基于低成本 STM32 的图形应用

关键字：TouchGFX，Framebuffer，图形

前言

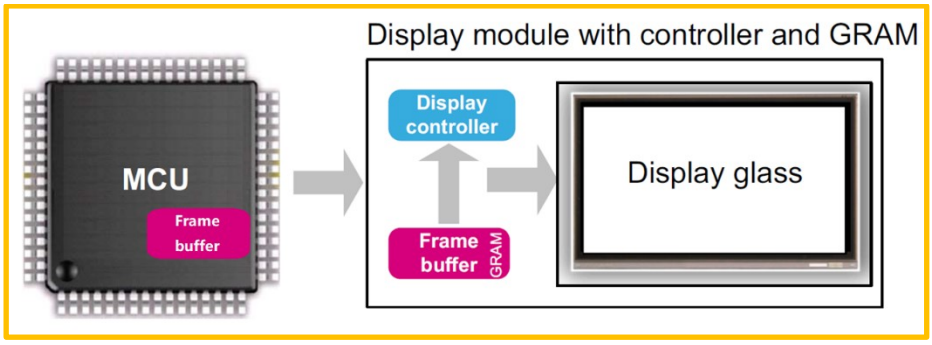
越来越多的智能设备会基于 STM32 实现图形界面，而 TouchGFX 是专门用于 STM32 的图形界面设计软件，使图形界面能达到类似智能手机的显示效果。通常，在支持 FMC、LTDC、MIPI-DSI 等 LCD 接口的 STM32，都有着比较丰富的内存资源，SRAM 存放帧缓冲也毫无压力。但在一些成本敏感的产品上，STM32 内存较小，不足以存放完整的帧缓冲，这类产品又如何使用 TouchGFX 来做图形界面应用呢？本文将介绍 TouchGFX 的部分帧缓冲特性，以及基于 STM32G0 系列 的移植过程。

部分帧缓冲

通常在做图形应用时，需要在 MCU 的 SRAM 里申请一块内存作为帧缓冲：framebuffer。比如使用一个分辨率为 240x320，16 位色深的 LCD，需要申请的内存大小为：240x320x2 = 153600 Bytes。而很多低成本的 MCU 内存较小，根本无法满足图形应用对内存的需求。不过，TouchGFX 现在支持部分帧缓冲，仅使用较小的内存作为帧缓冲即可支持图形应用。

Partial Framebuffer，部分帧缓冲

对内置了 GRAM 的 LCD 来说，图形显示的过程可以概括如下图：



1. MCU 将图形界面画到内部 Framebuffer
2. 画图过程结束后，将 Framebuffer 内容传输到 LCD GRAM
3. 由 LCD controller 将 GRAM 内容显示到屏幕

如果使用完整的 framebuffer，在上面第 2 步进行传输时，每次传输的都是一整屏的数据。而在很多场景中，LCD 屏幕上变化的区域并不大。比如一个控制界面，可能只有一个小图标会有变化，其它区域都保持不变。我们可以只更新屏幕变化的区域。

部分帧缓冲与此类似，framebuffer 可以被分成许多个小的缓存块去更新和传输，而这些小块的缓存可以被重用，这样可以极大的减小图形应用对内存的需求。而使用 partial framebuffer 的方式，仅有稍许限制：

1. LCD 模块必需要内置 GRAM
2. 在复杂界面时可能会有撕裂效果

部分帧缓冲在 STM32G071 上的应用

目标板上的 STM32G071 内置 128K flash，36K SRAM，外接 SPI 接口的 LCD，分辨率 240x320 像素，16 位色。按此 LCD 的分辨率，要支持 TouchGFX 图形应用，必须使用部分帧缓冲特性。

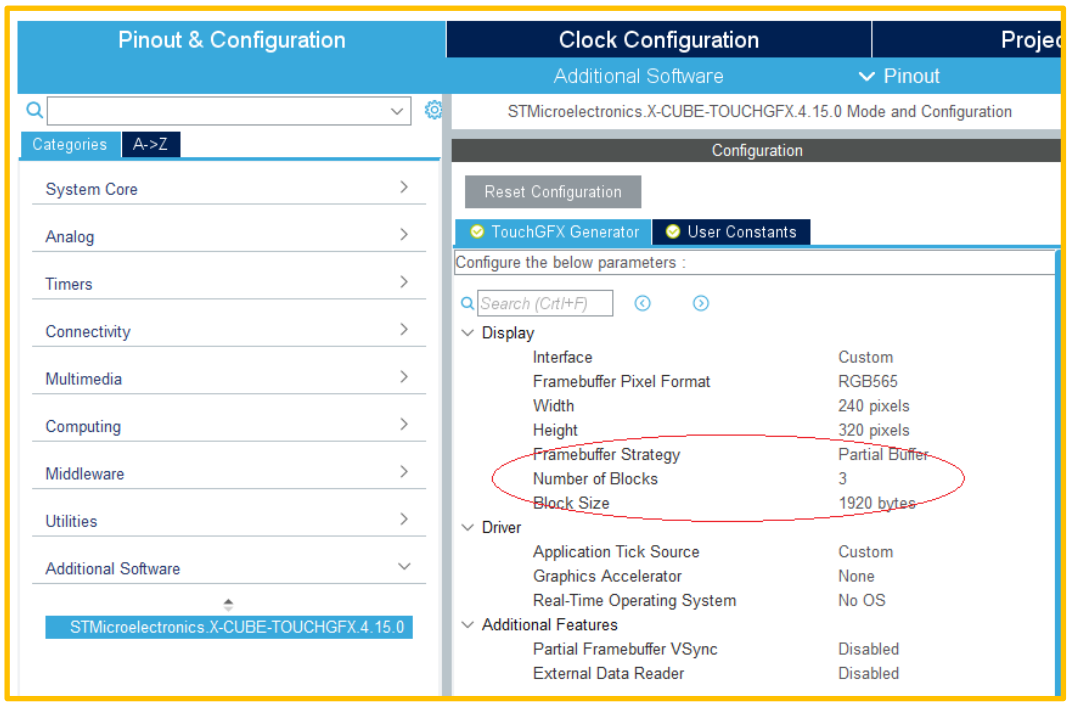
配置过程

整个配置过程主要包括以下四个方面：

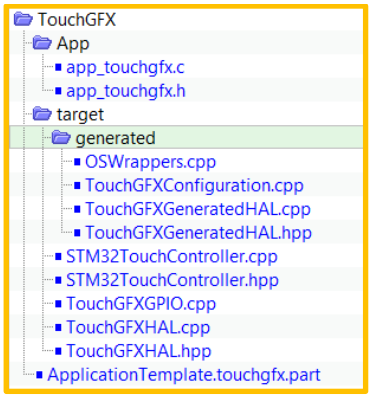
一、在 CubeMX 中，配置 TouchGFX generator 的 display 属性如下图：

- 像素格式: RGB565
- 宽: 240 像素
- 高: 320 像素
- 帧缓冲策略: Partial Buffer

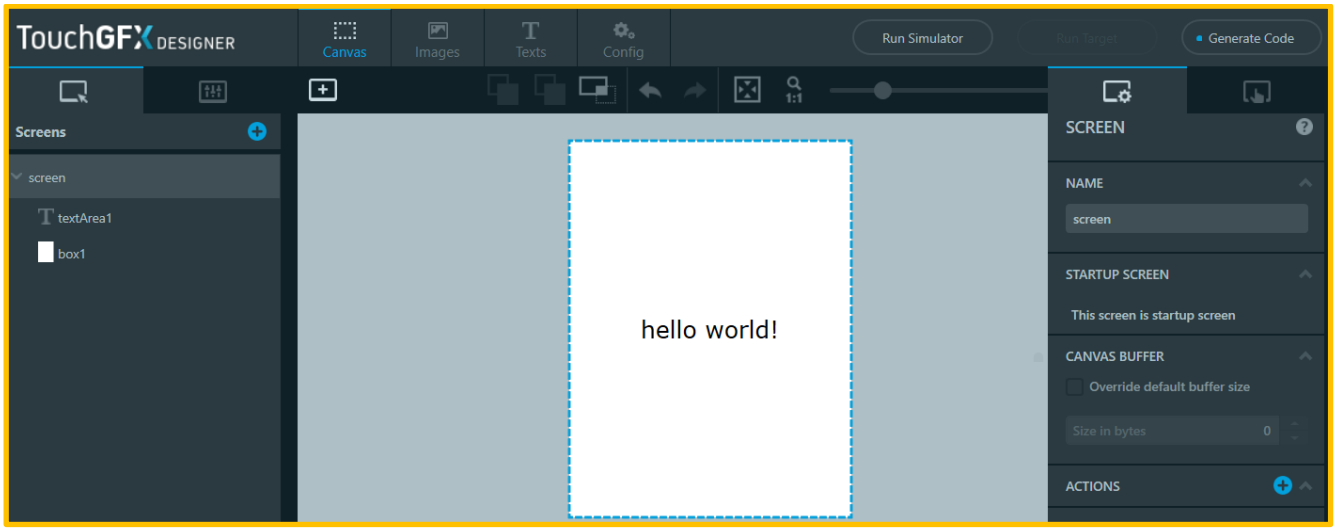
缓冲块个数和缓冲块大小都可以根据需要调整



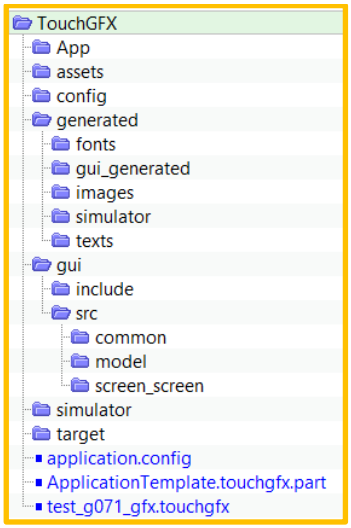
二、配置完成后，先由 CubeMX 生成代码。可以看到，会生成 TouchGFX 目录，以及该目录下的配置文件与部分源码文件。



三、打开 ApplicationTemplate.touchgfx.part 文件，即可打开 TouchGFX Designer 工具，如下图。比如，可以添加 box 控件作为背景，textArea 控件来显示文本。



然后通过 Generate Code 生成 UI 相关的代码，主要包含 generated 和 gui 两个目录。generated 目录包含生成的资源文件，如字体信息、文本信息、图片资源等，还包括各个 screen 的基类代码。而 gui 目录则包含 screen 的派生类代码。用户可基于 gui 目录进行 UI 后续的开发。



四、添加屏幕底层驱动

生成代码后，还需要添加屏幕的底层驱动和画图接口程序。到此为止，与 GUI 相关的配置与代码框架已经全部完成。

适配过程

完成以上配置后，已经可以将工程编译下载到目标板运行，但还不能显示图形界面。打开 main.c 文件可以看到，生成的代码中已经完成了 touchgfx 框架初始化及任务调用。还需要完成以下三个步骤才能显示图形界面。

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();

  MX_CRC_Init();          // TouchGFX 需要使用 CRC
  MX_TouchGFX_Init();     // 初始化 TouchGFX 框架

  while (1)
  {
    MX_TouchGFX_Process();       // 在主循环中运行 TouchGFX 任务
  }
}
```

1， 在 main 函数中加入 LCD 的初始化代码，比如 st7789v 的初始化：

```
void ctm_lcd_init(void)
{
    lcd_drv = &st7789v_drv;
    lcd_drv->Init();
    lcd_drv->DisplayOn();
}
```

2， 用 tim 产生周期性的中断，作为 TouchGFX tick 节拍（通常 LCD 刷新率 60Hz，tim 定时周期配置约为 16ms）

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if (htim->Instance == TIM7) {
      touchgfxSignalVSync();     // 触发 touchgfx 任务运行
    }
}
extern "C" void touchgfxSignalVSync(void)
{
  /* VSync has occurred, increment TouchGFX engine vsync counter */
  touchgfx::HAL::getInstance()->vSync();

  /* VSync has occurred, signal TouchGFX engine */
  touchgfx::OSWrappers::signalVSync();
}
```

3， 实现刷新 LCD GRAM 函数，在 TouchGFXHAL::flushFrameBuffer 中更新部分 LCD 区域

```
volatile uint8_t vol_isTransmittingData = 0;
extern "C" int touchgfxDisplayDriverTransmitActive()
{
  return vol_isTransmittingData;
}
extern "C" void touchgfxDisplayDriverTransmitBlock(const uint8_t* pixels, uint16_t x, uint16_t y,
uint16_t w, uint16_t h)
{
  //partial fb
  vol_isTransmittingData = 1;
  lcd_draw_image_tgfxPBuf(x,y,w,h,pixels);       // 实际更新 LCD，从坐标（x,y）开始，宽 w，高 h 的区域
  vol_isTransmittingData = 0;
  startNewTransfer();                            // 传输完成需调用
}
```

lcd_draw_image_tgfxPBuf 函数由用户根据 LCD 规格来实现，会根据 TouchGFX 框架给出的坐标和宽高参数，来更新 LCD 指定区域。

实现这三个步骤后，便可以在 G071 目标板上运行并显示图形界面了。

小结

借助 TouchGFX 部分帧缓冲功能，可以在低成本 MCU 上实现图形应用。本文档介绍了低成本 MCU 图形应用的配置流程，具体实现过程可参考电堂科技网站《STM32 & X-Cube-TouchGFX GUI 开发实践在线课程》之 X-Cube-TouchGFX 快速上手视频。

重要通知 – 请仔细阅读

意法半导体公司及其子公司 （“ST”）保留随时对 ST 产品和 / 或本文档进行变更的权利，恕不另行通知。买方在订货之前应获取关于 ST 产品的最新信息。 ST 产品的销售依照订单确认时的相关 ST 销售条款。

买方自行负责对 ST 产品的选择和使用， ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的 ST 产品如有不同于此处提供的信息的规定，将导致 ST 针对该产品授予的任何保证失效。

ST 和 ST 徽标是 ST 的商标。若需 ST 商标的更多信息，请参考 *www.st.com/trademarks*。所有其他产品或服务名称均为其 各自所有者的财产。

本文档是 ST 中国本地团队的技术性文章，旨在交流与分享，并期望借此给予客户产品应用上足够的帮助或提醒。若文中内容存有局限或与 ST 官网资料不一致，请以实际应用验证结果和 ST 官网最新发布的内容为准。您拥有完全自主权是否采纳本文档（包括代码，电路图等）信息，我们也不承担因使用或采纳本文档内容而导致的任何风险。

本文档中的信息取代本文档所有早期版本中提供的信息。