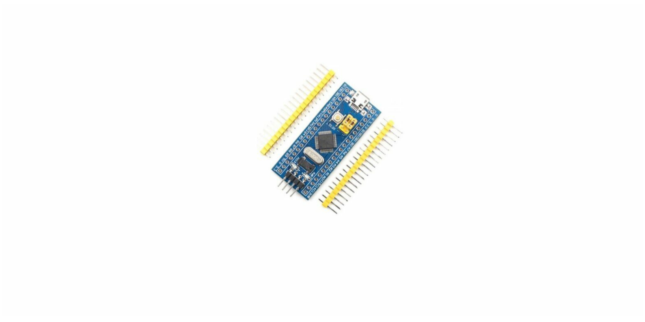
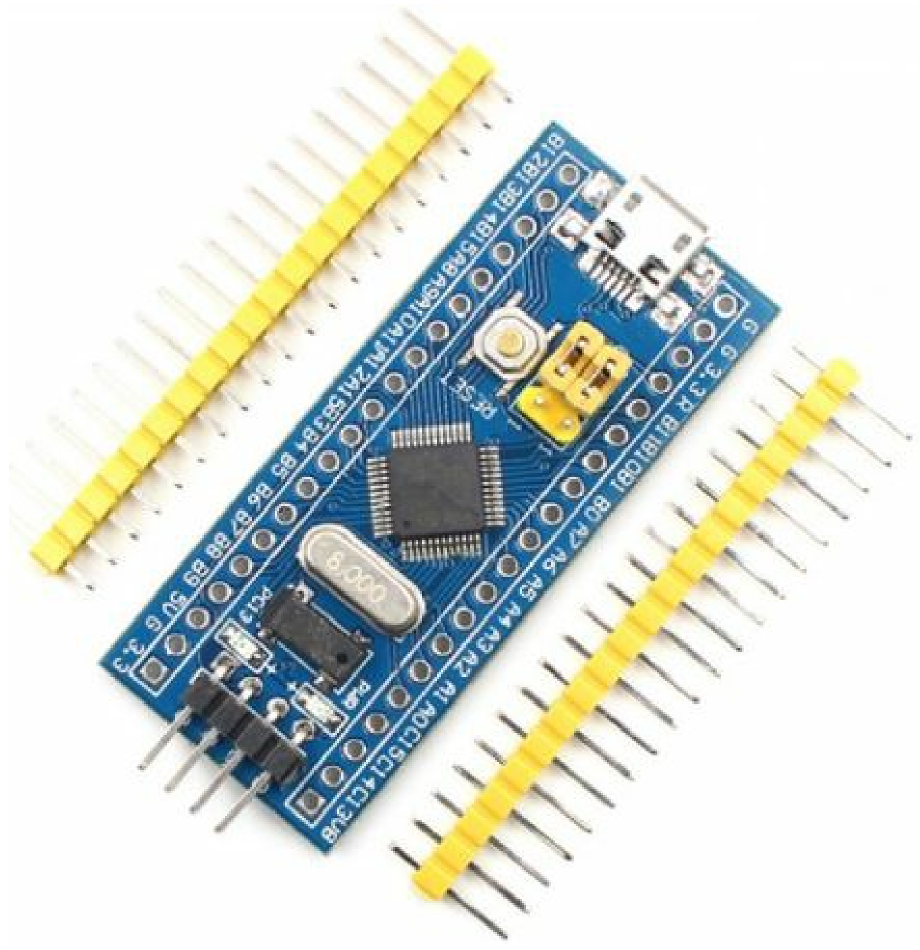


STM32F103C8T6 Minimum System Development Board User Manual

Contents

STM32

STM32F103C8T6 Minimum System Development Board



Product Information

The STM32F103C8T6 ARM STM32 Minimum System Development Board Module is a development board that is based on the STM32F103C8T6 microcontroller. It is designed to be programmed using the Arduino IDE and is compatible with various Arduino clones, variations, and third-party boards like the ESP32 and ESP8266.

The board, also known as the Blue Pill Board, operates at a frequency approximately 4.5 times higher than an Arduino UNO. It can be used for various projects and can be connected to peripherals such as TFT displays.

The required components to build projects with this board include the STM32 Board, FTDI Programmer, Color TFT display, Push Button, Small Breadboard, Wires, Power Bank (optional for stand-alone mode), and USB to Serial Converter.

Schematic

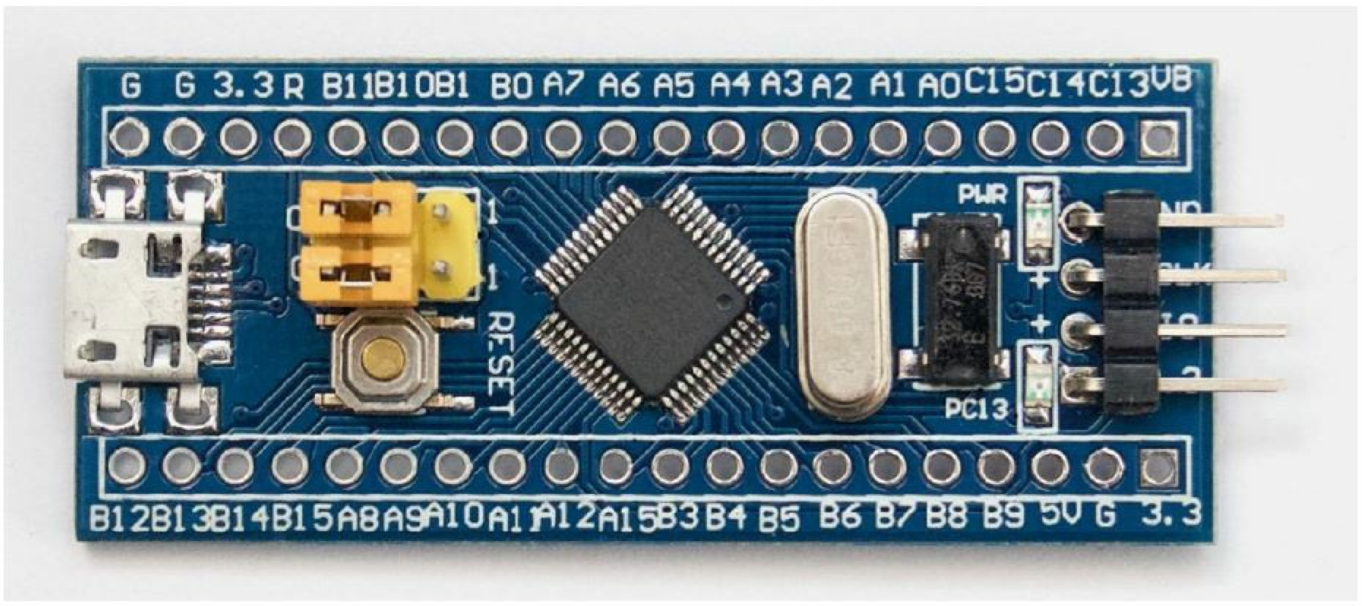
To connect the STM32F1 board to the 1.8 ST7735-based coloured TFT Display and a push button, follow the pin-to-pin connections described in the provided schematics.

Setting up the Arduino IDE for STM32

1. Open the Arduino IDE.
2. Go to Tools -> Board -> Board Manager.
3. In the dialogue box with a search bar, search for “STM32F1” and install the corresponding package.
4. Wait for the installation procedure to complete.
5. After installation, the STM32 board should now be available for selection under the Arduino IDE board list.

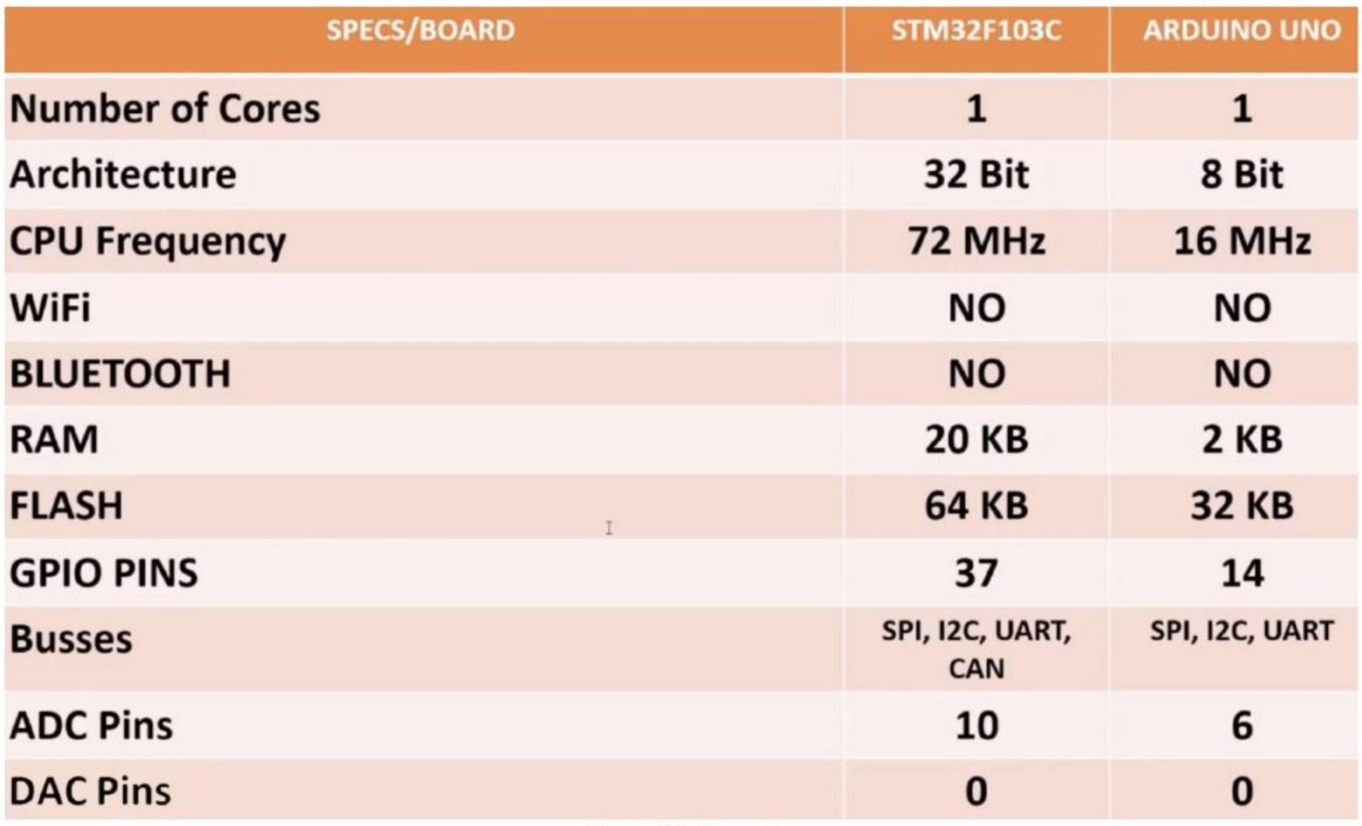
Programming STM32 boards with the Arduino IDE

Since its inception, the Arduino IDE has demonstrated the desire to support all kinds of platforms, from Arduino clones and variations of different manufacturers to third-party boards like the ESP32 and ESp8266. As more people get familiar with the IDE, they are beginning to support more boards that are not based on ATMEL chips and for today’s tutorial we will look at one of such boards. We will examine how to program the STM32-based, STM32F103C8T6 development board with the Arduino IDE.


G G 3.3 R B11 B10 B1 B0 A7 A6 A5 A4 A3 A2 A1 A0 C15 C14 C13 VB
PWR
RESET
PC13
B12 B13 B14 B15 A8 A9 A10 A11 A12 A15 B3 B4 B5 B6 B7 B8 B9 5V G 3.3

Blue Pill Board

The STM32 board to be used for this tutorial is none other than the STM32F103C8T6 chip-based STM32F1 development board commonly referred to as “Blue Pill” in line with the blue colour of its PCB. Blue Pill is powered by the powerful 32-bit STM32F103C8T6 ARM processor, clocked at 72MHz. The board operates on 3.3v logic levels but its GPIO pins have been tested to be 5v tolerant. While it does not come with WiFi or Bluetooth like the ESP32 and Arduino variants, it offers 20KB of RAM and 64KB of flash memory which makes it adequate for large projects. It also possesses 37 GPIO pins, 10 of which can be used for Analog sensors since they have ADC enabled, along with others that are enabled for SPI, I2C, CAN, UART, and DMA. For a board that costs around $3, you will agree with me that these are impressive specs. A summarized version of these specifications compared with that of an Arduino Uno is shown in the image below.



SPECS/BOARD	STM32F103C	ARDUINO UNO
Number of Cores	1	1
Architecture	32 Bit	8 Bit
CPU Frequency	72 MHz	16 MHz
WiFi	NO	NO
BLUETOOTH	NO	NO
RAM	20 KB	2 KB
FLASH	64 KB	32 KB
GPIO PINS	37	14
Busses	SPI, I2C, UART, CAN	SPI, I2C, UART
ADC Pins	10	6
DAC Pins	0	0

Blue Pill Board

Based on the specs above, the frequency at which Blue Pill operates is about 4.5 times higher than an Arduino UNO, for today's tutorial, as an example on how to use the STM32F1 board, we will connect it to a 1.44″ TFT display and program it to calculate the "Pi" constant. We will note how long it took the board to obtain the value an compare it with the time it takes an Arduino Uno to perform the same task.

Required Components

The following components are required to build this project;

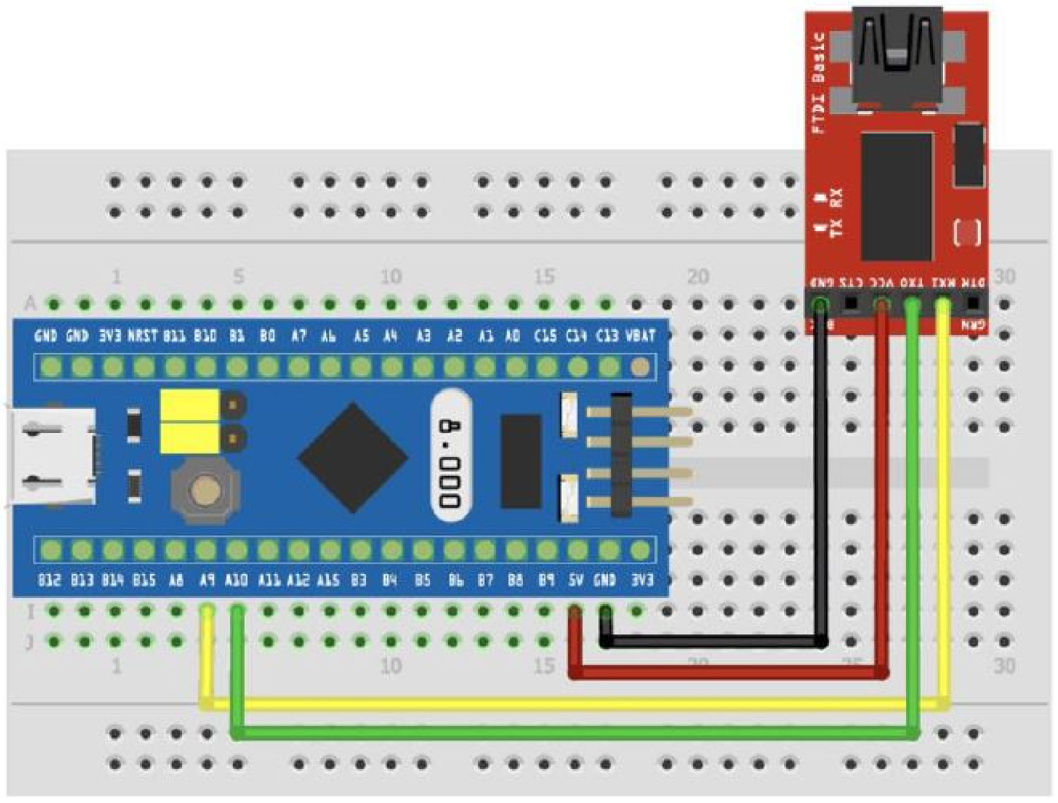
- STM32 Board
- FTDI Programmer
- Colour TFT
- Push Button
- Small Breadboard
- Wires
- Power Bank
- USB to Serial Converter

As usual, all the components used for this tutorial can be bought from the attached links. The power bank is however only needed if you want to deploy the project in a stand-alone mode.

Schematic

- As mentioned earlier, we will connect the STM32F1 board to the 1.8″ ST7735 based coloured TFT Display along with a push button.
- The push button will be used to instruct the board to start the calculation.

- Connect the components as shown in the schematic below.


FTDI Basic
TX RX
DTR RXI TXO VCC CTS GND
GND GND 3V3 NRST B11 B10 B1 B0 A7 A6 A5 A4 A3 A2 A1 A0 C15 C14 C13 VBAT
B12 B13 B14 B15 A8 A9 A10 A11 A12 A15 B3 B4 B5 B6 B7 B8 B9 5V GND 3V3
8.000

To make the connections easy to replicate, the pin-to-pin connections between the STM32 and the display is described below.

STM32 – ST7735

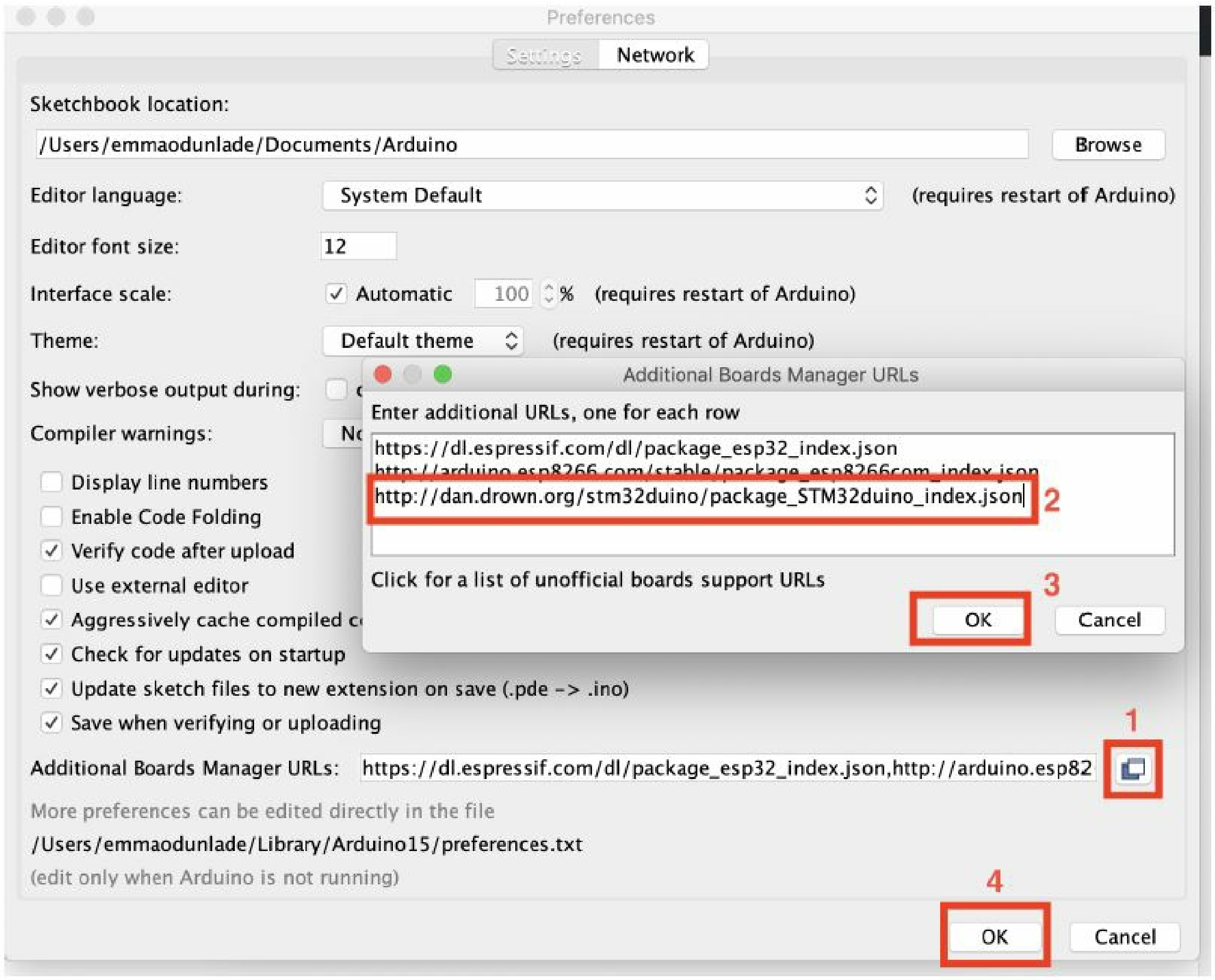
```
5V - VCC
GND - GND
PA2 - CS
PA3 - DC
PA4 - RST
PA5 - SCK
PA7 - SDA
3.3V - LED
```

Go over the connections once again to be sure everything is as it should be as it tends to get a little bit tricky. With this done, we proceeded to set up the STM32 board to be programmed with the Arduino IDE.

Setting up the Arduino IDE for STM32

- As with most boards not made by Arduino, a bit of setup needs to be done before the board can be used with the Arduino IDE.
- This involves installing the board file either via the Arduino Board Manager or downloading from the internet and copying the files into the hardware folder.
- The Board Manager route is the less tedious one and since the STM32F1 is among the listed boards, we will go that route. Start by adding the link for the STM32 board to the Arduino preference lists.
- Go to File -> Preferences, then enter this URL (

http://dan.drown.org/stm32duino/package_STM32duino_index.json) in the box as indicated below and click ok.



Sketchbook location: /Users/emmaodunlade/Documents/Arduino

Editor language: System Default (requires restart of Arduino)

Editor font size: 12

Interface scale: ✓ Automatic 100 % (requires restart of Arduino)

Theme: Default theme (requires restart of Arduino)

Show verbose output during:

Compiler warnings:

- [ ] Display line numbers
- [ ] Enable Code Folding
- [✓] Verify code after upload
- [ ] Use external editor
- [✓] Aggressively cache compiled c
- [✓] Check for updates on startup
- [✓] Update sketch files to new extension on save (.pde -> .ino)
- [✓] Save when verifying or uploading

Additional Boards Manager URLs: https://dl.espressif.com/dl/package_esp32_index.json,http://arduino.esp82

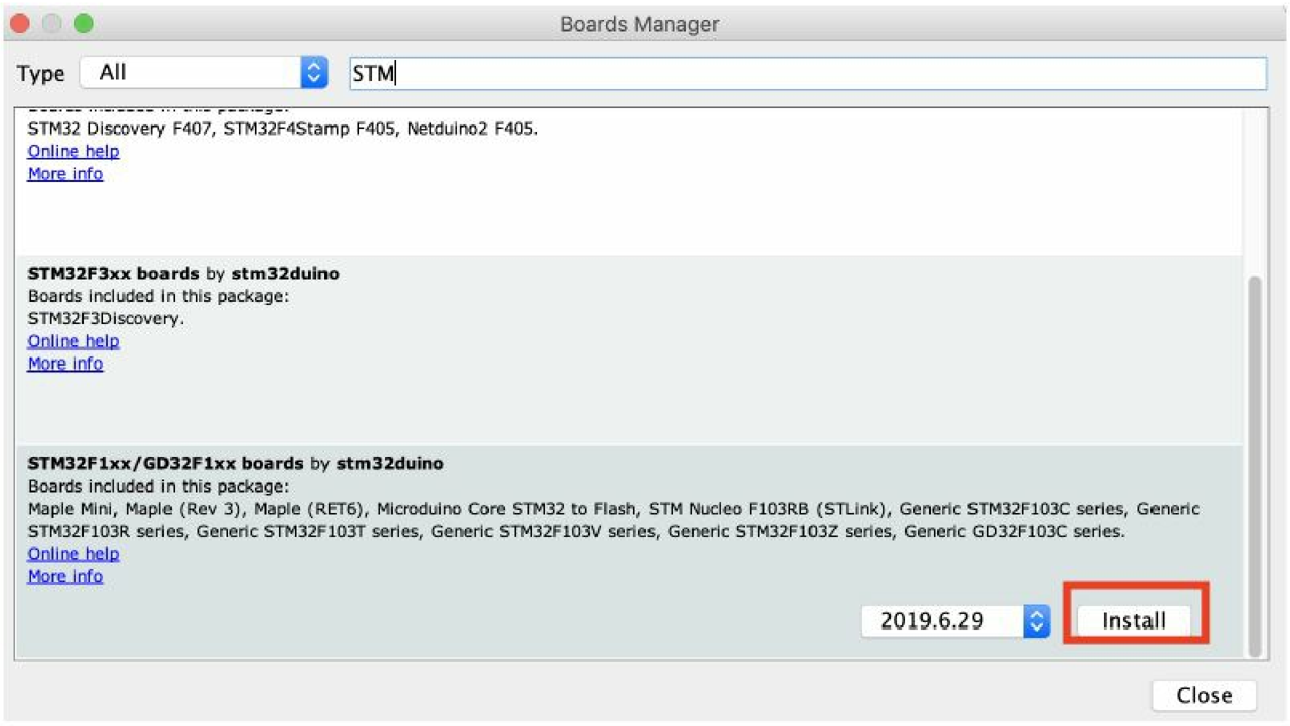
More preferences can be edited directly in the file
/Users/emmaodunlade/Library/Arduino15/preferences.txt
(edit only when Arduino is not running)

Enter additional URLs, one for each row

```
https://dl.espressif.com/dl/package_esp32_index.json
http://arduino.esp8266.com/stable/package_esp8266com_index.json
http://dan.drown.org/stm32duino/package_STM32duino_index.json
```

Click for a list of unofficial boards support URLs

- Now go to Tools -> Board -> Board Manager, it will open a dialogue box with a search bar. Search for STM32F1 and install the corresponding package.


Boards Manager

Type All STM

STM32 Discovery F407, STM32F4Stamp F405, Netduino2 F405.
Online help
More info

STM32F3xx boards by **stm32duino**
Boards included in this package:
STM32F3Discovery.
Online help
More info

STM32F1xx/GD32F1xx boards by **stm32duino**
Boards included in this package:
Maple Mini, Maple (Rev 3), Maple (RET6), Microduino Core STM32 to Flash, STM Nucleo F103RB (STLink), Generic STM32F103C series, Generic STM32F103R series, Generic STM32F103T series, Generic STM32F103V series, Generic STM32F103Z series, Generic GD32F103C series.
Online help
More info

- The installation procedure will take a few seconds. After that, the board should now be available for selection under the Arduino IDE board list.

Code

- The code will be written the same way we'd write any other sketch for an Arduino project, with the only difference being the way the pins are referenced.
- To be able to easily develop the code for this project, we will use two libraries which are both modifications of standard Arduino Libraries to make them compatible with the STM32.
- We will use the modified version of the Adafruit GFX and the Adafruit ST7735 libraries.
- Both libraries can be downloaded via the links attached to them. As usual, I will be doing a short breakdown of the code.
- We start the code by importing the two libraries that we will use.

```
#include <Adafruit_GFX_AS.h>
//https://github.com/rogerclarkmelbourne/Arduino_STM32/tree/master/STM32F1/librarie
s/Adafruit_GFX_AS

#include "Adafruit_ST7735.h" // https://github.com/KenjutsuGH/Adafruit-ST7735-
Library
```

- Next, we define the pins of the STM32 to which the CS, RST, and DC pins of the LCD are connected.

```
#define cs     PA2

#define rst    PA4

#define dc     PA3
```

- Next, we create some colour definitions to make it easy to use colours by their names in the code later instead of by their hex values.

```
// Color definitions

#define BLACK    0x0000

#define BLUE     0x001F

#define RED      0xF800

#define GREEN    0x07E0

#define CYAN     0x07FF

#define MAGENTA  0xF81F

#define YELLOW   0xFFE0

#define WHITE    0xFFFF
```

- Next, we set the number of iterations we want the board to go through along with the refresh duration for the progress bar to be used.

```
#define ITERATIONS 500000L    // number of iterations

#define REFRESH_TFT 7500      // refresh bar every 7500 iterations

#define ACTIVATED LOW
```

- With this done, we create an object of the ST7735 library which will be used to reference the display throughout the entire project.
- We also indicate the pin of the STM32 to which the pushbutton is connected and create a variable to hold its state.

```
Adafruit_ST7735 tft = Adafruit_ST7735(cs, dc, rst);



int percent = 0;

int progress = 1;

const int buttonPin = PB9;

int buttonState = 0;
```

- With this done, we move to the void setup() function.
- We start by setting the pinMode() of the pin to which the pushbutton is connected, activating an internal pull-up resistor on the pin since the pushbutton connects to the ground when pressed.

```
void setup(void) {

  pinMode(buttonPin, INPUT_PULLUP);
```

- Next, we initialize serial communication and the screen, setting the background of the display to black and calling the print () function to display the interface.

```
  Serial.begin(9600);

  tft.initR(INITR_BLACKTAB);   // initialize a ST7735S chip, black tab

  tft.fillScreen(ST7735_BLACK);

  printUI();

}
```

- Next is the void loop() function. The void loop function is quite simple and short, thanks to the use of libraries/functions.
- We start by reading the state of the push button. If the button has been pressed, we remove the current message on the screen using the removePressKeyText() and draw the changing progress bar using the drawBar() function.
- We then call the start calculation function to obtain and display the value of Pi along with the time it took to calculate it.

```
void loop() {

  buttonState = digitalRead(buttonPin);
  if (buttonState == ACTIVATED) {
    removePressKeyText();
    drawBar();
    startCalculation();
  }
```

- If the pushbutton is not pressed, the device stays in Idle mode with the screen demanding that a key be pressed to interact with it.

```
else {

  }
```

- Finally, a delay is inserted at the end of the loop to give a bit of time before sketching “loops”.

```
delay(10);

}
```

- The remaining part of the code is the functions called to achieve the tasks from drawing the bar to calculating the Pi.
- Most of these functions have been covered in several other tutorials that involve the use of the ST7735 display.

```
void fillBar(int percent)

{

    int counter =60;

    percent = map(percent,0,100,5,121);

    for(counter = 60; counter<75;counter++)

    {

     tft.drawFastHLine(5, counter, percent, YELLOW );

    }

}



void printUI()

{

 tft.setCursor(30,5);

 tft.setTextColor(RED);

 tft.setTextSize(1);

 tft.print("PI BENCHMARK");
 tft.setCursor(30,60);

 tft.setTextColor(WHITE);

 tft.setTextSize(1);

 tft.print("PRESS KEY");



 tft.setCursor(5,120);

 tft.setTextColor(RED);

 tft.setTextSize(1);

 tft.print("PI:");



 tft.setCursor(5,140);

 tft.setTextColor(RED);

 tft.setTextSize(1);

 tft.print("TIME:");

}



void removePressKeyText()

{

 tft.setCursor(30,60);

 tft.setTextColor(BLACK);

 tft.setTextSize(1);

 tft.print("PRESS KEY");

}



void drawBar()

{

  tft.drawRect(5,60,120,15, YELLOW);

}
```

```
void startCalculation()

{

 unsigned long start, time;

 unsigned long niter=ITERATIONS;

 int LEDcounter = 0;

 unsigned long i;

 float x = 1.0;

 float pi=1.0;



 Serial.begin(9600);

 Serial.print("Beginning ");

 Serial.print(niter);

 Serial.println(" iterations...");

 Serial.println();



 start = millis();

 for ( i = 2; i < niter; i++) {

   x *= -1.0;

   pi += x / (2.0f*(float)i-1.0f);

    if (LEDcounter++ > REFRESH_TFT) {
     LEDcounter = 0;

     progress++;

     percent = progress*100/(ITERATIONS/ REFRESH_TFT);

     fillBar(percent);

    }

 }

 time = millis() - start;



 pi = pi * 4.0;
 Serial.print("# of trials = ");

 Serial.println(niter);

 Serial.print("Estimate of pi = ");

 String piString = String(pi,7);

 String timeString = String(time)+"ms";

 Serial.println(piString);



 Serial.print("Time: "); Serial.print(time); Serial.println(" ms");



 tft.setCursor(40,120);

 tft.setTextColor(GREEN);

 tft.setTextSize(1);

 tft.print(piString);



 tft.setCursor(40,140);

 tft.setTextColor(GREEN);

 tft.setTextSize(1);

 tft.print(timeString);

}
```

- The complete code for the project is available below and is attached under the download section.

```
#include <Adafruit_GFX_AS.h>
//https://github.com/rogerclarkmelbourne/Arduino_STM32/tree/master/STM32F1/librarie
s/Adafruit_GFX_AS

#include "Adafruit_ST7735.h" // https://github.com/KenjutsuGH/Adafruit-ST7735-
Library



//Please note that you need to delete the Adafruit ST7735 library for Arduino if
you have it installed, else the

//sketch won't compile.
```

```
#define rst    PA4

#define dc     PA3


// Color definitions

#define BLACK    0x0000

#define BLUE     0x001F

#define RED      0xF800

#define GREEN    0x07E0

#define CYAN     0x07FF

#define MAGENTA  0xF81F

#define YELLOW   0xFFE0

#define WHITE    0xFFFF


#define ITERATIONS 500000L    // number of iterations

#define REFRESH_TFT 7500      // refresh bar every 7500 iterations

#define ACTIVATED LOW


#if defined(__SAM3X8E__)

    #undef __FlashStringHelper::F(string_literal)

    #define F(string_literal) string_literal

#endif


Adafruit_ST7735 tft = Adafruit_ST7735(cs, dc, rst);


int percent = 0;

int progress = 1;

const int buttonPin = PB9;
```

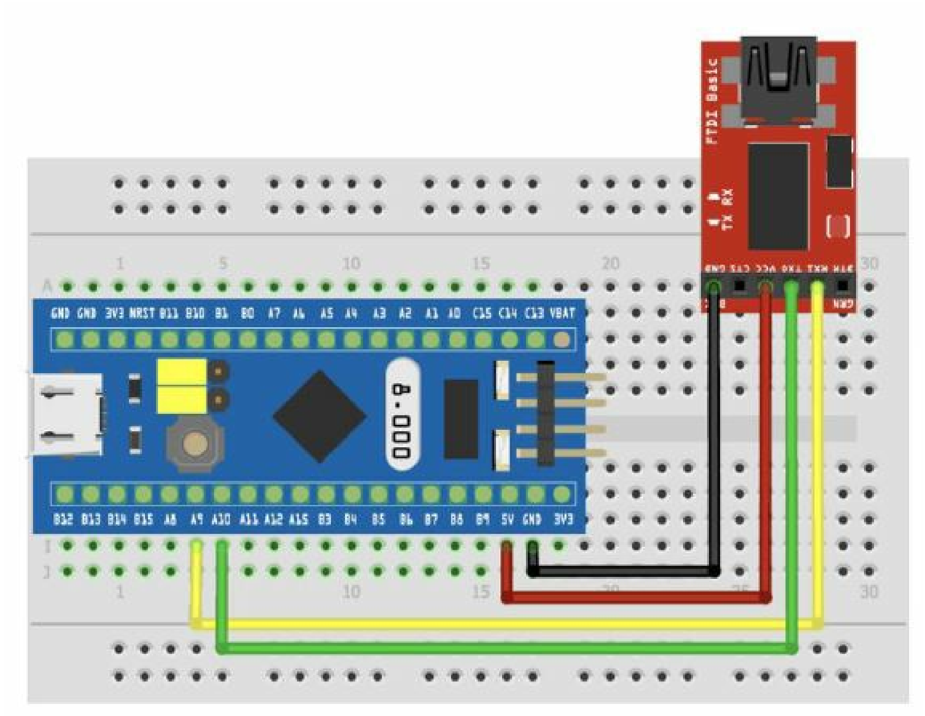
```
int buttonState = 0;


void setup(void) {


  pinMode(buttonPin, INPUT_PULLUP);


  Serial.begin(9600);


  tft.initR(INITR_BLACKTAB);   // initialize a ST7735S chip, black tab


  tft.fillScreen(ST7735_BLACK);


  printUI();


}


void loop() {



  buttonState = digitalRead(buttonPin);

  if (buttonState == ACTIVATED) {

    removePressKeyText();

    drawBar();

    startCalculation();

  }

  else {

  }


delay(10);

}
```

```
void fillBar(int percent)

{

    int counter =60;

    percent = map(percent,0,100,5,121);

    for(counter = 60; counter<75;counter++)

    {

     tft.drawFastHLine(5, counter, percent, YELLOW );

    }

}


void printUI()

{

 tft.setCursor(30,5);

 tft.setTextColor(RED);

 tft.setTextSize(1);

 tft.print("PI BENCHMARK");


 tft.setCursor(30,60);

 tft.setTextColor(WHITE);

 tft.setTextSize(1);

 tft.print("PRESS KEY");


 tft.setCursor(5,120);

 tft.setTextColor(RED);

 tft.setTextSize(1);

 tft.print("PI:");


 tft.setCursor(5,140);
 tft.setTextColor(RED);

 tft.setTextSize(1);

 tft.print("TIME:");

}



void removePressKeyText()

{

 tft.setCursor(30,60);

 tft.setTextColor(BLACK);

 tft.setTextSize(1);

 tft.print("PRESS KEY");

}



void drawBar()

{

  tft.drawRect(5,60,120,15, YELLOW);

}



void startCalculation()

{

 unsigned long start, time;

 unsigned long niter=ITERATIONS;

 int LEDcounter = 0;

 unsigned long i;

 float x = 1.0;

 float pi=1.0;



 Serial.begin(9600);

 Serial.print("Beginning ");
Serial.print(niter);

Serial.println(" iterations...");

Serial.println();


start = millis();

for ( i = 2; i < niter; i++) {

  x *= -1.0;

  pi += x / (2.0f*(float)i-1.0f);

   if (LEDcounter++ > REFRESH_TFT) {

    LEDcounter = 0;

    progress++;

    percent = progress*100/(ITERATIONS/ REFRESH_TFT);

    fillBar(percent);

   }

}

time = millis() - start;


pi = pi * 4.0;


Serial.print("# of trials = ");

Serial.println(niter);

Serial.print("Estimate of pi = ");

String piString = String(pi,7);

String timeString = String(time)+"ms";

Serial.println(piString);


Serial.print("Time: "); Serial.print(time); Serial.println(" ms");


tft.setCursor(40,120);
```

```
 tft.setTextColor(GREEN);

 tft.setTextSize(1);

 tft.print(piString);



 tft.setCursor(40,140);

 tft.setTextColor(GREEN);

 tft.setTextSize(1);

 tft.print(timeString);

}
```

Uploading Code to the STM32

- Uploading sketches to the STM32f1 is a little bit complex compared to standard Arduino-compatible boards. To upload code to the board, we need an FTDI-based, USB-to Serial converter.
- Connect the USB to serial converter to the STM32 as shown in the schematics below.

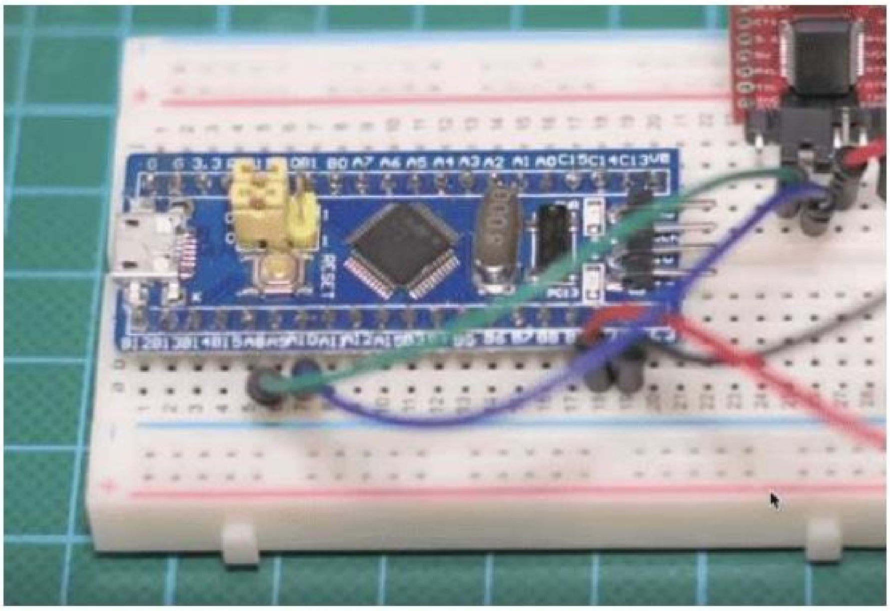

FTDI Basic
TX RX
GND CTS VCC TXO RXI DTR
GND GND 3V3 NRST B11 B10 B1 B0 A7 A6 A5 A4 A3 A2 A1 A0 C15 C14 C13 VBAT
B12 B13 B14 B15 A8 A9 A10 A11 A12 A15 B3 B4 B5 B6 B7 B8 B9 5V GND 3V3

Connect FTDI to STM32

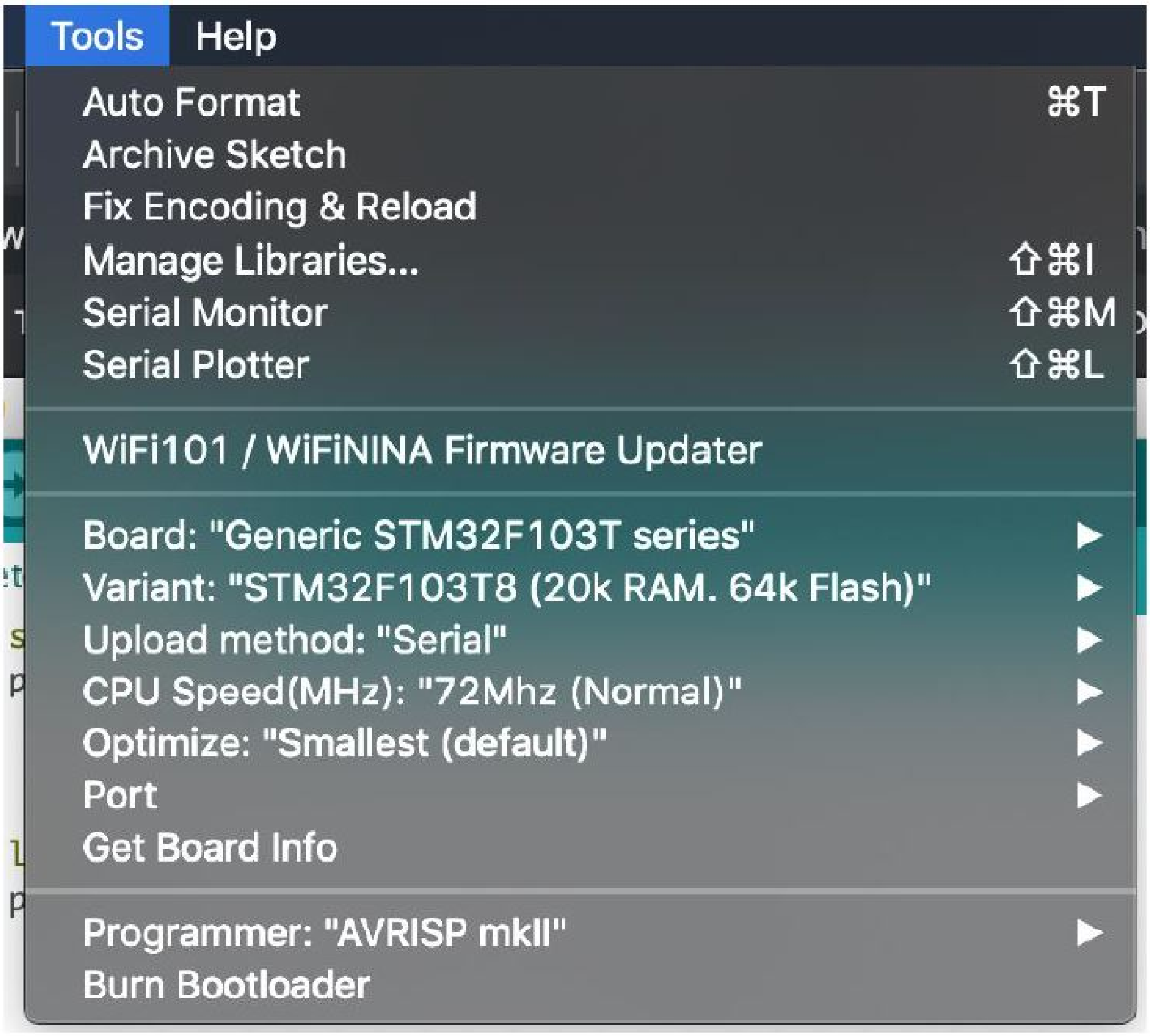
Here is a pin-to-pin map of the connection

FTDI – STM32

- With this done, we then change the position of the board's state jumper to position one (as shown in the gif below), to put the board in programming mode.
- Press the reset button on the board once after this and we are ready to upload the code.



- On the computer, ensure you select “Generic STM32F103C board” and select serial for the upload method after which you can hit the upload button.



Tools Help
Auto Format ⌘T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... ⇧⌘I
Serial Monitor ⇧⌘M
Serial Plotter ⇧⌘L
WiFi101 / WiFiNINA Firmware Updater
Board: "Generic STM32F103T series" ▶
Variant: "STM32F103T8 (20k RAM. 64k Flash)" ▶
Upload method: "Serial" ▶
CPU Speed(MHz): "72Mhz (Normal)" ▶
Optimize: "Smallest (default)" ▶
Port ▶
Get Board Info
Programmer: "AVRISP mkII" ▶
Burn Bootloader

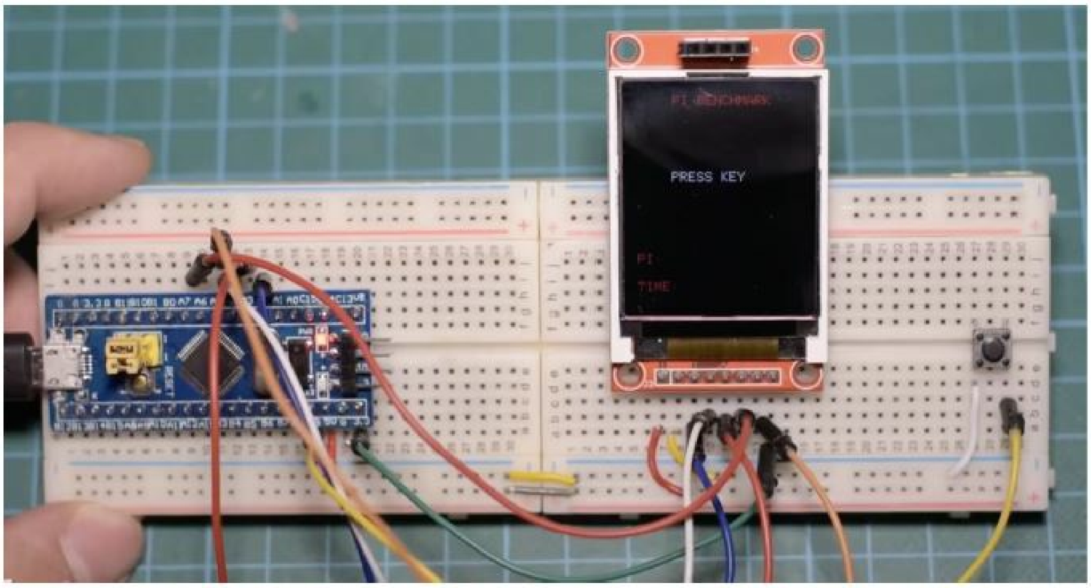
Select the board and set upload method to serial

- Once the Upload is complete, change the state jumper to position **“O”** This will put the board in “run” mode and it should now start running based on the code uploaded.

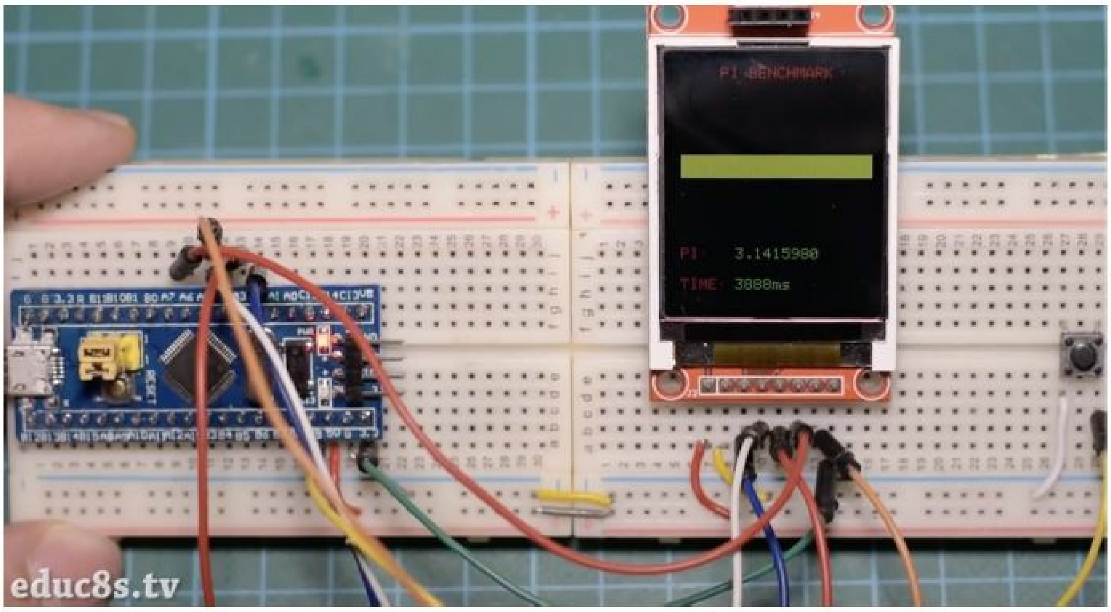
- At this point, you can disconnect the FTDI and power the board over its USB. In case the code does not run after powering, ensure you have restored the jumper properly and recycle power to the board.

Demo

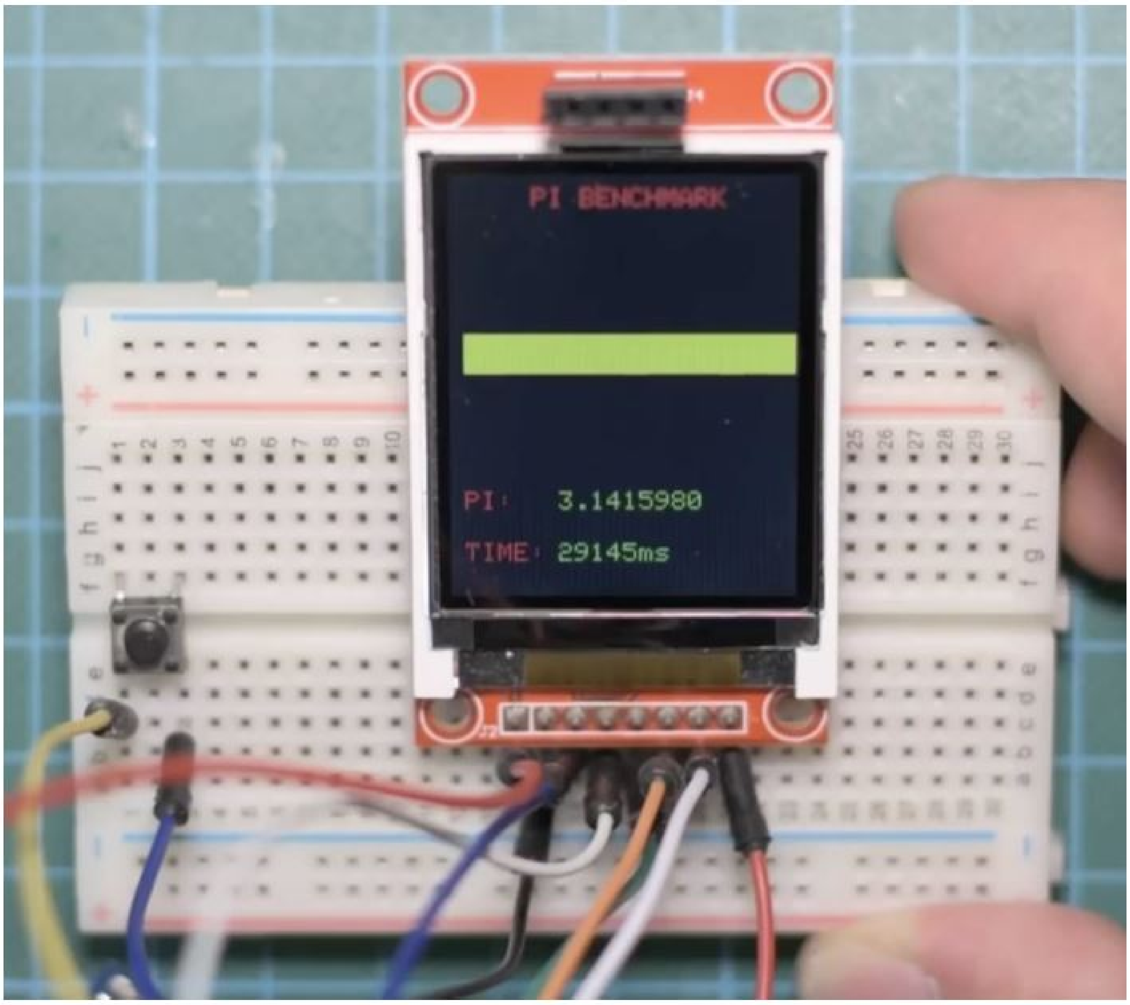
- With the code complete, follow the upload process described above to upload the code to your setup.
- You should see the display come up as shown in the image below.



- Press the push button to start the calculation. You should see the progress bar slide gradually until the end.
- At the end of the process, the value of Pi is displayed along with the time which the calculation took.



- The same code is implemented on an Arduino Uno. The result is shown in the image below.

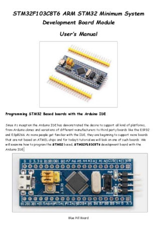


```
PI BENCHMARK

PI:   3.1415980
TIME: 29145ms
```

Arduino Speed Test

- Comparing these two values, we see that “Blue Pill” is over 7 times faster than the Arduino Uno.
- This makes it ideal for projects which involve heavy processing and time constraints.
- The small size of the Blue Pill also serves as an advantage here as it is only a bit bigger than the Arduino Nano and it can be used in places where the Nano won’t be fast enough.

Documents / Resources

	[STM32 STM32F103C8T6 Minimum System Development Board](#) [pdf] User Manual STM32F103C8T6 Minimum System Development Board, STM32F103C8T6, Minimum System Development Board, System Development Board, Development Board, Board

References

- [dan.drown.org/stm32duino/package_STM32duino_index.json](#)

- 1.8 inch 1.8" TFT LCD Display module ST7735S 128x160 51/AVR/STM32/ARM 8/16 bit | eBay
- FT232RL 3.3V 5.5V FTDI USB to TTL Serial Adapter Module for Arduino Mini Port US | eBay
- educ8s.tv/part/Powerbank
- Mini Universal Solderless Breadboard 400 Contacts Tie-points Available AL | eBay
- Dupont Jumper Wire Ribbon Cable 1P-1P for Arduino Breadboard F-F/M-M/F-M 40p | eBay
- GitHub - KenjutsuGH/Adafruit-ST7735-Library: This is a library for the Adafruit 1.8" SPI display http://www.adafruit.com/products/358 and http://www.adafruit.com/products/618
- Arduino_STM32/STM32F1/libraries/Adafruit_GFX_AS at master · rogerclarkmelbourne/Arduino_STM32 · GitHub
- Using the ST7735 1.8" Color TFT Display with Arduino - Electronics-Lab.com
- electronics-lab.com/wp-content/uploads/2019/06/Screen-Shot-2019-06-29-at-7.29.09-PM.png
- electronics-lab.com/wp-content/uploads/2019/06/Screen-Shot-2019-06-30-at-12.25.45-AM.png
- electronics-lab.com/wp-content/uploads/2019/06/Screen-Shot-2019-06-30-at-12.36.07-AM.png
- electronics-lab.com/wp-content/uploads/2019/06/Screen-Shot-2019-06-30-at-12.36.27-AM.png
- electronics-lab.com/wp-content/uploads/2019/06/Screen-Shot-2019-06-30-at-12.40.46-AM-1.png
- electronics-lab.com/wp-content/uploads/2019/08/jumper.gif
- User Manual