Visual Code - ESP32 debug

For successful debugging/uploading firmware to ESP32 via JTAG it is necessary to leave GPIO 12-14 untouched. Because then collisions occur.

For mac OS it is convenient to install **brew install libusb**

Development environment and installed extensions from Marketplace

Visual Code	
PlatformIO IDE	
CMake	
CMake Tools	
Cortex-Debug	
GNU debugger	
C/C++, C/C++ Extension Pack	

Project settings

example:

```
[env:esp32dev]
platform = espressif32
board = esp32dev
framework = arduino
upload_port = /dev/cu.usbserial-1443100
monitor_speed = 115200
board_build.mcu = esp32
board_build.f_cpu = 240000000L
debug_tool = esp-prog
upload_protocol = esp-prog
```

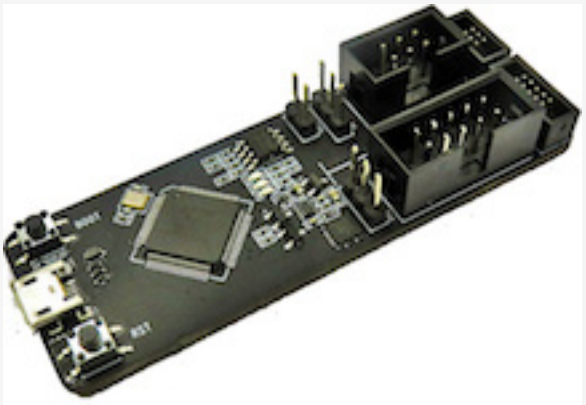
`platform`	
`board`	
`framework`	
`upload_port`	
`monitor_speed`	
`board_build.mcu`	
`board_build.f_cpu`	
`debug_tool`	

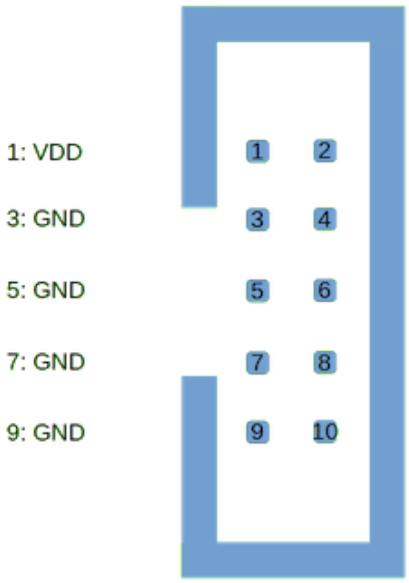
`upload_protocol`	
`debug_init_break`	**empty string** = disable initial breakpoint **tbreak app_main**= stops in app_main **tbreak loop** = stops in the void loop() **break main.cpp:13** = stop in main.cpp at line 13 **tbreak Reset_Handler** - stops at void Reset_Handler(void)

Programmers - JTAG

ESP-PROG - best for ESP32

The programmer includes a JTAG interface and a serial link interface. The programmer is based on FTDI chip, where it is announced to the system as 2 serial lines. The first serial line is used for JTAG and the following one is for the serial programmer.	Example:
	
	`/dev/cu.usbserial-1443100 - JTAG`
	`/dev/cu.usbserial-1443101 - SERIAL`

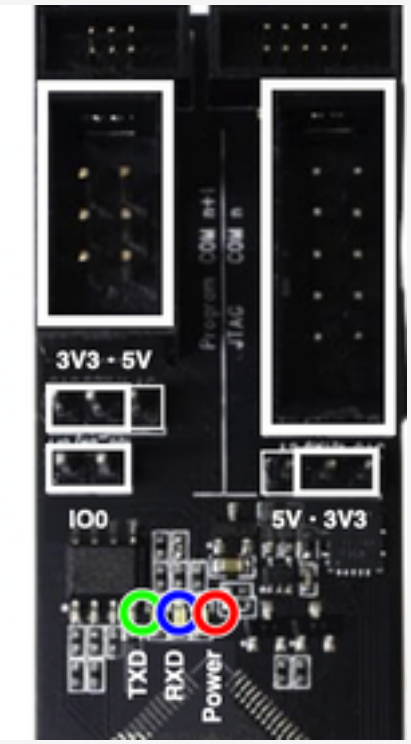
	`[env:esp32dev]`
	`platform = espressif32`
	`board = esp32dev`
	`framework = arduino`
	`upload_port =`
	`/dev/cu.usbserial-1443100`
	`monitor_speed = 115200`
	`board_build.mcu = esp32`
	`board_build.f_cpu = 240000000L`
	`debug_tool = esp-prog`
	`upload_protocol = esp-prog`



JTAG - ESP32

VDD - 3V3 (connect only if ESP32 is not powered from USB or app.)

GND - GND
TMS - GPIO 14
TCK - GPIO 13
TDO - GPIO 15
TDI - GPIO 12



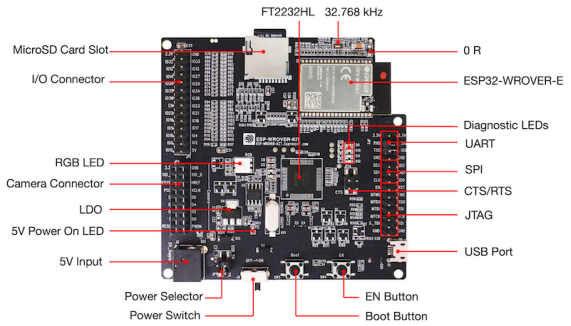
Program			
IO0	6	5	RXD
GROUND	4	3	TXD
VDD	2	1	RESET

VTref	VDD	SUPPLY
3V3		5V

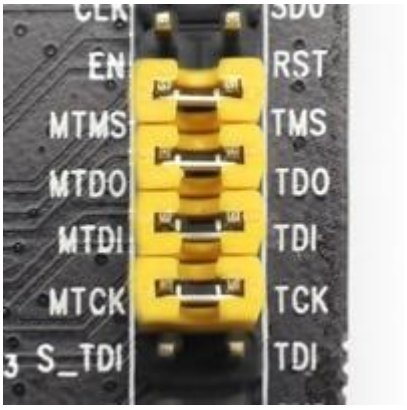
JTAG			
NC	10	9	GROUND
TDI	8	7	GROUND
TDO	6	5	GROUND
TCK	4	3	GROUND
TMS	2	1	VDD

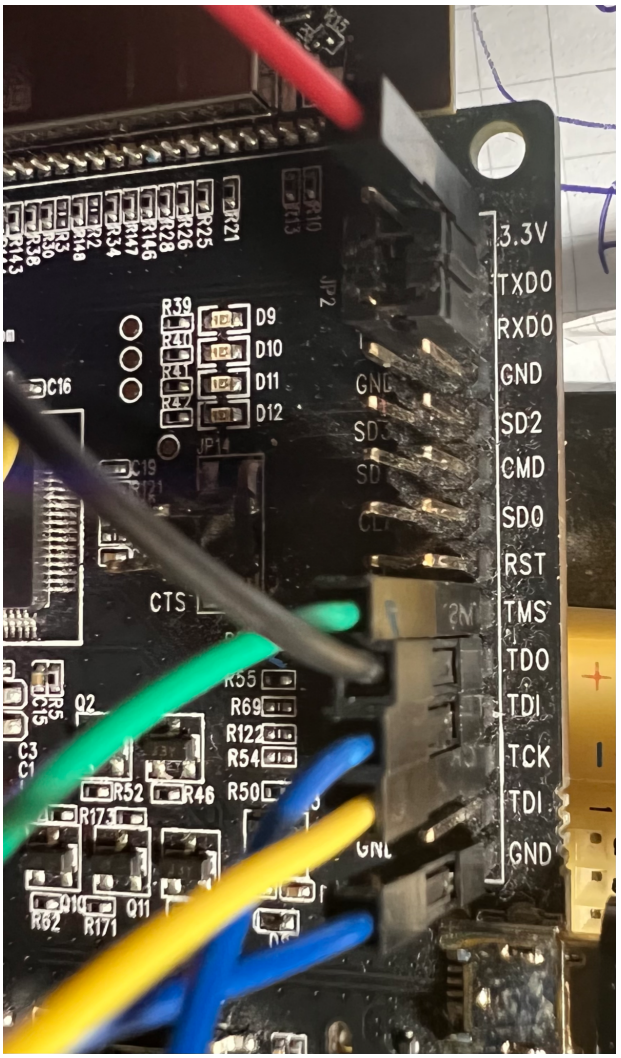
SUPPLY	VDD	VTref
5V		3V3

ESP-WROVER-KIT



Disconnect the jumpers for JTAG from the development kit and connect them to the developed application. Only the JTAG programmer is used



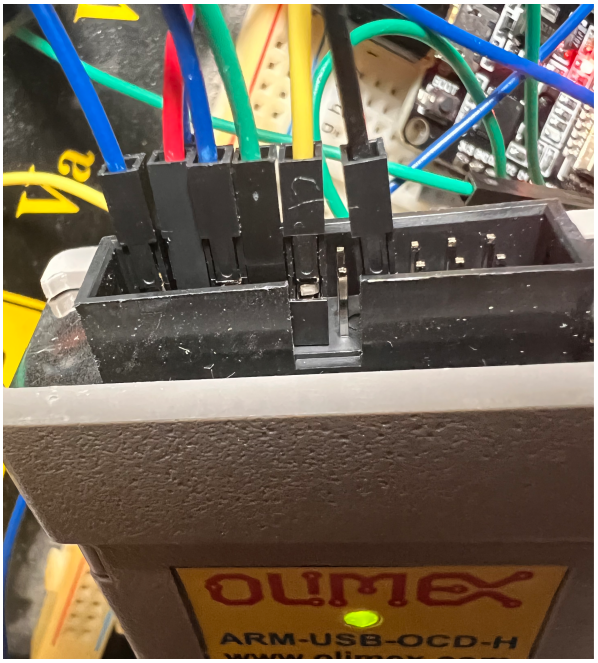
<table>
<tr>
<td> 
</td>
<td>JTAG - ESP32

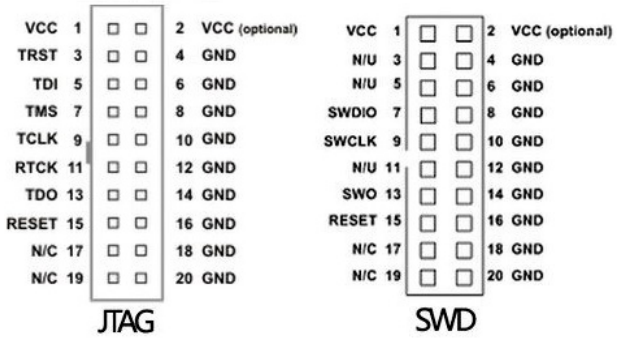
VDD (1) - 3V3 (connect only if ESP32 is not powered from USB or app.)

GND (4) - GND
TMS (7) - GPIO 14
TCK (9) - GPIO 13
TDO (13) - GPIO 15
TDI (5) - GPIO 12</td>
</tr>
<tr>
<td></td>
<td><pre>
[env:esp32dev]
platform = espressif32
board = esp32dev
framework = arduino
upload_port =
/dev/cu.usbserial-1443100
monitor_speed = 115200
board_build.mcu = esp32
board_build.f_cpu = 240000000L
debug_tool = esp-prog
upload_protocol = esp-prog
</pre></td>
</tr>
<tr>
<td>The programmer includes a JTAG interface and a serial link interface. The programmer is based on FTDI chip, where it is announced to the system as 2 serial lines. The first serial line is used for JTAG and the following one is for the serial programmer.</td>
<td>Example:

<pre>
/dev/cu.usbserial-1443100 - JTAG
/dev/cu.usbserial-1443101 - SERIAL
</pre></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</table>

OLIMEX - ARM-USB-OCD-H





SWD - only with special module !!

JTAG - ESP32

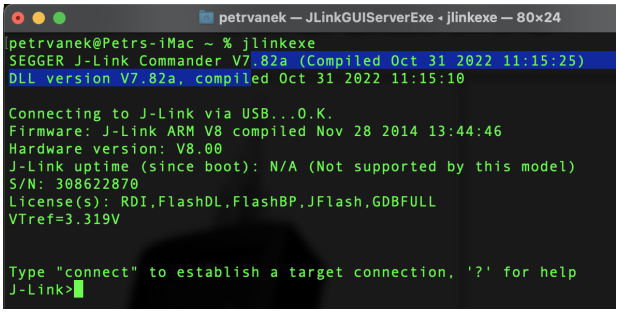
TARGET_POWER (19) - 5V!!! - it does not supply enough current to power the ESP32, do not use!

VDD (1) - 3V3 - must be connected to the application board !
GND (4) - GND
TMS (7) - GPIO 14
TCK (9) - GPIO 13
TDO (13) - GPIO 15
TDI (5) - GPIO 12
TRST (3) - RESET (EN)
Note:
This programmer requires power from the application. And ESP32 must have sufficient power supply. Otherwise, the CPU ID will be reset.

For this programmer it is not necessary to enter the serial port, it finds it by VID, PID.

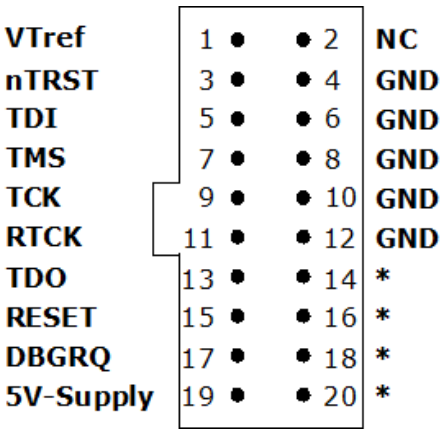
```
[env:esp32dev]
platform = espressif32
board = esp32dev
framework = arduino
monitor_speed = 115200
board_build.mcu = esp32
board_build.f_cpu = 240000000L
debug_tool = olimex-arm-usb-ocd-h
upload_protocol = olimex-arm-usb-ocd-h
```


JLINK - SEGGER V8



installed drivers from SEGGER

Note:
Sometimes the problem is that the debugging information is not linked with the current position in the code. I don't know how to solve it yet.



JTAG - ESP32

TARGET_POWER (19) - it does not supply enough current to power the ESP32, do not use!

VDD (1) -
GND (4) - GND
TMS (7) - GPIO 14
TCK (9) - GPIO 13
TDO (13) - GPIO 15
TDI (5) - GPIO 12
TRST (3) - RESET (EN)

	[env:esp32dev] platform = espressif32 board = esp32dev framework = arduino monitor_speed = 115200 board_build.mcu = esp32 board_build.f_cpu = 240000000L **upload_protocol = jlink** **debug_tool = jlink**

Visual Code - RP2040 debug

For the debug and upload of the RP2040, the SWD (Serial Wire Debug) must be used.

MAC Prepare

```
$ brew doctor
$ brew update
$ brew upgrade

$ brew install armmbed/formulae/arm-none-eabi-gcc


# cd into a directory where you want to store the files
$ git clone -b master https://github.com/raspberrypi/pico-sdk.git
$ cd pico-sdk
$ git submodule update --init

# Get the example code
$ cd ..
$ git clone -b master
https://github.com/raspberrypi/pico-examples.git


$ cd pico-examples
$ mkdir build
```

```
$ cd build

# Set the path to where the SDK can be found
$ export PICO_SDK_PATH=../../pico-sdk

# Get the build directory ready for cmake
$ cmake ..


$ cd blink
$ make -j4
...
[100%] Linking CXX executable blink.elf
[100%] Built target blink
```

PICO Probe

firmware fo rp2040 RaspberryPico as WSD - https://github.com/raspberrypi/picoprobe

```
# You might need to set the PICO_SDK_PATH again if you closed the
terminal at some point
# export PICO_SDK_PATH=../../pico-sdk

$ git clone https://github.com/raspberrypi/picoprobe.git
$ cd picoprobe
$ mkdir build
$ cd build
$ cmake ..
$ make -j4
...
[100%] Linking CXX executable picoprobe.elf
[100%] Built target picoprobe
```

Press BOOTSEL and uload picoprobe image into RP2040 from /build/picoprobe.uf2

OpenOCD

```
$ brew install libtool automake libusb wget pkg-config gcc texinfo
...
# Note this output
If you need to have texinfo first in your PATH, run:
  echo 'export PATH="/usr/local/opt/texinfo/bin:$PATH"' >> ~/.zshrc

$ export PATH="/usr/local/opt/texinfo/bin:$PATH"
$ git clone https://github.com/raspberrypi/openocd.git --branch
picoprobe --depth=1
```

```
$ cd openocd
$ ./bootstrap
$ ./configure --enable-picoprobe --disable-werror
$ make -j4
$ make install

sometimes for translation it is necessary to disable nullable
export CFLAGS=-Wno-nullability-completeness
```

Start openOCD

Connect PicoProbe with RP2040 via SWD and start openocd.

```
openocd -f interface/picoprobe.cfg -f target/rp2040.cfg -s tcl
```

After testing openOCD you can run the following. OpenOcd must boot and display the TCP ports for debugging. For example. "Listening on port 3333 for gdb connections" **If everything is ok, stop running with CTRL+C and return to VisualCode.**

Build project in Visual Code

Select **active kit** (e.g GCC 10.3.1 arm-none-eabi) and **Build**

after the build the resulting files in the build directory are created
`project-name/build/project-name.uf2`
`project-name/build/project-name.elf`
and others

Debug project in Visual Code

`.vscode/settings.json`
sets cortex-debug.openocdPath if not on the path

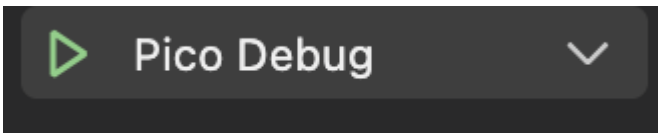
```
{

   "cmake.statusbar.advanced": {
       "debug": {
           "visibility": "hidden"
       },
```

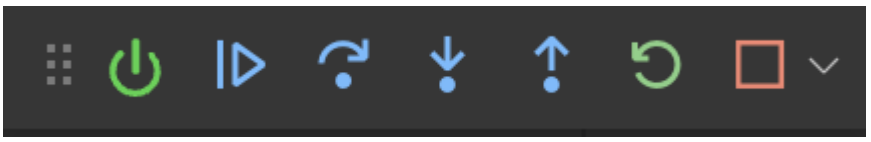
```
        "launch": {
            "visibility": "hidden"
        },
        "build": {
            "visibility": "default"
        },
        "buildTarget": {
            "visibility": "hidden"
        }
    },
    "cmake.buildBeforeRun": true,
    "C_Cpp.default.configurationProvider": "ms-vscode.cmake-tools",
    "cortex-debug.openocdPath": ""
}
```

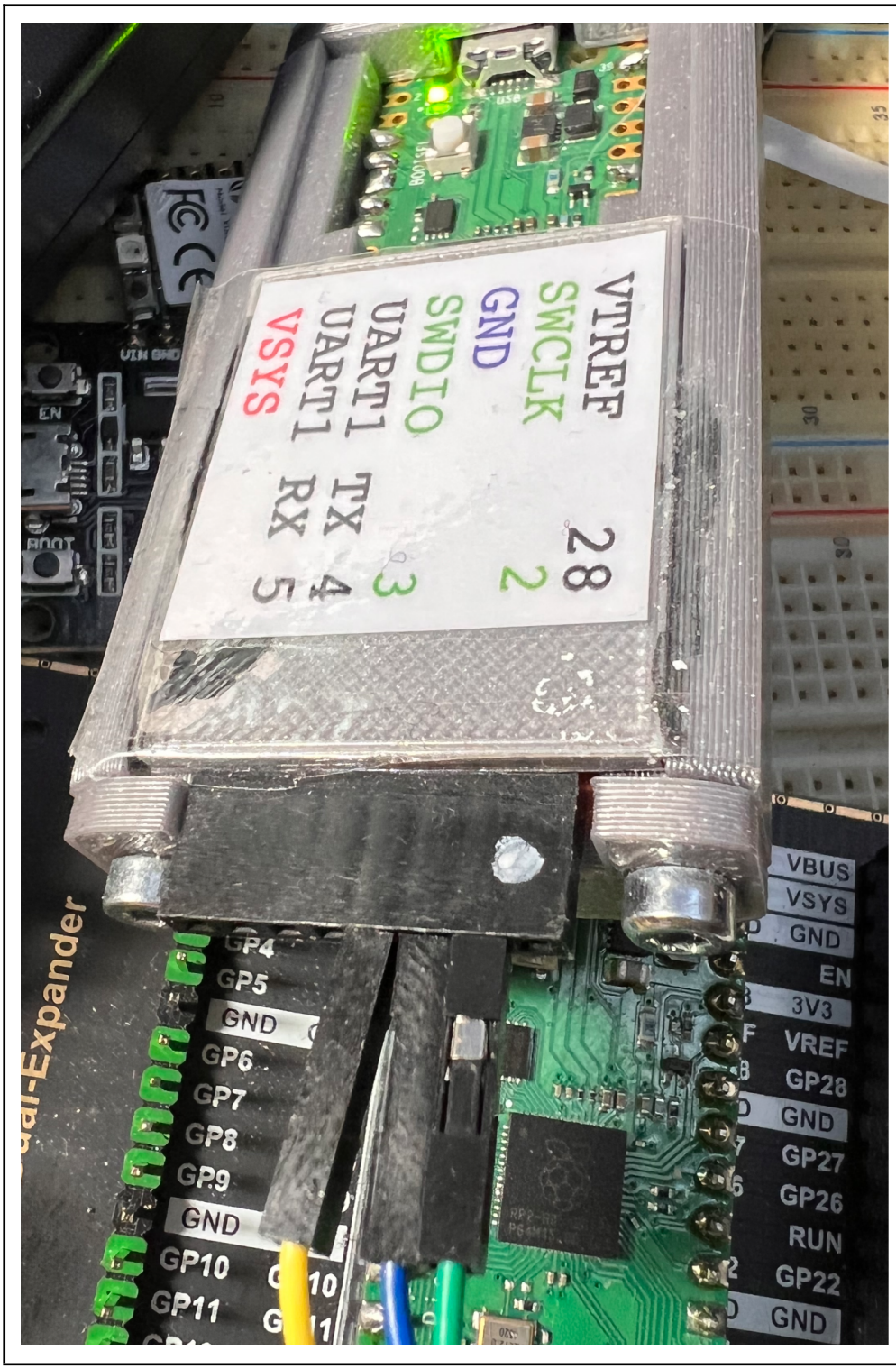
.vscode/launch.json

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Pico Debug",
            "cwd": "${workspaceRoot}",
            "executable": "${command:cmake.launchTargetPath}",
            "request": "launch",
            "type": "cortex-debug",
            "servertype": "openocd",
            "gdbPath" : "arm-none-eabi-gdb",
            "device": "RP2040",
            "configFiles": [
                "interface/picoprobe.cfg",
                "target/rp2040.cfg"
            ],
            "svdFile": "${env:PICO_SDK_PATH}/src/rp2040/hardware_regs/rp2040.svd",
            "runToEntryPoint": "main",
            "postRestartCommands": [
                "break main",
                "continue"
            ]
        }
    ]
}
```

Switch to debug mode as follows.







take a look at the project git: **pico-probe-example**

VCode RP2040 CMSIS-DAP

Debugging based on the CMSIS-DAP standard using one RP2040 core as a debugger (uses the GDB protocol.). The advantage is that no additional dahdware is needed. The disadvantage is that you cannot use second core and own USB. First you need to load the pico-debug .uf2 into

the RP2040. There are 2 versions of MAXRAM and GIMMECACHE. GIMMECACHE is preferred, which has 248kBytes (94% of the total) of SRAM. CMSIS-DAP is held in memory only until power is lost. After reset/power loss, it needs to be reloaded.

https://github.com/majbthrd/pico-debug/releases

- Standard insertion via BOOTSEL
- Make the project the same as for the RP2040 SDK
- Replace the configFiles section for cmsis-dap in **launch.json**

```
"configFiles": [
                "interface/cmsis-dap.cfg",
                "target/rp2040-core0.cfg"
            ],
            "openOCDLaunchCommands": [
                "transport select swd",
                "adapter speed 4000"
            ],
```

take a look at the project git: **pico-debug**

RP2040 Visual Code with Arduino framework & without debug

Create a new project in Platformio for Rasberry platform and Arduino framework. The compilation and first upload to RP2040 is necessary via BOOTSEL

picotool - communication with the RP2040 can only be done using a special tool, which is provided by Rasberry and is part of the SDK.

Note: For MAC OS : brew install libusb

```
[env:pico]
platform = raspberrypi
board = pico
framework = arduino
```

without picoprobe with this setting you have to **press BOOTSEL for the first time** and further uploads work without pressing the button.

Note: upload_protocol = picotool

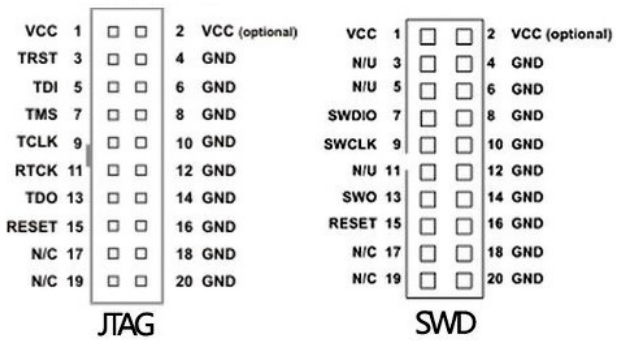
```
tools for BOOTSEL mode operation (more
getting-started-with-pico.pdf - official doc)


"/Users/HOME/.platformio/packages/tool-rp2040tools/picotool"
info -a
```

take a look at the project git: **pico-arduino**

RP2040 Visual Code with Arduino framework & debug - JLINK

Currently supported debugging interfaces under Arduino framework are: **cmsis-dap, jlink, raspberrypi-swd**.

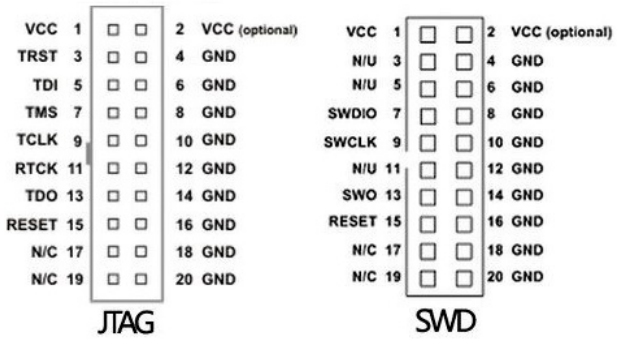
Example with j-link - segger

used SWD connection
SWDIO, SWCLK, GND
 	
jlink requires connection of VREF pin (3V3 on RPPICO) to VCC pin 1 on j-link.
	`[env:pico]`
	`platform = raspberrypi`
	`board = pico`
	`framework = arduino`
	`debug_tool = jlink`
	`upload_protocol = jlink`

take a look at the project git: **pico-arduino-debug**

RP2040 Visual Code with RP2040 SDK & debug - JLINK

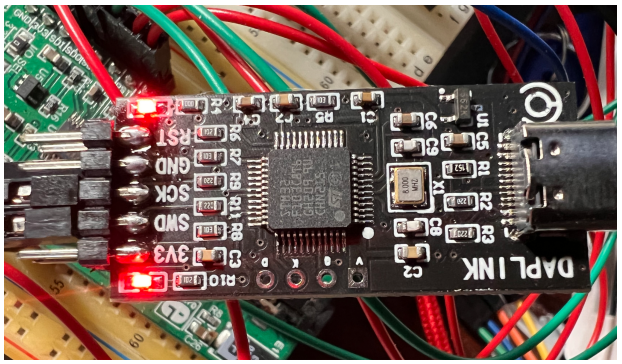
<table>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td>used SWD connection
SWDIO, SWCLK, GND

jlink requires connection of VREF pin (3V3 on RPPICO) to VCC pin 1 on j-link.</td>
<td><code>launch.json</code>

<pre>
{
   "version": "0.2.0",
   "configurations": [
       {
           "name": "Cortex Debug jlink",
           "cwd": "${workspaceRoot}",
           "executable":
"${command:cmake.launchTargetPath}",
           "request": "launch",
           "type": "cortex-debug",
           "serverpath":
"/usr/local/bin/JLinkGDBServerCL",
           "servertype": "jlink",
           "gdbPath": "arm-none-eabi-gdb",
           "interface": "swd",
           "device": "RP2040_M0_0",
           "svdFile":
"${env:PICO_SDK_PATH}/src/rp2040/hardware_regs/rp2040.svd",
           "showDevDebugOutput": "parsed",
           "runToMain": true,
           "postRestartCommands": [
           "break main",
           "continue"
           ],
           "postLaunchCommands": [],
       }
   ]
}
</pre></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</table>

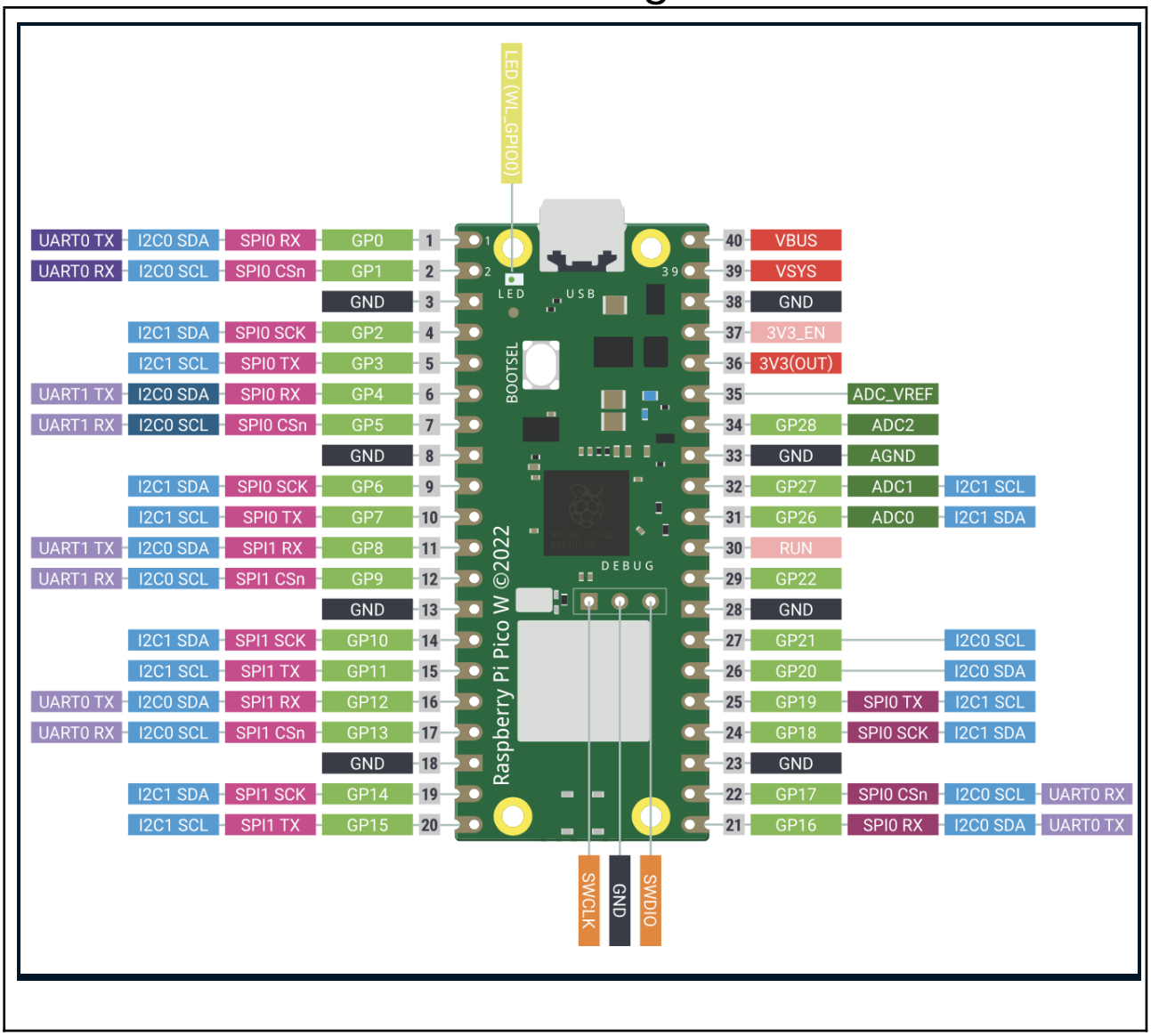
take a look at the project git: **pico-debug-jlink**

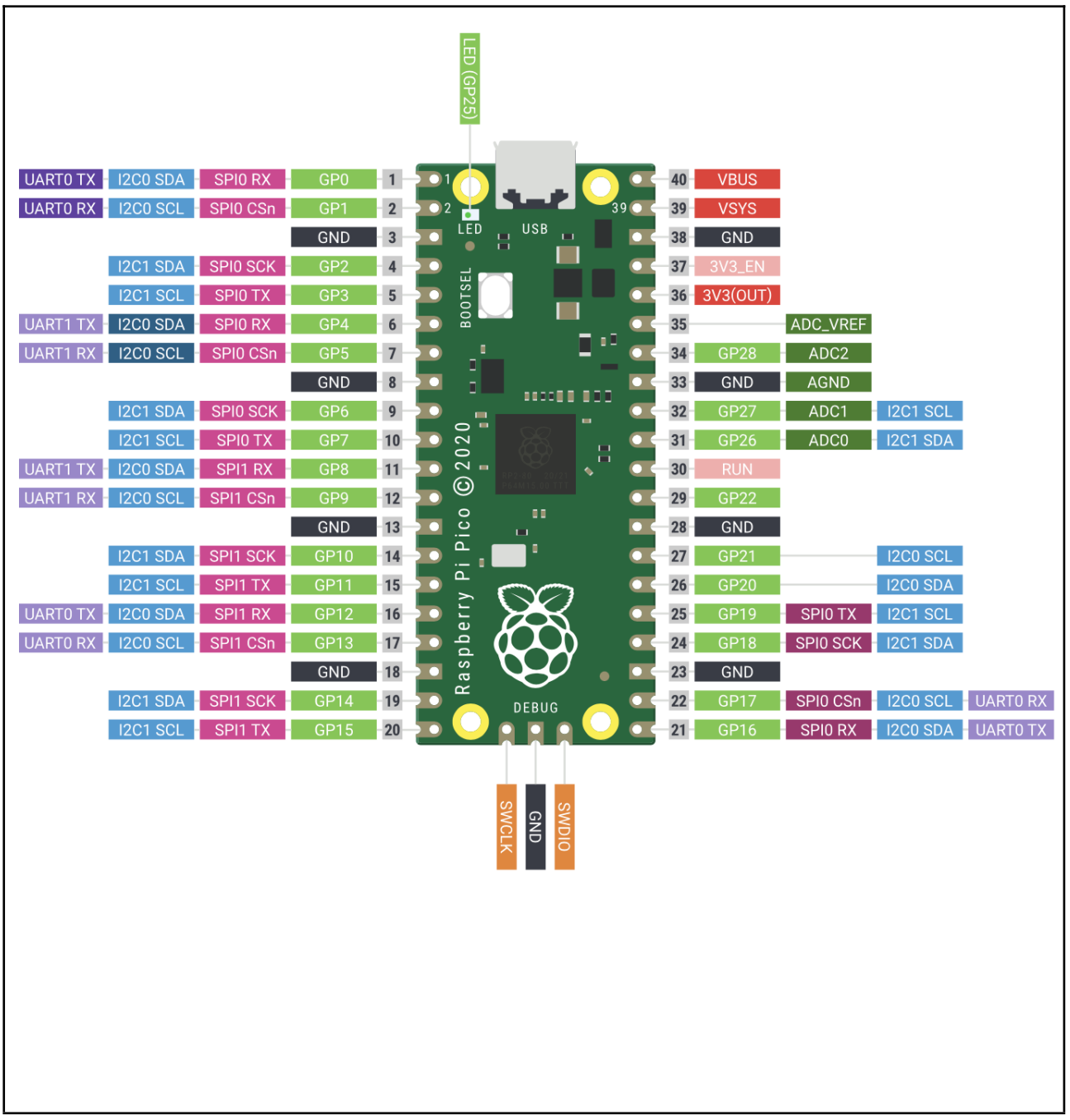
RP2040 Visual Code with DAPLINK Arduino & debug

	platformino.ini [env:pico] platform = raspberrypi board = pico framework = arduino debug_tool = cmsis-dap upload_protocol = cmsis-dap
https://github.com/ARMmbed/DAPLink	**the cheap version of the DAP link is very slow for debugging.**

take a look at the project git: **pico-debug-daplink**

Connected to SWD - hw debug




LED (GP25)
UART0 TX I2C0 SDA SPI0 RX GP0 1
UART0 RX I2C0 SCL SPI0 CSn GP1 2
GND 3
I2C1 SDA SPI0 SCK GP2 4
I2C1 SCL SPI0 TX GP3 5
UART1 TX I2C0 SDA SPI0 RX GP4 6
UART1 RX I2C0 SCL SPI0 CSn GP5 7
GND 8
I2C1 SDA SPI0 SCK GP6 9
I2C1 SCL SPI0 TX GP7 10
UART1 TX I2C0 SDA SPI1 RX GP8 11
UART1 RX I2C0 SCL SPI1 CSn GP9 12
GND 13
I2C1 SDA SPI1 SCK GP10 14
I2C1 SCL SPI1 TX GP11 15
UART0 TX I2C0 SDA SPI1 RX GP12 16
UART0 RX I2C0 SCL SPI1 CSn GP13 17
GND 18
I2C1 SDA SPI1 SCK GP14 19
I2C1 SCL SPI1 TX GP15 20
40 VBUS
39 VSYS
38 GND
37 3V3_EN
36 3V3(OUT)
35 ADC_VREF
34 GP28 ADC2
33 GND AGND
32 GP27 ADC1 I2C1 SCL
31 GP26 ADC0 I2C1 SDA
30 RUN
29 GP22
28 GND
27 GP21 I2C0 SCL
26 GP20 I2C0 SDA
25 GP19 SPI0 TX I2C1 SCL
24 GP18 SPI0 SCK I2C1 SDA
23 GND
22 GP17 SPI0 CSn I2C0 SCL UART0 RX
21 GP16 SPI0 RX I2C0 SDA UART0 TX
1
2
39
LED
USB
BOOTSEL
Raspberry Pi Pico © 2020
DEBUG
SWCLK
GND
SWDIO

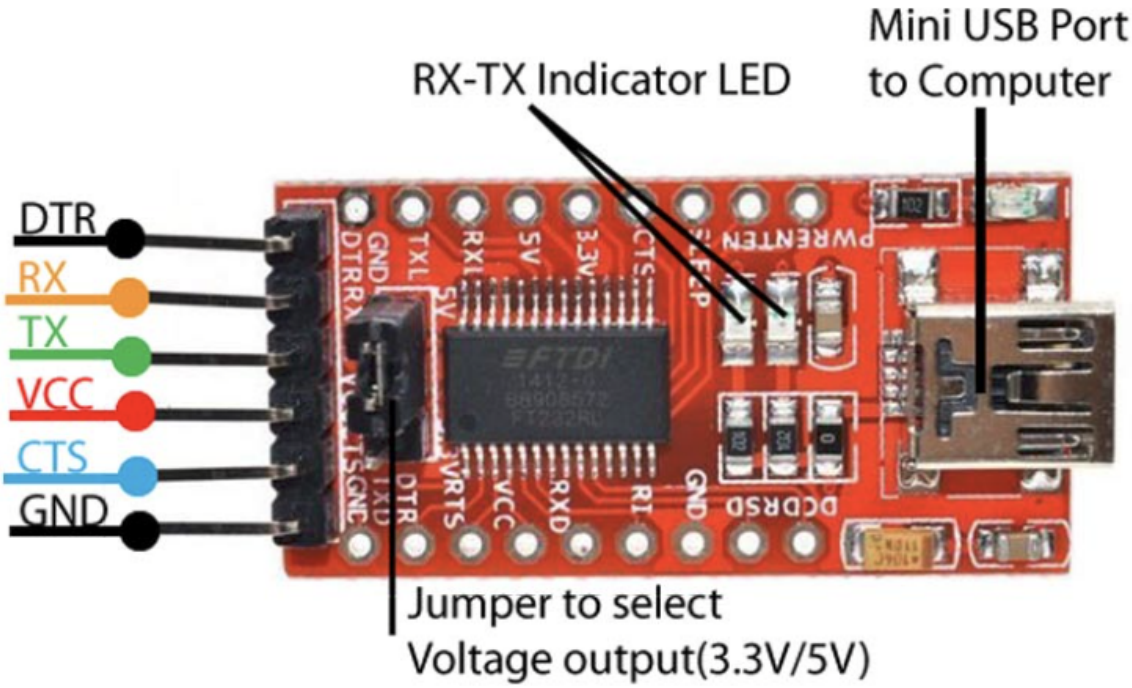
RP2040

Infineon 43439

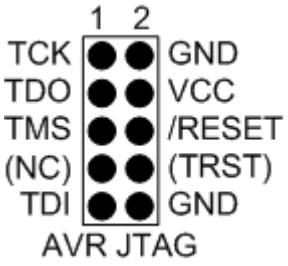
PICO as PICO PROBE

- Pico VSYS -> Target 5V (optional to power the target with 5V)
- **Pico GND -> Target GND**
- **Pico GP2 -> Target SWCLK**
- **Pico GP3 -> Target SWDIO**
- Pico GP4/UART1 Tx -> Target Rx (optional for UART connection)
- Pico GP5/UART1 Rx -> Target Tx (optional for UART connection)

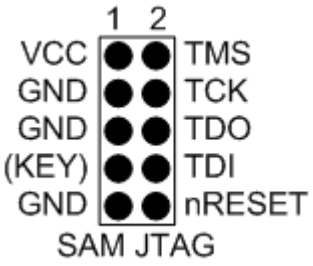
FTDI232



ATMEL-ICE

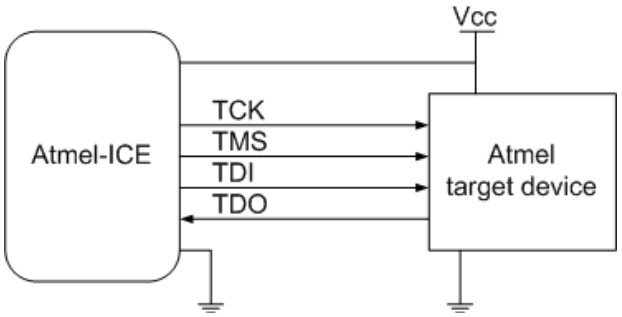
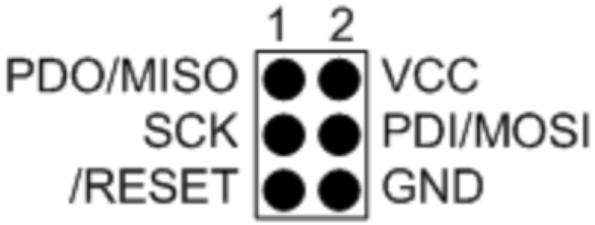
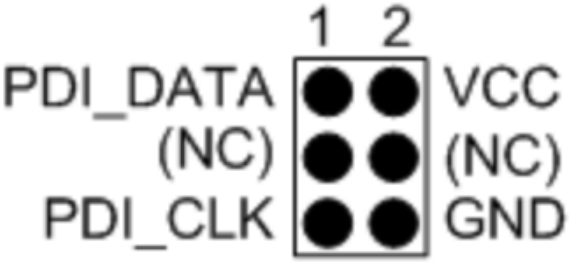
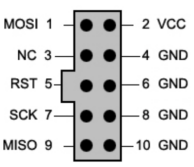


Connector on the programmer AVR
50-mil variants



Connector on the programmer SAM
50-mil variants

TCK 1 TMS 5 TDI 9 TDO 3 nTRST 8	TCK 4 TMS 2 TDI 8 TDO 6 nTRST -

nSRST 6 VTG 4 GND 2,10	nSRST 10 VTG 1 GND 3,5,9
	
SPI/debugWIRE 	PDI 
aWire 	TPI 
	10-pin AVR ISP header is also available through our extra free 6-pin to 10-pin ISP cable 

ATSAMD51N19A

Notes:
SAMSD

jlinkexe -device atsamd51g18a -if SWD -speed 4000 -autoconnect 1