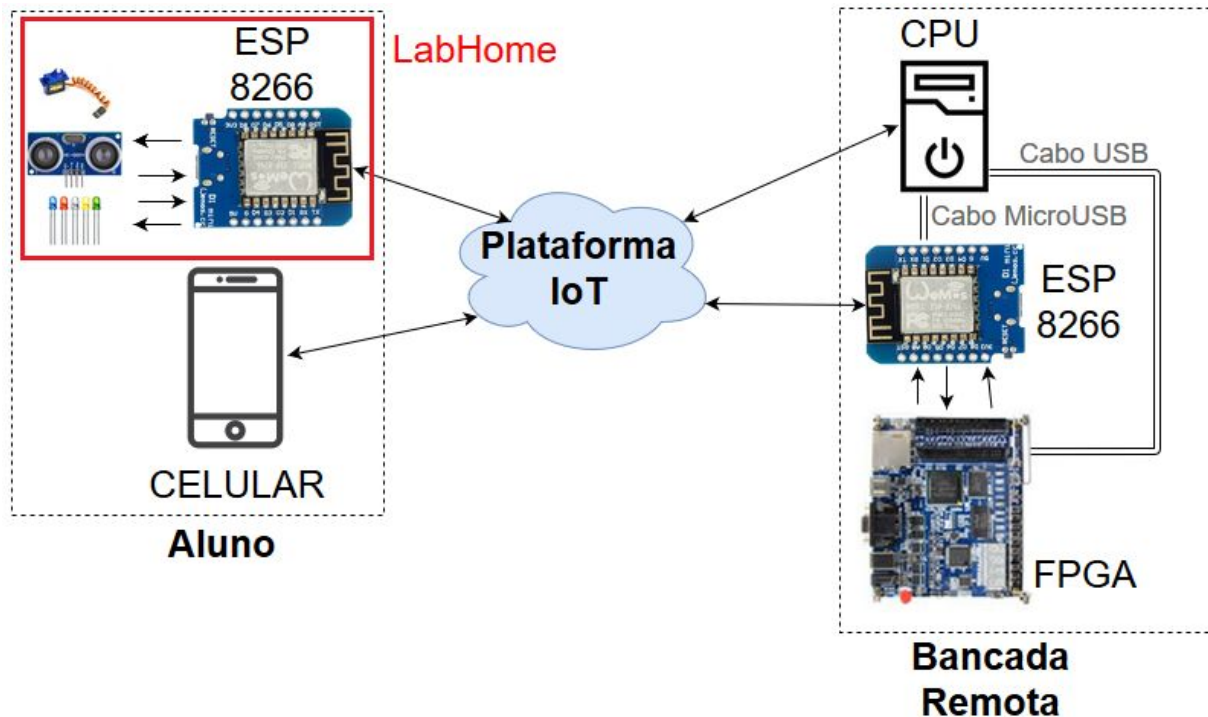


Manual do Projeto LabEAD

1) Apresentação do Projeto

O projeto LabEAD tem como propósito permitir ao aluno, localizado em sua residência, interagir com o aparato experimental disponível no laboratório. A figura a seguir apresenta uma visão alto nível do LabEAD no contexto da disciplina de laboratório digital.

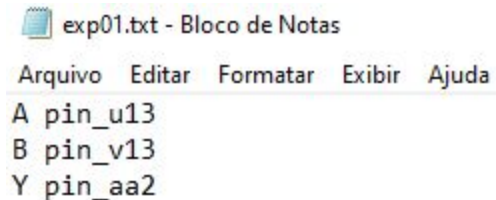


A base do projeto é uma **plataforma IoT**, capaz de permitir a comunicação entre o aluno, através de seu celular, com a chamada **bancada remota**, presente no laboratório. Circuitos integrados **ESP8266** são utilizados devido ao seu módulo WiFi embutido, facilitando assim a comunicação com a plataforma. O aluno possui a opção de integrar um laboratório em sua residência, aqui denominado **HomeLab**, com a bancada remota, permitindo assim uma sincronização entre os sinais de entrada/saída da **FPGA** com os componentes eletrônicos localizados em sua residência.

A integração entre a plataforma IoT e os ESPs é feita através de um **script .ino**, já a integração entre a plataforma IoT e a CPU localizada na bancada remota é feita através de um **script Python**. Esses *scripts* realizam a comunicação com o **Blynk**, que é a plataforma IoT aqui utilizada. Ambos necessitam estar presentes na bancada remota para o projeto funcionar corretamente. Cabe ressaltar aqui que a função do LabEAD no contexto da disciplina de Laboratório Digital é permitir ao aluno ter acesso ao ciclo completo de projeto de um sistema digital abordado na disciplina. Parte-se do princípio que o aluno irá realizar a codificação do sistema digital utilizando uma linguagem de descrição de hardware em sua própria residência, sendo a infraestrutura aqui apresentada responsável pela compilação, carga e testes do sistema.

digital descrito na placa FPGA.

Além dos arquivos correspondentes aos scripts apresentados, mais dois arquivos são necessários para permitir a correta execução do projeto. São eles: **qar do projeto no Quartus** e **um arquivo txt contendo a designação da pinagem**. Ambos podem ser baixados do Google Drive, através de funcionalidade inclusa no script Python, e o arquivo de designação da pinagem também pode ser criado dentro do próprio script Python. O formato do arquivo de designação de pinagem segue o padrão *pinagem pin*, conforme figura a seguir



exp01.txt - Bloco de Notas

Arquivo Editar Formatar Exibir Ajuda

A pin_u13
B pin_v13
Y pin_aa2

Ambos os arquivos devem estar em uma mesma pasta com os arquivos correspondentes ao script Python para o correto funcionamento.

2) Troca do Token

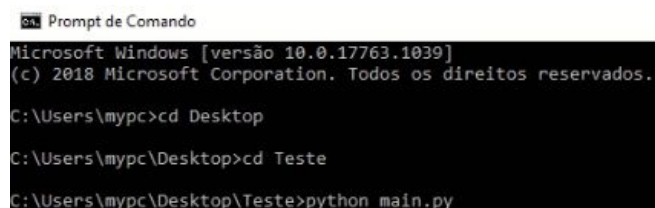
O primeiro passo para o uso do projeto é adicionar o token de autenticação do aluno no script Python. Isso é feito alterando o valor da variável *auth_token* do arquivo *terminal.py*.

O valor do token não precisa ser alterado no script .ino, pois isso é feito automaticamente pela script Python durante sua execução.

```
44 # Algumas constantes para uso nos outros arquivos:
45 #   auth_token - token de autenticao do projeto no Blynk
46 #   pin        - pino virtual onde esta mapeado o terminal do Blynk
47 #   termget    - http request para fazer get (ler) no terminal do Blynk
48 #   termupdate - http request para fazer update (escrever) no terminal do Blynk
49 #   db         - Variavel de depuracao da funcao escreve_terminal
50 auth_token = "A2d3RGwoPfdfVMIAelKNz3_tzrodyBSd"
51 termpin    = "v0"
52 termget    = "http://blynk-cloud.com/" + auth_token + "/get/" + termpin
53 termupdate = "http://blynk-cloud.com/" + auth_token + "/update/" + termpin + "?value="
54 db         = 1
```

3) Executar o Script Python

Pelo terminal do Windows, vá até a pasta onde estão localizados os códigos e digite *python main.py* para começar a execução do script.



Prompt de Comando

Microsoft Windows [versão 10.0.17763.1039]
(c) 2018 Microsoft Corporation. Todos os direitos reservados.

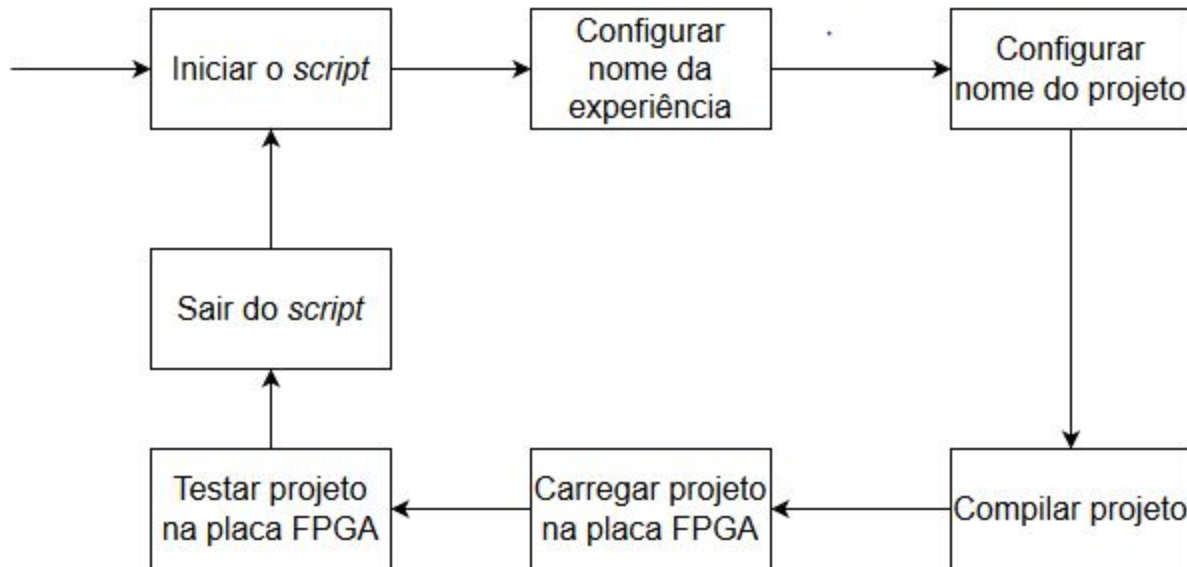
C:\Users\myipc>cd Desktop

C:\Users\myipc\Desktop>cd Teste

C:\Users\myipc\Desktop\Teste>python main.py

4) Fluxo Básico de Execução do Script Python

A figura a seguir ilustra o fluxo **básico** de execução do *script*, partindo-se do princípio que tanto o arquivo qar quanto o arquivo txt de designação de pinagem já estão localizados na bancada remota.



Cada uma das etapas apresentada na figura ocorre mediante a um comando enviado no terminal do Blynk, pelo aluno em seu celular. A tabela a seguir apresenta cada comando com sua etapa correspondente.

	comando	função
1	i	iniciar o <i>script</i>
2	exp	Configurar o nome da experiência (arquivo de designação de pinagem)
3	pr	Configurar o nome do projeto (arquivo QAR)
4	co	compilar o projeto com Intel Quartus Prime
5	ca	carregar o projeto na placa FPGA
6	t	iniciar o teste do projeto
7	s	sair do <i>script</i>

5) Opções do Menu do Script Python

Uma vez apresentado o ciclo básico de execução, é possível abordar todas as funcionalidades disponíveis no *script*. Estas são apresentadas em seu menu, logo após o processo de inicialização do script (comando *i*). A tabela a seguir apresenta todas as opções implementadas no menu do *script*.

comando	abreviatura	função
carregar	ca	carregar um projeto na placa FPGA
compilar	co	compilar um projeto qar com Intel Quartus Prime
testar	t	alterar os valores de entrada
pinar	pi	montar um arquivo de pinagem
exp	e	mudar o nome da experiência
projeto	pr	mudar o nome do projeto (arquivo qar)
baixar	b	baixar um arquivo do Google Drive
mostrar	m	mostrar o valor atual dos parâmetros de exp/projeto
arquivos	ar	mostrar os arquivos/diretórios
remover	r	remover um arquivo/diretório
ajudar	a	mostrar novamente o menu do <i>script</i>
sair	sa	sair do <i>script</i>

A seguir será feita uma subseção para cada comando, visando abordar cada um deles em maior detalhe.

5.1) Comando carregar (ca)

Este comando tem a função de carregar um arquivo executável da placa FPGA (arquivo SOF) na mesma. Ele só pode ser executado após a execução do comando de compilar o projeto. Ele foi implementado com um mecanismo de “tolerância a erros” de tal forma que caso um aluno tente fazer uma designação de pinos não permitida, o processo será finalizado e uma mensagem de erro será apresentada.

5.2) Comando compilar (co)

Este comando tem a função de compilar um projeto utilizando o Quartus Prime, através de chamadas pela linha de comando. Ele só pode ser executado após tanto o nome da experiência quanto o do projeto terem sido configurados. Sua execução consiste em extrair os arquivos presentes no QAR em uma nova pasta, limpar qualquer mapeamento de pinos feita previamente, utilizar o mapeamento presente no arquivo TXT de designação de pinos e compilar o projeto do sistema digital.

5.3) Comando testar (t)

Este comando tem a função de testar um projeto carregado previamente na placa FPGA. Para tal, ele realiza a troca de token no script .ino seguido de sua compilação e carga no ESP8266 com o Arduino CLI (*command line interface*). Esse processo ocorre apenas na primeira vez que o aluno utiliza o script após sua inicialização, não sendo repetido enquanto ele não sair do script. Logo após é possível alterar os valores de entrada da placa FPGA seja por comandos no terminal ou então por botões mapeados em pinos virtuais do Blynk.

5.4) Comando mapear pinos (pi)

Este comando tem a função de criar um arquivo TXT de designação de mapeamento de pinos dentro do próprio script. O aluno pode escolher um nome para o arquivo TXT e em seguida fornecer dados correspondentes às entradas/saídas do sistema digital e a pinagem desejada.

5.5) Comando experiência (e)

Este comando tem como função alterar o parâmetro correspondente ao nome da experiência, utilizando pelo *script*. Cada vez que esse comando é executado o aluno precisará compilar novamente o projeto antes de poder realizar sua carga na placa FPGA.

5.6) Comando projeto (pr)

Este comando tem como função alterar o parâmetro correspondente ao nome do projeto, utilizado pelo *script*. Cada vez que esse comando é executado o aluno precisará compilar novamente o projeto antes de poder realizar sua carga na placa FPGA.

5.7) Comando baixar (b)

Este comando tem como função baixar um arquivo disponível no Google Drive do aluno. Para tal é necessário que o aluno disponibilize o arquivo em seu drive com a opção “Qualquer um que possua o link de acesso”. Será necessário escrever o ID do arquivo, que é um dos campos do link de compartilhamento do mesmo.

5.8) Comando mostrar (m)

Este comando tem como função mostrar ao aluno os valores atuais dos parâmetros correspondentes ao nome da experiência e ao nome do projeto.

5.9) Comando arquivos (ar)

Este comando tem como função permitir aos alunos a visualização dos arquivos/diretórios presentes na sua pasta da bancada remota. Os arquivos correspondentes aos *scripts* (Python e .ino) não são apresentados por se tratar de arquivos reservados.

5.10) Comando remover (r)

Este comando tem como função permitir ao aluno remover remotamente arquivos/diretórios presentes na sua pasta da bancada remota. Os arquivos correspondentes aos *scripts* (Python e .ino) não são apresentados por se tratar de arquivos reservados.

5.11) Comando ajuda (a)

Este comando tem como função mostrar novamente o menu do *script*.

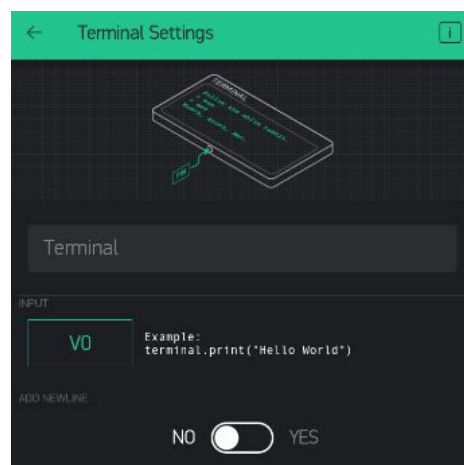
5.12) Comando sair (s)

Este comando tem como função sair do *script*.

6) Funcionamento do Script .ino

O script .ino é uma descrição em linguagem de alto nível do código que controla o ESP8266, e é o responsável pela interação e monitoramento em tempo real com o projeto carregado na placa FPGA. Foi desenvolvido em Arduino IDE, com o uso das bibliotecas Blynk (serviço de computação em nuvem para Internet das Coisas), Software Serial (para comunicação serial UART), Wire (comunicação com módulo expensor PCF8574), e biblioteca específica de comunicação WiFi para a placa de desenvolvimento Wemos D1 mini, que possui como base o ESP8266.

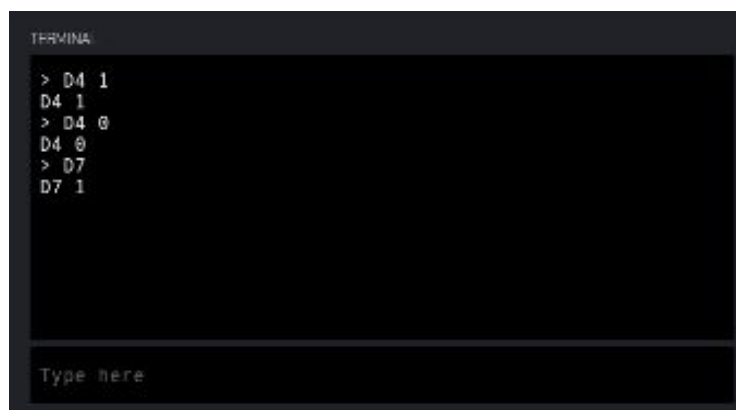
Com o script .ino, é possível por linha de comando controlar e monitorar pinos conectados aos pinos de entrada/saída da FPGA, por meio de um terminal virtual no aplicativo mobile Blynk, mapeado no pino virtual V0.



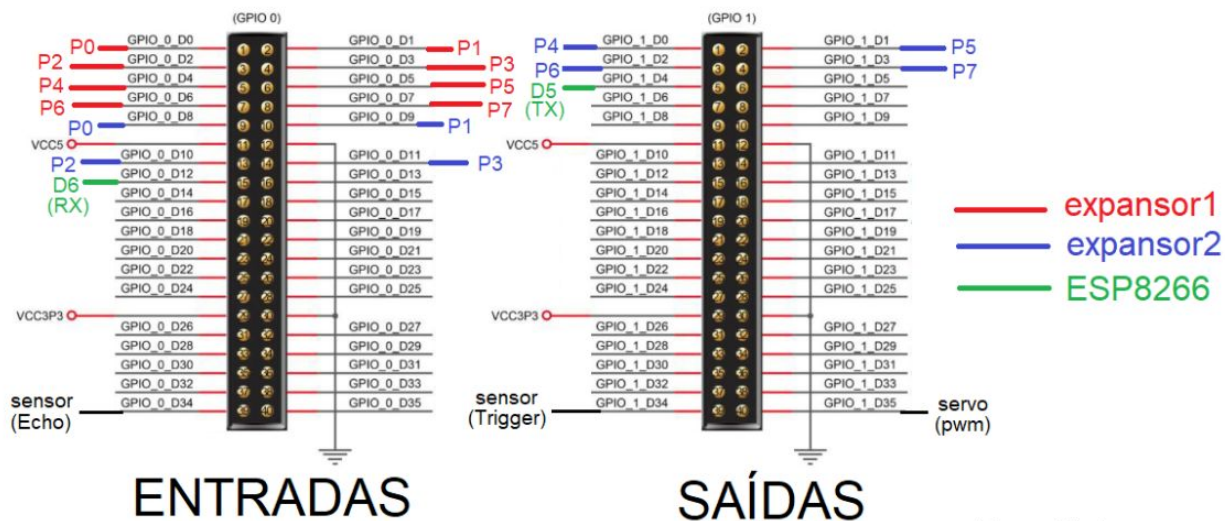
O controle de um pino físico do ESP8266 (D4, que é o led embutido na placa Wemos) está ilustrada na figura abaixo. Através do comando “D4 1”, o usuário comanda o pino D4 do ESP8266 para o nível alto. Com o comando “D4 0”, o usuário comanda o pino D4 para nível baixo. Observe que o led embutido na placa Wemos D1 mini é ativo em baixo.



O monitoramento está exemplificado na figura abaixo. A partir do nome do pino, o script .ino realiza a leitura deste pino e retorna o resultado no próprio terminal virtual V0.

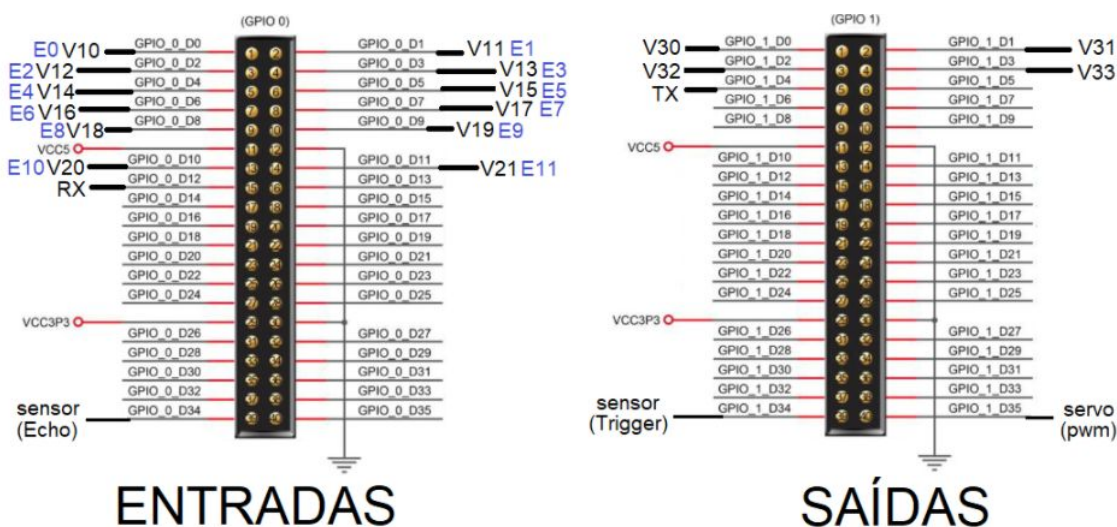


Os pinos de entrada/saída da FPGA (GPIO) estão conectados diretamente nos pinos D5 e D6 do ESP8266, enquanto diversos outros pinos estão conectados nos pinos de duas placas expansoras PCF8574., conforme a figura a seguir.



Através da biblioteca Wire que implementa o protocolo I2C, o ESP8266 controla os 16 pinos dos 8 módulos expansores, uma vez que um módulo expensor está no endereço 0 e outro está no endereço 1 (a configuração de endereços é realizada através de jumpers nos módulos PCF8574). A integração com a biblioteca Blynk permite o mapeamento dos pinos físicos (do próprio ESP8266 e dos expansores) em pinos virtuais. Por sua vez, estes pinos virtuais podem ser mapeados em botões e leds virtuais no aplicativo Blynk.

Desta forma, os pinos da placa expansora podem ser comandados a partir do terminal virtual V0, considerando o mapeamento de E0 a E11, ou então por meio de pinos virtuais mapeados em *widgets* no aplicativo mobile Blynk. Espera-se que as duas possibilidades de interação tornem possível uma maior liberdade na escolha de interfaces personalizadas por experimento, e tornem possível a execução de todos os experimentos com os créditos do nível gratuito da conta do Blynk. A figura abaixo mostra o mapeamento nos pinos do terminal virtual V0 e nos demais pinos virtuais.



Os pinos virtuais nos GPIOs de entrada (V10-V21) podem ser mapeados em botões no próprio Blynk.
Os pinos virtuais nos GPIOs de saída (V30-V33) podem ser mapeados em leds no próprio Blynk.
Os valores em azul (E0-E11) são alterados na opção de teste do script (ex: E0 1, E0 0)

O mapeamento completo da ferramenta está presente na tabela abaixo:

FPGA		ESP8266	expansor 1	expansor 2	Blynk	
-	-	D4	-	-	D4	-
A13 (RX)	GPIO_1_D4	D5 (TX)	-	-	-	-
R21 (TX)	GPIO_0_D12	D6 (RX)	-	-	-	-
N16	GPIO_0_D0	-	P0	-	V10	E0
B16	GPIO_0_D1	-	P1	-	V11	E1
M16	GPIO_0_D2	-	P2	-	V12	E2
C16	GPIO_0_D3	-	P3	-	V13	E3
D17	GPIO_0_D4	-	P4	-	V14	E4
K20	GPIO_0_D5	-	P5	-	V15	E5
K21	GPIO_0_D6	-	P6	-	V16	E6
K22	GPIO_0_D7	-	P7	-	V17	E7
M20	GPIO_0_D8	-	-	P0	V18	E8
M21	GPIO_0_D9	-	-	P1	V19	E9
N21	GPIO_0_D10	-	-	P2	V20	E10
R22	GPIO_0_D11	-	-	P3	V21	E11
H16	GPIO_1_D0	-	-	P4	V30 (LED)	-
A12	GPIO_1_D1	-	-	P5	V31 (LED)	-
H15	GPIO_1_D2	-	-	P6	V32 (LED)	-
B12	GPIO_1_D3	-	-	P7	V33 (LED)	-

Uma demonstração da ferramenta com as placas expansoras está disponível em:

<https://www.youtube.com/watch?v=Tn6grGhI6fI>