IDE do Arduino.

Ponte H

O próximo passo foi a escolha do circuito que comporia a ponte H. Conforme descrito na Seção 3.2, utilizar um CI que implemente a ponte H tras vários benefícios perante um circuito discreto. O CI LM298N [19] surgiu como uma escolha viável, pois implementa duas pontes H, controle lógico dos transistores internos, controle de habilitação e conexão externa para medição de corrente no motor, de forma individual para cada uma das pontes H conforme pode ser visto na Figura 3.24. É capaz de controlar cargas que trabalhem com até 50 Volts e corrente de 2 Ampères (recomendado que haja dissipação de até 25 Watts por ponte H nas cargas em regime de funcionamento contínuo) e possibilidade de ligação em paralelo das duas pontes H para controle de correntes até 4 Ampères,

características elétricas suficientes para controlar grande parte dos motores DC e de passo utilizados em projetos de robótica.
Além das características técnicas, o L298N é de fácil obtenção por ser integrante de vários módulos comerciais para o Arduino e tem baixo custo comparado a outros CIs que integram a mesma função. Uma das poucas desvantagens deste CI é que não integra os diodos de recuperação dos transístores, precisando serem adicionados externamente.

Regulador de tensão

Para que não haja necessidade de duas fontes de alimentação distintas para o circuito, foi utilizado um regulador de tensão para adequar os níveis de tensão da alimentação usada para os motores para alimentar o ATmega328 e o circuito lógico do LM298N. O ATmega328 [3] é capaz de trabalhar em tensões que variam de 1.8V a 5.5V no entanto para operar com clock de 16MHZ a alimentação deve ser de 5V conforme especificado pelo fabricante. O circuito lógico do LM298N [19] demanda uma alimentação tipicamente de 5V, portanto foi escolhido um regulador de tensão positiva de 5V para alimentar esses dois componentes.
Um CI bastante popular, de baixo custo e fácil obtenção que cumpre essa função é o 7805. Este CI possui uma versão *low power*, o 78L05 [32] que fornece tipicamente 100mA de corrente de saída podendo chegar até 250mA, rejeição de ruído, proteção térmica, dentre outras características técnicas. Pode ser encontrado com encapsulamento TO-92, assim ocupa pouco espaço e não necessita de dissipador térmico adicional. Essas características tornam o 78L05 uma escolha viável para o protótipo.

Resistores sensores

Os *resistores sensores* (conforme mencionados na documentação do CI L298N [19]) são resistores de filme metálico ou carbono (não indutores) ligados em série com o circuito de potência e com valores de resistência baixos o suficiente para que a queda de tensão provocada por eles interfira o mínimo possível na entrega de potência à carga.
O ADC do ATmega328 mede a queda de tensão sobre estes resistores e por relação direta dada pela *Lei de Ohm* podemos inferir o valor da corrente que circula pelos resistores, que é a mesma que circula pelos motores dado que estes fazem parte de um circuito em série. O valor da resistência escolhida para cada ponte H foi de 0.5Ω pelos seguintes motivos: é possível se obter através da ligação em paralelo de dois resistores de 1Ω, sendo este um valor comercial fácil de ser obtido e supondo que a ponte H trabalhe com valor máximo de corrente nominal de 2A, pela *Lei de Ohm* teremos uma queda de tensão mensurável nos resistores de 1V e este valor é próximo do fundo de escala dos ADCs do ATmega328 (quando usam a tensão interna de referência de 1.1V), o que permite melhorar a precisão das medidas diminuindo o erro provocado pela quantização do ADC.

Mapeamento dos pinos do ATmega328 ao L298N

A Figura 3.4 mostra a pinagem e respectivas funcionalidades do ATmega328 e a Figura 3.24 o diagrama interno, pinagem e respectivas funcionalidades do CI L298N.
A ligação entre o microcontrolador e o driver de motores é a critério do projetista, desde que o pino do microcontrolador ofereça a funcionalidade requerida pelo respectivo pino do CI do driver.
A Tabela 4.1 mostra o mapeamento feito dos pinos do ATmega328 ao L298N, que representa a especificação dos três sinais (*Input 1-4*, *EnA,B (PWM)* e *Sense A,B*) que interligam os blocos lógicos *ATmega328* e *L298* do diagrama em blocos mostrado na Figura 4.1.

Tabela 4.1: Mapeamento dos pinos do ATmega328 ao L298N e respectivos sinais lógicos

Pino ATmega328	**Função**	**Pino L298N**	**Função**	**Sinal (Figura 4.1)**
6	Saída digital	5	In1	Input 1-4
11	Saída digital	7	In2	Input 1-4
12	Saída digital	10	In3	Input 1-4
13	Saída digital	12	In4	Input 1-4
15	PWM	6	EnA	EnA,B (PWM)
16	PWM	11	EnB	EnA,B (PWM)
23	ADC	1	SenseA	Sense A,B
24	ADC	15	SenseB	Sense A,B

Diagrama esquemático do circuito

Após a concepção lógica, escolha dos principais componentes e suas conexões, a próxima etapa do projeto do *hardware* consiste na determinação das conexões físicas dos componentes principais e agregação de componentes extras que têm a finalidade de atender às recomendações dos fabricantes, como também dar suporte a alguma funcionalidade dos componentes principais. Ao final deste processo se obtém o *diagrama esquemático*, que é uma representação completa do circuito.

Os principais componentes do protótipo são os circuitos integrados L298N, 78L08 e o ATmega328. Os critérios de escolha dos componentes adicionais, seus códigos representativos e valores são descritos a seguir e são representados no diagrama esquemático do protótipo conforme mostra a Figura 4.2.

L298N: D1 a D8 têm a função de diodos de recuperação, externos ao CI; Na documentação é sugerido que um capacitor de 100nF seja inserido próximo aos pinos Vss e Vs, sendo estes os capacitores C8 e C7 respectivamente. Também é sugerido que seja inserido um capacitor de maior valor quando a alimentação se encontra longe do CI, portanto foi inserido no circuito o capacitor C1.

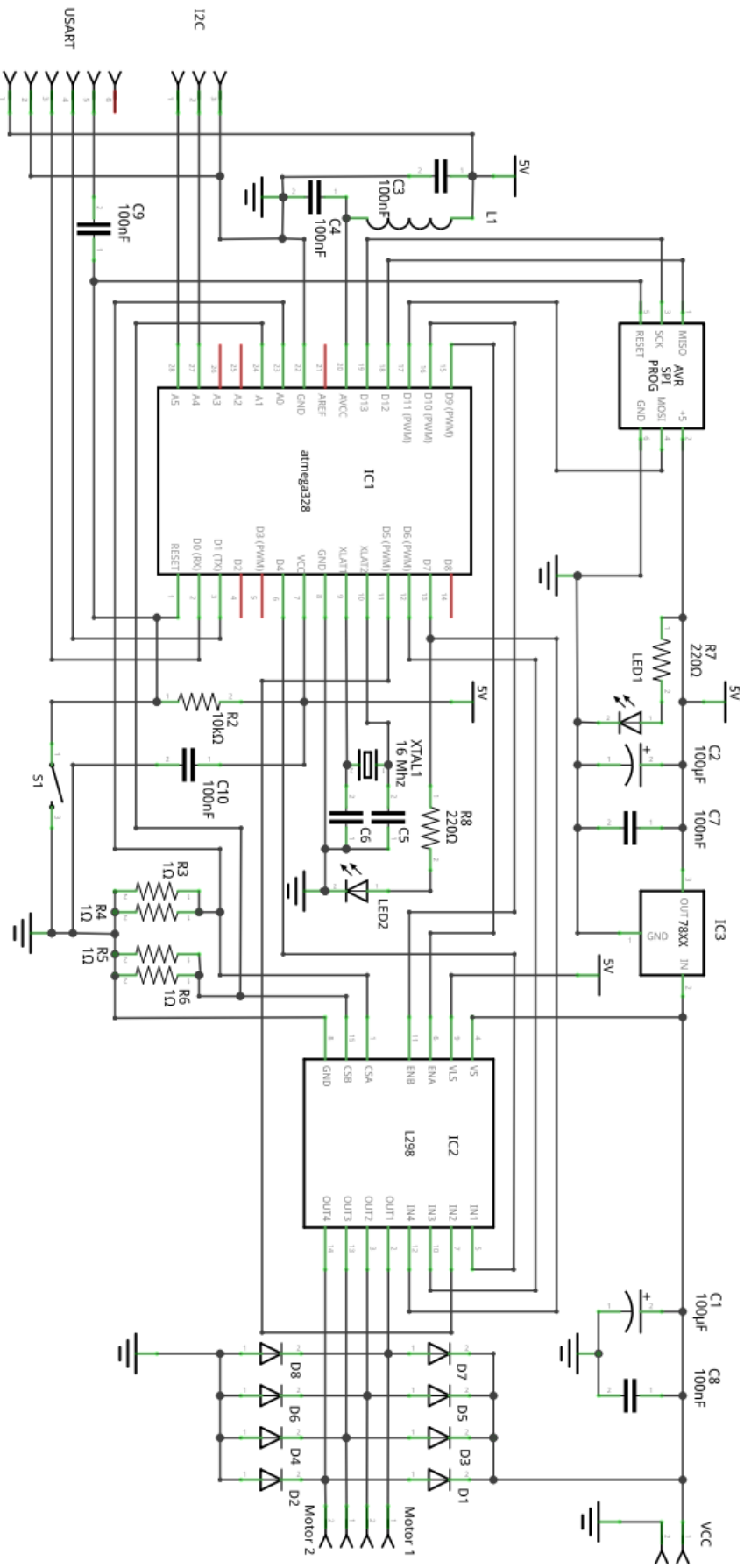


Figura 4.2: Diagrama esquemático do Motorino

78L05: O fabricante sugere que sejam colocados capacitores maiores que 100nF próximo ao CI nos pinos de entrada e saída. No circuito este papel é desempenhado pelos capacitores C1 e C2.

ATmega328: O microcontrolador utiliza circuito oscilador externo de 16Mhz para compatibilidade com o Arduino. Este circuito é composto por XTAL1, C5 e C6; O fabricante sugere que seja inserido um filtro LC no pino AVcc para evitar interferências devido ao cheveamento do microcontrolador, papel desempenhado por L1 e C4 como também é sugerido a inserção do capacitor C10 próximo ao pino Vcc. Além destes, foram inseridos o resistor de *pull-up* R2 para o pino de *reset* e a chave S1 para fazer *reset* manual.

Montagem do protótipo

Com base no diagrama esquemático mostrado na Figura 4.2 foi montado um protótipo do Motorino em placa de circuito impresso genérica ilhada de tamanho 5x7 cm conforme pode ser visto na Figura 4.3.

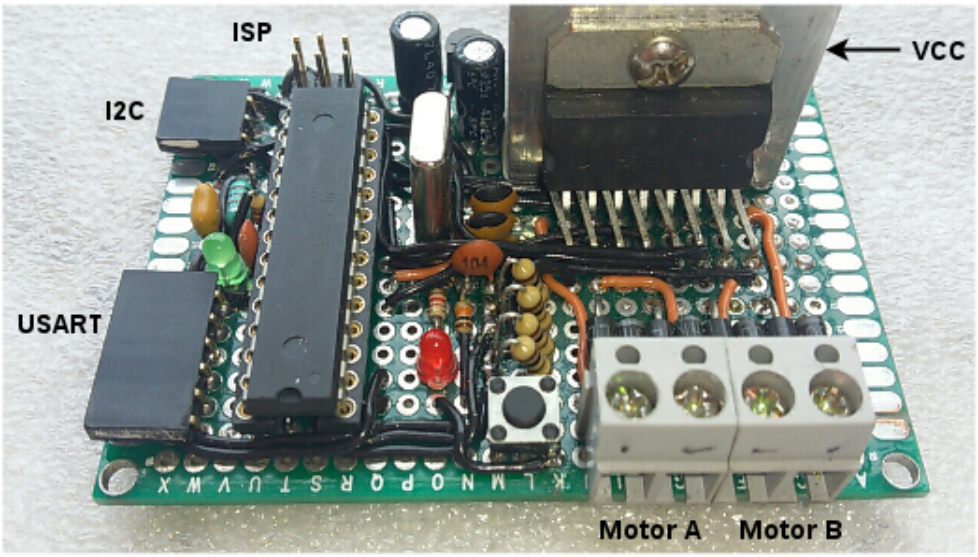


Figura 4.3: Placa do protótipo de *hardware* do Motorino

Na Figura 4.3 são indicadas as interfaces para comunicação via USART e I2C (à esquerda), comunicação via ISP (na parte superior esquerda), a conexão para alimentação do circuito (na parte superior direita) e as conexões para os motores A e B (na parte frontal).

4.2 Biblioteca Motorino

Uma biblioteca para a plataforma Arduino é um *software* que fornece funcionalidade específica a um *hardware* agregado ao microcontrolador. O uso de uma biblioteca simplifica o desenvolvimento de aplicações, pois precisa somente ser referenciada via diretiva *include* no programa em desenvolvimento para que suas funções possam ser acessadas e utilizadas pelo desenvolvedor. Assim podemos estender o uso do Arduino incorporando bibliotecas específicas durante o desenvolvimento de uma aplicação.

A biblioteca do Motorino inicializa os recursos de *hardware* embarcados no microcontrolador necessários ao funcionamento da placa e fornece acesso de forma simplificada às funções de controle e leitura de parâmetros da ponte H incorporada, promovendo abstração do *hardware* e de rotinas de controle de motores DC. A biblioteca foi escrita em linguagem C++ e a classe Motorino conta com as estruturas e métodos conforme listados a seguir.

Estrutura: *Motor_t*

Função: Armazena os valores instantâneos referentes à configuração e ao estado do referido motor. A classe implementa uma instância desta estrutura para cada um dos dois motores controlados.

Membros:
uint8_t enable Pino do microcontrolador que exerce a função de habilitação;
uint8_t input_a Pino do microcontrolador conectado ao *Controle A* da ponte H;
uint8_t input_b Pino do microcontrolador conectado ao *Controle B* da ponte H;
uint8_t i_sense Pino do microcontrolador conectado ao resistor sensor;
uint8_t duty_cycle Valor corrente do *duty cycle* (potência do motor: 0 a 100);
uint8_t direction Direção do motor (0 = parado; 1 = horário; 2 = anti-horário);
int16_t current Último valor da corrente medida, em miliampères;

void init()

Função: Executado durante a instanciação da classe. Configura como saída e habilita função do PWM os pinos do microcontrolador conectados às entradas de habilitação da ponte H; Configura como saída digital os pinos do microcontrolador conectados às entradas de controle da ponte H; Habilita os ADCs dos pinos do microcontrolador conectados aos resistores sensores; Inicializa o *duty cycle* dos PWMs com valor zero; Inicializa ambos motores na condição de "parado"; Habilita a referência interna de tensão para os ADCs (1.1V);
Parâmetros: Nenhum;
Retorno: Nenhum;

Motor_t* selectMotor(uint8_t motor)

Função: Selecionar um dos motores;
Visibilidade: Privada;
Parâmetros: Inteiro 1 ou 2 (referente ao número do motor);
Retorno: Ponteiro para a estrutura do referido motor;

void setPower(uint8_t motor, uint8_t duty_cycle)

Função: Alterar o *duty cycle* (potência) do motor;
Visibilidade: Pública;
Parâmetros: Número do motor, valor do *duty cycle*;
Retorno: Nenhum;

void setDirection(uint8_t motor, uint8_t direction)

Função: Alterar a direção de rotação ou parar o motor;
Visibilidade: Pública;
Parâmetros: Número do motor, direção do motor;
Retorno: Nenhum;

uint8_t getPower(uint8_t motor)

Função: Ler o valor do *duty cycle* (potência) do motor;
Visibilidade: Pública;
Parâmetros: Número do motor;
Retorno: Valor do *duty cycle*;

uint8_t getDirection(uint8_t motor)

Função: Ler o valor referente a direção do motor;
Visibilidade: Pública;
Parâmetros: Número do motor;
Retorno: Valor correspondente a direção do motor;

int16_t readCurrent(uint8_t motor)

Função: Ler o valor instantâneo da corrente do motor;
Visibilidade: Pública;
Parâmetros: Número do motor;
Retorno: Corrente do motor em miliampères;

void revMotion(uint8_t motor)

Função: Inverter o sentido de rotação do motor;
Visibilidade: Pública;
Parâmetros: Número do motor;
Retorno: Nenhum;

String cmdExec(char* str)

Função: Interpretar um string e executar a função correspondente;
Visibilidade: Pública;
Parâmetros: String com a função e parâmetro codificados;
Retorno: String com o resultado da função executada ou informação de erro;

Uso do método *cmdExec*

O método *cmdExec* foi implementado com a intenção de atuar como um agente facilitador ao acesso das funcionalidades proporcionadas pelos demais métodos públicos. Com funcionamento semelhante a uma API (*Application Programming Interface*) é especialmente útil quando o Motorino é controlado por um dispositivo externo conectado às interfaces UART, I2C, SPI ou mesmo através de um dispositivo WIFI ou Bluetooth agregado aos pinos de I/O livres. O uso deste método deve ser feito pela aplicação principal que gerencia a interface de comunicação lendo o string recebido, chamando o método e de forma opcional gerenciando ou reencaminhando à interface o string de retorno do método com o resultado da operação.

Entrada do método *cmdExec*

O string de entrada é formado por dois blocos seperados pelo caratere "&". O primeiro indica sobre qual motor será executada a ação e o segundo com a ação e parâmetro (caso exista) possuindo o seguinte formato:

[Token_Motor]=[1 ou 2]&[Token_Função]=[Parâmetro]

Os *tokens* são compostos pelo caracteres maiúsculos: M (Motor), D (Direção), P (Potência), R (Reversão), C (Corrente) e indica o elemento sobre qual será efetuada a ação.
O *parâmetro*, quando se refere a uma ação de escrita deve possuir um valor numérico dentro de um intervalo válido para a ação correspondente e quando se refere a uma ação de leitura deve ser utilizado como parâmetro o caractere "?".
A Tabela 4.2 mostra quais tokens são utilizados para as operações de escrita, ações correspondentes e os valores de parâmetros válidos para as respectivas operações.

Tabela 4.2: Operações de escrita gerenciadas pelo método *cmdExec*

Token	**Ação**	**Intervalo de valores de parâmetro válidos**
M	Seleciona Motor	1 ou 2
D	Sentido de rotação do motor	0 = parado; 1 = horário; 2 = anti-horário;
P	Potência do motor	Inteiro de 0 a 100
R	Reversão da rotação do motor	(Nenhum)

Exemplos:

M=1&P=50 ⇒ Determina em 50% a potência do motor 1;

M=2&D=0 ⇒ Interrompe o movimento do motor 2;

M=1&R ⇒ Inverte a rotação do motor 1;

A Tabela 4.3 mostra quais tokens são utilizados em operações de leitura e os respectivos intervalos de valores de retorno considerados válidos.

Tabela 4.3: Operações de leitura gerenciadas pelo método *cmdExec*

Token	**Leitura**	**Intervalo de valores de retorno esperados**
D	Sentido de rotação do motor	0 = parado; 1 = horário; 2 = anti-horário;
P	Potência do motor	Inteiro de 0 a 100
C	Corrente do motor	Inteiro de 0 a 2200

Exemplos:

M=1&P=? ⇒ Leitura da potência do motor 1;

M=2&D=? ⇒ Leitura da direção do motor 2;

M=1&C=? ⇒ Leitura da corrente do motor 1;

Retorno do método *cmdExec*

O metodo *cmdExec* retorna um string para cada ação informando o resultado da ação ou erro. O formato do string depende do tipo de operação. Uma operação de leitura bem sucedida deve retornar um string numérico, o uso de um token desconhecido gera como resposta o string *?:CMD* e para os demais casos o retorno tem a forma:

[Token]:[ERR | OK!]

Exemplos de entradas e respectivos retornos:

M=3&P=10 ⇒ M:ERR

M=2&D=2 ⇒ D:OK!

M=1&C=? ⇒ 1000

M=1&P=? ⇒ 33

Z=2&D=1 ⇒ ?:CMD

M=1&R ⇒ R:OK!

A Tabela 4.4 apresenta um resumo das possibilidades de retorno do método *cmdExec* dados a operação representada por seu respectivo token, o tipo de operação e o motivo que ocasionou tal resposta.

Tabela 4.4: Possíveis strings de retorno do método *cmdExec*

Tokens	**Tipo de Oper.**	**Retorno**	**Motivo**
	Leitura / Escrita	?:CMD	Token não identificado
[D | P | C]	Leitura	Inteiro	Operação bem sucedida
[D | P | C]	Leitura	[D | P | C]:ERR	Parâmetro diferente de "?"
[M | D | P | R]	Escrita	[M | D | P | R]:OK!	Operação bem sucedida
[M | D | P | R]	Escrita	[M | D | P | R]:ERR	Parâmetro inválido

A biblioteca do Motorino não gerencia retornos informativos de erro ou de operação bem sucedida do método *cmdExec*, o gerenciamento e tomadas de decisões devem ser feitos na aplicação principal ou no *software* do dispositivo conectado que comanda o Motorino.

4.3 Integração com a IDE do Arduino

Para que seja possível trabalhar com a placa do Motorino utilizando a IDE do Arduino é necessário que o mesmo identifique a placa e agregue a sua biblioteca. Nesta seção são descritos os procedimentos para que seja feito o reconhecimento do *hardware* e *software* desenvolvidos diretamente na estrutura da IDE versão 1.6.10.

4.3.1 Integração da Biblioteca

Na plataforma Arduino existem três categorias diferentes de bibliotecas de software:

Biblioteca Core : Biblioteca base, fornece as funcionalidades básicas e essenciais e vem instalada na IDE do Arduino por padrão. Para a utilização das funcionalidades desta biblioteca não é necessário fazer a inclusão do arquivo de cabeçalho no programa principal. A biblioteca *core* do Arduino encontra-se na pasta:
`(raiz-da-ide)/hardware/arduino/avr/cores/arduino/`

Bibliotecas Padrão : Fornecem funcionalidades básicas porém não essenciais e menos utilizadas, como por exemplo, bibliotecas para o uso da EEPROM, interfaces I2C e SPI. Também vêm instaladas na IDE do Arduino por padrão porém é necessário que se faça a inclusão do arquivo de cabeçalho no programa principal para ter acesso à biblioteca que oferece suporte ao *hardware* desejado. Encontram-se na pasta:
`(raiz-da-ide)/hardware/arduino/avr/libraries/`

Bibliotecas Adicionais : Fornecem suporte a *hardwares* de terceiros. São disponibilizadas por desenvolvedores diversos que contribuem voluntariamente com software para a plataforma e não são distribuídas por padrão com o IDE do Arduino, precisam ser instaladas. Elas oferecem funções adicionais a bibliotecas existentes ou novas funcionalidades não presentes em nenhuma biblioteca padrão, permitindo estender o uso do Arduino de forma praticamente ilimitada. Estas bibliotecas são instaladas na pasta:
`(raiz-da-ide)/libraries/(nome-da-biblioteca)/`

A biblioteca do Motorino foi adicionada com a categoria *Biblioteca Adicional* e para ser reconhecida pela IDE do Arduino deve seguir as especificações de estrutura de pastas, arquivos e metadados conforme documentação na Wiki do Arduino que podem ser encontradas no seguinte endereço: `https://github.com/arduino/Arduino/wiki/Arduino-IDE-1.5:-Library-specification`
A estrutura da biblioteca do Motorino foi organizada conforme mostrado na Figura 4.4.

```
alexandre@desktop:~$ tree arduino-1.6.10/libraries/Motorino
arduino-1.6.10/libraries/Motorino
├── examples
├── library.properties
├── README.adoc
└── src
    ├── Motorino.cpp
    └── Motorino.h
```

Figura 4.4: Estrutura de pastas e arquivos da biblioteca do Motorino

examples : Pasta detinada a aplicações com exemplos de uso da biblioteca;
librarie.properties : Arquivo de metadados da biblioteca utilizado pela IDE;
README.adoc : Arquivo com informações sobre licença e uso da biblioteca;
src : Pasta com arquivos fonte e de cabeçalho da biblioteca;

A instalação da biblioteca pode ser feita de três maneiras diferentes: Copiando-a para seu devido local de destino, através da IDE por meio de um arquivo local no formato *zip* ou utilizando o gerenciador de bibliotecas da IDE que busca os arquivos remotamente. Após a instalação a biblioteca deve ser reconhecida pela IDE como mostra a Figura 4.5.

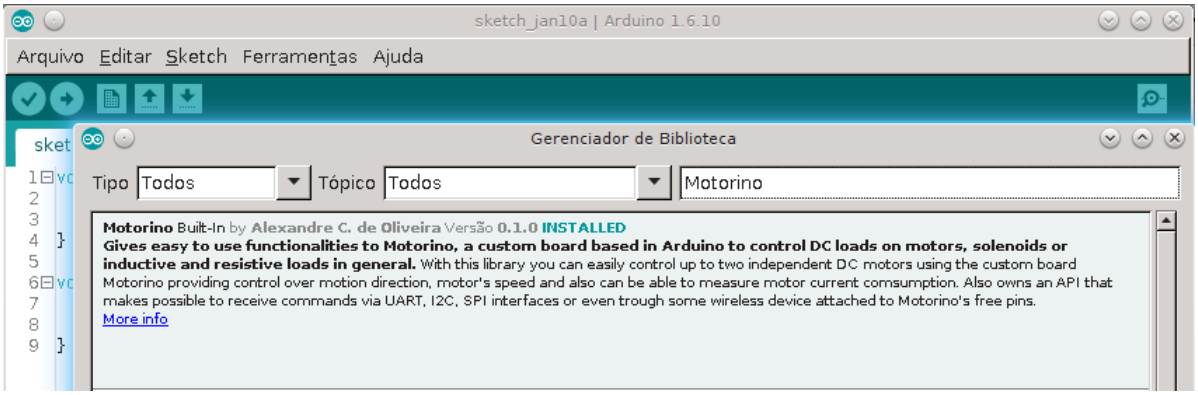


Figura 4.5: Biblioteca do Motorino integrada à IDE do Arduino

4.3.2 Integração da Placa

A partir da versão 1.6.4 a IDE do Arduino conta com um sistema de gerenciamento de placas onde é possível fazer a integração de placas de terceiros bem como dar suporte a todo o *software* necessário ao funcionamento das mesmas (arquivos de configuração e metadados, bibliotecas *core* e *padrão*, bootloaders, gravadores) de forma independente e isolada das placas oficiais do Arduino através de repositórios onde o mantenedor é o único responsável por todo o conteúdo e atualizações.
Para que uma placa de terceiros seja reconhecida, a IDE do Arduino busca os arquivos de configuração dentro de subpastas com a seguinte estrutura:
`(raiz-da-ide)/hardware/(fabricante)/(arquitetura)/`

A estrutura dos arquivos e pastas que definem o *hardware* do Motorino foi baseada nas placas oficias do Arduino usando-se apenas a estrutura básica, mantendo uma versão dos elementos exclusivos do Motorino e os de uso compartilhado mantém ligação com elementos padrões do Arduino através de links simbólicos. A documentação oficial do Arduino sobre especificação de *hardware* de terceiros encontra-se em: `https://github.com/arduino/Arduino/wiki/Arduino-IDE-1.5-3rd-party-Hardware-specification`
A estrutura ficou definida como mostra a Figura 4.6.

```
alexandre@desktop:~$ tree arduino-1.6.10/hardware/motorino
arduino-1.6.10/hardware/motorino
└── avr
    ├── boards.txt
    ├── bootloaders
    │   └── optiboot
    │       ├── boot.h
    │       ├── Makefile
    │       ├── optiboot_atmega328.hex
    │       ├── optiboot_atmega328.lst
    │       ├── optiboot.c
    │       ├── pin_defs.h
    │       ├── README.TXT
    │       └── stk500.h
    ├── cores -> ../../arduino/avr/cores
    ├── libraries -> ../../arduino/avr/libraries
    ├── platform.txt -> ../../arduino/avr/platform.txt
    └── variants
        └── standard
            └── pins_arduino.h
```

Figura 4.6: Estrutura de arquivos e pastas da IDE para o hardware do Motorino

boards.txt : Arquivo que contém definições e metadados para as placas suportadas;
bootloaders : Bootloader usado no Motorino;
cores : Ligação com a biblioteca de base do Arduino;
libraries : Ligação com a biblioteca de padrão do Arduino;
platform.txt : Ligação com o arquivo de configuração do Arduino que contém informações sobre certos aspectos que são específicos da plataforma AVR (flags do compilador, caminhos e bibliotecas do sistema).

variants : Definições de nome, identificador e funções dos pinos do microcontrolador para a IDE utilizado em cada variante de *hardware* de placa;

Dentre estes arquivos o que merece maior atenção é o `boards.txt`. Ele é responsável por identificar a placa na IDE, definir a MCU e suas características, compilador, ferramenta e velocidade de gravação, bootloader utilizado, core utilizado, variante de *hardware*, etc. A configuração do Motorino teve como base a seção referente à configuração da placa do *Arduino Uno* (devido as semelhanças entre os *hardwares*) e a documentação de referência obtida a partir do site:
`https://code.google.com/archive/p/arduino/wikis/Platforms.wiki`

Com a placa perfeitamente integrada é possível gravar o *bootloader*, compilar código fonte, promover comunicação com o PC via UART, carregar código compilado para a placa, enfim, todas as funcionalidades para uma placa oficial do Arduino também estarão disponíveis para a placa do Motorino. A Figura 4.7 mostra um exemplo de compilação de código feita baseada no *hardware* do Motorino.

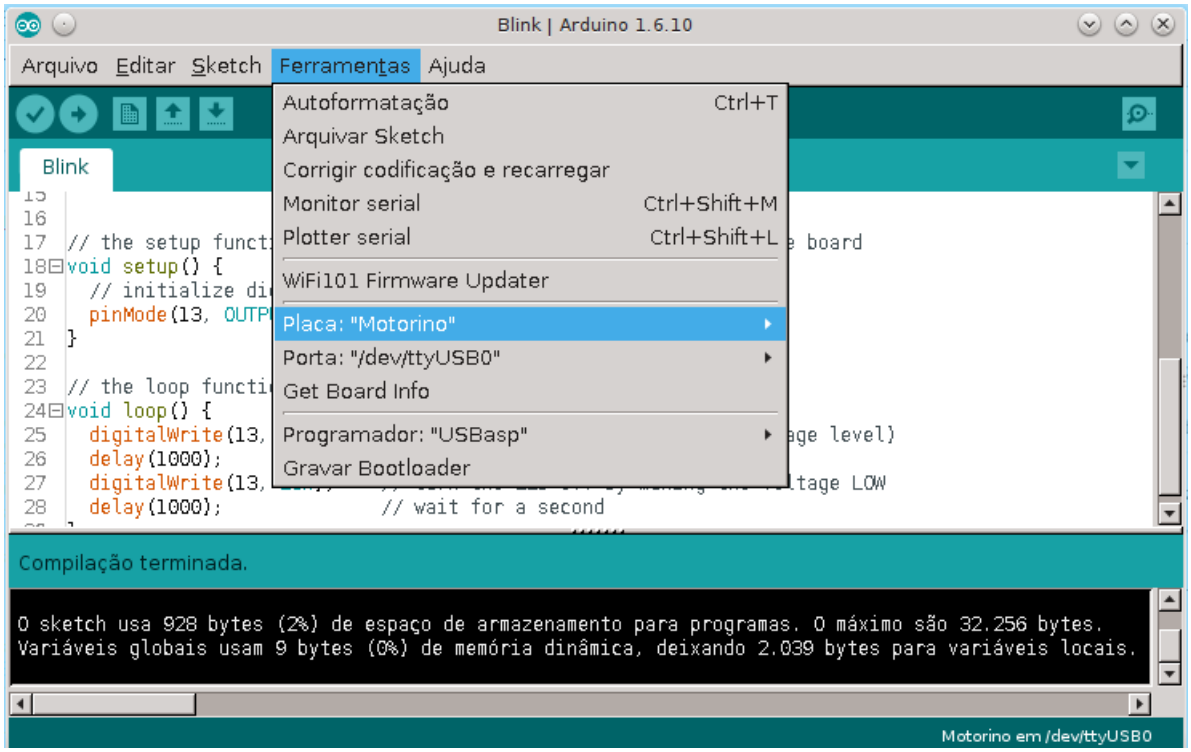


Figura 4.7: Hardware do Motorino integrado à IDE do Arduino

4.4 Gravação do Bootloader

O *bootloader* é um pequeno trecho de código que é executado imediatamente após o microcontrolador ser energizado ou sofrer um processo de *reset*. Apesar de seu uso não ser obrigatório, o *bootloader* é um recurso muito útil pois pode ser usado para configurar funcionalidades de hardware e executar certas funções durante o processo de inicialização. Após a execução do código do *bootloader* o controle é transferido para a área de memória não volátil onde se encontra o código da aplicação. O ATmega328 possui um suporte nativo ao uso do *bootloader* onde é possível separar a memória *flash* em dois blocos distintos, *bootloader* e aplicação, se auto-atualizar e até reprogramar toda a memória *flash* interna do microcontrolador. Os detalhes sobre as funcionalidades do suporte ao *bootloader* no ATmega328 são encontrados no Capítulo 30 do *datasheet*.

Bootloader do Motorino

O Motorino utiliza o *bootloader* do Arduino, sendo este o mesmo para todas as placas que utilizam o microcontrolador ATmega328. Além de configurar alguns recursos de hardware para seu uso através da IDE, ele inicializa a UART e aguarda a comunicação via PC para que receba o código compilado da aplicação programada via IDE e o grava no espaço de memória destinado à aplicação. Promovendo uma comunicação direta entre microcontrolador e PC evita-se o uso de um programador cada vez que for necessário carregar um novo código de aplicação tornando o processo mais rápido e prático. O *bootloader*, versão compilada e código fonte, acompanham a IDE do Arduino e podem ser encontrados na pasta `(raiz-da-ide)/hardware/arduino/avr/bootloaders`.

Gravador AVR-ISP

O microcontrolador, quando novo, não possui *software* algum embarcado que auxilie no processo de gravação de sua memória *flash*, mas possui suporte em *hardware* para que a programação seja feita através da interface *SPI* com o auxílio de um gravador do tipo *AVR-ISP*, conforme descrito no Capítulo 1 do *datasheet*. O gravador utilizado neste trabalho é uma versão comercial do gravador *USBasp - USB programmer for Atmel AVR controllers* de *hardware* e *software* abertos, disponíveis em `http://www.fischl.de/usbasp/`. No ambiente Linux, o *USBasp* é reconhecido automaticamente (sem a necessidade de instalação de drivers de dispositivo), conforme pode ser visto na Figura 4.8.

```
alexandre@desktop:~$ dmesg | tail -n 5
[1394631.589401] usb 6-1.3: new low-speed USB device number 75 using ehci-pci
[1394631.703791] usb 6-1.3: New USB device found, idVendor=16c0, idProduct=05dc
[1394631.703798] usb 6-1.3: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[1394631.703804] usb 6-1.3: Product: USBasp
[1394631.703807] usb 6-1.3: Manufacturer: www.fischl.de
```

Figura 4.8: Reconhecimento do gravador USBasp pelo sistema operacional

Processo de gravação do bootloader

O processo de gravação utilizando-se a IDE do Arduino é bastante simples tendo em vista que possui esta funcionalidade e automatiza todo o processo. Após a integração da placa à IDE do Arduino, conforme descrito na Seção 4.3.2, basta que na IDE sejam selecionados a placa, o gravador apropriado, a porta na qual o gravador foi reconhecida pelo sistema e executar a função "Gravar Bootloader"através do menu "Ferramentas".
O resultado da execução do processo bem como eventuais erros são mostrados na área de log da própria IDE, conforme pode ser visto na Figura 4.9.

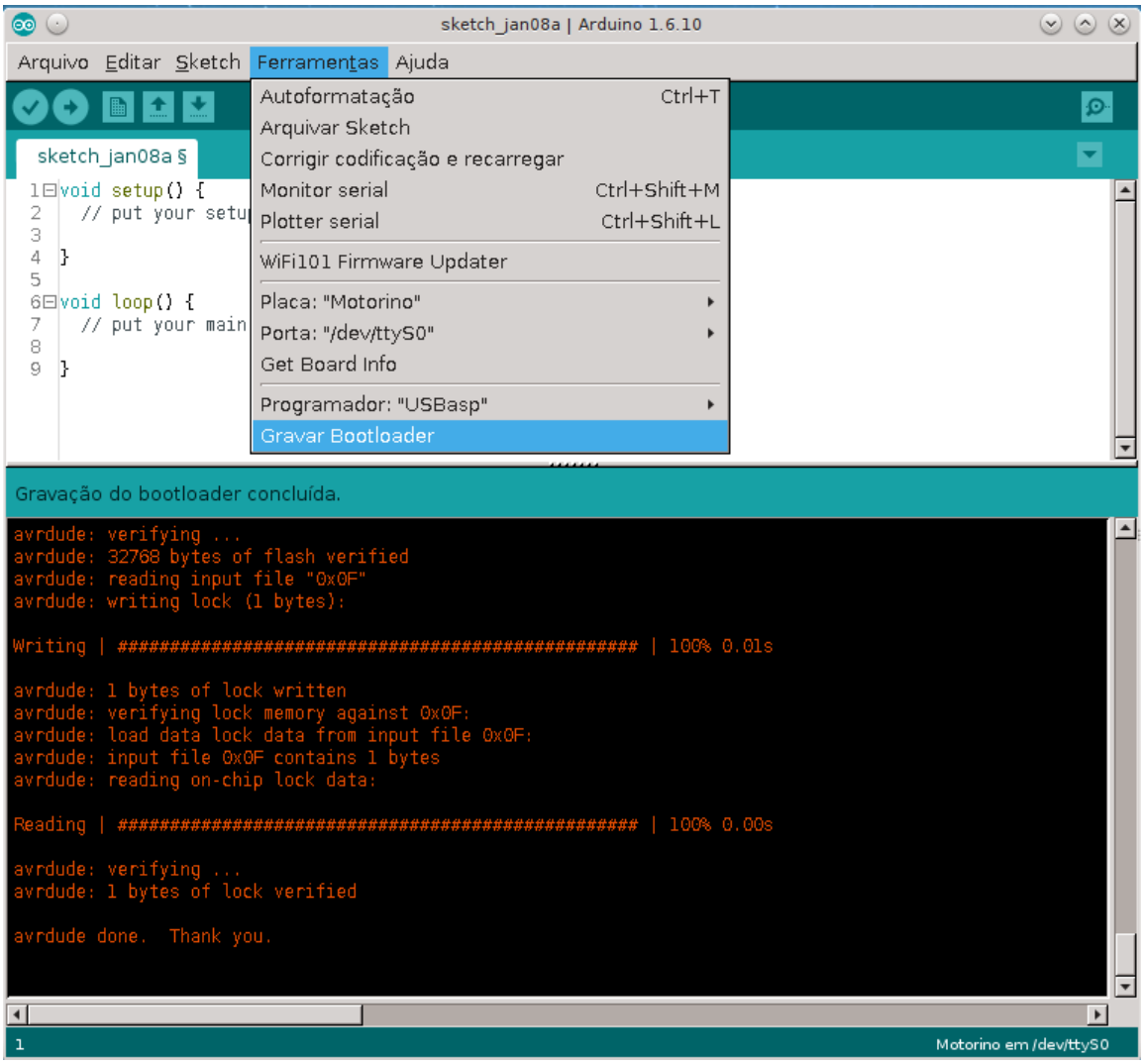


Figura 4.9: Gravação do bootloader via IDE do Arduino

O reconhecimento do microcontrolador da placa bem como a imagem do *bootloader* a ser gravado dependem da configuração específica da placa contida no arquivo `(raiz-da-ide)/hardware/motorino/avr/boards.txt` conforme descrito na Seção 4.3.2.

4.5 Conexão com o PC

O Motorino foi projetado para ser integrado e programado através da IDE do Arduino utilizando os mesmo procedimentos de uma placa oficial, entretanto devido à simplificação do projeto de *hardware* ele não conta com um circuito adicional capaz de conectá-lo diretamente na porta USB do PC como acontece com as placas oficiais.
Para conectar o Motorino ao PC via porta USB foi utilizado um conversor USB para UART com níveis de tensão TTL, compatíveis com o microcontrolador. O conversor usado é baseado no CI CP2102 [14] sendo reconhecido automaticamente pelo kernel do Linux conforme pode ser visto na Figura 4.10.

```
alexandre@desktop:~$ dmesg | tail -n 9
[123860.220839] usb 6-1.1.4: new full-speed USB device number 8 using ehci-pci
[123860.331013] usb 6-1.1.4: New USB device found, idVendor=10c4, idProduct=ea60
[123860.331020] usb 6-1.1.4: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[123860.331025] usb 6-1.1.4: Product: CP2102 USB to UART Bridge Controller
[123860.331029] usb 6-1.1.4: Manufacturer: Silicon Labs
[123860.331033] usb 6-1.1.4: SerialNumber: 0001
[123860.332092] cp210x 6-1.1.4:1.0: cp210x converter detected
[123860.420726] usb 6-1.1.4: reset full-speed USB device number 8 using ehci-pci
[123860.529943] usb 6-1.1.4: cp210x converter now attached to ttyUSB0
```

Figura 4.10: Reconhecimento do conversor USB-UART/TTL pelo sistema operacional

Os conector da porta USART do Motorino foi montado de forma a ser compatível pino-a-pino com o adaptador USB-UART/TTL escolhido como mostrado na Figura 4.11.

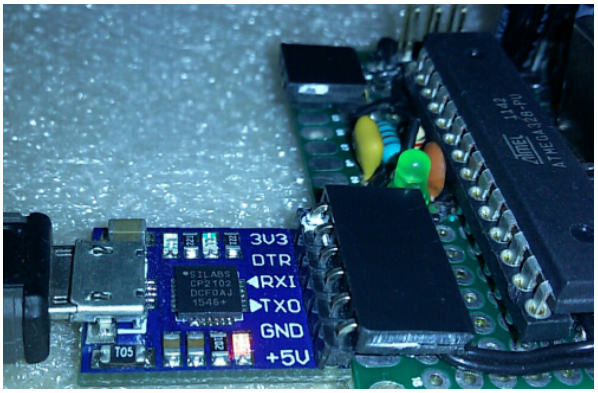


Figura 4.11: Adaptador USB para UART/TTL baseado no CI CP2102

Além da comunicação serial, este adaptador fornece alimentação de 5V para que ocorra o processo de gravação sem a necessidade do Motorino estar conectado à alimentação. Além disso, o pino *DTR (Data Terminal Ready)* é capaz de fornecer um pulso negativo no início da transmissão de dados do processo de gravação da aplicação via IDE. O *hardware* do Motorino reconhece este pulso e faz o processo de *reset* do microcontrolador, desta forma o código do *bootloader* é executado e inicia-se o processo de gravação da aplicação na memória *flash* interna.

4.6 Síntese do Protótipo

O desenvolvimento do protótipo foi composto de três partes: O projeto e montagem do *hardware* de acordo com as funcionalidades pretendidas, o desenvolvimento de uma biblioteca que fornece métodos para o controle de motores DC e a integração na IDE do Arduino dos artefatos de *hardware* e *software* desenvolvidos.

O projeto do *hardware* teve início com a elaboração do diagrama em blocos que serviu de base para as conexões dos principais componentes: O microcontrolador ATmega328 e o CI L298 como ponte H. A escolha dos demais componentes eletrônicos teve influência nas recomendações dos fabricantes do microcontrolador e do CI de ponte H conforme descrito em seus *datasheets*. Também foram adicionados LEDs para sinalização, chave para o *reset* manual e conectores para a alimentação, motores e portas de comunicação do microcontrolador. Este processo foi documentado em forma de diagrama esquemático que serviu de guia para a montagem da placa eletrônica.

A etapa seguinte consistiu no desenvolvido uma biblioteca nos moldes da plataforma Arduino que tem por função inicializar os pinos e funcionalidades do microcontrolador e fornecer métodos para o controle lógico de motores DC. Esta biblioteca é capaz de controlar o comportamento de dois motores DC independentes alterando suas propriedades de potência e direção como também é capaz de medir a corrente que circula pelos motores. Para facilitar a integração com dispositivos gerenciadores externos conectados à placa, a biblioteca conta com um método que funciona como uma API, recebe comandos e retorna resultados por meio de strings estruturados de tamanho reduzido.

A biblioteca desenvolvida foi adicionada à IDE do Arduino obedecendo aos critérios de estrutura de pastas e arquivos de configuração de acordo com o padrão exigido para as bibliotecas personalizadas (de terceiros), o que permite que o código esteja disponível para inclusão nas aplicações desenvolvidas na IDE.

O processo de integração *hardware* também precisou seguir critérios de estruturas de pastas e arquivos de configuração previstos no processo de customização da IDE. Os elementos de *software* da integração, exclusivos da placa foram mantidos isolados, com versionamento próprio, enquanto as bibliotecas *core* e *padrão* do Arduino são de uso compartilhado.

Com a placa integrada, foi possível selecioná-la e utilizar os procedimentos previstos na IDE para a gravação do *bootloader* na memória *flash* do microcontrolador, utilizando para tal tarefa um gravador *avr-isp*.

Ao final destas três etapas, foi obtido um *hardware* funcional e integrado, onde é possível produzir e gravar aplicações da mesma forma como é feito com as placas oficiais do Arduino.

4.7 Considerações

Este capítulo mostrou todas as etapas necessárias para a construção do Motorino, o projeto e montagem do *hardware*, escolhas de projeto e estrutura da biblioteca que dá funcionalidade à placa e os processos de integração de *hardware* e *software* na IDE para que seja possível trabalhar com o Motorino da mesma forma como é feito com as placas oficiais do Arduino.

O próximo capítulo apresenta testes e validação feitos com o Motorino e análise dos resultados obtidos com o objetivo de validar as funcionalidades propostas para este trabalho.

Capítulo 5

Testes e Resultados

Este capítulo apresenta testes qualitativos e quantitativos realizados com o protótipo montado e biblioteca desenvolvida visando verificar o atendimento das funcionalidades propostas para o trabalho, sendo feita uma análise dos resultados obtidos em cada teste.

5.1 Funcionalidades da biblioteca

Os testes apresentados nesta seção têm por objetivo verificar e validar os métodos usados para controle lógico sobre a carga bem como a capacidade de receber comandos através de um dispositivo externo. A placa foi alimentada por uma fonte de 12Vcc e manteve-se conectada na porta USB do PC através da UART com o uso de um adaptador USB-UART/TTL conforme a Figura 4.11. O algoritmo base em execução no microcontrolador usado nos testes foi codificado conforme o Algoritmo 1.

Algoritmo 1 Aplicação base para realização de testes

```
1: buffer_Rx ← FALSE
2: while TRUE do
3:     if Serial.available() then
4:         buffer_Rx ← Serial.read()
5:     end if
6:     if buffer_Rx then
7:         Serial.write(buffer_Rx)
8:         tempo ← micros()
9:         buffer_Tx ← Motorino.cmdExec(buffer_Rx)
10:        tempo ← micros() − tempo
11:        Serial.write(buffer_Tx)
12:        Serial.write(tempo)
13:        buffer_Rx ← FALSE
14:    end if
15: end while
```

5.1.1 Execução de comandos via comunicação serial

Neste teste os comandos foram enviados à placa via *Serial Monitor* da IDE do Arduino e foram executadas na placa via método *cmdExec()*. A Figura 5.1 mostra os comandos enviados e respectivos retornos conforme descritos na Seção 4.2.

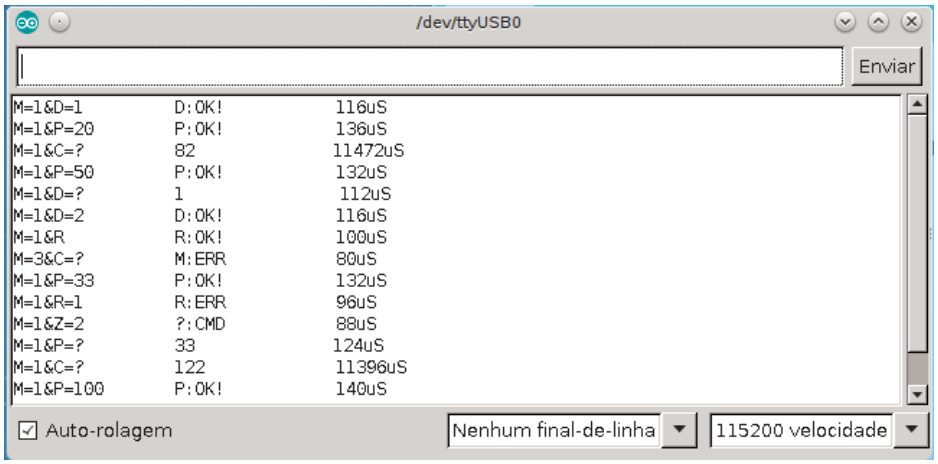


Figura 5.1: Comandos enviados via *Serial Monitor* e respectivos retornos

Foram usadas como cargas duas lâmpadas incandescentes de 12V polarizadas inversamente uma em relação a outra através de diodos de forma que somente uma acenda conforme a polaridade da ponte H, desta forma é possível fazer uma análise qualitativa de forma visual dos efeitos provocados pela execução dos comandos sobre a carga conforme mostram as imagens da Figura 5.2.

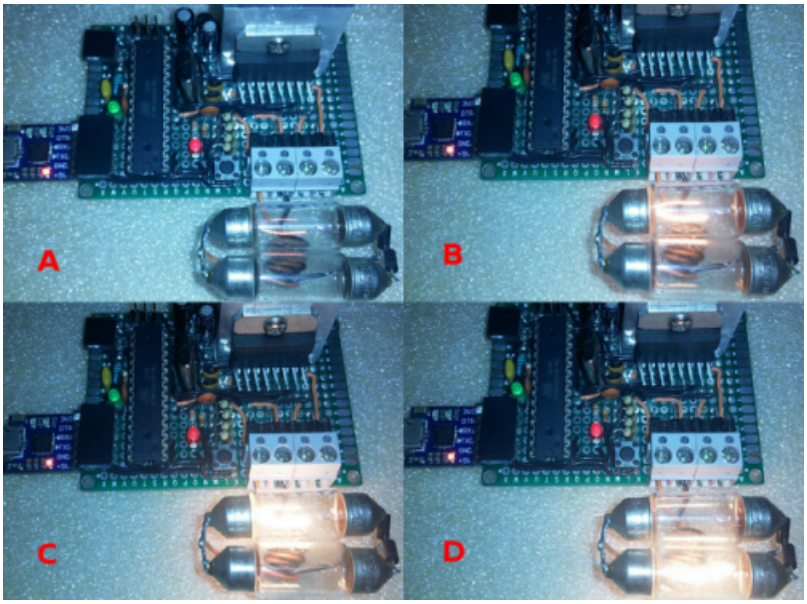


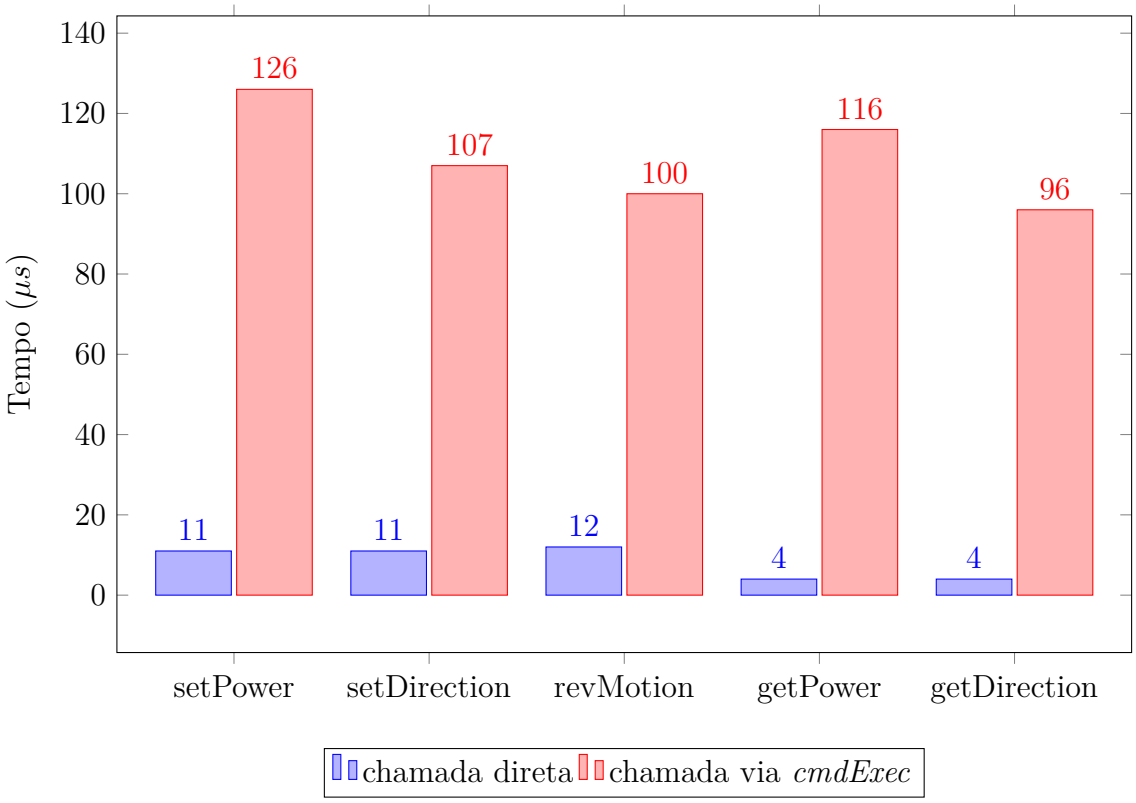
Figura 5.2: Visualização dos efeitos dos comandos enviados via *Serial Monitor*

Nas imagens *A, B e C* da Figura 5.2 foram enviados comandos para alterar a potência sobre a carga e na imagem *D* foi feita a reversão da polaridade da ponte H.

5.1.2 Tempo de execução dos métodos da biblioteca

Para este teste foi adicionada ao Algoritmo 1 a execução repetitiva das chamadas dos métodos públicos que compõem a classe Motorino, descritas na Seção 4.2, de modo autônomo, sem a necessidade de esperar pelos comandos via *Serial Monitor*.
Para obtenção dos tempos de execução, foi usada a função *micros()* do *core* do Arduino e calculada a média aritmética de mil chamadas de cada método público de forma direta e também por intermédio do método *cmdExec*.
Os tempos medidos foram dispostos conforme o gráfico da Figura 5.3.

Figura 5.3: Tempos de execução dos métodos da biblioteca



O método *readCurrent()* não consta no gráfico da Figura 5.3 pois foram obtidos os tempos médios de 11276μs para chamada direta e 11382μs para chamada via método *cmdExec*, divergindo em torno de mil vezes da ordem de grandeza das demais medidas, fato discutido na Seção 5.1.3, a seguir.

5.1.3 Análise dos resultados

O teste de execução remota, via comunicação serial, foi realizado com natureza qualitativa, foram testadas todas as possibilidades de comandos válidos e inválidos conforme listados na seção 4.2 e tabela 4.4. O método se mostrou perfeitamente funcional, executando os comandos e retornando os resultados exatamente como esperado.

O teste de tempo de execução dos métodos da biblioteca com chamada direta e chamada via método *cmdExec()* foi feito com o propósito de identificar atrasos nos comandos enviados de forma remota que venham causar prejuízo na responsividade quando em regime de dispositivo escravo. Houve um aumento significativo no tempo de resposta dos comandos executados via *cmdExec()* devido ao processo de tratamento de strings que o método implementa para interpretar o comando e selecionar a ação correspondente, no entanto o tempo médio de 109 microssegundos indica que é possível executar mais de 9000 comandos por segundo, podendo ser considerado irrelevante perante o tempo de resposta da grande maioria de dispositivos eletromecânicos.
Uma medida de tempo discrepante ocorreu por conta do método *readCurrent()* que apresentou um tempo de resposta médio de aproximadamente $11300\mu s$. O algoritmo do método *readCurrent()* faz 100 medidas com função *analogRead()* do *core* do Arduino, considera as que retornaram valor maior que zero e faz uma média aritmética para se obter o valor de tensão lido que é usado no cálculo da corrente. Este procedimento foi adotado pois o *hardware* do PWM trabalha de modo independente e não há como detectar por *software* o período do ciclo ativo do PWM para disparar o processo de medida. Este problema foi incluso nos trabalhos futuros para que seja feita uma pesquisa de alternativas mais eficientes para o processo de medição de corrente.

5.2 Controle PWM da Ponte H e efeito sobre a carga

Os testes realizados nesta seção tem por objetivo validar o método de modulação por largura de pulso - PWM - gerada pelo microcontrolador e aplicada nos pinos de habilitação da ponte H para que a mesma exerça o controle de potência sobre a carga.
A placa foi alimentada por uma fonte de 12Vcc e usado como carga uma resistência de 10Ω (tolerância de 5%) para que se obtenha uma interação elétrica linear facilitando assim a análise dos dados obtidos. A placa manteve-se conectada ao PC via USB, o controle dos parâmetros de funcionamento da placa foram feitos por comandos enviados através do *Serial Monitor* da IDE do Arduino e a aplicação para os testes executada na placa foi codificada de acordo com o Algoritmo 1.

5.2.1 Frequência de funcionamento

Este teste consiste na verificação da frequência de funcionamento do PWM onde, devido ao uso da função *analogWrite*() do Arduino, o modo e frequência do PWM são configurados via inicialização da biblioteca *core*.
O período e a frequência de funcionamento do PWM foram medidos com o uso de um osciloscópio. A Figura 5.4 mostra um período de onda de $2.04ms$ que implica numa frequência de $490.20Hz$ conforme destacado pela área em vermelho.

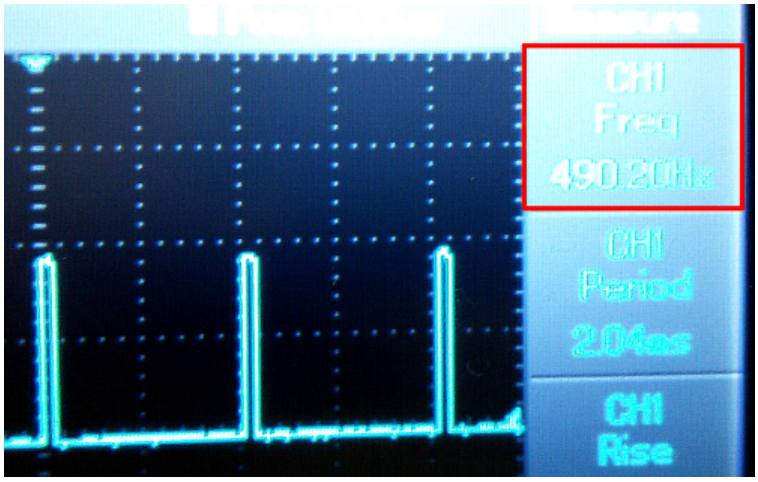
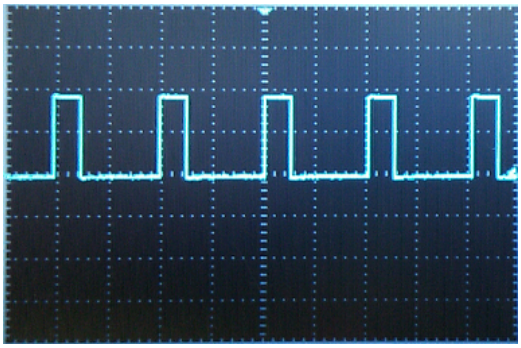


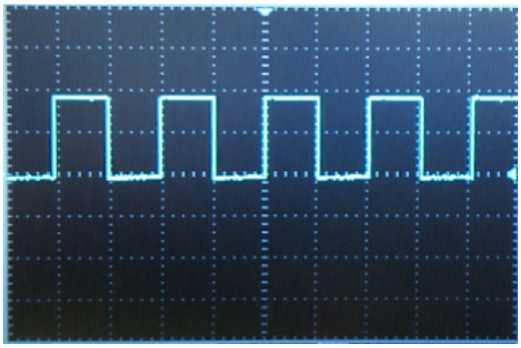
Figura 5.4: Frequência do PWM gerado pelo microcontrolador

5.2.2 Forma de onda e tensão média

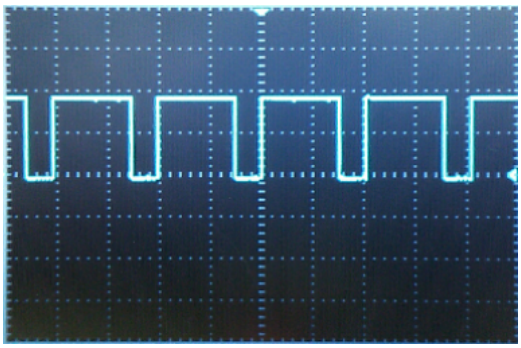
Neste teste são analisados a forma de onda e a tensão média sobre a carga em função da variação do *duty cycle* com o uso de um osciloscópio. Também é levada em consideração a influência dos *resistores sensores* nas características elétricas da carga controlada. As imagens da Figura 5.5 ilustram as formas de onda observadas sobre a carga para quatro valores distintos do *duty cycle*.



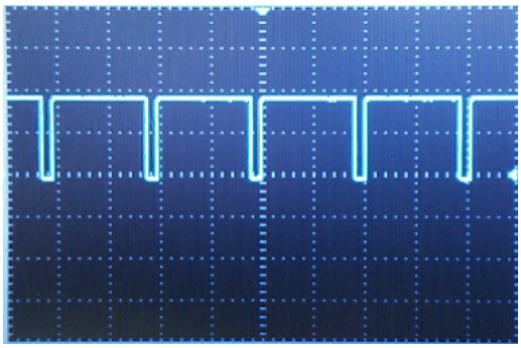
(a) Duty Cycle de 25%



(b) Duty Cycle de 50%



(c) Duty Cycle de 75%



(d) Duty Cycle de 90%

Figura 5.5: Forma de onda sobre a carga com variação de *duty cycle*

O procedimento para aquisição de dados para a análise foi baseado na alteração gradual do *duty cycle*, partindo de 0% até 100% em intervalos de 5% e feitas as respectivas medidas da tensão média sobre a carga. A determinação da corrente que circula pela carga foi feita de modo indireto através da *Lei de Ohm* conforme Equação 5.1.

$$I_i(mA) = \frac{Vi}{R_{carga}} \times 10^3 \tag{5.1}$$

Onde:
I_i é a corrente estimada, em miliampères;
V_i é a tensão medida sobre a carga, em volts;
R_{carga} é a resistência da carga, em ohms;

O resistor de carga utilizado tem precisão de 5% e a sua medição indicou 9.8Ω, valor este utilizado nos cálculos. A determinação de tensão sobre os *resistores sensores*, de 0.5Ω, também foi feita de forma indireta, isolando V_i e substituindo R_{carga} na Equação 5.1.
Os dados coletados e calculados para análise foram agrupados na Tabela 5.1.

Tabela 5.1: Tensão média na carga e resistores sensores em função do *duty cycle*

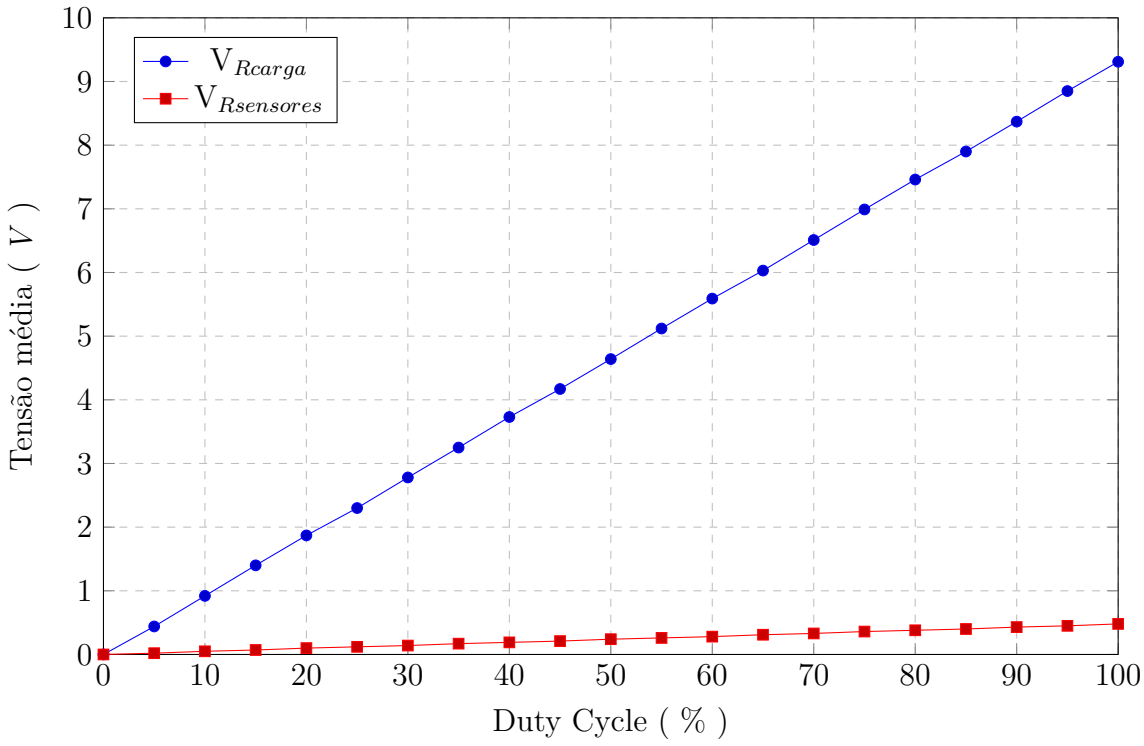
Duty Cycle (%)	$\mathbf{I}_{Rcarga}$ **(mA)**	$\mathbf{V}_{Rcarga}$	$\mathbf{V}_{Rsensores}$
0	0	0.00	0.00
5	45	0.44	0.02
10	94	0.92	0.05
15	143	1.40	0.07
20	191	1.87	0.10
25	235	2.30	0.12
30	284	2.78	0.14
35	332	3.25	0.17
40	381	3.73	0.19
45	426	4.17	0.21
50	474	4.64	0.24
55	522	5.12	0.26
60	570	5.59	0.28
65	615	6.03	0.31
70	664	6.51	0.33
75	713	6.99	0.36
80	761	7.46	0.38
85	806	7.90	0.40
90	854	8.37	0.43
95	903	8.85	0.45
100	950	9.31	0.48

5.2.3 Análise dos resultados

O teste de medição da frequência do PWM é trivial já que o uso da função *analogWrite()* do Arduino para configuração e geração do PWM implica nas configurações pré-definidas de modo de funcionamento e divisor do contador do respectivo *timer* conforme consta no método *init()* do arquivo fonte *wiring.c* do *core* do Arduino. O Motorino usa os pinos 15 e 16 do ATmega328 para controle PWM, ambos controlados pelo *Timer1*.
No código fonte do *wiring.c* existe um comentário justificando a configuração de funcionamento do *Timer1* no modo *Phase-Correct PWM* como sendo mais adequado para o controle de motores. É feito o ajuste do contador para trabalhar com 8 bits e o divisor com o valor *64*. Sabendo que o *clock* do microcontrolador é de $16Mhz$ e aplicando a Equação 3.2 obtemos o valor de $490.20Hz$, exatamente como medido.

No testes de forma de onda e tensão média sobre a carga, durante o processo de variação do *duty cycle* notou-se que a forma variou proporcionalmente ao ciclo ativo mantendo sua forma e período estáveis, constatando-se assim que com a frequência usada, o circuito integrado L298N é capaz de transferir à carga os sinais de controle recebidos do microcontrolador de forma eficiente.
Os dados agrupados da Tabela 5.1 referentes aos valores de tensão média sobre a carga e de tensão média sobre os resistores sensores foram plotados em função do *duty clycle* para uma análise visual de comportamento, mostrados no gráfico da Figura 5.6.

Figura 5.6: Tensão média em função do *duty cycle*



As curvas do gráfico da Figura 5.6 mostram uma relação linear e diretamente proporcional da tensão média em função do *duty cycle* agindo sobre o resistor de carga e resistores sensores, exatamente como esperado de um sistema de controle de potência do método PWM. No entanto nota-se que a tensão máxima sobre a carga não corresponde à tensão de 12Vcc fornecida pela fonte. Conforme pode ser percebido através das figuras 3.21 e 3.24, os resistores sensores, a carga e os transístores internos da ponte H formam um circuito série onde cada um destes elementos retém uma parcela da tensão da fonte, de modo que na condição de máxima tensão média (*duty cycle* = 100%) temos:

$$\begin{aligned} V_{Fonte} &= V_{Ponte_H} + V_{Rcarga} + V_{Rsensores} \\ 12 &= V_{Ponte_H} + 9.31 + 0.48 \\ \implies V_{Ponte_H} &= 2.21\,Volts \end{aligned}$$

Assim, nas condições deste teste temos a tensão de $2.21\,V$ sobre a ponte H e $0.48\,V$ sobre os resistores sensores, condição que diminui em aproximadamente 22% a transferência de potência para a carga. De acordo com o *datasheet* do CI L298N [19], os transístores da ponte H podem reter entre $1.8\,V$ e $3.2\,V$ quando saturados (condição de condução total de corrente) e conduzindo uma corrente de $1A$, dessa forma o componente apresenta funcionamento em acordo com as especificações do fabricante.
Caso esta perda de potência for crítica para o funcionamento da carga existem duas maneiras de minimizar o problema:
1) Curto-circuitar os resistores sensores caso não seja necessário usar o monitoramento embarcado de corrente sobre a carga. Este procedimento não traz prejuízo ao circuito mas proporciona um baixo ganho: Em torno de 4% conforme condições deste teste.
2) Aumentar a tensão de alimentação do circuito, dessa forma a queda de tensão sobre a ponte H e sobre os resistores sensores ainda permite obter a tensão nominal sobre a carga. A tensão adicional da alimentação deve ser previamente estimada para que a tensão máxima sobre a carga não ultrapasse seu limite especificado de funcionamento.

5.3 Medição embarcada de corrente da carga

Os testes realizados nesta seção têm por objetivo validar o processo de medição de corrente na carga por meio de comparação entre os valores retornados pelo método *readCurrent()* e os valores teóricos calculados, ambos em função da variação do *duty cycle*. As conexões com a placa são idênticas às da Seção 5.2: Alimentação por fonte de 12Vcc, a carga constitui de um resistor de 10Ω com tolerância de 5% (9.8Ω medidos) e comandos enviados através do *Serial Monitor* da IDE do Arduino. A aplicação para os testes em execução na placa foi codificada de acordo com o Algoritmo 1.

5.3.1 Obtenção dos dados

Os valores de retorno do método *readCurrent()* foram obtidos mediante alteração do *duty cycle* de 0% a 100% em intervalos de 5%. Como descrito na Seção 4.1, a corrente que circula pelos *resistores sensores* é a mesma que circula pela carga, então foram utilizados os valores de corrente na carga da Tabela 5.1 e todos os dados agrupados na Tabela 5.2.

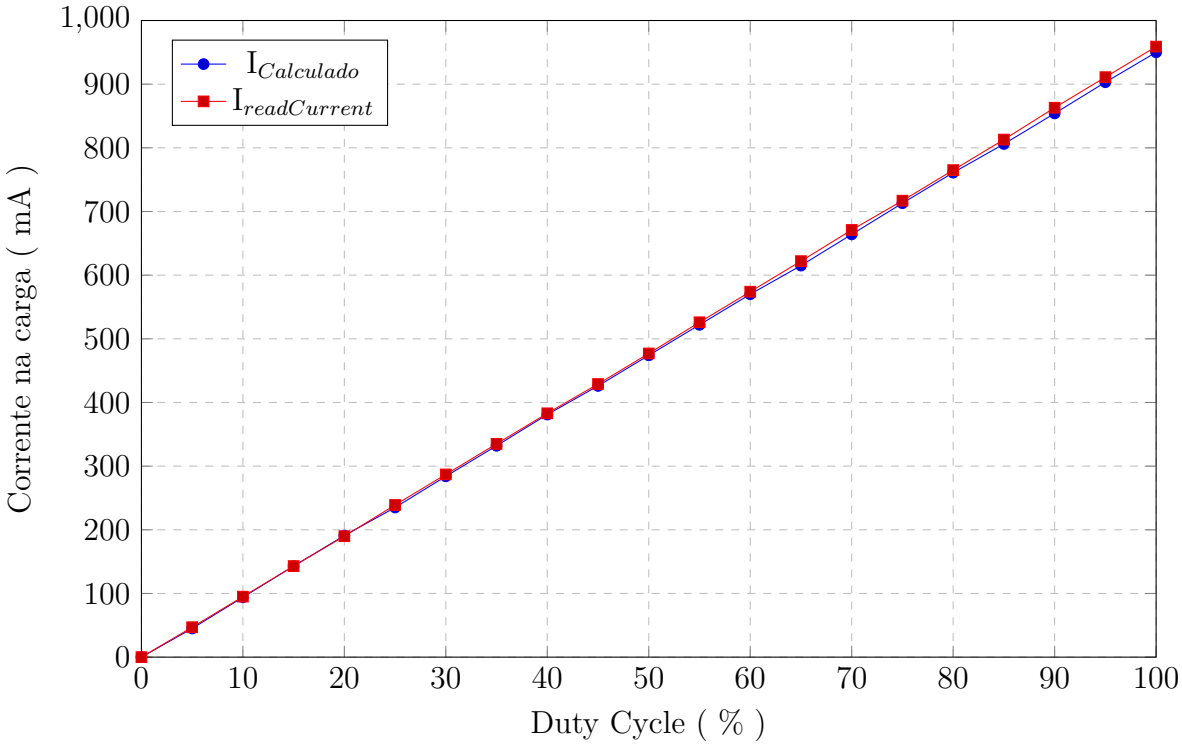
Tabela 5.2: Tensão e corrente na carga em função do duty cycle

Duty Cycle (%)	$\mathbf{I}_{Calculado}$ **(mA)**	$\mathbf{I}_{readCurrent}$ **(mA)**
0	0	0
5	45	47
10	94	95
15	143	143
20	191	190
25	235	239
30	284	287
35	332	335
40	381	383
45	426	429
50	474	477
55	522	526
60	570	574
65	615	622
70	664	671
75	713	717
80	761	765
85	806	813
90	854	863
95	903	911
100	950	959

5.3.2 Análise dos resultados

Os dados referentes aos valores de corrente retornadas pelo método *readCurrent()* e os valores de corrente calculados conforme constam na Tabela 5.2 foram postos em função do *duty cycle* conforme mostra o gráfico da Figura 5.7. Desta forma é possível visualizar o comportamento da variação de corrente sobre a carga e comparar possíveis discrepâncias entre as duas curvas.

Figura 5.7: Corrente na carga em função do *duty cycle*



As duas curvas do gráfico da Figura 5.7 mostram um comportamento linear, esperado de cargas resistivas, indicando assim que os métodos de controle nem o circuito da placa não introduzem variações indesejadas na alimentação fornecida à carga controlada e a sobreposição das curvas indica a coerência dos valores retornados pelo método *readCurrent()* frente aos valores calculados baseados em medições feitas com um equipamento profissional. Nota-se tanto pela tabela quanto pelo gráfico alguns valores não coincidentes. Essas diferenças podem ter sido ocasionadas pela propagação de erros inerentes dos processos de obtenção dos valores, conforme listados:

Corrente calculada: Precisão do voltímetro, precisão do ohmímetro, arredondamento de valores fracionários obtidos nos cálculos;

Corrente retornada pelo método readCurrent(): Considerado o valor nominal dos resistores sensores (foram usados resistores com tolerância de 5%), erros de quantização dos ADCs (devido a resolução), cálculo da corrente utilizando números de ponto flutuante, arredondamento de valores fracionários obtidos nos cálculos;

5.4 Considerações

Neste capítulo foram apresentados uma sequência de testes realizados para verificar se o protótipo de *software* e *hardware* desenvolvidos atendem as propostas do trabalho. Para cada teste realizado foi apresentado o objetivo, método de aquisição de dados e posteriormente feita uma análise dos dados coletados, validando a proposta de desenvolvimento.

O próximo capítulo apresenta uma discussão sobre o desenvolvimento e resultados dos testes como também propostas de trabalhos futuros que visam melhorar e ampliar as funcionalidades baseadas nas experiências adquiridas ao longo do processo do desenvolvimento e testes deste trabalho.

Capítulo 6

Discussão e Conclusão

Neste capítulo são discutidos os resultados alcançados com o trabalho, como também propostas de melhorias de *hardware* e demandas de *software*.

6.1 Pareceres sobre o trabalho

Neste trabalho foi desenvolvido um módulo para controle de cargas que pode operar em regime autônomo ou controlado remotamente com foco no controle de motores DC e baseado na plataforma de *hardware* e *software* livres do Arduino.

Os testes feitos demonstram que API desenvolvida foi capaz de funcionar com eficiência, executando comandos e retornando resultados por meio de uma *interface* remota exatamente da forma que ela foi projetada.

Na execução autônoma, os métodos públicos tiveram médias de tempo de 8.4μs para chamada direta e 103μs para chamadas via API. Mesmo com o aumento significativo de tempo imposto pela API é possível executar mais de 9000 comandos por segundo via API, não sendo um tempo crítico para o controle da grande maioria de atuadores eletromecânicos. Uma medida discrepante ficou por conta do método embarcado de leitura de corrente na carga que apresentou tempo médio de 11.3ms sendo listado como trabalho futuro o estudo de alternativas que contribuam para melhorar seu tempo de resposta.

O circuito também se mostrou capaz de reproduzir com eficiência sobre a carga o controle PWM fornecido pelo microcontrolador. Nas condições de teste foi notada uma perda de aproximadamente 22% na transferência de potência da fonte para a carga, porém este valor pode ser amortizado aumentando de forma sistemática a tensão de alimentação da placa para que a queda de tensão sobre a ponte H tenha menor relevância e o método de medição de corrente embarcada na placa pode ser considerado válido pois mostrou uma boa precisão nos valores de corrente medidos.

Em vista dos resultados obtidos, conclui-se que o protótipo desenvolvido foi capaz de cumprir com os objetivos propostos para este trabalho.

Além do controle de motores DC, uma placa especializada em controlar cargas DC de baixa tensão e integrada na plataforma do Arduino, mantém os propósitos didáticos da plataforma e devido a redução de tamanho (não precisa de *shields* ou módulo atuador de potência) tem potencial para ser uma solução de baixo custo, definitiva e integrada a sistemas mais complexos, como acionamento de cargas via *IoT*, controle de estabilidade para *drones*, controle de motores para impressoras 3D, controle de acesso em portões com trava elétrica, controle para iluminação com LEDs, dentre muitos outros.

6.2 Trabalhos Futuros

Diante dos resultados obtidos e da experiência adquirida durante o processo de desenvolvimento, ficam em aberto algumas possibilidades de trabalhos futuros que visam ampliar e melhorar as funcionalidades da plataforma construída.
A seguir, uma lista de melhorias que podem ser feitas para o *hardware* e para o *software* do Motorino:

6.2.1 Melhorias no hardware

- Inserção de conectores para o uso de pinos de I/O ociosos para agregação de sensores, atuadores, sinalizadores, displays e afins;
- Adicionar resistores de *pull-up* nas linhas *SCL* e *SDA* do barramento I^2C.
- Projeto de uma placa de circuito impresso para padronizar a reprodução do hardware;
- Uso de componentes SMD (*Surface Mounting Device*) para diminuir o tamanho da placa melhorando sua integração física a outros sistemas;
- Adição de *hardware* externo à MCU para sincronização dos pinos de *enable* do CI L298N para utilização de controle PWM com ambas pontes H em paralelo;
- Melhorar a precisão na medida da corrente nos motores com resistores de precisão de 1%;

6.2.2 Melhorias no software

- Especialização da biblioteca em versões distintas para o dispositivo controlador (*master*) e dispositivos controlados (*slaves*);
- Adicionar suporte para motores de passo de duas fases;
- Adicionar suporte a servo motores;
- Adicionar suporte a *encoders* e chaves de fim de curso;
- Melhora do método de medição de corrente nas Pontes H;
- Adicionar controle simultâneo de ambas pontes H do CI L298N;
- Integrar a placa do Motorino à IDE do Arduino via gerenciador de placas;
- Integrar biblioteca do Motorino à IDE do Arduino via sistema gerenciador;
- Atualizar o makefile do *bootloader* para versões mais recentes do compilador *avr-gcc*;

Referências Bibliográficas

[1] Site: 8Bit-Homecomputermuseum. Placa principal do computador NewBrain. http://www.8bit-homecomputermuseum.at/repair/newbrain/small/NewBrain_mainboard_explain.jpg. Online; acessado em Novembro de 2016.

[2] Arduino. Arduino Products. https://www.arduino.cc/en/Main/Products.

[3] Atmel. Datasheet MCU ATmega328. http://www.atmel.com/Images/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_datasheet.pdf. Online; acessado em Novembro de 2016.

[4] J.M. Sosa; S. León; J.M.Cerezo; A.Vega. Reuse of 3d printers for laboratory training system control and automation using microcontrollers and plcs. *2016 Technologies Applied to Electronics Teaching (TAEE)*, 2016.

[5] Massimo Banzi. *Getting Started with Arduino*. Make Books, 2 edition, 2011.

[6] Hernando Barragán. The Untold History of Arduino. https://arduinohistory.github.io.

[7] Site: Instituto Newton C. Braga. O que é PWM. http://www.newtoncbraga.com.br/index.php/robotica/5169-mec071a. Online; acessado em Novembro de 2016.

[8] A. C. Newell D.P. Siewiorek, G. Bell. *PComputer Structures: principles and examples*. McGraw-hill, 1982.

[9] Texas Instruments. Datasheet CI L293. http://www.ti.com/lit/ds/symlink/l293.pdf. Online; acessado em Novembro de 2016.

[10] Texas Instruments. Datasheet CI LMD18200. http://www.ti.com/lit/ds/symlink/lmd18200.pdf. Online; acessado em Novembro de 2016.

[11] Texas Instruments. Datasheet CI MAX232. http://www.ti.com/lit/ds/symlink/max232.pdf. Online; acessado em Novembro de 2016.

[12] Harald Molle John-David Warren, Josh Adams. *Arduino Robotics*. Apress, 1 edition, 2011.

[13] Lukas Müller; Masihuddin Mohammed; Jonathan W. Kimball. Using the arduino uno to teach digital control of power electronics. *2015 IEEE 16th Workshop on Control and Modeling for Power Electronics (COMPEL)*, 2015.

[14] Silicon Labs. Datasheet CI CP2102. `https://www.silabs.com/Support%20Documents/TechnicalDocs/CP2102-9.pdf`. Online; acessado em Novembro de 2016.

[15] A. M. S. Laurent. *Understanding open source and free software licensing*. O'Reilly Media, 2004.

[16] Eduardo J. Moya; Víctor Cervero; David López. Full building of a sun tracker and control. *2015 23rd Mediterranean Conference on Control and Automation (MED)*, 2015.

[17] Anil K. Maini. *Digital Electronics: Principles, Devices and Applications*. Wiley, 1 edition, 2007.

[18] Microchip. Datasheet PIC18F2431. `http://ww1.microchip.com/downloads/en/DeviceDoc/39616d.pdf`. Online; acessado em Novembro de 2016.

[19] ST Microelectronics. Datasheet CI L298. `http://www.st.com/resource/en/datasheet/l298.pdf`. Online; acessado em Novembro de 2016.

[20] Allegro MicroSystems. Datasheet CI A4988. `https://www.pololu.com/file/download/a4988_DMOS_microstepping_driver_with_translator.pdf?file_id=0J450`. Online; acessado em Novembro de 2016.

[21] J. Noble. *Programming Interactivity: A Designer's Guide to Processing, Arduino, and Operiframeworks*. O'Reilly Media, 2009.

[22] Larry O'Cull Richard H. Barnett, Sarah Cox. *Embedded C Programming and the Atmel AVR*. Thomson, Delmar Learning, 2 edition, 2007.

[23] Jeeva B.; Swapnil A.; Rajesh J.; Arindrajit C.; Sreedhara S. Development of custom-made engine control unit for a research engine. *2014 2nd International Conference on Emerging Technology Trends in Electronics, Communication and Networking*, 2014.

[24] Fairchild Semiconductors. Datasheet CI 74164. `https://www.fairchildsemi.com/datasheets/74/74VHC164.pdf`. Online; acessado em Novembro de 2016.

[25] NXP Semiconductors. Especificação oficial I2C. http://www.nxp.com/documents/user_manual/UM10204.pdf. Online; acessado em Novembro de 2016.

[26] NXP Semiconductors. Serial Bus Overview. http://www.nxp.com/documents/application_note/AN10216.pdf.

[27] Vincent Sieben. *A High Power H-Bridge*. Canada Revision, 1 edition, 2003.

[28] Timothy L. Skvarenina. *The Power Electronics Handbook*. CRC Press LLC, 1 edition, 2002.

[29] Site: Sparkfun. Serial Communication. https://learn.sparkfun.com/tutorials/serial-communication. Online; acessado em Novembro de 2016.

[30] Site: Sparkfun. Serial Peripheral Interface. https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi. Online; acessado em Novembro de 2016.

[31] Site: IEEE Spectrum. The Making of Arduino. http://spectrum.ieee.org/geek-life/hands-on/the-making-of-arduino. Online; acessado em Novembro de 2016.

[32] STMicroelectronics. Datasheet CI L78L series. http://www.st.com/resource/en/datasheet/l78l.pdf. Online; acessado em Novembro de 2016.

[33] C. Thompson. Build it. share it. profit. can open source hardware work. *Wired Magazine, vol. 16, no. 11, pp. 16-11*, 2008.

[34] Alan Trevennor. *Pratical AVR Microcontrollers*. Apress, 1 edition, 2012.

[35] J.S. George; J.C. Johnson; G. Senthilkumaran; B.P.C. Rao; S. Venugopal. Obstacle avoidance and orientation determination for a remote operated vehicle. *International Conference on Magnetics, Machines & Drives*, 2014.

[36] Allan G. Weber. Using the i2c interface on the atmega328p and mc908jl16. *USC Viterbi School of Engeneering*, 2015.

[37] Wikipedia. Grundy NewBrain. https://en.wikipedia.org/wiki/Grundy_NewBrain. Online; acessado em Novembro de 2016.

[38] Wikipedia. Intel 8080. https://en.wikipedia.org/wiki/Intel_8080. Online; acessado em Novembro de 2016.

[39] Wikipedia. Zilog Z80. https://en.wikipedia.org/wiki/Zilog_Z80. Online; acessado em Novembro de 2016.