TRABALHO DE GRADUAÇÃO

DESENVOLVIMENTO DE UM MÓDULO DE MEDIÇÃO DE ENERGIA WIRELESS COM TRANSMISSOR EM TEMPO REAL DE PARÂMETROS PARA CARGAS DE ATÉ 10KW

André Luiz Siega Nepomuceno

Gabriel Calache Cozendey

Brasília, abril de 2013

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

DESENVOLVIMENTO DE UM MÓDULO DE MEDIÇÃO DE ENERGIA WIRELESS COM TRANSMISSOR EM TEMPO REAL DE PARÂMETROS PARA CARGAS DE ATÉ 10KW

André Luiz Siega Nepomuceno

Gabriel Calache Cozendey

Relatório submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação

Banca Examinadora

Prof. Adolfo Bauchspiess, ENE/UnB _____
Orientador

Prof. Eduardo Stockler Tognetti, ENE/UnB _____
Examinador externo

Prof. Lélio Ribeiro Soares Júnior, ENE/UnB _____
Examinador externo

Dedicatórias

Aos meus pais, amigos, família e professores

Gabriel Calache Cozendey

Dedico este trabalho aos meus pais, que são os grandes alicerces dessa importante conquista.

André Luiz Siega Nepomuceno

RESUMO

O presente trabalho apresenta o desenvolvimento de um módulo *wireless*, fazendo uso de rede uma XBee, de medição de energia para redes monofásicas que suporta cargas de até 10 kW (até 230 V e 40 A). Este módulo também conta com um programa supervisório desenvolvido em ScadaBR que monitora parâmetros de energia tais como potência ativa, potência reativa, tesão e corrente de forma autônoma. Este módulo também pode ser expandido para funcionar como atuador liga/desliga na carga.

ABSTRACT

In the present work we present the defelopment of a wireless single-phase energy meter module, using a XBee network, that supports loads up to 10 kW (up to 230 V and 40 A). This module also features a supervisory software developed in ScadaBR that autonomously monitors energy parameters such as active power, reactive power, voltage and current. This module can also be expanded to function as an on-load on/off actuator.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Latinos

T	Período	[s]
V	Tensão	[V]
I	Corrente	[A]
P	Potência	[W]
S	Potência Aparente	[VA]

Símbolos Gregos

ω	Frequência Angular	[rad/s]
θ	Fase	[rad]

Subscritos

RMS	Root Mean Square
out	Saída
in	Entrada

Sobrescritos

$\rightharpoonup$	Fasor

Siglas

AC	Alternate Current
A/D	Analógico/Digital
API	Application Programming Interface
CAD	Computer-Aided Design
CD	Compact Disc
CI	Circuito Integrado

CS	Chip Select
CMOS	Complementary Metal-Oxide-Semiconductor
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSMA/CA	Carrier Sense Multiple Access/Collision Avoidance
D/A	Digital/Analógico
DC	Direct Current
DCS	Distributed Control Systems
FIFO	First In First Out
FP	Fator de Potência
GND	Ground
HMI	Human-Machine Interface
ICSP	In-Circuit Serial Programming
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
I/O	Input/Output
ISA	International Society of Automation
ISM	Industrial, Scientific and Medical
LARA	Laboratório de Robótica e Automação
LED	Light-Emitting Diode
MISO	Master In Slave Out
MOSI	Master Out Slave In
OPC	OLE for Process Control
PAN	Personal Area Network
PC	Personal Computer
PCB	Printed Circuit Board
PLC	Programmable Logic Controller
PWM	Pulse-Width Modulation
RAM	Random-Access Memory
RF	Radiofrequência
RMS	Root Mean Square
RTU	Remote Terminal Unit
SCADA	Supervisory Control and Data Acquisition
SCK	Source Clock
SPI	Serial Peripheral Interface
TCP/IP	Transmission Control Protocol/Internet Protocol
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus

Capítulo 1

Introdução

Este capítulo trata sobre o contexto do trabalho, a motivação, as possíveis aplicações, além de descrever quais são os objetivo a serem atingidos e qual a sua estrutura do conteúdo.

1.1 Contextualização

O uso racional da energia é um tema de grande relevância. Tanto na indústria como na vida cotidiana se observa uma tendência em reduzir desperdícios para aumentar o rendimento dos recursos, entre eles energia elétrica. Esta prática não é apenas uma forma de reduzir custos e aumentar a competitividade, ela também reflete a conscientização atual em relação à necessidade e benefícios de um modelo de desenvolvimento sustentável. No contexto de automação predial o monitoramento contínuo do consumo de energia permite o desenvolvimento de algoritmos que aumentam a eficiência energética considerando o valor em tempo real dos parâmetros relevantes ao processo. Por exemplo, podemos identificar aparelhos defeituosos, apresentando consumo energético acima do usual, ou a necessidade da reposição do gás de um compressor de aparelho de ar condicionado prontamente com o auxilio desta informação. Os parâmetros de energia considerados relevantes para algoritmos de eficiência energética são: tensão, corrente, frequência, potência total, potência ativa, potência reativa e fator de potência. Neste trabalho procuramos desenvolver um módulo que faça a medição de parâmetros energéticos remotamente e os apresente em um programa supervisório sendo executado em um computador pessoal.

1.2 Objetivos

Para o ambiente de automação predial do Laboratório de Robótica e Automação (LARA) deverá ser criada uma rede *wireless* demonstrativa abrangendo o monitoramento de consumo de energia de 3 aparelhos de ar condicionado integrados a um sistema supervisório SCADA.

1.3 Estrutura do Documento

O presente trabalho segue a estrutura apresentada a seguir: no capitulo 2 faremos uma revisão teórica sobre energia e medição de energia com o intuito de esclarecer os parâmetros relevantes ao trabalho. Depois, no capitulo 3 fazemos o detalhamento dos recursos empregados neste trabalho, explicaremos suas funcionalidades e funções no contexto do nosso projeto. Também serão explicados os conceitos de software supervisório e SCADA. A seguir no capitulo 4 mostraremos de forma detalhada como estes componentes interagem para formar nosso modulo de medição e atingirmos ao resultado desejado. Já no capitulo 5 apresentaremos o modulo completo e resultados obtidos em dois testes: o primeiro com um secador de cabelo e o segundo com três aparelhos de ar condicionado do LARA. Por fim o capitulo 6 apresenta as conclusões do projeto, uma lista de *bugs* conhecidos e propostas de trabalhos futuros.

Capítulo 2

Revisão Teórica

Este capítulo aborda os principais fundamentos teóricos que tratam sobre medição de energia e aquisição de dados. Na primeira parte é apresentada uma breve revisão sobre os conceitos por trás da medição de energia e no final há uma breve introdução aos princípios básicos de aquisição de dados e conversão analógica para digital.

2.1 Medição de Energia

2.1.1 Valor Eficaz (RMS)

Nas busca de um critério eficiente para se medir a energia transferida por uma rede, foi adotado o critério do valor eficaz ou *Root Mean Square*, dado por:

$$X_{\mathrm{RMS}} = \sqrt{\frac{1}{T}\int_{t}^{t+T} [x(t)]^2 dt}$$

Onde $x(t)$ representa o valor instantâneo da grandeza (e.g. tensão ou corrente) e T o período do sinal.

O valor eficaz, em termos estatísticos, é a medida da magnitude de uma quantidade que varia. É especialmente útil em medições onde as grandezas medidas variam entre valores negativos e positivos, como em sinais senoidais.

2.1.2 Potência em Circuitos de Corrente Alternada

Considerando a energia elétrica que flui entre duas redes, a potência instantânea é definida como

$$p(t) = v(t)i(t)$$

quando, idealmente,

$$v(t) = V \sin(\omega t) \tag{2.1}$$

$$i(t) = I \cos(\omega t - \theta) \tag{2.2}$$

Onde $v(t)$ representa tensão instantânea, V a tensão máxima, ω a frequência angular e θ a diferença de fase entre $v(t)$ e $i(t)$.

A partir das equações 2.1 e 2.2, tem-se que

$$p(t) = v(t)i(t) = V \sin(\omega t) I \cos(\omega t - \theta) \tag{2.3}$$

Simplificando o produto trigonométrico da equação 2.3, obtém-se

$$p(t) = \frac{VI}{2}(\cos(\theta) - \cos(2\omega t - \theta)) \tag{2.4}$$

Como a potência instantânea muda a todo instante, é conveniente utilizar seu valor médio a partir da equação 2.4, ou seja,

$$P_{\text{média}} = \frac{1}{T}\int_{t}^{t+T} p(t)dt = V_{RMS} I_{RMS} \cos(\theta)$$

Este resultado representa a taxa de variação média da energia que flui na rede. Essa potência, denominada potência ativa, representa a taxa de energia consumida no circuito.

2.1.3 Potência Complexa

Utilizando a notação fasorial para corrente e tensão,

$$\vec{I} = \mathrm{I}e^{\mathrm{j}\alpha} \text{ e } \vec{V} = \mathrm{V}e^{\mathrm{j}\beta} \tag{2.5}$$

define-se a potência complexa que flui entre duas redes como

$$\vec{S} = \vec{V}\vec{I^*} \tag{2.6}$$

onde $\vec{I^*}$ é o complexo conjugado de $\vec{I}$.

Assim, a partir das equações 2.5 e 2.6:

$$\vec{S} = \vec{V}\vec{I^*} = \mathrm{V}e^{\mathrm{j}\beta}\mathrm{I}e^{\mathrm{j}\alpha} = VIe^{j(\beta-\alpha)} = VIe^{j\phi}$$

A magnitude de $\vec{S}$ é denominada potência aparente:

$$\left|\vec{S}\right| = S = VI$$

Na forma retangular,

$$\vec{S} = VI\cos(\phi) + jVIsen(\phi) = P + jQ$$

Onde P é denominada potência ativa (ou real) e Q é denominada potência reativa.

Os sinais de P e Q estão associados ao sinal de ϕ. Se $P > 0$, associa-se uma taxa de variação média de energia positiva fluindo. Se $P < 0$, o sentido do fluxo é oposto.

A potência reativa, ou Q, está relacionada ao armazenamento de energia em impedâncias capacitivas ou indutivas. Esta grandeza não contribui para o trabalho realizado pelo sistema em questão. Adota-se as seguintes unidades para cada grandeza:

S: volt-ampere (VA)

P: watt (W)

Q: volt-ampere-reativo (VAR)

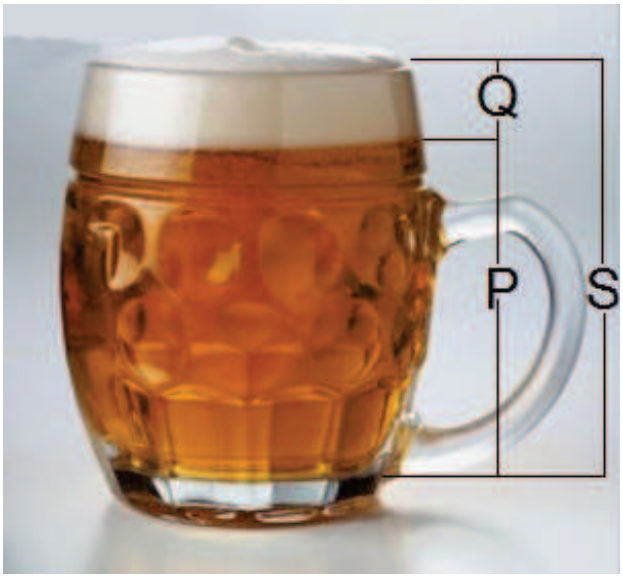

Figura 2.1: Analogia entre copo de cerveja e potência complexa

Para fins didáticos, é comum ilustrar a relação entre as potências com um copo cheio de cerveja (figura 2.1). O volume total do copo corresponde à potência aparente S; O volume ocupado pelo líquido corresponde a potência ativa P; E o volume ocupado pela espuma corresponde a potência reativa Q. Nesta comparação, a potência ativa (o líquido) é quem de fato representa o trabalho do sistema. A potência reativa representa uma energia que é indispensável para o funcionamento de sistemas que possuem componentes reativos (i.e. capacitivos ou indutivos), porem que não realiza trabalho. A potência aparente (volume total do copo) limita essas duas grandezas, cujo valor total é a soma das duas.

2.1.4 Potência Reativa

O fluxo de potência reativa é necessário em um sistema de transmissão AC para que haja transferência de energia real (potência ativa) para a rede. Em tais sistemas AC, a energia é armazenada temporariamente em elementos reativos, o que pode resultar em uma reversão periódica do fluxo. A parte do fluxo restante, que não foi revertida, é a potência ativa, isto é, a energia que é utilizada para realizar trabalho. O fluxo reativo que fica armazenado em elementos capacitivos ou indutivos, na forma de campos magnéticos ou elétricos, é periodicamente retornado a fonte de energia.

2.1.5 Fator de Potência

O fator de potência de um sistema elétrico é dado pela razão entre a potência ativa e a potência aparente. Devido a energia reativa armazenada no sistema ou a não-linearidades na carga, a potência reativa pode ser maior que a potência ativa, diminuindo o rendimento total do sistema.

Um sistema com um baixo fator de potência absorve mais corrente que outro com um maior fator de potência para realizar o mesmo trabalho. Correntes maiores aumentam as perdas de energia ao longo do sistema, demanda fios mais grossos (e mais caros), além de requerer equipamentos mais específicos.

O fator de potência é definido como

$$FP = \frac{P}{S} \tag{2.7}$$

No caso de um sinal senoidal, a equação 2.7 pode ser escrita da seguinte forma:

$$|FP| = |\cos(\phi)| = \frac{|P|}{|S|}$$

2.2 Instrumentos de Medição de Energia

Com o crescimento da cultura do uso racional da energia elétrica - além da necessidade de se avaliar a qualidade da energia fornecida por companhias energéticas - técnicas, tecnologias e padronizações da medição de energia ganharam grande atenção em inúmeras áreas de estudo.

A ideia principal de um instrumento de medição de energia é medir e indicar o valor de alguma grandeza elétrica. Também é interessante armazenar tais medidas para construir históricos e/ou fazer análises mais aprofundadas.

2.2.1 Conversor Analógico-Digital

Instrumentos de medição fazem uso de diferentes tipos de conversores A/D. Várias arquiteturas são possíveis, mas todas produzem o mesmo resultado: a quantificação de uma amostra de sinal analógico em um sinal digital equivalente.

A figura 2.2 mostra uma versão muito simples de um conversos A/D. Nesse exemplo, um contador binário é incrementado *bit* a *bit* a cada pulso do *clock*, até que V_{out} seja igual a V_{in}. Esse tipo de conversor é chamado de "rampa e contador digital" pois a forma de onda de V_{out} lembra muito uma rampa de pulsos unitários ou uma escada.

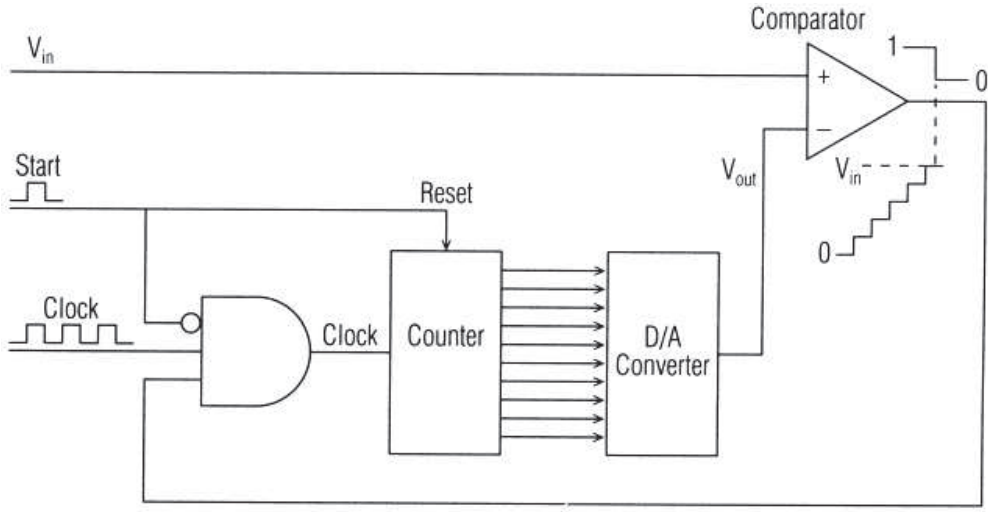

Figura 2.2: Conversor A/D

O circuito opera da seguinte maneira:

1. Um pulso de inicio (Start) é gerado, zerando assim o contador. Ele também inibe a porta AND para que o *clock* não desencadeie incrementos indevidos.
2. Com o contador em zero, $V_{out} = 0$, assim a saída do comparador é 1.
3. Quando a entrada Start e colocada no nível 0, a porta AND é habilitada, fazendo com que os pulsos de *clock* gere incrementos.
4. Com o avanço do contador, o saída do conversor digital/analógico incrementa a cada passo, aumentando a tensão de V_{out} de acordo com a sua resolução.
5. Quando $V_{out} \geq V_{in}$, a saída do comparador se torna 0, desabilitando os pulsos de *clock*. Assim, o processo de conversão A/D está completo, e o equivalente do sinal analógico é a saída do contador digital.

2.3 Resolução

Podemos ilustrar o conceito de resolução através de um exemplo.

Partindo do conversor A/D da figura 2.2, vamos supor que seu conversor D/A possua uma entrada de 10 bits e uma escala de saída de 10,23 V. Vamos supor também que o comparador pode detectar diferenças de no mínimo 1 mV e que V_{in} vale 3,728 V.

O conversor D/A possui uma entrada de 10 bits, ou seja, o maior número possível de incrementos do contador é $(2^{10} - 1) = 1023$. Sabendo que uma tensão de saída de 10,23 V é atingida quando o contador atinge 1023, então podemos concluir que a cada pulso de *clock* (ou um incremento no contador) temos um incremento em V_{out} de $10,23/1023 = 10$ mV. Assim, sabendo que $V_{in} = 3,728$ V e que o limiar do comparador é 1 mV, V_{out} tem que atingir V_{in} em 373 incrementos ou pulsos de *clock*.

No final da conversão, o a saída do contador é 373 ou 0101110101 em binário. Esse é o equivalente digital para a tensão V_{in}. A resolução desse conversor A/D é igual ao incremento do conversor D/A, que é 10 mV ou aproximadamente $0,01/10,23 * 100\% = 0,1\%$.

De outro ponto de vista, a entrada V_{in} pode assumir infinitos valores entre 0 e 10,23 mV. Entretanto, a saída do conversor A/D será sempre limitada aos 1024 valores discretos (0,00 0,01 0,02 ... 10,23 V). Essa limitação da discretização de um valor analógico é chamada resolução.

2.4 Acurácia

A acurácia do conversor D/A não relacionada com a resolução do mesmo. O que define a acurácia do conversor é a precisão ou perfeição do seus componentes, como os resistores que com compõe, os comparadores, amplificadores operacionais, tensões de referência, entre outros. Por exemplo, se os resistores que compõe o conversor D/A possuem tolerância de 1%, isso gera um erro inerente nas medições que é somado ao efeito quantização do conversor. A acurácia também depende da escala do sinais envolvidos, já que os valores de tolerância ou perfeição são dados relativos às grandezas envolvidas.

2.5 Precisão

Precisão define o grau de variação de uma mesma medição. No caso do conversor, ao se aplicar uma mesma tensão de entrada V_{in}, a saída nem sempre será determinística. Este efeito pode ser causados por flutuações no circuito ou interferência externas. Por exemplo, se temos uma diferença de tensão no comparador que está muito próxima do seu limiar (1 mV no exemplo), a saída pode oscilar entre as duas medidas mais próxima.

Os conceitos de acurácia e precisão são ilustrados pela figura 2.3

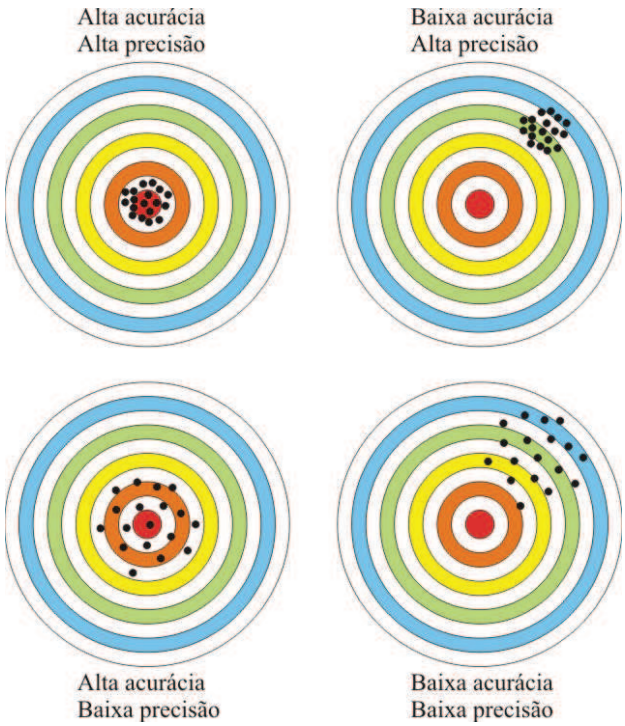

Figura 2.3: Ilustração dos conceitos de acurácia e precisão

Capítulo 3

Recursos

Neste capitulo apresentaremos uma visão geral do projeto e descreveremos todos os materiais, componentes, softwares e protocolos que foram utilizados para a realização do trabalho. Abordaremos em detalhes as propriedades relevantes de cada um dos constituintes do projeto. Ao final do capitulo já teremos uma ideia de como nosso projeto se desenvolve. A seguir no capitulo 3 acompanharmos como estes elementos se encaixam no contexto do nosso projeto.

3.1 Visão Geral do Projeto

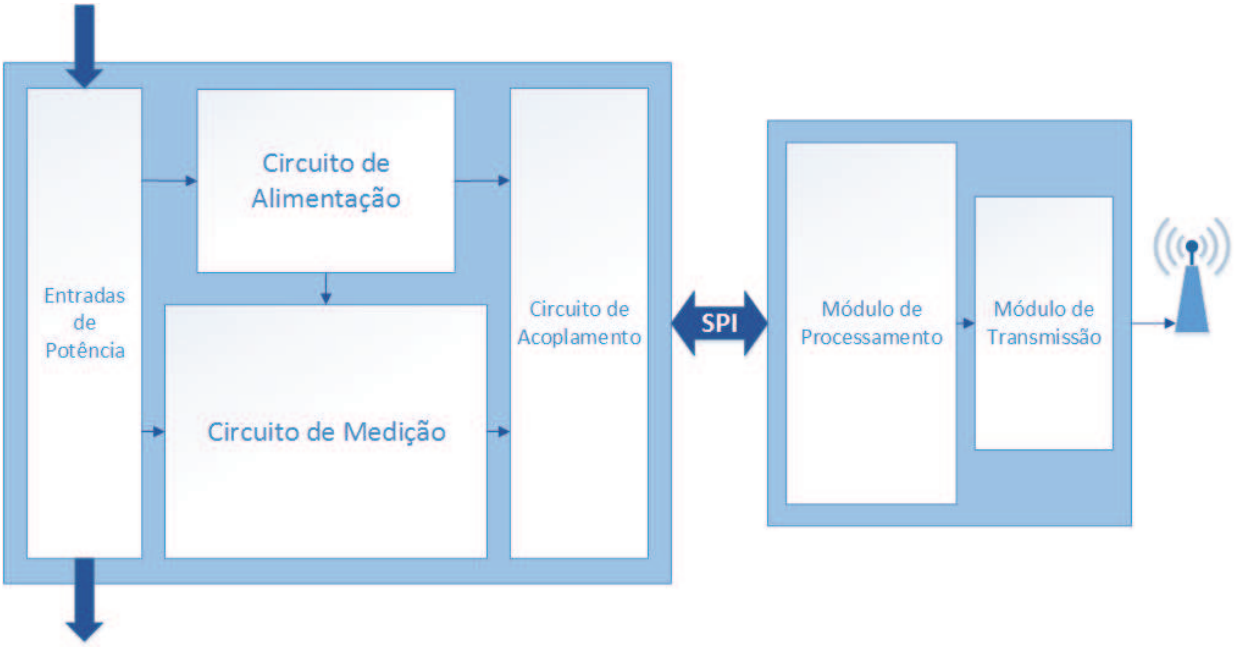

Figura 3.1: Visão Geral do Projeto

O nosso trabalho foi iniciado a partir da escolha do SA9903B como nosso medidor de energia

devido a sua disponibilidade imediata no laboratório, praticidade no formato dos dados e comunicação padronizada. Este CI é compatível com nossos requisitos de projeto que são de 230 V e 40 A, totalizando 9200 W. A partir das especificações do projeto e deste medidor foram escolhidos os demais componentes e o restante do circuito foi projetado.

O circuito completo foi idealizado como ilustrado na figura 3.1 e é dividido em dois grandes sub-módulos: o de medição, que foi projetado e fabricado por nós mesmos, e o de transmissão, composto por um Arduino e XBee.

3.2 Resistor Shunt TGHGCR0050FE-ND

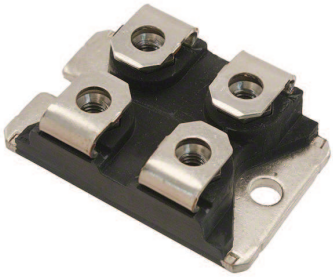

Figura 3.2: TGHGCR0050FE-ND

A medição de energia da rede é feita de maneira monofásica em uma das linhas de alimentação. O acesso a esta linha é feito por meio de um resistor *shunt*. Resistores *shunt* possuem características especificas em relação a outros resistores e são particularmente úteis para medição de corrente nas quais não podemos ter um acesso direto devido a seu elevado valor. Assim ao invés de medir a corrente diretamente como ao se utilizar galvanômetro, medimos a queda de tensão neste resistor *shunt* e derivamos a corrente (Lei de Ohm). Como principais características de um resistor *shunt* podemos citar: uma resistência muito baixa para evitar perdas por dissipação e ter uma queda de tensão desprezível em relação ao sistema e uma resistência conhecida e precisa (com erro percentual baixo) para termos maior precisão na medida da corrente.

O nosso resistor *shunt* foi escolhido como especificado no *datasheet* do SA9903B, faz a medição de energia e pede uma queda de tensão de 200 mV para a corrente máxima do medidor (Imax). No nosso caso desejamos medir uma linha de 230 V e no máximo 40 A. Logo

$$R_{\text{shunt}} = 0.2/40 = 0.005\Omega$$

O resistor escolhido é o TGHGCR0050FE-ND ilustrado na figura 3.2 devido a sua disponibilidade do mercado e a dificuldade de se encontrar alternativas para valores tão diminutos de resistência. Este resistor *shunt* possui uma resistência de 0.005 Ω com tolerância de 1%, opera em até 100 W e 50 A a 70 °C, é não indutivo e possui 4 terminais para isolar o caminho de escoamento da corrente do caminho usado para a medição desta. Para nossas especificações teremos

uma potência dissipada de $P = \mathrm{RI}^2 = 0.005 * 40^2 = 8\ W$ e não será necessária a utilização de um dissipador de calor. Fazendo deste resistor, apesar da falta de opções no mercado de consumidor, uma excelente escolha para nosso projeto.

3.3 Sames SA9903B

O circuito integrado SA9903B, da Sames, é um medidor de energia/potência monofásico que mede potência ativa, reativa, tensão RMS e frequência da rede, seu diagrama de blocos está ilustrado na figura 3.3. Tais medidas podem ser lidas em registradores de 24 bits via barramento SPI.

Este circuito integrado se encaixa perfeitamente na proposta do trabalho, que exige cálculos de gasto de energia em ambientes prediais e permite supervisão e controle do consumo.

O SA9903B está disponível em encapsulamento plástico *dual-in-line*, assim como em *small-outline*, ambos com 20 pinos (padrões PDIP20 e SOIC20, respectivamente).

3.3.1 Características Gerais

- Medição bidirecional de potência ativa e reativa
- Medição de corrente e tensão RMS
- Barramento SPI
- Atende as normas IEC 61036 (medidor de watt hora AC classe 1) e IEC 61286 (medidor de VAR hora)
- Erro de menos de 1% para medição da potência ativa, em uma escala de 1:1000
- Erro de menos de 2% para medição da potência reativa, em uma escala de 1:1000
- Proteção contra descarga eletrostática
- Baixo consumo de energia: menos de 25 mW
- Adaptável a diferentes sensores de corrente
- Tensão de referência precisa e integrada

3.3.2 Funcionalidades

O SA9903B é um circuito integrado analógico/digital, fabricado com tecnologia CMOS, que realiza medição de potência ativa, potência reativa, tensão RMS e frequência da rede. Este circuito inclui todas as funções necessárias para a medição de potência e energia monofásica, tais como:

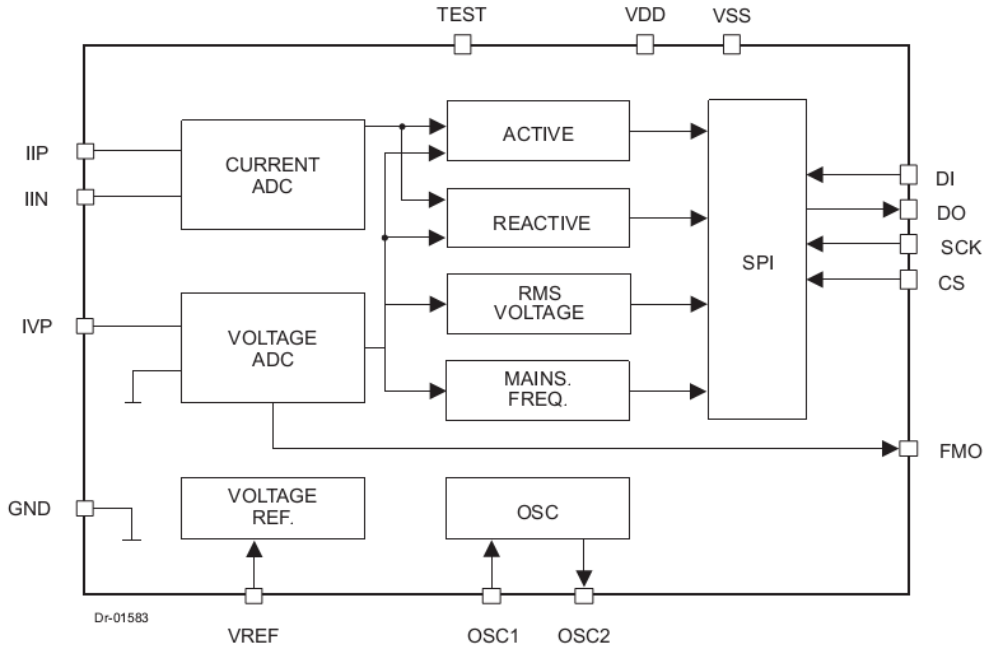

Figura 3.3: Diagrama de blocos do SA9903B

conversão A/D para medição da corrente e da tensão, cálculo de potência e integração da energia ao longo do tempo.

O CI integra a potência ativa e reativa instantânea em registradores de 24 bits. A tensão RMS e frequência são constantemente medidas e armazenadas nos respectivos registradores. Existe também uma saída que indica a passagem da tensão da tensão (*zero crossover* - FMO).

Sua interface SPI possui saídas *tri-state* que permitem a integração de vários dispositivos num único barramento.

3.3.3 Sinais de Entrada

3.3.3.1 Configuração das Entradas Analógicas

A configuração interna dos circuitos de entrada é apresentada na figura 3.4. Tais entradas são protegidas contra descarga eletrostática graças aos diodos de proteção. A retroalimentação nos amplificadores gera o curto-circuito virtual nas entradas, produzindo uma réplica exata do sinal de entrada nos circuitos processadores de sinal analógico. As entradas dos sensores são idênticas. Ambas são diferenciais e suportam picos de corrente de até $\pm 25\mu A$. Um dos terminais do sensor de tensão (IVP) é aterrado internamente, o que pode ser feito graças à baixa sensibilidade a sinais e impedâncias parasitas.

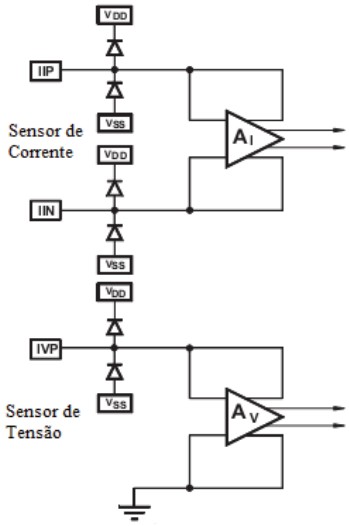

Figura 3.4: Configuração interna das portas de entrada analógica

3.3.3.2 Entrada do Sensor de Corrente (pinos IIP e IIN)

Tipicamente, resistores que limitam a corrente são ligados a essas entradas. Em condições nominais, o valor desses resistores é calculado de forma que a corrente nominal na entrada seja $16\mu A_{RMS}$. O sensor satura para correntes que ultrapassam $\pm 25\mu A$.

3.3.3.3 Entrada do Sensor de Tensão (pino IVP)

Tipicamente a tensão da rede é reduzida para 14 V_{RMS} nas condições nominais. Para tal fim um divisor de tensão pode ser utilizado. A corrente no sensor de tensão deve ser limitada a 14 μA_{RMS}. O sensor satura para correntes que ultrapassam ± 25 μA.

3.3.3.4 Tensão de Referência (pino VREF)

O pino VREF deve estar ligado a um resistor de polarização, cujo valor ideal é de 24 $k\Omega$.

3.3.3.5 Serial Clock (SCK)

O pino SCK é utilizado para sincronizar a troca de dados entre o microcontrolador e o SA9903B. O sinal de *clock* é gerado pelo controlador e determina as taxas de transmissão dos pinos de entrada e saída de dados (DI e DO, respectivamente).

3.3.3.6 Serial Data IN (DI)

O pino DI é a entrada serial de dados. Os dados são transferidos a uma taxa determinada pelo *clock* serial (SCK) e a transferência ocorre somente quando a entrada chip *select* (CS) estiver

ativa.

3.3.3.7 Chip Select (CS)

A entrada CS é utilizada para endereçar o CI. Um nível lógico alto ativa a troca de dados.

3.3.4 Sinais de Saída

3.3.4.1 Serial Data Out (DO)

O pino DO é a saída serial de dados. O *clock* serial (CSK) determina a taxa de transmissão dos dados, que são transmitidos apenas quando CS estiver ativo. Essa saída é *tri-state* quando CS está desativado.

3.3.4.2 Sensor de Passagem por Zero (FMO)

A saída FMO gera um sinal que segue as passagens da tensão de rede por zero.

3.3.4.3 Interface SPI

A interface SPI (acrônimo para *Serial Peripheral Interface bus*) é um barramento síncrono utilizado na transferência de dados entre o controlador e o SA9903B. Os pinos DO (*Serial Data Out*), DI (*Serial Data In*), CS (*Chip Select*) e SCK (*Serial Clock*) são utilizados na execução do barramento. Neste caso, o SA9903B é configurado como dispositivo escravo (*slave*) e o controlador é configurado como dispositivo mestre (*master*). A entrada CS inicia e termina a transferência de dados. O sinal SCK (gerado pelo controlador) sincroniza os sinais a leitura e escrita de dados. DO e DI são, respectivamente, a saída e entrada de dados do SA9903B.

3.3.4.4 Acesso aos Registradores

O SA9903 contém quatro registradores de 24 bits que armazenam a potência ativa, reativa, tensão da rede e frequência da rede. O endereço de cada registrador é mostrado na tabela 3.1.

Tabela 3.1: Endereço dos registradores no SA9903B

Registrador	Bits de Cabeçalho	x	x	A5	A4	A3	A2	A1	A0
potência Ativa	1	1	0	X	X	0	0	0	0
potência Reativa	1	1	0	X	X	0	0	0	1
Tensão	1	1	0	X	X	0	0	1	0

A sequência 110 (0x06) deve proceder ao endereço de 6 bits do registrador que está sendo acessado. Quando CS está ativo, os dados no pino DI são validados pelo SA9903B na borda de subida de SCK, como mostrado na figura 3.5.

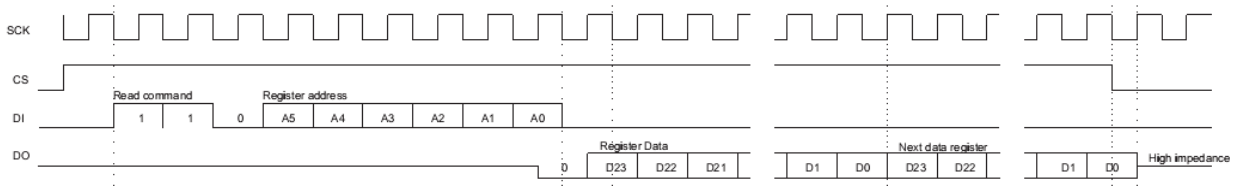

Figura 3.5: Forma de onda da comunicação SPI para o SA9903B

Os registradores devem ser lidos individualmente, sem restrição a ordem de acesso. Depois que um registrador é lido seu valor é continuamente transmitido até que CS seja inativo ou outro registrador seja selecionado. Os 9 bits necessários para o acesso ao registrador podem ser preenchidos com um prefixo de zeros caso o controlador requeira palavras de tamanho fixo (8 bits geralmente) para o SPI. Por exemplo, a seguinte sequencia seria válida:

Tabela 3.2: Exemplo de sequencia de leitura no SA9903B

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Valor	0	0	0	0	0	0	0	1	1	0	A5	A4	A3	A2	A1	A0

3.3.4.5 Formato dos Dados

Após o último bit menos significante do endereço do registrador ser transmitido na borda de subida do *clock* (SCK), a saída D0 fica no nível baixo na borda de descida de SCK. A cada borda de descida subsequente em SCK, um novo bit será validado na saída DO.

O conteúdo de cada registrador possui 24 bits. O bit mais significante é transmitido primeiro.

3.3.4.6 Registradores de potência Ativa e Reativa

Os registradores de potência ativa e reativa são acumuladores de 24 bits, que são incrementados ou decrementados a uma taxa de 320 mil amostras por segundo, nas condições nominais.

O valor do registrador será incrementado para um fluxo positivo de energia e decrementado para um fluxo negativo, como na figura 3.6.

Nas condições nominais, os registradores serão reiniciados a cada 52 segundos. O controlador precisa levar esse contexto em conta durante o calculo da diferença entre duas medidas sucessivas.

3.3.4.7 Registrador de Tensão

O valor do registrador de tensão armazena a tensão RMS. Tal valor é medido com precisão menor que 1% para uma escala de 50% a 115% da tensão nominal.

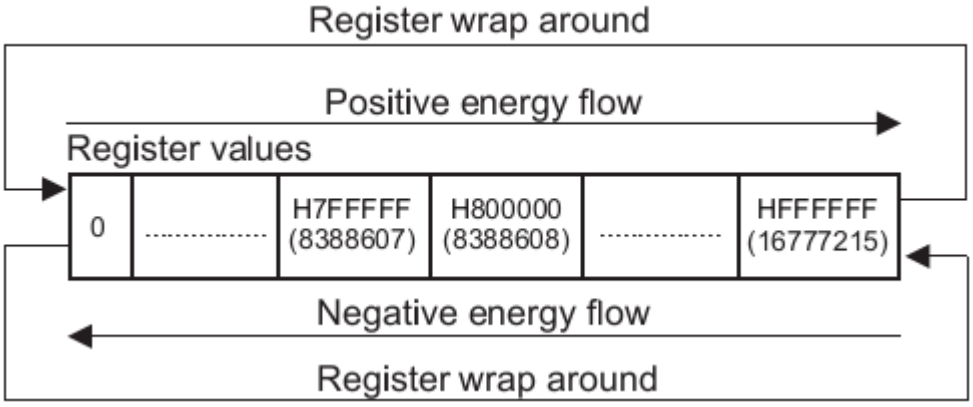

Figura 3.6: Funcionamento do registrador acumulador de potência ativa/reativa

3.3.4.8 Registrador de Frequência

O registrador de frequência contem informações sobre a frequência da rede. Apenas os 10 bits menos significantes (0 a 9) são utilizados para o cálculo da frequência. O bit mais significante (23) muda de estado na borda de subida da tensão.

3.3.5 Cálculos de potência

Sinais de potência instantânea são gerados pela multiplicação dos sinais de corrente e tensão.

$$\text{potência ativa} = VI\cos(\theta)$$

$$\text{potência reativa} = VI\sin(\theta)$$

Tais sinais são continuamente acumulados nos respectivos registradores.

3.3.6 Utilizando os Valores dos Registradores

3.3.6.1 potência Ativa e Reativa

A potência ativa e reativa medida pelo SA9903B são calculadas como se segue:

$$\text{potência (W ou VAr)} = V_{RATED} * I_{RATED} * N/INT_{TIME}/320000$$

Onde:

V_{RATED} = Tensão nominal

I_{RATED} = Corrente nominal

N = Diferença entre os registradores para leituras sucessivas

INT_{TIME} = Diferença de tempo entre leituras sucessivas (em segundos)

3.3.6.2 Tensão da Rede

A tensão RMS medida é calculada da seguinte forma:

$$\text{Tensão (V)} = V_{RATED} * V_{REGISTER}/700$$

Onde:

V_{RATED} = Tensão nominal

$V_{REGISTER}$ = Valor do registrador de tensão

3.3.6.3 Frequência da Rede

Apenas os 10 bits menos significativos (0 a 9) do registrador são utilizados para calculo da frequência. Tal valor é calculado como se segue:

$$\text{Frequência (Hz)} = F_{CRYSTAL}/256/F_{REGISTER}$$

Onde:

$F_{CRYSTAL}$ = Frequência do cristal externo

$F_{REGISTER}$ = Valor dos bits de 0 a 9 do registrador

3.4 Optoacoplador 6N137

Um optoacoplador é um dispositivo eletrônico projetado para transferir sinais elétricos utilizando ondas eletromagnéticas, o que proporciona isolamento físico e elétrico entre sua entrada e saída. Sua principal função é evitar que sinais de alta tensão ou estados transientes danifiquem circuitos mais sensíveis ou provoquem distorções em sinais.

Como mostrado na figura 3.7, um optoacoplador é constituído basicamente de um emissor de luz, geralmente um LED, e um detector, geralmente um fototransistor.

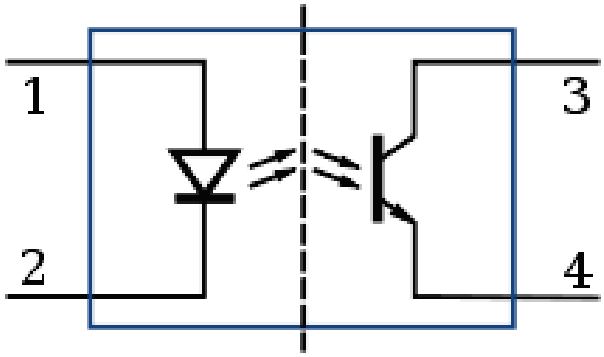

Figura 3.7: Circuito básico de um optoacoplador

O 6N137 é um optoacoplador com um canal unidirecional e consiste de um LED opticamente acoplado a um fotodetector de alta velocidade. Possui um *shield* interno com rejeição em modo comum (*common mode rejection*) de até 10 $kV/\mu s$ e suporta velocidades de até 10 Mb/s. Neste trabalho, o optoacoplador é necessário para isolar os circuitos de medição (por onde fluem altas tensões) do controlador. Além disso, ele tem a função de tradutor de tensões, transformando níveis lógicos de -2,5 V (baixo) a +2,5 V (alto) para 0 V (baixo) a 5,0V (alto), desempenhando um papel fundamental na comunicação entre dispositivos.

3.5 Arduino

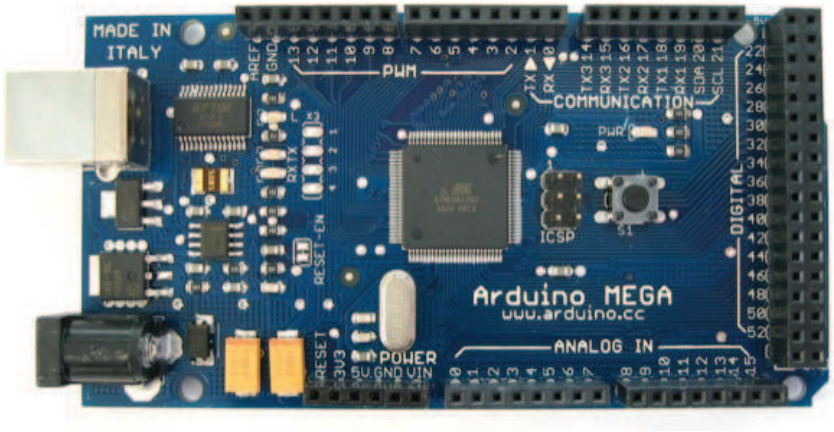

Figura 3.8: Arduino Mega

O Arduino é uma plataforma *open-source* de prototipagem de *hardware* baseada em uma simples placa de microcontrolador e uma interface de desenvolvimento de *software*. Seu objetivo é tornar o desenvolvimento de projetos de eletrônica mais acessíveis, fornecendo uma interface e bibliotecas extremamente amigáveis ao usuário. O *hardware* consiste em uma placa com I/Os disponíveis para uso imediato, permitindo o acesso simplificado a praticamente todas as funcionalidades disponibilizadas pelo microcontrolador utilizado. O *software* consiste em uma IDE com compilador para código em C e de um *bootloader* para o microcontrolador, permitindo a gravação serial diretamente a partir de uma porta serial virtual. Existem atualmente diversas versões de Arduino disponíveis no mercado, todas elas usando microcontroladores da Atmel, cada uma com características próprias e atendendo a diversas necessidades. Por se tratar também de um Projeto *open-hardware*, temos disponíveis todos os esquemáticos das placas proporcionando grande transparência ao desenvolvermos um projeto, além de um baixo custo, já que qualquer um pode montar própria placa. Por suas qualidades o Arduino atraiu diversos desenvolvedores não só de *hardware* mas de *software* também, e hoje temos disponíveis uma enorme quantidade de bibliotecas para Arduino que facilitam enormemente qualquer tipo de desenvolvimento.

No nosso projeto utilizamos os Arduino MEGA (figura 3.8) já disponíveis no LARA. O Arduino MEGA é uma placa baseada no microcontrolador ATmega1280, possuindo 128 kB de memória flash e 8 kB de memória RAM, 54 pinos digitais de entrada e saída onde 14 deles podem ser

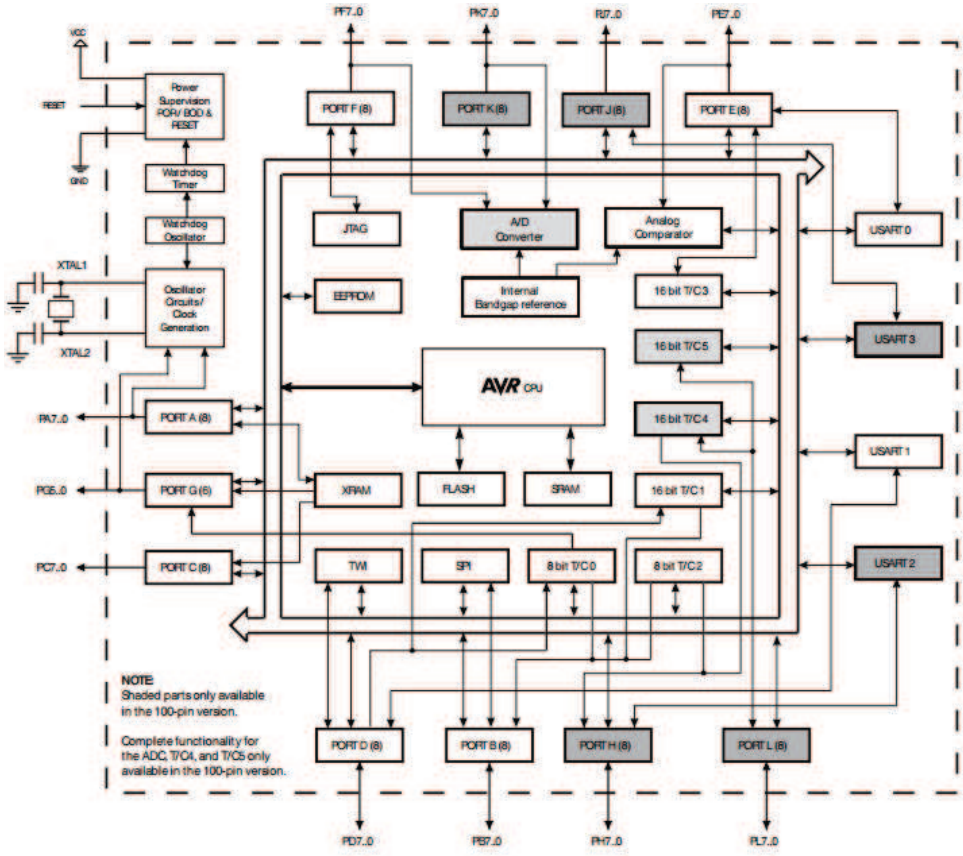

Figura 3.9: Arquitetura interna do microcontrolador ATmega1280

utilizados como saídas PWM, possui 16 entradas analógicas, 4 UARTs, um cristal de 16 MHz, uma conexão USB, um conector de força, um conector ICSP e um botão de reset. A arquitetura interna do ATmega1280 é detalhada na figura 3.9. A placa possui o circuito necessário para o correto funcionamento do microcontrolador. Ela também é compatível com os *shields* XBee disponíveis.

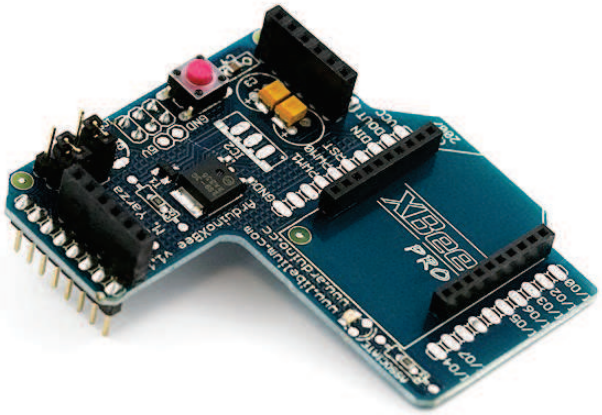

Figura 3.10: Shield XBee

Shields são placas que podem ser encaixadas em cima de uma placa Arduino , estendendo assim suas funcionalidades. Os *shields* seguem a mesma filosofia que as placas originais, sendo fáceis de montar, baratos e *open-hardware*.

Neste trabalho foram usados *shields* XBee (figura 3.10) para facilitar a interface do Arduino com nossos módulos XBee evitando a necessidade do uso de mais circuitos periféricos.

3.6 XBee

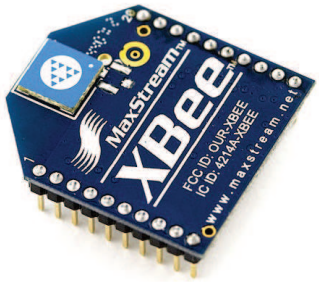

Figura 3.11: XBee

XBee é o nome dado a uma família de módulos de radiofrequência desenvolvidos e distribuídos pela Digi International e por sua filial MaxStream. Esta família tem como características implementar módulos de radiofrequência de baixo custo e baixo consumo para aplicações especificadas pelo protocolo IEEE 802.15.4 e/ou ZigBee™. Adicionalmente, a maioria das famílias XBee integram alguma forma de controle de fluxo, I/Os, conversores A/D e indicadores de potência de sinal. A versão de XBee disponível no LARA é a 802.15.4 Series 1 (figura 3.11, que implementa basicamente um protocolo próprio da Digi baseado em ZigBee™com foco em aplicações de ponto a ponto ou ponto a multiponto.

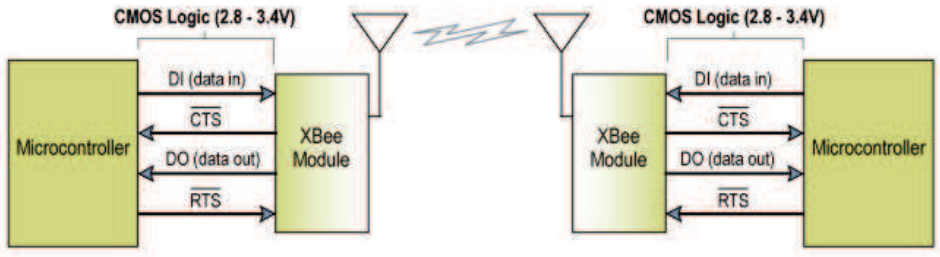

Figura 3.12: Exemplo de comunicação serial

Os módulos XBee se comunicam com seu dispositivo mestre graças a uma comunicação serial assíncrona. Através desta porta serial o modulo pode se comunicar com qualquer UART compatível a nível lógico e de tensão; ou fazendo o uso de um tradutor de nível logico com qualquer tipo de dispositivo serial. Por exemplo fazendo o uso uma placa com tradutor para RS-232 ou USB acoplada a um PC, ou como ilustrado na figura 3.12 diretamente entre microcontroladores.

3.6.1 Controle de Fluxo

Operação no modo 'Bridge' ou Transparente: o XBee por padrão opera no modo chamado de Transparente. Este é o modo que utilizaremos no nosso trabalho. Quando operando neste modo, os módulos funcionam como um substituto a uma linha serial - todo dado UART recebido no pino de Data In é colocado no *buffer* para transmissão RF. Quando dados são recebidos por RF, eles são transmitidos pelo pino de Data Out. Este fluxo de informação é ilustrado na figura 3.13.

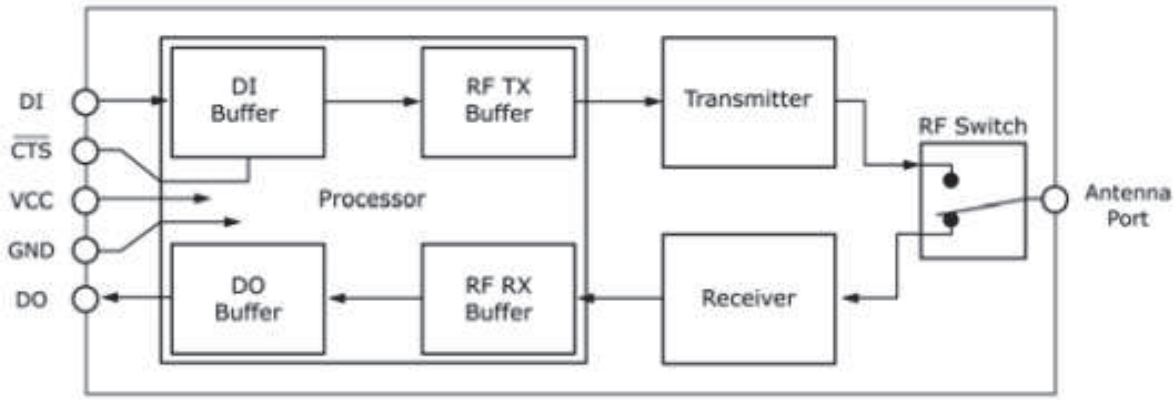

Figura 3.13: Fluxo de dados em uma rede XBee

3.6.2 Modos de operação

A estrutura de controle de fluxo do XBee tem repercussões diretas nos modos de operação deste, especialmente em relação a comunicação. Como temos um *switch* na antena, nota-se que podemos receber, ou transmitir dados, mas nunca fazer os dois de maneira simultânea.

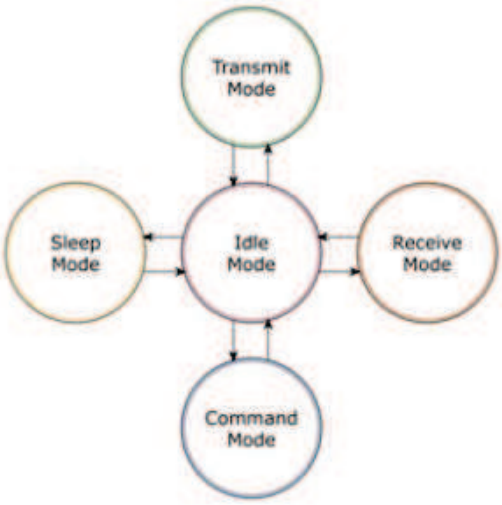

Figura 3.14: Modos de operação

Assim conforme ilustrado na figura 3.14 vemos que o XBee se comporta como uma maquina de estados, que além do *Idle* (Espera) quando não está ocupado, tem mais 4 estados:

- Modo de Transmissão, onde dados são enviados pelo canal de RF
- Modo de Recepção, onde dados são recebidos pelo canal de RF
- Modo Sleep, onde o XBee entre em um estado de baixo consumo de energia, desabilitando as demais funções, útil quando não está sendo utilizado.
- Modo de Comando, modo onde podemos configurar ou ler parâmetros do XBee, tais como endereço de envio, *baud rate*, Frequência de Operação, etc. Neste estados os caracteres recebidos pelo modulo são interpretados como comandos.

3.6.3 Protocolo IEEE 802.15.4

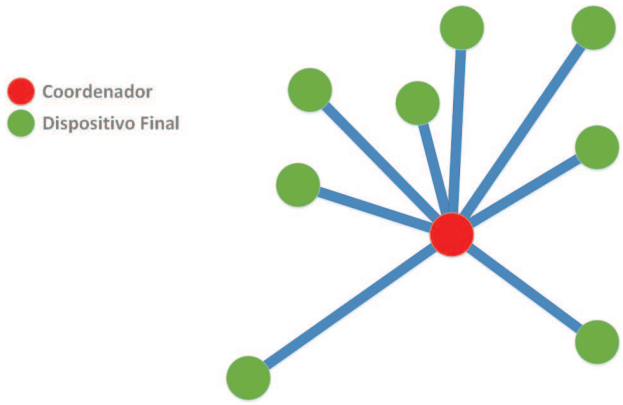

Figura 3.15: Exemplo de Rede com topologia de Ponto a Multiponto (também conhecida como Estrela)

O protocolo IEEE 802.15.4 como diz o nome é mantido pelo IEEE (Institute of Electrical and Electronics Engineers) e é a base para as especificações ZigBee^{TM}, ISA100.11a, WirelessHART, e MiWi que estendem as especificações deste protocolo. O protocolo IEEE 802.15.4 prove as especificações para as camadas inferiores destes outros protocolos de comunicação, garantindo baixo custo, onipresença e baixa velocidade de comunicação da ordem de 250 kbit/s. Como principais características este protocolo possui adequação para aplicações de tempo real pois garante a reserva intervalos de tempo, não colisão via CSMA/CA e suporte integrado para comunicações seguras. Os dispositivos também possuem funções de gerenciamento de energia tais quais detecção de energia e qualidade de ligação que os tornam particularmente interessantes para aplicações de baixo consumo.

O protocolo IEEE 802.15.4 utiliza bandas de operação ISM (*Industrial, Scientific and Medical*) definidas como:

- 868,0-868,6 MHz: na Europa, permite apenas 1 canal de comunicação.

- 902,0-928,0 MHz: na América do Norte e Austrália, permite 30 canais de comunicação desde 2006.
- 2400,0-2483,5 MHz: no resto do Mundo, permite 16 canais de comunicação.

3.6.4 Adaptador CON-USBBEE ROGERCOM

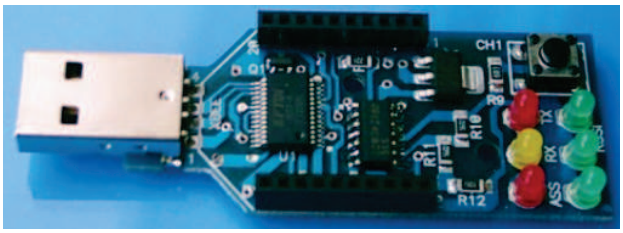

Figura 3.16: Rogercom

Para realizar a comunicação do PC com os XBees utilizados fazemos uso do adaptador CON-USBBEE fabricado pela Rogercom (figura 3.16), empresa brasileira que fornece soluções para aplicações envolvendo módulos XBee/ZigBee. O adaptador CON-USBBEE usa um chip conversor USB/Serial (FTDI Chip) para criar uma porta Serial Virtual no computador e permitir a comunicação deste com o XBee; regulador de tensão para alimentação do XBee; LEDs RSSI que indicam a força do sinal de RF; LEDs indicadores de transmissão recepção (TX e RX); LED indicando a associação a uma rede (ASS) e um botão de *reset* do módulo XBee. Este adaptador é utilizado tanto para a configuração dos parâmetros dos XBees utilizados quanto para o acoplamento da rede XBee ao nosso programa Supervisório.

3.7 Protocolo MODBUS

MODBUS é um protocolo da camada de aplicação (no modelo OSI) projetado para transportar mensagens. Ele fornece comunicação cliente/servidor onde os dispositivos podem estar conectados a diferentes tipos redes e barramentos. É um protocolo praticamente padrão na indústria desde 1979, com uma grande comunidade de suporte e está continuamente em crescimento. Este protocolo é muito bem aceito no contexto industrial devido as seguintes características:

- Foi desenvolvido com foco em aplicações industriais
- É *open-source* e livre de royalties
- Fácil de implantar e manter
- Transmite dados pela rede sem restrições de fornecedor ou fabricante

O protocolo MODBUS permite que muitos dispositivos se comuniquem numa mesma rede, sendo muito utilizado para interligar RTUs a sistemas supervisórios SCADA.

Na sua essência, o protocolo MODBUS é um protocolo de requisição/resposta que oferece serviços e funcionalidades especificadas por um *function code* e atualmente está implementado usando TCP/IP, transmissão serial assíncrona e para redes com sistema de *token* (MODBUS PLUS). Neste trabalho utilizamos o modo de transmissão serial com portas USB e dispositivos de transmissão *wireless*.

3.7.1 Comunicação e Dispositivos

Cada dispositivo que utiliza o protocolo MODBUS recebe um endereço único. Ou seja, cada comando possui no seu cabeçalho o endereço do dispositivo de destino e somente esse dispositivo executara o comando, mesmo que outros dispositivos recebam a mensagem. Além disso, todos os comandos MODBUS possuem informações de verificação da integridade dos dados.

Um quadro (mensagem completa) do protocolo MODBUS pode ter vários formatos, de acordo com a forma de transmissão utilizada. O formato do quadro utilizado para RTUs é descrito na tabela 3.3.

Tabela 3.3: Quadro do protocolo MODBUS

Nome	Comprimento	Função
Início	3,5c	Silencio mínimo para início da transmissão
Endereço	8bits	Endereço da RTU de destino
Função	8bits	Código da função
Dados	N*8bits	Dados da mensagem. N varia com a função
CRC	16bits	Checagem de erros
Fim	3,5c	Silencio mínimo no final da mensagem

3.7.2 Funções Suportadas

A tabela 3.4 representa, de forma resumida, as principais funções oferecidas pela biblioteca MODBUS. Neste trabalho a função que lê registradores do tipo *holding* foi extensivamente utilizada; A RTU faz as medições e as armazena em registradores para posteriormente serem lidas pelo supervisório.

3.7.3 Limitações do protocolo MODBUS

- O protocolo MODBUS, apesar de ser amplamente utilizado pela indústria, apresenta algumas limitações:
- Os tipos de dados são limitados a variáveis simples, por exemplo, inteiros ou números de ponto flutuante
- Não existe padronização para o tipo de registrador lido, por exemplo, não há discriminação se o registrador armazena temperatura, tensão ou outra grandeza

Tabela 3.4: Funções do protocolo MODBUS

Função	Código
Ler Entrada Discreta	2
Ler estado da bobina	1
Escrever estado de uma bobina	5
Escrever estado de várias bobinas	15
Ler registrador de entrada	4
Ler registrador de *holding*	3
Escrever um registrador	6
Escrever vários registradores	16
Ler/Escrever vários registradores	23
Escrever registrador com máscara	22
Ler fila FIFO	24
Ler arquivo	20
Escrever arquivo	21
Ler status de exceção	7
Diagnostico	8
Ler contador de eventos da porta COM	11
Ler registro de eventos da porta COM	12
Reportar identificação do escravo	17
Ler identificação do dispositivo	43
Transportar encapsulamento de interface	43

- Partindo do principio de que MODBUS é um protocolo mestre/escravo, não existe a possibilidade de um dispositivo da rede relatar erros. O dispositivo mestre deve fazer estas requisições para verificar o erro
- MODBUS pode endereçar apenas 247 dispositivos em um único barramento
- O protocolo não oferece proteção contra comandos não autorizados ou interceptação dos dados

3.8 SCADA

SCADA (*supervisory control and data acquisition*) é um tipo de sistema de controle e supervisão de processos, na sua maioria processos industriais (ICS, *industrial control system*). Consiste basicamente de um sistema (ou planta) supervisionado e controlado por um computador. Sistemas SCADA se distinguem de outros sistemas de controle devido a sua capacidade de lidar com processos em larga escala, que pode incluir múltiplas plantas e grandes distancias.

SCADA tem sido usado desde os primeiros sistemas de controle. Os primeiros "SCADA" faziam a aquisição de dados através de painéis indicadores, lâmpadas e registradores. Um operador

supervisionava e controlava manualmente a planta através de botões e outros aparatos, exercendo assim o papel de controlador (figura 3.17). Em alguns lugares ainda é possível encontrar esse tipo de aparato.

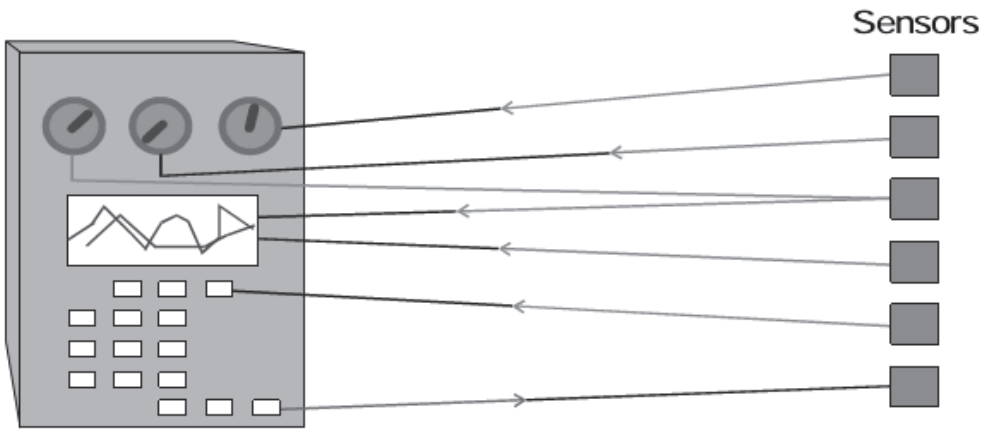

Figura 3.17: SCADA "sensor para painel"

3.8.1 Sistemas SCADA modernos

Em processos industriais modernos, como manufatura em larga escala, indústrias de mineração, serviços públicos e privados, entre outros, a telemetria é geralmente essencial para conectar equipamentos e sistemas separados por longas distancias.

SCADA faz alusão à combinação da telemetria e aquisição de dados, onde é necessário coletar informações, processar, indicar e armazenar tais informações em uma central e, geralmente, enviar as informações de controle para os atuadores necessários. Computares, CPUs, PLCs (*programmable logic controller*) e DCSs (*distributed control systems*) são geralmente empregados para desempenhar tais funções (figura 3.18).

A utilização de sistemas SCADA com PLCs ou DCSs apresentam inúmeras vantagens:

- A central pode armazenar grandes quantidades de dados
- As informações podem ser representadas como o usuário quiser
- Milhares de sensores em uma grande área podem ser conectados ao sistema
- O operador pode incorporar simulações ao sistema
- Vários tipos de dados podem ser coletados dos terminais remotos
- Os dados podem ser visualizados virtualmente de qualquer lugar

Porem, como *trade-off*, tais sistemas também possuem algumas desvantagens:

- Os sistemas são geralmente mais complexos

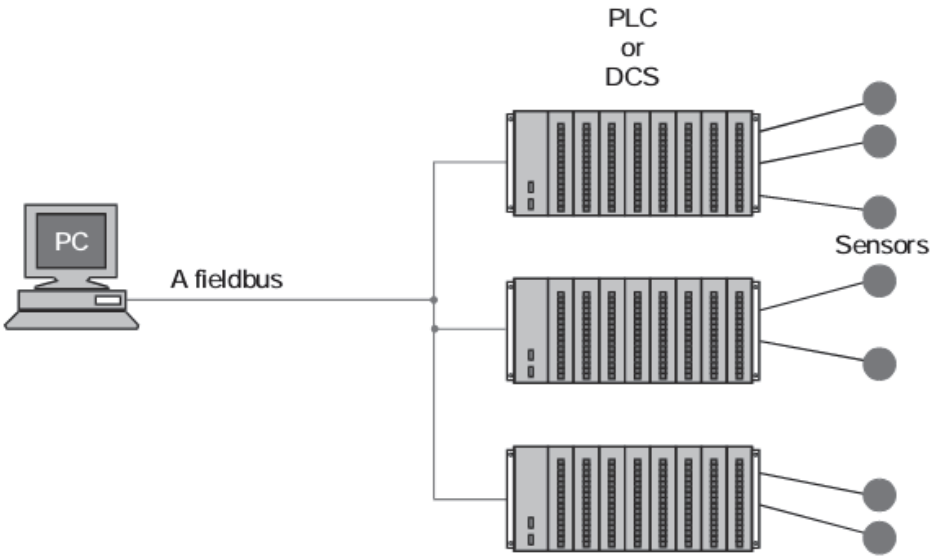

Figura 3.18: Exemplo de arquitetura SCADA

- É necessário conhecimento mais especializado
- Os operadores ainda são limitados ao ”poder de visão” do PLC

3.8.2 Hardware em um sistema SCADA

Um sistema SCADA consiste em um numero de RTUs (*remote terminal station*) coletando dados e enviando a uma central via uma rede de comunicação. A central disponibiliza tais dados ao operador, que pode realizar as tarefas de controle remotamente. A precisão dos dados permite que sejam feitas otimizações no processo, tornando-os assim mais eficientes, rentáveis e seguros.

Em um sistema SCADA mais complexo, existem basicamente cinco níveis de hierarquia:

- Instrumentação, sensores e sistemas de controle
- Atuadores e RTUs
- Sistema de comunicação
- Central
- Departamento comercial de processamento computacional de dados

As RTUs fornecem a interface entre os sensores de cada área.

Os sistemas de comunicação provem um meio de comunicação entre as áreas remotas e a central. Essa comunicação pode ser por fio, fibra ótica, rádio, telefone ou até mesmo satélite. Protocolos e detecção de erros são utilizados para um funcionamento mais eficiente.

A central agrega os dados das várias RTUs e geralmente oferecem ao operador uma interface para mostrar todas as informações necessárias e controlar as diversas plantas. Em sistemas em larga escala, geralmente emprega-se subcentrais.

3.8.3 Software em um sistema SCADA

A parte de software é responsável por todo o gerenciamento, processamento, interface, controle e tomada de decisões de um sistema SCADA.

Os componentes típicos de um sistema SCADA são mostrados na figura 3.19

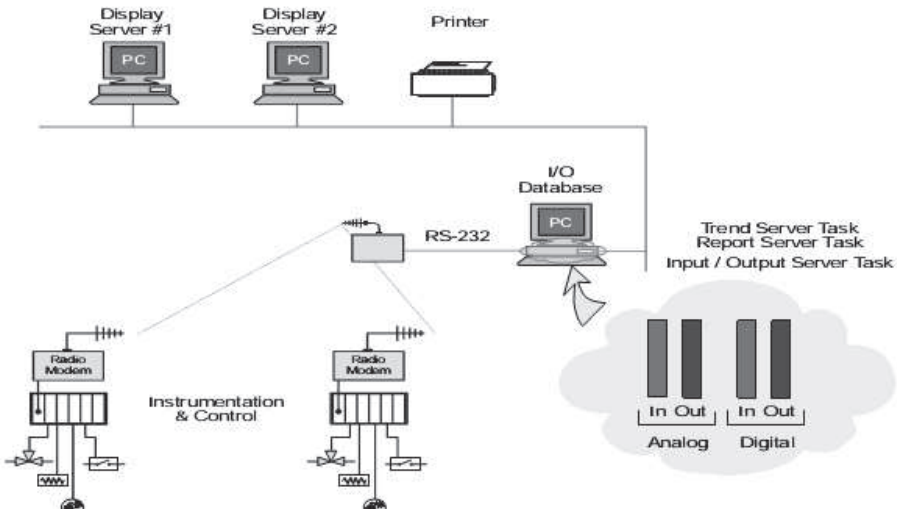

Figura 3.19: Componentes típicos de um sistema SCADA

3.8.4 ScadaBR

O ScadaBR é um sistema supervisório *open-source*, gratuito e multiplataforma (baseado em Java) que pode ser utilizado em aplicações automatizadas de qualquer tipo: indústrias, laboratórios, automação predial, saneamento, sistemas de energia, entre outros. Também serve como uma camada de integração entre softwares de terceiros através de *web-service*. São características relevantes do programa:

- Comunicação com sensores, PLCs e outros dispositivos através dos protocolos OPC, Modbus Serial e TCP, ASCII, DNP3, IEC101, Bacnet, entre outros
- Criação e edição de telas HMI
- Geração de gráficos e relatórios
- Controle de acesso de usuários
- Scripting

- Gerenciamento de alarmes e eventos
- API para Web-Services em geral

Ao executar o ScadaBR, o mesmo pode ser acessado a partir de um navegador de internet. Sua interface é de fácil utilização e já possui ferramentas de visualização de gráficos, variáveis, configurações, alertas, protocolos, estatísticas, entre outros. Após configurar os protocolos de comunicação e aquisição de dados é possível definir variáveis de entrada ou saída, construir uma interface gráfica simples e automatizar processos. Detalhes sobre como instalar, configurar e utilizar o programa podem ser encontrados no manual do mesmo.

Neste trabalho, configuramos a *data-source* (fonte de dados) como uma porta serial USB do computador, que por sua vez está conectada a um dispositivo de transmissão sem fio de dados (mais especificamente uma placa Rogercom com um Xbee). Utilizamos o protocolo Modbus (mais detalhes na seção 3.7) para realização a transmissão de dados entre as RTUs e o computador. Este *data-source* envia requisições de leituras para todas as RTUs a cada 500 ms, sendo que entrega dos dados nem sempre é garantida devidos a limitações da rede *wireless*. Todas as leituras adquiridas são armazenadas em *data-points*.

Um *data-point* é uma coleção de todas as leituras de uma grandeza. Neste trabalho, cada *data-point* representa uma grandeza medida pela placa de medição de energia, além de um *data-point* que é utilizado para verificação de erros de transmissão. São grandezas medidas: potência ativa, potência reativa, tensão e frequência da rede, além das falhas de transmissão do protocolo Modbus.

O monitoramento dos dados é feito de duas maneiras: uma delas é fazendo o uso de uma *watch list*, que é uma lista dinâmica de variáveis e suas respectivas leituras, com informações úteis como histórico de valores e visualização gráfica desse histórico. A outra maneira é criar uma representação gráfica desses pontos, montando uma interface gráfica analógica a um painel de controle, com gráficos, imagem de fundo, botões e outras funcionalidades.

Capítulo 4

Desenvolvimento

Neste capítulo detalharemos todas as etapas do desenvolvimento do hardware, desde testes com protoboard até o projeto da PCB. Detalharemos também as estruturas dos softwares desenvolvidos com o auxílio de fluxogramas.

4.1 Projeto de Hardware

Nosso módulo de medição consiste em três sub circuitos: circuito de alimentação, circuito de medição e circuito de acoplamento conforme ilustrado na figura 3.1. Os parâmetros de entrada a serem seguidos são de 230 V e 40 A. A seguir vamos detalhar os esquemáticos dos circuitos, o projeto da PCB e ilustrar com algumas fotos os circuitos montados para a realização de testes.

4.1.1 Alimentação

Este é nosso circuito de alimentação, consiste de uma fonte simétrica de 2,5V -2,5V onde a saída é a queda de tensão em diodos *zener*. Ela é ligada diretamente na linha Neutral da rede elétrica, seu projeto foi baseado no circuito de aplicação do SA9903B.

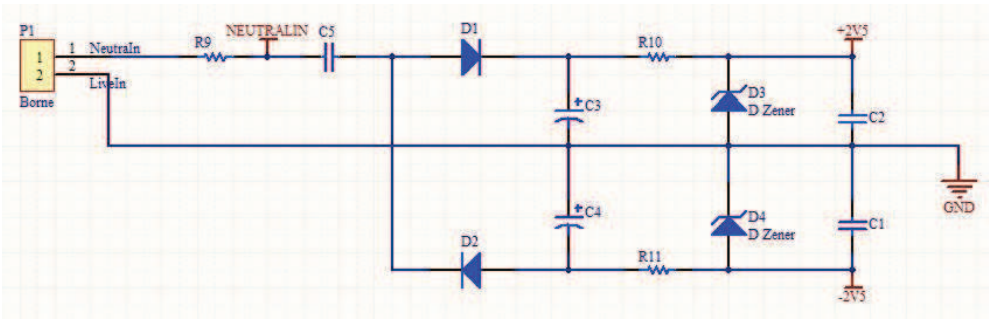

Figura 4.1: Fonte de alimentação

Notem que como a fonte de alimentação é ligada antes do resistor *shunt*, logo do circuito de medição, que esta não interfere na medida do consumo de energia.

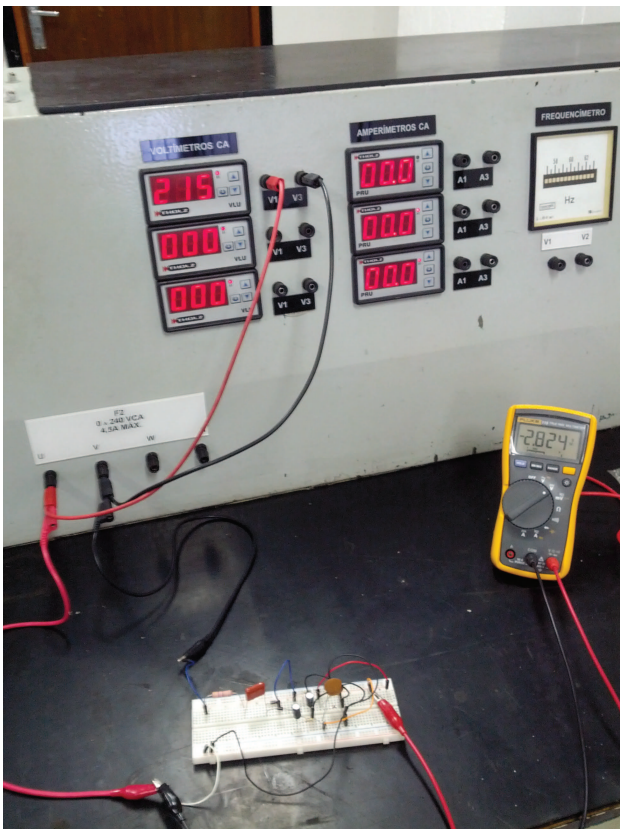

Figura 4.2: Fonte de alimentação em funcionamento. A bancada está funcionando como a rede (220 V) e a leitura no multímetro é a tensão de saída da fonte.

4.1.2 Medição

A figura 4.3 apresenta nosso circuito de medição, ele é responsável por fornecer todos os dados necessários sobre a rede a qual está acoplado. Seus dados são fornecidos pelo SA9903B em sua saída SPI no modo escravo. O circuito também foi baseado no circuito de aplicação do SA9903B. Respeitando as características de funcionamento deste *microchip*.

4.1.3 Cálculo dos Valores das Resistências

Deseja-se que o nosso circuito meça, para tensões de 230 V_{RMS}, correntes de até 40 A_{RMS}. Os resistores de entrada dos sensores devem ser calculados de forma que tais sinais de entrada sejam compatíveis com o SA9903B.

4.1.3.1 Entrada dos Sensores de Corrente (IIP e IIN)

Em condições nominais temos que a queda de tensão no *shunt* é de 200 mV_{RMS}. Para tal queda de tensão, requere-se uma corrente de 16 μA_{RMS} nas entradas do sensor de corrente. Logo, temos que

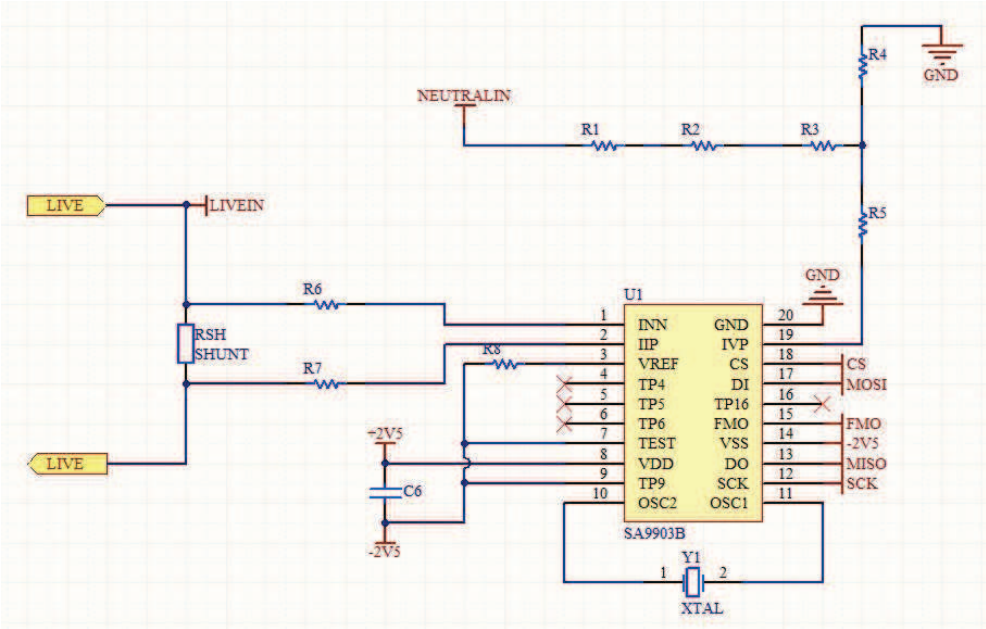

Figura 4.3: Circuito de medição de tensão, corrente e potência, utilizando um SA9903B

$$0,2 = 16 * 10^{-6} * 2R$$

$$R = 6250\ \Omega$$

No mercado temos resistores de 6,2 $k\Omega$ (erro de -0,8%), valor escolhido para o projeto. No circuito, tais resistores são o R_6 e R_7 (figura 4.3).

4.1.3.2 Entrada do Sensor de Tensão (IVP)

Em condições nominais, requere-se que a tensão no pino IVP seja de 14 V_{RMS} e a corrente de $14\mu A_{RMS}$. Foi utilizado um divisor de tensão para tal finalidade, como se segue:

$$230\frac{R_2}{R_1 + R_2} = 14$$

Escolhendo $R_2 = 24\ k\Omega$, tem-se que $R_1 = 370,286\ k\Omega$.

O valor para R_1 foi aproximado com 1 resistor de 150 $k\Omega$ em série com 2 resistores de 110 $k\Omega$, totalizando 370 $k\Omega$ (erro insignificante). Tais resistores, no circuito (figura 4.3), são os R_1 (150 $k\Omega$), R_2, R_3 (110 $k\Omega$) e R_4(24 $k\Omega$).

4.1.4 Acoplamento

O nosso circuito de medição opera com faixas de tensão de -2,5 V a +2,5 V. Entretanto, os dispositivos utilizados como mestre operam em outras faixas. O Arduino, por exemplo, opera

entre 0 V e 5,0 V. Precisamos então fazer uma tradução destes níveis lógicos para garantir o funcionamento da comunicação SPI. Além disto ao utilizarmos optoacopladores na interface do circuito promovemos a proteção do módulo microcontrolador com um isolamento óptico, separando os circuitos ligados na rede elétrica do circuito de 5,0 V do módulo microcontrolador.

4.1.4.1 Entrada

Para acoplamentos de entrada de sinal no circuito de medição é utilizada a seguinte configuração, que indica o pino MOSI mas que é válida também para CS e SCK.

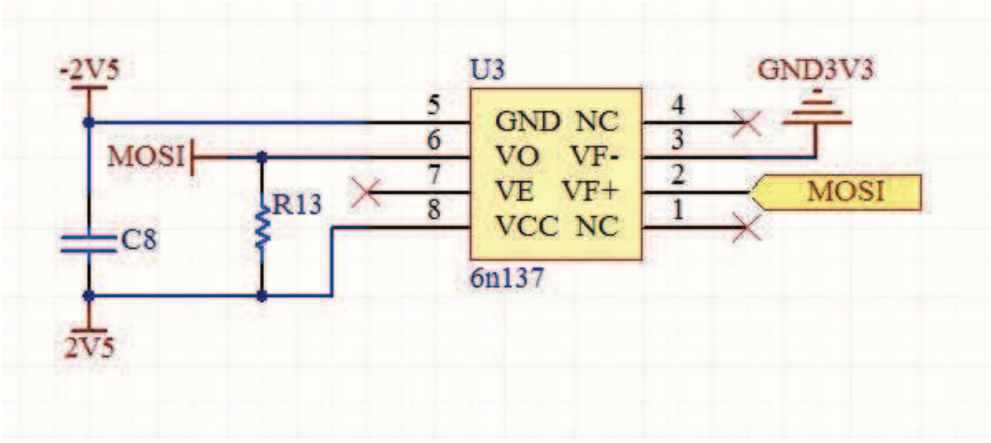

Figura 4.4: Ligação entre a saída de dados do módulo *wireless* com a entrada do SA9903B

4.1.4.2 Saída

Para acoplamentos de saída de sinal no circuito de medição é usada a seguinte configuração, ilustrada na figura 4.5 para o pino MISO mas que é válida também para a saída FMO.

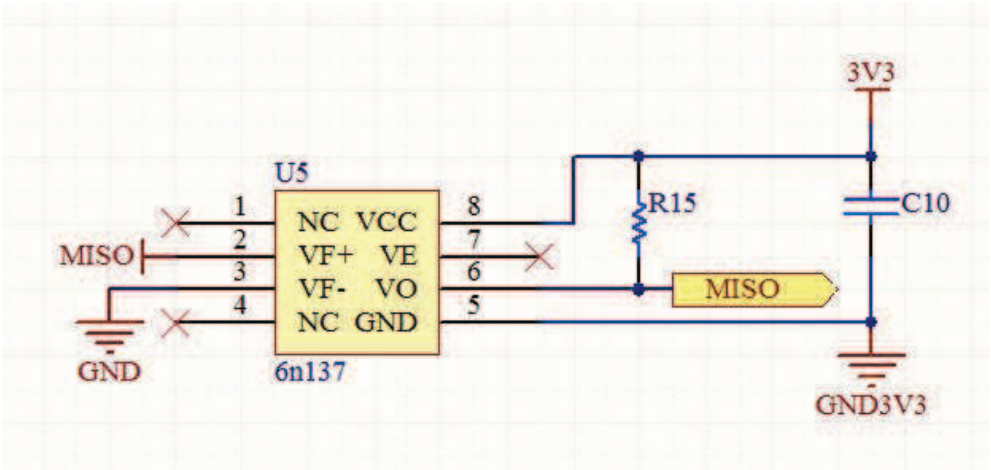

Figura 4.5: Ligação entre a saída de dados do SA9903B com a entrada do módulo *wireless*

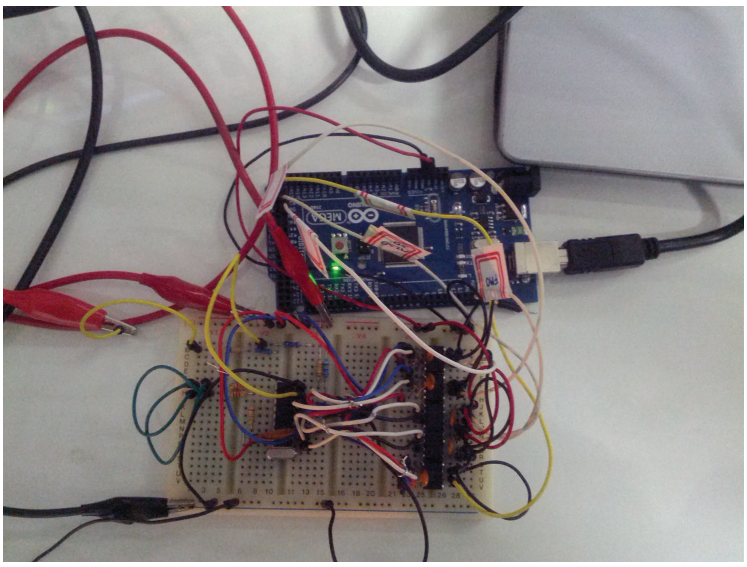

Figura 4.6: Detalhe da montagem do circuito de medição. A placa superior é o um Arduino (controlador), que serve de interface entre o SA9903B, e a placa inferior é o módulo de medição.

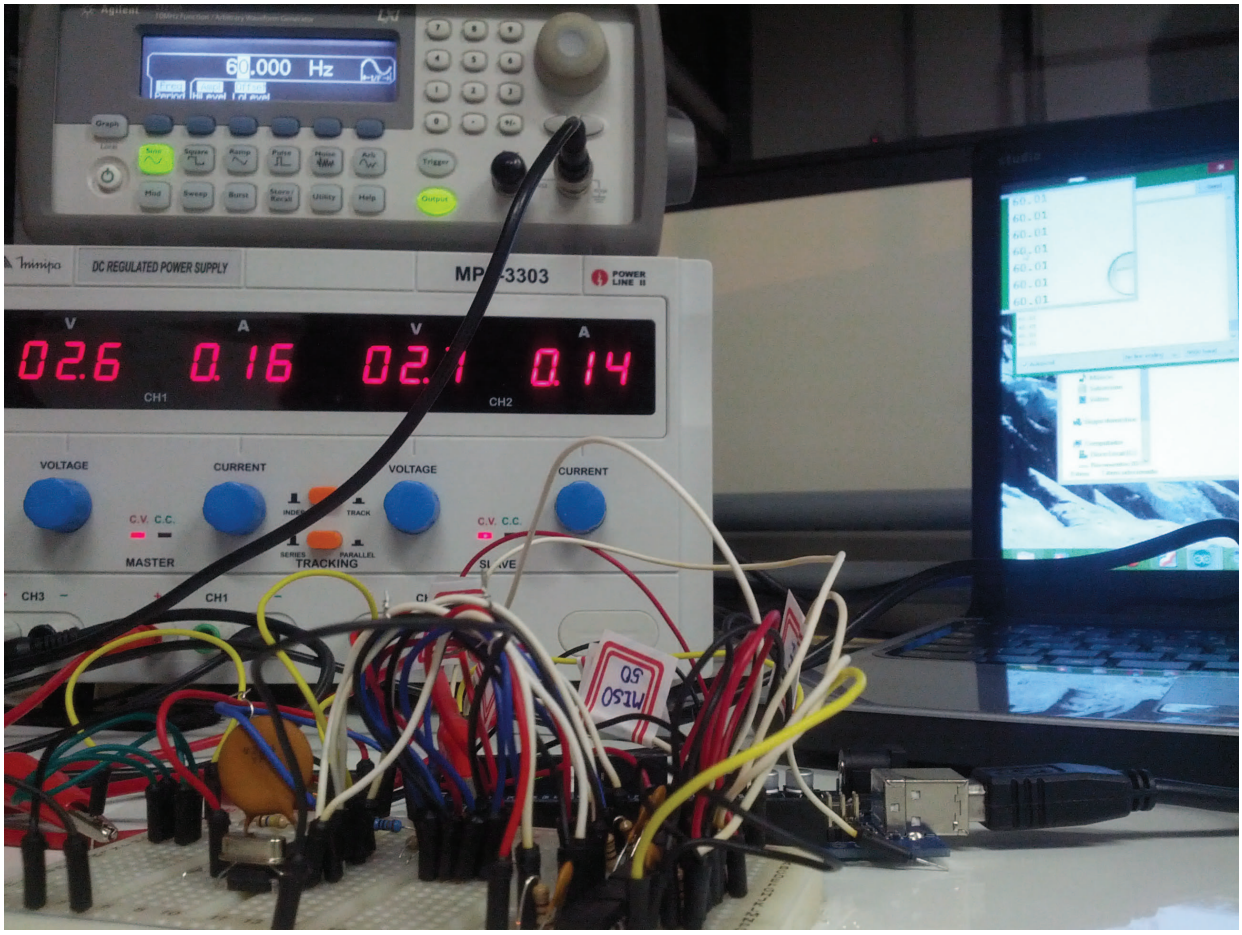

Figura 4.7: Sistema completo em funcionamento. Na foto podemos ver o gerador de funções (à esquerda) gerando uma onda senoidal de 60 Hz e a leitura desta mesma frequência sendo feita no PC, utilizando o SA9903B.

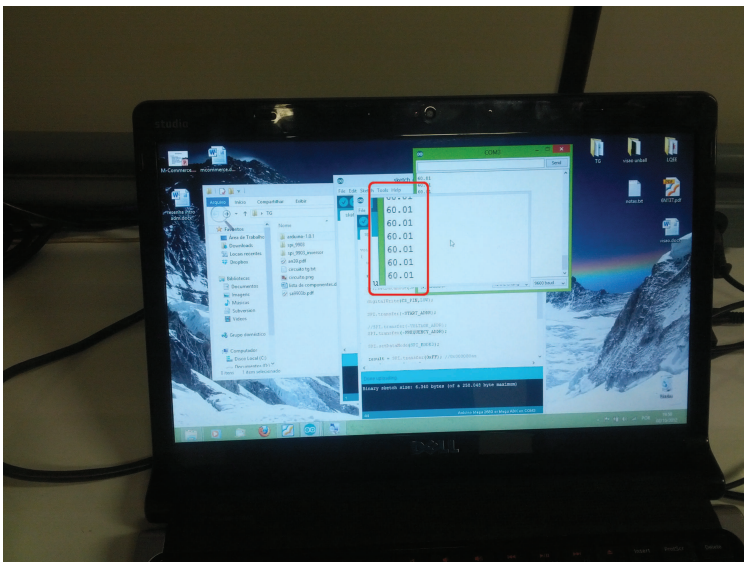

Figura 4.8: Detalhe da frequência medida: 60,01Hz

4.1.5 Modulo de Processamento e Transmissão

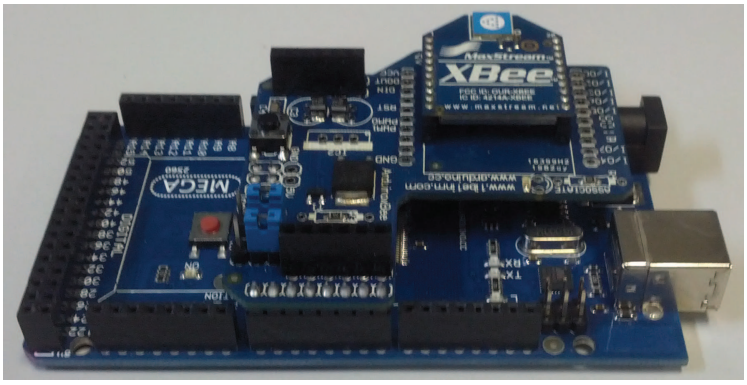

Figura 4.9: Montagem do Arduino com o *shield* e XBee

O módulo que faz o processamento dos dados consiste em um Arduino, um *shield* XBee e um XBee. Os pinos que devem ser conectados a PCB para a comunicação serial e alimentação são determinados em *software*.

Neste trabalho foram definidos como ligados nos seguintes pinos do Arduino:

- 3v3: Pino de saida 5
- GND: Pino de GND
- MISO: Pino 50
- MOSI: Pino 51
- SCK: Pino 52

- CS: Pino 48
- FMO: Pino 49

Não existe trabalho de *hardware* necessário neste módulo, apenas de configuração dos rádios e programação do Arduino. Estas etapas serão detalhadas na próxima sessão.

4.1.6 Projeto no Altium

O esquemático do módulo foi desenvolvido no Altium Designer Suite, um programa CAD próprio para o *design* de circuitos eletrônicos. O esquemático completo do circuito se encontra no Anexo II. Neste mesmo *software* foram desenvolvidos os *footprints* e trilhas que compõem nossa PCB. Os componentes utilizados na placa se encontram detalhados no Anexo I. Todos estes arquivos se encontram no CD (Anexo IV) entregue em conjunto com o trabalho e podem servir de base para trabalhos futuros. Os arquivos de manufatura gerados pelo programa foram enviados para uma empresa chinesa (`www.seeedstudio.com`) que confecciona placas eletrônicas a um preço competitivo.

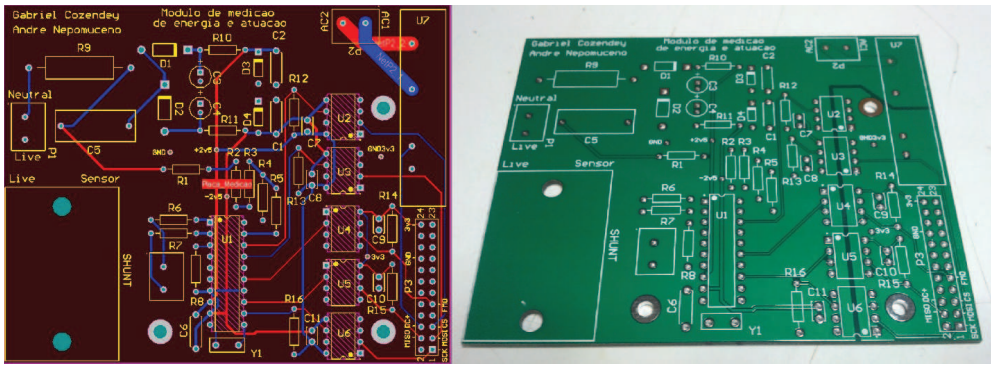

Figura 4.10: Projeto da PCB e placa fabricada

4.1.7 Testes, Soldagem e Montagem

As placas recebidas foram soldadas, montadas e testadas no LARA pelos próprios autores. Também foi realizada a troca das tomadas dos ar condicionados para o padrão novo, já que não se encontram tomadas do antigo padrão no comércio.

Ilustramos na figura 4.11 um dos testes realizados com a placa parcialmente soldada. No multímetro lemos o valor de saída de alimentação com o circuito em operação.

A figura 4.12 é uma foto da nossa placa em sua versão final, já com as tomadas de entrada e saída e o resistor shunt.

Figura 4.11: Testes realizados com a fonte projetada

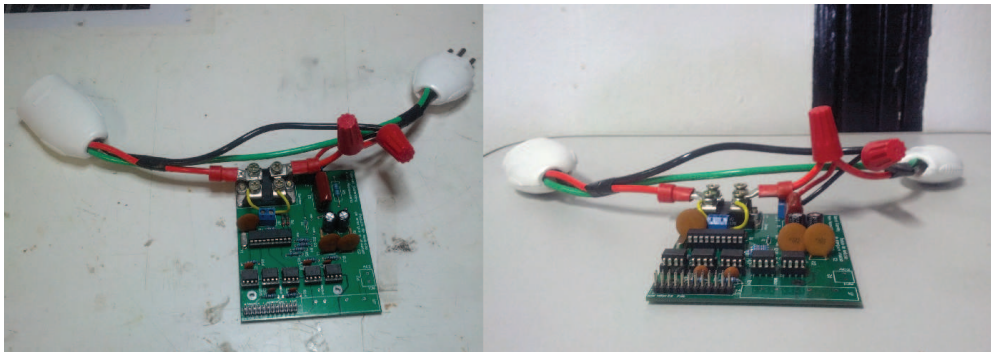

Figura 4.12: Versão final da placa soldada e montada

4.2 Projeto de Software

O software está dividido em 4 partes básicas: biblioteca MODBUS (em detalhes na seção 3.7), biblioteca de leitura do CI SA9903B, programa principal que é executado no microcontrolador e o supervisório ScadaBR (em detalhes na seção 3.8). O diagrama da figura 4.13 mostra como é a estrutura básica do software. Todos os códigos foram feitos utilizando as linguagens de programação C e C++.

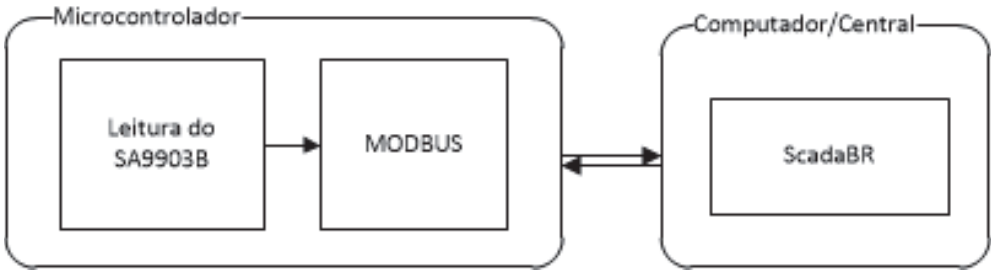

Figura 4.13: Arquitetura geral do software de aquisição e supervisão

4.2.1 Biblioteca SA9903B.h

Para que haja comunicação entre o microcontrolador e o SA9903B foi necessário programar uma biblioteca de comunicação que acesse os registradores do SA9903B e traduza os valores lidos. Além disso, o calculo da potência ativa e reativa é de certa forma delicado, pois exige medições de intervalos de tempo e verificação de *overflow* dos registradores. Basicamente o que o programa faz é acessar os registradores do SA9903B via protocolo SPI, armazenar os valores lidos e traduzi-los para valores de tensão, frequência e potência.

Antes dos valores serem lidos, a função SA9903B::Setup() deve ser chamada. Essa função inicializa e configura pinos de entrada/saída do microcontrolador e inicializa o protocolo SPI com as configurações adequadas. Essa etapa é essencial para o funcionamento do sistema.

Com a biblioteca inicializada, as leituras são feitas através da função SA9903B::Run(), que deve ser executada em um *loop*. Esta função realiza todos os procedimentos de comunicação e leitura dos registradores. A cada passagem de tensão da rede por zero (meio ciclo de onda), todos os registradores são lidos e o intervalo de tempo entre essas leituras é armazenado para cálculos posteriores de potência. Quando a leitura é feita com sucesso, ao chamar a função SA9903B::GetUpdatedStatus() a mesma retorna verdadeiro, indicando que os valores das leituras foram atualizados e estão prontos para serem lidos.

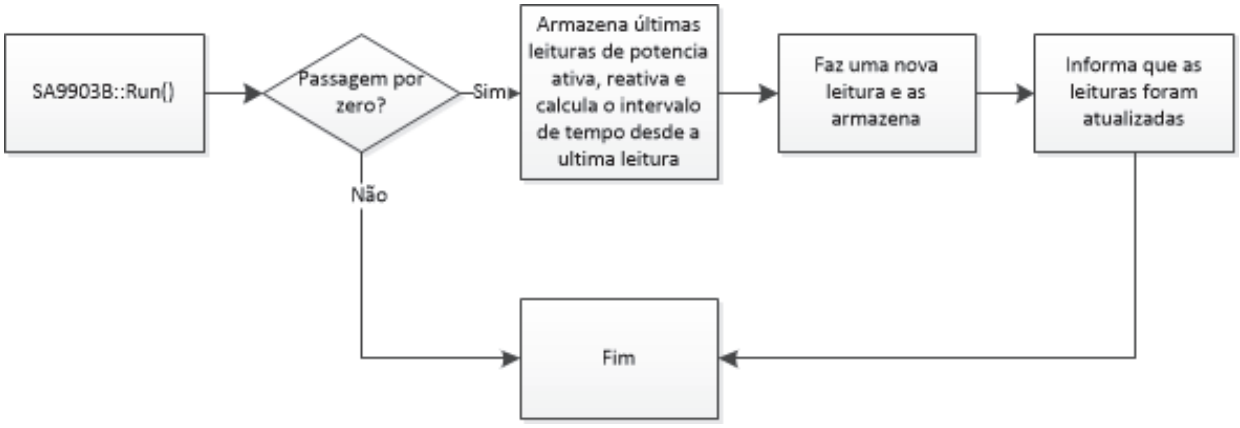

Figura 4.14: Fluxograma da biblioteca SA9903B.h

Para acessar os valores lidos e traduzidos, foram implementadas 4 funções, de acordo com a tabela a seguir:

Tabela 4.1: Funções de aquisição da biblioteca SA9903B.h

Função	Descrição
SA9903B::GetVoltage()	Ultimo valor de tensão RMS lido [V]
SA9903B::GetFrequency()	Ultimo valor de frequência lido [Hz]
SA9903B::GetActive()	Ultimo valor de potência ativa lido [W]
SA9903B::GetReactive()	Ultimo valor de potência reativa lido [VAr]

4.2.2 Biblioteca SimpleModbusSlave.h

Para que o protocolo MODBUS seja utilizado pelo microcontrolador, foi utilizada a biblioteca simple-modbus, que pode ser encontrada em `https://code.google.com/p/simple-modbus/`. Mais especificamente, utilizamos apenas a biblioteca para o terminal escravo, a SimpleModbusSlave.h (inclusa no pacote simple-modbus).

A biblioteca é muito simples, possui apenas as funções 3 (ler registrador de *holding*) e 16 (escrever vários registradores) do protocolo, onde apenas a 3 é utilizada neste trabalho.

Na sua implementação, possui duas funções: modbus_configure() e modbus_update(). A função modbus_configure() inicializa a biblioteca e configura parâmetros como a taxa de bits (ou *baud rate*), o endereço do escravo e o tamanho do registrador de *holding*. Essa função é geralmente chamada apenas uma vez no inicio do programa.

A função modbus_update() atualiza os valores dos registradores e faz a verificação de erros na comunicação, como perdas de pacotes ou falha de CRC. Essa função é chamada toda vez que as leituras do SA9903B são atualizadas.

4.2.3 Estrutura geral do programa no microcontrolador

O programa que é executado no microcontrolador roda em um ciclo infinito. Primeiramente ele inicia todas as bibliotecas necessárias e depois entra num *loop*. A cada ciclo a leitura de dados é executada e, caso haja atualização, os registradores do MODBUS são atualizados.

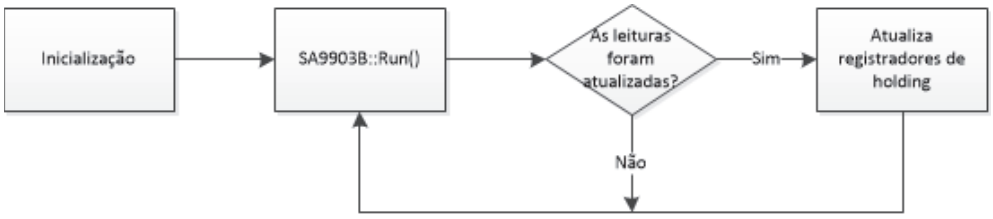

Figura 4.15: Estrutura geral do programa que é executado pelo microcontrolador

4.3 Topologia

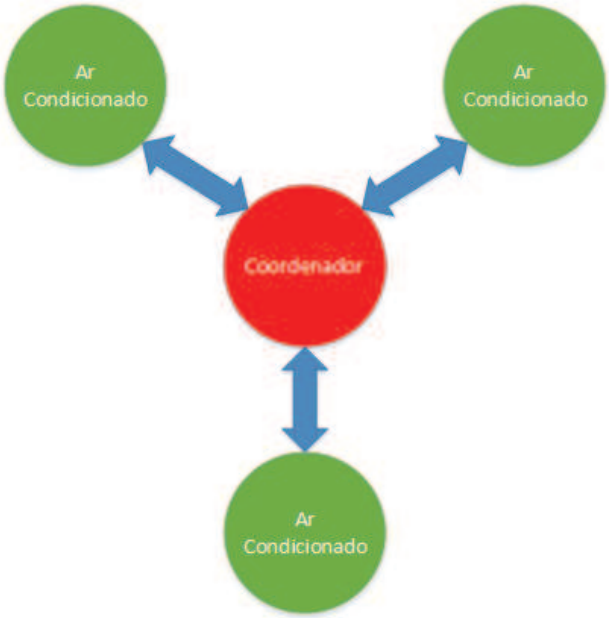

Figura 4.16: Topologia em estrela utilizada no monitoramento dos aparelhos de ar condicionado

A rede XBee utilizada tem a topologia de Estrela, cada um dos nós sensores posicionados em 3 aparelhos de ar-condicionado são os nós finais e o nó acoplado ao PC é nosso coordenador. Para determinarmos a topologia de uma rede XBee precisamos configurar os módulos individualmente para apresentarem o comportamento desejado. No caso os nós devem se comunicar apenas com o coordenador, e o coordenador com todos os nós. Para isto usamos o X-CTU, um software distribuído pela própria Digi que facilita a configuração dos módulos XBee (que também podem ser configurados via comandos AT pela porta serial) provendo uma interface gráfica de fácil utilização.

4.3.1 X-CTU

O X-CTU é composto por 4 abas:

- *PC Settings Tab*: Onde configuramos os parâmetros da comunicação do PC com o XBee a ser configurado tais como *baud rate* e Porta Com a ser utilizada.
- *Range Test Tab*: Permite realizar testes de comunicação entre dois rádios.
- *Terminal*: Permite o acesso a portas COM do PC com um programa emulador de Terminal. Podemos configurar os XBees fazendo uso de comandos AT por esta interface ou simplesmente ler e enviar dados pela porta serial.
- *Modem Configuration*: Esta é a aba mais importante, já que permite a configuração dos módulos de maneira simples e eficiente, além de nos dar a informação de qual é a configuração atual de cada Módulo.

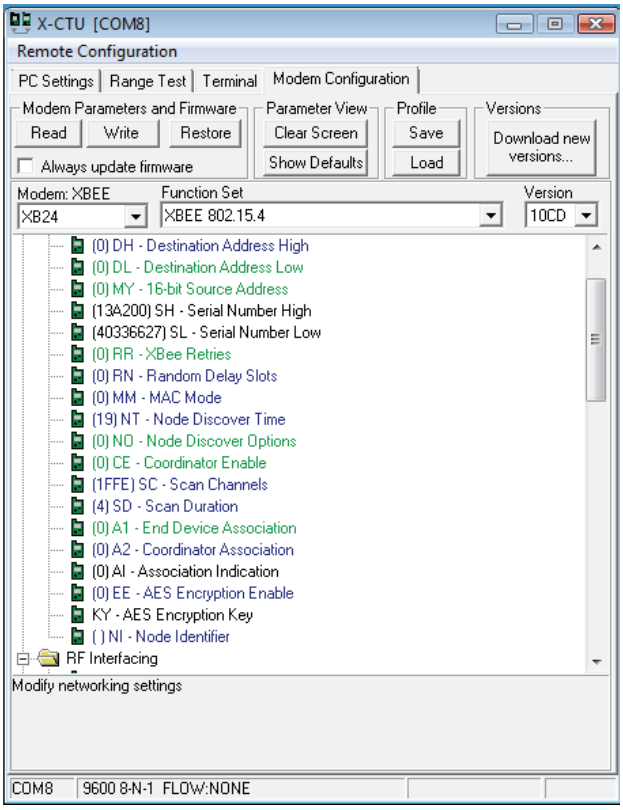

Figura 4.17: Interface gráfico do X-CTU

4.3.2 Configurações dos Módulos

A seguir está o detalhamento dos parâmetros relevantes para montarmos uma rede com a topologia desejada. Os demais parâmetros podem ser mantidos com os valores padrão.

- *Channel* (CH): Determina a frequência de transmissão do modulo XBee segundo a seguinte formula:

  $$2.405 + (CH - 11) * 5 \text{ (em MHz)}$$

  Os módulos precisam ter o mesmo CH para se comunicar entre si. Utilizamos o CH padrão.

- Pan ID (ID): Determina qual em qual PAN (*Personal Area Network*) ID o módulo XBee atua. Os módulos precisam ter a mesma PAN ID para poderem se comunicar entre si. No nosso projeto usamos ID=2013 em todos os módulos.

- *Source Adress* (MY): Comando MY, corresponde ao endereço de 16 Bits do XBee. Na nossa rede MY=0x01 para o Coordenador e já que o Modbus faz o endereçamento podemos usar o mesmo MY=0x02 para todos os Nós.

- *Destination Adress* (DH e DL): Comandos DH e DL, definem o endereço de destino usado para as transmissões do modulo. Na nossa rede DL=0x00000001 para os Nós e DL= 0x0000FFFF (Modo *Broadcast*) ou DL=0x00000002 para o Coordenador.

- *Sleep Mode* (SM): Permite determinar o modo de *Sleep* utilizado pelo módulo. Para nosso projeto é importante que os módulos não entrem em Sleep, já que atualizamos os valores em tempo real.
- *Baud Rate* (BD): Este comando determina a *baud rate* utilizada na comunicação entre o XBee e seu mestre. A *baud rate* é a frequência do sinal utilizado na comunicação, é importante que seja a mesma entre mestre e escravo. No nosso projeto utilizamos BD= 6 que corresponde a 57600 bps.
- API *Enable* (AP): Este parâmetro determina o uso ou não do modo de API. No nosso caso este modo não é desejado. É importante que AP=0 habilitando o modo de operação transparente.

Capítulo 5

Resultados

Neste capitulo apresentaremos os resultados obtidos em dois experimentos. O primeiro com apenas um módulo acoplado a um secador de cabelo e o segundo com três módulos acoplados a três aparelhos de ar condicionado do laboratório.

5.1 Testes com secador de cabelo

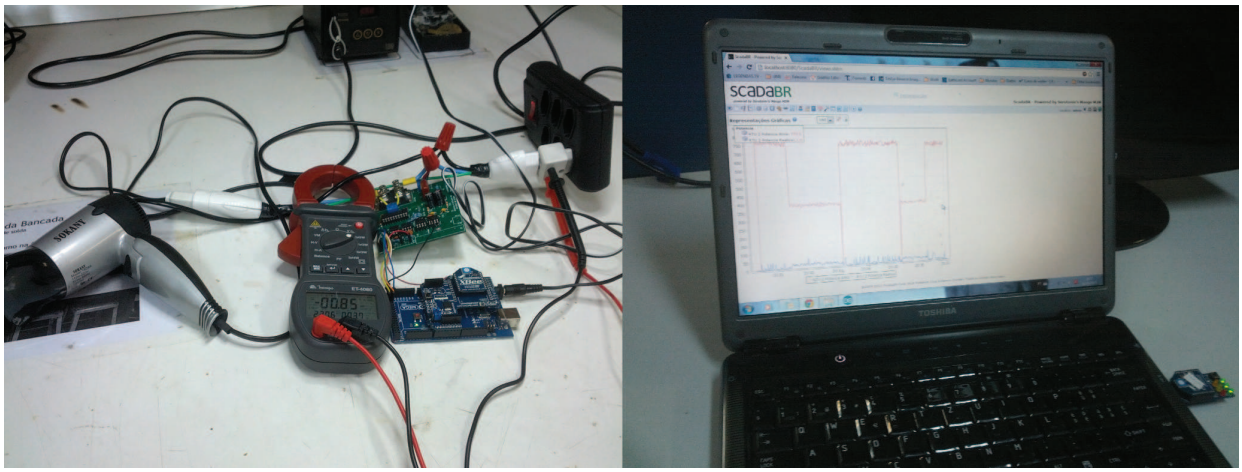

Figura 5.1: Montagem para teste com secador e PC

Após verificar todo o hardware, comunicação e software, fizemos o seguinte experimento para verificar o funcionamento do sistema: utilizando um secador de cabelo de comportamento conhecido em todos os seus modos de operação e um wattímetro, configuramos o sistema supervisório para coletar dados de consumo de energia e fizemos a montagem da figura 5.2.

Estes testes foram realizados para testar a resposta do sistema a diferentes entradas em uma aplicação real. O fato de podermos controlar a potência do secador permite sabermos se o comportamento das medições é condizente com o esperado. De fato, como se observa na figura 5.3 observamos a medida de consumo 0 W com o secador desligado, em torno de 770 W na potência 2 e 390 W na potência 1. Também observamos picos para 0 W quando não mudamos de uma potência para outra de maneira rápida o suficiente.

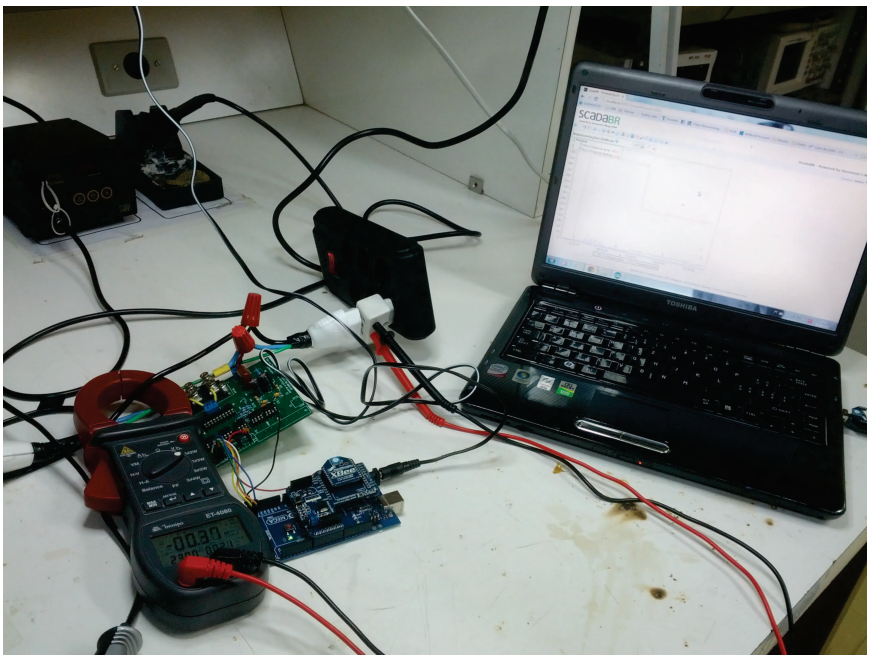

Figura 5.2: Secador em modo de baixo consumo

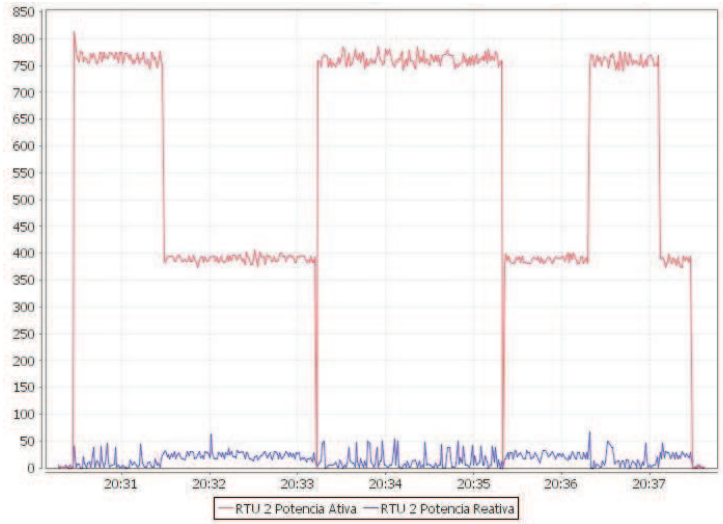

Figura 5.3: Gráficos da potência ativa e reativa com o secador nos modos e baixo e alto consumo

5.2 Testes com ar condicionados

Figura 5.4: Módulos de medição ligados aos seus respectivos ar condicionados

Após a conclusão dos testes com o secador de cabelo passamos a realizar testes com os aparelhos de ar condicionado do LARA. Primeiramente com apenas um aparelho e em seguida com dois e três aparelhos. A seguir vamos apresentar os resultados obtidos com 3 aparelhos que é o objetivo deste trabalho. É importante notar que ao aumentarmos o numero de ar condicionados, e assim de RTUs, foi observada uma perda crescente de pacotes de dados na transmissão. Estas perdas são consideradas normais e são devidas a conflitos temporais e ao meio de transmissão. Na figura abaixo observamos três módulos acoplados a três ar condicionados do LARA. E na figura 5.5 observamos as medições feitas em tempo real sendo exibido no ScadaBR.

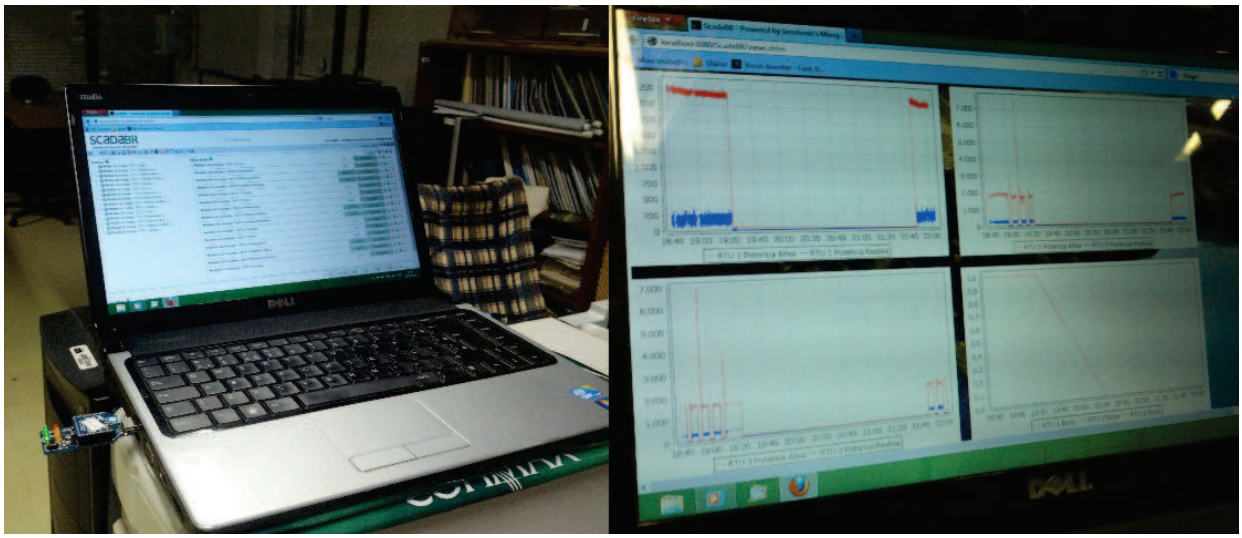

Figura 5.5: PC central com módulo XBee acoplado e detalhe dos gráficos em tempo real do ScadaBR

A figura 5.6 é uma captura de tela do ScadaBR mostrando todas as três RTUs em operação. Nesta tela podemos acompanhar os valores instantâneos medidos pelos três módulos.

Já nesta próxima captura de tela exibida na figura 5.7 mostramos os gráficos que são plotados em tempo real pelo programa supervisório. Temos assim nesta aba do programa informação temporal sobre as variações dos parâmetros de energia monitorados por nossa rede.

Na figura 5.8 podemos acompanhar o comportamento de um ar condicionado em detalhe. São observados facilmente os períodos em que o compressor está ligado e períodos em que apenas o ventilador está operando. Também é observado um pico no consumo de energia no momento do acionamento dos compressores do ar condicionado.

5.3 Considerações sobre Calibração e Precisão do Sistema

A primeira anomalia na medição constatada foi ao medirmos a tensão. Queremos que a tensão no IPV (sessão 4.1.3.2) seja de 14 V, no entanto medimos uma tensão de aproximadamente 13,5 V. Isto de alguma forma se reflete ao lermos o registrador de tensão que apontava uma tensão de 260 V enquanto mediamos 230 V com um multimetro. Este erro foi corrigido via software

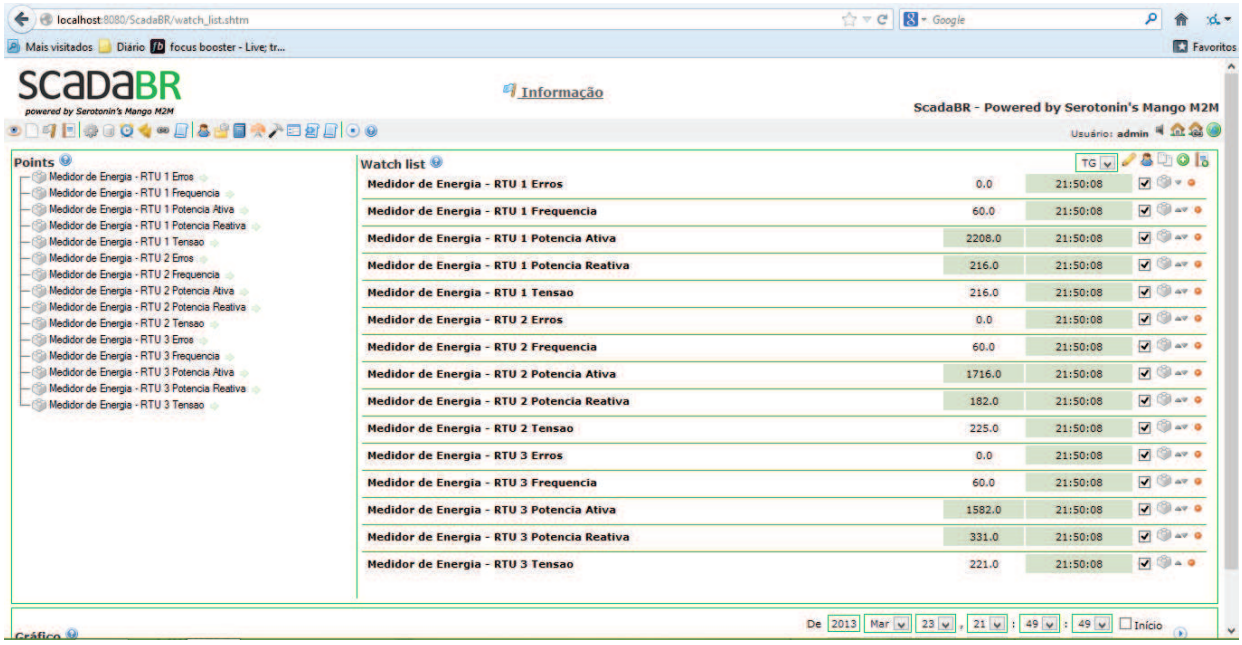

Figura 5.6: Tela de exibição dos valores instantâneos no ScadaBR

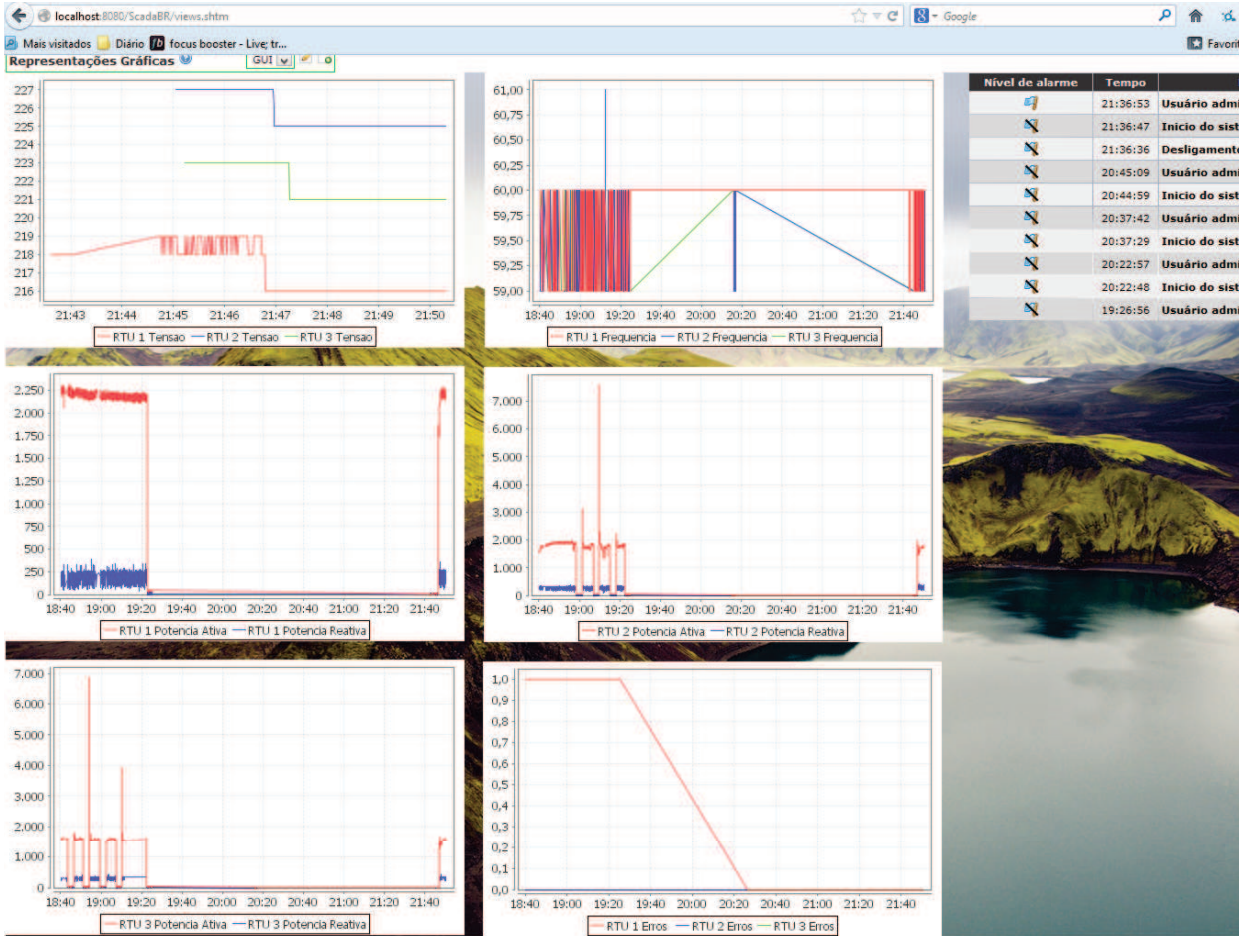

Figura 5.7: Tela de exibição dos gráficos no ScadaBR

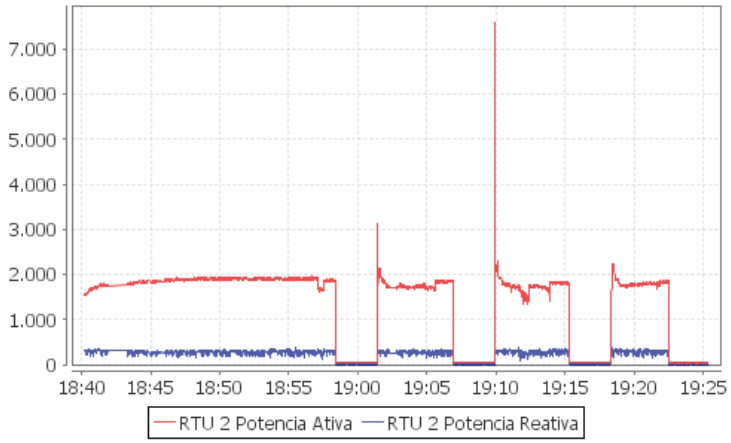

Figura 5.8: Detalhe do pico de consumo no momento do acionamento do compressor

ao introduzirmos um fator de correção para a tensão nominal. Acreditamos que esta não seja a solução ideal, mas foi adequada para o propósito deste trabalho.

Já no experimento com secador de cabelo no modo de baixa potência, o wattímetro leu um consumo instantâneo de 0,37 kW (figura 5.2), enquanto nosso sistema mediu aproximadamente 390 W (desvio de 5%). No modo de alta potência, o wattímetro leu 0,85 kW (figura 5.1) e o nosso sistema leu aproximadamente 770 W, o que resulta num desvio de aproximadamente 10%. Os consumos medidos podem ser verificados no gráfico da figura 5.3.

Estes resultados foram considerados satisfatórios pois, apesar da relativamente baixa acurácia em relação ao wattímetro, as leituras são proporcionais e condizentes com os resultados esperados.

Teoricamente (e também observado na prática) este erro está associado à tolerâncias na malha de resistores (que condicionam os sinais de tensão e corrente) e aparenta ter um comportamento linear, o que pode ser resolvido via calibração dos parâmetros utilizados nos cálculos das leituras no software. Além disso, existe o questionamento quanto a calibração e precisão do wattímetro.

Capítulo 6

Conclusão

O modulo desenvolvido se mostrou plenamente funcional e satisfez seu propósito inicial. Conseguimos efetuar a medição *wireless* dos três aparelhos de ar condicionado do LARA em tempo real e apresentar os resultados em nosso programa supervisório, o ScadaBR. Este trabalho já abre as portas para a implementação de uma rede completa com medição e atuação em tempo real no ambiente do laboratório, e estudos sobre essa mesma rede. No entanto apesar de ter atingido as expectativas do projeto, o modulo desenvolvido deve ser considerado como um protótipo. Por se tratar de sua primeira iteração foram constatadas diversas melhorias diretas que podem ser implementadas como detalhado na sessão 6.1 de *bugs* conhecidos. Na sessão de Trabalhos Futuros expomos idéias de projetos para os quais esperamos poder contribuir com este trabalho.

6.1 Bugs Conhecidos

Fatidicamente como em todo projeto de *hardware* existem erros ou *bugs* nas primeiras iterações do projeto e o nosso não é isento deles. A seguir vamos detalhar os *bugs* conhecidos e também propor melhorias para áreas criticas do circuito.

- Furos de dimensão errada: Os furos no *footprints* dos Bornes P1 de entrada do circuito de alimentação, P2 para o relé de estado sólido e o Borne acoplado ao resistor *shunt*, possuem furos menores que as pernas destes, sendo necessário limar-los para realizar o encaixe. Possivelmente os furos do relé também são menores que o desejado, mas não foram testados neste trabalho.
- *Header* para interface com circuitos esternos não compatível com a placa ZigBit do LARA: As trilhas que realizam a comunicação SPI foram projetadas levando em conta os parâmetros informados para realizar a comunicação SPI com a placa ZigBit desenvolvida no LARA. No entanto os pinos informados são na realidade usados para a gravação SPI do modulo ZigBit, e não para a comunicação. Estas trilhas devem ser movidas para os pinos correspondentes a UART do ZigBit para as placas serem compatíveis sem alterações posteriores.
- Circuito de alimentação ineficiente: O circuito empregado não atendeu as nossas expectativas,

e foi fonte de diversos problemas. A fonte além de apresentar baixa eficiência, não possui regulador de corrente, não suporta picos de corrente podendo ser danificada, apresenta uma queda de tensão alta e não é isolada da rede. Sugerimos que seja substituída nas próximas iterações da placa por uma fonte com transformador e regulador de tensão ou conversor DC-DC.

- Vida útil com baterias ineficiente: Ao realizarmos testes alimentando a parte processamento e transmissão com baterias de 9 V comerciais percebemos que sua vida útil era inferior a quinze minutos. Acreditamos que isto seja devido ao alto consumo dos optoacopladores utilizados. Propomos ao modificarmos a alimentação seja projetada uma saída para alimentar esta parte do circuito. O isolamento provido pelos optoacopladores seria sacrificado, mas se a fonte for isolada da rede acreditamos que não haverá riscos para o restante do circuito. Assim não teríamos a necessidade do uso de uma segunda fonte ou baterias como é o caso atualmente.

6.2 Trabalhos Futuros

Este trabalho espera abrir portas para diversos projetos no ambiente de automação predial do LARA, alguns mais diretos, outros menos, mas todos de interesse imediato para qualquer aplicação que necessite da leitura dos parâmetros em tempo real. São estes:

- Correção dos *bugs* apresentados e projeto de uma nova fonte para a placa.
- Implementação e realização de testes de atuação com relé de estado sólido. A PCB projetada já possui o espaço designado para o relé T2405Z-M da Teletronic ilustrado na figura **??**, no entanto devido a falta destes componentes no período não foram realizados testes.

Figura 6.1: Relé de estado sólido T2405Z-M da Teletronic e esquemático interno

- Realização de testes de calibração e de consumo do próprio módulo desenvolvido.
- Implementação de uma rede ZigBee com módulos ZigBit disponíveis no LARA e implementação de uma biblioteca Modbus para ZigBit. Utilizar módulos ZigBit para fazer a parte de processamento e transmissão representa uma economia de hardware para a implementação da rede já que substituímos o Arduino e XBee por um ZigBit. A necessidade do desenvolvimento de uma biblioteca Modbus é devido ao fato de não possuirmos programas supervisórios

que se comuniquem com o padrão ZigBee. Bibliotecas para outros protocolos utilizados em supervisórios podem ser implementadas se forem consideradas mais vantajosas.

- Testes de consumo de aparelhos de ar condicionado ao longo do tempo levando em conta parâmetros externos tais como temperatura externa, umidade, numero de indivíduos no laboratório

- Integração dos módulos com demais projetos ligados ao ambiente de automação residencial do LARA para a criação de um ambiente inteligente que faça a atuação e medição do consumo de energia dos ar condicionados.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] ALMEIDA, W. G. de; FREITAS, F. D. *Circuitos Polifásicos*. : Finatec-Fundação de Empreendimentos Científicos e Tecnológicos, UnB, 1995.

[2] EEI. *Handbook for Electricity Metering*. Décima edição. : Edison Electric Institute, 2002.

[3] WRIGHT, D. B. e E. *Pratical SCADA for Industry*. Primeira edição. : IDC Technologies, 2003.

[4] SAMES. *SA9903B-Single Phase Power / Energy IC with SPI Interface*.

[5] DIGI INTERNATIONAL INC. *XBee/XBee-PRO RF Modules*.

[6] DATASHEET TGHC Series Precision Current Sente Resistors.

[7] DATASHEET Single-Channel: 6N137, HCPL2601, HCPL 2611.

[8] SOCIETY, I. C. *802.15.4d: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Lor-Rate Wireless Personal Area Networks (WPANs)*. 2009.

[9] DATASHEET Atmel ATmega1281V.

[10] MANUAL do Adaptador CON-USBBEE.

[11] DATASHEET Rele de Estado Sólido T2405Z-M.

[12] DIGI INTERNATIONAL INC. *X-CTU: Configuration and Test Utility Software Users Guide*.

[13] ATMEL Corporation. Outubro 2012. Disponível em: <http://www.atmel.com>.

[14] THE Modbus Organization. Março 2013. Disponível em: <http://www.modbus.org>.

[15] MODBUS - Wikipedia, the free encyclopedia. Março 2013. Disponível em: <https://en.wikipedia.org/wiki/Modbus>.

[16] MODBUS Application Protocol Specification, v1.1b3.

[17] SCADA - Wikipedia, the free encyclopedia. Janeiro 2013. Disponível em: <http://en.wikipedia.org/wiki/SCADA>.

[18] SCADABR. Dezembro 2012. Disponível em: <http://www.scadabr.com.br>.

[19] ARDUINO - HomePage. Outubro 2012. Disponível em: <http://www.arduino.cc>.

ANEXOS

I. TABELA DE COMPONENTES

Tabela I.1: Tabela de Componentes

Componente	Descrição	Quantidade
U1	SA9903B PDIP20	1
U2, U3, U4, U5, U6	6N137 PDIP8	5
R1	150 kΩ 1/4 W 1%	1
R2, R3	110 kΩ 1/4 W 1%	2
R4	24 kΩ 1/4 W 1%	1
R5	1 MΩ 1/4 W 1%	1
R6, R7	6,2 kΩ 1/4 W 1%	2
R8	24 kΩ 1/4 W 1%	1
R9	47 Ω 2 W 5%	1
R10, R11	680 Ω 1/4 W 5%	2
R12, R13, R14, R15, R16	1 kΩ 1/4 W 5%	5
RSH	TGHGCR0050FE-ND	1
C1, C2	220 nF 16 V cerâmico	2
C3, C4	220 uF 25 V eletrolítico	2
C5	470 nF 250 VAC poliéster	1
C6	820 nF 16 V cerâmico	1
C7, C8, C9, C10, C11	100 nF 16 V cerâmico	5
D1,D2	1N4003	2
D3, D4	Zener 2.4 V	2
X1	Cristal 3.579545 MHz	1
J1	Headers	varios

II. DIAGRAMAS ESQUEMÁTICOS

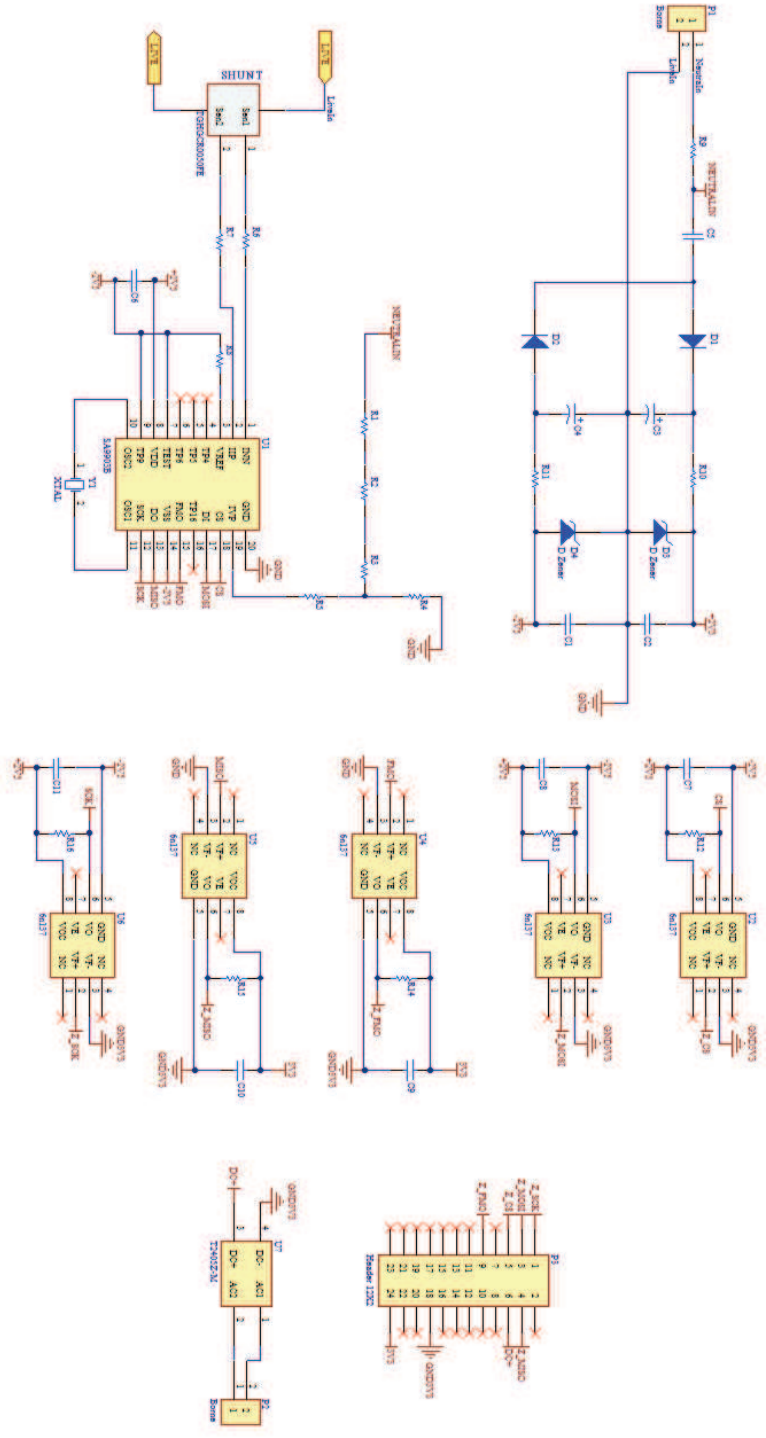

Figura II.1: Esquemático completo da placa de medição

III. CÓDIGOS FONTE

III.1 versao_final.cpp

```
#include "sa9903b.h"
#include <SPI.h> //nao sei pq tem que dar include aqui tb, ja que ja existe um
    include em sa9903b.cpp
#include <SimpleModbusSlave.h>

#define SLAVE_ID 1

enum {ERRORS,VOLTAGE,FREQUENCY,ACTIVE,REACTIVE,REGISTER_SIZE};

SA9903B sa9903b;

unsigned int reg[REGISTER_SIZE] = {0};

void setup ()
{
  sa9903b.Setup();
  modbus_configure(57600,SLAVE_ID,0,REGISTER_SIZE);
}

void loop ()
{
  sa9903b.Run();

  if (sa9903b.GetUpdatedStatus())
  {
    reg[VOLTAGE] = sa9903b.GetVoltage();
    reg[FREQUENCY] = sa9903b.GetFrequency();
    reg[ACTIVE] = abs(sa9903b.GetActive());
    reg[REACTIVE] = abs(sa9903b.GetReactive());
    reg[ERRORS] = modbus_update(reg);
  }
}
```

codigos/versao_final.cpp

III.2 sa9903b.h

```
#ifndef SA9903B_H_GUARD
#define SA9903B_H_GUARD

#include <Arduino.h>

```

```
#define START_ADDR 0x01
#define ACTIVE_ADDR 0x80
#define REACTIVE_ADDR 0x81
#define VOLTAGE_ADDR 0x82
#define FREQUENCY_ADDR 0x83

#define CS_PIN 48
#define FMO_PIN 49

//#define RATED_VOLTAGE 230.0
//#define RATED_CURRENT 40.0
//#define RATED_VOLTAGE 235.013
#define CORRECTION_FACTOR 230.0/265.0
#define RATED_VOLTAGE 230.0*CORRECTION_FACTOR
#define RATED_CURRENT 39.68
#define CRYSTAL_FREQUENCY 3579545.0

class SA9903B
{
public:
  SA9903B();
  ~SA9903B();

  void Setup();
  void Run();

  float GetVoltage();
  float GetFrequency();
  float GetActive();
  float GetReactive();
  boolean GetUpdatedStatus();

private:
  long active;
  long last_active;
  long reactive;
  long last_reactive;
  long frequency;
  long voltage;

  int last_fmo;

  unsigned long last_time;
  unsigned long elapsed_time;

  boolean updated;

  long Read(byte address);
  void ReadAll();
};

#endif //SA9903B_H_GUARD
```

codigos/sa9903b.h

III.3 sa9903b.cpp

```
#include "sa9903b.h"
#include <SPI.h>

SA9903B::SA9903B()
{
  active = 0;
  last_active = 0;
  reactive = 0;
  last_reactive = 0;
  frequency = 0;
  voltage = 0;

  last_fmo = LOW;

  last_time = 0;
  elapsed_time = 0;

  updated = false;
}

SA9903B::~SA9903B()
{

}

void SA9903B::Setup()
{
  SPI.begin();
  SPI.setBitOrder(MSBFIRST);
  SPI.setClockDivider(SPI_CLOCK_DIV64);

  pinMode(FMO_PIN,INPUT);

  pinMode(CS_PIN,OUTPUT);
  digitalWrite(CS_PIN,HIGH);
}

long SA9903B::Read(const byte address)
{
  long result = 0;

  digitalWrite(CS_PIN,LOW);

  SPI.setDataMode(SPI_MODE2);
```

```
45    SPI.transfer(~START_ADDR);
      SPI.transfer(~address);
47
      SPI.setDataMode(SPI_MODE3);
49    result = SPI.transfer(0xFF);
      result = result << 8;
51    result |= SPI.transfer(0xFF);
      result = result << 8;
53    result |= SPI.transfer(0xFF);

55    digitalWrite(CS_PIN,HIGH);

57    return ~result;
  }
59
  void SA9903B::ReadAll()
61{
      active = Read(ACTIVE_ADDR);
63    reactive = Read(REACTIVE_ADDR);
      frequency = Read(FREQUENCY_ADDR);
65    voltage = Read(VOLTAGE_ADDR);
  }
67
  float SA9903B::GetVoltage()
69{
      voltage &= 0x00FFFFFF;
71    return RATED_VOLTAGE/700.0*voltage;
  }
73
  float SA9903B::GetFrequency()
75{
      frequency &= 0x000003FF;
77    return CRYSTAL_FREQUENCY/256.0/frequency;
  }
79
  float SA9903B::GetActive()
81{
      active &= 0x00FFFFFF;
83    float time = elapsed_time/1000000.0;
      float diff;
85
      if (active < 0x100000 && last_active > 0xEFFFFF)
87    {
        diff = (active+0xFFFFFF)-last_active;
89    }
      else if (active > 0xEFFFFF && last_active < 0x100000)
91    {
        diff = active-(last_active+0xFFFFFF);
93    }
      else
95    {
        diff = active-last_active;
```

```
97    }
98
99    return RATED_VOLTAGE*RATED_CURRENT*diff/time/320000.0;
100 }
101
102 float SA9903B::GetReactive()
103 {
104   reactive &= 0x00FFFFFF;
105   float time = elapsed_time/1000000.0;
106   float diff;
107
108   if (reactive < 0x100000 && last_reactive > 0xEFFFFF)
109   {
110     diff = (reactive+0xFFFFFF)-last_reactive;
111   }
112   else if (reactive > 0xEFFFFF && last_reactive < 0x100000)
113   {
114     diff = reactive-(last_reactive+0xFFFFFF);
115   }
116   else
117   {
118     diff = reactive-last_reactive;
119   }
120
121   return RATED_VOLTAGE*RATED_CURRENT*diff/time/320000.0;
122 }
123
124 void SA9903B::Run()
125 {
126   int fmo = digitalRead(FMO_PIN);
127
128   if (fmo == HIGH && last_fmo == LOW)
129   {
130     last_active = active;
131     last_reactive = reactive;
132
133     unsigned long time = micros();
134
135     ReadAll();
136
137     elapsed_time = time - last_time;
138     last_time = time;
139
140     updated = true;
141   }
142
143   last_fmo = fmo;
144 }
145
146 boolean SA9903B::GetUpdatedStatus()
147 {
148   if (updated)
```

```
149   {
        updated = false;
151     return true;
      }
153   return false;
}
```

codigos/sa9903b.cpp

III.4 SimpleModbusSlave.h

```
#ifndef SIMPLE_MODBUS_SLAVE_H
2 #define SIMPLE_MODBUS_SLAVE_H

4 /*
  SimpleModbusSlave allows you to communicate
6 to any slave using the Modbus RTU protocol.

8 The crc calculation is based on the work published
  by jpmzometa at
10 http://sites.google.com/site/jpmzometa/arduino-mbrt

12 By Juan Bester : bester.juan@gmail.com

14 The functions implemented are functions 3 and 16.
  read holding registers and preset multiple registers
16 of the Modbus RTU Protocol, to be used over the Arduino serial connection.

18 This implementation DOES NOT fully comply with the Modbus specifications.

20 Specifically the frame time out have not been implemented according
  to Modbus standards. The code does however combine the check for
22 inter character time out and frame time out by incorporating a maximum
  time out allowable when reading from the message stream.
24
  These library of functions are designed to enable a program send and
26 receive data from a device that communicates using the Modbus protocol.

28 SimpleModbusSlave implements an unsigned int return value on a call to
      modbus_update().
  This value is the total error count since the slave started. It's useful for fault
       finding.
30
  This code is for a Modbus slave implementing functions 3 and 16
32 function 3: Reads the binary contents of holding registers (4X references)
  function 16: Presets values into a sequence of holding registers (4X references)
34
  All the functions share the same register array.
36
  Exception responses:
```

```
38  1 ILLEGAL FUNCTION
    2 ILLEGAL DATA ADDRESS
40  3 ILLEGAL DATA VALUE

42  Note:
    The Arduino serial ring buffer is 128 bytes or 64 registers.
44  Most of the time you will connect the arduino to a master via serial
    using a MAX485 or similar.
46
    In a function 3 request the master will attempt to read from your
48  slave and since 5 bytes is already used for ID, FUNCTION, NO OF BYTES
    and two BYTES CRC the master can only request 122 bytes or 61 registers.
50
    In a function 16 request the master will attempt to write to your
52  slave and since a 9 bytes is already used for ID, FUNCTION, ADDRESS,
    NO OF REGISTERS, NO OF BYTES and two BYTES CRC the master can only write
54  118 bytes or 59 registers.

56  Using the FTDI converter ic the maximum bytes you can send is limited
    to its internal buffer which is 60 bytes or 30 unsigned int registers.
58
    Thus:
60
    In a function 3 request the master will attempt to read from your
62  slave and since 5 bytes is already used for ID, FUNCTION, NO OF BYTES
    and two BYTES CRC the master can only request 54 bytes or 27 registers.
64
    In a function 16 request the master will attempt to write to your
66  slave and since a 9 bytes is already used for ID, FUNCTION, ADDRESS,
    NO OF REGISTERS, NO OF BYTES and two BYTES CRC the master can only write
68  50 bytes or 25 registers.

70  Since it is assumed that you will mostly use the Arduino to connect to a
    master without using a USB to Serial converter the internal buffer is set
72  the same as the Arduino Serial ring buffer which is 128 bytes.

74  The functions included here have been derived from the
    Modbus Specifications and Implementation Guides
76
    http://www.modbus.org/docs/Modbus_over_serial_line_V1_02.pdf
78  http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf
    http://www.modbus.org/docs/PI_MBUS_300.pdf
80 */

82 #include "Arduino.h"

84 // function definitions
   void modbus_configure(long baud, byte _slaveID, byte _TxEnablePin, unsigned int
       _holdingRegsSize);
86 unsigned int modbus_update(unsigned int *holdingRegs);


88
```

```
#endif
```

codigos/SimpleModbusSlave.h

III.5 SimpleModbusSlave.cpp

```
#include "SimpleModbusSlave.h"

#define BUFFER_SIZE 128

// frame[] is used to recieve and transmit packages.
// The maximum serial ring buffer size is 128
unsigned char frame[BUFFER_SIZE];
unsigned int holdingRegsSize; // size of the register array
unsigned char broadcastFlag;
unsigned char slaveID;
unsigned char function;
unsigned char TxEnablePin;
unsigned int errorCount;
unsigned int T1_5; // inter character time out
unsigned int T3_5; // frame delay

// function definitions
void exceptionResponse(unsigned char exception);
unsigned int calculateCRC(unsigned char bufferSize);
void sendPacket(unsigned char bufferSize);

unsigned int modbus_update(unsigned int *holdingRegs)
{
  unsigned char buffer = 0;
  unsigned char overflow = 0;

  while (Serial.available())
  {
    // The maximum number of bytes is limited to the serial buffer size of 128
        bytes
    // If more bytes is received than the BUFFER_SIZE the overflow flag will be set
         and the
    // serial buffer will be red untill all the data is cleared from the receive
        buffer.
    if (overflow)
      Serial.read();
    else
    {
      if (buffer == BUFFER_SIZE)
        overflow = 1;
      frame[buffer] = Serial.read();
      buffer++;
    }
    delayMicroseconds(T1_5); // inter character time out
```

```
  }

  // If an overflow occurred increment the errorCount
  // variable and return to the main sketch without
  // responding to the request i.e. force a timeout
  if (overflow)
    return errorCount++;

  // The minimum request packet is 8 bytes for function 3 & 16
  if (buffer > 7)
  {
    unsigned char id = frame[0];

    broadcastFlag = 0;

    if (id == 0)
      broadcastFlag = 1;

    if (id == slaveID || broadcastFlag) // if the recieved ID matches the slaveID
        or broadcasting id (0), continue
    {
      unsigned int crc = ((frame[buffer - 2] << 8) | frame[buffer - 1]); // combine
          the crc Low & High bytes
      if (calculateCRC(buffer - 2) == crc) // if the calculated crc matches the
          recieved crc continue
      {
        function = frame[1];
        unsigned int startingAddress = ((frame[2] << 8) | frame[3]); // combine the
            starting address bytes
        unsigned int no_of_registers = ((frame[4] << 8) | frame[5]); // combine the
            number of register bytes
        unsigned int maxData = startingAddress + no_of_registers;
        unsigned char index;
        unsigned char address;
        unsigned int crc16;

        // broadcasting is not supported for function 3
        if (!broadcastFlag && (function == 3))
        {
          if (startingAddress < holdingRegsSize) // check exception 2 ILLEGAL DATA
              ADDRESS
          {
            if (maxData <= holdingRegsSize) // check exception 3 ILLEGAL DATA VALUE
            {
              unsigned char noOfBytes = no_of_registers * 2;
              unsigned char responseFrameSize = 5 + noOfBytes; // ID, function,
                  noOfBytes, (dataLo + dataHi) * number of registers, crcLo, crcHi
              frame[0] = slaveID;
              frame[1] = function;
              frame[2] = noOfBytes;
              address = 3; // PDU starts at the 4th byte
              unsigned int temp;
```

```
87
                for (index = startingAddress; index < maxData; index++)
89              {
                  temp = holdingRegs[index];
91                frame[address] = temp >> 8; // split the register into 2 bytes
                  address++;
93                frame[address] = temp & 0xFF;
                  address++;
95              }

97              crc16 = calculateCRC(responseFrameSize - 2);
                frame[responseFrameSize - 2] = crc16 >> 8; // split crc into 2 bytes
99              frame[responseFrameSize - 1] = crc16 & 0xFF;
                sendPacket(responseFrameSize);
101           }
              else
103             exceptionResponse(3); // exception 3 ILLEGAL DATA VALUE
            }
105         else
              exceptionResponse(2); // exception 2 ILLEGAL DATA ADDRESS
107       }
          else if (function == 16)
109       {
            // check if the recieved number of bytes matches the calculated bytes
                minus the request bytes
111         // id + function + (2 * address bytes) + (2 * no of register bytes) +
                byte count + (2 * CRC bytes) = 9 bytes
            if (frame[6] == (buffer - 9))
113         {
              if (startingAddress < holdingRegsSize) // check exception 2 ILLEGAL
                  DATA ADDRESS
115           {
                if (maxData <= holdingRegsSize) // check exception 3 ILLEGAL DATA
                    VALUE
117             {
                  address = 7; // start at the 8th byte in the frame
119
                  for (index = startingAddress; index < maxData; index++)
121               {
                    holdingRegs[index] = ((frame[address] << 8) | frame[address + 1])
                        ;
123                 address += 2;
                  }
125
                  // only the first 6 bytes are used for CRC calculation
127               crc16 = calculateCRC(6);
                  frame[6] = crc16 >> 8; // split crc into 2 bytes
129               frame[7] = crc16 & 0xFF;

131               // a function 16 response is an echo of the first 6 bytes from the
                      request + 2 crc bytes
                  if (!broadcastFlag) // don't respond if it's a broadcast message
```

```
133                     sendPacket(8);
                      }
135                   else
                        exceptionResponse(3); // exception 3 ILLEGAL DATA VALUE
137                 }
                    else
139                   exceptionResponse(2); // exception 2 ILLEGAL DATA ADDRESS
                  }
141               else
                    errorCount++; // corrupted packet
143             }
                else
145               exceptionResponse(1); // exception 1 ILLEGAL FUNCTION
            }
147         else // checksum failed
              errorCount++;
149       } // incorrect id
    }
151   else if (buffer > 0 && buffer < 8)
      errorCount++; // corrupted packet
153
    return errorCount;
155 }

157 void exceptionResponse(unsigned char exception)
    {
159   errorCount++; // each call to exceptionResponse() will increment the errorCount
      if (!broadcastFlag) // don't respond if its a broadcast message
161   {
        frame[0] = slaveID;
163     frame[1] = (function | 0x80); // set the MSB bit high, informs the master of an
            exception
        frame[2] = exception;
165     unsigned int crc16 = calculateCRC(3); // ID, function + 0x80, exception code ==
            3 bytes
        frame[3] = crc16 >> 8;
167     frame[4] = crc16 & 0xFF;
        sendPacket(5); // exception response is always 5 bytes ID, function + 0x80,
            exception code, 2 bytes crc
169   }
    }
171

    void modbus_configure(long baud, unsigned char _slaveID, unsigned char _TxEnablePin
        , unsigned int _holdingRegsSize)
173 {
      slaveID = _slaveID;
175   Serial.begin(baud);

177   if (_TxEnablePin > 1)
      { // pin 0 & pin 1 are reserved for RX/TX. To disable set txenpin < 2
179     TxEnablePin = _TxEnablePin;
        pinMode(TxEnablePin, OUTPUT);
```

```
181     digitalWrite(TxEnablePin, LOW);
    }
183
    // Modbus states that a baud rate higher than 19200 must use a fixed 750 us
185 // for inter character time out and 1.75 ms for a frame delay.
    // For baud rates below 19200 the timeing is more critical and has to be
        calculated.
187 // E.g. 9600 baud in a 10 bit packet is 960 characters per second
    // In milliseconds this will be 960characters per 1000ms. So for 1 character
189 // 1000ms/960characters is 1.04167ms per character and finaly modbus states an
    // intercharacter must be 1.5T or 1.5 times longer than a normal character and
        thus
191 // 1.5T = 1.04167ms * 1.5 = 1.5625ms. A frame delay is 3.5T.

193 if (baud > 19200)
    {
195   T1_5 = 750;
      T3_5 = 1750;
197 }
    else
199 {
      T1_5 = 15000000/baud; // 1T * 1.5 = T1.5
201   T3_5 = 35000000/baud; // 1T * 3.5 = T3.5
    }
203
    holdingRegsSize = _holdingRegsSize;
205 errorCount = 0; // initialize errorCount
}
207
unsigned int calculateCRC(byte bufferSize)
209 {
  unsigned int temp, temp2, flag;
211 temp = 0xFFFF;
  for (unsigned char i = 0; i < bufferSize; i++)
213 {
    temp = temp ^ frame[i];
215   for (unsigned char j = 1; j <= 8; j++)
    {
217   flag = temp & 0x0001;
      temp >>= 1;
219   if (flag)
        temp ^= 0xA001;
221   }
  }
223 // Reverse byte order.
  temp2 = temp >> 8;
225 temp = (temp << 8) | temp2;
  temp &= 0xFFFF;
227 return temp; // the returned value is already swopped - crcLo byte is first &
      crcHi byte is last
}
229
```

```
void sendPacket(unsigned char bufferSize)
{
  if (TxEnablePin > 1)
    digitalWrite(TxEnablePin, HIGH);

  for (unsigned char i = 0; i < bufferSize; i++)
    Serial.write(frame[i]);

  Serial.flush();

  // allow a frame delay to indicate end of transmission
  delayMicroseconds(T3_5);

  if (TxEnablePin > 1)
    digitalWrite(TxEnablePin, LOW);
}
```

codigos/SimpleModbusSlave.cpp

IV. DESCRIÇÃO DO CONTEÚDO DO CD