UFABC

Estudo sobre sincronismo de coleta de dados entre placas Arduíno

São Bernardo do Campo – Brasil

2021

UFABC

Estudo sobre sincronismo de coleta de dados entre placas Arduíno

Relatório técnico final apresentado para o Edital N^o 01/2020.

Universidade Federal do ABC – UFABC

Engenharia Biomédica

Centro de Engenharia, Modelagem e Ciências Sociais Aplicadas

São Bernardo do Campo – Brasil

2021

Lista de ilustrações

Sumário

Resumo

Buscar desenvolver opções que permitam versatilidade e alternativas menos invasivas é algo vital na medicina. É nesta situação que a Tomografia por Impedância Elétrica (TIE) (MARTINS et al., 2019) se encontra, uma técnica para obter imagens a partir da medição da impeditividade dos tecidos de determinado corpo. Neste projeto, buscou-se estudar como o sincronismo afeta a coleta de dados entre diferentes microcontroladores, especificamente em Arduínos (ARDUINO, 2020), utilizados por causa do baixo custo, versatilidade na montagem de circuitos e a facilidade na programação. Para tal, foi utilizado o protocolo de comunicação serial I2C, que utiliza-se de um microcontrolador principal, ou master, que se comunica com outros dispositivos periféricos, ou slaves, que serão outros microcontroladores para este projeto de pesquisa. Através da comunicação I2C, foi possível perceber que existe um atraso constante no recebimento dos dados por cada placa e, consequentemente, indica que poderiam ser feitas calibrações via software que permitam coordenar melhor a coleta.

1 Introdução

1.1 Tomografia por impedância elétrica

Para garantir o melhor tratamento possível na medicina moderna são necessários diversos métodos de diagnóstico para se obter imagens vitais na análise e no tratamento de pacientes hospitalares. Dentre os métodos mais utilizados para a obtenção de imagens destacam-se a utilização dos raios-x, ressonância magnética, tomografia nuclear e ultrassonografia. Apesar de todas as vantagens dessas formas de análise, existem pacientes em condições específicas que não podem se locomover à estes tipos de equipamentos, tais como pacientes que necessitem de respiração assistida ou com extrema dificuldade de movimentação, sem contar o custo elevado destes equipamentos, dificultando a utilização em regiões carentes ou sistemas públicos de saúde.

Uma opção a estas formas de avaliação é a TIE (Figura 1), uma técnica capaz de determinar a distribuição de impeditividades de determinado corpo a partir de impulsos elétricas.

Figura 1 – Tomografia por Impedância Elétrica em paciente. Extraído de (VENTILAÇÃO. . . , 2016).

O seu funcionamento se baseia na distribuição de eletrodos ao redor da área a ser examinada, que recebem uma determinada corrente elétrica e medem a impeditividade, permitindo que um programa crie uma imagem da área avaliada, como os pulmões por exemplo.

Apesar de ainda não ser amplamente utilizada, a TIE apresenta um grande potencial devido ao seu tamanho, mobilidade e menor custo, assim como a capacidade de fornecer imagens em alta resolução temporal e em tempo real. Dentre as diversas aplicações da TIE, destacam-se o monitoramento da função gastrointestinal, monitoramento da função pulmonar, monitoramento do sistema circulatório e detecção de câncer de mama (MARTINS et al., 2019). Por esses motivos, a TIE deve se tornar progressivamente mais utilizada globalmente.

1.2 Arduíno

O Arduíno, projeto open-source criado na Itália em 2005, é um microcontrolador de baixo custo que possibilita ao desenvolvedor utilizá-lo em diversas aplicações desde as mais simples que envolvam somente um Arduíno e poucas linhas de código até as mais complexas, que demandem mais do que somente um Arduíno. O hardware apresenta um microcontrolador Atmel AVR capaz de armazenar a programação feita através de seu cabo adaptador para USB (KUSHNER, 2011).

Figura 2 – Arduíno Uno. Extraído de (ARDUINO. . . ,)

Existem diversos modelos diferentes de placas Arduíno, tais como o Arduíno Uno (figura 2), Arduíno Leonardo, Arduíno Nano, Arduíno Micro, dentre outros. Todos os Arduínos, apesar de apresentarem diferentes formas, possuem componentes necessários para serem versáteis e providenciarem um sistema capaz de desenvolver uma grande variedade de projetos.

O Arduíno pode ser alimentado por uma fonte externa independente ou pode ser alimentado direto pelo adaptador USB conectado a um computador, de onde é possível

programá-lo utilizando uma linguagem própria do Arduíno que é muito similar à C e C++, pela plataforma Arduíno IDE (ARDUINO, 2015).

Figura 3 – Tela do Arduíno IDE apresentando um exemplo de programa que pisca um LED da placa. Fonte: Elaborado pelo autor.

A figura 3 acima demonstra um exemplo básico de programa feito na IDE Arduíno, cujo objetivo é acender um LED da própria placa durante um segundo e desligá-lo, dentro de um *loop* infinito. É importante destacar duas partes muito importantes de todos os códigos arduino que são as funções de setup e loop, a primeira tem função de definir quais as funções dos pinos de entrada e saída e a segunda efetivamente executa executa o código indefinidamente.

É importante destacar que esta IDE já apresenta bibliotecas capazes de lidar tanto com o barramento I2C, biblioteca wire, quanto com o barramento SPI, biblioteca SPI.

Além disso, é possível utilizar técnicas como a PWM (do inglês *Pulse Width Modulation*) para a geração de sinais a partir de um Arduíno, em que o microprocessador

gera um sinal de onda quadrada a partir da mudança de tensão total (5V) para a 0V, gerando um padrão frequente variável (PWM...,).

1.2.1 Comunicação I2C

A comunicação I2C é um protocolo de comunicação serial síncrono half-duplex (TANNENBAUM, 2002), ou seja, somente um dispositivo pode se comunicar por vez, enviando a informação para outro dispositivo (NXP, 2014). Este protocolo trabalha com um barramento, o que permite que diversos dispositivos sejm conectados ao mesmo tempo. Cada dispositivo conectado ao barramento recebe um endereço único, que é usado para ser reconhecido e interagir com os outros componentes. O barramento é composto pelos fios SDA (serial data) e SCL (serial clock), que são responsáveis pelo transporte da informação aos dispositivos conectados ao barramento (NXP, 2014), como visto na figura 4. Como este protocolo permite que mais de um aparelho capaz de controlar o barramento esteja conectado, é necessário indicar um único dispositivo para esse controle, chamado de *master*. Esse dispositivo irá gerar o sinal de sincronismo (*clock*) para a transferência de dados, e todos os demais dispositivos trabalham como escravos (*slave*).

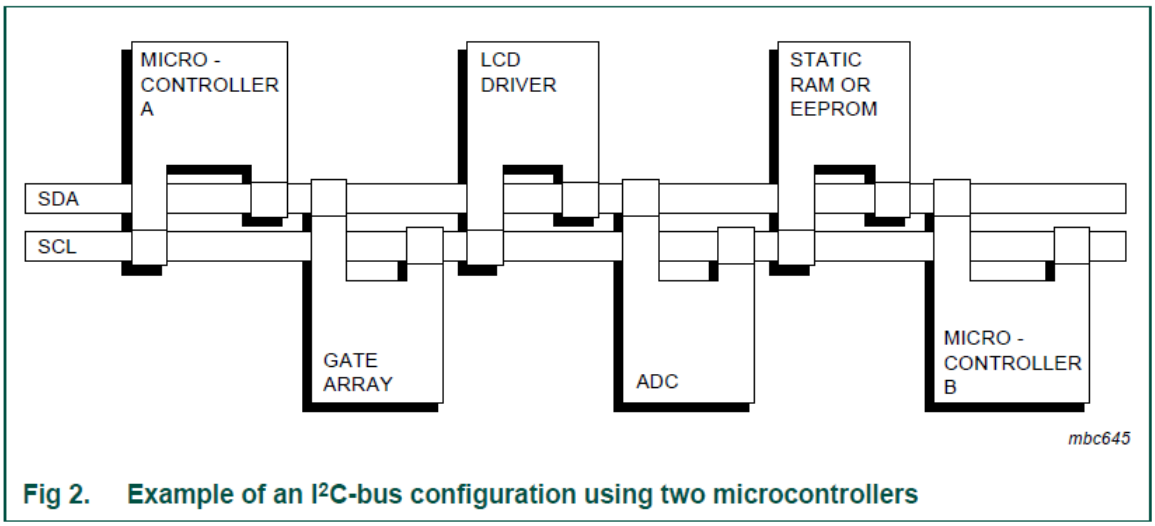

Figura 4 – Exemplo de barramento I2C. Extraído de (NXP, 2014).

1.2.2 Processing

Processing é uma linguagem de programação voltada à criação de gráficos interativos criada em 2001, com uma grande gama de bibliotecas próprias e, assim como a linguagem de programação do arduino, apresenta um ambiente de desenvolvimento integrado (em inglês *integrated development environment*, IDE).

A utilização do processing se mostra necessária dada a fácil interação que e possível ser feita com a IDE do próprio arduino, sendo possível receber e enviar informações. Além

de ser possível analisar os dados, pode-se também salvá-los em tabelas em arquivos .csv.

O ambiente de desenvolvimento do processing é simples e permite uma fácil análise de código, apresentando uma área de mensagem (*Message Area*) e um console, como visto na figura 5, que auxiliam quando necessário identificar algum erro de codificação. O ambiente também apresenta funcionalidades básicas, como um editor de texto, uma barra de ferramentas e diversas abas para navegar entre diferentes códigos.

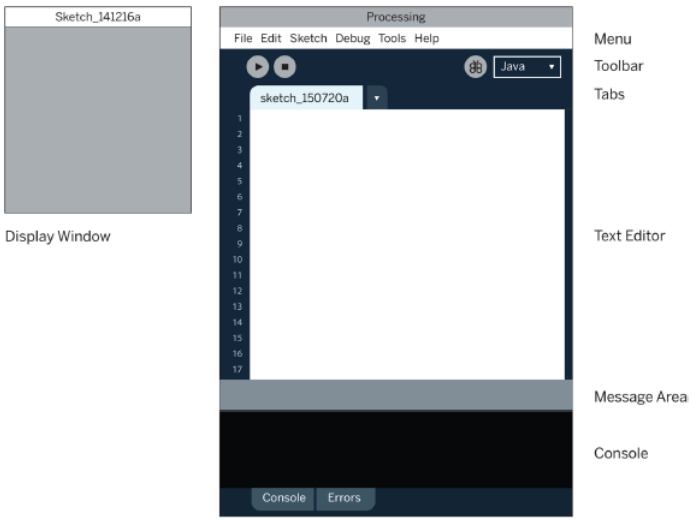

Figura 5 – Ambiente de desenvolvimento Processing. Extraído de (REAS CASEY; FRY, 2015).

1.2.3 Aliasing

Aliasing é um problema causado por uma distorção na amostragem em que um sinal analógico contínuo F qualquer é convertido para um sinal digital a partir de uma amostragem discreta de frequência menor que duas vezes a frequência de F, violando o critério de Nyquist (KESTER,), transformando um sinal de alta frequência em um de baixa frequência, exemplificado pela figura 6. Isso permite que ruídos provenientes de fontes externas, como uma lâmpada ou outro equipamento eletrônico, possam interferir na frequência investigada, acarretando na junção de ambos os sinais na amostragem e tornando-os indistinguíveis um do outro.

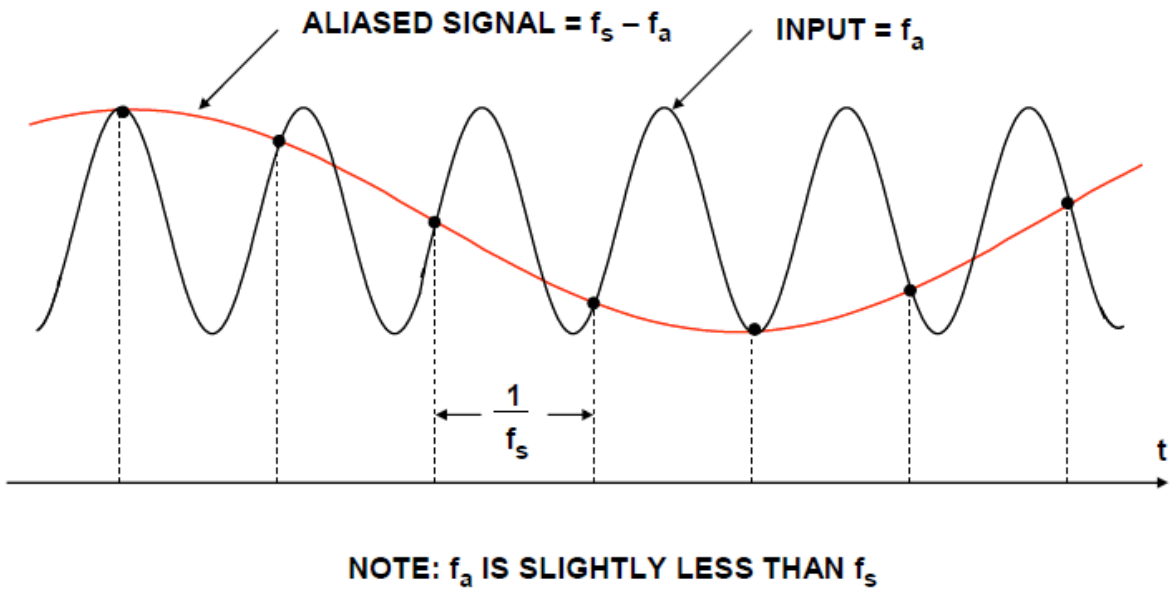

Figura 6 – Sinal analógico submetido a uma amostragem de frequência menor que $2f_s$. Extraído de (KESTER,).

De forma a evitar tais problemas, é necessário utilizar uma frequência de amostragem maior que duas vezes a estudada e remover qualquer alta frequência antes de digitalizar o sinal, utilizando filtros anti-aliasing analógicos.

2 Objetivos

2.1 Objetivos gerais

Este projeto teve como escopo avaliar o sincronismo na coleta de dados entre placas arduino utilizando-se o protocolo de comunicação I2C, de modo a entender se esta é uma forma viável de sincronia na coleta de dados para a montagem de um equipamento de TIE.

2.2 Objetivos específicos

Para atender aos objetivos gerais, os seguintes objetivos específicos deverão ser atendidos:

- Estudo e implementação de barramentos de comunicação I2C;
- Estudo e implementação de programas para geração e leitura de sinais digitais;
- Desenvolvimento de um programa para coleta de dados analógicos pelo Arduíno;
- Verificação do atraso na fase entre sinais analógicos coletados em diferentes placas Arduíno, quando o início da coleta é controlado via comunicação I2C.

3 Metodologia

Vou descrever o que foi feito, e ver se precisa mudar algo nos objetivos

De modo a atender os objetivos do projeto, é prevista a execução dos testes descritos a seguir. A programação das placas Arduíno será feita através da IDE do Arduíno, e a montagem dos circuitos será feita em *protoboards*. A IDE do Arduíno possui códigos de exemplos com implementação dos protocolos de comunicação SPI e I2C, assim como geração e leitura de sinais digitais, uso de interrupções e leitura de dados analógicos, nos quais serão baseados os programas a serem desenvolvidos neste projeto.

3.1 Filtro passa-baixa

Foi utilizado um filtro passa-baixa para controlar o sinal digital e tentar torná-lo mais similar a uma onda senoidal analógica, vide figura 7, devido à sua capacidade de atenuação de sinais de alta frequência. O capacitor é de 40 nano Farad e o resisto é de 100 kilo ohms, resultando numa frequência de corte de 40 Hz.

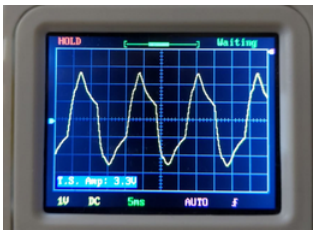

Figura 7 – Registro da onda gerado pelo arduino master. Elaborado pelo autor.

3.2 Comunicação Serial

Este teste visa a compreensão do funcionamento e implementação do protocolo de comunicação I2C. O teste feito utiliza-se do código que gera o sinal para que cada placa colete um sinal conhecido e que o retransmita para a placa mestre quando exigido e, para tal, foi utilizada a biblioteca da própria IDE arduino voltada para o barramento I2C, a wire.

Assim, foi implementado um barramento de comunicação I2C entre três placas Arduíno, visando a posterior avaliação na diferença de coleta entre as placas periféricas.

3.3 Geração de sinais digitais pelo código master

Nesta etapa foi desenvolvido um código para o arduino master cuja função é gerar um sinal digital, enviar este sinal aos arduínos e coletar os sinais através do barramento utilizado, mo caso o protocolo I2C, mediante um envio de sinal do master para indicar o início da leitura. O código depende do cálculo feito pelos slaves que reconhecem a quantidades de vezes que o valor foi de 0 a 5 volts, o equivalente a uma "subida"no sinal, e então imprime no monitor serial a diferença na quantidade de valores registrados por cada um, além de imprimir também o período calculado de acordo com os slaves.

Resumidamente, a ideia nestes testes é registrar em ambos arduinos os pulsos enviados, de forma que seja possível analisar o atraso entre os sinais coletados pelas diferentes placas em cada um dos diferentes métodos a serem avaliados na etapa de verificação de fase.

3.4 Coleta de sinais pelo código slave

Conjuntamente com o código master, foi feito um código que coleta o sinal enviado medindo a quantidade de variações de 0 a 5 volts do sinal. Além disso, foram criadas duas funções, uma para receber a quantidade de picos a serem lidas e a outra para atender ao pedido do master para envio dos dados coletados.

3.5 Verificação do atraso

Por último, a verificação do atraso entre os sinais captados tem como finalidade cumprir o objetivo principal definido neste trabalho..

Para isso, foi enviado um sinal digital gerado por uma das placas arduinos, onde o início da coleta destas é controlado por um comando serial I2C. Em seguida, foi calculada a diferença na quantidade de dados coletados, medindo a diferença na quantidade de subidas lidas e calculando o período e a frequência medidos por cada slave.

4 Resultados

4.1 Comunicação Serial

Para a comunicação serial foi feito um teste utilizando a comunicação I2C. O objetivo do teste, como previamente descrito, foi enviar sinais digitais que seriam lidos pelos microcontroladores periféricos, que se comunicam com a placa master para calcularem a diferença na coleta de sinal e o período.

4.1.1 Protocolo I2C

4.1.1.1 Código Master

Em termos gerais código I2C master envia os sinais digitais para serem lidos por qualquer microcontrolador que esteja "atento"aos sinais, ou seja, que estejam conectados ao sinal de saída do filtro passa-baixa, e, após autorizar o início da leitura dos sinais, requisita que retornem os sinais medidos.

O bloco de setup basicamente inicializa o monitor serial e a comunicação I2C, assim como inicializa o pino 6 como saída de sinal e o define como LOW na saída de sinal, vide a figura 8.

```
void setup() {
  Serial.begin(115200); //Inicializa o SerialMonitor.
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB
  }
  Serial.println("Inicio da leitura"); //Marca o início dos dados.

  pinMode(PWMpin, OUTPUT); //Define o PWMpin como saída, emissor da PWM.
  digitalWrite(PWMpin, LOW);

  Wire.begin();
}
```

Figura 8 – Bloco setup do código master I2C. Elaborado pelo autor.

O código, então, envia aos slaves um sinal que autoriza o início da leitura, como visto na figura 9 a seguir.

```
void ComecaLeitura(){ // Não pode ter serial aqui pois atrasa o sincronismo
  //Primeiro.
  Wire.beginTransmission(SLAVE1);
  Wire.write(partida);
  Wire.endTransmission();
  //Segundo.
  Wire.beginTransmission(SLAVE2);
  Wire.write(partida);
  Wire.endTransmission();
}
```

Figura 9 – Bloco permitindo o início da leitura. Elaborado pelo autor.

A seguir, o código envia os sinais alternando o valor do pino de saída de HIGH para LOW, com um delay de um mili segundo e, após 10 picos enviados, exige o retorno dos sinais medidos pelos slaves, como visto na figura 10.

```
void ReceiveData(int placa, int i){
  Wire.requestFrom(placa, i);
  int a = 0, b = 0;
  if (placa == SLAVE1){ //armazena os valores lidos pelo slave.
    while (Wire.available()) {
      val1[a] = Wire.read();
      a++;
    }
  }
  else{
    while (Wire.available()){
      val2[b] = Wire.read();
      b++;
    }
  }
```

Figura 10 – Bloco requisitando as quantidades de sinais lidos pelos slaves. Elaborado pelo autor.

Por fim, o código master calcula a diferença de picos e o atraso medidos pelo código slave e o período e a frequência de medição para cada slave, vide figuras 11 e 12.

```
void ImprimeInfo(){
  int diff; //Usada para imprimir a diferença entre os arduinos.

  //Calcula a diferença dos picos e manda pela serial
  for (int num = 0; num < NPICOS; num++){
    diff = abs(val1[num] - val2[num]);
    Serial.print(num);
    Serial.print(": ");
    Serial.print(val1[num]);
    Serial.print('\t'); //Espaçamento equivalente a um Tab
    Serial.print(val2[num]);
    Serial.print('\t');
    Serial.print(diff);
    Serial.print("\tAtraso:\t");
    Serial.print((1000.0/9600.0)*diff);
    Serial.println("ms");
  }
```

Figura 11 – Bloco que imprime a quantidade de sinais lidos e seu atraso. Elaborado pelo autor.

```
for (int num = 1; num < NPICOS; num++){
  int periodo1 = (float)val1[num]-(float)val1[num-1];
  int periodo2 = (float)val2[num]-(float)val2[num-1];
  Serial.print(num);
  Serial.print(": Periodo\t");
  Serial.print(periodo1);
  Serial.print('\t'); // Espaçamento equivalente a um Tab
  Serial.print(periodo2);
  Serial.print("\tFreq Sinal:\t");
  Serial.print(9600.0/(float)periodo1);
  Serial.print('\t'); // Espaçamento equivalente a um Tab
  Serial.println(9600.0/(float)periodo2);
}
Serial.println(" ");
```

Figura 12 – Bloco que imprime o período dos sinais lidos e sua frequência. Elaborado pelo autor.

4.1.1.2 Código Slave

O bloco de setup inicializa a entrada do slave no barramento I2C, registra o evento de recebimento de sinal do master, registra o pedido de envio de leitura que é feito pelo arduino master. Por fim, o código também define um pino específico para ler se está em HIGH ou LOW, definindo assim o endereço para ingresso no barramento I2C, usado para diferenciar os microcontroladores. Como pode ser visualizado na figura 13.

```
void setup(){
  pinMode(END_PIN, INPUT_PULLUP); // Pino que define endereço do Slave
  delay(50);
  if (digitalRead(END_PIN)){      // Se END_PIN não estiver aterrado...
    ENDERECO = SLAVE1;
  }
  else {                          // Se END_PIN estiver aterrado...
    ENDERECO = SLAVE2;
  }
  Wire.begin(ENDERECO);           // Inicializa a comunicação I2C, ingressando no barramento.
  Wire.onReceive(receiveEvent);   // Registra evento.
  Wire.onRequest(receiveRequest); // Registra request.

  pinMode(SIGN_PIN, INPUT);       // Configura o pino analógico SIGN_PIN como entrada.

  mediu = 0;
}
```

Figura 13 – Bloco de setup dos microcontroladores periféricos. Elaborado pelo autor.

O código slave, ao receber o sinal no barramento de que está autorizado a iniciar a leitura, através de um evento enviado pelo mestre como visto na figura 14, começa a coleta de sinais a partir de um dos pinos analógicos definidos no início do código como entrada. Após a leitura do número máximo de sinais é feita uma análise no bloco de repetição para verificar quantas subidas foram detectadas e, caso não tenham sido encontradas as quantidades exigidas, define o vetor de armazenamento como um vetor nulo, vide figura 15.

```
void receiveEvent(){
  NPICOS = Wire.read();
  if (NPICOS > NPICOS_MAX)
    NPICOS = NPICOS_MAX;
  partida = 200;
}
```

Figura 14 – Bloco repetição dos microcontroladores periféricos. Elaborado pelo autor.

```
void loop() {
  if (partida == 200){                  // Verifica sinal de início de leitura.
    for (int idx = 0; idx < NPONTOS; idx++){
      sinal[idx] = analogRead(SIGN_PIN); // Armazena a tensão (sampling aprox. 9600Hz)
    }

    int cont = 0;
    for (int idx = 0; idx < NPONTOS-1; idx++){
      if( (sinal[idx] < 512) && (sinal[idx+1] >= 512)){ // detecta subida
        kont[cont] = (char)idx;
        cont++;
      }
      if (cont >=NPICOS) break;          // se encontrou NPICOS subidas, termina procura
    }
    if (cont < NPICOS){                  // se não encontrou NPICOS subidas, completa com 255
      for (int idx = cont; idx < NPICOS; idx++)
        kont[idx] = (char)255;
    }
    partida = 0;
    mediu = 1;
  }
```

Figura 15 – Bloco de repetição dos microcontroladores periféricos. Elaborado pelo autor.

Por fim, o algoritmo aguardo o pedido do master para envio dos sinais medidos, através do barramento I2C, redefinindo o vetor com a medições como um vetor nulo, a fim de se evitar repetições no envio do sinal, vide figura 16.

```
void receiveRequest(){
  if (mediu == 1){
    Wire.write(kont, NPICOS); //Envia o número de medidas para o master.
    mediu = 0;
    for (int idx = 0; idx < NPICOS; idx++) // para garantir que não vai enviar o mesmo valor novamente
      kont[idx] = (char)255;

  }
```

Figura 16 – Bloco de repetição dos microcontroladores periféricos. Elaborado pelo autor.

4.2 Geração e captura de sinais

4.2.1 Sinais digitais e filtro passa-baixa

Os testes e desenvolvimentos para os sinais digitais feitos utilizaram 3 arduinos de modelos diferentes interagindo entre si, o arduino Uno responsável por gerar o sinal (master) e ambos os Nanos responsáveis por capturar os sinais. Os modelos utilizados foram Arduino Uno e Nano.

Os sinais foram inseridos em um filtro passa-baixa para que fosse possível emular um sinal analógico e que fosse possível utilizar a função AnalogRead para a leitura de sinais.

4.2.1.1 Leitura de sinais digitais

Previamente a ser decidido que os arduinos iriam fazer todos os cálculos de atraso, período e frequência, foram desenvolvidos programas através da linguagem Processing, cuja função era coletar os dados analógicos e permitir que fossem avaliados através de planilhas excel.

Portanto, o código processing é capaz de interagir com o monitor serial para coletar os sinais recebidos pelos arduinos e armazenar estes dados em uma tabela csv. Para que conseguisse ler os sinais, foi necessário importar a biblioteca serial presente no processing, o que permitiu criar dois objetos *Serial*, um para o Arduino Uno ("Unoport"), utilizado como master, e outro para nano ("Nanoport"), utilizado como slave, ambos responsáveis por permitir a leitura dos sinais enviados ao monitor serial por cada arduino.

Outras variáveis importantes criadas foram uma variável *Table*, responsável por armazenar os sinais enviados em uma tabela csv, duas variáveis *String* que guardam os valores na tabela, há ainda uma variável inteira responsável por limitar o número de leituras e outra que conta o número de leituras feitas. Por fim, duas variáveis inteiras servem para contar o número de vezes em que o código dicou preso dentro dos dois bloco try/catch e uma terceira inteira representa *LineFeed* em ASCII, usada para parar a leitura do sinal quando este muda de linha no monitor serial do arduino. Podemos observar todas as variáveis citadas acima na figura 17.

```
5  import processing.serial.*; //importa a biblioteca serial.
6  Serial Unoport; // Cria um objeto da classe serial, serve para ler se há sinal na porta do Arduino.
7  Serial Nanoport; // Mesma função do anterior, mas para o Nano.
8  Table table; //Cria uma variável tabela que serve pra armazenar os dados.
9
10 int numLeituras = 51; //Essa variável serve para limitar o número de leituras.
11 int numCount = 1; //Essa serve para contar o número de leituras.
12 int times = 0;
13 int vezes = 0; //Essas variáveis servem para contar o número de vezes que o try/catch rodou.
14
15 int lf = 10; // Nova linha (Linefeed) em ASCII.
16 String UnoString = null; // String onde sera guardada a mensagem lida pelo arduino Uno.
17 String NanoString = null; //Serve para guardar a mensagem do Nano.
18
```

Figura 17 – Variáveis Processing. Elaborado pelo autor.

O bloco setup apresenta duas funções principais para este código que são inicializar as portas seriais em que os arduinos estão conectados para permitir que leiam as informações enviadas pelos arduinos e inicializar a variável de tabela, vide figura 18.

```
19 void setup(){//Prepara o programa.
20  String portUno = Serial.list()[0]; // Porta no Windows é a primeira, use para o Uno.
21  String portNano = Serial.list()[1]; // Porta do Nano.
22  println("Portas seriais disponiveis:");
23  println(Serial.list());
24  println("Portas seriais em uso:");
25  println(portUno, portNano);
26
27  table = new Table();
28
29  // Inicializa as portas.
30  Unoport = new Serial(this, portUno, 9600);
31  Nanoport = new Serial(this, portNano, 9600);
32
```

Figura 18 – Setup Processing. Elaborado pelo autor.

A outra função são os blocos try/catch, um para cada arduino, servem para evitar erros de leitura que possam causar um exceção e parar a execução do programa. Pela figura 19 podemos observar, além do bloco try/catch, a utilização da variável que termina a leitura quando o sinal enviado muda de linha. Por fim, o bloco setup ainda cria as colunas contagem, sinal 1 e sinal 2.

```
34  try { //Try catch do arduino uno
35    Unoport.clear();
36    while (Unoport.readStringUntil(lf) == null){
37      println("Preso no while Unoport");
38      times = times + 1;
39    }
40    UnoString = Unoport.readStringUntil(lf);
41    UnoString = null;
42  }
43  catch (Exception e){
44    println("Deu algum problema no Unoport");
45  }
46  finally{
47    println("Os ciclos do while Unoport foram: ", times);
48  }
49  try { //Try catch do arduino uno
50    Nanoport.clear();
51    while (Nanoport.readStringUntil(lf) == null){
52      println("Preso no while Nanoport");
53      vezes = vezes + 1;
54    }
55    NanoString = Nanoport.readStringUntil(lf);
56    NanoString = null;
57  }
58  catch (Exception e){
59    println("Deu algum problema no Nanoport");
60  }
61  finally {
62    println("Os ciclos do while do Unoport foram: ", times);
63    println("Os ciclos do while do Nanoport foram: ", vezes);
64  }
```

Figura 19 – Try/Catch Processing. Elaborado pelo autor.

Dentro da função draw existe uma condicional que garante que a leitura seja feita somente se tiver sinal disponível em ambos arduinos e que estes sejam não nulos. A seguir, outra condicional verifica se as strings recebidas não estão vazias e, caso não estejam, converte para inteiro e armazena os valores na tabela a ser salva ao término das leituras.

```
void draw(){
  if (Unoport.available() > 0 && Nanoport.available() > 0){ // a leitura da serial deve ficar aqui dentro!
    UnoString = Unoport.readStringUntil(lf);
    NanoString = Nanoport.readStringUntil(lf);

    if (UnoString != null && NanoString != null){
      println(UnoString, NanoString);
      TableRow novaLinha = table.addRow();

      // remove quebra de linhas da string
      UnoString = UnoString.replace('\n', ' '); // troca quebra de linha por espaço
      int valor_Uno = Integer.valueOf(UnoString.trim()); // converte para inteiro. O trim() remove o espaço do final

      NanoString = NanoString.replace('\n', ' ');
      int valor_Nano = Integer.valueOf(NanoString.trim());

      // coloca dados novos na tabela
      novaLinha.setInt("Contagem", numCount);
      novaLinha.setInt("Sinal 1", valor_Uno);
      novaLinha.setInt("Sinal 2", valor_Nano);
      numCount++;
      //println(sinalIn);
      println(numCount);
    }
  }
  if (numCount == numLeituras){
    saveTable( table, "data/arquivo9.csv");
    println("Finalizado");
    exit();
  }
}
```

Figura 20 – Função draw de Processing. Elaborado pelo autor.

4.3 Montagem de circuito

Foram feitas diversas conexões utilizando-se jumpers, sendo possível dividi-las em conexões de comunicação, para o protocolo I2C, conexão de alimentação e conexão de sinal. Os pinos de comunicação são padrão para o arduino Uno e Nano, sendo necessário conectar os pinos A4 e A5 das placas entre si. Todos os arduinos foram aterrados entre si, pino GND, e foram alimentados pela tensão de 5 volts do arduino master. O sinal foi enviado pelo master através do pino D6 e coletado pelos pinos A2 dos slaves. Por fim, o segundo slave teve o pino D4 conectado ao pino GND, de forma a se definir o endereço deste.

4.4 Verificação do atraso

Para a verificação do atraso, decidiu-se por utilizar o microcontrolador master, um arduino uno, para efetuar os cálculos e imprimir no monitor serial diretamente a quantidade de medidas feitas e o atraso registrado entre cada arduino, além de imprimir o período e a frequência, como pode ser visto na coleta de dados da figura 21.

```
Inicio da leitura
0: 37   35      2       Atraso: 0.21ms
1: 55   53      2       Atraso: 0.21ms
2: 73   71      2       Atraso: 0.21ms
3: 91   89      2       Atraso: 0.21ms
1: Periodo      18      18      Freq Sinal:     533.33  533.33
2: Periodo      18      18      Freq Sinal:     533.33  533.33
3: Periodo      18      18      Freq Sinal:     533.33  533.33

0: 37   35      2       Atraso: 0.21ms
1: 55   53      2       Atraso: 0.21ms
2: 73   71      2       Atraso: 0.21ms
3: 91   89      2       Atraso: 0.21ms
1: Periodo      18      18      Freq Sinal:     533.33  533.33
2: Periodo      18      18      Freq Sinal:     533.33  533.33
3: Periodo      18      18      Freq Sinal:     533.33  533.33

0: 37   35      2       Atraso: 0.21ms
1: 55   53      2       Atraso: 0.21ms
2: 73   71      2       Atraso: 0.21ms
3: 91   89      2       Atraso: 0.21ms
1: Periodo      18      18      Freq Sinal:     533.33  533.33
2: Periodo      18      18      Freq Sinal:     533.33  533.33
3: Periodo      18      18      Freq Sinal:     533.33  533.33
```

Figura 21 – Bloco repetição dos microcontroladores periféricos. Elaborado pelo autor.

O que a imagem anterior diz é que, em todos os testes feitos, o segundo arduino sempre lia duas subidas a menos, como podemos visualizar na linha 0 do primeiro bloco, com 37 leituras sendo feitas pelo primeiro slave e 35 leituras pelo segundo, representando 21 mili segundos de atraso. Além disso, Considerando o intervalo entre as subsequentes medidas, é possível perceber que foi constante a medida de 18 subidas no valor do sinal, para ambos os microcontroladores periféricos, representando uma frequência de 533,33 Hz. A montagem pode ser vista na figura 22.

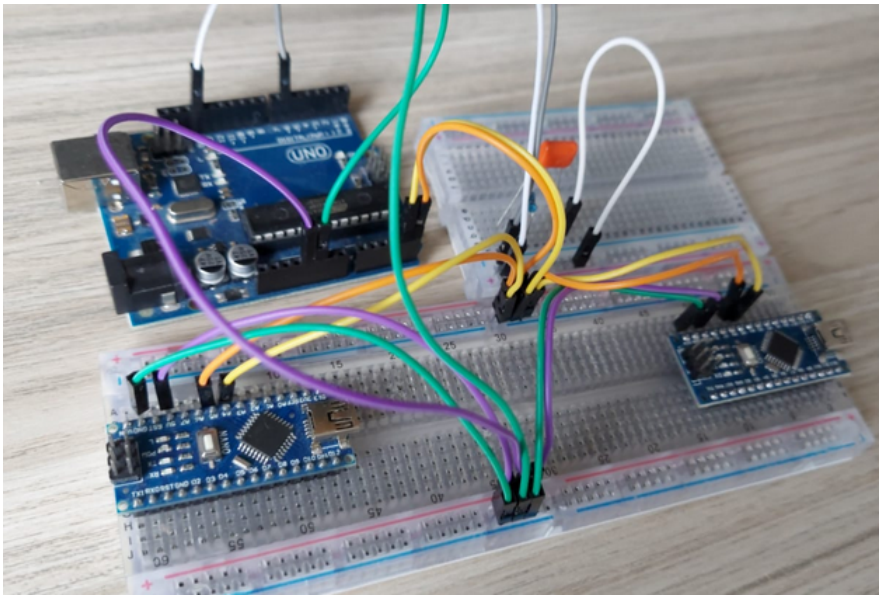

Figura 22 – Circuito montado sem a conexão dos pinos de coleta e envio dos sinais. Elaborado pelo autor.

5 Conclusão

Após análise dos sinais coletados pelos arduinos slaves, foi possível perceber um atraso constante na leitura dos sinais entre os arduinos responsáveis pela coleta. Isto indica que medidas podem ser tomadas para corrigir esse atraso, tais como mudanças no código que tentem minimizar este problema, tal como um atraso forçado no início da leitura do primeiro arduino slave que o faça aguardar 21 mili segundos antes de iniciar a leitura. Neste projeto não foi possível utilizar de sinais analógicos para serem coletados pelas placas, para tal, foi utilizado um sinal digital modulado de corrente contínua que atravessa um filtro passa-baixa.

Referências

ARDUINO, S. A. Arduino. *Arduino LLC*, 2015. Citado na página 7.

ARDUINO, U. Disponível em:< https://www. arduino. cc/en/main/arduinoboarduno>. *Acesso em*, 2020. Citado na página 4.

ARDUINO Uno. Disponível em: <https://store-cdn.arduino.cc/usa/catalog/product/cache/1/image/520x330/604a3538c15e081937dbfbd20aa60aad/a/0/a000066_featured_5.jpg>. Acesso em: 21 may 2020. Citado 2 vezes nas páginas 2 e 6.

KESTER, W. What the nyquist criterion means to your sampled data system design. Citado 3 vezes nas páginas 2, 9 e 10.

KUSHNER, D. The making of arduino. *IEEE spectrum*, v. 26, 2011. Citado na página 6.

MARTINS, T. de C. et al. A review of electrical impedance tomography in lung applications: Theory and algorithms for absolute images. *Annual Reviews in Control*, Elsevier, 2019. Citado 2 vezes nas páginas 4 e 6.

NXP, S. I^2c bus–i^2c bus specification and user manual. *Available in:< http://www.nxp.com/documents/user_manual/UM10*, v. 204, 2014. Citado 2 vezes nas páginas 2 e 8.

PWM, ARDUINO. Disponível em: <<https://www.arduino.cc/en/Tutorial/PWM>>. Acesso em: 27 may 2020. Citado na página 8.

REAS CASEY; FRY, B. *Reas, Casey, and Ben Fry. Getting Started with Processing: A Hands-on introduction to making interactive Graphics.* [S.l.]: Maker Media, Inc., 2015. Citado 2 vezes nas páginas 2 e 9.

TANNENBAUM, A. S. *Computer networks.* [S.l.]: Pearson Education India, 2002. Citado na página 8.

VENTILAÇÃO em posição prona e monitorização da ventilação com impedância elétrica em paciente com SDRA por H1N1. 2016. Disponível em: <<https://www.sopati.com.br/lermais_materias.php?cd_materias=423&friurl=_-Surto-do-Virus-Influenza-A-H1N1-atinge-Sao-Paulo-e-mata-mais-de-250-pessoas-por-insuficiencia-respiratoria-aguda-_>>. Acesso em: 21 may 2020. Citado 2 vezes nas páginas 2 e 5.