PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ

AUTOMAÇÃO RESIDENCIAL via WEB

CURITIBA 2010

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ

ARTHUR BRUGNARI

LUIZ HENRIQUE MUSSI MAESTRELLI

AUTOMAÇÃO RESIDENCIAL via WEB

Projeto Final apresentado ao Programa de Aprendizagem de PROJETO FINAL II, do Curso de Graduação em Engenharia de Computação da Pontifícia Universidade Católica do Paraná.

_____

Professor Orientador: Afonso Miguel

CURITIBA 2010

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ

Sumário

CURITIBA 2010

1. RESUMO

Este projeto tem como objetivo desenvolver um ambiente WEB centralizado de informações sobre dispositivos micro-controlados de baixa potência e baixo consumo. O projeto será demonstrado em dois ambientes distintos, o primeiro simulando um ambiente residencial comum, no qual haverá um dispositivo mestre que terá por objetivo a manutenção e controle dos dispositivos atuadores e dos demais dispositivos escravos. Com isso a intenção será de gerenciar os ambientes controlando seus dispositivos remotamente.

O principal propósito do projeto é o de fazer o cliente interagir com o ambiente utilizando os dispositivos atuadores via web, dentre alguns exemplos possíveis seriam eles: ligar luzes, acionar equipamentos, entre outras funcionalidades que estejam ligadas na rede elétrica e implementadas com atuadores.

2. INTRODUÇÃO

O papel da automação vai além do aumento da eficiência e qualidade de vida no âmbito residencial, mas também pode ser focada a ambientes corporativos, ela está intimamente ligada ao uso eficaz da energia, sendo importante para a economia e o meio ambiente.

O objetivo do presente projeto visa disponibilizar ao cliente a possibilidade de interagir com as novas tecnologias de automação que poderão ser utilizadas tanto para automação residencial, mas também poderão ser utilizadas em diversas empresas, unidades de saúde, dentre outros. Em todos os casos possibilitará ao cliente atuar com acesso total nos dispositivos elétricos que estiverem com os atuadores ligados a central, através de uma conexão remota (via WEB), também poderão automatizar funções cotidianas, pois terá acesso a sensores e atuadores através do ambiente WEB, provendo serviços como segurança remota. Isso é obtido através de um projeto único que envolve infra-estrutura, dispositivos e software de controle, cuja meta é garantir ao usuário a possibilidade de acesso e de controle do ambiente automatizado, dentro ou fora da mesma, via web.

A figura 1 ilustra as possibilidades possíveis para automação de uma residência.

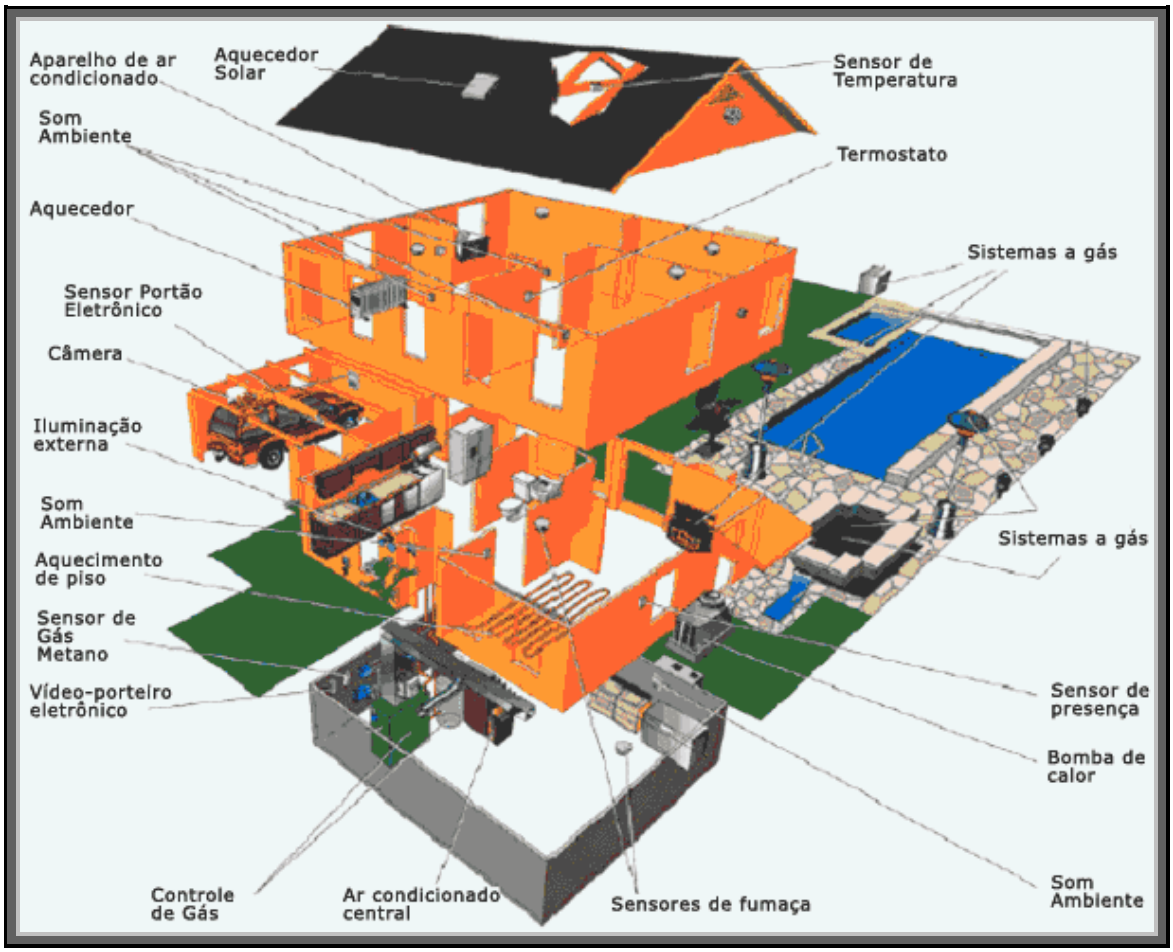

Figura 1 – Possíveis formas de automação.

Para conclusão do presente projeto foram implementados dois módulos, os quais serão detalhados no decorrer deste documento. Foi desenvolvido um servidor web que faz todo o gerenciamento das requisições vindas da internet, e este aciona os atuadores em questão para retornarem a informação desejada, o servidor WEB e o processamento das requisições são processados em uma placa de desenvolvimento Arduino, e para a transmissão do sinal utilizamos antenas RF.

No módulo atuador, foi desenvolvido um sistema que recebe o sinal vindo da Central através do dispositivo RF, e assim a placa de desenvolvimento Arduino Nano recebe e processa toda as informações, e atua conforme solicitação do usuário. Foram feitas todas as validações necessárias

para a certificação da integridade dos módulos, que também esta detalhado no decorrer do documento.

O presente documento listará os seguintes tópicos abaixo:

2.1. DETALHAMENTO DO PROJETO:

Descreve aspectos tecnológicos relevantes aos recursos utilizados para implementação do presente projeto.

2.2. TESTES E RESULTADOS:

Descreve os testes que foram realizados para atender as validações.

2.3. CONCLUSÃO:

Destaca os principais aspectos do projeto, resumindo os seus objetivos.

2.4. REFERÊNCIAS BIBLIOGRÁFICAS:

Detalhamento dos recursos utilizados para pesquisas, desenvolvimento, e aprimoramento de conteúdos referente ao projeto que foi desenvolvido..

2.5. ANEXOS:

Item que contem informação adicionais do projeto.

3. DETALHAMENTO DO PROJETO

A seguir, encontra-se o Diagrama de Blocos (Figura 2) geral do projeto, contendo os dois módulos a serem desenvolvidos, além de uma descrição detalhada do funcionamento tanto de hardware quanto de software de cada módulo utilizado no projeto.

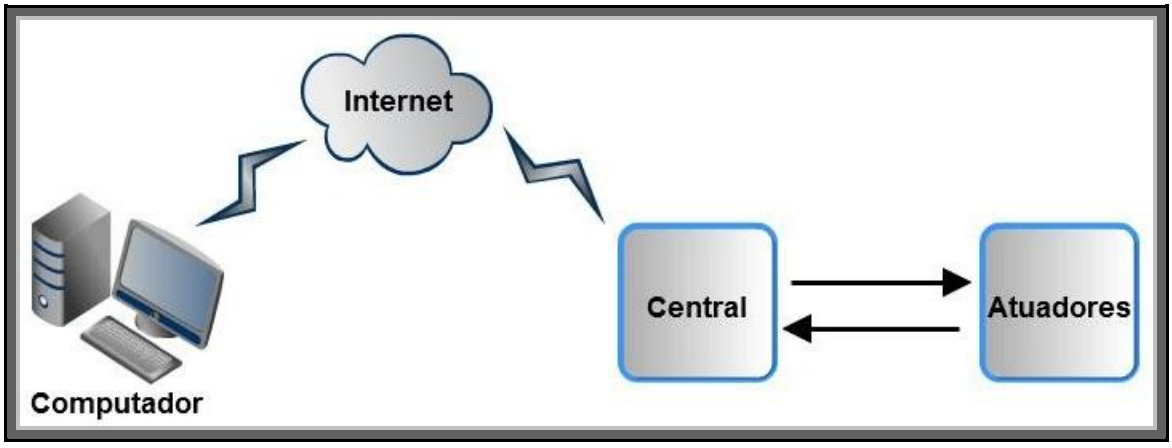

Figura 2 – Diagrama de blocos geral do projeto.

A figura 3, que segue logo abaixo, ilustra cada módulo juntamente com seus componentes internos e estes estarão se comunicando através das antenas RF.

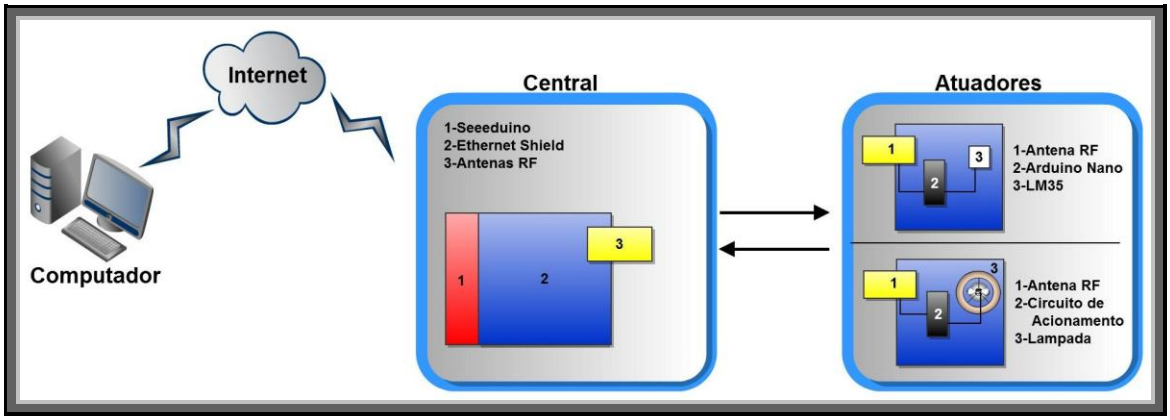

Figura 3 – Diagrama de blocos detalhados do projeto.

3.1. CENTRAL

Central é o módulo responsável pelo gerenciamento e encaminhamento dos dados recebidos a partir de outros módulos. A Central é composta por três dispositivos de hardware sendo eles: Seeduino v1.1, Arduino Ethernet Shield e uma antena RF/Serial (APC220), os quais serão detalhados a seguir.

3.1.1. SEEEDUINO

Seeeduino é um computador físico baseado numa simples plataforma de hardware livre, projetada com um microcontrolador de placa única, com suporte de entrada/saída embutido e uma linguagem de programação padrão, que é essencialmente C/C++.

Uma placa Seeeduino é composta por um controlador, algumas linhas de E/S digital e analógica, além de uma interface serial ou USB, para interligar-se ao hospedeiro, que é usado para programá-la e interagi-la em tempo real.

Sua placa consiste em um microcontrolador Atmel AVR de 8 bits, com componentes complementares para facilitar a programação e incorporação para outros circuitos. Um importante aspecto é a maneira padrão que os conectores são expostos, permitindo o CPU ser interligado a outros módulos expansivos, conhecidos como Shields.

A descrição abaixo resume algumas características do Seeeduino.

- Microcontroller ATmega168
- Operating Voltage 5V/3.3V
- Input Voltage (recommended) 7-12 V
- Input Voltage (limits) 6-20 V
- Digital I/O Pins 14 (of which 6 provide PWM output)
- Analog Input Pins 8
- DC Current per I/O Pin 40 mA
- DC Current for 3.3V Pin 50 mA
- Flash Memory 16 KB (of which 2 KB used by boot loader)
- SRAM 1 KB
- EEPROM 512 bytes
- Clock Speed 16 MHz

O Seeduino vem gravado um bootloader que permite que você faça o upload do novo código para ele sem a utilização de um programador de hardware externo. Ele se comunica utilizando o protocolo STK500 original. O Arduino Software é o compilador utilizado para o upload do novo código.

Os projetos e esquemas de hardwares são distribuídos sob a licença Creative Commons Attribution Share-Alike 2.5, e estão disponíveis em sua página oficial. Arquivos de layout e produção para algumas versões também estão hospedadas. O código fonte para o IDE e a biblioteca de funções da placa são disponibilizadas sob a licença GPLv2 e hospedadas pelo projeto Google Code. Na figura 4 encontra – se detalhado o módulo Seeeduino.

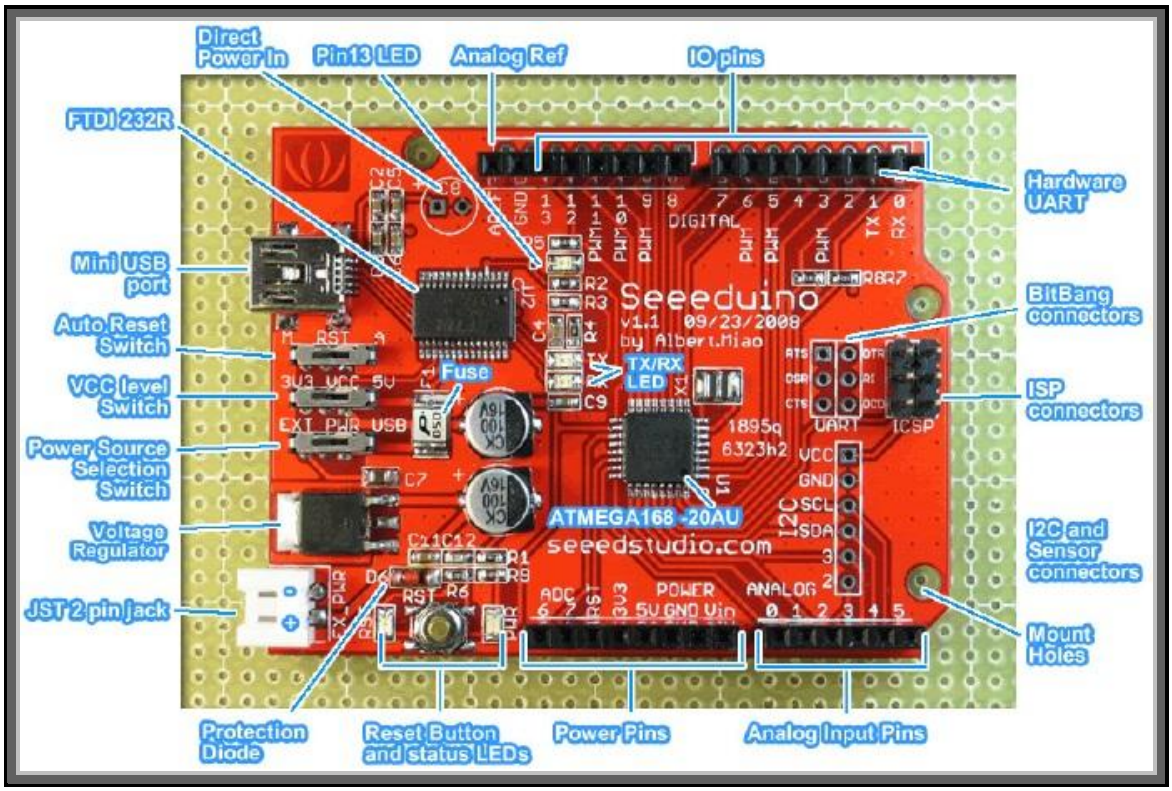

Figura 4 – Detalhamento do módulo Seeeduino.

3.1.2. ETHERNET SHIELD

O Arduino Ethernet Shield V1.1 é compatível com os padrões das placas Seeeduino. O ENC28J60 é um controlador Ethernet stand-alone com um padrão da indústria Serial Peripheral Interface (SPI). Ele é projetado para servir como uma rede Ethernet com interface para qualquer controlador equipado com SPI. O ENC28J60 satisfaz todas as especificações IEEE 802.3, e possui duas camadas sendo uma a camada PHY(Physical Layer) e a outra MAC (Medium Access Layer) e uma tomada RJ45 padrão.

Na figura 5 encontra-se os parâmetros e valores do ENC28J60. Na figura 6 encontra-se o módulo Ethernet Shield.

Parameter Name	Value
MAC	Yes
PHY	Yes
TX/RX RAM Buffer(bytes)	8192
Interrupt Pin	1
LEDs	2
Op. Voltage (V)	3.3
Temp. Range Min. (°C)	-40
Temp. Range Max. (°C)	85
Max. Speed (MHz)	25
Interface	SPI
Pre-Programmed MAC Address	No
Security Engines	No
Standalone Ethernet Controller	10Base-T

Figura 5 – Parâmetros e valores do ENC28J60.

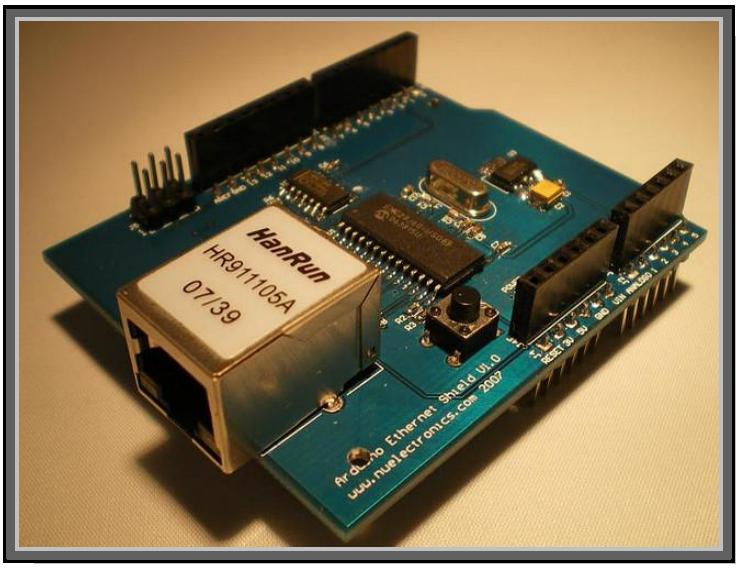

Figura 6 – Etherneth Shield.

O software Ethernet Shield é no formato da biblioteca Seeeduino. A biblioteca é implementada com base em arduino.cc que contém uma pilha TCP/IP open-source para ATMEGA88 e ENC28J60. Na figura 6 se encontra o diagrama de blocos do ENC28J60.

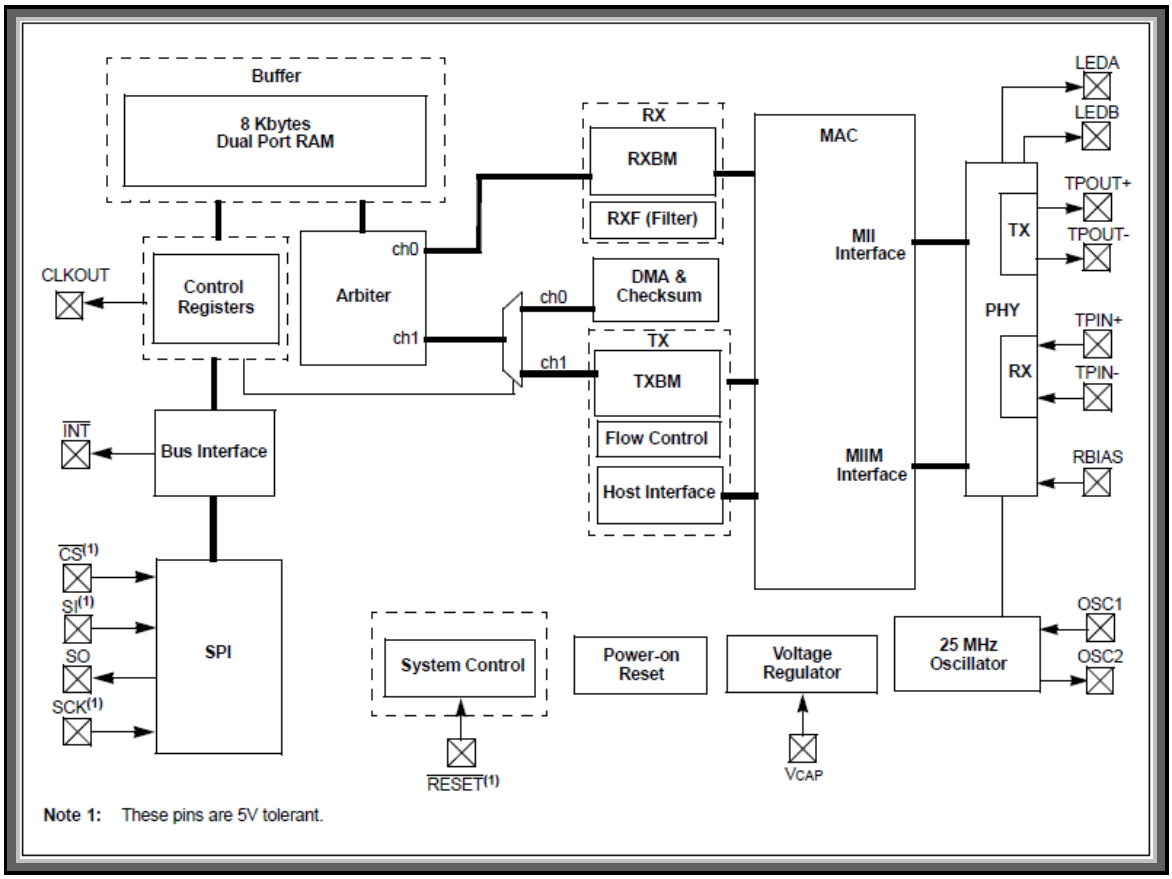

Figura 7 – Diagrama de blocos do ENC28J60.

O TCP é um padrão protocolo para estabelecer uma conexão. Para fazer isso, uma série de pacotes deve ser trocados entre os dois lados para estabelecer a conexão, então assim os pacotes de dados podem ser trocados. Normalmente, um complicado estado máquina é necessário para implementar o protocolo TCP.

Para o Seeeduino com ATMEGA168, um microcontrolador AVR 8-bit com 1K SRAM, é impossível implementar uma pilha TCP completa. Além disso, a web Page para microcontroladores de 8 bits, é normalmente usada para controlar uma lâmpada ou ler um sensor de temperatura. Portanto, em vez de implementar protocolo TCP completo, um único pacote de dados protocolo TCP é utilizado. Todo conteúdo web, incluindo todas as tags HTML, deve estar em um pacote. O comprimento do pacote é limitado pelo tamanho da SRAM, atualmente metade do espaço de memória RAM (500 bytes) é usada para buffer de pacotes de rede. Isso é suficiente para páginas simples como a que iremos implementar.

Como mencionado anteriormente um único pacote de dados do protocolo TCP é utilizado, o WEB Server está implementado diretamente na memória do ATMEGA 168, por questões de espaço em memória o tamanho da web Page será bem reduzido.

A página WEB será desenvolvida em HTML, iremos disponibilizar uma autenticação de usuário e senha, para garantirmos a segurança do cliente, a partir desta autenticação o cliente será redirecionado a sua página personalizada onde terá todos os atuadores disponíveis para o cliente interagir em seu ambiente residencial.

3.1.3. DISPOSITIVO RF (mestre)

O APC220 é altamente versátil, é uma solução de rádio de baixa potência que é fácil de configurar e integrar em qualquer projeto, que exija uma comunicação sem fio RF.

O APC220 compõe o terceiro hardware do módulo Central, ele é responsável por receber as informações vindas via serial do processador ATMEGA na sua porta Rx, e disponibilizará na sua porta Tx a informação à ser enviada para o dispositivo RF(escravo).

O diagrama de blocos abaixo demonstra o funcionamento do módulo Central.

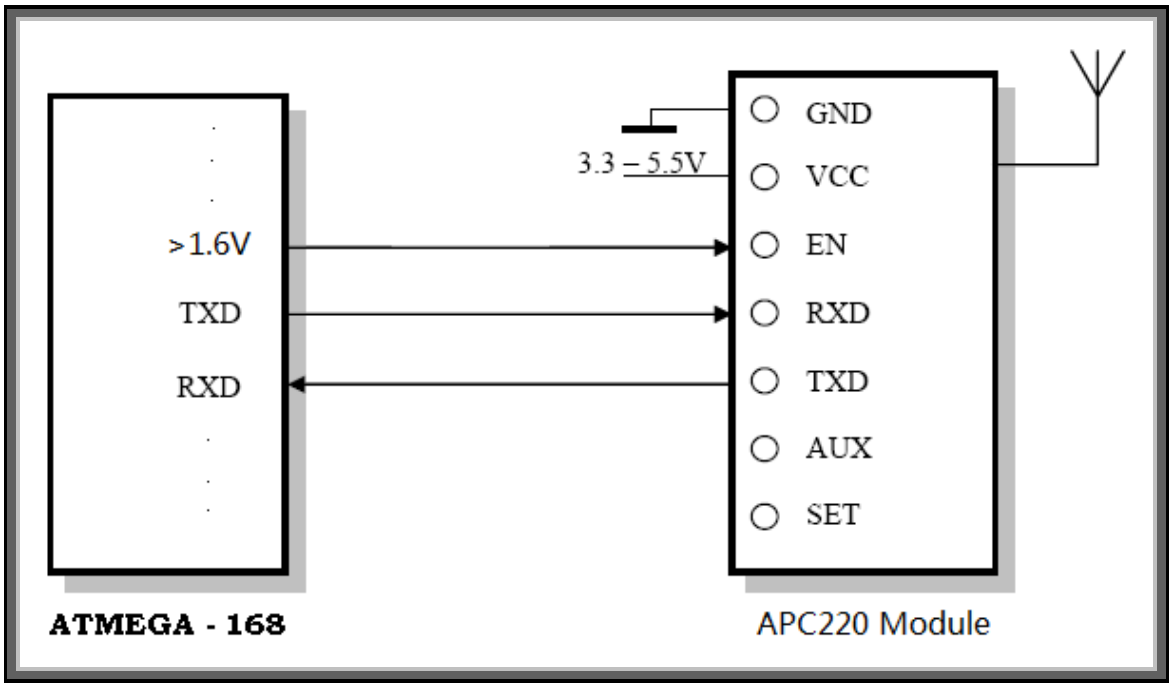

Figura 8 – Diagrama de blocos módulo Central.

3.1.4. IMPLEMENTAÇÃO

A central foi desenvolvida basicamente em duas partes: servidor web e tratamento das informações vinda do atuador. Todas essas implementações foram desenvolvidas em linguagem C e HTML.

Resumindo, a central é responsável pelo tratamento das requisições web e também pela interpretação das respostas vindas dos atuadores, para isso ela é composta por um módulo Seeduino, um Ethernet Shield e um dispositivo RF.

3.2. ATUADOR

Atuador é um elemento que produz movimento, atendendo a comandos que podem ser manuais ou automáticos, ou seja, qualquer elemento que realize um comando recebido de outro dispositivo, com base em uma entrada ou critério a ser seguido.

O atuador será conectado a um interruptor na rede elétrica da casa automatizada. O módulo atuador é composto por um Arduino Nano e por um dispositivo RF (escravo), que serão detalhados logo abaixo.

3.2.1. ARDUINO NANO

O Arduino Nano é uma placa pequena e completa, com um processador Atmega328 (Arduino Nano 3.0), ele é uma placa de quatro camadas com planos para a alimentação e para terra para auxiliar a fornecer energia suficiente para os CIs durante o chaveamento e reduzir o ruído (EMC) durante o chaveamento de alta velocidade dos pinos de entrada e saída. O plano do terra auxilia na redução de radiação (EMI). O plano de alimentação é de baixa indutância e assim quaisquer transientes que possam surgir na linha de alimentação serão de baixo nível. O Arduino Nano é desenhado e produzido pela Gravitech.

Abaixo se encontra listados algumas características do Arduino Nano.

- Microcontroller ATmega168
- Operating Voltage 5V
- Input Voltage (recommended) 7-12 V
- Input Voltage (limits) 6-20 V
- Digital I/O Pins 14 (of which 6 provide PWM output)
- Analog Input Pins 8
- DC Current per I/O Pin 40 mA
- Flash Memory 16 KB (of which 2 KB used by boot loader)
- SRAM 1 KB
- EEPROM 512 bytes
- Clock Speed 16 MHz

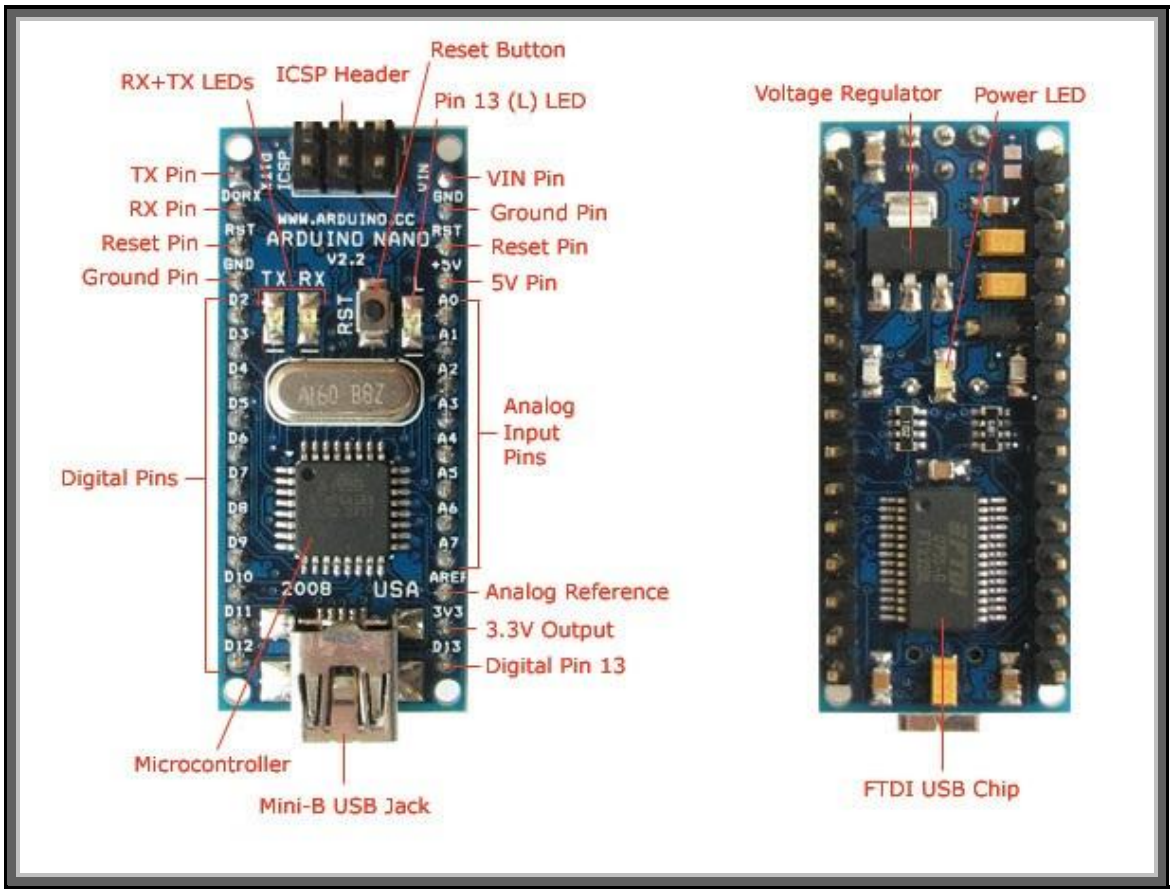

Figura 9 – Arduino Nano.

3.2.2. DISPOSITIVO RF (escravo)

O APC220 é a segunda peça chave a compor o módulo atuador, ele é responsável por receber as informações vindas do dispositivo RF (mestre) e através da interface UART/TTL disponibilizará na sua porta Rx a informação à ser processada pelo Arduino Nano.

Algumas especificações do APC220 são detalhadas abaixo.

- Working frequency: 431 MHz to 478 MHz
- Operating Range: 3.3-5.5V
- Interface: UART/TTL
- Baud rate: 1200-19200 bps
- Receive Buffer: 256 bytes

O diagrama de blocos abaixo demonstra o funcionamento geral desse módulo.

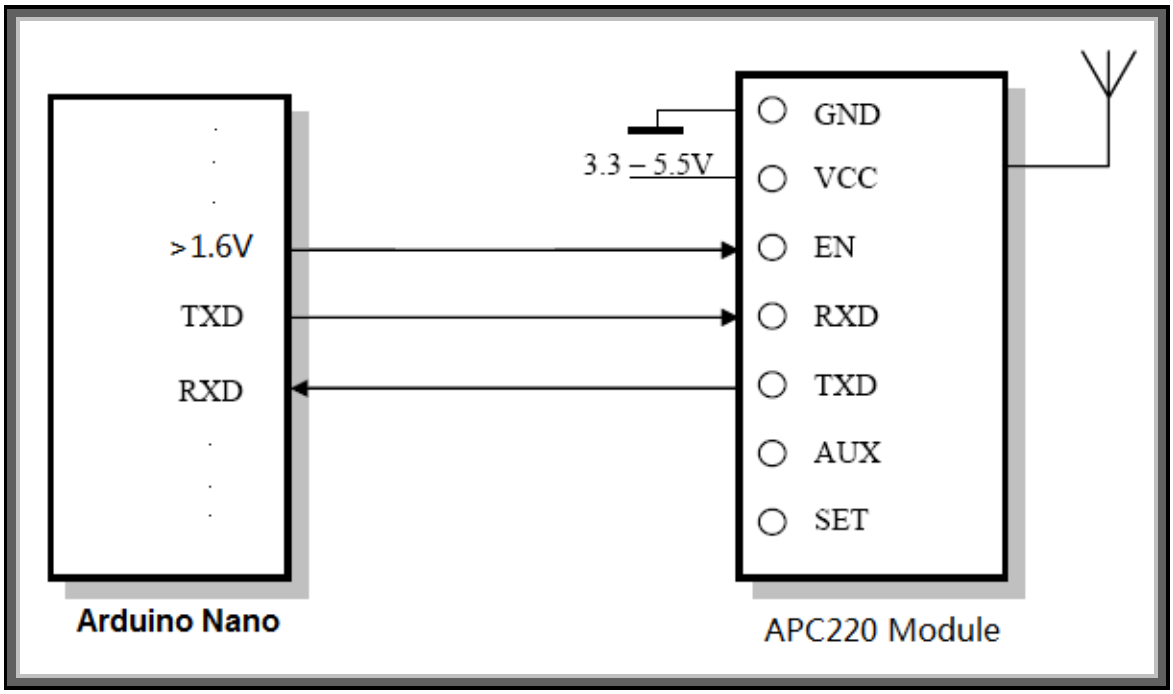

Figura 10 – Funcionamento geral do módulo.

3.2.3. IMPLEMENTAÇÃO

A implementação do Arduino Nano foi desenvolvida em linguagem C, onde este basicamente aguarda a chegada de requisições vindas da central via RF, para enviar uma resposta do que foi solicitado, no nosso caso o retorno da temperatura ambiente, que é fornecida pelo LM35. Circuitos e códigos referentes a esta implementação pode ser encontrado na seção de anexos (7.1).

3.3 CIRCUITO DE ACIONAMENTO

O Circuito de Acionamento é composto por um MOC 3021 e um circuito de detecção de corrente, que serão detalhados logo abaixo.

3.3.1 MOC3021

O MOC3023 é um opto acoplador que tem como função isolar a parte de potência de um circuito, da parte mais sensível a grandes tensões. É composto basicamente de um LED emissor e um DIAC (diodo que conduz dos dois lados). É de saída triac por que permite o acionamento de um triac em sua saída.

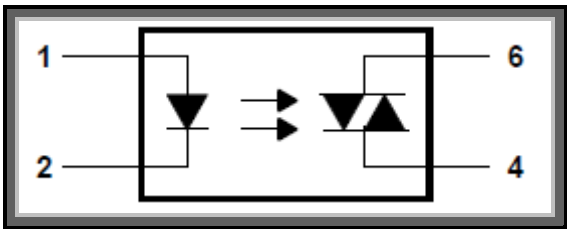

Figura 11 – MOC3023.

3.3.2. IMPLEMENTAÇÃO

A implementação do circuito de acionamento foi feita através da criação de uma placa de circuito impresso onde esta tem a função de fazer o opto acoplamento do circuito de alta potencia no caso a lâmpada, do Seeduino que trabalha com baixa potencia, foi desenvolvido este mecanismo para uma questão de isolamento dos dispositivos.

Na implementação utilizamos dois resistores, um de 270 ohms e um de 180 ohms, um MOC 3023, um BT136 e dois conectores. Como demonstrado o esquemático baixo.

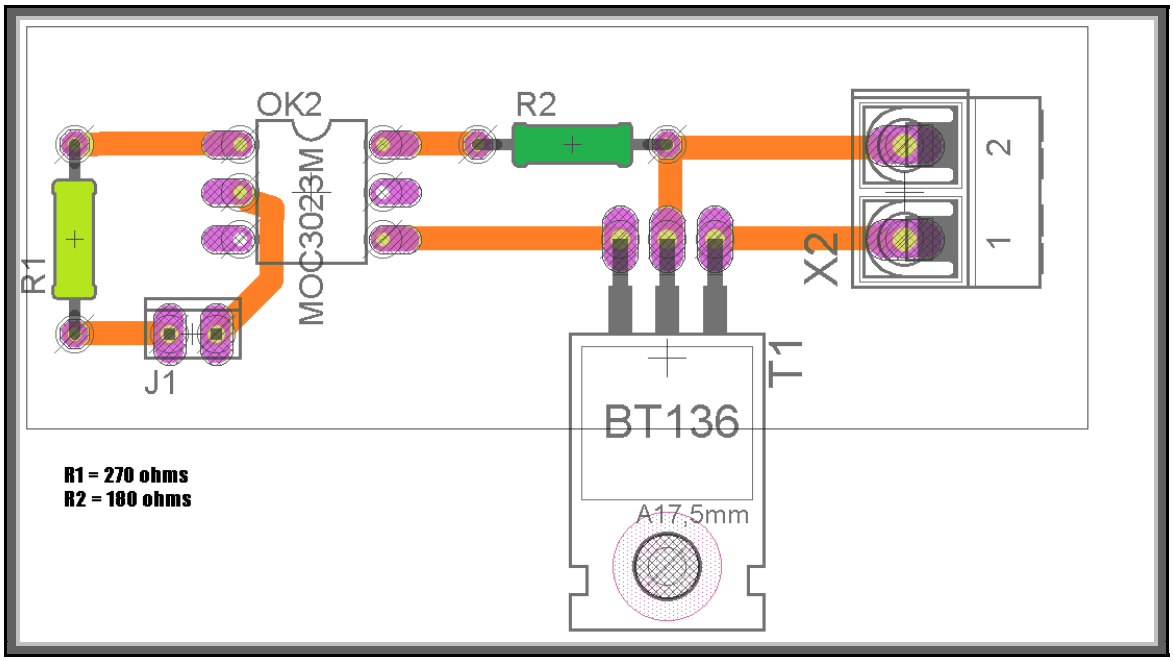

Figura 12 – Circuito de Acionamento.

3.4. PROCEDIMENTOS DE INTEGRAÇÃO

A integração dos módulos, central e atuadores, foram feitos através da validação das informações transmitidas através das antenas RF. Pois a partir destas informações obtidas conseguimos identificar possíveis erros de comunicação entre os módulos e resolve-los para uma integração concreta dos módulos.

4. TESTES

4.1. TESTE DA CENTRAL

O teste com a central engloba todo o funcionamento da placa “Seeeduino” juntamente com o “Ethernet Shield” e o dispositivo RF. A programação de acionamento e gerenciamento foi feita em linguagem C, pois o processador do Seeeduino suporta este tipo de linguagem, e para a interface com o usuário foi desenvolvida uma pagina em HTML esta onde o cliente pode encontrar seus dispositivos de atuação online. O dispositivo RF foi configurado para receber informações vindas do servidor web e transmiti-las para a outra antena que estará conectada ao atuador.

Os testes realizados foram os seguintes com suas descrições e suas respostas:

→ **Teste de gravação do processador (Seeeduino):** Configuramos uma COM Virtual para que pudesse ser feita a programação, em linguagem C, no processador ATMEGA, utilizando um compilador próprio do hardware. Tendo como satisfatório o resultado obtido. A figura abaixo mostra que a gravação para o processador foi executada com sucesso.

Figura 13 – Confirmação da gravação no processador ATMEGA.

→ **Teste de validação do servidor:** Após a codificação do servidor web, em linguagem C, a maneira que encontramos para valida-lo foi utilizar o prompt de comandos do Windows, e nele utilizamos o comando “ping”, com

isso verificamos a resposta obtida pelo servidor. A figura abaixo é o trecho de código implementado onde podemos verificar o MAC address e o IP que foram setados no dispositivo para que ele pertença a uma rede local.

```
static uint8_t mymac[6] = {0x54,0x55,0x58,0x10,0x00,0x24};
static uint8_t myip[4] = {192,168,1,115};
static char baseurl[]="http://192.168.1.115/";
static uint16_t mywwwport =80; // listen port for tcp/www (max range 1-254)
```

Figura 14 – Parte da codificação do Servidor WEB.

Podemos fazer a confirmação pela figura abaixo que a resposta do comando “ping” ao IP previamente setado foi validada.

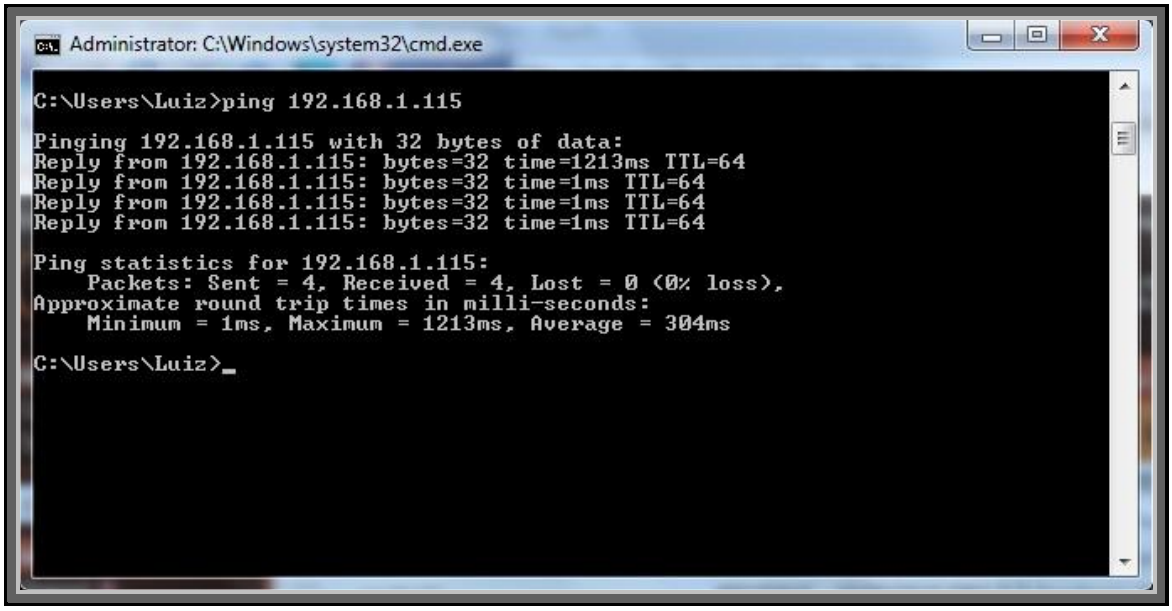

Figura 15 – Prompt de comando com a resposta do “ping”.

→ **Teste das Antenas RF:** O teste realizado com as antenas RF foi feito da seguinte maneira: conectamos o dispositivo RF(mestre) ao Ethernet Shield, e o dispositivo RF(escravo) conectado a um computador qualquer, apos a interação do cliente com a pagina web e subseqüente com o servidor web, programamos o processador do Seeeduino para enviar via serial a informação pela sua porta de saída. O dispositivo RF (mestre) estando ligado ao Ethernet Shield recebe essa informação e envia o sinal via radio freqüência até o outro dispositivo RF(escravo), ligada ao computador a qual recebera os dados transmitidos e demonstrara o recebimento da informação em um serial

monitor(HyperTerminal) do próprio compilador do Arduino. O teste de validação foi satisfatório, conforme a figura abaixo.

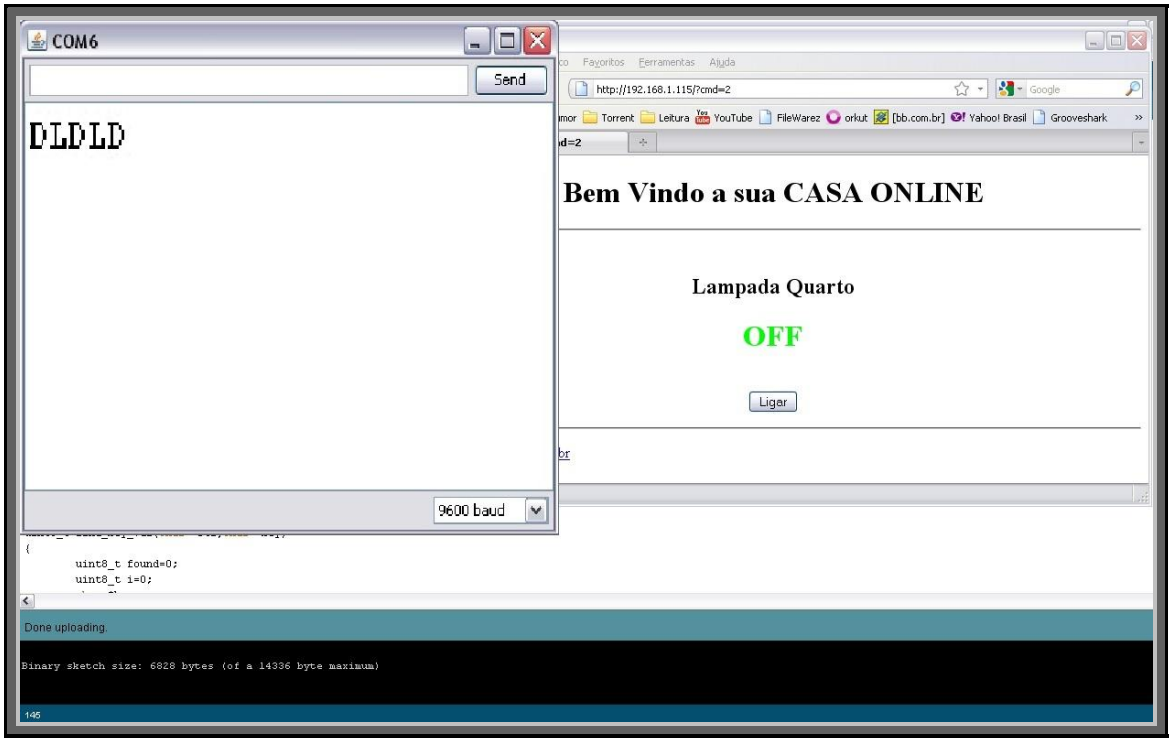

Figura 16 – Pagina WEB e Janela de monitoração.

→ **Teste da página WEB:** O teste que foi aplicado na página WEB foi feito visando uma possível incompatibilidade do mesmo com 3 browsers disponíveis no mercado (Internet Explorer, Chrome, Firefor), procuramos testar a resposta da página em cada um deles. Ainda testamos os botões disponíveis na página web, para isso criamos macros que simulavam os cliques do mouse, para confirmar a eficácia da resposta e a confiança na informação enviada. Em ambos os testes a resposta obtida foi satisfatória. A figura abaixo mostra o teste no browsers.

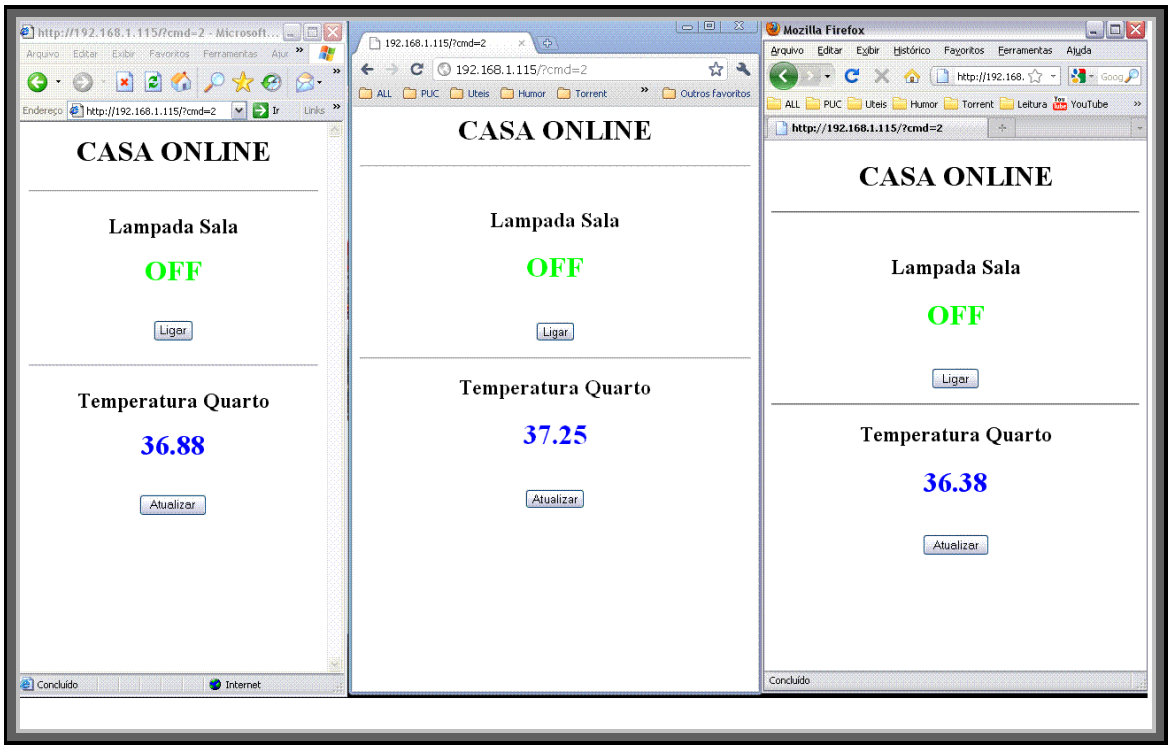

Figura 17 – Compatibilidade da página WEB com Browsers.

→ **Teste dos dispositivos em conjunto:** O teste final da central foi feito com todos os módulos unidos, são eles: Seeeduino, Ethernet Shield, Antenas RF, e todas as codificações como pagina HTML, Servidor WEB e programação do processador ATMEGA. Conseguimos fazer o Ethernet Shield que possui o servidor web rodando internamente nele, e a pagina HTML, serem acessados por um computador externo dentro da rede local e com isso o cliente interagir nos atuadores que estavam codificados no Seeeduino, o teste da antena RF foi feita simulada em um serial monitor do próprio compilador Arduino como mencionado acima. O resultado obtido com a integração destas foi o esperado, pois todas se comunicaram com perfeição.

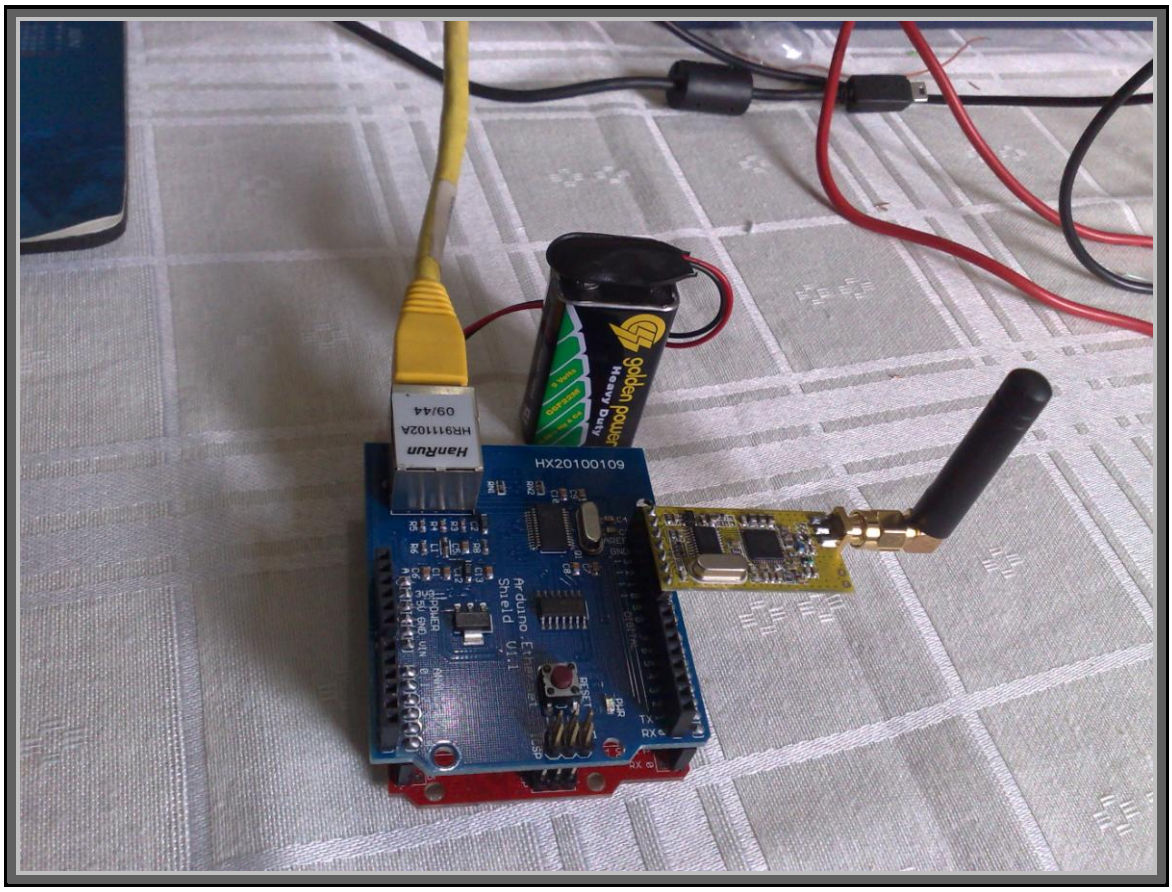

Figura 18 – Módulos da central conectados.

4.2. TESTES DO ATUADOR - LAMPADA

Os testes aplicados neste atuador foram para verificar a consistência na resposta do atuador quando acionado com a informação proveniente da pagina WEB, conseqüentemente do desejo do usuário, e para isto utilizamos a mesma macro criada para testar os botões na pagina WEB. Assim conseguimos simular varias vezes as informações sendo enviadas para o dispositivo e analisar o seu comportamento. A resposta obtida foi satisfatória.

4.3. TESTES DO ATUADOR - TEMPERATURA

Os testes aplicados neste atuador foram para verificar a consistência da informação transmitida por ele, e para isto sobrecarregamos o envio da informação e constatamos que ele não perde a consistência mesmo depois de muito tempo de funcionamento. Outro teste feito foi referente ao possível envio de “lixo” para o servidor, e mais uma vez a implementação se comportou como o esperado. Testamos ainda possíveis interferências externas nas antenas,

simulado com outros instrumentos que trabalham com radio freqüência, e mais uma vez foi constatado um ótimo desempenho do atuador.

4.4. TESTE GERAL DO PROJETO

Para finalizar os testes foram integrados todos os módulos, e a partir de um computador externo, conseguimos simular todas as funcionalidades propostas no escopo do projeto, tais como:

→ Acessar a pagina WEB de outro computador.

→ Envio de requisições do servidor para os atuadores.

→ Processamento da informação com consistência.

→ Servidor ativo.

→ Pagina WEB compatível com navegadores.

→ Comunicação sem perda de dados entre as antenas RF.

→ Resposta precisa dos atuadores.

→ Tempo de resposta do sensor de temperatura.

→Velocidade de envio das informações entre os módulos.

→Confiabilidade dos hardwares utilizados.

→Inclusão da Central em uma rede local.

→Mobilidade dos atuadores.

4.5. TABELA DE TESTES E VALIDAÇÕES

Tipos de Testes/Validações	Resultado Esperado	Resultado Obtido
Teste de gravação do processador (Seeeduino)	O Processador ATMEGA reconhecer e executar com a codificação feita em C.	Satisfatório
Teste de validação do servidor	O servidor web estar ativo e respondendo as interações do cliente.	Satisfatório

Teste das Antenas RF	As antenas RF trocarem informações sem perda de dados.	Satisfatório
Confiabilidade dos hardwares utilizados	Hardwares se comportarem como previsto na documentação do fabricante.	Satisfatório
Inclusão da Central em uma rede local	Central adquirir IP previamente setado.	Satisfatório
Teste dos dispositivos em conjunto	Todos os módulos integrados e se comunicando.	Satisfatório

Tabela 1 – Testes e Validações.

5. CONCLUSÃO

Este documento apresentou detalhadamente o estudo sobre automação residencial via web. Demonstramos as etapas que serão desenvolvidas uma a uma com especificações, diagrama de blocos, circuitos e pinagens, também foram esclarecido os tipos de testes de validação que serão executados em cada parte do projeto, sempre visando o bom funcionamento de cada módulo e buscando um produto final de qualidade e consistência.

Com o inicio da fase de integração dos módulos, detectamos que o hardware escolhido juntamente com sua tecnologia, no caso o kit de desenvolvimento MSP430-RF2500 que implementa o protocolo SimpliciTi, não atenderiam as especificações iniciais do projeto, a grande dificuldade foi na integração dos módulos, pois como planejado cada módulo foi desenvolvido individualmente, assim de uma maneira pouco agradável aprendemos com ocorrido.

Como previsto anteriormente na tabela de riscos, se a fase de integração dos módulos não fosse satisfatória, medidas de contingência seriam adotadas. A medida adotada foi à alteração do escopo, mais especificamente a mudança de todo o hardware do projeto.

Após pesquisas mais aprofundadas em hardwares que poderiam suprir as especificações do projeto, decidimos então adotar a tecnologia Arduino, dentro desta definimos três novos módulos: Seeeduino, Arduino Ethernet Shield, e para a comunicação com os módulos atuadores será usado o dispositivo RF(APC220).

A mudança de hardware traz benefícios ao projeto, pois um grande diferencial deste novo produto é que seus conectores possuem uma forma padrão, permitindo o Seeeduino ser interligado a outros módulos expansivos, conhecidos como Shields.

Após toda a migração de hardware, foi iniciada toda a parte de desenvolvimento do protótipo, onde conseguimos validar e testar grande parte dos itens propostos no documento.

Na etapa de conclusão do projeto, tivemos alguns imprevistos com a integração do módulo atuador, pois o processador escolhido para fazer essa integração não foi satisfatória. Sendo assim optamos por trocar o processador AT89C2051, por um Arduino Nano(ATMEGA 328) que supriu todas as nossas necessidades.

Outro imprevisto que é muito importante citar, é a pouca memória disponível no processador da Central(Seeeduino), este que por sua vez é responsável por criar o pacote TCP a ser enviado, esse pacote fica limitado ao tamanho da memória disponível (1KB), ou seja, há um limite de informações que podem ser enviados de uma única vez. Isto limita a quantidade de atuadores que podem ser controlados pela central, visto que todas as informações dos atuadores são inclusas nesses pacotes.

Algumas melhorias futuras seriam a escolha de um processador que tivesse uma capacidade maior de memória ou até a inclusão de uma memória externa, para solucionar o problema.

A equipe que desenvolveu o presente projeto conclui que o mesmo atingiu os objetivos propostos e declarados no escopo do projeto, claro com algumas alterações necessárias e previstas.

Esse projeto serviu como uma grande experiência e aprendizado para os membros dessa equipe, onde aprendemos a lidar com prazos, riscos funcionais do projeto, medidas de contingencia e o mais importante de tudo entender que planejar um projeto desse porte, requer muito estudo e conhecimento das tecnologias existentes, e além disso gera uma imensa responsabilidade para os desenvolvedores.

Face ao exposto documento terminamos este projeto com a sensação e o sentimento de dever comprido.

6. REFERÊNCIAS BIBLIOGRÁFICAS

[1] BOLZANI, Caio Augustus M. - Residências Inteligentes. São Paulo: Livraria da Física, 2004.

[2] CHAMUSCA, Alexandre - Domotica & Segurança Electrónica. Ingenium Ed. 2006.

[3] Home Automation Superstore. Disponível em:
<http://www.smarthome.com/_/index.aspx>
Acesso em: 13 de Abril de 2010.

[4] Introduction to SimpliciTI. Disponível em:
<http://focus.ti.com/lit/ml/swru130b/swru130b.pdf>
Acesso em: 12 de Abril de 2010.

[5] Texas Instruments. Disponível em:
<http://www.ti.com/lprf>
Acesso em: 12 de Maio de 2010.

[6] Associação Brasileira de Automação Residencial. Disponível em:
< http://www.aureside.org.br/default.asp>
Acesso em: 18 de Abril de 2010.

[7] Apache Server. Disponível em:
<http://httpd.apache.org >
Acesso em: 05 de Abril de 2010.

[8] Apache Server. Disponível em:
<http://sites.google.com/site/mcolepicolo/simplici>
Acesso em: 01 de Junho de 2010.

[9] Arduino Home Page. Disponível em:
<http://arduino.cc/>
Acesso em: 16 de Agosto de 2010.

[10] Hello! Seeeduino. Disponível em:
<http://seeedstudio.com/depot/datasheet/Hello_Seeeduino.pdf/>
Acesso em: 25 de Agosto de 2010.

[11] Language Reference. Disponível em:
<http://arduino.cc/en/Reference/HomePage>
Acesso em: 01 de Agosto de 2010.

[12] Web Server. Disponível em:
<http://arduino.cc/en/Tutorial/WebServer>
Acesso em: 20 de Outubro de 2010.

[13] Arduino Nano. Disponível em:
<http://www.arduino.cc/en/Main/ArduinoBoardNano>
Acesso em: 20 de Outubro de 2010.

[14] Serial. Disponível em:
<http://arduino.cc/en/Reference/Serial>
Acesso em: 22 de Outubro de 2010.

[15] Manual do usuário. Disponível em:
<http://www.arduino.cc/en/uploads/Main/ArduinoNanoManual23.pdf>
Acessado em: 22 de Outubro de 2010.

[16] Arduino Development Environment. Disponivel em:
<http://www.arduino.cc/en/Guide/Environment>
Acessado em: 22 de Outubro de 2010.

[17] Arduino Nano and Ethernet Shield. Disponível em:
<http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1247619986>
Acessado em: 22 de Outubro de 2010.

[18] Arduino Playground. Disponivel em:
<http://www.arduino.cc/playground/Code/InfraredReceivers>
Acessado em: 24 de Outubro de 2010.

[19] Normas ABNT. Disponível em:
<http://www.abnt.org.br/>
Acessado em: 03 de Novembro de 2010.

[20] Seeeduino & Scematic or Board diagram. Disponivel em:
<http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1239283798>
Acessado em: 01 de Agosto de 2010.

[21] Figura 1. Disponível em:
<http://todaoferta.uol.com.br/comprar/apostila-automacao-residencial-casa-inteligente-380-pags-8IX9PC3Z0R#rmcl>
Acessado em: 12 de Maio de 2010.

[22] Figura 4. Disponível em:
<http://seeedstudio.com/depot/datasheet/Hello_Seeeduino.pdf>
Acessado em: 20 de Agosto de 2010.

[23] Figura 5. Disponível em:
< http://www.arduino.cc/en/Main/ArduinoEthernetShield>
Acessado em: 20 de Agosto de 2010.

[24] Figura 6. Disponível em:
< http://www.arduino.cc/en/Main/ArduinoEthernetShield>
Acessado em: 20 de Agosto de 2010.

[25] Figura 7. Disponível em:
< http://www.arduino.cc/en/Main/ArduinoEthernetShield>
Acessado em: 20 de Agosto de 2010.

[26] Figura 9. Disponível em:
<http://www.arduino.cc/en/Main/ArduinoBoardNano>
Acessado em: 30 de Outubro de 2010.

7. ANEXOS

7.1. CODIFICAÇÃO SERVIDOR WEB

```
#include <SoftwareSerial.h>
#include "etherShield.h"
static uint8_t mymac[6] = {0x54,0x55,0x58,0x10,0x00,0x24};
static uint8_t myip[4] = {192,168,1,115};
static char baseurl[]="http://192.168.1.115/";
static uint16_t mywwwport =80; // listen port for tcp/www (max range 1-254)
#define BUFFER_SIZE 500
static uint8_t buf[BUFFER_SIZE+1];
#define STR_BUFFER_SIZE 22
static char strbuf[STR_BUFFER_SIZE+1];
EtherShield es=EtherShield();
// prepare the webpage by writing the data to the tcp send buffer
uint16_t print_webpage(uint8_t *buf, byte on_off);
int8_t analyse_cmd(char *str);
// LED cathode connects the Pin4, anode to 5V through 1K resistor
#define LED_PIN  4
#define rxPin 0
#define txPin 1
SoftwareSerial mySerial =  SoftwareSerial(rxPin, txPin);
int conf;
char temperatura[10];
unsigned int recebeu_temp;

void setup(){
        /*initialize enc28j60*/
             es.ES_enc28j60Init(mymac);
        es.ES_enc28j60clkout(2); // change clkout from 6.25MHz to 12.5MHz
        delay(10);
        es.ES_enc28j60PhyWrite(PHLCON,0x880);
            delay(500);
            es.ES_enc28j60PhyWrite(PHLCON,0x990);
            delay(500);
            es.ES_enc28j60PhyWrite(PHLCON,0x880);
            delay(500);
            es.ES_enc28j60PhyWrite(PHLCON,0x990);
            delay(500);
            es.ES_enc28j60PhyWrite(PHLCON,0x476);
            delay(100);
       //init the ethernet/ip layer:
       es.ES_init_ip_arp_udp_tcp(mymac,myip,80);
```

```
            pinMode(LED_PIN, OUTPUT);
            digitalWrite(LED_PIN, LOW);  // switch on LED
        // start serial port at 9600 bps:
        pinMode(rxPin, INPUT);
        pinMode(txPin, OUTPUT);
        Serial.begin(9600);
        recebeu_temp = 0;
        delay(500);
 }
void loop(){
  uint16_t plen, dat_p;
  int8_t cmd;
  byte on_off = 1;
  leTemperatura();
/*----- TCP ------*/
  plen = es.ES_enc28j60PacketReceive(BUFFER_SIZE, buf);
    /*plen will ne unequal to zero if there is a valid packet (without crc error) */
    if(plen!=0){
    // arp is broadcast if unknown but a host may also verify the mac address by sending it to a
unicast address.
    if(es.ES_eth_type_is_arp_and_my_ip(buf,plen)){
      es.ES_make_arp_answer_from_request(buf);
      return;
    }
    // check if ip packets are for us:
    if(es.ES_eth_type_is_ip_and_my_ip(buf,plen)==0){
      return;
    }
    if(buf[IP_PROTO_P]==IP_PROTO_ICMP_V &&
buf[ICMP_TYPE_P]==ICMP_TYPE_ECHOREQUEST_V){
      es.ES_make_echo_reply_from_request(buf,plen);
      return;
    }
    // tcp port www start, compare only the lower byte
    if (buf[IP_PROTO_P] == IP_PROTO_TCP_V&&buf[TCP_DST_PORT_H_P] ==
0&&buf[TCP_DST_PORT_L_P] == mywwwport){
      if (buf[TCP_FLAGS_P] & TCP_FLAGS_SYN_V){
        es.ES_make_tcp_synack_from_syn(buf); // make_tcp_synack_from_syn does already send
the syn,ack
        return;
      }
      if (buf[TCP_FLAGS_P] & TCP_FLAGS_ACK_V){
        es.ES_init_len_info(buf); // init some data structures
        dat_p=es.ES_get_tcp_data_pointer();
        if (dat_p==0){ // we can possibly have no data, just ack:
```

```
        if (buf[TCP_FLAGS_P] & TCP_FLAGS_FIN_V){
          es.ES_make_tcp_ack_from_any(buf);
        }
        return;
      }
      if (strncmp("GET ",(char *)&(buf[dat_p]),4)!=0){
         // http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
         plen=es.ES_fill_tcp_data_p(buf,0,PSTR("HTTP/1.0 200 OK\r\nContent-Type:
text/html\r\n\r\n<h1>200 OK</h1>"));
         goto SENDTCP;
      }
           if (strncmp("/ ",(char *)&(buf[dat_p+4]),2)==0){
             plen=print_webpage(buf, on_off);
         goto SENDTCP;
       }
      cmd=analyse_cmd((char *)&(buf[dat_p+5]));
      if (cmd==2){
            on_off=1;
           digitalWrite(LED_PIN, LOW);  // switch on LED
            Serial.print('L', BYTE);
            }
      else if (cmd==3){
            on_off=0;
           digitalWrite(LED_PIN, HIGH);  // switch off LED
            Serial.print('D', BYTE);
      }
      plen=print_webpage(buf, on_off);
SENDTCP:  es.ES_make_tcp_ack_from_any(buf); // send ack for http get
         es.ES_make_tcp_ack_with_data(buf,plen); // send data
     }
   }
  }
}
// The returned value is stored in the global var strbuf
uint8_t find_key_val(char *str,char *key){
      uint8_t found=0;
      uint8_t i=0;
      char *kp;
      kp=key;
      while(*str &&  *str!=' ' && found==0){
            if (*str == *kp){
                  kp++;
                  if (*kp == '\0'){
                        str++;
                        kp=key;
```

```
                        if (*str == '='){
                                found=1;
                        }
                }
        }else{
                kp=key;
        }
        str++;
    }
    if (found==1){
        // copy the value to a buffer and terminate it with '\0'
        while(*str &&  *str!=' ' && *str!='&' && i<STR_BUFFER_SIZE){
                strbuf[i]=*str;
                i++;
                str++;
        }
        strbuf[i]='\0';
    }
    return(found);
}
int8_t analyse_cmd(char *str){
    int8_t r=-1;

    if (find_key_val(str,"cmd")){
        if (*strbuf < 0x3a && *strbuf > 0x2f){
                // is a ASCII number, return it
                r=(*strbuf-0x30);
        }
    }
    return r;
}
/*-----WEB SERVER------*/
uint16_t print_webpage(uint8_t *buf, byte on_off){
  int i=0;
    uint16_t plen;
    plen=es.ES_fill_tcp_data_p(buf,0,PSTR("HTTP/1.0 200 OK\r\nContent-Type:
text/html\r\n\r\n"));
    plen=es.ES_fill_tcp_data_p(buf,plen,PSTR("<center><p><h1>CASA ONLINE</h1></p> "));
    plen=es.ES_fill_tcp_data_p(buf,plen,PSTR("<hr><br><form METHOD=get action=\""));
    plen=es.ES_fill_tcp_data(buf,plen,baseurl);
    plen=es.ES_fill_tcp_data_p(buf,plen,PSTR("\">"));
    plen=es.ES_fill_tcp_data_p(buf,plen,PSTR("<h2>Lampada Sala</h2>"));
      plen=es.ES_fill_tcp_data_p(buf,plen,PSTR("<h1><font color=\"#00FF00\">"));
    if(on_off)
      plen=es.ES_fill_tcp_data_p(buf,plen,PSTR("OFF"));
```

```
        else
            plen=es.ES_fill_tcp_data_p(buf,plen,PSTR("ON"));

        plen=es.ES_fill_tcp_data_p(buf,plen,PSTR("</font></h1><br> ") );
        if(on_off){
            plen=es.ES_fill_tcp_data_p(buf,plen,PSTR("<input type=hidden name = cmd
value=3>"));
            plen=es.ES_fill_tcp_data_p(buf,plen,PSTR("<input type=submit value =
\"Ligar\"></form>"));
        }
        else {
            plen=es.ES_fill_tcp_data_p(buf,plen,PSTR("<input type=hidden name = cmd
value=2>"));
            plen=es.ES_fill_tcp_data_p(buf,plen,PSTR("<input type=submit value =
\"Desligar\"></form>"));
        }
        plen=es.ES_fill_tcp_data_p(buf,plen,PSTR("<hr>"));
         plen=es.ES_fill_tcp_data_p(buf,plen,PSTR("<h2>Temperatura Quarto</h2><h1><font
color=\"#0000FF\">"));
        i=0;
        while (temperatura[i] != '\0') {
              buf[TCP_CHECKSUM_L_P+3+plen]=temperatura[i++];
              plen++;
        }
        plen=es.ES_fill_tcp_data_p(buf,plen,PSTR("</h1><br><button
onclick=\"javascript:location.reload(true)\">Atualizar</button></center>"));
        return(plen);
}
void leTemperatura(){
  int pos=0;
  if(recebeu_temp < 1){
    recebeu_temp++;
    Serial.print('H');
    delay(200);
  }
  else if( recebeu_temp == 1){
    recebeu_temp++;
    if (Serial.available() > 0){
      conf = Serial.read();
      while(conf != 'T'){
        conf = Serial.read();
      }
      conf = Serial.read();
      while(conf != '&'){
        temperatura[pos++] = conf;
```

```
        conf = Serial.read();
      }
      temperatura[pos] = '\0';
      recebeu_temp = 0;
    }
  }
  else if ( recebeu_temp < 9999999 ){
    recebeu_temp++;
  }
  else {
    recebeu_temp = 0;
  }
}
```

7.2. CODIFICAÇÃO ARDUINO NANO

```
#include <SoftwareSerial.h>
#define LED_PIN 2
#define rxPin 0
#define txPin 1
SoftwareSerial mySerial =  SoftwareSerial(rxPin, txPin);
int incomingByte = 0;
int pin = 0; // analog pin
float tempc = 0; // temperature variables
int samples[8]; // variables to make a better precision
int i;

void setup(){
pinMode(rxPin, INPUT);
pinMode(txPin, OUTPUT);
Serial.begin(9600); // start serial communication
pinMode(LED_PIN, OUTPUT);
digitalWrite(LED_PIN, HIGH);
}
void loop(){
  if (Serial.available() > 0) {
  // read the incoming byte:
  incomingByte = Serial.read();
    if (incomingByte == 'H') { //verifica se a informação é igual a H e se for:
      printTemp(leTemp());
      tempc = 0;
    }
  }
}
float leTemp(){
```

```
for(i = 0; i<=7; i++){ // gets 8 samples of temperature

  samples[i] = ( 5.0 * analogRead(pin) * 100.0) / 1024.0;
  tempc = tempc + samples[i];
  delay(100);
}
tempc = tempc/8.0; // better precision
return tempc;
}
void printTemp(float tempCelcius){
  Serial.print('T');
  Serial.print(tempCelcius,2);
  Serial.print('&');
}
```