UNIVERSIDADE PRESBITERIANA MACKENZIE
ESCOLA DE ENGENHARIA
ENGENHARIA ELÉTRICA/ ELETRÔNICA

GREGORY WANDERLEY KIYOSHI KAGA
VICTOR SHINDY KURODA

ELETROCARDIOGRAMA (ECG): ESTUDO DE CASO COM ARDUINO

São Paulo
2014

GREGORY WANDERLEY KIYOSHI KAGA 3102174-3
VICTOR SHINDY KURODA 3098901-9

ELETROCARDIOGRAMA (ECG): ESTUDO DE CASO COM ARDUINO

Projeto apresentado ao Curso de Engenharia Eletrônica da Escola de Engenharia da Universidade Presbiteriana Mackenzie, como requisito parcial para a obtenção da aprovação na disciplina Microprocessadores II.

ORIENTADOR: PROF.MS. IVAIR REIS NEVES ABREU

São Paulo
2014

SUMÁRIO

Eletrocardiograma (ECG): Estudo de caso com Arduino

1- Objetivo

Implementar um programa na interface de programação open-source do Arduino de um Eletrocardiograma (ECG) para analisar sinais elétricos do funcionamento de um coração. Aplicar conhecimentos adquiridos durante as aulas de Microprocessadores II. Desenvolver autonomia e raciocínio lógico, criar habilidade e conhecimento na plataforma Arduino e seus periféricos para desenvolver projetos eletrônicos embarcados, através de ferramentas utilizadas em engenharia eletrônica. Utilizar os conhecimentos adquiridos durante o curso para desenvolver dispositivos eletrônicos associados as necessidades da área da saúde como estudo de caso.

2- Arduino

Arduino é uma placa de controle de entrada de dados (sensores) e saídas de dados (motores e leds), com cristal oscilador de 16MHz, um regulador de tensão de 5V, botão de reset, plugue de alimentação, pinos conectores e alguns leds para facilitar a verificação do funcionamento do dispositivo. A porta USB fornece alimentação quando estiver conectada ao computador e a tensão de alimentação quando desconectado pode variar de 7V a 12V, graças ao regulador presente na placa.

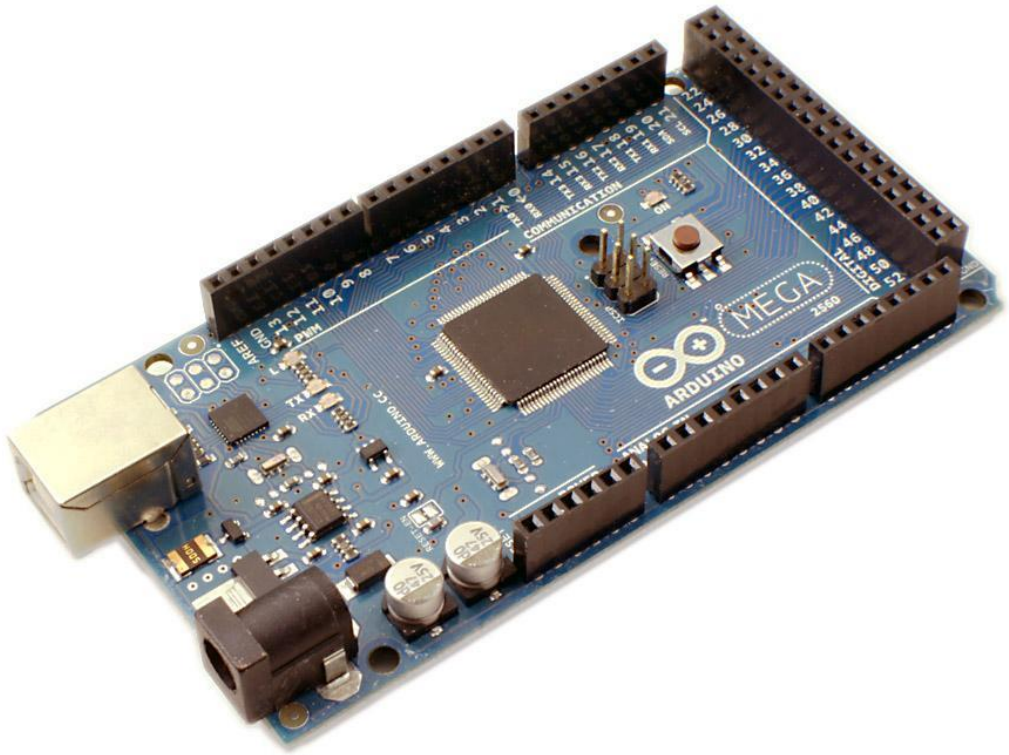
No Arduino, informações ou ordens são transmitidas de um computador para a placa através de *Bluetooth*, *wireless*, USB e infravermelho, por exemplo. Tais informações devem ser traduzidas utilizando a linguagem *Wiring* baseada em C/C++.

(Trecho adaptado de: Introdução ao Arduino. Grupo de Robótica. UFMS 2012. Destacom - Despertando Talentos em Computação)

2.1- Arduino Mega 2560 R3

O Arduino é uma plataforma de informática open-source (código aberto). É basicamente uma placa de circuito impresso com entradas e saídas digitais ou analógicas. Tem por base a linguagem C, tendo um software próprio para desenvolvimento. O sistema Arduino pode ser executado com autonomia própria ou via software, conectado via um computador.

O Arduino Mega 2560 R3 é uma placa que possui um microcontrolador Atmega2560. Contém 54 pinos digitais (entrada/saída): 15 pinos podem ser utilizados como saída PWM (2 a 13 e 44 a 46), 16 pinos lógicos, 4 USARTs (Portas Seriais de Hardware), um cristal oscilador de 16MHz, entrada USB, entrada de alimentação, soquete de comunicação ICSP e um botão reset. A alimentação pode ser feita por um cabo USB, fonte de alimentação AC-DC ou bateria. O Mega 2560 é compatível com a maioria dos shields fabricados para o Arduino 2009 ou Diecimila. Opera em sistemas: Windows, Linux e Mac OS X.

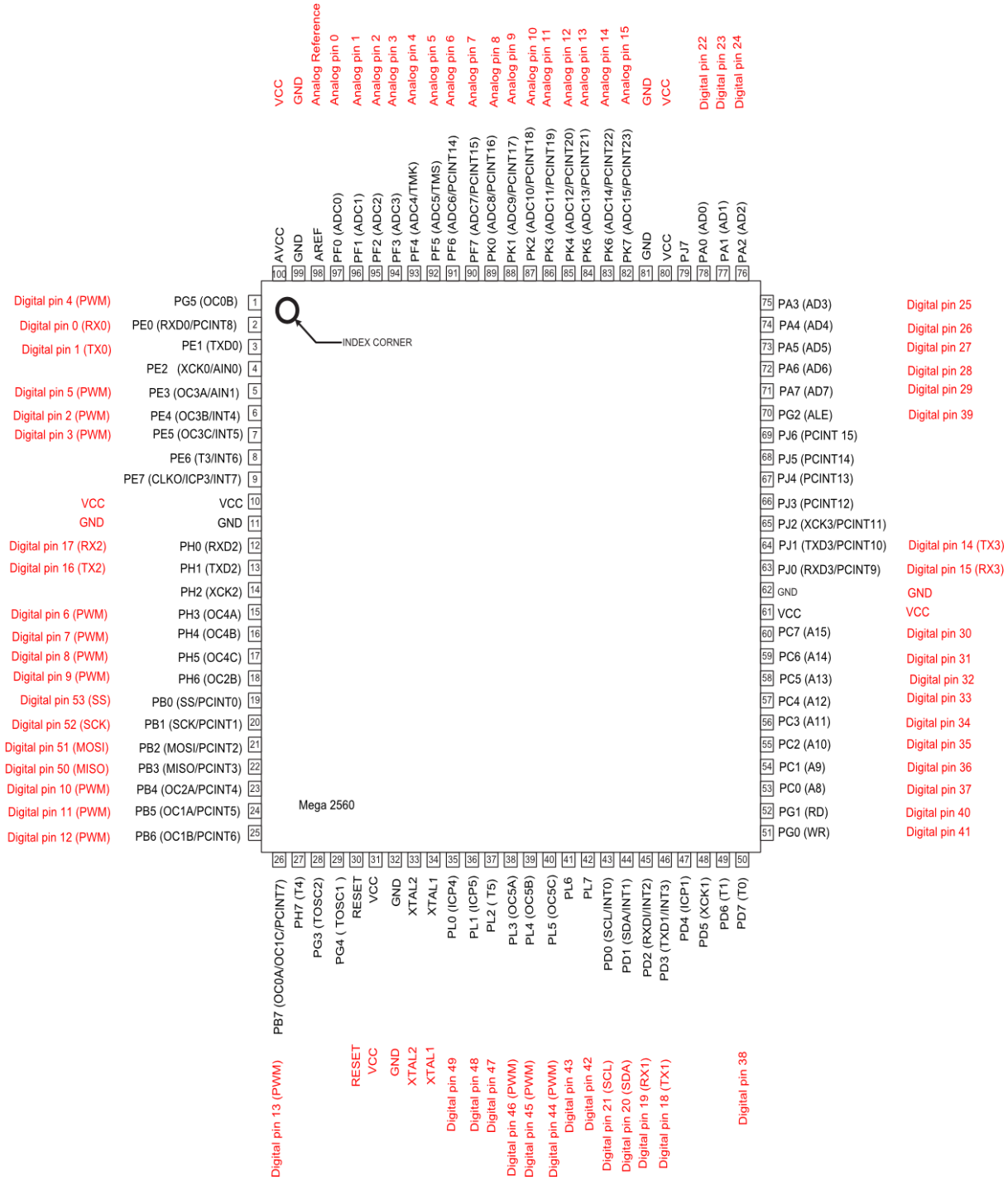


2.1.1- Características Técnicas

Tamanho:	5,3cm x 10,2cm x 1,0cm
Microcontrolador:	ATmega2560
Tensão de operação:	5V
Tensão de entrada (recomendada):	7-12V
Tensão de entrada (limites):	6-20V
Pinos de entrada/saída (I/O) digitais:	54 (dos quais 14 podem ser saídas PWM)
Pinos de entrada analógicas:	16
Corrente DC por pino I/O:	40mA
Corrente DC para pino de 3,3V:	50mA
Memória Flash:	256KB (8KB são usados pelo *bootloader)*
SRAM:	8KB
EEPROM:	4KB
Velocidade de Clock:	16MHz

2.1.2- Pinagem do Processador

Na figura a seguir estão os 100 pinos do processador embarcado no Arduino Atmega 2560 com sua numeração, nome do pino e o nome mapeado no chip.



NÚMERO DO PINO	NOME DO PINO	NOME DO PINO MAPEADO
1	PG5 (OC0B)	Digital pin 4 (PWM)
2	PE0 (RXD0/PCINT8)	Digital pin 0 (RX0)
3	PE1 (TXD0)	Digital pin 1 (TX0)
4	PE2 (XCK0/AIN0)	
5	PE3 (OC3A/AIN1)	Digital pin 5 (PWM)
6	PE4 (OC3B/INT4)	Digital pin 2 (PWM)
7	PE5 (OC3C/INT5)	Digital pin 3 (PWM)
8	PE6 (T3/INT6)	
9	PE7 (CLKO/ICP3/INT7)	
10	VCC	VCC
11	GND	GND
12	PH0 (RXD2)	Digital pin 17 (RX2)
13	PH1 (TXD2)	Digital pin 16 (TX2)
14	PH2 (XCK2)	
15	PH3 (OC4A)	Digital pin 6 (PWM)
16	PH4 (OC4B)	Digital pin 7 (PWM)
17	PH5 (OC4C)	Digital pin 8 (PWM)
18	PH6 (OC2B)	Digital pin 9 (PWM)
19	PB0 (SS/PCINT0)	Digital pin 53 (SS)
20	PB1 (SCK/PCINT1)	Digital pin 52 (SCK)
21	PB2 (MOSI/PCINT2)	Digital pin 51 (MOSI)
22	PB3 (MISO/PCINT3)	Digital pin 50 (MISO)
23	PB4 (OC2A/PCINT4)	Digital pin 10 (PWM)
24	PB5 (OC1A/PCINT5)	Digital pin 11 (PWM)
25	PB6 (OC1B/PCINT6)	Digital pin 12 (PWM)
26	PB7 (OC0A/OC1C/PCINT7)	Digital pin 13 (PWM)
27	PH7 (T4)	
28	PG3 (TOSC2)	
29	PG4 (TOSC1)	
30	RESET	RESET
31	VCC	VCC
32	GND	GND
33	XTAL2	XTAL2
34	XTAL1	XTAL1
35	PL0 (ICP4)	Digital pin 49
36	PL1 (ICP5)	Digital pin 48
37	PL2 (T5)	Digital pin 47

38	PL3 (OC5A)	Digital pin 46 (PWM)
39	PL4 (OC5B)	Digital pin 45 (PWM)
40	PL5 (OC5C)	Digital pin 44 (PWM)
41	PL6	Digital pin 43
42	PL7	Digital pin 42
43	PD0 (SCL/INT0)	Digital pin 21 (SCL)
44	PD1 (SDA/INT1)	Digital pin 20 (SDA)
45	PD2 (RXD1/INT2)	Digital pin 19 (RX1)
46	PD3 (TXD1/INT3)	Digital pin 18 (TX1)
47	PD4 (ICP1)	
48	PD5 (XCK1)	
49	PD6 (T1)	
50	PD7 (T0)	Digital pin 38
51	PG0 (WR)	Digital pin 41
52	PG1 (RD)	Digital pin 40
53	PC0 (A8)	Digital pin 37
54	PC1 (A9)	Digital pin 36
55	PC2 (A10)	Digital pin 35
56	PC3 (A11)	Digital pin 34
57	PC4 (A12)	Digital pin 33
58	PC5 (A13)	Digital pin 32
59	PC6 (A14)	Digital pin 31
60	PC7 (A15)	Digital pin 30
61	VCC	VCC
62	GND	GND
63	PJ0 (RXD3/PCINT9)	Digital pin 15 (RX3)
64	PJ1 (TXD3/PCINT10)	Digital pin 14 (TX3)
65	PJ2 (XCK3/PCINT11)	
66	PJ3 (PCINT12)	
67	PJ4 (PCINT13)	
68	PJ5 (PCINT14)	
69	PJ6 (PCINT 15)	
70	PG2 (ALE)	Digital pin 39
71	PA7 (AD7)	Digital pin 29
72	PA6 (AD6)	Digital pin 28
73	PA5 (AD5)	Digital pin 27
74	PA4 (AD4)	Digital pin 26
75	PA3 (AD3)	Digital pin 25
76	PA2 (AD2)	Digital pin 24

77	PA1 (AD1)	Digital pin 23
78	PA0 (AD0)	Digital pin 22
79	PJ7	
80	VCC	VCC
81	GND	GND
82	PK7 (ADC15/PCINT23)	Analog pin 15
83	PK6 (ADC14/PCINT22)	Analog pin 14
84	PK5 (ADC13/PCINT21)	Analog pin 13
85	PK4 (ADC12/PCINT20)	Analog pin 12
86	PK3 (ADC11/PCINT19)	Analog pin 11
87	PK2 (ADC10/PCINT18)	Analog pin 10
88	PK1 (ADC9/PCINT17)	Analog pin 9
89	PK0 (ADC8/PCINT16)	Analog pin 8
90	PF7 (ADC7)	Analog pin 7
91	PF6 (ADC6)	Analog pin 6
92	PF5 (ADC5/TMS)	Analog pin 5
93	PF4 (ADC4/TMK)	Analog pin 4
94	PF3 (ADC3)	Analog pin 3
95	PF2 (ADC2)	Analog pin 2
96	PF1 (ADC1)	Analog pin 1
97	PF0 (ADC0)	Analog pin 0
98	AREF	Analog Reference
99	GND	GND
100	AVCC	VCC

2.1.3- Caracterização do Arduino Mega

Na figura 1 estão indicadas as pinagens e na figura 2 os Ports do Arduino Mega.

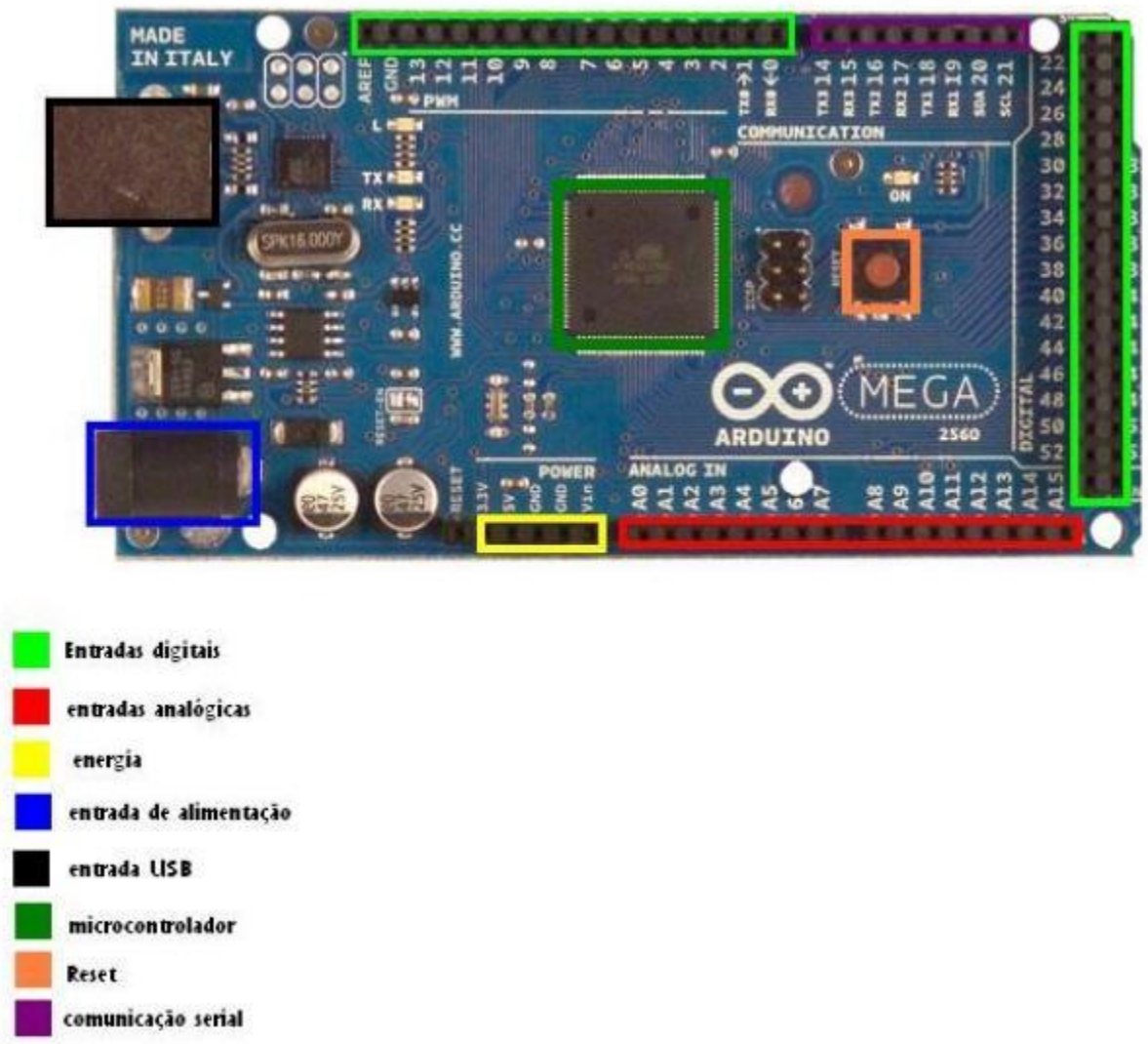


Figura 1 - Pinagens

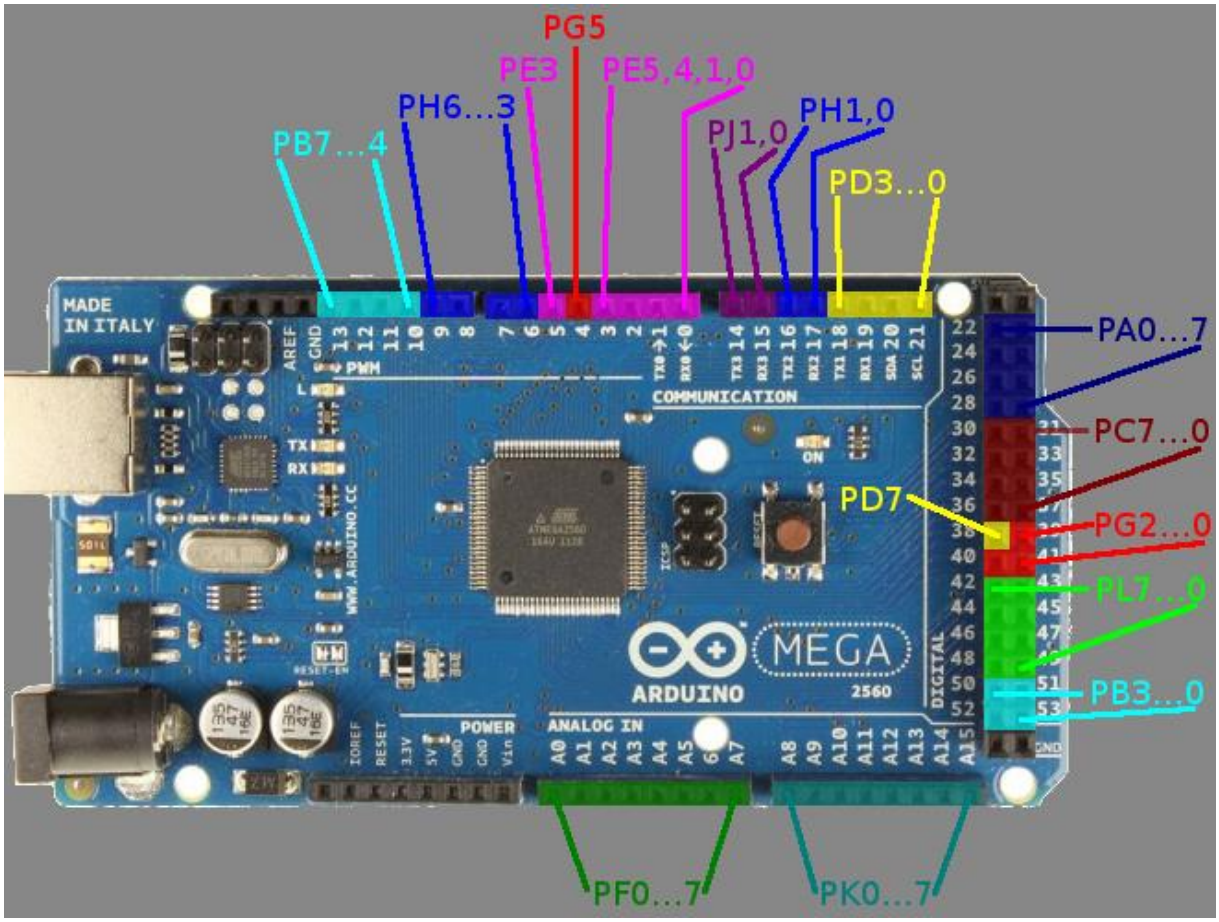


Figura 2 - Ports

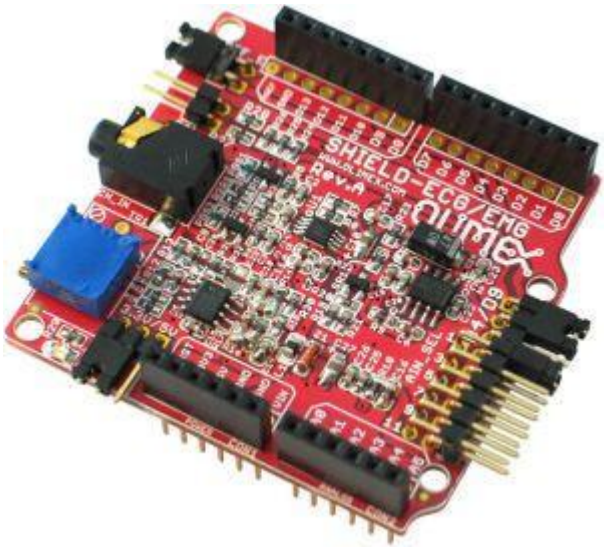
3- Dispositivos Periféricos

Os Dispositivos Periféricos são acoplados ao Arduino para diversas aplicações. Dependendo do enfoque do projeto, um Arduino pode conter um shield (placa específica para determinada aplicação) ou mais de um e periféricos, como por exemplo: sensores, servo motores, entre outros.

3.1- Shields

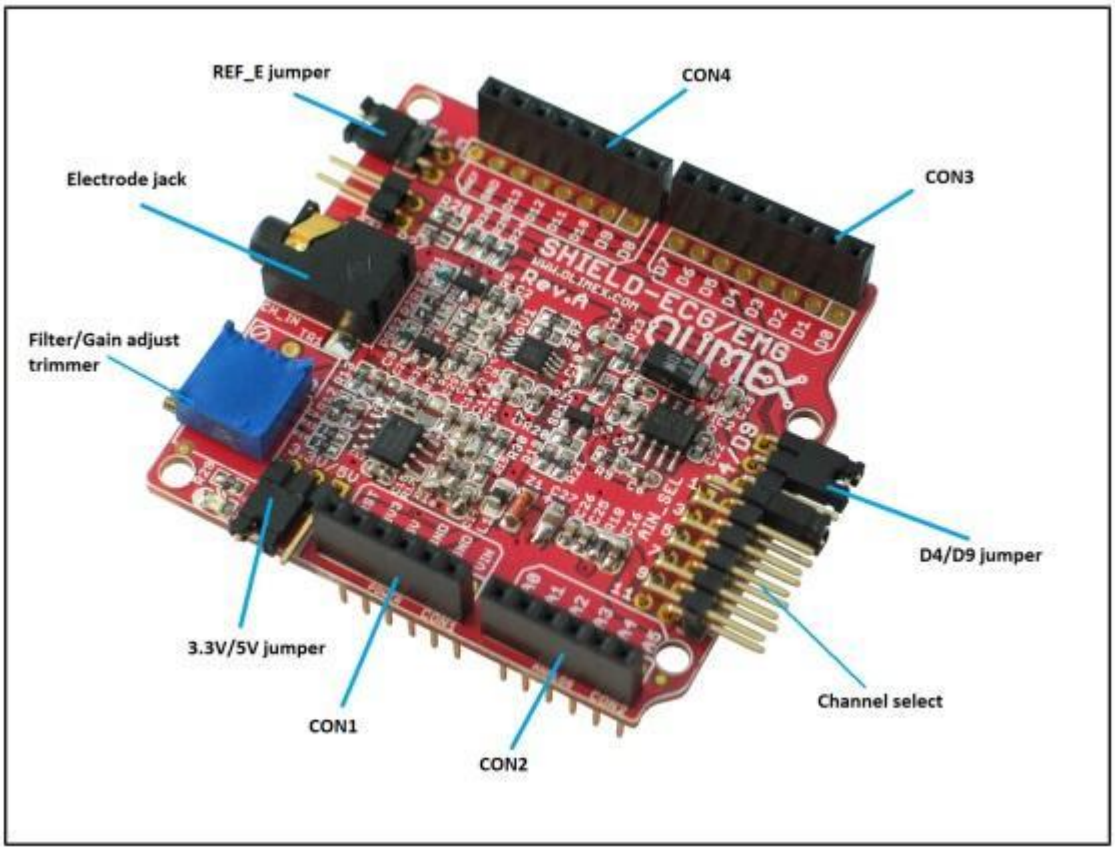
Shields *(em inglês: escudo)* são placas de circuito impresso com uma função específica. São plugados no topo da placa do Arduino, extendendo suas aplicações. Os diferentes shields seguem a mesma filosofia: são fáceis de montar e baratas para produzir.

3.2- Shield EKG/EMG

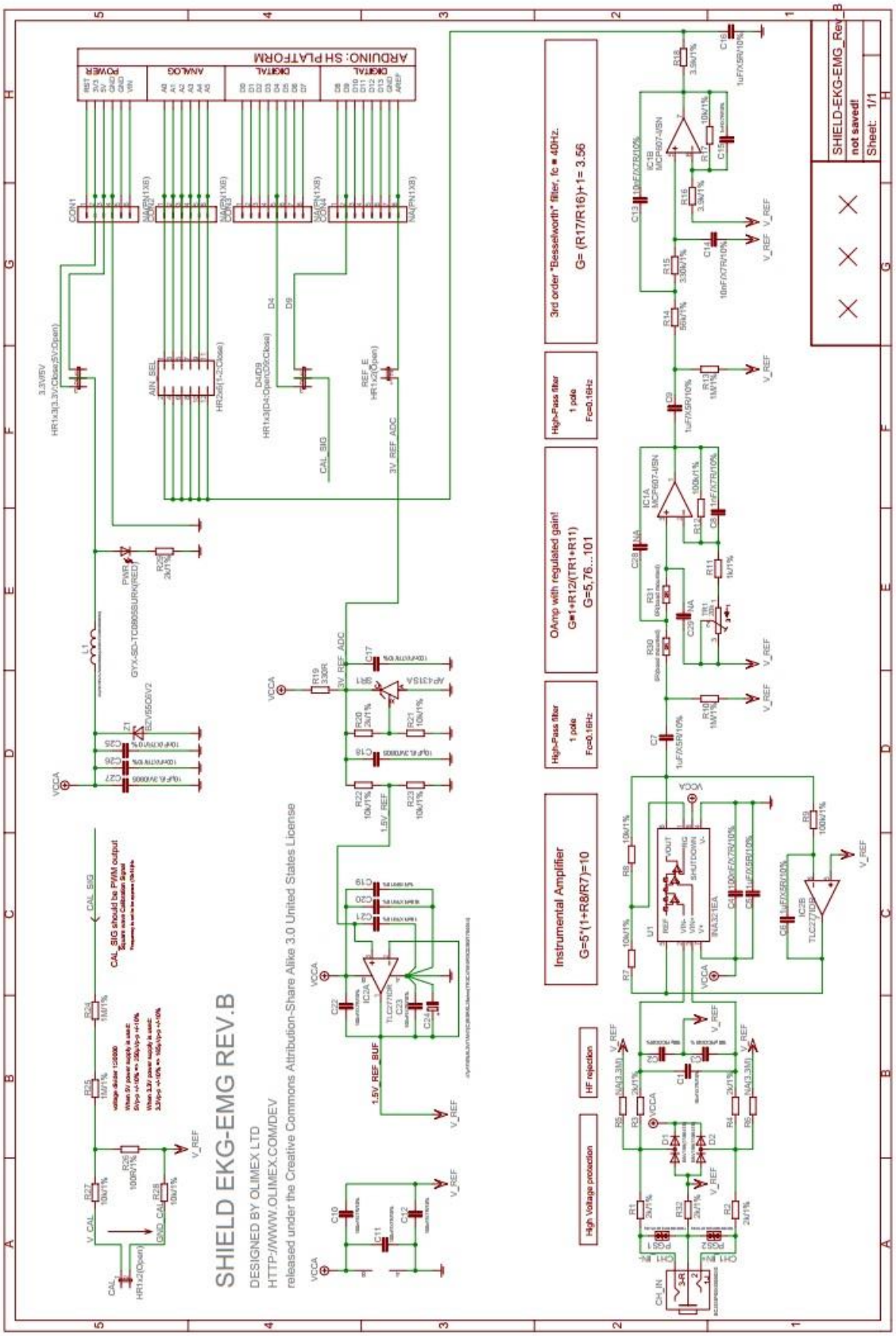


O Shield EKG-EMG é um módulo de extensão do Arduino. Foi desenvolvido com o intuito de ser um protótipo de Eletrocardiograma e Eletromiografia para fins de estudos e programação com Arduino.

3.2.1- Layout


REF_E jumper
CON4
Electrode jack
CON3
Filter/Gain adjust trimmer
D4/D9 jumper
3.3V/5V jumper
Channel select
CON1
CON2

3.2.2- Esquema elétrico do shield



O esquema do EKG é simples, pois seu funcionamento se resume na conversão de dados analógicos para digital. Seus pinos da Con1 são conectados de acordo com o do Arduino e a posição do jumper (3,3V ou 5V). A Con2 é destinada aos dados analógicos, portanto, os dados de entrada. Os dados de entrada são obtidos a partir do sensor conectado no EKG. Após a recepção dos dados, os mesmos serão transmitidos para o Arduino através da conexão entre os pinos da Con2 e o Arduino. A Con3 e a Con4 são destinadas aos dados digitais e é onde ocorrerá a conversão dos dados analógicos para digitais, utilizando, como método de conversão, o CAL_SIG, sinal de onda quadrada. Estes dados então serão transmitidos para o Arduino que por sua vez transmite para o computador onde é possível visualizar sua forma de onda utilizando os softwares ElecGuru ou FreeHC.

O sensor conectado ao EKG é utilizado para a coleta de dados a partir de uma pessoa. Estes dados então serão transmitidos para o EKG, onde passarão por diversos circuitos até serem recebidos pela Con2. Primeiramente, os dados serão testados de acordo com sua tensão e frequência. O EKG é protegido contra alta tensão e irá rejeitar qualquer frequência alta, visto que para um eletrocardiograma as frequências e tensões coletadas são baixas. Em seguida o sinal será amplificado em 10 vezes e terão suas componentes de alta frequência cortadas de acordo com a frequência de corte de 0.16Hz. O sinal passará novamente por um amplificador com ganho regulado entre 5,76 a 101, cortando novamente as frequências acima de 0.16Hz. Finalmente, o sinal passa por um filtro de terceira ordem de Besselworth com frequência de corte 40Hz e é amplicado em 3.56x, chegando assim na Con2.

3.2.3- Conectores no Arduino

Os conectores seguem o padrão Arduino para conexão de shields. O shield vem com conectores soldados, prontos para montar e compatíveis com as placas, sendo possível montar mais de um shield no sistema.

Pin #	POWER CON1	ANALOG CON2	DIGITAL CON3	DIGITAL CON4
1	RST	A0	D0	D8
2	3.3V	A1	D1	D9
3	5V	A2	D2	D10
4	GND	A3	D3	D11
5	GND	A4	D4	D12
6	Vin	A5	D5	D13
7	-	-	D6	GND
8	-	-	D7	AREF

6 e 8 pinos conectados e montados (COM1, COM2, COM 3 E COM4)

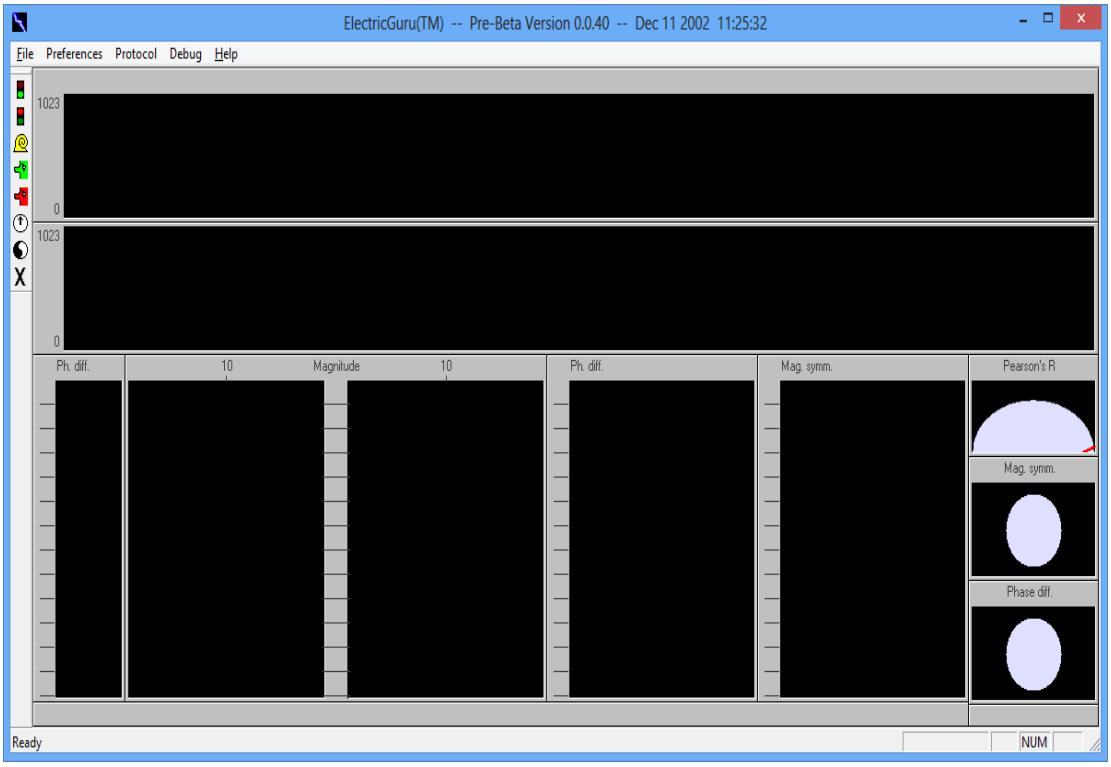
3.3- Sensor Eletrodo Passivo

Os sensores eletrodos passivos para o Shield-EKG-EMG permitem ao Arduino capturar sinais de Eletrocardiografia/Eletromiografia. Você pode monitorar seus batimentos cardíacos e registrá-los pelo seu pulso ou ainda reconhecer movimentos monitorando e analisando as atividades do seu músculo. Este sensor é conectado ao shield, trabalhando como receptor de sinais para o estudo.

Os eletrodos são rotulados como L para braço esquerdo, R para braço direito e D para terra.

4- Software

Para testar o funcionamento do projeto, será utilizado o código (Anexo I) fornecido pelo fabricante do shield, com as devidas modificações necessárias. Também será utilizado outro software, o ElecGuru, também fornecido pela Olimex, para testar, capturar e registrar os dados obtidos pelo ECG no próprio computador.



ElecGuru

5- Experimento

5.1- Download de softwares e bibliotecas

Para a realização deste experimento será necessário o download de biblioteca e softwares específicos para o funcionamento do shield EKG/EMG (*EKG no inglês - ECG*) fabricando pela empresa Olimex. Site do distribuidor da Olimex para download dos arquivos: (*http://microcontrollershop.com/product_info.php?products_id=4648*).

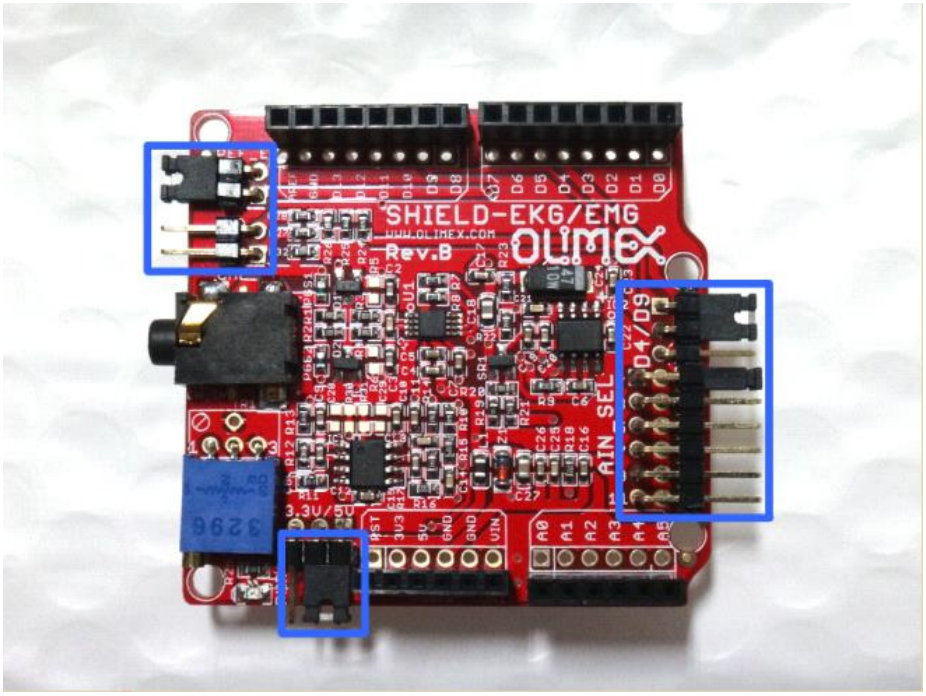
Segue abaixo a lista dos arquivos que deverão ser baixados:

- Electric Guru monitoring software;
  Irá monitorar e exibir as formas de onda do eletrocardiograma do indivíduo.
- Arduino Demo para uso com Olimexino-328 (ou Standard Arduino) e o Electric Guru Software.
  Arduino Demo será o arquivo fonte (modelo) que deverá ser inserido na plataforma IDE do Arduino (ShieldEkgEmgDemo.ino).

Após o download dos softwares da Olimex, a biblioteca necessária é a FlexiTimer2. Esta biblioteca pode ser encontrada no próprio site do Arduino (*http://playground.arduino.cc/Main/FlexiTimer2*). A biblioteca FlexiTimer2 é usada para setup do canal analógico da amostra de frequências e pacote de update. Sempre que ocorrer interrupções, o pacote de leitura é enviado para o PC. Além disso, o CAL_SIG também é gerado, não sendo necessário utilizar a biblioteca *TimerOne.*

5.2- Configuração de jumpers do shield EKG/EMG

Depois do download dos arquivos, antes de iniciar o experimento, deve-se seguir o passo-a-passo para a configuração de pinos do shield. Nesta plataforma estão presentes soquetes e pinos fixos e outros pinos os quais podem ser alterados pelo usuário através de jumpers.



Shield EKG/EMG com destaque para os jumpers

Existem 5 jumpers no shield EKG/EMG:

- **3.3V/5V**

O jumper de tensão controla o circuito de potência. Pode ser alimentado com 3.3V ou 5V. O estado padrão é o de 3.3V.

- **REF_E**

Se for utilizado apenas um shield, este jumper tem que ser fechado. Porém, se for utilizado mais de um shield, o primeiro deverá estar fechado, enquanto que os Shields montados acima deverão estar abertos (sem jumper). O padrão é fechado.

- **AIN_SEL**

Jumper responsável por qual canal o shield EKG/EMG irá utilizar. Se você utilizar mais de um shield EKG/EMG, deverá ter o primeiro shield no jumper 1, o segundo shield no jumper 3 e assim por diante. O padrão é o jumper na posição 1.

- **D4/D9**

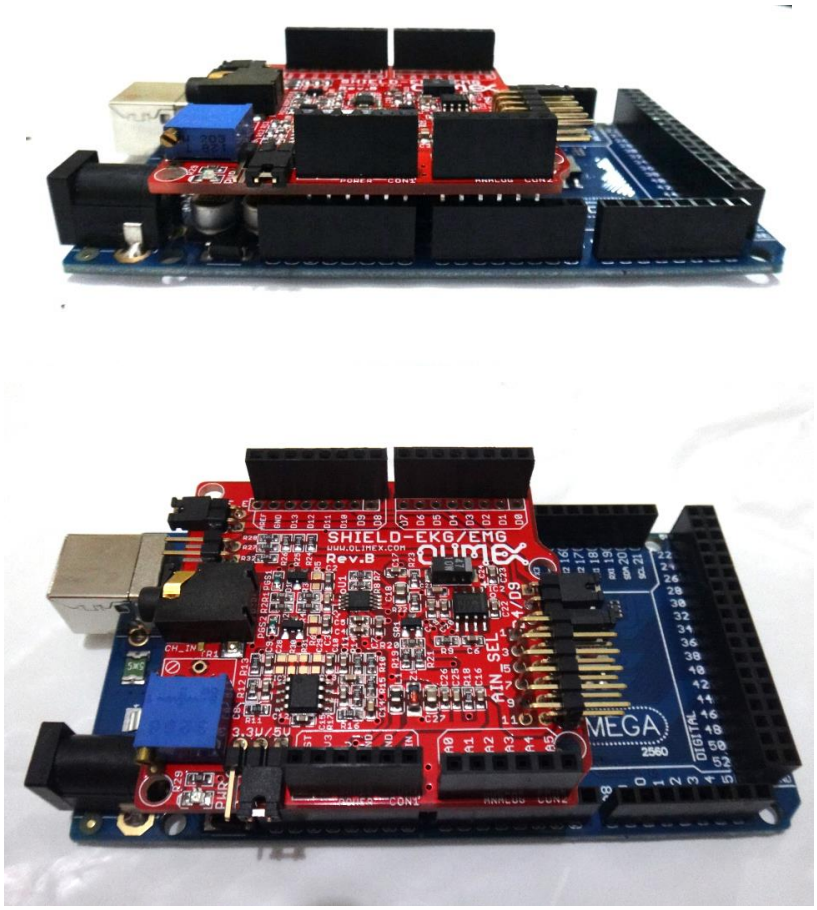
Controla os pinos D4/D9. Alguns processadores utilizam o pino D9 como padrão, então se for necessário deverá alterá-lo para o pino D4. O padrão é o D9.

- **CAL**

Jumper usado como feedback da calibração e requer um cabo adicional para sua utilização.

5.3- Montagem

Como neste estudo de caso foi utilizado o Arduino Mega 2560, as imagens a seguir exemplificam como ficará encaixado o shield no arduino.



Shield EKG/EMG acoplado no Arduino Mega.

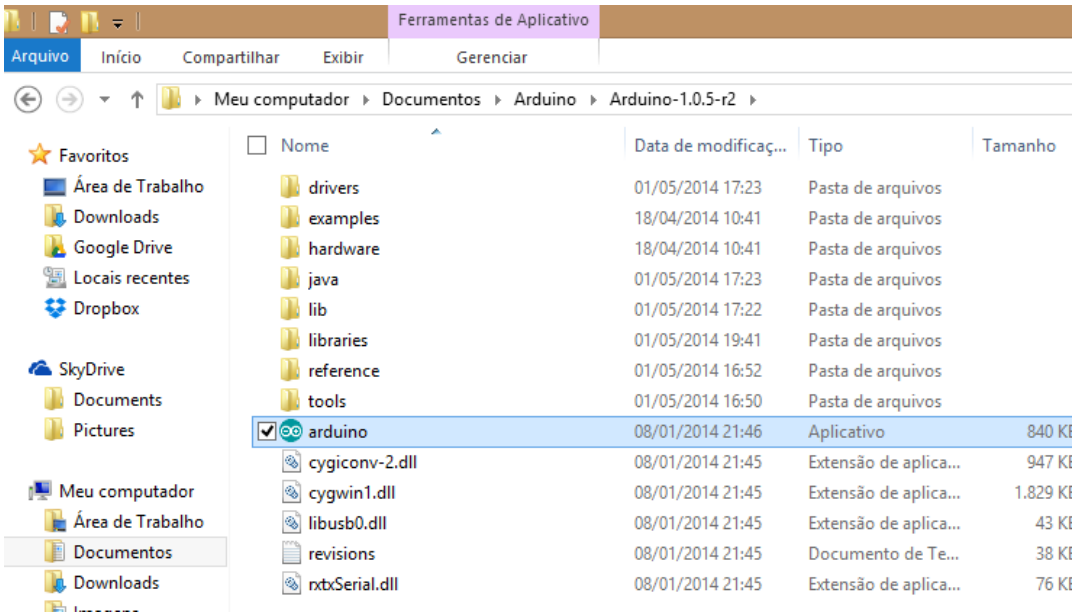
Os pinos do shield que deverão ser encaixados nos pinos correspondentes do arduino estão demonstrados na tabela da seção 3.2.2 deste documento.

Caso o usuário utilize o Arduino Uno, a configuração de montagem será a mesma. A única diferença é que não terá tantos ports sem conexão como no Arduino Mega.

Conecte o cabo USB na porta do PC e no Arduino. Insira o os cabos do sensor e pronto! A montagem está finalizada.

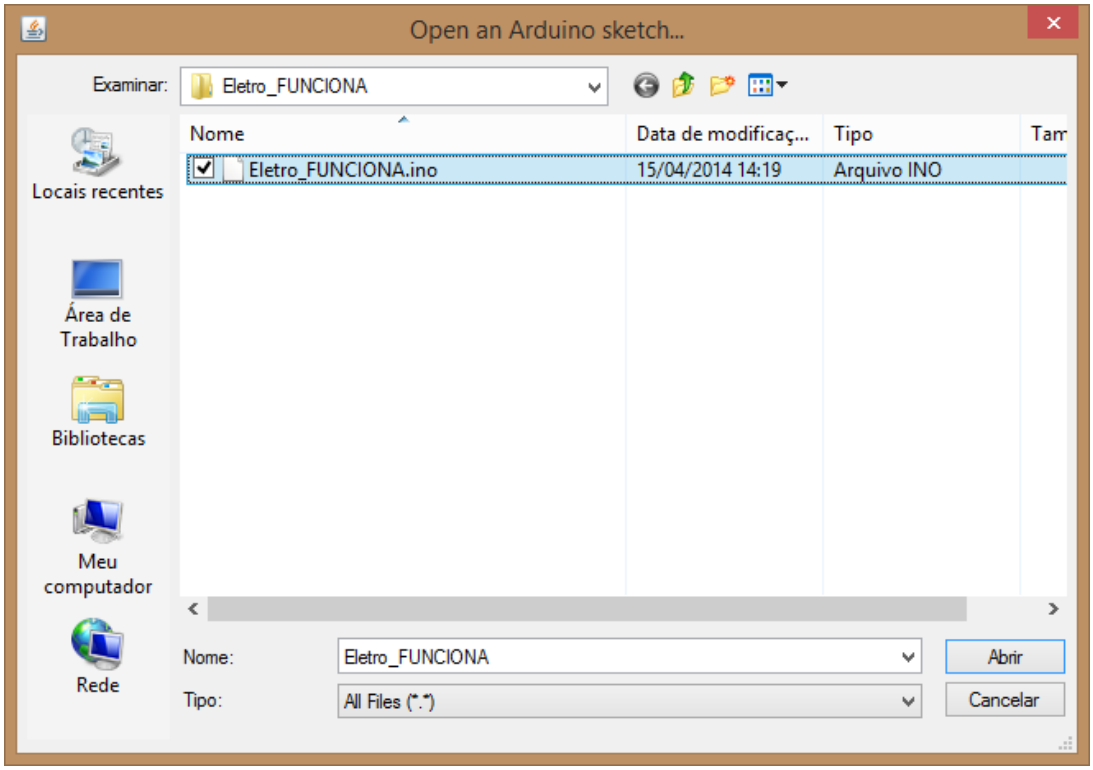
5.4- Configuração de software

1) Abra o software do ambiente de desenvolvimento do Arduino.



2) Configure a porta COM de acordo com as opções disponíveis.

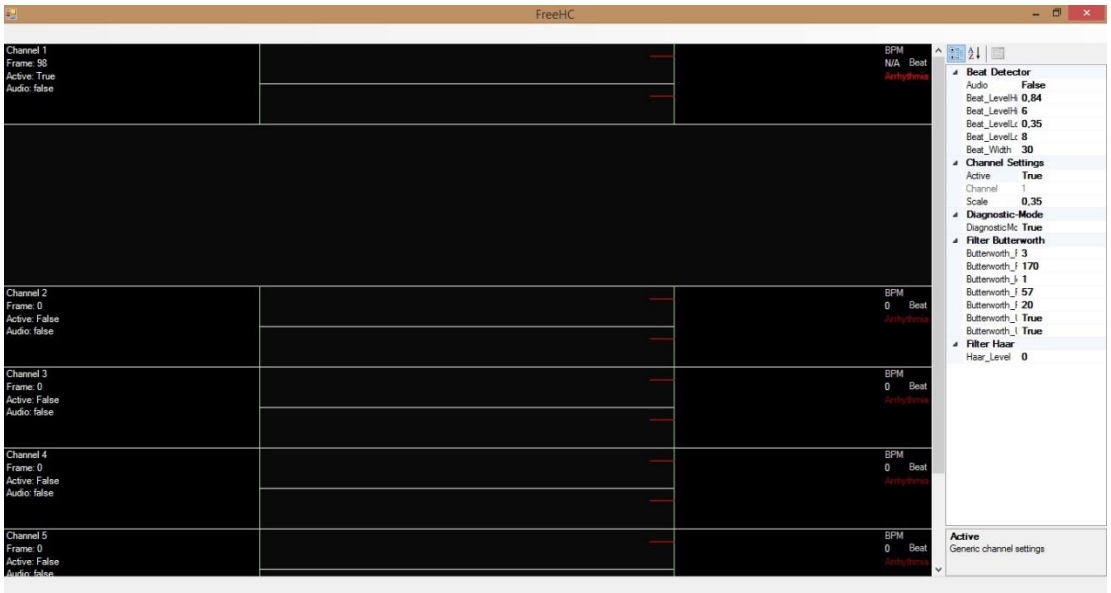
3) Vá em File > Open... e abra o arquivo .ino.



4) Clique na opção para verificar se o código possui erros e depois faça o upload para a placa.

5) Abra o software de monitoramento das formas de onda do eletrocardiograma. Escolha entre os dois softwares disponíveis neste projeto: ElecGuru e FreeHC.

Electric Guru

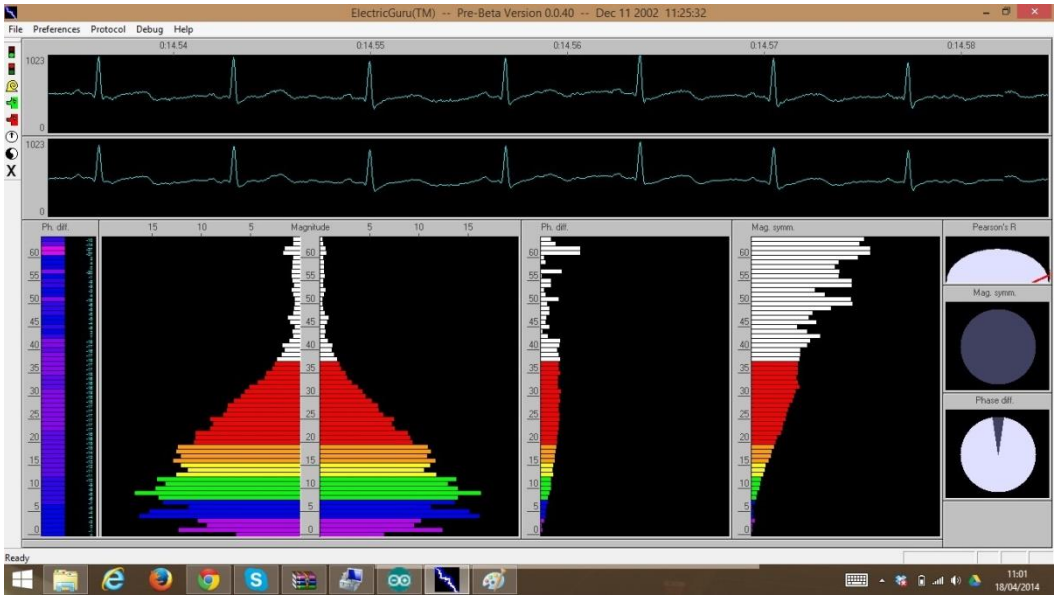


FreeHC

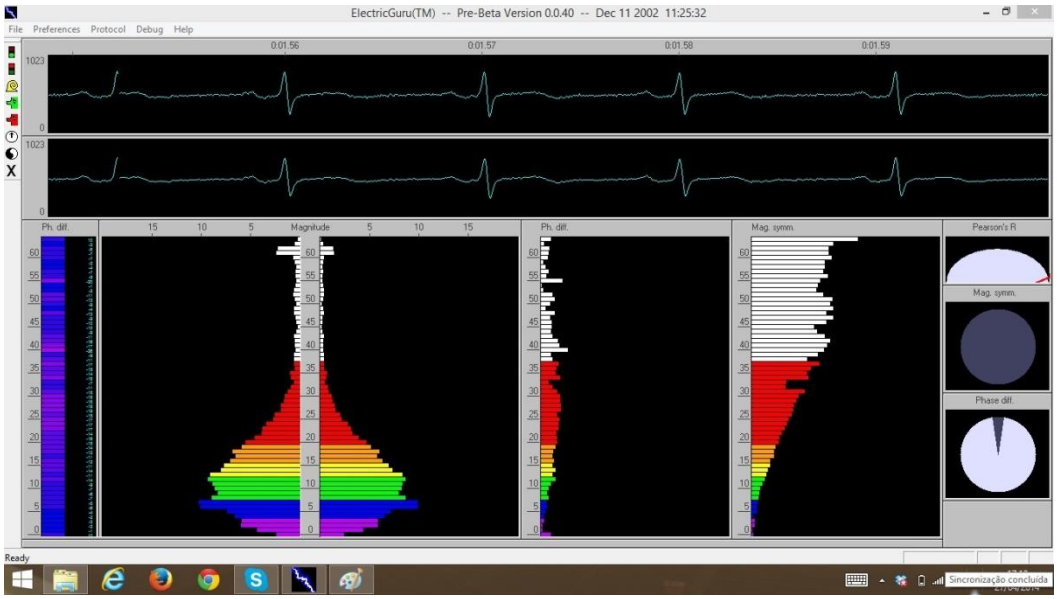
6) Existem 3 cabos dos sensores (Right, Left e D para o terra). O cabo Right deve ser colocado no pulso direito, o Left no esquerdo e o D no tornozelo direito. É importante o uso deste último sensor, pois ele é o terra e será ele que completará a polarização analisada pelos sensores.

5.5- Exemplo de análises

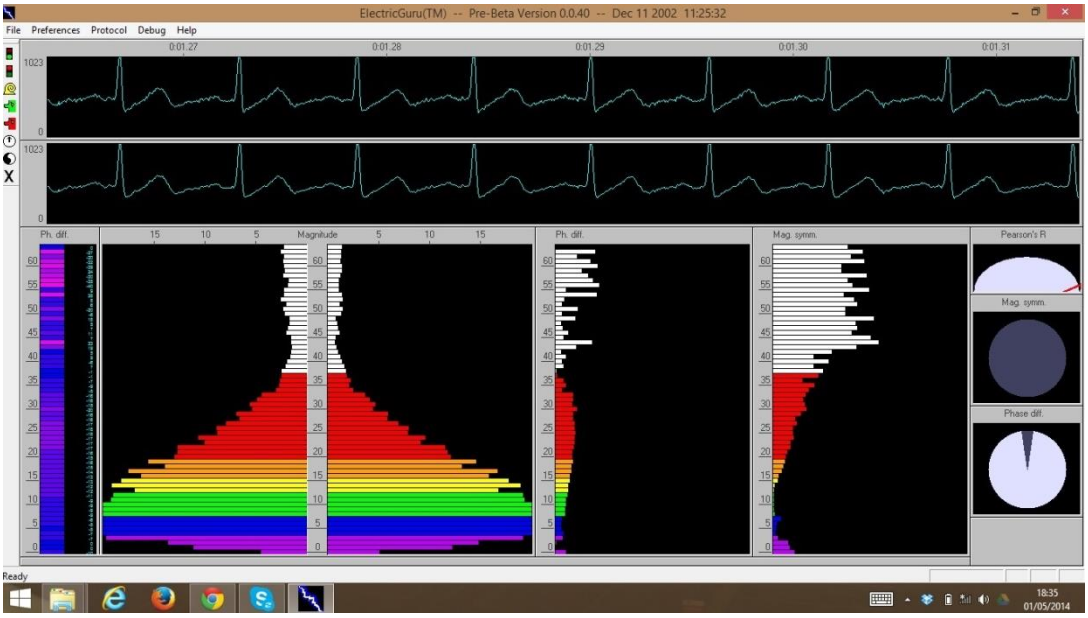
A seguir estão exemplos de eletrocardiogramas coletados de voluntários para estudo do funcionamento do ECG para arduino.



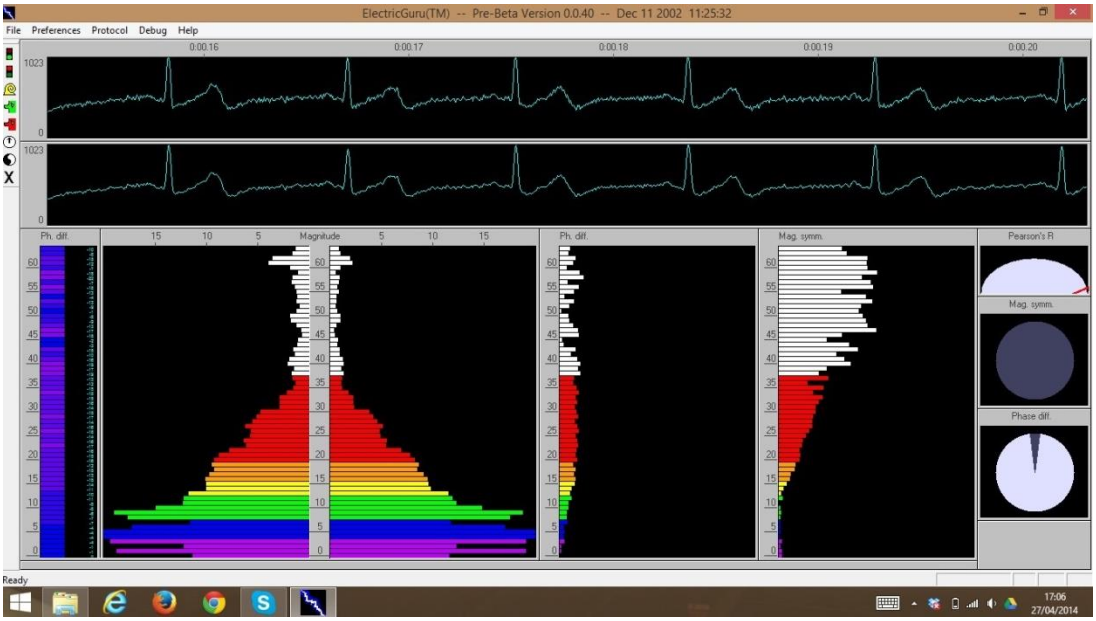
Homem de 56 anos sem problemas cardíacos.



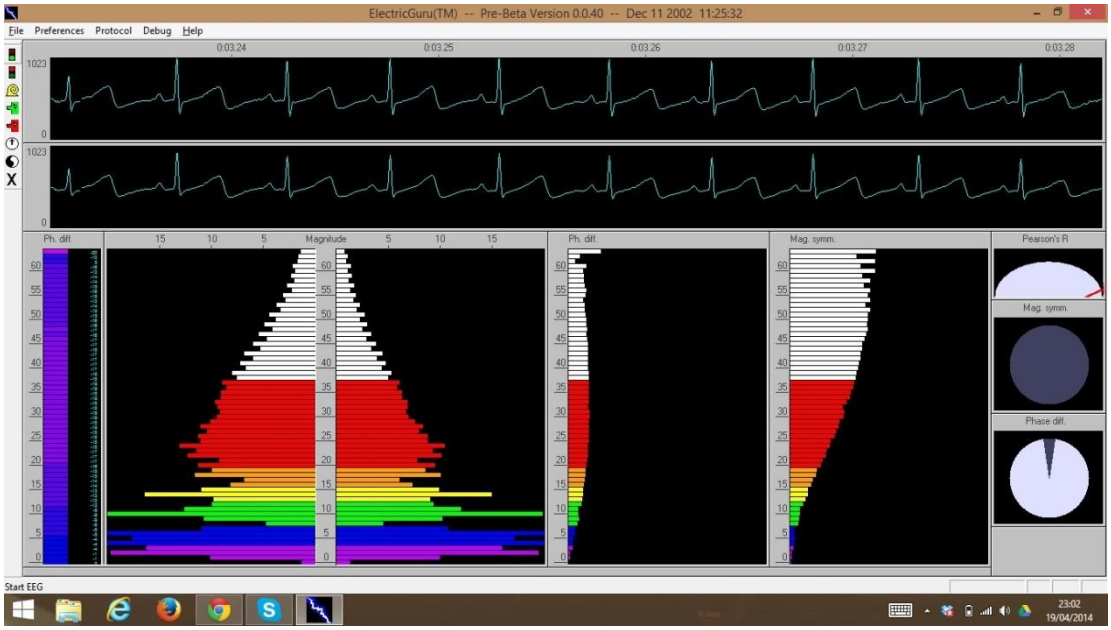
Mulher de 26 anos que fez cirurgia cardíaca (ablação).



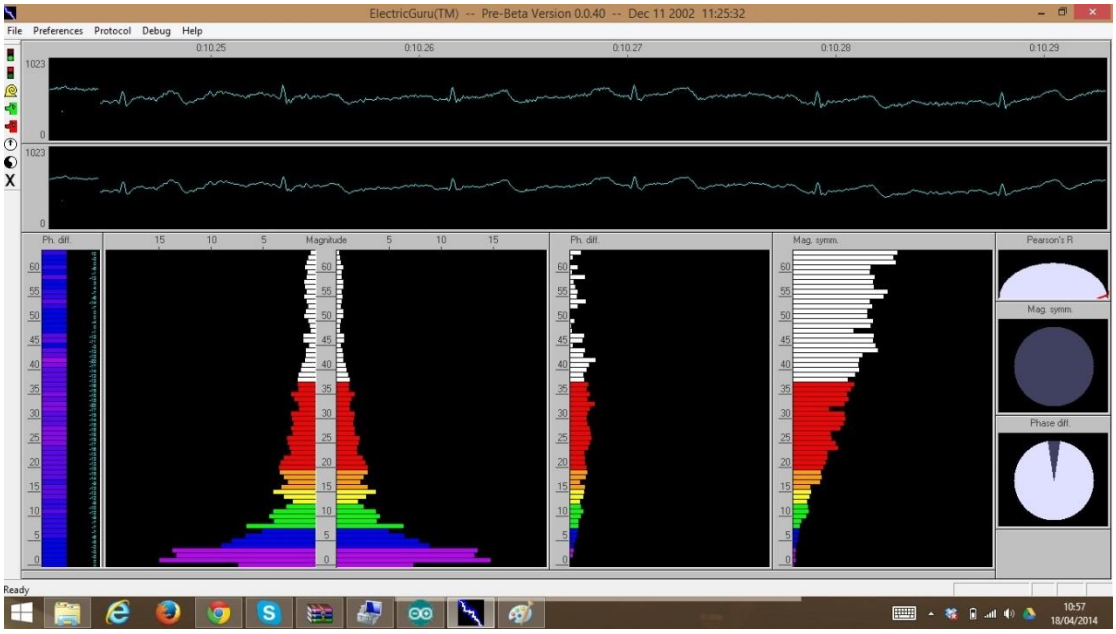
Homem de 22 anos sem problemas cardíacos.



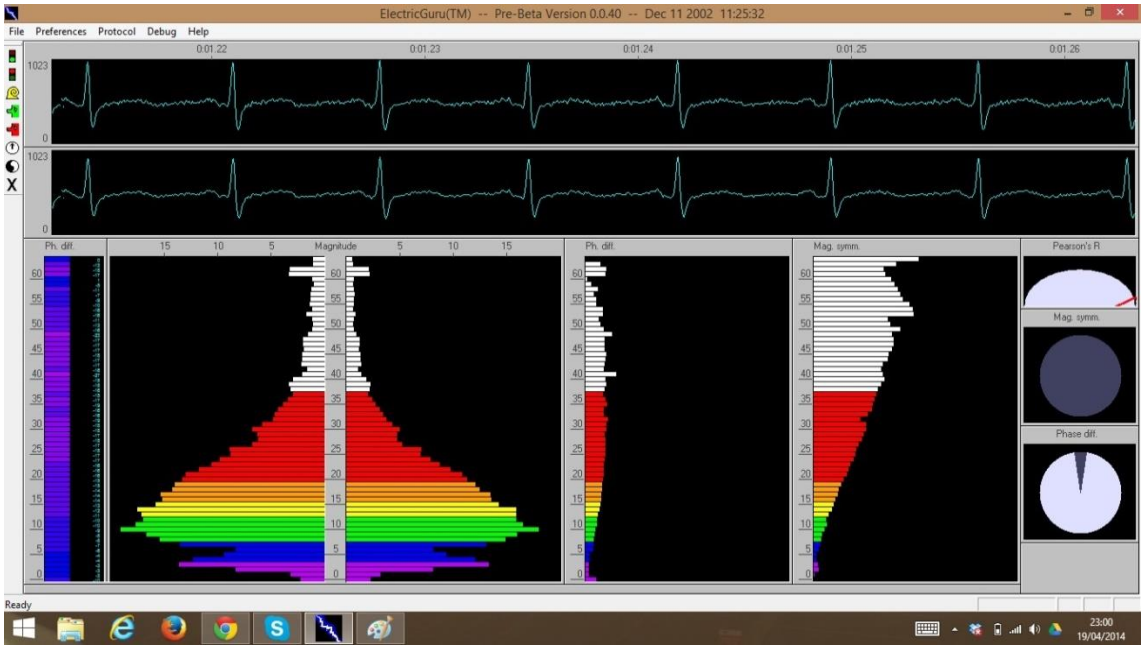
Homem de 24 anos sem problemas cardíacos.



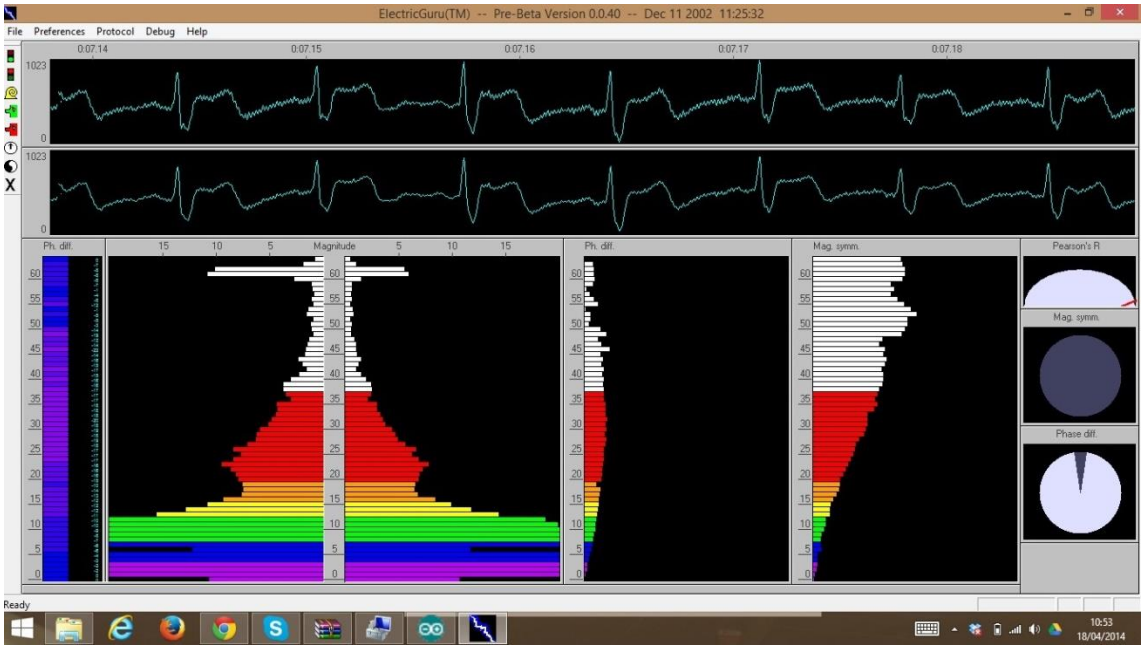
Criança do sexo feminino de 2 anos de idade sem problemas cardíacos.



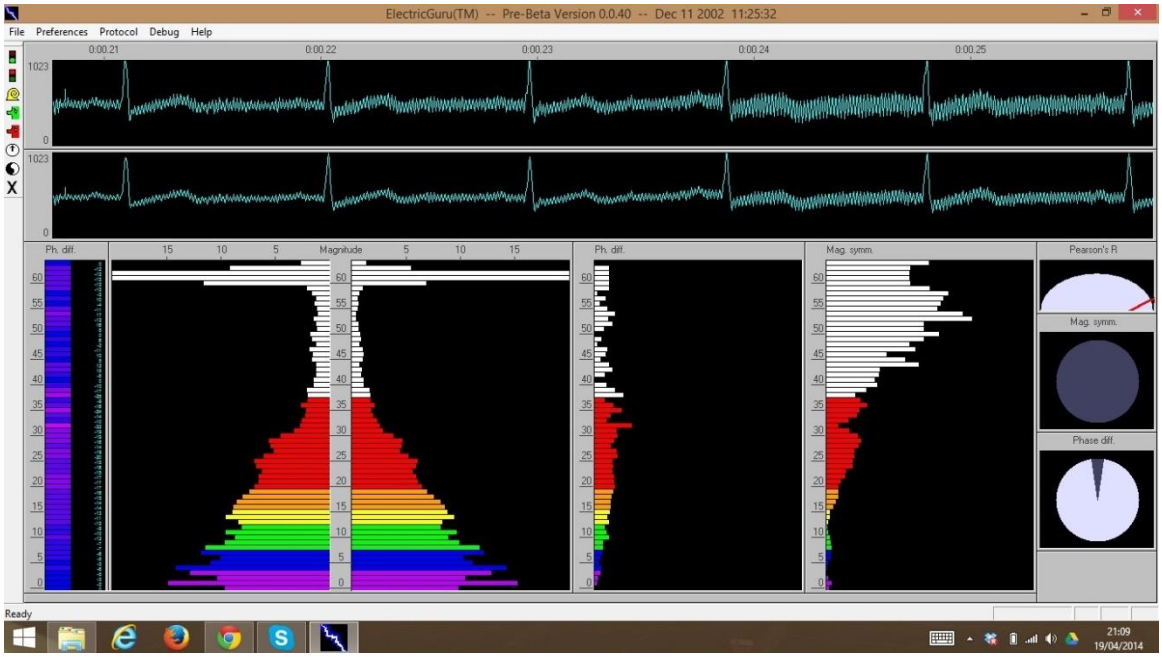
Mulher de 36 anos sem problemas cardíacos.



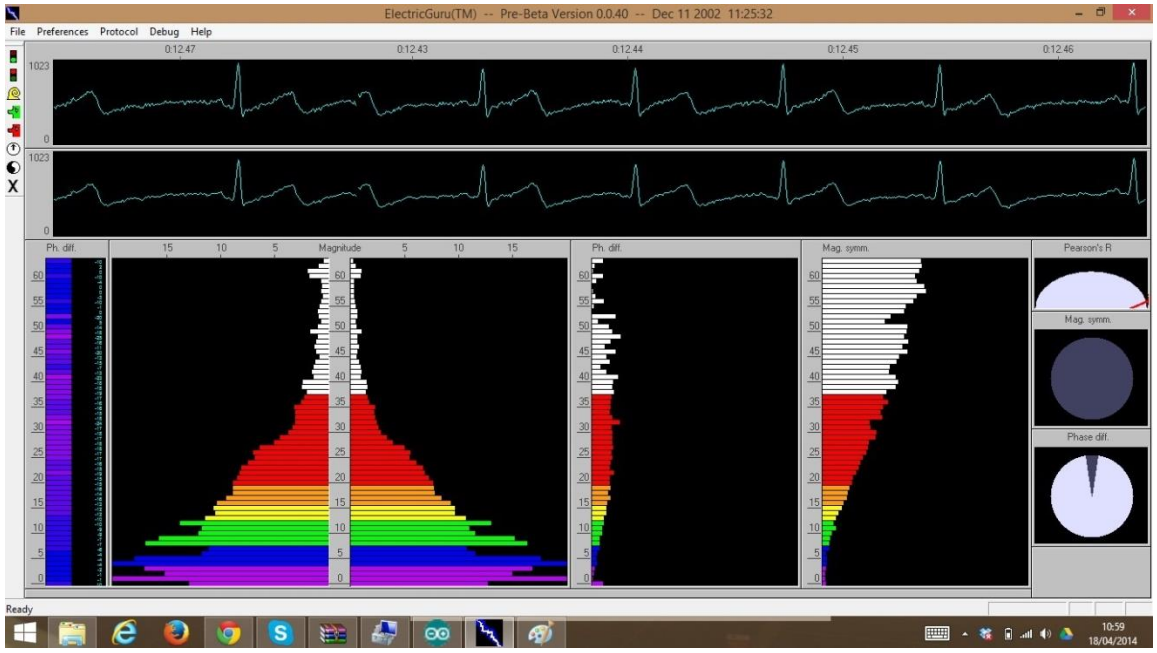
Mulher de 51 anos sem problemas cardíacos.



Mulher de 55 anos com isquemia.

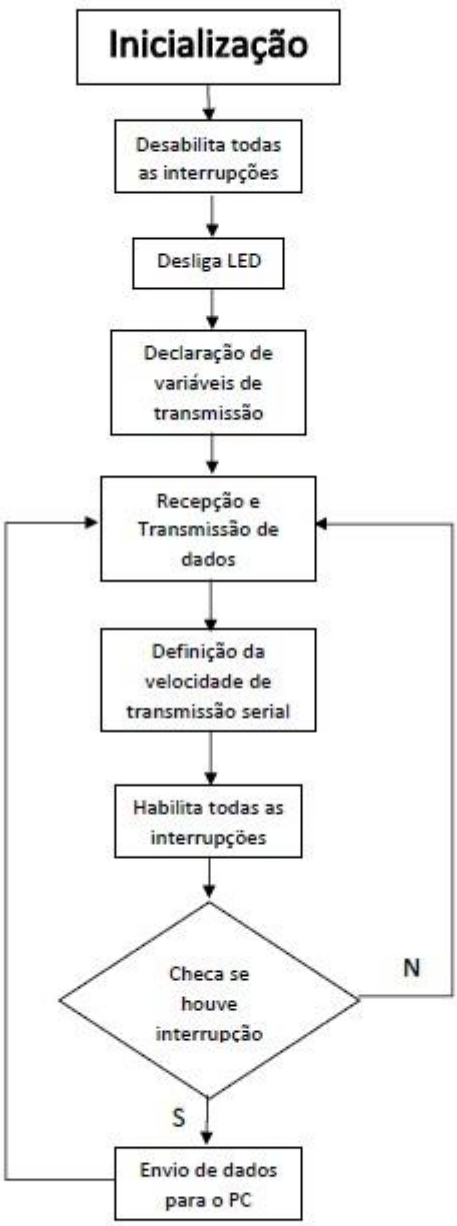


Homem de 69 anos sem o uso do sensor D (terra). Sinal com bastante ruído.



Homem de 22 anos sem problemas cardíacos.

6- Explicação do código em linguagem C



INICIALIZAÇÃO

Esta etapa consiste na inclusão de bibliotecas auxiliares, definição do número de canais a ser utilizado, tamanho do cabeçalho, número de pacotes, frequência de amostragem, a port do LED e do CAL_SIG a serem usados, variáveis voláteis a serem utilizadas e funções.

Para este projeto, os valores adotas foram conforme a seguir:

- Número de Canais = 6
- Tamanho do Cabeçalho = 4

- Número de pacotes = Número de Canais x 2 + Tamanho do Cabeçalho + 1
- Frequência de amostragem = 256Hz
- Led = 13
- Cal_Sig = 9

Foram criadas funções para desligar e ligar o LED e o Cal_Sig. O Cal_Sig é uma saída PMW que calibra o sinal de uma onda quadrada.

Programa

Inicialmente o código desabilita todas as interrupções para que o programa colete os dados necessários sem que aja qualquer interrupção. O LED inicializado é desligado, sendo ligado posteriormente quando necessário.

As variáveis referentes ao número de pacotes são escritas durante a etapa de declaração das variáveis de transmissão, conforme lista abaixo:

- TXBuf[0] = 0xa5;
- TXBuf[1] = 0x5a;
- TXBuf[2] = 2;
- TXBuf[3] = 0;
- TXBuf[4] = 0x02;
- TXBuf[5] = 0x00;
- TXBuf[6] = 0x02;
- TXBuf[7] = 0x00;
- TXBuf[8] = 0x02;
- TXBuf[9] = 0x00;
- TXBuf[10] = 0x02;
- TXBuf[11] = 0x00;
- TXBuf[12] = 0x02;
- TXBuf[13] = 0x00;
- TXBuf[14] = 0x02;
- TXBuf[15] = 0x00;
- TXBuf[2 * NUMCHANNELS + HEADERLEN] = 0x01;

Neste código é utilizada a biblioteca FlexiTimer2 e conforme explicado anteriormente, sua função e definir os canais analógicos para a frequência de amostragem e a atualização dos dados. Durante sua execução utilizada como dados de entrada o valor determinado pela divisão de 1024 pela frequência de amostragem e uma outra função que lê os seis dados de entrada ADC e os armazena nos pacotes. Seu funcionamento ocorre na etapa de recepção e transmissão de dados. Durante a execução da função também é gerado o sinal CAL_SIG.

A próxima etapa consiste na definição da velocidade de transmissão serial, definida em 57600 bps neste projeto. Em seguida é habilitado todas as interrupções, onde no caso de que ocorra uma, o EKG irá transmitir os dados armazenados para o computador, onde será possível visualizar uma onda com o auxílio dos softwares ElecGuru ou FreeHC. Em seguida, independente se houve ou não interrupção, o software retornará a etapa de recepção e transmissão de dados, realizando as operações citadas anteriormente novamente.

7- Comentários e Observações

Depois do desenvolvimento do ECG para Arduino, identificou-se a necessidade de aprimoramento deste dispositivo para fins mais sofisticados em engenharia, bem como a produção de um filtro que possa atenuar o ruído visto em alguns testes com voluntários. Tal ruído tem como consequência a exposição a iluminação e ruídos sonoros no ambiente. Para uma análise ideal, sugere-se ambiente escuro e com pouco ou nada de ruído sonoro, assim o resultado no software de monitoramento terá uma aparência mínima de interferência.

Com o intuito de melhorar o dispositivo, futuramente podem ser implementadas aplicações adicionais, como por exemplo: shield ethernet, GPRS e LCD.

O projeto encontra-se disponível no Youtube:
https://www.youtube.com/watch?v=DAo6LSyC3tQ

8- Conclusão

É importante salientar a importância da engenharia não só em projetos que envolvem tecnologias que nos rodeiam, como telefonia, bancos, automóveis, bem como a relação da engenharia com a área médica. Apesar de ser uma área pouco divulgada e explorada no Brasil, é de suma importância para que aparelhos funcionem adequadamente, gerem resultados exatos e precisos e para que médicos consigam analisar exames, cirurgias e outras medidas cabíveis na saúde com a devida tecnologia de ponta graças aos engenheiros que possuem conhecimento específico neste setor. Sendo assim, os engenheiros tem a responsabilidade de manter, corrigir e orientar o funcionamento, desenvolvimento e manutenção destes aparelhos e dispositivos hospitalares.

10- Referência Bibliográfica

http://destacom.ufms.br/mediawiki/images/9/9f/Arduino_Destacom.pdf

http://microcontrollershop.com/product_info.php?products_id=4648

http://microcontrollershop.com/product_info.php?products_id=4834

https://www.olimex.com/Products/Duino/Shields/SHIELD-EKG-EMG/open-source-hardware

http://www.picstopin.com/243/-topics-arduino-duemilanove-projects-led-serial/

http://pt.wikipedia.org/wiki/Arduino#Acess.C3.B3rios

http://www.wordreference.com/enpt/shield

Anexo I

```
/************************************************************/
/* Demo program for:                                        */
/*    Board: SHIELD-EKG/EMG + Olimexino328                  */
/*  Manufacture: OLIMEX                                     */
/*  COPYRIGHT (C) 2012                                      */
/*  Designed by:  Penko Todorov Bozhkov                     */
/*   Module Name:   Sketch                                  */
/*   File   Name:   ShieldEkgEmgDemo.ino                    */
/*   Revision:  Rev.A                                       */
/*    -> Added is suppport for all Arduino boards.          */
/*       This code could be recompiled for all of them!     */
/*   Date: 19.12.2012                                       */
/*   Built with Arduino C/C++ Compiler, version: 1.0.3      */
/************************************************************/
/************************************************************
Purpose of this programme is to give you an easy way to
connect Olimexino328 to ElectricGuru(TM), see:
https://www.olimex.com/Products/EEG/OpenEEG/EEG-SMT/resources/ElecGuru40.zip
where you'll be able to observe yours own EKG or EMG signal.
It is based on:
************************************************************
* ModularEEG firmware for one-way transmission, v0.5.4-p2
* Copyright (c) 2002-2003, Joerg Hansmann, Jim Peters, Andreas Robinson
* License: GNU General Public License (GPL) v2
************************************************************
For proper communication packet format given below have to be supported:
////////////////////////////////////////////////
////////// Packet Format Version 2 ////////////
////////////////////////////////////////////////
// 17-byte packets are transmitted from Olimexino328 at 256Hz,
// using 1 start bit, 8 data bits, 1 stop bit, no parity, 57600 bits per second.

// Minimial transmission speed is 256Hz * sizeof(Olimexino328_packet) * 10 = 43520 bps.

struct Olimexino328_packet
{
  uint8_t       sync0;          // = 0xa5
  uint8_t       sync1;          // = 0x5a
  uint8_t       version;        // = 2 (packet version)
  uint8_t       count;          // packet counter. Increases by 1 each packet.
  uint16_t      data[6];        // 10-bit sample (= 0 - 1023) in big endian (Motorola) format.
  uint8_t       switches;       // State of PD5 to PD2, in bits 3 to 0.
```

```
};
*/
/*************************************************************/
#include <compat/deprecated.h>
#include <FlexiTimer2.h>
//http://www.arduino.cc/playground/Main/FlexiTimer2

// All definitions
#define NUMCHANNELS 6
#define HEADERLEN 4
#define PACKETLEN (NUMCHANNELS * 2 + HEADERLEN + 1)
#define SAMPFREQ 256                    // ADC sampling rate 256
#define TIMER2VAL (1024/(SAMPFREQ))     // Set 256Hz sampling frequency
#define LED1  13
#define CAL_SIG 9

// Global constants and variables
volatile unsigned char TXBuf[PACKETLEN];  //The transmission packet
volatile unsigned char TXIndex;           //Next byte to write in the transmission packet.
volatile unsigned char CurrentCh;         //Current channel being sampled.
volatile unsigned char counter = 0;       //Additional divider used to generate CAL_SIG
volatile unsigned int ADC_Value = 0;  //ADC current value

//~~~~~~~~~~
// Functions
//~~~~~~~~~~

/*****************************************************/
/*  Function name: Toggle_LED1                       */
/*  Parameters                                       */
/*    Input   :  No                                  */
/*    Output  :  No                                  */
/*    Action: Switches-over LED1.                    */
/*****************************************************/
void Toggle_LED1(void){

 if((digitalRead(LED1))==HIGH){ digitalWrite(LED1,LOW); }
 else{ digitalWrite(LED1,HIGH); }

}


/*****************************************************/
/*  Function name: toggle_GAL_SIG                    */
/*  Parameters                                       */
/*    Input   :  No                                  */
/*    Output  :  No                                  */
```

```
/*    Action: Switches-over GAL_SIG.                */
/*****************************************************/
void toggle_GAL_SIG(void){

 if(digitalRead(CAL_SIG) == HIGH){ digitalWrite(CAL_SIG, LOW); }
 else{ digitalWrite(CAL_SIG, HIGH); }

}


/*****************************************************/
/*  Function name: setup                          */
/*  Parameters                                    */
/*    Input   :  No                               */
/*    Output  :  No                               */
/*    Action: Initializes all peripherals         */
/*****************************************************/
void setup() {

 noInterrupts();  // Disable all interrupts before initialization

 // LED1
 pinMode(LED1, OUTPUT);  //Setup LED1 direction
 digitalWrite(LED1,LOW); //Setup LED1 state
 pinMode(CAL_SIG, OUTPUT);

 //Write packet header and footer
 TXBuf[0] = 0xa5;    //Sync 0
 TXBuf[1] = 0x5a;    //Sync 1
 TXBuf[2] = 2;       //Protocol version
 TXBuf[3] = 0;       //Packet counter
 TXBuf[4] = 0x02;    //CH1 High Byte
 TXBuf[5] = 0x00;    //CH1 Low Byte
 TXBuf[6] = 0x02;    //CH2 High Byte
 TXBuf[7] = 0x00;    //CH2 Low Byte
 TXBuf[8] = 0x02;    //CH3 High Byte
 TXBuf[9] = 0x00;    //CH3 Low Byte
 TXBuf[10] = 0x02;   //CH4 High Byte
 TXBuf[11] = 0x00;   //CH4 Low Byte
 TXBuf[12] = 0x02;   //CH5 High Byte
 TXBuf[13] = 0x00;   //CH5 Low Byte
 TXBuf[14] = 0x02;   //CH6 High Byte
 TXBuf[15] = 0x00;   //CH6 Low Byte
 TXBuf[2 * NUMCHANNELS + HEADERLEN] =  0x01;        // Switches state

 // Timer2
```

```
        // Timer2 is used to setup the analag channels sampling frequency and packet
update.
        // Whenever interrupt occures, the current read packet is sent to the PC
        // In addition the CAL_SIG is generated as well, so Timer1 is not required in this
case!
        FlexiTimer2::set(TIMER2VAL, Timer2_Overflow_ISR);
        FlexiTimer2::start();

        // Serial Port
        Serial.begin(57600);
        //Set speed to 57600 bps

        // MCU sleep mode = idle.
        //outb(MCUCR,(inp(MCUCR) | (1<<SE)) & (~(1<<SM0) | ~(1<<SM1) | ~(1<<SM2)));

        interrupts();  // Enable all interrupts after initialization has been completed
       }

       /*****************************************************/
       /*  Function name: Timer2_Overflow_ISR              */
       /*  Parameters                                       */
       /*    Input   :  No                                  */
       /*    Output  :  No                                 */
       /*    Action: Determines ADC sampling frequency.    */
       /*****************************************************/
       void Timer2_Overflow_ISR()
       {
         // Toggle LED1 with ADC sampling frequency /2
         Toggle_LED1();

         //Read the 6 ADC inputs and store current values in Packet
         for(CurrentCh=0;CurrentCh<6;CurrentCh++){
           ADC_Value = analogRead(CurrentCh);
           TXBuf[((2*CurrentCh) + HEADERLEN)] = ((unsigned char)((ADC_Value & 0xFF00)
>> 8)); // Write High Byte
           TXBuf[((2*CurrentCh) + HEADERLEN + 1)] = ((unsigned char)(ADC_Value &
0x00FF));       // Write Low Byte
         }

         // Send Packet
         for(TXIndex=0;TXIndex<17;TXIndex++){
           Serial.write(TXBuf[TXIndex]);
         }

         // Increment the packet counter
         TXBuf[3]++;
```

```
  // Generate the CAL_SIGnal
  counter++;                // increment the devider counter
  if(counter == 12){      // 250/12/2 = 10.4Hz ->Toggle frequency
    counter = 0;
    toggle_GAL_SIG(); // Generate CAL signal with frequ ~10Hz
  }
}


/*****************************************************/
/*  Function name: loop                              */
/*  Parameters                                       */
/*    Input   :  No                                  */
/*    Output  :  No                                  */
/*    Action: Puts MCU into sleep mode.              */
/*****************************************************/
void loop() {

 __asm__ __volatile__ ("sleep");

}
```