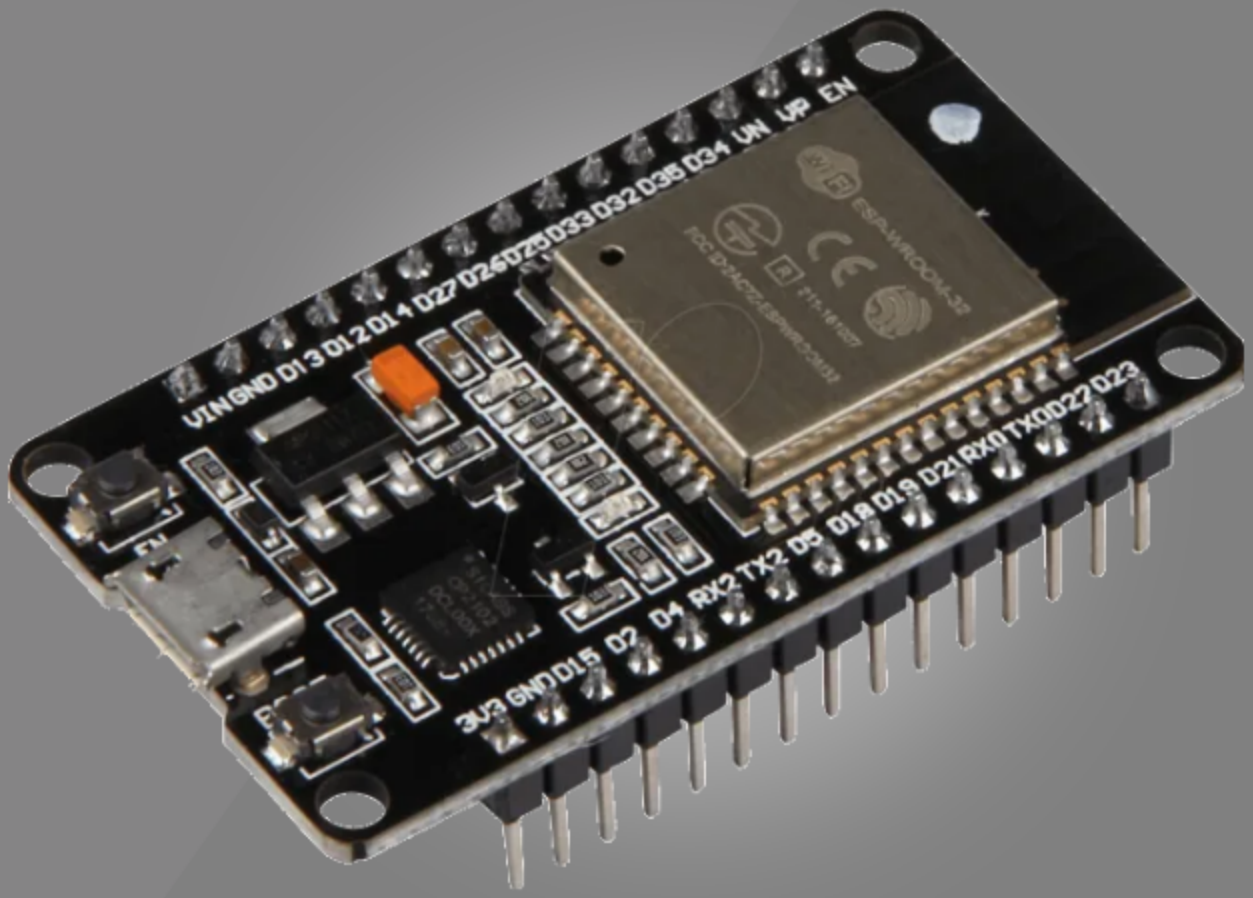
Parte 1

Coleção ESP32 do Embarcados

Comece a programar o ESP32



EMBARCADOS

Ebook - Parte 1

Olá,

Obrigado por baixar o nosso ebook: Introdução ao ESP32 - Parte 1. Esse ebook traz uma coleção de textos já publicados no Embarcados, escrito por diversos autores.

Fizemos um compilado de textos que consideramos importantes para os primeiros passos com o ESP-32. Espero que você aproveite esse material e lhe ajude em sua jornada.

Continuamos com a missão de publicar textos novos diariamente no site.

Um grande abraço.

Equipe Embarcados.

Sumário

ESP32: o sucessor do ESP8266

Autor: Fábio Souza

A Espressif Systems, divulgou hoje uma nota no twitter sobre a chegada do ESP32 para beta testes. O ESP32 será o sucessor do ESP8266 e trará novos recursos e melhorias, como por exemplo conexão WiFi e Bluetooth no mesmo módulo.

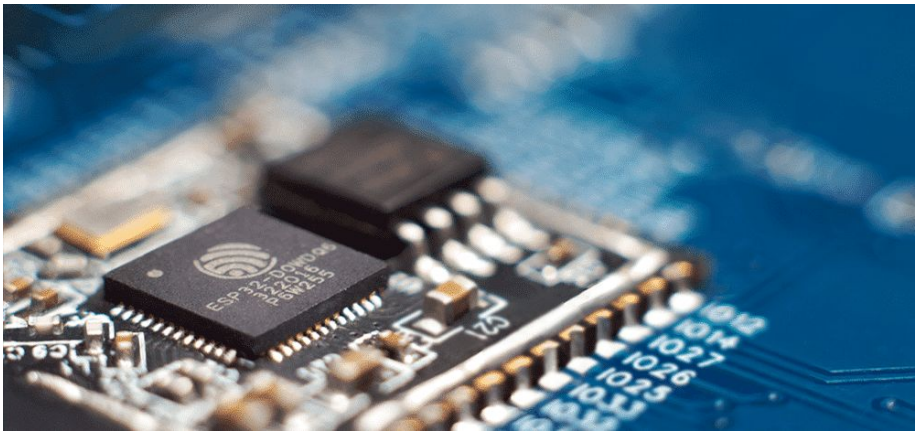


Figura 1 - Módulo com o novo SoC

A seguintes características estarão disponíveis no novo SoC:

- WiFi mais rápido: O novo WiFi foi melhorado para suportar velocidade HT40 (144,4 Mbps).
- Bluetooth e Bluetooth Low Energy;
- 2 processadores Tensilica L108 trabalhando a 160 MHz;
- Low Power: diversos modos de funcionamento para baixo consumo;
- Variedades de periféricos: Touch capacitivo, ADCs, DACs, I2C. UART, SPI, SDIO, I2S, RMII, PWM, mas ainda não terá USB;
- Mais memória RAM: ~400 KB;
- Segurança melhorada: Aceleradores por hardware para AES e SSL, com diversas melhorias;
- APIs simplificadas: a API está sendo melhorada. O desenvolvimento ainda está em progresso e logo estará disponível.

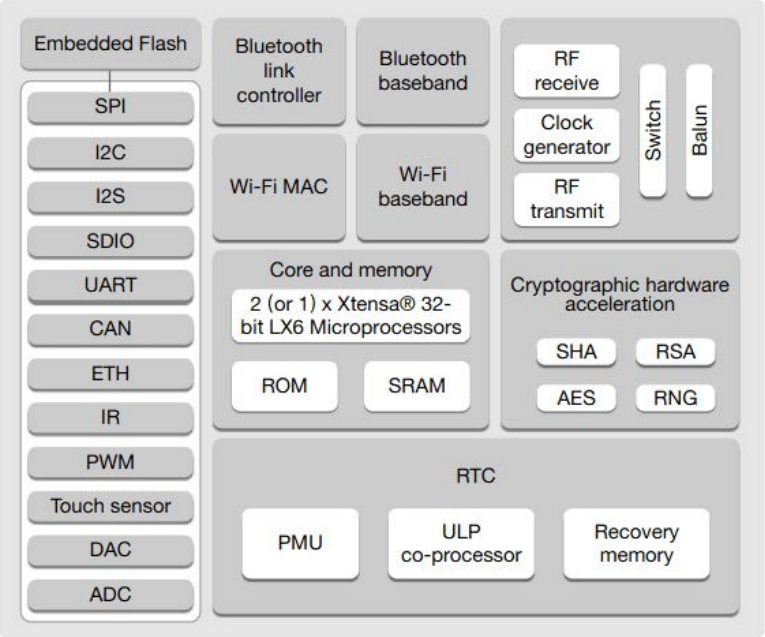


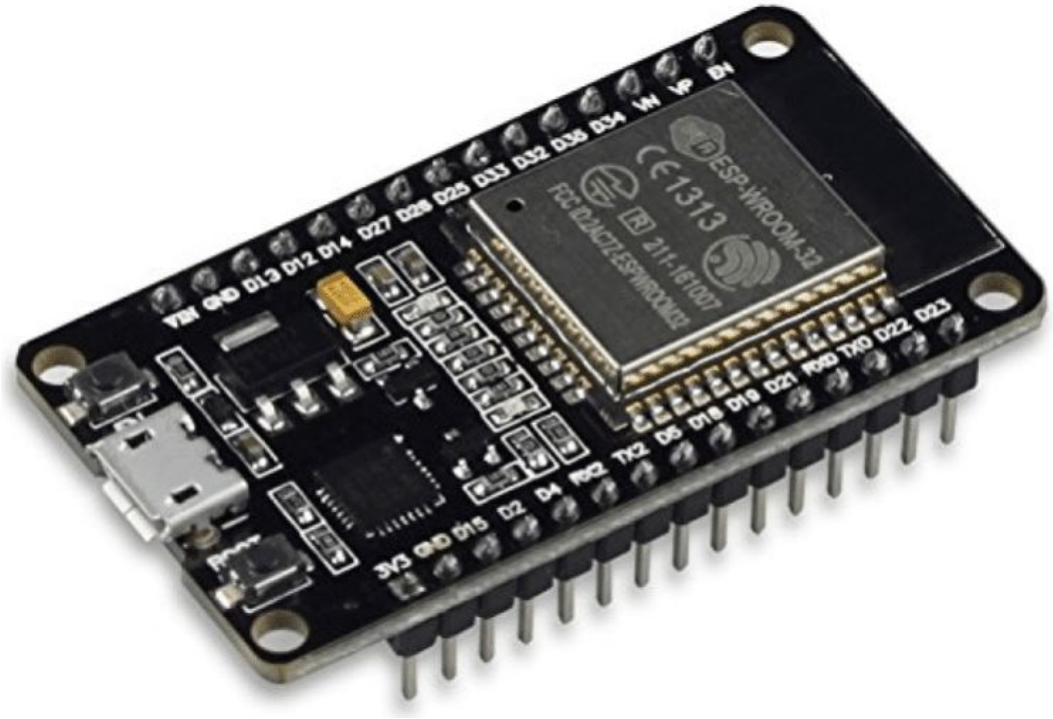
Figura 2 - Diagrama de blocos do ESP32

O programa de testes iniciará em breve, segundo o comunicado da Espressif System serão enviadas placas para desenvolvedores selecionados nas próximas duas semanas. Porém serão apenas 200 placas inicialmente.

No comunicado, eles também fizeram o convite para desenvolvedores de todo o mundo, enviem o seu interesse para trabalhar juntos com eles, seja em Shanghai ou onde estiver, pois a empresa está sempre em busca novos talentos para o time.

Configurando o ambiente de desenvolvimento do ESP32 no Windows

Autor: Gabriel Almeida



O ESP32 é um novo microcontrolador, sucessor do ESP8266, criado e desenvolvido pela Espressif Sistemas, lançado em setembro de 2016. Sua principal inovação, assim como seu antecessor, consiste no uso de módulos de conectividade sem fio para projetos de IoT. O novo dispositivo conta com WiFi, Bluetooth e com um arsenal de periféricos que o torna superior ao Arduíno, nos quesitos de memória, processamento, entre outros.

A finalidade deste artigo é auxiliar o leitor na instalação e configuração do ambiente para o desenvolvimento de trabalhos e projetos que venham a utilizar o ESP32.

As principais plataformas de desenvolvimento de aplicações para o ESP32 são:

- ESP-IDF: framework desenvolvido pela própria ESPRESSIF para o ESP32. É um ambiente mais robusto, porém complicado de usar;
- Arduino IDE: é possível configurar o ambiente do próprio Arduino para manipular o ESP32. Por ser um ambiente mais conhecido, se torna um meio mais simples e acessível para iniciantes.

Instalação do módulo da ESP32 no IDE do Arduino

No caso do Windows, o procedimento de instalação se divide nas seguintes etapas:

- Baixar e instalar o Python (versão 2.7 ou superior);
- Baixar e instalar o Git;
- Clonar o repositório;
- Executar a aplicação get.exe.

O Python e o Git podem ser baixados dos seguintes sites, respectivamente:

- https://www.python.org/downloads/
- https://git-scm.com/downloads/

Para clonar o repositório, execute a interface gráfica do Git:

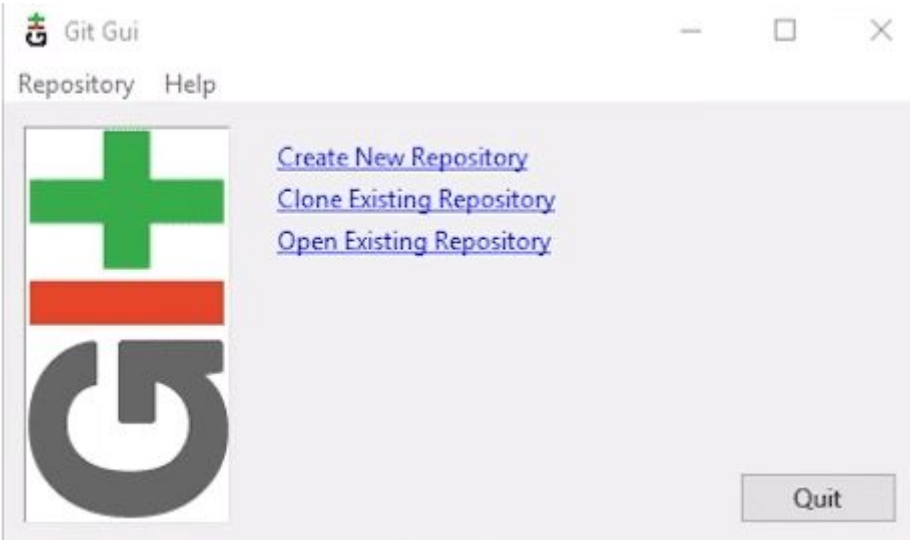


Figura 1: Interface gráfica do Git.

Selecione a opção "Clone Existing Repository". No campo "Source location", coloque o seguinte link:

https://github.com/espressif/arduino-esp32.git

No campo "Target Directory", selecione o local que o IDE do Arduino foi instalado. Deve ser algo parecido com:

C:/Users/[SEU_NOME_USUARIO]/Documents/Arduino/hardware/espressif/esp32

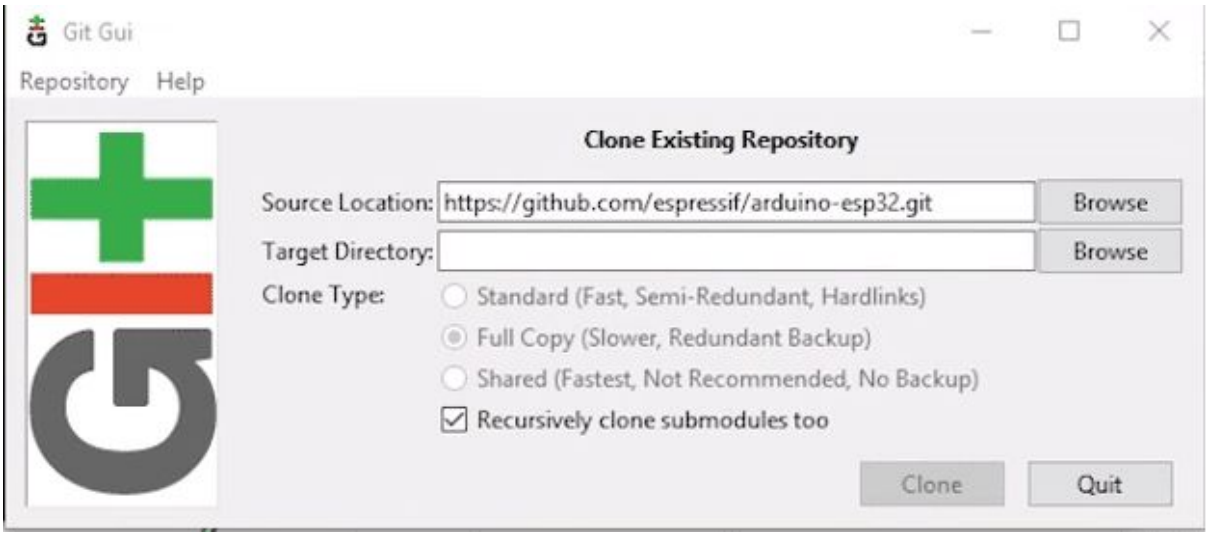


Figura 2: Menu para clonar um repositório.

Após a instalação, na pasta tools, execute o arquivo get.exe e aguarde que os programas sejam baixados e configurados. Depois desse procedimento, o prompt de comando será fechado automaticamente.

Isso encerra a instalação do módulo do ESP32. Agora, para realizar projetos com essa placa, basta programar normalmente no IDE do Arduino, porém, ao invés de selecionar uma placa do Arduino, selecione a sua versão do ESP32.

Instalação do módulo da ESP32 com o ESP-IDF

Continuando com o Windows, para desenvolver aplicações para o ESP32, você precisará:

- Do Toolchain, que é um conjunto de ferramentas de compilação;
- Do ESP-IDF, que contém a API para o ESP32 e scripts para operarem a Toolchain;
- Um editor de texto para escrever programas;
- Da placa ESP32 e um cabo para conectá-la ao computador.

A preparação do ambiente de desenvolvimento consiste em 3 passos:

- Configurar a Toolchain;
- Baixar o ESP-IDF do GitHub;
- Instalar e configurar o Eclipse.

Uma vez que o ambiente esteja configurado, vamos à aplicação. O processo de configurar a aplicação pode ser dividido em 4 etapas:

- Configurar o projeto e escrever o código;
- Compilar o projeto e anexá-lo à aplicação;
- Executar o projeto no ESP32;
- Monitorar e corrigir os erros da aplicação.

Seguindo o primeiro passo, vamos configurar a Toolchain. O jeito mais rápido de proceder é instalar uma pré-definida, que pode ser baixada aqui.

Extraia o arquivo zip em C: e depois abra o terminal MINGW32, localizado em C:msys32mingw32.exe. Crie uma pasta com o nome de esp, com o seguinte comando:

mkdir -p ~/esp

Essa pasta será a localização padrão para desenvolver aplicações do ESP32.

Obtendo o ESP-IDF

Além da Toolchain, você também precisará das bibliotecas específicas do ESP32. Para obtê-las, digite no mesmo terminal utilizado anteriormente o seguinte:

```
cd ~/esp
git clone --recursive https://github.com/espressif/esp-idf.git
```

A Toolchain acessa o ESP-IDF utilizando a variável de ambiente IDF_PATH. Se esta variável não estiver definida, os projetos não são devidamente executados. Essa variável deve ser configurada toda vez que o computador é iniciado, ou permanentemente, no perfil do usuário. Para fazer isso, siga as seguintes instruções:

- Crie um novo arquivo script na pasta: C:/msys32/etc/profile.d/,, com o nome de export_idf_path.sh;
- Identifique o caminho para o diretório do ESP-IDF. Deve ser algo parecido com C:msys32homeuser-nameespesp-idf;
- Adicione no script o comando export, da seguinte forma: export IDF_PATH="C:/msys32/home/user-name/esp/esp-idf"

- Se você não quiser que a variável IDF_PATH seja definida permanentemente, basta abrir o terminal MSYS2 e digitar o comando: export IDF_PATH="C:/msys32/home/user-name/esp/esp-idf"

Configurando o projeto

Para exemplificar o processo de criação e manipulação de um projeto no ESP32, vamos utilizar um código pré-definido, que pode ser encontrado no diretório examples, no ESP-IDF. Primeiramente vamos copiar o projeto hello_world para o diretório esp, digitando o seguinte comando no terminal:

```
cd ~/esp
cp -r $IDF_PATH/examples/get-started/hello_world
```

Agora, na janela do terminal, vá para o diretório hello_world, digitando:

```
cd /esp/hello_world
```

Inicie o menu de configuração do projeto, digitando:

```
make menuconfig
```

O seguinte menu será apresentado, após alguns instantes:

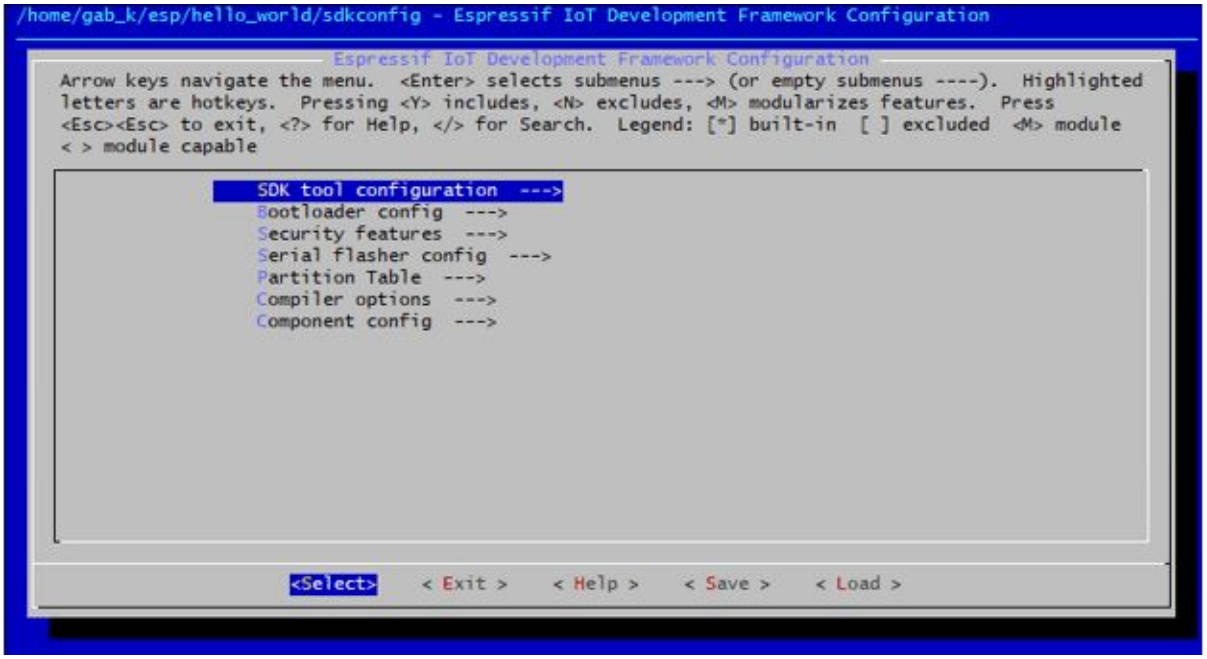


Figura 3: Menu de configuração do ESP-IDF.

Uma vez no menu, navegue para Serial flasher config > Default serial port e digite a porta serial que será utilizada pelo ESP32:

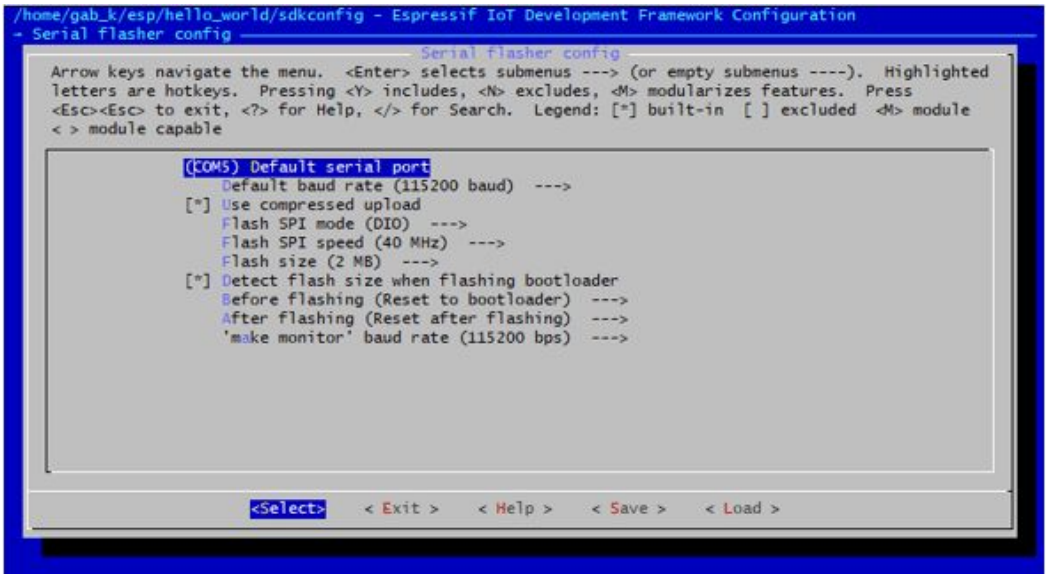


Figura 4: Menu de configuração do flasher.

Assim que as configurações estão devidamente atualizadas, é possível compilar a aplicação e todos os componentes do ESP-IDF. Para fazer isso, digite no terminal:

```
make flash
```

E, se não houver nenhum problema, você verá mensagens descrevendo o processo de carregamento.

Monitor

Para ver se a aplicação “hello_world” está de fato funcionando, digite:

```
make monitor
```

Algumas linhas abaixo, você deve ver a mensagem “Hello world!” aparecendo no terminal.

Isso encerra o processo de configuração de um projeto utilizando o ESP-IDF. Agora que o ambiente está devidamente configurado, as aplicações do ESP32 podem ser trabalhadas.

Referências

1.https://github.com/espressif/arduino-esp32/blob/master/README.md#installation-instructions
2.https://esp-idf.readthedocs.io/en/latest/get-started/

Configurando o ambiente Eclipse para o ESP32

Autor: Muriel Costa

O Eclipse é uma IDE de código aberto muito completa e querida entre desenvolvedores, além de muito flexível permitindo diversos tipos de personalizações. Neste artigo mostro como usá-la para criar um ambiente de desenvolvimento para o ESP32, de forma que dispense a linha de comando para compilar e gravar o projeto. Faremos tudo graficamente.

É importante salientar que este post não cobre a parte da configuração do ESP IDF, portanto, você já deve ter o ambiente configurado para dar prosseguimento. Você pode ver como configurar o ESP IDF neste artigo do Gabriel Almeida. Vale lembrar também que este artigo se passa em um ambiente Windows em minha máquina e talvez alguns passos não se apliquem para usuários Linux.

Download e instalação do Eclipse

Primeiramente você deve fazer o download da ferramenta no site oficial. Na janela que irá abrir você deve escolher a instalação do "Eclipse IDE for C/C++ Developers", conforme a figura abaixo.

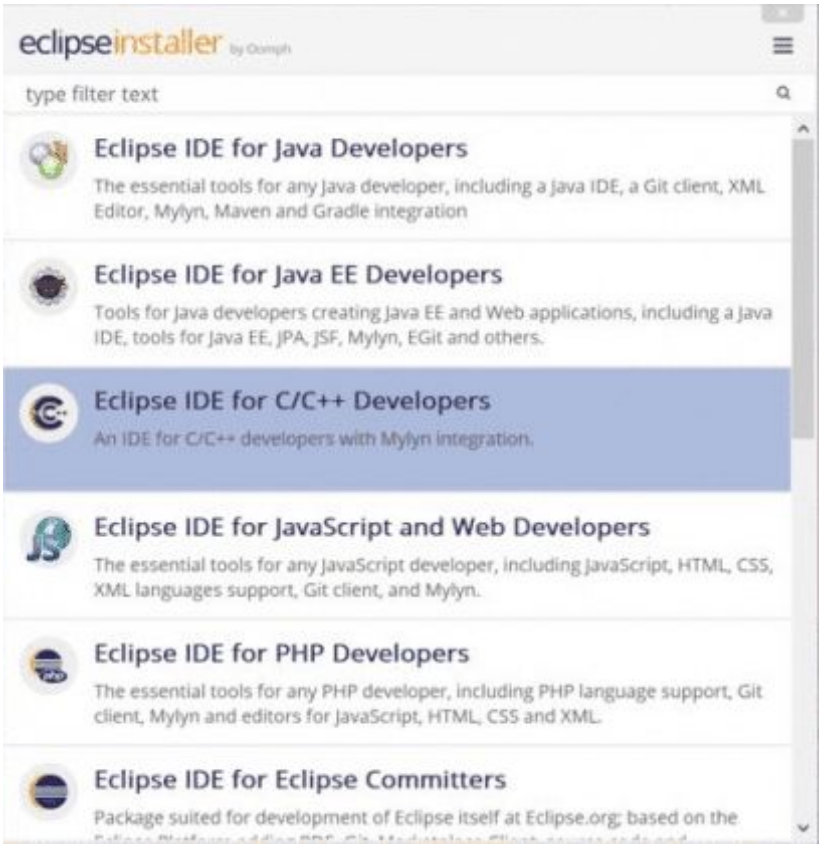


Figura 1: Instalando o eclipse

Após escolher o pacote, os próximos passos serão escolher o diretório de instalação e aguardar a conclusão. Caso você tenha dúvida na instalação da ferramenta, você pode conferir o guia de instalação oficial.

Importando e configurando o projeto

Agora é hora de portar um projeto para o Eclipse! Cuidado ao seguir os passos deste tutorial ao pé da letra, já que pode haver algumas pequenas divergências entre o seu ambiente e a minha instalação e pasta de projetos.

Reconhecendo a estrutura da pasta de projeto

Navegue até a pasta msys32 e entre na pasta que você criou para alocar a pasta esp-idf. No meu caso é a pasta ESP32 e é nela que também fica as pastas dos meus projetos.

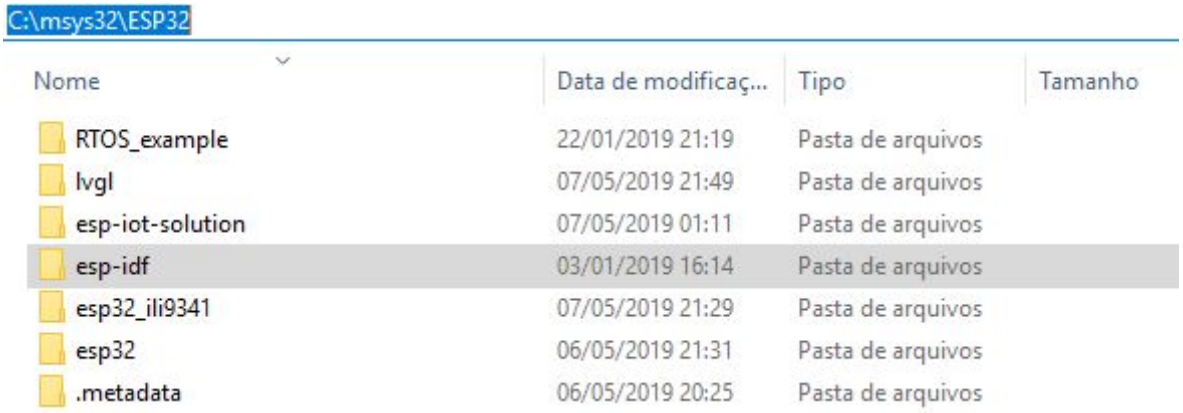


Figura 2: Estrutura da pasta de projetos

Agora é hora de criar um novo projeto, vamos importar um exemplo de projeto "hello world" localizado na pasta esp-idf. Entre na pasta esp-idf -> examples -> get-started. Na

pasta "get-started" você encontrará uma pasta com o nome "hello_world", copie-a na pasta "ESP32" (sua pasta de projetos) ao lado da pasta esp-idf.

O resultado deve ficar como na figura abaixo.

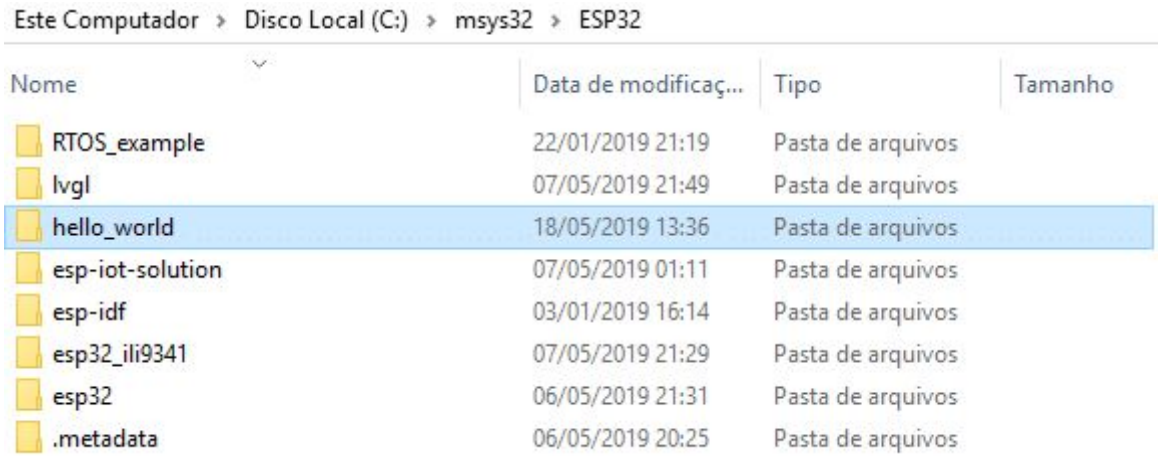


Figura 3: Projeto "hello_world" colado na pasta de projetos

Vamos iniciar o eclipse para dar início à importação e configuração do nosso projeto!

Abra o eclipse e na barra superior clique em file -> import. Obs: Se na inicialização ele pedir pra selecionar um endereço padrão para projetos, selecione a pasta ESP32.

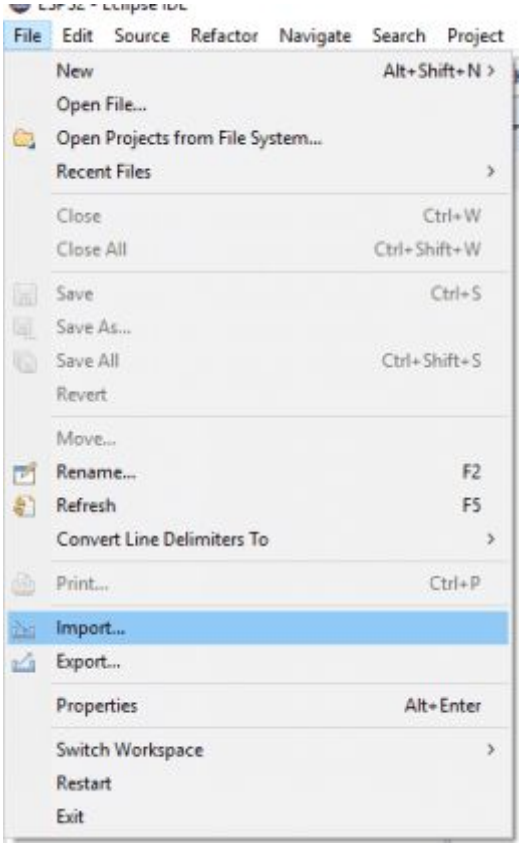


Figura 4: Importando projeto.

Na janela de importação selecione **C/C++ -> Existing Code as Makefile Project**.

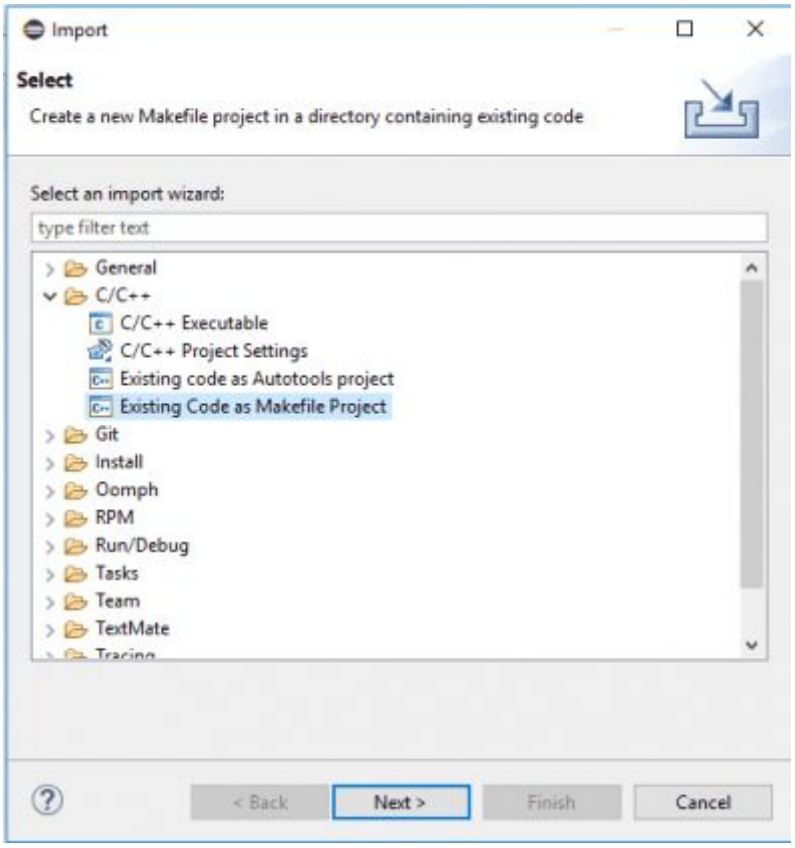


Figura 5: Parâmetros para importação do projeto.

No campo Project Name coloque o nome do seu projeto. No nosso caso, como importamos o hello World, vou mantê-lo. Em Existing Code Location precisamos apontar o endereço onde está o nosso projeto, este diretório deve ser o que contém o arquivo Makefile do nosso projeto.

Selecione a Toolchain Cross GCC e clique em Finish.

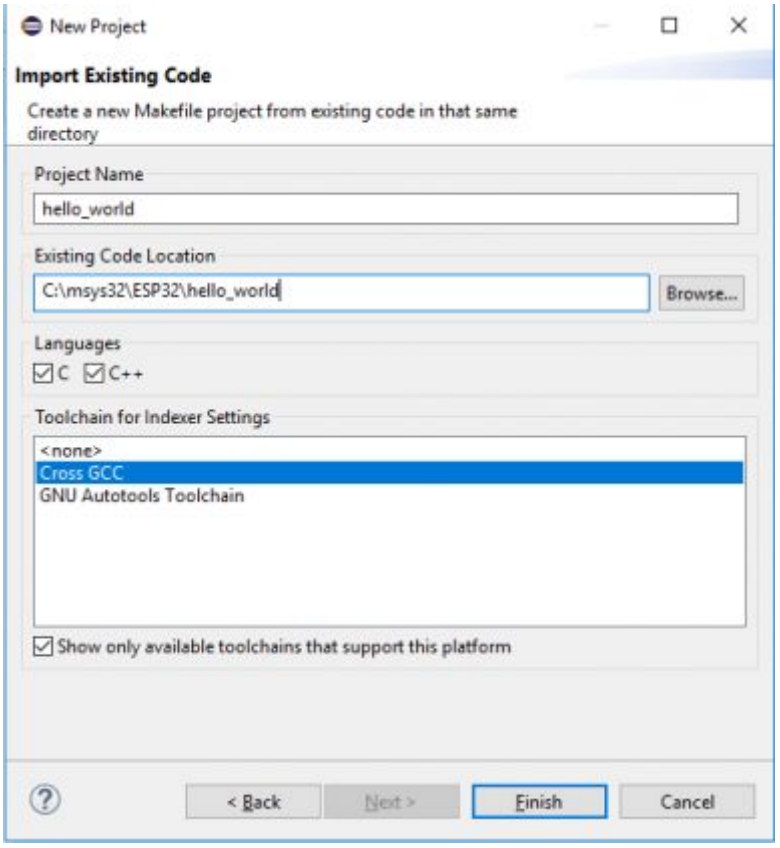


Figura 6: Nomeando projeto e escolhendo Toolchain.

Bom, nosso projeto já está no eclipse e o resultado deve ser como a figura abaixo. Agora a gente precisa configurá-lo, vamos lá!

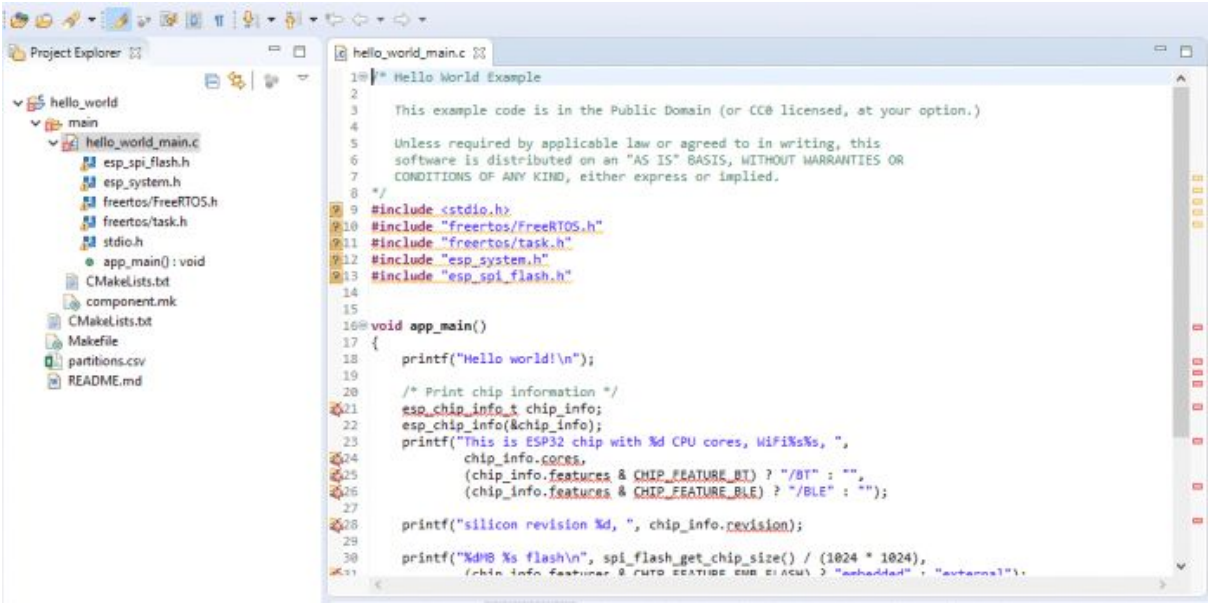


Figura 7: Aparência do eclipse com o projeto aberto

O seu projeto fica aberto no campo do Project explorer, clique nele com o botão direito e clique em propriedades.

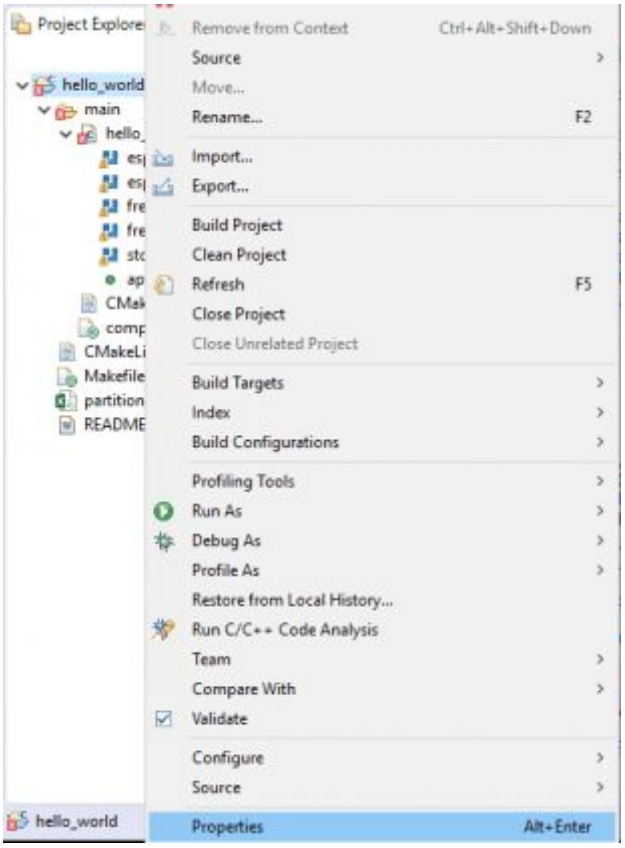


Figura 8: Acessando menu de configuração do projeto.

- Clique em C/C++ Build -> Enviroment e na janela vamos adicionar algumas variáveis e modificar outra.
- Clique em Add e preencha o campo name com "BATCH_BUILD" e valor 1.
- Clique em Add novamente e preencha o campo name com "IDF_PATH" e o valor deve ser o caminho da pasta esp-idf, no meu caso o valor é "C:\msys32\ESP32\esp-idf".
- Clique duas vezes na variável "PATH" para editá-la, no campo value, adicione ao final os seguintes valores ;C:\msys32\mingw32\bin;C:\msys32\opt\xtensa-esp32-elf\bin;C:\msys32\usr\bin

Atenção: não é para apagar o campo value do PATH, é apenas para acrescentá-la ao final.

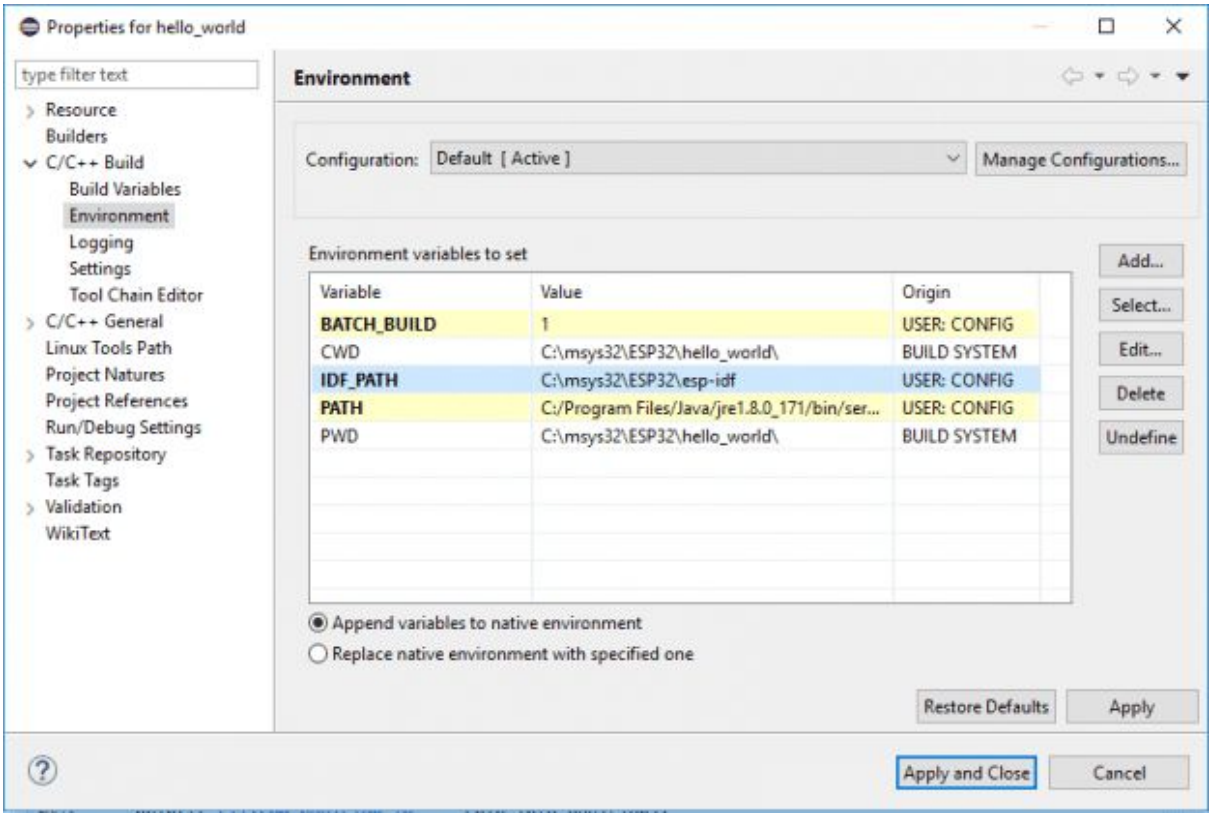


Figura 9: Alterando variáveis do ambiente.

Clique em C/C++ Build e na aba Builder Setings desmarque a opção "use default build command", coloque o seguinte valor no campo Build command: python ${IDF_PATH}/tools/windows/eclipse_make.py

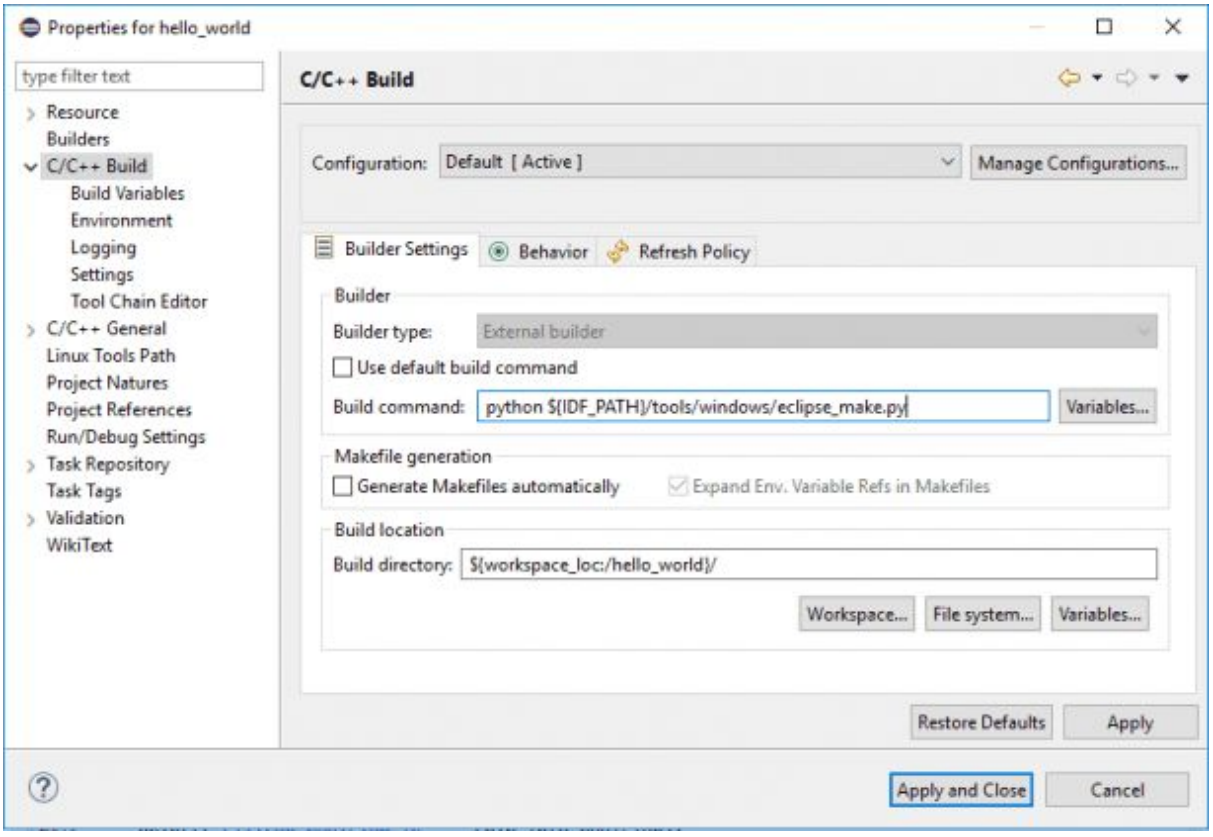


Figura 10: Alterando o comando de build.

Na aba Behavior habilite o campo Enable parallel build para dar mais velocidade ao processo de build.

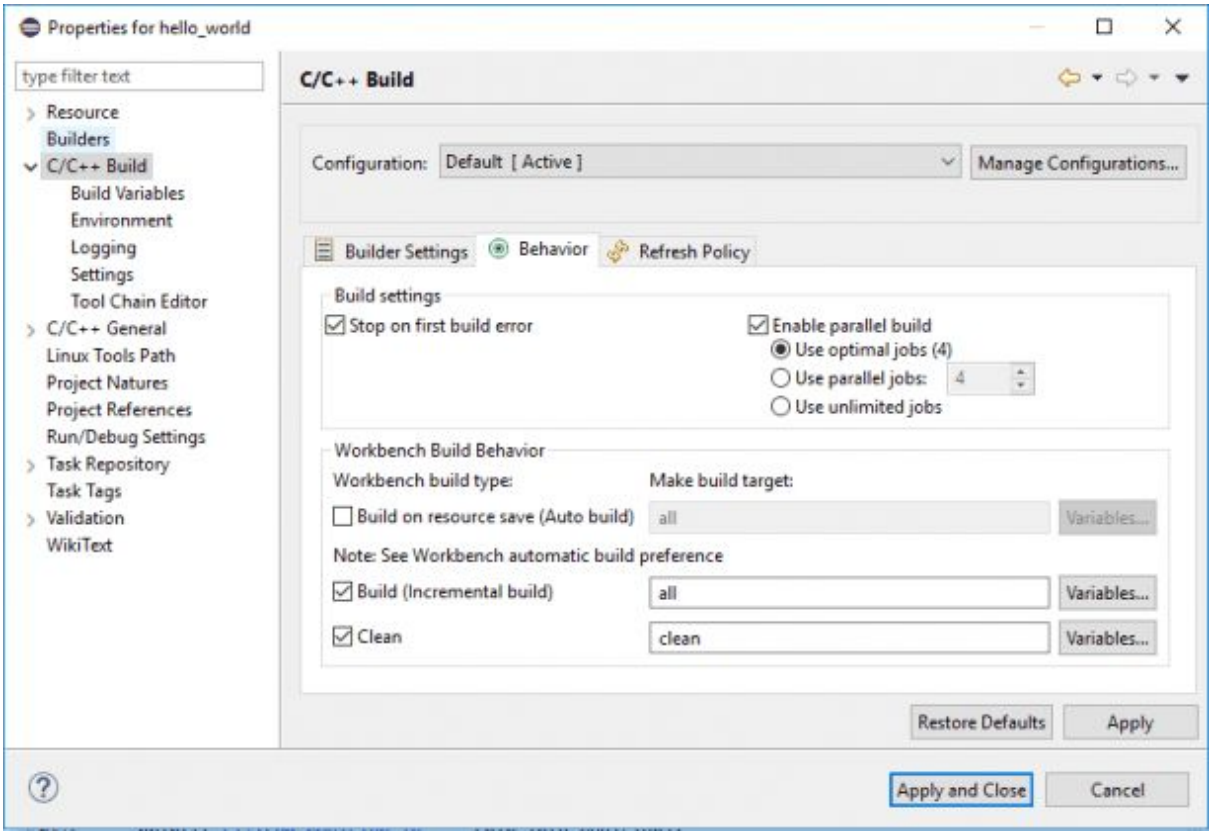


Figura 11: Habilitando paralelismo.

Vá para "C/C++ General -> Preprocessor Include Paths, Macros...". Na aba Providers selecione CDT Cross GCC Built-in Compiler Settings e mude o “Command to get compiler specs” para xtensa-esp32-elf-gcc ${FLAGS} -std=c++11 -E -P -v -dD "${INPUTS}".

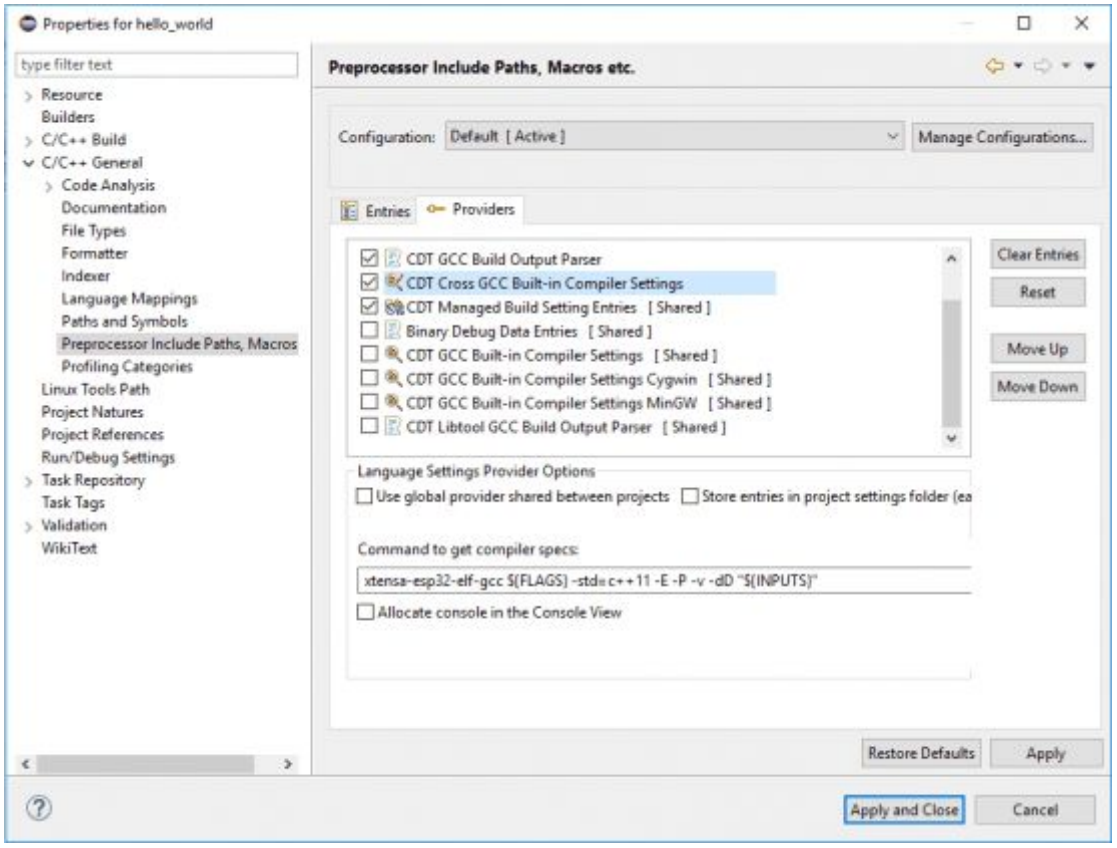


Figura 12: Modificando "Command to get compiler specs".

Agora selecione o “CDT GCC Build Output Parser" e mude o “Compiler command pattern” para xtensa-esp32-elf-(gcc|g\+\+|c\+\+|cc|cpp|clang)

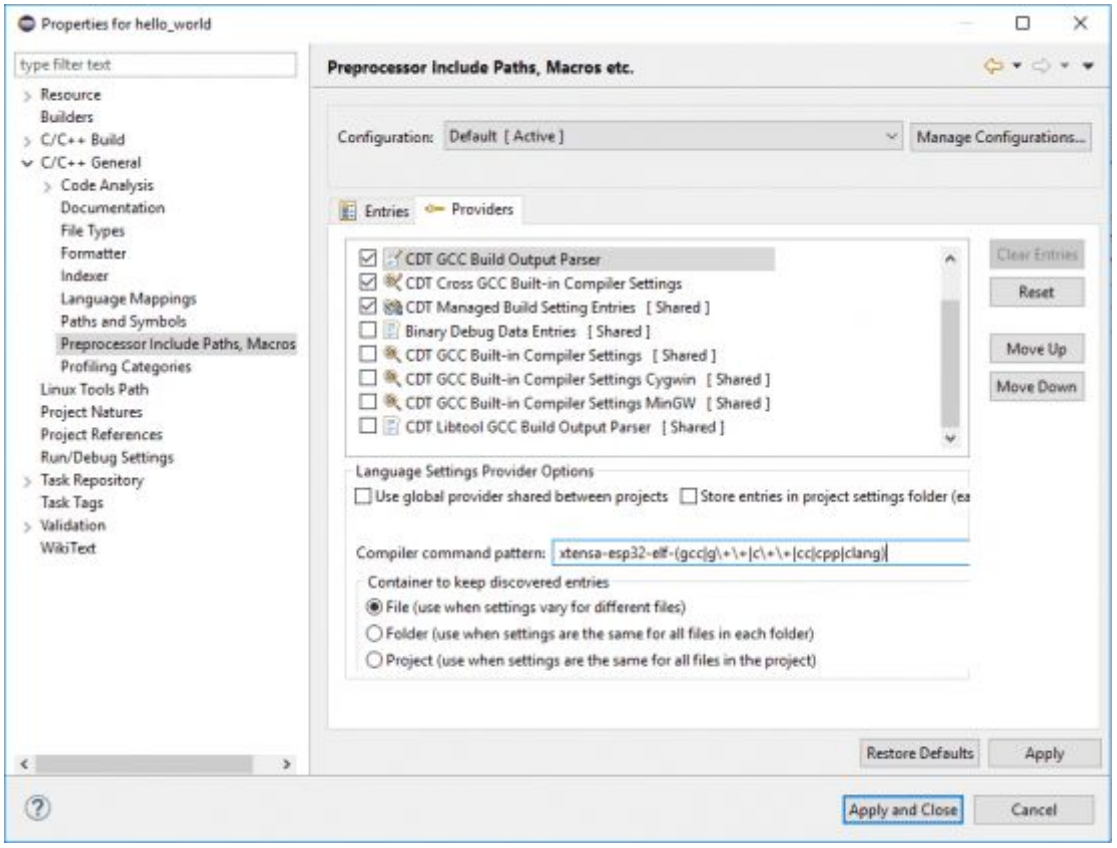


Figura 13: Modificando "Compiler command pattern".

Vá para C/C++ General -> Indexer, habilite o campo "Enable project specific settings" e desabilite o campo "Allow heuristic resolution of includes".

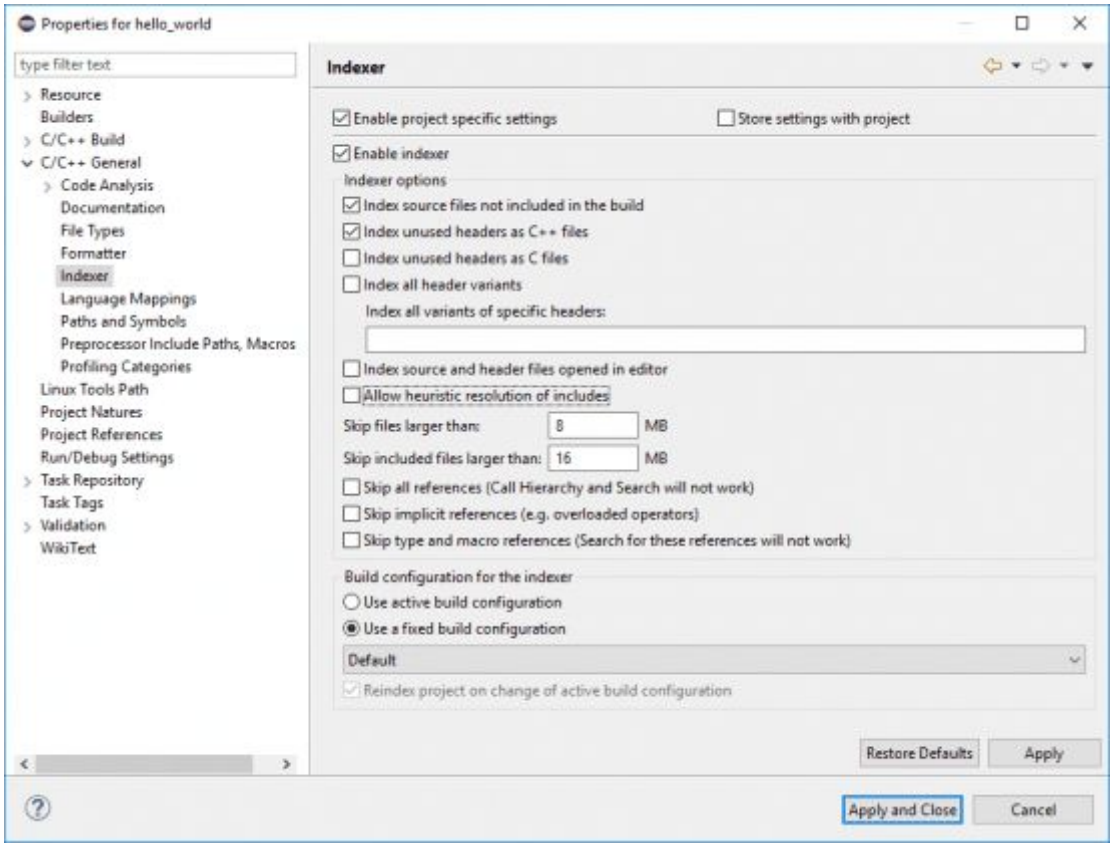


Figura 14: Configurando indexer.

Tudo certo com as propriedades do nosso projeto, agora vamos clicar em "Apply and Close". Agora é hora de adicionar os build targets, usamos eles para build, flash e rebuild.

Com o botão direito na pasta do projeto clique em "Build Targets -> Create...".

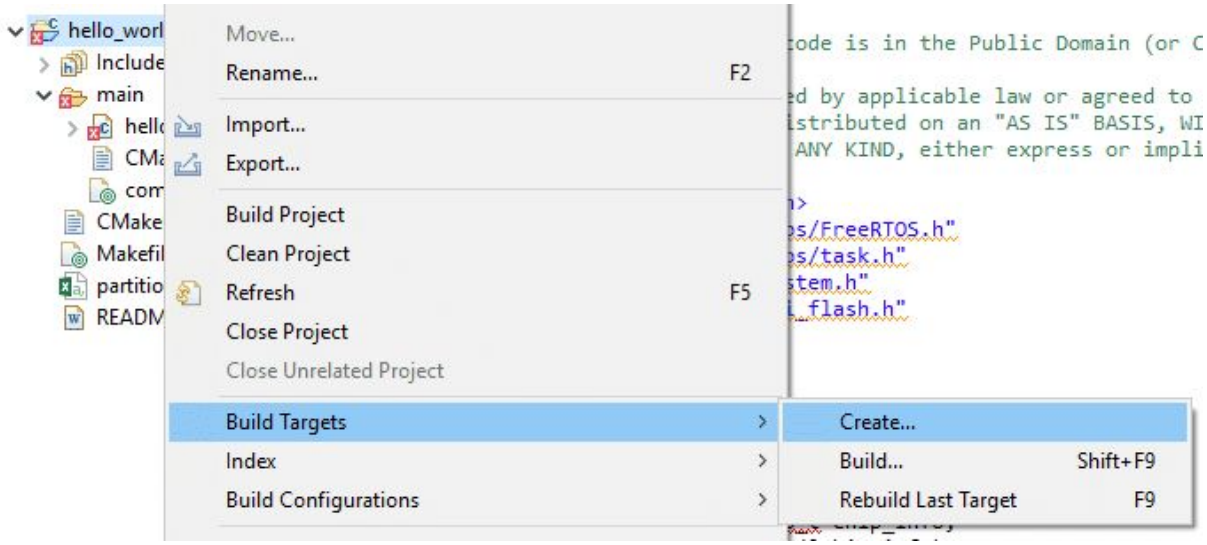


Figura 15: Criando targets.

No campo Target name preencha com "all" e pressione OK. Repita novamente o processo para os seguintes nomes: "clean" e "flash".

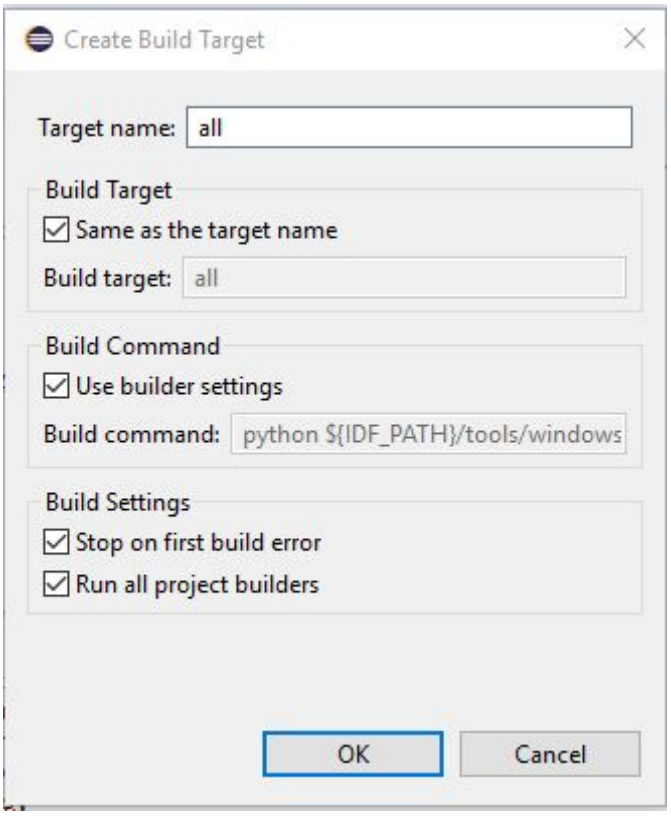


Figura 16: Criando target.

O resultado esperado deve ser como na figura abaixo.

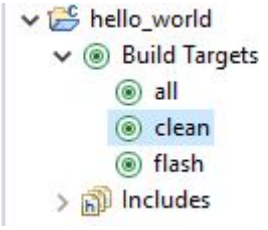


Figura 17: Lista de targets.

Vamos na pasta raiz da instalação do ambiente ESP IDF, a pasta msys32 e execute o aplicativo mingw32. Agora use o comando cd para adentrar no diretório da nossa pasta do projeto:

```
$ cd /c/msys32/ESP32/hello_world
```

Execute o comando:

```
$ make menuconfig
```

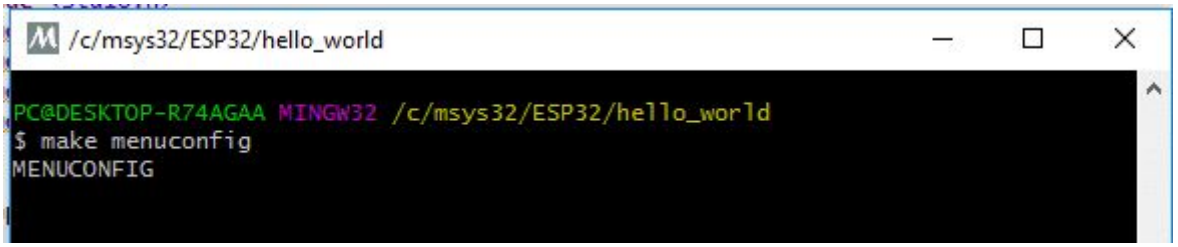


Figura 18: Acessando MENUCONFIG através do mingw32.

O resultado deve ser como na figura abaixo. Use o menu de navegação para ir até o serial flasher config.

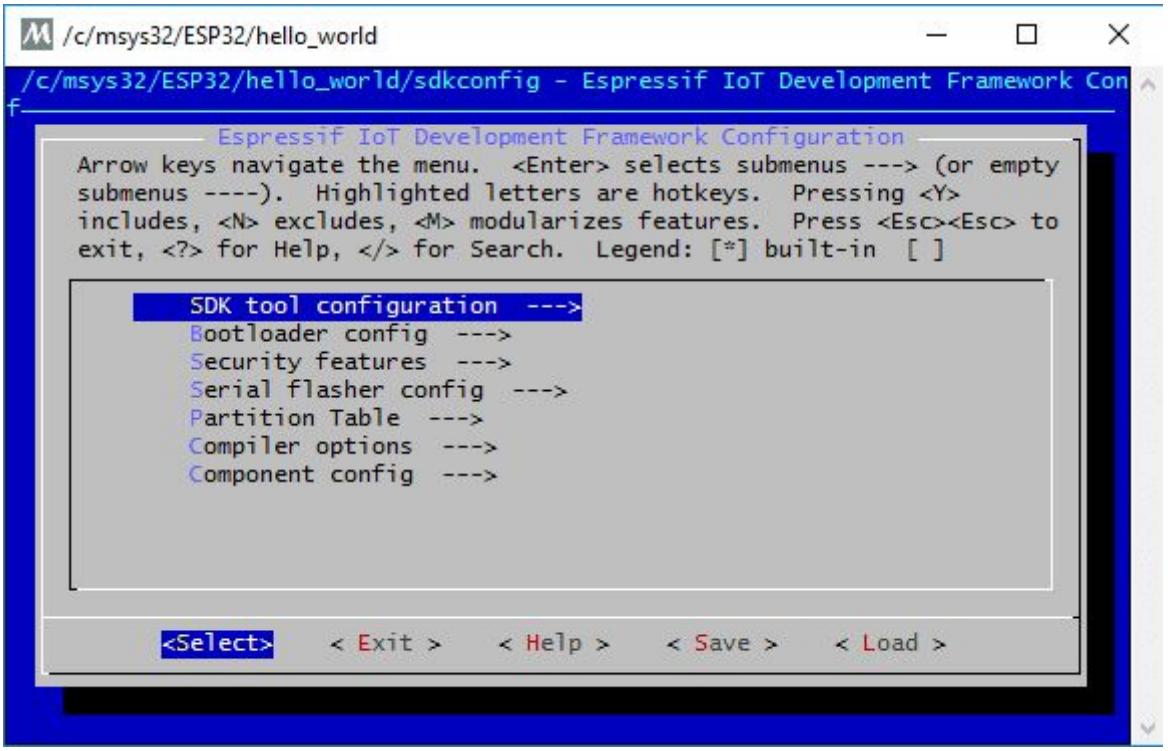


Figura 19: Acessando menu de configuração serial.

Preencha o campo default serial port com a porta em que está o seu dispositivo ESP32, no meu caso é a COM5. Salve e saia do menu.

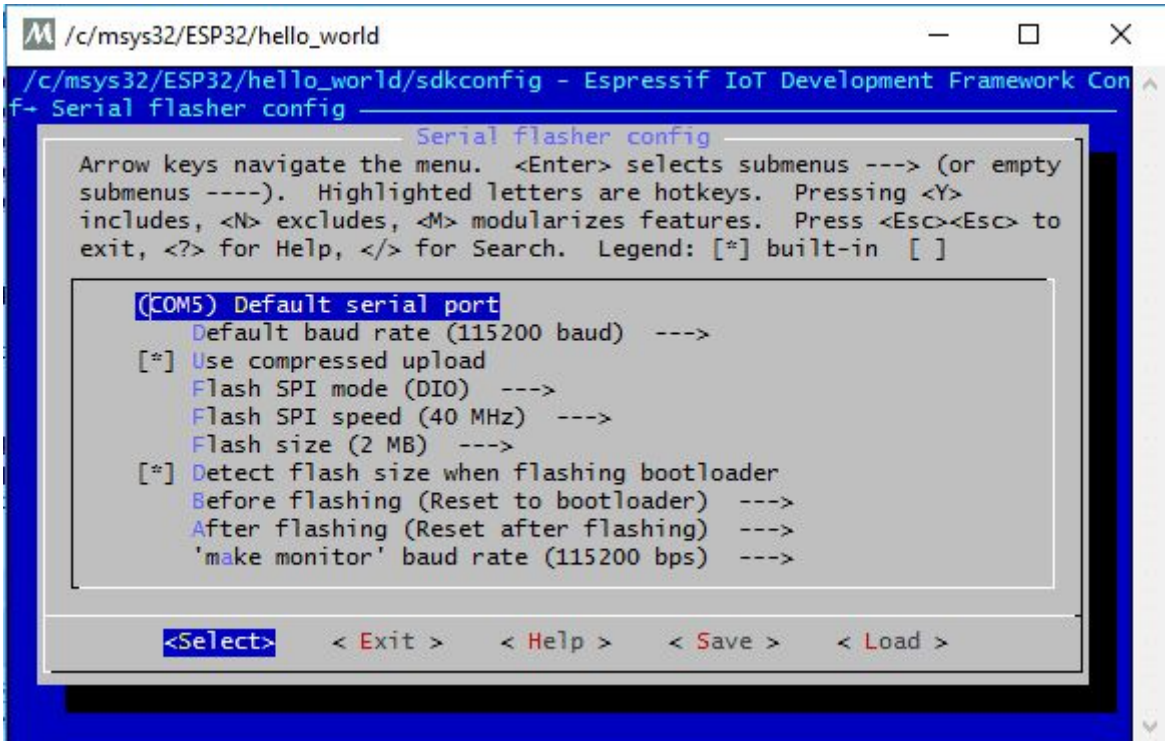


Figura 20: Configurando porta serial do ESP32.

Volte para o eclipse e nos build targets que criamos, clique em all duas vezes. A primeira execução pode demorar um pouco, mas em breve seu projeto estará compilado! Note que os erros que antes apareciam no main.c agora sumiram.

Para gravar o código na placa, clique no flash duas vezes.

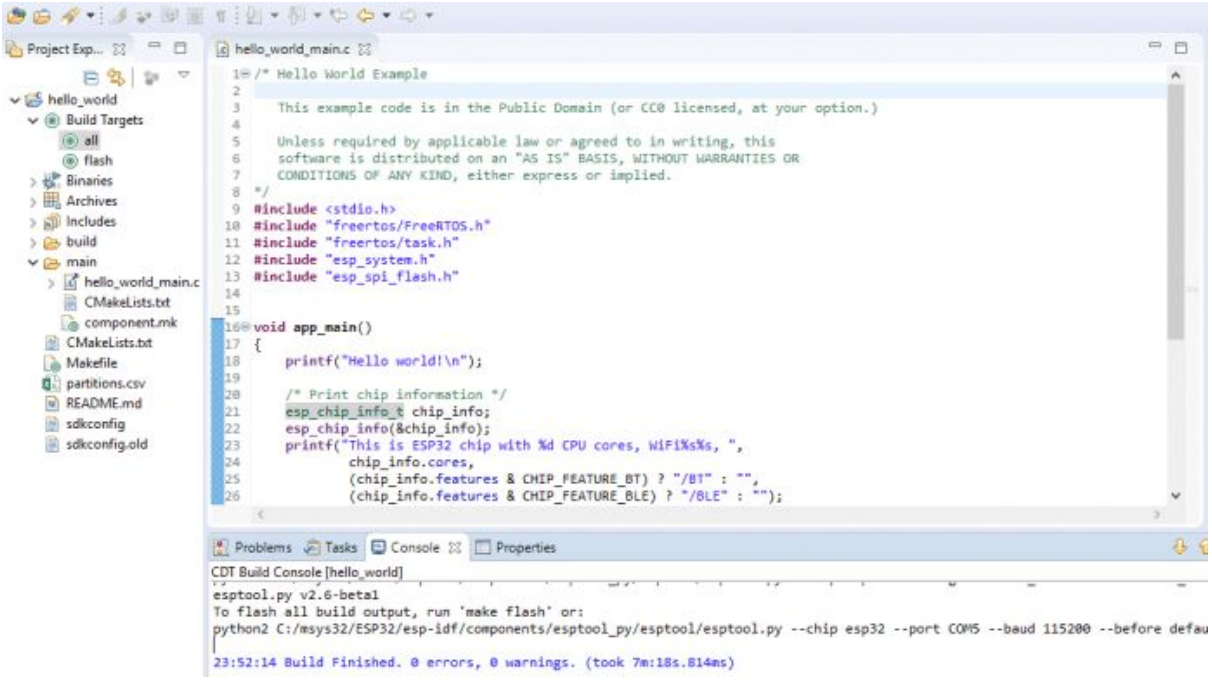


Figura 21: Projeto compilado sem erros.

Resolvendo possíveis erros na hora de compilar no eclipse:

Talvez você tenha tido o seguinte problema na hora de compilar:

*File "C:/msys32/sdk/esp-idf/components/esptool_py/esptool/esptool.py", line 24, inimport serialImportError: No module named serialmake[1]: *** [/sdk/esp-idf/components/esptool_py/Makefile.projbuild:49: /sdk/esp-idf/myexamples-01092016/02_blink/build/bootloader/bootloader.bin] Error 1make: *** [/sdk/esp-idf/components/bootloader/Makefile.projbuild:31: /sdk/esp-idf/myexamples-01092016/02_blink/build/bootloader/bootloader.bin] Error 2*

Isto pode estar relacionado à sua versão do python instalada ou a forma como ele é chamado. Uma solução que resolveu no meu caso foi alterar o nome do interpretador python de python para python2.

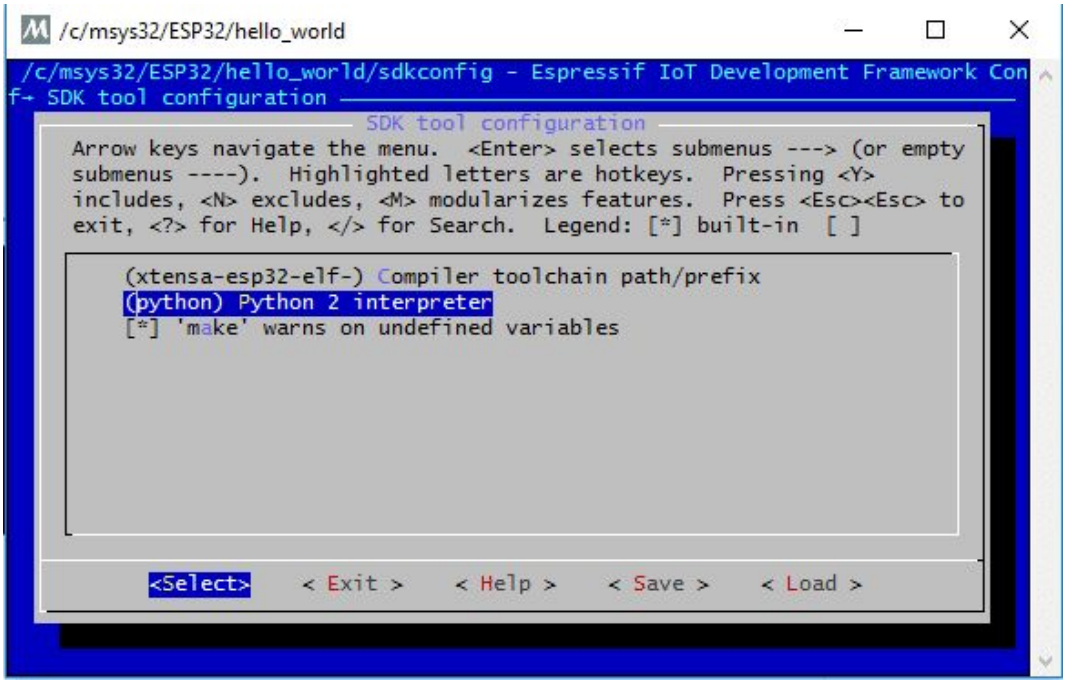


Figura 23: Modificando nome de chamada do interpretador python.

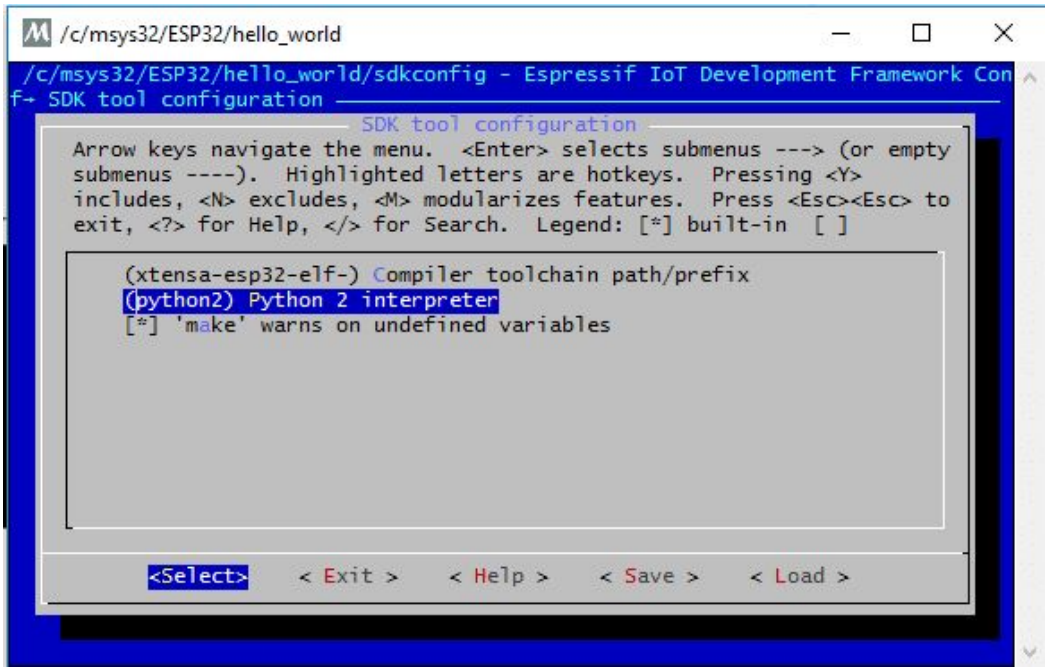


Figura 24: Nome de chamada do interpretador python modificado.

Bom, é isso galera! Espero que tenham gostado.

Referências

1.https://github.com/espressif/esptool/issues/171
2.https://docs.espressif.com/projects/esp-idf/en/latest/get-started/eclipse-setup.html

ESP32 - conhecendo os pinos de Strapping

Autor: Alexandre Fernandes dos Anjos

Projetos com esse poderoso microcontrolador da Espressif vem se tornando muito frequentes, oferecendo conectividade sem fio, Wi-Fi e Bluetooth, com custos muito competitivos, permitindo criar variadas aplicações para IoT. Vou trazer aqui neste artigo, detalhes sobre os pinos de strapping do ESP32, para evitar problemas nos seus projetos baseados nessa plataforma.

O que são os pinos de Strapping?

Os pinos de strapping definem algumas configurações do ESP32. Durante o curto processo de inicialização da CPU, é lido o estado lógico presente na entrada de cada um

desses pinos especiais e esses estados (0 ou 1) são por sua vez, escritos num registrador chamado ”GPIO_STRAPPING”.

Após o processo de inicialização, esses pinos voltam a funcionar com sua função normal.

Os bits deste registrador configura o modo de boot, a tensão presente no pino VDD_SDIO e outras funções iniciais que vemos na tabela abaixo:

Tensão do LDO interno (VDD_SDIO)					
Pino	Default	3,3V		1,8V	
GPIO_12	Pull-down	0		1	
Modo de Boot					
Pino	Default	SPI Boot		Download Boot	
GPIO_0	Pull-up	1		0	
GPIO_2	Pull-down	Não importa		0	
Habilita ou desabilita a saída de Log no pino de comunicação serial U0TXD					
Pino	Default	Habilita saída de Log		Desabilita saída de Log	
GPIO_15	pull-up	1		0	
Temporizações SDIO Slave					
Pino	Default	Falling-edge Sampling Falling-edge Output	Falling-edge Sampling Rising-edge Output	Rising-edge Sampling Falling-edge Output	Rising-edge Sampling Rising-edge Output
GPIO_15	Pull-up	0	0	1	1
GPIO_5	Pull-up	0	1	0	1

Figura 1 - Tabela funções dos pinos de strapping

O ESP32 têm cinco pinos de strapping: GPIO_12, GPIO_0, GPIO_2, GPIO_15 e GPIO_5.

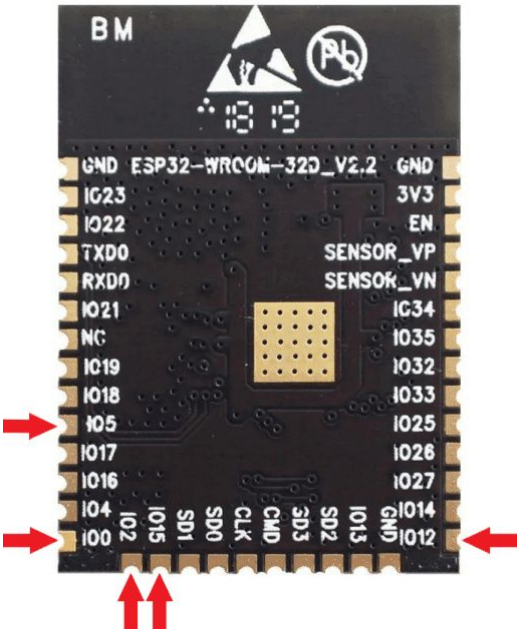


Figura 2 - Destaque dos pinos no módulo ESP32-WROOM-32

Cada pino de strapping é conectado internamente a um resistor de pull-up ou pull-down interno durante a inicialização do chip (ver coluna “Default” na tabela da Fig.01). Consequentemente, se o pino está desconectado ou o circuito externo conectado é de alta impedância, o pull-up/pull-down interno determina o nível de entrada default dos pinos de strapping. Para alterar os valores default, o usuário pode aplicar as resistores externos para alterar o estado lógico da entrada.

Entendendo as funções dos pinos de strapping

Vamos entender o que exatamente cada pino de strapping faz, seguindo a tabela da Fig.01:

GPIO_12: Tensão do LDO interno do chip: Se este pino estiver desconectado (pull-down interno) ou com um nível lógico 0 na sua entrada durante a inicialização, o pino VDD_SDIO, que é uma saída de tensão de alimentação externa do chip, usado para alimentação de periféricos, terá tensão de 3V3 na sua saída. Agora se tiver com nível lógico 1, sua saída será de 1V8.

GPIO_0 e GPIO_2: Modo de Boot: Se ambos os pinos estiverem desconectados ou GPIO_0 em nível lógico 1 na sua entrada, o ESP32 irá executar o firmware gravado na memória FLASH pelo barramento SPI. Para entrar em modo de gravação do ESP32 (download boot) ambos pinos devem estar em nível lógico 0.

Esses três pinos de strapping acima são os que mais causam problemas. Vamos entender o porquê logo mais.

GPIO_15: Log de inicialização na saída do pino de comunicação serial U0TXD: Se este pino estiver desconectado (pull-up interno) ou com um nível lógico 1 na sua entrada, teremos na saída do pino U0TXD, que é referente à comunicação serial do chip, o Log de inicialização do chip, com diversas informações relevantes. Em nível lógico 0 esse Log é desabilitado.

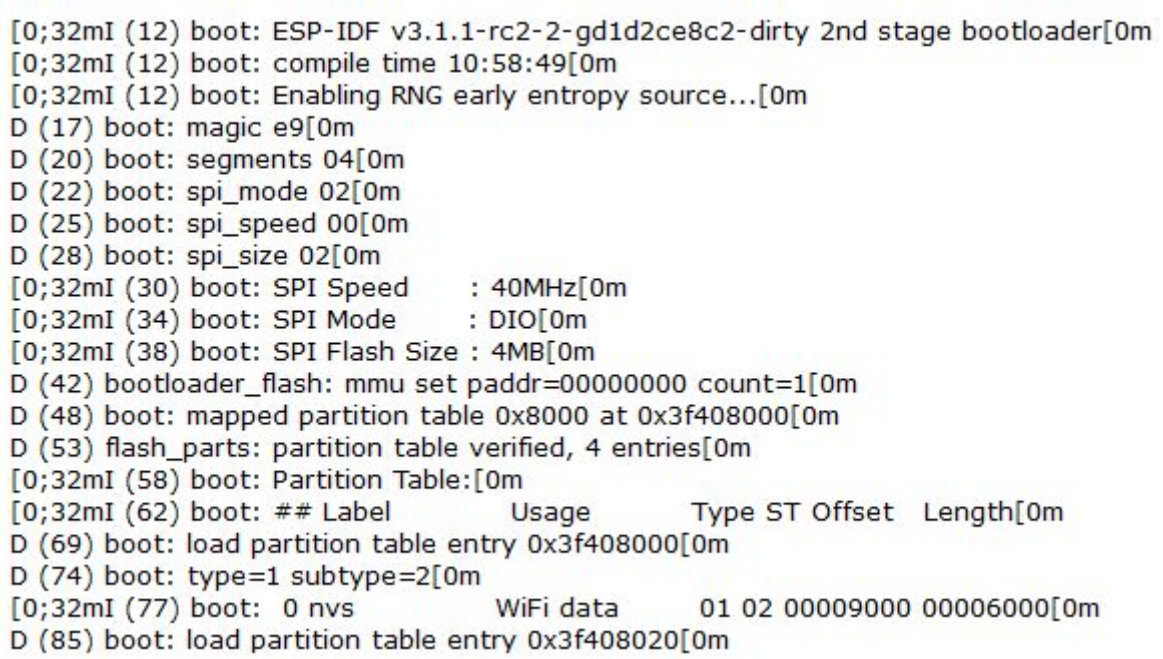


Figura 3 - Trecho de Log de inicialização no pino U0TXD

GPIO_5 e GPIO_15: Temporizações do SDIO Slave: Esse pinos definem as características da comunicação do chip com periférico SD card. Se ambos estiverem desconectados (pull-up interno) ou nível lógico 1, a comunicação será do tipo Rising-edge Sampling / Rising-edge Output.

Enfim, os problemas

Os problemas começam quando esquecemos desses pinos de strapping e de seus estados lógicos quando estamos definindo os GPIOs no esquemático do projeto.

Veja o esquemático abaixo, definimos o barramento I2C nos GPIO_12 e GPIO_13:

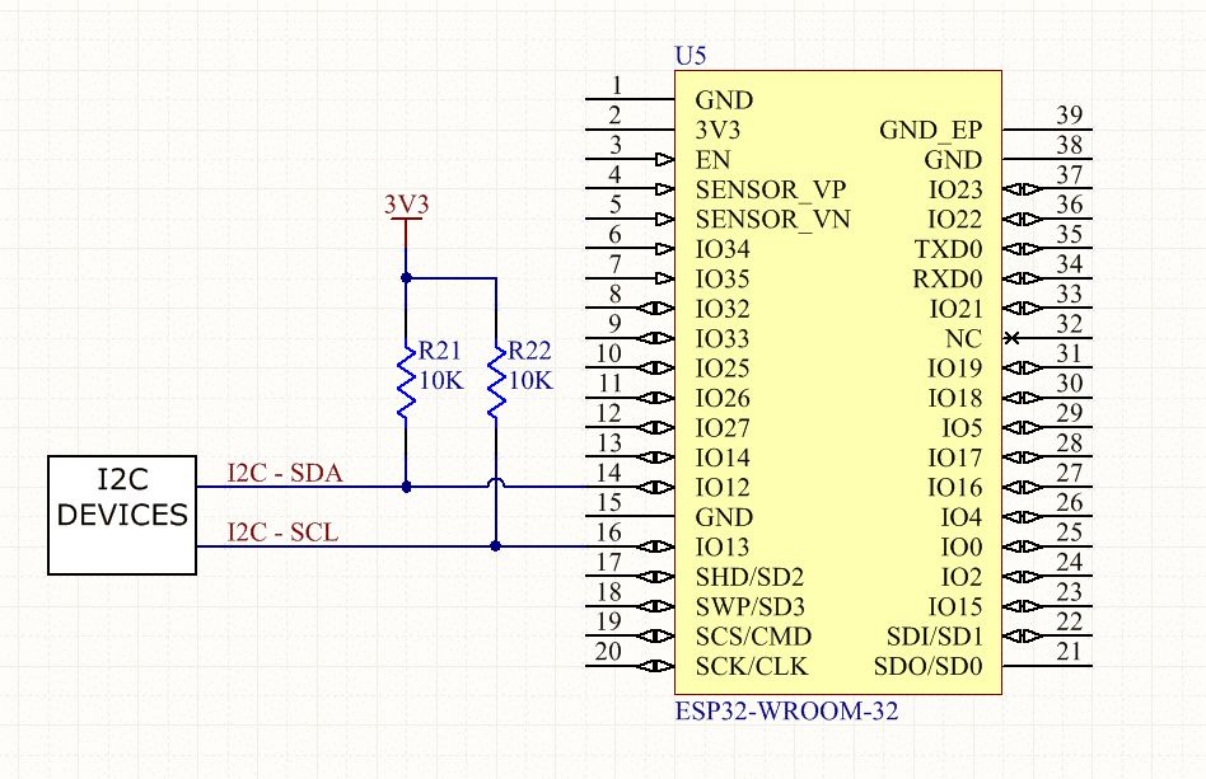


Figura 4 - Barramento I2C

Esquecemos que o GPIO_12 (definido como SDA do I2C) é um pino de strapping, roteamos e montamos a placa do projeto dessa forma. O que vai acontecer ao ligar a placa?

- Simplesmente nada vai funcionar: nada na saída serial, nada de gravação, nada de boot.

Sabemos que existe necessidade de resistores de pull-up no barramento de comunicação I2C para que o mesmo funcione adequadamente. Vide R21 e R22 no esquemático.

Colocando esses resistores para que o I2C funcione, colocamos o GPIO_12 em nível lógico 1... Aí está o problema!!!

Com o nível lógico em 1 definido durante a inicialização do ESP32, a memória FLASH interna ao módulo que tem a tensão de alimentação correta de 3V3, passou a ser alimentada com 1V8. Abaixo da tensão de trabalho recomendada.

A Fig.05 abaixo mostra como é ligado a memória FLASH internamente ao módulo. A alimentação da memória (pino 8 - VCC) provêm do pino VDD_SDIO, que tem sua tensão de saída definido pelo GPIO_12 durante a inicialização.

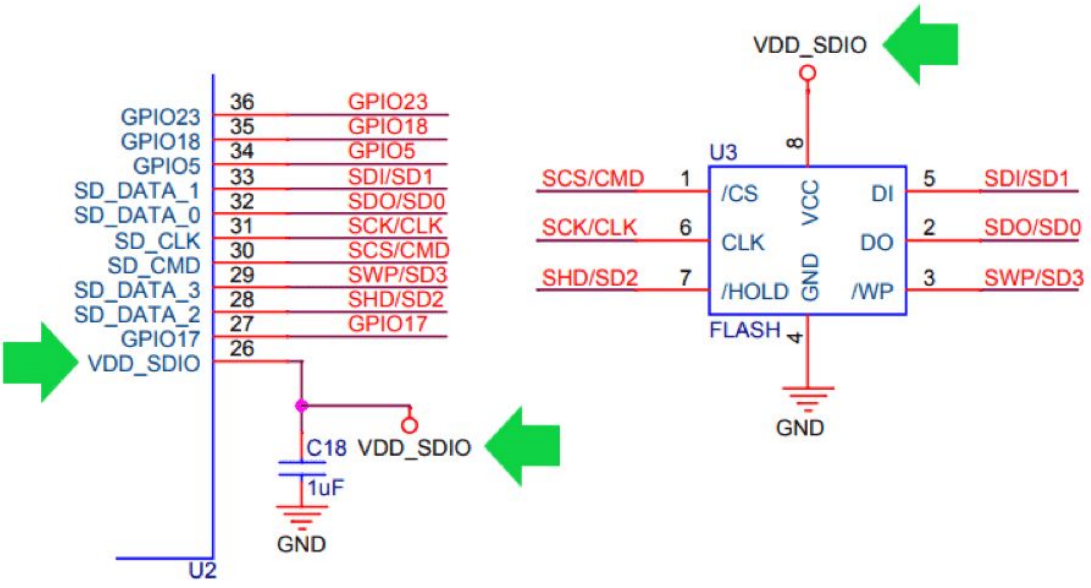


Figura 5 - Ligação memória FLASH interno ao módulo

Tem até uma nota no datasheet sobre essa situação (MTDI é o GPIO_12):

Note:

- Firmware can configure register bits to change the settings of "Voltage of Internal LDO (VDD_SDIO)" and "Timing of SDIO Slave" after booting.
- The module integrates a 3.3 V SPI flash, so the pin MTDI cannot be set to 1 when the module is powered up.

Figura 6 - Nota no datasheet do ESP32

Outra situação:

Um simples botão ligado ao GPIO_2. Um resistor de pull-up e o botão ligado ao GND. Vemos isso em diversos projetos. Veja na Fig.07 abaixo:

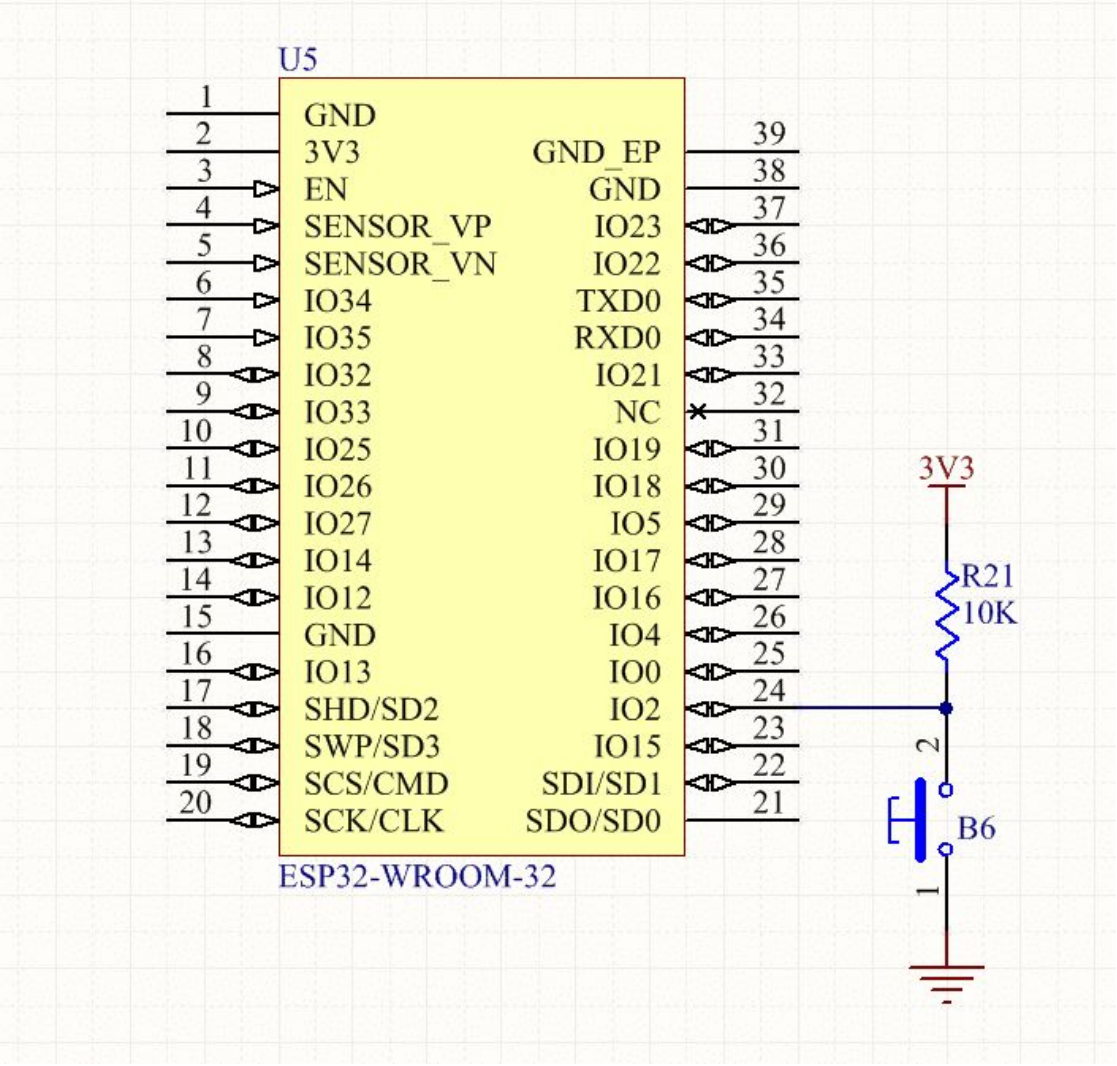


Figura 7 - Ligação botão ao GPIO_2

Mais uma vez teremos problemas, dessa vez o ESP32 não entrará em modo gravação. O GPIO_2 define o modo de boot e para entrar em modo download (gravação), ambos

pinos, GPIO_2 respectivamente e também o GPIO_0 devem estar em nível lógico 0. E nesse caso o GPIO_2 está permanentemente em nível lógico 1 através do resistor de pull-up R21.

Essa outra situação na Fig.08 não traz tantos problemas funcionais, mas pode provocar estranheza a quem espera a saída do Log de inicialização sendo printado na serial do chip.

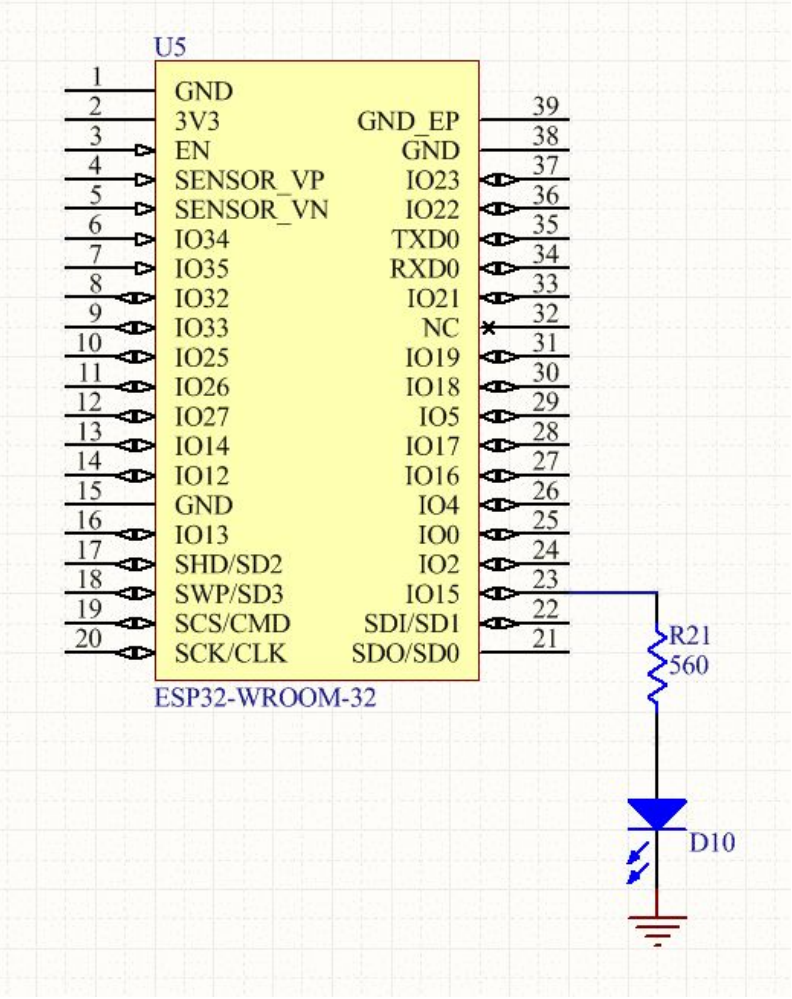


Figura 8 - LED ligado ao GPIO_15

Definimos no programa o GPIO_15 como saída para ativação de um LED.

Durante a inicialização do ESP32, o GPIO_15 está sendo ligado ao GND através do resistor R21 e do LED D10 polarizado diretamente. Com o GPIO_15 em nível lógico 0 temos o desabilitação do Log de saída.

Conclusão - ESP32 - conhecendo os pinos de Strapping

O datasheet é sempre o melhor amigo de um desenvolvedor de hardware. A regra de ouro que fica aqui é a leitura atenciosa do datasheet e dos outros documentos pertinentes, sempre quando começar a trabalhar com um novo dispositivo (Ah, isso inclui os documentos de Erratas também!). Atente-se aos detalhes e as notas do fabricante. Faça uma lista dos pontos importantes a serem observados durante o projeto (check-list).

Isso ajudará a evitar erros, mas caso algum passar, ajudará para um debug mais rápido.

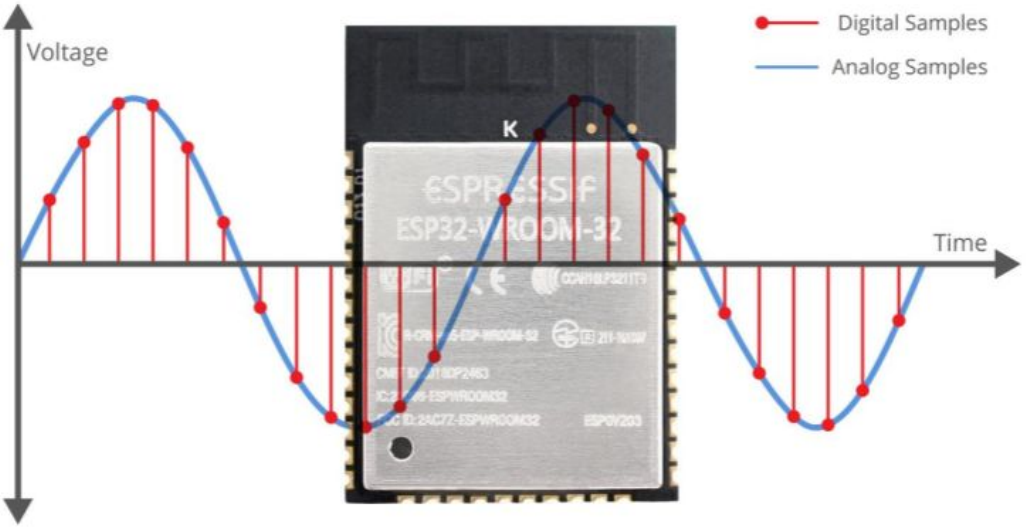
Referências

1.https://www.espressif.com/en/support/download/documents/chips
2.https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
3.https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf

Publicado originalmente no Embarcados, no dia 18/02/2020: link para o artigo original, sob a Licença Creative Commons Atribuição-CompartilhaIgual 4.0 Internacional.

ESP32 - Analisando e corrigindo o ADC interno

Autor: José Morais



Talvez você já tenha visto em algum lugar que o ADC do ESP32 não é linear ou tem muito erro nas leituras, mas o que isso significa na prática? Como podemos resolver? Quem irá

nos ajudar? É o que vamos analisar e tentar corrigir neste artigo dedicado ao ADC do ESP32.

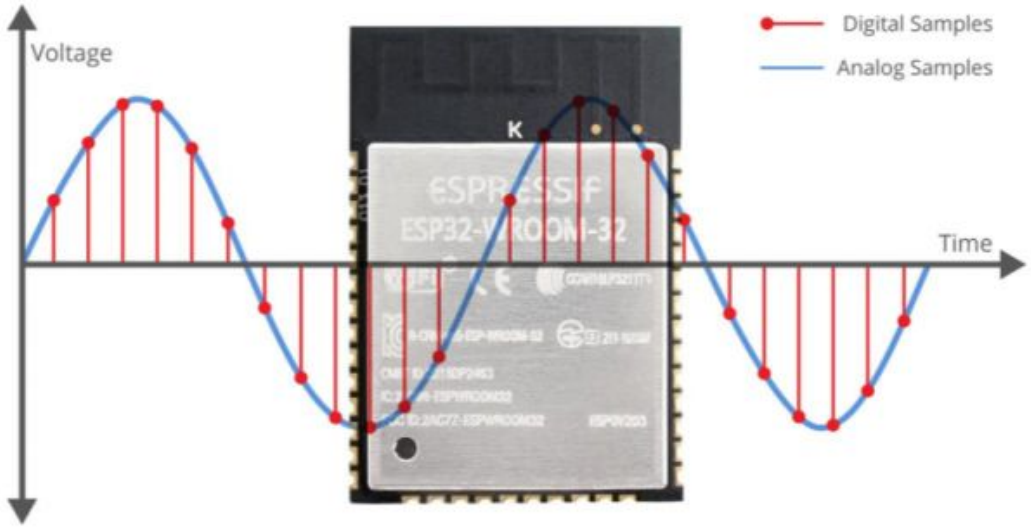


Figura 1 - ESP32 ADC.

ESP32 e seu peculiar ADC

Depois de muitas reclamações a respeito do ADC no ESP8266, principalmente sobre conter apenas 1 canal (1 V, 10 bits, SAR ADC) para essa tarefa, a Espressif ouviu os seus usuários. No ESP32 incluiu 18 canais (1,1 - 3,9 V, 9 - 12 bits, SAR ADC), que foi separado em 2 controladores, ADC1 (8 canais) e ADC2 (10 canais). Isso nos permite, por exemplo, ler 2 canais paralelamente ou até com DMA para protocolos de áudio e vídeo como I2S, um baita avanço!

Alguns detalhes curiosos do ADC no ESP32

- Ambos controladores podem ser usados no domínio digital e RTC, que são focados em velocidade (2 M Samples/Seg) e baixo consumo (200 K Samples/Seg), respectivamente;
- Resolução configurável: 9, 10, 11 e 12 bits;
- Tensão máxima configurável (atenuação interna): 0 dB (1,1 V), 2,5 dB (1,5 V), 6 dB (2,2 V), 11 dB (3,9 V limitado pelo VDD_A);
- Faixas de tensão recomendadas para melhor precisão:
    - 0dB: 100 - 950mV;
    - 2.5dB: 100 - 1250mV;
    - 6dB: 150 - 1750mV;
    - 11dB: 150 - 2450mV.
- O ADC2 não pode ser usado enquanto o Wi-Fi estiver ligado, pois o canal é compartilhado com o driver do módulo de comunicação sem fio. Um meio para contornar isso seria desligar o WiFi durante a medição e depois ligá-lo novamente;
- As medições do ADC ficarão mais ruidosas enquanto WiFi estiver ligado, muitas vezes por conta de má alimentação e/ou filtragem do sinal;
- ULP permite leitura ADC mesmo durante deep sleep;
- Cada ESP32 pode ter até 6 % de diferença nas leituras, boa parte devido à tensão de referência (Vref) do ADC haver grande variação (1000 - 1200 mV). Essa variação na Vref ocasiona leituras diferentes entre os chips, veja na figura 1 abaixo a comparação de 2 ESP32 com Vref diferentes.

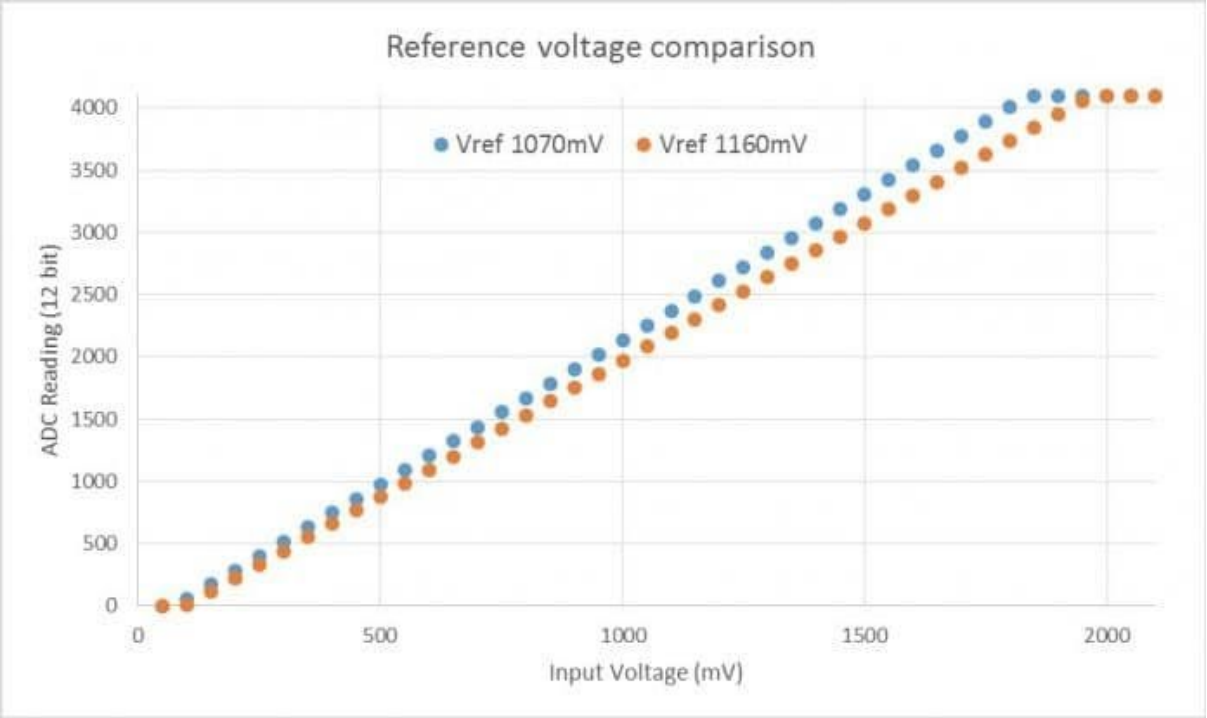


Figura 2 - Leituras com Vref diferentes.

Testando o ADC

Agora que já sabemos um pouco mais sobre os detalhes do ADC, vamos fazer alguns testes na prática para analisar suas curvas.

As medições foram feitas com média simples de 100 amostras intervaladas por 30 us cada, sem capacitores ou qualquer método além da média simples para filtragem. Em todos os testes, vamos adotar uma leitura com o Vref padrão (“ADC Descal”) e outra com a API de calibração utilizando o Vref verdadeiro (“ADC Cal”), que vem gravado na memória dos ESP32 mais novos, fabricados de 2018 para frente. Após os testes, vamos entender como utilizar a API de calibração disponibilizada pela Espressif.

A fórmula utilizada nas medições sem a calibração foi:

$$\left(\frac{Vm}{Resolução}\right) * ADC$$

Vm: 1,1 ou 3,3 V.

Resolução: 1024 ou 4096.

ADC: Valor lido do pino.

Teste A

Nesse teste (figuras 3 e 4), vamos analisar o ADC1 (GPIO36) e seu erro com 0 dB para as seguintes opções:

- Wi-Fi OFF;
- 10 e 12 bits;
- 0 dB (0 - 1100 mV);
- Vref ADC Descal: 1100 mV;
- Vref ADC Cal: 1072 mV.

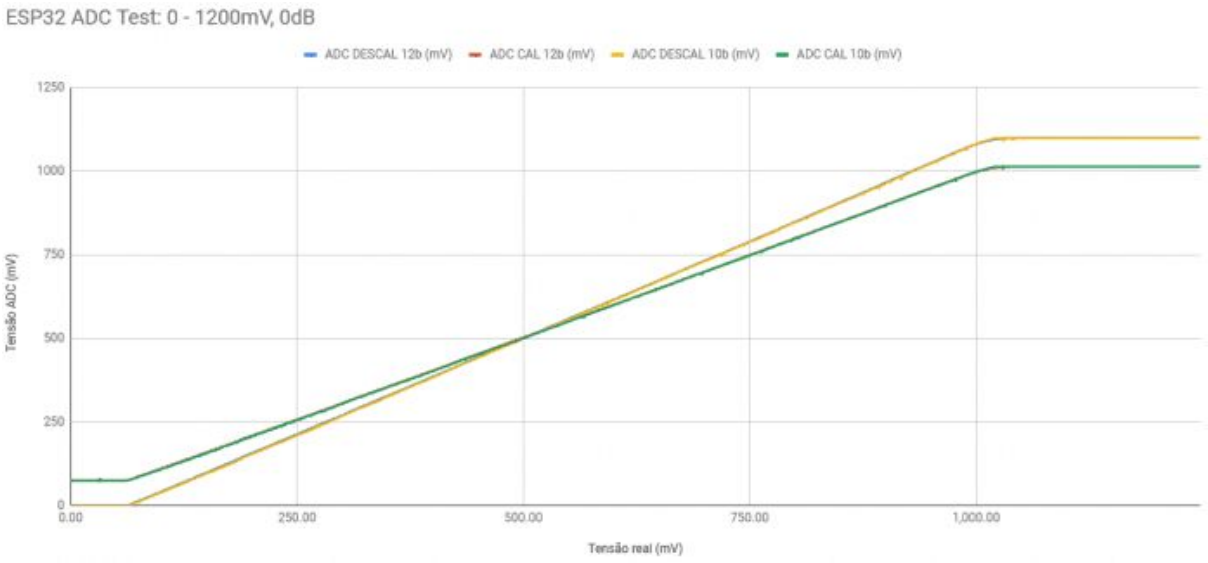


Figura 3 - Curvas 0dB.

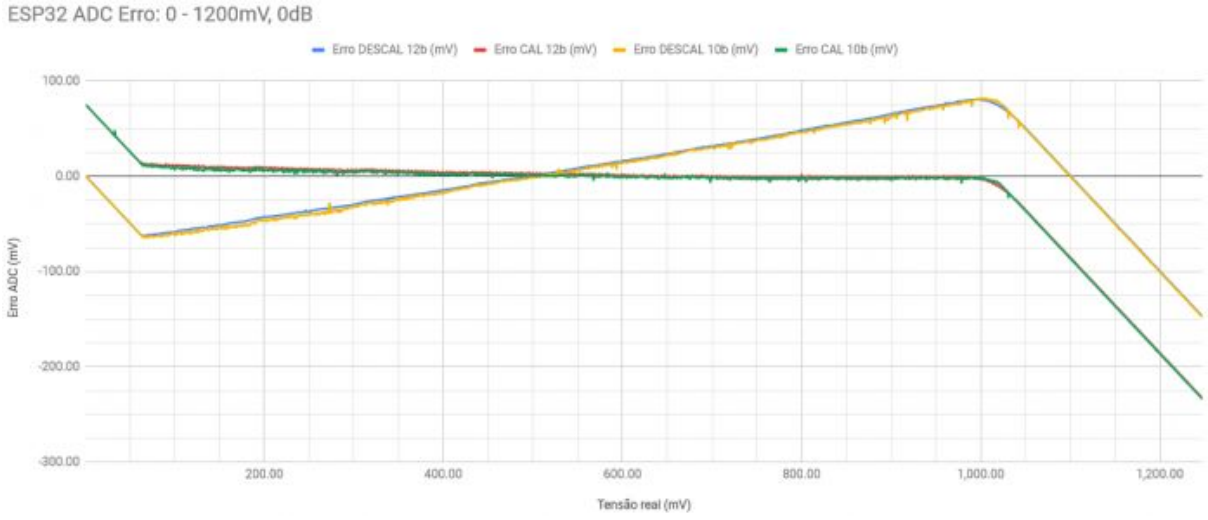


Figura 4 - Erro das curvas 0dB.

Observando as figuras 3 e 4 que mostram a leitura e erro do ADC para “0 dB, 10 e 12 bits com Wi-Fi OFF”, podemos ver que a curva utilizando a API de calibração se manteve com erro <= 5 mV em toda faixa de tensão recomendada, tendendo a 0 mV. As leituras de 10 e 12 bits foram praticamente idênticas, não havendo alguma discrepância considerável na

ordem dos mV, mas caso você precise de leituras mais precisas e consistentes, refaça os testes com sua placa e análise os resultados.

Apesar do erro com a calibração tendendo a ~0 mV na faixa de tensão recomendada pela Espressif (100 - 950 mV), fora desses limites, o erro é grande e pode ser totalmente inaceitável, descartando essas leituras em muitos projetos.

Teste B

Nesse teste (figuras 5 e 6), vamos analisar o ADC1 (GPIO36) e seu erro com 0 dB para as seguintes opções:

- WiFi ON;
- 10 e 12 bits;
- 0dB (0 - 1100 mV);
- Vref ADC Descal: 1100 mV;
- Vref ADC Cal: 1072 mV.

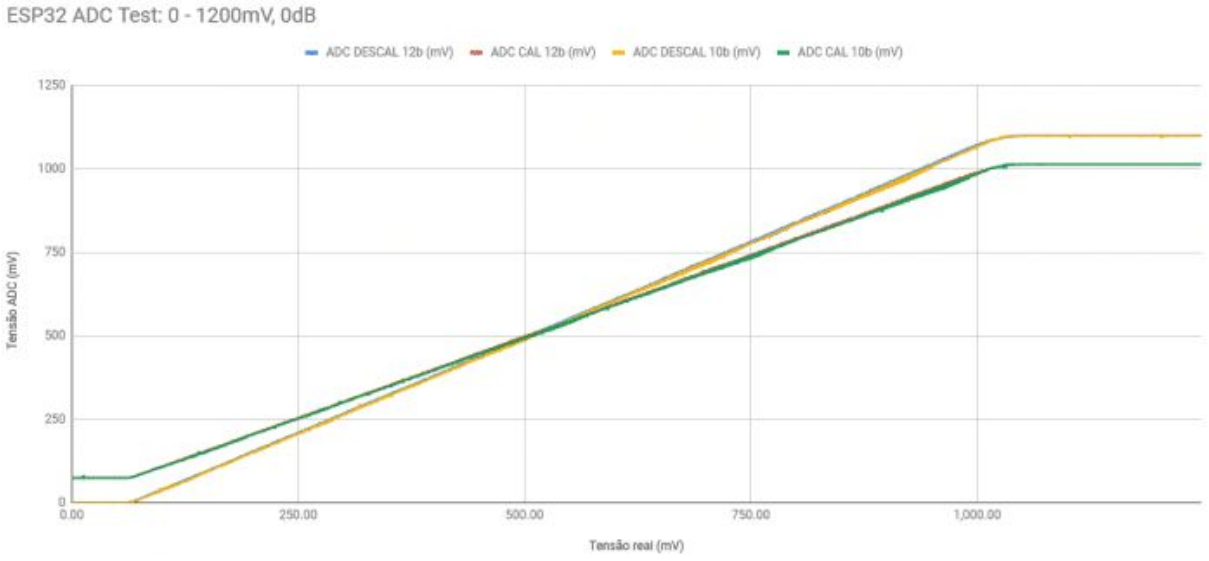


Figura 5 - Curvas 0dB.

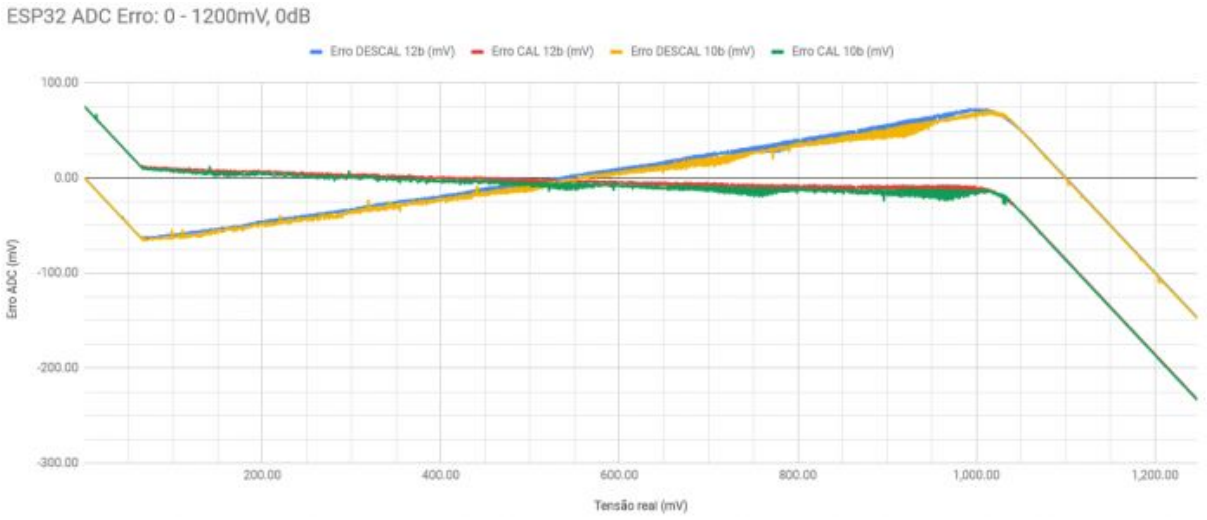


Figura 6 - Erro curvas 0dB.

Observando as figuras 5 e 6 que se diferenciam do teste A apenas pelo Wi-FI ON, podemos ver que os erros na medição calibrada, próximo ao fim, foi maior que com o Wi-Fi OFF. Além disso, em toda a faixa de medição, houve ruído perceptível no gráfico, podendo haver necessidade de filtros analógicos ou digitais melhores.

Agora vamos mudar um pouco as coisas, onde a maioria que programa pela Arduino IDE reside. Os dois testes a seguir serão com a atenuação interna de 11 dB, que permite a leitura de 0 - 3,9 V (limitado pelo VDD_A), padrão na Arduino IDE. A própria Espressif diz que com 11 dB perde a linearidade enquanto 0 dB não, então vamos observar.

Teste C

Nesse teste (figuras 7 e 8), vamos analisar o ADC1 (GPIO36) e seu erro com 11 dB para as seguintes opções:

- WiFi OFF;
- 10 e 12 bits;
- 11 dB (0 - 3900 mV);
- Vref ADC Descal: 1100 mV;
- Vref ADC Cal: 1072 mV.

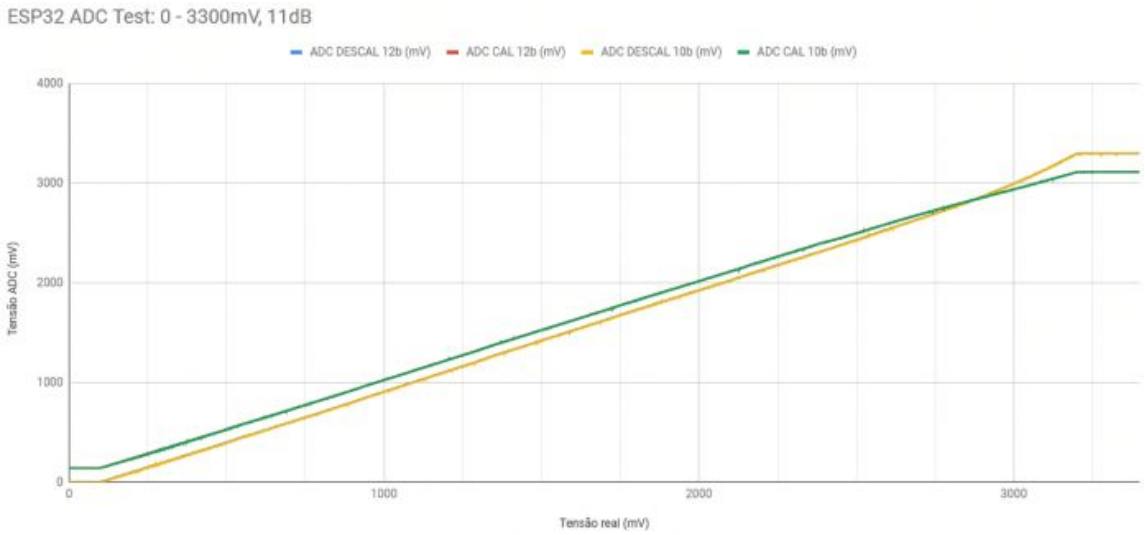


Figura 7 - Curvas 11dB.

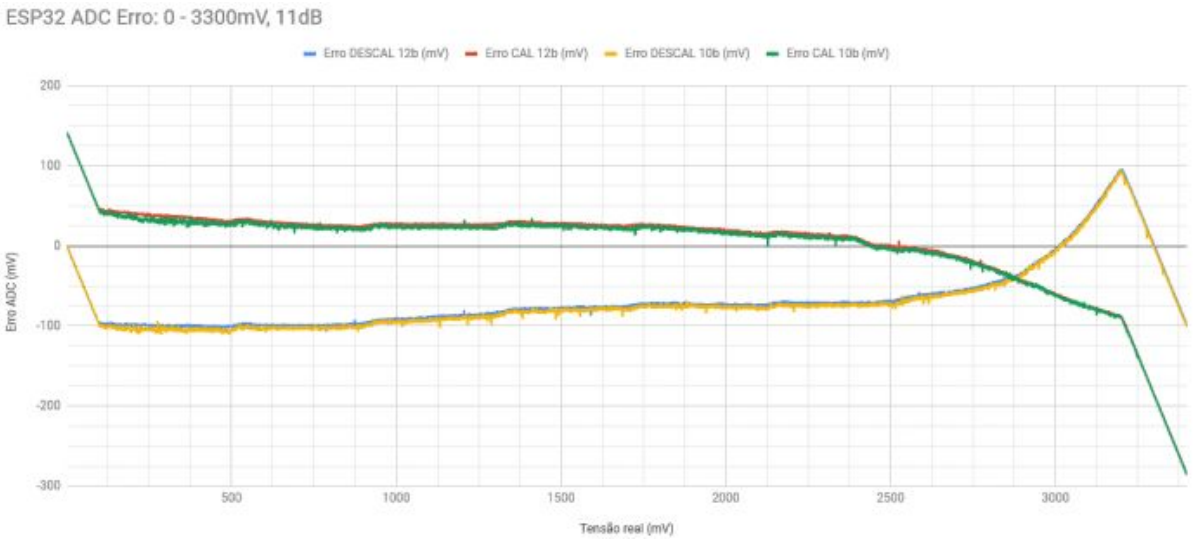


Figura 8 - Erro curvas 11dB.

Podemos perceber que os erros aumentaram e também ficou bem menos linear, principalmente após 2 V. Mesmo utilizando a calibração com Vref verdadeiro, não foi o suficiente para aproximar o erro de ~0 mV, ficando próximos a ~25 mV na faixa recomendada.

Teste D

Nesse teste (figuras 9 e 10), vamos analisar o ADC1 (GPIO36) e seu erro com 11 dB para as seguintes opções:

- WiFi ON;
- 10 e 12 bits;
- 11 dB (0 - 3900 mV);
- Vref ADC Descal: 1100 mV;

- Vref ADC Cal: 1072 mV.

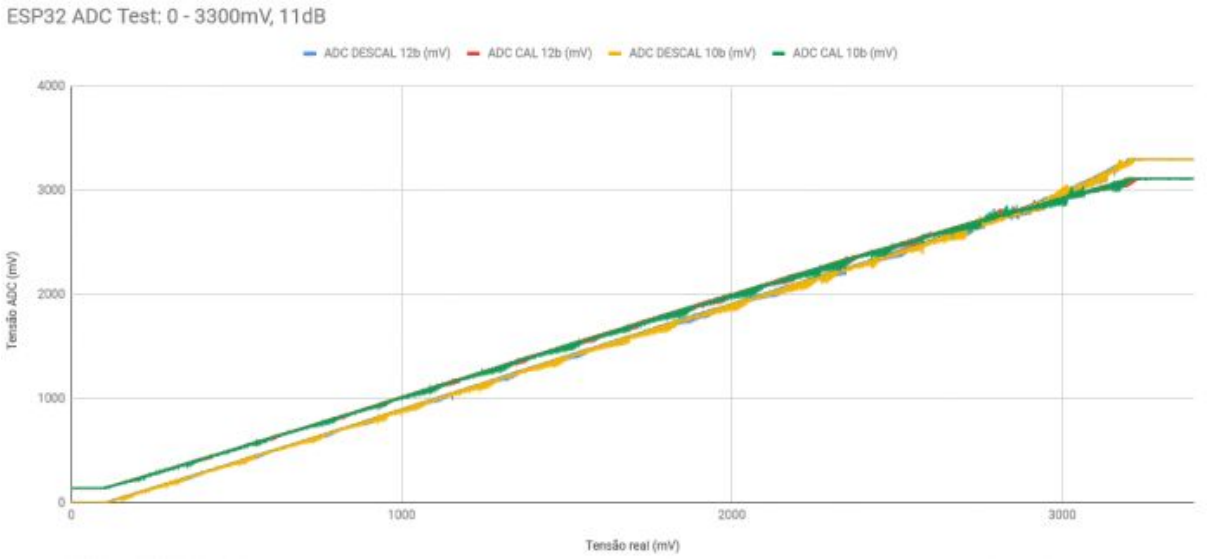


Figura 9 - Curvas 11dB.

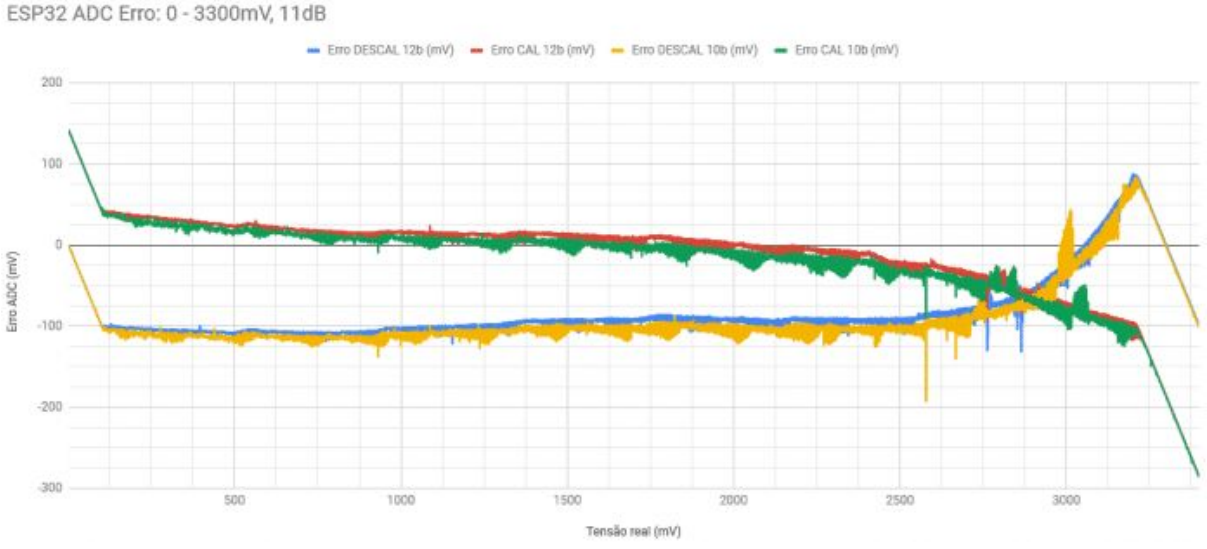


Figura 10- Erro curvas 11dB.

Novamente, repetimos o mesmo teste que o anterior, mas com o Wi-Fi ON. As curvas permanecem parecidas, entretanto, o ruído gerado pelo WiFi é bem notável, sendo até desaconselhável usar o ADC com WiFi ON em produtos que precise de leituras estáveis.

Algumas medidas para tentar melhorar a leitura é usar um atenuador externo como divisor resistivo em 0 dB e adicionar filtros analógicos/digitais.

API de calibração

A Espressif, após tantas reclamações dos usuários, começou a disponibilizar uma biblioteca (API) para calibração do ADC com tabelas para comparação que também utilizam dois métodos para afinar a leitura entre os chips, que vêm gravados na memória (eFuse) dos ESP32 mais recentes fabricados de 2018 em diante:

- Vref: A calibração com Vref representa o valor verdadeiro do Vref naquele ESP32. Quando disponível, é utilizado pela API ao invés do padrão (veremos mais a seguir);
- Two Point: A calibração de dois pontos tem as leituras do ADC1 e ADC2 para 150 mV e 850 mV naquele ESP32. Se disponível, este método tem mais prioridade que o Vref quando utilizado pela API.

A API do ADC faz todo o trabalho de compensar as curvas de acordo com estes e outros valores gravados na memória. Mais detalhes nas referências no fim do artigo. Se ainda não for o suficiente, você pode efetuar suas próprias análises e calibrações individuais das placas com alguma fórmula de correção, como a Espressif fez em sua linha de montagem, porém, de forma automatizada.

E se o meu ESP32 for anterior a 2018, vou poder utilizar essa calibração? Sim, é o que vamos testar agora, alterando as tensões de referência para os máximos, padrão e verdadeiro.

Teste E

Nesse teste (figuras 11 e 12), vamos analisar o ADC1 (GPIO36) e seu erro com 0 dB para as seguintes opções:

- WiFi OFF;
- 12 bits;
- 0 dB (0 - 1100 mV);
- Vref: 1000, 1100, 1200, 1072 mV.

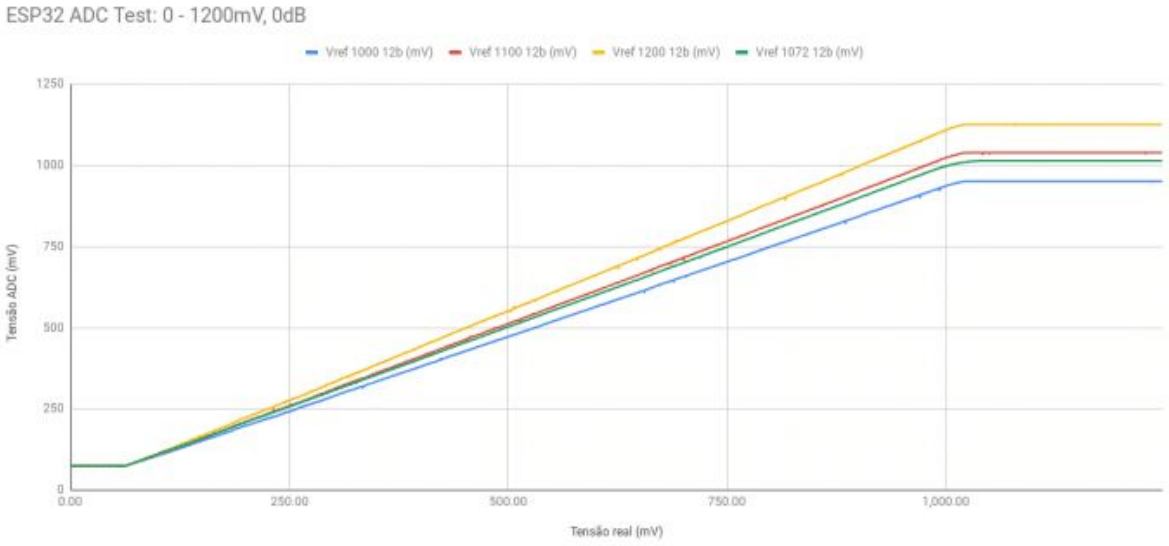


Figura 11 - Curvas de Vref 0dB.

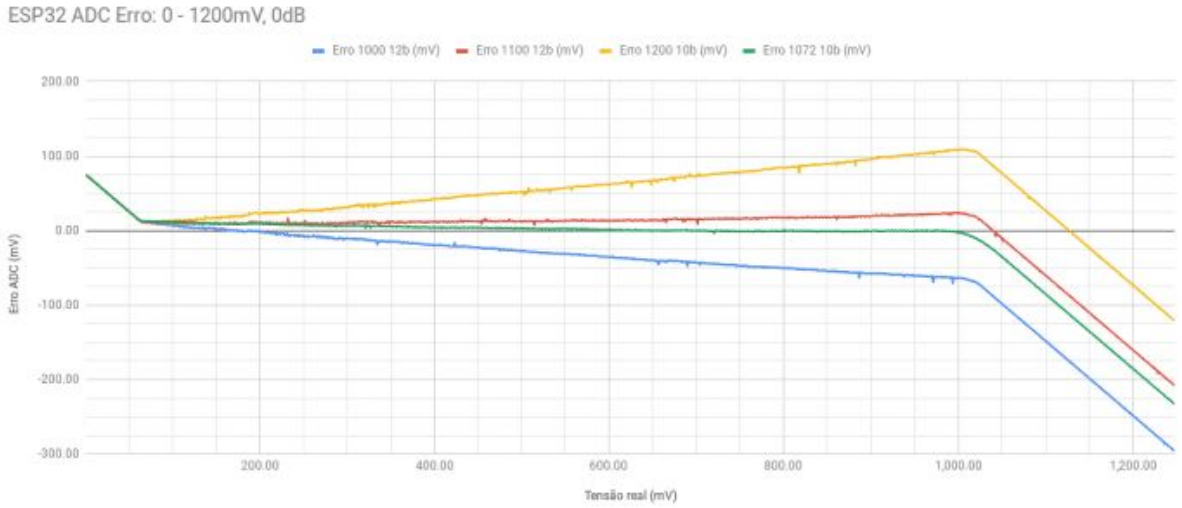


Figura 12 - Erros de curvas Vref 0dB.

Observando as figuras 10 e 11, podemos perceber que há grande variação assumindo os piores casos (1000 e 1200 mV). Apenas para a curva de 1072 mV que foi utilizado o Vref gravado na memória, sendo o restante definido no código. Qualquer uma dessas curvas ainda é melhor do que se utilizar o método sem calibração como visto nas figuras 2, 3, 4 e 5.

Teste F

Nesse teste (figuras 13 e 14), vamos analisar o ADC1 (GPIO36) e seu erro com 11 dB para as seguintes opções:

- WiFi OFF;
- 12 bits;
- 11 dB (0 - 3900 mV);
- Vref: 1000, 1100, 1200, 1072 mV.

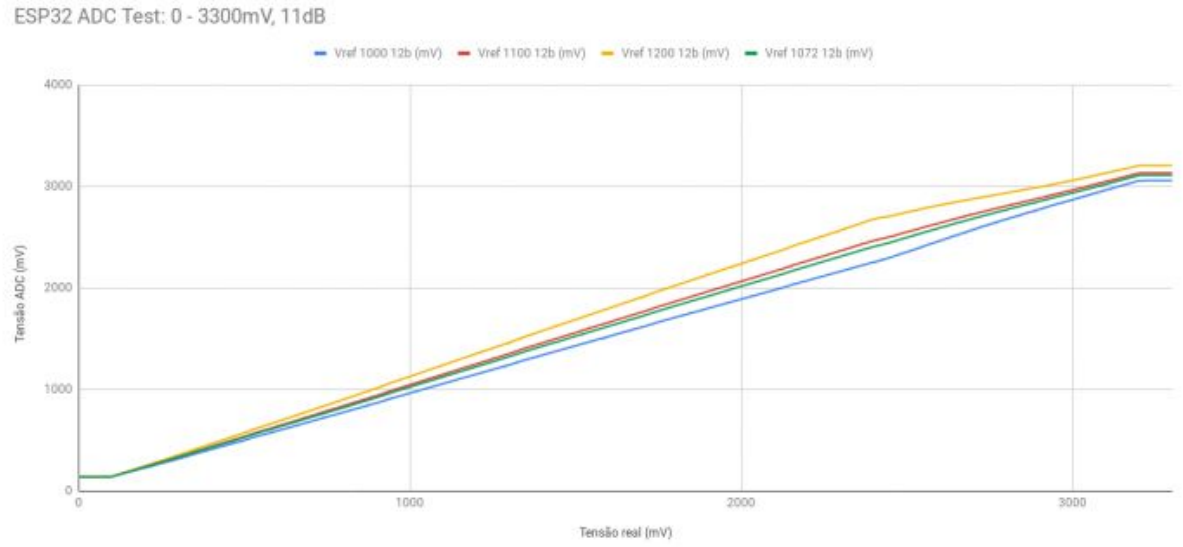


Figura 13 - Curvas de Vref 11dB.

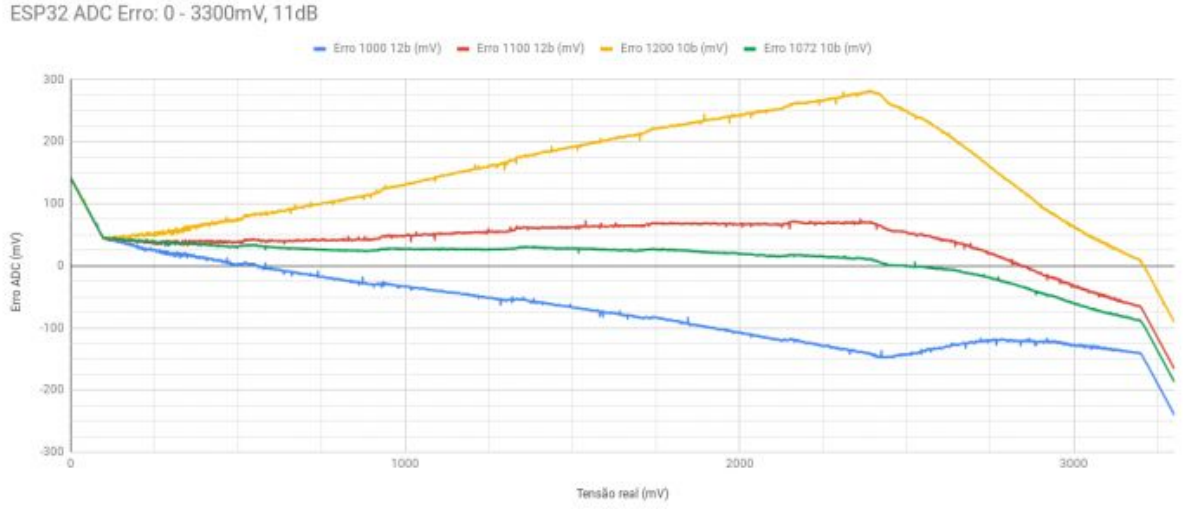


Figura 14 - Erro de curvas Vref 11dB.

Observando as figuras 13 e 14, os erros com Vref 1200 mV chegam até próximo aos 300 mV, sendo totalmente descartável em muitos produtos. Mesmo assumindo o Vref verdadeiro, não foi capaz do erro tender a ~0 mV como em 0 dB. Novamente, qualquer

uma dessas curvas será muito melhor do que sem a API de calibração, como visto nas figuras 7, 8, 9 e 10.

Vamos então colocar a mão na massa e utilizar essa API de calibração, que faz todo o trabalho sujo para nós. O código é simples, bastando apenas inicializar a estrutura interna para calibração e depois converter o valor RAW lido para mV com a função específica da API. A API utilizará o melhor método para afinar a leitura sempre que disponível.

Código

```
#include <driver/adc.h>
#include <esp_adc_cal.h>
#include <freertos/FreeRTOS.h>
#include <freertos/task.h>
#include <esp_err.h>
#include <esp_log.h>

esp_adc_cal_characteristics_t adc_cal;//Estrutura que contem as informacoes para calibracao


void app_main()
{
        adc1_config_width(ADC_WIDTH_BIT_12);//Configura a resolucao
        adc1_config_channel_atten(ADC1_CHANNEL_0, ADC_ATTEN_DB_11);//Configura a atenuacao


    esp_adc_cal_value_t adc_type = esp_adc_cal_characterize(ADC_UNIT_1, ADC_ATTEN_DB_11, ADC_WIDTH_BIT_12,
1100, &adc_cal);//Inicializa a estrutura de calibracao

  if (adc_type == ESP_ADC_CAL_VAL_EFUSE_VREF)
        {
                ESP_LOGI("ADC CAL", "Vref eFuse encontrado: %umV", adc_cal.vref);
        }
        else if (adc_type == ESP_ADC_CAL_VAL_EFUSE_TP)
        {
                ESP_LOGI("ADC CAL", "Two Point eFuse encontrado");
        }
        else
        {
```

```
            ESP_LOGW("ADC CAL", "Nada encontrado, utilizando Vref padrao: %umV", adc_cal.vref);
    }


    while (1)
    {
            /*
                    Obtem a leitura RAW do ADC para depois ser utilizada pela API de calibracao

                    Media simples de 100 leituras intervaladas com 30us
            */
            uint32_t voltage = 0;
            for (int i = 0; i < 100; i++)
            {
                    voltage += adc1_get_raw(ADC1_CHANNEL_0);//Obtem o valor RAW do ADC
                    ets_delay_us(30);
            }
            voltage /= 100;


            voltage = esp_adc_cal_raw_to_voltage(voltage, &adc_cal);//Converte e calibra o valor lido (RAW) para mV
            ESP_LOGI("ADC CAL", "Read mV: %u", voltage);//Mostra a leitura calibrada no Serial Monitor



            vTaskDelay(pdMS_TO_TICKS(1000));//Delay 1seg
    }
}
```

Testando este código, você também vai descobrir se seu ESP32 tem os valores do Vref ou Two Point gravados na memória, além de verificar a tensão calibrada sendo mostrada no Serial Monitor.

Atenção, é necessário que as opções de Vref e/ou Two Point estejam habilitadas no MENUCONFIG para API conseguir utilizar, o que vem por padrão ON, mas pode ser necessário você olhar manualmente se está ativado. Mais informações nas referências.

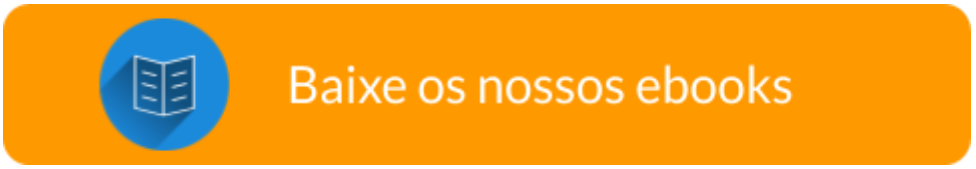
Conclusão

Após vários testes com o ADC no ESP32, podemos chegar à conclusão que ele realmente não é tão bom para alguns produtos, mas a calibração da Espressif é muito útil e serve para muitos outros, ainda mais quando 0 dB. Se você precisa de algo ainda mais sofisticado, pode utilizar filtros digitais e/ou analógicos para deixar um sinal mais agradável, além de aplicar alguma fórmula matemática para correção da curva do ADC (Excel pode fazer isso!).

Caso seu produto demande um ADC melhor, seria aconselhável utilização de um ADC externo, que é dedicado a esta função e costuma ser muito melhor que ADC embutidos de microcontroladores. Continue estudando maneiras para calibração de ADC's e diga para gente, nos comentários, o método utilizado!

Referências

1.https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/adc.html#
2.https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/adc.html#adc-calibration
3.https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf



ESP32 - Segurança e proteção da flash

Autor: José Maia

Neste artigo trataremos, de forma fácil e rápida, um assunto muito importante para quem pretende comercializar produtos com o ESP32, a segurança do seu hardware com o código presente na memória flash, a fim de impedir clonagem, furto e etc.

Utilizaremos a ESP-IDF v4.0-dev-76-g96aa08a0f-dirty (Ubuntu) para todos artigos desta série e não será abordado sobre como utilizar a IDF, sendo dever do leitor conhecer o funcionamento. Mais sobre a IDF: https://github.com/espressif/esp-idf

Figura 1 - Protegendo o ESP32.

Explicando em miúdos

Todo código (firmware) transferido ao ESP32 fica salvo, na maioria das versões, na memória flash externa, que diminui ainda mais a segurança, já que alguém pode simplesmente removê-la para leitura em um hardware externo e clonar, em segundos, nosso código que pode ter demorado anos para ser desenvolvido. Mesmo se a flash for embutida, como na versão "PICO", é possível exportar todo conteúdo da flash com

apenas um comando no terminal. Então, se todo conteúdo pode ser facilmente obtido, devemos nos proteger e é isso que a criptografia da flash do ESP32 nos proporciona.

Criptografia da flash

Atenção

- Não abordaremos todas funções e características da criptografia da flash, sendo necessário que você estude MUITO BEM a documentação oficial, a fim de evitar qualquer dor de cabeça que pode ocorrer conforme a IDF se atualiza. Não somos responsáveis por qualquer uso errado de sua parte.
- A criptografia da flash limita como e/ou quantas vezes é possível fazer upload de novos códigos. Caso feito incorretamente, você pode perder seu ESP32 e não será mais possível regrava-lo.
- Abordaremos, por motivos de didática, apenas sobre a criptografia com uma chave pré-gerada, assim, podemos regravar o ESP32 sem qualquer restrição de quantidade.
- Em ambientes que é necessário a maior segurança disponível, você não deve utilizar uma chave pré-gerada, deixando o próprio ESP32 gerar a sua, sendo individual para cada hardware. Além de também habilitar o Secure boot que não abordaremos aqui.

A criptografia da flash (AES-256) é uma característica presente no ESP32 que criptografa o conteúdo presente na flash. Quando habilitado, leituras físicas sem a chave não são suficientes para recuperar o conteúdo. Sendo assim, nos protegemos de quem tentar exportá-la para clonagem e etc. A chave é gravada em um bloco de eFuse, que pode ser protegido contra leitura e escrita (padrão) e, conhecendo a chave, podemos regravar

códigos sem a limitação de quantidade, diferentemente do caso onde o ESP32 gera sua própria chave, onde estamos limitados em até 3 uploads físicos.

Vamos observar alguns itens relevantes sobre a criptografia:

- Em um upload plaintext, o binário original (cru) é enviado ao microcontrolador.
- Em um upload criptografado, o binário é enviado ao microcontrolador já criptografado pela IDF.
- O eFuse "FLASH_CRYPT_CNT" (7-bit) é responsável pela permissão de uploads plaintext, pela contagem de uploads físicos plaintext (até 3x) e pelo controle do bootloader para criptografar o conteúdo da flash. Pode ser protegido contra R/W. Após 3 uploads plaintext, este eFuse chegará em seu máximo e aceitará apenas uploads criptografados.
    - Quando for um número par, o bootloader irá criptografar todo conteúdo da flash, logo, é necessário o upload plaintext.
    - Quando for um número ímpar, o bootloader não irá criptografar o conteúdo da flash, logo, é necessário o upload criptografado.
- Se a chave não for conhecida (pré-gerada), temos no máximo 3 uploads físicos disponíveis (plaintext), que também nos permite desabilitar a criptografia. Se o "FLASH_CRYPT_CNT" for protegido enquanto ímpar, não será possível novos uploads plaintext.
- Os binários "Bootloader", "Partition table", "OTA DATA", todas "APP (seu código)" e as partições marcadas com a flag "encrypted" na tabela de partição serão criptografados. Partições que não estiverem marcados com "encrypted" não serão criptografados e poderão ser lidos externamente, tenha atenção ao utilizar APIs para acesso da flash como NVS e SPIFFS.
- Se o "FLASH_CRYPT_CNT" não for protegido corretamente e/ou ainda houver alguma tentativa para upload plaintext, invasores podem inserir códigos maliciosos e ler o conteúdo de forma descriptografada pelo próprio ESP32 sem conhecimento

da chave. Por causa disto, é comum protegê-lo contra escrita, a fim de evitar uploads plaintext.

Cientes dos detalhes básicos (há dezenas de detalhes extras na documentação oficial que você deve ler antes de efetuar os testes abaixo), vamos testar a criptografia da flash e ver se realmente funciona! Lembrando que utilizaremos a IDF no Ubuntu e quase todos comandos podem mudar de acordo com seu computador, projeto, endereços e muitas outras coisas. Os comandos utilizados aqui podem não servir para você, logo, terá que alterá-los de acordo com seu projeto, caminho de arquivos e etc. Este artigo é apenas uma demonstração e deve ser tomado como base, não seguido ao pé da letra. Os scripts utilizados estão na pasta da IDF, "esp-idf/components/esptool_py/esptool/", tome bastante atenção ao uso dos caminhos dos arquivos e scripts, pois você pode estar tentando rodar o comando no local errado. Também tenha atenção na porta utilizada, seu ESP32 pode estar em uma porta diferente da nossa.

Primeiramente, vamos testar um código simples apenas para verificar sobre o que foi citado acima, sobre a leitura (clonagem) da memória flash sem proteção. Você pode ignorar essa parte.

```
void app_main()
{

	while (1)
	{
		ESP_LOGI("ESP32", "Embarcados...");
		vTaskDelay(pdMS_TO_TICKS(1000));
	}
}
```

Quando fazemos um upload padrão para placa, o computador utiliza um script python (esptool.py) com o comando "make flash", que basicamente compila e faz upload

(plaintext). Com o código enviado, vamos fazer uma leitura (sumário) dos eFuses para comparação após ativar a criptografia:

```
python espefuse.py --port /dev/ttyUSB0 summary

espefuse.py v2.6
Connecting....._
EFUSE_NAME (Block) Description = [Meaningful Value] [Readable/Writeable] (Hex Value)
----------------------------------------------------------------------------------------
Security fuses:
FLASH_CRYPT_CNT (BLOCK0)  Flash encryption mode counter = 0 R/W (0x0)
FLASH_CRYPT_CONFIG (BLOCK0)  Flash encryption config (key tweak bits) = 0 R/W (0x0)
CONSOLE_DEBUG_DISABLE (BLOCK0)  Disable ROM BASIC interpreter fallback = 1 R/W (0x1)
ABS_DONE_0 (BLOCK0)  secure boot enabled for bootloader = 0 R/W (0x0)
ABS_DONE_1 (BLOCK0)  secure boot abstract 1 locked = 0 R/W (0x0)
JTAG_DISABLE (BLOCK0)  Disable JTAG = 0 R/W (0x0)
DISABLE_DL_ENCRYPT (BLOCK0)  Disable flash encryption in UART bootloader = 0 R/W (0x0)
DISABLE_DL_DECRYPT (BLOCK0)  Disable flash decryption in UART bootloader = 0 R/W (0x0)
DISABLE_DL_CACHE (BLOCK0)  Disable flash cache in UART bootloader = 0 R/W (0x0)
BLK1  Flash encryption key
 = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 R/W
BLK2  Secure boot key
 = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 R/W
BLK3  Variable Block 3
 = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 R/W

Efuse fuses:
WR_DIS (BLOCK0)  Efuse write disable mask = 0 R/W (0x0)
RD_DIS (BLOCK0)  Efuse read disablemask = 0 R/W (0x0)
CODING_SCHEME (BLOCK0)  Efuse variable block length scheme = 0 R/W (0x0)
KEY_STATUS (BLOCK0)  Usage of efuse block 3 (reserved) = 0 R/W (0x0)

Config fuses:
XPD_SDIO_FORCE (BLOCK0)  Ignore MTDI pin (GPIO12) for VDD_SDIO on reset = 0 R/W (0x0)
XPD_SDIO_REG (BLOCK0)  If XPD_SDIO_FORCE, enable VDD_SDIO reg on reset = 0 R/W (0x0)
XPD_SDIO_TIEH (BLOCK0)  If XPD_SDIO_FORCE & XPD_SDIO_REG, 1=3.3V 0=1.8V = 0 R/W (0x0)
SPI_PAD_CONFIG_CLK (BLOCK0)  Override SD_CLK pad (GPIO6/SPICLK) = 0 R/W (0x0)
SPI_PAD_CONFIG_Q (BLOCK0)  Override SD_DATA_0 pad (GPIO7/SPIQ) = 0 R/W (0x0)
SPI_PAD_CONFIG_D (BLOCK0)  Override SD_DATA_1 pad (GPIO8/SPID) = 0 R/W (0x0)
SPI_PAD_CONFIG_HD (BLOCK0)  Override SD_DATA_2 pad (GPIO9/SPIHD) = 0 R/W (0x0)
SPI_PAD_CONFIG_CS0 (BLOCK0)  Override SD_CMD pad (GPIO11/SPICS0) = 0 R/W (0x0)
DISABLE_SDIO_HOST (BLOCK0)  Disable SDIO host = 0 R/W (0x0)

Identity fuses:
MAC (BLOCK0)  Factory MAC Address
```

```
 = b4:e6:2d:96:dc:41 (CRC 84 OK) R/W
CHIP_VER_REV1          Silicon Revision 1                    = 1 R/W (0x1)
CHIP_VERSION           Reserved for future chip versions     = 2 R/W (0x2)
CHIP_PACKAGE           Chip package identifier               = 0 R/W (0x0)

Calibration fuses:
BLK3_PART_RESERVE   BLOCK3 partially served for ADC calibration data  = 0 R/W (0x0)
ADC_VREF       Voltage reference calibration                 = 1114 R/W (0x2)

Flash voltage (VDD_SDIO) determined by GPIO12 on reset (High for 1.8V, Low/NC for 3.3V).
```

Sabendo que este ESP32 não está protegido, podemos exportar o conteúdo da flash e usar para clonagem, engenharia reversa ou o que der na cabeça! Vamos ver se achamos a palavra usada no ESP_LOGI(), “Embarcados...”, ao ler o conteúdo da memória:

```
python esptool.py --port /dev/ttyUSB0 read_flash 0x0 4096000 dump.bin

esptool.py v2.6
Serial port /dev/ttyUSB0
Connecting....
Detecting chip type... ESP32
Chip is ESP32D0WDQ6 (revision 1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
MAC: b4:e6:2d:96:dc:41
Uploading stub...
Running stub...
Stub running...
4096000 (100 %)
4096000 (100 %)
Read 4096000 bytes at 0x0 in 367.1 seconds (89.3 kbit/s)...
Hard resetting via RTS pin...
```

Utilizando o comando “hexdump” para ler o arquivo gerado, conseguimos achar a palavra utilizada no código sem criptografia:

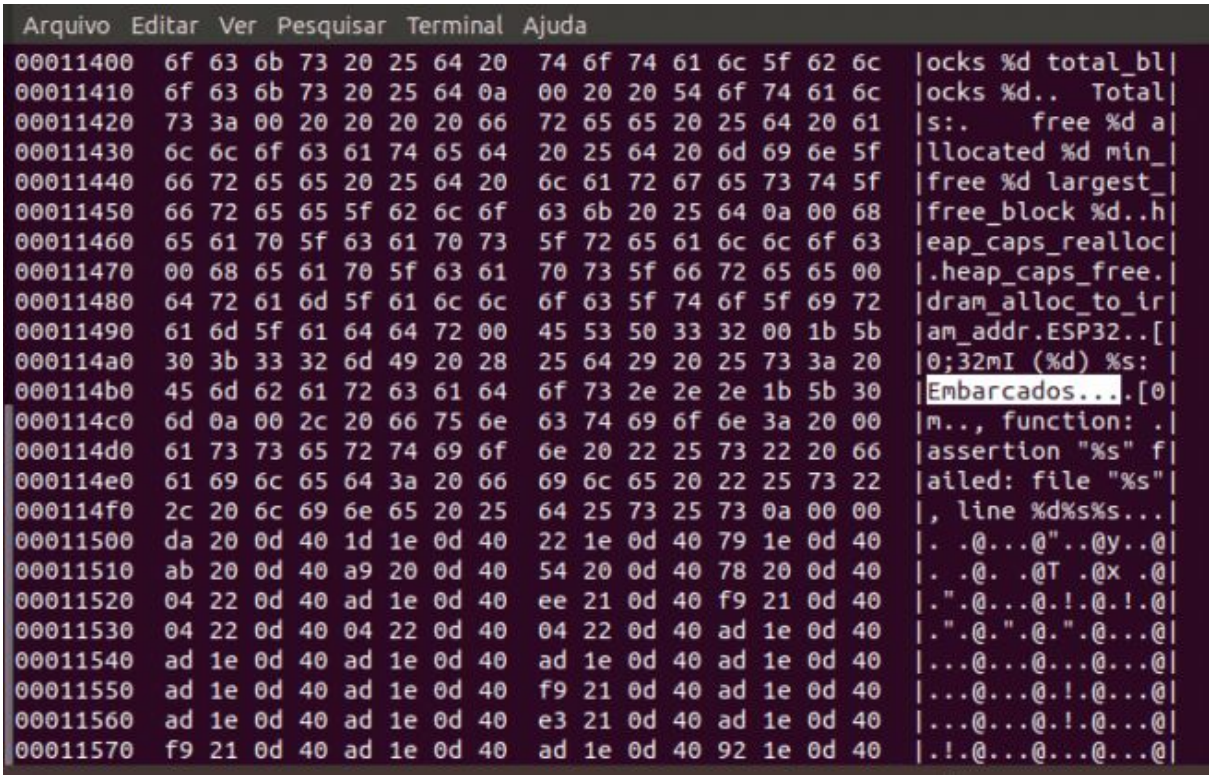


Figura 2 - Memória do ESP32 clonada antes da criptografia.

Agora que a clonagem foi demonstrada com apenas um comando no terminal, indicando que seu produto sem proteção está totalmente vulnerável nas mãos de alguém, vamos nos proteger ativando a criptografia.

Criando a chave

```
python espsecure.py generate_flash_encryption_key key.bin
```

Utilizaremos o próprio script da IDF para criar uma chave (256b), mas você pode utilizar qualquer método ou chave que desejar. O comando exportará a chave no arquivo

“key.bin”, que você deve deixar uma cópia dentro da pasta do seu projeto, ficando algo similar com o nosso:

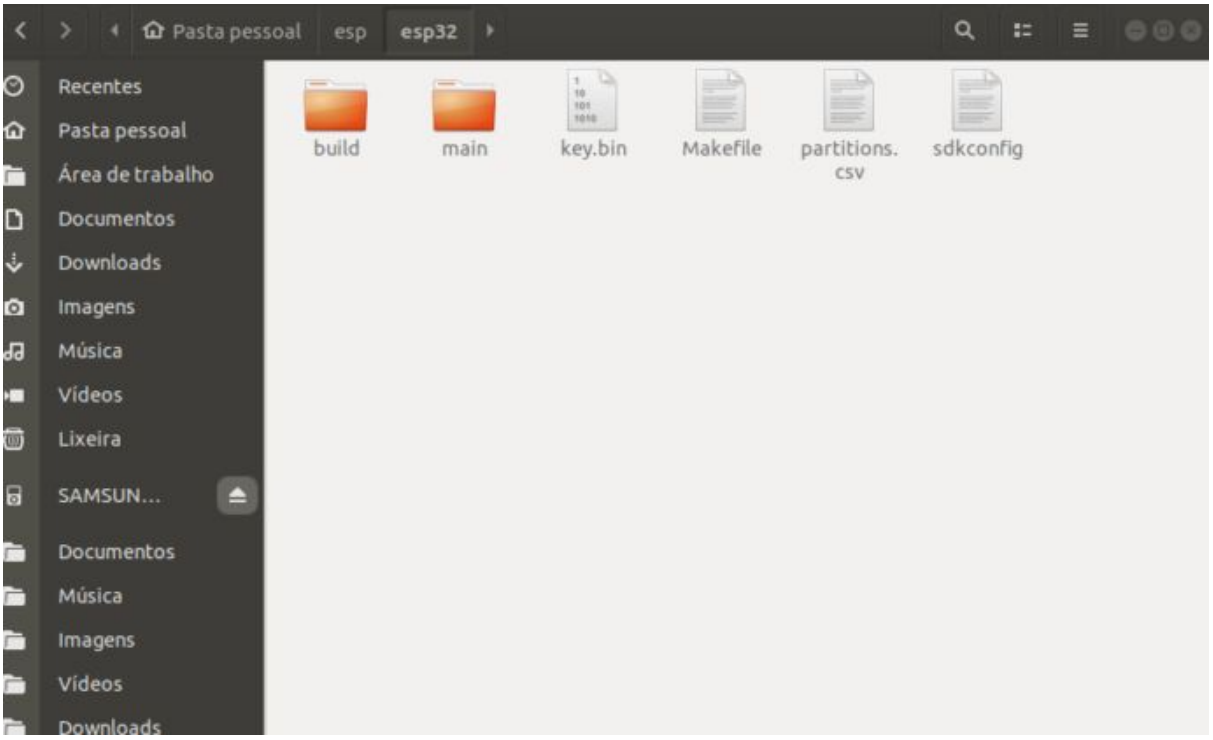


Figura 3 - Pasta do projeto com chave.

Gravando a chave pré-gerada no eFuse

```
python espefuse.py --port /dev/ttyUSB0 burn_key flash_encryption key.bin

espefuse.py v2.6
Connecting........_____...
Write key in efuse block 1. The key block will be read and write protected (no further changes
or readback). This is an irreversible operation.
Type 'BURN' (all capitals) to continue.
BURN
Burned key data. New value: 56 f7 1f 29 a6 6c 01 20 c5 ee 6b 55 79 36 d7 90 73 b3 f6 2d 48 a2
dc 03 b8 ad dc cb 1b 31 fe e5
```

```
Disabling read/write to key efuse block...
```

O comando acima gravou nossa chave no eFuse e automaticamente protegeu contra R/W, o que impede de qualquer um conseguir lê-la ou alterá-la. Apesar deste comando escrever no terminal a chave gravada, ao tentar ler a chave diretamente do eFuse (sumário) como feito anteriormente, é retornado:

```
BLK1                Flash encryption key
  = ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? -/-
```

Agora com uma chave conhecida no eFuse, temos a possibilidade de uploads ilimitados (criptografados), o que é muito interessante na fase de desenvolvimento.

Ativando a criptografia da flash

Agora com a chave gravada, basta ativar a criptografia da flash no “menuconfig” em “Security features”.

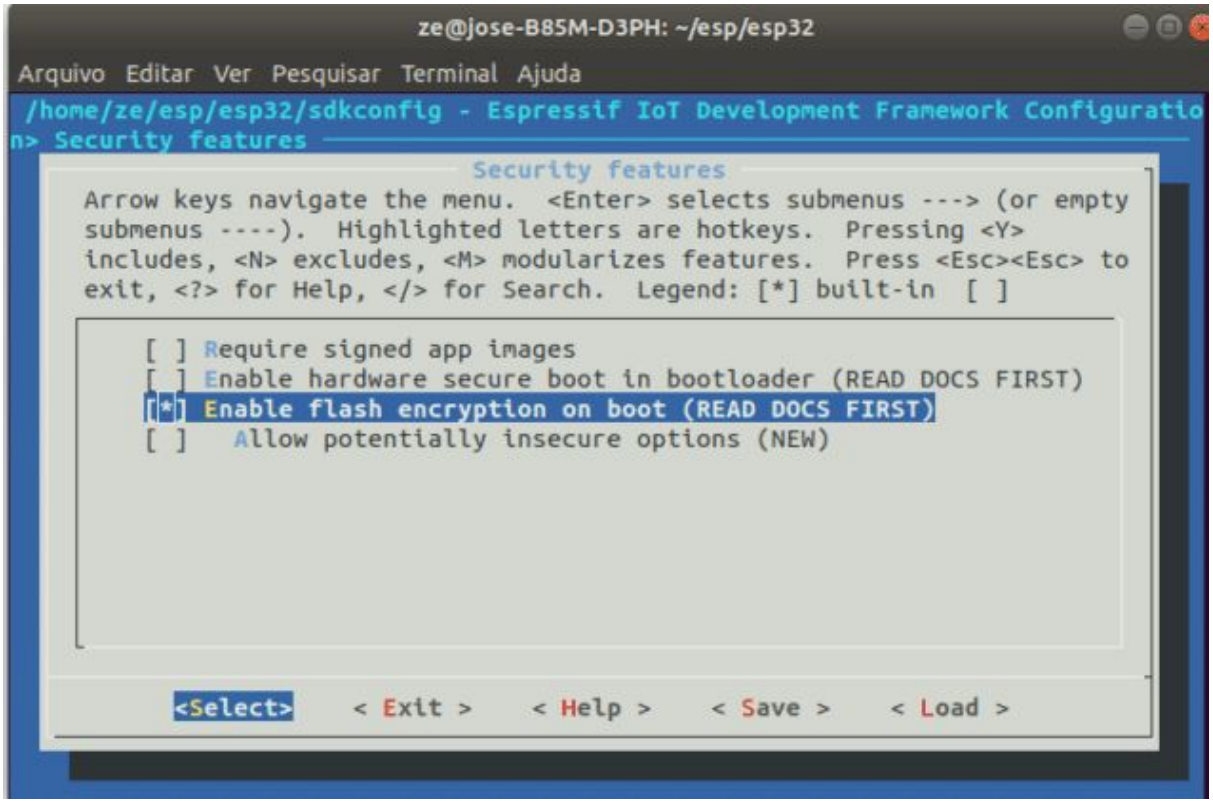


Figura 4 - Ativando a criptografia da flash.

Atenção, se você não gravou a chave pré-gerada, ao dar o upload de um novo código com a criptografia ativada, o próprio ESP32 irá gerar uma chave que nem você, nós ou a Espressif poderá ler, lhe restando 3 uploads (plaintext) e/ou tentativas para desativar a criptografia, tome cuidado! Atualizações OTA são ilimitadas mesmo sem conhecimento da chave.

Após ativar a criptografia, vamos refazer o upload do código utilizado anteriormente de forma padrão, como utilizado antes (upload plaintext). Nesse primeiro boot, o bootloader irá criptografar todo conteúdo da memória e reiniciará, esse processo pode demorar um pouco então aguarde. Após o ESP32 reiniciar, indicando que a criptografia foi ativada, vamos fazer uma nova leitura da flash para ver se encontramos a palavra “Embarcados...” novamente.

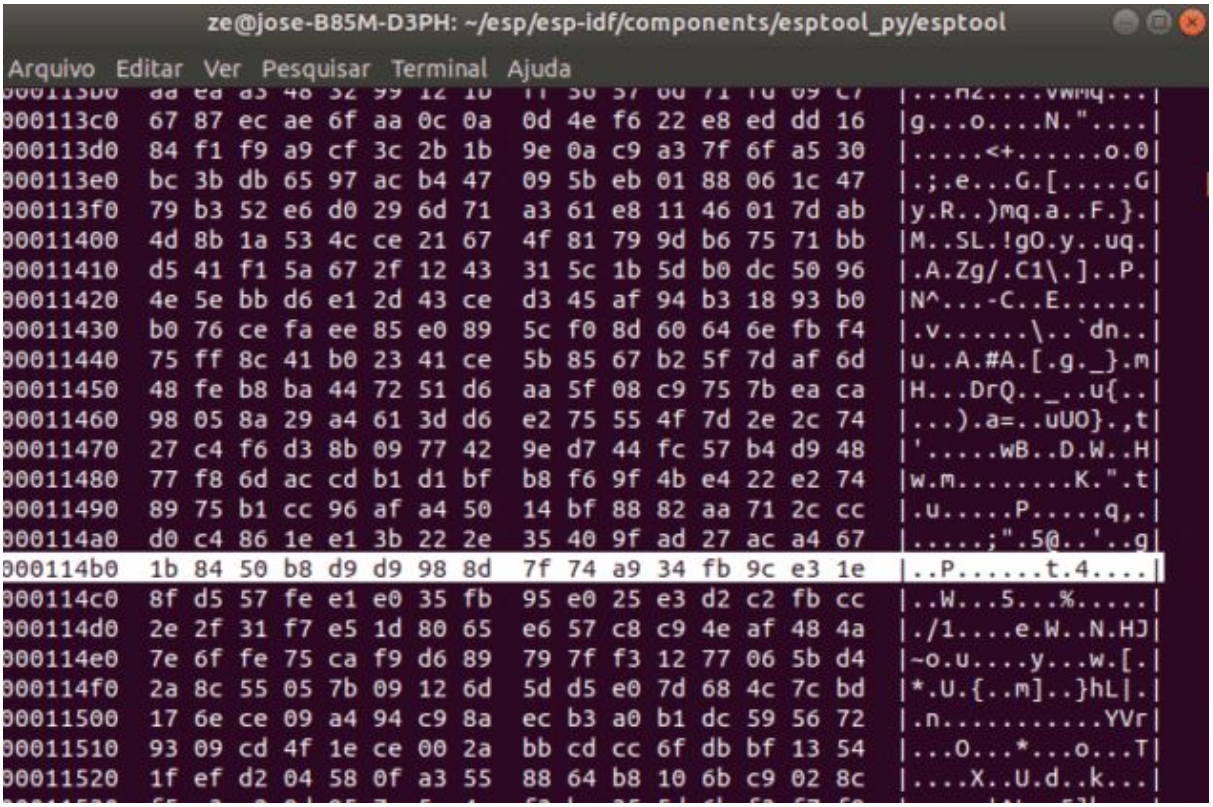


Figura 5 - Memória do ESP32 clonada após criptografia.

Além da palavra "Embarcados..." não ser encontrada, nenhum texto legível foi visto, o que anteriormente era facilmente visto (Strings utilizadas pela própria IDF). O mesmo endereço que encontramos a palavra anteriormente, agora, não passa de um texto corrompido para quem tentar ler sem a chave!

Se olharmos o sumário dos eFuses novamente, podemos ver que o "FLASH_CRYPT_CNT" foi de 0 para 1, indicando que agora só aceita uploads criptografados, logo, se utilizarmos o upload padrão (plaintext), o ESP irá ficar em boot-loop com a seguinte mensagem:

```
make flash monitor -j4

ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
flash read err, 1000
```

```
ets_main.c 371
ets Jun  8 2016 00:22:57

rst:0x10 (RTCWDT_RTC_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
flash read err, 1000
ets_main.c 371
ets Jun  8 2016 00:22:57

rst:0x10 (RTCWDT_RTC_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
flash read err, 1000
ets_main.c 371
ets Jun  8 2016 00:22:57
```

O “FLASH_CRYPT_CNT” não é protegido contra escrita por padrão, logo, alguém ainda pode desativar a criptografia, fazer upload (plaintext) e ler sua flash de forma descriptografada, logo, devemos proteger este eFuse contra escrita ENQUANTO estiver em um número ímpar ou utilizar o Secure boot.

Protegendo o eFuse

Vamos proteger o eFuse “FLASH_CRYPT_CNT” contra escritas para impossibilitar qualquer tipo de upload sem conhecimento da chave, isso automaticamente permite que suas placas estejam protegidas de uploads não permitidos, forçando seu hardware a aceitar apenas códigos que tenham sidos criptografados com a chave. Apesar desta proteção funcionar com o mesmo intuito do Secure boot (impossibilitar uploads não permitidos), há alguns casos em que manter o Secure boot ativado pode ser melhor, entretanto, na maioria dos casos, protegendo o eFuse já não precisamos do Secure boot (pesquise melhor sobre isso se for usar em ambientes agressivos onde a segurança deve prevalecer).

Antes de proteger a escrita, devemos certificar-se que ele está em algum número ímpar através do sumário, que nos retornou “FLASH_CRYPT_CNT Flash encryption mode counter = 1 R/W (0x1)”

```
python espefuse.py --port /dev/ttyUSB0 write_protect_efuse FLASH_CRYPT_CNT

espefuse.py v2.6
Connecting........_
Permanently write-disabling efuse FLASH_CRYPT_CNT. This is an irreversible operation.
Type 'BURN' (all capitals) to continue.
BURN
```

A partir de agora, é impossível desabilitar a criptografia ou enviar códigos que não estejam criptografados pela chave presente no ESP32.

Ok, se o upload padrão não funciona mais, o que faremos? Criptografamos antes de enviar! Apesar de ser utilizado AES-256, a Espressif usa métodos diferentes de funcionamento (detalhes na documentação oficial), logo, precisamos criptografar pelo próprio script da IDF

Criptografando os binários

O upload agora deve ser feito “manualmente”, sendo necessário criptografar e enviar os binários criptografados. Essa parte depende muito de projeto para projeto, principalmente nos caminhos de arquivos e endereço da memória, tome muita atenção.

5.1 Crie uma pasta “enc” dentro do seu projeto, ela irá guardar os binários criptografados pelo script.

5.2 Com o terminal aberto na pasta do seu projeto, use o comando “make all” para descobrir o que e onde os arquivos são gravados.

```
make all -j4

Toolchain path: /home/ze/esp/xtensa-esp32-elf/bin/xtensa-esp32-elf-gcc
Toolchain version: crosstool-ng-1.22.0-80-g6c4433a
Compiler version: 5.2.0
Project is not inside a git repository, will not use 'git describe' to determine PROJECT_VER.
App "esp32" version: 1
Python requirements from /home/ze/esp/esp-idf/requirements.txt are satisfied.

To flash all build output, run 'make flash' or:
python /home/ze/esp/esp-idf/components/esptool_py/esptool/esptool.py --chip esp32 --port /dev/ttyUSB0 --baud 921600 --before default_reset --after hard_reset write_flash -z --flash_mode dio --flash_freq 80m --flash_size detect 0x363000 /home/ze/esp/esp32/build/ota_data_initial.bin 0x1000 /home/ze/esp/esp32/build/bootloader/bootloader.bin 0x10000 /home/ze/esp/esp32/build/esp32.bin 0x8000 /home/ze/esp/esp32/build/partitions.bin
```

Podemos ver que nesse nosso projeto, onde utilizamos OTA e uma tabela de partições (custom), é feito o upload de 4 binários nos seus respectivos endereços.

ota_data_initial.bin: 0x363000.

bootloader.bin: 0x1000.

esp32.bin (o código em si): 0x10000.

partitions.bin: 0x8000.

Iremos criptografá-los individualmente e posteriormente, efetuar o upload. Não se esqueça que os caminhos dos arquivos devem ser alterados para o seu projeto.

5.3 Com o terminal aberto na pasta de scripts, use os comandos abaixo e não se esqueça de tomar muita atenção com os endereços de memória e caminho dos binários.

```
python espsecure.py encrypt_flash_data --keyfile key.bin --address 0x1000 -o /home/ze/esp/esp32/enc/bootloader.bin /home/ze/esp/esp32/build/bootloader/bootloader.bin

python espsecure.py encrypt_flash_data --keyfile key.bin --address 0x8000 -o /home/ze/esp/esp32/enc/partitions.bin /home/ze/esp/esp32/build/partitions.bin

python espsecure.py encrypt_flash_data --keyfile key.bin --address 0x10000 -o /home/ze/esp/esp32/enc/esp32.bin /home/ze/esp/esp32/build/esp32.bin

python espsecure.py encrypt_flash_data --keyfile key.bin --address 0x363000 -o /home/ze/esp/esp32/enc/ota_data_initial.bin /home/ze/esp/esp32/build/ota_data_initial.bin
```

5.4 Faça o upload dos binários criptografados pelo comando:

```
python esptool.py --chip esp32 --port /dev/ttyUSB0 --baud 921600 write_flash --flash_mode dio
--flash_freq 80m --flash_size detect 0x1000 /home/ze/esp/esp32/enc/bootloader.bin 0x8000
/home/ze/esp/esp32/enc/partitions.bin 0x10000 enc/esp32.bin 0x363000
/home/ze/esp/esp32/enc/ota_data_initial.bin
```

Após o upload criptografado, o ESP32 iniciará normalmente como se nada tivesse acontecido. Esse método apesar de ser manual, pode ser automatizado por um script (bash), basta colocá-los num arquivo e executar no terminal.

Agora que a proteção do ESP32 esta ativada, podemos ficar mais relaxados na questão sobre alguém clonar nosso firmware, já que não será possível sem a chave gravada. Você deve ler toda documentação oficial e também pode ser interessante utilizar o Secure boot sem criptografia da flash, análise seu projeto e mãos na massa!

Referências

1. https://docs.espressif.com/projects/esp-idf/en/latest/security/flash-encryption.html

Controlando ESP32 via WiFi com validação por endereço MAC

Autor:Roger Moschiel

O macaddress é um endereço físico único em formato hexadecimal com 6 bytes, que é atribuído a todo hardware feito para comunicação em rede, e como a identificação é

única, ela pode ser usada para fazer o controle de acesso em uma rede local, onde por exemplo, os roteadores permitem o bloqueio de endereços MAC específicos.

Neste tutorial vamos aplicar essa funcionalidade na ESP32, usando o Arduino IDE para programa-la no modo AP, e validar o acesso de dispositivos conectados através do endereço MAC. Somente após feita a validação será permitido ao usuário utilizar o sistema.

Importante informar que esse tutorial é de caráter didático, uma vez que, apesar do MAC ser único para cada dispositivo, é possível alterá-lo usando técnicas específicas, o que não garante segurança efetiva contra usuários não autorizados. O código completo usado nesse tutorial está disponível na minha conta do Github.

Instalando bibliotecas

Existe um complemento para o Arduino IDE afim de suportar as placas da ESP32, para fazer a instalação confira o tutorial para Windows do Gabriel Almeida. Para outros sistemas operacionais, siga as instruções na conta Github dos fornecedores do complemento.

Também iremos usar a biblioteca ESP_ClientMacaddress disponível no meu Github.

Configurando WiFi no modo AP (Access Point)

Um Access Point é uma Rede de Conexão Local Wireless (WLAN), por onde podemos conectar um dispositivo (client) a ESP (host) via WiFi.

Primeiro importamos a seguinte biblioteca para configurar a conexão WiFi na ESP.

```
#include <WiFi.h>
```

Escolha o SSID e a Senha que desejar, essa será a forma de identificar a rede na qual o usuário deverá conectar-se:

```
const char* ssid = "meu_ssid";
const char* senha = "minha_senha";
```

WiFi Server configurado na porta 80:

```
WiFiServer server(80);
```

Dentro do **setup()**, para setar a ESP no modo AP usamos este comando:

```
WiFi.softAP(ssid, senha);
```

E para iniciar o servidor:

```
server.begin();
```

Neste ponto seu código deve estar dessa forma:

```
#include <WiFi.h> //funcionalidades de conexão wifi, como AP e WebServer
//Crendenciais do ponto de acesso
const char *ssid = "meu_ssid";
const char *senha = "minha_senha";
WiFiServer server(80); //Porta padrão 80
void setup() { //Inicializa serial
Serial.begin(115200);
Serial.println(); //Configura ESP no modo
APSerial.printf("Configurando ponto de acesso '%s'n", ssid);
WiFi.softAP(ssid, senha);
server.begin();
Serial.println("Configuração conluída"); }

void loop() { }
```

Agora grave o código na ESP, e então use um celular pra verificar se a rede WiFi aparece disponível, mas não conecte ainda.

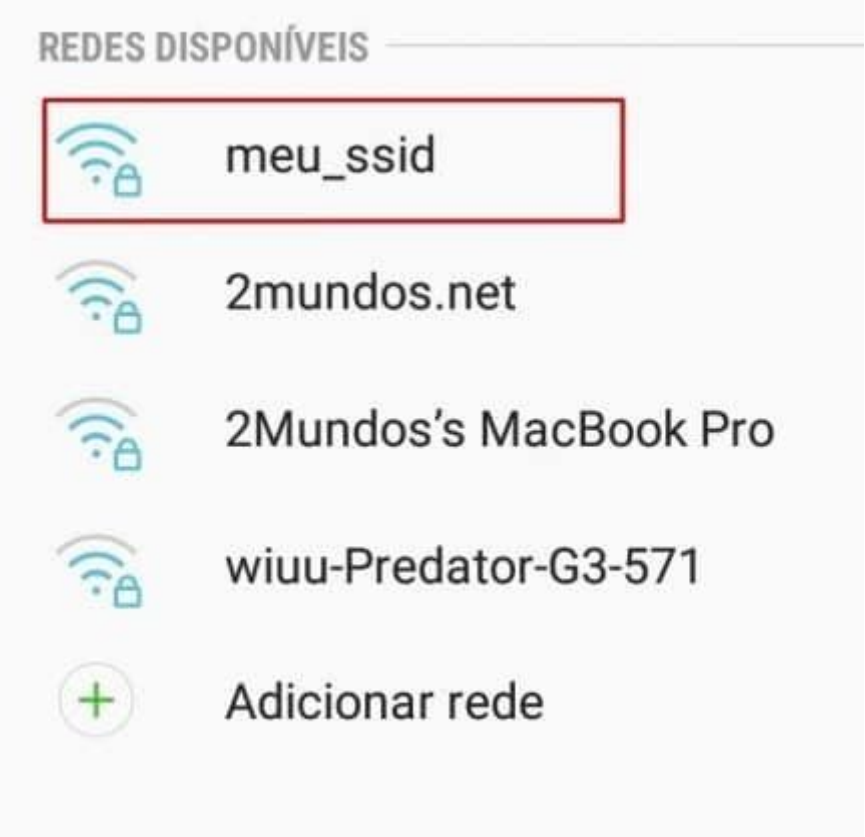


Fig 1 – Redes Disponíveis

Enviando comandos através dos cabeçalhos HTTP do client

Edite o loop() de acordo com o código a seguir.

```
void loop() {

  //Verifica se existe algum client na rede
```

```
  WiFiClient client = server.available();
  //Caso positivo, imprime "Novo Client" no monitor
  if (client){
    Serial.println("Novo Client");
    //Enquanto client conectado, verifica se existem bytes a serem lidos
    //e concatena os bytes recebidos na String cabecalho;
    while (client.connected()){
      if (client.available()){
        cabecalho += (char)client.read();
        //Se receber nova linha em branco, encerrou leitura dos dados
        if (cabecalho.endsWith("nrn")){
          Serial.println(cabecalho); //imprime cabeçalhos http recebidos
          //iniciamos a resposta http com o código OK(200),
          //o tipo de conteúdo a ser enviado e tipo de conexão.
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type:text/html");
          client.println("Connection: close");
          client.println();
          //INSIRA AQUI SUA APLICAÇÃO

          break; //sai do while loop
        }
      }
    }
    cabecalho = ""; //ao encerrar conexão, limpa variável cabecalho
    client.flush(); client.stop();
    Serial.println("Client desconectado."); Serial.println();
  }
}
```

Assim durante a execução do loop(), a ESP estará sempre verificando se foi feita alguma conexão. Uma vez recebida uma solicitação do client, iremos salvar os dados recebidos na String cabeçalho, e com o comando client.println() respondemos com o status OK(código 200) para sinalizar ao client que recebemos os dados com sucesso. Mais

adiante, o campo comentando "INSIRA AQUI SUA APLICACAO" será o local onde iremos programar as funcionalidades de controle.

Grave o código, e com um celular conecte-se a rede WiFi que você nomeou, e então no navegador digite o endereço IP 192.168.4.1 do nosso host(ESP), fazendo isso, no monitor do Arduino IDE teremos o output com os cabeçalhos HTTP recebidos com as informações do dispositivo.

Fig 2 – Resposta no Monitor Serial do Arduino IDE

Experimente digitar no seu navegador o IP da ESP seguido de qualquer texto, por exemplo digite 192.168.4.1/teste, e verá que o conteúdo da url digitada também é capturado, logo, através da url podemos determinar comandos a serem enviados para a ESP.

Fig 3 – Comando recebido através dos cabeçalhos HTTP

Vamos usar essa estratégia para enviar comandos a ESP através do navegador. Neste caso enviaremos comandos para alternar o estado do LED embarcado no módulo da ESP32, que está conectado ao pino 2 da placa.

No início do setup(), adicione um comando para configurar o pino que controla o LED como saída.

```
void setup(){
  //Configura o pino conectado ao LED como saída
  pinMode(2, OUTPUT);
  .......
}
```

Dentro do loop(), abaixo do comentário "INSIRA AQUI SUA APLICAÇÃO", vamos projetar o sistema para fazer a leitura dos comandos LED_ON e LED_OFF, assim podendo controlar pelo navegador o estado do LED, da seguinte maneira:

```
...
//INSIRA AQUI SUA APLICAÇÃO
if(cabecalho.indexOf("GET /LED_ON")>= 0){
  digitalWrite(2, true); //Acende o LED
}else if(cabecalho.indexOf("GET /LED_OFF")>= 0){
  digitalWrite(2, false); //Apaga o LED
}
...
```

Grave o código, conecte-se a rede WiFi da ESP, e no navegador digite 192.168.4.1/LED_ON para acender o LED, ou 192.168.4.1/LED_OFF para apagar o LED.

Controlando ESP32 através de uma página web com HTML

Agora temos que criar nossa página web, para isso criamos a função run_html(), que receberá como parâmetro o próprio client.

```
void run_html(WiFiClient client) {
}
```

Então começamos o código html com as seguintes tags usadas para iniciar um documento html.

```
<!DOCTYPE html><html>
```

Os seguintes comandos CSS do <head > na nossa página alinham o conteúdo de acordo com o tamanho da tela do dispositivo.

```
<head><style media='screen' type='text/css'>
  html{display:inline-block;margin:10px auto;text-align:center;}
</style></head>
```

Em seguida fazemos o corpo da nossa página que é redigida entre as tags <body></body>, e no corpo inserimos um cabeçalho com o título 'ACIONAMENTO LED' usando as tags <h1></h1>.

```
<h1 style='font-size:80px'>ACIONAMENTO LED</h1>
```

Ainda no corpo da página, inserimos um botão entre as tags <button></button> com o título 'ON' e comprimento de 200px e tamanho da fonte de 80px.

```
<button style='width:200px;font-size:80px'>ON</button>
```

Anteriormente para acender o LED, tivemos que digitar no navegador o endereço IP mais o comando LED_ON, desta vez vamos atribuir essa função ao botão que acabamos de criar.

Para isso colocamos a linha de código do botão entre as tags <a> com o link de redirecionamento, e usamos a tag <p> para posicionar os botões verticalmente.

```
<p><a href='/LED_ON'>
  <button style='width:200px;font-size:80px'>ON</button>
</a></p>
```

E de forma semelhante fazemos mais um botão para desligar o LED.

```
<p><a href='/LED_OFF'>
  <button style='width:200px;font-size:80px'>OFF</button>
</a></p>
```

Finalmente encerramos o documento html com a tag </html>.

Dentro da função run_html(), vamos armazenar o conteúdo HTML na String html_content, e usar o comando client.println() afim de a ESP enviar o código html para o client, assim a função fica dessa forma:

```
void run_html(WiFiClient client){
```

```
  String html_content = \
  "<!DOCTYPE html><html>" \
    "<head><style media='screen' type='text/css'>" \
      "html{display:inline-block;margin:10px auto;text-align:center;}" \
    "</style></head>" \
    "<body>" \
      "<h1 style='font-size:40px'>Acionamento LED</h1>" \
      "<p><a href='/LED_ON'>" \
        "<button style='width:200px;font-size:80px'>ON</button>" \
      "</a></p>" \
      "<p><a href='/LED_OFF'>" \
        "<button style='width:200px;font-size:80px'>OFF</button>" \
      </a></p>" \
    "</body>" \
  "</html>";

  client.println(html_content);
}
```

Agora que nossa página web está pronta, voltaremos ao loop(), e no campo "INSIRA AQUI SUA APLICAÇÃO" adicionamos uma chamada pra função run_html().

```
//INSIRA AQUI SUA APLICAÇÃO HTML
run_html(client);
if(cabecalho.indexOf("GET /LED_ON")>= 0){
  digitalWrite(2, true); //Acende o LED
}else if(cabecalho.indexOf("GET /LED_OFF")>= 0){
  digitalWrite(2, false); //Apaga o LED
}
...
```

Grave o código na ESP e conecte-se a rede WiFi com seu celular ou computador, e no navegador digite o IP 192.168.4.1 para carregar a página html. Agora, ao invés de digitar

os comandos LED_ON ou LED_OFF no seu navegador para acionar o LED, você poderá fazer isso simplesmente acionando os botões da página, e a solicitação http será enviada do client (seu navegador) para o host (ESP).

Fig 4 - Página web para controle do LED

Validando acesso por WiFi macaddress

Importe a biblioteca ESP_ClientMacaddress.

```
#include <ESP_ClientMacaddress.h>
```

Acesse as configurações WiFi do seu celular e procure pelo endereço MAC (macaddress), no link você tem acesso a instruções de como visualizar o macaddress para Android, iOS, Windows Phone e PC.

Fig 5 - Endereço MAC de um celular com Android

De volta ao código, crie um array para armazenar os bytes do endereço MAC de quaisquer dispositivos que deseje autorizar o acesso.

Neste exemplo vou usar 3 macaddress fictícios, onde também deve-se definir a quantidade de dispositivos.

```
#define NUM_DISPOSITIVOS 3

uint8_t macList[NUM_DISPOSITIVOS][6] = {
```

```
  {0xA7,0x16,0xD0,0xA6,0x45,0x3B},
  {0xB8,0x17,0xE0,0xA7,0x55,0x3C},
  {0xC9,0x18,0xD0,0xA8,0x65,0x3D}
};
```

Crie a váriavel booleana mac_conhecido.

```
bool mac_conhecido;
```

Agora instanciamos a classe ClientMacaddress, passando como parâmetro a lista de endereços MAC e a quantidade de dispositivos autorizados.

```
ClientMacaddress clientMac(macList, NUM_DISPOSITIVOS);
```

Altere o código do campo "INSIRA AQUI SUA APLICAÇÃO" como demonstrado a seguir:

```
...

//INSIRA AQUI SUA APLICAÇÃO HTML

//Variável 'm' aponta para o endereço MAC do client

uint8_t *m = clientMac.getAddr(client);
Serial.printf("Macaddress:%.2X:%.2X:%.2X:%.2X:%.2X:%.2Xn",
              m[0],m[1],m[2],m[3],m[4],m[5]);

//determina se o endereço MAC do client é conhecido
mac_conhecido = clientMac.isKnown(m);

run_html(client); //envia ao client conteúdo HTML

//Se client possui MAC conhecido, libera acesso para controlar o LED
if(mac_conhecido){
```

```
    Serial.println("mac ok");
    if(cabecalho.indexOf("GET /LED_ON")>= 0){
      digitalWrite(2, true); //Acende o LED
    }else if(cabecalho.indexOf("GET /LED_OFF")>= 0){
      digitalWrite(2, false); //Apaga o LED
    }
  }else{
    Serial.println("mac não autorizado");
  }
  ...
```

Com essa alteração, o comando getAddr() armazena na variável m o endereço MAC do client, em seguida o comando isKnown() seta a variável mac_conhecido como true se o MAC for conhecido, ou false caso seja desconhecida, dessa forma impedindo que dispositivos não autorizados tenham acesso ao controle do LED.

E pra finalizar, alteramos a função run_html(), de forma que a página web só libere acesso aos botões ON e OFF se o client tiver um endereço MAC conhecido, caso contrário carregamos a mensagem de alerta "DISPOSITIVO NÃO AUTORIZADO".

```
void run_html(WiFiClient client){
  String html_content = \
  "<!DOCTYPE html><html>" \
    "<head><style media='screen' type='text/css'>" \
      "html{display:inline-block;margin:10px auto;text-align:center;}" \
    "</style></head>" \
    "<body>" \
    "<h1 style='font-size:40px'>Acionamento LED</h1>"; \
    if(mac_conhecido){
      html_content += \
      "<p><a href='/LED_ON'>" \
        "<button style='width:200px;font-size:80px'>ON</button>" \
      "</a></p>" \
      "<p><a href='/LED_OFF'>" \
        "<buttonstyle='width:200px;font-size:80px'>OFF</button>" \
```

```
      "</a></p>";
    }else{
      html_content += \
      "<p style='color:red;font-size:40px'>DISPOSITIVO NAO AUTORIZADO</p>";
    }
    html_content += \
    "</body>" \
  "</html>";

  client.println(html_content);
}
```

Grave o código, e desta vez, usando um dispositivo qualquer que você NÃO tenha cadastrado o endereço MAC, conecte-se a rede WiFi da ESP e digite o IP 192.168.4.1 no seu navegador. Assim o usuário será notificado que o dispositivo não é autorizado a acessar o sistema.

Fig 6 - Página web com mensagem de alerta

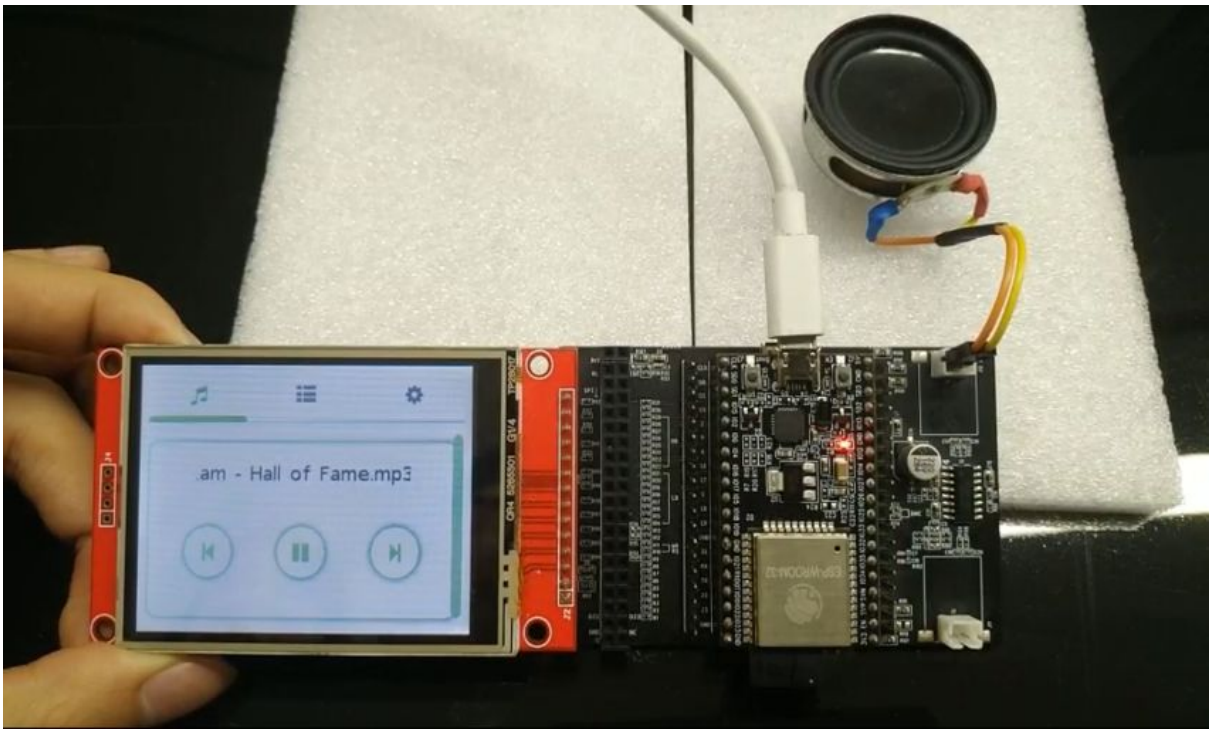
Acabamos!

Neste tutorial você aprendeu a como construir um Access Point com um módulo ESP32, usando a biblioteca ESP_ClientMacaddress para realizar a validação do Endereço MAC do dispositivo conectado.

Vimos um exemplo simples de como controlar um LED usando como interface uma página web, agora você pode trabalhar com base neste exemplo substituindo o LED para acionar um relé ou qualquer outro dispositivo, e também customizar a página web.

Espressif anuncia suporte para bibliotecas gráficas no ESP32

Autor:Muriel Costa



A Espressif anunciou dia 04/01/2019 em seu site que o ESP32 passa a ter suporte oficial para as bibliotecas gráficas LittlevGL e uGFX.

A previsão é que trabalhar com interfaces gráficas de usuário no ESP32 fique mais fácil e descomplicado, já que as bibliotecas agora estão disponíveis oficialmente. Com esta jogada a Espressif busca não só diminuir o tempo de desenvolvimento de produtos, mas também atender uma fatia ainda maior de mercado com o público que busca recursos gráficos com o usuário, principalmente de IOT, seu maior alvo.

LittlevGL

A LittlevGL é uma biblioteca gráfica livre e de código aberto, fornecendo tudo o que é necessário para criar uma GUI (Graphical User Interface) embutida com elementos gráficos fáceis de usar, belos efeitos visuais e baixo consumo de memória. A GUI personalizada pode ser criada com blocos fáceis de usar, como botões, gráficos, imagens, listas, controles deslizantes, interruptores ou um teclado. A biblioteca é gratuita e totalmente open source.

Figura 1: Aplicação usando ESP32 e biblioteca gráfica LittlevGL.

Principais recursos

- Blocos / Widgets de construção poderosos: botões, gráficos, listas, controles deslizantes, imagens etc.
- Efeitos gráficos avançados: animações, anti-aliasing, opacidade, rolagem suave, etc.
- Suporta vários dispositivos de entrada: touchpad, mouse, teclado, codificador, etc.
- Suporte multilíngue: codificação UTF-8.
- Elementos gráficos totalmente personalizáveis.

- Suporte para todos os tipos de microcontroladores e displays (independente de hardware).
- Altamente escalável: pode operar com memória mínima (80 KB Flash, 10 KB RAM).
- Suporte para sistema operacional, memória externa e GPU (opcional).
- Operação de buffer de quadro único com os mesmos efeitos gráficos avançados.
- Escrito em C para compatibilidade máxima (também compatível com C ++).
- Simulador multiplataforma: suporta design de GUI no PC sem hardware embarcado.

Para saber mais sobre os recursos do LittlevGL nos módulos da Espressif visite este repositório no GitHub.

µGFX

O µGFX foi projetado para ser a menor, mais rápida e mais avançada biblioteca incorporada para telas e displays touch, fornecendo tudo o que é necessário para construir uma GUI embutida com todos os recursos. Uma das principais vantagens do µGFX é que ele é leve, porque todos os recursos não utilizados estão desabilitados e não estão vinculados ao binário finalizado. Além disso, o µGFX é modular, portátil e tem seu código fonte completo disponível para todos os usuários, sendo pago apenas para uso comercial.

Principais recursos

- Pequeno e leve.
- Totalmente personalizável e extensível.
- Altamente portátil.
- Suporta monitores monocromáticos, em tons de cinza e coloridos.
- Suporta aceleração de hardware.

- Mais de 50 drivers prontos para uso.
- Escrito em C, mas também pode ser usado com C ++.
- Livre para usos não comerciais.
- Código fonte completo disponível.
- Funciona em sistemas de baixa RAM; buffer de quadros não é necessário para a maioria dos monitores.
- Reentrância totalmente multi-threading; Desenhos na tela podem ocorrer a partir de qualquer thread, a qualquer momento!

Para saber mais sobre os recursos do µGFX nos módulos da Espressif visite este repositório no GitHub.

Aplicações demo e casos de uso

Os resultados das aplicações podem atender diversos cenários, veja abaixo algumas implementações.

Audio player

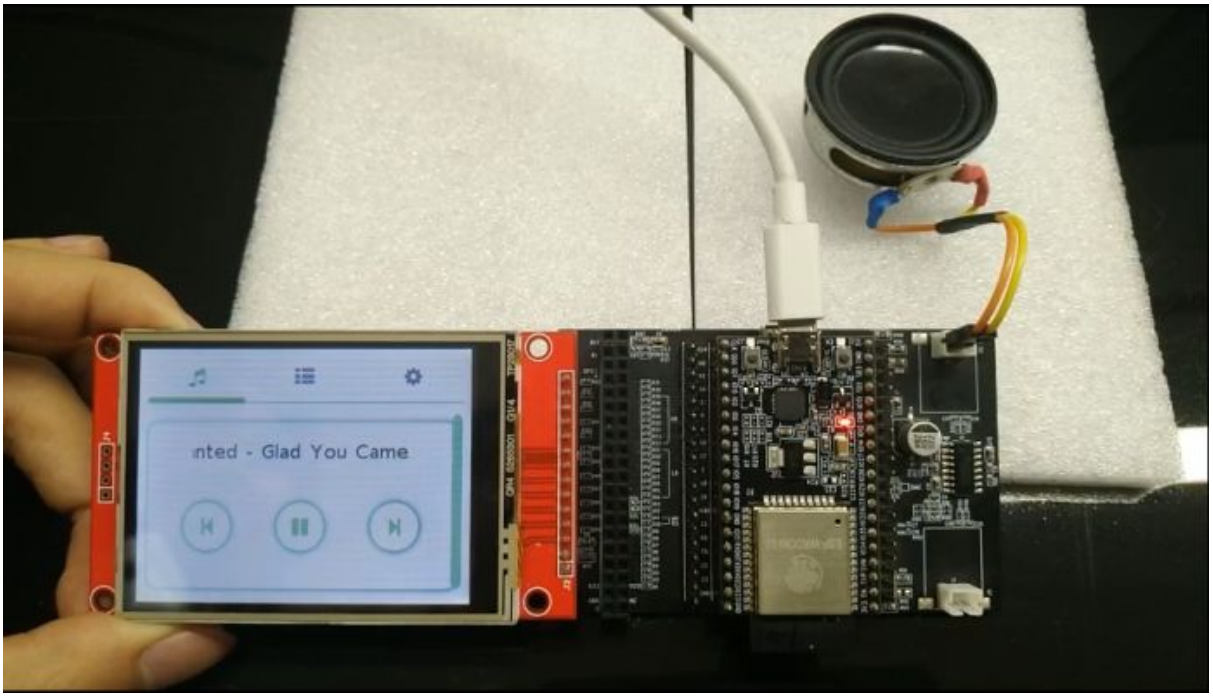


Figura 2: Aplicação do audio player no ESP32.

Clique aqui para ver o vídeo da aplicação no site da Espressif.

Termostato para ar condicionado

Figura 3: Aplicação do termostato no ESP32.

Clique aqui para ver o vídeo da aplicação no site da Espressif.

Painel de controle para máquinas de café

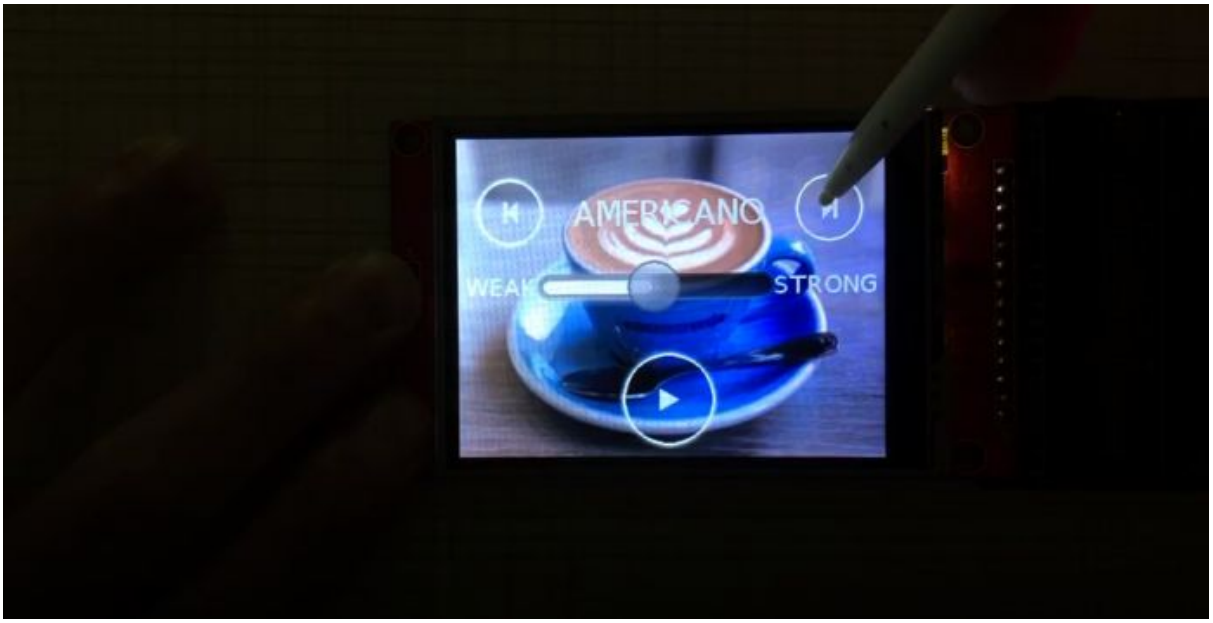


Figura 4: Aplicação do painel para máquinas de café no ESP32.

Clique aqui para ver o vídeo da aplicação no site da Espressif.

Referência

1.ESP32 Modules Now Support LittlevGL and µGFX

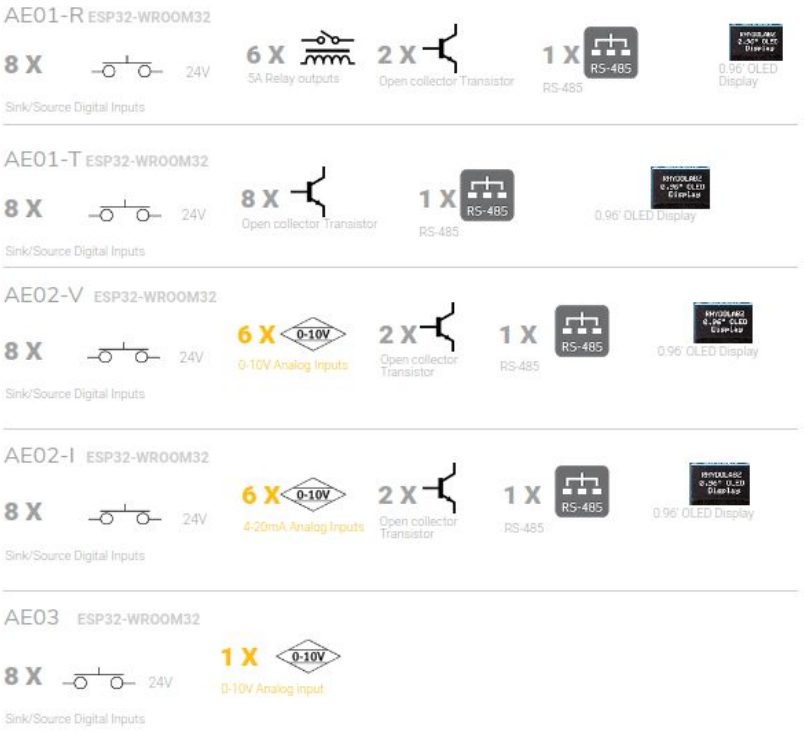
NORVI IIOT - ESP32 para projetos industriais

Autor: Fábio Souza

O ESP32 vem sendo utilizado em diversas aplicações, inclusive industriais. Aproveitando os recursos de conectividade e o poder de processamento, aliados ao baixo custo, permite o desenvolvimento de diversos tipos de controladores e sensores.

Como parte dessa tendência, a empresa Norvi lançou o NORVII IIOT industrial controller, um controlador baseado no módulo ESP32-WROOM-32, que vem com diversos recursos, entre eles uma tela OLED ou TFT de 0,96" a 1,44", montado em uma caixa padrão industrial para montagem em trilhos.

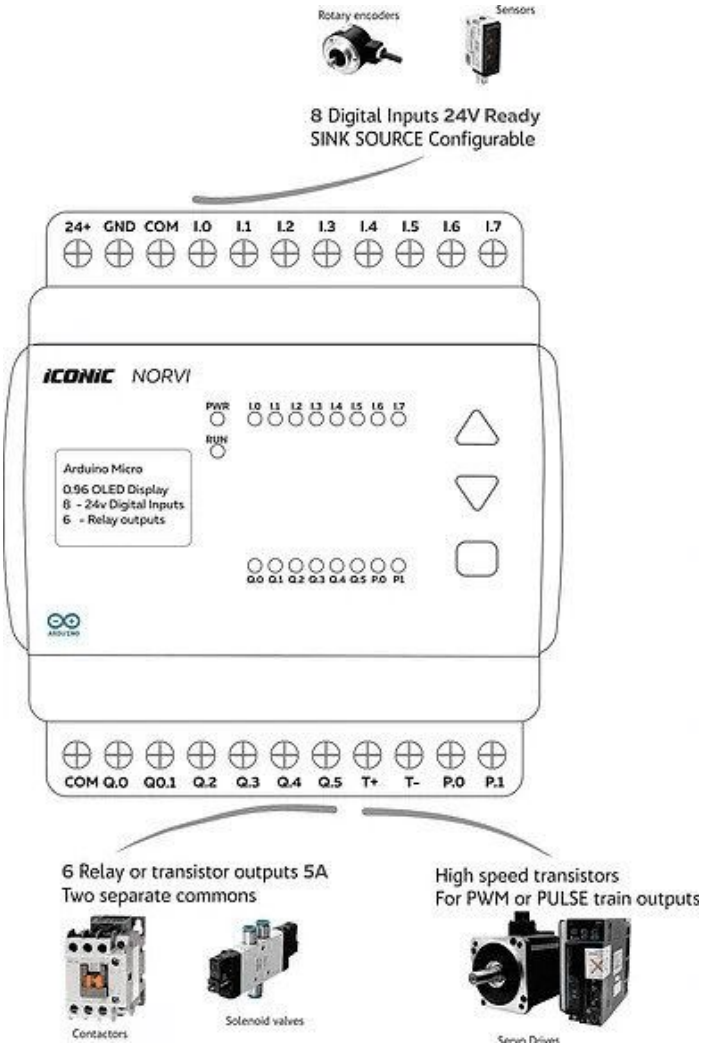
Existem 5 opções de produtos nas três séries disponíveis (AE01, AE02 e AE03):



Resumo de recursos do NORVII IIOT industrial controller

- Módulo sem fio - ESP32-WROOM32 com processador dual-core ESP32 a 160 MHz, SRAM de 520 Kbytes, Flash de 4 Mbit, WiFi 802.11 b/g/n e Bluetooth 4.2
- Armazenamento - slot para cartão microSD opcional
- Display - OLED embutido de 0,96 ". Tela TFT de 0,96 "ou tela TFT de 1,44"
- Comunicação - módulo RS-485, WiFi, Bluetooth, LoRa ou NB-IoT (opcional)
- I/Os:
    - 8x Entradas Digitais
    - 2x saídas de transistor de até 24V
    - Apenas série AE01 - 6x saídas a relé
    - Apenas série AE02 - Entradas analógicas 6x de 16 bits com corrente de 4 - 20mA (AE02-I) / 0 - 10V (AE02-V)
- LED de energia, LED RUN, 8x LEDs para I0 ..... I7, 6x LEDs R0 ..... R5, 2x LEDs para T0... .T1
- Sensor de temperatura MAX31856 (opcional)
- DS3231 RTC opcional com backup de bateria
- Fonte de alimentação - entrada nominal de 24 Vcc
- Dimensões - 90,50 x 56,60 x 60,60 mm

Aplicação típica do NORVI IIOT



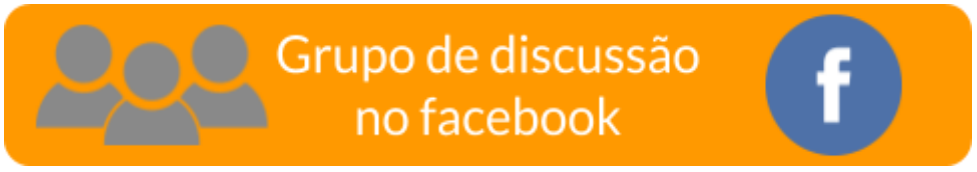
O PLC pode ser programado com o Arduino IDE e a empresa - Iconic Devices - fornece bibliotecas do Arduino e programas exemplos para entradas, saídas, RTC, sensor de temperatura e I2C, etc.

Confira o vídeo de apresentação do produto:

A empresa também fornece módulos de expansão para o NORVII IIOT industrial controller para aplicações LoRA e NB-IOT

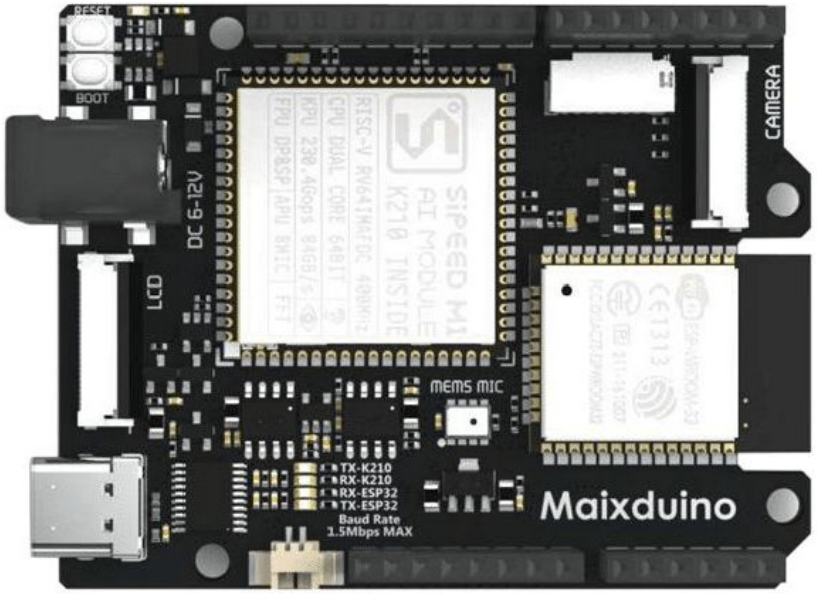
Para mais detalhes, acesse a página do produto: NORVI IIOT - ESP32 for industrial projects

Publicado originalmente no Embarcados, no dia 18/12/2019: link para o artigo original, sob a Licença Creative Commons Atribuição-CompartilhaIgual 4.0 Internacional.



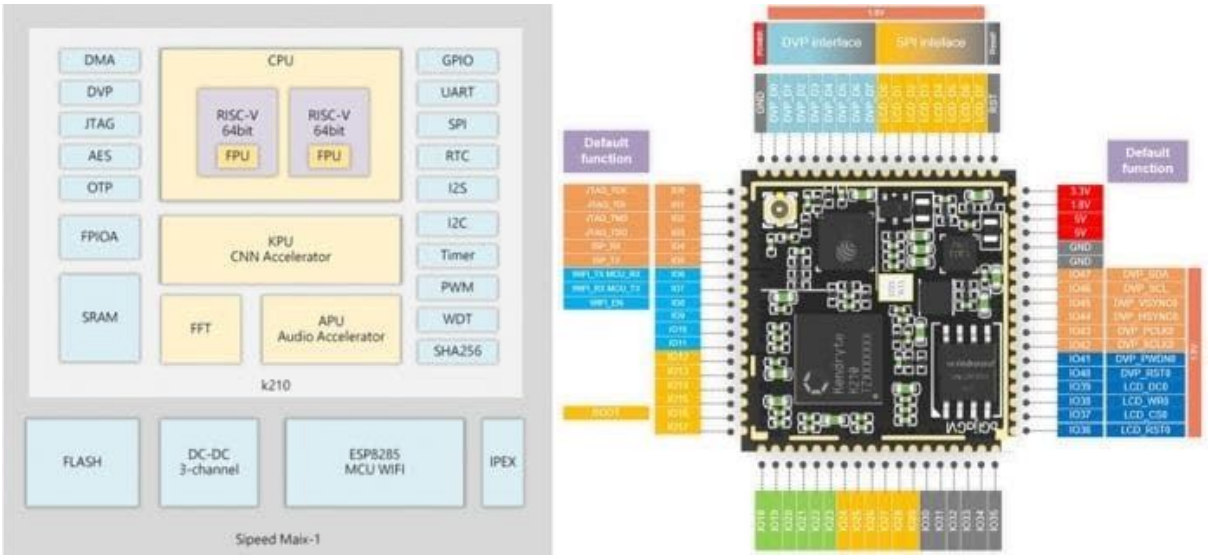
Maixduino: uma super placa com RISC-V AI e ESP32

Autor: Fábio Souza

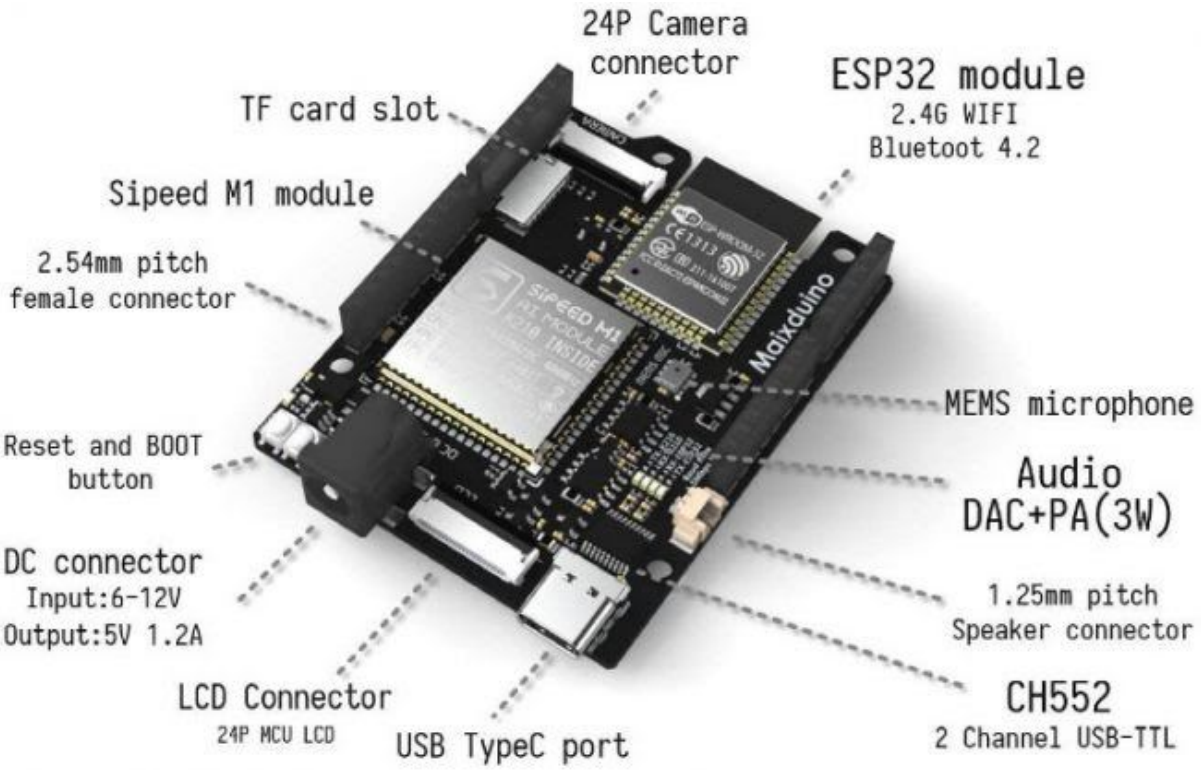


A ISA RISC V vem se tornando popular e algumas aplicações estão sendo criadas com ela. Recentemente foi lançado Kendryte K210, um processador dual core RISC-V de 64 bits, com rede neural e acelerador de áudio, permitindo aplicações de inteligência artifical na

borda, como reconhecimento de objetos e processamento de fala. O processador K120 foi integrado no módulo Sipeed MAIX-I:

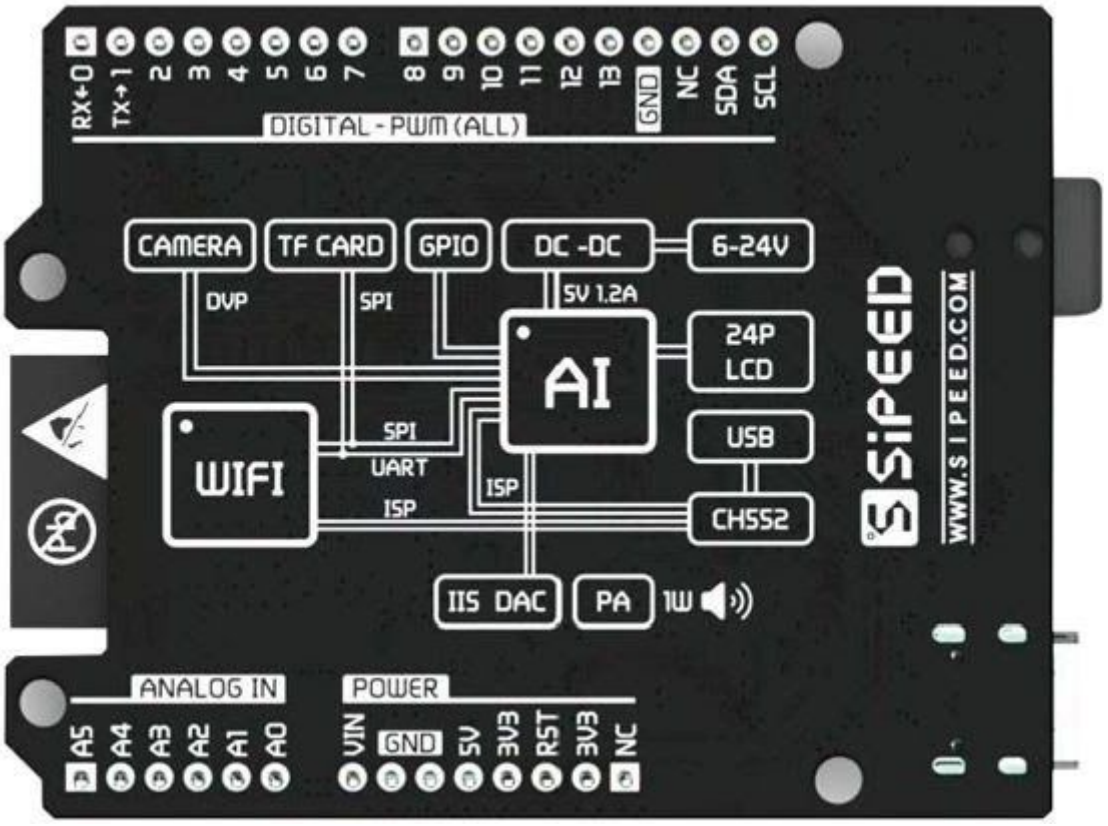


Agora, baseado nesse mesmo módulo, foi lançada a placa Maixduino, uma placa com form factor do Arduino UNO, que traz também um ESP32.



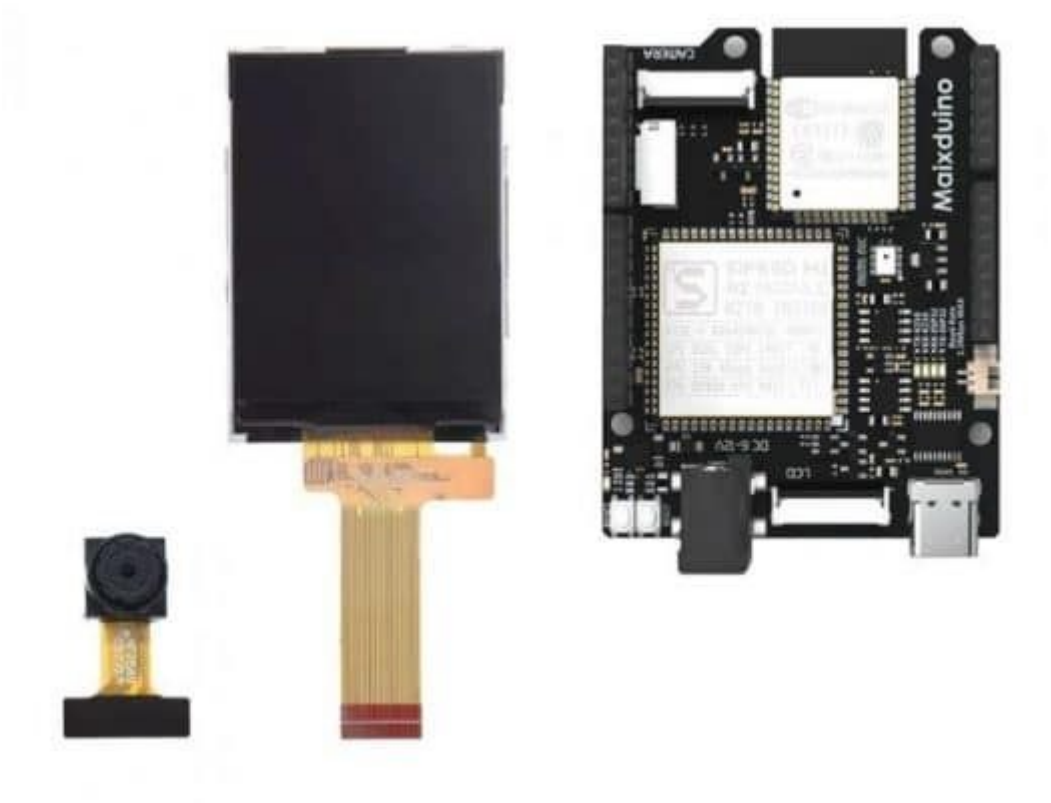
A Maixduino pode ser programada pela IDE maixPy (MicroPython), Arduino IDE, OpenMV IDE e PlatformIO IDE. Tem suporte para Tiny-Yolo, Mobilenet e TensorFlow Lite para deep learning. Também tem suporte para identificação de imagem QVGA @ 60fps or VGA @ 30fps.

A placa ainda traz alguns recursos interessantes, como microfone, conector para câmera, slot para SD card, saída de áudio, conector para LCD, e diversos GPIOs:



Com o foco em prototipagem rápida de soluções com inteligência artificial e reconhecimento de imagens e som na borda, algumas aplicações típicas incluem asa inteligente (limpadores de robô ou alto-falantes inteligentes), dispositivos médicos, Indústria 4.0 (separação inteligente ou monitoramento de equipamentos elétricos), bem como agricultura e educação.

A placa de desenvolvimento Maixduino é vendida hoje como parte de um kit que inclui um módulo de câmera OV2640 e um display TFT de 2,4 polegadas, custando $ 23,90.



O vídeo a seguir exibe uma aplicação feita com Sipeed MAIX-I:

O que achou dessa placa? Deixe seu comentário abaixo.

Considerações Finais

Chegamos ao final do nosso ebook.

Caso você tenha alguma dúvida, não deixe ela sem resposta. Você pode entrar e contato com o autor através da seção de comentários do artigo, ou então participar da nossa comunidade, interagindo com outros profissionais da área:

Comunidade Embarcados no Facebook

Comunidade Embarcados no Telegram

Comunidade Embarcados no LinkedIn

Caso você tenha encontrado algum problema no material ou tenha alguma

sugestão, por favor, entre em contato conosco. Sua opinião é muito importante

para nós: contato@embarcados.com.br

Siga o Embarcados na Redes Sociais


https://www.facebook.com/osembarcados/


https://www.instagram.com/portalembarcados/


https://www.youtube.com/embarcadostv/


https://www.linkedin.com/company/embarcados/


https://twitter.com/embarcados