**Modifications to BigTreeTech TFT35 V3.0 Source Code**

*C:\Users\tony\Documents\3D Printers\Marlin Firmware\BIGTREETECH-TouchScreenFirmware-master\Copy to SD Card root directory to update* - Unified Menu Material theme\config.ini

36,2: #//TG 11/1/20 changed from 5 to 6 for 250000 baud

151,2: #//TG 11/1/20 changed this

185,2: #//TG 11/1/20 changed BED was 150

202,2: #//TG 11/1/20 changed these

272,2: #//TG 11/1/20 modified these (6 maximum)

313,3: ##//TG 11/1/20 enabled this

318,2: #//TG 11/1/20 changed this was 0, SWX1 sensor is non-inverting

323,2: #//TG increased due to false triggers (was 100)

439,2: #//TG 11/1/20 enabled this

444,2: #//TG 11/1/20 enabled this

*C:\Users\tony\Documents\3D Printers\Marlin Firmware\BIGTREETECH-TouchScreenFirmware-master\TFT\src\User\API\menu.c*

5,1: //TG this code draws, handles, processes, and responds to menus

*C:\Users\tony\Documents\3D Printers\Marlin Firmware\BIGTREETECH-TouchScreenFirmware-master\TFT\src\User\API\parseACK.c*

550,33: // Parse and store ABL type //TG Is this broken??

*C:\Users\tony\Documents\3D Printers\Marlin Firmware\BIGTREETECH-TouchScreenFirmware-master\TFT\src\User\API\Settings.h*

30,40: #define ICON\_FLASH\_SIGN 20201031 //TG (YYYYMMDD) change if any icon(s) is added or removed

*C:\Users\tony\Documents\3D Printers\Marlin Firmware\BIGTREETECH-TouchScreenFirmware-master\TFT\src\User\Menu\FeatureSettings.c*

4,1: //TG Layout for the page

55,1: //TG here is where you would expand the list if needed, but also need to expand

111,5: //TG this list item has only a LABEL value, no ON/OFF TOGGLE (red dot)

361,29: #ifdef LED\_COLOR\_PIN //TG only on the E3 version

372,35: #ifdef LCD\_LED\_PWM\_CHANNEL //TG LCD brightness control

440,35: void menuFeatureSettings(void) //TG handles the Feature List Menu up,dn, and back buttons

449,51: key\_num = menuKeyGetValue(); //TG get a pressed key?

452,51: case KEY\_ICON\_5: //TG up arrow, load prev list

462,51: case KEY\_ICON\_6: //TG dn arrow, load next list

472,51: case KEY\_ICON\_7: //TG back arrow, load prev infoMenu

C:\Users\tony\Documents\3D Printers\Marlin Firmware\BIGTREETECH-TouchScreenFirmware-master\TFT\src\User\Menu\SettingsMenu.c

120,59: {ICON\_BACKGROUND, LABEL\_BACKGROUND}, //TG available button?

121,59: {ICON\_BACKGROUND, LABEL\_BACKGROUND}, //TG available button?

**Creating/Adding custom icon buttons to menus**

To create a new icon for a button to be used in menus, first choose a name for the icon **CUSTOM\_1**, then create a .bmp file for the button and name it **CUSTOM\_1.bmp**

Now, to be able to use it in menus do the following:

First, we must add a few new keywords into these 4 files so the rest of the code knows about our new icon:

*TFT\src\User\API\icon\_list.inc* add **X\_ICON (CUSTOM\_1) this makes ICON\_CUSTOM\_1**

*TFT\src\User\API\Language\Language.inc* add **X\_WORD (CUSTOM\_1) this makes LABEL\_CUSTOM\_1**

*TFT\src\User\API\Language\language\_en.h* add **#define STRING\_CUSTOM\_1 "TG Menu"**

*TFT\src\User\API\Language\language\_keywords.h* add #**define LANG\_KEY\_CUSTOM\_1 "label\_custom\_1:"**

*Note: language\_keyowrds.h is only needed if config file option is desired!*

Any time a new keyword is added to the 4 files above, we have to change these signs in *TFT\src\User\API\Settings.h*:

#define **LANGUAGE\_FLASH\_SIGN** 20201107 //(YYYYMMDD) change if any keyword(s) in language pack is added or removed

#define **ICON\_FLASH\_SIGN** 20201107 //(YYYYMMDD) change if any icon(s) is added or removed

Now, to implement the new icon as a button in a menu, we find an empty slot in one of the existing menus and

modify the **const MENUITEMS** array for that parent menu. For example, we'll use the Settings Menu as the parent

menu for our new button because it has some empty positions in it (*each menu page can have 7 buttons max*).

So open the code file for the Settings Menu (TFT*\src\User\Menu\SettingsMenu.c* in this case).

We see there is a **const MENUITEMS** array named **settingsItems** and we also see this array has two unused

entries at indexes 5 and 6 (they're set to **ICON\_BACKGROUND,LABEL\_BACKGROUND** which means empty).

So we can add our new button in like so, replacing with our new values as shown below:

const MENUITEMS settingsItems = {

// title

LABEL\_SETTINGS,

// icon label

{{ICON\_SCREEN\_SETTINGS, LABEL\_SCREEN\_SETTINGS},

{ICON\_MACHINE\_SETTINGS, LABEL\_MACHINE\_SETTINGS},

{ICON\_FEATURE\_SETTINGS, LABEL\_FEATURE\_SETTINGS},

{ICON\_SCREEN\_INFO, LABEL\_SCREEN\_INFO},

{ICON\_CONNECTION\_SETTINGS, LABEL\_CONNECTION\_SETTINGS},

{**ICON\_CUSTOM\_1, LABEL\_CUSTOM\_1**}, <-- we overwrite previous ICON\_BACKGROUND, LABEL\_BACKGROUND

{ICON\_BACKGROUND, LABEL\_BACKGROUND}, with the icon name and text label of our custom icon

{ICON\_BACK, LABEL\_BACK},}

};

Next, we need to add some code to handle what happens when our custom button is pressed. There's usually

a code block like the one shown shown below for the parent menu. It’s usually found not too far after the

**const MENUITEMS <parentmenu\_name>** block we modified above. Put your handling code in this code block.

while(infoMenu.menu[infoMenu.cur] == **menuSettings**)

{

key\_num = menuKeyGetValue();

switch(key\_num)

{

case KEY\_ICON\_0:

infoMenu.menu[++infoMenu.cur] = menuScreenSettings;

break;

case KEY\_ICON\_1:

mustStoreCmd("M503 S0\n");

infoMenu.menu[++infoMenu.cur] = menuMachineSettings;

break;

case KEY\_ICON\_2:

infoMenu.menu[++infoMenu.cur] = menuFeatureSettings;

break;

case KEY\_ICON\_3:

infoMenu.menu[++infoMenu.cur] = menuInfo;

break;

case KEY\_ICON\_4:

infoMenu.menu[++infoMenu.cur] = menuConnectionSettings;

break;

case KEY\_ICON\_5: <-- here we add the case for custom\_1 button

**infoMenu.menu[++infoMenu.cur] = menuTGmenu; at position 5 in the menu. You can call**

break; another menu or handle some task here.

Also notice there is no KEY\_ICON\_6,

case KEY\_ICON\_7: since that menu position is blank.

infoMenu.cur--;

break;

default:

break;

}

loopProcess();

}

Finally, we need to create a .c and .h module to perform some action (i.e. handle the **menuTGmenu** call above). Here is the C code file, **TGmenu.c**

#include "TGmenu.h"

#include "SendGcode.h"

#include "includes.h"

//extern char gcodeBuf[CMD\_MAX\_CHAR];

//char\* gcodeBufPtr = &gcodeBuf[0];

void menuTGmenu(void) <-- here is the new function to draw new menu

{

//TG examples of variable usage

/\*

static uint8\_t ublSlot;

static bool ublIsSaving = true;

static bool ublSlotSaved = false;

\*/

MENUITEMS TGmenuItems = { <-- this is the new MENUITEMS structure

// title

LABEL\_TGMENU,

// icon label

{{ICON\_Z\_0, LABEL\_Z\_0},

{ICON\_Z\_300, LABEL\_Z\_300},

{ICON\_M503, LABEL\_M503},

{ICON\_BACKGROUND, LABEL\_BACKGROUND},

{ICON\_BACKGROUND, LABEL\_BACKGROUND},

{ICON\_BACKGROUND, LABEL\_BACKGROUND},

{ICON\_BACKGROUND, LABEL\_BACKGROUND},

{ICON\_BACK, LABEL\_BACK}}

};

KEY\_VALUES key\_num = KEY\_IDLE;

The switch below is not used, just included to show how to modify an existing menu dynamically

//TG this looks like an example of how to modify an existing menu on the fly based on

// information or other variables elsewhere in code

/\* switch (infoMachineSettings.leveling)

{

case BL\_BBL:

autoLevelingItems.title.index = LABEL\_ABL\_SETTINGS\_BBL;

break;

case BL\_UBL:

autoLevelingItems.title.index = LABEL\_ABL\_SETTINGS\_UBL;

autoLevelingItems.items[1].icon = ICON\_EEPROM\_SAVE;

autoLevelingItems.items[1].label.index = LABEL\_SAVE;

autoLevelingItems.items[2].icon = ICON\_EEPROM\_RESTORE;

autoLevelingItems.items[2].label.index = LABEL\_LOAD;

break;

default:

break;

}

\*/

menuDrawPage(&TGmenuItems); <-- call to draw the menu

while (infoMenu.menu[infoMenu.cur] == menuTGmenu) <-- now wait in loop for an item to be pressed

{

key\_num = menuKeyGetValue(); <-- find out what menu item was touched

switch (key\_num) <-- do something for current touched menu item

{

case KEY\_ICON\_0:

storeCmd("G1 Z0\n");

break;

case KEY\_ICON\_1:

storeCmd("G1 Z300\n");

break;

case KEY\_ICON\_2:

/\*

gcodeBuf[0]=(char)'M';

gcodeBuf[1]=(char)'5';

gcodeBuf[2]=(char)'0';

gcodeBuf[3]=(char)'3';

gcodeBuf[4]=(char)'/n';

gcodeBuf[5]=0;

storeCmd(gcodeBuf);

\*/

storeCmd("M503\n");

//gcodeBuf[0] = 0;

infoMenu.menu[++infoMenu.cur] = menuTerminal;

break;

case KEY\_ICON\_7: <-- the “back” item sets current menu to previous

infoMenu.cur--;

break;

default:

break;

}

loopProcess();

}

}

And here is the .h file, **TGmenu.h**

#ifndef \_TGMENU\_H\_

#define \_TGMENU\_H\_

#ifdef \_\_cplusplus

extern "C" {

#endif

#include <stdbool.h>

void menuTGmenu(void);

#ifdef \_\_cplusplus

}

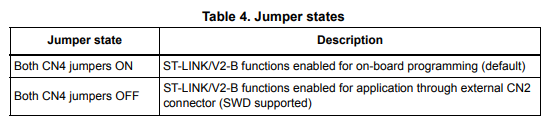
#endif

#endif

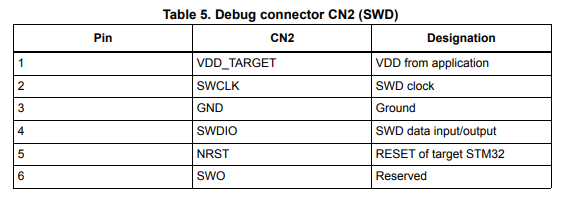
We are done!

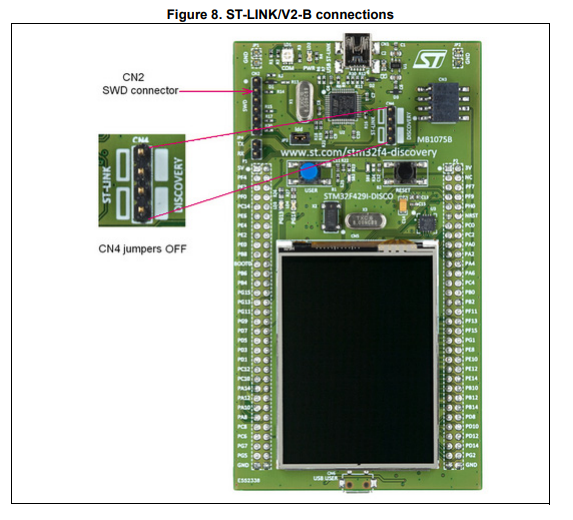
**Notes on Debugging the BTT TFT35 using PlatformIO and STLink probe**

If you don’t have a STLink probe, the STM32F4 discovery kit has a STLink V2 on-board and it can be configured to talk to the on-board STM32 chip or an External application board. To use the STM32F4 discovery kit as a pure STLink probe, remove the two jumpers which changes the board from Discovery to STLink mode.



Next, connect the external target board to connector CN2 (pinout below).





**Prepare the PlatformIO project for debugging**

The source code project from BigTreeTech on GitHub is downloaded as a zip file. Once unzipped, opening the folder in PlatformIO should allow you to compile without errors. However, to use a debugger, several items need to be modified in the *platformio.ini* file (or through the platformio project GUI):

* Add a custom board definition describing the board, mcu, flash, and RAM, etc.
* Copy the custom board.json file to the buildroot\boards folder (and to whichever platform folder you want in ***%USER%\.platformio\platforms\CHOSEN PLATFORM\boards***, so it will be available to any project)
* Change ***platformio.ini*** settings so that a ***firmware.elf*** and ***firmware.bin*** are created
* Setup PlatformIO debug options

**Custom board creation**

You can skip this step if the board you’re debugging is already available in PlatformIO. This wasn’t the case for the BTT35 and it’s STM32F207VC chip, so one must be custom made.

You can start by finding an existing board that resembles your board and/or chip and use it as a base to modify.

There was an existing json file for the STM32F207VC in the ***buildroot\boards*** folder of the downloaded GitHub project, and an existing ST-Nucleo-F207ZG board with our chip in the PlatformIO boards database, so we’ll use those as a base and make a few small changes.

***One important note was to add the “product\_line” token because a recent update to PlatformIO caused compile failures without it!***

Here is the STM32F207VC.json board file and the changes (*some from ST-Nucleo-F207ZG*) needed in yellow:

{

"build": {

"core": "stm32",

"cpu": "cortex-m3",

"extra\_flags": "-DSTM32F207xx",

"f\_cpu": "120000000L",

"mcu": "stm32f207vct6",

"product\_line": "STM32F207xx", <- needed to compile with latest PIO updates

"variant": "stm32f2"

"hwids": [ <- maybe not necessary but added to identify chip

[

"0x0411",

"0x0002"

]

]

},

"connectivity": [

"can",

"ethernet"

],

"debug": {

"default\_tools": [

"stlink" <- was cmsis-dap, needs to be stlink

],

"jlink\_device": "STM32F207VC", <- make sure this is the right chip

"onboard\_tools": [

"stlink" <- was cmsis-dap, needs to be stlink

],

"openocd\_target": "stm32f2x",

"svd\_path": "STM32F20x.svd"

},

"frameworks": [

"arduino",

“cmsis”, <- extra supported framework choice added

"mbed",

"stm32cube", <- tried stm32cube, but had HAL errors on compile

"libopencm3", <- extra supported framework choice added

"zephyr" <- extra supported framework choice added

],

"name": "BTT TFT35 V3", <- give it a new appropriate name

"upload": {

"maximum\_ram\_size": 49152, <- make sure flash and ram sizes are correct!

"maximum\_size": 262144,

"protocol": "stlink", <- was cmsis-dap, needs to be stlink

"protocols": [

"jlink",

"stlink",

"blackmagic",

"mbed",

"cmsis-dap", <- extra supported protocol added

"dfu" <- extra supported protocol added

]

},

"url": "http://www.st.com/content/st\_com/en/products/microcontrollers/stm32-32-bit-arm-cortex-mcus/stm32f2-series/stm32f207/stm32f207vc.html", <- updated to point to correct link for this chip

"vendor": "ST"

}

The above file was renamed as ***btt\_tft35\_v3.json*** and saved to the project ***buildroot\boards*** folder and to ***%USER%\.platformio\platforms\ststm32\boards*** and ***ststm32@8.0.0\boards*** folders.

**Making required changes to platform.ini for debugging**

There are a lot of sections in this file, mostly concerned with all the optional environments for each of the BTT TFT variants. We can really simply this by removing all the un-needed targets, after all we only have the TFT35 V3.0 to work on so why not?

Hence we’ll only look at the sections that need to be changed for **stlink** debugging to work.

The ini file begins with section **[platformio]**, here is defined the source folder to be compiled, the path to the boards\_dir, and the default\_envs to be built. In our case, the only environment we want built is **BIGTREE\_TFT35\_V3\_0**, although there can be multiples.

Next section is **[common]**, use this to define keys and values that can be included into other sections by using the ${common.keyname} syntax to include them.

Next is a custom section named **[stm32f2xx]**, it defines a **default\_src\_filter** key that includes the default\_src\_filter from the **[common]** section, and adds a few more filters particular to the STM32F2xx series chips. A **build\_flags** key is defined that includes the build\_flags from the **[common]** section and additionally defines *“\_\_STATIC\_INLINE”* for the compiler as well as including some source/library files unique to the STM32F2xx series chips.

Finally we come to our defined environment section **[env:BIGTREE\_TFT35\_V3\_0]**. The env: prefix denotes that this is an environment which can be chosen for the project. Here we define the platform, framework, and board that is pertinent to our system we want to debug. We also define the **upload\_protocol** and **debug\_tool** keys for our system. We define the **src\_filter** and **build\_flag** keys by expanding upon those in the **[stm32f2xx] section.**

An ***extra\_scripts*** key can also be put here or in the **[common]** section if needed.

The **debug\_tool** key is set here as **stlink**.

*Note: The “as received from GitHub” platformio.ini file has an* ***extra\_scripts*** *key in the* ***[common]*** *section or the* ***[env:BIGTREE\_TFT35\_V3\_0]*** *section. One entry is* ***pre:buildroot/scripts/custom\_filename.py*** *which renames the normal compiler output of firmware.bin to the form “HARDWARE”.”SOFTWARE\_VERSION”, resulting in the file* ***BIGTREE\_TFT35\_V3.0.26.x.bin****. Optionally another* ***entry***  *key* ***post:buildroot/scripts/short\_out\_filename.py*** *is used to shorten the filename and get rid of the “.x”.*

*Both of these entries need to be commented out so the standard compiler output will produce the files* ***firmware.bin*** *and* ***firmware.elf*** *(needed for the debugger) !*

Now let’s take a closer look at the platform.ini with all the changes for debugging:

[platformio]

src\_dir = TFT

boards\_dir=buildroot\boards ; use boards dir in project, comment out to use boards in core\_dir

default\_envs = BIGTREE\_TFT35\_V3\_0 ; our default build environment to use

[common]

; src\_filter +/- determines what files to include when building

; use Hal/stm32f2\_f4xx>, not Hal/stm32f10x, maybe better to move these to [stm32f10x], [stm32f2xx] sections

default\_src\_filter = +<src/\*> -<src/Libraries> -<src/User/Hal/stm32f10x> +<src/User/Hal/stm32f2\_f4xx>

build\_flags =

-fmax-errors=5

-g3 <- g3 for full debug symbols, g is normal release build

-ggdb

-Wno-missing-braces

-DUSE\_STDPERIPH\_DRIVER=

-ITFT/src/User/Fatfs <-specifically tell compiler/linker to use these sources

-ITFT/src/User/Hal

-ITFT/src/User/Menu

-ITFT/src/User/Variants

-ITFT/src/User

-ITFT/src/User/API

-ITFT/src/User/API/UI

-ITFT/src/User/API/Gcode

-ITFT/src/User/API/Language

-ITFT/src/User/API/Vfs

-ITFT/src/User/Hal

-ITFT/src/User/Hal/STM32\_USB\_HOST\_Library/Core/inc

-ITFT/src/User/Hal/STM32\_USB\_HOST\_Library/Class/MSC/inc

-ITFT/src/User/Hal/STM32\_USB\_HOST\_Library/Usr/inc

-ITFT/src/User/Hal/STM32\_USB\_OTG\_Driver/inc

-DSOFTWARE\_VERSION=26.x <- used by custom\_filename.py

-DSOFTWARE\_VERSION\_SHORT=26 <- used by short\_out\_filename.py

extra\_scripts =

pre:buildroot/scripts/custom\_filename.py <- these can stay here, but we will not use them in

post:buildroot/scripts/short\_out\_filename.py <- our BIGTREE\_TFT35\_V3\_0 environment, instead we

post:buildroot/scripts/auto\_gen\_language\_pack.py <- will define our own

[stm32f10x] <- this is not used, maybe it can be deleted?

default\_src\_filter = ${common.default\_src\_filter} +<src/Libraries/cmsis/stm32f10x> +<src/Libraries/fwlib/stm32f10x> +<src/User/Hal/stm32f10x>

build\_flags =

${common.build\_flags}

-D\_\_STATIC\_INLINE=

-ITFT/src/Libraries/cmsis/Core-CM3

-ITFT/src/Libraries/cmsis/stm32f10x

-ITFT/src/Libraries/fwlib/stm32f10x

-ITFT/src/Libraries/fwlib/stm32f10x/inc

-ITFT/src/User/Hal/stm32f10x

[stm32f2xx] <- we want this one

; don't use cmsis/stm32f2xx, but use fwlib/stm32fxx and use HAL from project

default\_src\_filter = ${common.default\_src\_filter} -<src/Libraries/cmsis/stm32f2xx> +<src/Libraries/fwlib/stm32f2xx> +<src/User/Hal/stm32f2\_f4xx>

build\_flags =

${common.build\_flags}

-D\_\_STATIC\_INLINE=

-ITFT/src/Libraries/cmsis/Core-CM3 <-specifically tell compiler/linker to use these sources

-ITFT/src/Libraries/cmsis/stm32f2xx

-ITFT/src/Libraries/fwlib/stm32f2xx

-ITFT/src/Libraries/fwlib/stm32f2xx/inc

-ITFT/src/User/Hal/stm32f2\_f4xx

[env:BIGTREE\_TFT35\_V3\_0] <- our desired environment section

build\_type = debug

debug\_build\_flags = -O1 –ggdb3 –g3

debug\_tool = stlink ; use stlink for debugging

debug\_load\_mode = modified ; upload to target only when firmware is modified

debug\_load\_cmds =

set remotetimeout 120 ; set target response timeout to 2 minutes

show remotetimeout ; display value to make sure it set

platform = ststm32@11.0.0 ; specifically use version 11.0.0

framework = cmsis

board = btt\_tft35\_v3 ; custom board in buildroot\boards

;upload\_protocol = stlink ; stlink not supported by cmsis-dap, so we have to use stlink here

upload\_protocol = custom <- define a custom protocol cause the built-in upload writes flash at 0x08000000

upload\_command = $PROJECT\_PACKAGES\_DIR\tool-stm32duino\texane-stlink\st-flash write $SOURCE 0x8008000

; don't include Startup folder from project, it conflicts with same files in core\_dir

src\_filter = ${stm32f2xx.default\_src\_filter} -<src/Libraries/Startup/stm32f2xx>

extra\_scripts =

;pre:buildroot/scripts/custom\_filename.py ;comment out to keep output as firmware.bin and firmware.elf

buildroot/scripts/stm32f2xx\_0x8000\_iap.py

;post:buildroot/scripts/bin2elf.py <- I’m testing this to copy named.bin to named.elf instead

build\_flags =

${stm32f2xx.build\_flags}

-DSTM32F2XX=

-DHSE\_VALUE=8000000ul

-DVECT\_TAB\_FLASH=0x08008000 <- for some reason this is ignored so we have a custom upload protocol texane-stlink with base set to 0x08008000

-DRAM\_SIZE=48 ; Available RAM size in kbytes

-DHARDWARE="BIGTREE\_TFT35\_V3.0" <- used by custom\_filename.py

-DHARDWARE\_SHORT="B35V30" <- used by custom\_filename.py

-DTFT35\_V3\_0=

extra\_scripts =

pre:buildroot/scripts/custom\_filename.py ; changes filenames to BIGTREE\_TFT35\_V3.0.26.x

buildroot/scripts/stm32f2xx\_0x8000\_iap.py ; linker script

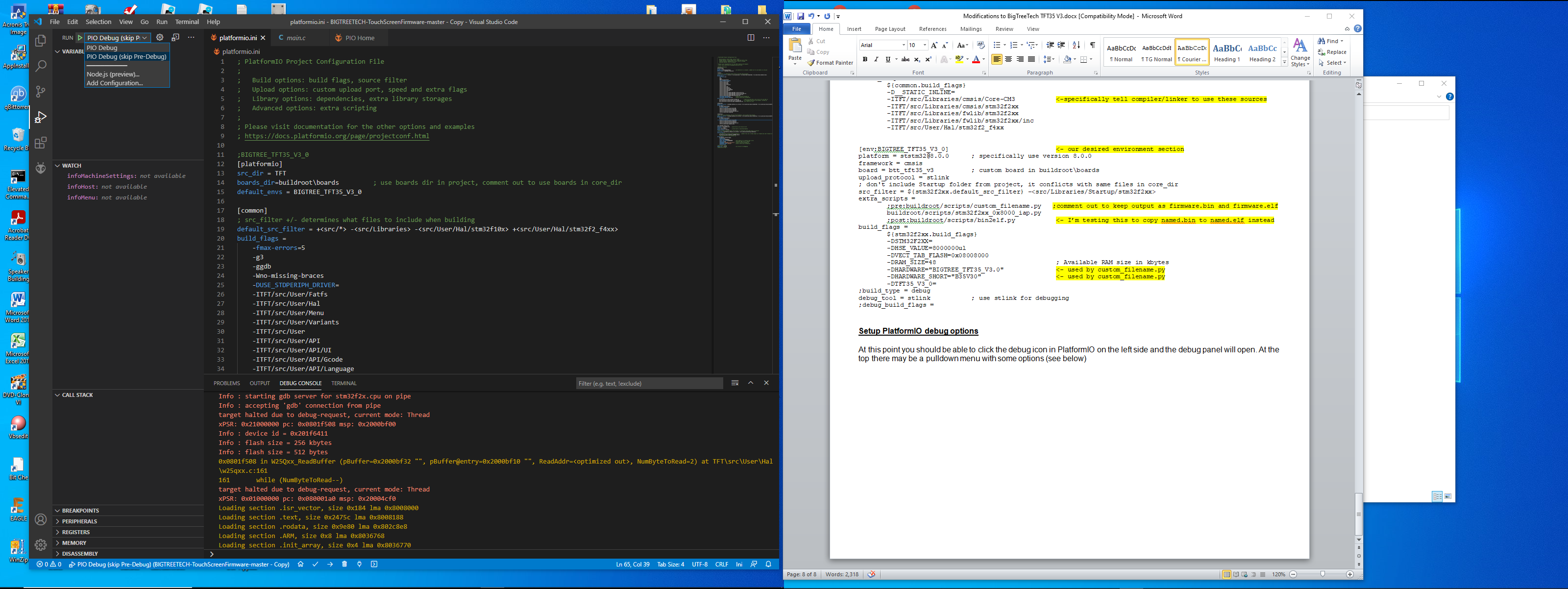
post:buildroot/scripts/bin2elf.py ; names the elf file for debugger

;post:buildroot/scripts/auto\_gen\_language\_pack.py ;optional – rebuilds language.ini

We also need to make sure we have all the ***extra\_scripts*** present in buildroot\scripts folder, so put a copy of our custom made script file ***bin2elf.py*** there. This script copies an ELF file to the output directory for debugging build with a unique filename other than ***firmware.bin***.

**Setup PlatformIO debug options**

At this point you should be able to click the debug icon in PlatformIO on the left side and the debug panel will open. At the top there may be a pulldown menu with some options (see below)



We are almost ready to Click a debug option and Run!

You must click the one that ***does not skip*** the Pre Debug step if you haven’t built the project, or you may get a task error.

After doing this once you should be able to run either option.

You can also run by clicking the PIO bug icon to bring up project tasks. Then select a task and under the advanced tab you can do a number of things like test, check, Pre-Debug, Build, Upload, etc.

If you get errors, you may have to check ***launch.json*** to get things fixed.

Appendix A: Here is the ***bin2elf.py*** script

Import("env") <- gives access to the PIO environment variables

import shutil <- shell utilities for the copyfile command

def bin2elf(source, target, env): <- defines a function call for the code below

print("copying firmware to ELF...")

firmware\_name = env['PROGNAME'] <- gets the HARDWARE.VER name i.e. BIGTREE\_TFT35\_V3.0.26.x

firmware\_path = env.subst('$BUILD\_DIR') + "/" <- gets the path to the build output i.e. pio\build\env\_name #print(firmware\_name) #print(firmware\_path)

shutil.copyfile(firmware\_path + firmware\_name, firmware\_path + firmware\_name + ".elf") <- copy to an .elf extension

print("elf file created.")

#env.Replace($PROGNAME="firmware\_%s" % "elf" )

env.AddPostAction("buildprog", bin2elf) <- adds this function to the environment’s PostAction list

to execute a script AFTER the compile/link completes.

It has to be done this way. Simply adding the ***script.py***

to the ***extra\_scripts***= keyword post: prefix (***post:script.py***)

does run it at the end of compiler/linker, it will run before!