Hardware selection guide

There are many parameters to evaluate when choosing hardware platform. This article attempts to explain what the best choices of hardware components are strictly in relation to TouchGFX graphics rendering and the UI performance you expect to achieve for your product. Naturally there are other important factors such as cost, availability, compatibility or vendor preference that are not taken into account here.

Still this article mentions many potential pitfalls in hardware choice, and we very much recommend reading it before making a final decision on components.

Needed components

In general you will need the following five components. These will be described in detail in the following sections.

1. A microcontroller
2. (External) RAM for frame buffers
3. (External) flash for storing graphics data
4. A display
5. A touch controller

Microcontroller

MCUs can be divided in versions with and without an integrated TFT controller peripheral. We recommend choosing an MCU with a TFT controller since it is more flexible and typically a cheaper overall solution because you do not need a display with integrated video RAM and display controller. TouchGFX does not support rendering directly into integrated video RAM so even if your display has integrated RAM you will still need an RAM for containing at least one frame buffer. For details on running TouchGFX on MCUs without TFT controllers, please see MCUs without TFT controller.

Since the main bottleneck is the ability to transfer pixels across the external memory bus, the clock frequency of the processor is quite important. Usually the external memories are clocked at half the core frequency.

In order to be able to render directly from the non-volatile graphics storage to the frame buffer, we recommend choosing an MCU which supports memory-mapping of your desired type of external flash. See more about this in the section on non-volatile storage.

Internal memory requirements

We recommend choosing an MCU with at least 32Kbytes of SRAM and 128Kbytes of flash. For

Recently viewed articles

Achieving better performance with CacheableContainer

Caching bitmaps on systems with limited RAM

TouchGFX HAL Development

X-CUBE-TOUCHGFX User Guide

Overview of TouchGFX

Related articles

TouchGFX memory requirements

MCUs without TFT controller

Changing to a different display

Integrating CubeMX and TouchGFX

Dynamic Bitmaps: Load images at runtime

details on the footprint of TouchGFX, please seeTouchGFX memory requirements

Frame buffer memory

A single frame buffer will in 16-bit colors take up `width x height x 2` bytes of memory, so for a 480x272 display this is roughly 256Kbytes. Therefore the frame buffer can only fit in internal SRAM with small screens. Internal SRAM is very fast, so a single frame buffer is often adequate when placed in internal SRAM. But usually you will need an external memory for holding frame buffers, and when placed externally you will need to employ double buffering due to the slower access time. We recommend choosing an SDRAM with 32-bit data bus since transfer speed to and from SDRAM is paramount. For exact parts, please refer to the supported evaluation boards which all have SDRAM with acceptable performance. Based on the above the amount of SDRAM you will need is `width x height x 2 x 2`. TouchGFX has a special concept of an `animationStorage` which is an optional third frame buffer. This is needed if you wish to use the full-screen sliding transition effect when switching between different screens in your application. Therefore you should be well covered with a 2-4Mbyte SDRAM depending on screen resolution. The SDRAM can also be used to store dynamic bitmaps. If you need dynamic bitmaps in your application, you may need more SDRAM.

Non-volatile storage

In most cases you will need an external non-volatile storage for graphics data. The choice of which type of storage to use has great impact on performance and must be considered with care. We recommend choosing a memory type which can be memory-mapped on your chosen microcontroller, since this means TouchGFX can render directly from the non-volatile storage into the frame buffer. If you instead choose a memory type which is not memory-mapped, you will need to cache all the bitmap data in SDRAM before using them. Read more about this in Non-memory mapped external flash.

Examples of memory mapped storage types:

- Parallel NOR flashes (universal)
- QuadSPI NOR (On e.g. STM32F469, STM32F746)

Examples of memory types that are *not* memory mapped and thus require caching

- All NAND-based flashes
- SD cards
- Anything with a file system
- QuadSPI NOR if your microcontroller does not include memory-mapping for it (e.g. STM32F429)

Note: Quad-SPI NOR flashes are substantially faster and cheaper than parallel NOR flashes and also require fewer pins. Therefore we greatly recommend choosing a quad-SPI flash and a microcontroller that supports memory-mapping of these.

Memory requirements

Typically is the external flash only used to store graphics data. So the amount of storage you need depends on how many bitmaps you have and their size. An opaque bitmap will take up `width x height x 2` bytes. A bitmap that contains alpha channel will take up `width x height x 4` bytes. If you have an idea of the scope of your application, you can use the above to calculate rough storage requirements. As a rule of thumb we generally recommend:

- 8MBytes for applications in 480x272 or smaller resolution
- 16Mbytes for applications in 640x480 or 800x480

Frequently the amount of storage needed is not known when the hardware is chosen. In this case we recommend choosing a high capacity part (16/32MBytes) for use during development and substitute for a smaller pin-compatible version before product release.

Platform Compatibility

There are many component combinations that are well suited for graphics rendering. It is quite likely that other requirements for your product will drive the MCU choice. Still it is important to align the requirements for the UI with your hardware choice. Some very important things to consider are:

1. What is the desired resolution of the display? The amount of pixels make a huge difference on performance as the all the pixels needs to be transferred on the bus, and in some cases calculated.
2. How big a part of the screen do you expect to be able to update at high frame rates when doing animations? E.g. do you need to support full-screen animations at 60 frames per second?
3. Do you intend to rely heavily on image scaling/rotation, alpha blending, blurring effects etc. which are expensive to calculate?
4. How much other work is the MCU going to perform while rendering graphics?
5. To what degree are you willing to spend time performance optimizing your application?

The above should help you get an idea of how powerful a platform you should choose. We would also recommend deploying some of our larger demo applications on an evaluation board resembling your target platform to get an idea of what is possible in terms of user interface. These demos include a readout of the MCU load percentage.

The following table lists a range of commonly used platforms and what kinds of user interfaces can be realised on them. Please note that it is very difficult to directly compare various solutions because there are individual strengths, so please use this table as a rough guideline only.

Table legend

Least concern. You should be able to implement pretty much any UI with this platform.

This level will be sufficient for the vast majority of applications, with only minor need for concern if you have very taxing full-screen animations.

Sufficient for most applications, but limited ability to do full-screen animations and texture mapping.

Sufficient for basic applications where animations only take up a small part of the screen area and texture mapping is used very sparingly or not at all.

Capable of doing only the most basic applications.

MCU/Resolution	**320x240**	**480x272**	**640x480**	**800x480**	**1024x768**

MCU/Resolution **320x240** **480x272** **640x480** **800x480** **1024x768**

Cortex M3/M4, up to 150MHz Parallel NOR flash or cached					
Cortex M3/M4, up to 150MHz Quad-SPI NOR flash					
Cortex M3/M4, 150 - 200MHz Parallel NOR flash or cached					
Cortex M3/M4, 150 - 200MHz Quad-SPI NOR flash					
Cortex M7, 200 - 300MHz Quad-SPI NOR flash					

The table above assumes a 32-bit SDRAM and a MCU work load of other tasks of less than 50%. If any of these assumptions are not true for your platform, subtract one level for each.

Choosing display

Choosing the right display can be difficult as there are many options, many levels of quality and a substantial price variations. A detailed guide is out of scope for this article, but we will point out some important considerations that might otherwise be overlooked:

- Attempt to find a display with square pixels. Otherwise the graphics will look slightly skewed, which would have to be handled by your graphics designers by adjusting the source images to compensate. Any runtime effects like drawing circles, stretching images etc. will also be affected and there is no built-in mechanism in TouchGFX to handle this.
- There are great differences in backlight and colors for various display models, so obtain a sample of the desired display and see if the color representation is acceptable. Also note that colors will often look slightly different on the display compared to what the graphics designers see on their PC monitor.
- Most displays provide either 18 or 24 bit color depth, whereas the TouchGFX frame buffer is maybe 16-bit. Some MCUs will upscale to 24-bit when outputting, but if there is only 16-bit output we recommend wiring any excess color input pins of the display (least significant) to ground. 18 bit displays are preferred over 24 bit, as the loss of brightness from unused bits is smaller. But even with 24-bit displays the loss is only 3% so it is hardly noticable in practice.
- Consider viewing angle requirements for your product. Some displays have at least one poor viewing angle.
- Consider the scanline direction and mounting orientation of the display. TouchGFX supports a 90 degree rotation so you can run landscape mode UIs on a portrait mode display or vice versa, but there are no builtin mechanism for e.g. turning the UI upside down. Some displays support several mounting orientations though.

Choosing touch input

Basically there are two types of touch input: capacitive and resistive touch. They behave quite differently and are also at different price points so it is worth considering which to choose.

Resistive touch

Resistive touch reacts to physical pressure being applied to the display, and can be actuated by anything (finger, pen, stylus). It is a cheaper technology than capacitive, and the touch readings are typically done by doing A/D conversions on analog pins of the MCU. Resistive touch works poorly with gesture recognition (dragging, swiping) because it relies on uniform pressure being applied while doing the interaction, and that is not what users will typically do in practice. Also it has a tendency of requiring more force to actuate than at least some users have come to expect. Finally, a resistive touch film is normally not completely transparent and therefore your display will loose some brightness and will not appear as crisp and clear as a capacitive touch displays.

Capacitive touch

When using capacitive touch, there will be a small dedicated touch controller IC typically mounted on the display tail. The MCU communicates with the touch controller using I2C or SPI. Capacitive touch is sensitive to being *touched* as opposed to applying pressure which means that it is much better suited for gesture recognition and in general feels more natural for smartphone users compared with resistive touch. Capacitive touch displays normally requires that the object touching the screen is a finger. This has advantages such as being able to wipe the screen with a cloth without risking to click anything but also has drawbacks such as it cannot be operated using gloves. Note that capacitive displays are more sensitive to electromagnetic interface, power supply variations and environment changes. We recommend consulting with your display vendor for details on how to cope with this.

Based on the above we recommend resistive touch displays only on very price sensitive applications where touch interaction is mainly click-based, with little or no gesture input or in cases where the touch actuation is done by something else than a finger.

Was this article helpful? 10 out of 10 found this helpful

Have more questions? Please create a post on the forum.