AN2585

应用笔记

STM32F101xx和STM32F103xx内核和系统外设应用示例

介绍

STM32F10xxx 以ARM公司为高级微控制器而设计的最新的Cortex™-M3为内核。它的特定的Thumb®-2 指令集使用16位代码密度，表现出32位的性能。

STM32F10xxx 有三个低功耗模式，它们使用嵌入的 8 MHz RC振荡器而具有快速启动能力。STM32F10xxx 同样内嵌一个实时时钟，该实时时钟参考一个内部的32 KHz RC 或一个外部石英振荡器。

另外STM32F10xxx 拥有V_{BAT} 功能，使它能够为移动应用和超低功耗应用以电池为能源运行。

安全性也是STM32F10xxx 的关键特性，它有着内嵌的复位电路，双看门狗架构(包含一个有自己的时钟源的独立的看门狗)，一个备用时钟以防主振荡器故障，和防篡改及备份寄存器的功能。

这份应用笔记是为了提供上面列举STM32F10xxx的不同特性的应用示例。Cortex-M3 和许多外设被编址：NVIC，SysTick，DMA，RCC，EXTI，PWR，BKP，RTC，Flash 存储器，IWDG和WWDG. 这份文档，它相关的固件，和其他类似的应用笔记是和STM32F10xxx 固件库相配套的。这些都可以从ST微电子的网站下载：www.st.com.

1 Cortex™-M3 内核

这一部分提供了关于怎样使用Cortex-M3™ 内核的一些特性的实践信息。

1.1 如何用Cortex-M3 位绑定（bit-band）存储访问

1.1.1 综述

The Cortex-M3 内存映射包括两个bit-band区域。这些区域把别名区中每个字映射为bit-banding区的1位。在别名区中写一个字跟在bit-banding存储区对目标 位进行一次读/修改/写操作有同样的效果。

在STM32F 10XXX中，外设寄存器和SRAM 被映射到一个bit-band区。这样就可以进行原子的bit-band读和写操作。

下面这个公式 显示了如何 通过 bit-band区的相应位获得别名区的每个字 ：bit_word_addr = bit_band_base +（byte_offset * 32）+（bit_number * 4）

其中：

bit_word_addr 是别名区中 映射到 目标位 的 字 的地址。

bit_band_base 是别名区的起始地址

byte_offset 是 包含目标位 的bit-band区 的字节数

bit_number 是 目标位 的位置(0 – 31)

下面的示例说明了如何将SRAM中的bit-band区ox20000300 的字节的第2位映射到别名区。

0x2200 6008 = 0x2200 0000 + (0x300 × 32) + (2 × 4)

向地址0x2200 6008 进行 写 操作 跟 向SRAM的别名区 地址为0x2000 0300 的字节的bit 2 进行读/修改/写操作有同样的效果。

读地址0x2200 6008 返回在SRAM 地址0x2000 0300 的这一字节的第2位的值(0x01: 位置位；0x00: 位复位).

1.1.2 **固件描述**

下面的示例显示了如何用Cortex-M3 bit-band 访问 对SRAM上的一变量进行原子的read-modify-write操作。

相关程序声明了3个 宏来操作这个可变位，使用bit-band访问重置，设置，读一个特定位

```
#define RAM_BASE 0x20000000
#define RAM_BB_BASE 0x22000000
#define Var_ResetBit_BB(VarAddr, BitNumber) \
(*(vu32 *) (RAM_BB_BASE | ((VarAddr - RAM_BASE) < < 5) | RAM_BB_BASE |
((BitNumber) < < 2)) = 0)
```

```
#define Var_SetBit_BB(VarAddr, BitNumber) \
(*(vu32 *) (RAM_BB_BASE | ((VarAddr - RAM_BASE) << 5) | RAM_BB_BASE |
((BitNumber) << 2)) = 1)
#define Var_GetBit_BB(VarAddr, BitNumber) \
(*(vu32 *) (RAM_BB_BASE | ((VarAddr - RAM_BASE) << 5) | RAM_BB_BASE |
((BitNumber) << 2)))
```

作为一个示例，我们声明一个变量，它的值通过上面定义的宏 改变了。

这个固件在STM32F 10XXX固件库中被作为 Cortex-M3 示例1，可从ST微电子公司微控制器的网站获取。

1.2 如何 修正 Cortex-M3 的特权线程模式 和 使用的堆栈

1.2.1 概述

这个示例 显示了如何 修正 Cortex-M3 的线程模式的特权访问和堆栈。

1.2.2 固件描述

Cortex-M3 线程模式 在复位时进入，在异常返回时也可以进入。

相关程序被用来：

将线程模式堆栈 由主栈转换到进程堆栈。

将线程模式 由特权转换到非特权。

将转线程模式 由非特权转回到特权 。

为了监视线程模式中堆栈的使用和代码的特权/非特权访问级别，在程序中一个变量集是可用的。也可以使用调试器的 ‘ Cortex寄存器’ 窗口。

此固件 在STM32F 10XXX固件库中被作为 Cortex-M3 示例2，可从ST微电子公司微控制器的网站获取。

2 NVIC 应用示例

这一部分提供了STM32F10xxx NVIC外设的应用实践程序示例

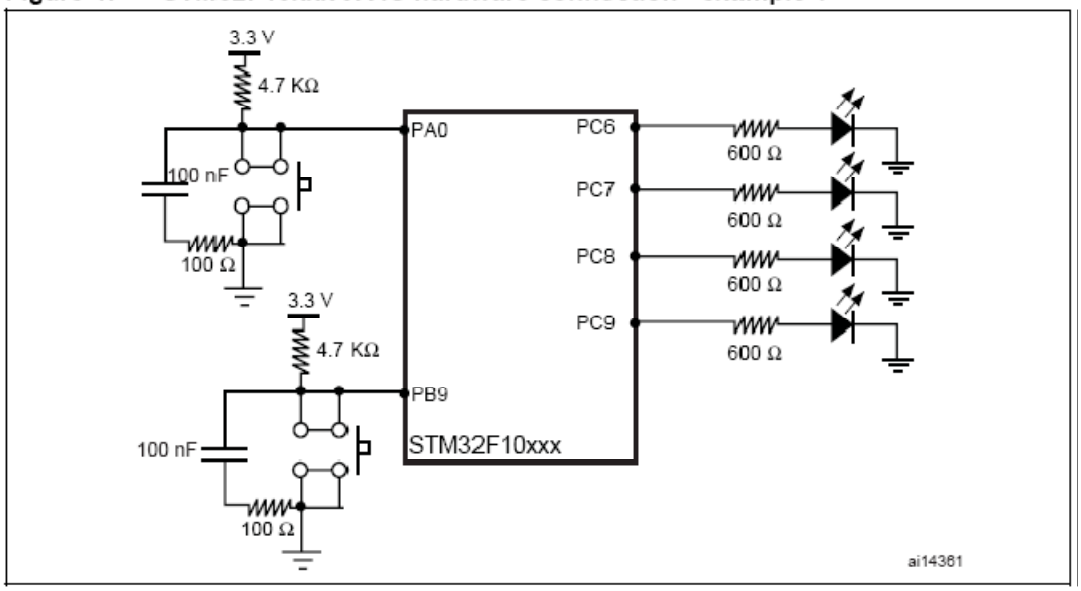
2.1 STM32F10xxx NVIC：抢占和子优先级处理

介绍如何使用NVIC（嵌套向量中断控制器）固件库来演示Cortex-M3优先级机制（抢占优先级和子优先级）。给出的一个示例使用了NVIC驱动的一些功能 ：使能，设置优先级，获得活动IRQ状态，清除IRQ挂起位等。

2.1.1 硬件描述

下图展示了示例程序使用的硬件连接

Figure 1. STM32F10xxx NVIC hardware connection - example 1



2.1.2 固件描述

提供的固件库包括了NVIC驱动，该驱动通过一系列的函数来支持所有的Cortex-M3异常和IRQ通道处理。给出的示例使用了大部分的功能。在这个示例程序中，使用了2根EXTI（外部中断） 线（Line0&Line9）和SysTick中断处理。在EXTI线的每一个下降沿产生一个中断。

这些中断配置如下：

EXTI Line0：抢占优先级=抢占优先级值

子优先级=0

EXTI Line9：抢占优先级=0

子优先级=1

SysTickHandler：抢占优先级=！抢占优先级值

子优先级=0

首先，抢占优先级值等于0，EXTI Line0拥有的抢占优先级比SysTick Handler高。

在EXTI Line9中断处理程序，将EXTI Line0和SysTick抢占优先级翻转（优先级1的变0，优先级0

的变1）。在EXTI Line0中断处理程序，设置SysTick中断挂起位。仅当SysTick的优先级比EXTI Line0高的时候SysTick ISR抢占EXTI Line0 ISR。

1 EXTI Line9第一次发生中断时，SysTick的抢占优先级比EXTI Line0高，所以当EXTI Line0发生中断时，SysTick ISR执行，PreemptionOccured变量为TRUE，同时连接到PC6，PC7，PC8 PC9的led灯翻转。

2 当下一次EXTI Line9发生时，由于在前面一次EXTI Line9的中断处理程序将EXTI Line0和SysTick的优先级翻转，现在SysTick抢占优先级比EXTI Line0低，当EXTI Line0中断发生时，PreemptionOccured变量为false，同时连接到PC6，PC7，PC8 PC9的led停止翻转。

以上的1和2在一个无限循环中重复进行。

可参见NVIC示例程序一（STMicroelectronics 微控制器的网站）。

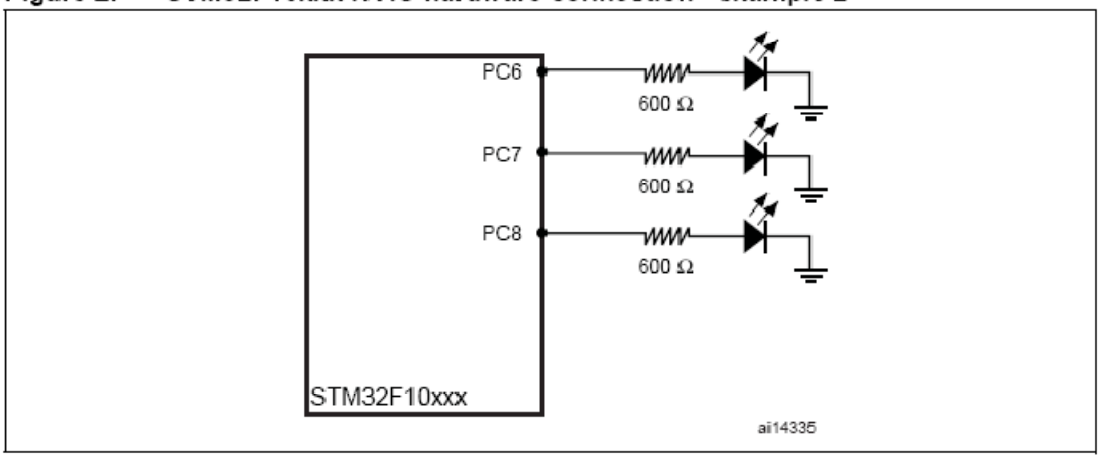
2.2 STM32F10xxxs NVIC：IRQ通道中断过程

介绍了如何使用NVIC固件库来实现几个不同优先级的IRQ通道。

2.2.1 硬件描述

示例程序硬件连接如下

Figure 2. STM32F10xxx NVIC hardware connection - example 2



2.2.2 固件描述

提供的固件库包括了NVIC驱动 ,它通过一系列的函数来支持所有的Cortex-M3异常和IRQ通道处理。给出的示例使用了大部分的功能。

在这个示例中，使用了三个IRQ通道：TIM2，TIM3和TIM4中断。这三个定时器配置为在每次定时器更新事件发生时产生一个中断。IRQ通道的IRQ抢占优先级是递增的，即TIM2抢占优先级为0 TIM4抢占优先级为2。

在每个中断处理程序：

TIM2的中断处理程序每1s一次将连接到PC6的led灯翻转（亮变暗或暗变亮）

TIM3的中断处理程序每2s一次将连接到PC7的led灯翻转（亮变暗或暗变亮）

TIM4的中断处理程序每3s一次将连接到PC8的led灯翻转（亮变暗或暗变亮）

可参见NVIC示例程序2（STMicroelectronics 微控制器的网站）。

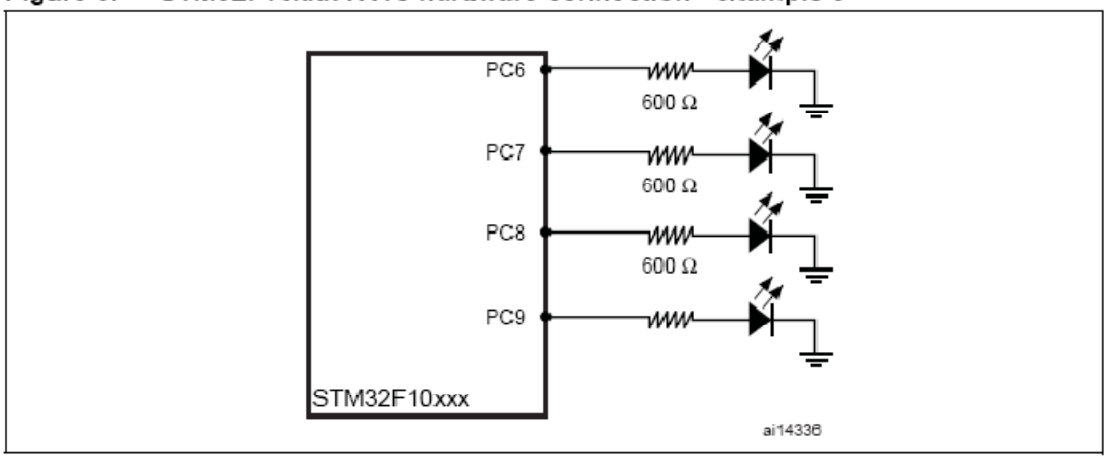
2.3 STM32F10XXX NVIC：系统中断处理

介绍了如何使用NVIC固件库来实现几个不同优先级的系统中断处理

2.3.1 硬件描述

硬件连接如下

Figure 3. STM32F10xxx NVIC hardware connection - example 3



2.3.2 固件描述

提供的固件库包括了NVIC驱动 ,它通过一系列的函数来支持所有的Cortex-M3异常和IRQ通道处理。给出的示例使用了大部分的功能。

在这个示例中，使用了四个系统处理程序：NMI，PSV，SVCall和SysTick处理程序。这些中断处理程序从SysTick处理程序中级联调用。

中断处理优先级配置如下：

NMI抢占优先级为-1（固定优先级）

PSV为0

SVCall 为1

SysTick为2

SysTick定时器配置为每秒产生一个中断。

在SysTick中断处理程序中，连接到PC6的灯每秒点亮一次并执行SVC指令。它激活SVCall中断处理程序并抢占当前指令流。在SVCall中断处理程序中，连接到PC7的LED灯翻转（亮变暗或暗变亮），软件置位PSV中断挂起位。由于PSV拥有高的优先级，它抢占SVCall中断处理程序并将连接到PC8的LED翻转（亮变暗或暗变亮），在PSV中断处理程序中， 又将NMI中断挂起位置位，激活NMI中断，导致连接到PC9的LED翻转（亮变暗或暗变亮）。

可参见NVIC示例3（STMicroelectronics 微控制器的网站）。

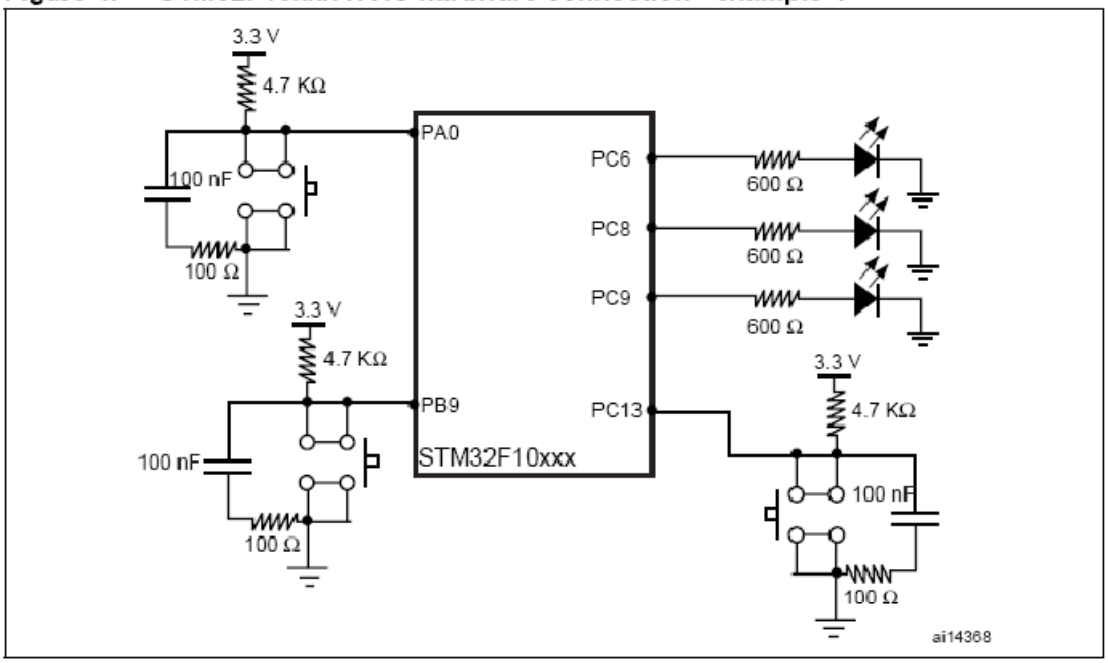
2.4 STM32F10xxx NVIC：WFE和WFI模型

介绍了如何使用NVIC固件库来演示Cortex-M3低功耗模式：WFE（等待事件）和WFI（等待中断）

2.4.1 硬件描述

下图是硬件连接

Figure 4. STM32F10xxx NVIC hardware connection - example 4



2.4.2 固件描述

给出的固件库包含了NVIC驱动，它通过一系列的函数支持所有的Cortex-M3异常和IRQ通道处理。给出的示例使用了大部分的功能。

在这个示例程序中，使用了三个EXTI Lines（Line0，Line9和Line13）。在每个下降沿产生一个中断或事件。EXTI Line0和Line13配置为中断模式而EXTI Line9配置为事件模式。用户必须通过在主代码中已定义的五条语句来选择执行哪个低功耗模式。

Wait For Interrupt（WFI）Sleep On Exit：在这种情况下，Cortex-M3在异常返回时自动进入WFI模式，并不需要WFI指令。如果异常拥有足够高的优先级，那么它将被执行，Cortex-M3返回到WFI模式。为了获得这些行为，可按如下步骤：

选择#WFISLEEPONEXIT定义的状态，而把其它的状态注释掉

运行示例程序

在EXTI Line13事件第一次发生时， Cortex-M3 SLEEPONEXIT位被置位。在异常返回后系统进入WFI模式。

EXTI Line0每次事件发生，连接到PC8的LED翻转（亮变暗或暗变亮）。

任何EXTI LINE0和Line13的下降沿都将执行EXTI中断。从ISR退出后，Cortex-M3将进入WFI模式。只要EXTI Line0拉低，Cortex-M3将退出WFI模式，连接到PC8的LED灯将点亮Cortex-M3将返回到WFI模式。以上动作在一个无限循环中重复。

Wait For Interupt（WFI）Sleep Now：在这个模式下WFI指令直接执行。为了获得这些行为，可按如下步骤：

选择#WFISLEEPNOW定义的状态，而把其它的状态注释掉

运行示例程序

在EXTI Line13的中断处理程序，连接到PC6的LED灯翻转，Cortex-M3进入WFI模式

使用连接到EXTI Line0的按键将Cortex-M3从中唤醒，这将连接到PC8的LED翻转。

以上行为在一个无限循环中重复出现。

Wait For Event（WFE）SEV ON PEND：在这个模式下，WFE指令之际执行，主执行代码由PRIMASK（main priority=0）来加速。步骤如下：

选择#WFESEVONPEND定义的状态而把其他的状态注释掉

运行示例程序

在EXTI Line13中断服务程序中 ,执行的优先级上升到0(一旦优先级提升 ,没有其他的中断执行），连接到PC6的LED灯翻转，Cortex-M3进入WFE模式

使用连接到EXTI Line0的按键把Cortex-M3唤醒。它不会将连接到PC8的LED翻转但会把系统从WFE（SEV ON PEND）唤醒。EXTI Line0中断从非挂起状态转换为挂起状态，因此系统会从WFE唤醒（EXTI Line0中断没有足够的优先级来运行，因为主执行优先级使用PRIMASK寄存器提升为0）

连接到EXTI Line9的按键配置为事件模式，它也能用来唤醒Cortex-M3。

如果#RTC-Alarm-WFEWakeUp定义的状态使能，RTC 警告将在6秒后唤醒Cortex-M3（如果在这段时间内EXTI Line0和EXTI Line9都没有用来唤醒Cortex-M3的话）。

Wait For Event（WFE）SEV ON EVENT：在这个模式下，WFE执行直接执行。步骤如下：

选择#WFESEVONEVENT定义的状态而把其他得状态注释掉

运行示例程序

在EXTI Line13中断处理程序，连接到PC6的LED翻转，Cortex-M3进入WFE模式

使用连接到EXTI Line0的按键把Cortex-M3唤醒，它将连接到PC8的LED翻转。

连接到EXTI Line9的按键配置为事件模式，它也能用来唤醒Cortex-M3。

如果#RTC-Alarm-WFEWakeUp定义的状态使能，RTC alarm将在6秒后唤醒Cortex-m3（如果在这段时间内EXTI Line0和EXTI Line9都没有用来唤醒Cortex-M3）。

这一系列动作将在一个无限循环中重复

连接到PC9的LED翻转表示Cortex-M3处于运行模式而非低功耗模式。

可参见NVIC示例程序4（STMicroelectronics 微控制器的网站）。

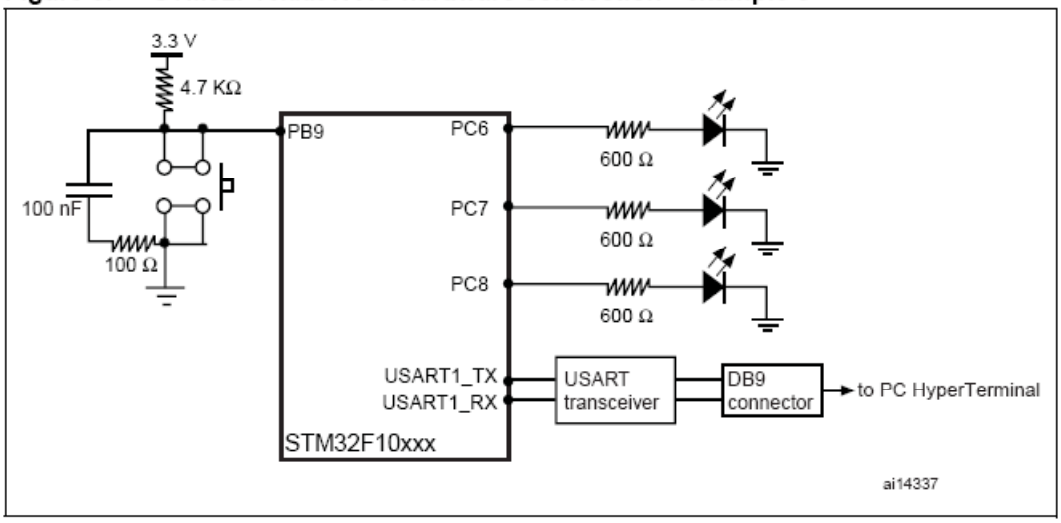
2.5 STM32F10xxxNVIC：WFI模式下的DMA

介绍了DMA传输使能下如何进入WFI模式，DMA传输结束中断把系统从这个模式下唤醒。

2.5.1 硬件描述

Figure 5显示了四个LED连接到STM32F10xxx，USART1信号使用RS232收发器连接到DB9连接器。然后再用一个空调制阴性/阴性RS232线将PC串口也连接到DB9连接器的另一端。

Figure 5. STM32F10xxx NVIC hardware connection - example 5



2.5.2 固件描述

提供的固件库包含能通过一套函数支持所有Cortex-M3异常和IRQ通道处理程序的NVIC驱动。

在相关的固件中，系统时钟为72MHz，DMA通道5配置为从USART1数据寄存器传输10块数据到预定义的缓冲区DST-Buffer，然后在传输完成后产生一个中断。

USART1从超级终端上接受数据

连接到PC6的LED以一定的频率翻转，这个频率依赖与系统时钟，它用来显示MCU是处于WFI还

是RUN模式。

EXTI Line9下降沿将导致Cortex-M3进入WFI模式，连接到PC6的LED停止翻转。为了将Cortex-M3从WFI模式中唤醒，从超级终端中发送一系列的数：0，1，2，3，4，5，6，7，8，9到USART1，这些数据通过DMA传输到预定义的缓冲区中。产生一个中断，然后系统从WFI模式退出。

连接到PC6的LED再次开始翻转，如果缓冲区的数据正确接收，则连接到PC7的LED翻转，否则连接到PC8的LED翻转。

可参见NVIC示例程序5（ STMicroelectronics 微控制器的网站）。

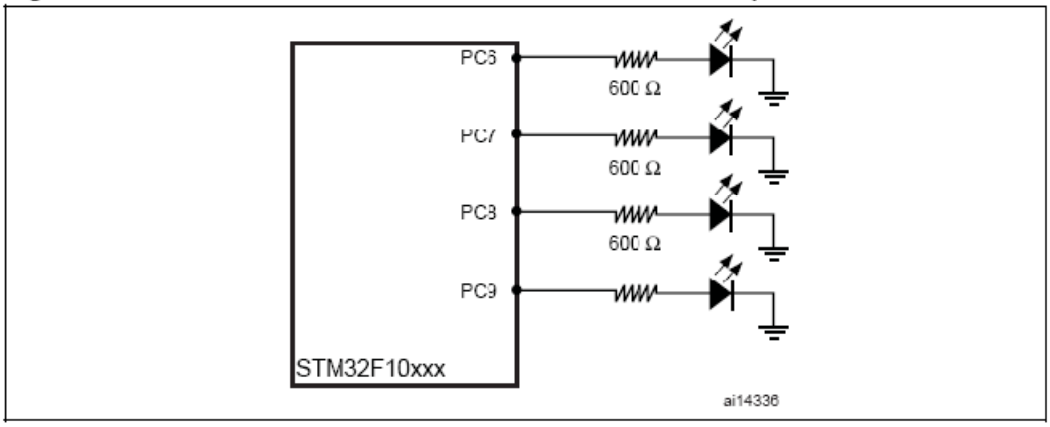
2.6 STM32F10xxx NVIC：用偏移实现中断向量表

介绍了如何使用NVIC固件库将Cortex-M3向量表重定位到一个特殊的地址，而不是缺省的地址。

2.6.1 硬件描述

下图为示例硬件连接

Figure 6. STM32F10xxx NVIC hardware connection - example 6



2.6.2 **固件描述**

给出的固件库介绍了如何使用NVIC固件库重定位向量表到一个特殊的地址。

这可以用来编译一个程序，该程序将通过Flash存储器的段0中预先编写好的应用程序装载入Flash存储器。

这样的应用程序能够在应用中编程(IAP、通过USART)或者是设备固件提升(DFU ,通过USB)。他们可以在ST的网站上下载。

相关的程序为基于SysTick计数到终点中断实现一个延时功能。定时翻转连接到PC6 ,PC7 ,PC8 ,PC9四个LED，定时时间有Delay函数定义。

当使用IAP加载程序时，向量表必须位于0X08002000地址

使用DFU加载程序时， 向量比必须位于0X08003000地址

可参见NVIC示例程序6（ STMicroelectronics 微控制器的网站）。

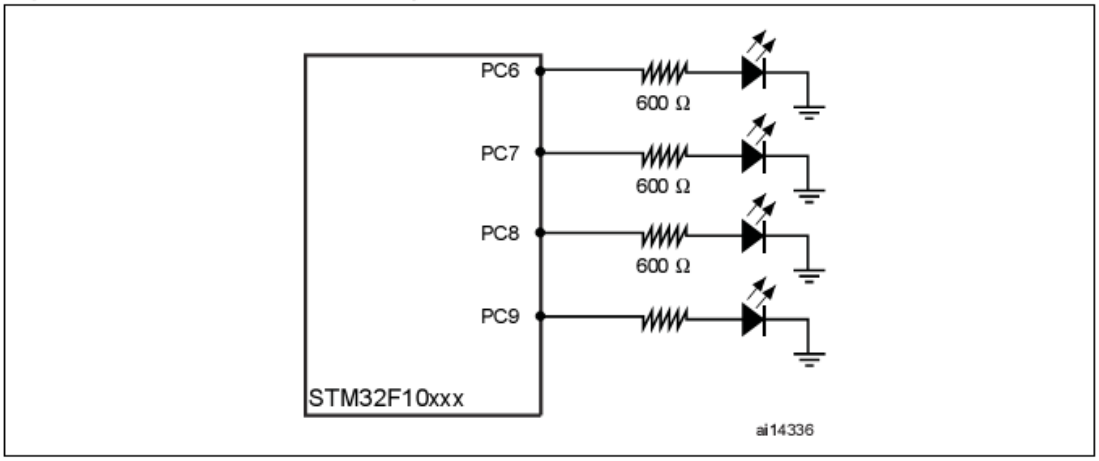
3 怎样使用STM32F10xxx SysTick

这部分描述了怎样配置Cortex-M3™ 系统时钟（SysTick）以生成一个时钟基准。将给出一个使用了SysTick大部分功能的示例。设置重装值，使能和禁止计数器，使能中断请求。

3.1 硬件描述

硬件电路如下

Figure 7. STM32F10xxx example LED connection



3.2 固件描述

提供的固件库中包含有SysTick驱动，他通过一系列的函数来支持所有的SysTick的特性。并提供了一个使用了大部分该功能的示例

这个示例展示了如何配置SysTick来产生一个1ms的定时基准。系统时钟为72MHz，SysTick的时

钟频率为AHB时钟除以8。

延时功能就是基于SysTick 计时结束事件。

四个LED连接到PC6 PC7 PC8 PC9 ，并且定时翻转（定时时间由延时函数定义）

可参见SysTick示例（STMicroelectronics 微控制器的网站）

4 DMA 应用示例

这一部分提供了STM32F10xxx DMA外设使用示例

4.1 STM32F10xxx 使用DMA从Flash到RAM数据传输

这一节介绍了如何使用DMA把数据缓冲区的内容从Flash内存中传到RAM中。

4.1.1 硬件描述

不需要硬件连接

4.1.2 固件描述

给出的固件库中包含有DMA驱动，它通过一系列的函数来支持所有的DMA功能。

DMA通道6配置用来将储在FlashMemory中的32位字数据传输到定义在RAM中的接受缓冲区。通过软件触发。DMA通道6的Memory-Memory传输使能。源端和目的端的地址增加也是使能的。通过设置DMA通道6的Channel Enable位来开始DMA传输。在传输结束后产生一个传输完成中断，在中断处理程序中，读取还需要传输的数据，直到剩下的数据个数为0。清除传输中断挂起位。比较源缓冲区和目的缓冲区的数据来检测是否所有的数据正确地传输。

参见DMA示例一（STMicroelectronics 微控制器的网站）。

4.1.3 总结

使用DMA从内存到内存传输数据可以较少代码的大小和执行时间。只需要使用一个DMA通道，在软件触发传输开始后，可以传输高达65536块数据。

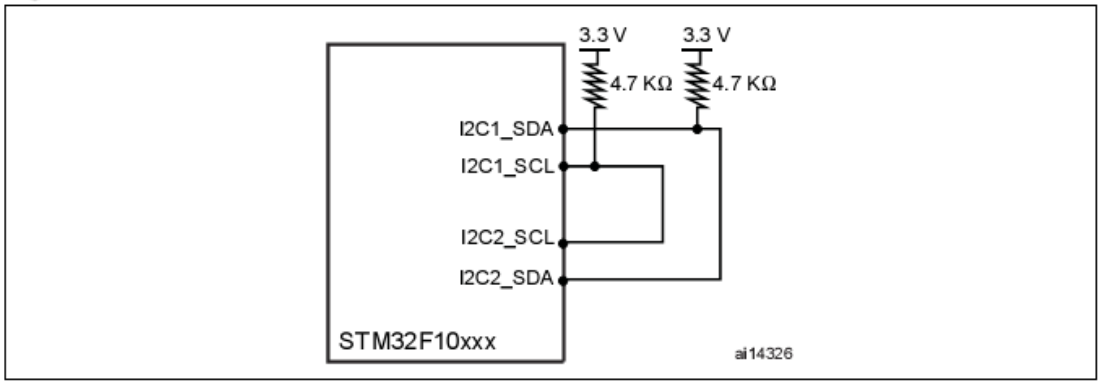
4.2 STM32F10xxx I2C-I2C使用DMA通讯

该节介绍如何使用DMA设置I2C-I2C数据通讯以实现传输和接收。

4.2.1 硬件描述

下图展示了STM32F10xxx I2C1和I2C2的典型连接。I2C1和I2C2的数据引脚（SDA）连接到一起，I2C1和I2C2的时钟（SCL）连接到一起，两根线都接上上拉电阻。

Figure 8. STM32F10xxx I^2C-I^2C communication



4.2.2 固件描述

提供的固件库中包含有DMA和I2C的驱动，它通过一系列的函数来支持所有的DMA功能和I2C通讯。

I2C1设置为主传输这而I2C2则为从接收者。DMA通道5配置为将从I2C2中接收的数据存储到I2C2Rx缓冲区（接收缓冲区）。通道6把I2C1Tx缓冲区（发送缓冲区）中的数据传输到I21DR寄存器以用来传输到I2C2。一旦开始后而且知道了从地址 ,I2C1和I2C2的DMA都被使能。一旦 I2C1_CR2 和 I2C2_CR2 寄存器I2C DMAEN置位, I2C1 Tx 缓冲区数据就开始通过DMA通道5传输 ,同时I2C2通过DMA通道6接收数据到I2C2 Rx缓冲区。然后接收缓冲区和发送缓冲区的数据比较来检测传输是否正确。

参见DMA示例二（STMicroelectronics 微控制器的网站）。

4.2.3 总结

在没有标识检测和DR寄存器访问的时候，在I2C通讯中使用DMA可以减少执行时间和代码大小。

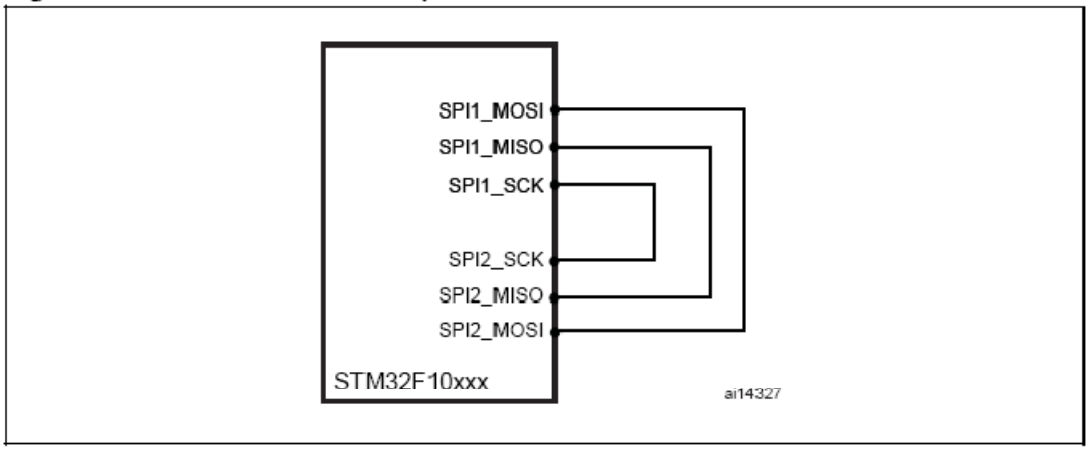
4.3 STM32F10xxx使用DMA进行全双工SPI-SPI通讯

该节介绍了如何使用DMA设置SPI-SPI的全双工通讯，通讯结尾DMA自动产生一个CRC检验。

4.3.1 硬件描述

下图为STM32F10xxxSPI1和SPI2的典型连接，SPI1和SPI2的数据输入MOSI连在一起，数据输出MISO连在一起，SCK连接到一起。并且由于NSS管理，NSS引脚保持断开。

Figure 2. STM32F10xxx full-duplex SPI-SPI communication



4.3.2 固件描述

提供的固件库包含有所有的DMA和SPI驱动，他通过一系列的函输来支持所有的DMA特性和SPI通讯。

NSS引脚由软件配置，来设置SPI1 作为主设备， SPI2作为从设备。DMA 通道2 和通道4 被配置用来存储由SPI1接收进入SPI1 接收缓冲区的数据，和由SPI2接收进入SPI2接收缓冲区。DMA 通道3被配置用来传送数据，从SPI1 发送缓冲区到SPI1_DR 寄存器，以传输到SPI2。DMA 通道5被配

置用来传送数据，从SPI2 发送缓冲区到SPI2_DR 寄存器，以传输到SPI1。

当在SPI1_CR2 和 SPI2_CR2 寄存器中两个SPI TxDMAEN 和RxDMAEN 被置位，DMA 通道3 和通道5开始分别将SPI1 和SPI2 发送缓冲区中的数据发送。 同时，在SPI1 和SPI2 接收到的数据分别被DMA 通道2和通道4传送到SPI1 和SPI2 接收缓冲区。此后系统检查被DMA通道接收和传送的所有数据。在SPI1 和SPI2 上最后收到的数据是SPI2 和SPI1分别发送的CRC校验值。发送和接收缓冲区被比较，以检查所有数据是否被正确传送。这个固件作为*DMA* 示例3 在STM32F10xxx 固件库中提供，可以从ST微电子公司微控制器网站下载。

4.3.3 **总结**

SPI通讯中使用DMA可以减少执行时间和代码大小 ,而其没有标识检测和没有任何DR寄存的访问。使用DMA时在数据传输结束后自动传输CRC。

4.4 STM32F10xxx中使用DMA进行外设 - 外设的数据传输

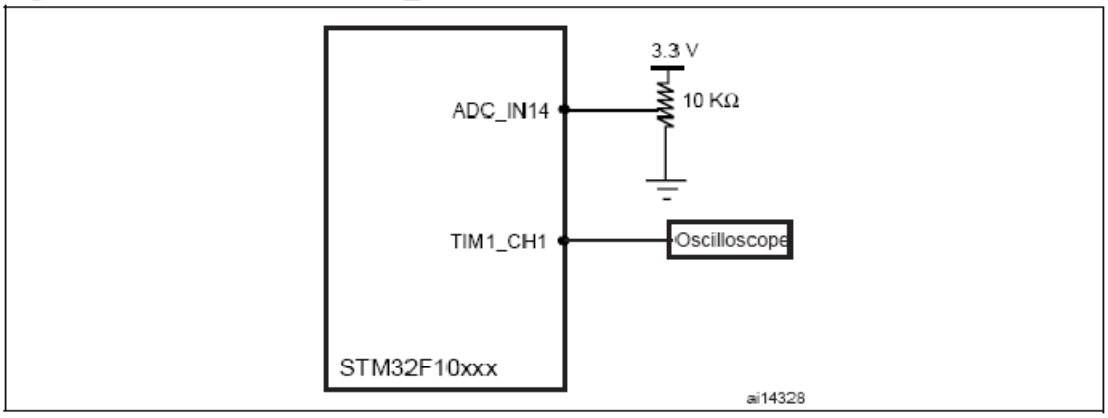
该节介绍如何使用DMA在两个外设间传输数据：数据从ADC_DR传输到TIM1_CCR1寄存器。

4.4.1 **硬件描述**

下图展示了分压计和ADC_IN14的典型连接。TIM通道输出引脚（TIM1_CH1）连接到示波器以

便显示ADC_IN14模拟输入信号的占空比。

Figure 3. STM32F10xxx ADC_IN14 connection



4.4.2 固件描述

给出的固件库中包含了DMA，ADC和TIM1驱动，这些驱动通过一系列的函数来支持所有的DMA，ADC，TIM1的所有功能。

ADC_IN10配置为连续变化，TIM1-CH1配置为产生PWM信号。DMA通道1用来将ADC_IN14产生的变化值从ADC1_DR循环的传输到TIM1_CCR1。这个有序的传输持续地将TIM1-CH1的占空比更新为ADC-IN14的变化值。

可参见DMA示例四（STMicroelectronics 微控制器的网站）。

4.4.3 总结

DMA可以用来从一个外设传输数据到另外一个外设去，只要两个设备都支持DMA传输。这样减少了代码的大小和软件的执行时间，而且使得用户可以容易的在外设间传输数据。DMA循环模式也允许在没有人为干预的情况下持续传输数据。

5 RCC 应用示例

这部分提供了关于STM32F10xxx的RCC外设的使用（复位和时钟控制）的实际应用示例

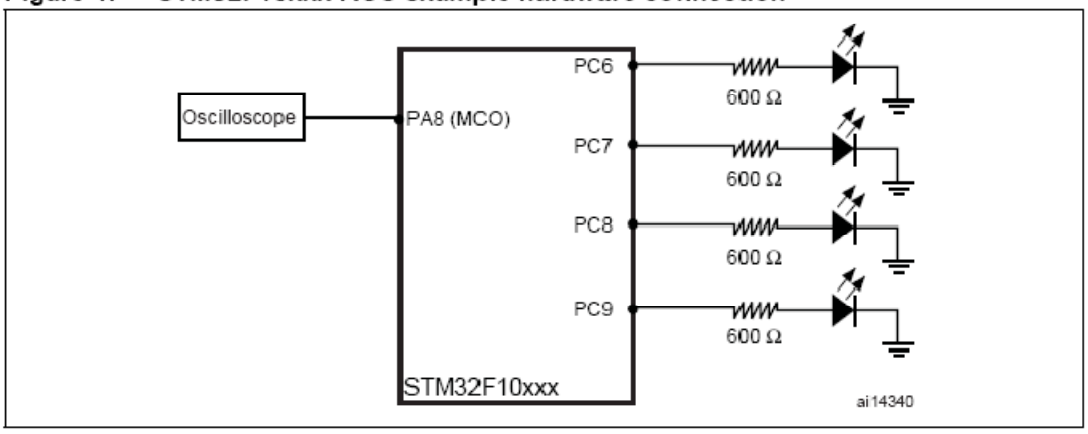
5.1 如何使用STM32F10xxx RCC

介绍了如何配置系统时钟源和AHB，APB2，APB1预分频。给出了一个使用了大部分RCC驱动功能的示例程序。

5.1.1 硬件描述

下图给出了硬件连接的示例

Figure 1. STM32F10xxx RCC example hardware connection



5.1.2 固件描述

给出的固件库中包含了RCC驱动，它通过一系列的函数支持所有的RCC特性。给出的示例程序也使用到了大部分的功能。它展示了如何配置系统时钟源和AHB，APB2，APB1预分频。系统时钟频率通过使用了PLL（锁频环）配置为72MHz，

为了调试，示例程序给出了如何使用RCC_GetClocksFreq函数来重新获得当前的状态和不同片上时钟的频率。RCC_ClockFreq结构体包含不同的片上时钟的频率，能够使用工具链调试器

这个示例也演示了高速外部时钟（HSE）错误检测：当HSE时钟消失后（损坏或者是没有连接到外部晶振），HSE时钟和PLL被禁用（但PLL配置没有改变），HIS时钟就选择为系统的时钟源并产生一个不可屏蔽中断。在不可屏蔽中断服务程序中（ISR），使能HSE和HSE可用中断。当HSE时钟恢复后，HSERDY中断产生，在RCC ISR中，系统时钟配置恢复为原来的状态（HSE时钟没有出现错误前的状态），HSE时钟能够通过MCO引脚（PA8）监视。

四个LED灯连接到PC6 PC7 PC8 PC9引脚，它们以一定的频率翻转，这个频率取决于系统时钟。

可参见RCC示例程序（STMicroelectronics 微控制器的网站）

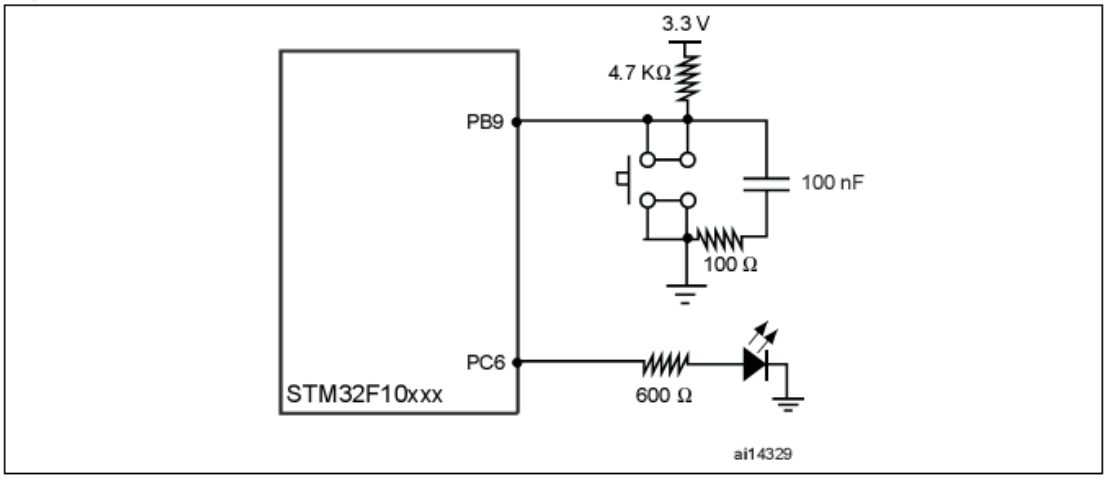
6 怎样使用STM32F10xxx EXTI 控制器

该节介绍了如何使用EXTI固件库把GPIO引脚配置为一个外部中断，如何翻转一个LED灯。给出的示例程序使用了大部分EXTI驱动的功能：配置，使能，软件挂起位，获得EXTI状态，清除挂起位。

6.1 硬件描述

下图显示了如何将一个按钮连接到EXTI线上，一个LED接到随意的一个GPIO引脚。按钮用作EXTI的边沿触发而LED用来指示灯。

Figure 12. STM32F10xxx EXTI Line connection



6.2 固件描述

提供的固件库包括了EXTI驱动，它通过一系列的函数支持所有的EXTI特性。并给出了一个使用了大部分功能的示例。

在这个示例程序中，EXTI Line9配置为在下降沿产生中断。在中断处理程序中，连接到PC6的LED灯翻转，也就是说在每次下降沿翻转。

可参见EXTI示例程序（STMicroelectronics 微控制器的网站）。

6.3 总结

使用EXTI控制器，我们所需要做的就是简单的硬件连接和简单的软件配置。

7 PWR应用示例

这部分提供了STM32F10xxx PWR外设的实际应用示例

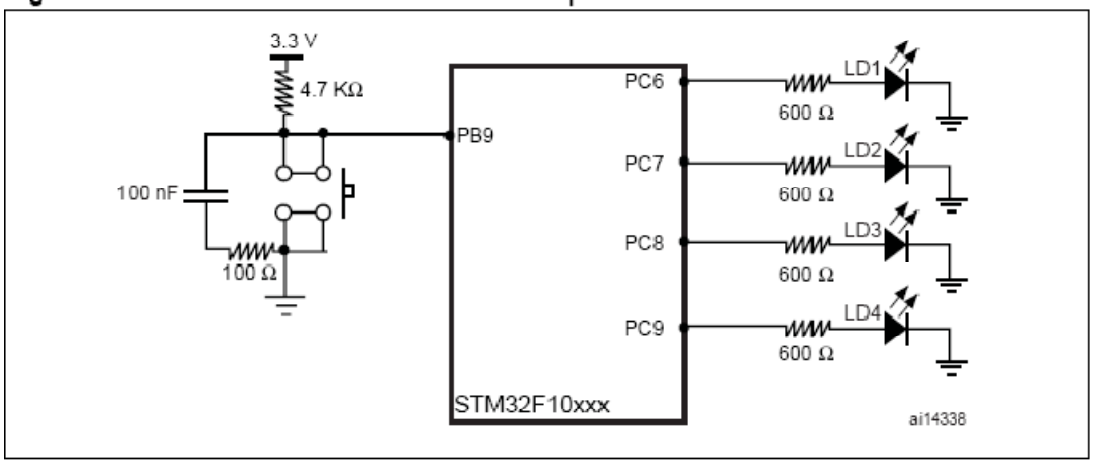
7.1 STM32F10xxx的停止模式

这节介绍了使用PWR固件库如何使系统进入停止模式，如何使用EXTI Line中断唤醒系统。并给出了一个使用了大部分PWR驱动功能的示例程序。

7.1.1 硬件描述

下图给出的是示例程序使用的硬件连接

Figure 1. STM32F10xxx STOP mode example hardware connection



7.1.2 固件描述

给出的固件库包含有PWR驱动，它通过一系列的函数来支持所有PWR的特性。给出了一个使用了大部分功能的示例程序

示例程序展示了系统如何进入STOP模式，使用EXTI Line中断从STOP模式唤醒。EXTI Line源为PB9和RTC定时器。EXTI line9（PB9）配置为在上升沿产生一个中断，EXTI Line17（RTC Alarm）配置为在上升沿产生中断，RTC定时器使用低速外部振荡器，定时基准为1秒。系统时钟使用高速外部振荡器为72MHz。

系统进入/退出STOP模式：

系统启动2秒后，RTC配置为3秒后产生一个Alarm事件。然后系统进入STOP模式。为了把系统从STOP模式中唤醒，必须在EXTI Line9上产生一个上升沿，否则RTC Alarm将在3秒后唤醒系统。一旦退出STOP，系统时钟就重新配置为原来的状态（因为HSE和PLL在STOP模式下禁用）。

延迟一段时间后，系统重新进入STOP模式，采用上述的方式退出。而且在无限循环中重复这个动作。

四个LED分别连接到PC6 (LD1), PC7 (LD2), PC8 (LD3) 和PC9 (LD4) 引脚来监视系统的状态。

- LD1 亮/ LD4 暗: 系统在RUN 模式
- LD1 暗 / LD4 亮: 系统在STOP模式
- 如果使用EXTILine9退出STOP模式，LD2翻转
- 如果使用EXTILine17（RTC Alarm）退出STOP模式，LD3翻转

可参见PWR示例程序一。

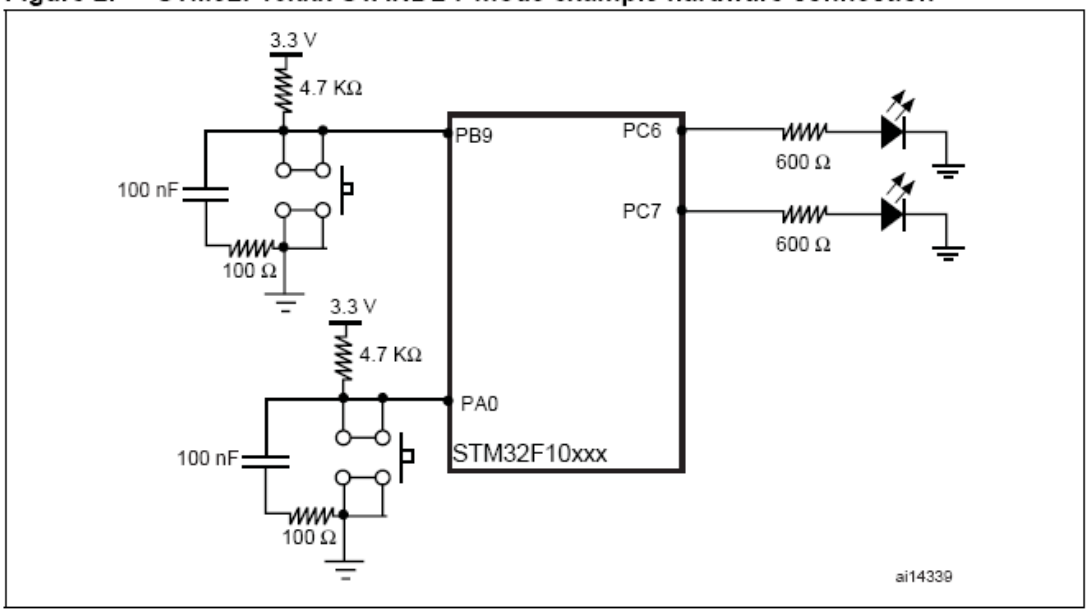
7.2 STM32F10xxx 待机模式

介绍如何使用PWR固件库使系统进入STANDBY模式，如何使用外部RESET，RTC Alarm或者WKUP引脚来唤醒系统。给出的示例使用了大部分PWR驱动功能。

7.2.1 硬件描述

硬件连接如下如图

Figure 2. STM32F10xxx STANDBY mode example hardware connection



7.2.2 固件描述

这个示例展示了如何使用PWR固件库使系统进入STANDBY模式，如何使用外部RESET，RTC Alarm或者WKUP引脚来唤醒系统。给出的示例使用了大部分PWR驱动功能。

在相应的固件中，系统时钟设置为72MHz，EXTI Line9(PB9)配置为在下降沿产生中断，SysTick为每250ms产生一次中断。在SysTick中断处理程序中，连接到PC6的LED灯点亮，而不管系统是处在STANDBY模式还是处在RUN模式。

当在EXTI Line9检测到下降沿，产生一个中断，在EXTI中断处理程序中，RTC配置为每3秒钟产生Alarm事件，然后系统进入STANDBY模式，PC6引脚停止连接到LED（LED灯灭），一个上升沿作用在WKUP引脚或者是外部RESET将系统从STANDBY模式中唤醒。如果3秒内，WKUP和RESET都没有发生，RTC Alarm将唤醒系统 。

系统从STANDBY模式中唤醒后，程序的执行重新开始，PC6引脚连接，PC7引脚设置为高，RTC配置没有改变。所以没有必要重新配置RTC。

两个LED灯连接到PC6（LD1） PC7（LD2）引脚用来监视系统的状态。

LD1点亮：系统在运行状态

LD1关闭/LD2关闭：系统在STANDBY模式

LD2点亮：系统从STANDBY模式中唤醒

可参见PWR示例程序2（STMicroelectronics 微控制器的网站）。

8 BKP 应用示例

这部分提供了STM32F10xxx BKP外设的实际应用示例

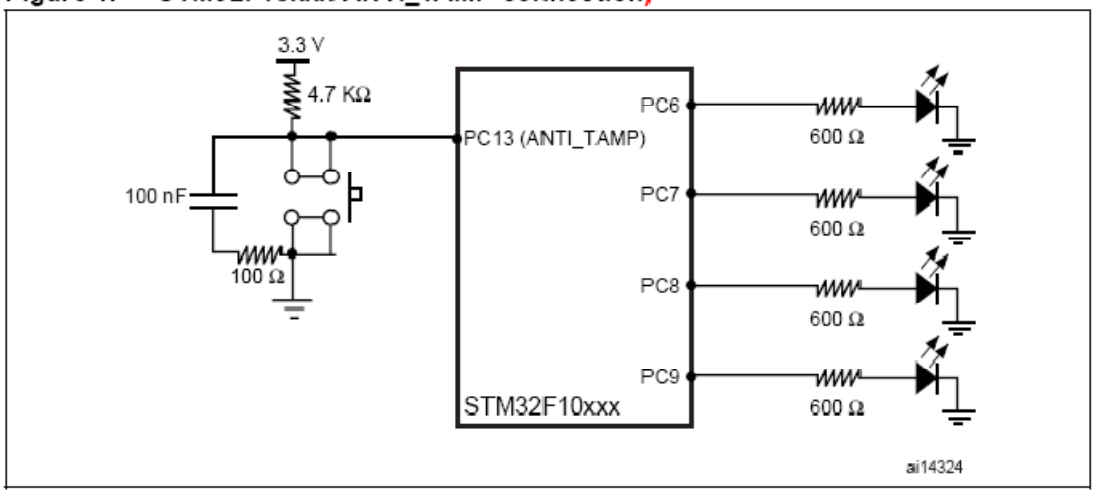
8.1 如何 读/写 数据 从/到 备份数据寄存器

这一节描述了如何 读/写 数据 从/到 备份数据寄存器。下面的例子应用了BKP 驱动的绝大部分功能。

8.1.1 硬件描述

图1 显示了一个硬件连接的示例。

Figure 1. STM32F10xxx ANTI_TAMP connection)



8.1.2 固件描述

固件包括了BKP 驱动，它通过一系列函数来支持 所有的BKP特征。提供的一个示例使用大部分函数来显示 如何 读/写 数据 从/到 备份数据寄存器。它也显示了干涉检测特性。

相关的固件表现如下：

设定 管脚ANTI_TAMP 为 低有效，并使能Tamper(干涉)中断。

它 写 数据到所有的 备份数据寄存器，然后检查这些数据是否正确写入。如果是，连接到 PC6 的 LED 变亮 ，否则 连接到 PC7 的 LED 变亮。

当将一个低电平作用到管脚ANTI_TAMP(PC13)时，备份数据寄存器 被 复位 ， Tamper(干涉)中断形成。相应的中断服务程序(ISR) 检测备份数据寄存器 是否全被清空。如果是，连接到 PC8 的 LED 变亮，否则连接到 PC9 的 LED 变亮。

这个固件在STM32F 10XXX 固件库中被作为 BKP 示例1 ，可从www.st.com中获取。

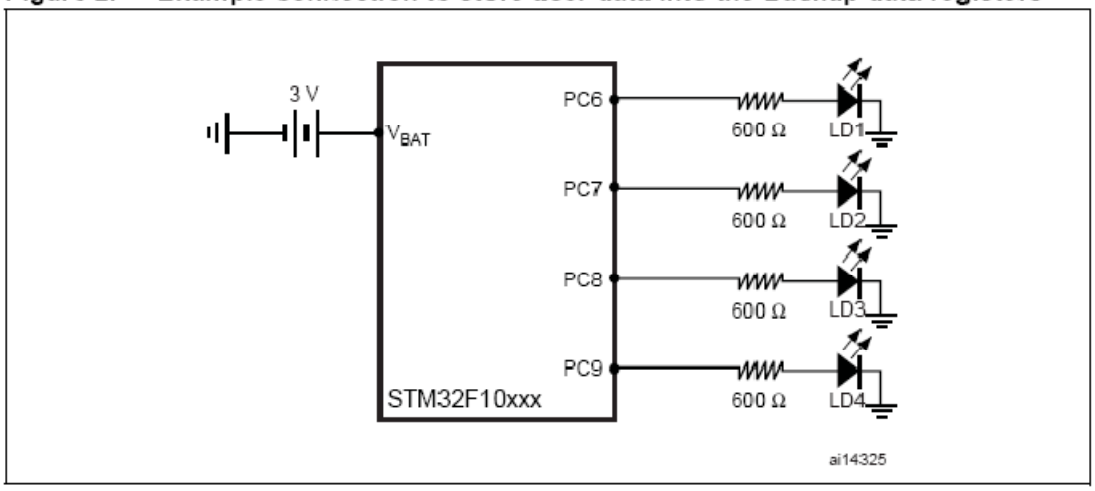
8.2 如何 存储用户数据 到 备份数据寄存器

这部分描述了如何 存储用户数据 到 备份数据寄存器。下面的例子应用了BKP 驱动的绝大部分功能。

8.2.1 硬件描述

图2 显示了一个硬件连接的示例。

Figure 2. Example connection to store user data into the Backup data registers



8.2.2 固件描述

固件包括了BKP 驱动，它通过一系列函数来支持 所有的BKP特征。提供的一个示例使用大部分函数来显示 如何 存储用户数据 到 备份数据寄存器。

当V_{DD}被关掉后，备份区域仍可以由V_{BAT}供电，所以当VBAT引脚连着电池的时候备份区域的内容不会丢失。

程序表现如下：

启动后 程序 核对 电路板是否 开启电源。如果是，备份数据寄存器中的数据被核对。

如果有一个 电池 连接到 V_{BAT} 管脚，寄存器中的数据被保留。

如果没有 电池 连接到那个V_{BAT}管脚 ， 寄存器中的数据 丢失。

四个连接到管脚 PC6（LD1），PC7（LD2），PC8（LD3），PC9（LD4）的LEDs用来按如下规则指示系统状态：（ Power On Reset ：上电复位，缩写为POR）

LD3亮/LD1亮：一个POR 发生了，并且 BKP数据寄存器中的数据正确。

LD3亮/LD2亮：一个POR发生了，并且BKP数据寄存器中的数据不正确 或者 它们 还没有被编程（如果程序是第一次被执行）

LD3关/LD1关/LD2关：POR没发生

LD4开：程序正在运行中

这个固件在STM32F 10XXX 固件库中被作为 BKP 示例2 ，可从www.st.com中获取。

9 RTC应用示例

这一部分提供了关于STM32F10xxx RTC外设的使用的实际应用示例。

9.1 STM32F10xxx RTC和备份域（backup domain或BKP domain）

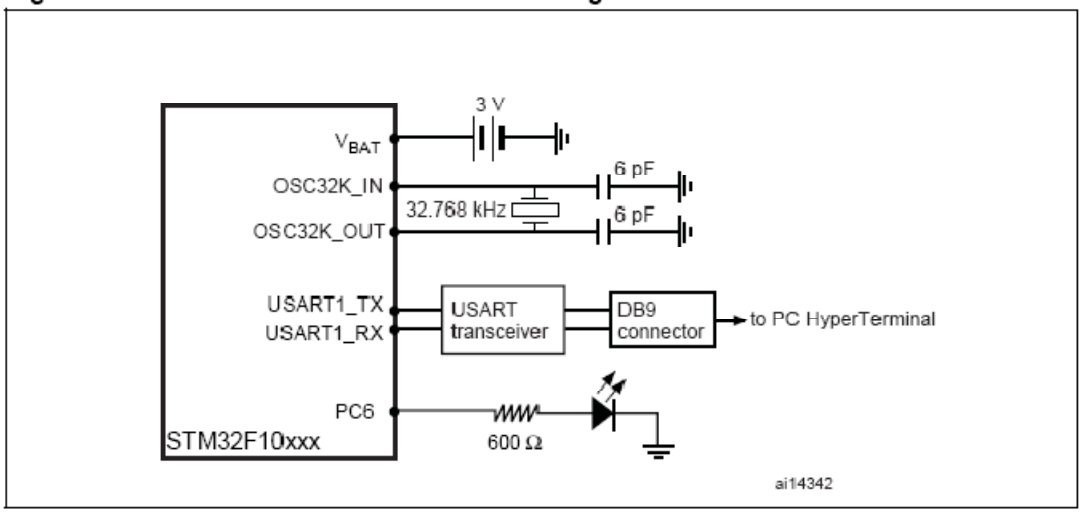
介绍如何使用RTC外设。在演示程序中，它演示了如何在预分频和中断方面设置RTC外设，来保

存时间和产生一个第二次中断

9.1.1 硬件描述

下图展示了RTC硬件的典型配置：外部晶振，电源。还有一个LED灯来指示秒。

Figure 1. STM32F10xxx RTC hardware configuration



9.1.2 固件描述

提供的固件库中包含有RTC驱动，通过一系列的函数来支持所有的RTC功能。并给出了一个使用了大部分功能的示例。

RTC时钟源可以是LSI或者LSE。通过main.c文件中的#define RTCClockSource_LSI 或者#define RTCClockSource_LSE来选择时钟。RTC时钟能够在Tamper引脚（PC13）输出。为了使能这个功能，去掉Main.c文件RTCClockOutput_Enable的注释。

RTC是在BKP域，当V_{DD}关掉后，可由V_{BAT}供电。所以在电池连接到V_{BAT}引脚时RTC的配置不会丢失。备份数据寄存器1中（BKP_DR1）写入一个关键值以指示RTC是否已经配置好。

程序按以下顺序执行：

1 程序启动后检查备份寄存器1的值

- 寄存器1的值不正确（BKP-DR1的值不正确或者第一次执行时没有编程写入），RTC被配置并且会要求使用者设置时间

- 寄存器1的值正确：意味着RTC已经配置过了，将时间显示在超级终端上。

2 当外部复位而BKP域没有复位，那么RTC的配置不会丢失。

3 当电源复位发生时：

- 电池连接到V_{BAT}引脚时 BKP域不会复位，RTC配置不会丢失。

- 电源没有接V_{BAT}引脚时 BKP复位，RTC配置丢失

在RTC中断服务程序中，连接到PC6的LED每1秒翻转一次。

c库函数Printf重定向到USART1，打印的消息会通过USART1输出到超级终端上。

参见RTC示例程序，可从www.st.com中获取。

10 Flash存储器应用示例

这一部分提供了关于STM32F10xxx Flash存储器的使用的实际应用示例。

10.1 怎样对STM32F10xxx Flash存储器编程

介绍了如何使用STM32f10xxx的Flash存储器固件库来对Flash存储器编程

10.1.1 固件描述

提供的固件库中包含有Flash存储器的驱动，该驱动通过一系列的函数支持所有的Flash存储器特性。给出的示例程序使用了大部分的功能。

复位后，Flash存储器的编程/擦写被控制器锁定。使用Flash-Unlock函数来解锁。

在将要编程前，使用Flash 存储器页擦除特性，执行一个擦除操作。擦除程序开始计算要使用的页数，这些页使用FLASH_ErasePage一个个地擦除。

一旦这些操作完成后，使用FLASH-ProgramWord来执行写操作。检测写入的数据，操作的结果放在MemoryProgramStatus变量中。

可参见FLASH示例1。

10.2 如何使能和禁用STM32F10xxx Flash存储器的写保护

介绍如何使用Flash存储器固件库来使能和禁止STM32F10xxx的Flash存储器的写保护。

10.2.1 固件描述

提供的固件库种包含有Flash存储器驱动，它通过一系列的函数来支持所有的Flash存储器特性，并给出了一个使用了大部分功能的示例程序。

10.2.2 使能写保护

首先，去掉#define WriteProtection-Enable行的注释来使能写保护

为了保护一些页，使用者可以调用FLASH-EnableWriteProtection函数.

这个函数的参数定义了要保护的页的数量，例如，要保护的页为page 24到page 31这8个页，参数为FLASH-WRProt-Pages24to27|FLASH-WRProt-Pages28to31。

为了能够加载新操作的字节值，系统必须复位，可以使用NVIC-GenerateSystemReset函数就是用来复位系统的。

10.2.3 禁止写保护

要禁用写保护，请将#define WriteProtection-Disable行的注释去掉。

要禁用STM32F10xxxFlashMemory的写保护，必须要擦除信息存储块。可以使用FLASH-EraseOptionBytes来完成。

为了能够加载新的操作字节值，需要一次系统复位，可以使用NVIC-GenerateSystemReset函数来完成。

如果页没有写保护，先擦除然后再写。

可参见Flash示例程序二，可从www.st.com中获取。

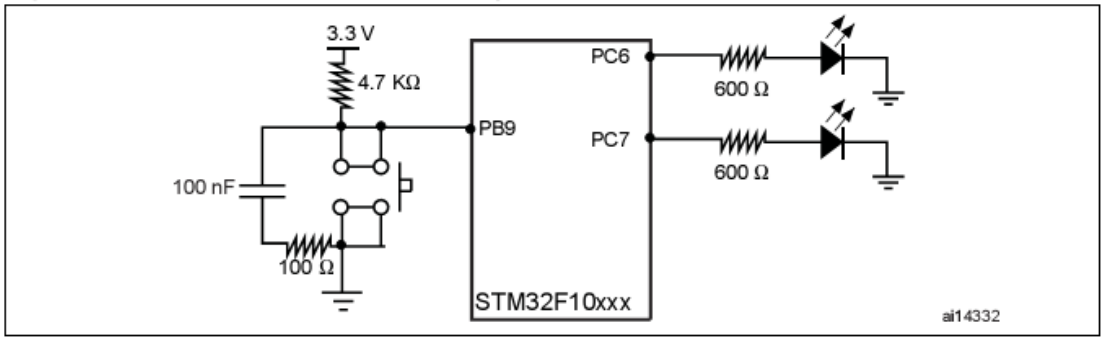
11 怎样使用STM32F10xxx IWDG

这一部分介绍了如何配置独立看门狗（IWDG）以定时重新装入IWDG计数器。并给出了一个使用了IWDG大部分功能的示例程序。

11.1 硬件描述

示例程序的硬件电路如下

Figure 18. STM32F10xxx IWDG example hardware connection



11.2 固件描述

给出的固件库包括了IWDG驱动，它通过一系列的函数来支持所有的功能。并给出了一个示例程

序，显示了如何用系统tick定时中断SysTick来重装IWDG计数器，IWDG的超时值为350ms（超时值可能因为LSI RC频率不稳定而有所变化）。

SysTick配置为每250ms产生一个中断。在SysTick的中断处理程序中，IWDG定时器重装来阻止IWDG复位，连接到PC7的LED灯翻转。

EXTI Line9连接到PB9引脚而且配置为在下降沿产生中断。

在NVIC，EXTI Line[9：5]中断向量使能，并且优先级等于0，SysTick中断向量的优先级为1（EXTIIT>SYSTICK IT）

在EXTI Line9用来模拟软件错误：当EXTI Line9事件触发时候（通过按下STM32F10B-EVAL板的按钮），产生相应的中断，在中断处理程序中，连接到PC7的LED灯翻转，并且不清除中断挂起位。CPU执行EXTI Line9的中断处理程序，进入不了SysTick的中断处理程序（IWDG计数器没有重装），所以当IWDG计数到0时，IWDG产生一个复位。

如果EXTI Line9的事件没有发生，IWDG计数器能够在SysTick的中断处理程序中重装，阻止了IWDG的复位。

如果IWDG复位了，LED灯将在系统恢复操作中点亮。

在这个示例程序中，系统的时钟由HSE（高速外部时钟 8MHz）提供

参见IWDG示例程序。可从www.st.com中获取

12 怎样使用STM32F10xxx WWDG

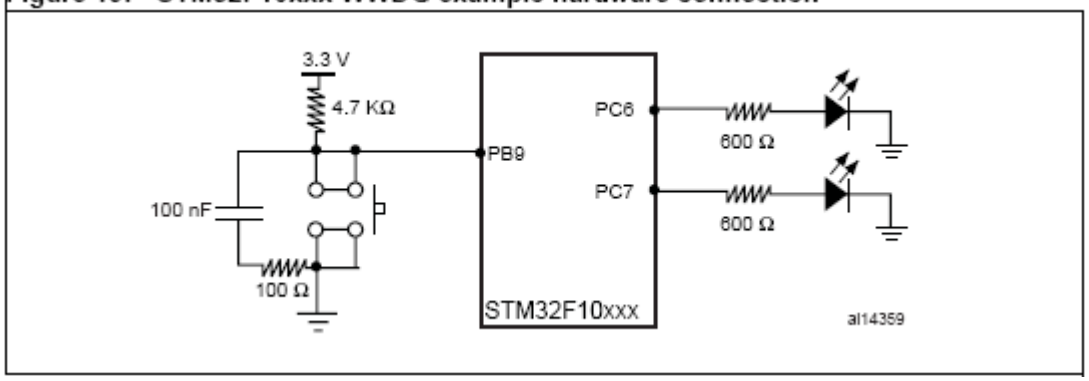
这一部分描述了怎样配置WWDG以定时更新WWDG计数器。将给出一个使用了大部分WWDG

功能的示例。

12.1 硬件描述

下图表现了示例中使用的硬件连接

Figure 19. STM32F10xxx WWDG example hardware connection



12.2 固件描述

提供的固件包含WWDG驱动，它通过一个函数集支持所有WWDG特性。将提供一个有大部分函数使用的示例。这个示例表明怎样定时更新WWDG计数器，使用early wakeup interrupt：早唤醒中断，缩写为EWI。

WWDG 超时被设置为262 ms，刷新窗口被设置为41h，且EWI 使能。当WWDG 计数器达到40h，将产生EWI。在WWDG 的ISR里，计数器被刷新以阻止WWDG复位并且连接到PC7的LED 被转换。

EXTI Line9 被连接到PB9引脚，并配置为在信号的下降沿生成一个中断。

NVIC (嵌套向量中断控制器)中，EXTI Line[9:5] 中断向量以优先级等于0被使能，且WWDG中

断向量以优先级等于1被使能(EXTI IT > WWDG IT).

EXTI Line9 被用作模拟软件故障：EXTI Line9 事件一发生(按压STM3210B评估板上的关键按钮)，相应的中断将被处理。在ISR，连接到PC7的LED被关闭，且EXTI Line9 挂起位没有被清除。所以CPU无限的执行EXTI Line9 ISR ，WWDG ISR从不被执行(WWDG 计数器没有被更新)。因此，当WWDG 计数器降至3Fh，WWDG 的复位发生。.

如果EXTI Line9 事件没有发生，在WWDG ISR中WWDG 计数器不断地更新，且没有WWDG的重置。

如果WWDG复位发生了，当系统从重置中恢复以后，连接到PC8的LED 开启。

在示例中系统时钟由高速外部时钟(HSE，8 MHz)带动。这个固件将作为 *WWDG* 示例在STM32F10xxx 固件库中提供，可以从ST微电子公司的微控制器网站上下载。

13 结论

这些在应用笔记中描述的示例，将和STM32F10xxx 固件库一起提供，让用户更容易理解产品系统外设。也省去了用户处理外设底层寄存器和位的麻烦。

14 修订记录

表1 修订记录

日期	修订	改变
2007-7-10	1	初次发布

15 版权声明：

MXCHIP Corporation拥有对该中文版文档的所有权和使用权

意法半导体（ST）拥有对英文原版文档的所有权和使用权