Universidade Católica de Pelotas
Centro politécnico
Curso de Engenharia Eletrônica
Disciplina de Instrumentação Eletrônica
Semestre 2013/1

Wattímetro

Desenvolvido por
Pedro Carvalho Santos Martinez

Relatório Final da Disciplina de
Instrumentação Eletrônica
http://olaria.ucpel.tche.br/ie2013

Pelotas, Julho de 2013

Índice de Figuras

Sumário

1 Introdução

O desenvolvimento deste trabalho visa pôr em prática os conceitos expostos e debatidos na disciplina de instrumentação. Esta componente do currículo do curso de engenharia eletrônica. Assim consequentemente ao desenvolvimento visa a aprovação desta disciplina será em parte obtida ao alcançar os objetivos, compostos de projeto prático, relatório escrito, e apresentação.

Este trabalho visa a implementação física de dois sensores: corrente e tensão. Estes dois sensores em conjunto servem não somente para medição destas duas grandezas físicas, mas juntos servem para determinar também a potência em watts de determinada carga.

A principal motivação para este trabalho ser desenvolvido utilizando estes dois tipos de sensores, visa a utilização intelectual e prática dos conceitos a seguir apresentados, no trabalho final de graduação ou TCC que por mim será implementado, neste caso em especifico um medidor eletrônico de consumo energético inteligente, ou simplesmente *smart meter* (figura 1), como são chamados estes tipos de medidores residências de energia.

Figura 1 - Smart Meter (www.spiral-group.co.uk).

Para este fim é de suma importância a obtenção de resultados satisfatórios no âmbito da precisão, em implementações com corrente alternada.

Da seguinte forma os objetivos fazem parte da motivação:

- Visando resultados confiáveis na medição das grandezas em questão;
- O desenvolvimento de uma interface gráfica (Software) para interação;
- A criação de um hardware display, para acompanhamento das medidas diretamente no medidor;
- E se possível, aferição dos resultados obtidos na implementação.

Nenhuma possibilidade de aprimoramento ou funções extras estão limitadas aos objetivos a cima citados, sempre buscando o aperfeiçoamento através de sugestões e observações não só professores, mas também colegas de curso.

Para a implementação deste projeto será utilizado um Arduino, que é uma plataforma de prototipagem livre. E para interface visual um software desenvolvido em linguagem C++.

2 Fundamentação teórica

Para o desenvolvimento deste projeto foram de grande importância a revisão de alguns conceitos teóricos, para uma melhor compreensão de funcionamento de sensores e componentes utilizados no projeto.

2.1 Efeito Hall

Os sensores de efeito Hall são o que existem de mais moderno na medição de corrente, já que agregam grandes vantagens de sua "concorrência", pois possuem isolação galvânica como o transformador de corrente e invólucro reduzido como os resistores Shunt.

Sua fundamentação inicia-se no ano de 1819 quando o físico dinamarquês Oersted fez um experimento simples, onde observou que ao aproximar-se uma bússola de um condutor percorrido por corrente, a mesma mudava de direção. Após em 1833, Lenz elaborou sua lei, dizendo que: "O sentido da corrente é o oposto da variação do campo magnético que lhe deu origem". Em cima disto em 1879 o pesquisador Edwin H. Hall descobriu um fenômeno, o qual seria chamado de efeito Hall. Considerando um condutor retangular, percorrido por uma corrente I, e aplicando um campo de indução magnética perpendicular com a densidade de corrente, observa-se que os elétrons passantes sofrem uma força de arrasto, causando uma diferença de potencial entre o lado do condutor que sofre o campo de indução e lado que não sofre. Tal tensão é chamada de tensão Hall (SADIKU, 2013).

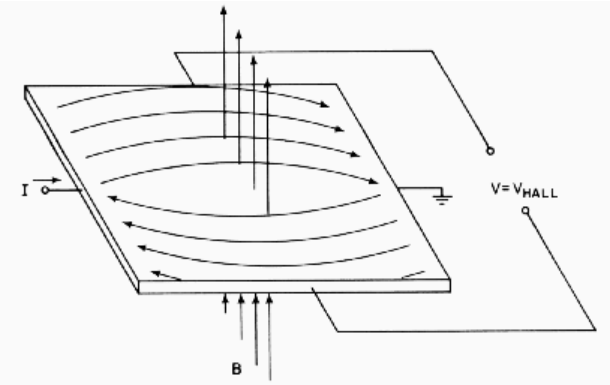


Figura 2 - Efeito Hall.

A medição de corrente elétrica por efeito Hall basicamente consiste em um material ferromagnético e uma placa semicondutora de forma que o campo magnético gerado pela corrente que percorre o material ferromagnético em questão, e fica posicionado perpendicularmente a esta placa e isolados eletricamente entre si. Esta placa é percorrida por uma corrente constante e conhecida. A corrente primária a ser mensurada gera então um campo magnético que causa uma força de arrasto nos elétrons conduzidos na placa

semicondutora, gerando assim uma diferença de potencial proporcional a corrente a primária (BALBINOT, 2010).

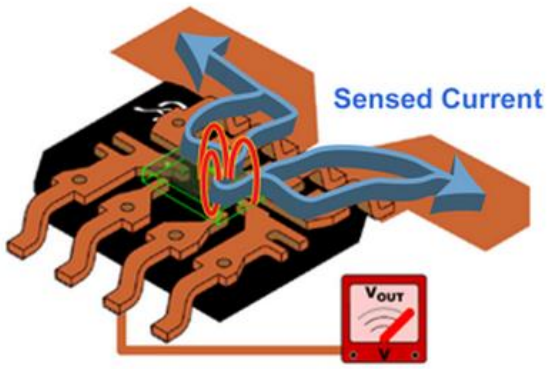


Figura 3 - Sensor de efeito Hall com encapsulamento em circuito integrado (Allegro Micro, 2013).

2.2 Divisor de tensão

O divisor de tensão (figura 3) nada mais é que um método utilizado para criar uma tensão de saída (Vout) que é proporcional a uma tensão de entrada (Vin). Que também pode servir como um sinal atenuador de baixas frequência.

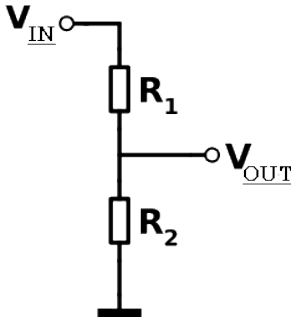


Figura 4 - Esquema de um divisor de tensão.

A tensão de saída pode ser determinada pela equação 1.

$$V_{\text{out}} = \frac{R_2}{R_1 + R_2} \cdot V_{\text{in}} \tag{1}$$

Onde R1 e R2 são resistores.

2.3 Sensor de corrente

O sensor base para o desenvolvimento deste projeto é o ACS712 da fabricante Allegro®. Se destaca por ser uma solução econômica, compacta e

precisa para o sensoriamento com aplicações industriais, comerciais e em sistemas de comunicação, tanto de corrente contínua como alternada. O encapsulamento do dispositivo permite uma fácil implementação pelo usuário. Suas aplicações típicas incluem controle de motores, detecção e controle de cargas, chaveamento de sistemas de energia e proteção contra sobre corrente.

A corrente aplicada flui através do “caminho” de condução e gera um campo magnético que o CI HALL transforma numa tensão proporcional na saída a esta corrente percorrida.

A saída do componente sofre um acréscimo de tensão quando a corrente aumenta quando a corrente flui no sentido dos pinos 1 e 2, para os pinos 3 e 4. A resistência interna do dispositivo, com isto a uma dissipação de potência muito baixa. A espessura da trilha de condução da corrente primária permite uma sobre corrente de até 5x o valor nominal, e também possui um isolamento elétrico entre os pinos percorridos pela corrente em questão e a saída do dispositivo (sensor Hall – pinos 5 a 8).

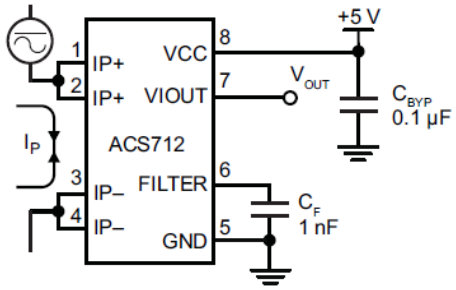


Figura 5 - Aplicação típica do ACS712 (ACS712-DS, 2012).

Algumas características do dispositivo:

- Baixo nível de ruído do sinal analógico.
- A largura de banda do dispositivo é configurado através do pino (6) FILTER.
- 5us de tempo de resposta da saída.
- Largura de banda de 80 kHz.
- Erro total de saída de 1,5% (op. Em 25ºC).
- Resistência interna de 1.2mΩ.
- 2.1kVrms de isolação mínima dos pinos 1-4 para 5-8.
- Tensão de operação de 5.0v.
- 66 a 185mV/A de sensibilidade na saída.
- Tensão de saída proporcionais para operação em DC ou AC.
- Histerese magnética quase nula na saída.

Para o desenvolvimento deste trabalho foi utilizado o modelo ACS712-30A, que admite correntes de aproximadamente 30 ampères. Existem outros modelos deste dispositivo como o ACS712-05A e o 20A, que admitem respectivamente correntes de até 5 e 20 ampères como seus limites de operação. Suas escalas respectivamente possuem uma variação diferente, com uma precisão maior do 05A, se comparado com o 20A, e do 20A se comparado com 30A.

Tabela 1 - Tabela com limites de operação e sensibilidade de escala do ACS712 (ACS712 – DS, 2012).

Part Number	Packing*	T_A (°C)	Optimized Range, I_P (A)	Sensitivity, Sens (Typ) (mV/A)
ACS712ELCTR-05B-T	Tape and reel, 3000 pieces/reel	–40 to 85	±5	185
ACS712ELCTR-20A-T	Tape and reel, 3000 pieces/reel	–40 to 85	±20	100
ACS712ELCTR-30A-T	Tape and reel, 3000 pieces/reel	–40 to 85	±30	66

2.4 Sensor de tensão

O sensor de tensão é composta por duas etapas, a primeira o rebaixamento da tensão da rede, utilizando um divisor de tensão, e posterior a isso uma etapa de retificação desta tensão rebaixada. Esta etapa é de simples entendimento e não utiliza nenhum componente especial (CI).

A entrada do conversor ADC (analógico-digital converter) no Arduino é limitada a 5v, portanto por questões de segurança foi trabalhado com tensões bem abaixo do limite na casa dos 1V na saída do divisor de tensão. A equação 2 mostra o método para calcular o fator de redução dos 220Vac da tensão da rede para uma tensão de 1V.

$$Relação = \frac{v_{max} \cdot \sqrt{2}}{1v} = \frac{220v \cdot \sqrt{2}}{1v} = 311{,}25 \qquad (2)$$

O fator de redução de acordo com a equação 1 é de aproximadamente 622:1. Parte desta redução obtemos utilizando um transformador rebaixador de tensão, com relação de 10:1, o que resulta 22Vac na sua saída.

Figura 6- Transformador rebaixador semelhante (www.yhdc.com).

Após isto é utilizado um divisor simples de tensão, com dois resistores com a relação de redução complementar faltante, 22V para 1V, mas não podemos esquecer que o circuito retificador faz parte deste cálculo. Assim devemos considerar as quedas dos diodos que componham o retificador.

O grande problema deste método é a falta de isolação da tensão da rede, com a nossa parte lógica, composta de microcontrolador e computador. Para amenizar este risco, além do transformador também foi adicionado um diodo zener de 3,3V em paralelo com a carga, para servir como uma espécie de válvula de escape, caso aconteçam sobre tensões este dispositivo não permitirá que a tensão da carga ultrapasse este valor por ele determinado.

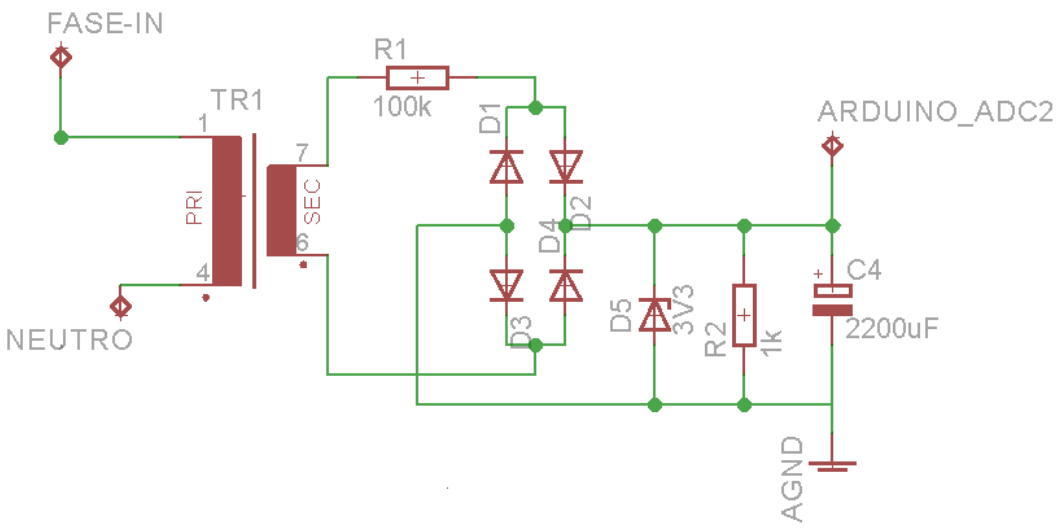


Figura 7 - Circuito medidor de tensão.

Na figura 6 podemos observar o diagrama esquemático do sensor de tensão que utilizado para o desenvolvimento deste trabalho. Composto por transformador (TR1), divisor de tensão (R1 e R2), retificador (D1, D2, D3 e D4), proteção (D5) e capacitor para uma maior linearização e eliminar espúrios de oscilação na tensão a ser medida.

2.5 Potência

Para este projeto foi levado em conta apenas a potência ativa, que é toda a energia que resulta em trabalho de determinado equipamento.

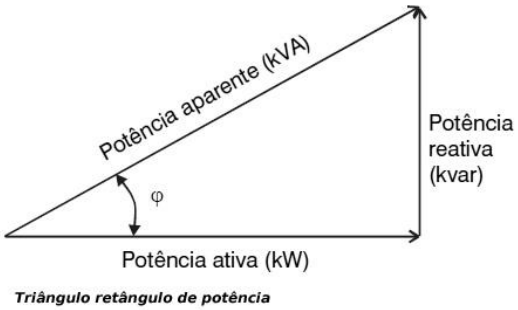


Figura 8 - Triângulo de potências.

É notado na figura 8 que a potência ativa não é influenciada pela defasagem entre as fases de corrente e tensão. E é medida em watts.

$$P_{ativa} = \frac{1}{T}\int_0^T V(t).I(t).dt \tag{3}$$

Leva em conta a tensão e corrente média no tempo e é calculada de acordo com a equação 3.

3 Arduino

Trata-se de uma plataforma de prototipagem de hardware livre baseada no microcontrolador Atmega328. A linguagem usada na programação tem origem no *Wiring*, essencialment C/C++.

Tem como diferencial o desenvolvimento e aperfeiçoamento de software e hardware por uma comunidade que divulga seus códigos de forma livre, num sistema *open-source*.

O Arduino Duemilanove (2009, em italiano) tem 14 pinos de entrada ou saída dos quais 6 podem ser usados como saída PWM. Há 6 entradas analógicas, 1 cristal de 16 MHz, conexão USB, a tensão operacional é de 5 V. Além, há um conector ICSP e 1 botão de reset.

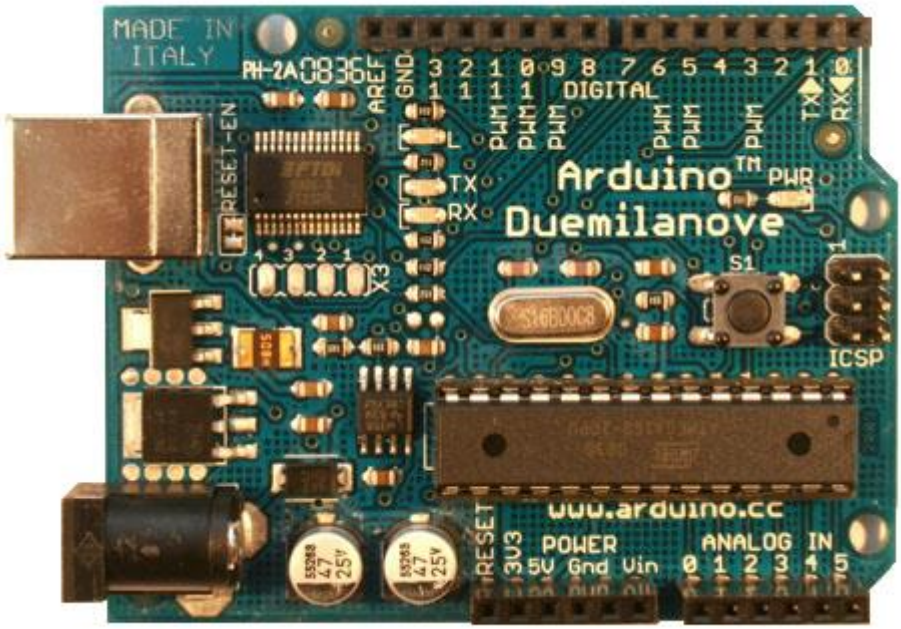


Figura 9 - Arduino Duemilanove (www.arduino.cc).

A placa pode operar com uma alimentação externa de 6 a 20 V, sendo o recomendado tensão entre 7 e 12 V. Possui um proteção extra contra sobrecorrente (acima de 500 mA). Liga ao ser conectado ao computador, através de um adaptador para a rede, ou ainda com o uso de bateria. Possui memória FLASH de 32 KB, SRAM de 2 KB e EEPROM de 1 KB. A corrente contínua por pino de entrada e saída é de 40 mA, tendo portanto, uma operação de *low power*.

Usando as funções *pinMode (), digitalWrite () e digitalRead ()*, cada um dos 14 pinos podem ser usados como entrada ou saída. Os pinos possuem resistores *pull-up* interno. Outros pinos com funções especializadas: Serial 0-RX e 1-TX, usados para receber e transmitir, através da comunicação serial, dados TTL. O PWM assumem os pinos 3, 5, 6, 9, 10 e 11, que fornecem uma saída analógica de 8-bits, cuja função correspondente é *analogWrite ().* O pino 13 está conectado ao LED.

A placa possui 6 entradas analógicas que estão ligadas a conversores analógico-digitais de 10-bits (para valores transformados em valores entre 0 e 1023). A referência de tensão, que pode ser programada, retorna tensão entre 0 e 5 V.

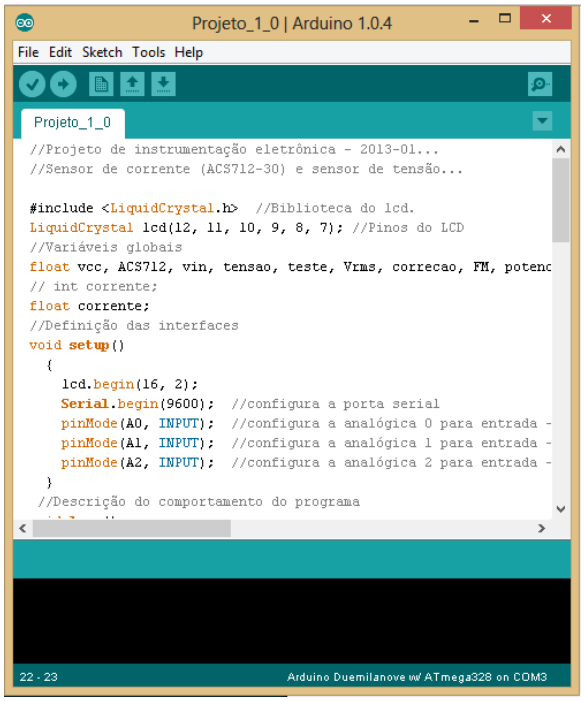


Figura 10 - Arduino IDE 1.0.4 (www.Arduino.cc)

O Arduino IDE (figura 10) é uma aplicação multiplataforma escrita em Java derivada dos projetos Processing e Wiring. É esquematizado para introduzir a programação a estudantes e a pessoas não familiarizadas com o desenvolvimento de software. Inclui um editor de código com recursos de realce de sintaxe, parênteses correspondentes e identação automática, sendo capaz de compilar e carregar programas para a placa com um único clique. Com isso não há a necessidade de editar Makefiles ou rodar programas em ambientes de linha de comando.

4 O instrumento

Podemos dividir o instrumento físico em quatro diferentes partes distintas, que juntos formam o wattímetro:

- Corrente.
- Tensão.
- Display.
- Microcontrolador.

Neste capitulo será exposto as ligações do circuito e um descritivo de seu funcionamento.

É de suma importância ressaltar que o código completo da programação do Arduino pode ser conferido no apêndice B. Desta forma podem ser elucidadas algumas dúvidas em relação a programação.

4.1 Corrente

Foi utilizado um kit (figura 11) para o sensor de corrente. Este kit é composto do sensor de efeito hall ACS712, terminais, capacitores e resistor SMD ("superficial monting device") para facilitar na prototipação e adicionar praticidade ao projeto.

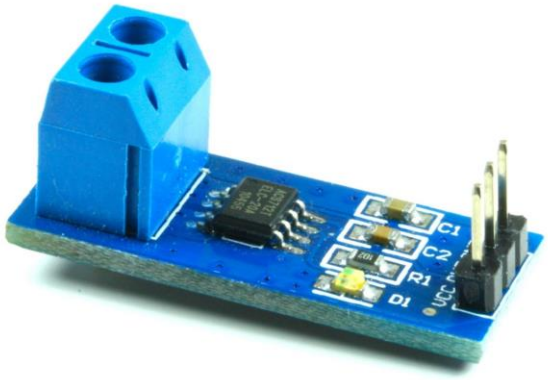


Figura 11 - Kit com sensor de corrente ACS712.

A ligação entre o sensor de corrente ACS712 e o microcontrolador Arduino pode ser observado na figura 12.

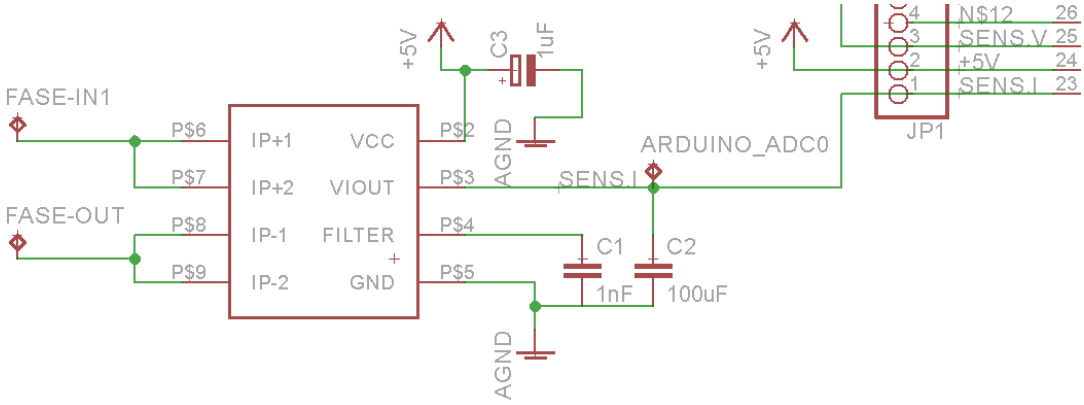


Figura 12 - A ligação entre o ACS712 e o Arduino.

Basicamente o sensor é alimentado por uma tensão de +5Vcc no pino apropriado, e ligado ao GND. A saída deste circuito para o Arduino é definido por Vcc/2 quando a corrente que circula nos terminais do outro extremo do sensor é zero. Quando a corrente percorre o sentido dos pinos 1 e 2 para o 3 e 4 do ACS712 a tensão de saída do sensor varia de acordo com a tabela 1 no sentido de incremento de tensão. Já no sentido contrário a tensão varia com o decremento da mesma.

```
//´Cálculo da corrente

  for(int i = 0; i < 1000; i++) //condição para realizar o cálculo da média da medida.
  {
    corrente = corrente + ((((vcc/2000) - (ACS712/1000))/0.185)*0.83*0.8771)/1000;
    Vrms = (((vin)) * FM)/1000;  //Variável resultado Vrms.
    delay(1);
  }

  if(ACS712 > 2450.00)
  {
    corrente = 0.00;
  }
```

Figura 13 - Lógica do cálculo de corrente e tensão.

O entrada ADC0 (“analogic to digital converter”) do Arduino recebe este sinal de saída do sensor de corrente em níveis de tensão Vcc, e realiza o cálculo da corrente que circula no sensor, em função da relação estabelecida na tabela 1.

Na figura 13 é possível identificar a lógica deste cálculo no Arduino. Primeiramente é armazenada em uma variável nomeada de ‘*ACS712’ para* o valor de tensão do sensor, e após sua normalização de escala é realizada uma média da variável, que foi incrementada a cada mili segundo, ao dividirmos este valor total por mil. Na mesma figura 13 é notada a existência de uma lógica ‘*if’* para eliminar a corrente negativa, já que não é o objetivo deste projeto a medição de correntes em sentido bidirecional, apenas no sentido rede-carga.

4.2 Tensão

A lógica da medição da tensão da rede AC é bem simples e toda ela já foi exposta no capitulo 2.4. Desta forma será exposta a lógica do Arduino para esta medição.

Na figura 13 e dentro da lógica '*for'* é de fácil entendimento o cálculo da tensão da rede. A variável 'Vrms' do código da figura é igual ao valor lido no sensor de tensão (variável ' vin'), que varia na casa de 800mVcc para tensões de rede na casa dos 220Vac, e multiplicado pelo valor da escala de redução ou fator de multiplicação ('FM') por 272,8. Após isto é realizada uma média destes valores lidos, na mesma lógica que foi calculada a média do valor da corrente.

O sensor de tensão utiliza o conversor analógico-digital ADC2 do Arduino (figura 7).

4.3 Display

Foi utilizado um display lcd 20x2 (figura 14), vinte colunas e duas linhas, para uma visualização dos resultados obtidos durante a medição sem a necessidade de utilizar o software para tal. Ficando a critério do usuário qual a interface utilizar, tendo a opção de utilização de ambas.

Figura 14 - Display LCD 20x2.

Em seu esquema de ligação (figura 15) é possível identificar um potenciômetro, para regular o contraste, e a disposição da ligação entre o display e o Arduino.

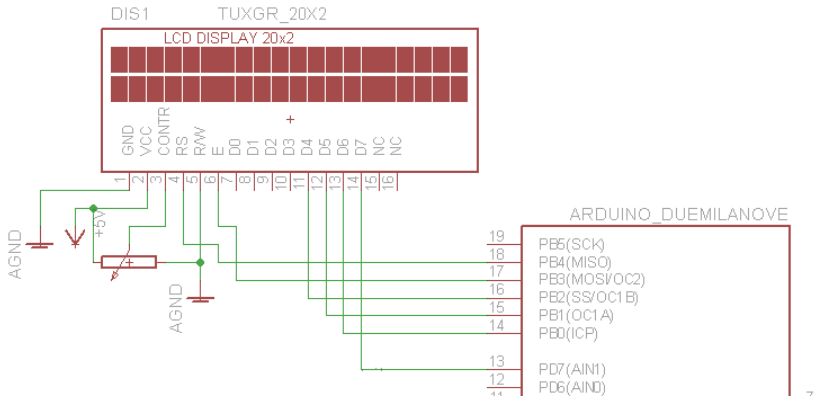


Figura 15 - Display 20x2 e Arduino.

4.4 Microcontrolador

No capítulo 3 já houve uma detalhada explicação sobre o funcionamento e as configurações básicas do Arduino. Assim sendo vamos discutir a disposição das ligações dos sensores no kit.

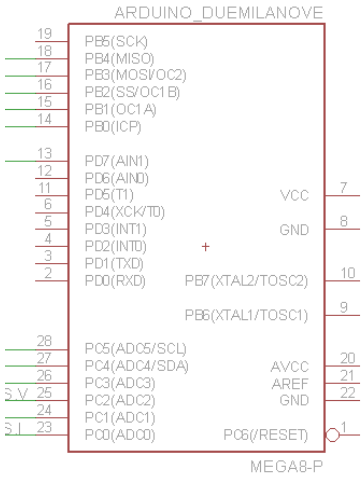


Figura 16 - Esquemático do Arduino.

Para o projeto foram utilizadas três portas ADC, para o monitoramento dos sensores, e nove portas configuradas como saídas digitais para a ligação do display de lcd.

O código desenvolvido para o Arduino em C++ é relativamente pequeno para esta aplicação deste projeto, cerca de 7kb, e foi todo ele desenvolvido e compilado na interface IDE do próprio Arduino (figura 10) e pode ser visualizado em sua integralidade no apêndice b deste relatório.

5 O software

O software (figura 17) para monitoração dos valores e avaliação dos resultados foi todo ele desenvolvido no Microsoft Visual Studio[1]. E seu desenvolvimento visou os seguintes tópicos:

- Comunicação serial[2] entre o Arduino e o programa.
- Visualização dos valores numéricos das medidas dos sensores de corrente e tensão, além do valor da potência calculada em função destes sensores, todos estes em tempo real.
- Visualização de um gráfico de tensão, corrente e potência.
- Opção de gravar um arquivo em .txt com os respectivos valores.

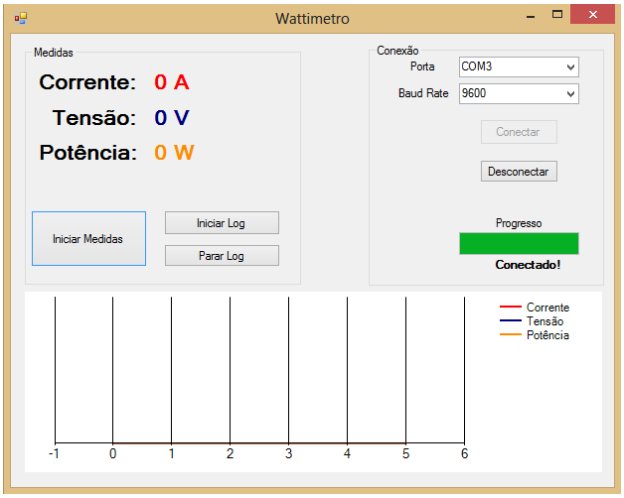


Figura 17 - Software desenvolvido.

5.1 Comunicação serial

A comunicação serial do software desenvolvido no Visual Studio é de simples aplicação. São dois parâmetros a serem escolhidos de acordo com a área destacada em vermelho da figura 18.

[1] O Microsoft Visual Studio é um pacote de programas da Microsoft para desenvolvimento de software especialmente dedicado ao .NET Frameworke às linguagens Visual Basic (VB), C, C++, C# (C Sharp) e J# (J Sharp).

[2] Comunicação serial é o processo de enviar dados um bit de cada vez, sequencialmente, num canal de comunicação ou barramento. É diferente da comunicação paralela, em que todos os bits de cada símbolo são enviados juntos. A comunicação serial é usada em toda comunicação de longo alcance e na maioria das redes de computadores, onde o custo de cabos e as dificuldades de sincronização tornam a comunicação paralela impraticável.

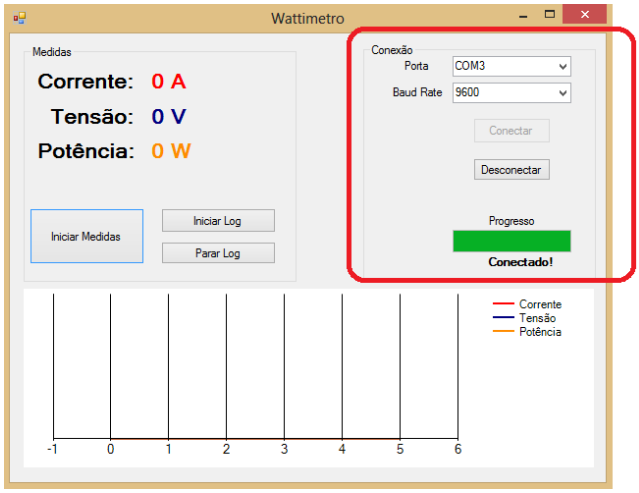


Figura 18 - Comunicação serial em destaque.

No campo porta do código em C++ do programa, automaticamente é disponibilizado para a escolha do usuário apenas as portas COM disponíveis, evitando assim a confusão em operador em qual porta estaria sendo utilizado pelo dispositivo, neste caso o Arduino.

Após a seleção da porta COM, o próximo e último campo a ser determinado para o início da conexão, é a taxa de comunicação ou baud rate. Este parâmetro depende de como o código de programação do Arduino foi desenvolvido, neste caso em especifico foi escolhido no Arduino uma baud rate de 9600bps, portanto deve ser escolhido no software do Visual Studi a mesma velocidade de comunicação do microcontrolador, 9600bps.

Ao clicar no botão conectar inicia-se o processo de comunicação entre o Arduino e o software. Logo abaixo aparecerá a barra de progresso e a mensagem de “Conectado!” se ocorrido sucesso na conexão, ou “Erro!” no insucesso.

5.2 Visualização dos valores medidos

A visualização dos três valores, ocorre na área em destaque da figura18. O processo de início na medida dos valores se dá ao clicar no botão “Iniciar Medidas”.

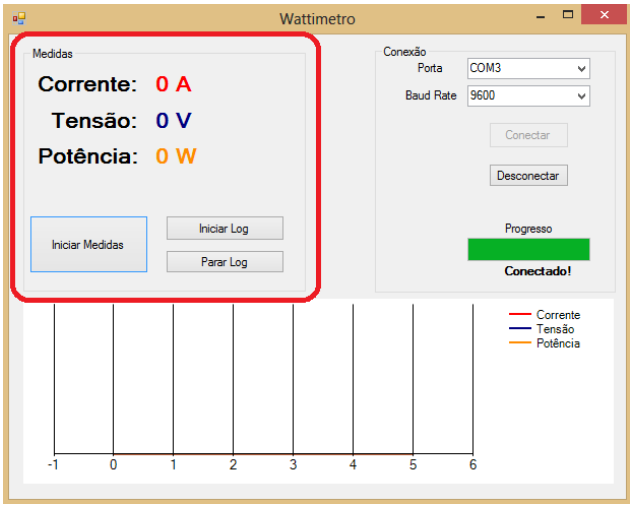


Figura 19 - Valores medidos em destaque.

O software foi descrito para a utilização de uma lógica ‘switch case’ para a identificação do byte de cada valor. Como são medidas e portanto bytes diferentes a serem enviados do Arduino para o computador, foi criada uma lógica de identificação de cada byte adicionando uma letra ou ‘string’, assim o campo corrente por exemplo foi adicionado a letra C ao valor de corrente lido. Posterior a isto o campo corrente da figura 19 recebe o byte composto pela letra C mais o valor da corrente. Esta lógica replicada para tensão e potência e pode ser visualizada com mais detalhes no apêndice c, que traz o código completo desenvolvido no Visual Studio.

5.3 Gráfico

Para o gráfico foi utilizada a função ‘chart’ do Visual Studio, e seu funcionamento é bem simples. No eixo X do gráfico fica a variável de tempo em segundos e no eixo Y o valor absoluto das medias de tensão, corrente e potência.

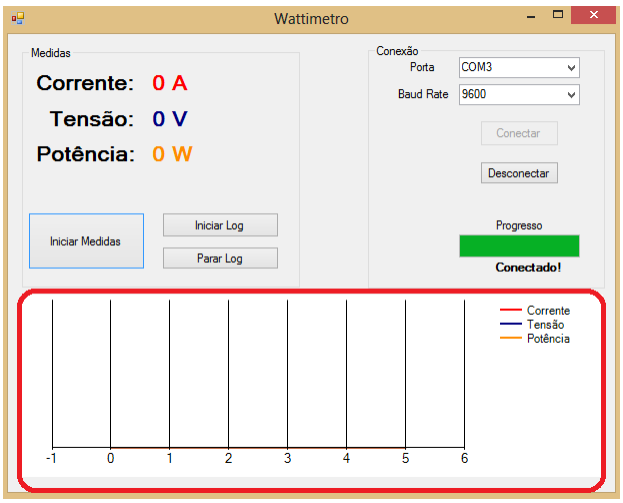


Figura 20 - Campo gráfico em destaque.

Cada variável possui uma cor diferente como pode ser observado na figura 20, sendo a corrente em vermelho, tensão em azul e a potência em laranja.

O processo inicia-se automaticamente ao se iniciarem as medidas, ao breve click do botão “Iniciar Medidas”.

5.4 Histórico em .txt

O arquivo com o histórico dos valores medidos é gerado em uma extensão .txt. O local de gravação foi determinado no código do apêndice c, neste caso uma pasta foi criada dentro da raiz onde está salvo o programa. Ao clicarmos no botão “Iniciar log”, em destaque na figura 19, inicia-se a geração do log de valores. Os valores serão salvos de dez em dez segundos, e ao encerrarmos o programa ou clicar no botão “Parar log”.

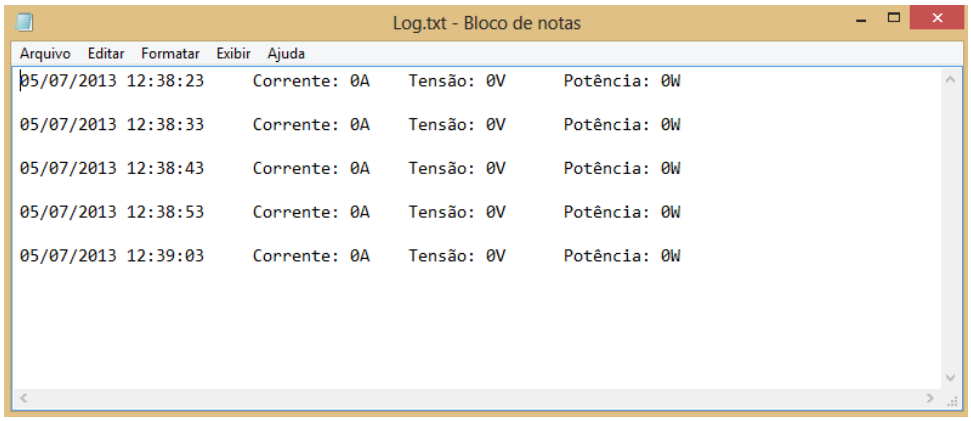


Figura 21 - Log dos valores medidos em .txt.

Quando o processo de gravação se encerra basta acessar o local onde foi gravado o arquivo e abrir o mesmo com bloco de notas ou um outro programa compatível com a extensão em .txt.

6 Comportamento e resultados

Após a etapa de revisão teórica e um estudo sobre o comportamento dos sensores, foram desenvolvidos na ordem, a montagem do wattímetro no protoboard (figura 22), o código do microcontrolador (Arduino) e por último o software em Visual Studio.

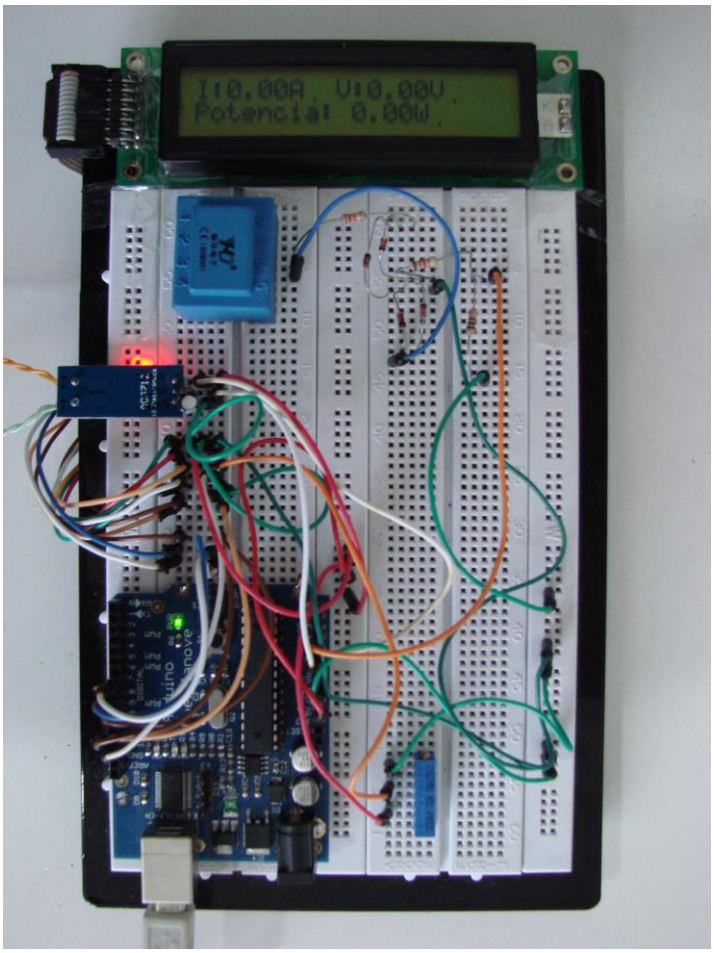


Figura 22 - Imagem do wattimetro.

Testes e aprimoramentos foram realizados, tanto no software como na concepção dos sensores, ao longo das aulas da disciplina de instrumentação eletrônica.

A validação dos resultados se deu pelo teste em laboratório, que consistiu em ligar uma carga na tensão CA da rede de energia, neste caso 220Vac, e a colocação de um amperímetro em série com o sensor de corrente ACS712, e um multímetro colocado em paralelo com a carga para medir a tensão AC.

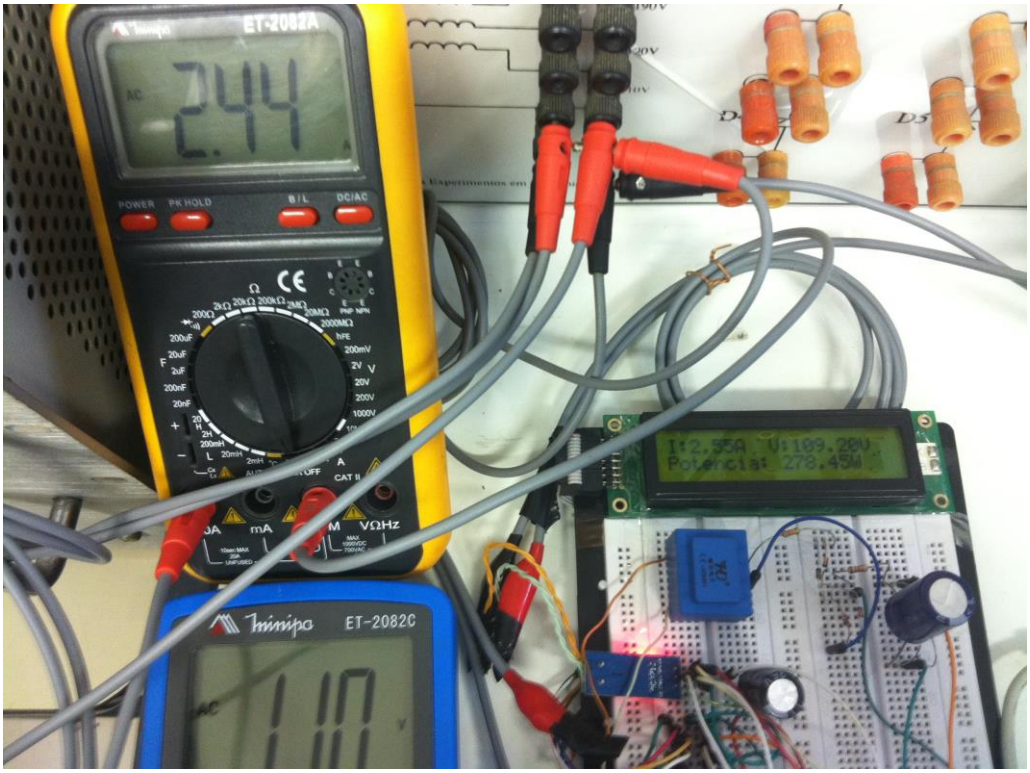


Figura 23 - Testes do wattímetro no laboratório.

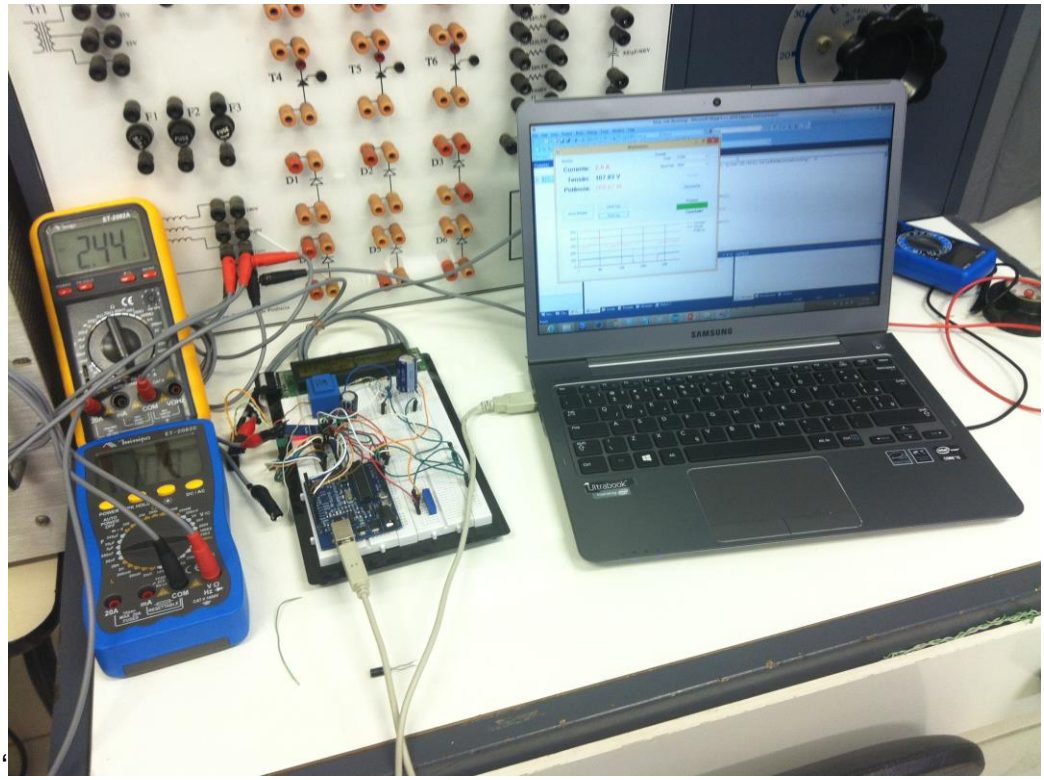


Figura 24 - Teste do software.

Os resultados foram muito satisfatórios com um desvio de tensão em torno de ± 0,5Vac, e para a corrente em torno de ± 0,03ª.

Dificuldades foram encontradas em vários pontos do projeto, mas pode-se destacar principalmente no desenvolvimento do software a etapa mais

dificultosa. Conversão de variáveis, comunicação serial entre o software e o Arduino foram as principais dificuldades, além da solução de um problema causado pelo acumulo no buffer do software, que causava um atraso acumulativo nas medições.

Apesar das dificuldades e dos problemas encontrados durante as etapas do projeto, foi um grande aprendizado para mim ao superar estas dificuldades, que justamente é um dos objetivos dos trabalhos acadêmicos.

Objetivos alcançados e aprendizados acumulados nesta que foi para mim a mais prazerosa disciplina do semestre.

7 Referências Bibliográficas

1. SADIKU, MATHEW N. O. "Elementos de Eletromagnetismo". 3ª Edição. Editora Bookman. Porto Alegre, 2004.

2. BALBINOT, A.; BRUSAMARELLO, V. J. “Instrumentação e Fundamentos de Medidas”. 2ª Edição. Vol. 1. Editora LTC. 2010.

3. Allegro Micro. “*Current IC Sensors*”. Disponível em: http://www.allegromicro.com/en/Products/Current-Sensor-ICs/Zero-To-Fifty-Amp-Integrated-Conductor-Sensor-ICs.aspx. Acessado em 07 de junho de 2013.

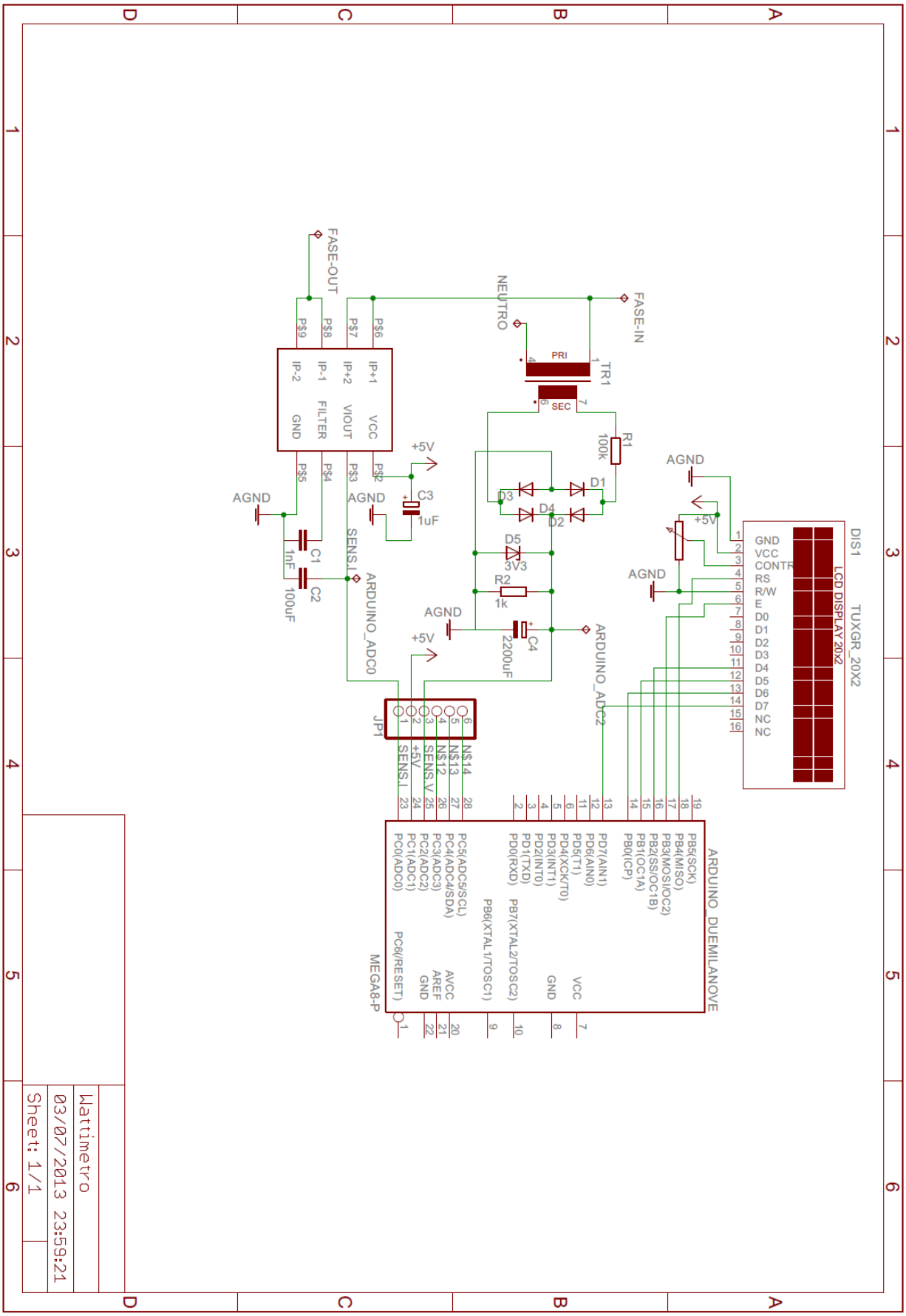
4. ACS712. Data sheet: Fully Integrated, Hall Effect-Based Linear Current Sensor IC with 2.1 kVRMS Isolation and a Low-Resistance Current Conductor. Rev 15, 2012. Disponível em: http://www.allegromicro.com/~/media/Files/Datasheets/ACS712-Datasheet.ashx

5. PE2409-1 – 0.5Va. Data sheet. Disponível em: http://www.yhdc.com/english/productshow.asp?id=53

6. Site do projeto Arduino – Disponível em: 05/07/2013 http://www.arduino.cc

8 Apêndices

8.1 Apêndice A - Esquemático



8.2 Apêndice B - Arduino

```
//----------------------UCPel---------------------//
 //Projeto de instrumentação eletrônica - 2013-01...//
 //Sensor de corrente (ACS712-30) e sensor de tensão//
 //Pedro Martinez - Julho de 2013.....//

  //Variáveis Globais e declaração de bibliotecas
 #include <LiquidCrystal.h>  //Biblioteca do lcd.
 LiquidCrystal lcd(12, 11, 10, 9, 8, 7); //Pinos do LCD
 //Variáveis globais
 float vcc, ACS712, vin, tensao, teste, Vrms, correcao, FM, potencia;
 // int corrente;
 float corrente;
 //Definição das interfaces
 void setup()
  {
    lcd.begin(16, 2);    //descrição das dimensões do display LCD.
    Serial.begin(9600);  //configura da taxa de transmissão da porta serial.
    pinMode(A0, INPUT);  //configura a analógica 0 para entrada - ACS712 -> output.
    pinMode(A1, INPUT);  //configura a analógica 1 para entrada - Vcc (variável utlizada para o
cálculo da corrente).
    pinMode(A2, INPUT);  //configura a analógica 2 para entrada - Sensor de tensão.
  }

  //Descrição do comportamento do programa
 void loop()
  {
    vcc = analogRead(A1);  //iguala variável ao valor lido na analógica 1.
    ACS712 = analogRead(A0);  //iguala variável ao valor lido na analógica 0.
    vin = analogRead(A2);  //iguala variável ao valor lido na analógica 2.
    vcc = map(analogRead(A1), 0, 1023, 0, 5000);  //Determina o fundo de escala da variável vcc vcc->
ADC com 10bits (0 a 1023) -> 4,88 de variação por bit.
    ACS712 = map(analogRead(A0), 0, 1023, 0, 5000);  //Determina o fundo de escala da variável vcc2-
> ADC com 10bits (0 a 1023) -> 4,88 de variação por bit.
    vin = map(vin, 0, 1023, 0, 5000);  //Determina o fundo de escola da medida vin.
    corrente = 0; //Possivel ateração do valor para correção da medida da variável corrente.
    //Vrms = 0;  //Possivel ateração do valor para correção da medida da variável Vrms.
    FM = 272.62;  //Fator de multiplicação para a medida de Vrms, 220v(rede) -> 14v(trafo) ->
11,9v(retificador) -> 11,48v(res. 27k) -> 0,412v(res. 1k)(saida = vin).

    //´Cálculo da corrente, tensão e potência.
    for(int i = 0; i < 1000; i++) //condição para realizar o cálculo da média da medida.
    {
      corrente = corrente + ((((vcc/2000) - (ACS712/1000))/0.185)*0.83*0.8771)/1000;  //lógica para o
cálculo da corrente ACS712-05.
      Vrms = (((vin)) * FM)/1000;  //Variável para o calculo do resultado da tensão (Vrms).
      potencia = corrente*Vrms; //calculo da potência - P=IxV.
      delay(1);
    }

    //condição para evitar correntes negativas quando não houver corrente na carga.
    if(ACS712 > 2450.00)
    {
      corrente = 0.00;//valor zera corrente.
    }
```

```
  //envio dos bytes resultados para a serial, e display lcd.
   Serial.print("C");//string de identificação do byte corrente.
   Serial.print(corrente, 2);//variável corrente com duas casas decimais depois da virgula.
   Serial.write(10);//equivalente decimal para NL (new line).
   Serial.print("T");//string de identificação do byte potência.
   Serial.print(Vrms, 2);//variável tensão com duas casas decimais depois da virgula.
   Serial.write(10);//equivalente decimal para NL (new line).
   Serial.print("P");//string de identificação do byte tensão.
   Serial.print(potencia, 2);//variável potência com duas casas decimais depois da virgula.
   Serial.write(10);//equivalente decimal para NL (new line).
   lcd.setCursor(0, 0);//posição da escrita no lcd, coluna zero e linha zero.
   lcd.print("I:");//string inicial da escrita corrente.
   lcd.print(corrente);//valora da corrente medida escrita após a string "I:".
   lcd.println("A");//simbolo da corrente escrito depois do valor.
   lcd.print("V:");//string de tensão.
   lcd.print(Vrms);//valor de tensão.
   lcd.println("V");//simbolo da tensão.
   lcd.display();//encerra o setCursor(0,0).
   lcd.setCursor(0, 1);//posição da escrita no lcd, coluna zero, linha um.
   lcd.print("Potencia: ");//string de potência.
   lcd.print(potencia);//valor da potência.
   lcd.println("W");//simbolo da potência.
   lcd.display();//encerra o setCursor(0,1).
  }
```

8.3 Apêndice C – Visual Studio

```
#pragma once

namespace Amp_volt {

	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;
	using namespace System::IO::Ports;
	using namespace System::IO;

	/// <summary>
	/// Summary for Form1
	/// </summary>
	public ref class Form1 : public System::Windows::Forms::Form
	{
	public:

		static double corrente = 0;
	    static double tensao = 0;
		static int time = 0;
		static double potencia = 0;
		static String^ filename = "D://Instrumentação/Projeto/Log_Amp_Volt/Log.txt";
		static StreamWriter^ sw = gcnew StreamWriter(filename);
		static int g;
		static int i;
	public:


		Form1(void)
		{
			InitializeComponent();
			findPorts();
```

```
		}

	protected:
		/// <summary>
		/// Clean up any resources being used.
		/// </summary>
		~Form1()
		{
			if (components)
			{
				delete components;
			}
		}
	private: System::Windows::Forms::ComboBox^  comboBox1;
	protected:
	private: System::Windows::Forms::ComboBox^  comboBox2;
	private: System::Windows::Forms::Label^  label1;
	private: System::Windows::Forms::Label^  label2;
	private: System::Windows::Forms::Button^  button1;
	private: System::Windows::Forms::Button^  button2;
	private: System::Windows::Forms::Label^  label3;
	private: System::Windows::Forms::ProgressBar^  progressBar1;
	private: System::Windows::Forms::Label^  label4;
	private: System::Windows::Forms::Label^  label5;
	private: System::Windows::Forms::Label^  label6;
	private: System::Windows::Forms::Label^  label7;
	private: System::Windows::Forms::Label^  label8;
	private: System::Windows::Forms::Label^  label9;
	private: System::Windows::Forms::Label^  label10;
private: System::Windows::Forms::DataVisualization::Charting::Chart^  chart1;
	private: System::Windows::Forms::Button^  button3;
	private: System::Windows::Forms::Button^  button4;
private: System::Windows::Forms::Button^  button5;
	private: System::IO::Ports::SerialPort^  serialPort1;
	private: System::Windows::Forms::Timer^  timer1;
	private: System::Windows::Forms::GroupBox^  groupBox1;
	private: System::Windows::Forms::GroupBox^  groupBox2;
	private: System::ComponentModel::IContainer^  components;

	private:
		/// <summary>
		/// Required designer variable.
		/// </summary>

#pragma region Windows Form Designer generated code
		/// <summary>
		/// Required method for Designer support - do not modify
		/// the contents of this method with the code editor.
		/// </summary>
		void InitializeComponent(void)
		{
			this->components = (gcnew System::ComponentModel::Container());
			System::Windows::Forms::DataVisualization::Charting::ChartArea^
chartArea1 = (gcnew System::Windows::Forms::DataVisualization::Charting::ChartArea());
			System::Windows::Forms::DataVisualization::Charting::Legend^  legend1
= (gcnew System::Windows::Forms::DataVisualization::Charting::Legend());
			System::Windows::Forms::DataVisualization::Charting::Series^  series1
= (gcnew System::Windows::Forms::DataVisualization::Charting::Series());
			System::Windows::Forms::DataVisualization::Charting::Series^  series2
= (gcnew System::Windows::Forms::DataVisualization::Charting::Series());
			System::Windows::Forms::DataVisualization::Charting::Series^  series3
= (gcnew System::Windows::Forms::DataVisualization::Charting::Series());
			this->comboBox1 = (gcnew System::Windows::Forms::ComboBox());
			this->comboBox2 = (gcnew System::Windows::Forms::ComboBox());
			this->label1 = (gcnew System::Windows::Forms::Label());
			this->label2 = (gcnew System::Windows::Forms::Label());
			this->button1 = (gcnew System::Windows::Forms::Button());
			this->button2 = (gcnew System::Windows::Forms::Button());
			this->label3 = (gcnew System::Windows::Forms::Label());
			this->progressBar1 = (gcnew System::Windows::Forms::ProgressBar());
			this->label4 = (gcnew System::Windows::Forms::Label());
			this->label5 = (gcnew System::Windows::Forms::Label());
			this->label6 = (gcnew System::Windows::Forms::Label());
			this->label7 = (gcnew System::Windows::Forms::Label());
```

```
                        this->label8 = (gcnew System::Windows::Forms::Label());
                        this->button3 = (gcnew System::Windows::Forms::Button());
                        this->serialPort1 = (gcnew System::IO::Ports::SerialPort(this-
>components));
                        this->timer1 = (gcnew System::Windows::Forms::Timer(this-
>components));
                        this->groupBox1 = (gcnew System::Windows::Forms::GroupBox());
                        this->groupBox2 = (gcnew System::Windows::Forms::GroupBox());
                        this->button5 = (gcnew System::Windows::Forms::Button());
                        this->button4 = (gcnew System::Windows::Forms::Button());
                        this->label9 = (gcnew System::Windows::Forms::Label());
                        this->label10 = (gcnew System::Windows::Forms::Label());
                        this->chart1 = (gcnew
System::Windows::Forms::DataVisualization::Charting::Chart());
                        this->groupBox1->SuspendLayout();
                        this->groupBox2->SuspendLayout();
                        (cli::safe_cast<System::ComponentModel::ISupportInitialize^  >(this-
>chart1))->BeginInit();
                        this->SuspendLayout();
                        //
                        // comboBox1
                        //
                        this->comboBox1->FormattingEnabled = true;
                        this->comboBox1->Location = System::Drawing::Point(91, 13);
                        this->comboBox1->Name = L"comboBox1";
                        this->comboBox1->Size = System::Drawing::Size(121, 21);
                        this->comboBox1->TabIndex = 0;
                        //
                        // comboBox2
                        //
                        this->comboBox2->FormattingEnabled = true;
                        this->comboBox2->Items->AddRange(gcnew cli::array< System::Object^
>(5) {L"9600", L"19200", L"38400", L"57600", L"115200"});
                        this->comboBox2->Location = System::Drawing::Point(91, 40);
                        this->comboBox2->Name = L"comboBox2";
                        this->comboBox2->Size = System::Drawing::Size(121, 21);
                        this->comboBox2->TabIndex = 1;
                        //
                        // label1
                        //
                        this->label1->AutoSize = true;
                        this->label1->Location = System::Drawing::Point(39, 16);
                        this->label1->Name = L"label1";
                        this->label1->Size = System::Drawing::Size(32, 13);
                        this->label1->TabIndex = 2;
                        this->label1->Text = L"Porta";
                        //
                        // label2
                        //
                        this->label2->AutoSize = true;
                        this->label2->Location = System::Drawing::Point(27, 43);
                        this->label2->Name = L"label2";
                        this->label2->Size = System::Drawing::Size(58, 13);
                        this->label2->TabIndex = 3;
                        this->label2->Text = L"Baud Rate";
                        //
                        // button1
                        //
                        this->button1->Location = System::Drawing::Point(112, 76);
                        this->button1->Name = L"button1";
                        this->button1->Size = System::Drawing::Size(79, 26);
                        this->button1->TabIndex = 4;
                        this->button1->Text = L"Conectar";
                        this->button1->UseVisualStyleBackColor = true;
                        this->button1->Click += gcnew System::EventHandler(this,
&Form1::button1_Click);
                        //
                        // button2
                        //
                        this->button2->Location = System::Drawing::Point(112, 117);
                        this->button2->Name = L"button2";
                        this->button2->Size = System::Drawing::Size(79, 23);
                        this->button2->TabIndex = 5;
                        this->button2->Text = L"Desconectar";
```

```
                        this->button2->UseVisualStyleBackColor = true;
                        this->button2->Click += gcnew System::EventHandler(this,
&Form1::button2_Click);
                        //
                        // label3
                        //
                        this->label3->AutoSize = true;
                        this->label3->Location = System::Drawing::Point(125, 174);
                        this->label3->Name = L"label3";
                        this->label3->Size = System::Drawing::Size(54, 13);
                        this->label3->TabIndex = 6;
                        this->label3->Text = L"Progresso";
                        //
                        // progressBar1
                        //
                        this->progressBar1->Location = System::Drawing::Point(91, 190);
                        this->progressBar1->Name = L"progressBar1";
                        this->progressBar1->Size = System::Drawing::Size(121, 23);
                        this->progressBar1->TabIndex = 7;
                        //
                        // label4
                        //
                        this->label4->AutoSize = true;
                        this->label4->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 8.25F, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(0)));
                        this->label4->Location = System::Drawing::Point(125, 216);
                        this->label4->Name = L"label4";
                        this->label4->Size = System::Drawing::Size(11, 13);
                        this->label4->TabIndex = 8;
                        this->label4->Text = L".";
                        //
                        // label5
                        //
                        this->label5->AutoSize = true;
                        this->label5->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 15.75F, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(0)));
                        this->label5->Location = System::Drawing::Point(9, 24);
                        this->label5->Name = L"label5";
                        this->label5->Size = System::Drawing::Size(110, 25);
                        this->label5->TabIndex = 9;
                        this->label5->Text = L"Corrente:";
                        //
                        // label6
                        //
                        this->label6->AutoSize = true;
                        this->label6->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 15.75F, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(0)));
                        this->label6->ForeColor = System::Drawing::Color::Red;
                        this->label6->Location = System::Drawing::Point(125, 24);
                        this->label6->Name = L"label6";
                        this->label6->Size = System::Drawing::Size(19, 25);
                        this->label6->TabIndex = 10;
                        this->label6->Text = L".";
                        //
                        // label7
                        //
                        this->label7->AutoSize = true;
                        this->label7->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 15.75F, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(0)));
                        this->label7->Location = System::Drawing::Point(22, 60);
                        this->label7->Name = L"label7";
                        this->label7->Size = System::Drawing::Size(97, 25);
                        this->label7->TabIndex = 11;
                        this->label7->Text = L"Tensão:";
                        //
                        // label8
                        //
                        this->label8->AutoSize = true;
                        this->label8->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 15.75F, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
```

```
                        static_cast<System::Byte>(0)));
                    this->label8->ForeColor = System::Drawing::Color::Navy;
                    this->label8->Location = System::Drawing::Point(125, 60);
                    this->label8->Name = L"label8";
                    this->label8->Size = System::Drawing::Size(19, 25);
                    this->label8->TabIndex = 12;
                    this->label8->Text = L".";
                    //
                    // button3
                    //
                    this->button3->Location = System::Drawing::Point(6, 166);
                    this->button3->Name = L"button3";
                    this->button3->Size = System::Drawing::Size(117, 57);
                    this->button3->TabIndex = 13;
                    this->button3->Text = L"Iniciar Medidas";
                    this->button3->UseVisualStyleBackColor = true;
                    this->button3->Click += gcnew System::EventHandler(this,
&Form1::button3_Click);
                    //
                    // serialPort1
                    //
                    this->serialPort1->DataReceived += gcnew
System::IO::Ports::SerialDataReceivedEventHandler(this, &Form1::serialPort1_DataReceived);
                    //
                    // timer1
                    //
                    this->timer1->Interval = 1000;
                    this->timer1->Tick += gcnew System::EventHandler(this,
&Form1::timer1_Tick);
                    //
                    // groupBox1
                    //
                    this->groupBox1->Controls->Add(this->label4);
                    this->groupBox1->Controls->Add(this->progressBar1);
                    this->groupBox1->Controls->Add(this->label3);
                    this->groupBox1->Controls->Add(this->button2);
                    this->groupBox1->Controls->Add(this->button1);
                    this->groupBox1->Controls->Add(this->label2);
                    this->groupBox1->Controls->Add(this->label1);
                    this->groupBox1->Controls->Add(this->comboBox2);
                    this->groupBox1->Controls->Add(this->comboBox1);
                    this->groupBox1->Location = System::Drawing::Point(360, 10);
                    this->groupBox1->Name = L"groupBox1";
                    this->groupBox1->Size = System::Drawing::Size(237, 244);
                    this->groupBox1->TabIndex = 14;
                    this->groupBox1->TabStop = false;
                    this->groupBox1->Text = L"Conexão";
                    //
                    // groupBox2
                    //
                    this->groupBox2->Controls->Add(this->button5);
                    this->groupBox2->Controls->Add(this->button4);
                    this->groupBox2->Controls->Add(this->label9);
                    this->groupBox2->Controls->Add(this->button3);
                    this->groupBox2->Controls->Add(this->label10);
                    this->groupBox2->Controls->Add(this->label8);
                    this->groupBox2->Controls->Add(this->label7);
                    this->groupBox2->Controls->Add(this->label6);
                    this->groupBox2->Controls->Add(this->label5);
                    this->groupBox2->Location = System::Drawing::Point(14, 12);
                    this->groupBox2->Name = L"groupBox2";
                    this->groupBox2->Size = System::Drawing::Size(278, 241);
                    this->groupBox2->TabIndex = 15;
                    this->groupBox2->TabStop = false;
                    this->groupBox2->Text = L"Medidas";
                    //
                    // button5
                    //
                    this->button5->Location = System::Drawing::Point(140, 200);
                    this->button5->Name = L"button5";
                    this->button5->Size = System::Drawing::Size(117, 23);
                    this->button5->TabIndex = 19;
                    this->button5->Text = L"Parar Log";
                    this->button5->UseVisualStyleBackColor = true;
```

```
                        this->button5->Click += gcnew System::EventHandler(this,
&Form1::button5_Click);
                        //
                        // button4
                        //
                        this->button4->Location = System::Drawing::Point(140, 166);
                        this->button4->Name = L"button4";
                        this->button4->Size = System::Drawing::Size(117, 25);
                        this->button4->TabIndex = 18;
                        this->button4->Text = L"Iniciar Log";
                        this->button4->UseVisualStyleBackColor = true;
                        this->button4->Click += gcnew System::EventHandler(this,
&Form1::button4_Click);
                        //
                        // label9
                        //
                        this->label9->AutoSize = true;
                        this->label9->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 15.75F, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(0)));
                        this->label9->ForeColor = System::Drawing::Color::DarkOrange;
                        this->label9->Location = System::Drawing::Point(125, 95);
                        this->label9->Name = L"label9";
                        this->label9->Size = System::Drawing::Size(19, 25);
                        this->label9->TabIndex = 17;
                        this->label9->Text = L".";
                        //
                        // label10
                        //
                        this->label10->AutoSize = true;
                        this->label10->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 15.75F, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(0)));
                        this->label10->Location = System::Drawing::Point(9, 95);
                        this->label10->Name = L"label10";
                        this->label10->Size = System::Drawing::Size(111, 25);
                        this->label10->TabIndex = 16;
                        this->label10->Text = L"Potência:";
                        //
                        // chart1
                        //
                        chartArea1->Name = L"ChartArea1";
                        this->chart1->ChartAreas->Add(chartArea1);
                        legend1->Name = L"Legend1";
                        this->chart1->Legends->Add(legend1);
                        this->chart1->Location = System::Drawing::Point(14, 260);
                        this->chart1->Name = L"chart1";
                        series1->ChartArea = L"ChartArea1";
                        series1->ChartType =
System::Windows::Forms::DataVisualization::Charting::SeriesChartType::Line;
                        series1->Color = System::Drawing::Color::Red;
                        series1->Legend = L"Legend1";
                        series1->Name = L"Corrente";
                        series2->ChartArea = L"ChartArea1";
                        series2->ChartType =
System::Windows::Forms::DataVisualization::Charting::SeriesChartType::Line;
                        series2->Color = System::Drawing::Color::Navy;
                        series2->Legend = L"Legend1";
                        series2->Name = L"Tensão";
                        series3->ChartArea = L"ChartArea1";
                        series3->ChartType =
System::Windows::Forms::DataVisualization::Charting::SeriesChartType::Line;
                        series3->Color = System::Drawing::Color::DarkOrange;
                        series3->Legend = L"Legend1";
                        series3->Name = L"Potência";
                        this->chart1->Series->Add(series1);
                        this->chart1->Series->Add(series2);
                        this->chart1->Series->Add(series3);
                        this->chart1->Size = System::Drawing::Size(581, 182);
                        this->chart1->TabIndex = 16;
                        this->chart1->Text = L"chart1";
                        //
                        // Form1
                        //
```

```
                        this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
                        this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
                        this->ClientSize = System::Drawing::Size(607, 457);
                        this->Controls->Add(this->chart1);
                        this->Controls->Add(this->groupBox2);
                        this->Controls->Add(this->groupBox1);
                        this->Name = L"Form1";
                        this->Text = L"Wattimetro";
                        this->Load += gcnew System::EventHandler(this, &Form1::Form1_Load);
                        this->groupBox1->ResumeLayout(false);
                        this->groupBox1->PerformLayout();
                        this->groupBox2->ResumeLayout(false);
                        this->groupBox2->PerformLayout();
                        (cli::safe_cast<System::ComponentModel::ISupportInitialize^  >(this-
>chart1))->EndInit();
                        this->ResumeLayout(false);

                }
#pragma endregion

           private: void findPorts(void)
                           {
                                    array<Object^>^objectArray = SerialPort::GetPortNames();
                                    this->comboBox1->Items->AddRange( objectArray );
                           }
       private: System::Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
                                {
                                         timer1->Stop();
                                }
       private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e)
                                {
                                        this->label4->Text=String::Empty;
                                    if(this->comboBox1->Text==String::Empty || this-
>comboBox2->Text==String::Empty)
                                        this->label4->Text="Indefinido!";
                                    else {
                                                try{
                                                        // Verifica se a porta serial não
esta fechada.
                                                if(!this->serialPort1->IsOpen)
                                                        {
                                                                this->serialPort1-
>PortName=this->comboBox1->Text;
                                                        this->serialPort1-
>BaudRate=Int32::Parse(this->comboBox2->Text);
                                                    //Abre a porta serial
                                                        this->serialPort1->Open();
                                                        this->progressBar1->Value=100;
                                                        this->label4->Text="Conectado!";
                                                }
                                                        else
                                                                {
                                                                        this->label4->Text="
Erro!";
                                                                }
                                                    }
                                                catch(UnauthorizedAccessException^)
                                                {
                                                        this->label4->Text=" Falhou!";
                                                }
                                        }
                                }
private: System::Void button2_Click(System::Object^  sender, System::EventArgs^  e)
                 {
                        this->serialPort1->Close();
                        this->button1->Enabled=true;
                        // uatualiza a barra de progresso
                        this->progressBar1->Value=0;
                        // abilita o botão de leitura
                        this->button2->Enabled = true;
                        // abilita o botão de iniialização
                        this->label4->Text="Desconectado!";
                        this->comboBox1->Text==String::Empty;
                        this->comboBox2->Text==String::Empty;
```

```
                        this->label6->Text=String::Empty;
                        this->label8->Text=String::Empty;
                        this->label9->Text=String::Empty;
                        this->timer1->Stop();
                }
private: System::Void button3_Click(System::Object^  sender, System::EventArgs^  e)
                {
                         timer1->Start();
                         // verifica se a porta esta apta para leitura
                     if(this->serialPort1->IsOpen)
                         {
                         // reinicia a string lida
                         this->label6->Text = String::Empty;
                         // mensagem de timeout
                         try
                         {
                         this->label6->Text=this->serialPort1->ReadLine();
                         this->serialPort1->ReadLine();
                         }
                         catch(TimeoutException^)
                         {
                          this->label6->Text="Timeout!";
                         }
                         // disabilita o botão de inicio
                         this->button1->Enabled = false;
                         // abilita o botão desconectar
                         this->button2->Enabled = true;
                         }
                         else
                         // mensagem de erro
                         this->label6->Text="Porta Não aberta";
                }
private: System::Void timer1_Tick(System::Object^  sender, System::EventArgs^  e)
                {
                        label6->Text=corrente.ToString() + " A";
                        label8->Text=tensao.ToString() + " V";
                        label9->Text=potencia.ToString() + " W";
                        chart1->Series["Corrente"]->Points->AddXY(time, corrente);
                        chart1->Series["Tensão"]->Points->AddXY(time, tensao);
                        chart1->Series["Potência"]->Points->AddXY(time, potencia);
                        time++;

                       serialPort1->DiscardInBuffer();

                        if (i == 10 && g == 1)
                        {
                                sw->WriteLine(DateTime::Now + "\r\t" + "Corrente: " +
corrente.ToString() + "A\r\t" + "Tensão: " + tensao.ToString() + "V\r\t" + "Potência: " +
potencia.ToString() + "W\r\n");
                                i = 0;
                        }
                        i++;
                        }

private: System::Void serialPort1_DataReceived(System::Object^  sender,
System::IO::Ports::SerialDataReceivedEventArgs^  e)
                {
                        static char dado;
                        dado = serialPort1->ReadByte();
                        switch(dado)
                        {
                                case 0x43:
                                        corrente = Convert::ToDouble(serialPort1-
>ReadLine());
                                        break;
                                case 0x54:
                                        tensao = Convert::ToDouble(serialPort1->ReadLine());
                                        break;
                                case 0x50:
                                        potencia = Convert::ToDouble(serialPort1-
>ReadLine());
                                        break;
                        }
                }
```

```
private: System::Void button4_Click(System::Object^  sender, System::EventArgs^  e)
                 {
                        g = 1;
                 }


private: System::Void button5_Click(System::Object^  sender, System::EventArgs^  e)
                 {
                        sw->Close();
                        g = 0;
                 }
};
}
```