

FACULDADE DE TECNOLOGIA DE SÃO PAULO  
DEPARTAMENTO DE SISTEMAS ELETRÔNICOS  
CURSO SUPERIOR DE TECNOLOGIA EM ELETRÔNICA INDUSTRIAL

## **OSCILOSCÓPIO VIA ARDUINO**

**Raphael Yutaka Marques**

**TRABALHO DE CONCLUSÃO DE CURSO**

SÃO PAULO 2016

# **OSCILOSCÓPIO VIA ARDUINO**

Trabalho de Conclusão de Curso,  
apresentado ao Departamento de  
Sistemas Eletrônicos, da Faculdade de  
Tecnologia de São Paulo – FATEC-SP,  
como requisito parcial para conclusão da  
graduação.

Orientador: Prof. Dr. Francisco  
Tadeu Degasperi

SÃO PAULO 2016

## **TERMO DE APROVAÇÃO**

### **OSCILOSCÓPIO VIA ARDUINO**

Este trabalho de conclusão de disciplina foi apresentado no dia 12 de Agosto de 2016, como requisito parcial para aprovação no curso de Eletrônica Industrial, outorgado pela Faculdade de Tecnologia de São Paulo. Após deliberação, os examinadores consideraram o trabalho aprovado.

---

Departamento de Sistemas Eletrônicos

Dedico este trabalho à minha família e amigos, os  
quais me incentivaram desde o início.

## **AGRADECIMENTO**

Agradeço à todo o coletivo de professores representantes do curso de Eletrônica Industrial, da Fatec São Paulo, por todos os ensinamentos passados durante minha formação.

## RESUMO

Este trabalho aborda o projeto, desenvolvimento, construção e testes de um osciloscópio via Microncontrolador Arduino. O projeto é composto por dois pontos principais: o de coleta dos dados e o módulo gráfico. A coleta é feita através do conversor AD do Arduino, onde o mesmo fica encarregado de processar e enviar a leitura digital via comunicação Serial. O módulo gráfico (Processing), por sua vez, interpreta os dados recebidos e disponibiliza em um plano 2D na tela do computador. É possível obter o valor da frequência e tensão máxima, assim como, pausar e aplicar um zoom na imagem gerada. Este sistema pode ser uma boa opção, pois, o seu custo de operação é baixo quando comparado ao osciloscópio convencional.

**Palavras chave:** Arduino. Processing. Comunicação Serial.

## LISTA DE ILUSTRAÇÕES

Figura 1 - Eixos X, Y e Z num osciloscópio .....	11
Figura 2 - Sinais não periódicos .....	12
Figura 3 - Sinais periódicos .....	12
Figura 4 - Amplitude do sinal .....	12
Figura 5 - Layout do Arduino. ....	15
Figura 6 - Pinagem Arduino. ....	16
Figura 7 - Conectores de Alimentação. ....	16
Figura 8 - Diagrama de funcionamento do conversor A/D.....	18
Figura 9 - Possíveis valores do clock do conversor A/D.....	19
Figura 10 - Possíveis valores do clock do conversor A/D. ....	19
Figura 11 - Código para configuração do prescaler.....	19
Figura 12 - Diagrama de amostras por tempo.....	20
Figura 13 - Código demonstrando o uso da função "Serial.write()" . ....	21
Figura 14 - Código demonstrando o uso da função "Serial.print()".....	22
Figura 15 - Código da interrupção da comunicação Serial. ....	23
Figura 16 - Interface do terminal serial durante testes realizados na placa.....	23
Figura 17 - Exemplo de utilização do vetor. ....	24
Figura 18 - Ambiente IDE do Processing. ....	25
Figura 19 - Modo Contínuo. ....	26
Figura 20 - Aquisição da amostra. ....	27
Figura 21 - Vetor de amostras .....	27
Figura 22 - Aquisição no modo "Amostragem".....	28
Figura 23 - Envio dos dados via comunicação Serial. ....	28
Figura 24 - Rotina de leitura do vetor no Processing.....	29
Figura 25 - Funções do Processing. ....	29
Figura 26 - Esquema do circuito elétrico .....	30
Figura 27 - Circuito final. ....	31
Figura 28 - Determinando fator de aquisição. ....	32
Figura 29 - Teste de velocidade de leitura com prescaler de 16 Mhz. ....	33
Figura 30 - Tempo de envio de um byte. ....	34
Figura 31 - Tempo de estouro do ADCBuffer. ....	35

Figura 32 - Interrupção responsável por gerar o sinal.....	36
Figura 33 - Onda quadrada simulada pelo gerador de sinais do Arduino. ....	36
Figura 34 - Onda distorcida. ....	37
Figura 35 - Comparação do Osciloscópio amador com o profissional - Onda Quadrada. .....	38
Figura 36 - Comparação do Osciloscópio amador com o profissional - Onda Triangular. .....	39
Figura 37 - Comparação do Osciloscópio amador com o profissional - Onda Senoidal. .....	40
Figura 38 - Rotina para leitura da frequência. ....	41
Figura 39 - Onda sem fator de aquisição ....	42
Figura 40 - Onda com fator de aquisição implementado.....	43



# SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b>	<b>11</b>
<b>1.1</b>	<b>TEMA</b>	<b>13</b>
<b>1.2</b>	<b>OBJETIVOS</b>	<b>14</b>
1.2.1	GERAL	14
1.2.2	OBJETIVOS ESPECÍFICOS	14
<b>1.3</b>	<b>JUSTIFICATIVA</b>	<b>14</b>
<b>2</b>	<b>FUNDAMENTAÇÃO TEÓRICA</b>	<b>15</b>
<b>2.1</b>	<b>ARDUINO</b>	<b>15</b>
<b>2.2</b>	<b>CONVERSOR A/D</b>	<b>17</b>
<b>2.3</b>	<b>COMUNICAÇÃO SERIAL</b>	<b>20</b>
2.3.1	TERMINAL SERIAL	23
<b>2.4</b>	<b>LINGUAGEM C/C++</b>	<b>24</b>
2.4.1	VETOR	24
<b>2.5</b>	<b>PROCESSING</b>	<b>25</b>
2.5.1	MODO "TEMPO REAL"	26
2.5.2	MODO DE AMOSTRAGEM	27
2.5.3	FUNÇÕES DO PROCESSING	29
<b>2.6</b>	<b>CIRCUITO ELÉTRICO</b>	<b>30</b>
<b>2.7</b>	<b>FATOR DE AQUISIÇÃO</b>	<b>32</b>
<b>3</b>	<b>RESULTADOS E DISCUSSÕES</b>	<b>32</b>
<b>3.1</b>	<b>TAXA DE CONVERSÃO</b>	<b>33</b>
<b>3.2</b>	<b>TAXA DE TRANSMISSÃO</b>	<b>34</b>
<b>3.3</b>	<b>TEMPO DE ESTOURO DO VETOR ADCBUFFER</b>	<b>35</b>
<b>3.4</b>	<b>TESTES COM GERADOR DE SINAIS</b>	<b>36</b>

<b>3.5</b>	<b>EFICIÊNCIA DO FATOR DE AQUISIÇÃO</b>	<b>42</b>
<b><u>4</u></b>	<b><u>CONCLUSÃO</u></b>	<b><u>43</u></b>
<b><u>5</u></b>	<b><u>MELHORIAS</u></b>	<b><u>44</u></b>
<b><u>6</u></b>	<b><u>REFERÊNCIAS</u></b>	<b><u>45</u></b>

# 1 INTRODUÇÃO

O osciloscópio é um instrumento de medição, que permite visualizar graficamente sinais elétricos. Na maioria das aplicações, o osciloscópio mostra como é um sinal elétrico varia no tempo. Neste caso, o eixo vertical (Y) representa a amplitude do sinal (tensão) e o eixo horizontal (X) representa o tempo.

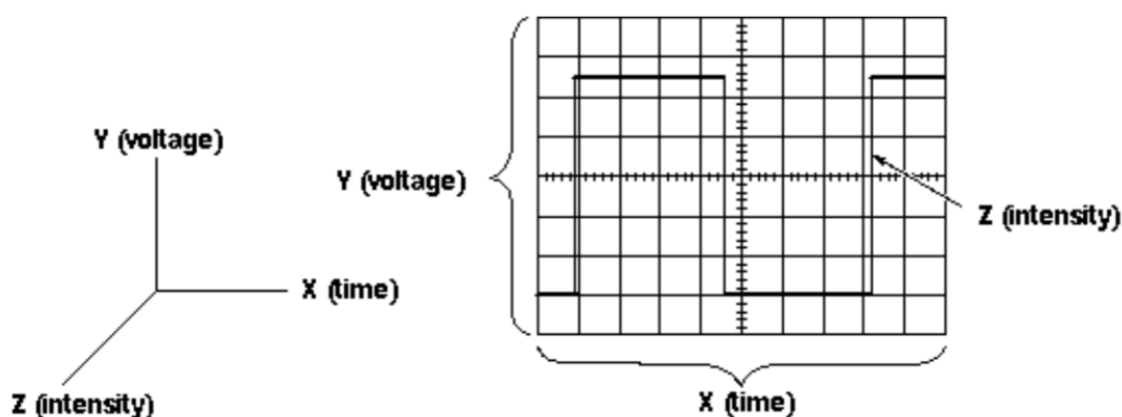


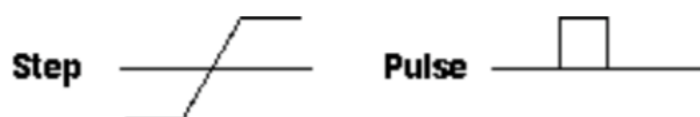
Figura 1 - Eixos X, Y e Z num osciloscópio

Fonte: <http://www.ceset.unicamp.br/~leobravo/TT%20305/O%20Osciloscopio.pdf>

O osciloscópio tem um aspecto que se assemelha a um televisor, com exceção da grade desenhada no écran e a grande quantidade de comandos. O painel frontal do osciloscópio tem os comandos divididos em grupos, organizados segundo a sua funcionalidade. Existe um grupo de comandos para o controle do eixo vertical (amplitude do sinal), outro para o controle do eixo horizontal (tempo) e outro ainda para controlar os parâmetros do écran. [6]

O osciloscópio é utilizado por diversos profissionais, nas mais variadas aplicações, tais quais: reparação de televisores, análise do funcionamento das unidades eletrônicas de controle dos automóveis, a análise de vibrações (de um motor, por exemplo), o projeto de circuitos de condicionamento de sinal ou sistemas biomédicos.

Embora os osciloscópios digitais permitam analisar sinais transitórios (que só acontecem um vez), tal como os apresentados na Figura 2, o osciloscópio é, por princípio, um instrumento de medição adequado a analisar sinais periódicos. [6]



**Figura 2 - Sinais não periódicos**

Fonte: : <http://www.ceset.unicamp.br/~leobravo/TT%20305/O%20Osciloscopio.pdf>

Os sinais periódicos, também denominados de ondas, representam a variação de grandezas que se repetem (periodicamente) no tempo.

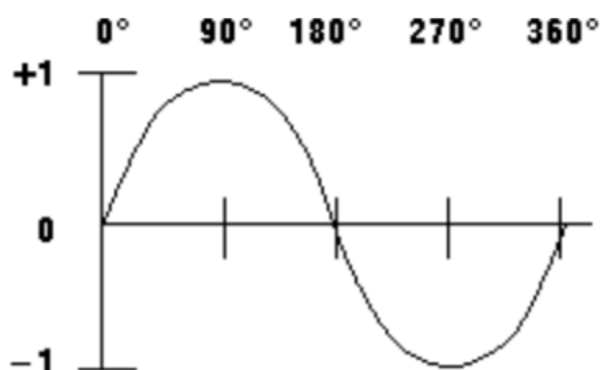


**Figura 3 - Sinais periódicos**

Fonte: : <http://www.ceset.unicamp.br/~leobravo/TT%20305/O%20Osciloscopio.pdf>

Se um sinal se repete no tempo, ele tem uma frequência de repetição. Esta frequência ( $f$ ) é medida em Hertz (Hz) e é igual ao número de vezes que o sinal se repete por segundo (número de ciclos por segundo). Analogamente, um sinal periódico tem um período ( $T$ ), que é o tempo que o sinal leva a completar um ciclo. O período e a frequência são inversos um do outro, isto é,  $f = 1/T$ .

Com um osciloscópio conseguimos medir amplitudes de sinais, nomeadamente amplitudes de pico e pico-a-pico.



**Figura 4 - Amplitude do sinal**

Fonte: <http://www.ceset.unicamp.br>

Os equipamentos eletrônicos podem ser divididos em analógicos ou digitais. Enquanto que o equipamento analógico trabalha com tensões continuamente variáveis, o equipamento digital apenas distingue dois níveis diferentes de tensão (dois níveis lógicos, 0 e 1).

Os osciloscópios também podem ser analógicos ou digitais. Os osciloscópios analógicos constituem de duas placas (placas horizontais) que criam um campo elétrico de acordo com a tensão lida. Provocando assim, o desvio (vertical, dado que as placas são horizontais) de um feixe de eletrons que se desloca para o écran. Isto permite observar o valor da amplitude do sinal no eixo vertical. [6]

Os osciloscópios digitais retiram amostras do sinal original, amostras estas que são convertidas para um formato digital (binário) através da utilização de um conversor analógico/digital. Esta informação digital é armazenada numa memória e seguidamente reconstruída e representada no écran. [6]

Em muitas aplicações, pode utilizar-se tanto um osciloscópio analógico como um digital. Contudo, cada um deles possui características particulares, tornando-os adequados para uma dada tarefa.

Os osciloscópios analógicos eram normalmente preferidos quando era necessário visualizar sinais com variações muito rápidas (altas frequências) em tempo-real. O desenvolvimento dos osciloscópios digitais fez com que os osciloscópios analógicos fossem ultrapassados em respeito à tecnologia.

Os osciloscópios digitais permitem o armazenamento e posterior visualização das formas de onda, nomeadamente de acontecimentos que ocorrem apenas uma vez. Eles permitem ainda processar a informação digital do sinal ou enviar esses dados para um computador para serem processados e/ou armazenados. [5]

## **1.1 Tema**

Atualmente, temos a necessidade, que exige que os sistemas e processos sejam mais eficientes e baratos. A ascensão do mercado asiático no ramo da tecnologia, em especial a China, tem barateado o custo de produção de uma série de aparelhos eletro-eletrônicos.

Sendo assim, torna-se necessário a utilização de novas técnicas, visando menor custo e maior eficiência na operação. A tendência em desenvolver circuitos mais eficientes, motivou esse mercado que é um dos mais precisos hoje em dia, gerando diversos tipos de fabricação e técnicas para seu desenvolvimento.

Este projeto visa, justamente, desenvolver um equipamento que tem um custo relativamente alto, com um preço mais acessível.

## **1.2 Objetivos**

### *1.2.1 Geral*

Desenvolver um osciloscópio com baixo custo de operação e implementação, porém, com alta precisão e faixa de trabalho considerável.

### *1.2.2 Objetivos Específicos*

Visando atingir o objetivo principal, alguns objetivos específicos são requeridos, entre eles:

- Desenvolver um protótipo que seja eficiente, visando menores custos de produção e manutenção.
- Elaborar algoritmos confiáveis que, de uma maneira distribuída, consigam interpretar e executar as informações necessárias.
- Interpretar e disponibilizar os dados de uma forma precisa e organizada.

## **1.3 Justificativa**

Esse trabalho foi feito pensando em desenvolver o conhecimento nessa área da tecnologia e contribuir com a comunidade Open-Source no Brasil.

## 2 FUNDAMENTAÇÃO TEÓRICA

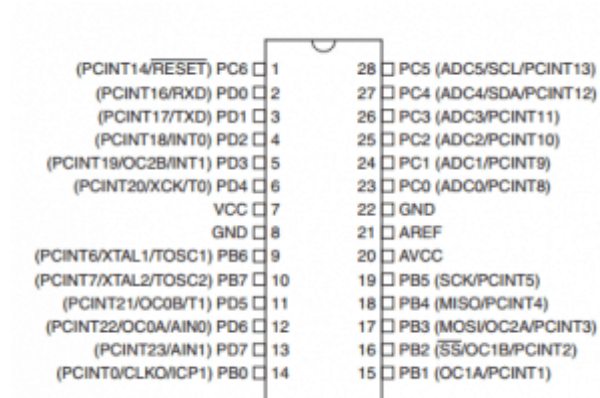
### 2.1 Arduino

É uma plataforma de prototipagem eletrônica, *open-source* e de placa única, projetada com um microcontrolador Atmel AVR. A linguagem de programação é baseada em C/C++. Possui entradas e saídas digitais tanto como analógicas. Sua interação com o controlador, é feita através da interface Serial via USB.



Figura 5 - Layout do Arduino.

Sua placa consiste em um microcontrolador Atmel AVR de 8 bits, com componentes complementares para facilitar a programação e incorporação para outros circuitos. O modelo usado neste trabalho, utiliza um ATmega328 como processador principal e um oscilador de cristal de 16Mhz. Conta com uma memória EEPROM de 1KB, 32KB de Flash e 2KB de RAM. Conta com 28 pinos, sendo que 23 pinos podem ser usados como pinos de I/O.



**Figura 6 - Pinagem Arduino.**

Como periféricos, possui uma USART de até 250kbps uma SPI, que vai a até 5MHz, e uma I2C que pode operar até 400kHz. Conta também, com um comparador analógico interno e 3 TIMERS, além de 6 PWMs. A corrente por pino, deve ser limitada à no máximo, 40mA.

Este modulo ainda possui dois modos de alimentação, via USB e fonte externa (6 ~ 20V), via conector Jack. Para a fonte de alimentação externa é recomendável um valor de entre 7V e 12V, para evitar danos a placa do Arduino. Ambas as formas de alimentação possuem circuitos, na entrada, que tem a função de proteger a placa em caso de alguma anormalidade.

O Arduino possui também, conectores de alimentação para conexão de outros módulos à placa. Saídas de 3,3V , 5V e GND (Terra) são uns dos exemplos de conectores de alimentação encontrados na placa.



**Figura 7 - Conectores de Alimentação.**



Para estabelecer a comunicação USB com o computador, o Arduino possui um microcontrolador ATMEGA16U2. Este, por sua vez, é o responsável pelo upload do código binário ao processador principal. Este módulo possui um conector ICSP para gravação e atualização do firmware do sistema.

## **2.2 Conversor A/D**

O Atmega328 possui internamente um conversor A/D de aproximação sucessivas de 10 bits de resolução com precisão de  $\pm 2$  LSBs. Possui até 8 canais de entradas multiplexados, dependendo do encapsulamento. No caso do Atmega328 PDIP, do Arduino UNO, apresenta apenas 6 canais, como pode-se verificar na placa. Por existir apenas 1 conversor A/D, só poderá ser selecionado 1 canal por vez para conversão, isso é feito através da configuração dos registradores internos. O diagrama de blocos do conversor A/D é exibido a seguir:

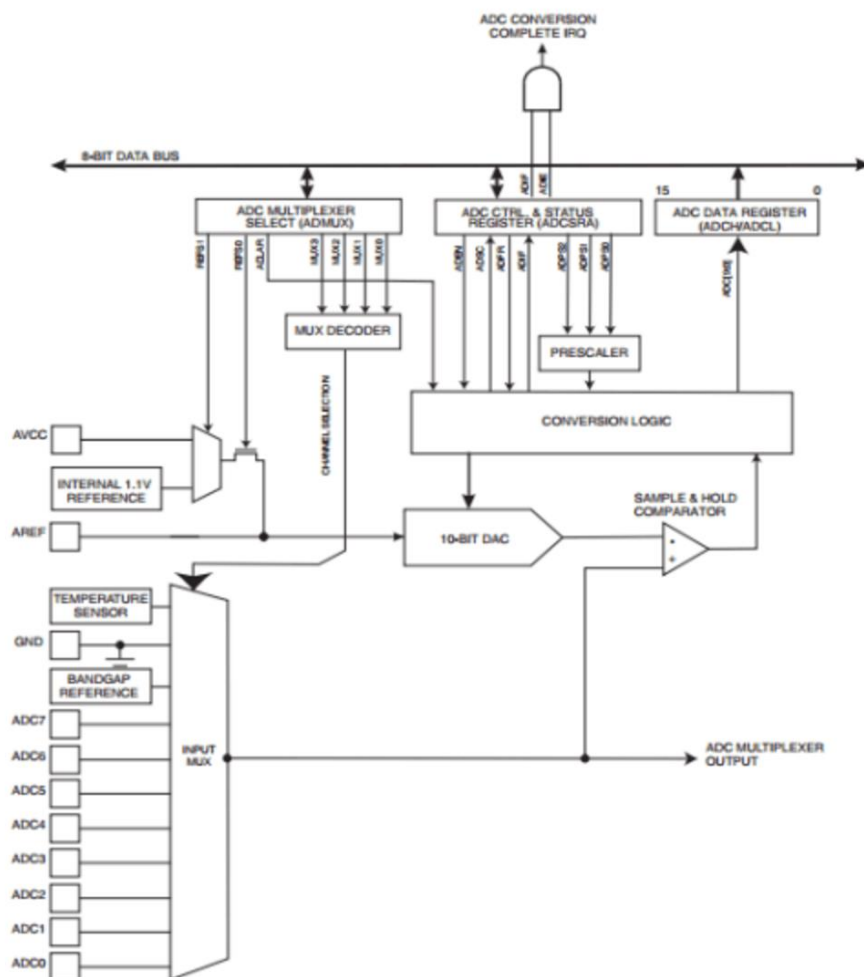
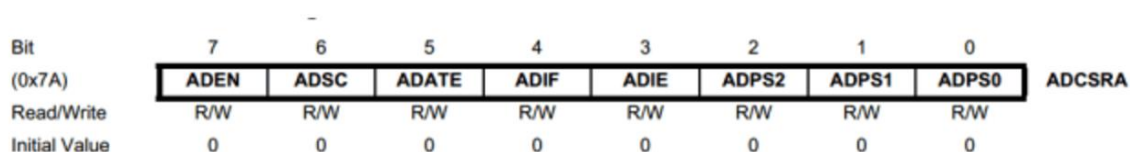


Figura 8 - Diagrama de funcionamento do conversor A/D.

O Atmega328 possui tensão de referência interna de 1,1 V, que pode ser selecionada por software. Apresenta também um pino externo para uma tensão de referência diferente de VCC ou a referência interna de 1,1 V. O valor de tensão de entrada deve estar entre 0V e o valor de tensão de referência, não ultrapassando o valor de VCC. Neste trabalho foi utilizada a referencia externa para leitura do sinal analógico.

A conversão gera um resultado de 10 bits de precisão, necessitando assim, de dois registradores para gravar o valor convertido.

Basicamente, existem dois modos de conversão, o simples e o contínuo. No modo simples é necessário a inicialização de cada conversão. Quando a conversão é finalizada os registradores de dados são preenchidos e o bit **ADIF** é colocado em 1. Para iniciar uma conversão deve-se ligar o bit **ADSC**. Esse bit permanecerá em 1 enquanto a conversão está em processo, e passará para 0 no final da conversão.



**Figura 9 - Possíveis valores do clock do conversor A/D.**

Fonte: <http://www.embarcados.com.br/>

A taxa de conversão, recomendada para este módulo, é de 50kHz a 200kHz para a resolução de 10 bits. O byte do prescaler do clock do oscilador controla o clock do conversor A/D. Deste modo, clock do conversor A/D será uma fração do clock do oscilador principal, conforme o fator do prescaler. Os valores para seleção do clock, são definidos pelo registrador ADCSRA, pelos bits ADPS2:0.

O clock do conversor pode assumir os seguintes valores:

Clico do Oscilador	prescaler	Clock do Conversor A/D
16 MHz	2	8 MHz
	4	4 MHz
	8	2 MHz
	16	1 MHz
	32	500 kHz
	64	250 kHz
	128	125 kHz

**Figura 10 - Possíveis valores do clock do conversor A/D.**

Fonte: Autor.

A configuração do prescaler é feita no início do código, onde seus valores são configurados como constantes. Deste modo, foi possível fazer a alteração do clock do conversor A/D com mais facilidade.

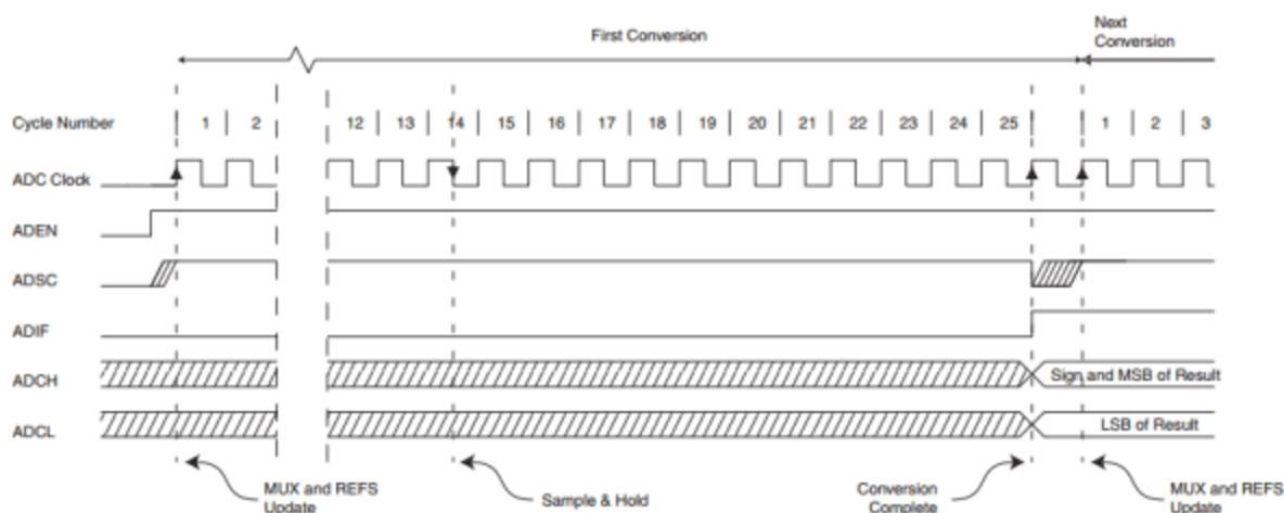
```
const unsigned char PS_16 = (1 << ADPS2);
const unsigned char PS_32 = (1 << ADPS2) | (1 << ADPS0);
const unsigned char PS_64 = (1 << ADPS2) | (1 << ADPS1);
const unsigned char PS_128 = (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
```

**Figura 11 - Código para configuração do prescaler.**

Fonte: Autor.

Para definir a velocidade de clock, basta igualar o valor do registrador ADCSRA aos valores das constantes definidas para os prescalers (PS\_16; PS\_32...). Neste trabalho, foi utilizado um prescaler de 16, ou seja, a maior velocidade possível. Esta prática visa aumentar a faixa de trabalho do osciloscópio, pois, é possível fazer um maior número de leituras num determinado intervalo de tempo.

Uma conversão normal, com prescaler em 128 bits, exige 13 pulsos de clock para finalizar a conversão, Sendo assim, para calcular a quantidade de amostras por segundo que conseguimos obter, basta dividir o valor do clock por 13. No caso do Arduino, a primeira conversão leva 25 pulsos de clock para finalizar. Deste modo, temos a seguir, um diagrama no tempo x amostras:



**Figura 12 - Diagrama de amostras por tempo.**

Fonte: <http://www.embarcados.com.br/>

Os valores obtidos com o prescaler utilizado neste trabalho serão discutidos no tópico "Resultados e discussões"

## 2.3 Comunicação Serial

A placa Arduino UNO é programada através da comunicação serial, pois o microcontrolador vem programado com o bootloader. Dessa forma não há a necessidade de um programador para fazer a gravação (ou upload) do binário na placa. A comunicação é feita através do protocolo STK500.

A UART, mais conhecida como Comunicação Serial, é ligada ao Arduino através dos pinos digitais 0 (RX) e 1 (TX). Esses mesmos pinos estão ligados ao

microcontrolador ATMEGA16U2, responsável por fazer o controle dessa comunicação.

A plataforma Arduino possui em sua biblioteca uma variedade de funções para manipulação de dados através de comunicação serial, são elas:

- **Serial.begin():** essa é a função usada para configuração dos parâmetros da comunicação Serial. No caso do Arduino UNO, temos apenas uma UART disponível. Parâmetros como, taxa de comunicação (bps), bits de paridade e “stop bit” são configuráveis nessa função.

**Sintaxe:** Serial.begin(velocidade), Serial.begin(velocidade, parâmetros)

Neste trabalho, foi utilizado a taxa de bits de 115200 bps. Segundo o datasheet do ATmega328, essa taxa de transmissão é a maior possível sem haver perdas significativas de dados. Nenhum bit de paridade foi utilizado, pois, o próprio código do Arduino ou do Processing, valida a informação na recepção dos dados. Abaixo, exemplo da aplicação no código utilizado:

- **Serial.write():** essa função é utilizada para enviar a informação através do pino TX. A informação é enviada no formato de um byte.

**Sintaxe:** Serial.write(val), Serial.write(str), Serial.write(buf, len).

Nessa função podemos enviar tanto uma variável, quanto um vetor de informações. No caso, usamos essa função para enviar o valor da tensão processada pelo conversor A/D. Devido a informação se limitar a um byte, o valor máximo para a tensão nesse caso, seria de 255. Caso contrário, ultrapassaria dos limites de um byte (conjunto de 8 bits ou  $2^7$ ). Deste modo, como a conversão tem uma definição de 10 bits ( $2^9 = 1024$ ), o valor convertido pelo módulo A/D é dividido por 4, para que a variável possua o valor máximo de 255 ( $2^7$ ) e seja possível enviá-la através da função **Serial.write()**.

```
int val = analogRead(ANALOG_IN); // aquisição da amostra do conversor A/D
Serial.write(val/4);              // envio da informação através da porta Serial
```

**Figura 13 - Código demonstrando o uso da função "Serial.write()".**

**Fonte: Autor.**

- **Serial.print():** essa função também é utilizada para enviar dados, porém, ela os envia no formato ASCII (String).

**Sintaxe:** Serial.print(val)

Com essa função, é possível enviar os mais variados dados. Desde caracteres à frases com espaços e acentos.

Sendo assim, essa função foi utilizada para enviar o vetor que comporta a sequência de leituras do módulo A/D. Os dados que foram guardados em cada posição do vetor “ADCBuffer”, são enviados um a um, separados pela vírgula ( , ). Deste modo, é possível identificar cada leitura como única no receptor. Abaixo, uma amostra do dado enviado com a função Serial.print():

```
for ( int ADCCounter1 = 0; ADCCounter1 <= ADSizeBuffer; ADCCounter1++ ){
    Serial.print( ADCBuffer[ADCCounter1] );          // envio do dado através da porta Serial
    Serial.print(',');                               // caracter delimitador de amostras
}
Serial.println();      // "fim da linha" - indica o final da transmissão para o receptor
```

**Figura 14 - Código demonstrando o uso da função "Serial.print()".**

**Fonte: Autor.**

- **Serial.available():** essa função valida se há informação para ser lida pelo Arduino.

**Sintaxe:** Serial.available()

Essa função retorna apenas o valor de bytes disponível para leitura do Arduino. Sendo assim, foi criada uma interrupção que será acionada toda vez que o valor retornado por essa função for > 0.

Essa funcionalidade foi utilizada para interação do Processing com o Arduino. Esta função, auxiliou na configuração do modo de aquisição disponível no Arduino (tempo real ou amostragem). Esses modos serão abordados ao longo desse documento.

Abaixo, o código utilizado para tratar essa interrupção e seleção do modo de aquisição.

```

void serialEvent(){
  if (Serial.available()){
    CharSerialRX = Serial.read();

    if (CharSerialRX == 'm')
      modoOperacao = !modoOperacao;
  }
}

```

Figura 15 - Código da interrupção da comunicação Serial.

Fonte: Autor.

- **Serial.read():** essa função faz a leitura do byte disponível.

**Sintaxe:** Serial.read()

No exemplo acima, essa função é usada para ler o byte e atribuir o seu valor a uma variável já declarada no programa.

### 2.3.1 Terminal Serial

O IDE trás um terminal serial que auxilia na visualização dos dados que circulam através da porta serial.

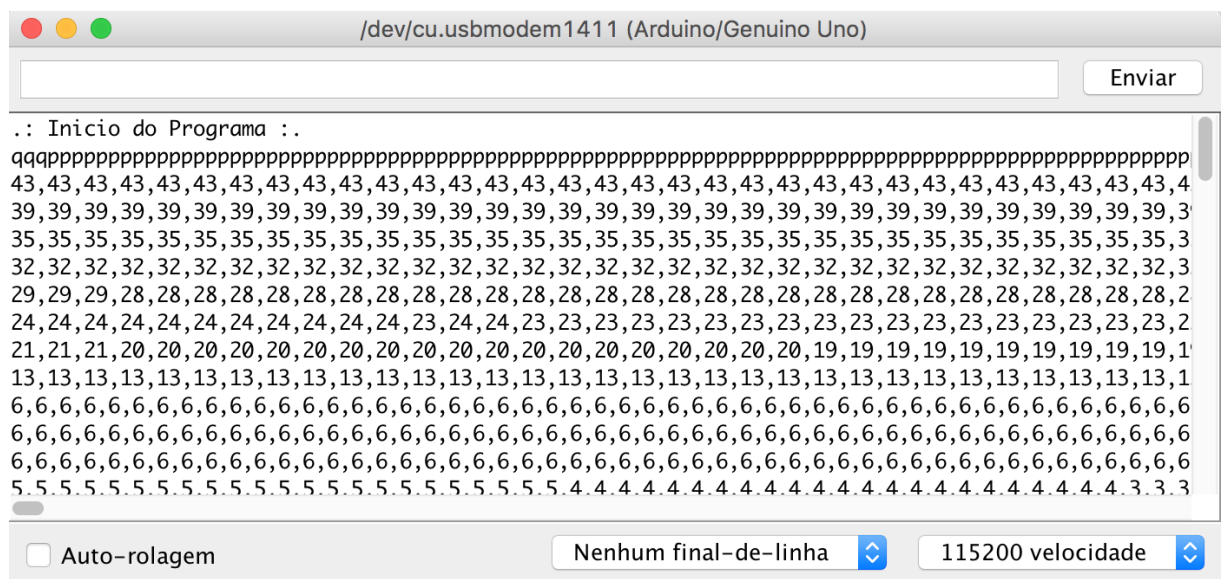


Figura 16 - Interface do terminal serial durante testes realizados na placa.

Fonte: Autor.

Através dessa interface, foi possível interpretar a forma como o Arduino trata os dados na porta serial e também, auxiliou na correção de muitos erros relacionados a porta Serial.

## 2.4 Linguagem C/C++

A finalidade desse tópico é explicar algumas das funcionalidades disponíveis na linguagem C, que foram utilizadas nesse trabalho. Não será abordado aqui quaisquer outros temas relacionados a linguagem C.

### 2.4.1 Vetor

Em C, o vetor é uma estrutura de dados indexada, que pode armazenar uma determinada quantidade de valores do mesmo tipo.

Essa funcionalidade foi explorada a fim de facilitar o acesso aos valores obtidos pelo conversor A/D na leitura do sinal analógico. Cada valor obtido pelo conversor, é guardado em uma posição do vetor.

Guardando as amostras num vetor, conseguimos ler com muito mais velocidade o sinal analógico e assim, aumentar a faixa de leitura do nosso osciloscópio.

Abaixo, um exemplo de utilização de vetor, em linguagem C:

```
byte ADCBuffer[1280]; // declarando o vetor
```

O código acima, tem a função de declarar o vetor para o programa. Este procedimento é feito no "setup" do Arduino. No caso, o vetor se chama "ADCBuffer" e contém 1280 posições disponíveis para salvar dados.

```
for ( int ADCCounter = 0; ADCCounter <= ADSizeBuffer; ADCCounter++ ){  
    //byte value;  
    byte val = analogRead(ANALOG_IN); // leitura do conversor A/D  
    ADCBuffer[ADCCounter] = val/4; // gravando os 8 bits mais significativos da amostra  
}
```

**Figura 17 - Exemplo de utilização do vetor.**

**Fonte: Autor.**

Acima, uma demonstração da utilização do vetor, para gravar as amostras lidas sequencialmente. O laço de repetição "for" tem a finalidade de indexar o vetor a cada

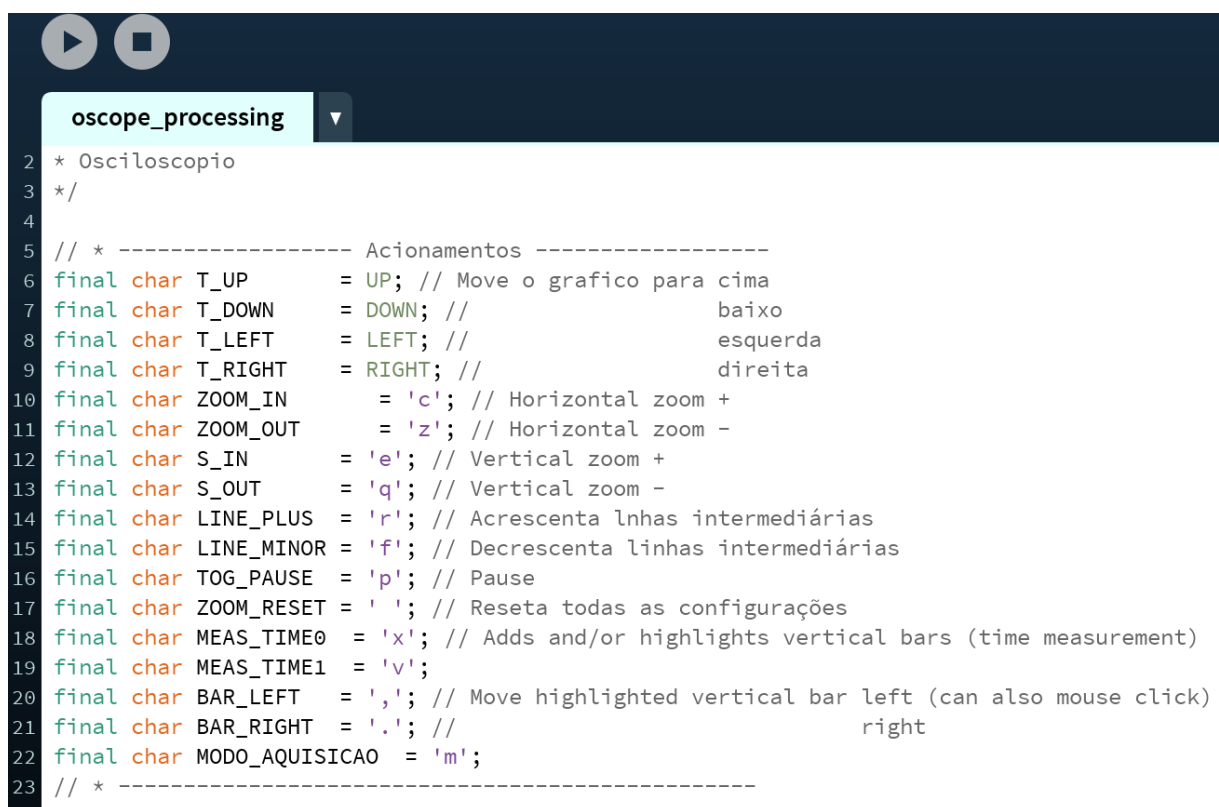


leitura do conversor A/D. Deste modo, conseguimos gravar as amostras numa ordem para depois utilizarmos para plotar o gráfico no Processing.

## 2.5 Processing

Processing é uma linguagem de programação de código aberto e ambiente de desenvolvimento integrado (IDE), assim como no Arduino. Com este software é possível desenvolver códigos capazes de gerar gráficos e/ou imagens na tela do computador.

Essa ferramenta é baseada na linguagem de programação Java, possuindo diversos recursos dessa linguagem. A forma como você escreve o código e trata as variáveis se assemelha bastante à linguagem Java, tornando-o um software bastante completo.



```
2 * Osciloscópio
3 */
4
5 // * ----- Acionamentos -----
6 final char T_UP      = UP; // Move o grafico para cima
7 final char T_DOWN    = DOWN; //
8 final char T_LEFT    = LEFT; //
9 final char T_RIGHT   = RIGHT; //
10 final char ZOOM_IN   = 'c'; // Horizontal zoom +
11 final char ZOOM_OUT  = 'z'; // Horizontal zoom -
12 final char S_IN      = 'e'; // Vertical zoom +
13 final char S_OUT     = 'q'; // Vertical zoom -
14 final char LINE_PLUS = 'r'; // Acrescenta linhas intermediárias
15 final char LINE_MINOR = 'f'; // Decrescenta linhas intermediárias
16 final char TOG_PAUSE = 'p'; // Pause
17 final char ZOOM_RESET = ' '; // Reseta todas as configurações
18 final char MEAS_TIME0 = 'x'; // Adds and/or highlights vertical bars (time measurement)
19 final char MEAS_TIME1 = 'v';
20 final char BAR_LEFT  = ','; // Move highlighted vertical bar left (can also mouse click)
21 final char BAR_RIGHT = '.'; //
22 final char MODA_AQUISICAO = 'm';
23 // * -----
```

Figura 18 - Ambiente IDE do Processing.

Fonte: Autor.

A interface com o Arduino se fez através da comunicação Serial. No caso, o Processing é o responsável por interpretar as amostras do sinal analógico e disponibilizar num gráfico de tensão x tempo.

Como todo código, o Processing tem uma função principal, a qual será responsável por executar todo o código. Esta se chama "draw". É nessa função que o código será processado e o gráfico gerado. (exemplo nos anexos)

De início, o Processing exige que seja definido o tamanho do nosso gráfico. No caso, escolhemos o tamanho de 1280 x 640 pixels. Neste plano, cada amostra será representada por um pixel no eixo horizontal na medida em que estas forem processadas.

Quase que em toda sua atividade, o Processing faz o papel de receptor das mensagens, com exceção do momento em que a ferramenta envia ao Arduino o modo de operação em que o Arduino deve operar.

Para a aquisição e processamento das amostras, temos dois modos de operação disponíveis:

### 2.5.1 Modo "Tempo Real"

O modo "Tempo Real", é a forma de captura e processamento das amostras em tempo real. No Arduino, quando o modo contínuo está selecionado, este, captura a amostra do conversor A/D e envia, em seguida, através da comunicação Serial, o valor para o Processing. Este, por sua vez, processa a amostra e disponibiliza no gráfico imediatamente.

Este modo de aquisição, demonstrou algumas limitações pois, a disponibilidade da informação é quase que em tempo real. Assim que o Arduino captura a amostra, o mesmo já envia e é disponibilizado no gráfico pelo Processing. Sendo assim, este modo funciona apenas para frequências abaixo de 150 Hz. Neste caso, o limite de leitura é imposto pelo hardware do Arduino, o qual tem uma resposta de processamento mais lenta que o Processing.

```
if (!modoOperacao){  
    int val = analogRead(ANALOG_IN);    // aquisição da amostra do conversor A/D  
    Serial.write(val/4);                 // envio da informação através da porta Serial  
}
```

**Figura 19 - Modo Contínuo.**

**Fonte: Autor.**

O código acima, demonstra o modo contínuo de aquisição das amostras. Assim que o Arduino captura a amostra o mesmo já o envia através da comunicação Serial. O valor é dividido por 4, pois, a função "Serial.write" trabalha apenas com bytes, conforme já discutido nesse documento.

No receptor (Processing), há uma função que sempre aguarda um valor para leitura na comunicação Serial. Como o processamento do computador é superior ao do Arduino, muitas vezes o Processing irá verificar a disponibilidade de alguma amostra, porém, o Arduino ainda não mandou. Para este cenário, o valor da amostra no Processing assume o valor de "-1", o que significa que não foi possível capturar nenhuma amostra. Esse procedimento se repete até que o Arduino tenha enviado alguma amostra e seja possível disponibilizá-la no gráfico.

```
void adquiereMedicao() {  
    val = -1; // valor default caso nao tenha nenhuma amostra disponivel  
    if ( !modoOperacao ){  
        while (port.available () > 0) {  
            val = port.read(); // leitura da amostra salva na variável "val"  
            val /= 4; // variavel multiplicada por 4  
        }  
    }  
}
```

**Figura 20 - Aquisição da amostra.**

**Fonte: Autor.**

A linha "while (port.available () > 0)" verifica se há alguma amostra para ser lida.

Após a captura da amostra, esta é enviada a um vetor que guarda as amostras numa ordem linear. Sempre que uma nova amostra é processada, o código acrescenta seu valor à última posição do vetor, limitado a 1280 posições (tamanho horizontal da tela) e imprime o novo vetor com o valor acrescido na tela.

```
void plotaMedicoesContinuo(int value) {  
    for (int i=0; i<width-1; i++){  
        medicoes[i] = medicoes[i+1]; //adianta uma posição para todas as amostras guardadas no vetor  
    }  
    medicoes[width-1] = value; //adiciona na ultima posição do vetor a última amostra lida  
}
```

**Figura 21 - Vetor de amostras**

**Fonte: Autor**

A função que imprime os valores na tela, é parte da função principal "draw()".

### 2.5.2 Modo de Amostragem

Este modo de operação foi desenvolvido pensando em resolver as limitações de funcionamento do modo "Tempo Real". O grande desafio aqui foi, aumentar a faixa de frequência que o osciloscópio poderia obter uma leitura com precisão. Para isso, foi perdida a característica de tempo real que tínhamos no modo anterior.

Neste modo, a aquisição das amostras pelo Arduino é feita de maneira repetitiva, levando em média 20 us entre uma leitura e outra. A diferença está no modo como é armazenada essas amostras e como é feito o seu envio. Para aumentar a frequência de leitura, foi usado um vetor (do tipo byte) para armazenar as amostras obtidas. O microcontrolador irá armazenar essas amostras até que o limite de posições do array seja atingido. O limite do nosso array é, exatamente, o tamanho horizontal do nosso gráfico no Processing, ou seja, 1280 posições.

```
if (modoOperacao){
  for ( int ADCounter = 0; ADCounter <= ADSizeBuffer; ADCounter++ ){ // laço de repetição para limitar o tamanho máximo do vetor
    byte val = analogRead(ANALOG_IN); // leitura do conversor A/D
    ADCBuffer[ADCounter] = val/4; // considera apenas os 8 bits mais significativos
  }
}
```

**Figura 22 - Aquisição no modo "Amostragem".**

**Fonte: Autor.**

Após obtida as 1280 amostras, o Arduino irá enviar pela comunicação Serial, todas as amostras gravadas no vetor. Para essa tarefa, utilizamos a função "Serial.print()", a qual envia os dados no formato de texto. Contudo, como as amostras estão sendo enviadas em formato de texto, uma na sequência da outra, o receptor não consegue identificar o valor exato de cada uma.

```
for ( int ADCounter1 = 0; ADCounter1 <= ADSizeBuffer; ADCounter1++ ){
  Serial.print( ADCBuffer[ADCounter1] ); // envio do dado através da porta Serial
  Serial.print(','); // caracter delimitador de amostras
}
Serial.println(); // "fim da linha" - indica o final da transmissão para o receptor
}
```

**Figura 23 - Envio dos dados via comunicação Serial.**

**Fonte: Autor.**

A fim de separar os valores no texto enviado, é acrescentado uma vírgula após o envio de cada amostra. Deste modo, o receptor tem um delimitador para identificar e processar as amostras separadamente. No final da transmissão do vetor, é enviado uma quebra de linha "Serial.println()" para que o receptor identifique o final da mensagem.

```

if ( modoOperacao && !pause ){
    val = 0;
    String valores = port.readStringUntil('\n');    // leitura do texto enviado.
    if (valores != null){                          // se o valor for diferente de vazio
        valores = trim(valores);                  // remove caracteres em branco
        ADCBuffer = int(split(valores, ','));      // separa por "," as amostras ao longo do vetor "ADCBuffer"
    }
}

```

**Figura 24 - Rotina de leitura do vetor no Processing.**

**Fonte: Autor.**

O Processing interpreta o texto enviado e guarda seus valores num vetor “ADCBuffer”, separando pela vírgula. Após essa rotina, o código plota os gráfico utilizando esse vetor como referencia. Como o vetor contém 1280 posições, toda a tela é preenchida com as amostras recebidas. O tempo médio para todo esse processo acontecer é de 20 ms.

### 2.5.3 Funções do Processing

Função	Tecla
Horizontal Zoom +	c
Horizontal Zoom -	z
Vertical Zoom +	e
Vertical Zoom -	q
Linhas intermediárias +	r
Linhas intermediárias -	f
Pause	p
Barra de tempo +	x
Barra de tempo -	v
Mover barra de tempo	Mouse
Modo de Operação	m

**Figura 25 - Funções do Processing.**

**Fonte: Autor.**

## 2.6 Circuito Elétrico

O circuito elétrico tem a finalidade de preparar o sinal analógico para ser lido pelo Arduino. Uma das limitações do Arduino é que o leitor analógico "analog.Read", apenas consegue interpretar níveis de tensão que estejam entre 0 e +5V.

Sendo assim, foi desenvolvido um esquema elétrico capaz de manipular o sinal analógico e torná-lo acessível para o microcontrolador. O circuito é constituído de dois divisores de tensão e um potenciômetro para controle do fator de aquisição. (o fator de aquisição será abordado no item 6).

Segue abaixo, o esquema elétrico desenvolvido para esse projeto:

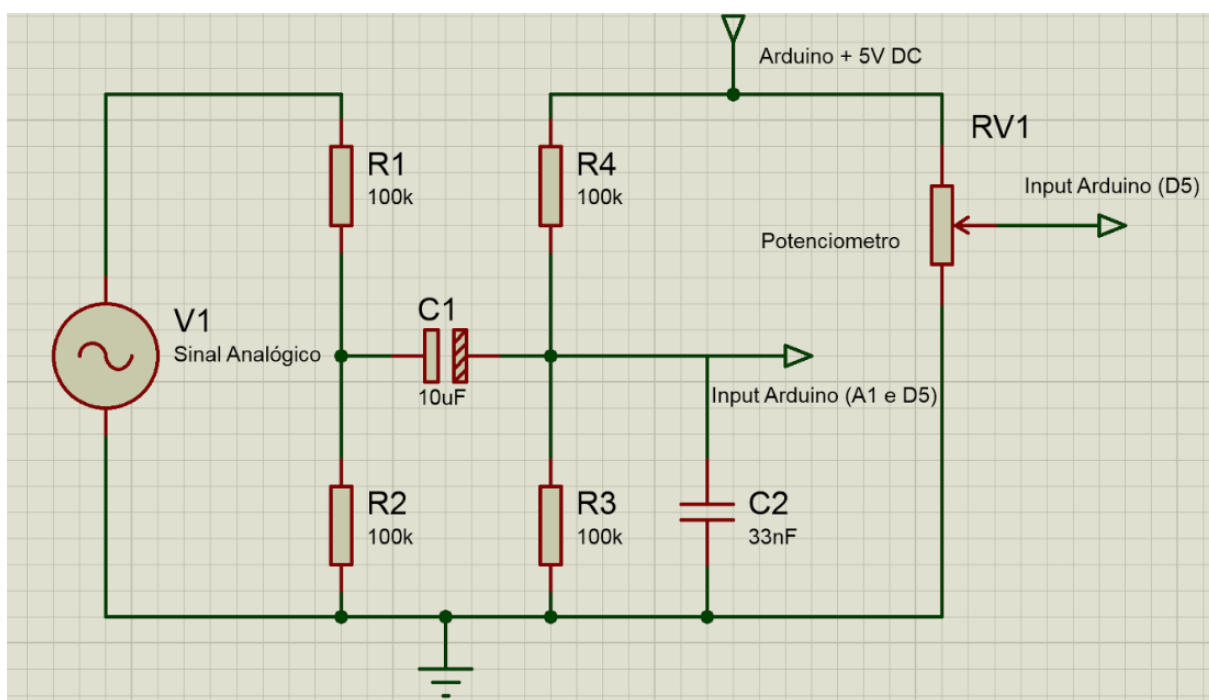


Figura 26 - Esquema do circuito elétrico

Fonte: Autor

Para aproveitar o máximo da faixa de leitura do Arduino, 0 a 5 volts, devemos fazer com que o sinal analógico oscile exatamente no meio dessa faixa, ou seja, em torno de +2,5V. Para isso, foi implementado um sinal 2,5V DC, junto a saída do sinal analógico, o qual tem a função de "DC Offset".

O sinal DC, quando combinado a um AC, tem a função de elevar o nível de tensão em qual a componente AC oscila. Sendo assim, o nível de 2,5V se torna o

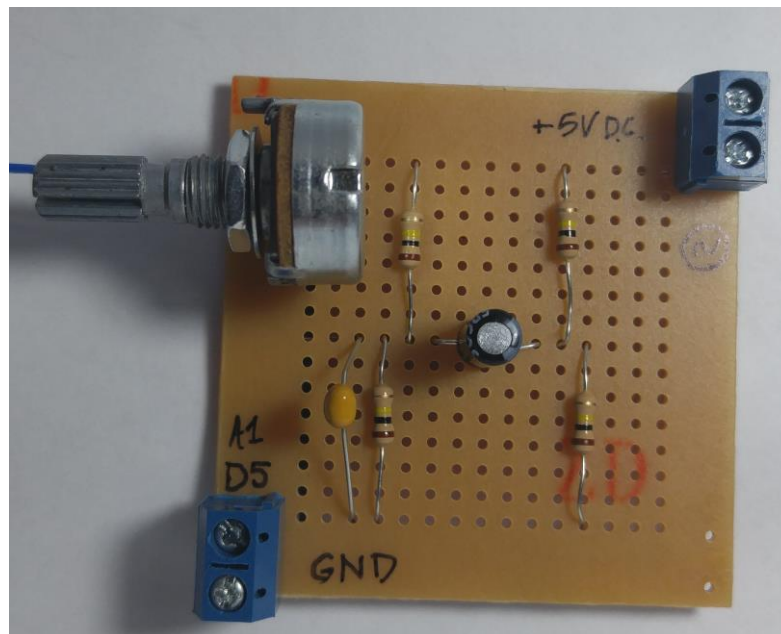
centro de oscilação do sinal AC e não mais a referência ( 0V ). Deste modo, podemos ter uma componente AC de até 2,5V de pico, ou 5V de Vpp.

A intensidade de +2,5V DC é obtida através do divisor de tensão do lado direito, o qual possui dois resistores de 100 kOhms combinados com uma tensão de +5V DC originada do Arduino.

Os capacitores tem apenas a função de filtrar o sinal, evitando ruídos durante a passagem do sinal pelo circuito.

O potenciômetro do lado direito do circuito, é o controle do fator de aquisição no modo de "Amostragem". Este, está ligado ao +5 V do Arduino e a outra ponta no terra. Esse tema será detalhado no próximo item.

O circuito foi montado em uma base de fenolíte, previamente furada, onde só foi preciso posicionar os componentes e soldá-los. As trilhas foram criadas através da ligação dos componentes por solda e também, aproveitando as conexões disponíveis na placa. Segue abaixo, a imagem do circuito final:



**Figura 27 - Circuito final.**

**Fonte: Autor.**

## 2.7 Fator de aquisição

Visando adquirir a maior taxa de leitura de frequência possível, foi implementado, no "Modo de Amostragem" um artifício para se controlar o fator de aquisição.

O fator de aquisição nada mais é que um intervalo de tempo que o microcontrolador irá aguardar entre uma amostra e outra. Esse fator de aquisição é determinado manualmente, via potenciômetro. Este potenciômetro tem a função de controlar o nível de tensão que chega ao pino A5 do Arduino. Este, por sua vez, faz uma leitura "**analog.Read**" do nível de tensão e atribui um valor para intervalo de tempo, seguindo uma regra de 3 básica. Segue abaixo, uma amostra do código:

```
// Determinando o fator de aquisição -----  
potdelay = analogRead(A5);  
potdelay = map(potdelay, 0, 1023, 0, 200);  
//Serial.println(potdelay);  
if ( potdelay <= 3 ){ // enquanto a tensão no potenciômetro for menor que 0,37V, a variável "potdelay" adquire valor NULO.  
    potdelay = 0;  
}  
// -----
```

**Figura 28 - Determinando fator de aquisição.**

**Fonte: Autor.**

A variável "potdelay" contém o valor de tempo, em microsegundos, que o Arduino irá aguardar até adquirir uma nova amostra. Essa tática, tem a finalidade de aumentar o intervalo de tempo entre uma amostra e outra. Isso faz com que para frequências mais baixas (20 à 1 kHz) o modo de amostragem também seja eficiente, pois, estamos aumentando o espaçamento entre a aquisição de amostra e outra.

Os resultados dessa implementação serão discutidos no item 6.

## 3 Resultados e discussões

Este capítulo tem como objetivo, apresentar a eficiência do projeto proposto. Alguns parâmetros como, faixa de operação, velocidade de transmissão, capacidade da comunicação Serial, velocidade de processamento, tempo de estouro do vetor ADCBuffer, serão abordados nesse item. E por fim, expor algumas melhorias observadas durante o desenvolvimento deste trabalho.



### 3.1 Taxa de conversão

Um dos parâmetros mais críticos do projeto é a velocidade do clock do Arduino. Este modelo, conta com um oscilador de 16 Mhz, o que nos gera um ciclo de máquina de 0,25 us. Ou seja, para cada instrução, o processador irá levar 0,25 us para completá-la. Lembrando que, algumas tarefas levam mais de um ciclo de máquina, como por exemplo, o próprio conversor A/D.

O conversor A/D, tem uma velocidade de clock baseada no clock principal do Arduino, sendo uma parcela desta. O conversor, leva em média 110 us, para converter uma amostra analógica em digital. Porém, podemos diminuir esse tempo, utilizando o prescaler do conversor, conforme comentado anteriormente.

Para se ter uma ideia, o tempo de leitura, cai 80% se compararmos a velocidade com o prescaler configurado no máximo e no mínimo. Segue uma amostra:

```
}

void loop() {

  if (!modoOperacao){
    double inicio = micros();
    int val = analogRead(ANALOG_IN); // aquisição da
    double finale = micros();
    Serial.println ( finale - inicio );
  }
}
```

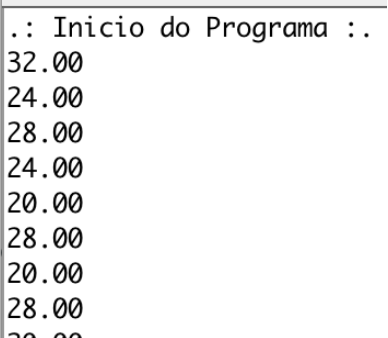


Figura 29 - Teste de velocidade de leitura com prescaler de 16 Mhz.

Fonte: Autor.

Com o prescaler configurado em 16, temos uma velocidade de leitura de 20 us para cada amostra. Foi possível obter esse valor experimentalmente, através do auxílio da função “micros()”. A função **micros()**, contém o tempo de execução do programa, conforme demonstrado no exemplo. Do lado direito da tela, temos os tempos de conversão. A primeira conversão levou 32 us. Este tempo maior, está previsto no datasheet do Arduino, onde o fabricante expõe que na primeira conversão, o hardware leva mais tempo para estabilizar.

É possível aumentar a velocidade com que o conversor opera, porém, a confiabilidade na resolução diminui. Para frequências acima de 1 Mhz, recomenda-se utilizar menos que 10 bits de resolução. Como nesse projeto foi considerado apenas os 8 bits mais significantes, o aumento do prescaler do oscilador não demonstrou ser

um problema. Deste modo, utilizamos uma velocidade de 2 Mhz para o oscilador. Podendo até ser aumentada, caso necessário.

### 3.2 Taxa de transmissão

A taxa de transmissão está diretamente relacionada a velocidade da comunicação serial, ou seja, os bps (bits por segundo). Comumente, usa-se 9600 bps para se estabelecer uma comunicação serial entre dois hardwares. Contudo, neste caso, foi preciso aumentar essa velocidade para que a resposta do sistema fica-se mais rápida e eficiente.

Como já foi comentado anteriormente, a velocidade de operação do Arduino é inferior ao processamento do computador (onde está rodando o Processing). Sendo assim, a análise foi voltada exclusivamente para o Arduino, a fim de identificar os limites de operação do sistema.

Segundo a documentação do Arduino, o fabricante garante uma velocidade de transmissão, sem perda de informação, de até 115200 bps. Com essa taxa de transmissão, foi observado que, para se transmitir um byte, o microncontrolador levava 60 us.

```
void loop() {  
  
  if (!modoOperacao) {  
    //double inicio = micros();  
    int val = analogRead(ANALOG_IN); // aquisição da amostra  
    //double finale = micros();  
    //Serial.println ( finale - inicio );  
    double inicio = micros();  
    Serial.write(val/4); // envio da informação  
    double finale = micros();  
    Serial.println ( finale - inicio );  
  }  
}
```

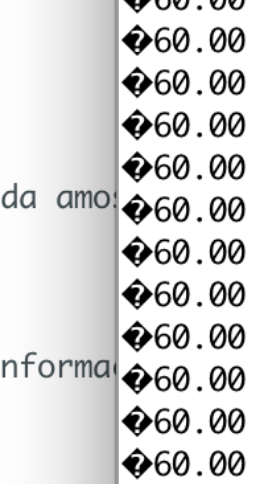


Figura 30 - Tempo de envio de um byte.

Fonte: Autor.

Ao comparar esse valor ao da taxa de conversão, verificou-se que, o tempo para se transmitir a informação era maior do que para se obter a amostra. Este fato, estimulou a busca por aumentar a taxa de transmissão, melhorando assim, a resposta do sistema.

Após algumas pesquisas e testes, foi possível aumentar a velocidade de transmissão até 500000 bps. Não é recomendado pelo fabricante utilizar velocidades dessa grandeza, porém, observou-se que não havia perdas de informação. Utilizando essa velocidade, foi possível transmitir um byte em 5 us, ou seja, um ganho de 77% na velocidade de transmissão.

Com essa prática, foi possível diminuir o tempo de envio, fazendo com que este fosse menor do que a taxa de conversão. (20 us).

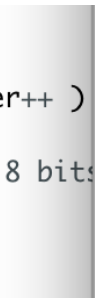
Desta forma, temos uma releitura do sinal analógico, a cada 27 us, em média. Sendo, 20 us para adquirir a amostra + 5 us para se transmitir o dado + 2 us de processamento das rotinas do Arduino. Sendo assim, uma nova amostra é adquirida a cada 27 us.

### 3.3 Tempo de Estouro do vetor ADCBuffer

O vetor ADCBuffer é responsável por guardar os valores das amostras quando o modo de "Amostragem" está ativo. Este vetor possui 1280 posições, sendo que cada posição representa uma amostra. Sendo assim, foi calculado o tempo de estouro deste, ou seja, quanto tempo o Arduino leva para ler e armazenar as 1280 amostras.

Para isso, foi usado a função "millis()" para auxiliar na contagem do tempo. No exemplo abaixo, podemos ver o funcionamento desta função:

```
if (modoOperacao) {
  double inicio = millis(); // tempo de inicio
  for ( int ADCCounter = 0; ADCCounter <= ADSizeBuffer; ADCCounter++ )
    int val = analogRead(A0); // leitura do conversor A/D
    ADCBuffer[ADCCounter] = val/4; // considera apenas os 8 bits
  }
  double finale = millis(); // tempo de fim
  Serial.println(finale - inicio); // diferença de tempo
```



**Figura 31 - Tempo de estouro do ADCBuffer.**

**Fonte: Autor.**

No caso, a função millis(), retorna o valor de tempo em milissegundos. Sendo assim, o Arduino levou em média, 14 ms para processar as 1280 amostras.

Para calcular o tempo de aquisição de cada amostra, basta dividir o tempo de estouro do vetor, pela quantidade de amostras (1280). Temos então, um tempo médio de 11 us para o processamento completo de cada amostra.

Esse valor ainda pode ser baixado para 5 us para cada amostra. Basta diminuir ainda mais o prescaler. Contudo, o sistema se mostrou ineficiente em algumas amostras utilizando prescaler muito baixos. Sendo assim, ficou fixado o valor de 4 MHz para o clock do conversor A/D, pois, foi o valor mais rápido e o qual entregou mais confiança.

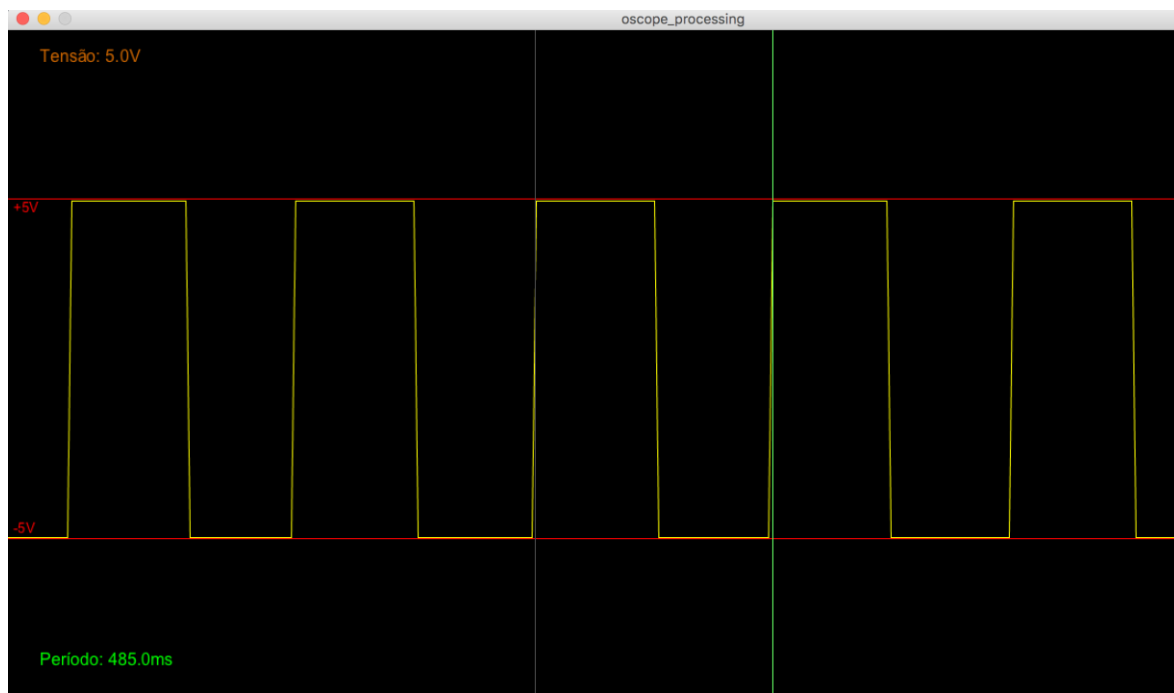
### 3.4 Testes com gerador de sinais

No início, os testes para a leitura do sinal, foram realizados sem o auxílio de um verdadeiro gerador de sinais. Foi implementado no código fonte, uma rotina para controlar o sinal digital numa das portas de I/O do microcontrolador. Essa rotina tinha a simples função de trocar o sinal digital de tempos em tempos, fazendo com que a forma de onda naquela porta oscilasse de 0 para 1. Esse efeito gera um sinal no formato de onda quadrada.

```
ISR(TIMER1_COMPA_vect) {  
    digitalWrite(DIGITAL_OUT, digitalRead(DIGITAL_OUT) ^ 1); // inverte o estado da porta digital 13,  
                                                                // a cada intervalo de tempo estipulado por OCR1A.
```

**Figura 32 - Interrupção responsável por gerar o sinal.**

Fonte: <https://www.embarcados.com.br>.



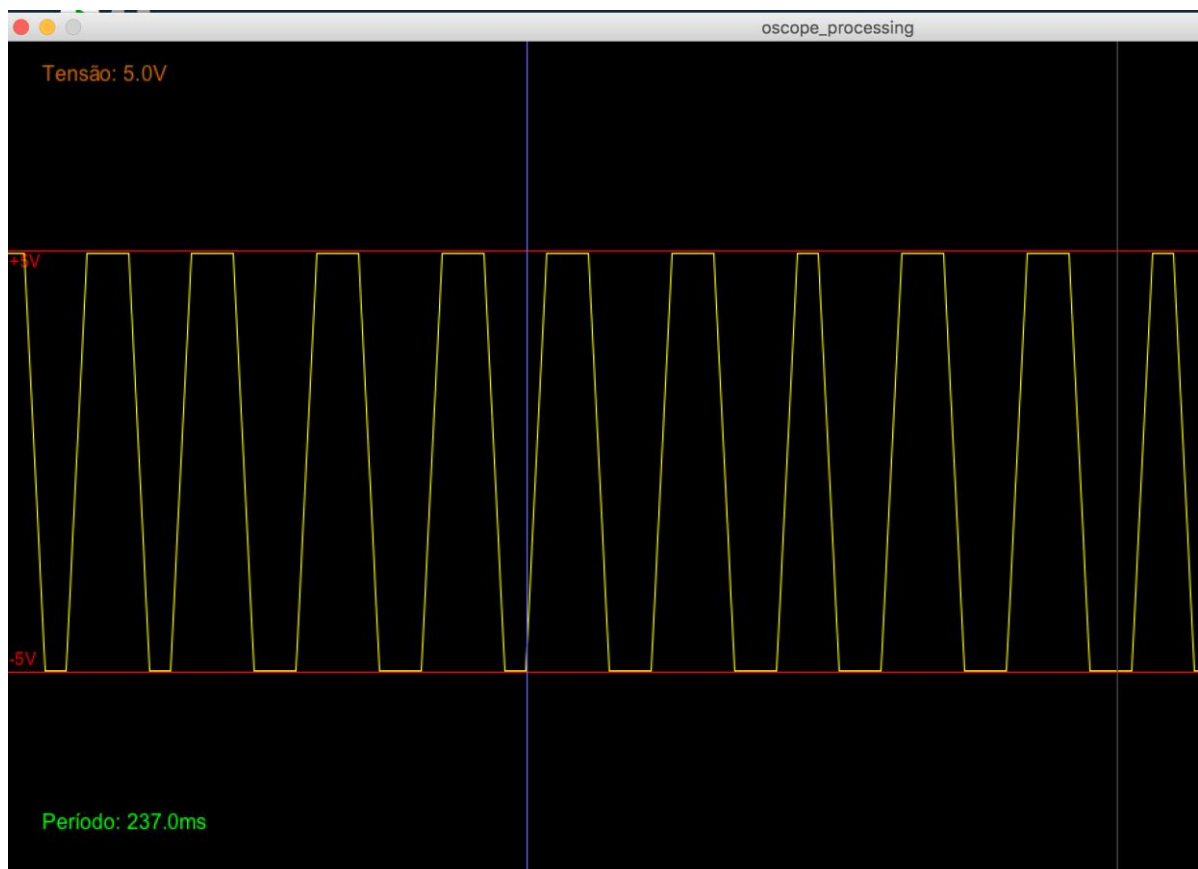
**Figura 33 - Onda quadrada simulada pelo gerador de sinais do Arduino.**

Fonte: Autor.

A imagem acima demonstra o sinal gerado pela rotina de interrupção do próprio Arduino. Este sinal se repete a cada 485 ms, conforme a imagem.

O período foi adquirido através da diferença entre as duas linhas verticais desenhadas sobre a onda. Cada ponto da tela, contém um valor de tempo fixado pelo vetor "vetorTempo". Essa forma de medir o tempo funciona apenas para o modo de Operação "Tempo Real".

Não foi possível gerar sinais com uma frequência mais elevada, pois, verificou-se que o desempenho do Arduino caía consideravelmente. Nestes casos, a onda ficava distorcida, conforme abaixo:



**Figura 34 - Onda distorcida.**

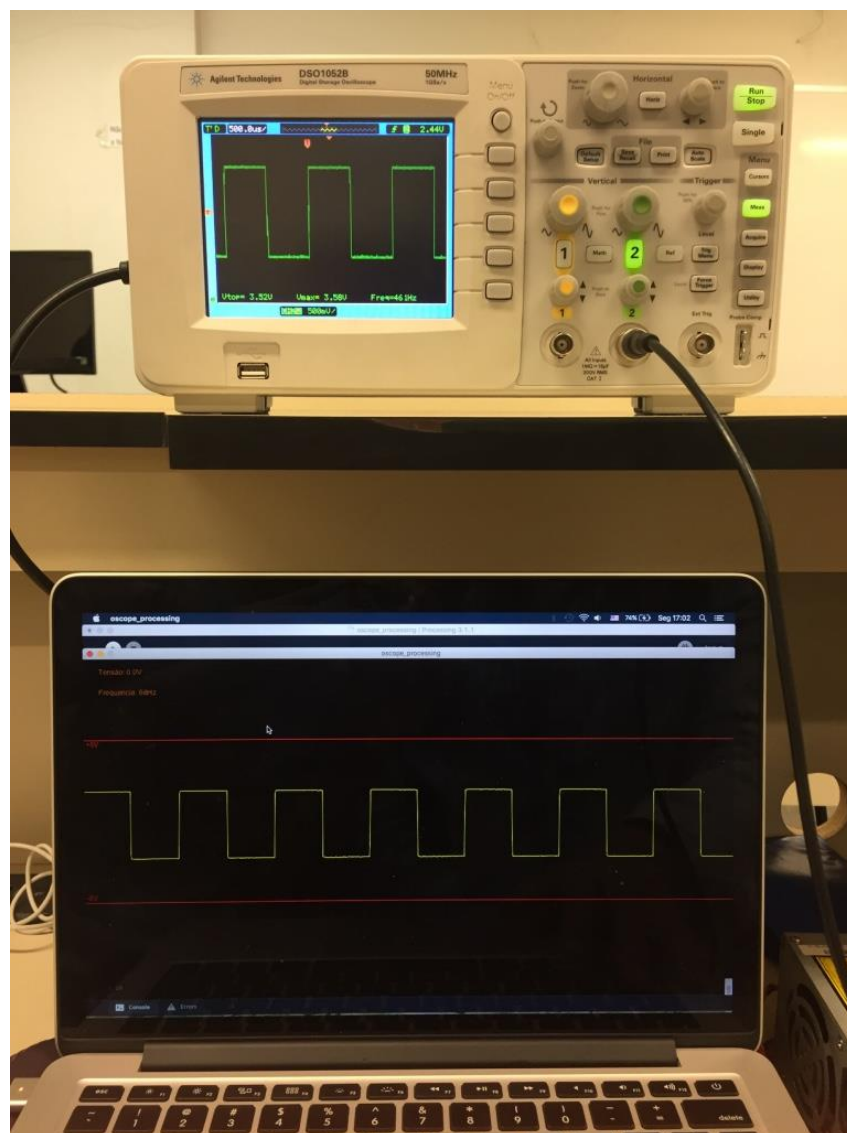
**Fonte: Autor.**

Devido ao próprio microcontrolador estar simulando a onda que irá ler, isso causa um atraso na leitura da onda, a qual fica distorcida. Este modo de leitura demonstrou uma onda nítida na tela até 25 Hz. A explicação para essa baixa eficiência é devido ao próprio microcontrolador estar gerando a onda e que o envio do dado e a

leitura da onda são feitas intercaladas, ou seja, a cada amostra obtida, esta é enviada para o Processing na sequência. Essa lógica cria um sistema com um tempo de resposta imediato, pois, a amostra é imprimida na tela logo depois da sua aquisição.

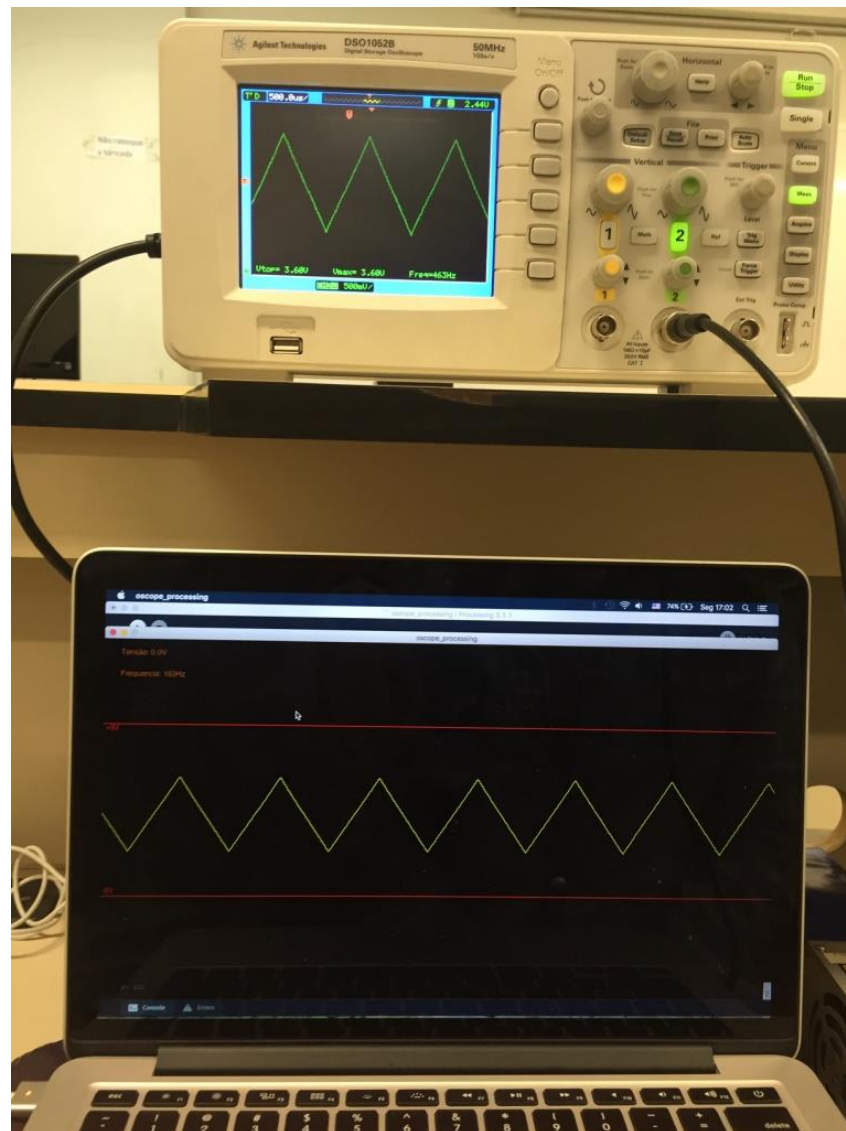
Sendo assim, o modo "Tempo Real" demonstrou-se eficiente para sinais de baixas frequências.

Também foram realizados testes com o gerador de sinais do laboratório de circuitos da Fatec-SP. Este teste tinha como objetivo demonstrar o funcionamento do projeto ao lado de um osciloscópio profissional. Foram simuladas as mais diversas ondas como: triângulares, quadradas e senoidais para aferir sobre a eficiência do projeto. Segue abaixo, alguns exemplos desses sinais gerados que foram captados pelo osciloscópio amador:



**Figura 35 - Comparação do Osciloscópio amador com o profissional - Onda Quadrada.**

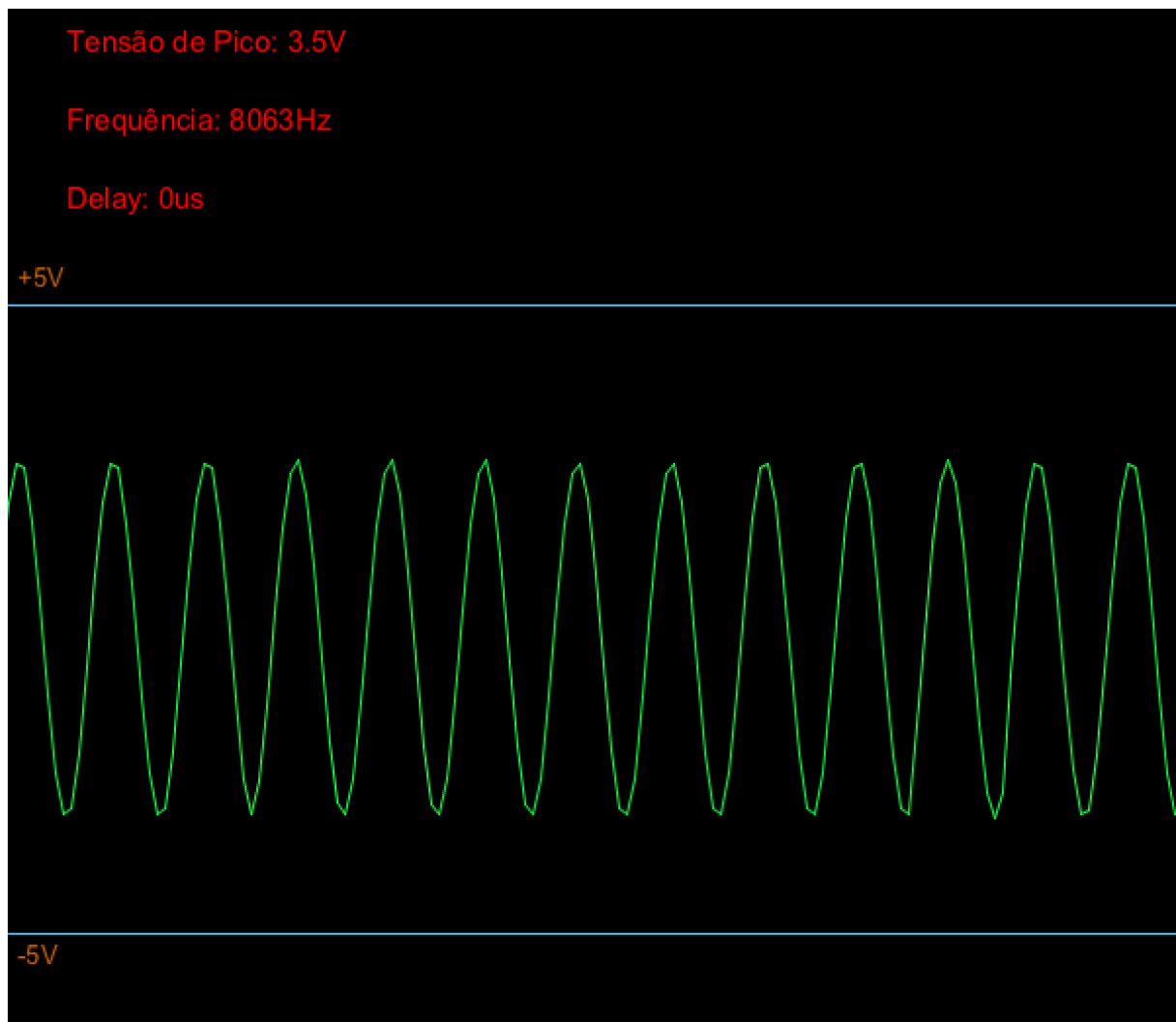
**Fonte: Autor.**



**Figura 36 - Comparação do Osciloscópio amador com o profissional - Onda Triangular.**

**Fonte: Autor.**





**Figura 37 - Comparação do Osciloscópio amador com o profissional - Onda Senoidal.**

**Fonte: Autor.**

Essas imagens foram captadas utilizando o modo de “Amostragem” para interpretar os sinais do gerador. Esse modo demonstrou-se mais eficiente pois, conseguiu interpretar sinais de 4 Hz até 12 kHz, dependendo do formato da onda.

Devido ao seu formato mais abrupto, a onda triangular começou a distorcer para sinais com frequências acima de 9 kHz. Já a onda quadrada se manteve nítida até os 10 kHz. A senoidal foi o formato que obteve os melhores resultados no osciloscópio amador, podendo ser interpretada até 12 kHz.

A medição da frequência neste modo de operação é feita através do auxílio da biblioteca "FreqCount.h". Antes de completar o vetor ADCBuffer com as 1280 amostras, o Arduino faz a leitura da frequência da onda, através da porta digital 5, e salva na variável "frequencia" e envia o valor para o Processing.



```
//Código do medidor de frequência adaptado do site "https://www.pjrc.com/teensy/td\_libs\_FreqCount.html"---
while (FreqCount.available()) {
    frequencia = FreqCount.read();
    if ( frequencia == NULL ){
        //Serial.println("Não foi possível medir a frequência");
    }
    //else Serial.println(frequencia);
}
//-----

for ( int ADCounter1 = 0; ADCounter1 <= ADSizeBuffer; ADCounter1++ ) {
    Serial.print( ADCBuffer[ADCounter1] );          // envio do dado através da porta Serial
    Serial.print(',');                             // caracter delimitador de amostras
}
Serial.print(frequencia);
```

**Figura 38 - Rotina para leitura da frequência.**

**Fonte: Autor.**

Este, por sua vez, processa essa informação junto com as 1280 amostras e apresenta o valor no canto superior esquerdo da tela juntamente com o formato da onda.

O valor da frequência demonstrou ter uma precisão bem acurada quando comparada ao valor apresentado no osciloscópio profissional. O erro percentual ficou até 5% nas diversas comparações feitas. Assim como todas as outras, a biblioteca para medição da frequência também consome processamento do microcontrolador. Deste modo, observou-se que ao inibir a medição da frequência a faixa de leitura de frequência aumentava em até 10%, ou seja, o osciloscópio amador foi capaz de interpretar com clareza, frequências pouco acima de 12 kHz.

Contudo, apesar desse modo de operação ter uma faixa de frequência muito maior que o primeiro, este, não representa exatamente o formato da onda naquele determinado instante. O atraso em relação ao sinal foi calculado com o auxílio da função "millis()", cujo funcionamento já foi abordado neste relatório. O valor retornado por essa função foi de 4 ms de diferença entre o sinal apresentado na tela e o real. Este valor é imperceptível ao ser humano, porém, é um valor que devemos considerar em caso de aplicações mais precisas.

### 3.5 Eficiência do fator de aquisição

O fator de aquisição foi um artifício que ajudou a ampliar a faixa de frequência em que podemos utilizar o osciloscópio. Devido ao delay, gerado pela intensidade de tensão lida no potenciômetro, podemos aumentar a faixa de frequência consideravelmente, pois, o tempo de preechimento do vetor ADCBuffer fica maior. Isso funciona muito bem para as frequências mais baixas. Anteriormente a implementação desse módulo, o osciloscópio só conseguia um período completo de onda, a partir de 1 kHz. As frequências mais baixas que esse valor, apresentavam um sinal incompleto e difícil de ser interpretado. Segue abaixo, uma demonstração dessa utilidade. A onda usada como exemplo é uma senoidal de 14 Hz.

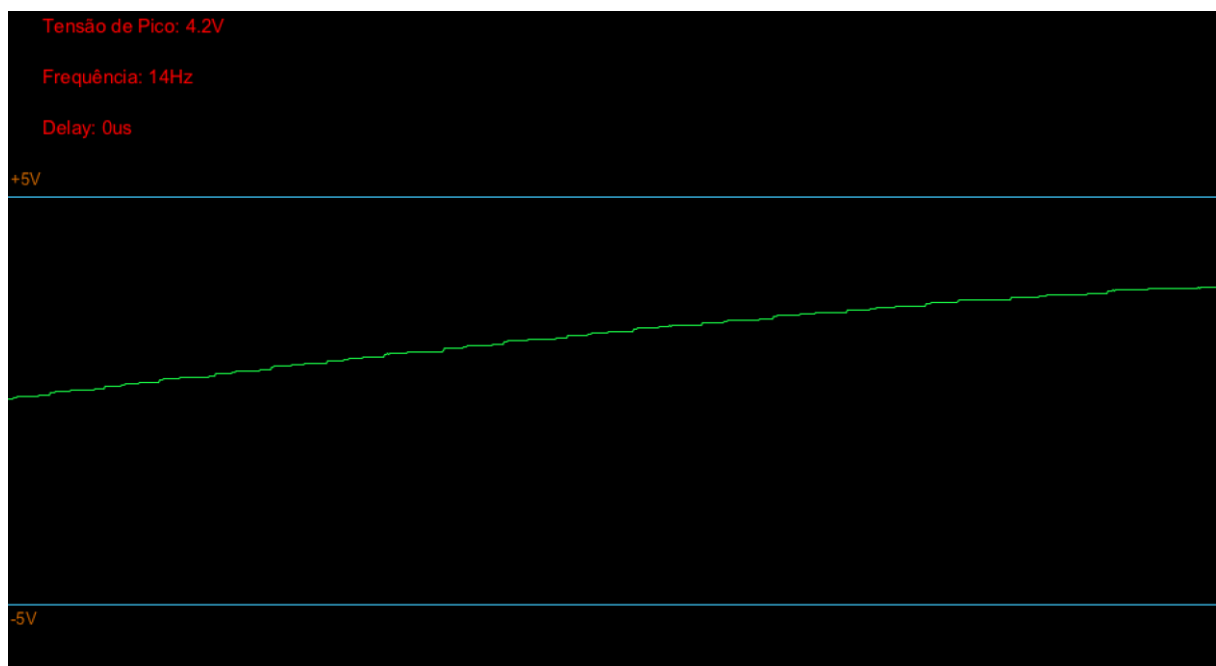
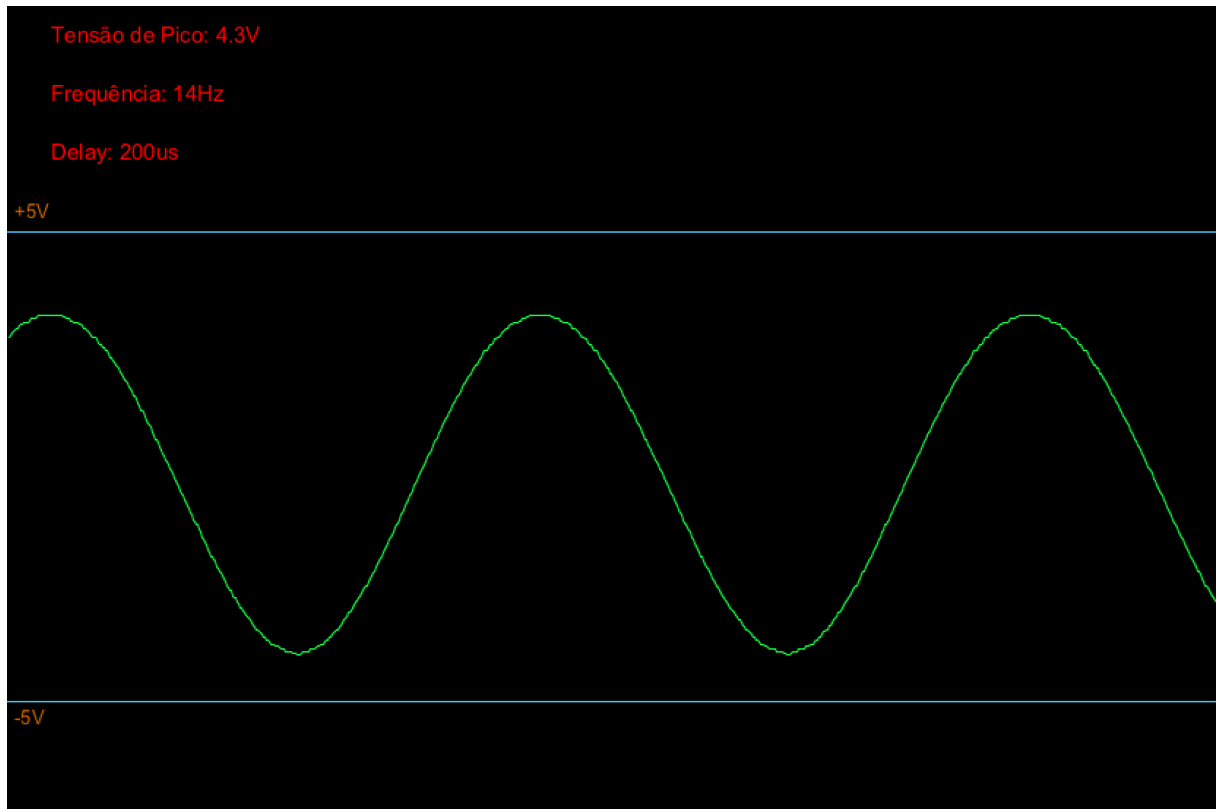


Figura 39 - Onda sem fator de aquisição

Fonte: Autor

Abaixo, o mesmo sinal, porém, com o fator de aquisição no máximo, ou seja, 200 US.



**Figura 40 - Onda com fator de aquisição implementado**

**Fonte: Autor**

Sendo assim, podemos observar o ganho que o fator de aquisição impõe ao projeto, permitindo ampliar a faixa de leitura.

## **4 CONCLUSÃO**

O projeto final apresentado demonstra bastante eficiência dentro da sua margem de leitura. Podendo interpretar com clareza sinais de -10V a +10V e frequências de 0 a 12 kHz. Para as aplicações mais comuns, esse osciloscópio se demonstra mais do que o suficiente na entrega de uma boa leitura e definição da imagem. O erro percentual para a medição da tensão, não ultrapassou 5% nos piores dos casos. Já para a medição da frequência temos um erro menor ainda, chegando a no máximo 3% durante os testes realizados.

Se comparado a um osciloscópio profissional, este projeto consegue replicar, com confiabilidade, as suas funções mais importantes. Pensando em um possível desenvolvimento de um produto, esse projeto demonstra um custo de produção muito menor que de um osciloscópio profissional. Por se tratar de uma plataforma 100% Open-Source, o cliente também poderia acrescentar alguma funcionalidade ou modificar o código, de acordo com a sua necessidade, sem custos adicionais e com muita facilidade, bastando apenas ter o conhecimento da programação.

Contudo, como toda plataforma Open-Source, a facilidade de manipulação do código impõe alguns problemas como: bugs e até mesmo inconsistências do sistema, que muitas vezes são criados por usuários amadores.

Em suma, o projeto se destaca pela sua facilidade de manipulação e custo de desenvolvimento.

## **5 MELHORIAS**

Algumas das melhorias seria introduzir, em um único circuito, uma forma de ler sinais DC e AC. Devido ao capacitor de junção, entre os dois divisores de tensão, esse circuito aceita apenas sinais AC. A única forma de ler sinais DC, seria conectar a entrada diretamente na porta do Arduino, porém, este sinal deve ter entre 0 e 5V. Sendo assim, o circuito desenvolvido aqui, serve apenas para preparar o sinal AC para ser lido.

Outra melhoria no hardware, seria introduzir um diodo de proteção na saída para o microcontrolador. Este diodo deve ser limitador de 5V, para proteger o circuito interno do Arduino de eventuais sobrecargas na fonte.

Ainda abordando o hardware, poderia ser trocado o microcontrolador para um mais potente, ou um que aceite tecnologia PLL (Phase-Locked-Loop). Essa tecnologia é capaz de emular uma frequência, à partir da gerada pelo oscilador. Em outras palavras, este artifício está para frequência, assim como o amplificador está para tensão. Se conseguíssemos aumentar o clock de processamento, seria possível executar o código mais rapidamente e assim, adquirir amostras do conversor A/D com mais frequência, e por fim, aumentar a faixa de leitura do projeto.

Agora falando sobre o software, poderia ser implementado, para correção do formato da onda, um filtro digital. Alguns filtros foram pesquisados e um dos que mais parece surtir efeito é o filtro baseado na média móvel, ou Gaussiana. A princípio, esse filtro poderia ser implementado no código do Processing, pois, este tem o poder de processamento mais rápido e também, para evitar sobrecarregar o microcontrolador com outras tarefas.

## 6 REFERÊNCIAS

[1] <https://processing.org/>

[2] <https://www.arduino.cc/>

[3] FreqCount Library Disponível em:

[https://www.pjrc.com/teensy/td\\_libs\\_FreqCount.html](https://www.pjrc.com/teensy/td_libs_FreqCount.html) Acessado em: 16/05/2016

[4] <http://www.instructables.com/>

[5] EVANS, M., NOBLE, J. , HOCHENBAUM, J. , ARDUINO EM AÇÃO, EDIÇÃO 1, 2013.

[6] ABC DO OSCILOSCÓPIO 2a Edição Disponível em:

<http://www.ceset.unicamp.br/~leobravo/TT%20305/O%20Osciloscopio.pdf>

Acessado em: 10/08/2016.

[7] Datasheet ATMEGA328 Disponível em:

[http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P\\_datasheet\\_Complete.pdf](http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf) Acessado em: 10/08/2016

## **ANEXOS**

## Código do Arduino

// Osciloscópio

#include <FreqCount.h> // biblioteca para determinação da frequência

#define ANALOG\_IN 1 // porta de entrada analógica A1

#define DIGITAL\_OUT 13 // saída para o gerador de onda quadrada D13

#ifndef cbi

#define cbi(sfr, bit) (\_SFR\_BYTE(sfr) &= ~\_BV(bit))

#endif

#ifndef sbi

#define sbi(sfr, bit) (\_SFR\_BYTE(sfr) |= \_BV(bit))

#endif

boolean modoOperacao = false; // false = modo continuo ; true = modo de amostragem

bool GERADOR = true; // habilita/desabilita gerador de onda

unsigned long frequencia; // valor da frequência

int potdelay = 0;

byte ADCBuffer[1280]; // declarando o vetor

int ADSizeBuffer = 1279; // tamanho do vetor (ADCBuffer - 1)

byte value = 0; // variável auxiliar

char CharSerialRX; // variável auxiliar da comunicação Serial

// constante para configuração do prescaler

const unsigned char PS\_16 = (1 << ADPS2);

const unsigned char PS\_32 = (1 << ADPS2) | (1 << ADPS0);

const unsigned char PS\_64 = (1 << ADPS2) | (1 << ADPS1);

const unsigned char PS\_128 = (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);

```

void setup() {

    Serial.begin(250000);
    FreqCount.begin(1000);

    Serial.println(".: Inicio do Programa .:");
    delay(500);

    // O código abaixo foi adquirido do site http://www.embarcados.com.br
    ("gerador de sinais" via estouro do TIMER1)-----

    // Gerador de sinais via interrupção
    if (GERADOR) {
        pinMode(DIGITAL_OUT, OUTPUT);

        // Inicializa o Timer1
        //noInterrupts();          // desabilita todas as interrupções
        TCCR1A = 0;
        TCCR1B = 0;
        TCNT1 = 0;

        /* 1. "16000000 / 256 = 62500 (Frenquencia do Oscilador / prescaler)"
           2. (62500 / 2Hz = 31250) "Frequencia desejada" 31250 = OCR1A
        */
        OCR1A = 31250;          // definindo tempo de estouro

        TCCR1B |= (1 << WGM12); // modo CTC
        TCCR1B |= (1 << CS12);  // 256 prescaler
        TIMSK1 |= (1 << OCIE1A); // habilita timer para a interrupção
        interrupts();          // habilita todas as interrupções configuradas
    }
}

```



```

//TIMSK0 &= ~_BV(TOIE0); // desliga o TIMER0
//-----
-----

}

// configurando o preescaler do ADC:
ADCSRA &= ~PS_128; //limpa configuração da biblioteca do arduino
//ADCSRA |= PS_128; // 64 prescaler
//ADCSRA |= PS_64; // 64 prescaler
//ADCSRA |= PS_32; // 32 prescaler
//ADCSRA |= PS_16; // 16 prescaler
cbi(ADCSRA, ADPS2) ; sbi(ADCSRA, ADPS1) ; sbi(ADCSRA, ADPS0) ;

}

ISR(TIMER1_COMPA_vect) {
    digitalWrite(DIGITAL_OUT, digitalRead(DIGITAL_OUT) ^ 1); // inverte o
estado da porta digital 13,
// a cada intervalo de tempo estipulado
por OCR1A.
}

void serialEvent() {
    if (Serial.available()) {
        CharSerialRX = Serial.read();

        if (CharSerialRX == 'm')
            modoOperacao = !modoOperacao;
    }
}

```

```

void loop() {

    if (!modoOperacao) {
        //double inicio = micros();
        int val = analogRead(ANALOG_IN);    // aquisição da amostra do conversor
A/D
        //double finale = micros();
        //Serial.println ( finale - inicio );
        //double inicio = micros();
        Serial.write( val/4 );    // envio da informação através da porta Serial
        //double finale = micros();
        //Serial.println ( finale - inicio );
    }

    if (modoOperacao) {
        potdelay = analogRead(A5);
        potdelay = map(potdelay, 0, 1023, 0, 200);
        //Serial.println(potdelay);
        if ( potdelay <= 3 ){    // enquanto a tensão no potenciometro for
menor que 0,37V, a variável "potdelay" adquire valor NULO.
            potdelay = 0;
        }

        for ( int ADCCounter = 0; ADCCounter <= ADSizeBuffer; ADCCounter++ ) { // laço
de repetição para limitar o tamanho máximo do vetor
            int val = analogRead(ANALOG_IN);    // leitura do conversor A/D
            ADCBuffer[ADCCounter] = val/4;    // considera apenas os 8 bits mais
significativos
            delayMicroseconds(potdelay);
        }
    }
}

```

```

//Código do medidor de frequência adaptado do site
"https://www.pjrc.com/teensy/td_libs_FreqCount.html"-----
while (FreqCount.available()) {
    frequencia = FreqCount.read();
    if ( frequencia == NULL ){
        // Serial.println("Não foi possível medir a frequencia");
    }
    // else Serial.println(frequencia);
}

//-----
-----

for ( int ADCCounter1 = 0; ADCCounter1 <= ADSizeBuffer; ADCCounter1++ ) {
    Serial.print( ADCBuffer[ADCCounter1] );    // envio do dado através da porta
Serial
    Serial.print(',');                        // caracter delimitador de amostras
}
    Serial.print(frequencia);
    Serial.print(',');
    Serial.print(potdelay);
    Serial.print(',');
    Serial.println();                        // "fim da linha" - indica o final da transmissão para o
receptor*/
}
}

```

