ÁREA DE INFORMÁTICA

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - ADS

HUGO DE OLIVEIRA

BIBLIOTECA DE COMUNICAÇÃO SERIAL PARA ARDUINO

TRABALHO DE CONCLUSÃO DE CURSO - TCC

CARAGUATATUBA

2015

BIBLIOTECA DE COMUNICAÇÃO SERIAL PARA ARDUINO

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de Tecnólogo da Área de Informática, do Instituto Federal de São Paulo.

Orientador:
Prof. Dr. Ederson Rafael Wagner.

CARAGUATATUBA

2015

TERMO DE APROVAÇÃO

BIBLIOTECA DE COMUNICAÇÃO SERIAL PARA ARDUINO

por

HUGO DE OLIVEIRA

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 25 de fevereiro de 2015 como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas (ADS). O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados, a qual após deliberação considerou o trabalho aprovado.

_____

Prof. Ederson Rafael Wagner

Orientador

_____

Prof. Eduardo Pereira de Sousa

Presidente

_____

Prof. Wanderson Santiago dos Reis

Membro

“Nós não temos a chance de fazer muitas coisas, e cada uma deve ser realmente excelente. Porque esta é a nossa vida. A vida é breve, e então você morre, sabe? E todos nós escolhemos o que fazer com as nossas vidas. Então é melhor que seja muito bom. É melhor valer a pena”.

Steve Jobs

Dedico este trabalho a minha família pelo apoio diario.

AGRADECIMENTOS

Agradeço primeiramente a Deus pela força e determinação que tens me dado.

Ao meu orientador Prof. Dr. Ederson pela sabedoria com que me guiou nesta trajetoria.

Gostaria de deixar registrado também o meu reconhecimento a minha família, pois acredito que sem o apoio deles seria muito difícil vencer este desafio.

Aos meus professores que me incentivaram a continuar buscando conhecimento, orientando sempre a buscar informações em melhores acervos, me ensinando principalmente a me virar.

Enfim, a todos que por algum motivo contribuíram para a realização desta pesquisa.

Certamente estes parágrafos não irão atender a todas as pessoas que fizeram parte dessa importante fase da minha vida. Portanto, desde já peço desculpas àquelas que não estão presentes entre estas palavras, mas que sempre farão parte do meu pensamento e da minha gratidão.

RESUMO

Arduino é uma plataforma de prototipagem eletrônica criada com o objetivo de permitir o desenvolvimento de controle de sistemas interativos, de baixo custo e acessível a todos. Todo seu material (*software*, bibliotecas e *hardware*) é *open source* (código aberto), ou seja, conserva se a filosofia de usar e compartilhar por todos sem restrições. Sua plataforma é composta essencialmente de duas partes: o *Hardware* e o *Software*. Este trabalho visa a elaboração de uma biblioteca de comunicação entre a placa micro controladora "Arduino Uno" com dispositivos que possuam comunicação serial e que através de um protocolo mapeia a entrada e saída de dados. Para elaboração deste projeto realizou se a análise de suas características, isto é, explicação dos principais métodos da linguagem nativa de desenvolvimento, metodologia de implementação e o detalhamento das funcionalidades. O trabalho foi de caráter descritivo e bibliográfico. Com o resultado satisfatório e o objetivo alcançado concluiu se que esta biblioteca poderá contribuir tanto para os desenvolvedores e pesquisadores das linguagens abordadas quanto para os usuários que queiram estudar, implementar ou modificar esta comunicação, pois seu código fonte é de uso livre e todos poderão remanejá-lo de acordo com suas necessidades e/ou preferências.

Palavras-Chave: Biblioteca Livre. Protocolo. Comunicação Serial. Arduino.

ABSTRACT

Arduino is an electronic prototyping platform created in order to allow the development of interactive control systems, low-cost and accessible to all. All your stuff (software, libraries and hardware) is open source, ie retains the philosophy of being reproduced and used by everyone without restrictions. Its platform is essentially composed of two parts: the hardware and the software. This work aims at the development of a communication library between the micro controller board "Arduino Uno" with devices that have serial communication and through a protocol maps the input and output data. Due to lack of a free library, ready to use and make this type of communication, the opportunity to develop it. To prepare this project was carried out the analysis of its characteristics, that is, explanation of the main methods of the native language development, implementation methodology and the details of the features. The study was descriptive and bibliographic. With the satisfactory result and the goal achieved concluded that this library can contribute both to developers and researchers of the studied languages and for users who want to study, implement or modify this communication because its source code is free to use for everyone to redeploys it according to your needs and / or preferences.

Keywords: Free Library. Protocol. Serial Communication. Arduino.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

A/D *(Analógico-Digital)*

CPU (*Central Processing Unit*)

GND *(graduated neutral density filter)*

GPS *(Global Positioning System)*

IDE (*Integrated Development Environment*)

I/O (*Input/Output*)

ISR *(Interrupt Service Routine)*

KB (*Kilobyte*)

MHz *(Megahert)*

PC (*Personal Computer*)

PWM (*Pulse-Width Modulation*)

RX (*Receiver*)

SPI (*Serial Peripheral Interface*)

TX (*Transmitter*)

USB (*Universal Serial Bus*)

LISTA DE ACRÔNIMOS

BIT (*Binary digit*)

EEPROM (*Electrically Erasable Programmable Read Only Memory*)

LED (*Light Emitting Diode*)

OSI (*Open Source Initiative*)

RAM (*Random Access Memory*)

ROM (*Read-Only Memory*)

SRAM (*Static Random Access Memory*)

WI-FI (*Wireless Fidelity*)

IDE *(Integrated Development Environment)*

SUMÁRIO

1. INTRODUÇÃO

A plataforma Arduino está sendo cada vez mais utilizada nos meios comercial, residencial e acadêmico devido o grande avanço na área da informática e modernização da população. Seu uso é motivado principalmente pelo baixo preço e pela possibilidade de facilitar a vida de seus usuários.

Abrange diferentes campos de atuação como, por exemplo, a automação residencial que permite a unificação dos controles de dispositivos eletrônicos, de iluminação e de segurança em um único aparelho. A conexão pode ser realizada com qualquer dispositivo, como um *ethernet shield** ou mesmo um servidor conectado a internet para gerenciar dispositivos remotamente.

A comunicação serial é um dos principais meios de comunicação de dados utilizado entre Arduinos e outros dispositivos, é comum ver esse tipo de comunicação com diversos tipos de sensores e atuadores.

Sendo assim, despertou se o interesse no desenvolvimento de uma biblioteca dedicada que facilite sua manipulação de forma simples, clara e capaz de comunicar com outros dispositivos. De modo geral, será de grande valia para desenvolvedores, estudantes, micro empreendedores em suas startups ou até curiosos sobre o assunto.

A biblioteca e sua documentação será disponibilizada de forma gratuita, isto é, compartilhada se possível através do próprio site da fabricante e pelo acervo digital do IFSP – Campus Caraguatatuba.

2. OBJETIVO

Este trabalho tem por objetivo o desenvolvimento de uma biblioteca livre capaz de realizar a comunicação serial entre um Arduino Uno e outros dispositivos compatíveis de manipulação simplificada. Esta biblioteca deve possuir um protocolo com os principais métodos relacionados a comunicação serial e sua característica primordial é

*_Ethernet shield_ é um componente que permite que uma placa Arduino se conecte a internet ou rede local.

o mapeamento das portas de entrada e saída de dados da placa micro controladora Arduino Uno.

Com o mapeamento é possível realizar a escrita de uma informação ou de várias ao mesmo tempo em uma porta digital. Além da escrita é possível realizar a leitura de uma ou todas as portas analógicas.

Além da escrita individual das informações é possível realizar a leitura de todas as informações que estão passando pelo Arduino e montar um protocolo que contenha todas essas informações de uma forma mais estruturada.

3. EMBASAMENTO TEÓRICO

É muito importante uma introdução no assunto que englobe todos os conceitos chaves a respeito do funcionamento da placa micro controladora Arduino Uno e assim será possível compreender melhor como a biblioteca e o protocolo se comportarão diante das situações propostas.

3.1 O ARDUINO

Surgiu em 2005 na Itália por um professor chamado Massimo Banzi que desejava ensinar eletrônica e programação de computadores a seus alunos de design para que usassem em seus projetos de arte, interatividade e robótica. Porém, ensinar eletrônica e programação para quem não é da área é algo complicado e outro empecilho da época era a inexistência de placas controladoras baratas e poderosas (GRUPO DE ROBÓTICA – UFMS 2012).

Apesar das dificuldades encontradas, Massimo não desistiu e juntamente com David Cuartielles – engenheiro eletrônico espanhol em visita a Itália – decidiram projetar sua própria placa denominada de Arduino. Assim, escolheu um de seus alunos:

David Mellis que ficou responsável pela criação da linguagem de programação desta placa (GRUPO DE ROBÓTICA – UFMS 2012).

Posteriormente tornou se um grande sucesso, pois várias pessoas mesmo as que não sabiam programar, conseguiam fazer uso do Arduino para quaisquer finalidades, desde ler sensores e fazer luzes piscarem até controlar motores.

Segundo McRoberts (2011, p.22), “O Arduino é o que chamamos de plataforma de computação física ou embarcada, ou seja, um sistema que pode interagir com seu ambiente por meio de *hardware* e *software*.”

3.1.1 *Hardware*

Atualmente existem diversos modelos de placas Arduino, como por exemplo, “Arduino Mega”, “Arduino Leonardo”, “Arduino Nano”, “Arduino Ethernet” e entre outros, porém o modelo com o qual iremos utilizar será o modelo “Arduino Uno” por ser a opção mais didática e uma das mais acessíveis em termos financeiros.

O hardware do Arduino é simples e muito eficiente. A figura 1 representa a composição do hardware.

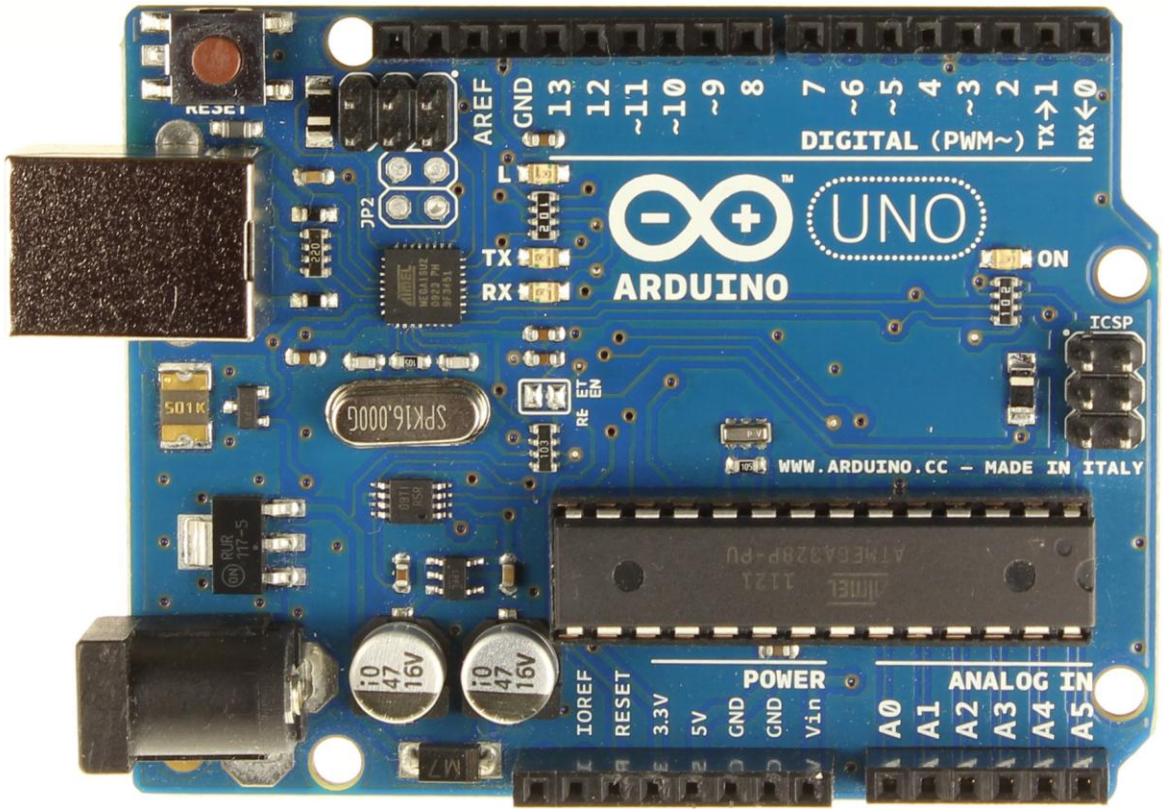


Figura 1: Placa Arduino Uno utilizada no desenvolvimento deste projeto.

Fonte: (adaptada de Arduino.cc, 2014)

Alimentação: pode ser através da conexão *USB* ou por *Power Jack* – fonte de energia externa. Se a energia fornecida for menos de 5 volts, a placa pode tornar se instável e se for fornecida mais do que 12 volts, o regulador de voltagem pode superaquecer e com isso danificar a placa. É recomendado que se utilize energia de 7 a 12 volts (ARDUINO, 2015). Há sete conectores de alimentação – *IOREF*; *RESET*; *VIN*; 5V; 3V3 e dois *GND* – *IOREF* é um pino que fornece tensão de referência para que os shields possam selecionar o tipo de interface; o *RESET* é um pino para reiniciar a placa do Arduino; *VIN* é a entrada de alimentação para a placa do Arduino quando se está utilizando uma fonte de alimentação externa que pode ser um *shield* ou bateria externa; o 5V é um pino que fornece alimentação de 5 volts para *shields* e outros circuitos externos, ou seja, é uma fonte de alimentação regulada usada para o micro controlador e para outros componentes da placa que pode vir tanto do *VIN* pelo regulador embarcado quanto da conexão *USB* ou quaisquer outras fontes reguladas em 5V; 3V3

é uma fonte de 3,3 volts gerada pelo regulador embarcado para alimentação de *shields* e módulos externos e suporta uma corrente máxima de 50mA; e os dois pinos *GND* são pinos terra (SOUZA, 2013).

ICSP – In Circuit Serial Programming: é um método de gravação que os dispositivos programáveis possuem e quer dizer que primeiro monta se o circuito na placa e depois programa se o dispositivo através de interface serial (ARDUINO, 2015).

Núcleo *CPU*: é encontrado na arquitetura do *hardware*. É um micro controlador ou mais precisamente, uma *CPU* de tamanho pequeno, mas muito importante. Possui memória *RAM*, memória programa – *ROM*, uma unidade de processamento de aritméticas e os dispositivos de entrada/saída, sendo praticamente um computador completo em um chip. Tal chip contém todo o *hardware* para obter, processar e devolver os dados vindos externamente. Dentro da sua *CPU* existe um *firmware* que carrega as instruções para realizar o funcionamento da placa. Além disso, seus desenvolvedores optaram em fazer uso da linha ATMega de micro computadores da empresa ATMEL (ERUS, 2012). A tabela 1 representa o resumo das principais características do Arduino Uno.

Micro controlador:	ATMega328
Tensão de operação:	5volts
Tensão de entrada:	(Recomendada): 7-12volts
Tensão de entrada (limites):	6-20volts
Pinos de entrada/saída digitais:	14 (6 podem fornecer saída *PWM*)
Pinos de entrada analógica:	6
Corrente *DC* por pino de E/S:	40*Ma*
Memória *Flash*:	32*KB*

***SRAM*:**	2*KB*
***EEPROM*:**	1*KB*
Frequência de *clock*:	16*MHz*

Tabela 1 - Resumo das Principais Características do Arduino Uno.
Fonte: (adaptada de Arduino.cc, 2014)

Firmware é um *software* que é carregado dentro da memória do micro controlador. Tecnicamente é a combinação de uma memória *ROM*, somente para leitura e um programa que fica gravado neste tipo de memória com instruções do funcionamento da placa (ERUS, 2012).

Botão *Reset*: tem a mesma função do pino *Reset* e é utilizado para reiniciar o micro controlador. Caso o pino seja forçado ao nível do terra, o micro controlador é reinicializado (HECO, 2015).

LED L: é interligado ao pino digital 13 e para utilizá lo é necessário enviar um nível alto e configurar tal pino como saída (HECO, 2015).

LED TX: representa o envio de comunicação de dados pelo pino digital 1 ou porta *USB* (HECO, 2015).

LED RX: representa a recepção de comunicação de dados pelo pino digital 0 ou porta *USB* (HECO, 2015).

AREF: representa a tensão de referência do conversor *A/D* (HECO, 2015).

GND: é a saída ground, ou seja é o pino terra (HECO, 2015).

ISR – Interrupt Service Routine ou Serviço de Interrupção de Rotina: são funções configuráveis por *software* e atribuídas apenas aos pinos digitais 2 e 3 para que uma interrupção seja tratada. Caso não haja necessidade de *ISR* pode se utilizar os pinos digitais 2 e 3 para qualquer outra finalidade (HECO, 2015).

Todos os pinos analógicos ou digitais possuem mais de uma função, ou seja,

podem ser tanto de entrada ou de saída e alguns podem servir para leituras analógicas e também como entrada digital. As funções são escolhidas pelo usuário quando este escrever um programa para sua placa. Na placa do Arduino, os pinos úteis do micro controlador são expostos e representados por conectores fêmeas, onde podem ser encaixados conectores para construir o circuito externo (ERUS, 2012).

No total existem 20 pinos que podem ser utilizados como entradas digitais. 14 pinos são digitais e 6 são analógicos.

Quando um pino é programado para funcionar como entrada digital, através do programa é enviado um comando que ao ser executado efetua a leitura da tensão aplicada. Só então após a execução deste comando, sabemos se o pino encontra se em um estado alto ou baixo e na prática, o programa consegue identificar se um pino está alimentado com 0 ou 5 volts. Tal função é utilizada para verificar se um botão está pressionado ou se um sensor está apresentando influência de algo do mundo externo. A função de entrada digital apenas entrega 0 ou 1, com ou sem tensão, por isso não é possível saber quanta tensão está sendo aplicada no pino (ERUS, 2012).

As 6 entradas analógicas do Arduino Uno, ao contrário de uma entrada digital, é capaz de medir a tensão aplicada no pino. Por meio da entrada analógica, podemos utilizar sensores que convertem alguma grandeza física em um valor de tensão que posteriormente é lido pela entrada analógica (ERUS, 2012).

Os pinos analógicos são portas de entrada de dados, isto é, não são portas que realizam a leitura de algum dispositivo através do método de leitura "*analogRead()*", como por exemplo, um potenciômetro. O comando *analogRead* converte a tensão de entrada, de 0 a 5 volts, a um valor digital entre 0 e 1023. Isto é feito por um circuito no interior do Arduino chamado de analógico-digital do conversor ou ADC. Caso haja necessidade é possível converter estes pinos analógicos em pinos digitais através de uma configuração por *software*.

Com relação as saídas digitais, com uma podemos fazer com que um pino tenha 0 ou 5 volts. Se programarmos o pino como saída digital, este pode acender um *LED*, ligar um relé, acionar um motor e entre outras infinidades de ações. Além disso, é

possível programar, utilizando um ou mais pinos para controlar um bloco de pinos (ERUS, 2012).

O Arduino também possui pinos com funções especiais, os quais são denominamos de *PWM* ou modulação por largura de pulso. O controle digital é usado para criar uma onda quadrática, em outras palavras: um sinal alternado entre ligado e desligado. Este padrão ligado/desligado pode simular tensões entre totalmente ligado (5 volts) e totalmente desligado (0 volts). A duração do tempo é chamada de largura de pulso. Se repetir o padrão ligar/desligar suficientemente rápido com um diodo emissor de luz, por exemplo, o resultado será como se o sinal fosse uma tensão constante entre 0 e 5 volts, possibilitando o controle da luminosidade do *LED* gradativamente (ARDUINO, 2014).

A figura 2 representa as ondas quadráticas das portas *PWM*. As linhas verdes significam um período de tempo regular. Esta duração ou período é o inverso da frequência de *PWM*. Em outras palavras, com a frequência *PWM* do Arduino em cerca de 500Hz, as linhas verdes mediriam 2 milissegundos cada. Uma chamada para *analogWrite()* é em uma escala de 0 a 254, de tal modo que *analogWrite(254)* solicita um ciclo de trabalho de 100% (sempre) e *analogWrite(127)* é um ciclo de trabalho de 50% (em metade do tempo), por exemplo (ARDUINO, 2014).

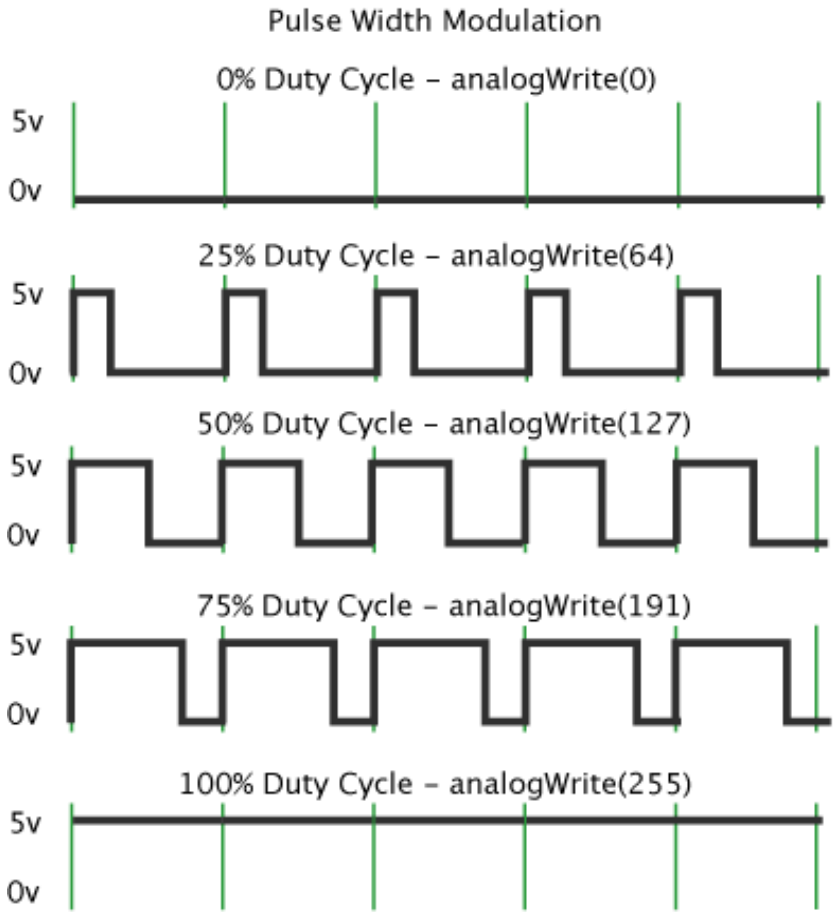


Figura 2 – Ondas quadráticas das portas *PWM*s.
Fonte: (adaptada de Arduino.cc, 2014)

A figura 3 mostra as 6 portas *PWM* em uma placa do Arduino Uno. Nelas as portas *PWM* são identificados por um "~" na frente do número referente a porta digital. Estas entradas permitem a obtenção de resultados analógicos por meio de sinal digital.

Figura 3 – Portas *PWM*s na Placa Arduino Uno.
Fonte: (adaptada de arduinoecia, 2014)

A arquitetura do *hardware* de um Arduino é representada pela figura 4.

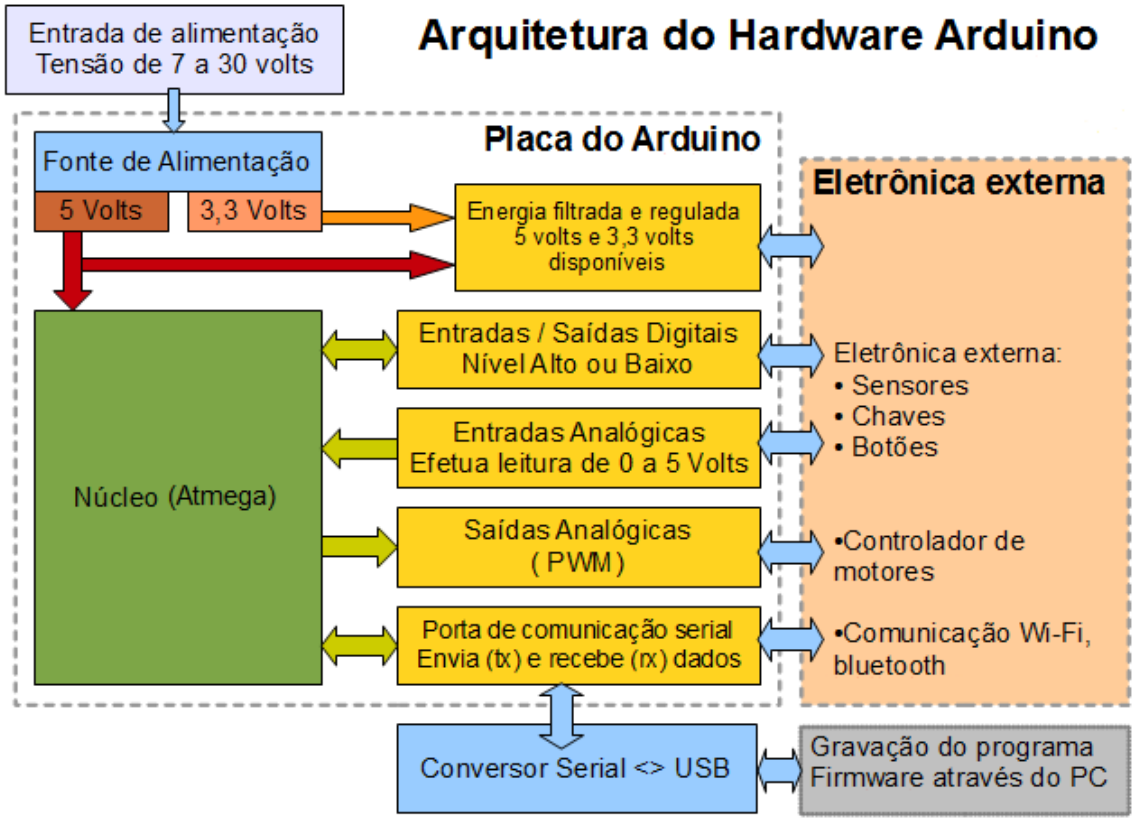


Figura 4 – Arquitetura do *hardware* de um Arduino Uno.
Fonte: (adaptada de robotizando, 2014)

Outras placas que possuem finalidades diferentes podem ser agregadas a placa micro controladora Arduino e a estas placas damos o nome de *shields*.

3.1.1.1 *Shields*

São placas de circuito modular capazes de serem adicionadas ao Arduino para obtenção de diversas funcionalidades extras como, por exemplo, conectá lo a *internet*, a um sensor de luminosidade, de temperatura ou movimento, som, *GPS* e/ou quaisquer funções que desejar (SPARKFUN, 2014).

A figura 5 mostra inúmeros *shields* que podem ser acoplados a placa do Arduino, possibilitando uma diversificação de funcionalidades.

Figura 5 – Diversificação de *Shields*.
Fonte: (adaptada de sparkfun, 2014)

3.1.1.2 Comunicação com o *Hardware*

O Arduino Uno tem uma série de facilidades para se comunicar seja com outro Arduino, um computador ou outros micro controladores com comunicação serial através da porta *USB* disponível nos pinos digitais 0 (*RX*) e 1 (*TX*) (ARDUINO, 2015).

No *software*, a comunicação serial é feita em uma porta denominada *COM* virtual e para seu funcionamento adequado não é exigido nenhum *driver* externo, exceto

quando utilizado no *Windows*, quando é solicitado um arquivo com extensão "*.inf*" (MULTILÓGICA, 2015).

Quando os dados estão sendo transmitidos através do chip *USB* para a serial e quando são transmitidos da conexão *USB* para o computador, os *LEDs RX* e *TX* da placa piscam (ARDUINO, 2015).

Além disso, o Arduino possui diversas portas de entrada e saída. Algumas apresentam propriedades particulares, o que as difere na leitura e escrita das portas analógicas e digitais.

3.1.1.2.1 Escritas *digitalWrite* e *analogWrite*

O comando "*digitalWrite*" realiza a escrita de um sinal digital, possuindo a seguinte sintaxe: "*digitalWrite(pino,valor)*". Na programação, a palavra "pino" é substituída pela posição em que o pino se encontra na placa do Arduino e a palavra "valor" substituída por um dos dois possíveis estados: *HIGH* ou *LOW* que significam respectivamente ligado e desligado.

O mesmo ocorre com o comando "*analogWrite*" que realiza a escrita de um sinal *PWM*, o que gera um resultado analógico, possuindo a seguinte sintaxe: "*analogWrite(pino,valor)*". Consequentemente, a palavra "pino" será substituída pela posição em que o pino se encontra na placa e a palavra "valor" será substituída por um valor a ser utilizado pelo método.

3.1.1.2.2 Leituras *digitalRead* e *analogRead*

Na programação, a palavra "pino" será substituída por um número referente a posição em que o pino se encontra na placa, o qual será conectado a um dispositivo

que poderá ser um leitor de temperatura, um sensor de proximidade ou até mesmo um sensor de luminosidade.

A função "*digitalRead*" realiza a leitura de um sinal digital, ela possui 2 tipos de retornos: 0 (*LOW*) ou 1 (*HIGH*) que respectivamente significam desligado e ligado, possui a sintaxe: "*digitalRead(pino)*".

A função "*analogRead*" realiza a leitura de um sinal analógico geralmente alternado entre uma tensão de 0 a 5 volts, possui um retorno que varia de 0 a 1023 e possui a sintaxe: "*analogRead(pino)*".

3.1.2 *Software*

O ambiente de desenvolvimento do Arduino é um compilador que utiliza interface gráfica construída em *Java*. Resumidamente é um programa conhecido como *IDE* Arduino, uma ferramenta comumente usada para desenvolvimento de projetos Arduino, podendo ser manipulada em sistemas operacionais *MacOS*, *Windows* e *Linux*, trabalhando com linguagens de programação C e C++ com versão 1.0.6 estável, apesar de já estar liberada a versão 1.5.8 beta para quem quiser testar (ARDUINO, 2015).

O programa é *open source*, oferecido gratuitamente aos usuários, mas seus desenvolvedores deixam bem claro que não se responsabilizam pelas ações dos usuários ao utilizá lo, isto é, segundo a documentação do site Arduino.cc, o *software* do Arduino é oferecido "como está" e não fazem quaisquer garantias de qualquer natureza que diz respeito a sua funcionalidade, operacionalidade ou uso, incluindo sem limitação, quaisquer garantias de comercialização, adequação a um determinado fim ou violação. Não se responsabilizam por danos diretos, indiretos, consequente, acidental ou danos especiais, incluindo sem limitação, receitas perdidas, lucros, perdas resultantes da interrupção de negócios ou perda de dados, independentemente da forma de ação ou teoria jurídica sob que a responsabilidade possa afirmar, mesmo que avisado da possibilidade ou probabilidade de tais danos (ARDUINO, 2015).

As duas funções básicas da *IDE* Arduino são: permitir o desenvolvimento de um *software* e seu envio a placa para que possa ser executado. Para realizar o *download* do *software IDE* basta ir a página oficial do Arduino (http://www.arduino.cc/), clicar em "*Download*", escolher a versão desejada e o seu Sistema Operacional (*Linux*, *Windows* ou *MacOS*) e baixá lo. Depois de instalado é só abrir o *IDE* e começar a utilizá lo (ERUS, 2012).

Após a inicialização do *software* é preciso escolher o tipo de placa que será utilizada, por isso deve se clicar em "*Tools*", selecionamos "*Board*" e a escolhemos. Além disso, o *software IDE* possui uma imensidão de exemplos e para utilizá los é só clicar em "*File*", depois em "*Examples*" e escolher qual deseja testar ou visualizar, isso depende da sua necessidade (ERUS, 2012).

A figura 6 mostra como realizar as instruções dadas acima para que possamos configurar o *IDE* antes de seu uso.

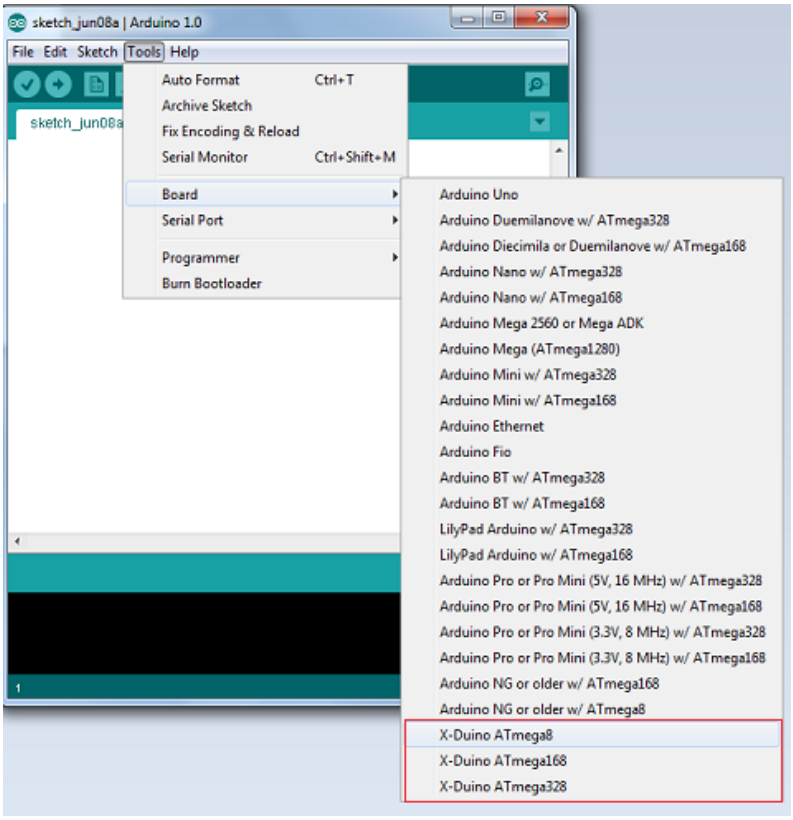


Figura 6 – Configurações a serem realizadas antes do uso do *IDE*.

Fonte: (adaptada de robótica, 2014)

Dependendo da placa a ser utilizada, talvez seja necessário informar a sua porta de comunicação. Isto é feito clicando no menu "*Tools*", selecionando "*Serial Port*" e por fim, clicando em uma porta serial (FILHO, 2012).

A figura 7 mostra como é realizado a escolha da porta correta, onde sua placa está conectada.

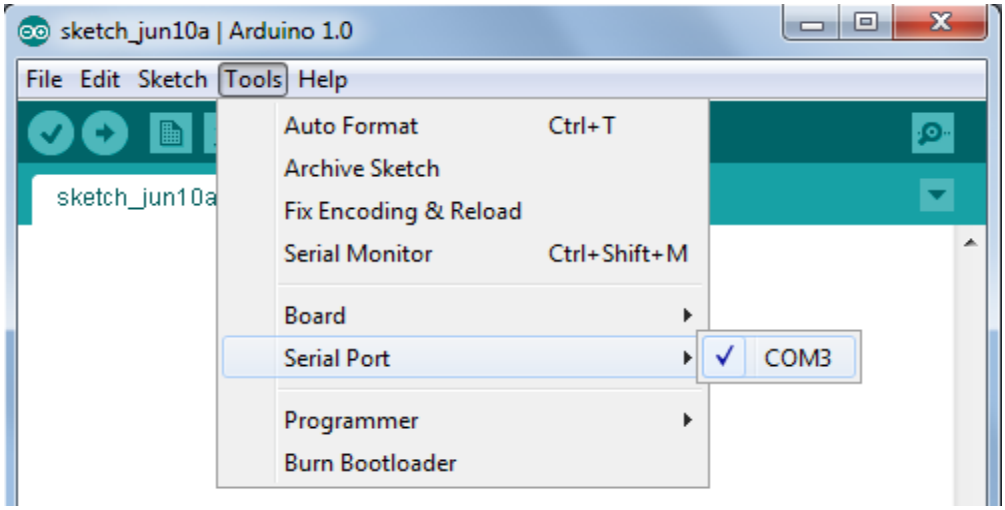


Figura 7 – Escolha da porta de comunicação.
Fonte: (cópia da tela, 2014)

A figura 8 mostra como é o ambiente de programação do Arduino. Sempre que um *script* for executado é possível visualizar e enviar ou receber dados para a placa Arduino via comunicação serial.

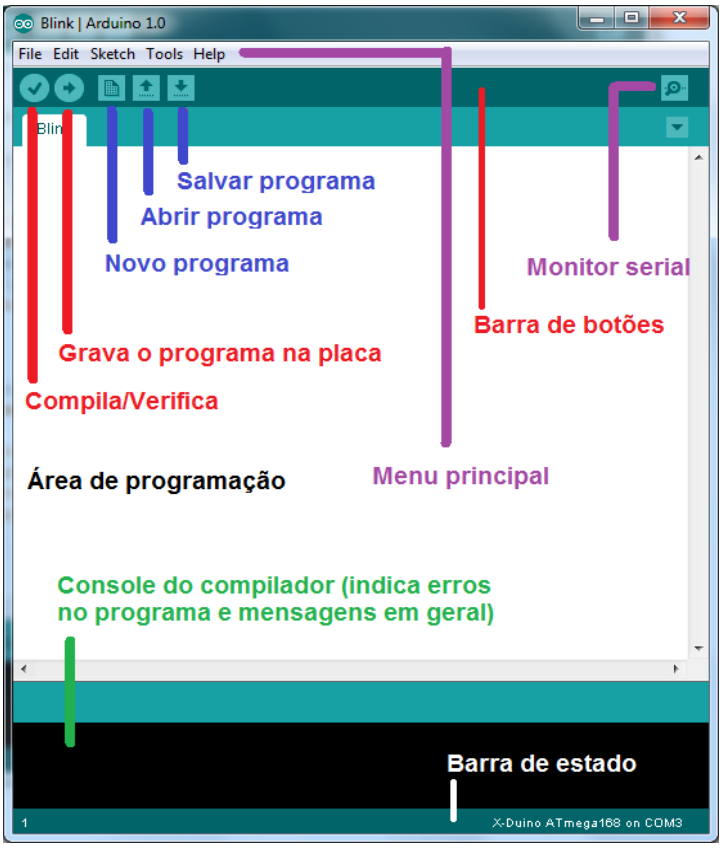


Figura 8 – Ambiente de programação do Arduino.
Fonte: (adaptada de robotizando, 2014)

Na figura 9 é possível visualizarmos um *script* exemplo que tem como função básica ligar ou desligar uma luz de *LED* conectada na porta digital 13 do Arduino. Neste simples exemplo também é possível ver a função *void setup()* e *void loop()*, ambas destacadas na cor laranja. Essas e outras funções são indispensáveis para o adequado funcionamento das ações que queremos realizar com o Arduino. A seguir veremos o resumo das mais usadas.

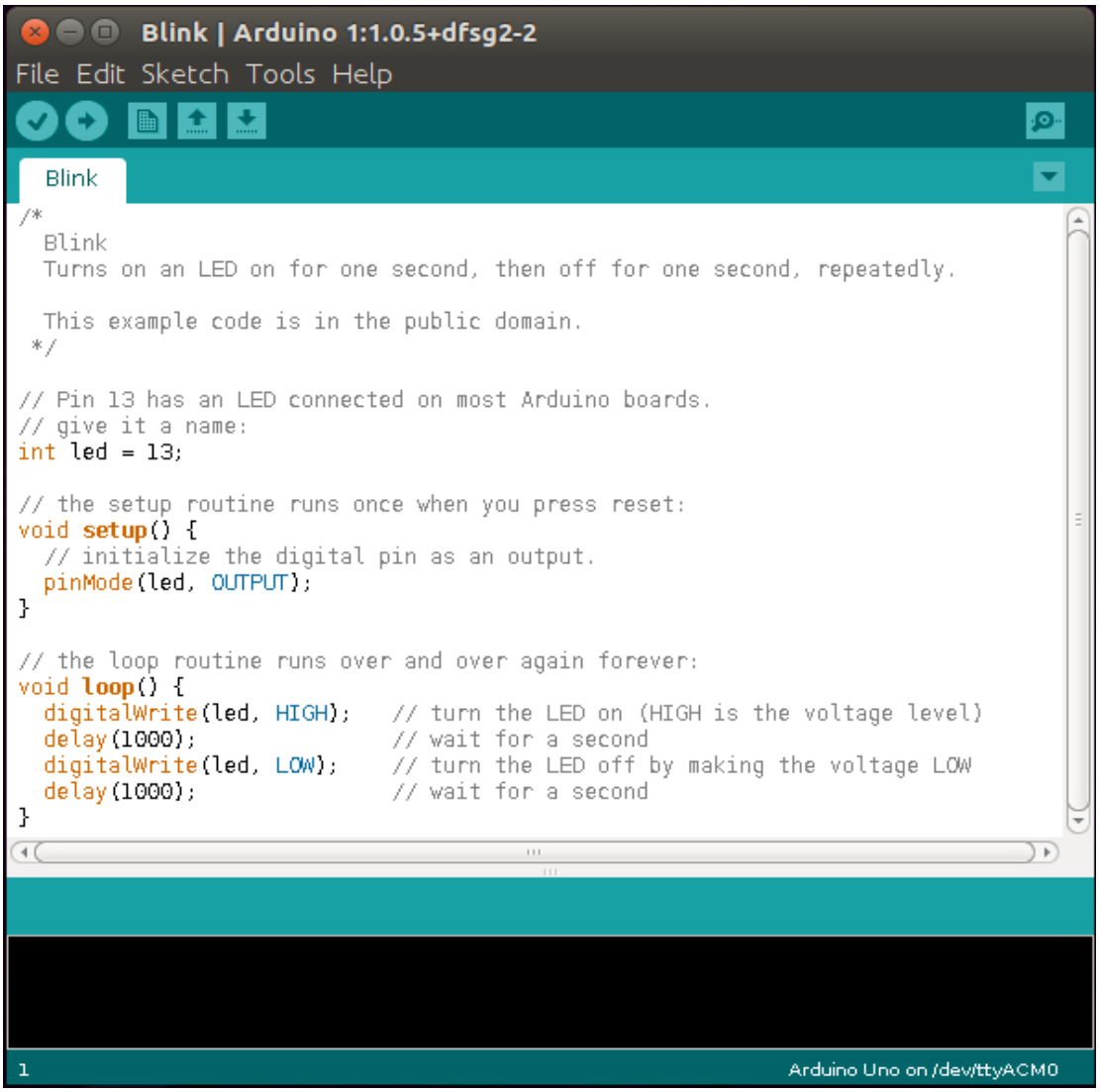


Figura 9 – Código exemplo "*Blink*" no Arduino.

Fonte: (cópia da tela, 2014)

3.1.2.1 Função *void setup*

A função *void setup()* é executada quando um algoritmo também conhecido como programa inicia. Sua função é inicializar o uso de bibliotecas, *pinMode*(s), além de definir valores iniciais para variáveis (ARDUINO, 2015).

Esta função é executada somente uma vez, quando a placa é energizada ou cada vez que a placa for “resetada”. É a primeira função a ser executada e é a encarregada de posteriormente chamar a função *void loop()* (ARDUINO, 2015).

```
Exemplo: void setup()  {
// declara o pino digital como saída:
pinMode(ledPin, OUTPUT);
}
```

3.1.2.2 Função *Serial.begin*

Para visualizarmos os valores no *Serial Monitor*, é necessário chamarmos a função *Serial.begin()* dentro da função *void setup()*, que por sua vez é usada para definir e ajustar a taxa de transferência de *bits* por segundo para transmissão de dados serial. Vale ressaltar que essas taxas podem ser: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 ou 115200, mas geralmente é usada a taxa de 9600bps. Também podemos especificar outras velocidades, como por exemplo, para a comunicação através dos pinos 0 e 1 com um componente que requer uma taxa específica (ARDUINO, 2015).

Quanto aos parâmetros, *int* velocidade é em *bits* por segundo (*baud*) e não há retorno (ARDUINO, 2015).

```
Exemplo: void setup () {
Serial.begin(9600);
// abre a porta serial e ajusta a taxa de transferência de dados para 9600bps
}
void loop() {}
```

3.1.2.3 Função *pinMode*

A função *pinMode* serve para configurar uma porta especificada para se comportar como entrada ou saída. Possui os seguintes parâmetros: "*pin*" é o número da porta que você deseja configurar e "*mode*" é o *INPUT* para entrada e *OUTPUT* para saída. Ambos não tem retorno. Além disso, as portas analógicas poderão ser utilizadas como portas digitais, referindo as como A0, A1 e assim por diante (ARDUINO, 2015).

Exemplo: // declara o pino digital 13 como entrada:

pinMode(13, INPUT);

// ou

// declara o pino digital 13 como saída:

pinMode(13, OUTPUT);

3.1.2.4 Função *void loop*

Depois de criar a função *void setup()*, a função *void loop()* faz exatamente o que seu nome sugere, ou seja, faz *loops* consecutivamente, permitindo que o programa mude e responda. Esta função irá executar repetidamente o código que estiver dentro dela até que se tenha uma condição que a interrompa (ARDUINO, 2015).

Exemplo: void loop() {

//Neste exemplo temos um valor sendo escrito em um pino digital.

digitalWrite(valor,pino);

}

3.1.2.5 Função *delay*

O *delay* interrompe o programa para a quantidade de tempo em milissegundos, especificado como parâmetro. Neste caso há 1000 milissegundos em um segundo e “ms” é o número de milissegundos para se fazer uma pausa dada como *unsigned long*, não tendo nada como retorno (ARDUINO, 2015).

```
Exemplo: int ledPin = 13   // LED conectado no pino digital 13
void setup()
{
pinMode(ledPin, OUTPUT);    // define o pino digital como saída
}
void loop()
{
digitalWrite(ledPin, HIGH);    // liga um LED
delay(1000);   // espera por 1 segundo
digitalWrite(ledPin, LOW);   // desliga um LED
delay(1000);   // espera por 1 segundo
}
```

4. PROTOCOLO

Protocolo é uma convenção que controla e possibilita uma conexão, comunicação e/ou transferência de dados entre dois sistemas. Teoricamente pode ser definido como regras que governam ou comandam a sintaxe, semântica e sincronização da comunicação. Os protocolos podem ser implementados pelo *hardware*, *software* ou por

uma combinação dos dois. Como alguns exemplos de protocolos de comunicação pode se citar: *RS232*, *RS485*, *I2C*, *TTL* e *ISP* (REISSWITZ, 2012).

O protocolo de comunicação desenvolvido para este projeto consiste no mapeamento das portas digitais, analógicas e *PWM* da placa de prototipação Arduino Uno e na transferência destes valores como texto para outro dispositivo. Ele permitirá a padronização da comunicação serial entre os dispositivos compatíveis com a comunicação e o Arduino.

A elaboração deste protocolo contará com os seguintes componentes: a inserção de caracteres verificadores iniciais e finais, leitura das portas analógicas, leitura e escrita das portas digitais comuns, escrita das portas *PWM*, geração do *checksum* e leitura e escrita de todo o protocolo. O protocolo será transformado em uma biblioteca que possui todas as funcionalidades propostas centralizadas.

Os detalhes das funcionalidades do protocolo serão vistas a seguir.

4.1 ESTRUTURA DO PROTOCOLO

Leitura:

“@@@@a0,a1,a2,a3,a4,a5,d2,d4,d7,d8,d12,d13,*checksum*!!!!”

Legenda:

“@@@@” representam os caracteres verificadores iniciais.

“a0,a1,a2,a3,a4,a5” representam as portas analógicas.

“d2,d4,d7,d8,d12,d13” representam as portas digitais comuns.

“*checksum*” representa o *checksum* do protocolo.

“!!!!” representa o fim do protocolo.

Segue um exemplo que ocorre na prática com valores aleatórios: “@@@@1023,0000,1023,0000,1023,1023,0,0,0,0,1,1,1532!!!!”

Legenda:

“@@@@” representam os caracteres verificadores iniciais.

“0 a 1023” representam valores das portas analógicas.

“0 ou 1” representam as portas digitais comuns.

“1532” representa o *checksum* gerado de acordo com as portas lidas.

“!!!!” representa o fim do protocolo.

Observação: as portas digitais comuns 0 e 1 não são utilizadas para leitura e/ou escrita, pois são responsáveis pela comunicação serial e por isso não aparecem no protocolo.

Escrita:

“@@@@a0,a1,a2,a3,a4,a5,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11,d12,d13,*checksum*!!!!”

Legenda:

“@@@@” representam os caracteres verificadores iniciais.

“a0 a1 a2 a3 a4 a5” representam as portas analógicas.

“d2 d4 d7 d8 d12 d13” representam as portas digitais comuns.

“d3 d5 d6 d9 d10 d11” representam as portas digitais *PWM*s.

“*checksum*” representa o *checksum* do protocolo.

“!!!!” representa o fim do protocolo.

Segue um exemplo que ocorre na prática com valores aleatórios:

“@@@@1023,1023,1023,1023,1023,1023,0,254,0,254,000,0,1,123,254,000,1,1, 1532!!!!”

Legenda:

“@@@@” representam os caracteres verificadores iniciais.

“0 a 1023” representam valores das portas analógicas.

“0 ou 1” representam as portas digitais comuns.

“0 a 254” representam as portas digitais *PWM*s.

“1532” corresponde ao número do *checksum*.

“!!!!” representa o fim do protocolo.

Como delimitadores iniciais da função são utilizados 4 arrobas (@) e como delimitadores finais 4 exclamações (!). Depois das 4 arrobas, o texto transferido possuirá os valores das 5 portas analógicas e os valores das 13 portas digitais, as quais serão subdivididas em digitais e *PWM*.

As portas digitais são de entrada e saída de informação e seus valores são 1 (um) ou 0 (zero). As portas digitais *PWM* podem enviar valores entre 0 e 254 e as analógicas, valores entre 0 e 1023.

As portas analógicas são portas apenas de entrada de dados, isto é, servem para leitura de algum dispositivo ou *shield* que agregue novas funcionalidades. Como exemplo de leitura podemos citar: sensores de temperatura, luminosidade, velocidade, movimento, entre outros.

O protocolo possui diversas funções (métodos) que realizam ações específicas, por exemplo: leitura das portas analógicas, leitura das portas digitais e criação do *checksum*, a fim de dividir uma grande funcionalidade em partes menores e que somente serão executadas quando realmente houver necessidade. Nesse sentido serão economizadas execuções desnecessárias de código.

A seguir serão descritas e demonstradas algumas das principais funções do protocolo.

4.1.1 Funções de Leitura

São funções utilizadas para auxiliar na montagem do protocolo de acordo com os valores lidos da placa micro controladora Arduino e que será enviado a algum outro aparelho ou dispositivo que possua comunicação serial. Estas funções são responsáveis pela inserção dos caracteres verificadores iniciais e finais, leituras das portas analógicas, leitura das portas digitais comuns, leituras das portas digitais tipo *PWM* e geração do *checksum*.

4.1.1.1 Leitura de Todas as Portas Analógicas

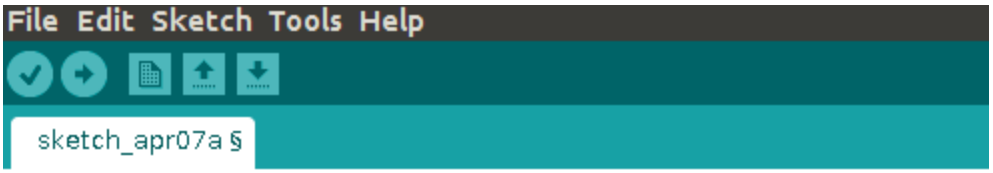
Nesta funcionalidade é usado o método `ler_analogicas()`. Possui função de ler todas as informações das portas analógicas, isto é, retornará o valor de todas as portas (A0, A1, A2, A3, A4 e A5) independente do que estiver conectado nelas. Tais valores estarão juntos e na mesma ordem das portas.

Exemplo:

Chamada do método: `ComSerial::ler_analogicas()`

Resposta: 0000,1023,0000,1023,0000,0000

Código desenvolvido:




```
// LER ANALOGICAS
String ComSerial::ler_analogicas(){
  String analogicas = "", aux = "";
  for(int i=0; i<6; i++){
    aux = (String) analogRead(i);
    if(aux.length() == 1) {
        aux = "000" + aux;
    }else if(aux.length() == 2){
        aux = "00" + aux;
    }else if(aux.length() == 3){
        aux = "0" + aux;
    }else{
    }
    analogicas += aux; // INSERE 0s A ESQUERDA ATE O TAMANHO DESEJADO
    if(i != 5){
      analogicas += ",";
    }
  }
        return analogicas;
}
```

Figura 10 – Código que realiza a leitura de todas as portas analógicas.

Fonte: (cópia da tela, 2015)

4.1.1.2 Leitura de Uma Porta Analógica

Nesta funcionalidade é usado o método `ler_analogica(porta)`. Possui a função de ler a informação de uma porta analógica específica, que por sua vez é informada no momento da chamada do método.

Quando o usuário chama a função passando a porta 2, por exemplo, ele pode ter um resultado variável entre 0 e 1023, correspondente ao que estiver ligado a porta naquele momento.

Exemplo:

Chamada do método: `ComSerial::ler_analogica(2)`

Resposta: 1023

Código desenvolvido:

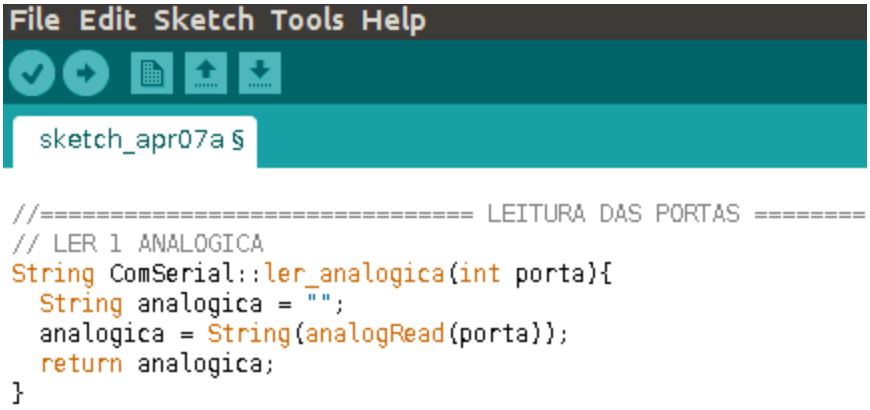


Figura 11 – Código que realiza a leitura de uma porta analógica.
Fonte: (cópia da tela, 2015)

4.1.1.3 Leitura de Todas as Portas Digitais Comuns

Nesta funcionalidade é usado o método `ler_digitais()`. Possui a função de ler todas as informações dos pinos digitais comuns, isto é, retornará o valor de todos os pinos (D2, D4, D7, D8, D12 e D13) independente do que estiver conectado neles. Os valores estarão juntos e na mesma ordem das portas.

Exemplo:

Chamada do método: `ComSerial::ler_digitais()`

Resposta: 1,0,1,1,0,1

Código desenvolvido:

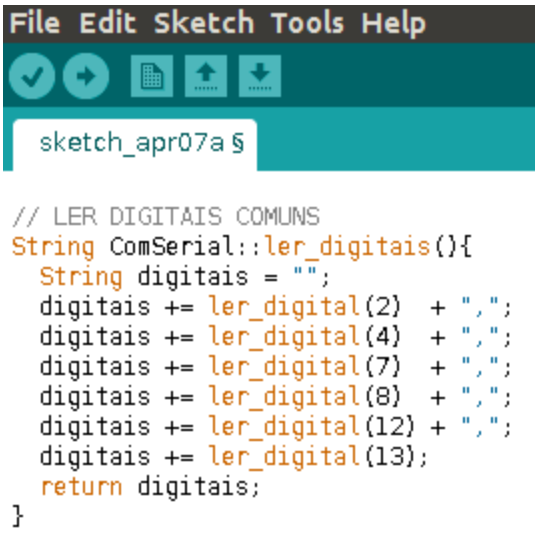


Figura 12 – Código que realiza a leitura de todas as portas digitais comuns.
Fonte: (cópia da tela, 2015)

4.1.1.4 Leitura de Uma Porta Digital Comum

Nesta funcionalidade é usado o método `ler_digital(porta)`. Possui a função de ler a informação de uma porta digital comum específica, que por sua vez é informada no momento da chamada do método.

Quando o usuário chama a função passando a porta 2, por exemplo, ele pode ter um resultado que pode ser 0 (desligado) ou 1 (ligado). Este valor corresponderá ao que estiver ligado a porta naquele momento.

Exemplo:

Chamada do método: `ComSerial::ler_digital(2)`

Resposta: 1

Código desenvolvido:

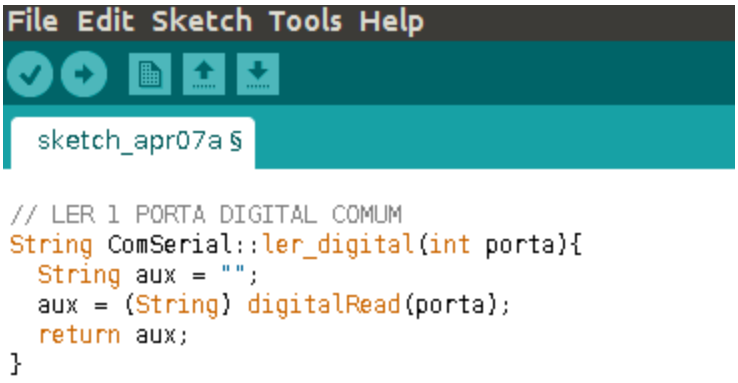


Figura 13 – Código que realiza a leitura de uma porta digital comum.
Fonte: (cópia da tela, 2015)

4.1.2 Funções de Escrita

São funções utilizadas para auxiliar na desmontagem do protocolo de acordo com os valores recebidos de algum aparelho ou dispositivo que possua comunicação serial. Estas funções são responsáveis pela verificação dos caracteres iniciais e finais, verificação das portas digitais comuns e digitais tipo *PWM*, além da verificação do *checksum*. Após as verificações é realizada a escrita dos valores nas portas digitais correspondentes a sua posição.

4.1.2.1 Escrita em Todas as Portas Digitais Comuns

Nesta funcionalidade é usado o método `escrever_digitais()`. Possui a função de escrever todas as informações nos pinos digitais comuns, isto é, escreverá o valor informado ao método (parâmetro) em todos os pinos (D2, D4, D7, D8, D12 e D13) independente do que estiver conectado neles.

Exemplo:

Chamada do método: `ComSerial::escrever_digitais(1,1,1,1,1,1)`

Resposta: Confirmação de escrita com sucesso ou de falha na escrita caso algum valor esteja errado.

Código desenvolvido:

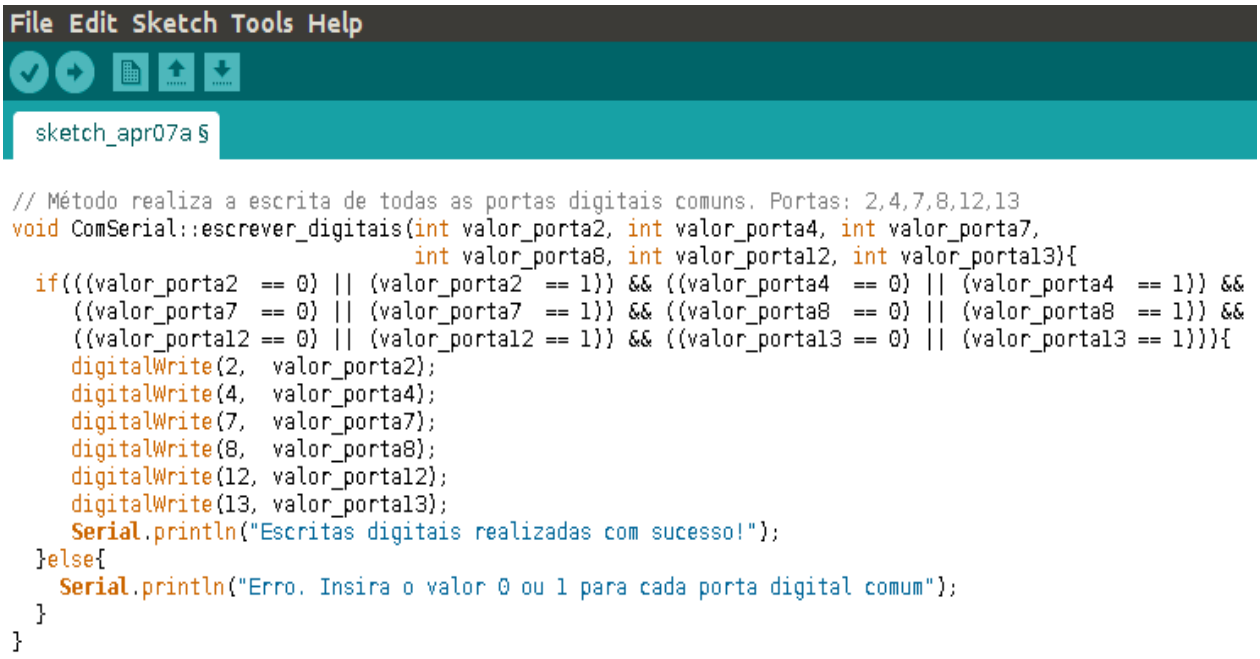


Figura 14 – Código que realiza a escrita de todas as portas digitais comuns.

Fonte: (cópia da tela, 2015)

4.1.2.2 Escrita em Uma Porta Digital Comum

Nesta funcionalidade é usado o método `escrever_digital(porta, valor)` que possui função de enviar ou escrever um valor em uma porta digital comum específica, que por sua vez, é informada no momento da chamada do método. Quando o usuário chama a função informando o valor igual a 1 para a porta 02, isto quer dizer

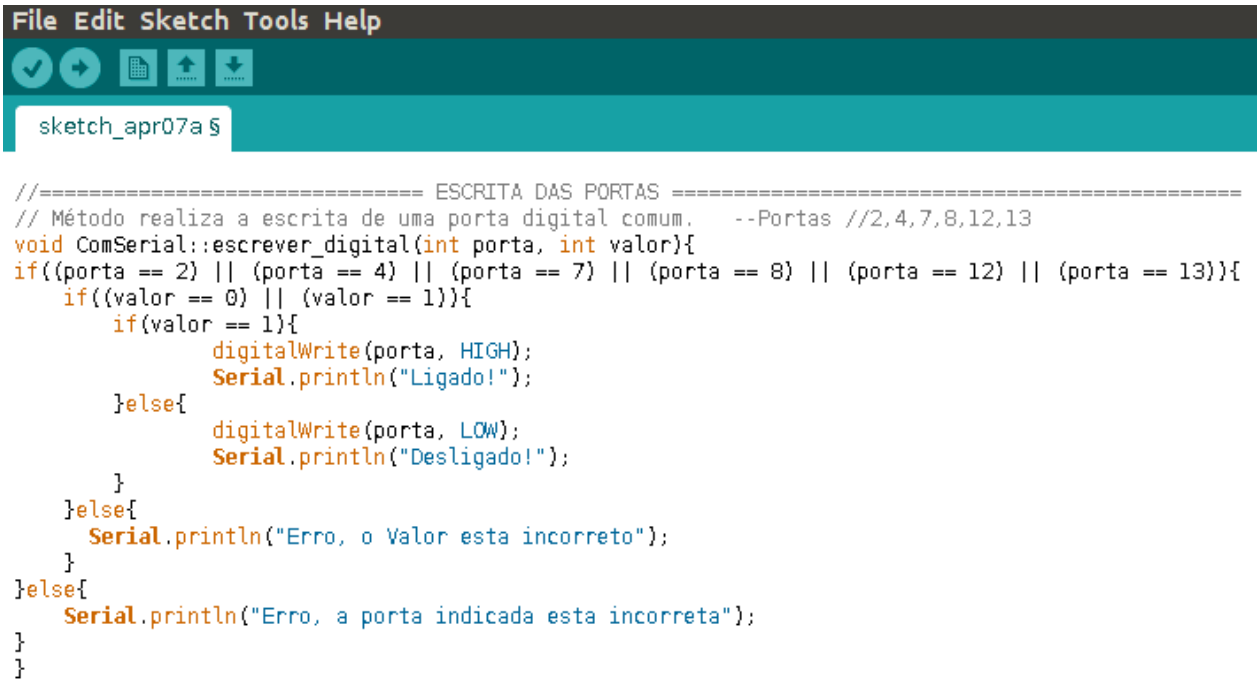
que o método enviará este valor 1 (ligado) para a porta digital comum 02 e quando informado o valor for igual a 0 para a porta 02, o método enviará este valor 0 (desligado) para a porta digital correspondente.

Exemplo:

Chamada do método: `ComSerial::escrever_digital(02,1);`

Reposta: Confirmação de escrita com sucesso ou de falha na escrita caso algum valor esteja errado.

Código desenvolvido:




```
//============================== ESCRITA DAS PORTAS ============================================
// Método realiza a escrita de uma porta digital comum.   --Portas //2,4,7,8,12,13
void ComSerial::escrever_digital(int porta, int valor){
if((porta == 2) || (porta == 4) || (porta == 7) || (porta == 8) || (porta == 12) || (porta == 13)){
    if((valor == 0) || (valor == 1)){
        if(valor == 1){
                digitalWrite(porta, HIGH);
                Serial.println("Ligado!");
        }else{
                digitalWrite(porta, LOW);
                Serial.println("Desligado!");
        }
    }else{
      Serial.println("Erro, o Valor esta incorreto");
    }
}else{
    Serial.println("Erro, a porta indicada esta incorreta");
}
}
```

Figura 15 – Código que realiza a escrita de uma porta digital comum.

Fonte: (cópia da tela, 2015)

4.1.2.3 Escrita em Todas as Portas Digitais *PWM*

Nesta funcionalidade é usado o método `escrever_pwms()`. Possui a função de escrever todas as informações nos pinos digitais *PWM*s, isto é, escreverá o valor informado ao método (parâmetro) em todos os pinos (D3, D5, D6, D9, D10 e D11) independente do que estiver conectado neles.

Exemplo da chamada do método:

```
ComSerial::escrever_pwms(254,000,254,254,000,000)
```

Resposta: Confirmação de escrita com sucesso ou de falha na escrita caso algum valor esteja errado.

Na imagem abaixo serão exibidas as conversões somente com as portas 03 e 05, pois o código completo seria muito extenso para esta demonstração. Entretanto, será disponibilizada a forma completa no final deste documento.

Código desenvolvido:

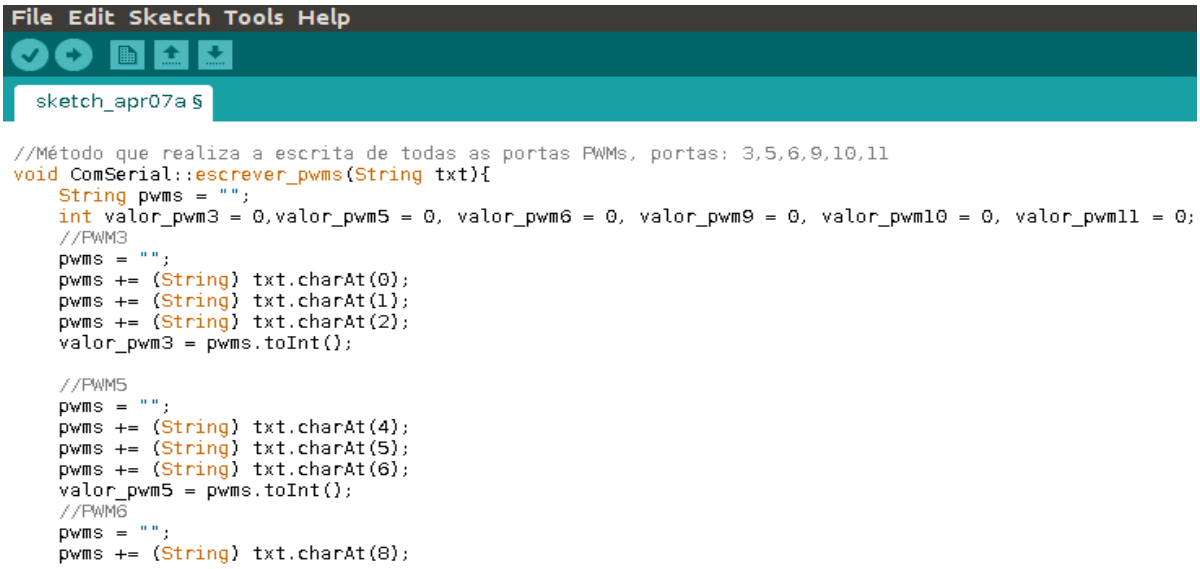


Figura 16 – Código que realiza a escrita de todas as portas *PWMs*.

Fonte: (cópia da tela, 2015)

4.1.2.4 Escrita em Uma Porta Digital *PWM*

Nesta funcionalidade é usado o método `escrever_pwm(porta, valor)` que possui a função de enviar ou escrever um valor em uma porta digital comum específica, que por sua vez é informada no momento da chamada do método. Quando o usuário chama a função informando o valor igual a 254 para a porta 3, isto quer dizer que o método enviará este valor 254 (valor máximo de pulso) para a porta *PWM* 3.

Exemplo:

Chamada do método: `ComSerial::escrever_pwm(03,254)`

Reposta: Confirmação de escrita com sucesso ou de falha na escrita caso algum valor esteja errado.

Código desenvolvido:

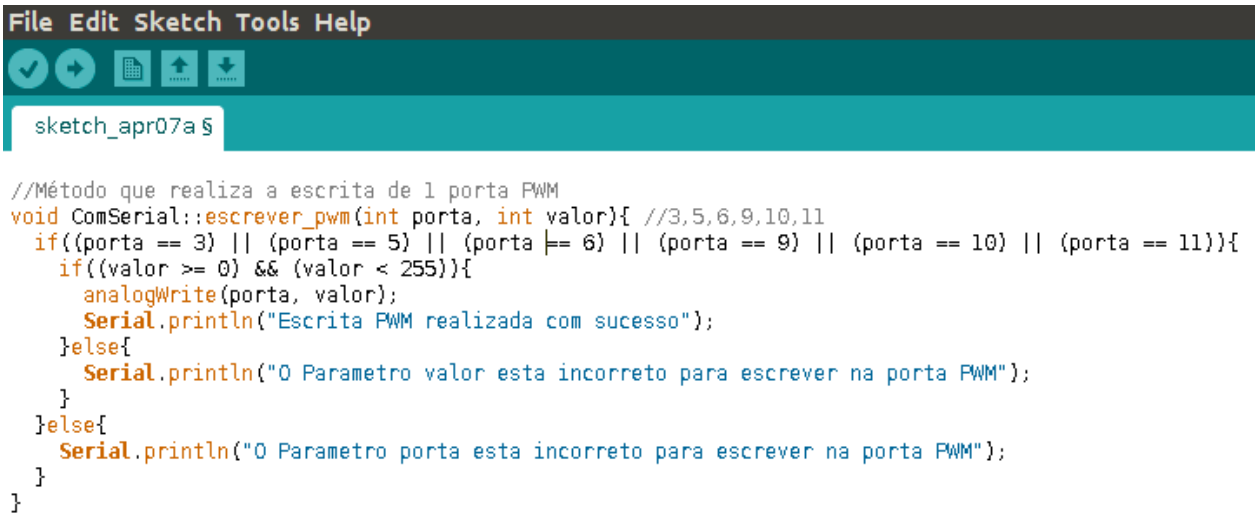


Figura 17 – Código que realiza a escrita de uma porta *PWM*.

Fonte: (cópia da tela, 2015)

4.1.3 *Checksum*

O *checksum* é um número verificador que serve basicamente para saber se a informação enviada é a mesma recebida por outros dispositivos, ou seja, com base nas

informações do protocolo ela gera um número que é verificado por outros dispositivos, garantindo assim a integridade dos dados. O cálculo do *checksum* neste protocolo é gerado diferentemente para leitura e para saída dos dados.

Leitura: O valor de cada porta analógica é dividido por 4 e posteriormente somado com os valores encontrados nas portas digitais comuns.

Pode ser expressa por:

CHECKSUM = SOMA (CADA_ANALOGICA / 4) + DIGITAIS_COMUNS

Recebimento: O valor de cada porta analógica é dividido por 4, somado juntamente com os valores das portas *PWM* divididos por 3 e somados com os valores encontrados nas portas digitais comuns.

Pode ser expressa por:

CHECKSUM = SOMA (CADA_ANALOGICA / 4) + SOMA (CADA_*PWM* / 3) + DIGITAIS_COMUNS

Quando o protocolo é enviado, ele realiza a geração do número do *checksum*. Esse cálculo é de acordo com os valores lidos e passa a ser adicionado ao final do protocolo (antes dos caracteres delimitadores finais "!!!!"). O processo de recebimento desta fórmula é utilizado para verificar a integridade da informação recebida e assim transmitir segurança tanto para quem envia quanto para quem recebe estes dados.

4.1.4 Leitura de Todo Protocolo

A função de leitura do protocolo tem a finalidade ler as informações que o Arduino está transmitindo ou recebendo no momento (de algum sensor ou placa conectada, por exemplo), e as deixar disponíveis em um texto que poderá ser enviado a outro dispositivo por meio da comunicação serial. Deste modo, os valores lidos pelo Arduino podem se tornar valores utilizados na comunicação entre outros equipamentos. Para montagem desse protocolo é realizada a geração de caracteres delimitadores iniciais e

finais (para não ter problemas com lixo de memória), a leitura de todas as portas analógicas, todas as portas digitais comuns e digitais *PWM*, além da geração do *checksum* para melhorar a segurança dos dados transmitidos.

Nesta funcionalidade é usado o método "`ler_protocolo()`", no qual possui a função de ler todas as informações de todos os pinos (ou portas) independente do que estiver conectado neles.

Exemplo:

Chamada do método: `ComSerial::ler_protocolo()`

Resposta: É retornada então um texto que contém todos os valores lidos em todas as portas do Arduino, com exceção é claro das portas *PWM* que são somente portas de escrita. O texto que será retornado possui a estrutura:

"@@@@0000,1023,1023,0000,0000,1023,1,0,1,0,1,0,1534!!!!".

Código desenvolvido:

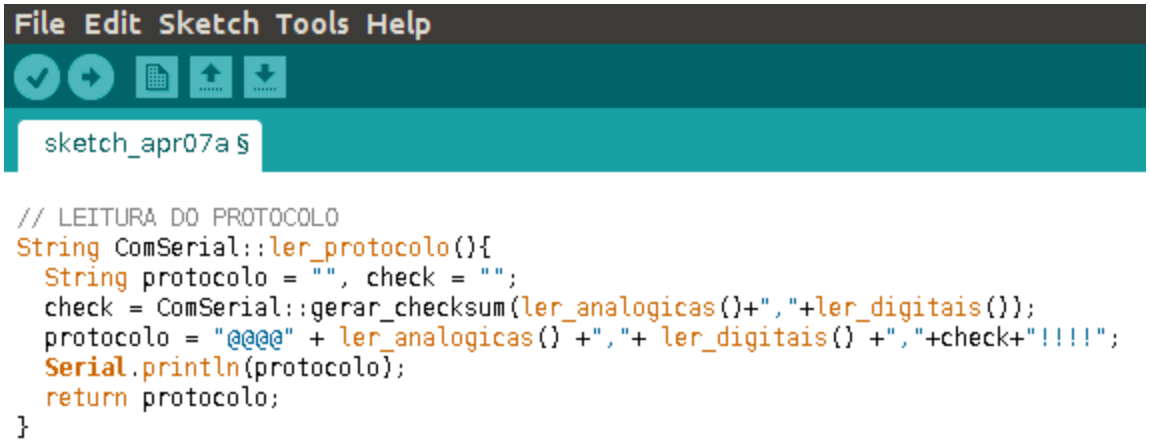


Figura 18 – Código que realiza a leitura de todo o protocolo no Arduino.

Fonte: (cópia da tela, 2015)

4.1.5 Escrita de Todo Protocolo

A função de envio do protocolo tem a finalidade escrever as informações que algum dispositivo transmitiu no momento, seja ele algum outro equipamento compatível com a comunicação ou mesmo um usuário que queira testar o envio de informações para o Arduino, por exemplo. Essa função escreve o protocolo que foi recebido de outro dispositivo por meio da comunicação serial. Deste modo, os valores recebidos pelo Arduino são desmontados, isto é, o protocolo em primeiro lugar realiza a verificação do *checksum* para verificar a segurança dos dados recebidos, depois faz a separação dos caracteres delimitadores iniciais e finais (garantindo que não foi transmitido nenhum lixo de memória) e por fim é realizada a separação de todas as portas analógicas, todas as portas digitais comuns e digitais *PWM*, além da geração do *checksum* para melhorar a segurança dos dados transmitidos.

Nesta funcionalidade é usado o método "`escrever_protocolo (String proto)`", no qual possui a função de escrever todas as informações de todos os pinos (ou portas) recebidos, independente do que estiver conectado neles.

Exemplo:

Chamada do método: `ComSerial::escrever_protocolo (String proto)`

"*String proto*" é um parâmetro (é o próprio protocolo) que a função necessita para realizar o desmembramento das variáveis e assim poder escrever nas portas correspondentes. É interessante lembrar que as portas analógicas não recebem escrita, pois são apenas de leitura.

Resposta: É retornada uma mensagem dizendo se a escrita foi realizada com sucesso ou se houve algum erro durante sua execução.

Código desenvolvido:

Figura 19 – Código que realiza a escrita de todo o protocolo no Arduino.

Fonte: (cópia da tela, 2015)

5. BIBLIOTECA LIVRE

As bibliotecas são como um conjunto de subprogramas (contendo classes e métodos próprios) independentes e que geralmente são utilizados para incorporar alguma funcionalidade extra ao desenvolvimento do seu programa. As alterações que a biblioteca implica no seu programa, isto é, as alterações realizadas por ela são de forma modular e deste modo temos bibliotecas que alteram apenas partes bem pequenas do seu programa (fazendo alguma conversão de valores ou tipos de arquivos) ou até mesmo uma alteração substancial de todo o seu código. Neste projeto a biblioteca

desenvolvida tem funções claramente simples, voltadas a comunicação serial e assim como outras bibliotecas Arduino também é desenvolvida em linguagem de programação C++.

Outro exemplo é a biblioteca *wire* que implementa o protocolo de comunicação *bus I2C (TWI)* e a biblioteca *SPI* que implementa o protocolo para comunicação do barramento *SPI* (ARDUINO, 2015). A seguir será demonstrado o processo de como o desenvolvimento da biblioteca foi realizado:

Foi criada uma pasta dentro da pasta "*libraries*" que fica dentro da pasta Arduino (*/usr/share/arduino/libraries/ComSerial*) com o mesmo nome que será chamada a biblioteca e que possui o seguinte conteúdo:

1. A pasta "*examples*" conterá os exemplos que apareceram na *IDE* do Arduino;
2. O Arquivo "*keywords.txt*" *que serve* para as palavras da biblioteca mudarem de cor na *IDE*;
3. Os arquivos com extensão "*.h*" e "*.cpp*" *possuem* os códigos da biblioteca.

A imagem a seguir demonstra esses arquivos:

Figura 20 – Arquivos necessários para criação da biblioteca.

Fonte: (cópia da tela, 2015)

5.1 CABEÇALHOS DA BIBLIOTECA SERIAL

Para que as declarações da biblioteca não sejam inseridas mais de uma vez em um projeto é necessário editar o arquivo "*ComSerial.h*", que é o arquivo de cabeçalho da biblioteca.

Assim foi adicionado no início do arquivo o código "*#ifndef COMSERIAL_H*", "*#define ComSerial_H*" e no final do arquivo o código "*#endif*".

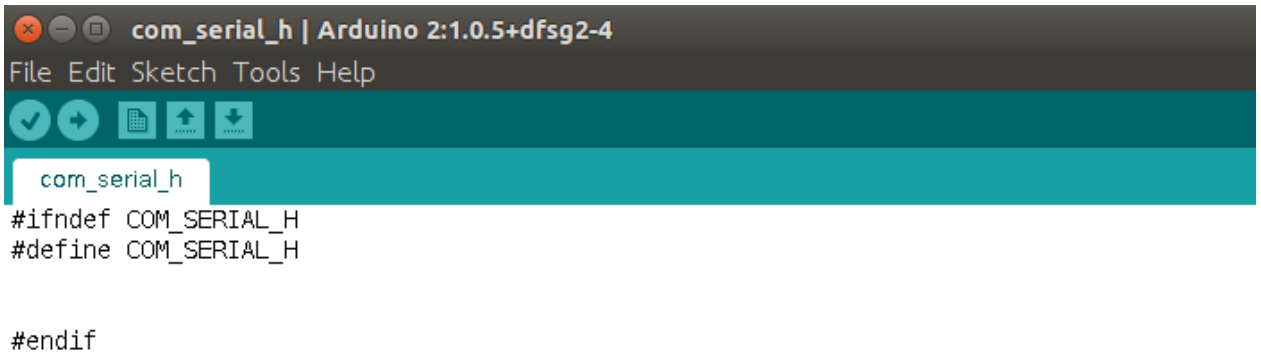


Figura 21 – Cabeçalhos da biblioteca serial.

Fonte: (cópia da tela, 2015)

5.2 CABEÇALHO DA BIBLIOTECA ARDUINO

Para ter acesso às funções do Arduino é necessário fazer uso da biblioteca Arduino inserindo o arquivo de cabeçalho "*Arduino.h*" no meio do cabeçalho criado acima. O código usado foi: "*#include <Arduino.h>*".

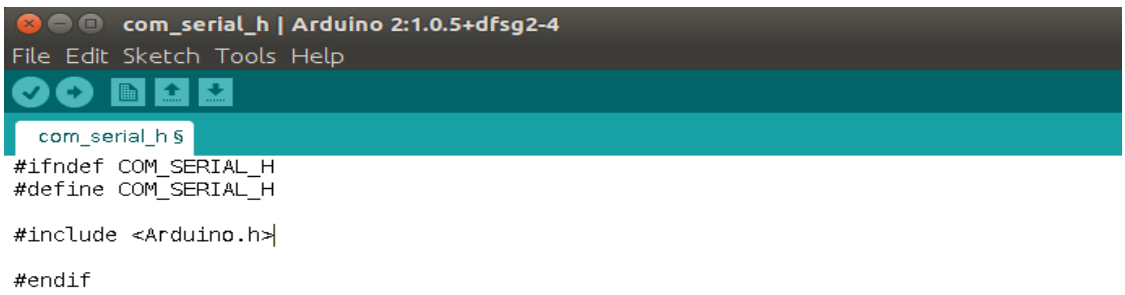


Figura 22 – Cabeçalho da biblioteca Arduino.

Fonte: (cópia da tela, 2015)

5.3 CONSTRUTOR DA CLASSE ComSerial

Esta classe possui um construtor que recebe os métodos públicos que serão acessados por quem for utilizar a biblioteca. Ainda com o arquivo "*ComSerial.h*" será adicionado tais métodos, como demonstrado na imagem abaixo:

Figura 23 – Construtor da classe da biblioteca.

Fonte: (cópia da tela, 2015)

5.4 FUNCIONALIDADES DA BIBLIOTECA

No arquivo *"ComSerial.cpp"* os métodos demonstram as ações que os seus nomes propõem, fazendo uso das funções da biblioteca Arduino.

Exemplo de parte do código desenvolvido:

Figura 24 – Funcionalidades da biblioteca.

Fonte: (cópia da tela, 2015)

5.5 RECONHECIMENTO DE CORES NO ARDUINO

O arquivo *"keywords.txt"* deve ser editado e ficar da seguinte forma:

1- Cada linha tem o nome da palavra-chave seguida por uma tabulação (não é um espaço), seguidos pelo tipo da palavra.

2- As classes devem ser do tipo *KEYWORD1* para que sejam coloridas de laranja.

3- Os métodos devem ser do tipo *KEYWORD2* para que sejam coloridas de marrom.

4- Terminando de definir o tipo de palavra para seu script, o ambiente do Arduino deve ser reiniciado para que as modificações sejam aplicadas.

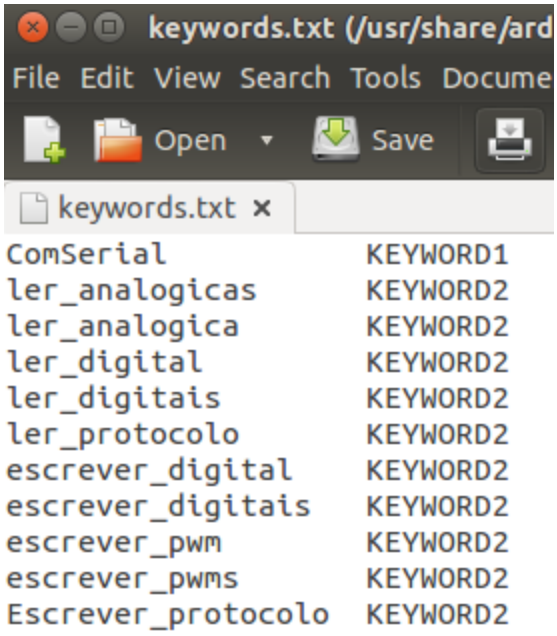


Figura 25 – Código de reconhecimento de cores do Arduino.

Fonte: (cópia da tela, 2015)

5.6 ARQUIVOS EXEMPLOS

Uma vez que a biblioteca está pronta é interessante a criação de arquivos exemplos que demonstrem como utilizar as funções da sua classe. Para isso dentro da pasta Arduino encontramos uma subpasta chamada "*examples*" (exemplos) e é dentro dela que será colocado o arquivo que demonstra como utilizar a biblioteca.

Código exemplo:


```
void loop(//  FUNÇOES ACEITAS PELA BIBLIOTECA
//================ LEITURA DAS PORTAS ====================
//    --METOTO PARA LER UMA ANALOGICA
    ComSerial.ler_analogica(0);

//    --METODO PARA LER TODAS AS ANALOGICAS
    ComSerial.ler_analogicas();

//    --METODO PARA LER 1 DIGITAL
    ComSerial.ler_digital(2);

//    --METODO PARA LER TODAS DIGITAIS COMUNS
    ComSerial.ler_digitais();

//    --METODO PARA LER TODO O PROTOCOLO
    ComSerial.ler_protocolo();

//================ ESCRITA DAS PORTAS ====================
//    -- METODO PARA ESCREVER 1 DIGITAL
    ComSerial.escrever_digital(4,1);

//    -- METODO PARA ESCREVER TODAS DIGITAIS
    ComSerial.escrever_digitais(0,1,1,1,1,1);

//    -- METODO PARA ESCREVER 1 PWM
    ComSerial.escrever_pwm(3,50);

//    -- METODO PARA ESCREVER TODAS AS PWMs
    ComSerial.escrever_pwms("254,254,254,254,254,254");

//    --METODO PARA ESCREVER TODO O PROTOCOLO
    ComSerial.escrever_protocolo("@@@@1023,1023,1023,1023,1023,1023,1,254,1,254,254,1,1,254,254,254,1,1,1622!!!!");
}
```

Figura 26 – Código de exemplo de uso da biblioteca serial.

Fonte: (cópia da tela, 2015)

6. CONCLUSÕES E TRABALHOS FUTUROS

Utilizando a biblioteca de comunicação serial desenvolvida é possível chamar uma função simples e intuitiva, como por exemplo, `ler_analogicas()` e o resultado com os valores de todas as portas do tipo analógicas estarão entregues em poucos segundos. Da mesma forma acontece quando se torna necessário escrever em uma saída do tipo digital, pois basta chamar a função `escrever_digital(porta,valor)` informando o valor à ser escrito e a porta onde será escrita, que muito rapidamente o Arduino estará escrevendo nesta saída.

O tratamento dos valores contidos no protocolo da biblioteca serial é um ponto considerável, pois faz com que todos os valores retornem sempre na mesma posição da sua estrutura lógica e com o número de *bits* pré estabelecido para que a aplicação não consuma mais memória que o necessário.

Contudo, a função de destaque é a de conseguir enviar os valores lidos do seu Arduino a outro dispositivo que possua comunicação serial. Do mesmo modo e com sentido inverso, a biblioteca consegue segmentar o protocolo recebido de outros dispositivos para que possa enviar os valores às portas correspondentes a placa micro controladora.

De modo geral, os benefícios gerados pela biblioteca serial são visíveis e indicam possibilidades que podem auxiliar diversas aplicações que usem comunicação serial, como por exemplo, na comunicação entre dois ou mais Arduinos (muito comum em robôs), um Arduino com uma câmera, impressora, computador ou qualquer outro dispositivo que possua comunicação serial.

Entretanto, como trabalhos futuros o projeto ainda será aprimorado com a criação de uma criptografia para melhorar a segurança. A biblioteca ainda será submetida a análise do fabricante Arduino e se aprovada o resultado será a distribuição gratuita online.

7. REFERÊNCIAS BIBLIOGRÁFICAS

ARDUINO. **Arduino Uno**: Communication. Disponível em URL: <http://Arduino.cc/en/Main/ArduinoBoardUno>. Acesso em 25/01/2015.

ARDUINO. **Arduino Uno**: Power. Disponível em URL: <http://Arduino.cc/en/Main/ArduinoBoardUno>. Acesso em 19/01/2015.

ARDUINO. **begin ( )**. Disponível em URL: <http://arduino.cc/en/Serial/begin>. Acesso em 27/01/2015.

ARDUINO. **delay( )**. Disponível em URL: <http://arduino.cc/en/reference/delay>. Acesso em 27/01/2015.

ARDUINO. **Download the Arduino Software**. Disponível em URL: <http://arduino.cc/en/Main/software>. Acesso em 26/01/2015.

ARDUINO. **loop( )**. Disponível em URL: <http://arduino.cc/en/pmwiki.php?n=Reference/Loop>. Acesso em 27/01/2015.

ARDUINO. **pinMode( )**. Disponível em URL: <http://arduino.cc/en/Reference/pinMode>. Acesso em 27/01/2015.

ARDUINO. **PWM**. Disponível em URL: <http://Arduino.cc/en/Tutorial/PWM>. Acesso em 07/07/2014.

ARDUINO. **setup( )**. Disponível em URL: <http://arduino.cc/en/Reference/Setup>. Acesso em 27/01/2015.

ARDUINO. **SPI library**. Disponível em URL: <http://Arduino.cc/en/Reference/SPI>. Acesso em 25/01/2015.

ERUS – Equipe de Robótica UFES. **Minicurso Arduino**. Disponível em URL: <http://www.inf.ufes.br/~erus/arquivos/materiais/Minicurso%20Arduino%20JACEE%202012.pdf>. Acesso em 27/05/2014.

FILHO, Daniel O. Basconcello. **Curso de Arduino**: Aula 3 – O Software do Arduino. Disponível em URL: <http://www.robotizando.com.br/curso_arduino_software_pg2.php>. Acesso em 27/05/2014.

GRUPO DE ROBÓTICA – UFMS 2012. **Introdução ao Arduino**. Mato Grosso do Sul: DESTACOM – Despertando Talentos em Computação. Disponível em URL: <http://destacom.ufms.br/mediawiki/images/9/9f/Arduino_Destacom.pdf>. Acesso em 19/01/2015.

HECO – Mecatrônica. **Arduino Uno**: Guia do Usuário. Disponível em URL: <http://www.hecomecatronica.com.br/Artigos%20e%20Documentos/Artigos%20Tecnicos/Documentos/01%20Arduino%20uno%20-%20guia%20do%20usuario.pdf>. Acesso em 21/01/2015.

MCROBERTS, Michael. **Arduino Básico**. São Paulo: Novatec, 2011.

MULTILÓGICA – Shop Open Source Hardware. **Arduino Uno R3**. Disponível em URL: <http://multilogica-shop.com/Arduino-Uno>. Acesso em 25/01/2015.

REISSWITZ, Flávia. **Análise de Sistemas Vol. 2**: Tecnologia Web & Redes. Joinville/SC: Clube de Autores Publicações S/A, 2012.

SOUZA, Fábio. **Arduino Uno**. Disponível em URL: <http://www.embarcados.com.br/Arduino-uno/>. Acesso em 25/01/2015.

SPARKFUN. **Arduino Shields**. Disponível em URL: <https://learn.sparkfun.com/tutorials/Arduino-shields/all>. Acesso em 17/07/2014.

8. APÊNDICE A – CÓDIGO

SCRIPT DO ARQUIVO CABEÇALHO ComSerial

ARQUIVO "*ComSerial.h*"

```
1  #ifndef ComSerial_H
2  #define ComSerial_H
3  #include <Arduino.h>
4
5  class ComSerial{
6  public:       ComSerial(int valor);
7
8  //============  ESCRITA DAS PORTAS ====================
9  void escrever_digital(int porta, int valor);
10 void escrever_digitais(int valor_porta2, int valor_porta4,int valor_porta7,
11                        int valor_porta8, int valor_porta12, int valor_porta13);
12 void escrever_pwm(int  porta, int valor);
13 void escrever_pwms(String  txt);
14 void escrever_protocolo(String proto);
15
16 //=============  LEITURA DAS PORTAS ====================
17 String ler_analogica(int porta);
18 String ler_analogicas();
19 String ler_digital(int porta);
20 String ler_digitais();
21 String ler_protocolo();
22
23 //=============  OUTRAS FUNÇÕES ====================
24 private String gerar_checksum(String txt);
25 private boolean verificar_delimitadores(String proto);
26 private boolean verificar_digitais(String digitais);
27 private boolean verificar_pwms(String  pwms);
28 private boolean verificar_checksum(String proto);
29
30 private: int valor;
31 };
32 #endif
```

SCRIPT DO ARQUIVO ComSerial

ARQUIVO "_ComSerial.cpp_"

```
#include "ComSerial.h"
#include "string.h"

ComSerial::ComSerial(int pino){
        pinMode(pino, OUTPUT);
}

//============================== LEITURA DAS PORTAS
==============================================
// LER 1 ANALOGICA
String ComSerial::ler_analogica(int porta){
  String analogica = "";
  analogica = String(analogRead(porta));
  return analogica;
}

// LER ANALOGICAS
String ComSerial::ler_analogicas(){
  String analogicas = "", aux = "";
  for(int i=0; i<6; i++){
    aux = (String) analogRead(i);
    if(aux.length() == 1) {
       aux = "000" + aux;
    }else if(aux.length() == 2){
       aux = "00" + aux;
    }else if(aux.length() == 3){
       aux = "0" + aux;
    }else{
    }
    analogicas += aux; // INSERE 0s A ESQUERDA ATE O TAMANHO DESEJADO
    if(i != 5){
      analogicas += ",";
    }
  }
        return analogicas;
}

// LER 1 PORTA DIGITAL COMUM
String ComSerial::ler_digital(int porta){
  String aux = "";
  aux = (String) digitalRead(porta);
  return aux;
```

```
}

// LER DIGITAIS COMUNS
String ComSerial::ler_digitais(){
  String digitais = "";
  digitais += ler_digital(2)  + ",";
  digitais += ler_digital(4)  + ",";
  digitais += ler_digital(7)  + ",";
  digitais += ler_digital(8)  + ",";
  digitais += ler_digital(12) + ",";
  digitais += ler_digital(13);
  return digitais;
}

// LEITURA DO PROTOCOLO
String ComSerial::ler_protocolo(){
  String protocolo = "", check = "";
  check = ComSerial::gerar_checksum(ler_analogicas()+","+ler_digitais());
  protocolo = "@@@@" + ler_analogicas() +","+ ler_digitais() +","+check+"!!!!";
  Serial.println(protocolo);
  return protocolo;
}

//============================== ESCRITA DAS PORTAS
================================================
// Método realiza a escrita de uma porta digital comum.  --Portas //2,4,7,8,12,13
void ComSerial::escrever_digital(int porta, int valor){
if((porta == 2) || (porta == 4) || (porta == 7) || (porta == 8) || (porta == 12) || (porta ==
13)){
   if((valor == 0) || (valor == 1)){
            if(valor == 1){
            digitalWrite(porta, HIGH);
            Serial.println("Ligado!");
            }else{
            digitalWrite(porta, LOW);
            Serial.println("Desligado!");
            }
   }else{
     Serial.println("Erro, o Valor esta incorreto");
   }
}else{
   Serial.println("Erro, a porta indicada esta incorreta");
}
}

// Método realiza a escrita de todas as portas digitais comuns. Portas: 2,4,7,8,12,13
```

```
121 void ComSerial::escrever_digitais(int valor_porta2, int valor_porta4, int valor_porta7, int
122 valor_porta8, int valor_porta12, int valor_porta13){
123   if(((valor_porta2  == 0) || (valor_porta2  == 1)) && ((valor_porta4  == 0) || (valor_porta4
124 == 1)) &&
125      ((valor_porta7  == 0) || (valor_porta7  == 1)) && ((valor_porta8  == 0) || (valor_porta8
126 == 1)) &&
127      ((valor_porta12 == 0) || (valor_porta12 == 1)) && ((valor_porta13 == 0) ||
128 (valor_porta13 == 1))){
129      digitalWrite(2,  valor_porta2);
130      digitalWrite(4,  valor_porta4);
131      digitalWrite(7,  valor_porta7);
132      digitalWrite(8,  valor_porta8);
133      digitalWrite(12, valor_porta12);
134      digitalWrite(13, valor_porta13);
135      Serial.println("Escritas digitais realizadas com sucesso!");
136   }else{
137     Serial.println("Erro. Insira o valor 0 ou 1 para cada porta digital comum");
138   }
139 }
140
141 //Método que realiza a escrita de 1 porta PWM
142 void ComSerial::escrever_pwm(int porta, int valor){ //3,5,6,9,10,11
143   if((porta == 3) || (porta == 5) || (porta == 6) || (porta == 9) || (porta == 10) || (porta ==
144 11)){
145     if((valor >= 0) && (valor < 255)){
146       analogWrite(porta, valor);
147       Serial.println("Escrita PWM realizada com sucesso");
148     }else{
149       Serial.println("O Parametro valor esta incorreto para escrever na porta PWM");
150     }
151   }else{
152     Serial.println("O Parametro porta esta incorreto para escrever na porta PWM");
153   }
154 }
155
156 //Método que realiza a escrita de todas as portas PWMs, portas: 3,5,6,9,10,11
157 void ComSerial::escrever_pwms(String txt){
158     String pwms = "";
159     int valor_pwm3 = 0,valor_pwm5 = 0, valor_pwm6 = 0, valor_pwm9 = 0, valor_pwm10
160 = 0, valor_pwm11 = 0;
161     //PWM3
162     pwms = "";
163     pwms += (String) txt.charAt(0);
164     pwms += (String) txt.charAt(1);
165     pwms += (String) txt.charAt(2);
166     valor_pwm3 = pwms.toInt();
```

```
167
168       //PWM5
169       pwms = "";
170       pwms += (String) txt.charAt(4);
171       pwms += (String) txt.charAt(5);
172       pwms += (String) txt.charAt(6);
173       valor_pwm5 = pwms.toInt();
174       //PWM6
175       pwms = "";
176       pwms += (String) txt.charAt(8);
177       pwms += (String) txt.charAt(9);
178       pwms += (String) txt.charAt(10);
179       valor_pwm6 = pwms.toInt();
180       //PWM9
181       pwms = "";
182       pwms += (String) txt.charAt(12);
183       pwms += (String) txt.charAt(13);
184       pwms += (String) txt.charAt(14);
185       valor_pwm9 = pwms.toInt();
186       //PWM10
187       pwms = "";
188       pwms += (String) txt.charAt(16);
189       pwms += (String) txt.charAt(17);
190       pwms += (String) txt.charAt(18);
191       valor_pwm10 = pwms.toInt();
192     //PWM11
193       pwms = "";
194       pwms += (String) txt.charAt(20);
195       pwms += (String) txt.charAt(21);
196       pwms += (String) txt.charAt(22);
197       valor_pwm11 = pwms.toInt();
198
199       if(((valor_pwm3  >= 0)  && (valor_pwm3  < 255)) &&
200         ((valor_pwm5  >= 0)  && (valor_pwm5  < 255)) &&
201         ((valor_pwm6  >= 0)  && (valor_pwm6  < 255)) &&
202
203         ((valor_pwm9  >= 0)  && (valor_pwm9  < 255)) &&
204         ((valor_pwm10 >= 0)  && (valor_pwm10 < 255)) &&
205         ((valor_pwm11 >= 0)  && (valor_pwm11 < 255))){
206
207         analogWrite(3,  valor_pwm3);
208         analogWrite(5,  valor_pwm5);
209         analogWrite(6,  valor_pwm6);
210         analogWrite(9,  valor_pwm9);
211         analogWrite(10,  valor_pwm10);
212         analogWrite(11,  valor_pwm11);
```

```
213         Serial.println("Portas PWMs escritas com sucesso");
214       }else{
215         Serial.println("Ha valores incorretos para as portas PWMs");
216       }
217    }
218
219  //================  ESCRITA DO PROTOCOLO ==================
220  //Realiza a escrita do protocolo recebido por algum dispositivo
221   void ComSerial::escrever_protocolo(String proto){
222     // VERIFICACOES
223     if(verificar_delimitadores(proto) == true){
224       if(verificar_checksum(proto)){
225         if(verificar_digitais(proto) == true){
226           if(verificar_pwms(proto)  == true){
227
228                   //ESCRITA DAS PORTAS DIGITAIS
229                   escrever_digitais((proto[34]-'0'),(proto[40]-'0'),(proto[50]-'0'),(proto[52]-
230  '0'),(proto[66]-'0'),(proto[68]-'0'));
231
232                 //ESCRITA DAS PORTAS PWMs
233                 String aux = "";
234                 //PWM 3
235                 aux  = (String) proto[36];
236                 aux += (String) proto[37];
237                 aux += (String) proto[38];
238                 aux += ",";
239
240                 //PWM 5
241                 aux += (String) proto[42];
242                 aux += (String) proto[43];
243                 aux += (String) proto[44];
244                 aux +=  ",";
245
246                 //PWM 6
247                 aux += (String) proto[46];
248                 aux += (String) proto[47];
249                 aux += (String) proto[48];
250                 aux += ",";
251
252                 //PWM 9
253                 aux += (String) proto[54];
254                 aux += (String) proto[55];
255                 aux += (String) proto[56];
256                 aux += ",";
257
258                 //PWM 10
```

```
259                 aux += (String) proto[58];
260                 aux += (String) proto[59];
261                 aux += (String) proto[60];
262                 aux = ",";
263 
264                 //PWM 11
265                 aux += (String) proto[62];
266                 aux += (String) proto[63];
267                 aux += (String) proto[64];
268 
269                 escrever_pwms(aux);
270 
271               }else{
272               }
273             }else{
274             }
275           }else{
276           }
277         }else{
278         }
279       }
280 
281       //===========   OUTRAS FUNÇÕES ======
282       //METODO QUE REALIZA A GERACAO DO CHECKSUM
283       String ComSerial::gerar_checksum(String txt){
284       int checksum = 0;
285       String temp = "";
286 
287       //txt = "1023,1023,1023,1023,1023,1023,1,1,1,1,1,1";
288 
289       //ANALOGICAS
290       //A0
291       temp = "";
292       for(int i=0; i<4; i++){ // POSICAO DE 0 A 3
293        temp += (String) txt[i];
294       }
295       checksum += temp.toInt() / 4;
296 
297       //A1
298       temp = "";
299       for(int i=5; i<9; i++){ //POSICAO DE 5 A 8
300         temp += (String) txt[i];
301       }
302       checksum += temp.toInt()/4;
303 
304       //A2
```

```
305   temp = "";
306   for(int i=10; i<14; i++){ // POSICAO DE 10 A 13
307    temp += (String) txt[i];
308   }
309   checksum += temp.toInt()/4;
310
311   //A3
312   temp = "";
313   for(int i=15; i<19; i++){ // POSICAO DE 15 A 18
314    temp += (String) txt[i];
315   }
316   checksum += temp.toInt()/4;
317
318   //A4
319   temp = "";
320   for(int i=20; i<24; i++){ // POSICAO DE 20 A 23
321    temp += (String) txt[i];
322   }
323   checksum += temp.toInt()/4;
324
325   //A5
326   temp = "";
327   for(int i=25; i<29; i++){ // POSICAO DE 25 A 28
328     temp += (String) txt[i];
329   }
330   checksum += temp.toInt()/4;
331   Serial.println(checksum);
332
333   //DIGITAIS
334   //D2
335   temp = "";
336   temp += (String) txt[30];
337   checksum += temp.toInt();
338
339   //PWM D3 NAO SERA ADICIONADA
340
341   //D4
342   temp = "";
343   temp += (String) txt[32];
344   checksum += temp.toInt();
345
346   //PWM D5 NAO SERA ADICIONADA
347   //PWM D6 NAO SERA ADICIONADA
348
349   //D7
350   temp = "";
```

```
351   temp += (String) txt[34];
352   checksum += temp.toInt();
353
354   //D8
355   temp = "";
356   temp += (String) txt[36];
357   checksum += temp.toInt();
358
359   //PWM D9 NAO SERA ADICIONADA
360   //PWM D10 NAO SERA ADICIONADA
361   //PWM D11 NAO SERA ADICIONADA
362
363   //D12
364   temp = "";
365   temp += (String) txt[38];
366   checksum += temp.toInt();
367
368   //D13
369   temp = "";
370   temp += (String)txt[40];
371   checksum += temp.toInt();
372
373   String aux = "";
374   aux  = (String) checksum;
375
376   Serial.println(aux);
377
378   if(aux.length() == 3){
379     aux = "0" + aux;
380   }else{
381   }
382       return aux;
383   }
384
385   //==========  METODOS VERIFICADORES ======================
386   // Verificador dos caracteres especiais iniciais e finais
387   boolean ComSerial::verificar_delimitadores(String proto){
388     String verifica = "";
389     verifica =  (String) proto[0];
390     verifica += (String) proto[1];
391     verifica += (String) proto[2];
392     verifica += (String) proto[3];
393
394     verifica += (String) proto[74];
395     verifica += (String) proto[75];
396     verifica += (String) proto[76];
```

```
397     verifica += (String) proto[77];
398
399    if((verifica[0] == '@') && (verifica[1] == '@') && (verifica[2] == '@') && (verifica[3] ==
400   '@') &&
401       (verifica[4] == '!') && (verifica[5] == '!') && (verifica[6] == '!') && (verifica[7] == '!')){
402      Serial.println("Caracteres verificadores iniciais e finais reconhecidos");
403      return true;
404    }
405    else{
406      Serial.println("Erro ao reconhecer verificadores iniciais e finais");
407      return false;
408    }
409   }
410
411   // Verificação das digitais comuns:  2,4,7,8,12,13
412   boolean ComSerial::verificar_digitais(String digitais){
413     String proto = "";
414     proto += (String)digitais[34];
415     proto += (String)digitais[40];
416     proto += (String)digitais[50];
417     proto += (String)digitais[52];
418     proto += (String)digitais[66];
419     proto += (String)digitais[68];
420
421    if(((proto[0] == '0')  || (proto[0] == '1')) && ((proto[1] == '0') || (proto[1] == '1')) &&
422       ((proto[2] == '0')  || (proto[2] == '1')) && ((proto[3] == '0') || (proto[3] == '1')) &&
423       ((proto[3] == '0')  || (proto[3] == '1')) && ((proto[4] == '0') || (proto[4] == '1'))){
424        Serial.println("Portas digitais comuns reconhecidas na verificacao do protocolo");
425
426        digitalWrite(2, proto[0]);
427        digitalWrite(4, proto[1]);
428        digitalWrite(7, proto[2]);
429        digitalWrite(8, proto[3]);
430        digitalWrite(12, proto[4]);
431        digitalWrite(13, proto[5]);
432
433        return true;
434    }else{
435      Serial.println("Portas digitais comuns NAO reconhecidas na verificacao do
436   protocolo");
437      return false;
438    }
439   }
440
441   // VERIFICACAO DAS PWMs
442   boolean ComSerial::verificar_pwms(String pwms){
```

```
443   int v3=0, v5=0, v6=0, v9=0, v10=0, v11=0;
444   // JUNTANDO AS PORTAS PWMs
445     //PORTA 3
446     String temp = "";
447     temp += (String) pwms[36];
448     temp += (String) pwms[37];
449     temp += (String) pwms[38];
450     v3 = temp.toInt();
451     // PORTA 5
452     temp = "";
453     temp += (String) pwms[42];
454     temp += (String) pwms[43];
455     temp += (String) pwms[44];
456     v5 = temp.toInt();
457     //PORTA 6
458     temp = "";
459     temp += (String) pwms[46];
460     temp += (String) pwms[47];
461     temp += (String) pwms[48];
462     v6 = temp.toInt();
463     //PORTA 9
464     temp = "";
465     temp += (String) pwms[54];
466     temp += (String) pwms[55];
467     temp += (String) pwms[56];
468     v9 = temp.toInt();
469     //Porta 10
470     temp = "";
471     temp += (String) pwms[58];
472     temp += (String) pwms[59];
473     temp += (String) pwms[60];
474     v10 = temp.toInt();
475     //Porta 11
476     temp = "";
477     temp += (String) pwms[62];
478     temp += (String) pwms[63];
479     temp += (String) pwms[64];
480     v11 = temp.toInt();
481
482    if(((v3 >=0) || (v3 <255)) && ((v5 >=0) || (v5 <255))&&
483       ((v6 >=0) || (v6 <255)) && ((v9 >=0) || (v9 <255))&&
484       ((v10 >=0) || (v10 <255)) && ((v11 >=0) || (v11 <255))){
485         Serial.println("Portas digitais PWMs reconhecidas na verificacao do protocolo");
486         analogWrite(3, v3);
487         analogWrite(5, v5);
488         analogWrite(6, v6);
```

```
489         analogWrite(9, v9);
490         analogWrite(10, v10);
491         analogWrite(11, v11);
492         return true;
493       }else{
494         Serial.println("Portas digitais PWMs NAO reconhecidas na verificacao do protocolo");
495         return false;
496       }
497     }
498
499     //VERIFICACAO DO CHECKSUM
500     boolean ComSerial::verificar_checksum(String proto){
501       String tmp = "", check_do_protocolo = "";
502       int checksum = 0;
503
504     // ANALOGICAS
505      for(int i=4; i<33; i++){
506       tmp += (String) proto[i];
507         if(i==7||i==12||i==17||i==22||i==27||i==32){
508           checksum += (tmp.toInt()/4);
509         }
510      }
511
512     // DIGITAIS
513     //D2
514      tmp = "";
515      tmp = (String) proto[34];
516      checksum += (tmp.toInt());
517
518     //D3(PWM)
519      tmp = "";
520      for(int i=36; i<39; i++){/// POSICAO 36 A 38
521        tmp += (String) proto[i];
522      }
523      checksum += (tmp.toInt()) / 3;
524
525     //D4
526      tmp = "";
527      tmp = (String) proto[40];
528      checksum += (tmp.toInt());
529
530     //D5(PWM)
531      tmp = "";
532      for(int i=42; i<45; i++){ // POSICAO 42 A 44
533        tmp += (String) proto[i];
534      }
```

```
535   checksum += (tmp.toInt()/3);
536
537   //D6(PWM)
538   tmp = "";
539   for(int i=46; i<49; i++){ // POSICAO 46 A 48
540     tmp += (String) proto[i];
541   }
542   checksum += (tmp.toInt()/3);
543
544   //D7
545   tmp = "";
546   tmp = (String) proto[50];
547   checksum += (tmp.toInt());
548
549   //D8
550   tmp = "";
551   tmp = (String) proto[52];
552   Serial.println("\nD8");
553   Serial.println(tmp.toInt());
554   checksum += (tmp.toInt());
555
556   //D9(PWM)
557   tmp = "";
558   for(int i=54; i<57; i++){ // POSICAO 54 A 56
559     tmp += (String) proto[i];
560   }
561   Serial.println("\nD9");
562   Serial.println(tmp.toInt()/3);
563   checksum += (tmp.toInt()/3);
564
565   //D10(PWM)
566   tmp = "";
567   for(int i=58; i<61; i++){ // POSICAO 58 A 60
568   tmp += (String) proto[i];
569   }
570   Serial.println("\nD10");
571   Serial.println(tmp.toInt()/3);
572   checksum += (tmp.toInt()/3);
573
574   //D11(PWM)
575   tmp = "";
576   for(int i=62; i<65; i++){ // POSICAO 62 A 64
577     tmp += (String) proto[i];
578   }
579   Serial.println("\nD11");
580   Serial.println(tmp.toInt()/3);
```

```
581     checksum += (tmp.toInt()/3);
582
583    //D12
584     tmp = "";
585     tmp = (String) proto[66];
586     Serial.println("\nD12");
587     Serial.println(tmp.toInt());
588     checksum += (tmp.toInt());
589
590    //D13
591     tmp = "";
592     tmp = (String) proto[68];
593     Serial.println("\nD13");
594     Serial.println(tmp.toInt());
595     checksum += (tmp.toInt());
596
597     //CHECKSUM RECEBIDO
598     check_do_protocolo =  (String) proto[70];
599     check_do_protocolo += (String) proto[71];
600     check_do_protocolo += (String) proto[72];
601     check_do_protocolo += (String) proto[73];
602
603     tmp = "";
604     tmp = (String) checksum;
605
606     Serial.println(tmp);
607
608      if(tmp == check_do_protocolo){
609          Serial.println("Passou no teste do checksum");
610          return true;
611      }else{
612          Serial.println("NAO Passou no teste do checksum");
613          return false;
614       }
615    }
```

SCRIPT EXEMPLO DE USO DA BIBLIOTECA

ARQUIVO EXEMPLO "_ComSerialExample.ino_"

```
616 #include <ComSerial.h>
617
618 ComSerial ComSerial(2);
619
620 void setup(){
621   Serial.begin(9600);
622 }
623
624 void loop(//  FUNÇOES ACEITAS PELA BIBLIOTECA
625 //================  LEITURA DAS PORTAS ====================
626 //   --METOTO PARA LER UMA ANALOGICA
627     ComSerial.ler_analogica(0);
628
629 //   --METODO PARA LER TODAS AS ANALOGICAS
630     ComSerial.ler_analogicas();
631
632 //   --METODO PARA LER 1 DIGITAL
633     ComSerial.ler_digital(2);
634
635 //   --METODO PARA LER TODAS DIGITAIS COMUNS
636     ComSerial.ler_digitais();
637
638 //   --METODO PARA LER TODO O PROTOCOLO
639     ComSerial.ler_protocolo();
640
641 //================  ESCRITA DAS PORTAS ====================
642 //   -- METODO PARA ESCREVER 1 DIGITAL
643     ComSerial.escrever_digital(4,1);
644
645 //   -- METODO PARA ESCREVER TODAS DIGITAIS
646     ComSerial.escrever_digitais(0,1,1,1,1,1);
647
648 //   -- METODO PARA ESCREVER 1 PWM
649     ComSerial.escrever_pwm(3,50);
650
651 //   -- METODO PARA ESCREVER TODAS AS PWMs
652     ComSerial.escrever_pwms("254,254,254,254,254,254");
653
654 //   --METODO PARA ESCREVER TODO O PROTOCOLO
655
656 ComSerial.escrever_protocolo("@@@@1023,1023,1023,1023,1023,1023,1,254,1,254,
657 254,1,1,254,254,254,1,1,1622!!!!");
```

```
}
```

SCRIPT KEYWORDS DA BIBLIOTECA

ARQUIVO DE RECONHECIMENTO DE CORES "_keywords.txt_"

```
ComSerial           KEYWORD1
ler_analogicas      KEYWORD2
ler_analogica       KEYWORD2
ler_digital         KEYWORD2
ler_digitais        KEYWORD2
ler_protocolo       KEYWORD2
escrever_digital    KEYWORD2
escrever_digitais   KEYWORD2
escrever_pwm        KEYWORD2
escrever_pwms       KEYWORD2
Escrever_protocolo  KEYWORD2
```