CEEL - ISSN 2596-2221
Universidade Federal de Uberlândia
07 a 11 de dezembro de 2020

# SISTEMA REGISTRADOR E DETECTOR DE INTERRUPÇÕES MOMENTÂNEAS NO FORNECIMENTO DE ENERGIA ELÉTRICA COM ESP32

Otávio Cosme Matias*[1], Lucas Frederico Jardim Meloni[1], Rodrigo Menezes Sobral Zacaroni[1] e Gustavo Lobato Campos[1]

[1]IFMG – Instituto Federal de Minas Gerais – *Campus* Formiga

***Resumo* – Interrupções no fornecimento de energia elétrica causam severos prejuízos aos consumidores como, paradas em linhas de produção, falhas em sistemas de refrigeração e perdas de insumos. Registrar e a sinalizar a data e horário destas falhas é importante para futuras ações preventivas ou dimensionar fontes auxiliares de energia. Neste artigo será proposto um sistema de monitoramento e registro de interrupções momentâneas de energia, através da plataforma ESP32, que será capaz de registrar e sinalizar de forma remota os momentos em que ocorreram as falhas, através de mensagens enviadas via *Bluetooth*. Serão discutidos os aspectos de construção dos circuitos de alimentação, detecção de falhas e da saída a relé e programação desenvolvida para registrar a data e hora das falhas e envio de mensagens.**

***Palavras-Chave* – ESP32, Bluetooth, Arduino, Monitoramento de fornecimento.**

## MOMENTARY ELECTRICITY SUPPLY INTERRUPTION DETECTOR AND REGISTRATION SYSTEM USING ESP32

***Abstract* – Short time interruptions on electricity supplying may cause several consumers losses, such as production lines stops or failures in refrigeration systems. Thus, registering and signalizing the time of occurrence is important for taking preventive and corrective actions. This paper presents a system to monitor the power supply interrupts, using the ESP32 platform will be proposed. This system is able to remotely register and signal the moments when the failures occurred, by sending messages through the Bluetooth protocol. The construction aspects of the supply circuits, fault detection and relay output will be shown, as well as the algorithm used to record the date and time of the faults and sending messages.**

***Keywords* - ESP32, Bluetooth, Arduino, Supply monitoring.**

*otaviocmatias@gmail.com

## I. INTRODUÇÃO

Muitos consumidores preocupam-se em monitorar seu consumo de energia elétrica de forma independente, seja para confrontar seus registros com as leituras realizadas pela concessionária, ou para adquirir informações que possam ser usadas para readequar seus perfis de consumo [1]–[3]. Detectar a ocorrência de interrupções do fornecimento de energia também é fundamental para evitar danos na produção de produtos perecíveis [4], ou interrupção do funcionamento de sistemas de telecomunicação [5]. Com a detecção é possível realizar o acionamento de fontes auxiliares de energia, como grupos geradores emergenciais [6].

A regulamentação da qualidade da energia elétrica é estabelecida pelo Módulo 8 do Procedimentos de Distribuição de Energia Elétrica no Sistema Elétrico Nacional (PRODIST) [7]. Neste documento, a interrupção momentânea de tensão é caracterizada por eventos com duração inferior ou igual a 3 segundos, com amplitude máxima não sendo inferior a 10% (0,1 p.u.).

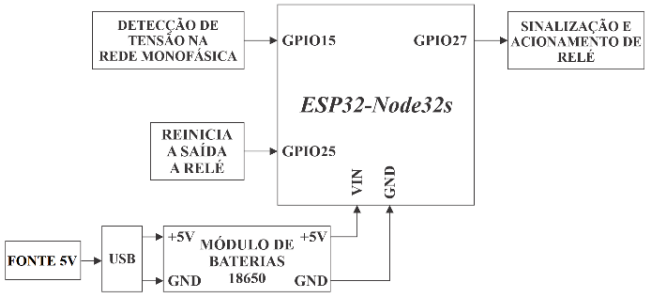
No ambiente industrial, existem ferramentas dedicadas ao registro e monitoramento de grandezas elétricas, como medidores de energia com a capacidade de *data logger* [8]. Entretanto estes equipamentos são caros e economicamente inviáveis para pequenos consumidores ou usuários residenciais.

Neste artigo propõe-se um sistema microcontrolado de baixo custo, baseado em ESP-32, capaz de detectar interrupções momentâneas no fornecimento de energia, informar remotamente a data e horário da ocorrência e acionar fontes auxiliares para suprimento de energia, como pequenos grupos geradores. Todos os circuitos e recursos de *software* foram desenvolvidos pelos autores, com o intuito de promover uma forma simples o registro das interrupções. Serão discutidos os aspectos construtivos dos circuitos utilizados e também a programação desenvolvida para realizar o registro do tempo e o envio das mensagens.

## II. DESCRIÇÃO DO SISTEMA REGISTRADOR

A Figura 1 mostra uma representação em diagrama de blocos do subsistema do detector e registrador de interrupções momentâneas de energia. Existem três subsistemas envolvidos, sendo um responsável pela detecção da tensão da rede monofásica, outro para acionamento de uma saída relé, para acionamento de sistemas emergenciais de fornecimento de energia, e um último responsável pela reinicialização. Para alimentação dos circuitos, utilizou-se um módulo de baterias de lítio, com capacidade para alimentar os componentes eletrônicos durante a queda de energia e recarregar durante a presença da tensão da rede.

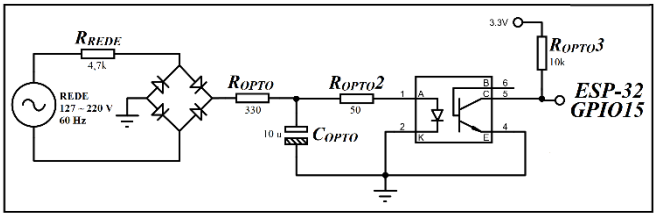
Figura 1: Representação do sistema detector em diagrama de blocos. (Fonte: produzido pelos autores).




### *A. Circuito de detecção de tensão*

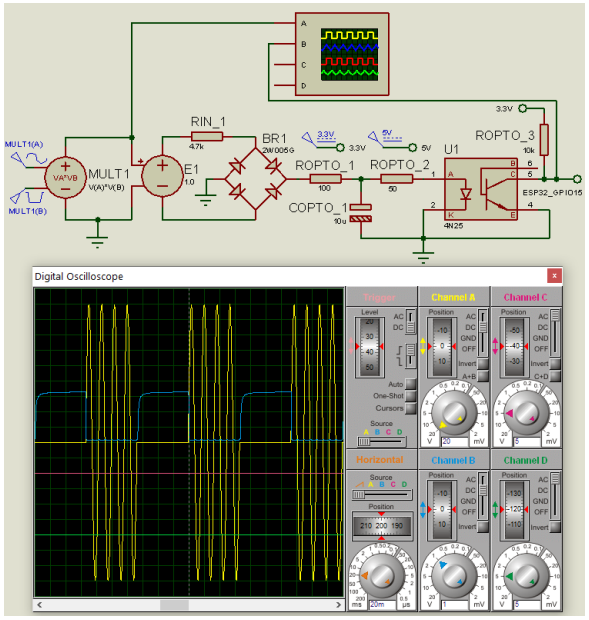
A Figura 2 ilustra o circuito utilizado para detectar a tensão da rede. A rede elétrica é conectada a uma ponte de diodos, contendo um filtro RC formado pelo resistor $R_{OPTO}$ e pelo capacitor $C_{OPTO}$. A saída do filtro é conectada à entrada de um optoacoplador com saída à transistor. O divisor de tensão formado pelos resistores $R_{REDE}$ e $R_{OPTO}$ fornece uma tensão ao capacitor $C_{OPTO}$, que carregará e produzirá uma tensão contínua, que por sua vez acionará o *LED* interno do optoacoplador. Isso faz com que o transistor de saída seja saturado, produzindo um nível lógico baixo no pino GPIO15 do ESP32. Caso ocorra uma queda no fornecimento de energia, valor nulo de tensão, o transistor do optoacoplador entrará em corte e o nível lógico em GPIO15 será alto.

Figura 2: Circuito para detecção da tensão da rede elétrica. (Fonte: produzido pelos autores).



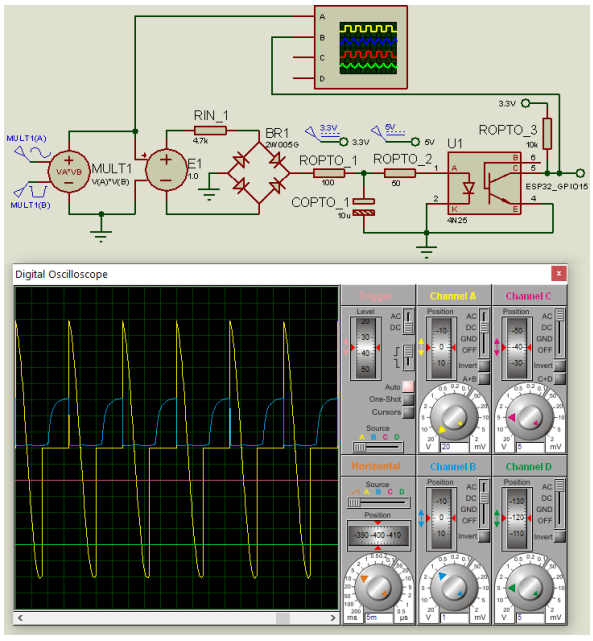

A Figura 3 mostra uma simulação realizada através do software *Labcenter® Proteus Demo*. A forma de onda em amarelo, do canal *Channel A*, demonstra a rede elétrica sofrendo consecutivas interrupções momentâneas. A forma de onda em azul, do canal *Channel B*, ilustra a saída do optoacoplador, que irá para o GPIO15 do ESP-32. Nota-se que uma tensão de 3,3 V é produzida sempre que o fornecimento é interrompido. Portanto, através da detecção das transições por borda de subida no GPIO15 é possível registrar o momento em que ocorreu a falha.

Figura 3: Simulação do circuito de detecção de tensão da rede elétrica no *Labcenter Proteus® Demo*. (Fonte: produzido pelos autores).




A Figura 4 mostra a simulação de um caso muito severo, onde a tensão da rede elétrica é interrompida a cada meio ciclo e a aplitude também é 10% menor. Neste caso o detector ainda é capaz de informar corretamente as ocorrências para o ESP-32.

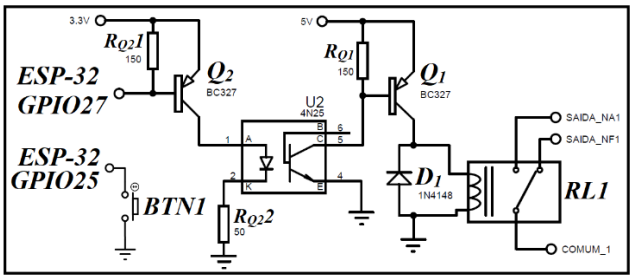
Figura 4: Simulação do circuito de detecção de tensão da rede elétrica no *Labcenter Proteus® Demo*. (Fonte: produzido pelos autores).



### *B. Circuito da saída à rele e reinicialização*

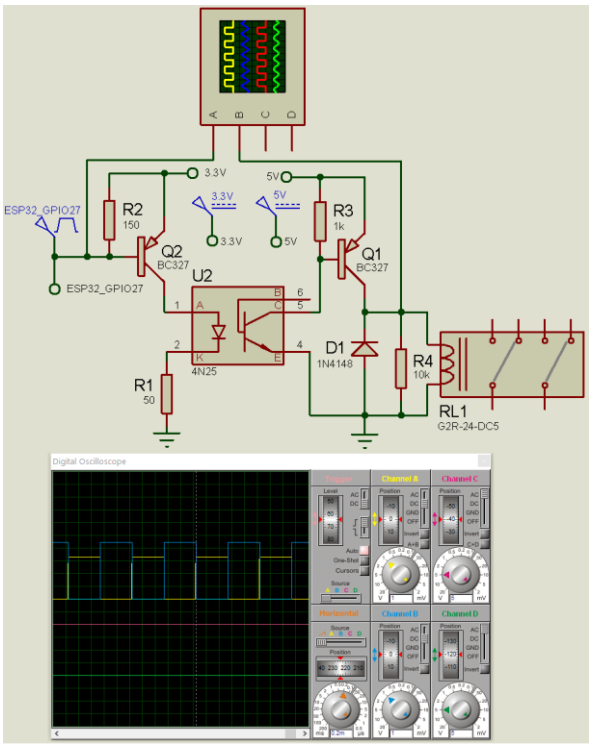
A Figura 5 mostra o circuito para acionamento da fonte auxiliar de energia, através do relé acionado pelo GPIO 27 do módulo ESP32. A programação desenvolvida enviará um nível lógico baixo em GPIO27, fazendo com que o transistor Q2 sature. A condução de Q2 também fará o diodo interno do optoacoplador conduzir, permitindo que o transistor de saída sature Q1 e acione o relé RL1. Através do botão BTN1 e da entrada digital GPIO25, configurada com um resistor *pull-up* interno, será possível desativar a saída à relé e reiniciar os registros.

Figura 5: Circuito para acionamento de relé. (Fonte: produzido pelos autores).




A Figura 6 ilustra uma simulação do circuito de acionamento do relé de saída. O sinal aplicado ao GPIO27 do ESP- 32 é mostrado pela forma de onda amarela, do canal *Channel A*, e a tensão no relé RL1 é mostrada na forma de onda azul do canal *Channel B*.

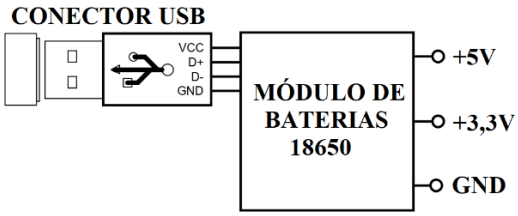
Figura 6: Simulação do circuito para acionamento de relé no *Labcenter Proteus® Demo*. (Fonte: produzido pelos autores).




### *C. Circuitos de alimentação e baterias*

A Figura 7 ilustra o circuito alimentação do ESP-32 e demais componentes. Este circuito utiliza baterias de lítio 18650 e é capaz de proporcionar uma longa duração, de até três horas sem a necessidade de carregamento, sem a utilização de modos de baixo consumo de energia ou *sleep* [9]. Este módulo também inclui um circuito de controle de carga das baterias, que permite sua recarga através de uma fonte de alimentação externa de 5V e de um cabo USB.

Figura 7: Circuito para o sistema de alimentação com baterias. (Fonte: produzido pelos autores).




## III. DESCRIÇÃO DO MÓDULO ESP-32

Os módulos ESP32 são dispositivos dedicados ao desenvolvimento de aplicações portáteis e dispositivos IoT conectados à Internet (*Internet of Things*), como projetos de automação residencial [10]. O suporte a padrões de comunicação é um destaque para o ESP, que além de suportar os protocolos I2C, SPI e UART serial, também possui I2S, CAN, Ethernet, IEEE 802.11 b/g/n/e/i e Bluetooth v4.2 BR/EDR e BLE [11]. Outra grande vantagem é a sua programação, que se dá através da linguagem C/C++ no ambiente *Arduino IDE*, o que também possibilita a migração de alguns projetos já desenvolvidos com outros modelos de placas [10].

### *A. Real Time Clock (RTC)*

O ESP32 possui um relógio de tempo real (*Real Time Clock – RTC*), integrado a placa com 8 KB de memória SRAM, permitindo manter a data atualizada mesmo em estado de *deep-sleep* [11].

A data obtida pelo RTC é do formato *Unix Timestamp*, sendo os segundos corridos desde o primeiro dia de janeiro de 1970 [12]. Por isso, a data depois de obtida é transformada no formato dia/mês/ano horas:minutos:segundos e enviada para o aplicativo.

Segundo [12], para utilizar este recurso, inicialmente carregam-se as bibliotecas *time.h* e *sys/time.h*, conforme mostrado no código abaixo:

```
#include <time.h>
#include <sys/time.h>
```

Em seguida devem ser criadas as estruturas *timeval* de leitura do RTC, como ilustrado abaixo:

```
timeval tv;
```

Para configurar o horário inicial, deve-se utilizar a propriedade *tv_sec*, conforme mostrado abaixo, utilizando-se a conversão do horário atual para o *UNIX timestamp*. Em seguida utiliza-se a função *settimeofday()*, que recebe como argumentos o endereço da variável *tv*.

```
tv.tv_sec = 1599308250;
settimeofday(&tv, NULL);
```

A leitura da data é feita através da função *strftime()* que salva os formatos em um vetor de caracteres *data_formatada* e na estrutura *tm data*, como mostrado abaixo:

```
struct tm data;
char data_formatada[64];
strftime(data_formatada, 64, "%d/%m/%Y %H:%M:%S", &data);
```

### *B. Bluetooth Classic*

Para enviar os dados da placa ESP32 foi utilizada a comunicação *Bluetooth*®, sendo que a placa já conta com o recurso integrado. Este protocolo de comunicação é muito comum em comunicações entre dispositivos móveis e também em projetos de automação com placas *Arduino*. A versão disponível é *Bluetooth* v4.2BR/EDR com taxa de transmissão de até 4Mbps [11]. A biblioteca utilizada para comunicação é *BluetoothSerial.h*, carregada através do comando:

```
#include "BluetoothSerial.h"
```

Em seguida, deve-se inicializar um objeto chamado *BluetoothSerial* SerialBT, para que seja possível utilizar os métodos para envio das mensagens:

```
BluetoothSerial SerialBT;
```

Para enviar uma informação através do protocolo *Bluetooth*, utilizam-se as funçoes *print* e *println*, conforme mostrado abaixo:

```
SerialBT.println(data_formatada);
```

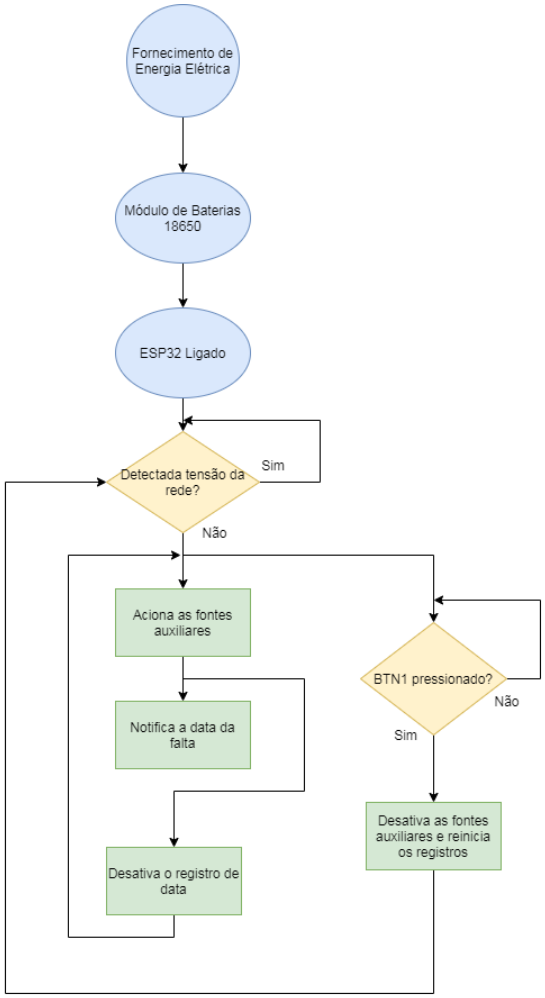
### *C. Aplicativo Serial Bluetooth Terminal*

A placa irá se comunicar com um celular Android através do aplicativo *Serial Bluetooth Terminal*. O aplicativo é um terminal orientado por linhas para dispositivos com interface serial que permite a conversão *Bluetooth*® para serial em um dispositivo Android [13]. A conexão estabelecida entre os dispositivos é bidirecional, ou seja, recebe e envia dados.

## IV. FUNCIONAMENTO DO CÓDIGO

O código desenvolvido consiste na detecção de falha na tensão da rede, acionando fontes auxiliares de energia e notificando ao usuário a data da falha, após a restituição do fornecimento de energia o processo é reiniciado através de um botão. A Figura 8 apresenta o fluxograma de execução do código.

Figura 8: Fluxograma para explicação do funcionamento do programa do detector de interrupções momentâneas no fornecimento de energia. (Fonte: produzido pelos autores).




Na Figura 8, realiza-se inicialmente a leitura da entrada digital GPIO15, para verificar se ocorreu a falha no fornecimento de energia. Caso tenha ocorrido, realiza-se a leitura do RTC através da função *gmtime(&tt)*, conforme ilustrado abaixo. Neste momento também aciona-se o relé de emergência, através do GPIO27.

```
if(digitalRead(15) == HIGH){
  digitalWrite(27,HIGH);
   struct tm data;
   time_t tt = time(NULL);
  data = *gmtime(&tt);
}
```

Por fim, realiza-se a conversão da data e envia-se a mensagem através de *Bluetooth*, conforme mostrado abaixo:

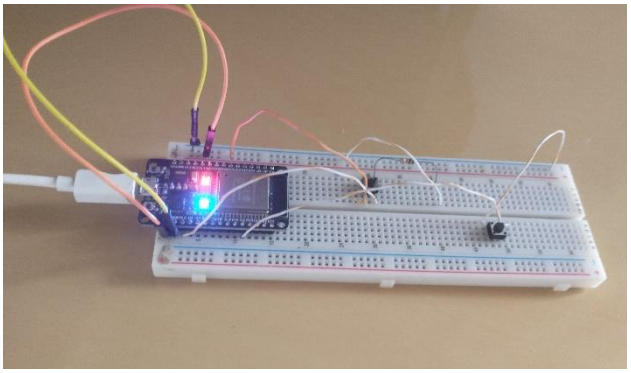
```
struct tm data;
char data_formatada[64];
strftime(data_formatada, 64, "%d/%m/%Y %H:%M:%S", &data);
SerialBT.println(data_formatada);
```

Também é possível armazenar o valor numérico do *Unix timestamp* em um vetor de dados, salvo na memória interna do ESP32, para registrar as datas de ocorrência das falhas de energia.

## V. TESTES DO SISTEMA REGISTRADOR DE FALTA

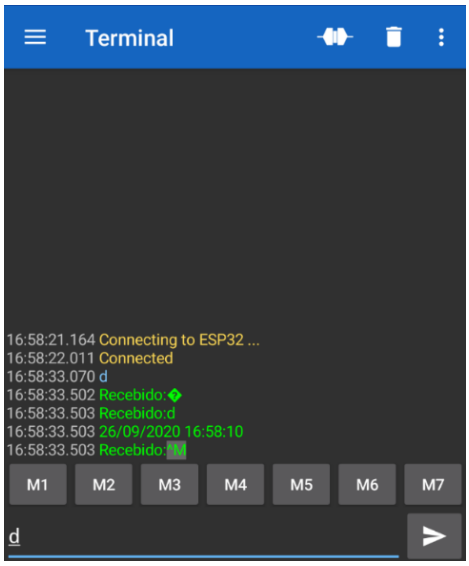
A Figura 9 mostra o teste realizado para a saída à relé e reinicialização. Pela característica de entrada digital do circuito detector de tensão, foi utilizado um botão para enviar o níveis lógicos alto e baixo para o GPIO27, de modo a acionar o relé. Para simular o relé, utilizou-se o GPIO2, LED interno da placa ESP32 DevKit C, para verificar a saída. A entrada digital GPIO25, configurada com um resistor *pull-up* interno, possibilita desativar a saída à relé e reiniciar os registros.

Figura 9: Teste do circuito da saída à relé e reinicialização. (Fonte: produzido pelos autores).



A Figura 10 apresenta o resultado obtido do teste da montagem da Figura 8. O aplicativo *Bluetooth Serial Terminal* conseguiu receber as informações do *ESP32* contendo a data da ocorrência, juntamente com a hora, minutos e segundos.

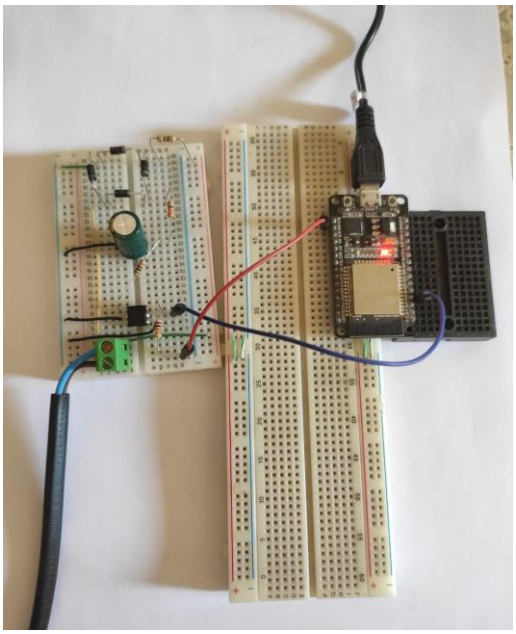
Figura 10: Interface do Aplicativo *Serial Bluetooth Terminal*. (Fonte: captura de tela produzida pelos autores).




Posteriormente, foi feita uma montagem com os componentes para o detector de tensão, mostrada na Figura 11.

Neste caso, também confirmou-se os resultados de simulação mostrados na Figura 3, sendo obtido através do teste o pleno funcionamento desse circuito.

Figura 11: Circuito detector de tensão. (Fonte: produzido pelos autores).



Até o momento da confecção deste artigo, apenas o circuito detector de tensão foi testado na prática. Futuramente pretende-se montar e também integrar os módulos de bateria e acionamento por relé.

## VI. CONCLUSÕES

Este artigo apresentou o desenvolvimento de uma aplicação de monitoramento do fornecimento de energia elétrica de baixo custo com notificação via *Bluetooth*®. Os testes do monitoramento implementado funcionaram corretamente, permitindo ao ESP32 notificar o usuário sobre a falta e acionar as fontes de emergência para garantir o fornecimento de energia elétrica durante a falha.

Futuramente, pretende-se construir uma placa de baixo custo para esse sistema e realizar novos testes, assim como expandir as opções de notificação, possibilitando a notificação por e-mail, por exemplo.

## AGRADECIMENTOS

Os autores agradecem ao IFMG – Campus Formiga, em especial ao GSE – Grupo de Soluções em Engenharia, pela interação e colaboração no desenvolvimento do presente trabalho.

## REFERÊNCIAS

[1] J. L. G. de BRITO, *Sistema para monitoramento de consumo de energia elétrica particular , em tempo real e não invasivo utilizando a tecnologia Arduino*. Londrina - PR: Trabalho de Conclusão de Curso Universidade Estadual de Londrina, 2016.

[2] B. A. S. Oliveira, “Desenvolvimento de um protótipo de sistema de monitoramento de energia elétrica via internet,” vol. 12, no. 1, pp. 48–61, 2018.

[3] M. ROSA, *Monitoramento de interrupções de fornecimento em uma rede elétrica monofásica residencial utilizando microcontrolador e supervisório scadabr*. Ponta Grossa: Paraná, Trabalho de Conclusão de Curso Universidade Tecnológica Federal do, 2016.

[4] G. R. Teixeira, *Desenvolvimento de Sistema Microcontrolado para Redução de perdas associadas ao processo de refrigeração do leite em pequenas propriedades*. Formgia 0 MG, 2018.

[5] A. B. d’ALCANTARA E. AMARAL, *O impacto da insuficiência no fornecimento de energia elétrica nas empresas brasileiras do setor de telecomunicações*, vol. 1, no. 1. Dissertação de Mestrado Escola Politécnica da Universidade de São Paulo, 2017.

[6] G. M. Affonso, *Automação de grupo gerador para utilização em emergência e redução da demanda no horário de ponta*. Bagé: Trabalho de Conclusão de Curso Universidade Federal do Pampa, 2017.

[7] ANEEL, *Procedimentos de Distribuição de Energia Elétrica no Sistema Elétrico Nacional - PRODIST - Módulo 8 - Qualidade da Energia Elétrica*. 2020.

[8] Fluke, *1732/1734 Energy Logger - Manual do Usuário*. Fluke, 2017.

[9] T. K. Hareendran, “Portable Power - 18650 Battery Shield for Raspberry Pi & Arduino,” *Electro Schematics*. https://www.electroschematics.com/battery-shield/ (accessed Sep. 28, 2020).

[10] J. W. Santos, R. Capelin, and D. E. L. Junior, *Sistema De Automatização Residencial De Baixo Custo Controlado Pelo Microcontrolador Esp32 E*. Ponta Grossa: Trabalho de Conclusão de Curso Universidade Tecnológica Federal do Paraná, 2019.

[11] Espressif, “ESP 32 Datasheet,” 2020. https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. (accessed Sep. 16, 2020).

[12] J. Morais, “ESP32 - Utilizando o RTC Interno para Datas,” *Silício, Portal Vida de*, 2019. https://portal.vidadesilicio.com.br/esp32-utilizando-o-rtc-interno-para-datas/ (accessed Oct. 02, 2020).

[13] K. Morich, “Kai Morich’s Android Apps,” *Google Play Store*, 2020. https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal&hl=pt (accessed Oct. 02, 2020).