

部品表

部品番号	型番/値	種類	購入情報(通販コード)	注
A1	AE-FT234X	USB シリアル I/F	秋月 M-08461	
C1	10uF	積層セラミック	秋月 P-03095	5mm ピッチ
C2,C4,C6, C7,C8,C9	0.1uF	積層セラミック	秋月 P-04064	2.54mm ピッチ
C3,C5	1uF	積層セラミック	秋月 P-04066	2.54mm ピッチ
C10,C11	33pF	セラミックコンデンサ	秋月 P-11138	ディスク型
D1,D2,D3	1SS178	ダイオード	秋月 I-07783	
D4		LED		3mm 砲弾型
J1	MJ-4PP-9	3.5φ4 極ジャック	秋月 C-06070	
J2	B4B-XH-A	ボックスピンヘッダ	秋月 C-12249	オプション
J3		2x9 ピンヘッダ		オプション 2.54mm ピッチ
J4	PJ-2694	2.5φ4 極ジャック	aitendo PJ-2694	
JP1	2P ピンヘッダ ジャンパーピン	ピンヘッダ ジャンパーピン	秋月の細ピンヘッダ等 秋月 P03687 等	2.54mm ピッチ
Q1,Q2	DTC143EL	デジタルトランジスタ	秋月 I-12469	
R1,R3,R4, R5,R6*,R7	10kΩ	抵抗 1/6W	秋月 R-16103	
R2	100kΩ	抵抗 1/6W	秋月 R-16104	
R8	1kΩ	抵抗 1/6W	秋月 R-16102	LED 用抵抗
RV1	50kΩ	半固定抵抗	秋月 P-03281	Bourns3362P
RV2,RV3	100kΩ	半固定抵抗	秋月 P-03283	Bourns3362P
U1	ESP-WROOM-32	WiFi モジュール	秋月 M-11647	
U2	TCM3105 IC ソケット 16P	モデムチップ IC ソケット	丹青通商?	
U3	LP2950L-3.3V	レギュレータ	秋月 I-08749	
Y1	水晶 4.4336MHz	水晶発振子	aitendo HC49S	4.433619MHz 選択

*R6 の抵抗値は無線機の仕様に合わせて下さい

J1/J4 と J2 は無線機との接続に合わせてどちらかあれば OK。

この他に無線機との接続用のケーブルが必要。

(J1/J4 を使う場合は、3.5φミニプラグケーブルと 2.5φミニミニプラグケーブル。J2 を使う場合は、ケーブルを自作する必要がある)

※WiFi を使う場合は、U3 を TA48033S に変更(秋月入手可能、向きに注意。C5,C6 も付属のコンデンサに変える)。更に AE-FT234X のリセットブルヒューズを容量の大きいものに取り替える。

※ESP-WROVER-B を使う場合は、C1 を 22uF に替える

半田付け

半田付けは以下の順序で行う。

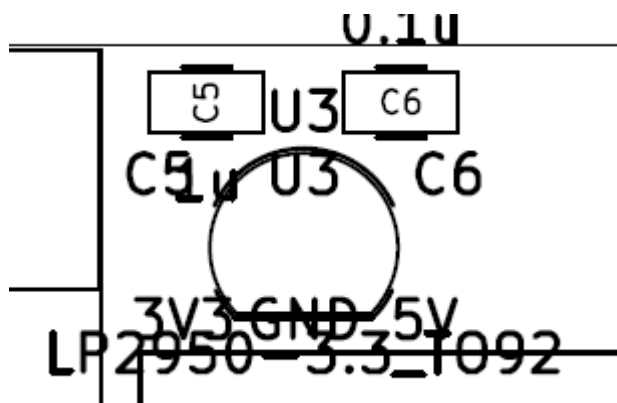
1. WiFi モジュールの取付け

テストで導通を確認する。

2. レギュレータ, C5, C6、USB シリアル、JP1 の取付け

U3 は足が左から OUT(3.3V), GND, IN(5V)となるように取り付ける。

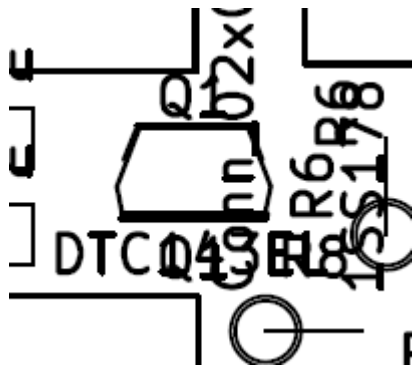
C5 に極性がある場合は右側がマイナスになるように取り付ける。



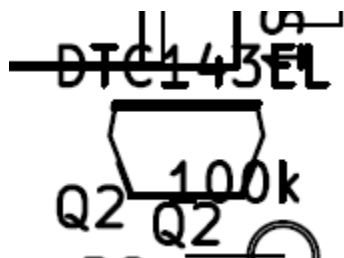
JP1 をショートさせて、WiFi モジュールにファームウェアの書き込みが出来るか確認する。

3. 残りの部品の半田付け

Q1 は型番が読める向きに取り付ける(足が左から 1,2,3 の順)。



Q2 は型番の面を奥に向ける(足が左から 3,2,1 の順)。



ダイオードはランドが四角の側(シルクの○がある側)がカソードとなるように取り付ける。

4. つなぐ無線機に合わせて JP2,JP3,JP4,JP5 の半田ジャンパを設定

アイコム／スタンダード

JP2 : ショート

JP3 : 1-2 間をショート

JP4 : JP4 の 1 と 2.5φ ジャックの端子 2 をジャンパでショートする

JP5 : オープン

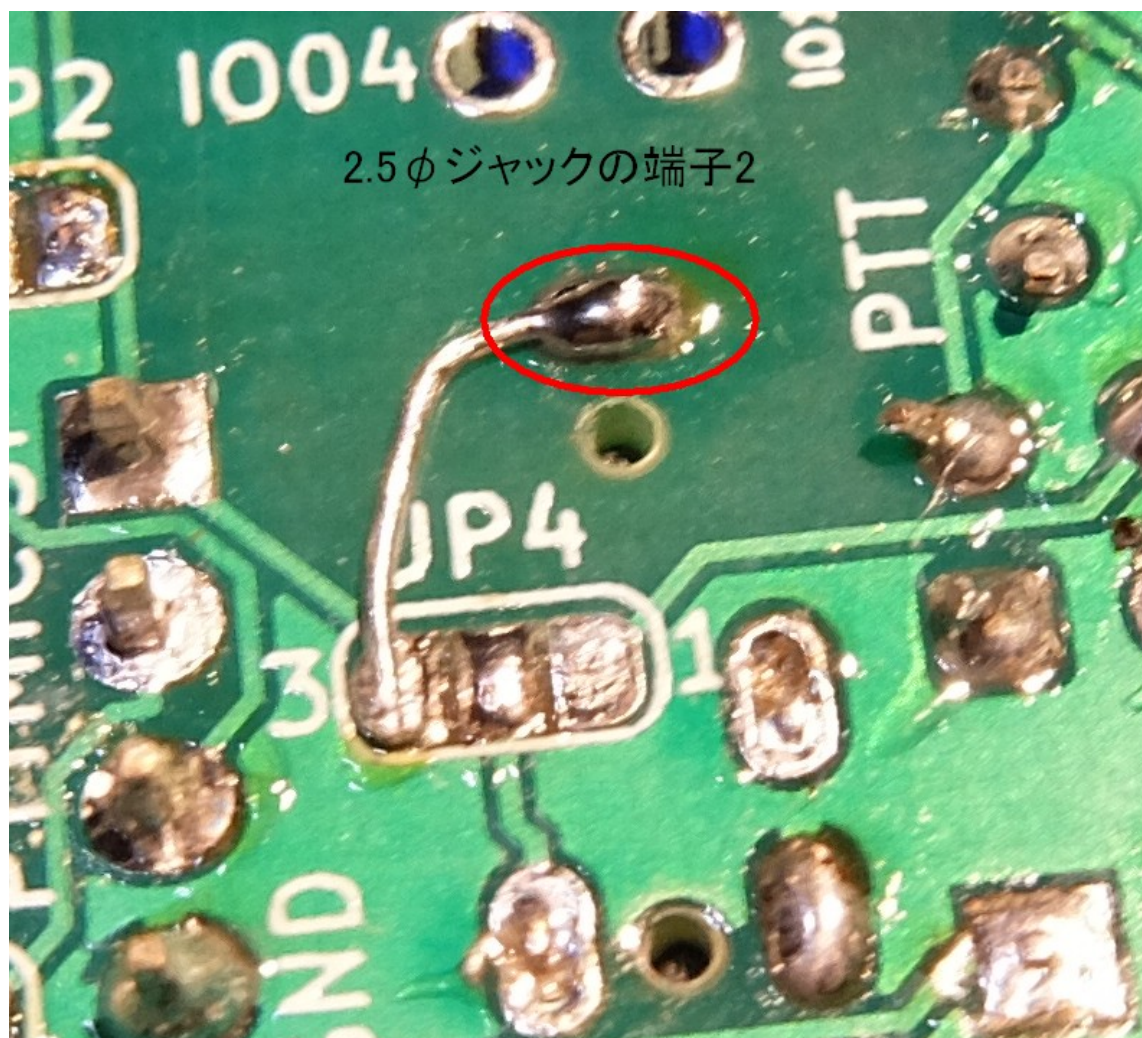
ケンウッド

JP2 : オープン

JP3 : 2-3 間をショート

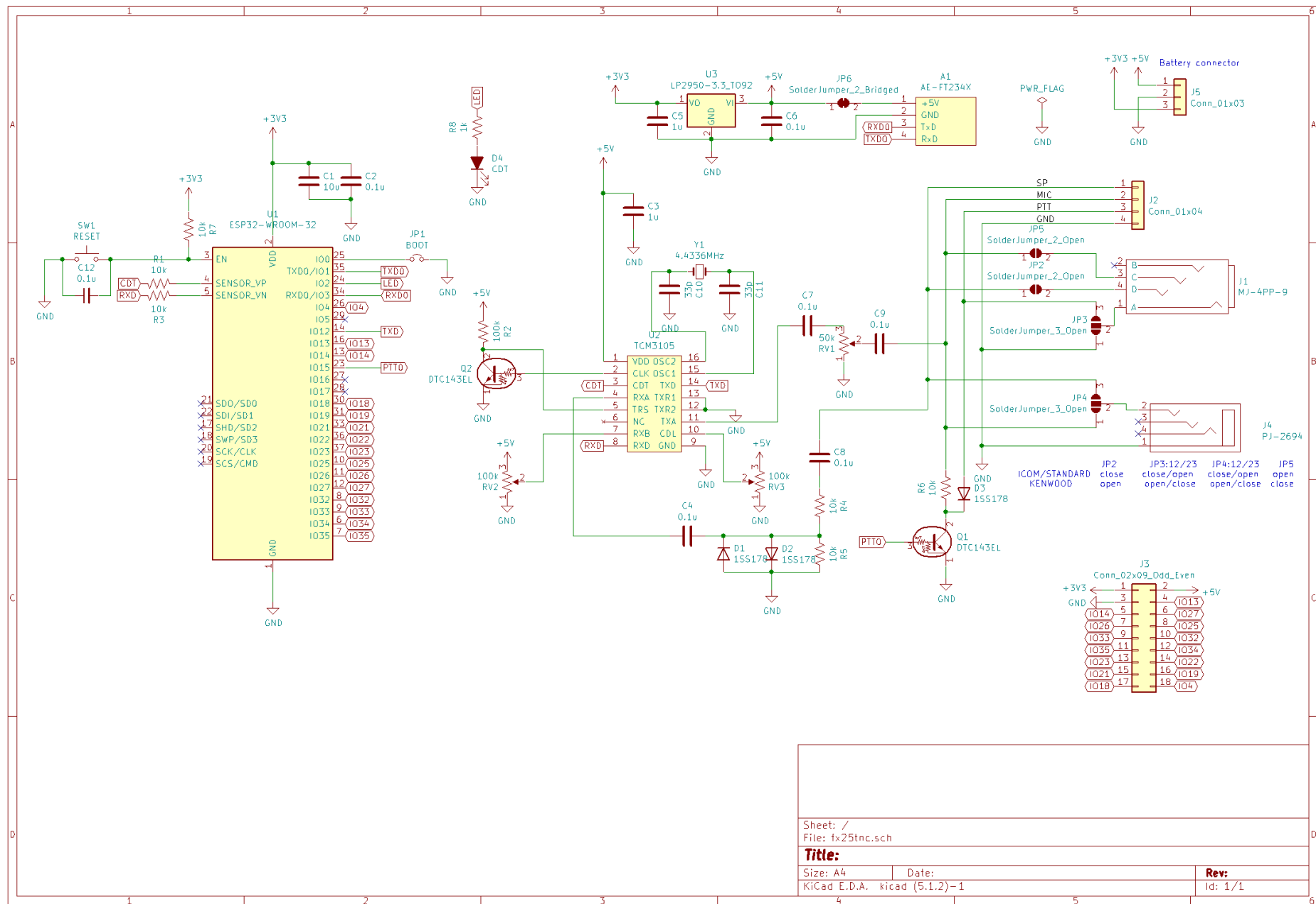
JP4 : JP4 の 3 と 2.5φ ジャックの端子 2 をジャンパでショートする

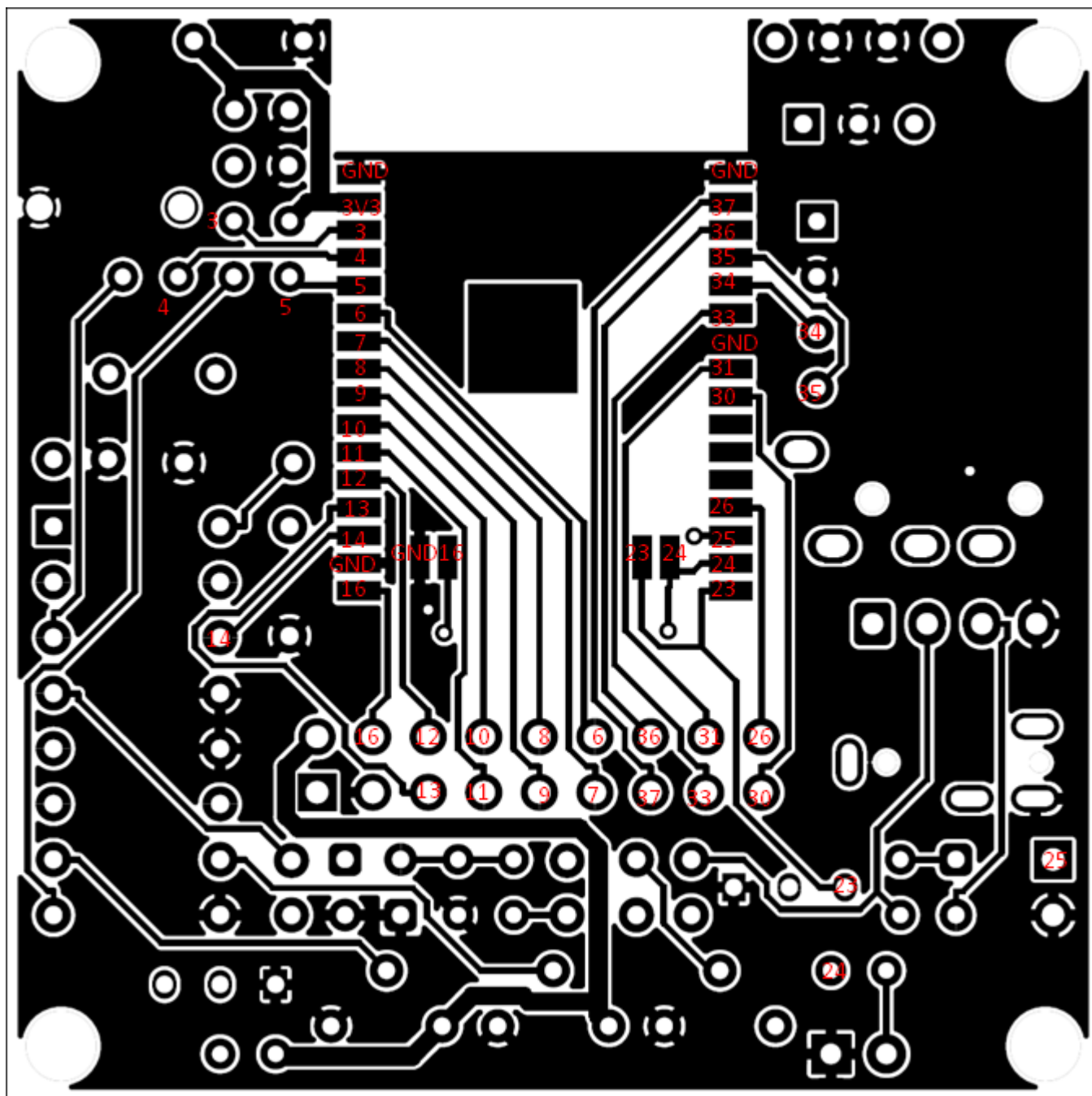
JP5 : ショート



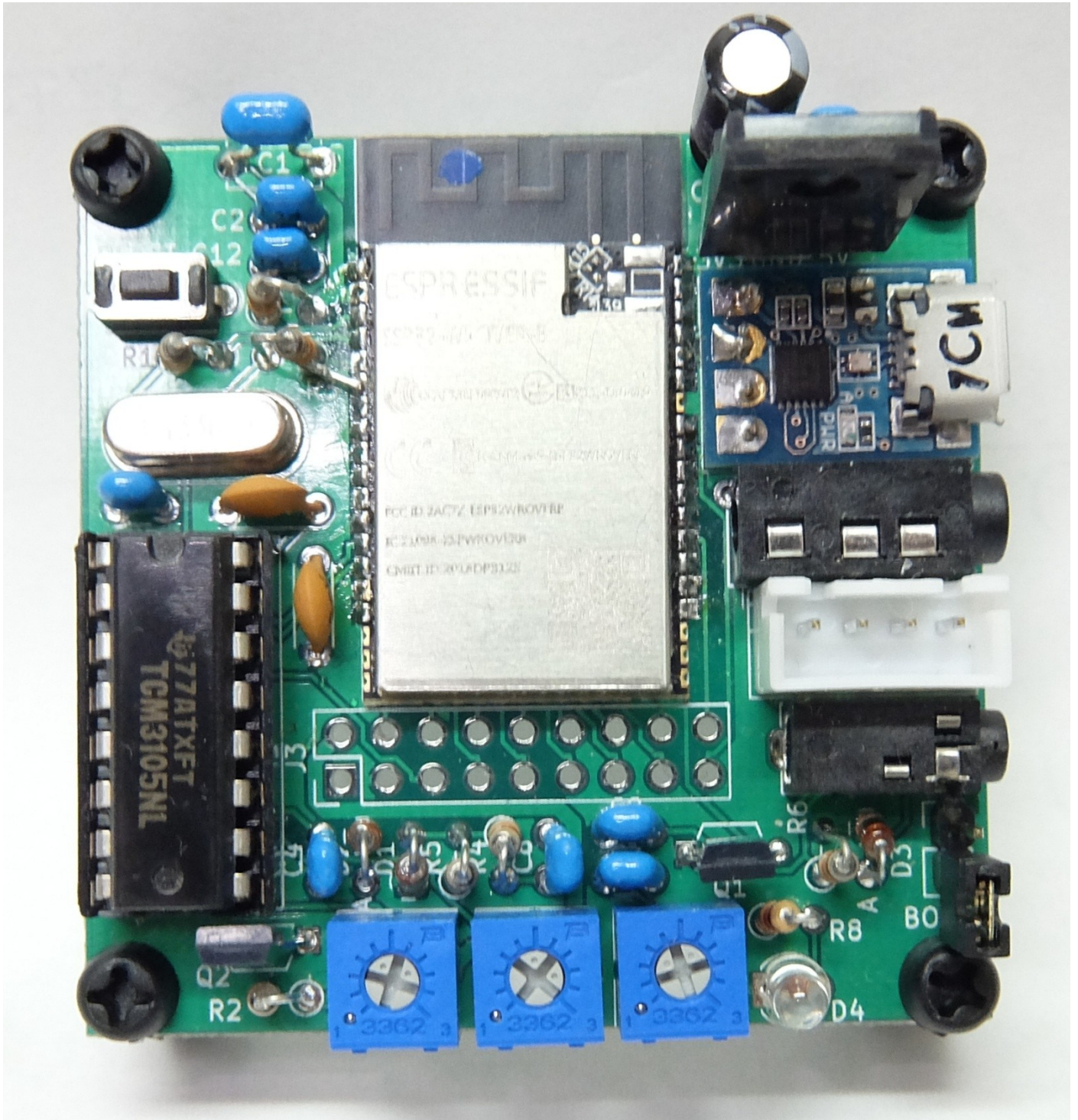
調整

TCM3105 の RXB(7 ピン)を 2.7V に RV2(左)で調整、CDL(10 ピン)を 3.3V に RV3(中央)で調整する。
RV1(右)で送信レベルを調整する。





ESP-WROOM-32 の半田付け確認用接続図



完成品

ファームウェアの書き込み

ファームウェアの書き込みは、WROOM32 の開発環境をインストールして、GitHub から FX25 TNC のソースコードを持ってきて、コンパイルして WROOM32 へシリアル経由で書き込みます。

WROOM32 の開発環境(ESP-IDF)のインストール

下記の URL の Get Started 等を参考に開発環境をインストールします。

ESP-IDF Programming Guide — ESP-IDF Programming Guide

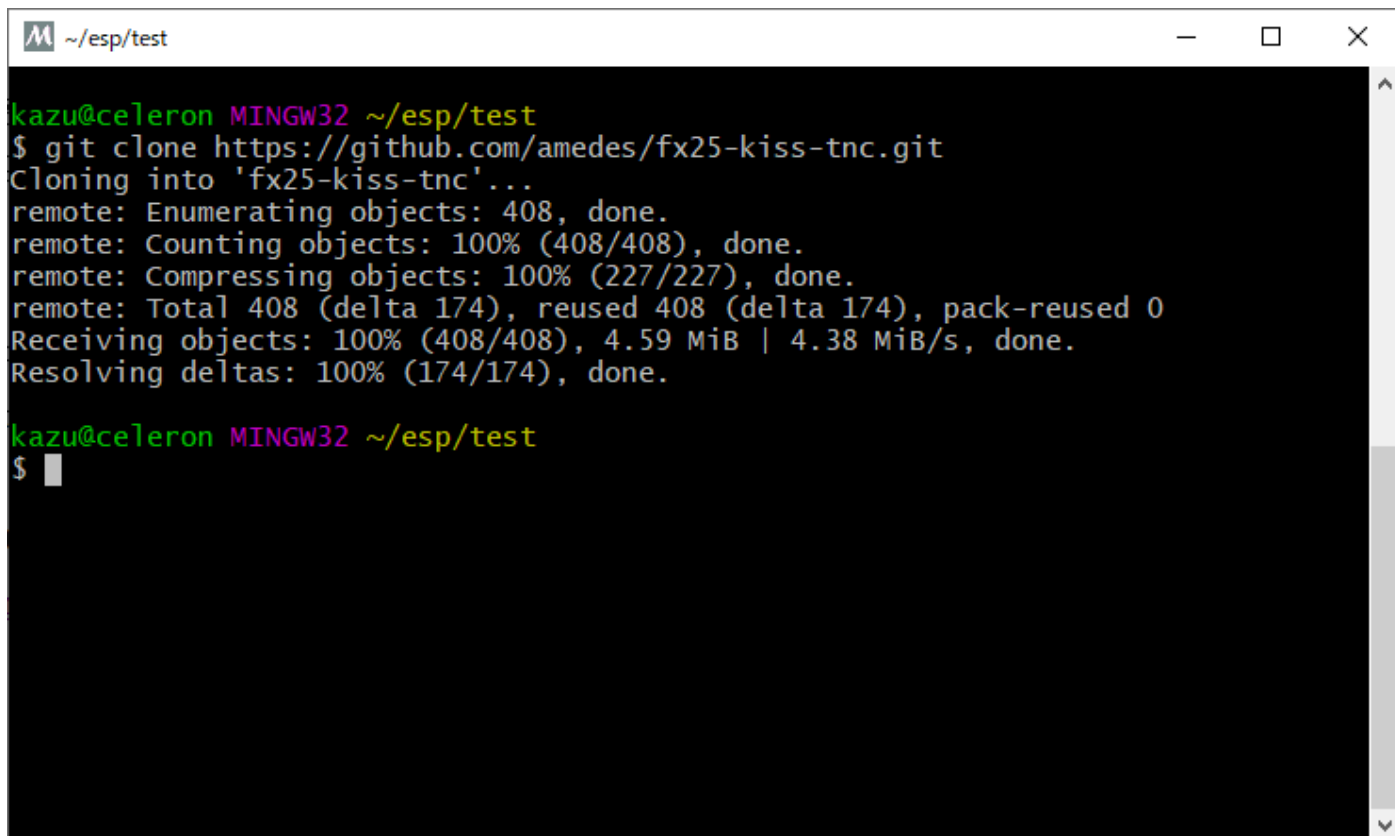
<https://docs.espressif.com/projects/esp-idf/en/latest/>

FX25 TNC のソースコードの入手

ESP-IDF の shell 上で、

```
git clone https://github.com/amedes/fx25-kiss-tnc.git
```

とすると、fx25-kiss-tnc というディレクトリ以下にソースが入ります。

A screenshot of a terminal window titled '~ /esp/test'. The terminal shows the execution of the command 'git clone https://github.com/amedes/fx25-kiss-tnc.git'. The output indicates that the repository is being cloned into 'fx25-kiss-tnc'. It shows progress for enumerating, counting, and compressing objects, and finally receiving the objects (4.59 MiB) and resolving deltas. The prompt returns to 'kazu@celeron MINGW32 ~/esp/test \$' with a cursor.

```
kazu@celeron MINGW32 ~/esp/test
$ git clone https://github.com/amedes/fx25-kiss-tnc.git
Cloning into 'fx25-kiss-tnc'...
remote: Enumerating objects: 408, done.
remote: Counting objects: 100% (408/408), done.
remote: Compressing objects: 100% (227/227), done.
remote: Total 408 (delta 174), reused 408 (delta 174), pack-reused 0
Receiving objects: 100% (408/408), 4.59 MiB | 4.38 MiB/s, done.
Resolving deltas: 100% (174/174), done.

kazu@celeron MINGW32 ~/esp/test
$
```

ディレクトリ fx25-kiss-tnc へ移動して、

```
make menuconfig
```

とすると、メニュー画面が現れるので、"Serial flasher config"を選んで、"Default serial port"を FX25 TNC のシリアルに合わせて書き換えて下さい。(com2 など)


```
~/esp/test/fx25-kiss-tnc

/home/kazu/esp/test/fx25-kiss-tnc/sdkconfig - Espressif IoT Development Framework

Espressif IoT Development Framework Configuration
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]

  SDK tool configuration --->
  Application manager --->
  Bootloader config --->
  Security features --->
  Serial flasher config --->
  FX.25 KISS TNC Configuration --->
  Partition Table --->
  Compiler options --->
  Component config --->

<Select>  <Exit>  <Help>  <Save>  <Load>
```

```
~/esp/test/fx25-kiss-tnc

/home/kazu/esp/test/fx25-kiss-tnc/sdkconfig - Espressif IoT Development Framework
r+ Serial flasher config

Serial flasher config
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]

  (com2) Default serial port
    Default baud rate (115200 baud) --->
    [*] Use compressed upload
    Flash SPI mode (DIO) --->
    Flash SPI speed (40 MHz) --->
    Flash size (2 MB) --->
    [*] Detect flash size when flashing bootloader
    Before flashing (Reset to bootloader) --->
    After flashing (Reset after flashing) --->
    'make monitor' baud rate (115200 bps) --->

  <Select>  <Exit>  <Help>  <Save>  <Load>
```

“<Exit>”で戻って、必要なら“FX.25 KISS TNC Configuration”でデフォルト値を変更してください。
(todo 詳細説明)

```
~/esp/test/fx25-kiss-tnc
/home/kazu/esp/test/fx25-kiss-tnc/sdkconfig - Espressif IoT Development Framework
r+ FX.25 KISS TNC Configuration

FX.25 KISS TNC Configuration
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]

Default TNC Protocol (FX.25 with Parity 16) --->
Default baud rate (115200 baud) --->
(50) TXDELAY default value
(63) Persistence parameter default value
(10) Slot time default value
[ ] Full duplex
[ ] Enable WiFi
[ ] Enable Beacon
(39) GPIO No. connected to RXD
(36) GPIO No. connected to CDT
.(+)

<Select> <Exit> <Help> <Save> <Load>
```

必要な変更が完了したら、”<Exit>”で menuconfig を終了してください。(終了に時間がかかることがあります)

```
~/esp/test/fx25-kiss-tnc
remote: Total 408 (delta 174), reused 408 (delta 174), pack-reused 0
Receiving objects: 100% (408/408), 4.59 MiB | 4.38 MiB/s, done.
Resolving deltas: 100% (174/174), done.

kazu@celeron MINGW32 ~/esp/test
$ cd fx25-kiss-tnc/

kazu@celeron MINGW32 ~/esp/test/fx25-kiss-tnc
$ make menuconfig
DEFCONFIG
#
# configuration written to /home/kazu/esp/test/fx25-kiss-tnc/sdkconfig
#
MENUCONFIG
configuration written to /home/kazu/esp/test/fx25-kiss-tnc/sdkconfig

*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

GENCONFIG
App "fx25-kiss-tnc" version: cc65f26

kazu@celeron MINGW32 ~/esp/test/fx25-kiss-tnc
$ make
```

make

とすると、コンパイルがはじまります。

ライブラリ等のコンパイルも行われるため時間がかかります(10 分ぐらい?)。

```
~/esp/test/fx25-kiss-tnc
Generating libspi_flash.a.sections_info
Generating libspiffs.a.sections_info
Generating libtcp_transport.a.sections_info
Generating libtcp_adapter.a.sections_info
Generating libulp.a.sections_info
Generating libunity.a.sections_info
Generating libvfs.a.sections_info
Generating libwear_levelling.a.sections_info
Generating libwifi_provisioning.a.sections_info
Generating libwpa_supPLICANT.a.sections_info
Generating libxtensa-debug-module.a.sections_info
Generating esp32.project.ld
LD build/fx25-kiss-tnc.elf
esptool.py v2.6
To flash all build output, run 'make flash' or:
python /home/kazu/esp/esp-idf/components/esptool_py/esptool/esptool.py --chip es
p32 --port com2 --baud 115200 --before default_reset --after hard_reset write_fl
ash -z --flash_mode dio --flash_freq 40m --flash_size detect 0x1000 /home/kazu/e
sp/test/fx25-kiss-tnc/build/bootloader/bootloader.bin 0x10000 /home/kazu/esp/tes
t/fx25-kiss-tnc/build/fx25-kiss-tnc.bin 0x8000 /home/kazu/esp/test/fx25-kiss-tnc
/build/partitions_singleapp.bin

kazu@celeron MINGW32 ~/esp/test/fx25-kiss-tnc
$ make flash
```

コンパイルが終わったら、FX25 TNC を USB でつないで、JP1 をショートした状態でリセットボタンを押します。

```
make flash
```

とするとファームウェアの書き込みが行われます。

```
~/esp/test/fx25-kiss-tnc
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Flash params set to 0x0220
Compressed 24464 bytes to 14628...
Wrote 24464 bytes (14628 compressed) at 0x00001000 in 1.3 seconds (effective 150
.3 kbit/s)...
Hash of data verified.
Compressed 192608 bytes to 95575...
Wrote 192608 bytes (95575 compressed) at 0x00010000 in 8.5 seconds (effective 18
1.5 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 103...
Wrote 3072 bytes (103 compressed) at 0x00008000 in 0.0 seconds (effective 786.4
kbit/s)...
Hash of data verified.

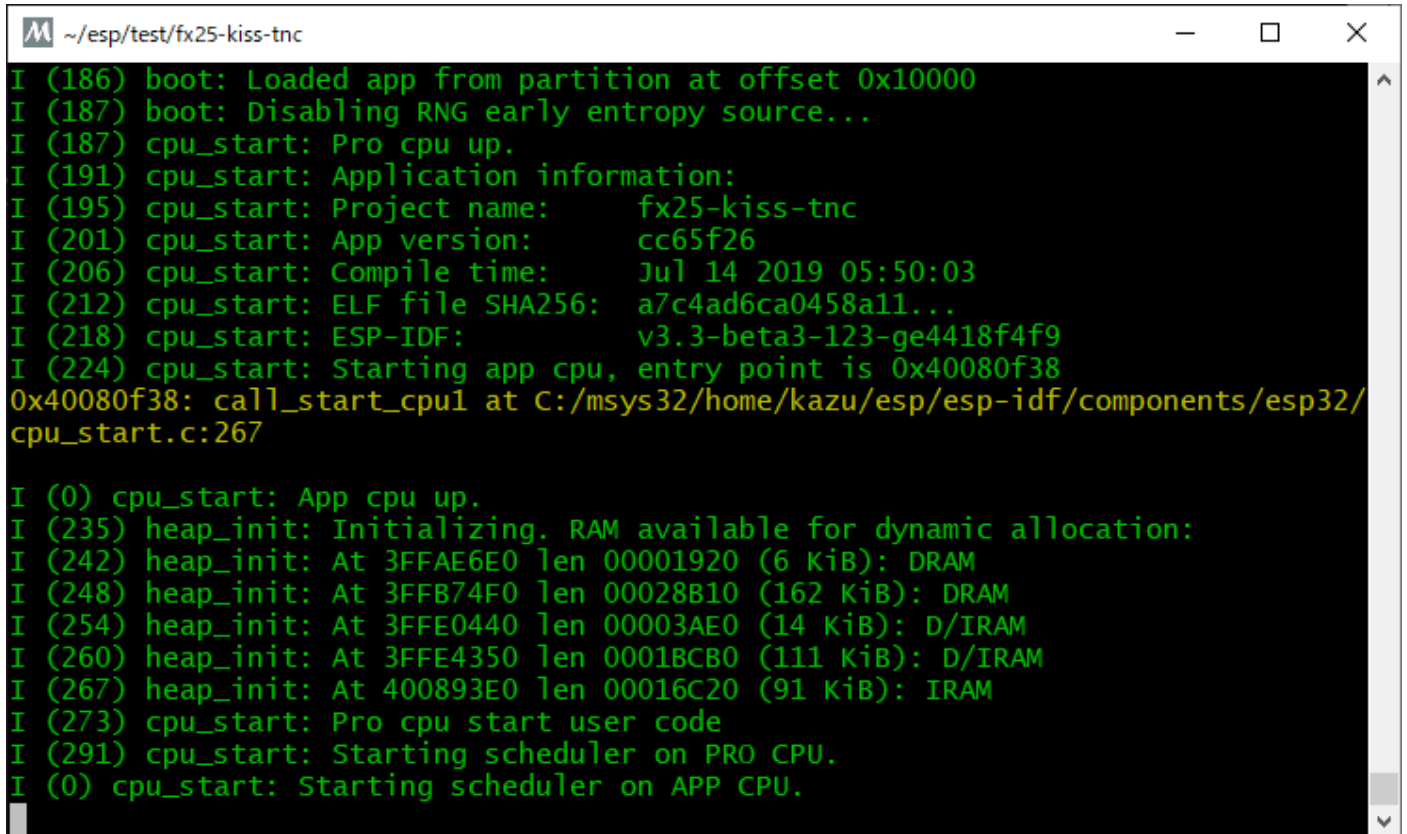
Leaving...
Hard resetting via RTS pin...

kazu@celeron MINGW32 ~/esp/test/fx25-kiss-tnc
$ make monitor
```

JP1 をオープンにしてリセットすると TNC が動作を始めます。

make monitor

とするとシリアルを監視できます。リセットするとブートメッセージが表示されるはずです。抜けるには Control-] です。

A screenshot of a terminal window titled '~/.esp/test/fx25-kiss-tnc'. The terminal displays a series of boot logs in green text on a black background. The logs show the boot process starting with 'boot: Loaded app from partition at offset 0x10000', followed by 'boot: Disabling RNG early entropy source...'. Then, 'cpu_start: Pro cpu up.' is shown. Next, 'cpu_start: Application information:' is displayed, followed by several lines of application details: 'Project name: fx25-kiss-tnc', 'App version: cc65f26', 'Compile time: Jul 14 2019 05:50:03', 'ELF file SHA256: a7c4ad6ca0458a11...', and 'ESP-IDF: v3.3-beta3-123-ge4418f4f9'. The logs then show 'Starting app cpu, entry point is 0x40080f38' and a call to 'call_start_cpu1' at a specific address. Finally, 'cpu_start: App cpu up.' is shown, followed by 'heap_init: Initializing. RAM available for dynamic allocation:' and several lines of memory allocation details for DRAM and IRAM. The logs end with 'cpu_start: Pro cpu start user code', 'Starting scheduler on PRO CPU.', and 'Starting scheduler on APP CPU.'.

```
I (186) boot: Loaded app from partition at offset 0x10000
I (187) boot: Disabling RNG early entropy source...
I (187) cpu_start: Pro cpu up.
I (191) cpu_start: Application information:
I (195) cpu_start: Project name:      fx25-kiss-tnc
I (201) cpu_start: App version:      cc65f26
I (206) cpu_start: Compile time:     Jul 14 2019 05:50:03
I (212) cpu_start: ELF file SHA256:  a7c4ad6ca0458a11...
I (218) cpu_start: ESP-IDF:         v3.3-beta3-123-ge4418f4f9
I (224) cpu_start: Starting app cpu, entry point is 0x40080f38
0x40080f38: call_start_cpu1 at C:/msys32/home/kazu/esp/esp-idf/components/esp32/
cpu_start.c:267

I (0) cpu_start: App cpu up.
I (235) heap_init: Initializing. RAM available for dynamic allocation:
I (242) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (248) heap_init: At 3FFB74F0 len 00028B10 (162 KiB): DRAM
I (254) heap_init: At 3FFE0440 len 00003AE0 (14 KiB): D/IRAM
I (260) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (267) heap_init: At 400893E0 len 00016C20 (91 KiB): IRAM
I (273) cpu_start: Pro cpu start user code
I (291) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.
```

以上