

基于STM32CubeMX的GUI应用开发 ——TouchGFX版

Feb 19, 2019

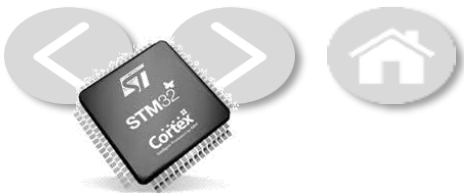
V0.3



- 背景知识简述
- 演示例
 - 开发环境 & 流程
 - 组件构成
 - 组件参数来源分析及配置
 - TouchGFX Designer界面介绍
 - GUI简单示例
 - 文件结构
 - 显示、触摸驱动实现
 - 工程编译&运行
- 扩展
 - TouchGFX应用例
- 资源&参考



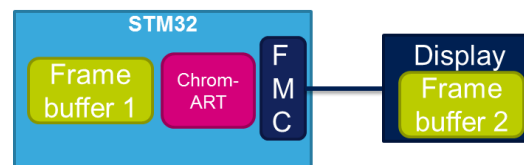
背景知识简述



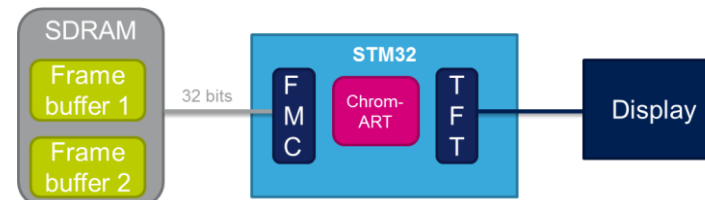
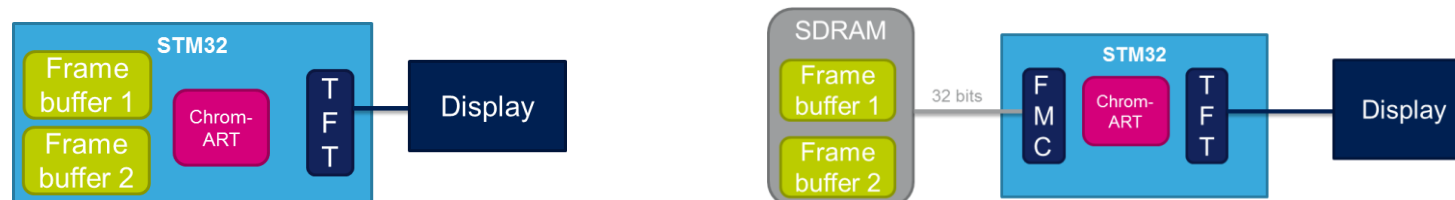
先进的MCU图形产品线

支持多种图形显示接口

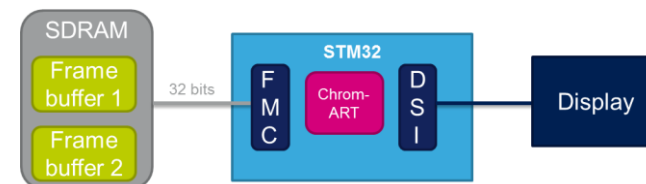
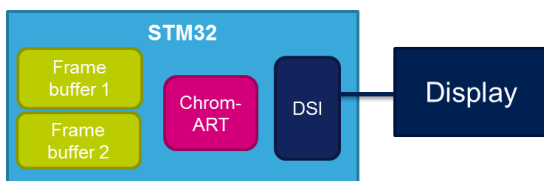
- 小分辨率: Intel 8080 and Motorola 6800 LCD



- 中等分辨率(高达XGA)和MIP低功耗屏:TFT控制器



- 中等分辨率,高像素GUI:MIPI-DSI接口





STM32在GUI应用中的位置

5

Series	Core	Frequency	Graphic features	Display controllers	Supported resolutions
STM32F1	Cortex-M3	72 MHz	-	8080/6800 parallel IF	Up to CGA (320x200) / QVGA (320x240)
STM32F2	Cortex-M3	120 MHz	-		Up to QVGA (320x240) / WQVGA (272x480)
STM32L4	Cortex-M4	80 MHz	Chrom-ART accelerator™		Up to WQVGA (272x480) or 450x450 round displays
STM32L4+	Cortex-M4	120 MHz	Chrom-ART accelerator™, Chrom-GRC™	8080/6800 parallel IF, LCD-TFT controller, MIPI-DSI controller	Up to QVGA (320x240) / WQVGA (272x480)
STM32F4 access and foundation lines	Cortex-M4	100 and 180 MHz	-	8080/6800 parallel IF	Up to XGA (1024x768)
STM32F4 advanced lines	Cortex-M4	180 MHz	Chrom-ART accelerator™	8080/6800 parallel IF, LCD-TFT controller, MIPI-DSI controller	
STM32F7	Cortex-M7	216 MHz	Chrom-ART accelerator™, JPEG Codec	8080/6800 parallel IF, LCD-TFT controller, MIPI-DSI controller	
STM32H7	Cortex-M7	400 MHz			

GUI开发流程（TouchGFX实现）

6

- TouchGFX提供了可视化GUI开发方法。方便快捷实现GUI效果。
- STM32CubeMX提供了软件工程的生成能力，开发者能够根据目标板，选择相应的功能和配置。生成自定义平台的软件工程，快速搭建GUI应用开发环境。



利用TouchGFX Designer将想法快速呈现。



所见即所得，支持拖拽方式，快速容易地完成设计。



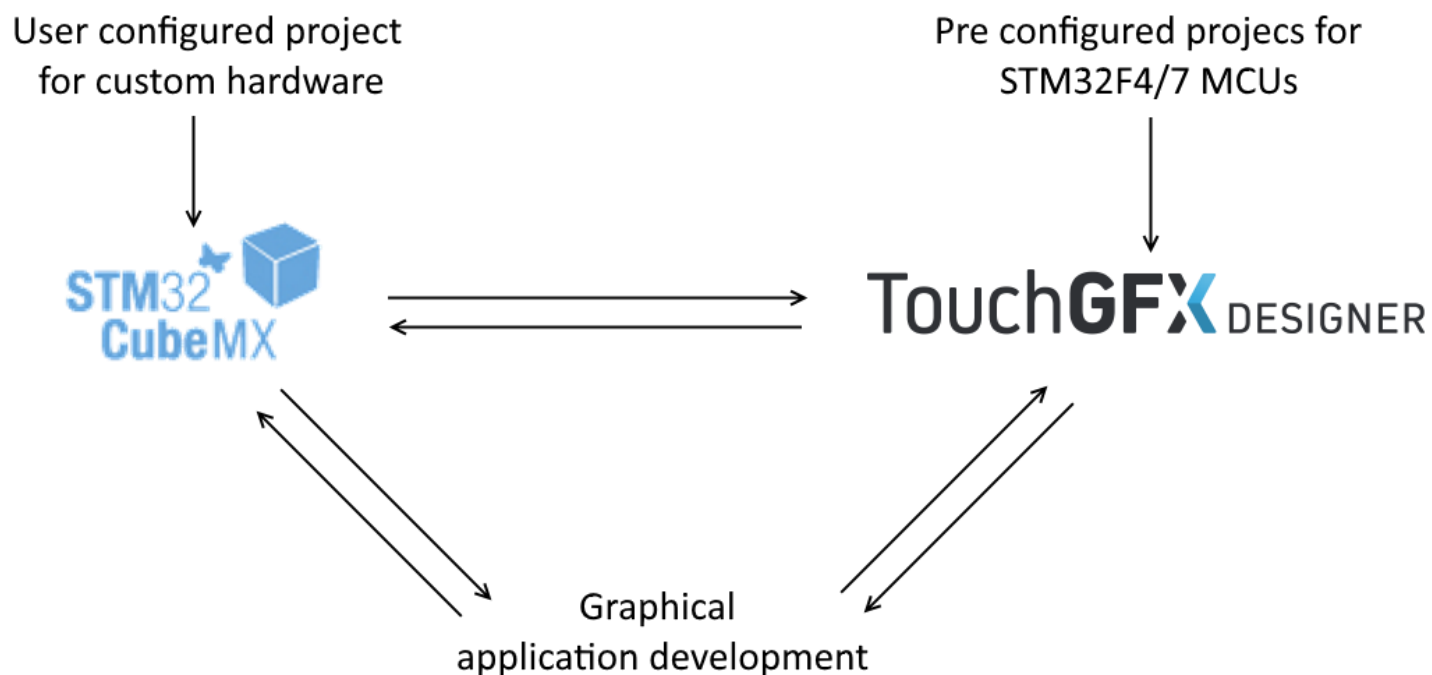
一键生成应用源码，并关联至软件工程中，快速观看在目标板上的显示效果。



STM32CubeMX和TouchGFX联合开发

7

- STM32CubeMX和TouchGFX的联合应用，可以使开发者快速完成GUI应用平台的搭建和GUI应用设计及实现。
- 开发框架如下所示。



- STM32CubeMX v5.0.0 对于GUI中间件的支持情况，如下表所示。表中仅罗列了包含LTDC的STM32系列。
- 另外，TouchGFX可以在ST和TouchGFX提供的示例基础上，进行针对目标平台的修改，实现GUI应用平台的搭建。

GUI MW support list		
STM32CubeMX版本	5.0.0	
注： 罗列了包含LTDC的STM32系列		
	STemWin	TouchGFX
STM32L4R7	Y	N
STM32L4R9	Y	N
STM32L4S7	Y	N
STM32L4S9	Y	N
STM32F429	Y	Y
STM32F439	Y	Y
STM32F469	Y	Y
STM32F479	Y	Y
STM32F746	Y	Y
STM32F750	Y	Y
STM32F756	Y	Y
STM32F767	Y	Y
STM32F769	Y	Y
STM32F777	Y	Y
STM32F778	Y	Y
STM32F779	Y	Y
STM32H743	N	N
STM32H750	N	N
STM32H753	N	N



演示例（包含参数详解）



• 准备工作

- 演示例中采用STM32F769I-Disco作为硬件平台。

需求分析	开发工具	开发参考资料
☐ 480x800 DSI屏 ☐ GUI应用(TouchGFX) ☐ ...	☐ STM32F769I-DISCO板 ☐ STM32CubeProg ☐ STM32CubeMX v5.0.0 ☐ EWARM v8.30.1 ☐ TouchGFX-4.10.0 ☐	☐ 原理图 ☐ 参考手册 & ST技术文档 ☐ 液晶屏手册(OTM8009A) ☐ 触摸控制器手册(FT6026) ☐ 外扩SDRAM数据手册(MT48LC4M32B2) ☐ STM32CubeF7库

- 其中，在STM32CubeMX中，采用STM32CubeF7 v1.14.0
- STM32CubeMX支持如下工具链，在本文中采用EWARM v8.30.1



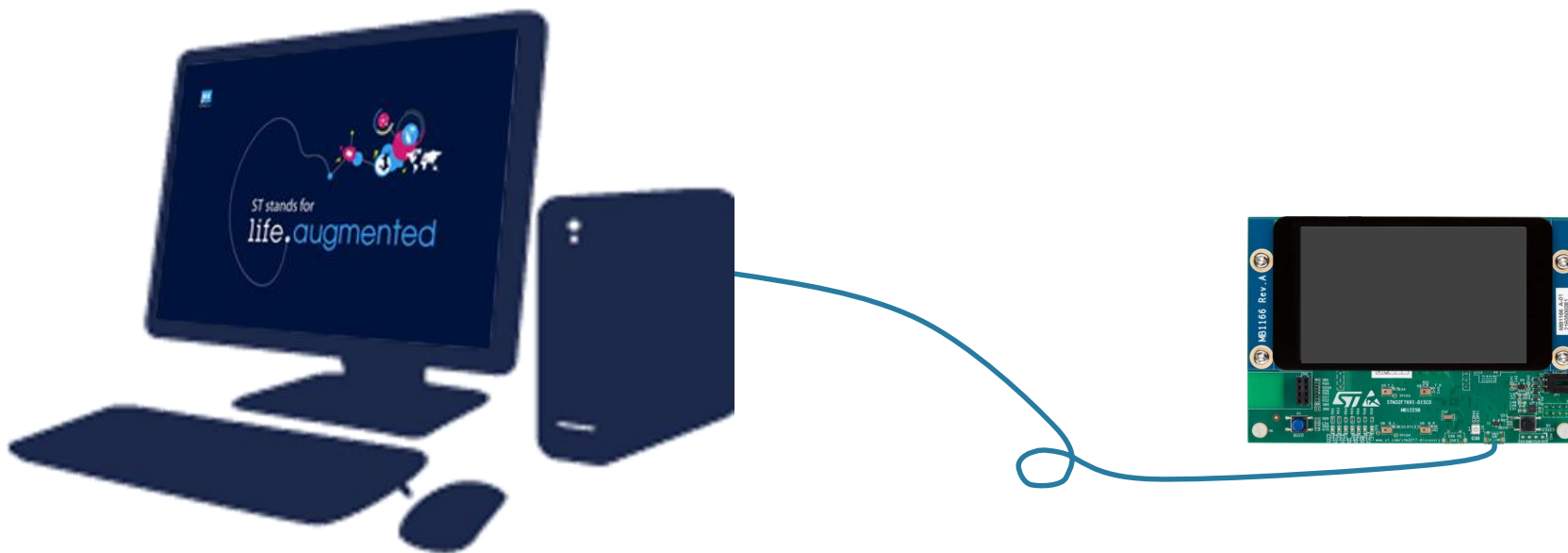


- 主机

- 安装STM32CubeMX 5.0 或以上版本
- 安装STM32CubeProgrammer或者ST-Link Utility
- 安装对应的IDE (Eg. IAR/KEIL)
- 在STM32CubeMX中载入STM32Cube软件包

- GUI板

- STM32F769I-DISCO (演示用)
 - STM32F769NIH6U
 - 480x800 MIPI屏 (带触摸面板)
 - 外扩SDRAM



- STM32F769I-DISCO推荐获取途径



STM32F769I-DISCO
ST 意法原装正品开...

¥681



使用淘宝扫一扫



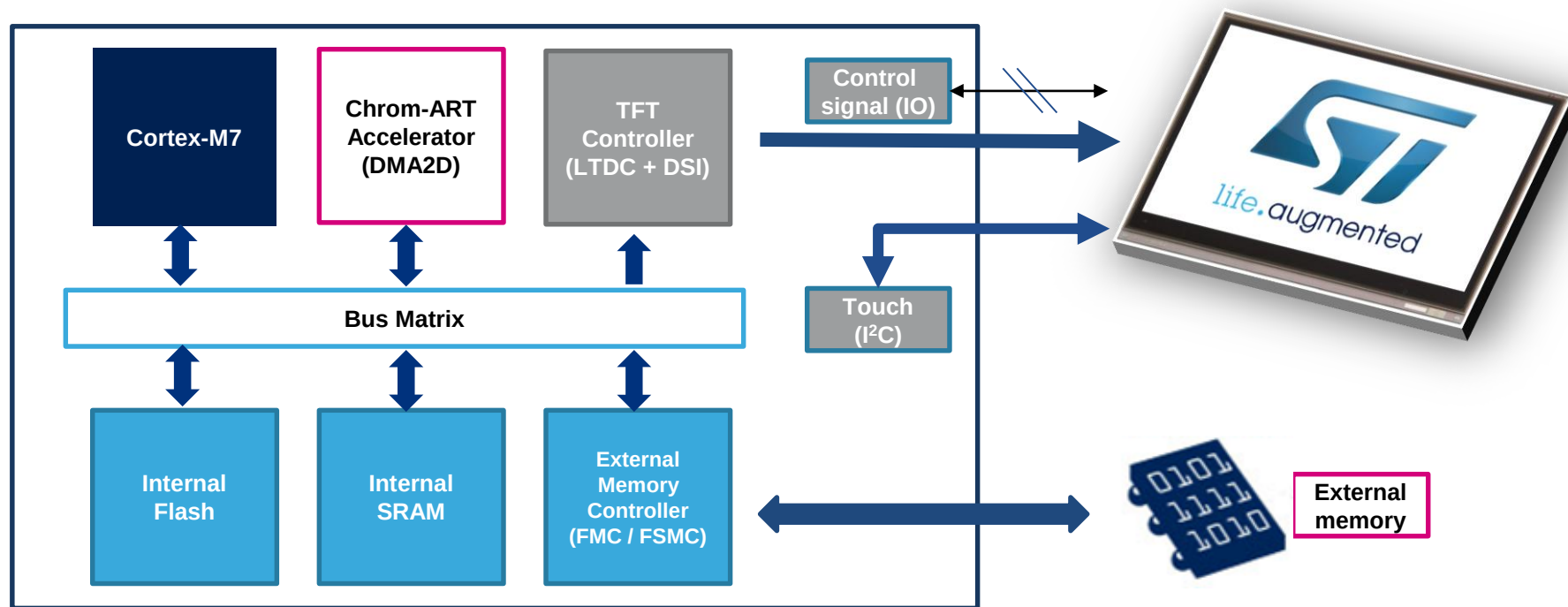
保存图片
到相册



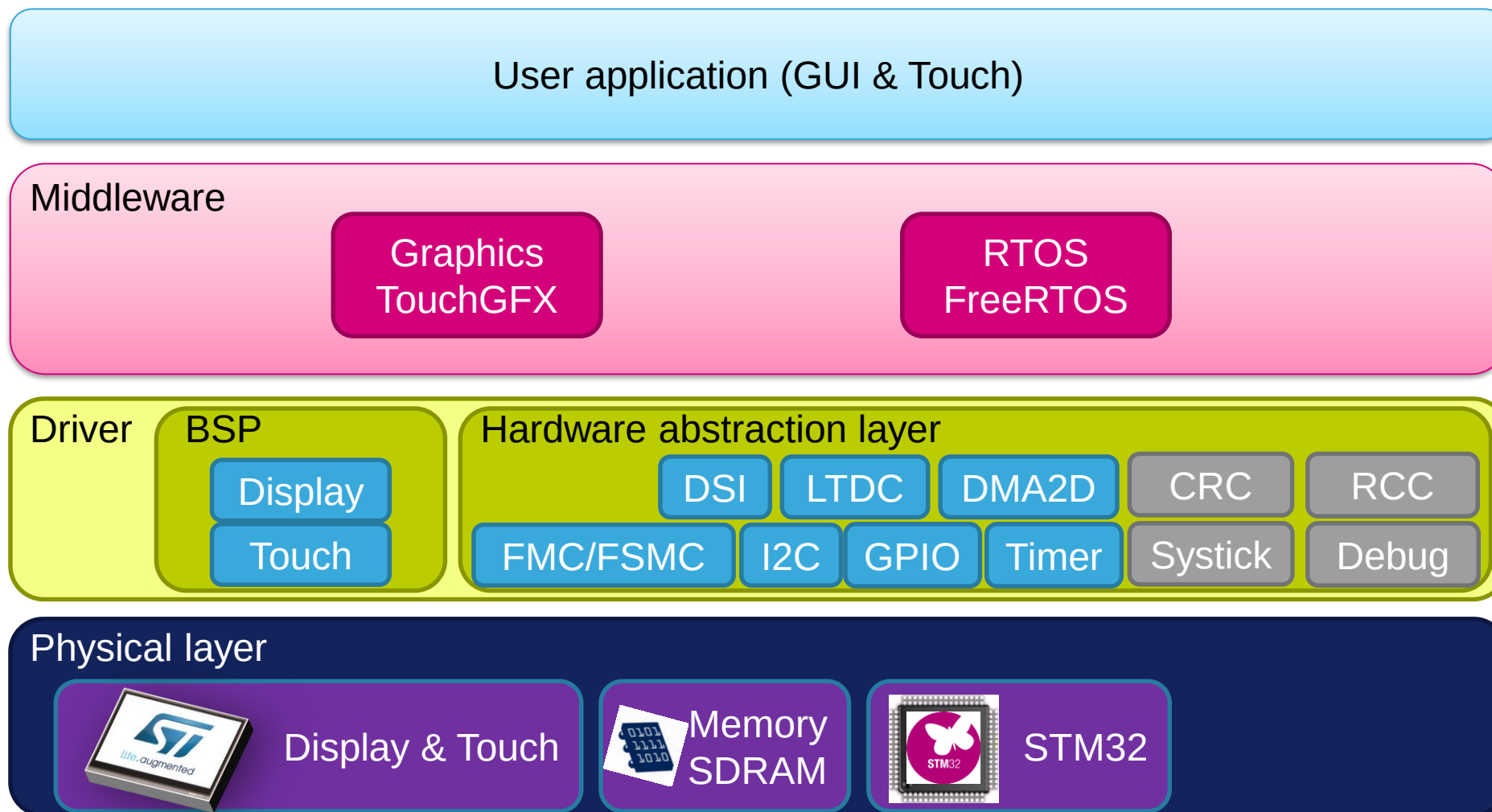
打开淘宝
立即看见

本次分享30天内有效

- 应用框架



- 应用框架



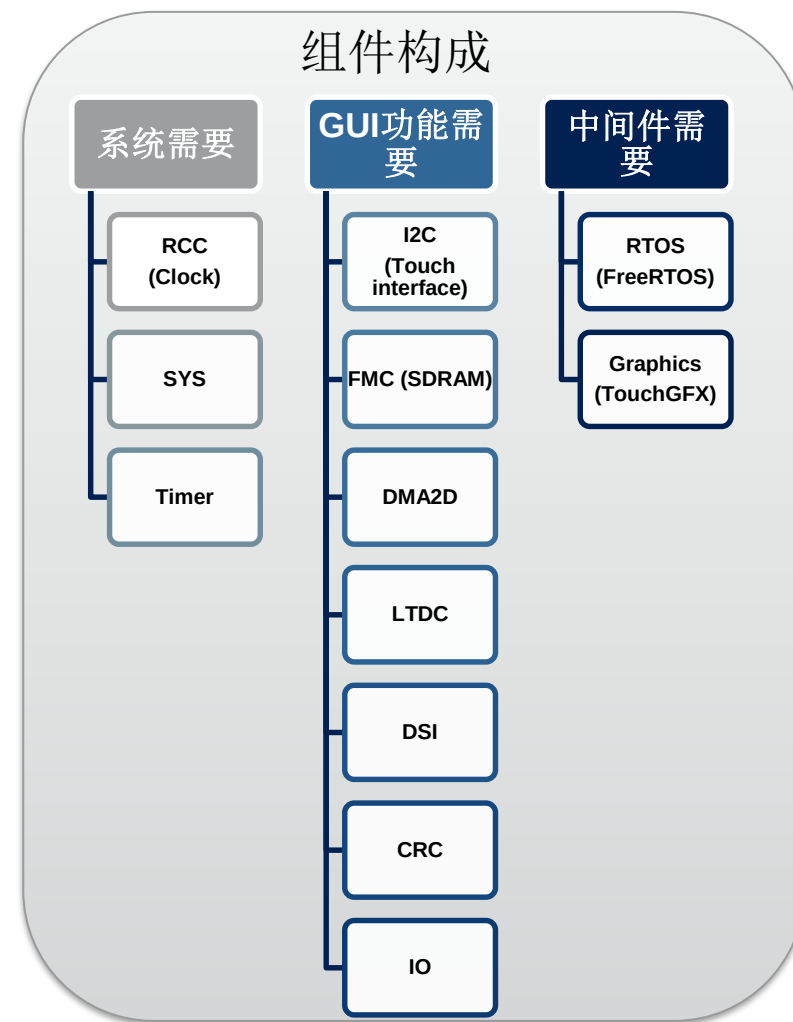


实现示例（续）

15

- IP组件（系统需要）

- 利用STM32CubeMX开发TouchGFX应用，在STM32CubeMX中的主要工作，集中在对使用的IP组件，根据应用需要、硬件连接、器件支持特性等进行配置。
- 演示例中，以STM32F769I-DISCO板作为目标板，需要用到组件构成如右图所示。





- 演示例中，开发流程如下所示。
- STM32CubeMX覆盖了红色框中部分。

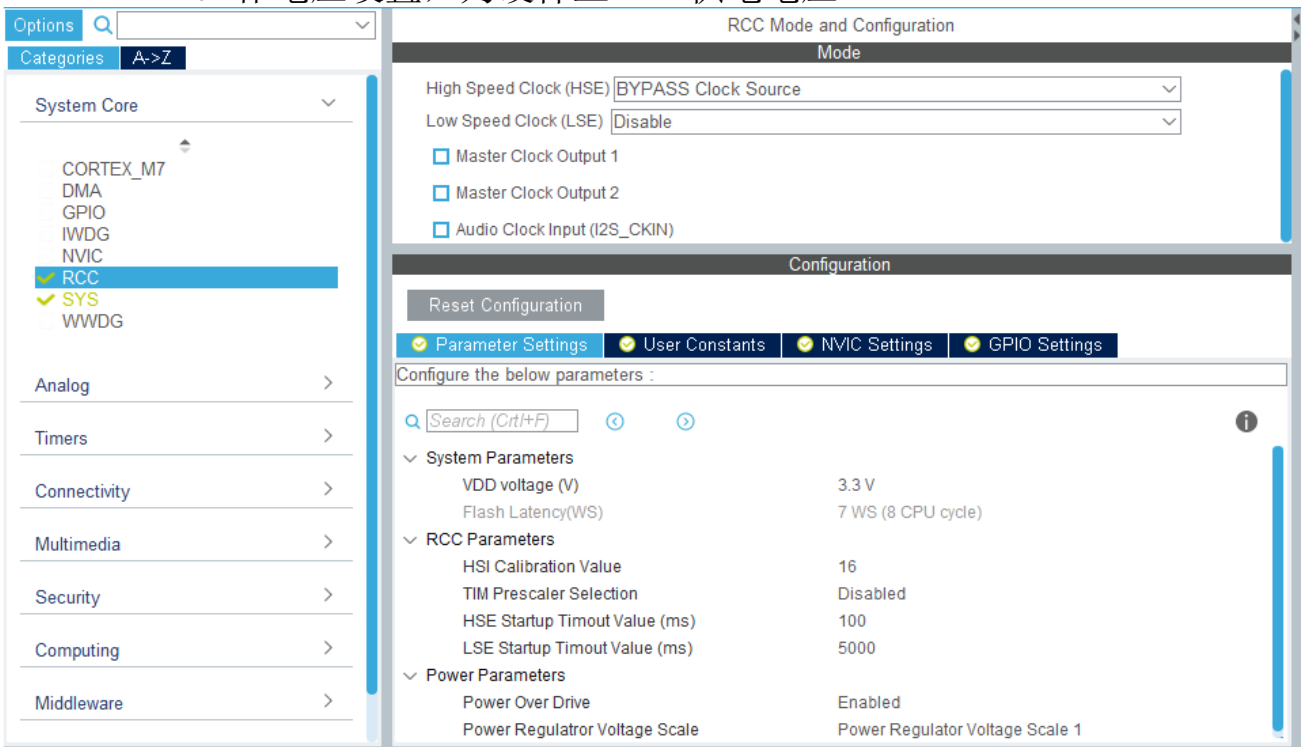




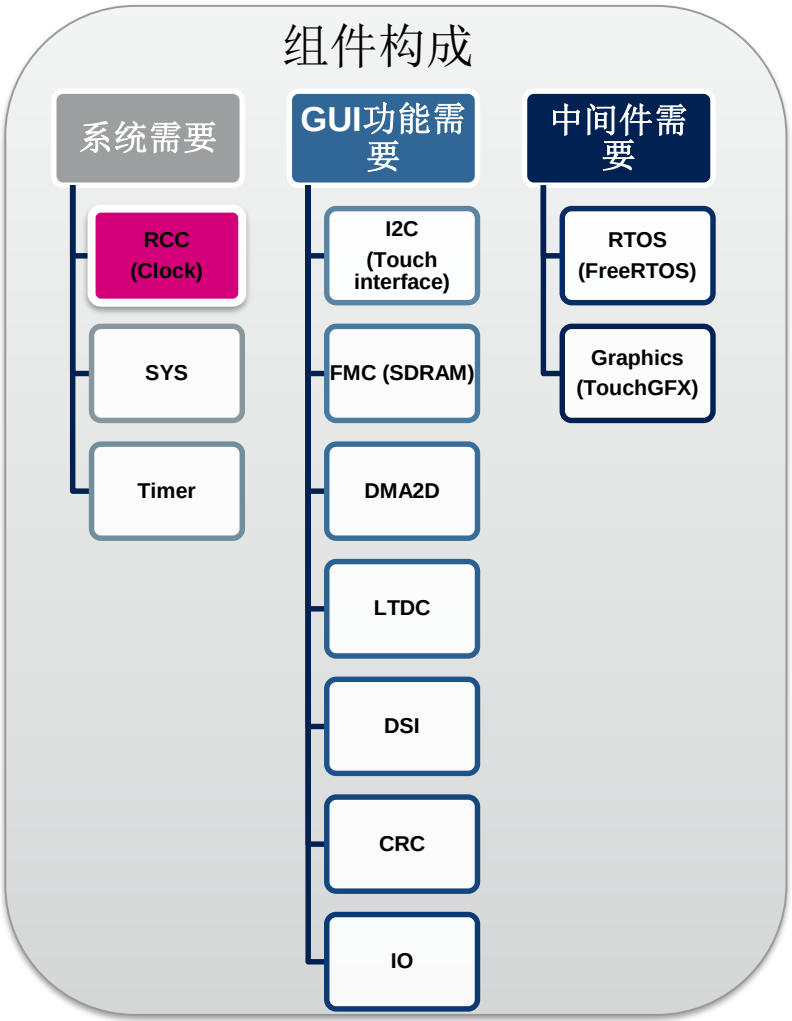
• 组件（系统需要）

• RCC

- 时钟源。根据应用选择。
 - 采用内部时钟，不用选择；
 - 采用外部高速无源时钟，选择BYPASS Clock Source；
 - 采用外部高速有源时钟，选择Crystal/Ceramic Resonator
- VDD。工作电压设置，为硬件上VDD供电电压。



组件构成





实现示例（续）

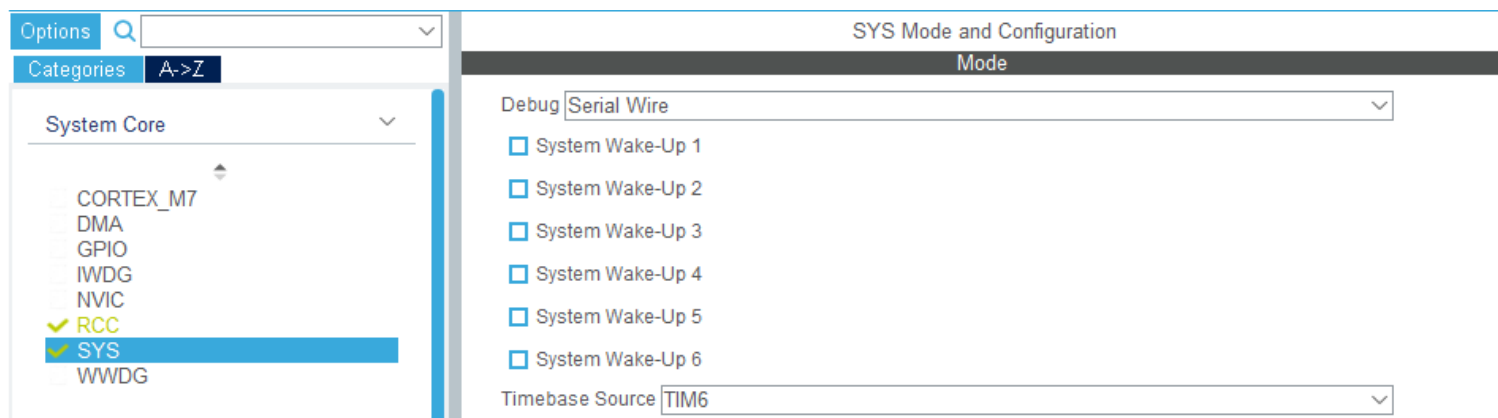
18



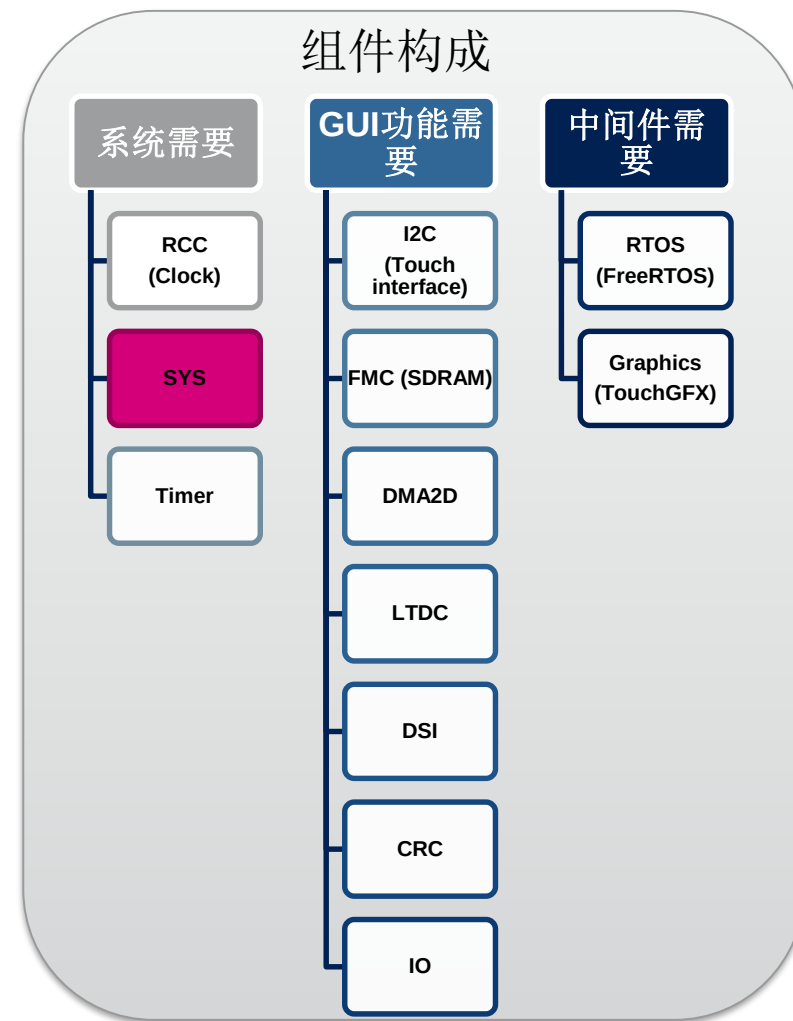
• 组件（系统需要）

• SYS

- Debug
 - 调试接口。
- Timebase source
 - HAL库延时函数HAL_Delay关联的定时器



组件构成



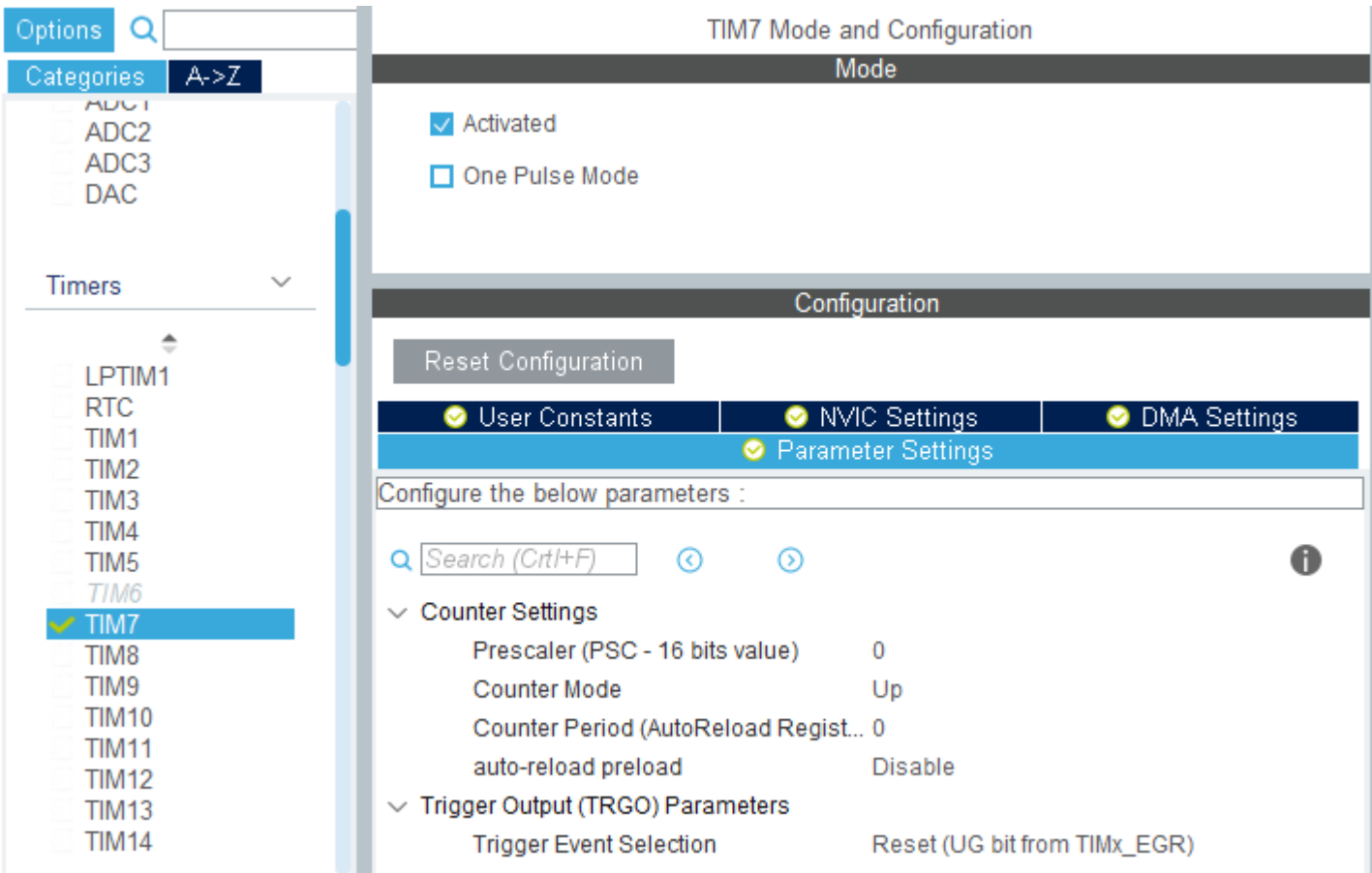


实现示例（续）

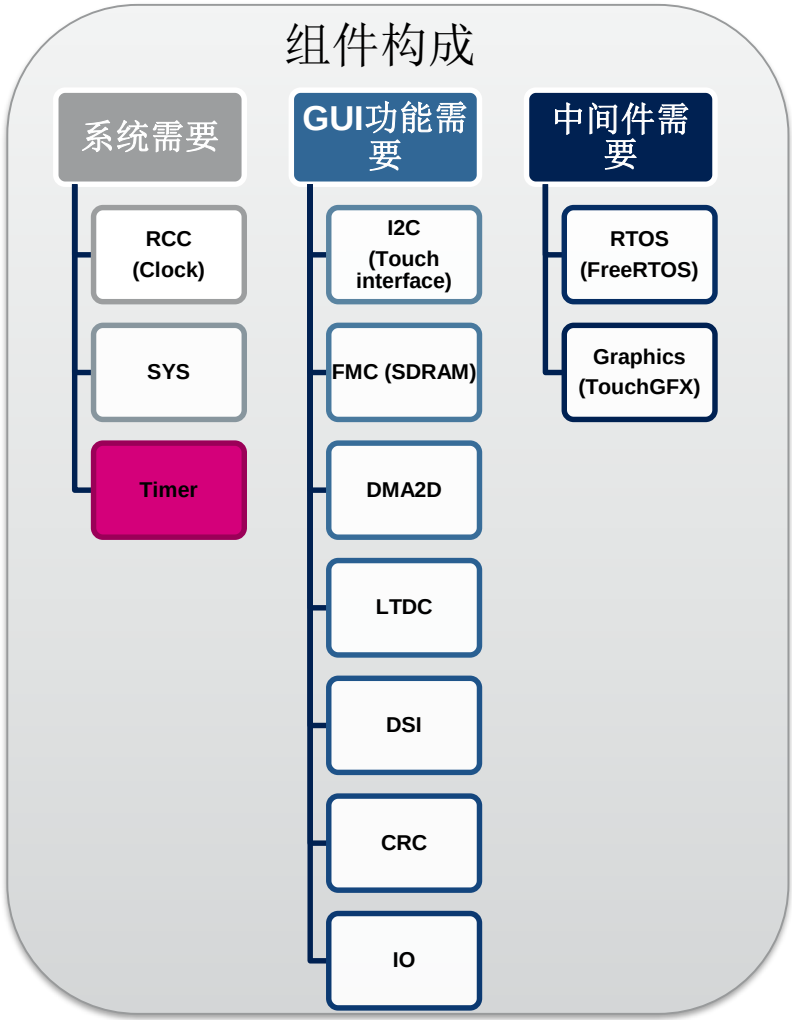


• 组件（系统需要）

• Timer



组件构成

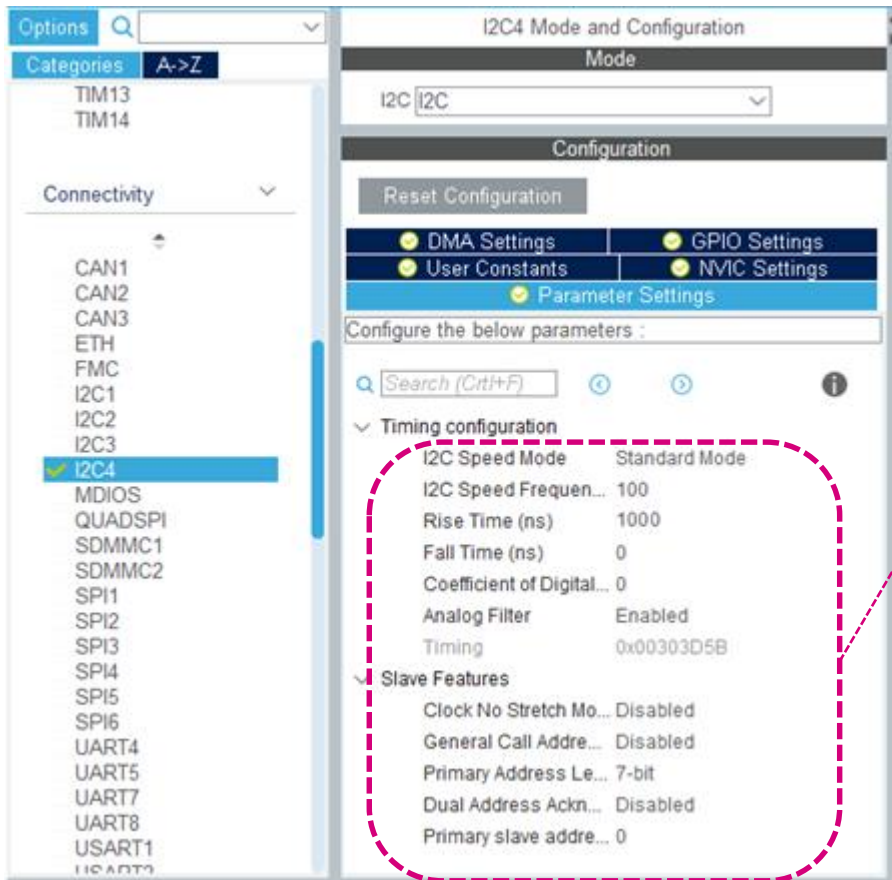




• 组件（GUI功能需要）

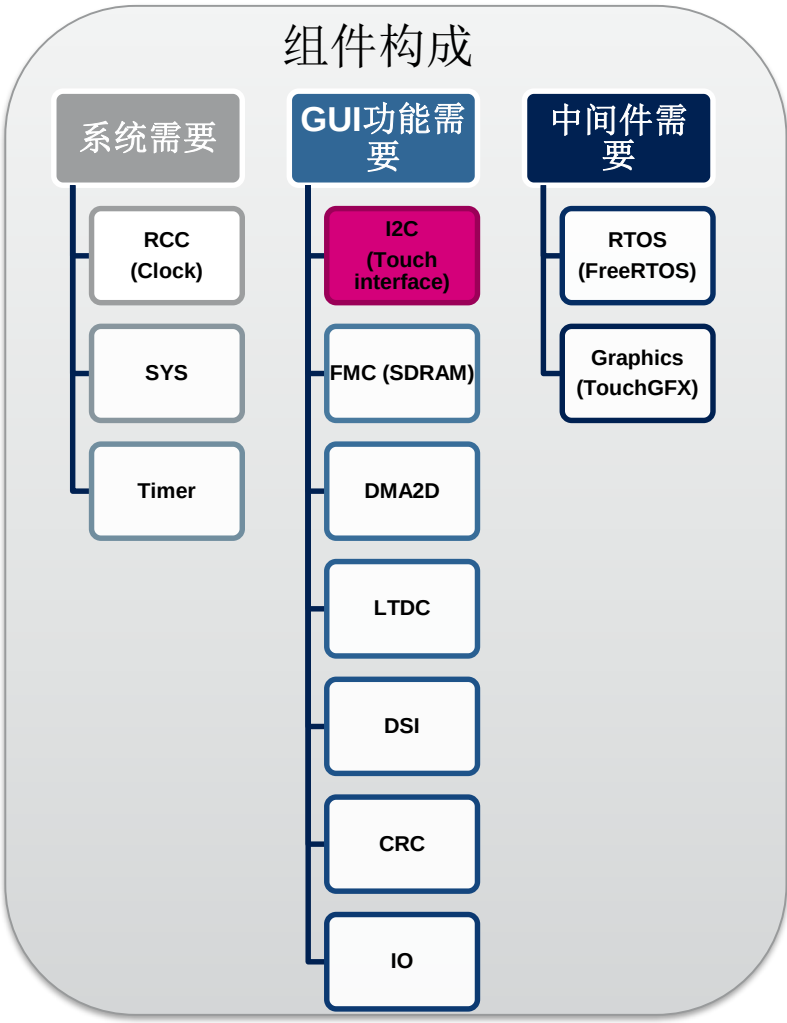
• I2C

- 触摸芯片访问接口（在STM32F769I-DISCO上，与I2C4连接）
- 选择&配置I2C，如下图



- 根据触摸板控制器I2C接口参数要求，完成参数配置。
- 左图参数对应为演示例中采用的触摸控制器FT6026

组件构成





• 组件（GUI功能需要）

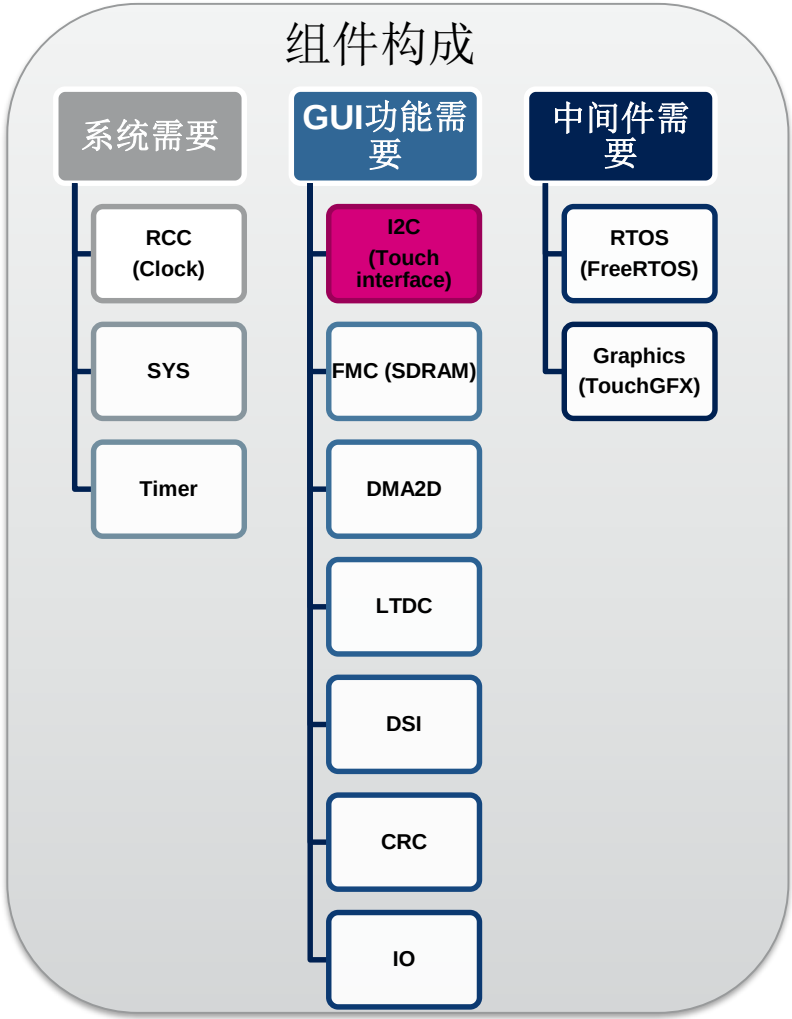
• I2C

- 由于STM32 外设为了灵活性，提供了不同引脚实现同一个外设功能。在应用时，根据硬件连接，选择具体关联引脚
- 默认关联的I2C接口引脚与实际硬件采用可能不同，检查&调整

表. 触摸面板接口 连接情况

功能	名称	引脚				描述
		引脚号	模式	上下拉	速度	
触摸屏 (FT 6206)	LCD_INT	PI13	IT_FALLING	PULL_UP	FAST	触摸触发中断信号
	LCD_SDA	PB7	AF11_I2C4	NOPULL	FAST	I2C数据线
	LCD_SCL	PD12	AF4_I2C4	NOPULL	FAST	I2C时钟线

注：演示例中，采用轮询的方式采集触摸坐标信息，没有配置和使用 LCD_INT功能。





实现示例（续）

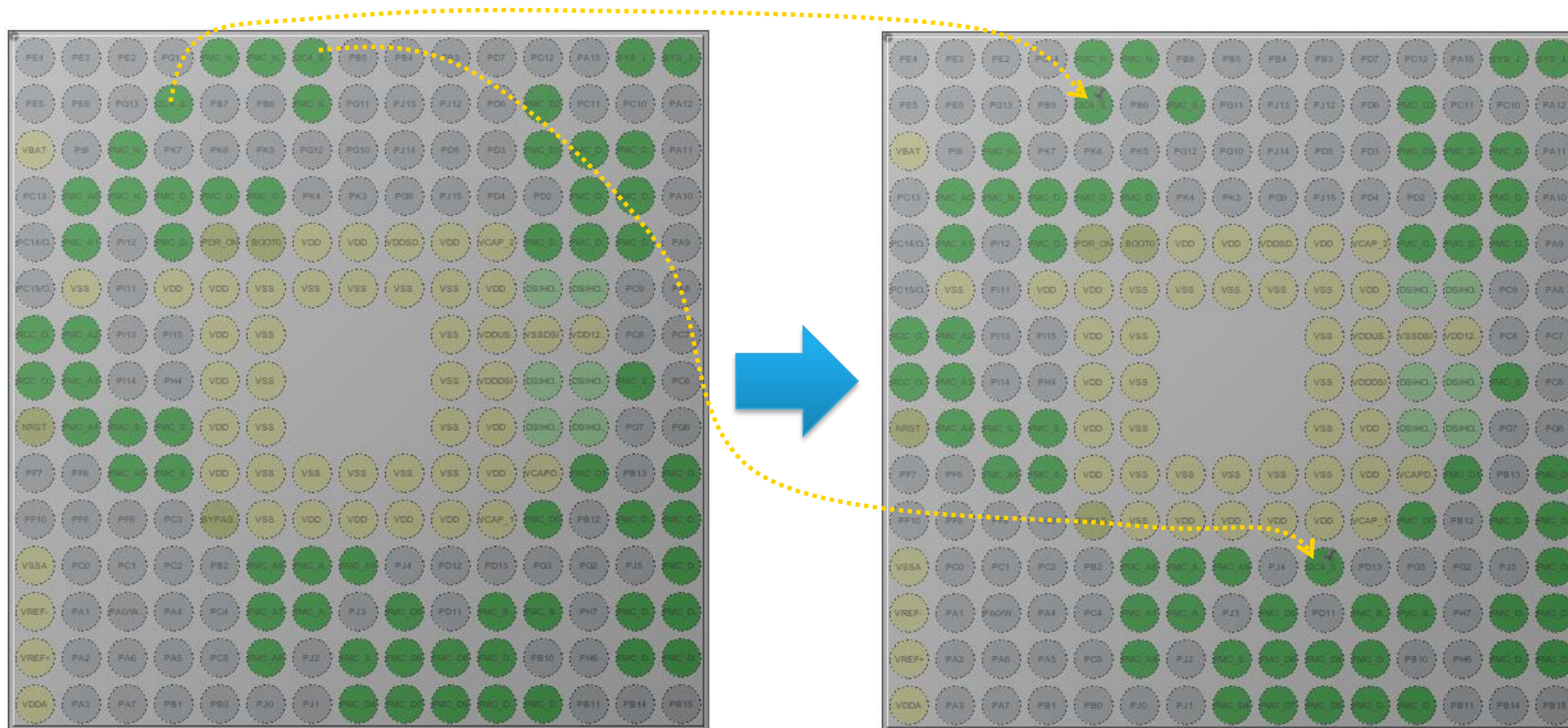
22



• 组件（GUI功能需要）

• I2C

- 默认I2C4关联引脚与实际硬件连接不一致，调整与硬件连接保持一致。





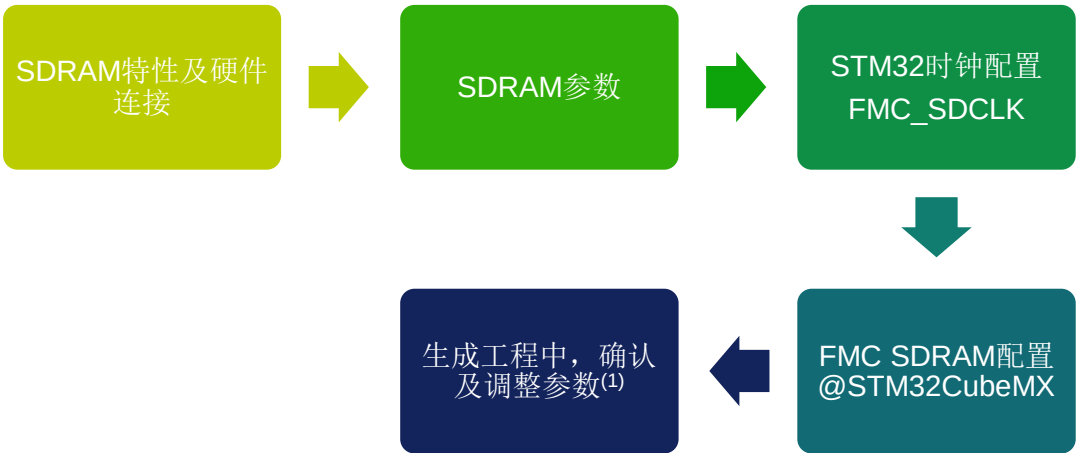
实现示例（续）

23

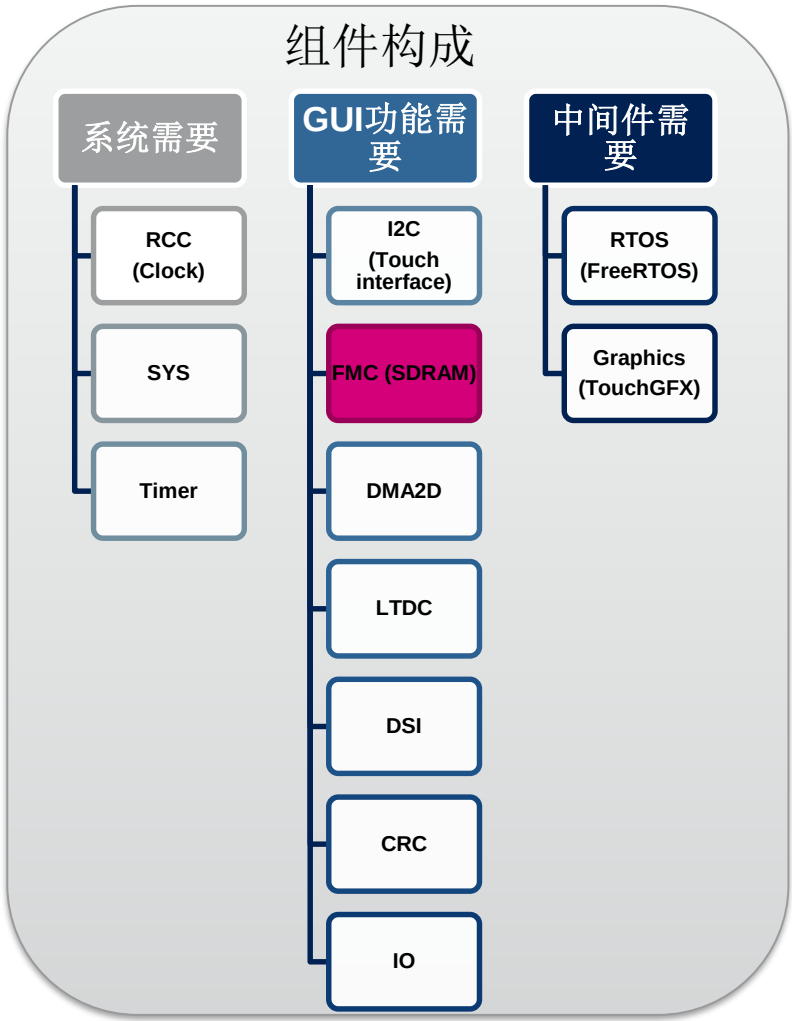
• 组件（GUI功能需要）

• FMC（外部存储器）

- 不同的存储器、硬件连接和时钟配置，对应FMC配置参数不同。
- 首先，呈现在STM32CubeMX上FMC配置。可以了解FMC主要的一些配置项。
- 然后，以STM32F769I-Disco板上的SDRAM (型号MT48LC4M32B2B5-6A)为例，介绍如何根据实际情况，获取配置参数。



FMC配置流程图（以演示例中采用的SDRAM为例）





• 组件（GUI功能需要）

• FMC（外部存储器）

• FMC配置（STM32CubeMX中）

FMC Mode and Configuration

Mode

- > NOR Flash/PSRAM/SRAM/ROM/LCD 1
- > NOR Flash/PSRAM/SRAM/ROM/LCD 2
- > NOR Flash/PSRAM/SRAM/ROM/LCD 3
- > NOR Flash/PSRAM/SRAM/ROM/LCD 4
- > NAND Flash 1
- ✓ SDRAM 1
- > ⚠ SDRAM 2

Clock and chip enable: SDCKE0+SDNE0

Internal bank number: 4 banks

Address: 12 bits (Max: 13 bits)

Data: 32 bits

Byte enable: 32-bit byte enable

Configuration

Reset Configuration

✓ NVIC Settings | ✓ GPIO Settings

✓ SDRAM 1 | ✓ User Constants

Configure the below parameters:

Search (Ctrl+F)

✓ SDRAM control

Bank	SDRAM bank 1
Number of column addresses...	8 bits
Number of row address bits	12 bits
CAS latency	3 memory clock cycles
Write protection	Disabled
SDRAM common clock	2 HCLK clock cycles
SDRAM common burst read	Enabled
SDRAM common read pip...	0 HCLK clock cycle

✓ SDRAM timing in memory clock ...

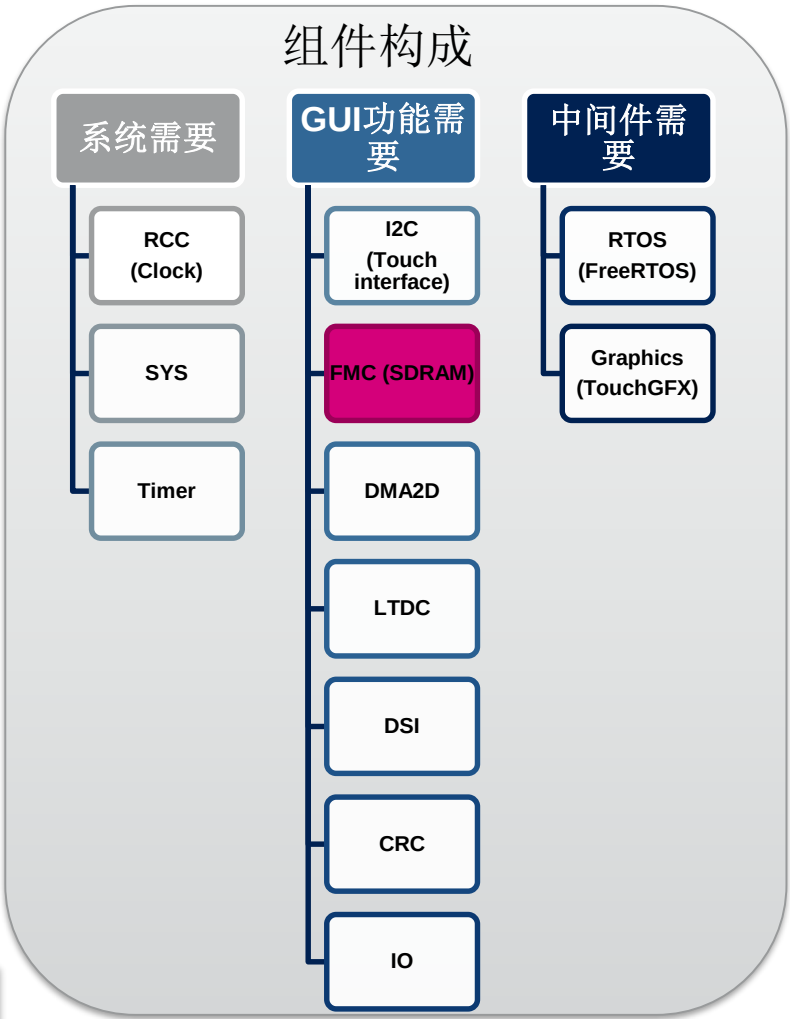
Parameter	Value
Load mode register to activ...	2
Exit self-refresh delay	7
Self-refresh time	4
SDRAM common row cycle...	7
Write recovery time	3
SDRAM common row prec...	2
Row to column delay	2

✓ NVIC Settings | ✓ GPIO Settings

✓ SDRAM 1 | ✓ User Constants

	Enabled	Preemption Priority	Sub Priority
NVIC Interrupt Table			
FMC global interrupt	✓	0	0

组件构成





• 组件（GUI功能需要）

• FMC（外部存储器）

• 配置参数获取

在进行FMC的配置前，需要先了解与FMC关联的SDRAM硬件连接情况。

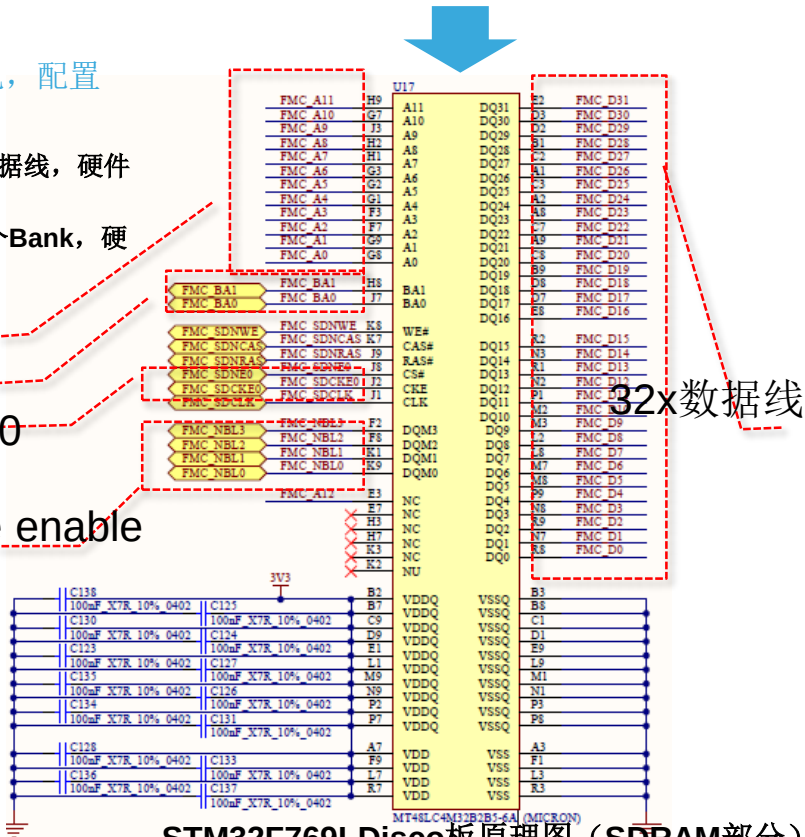
Parameter	4 Meg x 32
Configuration	1 Meg x 32 x 4 banks
Refresh count	4K
Row addressing	4K A[11:0]
Bank addressing	4 BA[1:0]
Column addressing	256 A[7:0]

图 摘自 MT48LC4M32B2数据手册

- 1. 根据SDRAM硬件支持和使用情况，配置SDRAM1

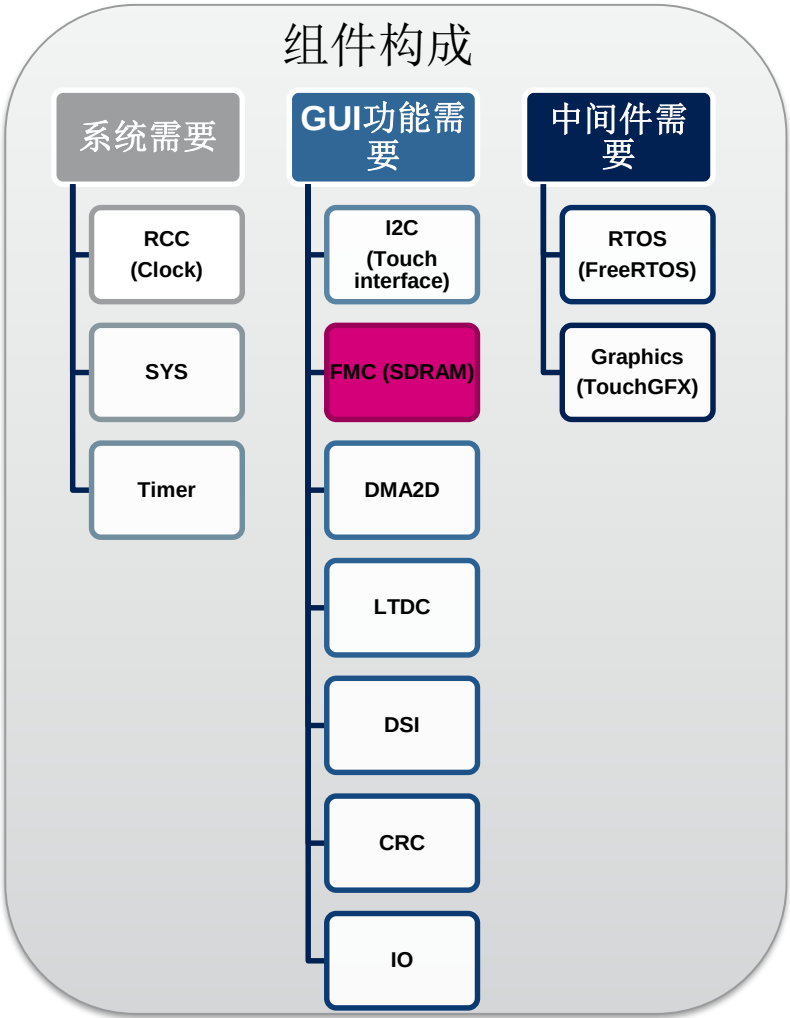
- MT48LC4M32B2B5-6A支持32位数据线，硬件上采用了32位数据线
- MT48LC4M32B2B5-6A内部含有4个Bank，硬件上采用了4个Bank
- ...

12x地址线
4x内部Banks
SDRAM Bank 0



STM32F769I-Disco板原理图（SDRAM部分）

组件构成



类别	MT48LC4M32B2B5-6A支持	硬件采用
SDRAM	-	SDCKE0 SDNE0
Bank	4	4
行地址 (Bit)	12	12
数据线	32	32
字节使能	32bit (4字节)	32bit (4字节)



• 组件（GUI功能需要）

• FMC（外部存储器）

• 配置参数获取

在进行FMC的配置前，需要先了解与FMC关联的SDRAM硬件连接情况。

- 1. 根据SDRAM硬件支持和使用情况，配置SDRAM1
 - MT48LC4M32B2B5-6A支持32位数据线，硬件上采用了32位数据线
 - MT48LC4M32B2B5-6A内部含有4个Bank，硬件上采用了4个Bank
 - ...

类别	MT48LC4M32B2B5-6A支持	硬件采用
SDRAM		SDCKE0+SDNE0
Bank	4	4
行地址 (Bit)	12	12
数据线	32	32
字节使能	32bit (4字节)	32bit (4字节)

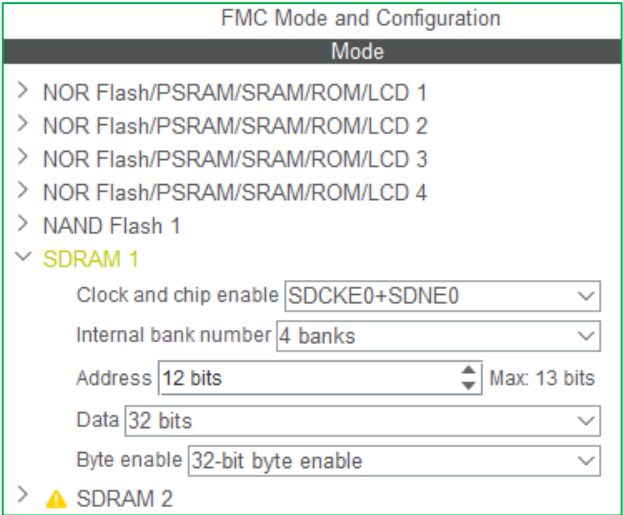
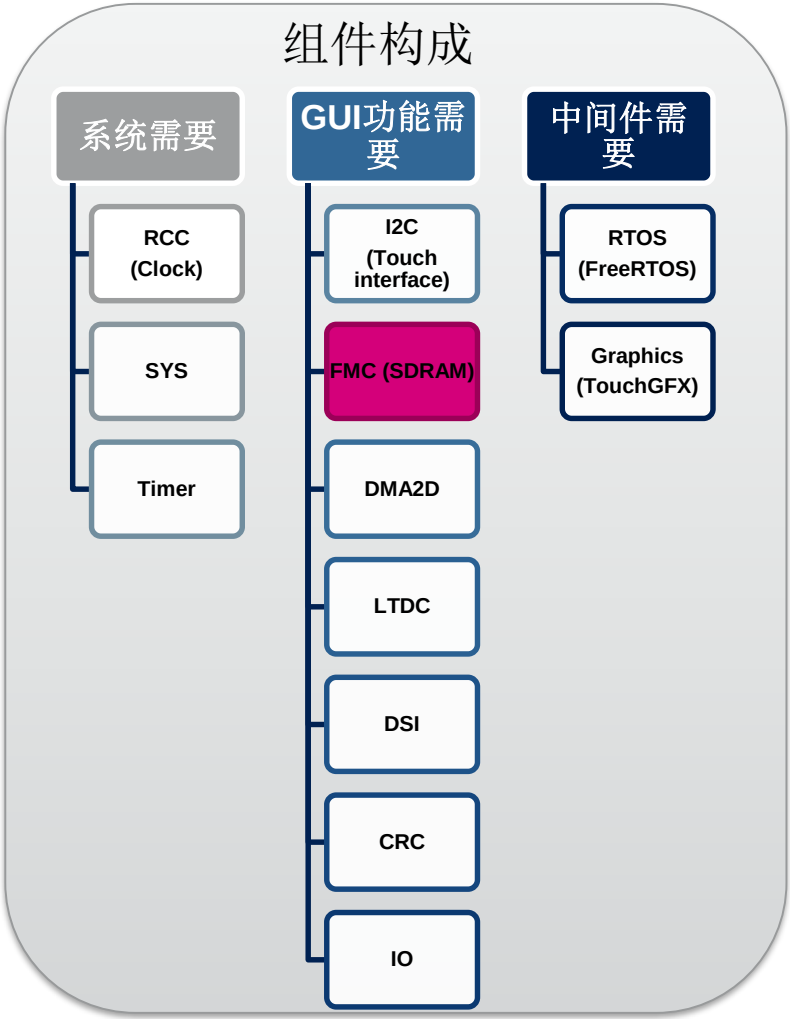


图. FMC SDRAM配置 (@STM32CubeMX)

组件构成





• 组件（GUI功能需要）

• FMC（外部存储器）

• 配置参数获取（接上页）

- 2. 根据硬件情况，确认/调整接口的I/O引脚到硬件对应的复用引脚。

例如，STM32F769NIH6 有三个I/O能复用为 FMC_SDCKE0，分别是PH2, PC3, PC5。硬件电路上，PH2与存储器的CKE连接。

确认FMC_SDCKE0是否被分配到PH2引脚，如果不是，可按照如下步骤更改：

- 1. 查询SDCKE0
- 2. 找到FMC_SDCKE0对应的管脚
- 3. Ctrl + 左键按下，出现可复用管脚
- 4. 拖动到符合硬件连接的PH2

根据硬件连接，确认/调整复用引脚

注：实测自动分配的引脚与STM32F769I-DISCO板采用引脚一致，可以直接跳过这个步骤

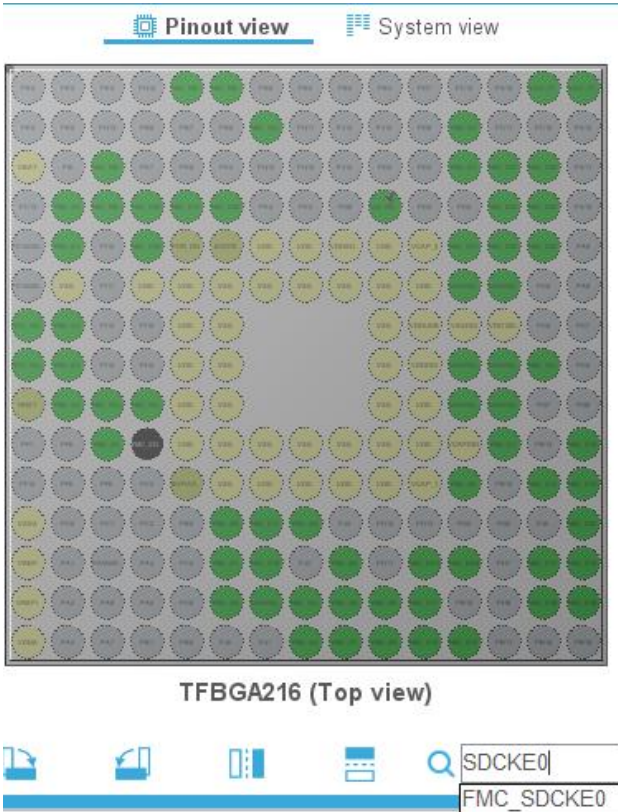
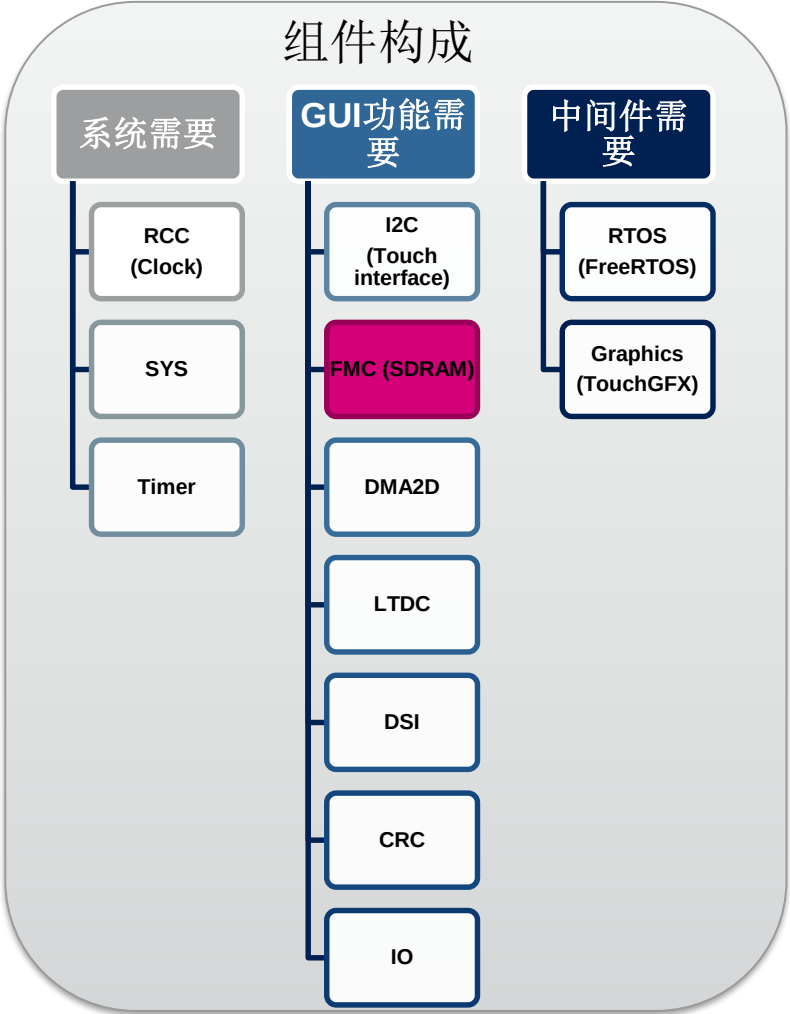


图. 引脚分布
（@STM32CubeMX）





• 组件（GUI功能需要）

• FMC（外部存储器）

• 配置参数获取（接上页）

- 3. 进入SDRAM1标签页
- 4. 根据**SDRAM数据手册**中参数，配置SDRAM控制参数及时序参数

根据SDRAM数据手册中参数及要求

根据应用需要，确定FMC时钟源频率

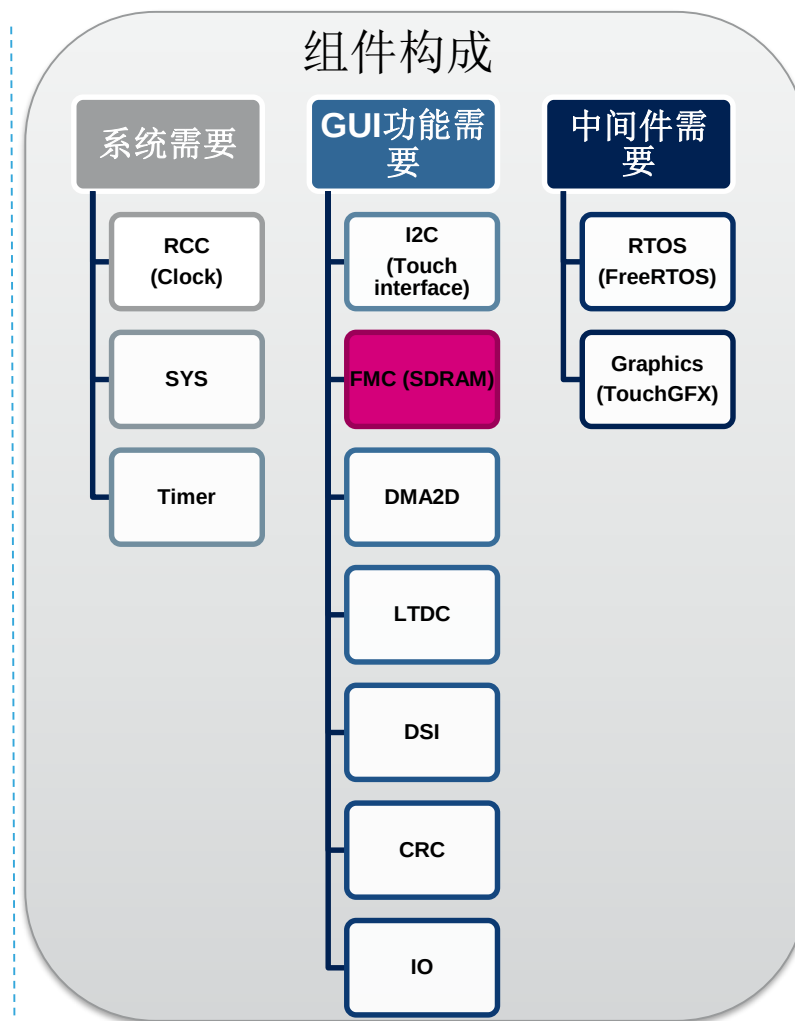


FMC配置参数

在STM32CubeMX中的FMC参数

实现示例（续）

28





• 组件（GUI功能需要）

• FMC（外部存储器）

• 配置参数获取（接上页）

- 3. 进入SDRAM1标签页
- 4. 根据SDRAM数据手册中参数，配置SDRAM控制参数及时序参数

首先，根据存储器数据手册，获取相关的电气参数。如下所示。

（SDRAM型号： **MT48LC4M32B2B5-6A** ）

Parameter	4 Meg x 32
Configuration	1 Meg x 32 x 4 banks
Refresh count	4K
Row addressing	4K A[11:0]
Bank addressing	4 BA[1:0]
Column addressing	256 A[7:0]

- Number of row address bits = 12
- Number of column address bits = 8

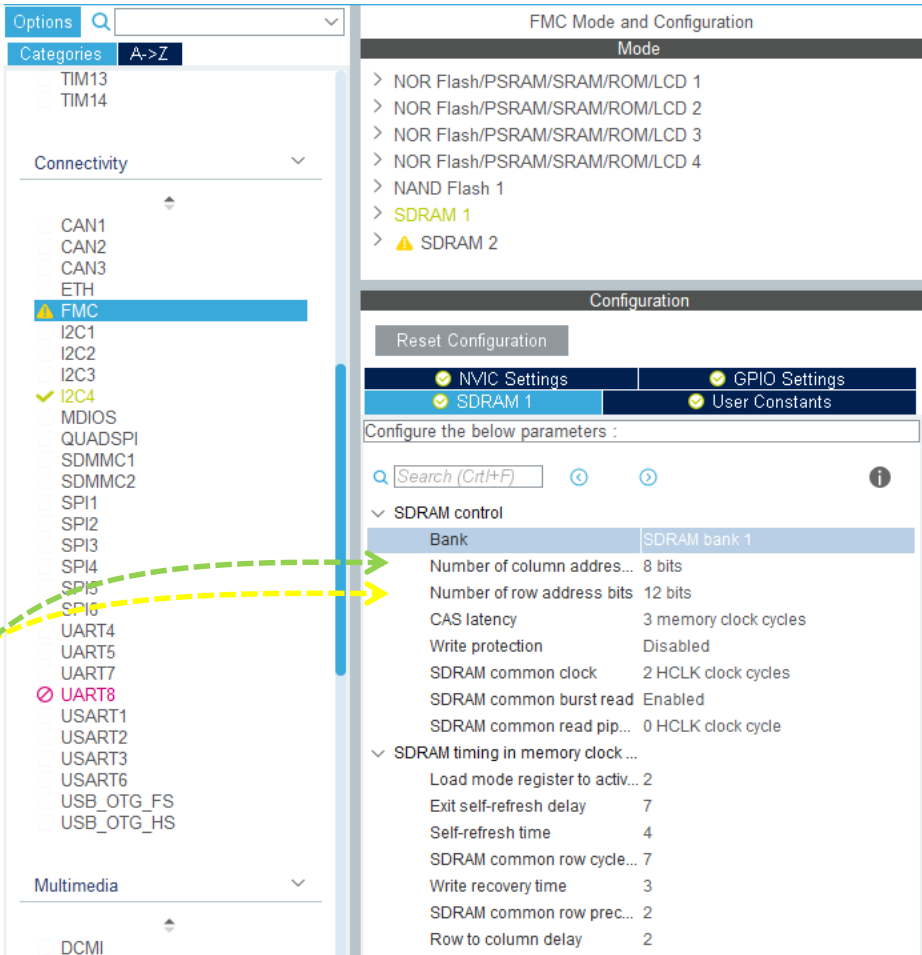


图. FMC SDRAM配置
（@STM32CubeMX）

图 摘自 MT48LC4M32B2数据手册



实现示例（续）

30



• 组件（GUI功能需要）

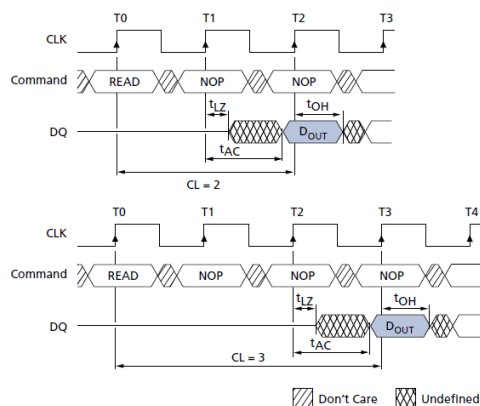
• FMC（外部存储器）

• 配置参数获取（接上页）

- 3. 进入SDRAM1标签页
- 4. 根据SDRAM数据手册中参数，配置SDRAM控制参数及时序参数

首先，根据存储器数据手册，获取相关的电气参数。如下所示。

（SDRAM型号： **MT48LC4M32B2B5-6A** ）



- 根据存储器描述，在CAS latency后，直接可以访问数据。所以SDRAM common read pipe delay(CAS latency后至访问数据的延时周期)为0。



- **MT48LC4M32B2B5-6A**支持Burst read
- **MT48LC4M32B2B5-6A**不支持写保护

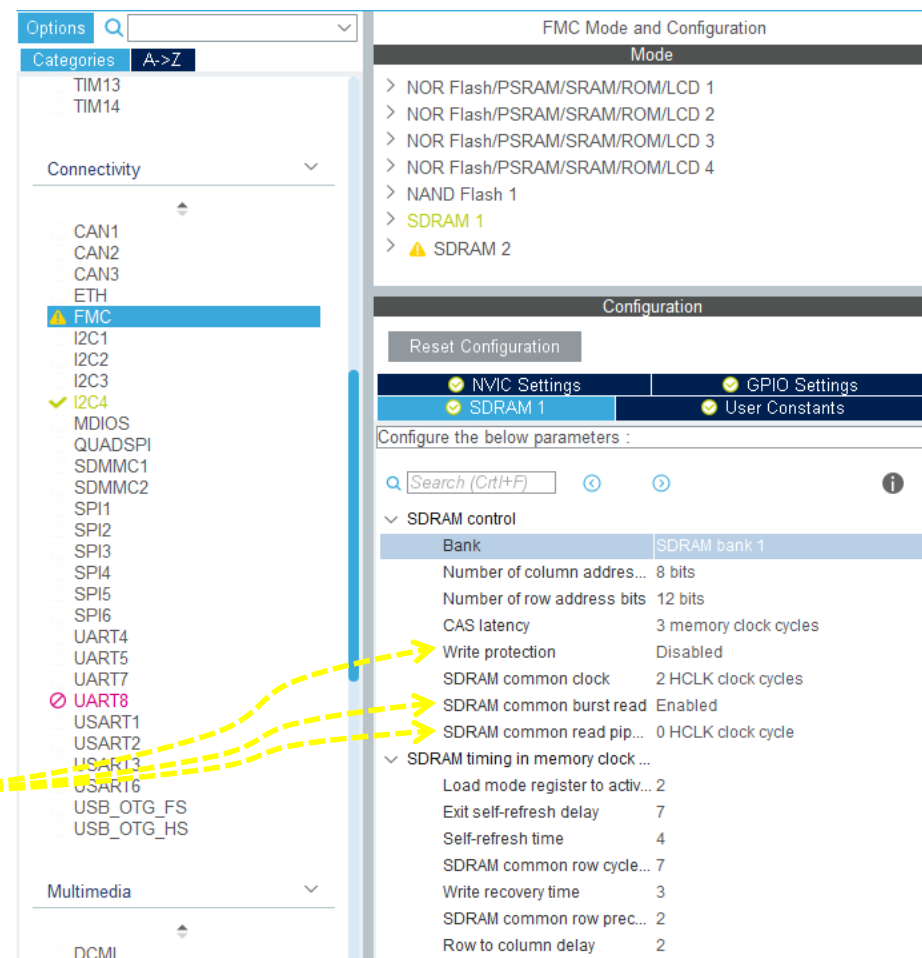


图. FMC SDRAM配置
（@STM32CubeMX）



• 组件（GUI功能需要）

• FMC（外部存储器）

• 配置参数获取（接上页）

- 3. 进入SDRAM1标签页
- 4. 根据SDRAM数据手册中参数，配置SDRAM控制参数及时序参数

首先，根据存储器数据手册，获取相关的电气参数。如下所示。

（SDRAM型号：MT48LC4M32B2B5-6A）

CL = CAS (READ) latency

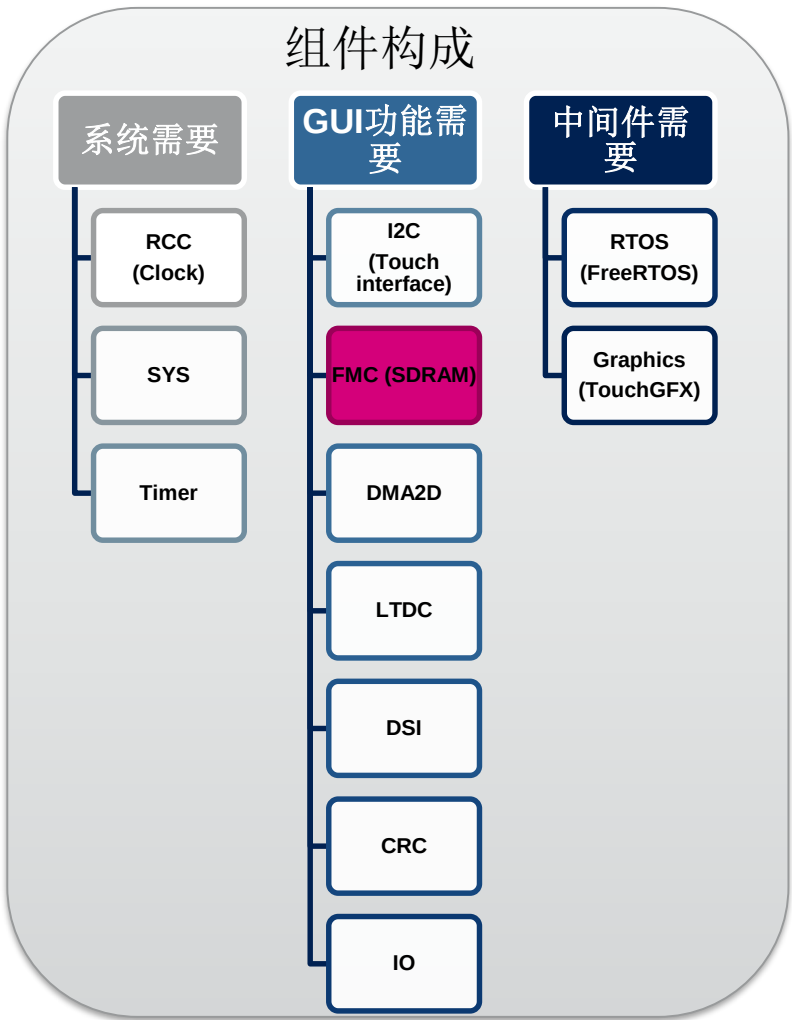
Speed Grade	Clock Frequency (MHz)	Target 'RCD-'RP-CL	'RCD (ns)	'RP (ns)	CL (ns)
-6A	167	3-3-3	18	18	18
-6	167	3-3-3	18	18	18
-7	143	3-3-3	20	20	21

• CAS (READ) latency = 18 ns

Parameter	Symbol	-6A						Unit	Notes
		Min	Max	Min	Max	Min	Max		
Clock cycle time	CL = 3 CL = 2 CL = 1	'CK(3) 'CK(2) 'CK(1)	6 10 20	- - 20	6 10 20	- - 20	- - 20	ns	8
ACTIVE-to-PRECHARGE command	'RAS	42	120.00	42	120.00	42	120.00	ns	
ACTIVE-to-ACTIVE command period	'RC	60	-	60	-	70	-	ns	10
AUTO REFRESH period	'RFC	60	-	60	-	70	-	ns	
ACTIVE-to-READ or WRITE delay	'RCD	18	-	18	-	20	-	ns	
Refresh period (4096 rows)	'REF	-	64	-	64	-	64	ms	
Refresh period - automotive (4096 rows)	'REFAT	-	16	-	16	-	16	ms	
PRECHARGE command period	'RP	18	-	18	-	20	-	ns	
ACTIVE bank a to ACTIVE bank b command	'RRD	12	-	12	-	15	-	ns	11
Transition time	'T	0.3	1.2	0.3	1.2	0.3	1.2	ns	3
WRITE recovery time	'WR	1	-	1	-	1	-	ns	12
		CLK + 7ns	-	CLK + 6ns	-	CLK + 7ns	-		
Exit SELF REFRESH-to-ACTIVE command	'XSR	12	-	12	-	14	-	ns	13
		67	-	70	-	70	-	ns	14

- FMC SDCLK
- Self-refresh ≥ 42ns
- SDRAM common row cycle delay ≥ 60ns
- Auto refresh ≥ 60ns
- Row to column delay ≥ 18ns
- SDRAM common row precharge delay ≥ 18ns
- Write recovery time ≥ 1CLK + 7ns
- Exit self-refresh delay ≥ 67ns

组件构成





• 组件（GUI功能需要）

• FMC（外部存储器）

• 配置参数获取（接上页）

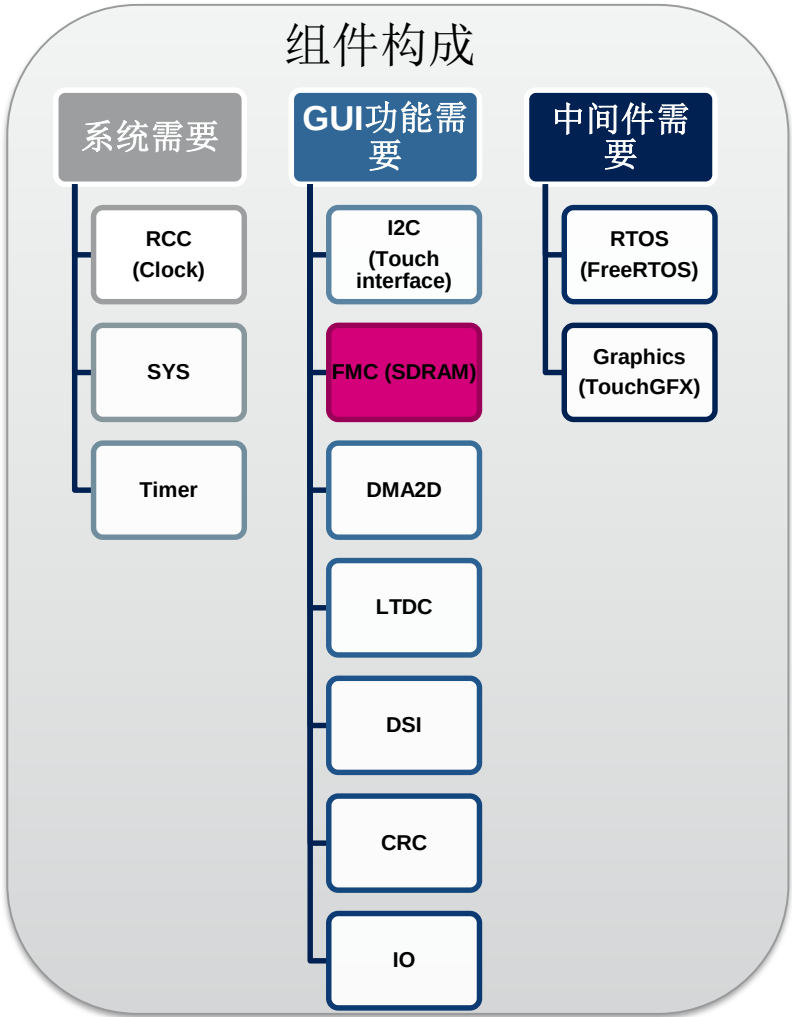
- 3. 进入SDRAM1标签页
- 4. 根据SDRAM数据手册中参数，配置SDRAM控制参数及时序参数

首先，根据存储器数据手册，获取相关的电气参数。如下所示。

（SDRAM型号：MT48LC4M32B2B5-6A）

Parameter	Symbol	-6	-6A	-7	Unit	Notes
READ/WRITE command to READ/WRITE command	t _{CCD}	1	1	1	t _{CK}	
CKE to clock disable or power-down entry mode	t _{CKED}	1	1	1	t _{CK}	15
CKE to clock enable or power-down exit setup mode	t _{PED}	1	1	1	t _{CK}	
DQM to input data delay	t _{DQD}	0	0	0	t _{CK}	
DQM to data mask during WRITES	t _{DQM}	0	0	0	t _{CK}	
DQM to data High-Z during READs	t _{DQZ}	2	2	2	t _{CK}	
WRITE command to input data delay	t _{DWD}	0	0	0	t _{CK}	
Data-in to ACTIVE command	CL = 3 t _{DAL} (3)	5	4	5	t _{CK}	16
	CL = 2 t _{DAL} (2)	4	4	4	t _{CK}	16
	CL = 1 t _{DAL} (1)	3	3	3	t _{CK}	16
Data-in to PRECHARGE command	t _{DPL}	3	3	3	t _{CK}	17
Last data-in to burst STOP command	t _{BDL}	1	1	1	t _{CK}	
Last data-in to new READ/WRITE command	t _{CDL}	1	1	1	t _{CK}	
Last data-in to burst PRECHARGE command	t _{RDL}	2	2	2	t _{CK}	17
LOAD MODE REGISTER command to ACTIVE or REFRESH command	t _{MRD}	2	2	2	t _{CK}	
Data-out to High-Z from PRECHARGE command	CL = 3 t _{ROH} (3)	3	3	3	t _{CK}	
	CL = 2 t _{ROH} (2)	2	2	2	t _{CK}	
	CL = 1 t _{ROH} (1)	1	1	1	t _{CK}	

Load mode register to active delay = 2 tCK





• 组件（GUI功能需要）

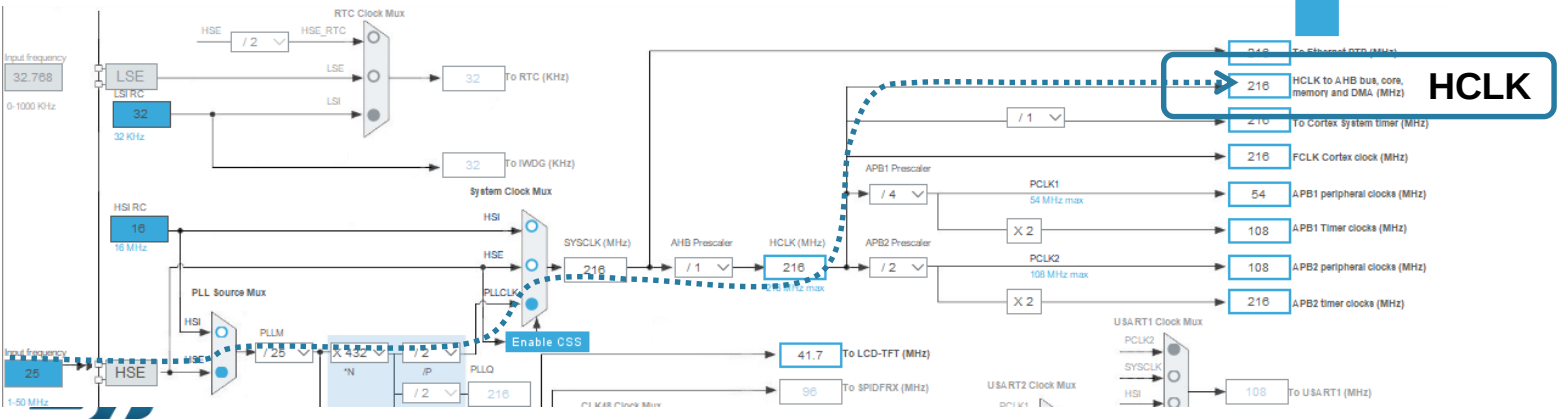
• FMC（外部存储器）

• 配置参数获取（接上页）

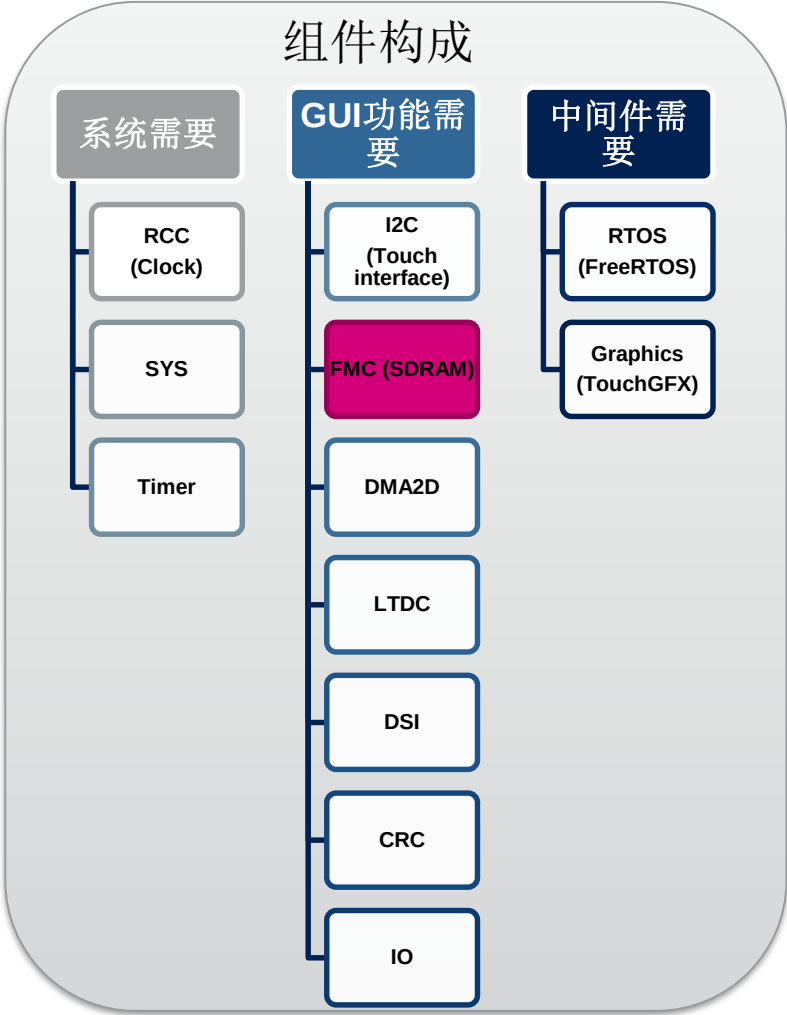
- 3. 进入SDRAM1标签页
- 4. 根据SDRAM数据手册，获取SDRAM的时间参数，配置SDRAM控制参数及时序参数

获取/设置主频。演示例中，采用最高主频。**FMC**时钟源频率路径如下。（更多时钟配置内容请参考“时钟配置”）

$$\text{FMC_SDCLK} = \text{HCLK} / \text{SDRAM common clock}$$



注：SDRAM common clock为STM32CubeMX中，SDRAM（FMC）配置项。具有SDRAM分频作用。





• 组件（GUI功能需要）

- FMC（外部存储器）

• 配置参数获取（接上页）

- 3. 进入SDRAM1标签页
- 4. 根据SDRAM数据手册中参数，配置SDRAM控制参数及时序参数
- MT48LC4M32B2B5-6A支持CAS latency (CL) : 1, 2, 3
- CAS (READ) latency = 18 ns

$$HCLK = 216MHz$$
$$FMC_SDCLK = \frac{HCLK}{SDRAM\ common\ clock}$$
$$\frac{CL}{FMC_SDCLK} \geq 18ns$$

SDRAM common clock 范围： 2, 3

为了实现更高的SDRAM访问频率，在能满足参数的前提下，SDRAM common clock选择2

$$CL = 2时, \frac{CL}{FMC_SDCLK} \approx 18.5ns ; CL = 3时, \frac{CL}{FMC_SDCLK} \approx 27.8ns$$

- 考虑到冗余，选择CL为3

注：SDRAM common clock为STM32CubeMX中，SDRAM（FMC）配置项。具有SDRAM分频作用。

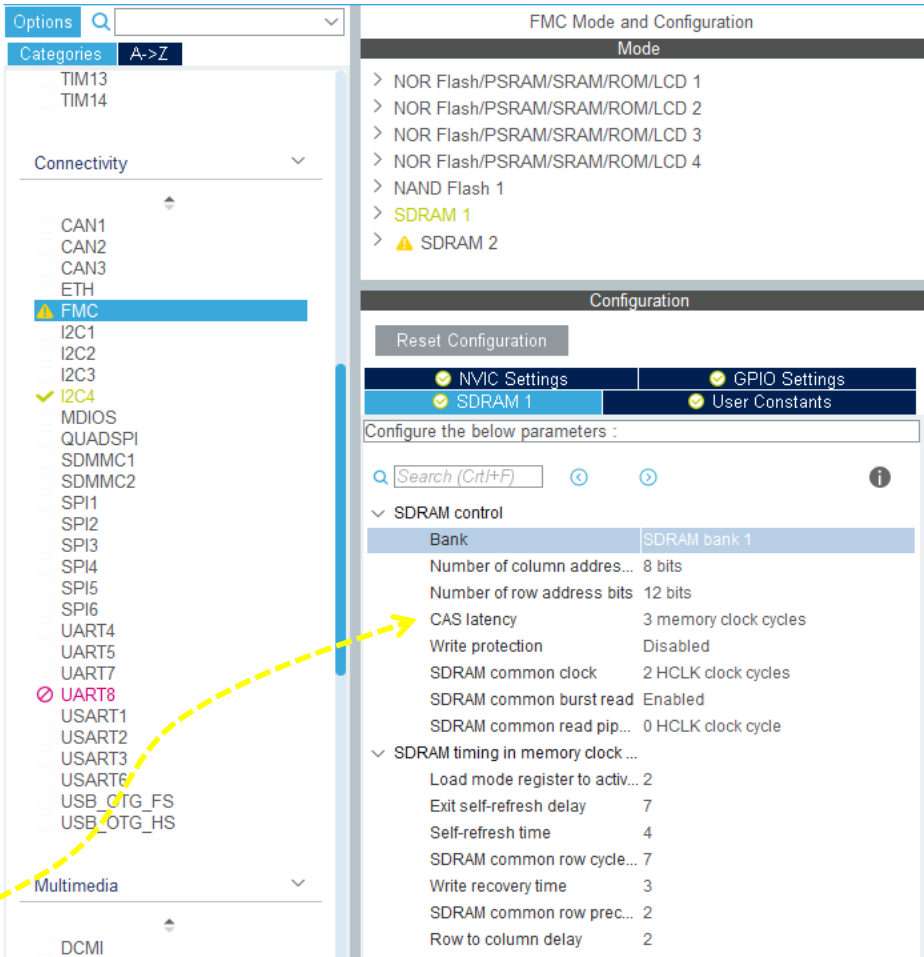


图. FMC SDRAM配置
（@STM32CubeMX）



• 组件（GUI功能需要）

• FMC（外部存储器）

• 配置参数获取（接上页）

- 3. 进入SDRAM1标签页
- 4. 根据SDRAM数据手册中参数，配置SDRAM控制参数及时序参数

参数	最小值
Load mode register to active delay	2 CLK
Exit self-refresh delay	67ns
Self-refresh	42ns
SDRAM common row cycle delay	60ns
Write recovery time	1 CLK +7ns
SDRAM common row precharge delay	18ns
Row to column delay	18ns



参数	最小值 (FMC SDCLK)
Load mode register to active delay	2
Exit self-refresh delay	7
Self-refresh	5
SDRAM common row cycle delay	7
Write recovery time	2
SDRAM common row precharge delay	2
Row to column delay	2

考虑到如下要求（STM32CubeMX中提示）：
WriteRecoveryTime >= SelfRefreshTime – RowToColumnDelay。设置其为3。

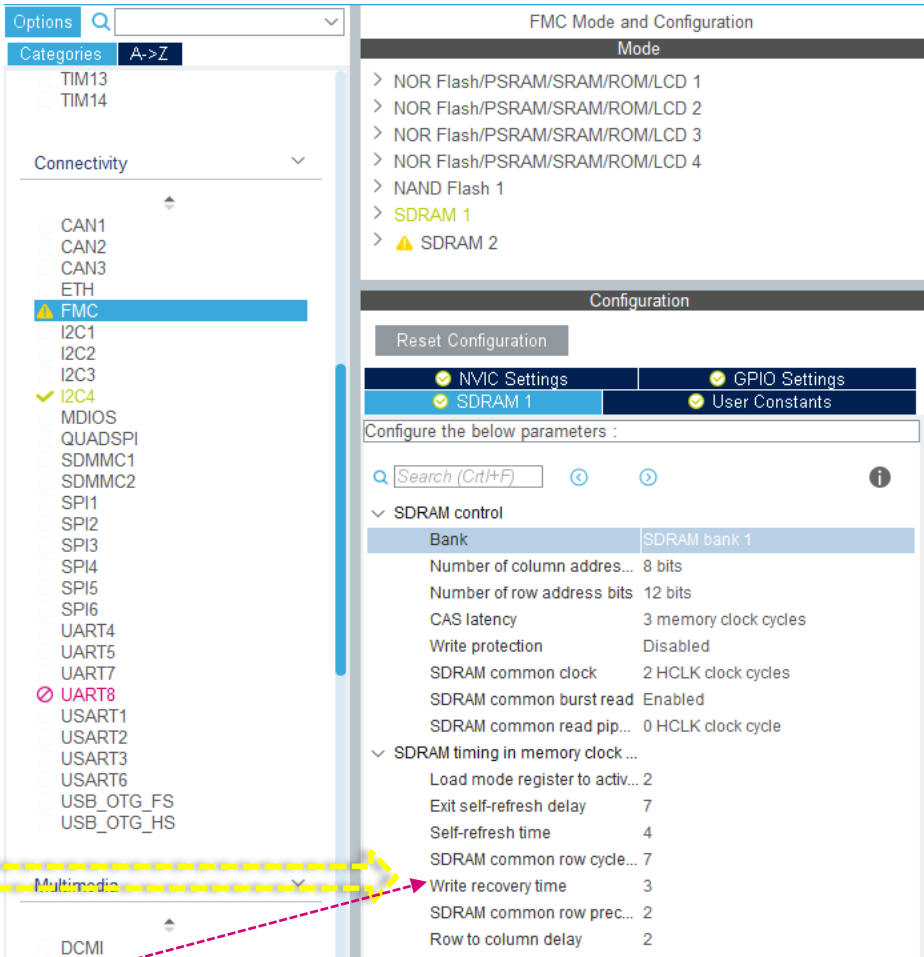


图. FMC SDRAM配置
（@STM32CubeMX）

• 在之前，确定了FMC_SDCLK = 216/2 MHz = 108MHz



注：STM32CubeMX生成初始化中，未对Self-refresh进行设置，演示例中Self-refresh参数无实际意义。



实现示例（续）

36



• 组件（GUI功能需要）

• DMA2D

- 2D图形加速器

• DMA2D在STM32系统架构中位置及作用

- 系统总线主从设备间数据传递/填充

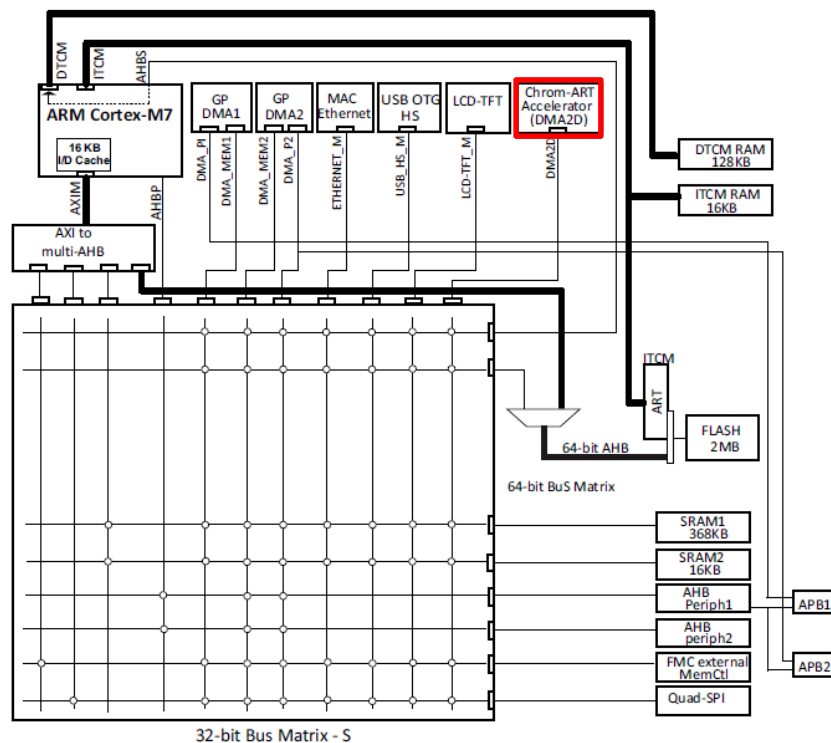
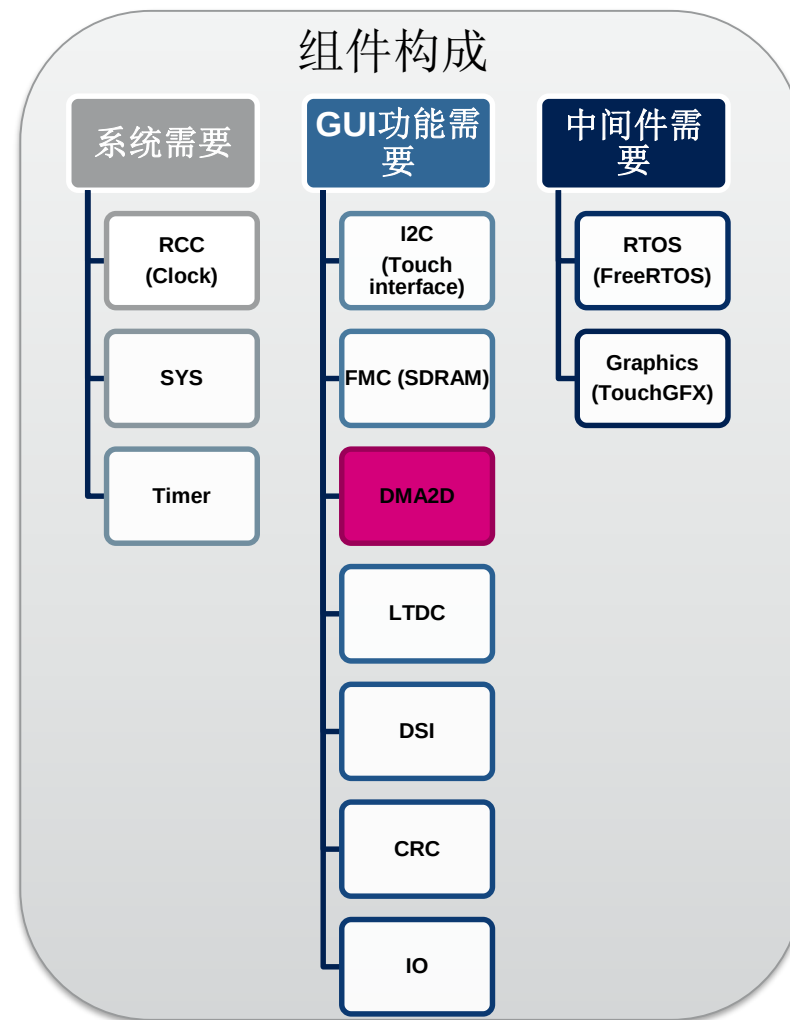


图. 系统结构图摘自参考手册RM0410

组件构成





• 组件（GUI功能需要）

- DMA2D
 - 2D图形加速器
 - 参数保持默认即可。

Options

Categories A-Z

UART4
UART5
UART7
❌ **UART8**
USART1
USART2
USART3
USART6
USB_OTG_FS
USB_OTG_HS

Multimedia

DCMI
✅ **DMA2D**
DSIHOST
HDMI_CEC
I2S1
I2S2
I2S3
JPEG
LTDC
SAI1

DMA2D Mode and Configuration

Mode

☑ Activated

Configuration

Reset Configuration

☑ Parameter Settings ☑ User Constants ☑ NVIC Settings

Configure the below parameters :

Basic Parameters

Transfer ModeMemory to Memory

Color ModeARGB8888

Output Offset0

Foreground layer Configuration

DMA2D Input Color ModeARGB8888

DMA2D ALPHA MODENo modification of the alpha chann...

Input Alpha0

Input Offset0

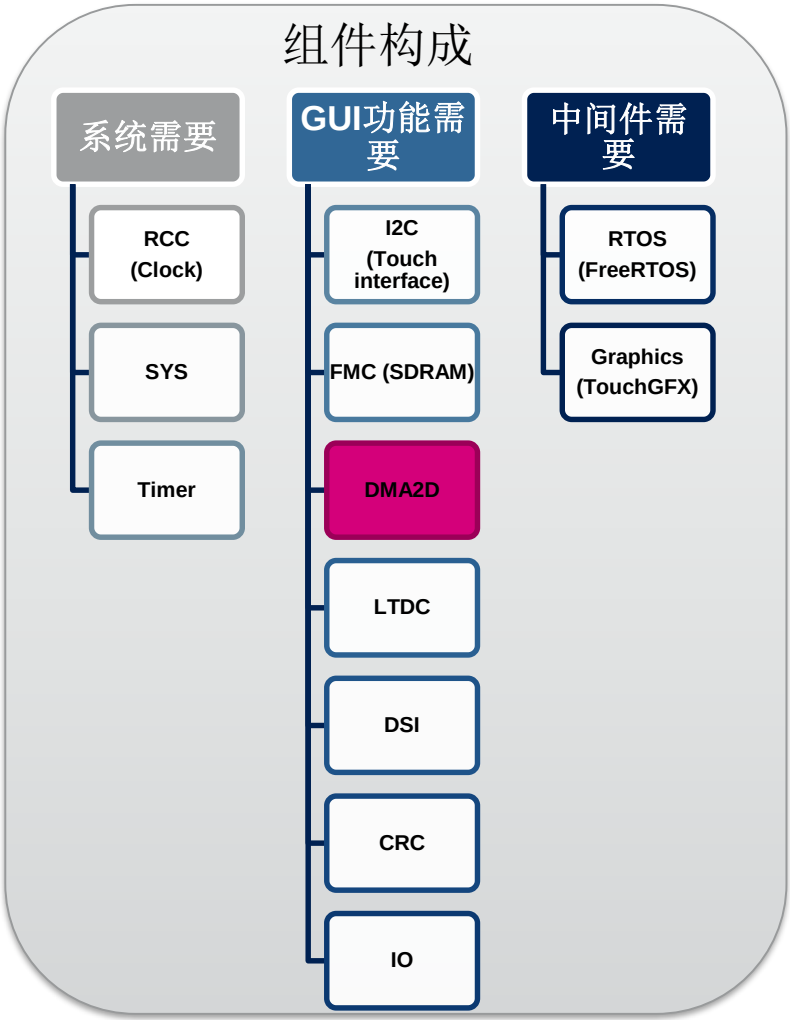
DMA2D ALPHA InversionRegular Alpha

DMA2D Red and Blue swapRegular mode (RGB or ARGB)

☑ Parameter Settings ☑ User Constants ☑ NVIC Settings

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
DMA2D global interrupt	☑	0	0

图. FMC SDRAM配置
（@STM32CubeMX）





实现示例（续）

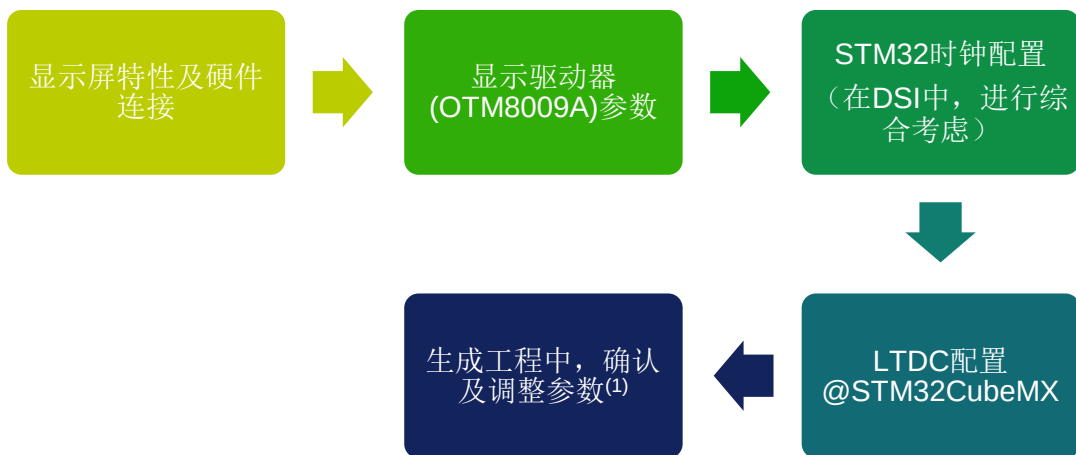
38



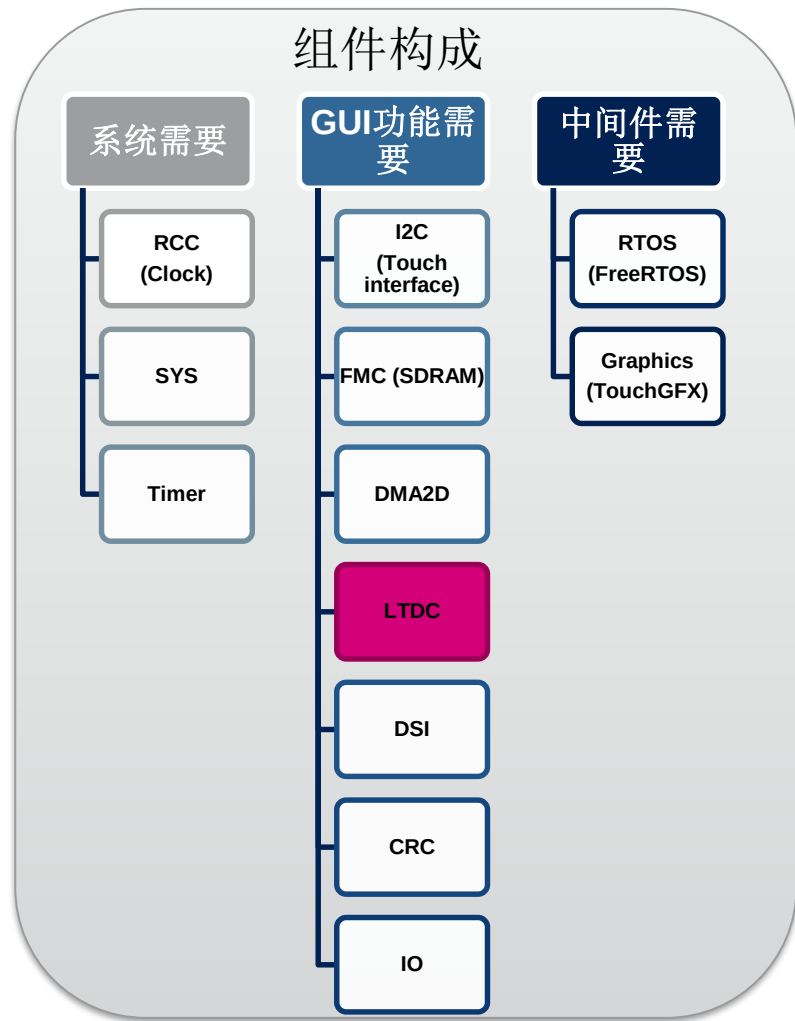
• 组件（GUI功能需要）

• LTDC（LCD-TFT Display Controller）

- 不同的显示驱动器，对应的LTDC配置参数不同。
- 首先，针对当前演示例，呈现在STM32CubeMX上LTDC配置。
- 然后，以STM32F769I-Disco板上的显示屏KJD KM-040TMP-02 (驱动器型号 **OTM8009A**)为例，介绍如何根据实际情况，获取配置参数。



LTDC配置流程图（以演示例中采用的OTM8009A为例）





- 组件（GUI功能需要）
 - LTDC配置（STM32CubeMX中）

• Parameter settings设置

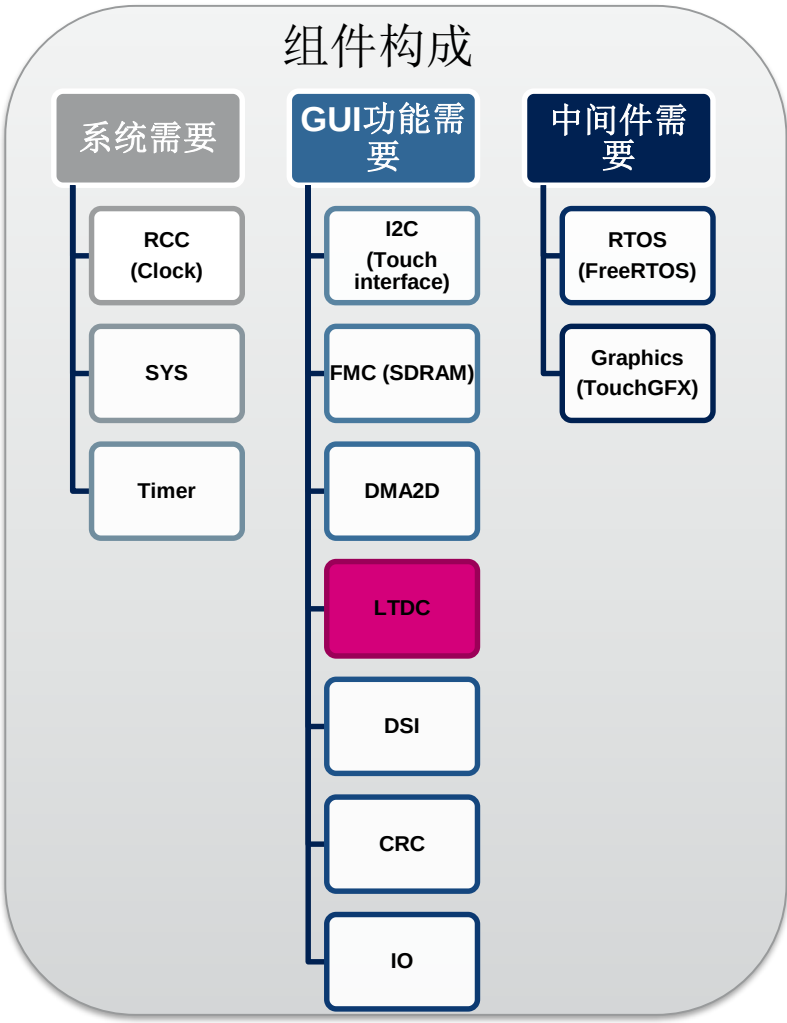
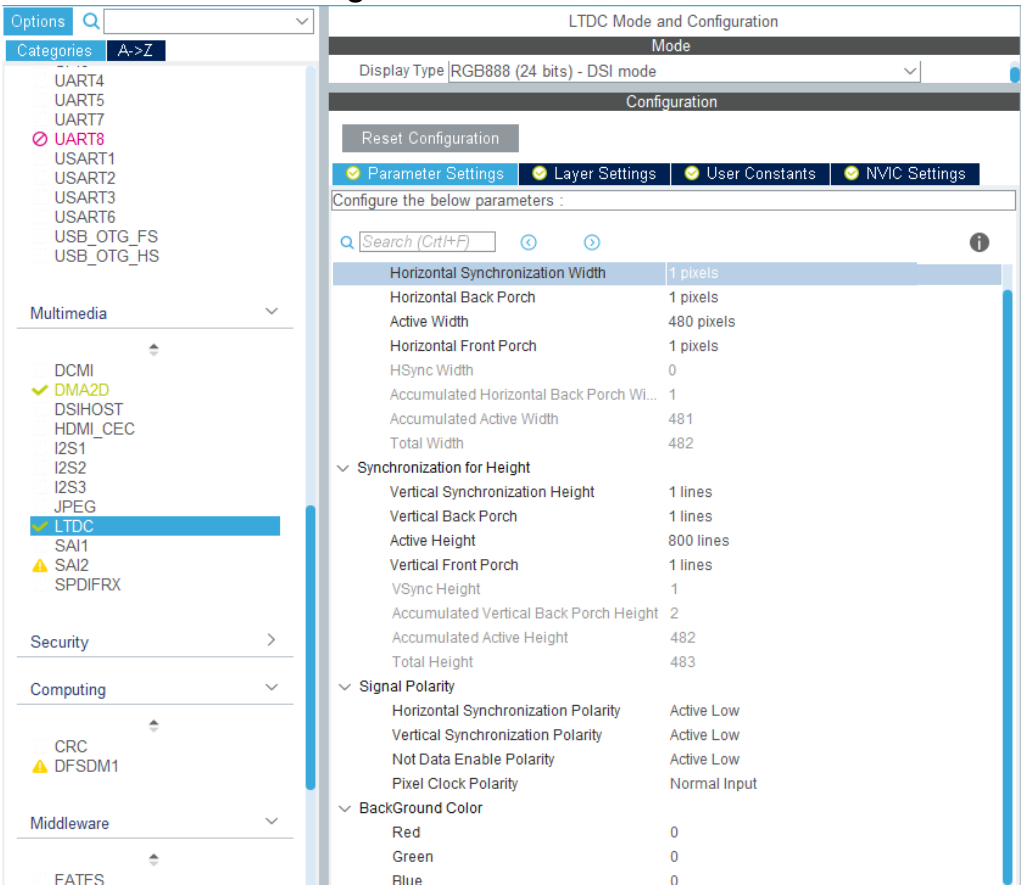


图. LTDC配置（@STM32CubeMX）



- 组件（GUI功能需要）
 - LTDC配置（STM32CubeMX中）
 - Layer settings 设置

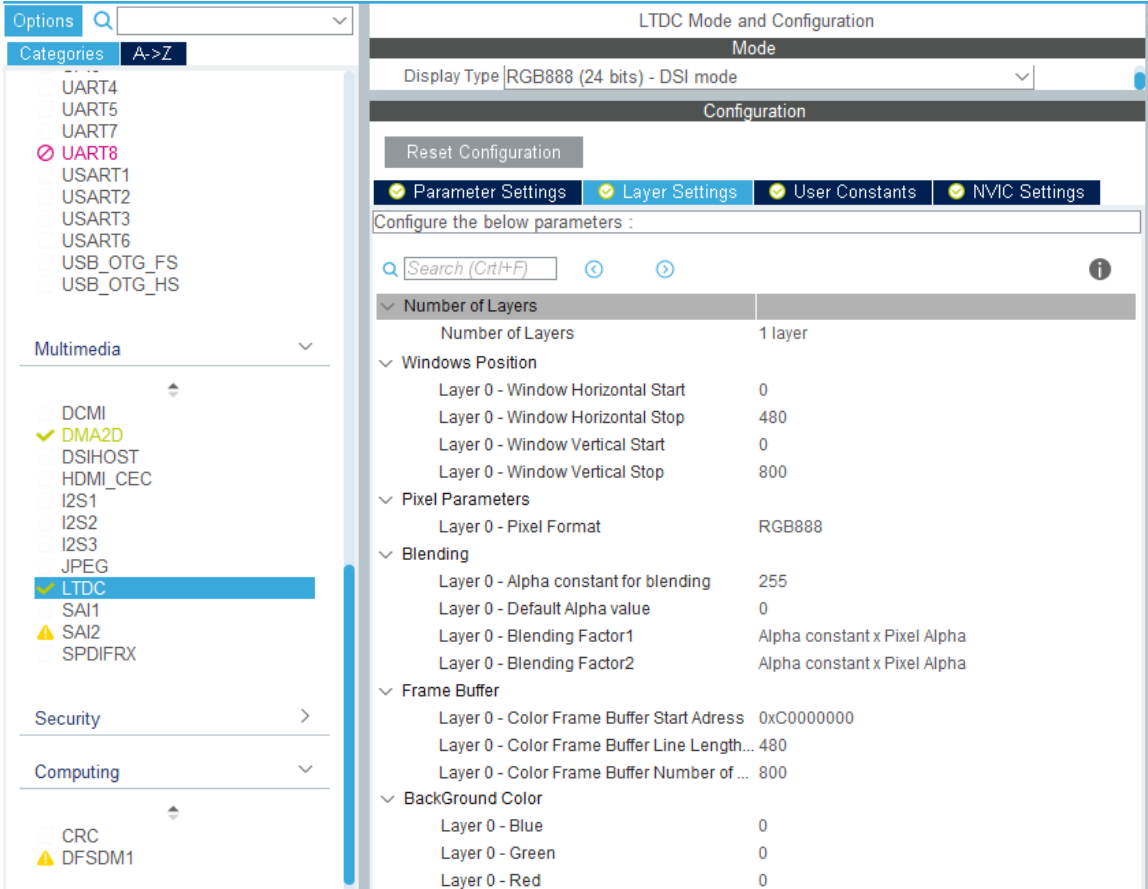
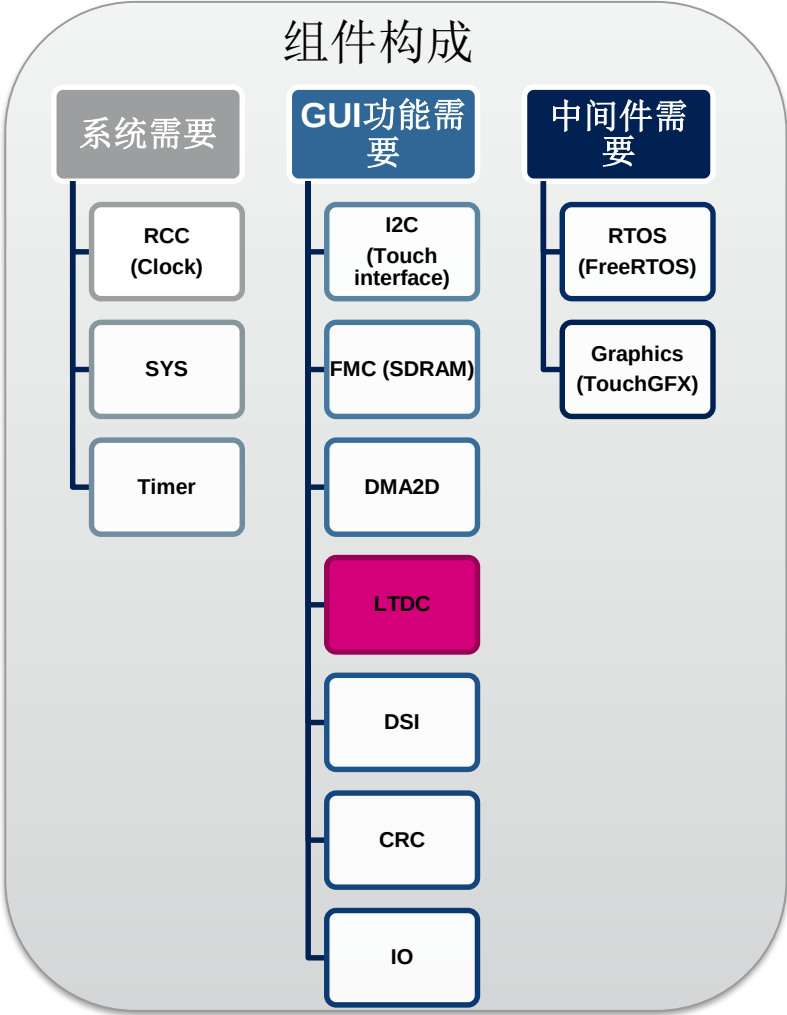


图. LTDC配置（@STM32CubeMX）



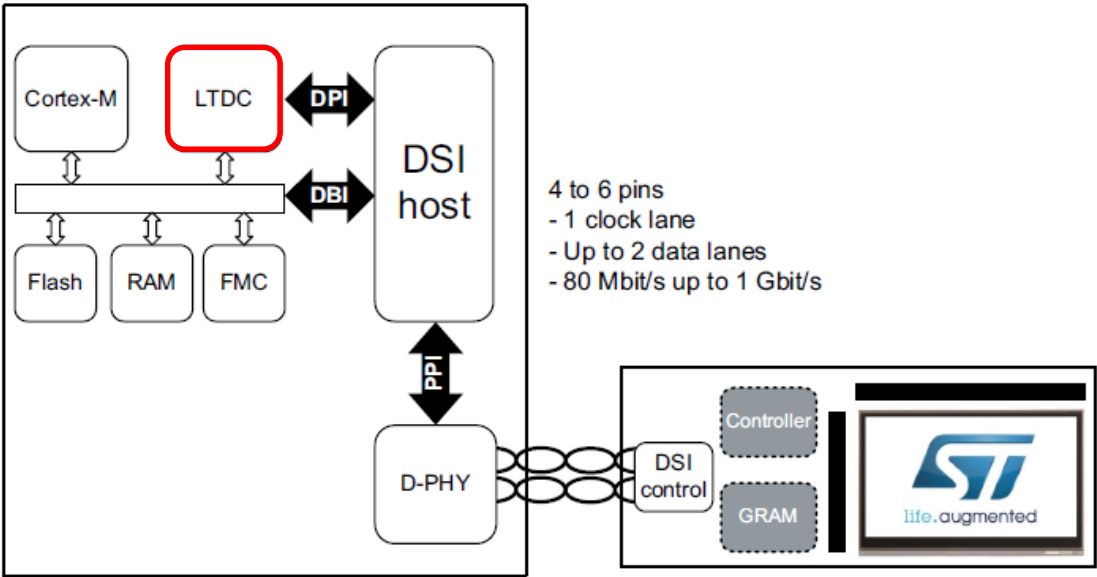


• 组件（GUI功能需要）

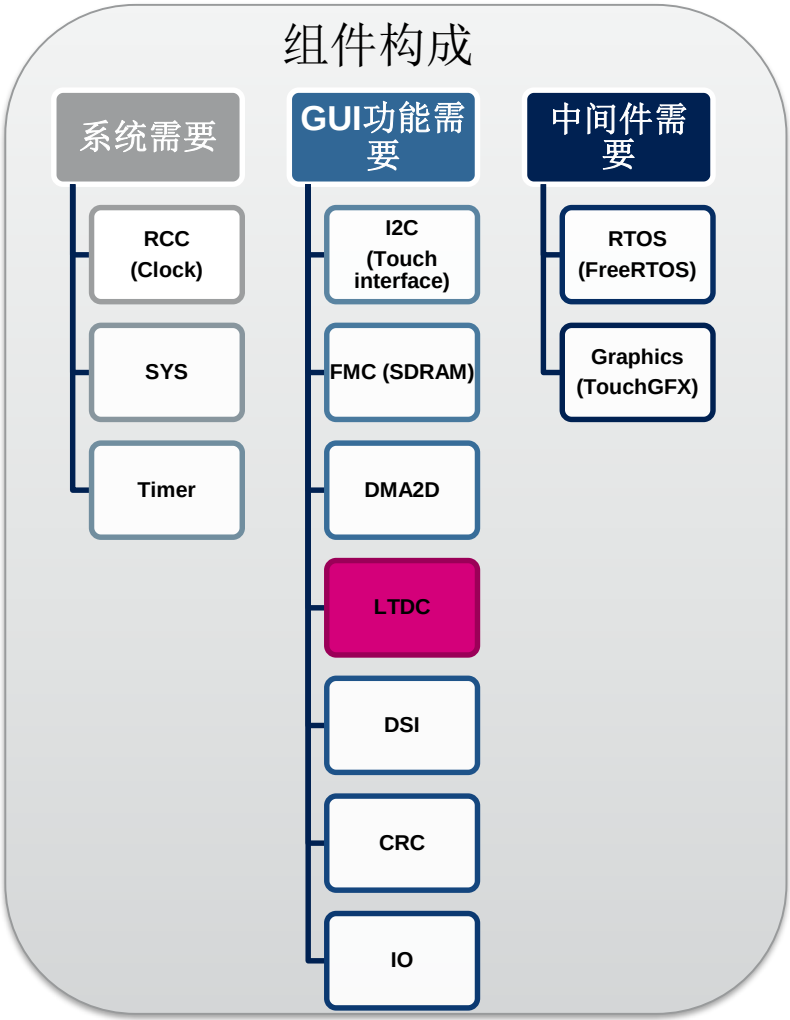
- LTDC（LCD-TFT Display Controller）
 - Parameter settings设置
- 配置参数获取

在进行LTDC的配置前，需要先了解采用DSI屏时，LTDC在GUI应用中的位置和作用。

如下框图所示，LTDC依然作为LCD-TFT显示控制器，但是不直接物理访问LCD。而是通过集成的DSI主器件和D-PHY，实现对DSI屏的控制。



组件构成





• 组件（GUI功能需要）

- LTDC（LCD-TFT Display Controller）
 - Parameter settings设置
- 配置参数获取
 - 1. 根据应用需要和显示屏支持，综合考虑，确定Display Type

应用需要

- 颜色深度
- 帧率
- 数据带宽
- 架构
- RAM

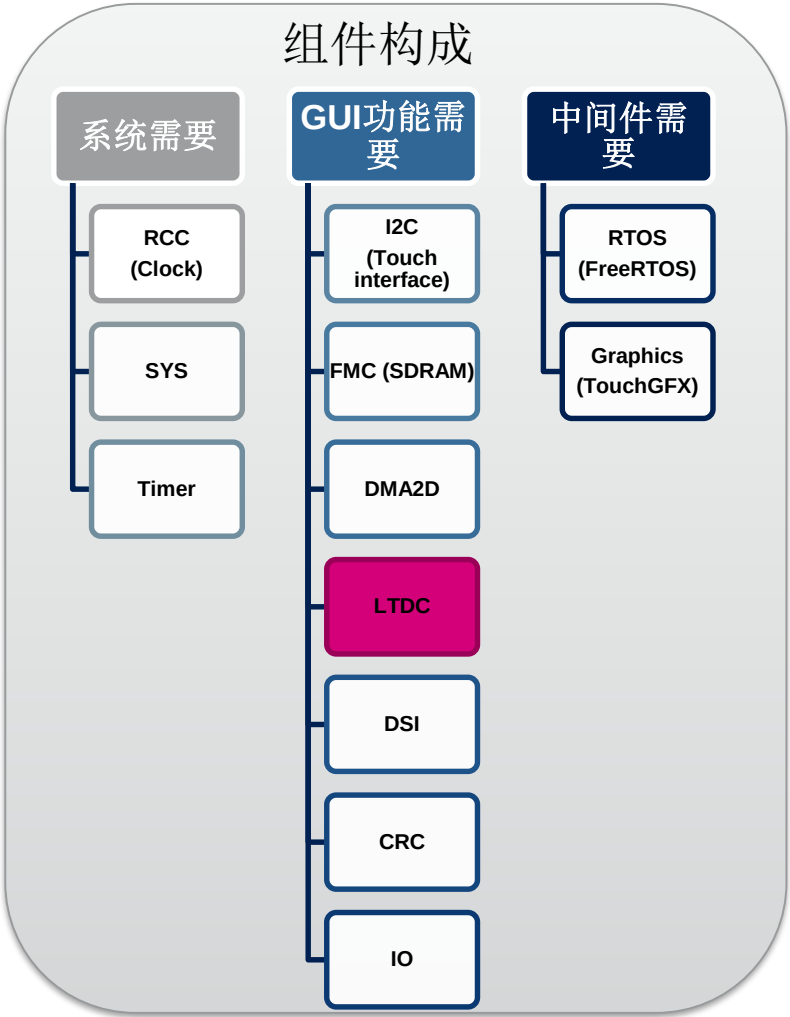
显示驱动器
OTM8009A支持

- RGB565
- RGB666
- RGB888

Display type(LTDC支持)

- RGB565 (Y/N DSI)
- RGB666 (Y/N DSI)
- RGB888 (Y/N DSI)

综合确定
Display Type。
演示例中，
Display type
选择为
RGB888 (DSI
mode)

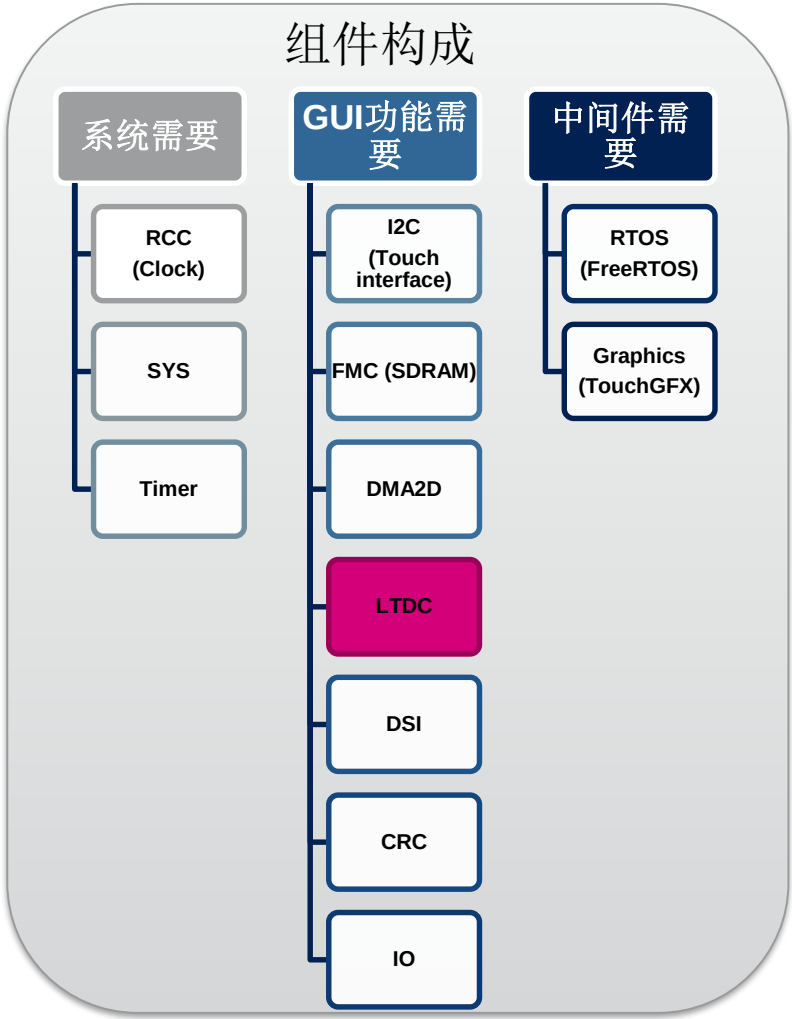




- 组件（GUI功能需要）
 - LTDC（LCD-TFT Display Controller）
 - Parameter settings设置
 - 配置参数获取
 - 2. 根据LTDC（DSI模式）和显示屏模组规范书，完成Parameter Settings配置
 - LTDC中同步信号时间参数和信号极性考虑因素，如下表。

屏类型	显示接口	模式	显示模组	
			同步信号时间参数	控制信号极性
RGB屏	LTDC		Y	Y
MIPI屏	DSI（内部关联LTDC）	Video	Y	N
		Command/Adapted Command	N	N

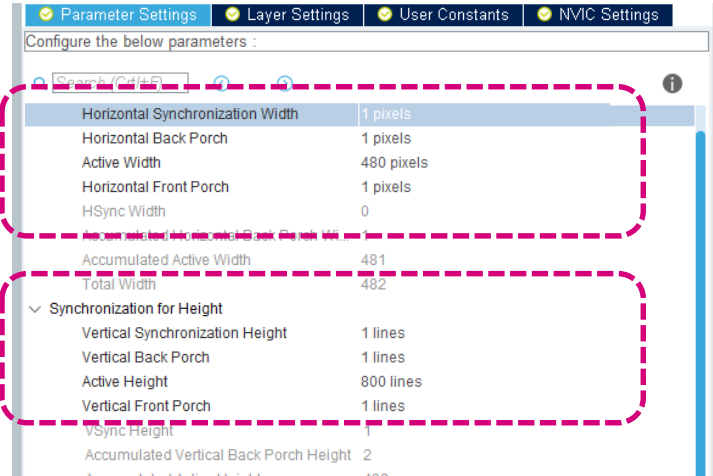
注： Y 表示需要根据显示模组中参数/特性进行匹配、配置。
N 不需要与显示模组中参数/特性匹配。但LTDC需保持与DSI中信号极性一致。





• 组件（GUI功能需要）

- LTDC（LCD-TFT Display Controller）
 - Parameter settings设置
- 配置参数获取
 - 2. 根据LTDC（DSI模式）和显示屏模组规范书，完成Parameter Settings配置
 - 显示模组支持多种分辨率，默认分辨率为480x800
 - 演示例中采用480x800分辨率
 - 注：演示例中采用DSI屏。在使用DSI模式，并且选择command mode时，配置中的HSW, HBP, HFP, VSH, VBP, VFP不影响显示屏同步参数，配置为最低即可；使用LTDC，不使用DSI情况下，需要根据显示屏模组规范书中电气参数，进行配置，如右表所示。



	类别	符号	前提	最小值	典型值	最大值	单位
Vertical	cycle period	T_{VP}			832		HS
	low pulse width	T_{VS}		1	10	63	HS
	front porch	T_{VFP}		4	16	63	HS
	back porch	T_{VBP}		3	15	63	HS
	blanking period	T_{VBL}	$T_{VS}+T_{VBP}+T_{VFP}$	8	32	128	HS
	active area	T_{VDISP}			800		HS
	Refresh rate	T_{VRR}	Frame rate	-	60	-	Hz
Horizontal	cycle period	T_{HP}		520	522	1024	PCLK
	low pulse width	T_{HS}		2	2	63	PCLK
	front porch	T_{HFP}		18	20	255	PCLK
	back porch	T_{HBP}		20	20	255	PCLK
	blanking period	T_{HBL}	$T_{HS}+T_{HBP}+T_{HFP}$	40	42	1024	PCLK
	active area	T_{HDSIP}			480		PCLK
		$f_{PCLKCYC}$		24	26.37	30.74	MHz

OTM8009A 480x800显示时间参数表





实现示例（续）

45



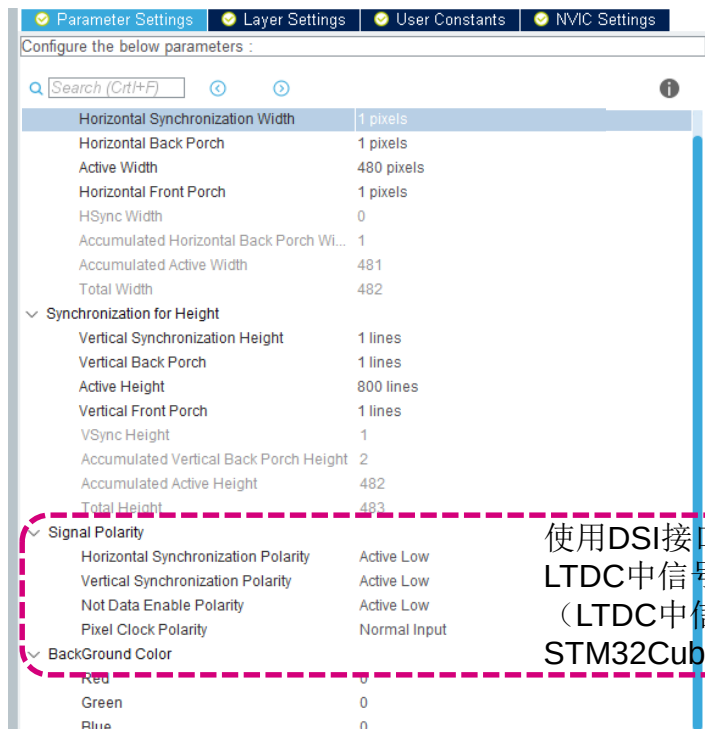
• 组件（GUI功能需要）

• LTDC（LCD-TFT Display Controller）

- Parameter settings设置

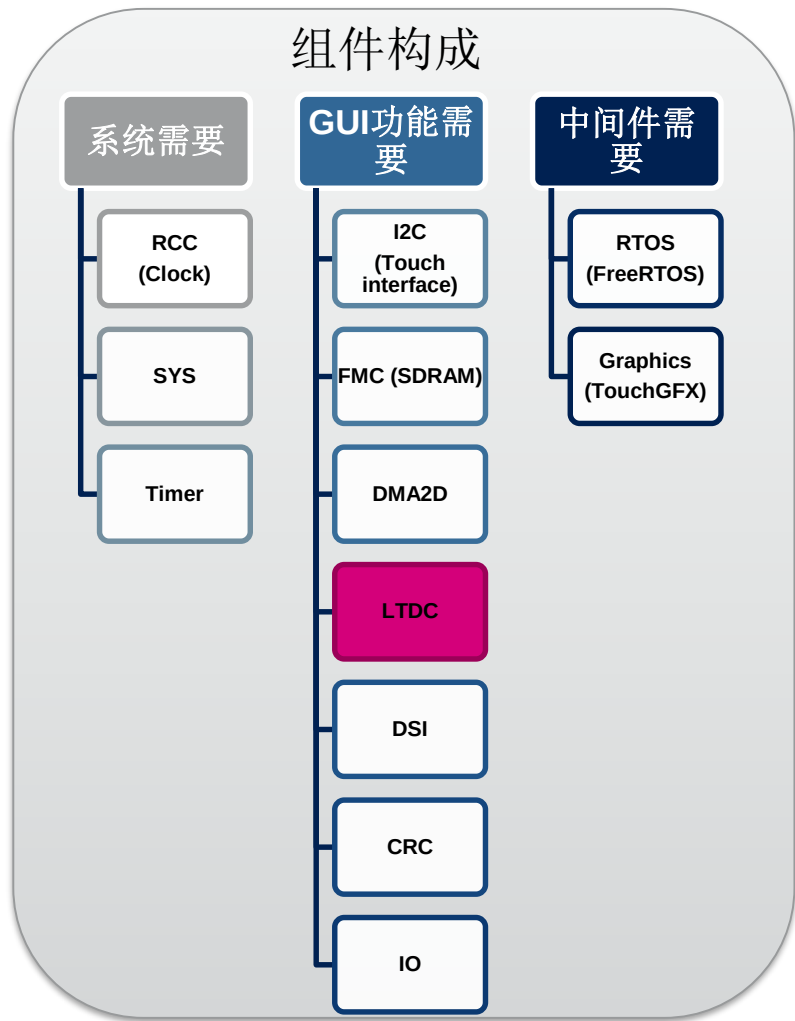
• 配置参数获取

- 2. 根据LTDC（DSI模式）和显示屏模组规范书，完成Parameter Settings配置



使用DSI接口屏时，STM32CubeMX中配置，LTDC中信号极性不需要修改。
（LTDC中信号极性与DSI中匹配即可，STM32CubeMX创建的工程中会自动匹配。）

组件构成





实现示例（续）

46



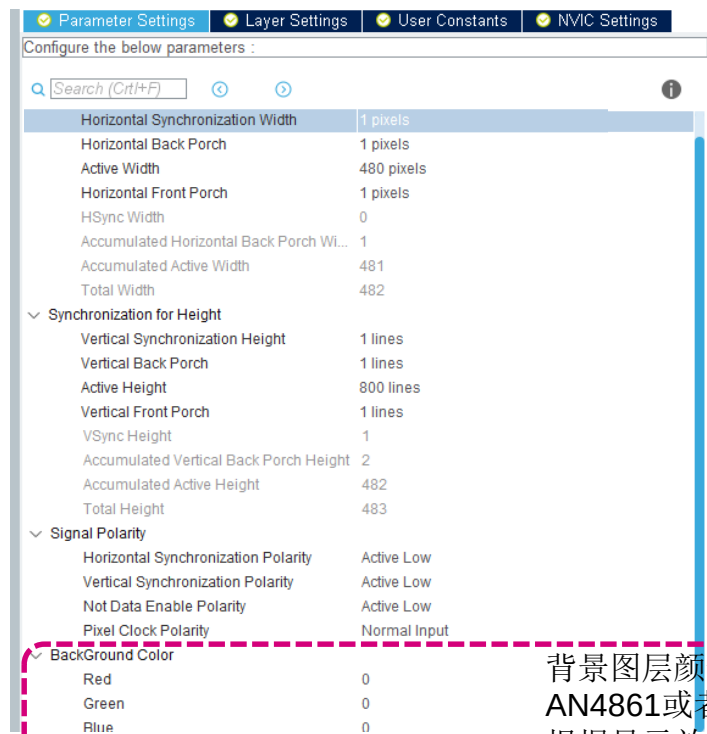
• 组件（GUI功能需要）

• LTDC（LCD-TFT Display Controller）

- Parameter settings设置

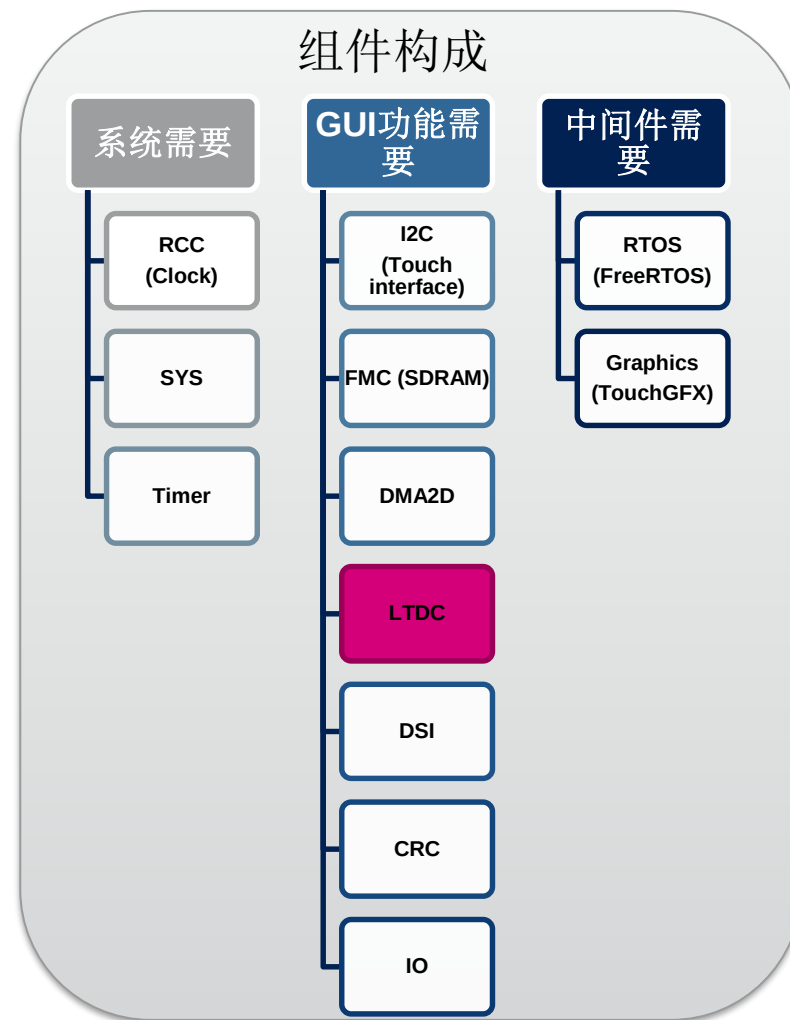
• 配置参数获取

- 2. 根据LTDC（DSI模式）和显示屏模组规范书，完成Parameter Settings配置



背景图层颜色（用于图层混合），详情请参考AN4861或者参考手册。
根据显示效果进行配置，演示例中保持默认。

组件构成





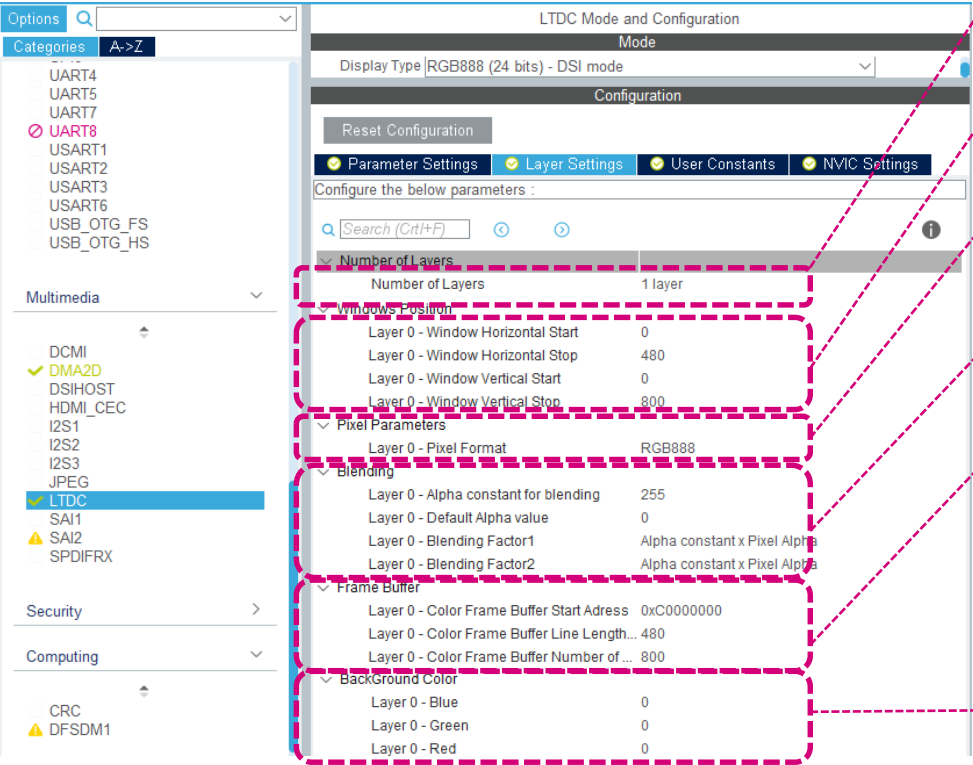
• 组件（GUI功能需要）

• LTDC配置（STM32CubeMX中）

• Layer settings 设置

• 图层配置

• 3. 根据显示要求和处理，完成Layer settings 配置



• 图层数（LTDC物理图层数，1~2）

• 图层中显示窗口起始和终止位置

• 图层颜色格式（对应FrameBuffer中像素数据存放格式。STM32CubeMX自动将Layer0的颜色格式关联到DSI的颜色编码）。

• 图层混合方式（请参考《GUI方案中ALPHA通道处理介绍》）

• 图层FrameBuffer

- 图层FrameBuffer首地址（演示例，FrameBuffer保存在外扩SDRAM。外扩SDRAM BANK1首地址 = 0xC0000000）
- FrameBuffer对应的行列像素个数，STM32CubeMX自动将其与Parameter settings中Active Width/Height关联。

• 图层默认颜色

• 定义默认图层颜色，详情请参考AN4861或者参考手册。

图. LTDC配置（@STM32CubeMX）



实现示例（续）

48



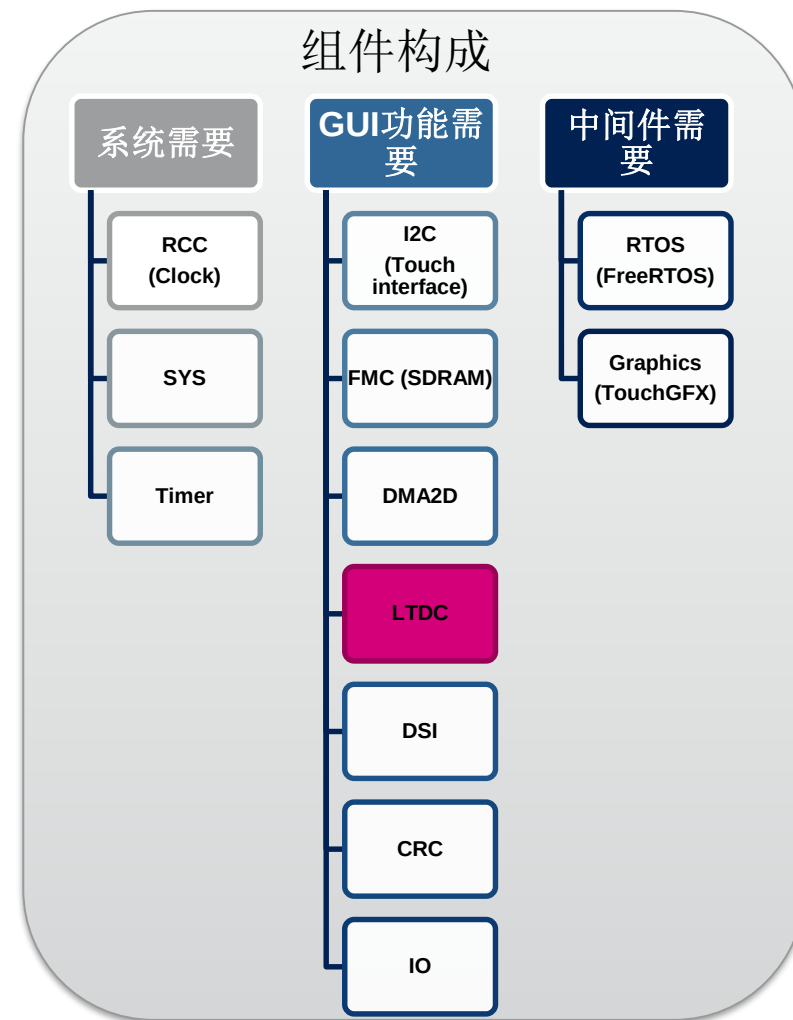
• 组件（GUI功能需要）

• LTDC配置（STM32CubeMX中）

- NVIC settings 设置

• 中断配置

- 3. 在使用DSI，并且DSI配置为Adapted command模式时，不强制要求开启LTDC中断。DSI配置会在后面介绍。





实现示例（续）

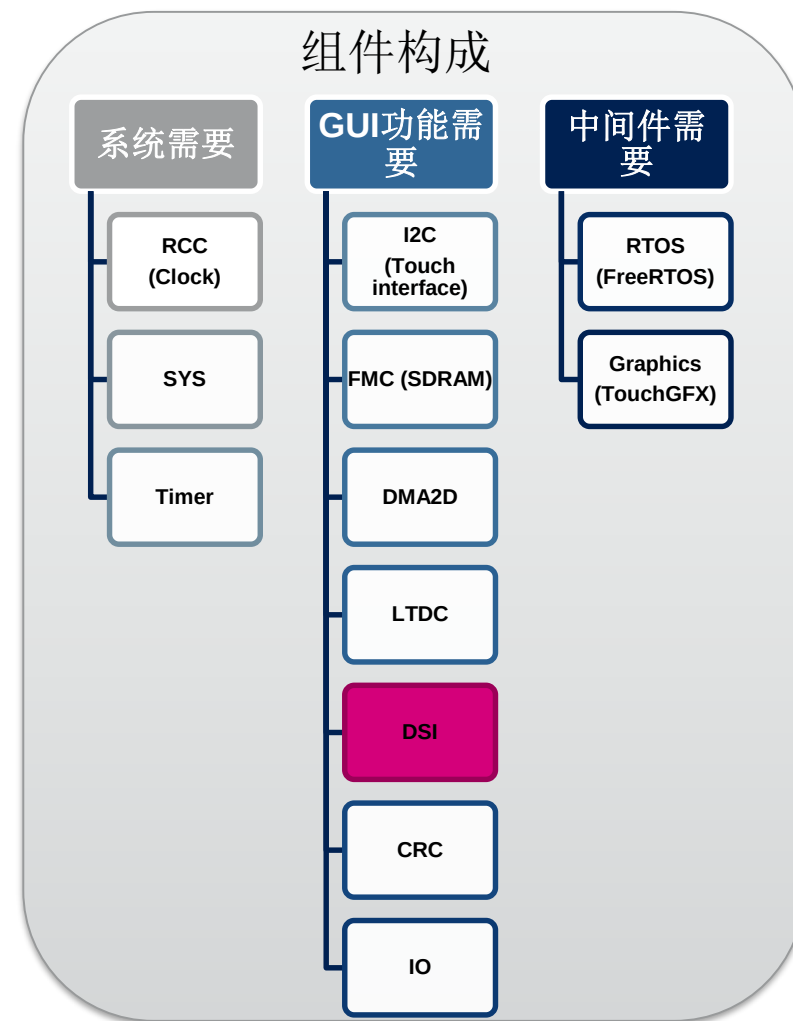
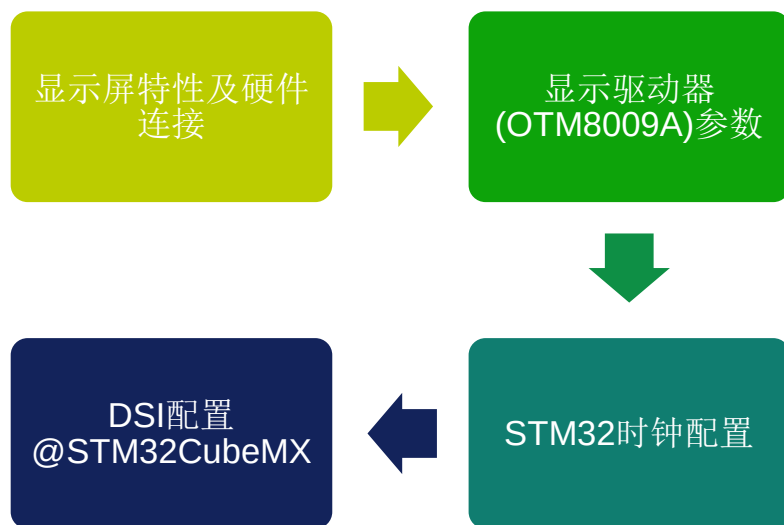
49



• 组件（GUI功能需要）

• DSI

- 不同的显示驱动器，对应的DSI配置参数不同。
- 首先，针对当前演示例，呈现在STM32CubeMX上DSI配置。
- 然后，以STM32F769I-Disco板上的显示屏KJD KM-040TMP-02 (驱动器型号 **OTM8009A**)为例，介绍如何根据实际情况，获取配置参数。

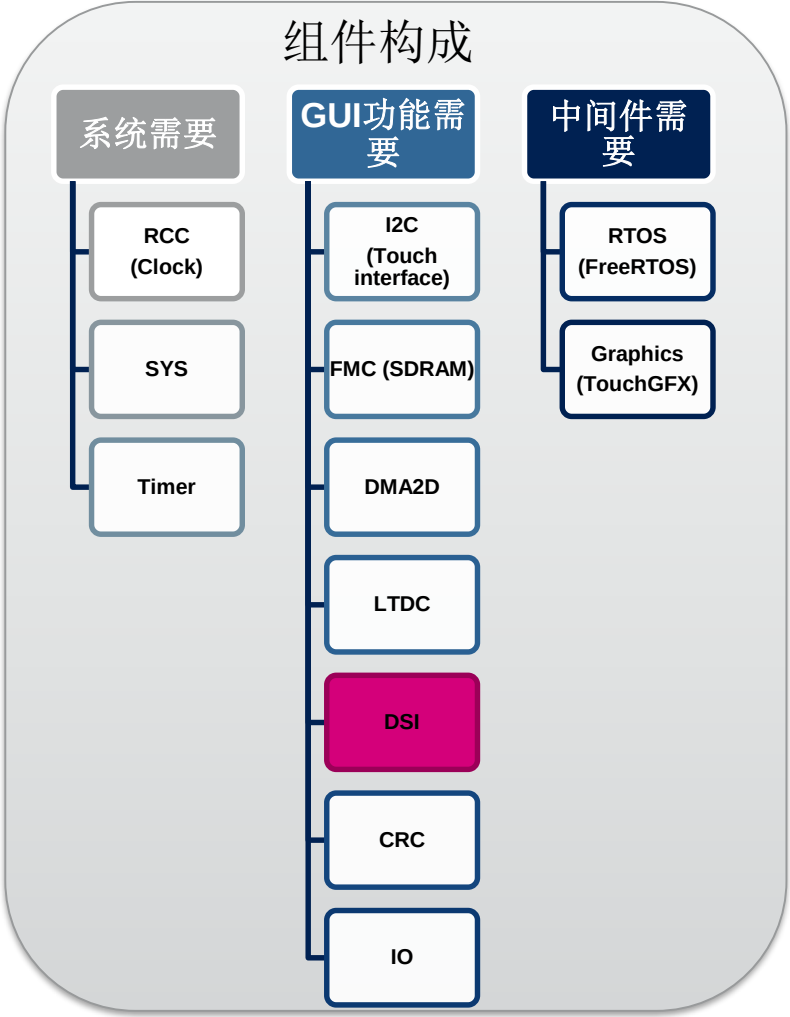
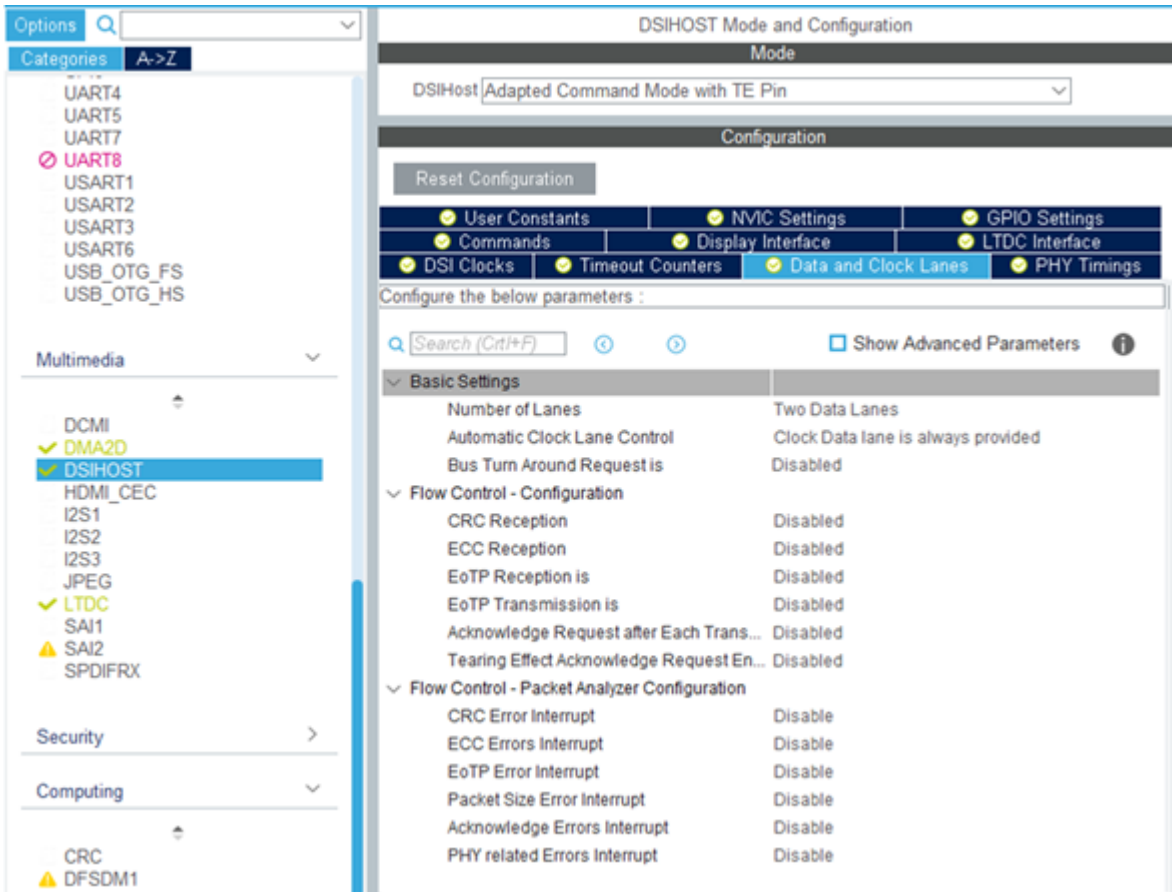




• 组件（GUI功能需要）

• DSI

- DSIHost 选择Adapted Command Mode with TE Pin
- Data and Clock Lanes（数据和时钟线配置）





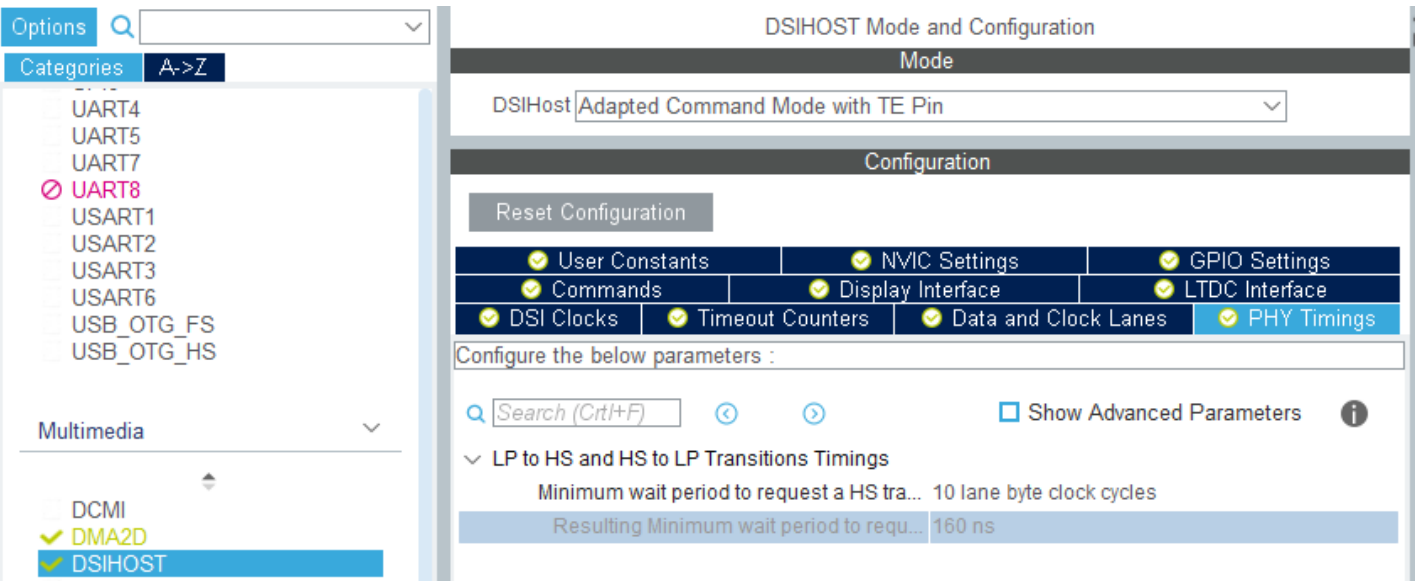
实现示例（续）

51

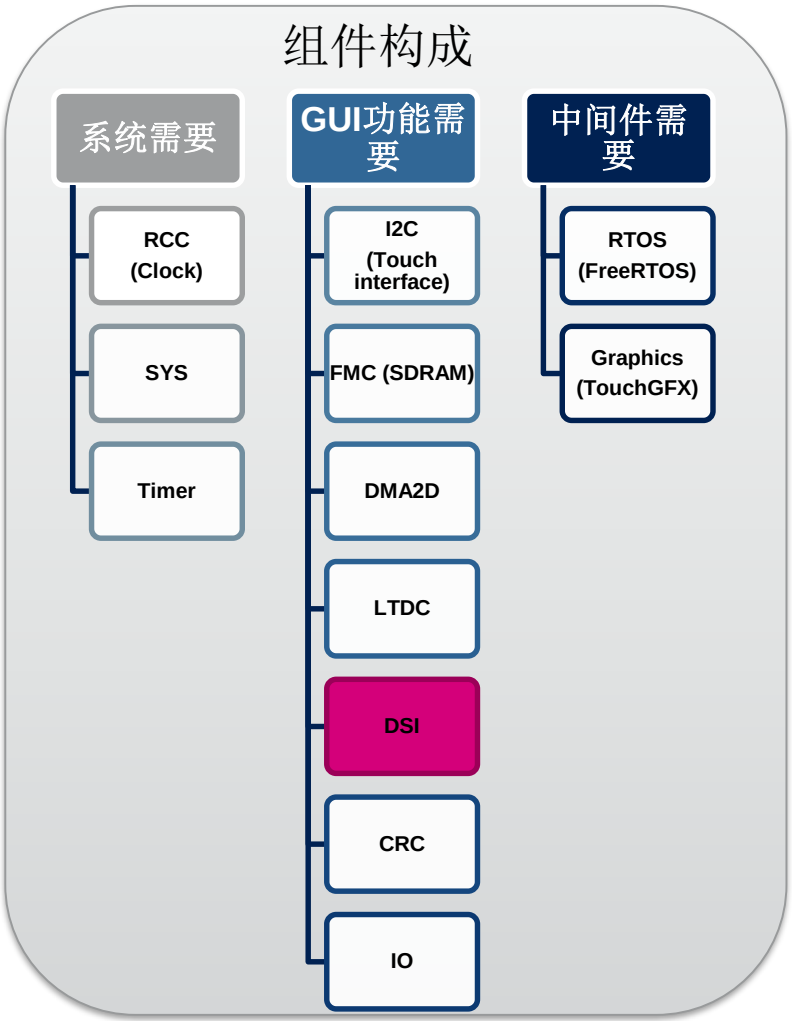
• 组件（GUI功能需要）

• DSI

• PHY Timings（PHY时间参数设置）



组件构成





• 组件（GUI功能需要）

• DSI

- Commands配置（命令参数设置）
- 根据显示驱动器OTM8009A中所述，有些命令仅支持LPDT（Low-power Mode）。简便考虑，命令统一设置为Low-power传输模式。

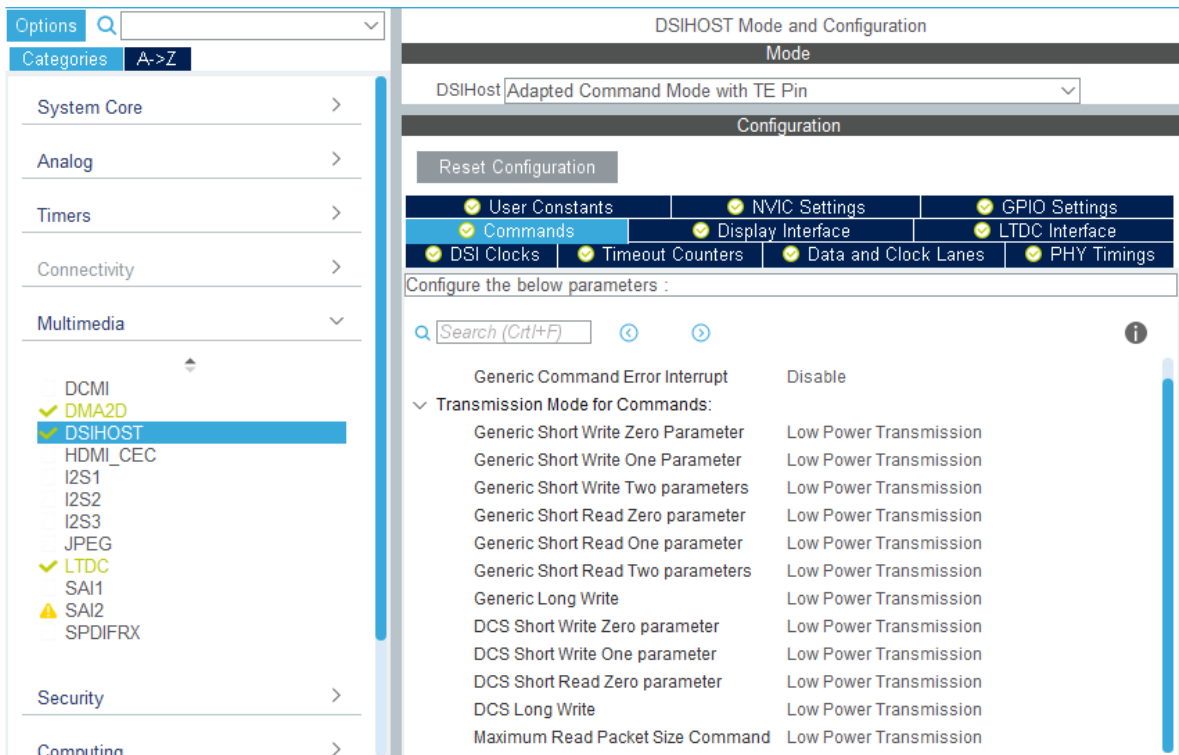
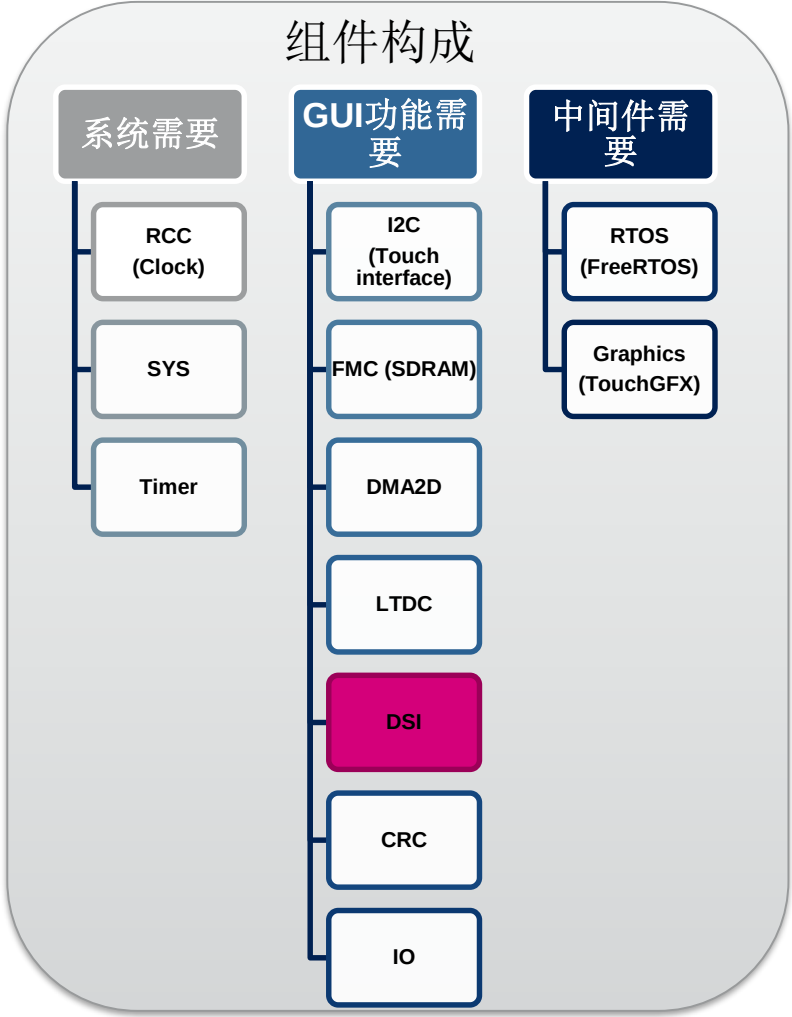


图. SDI Commands配置（@STM32CubeMX）

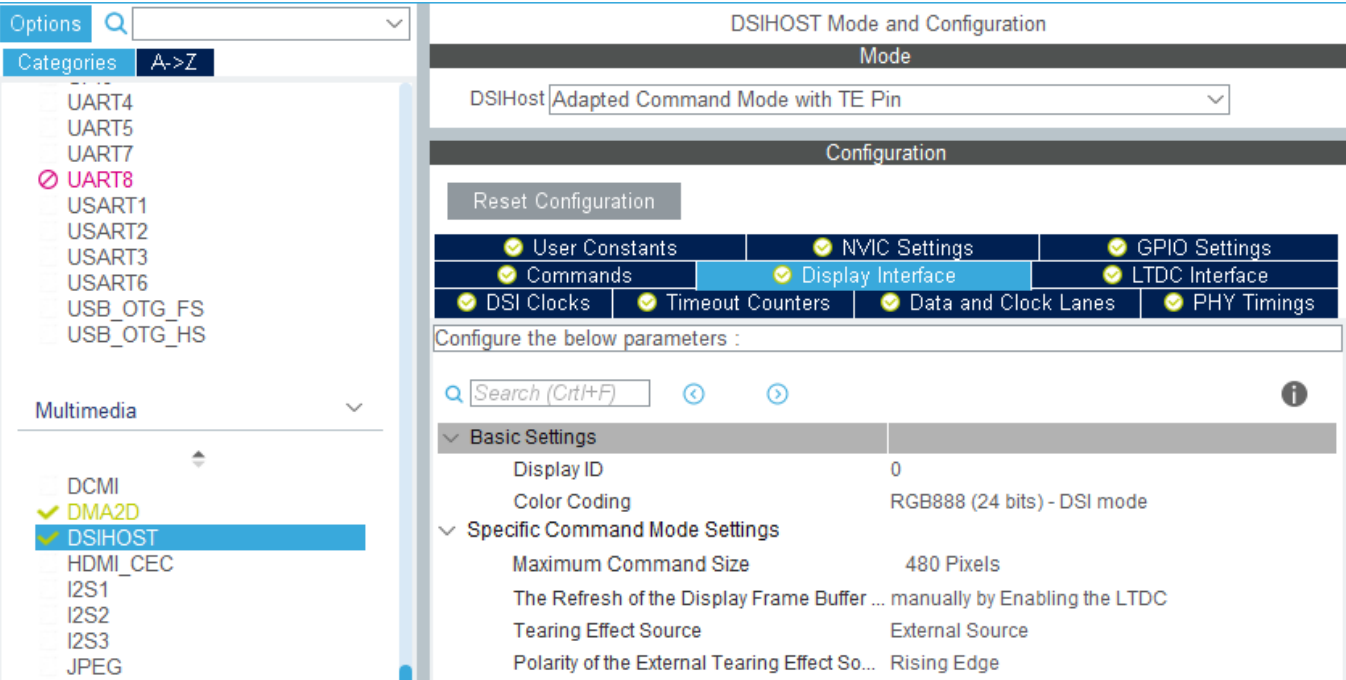




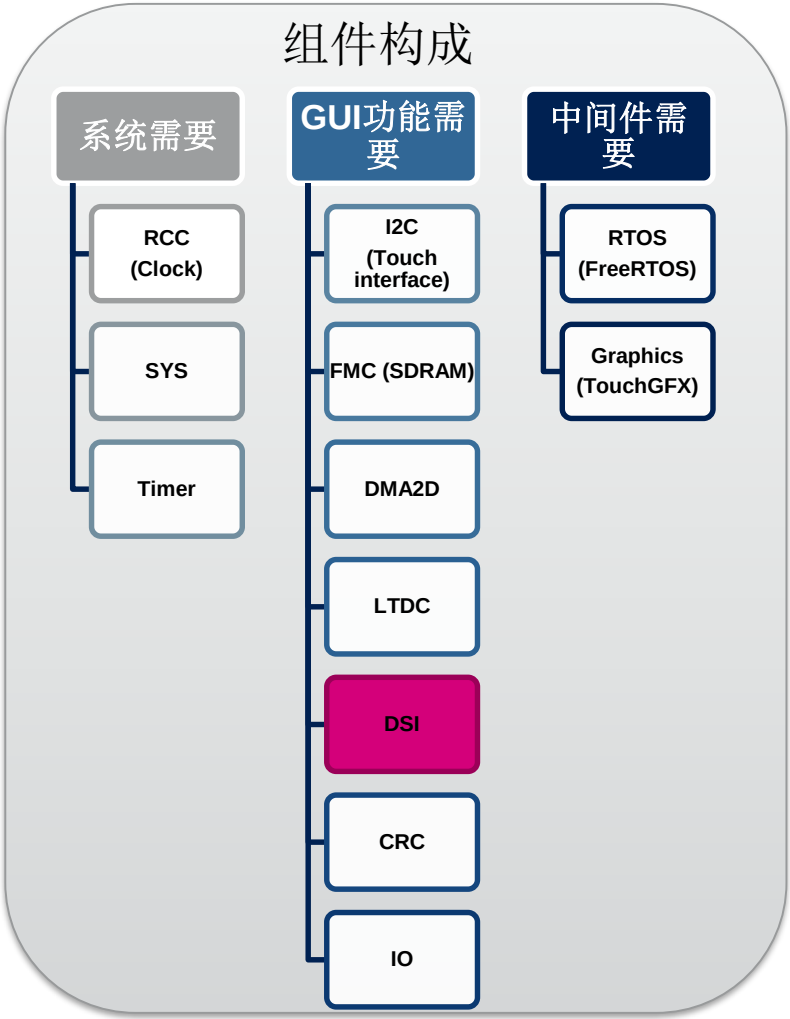
• 组件（GUI功能需要）

• DSI

• Display Interface（显示接口设置）



组件构成

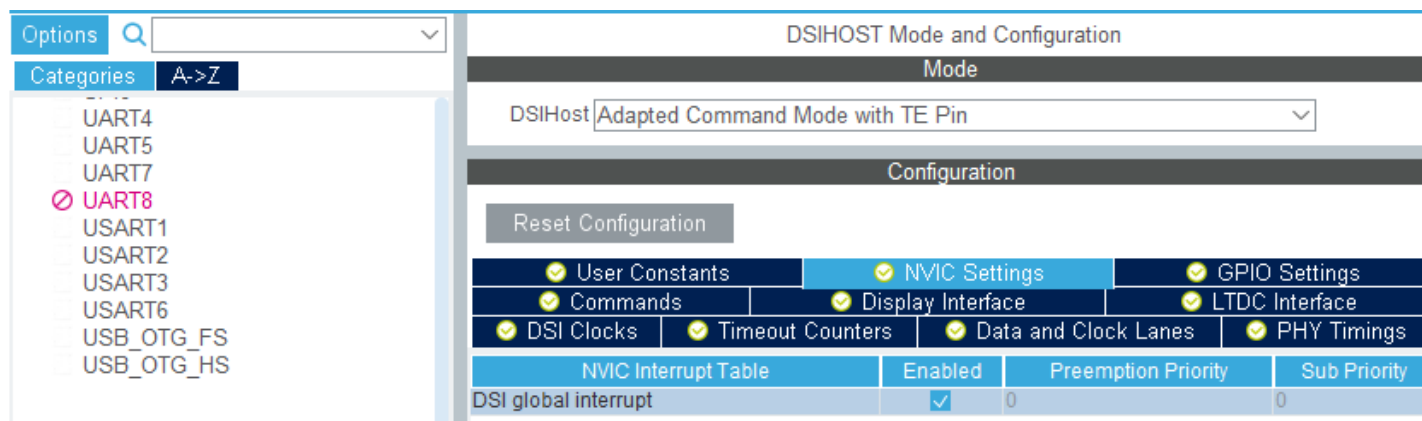




• 组件（GUI功能需要）

• DSI

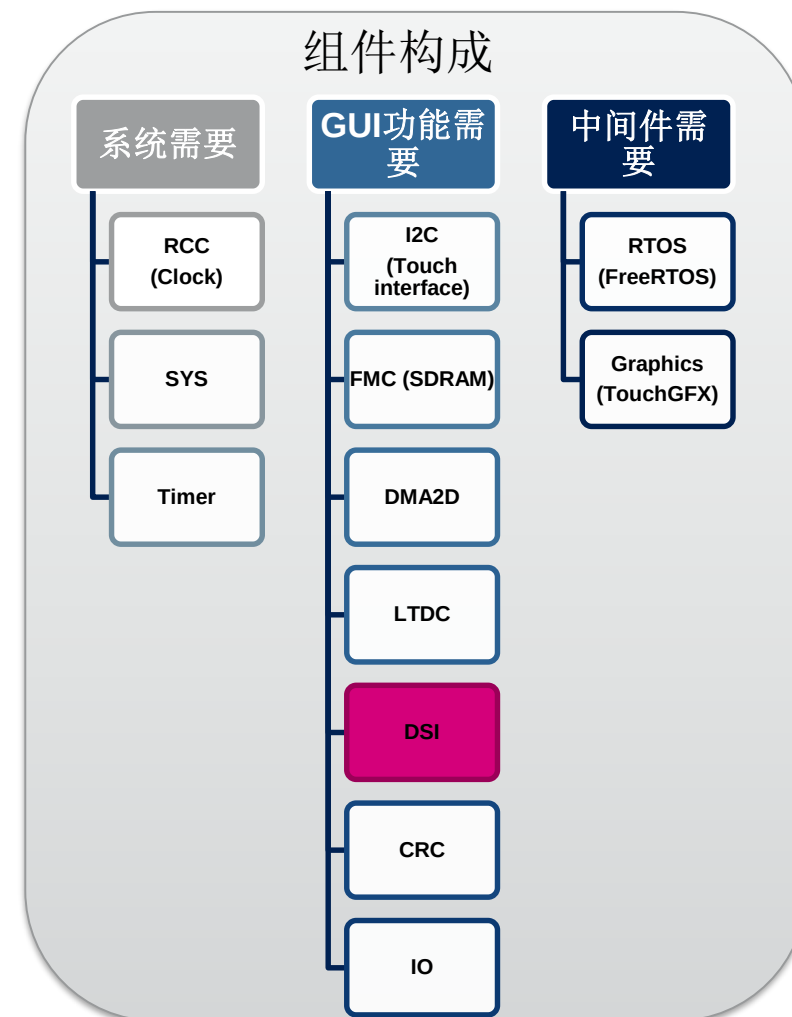
• NVIC Settings（中断配置）



实现示例（续）

54

组件构成





• 组件（GUI功能需要）

• DSI

- DSIHost 选择Adapted Command Mode with TE Pin

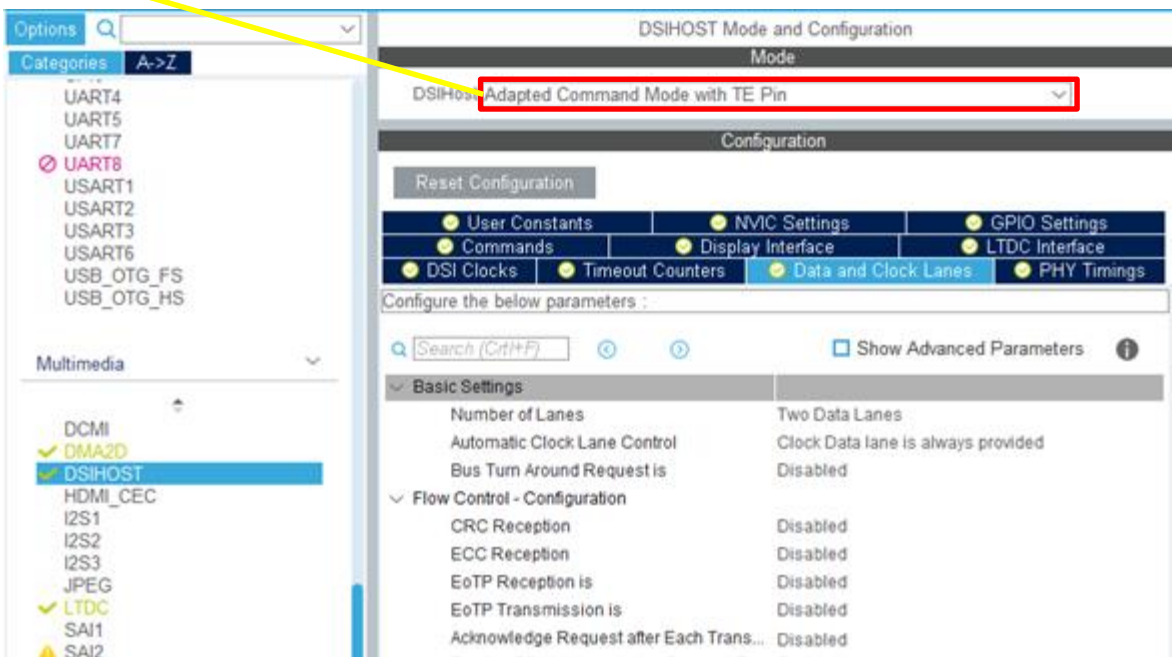
• DSI工作模式选择

- 1. STM32CubeMX v5.0.0中，如果选用TouchGFX，要求DSI 主控制器的工作模式为Adapted Command Mode with TE pin.

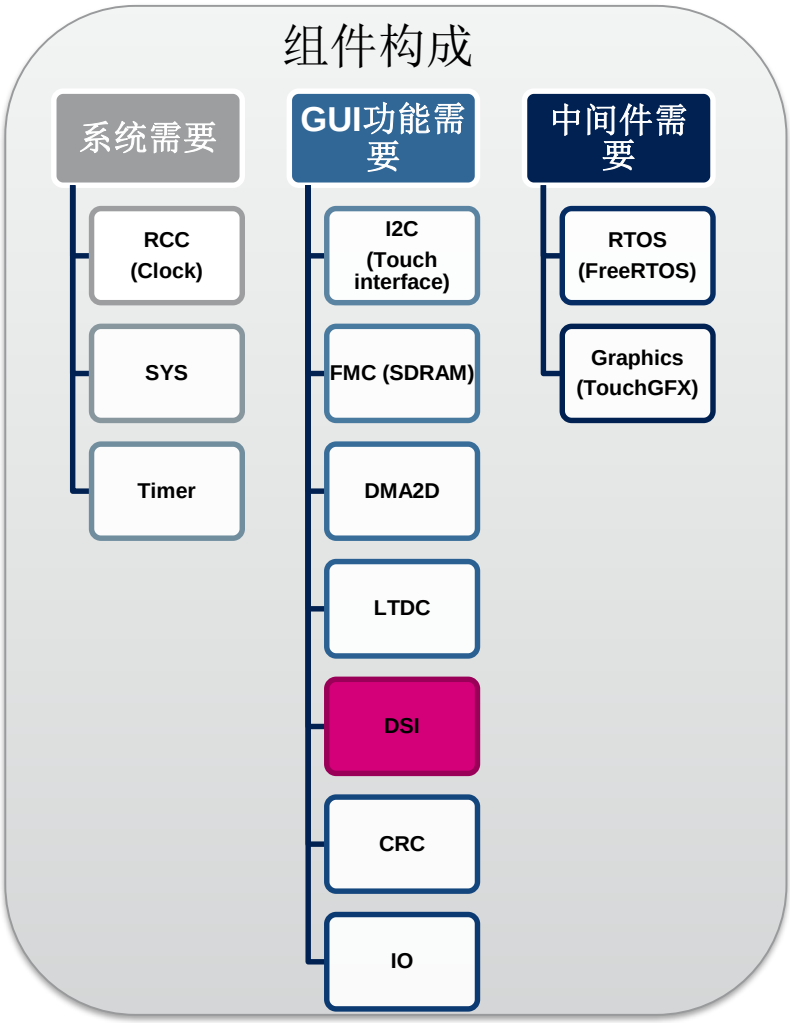
• 演示例中，采用STM32CubeMX+TouchGFX+DSI，这种开发组合，目前只能选择这种配置。

• 最高效更新GRAM

• TE信号能够实现GRAM刷新与显示刷新之间的同步



组件构成





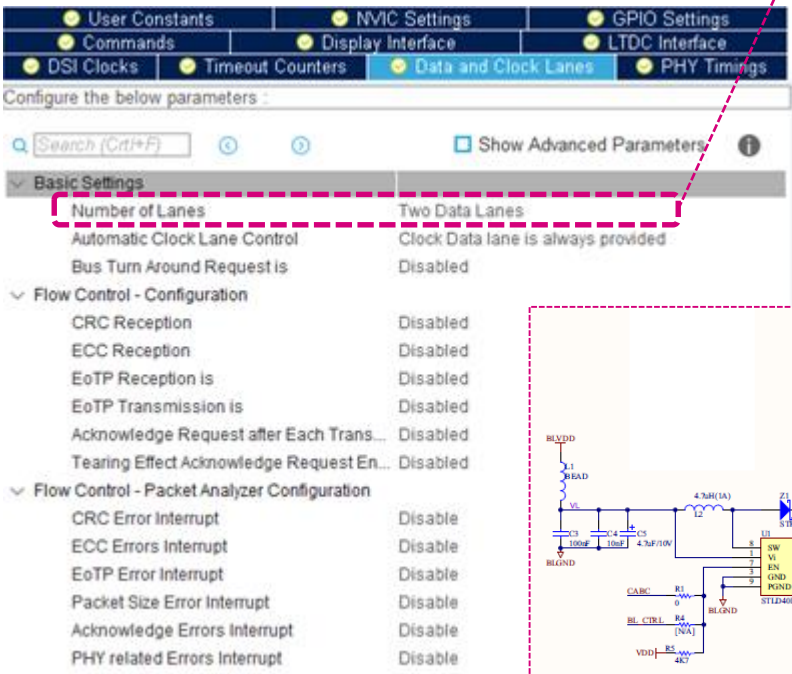
• 组件（GUI功能需要）

• DSI

- DSIHost 选择Adapted Command Mode with TE Pin

• 数据和时钟链配置

- 2. 根据STM32 DSI和显示驱动器的支持，以及硬件连接情况，再进行DSI数据速率要求的考虑基础上，选择数据链数。



数据链组数的选择考虑

- STM32 DSI支持1 或 2 数据链
- 演示例中，采用的显示驱动器otm8009a支持1 或 2 数据链
- 如下图，演示例中硬件连接两组数据链
- 考虑DSI数据速率要求。DSI数据速率的要求，由对最大像素时钟频率的要求决定。

- 在适配指令模式下，最大GRAM刷新时间受显示屏内部刷新频率的影响。为避免视觉伪影和撕裂，最长允许刷新时间必须小于1/(显示屏刷新频率)。

- 使用以下公式计算最长允许刷新时间：

最长刷新时间 = 1 / 显示屏刷新频率

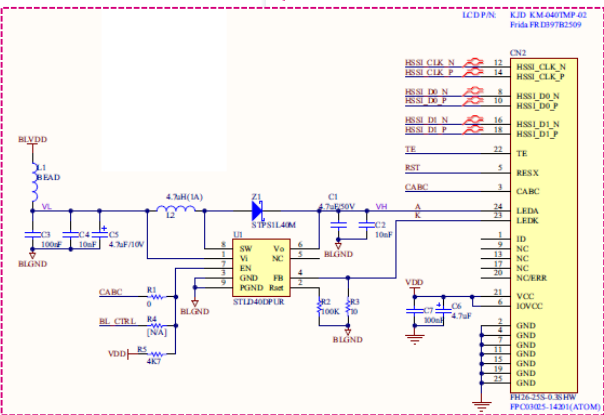
- 例如，演示例中，显示屏的内部显示刷新频率设置为65Hz，则最长允许内部帧缓存刷新时间为1/65 Hz（针对于演示例的实现机制，[详情参考](#)）。

- 可使用以下公式计算DSI链路BW：

DSI链路BW = FB大小 / 刷新时间 = HACT x VACT x 色深 / 刷新时间

- 使用最长刷新时间计算最小链路BW。它是可以避免显示屏侧视觉伪影的最小BW。

最小DSI链路BW = HACT x VACT x 色深 / 最长刷新时间





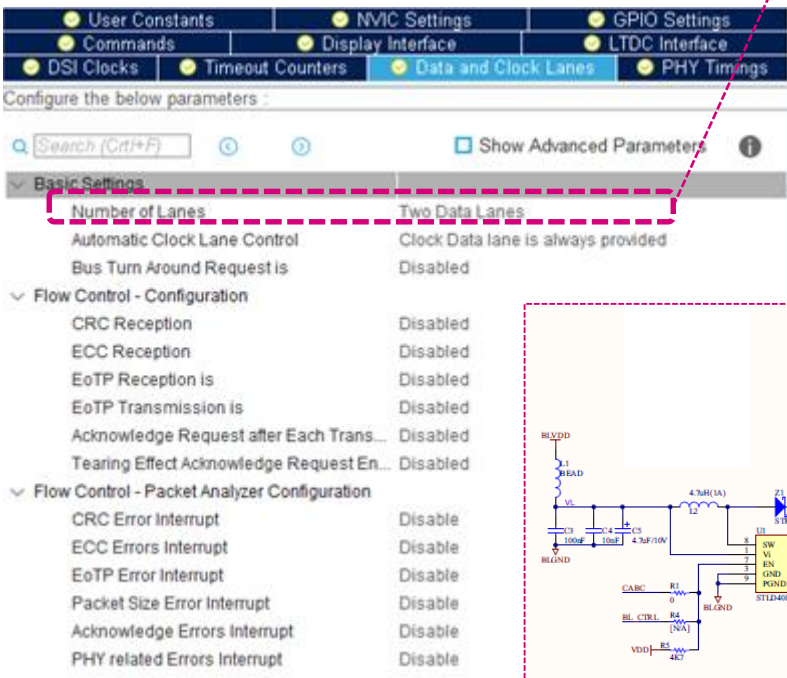
• 组件（GUI功能需要）

• DSI

- DSIHost 选择Adapted Command Mode with TE Pin

• 数据和时钟链配置

- 2. 根据STM32 DSI和显示驱动器的支持，以及硬件连接情况，再进行DSI数据速率要求的考虑基础上，选择数据链数。



数据链组数的选择考虑

- STM32 DSI支持1 或 2 数据链
- 演示例中，采用的显示驱动器otm8009a支持1 或 2 数据链
- 如下图，演示例中硬件连接两组数据链
- 考虑DSI数据速率要求。DSI数据速率的要求，由对最大像素时钟频率的要求决定。

- 在视频模式下，LTDC像素时钟和最小链路BW（带宽）受显示时序的影响。

- 可使用以下公式计算像素时钟：

像素时钟 = 总宽 x 总高 x 刷新频率

- 使用以下公式计算驱动显示屏所需的最小DSI链路BW：

最小DSI链路BW = 像素时钟 x 色深

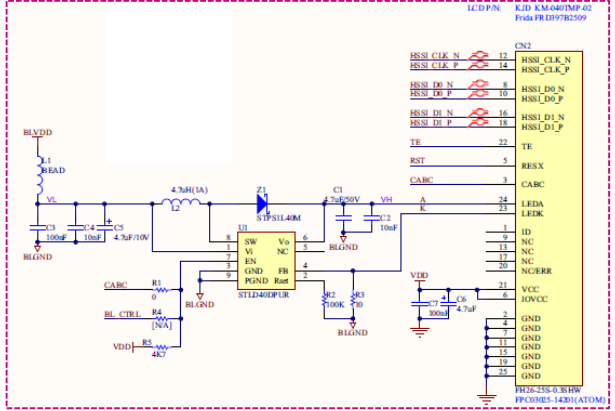
- 以具有以下时序的显示屏为例：

- HSA = 5, HBP = 35, HACT = 800, HFP = 35
- VSA = 2, VBP = 20, VACT = 480, VFP = 20
- 刷新速率 = 60 fps
- 色深 = 24 bpp

像素时钟 = 875 x 522 x 60 = 27.4 MHz

最小链路BW = 657 Mbit/s

- 这是驱动显示屏所需的最小链路BW。
- 由于最大通道速率为500 Mbit/s，仅使用一个通道无法驱动此显示屏。
- 可通过两个最小通道速率为328 Mbit/s的通道来驱动此显示屏。





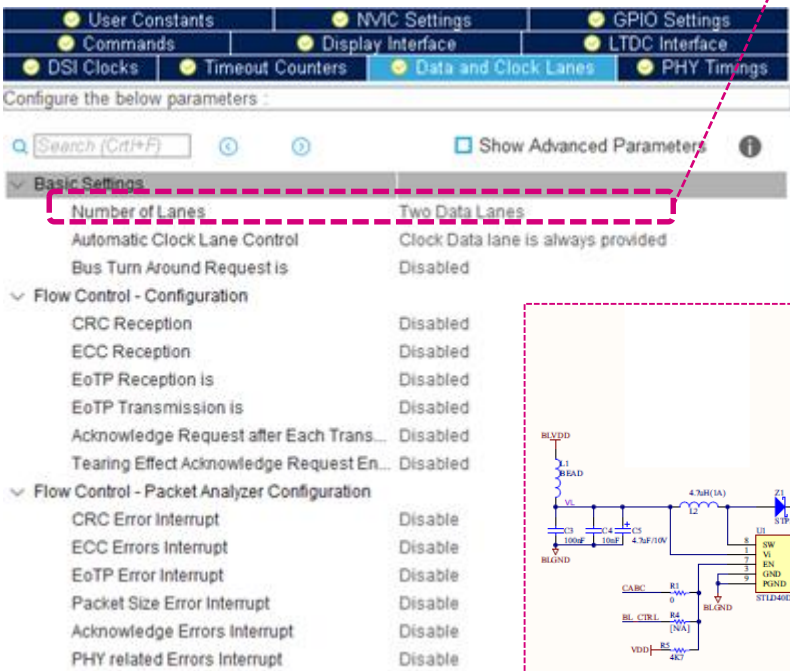
• 组件（GUI功能需要）

• DSI

- DSIHost 选择Adapted Command Mode with TE Pin

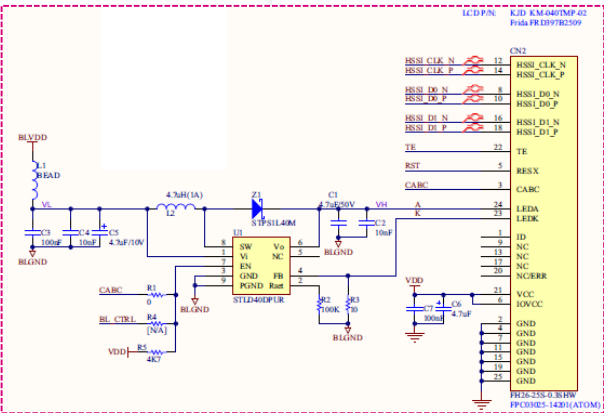
• 数据和时钟链配置

- 2. 根据STM32 DSI和显示驱动器的支持，以及硬件连接情况，再进行DSI数据速率要求的考虑基础上，选择数据链数。



数据链组数的选择考虑

- STM32 DSI支持1 或 2 数据链
- 演示例中，采用的显示驱动器otm8009a支持1 或 2 数据链
- 如下图，演示例中硬件连接两组数据链
- 考虑DSI数据速率要求。DSI数据速率的要求，由对最大像素时钟频率的要求决定。
 - 在适配指令模式下，最大GRAM刷新时间受显示屏内部刷新频率的影响。为避免视觉伪影和撕裂，最长允许刷新时间必须小于1/(显示屏刷新频率)。
 - 使用以下公式计算最长允许刷新时间：
$$\text{最长刷新时间} = 1 / \text{显示屏刷新频率}$$
 - 例如，演示例中，显示屏的内部显示刷新频率设置为65Hz，则最长允许内部帧缓存刷新时间为1/65 Hz（针对于演示例的实现机制，[详情参考](#)）。
- 可使用以下公式计算DSI链路BW：
$$\text{DSI链路BW} = \text{FB大小} / \text{刷新时间} = \text{HACT} \times \text{VACT} \times \text{色深} / \text{刷新时间}$$
- 使用最长刷新时间计算最小链路BW。它是可以避免显示屏侧视觉伪影的最小BW。
$$\text{最小DSI链路BW} = \text{HACT} \times \text{VACT} \times \text{色深} / \text{最长刷新时间}$$
- 演示例中，采用显示分辨率为480x800，色深为24 bpp，显示屏内部刷新频率为65 Hz：
$$\text{最小链路宽度} = 480 \times 800 \times 24 / (1/65) = 571.29 \text{ Mbit/s}$$
- STM32 DSI 1个数据通道（500Mbit/s），2个数据通道（1Gbit/s）
- 所以，采用2个数据链。





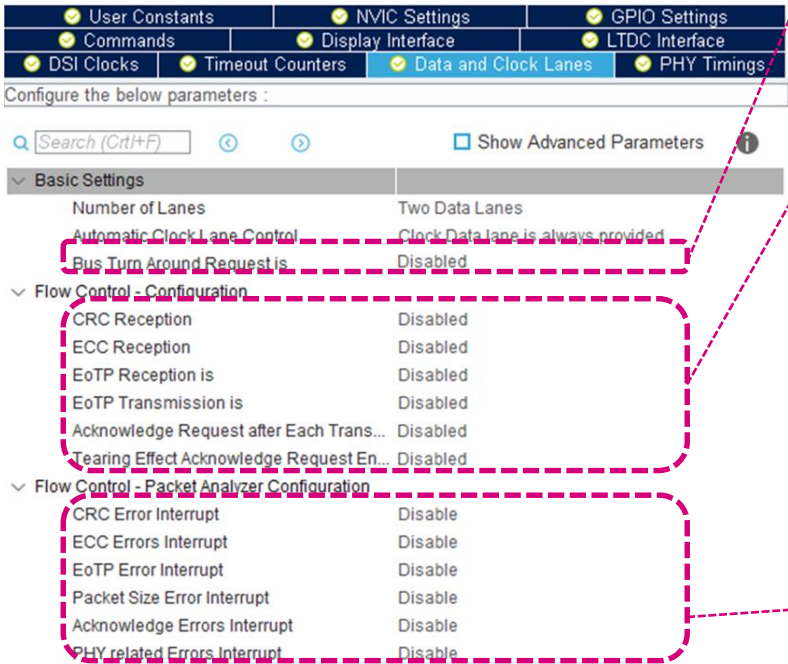
• 组件（GUI功能需要）

• DSI

- DSIHost 选择Adapted Command Mode with TE Pin

• 数据和时钟链配置

- 2. 根据STM32 DSI和显示驱动器的支持，以及硬件连接情况，再进行DSI数据速率要求的考虑基础上，选择数据链数。



• 总线反向请求。

- 需要反向通信（例如读取、回应和撕裂效应请求）时，必须使能总线转向。

• 协议流控。DSI主机提供流控功能，包括EoTp发送和接收、ECC和CRC接收以及总线转向控制。

• Acknowledge Request after Each Transmission

- 根据应用需要，考虑是否需要响应请求。这个特性被用来获取响应、检测错误、增加系统安全性。
- 使能后，DSI 主控制器在每个发送命令后，插入BTA处理。将总线控制器交予显示器。显示器返回响应或者错误信息。然后再将总线控制权归还
- 避免在适配指令模式的刷新过程中，使用这个特性。因为这样会增加大量开销，降低刷新速率。

• 采用撕裂效应回应请求时，Tearing Effect Acknowledge Request Enable需要使能。

• 错误中断配置

- 错误中断各项的详细介绍，请参考对应STM32的参考手册





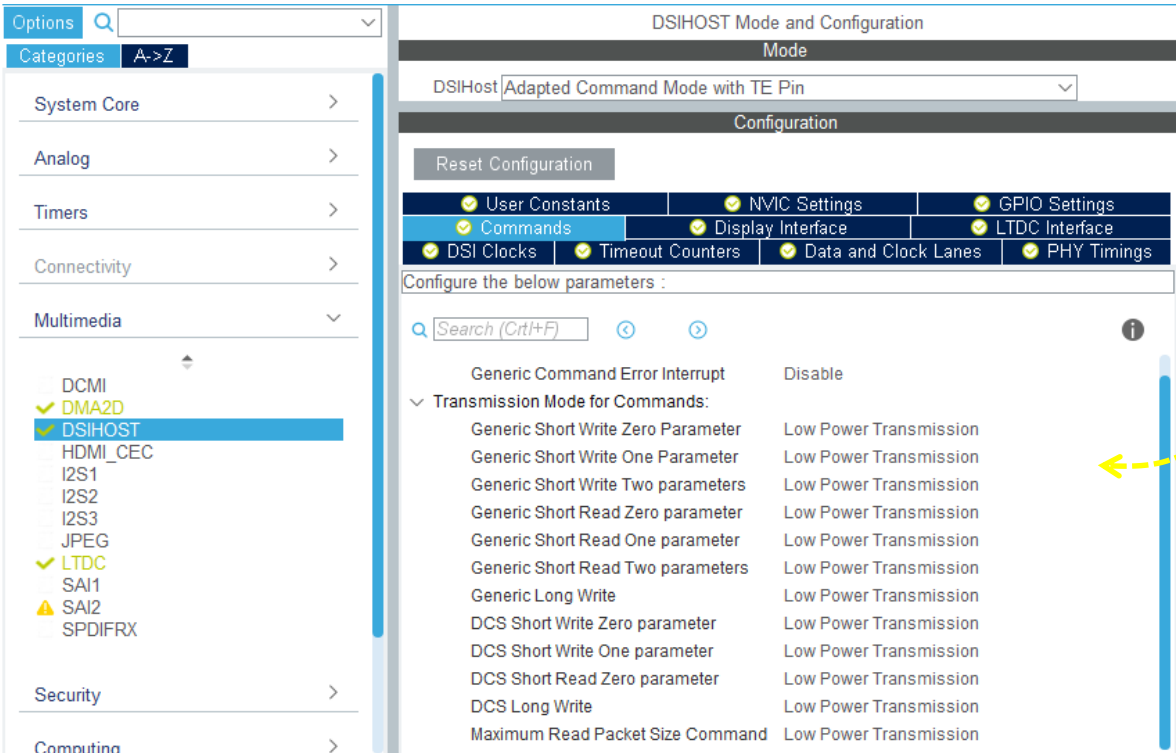
• 组件（GUI功能需要）

• DSI

Command	(Hex)	Write/Read/Command	Function	Parameter Number	MIPI Transmission Mode
ADRSFT	0000	W	Address Shift Function	1	LPDT
CMD2_ENA1	FF00	W	Enable Access Command2 "CMD2"	3	LPDT
CMD2_ENA2	FF80	W	Enable Access Orise Command2	2	LPDT
OTPSEL	A000	W/R	OTP Select Region	1	LPDT

部分命令表（摘自OTM8009A手册）

- Commands配置（命令参数设置）
- 根据显示驱动器OTM8009A中所述，有些命令仅支持LPDT（Low-power Mode）。简便考虑，命令统一设置为Low-power传输模式。



组件构成

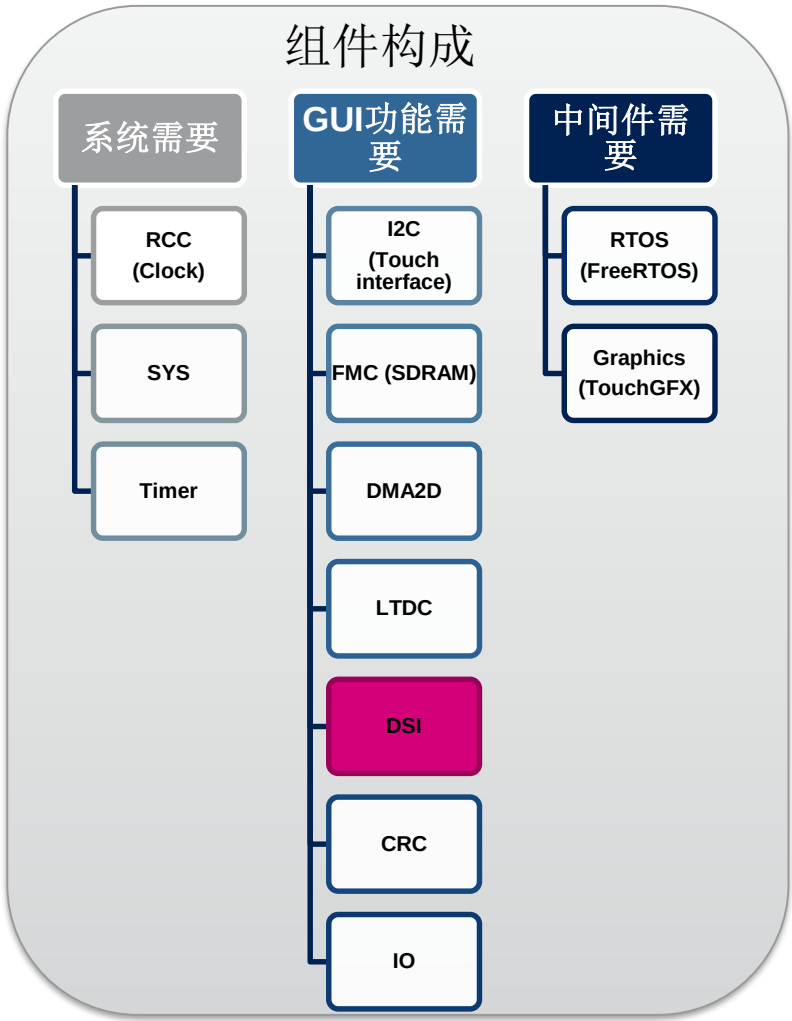


图. SDI Commands配置（@STM32CubeMX）



实现示例（续）

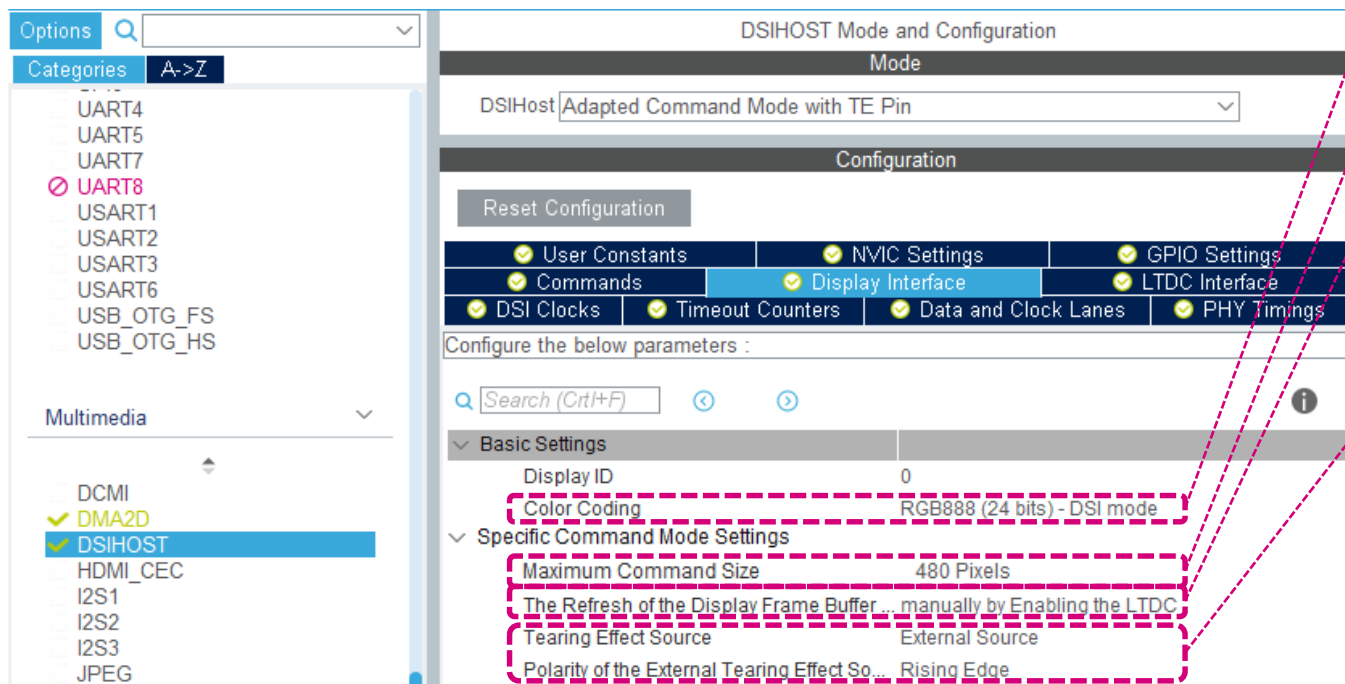
62



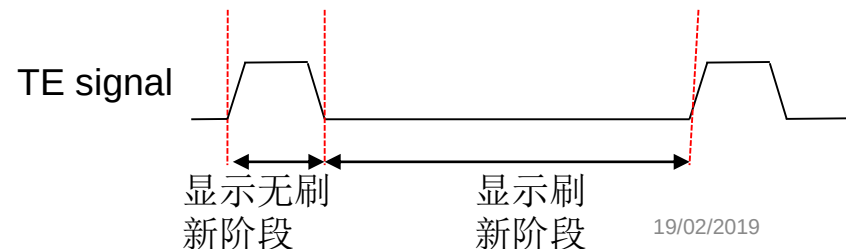
• 组件（GUI功能需要）

• DSI

• Display Interface（显示接口设置）



- 在STM32CubeMX中，颜色格式自动与LTDC的显示类型对应。
- 最大显示大小，自动关联为LTDC中Active Width大小。可根据需要调整。
- 帧缓存刷新模式。支持两种刷新模式。手动刷新可以自定义刷新时刻。演示例中，采用用户选择刷新。
 - 自动刷新。在TE事件时自动刷新
 - 手动刷新。用户选择执行刷新
- TE信号源及触发沿。STM32 DSI和OTM8009A支持两种TE信号源。演示例中，采用TE信号线。
 - 通过链路，无需任何额外的引脚。
 - 通过TE信号线。
- OTM8009A 输出TE信号如下图。为了在停止刷新时刻开始更新GRAM，选择上升沿触发。





实现示例（续）

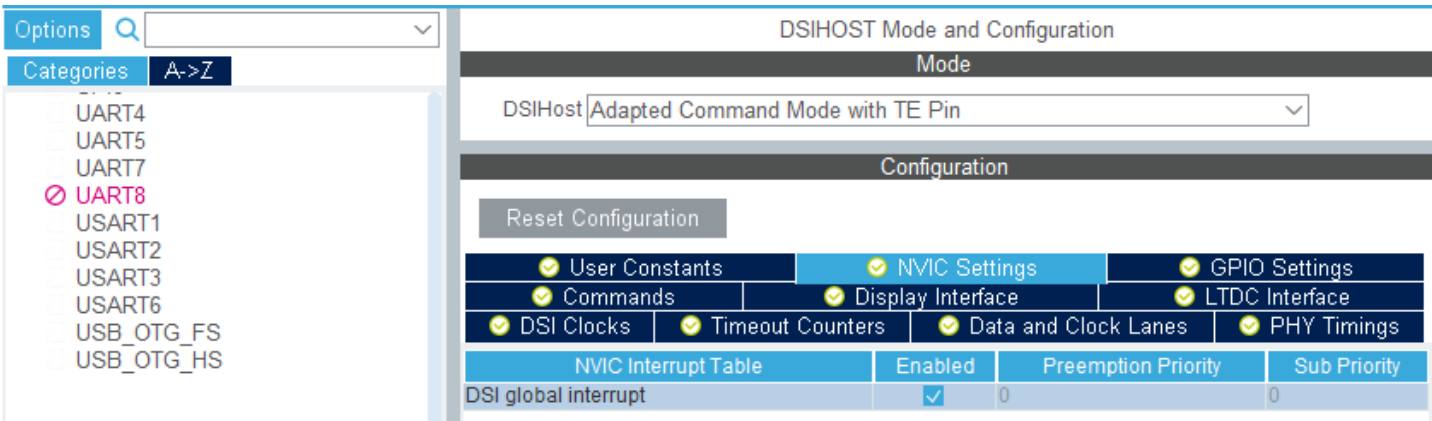
63

• 组件（GUI功能需要）

• DSI

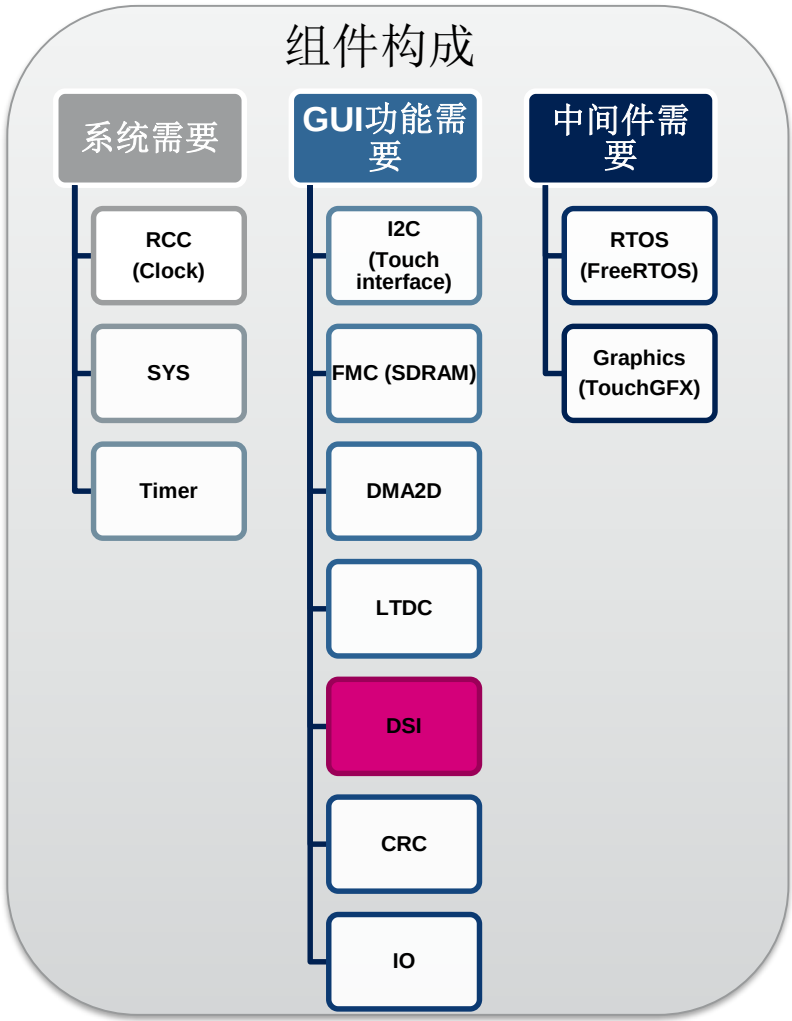
• NVIC Settings（中断配置）

• 使能中断。在中断服务函数中，实现对GRAM刷新等操作。



还有一些保持默认的配置参数没有进行介绍。更多详细内容请参考STM32参考手册。

组件构成





实现示例（续）

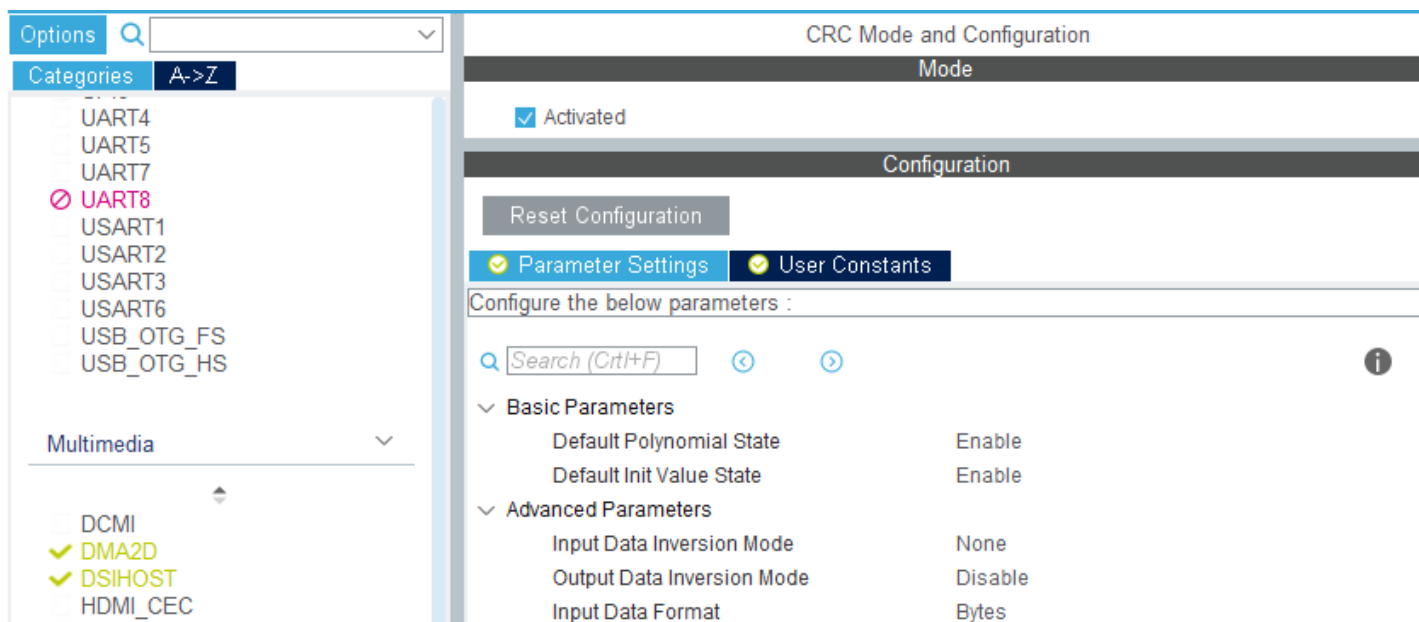
64



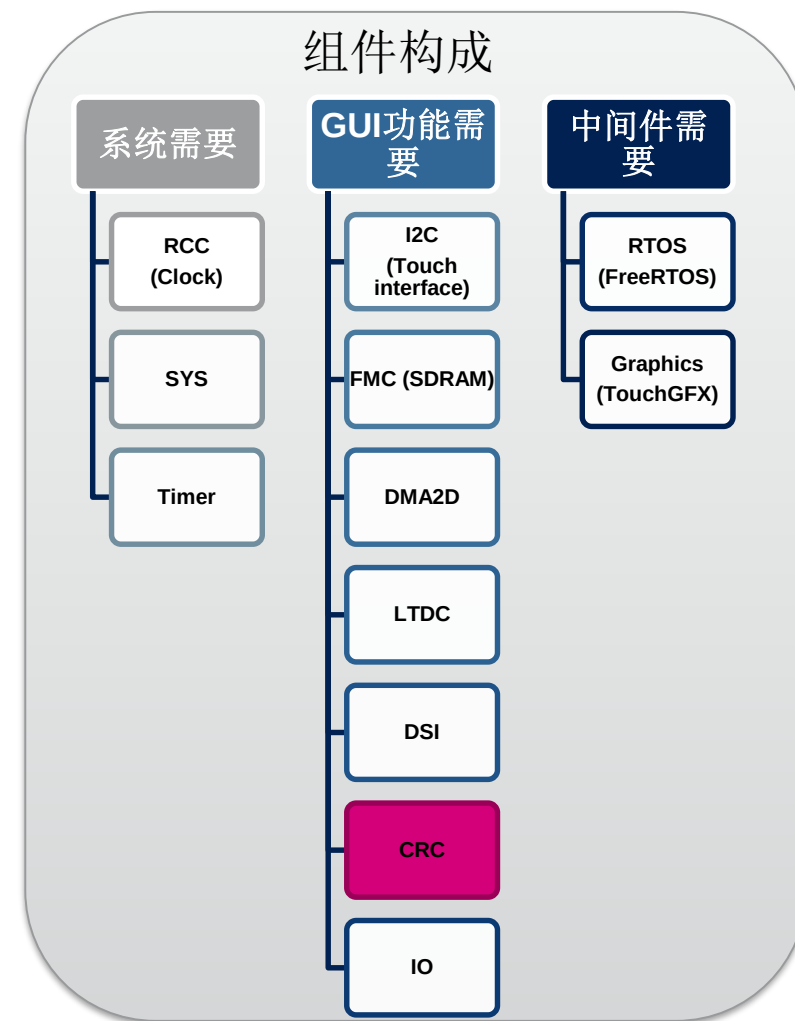
• IP组件（GUI功能需要）

• CRC

- TouchGFX中间件需要。保持默认值。



组件构成

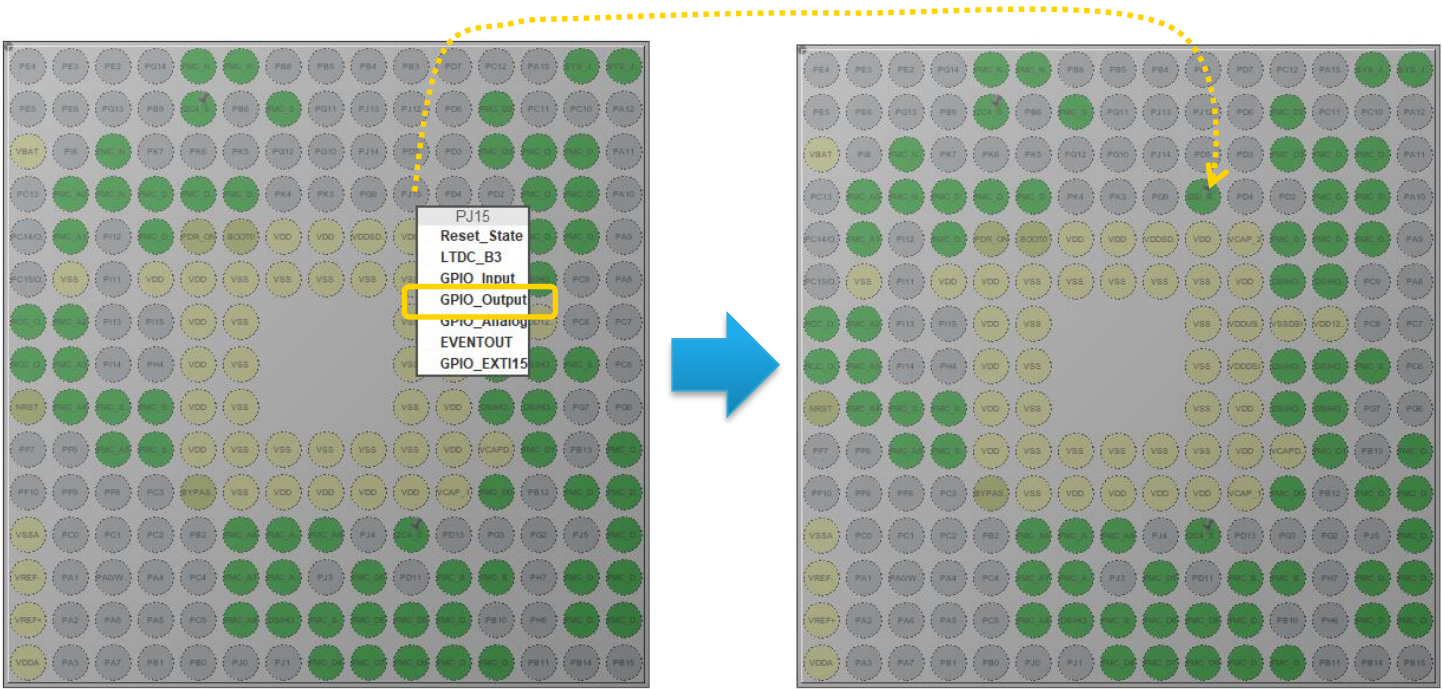




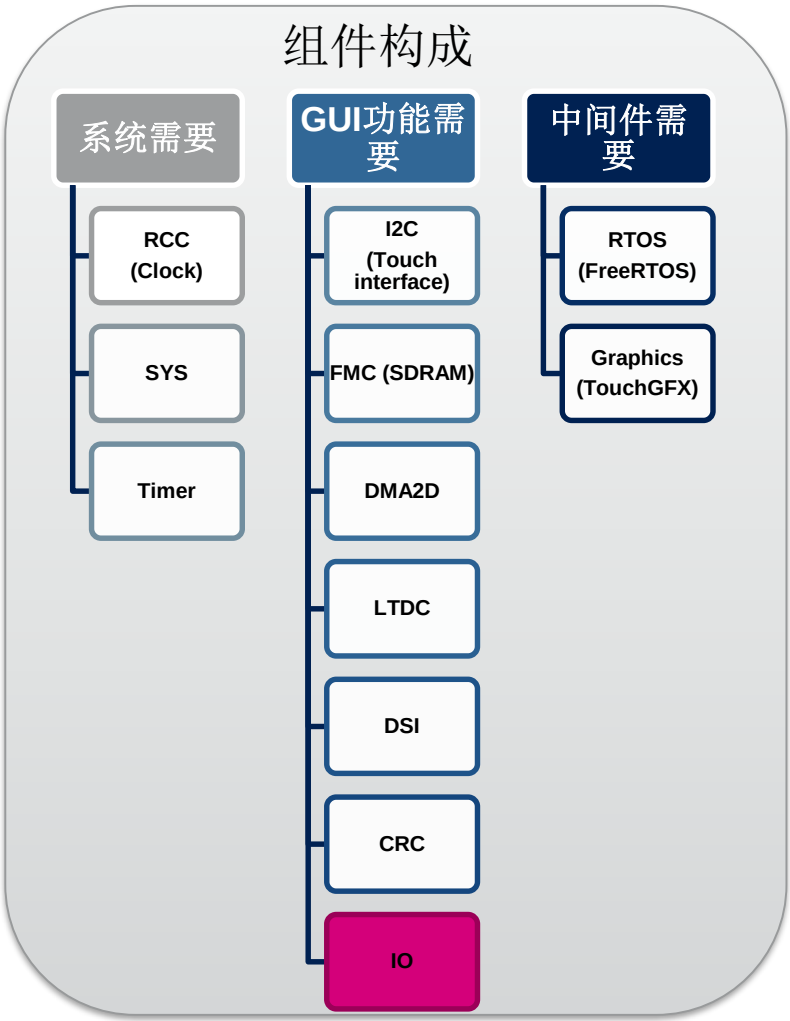
• 组件（GUI功能需要）

• IO

- 配置DSI屏复位信号线（根据显示屏需要和硬件连接，STM32F769I-Disco板上显示屏复位信号连接在STM32的PJ15）



组件构成

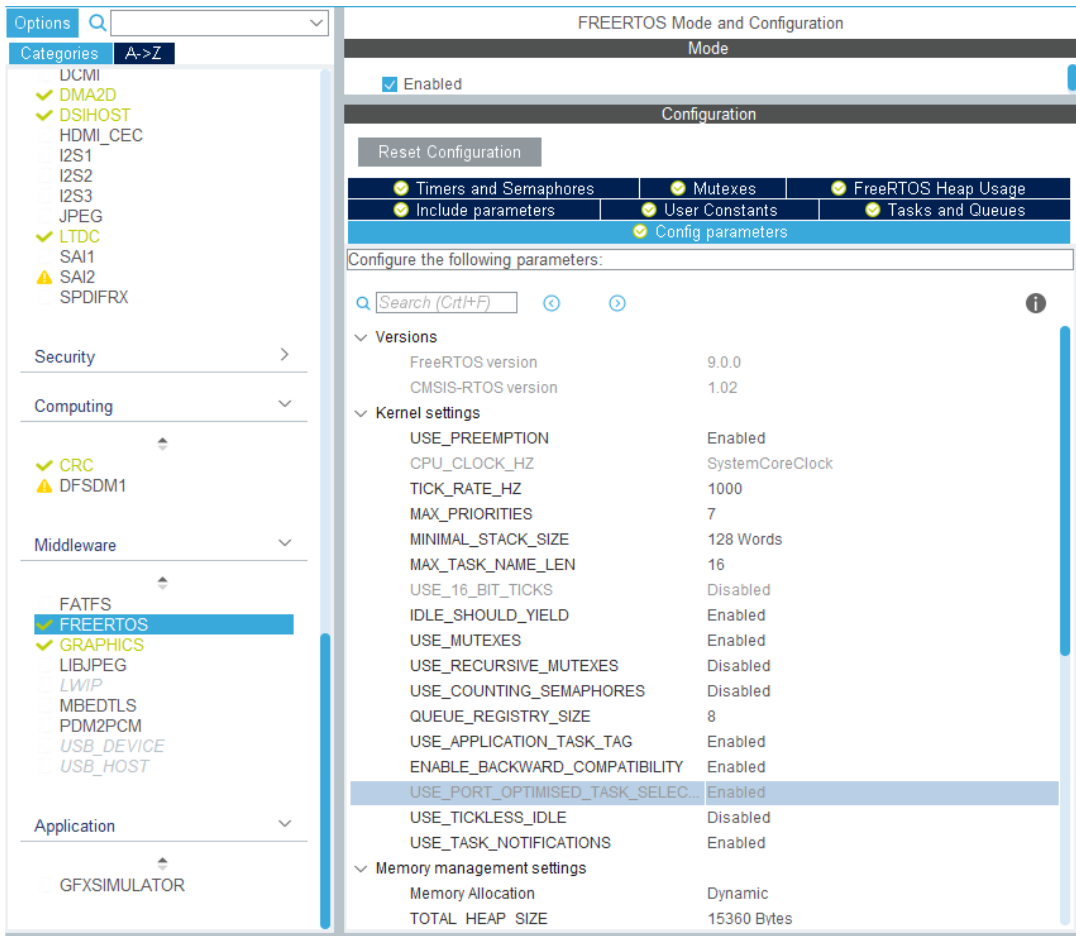




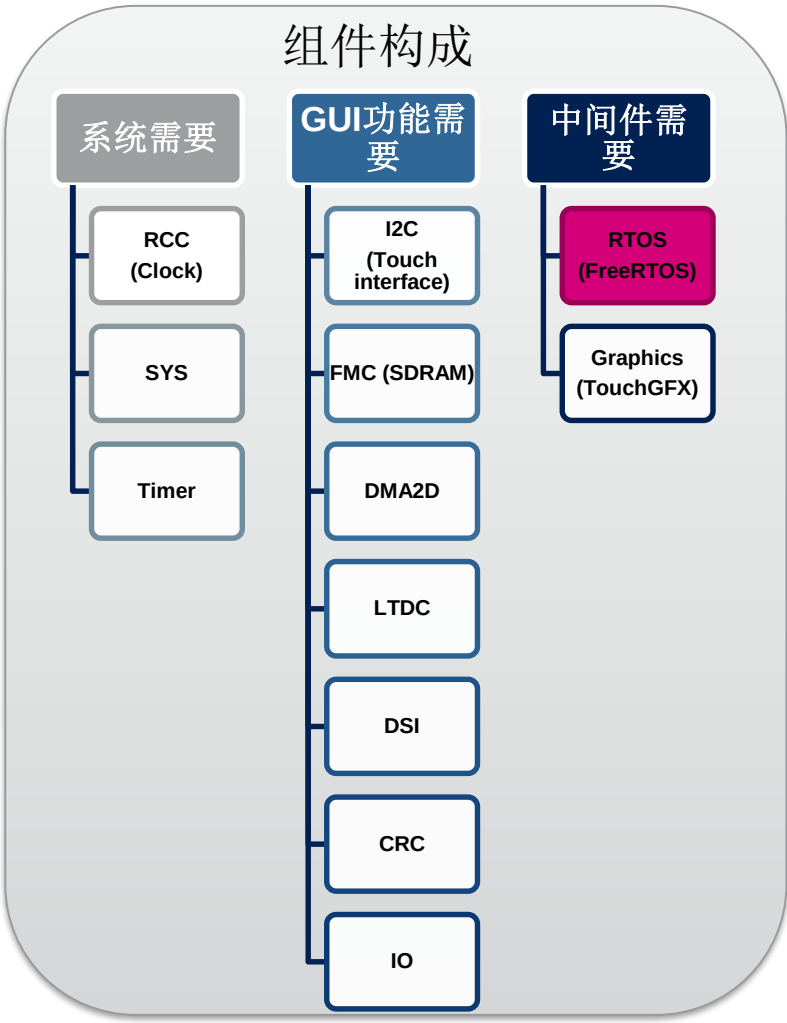
• 组件（中间件需要）

• FreeRTOS

• STM32CubeMX对TouchGFX开发支持，需要使用RTOS。



组件构成

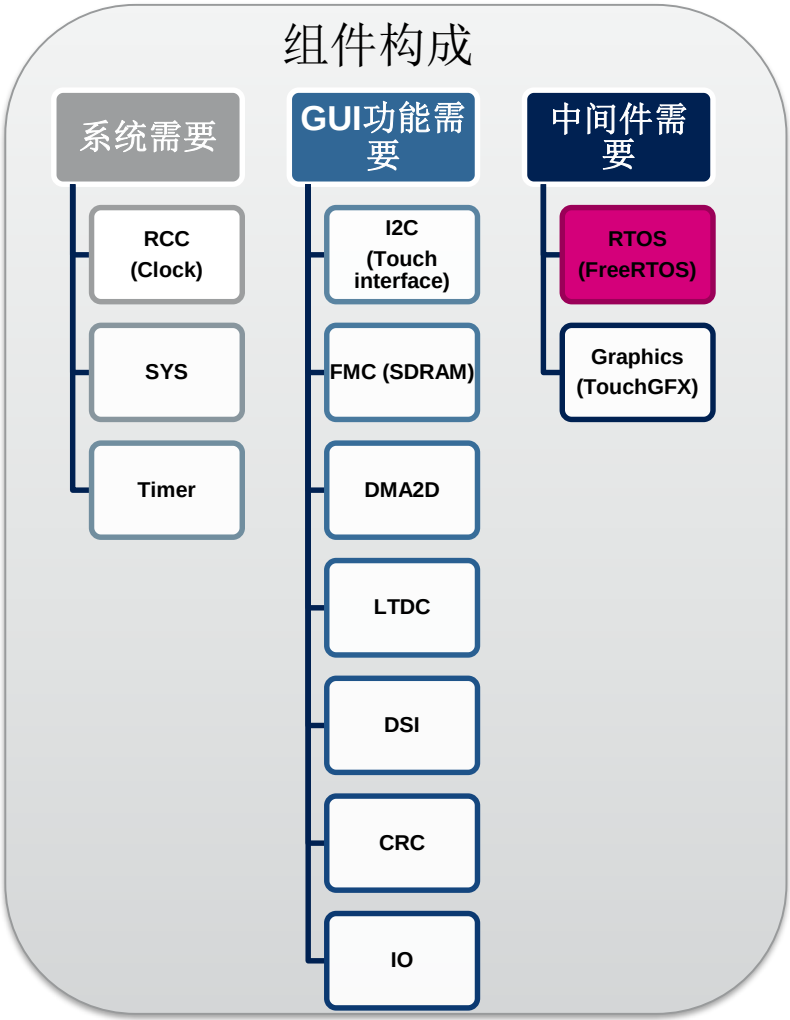
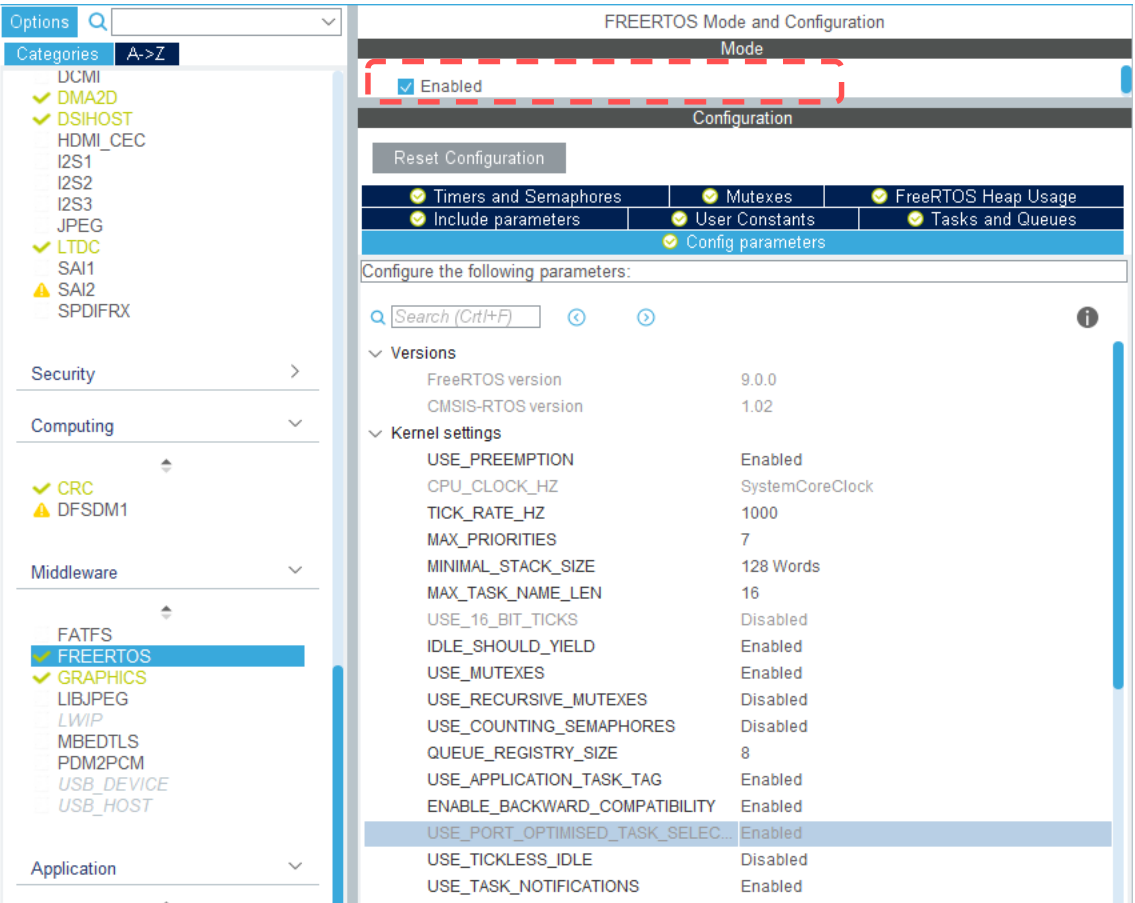




• 组件（中间件需要）

• FreeRTOS

- 1. 使能FreeRTOS。
 - STM32CubeMX对TouchGFX开发支持，需要使用RTOS。





实现示例（续）

68



• 组件（中间件需要）

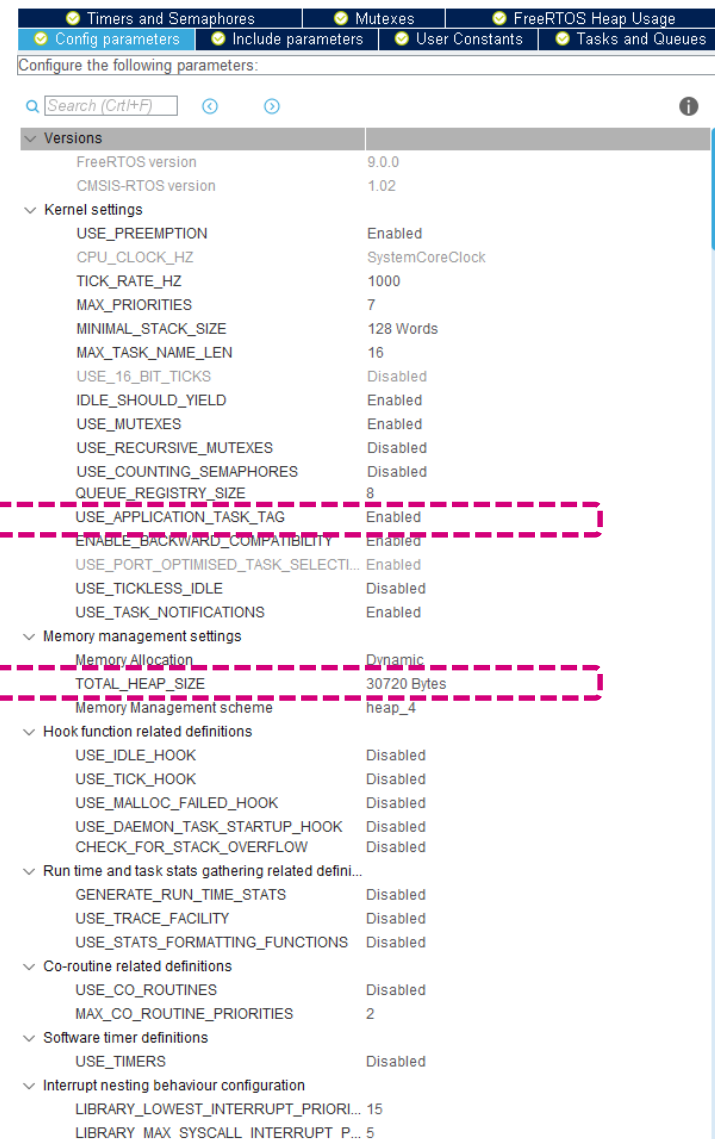
• FreeRTOS

• 2. 配置参数。

- 在使能FreeRTOS时，已经默认了一些常用配置。在这些配置基础上，针对应用情况，进行调整。
- 这里对调整部分进行介绍，其他部分请参考FreeRTOS帮助文档（[官网链接](#)）

- 需要使能USE_APPLICATION_TASK_TAG，测试MCU负载会涉及到这个功能的使用。

- 根据应用需要，调整TOTAL_HEAP_SIZE。调整分配给FreeRTOS的RAM空间大小。





实现示例（续）

69



• 组件（中间件需要）

• FreeRTOS

- 2. 配置包含参数。保持默认（涉及到其他应用功能时，根据需要进行调整）。

☒ Mutexes	☒ FreeRTOS Heap Usage
☒ Tasks and Queues	☒ Timers and Semaphores
☒ Config parameters	☒ Include parameters
	☒ User Constants

Configure the following parameters:

Search (Ctrl+F)

Include definitions

vTaskPrioritySet	Enabled
uxTaskPriorityGet	Enabled
vTaskDelete	Enabled
vTaskCleanUpResources	Disabled
vTaskSuspend	Enabled
vTaskDelayUntil	Disabled
vTaskDelay	Enabled
xTaskGetSchedulerState	Enabled
xTaskResumeFromISR	Enabled
xQueueGetMutexHolder	Disabled
xSemaphoreGetMutexHolder	Disabled
pcTaskGetTaskName	Disabled
uxTaskGetStackHighWaterMark	Disabled
xTaskGetCurrentTaskHandle	Disabled
eTaskGetState	Disabled
xEventGroupSetBitFromISR	Disabled
xTimerPendFunctionCall	Disabled
xTaskAbortDelay	Disabled
xTaskGetHandle	Disabled



实现示例（续）

70



• 组件（中间件需要）

• FreeRTOS

• 3. 增加任务堆栈空间

- 调整Stack Size空间大小

Task N...	Priority	Stack ...	Entry F...	Code ...	Param...	Allocati...	Buffer ...	Contro...
default...	osPrio...	4096	StartD...	Default	NULL	Dyna...	NULL	NULL

Queue Name	Queue Size	Item Size	Allocation	Buffer Name	Control Blo...
------------	------------	-----------	------------	-------------	----------------



Task Name: defaultTask

Priority: osPriorityNormal

Stack Size (Words): 4096

Entry Function: StartDefaultTask

Code Generation Option: Default

Parameter: NULL

Allocation: Dynamic

Buffer Name: NULL

Control Block Name: NULL

OK Cancel



实现示例（续）

71

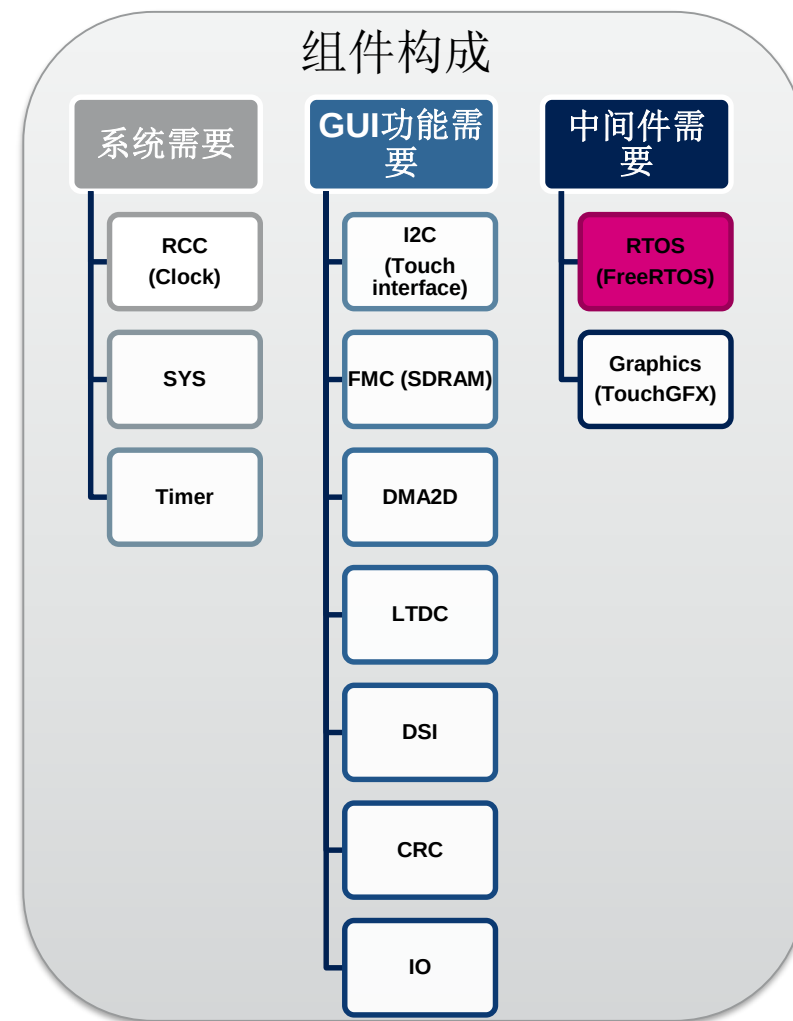


• 组件（中间件需要）

• FreeRTOS

- 除上述介绍配置，FreeRTOS还有其他可供配置项。保持默认（涉及到其他应用功能时，根据需要进行调整）。
- 更多详情可参考
 - [UM1718 STM32CubeMX for STM32 configuration and initialization C code generation](#)
 - [STM32 FreeRTOS培训：FreeRTOS基本原理介绍](#)
 - [STM32 RTOS培训:STM32嵌入式操作系统介绍](#)
 - [FreeRTOS ST April Training](#)
 - [FreeRTOS官网](#)

STM32Cube软件包中，提供了多个FreeRTOS示例可供参考。
[STM32CubeF7获取链接](#)





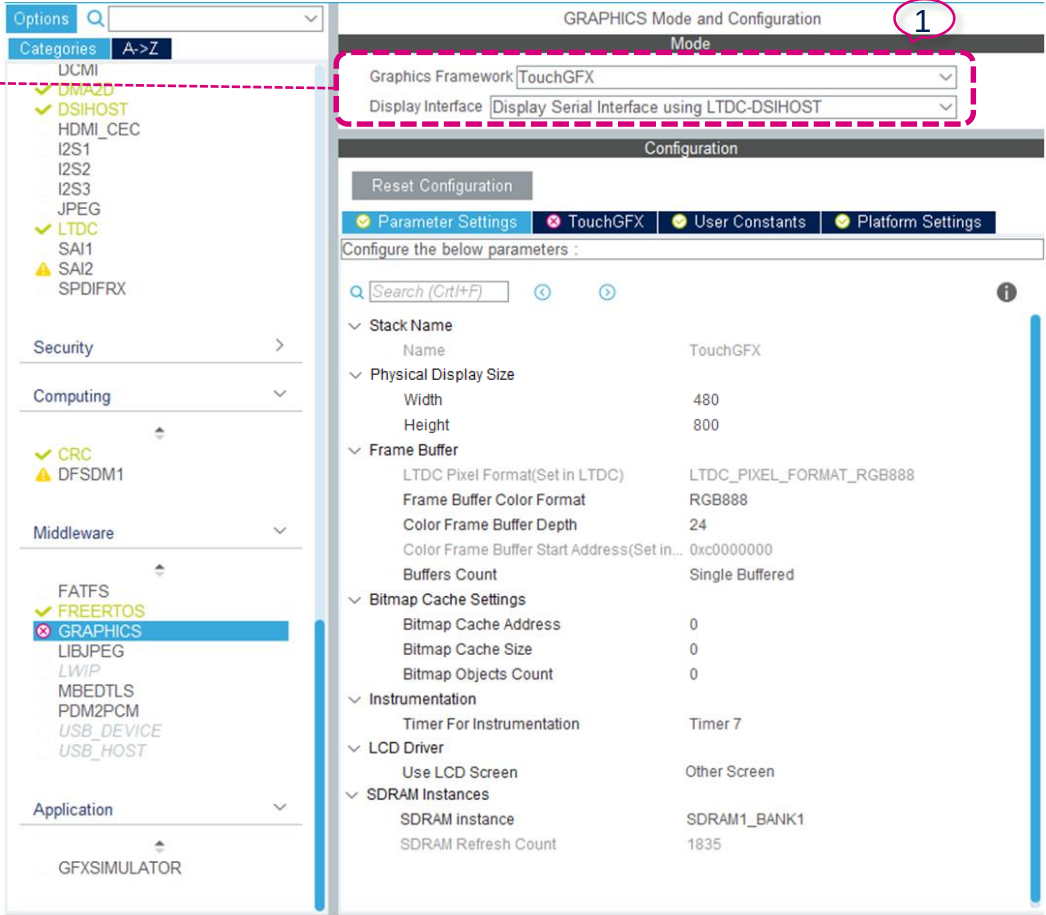
实现示例（续）



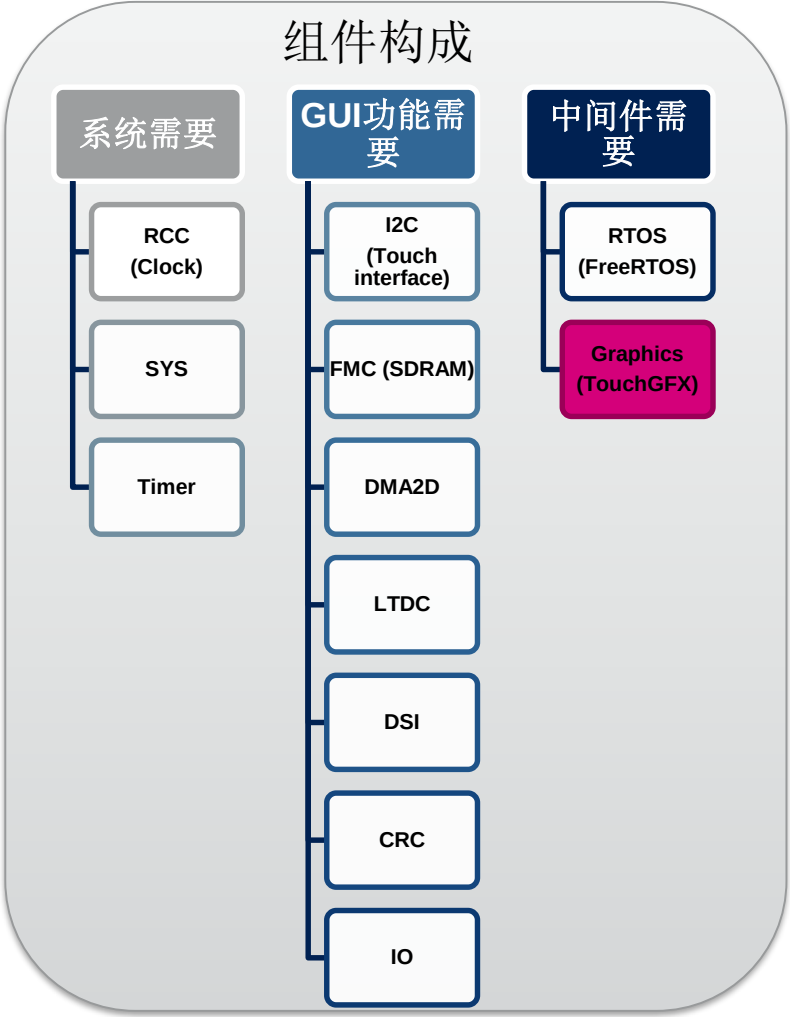
- 组件（GUI功能需要）
 - TouchGFX（GUI中间件）
 - Parameter Settings (参数设置)

- 选择TouchGFX中间件
- 选择显示器接口

- RGB接口屏选择LTDC
- DSI接口屏选择LTDC-DSIHOST
- 8080/6800接口屏选择FMC
- 注：演示例中，STM32F769I-DISCO连接DSI接口屏，选择LTDC-DSIHOST



组件构成



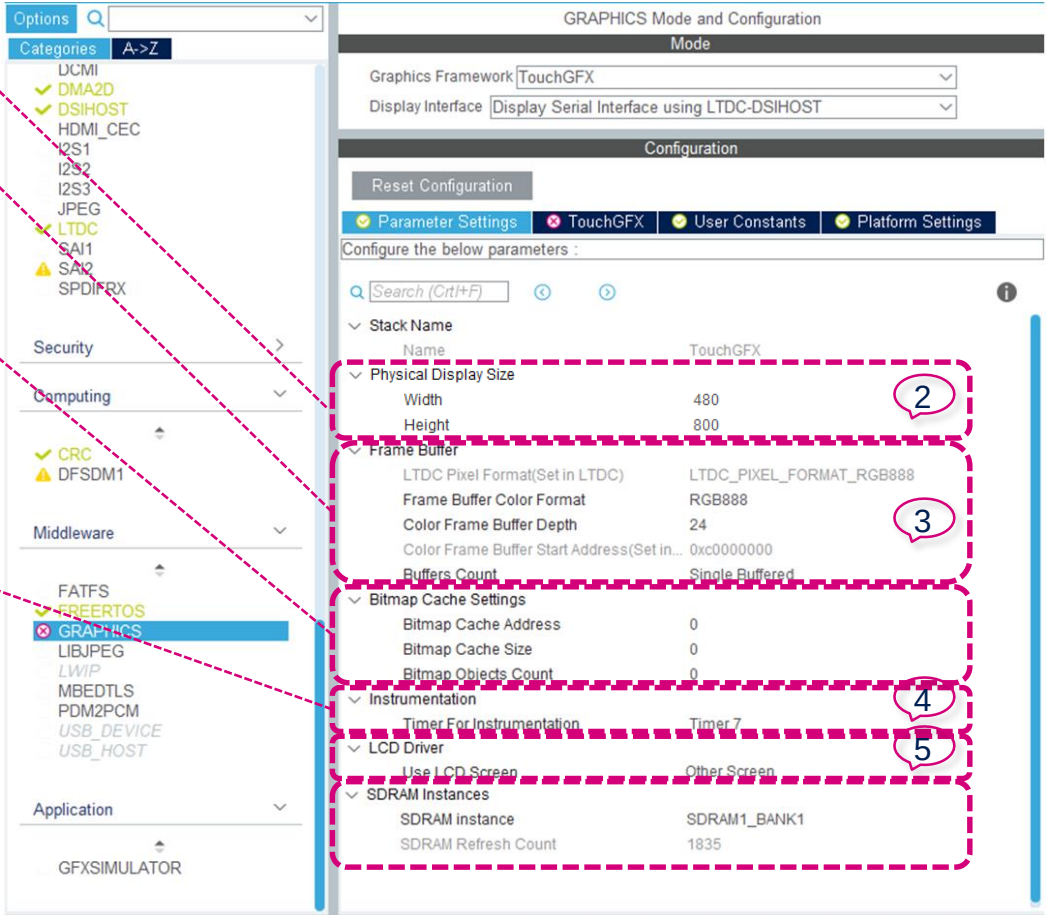


• 组件（GUI功能需要）

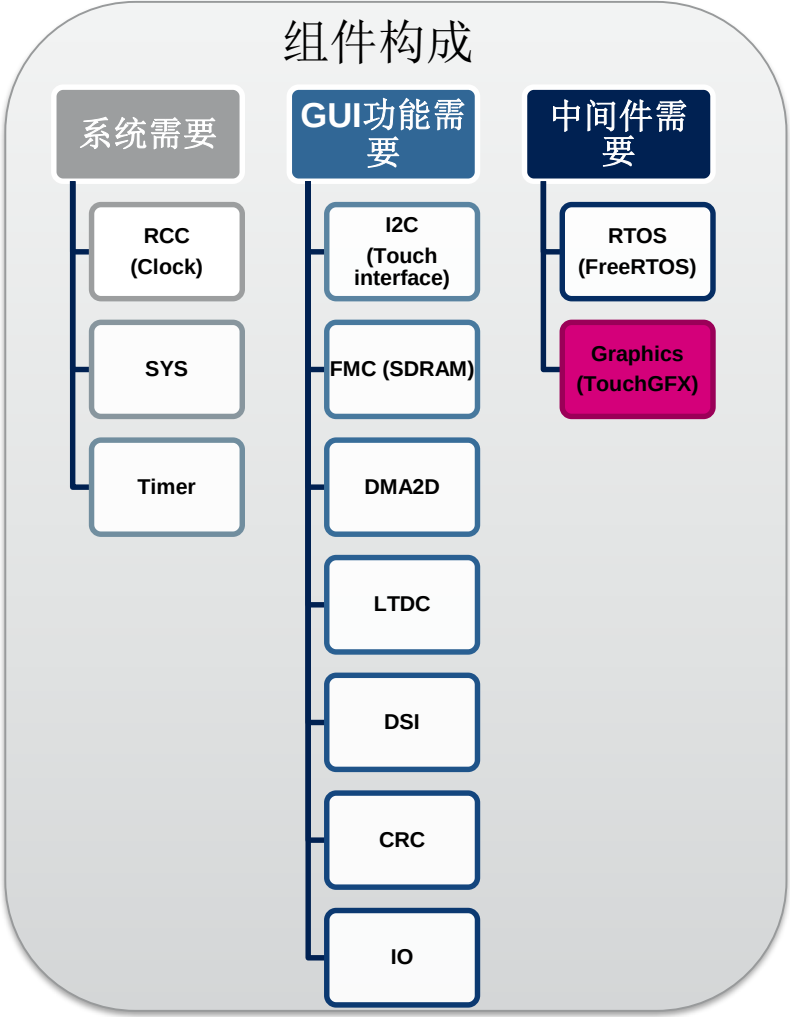
• TouchGFX（GUI中间件）

• Parameter Settings (参数设置)

- 对应为显示屏分辨率
- 设置中间件帧缓存
 - 帧缓存中颜色格式自定义关联至DSI接口颜色格式
- 设置位图缓存
 - 适用于位图存储于非存储器映射的存储空间时。
 - 演示例中没有用到，保持默认。
- 需要提供一路定时器
 - 为RTOS获取MCU负载，提供计数单元。用于在获取空闲任务外MCU主时钟消耗情况，进而获得MCU负载信息。



组件构成

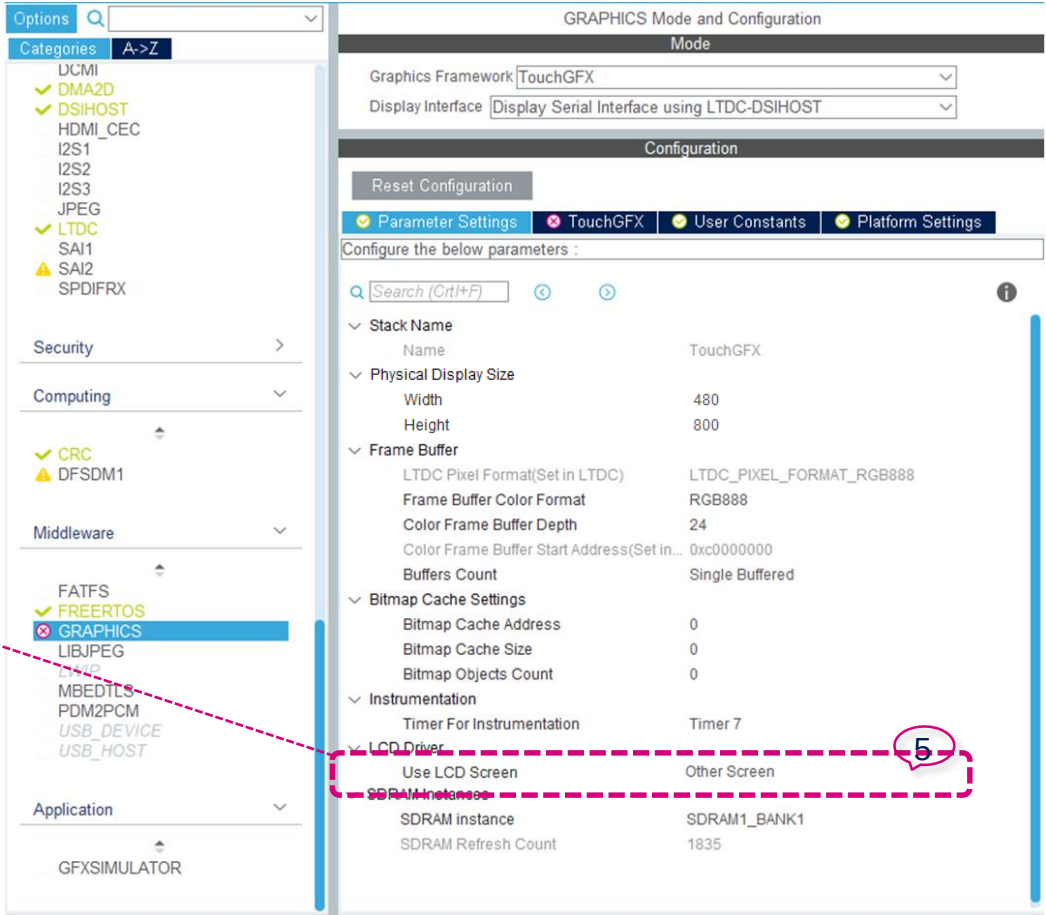




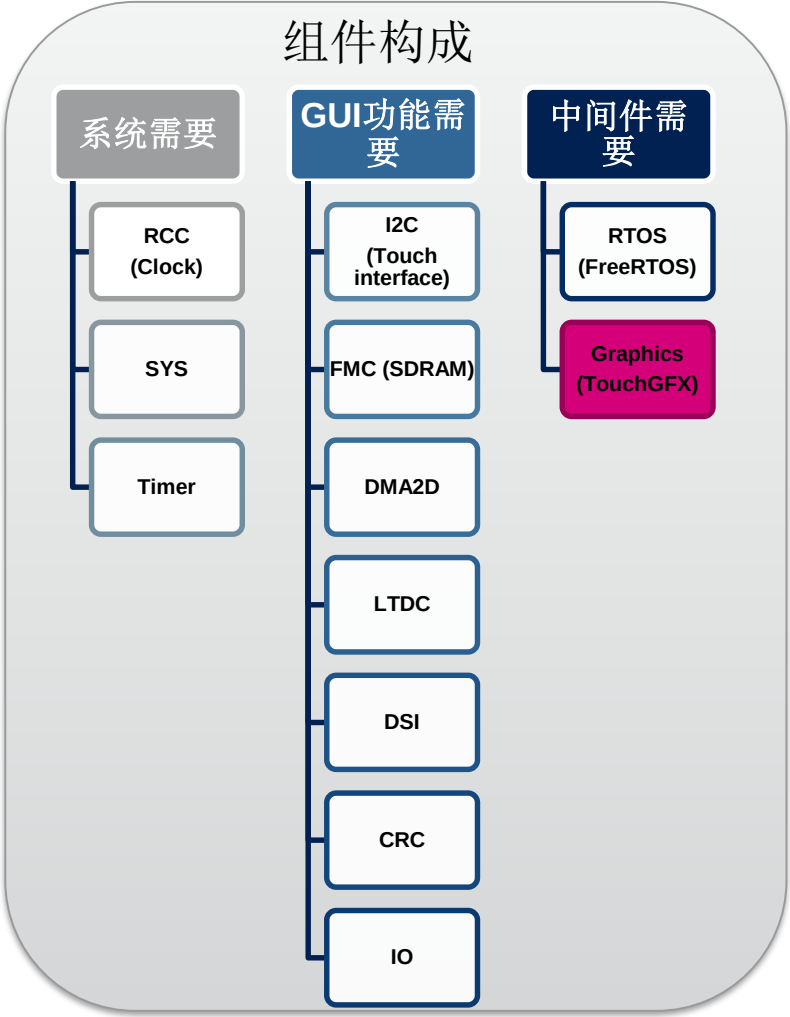
- 组件（GUI功能需要）
 - TouchGFX（GUI中间件）
 - Parameter Settings (参数设置)

LCD驱动

- 对于部分MCU，在STM32CubeMX中，已经集成了对应ST板上的显示屏驱动。例如演示例中，选择STM32F769NI，LCD驱动中包含了OTM8009A选项（为STM32F769I-DISCO、STM32F769I_EVAL板上显示屏的显示驱动器）。
- 除此之外，提供了Other Screen选项，适用于自定义显示屏。
- 演示例中，为呈现后面自定义显示屏细节，选择Other Screen模式



组件构成

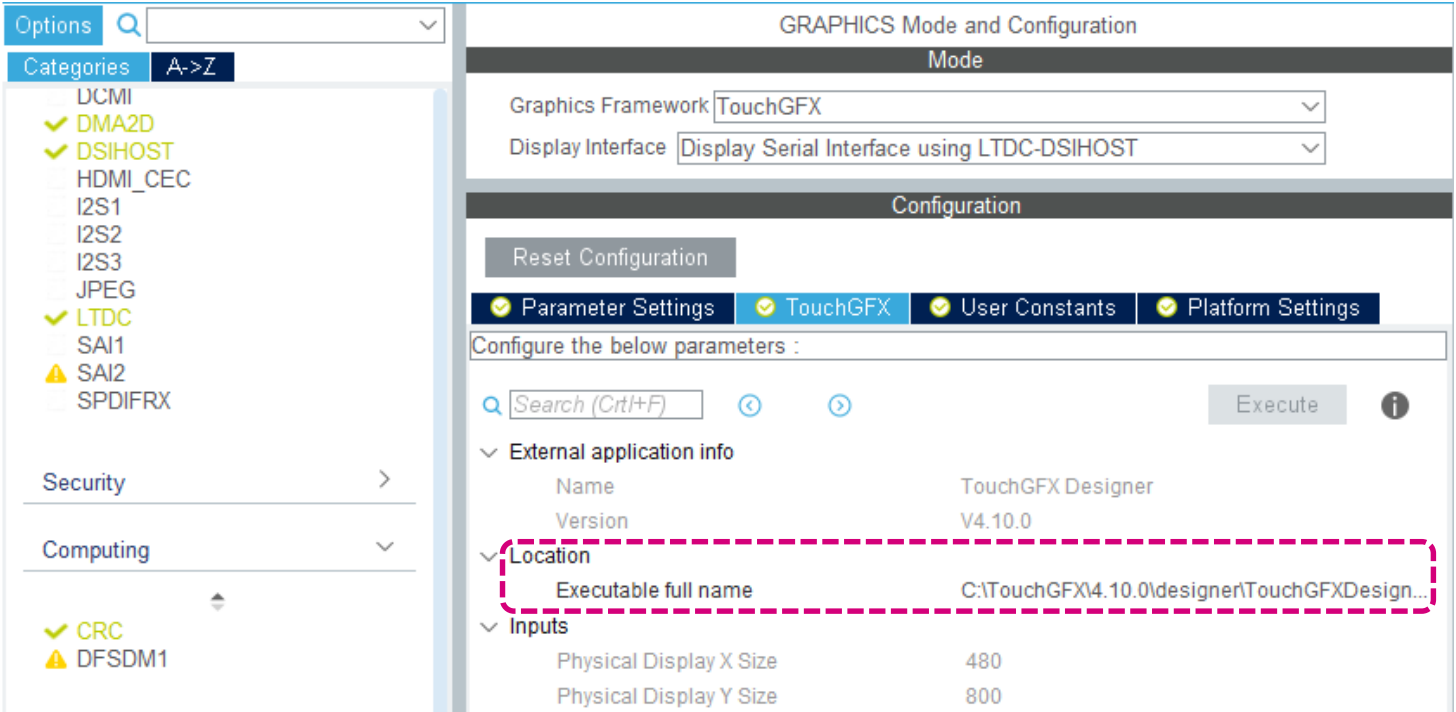




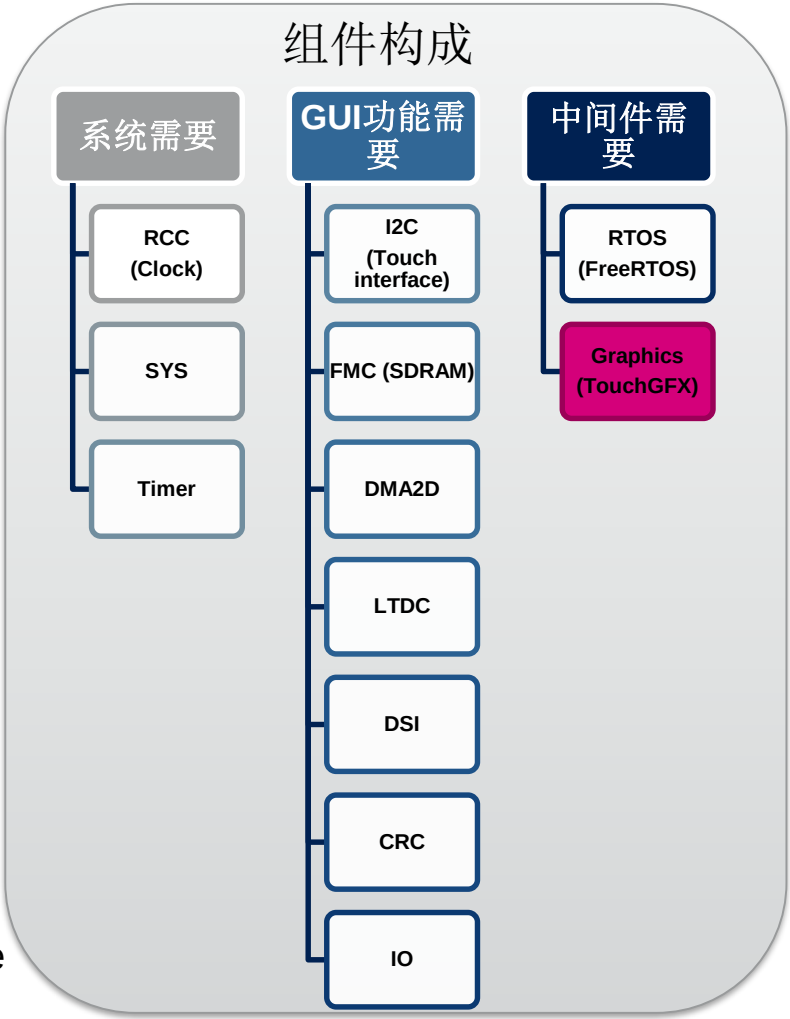
实现示例（续）



- 组件（GUI功能需要）
 - TouchGFX（GUI中间件）
 - 指定TouchGFX Designer执行文件 (PC辅助设计工具)



TouchGFXDesigner默认安装路径: C:\TouchGFX\4.10.0\designer\TouchGFXDesigner-4.10.0.exe





实现示例（续）

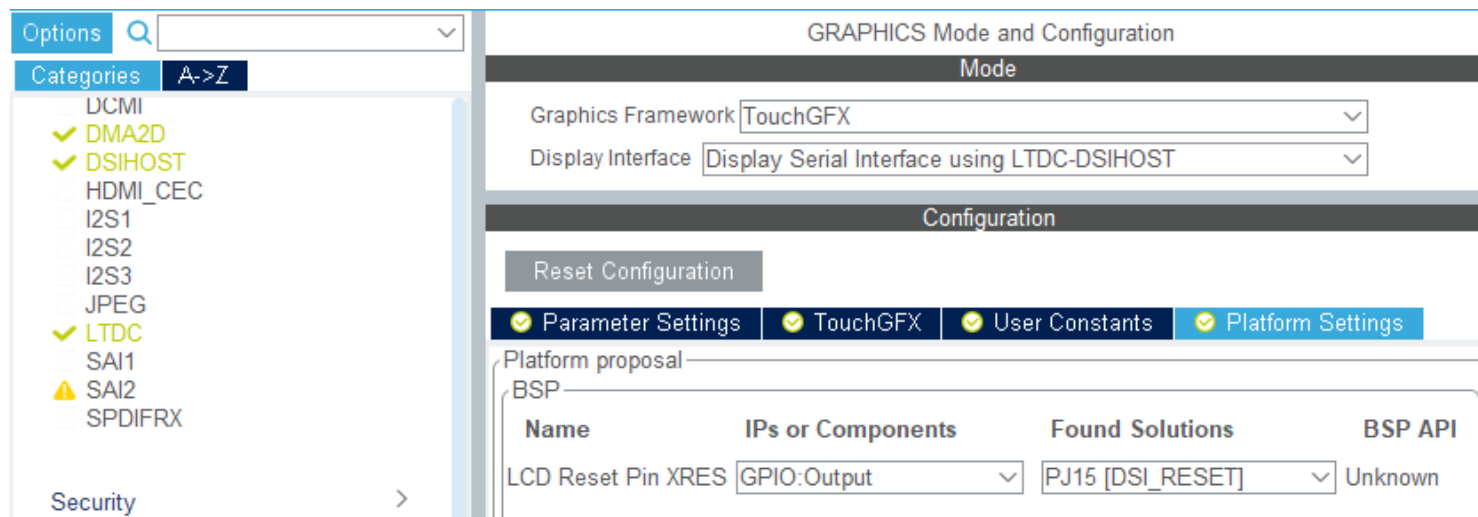
76



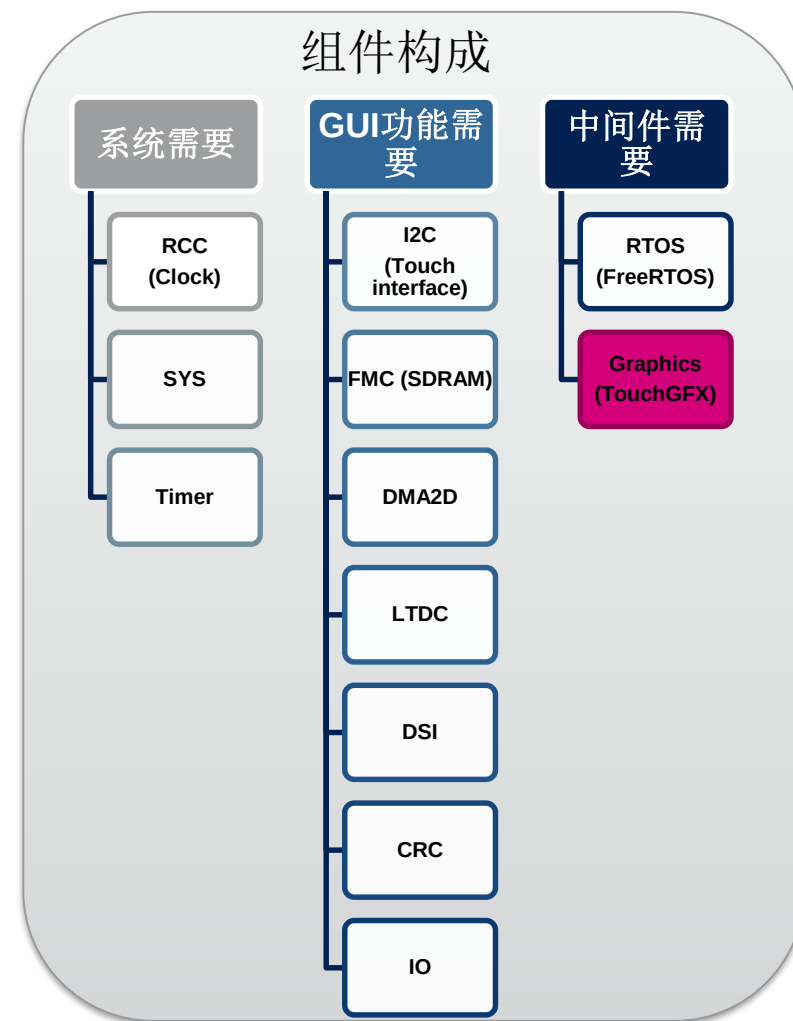
• 组件（GUI功能需要）

• TouchGFX（GUI中间件）

- 指定显示屏复位引脚（需要先至少配置一个引脚为IO输出）



组件构成

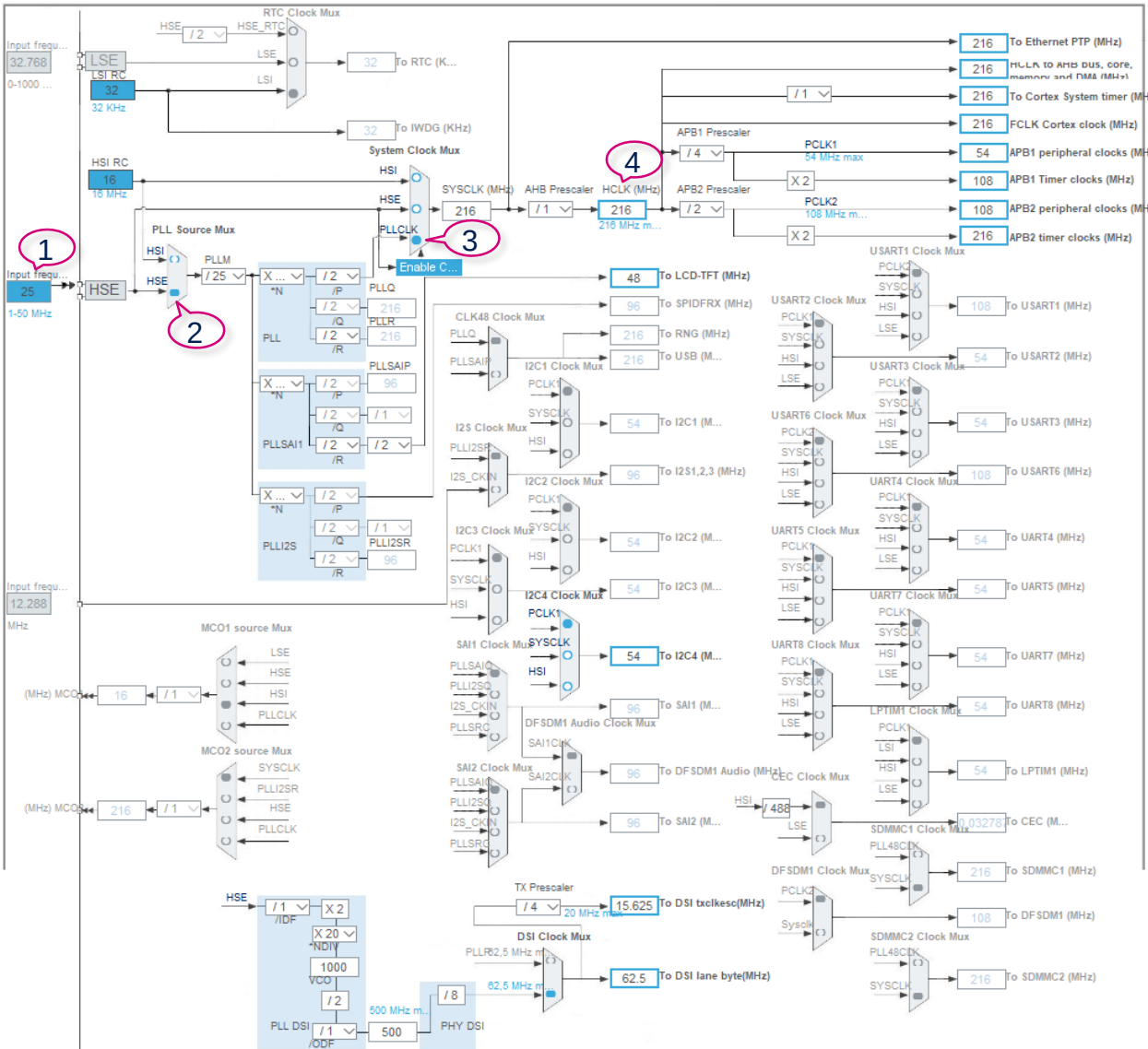
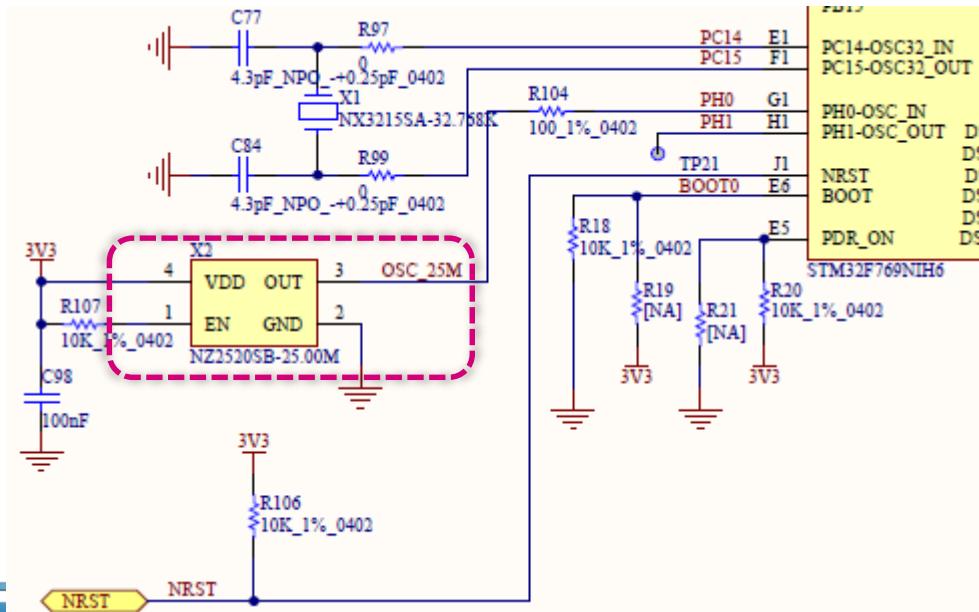




时钟配置

主频

- 如果选用外部时钟，根据外部时钟频率，设置HSE。
- 演示例原理图如下，外部时钟频率为25MHz。
- 根据应用需要，选择时钟源、是否倍频
- 根据应用需要，设置主频和其他时钟频率。
- 演示例中，采用外部时钟和内部PLL倍频，设置最高主频为216MHz。



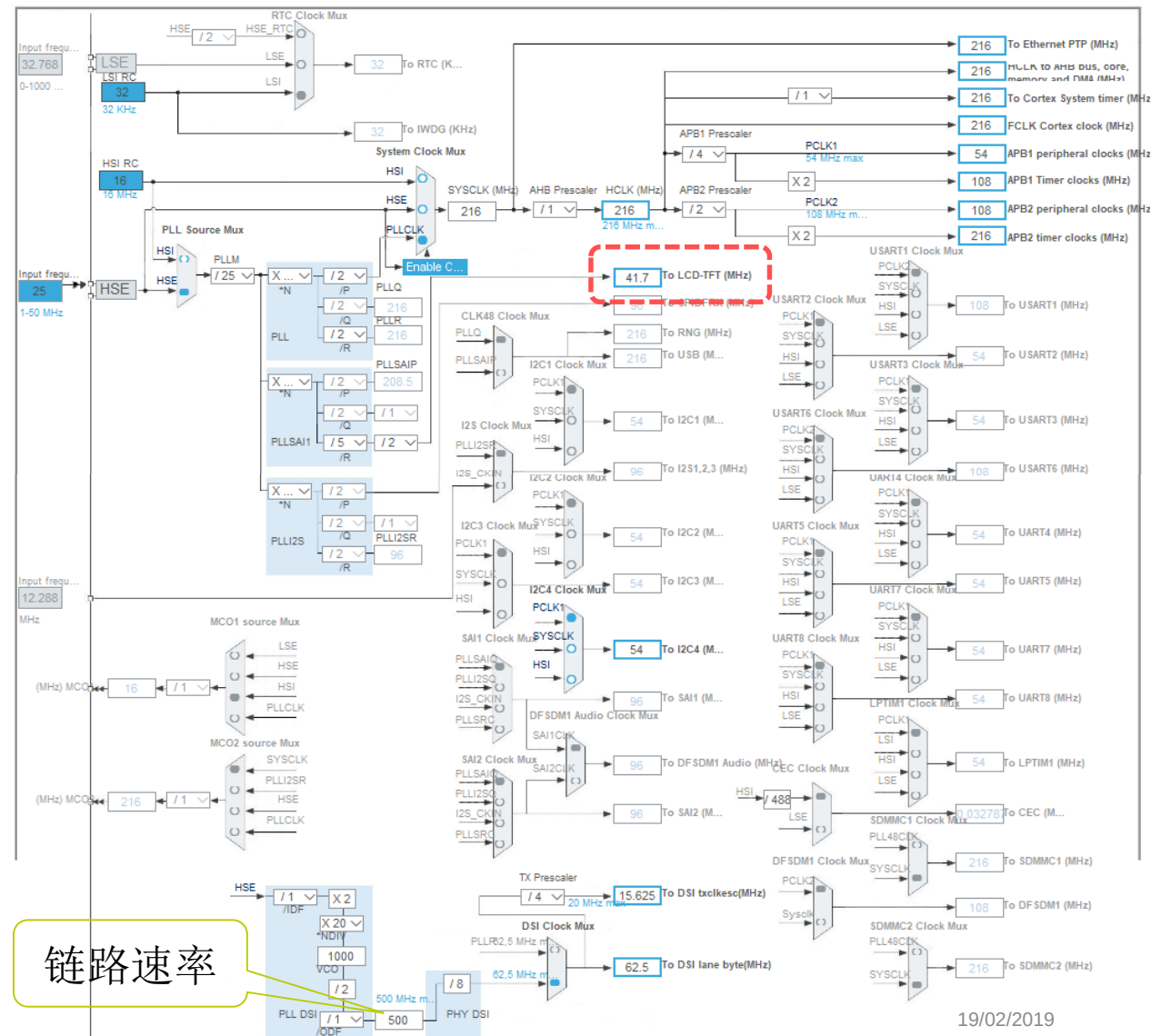


- LCD-TFT (48MHz $\rightarrow$ 41.7MHz)

- 在Adapted Command模式，将LCD_CLK（LCD_TFT）设置为支持的最大频率，来减少刷新时间。

$$LCD_{CLK} = \text{链路速率} * \text{链路数} / \text{每像素位}$$

- 演示例（工作在Adapted Command模式）中，链路速率为500MHz；数据链路数（Number of Lanes）为2；颜色格式为RGB888，对应为每像素24位。
- $LCD\ CLK = 500 * 2 / 24 \approx 41.7MHz$





实现示例（续）

79



• 设置工程

- 设置工程名、保存路径等
- STM32CubeMX支持生成IAR(EWARM)、MDK、TrueSTUDIO、SW4STM32等工程
 - 演示例中，采用IAR。
- 根据应用情况，调整堆栈大小
 - 演示例中，系统堆栈开销小，保持默认。
- 选择固件包版本，需要先添加固件包至STM32CubeMX
- 更多详细介绍请参考菜单栏Help\Help文档。



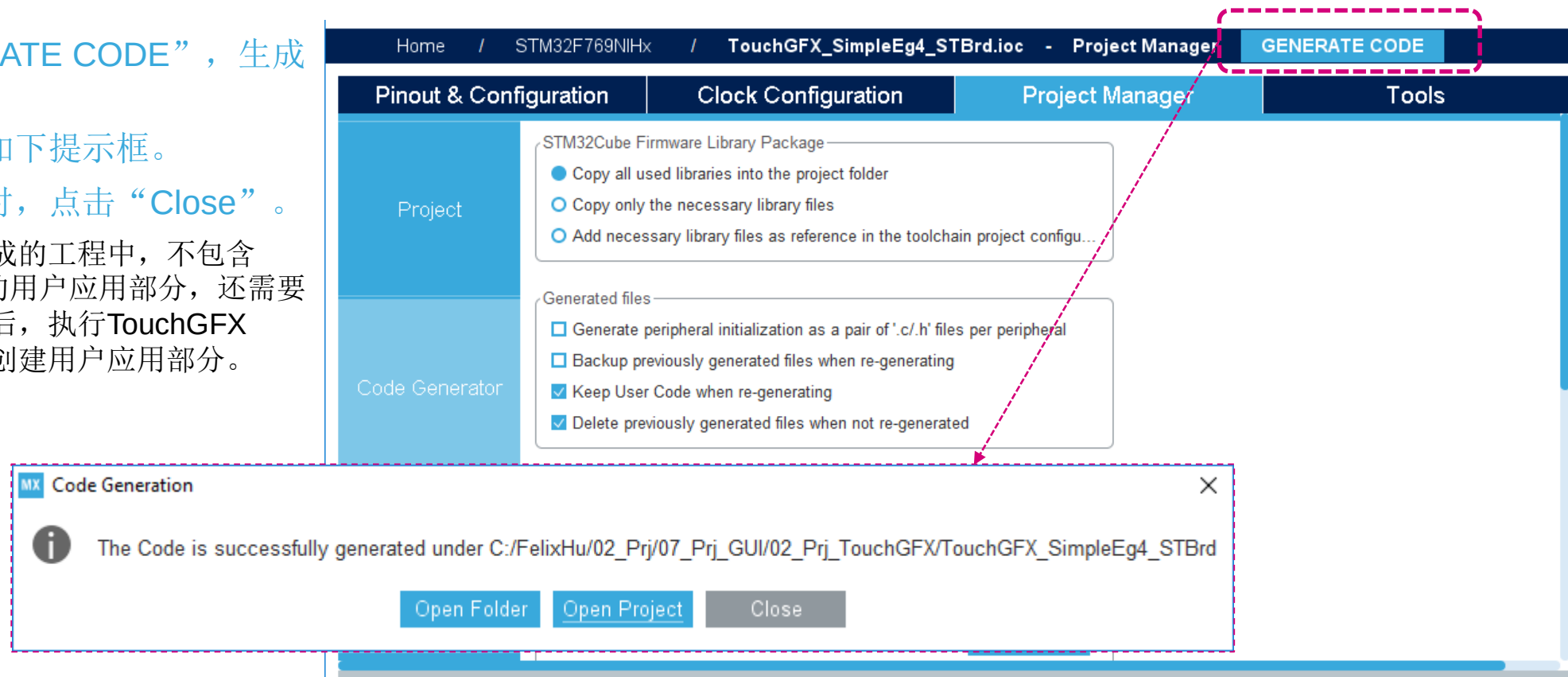
实现示例（续）

80



• 生成工程

- 点击“**GENERATE CODE**”，生成软件工程。
- 生成后，出现如下提示框。
- 首次生成工程时，点击“**Close**”。
 - 由于首次生成的工程中，不包含TouchGFX的用户应用部分，还需要在生成工程后，执行TouchGFX Designer来创建用户应用部分。





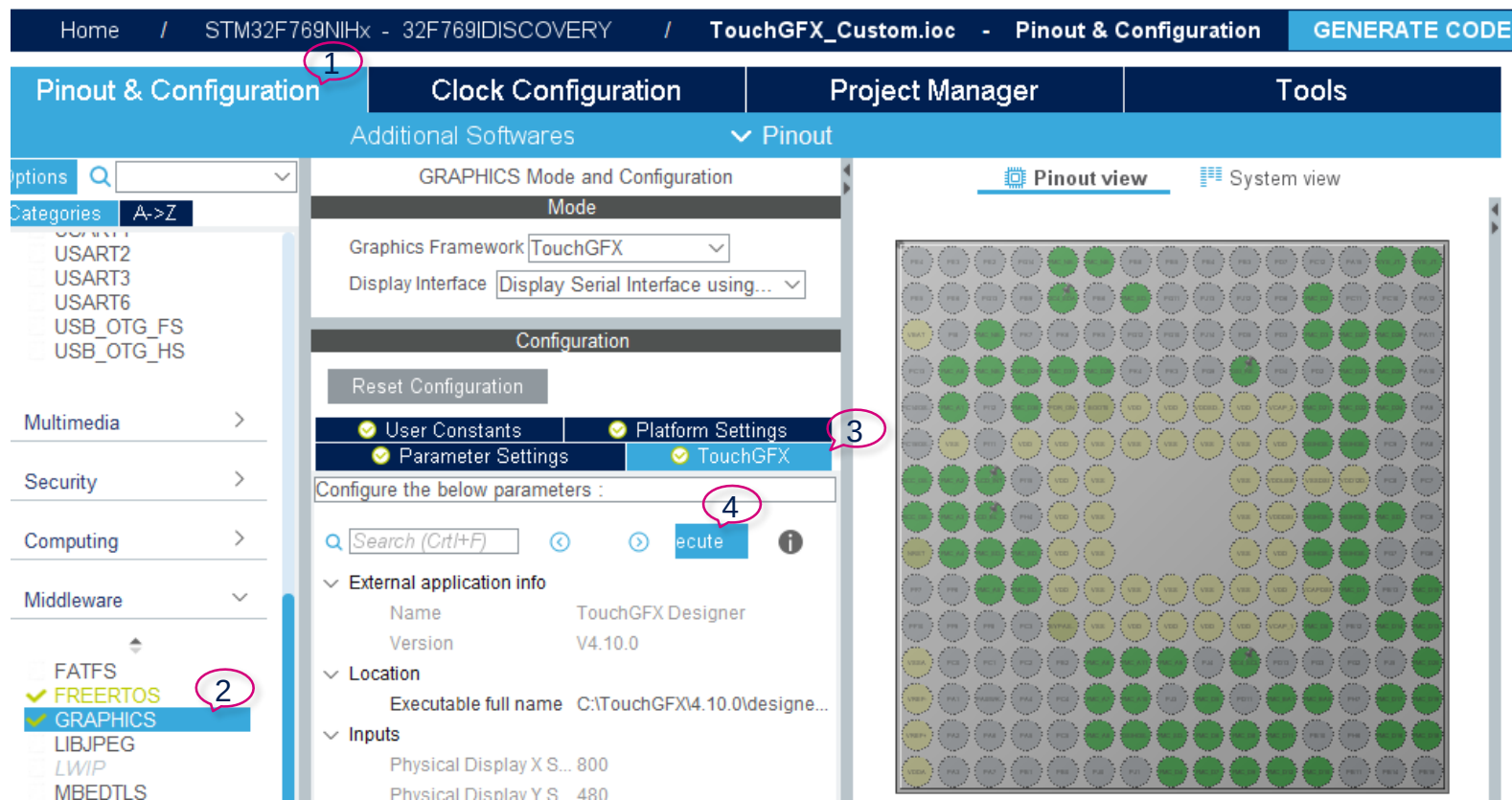
实现示例（续）

81



• 生成工程

- 通过右图步骤，回到TouchGFX中间件的配置界面
- 点击“Execute”，运行TouchGFX Designer





实现示例（续）



- 执行TouchGFX Designer
 - 利用PC辅助设计工具，设计GUI应用

MX Code Generation

The Code is successfully generated under C:/FelixHu/02_Prj/07_Prj_GUI/02_Prj_TouchGFX/TouchGFX_SimpleEg4_STBrd

Open Folder

Open Project

Close

Options

Categories A->Z

✓ CRC

⚠ DFSDM1

Middleware

FATFS

✓ FREERTOS

✓ GRAPHICS

LIBJPEG

LWIP

MBEDTLS

PDM2PCM

USB_DEVICE

USB_HOST

Application

GFXSIMULATOR

GRAPHICS Mode and Configuration

Mode

Graphics Framework TouchGFX

Display Interface Display Serial Interface using LTDC-DSIHOST

Configuration

Reset Configuration

✓ User Constants

✓ Platform Settings

✓ Parameter Settings

✓ TouchGFX

Configure the below parameters :

Search (Ctrl+F)

Execute

External application info

Name TouchGFX Designer

Version V4.10.0

Location

Executable full name C:\TouchGFX\4.10.0\designer\Touch...

Inputs

Physical Display X Size 800

Physical Display Y Size 480

TouchGFX Designer

File Edit Help

TouchGFX DESIGNER

Canvas

Run Simulator

Run Target

Generate Code

MyApplication

Screen1

Buttons

Button

Button With Label

Button With Icon

Toggle Button

Radio Button

Repeat Button

SCREEN

NAME

Screen1

STARTUP SCREEN

This screen is startup screen

CANVAS BUFFER

Override default buffer size

Size in bytes 0

Bootstrapping application

Browse Code

Detailed Log



实现示例（续）



• 执行TouchGFX Designer

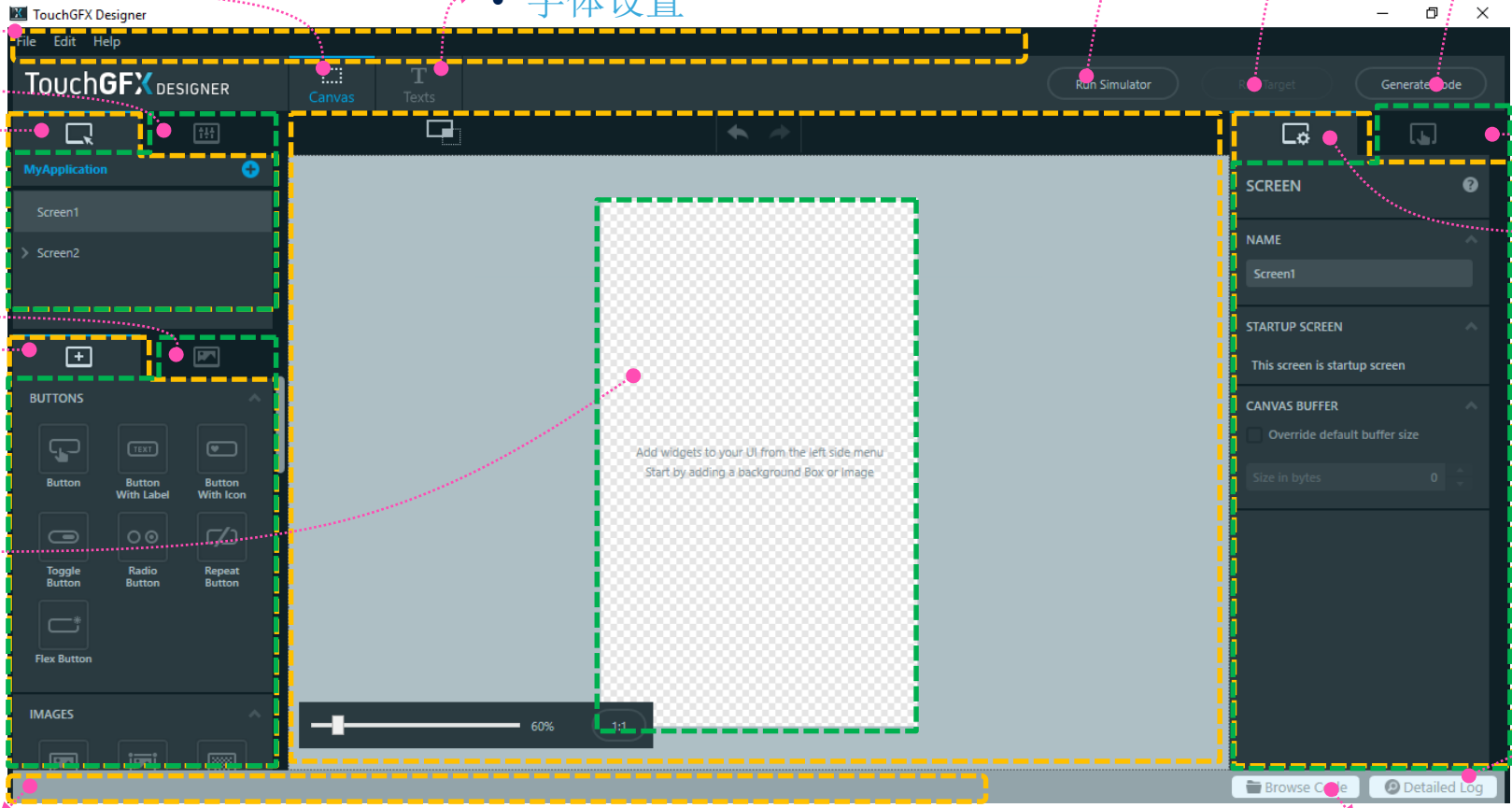
• TouchGFX Designer界面

• 在添加GUI应用前，先了解下TouchGFX Designer界面

- 画布
- 菜单栏
- 自定义控件
- 界面

- 图片素材标签页
- 控件标签页

- 设计窗口



• 状态栏

• 字体设置

- 仿真
- 运行目标
- 生成代码

- 交互
- 属性设置

• Log信息

• 打开代码所在目录 19/02/2019



实现示例（续）

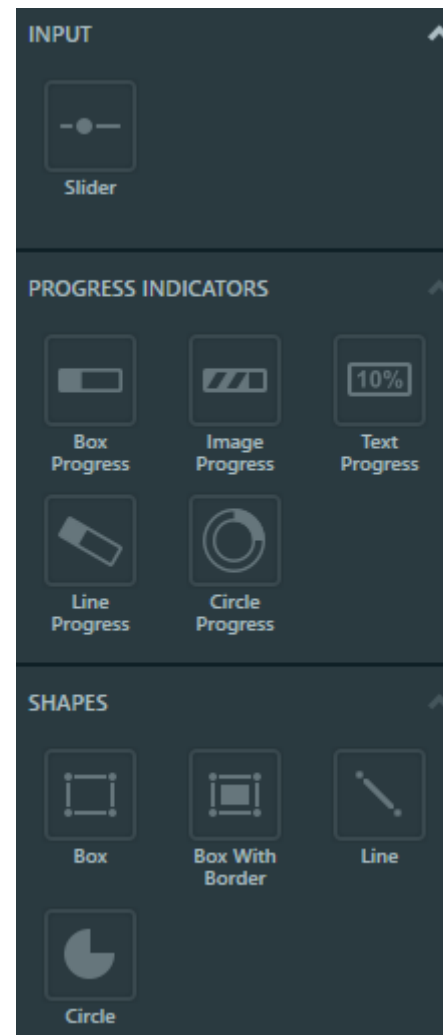
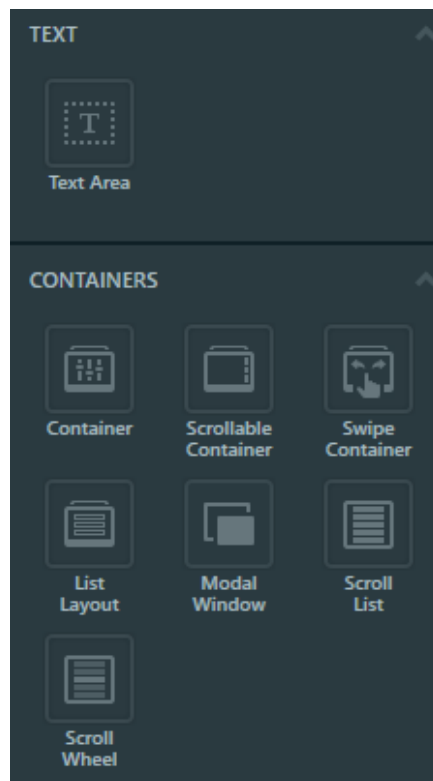
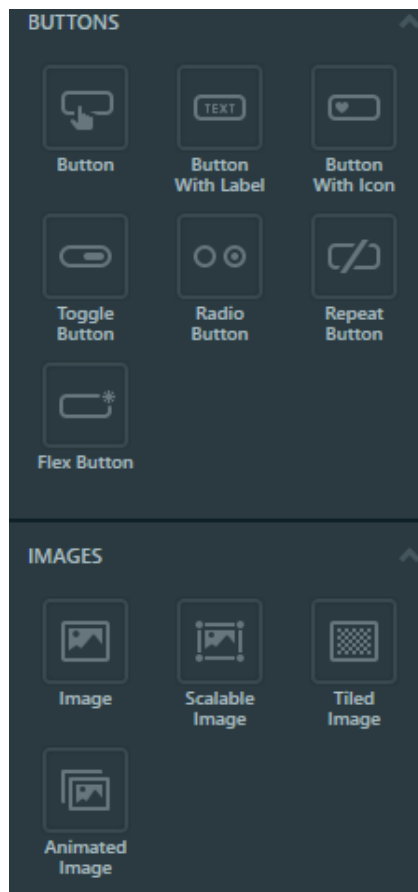
84



• 执行TouchGFX Designer

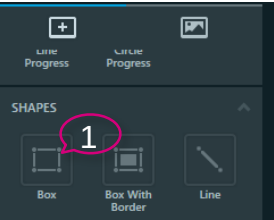
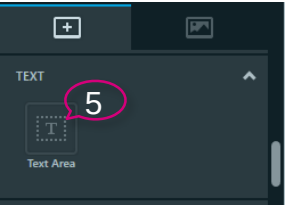
• TouchGFX Designer界面

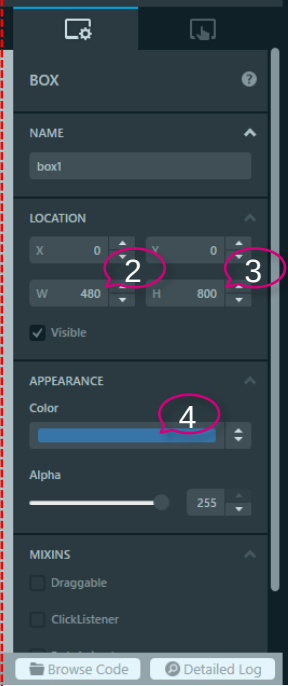
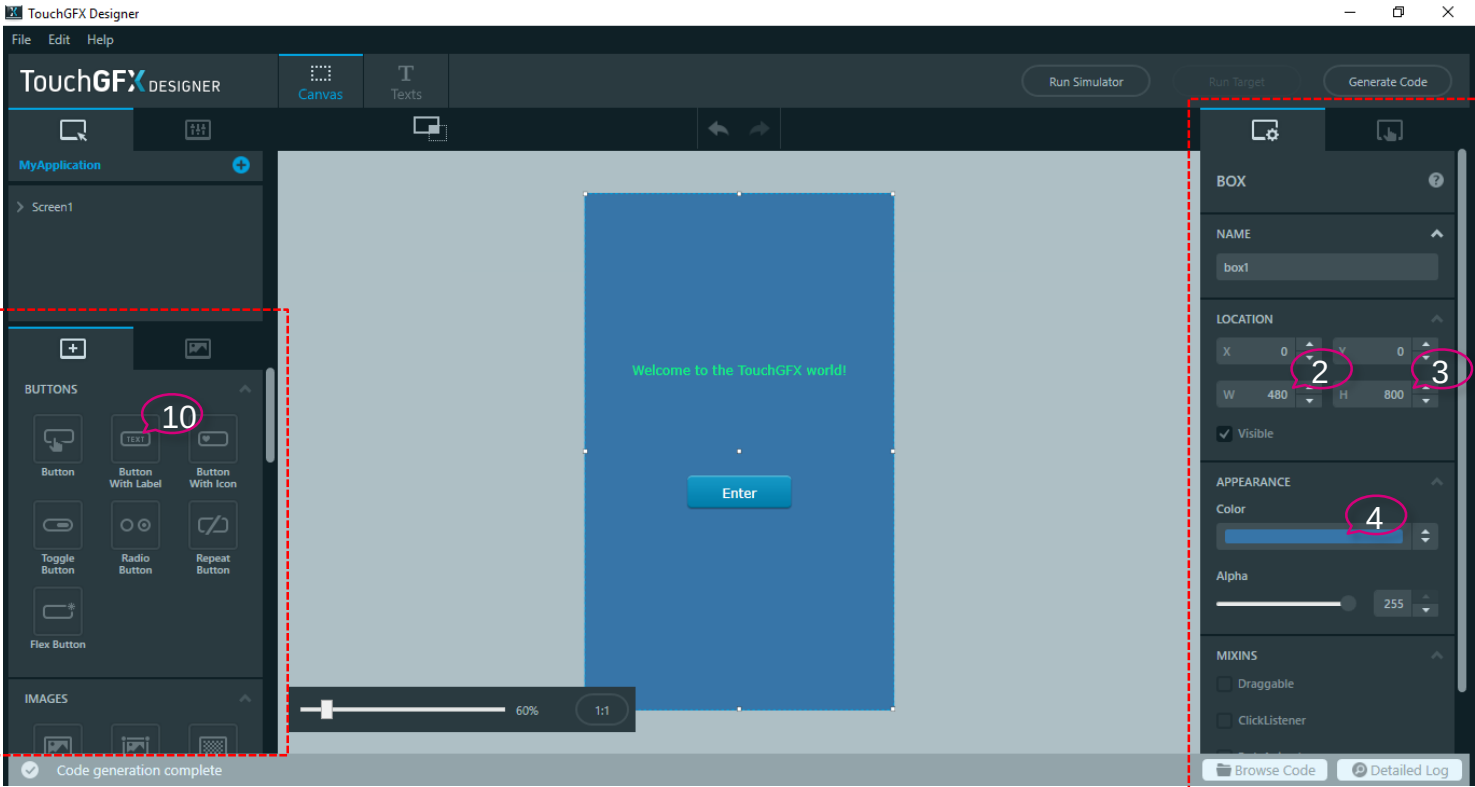
- 其中，控件标签页中，包含如下控件。方便GUI开发者快速实现GUI应用设计。

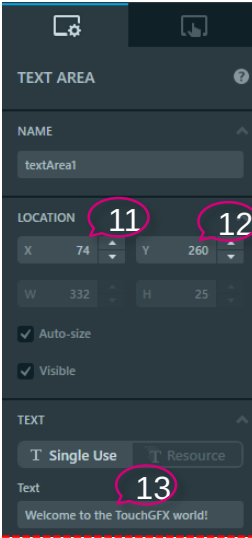
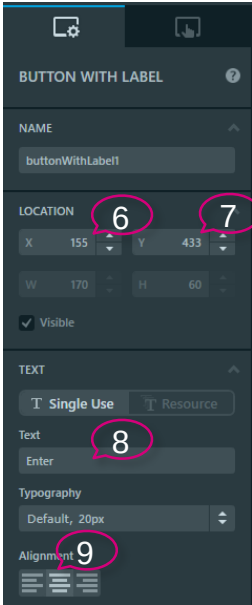




- 执行TouchGFX Designer
 - 添加GUI应用。
 - 简单示例，显示白色背景、一个按键和一串字符，如下图所示









19/02/2019



实现示例（续）

86



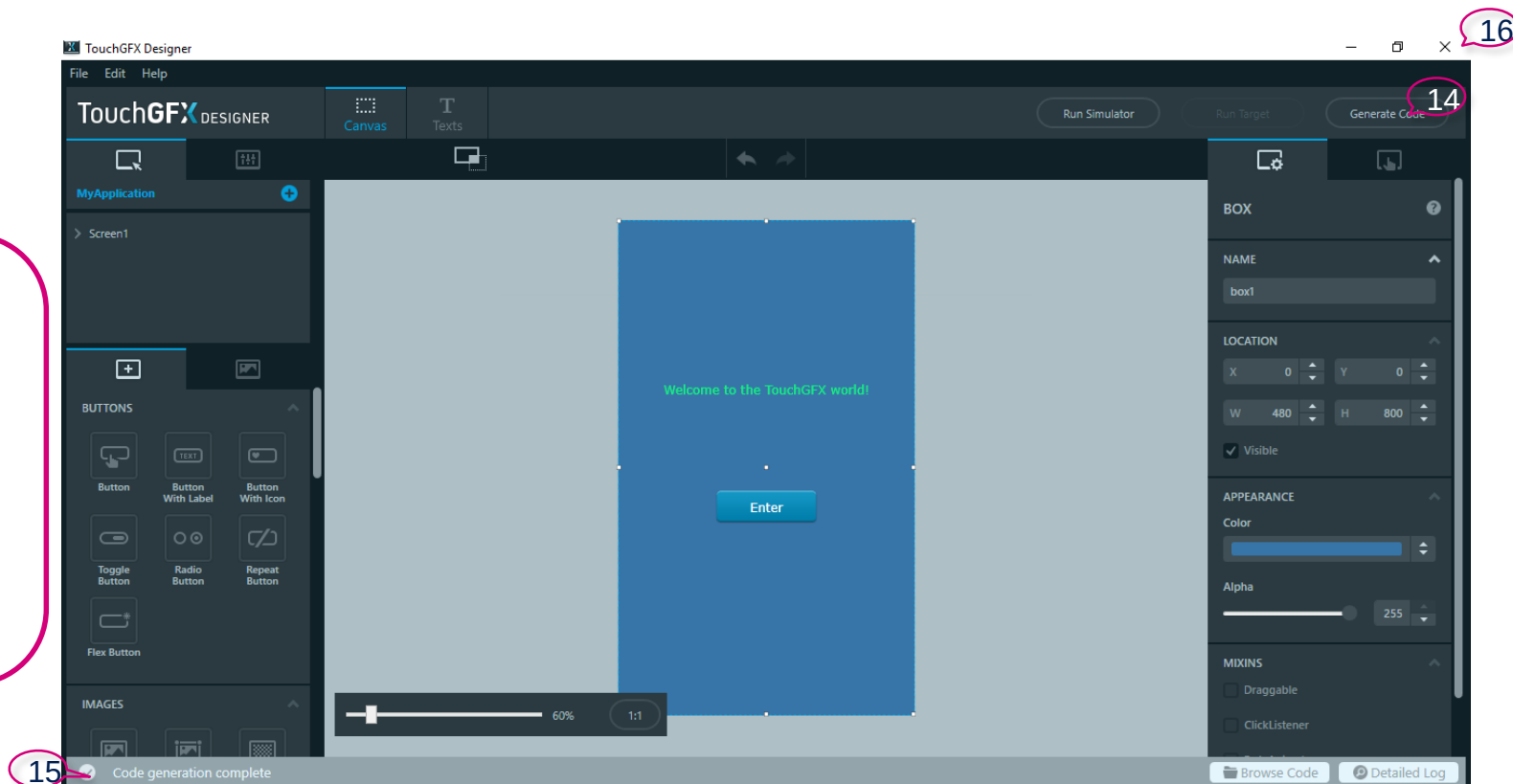
• 执行TouchGFX Designer

- 然后，点击右上角“Generate Code”，生成TouchGFX用户的应用代码。左下角会显示处理状态。
- 等待出现生成完成的提示后，关闭TouchGFX Designer，返回到STM32CubeMX。

注：详细TouchGFX Designer使用及GUI应用开发介绍，请参考：

[TouchGFX官网](#)

TouchGFX Designer
自带示例（通过
File\New选择）





实现示例（续）

87



• 执行TouchGFX Designer

- 至此，STM32CubeMX中配置工作完成。
- 再次点击STM32CubeMX中的“GENERATE CODE”，生成IDE工程。并且点击弹出菜单栏的“Open Project”。
- 工程中文件结构如图所示。

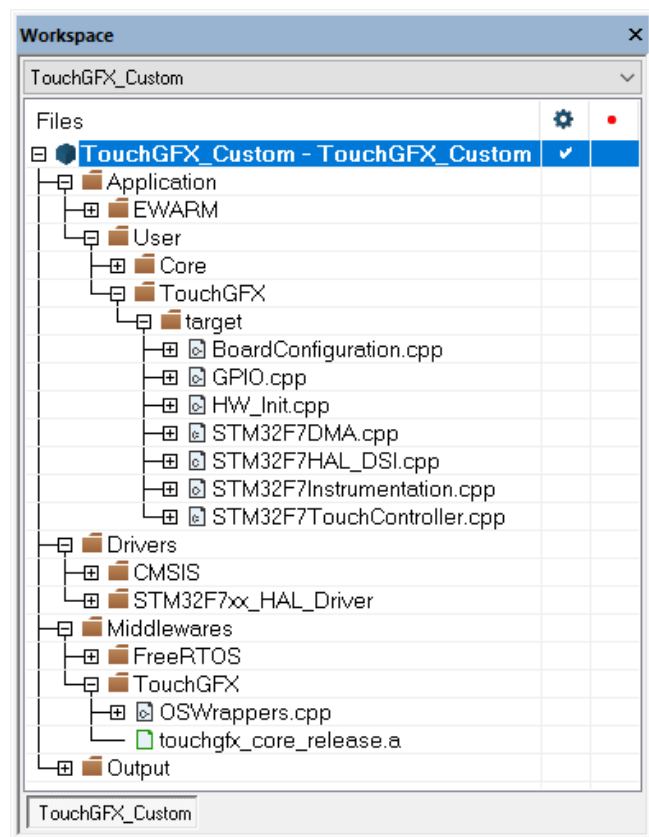


图1. IDE工程文件结构
(运行TouchGFX Designer前)

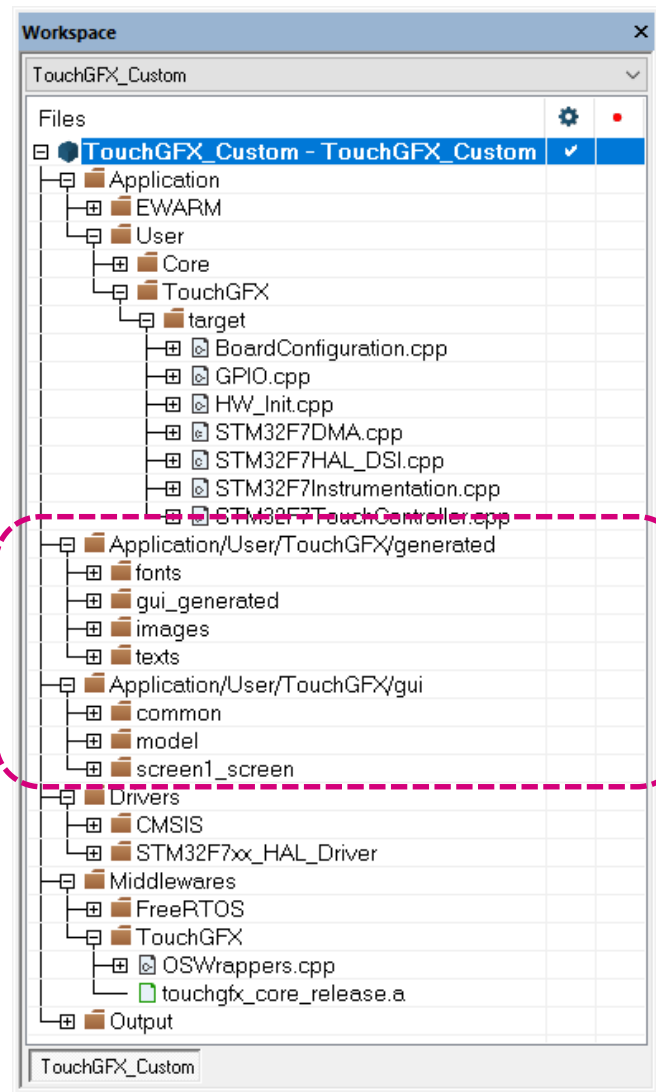


图2. IDE工程文件结构
(运行TouchGFX Designer后)



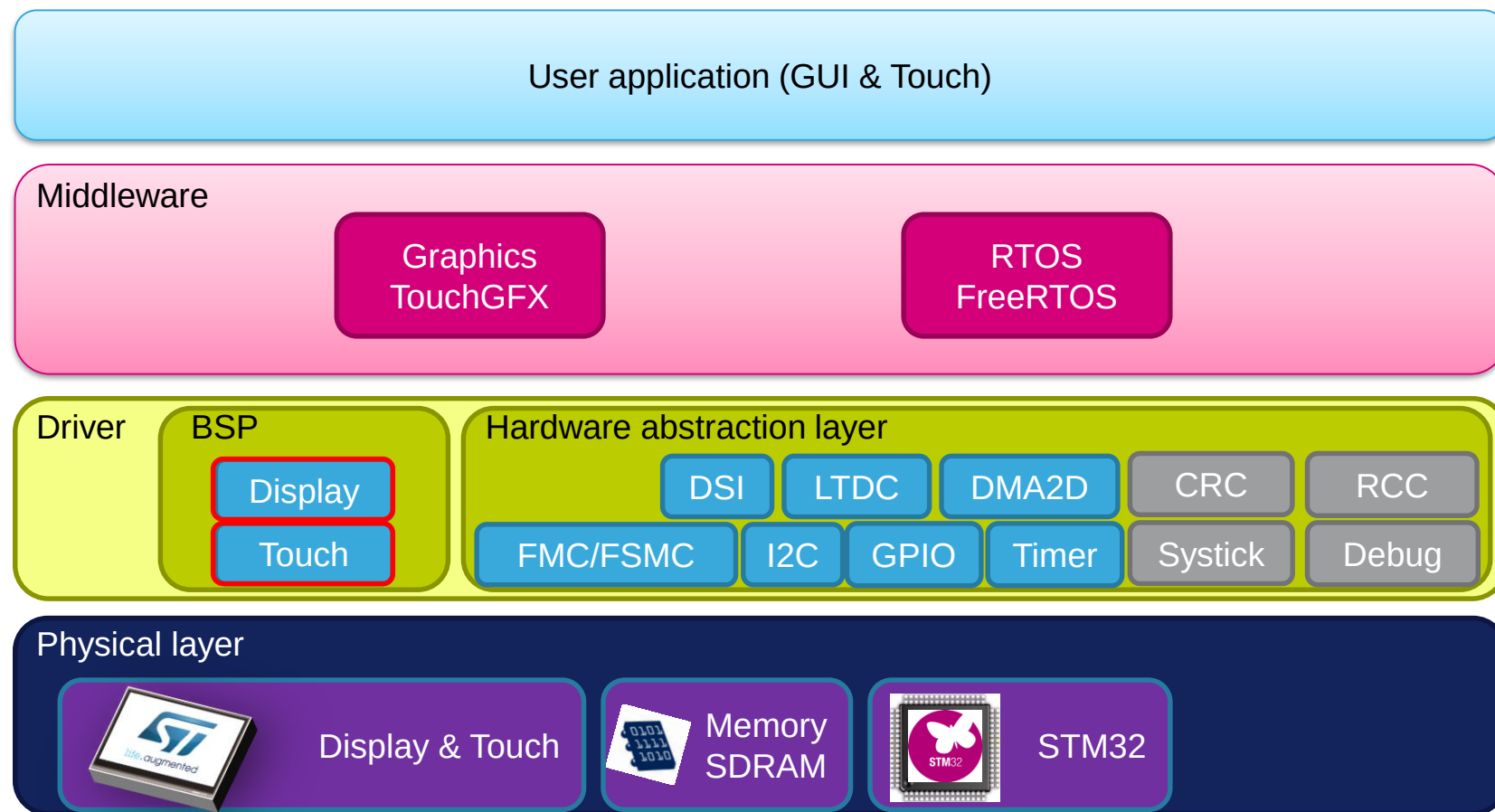
实现示例（续）

88



• 完善工程

- 通过上述步骤，生成的IDE工程，覆盖了右图除红色框外的组件。
- 完善工程内容
 - 添加显示驱动实现
 - 添加触摸驱动实现





实现示例（续）

89



• 完善工程

- 在完善工程前，首先介绍如下内容。
 - IDE工程的文件夹目录结构。
 - IDE工程中文件结构。
 - 显示驱动调用信息汇总。通过这部分介绍，能够了解在工程中何处添加哪些显示功能，从而实现显示驱动的添加。
 - 触摸驱动调用信息汇总。通过这部分介绍，能够了解在工程中何处添加哪些触摸功能，从而实现触摸驱动的添加。
- 通过上述介绍，对STM32CubeMX开发的TouchGFX应用工程整体情况，有一定的了解。在此基础上，再添加显示和触摸的驱动，完善工程。



实现示例（续）

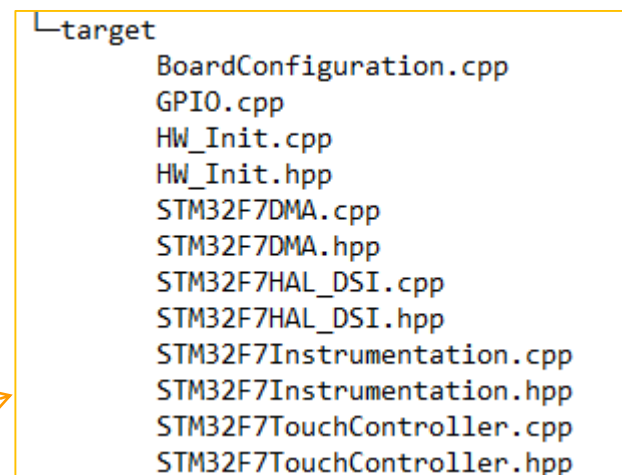
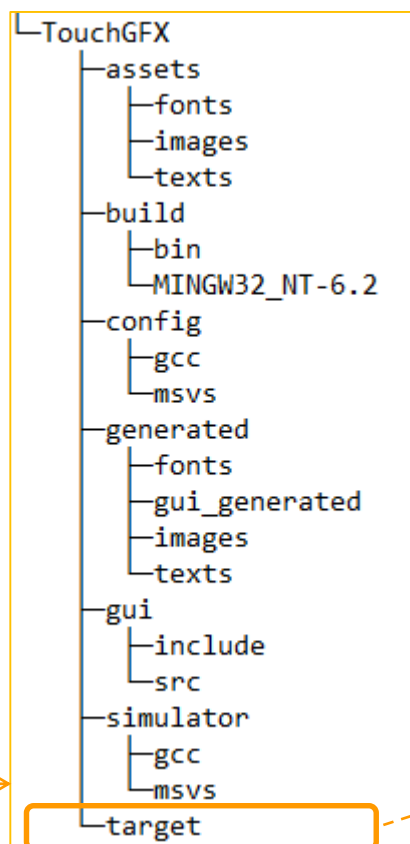
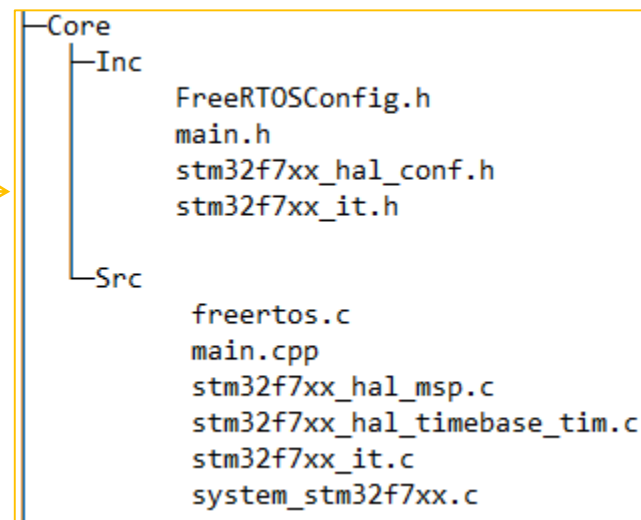
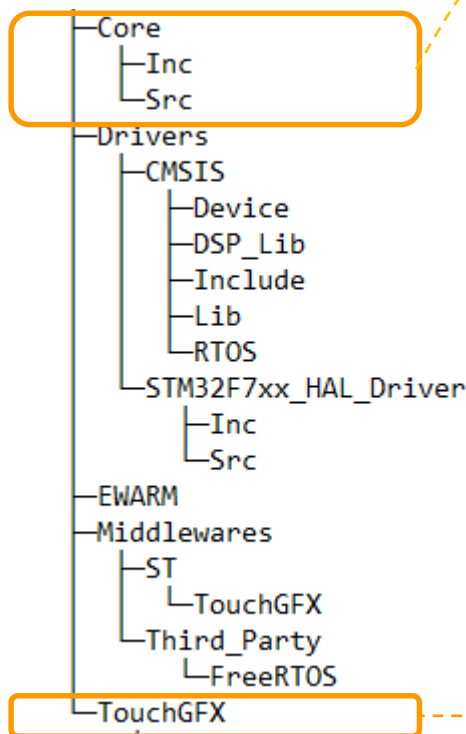
90



完善工程

在完善工程前，首先介绍如下内容。

- IDE工程的文件夹目录结构
- IDE工程中文件结构
- 显示驱动调用信息汇总
- 触摸驱动调用信息汇总





实现示例（续）

91



完善工程

在完善工程前，首先介绍如下内容。

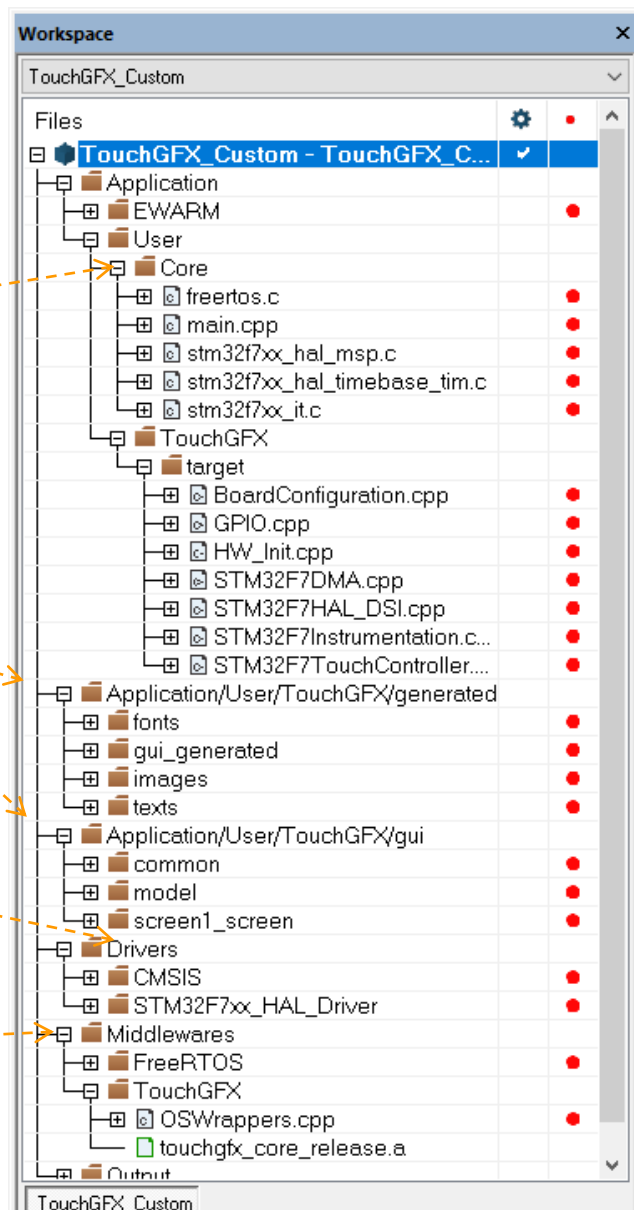
- IDE工程的文件夹目录结构
- IDE工程中文件结构
- 显示驱动调用信息汇总
- 触摸驱动调用信息汇总

用户部分（系统初始化、外扩存储器、显示&触摸等初始化，以及底层接口等实现）

TouchGFX框架、应用部分

HAL、BSP驱动

中间件（FreeRTOS、TouchGFX）





完善工程

- 在完善工程前，首先介绍如下内容。

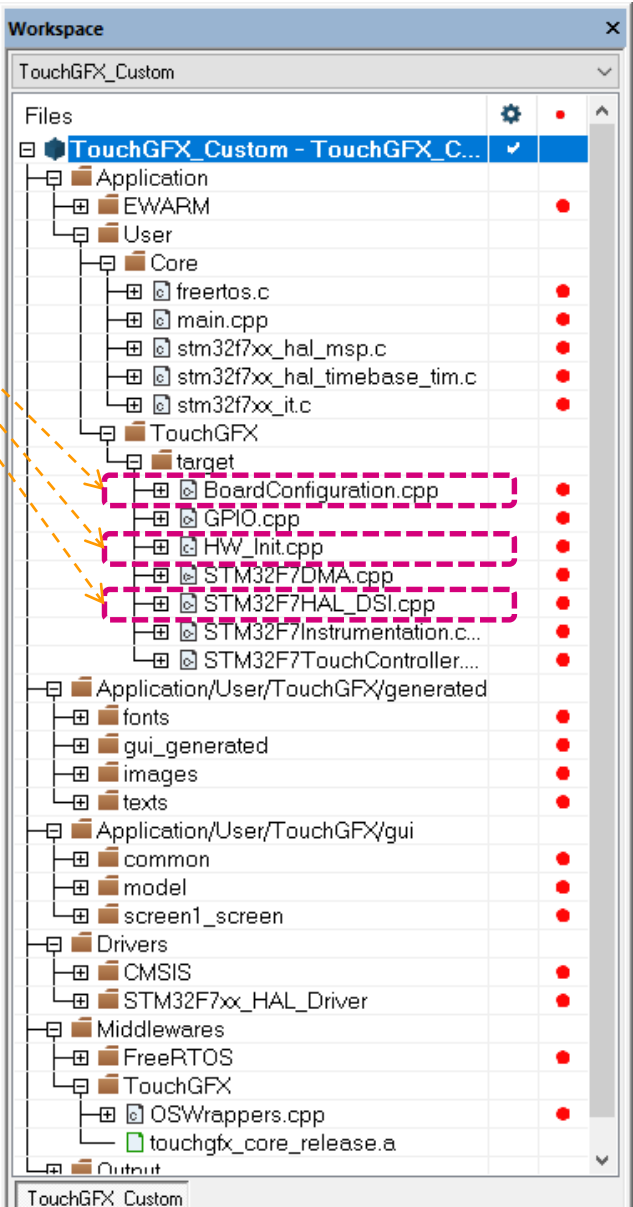
 - IDE工程的文件夹目录结构
 - IDE工程中文件结构
 - 显示驱动调用信息汇总
 - 触摸驱动调用信息汇总

显示驱动被
调用位置

显示驱动添加

- STM32CubeMX生成的TouchGFX应用工程中，预留了显示驱动功能的调用位置或者函数。在对应位置或者函数中，完成下表中功能/函数，即实现了显示驱动的添加。

驱动功能	调用函数/位置	源文件
显示屏初始化	MX_DSI_Init()	HW_Init.cpp
同步信息设置	LCD_ReqTear()	BoardConfiguration.cpp
设置更新区域的列范围	LCD_SetUpdateRegion()	BoardConfiguration.cpp
显示缓存刷新	HAL_DSI_EndOfRefreshCallback()	STM32F7HAL_DSI.cpp





实现示例（续）

93

完善工程

显示驱动添加实例

驱动功能	调用函数/位置	源文件
显示屏初始化	MX_DSI_Init()	HW_Init.cpp
同步信息设置	LCD_ReqTear()	BoardConfiguration.cpp
设置更新区域的列范围	LCD_SetUpdateRegion()	BoardConfiguration.cpp
显示缓存刷新	HAL_DSI_TearingEffectCallback()	STM32F7HAL_DSI.cpp
缓存刷新完成处理	HAL_DSI_EndOfRefreshCallback	STM32F7HAL_DSI.cpp

演示例中，直接采用ST提供的OTM8009A显示驱动文件otm8009A.c/.h。这里不对驱动本身进行介绍，而是侧重于介绍生成工程中需要添加的驱动功能和添加位置。

- otm8009a.c/.h获取路径：STM32CubeF7固件包\Drivers\BSP\Components\otm8009a
- 首先，将驱动文件otm8009a.c/.h文件复制到工程文件夹，如右图1所示。
- 在IAR工程中，添加驱动文件otm8009a.c到工程，如右图2所示。
- 在IAR工程的菜单栏\Option\Project\CC++ Compiler\Preprocessor\Additional include directories中添加驱动头文件所在路径。
- 在main.h中包含显示的驱动头文件 otm8009a.h
 - #include "otm8009a.h"

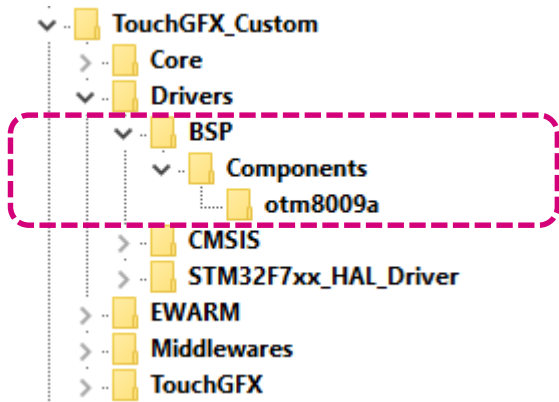


图1

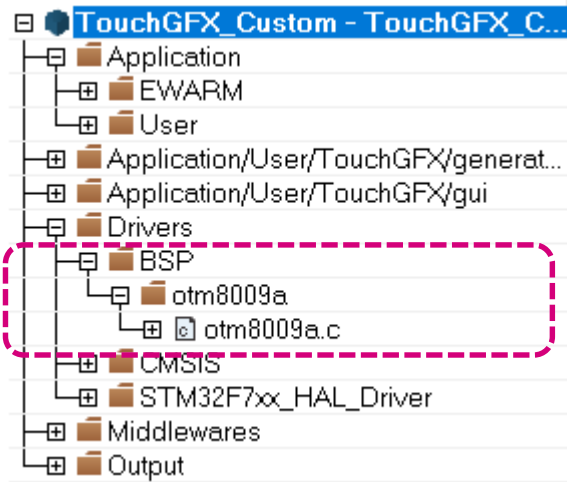


图2



- 完善工程
 - 显示驱动添加实例

驱动功能	调用函数/位置	源文件
显示屏初始化	MX_DSI_Init()	HW_Init.cpp
同步信息设置	LCD_ReqTear()	BoardConfiguration.cpp
设置更新区域的列范围	LCD_SetUpdateRegion()	BoardConfiguration.cpp
显示缓存刷新	HAL_DSI_TearingEffectCallback()	STM32F7HAL_DSI.cpp
缓存刷新完成处理	HAL_DSI_EndOfRefreshCallback	STM32F7HAL_DSI.cpp

- 演示例中，直接采用ST提供的OTM8009A显示驱动文件otm8009A.c/.h。这里不对驱动本身进行介绍，而是侧重于介绍生成工程中需要添加的驱动功能和添加位置。
 - otm8009a.c/.h获取路径：STM32CubeF7固件包\Drivers\BSP\Components\otm8009a

在MX_DSI_Init()中已经注释了显示驱动器初始化位置如下。

```
/* Initialize the LCD Display IC Driver (KoD LCD IC Driver)
 * depending on configuration set in 'hdsivideo_handle'.
 */
```

在注释后面添加初始化程序如下。其中OTM8009A_Init()定义在otm8009a.c中，涉及到的宏定义在otm8009a.h中。

```
/* Send Display off DCS Command to display */
HAL_DSI_ShortWrite(&(hdsi),
0,
DSI_DCS_SHORT_PKT_WRITE_P1,
OTM8009A_CMD_DISPOFF,
0x00);

/* Display driver initialization */
OTM8009A_Init(OTM8009A_FORMAT_RGB888,
OTM8009A_ORIENTATION_PORTRAIT);
```

```
void LCD_ReqTear(void)
{
    uint8_t ScanLineParams[2];
    uint16_t scanline = 800 - 1;           //TE信号触发行（相对于显示刷新方向）
    ScanLineParams[0] = scanline >> 8;
    ScanLineParams[1] = scanline & 0x00FF;
    HAL_DSI_LongWrite(&hdsi, 0, DSI_DCS_LONG_PKT_WRITE, 2,
OTM8009A_CMD_WRTE SCN, ScanLineParams);
    HAL_DSI_ShortWrite(&hdsi, 0, DSI_DCS_SHORT_PKT_WRITE_P1,
OTM8009A_CMD_TEEON, OTM8009A_TEEON_TELOM_VBLANKING_INFO_ONLY);
}
```



实现示例（续）

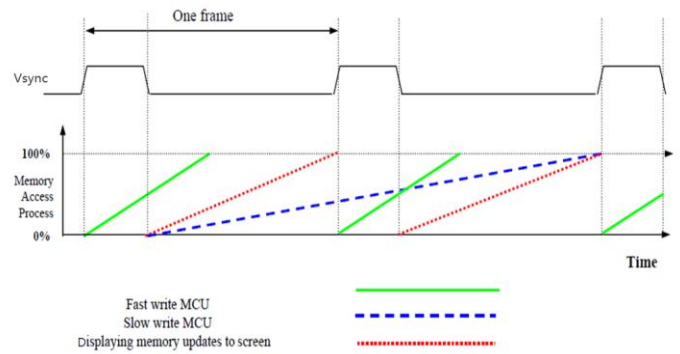


- 完善工程
 - 显示驱动添加实例

驱动功能	调用函数/位置	源文件
显示屏初始化	MX_DSI_Init()	HW_Init.cpp
同步信息设置	LCD_ReqTear()	BoardConfiguration.cpp
设置更新区域的列范围	LCD_SetUpdateRegion()	BoardConfiguration.cpp
显示缓存刷新	HAL_DSI_TearingEffectCallback()	STM32F7HAL_DSI.cpp
缓存刷新完成处理	HAL_DSI_EndOfRefreshCallback	STM32F7HAL_DSI.cpp

```
void LCD_ReqTear(void)
{
    uint8_t ScanLineParams[2];
    uint16_t scanline = 800 - 1;           //TE信号触发行（相对于显示刷新方向）
    ScanLineParams[0] = scanline >> 8;
    ScanLineParams[1] = scanline & 0x00FF;
    HAL_DSI_LongWrite(&hdsi, 0, DSI_DCS_LONG_PKT_WRITE, 2,
OTM8009A_CMD_WRTESCN, ScanLineParams);
    HAL_DSI_ShortWrite(&hdsi, 0, DSI_DCS_SHORT_PKT_WRITE_P1,
OTM8009A_CMD_TEEON, OTM8009A_TEEON_TELOM_VBLANKING_INFO_ONLY);
}
```

- Scanline的设置，决定了TE信号产生行。一般在TE信号处进行显示缓存刷新，能够有效避免撕裂现象。
- 撕裂现象是由于帧缓存刷新和显示屏刷新的不同步引起，有多种解决机制。TE信号的触发位置需要结合解决机制进行确认，简单罗列如下。
 - 1. 调整刷新方向，使保持一致，并且保证显示帧缓存的刷新周期
 - 2. 单FrameBuffer，分多显示区域，分时处理
 - 3. 多FrameBuffer，分时处理
- 演示例中采用第一种解决方法，并且考虑到帧缓存刷新速度比显示刷新快，对应于下图绿色和红色线。所以在显示刷新完毕（对应为TE触发位置设置在末尾），从而帧缓存刷新留有最多的时间。





- 完善工程
 - 显示驱动添加实例

驱动功能	调用函数/位置	源文件
显示屏初始化	MX_DSI_Init()	HW_Init.cpp
同步信息设置	LCD_ReqTear()	BoardConfiguration.cpp
设置更新区域的列范围	LCD_SetUpdateRegion()	BoardConfiguration.cpp
显示缓存刷新	HAL_DSI_TearingEffectCallback()	STM32F7HAL_DSI.cpp
缓存刷新完成处理	HAL_DSI_EndOfRefreshCallback	STM32F7HAL_DSI.cpp

- 演示例中，直接采用ST提供的OTM8009A显示驱动文件otm8009A.c/.h。这里不对驱动本身进行介绍，而是侧重于介绍生成工程中需要添加的驱动功能和添加位置。
 - otm8009a.c/.h获取路径：STM32CubeF7固件包\Drivers\BSP\Components\otm8009a

在LCD_SetUpdateRegionRight()函数后面，添加如下函数，设置更新全部区域的列范围

```
void LCD_SetUpdateRegionFull()
{
    HAL_DSI_LongWrite(&hdsi, 0,
        DSI_DCS_LONG_PKT_WRITE, 4,
        OTM8009A_CMD_CASET, pColFull);
}
```

其中，pColFull定义如下。（根据显示区域列范围）
uint8_t pColFull[] = {0x00, 0x00, 0x03, 0x1F}; /* 0 -> 799 */

同时，在STM32F7HAL_DSI.cpp文件中，LCD_SetUpdateRegionRight()函数声明后面，添加LCD_SetUpdateRegionFull()的声明，如下所示。
void LCD_SetUpdateRegionFull();



完善工程

显示驱动添加实例

驱动功能	调用函数/位置	源文件
显示屏初始化	MX_DSI_Init()	HW_Init.cpp
同步信息设置	LCD_ReqTear()	BoardConfiguration.cpp
设置更新区域的列范围	LCD_SetUpdateRegion()	BoardConfiguration.cpp
显示缓存刷新	HAL_DSI_TearingEffectCallback()	STM32F7HAL_DSI.cpp
缓存刷新完成处理	HAL_DSI_EndOfRefreshCallback	STM32F7HAL_DSI.cpp

演示例中，直接采用ST提供的OTM8009A显示驱动文件otm8009a.c/.h。这里不对驱动本身进行介绍，而是侧重于介绍生成工程中需要添加的驱动功能和添加位置。

- otm8009a.c/.h获取路径： STM32CubeF7固件包 \Drivers\BSP\Components\otm8009a

注： otm8009a.c/.h中OTM8009A_IO_Delay()函数，需要添加定义。演示例中，在BoardConfiguration.cpp中添加如下

```
void OTM8009A_IO_Delay(uint32_t Delay)
{
    HAL_Delay(Delay);
}
```

```
void HAL_DSI_TearingEffectCallback(DSI_HandleTypeDef *hdsi)
{
    GPIO::set(GPIO::VSYNC_FREQ);
    HAL::getInstance()->vSync();
    OSWrappers::signalVSync();
    if (!doubleBufferingEnabled && HAL::getInstance())
    {
        HAL::getInstance()->lockDMAToFrontPorch(refreshRequested);
    }
    if (refreshRequested && !displayRefreshing)
    {
        LCD_SetUpdateRegionFull();
        // Transfer a quarter screen of pixel data.
        HAL_DSI_Refresh(hdsi);
        displayRefreshing = true;
    } else
    {
        GPIO::clear(GPIO::VSYNC_FREQ);
    }
}
```

```
void HAL_DSI_EndOfRefreshCallback(DSI_HandleTypeDef *hdsi)
{
    if (displayRefreshing)
    {GPIO::clear(GPIO::VSYNC_FREQ);
    displayRefreshing = false;
    if (HAL::getInstance())
    {
        // Signal to the framework that display update has finished.
        HAL::getInstance()->frontPorchEntered();
    }
    }
}
```



实现示例（续）

98



• 完善工程

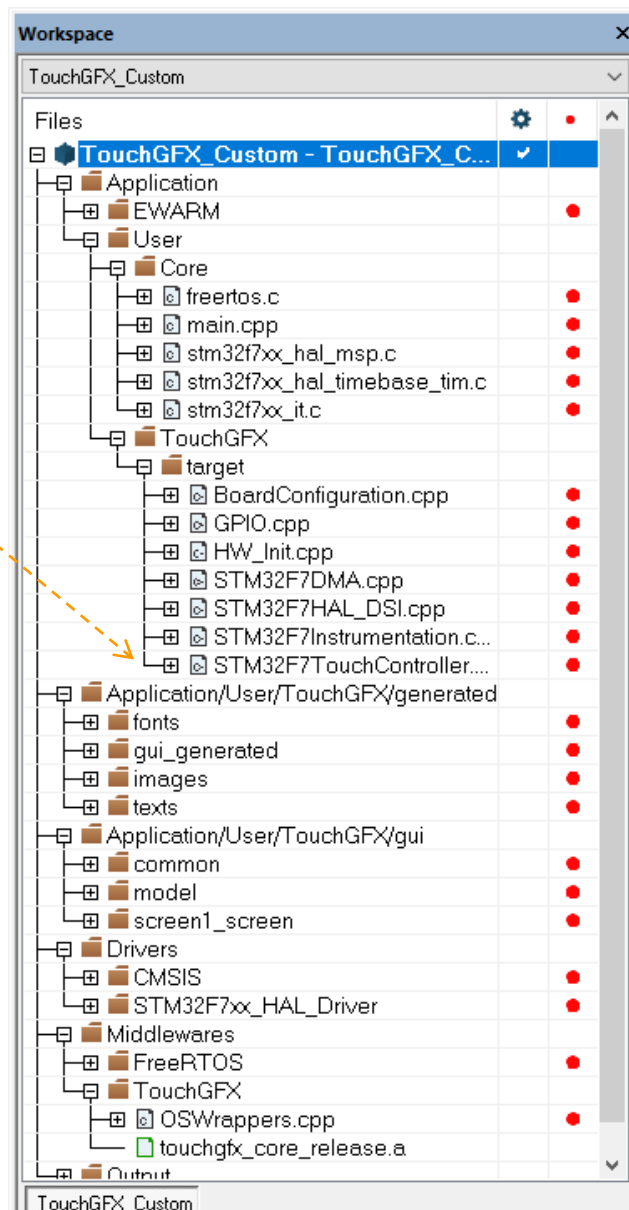
- 在完善工程前，首先介绍如下内容。

- IDE工程的文件夹目录结构
- IDE工程中文件结构
- 显示驱动调用信息汇总
- 触摸驱动调用信息汇总

• 触摸驱动添加

- STM32CubeMX生成的TouchGFX应用工程中，预留了触摸驱动功能的调用位置。需要完善的功能包含
 - 触摸板控制器的初始化
 - 触摸点信息更新
- 对应功能在STM32F7TouchController.cpp的两个函数中被调用
 - void STM32F7TouchController::init()
 - bool STM32F7TouchController::sampleTouch(int32_t& x, int32_t& y)

触摸功能被
调用位置





完善工程

- 触摸驱动添加

驱动功能	调用函数/位置	源文件
触摸板初始化	void STM32F7TouchController::init()	STM32F7TouchController.cpp
触摸检测及坐标更新	bool STM32F7TouchController::sampleTouch(int32_t& x, int32_t& y)	

演示例中，直接采用ST提供的触摸板驱动文件ft6x06.c/.h。这里不对驱动本身进行介绍，而是侧重于介绍生成工程中需要添加的驱动功能和添加位置。

- ft6x06.c/.h获取路径： STM32CubeF7固件包\Drivers\BSP\Components\ft6x06
- 首先，将驱动文件ft6x06.c/.h文件复制到工程文件夹，如右图1所示。
- 在IAR工程中，添加驱动文件ft6x06.c到工程，如右图2所示。
- 在IAR工程的菜单栏\Option\Project\CC++ Compiler\Preprocessor\Additional include directories中添加驱动头文件所在路径。
- 在main.h中包含显示的驱动头文件 ft6x06.h
 - #include “ft6x06.h”

注： 由于ft6x06.c/.h中用到了ST固件包中的ts.h，也需要将其添加复制到工程文件夹，并添加到工程和头文件路径中。

- ts.h获取路径： STM32CubeF7固件包\Drivers\BSP\Components\Common

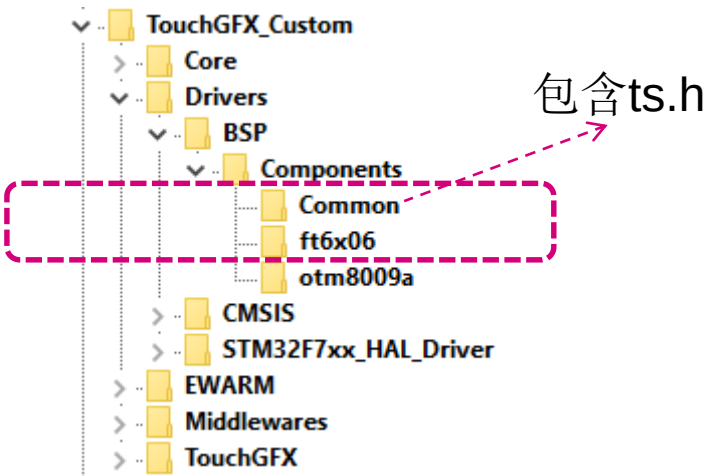


图1

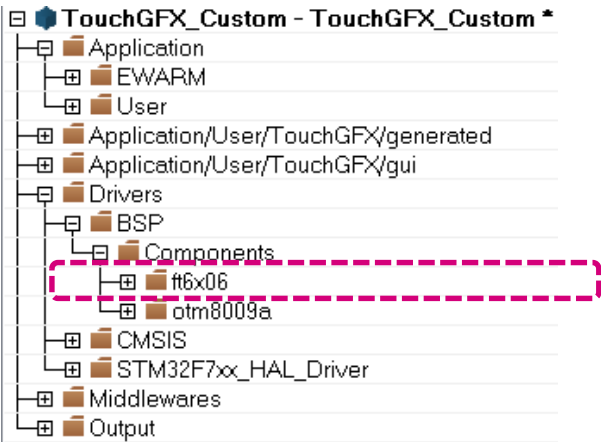


图2



- 完善工程
 - 触摸驱动添加

驱动功能	调用函数/位置	源文件
触摸板初始化	void STM32F7TouchController::init()	STM32F7TouchController.cpp
触摸检测及坐标更新	bool STM32F7TouchController::sampleTouch(int32_t& x, int32_t& y)	

- 演示例中，直接采用ST提供的触摸板驱动文件ft6x06.c/.h。这里不对驱动本身进行介绍，而是侧重于介绍生成工程中需要添加的驱动功能和添加位置。
 - ft6x06.c/.h获取路径： STM32CubeF7固件包 \Drivers\BSP\Components\ft6x06
 - 首先，将驱动文件ft6x06.c/.h文件复制到工程文件夹，如右图1所示。
 - 在IAR工程中，添加驱动文件ft6x06.c到工程，如右图2所示。
 - 在IAR工程的菜单栏\Option\Project\CC++ Compiler\Preprocessor\Additional include directories中添加驱动头文件所在路径。
 - 在main.h中包含显示的驱动头文件 ft6x06.h
 - #include “ft6x06.h”
 - 添加完触摸板驱动文件后，需要定义ft6x06.h中声明的外部函数，与触摸板控制器的通信接口建立关联。演示例中，在main.cpp中添加这部分函数体，如右侧所示。其中，变量hi2c4是CubeMX中选I2C4后生成。



```
/* USER CODE BEGIN 4 */
void TS_IO_Init(void)
{
    //已经在MX_I2C4_Init()中实现。STM32CubeMX中选中I2Cx
    //时，在生成的工程中会生成&调用MX_I2Cx_Init()函数。
}
void TS_IO_Write(uint8_t Addr, uint8_t Reg, uint8_t Value)
{
    HAL_I2C_Mem_Write (&hi2c4, Addr, (uint16_t)Reg,
    I2C_MEMADD_SIZE_8BIT,(uint8_t*)&Value, 1,1000);
}

uint8_t TS_IO_Read(uint8_t Addr, uint8_t Reg)
{
    uint8_t read_value = 0;
    HAL_I2C_Mem_Read(&hi2c4, Addr, Reg,
    I2C_MEMADD_SIZE_8BIT, (uint8_t*)&read_value, 1,1000);
    return read_value;
}

uint16_t TS_IO_ReadMultiple(uint8_t Addr, uint8_t Reg,
uint8_t *Buffer, uint16_t Length)
{
    return HAL_I2C_Mem_Read(&hi2c4, Addr, (uint16_t)Reg,
    I2C_MEMADD_SIZE_8BIT, Buffer, Length,1000);
}
void TS_IO_Delay(uint32_t Delay)
{
    HAL_Delay(Delay);
}
/* USER CODE END 4 */
```



- 完善工程
 - 触摸驱动添加

驱动功能	调用函数/位置	源文件
触摸板初始化	void STM32F7TouchController::init()	STM32F7TouchController.cpp
触摸检测及位置更新	bool STM32F7TouchController::sampleTouch(int32_t& x, int32_t& y)	

- 演示例中，直接采用ST提供的触摸板驱动文件ft6x06.c/.h。这里不对驱动本身进行介绍，而是侧重于介绍生成工程中需要添加的驱动功能和添加位置。

```
void STM32F7TouchController::init()
{
    /* USER CODE BEGIN F4TouchController_init */
    /* Add code for touch controller Initialization */
    //ft6x06_ts_drv.Init //生成工程已经包含对应通信接口初始化
    /* Software reset the TouchScreen */
    ft6x06_ts_drv.Reset(0x54); //触摸板控制器I2C地址 0x54
    /* Calibrate, Configure and Start the TouchScreen driver */
    ft6x06_ts_drv.Start(0x54);
    /* USER CODE END F4TouchController_init */
}
```



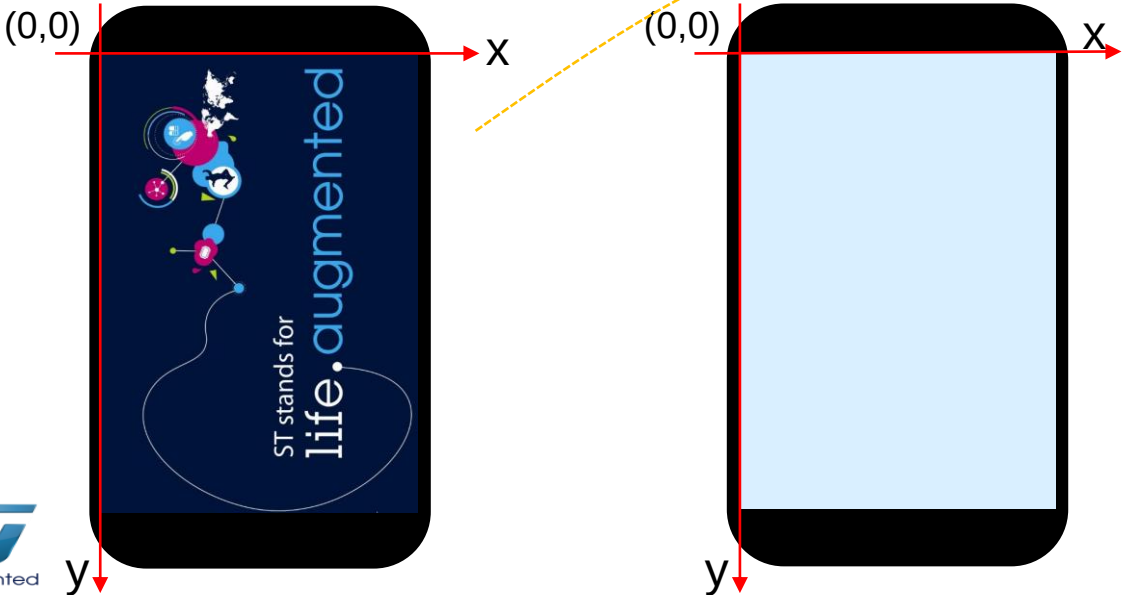

- 完善工程
 - 触摸驱动添加

驱动功能	调用函数/位置	源文件
触摸板初始化	void STM32F7TouchController::init()	STM32F7TouchController.cpp
触摸检测及位置更新	bool STM32F7TouchController::sampleTouch(int32_t& x, int32_t& y)	

```
bool STM32F7TouchController::sampleTouch(int32_t& x, int32_t& y)
{
    /* USER CODE BEGIN F4TouchController_sampleTouch */
    uint8_t touchDetected; /*!< Total number of active touches
    detected at last scan */
    uint16_t touchX;
    uint16_t touchY;

    touchDetected = ft6x06_ts_drv.DetectTouch(0x54);
    if (touchDetected)
    {
        /* Get each touch coordinates */
        ft6x06_ts_drv.GetXY(0x54, &touchX, &touchY);
        x = touchX;
        y = touchY;
        return true;
    }
    return false;
    /* USER CODE END F4TouchController_sampleTouch */
}
```

- 图中左侧为显示区域坐标方向
- 图中右侧为触摸面板默认坐标方向



演示例中，显示区域坐标方向与触摸面板的坐标方向一致

- 注：显示区域坐标方向，不一定与TouchGFX Designer中坐标方向一致。还会受到显示刷新方向等配置的影响。
- 演示例中，显示区域的坐标方向与TouchGFX Designer中坐标方向一致。



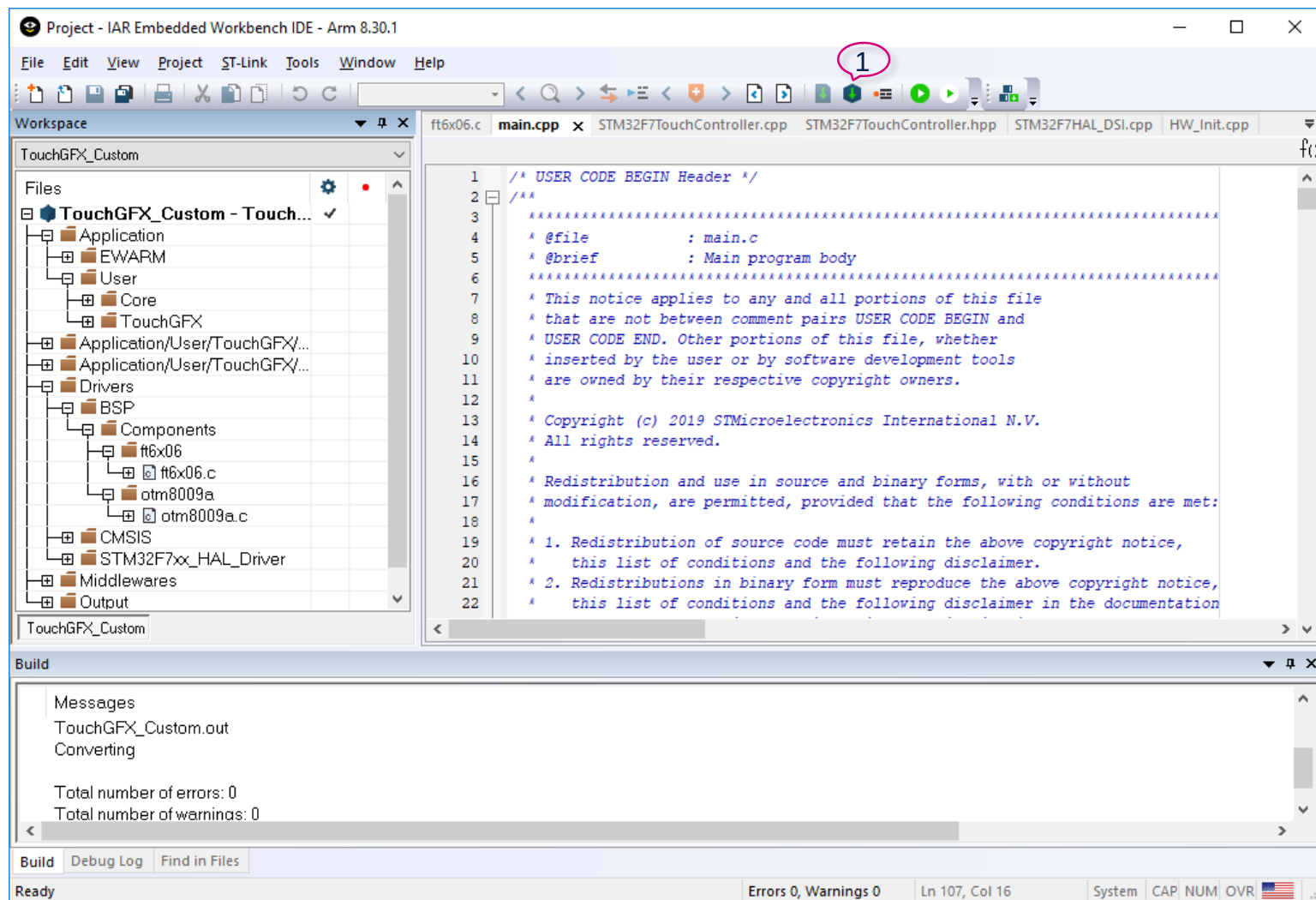
实现示例（续）

103



• 工程编译

- 点击“Make”进行工程编译获取应用固件





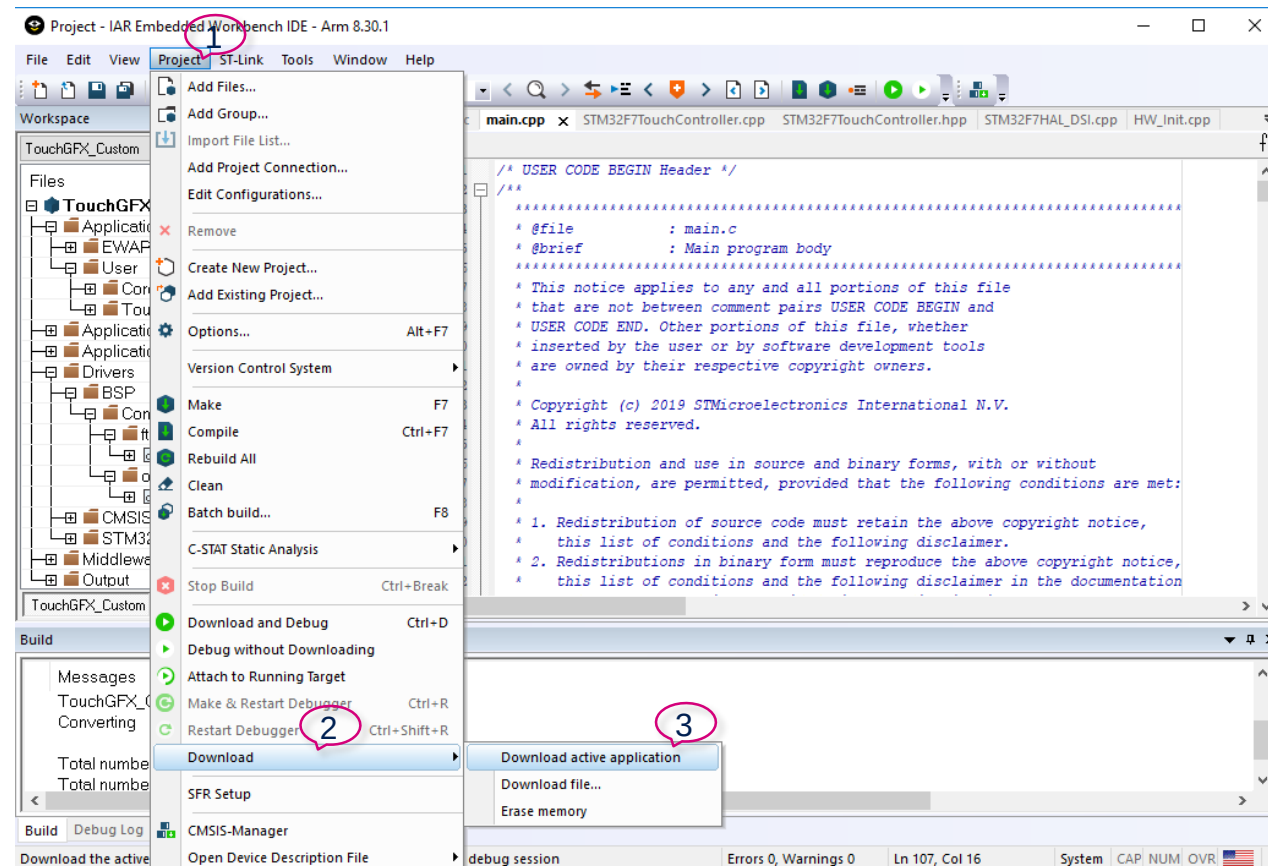
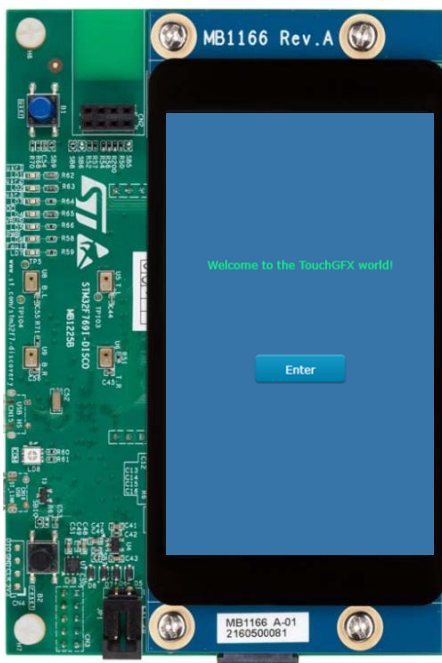
实现示例（续）

104



• 烧录&运行固件

- 利用Micro USB数据线，连接STM32F769I-DISCO板上CN16 USB ST_LINK至当前PC
- 点击菜单栏\Project\Download\Download application，进行固件烧录
- 烧录完成后，按一下STM32F769I-DISCO板上黑色RESET按键，重新启动运行应用。显示如下图。





实现示例（续）

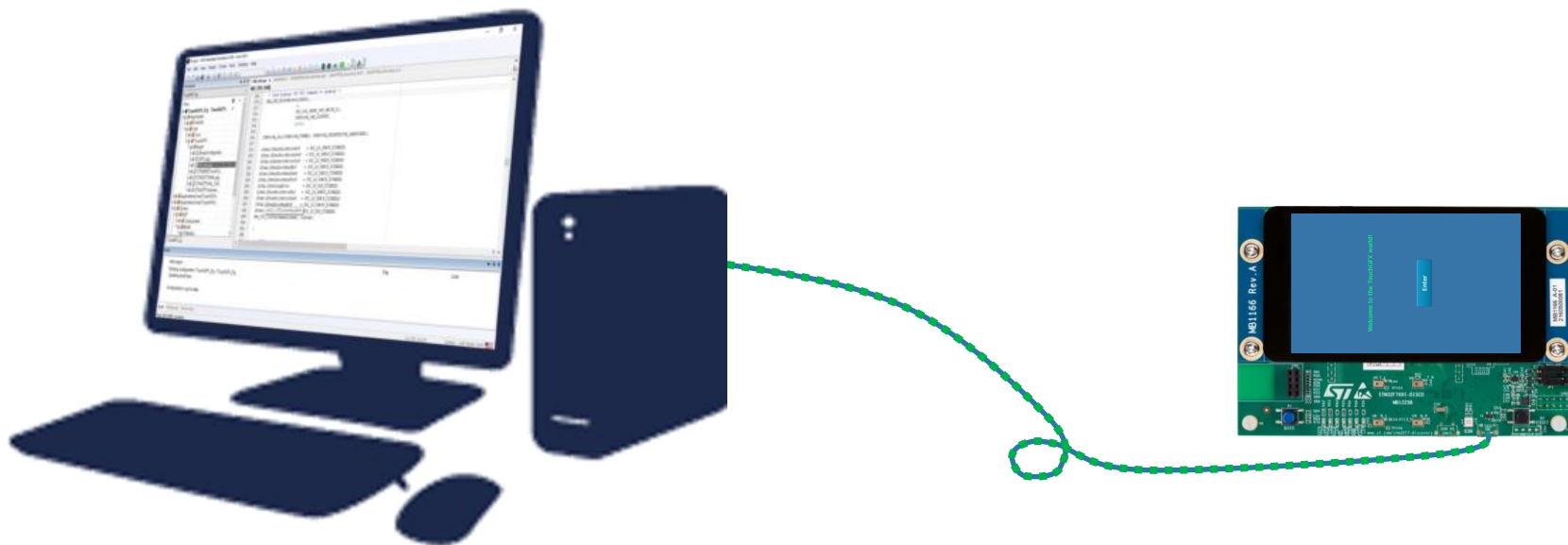
105

• 主机

- 安装STM32CubeMX 5.0 或以上版本
- 安装STM32CubeProgrammer或者ST-Link Utility
- 安装对应的IDE（Eg. IAR/KEIL）
- 包含STM32Cube软件包

• GUI板

- STM32F769I-DISCO（演示用）
 - 480x800 MIPI屏 (带触摸面板)
 - 外扩SDRAM





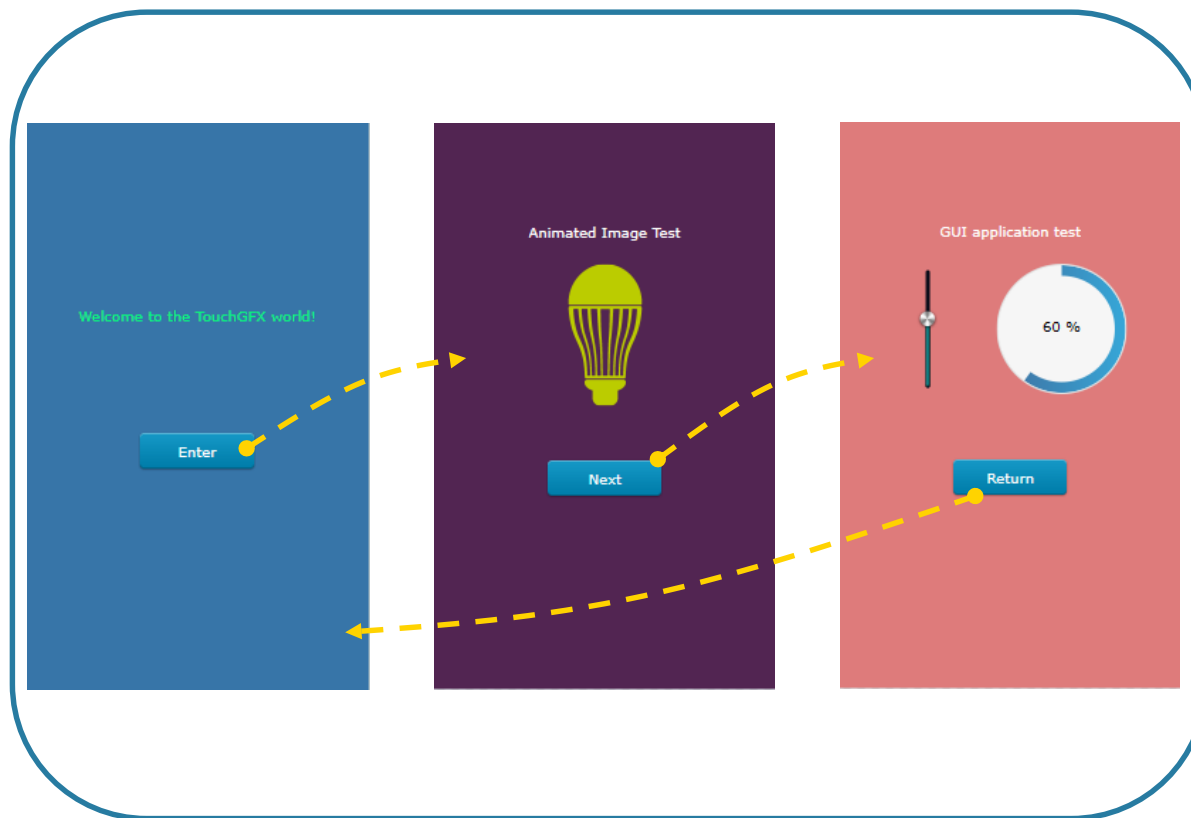
扩展_TouchGFX应用例1



扩展_TouchGFX应用例1（续）

107

- 基于已有环境增加GUI应用，实现如下所示的交互效果。



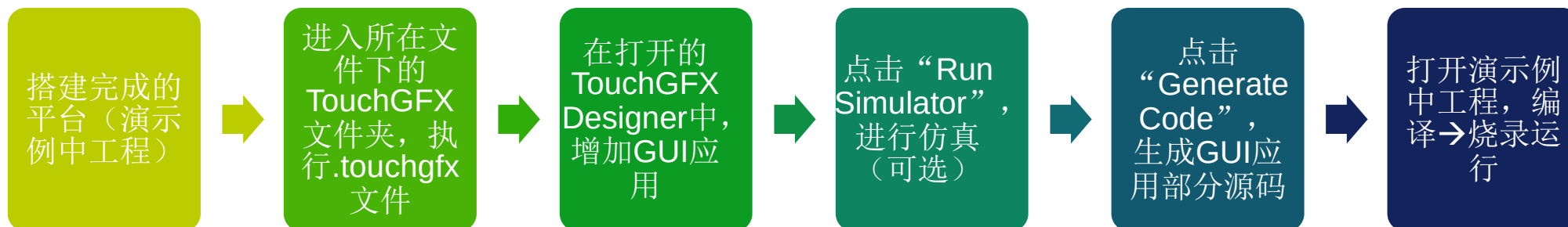


扩展_TouchGFX应用例1

108

- 基于已有环境增加GUI应用

- 之前的演示例中，展现了如何利用STM32CubeMX搭建TouchGFX应用开发平台。主要展现了：
 - STM32CubeMX搭建自定义STM32平台，实现对TouchGFX中间件的支持。
 - 显示和触摸驱动功能的添加和TouchGFX关联调用。
 - TouchGFX Designer中简单的GUI应用实现。
- 接下来，扩展介绍如何在已搭建平台上，增加GUI应用实现。
 - 首先，如果重新从STM32CubeMX执行TouchGFX，会导致已经存在的源文件被覆盖。这会导致，之前TouchGFX Designer设计的一些GUI内容丢失。通过直接运行生成的TouchGFX Designer工程.touchgfx，无信息丢失的打开TouchGFX Designer。
 - 设计流程，如下

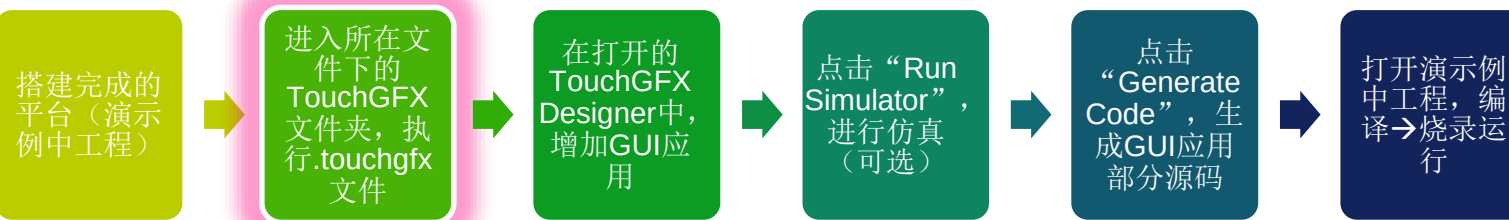
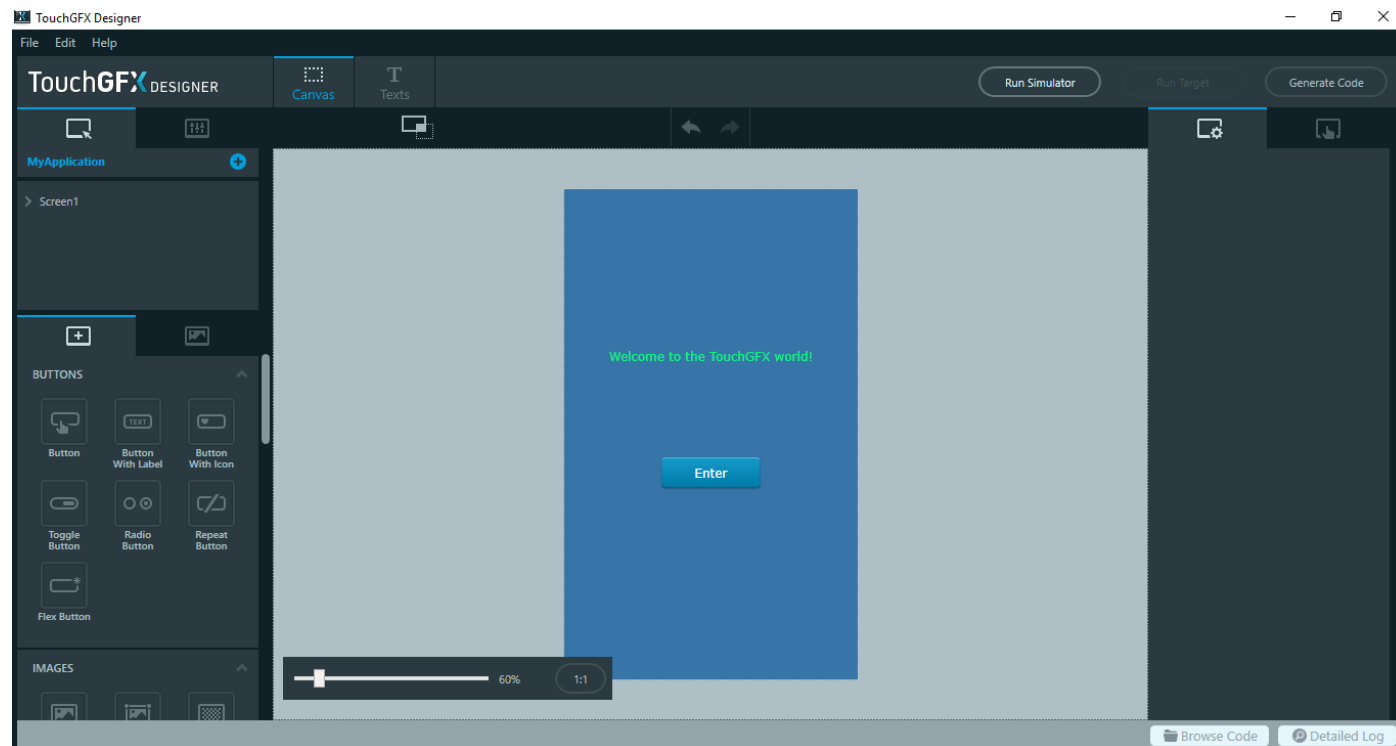
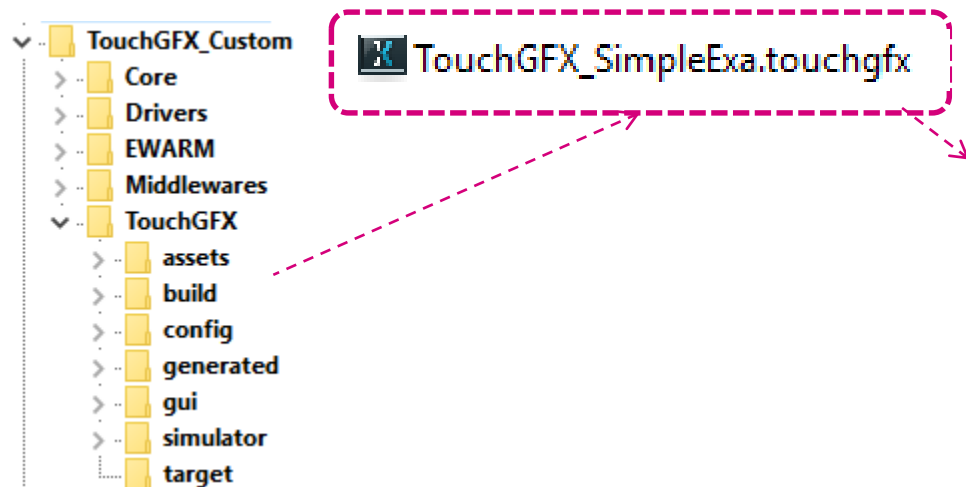




扩展_TouchGFX应用例1（续）

109

- 基于已有环境增加GUI应用
 - 进入所在文件下的TouchGFX文件夹，执行.touchgfx文件

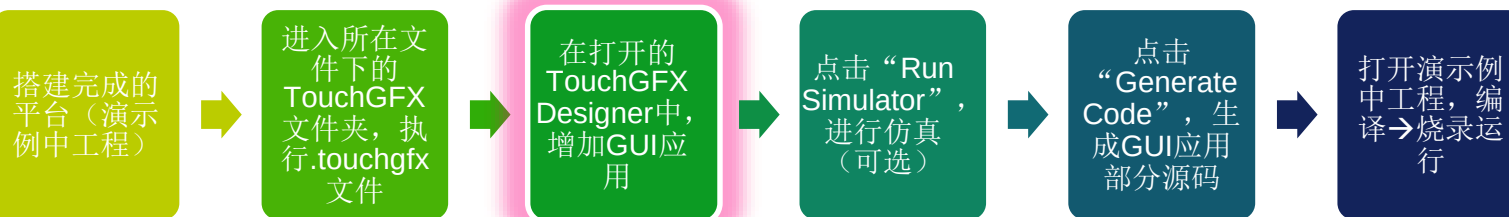
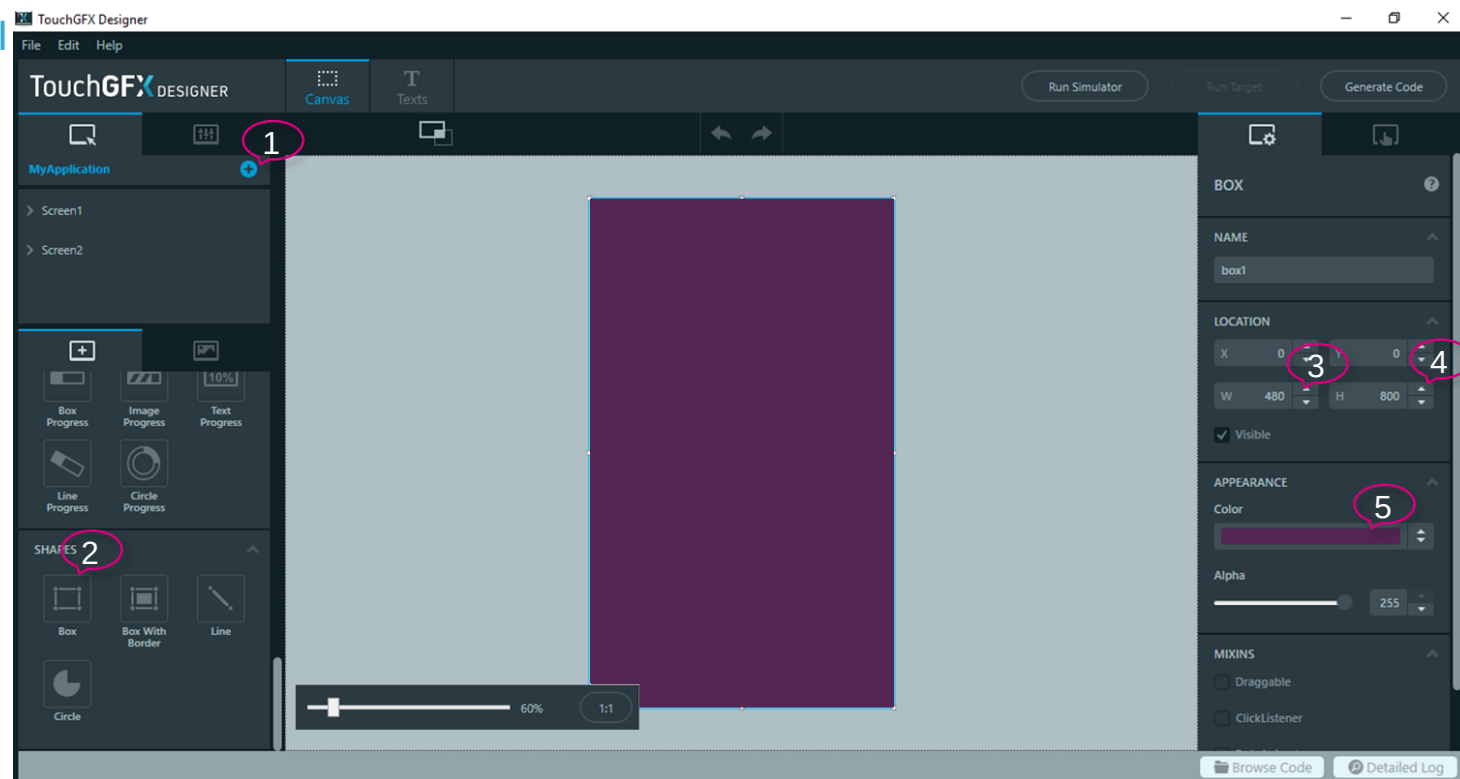




扩展_TouchGFX应用例1（续）

110

- 基于已有环境增加GUI应用
 - 在打开的TouchGFX Designer中，增加GUI应用
 - 添加新的界面Screen2
 - 设置背景色并设置大小

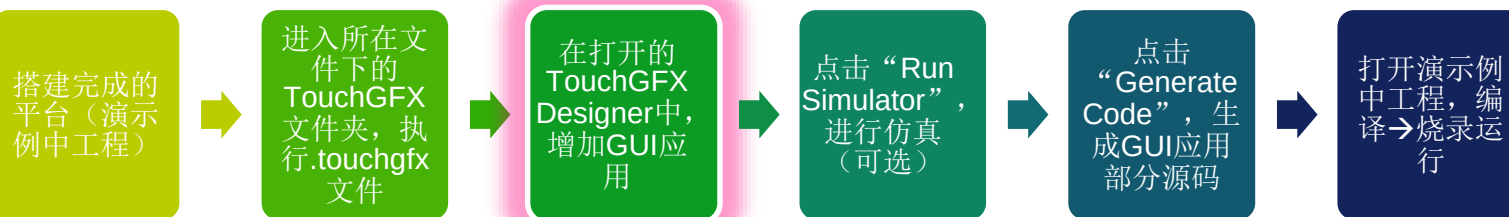
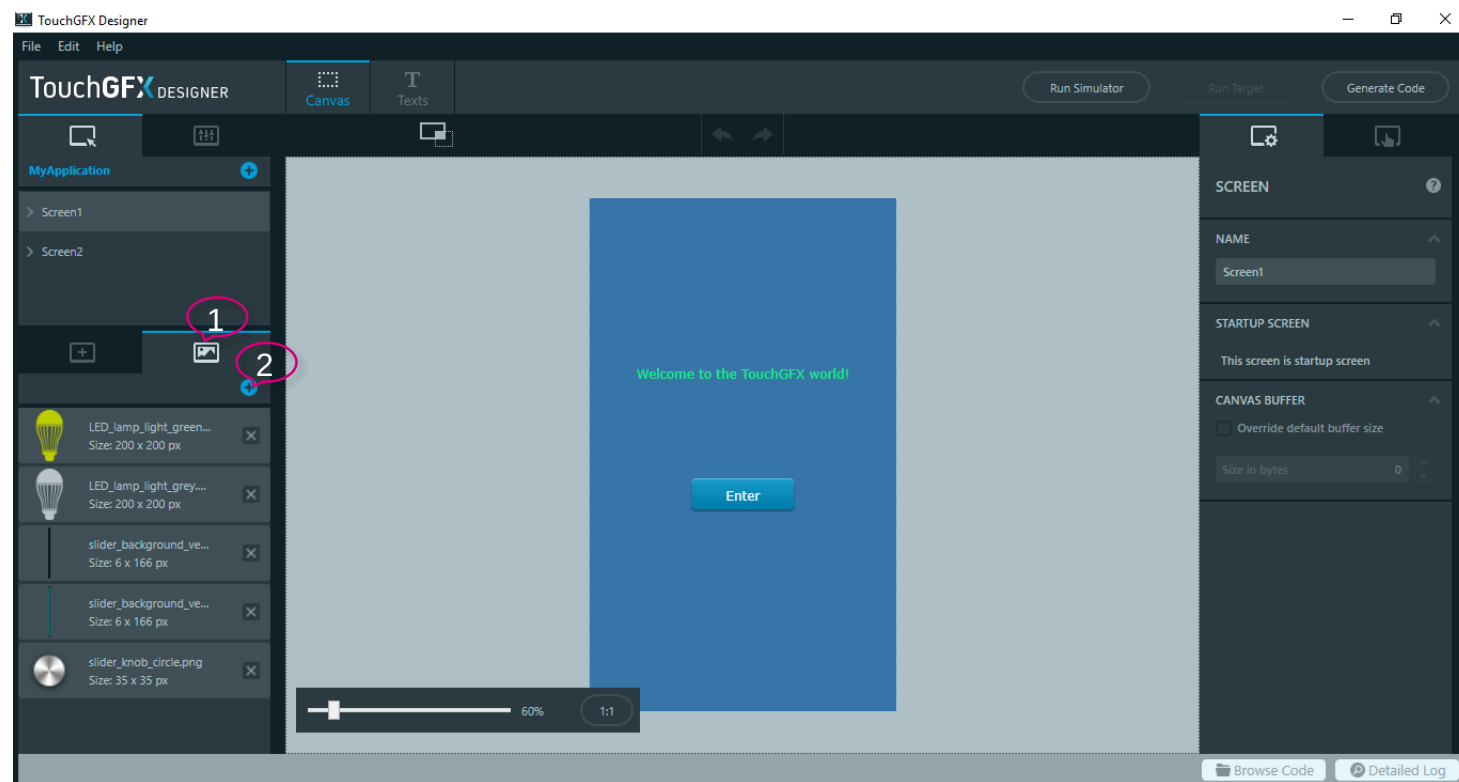




扩展_TouchGFX应用例1（续）

111

- 基于已有环境增加GUI应用
 - 在打开的TouchGFX Designer中，增加GUI应用
 - 添加图片素材
 - 点击“Enter”按钮
 - 按照右侧INTERACTIONS，配置按钮动作





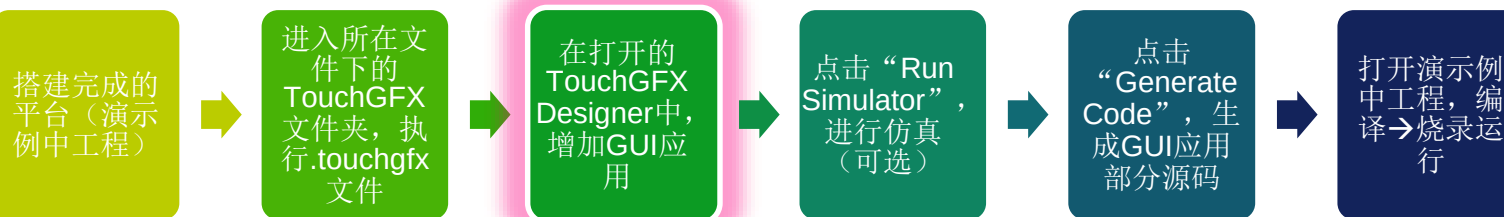
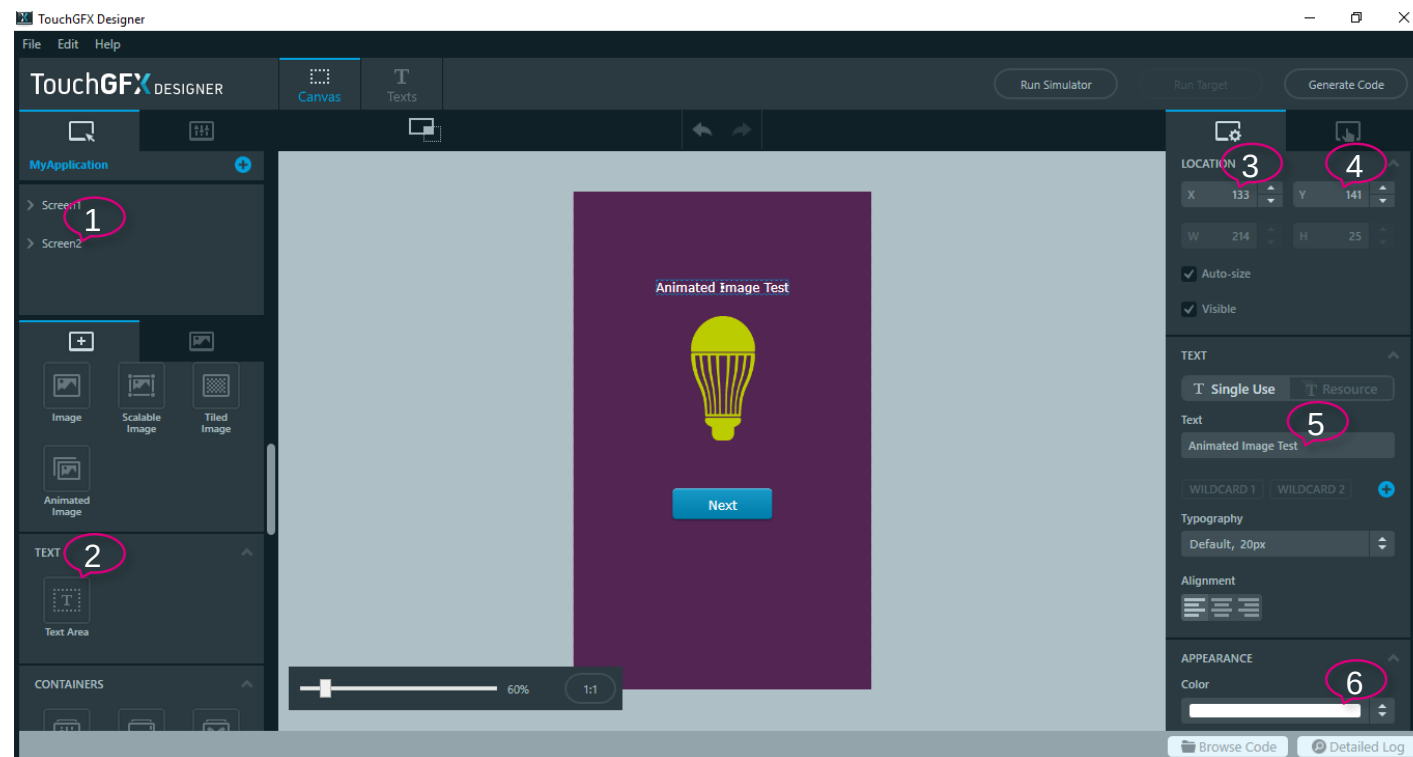
扩展_TouchGFX应用例1（续）

112

- 基于已有环境增加GUI应用

- 在打开的TouchGFX Designer中，增加GUI应用

- 选择界面Screen2
 - 添加文本
 - 设置文本位置
 - 设置文本内容
 - 设置文本颜色





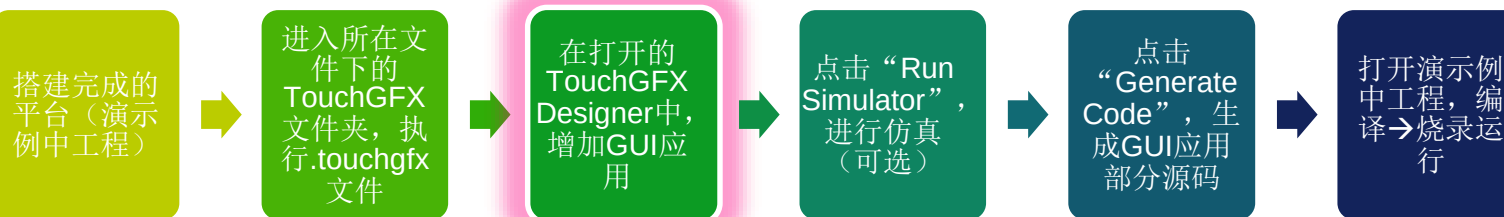
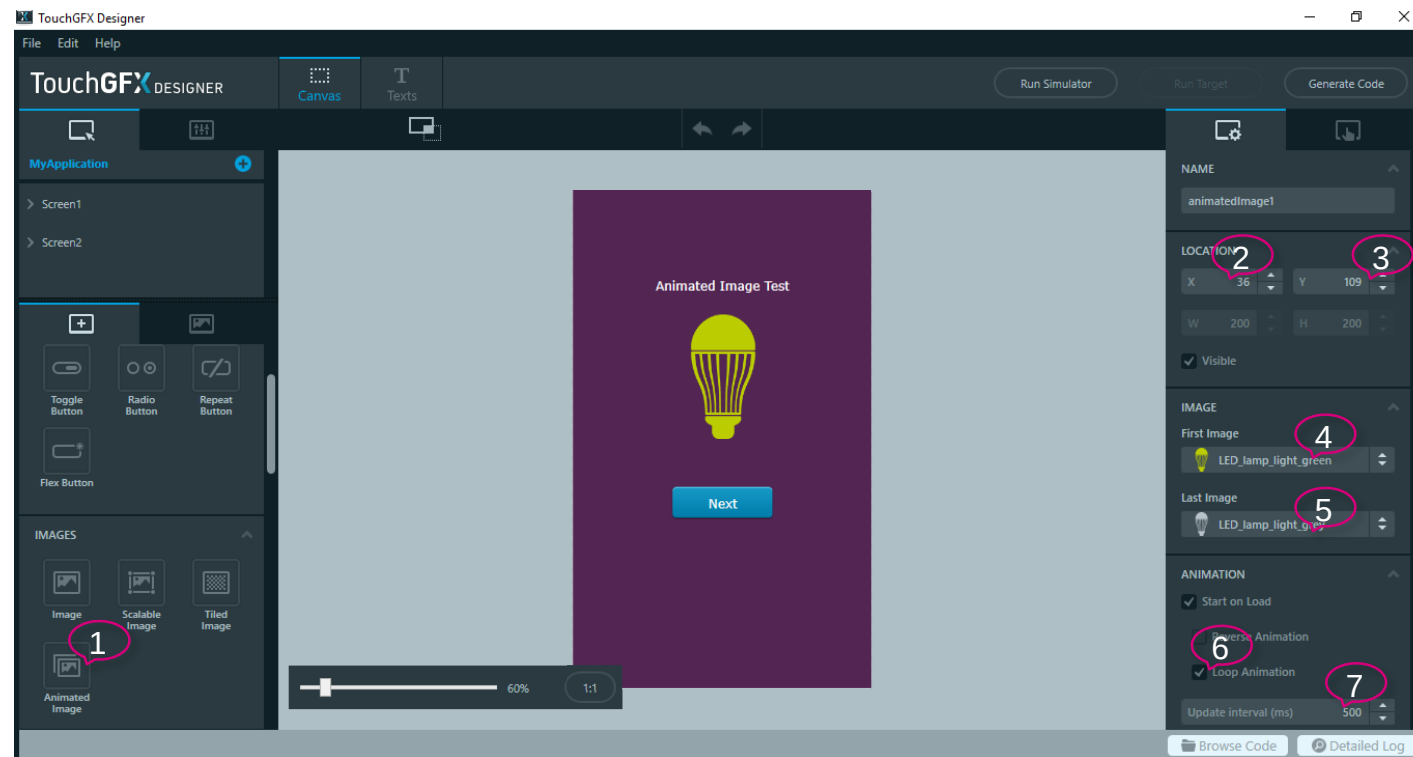
扩展_TouchGFX应用例1（续）

113

• 基于已有环境增加GUI应用

• 在打开的TouchGFX Designer中，增加GUI应用

- 增加动态图片
- 设置动态图片位置
- 设置动态图片对应的前后图片
- 选择动态图片为循环显示
- 设置交替显示周期（单位ms）
- 根据上述步骤添加&设置右侧选择按键和动态图片





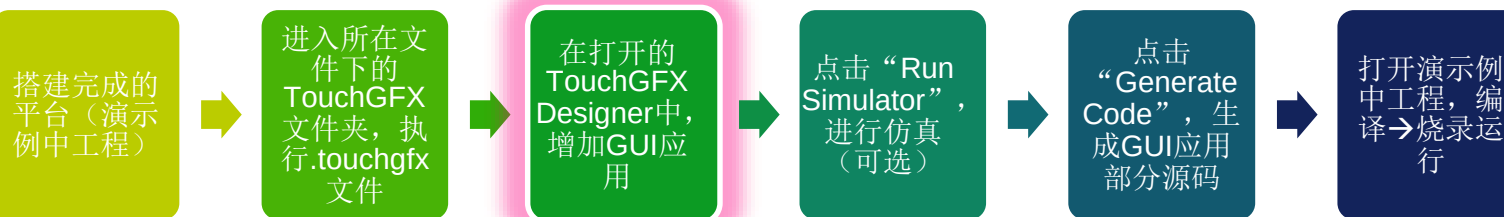
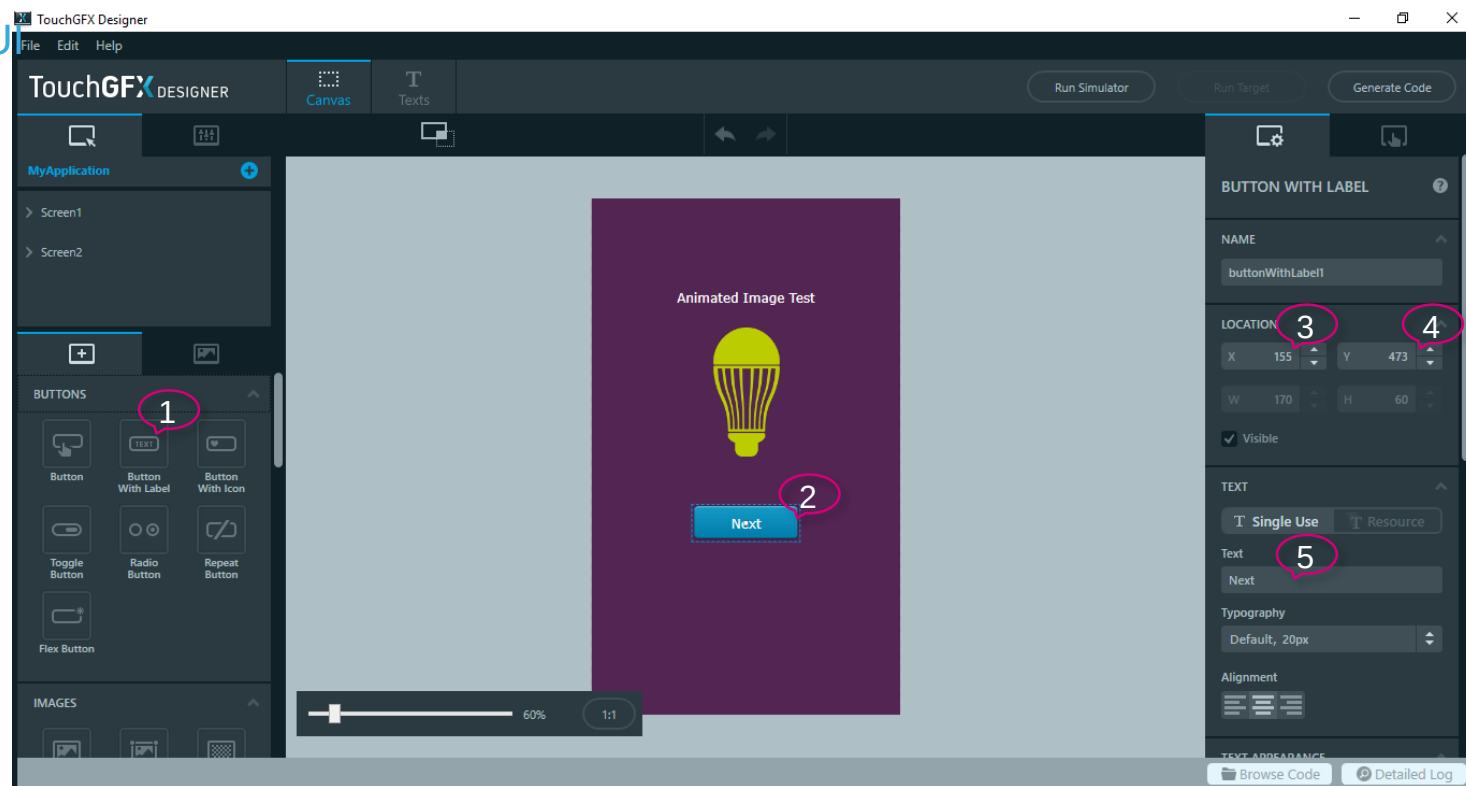
扩展_TouchGFX应用例1（续）

114

- 基于已有环境增加GUI应用

- 在打开的TouchGFX Designer中，增加GUI应用

- 增加标题按钮
 - 设置标题按钮位置
 - 更改按钮名称

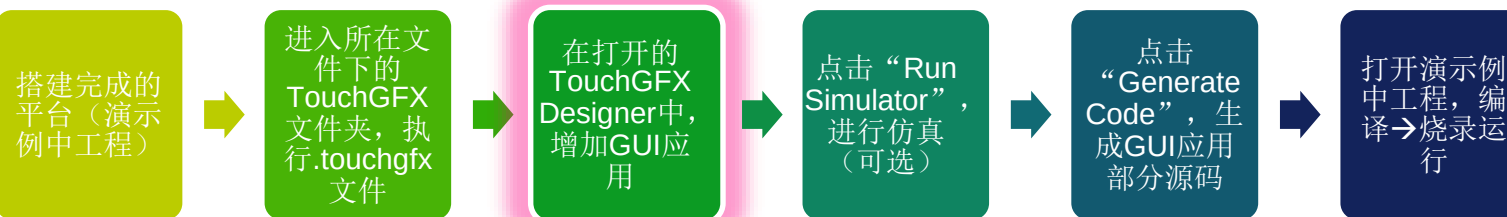
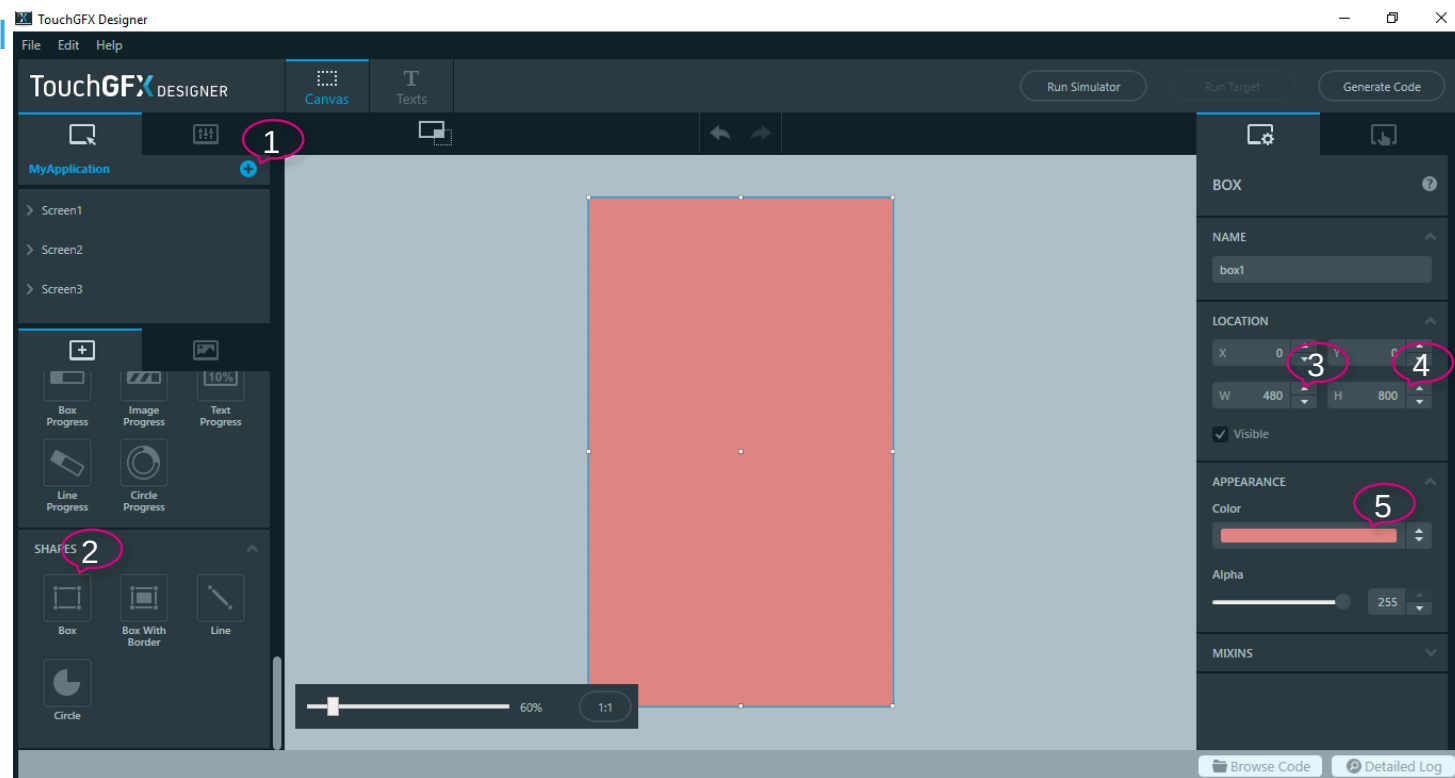




扩展_TouchGFX应用例1（续）

115

- 基于已有环境增加GUI应用
 - 在打开的TouchGFX Designer中，增加GUI应用
 - 添加新的界面Screen3
 - 设置背景色并设置大小

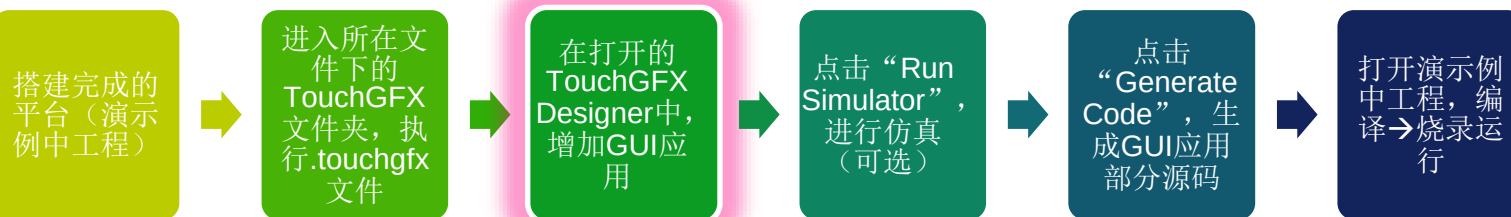
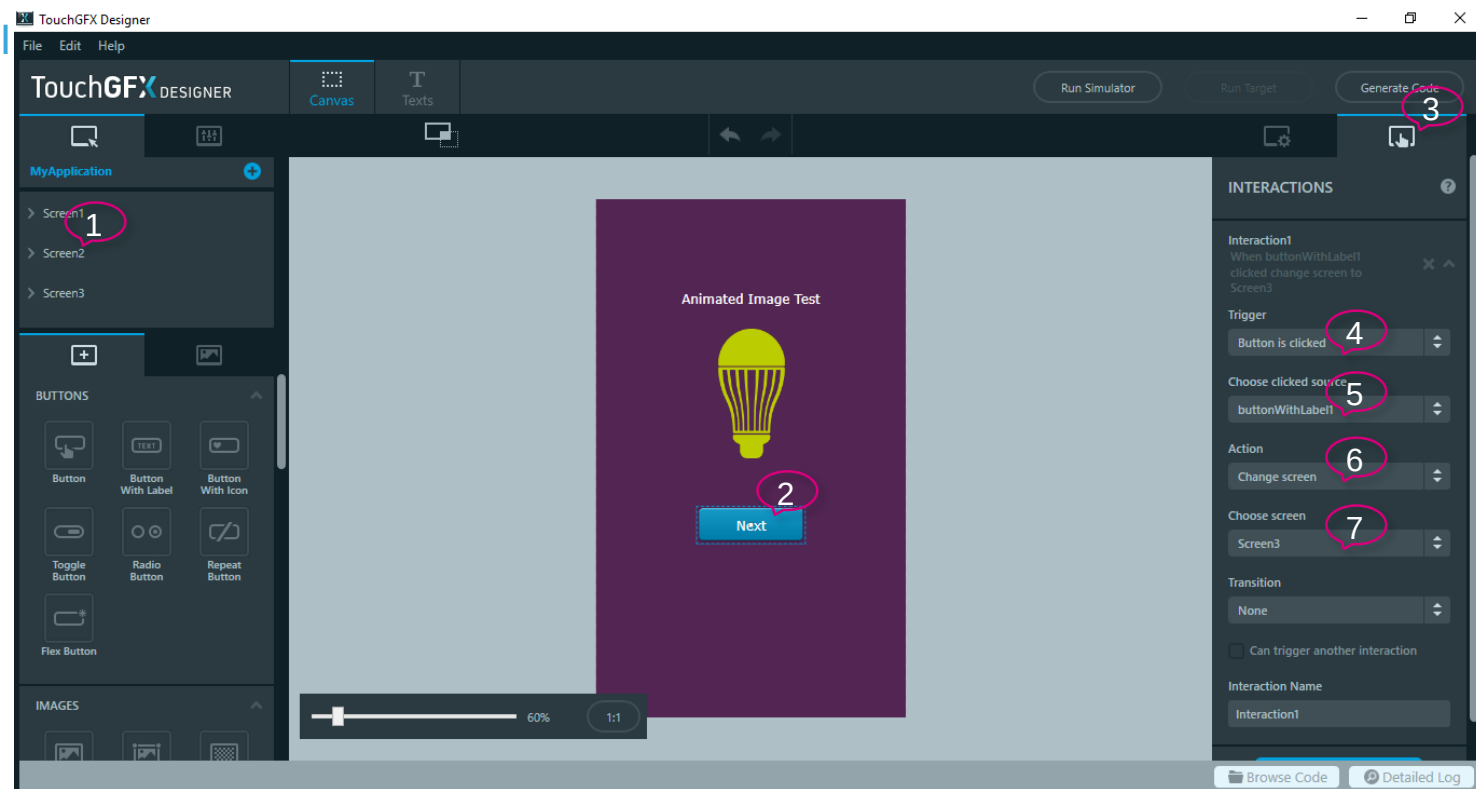




扩展_TouchGFX应用例1（续）

116

- 基于已有环境增加GUI应用
 - 在打开的TouchGFX Designer中，增加GUI应用
 - 添加按钮动作，使点击退回上一界面
 - 在生成GUI应用源码时，可以通过点击“Run Simulator”，运行仿真界面，查看设计效果。

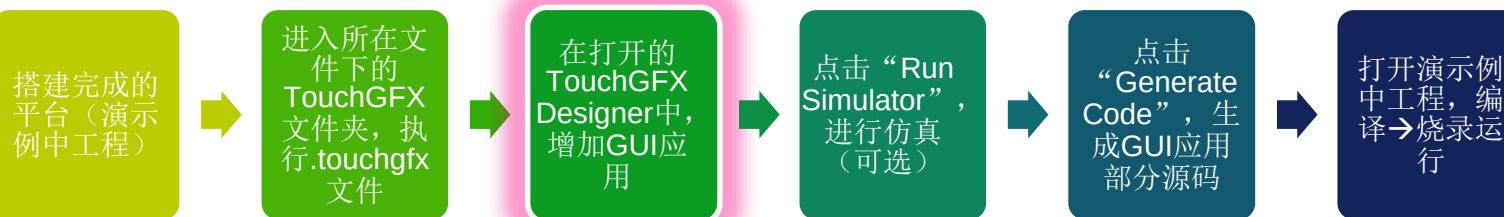
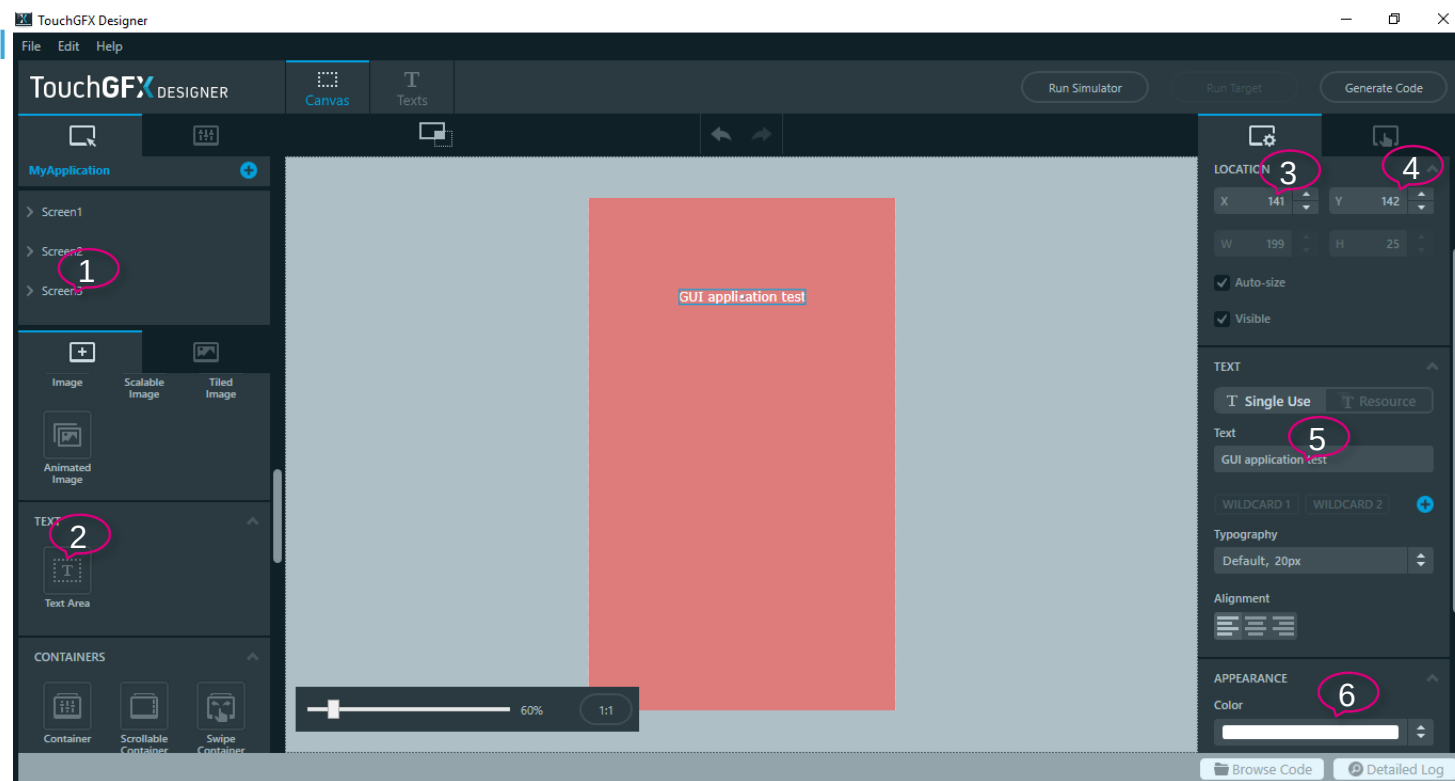




扩展_TouchGFX应用例1（续）

117

- 基于已有环境增加GUI应用
 - 在打开的TouchGFX Designer中，增加GUI应用
 - 添加文本
 - 设置位置、内容和颜色





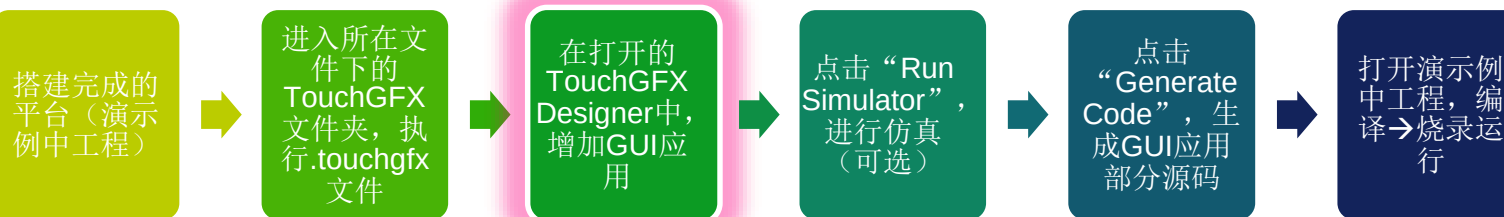
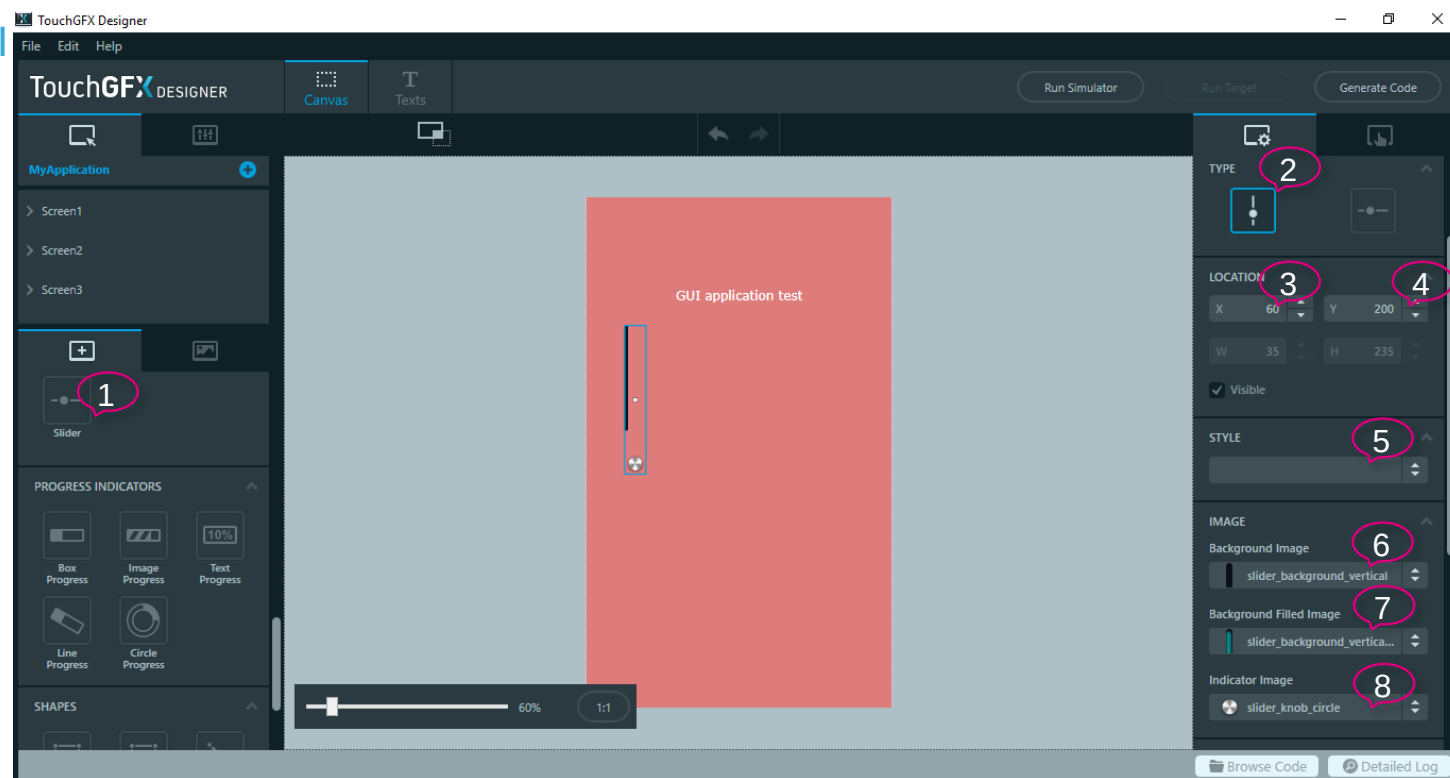
扩展_TouchGFX应用例1（续）

118

- 基于已有环境增加GUI应用

- 在打开的TouchGFX Designer中，增加GUI应用

- 添加滑动控件
 - 设置位置
 - 选择竖直模式
 - 选择No style
 - 设置背景图、填充图、指示图

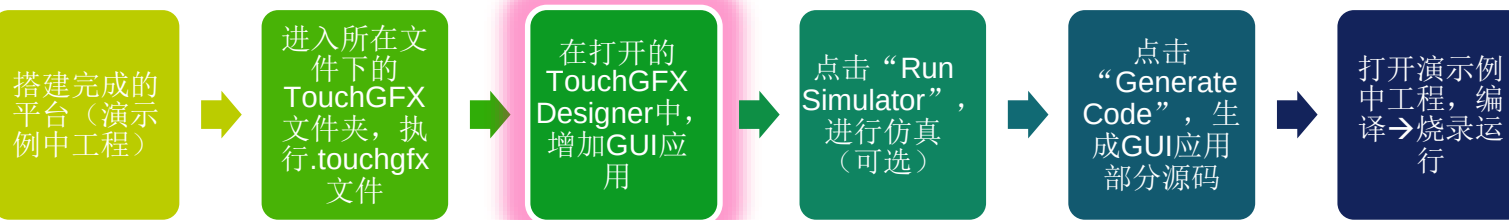
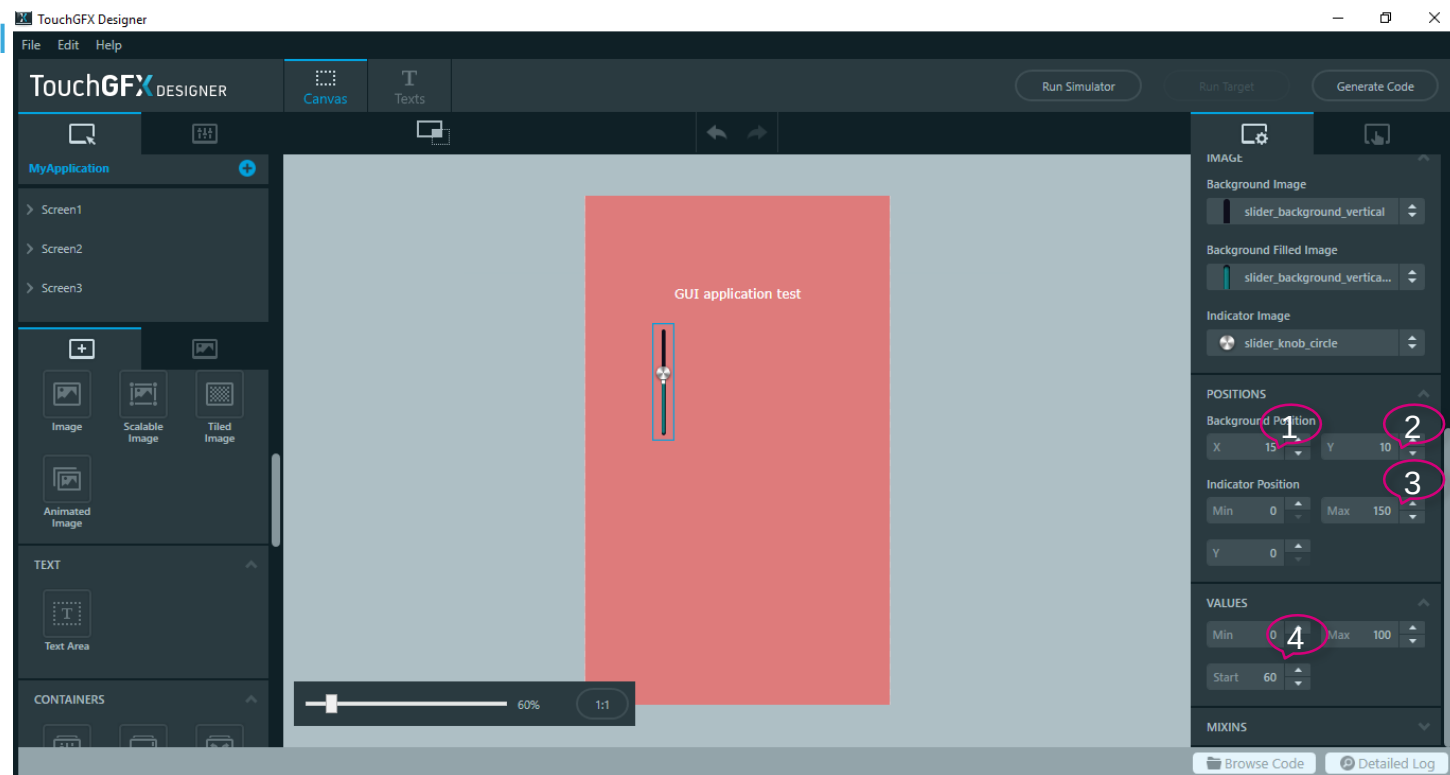




扩展_TouchGFX应用例1（续）

119

- 基于已有环境增加GUI应用
 - 在打开的TouchGFX Designer中，增加GUI应用
 - 设置滑动控件背景位置、指示图标位置
 - 设置滑动控件初始值为60





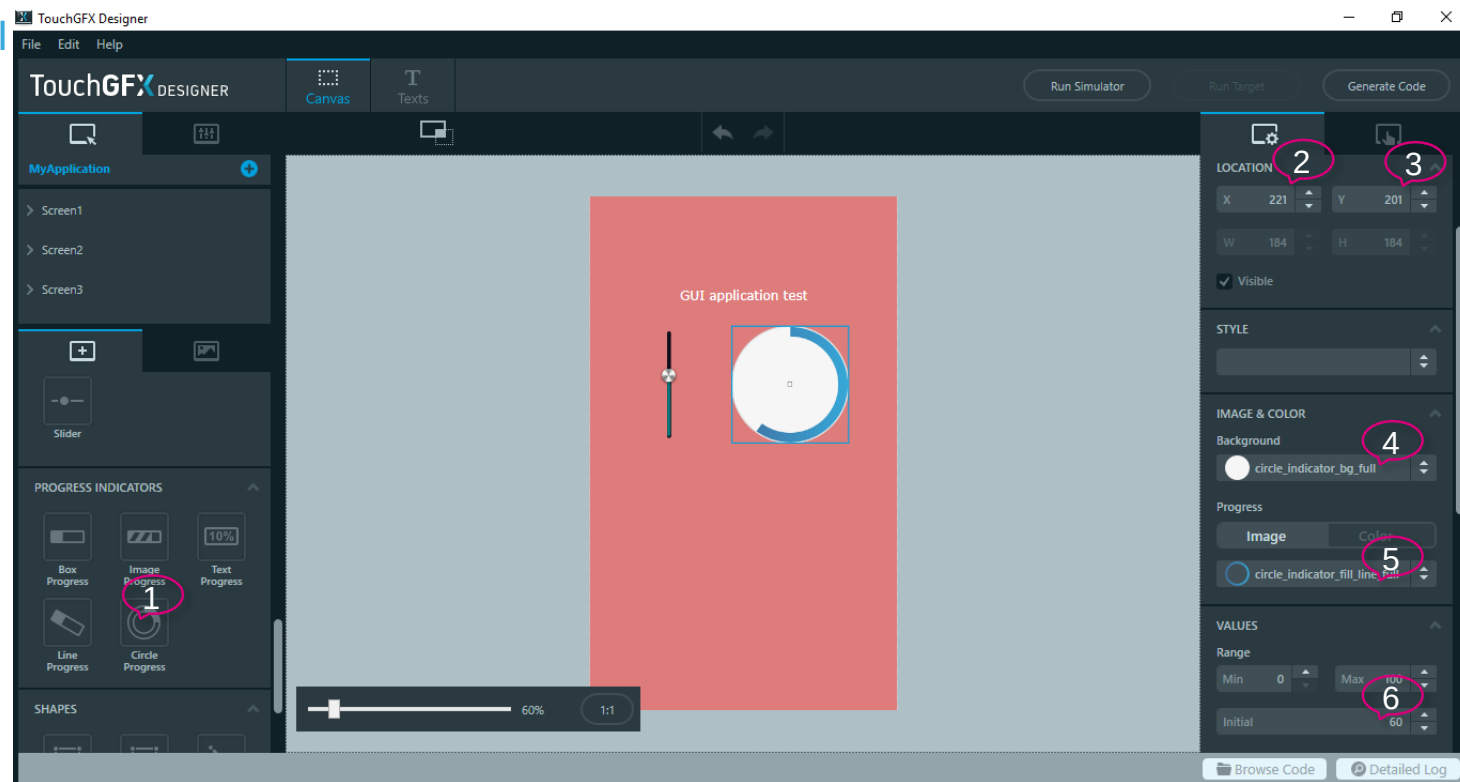
扩展_TouchGFX应用例1（续）

120

- 基于已有环境增加GUI应用

- 在打开的TouchGFX Designer中，增加GUI应用

- 添加圆形进度控件
 - 设置控件位置
 - 设置控件格式
 - 设置起始值



搭建完成的
平台（演示
例中工程）



进入所在文
件下的
TouchGFX
文件夹，执
行.touchgfx
文件



在打开的
TouchGFX
Designer中，
增加GUI应
用



点击“Run
Simulator”，
进行仿真
（可选）



点击
“Generate
Code”，生
成GUI应用
部分源码



打开演示例
中工程，编
译→烧录运
行



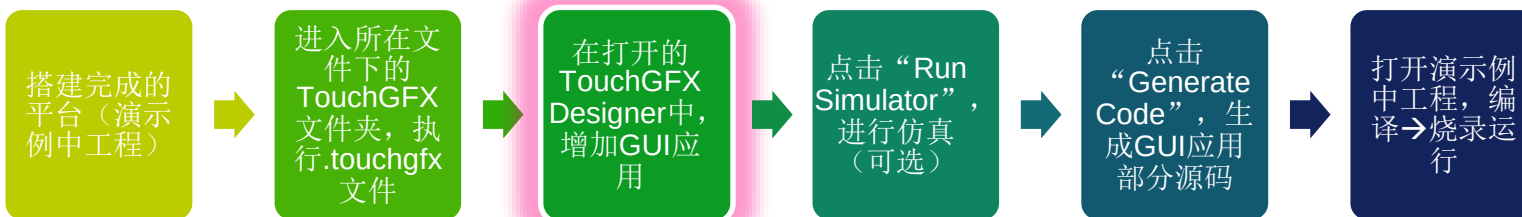
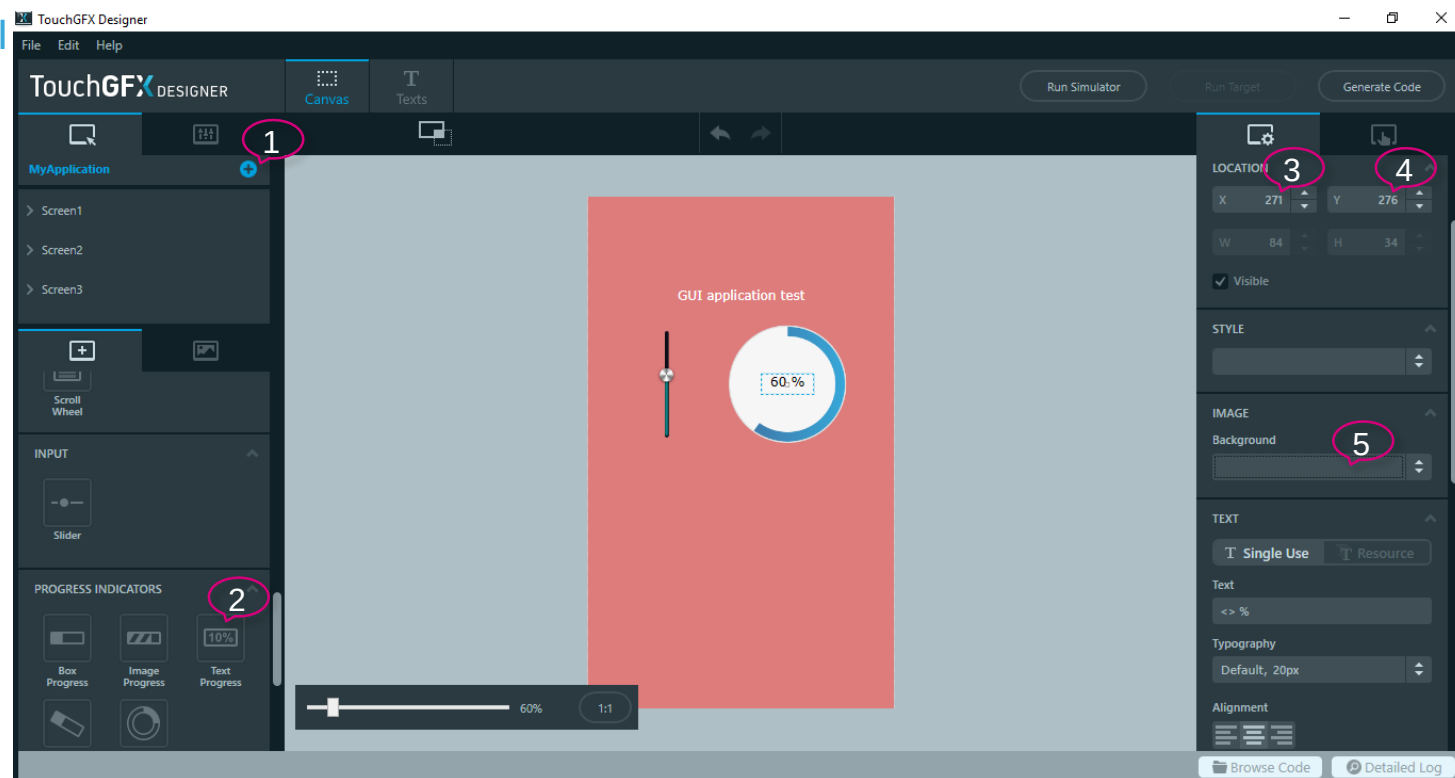
扩展_TouchGFX应用例1（续）

121

- 基于已有环境增加GUI应用

- 在打开的TouchGFX Designer中，增加GUI应用

- 添加文本进度控件
 - 设置控件位置
 - 取消背景





扩展_TouchGFX应用例1（续）

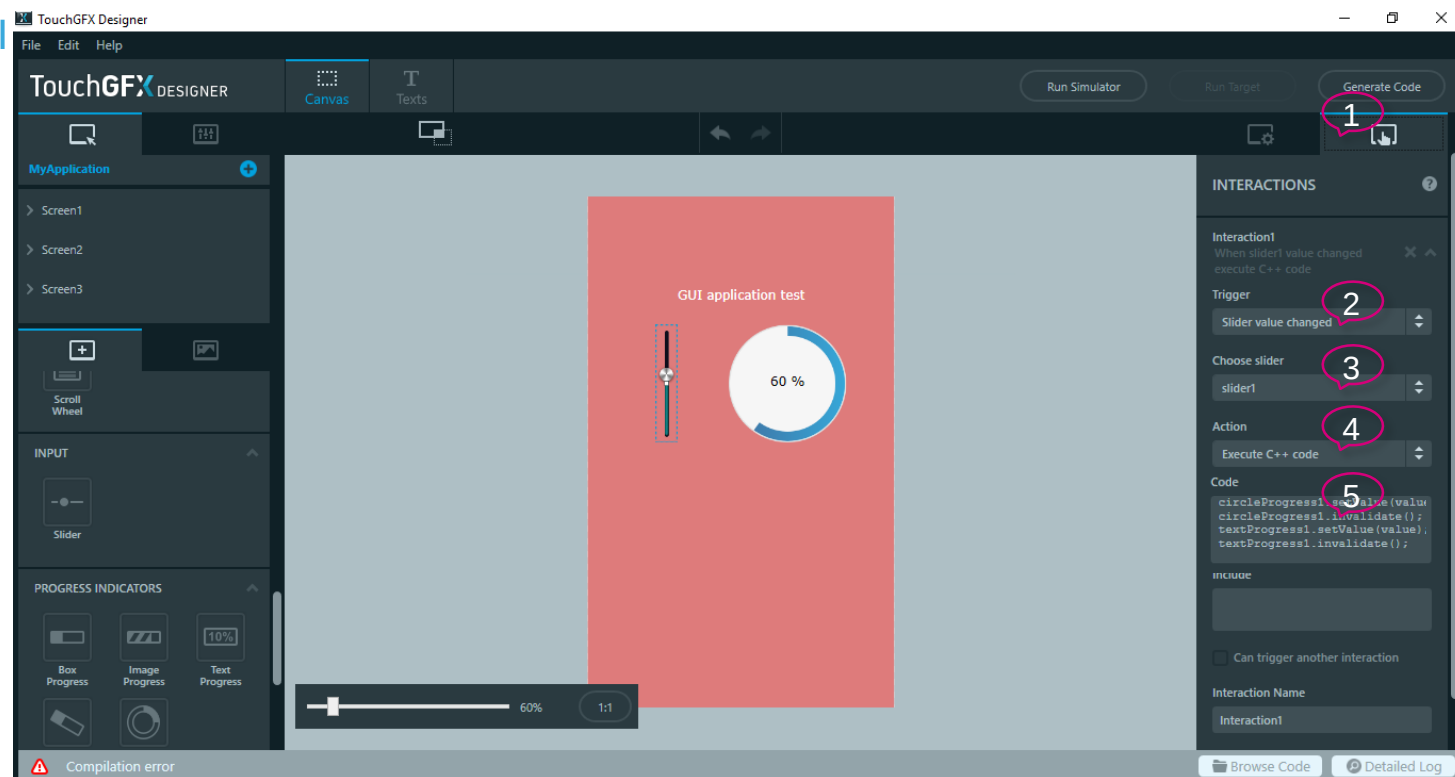
122

• 基于已有环境增加GUI应用

• 在打开的TouchGFX Designer中，增加GUI应用

- 添加交互
 - 关联到 “Slider value changed”
 - 在滑动控件值改变时，执行C++代码
- 动作关联到 “Execute C++ coder”
- 添加执行代码
 - circleProgress1.setValue(value);
 - circleProgress1.invalidate();
 - textProgress1.setValue(value);
 - textProgress1.invalidate();

注： 其中circleProgress1为之前添加的圆形进度控件名； textProgress1为之前添加的文本进度控件名



搭建完成的
平台（演示
例中工程）



进入所在文
件下的
TouchGFX
文件夹，执
行.touchgfx
文件



在打开的
TouchGFX
Designer中，
增加GUI应
用



点击“Run
Simulator”，
进行仿真
（可选）



点击
“Generate
Code”，生
成GUI应用
部分源码



打开演示例
中工程，编
译→烧录运
行



扩展_TouchGFX应用例1（续）

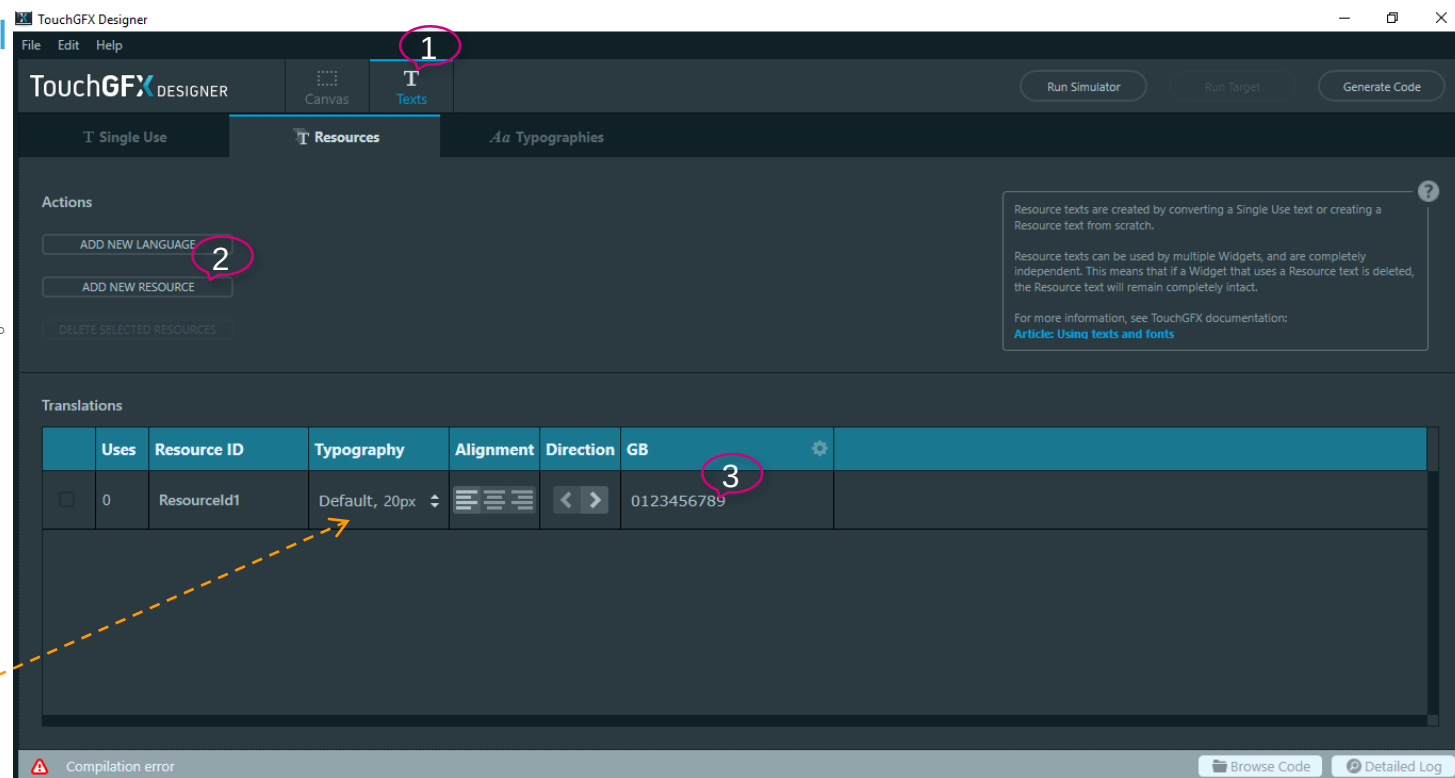
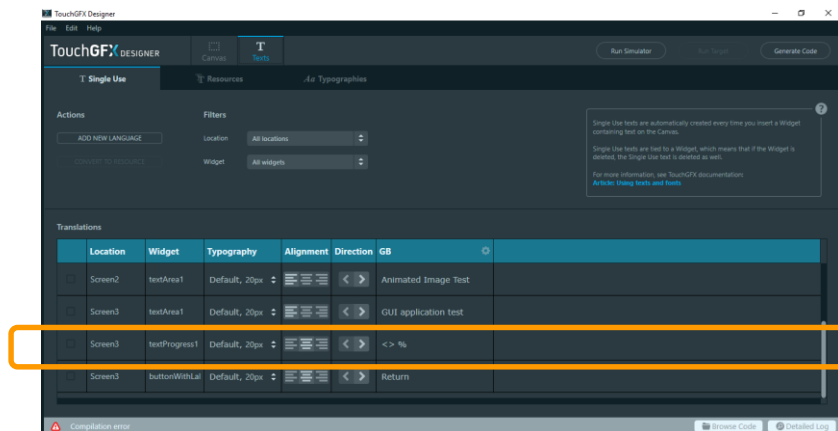
123

• 基于已有环境增加GUI应用

• 在打开的TouchGFX Designer中，增加GUI应用

• 添加字符信息

- 在运行过程中，文本进度控件显示的值根据滑动条的改变而改变，所以需要支持不同数字字符。
- 按照文本进度控件的字体，添加字符支持。包括0123456789



搭建完成的
平台（演示
例中工程）



进入所在文
件下的
TouchGFX
文件夹，执行
.touchgfx
文件



在打开的
TouchGFX
Designer中，
增加GUI应
用



点击“Run
Simulator”，
进行仿真
（可选）



点击
“Generate
Code”，生
成GUI应用
部分源码



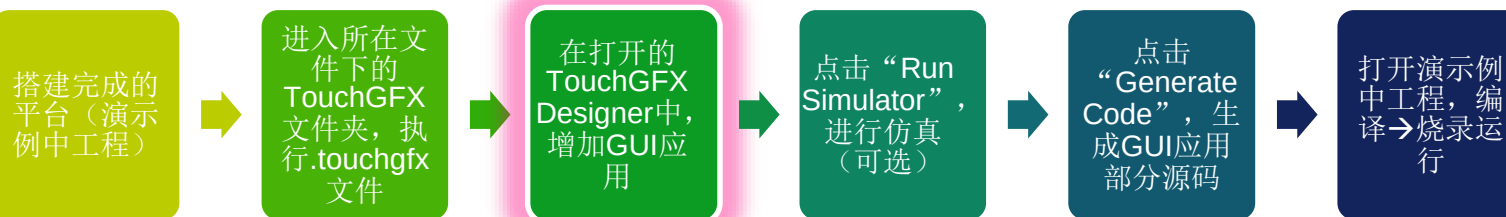
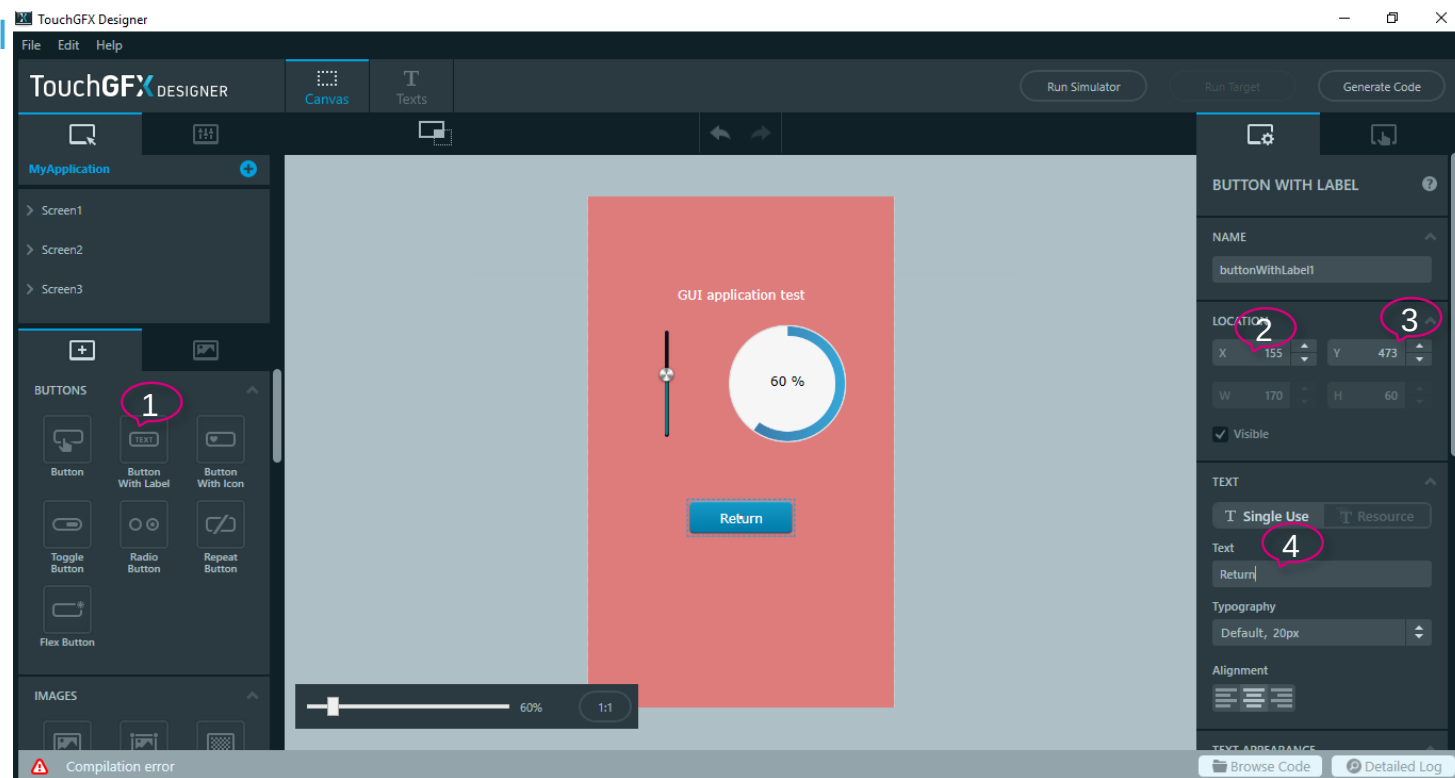
打开演示例
中工程，编
译→烧录运
行



扩展_TouchGFX应用例1（续）

124

- 基于已有环境增加GUI应用
 - 在打开的TouchGFX Designer中，增加GUI应用
 - 添加带标题的按钮
 - 设置按钮位置
 - 设置标题





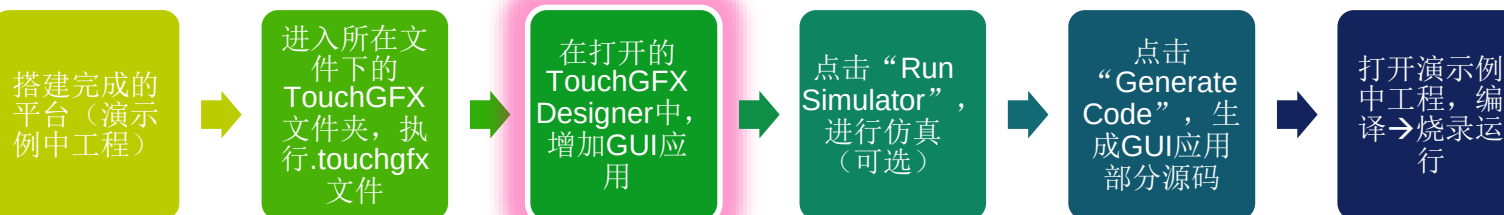
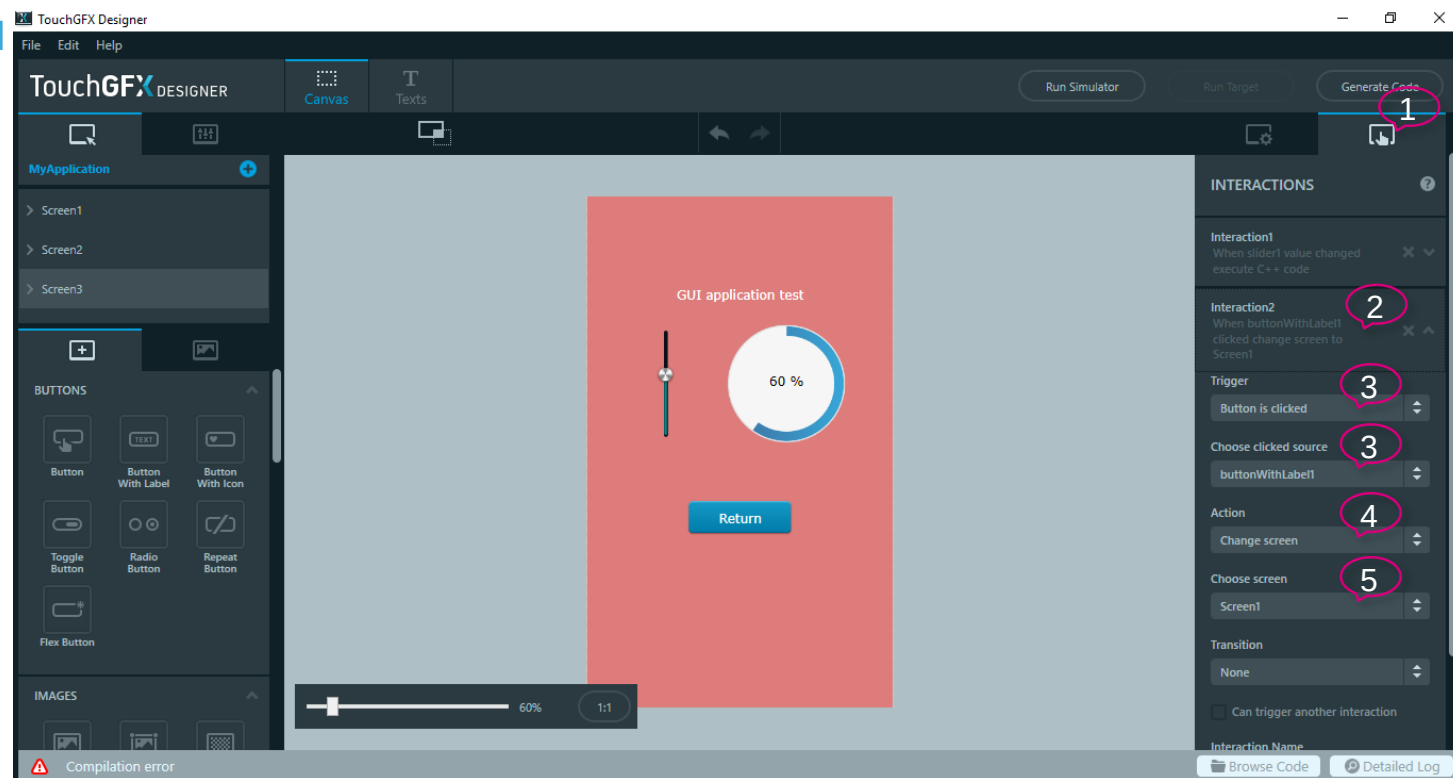
扩展_TouchGFX应用例1（续）

125

• 基于已有环境增加GUI应用

• 在打开的TouchGFX Designer中，增加GUI应用

- 添加交互
- 设置由按钮触发
- 关联触发源为Return按钮
- 动作关联为“Change screen”
- 设置按钮动作，跳转到Screen1.

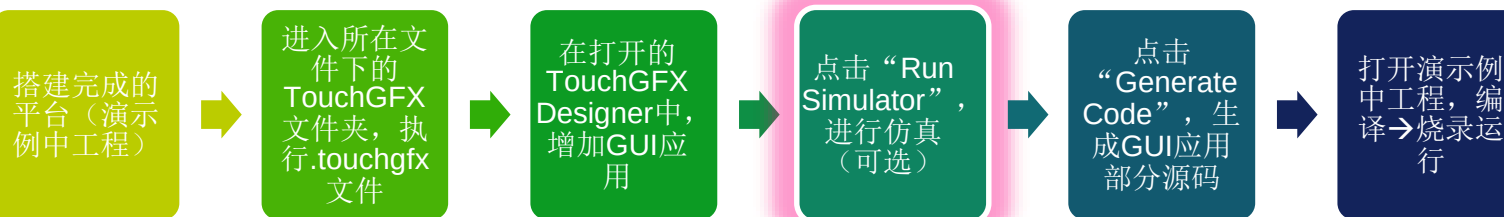
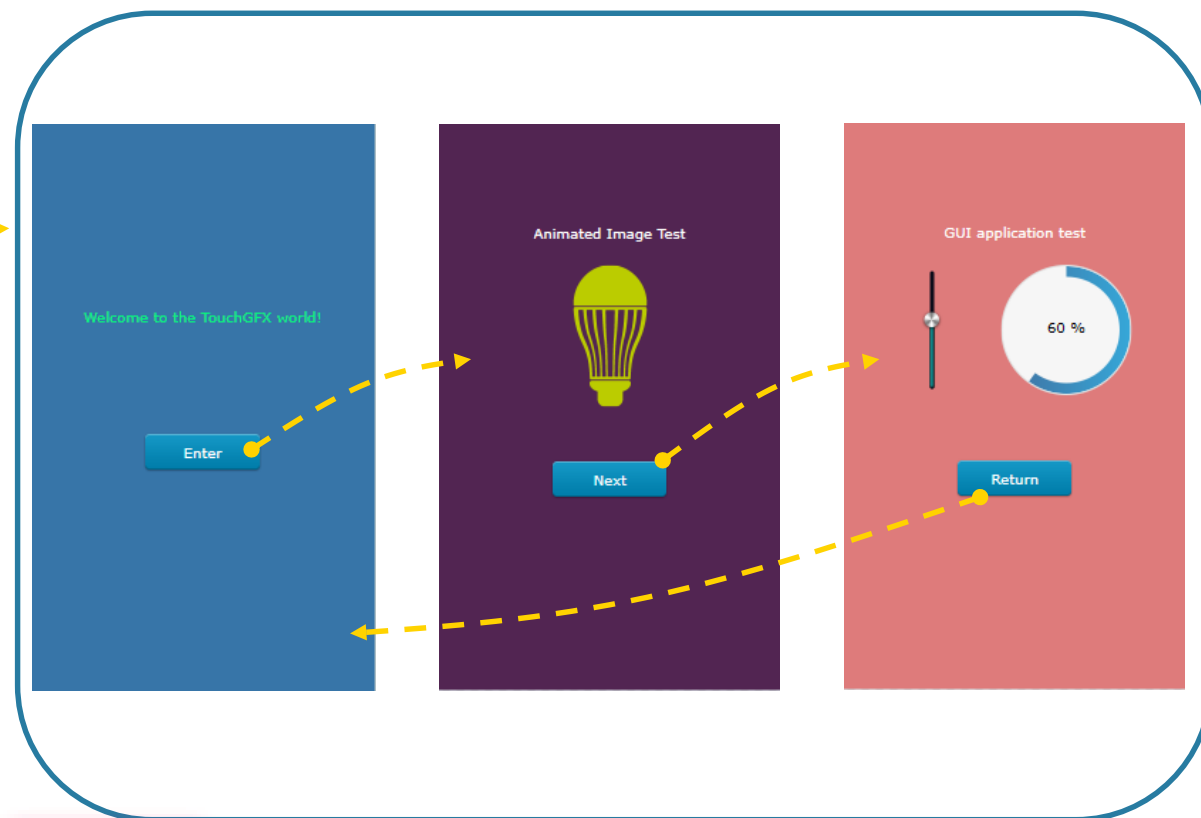
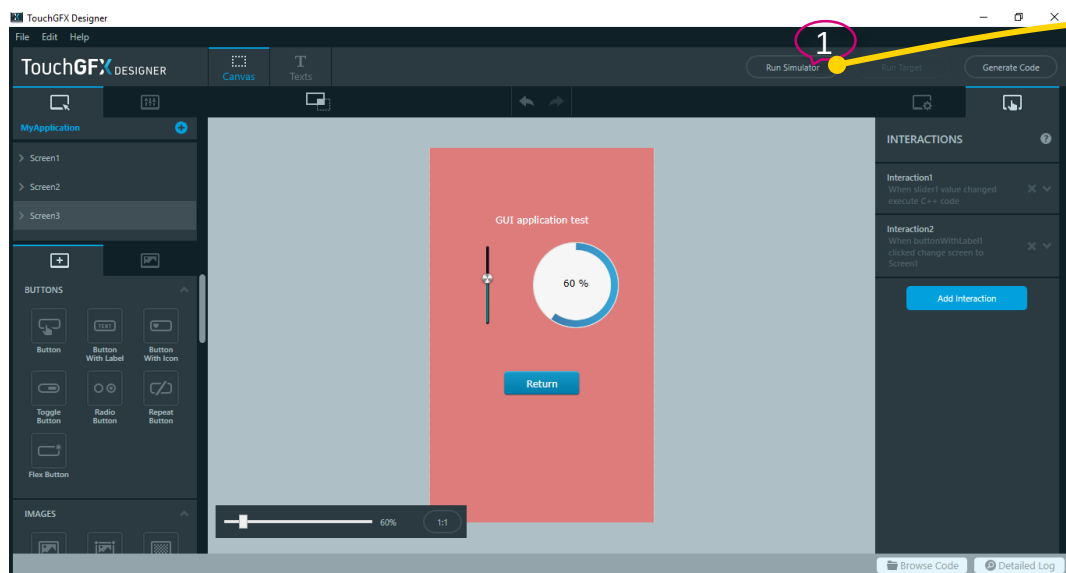




扩展_TouchGFX应用例1（续）

126

- 基于已有环境增加GUI应用
 - 在生成GUI应用源码时，可以通过点击“Run Simulator”，运行仿真界面，查看设计效果。

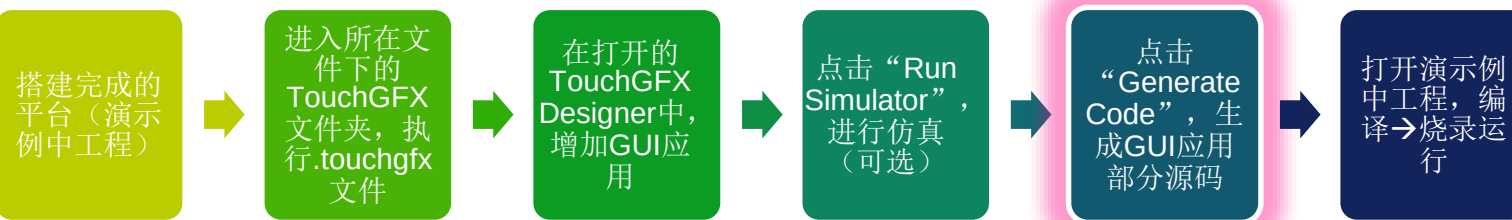
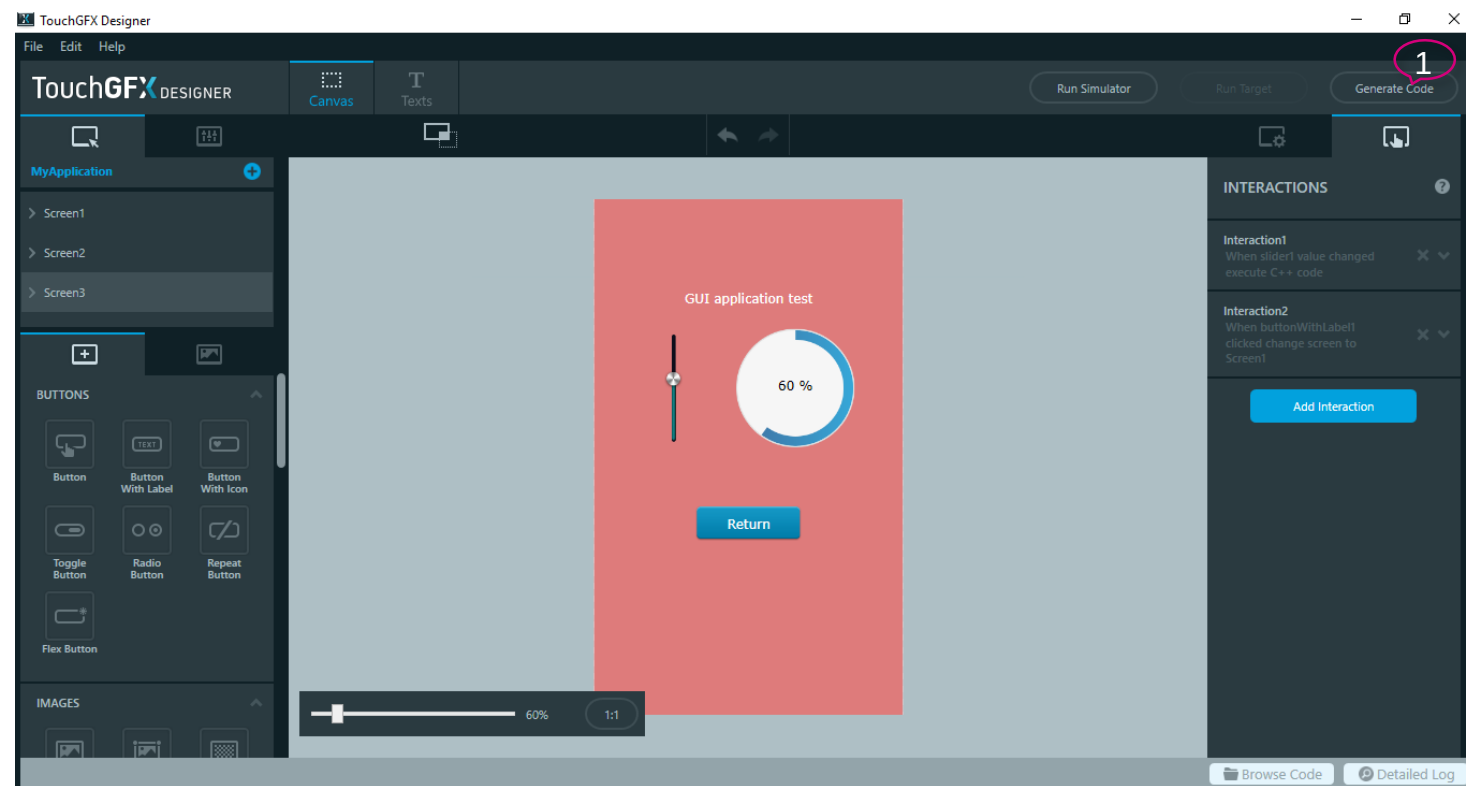




扩展_TouchGFX应用例1（续）

127

- 基于已有环境增加GUI应用
 - 点击“Generate Code”，生成GUI应用部分源码
 - 生成完成时，左下角提示“Code Generation Complete”

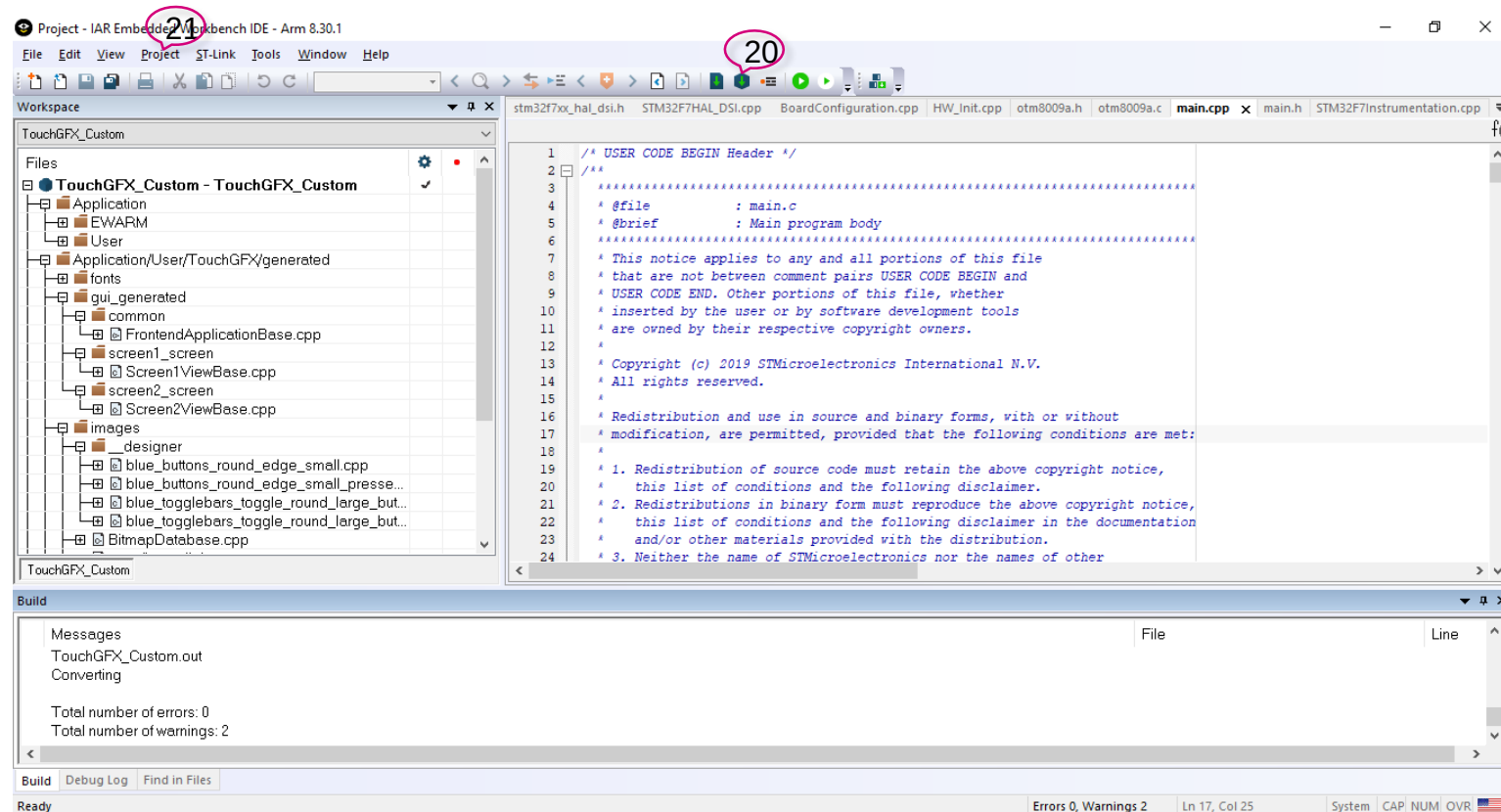




扩展_TouchGFX应用例1（续）

128

- 基于已有环境增加GUI应用
 - 打开演示例中工程，编译工程
 - 下载运行。通过点击菜单栏 **Project\Download\Download active application** 完成。在下载前，需要确保演示板已经与PC建立连接。



搭建完成的
平台（演示
例中工程）



进入所在文
件下的
TouchGFX
文件夹，执行
.touchgfx
文件



在打开的
TouchGFX
Designer中，
增加GUI应
用



点击“Run
Simulator”，
进行仿真
（可选）



点击
“Generate
Code”，生
成GUI应用
部分源码



打开演示例
中工程，编
译→烧录运
行



资源&参考

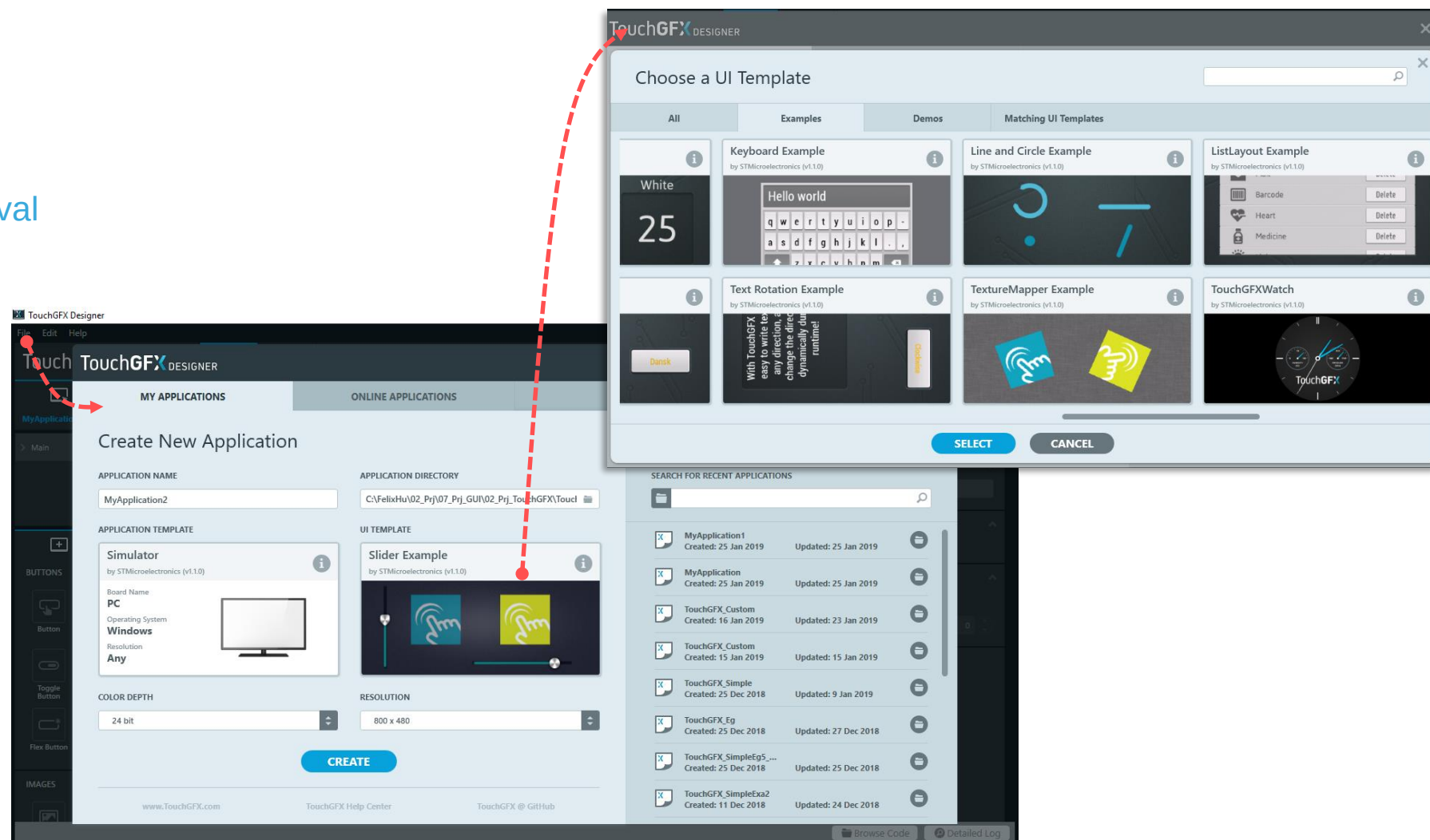


• 例程

- TouchGFX Designer
 - Demo
 - 例程
- Touchgfx-release-4.5.0-eval
 - Demo
 - 例程
 - 模板

• 仿真

- TouchGFX Designer
- MSVS
 - MSVS工程文件，位于工程文件夹的如下路径：
\\TouchGFX\\simulator\\msvs





- 文档

- 外扩存储器

- FSMC FMC_v1.1.pptx



- DMA2D

- [AN4943 Using the Chrom-ART Accelerator™ to refresh an LCD-TFT display on STM32L496xx L4A6xx L4Rxx L4Sxx microcontrollers](#)

- DSI

- [AN4860 DSI Host on STM32F469 F479, STM32F7x8 F7x9 and STM32L4R9 L4S9 MCUs_Rev 2](#)
 - [DSI Host（链接）](#)

- LTDC

- [AN4861 LCD-TFT display controller \(LTDC\) on STM32 MCUs](#)

- 视频

- GUI

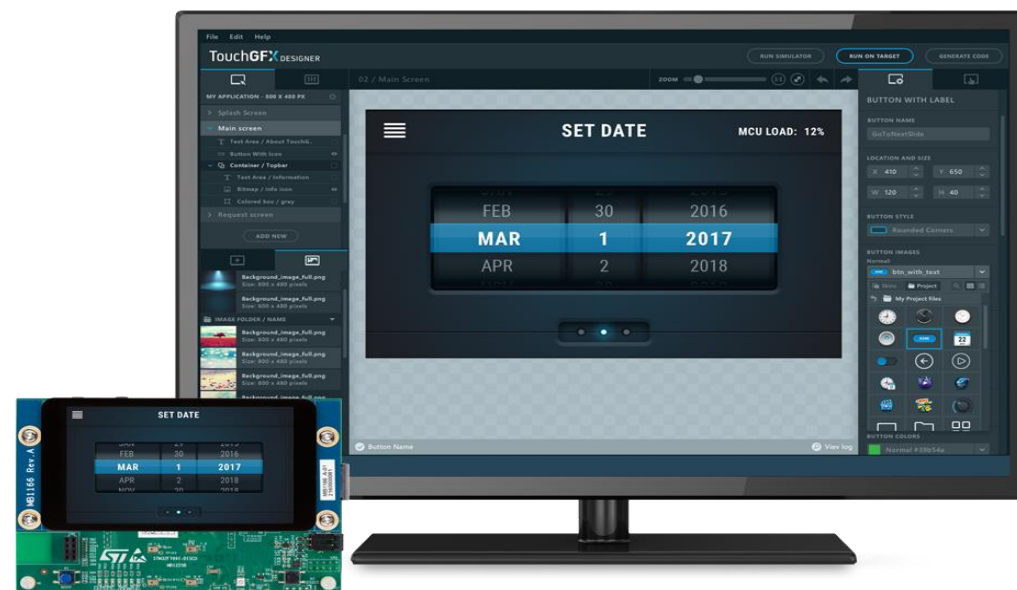
- Graphics with STM32 - 1 Introduction <https://youtu.be/k4bARbQ9zgA>
 - Graphics with STM32 - 2 Graphics basics <https://youtu.be/khh2-voq3Vo>
 - Graphics with STM32 - 3 Display Interfaces <https://youtu.be/hnzKhK5Vvao>
 - Graphics with STM32 - 4 STM32 Ecosystem for Embedded GUI <https://youtu.be/tCwbykWLMaI>
 - Graphics with STM32 - 5 STemWin Lab - SW4STM32 <https://youtu.be/EbS2rWvY7dg>
 - Graphics with STM32 - 6 Conclusion <https://youtu.be/A6HP15KDSIs>



参考设计和全球支持

更多链接

- [STM32 Graphic User Interface](#)
- [STM32 Graphics solutions Q&A](#)
- [TouchGFX knowledge base](#)
- [Graphics with STemWin MOOC](#)
- [TouchGFX Recorded Webinars](#)
- [ST on line support](#)
- [TouchGFX YouTube Channel](#)
- www.TouchGFX.com



- 在STM32开发板上一键运行示例或参考设计
- 在几分钟内设计自己的UI原型，并运行在STM32开发板上



For more information welcome to visit our website:

www.st.com/stm32

www.stmcu.com.cn

www.stmcu.org

