TouchGFX AL Development
Introduction

Agenda

1. Introduction
2. TouchGFX Abstraction Layer Introduction
3. Abstraction Layer Architecture
4. TouchGFX Generator

life.augmented

The agenda for this video is a short introduction to the aims of this presentation, to what the TouchGFX Abstraction layer is, to its architecture, and finaly an introduction to the TouchGFX Generator.

Introduction

Introduction

Goal of this presentation

- Getting to know the TouchGFX Abstraction Layer and TouchGFX Generator
  - Introduction to TouchGFX AL
  - Abstraction Layer Architecture
  - TouchGFX Generator

Target audience: Developers new to TouchGFX AL development

life.augmented

Documentation Link: Documentation 4

What to expect of this presentation: it is aimed to TouchGFX beginners and people interested in developing the Abstraction Layer for their project.
After this presentation you will have a general overview- and basic understanding of the Abstraction Layer between TouchGFX and the hardware it runs on. The Abstraction layer may be abbreviated to AL in the slides and documentation.

You will be introduced to the TouchGFX abstraction layer development process, to the key responsibilities of the Abstraction Layer and how to work with the TouchGFX Generator.

Introduction

Further reading

- You will find a lot of help afterwards in the TouchGFX documentation site:

  http://support.touchgfx.com/
- Slides in this presentation will refer to relevant documentation pages. Links will be in the lower right-hand corner of the slides
- A good place to start reading following this presentation is:

  TouchGFX AL Development - Introduction

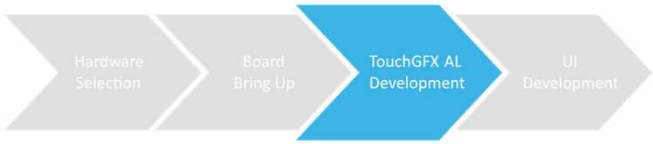
life.augmented

You will find the slides of this presentation in the TouchGFX online documentation (https://support.touchgfx.com/docs/resources/presentations#touchgfx-technical-introduction).
You can go through the other presentations, workshops and tutorials to understand how TouchGFX works and how to get started on your Graphical user interface project. (https://support.touchgfx.com/docs/resources/presentations and https://support.touchgfx.com/docs/tutorials/tutorial-01)

Introduction

TouchGFX AL Development

Main Activities:



Main components:

In this presentation we will focus on the TouchGFX AL development activity and the TouchGFX Generator development workflow

life.augmented

Documentation Link: TouchGFX AL Development Introduction 6

In a TouchGFX application, the TouchGFX Abstraction Layer is the software component that sits between the Board Initialization Code and the TouchGFX Engine. The Board bring up phase is explained in the corresponding presentation.
The TouchGFX Engine is not an output of any main activity but is the starting point for your TouchGFX project. The main task of the TouchGFX Abstraction layer is to tie together the Engine with the hardware and the operating system.

Abstraction Layer Introduction

The abstraction layer consists of two different parts, the Hardware Abstraction Layer and the Operating System Abstraction Layer.

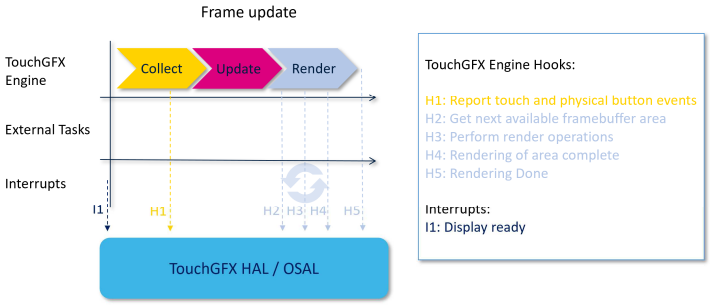
Responsibilities of the TouchGFX Engine

A TouchGFX Abstraction Layer consists of:

- Hardware Abstraction Layer (HAL)
- Operating System Abstraction Layer (OSAL)

Main responsibilities of the Engine:

- Collect inputs
- Update the Scene Model
- Render to the framebuffer



Available hooks and interrupts

life.augmented

Documentation Link: Responsibilities of the Abstraction Layer 8

The TouchGFX Engine's main responsibility is to update the framebuffer to reflect the current state of the application. It has a main loop that performs three basic steps forever

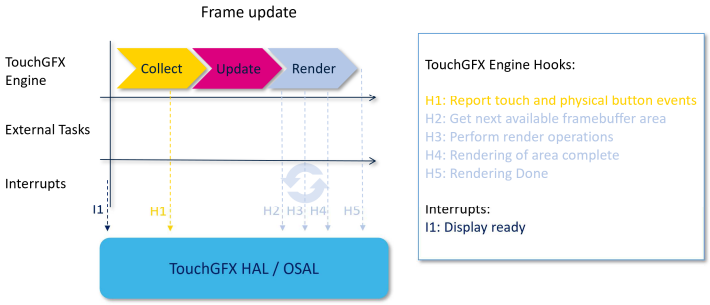
1. It first collects the inputs, like the coordinates of a user's touch or if a button has been pressed or released
2. It updates the application UI Model
3. and then renders the updated Model to the Framebuffer

The transfer of the framebuffer data to the display and the collection of external inputs tasks are not directly handled by the engine. They are delegated from the engine to the TouchGFX abstraction layer.

Responsibilities of the Abstraction Layer

Responsibilities of the AL:

- Synchronize TouchGFX Engine main loop with display transfer
- Report touch and physical button events
- Synchronize framebuffer access
- Report the next available framebuffer area
- Perform render operations
- Handle framebuffer transfer to display



Available hooks and interrupts

life.augmented

Documentation Link: Responsibilities of the Abstraction Layer 9

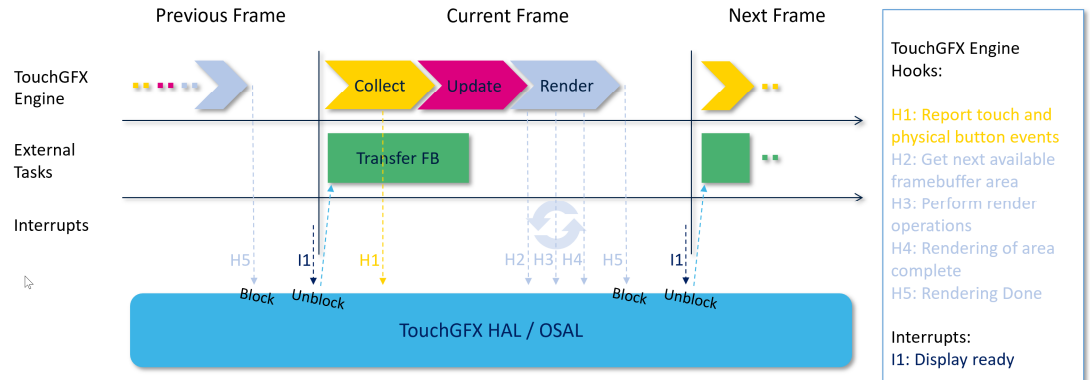
The main loop of the TouchGFX Engine will continuously update the framebuffer. This process must be synchronized with the actual update frequency and availability of the display to ensure that all frames will be transferred and displayed correctly on the display. This synchronization is the responsibility of the abstraction layer.
If the process is not properly synchronized, the main loop might overwrite the framebuffers before they have been transferred, and disturbing glitches will appear on the display.

The TouchGFX abstraction layer also has the responsibility of controlling the framebuffer memory area and its access, which means that all accesses to the framebuffer will go through the abstraction layer.

The responsibilities of the abstraction layer are further detailed in the documentation in the TouchGFX AL development introduction article.

Responsibilities of the Abstraction Layer

Parallel RGB (LTDC) display interface example :



Execution flow in a setup with two framebuffers and LTDC interface

life.augmented

Documentation Link: Responsibilities of the Abstraction Layer 10

The TouchGFX abstraction layer is a passive software module. It does not have its own thread so it must perform its actions through certain hooks. Those hooks are called from the TouchGFX Engine's main loop or through interrupts.
It is up to the AL developer to implement these hooks to ensure that the responsibilities of the abstraction layer are respected . If the AL developer needs other means to support the responsibilities, the developer can setup interrupts to activate at certain points
The role and timing of the hooks and interrupts can be seen in this example. The *I1: Display ready* interrupt is an example of a vertical synchronization interrupt. This describes the overall design of the AL for this setup.

Abstraction Layer Architecture

life.augmented

Abstraction Layer Architecture

Responsibility reminder

Responsibility	Area	Description
Synchronize TouchGFX Engine Main Loop with display transfer	HAL/OSAL (required)	TouchGFX Engine Main Loop blocks after rendering. Display signals TouchGFX Engine when ready to process new frames.
Report Touch and Physical Events	HAL (optional)	If available, touch- and physical events (e.g. buttons) can be reported to TouchGFX Engine.
Synchronize Framebuffer access	OSAL (required)	Ensures that only one actor accesses a framebuffer.
Report next available Framebuffer area	HAL (required)	Report where TouchGFX Engine should render next. Depends on the framebuffer strategy.
Perform Render Operations	HAL (optional)	Implementation of additional rendering capabilities, e.g. DMA2D.
Handle Framebuffer transfer to display	HAL (required)	Depends on display interface type. For e.g. SPI/FMC developers initiate the transfer manually when asked to.

life.augmented

Responsibilities description table

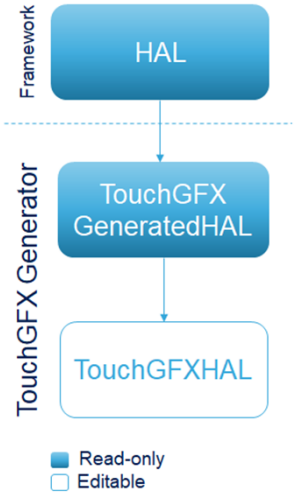
Documentation Link: Abstraction Layer Architecture

12

As described in the previous section, the TouchGFX AL has a particular set of responsibilities. The following table summarizes these responsibilities and in which area they intervene; Responsibilities are either implemented in the hardware Abstraction layer (so the part called hardware abstraction layer or HAL) or the operating system abstraction layer which synchronizes with TouchGFX Engine through a real-time operating system like FREERTOS. Note that not all steps are required to work with TouchGFX, like collecting data from physical events or performing additional rendering operations like the DMA2D.

Abstraction Layer Classes

- HAL responsibilities implemented in sub-classes of HAL based on STM32CubeMX MCU configuration
- OSAL automatically generated based on STM32CubeMX Middleware configuration
  - CMSIS RTOS V1 or V2
  - Other RTOS to be implement by the developer



Hierarchy of generated code

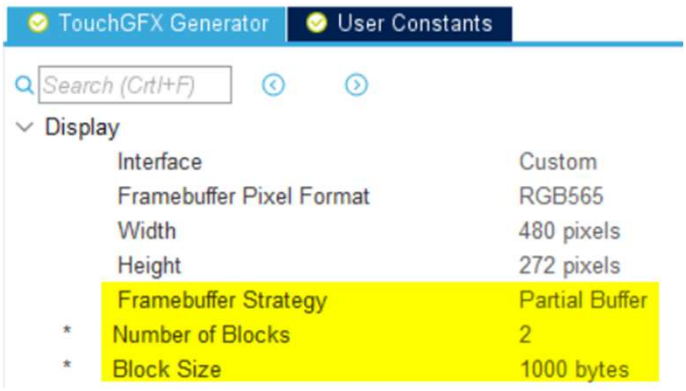
life.augmented

Documentation Link: Abstraction Layer Classes

The hardware abstraction layer is accessed by the TouchGFX Engine through sub-classes of the hardware abstraction layer, which are generated by the TouchGFX Generator. TouchGFX Generator is the main tool for the creation and development of the Abstraction Layer. It generates the part of the hardware abstraction layer configured in CubeMX. It also generates the Operating system abstraction layer.
The RTOS available are FREERTOS CMSIS V1 and V2. You can also work without an OS. For other RTOS it is the task of the developers to implement this operating system abstraction layer by themselves.
The common architecture of the hardware abstraction layer is shown in this figure. Changes through user code can be made in the file called TouchGFXHAL, like for example the number of framebuffers used and their addresses.
The Abstraction layer classes are further explained in the related articles in the documentation.

Advanced Topics

- Development steps documentation
- Scenarios
  - LTDC/Parallel RGB
  - FMC and SPI display interface
  - Framebuffer strategies
- Workshops



Partial framebuffer scenario

life.augmented

Documentation Link: Specific Scenarios 14

As the abstraction layer development and the steps to follow can be complicated, we recommend developers to go through the related articles. The responsibilities are further explained, with also sampled code from the generated files.

For concrete examples, some specific scenarios are gone through in the documentation, like how to support various display interfaces .

Workshops are also available on the official ST youtube channel and in the Resources section in the documentation. Most TouchGFX development workshops will and have to go through the TouchGFX abstraction layer and the TouchGFX Engine configuration in CubeMX.

TouchGFX Generator

This section is an introduction to TouchGFX Generator and not a user guide.

TouchGFX Generator

The TouchGFX Abstraction Layer is generated by TouchGFX Generator

- TouchGFX Generator is part of the X-CUBE-TouchGFX package

- When enabled, TouchGFX Generator creates a TouchGFX Abstraction Layer accordingly to the user settings

life.augmented

Documentation Link: TouchGFX Generator 16

TouchGFX Generator is a STM32CubeMX Additional-Software component that helps developers configure TouchGFX to run on their hardware platform. Based on the user's settings in CubeMX, the TouchGFX Generator will generate the files required to configure a working TouchGFX project. This include files for the TouchGFX Hardware Abstraction Layer, TouchGFX Operating System Abstraction Layer and TouchGFX Configuration.

Once code is generated through CubeMX, the TouchGFX project can be opened with the TouchGFX Designer tool where the UI is developed. TouchGFX Designer automatically adds any additional generated code files to the target IDE project that was configured for the project in CubeMX.

TouchGFX Generator

Dependencies group

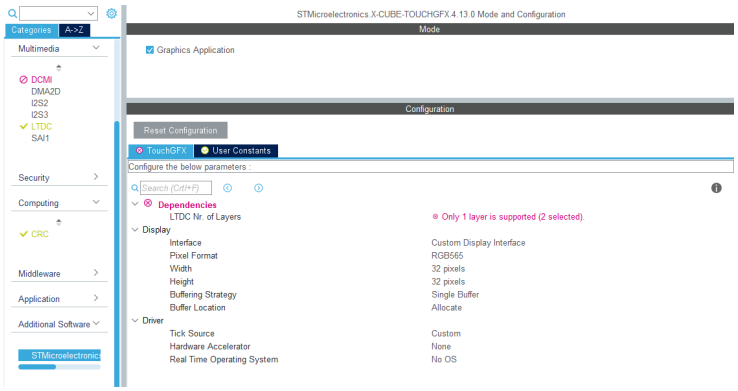
- List of warnings or information based on STM32CubeMX configuration

Display group

- Settings related to interface, format, dimensions and strategy

Driver group

- Settings related to driving and accelerating TouchGFX as well as selecting the OSAL



TouchGFX Generator user interface in STM32CubeMX

life.augmented

Documentation Link: TouchGFX Generator 17

The user interface of TouchGFX generator can be seen in this screenshot of CubeMX. It consists of three groups: the dependencies, display and driver groups

•***The Dependencies group*** - contains notifications to the developer about dependencies, warnings or concrete errors in the configuration. The group is hidden if no entries exist.

•***The Display*** group- which contains settings related to display such as interface, framebuffer bitdepth, width and height. The framebuffer strategies available are single and double framebuffers, or the partial frame buffer strategy. This strategy is aimed for scenarios looking to lower the memory the memory usage. These settings directly impact the size of the canvas of the TouchGFX project as well as the code generated for assets.

•***The Driver*** group - allows the user to opt-in for a number of ready-made drivers related to the tick source of the application, graphics acceleration and RTOS. Since CubeMX supports FreeRTOS (CMSIS RTOS v1 and v2) configurations, TouchGFX Generator provides drivers for each of these options.

TouchGFX Generator Code Architecture

The generated Abstraction Layer is created inside the TouchGFX folder when generating from CubeMX

- User code to be implemented in files not under the */target/generated* folder
  - Dedicated sections are highlighted in the code

```
.mxproject
myproject.ioc
├──Core
├──Drivers
├──EWARM
├──Middlewares
└──TouchGFX
    │   ApplicationTemplate.touchgfx.part
    ├──App
    │       app_touchgfx.c
    │       app_touchgfx.h
    └──target
        │   STM32TouchController.cpp
        │   STM32TouchController.hpp
        │   TouchGFXGPIO.cpp
        │   TouchGFXHAL.cpp
        │   TouchGFXHAL.hpp
        │
        └──generated
                OSWrappers.cpp
                TouchGFXConfiguration.cpp
                TouchGFXGeneratedHAL.cpp
                TouchGFXGeneratedHAL.hpp
```

Generated folder architecture

life.augmented

Documentation Link: Generated Code Architecture 18

CubeMX will create a *TouchGFX* folder for as project as seen in this screenshot. The folder always contains the same files, regardless of configuration, while the content of those files changes according to CubeMX and User configuration.
The listing below shows an overview of the content of a CubeMX project with TouchGFX Generator *enabled*. Thanks to the hierarchy of the hardware abstraction layer, the files that are not listed under the generated folder can be modified with user code in the dedicated zones for further configuration. You can for example allocate space for a third framebuffer called animation storage to enable complex animations in your project.

TouchGFX Generator

- Next step: TouchGFX GUI development in TouchGFX Designer
- Updates to TouchGFX Generator configuration reflected in TouchGFX Designer
  - Pixel format, screen dimensions, …
- Additional configuration code to be expected for custom hardware platforms
  - TouchGFX Generator configurations in the application templates of ST development kits can be used as source of inspiration

life.augmented

Documentation Link: TouchGFX Generator 19

After successfully setting the TouchGFX abstraction layer, The next step is now to start the graphical user interface application with TouchGFX Designer.
The settings and changes made in the Generator will be shown in Designer, like the size of the display

For custom hardware platforms the TouchGFX Generator can generate most of the abstraction layer. The remaining parts to be developed are pointed out through code comments in the generated files and notifications through the TouchGFX Generator.

Have a look at the TouchGFX abstraction layer workshop and TouchGFX generator documentation to further understand how to work with it. The application templates for specific ST development kits available in TouchGFX Designer can be used as source of configuration.

Links

Further reading

- You will find a lot of help afterwards in the TouchGFX documentation site:

  http://support.touchgfx.com/

- Slides in this presentation will refer to relevant documentation pages. Links will be in the lower right-hand corner of the slides
- A good place to start reading following this presentation is:

  TouchGFX AL Development – Introduction

- Write your questions in the ST forum “Community” to get help from other users and ST employees

  STMicroelectronics Forum

life.augmented

This is a reminder that the slides and additional knowledge can be found in the official documentation. The slides have links to the relevant articles. You can ask any questions on the official ST forum.
Thank you

Thank you

life.augmented

Thank you for listening to this introduction to the TouchGFX abstraction layer and to the TouchGFX Generator