**UNIVERSIDADE DO ESTADO DE SANTA CATARINA**
**CENTRO DE CIÊNCIAS TECNOLÓGICAS**
**DEPARTAMENTO DE ENGENHARIA ELÉTRICA**
**BACHARELADO EM ENGENHARIA ELÉTRICA**

**CIRO MANARIN NUNES DE SOUZA**

# SISTEMA DE SEGURANÇA RESIDENCIAL UTILIZANDO A PLATAFORMA ARDUINO E O SUPERVISÓRIO ELIPSE E3

**JOINVILLE – SC**
**2016**

**UNIVERSIDADE DO ESTADO DE SANTA CATARINA**
**CENTRO DE CIÊNCIAS TECNOLÓGICAS**
**DEPARTAMENTO DE ENGENHARIA ELÉTRICA**
**BACHARELADO EM ENGENHARIA ELÉTRICA**

**CIRO MANARIN NUNES DE SOUZA**

# SISTEMA DE SEGURANÇA RESIDENCIAL UTILIZANDO A PLATAFORMA ARDUINO E O SUPERVISÓRIO ELIPSE E3

Relatório final da disciplina de TCC-II submetido ao Curso de Bacharelado em Engenharia Elétrica, do Centro de Ciências Tecnológicas, da Universidade do Estado de Santa Catarina, como requisito parcial para obtenção do grau de Bacharelado de Engenharia Elétrica.

**Orientador:** Prof. Marcos Fergütz.

**Co-orientador:** Prof. Marco Shawn Meireles Machado.

**JOINVILLE – SC**
**2016**

**CIRO MANARIN NUNES DE SOUZA**

**SISTEMA DE SEGURANÇA RESIDENCIAL UTILIZANDO A PLATAFORMA ARDUINO E O SUPERVISÓRIO ELIPSE E3**

**Este trabalho foi julgado e aprovado em sua forma final, sendo assinado pelos professores da Banca Examinadora.**

**Joinville, 05 de Dezembro de 2016**

_____

**Prof. Marcos Fergütz**

_____

**Prof. Marco Shawn Meireles Machado**

_____

**Prof. Fabrício Noveletto**

Dedico este trabalho a toda minha família e amigos, pelo apoio e confiança que a mim foram direcionados.

## AGRADECIMENTOS

Nem que mil páginas de agradecimento fossem escritas, eu estaria agradecendo o suficiente, logo, deixo meus agradecimentos a todos aqueles que passaram de alguma forma pela minha vida e me direcionaram para esse momento.

Meu muito obrigado aos professores Marcos Fergütz e Marco Shawn, por terem aceitado a difícil tarefa de me orientar nesse trabalho, tarefa essa que concluíram com excelência.

O agradecimento a minha família começa pelo meu pai, Cid Nunes de Souza, que mesmo morando em outro país, nunca mediu esforços, para que nada faltasse na minha vida, e juntamente com minha mãe, Ronise Manarin, moldaram o meu caráter e nunca me deixaram desistir, mesmo que as vezes desistir parecesse a melhor saída. Obrigado as minhas irmãs, Malu e Luna, que suportaram a ausência do irmão, e muitas vezes me acolheram nos momentos em que precisava voltar pra casa. Aos meus avós, Nira, Amábile e Nilton, aos meus tios Niltinho e Caco, as minhas tias Graci e Soraia, aos primos Natália, Larissa e Wagner meu muito obrigado. E a família que mesmo não sendo de sangue, é família, Rodrigo, Ellen e tio João, meus sinceros agradecimentos.

Meu muito obrigado a Giovanna Sponchiado, não só pela ajuda direta neste trabalho, mas também pelo conforto quando tudo parecia estar desmoronando.

Ao amigo Alcides Taschini Junior, que me despertou o interesse pela engenharia, e me ensinou e apoiou muito, de inúmeras maneiras, o meu obrigado.

Aos colegas de faculdade, Jefson, Giovanni, Bruna Guelere, Adair, Rômulo, Tiago Yoshida, Marco, Guilherme Bento, Eduardo Wilsen, Verônica, Henrique Nunes, e todos os demais que de alguma forma acompanharam as batalhas diárias nas salas de aula, meu agradecimento. Ao colega de curso Kevin Matos de Sá, que se tornou ao longo de 5 anos dividindo as contas, não só um colega, mas um irmão, meu obrigado. Um muito obrigado a Cássia Cruz Luiz, que foi além de colega de curso, minha companheira durante grande parte da graduação, e hoje se tornou uma pessoa ímpar na minha vida.

Aos amigos que ficaram em Urussanga, Bruno, Filippe, Pelota, Zibu e Rapha, do time Baduga, meu muito obrigado. Aos que ficaram em Jaguaruna, Arno, Iryo, Murilo, Rodolfo, Saul, Baiano e Xico, do time Muritasha, meu muito obrigado.

## RESUMO

O crescimento da violência durante os anos aumentou também a preocupação da população com a sua segurança e dos seus bens materiais. O objetivo principal desse trabalho é desenvolver um sistema completo de segurança residencial, utilizando as ferramentas disponíveis na automação. O controle principal do sistema é executado em uma plataforma Arduino, e um sistema supervisório que informa a situação atual ao usuário. O projeto foi dividido em três partes, uma revisão bibliográfica sobre todos os recursos utilizados na sua execução. O desenvolvimento do projeto em si, com a programação do microcontrolador e do supervisório. E, por fim, a montagem de um protótipo que simule uma residência e permita implementar o sistema. Assim foi desenvolvido um sistema de segurança robusto e facilmente adaptável a qualquer residência, e que pode ser modificado para as diversas situações de acordo com cada necessidade.



**Palavras-chave**: Segurança, Alarme, Arduino, Automação, Supervisório.

## ABSTRACT

The increase in violence over the years has also increased the population's concern for their safety and material assets. The main objective of this work is to develop a complete residential security system, using the tools available in automation. The main system control runs on an Arduino platform, and a supervisory system that informs the current situation to the user. The project was divided into three parts, a bibliographic review of all the resources used in its execution. The development of the project itself, with microcontroller and supervisory programming. And, finally, the assembly of a prototype that simulates a residence and allows to implement the system. Thus a robust security system was developed and easily adaptable to any residence, and that can be modified for the different situations according to each need.



**Keywords**: Security, Alarm, Arduino, Automation, Supervisory.

# LISTA DE ILUSTRAÇÕES

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| SCADA | Supervisory Control and Data Acquisition |
| FTDI | Future Technology Devices International |
| USB | Universal Serial Bus |
| SMD | Surface Mount Device |
| PWM | Pulse Width Modulation |
| RISC | Reduced Instruction Set Computer |
| CLP | Controlador Lógico Programável |
| RTC | Real Time Clock |

## SUMÁRIO

# 1 INTRODUÇÃO

O uso da tecnologia para o bem da sociedade sempre será válido, e o avanço dessa tecnologia vem permitindo cada vez mais o seu uso com essa finalidade. Os microcontroladores são um exemplo de tecnologia, que tomou conta dos projetos eletrônicos, seja de maneira acadêmica ou comercial. O objetivo desse projeto é desenvolver um sistema de alarme automatizado, que tem o monitoramento de abertura de portas e janelas, controle de iluminação externa e interna, interface com o usuário através de um computador e um dispositivo de alarme em caso de invasão.

A lógica de programação foi desenvolvida utilizando a plataforma Arduino, através da placa Mega 2560. Toda a iluminação, sensores das aberturas, botões de seleção, e sirene de aviso são entradas e saídas da placa. A comunicação da placa com o computador é feita a partir da *shield* Ethernet W5100. Utilizando o protocolo TCP, a placa permite que o Arduino seja conectado a rede, dessa forma é possível interligar o microcontrolador ao sistema supervisório.

No supervisório Elipse E3, que é um sistema SCADA, foi projetada a interface de comunicação com o usuário. Essa interface mostra o estado atual da iluminação e das portas e janelas da casa, e também permite que o usuário interaja com algumas funcionalidades do sistema.

## 2 OBJETIVOS

Objetivo Geral: Projetar e implementar um sistema de segurança residencial, utilizando como controlador uma placa de plataforma Arduino, e um sistema supervisório utilizando o *software* Elipse E3.

Objetivos Específicos:

- Desenvolver e aprimorar conhecimentos na programação em linguagem C, utilizada na IDE da plataforma Arduino;

- Desenvolver e aprimorar conhecimentos na montagem de sistemas supervisórios, utilizando o supervisório Elipse E3;

- Adquirir conhecimentos na área de comunicação via protocolo Modbus TCP;

- Utilizar os conhecimentos de montagem de circuitos para desenvolver um protótipo que simule as condições de uma residência.

## 3 ARDUINO

O projeto Arduino começou na cidade de Ivrea, na Itália, no ano de 2005 (ARDUINO... 2010). Na ocasião, um grupo de pesquisadores decidiu desenvolver uma plataforma que fosse mais moderna que as atuais, e principalmente, que fossem financeiramente mais acessíveis aos estudantes, já que na época as plataformas mais utilizadas custavam em torno de 100 dólares.

O *hardware* foi montado em cerca de dois dias, após isso começou uma longa pesquisa para aperfeiçoar o *software*. Após o protótipo estar pronto foi decidido fazer um lote para ser vendido para escolas, que iriam testar a placa Arduino. Assim, foi feito uma pequena mudança no *layout* da placa, para possibilitar a produção em quantidade, e foram produzidas 200 unidades da primeira placa Arduino. Essas placas foram vendidas pelo preço de produção e mais um euro de lucro sobre cada placa, o que levou a nova placa a ser muito competitiva no mercado, apesar de ainda não estar sendo vendida abertamente ao público. Após as primeiras placas serem vendidas, veio o interesse de uma empresa online em vender os produtos, a sparkfun.com, que comprou 20 placas, estas foram rapidamente vendidas e hoje já se estima que foram vendidas mais de 150 mil placas Arduino.

Uma das principais características do projeto Arduino é que ele é inteiramente *open source*, ou seja, é um modelo de desenvolvimento onde qualquer um pode consultar, examinar ou modificar o produto. Isso permitiu que as pessoas montassem seus projetos sem precisar licenciar o produto final, além do que projetos eletrônicos que antes eram complicados se tornaram fáceis com a ajuda do Arduino. Porém, isso acabou também levando a criação de inúmeros “clones” do Arduino pelo mundo, fato esse que demonstrou o grande legado que o projeto e seus criadores deixaram, modificando a forma como os microcontroladores eram vistos.

Partindo da primeira placa Arduino observa-se, na Figura 1, a placa Arduino serial, que era vendida com componentes discretos já montada na placa, ou desmontada na forma de kits, mostrada na Figura 2. Essa placa usava o padrão RS232 para a comunicação com o computador e necessitava de alimentação externa.

*Figura 1- Placa Arduino serial montada*

Fonte: SOUZA, Fábio. **Placas Arduino - História até o Arduino UNO.** Disponível em: <http://www.embarcados.com.br/historia-ate-o-arduino-uno/>. Acesso em: 11 maio 2016.

*Figura 2 - Placa Arduino serial sem os componentes*

Fonte: SOUZA, Fábio. **Placas Arduino - História até o Arduino UNO.** Disponível em: <http://www.embarcados.com.br/historia-ate-o-arduino-uno/>. Acesso em: 11 maio 2016.

A primeira grande mudança na placa foi a inserção da conexão USB. Essa conexão foi feita através do chip FT232BM da FTDI. Outra novidade foi a opção de seleção da alimentação pela entrada USB ou pela fonte externa, através de um *jumper* inserido na placa. Essa versão foi a USB v2.0, mostrada na Figura 3, isso porque na versão inicial a pinagem USB foi montada de maneira errada, explicando o fato da versão 2.0.

*Figura 3- Placa Arduino USB v2.0*


Fonte: SOUZA, Fábio. **Placas Arduino - História até o Arduino UNO.** Disponível em: <http://www.embarcados.com.br/historia-ate-o-arduino-uno/>. Acesso em: 11 maio 2016.

A próxima novidade veio com a placa Arduino Extreme (Figura 4), que trouxe a maioria dos componentes em montagem SMD, e lançou os famosos conectores *headers* fêmea que, posteriormente, ficou conhecido popularmente como conector "padrão Arduino".

*Figura 4- Placa Arduino Extreme*


Fonte: SOUZA, Fábio. **Placas Arduino - História até o Arduino UNO.** Disponível em: <http://www.embarcados.com.br/historia-ate-o-arduino-uno/>. Acesso em: 11 maio 2016.

Seguindo essa linha de evolução, surge a placa Arduino NG (Nuova Generazione) mostrada na Figura 5, que trouxe a mudança do microcontrolador utilizado. Até esta placa, todas as anteriores utilizavam o ATmega8, enquanto a NG surgiu com o ATmega168 que dobrou a capacidade de memória, que era agora de 16KB. Outra mudança, menos significativa, foi a troca do conversor USB-Serial, pelo modelo FT232RL, também da FTDI, porém, este necessitava de menor quantidade de componentes externos.

*Figura 5 - Placa Arduino NG*


Fonte: SOUZA, Fábio. **Placas Arduino - História até o Arduino UNO.** Disponível em: <http://www.embarcados.com.br/historia-ate-o-arduino-uno/>. Acesso em: 11 maio 2016.

Por fim, chega a placa Arduino UNO, que hoje é dita como a placa Arduino mais conhecida e utilizada no mundo. Esta veio com modificações de *hardware*, como a substituição do conversor USB-serial por um microcontrolador, o ATmega8U2. Além disso, a placa foi redesenhada para melhor descrição das entradas e saídas e melhor identificação dos pinos. A placa Arduino UNO passou até hoje por 3 revisões, sendo que hoje é chamada de Arduino UNO rev3, conforme Figura 6.

*Figura 6 - Placa Arduino UNO rev3*


Fonte: SOUZA, Fábio. **Placas Arduino - História até o Arduino UNO.** Disponível em: <http://www.embarcados.com.br/historia-ate-o-arduino-uno/>. Acesso em: 11 maio 2016.

## 4 PLACA ARDUINO MEGA 2560

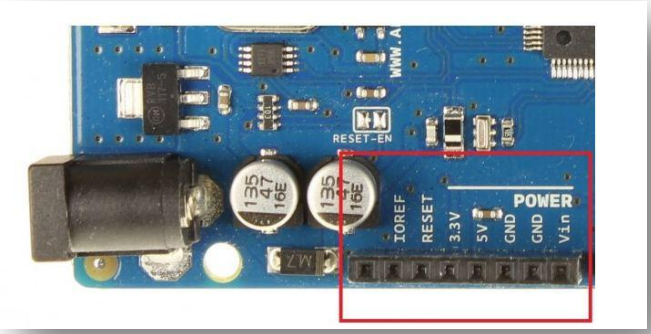
A placa utilizada nesse projeto é a placa Arduino Mega 2560, apresentada na Figura 7. É uma placa produzida com base na Arduino UNO, porém com maior capacidade. Foi planejada para suportar projetos maiores, e baseada no microcontrolador ATmega2560. Tem 54 pinos de entradas e saídas digitais, sendo que 15 destes podem ser usados como saídas PWM. Além disso, tem 16 entradas analógicas e 4 portas de comunicação serial. A seguir serão apresentados mais detalhes sobre os recursos da Arduino Mega 2560.

*Figura 7 - Placa Arduino Mega 2560*


Fonte: SOUZA, Fábio. **Arduino Mega 2560.** Disponível em: <http://www.embarcados.com.br/arduino-mega-2560/>. Acesso em: 11 maio 2016.

A alimentação da placa é feita através do cabo USB ou através de uma fonte externa por meio do *conector jack*, onde, nesse caso, o valor da fonte externa pode variar de 6V a 20V em corrente contínua. Porém, abaixo de 7V ocorrem oscilações nas saídas em 5V e, acima de 12V, o regulador de tensão fica sobrecarregado e pode danificar. Sendo assim, o ideal é utilizar valores de tensão entre 7V e 12V. Na placa ainda encontra-se um regulador de tensão de 3,3V, o U2-LP2985. Este componente oferece essa tensão continua com a finalidade de alimentar *shields* e módulos conectados à placa. Porém, o limite de corrente nessa porta é de 50 mA, e esse é um valor que se deve atentar durante os projetos. Na Figura 8, vemos o detalhe do conector de *shields* e das tensões mencionadas.

*Figura 8 - Detalhe dos conectores de alimentação para shields e módulos*

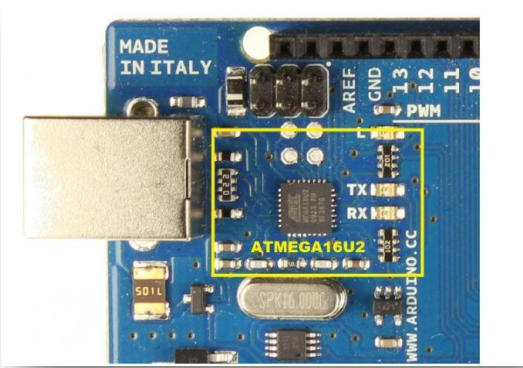



Fonte: Fonte: SOUZA, Fábio. **Arduino Mega 2560.** Disponível em: <http://www.embarcados.com.br/arduino-mega-2560/>. Acesso em: 11 maio 2016.

Além do pino 3,3V citado anteriormente, temos ainda o pino IOREF que fornece uma tensão de referência, para que *shields* que são alimentadas com 3,3V possam se adaptar para serem utilizados em 5V e vice-versa. O pino RESET, que é conectado ao reset do microcontrolador possibilitando um reset externo da placa. O pino 5V, para *shields* ou módulos alimentados com esta tensão, dois pinos GND que servem como referências ou terra. E, por último, o pino VIN que serve para alimentar a placa por uma *shield* ou uma bateria externa, ou ainda se for utilizado o pino *jack* para a alimentação, a tensão da fonte externa estará nesse pino.

A comunicação com o computador da placa é feita através de interface USB. O responsável por isso é o microcontrolador ATmega 16U2 (Figura 9) que faz a conversão USB-serial. Nesse microcontrolador são conectados os *leds* TX e RX, que controlados pelo *software* do microcontrolador indicam quando há envio e recepção de dados da placa para o computador. O ATmega16U2 está ligado ao ATMEL ATmega2560, que é o microcontrolador principal da placa. Através do pino 13, do primeiro, ligado ao reset do segundo, permite que a placa entre no modo *bootloader,* automaticamente, quando o usuário executa um *upload* na placa, sem a necessidade de fazer um *reset* manual antes de gravar na placa.

*Figura 9 - Detalhe do microcontrolador ATmega16U2 e dos leds TX e RX*




Fonte: Fonte: SOUZA, Fábio. **Arduino Mega 2560.** Disponível em: <http://www.embarcados.com.br/arduino-mega-2560/>. Acesso em: 11 maio 2016.

Como já mencionado, o microcontrolador principal da Arduino Mega 2560 é o ATMEL ATmega 2560, mostrado separadamente na Figura 10. Esse é um microcontrolador de 8 bits de arquitetura RISC avançada, opera em 16MHz, conta com 256kB de memória *flash*, sendo que 8kB são utilizados para o *bootloader*, ainda tem 8kB de memória RAM e 4kB de memória EEPROM.

*Figura 10 - Microcontrolador ATMEL ATmega2560*

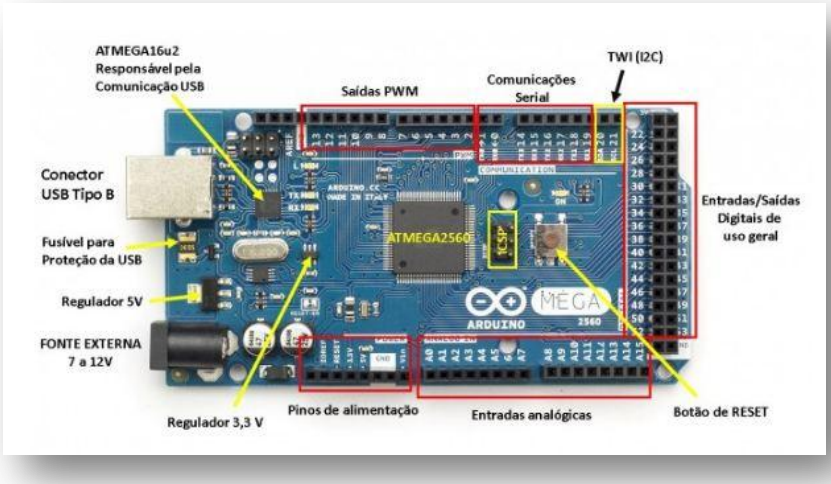

Fonte: Fonte: SOUZA, Fábio. **Arduino Mega 2560.** Disponível em: <http://www.embarcados.com.br/arduino-mega-2560/>. Acesso em: 11 maio 2016.

Os 54 pinos de entradas e saídas da Arduino Mega 2560 podem ser configurados conforme a necessidade do projeto. Esses pinos operam com tensão de 5V podendo fornecer até 40mA. Todos os pinos possuem um resistor de *pull-up* interno que podem ou não ser habilitados através do *software*. Existem ainda alguns pinos com funções especiais como mostrado a seguir:

- Comunicação Serial: Serial 1 nos pinos 18 e 19; Serial 2 nos pinos 16 e 17; e, Serial nos 3 pinos 14 e 15. Todos seguindo a ordem TX e RX. Existem ainda os pinos 0 e 1 que são a comunicação serial entre o conversor USB-serial e o ATmega2560, como já foi citado.
- Interrupções Externas: Pinos 2, 3, 18, 19, 20 e 21. Esses pinos podem ser configurados como interrupções externas com disparo na subida ou descida ou em níveis lógico alto ou baixo conforme a necessidade do projeto.
- PWM: Sinal PWM com 8 bits de resolução, pinos 2 a 13 e 44 a 46.

Além destes, ainda existem 16 entradas analógicas, pinos A0 a A15, que possuem resolução de 10 *bits*, ou seja, conversões de 0 a 1023 com tensão de referência de 5V. Na Figura 11 podemos visualizar os principais componentes e a estrutura da placa.

*Figura 11 - Identificação dos principais componentes da ATmega 2560*



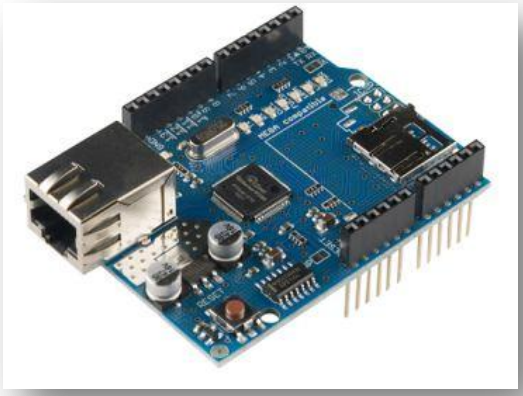

Fonte: Fonte: SOUZA, Fábio. **Arduino Mega 2560.** Disponível em: <http://www.embarcados.com.br/arduino-mega-2560/>. Acesso em: 11 maio 2016.

## 5 ETHERNET SHIELD

O fato de o projeto Arduino ser *open source*, possibilitou que inúmeras *shields* fossem desenvolvidas para as placas Arduino. Assim sendo, existem diversas *shields* no mercado, e elas permitem que o desenvolvedor estenda as funcionalidades da placa Arduino sem a necessidade de elaborar novos circuitos eletrônicos.

A Ethernet Shield W5100, vista na Figura 12, tem a função de colocar a placa Arduino *online* de uma maneira fácil e rápida. Ela baseia-se no *chip* WIZnet ethernet W5100 que fornece acesso a rede nos protocolos TCP ou UDP e é utilizada através das bibliotecas Arduino Ethernet Library e SD Library. Através dessa *shield* podemos conectar o Arduino à rede e controlá-lo através de um computador, celular ou *tablet* de qualquer parte. A *shield* ainda suporta até quatro conexões simultâneas e possui um conector RJ45 padrão.

*Figura 12 - Ehternet Shield W5100*



Fonte: THOMSEN, Adilson. **COMO COMUNICAR COM O ARDUINO ETHERNET SHIELD W5100.** Disponível em: <http://blog.filipeflop.com/arduino/tutorial-ethernet-shield-w5100.html>. Acesso em: 11 maio 2016.

# 6 MÓDULO REAL TIME CLOCK – RTC DS3231

O *Real Time Clock* (RTC) DS3231, em português, relógio de tempo real, é um relógio de alta precisão e baixo consumo de energia. Em sua placa vem anexado um sensor de temperatura e um cristal oscilador para melhorar sua exatidão.

O módulo, mostrado na Figura 13, é capaz de fornecer informações como segundo, minuto, dia da semana, dia do mês, mês e ano. Correções, como meses com menos de 31 dias e anos bissextos, são corrigidos automaticamente. Ele pode operar tanto no formato 12 horas, como 24 horas.

*Figura 13 - Módulo RTC DS3231*

Fonte: **REAL TIME CLOCK RTC DS3231.** Disponível em: <http://www.filipeflop.com/pd-1c7dbf-real-time-clock-rtc-ds3231.html> Acesso em: 31 outubro 2016.

Em caso de falha de energia, o DS3231, automaticamente aciona a bateria que acompanha o módulo, para evitar perda de dados. A tensão de operação do módulo, pode variar de 3,3 a 5V. O sensor de temperatura interno opera com 3°C de exatidão, na faixa de temperatura de 0 a 40°C.

# 7 SISTEMA SUPERVISÓRIO (SCADA)

Os sistemas supervisórios tem a finalidade de apresentar ao usuário uma interface de comunicação de alto nível com um processo que está sendo executado. Esses sistemas também são conhecidos como SCADA, e se refere a um sistema de supervisão e aquisição de dados. De maneira simples, o usuário pode através de uma interface montada no supervisório, monitorar em tempo real o processo e ainda fazer o controle do mesmo à distância.

Os primeiros sistemas SCADA eram basicamente telemétricos, e montados em painéis, conforme exemplo da Figura 14. Eles permitiam informar periodicamente o estado do processo. Essa informação era feita através de lâmpadas e indicadores no painel, sem necessariamente uma interface de aplicação com o usuário, que permitisse um fácil entendimento do processo sem profundos conhecimentos. Atualmente, os sistemas utilizam tecnologias computacionais e de comunicação para efetuar a coleta e transmissão de dados, e ainda para apresentar de forma amigável e com recursos gráficos mais elaborados, todo o processo ao usuário, conforme pode ser visto na Figura 15.

*Figura 14 - Exemplo dos primeiros painéis supervisórios*

Fonte: FISCHER, Guilherme da Silva. **Supervisório Elipse E3.** Joinville: Udesc, 2012.

Os supervisórios podem ser usados para processos industriais, como manufatura, geração de energia e muitos outros. Processos de infraestrutura

incluindo distribuição de água, tratamento de esgoto e sistemas de comunicação. E, ainda, em processos automatizados, em locais públicos ou privados, e esses abrangem o monitoramento e controle de iluminação, temperatura, ventilação e demais condições do ambiente.

*Figura 15 - Interface do supervisório de uma estação de mineração*


Fonte: FISCHER, Guilherme da Silva. **Supervisório Elipse E3.** Joinville: Udesc, 2012.

Algumas das vantagens dos sistemas supervisórios, em comparação com os painéis convencionais incluem: a redução de gastos com a montagem de painéis, já que o sistema SCADA é inteiramente virtual; a redução do espaço físico necessário, sendo que o SCADA pode ser acessado por um computador, *tablet* ou *smartphone*; a facilidade na operação, de modo que o sistema é apresentado ao usuário numa plataforma simples e intuitiva e pode ser controlado com alguns cliques na tela.

Para realizar a comunicação entre o processo e o usuário o supervisório precisa estar conectado com um dispositivo de controle, que pode ser um CLP, um microcontrolador ou outro dispositivo compatível com o SCADA. Após estarem conectados o SCADA identifica as variáveis numéricas envolvidas na aplicação, que são chamadas de *tags*, e através da necessidade do desenvolvedor essas *tags* podem representar pontos de entrada ou saída do processo, executar funções computacionais ou apenas representação gráfica na interface. Essas *tags* são ainda utilizadas para gerar alarmes, caso o valor da *tag* ultrapasse uma faixa ou condição pré-estabelecida pelo desenvolvedor, sendo possível programar o registro dos

alarmes em um banco de dados, ativar sons ou mudanças de cores na interface, ou, ainda, enviar mensagens por *email* ou celular do usuário.

Nesse projeto, o sistema SCADA a ser utilizado é o Elipse E3, que é um *software* que já possui grande aplicação na indústria.

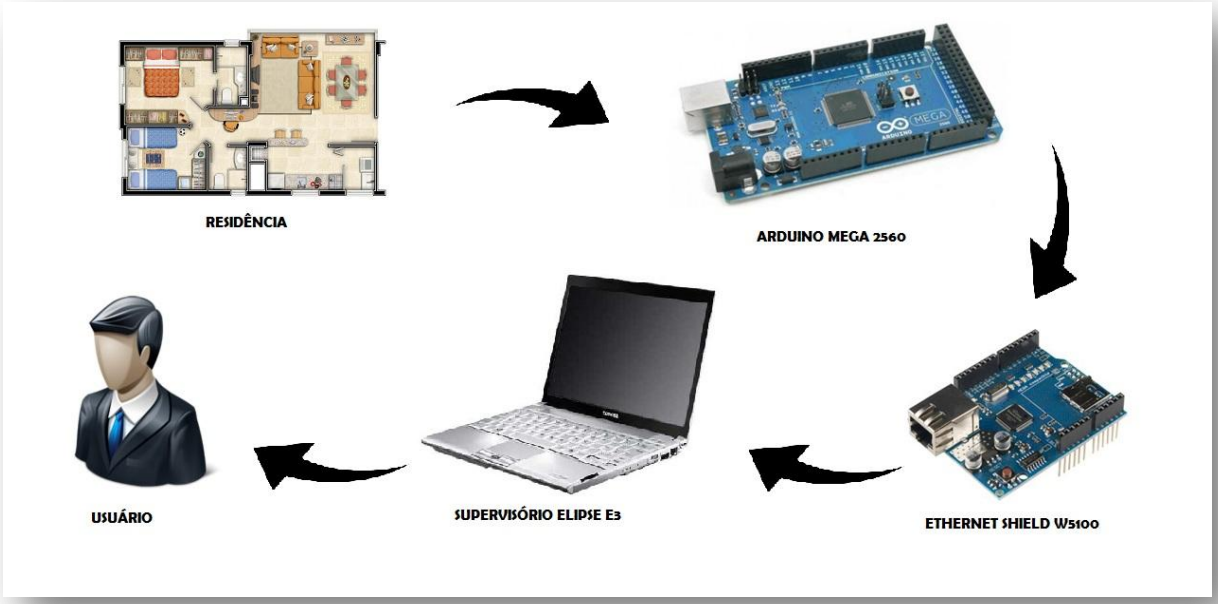
> Líder no mercado brasileiro, o Elipse E3 é uma consagrada plataforma para monitoramento e controle, oferecendo escalabilidade e constante evolução para diversos tipos de aplicações, desde simples interfaces HMI até complexos centros de operação em tempo real. Desenvolvido para atender aos requisitos atuais e futuros de conectividade, flexibilidade e confiabilidade, o E3 é o sistema SCADA ideal para o seu projeto, não importa o tamanho da sua necessidade. (ELIPSE, 2016)

Alguns dos benefícios do Elipse E3 incluem: a conexão com a maioria dos equipamentos de controle do mercado; a redução no tempo de desenvolvimento de projetos devido a padronização de aplicações por meio de bibliotecas; a permissão para editar e executar diversas bases de dados, simultaneamente; o fácil gerenciamento da aplicação, entre outros. Durante o desenvolvimento do projeto maiores informações sobre o *software* serão introduzidas ao leitor.

## 8 FUNCIONAMENTO DO ALARME

O sistema de alarme residencial é composto por sensores em todas as portas e janelas da casa, estes sensores são entradas do microcontrolador e informam se as portas e janelas encontram-se abertas ou fechadas. Todas as lâmpadas da casa são saídas do Arduino, ligadas em paralelo com a tecla física, ou seja, o usuário poderá acender as lâmpadas tanto fisicamente quanto via *software*. O sistema conta, ainda, com uma sirene que soa em caso de invasão da residência, dois botões de seleção do modo de alarme e outro para o desarme após o disparo. Na figura 16 é possível verificar como foi montado o projeto, desde a aquisição dos dados da residência até a chegada dos mesmos ao usuário.

*Figura 16 - Esquemático do sistema de alarme*




Fonte: Próprio autor.

O usuário tem três opções de seleção do modo de alarme:

**Alarme desligado:** Nesse modo as aberturas de portas e janelas não disparam o alarme, o usuário apenas poderá monitorar através da tela do supervisório quais as janelas e portas estão abertas e quais as lâmpadas estão acesas. Esse modo é utilizado durante o dia e quando houverem pessoas na casa.

**Alarme comum:** Esse modo só é acionado quando todas as portas e janelas estiverem fechadas, caso contrário um aviso aparece na tela do supervisório indicando que alguma porta ou janela precisa ser fechada. Após o acionamento, se alguma janela ou porta for violada, a sirene irá soar e todas as lâmpadas da casa irão acender. O alarme ficará nesse estado até que o botão de desarme seja acionado, lembrando que esse botão deverá ficar em um local estratégico da casa e não visível.

**Alarme viagem:** No modo de viagem o disparo do alarme funciona da mesma maneira que o modo anterior, porém, nesse caso, existirá um controle automático da iluminação da casa. Durante o período das 18h até às 23h as lâmpadas externas da casa irão permanecer acesas, e as lâmpadas internas da casa irão acender e apagar de maneira alternada e aleatória a cada 15 minutos, isso irá simular a presença de pessoas na casa, inibindo a ação de pessoas mau intencionadas.

Na tela do supervisório Elipse E3 irá constar a planta baixa da casa com as informações de portas, janelas e lâmpadas. Informações essas, passadas ao usuário de forma gráfica. Ainda, irá constar o modo do alarme atual, e os avisos pertinentes ao usuário.

Uma das vantagens desse sistema é que ele pode ser adaptado para qualquer residência, e também sofrer modificações de acordo com alguma preferência especifica do usuário. Ele pode ainda ser utilizado em prédios públicos, onde seria difícil o monitoramento de lâmpadas e janelas, devido ao alto número.

## 9 MONTAGEM DA INTERFACE COM O USUÁRIO

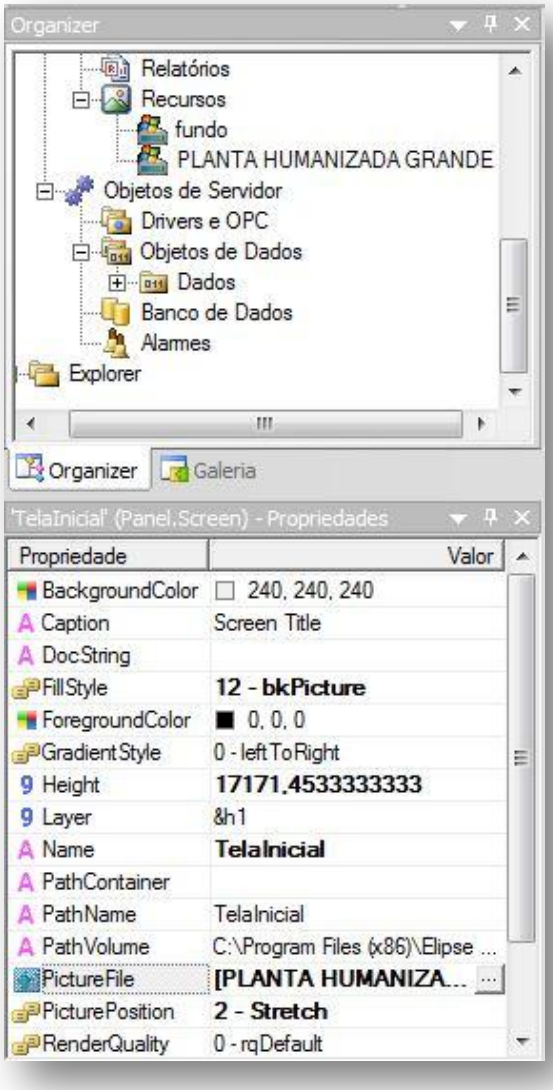
Através do sistema SCADA Elipse E3, foi feita a montagem da interface entre o alarme e o usuário. A residência utilizada nesse projeto possui três dormitórios, dois banheiros, cozinha e sala de estar. Na Figura 17 se pode ver a distribuição dos cômodos e sua legenda. Essas legendas serão as mesmas utilizadas na programação do Arduino e também do supervisório.

*Figura 17 - Planta baixa da residência*


Fonte: Próprio autor.

Para inserir a planta humanizada da residência numa tela do supervisório, foi necessário criar um recurso no projeto. Nesse recurso, inserimos a imagem da planta da residência, assim será possível utilizar a imagem em qualquer tela desejada ao longo do projeto. Após o recurso criado, foram modificados três parâmetros da tela inicial. Acessando as propriedades, mudamos o parâmetro *FillStyle* para 12-bkPicture e no parâmetro *PictureFile* foi inserido o recurso criado com a planta humanizada e, por fim, para estender a imagem por toda a tela, foi alterado o parâmetro *PicturePosition* para 2-Stretch. Assim, o fundo da tela inicial será a imagem da planta humanizada, como será visto mais a diante. Na Figura 18 pode-se observar os parâmetros citados até agora.

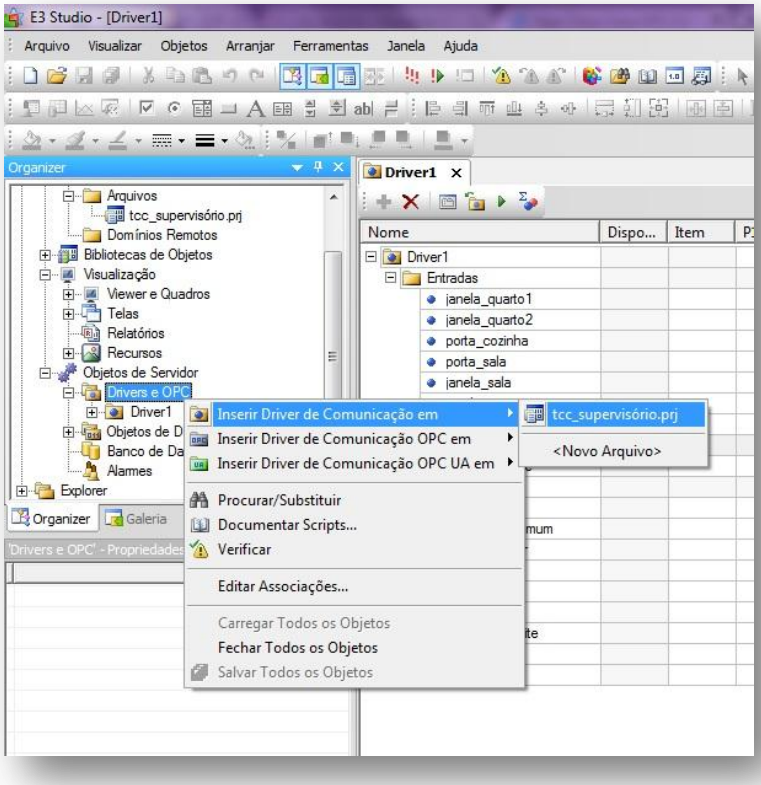
*Figura 18- Detalhe de parâmetros e recursos*




Fonte: Próprio autor.

Inicialmente, se inseriu o *drive* de comunicação na planta do supervisório. No *driver,* são inseridas as *tags* correspondentes às entradas e saídas do Arduino. Para inserir um *drive*, no *organizer*, se clica com o botão direito do *mouse* em ***Drivers e OPC***, em seguida, **Inserir Driver de Comunicação em**, e inserimos no nosso domínio o *driver* de comunicação Modbus, disponível no site da Elipse. Os passos para inserir o *driver,* estão mostrados na Figura 19.

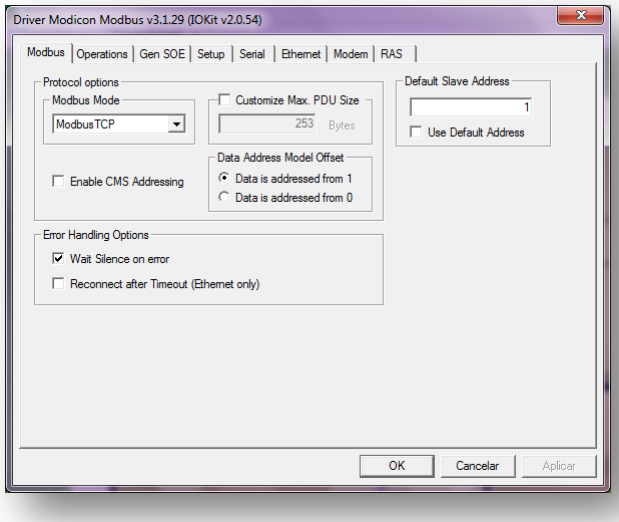
*Figura 19 - Inserindo um drive de comunicação*




Fonte: Próprio autor.

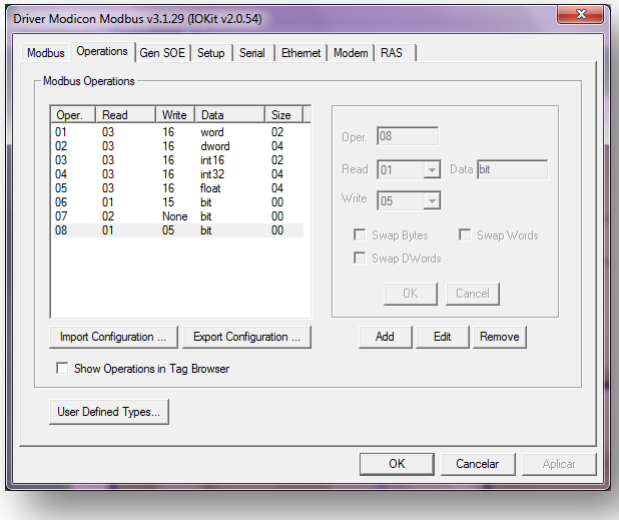
Quando o *driver* é inserido, são abertas telas de configuração. Nas figuras 20, 21, 22 e 23, mostradas a seguir, são apresentados os parâmetros usados na configuração do *driver.*

*Figura 20 - Configuração do driver aba Modbus*



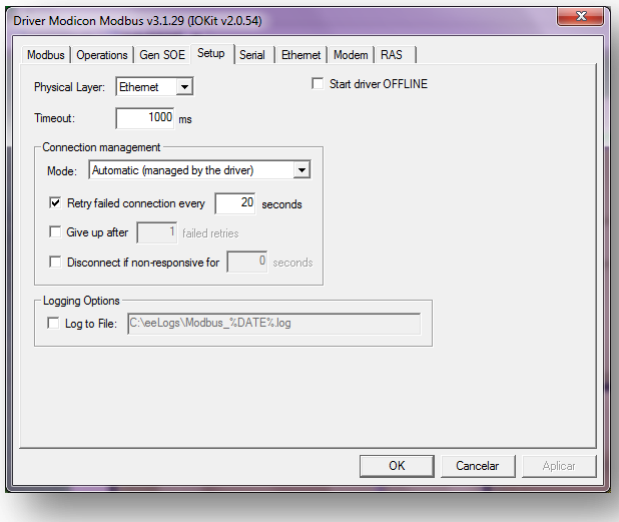

Fonte: Próprio autor

*Figura 21 - Configuração do driver aba Operations*



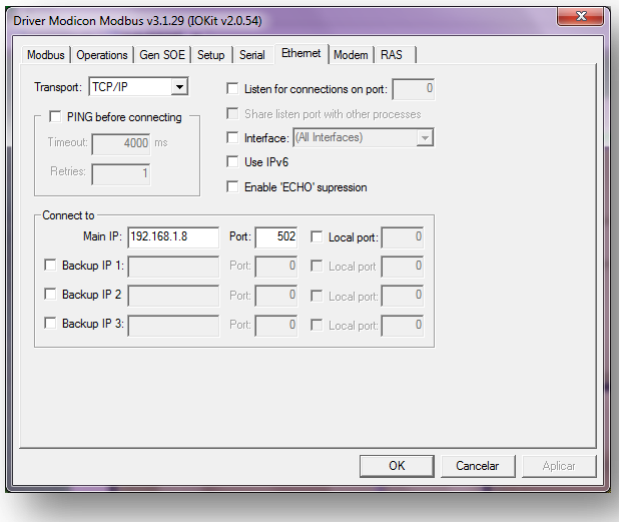

Fonte: Próprio autor

*Figura 22 - Configuração do driver aba Setup*




Fonte: Próprio autor

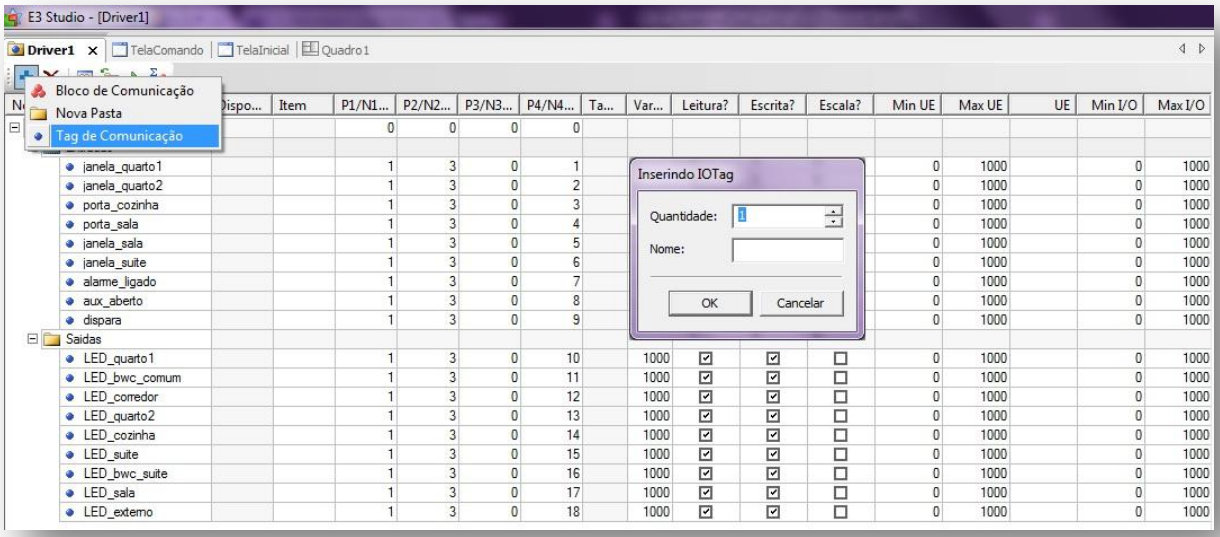
*Figura 23 - Configuração do driver aba Ethernet*




Fonte: Próprio autor

Após inserir e configurar o driver, foram criadas as *tags*, de entrada e de saída, com os mesmos nomes, que serão posteriormente declaradas no Arduino. Não é necessário declarar as variáveis e *tags,* com o mesmo nome, porém isso facilita para não haver confusão. Para inserir uma *tag* de comunicação, abrimos o *driver* inserido, clicando no **símbolo mais**, e em seguida em **Tag de Comunicação,** na janela que irá abrir, digitamos o número de *tags* e o respectivo nome. Esse procedimento fica esclarecido na Figura 24. Ainda nessa figura, podemos ver a configuração das *tags,* onde os pinos P2/N2 e P4/N4 são modificados. Para todas as *tags,* o pino P2/N2 recebe o valor 3, que indica de a *tag* será de leitura. Já o pino P4/N4, será o valor da memória onde o Arduino irá mandar o valor lido, nesse caso, foi numerado a partir do número 1 até o número 18.

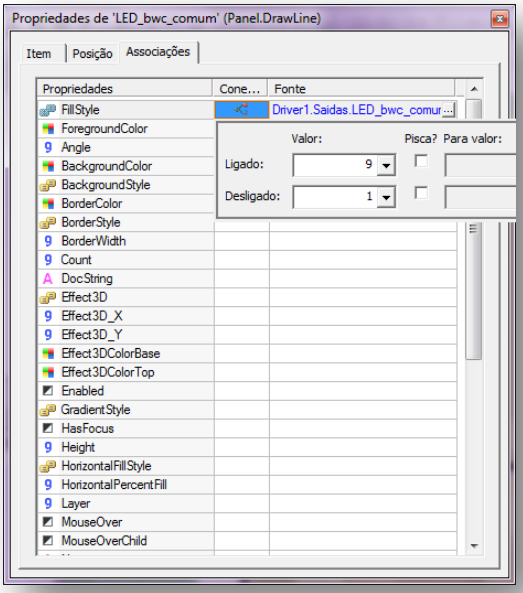
*Figura 24 - Detalhe do driver de comunicação*




Fonte: Próprio autor.

A primeira tela do supervisório, chamada de TelaInicial, é a principal, que contém a planta da casa, e o status da iluminação e das aberturas. Para o estado da iluminação utilizou-se formas no mesmo desenho do cômodo. Sendo que quando a lâmpada estiver acesa, a forma fica amarela transparente, e quando a lâmpada for apagada a forma fica totalmente transparente. Já nas aberturas, foi desenhada uma barra verde na posição das mesmas. Quando uma porta ou janela estiver fechada a barra fica totalmente verde, e quando estiver aberta a barra pisca em vermelho e branco. Para efetuar tal mudança de estado, as variáveis, criadas anteriormente, foram associadas às formas desenhadas no supervisório. No caso da iluminação foram criadas associações em dois parâmetros do objeto, primeiro na propriedade *FillStyle* foi associada o valor da *tag* correspondente, sendo que quando o valor da *tag* for *true*, o parâmetro é o número 9 – *SemiTransparent* e quando for *false* o parâmetro é o número 1 – *Hollow*. No primeiro caso, a figura irá ficar com um aspecto transparente e, no segundo, irá ficar apenas com o contorno do objeto. Esses detalhes podem ser vistos na Figura 25.

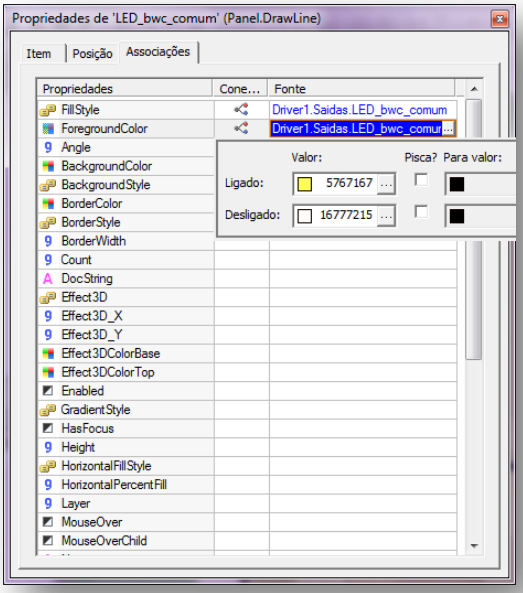
*Figura 25 - Detalhe das propriedades do objeto*




Fonte: Próprio autor.

Além disso, foi feita uma associação no parâmetro *ForegroudColor*. Novamente, foi associado o valor da *tag* correspondente, mas, dessa vez, em caso de um valor *true* o parâmetro fica na cor amarela e, em caso de *false,* fica na cor branca, como pode ser visto na Figura 26. Dessa maneira, se consegue o efeito citado anteriormente, luz acesa, cômodo amarelo transparente, luz apagada, cômodo totalmente transparente.

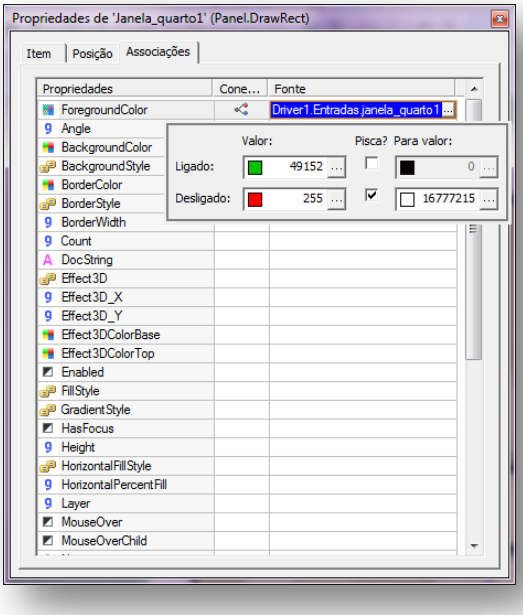
*Figura 26 - Detalhe das propriedades do objeto*




Fonte: Próprio autor.

No caso dos objetos das aberturas, apenas uma associação foi necessária. No parâmetro *ForegroundColor* foi associada a *tag* correspondente, sendo que quando a *tag* for *true* o parâmetro fica na cor verde e, quando for *false,* fica piscando na cor vermelha e branca. O detalhe dessa associação pode ser visto na Figura 27.

*Figura 27 - Detalhe da associação do objeto*




Fonte: Próprio autor.

Além de todos os itens já citados, essa tela ainda possui um display, que informa a situação atual da residência, casa segura ou casa invadida.

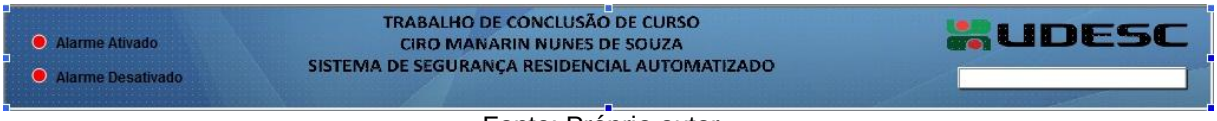
Todas as associações criadas até aqui foram do tipo digital, já que as *tags* são do tipo digital e retornam um valor *false* ou *true.* Na Figura 28 se pode visualizar a tela principal do supervisório, onde se encontram todos os objetos detalhados até o momento.

*Figura 28 - Tela principal do supervisório*


Fonte: Próprio autor.

Além da tela principal, foi criada uma segunda tela, que será apresentada acima da tela principal e apresentará algumas informações ao usuário. Essa tela foi chamada de TelaComando (Figura 29). Nela o usuário pode verificar o estado atual do alarme no canto esquerdo, sendo que a cor verde indica o estado atual. A tela possui no canto direito um *display*, onde podem ser mostradas informações ao usuário.

*Figura 29 - Tela de comando do supervisório*




Fonte: Próprio autor.

Para verificação do funcionamento, foi feita uma simulação de uma situação do dia-a-dia. Mudando os valores das *tags*, se pode verificar o funcionamento do supervisório. Nessa situação, na Figura 30, se verifica que o quarto 1, a cozinha e o BWC suíte estão com as luzes acesas, enquanto o resto dos cômodos estão com as luzes apagadas. E, ainda, a janela do quarto 1 e a porta da cozinha estão abertas. No caso das aberturas não se pode ver o efeito do pisca, apenas a cor branca. As demais situações, em estado normal, se apresentam em cor verde.

*Figura 30 - Sistema supervisório*


Fonte: Próprio autor.

## 10 PROGRAMAÇÃO ARDUINO

O início da programação se deu a partir da definição das entradas e saídas do projeto, e também quais os pinos da placa Arduino Mega 2560 iriam ser utilizados para cada uma das entradas e saídas. Para isso, foi utilizado o comando ***#define***, o qual é complementado com o nome da *tag* que será relacionada ao pino e do número do pino. Também foram declaradas algumas variáveis do tipo inteira, usando o comando ***int*** seguido do nome da variável e do seu valor inicial. Nas linhas de comandos da Figura 31, se vê como ficaram as definições do projeto.

*Figura 31: Definição dos pinos de entrada e saída*

```
//Definindo os pinos das entradas
#define janela_quarto1 22
#define janela_quarto2 24
#define porta_cozinha 26
#define porta_sala 28
#define janela_sala 30
#define janela_suite 32
#define alarme_ligado 34
#define alarme_viajem 36
#define desliga_alarme 38
//Definindo os pinos das saídas
#define LED_quarto1 33
#define LED_bwc_comum 35
#define LED_corredor 37
#define LED_quarto2 39
#define LED_cozinha 41
#define LED_suite 43
#define LED_bwc_suite 45
#define LED_sala 47
#define LED_externo 49
#define ilum_auto 40
#define buzzer 44
//Declaração de variáveis auxiliares
int aux_alarme_ligado = 0;
int aux_alarme_viajem = 0;
int dispara = 0;
int aux_aberto =0;
int aux_dispara;
unsigned long inicio =0;
byte contando = false;
```

Fonte: Próprio Autor

Além das definições, num momento inicial foi introduzida as bibliotecas para o uso da interrupção de tempo, da *shield ethernet* e da comunicação via *Modbus.* Esses comandos podem ser vistos na Figura 32.

*Figura 32 – Inclusão das bibliotecas*

```
#include <TimerOne.h> //Biblioteca da Interrupção de Tempo
#include <SPI.h>      //Bibliotecas da shield Ethernet
#include <Ethernet.h>
#include "Mudbus.h" //Bibliotecas da comunicação Modbus
#include "M0dbus.h"
```

Fonte: Próprio Autor

Após a inclusão das bibliotecas, foi criada uma função chamada **atualiza**, essa, tem como objetivo enviar os valores das variáveis utilizadas para o supervisório, por meio da comunicação *Modbus.* Os valores das memórias correspondentes, estão relacionados ao valor do pino P4/N4, visto no item 9. Porém, nos registradores o valor do pino é igual a n-1, onde n é o valor do pino. Assim, temos registradores variando de 0 a 17. A função **atualiza** pode ser vista na Figura 33.

*Figura 33 – Função atualiza*

```
//Função para atualizar as memórias da comunicação
//Essa função será chamada na interrupção de tempo
void atualiza()
{

  Mb.R[0] = digitalRead(janela_quarto1);
  Mb.R[1] = digitalRead(janela_quarto2);
  Mb.R[2] = digitalRead(porta_cozinha);
  Mb.R[3] = digitalRead(porta_sala);
  Mb.R[4] = digitalRead(janela_sala);
  Mb.R[5] = digitalRead(janela_suite);
  Mb.R[6] = (aux_alarme_ligado + aux_alarme_viajem);
  Mb.R[7] = aux_aberto;
  Mb.R[8] = aux_dispara;

  Mb.R[9] = digitalRead(LED_quarto1);
  Mb.R[10] = digitalRead(LED_bwc_comum);
  Mb.R[11] = digitalRead(LED_corredor);
  Mb.R[12] = digitalRead(LED_quarto2);
  Mb.R[13] = digitalRead(LED_cozinha);
  Mb.R[14] = digitalRead(LED_suite);
  Mb.R[15] = digitalRead(LED_bwc_suite);
  Mb.R[16] = digitalRead(LED_sala);
  Mb.R[17] = digitalRead(LED_externo);
}
```

Fonte: Próprio autor

Dentro da função ***void setup***, do programa, ficam as configurações que serão feitas apenas uma vez quando o programa for rodado. Nessa função foi iniciada a função de interrupção de tempo, onde se definiu que a função seria chamada a cada 700 milisegundos, e também qual a função seria chamada a cada interrupção, nesse caso a função **atualiza**. Além disso, foram configurados os pinos de entrada e saída que foram definidos anteriormente. Para isso, se utilizou a função ***pinMode***, que é utilizada informando o número, ou nome, caso tenha sido definido, do pino seguido da informação de entrada (*INPUT),* saída (*OUTPUT)* ou entrada com *pullup* interno ativo (*INPUT_PULLUP).*

Essas configurações são apresentadas na Figura 34, onde vemos que foi optado pelo *pullup* interno ativado para todas as entradas, pois os sensores e botões enviam sinal 0V quando acionados e, quando não acionados, o *pullup* irá garantir o nível lógico alto nas entradas, evitando o estado de alta impedância que pode causar erros na execução do programa.

*Figura 34 - Configurações iniciais*

```
Timer1.initialize(700);
Timer1.attachInterrupt(atualiza);
delay(5000);
Serial.begin(115200);
//Declarando as entradas e ativando PULLUP
  pinMode(janela_quarto1, INPUT_PULLUP);
  pinMode(janela_quarto2, INPUT_PULLUP);
  pinMode(porta_cozinha, INPUT_PULLUP);
  pinMode(porta_sala, INPUT_PULLUP);
  pinMode(janela_sala, INPUT_PULLUP);
  pinMode(janela_suite, INPUT_PULLUP);
  pinMode(alarme_ligado, INPUT_PULLUP);
  pinMode(alarme_viajem, INPUT_PULLUP);
  pinMode(desliga_alarme, INPUT_PULLUP);
//Declarando as saídas
  pinMode(LED_quarto1, OUTPUT);
  pinMode(LED_bwc_comum, OUTPUT);
  pinMode(LED_corredor, OUTPUT);
  pinMode(LED_quarto2, OUTPUT);
  pinMode(LED_cozinha, OUTPUT);
  pinMode(LED_suite, OUTPUT);
  pinMode(LED_bwc_suite, OUTPUT);
  pinMode(LED_sala, OUTPUT);
  pinMode(LED_externo, OUTPUT);
  pinMode(ilum_auto, OUTPUT);
  pinMode(buzzer, OUTPUT);
  pinMode(53, OUTPUT);
  pinMode(10,OUTPUT);
  digitalWrite(10,LOW);
}
```

Fonte: Próprio Autor

Ainda dentro da função ***void setup***, foram feitas as configurações finais da comunicação *Modbus*. Essas configurações incluem inicialização das portas serial e definição dos valores do endereço de *ip,* como pode ser visto na Figura 35.

*Figura 35 - Configurações da comunicação Modbus*

```
uint8_t mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0x51, 0x06 };
uint8_t ip[] = { 192, 168, 1, 50 };
uint8_t gateway[] = { 192, 168, 1, 1 };
uint8_t subnet[] = { 255, 255, 255, 0 };
Ethernet.begin(mac, ip, gateway, subnet);
```

Fonte: Próprio autor

Após as definições, configurações e declarações, é dado início ao *loop* principal do programa. A primeira medida foi a inicialização da função que executa a comunicação *Modbus*, é uma linha extremamente simples de comando, mas sem a qual, a comunicação não funciona. Esse comando pode ser visto na Figura 36.

*Figura 36 – Comando que inicializa a comunicação Modbus*

```
void loop()
{

Mb.Run();
```

Fonte: Próprio Autor

A primeira ação a ser desenvolvida na programação é a verificação da situação dos sensores, buscando identificar se há algum sensor acionado. A lógica está apresentada na Figura 37.

*Figura 37 - Lógica de disparo do alarme*

```
//LÓGICA DE DISPARO DO ALARME

if (!digitalRead(janela_quarto1)|| !digitalRead(janela_quarto2)|| !digitalRead(porta_cozinha) ||
                                  !digitalRead(porta_sala)|| !digitalRead(janela_sala)|| !digitalRead(janela_suite)){
  dispara=1;
}
  else{
    dispara=0;
  }
```

Fonte: Próprio Autor

Observa-se que se tem uma estrutura do tipo ***if else***. O comando utilizado na estrutura é o ***DigitalRead***, o qual permite realizar a leitura de uma entrada digital da placa Arduino. Para tanto, basta indicar o número do pino ou o nome designado para o pino com o comando ***#define*** já mencionado.

Pode ser observado, ainda na Figura 37, que, entre cada comando de leitura, há os caracteres "!" e "||", concatenando os comandos. A função do "!" é indicar uma negação, ou seja, a condição será verdadeira se o pino estiver no nível baixo. Já a "||" tem a função de indicar a condição *"ou/or"* entre as varias leituras. Assim, se uma das leituras for verdadeira, então, será armazenado o valor 1 na variável "*dispara*". Do contrário (*else)*, será armazenado o valor 0.

Ao se utilizar a variável *"dispara"* para armazenar a condição dos sensores, sempre que se precisar verificar como estão os sensores, basta analisar o valor armazenado na referida variável.

Seguindo as linhas de código do programa, se inicia a lógica do alarme comum. Primeiramente, se verifica se o botão de acionamento do alarme comum foi acionado e, caso tenha sido, se grava o valor um numa variável auxiliar denominada "*aux_alarme_ligado"*. Após a verificação, caso a variável auxiliar e também a variável "*dispara"* estejam com valor um, todas as luzes da casa irão piscar e o *buzzer* irá soar com intermitência de 0,5 segundos, assim como as lâmpadas, isso ocorrerá enquanto o botão de "desliga alarme" não for ativado.  A Figura 38 apresenta a lógica desenvolvida.

*Figura 38 - Lógica de funcionamento do alarme comum*

```
if(digitalRead(alarme_ligado)== LOW){ //INICIA LÓGICA DO ALARME COMUM
    aux_alarme_ligado = 1;
  }
if (aux_alarme_ligado==1 && dispara==1) {
    while (digitalRead(desliga_alarme)) {
    digitalWrite(ilum_auto, LOW);
    digitalWrite(buzzer, HIGH);
    digitalWrite(LED_quarto1, HIGH);
    digitalWrite(LED_quarto2, HIGH);
    digitalWrite(LED_suite, HIGH);
    digitalWrite(LED_corredor, HIGH);
    digitalWrite(LED_bwc_comum, HIGH);
    digitalWrite(LED_sala, HIGH);
    digitalWrite(LED_bwc_suite, HIGH);
    digitalWrite(LED_externo, HIGH);
    digitalWrite(LED_cozinha, HIGH);
    delay(500);
    digitalWrite(buzzer, LOW);
    digitalWrite(LED_quarto1, LOW);
    digitalWrite(LED_quarto2, LOW);
    digitalWrite(LED_suite, LOW);
    digitalWrite(LED_corredor, LOW);
    digitalWrite(LED_bwc_comum, LOW);
    digitalWrite(LED_sala, LOW);
    digitalWrite(LED_bwc_suite, LOW);
    digitalWrite(LED_externo, LOW);
    digitalWrite(LED_cozinha, LOW);
    delay(500);
    }
   }
```

Fonte: Próprio Autor

Na Figura 38 foram introduzidos alguns comandos ainda não utilizados. O primeiro é o duplo “&” entre as condições do ***if***. *I*sso indica a estrutura “e” ou *and* entre as condições, ou seja, só será verdadeiro se as duas condições forem satisfeitas. Outra função é a ***digitalWrite***, que possibilita escrever um valor alto (*HIGH)* ou baixo *(LOW)* numa saída do Arduino, assim como a ***digitalRead***, essa função precisa ser seguida do número ou nome do pino desejado. A função ***delay***, causa um atraso na passagem para a próxima linha de comando, o valor do atraso é inserido, e deve ser indicado em milissegundos, nesse caso, 500 milissegundos correspondentes a 0,5 segundos.

Na Figura 39, ocorre o início das linhas de comando que formam a lógica de programação do modo de alarme viajem. Na primeira estrutura condicional ***if***, se verifica o acionamento do botão de seleção do modo viajem. Caso o botão tenha sido pressionado, a variável auxiliar "*aux_alarme_viajem"* recebe o valor um, semelhante ao inicio da lógica do alarme comum. Na próxima estrutura de condição ***if***, a variável auxiliar é verificada, assim como a variável "*dispara*", que estará em zero caso nenhuma porta ou janela esteja aberta. Além disso, se utilizará pela primeira vez a variável inteira "*hora*", comparando a mesma com os valores 18 e 23, ou seja, caso esteja entre 18h e 23h, o alarme irá iniciar a simulação de pessoas na casa. Para esta simulação, outra cadeia de ***if***, testa a variável inteira "*minuto*", sendo que, a cada quarto de hora, ou a cada 15 minutos, um conjunto de lâmpadas da casa é aceso.

*Figura 39 - Início da lógica de programação do alarme viajem*

```
if(digitalRead(alarme_viajem)==LOW){
  aux_alarme_viajem = 1;
  digitalWrite(ilum_auto, LOW);
}
if (aux_alarme_viajem==1 && dispara==0 && hora>=18 && hora<23) {
    digitalWrite(LED_externo, HIGH);
    if (minuto>=0 && minuto<15){
    digitalWrite(LED_corredor, LOW);
    digitalWrite(LED_bwc_suite, LOW);
    digitalWrite(LED_suite, HIGH);
    digitalWrite(LED_cozinha, HIGH);
    }
else if (minuto>=15 && minuto<30){
    digitalWrite(LED_suite, LOW);
    digitalWrite(LED_cozinha, LOW);
    digitalWrite(LED_sala, HIGH);
    digitalWrite(LED_quarto2, HIGH);
}
else if (minuto>=30 && minuto<45){
    digitalWrite(LED_sala, LOW);
    digitalWrite(LED_quarto2, LOW);
    digitalWrite(LED_quarto1, HIGH);
    digitalWrite(LED_bwc_comum, HIGH);
}
else if (minuto>=45 && minuto<60){
    digitalWrite(LED_quarto1, LOW);
    digitalWrite(LED_bwc_comum, LOW);
    digitalWrite(LED_corredor, HIGH);
    digitalWrite(LED_bwc_suite, HIGH);
}
```

Fonte: Próprio Autor

A lógica apresentada na Figura 39, não foi de fato utilizada na programação. Isso porque ocorreu um problema na comunicação quando foi feita a utilização do RTC, e também, seria inviável, numa apresentação didática, aguardar os horários para visualizar o funcionamento do sistema. Sendo assim, as linhas da Figura 39 foram substituídas pelos códigos apresentados na Figura 40 e Figura 41. Mais detalhes sobre o problema de comunicação será discutido nas considerações finais.

Na Figura 40, foi desenvolvida uma lógica utilizando a função **millis**. Essa lógica irá incrementar uma variável **count** a cada ciclo de 5000 milisegundos, ou seja, a cada 5 segundos. Isso irá facilitar a observação das mudanças na apresentação didática.

*Figura 40 - Lógica de contagem de tempo*

```
if (aux_alarme_viajem && !dispara) {
  digitalWrite(LED_externo, HIGH);
  if(!contando){
    contando = true;
    inicio = millis();
  }
  if (millis()-inicio > 5000) {
    count = count + 1;
    inicio = millis();
  }
```

Fonte: Próprio Autor

Assim, na Figura 41, foi utilizada uma estrutura **switch case** para ler a variável **count** e criar os ciclos que irão alternar a iluminação. Foram criados 4 ciclos, e após isso a variável **count** é novamente inicializada e os ciclos recomeçam.

*Figura 41- Ciclos de iluminação aleatória*

```
    switch (count){
    case 1:
    digitalWrite(LED_corredor, LOW);
    digitalWrite(LED_bwc_suite, LOW);
    digitalWrite(LED_suite, HIGH);
    digitalWrite(LED_cozinha, HIGH);
    break;
    case 2:
    digitalWrite(LED_suite, LOW);
    digitalWrite(LED_cozinha, LOW);
    digitalWrite(LED_sala, HIGH);
    digitalWrite(LED_quarto2, HIGH);
    break;
    case 3:
    digitalWrite(LED_sala, LOW);
    digitalWrite(LED_quarto2, LOW);
    digitalWrite(LED_quarto1, HIGH);
    digitalWrite(LED_bwc_comum, HIGH);
    break;
    case 4:
    digitalWrite(LED_quarto1, LOW);
    digitalWrite(LED_bwc_comum, LOW);
    digitalWrite(LED_corredor, HIGH);
    digitalWrite(LED_bwc_suite, HIGH);
    break;
    default:
    count = 1;
    contando = false;
    }
}
```

Fonte: Próprio Autor

Em seguida são adicionadas novamente as linhas de comando para o disparo do alarme. Essas linhas são idênticas ao já apresentado na Figura 38, e que são novamente mostradas na Figura 42, finalizando, assim, a lógica do alarme viajem.

*Figura 42 - Lógica de disparo do alarme viajem*

```
if (aux_alarme_viajem == 1 && dispara == 1) {
while (digitalRead(desliga_alarme)) {

digitalWrite(buzzer, HIGH);
digitalWrite(LED_quarto1, HIGH);
digitalWrite(LED_quarto2, HIGH);
digitalWrite(LED_suite, HIGH);
digitalWrite(LED_corredor, HIGH);
digitalWrite(LED_bwc_comum, HIGH);
digitalWrite(LED_sala, HIGH);
digitalWrite(LED_bwc_suite, HIGH);
digitalWrite(LED_externo, HIGH);
digitalWrite(LED_cozinha, HIGH);
delay(500);
digitalWrite(buzzer, LOW);
digitalWrite(LED_quarto1, LOW);
digitalWrite(LED_quarto2, LOW);
digitalWrite(LED_suite, LOW);
digitalWrite(LED_corredor, LOW);
digitalWrite(LED_bwc_comum, LOW);
digitalWrite(LED_sala, LOW);
digitalWrite(LED_bwc_suite, LOW);
digitalWrite(LED_externo, LOW);
digitalWrite(LED_cozinha, LOW);
delay(500);
 }
}
```

Fonte: Próprio Autor

Para finalizar o código, foi criada uma estrutura para o momento em que o botão desliga alarme for acionado. Nessa estrutura, apresentada na Figura 43, as variáveis auxiliares são zeradas, a saída *ilum_auto* é colocada em nível alto, liberando o acionamento das lâmpadas pelos interruptores, e as saídas de controle automático da iluminação são colocadas em nível baixo.

*Figura 43 - Lógica da função desliga alarme*

```
//DESATIVANDO A ILUMINAÇÃO AUTOMÁTICA QUANDO O ALARME É DESLIGADO
if (digitalRead(desliga_alarme) == LOW) {
  aux_alarme_ligado = 0;
  aux_alarme_viajem = 0;
    digitalWrite(ilum_auto, HIGH);
    digitalWrite(LED_quarto1, LOW);
    digitalWrite(LED_quarto2, LOW);
    digitalWrite(LED_suite, LOW);
    digitalWrite(LED_corredor, LOW);
    digitalWrite(LED_bwc_comum, LOW);
    digitalWrite(LED_sala, LOW);
    digitalWrite(LED_bwc_suite, LOW);
    digitalWrite(LED_externo, LOW);
    digitalWrite(LED_cozinha, LOW);
  }
}
```

Fonte: Próprio Autor

No Anexo A é apresentado a programação do Arduino na íntegra.

## 11 MONTAGEM DO PROTÓTIPO

Para a apresentação didática e visualização do funcionamento do projeto, foi montado um protótipo que possibilitasse tais ações. A base do protótipo foi feita com uma chapa de madeira MDF de 6mm de espessura e 600x295mm (LxA). Sobre a chapa de MDF foi plotado, em adesivo, a planta utilizada no projeto, e já apresentada na Figura 16. Para simular as lâmpadas da residência, foram utilizados *leds* alto brilho de cor branca e diâmetro de 5mm, conforme mostra a Figura 44. Em série com o terminal positivo de cada *led*, foi inserido um resistor de 470Ω, com o intuito de diminuir a corrente que circula no *led,* impedindo assim, que o mesmo queime.

*Figura 44 - Led alto brilho branco*

Fonte:**BURGOS ELETRÔNICA**. Disponível em: <http://loja.burgoseletronica.net/leds-diversos/led-branco-alto-brilho-5-mm-10-pecas.html> Acesso em: 31 outubro 2016.

Para os sensores de portas e janelas, foram utilizados os sensores *reed switch,* mostrados na Figura 45. Os sensores *reed switch* são formados basicamente de duas lâminas de material ferroso, envoltas por uma capsula de vidro, que é preenchida por um gás inerte. Quando aproximamos um imã do sensor, as lâminas são atraídas e fecham um curto, possibilitando a passagem de corrente entre as duas extremidades. Estes sensores podem ser do tipo NA, contatos normalmente abertos, ou NF, contatos normalmente fechados. Nesse caso foram usados sensores com contatos NA.

*Figura 45 - Sensor reed switch tipo NA*

Fonte:**Baú da Eletrônica.** Disponível em: <http://www.baudaeletronica.com.br/sensor-magnetico-reed-switch.html> Acesso em: 31 outubro 2016

Além dos *leds* e sensores, foram utilizadas chaves *on/off* para funcionarem como interruptores das lâmpadas e, também, três chaves *push-button,* que servirão para selecionar entre alarme comum, alarme viajem e desligar alarme. Ainda, foi inserido no protótipo um *buzzer* de 5V, que servirá como a sirene do alarme, soando quando o mesmo for disparado. Os dispositivos citados estão mostrados na Figura 46.

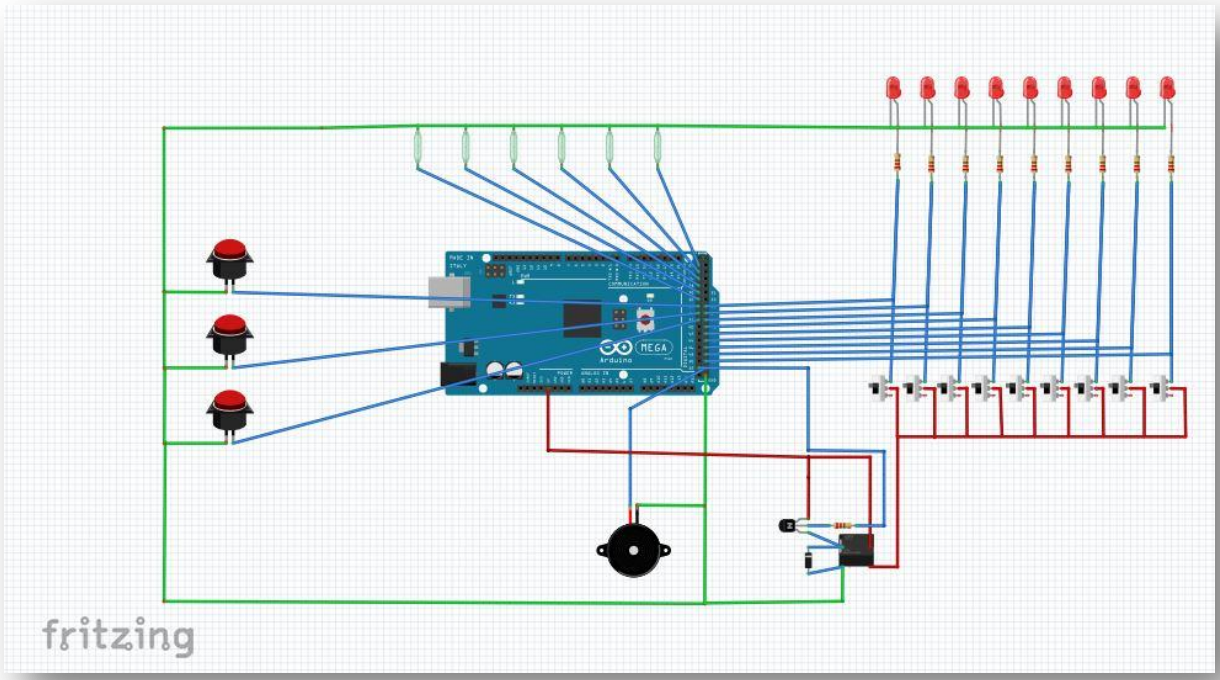
*Figura 46 - Chave push button, buzzer e chave on/off*

Fonte: Próprio Autor

Para auxiliar na montagem do protótipo e, também, no seu entendimento, foi desenvolvido um esquemático com os itens descritos acima, juntamente com a placa Arduino Mega 2560. O esquemático, que pode ser visto na Figura 47, foi feito

utilizando o *software* Fritzing, que é um *software* gratuito, e possui, além dos vários componentes eletrônicos, algumas das placas da plataforma Arduino.

*Figura 47 - Esquemático de montagem do protótipo*




Fonte: Próprio Autor

Além do esquemático acima, foi desenhado um diagrama unifilar, também com o intuito de auxiliar na montagem. Porém, não de uma forma didática, mas, sim, de maneira mais clara e objetiva. O diagrama pode ser visto no Anexo B, e foi desenhado utilizando o *software* AutoCAD, versão estudante.

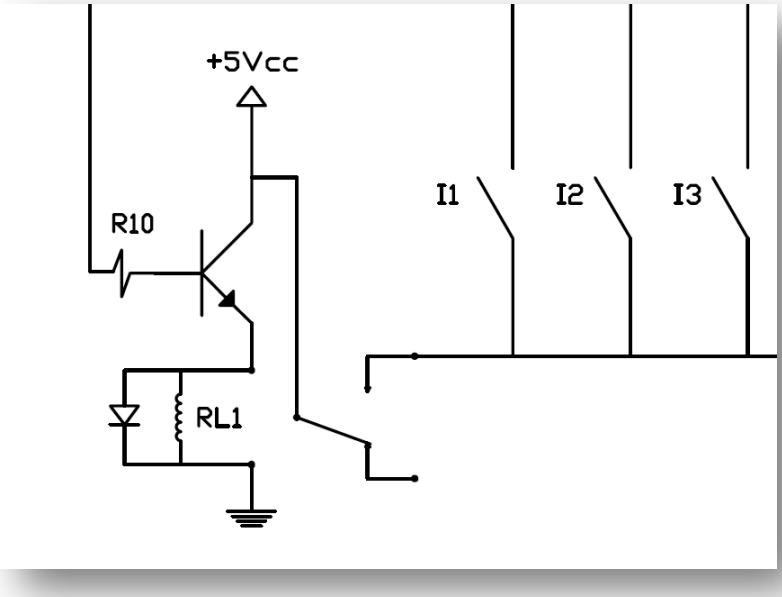
Tanto no esquemático, quanto no diagrama unifilar, se pode notar a presença de um transistor e um relé. Esses componentes foram integrados ao circuito para eliminar um risco de curto-circuito. Quando o alarme entrar em modo automático, o controle da iluminação será feito pela placa Arduino Mega 2560, através das saídas indicadas para cada lâmpada. Sendo assim, se o usuário deixasse um interruptor ligado e ativasse o alarme, poderia ocasionar um curto-circuito proveniente de dois sinais distintos de 5V.

Para evitar tal problema, foi utilizada uma saída da placa diretamente ligada na base de um transistor BC337. Assim, quando a saída for colocada em nível alto, uma corrente equivalente circulará entre emissor e coletor do transistor. O emissor do transistor foi conectado na bobina de um relé 5V/24V, logo, quando a corrente circular entre o coletor e emissor irá acionar a bobina do relé. Este, por sua vez, tem o terminal comum ligado à tensão de 5V e, no contato normalmente aberto (NA), foram ligados todos os terminais comuns das chaves interruptoras.

Desse modo, quando o programa entra no modo automático de controle da iluminação, a saída ligada na base do transistor é colocada em nível baixo e, como o relé não fecha o contato NA, o barramento dos interruptores não recebe a tensão de 5V, eliminando o risco de curto-circuito pelas chaves.

Além disso, foi introduzido um resistor de 1kΩ em série com a base do transistor, com a finalidade de evitar altas correntes que possam queimar o transistor. E, também, um diodo em anti-paralelo com a bobina do relé, para quando ocorrer o descarregamento da bobina não surgirem correntes reversas no emissor do transistor, o que também poderiam levar à queima do mesmo. Essa montagem pode ser visualizada com mais detalhe na Figura 48.

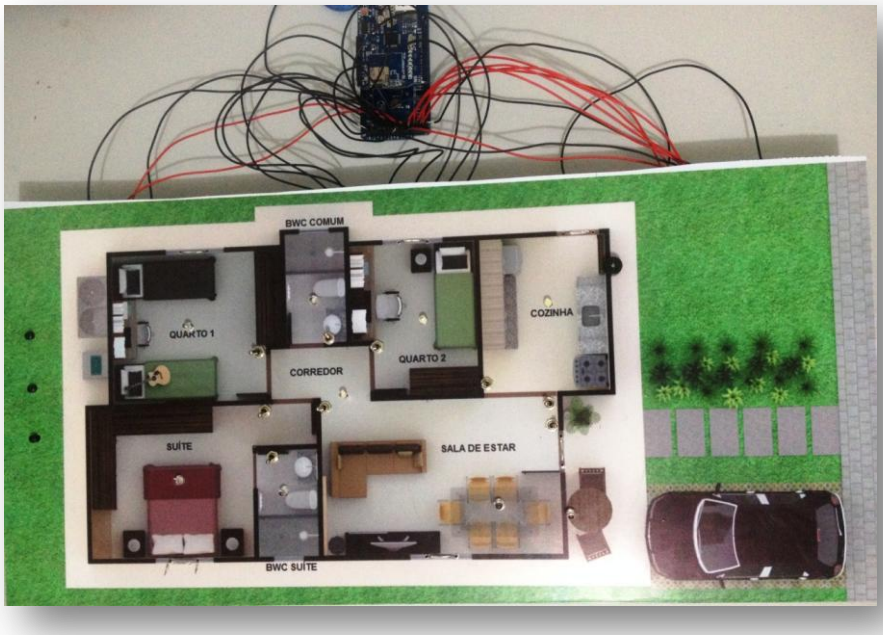
*Figura 48 - Detalhe do circuito de chaveamento da iluminação automática*




Fonte: Próprio Autor

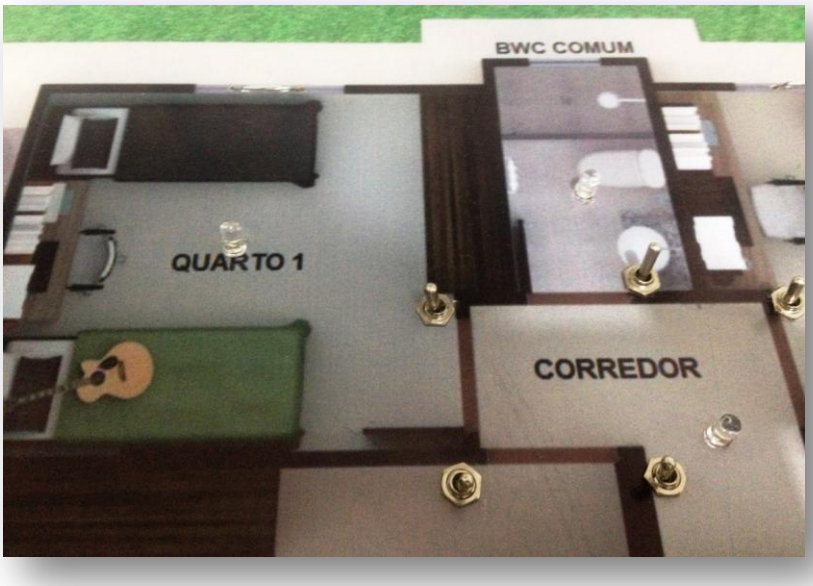
Por fim, na Figura 49, podemos ver uma visão geral do protótipo montado. Os *leds,* interruptores e sensores podem ser vistos com detalhe na Figura 50, assim como, o detalhe das chaves *push-button* são visualizados na Figura 51 e o *buzzer* do alarme na Figura 52.

*Figura 49 - Visão geral do protótipo*




Fonte: Próprio Autor

*Figura 50 - Detalhe dos leds, interruptores e sensores*




Fonte: Próprio Autor

*Figura 51 - Detalhe das chaves push-button de seleção do modo de alarme*


Fonte: Próprio Autor

*Figura 52 - Detalhe do buzzer do alarme*


Fonte: Próprio Autor

# 12 CONSIDERAÇÕES FINAIS

O resultado final do projeto foi satisfatório, pois foi possível cumprir todos os objetivos gerais e específicos previamente estabelecidos. Além disso, alguns problemas, encontrados ao longo do projeto, fizeram com que houvesse a necessidade da busca de novos conhecimentos para solucionar tais problemas.

Hoje em dia, com as grandes empresas entrando forte no mercado de automação residencial, popularizado pelo termo em inglês *domotics,* é perceptível que não seria viável a implementação deste projeto, tanto por limitações físicas quanto tecnológicas. Porém, o uso de plataformas como Elipse E3 e Arduino, nos mostra a grande abrangência que tais recursos podem proporcionar no campo da engenharia.

A comunicação entre o Elipse E3 e o Arduino não pode ser estabelecida utilizando o RTC, quando o RTC era inicializado a comunicação falhava, provavelmente por um choque de memórias ou registradores utilizados por cada biblioteca. Esse foi o único problema que não pode ser resolvido. Como o uso do RTC, não era inicialmente um objetivo do projeto, esse foi deixado de lado, e fica como sugestão para projetos futuros. Ainda como sugestão de novos projetos, poderia ser utilizado o Elipse Mobile como plataforma de supervisório, esse permite que o usuário acesse a planta via *smartphone*. Ou ainda, um supervisório gratuito, ou de baixo custo, que tornaria o projeto economicamente mais viável. Outra sugestão é a utilização de uma *shield* que possibilite fazer a comunicação via *wi-fi,* eliminando o uso de cabos *ethernet*.

## ANEXOS

### ANEXO A – Programação completa do Arduino.

```
#include <TimerOne.h> //Biblioteca da Interrupção de Tempo
#include <SPI.h>      //Bibliotecas da shield Ethernet
#include <Ethernet.h>
#include "Mudbus.h" //Bibliotecas da comunicação Modbus
#include "M0dbus.h"

Mudbus Mb; //Port 502 (defined in Mudbus.h) MB_PORT

//Definindo os pinos das entradas
#define janela_quarto1 22
#define janela_quarto2 24
#define porta_cozinha 26
#define porta_sala 28
#define janela_sala 30
#define janela_suite 32
#define alarme_ligado 34
#define alarme_viajem 36
#define desliga_alarme 38
//Definindo os pinos das saídas
#define LED_quarto1 33
#define LED_bwc_comum 35
#define LED_corredor 37
#define LED_quarto2 39
#define LED_cozinha 41
#define LED_suite 43
#define LED_bwc_suite 45
#define LED_sala 47
#define LED_externo 49
#define ilum_auto 40
#define buzzer 44
//Declaração de variáveis auxiliares
int aux_alarme_ligado = 0;
int aux_alarme_viajem = 0;
int dispara = 0;
int aux_aberto =0;
int aux_dispara;
unsigned long inicio =0;
byte contando = false;
int count = 1;

//Função para atualizar as memórias da comunicação
//Essa função será chamada na interrupção de tempo
void atualiza()
{
```

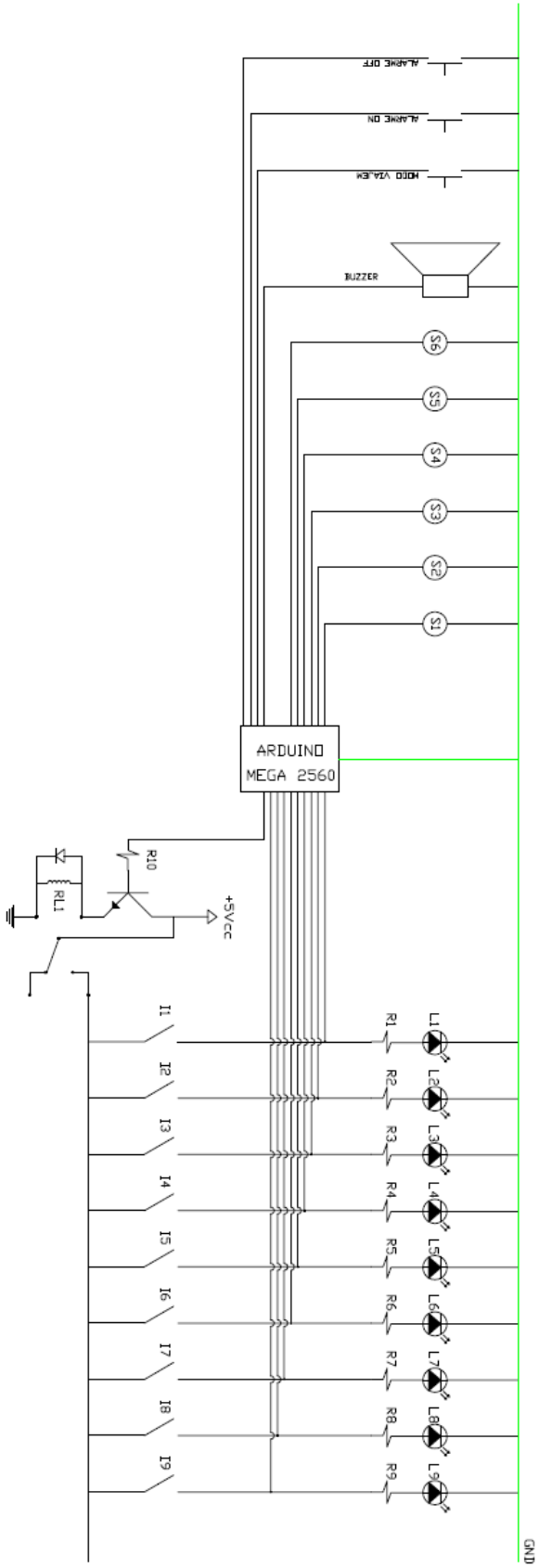
```
Mb.R[0] = digitalRead(janela_quarto1);
Mb.R[1] = digitalRead(janela_quarto2);
Mb.R[2] = digitalRead(porta_cozinha);
Mb.R[3] = digitalRead(porta_sala);
Mb.R[4] = digitalRead(janela_sala);
Mb.R[5] = digitalRead(janela_suite);
Mb.R[6] = (aux_alarme_ligado + aux_alarme_viajem);
Mb.R[7] = aux_aberto;
Mb.R[8] = aux_dispara;

Mb.R[9] = digitalRead(LED_quarto1);
Mb.R[10] = digitalRead(LED_bwc_comum);
Mb.R[11] = digitalRead(LED_corredor);
Mb.R[12] = digitalRead(LED_quarto2);
Mb.R[13] = digitalRead(LED_cozinha);
Mb.R[14] = digitalRead(LED_suite);
Mb.R[15] = digitalRead(LED_bwc_suite);
Mb.R[16] = digitalRead(LED_sala);
Mb.R[17] = digitalRead(LED_externo);
}

void setup()
{
uint8_t mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0x51, 0x06 };
uint8_t ip[] = { 192, 168, 1, 50 };
uint8_t gateway[] = { 192, 168, 1, 1 };
uint8_t subnet[] = { 255, 255, 255, 0 };
Ethernet.begin(mac, ip, gateway, subnet);

Timer1.initialize(700);
Timer1.attachInterrupt(atualiza);
delay(5000);
Serial.begin(115200);
//Declarando as entradas e ativando PULLUP
pinMode(janela_quarto1, INPUT_PULLUP);
pinMode(janela_quarto2, INPUT_PULLUP);
pinMode(porta_cozinha, INPUT_PULLUP);
pinMode(porta_sala, INPUT_PULLUP);
pinMode(janela_sala, INPUT_PULLUP);
pinMode(janela_suite, INPUT_PULLUP);
pinMode(alarme_ligado, INPUT_PULLUP);
pinMode(alarme_viajem, INPUT_PULLUP);
pinMode(desliga_alarme, INPUT_PULLUP);
//Declarando as saídas
pinMode(LED_quarto1, OUTPUT);
pinMode(LED_bwc_comum, OUTPUT);
pinMode(LED_corredor, OUTPUT);
pinMode(LED_quarto2, OUTPUT);
pinMode(LED_cozinha, OUTPUT);
pinMode(LED_suite, OUTPUT);
```

```
pinMode(LED_bwc_suite, OUTPUT);
pinMode(LED_sala, OUTPUT);
pinMode(LED_externo, OUTPUT);
pinMode(ilum_auto, OUTPUT);
pinMode(buzzer, OUTPUT);
pinMode(53, OUTPUT);
pinMode(10,OUTPUT);
digitalWrite(10,LOW);
}

void loop()
{
Mb.Run();
//LÓGICA DE DISPARO DO ALARME
if (!digitalRead(janela_quarto1) || !digitalRead(janela_quarto2) || !digitalRead(porta_cozinha) ||
!digitalRead(porta_sala) || !digitalRead(janela_sala) || !digitalRead(janela_suite)) {
        dispara = 1;
}
else {
        dispara = 0;
}
//INICIA LÓGICA DO ALARME COMUM
if (digitalRead(alarme_ligado) == LOW && dispara == 0) {
        aux_alarme_ligado = 1;
}
if (aux_alarme_ligado == 1 && dispara == 1) {
        while (digitalRead(desliga_alarme)) {
        aux_dispara = 1;
        digitalWrite(ilum_auto, LOW);
        digitalWrite(buzzer, HIGH);
        digitalWrite(LED_quarto1, HIGH);
        digitalWrite(LED_quarto2, HIGH);
        digitalWrite(LED_suite, HIGH);
        digitalWrite(LED_corredor, HIGH);
        digitalWrite(LED_bwc_comum, HIGH);
        digitalWrite(LED_sala, HIGH);
        digitalWrite(LED_bwc_suite, HIGH);
        digitalWrite(LED_externo, HIGH);
        digitalWrite(LED_cozinha, HIGH);
        delay(500);
        digitalWrite(buzzer, LOW);
        digitalWrite(LED_quarto1, LOW);
        digitalWrite(LED_quarto2, LOW);
        digitalWrite(LED_suite, LOW);
        digitalWrite(LED_corredor, LOW);
        digitalWrite(LED_bwc_comum, LOW);
        digitalWrite(LED_sala, LOW);
        digitalWrite(LED_bwc_suite, LOW);
        digitalWrite(LED_externo, LOW);
        digitalWrite(LED_cozinha, LOW);
```

```
            delay(500);
            }
}
// FINAL LÓGICA ALARME COMUM E INICIO LÓGICA ALARME VIAJEM
if (digitalRead(alarme_viajem) == LOW && dispara == 0 ) {
            aux_alarme_viajem = 1;
            digitalWrite(ilum_auto, LOW);
}
if (aux_alarme_viajem && !dispara) {
            digitalWrite(LED_externo, HIGH);
            if(!contando){
            contando = true;
            inicio = millis();
            }
            if (millis()-inicio > 5000) {
            count = count + 1;
            inicio = millis();
}
switch (count){
case 1:
digitalWrite(LED_corredor, LOW);
digitalWrite(LED_bwc_suite, LOW);
digitalWrite(LED_suite, HIGH);
digitalWrite(LED_cozinha, HIGH);
break;
case 2:
digitalWrite(LED_suite, LOW);
digitalWrite(LED_cozinha, LOW);
digitalWrite(LED_sala, HIGH);
digitalWrite(LED_quarto2, HIGH);
break;
case 3:
digitalWrite(LED_sala, LOW);
digitalWrite(LED_quarto2, LOW);
digitalWrite(LED_quarto1, HIGH);
digitalWrite(LED_bwc_comum, HIGH);
break;
case 4:
digitalWrite(LED_quarto1, LOW);
digitalWrite(LED_bwc_comum, LOW);
digitalWrite(LED_corredor, HIGH);
digitalWrite(LED_bwc_suite, HIGH);
break;
default:
count = 1;
contando = false;
}
}
if (aux_alarme_viajem == 1 && dispara == 1) {
            while (digitalRead(desliga_alarme)) {
```

```
            aux_dispara = 1;
            digitalWrite(ilum_auto, LOW);
            digitalWrite(buzzer, HIGH);
            digitalWrite(LED_quarto1, HIGH);
            digitalWrite(LED_quarto2, HIGH);
            digitalWrite(LED_suite, HIGH);
            digitalWrite(LED_corredor, HIGH);
            digitalWrite(LED_bwc_comum, HIGH);
            digitalWrite(LED_sala, HIGH);
            digitalWrite(LED_bwc_suite, HIGH);
            digitalWrite(LED_externo, HIGH);
            digitalWrite(LED_cozinha, HIGH);
            delay(500);
            digitalWrite(buzzer, LOW);
            digitalWrite(LED_quarto1, LOW);
            digitalWrite(LED_quarto2, LOW);
            digitalWrite(LED_suite, LOW);
            digitalWrite(LED_corredor, LOW);
            digitalWrite(LED_bwc_comum, LOW);
            digitalWrite(LED_sala, LOW);
            digitalWrite(LED_bwc_suite, LOW);
            digitalWrite(LED_externo, LOW);
            digitalWrite(LED_cozinha, LOW);
            delay(500);
            }
}
// FINAL LÓGICA ALARME VIAJEM
//DESATIVANDO A ILUMINAÇÃO AUTOMÁTICA QUANDO O ALARME É DESLIGADO
if (digitalRead(desliga_alarme) == LOW) {
            aux_alarme_ligado = 0;
            aux_alarme_viajem = 0;
            aux_dispara = 0;
            digitalWrite(ilum_auto, HIGH);
            digitalWrite(LED_quarto1, LOW);
            digitalWrite(LED_quarto2, LOW);
            digitalWrite(LED_suite, LOW);
            digitalWrite(LED_corredor, LOW);
            digitalWrite(LED_bwc_comum, LOW);
            digitalWrite(LED_sala, LOW);
            digitalWrite(LED_bwc_suite, LOW);
            digitalWrite(LED_externo, LOW);
            digitalWrite(LED_cozinha, LOW);
}

if((digitalRead(alarme_ligado) == LOW || digitalRead(alarme_viajem) == LOW) && dispara == 1){
            aux_aberto = 1;
}
else aux_aberto = 0;
}
```

ANEXO B – Diagrama unifilar de montagem do protótipo.



## REFERÊNCIAS


ARDUINO. **Arduino Ethernet Shield.** Disponível em: <https://www.arduino.cc/en/Main/ArduinoEthernetShield>. Acesso em: 11 maio 2016.

ARDUINO. **Arduino MEGA 2560.** Disponível em: <https://www.arduino.cc/en/Main/ArduinoBoardMega2560>. Acesso em: 11 maio 2016.

SOUZA, Fábio. **Documentário sobre Arduino.** Disponível em: <http://www.embarcados.com.br/documentario-sobre-arduino/>. Acesso em: 11 maio 2016.

SOUZA, Fábio. **Arduino Mega 2560.** Disponível em: <http://www.embarcados.com.br/arduino-mega-2560/>. Acesso em: 11 maio 2016.

SOUZA, Fábio. **Placas Arduino - História até o Arduino UNO.** Disponível em: <http://www.embarcados.com.br/historia-ate-o-arduino-uno/>. Acesso em: 11 maio 2016.

THOMSEN, Adilson. **COMO COMUNICAR COM O ARDUINO ETHERNET SHIELD W5100.** Disponível em: <http://blog.filipeflop.com/arduino/tutorial-ethernet-shield-w5100.html>. Acesso em: 11 maio 2016.

ARDUINO The Documentary. Gijón: Laboral Centro de Arte y Creación Industrial, 2010. P&B.

VIANNA, William da Silva. **SISTEMA SCADA SUPERVISÓRIO.** 2008. 67 f. Tese (Doutorado) - Curso de Eng. Elétrica, Instituto Federal Fluminense de EducaÇÃo CiÊncia e Tecnologia, Campo dos Goytacazes, 2008.

SOFTWARE, Elipse. **Elipse E3.** Disponível em: <http://www.elipse.com.br/port/e3.aspx>. Acesso em: 11 maio 2016.

SILVA, Ana Paula Gonçalves da; SALVADOR, Marcelo. **O que são sistemas supervisórios?** Disponível em: <http://www.wectrus.com.br/artigos/sist_superv.pdf>. Acesso em: 11 maio 2016.

SOFTWARE, Elipse. **SISTEMA DE SUPERVISÃO E CONTROLE DE TEMPO REAL PARA APLICAÇÕES CRÍTICAS.**Disponível em: <http://www.elipse.com.br/port/assets/folders/elipse_folder-E3.pdf>. Acesso em: 11 maio 2016.

FISCHER, Guilherme da Silva. **Supervisório Elipse E3.** Joinville: Udesc, 2012.

CAMPOS DA LUZ, Andreza et al. **MANUAL PARA ELABORAÇÃO DE TRABALHOS ACADÊMICOS DA UDESC: TRABALHO DE CONCLUSÃO DE CURSO E RELATÓRIO DE ESTÁGIO.** Santa Catarina: Florianópolis, 2014. Disponível em: http://www.cav.udesc.br/arquivos/id_submenu/373/cestcav_manual_relatorio_bu_udesc.pdf Acesso em: 22 novembro 2016