# AN5044
# 应用笔记

## STM32标准外设库迁移至STM32Cube底层库

## 引言

STM32价值主张倡导提供完整和全面的固件库，为开发人员提供构建嵌入式应用程序的初始框架。

该框架的核心是众所周知的标准外设库（SPL），所有中间件组件都使用SPL来访问STM32外设。

多年来，STM32产品不断发展，为开发人员提供各种解决方案以平衡成本，功耗和性能，其中尤为出名的是STM32Cube底层（LL）驱动程序。

对于STM32微控制器应用的设计人员而言，轻松地升级微控制器型号和/或更换为另一种固件解决方案是一件至关重要的事情。由于SPL已经停止研发，随着产品需求和规格不断增长，对所使用的各种外设提出了额外的要求，因此，这种迁移通常必不可少。

本应用笔记介绍了从现有使用STM32 SPL开发的基于STM32的应用程序迁移到使用STM32Cube LL驱动程序的任何其他类型的微控制器的步骤。

本应用笔记汇总了将基于STM32SPL的应用程序成功迁移到使用STM32CubeLLAPI所需的所有最重要信息。

它包含三个部分。

- STM32 SPL与STM32Cube LL架构概述：介绍两种解决方案以及二者之间的比较。
- STM32 SPL至STM32Cube LL手动迁移：详细说明了手动迁移基于标准外设库应用程序的步骤。
- STM32 SPL至STM32Cube LL自动迁移：利用SPL2LL-转换器迁移工具作为解决方案，自动迁移基于标准外设的应用程序。

# 目录

# 表格索引

# 图片索引

# 1 STM32 SPL与STM32Cube LL架构概述

本节介绍STM32标准外设库（SPL）和STM32Cube LL的架构，对两种解决方案之间的主要差异进行对比和总结。

## 1.1 STM32 SPL

### 1.1.1 概述

#### CMSIS

STM32 SPL的CMSIS由两层组成：第一层是内核外设访问层，包含名称定义、地址定义和辅助API，用于访问内核Cortex-Mx寄存器和外设。第二层是STM32外设访问层，定义了设备的所有外设寄存器、位域和存储器映射。

#### STM32 SPL驱动程序

该库基于模块化编程方法构建，确保构建主应用程序的多个组件之间保持独立。使用此库即可在广泛的产品系列和评估板上实现轻松移植，只需对公共部分的代码进行最小的更改。

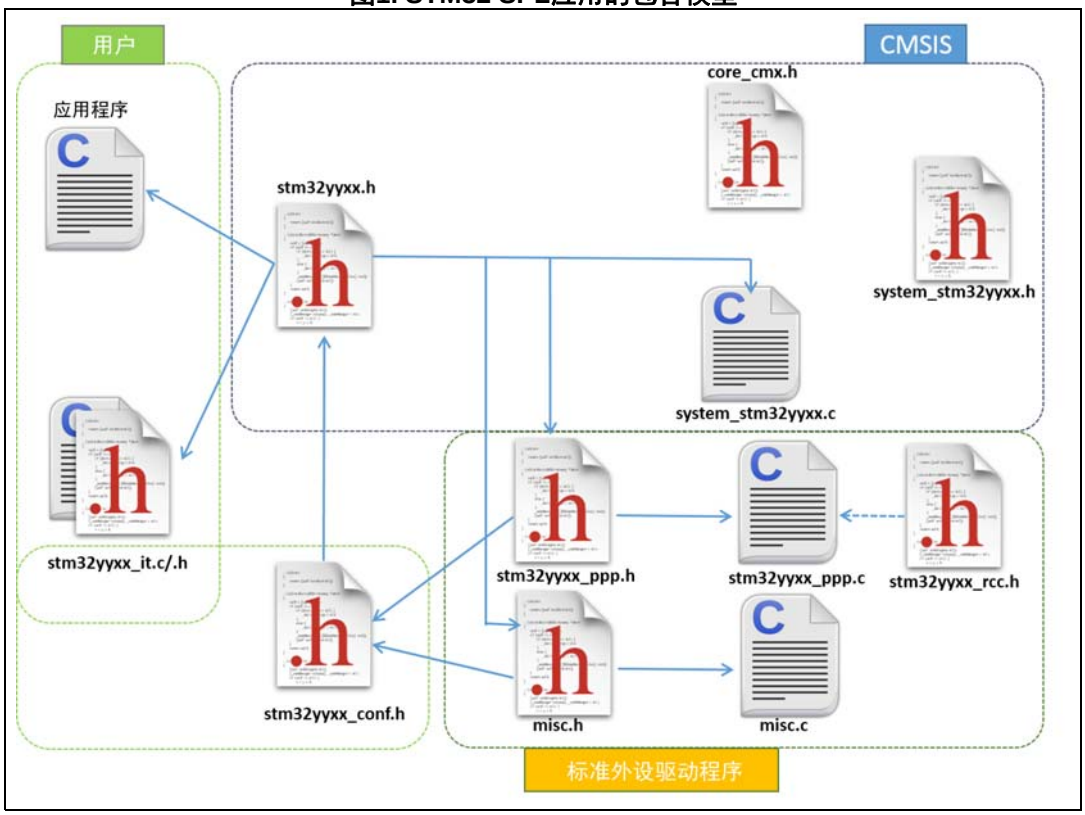
STM32 SPL驱动程序为每个外设提供驱动程序和头文件。每个驱动程序都包含一组涵盖所有外设功能的API。

通过检查所有库函数的输入值，还可以实现运行时故障检测。此类动态检查有助于增强软件的稳健性。因此，它适用于开发和调试用户应用程序。

### 1.1.2 包含模型

*图 1* 基于STM32 SPL的默认用户应用程序，介绍了文件包含模型。

**图1. STM32 SPL应用的包含模型**




每个STM32嵌入式外设都有一个源代码文件stm32yyxx_ppp.c和一个头文件stm32yyxx_ppp.h。stm32yyxx_ppp.c文件包含使用PPP外设所需的所有固件API。

stm32yyxx_conf.h文件被视为用户文件，根据应用程序中使用的外设进行自定义和定制。在运行任何应用程序之前，使用此文件指定与库驱动程序连接的参数集。

*表 1*对用户基于STM32 SPL的应用程序所引用的上述文件进行重新分组和介绍：

表1. STM32 SPL应用程序的文件描述

| - | 文件名 | 说明 |
|---|---|---|
| STM32 SPL | stm32yyxx_conf.h | 外设驱动程序配置文件。<br>用户可以使用模板来启用或禁用包含外设头文件。此文件还可通过预处理器定义USE_FULL_ASSERT，在编译固件库驱动程序之前启用或禁用库运行时故障。 |
| | stm32yyxx_ppp.h | PPP外设的头文件。 |
| | stm32yyxx_ppp.c | 用C语言编写的PPP外设驱动程序源代码文件。 |
| | stm32yyxx_it.h | 头文件，包括所有中断处理程序原型。 |
| | stm32yyxx_it.c | 模板源文件，包含Cortex-Mx异常的中断服务程序（ISR）。用户可以为所使用的外设添加额外的ISR（有关可用的外设中断处理程序的名称，请参见startup_stm32yyxx.s）。 |
| CMISIS | stm32yyxx.h | CMSIS Cortex-Mx STM32yyxx设备外设访问层头文件。<br>此文件是应用程序员在源代码中使用的唯一需要包含的文件。此文件包含：<br>– 配置区，可用于选择：<br>–目标应用程序所使用的设备<br>–在应用程序代码中使用或不使用外设的驱动程序（即，代码将基于直接访问外设寄存器而非驱动程序API），则此选项由#define USE_STDPERIPH_DRIVER控制<br>–改变一些应用特定的参数，例如HSE晶振频率<br>– 数据结构和所有外设的地址映射<br>– 外设的寄存器声明和位定义<br>– 用于访问外设寄存器硬件的宏 |
| | system_stm32yyxx.h | CMSIS Cortex-Mx STM32yyxx设备外设访问层系统头文件 |
| | system_stm32yyxx.c | CMSIS Cortex-Mx STM32yyxx设备外设访问层系统源文件 |

## 1.2 STM32Cube底层驱动程序

### 1.2.1 概述

作为STM32Cube固件HAL的一部分，底层（LL）驱动程序旨在提供面向专家的快速轻量级层，与HAL相比，更接近硬件。与HAL相反的是，对于不关注优化访问的外设而言，LL API并不适用。

底层（LL）驱动程序旨在提供：

- 一组函数，用于根据数据结构中指定的参数，对外设主要特性进行初始化
- 一组函数，用于使用每个字段的复位值填充初始化数据结构
- 函数，用于执行外设去初始化（外设寄存器恢复为默认值）
- 一组内联函数，用于直接和原子寄存器访问
- 由于LL驱动程序既可以在独立模式下使用（不使用HAL驱动程序），也可以在混合模式下使用（使用HAL驱动程序），因此完全独立于HAL
- 涵盖全部支持的外设特性。

底层驱动程序基于STM32外设的可用功能提供硬件服务。*表 2*列出了底层范围涵盖的STM32嵌入式外设：

**表2. LL-supported外设**

| 外设 | | 支持STM32Cube LL |
|---|---|---|
| 系统 | FLASH | 无 |
| | EXTI | 有 |
| | GPIO | 有 |
| | DMAX | 有 |
| | PWR | 有 |
| | RCC | 有 |
| | Cortex | 有 |
| | SYSCFG | 有 |
| | NVIC(1) | 否（CMSIS已涵盖） |
| 模拟 | ADC | 有 |
| | DAC | 有 |
| | COMP | 有 |
| | OPAMP | 有 |
| | DFSDM | 无 |
| 定时器 | RTC | 有 |
| | TIM | 有 |
| | LPTIM | 有 |
| | HRTIM | 有 |
| | WWDG | 有 |
| 密码 | CRC | 有 |
| | CRYP | 无 |
| | HASH | 无 |
| | RNG | 有 |

表2. LL-supported外设（续）

| 外设 | | 支持STM32Cube LL |
|---|---|---|
| 基本连接 | I2C | 有 |
| | UART/USART/LPUART | 有 |
| | SWPMI | 有 |
| | SPI/I2S | 有 |
| | SDMMC(SDIO) | 无 |

1. 在STM32 SPL中，misc.h/.c驱动程序涵盖了NVIC

LL API完全体现了硬件功能，提供单次操作。必须按照微控制器产品线参考手册中描述的编程模型来调用这些操作。

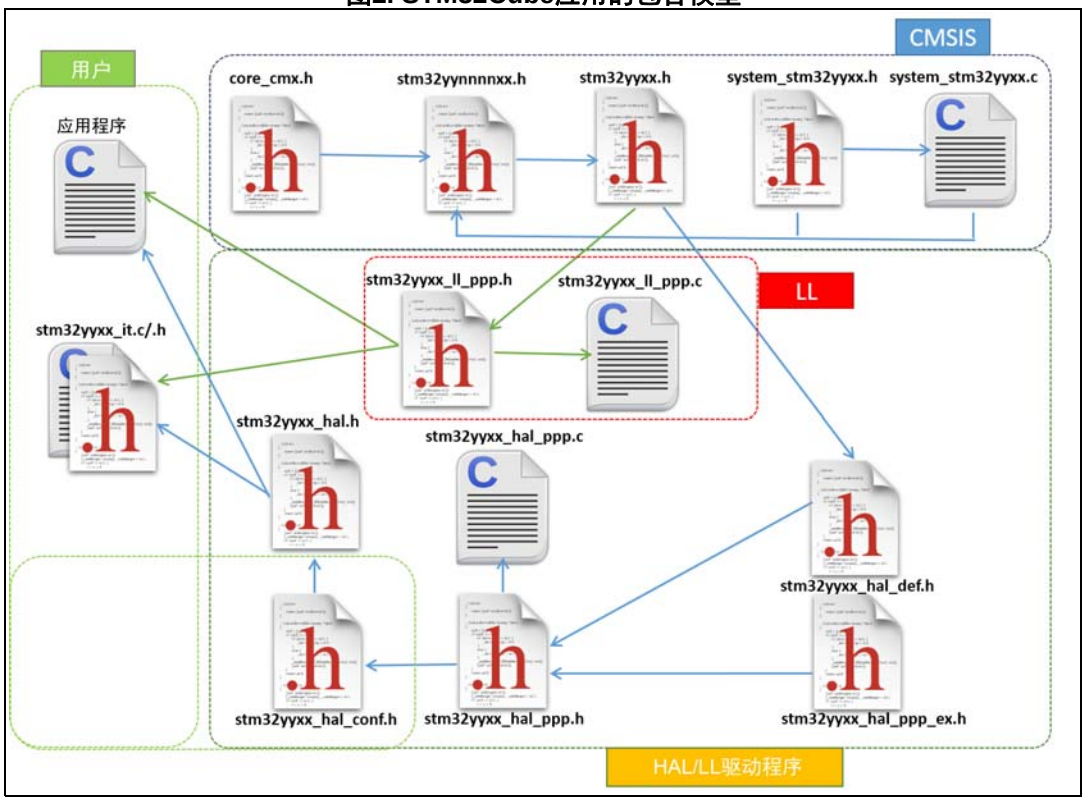
因此，LL服务不执行任何处理，不需要任何额外的存储器资源来保存它们的状态、计数器或数据指针：通过改变相关的外设寄存器内容即可执行所有操作。

通过物理实例（映射在外设基址寄存器上的外设寄存器结构）调用所有底层驱动程序，每个物理外设的底层驱动程序都在一个单独的头文件中，他们在一个模块中给出。

### 1.2.2 包含模型

作为STM32Cube应用程序，根据采用的驱动程序的包含模型不同，有不同的文件暴露给用户。*图 2*显示了通过STM32Cube HAL或底层驱动器包含的文件。

图2. STM32Cube应用的包含模型




根据STM32Cube包提供的不同内容，将STM32Cube应用程序分为三类：

- HAL应用程序：该应用程序仅基于HAL驱动程序，遵循蓝色箭头显示的包含模型。
- LL应用程序：由于LL驱动程序是独立的， 用户在开发应用程序时可以仅使用LL驱动程序，从源文件引用这些驱动程序即可。请注意，LL驱动程序不能相互包含，但必须仅包含CMSIS设备文件。因此，无需使用配置文件，且用户必须在其应用程序的入口点文件中包含所使用的驱动程序。
- 混合应用程序：用户调用HAL和LL驱动程序，使用二者的API来开发代码。驱动程序彼此独立，其包含方法互不干扰。

表3. STM32Cube LL应用程序的文件描述

| 文件名 | | 说明 |
|---|---|---|
| HAL | stm32yyxx_hal.c | 这是HAL初始化的通用部分 |
| | stm32yyxx_hal.h | 这是HAL初始化的通用部分的头文件 |
| | stm32yyxx_hal_ppp.c | 这是PPP驱动程序的c源文件。PPP驱动程序是一个独立模块，用户应在配置文件中启用对应的USE_HAL_PPP_MODULE定义，方可在项目中使用此驱动程序 |
| | stm32yyxx_hal_ppp.h | 这是PPP驱动程序的头文件。 |
| | stm32yyxx_hal_ppp_ex.h/.c | 这些文件是驱动程序内标准API集的扩展。 |
| | stm32yyxx_def.h | 通用的HAL资源，例如通用定义声明、枚举、结构和宏。 |
| LL | stm32yyxx_ll_ppp.h/c | 这是PPP底层驱动程序的h源文件。底层PPP驱动程序是一个独立模块。如要使用，应用程序应包含stm32yyxx_ll_ppp.h |
| CMSIS | stm32yyxx.h | CMSIS Cortex-Mx STM32yyxx系列外设访问层头文件。 |
| | stm32yynnnnxx.h | CMSIS Cortex-Mx STM32yyxx设备外设访问层头文件。 |
| | system_stm32yyxx.h/.c | 应在启动时调用此文件包含的API，即，在复位之后和跳转到 main 函数之前。 |

与HAL驱动程序相反，底层驱动程序并非基于过程模型构建，而是基于寄存器的简单访问操作。因此，底层驱动没有配置文件。

### 1.2.3 API定义级别和分类

底层驱动程序旨在提供一个抽象API层，涵盖STM32代码片段API和标准外设驱动程序功能。

底层驱动程序提供了一组互补的基本API，可自定义或替换高级进程。

每个底层外设驱动程序应涵盖以下三个API级别：

- 级别1：LL_PPP_WriteReg() / LL_PPP_ReadReg()（CMSIS寄存器操作的重定向）。
- 级别2：单次操作API（原子操作），其排序如下：
  - 外设激活/取消激活管理：启用或禁用外设块、子块或相关特性。
    示例：LL_PPP_Disable(PPPx)
  - 外设功能操作管理：开始或启动外设操作，或将外设设置为功能状态。
    示例：LL_PPP_Action()
  - 辅助操作。
    示例：IS_PPP_State(PPPx)

– 特定中断和状态标志管理：处理单个项目的状态和寄存器标志操作（获取、清除、启用、禁用）。
示例：LL_PPP_ClearFlag_XX()

请参见“stm32yyxx_ll_PPP.h”，了解上述API。

- 级别3：全局配置和初始化函数，涵盖“stm32yyxx_ll_PPP.c”提供的相关外设寄存器的完全独立操作。

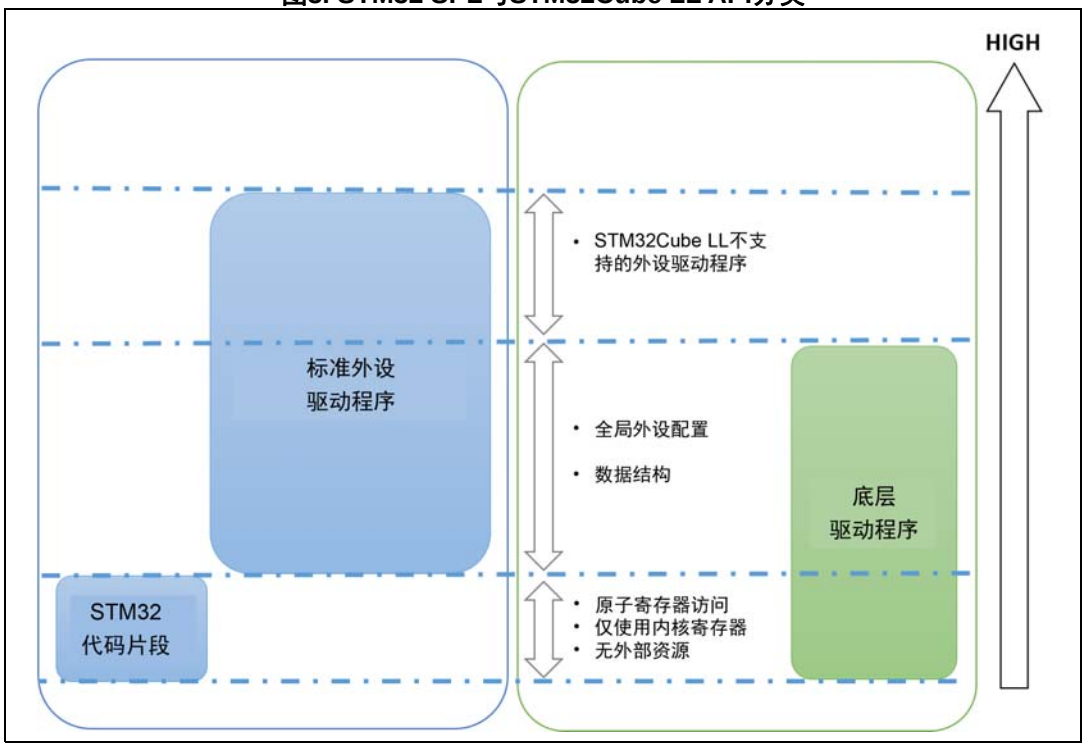
## 1.3 总结

*表 4*对STM32 SPL和STM32Cube CMSIS及其驱动程序组件进行了全面比较：

**表4. STM32 SPL与STM32Cube对比总结**

| - | STM32 SPL | STM32Cube |
|---|---|---|
| CMSIS | – 单个头文件（寄存器和位定义），适用于单个系列中的所有设备。用户需要对所考虑的设备中存在的外设进行分类。<br>– 用户应用程序需要管理不同产品线之间的中断处理程序名称的差异。<br>– system_stm32xx.c：在跳转到main()之前配置系统时钟 | – 给定系列中每个设备的头文件，显示产品的可用特性，为寄存器和位定义提供整齐有序的布局<br>– 从STM32 SPL迁移时，使用别名实现命名差异抽象化，例如中断处理程序的名称。<br>– system_stm32xx.c：此文件不再实施系统时钟配置。用户需要在主文件中执行此文件。 |
| 外设驱动程序 | – API管理简单的寄存器IO操作。必须由用户应用程序完成进程和错误管理。<br>– 面向外设的实现。如果外设特性发生重大更新，则兼容性受损。 | – LL API属于功能导向的，具有直接/原子寄存器访问特性。<br>– LL API反应了STM32系列的硬件功能。<br>– LL在相关外设寄存器上提供完全独立的操作。<br>– 与STM32 SPL的兼容性：不兼容。为底层定义的新API集。 |

图3. STM32 SPL与STM32Cube LL API分类




STM32Cube LL驱动程序提供了一组新的内联函数，可代替代码片段和标准外设驱动程序，允许直接和原子寄存器访问。它独立于HAL驱动程序，因此用户可以独立使用所支持的外设及其特性，且最小化占用的代码空间和存储器资源。因此，可以实现从STM32 SPL到STM32Cube LL的完整迁移。

# 2 STM32 SPL至STM32Cube LL手动迁移

本节介绍如何将基于STM32 SPL开发的应用程序迁移到部署STM32Cube LL的等价应用程序，该应用程序可以位于同一目标设备上，也可以位于不同的STM32系列上。

## 2.1 STM32 SPL与STM32Cube LL API等价

如上一章所示，底层驱动程序提供用于不同层的API和方法。它们包括从基本的和原子寄存器访问（无需外部资源）到更高级别的配置函数。LL驱动程序可以提供完全独立的操作，涵盖STM32 SPL API。

本节将详细介绍STM32 SPL和STM32Cube LL之间的等价。

### 2.1.1 NVIC中断配置

STM32xx_StdPeriph_Lib解决方案允许两种不同的方法来处理CMSIS 内核特性：

1. 通过core_cmx.h文件的CMSIS内核Cortex-Mx驱动程序
2. 通过misc.h/c文件的STM32标准外设驱动程序

但是，由于CMSIS core_cmx.h文件具有一整套宏，几乎涵盖了CMSIS内核Cortex-Mx特性，因此STM32Cube LL解决方案仅可使用CMSIS内核Cortex-Mx驱动程序的方法。

可使用两种迁移方法：

- 对于熟悉使用CMSIS内核Cortex-Mx驱动程序来管理中断的用户，此时对用户应用程序没有任何影响。
- 对于熟悉使用misc.c/.h驱动程序的用户，迁移时应遵循以下步骤：
  - 提取NVIC_InitTypeDef初始化结构中的配置，
  - 使用NVIC_SetPriorityGrouping() API设置优先级分组
  - 使用NVIC_EncodePriority() API对优先级进行编码
  - 使用NVIC_SetPriority() API设置新的IRQn优先级
  - 启用或禁用IRQn外部中断。

*表 5*列出了STM32 SPL和CMSIS内核驱动程序之间的Cortex-Mx等价：

表5. STM32 SPL和CMSIS核心驱动程序之间的Cortex-Mx等价

| STM32 SPL (misc.c/.h) | CMSIS (core_cmx.h) | 备注 |
|---|---|---|
| void NVIC_PriorityGroupConfig(uint32_t NVIC_PriorityGroup); | __STATIC_INLINE void NVIC_SetPriorityGrouping(uint32_t PriorityGroup); | - |
| void NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct); | __STATIC_INLINE void NVIC_SetPriorityGrouping(uint32_t PriorityGroup);<br>__STATIC_INLINE uint32_t NVIC_EncodePriority (uint32_t PriorityGroup, uint32_t PreemptPriority, uint32_t SubPriority);<br>__STATIC_INLINE void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)<br>__STATIC_INLINE void NVIC_EnableIRQ(IRQn_Type IRQn);<br>或<br>__STATIC_INLINE void NVIC_DisableIRQ(IRQn_Type IRQn); | 未在STM32Cube驱动程序上定义NVIC_InitTypeDef结构， |
| void NVIC_SetVectorTable(uint32_t NVIC_VectTab, uint32_t Offset); | 无等价 | SCB->VTOR = NVIC_VectTab \| (Offset & (uint32_t)0x1FFFFF80);<br>1- NVIC_VectTab：指定向量表位于RAM或FLASH存储器中。此参数可取如下值：<br>-NVIC_VectTab_RAM：向量表位于内部SRAM。<br>-NVIC_VectTab_FLASH：向量表位于内部FLASH。<br>2-偏移：向量表基本偏移字段。该数值必须为0x200的倍数。 |

表5. STM32 SPL和CMSIS核心驱动程序之间的Cortex-Mx等价（续）

| STM32 SPL (misc.c/.h) | CMSIS (core_cmx.h) | 备注 |
|---|---|---|
| void NVIC_SystemLPConfig(uint8_t LowPowerMode, FunctionalState NewState); | 无等价 | 1- NewState = ENABLE<br>SCB-> SCR \| = LowPowerMode;<br>2- NewState = DISABLE<br>SCB->SCR &= (uint32_t)(~(uint32_t)LowPowerMode);<br>LowPowerMode：指定系统的新模式，即进入低功耗模式。此参数可取如下值：<br>NVIC_LP_SEVONPEND：等待低功耗SEV。<br>- NVIC_LP_SLEEPDEEP：低功耗DEEPSLEEP请求。<br>NVIC_LP_SLEEPONEXIT：退出低功耗睡眠模式。 |
| void SysTick_CLKSourceConfig(uint32_t SysTick_CLKSource); | __STATIC_INLINE uint32_t SysTick_Config(uint32_t ticks); | - |

### 2.1.2 外设驱动程序

底层库支持大多数系统、模拟、定时器、密码和基本连接外设。

等价可以是一个与其他API匹配的简单API，或者是在复杂的处理情况下调用多个LL API。用户可能需要仔细检查给定API中被操作的寄存器，以找出相应的LL API。底层驱动程序还提供一组与现有的STM32 SPL中相匹配的管理结构和外设初始化和配置功能，从而节省了连续调用LL API的工作量。

ST网站上提供的SPL2LL-转换器迁移工具附带STM32Cube LL驱动程序支持的STM32外设的表格数据库。这些表格配备LL的直接等价值，涉及STM32系列部分特性的可用性。

这些数据库位于SPL2LL-转换器包中的“SPL2LL_User_Manual.chm”文件中。

### 2.1.3 迁移用例

在从STM32 SPL迁移代码至STM32Cube LL驱动程序期间，用户可以采用不同的方法来编写等价的LL代码。这些已被整理到最常见的API用例中，如下所示：

- 仅更改函数名称，参数顺序与等价LL所定义的保持一致。
  *GPIO_PinLockConfig(GPIOA, GPIO_Pin_0)* ↔ *LL_GPIO_LockPin(GPIOA, LL_GPIO_PIN_0)*
- 更新的函数名称，根据其中一个参数值，顺序与等价LL所定义的保持一致：
  *GPIO_WriteBit(GPIOA, GPIO_Pin_0, Bit_SET)* ↔ *LL_GPIO_SetOutputPin(GPIOA, LL_GPIO_PIN_0)*
- 更新的函数名称，可在等价LL函数中更改参数顺序：
  *TIM_ETRClockMode1Config(TIM1, TIM_ExtTRGPSC_DIV2, TIM_ExtTRGPolarity_Inverted, 0);*
  ↔

*LL_TIM_ConfigETR(TIM1, LL_TIM_ETR_POLARITY_INVERTED,*
*LL_TIM_ETR_PRESCALER_DIV2, LL_TIM_ETR_FILTER_FDIV1);*

- 使用具有功能相似性且无API等价的LL参数更新了函数名称：
*ADC_SelectDifferentialMode(ADC1, ADC_Channel_1, ENABLE)*
↔
*LL_ADC_SetChannelSingleDiff(ADC1, LL_ADC_CHANNEL_1,*
*LL_ADC_DIFFERENTIAL_ENDED)*
- 更新了函数名称，添加了新参数：
*CRC_GetIDRegister() ↔ LL_CRC_Read_IDR(CRC)*
- 多个STM32 SPL函数迁移至一个LL函数中，参数可能会随着STM32 SPL驱动程序所需的参数而变化：
*SPI_NSSInternalSoftwareConfig(SPI1,*
*SPI_NSSInternalSoft_Set);SPI_SSOutputCmd(SPI1, DISABLE);*
↔
*LL_SPI_SetNSSMode(SPI1, LL_SPI_NSS_SOFT);*
- 一个STM32 SPL函数迁移至多个LL函数，在等价函数之间共享参数：
*ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 1,*
*ADC_SampleTime_2Cycles5);*
↔
*LL_ADC_REG_SetSequencerRanks(ADC1, LL_ADC_REG_RANK_1,*
*LL_ADC_CHANNEL_1);*
*LL_ADC_SetChannelSamplingTime(ADC1, LL_ADC_CHANNEL_1,*
*LL_ADC_SAMPLINGTIME_2CYCLES_5);*
- 多个STM32 SPL函数迁移至多个LL函数中，可在等价函数中添加、减少和共享参数：
*RTC_CalibOutputConfig(RTC_CalibOutput_512Hz);*
*RTC_CalibOutputCmd(ENABLE);*
↔
*LL_RTC_DisableWriteProtection(RTC);*
*LL_RTC_CAL_SetOutputFreq(RTC, LL_RTC_CALIB_OUTPUT_512HZ);*
*LL_RTC_EnableWriteProtection(RTC);*
- 更新了函数名称，参数从STM32 SPL结构内容中填充。必须从结构中检索值并将其指定为参数：
*RTC_SetTime(RTC_Format_BCD, &RTC_TimeStruct);*
↔
*LL_RTC_TIME_Config(RTC, LL_RTC_TIME_FORMAT_AM_OR_24,*
*RTC_TimeStruct.Hours, RTC_TimeStruct.Minutes, RTC_TimeStruct.Seconds);*
- LL上没有明确的API等价。用户需使用直接寄存器访问进行迁移：
*NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x1000);*
↔
*SCB->VTOR = 0x08000000U | (0x1000U & 0x1FFFFF80U);*

## 2.2 项目创建

在迁移至STM32Cube底层驱动程序之前，请先确保具有正常运行的应用程序，避免产生不必要的问题。

如要更新应用程序代码以使用STM32Cube底层驱动程序，并在给定的STM32 MCU系列上运行，用户可以选择以下两种方法之一：

- **保持旧的应用程序环境**

1. 在应用程序库中创建一个目录，用户在该目录下添加目标设备的STM32Cube驱动程序。这应包含两个文件夹：
   – CMSIS：含有STM32Cube附带的新CMSIS文件。
   – STM32yyxx_HAL_Drivers：包含所需的驱动程序。
2. 更新工具链配置参数：
   – 链接器配置：设备连接和Flash 存储器加载程序。这些文件随最新版本支持目标设备 的工具链一起提供。若需更多信息，请参考工具链文档。
   – 项目文件：添加LL驱动程序的源文件和CMSIS “system_stm32yyxx.c”。

   *注： - 底层环境中没有配置头文件。在应用程序的入口点文件中完成对驱动程序的调用。当迁移需要来自STM32Cube HAL的API时，用户可以添加“stm32yyxx_hal_conf.h”来引用HAL驱动程序。*

   *- 通过添加私有SystemClock_Config() API，在迁移后的main.c文件上手动更新已在system_stm32yyxx.c中执行的默认SPL系统时钟配置。此API已存在于目标STM32的STM32Cube LL_Template项目中。*

   – 项目配置参数：为LL驱动程序和CMSIS文件添加必要的包含路径，并使用所用驱动程序和目标设备所需的关键字更新预处理器符号。
3. 使用STM32CubeLLAPI重写一部分应用程序代码。如要迁移，用户必须查找SPL2LL-转换器迁移工具中提供的STM32 SPL和STM32Cube LL等价。

- **使用可用的底层模板**

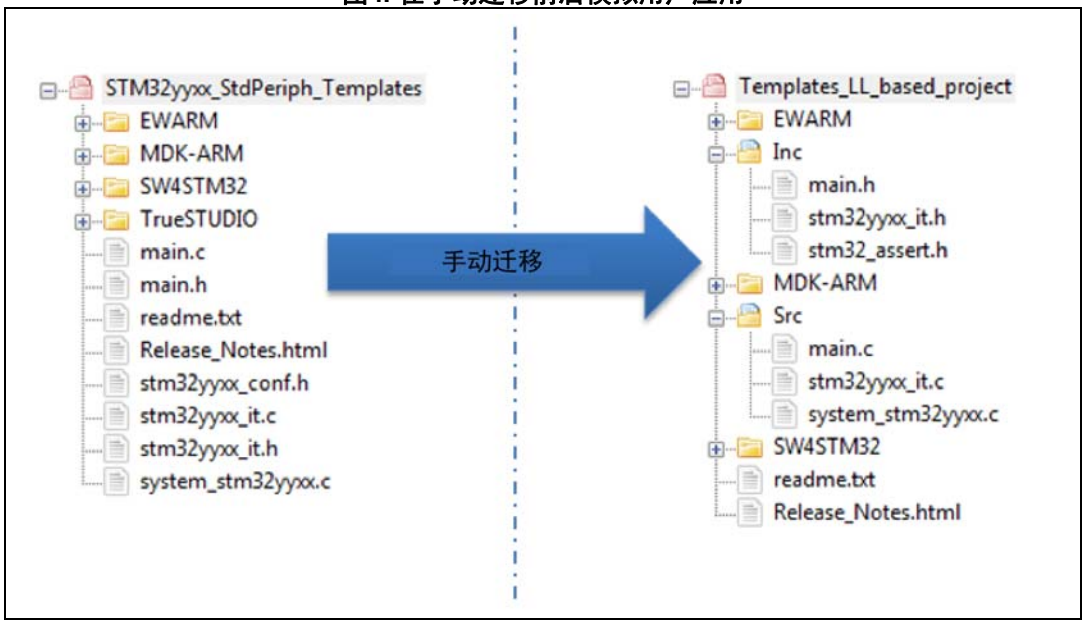
  较快速的方法是首先使用支持常用工具链的现成LL模板项目。它使用户省去了完成工具链配置步骤的时间，并消除了错误更新的风险。因此，用户可以使用LL API直接进入编码应用程序。

  STM32Cube固件包为给定STM32系列中的目标设备提供定制的LL模板项目。用户可以自由更新当前的硬件平台。

*注：* *STM32Cube附带丰富的示例集（总共70到90个不等），包括仅使用LL驱动程序或混合使用LL和HAL，用于演示如何使用不同的外设。这些示例通常与STM32 SPL提供的应用场景共享相同的应用场景，这使开发人员能够轻松快速地比较两种解决方案并开始使用STM32Cube LL。*

*图 4*说明了应用程序成功从基于STM32SPL模板项目迁移至STM32CubeTemplate_LL项目。

**图4. 在手动迁移前后模拟用户应用**



# 3 STM32 SPL至STM32Cube LL自动迁移

本节介绍使用SPL2LL-转换器迁移工具进行自动迁移的步骤。在本节中，用户将全面了解迁移工具及其特性，其产品系列和外设的覆盖范围，以及成功运行迁移工具的详细指南。

## 3.1 SPL2LL-转换器迁移工具说明

### 3.1.1 概述和特性

STM32 SPL2LL-转换器迁移工具是ST使用Perl编程语言开发的解决方案。此工具旨在将通过STM32SPL开发的现有应用程序升级到基于STM32CubeLL驱动程序的等价应用程序。使用此工具的过程涵盖了用户整个源文件的迁移。

首先从用户源文件开始，扫描每个文件以查找和提取STM32SPLAPI，然后在给定数据库中查找等价项，并基于STM32Cube LL驱动程序创建等价的用户源文件。

STM32 SPL2LL-转换器迁移工具具有以下特性：

- 支持STM32 SPL和STM32Cube LL支持的所有STM32系列
  - 从STM32 SPL驱动程序：STM32F0、STM32F10、STM32F2、STM32F30、STM32F37、STM32F4和STM32L1系列。
  - 至STM32Cube LL驱动程序：STM32F0、STM32F1、STM32F2、STM32F30、STM32F37、STM32F4、STM32F7、STM32L0、STM32L1和STM32L4系列。
- 支持LL框架涵盖的所有外设
  - 系统外设（RCC、PWR、FLASH、GPIO、EXTI、DMA和NVIC）
  - 模拟外设（ADC、DAC、COMP、OPAMP和CRS）
  - 定时器（RTC、TIM、LPTIM、HRTIM、IWDG和WWDG）
  - 通信外设（I2C、USART、LPUART和SPI/I2S）
  - 密码外设（CRC和RNG）
- 兼容Windows®、Linux®和MacOS®

该工具能够处理因可用外设版本变化引起的STM32系列之间的差异。这表明需要执行两个迁移用例，如*图 5*所示。

**图5. 自动迁移场景**

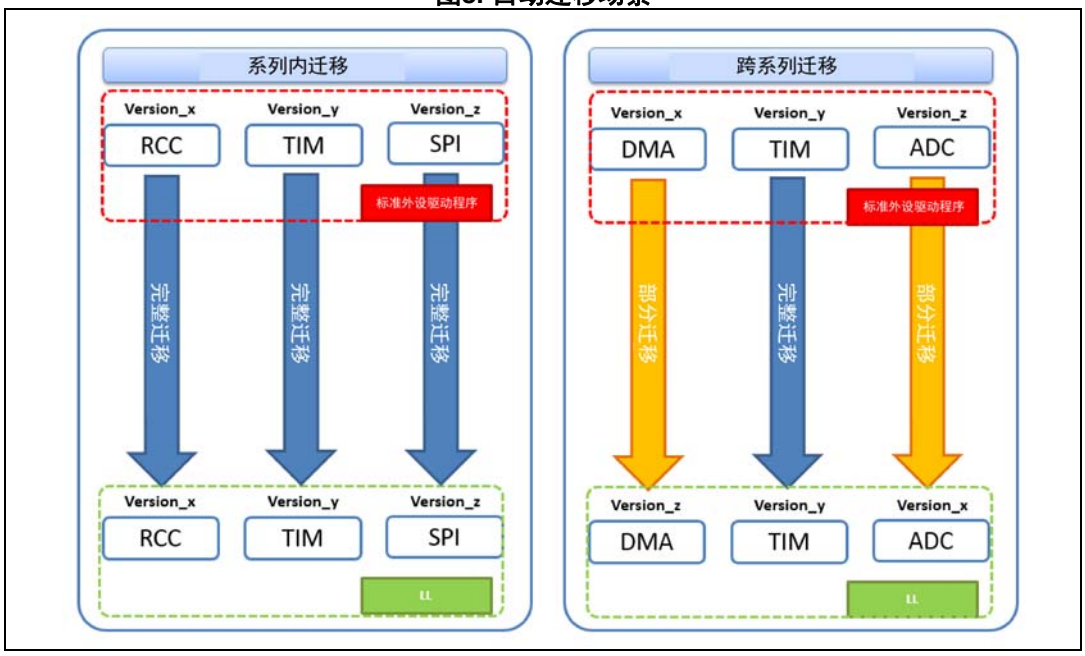


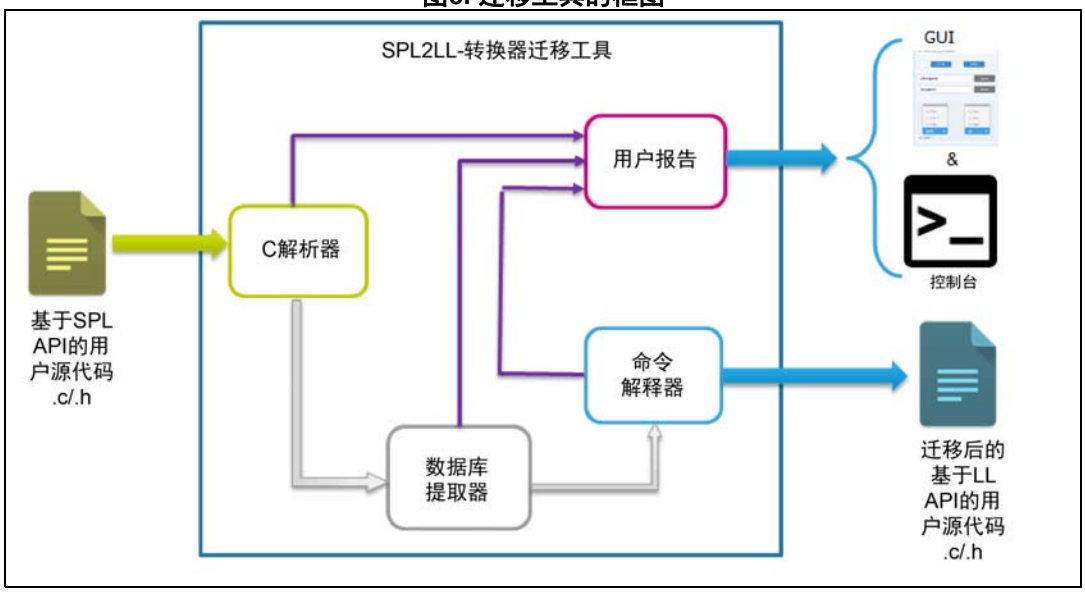

*图 5*提供有关两种迁移场景的信息：

- 系列内迁移：在同一系列中执行迁移（例如，从STM32SPLSTM32F4到STM32CubeLL STM32F4），其中，每个外设的API完整迁移到LL。
- 跨系列迁移：跨STM32系列执行迁移（例如，从STM32SPLSTM32F4到STM32CubeLL STM32L4）。在这种场景下，我们可以采用：
  - 部分代码迁移：该工具将源代码的一部分进行转换，由于在从一个外设版本到另一个外设版本可能缺少一些特征。迁移仅考虑常见特性。
  - 完整代码迁移：如果目标设备中的版本相同，则给定外设的所有STM32 SPL API都将进行完整转换。

### 3.1.2 SPL2LL-转换器迁移工具框图

该工具采用Perl编程语言开发，使用模块化架构。整个过程通过如*图 6* 所示的四个模块执行。

**图6. 迁移工具的框图**




- **C解析器**
  首先，该工具将用户的源文件作为输入。此模块允许逐行解析和扫描源文件，提取源代码。
  该工具能够区分用户的注释以及实际代码，从而将用户的注释原样复制到目标文件中。
  按4种类型，对基于STM32 SPL的代码进行排序和解释：
  - 文字：对STM32 SPL定义和结构命名进行分组。
  - 结构：对STM32 SPL驱动程序中的不同结构字段进行分组。
  - 函数：包括STM32 SPL驱动程序提供的所有函数。
  - 包含：识别代码中的文件包含。
- **数据库提取器**
  在此包内，STM32 SPL2LL-转换器迁移工具附带完整的一套XML文件，适用于STM32Cube LL驱动程序支持的STM32外设。该文件集构成XML数据库，迁移工具将此数据库视作源，从中提取LL等价，用于最初的代码解析。
  数据库的架构与先前C解析器模块解析数据的方式相匹配。
  提取器模块解析所有数据库并提取匹配的LL API，然后为结构和文字部分返回等效表达式；对于特性部分，STM32Cube LL函数使用指示迁移过程的相应命令进行命名。
- **命令解释器**
  一旦在数据库中发现了STM32SPLAPI，解释器模块便会执行STM32CubeLL等价API内置的命令，确保正确迁移。这些命令通常控制参数数量、顺序和默认值。

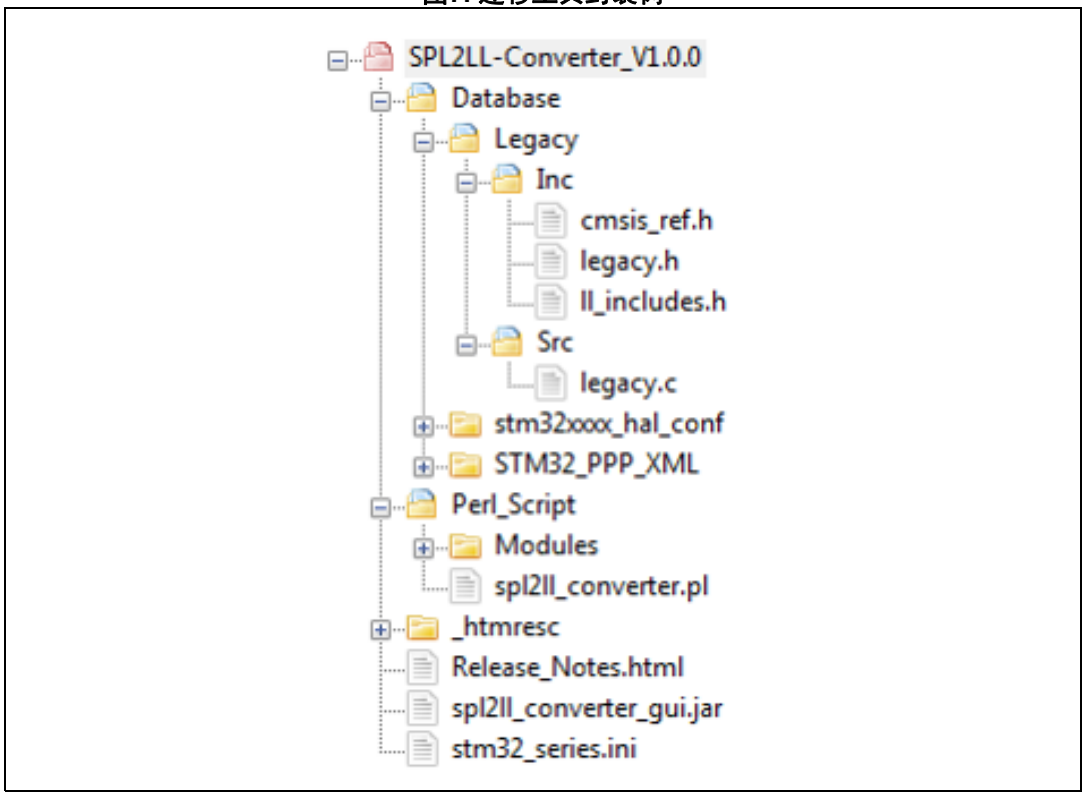
- **用户报告**
  迁移过程可能需要信息共享。为此，用户报告模块提供有关迁移进度的所有信息。报告涉及文件迁移状态、警告、错误和统计信息。
  在工具执行期间，报告可以作为控制台输出，也可生成为目标项目文件夹中可用的日志文件。

## 3.2 SPL2LL-转换器迁移工具使用指南

### 3.2.1 迁移工具包架构

SPL2LL-转换器迁移工具包含在一个包中，如*图 7*所示 。

**图7. 迁移工具封装树**



迁移包包含以下内容：

- 数据库文件夹：数据库所在的位置，包含：
  a) STM32_PPP_XML文件夹：包含一组XML文件，每个文件适用于一个支持的外设。
  b) stm32xxxx_hal_conf文件夹：包含最新STM32Cube_FW_YY包中提供的STM32xxxx HAL配置文件。需要替换STM32 SPL stm32yyxx_conf.h以保持与STM32Cube固件的兼容性。
  c) 历史文件夹：包含STM32 SPL API，其中，LL无直接等价，其内部复制了相同的STM32 SPL API实现。
- Perl脚本文件夹：其中存放了“spl2ll_converter.pl” Perl文件及其所需的组件。
- _htmresc文件夹：包含HTML文件中显示的STM32 SPL-STM32Cube LL API等价，适用于每个支持的外设。
- Release_Notes文件：简要介绍迁移工具和总体指南。
- spl2ll_converter_gui.jar：GUI应用程序的可执行文件。

### 3.2.2 SPL2LL-转换器迁移工具

在运行STM32 SPL2LL-转换器迁移工具之前，必须在主机上安装Perl。推荐使用ActiveState® Perl 5.24.1或更高版本。可以从ActiveState®网站（https://www.activestate.com）下载ActivePerl® 。

如要运行迁移工具，用户必须键入如下所述的命令：

perl spl2ll_converter.pl
--fsrc=<STM32_Source_Series>
--fdst=<STM32_Destination_Series>
--psrc=<STM32_Source_Directory>
--pdst=<STM32_Destination_Directory>

这些命令行参数表示源系列和目标系列，以及所需的路径：

--fsrc：STM32系列/设备源

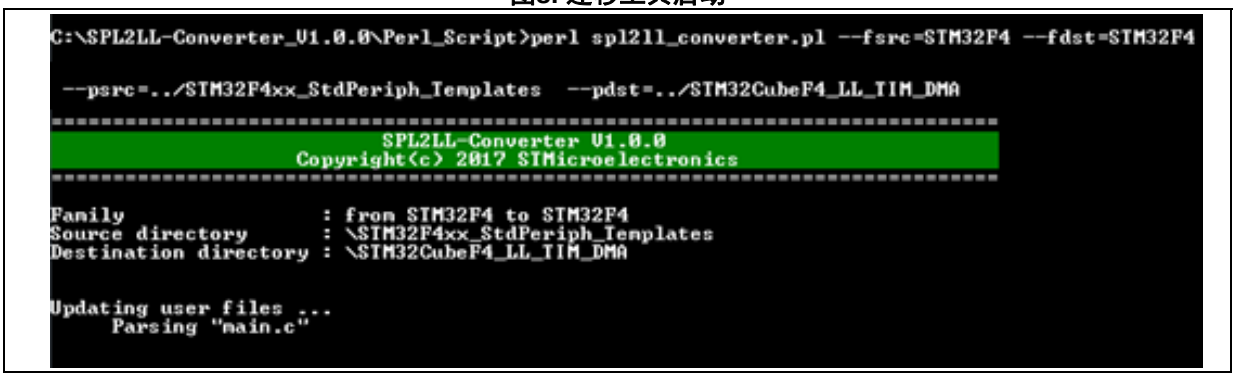
--fdst：STM32系列/设备目标位置

--psrc：基于STM32 SPL的源代码的用户目录路径

--pdst：输出代码的用户目录路径

*图 8*是终端屏幕截图，显示工具启动显示屏：

**图8. 迁移工具启动**



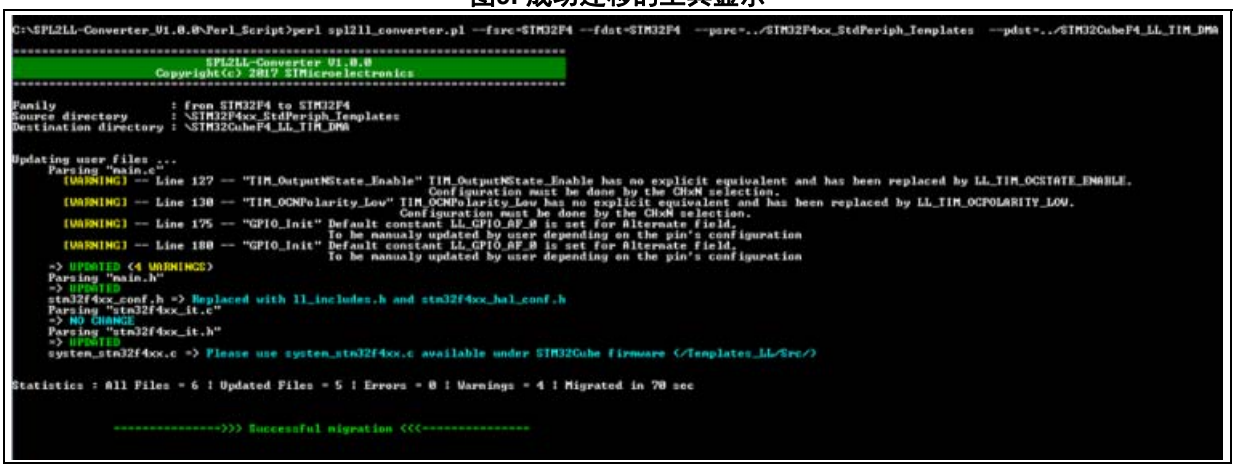

执行完成后，控制台输出将打印在终端上。它列出了工具的整体详细信息、迁移项目、有关用户文件的详细信息以及过程中的报告。

以下屏幕截图显示了上一个项目的两种不同的迁移场景：

- 第一个是从STM32F4到STM32F4的系列内迁移
- 第二个是从STM32F4到STM32F30的跨系列迁移。

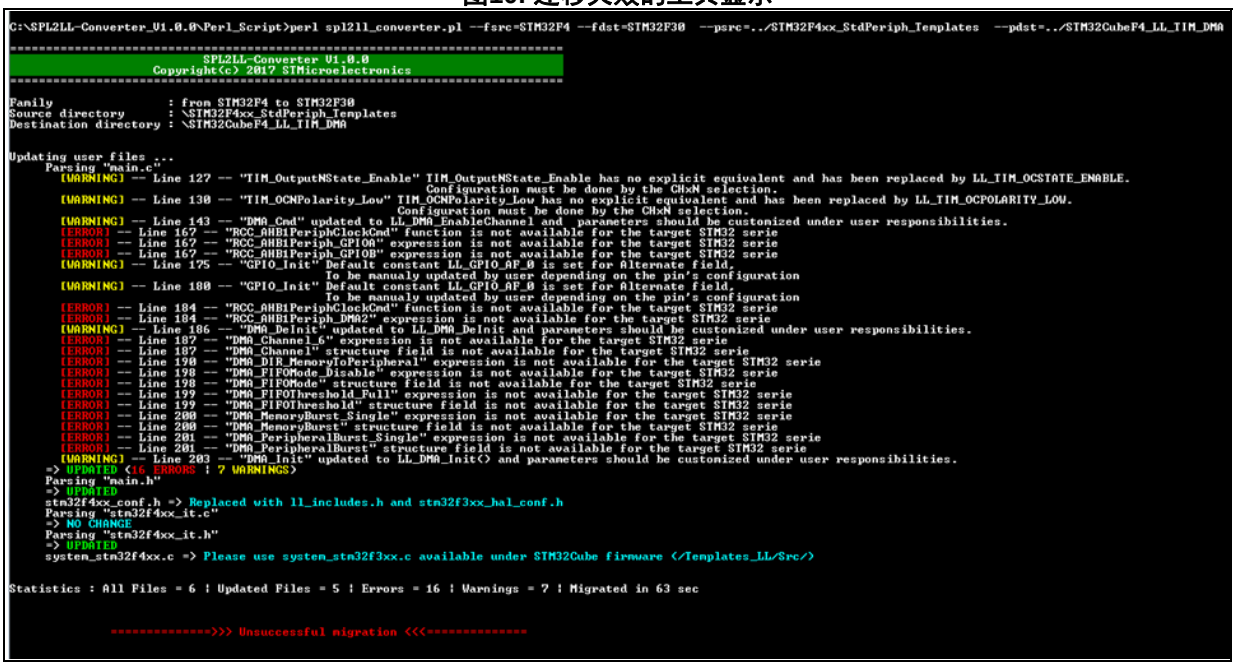
屏幕截图突出显示了迁移过程中用户显示屏的不同部分：

**图9. 成功迁移的工具显示**




1. 工具标题：显示工具标题和当前版本
2. 迁移参数：源系列和目标系列，以及源路径和目标路径
3. 用户文件状态：状态可以是“UPDATED”（已更新）、“NO CHANGE”（无修改），对于某些文件也可以是特定的用户消息

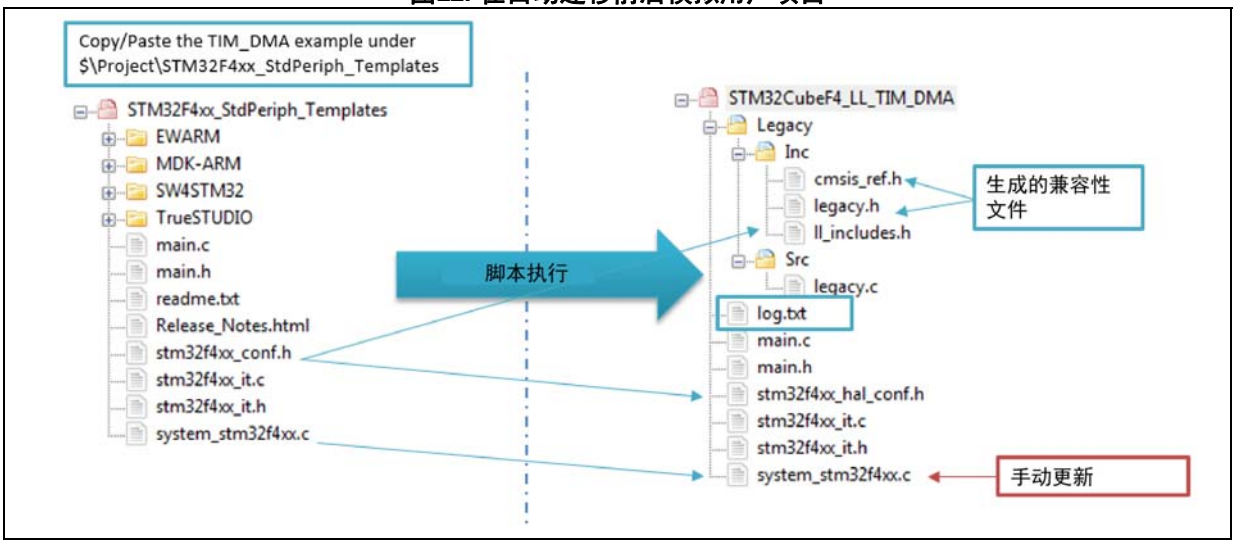
**图10. 迁移失败的工具显示**




4. 工具标题：显示工具标题和当前版本
5. 迁移参数：源系列和目标系列，以及源路径和目标路径
6. 用户文件状态：状态可以是“UPDATED”（已更新）、“NO CHANGE”（无修改），对于某些文件也可以是特定的用户消息

*图 11*是在STM32F4 StdPeriph_Lib TIM_DMA示例上应用的迁移过程示例：

**图11. 在自动迁移前后模拟用户项目**



执行后，该工具保持与初始项目相同的整体层次结构并应用一些更新。*图 11*突出显示这些更新：

- 只有源文件(.h/.c)才会迁移到新的LL项目中。
- 创建“Legacy”文件夹，其中包含迁移后API所需的、但LL不支持的源文件。“legacy.h/.c”表示STM32 SPL代码，“ll_includes.h”是迁移后“stm32yyxx_conf.h”等价的一部分，“cmsis_ref.h”包含所有STM32系列的CMSIS包含文件，是“legacy.h”的必要文件。
- “stm32yyxx_conf.h”的迁移导致其替换为“stm32yyxx_hal_conf.h”和“ll_includes”，以兼容STM32Cube。
- 创建“log.txt”文件，在迁移过程中显示全局报告。

在需要手动干预的情况下，用户可以检查生成的日志文件“log.txt”，在该文件中，用户可以找到打印的警告，用于指导调整LL源文件。

但是，如果由于目标设备的不可用特性而导致迁移不成功，用户则必须纠正报告的错误。

因此，必须结合基于STM32 SPL的应用程序，以兼容目标设备当前的特性。

### 3.2.3 用户应用程序迁移步骤及初始应用环境

本节介绍第一种方法的自动迁移，详情请见*第 2 节：STM32SPL至STM32CubeLL手动迁移*。

#### 先决条件

用户需要下载最新版本的目标系列STM32Cube包，在此可以找到LL和CMSIS驱动程序。

基于STM32 SPL的初始项目不应含有错误，应保证功能齐全，以避免不必要的迁移问题，在部分迁移时更易于进行手动更新。

#### 迁移指南

由于该工具仅迁移源文件，与所使用的工具链无关。因此，要求用户：

- **工具执行（自动步骤）**
  设置迁移环境后，开始该过程，如下所示：

1. 启动终端并指向工具位置。
2. 运行SPL2LL-转换器迁移工具。

- **用户项目定制（手动步骤）**

1. 使用STM32Cube_FW_YY包下的最新版本来更新旧的CMSIS文件夹。
2. 添加STM32xx_HAL_Driver文件夹，该文件夹含有同一封装内可用的LL/HAL源/头文件。
3. 创建新的目标项目，包括所需的LL/CMSIS驱动程序。
4. 设置配置（预处理器、启动文件、MCU选择、调试器）。
5. 更新“system_stm32yyxx.c”，将其从目标STM32Cube固件包中的“Templates_LL”文件夹中提取出来。

*注： 用户应在迁移后的main.c文件上手动实现SystemClock_Config() API，确保再现已在system_stm32yyxx.c中执行的系统时钟配置。*

6. 在项目中添加用户源文件。
7. 在生成的$Legacy/Src文件夹下添加“legacy.c”文件。

### 3.2.4 使用可用LL模板的用户应用程序迁移步骤

本节介绍第二种方法的自动迁移，详情请见*第 2节：STM32SPL至STM32CubeLL手动迁移*。

用户可以选择将STM32Cube LL模板用作目标项目。使用此方法，即可使用模板项目。

无需更新CMSIS文件或添加底层驱动程序，因为模板项目默认会引用它们。

因此，用户必须：

- **工具执行（自动步骤）**

1. 启动终端并指向工具位置。
2. 运行SPL2LL-转换器迁移工具。

- **用户项目定制（手动步骤）**

3. 在Inc和Src文件夹之间整理迁移后的源文件。
4. 在项目中添加迁移后的源文件。
5. 在生成的$Legacy/Src文件夹下添加“legacy.c”文件。

### 3.2.5 SPL2LL-转换器迁移工具限制

目前，SPL2LL-转换器迁移工具存在一些限制，这些限制依赖于STM32系列中的API实现和用户的编码方法：

- 该工具考虑超集设备中可用的LL API等价。
- 不支持多调用API：
  Function_X (arg_0, arg_1, function_y(arg_a, arg_b), arg_2) ;
- 将它们作为API参数调用时，不支持别名：
  #define ALIAS_0 STM32SPL_Literal_0
  Function_X(ALIAS_0) ;
  必须直接调用已定义的STM32 SPL文字，否则不考虑该行。
  Function_X(STM32SPL_literal_0);
- 部分函数不支持调用多文字作为参数：
  Function_X(LITERAL_0| LITERAL1 | LITERAL2);
  必须通过函数调用命令仅调用一个定义的文字，否则不考虑该行。
  Function_X(LITERAL_0);
  Function_X(LITERAL_1);
  Function_X(LITERAL_2);
- 对于跨系列迁移，一旦STM32外设版本之间的函数原型不完全兼容，则此函数将被视为特定功能。

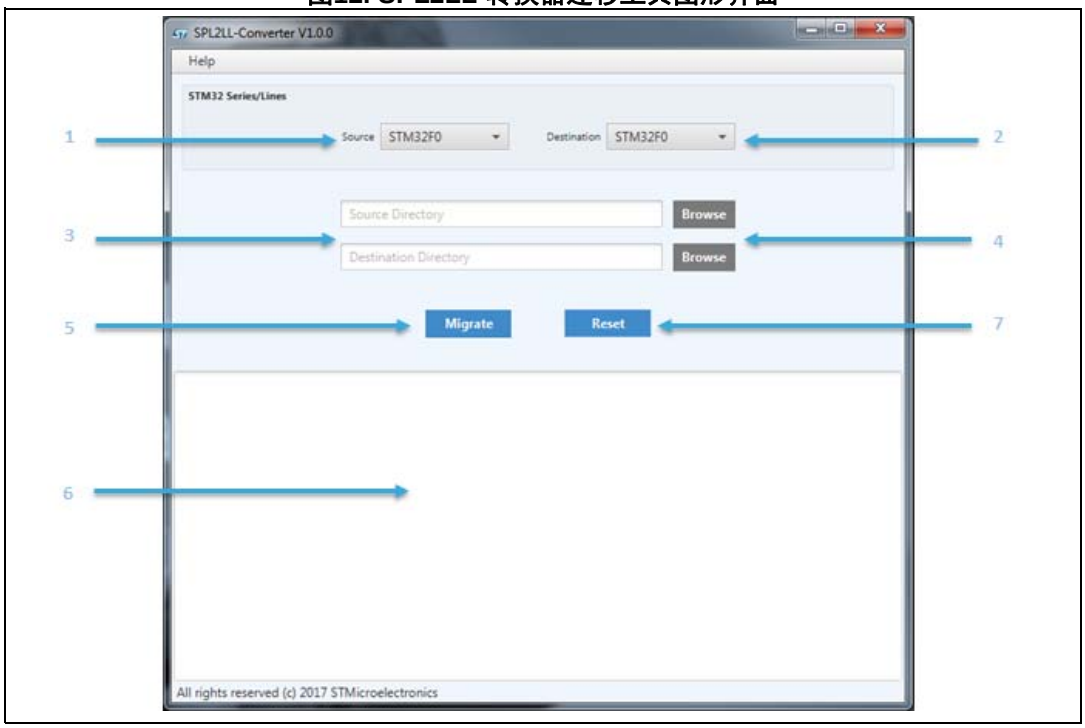
强烈建议下载最新版本的工具，以便用于任何潜在的错误修复和行为增强。

### 3.2.6 GUI应用程序

GUI应用程序，也可以通过使用提供的迁移工具进行自动迁移。

用户可以在工具包中找到此应用程序的可执行文件，名称为“spl2ll_converter_gui.jar”，如*图 12* 所示。

**图12. SPL2LL-转换器迁移工具图形界面**




此应用程序帮助不熟悉命令行的用户能够更快、更轻松地启动迁移进程。兼容Windows、Linux和macOS操作系统。

该界面描述如下：

1. 支持STM32 SPL的STM32产品系列/产品线列表（源）
2. 支持STM32Cube LL的STM32产品系列/产品线列表（目标）
3. 源和目标项目路径
4. 浏览路径按钮
5. 迁移开始按钮：调用迁移工具执行
6. 日志窗口：显示迁移进度和最终状态。
7. 复位按钮：重新初始化所有GUI控件和字段

在启动应用程序之前，您需要安装1.8.0或更高版本的Java运行时环境。

用户可以从Java下载网页获取最新版本。

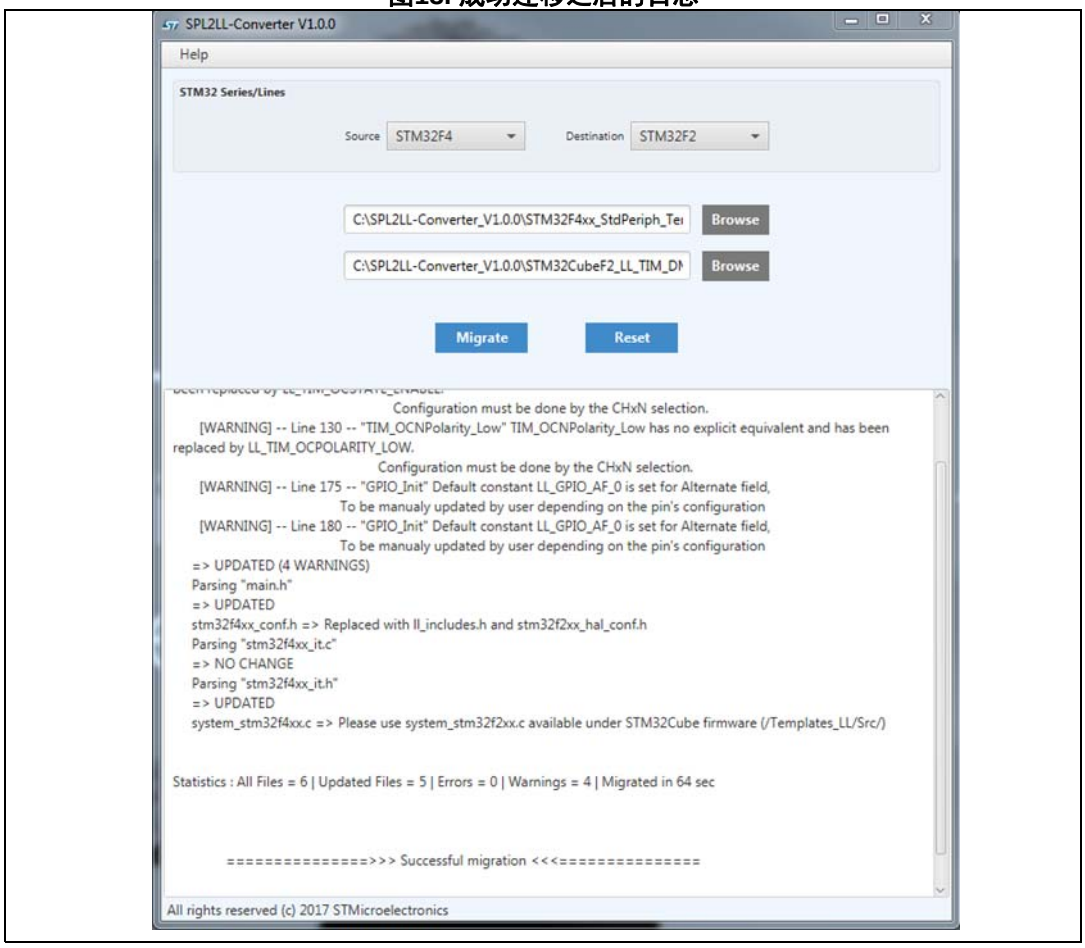
如要使用该应用程序，您需要：

- 运行“spl2ll_converter_gui.jar”。
- 选择需要互相迁移的STM32源和目标产品系列/产品线。
- 选择初始STM32 SPL应用程序的源目录。
- 选择目标目录，用于查找迁移后的源文件。
- 点击“迁移”按钮，等待迁移进程完成。

在整个迁移过程中，您可以在日志窗口中跟踪进度。完成后，将显示整体状态并生成日志文件。

下图显示了成功迁移后应用程序的外观：

**图13. 成功迁移之后的日志**



# 4 版本历史

表6. 文档版本历史

| 日期 | 版本 | 变更 |
|---|---|---|
| 2017年7月13日 | 1 | 初始版本。 |
| 2017年8月30日 | 2 | 删除封面的*表1：适用产品*。 |

表7. 中文文档版本历史

| 日期 | 版本 | 变更 |
|---|---|---|
| 2018年12月14日 | 1 | 中文初始版本。 |

**重要通知 - 请仔细阅读**

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

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

ST 不对任何知识产权进行任何明示或默示的授权或许可。

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

ST 和 ST 徽标是 ST 的商标。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代本文档所有早期版本中提供的信息。本文档的中文版本为英文版本的翻译件，仅供参考之用；若中文版本与英文版本有任何冲突或不一致，则以英文版本为准。