INSTITUTO
FEDERAL
Santa Catarina

Câmpus
São José

Documentação

Aplicação do Hardware para placa de acionamento

Curso: Engenharia de Telecomunicações
Disciplina: Bolsa de iniciação científica
Professor: Marcos Moecke

Aluno

Jamilly da Silva Pinheiro

21 de dezembro de 2023

Contents

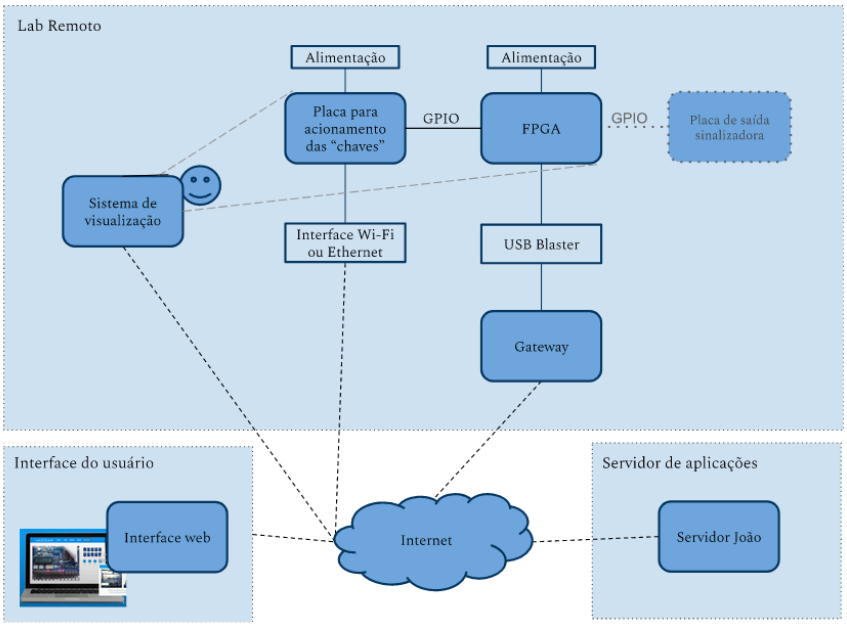
Pesquisa para aplicação do Hardware

Neste estágio inicial da bolsa, é importante considerarmos os elementos essenciais que serão necessários, sem entrar em detalhes específicos.

Serei responsável pela execução da parte física do projeto, ou seja, do hardware. Isso envolverá o desenvolvimento de uma placa auxiliar que permitirá o acionamento remoto das chaves e estará conectada à FPGA.

Na Figura 1, é possível observar o diagrama de bloco do kit contendo as ideias iniciais de forma visual.

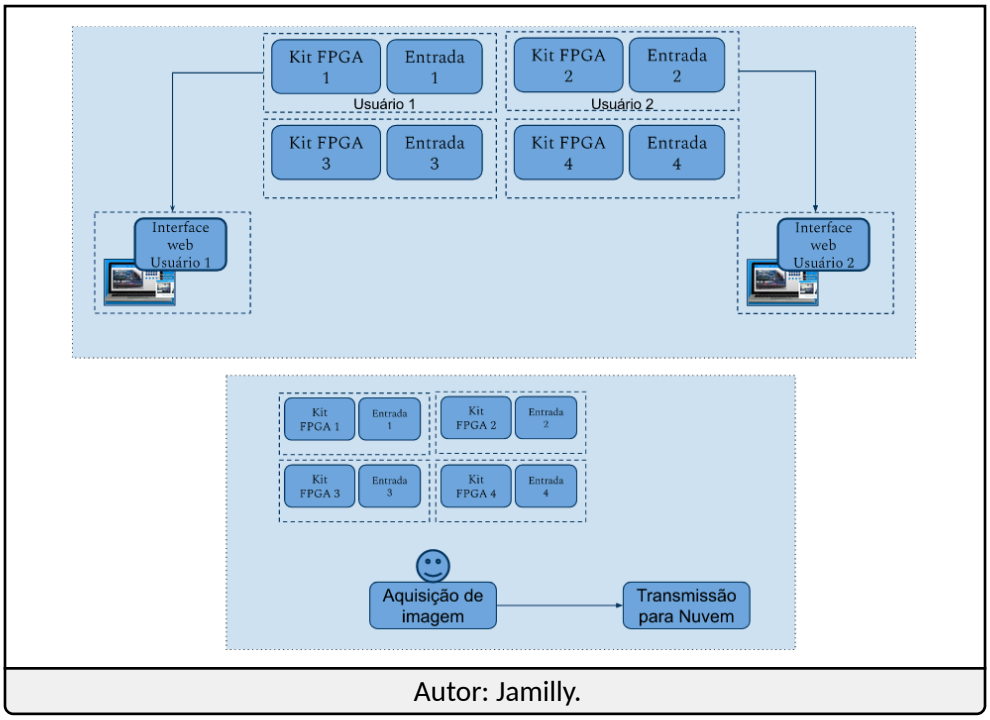
Diagrama de bloco do kit.



Autor: Jamilly.

Na Figura 2, apresentamos um diagrama de blocos abrangente que ilustra o processo de transmissão de imagens dos kits. A câmera irá capturar todos os kits disponíveis, e essa transmissão será integrada a um sistema que permitirá aos usuários visualizar e interagir, apenas com kit selecionado, juntamente com a placa de controle de entrada correspondente.

Diagrama de blocos geral



Autor: Jamilly.

No Quadro 1, são ressaltados os elementos importantes, essenciais e desejáveis que planejamos incorporar ao projeto.

Categoria da especificação	**Descrição**	**Acompanhamento do estado no projeto**
Essencial	Acionar remotamente as “chaves”, independente da programação do kit FPGA, Representar visualmente os estado lógico ‘0’ e ‘1”, usando leds Transmitir a imagem captada com qualidade suficiente para visualização das mudanças em tempo real A câmera captura todos os kits disponíveis, permitindo que o usuário selecione um kit específico para visualização, exibindo apenas o kit escolhido	

Importante	A placa auxiliar seja adaptável a uma boa variedade de kits de FPGAs., Representar visualmente os estado lógico ‘Z’. Representar visualmente os estado lógico sinal pulsante. Permitir pressionar mais de um botão momentaneamente	
Desejável	Simular o repique das chaves mecânicas, especificando o tempo de repique., Placa de saída que permita mostrar, leds, displays 7 segmentos, etc. Possuir uma imagem de qualidade suficiente para acompanhar em tempo real processos que são controlados pelo kit FPGA. Ex. motor posicionando. Indicar se a chave foi programada para ser utilizada na interface do usuário (colocar em cor cinza quando não for utilizada). Interação dos botões na interface do usuário. Mostrar o repique da chave para o usuário . A possibilidade de agendar a reserva do kit para um horário específico.	

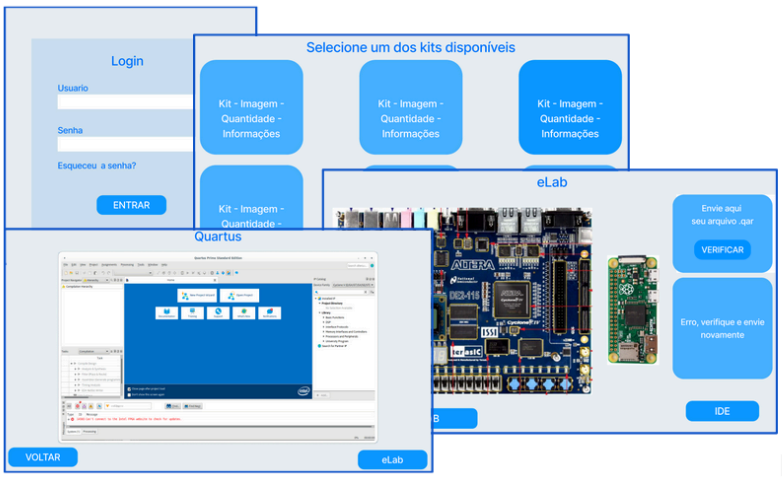
	Ter um sistema de gravação.	

A seguir, apresentamos alguns diagramas que representam a concepção inicial da interface web. Esta interface engloba as seguintes áreas:

- Tela de Login: Onde os usuários acessam a plataforma com suas credenciais;
- Seleção de Área de Trabalho: Nesta etapa, os usuários escolhem entre as áreas de “Quartus” ou “Laboratório Remoto” para iniciar suas atividades;
- Interface IDE de programação/simulação: Aqui, os usuários podem programar/simular seus projetos;
- Seleção de Kits: Os usuários têm a opção de escolher os kits de desenvolvimento com os quais desejam trabalhar;
- Agendamento de Kit: Permite aos usuários agendar o uso de kits específicos, facilitando a organização de recursos;
- Laboratório Remoto: Uma área onde os usuários podem acessar e realizar experimentos remotamente.

Para termos uma base de como ficará a interface do usuário, foi realizado um fluxo das tela através do Figma, que é um editor gráfico de vetor e prototipagem de projetos de design baseado principalmente no navegador web.

Diagrama geral da interface Web



Autor: Jamilly.

Neste link está a imagem das telas: https://drive.google.com/file/d/1ZBQa0sLrA-9ZE9UXbB6yWK92Ne_Z3bOt/view?usp=sharing

Neste link está o projeto do Figma: https://drive.google.com/file/d/1P5hyd5Po8cFqvZGeENyS5J5ulebRUe_E/view?usp=sharing

Observação: O projeto deve ser baixado e feito o Upload no Figma para poder ser editado e visualizado o fluxo das telas.

Link para acessar o Figma: https://www.figma.com/?fuid=

Placa de Acionamento

Para projetar a placa de acionamento, é essencial levar em consideração diversos aspectos. A fim de facilitar essa análise, foi levantado pontos em relação a cada placa, destacando os principais pontos a serem considerados.

Algumas placas requerem o uso de cartões SD, o que pode não ser a melhor abordagem devido ao risco de corrupção ou falha do cartão SD. Além disso, essas placas executam um sistema operacional que, por vezes, não pode ser facilmente replicado, e pode surgir complexidades ao tentar modificar o sistema para torná-lo compatível.

Existem placas que vêm equipadas com interfaces de câmera, ou podem adicionar uma câmera através do GPIO, mas acredito que não seja aconselhável combinar a placa de acionamento com a função da câmera. É crucial que a placa de acionamento desempenhe uma função específica, que é simular o acionamento das chaves por meio do GPIO.

Um ponto crucial a ser destacado é a seleção adequada entre a conexão Wi-Fi ou Ethernet, garantindo que a escolha não comprometa o funcionamento adequado de várias placas em um mesmo ambiente.

Quadro comparativo das placas

Características	**Arduino MEGA**	**RaspBerry Pi Pico**	**RaspBerry Pi Pico W**	**RaspBerry Pi Zero W**	**Esp32**	**RaspBerry Pi 4**
URL da placa	Datasheet Arduino MEGA	Datasheet Raspberry pi Pico	Datasheet Raspberry Pi Pico W	Datasheet Raspberry Pi zero W	Datasheet Esp32	Datasheet raspberry pi 4
Preço	A partir de RS 115,48	A partir de RS 47,40	A partir de RS 79,00	A partir de RS 149,90	A partir de RS 69,00	A partir de RS 579,40
Acesso Internet	Wi-Fi ou Ethernet (módulo externo)	Módulo externo	Wi-Fi e Bluetooth	Wi-Fi e Bluetooth	Wi-Fi e Bluetooth	Ethernet
Conector USB	Tipo B	Micro-B	Micro-B	micro USB	Micro-B	USB-C
Portas GPIO/ disponíveis	40 / 38	40 / 26	40 / 26	40 / 26	36 / 25	40 / 26
Cabos necessários em uso contínuo	Alimentação P2, RJ45	Alimentação USB micro-B	Alimentação USB micro-B	Alimentação USB micro-B	Alimentação USB micro-B	Alimentação USB micro-B, RJ45
Dimensão	108 x 53 x 12 mm	21 x 51.3 x 3.9 mm	21 x 51.3 x 3.9 mm	65mm x 30mm	27.5 x 51 x 7 mm	85 x 56 x 17mm

Tempo de inicialização	0 sec	0 sec	0 sec	0 sec	0 sec	2 min
Tem SO	Arduino	Boot Loader	Boot Loader	Linux (Cartão SD)	Boot Loader	Linux (Cartão SD)
Programação	C ou C++ ou .INO	Python, C e C++	Python, C e C++	Python	Python, C e C++	Python

Nota:

1) Se a placa for conectada via WIFI, seria importante usar um AP para conectar a rede da instituição via WIFI ou Ethernet.

2) Se a alimentação for diferente de 3,3 V, poderá ser necessário uma adaptação para a GPIO do kit FPGA.

3) O tempo de inicialização pode ser importante para desligar alguns módulos do eLab, quando não estão em uso.

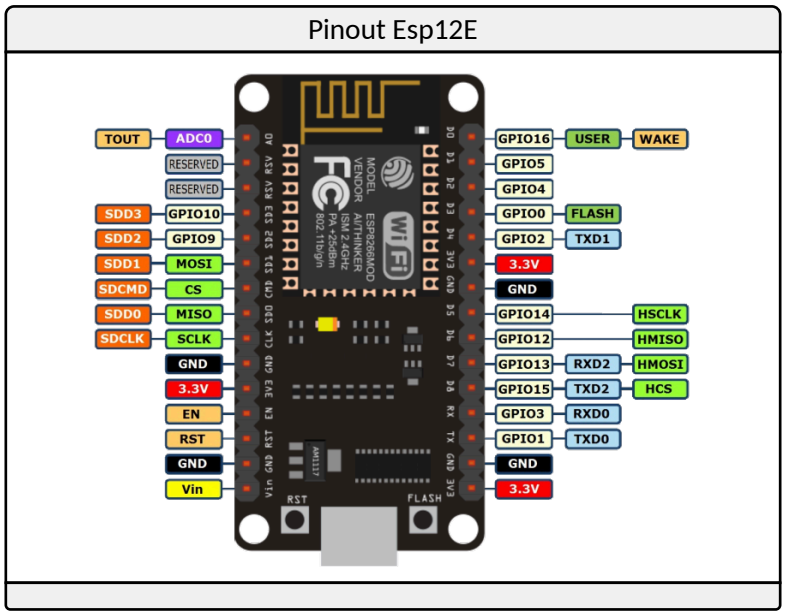
Características do projeto da placa

Características	**Essencial**	**Importante**	**Desejável**
Acesso Internet	no mínimo WIFI ou ethernet		
Taxa de transmissão	BAIXA		ALTA
Portas Digitais	16 portas		
Memória RAM			
Memória de dados			
Porta Analógica	Não	sim	sim
Alimentação	3,3V		
Interface para câmera de vídeo	não	não	sim
Custo	< RS 100,00		
Dimensão da placa	< 11 cm x 10 cm	<< FPGAs	<< FPGAs
Cabos de conexão	Mínimo possível (alimentação, cabo flat com kit FPGA)		
Tempo de inicialização	< 1 min		

Placas filha (módulos adicionais)	Mínimo possível		

Programação da Placa

Temos disponível a placa RaspBerry Pi Zero W e a Esp12E. Vamos iniciar o projeto da placa acionadora com Esp12E, a placa possui 13 pinos GPIO, porém apenas 11 são utilizáveis, que são os GPIOs 0, 2, 3, 4, 5, 10, 12, 13, 14, 15 e 16.



A programação da placa é feita pela PlatformIO, é utilizado o protocolo MQTT, para tratar as mensagens recebidas pelo servidor.

A biblioteca que utilizará o MQTT e fará a conexão da placa na rede Wi-fi, foi reaproveitada de um trabalho do João, apenas com as adaptações necessárias para a Esp12E. É realizado apenas a implementação da forma como vamos receber os dados do servidor e tratar de forma que o servidor e a placa "conversem" entre si, e a placa consiga identificar os comandos.

A implementação da biblioteca Elab, é onde vai ser tratado o que é recebido do servidor e a identificação dos pinos da placa.

Biblioteca Elab

- **Método `<string> separa(std::string &linha, char separador);`:**

◦ Este método usa um separador para separar uma string. Que são os dados enviados do servidor para a placa;

◦ Definimos que a placa irá receber os dados da seguinte forma: "C1:C2,C1:C2,CN:CN";

◦ O caractere 1 indica o pino da placa , já caractere 2 indica se está ligado, desligado ou em 3-state (quando o pino não foi configurado para ser usado);

◦ O número do caractere 1 representa um pino mapeado na placa. Quanto ao caractere 2, definimos que 0 é desligado, 1 ligado e 3 representa a condição de alta impedância.

- **Método `(ELab::getRealPin(int pin))`:**

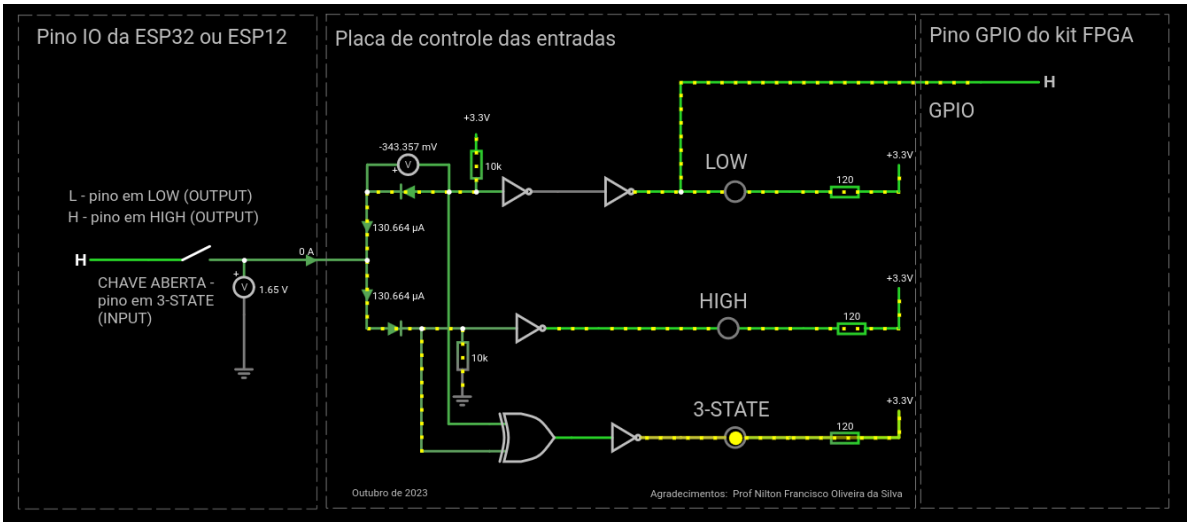
◦ Neste método é realizado o mapeamento para o pino real; ◦ O mapeamento depende do pinout de cada placa;

◦ O mapeamento da Esp12E é definido da seguinte forma

0 -> 10
1 -> 16
2 -> 5
3 -> 4
4 -> 0
5 -> 2
6 -> 14
7 -> 12
8 -> 13
9 -> 15
10 -> 3
Outros -> −1

Anteriormente, estabelecemos a placa Esp12E como nossa placa de controle, conectando seus pinos GPIO diretamente aos pinos GPIO da placa FPGA. No entanto, como um dos requisitos fundamentais do projeto é sinalizar o estado do botão, que pode ser nível alto, baixo ou 3-state, através de LEDs, foi elaborado o seguinte circuito para cada porta de saída:

Circuito integrado para acionamento dos Leds



Fonte: Marcos Moecke e Nilton Francisco Oliveira da Silva

Clique aqui parar acessar o circuito

Para essa finalidade, utilizou-se os circuitos integrados MC74HC86N, que contêm portas XOR, e o CI SN74HC04N, que apresenta portas NOT. Cada porta exige a implementação deste circuito. Diante disso, decidimos adotar um Dispositivo Programável de Lógica Complexa (CPLD), como intermediário, para gerenciar os três estados e enviar os sinais para a FPGA.

Dessa forma, a Esp12E enviará os dados nas seguinte portas do CPLD:

- Estabelecido 4 portas S (0,1,2,3), que indicarão os pinos a serem controlados, permitindo a representação de até 16 pinos;
- Uma porta para indicar o dado, que será 0 para nível baixo e 1 para alto;

- Uma porta para indicar o estado 3-state;
- Uma para para enviar o sinal de enable, que aplicará as configurações.

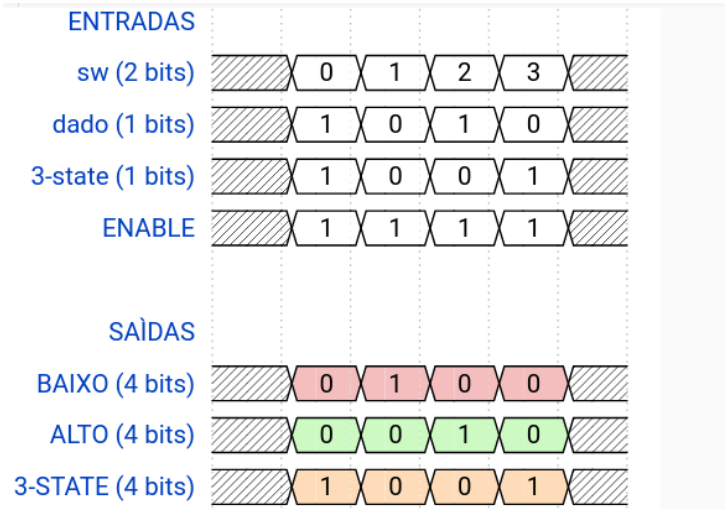
No total, serão 7 portas que enviarão o sinal para o CPLD. Na saída do CLPD para a FPGA terá a representação do estado de cada pino através dos LEDs, o qual não será mais necessário o circuito estabelecido anteriormente, pois o próprio CPLD fará o tratamento dos três estados.

A Esp12E será responsável por controlar o CPLD, por sua vez, o CPLD controlará a FPGA. Essa mudança resultou em modificações no arquivo de implementação da biblioteca Elab. Além disso, foi necessário realizar a implementação do CPLD em linguagem VHDL.

- **Método `(int decimal, int bitAmount)`:**
  - Este método trata a representação dos pinos de decimal para binário.
- **Método `(int pin, int command)`:**
  - Converte o número de decimal para binário;
  - Com o comando fornecido (0, 1 ou 2), configura-se os pinos de saída da Esp12E para o CPLD;
  - Aplico o comando fornecido command ao determinado pino passado por pin, dependendo do comando o pin ficará em nível alto, baixo ou 3-state;
  - Habilito o enable ao final do método.

O CPLD que controlará a FPGA terá a seguinte lógica

Diagrama de tempo



Fonte: Marcos Moecke

Link para acesso ao diagrama: https://wavedrom.com/editor.html

Ele receberá como entrada os valores de S (0, 1, 2, 3), dado, 3-state e enable. Conforme os valores de entrada estabelecidos, as saídas serão determinadas, e a saída do pino específica estará em um dos seguintes estados: baixo, alto ou 3-state.

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
```

```
entity placa is
  generic (
    N : natural := 3;
    M : natural := 3;
    nSel : natural := 1
  );
  port(
  clk : in std_logic;
  rst : in std_logic;
  sel : in std_logic_vector(nSel downto 0);
  dado : in std_logic;
  tri_state : in std_logic;
  gpio   : out std_logic_vector(N downto 0);
  red, green, yellow  : out std_logic_vector(M downto 0)
  );
end entity;

architecture ifsc_v1 of placa is
begin

  PROCESS (clk,rst)
  begin
    if rst = '1' then
      red <= (others => '0');
      green <= (others => '0');
      yellow <= (others => '1');
      gpio <= (others => 'Z');
    elsif rising_edge(clk) then
      if tri_state = '1' then
        yellow <= (others => '1');
        gpio <=  (others => 'Z');
      else
        red <= (others => dado);
        green <= (others => not dado);
        yellow <= (others => '0');
        gpio <= (others => dado);
      end if;
    end if;
  end process;
end architecture;

architecture ifsc_v2 of placa is
  signal sel_int : integer range 0 to 3;
  signal ena_cs : std_logic_vector(3 downto 0);
begin
  sel_int <= to_integer(unsigned(sel));
  with sel_int select
  ena_cs <= "0001" when 0,
        "0010" when 1,
        "0100" when 2,
        "1000" when others;

  process (clk,rst)
  begin
    if rst = '1' then
      red <= (others => '0');
      green <= (others => '0');
      yellow <= (others => '1');
      gpio <= (others => 'Z');
    elsif rising_edge(clk) then
      for k in 3 downto 0 loop
        if (ena_cs(k) = '1') then

```

```
68            if tri_state = '1' then
69              yellow(k) <= '1';
70              gpio(k) <=   'Z';
71            else
72              red(k) <= not dado;
73              green(k) <= dado;
74              yellow(k) <= '0';
75              gpio(k) <=  dado;
76            end if;
77          end if;
78        end loop;
79      end if;
80    end process;
81  end architecture;
```

Biblioteca ElabWifi

- **Método `::ELabWifi(const char* ssid, const char* pass):`**
    - Este método é o construtor da classe, ele é chamado quando um objeto é criado.
- **Método `LabWifi::~ELabWifi():`**
    - Este método libera a memória alocada para os parâmetros do ElabWifi.
- **Método `void ELabWifi::connect():`**
    - Este método se conecta na rede Wifi.
- **Método `void ELabWifi::disconnect():`**
    - Este método desconecta da rede Wifi.
- **Método `bool ELabWifi::status():`**
    - Este método retorna o status da conexão Wifi.
- **Método `const char* ELabWifi::getSSID():`**
    - Este método retorna o SSID da rede Wifi.
- **Método `const char* ELabWifi::getPass():`**
    - Este método retorna a senha da rede Wifi.
- **Método `String ELabWifi::localIP():`**
    - Este método retorna o IP local da rede Wifi.

Ele está, de forma genérica. Para a realização dos testes, utilizamos a placa Cyclone V GX Starter Kit, uma vez que ainda não possuímos o CPLD. Realizamos o mapeamento dos botões físicos para a execução dos testes. Os botões utilizados estão definidos em um componente, também criado no código VHDL.

Pontos a serem descartados:

- Como ainda não possuímos o CPLD não foi possível montar o kit FPGA completo, com a Esp12E enviando sinais, o CPLD tratando e a FPGA executando o comando recebido;
- O cabo flat entre o CPLD e a FPGA poderá sempre ser reaproveitado, pois a configuração dos pinos GPIO das FPGAs são as mesmas;
- O cabo flat entre a placa controladora e o CPLD pode ser alterado, pois, a placa controladora pode ser modificada.

Biblioteca ElabMqtt

- **Método `ELabMqtt(const char* server, int port, const char* user, const char* pass) : mqttClient(wifiClient):`**

    ◦ Método que recebe como parâmetro o endereço do servidor MQTT, a porta, o usuário e a senha.

- **Método `ELabMqtt::~ELabMqtt():`**

    ◦ Método que é o destrutor da classe ELabMqtt.

- **Método `bool ELabMqtt::connect():`**

    ◦ Método que conecta o Esp12E ao servidor MQTT.

- **Método `const int ELabMqtt::connectError():`**

    ◦ Este método retorna o estado da conexão.

- **Método `void ELabMqtt::sendMessage(char topic[], char message[]):`**

    ◦ Este método que envia a mensagem para o servidor, recebe como parâmetro o tópico e a mensagem.

- **Método `void ELabMqtt::receiveMessage(char topic[], void(*callback)(int)):`**

    ◦ Este método recebe como parâmetro o tópico e a função de callback.

- **Método `void ELabMqtt::poll():`**

    ◦ Este garante a conexão com o servidor MQTT.

- **Método `const char* ELabMqtt::getUser():`**

    ◦ Este método retorna o brokerUser, que é o usuário do servidor MQTT.

- **Método `const char* ELabMqtt::getPass():`**

    ◦ Este método retorna o brokerPass, que é a senha do servidor MQTT.

- **Método `const char* ELabMqtt::getServer():`**

    ◦ Este método retorna o brokerServer, que é o endereço do servidor MQTT.

- **Método `int ELabMqtt::available():`**

    ◦ Este método verifica se ainda possui caracteres a serem recebidos.

- **Método `int ELabMqtt::read():`**

    ◦ Este método lê os caracteres recebidos.

- **Método `String ELabMqtt::messageTopic():`**

    ◦ Este método retorna o tópico da mensagem recebida.

- **Método `ELabMqtt::onMessage(void(*callback)(int)):`**

    ◦ Este método ele chama a função de callback quando a mensagem é recebida.

- **Método `string ELabMqtt::getMessage():`**

    ◦ Este método retorna a mensagem recebida.

Rotina principal main

```
#include <Arduino.h>
#include <string.h>
#include "ELabMqtt.h"
#include "ELabWifi.h"
#include "ELab.h"
#include "list"

// Variáveis de inicialização

// Declaracao do servidor e da sua porta
char brokerServer[] = BROKER_SERVER;
int brokerPort = BROKER_PORT;

// Declaracao do usuario e senha do broker
char brokerUser[] = BROKER_USER;
char brokerPass[] = BROKER_PASS;

// Declaracao do ID do dispositivo
char ID[] = DEVICE_ID;

// Declaração do nome e senha da rede WiFi que o dispositivo irá se conectar
char SSID[] = WIFI_SSID;
char PASS[] = WIFI_PASS;

// Declaração do nome do dispositivo
char name[] = DEVICE_NAME;

// Tópico
char topic[16] = "eLab/";
char subTopic[16] = "xxxxxxxx";

// Mensagem de registro
char registerJsonMessage[32];

// Criando o objeto mqtt
ELabMqtt mqtt(brokerServer, brokerPort, brokerUser, brokerPass);

// Criando o objeto wifi
ELabWifi wifi(SSID, PASS);

// Indicador de registro no servidor
bool registered = false;

// Chamando as bibliotecas implementadas
ELab elab;
list<string> listaPinosComandos;
list<string> listaPinoComando;

// Função que será chamada quando uma mensagem for recebida
void onMqttMessage(int messageSize) {
  // we received a message, print out the topic and contents
  Serial.println("Received a message with topic '");
  Serial.print(mqtt.messageTopic());
  Serial.print("', length ");
  Serial.print(messageSize);
  Serial.println(" bytes:");

  // use the Stream interface to print the contents
```

```
   string mensagem_recebida;
   while (mqtt.available()) {
      mensagem_recebida += (char)mqtt.read();
   }

   if (mensagem_recebida == "ACK") {
     Serial.println("ACK recebido");
     registered = true;
     return;
   } else if (mensagem_recebida == registerJsonMessage) {
     return;
   }
   listaPinosComandos = elab.separa(mensagem_recebida, ',');

   for (auto pinoComando : listaPinosComandos) {
     listaPinoComando = elab.separa(pinoComando, ':');

     Serial.print("Valor: ");
     Serial.println(pinoComando.c_str());

     std::list<string>::iterator it = listaPinoComando.begin();
     int pino = std::stoi(*it);
     std::advance(it, 1);
     int comando = std::stoi(*it);
     int pinoReal = elab.getRealPin(pino);

     Serial.print("Pino (recebido): ");
     Serial.println(pino);
     Serial.print("Pino (traduzido): ");
     Serial.println(pinoReal);
     Serial.print("Comando: ");
     Serial.println(comando);

     elab.setPins(pino, comando);

     Serial.println("\n");

   }
}
void setup() {
   //Abrimos uma comunicacao serial para imprimir dados no Monitor Serial
   Serial.begin(115200);

   //definindo todos os pinos como entrada
   int pinoReal = 0;
   for (int pino = 0; pino < 11; pino++) {
       //traduzindo o pino para o pino real
       pinoReal = elab.getRealPin(pino);
       //definindo o pino como entrada
       pinMode(pinoReal, INPUT);
   }

   delay(1000);

   memcpy(subTopic, ID, 8);
   strcat(topic, subTopic);
   Serial.print("Tópico: ");
   Serial.println(topic);

```

```
  // Conecting to WiFi
  Serial.print("Conectando na rede WiFi");
  wifi.connect();
  while (! wifi.status()){
      Serial.print(".");
      delay(1000);
  }
  Serial.println(WiFi.localIP());
  // Connecting to MQTT server
  if (mqtt.connect()) {
      Serial.println("Conectado no broker MQTT com sucesso!");
  } else {
      Serial.print("Erro ao conectar-se no broker! Erro = ");
      Serial.println(mqtt.connectError());
  }
  // Initial message to register on the server
  sprintf(registerJsonMessage, "{\"name\":\"%s\",\"id\":\"%s\"}", name, ID);
  mqtt.sendMessage(topic, registerJsonMessage);
  mqtt.receiveMessage(topic, onMqttMessage);
  int waitCount = 0;
  while (! registered) {
    mqtt.poll();
    if (waitCount > 10) {
        Serial.println("Tentando novamente registrar-se...");
        waitCount = 0;
        mqtt.sendMessage(topic, registerJsonMessage);
    } else {
        Serial.println("Aguardando ACK");
        waitCount++;
    }
    Serial.println("Aguardando registro no servidor");
    delay(1000);
  }

}
void loop() {
  mqtt.poll();
  delay(100);
}
```

Link para acesso ao código da Placa Controladora: https://github.com/eLab-FPGA/Placa-Controladora
Link para acesso ao código do CPLD: https://github.com/eLab-FPGA/CPLD