TouchGFX 图形应用在亮屏时的低功耗实现

关键字：TouchGFX， Low Power

1. 图形应用和低功耗

随着智能设备的普及，越来越多的设备会用到图形界面，而在 STM32 MCU 上使用 TouchGFX，使得图形设计变得非常简单。其中智能手表、智能手环等产品对功耗要求比较高，这就需要在图形应用同时，结合芯片的低功耗模式来优化能耗。

在图形应用中使用低功耗，一般分为两种场景，一种是在关闭屏幕时，MCU 进入 STOP 模式，能极大的降低 MCU 功耗；另一种是在屏幕亮着的状态，MCU 进入 SLEEP 模式，也能降低 MCU 功耗。而在 STM32L4+芯片上，LPSLEEP 模式相比 SLEEP 模式功耗更低，本文将在 STM32L4R9 芯片上，结合 TouchGFX 和 LPSLEEP 来介绍如何在亮屏状态下实现 MCU 低功耗。

2. 低功耗实现原理介绍

在使用 TouchGFX 做图形应用时，会使用到 FreeRTOS 操作系统，使用 SysTick 定时产生 1ms 的中断来作为系统 tick。在所有应用任务空闲时，系统会调度 Idle 任务。此时我们能让 MCU 进入低功耗，从而达到降低 MCU 功耗的目的。

2.1. Tickless Idle

FreeRTOS 提供了 Tickless Idle 配置，在 FreeRTOSConfig.h 文件中，通过配置 configUSE_TICKLESS_IDLE 为不同的值来配置 FreeRTOS 是否在 Idle 任务里进入低功耗，默认为 0 表示不进入低功耗。本文中将配置为 2，自定义 Idle 任务中低功耗实现。

Table 1 Tickless Idle 配置

configUSE_TICKLESS_IDLE 配置	Idle 任务中低功耗实现方式
0	不使用低功耗
1	使用 FreeRTOS 默认的实现
2	用户自定义实现

FreeRTOS 中进入低功耗具体是由 vPortSuppressTicksAndSleep 函数实现。配置 configUSE_TICKLESS_IDLE 为 2 时，还需要重写 vPortSetupTimerInterrupt 函数。

2.2. TouchGFX 渲染与进 LPSLEEP 限制

根据参考手册描述，在进入 LPSLEEP 时要先切换到 LPRUN 模式，系统时钟要降低到 2MHz，因此在进入、退出 LPSLEEP 时会有时钟的切换过程。

在屏幕亮着的状态时，用户可能随时触摸屏幕，或者系统信息可能更新，因此屏幕显示的内容可能随时变化，如果此时 MCU 进入 LPSLEEP 状态，会导致系统响应不及时。另外，在屏幕内容更新时，L4R9 通过 DSI 向屏幕更新显示数据，此过程由硬件完成，在此过程中，软件也有可能会进入 LPSLEEP，这样系统时钟发生变化，会导致 DSI 更新显示数据异常，产生屏幕显示花屏的问题。

为了避免以上问题，在进入 LPSLEEP 前会检查当前状态，如果用户未操作屏幕，TouchGFX 已经完成渲染，DSI 刷新过程也完成，则允许系统进入 LPSLEEP 状态。这样既保证了系统正常工作，又达到了降低功耗的目的。

3. 实现过程

重写 vPortSetupTimerInterrupt 函数，主要是设置与休眠时间补偿相关的变量，初始化 systick 寄存器：

```
void vPortSetupTimerInterrupt( void )
{
  /* Calculate the constants required to configure the tick interrupt. */
  #if( configUSE_TICKLESS_IDLE == 2 )
  {
    ulTimerCountsForOneTick = ( configSYSTICK_CLOCK_HZ / configTICK_RATE_HZ );
    xMaximumPossibleSuppressedTicks = portMAX_24_BIT_NUMBER / ulTimerCountsForOneTick;
    ulStoppedTimerCompensation = portMISSED_COUNTS_FACTOR / ( configCPU_CLOCK_HZ /
        configSYSTICK_CLOCK_HZ );
  }
  #endif /* configUSE_TICKLESS_IDLE */

  /* Stop and clear the SysTick. */
  portNVIC_SYSTICK_CTRL_REG = 0UL;
  portNVIC_SYSTICK_CURRENT_VALUE_REG = 0UL;

  /* Configure SysTick to interrupt at the requested rate. */
  portNVIC_SYSTICK_LOAD_REG = ( configSYSTICK_CLOCK_HZ / configTICK_RATE_HZ ) - 1UL;
  portNVIC_SYSTICK_CTRL_REG = ( portNVIC_SYSTICK_CLK_BIT | portNVIC_SYSTICK_INT_BIT |
      portNVIC_SYSTICK_ENABLE_BIT );
}
```

重写 vPortSuppressTicksAndSleep 函数。在进入 LPSLEEP 前，会停止 systick，在休眠完成后，需要补偿系统 tick 值，示例中使用 LPTIM 计时来补偿系统 tick 值。在进入 LPSLEEP 前先进行状态检查，只要有以下情况，都不进入低功耗模式：

a) 触摸事件产生：nosleep_flg
b) DMA2D busy 状态：isDMAbusy_a()
c) DSI refreshing 状态：displayRefreshing
d) 渲染完成，请求 DSI 刷新：refreshRequested

```
void vPortSuppressTicksAndSleep( TickType_t xExpectedIdleTime )
{
  uint32_t xMaximumPossibleSuppressedTicks,ulReloadValue, ulCompleteTickPeriods,ulCount;
  xMaximumPossibleSuppressedTicks=portMAX_16_BIT_NUMBER * configTICK_RATE_HZ / lptim_CLOCK_HZ;

  if(nosleep_flg || isDMAbusy_a() || displayRefreshing || refreshRequested) {
    if(!isDMAbusy_a() && !displayRefreshing && !refreshRequested) {
      if(osKernelSysTick() > nosleep_flg_t0 + nosleep_time)
        nosleep_flg = 0;
    }
    return;
  }

  if( xExpectedIdleTime > xMaximumPossibleSuppressedTicks ) {
    xExpectedIdleTime = xMaximumPossibleSuppressedTicks;
  }

  portNVIC_SYSTICK_CTRL_REG &= ~portNVIC_SYSTICK_ENABLE_BIT;
  HAL_SuspendTick();

  ulReloadValue = (lptim_CLOCK_HZ * (xExpectedIdleTime - 1UL)/configTICK_RATE_HZ);

  __DSB();
  __ISB();
  LPTIM_ReloadMatch_flag=pdFALSE;

  if( eTaskConfirmSleepModeStatus() == eAbortSleep ) {
     portNVIC_SYSTICK_CTRL_REG |= portNVIC_SYSTICK_ENABLE_BIT;
     portNVIC_SYSTICK_LOAD_REG = configCPU_CLOCK_HZ/configTICK_RATE_HZ - 1UL;
     HAL_ResumeTick();
  } else {
    MX_LPTIM1_Init(LPTIM_PRESCALER_DIV2, ulReloadValue);

    if( xExpectedIdleTime > 0 ) {
      __DSB();
      enter_lpsleep();
      __ISB();
    }

    ulCount = HAL_LPTIM_ReadCounter(&Lptim1);
    HAL_LPTIM_TimeOut_Stop_IT(&Lptim1);

    HAL_ResumeTick();

    if( LPTIM_ReloadMatch_flag != pdFALSE ) {
      ulCompleteTickPeriods = xExpectedIdleTime - 1UL;
    } else {
      ulCompleteTickPeriods = (ulCount * configTICK_RATE_HZ)/ lptim_CLOCK_HZ;
    }

    portENTER_CRITICAL();
    uwTick += ulCompleteTickPeriods;
    vTaskStepTick( ulCompleteTickPeriods );
    portNVIC_SYSTICK_CTRL_REG |= portNVIC_SYSTICK_ENABLE_BIT;
    portNVIC_SYSTICK_LOAD_REG = configCPU_CLOCK_HZ/configTICK_RATE_HZ - 1UL;
    portEXIT_CRITICAL();
  }
}
```

TouchGFX 渲染过程包括两部分，CPU 渲染和 DMA2D 渲染。在 CPU 渲染过程中，TouchGFX 任务处于执行状态，系统不会主动调度 Idle 任务。而在 DMA2D 渲染过 TouchGFX 会让出

CPU，此时可能会切换到 Idle 任务执行，因此需要对 DMA2D 是否正在渲染进行判断。对 DMA2D 是否在渲染的判断如下：

```
extern "C" uint8_t isDMAbusy_a()
{
    return ((TouchGFXHAL*)(touchgfx::HAL::getInstance()))->isDMAbusy();
}
uint8_t TouchGFXHAL::isDMAbusy()
{
    bool dma_qe = dma.isDmaQueueEmpty();
    bool dma_run = dma.isDMARunning();
    return (uint8_t)(!dma_qe || dma_run);
}
```

4. 小结

在 TouchGFX 图形应用中，在屏幕亮着的情况下如果需要优化功耗，可考虑结合 FreeRTOS tickless 模式，使用 MCU 的低功耗模式达到节能省电的目的。

文档中所用到的工具及版本

TouchGFX 4.17
IAR 8.50.9
STM32L4R9-DISCO 板

LAT 中的附件


main.c


LPSLEEP_config.c


lptim_update_tick.c


TouchGFXHAL.cpp

版本历史

日期	版本	变更
2021 年 10 月 29 日	1.0	首版发布