**本文档配合源代码看.**

core/src/main.c只有一个头文件, 就是main.h, 里面只有针对自己使能的GPIO的Label与相应的Bank和GPIO Pin Number之间的对应关系. Main.h包含了**stm32f4xx\_hal.h**, 这是个比较复杂的头文件. 自己定义的函数文件如果要用到某个peripheral的HAL函数，也需要包含这个头文件.

void SystemClock\_Config(void)函数的declaration就在main.c的函数体里, 因为这个函数只会在main.c里用到, 所以其declaration和definition都在main.c中; 这个函数的作用完全是实现在.ioc文件里的clock configuration里的内容.

static void MX\_GPIO\_Init(void) 函数的declaration也在main.c的函数体里, 因为这个函数只会在main.c里用到, 所以其declaration和definition都在main.c中; 这个函数的作用是使能用到的GPIO的Bank CLK, 然后往这些GPIO写一个初始值, 然后再Init GPIO的数据结构.

**core/src/main.c – int main(void)里除了上面两个函数, 还有个在最一开始的HAL\_Init()**, 其他的就是那个while(1)了; **中断处理函数文件stm32f4xx\_it.c里跟SysTick有关的函数是**:

**void** **SysTick\_Handler**(**void**)

{

/\* USER CODE BEGIN SysTick\_IRQn 0 \*/

/\* USER CODE END SysTick\_IRQn 0 \*/

**HAL\_IncTick()**;

/\* USER CODE BEGIN SysTick\_IRQn 1 \*/

/\* USER CODE END SysTick\_IRQn 1 \*/

}

**stm32f4xx\_it.c**包含的头文件是main.h和**stm32f4xx\_it.h**. 后者没有包含任何头文件, 只是申明了一堆中断处理函数的prototype.

* 现在看**HAL\_Init()**和**HAL\_IncTick()**这两个函数.

这两个函数都declare/define在**stm32f4xx\_hal.h**/**stm32f4xx\_hal.c**中.

1. **先看HAL\_Init()**

/\*\*

\* @brief This function is used to initialize the HAL Library; **it must be the first**

\* **instruction to be executed in the main program** (before to call any other

\* HAL function), it performs the following:

\* Configure the Flash prefetch, instruction and Data caches.

\* **Configures the SysTick to generate an interrupt each 1 millisecond**,

\* **which is clocked by the HSI (at this stage, the clock is not yet**

\* **configured and thus the system is running from the internal HSI at 16 MHz)**.

\* Set NVIC Group Priority to 4.

\* Calls the HAL\_MspInit() callback function defined in user file

\* "stm32f4xx\_hal\_msp.c" to do the global low level hardware initialization

\* @note **SysTick is used as time base for the HAL\_Delay() function, the application**

\* **need to ensure that the SysTick time base is always set to 1 millisecond**

\* **to have correct HAL operation**.

\* @retval HAL status

\*/

**HAL\_StatusTypeDef** **HAL\_Init**(**void**)

{

/\* Configure Flash prefetch, Instruction cache, Data cache \*/

**#if** (INSTRUCTION\_CACHE\_ENABLE != 0U)

\_\_HAL\_FLASH\_INSTRUCTION\_CACHE\_ENABLE();

**#endif** /\* INSTRUCTION\_CACHE\_ENABLE \*/

**#if** (DATA\_CACHE\_ENABLE != 0U)

\_\_HAL\_FLASH\_DATA\_CACHE\_ENABLE();

**#endif** /\* DATA\_CACHE\_ENABLE \*/

**#if** (PREFETCH\_ENABLE != 0U)

\_\_HAL\_FLASH\_PREFETCH\_BUFFER\_ENABLE();

**#endif** /\* PREFETCH\_ENABLE \*/

/\* Set Interrupt Group Priority \*/

HAL\_NVIC\_SetPriorityGrouping(NVIC\_PRIORITYGROUP\_4);

**/\* Use systick as time base source and configure 1ms tick (default clock after Reset is HSI) \*/**

**HAL\_InitTick(TICK\_INT\_PRIORITY);**

/\* Init the low level hardware \*/

**HAL\_MspInit();**

/\* Return function status \*/

**return** *HAL\_OK*;

}

**HAL\_StatusTypeDef**结构体定义在**stm32f4xx\_hal\_conf.h**中, 而这个头文件是被stm32f4xx\_hal.h包含的.

**typedef** **enum**

{

*HAL\_OK* = 0x00U,

*HAL\_ERROR* = 0x01U,

*HAL\_BUSY* = 0x02U

*HAL\_TIMEOUT* = 0x03U

} **HAL\_StatusTypeDef**;

* 1. **看HAL\_InitTick(TICK\_INT\_PRIORITY)**

#define TICK\_INT\_PRIORITY ((uint32\_t)0U)

\_\_weak HAL\_StatusTypeDef **HAL\_InitTick**(uint32\_t TickPriority)

{

/\* Configure the SysTick to have interrupt in 1ms time basis\*/

**if** (**HAL\_SYSTICK\_Config(SystemCoreClock / (1000U / uwTickFreq)**) > 0U)

{

**return** *HAL\_ERROR*;

}

/\* Configure the SysTick IRQ priority \*/

**if** (TickPriority < (1UL << \_\_NVIC\_PRIO\_BITS))

{

HAL\_NVIC\_SetPriority(*SysTick\_IRQn*, TickPriority, 0U);

uwTickPrio = TickPriority;

}

**else**

{

**return** *HAL\_ERROR*;

}

/\* Return function status \*/

**return** *HAL\_OK*;

}

这个函数的前面的注释里有一句话：

**Care must be taken if HAL\_Delay() is called from a peripheral ISR process, the SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt**. Otherwise, the caller ISR process will be blocked. The function is declared as **\_\_weak** to be overwritten in case of another implementation in user file.

**1.1.1 看HAL\_SYSTICK\_Config(SystemCoreClock / (1000U / uwTickFreq)**

**/\* This variable is updated in three ways:**

**1) by calling CMSIS function SystemCoreClockUpdate()**

**2) by calling HAL API function HAL\_RCC\_GetHCLKFreq()**

**3) each time HAL\_RCC\_ClockConfig() is called to configure the system clock frequency**

**Note: If you use this function to configure the system clock; then there is no need to call the 2 first functions listed above, since SystemCoreClock variable is updated automatically.**

**\*/**

uint32\_t **SystemCoreClock = 16000000**;

在**stm32f4xx\_hal.h的开头把那三个变量声明为extern, 这样别的文件只需要include stm32f4xx\_hal.h就可以access那三个变量的本体：**

/\*\* @addtogroup HAL\_Exported\_Variables

\*/

**extern** \_\_IO uint32\_t **uwTick**;

**extern** uint32\_t **uwTickPrio**;

**extern** HAL\_TickFreqTypeDef **uwTickFreq**;

还定义了：

/\*\* @defgroup HAL\_TICK\_FREQ Tick Frequency

\* @{

\*/

**typedef** **enum**

{

*HAL\_TICK\_FREQ\_10HZ* = 100U,

*HAL\_TICK\_FREQ\_100HZ* = 10U,

*HAL\_TICK\_FREQ\_1KHZ* = 1U,

*HAL\_TICK\_FREQ\_DEFAULT*  = *HAL\_TICK\_FREQ\_1KHZ*

} HAL\_TickFreqTypeDef;

在**stm32f4xx\_hal.c的开头定义了这三个变量本体：**

\_\_IO uint32\_t **uwTick**;

uint32\_t **uwTickPrio** = (1UL << \_\_NVIC\_PRIO\_BITS); /\* Invalid PRIO \*/

HAL\_TickFreqTypeDef **uwTickFreq** = *HAL\_TICK\_FREQ\_DEFAULT*; /\* 1KHz \*/

**HAL\_SYSTICK\_Config(SystemCoreClock / (1000U / uwTickFreq)相当于HAL\_SYSTICK\_Config(16000000 / (1000U / uwTickFreq);**

/\*\*

\* @brief Initializes the System Timer and its interrupt, and starts the System Tick Timer.

\* Counter in free running mode to generate periodic interrupts.

\* @param TicksNumb Specifies the ticks Number of ticks between two interrupts.

\* @retval status: 0 Function succeeded.

\* 1 Function failed.

\*/

uint32\_t **HAL\_SYSTICK\_Config**(uint32\_t TicksNumb)

{

**return** SysTick\_Config(TicksNumb);

}

\_\_STATIC\_INLINE uint32\_t **SysTick\_Config**(uint32\_t ticks)

{

**if** ((ticks - 1UL) > SysTick\_LOAD\_RELOAD\_Msk)

{

**return** (1UL); /\* Reload value impossible \*/

}

SysTick->LOAD = (uint32\_t)(ticks - 1UL); /\* set reload register \*/

NVIC\_SetPriority (*SysTick\_IRQn*, (1UL << \_\_NVIC\_PRIO\_BITS) - 1UL); /\* set Priority for Systick Interrupt \*/

SysTick->VAL = 0UL; /\* Load the SysTick Counter Value \*/

SysTick->CTRL = SysTick\_CTRL\_CLKSOURCE\_Msk |

SysTick\_CTRL\_TICKINT\_Msk |

**SysTick\_CTRL\_ENABLE\_Msk**; **/\* Enable SysTick IRQ and SysTick Timer \*/**

**return** (0UL); /\* Function successful \*/

}

**SysTick\_Config()**定义在core\_cm4.h中;

#define SysTick\_LOAD\_RELOAD\_Msk (0xFFFFFFUL) **/\* 24位的寄存器 \*/**

/\*\*

\* @brief STM32F4XX Interrupt Number Definition, according to the selected device

\* in @ref Library\_configuration\_section

\*/

**typedef** **enum**

{

/\*\*\*\*\*\* Cortex-M4 Processor Exceptions Numbers \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

NonMaskableInt\_IRQn = -14, /\*!< 2 Non Maskable Interrupt \*/

MemoryManagement\_IRQn = -12, /\*!< 4 Cortex-M4 Memory Management Interrupt \*/

BusFault\_IRQn = -11, /\*!< 5 Cortex-M4 Bus Fault Interrupt \*/

UsageFault\_IRQn = -10, /\*!< 6 Cortex-M4 Usage Fault Interrupt \*/

SVCall\_IRQn = -5, /\*!< 11 Cortex-M4 SV Call Interrupt \*/

DebugMonitor\_IRQn = -4, /\*!< 12 Cortex-M4 Debug Monitor Interrupt \*/

PendSV\_IRQn = -2, /\*!< 14 Cortex-M4 Pend SV Interrupt \*/

**SysTick\_IRQn** = -1, /\*!< 15 Cortex-M4 System Tick Interrupt \*/

/\*\*\*\*\*\* STM32 specific Interrupt Numbers \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

WWDG\_IRQn = 0, /\*!< Window WatchDog Interrupt \*/

PVD\_IRQn = 1, /\*!< PVD through EXTI Line detection Interrupt \*/

TAMP\_STAMP\_IRQn = 2, /\*!< Tamper and TimeStamp interrupts through the EXTI line \*/

RTC\_WKUP\_IRQn = 3, /\*!< RTC Wakeup interrupt through the EXTI line \*/

FLASH\_IRQn = 4, /\*!< FLASH global Interrupt \*/

RCC\_IRQn = 5, /\*!< RCC global Interrupt \*/

EXTI0\_IRQn = 6, /\*!< EXTI Line0 Interrupt \*/

EXTI1\_IRQn = 7, /\*!< EXTI Line1 Interrupt \*/

EXTI2\_IRQn = 8, /\*!< EXTI Line2 Interrupt \*/

EXTI3\_IRQn = 9, /\*!< EXTI Line3 Interrupt \*/

EXTI4\_IRQn = 10, /\*!< EXTI Line4 Interrupt \*/

DMA1\_Stream0\_IRQn = 11, /\*!< DMA1 Stream 0 global Interrupt \*/

DMA1\_Stream1\_IRQn = 12, /\*!< DMA1 Stream 1 global Interrupt \*/

DMA1\_Stream2\_IRQn = 13, /\*!< DMA1 Stream 2 global Interrupt \*/

DMA1\_Stream3\_IRQn = 14, /\*!< DMA1 Stream 3 global Interrupt \*/

DMA1\_Stream4\_IRQn = 15, /\*!< DMA1 Stream 4 global Interrupt \*/

DMA1\_Stream5\_IRQn = 16, /\*!< DMA1 Stream 5 global Interrupt \*/

……

DCMI\_IRQn = 78, /\*!< DCMI global interrupt \*/

RNG\_IRQn = 80, /\*!< RNG global Interrupt \*/

FPU\_IRQn = 81 /\*!< FPU global interrupt \*/

} IRQn\_Type;

这里有一个很重要的地方：

**/\* This variable is updated in three ways:**

**1) by calling CMSIS function SystemCoreClockUpdate()**

**2) by calling HAL API function HAL\_RCC\_GetHCLKFreq()**

**3) each time HAL\_RCC\_ClockConfig() is called to configure the system clock frequency**

**Note: If you use this function to configure the system clock; then there is no need to call the 2 first functions listed above, since SystemCoreClock variable is updated automatically.**

**\*/**

uint32\_t **SystemCoreClock = 16000000**;

**在main()里, 紧接着HAL\_Init()的就是SystemClock\_Config(), 在这个函数里的最后, HAL\_RCC\_ClockConfig()被调用了.**

**总结：HAL\_Init()把SysTick配置成了1ms中断一次(中断由SysTick倒数到0时产生), 并启动了SysTick;**

1. **再看HAL\_IncTick()**

**它的定义在stm32f4xx\_hal.c**中:

/\*\*

\* @brief This function is called to increment a global variable "uwTick"

\* used as application time base.

\* @note In the default implementation, this variable is incremented each 1ms

\* in SysTick ISR.

\* @note This function is declared as \_\_weak to be overwritten in case of other

\* implementations in user file.

\* @retval None

\*/

\_\_weak **void** HAL\_IncTick(**void**)

{

uwTick += uwTickFreq;

}

**所以这个中断函数唯一做的事就是更新一下uwTick这个全局变量, 每次加1. 所以uwTick实际上是系统自运行以来经过的ms数.**

所以下面的函数也就不言自明了:

\_\_weak uint32\_t **HAL\_GetTick**(**void**)

{

**return** uwTick;

}

注意, \_\_weak void HAL\_Delay(uint32\_t Delay)是以ms为单位的;