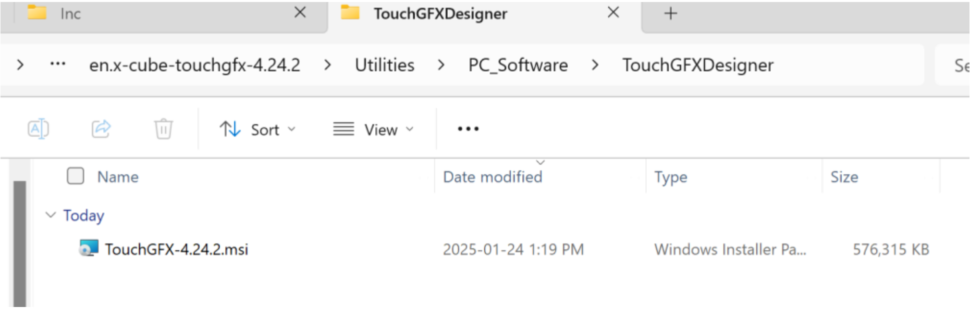
STM32F407 Discovery Kit with TFT LCD Display

Contents

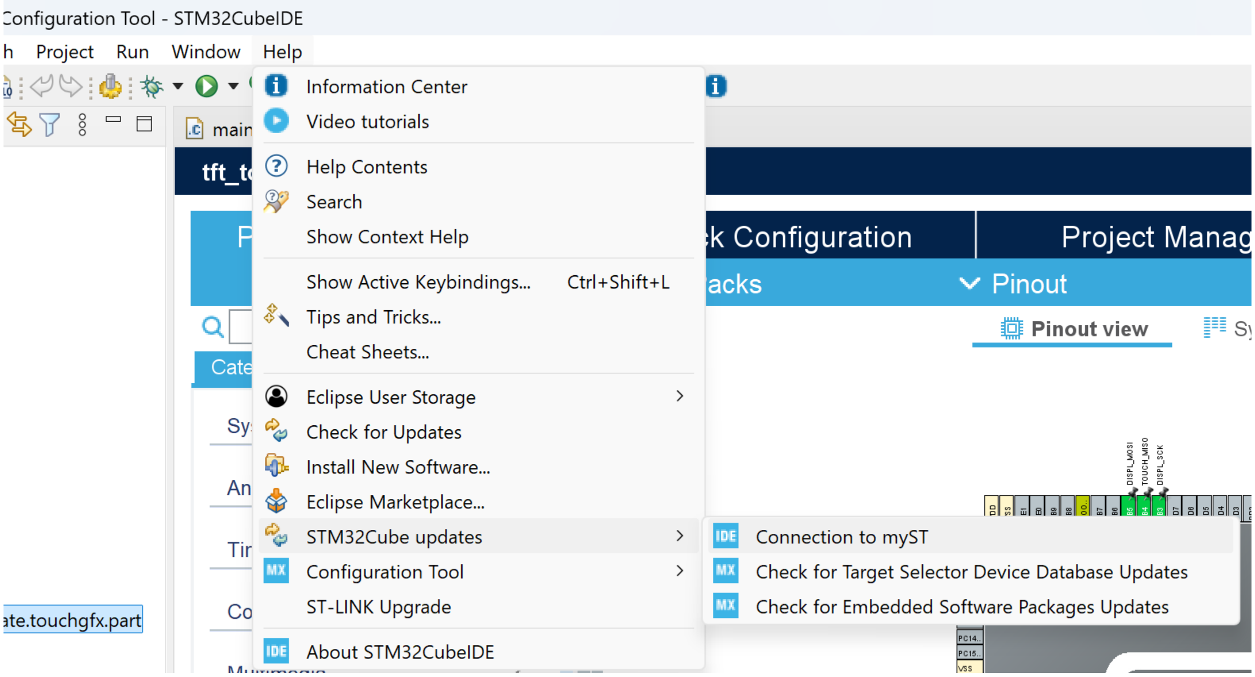
Create the project

Since this project will involve TouchGFX, begin by installing this graphical interface design software from https://www.st.com/en/embedded-software/x-cube-touchgfx.html. After extracting the zip file, navigate to `C:\Users\yourname\Downloads\en.x-cube-touchgfx-4.24.2\Utilities\PC_Software\TouchGFXDesigner` and double-click on the `.msi` file to open the installation wizard.

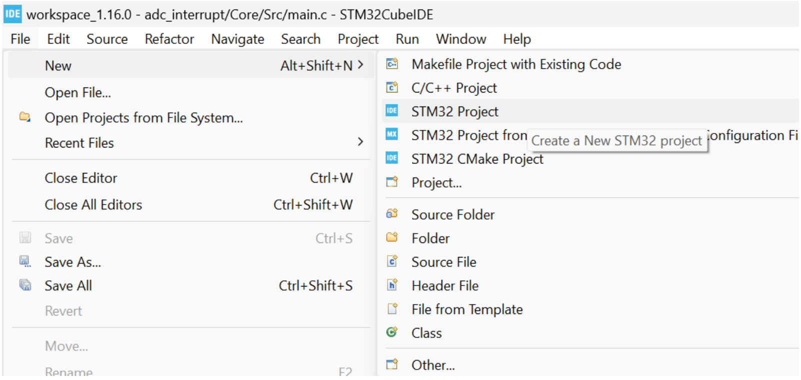

Inc
TouchGFXDesigner
en.x-cube-touchgfx-4.24.2 > Utilities > PC_Software > TouchGFXDesigner
Sort
View

Name	Date modified	Type	Size
Today			
TouchGFX-4.24.2.msi	2025-01-24 1:19 PM	Windows Installer Pa...	576,315 KB

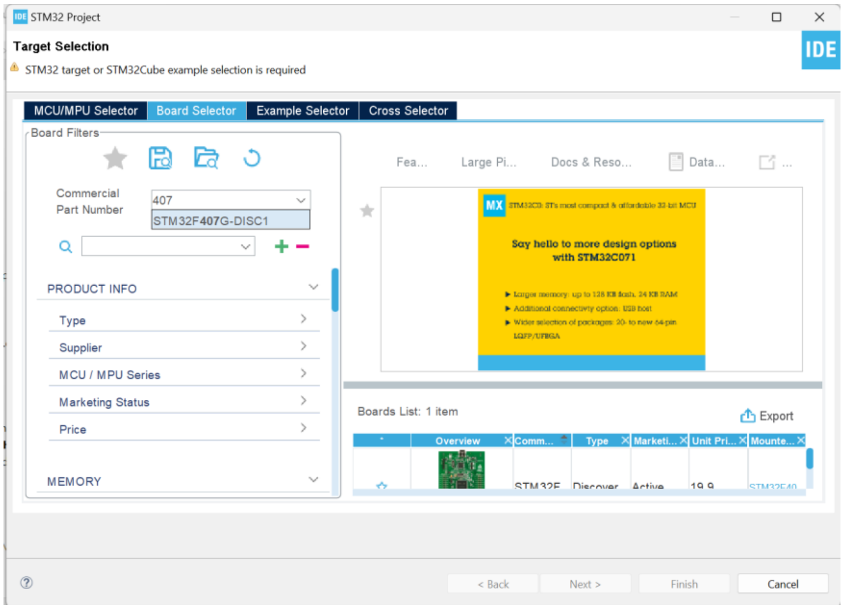
Note: If instead of building a project of your own you are opening an existing STM32 TouchGFX project within a new install of STM32CubeIDE, begin by signing in to myST at Help – STM32Cube updates – Connection to myST. When you click on the `.ioc` file for the project, support for TouchGFX will automatically install if you are signed in.



In STM32CubeIDE, choose File – New – STM32 Project.



A Target Selection window will open. Click on the Board Selector tab and type 407 in the Commercial Part Number box. Select the STM32F407-DISC1.


STM32 Project

Target Selection

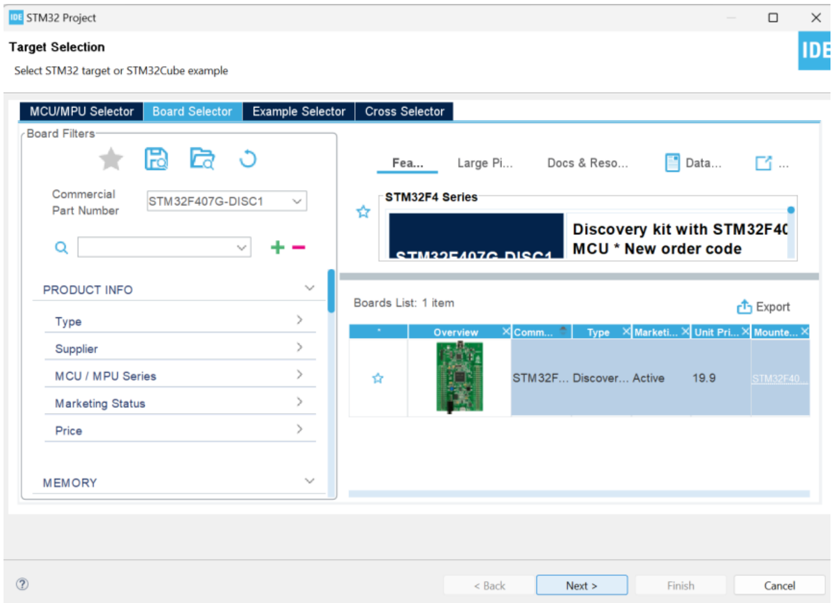
STM32 target or STM32Cube example selection is required

Commercial Part Number: 407

STM32F407G-DISC1

Boards List: 1 item

Click on the picture of the board on the right and choose Next.



STM32 Project

Target Selection

Select STM32 target or STM32Cube example

Commercial Part Number: STM32F407G-DISC1

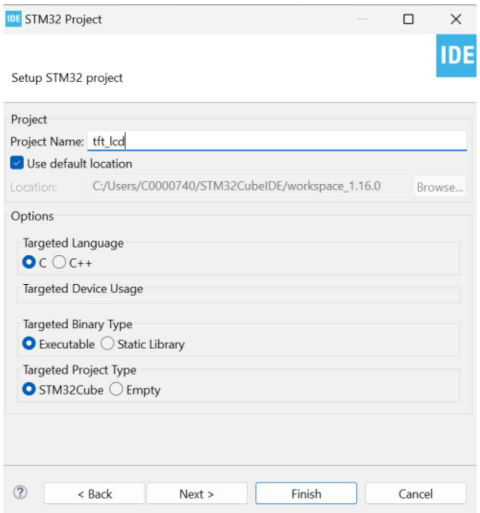
STM32F4 Series

Discovery kit with STM32F40 MCU * New order code

Boards List: 1 item

*	Overview	Comm...	Type	Marketi...	Unit Pri...	Mounte...
☆		STM32F...	Discover...	Active	19.9	STM32F40

Name your project `tft_lcd` and click Finish.



STM32 Project

Setup STM32 project

Project

Project Name: tft_lcd

☑ Use default location

Location: C:/Users/C0000740/STM32CubeIDE/workspace_1.16.0

Options

Targeted Language

● C ○ C++

Targeted Device Usage

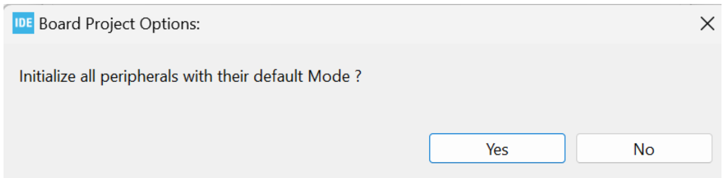
Targeted Binary Type

● Executable ○ Static Library

Targeted Project Type

● STM32Cube ○ Empty

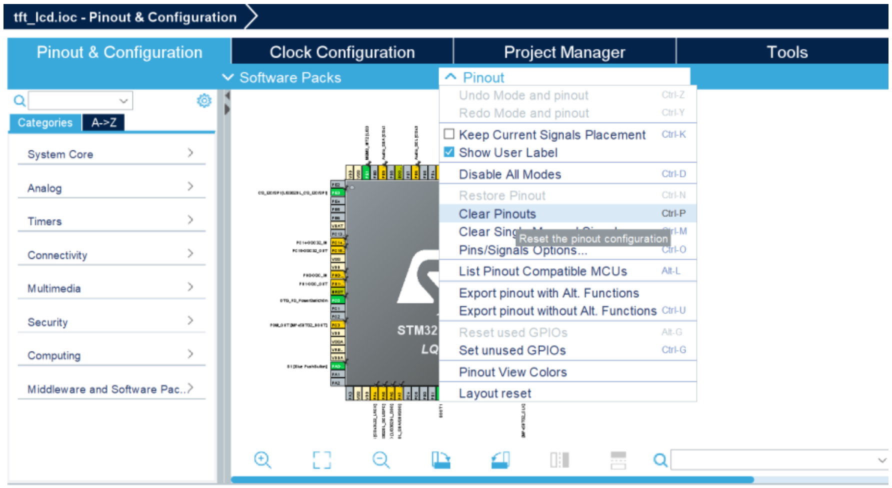
Click No when prompted to initialize peripherals. You will set up peripherals yourself.



Board Project Options:

Initialize all peripherals with their default Mode ?

When the configuration file opens, choose Clear Pinouts and say Yes to confirm.


tft_lcd.ioc - Pinout & Configuration
Pinout & Configuration
Clock Configuration
Project Manager
Tools
Software Packs
Pinout
Undo Mode and pinout Ctrl-Z
Redo Mode and pinout Ctrl-Y
□ Keep Current Signals Placement Ctrl-K
☑ Show User Label
Disable All Modes Ctrl-D
Restore Pinout Ctrl-N
Clear Pinouts Ctrl-P
Clear Sing... Ctrl-M
Reset the pinout configuration
Pins/Signals Options... Ctrl-O
List Pinout Compatible MCUs Alt-L
Export pinout with Alt. Functions
Export pinout without Alt. Functions Ctrl-U
Reset used GPIOs Alt-G
Set unused GPIOs Ctrl-G
Pinout View Colors
Layout reset
Categories
A->Z
System Core
Analog
Timers
Connectivity
Multimedia
Security
Computing
Middleware and Software Pac...
STM32
LQ

Configure the project

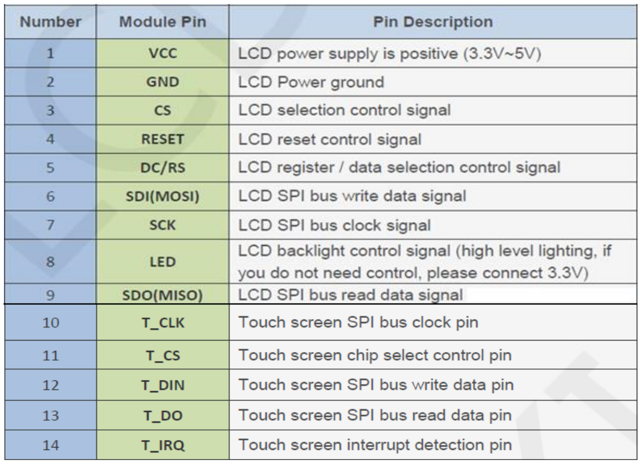
The ILI9341 TFT LCD thin film display offers both a liquid crystal display (LCD) and a thin film transistor (TFT) touchscreen.

The TFT LCD display has 14 pins, including power and ground. Key LCD pins are:

CS	LCD selection control signal
RESET	LCD reset control signal
DC/RS	LCD register/data selection control signal
SDI MOSI	LCD SPI bus write data signal
SCK	LCD SPI bus clock signal
LED	LCD backlight control signal

Key TFT pins are:

T_CLK	Touchscreen SPI bus clock signal
T_CS	Touchscreen selection control signal
T_DIN	Touchscreen SPI bus write signal
T_DO	Touchscreen SPI bus read signal
T_IRQ	Touchscreen interrupt detection pin



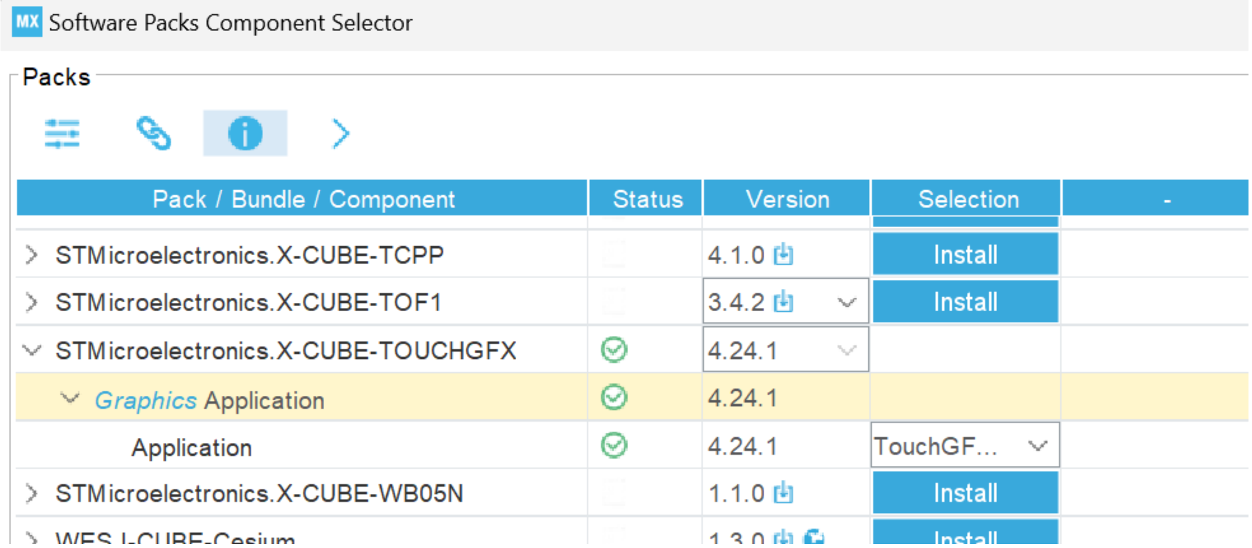
Number	Module Pin	Pin Description
1	VCC	LCD power supply is positive (3.3V~5V)
2	GND	LCD Power ground
3	CS	LCD selection control signal
4	RESET	LCD reset control signal
5	DC/RS	LCD register / data selection control signal
6	SDI(MOSI)	LCD SPI bus write data signal
7	SCK	LCD SPI bus clock signal
8	LED	LCD backlight control signal (high level lighting, if you do not need control, please connect 3.3V)
9	SDO(MISO)	LCD SPI bus read data signal
10	T_CLK	Touch screen SPI bus clock pin
11	T_CS	Touch screen chip select control pin
12	T_DIN	Touch screen SPI bus write data pin
13	T_DO	Touch screen SPI bus read data pin
14	T_IRQ	Touch screen interrupt detection pin

https://www.dragonwake.com/download/LCD/2.8inch_spi/2.8inch_SPI_Module_MSP2807_User_Manual_EN.pdf

The following directions are based on https://github.com/meldundas/ILI9XXX-XPT2046-STM32.

Open `tft_lcd.ioc`.

Under Middleware and Software Packs, locate TouchGFX and Install. After installation, make sure the Graphics Application TouchGFX Generator is selected.

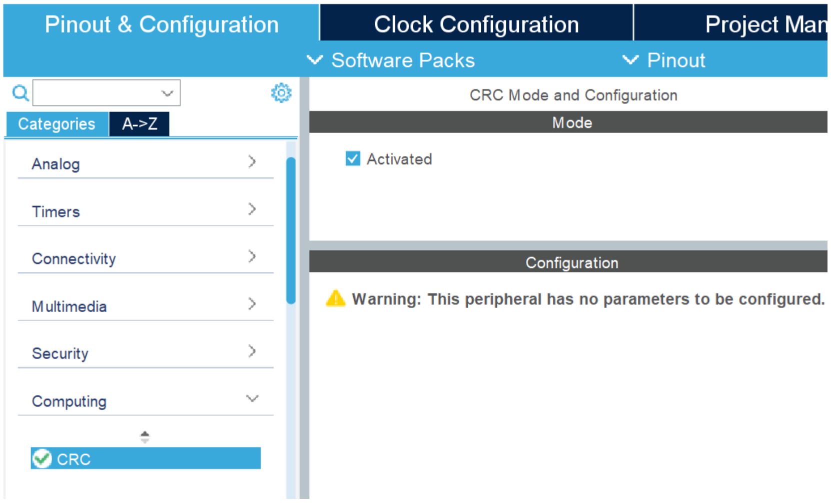


Software Packs Component Selector

Packs

Pack / Bundle / Component	Status	Version	Selection	-
STMicroelectronics.X-CUBE-TCPP		4.1.0	Install	
STMicroelectronics.X-CUBE-TOF1		3.4.2	Install	
STMicroelectronics.X-CUBE-TOUCHGFX	✓	4.24.1		
Graphics Application	✓	4.24.1		
Application	✓	4.24.1	TouchGF...	
STMicroelectronics.X-CUBE-WB05N		1.1.0	Install	
WES I-CUBE-Cesium		1.3.0	Install	

Once installation is complete, close the software packs component selector. Go to Computing – CRC and activate it. CRC is required for TouchGFX.



CRC Mode and Configuration

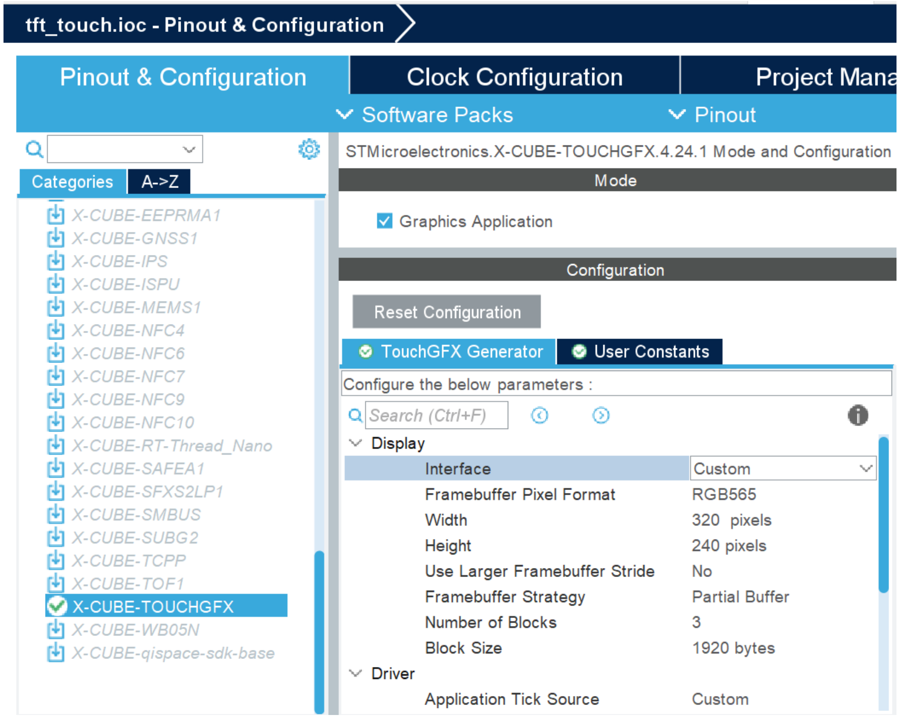
Mode

☑ Activated

Configuration

Warning: This peripheral has no parameters to be configured.

Return to Middleware and Software Packs, select X-CUBE-TOUCHGFX and activate by checking Graphics Application. Enter the parameters shown for the TFT LCD display, which is 320 pixels wide and 240 pixels high. Partial buffering lowers required memory usage.



STMicroelectronics.X-CUBE-TOUCHGFX.4.24.1 Mode and Configuration

Mode

☑ Graphics Application

Configuration

Configure the below parameters :

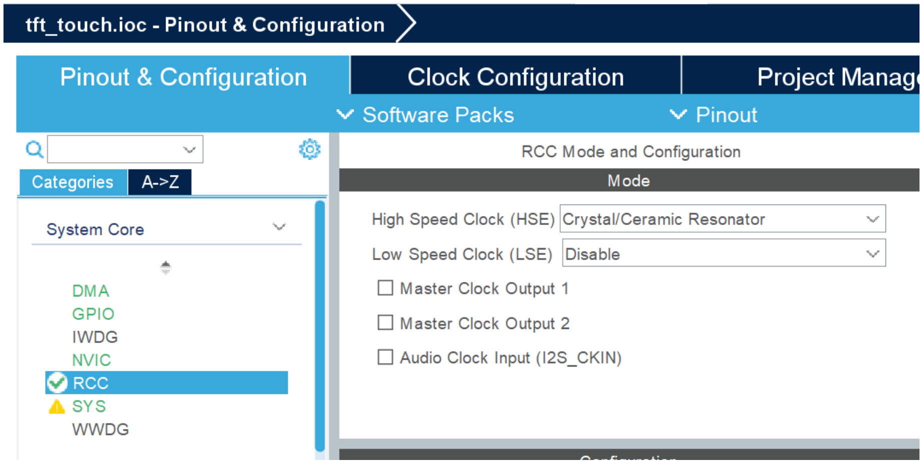
Display

Interface	Custom
Framebuffer Pixel Format	RGB565
Width	320 pixels
Height	240 pixels
Use Larger Framebuffer Stride	No
Framebuffer Strategy	Partial Buffer
Number of Blocks	3
Block Size	1920 bytes

Driver

Application Tick Source	Custom

Under System Core, select RCC and choose Crystal/Ceramic Resonator for the High Speed Clock.



RCC Mode and Configuration

Mode

High Speed Clock (HSE) Crystal/Ceramic Resonator

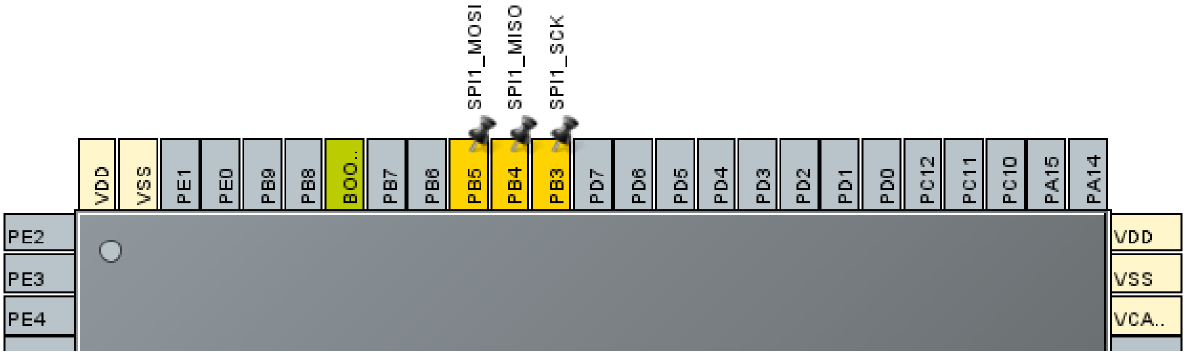
Low Speed Clock (LSE) Disable

☐ Master Clock Output 1

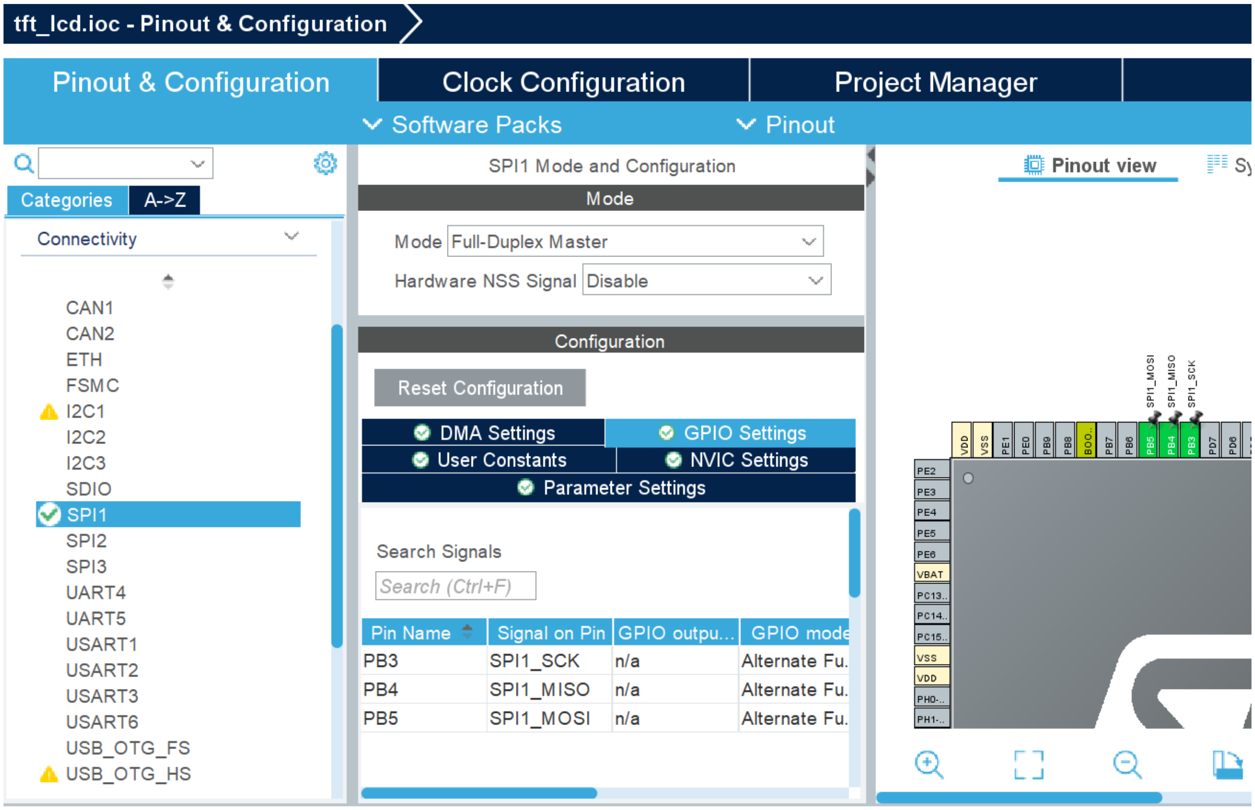
☐ Master Clock Output 2

☐ Audio Clock Input (I2S_CKIN)

In this project, a single serial peripheral interface (SPI) will be used to interact with both the LCD and the TFT. Click PB3 and select SPI1_SCK; click PB4 and select SPI1_MISO; click PB5 and select SPI_MOSI. (Note that there are many other pins that could serve these roles equally well.)



Under Connectivity, select SPI1 and choose Full Duplex Master mode. The GPIO settings confirm that pins PB3, PB4, and PB5 will be used for the SPI.



SPI1 Mode and Configuration

Mode

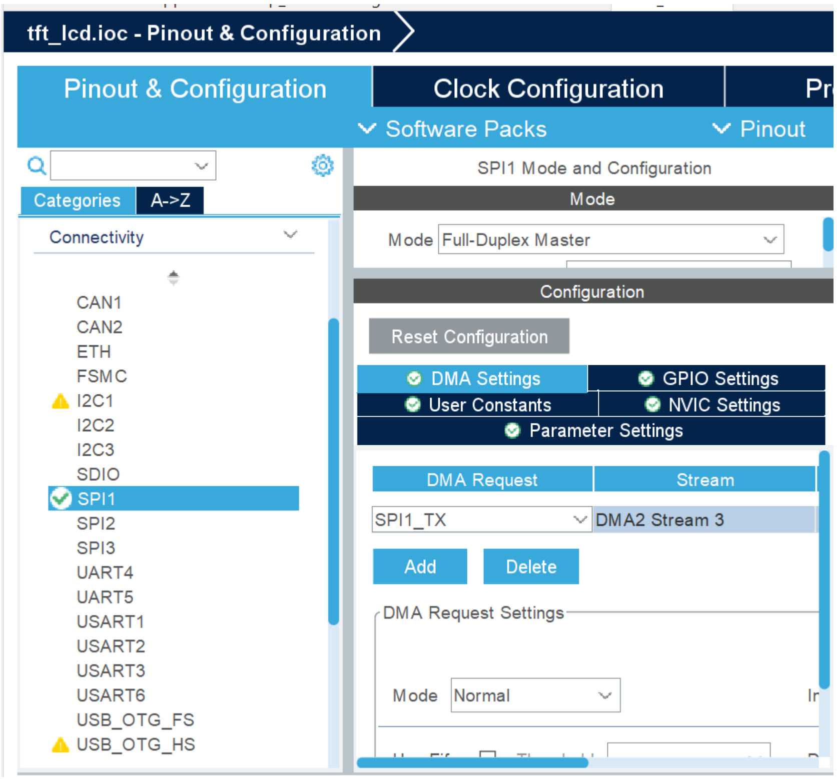
Mode Full-Duplex Master

Hardware NSS Signal Disable

Configuration

Pin Name	Signal on Pin	GPIO outpu...	GPIO mode
PB3	SPI1_SCK	n/a	Alternate Fu.
PB4	SPI1_MISO	n/a	Alternate Fu.
PB5	SPI1_MOSI	n/a	Alternate Fu.

Parameter Settings for SPI1 may be left as is. On the DMA settings, click Add and choose SPI1_TX.



SPI1 Mode and Configuration

Mode

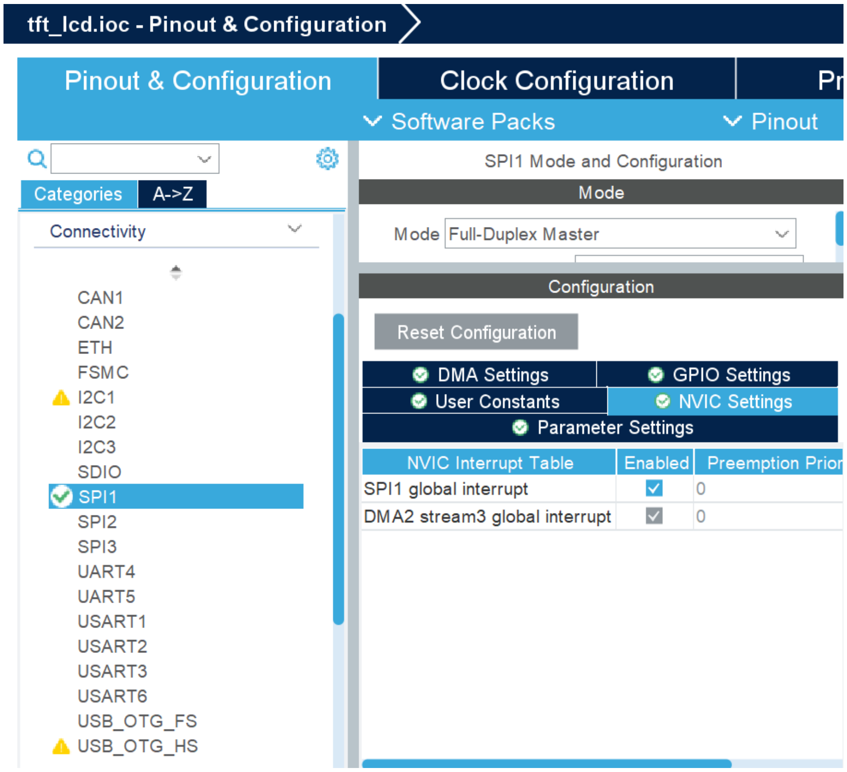
Mode Full-Duplex Master

DMA Request	Stream
SPI1_TX	DMA2 Stream 3

DMA Request Settings

Mode Normal

On the NVIC Settings tab, enable global interrupts for SPI1.



SPI1 Mode and Configuration

Mode

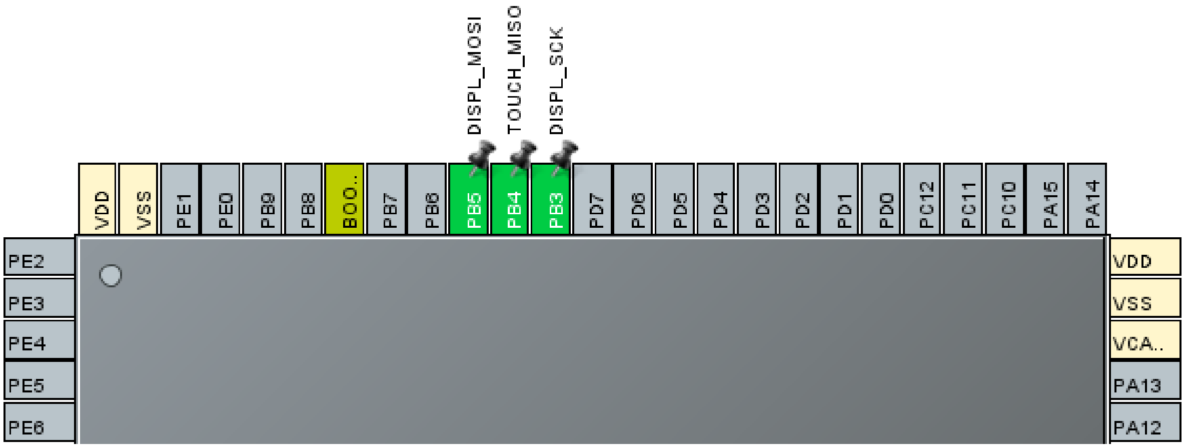
Mode Full-Duplex Master

NVIC Interrupt Table	Enabled	Preemption Prior
SPI1 global interrupt	☑	0
DMA2 stream3 global interrupt	☑	0

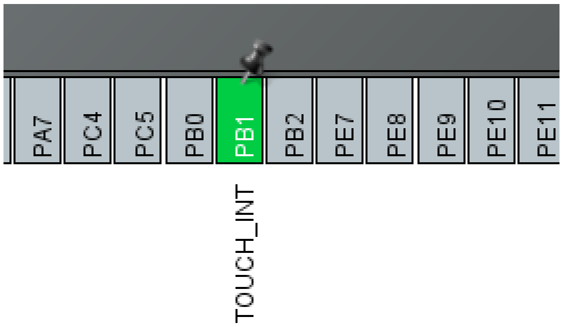
On the Pinout view, right-click on PB3, PB4, and PB5 to enter the following user labels:

PB3	DISPL_SCK
PB4	TOUCH_MISO
PB5	DISPL_MOSI

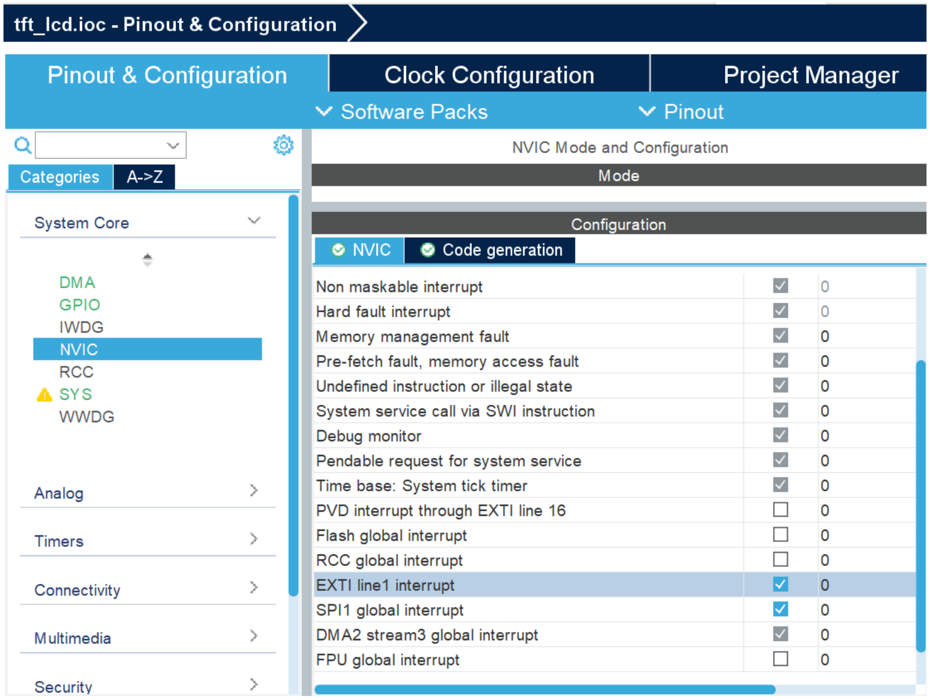
These labels reflect the fact that the slave LCD display will be receiving information from the master MCU (microcontroller unit) (MOSI: master out, slave in), while the slave touchscreen will be sending information to the master MCU (MISO: master in, slave out).



Click on pin PB1 and select GPIO_EXTI1. (Several other pins offer the same functionality.) Right-click on PB1 to enter the User Label TOUCH_INT. This pin will receive interrupts from the touchscreen.



Under System Core – NVIC, enable EXT line1 interrupt.



tft_lcd.ioc - Pinout & Configuration

Pinout & Configuration | Clock Configuration | Project Manager

Software Packs | Pinout

Categories | A->Z

System Core: DMA, GPIO, IWDG, NVIC, RCC, SYS, WWDG

Analog, Timers, Connectivity, Multimedia, Security

NVIC Mode and Configuration

Mode

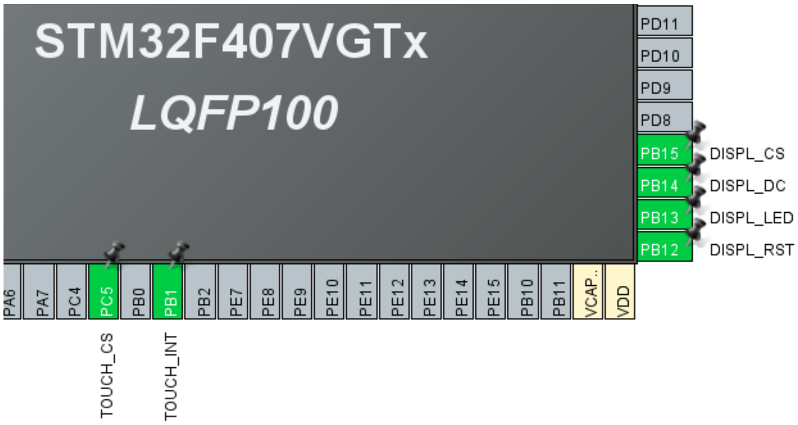
Configuration

NVIC | Code generation

Non maskable interrupt	☑	0
Hard fault interrupt	☑	0
Memory management fault	☑	0
Pre-fetch fault, memory access fault	☑	0
Undefined instruction or illegal state	☑	0
System service call via SWI instruction	☑	0
Debug monitor	☑	0
Pendable request for system service	☑	0
Time base: System tick timer	☑	0
PVD interrupt through EXTI line 16	□	0
Flash global interrupt	□	0
RCC global interrupt	□	0
EXTI line1 interrupt	☑	0
SPI1 global interrupt	☑	0
DMA2 stream3 global interrupt	☑	0
FPU global interrupt	□	0

Define each of the following pins as GPIO_Output, with the names and parameters shown. GPIO settings may be found under System Core – GPIO by clicking on the pin. Again, the pins used here are not unique, and many other choices would have the same functionality. Note that these DISPL_LED settings set on/off mode for the LCD backlight.

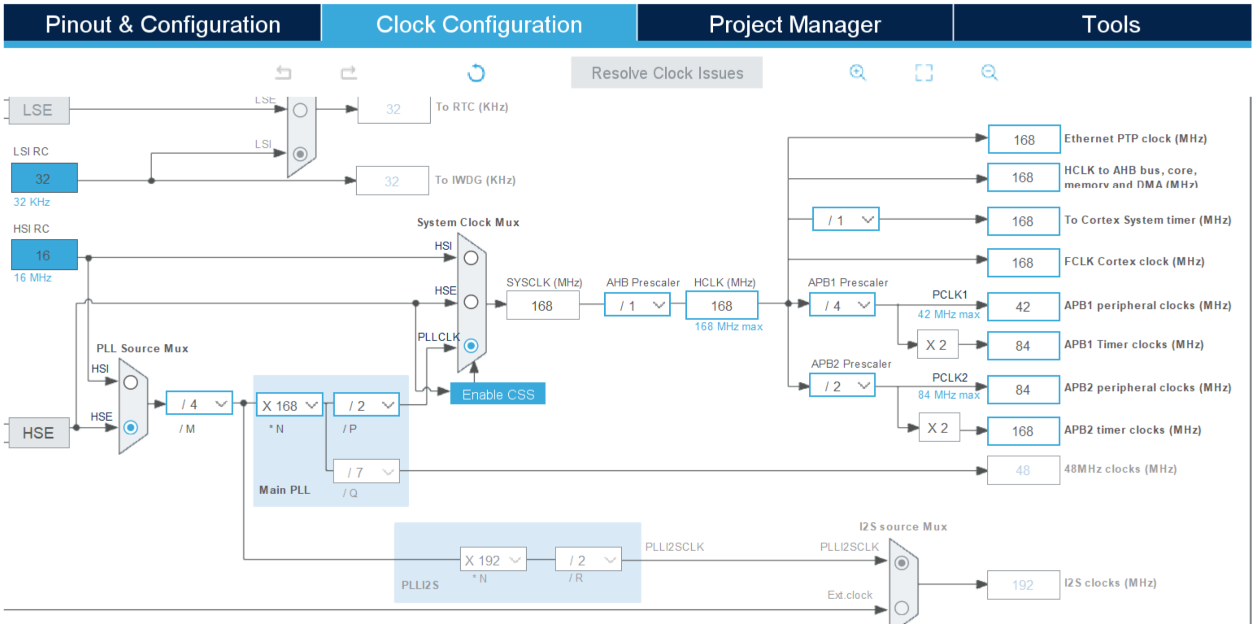
Pin	**User Label**	**Output Level**	**Mode**	**Pull up/down**
PB13	DISPL_LED	low	Output Push Pull	No pull-up and no pull-down
PB14	DISPL_DC	low	Output Push Pull	No pull-up and no pull-down
PB12	DISPL_RST	low	Output Push Pull	No pull-up and no pull-down
PB15	DISPL_CS	high	Output Push Pull	No pull-up and no pull-down
PC5	TOUCH_CS	high	Output Push Pull	No pull-up and no pull-down



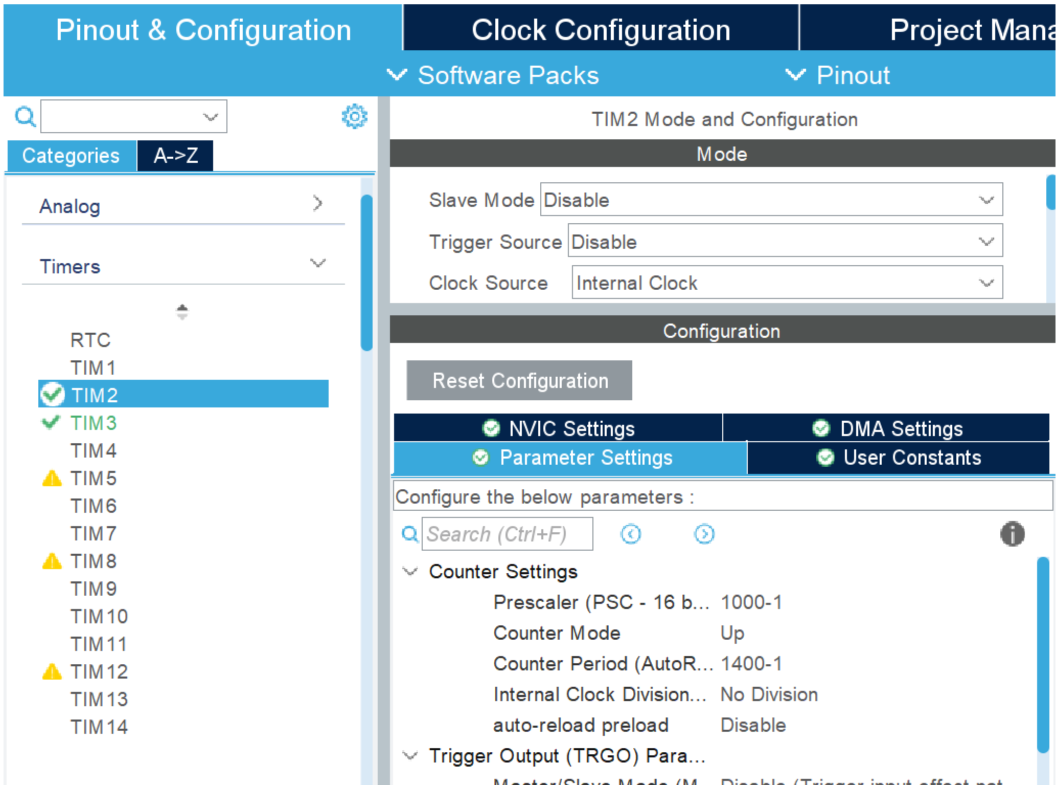
Set up a 60 Hz timer using TIM2, with Internal Clock source. Recall that the ADC sampling frequency is given by:

$$\text{sampling frequency} = \frac{\text{APB2 peripheral clock frequency}}{(\text{Prescaler} + 1)(\text{Counter Period} + 1)}$$

where the APB2 peripheral clock frequency can be verified on the Clock Configuration tab to be 84 MHz, when the HCLK is set to 168 MHz.



One possible solution to obtain 60 Hz is shown.



TIM2 Mode and Configuration

Mode

Slave Mode: Disable

Trigger Source: Disable

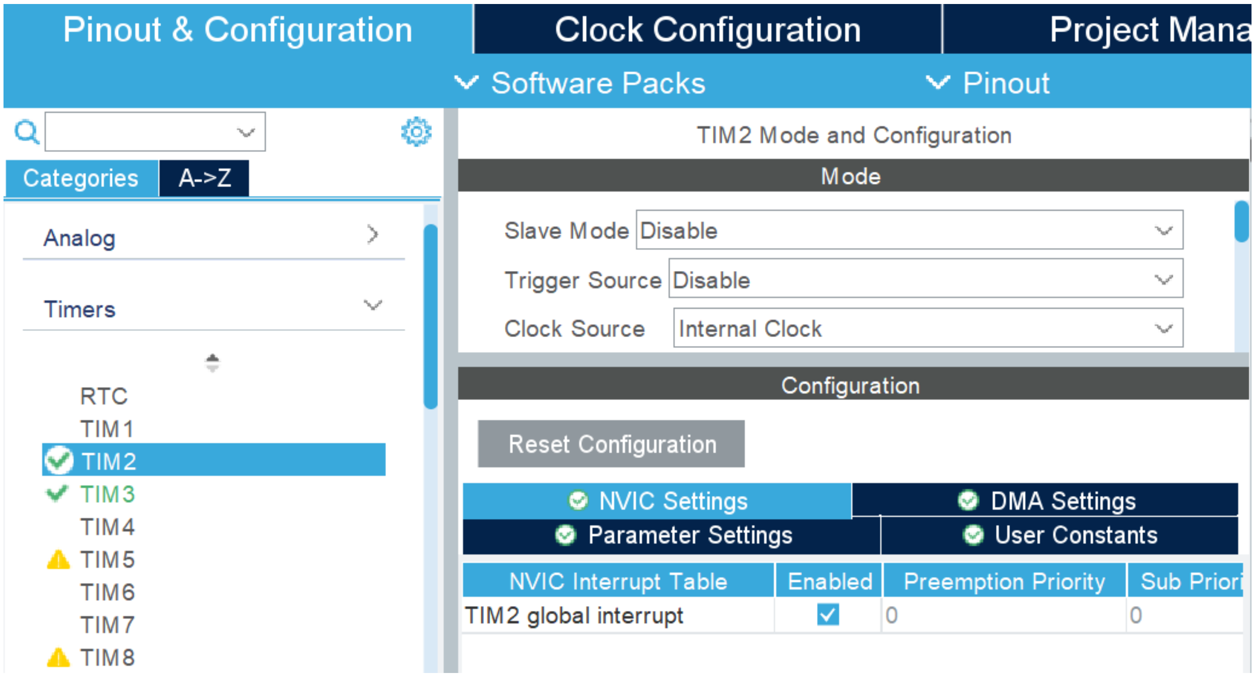
Clock Source: Internal Clock

Configuration

Configure the below parameters :

Counter Settings	
Prescaler (PSC - 16 b...	1000-1
Counter Mode	Up
Counter Period (AutoR...	1400-1
Internal Clock Division...	No Division
auto-reload preload	Disable
Trigger Output (TRGO) Para...	
Master/Slave Mode (M...	Disable (Trigger input effect not

Enable the interrupt for this timer.



TIM2 Mode and Configuration

Mode

Slave Mode: Disable

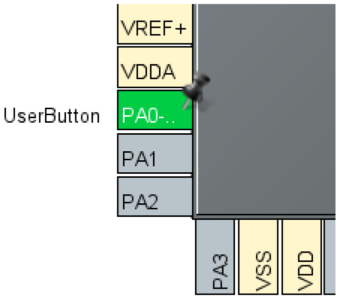
Trigger Source: Disable

Clock Source: Internal Clock

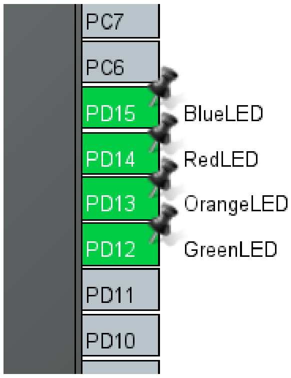
Configuration

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priori
TIM2 global interrupt	☑	0	0

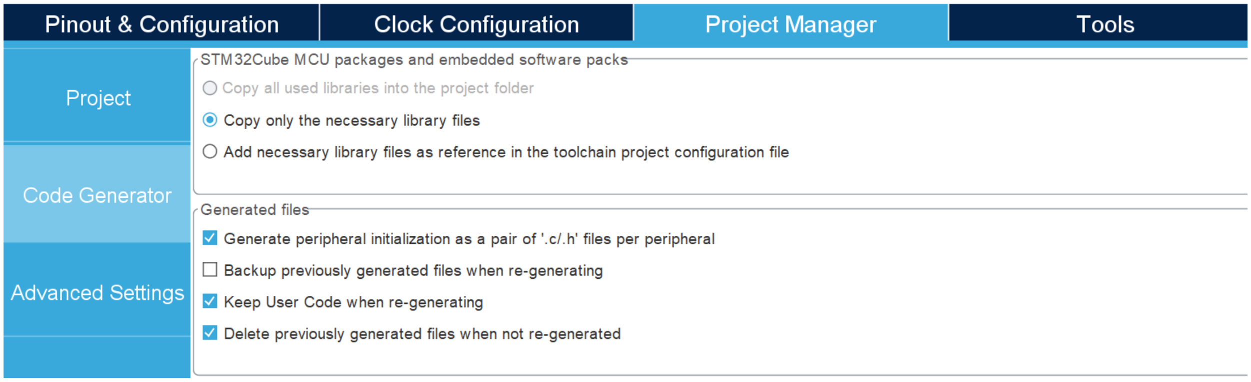
Later in this project you will use the user button on our Discovery board and also the LEDs. For convenience, you can configure them now. Make PA0 a GPIO Input and rename it `UserButton`. PA0 is the pin connected to the user button on the board.



The LEDs on the Discovery board are connected to pins PD12, PD13, PD14, and PD15. Make them GPIO Outputs and rename them as shown.



On the Project Manager tab, choose Code Generator and check off "Generate peripheral initialization as a pair of '.c/.h' files per peripheral."



STM32Cube MCU packages and embedded software packs

○ Copy all used libraries into the project folder

● Copy only the necessary library files

○ Add necessary library files as reference in the toolchain project configuration file

Generated files

☑ Generate peripheral initialization as a pair of '.c/.h' files per peripheral

□ Backup previously generated files when re-generating

☑ Keep User Code when re-generating

☑ Delete previously generated files when not re-generated

Generate code.

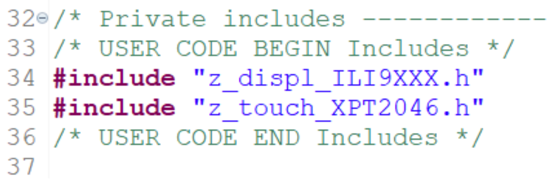
Adding driver files to the project

Download `ILI9XXX-XPT2046-STM32-main.zip` from https://github.com/meldundas/ILI9XXX-XPT2046-STM32.

Extract the files `z_displ_ILI9XXX.c` and `z_touch_XPT2046.c` and place them in the `Core/Src` folder of your project.

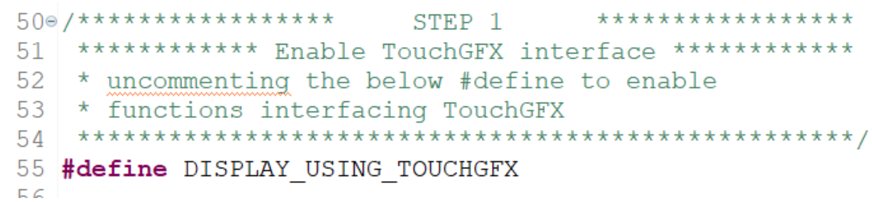
Extract the files `z_displ_ILI9XXX.h` and `z_touch_XPT2046.h` and place them in the `Core/Inc` folder of your project.

In STM32CubeIDE, from `Core/Inc`, open `main.h`. Add lines to include the `.h` files.



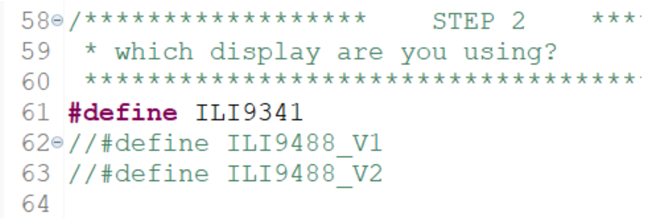
```
/* Private includes ------------
/* USER CODE BEGIN Includes */
#include "z_displ_ILI9XXX.h"
#include "z_touch_XPT2046.h"
/* USER CODE END Includes */

```

Open `z_displ_ILI9XXX.h`. Uncomment the `#define` to allow display using TouchGFX.



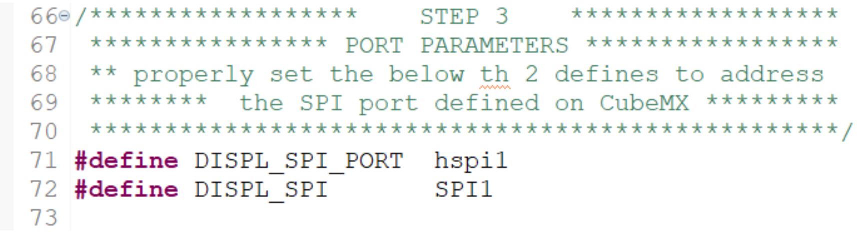
```
/*****************      STEP 1      *****************
 ************ Enable TouchGFX interface ************
 * uncommenting the below #define to enable
 * functions interfacing TouchGFX
 ***************************************************/
#define DISPLAY_USING_TOUCHGFX
```

Select the display you are using.



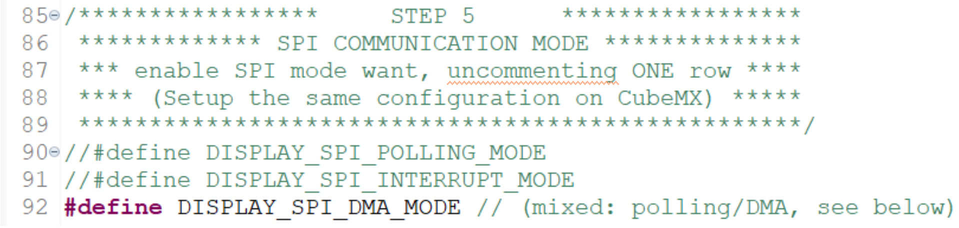
```
/*****************      STEP 2     ***
 * which display are you using?
 **********************************
#define ILI9341
//#define ILI9488_V1
//#define ILI9488_V2

```

Make sure the SPI port agrees with the SPI you are using, `SPI1`.



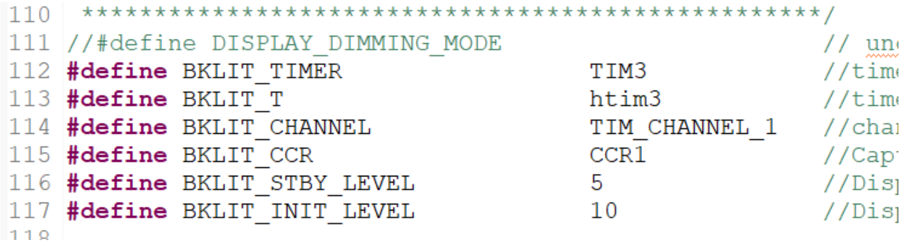
```
/*****************      STEP 3     *****************
 *************** PORT PARAMETERS ****************
 ** properly set the below th 2 defines to address
 ********  the SPI port defined on CubeMX *********
 **************************************************/
#define DISPL_SPI_PORT  hspi1
#define DISPL_SPI       SPI1

```

Select DMA mode for your SPI.



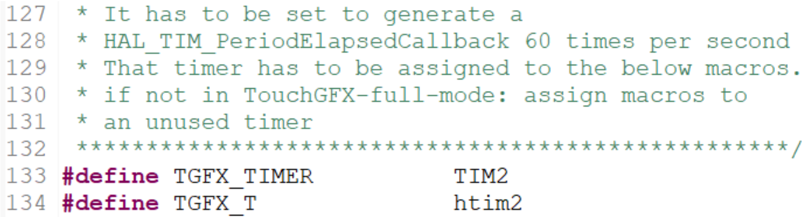
```
/*****************      STEP 5      *****************
 ************* SPI COMMUNICATION MODE **************
 *** enable SPI mode want, uncommenting ONE row ****
 **** (Setup the same configuration on CubeMX) *****
 ***************************************************/
//#define DISPLAY_SPI_POLLING_MODE
//#define DISPLAY_SPI_INTERRUPT_MODE
#define DISPLAY_SPI_DMA_MODE // (mixed: polling/DMA, see below)
```

Comment out DISPLAY_DIMMING_MODE, which means everything else in this section becomes irrelevant.



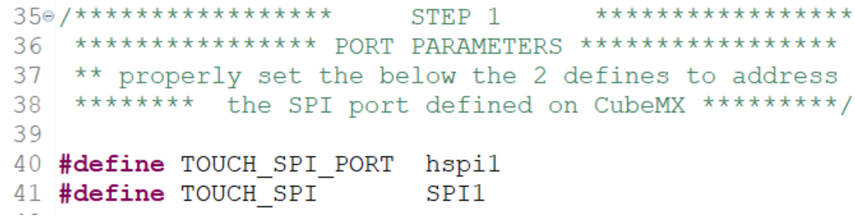
```
 *****************************************************/
//#define DISPLAY_DIMMING_MODE                          // un
#define BKLIT_TIMER                     TIM3            //tim
#define BKLIT_T                         htim3           //tim
#define BKLIT_CHANNEL                   TIM_CHANNEL_1   //cha
#define BKLIT_CCR                       CCR1            //Cap
#define BKLIT_STBY_LEVEL                5               //Dis
#define BKLIT_INIT_LEVEL                10              //Dis
```

Verify that Timer2 is designated as the 60 Hz time base for the display.



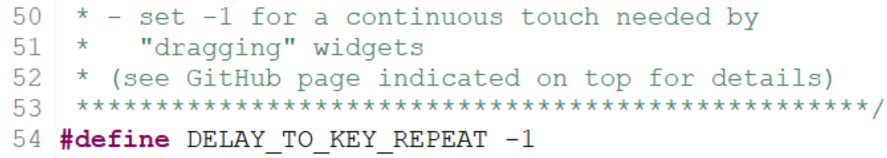
```
 * It has to be set to generate a
 * HAL_TIM_PeriodElapsedCallback 60 times per second
 * That timer has to be assigned to the below macros.
 * if not in TouchGFX-full-mode: assign macros to
 * an unused timer
 ***************************************************/
#define TGFX_TIMER          TIM2
#define TGFX_T              htim2
```

Open z_displ_XPT2046.h. Make sure the SPI port agrees with the SPI you are using, SPI1.



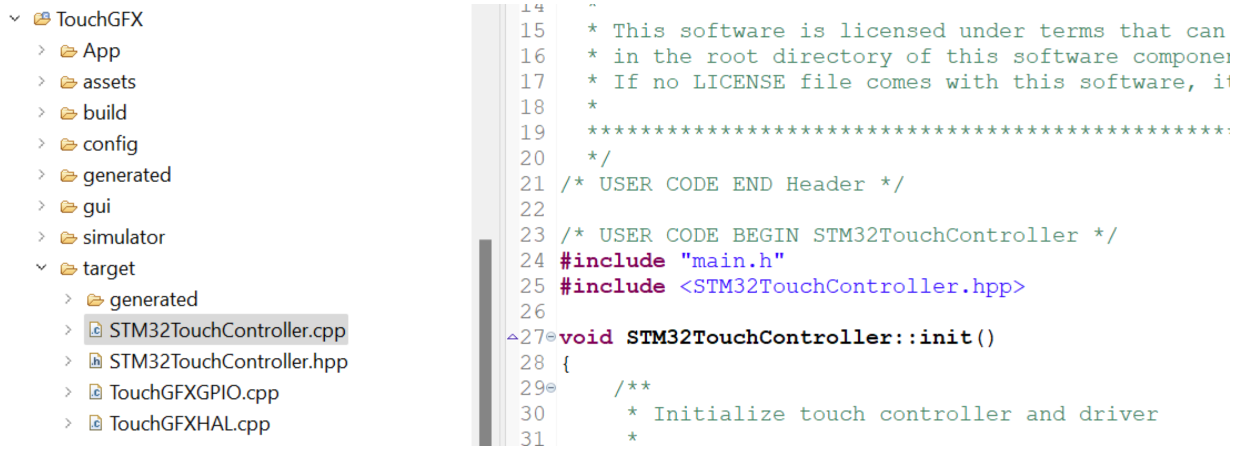
```
/*****************      STEP 1      *****************
 **************** PORT PARAMETERS *****************
 ** properly set the below the 2 defines to address
 ********  the SPI port defined on CubeMX *********/

#define TOUCH_SPI_PORT  hspi1
#define TOUCH_SPI       SPI1
```

Verify that DELAY_TO_KEY_REPEAT is set to -1.



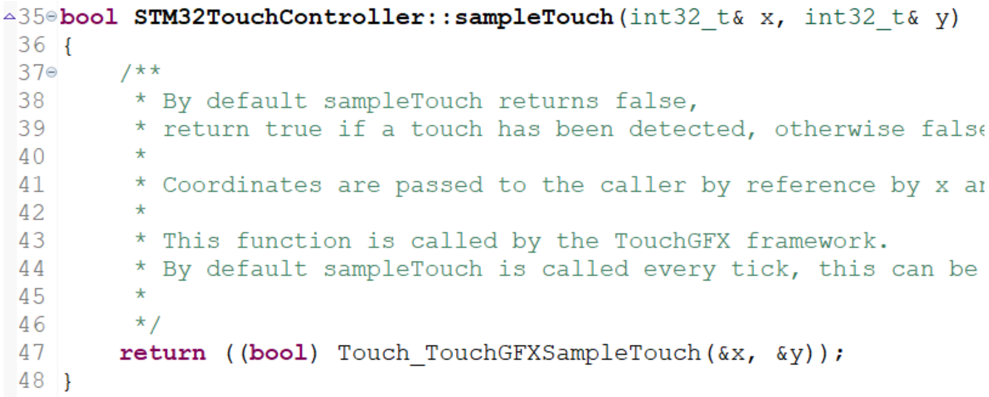
```
 * - set -1 for a continuous touch needed by
 *   "dragging" widgets
 * (see GitHub page indicated on top for details)
 ***************************************************/
#define DELAY_TO_KEY_REPEAT -1
```

Find the TouchGFX folder in your project. In the target folder, open STM32TouchController.cpp. Add #include "main.h" in the USER CODE area.



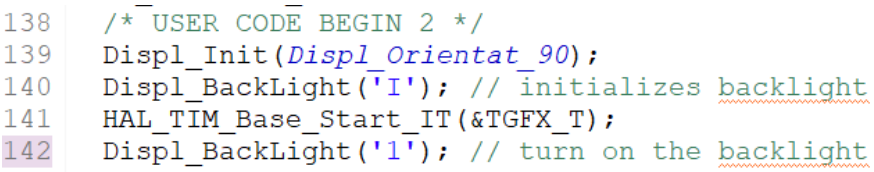
```
  * This software is licensed under terms that can
  * in the root directory of this software compone
  * If no LICENSE file comes with this software, i
  *
  ************************************************
  */
/* USER CODE END Header */

/* USER CODE BEGIN STM32TouchController */
#include "main.h"
#include <STM32TouchController.hpp>

void STM32TouchController::init()
{
    /**
     * Initialize touch controller and driver
     *
```

Also in the file `STM32TouchController.cpp`, modify the `sampleTouch` function, so that the function will return the x and y coordinates of a touch on the touchscreen.



```
bool STM32TouchController::sampleTouch(int32_t& x, int32_t& y)
{
    /**
     * By default sampleTouch returns false,
     * return true if a touch has been detected, otherwise false
     *
     * Coordinates are passed to the caller by reference by x ar
     *
     * This function is called by the TouchGFX framework.
     * By default sampleTouch is called every tick, this can be
     *
     */
    return ((bool) Touch_TouchGFXSampleTouch(&x, &y));
}
```

From `Core/Src`, open `main.c`. Add code in `USER CODE` area 2 to initialize the backlight and turn it on, as well as turn on the timer base for the display.

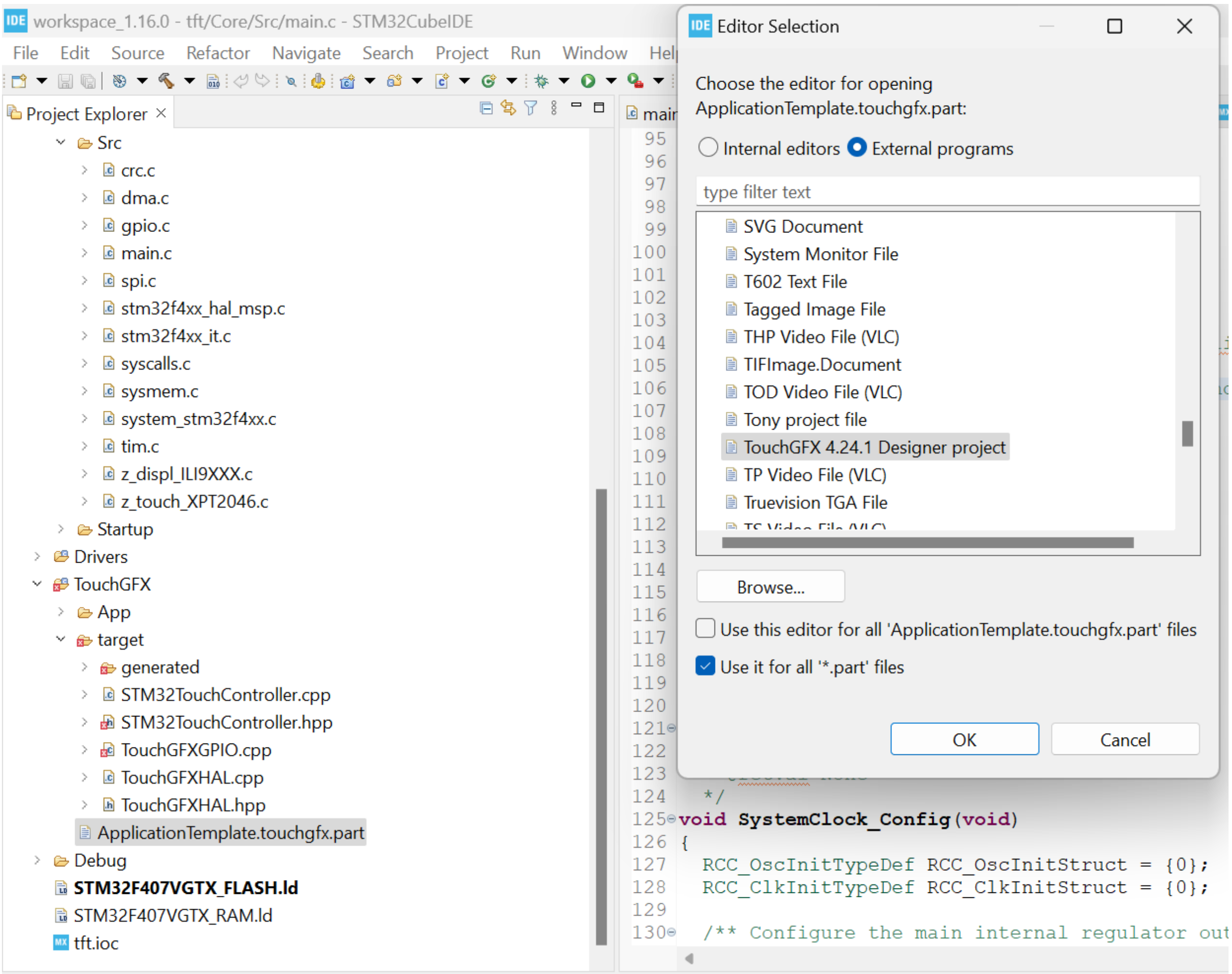


```
  /* USER CODE BEGIN 2 */
  Displ_Init(Displ_Orientat_90);
  Displ_BackLight('I'); // initializes backlight
  HAL_TIM_Base_Start_IT(&TGFX_T);
  Displ_BackLight('1'); // turn on the backlight
```

Your `while(1)` loop should already contain the line:

`MX_TouchGFX_Process();`

To make the link between your project in STM32CubeIDE and TouchGFX, right-click on your `ApplicationTemplate.touchgfx.part` file, choose Open With – Other, External programs, select TouchGFX 4.24.1 Designer Project, and use this for all `.part` files.



Choose the editor for opening ApplicationTemplate.touchgfx.part:

○ Internal editors ● External programs

☐ Use this editor for all 'ApplicationTemplate.touchgfx.part' files

☑ Use it for all '*.part' files

```
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator out
```

From now on, you only need to double-click on the `ApplicationTemplate.touchgfx.part` file in any project to go directly to TouchGFX. It may take a few moments for TouchGFX to open.

In STM32CubeIDE, click on the hammer 🔨 to compile your project. Don't be concerned about errors at this point, they should be resolved once code has been generated from TouchGFX.

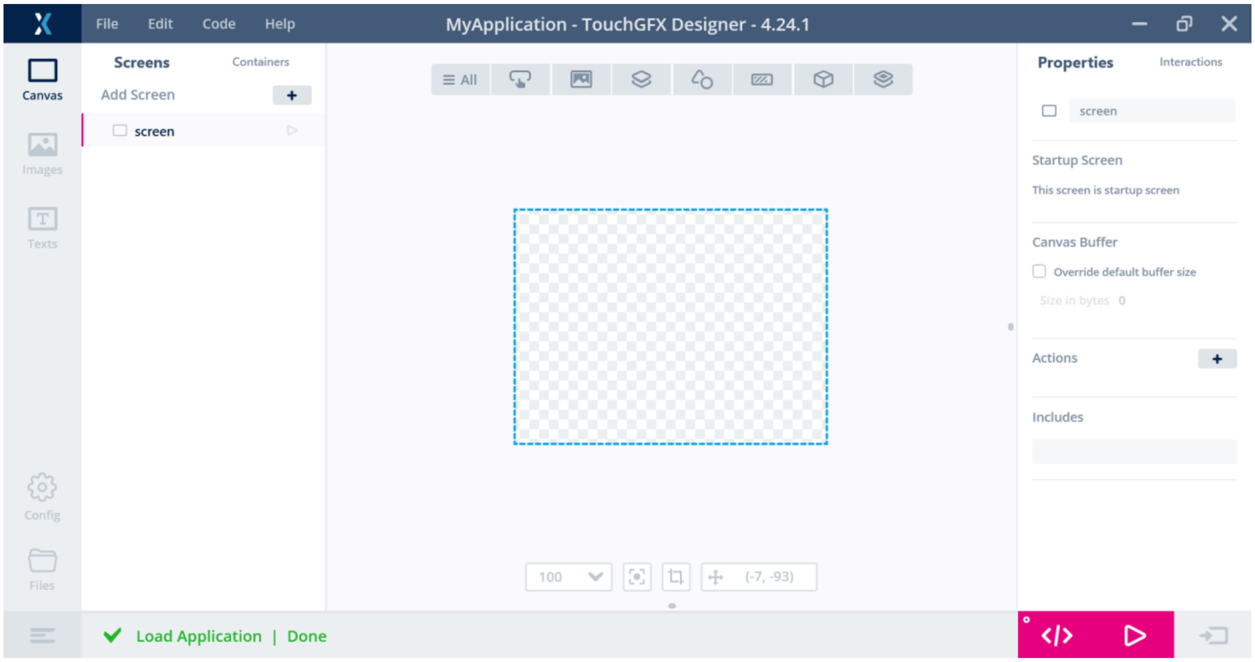
Creating a simple display

STM32 video: Designing UIs made easy with STM32 and TouchGFX

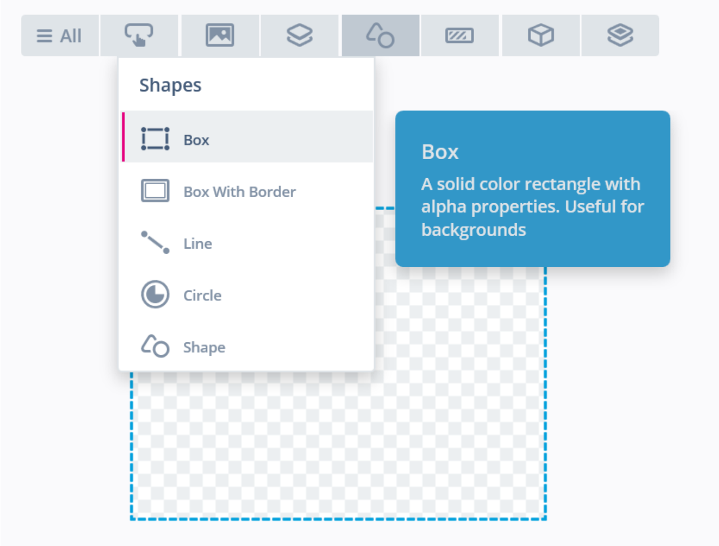
TouchGFX use cases:
https://www.st.com/content/ccc/resource/sales_and_marketing/promotional_material/brochure/group0/73/4e/dd/23/5b/fb/40/77/BRSTM32TGFX0721_web/files/BRSTM32TGFX0721_web.pdf/jcr:content/translations/en.BRSTM32TGFX0721_web.pdf

Double click on `ApplicationTemplate.touchgfx.part` if TouchGFX is not already open.

Once you close the Import GUI window you should see a canvas that is 320 pixels wide and 240 pixels high, as you specified.



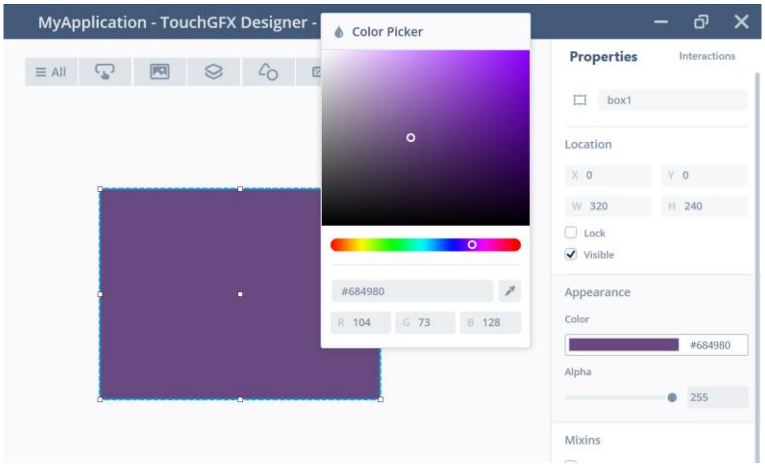
Select a box and drag it to fill the canvas, so that it can act as a background for your display.



Box

A solid color rectangle with alpha properties. Useful for backgrounds

At the right, select a colour for the box.



#684980

R 104 G 73 B 128

box1

X 0 Y 0

W 320 H 240

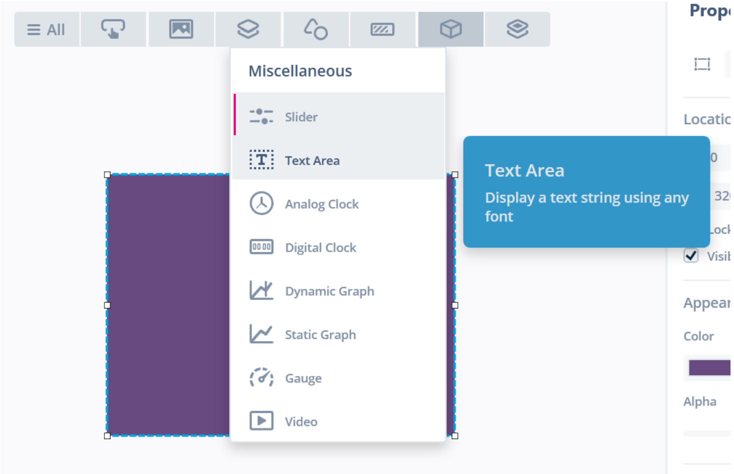
□ Lock

☑ Visible

Color #684980

Alpha 255

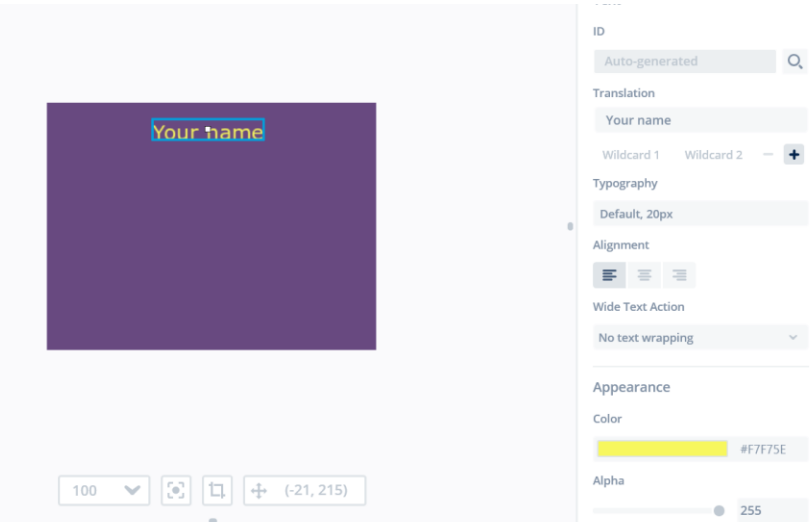
Select a Text Area and position it near the top of your canvas.



Text Area

Display a text string using any font

Type your name into the text area and select a colour for the text.



ID
Auto-generated
Translation
Your name
Wildcard 1 Wildcard 2
Typography
Default, 20px
Alignment
Wide Text Action
No text wrapping
Appearance
Color
#F7F75E
Alpha
255

To simulate your interface on your PC, click on the Run Simulator button at the bottom right.

▷

To integrate your interface with your STM32 project, click Generate Code <\>.

</>

When code generation is done, return to STM32CubeIDE.

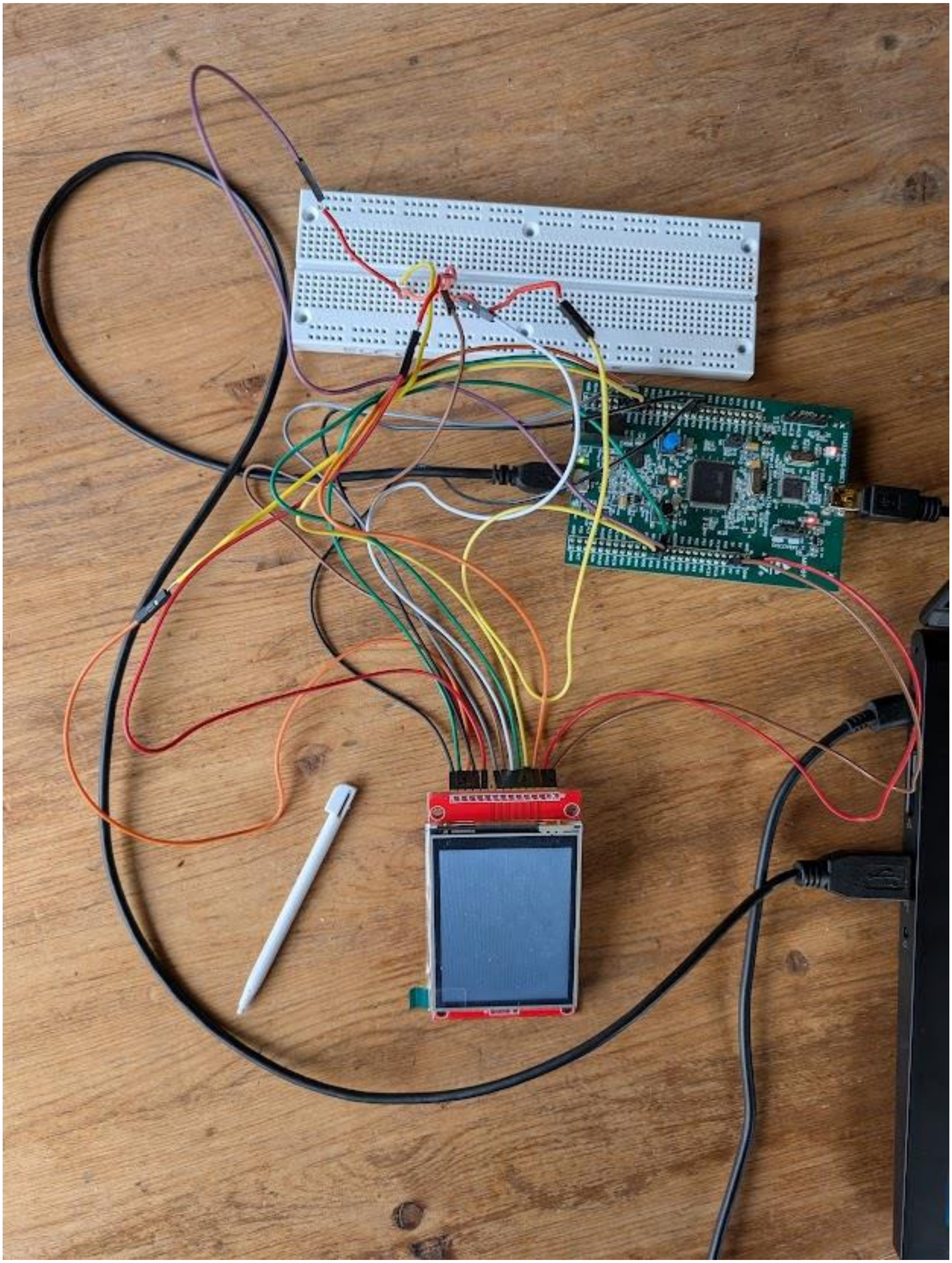
Wiring the TFT LCD display

Following the pins selected during the configuration of your project, the pins of the TFT LCD display must be connected to the pins of the STM32F407. Note that the LCD and the TFT of the display each have their own chip select, but both runs on the same clock. Both the LCD and the TFT are connected to the MOSI (master out, slave in), since both accept information from the MCU, but only the TFT is connected to the MISO (master in, slave out), since only the TFT sends information to the MCU.

Display Pin	**Display Pin Name**	**STM32F407 Pin**	**STM32F407 Pin Name**
1	VCC	3V	3V
2	GND	GND	GND
3	CS	PB15	DISPL_CS
4	RESET	PB12	DISPL_RST
5	DC/RS	PB14	DISPL_DC
6	SDI MOSI	PB5	DISPL_MOSI
7	SCK	PB3	DISPL_SCK
8	LED	PB13	DISPL_LED
9	SDO MISO	-	-
10	T_CLK	PB3	DISPL_SCK
11	T_CS	PC5	TOUCH_CS
12	T_DIN	PB5	DISPL_MOSI
13	T_DO	PB4	TOUCH_MISO
14	T_IRQ	PB1	TOUCH_INT

Use female to female jumper wires to connect the TFT LCD display to the STM32F407 Discovery board. You will need a breadboard since two pins of the display must be connected to PB3, and another two pins of the display must be connected to PB5.

Connect a mini USB cable from the Discovery board to your PC. Also connect a micro USB cable from the Discovery board to your PC.

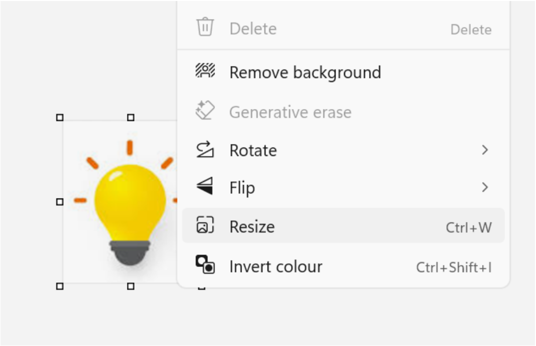


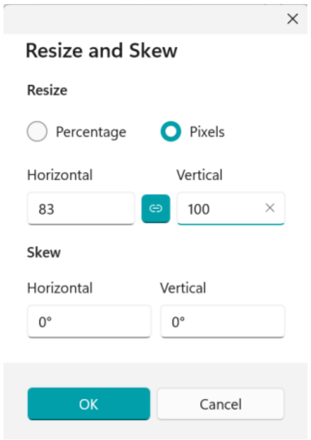
Control LCD with user button

You will modify your project so that you control images on your LCD display with the blue user button on your Discovery board.

Find images of a light bulb on and a light bulb off.

Separate into two images, ensuring the heights of the images are equal. In Microsoft Paint (or similar), right-click and Resize to about 100 vertical pixels, so that the images will be scaled well for the TFT LCD display. If the images are too large they will overflow flash memory and produce a compilation error. Your final images should have a size of just a few kB.





Resize and Skew

Resize

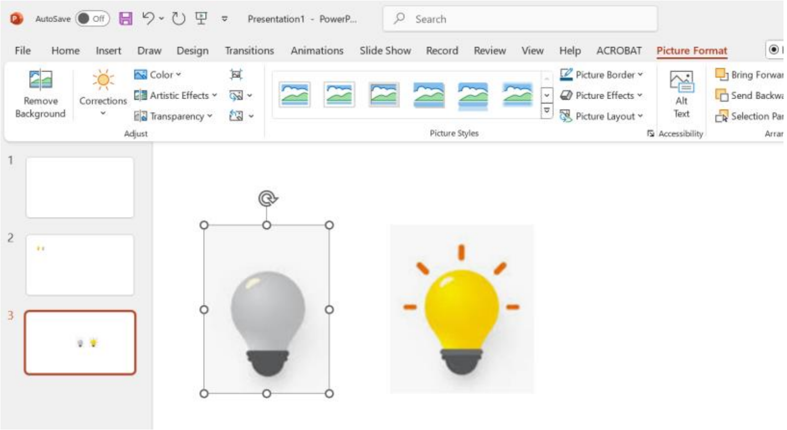
○ Percentage ● Pixels

Horizontal 83 Vertical 100

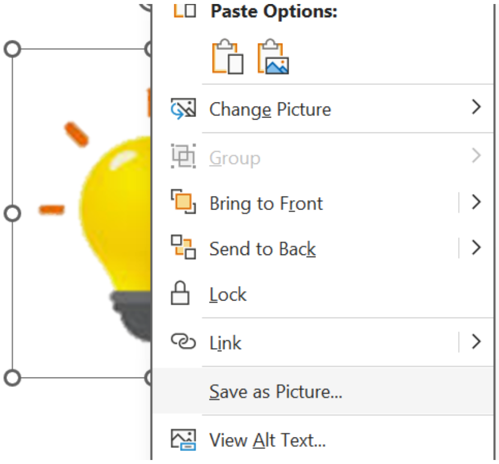
Skew

Horizontal 0° Vertical 0°

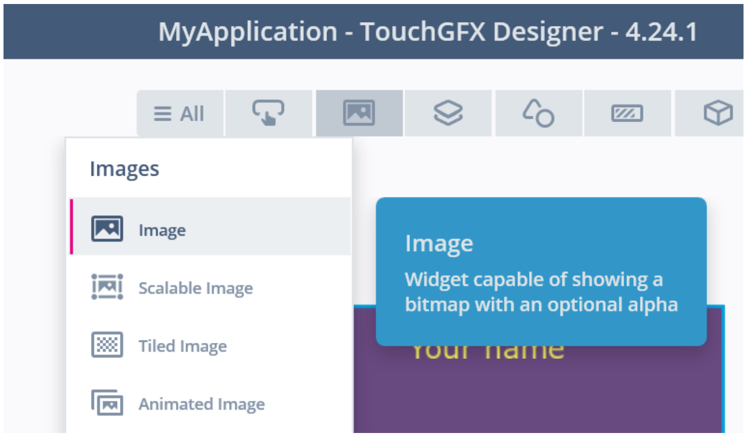
The light bulb images will look best on your display if they have transparent backgrounds. Paste your images onto a blank page in Microsoft PowerPoint. At the bottom right, increase magnification to zoom. Under Picture Format, click Remove Background. Mark Areas to Keep or Remove until you are satisfied, and then click Keep Changes.



For each image, right-click and choose Save as Picture... to save your images as light_off.png and light_on.png in your STM32CubeIDE project folder.



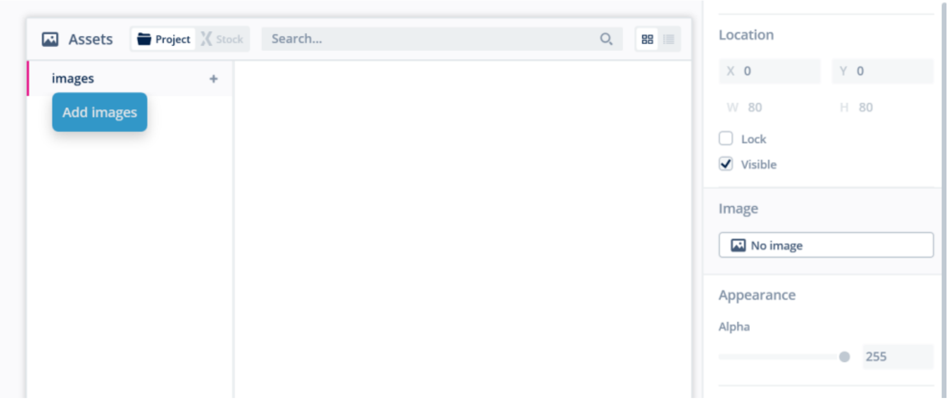
In TouchGPX, choose Image.


MyApplication - TouchGFX Designer - 4.24.1

Image

Widget capable of showing a bitmap with an optional alpha

Click on No image at the right, and then on the + sign beside images to find the off light bulb.



Location

X 0 Y 0

W 80 H 80

□ Lock

☑ Visible

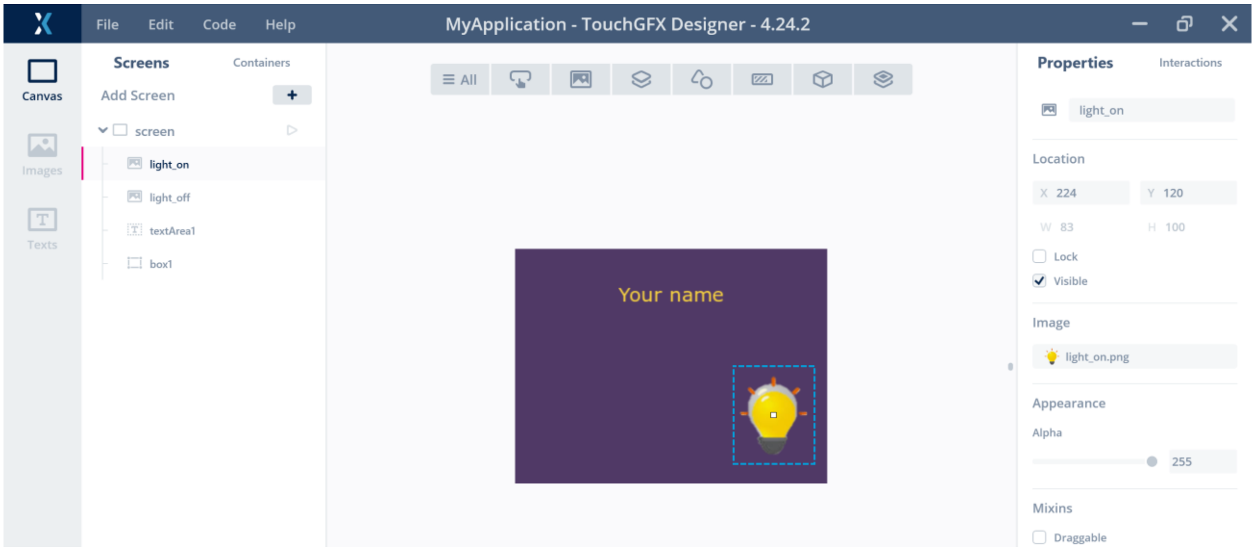
Image

No image

Appearance

Alpha 255

Place the off light bulb toward the bottom right of your canvas, and repeat with the on light bulb, so the images are exactly on top of one another. You can uncheck and recheck Visible for the on light bulb to help you align the images. Once you are happy with the alignment, uncheck the on light bulb so it is invisible. Change the names of your images (top of the properties panel) to light_on and light_off respectively; these names will be how you access this element of the display in your code.



light_on

Location

X 224 Y 120

W 83 H 100

□ Lock

☑ Visible

Image

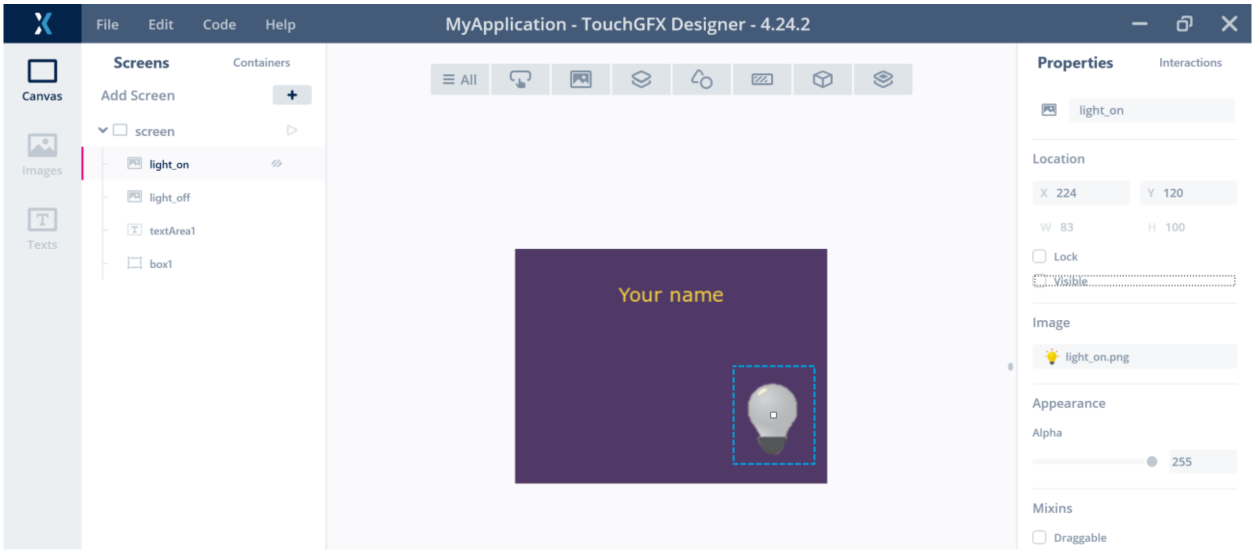
light_on.png

Appearance

Alpha 255

Mixins

□ Draggable



light_on

Location

X 224 Y 120

W 83 H 100

□ Lock

□ Visible

Image

light_on.png

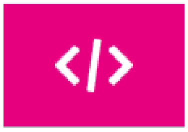
Appearance

Alpha 255

Mixins

□ Draggable

Click Generate Code <\>.



When TouchGFX generation is complete, you can return to STM32CubeIDE.

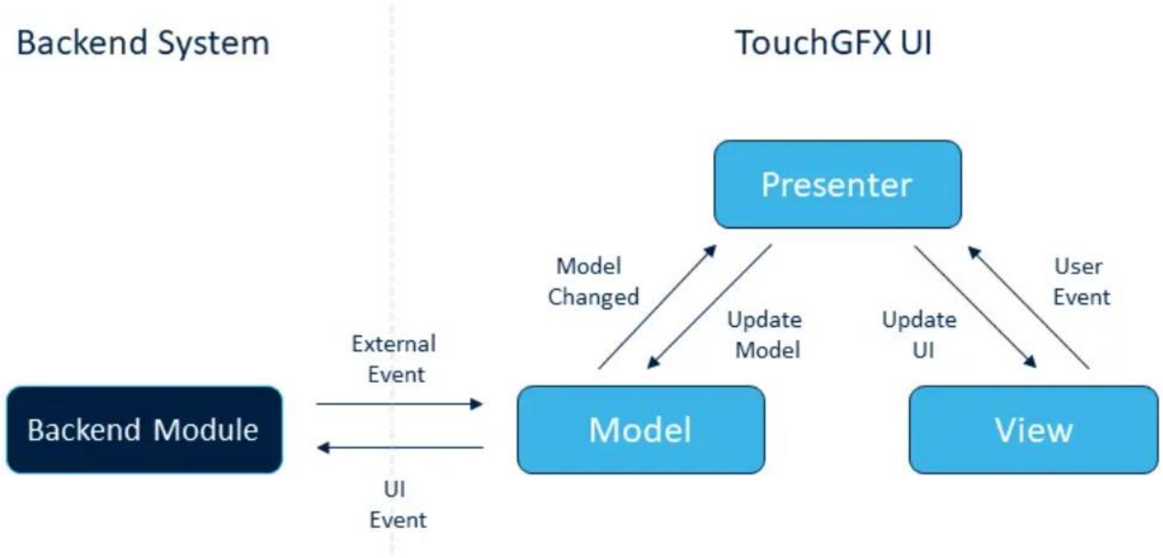
Model-View-Presenter

Model-View-Presenter (MVP) is an architecture that separates user interface concerns into three parts to make it easier to maintain, test, and reuse code.

The backend system is the non-user interface part of your project. It is a software component that receives events from the user interface (such as button clicks) and feeds events into the user interface (such as new measurements from sensors). Communication with the backend system is done from the Model.

The Presenter receives new data to be displayed from the Model via the ModelListener. The Presenter also receives user events from the View. The Presenter contains the logic of the interface, and decides what to do based on the events it receives from the Model or View. It communicates with both the Model and the View.

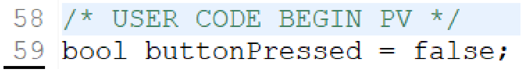
The View passively displays data and sends user events to the Presenter.



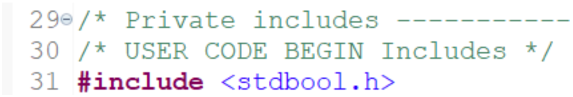
https://support.touchgfx.com/docs/development/ui-development/software-architecture/model-view-presenter-design-pattern

To implement MVP for the task of controlling the image of the light bulb with the hardware user button requires several steps:

1. In `main.c` define a global variable `buttonPressed` to the `USER CODE` private variables area. To use the `bool` type, you must add `#include <stdbool.h>` to the `USER CODE` include area.

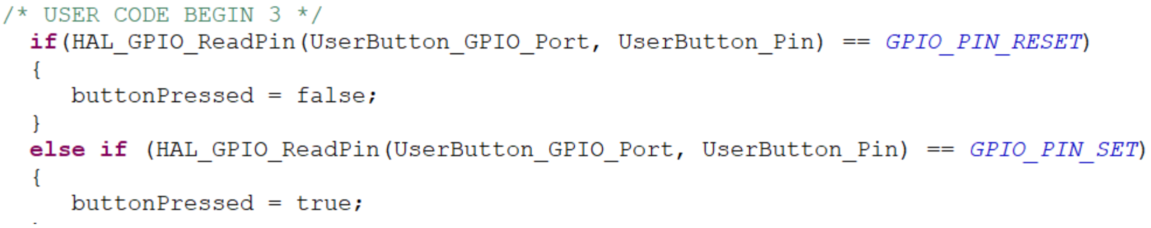


```
/* USER CODE BEGIN PV */
bool buttonPressed = false;
```



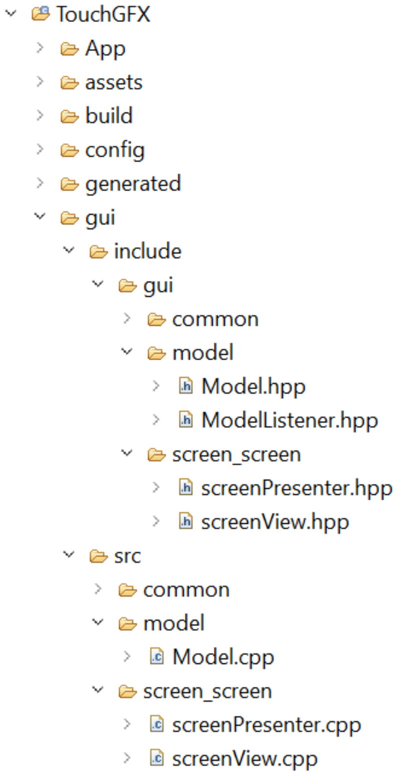
```
/* Private includes -----------
/* USER CODE BEGIN Includes */
#include <stdbool.h>
```

2. Add code to your `while(1)` loop in `main.c` to monitor the state of the blue user button. Note the code refers to the name `UserButton`, which was assigned during the configuration of your project.



```
/* USER CODE BEGIN 3 */
  if(HAL_GPIO_ReadPin(UserButton_GPIO_Port, UserButton_Pin) == GPIO_PIN_RESET)
  {
     buttonPressed = false;
  }
  else if (HAL_GPIO_ReadPin(UserButton_GPIO_Port, UserButton_Pin) == GPIO_PIN_SET)
  {
     buttonPressed = true;
```

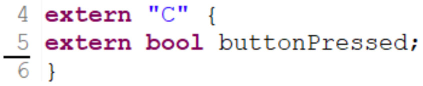
3. The files you will need to modify for MVP are all found in the `TouchGFX/gui` folder.



```
TouchGFX
  App
  assets
  build
  config
  generated
  gui
    include
      gui
        common
        model
          Model.hpp
          ModelListener.hpp
        screen_screen
          screenPresenter.hpp
          screenView.hpp
    src
      common
      model
        Model.cpp
      screen_screen
        screenPresenter.cpp
        screenView.cpp
```

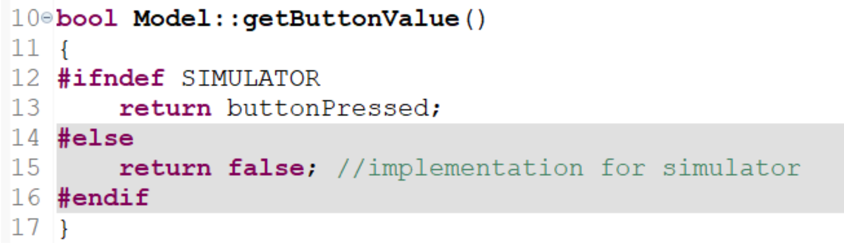
4. In `Model.cpp`:

   Declare `buttonPressed` as an `extern` variable:



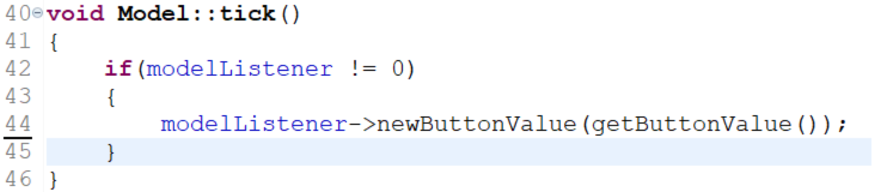
```
extern "C" {
extern bool buttonPressed;
}
```

   Create a getButtonValue function that returns the value of buttonPressed:



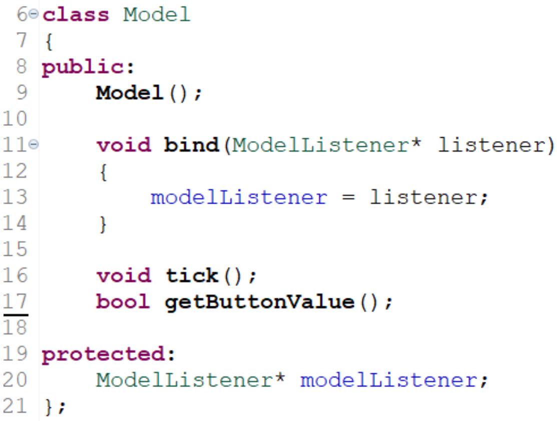
```
bool Model::getButtonValue()
{
#ifndef SIMULATOR
    return buttonPressed;
#else
    return false; //implementation for simulator
#endif
}
```

Add code to call `getButtonValue` from Model Listener’s function `newButtonValue`:



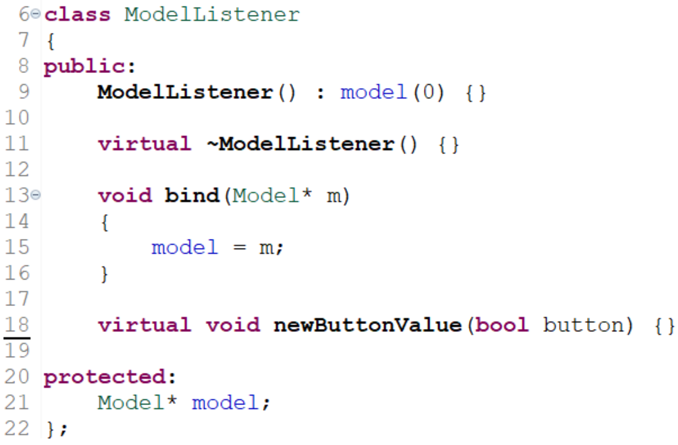
```
void Model::tick()
{
    if(modelListener != 0)
    {
        modelListener->newButtonValue(getButtonValue());
    }
}
```

5. In `Model.hpp`, declare a prototype for the `getButtonValue` function.



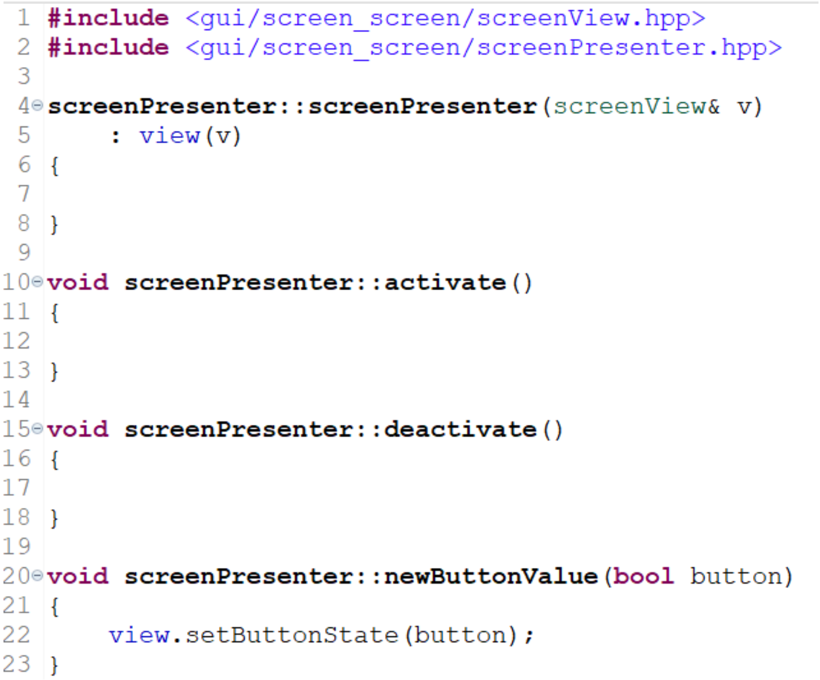
```
class Model
{
public:
    Model();

    void bind(ModelListener* listener)
    {
        modelListener = listener;
    }

    void tick();
    bool getButtonValue();

protected:
    ModelListener* modelListener;
};
```

6. In `ModelListener.hpp`, define an empty virtual `newButtonValue` function. In C++, the base class (`ModelListener` in this case) defines virtual functions that can be overridden in the derived classes (`ScreenPresenter` in this case).



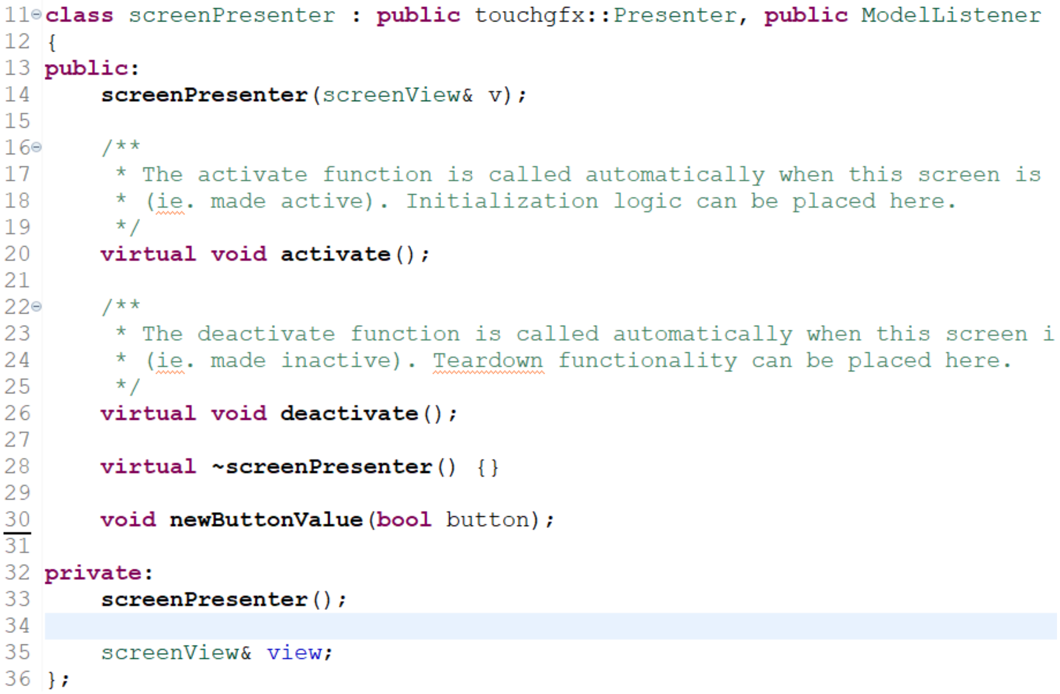
```
class ModelListener
{
public:
    ModelListener() : model(0) {}

    virtual ~ModelListener() {}

    void bind(Model* m)
    {
        model = m;
    }

    virtual void newButtonValue(bool button) {}

protected:
    Model* model;
};
```

7. In `screenPresenter.cpp`, write a `newButtonValue` function that will override the empty one in ModelListener. Note that this function calls a `setButtonState` function that will be defined in View.



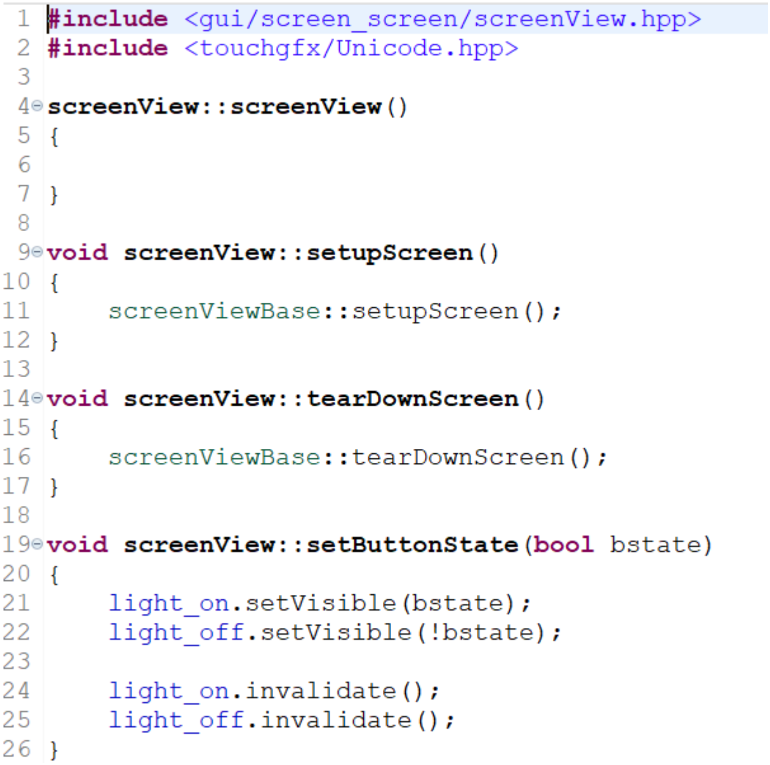
```
#include <gui/screen_screen/screenView.hpp>
#include <gui/screen_screen/screenPresenter.hpp>

screenPresenter::screenPresenter(screenView& v)
    : view(v)
{

}

void screenPresenter::activate()
{

}

void screenPresenter::deactivate()
{

}

void screenPresenter::newButtonValue(bool button)
{
    view.setButtonState(button);
}
```

8. In `screenPresenter.hpp`, write a prototype for the `newButtonValue` function.



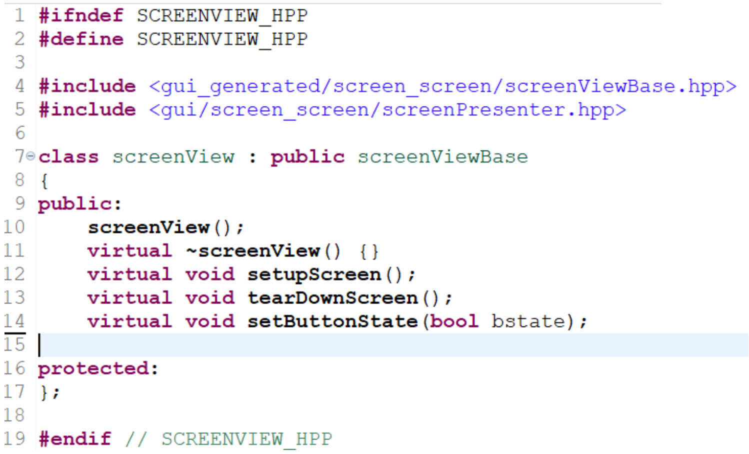
```
class screenPresenter : public touchgfx::Presenter, public ModelListener
{
public:
    screenPresenter(screenView& v);

    /**
     * The activate function is called automatically when this screen is
     * (ie. made active). Initialization logic can be placed here.
     */
    virtual void activate();

    /**
     * The deactivate function is called automatically when this screen i
     * (ie. made inactive). Teardown functionality can be placed here.
     */
    virtual void deactivate();

    virtual ~screenPresenter() {}

    void newButtonValue(bool button);

private:
    screenPresenter();

    screenView& view;
};
```

9. In `screenView.cpp`, write the `setButtonState` function, which changes what is seen on the LCD screen according to information sent from Presenter. The `invalidate` commands tell the LCD the current display is invalid and force it to update.



```
#include <gui/screen_screen/screenView.hpp>
#include <touchgfx/Unicode.hpp>

screenView::screenView()
{

}

void screenView::setupScreen()
{
    screenViewBase::setupScreen();
}

void screenView::tearDownScreen()
{
    screenViewBase::tearDownScreen();
}

void screenView::setButtonState(bool bstate)
{
    light_on.setVisible(bstate);
    light_off.setVisible(!bstate);

    light_on.invalidate();
    light_off.invalidate();
}
```

10. In `screenView.hpp`, add a prototype for the `setButtonState` function.



```
#ifndef SCREENVIEW_HPP
#define SCREENVIEW_HPP

#include <gui_generated/screen_screen/screenViewBase.hpp>
#include <gui/screen_screen/screenPresenter.hpp>

class screenView : public screenViewBase
{
public:
    screenView();
    virtual ~screenView() {}
    virtual void setupScreen();
    virtual void tearDownScreen();
    virtual void setButtonState(bool bstate);

protected:
};

#endif // SCREENVIEW_HPP
```

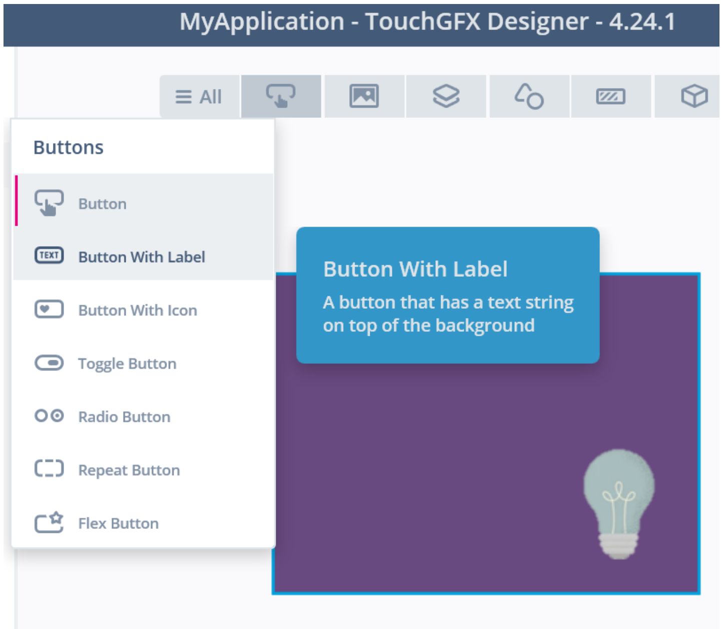
Build your project.

Run – Debug. Once your project has successfully downloaded, click Play/Resume ▶. When you press the blue user button on your Discovery board, the light bulb on your LCD display should change from off to on.

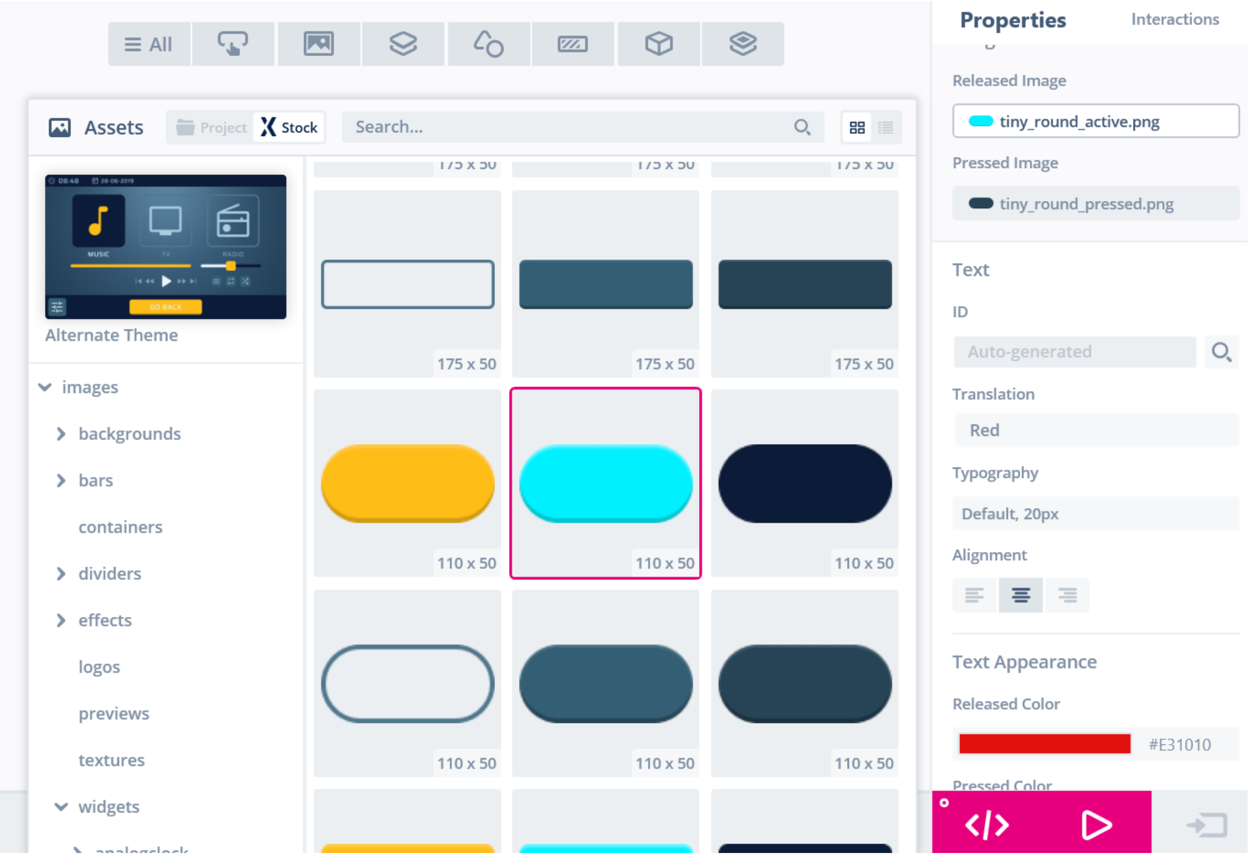
Control LEDs with TFT touchscreen

You will modify your project so that buttons on your touchscreen control LEDs on your Discovery board.

In TouchGFX, add a Button With Label. Name it redButton.



Select a reasonably small button image for the released and pressed versions of the button (which must have the same shape). Make “Red” the text for your button, and change the colour of the released text to red.



Released Image: tiny_round_active.png

Pressed Image: tiny_round_pressed.png

Text

ID: Auto-generated

Translation: Red

Typography: Default, 20px

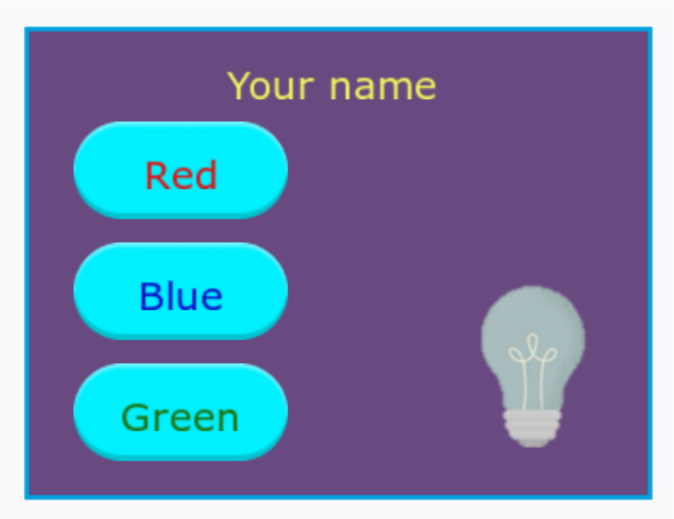
Alignment

Text Appearance

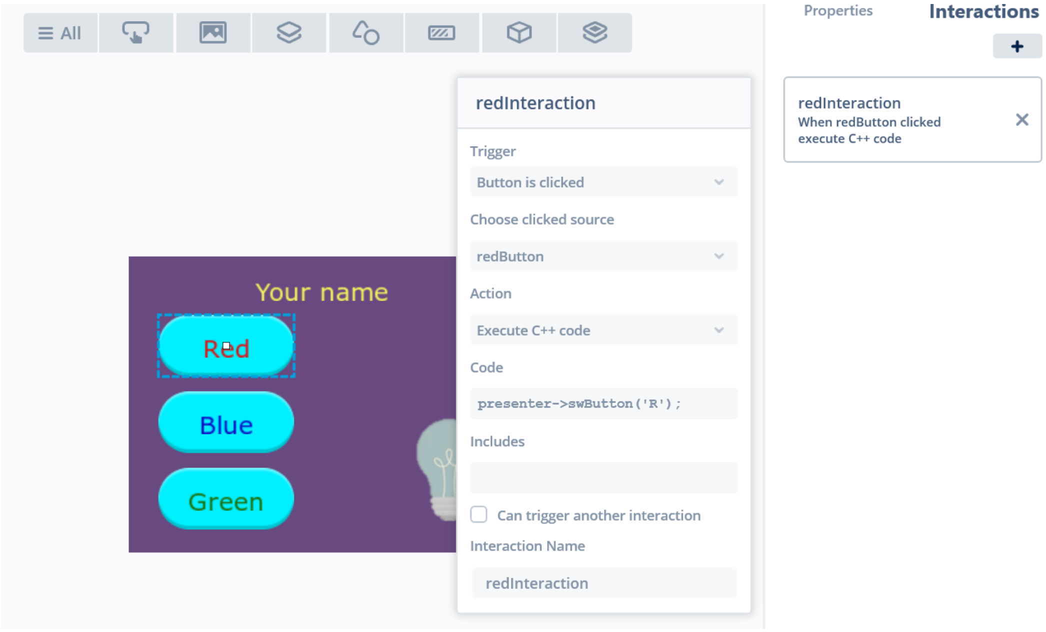
Released Color: #E31010

Pressed Color

Repeat for blueButton and greenButton buttons. When you are finished your display should have three buttons.



You will create interactions for each button. Click on Interactions at the top right and click on the + sign.



redInteraction

Trigger: Button is clicked

Choose clicked source: redButton

Action: Execute C++ code

Code: `presenter->swButton('R');`

Includes:

□ Can trigger another interaction

Interaction Name: redInteraction

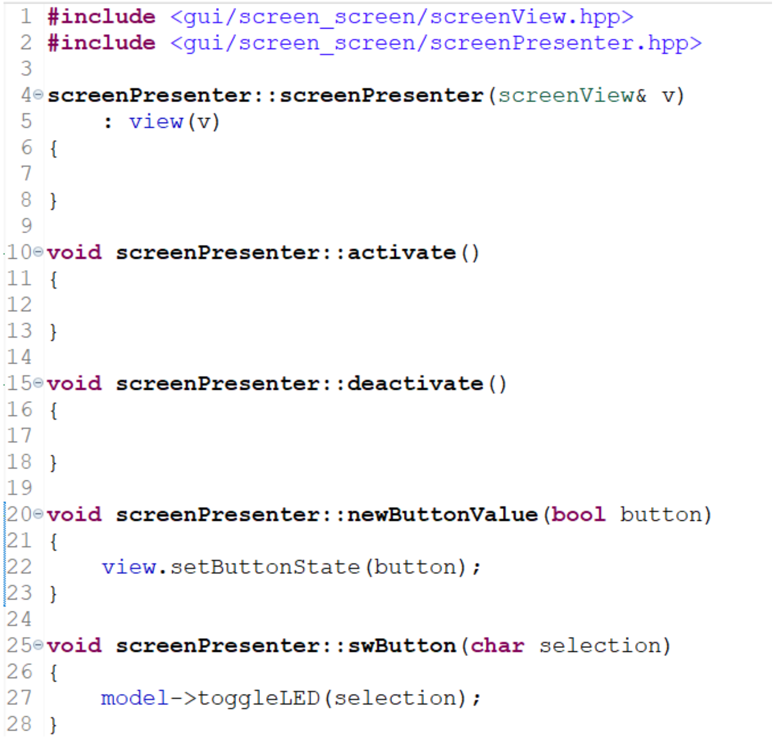
redInteraction
When redButton clicked
execute C++ code

Repeat for blueButton and greenButton, but in your code send 'B' and 'G' characters.

Generate code in TouchGFX. When done, return to STM32CubeIDE.

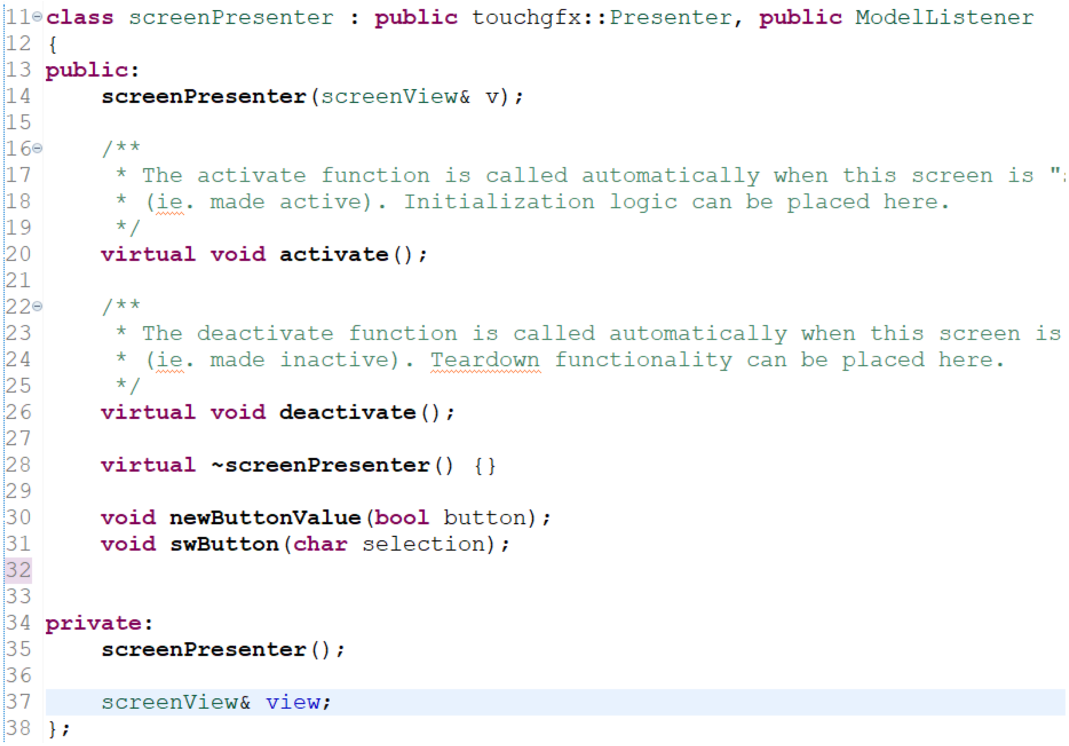
Several changes are needed to implement the button interactions:

1. In `screenPresenter.cpp`, implement the `swButton` function used by the button interactions in your interface. The selection argument will indicate which button was pressed, information that is provided by View to Presenter and by Presenter to Model. The `swButton` function will call a `toggleLED` function to toggle the appropriate LED.



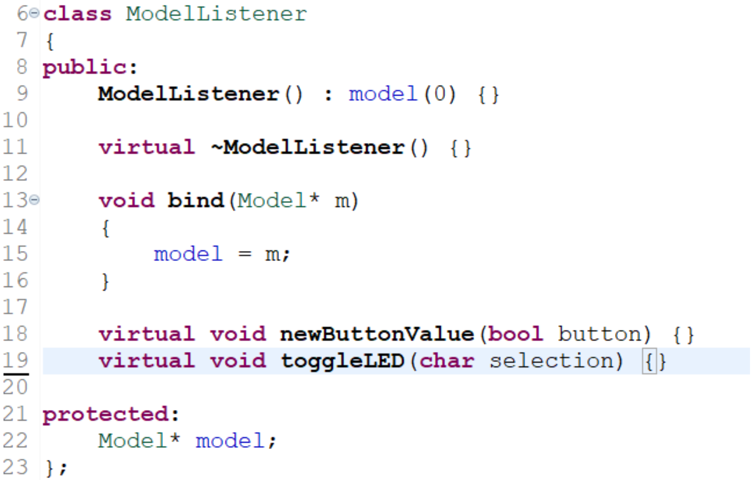
```
#include <gui/screen_screen/screenView.hpp>
#include <gui/screen_screen/screenPresenter.hpp>

screenPresenter::screenPresenter(screenView& v)
    : view(v)
{

}

void screenPresenter::activate()
{

}

void screenPresenter::deactivate()
{

}

void screenPresenter::newButtonValue(bool button)
{
    view.setButtonState(button);
}

void screenPresenter::swButton(char selection)
{
    model->toggleLED(selection);
}
```

2. In `screenPresenter.hpp`, make a prototype for the `swButton` function.



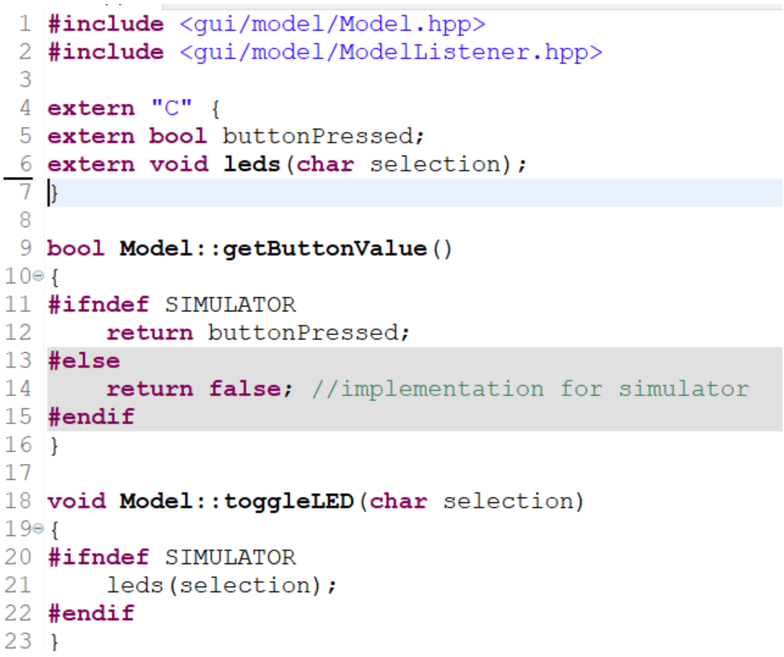
```
class screenPresenter : public touchgfx::Presenter, public ModelListener
{
public:
    screenPresenter(screenView& v);

    /**
     * The activate function is called automatically when this screen is "
     * (ie. made active). Initialization logic can be placed here.
     */
    virtual void activate();

    /**
     * The deactivate function is called automatically when this screen is
     * (ie. made inactive). Teardown functionality can be placed here.
     */
    virtual void deactivate();

    virtual ~screenPresenter() {}

    void newButtonValue(bool button);
    void swButton(char selection);


private:
    screenPresenter();

    screenView& view;
};
```

3. In `modelListener.hpp`, define an empty virtual `toggleLED` function.



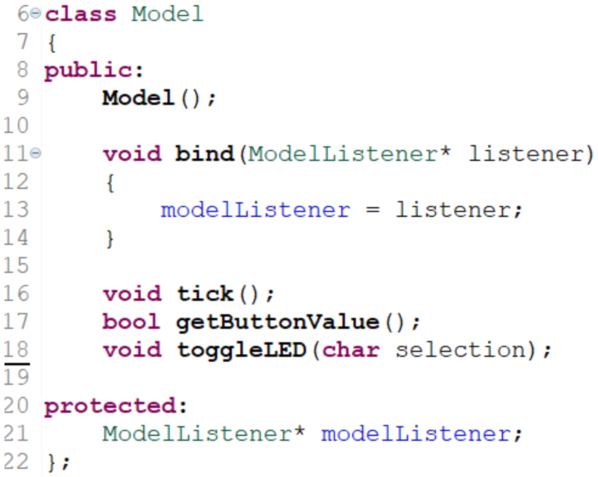
```
class ModelListener
{
public:
    ModelListener() : model(0) {}

    virtual ~ModelListener() {}

    void bind(Model* m)
    {
        model = m;
    }

    virtual void newButtonValue(bool button) {}
    virtual void toggleLED(char selection) {}

protected:
    Model* model;
};
```

4. In `Model.cpp`, implement the `toggleLED` function. The toggleLED function calls the function `leds`, which will be implemented in `main.c`. Also in `Model.cpp`, make an `extern` declaration for the `leds` function.



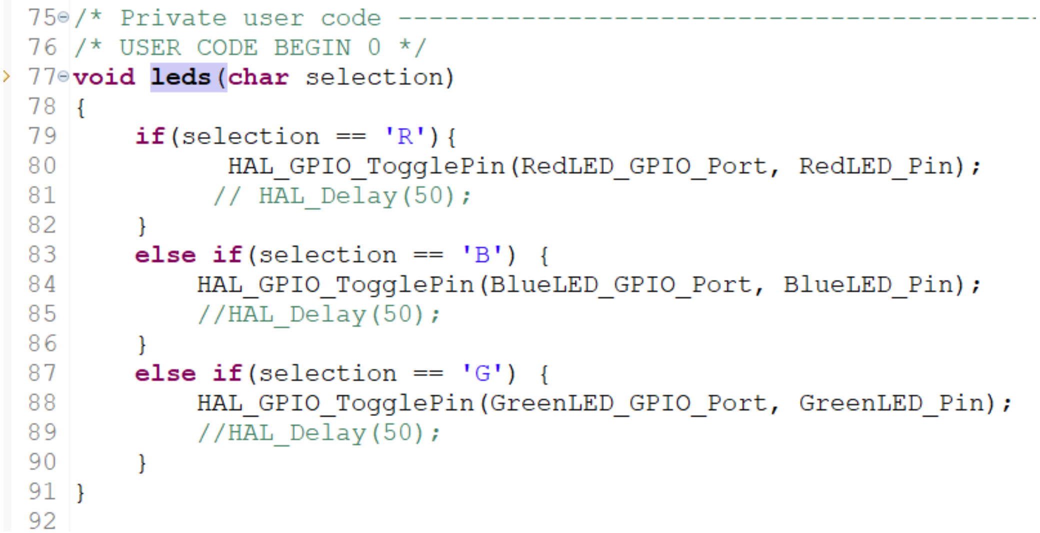
```
#include <gui/model/Model.hpp>
#include <gui/model/ModelListener.hpp>

extern "C" {
extern bool buttonPressed;
extern void leds(char selection);
}

bool Model::getButtonValue()
{
#ifndef SIMULATOR
    return buttonPressed;
#else
    return false; //implementation for simulator
#endif
}

void Model::toggleLED(char selection)
{
#ifndef SIMULATOR
    leds(selection);
#endif
}
```

5. In `Model.hpp`, provide a prototype for the `toggleLED` function.



```
class Model
{
public:
    Model();

    void bind(ModelListener* listener)
    {
        modelListener = listener;
    }

    void tick();
    bool getButtonValue();
    void toggleLED(char selection);

protected:
    ModelListener* modelListener;
};
```

6. From Core/Src, open main.c. In USER CODE area 0, define the leds function. Note that the function uses LED names assigned during the configuration of your project.



```
/* Private user code -----------------------------------------
/* USER CODE BEGIN 0 */
void leds(char selection)
{
    if(selection == 'R'){
          HAL_GPIO_TogglePin(RedLED_GPIO_Port, RedLED_Pin);
         // HAL_Delay(50);
    }
    else if(selection == 'B') {
        HAL_GPIO_TogglePin(BlueLED_GPIO_Port, BlueLED_Pin);
        //HAL_Delay(50);
    }
    else if(selection == 'G') {
        HAL_GPIO_TogglePin(GreenLED_GPIO_Port, GreenLED_Pin);
        //HAL_Delay(50);
    }
}

```

Compile your project.

Run – Debug. Play/resume.

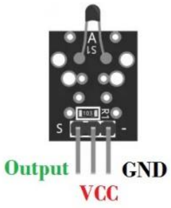
When you tap with the stylus on one of the buttons on your touchscreen, the matching LED on the Discovery board should toggle.

Troubleshooting note: Tapping a button on the screen should make its image change from the “released” version of the button to the “pressed” version of the button. If this fails to happen for any of your buttons, you will need to calibrate your touchscreen. The touch sensor is a transparent layer that is glued on top of the LCD display, so multiple orientations are possible. The touch on the display must be matched to the correct graphic position. Refer to Appendix 1 for calibration instructions.

Display temperature from thermistor

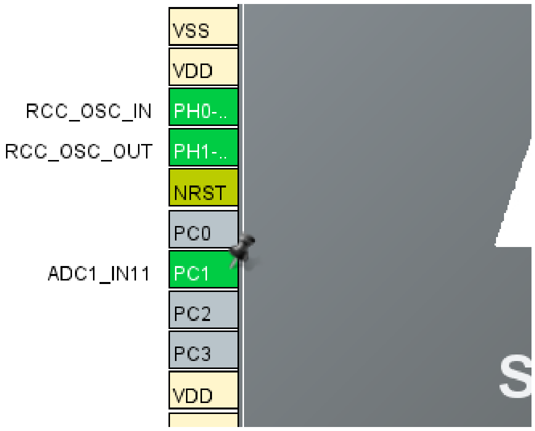
You will modify your project so that a thermistor’s temperature is displayed on your LCD screen.

The thermistor used in this project is a KY-013 thermistor. This module incorporates a 10 kΩ resistor in series with a thermistor. Pin 1 is connected to an ADC input pin on the Discovery board, pin 2 is connected to 3 V, and pin 3 is connected to ground.

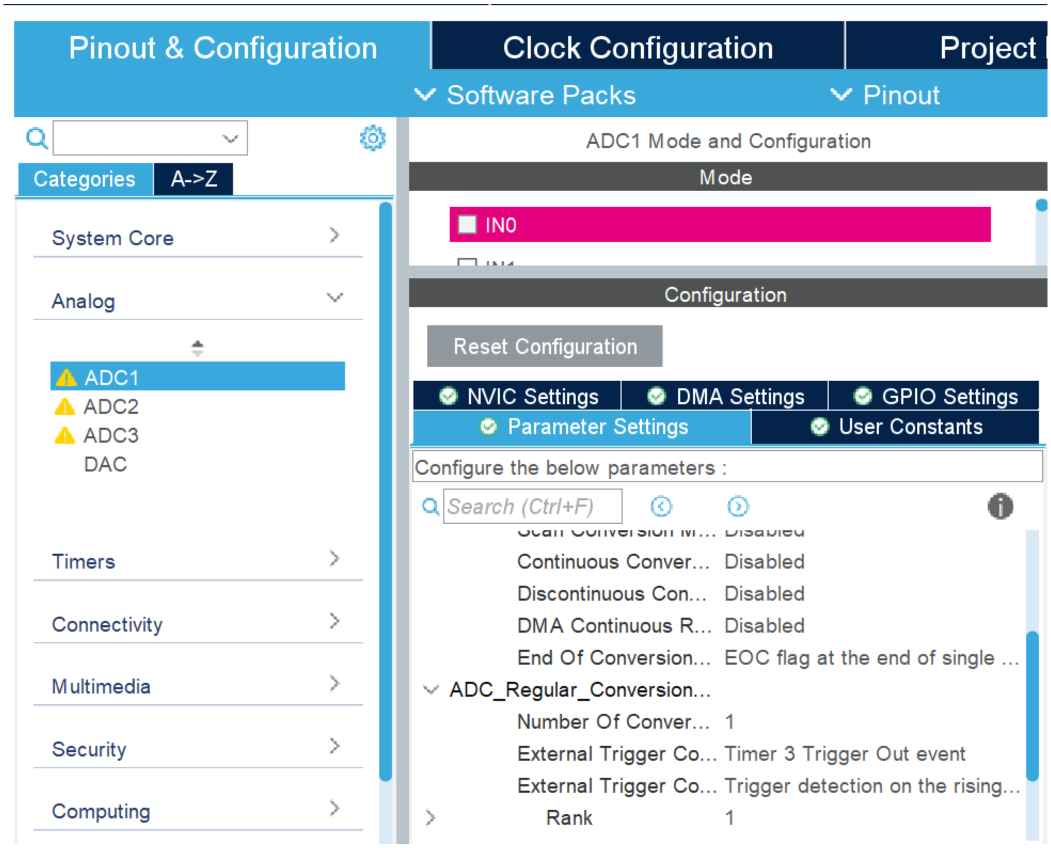


In STM32CubeIDE, open `tft_lcd.ioc`.

Select pin PC1 to be your ADC input. (Many pins can serve as ADC input.)



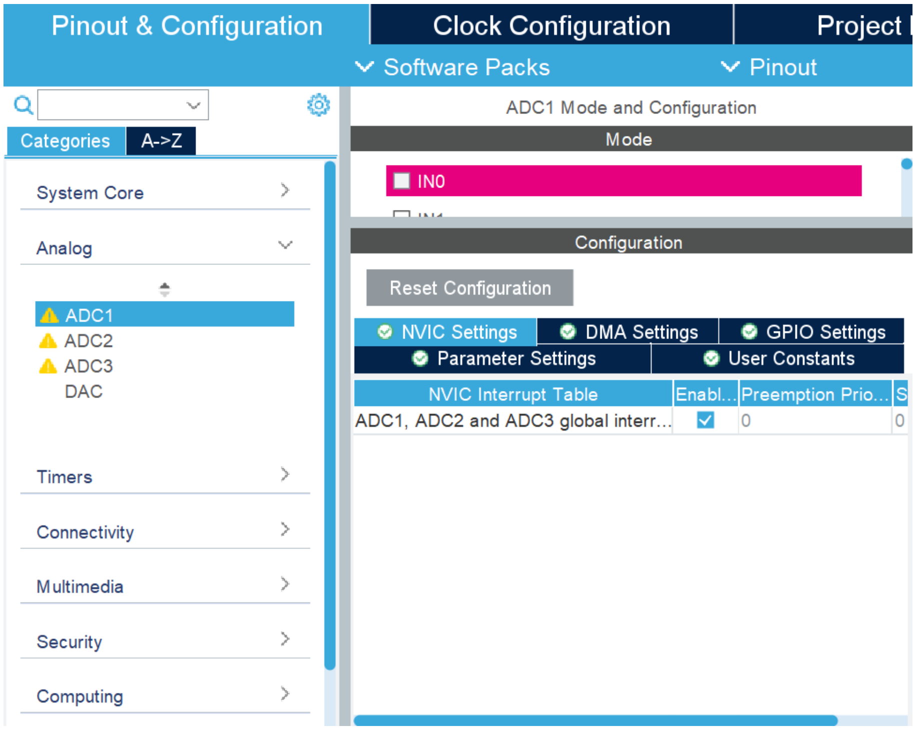
Your ADC interrupts will be triggered by a timer every 1 sec. Set your ADC parameters to trigger on Timer 3.



Configure the below parameters :

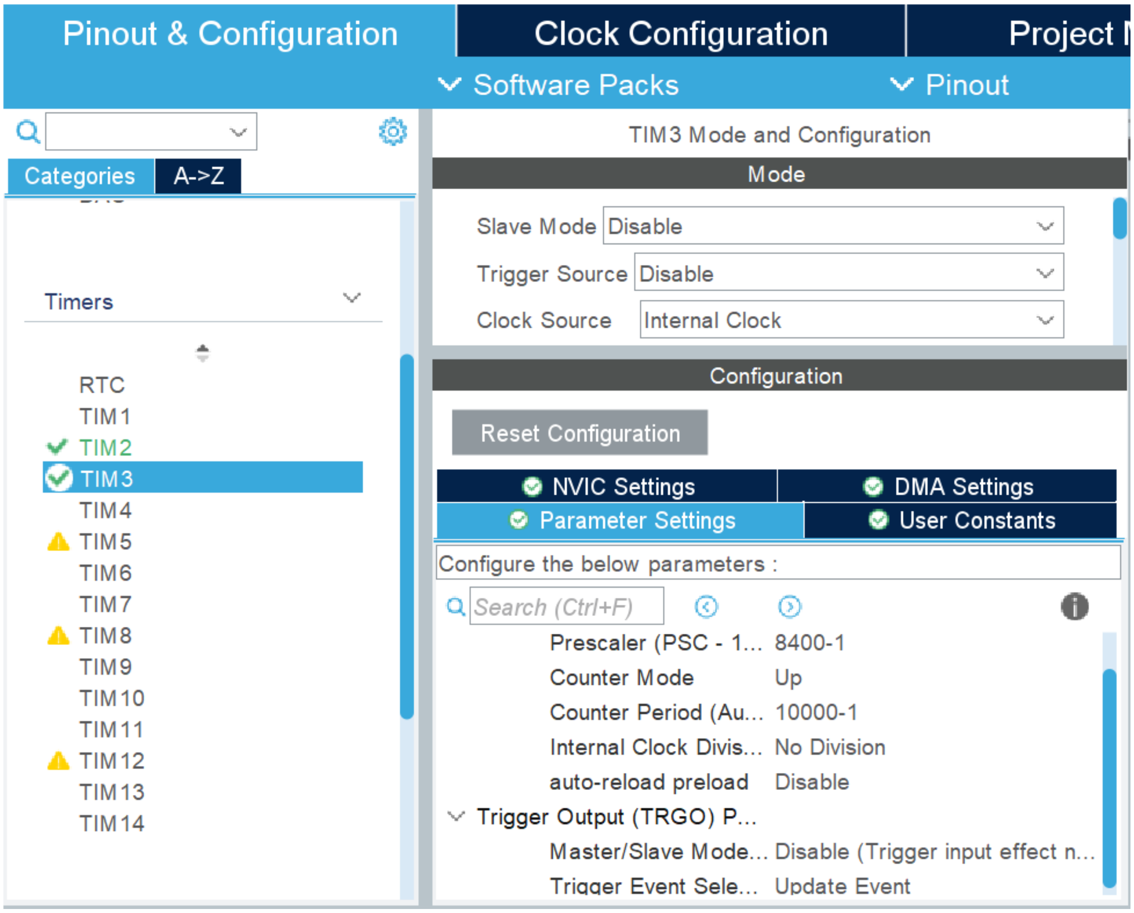
```
Scan Conversion M...      Disabled
Continuous Conver...      Disabled
Discontinuous Con...      Disabled
DMA Continuous R...       Disabled
End Of Conversion...      EOC flag at the end of single ...
ADC_Regular_Conversion...
Number Of Conver...       1
External Trigger Co...    Timer 3 Trigger Out event
External Trigger Co...    Trigger detection on the rising...
Rank                      1
```

Enable interrupts for your ADC.



NVIC Interrupt Table	Enabl...	Preemption Prio...	S
ADC1, ADC2 and ADC3 global interr...	☑	0	0

Set TIM3 to use Internal Clock as its source. Change Trigger Event Selection to Update Event. In order to achieve 1 Hz interrupts (when the internal clock is 168 MHz and the APB2 Peripheral clock is 84 MHz), set Prescaler to 8399 and Counter Period to 9999. Generate code.



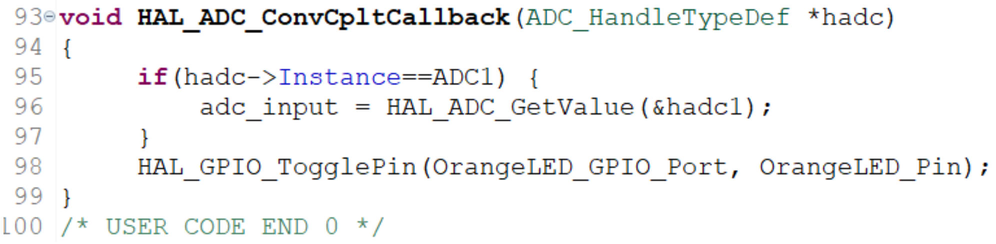
TIM3 Mode and Configuration

Slave Mode Disable
Trigger Source Disable
Clock Source Internal Clock

Configure the below parameters :

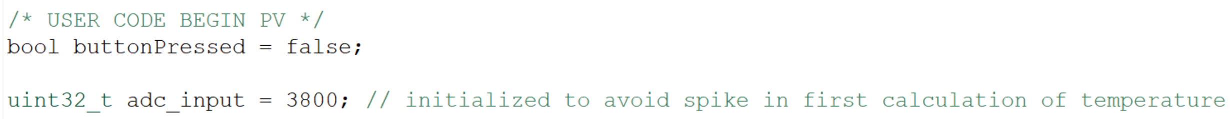
```
Prescaler (PSC - 1...      8400-1
Counter Mode               Up
Counter Period (Au...      10000-1
Internal Clock Divis...    No Division
auto-reload preload        Disable
Trigger Output (TRGO) P...
    Master/Slave Mode...   Disable (Trigger input effect n...
    Trigger Event Sele...  Update Event
```

In USER CODE area 0, add the function HAL_ADC_ConvCpltCallback, which is called on every ADC interrupt. At each interrupt, a value is read into adc_input and the orange LED is toggled, which should occur once per second.



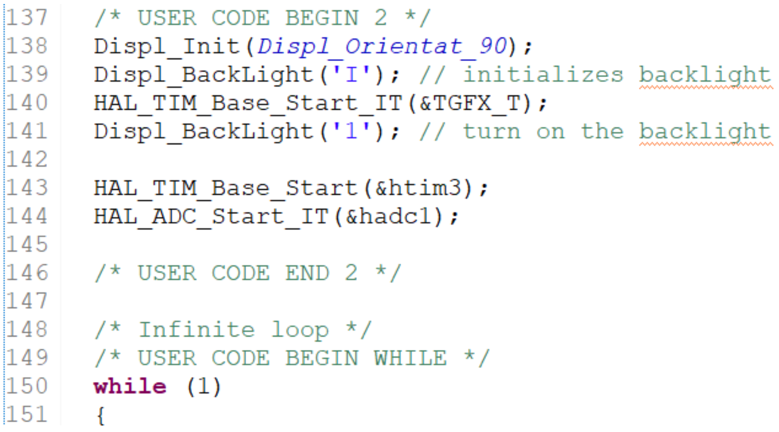
```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
{
    if(hadc->Instance==ADC1) {
        adc_input = HAL_ADC_GetValue(&hadc1);
    }
    HAL_GPIO_TogglePin(OrangeLED_GPIO_Port, OrangeLED_Pin);
}
/* USER CODE END 0 */
```

Add a declaration for adc_input to the USER CODE private variables area.



```
/* USER CODE BEGIN PV */
bool buttonPressed = false;

uint32_t adc_input = 3800; // initialized to avoid spike in first calculation of temperature
```

Add two lines in USER CODE area 2 to start the ADC timer (Timer 3) and also the ADC itself.



```
/* USER CODE BEGIN 2 */
Displ_Init(Displ_Orientat_90);
Displ_BackLight('I'); // initializes backlight
HAL_TIM_Base_Start_IT(&TGFX_T);
Displ_BackLight('1'); // turn on the backlight

HAL_TIM_Base_Start(&htim3);
HAL_ADC_Start_IT(&hadc1);

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
```

Inside `while(1)`, add code that computes thermistor resistance from an ADC value, and computes a Celsius temperature from the thermistor resistance. The resistance of the thermistor R_T can be found by solving the equation:

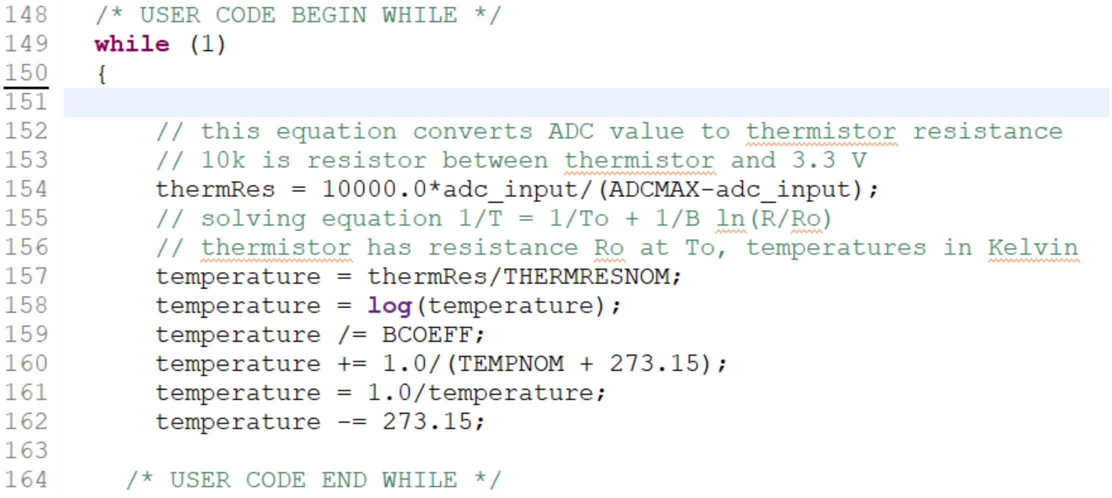
$$\frac{\mathrm{ADC}}{4095} = \frac{\mathrm{R_T}}{\mathrm{R_T} + \mathrm{R_o}}$$

where ADC is the integer reported by the ADC, 4095 is the maximum value returned from a 12-bit ADC, $\mathrm{R_T}$ is the resistance of the thermistor and $\mathrm{R_o} = 10\ \mathrm{k\Omega}$ is the value of the series resistor.

With the thermistor resistance, the temperature T in K may be computed with the Steinhart equation:

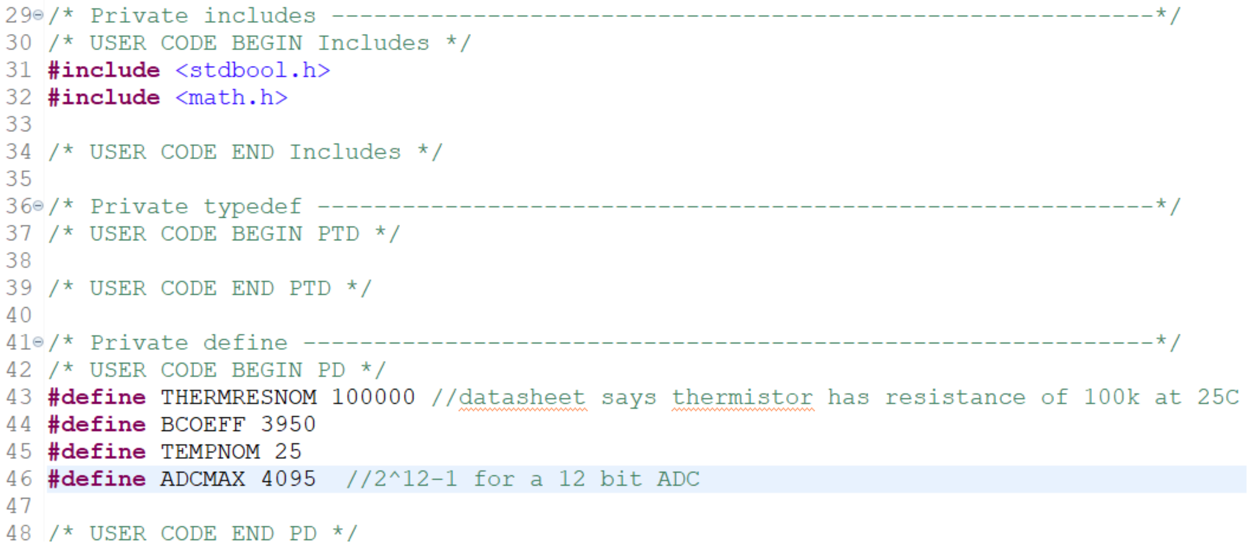
$$\frac{1}{\mathrm{T}} = \frac{1}{\mathrm{T_o}} + \frac{1}{\mathrm{B}} \ln\left(\frac{\mathrm{R_T}}{\mathrm{R_o}}\right)$$

where T_o is room temperature 298 K and B is the thermistor coefficient with a value of 3950. The K temperature can be converted to Celsius by subtracting 273.15.

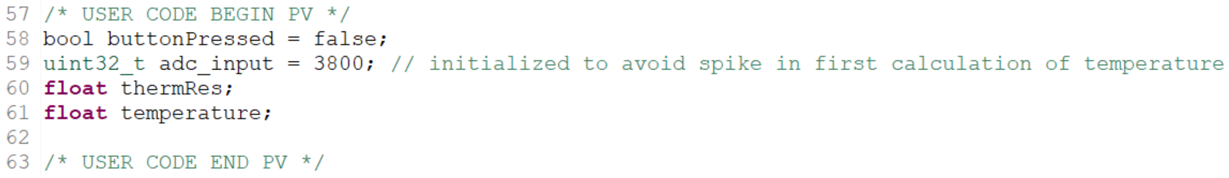


```
/* USER CODE BEGIN WHILE */
while (1)
{

    // this equation converts ADC value to thermistor resistance
    // 10k is resistor between thermistor and 3.3 V
    thermRes = 10000.0*adc_input/(ADCMAX-adc_input);
    // solving equation 1/T = 1/To + 1/B ln(R/Ro)
    // thermistor has resistance Ro at To, temperatures in Kelvin
    temperature = thermRes/THERMRESNOM;
    temperature = log(temperature);
    temperature /= BCOEFF;
    temperature += 1.0/(TEMPNOM + 273.15);
    temperature = 1.0/temperature;
    temperature -= 273.15;

  /* USER CODE END WHILE */
```

This code requires `#include <math.h>`, as well as the definition of constants and the declaration of variables:

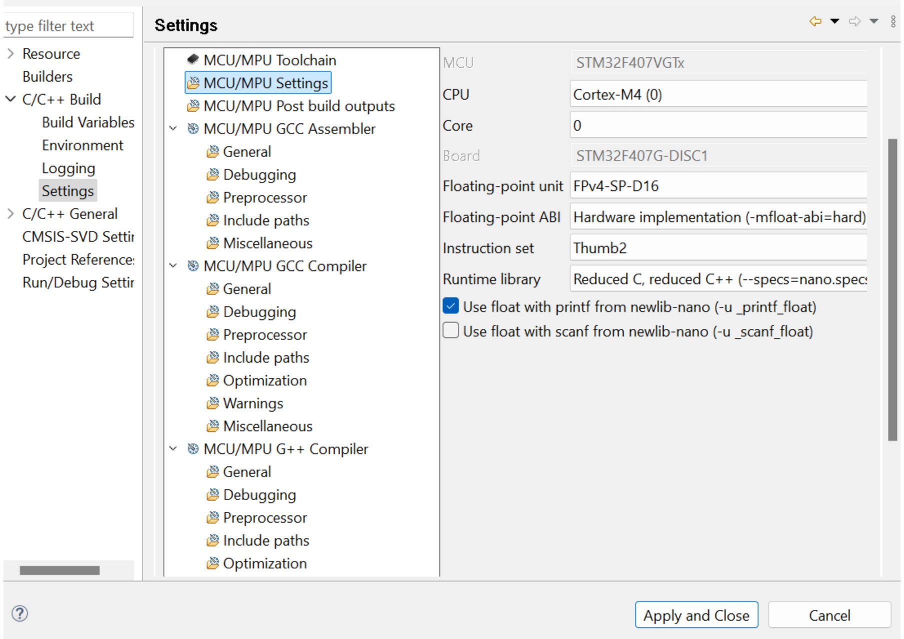


```
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdbool.h>
#include <math.h>

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define THERMRESNOM 100000 //datasheet says thermistor has resistance of 100k at 25C
#define BCOEFF 3950
#define TEMPNOM 25
#define ADCMAX 4095  //2^12-1 for a 12 bit ADC

/* USER CODE END PD */
```



```
/* USER CODE BEGIN PV */
bool buttonPressed = false;
uint32_t adc_input = 3800; // initialized to avoid spike in first calculation of temperature
float thermRes;
float temperature;

/* USER CODE END PV */
```

To enable printing of float values, right-click on your project name and click Properties. In C/C++ Build Settings – MCU/MPU Settings, check off “Use float with printf…”



MCU	STM32F407VGTx
CPU	Cortex-M4 (0)
Core	0
Board	STM32F407G-DISC1
Floating-point unit	FPv4-SP-D16
Floating-point ABI	Hardware implementation (-mfloat-abi=hard)
Instruction set	Thumb2
Runtime library	Reduced C, reduced C++ (--specs=nano.specs

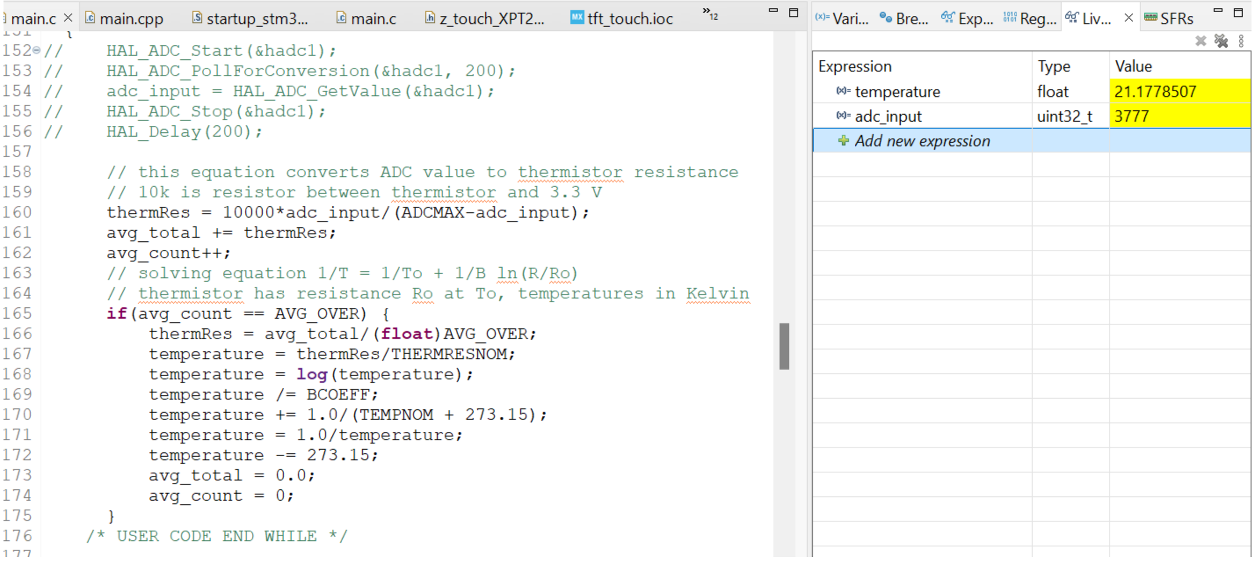
☑ Use float with printf from newlib-nano (-u _printf_float)

☐ Use float with scanf from newlib-nano (-u _scanf_float)

Add a KY-013 thermistor to your breadboard. Connect pin 1 to pin PC1, which is the ADC pin you configured. Connect pin 2 to 3 V and connect pin 3 to GND.

Compile your project. Run – Debug. Play/resume.

While your project is running, open the Live Expressions tab at the top right. Enter any variable you would like to monitor. Try putting your finger on the thermistor and verify that the temperature increases.

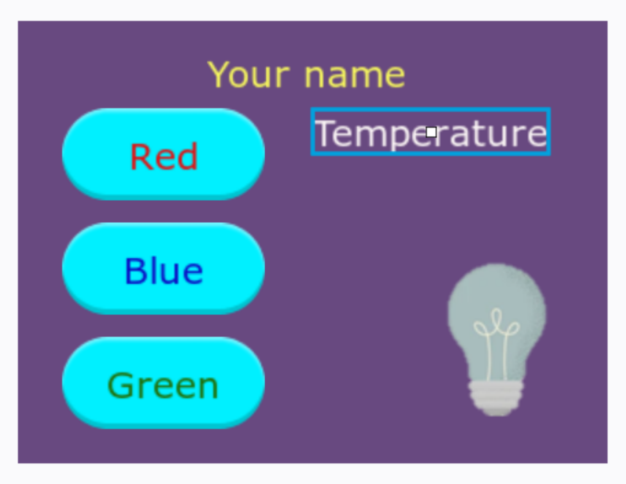


```
//    HAL_ADC_Start(&hadc1);
//    HAL_ADC_PollForConversion(&hadc1, 200);
//    adc_input = HAL_ADC_GetValue(&hadc1);
//    HAL_ADC_Stop(&hadc1);
//    HAL_Delay(200);

      // this equation converts ADC value to thermistor resistance
      // 10k is resistor between thermistor and 3.3 V
      thermRes = 10000*adc_input/(ADCMAX-adc_input);
      avg_total += thermRes;
      avg_count++;
      // solving equation 1/T = 1/To + 1/B ln(R/Ro)
      // thermistor has resistance Ro at To, temperatures in Kelvin
      if(avg_count == AVG_OVER) {
          thermRes = avg_total/(float)AVG_OVER;
          temperature = thermRes/THERMRESNOM;
          temperature = log(temperature);
          temperature /= BCOEFF;
          temperature += 1.0/(TEMPNOM + 273.15);
          temperature = 1.0/temperature;
          temperature -= 273.15;
          avg_total = 0.0;
          avg_count = 0;
      }
    /* USER CODE END WHILE */
```

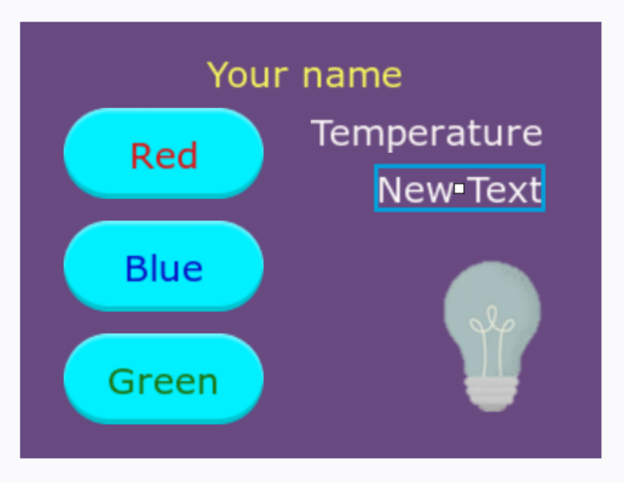
Expression	Type	Value
temperature	float	21.1778507
adc_input	uint32_t	3777
Add new expression		

In your project's TouchGFX folder, double-click on `ApplicationTemplate.touchgfx.part` to open TouchGFX (if it is not already open).

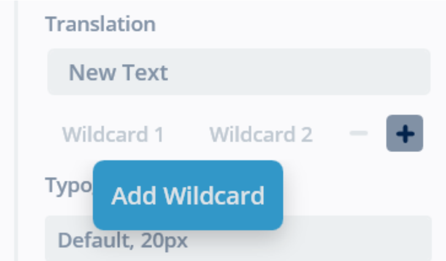
Add a Text Area to your interface and enter "Temperature."



Add a second Text Area.



At the right, click on the + sign below the text. A wildcard accepts values for display, whether numerical or character. Set up an initial value for the wildcard, as well as a buffer that will hold the characters that display the temperature value. Remove "New Text," and add a "C" after <value> for Celsius.



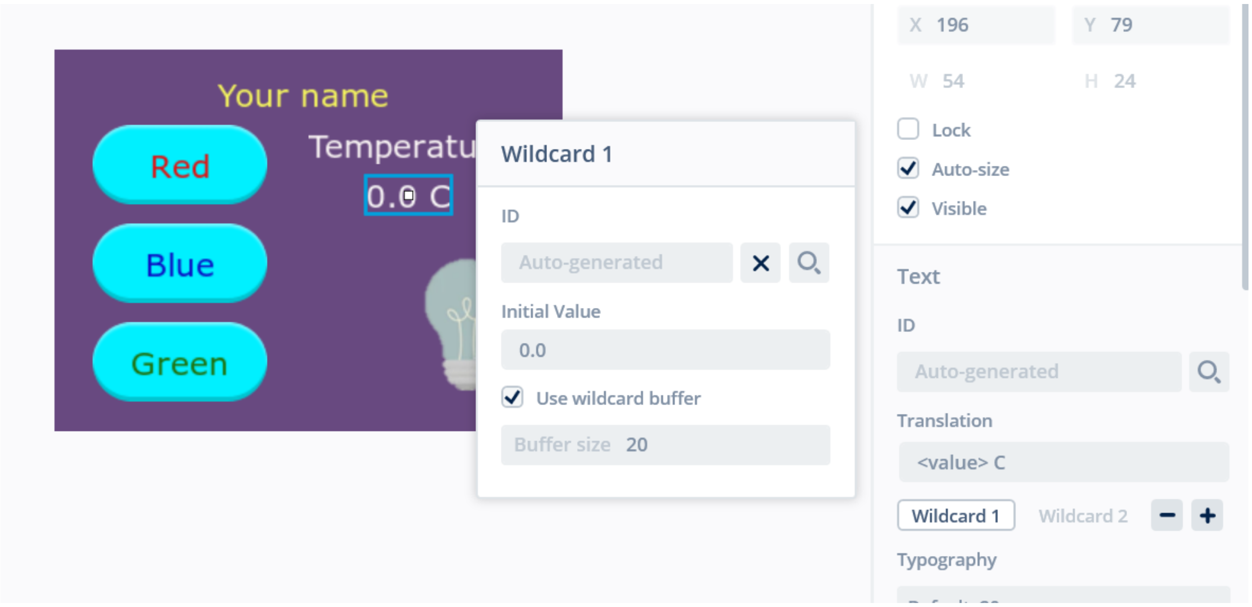
Translation

New Text

Wildcard 1 Wildcard 2

Typo

Default, 20px



Wildcard 1

ID: Auto-generated

Initial Value: 0.0

☑ Use wildcard buffer

Buffer size 20

X 196 Y 79

W 54 H 24

☐ Lock

☑ Auto-size

☑ Visible

Text

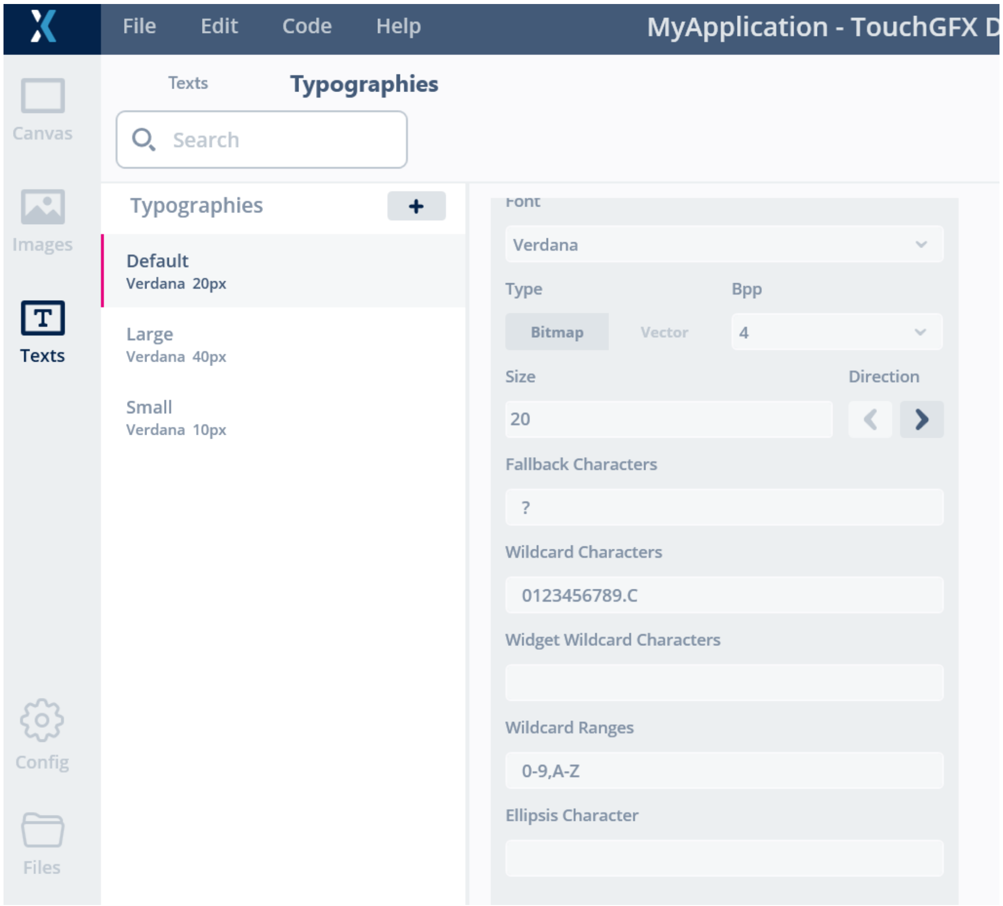
ID: Auto-generated

Translation: <value> C

Wildcard 1 Wildcard 2

Typography

It is necessary to specify the characters that are allowed in your wildcard buffer, as well as the legal ranges. At the left, click on Texts and go to the Typographies tab to set this up.



Default Verdana 20px

Large Verdana 40px

Small Verdana 10px

Font: Verdana

Type: Bitmap Vector

Bpp: 4

Size: 20

Direction

Fallback Characters: ?

Wildcard Characters: 0123456789.C

Widget Wildcard Characters:

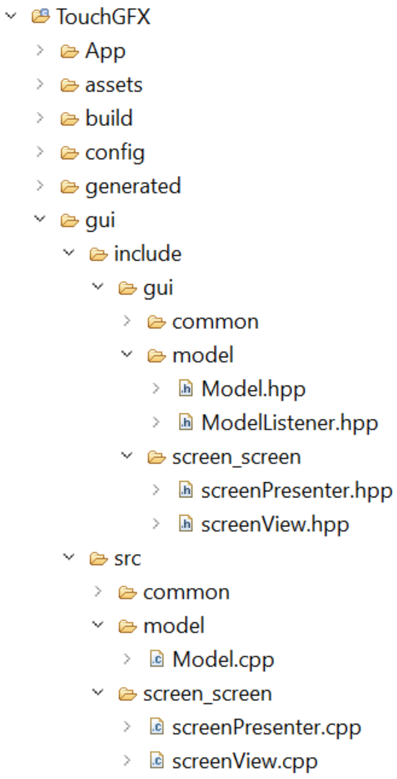
Wildcard Ranges: 0-9,A-Z

Ellipsis Character:

Generate TouchGFX code. When it is finished, return to STM32CubeIDE.

Modifications are needed for MVP:

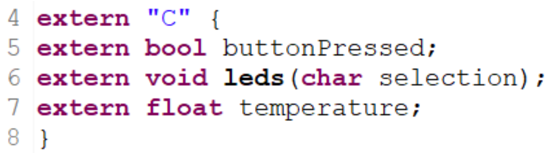
1. Recall that the files you will need to modify for MVP are all found in the `TouchGFX/gui` folder.



```
TouchGFX
  App
  assets
  build
  config
  generated
  gui
    include
      gui
        common
        model
          Model.hpp
          ModelListener.hpp
        screen_screen
          screenPresenter.hpp
          screenView.hpp
    src
      common
      model
        Model.cpp
      screen_screen
        screenPresenter.cpp
        screenView.cpp
```

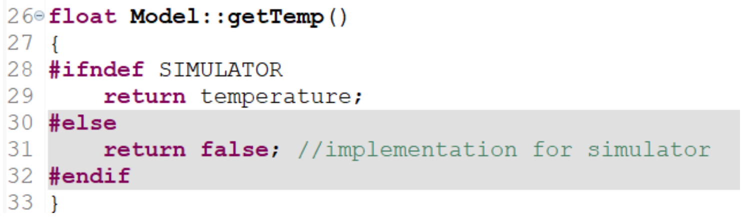
2. In `Model.cpp`:

   Declare `extern` variable for temperature that was declared in `main.c`:



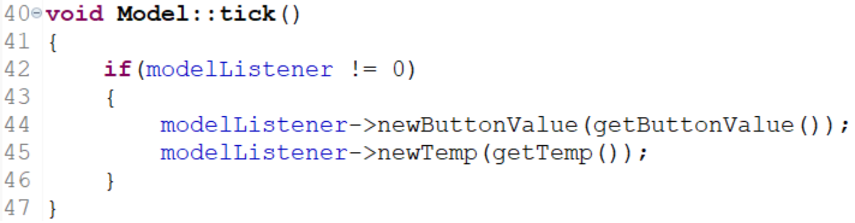
```
extern "C" {
extern bool buttonPressed;
extern void leds(char selection);
extern float temperature;
}
```

   Create a `getTemp` function:



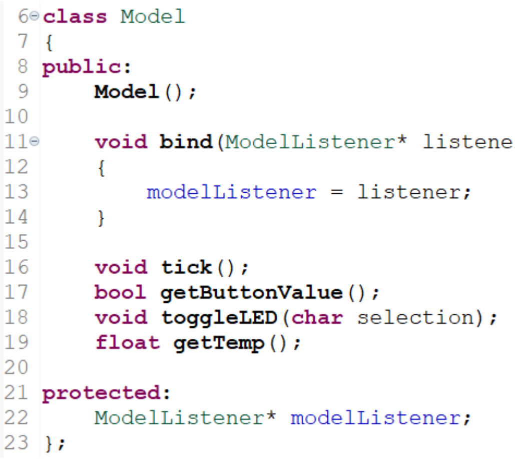
```
float Model::getTemp()
{
#ifndef SIMULATOR
    return temperature;
#else
    return false; //implementation for simulator
#endif
}
```

Add code to call `getTemp` from Model Listener's function `newTemp`:



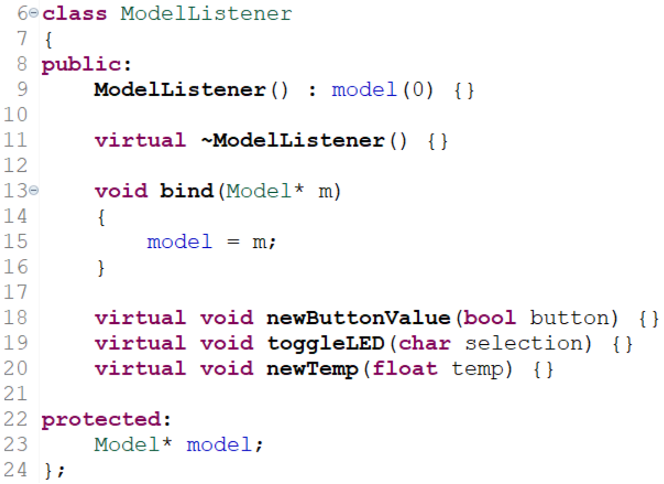
```
void Model::tick()
{
    if(modelListener != 0)
    {
        modelListener->newButtonValue(getButtonValue());
        modelListener->newTemp(getTemp());
    }
}
```

3. In `Model.hpp`, declare a prototype for the `getTemp` function.



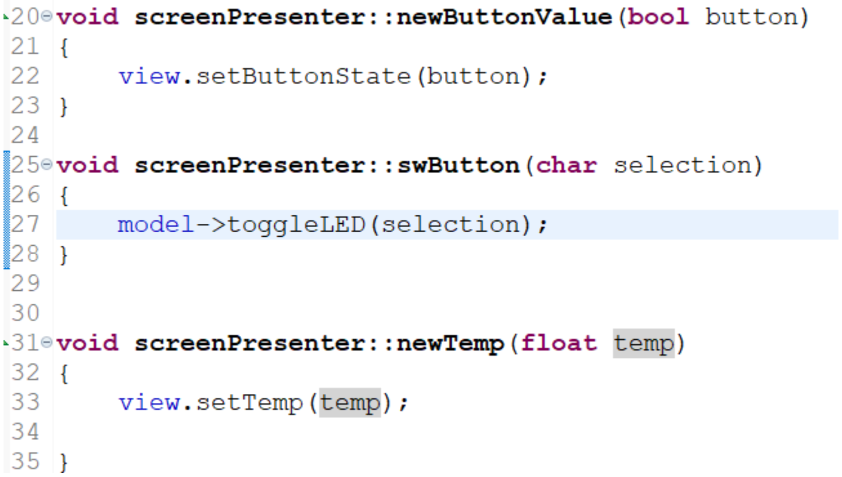
```
class Model
{
public:
    Model();

    void bind(ModelListener* listener)
    {
        modelListener = listener;
    }

    void tick();
    bool getButtonValue();
    void toggleLED(char selection);
    float getTemp();

protected:
    ModelListener* modelListener;
};
```

4. In `ModelListener.hpp`, define an empty virtual `newTemp` function.



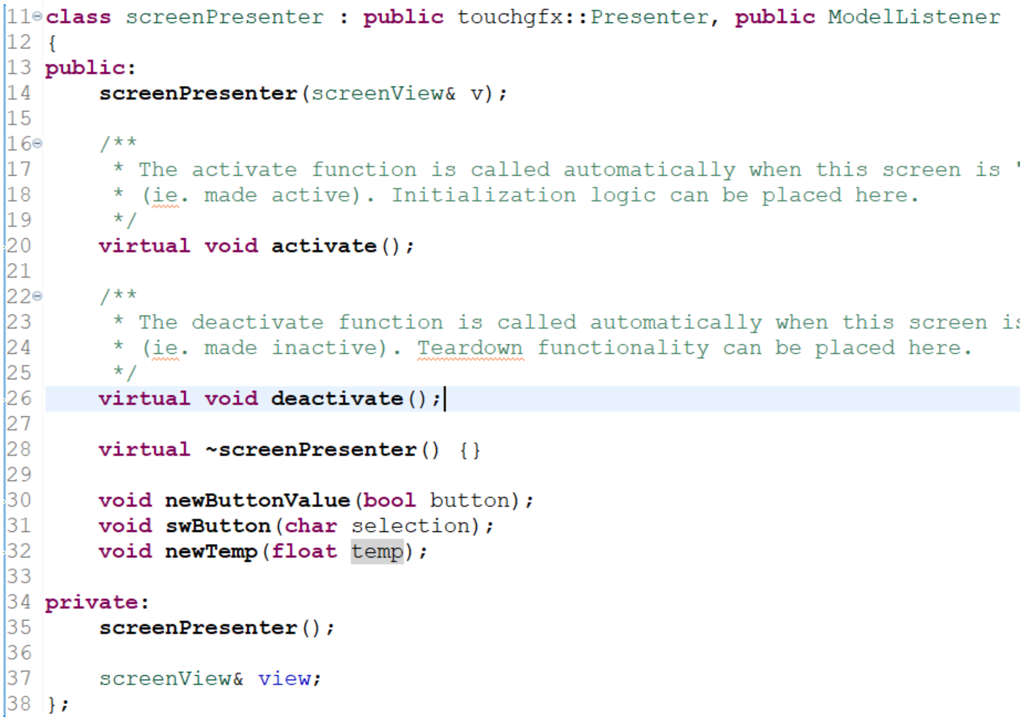
```
class ModelListener
{
public:
    ModelListener() : model(0) {}

    virtual ~ModelListener() {}

    void bind(Model* m)
    {
        model = m;
    }

    virtual void newButtonValue(bool button) {}
    virtual void toggleLED(char selection) {}
    virtual void newTemp(float temp) {}

protected:
    Model* model;
};
```

5. In `screenPresenter.cpp`, write a `newTemp` function that will override the empty one in ModelListener. Note that this function calls a `setTemp` function that will be defined in View.



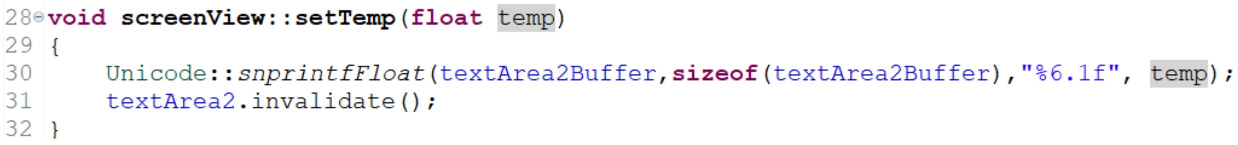
```
void screenPresenter::newButtonValue(bool button)
{
    view.setButtonState(button);
}

void screenPresenter::swButton(char selection)
{
    model->toggleLED(selection);
}


void screenPresenter::newTemp(float temp)
{
    view.setTemp(temp);

}
```

6. In `screenPresenter.hpp`, write a prototype for the `newTemp` function.



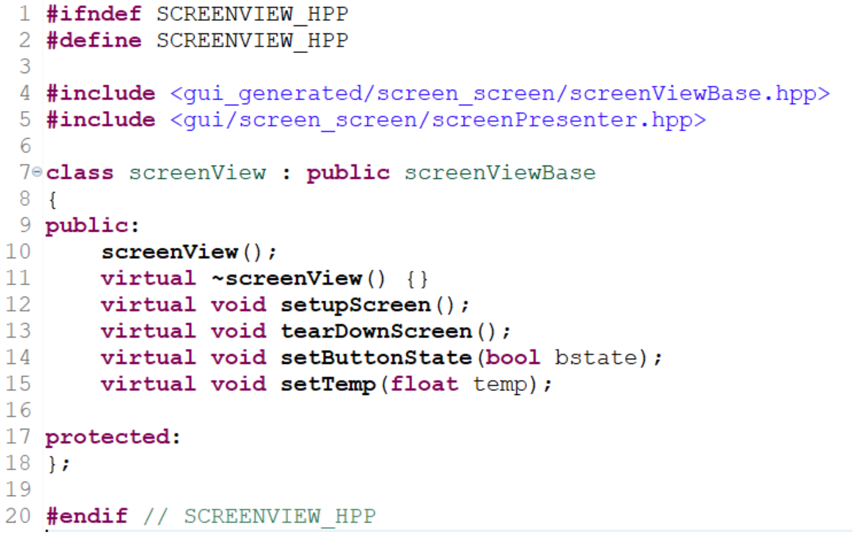
```
class screenPresenter : public touchgfx::Presenter, public ModelListener
{
public:
    screenPresenter(screenView& v);

    /**
     * The activate function is called automatically when this screen is '
     * (ie. made active). Initialization logic can be placed here.
     */
    virtual void activate();

    /**
     * The deactivate function is called automatically when this screen is
     * (ie. made inactive). Teardown functionality can be placed here.
     */
    virtual void deactivate();

    virtual ~screenPresenter() {}

    void newButtonValue(bool button);
    void swButton(char selection);
    void newTemp(float temp);

private:
    screenPresenter();

    screenView& view;
};
```

7. In `screenView.cpp`, write the `setTemp` function, which changes what is seen on the LCD screen according to information sent from Presenter. The temperature is displayed in a text area as a float value. The `invalidate` command tells the LCD the current display is invalid and forces it to update. Be sure to refer to the correct text area, that is, the area where your interface will print temperature may be textArea3 or textArea4. You can discover this information by clicking on the text area in TouchGFX. As with the light_on and light_off images, you can edit the text areas' names in TouchGFX.



```
void screenView::setTemp(float temp)
{
    Unicode::snprintfFloat(textArea2Buffer,sizeof(textArea2Buffer),"%6.1f", temp);
    textArea2.invalidate();
}
```

8. In `screenView.hpp`, add a prototype for the `setTemp` function.



```
#ifndef SCREENVIEW_HPP
#define SCREENVIEW_HPP

#include <gui_generated/screen_screen/screenViewBase.hpp>
#include <gui/screen_screen/screenPresenter.hpp>

class screenView : public screenViewBase
{
public:
    screenView();
    virtual ~screenView() {}
    virtual void setupScreen();
    virtual void tearDownScreen();
    virtual void setButtonState(bool bstate);
    virtual void setTemp(float temp);

protected:
};

#endif // SCREENVIEW_HPP
```

Compile your project. Run – Debug. Play/resume.

In addition to the functionality of the hardware button and the software buttons previously added, the thermistor's temperature should now be displayed on your LCD screen. It should provide live updates when you touch the thermistor.

Appendix 1 Calibrating the touchscreen

Creating a calibration project requires many of the same steps as you followed for your `tft_lcd` project. Instead of repeating all the steps, go to your STM32 workspace folder and make a copy of your `tft_lcd` project folder. Rename the copied folder `tft_lcd_calibrate`.

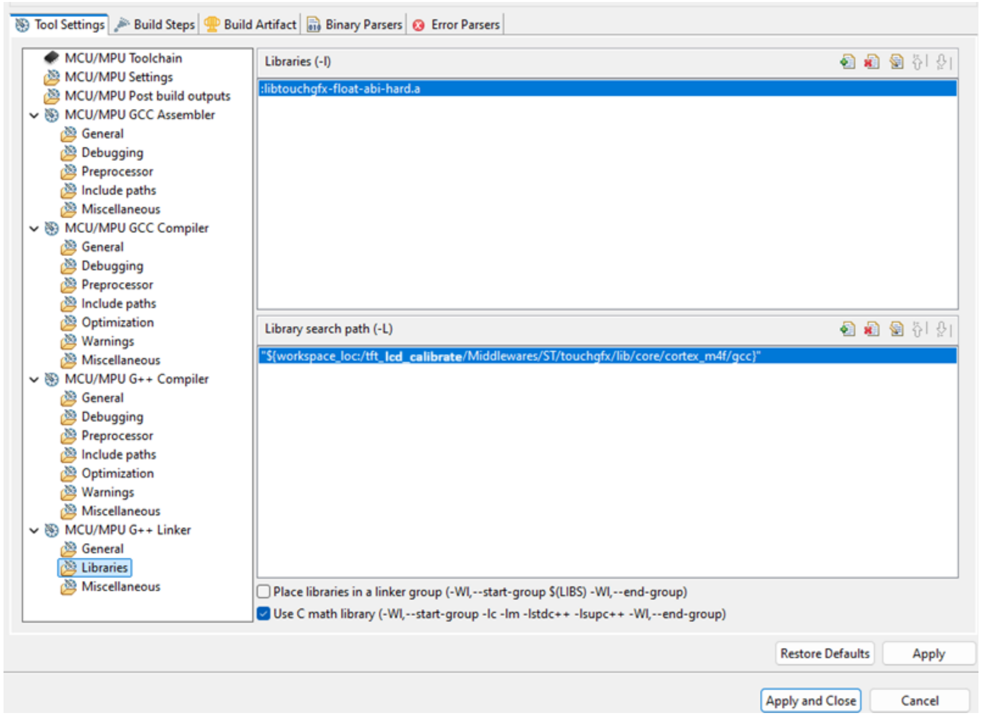
In the File Explorer, ensure hidden file are showing. If there is a `.git` folder in your project folder, delete it.

Open each of the following files in the Notepad and change every occurrence of `tft_lcd` to `tft_lcd_calibrate` (including in the filenames). The last three of the files are in the `TouchGFX` directory of your project.

```
.project
.cproject
tft_lcd.ioc
tft_lcd Debug.launch
ApplicationTemplate.touchgfx.part
target.config
tft_lcd.touchgfx
```

In STM32CubeIDE, choose File – Open Projects from File System..., locate your new `tft_lcd_calibrate` project folder and click Select Folder. Compile your project.

Right-click on your project's name to open its Properties. At the left, under C/C++ Build, choose Settings. Click on Libraries and verify that the library search path contains the `tft_lcd_calibrate` project name.



Libraries (-l)

:libtouchgfx-float-abi-hard.a

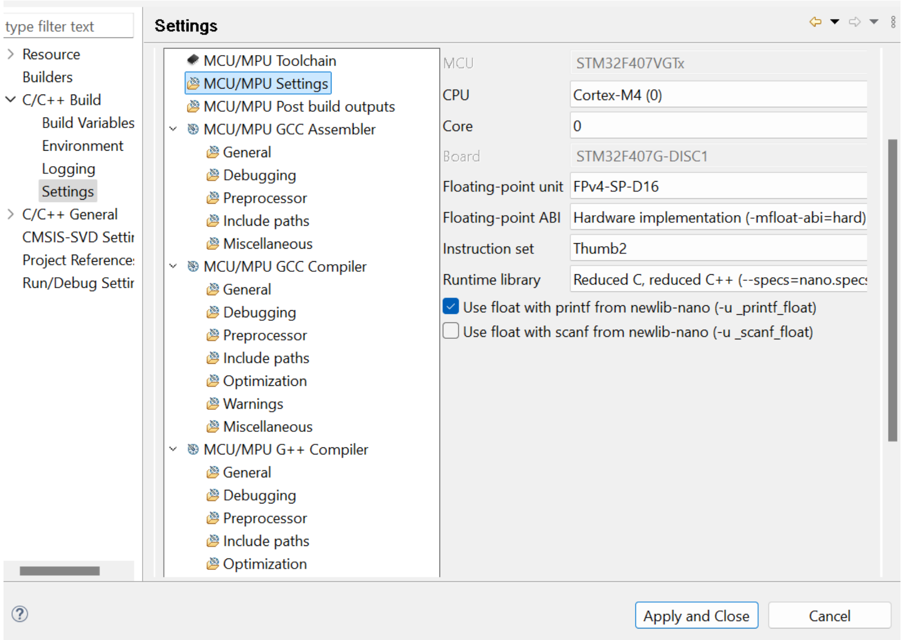
Library search path (-L)

"${workspace_loc:/tft_**lcd_calibrate**/Middlewares/ST/touchgfx/lib/core/cortex_m4f/gcc}"

☐ Place libraries in a linker group (-Wl,--start-group $(LIBS) -Wl,--end-group)

☑ Use C math library (-Wl,--start-group -lc -lm -lstdc++ -lsupc++ -Wl,--end-group)

Still in Properties, go to C/C++ Build Settings – MCU/MPU Settings. Check off "Use float with printf..." This will enable printing of float values.



MCU	STM32F407VGTx
CPU	Cortex-M4 (0)
Core	0
Board	STM32F407G-DISC1
Floating-point unit	FPv4-SP-D16
Floating-point ABI	Hardware implementation (-mfloat-abi=hard)
Instruction set	Thumb2
Runtime library	Reduced C, reduced C++ (--specs=nano.spec:

☑ Use float with printf from newlib-nano (-u _printf_float)

□ Use float with scanf from newlib-nano (-u _scanf_float)

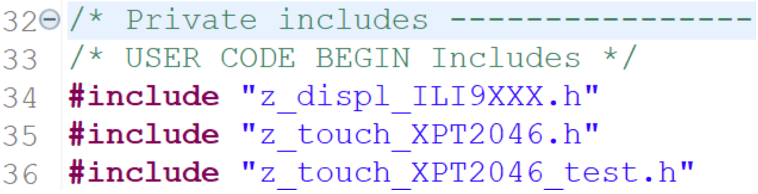
Some changes to code are required to change the original project to a calibration project:

1. From `ILI9XXX-XPT2046-STM32-main.zip` downloaded earlier from https://github.com/meldundas/ILI9XXX-XPT2046-STM32, extract the following files:

   Add `z_touch_XPT2046_test.c` and five fonts files (`font8.c`, `font12.c`, `font16.c`, `font20.c`, `font24.c`) to `Core/Src`.

   Add `z_touch_XPT2046_test.h` and `fonts.h` to `Core/Inc`.

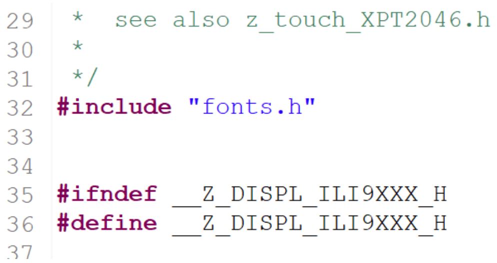
   In `main.h`, add a `#include` statement for `z_touch_XPT2046_test.h`.



```
/* Private includes ----------------
/* USER CODE BEGIN Includes */
#include "z_displ_ILI9XXX.h"
#include "z_touch_XPT2046.h"
#include "z_touch_XPT2046_test.h"
```

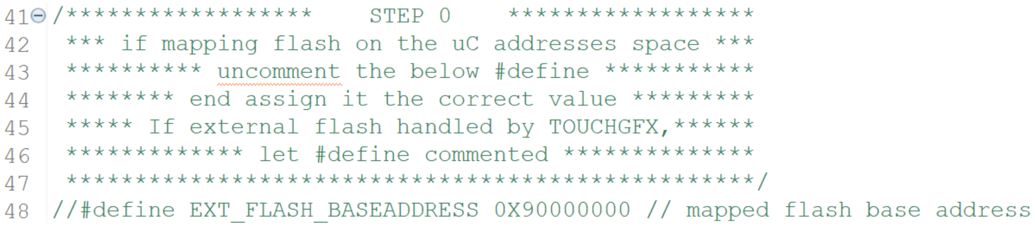
2. Go to the `Core/Inc` folder and open `z_displ_ILI9XXX.h`. In `z_displ_ILI9XXX.h`:

   Add a `#include` statement for `fonts.h`:



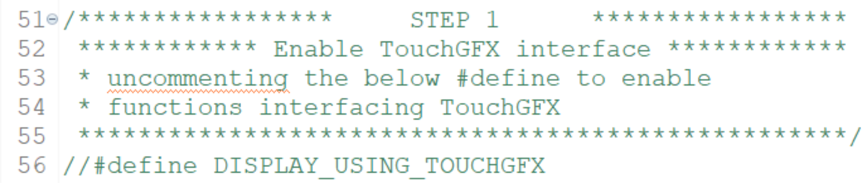
```
 *  see also z_touch_XPT2046.h
 *
 */
#include "fonts.h"


#ifndef __Z_DISPL_ILI9XXX_H
#define __Z_DISPL_ILI9XXX_H

```

Comment out `EXT_FLASH_BASEADDRESS`.



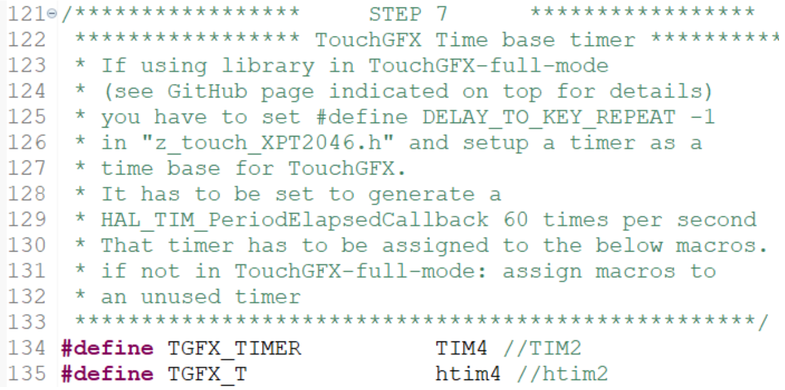
```
/******************    STEP 0    ******************
 *** if mapping flash on the uC addresses space ***
 ********** uncomment the below #define ***********
 ******** end assign it the correct value *********
 ***** If external flash handled by TOUCHGFX,******
 ************* let #define commented **************
 **************************************************/
//#define EXT_FLASH_BASEADDRESS 0X90000000 // mapped flash base address
```

Comment out `DISPLAY_USING_TOUCHGFX`.



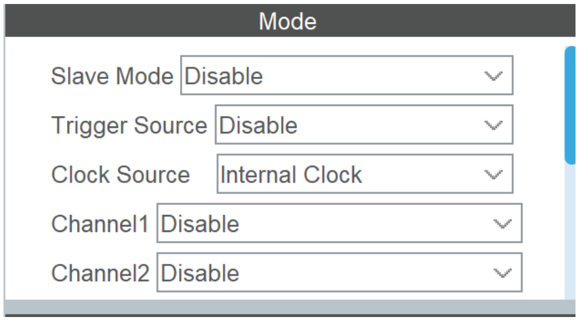
```
/*****************     STEP 1      *****************
 ************ Enable TouchGFX interface ************
 * uncommenting the below #define to enable
 * functions interfacing TouchGFX
 **************************************************/
//#define DISPLAY_USING_TOUCHGFX
```

Assign the TouchGFX timer to an unused timer, such as Timer 4.



```
/*****************     STEP 7      *****************
 ***************** TouchGFX Time base timer **********
 * If using library in TouchGFX-full-mode
 * (see GitHub page indicated on top for details)
 * you have to set #define DELAY_TO_KEY_REPEAT -1
 * in "z_touch_XPT2046.h" and setup a timer as a
 * time base for TouchGFX.
 * It has to be set to generate a
 * HAL_TIM_PeriodElapsedCallback 60 times per second
 * That timer has to be assigned to the below macros.
 * if not in TouchGFX-full-mode: assign macros to
 * an unused timer
 **************************************************/
#define TGFX_TIMER          TIM4 //TIM2
#define TGFX_T              htim4 //htim2
```

Open your .ioc file. Under Timers, select the timer you chose and set its clock source to Internal Clock. No need to configure anything else about this timer. Generate code.



Mode

Slave Mode Disable

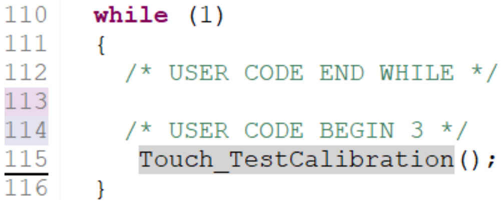
Trigger Source Disable

Clock Source Internal Clock

Channel1 Disable

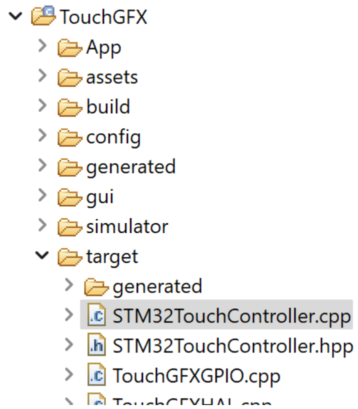
Channel2 Disable

3. From `Core/Src`, open `main.c`. Edit your `while(1)` loop:

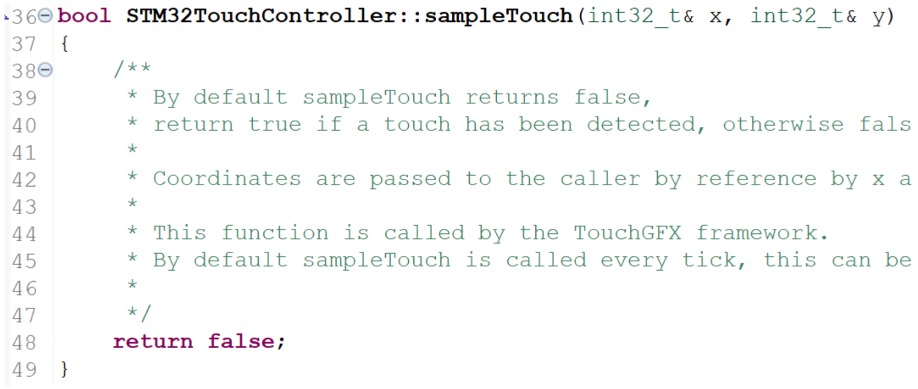


```
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
   Touch_TestCalibration();
}
```

4. From `TouchGFX/target`, open `STM32TouchController.cpp`.



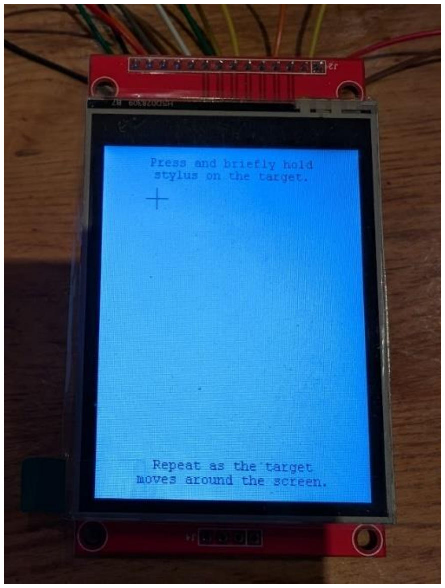
Edit the `sampleTouch` function to return `false`:



```
bool STM32TouchController::sampleTouch(int32_t& x, int32_t& y)
{
    /**
     * By default sampleTouch returns false,
     * return true if a touch has been detected, otherwise fals
     *
     * Coordinates are passed to the caller by reference by x a
     *
     * This function is called by the TouchGFX framework.
     * By default sampleTouch is called every tick, this can be
     *
     */
    return false;
}
```

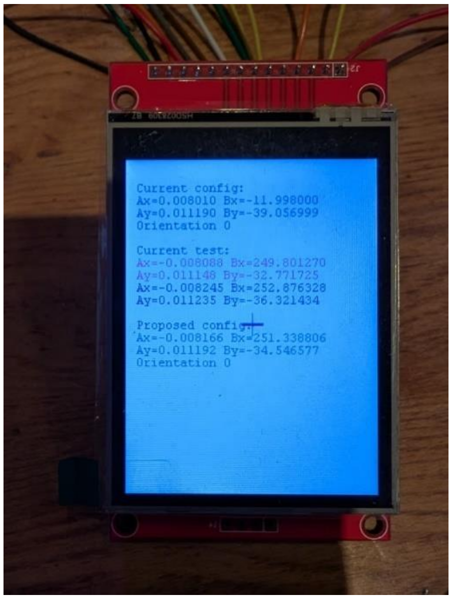
Compile your calibration project.

Run – Debug. Follow the directions on the screen and tap on the crosshairs.



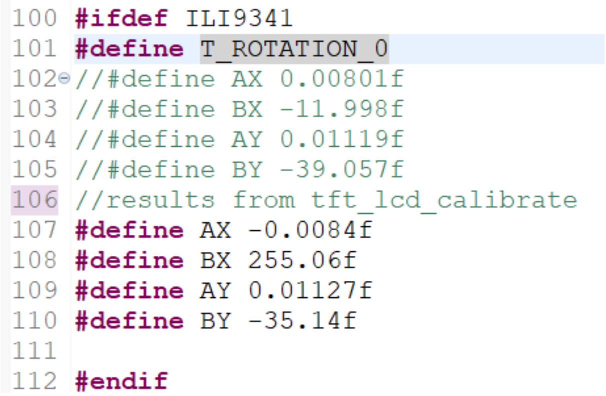
```
Press and briefly hold
stylus on the target.

Repeat as the target
moves around the screen.
```

At the conclusion of the test, a proposed configuration is given. Make a note of the numbers.



```
Current config:
Ax=0.008010 Bx=-11.998000
Ay=0.011190 By=-39.056999
Orientation 0

Current test:
Ax=-0.008088 Bx=249.801270
Ay=0.011148 By=-32.771725
Ax=-0.008245 Bx=252.876328
Ay=0.011235 By=-36.321434

Proposed config:
Ax=-0.008166 Bx=251.338806
Ay=0.011192 By=-34.546577
Orientation 0
```

Once you have completed the calibration activity, go back to your `tft_lcd` project. From `Core/Inc`, open `z_touch_XPT2046.h`. Locate the calibration parameters and enter the new values. Recompile and run-debug your `tft_lcd` project to verify that your buttons are behaving as expected.



```
#ifdef ILI9341
#define T_ROTATION_0
//#define AX 0.00801f
//#define BX -11.998f
//#define AY 0.01119f
//#define BY -39.057f
//results from tft_lcd_calibrate
#define AX -0.0084f
#define BX 255.06f
#define AY 0.01127f
#define BY -35.14f

#endif
```