UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS SOBRAL

Engenharia Elétrica
Instalações Elétrica Hospitalares

Gerenciamento de Blackout

Julio Cesar Ferreira Lima

Prof: Éber de Castro Diniz

Sobral, **Dezembro** de **2017**.

SUMÁRIO

INTRODUÇÃO

Histórico:

Inaugurada em 1925 a Santa Casa de Misericórdia de Sobral é referência regional em atendimento de saúde. O hospital foi idealizado pelo primeiro bispo da diocese de Sobral, Dom José Tupinambá da Frota.

Ao longo dos anos o hospital continuou evoluindo em tamanho, qualidade e dimensão assistencial, e a partir de 2007 passou a ser um hospital de ensino, buscando desenvolver novas tecnologias em parcerias com as universidades da região. Oferecendo também residência e internato em diversas especialidades médicas.

A Santa Casa de Misericórdia de Sobral possui hoje três unidades hospitalares que atendem juntas atendem cerca de 40 mil pacientes por mês, de diversos municípios da região. O hospital tem como objetivo atender em especial os mais pobres e desfavorecidos. Visando sempre a satisfação de seus colaboradores e usuários, além do reconhecimento como instituição de excelência em gestão e prestação de serviços.

Problemática:

Devido a frequentes faltas na rede de alimentação provida pela distribuidora ENEL, a qualidade de equipamentos caros como as unidades refrigeradoras estava a ser prejudicada, pois por não terem elementos de proteção poderiam danificar-se. Como exemplo, da falta de uma das fases poderia prejudicar muito a vida útil de uma unidade refrigeradora que possui um motor trifásico, já que tal máquina elétrica não poderá funcionar fora dos parâmetros aos quais ela foi projetada.

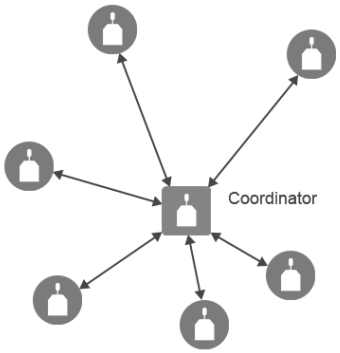
Também há outro problema, que na falta total, há a entrada do gerador que está subdimensionado para a demanda atual da Santa Casa, então com todas as unidades refrigeradores funcionando, a unidade de geração poderá não prover energia suficiente para abastecer todo o hospital em caso de um sinistro de falta total. Neste caso poderá faltar energias em unidades críticas, como UTI, UTI Neo Natal, cirurgias, dentre outras áreas do hospital.

Solução:

Por se tratar de um grupo de risco, a energia elétrica de um hospital não deve ser interrompida em grandes períodos de forma alguma. Então para resolução das problemáticas abordadas acima, foi elaborado pelo diretor do setor de Eletrônica, do hospital um sistema de controle centralizado, onde uma unidade iria verificar a falta ou sinistros que por ventura viessem a ocorrer, e caso ocorresse haveria o desligamento de todas as unidades refrigeradoras até a energia fornecida pela distribuidora pudesse retornar ao seu estado de operação nominal.

O a comunicação entre mestre e escravos sucedeu-se de forma em estrela onde no centro, encontra-se o mestre a fazer todas as medições e permitir ou não o funcionamento dos equipamentos escravos. A configuração em estrela segue conforme a Figura 1.1 abaixo.

Figura 1.1 – Topologia de comunicação em estrela.



Fonte: Google Imagens.

OBJETIVOS

Este trabalho destina-se a revitalização do sistema previamente instalado de controle de acionamento de unidades refrigeradoras, ausência da energia provida pela concessionária de energia, assim aperfeiçoando o sistema antigo e provendo um sistema mais moderno de controle. O uso de um controle mais moderno, embora mais complexo de implementação, torna possível a agregação de futura melhorias, como o controle temporizado, a seletividade remota, acesso a informações em tempo real por meio de um supervisório, além de futuras demandas oriundas das necessidades do hospital em conter o aumento do consumo de energia por meio de iniciativas que evitem o desperdício.

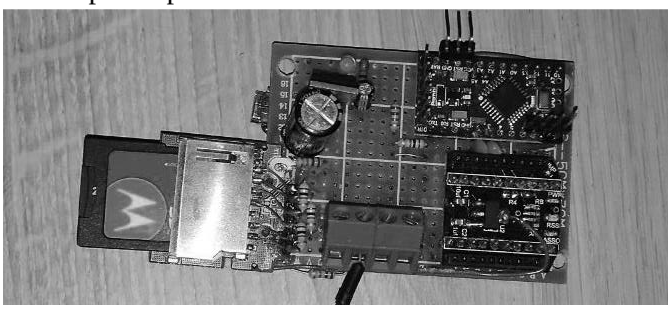
É importante frisar que este trabalho apenas visa a revitalização da unidade que outrora funcionara, mas foi desabilitada. No entanto as modernizações discorridas são muito aconselháveis, dado que elas podem proporcionar um ganho maior em relação a comodidade dos que necessitam das unidades refrigeradoras, assim como também e mais importante a economia de energia, podendo assim setores pouco utilizados em determinados setores poderem ser desabilitados de forma temporizada sem a necessidade de uma gestão pessoal para a execução dessas tarefas, que poderão ser executadas de forma autônoma, caso haja continuidade deste trabalho.

DESENVOLVIMENTO

Protótipo:

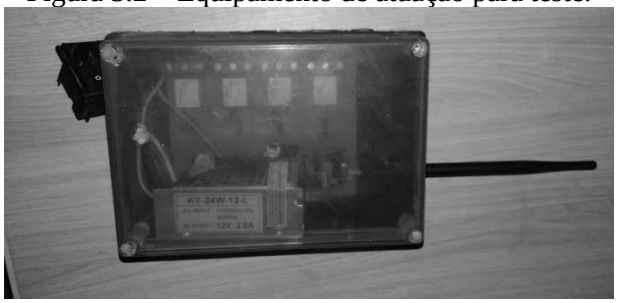
O protótipo desenvolvido é apenas uma placa onde está conectado o micro controlador uma unidade de armazenamentos de endereços de modo segura, esta feita em EEPROM, um RTC (Real Time Clock), assim como o elemento principal para a telemetria, o Xbee. Todos estes elementos foram testados e desenvolvidos interfaces para comunicação, com exceção do RTC, que houve problema quanto a entrega no prazo e não pode ainda ser utilizado.

Figura 3.1 – Placa protótipo de teste de armazenamento e transmissão de dados.



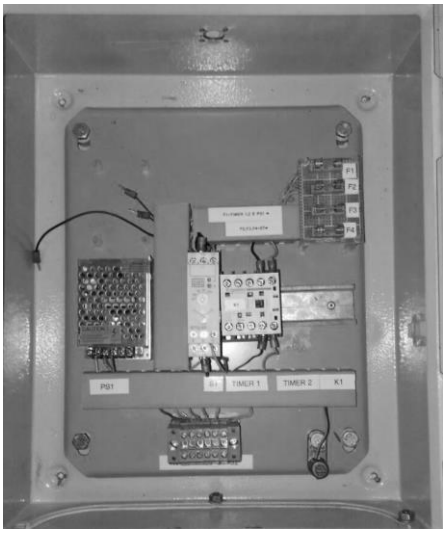
Fonte: Próprio autor.

Figura 3.2 – Equipamento de atuação para teste.



Fonte: Próprio autor.

Figura 3.3 – Quadro de comando onde estará situado a unidade controladora e os relés sensores.



Fonte: Próprio autor.

Configuração do Xbee:

Esta secção será destinada a um resumo das configurações dos parâmetros do Xbee SB1, para configuração em modo AP1. Toda a documentação deste projeto foi desenvolvida em um padrão de texto diferente, portanto encontra-se no anexo deste relatório.

Programação em C++:

A programa foi desenvolvida da forma mais robusta e tradicional para que o software pudesse futuramente ser escalável e ficasse fácil a identificação dos parâmetros e objetos. Todo o software encontra-se no anexo deste relatório.

.

REPOSITÓRIO PÚBLICO

Este trabalho não tem o intuito de ser descontinuado ao fim desta disciplina, de modo que ele seja perpetuado e possa contribuir para a formação de alguém que deseja um fim próximo ao que se destinou aqui. Todo este desenvolvimento, será postado em uma plataforma aberta, tanto os modelos de esquemático da placa que envolve o micro controlador quanto a o esquemático de da placa de potência.

Será adicionada uma licença pública de livre modificação e uso de tudo desenvolvido dentro deste projeto, chamada licença MIT.

O link para acesso do repositório, segue abaixo:

- https://github.com/juloko/Blackout-Control/

A seções dentro do repositório são divididas abaixo:

- code: Diretório destinado a colocação do código rodado dentro do micro controlador. O ambiente de desenvolvimento utilizado foi a IDE Arduino.
- examples: Diretório contendo os exemplos utilizados nas bibliotecas para testar cada parte do projeto.
- libraries: Diretório destinado a colocação das bibliotecas utilizadas dentro do código do micro controlador. As bibliotecas foram produzidas para a linguagem C, compatível com o compilador Arduino.
- kicad: Diretório contendo os projetos de desenvolvimento dos esquemáticos, placa, todos editáveis. Os arquivos encontram-se no formado do software KICAD.
- schematics: Diretório contendo os esquemáticos do transmissor e receptor, arduino pro mini, documentação do xbee, xbee sb1, e placa pronta a qual foi produzida neste trabalho.
- xbee-frames: Diretório contendo os protocolos de envio e recepção de dados do xbee no modo ap. Estes frames são importantes para a realização de testes.
- xbee-profiles: Diretório contendo a configuração pronta para ser carregada para os xbee, precisando apenas da mudança dos parâmetros de identificação, para o caso de a escalabilidade do projeto aumentar.
- REPORT: Arquivo com este relatório.
- README: Arquivo com explanação sobre aspectos gerais de configuração do xbee.

CONCLUSÃO

No processo de desenvolvimento, muito tempo foi gasto no entendimento dos protocolos que envolvem o envio de dados por meio do Xbee, e isso demandou muito tempo, não havia uma interface que se pudesse testar os Xbee's conhecida como *explorer*. Devido à demora em entrega do material necessário, verificou que o hardware necessário para atuar em tal tarefa era semelhante a comunicação serial de do Arduino, então foi feito um projeto para adaptação desta interface, que apresentou muitos problemas de início mas funcionou satisfatoriamente de modo que quando finalmente o *explorer* pode ser usado não se fazia mais necessário.

Outro problema foi o desenvolvimento da placa de conexão do Xbee com o Arduino, que de início por ser apenas um protótipo foi desenvolvida em uma placa perfurada, porém a demanda de circuitos aumentou e a tarefa tornou-se tortuosa, sendo necessária uma placa de circuito impresso mais robusta. O desenvolvimento desta, tornou-se satisfatória, pois há a possibilidade de expansão do projeto para outras unidades da Santa Casa de Misericódia.

Embora o C++ seja uma maneira robusta de programar, por o desenvolvimento de software para micros controladores ser feito de maneiras descentralizada, muitos percalços foram encontrados na programação, com lógicas entre as bibliotecas serem conflitantes. Também houveram problemas quanto ao mau gerenciamento de memória. Em determinada parte do projeto a memória flash necessária, aumentou de forma que o código começou a ter problemas, deste modo precisou-se da troca do micro controlador ATMEGA168-P para o ATMEGA328-P, que basicamente se diferencia na presença de maior memória flash e memória EEPROM.

Contudo o objetivo deste projeto foi alcançado, pois o projeto foi desenvolvido para revitalização do controla das unidades refrigeradoras. No entanto falta alguns ajustes finais na programação para que esteja cem por cento funcional, como também a transferência da placa protótipo para a placa final de modo que não hajam problemas futuros quanto a robustez mecânica.

Com o intuito de desenvolver um projeto pronto a estabilidade. É aconselhável que seja desenvolvido um sistema supervisório para inserção do endereço de futuras unidades controladas a distância, de maneira simplificada.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] **DIGI**. XBee/XBee-PRO S1 802.15.4 (Legacy). Disponível em: <https://www.digi.com/resources/documentation/digidocs/pdfs/90000982.pdf>. Acessado em: 01/12/2017.

[2] **DIGI.** Xbee®/XBee-PRO®, 802.15.4 (Legacy), Professional Kit, Getting Started Guide.
Disponível em:<http://ftp1.digi.com/support/documentation/90002160_A.pdf>. Acessado em: 01/12/2017.

[3] **ATMEL**. 8-bit AVR Microcontrollers, ATmega328/P, DATASHEET COMPLETE. Disponível em: <http://www.atmel.com/Images/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_datasheet.pdf>. Acessado em: 01/12/2017.

[5] **ATMEL.** 2-Wire, Serial EEPROM, AT24C32. Disponível em: <http://www.atmel.com/Images/doc0336.pdf>. Acessado em: 01/12/2017.

[6] **ATMEL**. DS3231, Extremely Accurate I2C-Integrated, RTC/TCXO/Crystal. Disponível em: <https://datasheets.maximintegrated.com/en/ds/DS3231.pdf>. Acessado em: 01/12/2017.

[7] **KICAD**. A Cross Platform and Open Source Electronics Design Automation Suite. Disponível em: <http://kicad-pcb.org/>. Acessado em: 01/12/2017.

ANEXOS

Biblioteca Desenvolvida para a STAC:

```
#ifndef BLACKOUT-CONTROL_H
#define BLACKOUT-CONTROL_H

#include <XBee.h>
#include <SdFat.h>
#include <SoftwareSerial.h>
#include <avr/wdt.h>

class RTC {
private:
    char day;
    char month;
    char year;
    char hour;
    char minute;
    char second;
public:
    void set_date(char day, char month, char year);
    void set_time(char hour, char minute, char second);
    int get_date(void);
    int get_time(void);
};

class Comunication {
private:
    // Define by default the AT commands for digital ports.
    uint8_t atCmd[2] = { 'D','0' };
    // Define by default the digital port to output and low.
    uint8_t atValue[1] = { 0x4 };
    // Create a Remote AT response object
    XBeeAddress64 remoteAddress = XBeeAddress64(0x00, BROADCAST_ADDRESS);
    // Create a Remote AT request object;
    RemoteAtCommandRequest remoteAtRequest = RemoteAtCommandRequest(remoteAddr
ess, atCmd, atValue, sizeof(atValue));
    // Create a Remote AT response object
    RemoteAtCommandResponse remoteAtResponse = RemoteAtCommandResponse();
public:
    // Create a Xbee object.
    XBee xbee = XBee();
    void remoteRequest(XBeeAddress64 remoteAddress, uint8_t dPort, uint8_t dSt
ate);
    uint8_t executeRemote();
    uint8_t setAndQueryRemote();
};

class Update {
    //CHANGE int TO STRING
public:
    void set_turn_off(int address);
    void turn_on(int address, char hour);
};

class Hardware {
private:
    static const uint8_t pin_substation_relay = 2;
    static const uint8_t pin_phase_relay = 3;
    static const uint8_t pin_chipSelect = 10;
    static const uint8_t pin_xbeeSelect = 7;
    static uint32_t delayHysteresis;
    static uint32_t tick;
public:
```

```
60.     Hardware(void);
61.     void print(String data);
62.     void println(String data);
63.     static void reset();
64.     static void inputLvl(void);
65.     static void outputLvl(void);
66.     static void set_TRISn(void);
67.     static void set_xbeeSelect(void);
68.     static void set_delayHysteresis(uint32_t _delayHysteresis);
69.     static void set_tick(uint32_t _tick);
70.     static uint32_t get_delayHysteresis(void);
71.     static uint32_t get_tick(void);
72.     static uint8_t get_pin_phase_relay(void);
73.     static uint8_t get_pin_substation_relay(void);
74.     static uint8_t get_pin_chipSelect(void);
75.     static uint8_t get_pin_xbeeSelect(void);
76.     static uint8_t get_state_substation_relay(void);
77.     static uint8_t get_state_phase_relay(void);
78. };
79.
80. class Database : public Hardware {
81. private:
82.     static SdFat sd;            // SdFat object declaration.
83.     static SdFile myFile;       // SdFile object declaration.
84.     String fileName = "address.csv";
85.     uint32_t sh, sl;
86.     static uint8_t act_h_d0, act_min_d0, dea_h_d0, dea_min_d0, act_h_d1, act_m
    in_d1, dea_h_d1, dea_min_d1,
87.         act_h_d2, act_min_d2, dea_h_d2, dea_min_d2, act_h_d3, act_min_d3, dea_
    h_d3, dea_min_d3, checksum;
88. public:
89.     Database();
90.     static Comunication xb;     // Comunication object declaration. The Comuni
    cation object build the Xbee too.
91.     void add(uint32_t sh = 0x00, uint32_t sl = BROADCAST_ADDRESS,
92.         uint8_t act_h_d0 = 20, uint8_t act_min_d0 = 30, uint8_t dea_h_d0 = 6,
    uint8_t dea_min_d0 = 30, uint8_t act_h_d1 = 20, uint8_t act_min_d1 = 30, uint8
    _t dea_h_d1 = 6, uint8_t dea_min_d1 = 30,
93.         uint8_t act_h_d2 = 20, uint8_t act_min_d2 = 30, uint8_t dea_h_d2 = 6,
    uint8_t dea_min_d2 = 30, uint8_t act_h_d3 = 20, uint8_t act_min_d3 = 30, uint8
    _t dea_h_d3 = 6, uint8_t dea_min_d3 = 30);
94.     String getLine(uint8_t line);
95.     uint8_t countLine();
96.     void split(String buffer);
97. };
98.
99. class BlackoutControl {
100.       public:
101.           static Database db; //Database object declaration. The Comunication
     object build the SdFile, Sd and Xbee too.
102.           static RTC rtc;
103.           static Hardware hard; // Hardware object declaration.
104.           BlackoutControl();
105.           void verify_substation_relay(void);
106.           void verify_phase_relay(void);
107.           void turnAllOff(void);
108.           void turnAllIn(void);
109.           void turnOneByOne(void);
110.           void turnIn(void);
111.           void turnOut(void);
112.       };
113.       #endif
```

Código Principal da Biblioteca:

```
#include "Blackout-Control.h"

// Define SoftSerial TX/RX pins
// Connect Arduino pin 8 to TX of usb-serial device
#define ssRX 8
// Connect Arduino pin 9 to RX of usb-serial device
#define ssTX 9
// Remember to connect all devices to a common Ground: XBee, Arduino and USB-Serial device
static SoftwareSerial nss(ssRX, ssTX);


Comunication Database::xb;
SdFat Database::sd;
SdFile Database::myFile;
//uint32_t Database::sh;
//uint32_t Database::sl;
uint8_t Database::act_h_d0;
uint8_t Database::act_min_d0;
uint8_t Database::dea_h_d0;
uint8_t Database::dea_min_d0;
uint8_t Database::act_h_d1;
uint8_t Database::act_min_d1;
uint8_t Database::dea_h_d1;
uint8_t Database::dea_min_d1;
uint8_t Database::act_h_d2;
uint8_t Database::act_min_d2;
uint8_t Database::dea_h_d2;
uint8_t Database::dea_min_d2;
uint8_t Database::act_h_d3;
uint8_t Database::act_min_d3;
uint8_t Database::dea_h_d3;
uint8_t Database::dea_min_d3;
uint8_t Database::checksum;

Database::Database() {
    // Setting the Xbee.
    //xb.xbee.begin(Serial);

    if (!sd.begin(get_pin_chipSelect(), SPI_FULL_SPEED)) nss.println("SD don't init.");

    // Open or create the file.
    if (!myFile.open("ADDRESS.TXT", O_RDWR | O_CREAT | O_AT_END)) nss.println("Opening address.txt for create failed.");
    else nss.println("File was created!");

    // Close the file.
    myFile.close();
}

uint8_t Database::countLine() {
    static const int line_buffer_size = 69 + 1;
    static char buffer[line_buffer_size];
    ifstream sdin("address.txt");
    uint8_t count = 0;

    while (sdin.getline(buffer, line_buffer_size, '\n') || sdin.gcount()) {
        // Print line;
        //nss.println("Counter " + String(count) + ": " + buffer);

        // Increase counter.
        count++;
```

```
62.     }
63.
64.     // Close sdin object.
65.     sdin.close();
66.
67.     // Return line count.
68.     return count;
69. }
70.
71. String Database::getLine(uint8_t line) {
72.     static const int line_buffer_size = 69 + 1;
73.     static char buffer[line_buffer_size];
74.     ifstream sdin("address.txt");
75.     int count = 0;
76.
77.     while (sdin.getline(buffer, line_buffer_size, '\n') || sdin.gcount()) {
78.         if (count == line) {
79.             // Print line, close sdin object and break the while.
80.             nss.println("Got Line " + String(count) + ": " + buffer);
81.             sdin.close();
82.             break;
83.         }
84.
85.         // Increase counter.
86.         count++;
87.     }
88.
89.     // Return line.
90.     return buffer;
91. }
92.
93. void Database::split(String buffer) {
94.     static uint8_t fields[17];
95.     String aux = "";
96.     uint8_t nComma = 0;
97.
98.     nss.print("Splitted Line: ");
99.
100.          for (uint8_t i = 0; i < buffer.length(); i++) {
101.              // Concatenate char until comma.
102.              aux.concat(buffer.charAt(i));
103.
104.              // If comma or end array, comma plus plus, save the value and c
    lean the String aux.
105.              if (buffer.charAt(i) == ',' || nComma == 17) {
106.                  nComma++;
107.                  if (nComma == 1) {
108.                      sh = aux.toInt();
109.                      nss.print(sh);
110.                      aux = "";
111.                  }
112.                  else if (nComma == 2) {
113.                      sl = aux.toInt();
114.                      nss.print(sl);
115.                      aux = "";
116.                  }
117.                  else if (nComma > 2) {
118.                      fields[i + 3] = aux.toInt();
119.                      nss.print(fields[i + 3]);
120.                      aux = "";
121.                  }
122.                  if (buffer.charAt(i) == ',') {
123.                      nss.print(",");
124.                  }
125.              }
126.          }
127.
```

```
128.            nss.println("");
129.        }
130.
131.        void Database::add(uint32_t sh = 0x00, uint32_t sl = BROADCAST_ADDRESS,

132.            uint8_t act_h_d0 = 20, uint8_t act_min_d0 = 30, uint8_t dea_h_d0 =
    6, uint8_t dea_min_d0 = 30, uint8_t act_h_d1 = 20, uint8_t act_min_d1 = 30, ui
    nt8_t dea_h_d1 = 6, uint8_t dea_min_d1 = 30,
133.            uint8_t act_h_d2 = 20, uint8_t act_min_d2 = 30, uint8_t dea_h_d2 =
    6, uint8_t dea_min_d2 = 30, uint8_t act_h_d3 = 20, uint8_t act_min_d3 = 30, ui
    nt8_t dea_h_d3 = 6, uint8_t dea_min_d3 = 30) {
134.            // Open the file for write the address and schedules.
135.            if (!myFile.open("address.txt", O_WRITE | O_AT_END)) {
136.                nss.println("Opening address.txt for add address failed.");
137.            }
138.            else {
139.                // If the file is opened, add address to it.
140.                // Address, SH and SL.
141.                myFile.print(sh, DEC);
142.                myFile.print(",");
143.                myFile.print(sl, DEC);
144.                myFile.print(",");
145.
146.                // Feeding the time of activation and deactivation of D"port".

147.                myFile.print(act_h_d0, DEC);
148.                myFile.print(",");
149.                myFile.print(act_min_d0, DEC);
150.                myFile.print(",");
151.                myFile.print(dea_h_d0, DEC);
152.                myFile.print(",");
153.                myFile.print(dea_min_d0, DEC);
154.                myFile.print(",");
155.                myFile.print(act_h_d1, DEC);
156.                myFile.print(",");
157.                myFile.print(act_min_d1, DEC);
158.                myFile.print(",");
159.                myFile.print(dea_h_d1, DEC);
160.                myFile.print(",");
161.                myFile.print(dea_min_d1, DEC);
162.                myFile.print(",");
163.                myFile.print(act_h_d2, DEC);
164.                myFile.print(",");
165.                myFile.print(act_min_d2, DEC);
166.                myFile.print(",");
167.                myFile.print(dea_h_d2, DEC);
168.                myFile.print(",");
169.                myFile.print(dea_min_d2, DEC);
170.                myFile.print(",");
171.                myFile.print(act_h_d3, DEC);
172.                myFile.print(",");
173.                myFile.print(act_min_d3, DEC);
174.                myFile.print(",");
175.                myFile.print(dea_h_d3, DEC);
176.                myFile.print(",");
177.                myFile.print(dea_min_d3, DEC);
178.
179.                // Report added.
180.                nss.print("The address,  ");
181.                nss.print(sh, HEX);
182.                nss.print(sl, HEX);
183.                nss.println(", was added.");
184.            }
185.
186.            // By the way, close the file.
187.            myFile.close();
188.        }
```

```
189.
190.       void Comunication::remoteRequest(XBeeAddress64 remoteAddress, uint8_t d
    Port, uint8_t dState) {
191.           // Set a D"dPort" port.
192.           // Add 48 to dPort to pass to ASCII.
193.           atCmd[1] = { dPort + 48 };
194.
195.           // Set a D"i" value.
196.           atValue[0] = { dState };
197.
198.           // Now reuse same object for D"dPort" command.
199.           remoteAtRequest.setRemoteAddress64(remoteAddress);
200.           remoteAtRequest.setCommand(atCmd);
201.           remoteAtRequest.setCommandValue(atValue);
202.           remoteAtRequest.setCommandValueLength(sizeof(atValue));
203.
204.           int tryTimes = 0;
205.           while (executeRemote() == dState && tryTimes < 5) {
206.               executeRemote();
207.               tryTimes++;
208.           }
209.       }
210.
211.       uint8_t Comunication::executeRemote() {
212.           // Set and query to compare
213.           setAndQueryRemote();
214.
215.           // It's a good idea to clear the set value so that the object can b
    e reused for a query.
216.           remoteAtRequest.clearCommandValue();
217.
218.           return setAndQueryRemote();
219.       }
220.
221.       uint8_t Comunication::setAndQueryRemote() {
222.           // Send the command.
223.           xbee.send(remoteAtRequest);
224.
225.           // Wait up until to 5 seconds for the status response.
226.           // If got a response.
227.           if (xbee.readPacket(5000)) {
228.
229.               // Should be an AT command response.
230.               if (xbee.getResponse().getApiId() == REMOTE_AT_COMMAND_RESPONSE
    ) {
231.                   xbee.getResponse().getRemoteAtCommandResponse(remoteAtRespo
    nse);
232.
233.                   // If response is OK.
234.                   if (remoteAtResponse.isOk()) {
235.
236.                       // Get value of input D"i".
237.                       if (remoteAtResponse.getValueLength() > 0) {
238.                           for (int i = 0; i < remoteAtResponse.getValueLength
    (); i++) {
239.                               // Set transation with the value of response.
240.                               return remoteAtResponse.getValue()[i];
241.                           }
242.                       }
243.                   }
244.                   else {
245.                       // Set transation with not OK.
246.                       return false;
247.                   }
248.               }
249.               else {
250.                   // Set transation with not OK.
```

```
                    return false;
                }
            }
            else {
                // Set transation with not OK.
                return false;
            }
            // Set transation with OK.
            return true;
        }

        uint32_t Hardware::delayHysteresis = 0;

        uint32_t Hardware::tick = 0;

        Hardware::Hardware(void) {
            // Setting the boudrate of Software Serial.
            nss.begin(9600);

            //Enable Xbee.
            set_xbeeSelect();

            // Setting the Extern Interruptions.
            set_TRISn();
            inputLvl();
        }

        void Hardware::reset() {
            // Start watchdog with the provided prescaller.
            // Some times to wait before reset: WDTO_15MS, WDTO_30MS, WDTO_60MS
    , WDTO_120MS, WDTO_250MS, WDTO_500MS, WDTO_1S, WDTO_2S, WDTO_4S, WDTO_8S
            wdt_enable(WDTO_15MS);
            // Wait for the prescaller time to expire.
            // Without sending the reset signal by using.
            // The wdt_reset() method stop the wdt.
            while (true) {}
        }

        void Hardware::set_TRISn(void) {
            //Throut the ports to input.
            pinMode(get_pin_phase_relay(), INPUT);
            pinMode(get_pin_substation_relay(), INPUT);
            //Throut the ports to output.
        }

        void Hardware::inputLvl(void) {
            //Throut the ports to input.
            //Throut the resistor to pullup.
            pinMode(get_pin_phase_relay(), INPUT_PULLUP);
            pinMode(get_pin_substation_relay(), INPUT_PULLUP);
            //Throut the resistor to pulldown.
        }

        void Hardware::outputLvl(void) {
            //Throut the ports to output.
            pinMode(get_pin_xbeeSelect(), OUTPUT);
        }

        void Hardware::print(String data) {
            nss.print(data);
        }

        void Hardware::println(String data) {
            nss.println(data);
        }

        void Hardware::set_delayHysteresis(uint32_t _delayHysteresis) {
```

```
317.            delayHysteresis = _delayHysteresis;
318.        }
319.
320.        uint32_t Hardware::get_delayHysteresis(void) {
321.            return delayHysteresis;
322.        }
323.
324.        void Hardware::set_tick(uint32_t _tick) {
325.            tick = _tick;
326.        }
327.
328.        uint32_t Hardware::get_tick(void) {
329.            return tick;
330.        }
331.
332.        uint8_t Hardware::get_state_substation_relay(void) {
333.            return digitalRead(pin_substation_relay);
334.        }
335.
336.        uint8_t Hardware::get_state_phase_relay(void) {
337.            return digitalRead(pin_phase_relay);
338.        }
339.
340.        uint8_t Hardware::get_pin_substation_relay(void) {
341.            return pin_substation_relay;
342.        }
343.
344.        uint8_t Hardware::get_pin_phase_relay(void) {
345.            return pin_phase_relay;
346.        }
347.
348.        uint8_t Hardware::get_pin_chipSelect() {
349.            return pin_chipSelect;
350.        }
351.
352.        uint8_t Hardware::get_pin_xbeeSelect() {
353.            return pin_xbeeSelect;
354.        }
355.
356.        void Hardware::set_xbeeSelect(void) {
357.            digitalWrite(get_pin_xbeeSelect(), HIGH);
358.        }
359.
360.        void BlackoutControl::verify_substation_relay(void) {
361.            if (hard.get_state_substation_relay() && hard.get_state_phase_relay
    ()) {
362.                nss.print("System OK, generator is off.");
363.                turnAllIn();
364.            }
365.            else if (!hard.get_state_substation_relay()) {
366.                nss.print("Blackout, generator in.");
367.                turnAllOff();
368.            }
369.        }
370.
371.        void BlackoutControl::verify_phase_relay(void) {
372.            if (hard.get_state_phase_relay() && hard.get_state_substation_relay
    ()) {
373.                nss.print("System OK, all phases are in.");
374.                turnAllIn();
375.            }
376.            else if (!hard.get_state_phase_relay()) {
377.                nss.print("Blackout, missing some phase.");
378.                turnAllOff();
379.            }
380.        }
381.
```

```
382.        // Hardware object declaration.
383.        Hardware BlackoutControl::hard;
384.
385.        // Database object declaration.
386.        Database BlackoutControl::db;
387.
388.        BlackoutControl::BlackoutControl() {
389.
390.        }
391.
392.        void BlackoutControl::turnAllOff(void) {
393.            // Increment i and send in broadcast the value of digital ports to
    low.
394.            for (uint8_t i = 0; i < 4; i++) {
395.                db.xb.remoteRequest(XBeeAddress64(0x00, BROADCAST_ADDRESS), i,
    4);
396.                delay(400);
397.            }
398.            nss.println(" - ALL OFF");
399.        }
400.
401.        void BlackoutControl::turnAllIn(void) {
402.            // Increment i and send in broadcast the value of digital ports to
    high.
403.            for (uint8_t i = 0; i < 4; i++) {
404.                db.xb.remoteRequest(XBeeAddress64(0x00, BROADCAST_ADDRESS), i,
    5);
405.                delay(400);
406.            }
407.            nss.println(" - ALL IN");
408.        }
409.
410.        void BlackoutControl::turnOneByOne(void) {
411.
412.        }
413.
414.        void BlackoutControl::turnIn(void) {
415.            // Change the status to automatic.
416.        }
417.
418.        void BlackoutControl::turnOut(void) {
419.            // Change the status to automatic.
420.        }
```

Código Principal do Arduino:

```
1.  //Libraries
2.  #include "Blackout-Control.h"
3.
4.  #define TIMEBASE 60000
5.
6.  //BlackoutControl object declaration.
7.  static BlackoutControl blk;
8.
9.  void setup() {
10.     // Attach the Extern Interruptions.
11.     attachInterrupt(digitalPinToInterrupt(blk.hard.get_pin_substation_relay())
    , extIntSubstation, FALLING);
12.     attachInterrupt(digitalPinToInterrupt(blk.hard.get_pin_phase_relay()), ext
    IntPhase, FALLING);
13.
14.     // Wait the Xbee configurations.
15.     delay(5000);
16.
17.     // Setting the boudrate of Xbee.
18.     Serial.begin(9600);
19.
20.     dbInit();
21. }
22.
23. int count = 0;
24. void loop()
25. {
26.     // XBEE TESTs
27.     // -> BROADCAST TEST
28.     //if (count % 2 == 0) {
29.     //  blk.db.xb.remoteRequest(XBeeAddress64(0x00, BROADCAST_ADDRESS), 0, 4);

30.     //}
31.     //else {
32.     //  blk.db.xb.remoteRequest(XBeeAddress64(0x00, BROADCAST_ADDRESS), 0, 5);

33.     //}
34.     // -> UNICAST TEST
35.     if (count % 2 == 0) {
36.         blk.db.xb.remoteRequest(XBeeAddress64(0x0013A200, 0x4091572D), 0, 4);

37.     }
38.     else {
39.         blk.db.xb.remoteRequest(XBeeAddress64(0x0013A200, 0x4091572D), 0, 5);

40.     }
41.     delay(1000);
42.     count++;
43.
44.     //DATABASE TEST
45.     //uint8_t nLines = blk.db.countLine();
46.     //for (uint8_t i = 0; i < nLines; i++) {
47.     //  delay(1000);
48.     //  blk.db.split(blk.db.getLine(i));
49.     //}
50.
51.     // MAIN TEST
52.     //if ((millis() - blk.hard.get_tick() > TIMEBASE)) {
53.     //  // Verify sisnisters.
54.     //  blk.verify_phase_relay();
55.     //  blk.verify_substation_relay();
56.     //  // Reset tick;
57.     //  blk.hard.set_tick(millis());
58.     //}
```

```
59. }
60.
61. void dbInit(void) {
62.     blk.db = Database();
63.     blk.db.add(1286656, 1083266861);
64. }
65.
66. void extIntSubstation(void) {
67.     if ((millis() - blk.hard.get_delayHysteresis() > TIMEBASE) || (blk.hard.ge
    t_delayHysteresis() == 0)) {
68.         //Verify sisnister.
69.         blk.verify_substation_relay();
70.         // Reset delay;
71.         blk.hard.set_delayHysteresis(millis());
72.     }
73. }
74.
75. void extIntPhase(void) {
76.     if ((millis() - blk.hard.get_delayHysteresis() > TIMEBASE) || (blk.hard.ge
    t_delayHysteresis() == 0)) {
77.         //Verify sisnister.
78.         blk.verify_phase_relay();
79.         // Reset delay;
80.         blk.hard.set_delayHysteresis(millis());
81.     }
82. }
```

Blackout-Control

Xbee, Arduino, Point-to-Multpoint, 802.15.4

Mandatory

0 - É necessária um canal para a rede PAN, todos os dispositivos devem ter o mesmo valor, o CH utilizado nesse projeto foi:

> CH = C

1 - É necessária um ID para a rede PAN, todos os dispositivos devem ter o mesmo valor, o ID utilizado nesse relatório foi:

> ID = 2017

2 - É necessária um a verificação se há um coordinator na rede, para isso todos os dispositivos devem estar com o parâmetro A1 na seguinte configuração, de modo que se os ID e CH forem os mesmos haverá comunicação:

> A1 = 100

3 - É necessária um a verificação se há um coordinator na rede, para isso todos os dispositivos devem estar com o parâmetro A1 na seguinte configuração, de modo que se os ID e CH forem os mesmos haverá comunicação:

> A2 = 0100

4 - É necessária a configuração dos parâmetros CE e SM para determinação se Coordinator, Router ou End device, esses determinam se são do tipo coordinator e se entram em modo sleep, respectivamente. Nenhum outro dispositivo pode ser coordinator se não tiver CM = 1. Router não podem ser coordinators. Apenas end devices podem entrar em modo sleep. Coordinator:

> CM = 0

> SM = 0

End device:

> CM = 0

> SM = 0

5 - Com o objetivo de configurar uma rede API, deve ser habilitado o parâmetro AP:

> AP = 1

6 - É necessária a configuração dos endereços de envio, para uma rede do tipo API, utilizando apenas um coordinator o endereço dele deve ser fixo e único enquanto que os demais routers e end devices devem ser um só, assim garantindo o envio por broadcast. Porém cada dispositivo terá seu único parâmetro MY garantindo assim o unicast entre os dispositivos. Coordinator:

> DH = 0 , DL = FFFF , MY=0

End device:

> DH = 0 , DL = 0 , MY=1

7 - Com o objetivo de manter o sistema seguro, foi adicionada uma encriptação no protocolo, habilitando o parâmentro EE e repetindo a mesma senha em todos os dispositivos.

> EE = 1

> KY = 1FF2FF3FF

8 - Para ser produzido o broadcast, deve ser ser colocado como parâmetro de endereço 64-bit, os seguintes códigos:

> DH = 00 00 00 00

> DL = 00 00 FF FF

9 - Para ser produzido um envio do tipo unicast, deve ser colocado como parâmetro de endereço 64-bit, os seguintes códigos:

> DH = SH

> DL = SL

10 - Com o objetivo do controle de dispositivo dentro de uma rede a distância, devem ser enviados comandos AT, com o uso de um micro controlador e de um biblioteca que auxilie o envio de frames com as respectivas tecnologias.

11 - O envio de frames com a plataforma XCTU, primeiramente deve ser selecionado o tipo de frame 0x17 - Remote AT Command. Em seguida preenchido os endereços de envio em caso de unicast e broadcast ver os casos 8 e 9.

12 - O comando AT enviado deve ser preenchido apenas pela sigla e não pelo adjetivo AT. Uma sigla é D0 que representa o estado da porta digital 0. E seu parâmetro deve ser inserido em HEX caso trate-se de um número.

13 - Devido a facilidade oferecida foram utilizados resistores de pull up na entrada do Xbee, para isso deve ser modificado parâmetro PR:

> PR = 1

14 - A frequência de monitoramento da variação das entradas do Xbee é determinado pelo parâmetro IR. Foi adotado o valor mínimo para melhorar a precisão de atuação.

> IR = 0

15 - O monitoramento da variaveis de entrada do Xbee é determinado pelo parâmetro IC, que deve ser configurado de acordo com a porta que deseja monitorar. Essa configuração é do tipo interrupção de entrada que dispara o acionamento de um endereço especifico contido em IA.

16 - Para resetar o Xbee deve ser conectar o pino 5 ao pino 10 com firmeza por alguns segundos.

17 -

Firmware

Modem	Function Set	Version
XB24	802.15.4	10ef

Coordinator

Parameter	Value	Comments
CH (Channel)	0x0C	Identical
ID (PAN ID)	0x2017	Identical
DH (Second. Address)	0x0	Identical
DL (Second. Address)	0x0	Identical
MY (Source Address)	0x0	Unique
CE (Coord. Enable)	1	Unique
A2 (Coord. Assoc.)	0x04	Unique
NI (Name)	COORD_STAC	Unique
AP	1	Identical
EE	1	Identical
KY	1FF2FF3FF	Identical

End Device

Parameter	Value	Comments

Parameter	Value	Comments
CH (Channel)	0x0C	Identical
ID (PAN ID)	0x2017	Identical
DH (Second. Address)	0x0	Identical
DL (Second. Address)	0x0	Identical
MY (Source Address)	0x01	Unique
CE (Coord. Enable)	0	Identical to end devices
A1 (End Dev Assoc.)	0x04	Identical to end devices
NI (Name)	END_STAC_XX	Unique
AP (Mode)	1	Identical
EE (Secure?)	1	Identical
KY (Key)	1FF2FF3FF	Identical

Network

> Broadcast

> Unicast

Addresses:

> Coordinator = 0013A20040915718

> Router_0001 = 0013A2004091572D

> Router_0002 = 0013A2004091572D

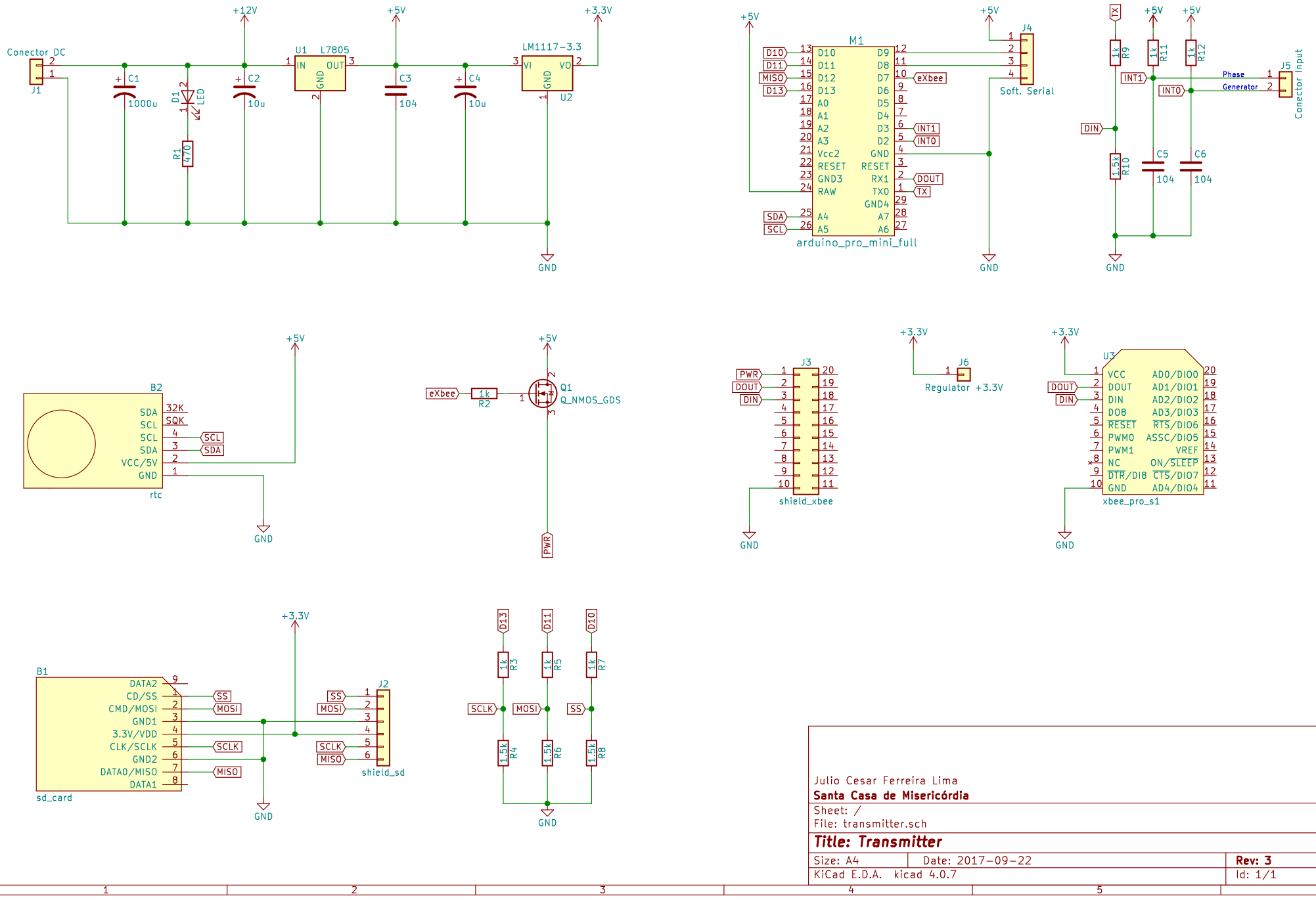
> Router_0003 = 0013A2004091572D

> Router_0004 = 0013A2004091572D

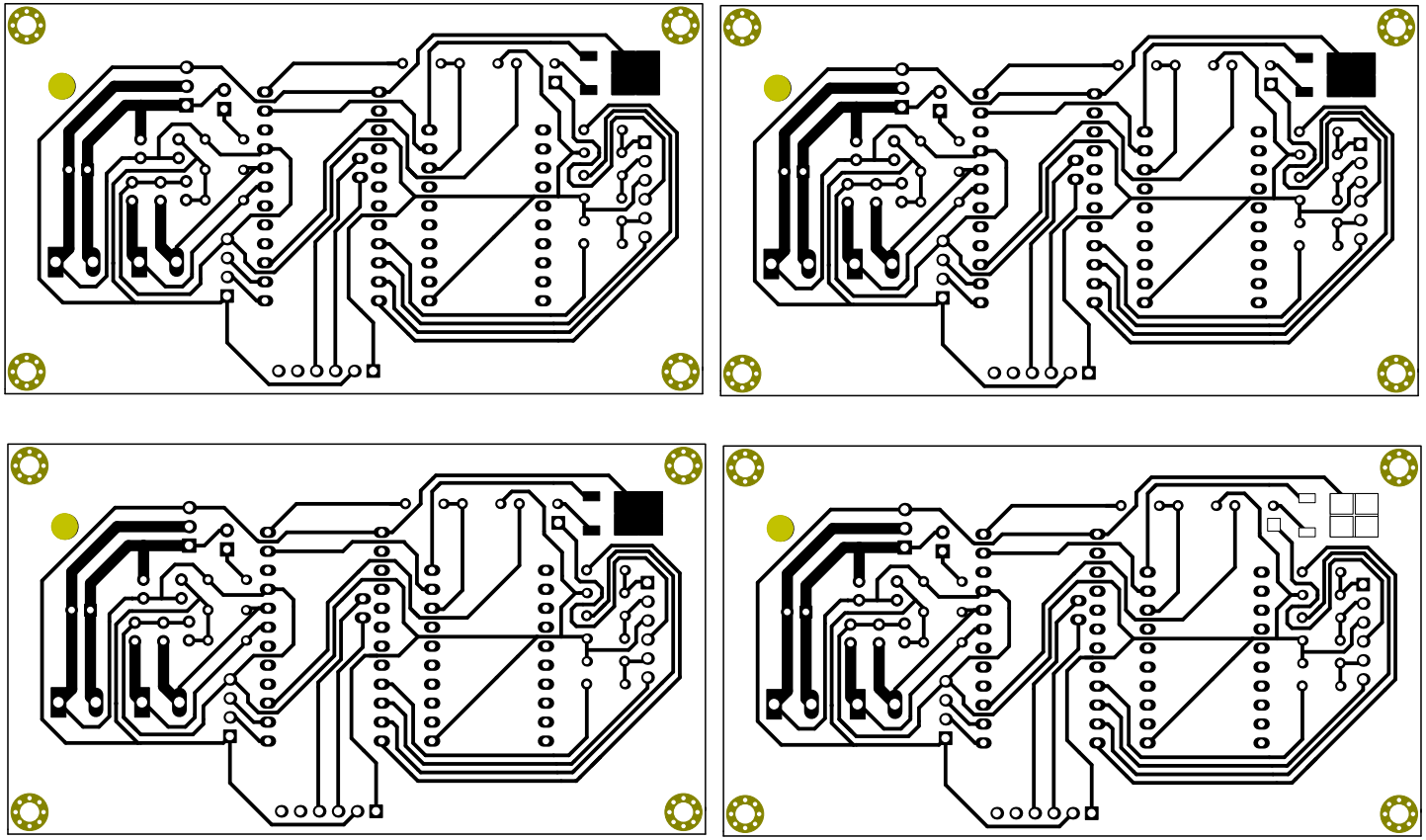
> Router_0005 = 0013A2004091572D

> Router_0006 = 0013A2004091572D

> Router_0007 = 0013A2004091572D

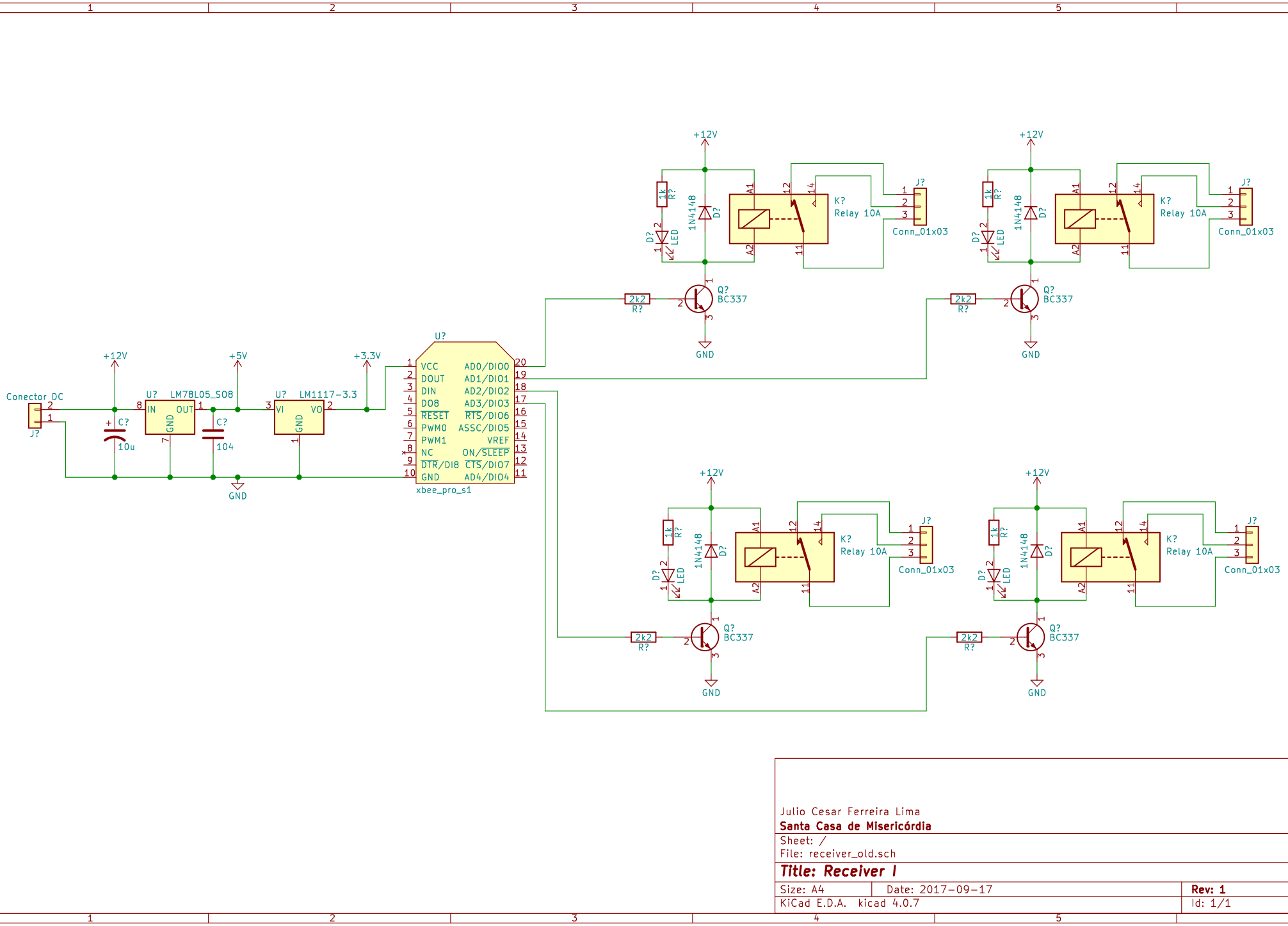


Julio Cesar Ferreira Lima	
Santa Casa de Misericórdia	
Sheet: /	
File: transmitter.sch	
Title: Transmitter	
Size: A4	Date: 2017-09-22
KiCad E.D.A. kicad 4.0.7	Id: 1/1
	Rev: 3



Julio Cesar Ferreira Lima
Santa Casa de Misericórdia

Sheet:		
File: transmitter.kicad_pcb		
Title: Transmitter		
Size: A4	Date: 2017-11-20	**Rev: 1.0**
KiCad E.D.A. kicad 4.0.7		Id: 1/1



Julio Cesar Ferreira Lima	
Santa Casa de Misericórdia	
Sheet: /	
File: receiver_old.sch	
Title: Receiver I	
Size: A4	Date: 2017-09-17
KiCad E.D.A. kicad 4.0.7	Id: 1/1
	Rev: 1