

Universidade Federal do ABC

Relatório do projeto

Dominó acionado pela Alexa

Docente:

Sandro Vatanabe

Discentes:

Gustavo Duarte Serafim

Matheus Ribeiro Barison Martins Silva

Pedro Henrique Pereira

15 de Dezembro de 2019

Universidade Federal do ABC

Relatório do projeto

Dominó acionado pela Alexa

Relatório para Avaliação na Disciplina de
Fundamentos de Robótica da Universidade
Federal do ABC.

Docente:

Sandro Vatanabe

Discentes:

Gustavo Duarte Serafim - 11054016

Matheus Ribeiro Barison Martins Silva - 11201721624

Pedro Henrique Pereira - 11109016

15 de Dezembro de 2019

Sumário

1	Introdução	2
2	Objetivos	3
3	Fluxograma da lógica de programação	4
4	Metodologia	7
4.1	Especificações do projeto	7
4.2	Configuração do Sinric e da Alexa	9
4.3	Programação do ESP8266	11
4.4	Configuração do Módulo IO	16
4.5	Modificação no programa do dominó	18
5	Resultados	20
5.1	Análise	20
5.2	Dificuldades	20
6	Conclusão	22
	Referências	23

1 Introdução

Alexa é uma assistente virtual inteligente desenvolvida pela Amazon. Os aparelhos Amazon Echo e Echo Dot são capazes de se conectar ao serviço e interagir com o usuário via comando de voz. A Alexa é capaz de fazer diversos serviços, como tocar músicas, listar tarefas, fazer pesquisas de tráfego, temperatura entre outras informações, além de controlar sistemas e dispositivos inteligentes conectados a ela [1]. Nosso projeto fará uso desta capacidade do sistema da Alexa em controlar dispositivos inteligentes.

O ESP8266 é um microcontrolador que possui conexão via Wi-Fi, ou seja, esse chip permite que controladores possam se conectar a uma rede sem fio por intermédio de conexões TCP/IP. São amplamente utilizados como uma “Ponte Serial-WiFi”, já que contam com poucos pinos. É possível receber dados por um aplicativo/WEB e enviar estes dados para um Arduino. O oposto também é possível, enviar dados do Arduino para um aplicativo ou página WEB [2]. No nosso projeto, o ESP8266 será o dispositivo inteligente controlado pela Alexa.

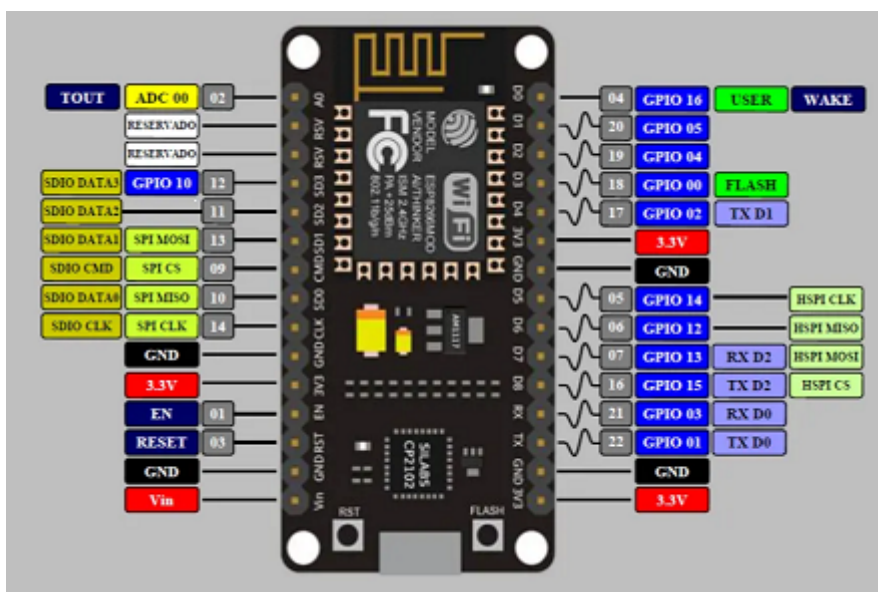


Figura 1: ESP8266 (vamuino, 2019)

O Sinric é um serviço online que fornece uma API (interface de programação de aplicações) para desenvolvimento de dispositivos inteligentes usando placas existentes como o ESP8266, Raspberry Pi ou até mesmo o Arduino [3]. Atualmente o serviço é gratuito e conta com uma versão paga chamada Sinric PRO. Diferente das outras APIs que existem, este serviço se diferencia pois gera uma chave (um ID) para cada dispositivo e o associa a um cadastro. Dessa forma, você está criando de fato um dispositivo inteligente de verdade, e não apenas simulando um dispositivo já existente. Em geral, as outras soluções costumam usar o ESP8266 para simular um dispositivo que já existe comercialmente, como as lâmpadas inteligentes da phillips, no entanto esse método é mais sensível a falhas.

2 Objetivos

O objetivo deste projeto é realizar o controle e a execução das atividades do robô, usando um microcontrolador (ESP8266) e a Alexa via comando de voz e/ou aplicativo. A interface entre o microcontrolador e o robô será o módulo IO.

3 Fluxograma da lógica de programação

O ponto chave do projeto está nas interfaces de comunicação. O fluxograma a seguir mostra de forma gráfica cada elemento do sistema.

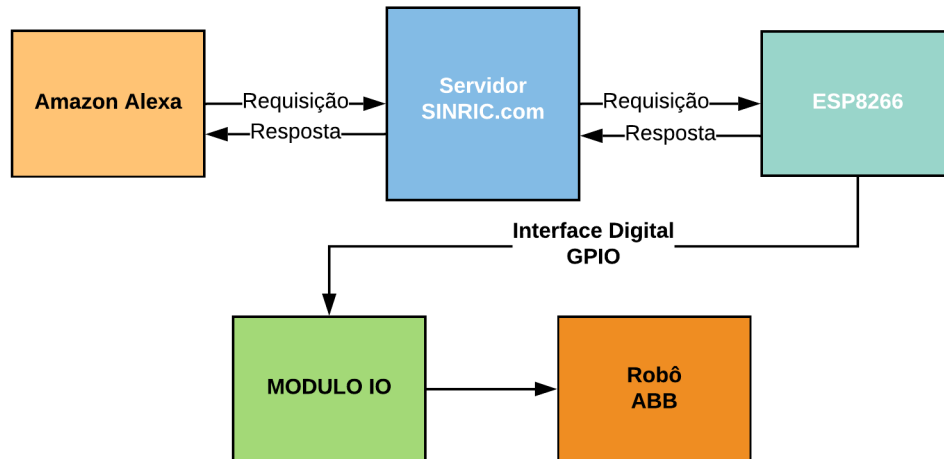


Figura 2: Diagrama representativo do sistema de comunicação

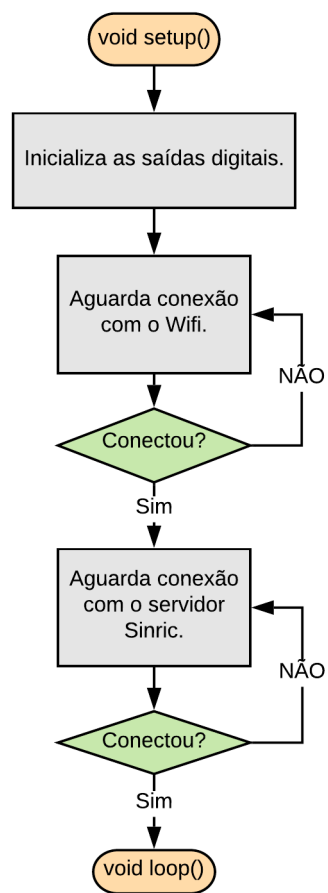


Figura 3: Fluxograma da função setup.

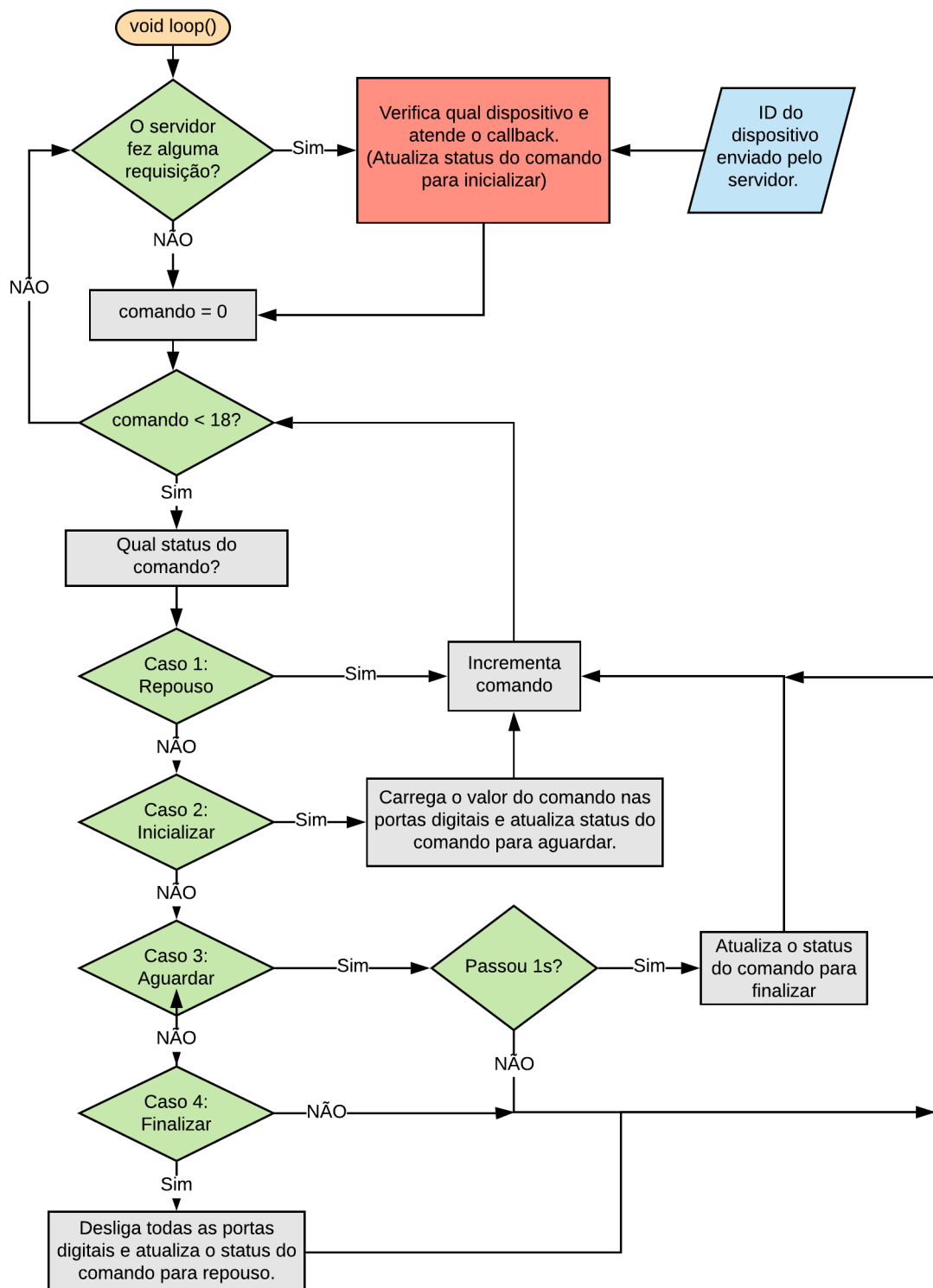


Figura 4: Fluxograma da função loop.

4 Metodologia

4.1 Especificações do projeto

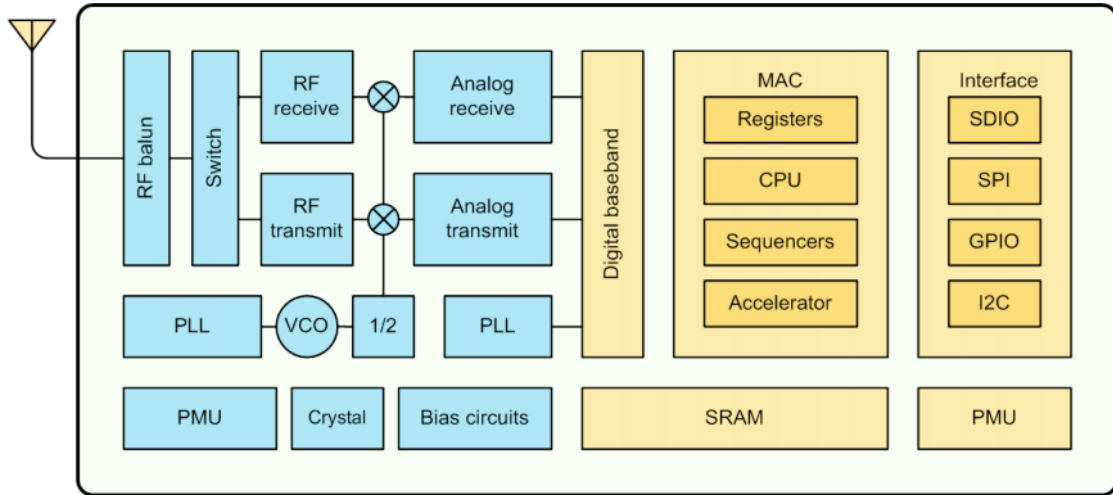


Figura 5: Diagrama de blocos do ESP8266

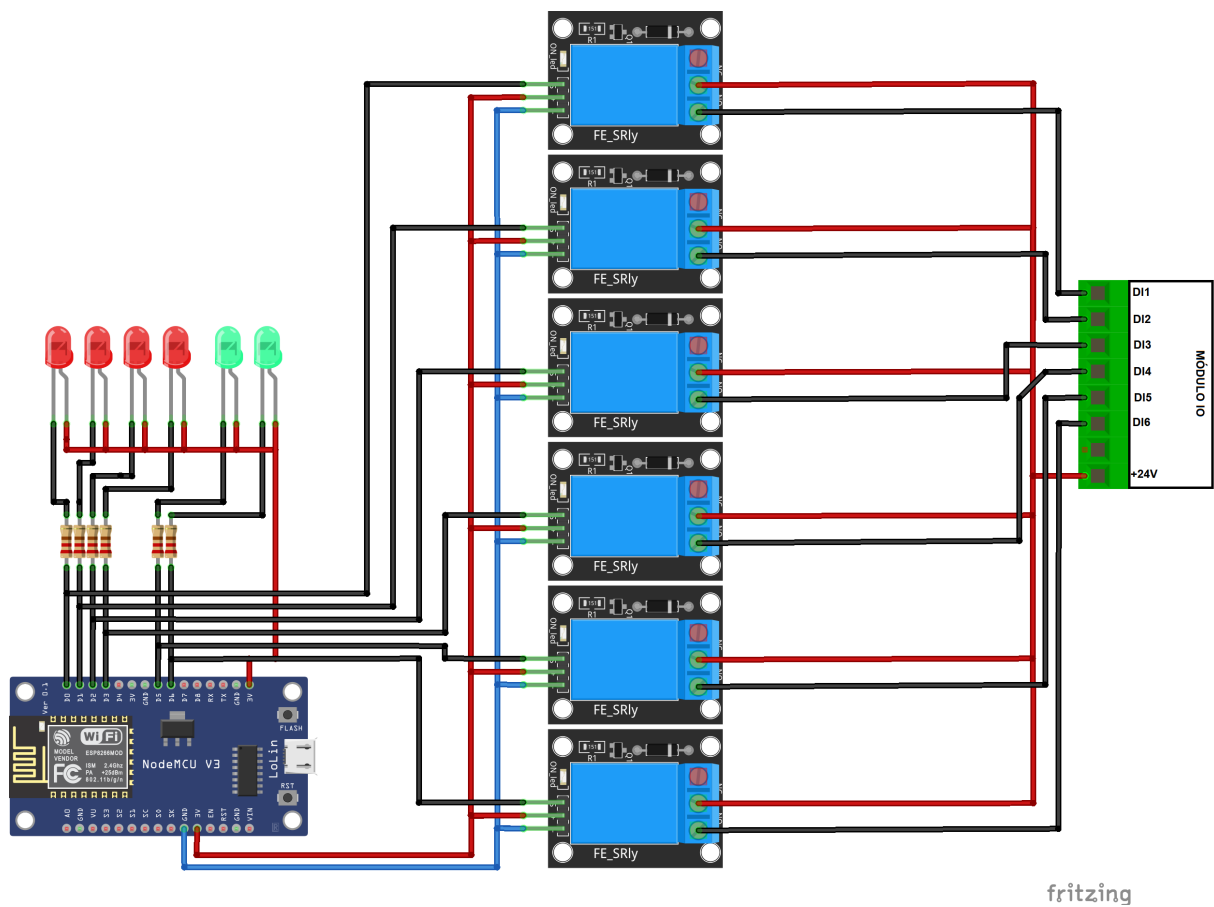


Figura 6: Esquema elétrico do Projeto

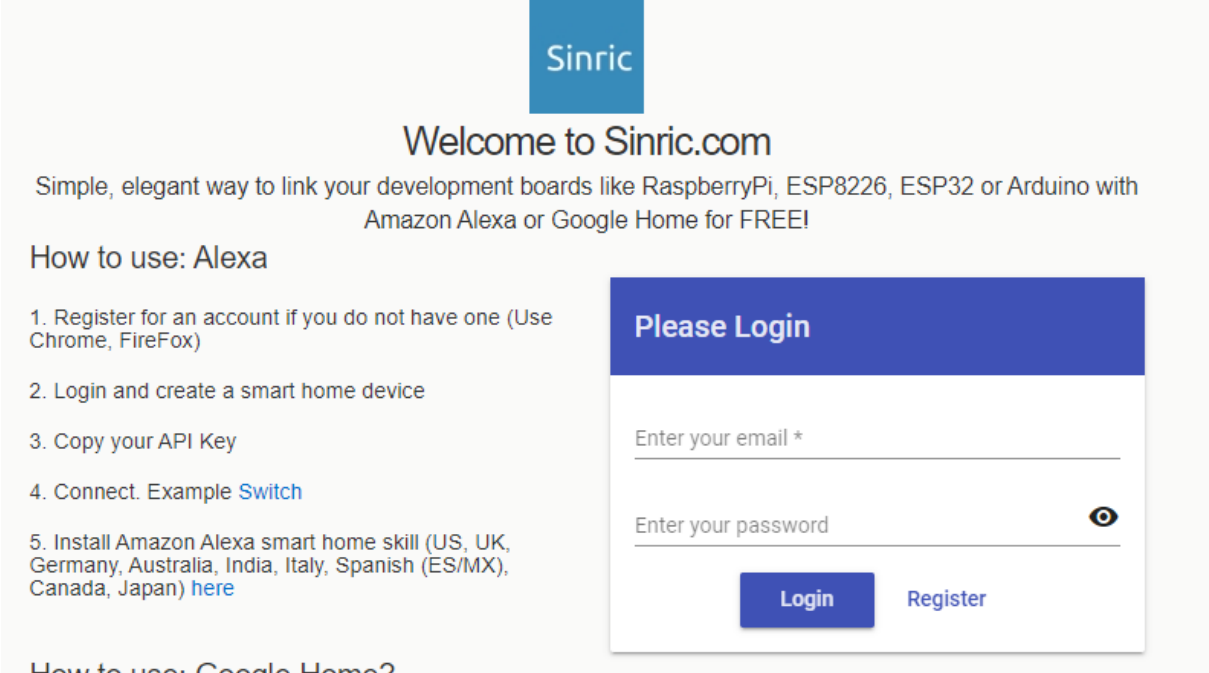
Categories	Items	Parameters
Wi-Fi	Standard	CCC / FCC / CE / TELEC / SRRC
	Protocols	802.11 b/g/n
	Frequency Range	2.4 G ~ 2.5 G (2400 M ~ 2483.5 M)
	Tx power	802.11 b: +20 dBm
		802.11 g: +17 dBm
		802.11 n: +14 dBm
	Rx Sensitivity	802.11 b: -91 dbm (11 Mbps)
		802.11 g: -75 dbm (54 Mbps)
		802.11 n: -72 dbm (MCS7)
	Antenna	PCB trace, external, IPEX connector, ceramic chip
Hardware	Peripheral interface	UART / SDIO / SPI / I2C / I2S / IR Remote Control
		GPIO / PWM
	Operating voltage	3.0 V ~ 3.6 V
	Operating current	Average: 80mA
	Operating temperature range	-40 °C ~ 125 °C
	Storage temperature range	-40 °C ~ 125 °C
	Package size	QFN32-pin (5 mm x 5 mm)
	External interface	N/A
Software	Wi-Fi mode	station / softAP / SoftAP + station
	Security	WPA / WPA2
	Encryption	WEP / TKIP / AES
	Firmware upgrade	UART Download / OTA (via network)
	Software development	SDK for customised development / cloud server development
	Network Protocols	IPv4, TCP / UDP / HTTP / FTP
	User configuration	AT Instruction Set, Cloud Server, Android/ iOS App

Figura 7: Especificações Elétricas do ESP8266[4].

4.2 Configuração do Sinric e da Alexa

Antes de começar a programação, é necessário configurar os dispositivos no site da Sinric e gerar seus IDs. Para isso, segue-se o seguinte procedimento:

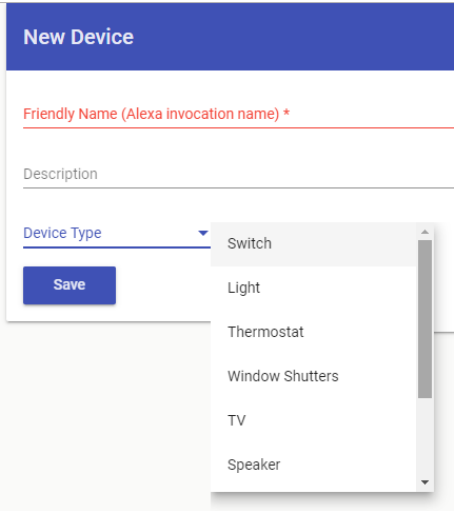
Passo 1: Criar uma conta no sinric.com [5]



The screenshot shows the Sinric.com homepage. At the top is the Sinric logo. Below it, the text "Welcome to Sinric.com" is displayed. A subtitle reads: "Simple, elegant way to link your development boards like RaspberryPi, ESP8226, ESP32 or Arduino with Amazon Alexa or Google Home for FREE!". Under the heading "How to use: Alexa", there is a list of five steps: 1. Register for an account if you do not have one (Use Chrome, FireFox); 2. Login and create a smart home device; 3. Copy your API Key; 4. Connect. Example [Switch](#); 5. Install Amazon Alexa smart home skill (US, UK, Germany, Australia, India, Italy, Spanish (ES/MX), Canada, Japan) [here](#). To the right of the steps is a "Please Login" form with fields for "Enter your email *" and "Enter your password" (with a toggle icon). Below the fields are "Login" and "Register" buttons. At the bottom left, the text "How to use: Google Home?" is partially visible.

Figura 8: Interface do site Sinric.com

Passo 2: Configurar os dispositivos e gerar as APIs



The screenshot shows the "New Device" form. It has a blue header with the text "New Device". Below the header are two text input fields: "Friendly Name (Alexa invocation name) *" and "Description". Below these fields is a "Device Type" dropdown menu. The dropdown is open, showing a list of device types: Switch, Light, Thermostat, Window Shutters, TV, and Speaker. A "Save" button is located below the dropdown menu.

Figura 9: Interface para criação de dispositivos.

No nosso projeto vamos usar apenas switches (on/off).

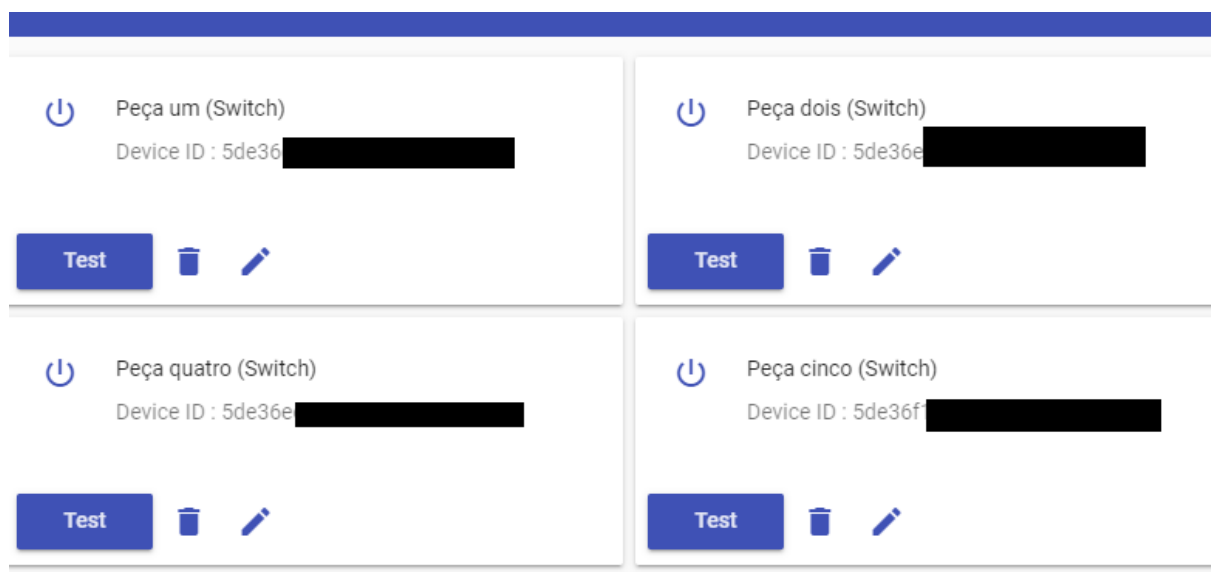


Figura 10: Dispositivos configurados

Passo 3: Adicionar o Skill do Sinric em seu login da Amazon



Figura 11: Skill do Sinric no site da amazon.

Neste ponto, basta baixar o aplicativo da Amazon no celular e fazer login que o Sinric já estará habilitado.

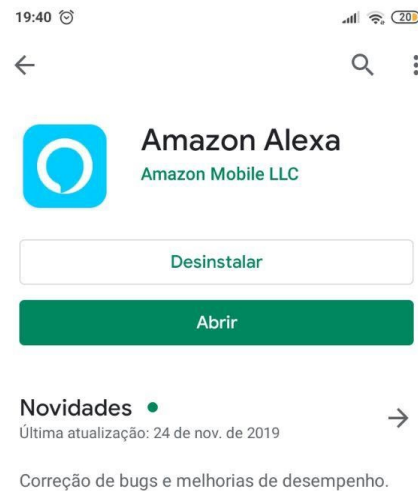


Figura 12: Aplicativo da Amazon Alexa na Playstore.

4.3 Programação do ESP8266

Feita a configuração dos dispositivos, estamos prontos para programar o ESP8266. Nosso programa foi criado a partir do exemplo "switch_basic.ino", e utiliza a biblioteca SinricClass disponibilizada no github do criador do Sinric [6]. Para fazê-lo funcionar, o programa no ESP8266 deve inicializar alguns parâmetros e funções necessárias para que o Sinric envie corretamente as informações para a placa. Um dos parâmetros que é necessário inicializar é a API Key, disponibilizada no seu login no site. Dentro do programa esse parâmetro será passado na função "Sinric.begin()".

```
22 // Definição da API-KEY
23 #define API_KEY "3c64b454-0668-45b
24
257 void setupSinric() {
258     Serial.print("Connecting Sinric");
259     Sinric.begin(API_KEY);
260     while (!Sinric.isConnected()) {
261         Serial.print(".");
262         delay(500);
263     }
264     Serial.println("connected!");
```

Figura 13: Definição da API_KEY no programa e chamada da função Sinric.begin.

Outro procedimento que deve ser feito é a definição de uma função de callback para cada dispositivo.

No caso dos switches, o callback possui a estrutura básica mostrada na figura 14.

```
92 // callbacks dos switches
93 void pc_01_callback(const String& deviceId, bool state) {
94     // Código do usuário
95     if(state){
96         stt_pc[pc_01] = inicializar;
97     }else{
98         stt_pc[pc_01] = finalizar;
99     }
100 //Retorna
101 }
102
```

Figura 14: Função de callback para os switches.

Os parâmetros "deviceId" e "state" são obrigatórios. Já o nome da função e o que é feito dentro dela, cabe ao usuário decidir. No nosso caso, a função modifica a flag de status da peça em questão para que ela passe de "repouso" para "inicializar", de acordo com o valor de "state" enviado pelo servidor. A referência para essa função de callback assim como o ID do dispositivo serão passados para um objeto do tipo SinricSwitch no momento da inicialização do Sinric, conforme a figura 15.

```
268 // Cria os objetos Sinric Switches, cada um com sua ID
269 for(int pc = pc_01; pc <= pc_16; pc++){
270     peca[pc] = &Sinric.add<SinricSwitch>(peca_ID[pc]);
271 }
272 dir = &Sinric.add<SinricSwitch>(ID_DIREITA);
273 esq = &Sinric.add<SinricSwitch>(ID_ESQUERDA);
274
275 // Passa a referência da função de callback para cada switch
276 peca[pc_01]->onPowerState(pc_01_callback);
277 peca[pc_02]->onPowerState(pc_02_callback);
278 peca[pc_03]->onPowerState(pc_03_callback);
279 peca[pc_04]->onPowerState(pc_04_callback);
280 peca[pc_05]->onPowerState(pc_05_callback);
281 peca[pc_06]->onPowerState(pc_06_callback);
```

Figura 15: Inicialização dos IDS e das funções de callback de cada switch

Após criar cada uma das funções de callback, a própria biblioteca fará o tratamento da comunicação entre o servidor e o ESP8266. A função que ficará continuamente verificando as requisições do servidor é a função Sinric.handle, localizada no loop.

```
322 void loop() {
323     Sinric.handle();
324 }
```

Figura 16: Inicialização dos IDS e das funções de callback de cada switch

A informação que precisamos passar para o robô é o valor numérico da peça escolhida e o lado escolhido para jogar a peça. Após alguns testes, optamos por criar o protocolo controlado pela rotina descrita no fluxograma da figura 4. Ele consiste em uma ação em 4 instantes:

1. Carregar o valor numérico da peça escolhida nas saídas D0..D3 (Referentes ao BIT0, ..., BIT3).
2. Setar a saída D6 (JOGAR).
3. Aguardar 1 segundo.
4. Desligar todas as saídas.

É importante observar que o módulo de relês usado no projeto é **ativo em zero**. Isso significa que para ligar o relê, precisamos colocar a saída digital em nível lógico 0. Por conta disso e para melhorar a "redabilidade" do código, fez-se essa inversão através do uso das palavras ON/OFF relativos aos relês no código. A partir de agora, ao nos referirmos a "ligar" ou "desligar" uma saída, estamos nos referindo aos relês.

```

47 // Definição auxiliar para inverter a lógica das
48 // portas digitais na hora de acionar o relê.
49 // Como o relê é ativo em LOW, definimos:
50 // ON -> LOW
51 // OFF -> HIGH
52 #define OFF HIGH
53 #define ON LOW

```

Figura 17: Inversão auxiliar dos níveis lógicos das portas.

Caso a ação seja escolher um lado, o valor carregado no primeiro passo é alterado para a saída DIR, onde esquerda e direita são escolhidos. As tabelas a seguir mostram os instantes do protocolo. Obs: 0 - Desligado, 1 - Ligado, X - Não importa, A - Variável.

Instante	JOGAR	DIR	BIT3	BIT2	BIT1	BIT0
1	0	X	A	A	A	A
2	1	X	A	A	A	A
3	1	X	A	A	A	A
4	0	X	0	0	0	0

Tabela 1: Exemplo de passos para selecionar peça.

Instante	JOGAR	DIR	BIT3...BIT0
1	0	A	X
2	1	A	X
3	1	A	X
4	0	0	X

Tabela 2: Exemplo de passos para selecionar lado.

Peça	D3	D2	D1	D0
1	0	0	0	0
2	0	0	0	1
3	0	0	1	0
4	0	0	1	1
5	0	1	0	0
6	0	1	0	1
7	0	1	1	0
8	0	1	1	1
9	1	0	0	0
10	1	0	0	1
11	1	0	1	0
12	1	0	1	1
13	1	1	0	0
14	1	1	0	1
15	1	1	1	0
16	1	1	1	1

Tabela 3: Equivalencia entre o numero da peça e as entradas D3..D0

A rotina que implementa esse protocolo é basicamente uma máquina de estados no loop, que verifica o status de cada peça. A máquina de estados possui 4 estados: repouso, inicializar, aguardar, finalizar.

```

66 enum status_jogada {
67     repouso, inicializar, aguardar, finalizar,
68 } stt_pc[QTDE_PECAS], stt_dir, stt_esq;
69

```

Figura 18: Declaração da variável de status da jogada de cada peça

O status da peça só sai de repouso se algum dos callbacks for acionado via Sinric. Ou seja, a comunicação só é inicializada quando ocorre uma requisição da Alexa. Quando isso ocorre, o passo de inicialização trata de carregar o valor selecionado nas portas digitais e setar a saída JOGAR para indicar que a jogada começou. Depois disso, o status da peça fica em "aguardar" e após 1 segundo passa para finalizar, onde as saídas são desativadas e o status da peça é atualizado para repouso novamente.

```

325 // Tarefa 2 - Realizar o protocolo p/ peças
326 for (int pc = pc_01; pc <= pc_16; pc++) {
327     switch (stt_pc[pc]) {
328         case repouso:
329             // Não faz nada...
330             break;
331         case inicializar:
332             //Carregar o valor nas digitais
333             t_init[pc] = millis();
334             carregar_peca(pc);
335             //Setar o JOGAR
336             digitalWrite(JOGAR, ON);
337             stt_pc[pc] = aguardar;
338             break;
339         case aguardar:
340             //Aguarda o tempo necessário
341             if (millis() - t_init[pc] > INTERVALO_AGUARDAR) stt_pc[pc] = finalizar;
342             break;
343         case finalizar:
344             digitalWrite(JOGAR, OFF);
345             carregar_peca(0); //Desliga todos os relês...
346             //fauxmo.setState(nome_da_saida[pc], false, 0);
347             stt_pc[pc] = repouso;
348             break;
349     }

```

Figura 19: Parte do código referente à máquina de estados do protocolo.

A função auxiliar "carregar_peca" serve apenas para ligar as saídas digitais. A vantagem no seu uso está na possibilidade de indexação do código. A mesma lógica se aplica quando o comando é de selecionar o lado. A única diferença é que o protocolo usa apenas a saída DIR.

```

397 void carregar_peca(int peca) {
398     // Função responsável por receber o valor da peça e transformá-lo em
399     //um valor binário nas saídas digitais.
400     digitalWrite(BIT0, !(peca & 1));
401     digitalWrite(BIT1, !(peca & 2));
402     digitalWrite(BIT2, !(peca & 4));
403     digitalWrite(BIT3, !(peca & 8));
404 }

```

Figura 20: Função responsável pelo carregamento do valor da peça nas saídas digitais.

Se pelo menos a parte de configuração do Sinric estiver pronta e for gravado no ESP8266, já será possível identificar novos dispositivos com a Alexa.

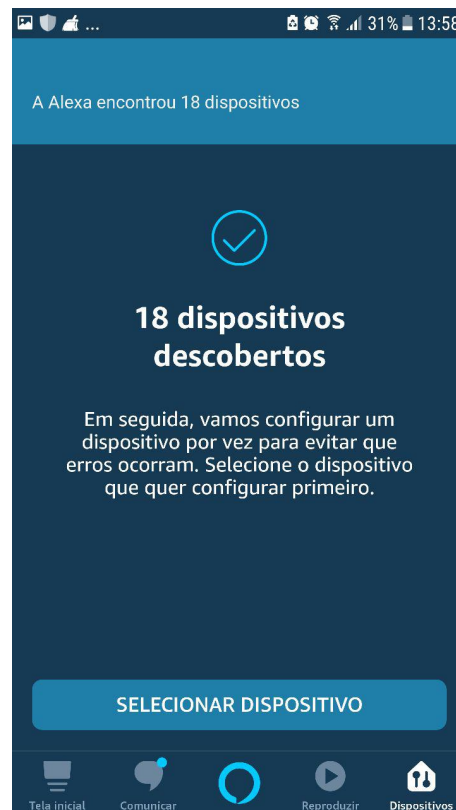


Figura 21: Descobrimto da Alexa identificando os dispositivos no ESP8266

4.4 Configuração do Módulo IO

Criação de entradas e/ou saídas digitais para a simulação do módulo IO.

Passo 1: Selecione a aba "Controller"

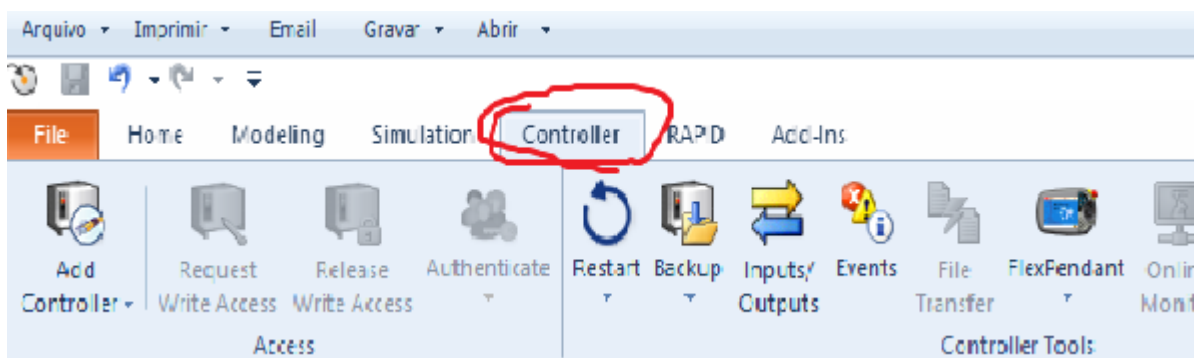


Figura 22: Criação de sinais no I/O

Passo 2: Seleccione a aba I/O System

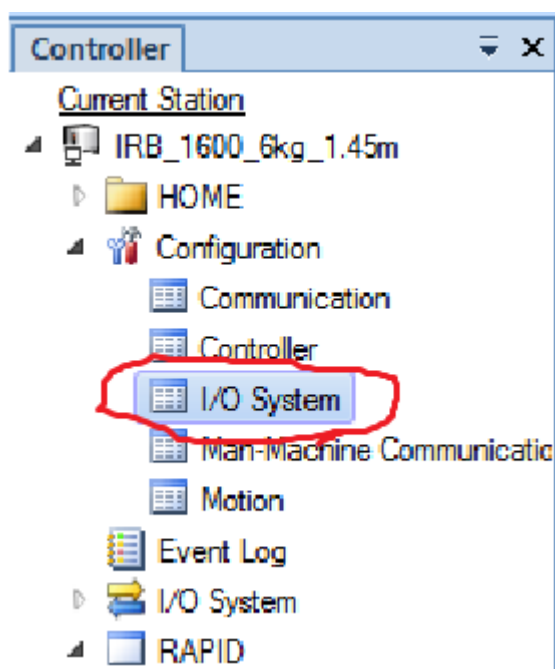


Figura 23: Criação de sinais no I/O

Passo 3: Configure o tópico "Signal"

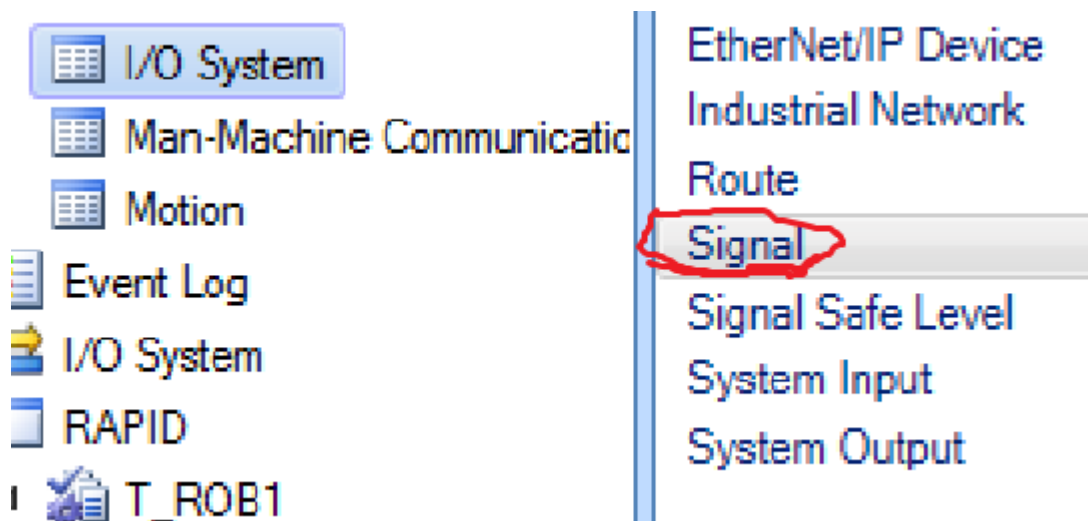


Figura 24: Criação de sinais no I/O

Passo 4: Na edição desse tópico é possível nomear o sinal bem como definir o seu tipo.

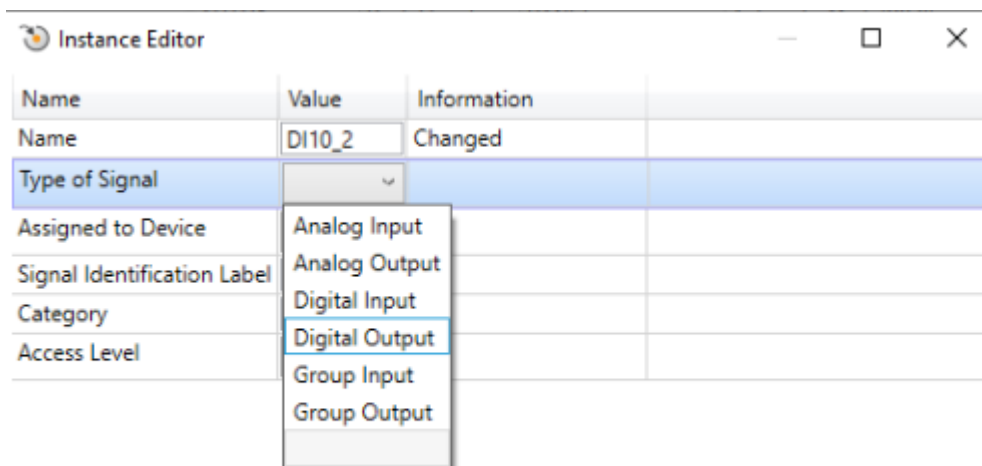


Figura 25: Criação de sinais no I/O

Após clicar em ok os sinais serão criados, com isso é possível simular valores (zero ou um) para cada sinal criado fazendo com que eles reajam a seus valores atribuídos para executar algum comando. No nosso caso, criamos seis entradas digitais que associadas operam algum comando, por exemplo, se a entrada DI10-1 assumisse o valor = 1 e todas as outras valor = 0 e fosse realizado um pulso no DI10-6 o operador entendia que deveria ser jogado a peça 1.

4.5 Modificação no programa do dominó

As únicas modificações necessárias no código original foram nas partes em que o jogador decide o que fazer na hora do jogo, a decisão de qual lado jogar quando é possível e qual peça jogar quando também for possível. Para fazer a modificação, verificamos o funcionamento das funções do RAPID referentes ao módulo IO [7]. A principal função utilizada foi a "WaitDI". Em conjunto com o nosso sinal de "JOGAR", o WaitDI servia para controlar o fluxo do programa, fluxo definido de acordo com o status dos sinais configurados conforme a seção anterior. O entendimento do programa do dominó desenvolvido por uma turma anterior e disponibilizado pelo prof. Sandro se deu através de diversas simulações e testes.

Esse primeiro trecho na figura 26 é o responsável pela escolha do lado em que o jogador quer jogar. Foram usados dois WaitDI antes de passar pra variável que da continuidade ao jogo. O primeiro serve para garantir que a entrada JOGAR estivesse desligada antes de começar a jogada. Isso evita que o programa entre em um *loop* infinito caso o jogador escolha uma peça inválida para jogar. Já o segundo serve para

esperar o sinal de jogar, que ocorre apenas depois que o valor da peça já foi carregado. A partir da entrada do sinal, o robô lê as entradas e armazena o valor da peça, dando continuidade ao jogo.

```
! VERIFICA SE É POSSÍVEL JOGAR A PEÇA DESEJADA, SE NÃO REALIZA A COMPRA DE PEÇAS
IF pecasJogo{pecasJogadores{escolhaPlayer{player},player},1}=pecaD OR pecasJogo{pecasJogadores{escolhaPlayer{player},player},2}=pecaD THEN
  IF ((pecasJogo{pecasJogadores{escolhaPlayer{player},player},1}=pecaE OR pecasJogo{pecasJogadores{escolhaPlayer{player},player},2}=pecaE) AND (player=1)) THEN
    !TPReadFK escolhaLado,"Deseja jogar para qual lado ", "Direita: "+NumToStr(pecaD,0),"Esquerda: "+NumToStr(pecaE,0),stEmpty,stEmpty,stEmpty;
    TPWrite "Deseja jogar para qual lado? Direita ou Esquerda?";
    WaitDI DI10_6,0;
    WaitDI DI10_6,1;
    !IF DI10_6 = 1 THEN
      IF DI10_5=1 THEN
        IF pecasJogo{pecasJogadores{escolhaPlayer{player},player},1}=pecaD THEN
          pecaD:=pecasJogo{pecasJogadores{escolhaPlayer{player},player},2};
        ELSE
          pecaD:=pecasJogo{pecasJogadores{escolhaPlayer{player},player},1};
        ENDIF
        direcaoJogada:=TRUE;
      ELSE
        IF pecasJogo{pecasJogadores{escolhaPlayer{player},player},1}=pecaE THEN
          pecaE:=pecasJogo{pecasJogadores{escolhaPlayer{player},player},2};
        ELSE
          pecaE:=pecasJogo{pecasJogadores{escolhaPlayer{player},player},1};
        ENDIF
        direcaoJogada:=FALSE;
      ENDIF
    !ENDIF
  ELSE
```

Figura 26: modificação no código original

Já esse segundo trecho na figura 27 é o responsável pela escolha da peça que o jogador quer jogar. Assim como no primeiro caso, aqui também foi utilizado o WaitDI com a mesma função, esperar o sinal de jogar após o valor da peça já ter sido carregado.

```
IF (player=1) THEN
  ! CÓDIGO PARA A PESSOA
  TPWrite "Sua vez";
  imprimePecas player,qtdPecasPlayer{player},1.5;
  WaitTime 0.5;

  WHILE escolhaPlayer{player}>qtdPecasPlayer{player} AND podeEscolher DO
    !TPReadNum escolhaPlayer{player},"Selecione a Peça para Jogar";
    TPWrite "Selecione a Peça para Jogar";
    WaitDI DI10_6, 1;
    pecaEscolhida := DI10_4 * 8 + DI10_3 * 4 + DI10_2 * 2 + DI10_1;
    escolhaPlayer{player} := pecaEscolhida;
    IF escolhaPlayer{player}>qtdPecasPlayer{player} OR pecasCompra{pecasJogadores{escolhaPlayer{player},player},1}=-2 THEN
      TPWrite "Número inválido, escolha novamente";
      escolhaPlayer{player}:=-100;
    ENDIF
  ENDWHILE
ELSE
```

Figura 27: Modificação do código original

5 Resultados

5.1 Análise

Apesar de imprevistos durante o desenvolvimento do projeto, obteve-se ao final resultados alinhados com o que foi proposto inicialmente, que era o controle do robô remotamente através da voz. Um aspecto do projeto que foi alterado durante o processo de desenvolvimento e que culminou no uso do Sinric revelou que **a comunicação entre o ESP8266 e o aplicativo da Amazon Alexa não exigia o uso da Amazon Echo Dot**, fato surpreendente pois imaginávamos que era necessário o assistente da Amazon para a comunicação com o aplicativo. Este foi claramente o maior resultado obtido por esse projeto, pois abre um leque imenso de possibilidades de projetos futuros.

Fora isso, todos os passos do projeto chegaram a um desfecho favorável, embora cada um deles apresentasse dificuldades a serem superadas: desde a etapa da elaboração do programa, onde foi desenvolvido o código a ser implementado no microcontrolador e realizado as modificações no arquivo original do domínio, passando pela etapa de simulação e testes, onde foi feita toda a configuração do simulador a fim do mesmo reconhecer as entradas do IO necessárias para a execução do projeto e, assim, poder simular de forma correta os comandos inseridos. Por fim a etapa da execução no robô, onde pode-se verificar o funcionamento do projeto tanto pela simulação de entradas digitais, função esta que está presente no dispositivo e acessível pelo flex pendant, tanto quanto pelos comandos ordenados via aplicativo e voz no módulo IO.

5.2 Dificuldades

Durante a realização do projeto houveram algumas dificuldades a serem superadas, dentre elas destacam-se:

1) A comunicação com a Alexa: Originalmente, estávamos desenvolvendo o projeto usando a biblioteca Fauxmo Alexa, que funciona como um dispositivo inteligente identificável pela Alexa ao simular uma lâmpada Phillips HUE no ESP8266. No entanto, o projeto feito dessa forma gerava uma série de problemas como a necessidade de conectar a Alexa (Echo Dot) e o ESP8266 na mesma rede. Além disso, muitas vezes a Alexa não detectava os dispositivos. Dessa forma, mudamos a forma como faríamos o dispositivo inteligente e passamos a utilizar a API do Sinric.

2) A simulação do IO no RobotStudio: No processo de modificação e teste do código, houve dificul-

dade em entender como funcionava a parte de simulação do IO no RobotStudio, mais especificamente como simular as entradas digitais do módulo, desafio este que foi superado após consulta às referências e algumas tentativas.

3) **[IMPORTANTE!]** Execução do código no robô: Após a realização dos ajustes no código e testes no RobotStudio, teve-se a necessidade de testar os comandos no robô fisicamente, fato este que culminou em alguns imprevistos. O Robô permite, através do Flex Pendant, simular as entradas e saídas digitais para verificar se o robô responde. Até esse ponto, os testes corriam bem. No entanto, quando passamos a tentar receber os sinais através do módulo IO, o robô já não mais respondia de acordo com o esperado. Acontece que quando habilita-se a função de simulação existente no equipamento, o mesmo simplesmente passa a ignorar os comandos REAIS recebidos através do IO. Ou seja, para o correto funcionamento do programa **é necessário a desativação da simulação de cada porta a ser utilizada efetivamente pelo módulo IO**. Essa particularidade foi descoberta graças a ajuda dos técnicos de laboratório e alguns minutos tentando possibilidades.

6 Conclusão

Através do projeto foi possível que os integrantes do grupo explorassem o software RobotStudio na parte de programação do robô bem como na sua simulação, além de interagir diretamente com o robô na hora dos testes. Além disso, o laboratório disponibilizou diversas ferramentas para que a execução do projeto fosse efetivado, isso permitiu que houvesse discussões acerca das problemáticas encontradas, com o intuito de encontrar as melhores soluções. Tudo isso resultou na conclusão do projeto de acordo com as premissas esperadas e obtendo inclusive resultados que não estavam previstos e que podem ser aproveitados futuramente.

A possibilidade de usar a Alexa sem precisar do Echo Dot abre um leque de possibilidade para projetos futuros. Em geral, pode-se usar a API do Sinric em conjunto com o aplicativo da Alexa e o Módulo IO para comandar qualquer tipo de processo no robô, desde que seja feita as adaptações necessárias dentro do código para a comunicação com a interface serial, além da elaboração de um protocolo. Especificamente para este projeto, seria interessante tentar fazer a comunicação inversa, ou seja, passar informações do robô para a Alexa, de forma a excluir a necessidade de usar o Flex Pendant como IHM (interface humano máquina) do robô. No entanto, deve-se atentar às limitações do módulo IO, pois contendo apenas entradas e saídas digitais, a densidade de informações transmitidas passam a ser menores do que se houvesse uma interface serial, sem contar na impossibilidade de se obter leituras analógicas, o que limita as possibilidades de projeto.

Referências

- [1] *Amazon Alexa*. Amazon Mobile LLC. URL: https://play.google.com/store/apps/details?id=com.amazon.dee.app&hl=pt_BR. Acesso em 09 de dezembro de 2019.
- [2] *Guia do usuário do ESP8266*. FilipeFlop. URL: <https://www.filipeflop.com/blog/guia-do-usuario-do-esp8266/>. Acesso em 09 de dezembro de 2019.
- [3] A. TENNAKOON. *Sinric API*. URL: <https://github.com/kakopappa/sinric>. Acesso em 08 de dezembro de 2019.
- [4] *ESP8266 System Description*. Espressif. URL: http://doc.switch-science.com/datasheets/0b-esp8266_system_description_en_v1.4.pdf. Acesso em 08 de dezembro de 2019.
- [5] A. TENNAKOON. *Sinric Official*. URL: <https://sinric.com/login>. Acesso em 08 de dezembro de 2019.
- [6] B. JÄGER. *SinricClass*. URL: <https://github.com/sivar2311/SinricClass>. Acesso em 08 de dezembro de 2019.
- [7] *Technical reference manual - RAPID Instructions, Functions and Data types*. ABB Robotics. URL: <https://usermanual.wiki/Document/RAPID20instructions.607735779/view>. Acesso em 09 de dezembro de 2019.