

STM32Cube 官方例程学习指南

STM32CubeMX 是 ST 官方提供的一个代码生成工具。使用该工具，通过图形化的配置方法，就能快速生成 STM32 的各种片上外设的初始化代码。CubeMX 生成的软件工程使用 HAL 库，HAL 库是 ST 以后主推的外设驱动库。另外 CubeMX 还提供了 FATFS、FreeRTOS、LWIP、USB 库等中间件的支持，配置之后生成软件工程，工程文件就包含了相应代码。

本文档以 STM32F4 系列为例，简要地分析官方提供的 Cube 例程。希望能够帮助 CubeMX 初学者快速掌握 STM32 的常用外设使用方法。文档不求全面，只讲常用的外设，对不常用的只进行概况性地描述。同时，文档只对例程进行直接分析，不对其他文件进行详述。

第一部分 准备工作

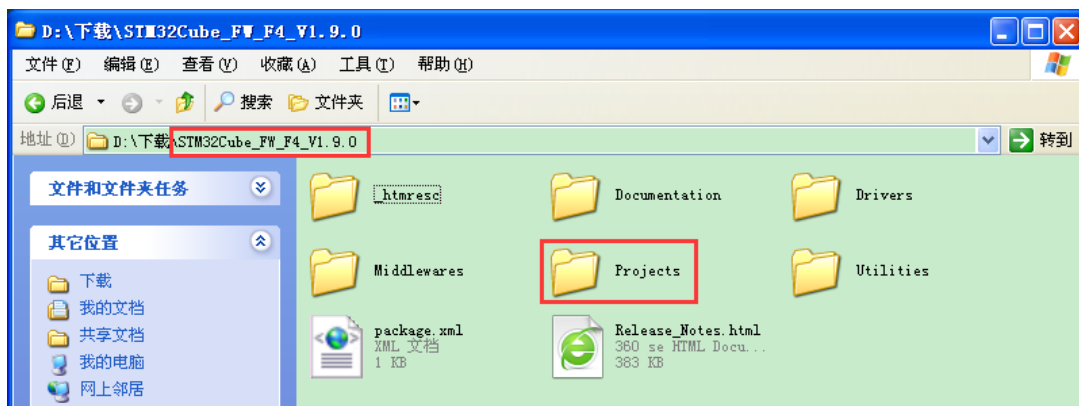
首先是下载 STM32CubeF4 支持包，可以到与非网 ST 社区搜索 STM32CubeF4，然后下载

The screenshot shows the STM32CubeF4 download page on the 23.Nucleo board page. The page title is "STM32CubeF4". The upload time is "2016-03-20". The description is "针对STM32F4系列的STM32Cube软件包（HAL驱动，USB，以太网，文件系统）(STM32Cube firmware for STM32 F4 series (HAL drivers, USB, Ethernet, File System, ...))".

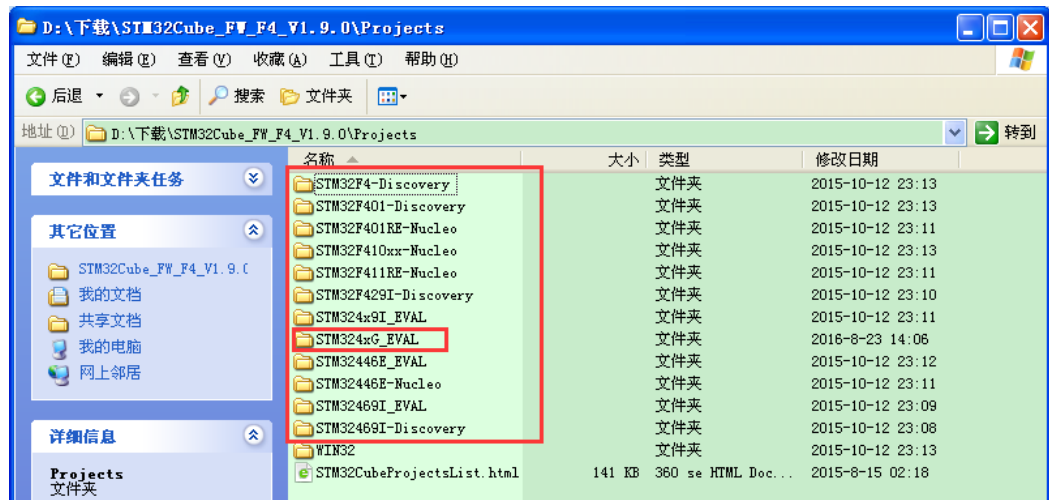
类型	文档标题	格式	版本	文件大小	下载次数
英文文档	DM00107720	pdf	6	350.32KB	3670
附件	STM32CubeF4		1.13.0	0	356

当前版本已经更新到 V1.13.0。点击附件中的 STM32CubeF4，转到下载链接地址。附件大小 300M 左右。本人当前使用的是 V1.9.0 版本的，例程相差不大，后面就用 V1.9.0 版本的例程进行分析。

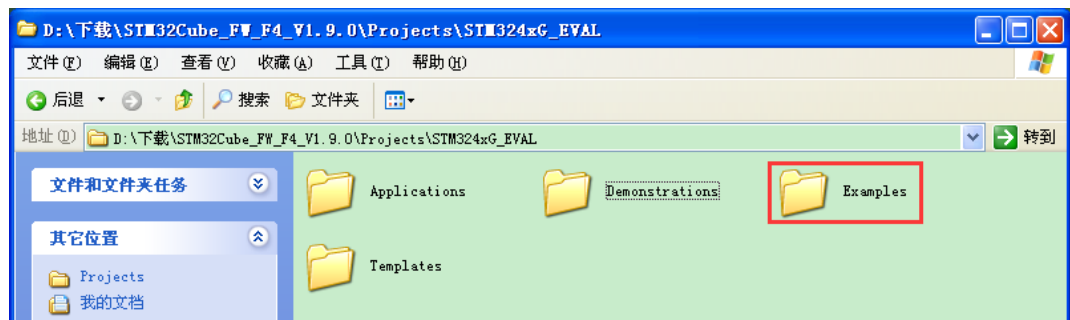
下载后解压，得到如下图的文件，其中例程放在 Projects 文件夹中：



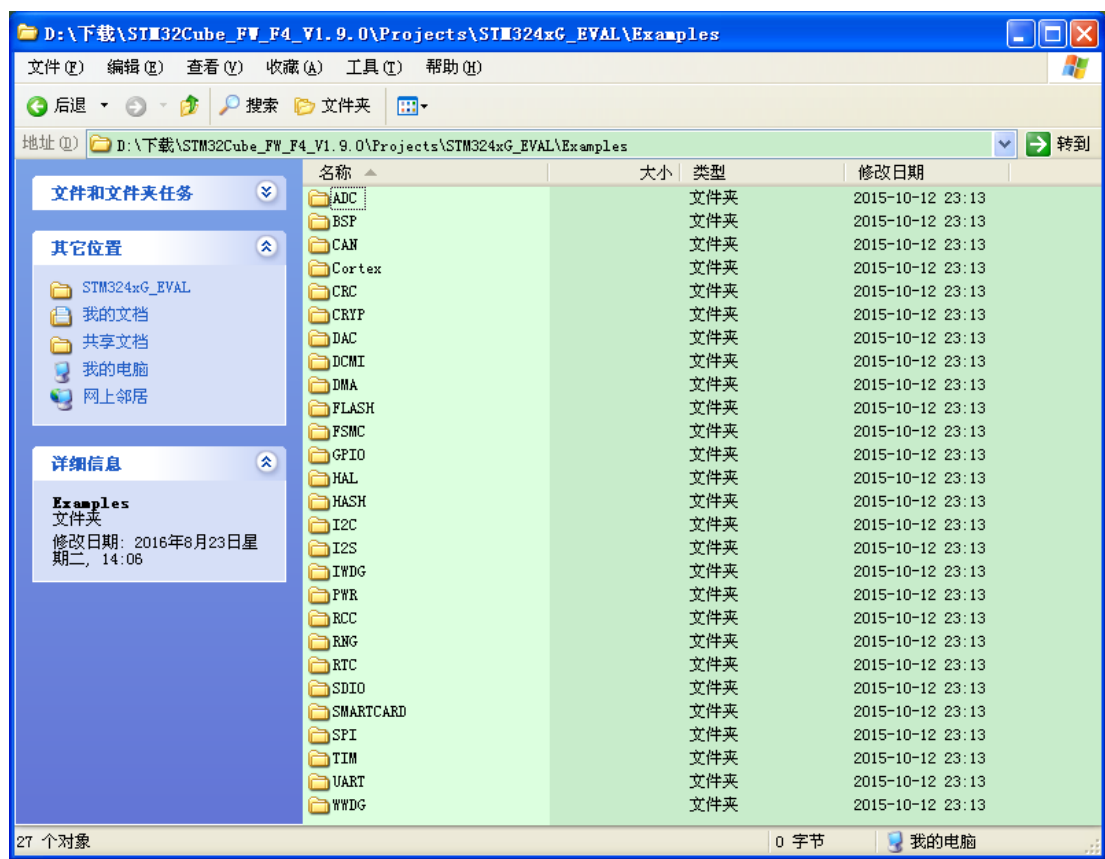
打开 Projects 文件夹，可以看到前 12 个文件夹分别官方提供的 12 款评估板，后面我们仅以 STM324xG_EVAL 评估板的例程为讲解内容。



STM324xG_EVAL 文件夹中，Examples 文件夹存放的就是片上外设的使用例程。(Applications 文件夹是 STM324xG_EVAL 相关的一些高级应用例程，如 FreeRTOS、FatFs、LwIP、USB 等，有一定基础之后可以学习这里面的内容。本文不作分析。)



Examples 文件夹提供了 27 个外设对应文件夹，每个文件夹包含若干个例程，后面将对常用的外设例程(不是全部)进行简要分析。

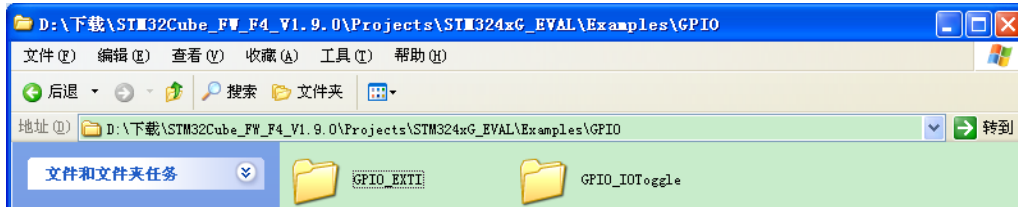


第二部分 例程分析

下面将挑选常用外设的例程进行分析，顺序是从简单的到复杂的。

第一章：GPIO

GPIO 共有两个例程：外部中断和 IO 翻转。



1. GPIO_IOToggle

打开...\GPIO_IOToggle\MDK-ARM 文件夹下的 MDK 工程，打开 main.c 文件。

如 main.c 文件开头的描述，本例程描述如何配置 GPIO 和通过 HAL API 函数使用 GPIO。

```
7  * @brief This example describes how to configure and use GPIOs through
8  * the STM32F4xx HAL API.
9  *****
```

看 main 函数：

```
66 int main(void)
67 {
68     /* STM32F4xx HAL library initialization:
69      - Configure the Flash prefetch, instruction cache and DMA2 memory
70      - Configure the SysTick to generate an interrupt
71      - Set NVIC Group Priority to 4
72      - Global MSP (MCU Support Package) initialization
73     */
74     HAL_Init();
75
76     /* Configure the system clock to 168 MHz */
77     SystemClock_Config();
78
79     /* -1- Enable GPIOA, GPIOB and GPIOI Clock (HSE configured) */
80     __HAL_RCC_GPIOA_CLK_ENABLE();
81     __HAL_RCC_GPIOB_CLK_ENABLE();
82     __HAL_RCC_GPIOI_CLK_ENABLE();
83
84     /* -2- Configure PG.6, PG.8, PI.9 and PC.7 IOs in output push-pull mode to
85      drive external LEDs */
86     GPIO_InitStruct.Pin = (GPIO_PIN_6 | GPIO_PIN_8);
87     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
88     GPIO_InitStruct.Pull = GPIO_PULLUP;
89     GPIO_InitStruct.Speed = GPIO_SPEED_FAST;
90
91     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
92
93     GPIO_InitStruct.Pin = GPIO_PIN_9;
94     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
95     GPIO_InitStruct.Pull = GPIO_PULLUP;
96     GPIO_InitStruct.Speed = GPIO_SPEED_FAST;
97
98     HAL_GPIO_Init(GPIOI, &GPIO_InitStruct);
99
100    GPIO_InitStruct.Pin = GPIO_PIN_7;
101    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
102    GPIO_InitStruct.Pull = GPIO_PULLUP;
103    GPIO_InitStruct.Speed = GPIO_SPEED_FAST;
104
105    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
106
107    /* -3- Toggle PG.6, PG.8, PI.9 and PC.7 IOs in an infinite loop */
108    while (1)
109    {
110        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_6);
111        /* Insert delay 100 ms */
112        HAL_Delay(100);
113        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_8);
114        /* Insert delay 100 ms */
115        HAL_Delay(100);
116        HAL_GPIO_TogglePin(GPIOI, GPIO_PIN_9);
117        /* Insert delay 100 ms */
118        HAL_Delay(100);
119        HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_7);
120        /* Insert delay 100 ms */
121        HAL_Delay(100);
122    }
123 }
```

分析：从注释可以看出，GPIO 的控制只需要 3 个步骤，使能 GPIO 时钟、配置 GPIO 模式、控制 GPIO 状态。前两个步骤的代码不需要用户手动输入，完全有 STM32CubeMX 生成，用户只需要在 CubeMX 中用图形化界面进行配置。(详细操作步骤可参考本人编写的 STM32Cube 学习笔记，或者其他 STM32Cube 入门教程。)步骤 3 的重点在 HAL_GPIO_TogglePin()函数，在控制 GPIO 反转的。其他控制 GPIO 输出状态的函数还有 HAL_GPIO_WritePin()。

2. GPIO_EXTI

该例程演示如何使用 GPIO 的外部中断功能。

打开...\GPIO_EXTI\MDK-ARM 文件夹下的 MDK 工程，打开 main.c 文件。先看 main 函数：

```
67 int main(void)
68 {
69     /* STM32F4xx HAL library initialization:
70      - Configure the Flash prefetch, instruction and Data caches
71      - Configure the SysTick to generate an interrupt each 1 msec
72      - Set NVIC Group Priority to 4
73      - Global MSP (MCU Support Package) initialization
74     */
75     HAL_Init();
76
77     /* Configure the system clock to 168 MHz */
78     SystemClock_Config();
79
80     /* Configure LED1 and LED2 */
81     BSP_LED_Init(LED1);
82     BSP_LED_Init(LED2);
83
84     /* Configure EXTI Line0 (connected to PA0 pin) in interrupt mode */
85     EXTI_Line0_Config();
86
87     /* Configure EXTI Line15 (connected to PG15 pin) in interrupt mode */
88     EXTI_Line15_10_Config();
89
90     /* Infinite loop */
91     while (1)
92     {
93     }
94 }
```

main 函数只有个语句，分 3 个部分。第一部分是系统时钟配置。第二部分是板级支持配置，本例中是初始化两个 LED 控制口。第三部分是配置外部中断。这三个部分的代码功能都可以通过 CubeMX 配置生成，不需要用户输入。

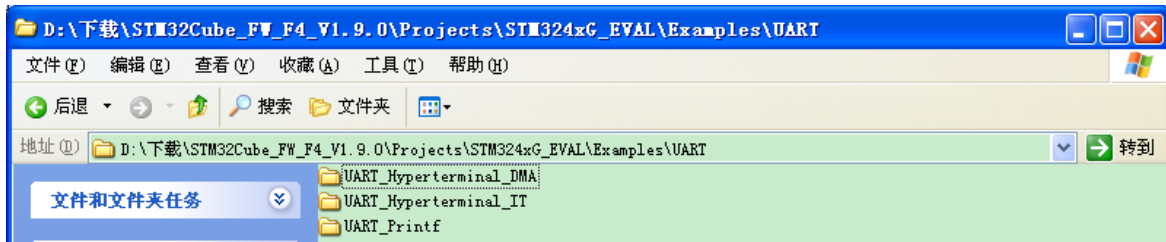
最后，要在回调函数中实现中断响应的功能代码。

```
203 /**
204  * @brief EXTI line detection callbacks
205  * @param GPIO_Pin: Specifies the pins connected EXTI line
206  * @retval None
207  */
208 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
209 {
210     if(GPIO_Pin == KEY_BUTTON_PIN)
211     {
212         /* Toggle LED2 */
213         BSP_LED_Toggle(LED2);
214     }
215
216     if(GPIO_Pin == WAKEUP_BUTTON_PIN)
217     {
218         /* Toggle LED1 */
219         BSP_LED_Toggle(LED1);
220     }
221 }
```

HAL_GPIO_EXTI_Callback()函数是 HAL 库的外部中断回调函数，所有的外部中断都是使用该函数。然后在处理时，判断是哪个引脚号对应的中断。

第二章：UART

UART 共有三个例程。



1. UART_Printf

打开...\UART_Printf\MDK-ARM 文件夹下的 MDK 工程，打开 main.c 文件。

如 main.c 文件开头描述，本例演示了如何将 printf() 函数的输出功能映射到串口上。

```

7  * @brief This example shows how to retarget the C library printf function
8  *       to the UART.
9  ****

```

看 main 函数：

```

75 int main(void)
76 {
77     /* STM32F4xx HAL library initialization:
78      - Configure the Flash prefetch, instruction and Data caches
79      - Configure the SysTick to generate an interrupt each 1 msec
80      - Set NVIC Group Priority to 4
81      - Global MSP (MCU Support Package) initialization
82     */
83     HAL_Init();
84
85     /* Configure the system clock to 168 MHz */
86     SystemClock_Config();
87
88     /*##-1- Configure the UART peripheral #####
89     /* Put the USART peripheral in the Asynchronous mode (UART Mode)
90     /* UART1 configured as follow:
91      - Word Length = 8 Bits
92      - Stop Bit = One Stop bit
93      - Parity = ODD parity
94      - BaudRate = 9600 baud
95      - Hardware flow control disabled (RTS and CTS signals) */
96     UartHandle.Instance = USARTx;
97
98     UartHandle.Init.BaudRate = 9600;
99     UartHandle.Init.WordLength = UART_WORDLENGTH_8B;
100    UartHandle.Init.StopBits = UART_STOPBITS_1;
101    UartHandle.Init.Parity = UART_PARITY_ODD;
102    UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
103    UartHandle.Init.Mode = UART_MODE_TX_RX;
104    UartHandle.Init.OverSampling = UART_OVERSAMPLING_16;
105
106    if (HAL_UART_Init(&UartHandle) != HAL_OK)
107    {
108        /* Initialization Error */
109        Error_Handler();
110    }
111
112    /* Output a message on Hyperterminal using printf function */
113    printf("\n\r UART Printf Example: retarget the C library printf function to the UART\n\r");
114
115    /* Infinite loop */
116    while (1)
117    {
118    }
119 }

```

main 函数分为 3 部分。第一部分是系统初时钟配置。第二部分是初始化 UART。第三部分是使用 printf() 函数输出一个语句。在 126~133 行，就是实现过程。

```

126 PUTCHAR_PROTOTYPE
127 {
128     /* Place your implementation of fputc here */
129     /* e.g. write a character to the EVAL_COM1 and Loop until the e
130     HAL_UART_Transmit(&UartHandle, (uint8_t *)&ch, 1, 0xFFFF);
131
132     return ch;
133 }
134

```


其中 `PUTCHAR_PROTOTYPE` 是一个宏，该宏已经在 `main` 文件开头给出。

```
57 /* Private function prototypes -----*/
58 #ifndef __GNUC__
59 /* With GCC/RAISONANCE, small printf (option LD Linker->Libraries->Small printf
60    set to 'Yes') calls __io_putchar() */
61 #define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
62 #else
63 #define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
64 #endif /* __GNUC__ */
```

因此，对于 MDK-ARM 使用的编译器，126~133 的函数等效于。

```
126 int fputc(int ch, FILE *f)
127 {
128     /* Place your implementation of fputc here */
129     /* e.g. write a character to the EVAL_COM1 and Loop until the e
130     HAL_UART_Transmit(&UartHandle, (uint8_t *)&ch, 1, 0xFFFF);
131
132     return ch;
133 }
134
```

实现该函数并包含 `stdio.h` 文件之后，程序中就可以用 `printf()` 通过 UART 输出字符串了。

2. UART_Hyperterminal_IT

打开... \UART_Hyperterminal_IT\MDK-ARM 文件夹下的 MDK 工程，打开 `main.c` 文件。

该例程演示如何使用串口发送和接收中断。

```
7  * @brief This sample code shows how to use STM32F4xx UART HAL API to transmit
8  *        and receive a data buffer with a communication process based on
9  *        IT transfer.
10 *        The communication is done with the Hyperterminal PC application.
11 *****
```

看 `main` 函数：

```
112 if(HAL_UART_Init(&UartHandle) != HAL_OK)
113 {
114     /* Turn LED3 on: in case of Initialization Error */
115     BSP_LED_On(LED3);
116     while(1)
117     {
118     }
119 }
120
121 /*##-2- Start the transmission process #####*/
122 /* While the UART in reception process, user can transmit data through
123    "aTxBuffer" buffer */
124 if(HAL_UART_Transmit_IT(&UartHandle, (uint8_t*)aTxStartMessage, TXSTARTMESSAGESIZE) != HAL_OK)
125 {
126     /* Turn LED3 on: Transfer error in transmission process */
127     BSP_LED_On(LED3);
128     while(1)
129     {
130     }
131 }
132
133 /*##-3- Put UART peripheral in reception process #####*/
134 /* Any data received will be stored "aRxBuffer" buffer : the number max of
135    data received is 10 */
136 if(HAL_UART_Receive_IT(&UartHandle, (uint8_t *)aRxBuffer, RXBUFFERSIZE) != HAL_OK)
137 {
138     /* Turn LED3 on: Transfer error in reception process */
139     BSP_LED_On(LED3);
140     while(1)
141     {
142     }
143 }
```

在 `main` 函数中已经标注了各个步骤的序号。步骤 1 就是初始化串口，和上一个例子形式差不多。

步骤 2 是演示 `HAL_UART_Transmit_IT()` 函数的用法，该函数的功能是将 `aTxStartMessage[]` 数组的 `TXSTARTMESSAGESIZE` 个字节数据发送出去，并使能发送完成中断，当发送完成后会调用一次 `HAL_UART_TxCpltCallback()` 回调函数。步骤 3 是演示 `HAL_UART_Receive_IT()` 函数的用法，用法和发送函数类似。功能是使能 UART 接收中断，接收的数据存入缓冲数组 `aRxBuffer[]`，在接收数据量达到 `RXBUFFERSIZE` 字节时调用一次 `HAL_UART_RxCpltCallback()` 回调函数。用户可以在回调函数中添加数据处理的代码。

```

145  /*##-4- Wait for the end of the transfer #####*/
146  /* Before starting a new communication transfer, you need to check the current
147     state of the peripheral; if it's busy you need to wait for the end of current
148     transfer before starting a new one.
149     For simplicity reasons, this example is just waiting till the end of the
150     transfer, but application may perform other tasks while transfer operation
151     is ongoing. */
152  while (HAL_UART_GetState(&UartHandle) != HAL_UART_STATE_READY)
153  {
154  }
155
156  /*##-5- Send the received Buffer #####*/
157  if(HAL_UART_Transmit_IT(&UartHandle, (uint8_t*)aRxBuffer, RXBUFFERSIZE) != HAL_OK)
158  {
159      /* Turn LED3 on: Transfer error in transmission process */
160      BSP_LED_On(LED3);
161      while(1)
162      {
163      }
164  }

```

步骤 4 是等待串口空闲。步骤 5 是再次发送数据。

```

166  /*##-6- Wait for the end of the transfer #####*/
167  while (HAL_UART_GetState(&UartHandle) != HAL_UART_STATE_READY)
168  {
169  }
170
171  /*##-7- Send the End Message #####*/
172  if(HAL_UART_Transmit_IT(&UartHandle, (uint8_t*)aTxEndMessage, TXENDMESSAGESIZE) != HAL_OK)
173  {
174      /* Turn LED3 on: Transfer error in transmission process */
175      BSP_LED_On(LED3);
176      while(1)
177      {
178      }
179  }
180
181  /*##-8- Wait for the end of the transfer #####*/
182  while (HAL_UART_GetState(&UartHandle) != HAL_UART_STATE_READY)
183  {
184  }
185
186  /* Infinite loop */
187  while (1)
188  {
189  }
190 }

```

步骤 6、7、8，过程和前面类似，注释已经说明清楚。

```

260 void HAL_UART_TxCpltCallback(UART_HandleTypeDef *UartHandle)
261 {
262     /* Turn LED1 on: Transfer in transmission process is correct */
263     BSP_LED_On(LED1);
264 }

```

```

273 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *UartHandle)
274 {
275     /* Turn LED2 on: Transfer in reception process is correct */
276     BSP_LED_On(LED2);
277 }

```

本例中回调函数的内容很简单，就是点亮 LED。

3. UART_Hyperterminal_DMA

该例程演示如何使用串口 DMA 发送和接收及中断，和 UART_Hyperterminal_IT 结构完全相同，只是把函数的后缀都改成了_DMA。而且回调函数都是一样的。

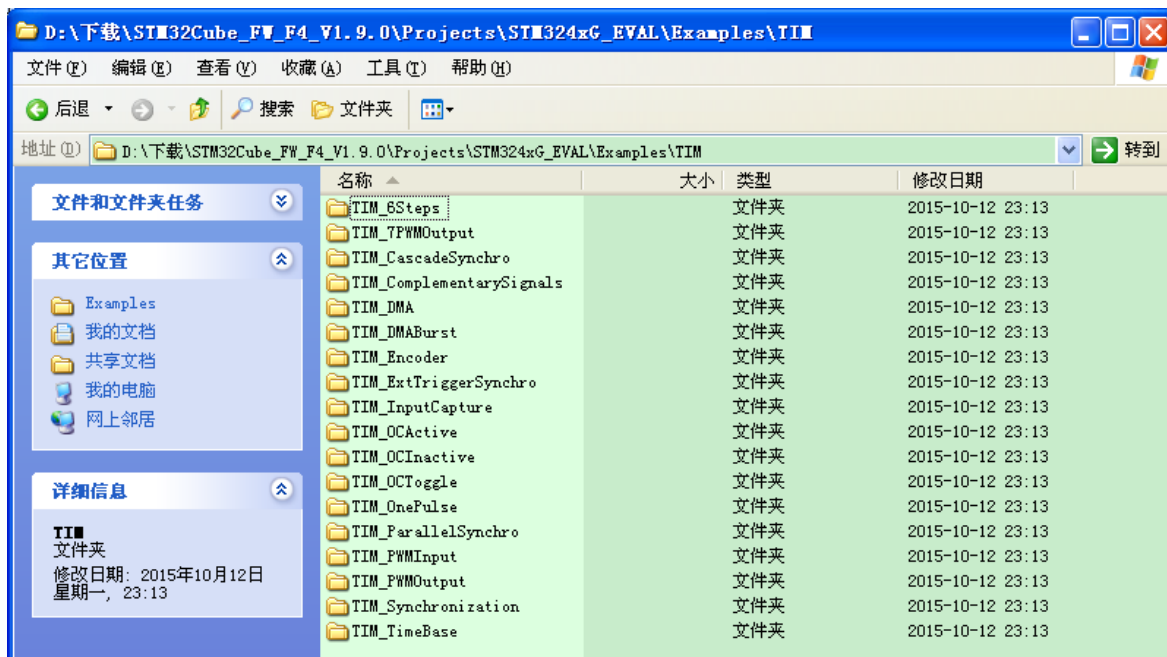
```

7  * @brief This sample code shows how to use STM32F4xx UART HAL API to transmit
8  *        and receive a data buffer with a communication process based on
9  *        DMA transfer.
10 *        The communication is done with the Hyperterminal PC application.
11 *****

```

第三章：TIM

TIM 共有 18 个例程。定时器是 STM32 中用途最多变的外设。下面分析几个典型应用例程。



1. TIM_TimeBase

打开...\TIM_TimeBase\MDK-ARM 文件夹下的 MDK 工程，打开 main.c 文件。

该例程是定时器最基本的应用，即定时中断功能。

```
7  * @brief This sample code shows how to use STM32F4xx TIM HAL API to generate
8  * a time base.
9  *****
```

看 main 函数：

```
109 /* Compute the prescaler value to have TIM3 counter clock equal to 10 KHz */
110 uwPrescalerValue = (uint32_t) ((SystemCoreClock / 2) / 10000) - 1;
111
112 /* Set TIMx instance */
113 TimHandle.Instance = TIMx;
114
115 /* Initialize TIM3 peripheral as follow:
116  + Period = 10000 - 1
117  + Prescaler = ((SystemCoreClock/2)/10000) - 1
118  + ClockDivision = 0
119  + Counter direction = Up
120 */
121 TimHandle.Init.Period = 10000 - 1;
122 TimHandle.Init.Prescaler = uwPrescalerValue;
123 TimHandle.Init.ClockDivision = 0;
124 TimHandle.Init.CounterMode = TIM_COUNTERMODE_UP;
125 if (HAL_TIM_Base_Init(&TimHandle) != HAL_OK)
126 {
127     /* Initialization Error */
128     Error_Handler();
129 }
130
131 /*##-2- Start the TIM Base generation in interrupt mode #####*/
132 /* Start Channell */
133 if (HAL_TIM_Base_Start_IT(&TimHandle) != HAL_OK)
134 {
135     /* Starting Error */
136     Error_Handler();
137 }
138
139 /* Infinite loop */
140 while (1)
141 {
142 }
143 }
```

一共只有两个步骤。步骤 1 是配置定时器，步骤 2 是启动定时器并使能中断。步骤 1 是通过 CubeMX 配置生成代码。步骤 2 需要用户手动添加。


```

145 /**
146  * @brief Period elapsed callback in non blocking mode
147  * @param htim : TIM handle
148  * @retval None
149  */
150 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
151 {
152     BSP_LED_Toggle(LED1);
153 }

```

HAL_TIM_PeriodElapsedCallback()是定时器 Update 中断回调函数。所有定时器的更新中断都使用该回调函数接口。因此，如果开启了多个定时器更新中断时，应该对中断源进行判断，如下图：

```

150 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
151 {
152     if (htim->Instance == TIMx) {
153         BSP_LED_Toggle(LED1);
154     }
155 }

```

2. TIM_PWMOutput

打开...\TIM_PWMOutput\MDK-ARM 文件夹下的 MDK 工程，打开 main.c 文件。

该例程演示怎么使用定时器的 PWM 模式产生 4 路 PWM 信号。

```

7  * @brief This sample code shows how to use STM32F4xx TIM HAL API to generate
8  * 4 signals in PWM.
9  ****

```

看 main 函数：

```

99  /*##-1- Configure the TIM peripheral #####*/
100  /* -----*/
132  /* Initialize TIMx peripheral as follow:
133  TimHandle.Instance = TIMx;
138
139
140  TimHandle.Init.Prescaler = uhPrescalerValue;
141  TimHandle.Init.Period = PERIOD_VALUE;
142  TimHandle.Init.ClockDivision = 0;
143  TimHandle.Init.CounterMode = TIM_COUNTERMODE_UP;
144  if (HAL_TIM_PWM_Init(&TimHandle) != HAL_OK)
145  {
146      /* Initialization Error */
147      Error_Handler();
148  }
149
150  /*##-2- Configure the PWM channels #####*/
151  /* Common configuration for all channels */
152  sConfig.OCMode = TIM_OCMode_PWM1;
153  sConfig.OCpolarity = TIM_OCPOLARITY_HIGH;
154  sConfig.OCFastMode = TIM_OCFAST_DISABLE;
155
156  /* Set the pulse value for channel 1 */
157  sConfig.Pulse = PULSE1_VALUE;
158  if (HAL_TIM_PWM_ConfigChannel(&TimHandle, &sConfig, TIM_CHANNEL_1) != HAL_OK)
159  {
160      /* Configuration Error */
161      Error_Handler();
162  }
163
164  /* Set the pulse value for channel 2 */
165  sConfig.Pulse = PULSE2_VALUE;
166  if (HAL_TIM_PWM_ConfigChannel(&TimHandle, &sConfig, TIM_CHANNEL_2) != HAL_OK)
167  {
168
169
170
171
172  /* Set the pulse value for channel 3 */
173  sConfig.Pulse = PULSE3_VALUE;
174  if (HAL_TIM_PWM_ConfigChannel(&TimHandle, &sConfig, TIM_CHANNEL_3) != HAL_OK)
175  {
176
177
178
179
180  /* Set the pulse value for channel 4 */
181  sConfig.Pulse = PULSE4_VALUE;
182  if (HAL_TIM_PWM_ConfigChannel(&TimHandle, &sConfig, TIM_CHANNEL_4) != HAL_OK)
183  {

```

步骤 1 是配置 TIM 外设。步骤 2 是配置 PWM 通道。这两步骤的代码可由 CubeMX 配置生。

```

183 {
184     /*##-3- Start PWM signals generation #####*/
185     /* Start channel 1 */
186     if(HAL_TIM_PWM_Start(&TimHandle, TIM_CHANNEL_1) != HAL_OK)
187     {
188         /* PWM Generation Error */
189         Error_Handler();
190     }
191     /* Start channel 2 */
192     if(HAL_TIM_PWM_Start(&TimHandle, TIM_CHANNEL_2) != HAL_OK)
193     {
194         /* Start channel 3 */
195         if(HAL_TIM_PWM_Start(&TimHandle, TIM_CHANNEL_3) != HAL_OK)
196         {
197             /* Start channel 4 */
198             if(HAL_TIM_PWM_Start(&TimHandle, TIM_CHANNEL_4) != HAL_OK)
199             {
200                 /* Infinite loop */
201                 while (1)
202                 {
203                 }
204             }
205         }
206     }
207 }

```

步骤 3 是启动各个 PWM 通道，就是调用 HAL_TIM_PWM_Start() 函数。该步骤的代码要用户添加。经过上述 3 个步骤，就可以在相应引脚输出 PWM 信号了。

3. TIM_InputCapture

打开...\TIM_InputCapture\MDK-ARM 文件夹下的 MDK 工程，打开 main.c 文件。

该例程演示 TIM 的输入捕获功能，用该功能测量信号的频率或周期。

```

7  * @brief This example shows how to use the TIM peripheral to measure only
8  * the frequency of an external signal.
9  *****

```

看 main 函数：

```

100 /*##-1- Configure the TIM peripheral #####*/
101 /* TIM1 configuration: Input Capture mode -----
102 */
103 /* Set TIMx instance */
104 TimHandle.Instance = TIMx;
105
106 /* Initialize TIMx peripheral as follow:
107 + Period = 0xFFFF
108 + Prescaler = 0
109 + ClockDivision = 0
110 + Counter direction = Up
111 */
112 TimHandle.Init.Period = 0xFFFF;
113 TimHandle.Init.Prescaler = 0;
114 TimHandle.Init.ClockDivision = 0;
115 TimHandle.Init.CounterMode = TIM_COUNTERMODE_UP;
116 TimHandle.Init.RepetitionCounter = 0;
117
118 if(HAL_TIM_IC_Init(&TimHandle) != HAL_OK)
119 {
120     /* Initialization Error */
121     Error_Handler();
122 }
123
124 /*##-2- Configure the Input Capture channel #####*/
125 /* Configure the Input Capture of channel 2 */
126 sICConfig.ICPolarity = TIM_ICPOLARITY_RISING;
127 sICConfig.ICSelection = TIM_ICSELECTION_DIRECTTI;
128 sICConfig.ICPrescaler = TIM_ICPSC_DIV1;
129 sICConfig.ICFilter = 0;
130 if(HAL_TIM_IC_ConfigChannel(&TimHandle, &sICConfig, TIM_CHANNEL_2) != HAL_OK)
131 {
132     /* Configuration Error */
133     Error_Handler();
134 }
135
136

```

步骤 1 配置 TIM 外设，步骤 2 配置输入捕获通道。

```

140  /*##-3- Start the Input Capture in interrupt mode #####*/
141  if(HAL_TIM_IC_Start_IT(&TimHandle, TIM_CHANNEL_2) != HAL_OK)
142  {
143      /* Starting Error */
144      Error_Handler();
145  }
146
147  /* Infinite loop */
148  while (1)
149  {
150  }
151  }

```

步骤 3 启动输入捕获功能，并使能相应中断。

```

158 void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
159 {
160     if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2)
161     {
162         if(uhCaptureIndex == 0)
163         {
164             /* Get the 1st Input Capture value */
165             uwIC2Value1 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_2);
166             uhCaptureIndex = 1;
167         }
168         else if(uhCaptureIndex == 1)
169         {
170             /* Get the 2nd Input Capture value */
171             uwIC2Value2 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_2);
172
173             /* Capture computation */
174             if (uwIC2Value2 > uwIC2Value1)
175             {
176                 uwDiffCapture = (uwIC2Value2 - uwIC2Value1);
177             }
178             else /* (uwIC2Value2 <= uwIC2Value1) */
179             {
180                 uwDiffCapture = ((0xFFFF - uwIC2Value1) + uwIC2Value2);
181             }
182
183             /* Frequency computation: for this example TIMx (TIM1) is clocked by
184              2xAPB2Clk */
185             uwFrequency = (2*HAL_RCC_GetPCLK2Freq()) / uwDiffCapture;
186             uhCaptureIndex = 0;
187         }
188     }
189 }

```

对捕获值的处理在中断回调函数中进行。一共捕获两次，两次捕获值的差值就是乘以定时器时钟周期，就得到信号的周期，其倒数就是信号频率。

4. TIM_PWMInput

该例程演示用 TIM 的输入捕获功能测量 PWM 信号的占空比。

打开...\TIM_PWMInput\MDK-ARM 文件夹下的 MDK 工程。

```

165  /*##-4- Start the Input Capture in interrupt mode #####*/
166  if(HAL_TIM_IC_Start_IT(&TimHandle, TIM_CHANNEL_2) != HAL_OK)
167  {
168      /* Starting Error */
169      Error_Handler();
170  }
171
172  /*##-5- Start the Input Capture in interrupt mode #####*/
173  if(HAL_TIM_IC_Start_IT(&TimHandle, TIM_CHANNEL_1) != HAL_OK)
174  {
175      /* Starting Error */
176      Error_Handler();
177  }
178
179  /* Infinite loop */
180  while (1)
181  {
182  }
183  }

```

关键步骤就是同一个定时器开启了两个输入捕获通道，一个捕获上升沿，另一个捕获下降沿。

```
190 void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
191 {
192     if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2)
193     {
194         /* Get the Input Capture value */
195         uwIC2Value = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_2);
196
197         if (uwIC2Value != 0)
198         {
199             /* Duty cycle computation */
200             uwDutyCycle = ((HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1)) * 100) / uwIC2Value;
201
202             /* uwFrequency computation
203             TIM4 counter clock = (RCC_Clocks.HCLK_Frequency)/2 */
204             uwFrequency = (HAL_RCC_GetHCLKFreq())/2 / uwIC2Value;
205         }
206         else
207         {
208             uwDutyCycle = 0;
209             uwFrequency = 0;
210         }
211     }
212 }
```

在捕获中断回调函数中进行数据处理。两次相邻的上升沿的时间差就是 PWM 信号周期，这两次上升沿之间有个下降沿，该下降沿和第一次上升沿的时间差就是 PWM 信号的占空比。

5. TIM_OCActive

该例程演示用 TIM 的比较匹配功能输出 4 路 PWM 信号。

打开...\TIM_OCActive\MDK-ARM 文件夹下的 MDK 工程。

```
100 /*##-1- Configure the TIM peripheral #####*/
101 /* -----*/
129 /* Initialize TIMx peripheral as follow:
135 TimHandle.Instance = TIMx;
136
137 TimHandle.Init.Period      = 65535;
138 TimHandle.Init.Prescaler   = uwPrescalerValue;
139 TimHandle.Init.ClockDivision = 0;
140 TimHandle.Init.CounterMode = TIM_COUNTERMODE_UP;
141 if (HAL_TIM_OC_Init(&TimHandle) != HAL_OK)
142 {
143     /* Initialization Error */
144     Error_Handler();
145 }
```

步骤 1 配置 TIM 外设。

```
147 /*##-2- Configure the Output Compare channels #####*/
148 /* Common configuration for all channels */
149 sConfig.OCMode = TIM_OCMode_ACTIVE;
150 sConfig.OCpolarity = TIM_OCPolarity_HIGH;
151
152 /* Set the pulse (delay1) value for channel 1 */
153 sConfig.Pulse = PULSE1_VALUE;
154 if (HAL_TIM_OC_ConfigChannel(&TimHandle, &sConfig, TIM_CHANNEL_1) != HAL_OK)
155 {
156     /* Configuration Error */
157     Error_Handler();
158 }
159
160 /* Set the pulse (delay2) value for channel 2 */
161 sConfig.Pulse = PULSE2_VALUE;
162 if (HAL_TIM_OC_ConfigChannel(&TimHandle, &sConfig, TIM_CHANNEL_2) != HAL_OK)
163 {
164     /* Configuration Error */
165     Error_Handler();
166 }
167
168 /* Set the pulse (delay3) value for channel 3 */
169 sConfig.Pulse = PULSE3_VALUE;
170 if (HAL_TIM_OC_ConfigChannel(&TimHandle, &sConfig, TIM_CHANNEL_3) != HAL_OK)
171 {
172     /* Configuration Error */
173     Error_Handler();
174 }
175
176 /* Set the pulse (delay4) value for channel 4 */
177 sConfig.Pulse = PULSE4_VALUE;
178 if (HAL_TIM_OC_ConfigChannel(&TimHandle, &sConfig, TIM_CHANNEL_4) != HAL_OK)
179 {
180     /* Configuration Error */
181     Error_Handler();
182 }
```

步骤 2 配置各通道的输出比较参数，包括输出模式、极性、比较值等。

```

184  /*##-3- Turn On LED1: use PG6 falling edge as reference #####*/
185  /* Turn on LED1 */
186  BSP_LED_On(LED1);
187
188  /*##-4- Start signals generation #####*/
189  /* Start channel 1 in Output compare mode */
190  if(HAL_TIM_OC_Start(&TimHandle, TIM_CHANNEL_1) != HAL_OK)
191  {
192      /* Starting Error */
193      Error_Handler();
194  }
195  /* Start channel 2 in Output compare mode */
196  if(HAL_TIM_OC_Start(&TimHandle, TIM_CHANNEL_2) != HAL_OK)
197  {
201  /* Start channel 3 in Output compare mode */
202  if(HAL_TIM_OC_Start(&TimHandle, TIM_CHANNEL_3) != HAL_OK)
203  {
207  /* Start channel 4 in Output compare mode */
208  if(HAL_TIM_OC_Start(&TimHandle, TIM_CHANNEL_4) != HAL_OK)
209  {
214  /* Infinite loop */
215  while (1)
216  {
217  }
218  }

```

步骤 3 点亮 LED1，这一步无关紧要。

步骤 4 启动各通道比较匹配输出功能。

6. TIM_OCInactive

该例程和 TIM_OCActive 的步骤完全相同，只是 PWM 输出信号的进行取反，并且在步骤 4 中启动个通道时使能了比较匹配中断。最后在中断函数中分别控制 LED 的熄灭。

```

191  /*##-4- Start signals generation #####*/
192  /* Start channel 1 in Output compare mode */
193  if(HAL_TIM_OC_Start_IT(&TimHandle, TIM_CHANNEL_1) != HAL_OK)
194  {
195      /* Starting Error */
196      Error_Handler();
197  }
198  /* Start channel 2 in Output compare mode */
199  if(HAL_TIM_OC_Start_IT(&TimHandle, TIM_CHANNEL_2) != HAL_OK)
200  {
204  /* Start channel 3 in Output compare mode */
205  if(HAL_TIM_OC_Start_IT(&TimHandle, TIM_CHANNEL_3) != HAL_OK)
206  {
210  /* Start channel 4 in Output compare mode */
211  if(HAL_TIM_OC_Start_IT(&TimHandle, TIM_CHANNEL_4) != HAL_OK)
212  {
217  /* Infinite loop */
218  while (1)
219  {
220  }
221  }

```

7. TIM_OCToggle

该例程演示如何用 TIM 的比较输出功能的翻转模式，输出 4 路频率不同的 50%占空比的方波信号。

打开...\TIM_OCToggle\MDK-ARM 文件夹下的 MDK 工程。

```

97  /*##-1- Configure the TIM peripheral #####*/
98  /* -----*/
137  /* Compute the prescaler value to have TIMx counter clock equal to 21 MHz */
138  uwPrescalerValue = (uint32_t)((SystemCoreClock / 2) / 21000000) - 1;
139
140  /* Initialize TIMx peripheral as follow:
141  *
142  *  - Period = 65535
143  *  - Prescaler = uwPrescalerValue
144  *  - ClockDivision = 0
145  *  - CounterMode = TIM_COUNTERMODE_UP
146  */
147  TimHandle.Instance = TIMx;
148  TimHandle.Init.Period = 65535;
149  TimHandle.Init.Prescaler = uwPrescalerValue;
150  TimHandle.Init.ClockDivision = 0;
151  TimHandle.Init.CounterMode = TIM_COUNTERMODE_UP;
152  if(HAL_TIM_OC_Init(&TimHandle) != HAL_OK)
153  {
154      /* Initialization Error */
155      Error_Handler();
156  }

```

步骤 1 配置 TIM 外设。


```

158  /*##-2- Configure the Output Compare channels #####*/
159  /* Output Compare Toggle Mode configuration: Channel1 */
160  sConfig.OCMode = TIM_OCMODE_TOGGLE;
161  sConfig.Pulse = uhCCR1_Val;
162  sConfig.OCpolarity = TIM_OCPOLARITY_LOW;
163  if(HAL_TIM_OC_ConfigChannel(&TimHandle, &sConfig, TIM_CHANNEL_1) != HAL_OK)
164  {
165      /* Configuration Error */
166      Error_Handler();
167  }
168
169  /* Output Compare Toggle Mode configuration: Channel2 */
170  sConfig.Pulse = uhCCR2_Val;
171  if(HAL_TIM_OC_ConfigChannel(&TimHandle, &sConfig, TIM_CHANNEL_2) != HAL_OK)
172  {
173
174  }
175
176  /* Output Compare Toggle Mode configuration: Channel3 */
177  sConfig.Pulse = uhCCR3_Val;
178  if(HAL_TIM_OC_ConfigChannel(&TimHandle, &sConfig, TIM_CHANNEL_3) != HAL_OK)
179  {
180
181  }
182
183  /* Output Compare Toggle Mode configuration: Channel4 */
184  sConfig.Pulse = uhCCR4_Val;
185  if(HAL_TIM_OC_ConfigChannel(&TimHandle, &sConfig, TIM_CHANNEL_4) != HAL_OK)
186  {
187
188  }

```

步骤 2 配置输出比较通道参数，包括输出模式、极性、比较值等。

```

193  /*##-3- Start signals generation #####*/
194  /* Start channel 1 in Output compare mode */
195  if(HAL_TIM_OC_Start_IT(&TimHandle, TIM_CHANNEL_1) != HAL_OK)
196  {
197      /* Starting Error */
198      Error_Handler();
199  }
200  /* Start channel 2 in Output compare mode */
201  if(HAL_TIM_OC_Start_IT(&TimHandle, TIM_CHANNEL_2) != HAL_OK)
202  {
203
204  }
205  /* Start channel 3 in Output compare mode */
206  if(HAL_TIM_OC_Start_IT(&TimHandle, TIM_CHANNEL_3) != HAL_OK)
207  {
208
209  }
210  /* Start channel 4 in Output compare mode */
211  if(HAL_TIM_OC_Start_IT(&TimHandle, TIM_CHANNEL_4) != HAL_OK)
212  {
213
214  }
215
216  /* Infinite loop */
217  while (1)
218  {
219
220  }
221
222
223

```

步骤 3 启动各通道的输出比较功能，并使能中断。

```

230 void HAL_TIM_OC_DelayElapsedCallback(TIM_HandleTypeDef *htim)
231 {
232     /* TIM3_CH1 toggling with frequency = 256.35 Hz */
233     if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)
234     {
235         uhCapture = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);
236         /* Set the Capture Compare Register value */
237         HAL_TIM_SET_COMPARE(&TimHandle, TIM_CHANNEL_1, (uhCapture + uhCCR1_Val));
238     }
239
240     /* TIM3_CH2 toggling with frequency = 256.35 Hz */
241     if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2)
242     {
243
244     }
245
246     /* TIM3_CH3 toggling with frequency = 256.35 Hz */
247     if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_3)
248     {
249
250     }
251
252     /* TIM3_CH4 toggling with frequency = 256.35 Hz */
253     if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_4)
254     {
255
256     }
257
258
259
260
261
262
263

```

关键的步骤是中断回调函数中。因为其实 TIM 硬件并不能同时输出多路不同频率的 PWM 信号，这里是通过不断修正各个通道的比较值，才实现了 4 路不同频率的方波信号。

8. TIM_OnePulse

该例程演示如何用 TIM 输出单脉冲信号。通过通道 2 输入信号，上升沿触发通道 1 产生脉冲信号。打开... \TIM_OnePulse \MDK-ARM 文件夹下的 MDK 工程。

```

7  * @brief This sample code shows how to use STM32F4xx TIM HAL API to generate
8  *       a one pulse signal
9  *****

```

看 main 函数:

```

94  /*##-1- Configure the TIM peripheral #####
95  /* -----
135 /* Initialize TIMx peripheral as follow:
142 TimHandle.Instance = TIMx;
143
144 TimHandle.Init.Period = 0xFFFF;
145 TimHandle.Init.Prescaler = uwPrescalerValue;
146 TimHandle.Init.ClockDivision = 0;
147 TimHandle.Init.CounterMode = TIM_COUNTERMODE_UP;
148
149 if(HAL_TIM_OnePulse_Init(&TimHandle, TIM_OPMODE_SINGLE) != HAL_OK)
150 {
151     /* Initialization Error */
152     Error_Handler();
153 }

```

步骤 1 配置 TIM 外设。

```

155 /*##-2- Configure the Channel 1 in One Pulse mode #####*/
156 sConfig.OCMode = TIM_OCMode_PWM2;
157 sConfig.OCpolarity = TIM_OCPOLARITY_HIGH;
158 sConfig.Pulse = 16383;
159 sConfig.ICPolarity = TIM_ICPOLARITY_RISING;
160 sConfig.ICSelection = TIM_ICSELECTION_DIRECTTI;
161 sConfig.ICFilter = 0;
162
163 if(HAL_TIM_OnePulse_ConfigChannel(&TimHandle, &sConfig, TIM_CHANNEL_1, TIM_CHANNEL_2) != HAL_OK)
164 {
165     /* Configuration Error */
166     Error_Handler();
167 }

```

步骤 2 配置通道 1 为单脉冲模式，输入通道是通道 2。

```

169 /*##-3- Start the One Pulse mode #####
170 if(HAL_TIM_OnePulse_Start(&TimHandle, TIM_CHANNEL_2) != HAL_OK)
171 {
172     /* Starting Error */
173     Error_Handler();
174 }
175
176 /* Infinite loop */
177 while (1)
178 {
179 }
180 }

```

步骤 3 启动单脉冲通道 2。

```

94  /*##-1- Configure the TIM peripheral #####*/
95  /* -----
96  TIM4 configuration: One Pulse mode
97  The external signal is connected to TIM4_CH2 pin (PB.07),
98  The Rising edge is used as active edge,
99  The One Pulse signal is output on TIM4_CH1 pin (PB.06)
100  The TIM_Pulse defines the delay value
101  The (TIM_Period - TIM_Pulse) defines the One Pulse value.
102
103  TIM4 input clock (TIM4CLK) is set to 2 * APB1 clock (PCLK1),
104  since APB1 prescaler is different from 1.
105  TIM4CLK = 2 * PCLK1
106  PCLK1 = HCLK / 4
107  => TIM4CLK = HCLK / 2 = SystemCoreClock / 2
108
109  TIM2CLK = SystemCoreClock/2, we want to get TIM2 counter clock at 42 MHz:
110  Prescaler = (TIM2CLK / TIM2 counter clock) - 1
111  Prescaler = ((SystemCoreClock / 2) / 42 MHz) - 1
112
113  The Autoreload value is 65535 (TIM4->ARR), so the maximum frequency value
114  to trigger the TIM4 input is 42000000/65535 = 641 Hz.
115
116  The TIM_Pulse defines the delay value, the delay value is fixed
117  to 390 us:
118  delay = CCR1/TIM4 counter clock
119  = 16383 / 42000000 = 390 us.
120  The (TIM_Period - TIM_Pulse) defines the One Pulse value,
121  the pulse value is fixed to 1.170 ms:
122  One Pulse value = (TIM_Period - TIM_Pulse) / TIM4 counter clock
123  = (65535 - 16383) / 42000000 = 1.170 ms.

```

从步骤 1 的注释中，详细描述了单脉冲的产生过程和参数计算。

9. TIM_DMA

该例程演示通过 TIM 更新请求触发 DMA 更新 TIM 的输出比较值，以改变 PWM 的占空比。

打开...\TIM_DMA\MDK-ARM 文件夹下的 MDK 工程。

```
7  * @brief This sample code shows how to use DMA with TIM1 Update request to
8  *       transfer Data from memory to TIM1 Capture Compare Register 3 (CCR3).
9  *****
```

看 main 函数：

```
104 /*##-1- Configure the TIM peripheral #####
105  */
134 /* Initialize TIM3 peripheral as follow:
141 TimHandle.Instance = TIMx;
142
143 TimHandle.Init.Period      = uhTimerPeriod;
144 TimHandle.Init.RepetitionCounter = 3;
145 TimHandle.Init.Prescaler   = 0;
146 TimHandle.Init.ClockDivision = 0;
147 TimHandle.Init.CounterMode = TIM_COUNTERMODE_UP;
148 if(HAL_TIM_PWM_Init(&TimHandle) != HAL_OK)
149 {
150     /* Initialization Error */
151     Error_Handler();
152 }
```

步骤 1 配置 TIM 外设。在此 RepetitionCounter 是一个重要参数，等于 3 就是 3 个周期触发一次更新请求。

```
154 /*##-2- Configure the PWM channel 3 #####
155 sConfig.OCMode      = TIM_OCMode_PWM1;
156 sConfig.OCpolarity  = TIM_OCPolarity_HIGH;
157 sConfig.Pulse       = aCCValue_Buffer[0];
158 if(HAL_TIM_PWM_ConfigChannel(&TimHandle, &sConfig, TIM_CHANNEL_3) != HAL_OK)
159 {
160     /* Configuration Error */
161     Error_Handler();
162 }
163
164 /*##-3- Start PWM signal generation in DMA mode #####
165 if( HAL_TIM_PWM_Start_DMA(&TimHandle, TIM_CHANNEL_3, aCCValue_Buffer, 3) != HAL_OK)
166 {
167     /* Starting PWM generation Error */
168     Error_Handler();
169 }
170
171 /* Infinite loop */
172 while (1)
173 {
174 }
175 }
```

步骤 2 配置 PWM 通道参数，包括输出模式、极性、脉冲宽度。步骤 3 以 DMA 模式启动 PWM 输出，注意 HAL_TIM_PWM_Start_DMA() 函数的最后两个参数，一个是数据地址，一个是数据量。

程序运行的过程：TIM 的定时周期由变量 uhTimerPeriod 确定，由于 RepetitionCounter=3，TIM 经过 3 个定时周期就触发一次 DMA 传输。DMA 传输是将 aCCValue_Buffer[] 的数据传入 TIM 的 CHANNEL_3 的比较匹配寄存器中，即改变 PWM 的占空比。第一次 DMA 传输的是 aCCValue_Buffer[0]，第二次传输 aCCValue_Buffer[1]，第三次传输 aCCValue_Buffer[2]。三次 DMA 传输就完成了整个循环周期。第四次又传输 aCCValue_Buffer[0]，如此往下循环。

10. TIM_DMABurst

该例程演示通过 TIM 更新请求触发 DMA 突发传输更新 TIM 的 ARR、RCR、CCR1 寄存器的值，以改变 PWM 的周期、占空比。

打开...\TIM_DMABurst\MDK-ARM 文件夹下的 MDK 工程。

```

94  /*##-1- Configure the TIM peripheral #####*/
95  /* -----*/
122  TimHandle.Instance = TIMx;
123
124  TimHandle.Init.Period            = 0xFFFF;
125  TimHandle.Init.RepetitionCounter = 0;
126  TimHandle.Init.Prescaler        = (uint16_t) ((SystemCoreClock / 24000000) - 1);
127  TimHandle.Init.ClockDivision    = 0;
128  TimHandle.Init.CounterMode      = TIM_COUNTERMODE_UP;
129  if(HAL_TIM_PWM_Init(&TimHandle) != HAL_OK)
130  {
131      /* Initialization Error */
132      Error_Handler();
133  }

```

步骤 1 配置 TIM 外设。

```

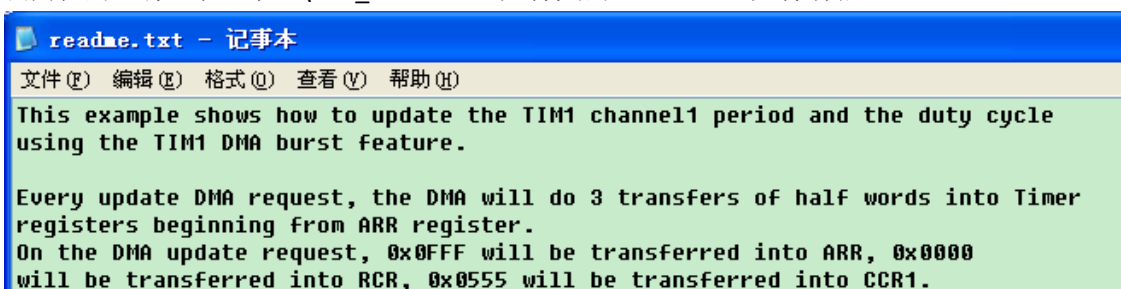
135  /*##-2- Configure the PWM channel 3 #####*/
136  sConfig.OCMode          = TIM_OCMODE_PWM1;
137  sConfig.OCpolarity       = TIM_OCPOLARITY_HIGH;
138  sConfig.Pulse            = 0xFFFF;
139  if(HAL_TIM_PWM_ConfigChannel(&TimHandle, &sConfig, TIM_CHANNEL_1) != HAL_OK)
140  {
141      /* Configuration Error */
142      Error_Handler();
143  }
144
145  /*##-3- Start PWM signal generation in DMA mode #####*/
146  if( HAL_TIM_PWM_Start(&TimHandle, TIM_CHANNEL_1) != HAL_OK)
147  {
148      /* Starting PWM generation Error */
149      Error_Handler();
150  }
151
152  /*##-4- Start DMA Burst transfer #####*/
153  HAL_TIM_DMABurst_WriteStart(&TimHandle, TIM_DMABASE_ARR, TIM_DMA_UPDATE,
154                              (uint32_t*)aSRC_Buffer, TIM_DMABURSTLENGTH_3TRANSFERS);
155
156  /* Infinite loop */
157  while (1)
158  {
159  }
160 }

```

步骤 2 配置 PWM 通道。步骤 3 启动 PWM 输出。步骤 4 启动 DMA 突发传输。

注意 HAL_TIM_DMABurst_WriteStart()中几个被标注的参数，他们只有可用的有效取值，查看该函数在 stm32f4xx_hal_tim.c 的位置，有详细的说明。

该例程的运行过程，在...\TIM_DMABurst 文件夹的 readme.txt 文件有描述。



readme.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

This example shows how to update the TIM1 channel1 period and the duty cycle using the TIM1 DMA burst feature.

Every update DMA request, the DMA will do 3 transfers of half words into Timer registers beginning from ARR register.

On the DMA update request, 0x0FFF will be transferred into ARR, 0x0000 will be transferred into RCR, 0x0555 will be transferred into CCR1.

“每次更新的 DMA 请求，DMA 就传输 3 个 Half Word 到定时器的寄存器，起始寄存器为 ARR。

在 DMA 更新请求中，0x0FFF 被传入 ARR，0x0000 传入 RCR，0x0555 传入 CCR1。”

对比例程代码，传输 3 个 Half Word，这是由 HAL_TIM_DMABurst_WriteStart()的最后一个参数值 TIM_DMABURSTLENGTH_3TRANSFERS 确定的。传输的三个值 0x0FFF、0x0000、0x0555，就是数组 aSRC_Buffer[3]的 3 个元素。对应的三个寄存器 ARR、RCR、CCR1，它们的地址是连续的。触发事件是 TIM 更新，由参数 TIM_DMA_UPDATE 确定。

readme.txt 的描述传输单位是“Half Word”，可能有误。因为 HAL_TIM_DMABurst_WriteStart()的第四个参数的类型是(uint32_t*)，而且 TIM 的所有寄存器占用空间是 32bit(虽然都只使用了低 16bit)，即上面的 ARR、RCR、CCR1 实际地址间隔是 4 字节。所以可以判断，DMA 传输的单位是 Word 而不是 Half Word。

11. TIM_ComplementarySignals

该例程演示使用 TIM1 产生 3 对互补 PWM 信号。

打开...\TIM_ComplementarySignals\MDK-ARM 文件夹下的 MDK 工程。

```
101  /*##-1- Configure the TIM peripheral #####
102  /* -----
103  /* Initialize TIM peripheral as follow:
104  /* Select the Timer instance */
105  TimHandle.Instance = TIM1;
106
107  TimHandle.Init.Prescaler      = uwPrescalerValue;
108  TimHandle.Init.Period        = PERIOD_VALUE;
109  TimHandle.Init.ClockDivision = 0;
110  TimHandle.Init.CounterMode   = TIM_COUNTERMODE_UP;
111  TimHandle.Init.RepetitionCounter = 0;
112
113  if(HAL_TIM_PWM_Init(&TimHandle) != HAL_OK)
114  {
115      /* Initialization Error */
116      Error_Handler();
117  }
```

步骤 1 是 TIM 外设基本配置。

```
164  /*##-2- Configure the PWM channels #####*/
165  /* Common configuration for all channels */
166  sPWMConfig.OCMode          = TIM_OCMode_PWM1;
167  sPWMConfig.OCpolarity      = TIM_OCPolarity_HIGH;
168  sPWMConfig.OCNPolarity    = TIM_OCNPolarity_HIGH;
169  sPWMConfig.OCIdleState    = TIM_OCIdleState_SET;
170  sPWMConfig.OCNIdleState   = TIM_OCIdleState_RESET;
171
172  /* Set the pulse value for channel 1 */
173  sPWMConfig.Pulse = PULSE1_VALUE;
174  if(HAL_TIM_PWM_ConfigChannel(&TimHandle, &sPWMConfig, TIM_CHANNEL_1) != HAL_OK)
175  {
176      /* Configuration Error */
177      Error_Handler();
178  }
179
180  /* Set the pulse value for channel 2 */
181  sPWMConfig.Pulse = PULSE2_VALUE;
182  if(HAL_TIM_PWM_ConfigChannel(&TimHandle, &sPWMConfig, TIM_CHANNEL_2) != HAL_OK)
183  {
184      /* Set the pulse value for channel 3 */
185      sPWMConfig.Pulse = PULSE3_VALUE;
186      if(HAL_TIM_PWM_ConfigChannel(&TimHandle, &sPWMConfig, TIM_CHANNEL_3) != HAL_OK)
187      {
188
189          /* Set the Break feature & Dead time */
190          sBreakConfig.BreakState      = TIM_BREAK_ENABLE;
191          sBreakConfig.DeadTime        = 11;
192          sBreakConfig.OffStateRunMode = TIM_OSSR_ENABLE;
193          sBreakConfig.OffStateIDLEMode = TIM_OSSI_ENABLE;
194          sBreakConfig.LockLevel      = TIM_LOCKLEVEL_1;
195          sBreakConfig.BreakPolarity  = TIM_BREAKPOLARITY_HIGH;
196          sBreakConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_ENABLE;
197
198          if(HAL_TIMEx_ConfigBreakDeadTime(&TimHandle, &sBreakConfig) != HAL_OK)
199          {
200              /* Configuration Error */
201              Error_Handler();
202          }
203      }
204  }
```

步骤 2 配置 PWM 通道占空比等参数，另外还设置了互补通道的死区时间、参数锁定等。

```
211  /*##-3- Start PWM signals generation #####
212  /* Start channel 1 */
213  if(HAL_TIM_PWM_Start(&TimHandle, TIM_CHANNEL_1) != HAL_OK)
214  {
215      /* Starting Error */
216      Error_Handler();
217  }
218  /* Start channel 1N */
219  if(HAL_TIMEx_PWMN_Start(&TimHandle, TIM_CHANNEL_1) != HAL_OK)
220  {
```



```

225  /* Start channel 2 */
226  if(HAL_TIM_PWM_Start(&TimHandle, TIM_CHANNEL_2) != HAL_OK)
227  {
231  /* Start channel 2N */
232  if(HAL_TIMEx_PWMN_Start(&TimHandle, TIM_CHANNEL_2) != HAL_OK)
233  {
238  /* Start channel 3 */
239  if(HAL_TIM_PWM_Start(&TimHandle, TIM_CHANNEL_3) != HAL_OK)
240  {
244  /* Start channel 3N */
245  if(HAL_TIMEx_PWMN_Start(&TimHandle, TIM_CHANNEL_3) != HAL_OK)
246  {
251  /* Infinite loop */
252  while (1)
253  {
254  }
255  }

```

步骤 3 就是启动各个 PWM 通道和互补通道的输出。

12. 另外 7 个例程只作简要说明

①TIM_6Steps 例程:

演示如何用高级定时器输出 6 步 PWM 信号，这个在无刷电机控制中使用比较多。

readme.txt 文件给出了例程的说明，并给出了 6 步状态表，以及各通道的波形示意图。

readme.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

This example shows how to configure the TIM1 peripheral to generate 6 Steps. The STM32F4xx TIM1 peripheral offers the possibility to program in advance the configuration for the next TIM1 outputs behaviour (step) and change the configuration of all the channels at the same time. This operation is possible when the COM (commutation) event is used.

The COM event can be generated by software by setting the COM bit in the TIM1_EGR register or by hardware (on TRC rising edge). In this example, a software COM event is generated each 1 ms: using the SysTick interrupt.

The TIM1 is configured in Timing Mode, each time a COM event occurs, a new TIM1 configuration will be set in advance. Only changed states are programmed.

The break Polarity is used at High level.

The following Table describes the TIM1 Channels states:

```
@verbatim
          | Step1 | Step2 | Step3 | Step4 | Step5 | Step6 | |
|---|---|---|---|---|---|---|
|Channel1| 1(PWM)|  0   |  0   |  0   |  0   | 1(PWM)|
|Channel1N|  0   |  0   | 1(PWM)| 1(PWM)|  0   |  0   |
|Channel2|  0   |  0   |  0   | 1(PWM)| 1(PWM)|  0   |
|Channel2N| 1(PWM)| 1(PWM)|  0   |  0   |  0   |  0   |
|Channel3|  0   | 1(PWM)| 1(PWM)|  0   |  0   |  0   |
|Channel3N|  0   |  0   |  0   |  0   | 1(PWM)| 1(PWM)|
-----|-----|-----|-----|-----|-----|

```

Channel1 (PA.08)

Channel1N (PB.13)

Channel2 (PE.11)

Channel2N (PB.14)

Channel3 (PE.13)

Channel3N (PB.15)

②TIM_7PWMOutput 例程:

演示如何用定时器输出 7 路 PWM 信号。它们的频率是相同的，且每一对互补输出引脚的占空比是联动的。

```
readme.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
This example shows how to configure the TIM1 peripheral to generate 7 PWM signals
with 4 different duty cycles (50%, 37.5%, 25% and 12.5%).

TIM1CLK = SystemCoreClock, Prescaler = 0, TIM1 counter clock = SystemCoreClock
SystemCoreClock is set to 168 MHz.

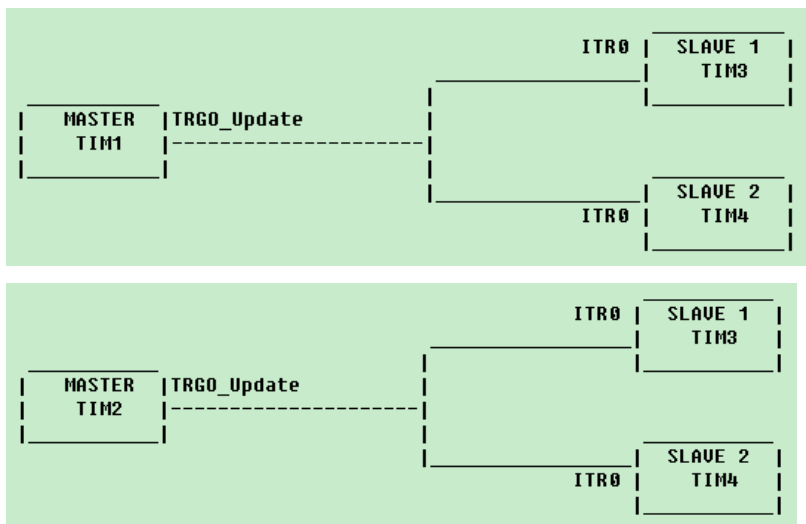
The objective is to generate 7 PWM signal at 17.57 KHz:
- TIM1_Period = (SystemCoreClock / 17570) - 1
The channel 1 and channel 1N duty cycle is set to 50%
The channel 2 and channel 2N duty cycle is set to 37.5%
The channel 3 and channel 3N duty cycle is set to 25%
The channel 4 duty cycle is set to 12.5%
The Timer pulse is calculated as follows:

- ChannelxPulse = DutyCycle * (TIM1_Period - 1) / 100

The TIM1 waveforms can be displayed using an oscilloscope.
```

③TIM_Synchronization 和④TIM_ParallelSynchro 例程:

这两个例程是差不多的，都是演示定时器并行同步。它们的 readme.txt 都有一个 TIM 连接示意图。



两个例程都是用一个定时器触发 TIM3 和 TIM4，使这两个定时器同步。

⑤TIM_CascadeSynchro 例程:

演示如何将 TIM 串联使用，将一个 TIM 的事件输出作为另一个 TIM 的触发输入。

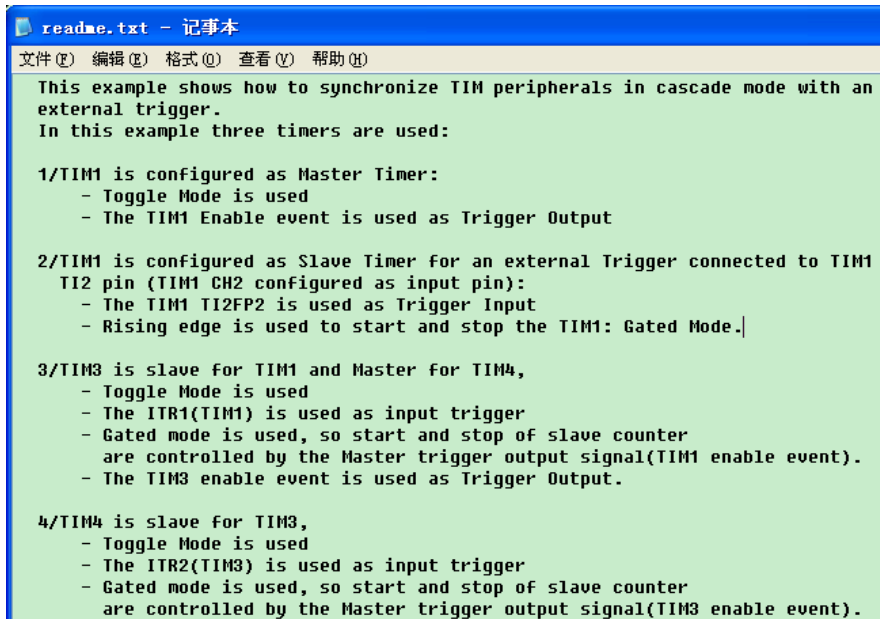
```
readme.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
1/ TIM2 is configured as Master Timer:
- PWM Mode is used
- The TIM2 Update event is used as Trigger Output

2)TIM3 is slave for TIM2 and Master for TIM4,
- PWM Mode is used
- The ITR1(TIM2) is used as input trigger
- Gated mode is used, so start and stop of slave counter
  are controlled by the Master trigger output signal(TIM2 update event).
- The TIM3 Update event is used as Trigger Output.

3)TIM4 is slave for TIM3,
- PWM Mode is used
- The ITR2(TIM3) is used as input trigger
- Gated mode is used, so start and stop of slave counter are controlled by the
  Master trigger output signal(TIM3 update event).
```

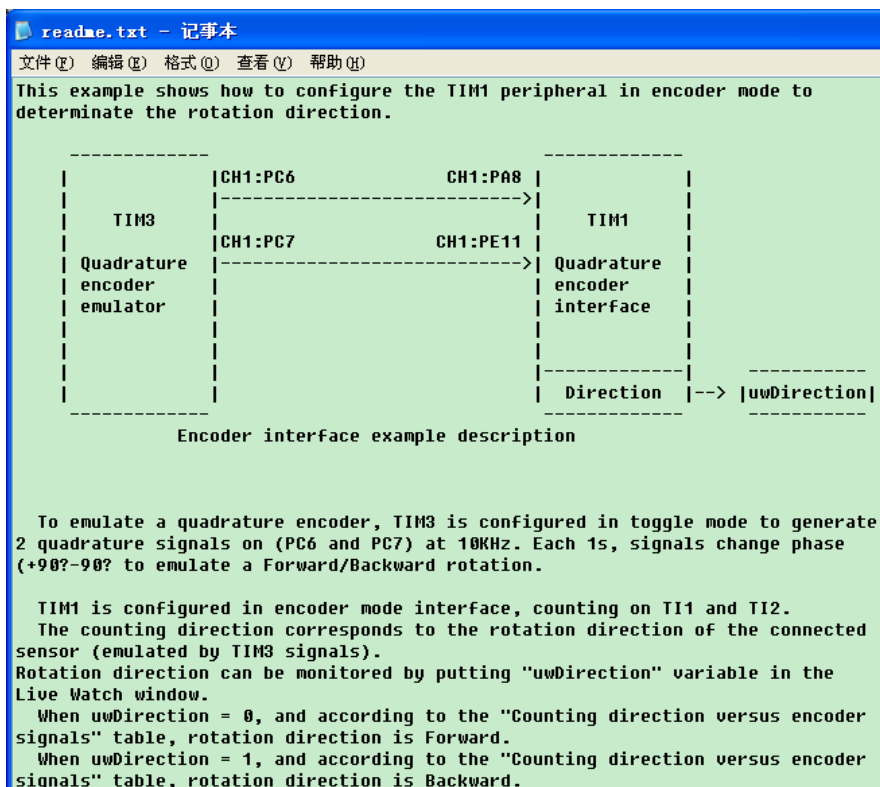
⑥TIM_ExtTriggerSynchro 例程:

和 TIM_CascadeSynchro 例程相似，只是 TIM1 使用外部信号触发进行启动和停止。



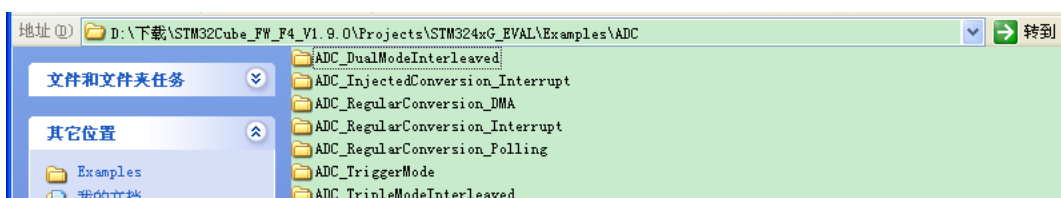
⑦TIM Encoder 例程:

演示编码器接口的使用。



第四章：ADC

ADC 共有 7 个例程。



1.ADC_RegularConversion_Polling

这是最简单和常用的方法，即查询。使用方法非常简单。

```
88  /*##-1- Configure the ADC peripheral #####
89  AdcHandle.Instance          = ADCx;
90
91  AdcHandle.Init.ClockPrescaler      = ADC_CLOCKPRESCALER_PCLK_DIV2;
92  AdcHandle.Init.Resolution          = ADC_RESOLUTION_12B;
93  AdcHandle.Init.ScanConvMode        = DISABLE;
94  AdcHandle.Init.ContinuousConvMode  = DISABLE;
95  AdcHandle.Init.DiscontinuousConvMode = DISABLE;
96  AdcHandle.Init.NbrOfDiscConversion = 0;
97  AdcHandle.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
98  AdcHandle.Init.ExternalTrigConv    = ADC_EXTERNALTRIGCONV_T1_CC1;
99  AdcHandle.Init.DataAlign           = ADC_DATAALIGN_RIGHT;
100 AdcHandle.Init.NbrOfConversion      = 1;
101 AdcHandle.Init.DMAContinuousRequests = DISABLE;
102 AdcHandle.Init.EOCSelection         = DISABLE;
103
104 if(HAL_ADC_Init(&AdcHandle) != HAL_OK)
105 {
106     /* Initialization Error */
107     Error_Handler();
108 }
109
110 /*##-2- Configure ADC regular channel #####
111 sConfig.Channel      = ADCx_CHANNEL;
112 sConfig.Rank          = 1;
113 sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
114 sConfig.Offset        = 0;
115
116 if(HAL_ADC_ConfigChannel(&AdcHandle, &sConfig) != HAL_OK)
117 {
118     /* Channel Configuration Error */
119     Error_Handler();
120 }
```

步骤 1 配置 ADC 外设，步骤 2 配置 ADC 通道。这两个步骤在 CubeMX 中进行配置即可，不需要用户添加代码。

```
123 /*##-3- Start the conversion process #####*/
124 if(HAL_ADC_Start(&AdcHandle) != HAL_OK)
125 {
126     /* Start Conversation Error */
127     Error_Handler();
128 }
129
130 /*##-4- Wait for the end of conversion #####*/
131 /* Before starting a new conversion, you need to check the current state of
132    the peripheral; if it is busy you need to wait for the end of current
133    conversion before starting a new one.
134    For simplicity reasons, this example is just waiting till the end of the
135    conversion, but application may perform other tasks while conversion
136    operation is ongoing. */
137 HAL_ADC_PollForConversion(&AdcHandle, 10);
138
139 /* Check if the continuous conversion of regular channel is finished */
140 if(HAL_ADC_GetState(&AdcHandle) == HAL_ADC_STATE_EOC_REG)
141 {
142     /*##-5- Get the converted value of regular channel #####*/
143     uhADCxConvertedValue = HAL_ADC_GetValue(&AdcHandle);
144 }
145
146 /* Infinite loop */
147 while (1)
148 {
149 }
150 }
```

步骤 3~5 是要用户添加的。步骤也很简单，就是调用划线的那几个函数：启动转换、等待转换结束、查询状态、获取转换值。

2.ADC_RegularConversion_Interrupt

演示 ADC 转换完成中断的使用。

```
105 if(HAL_ADC_Init(&AdcHandle) != HAL_OK)
106 {
107     /* Initialization Error */
108     Error_Handler();
109 }
110
111 /*##-2- Configure ADC regular channel #####
112 sConfig.Channel = ADCx_CHANNEL;
113 sConfig.Rank = 1;
114 sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
115 sConfig.Offset = 0;
116
117 if(HAL_ADC_ConfigChannel(&AdcHandle, &sConfig) != HAL_OK)
118 {
119     /* Channel Configuration Error */
120     Error_Handler();
121 }
122
123 /*##-3- Start the conversion process and enable interrupt
124 if(HAL_ADC_Start_IT(&AdcHandle) != HAL_OK)
125 {
126     /* Start Conversation Error */
127     Error_Handler();
128 }
129
130 /* Infinite loop */
131 while (1)
132 {
133 }
134 }
```

步骤非常简单。步骤 1、2 配置 ADC 外设和 ADC 通道。步骤 3 调用 HAL_ADC_Start_IT()函数启动 ADC 转换并使能转换完成中断。

```
218 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* AdcHandle)
219 {
220     /* Get the converted value of regular channel */
221     uhADCxConvertedValue = HAL_ADC_GetValue(AdcHandle);
222 }
```

然后在 ADC 转换完成中断回调函数中，获取转换值。

3.ADC_RegularConversion_DMA

演示 ADC 转换+DMA 的使用。

```
106 if(HAL_ADC_Init(&AdcHandle) != HAL_OK)
107 {
108     /* Initialization Error */
109     Error_Handler();
110 }
111
112 /*##-2- Configure ADC regular channel #####
113 /* Note: Considering IT occurring after each number of size of
114 /* "uhADCxConvertedValue" ADC conversions (IT by DMA end
115 /* of transfer), select sampling time and ADC clock with sufficient
116 /* duration to not create an overhead situation in IRQHandler.
117 sConfig.Channel = ADCx_CHANNEL;
118 sConfig.Rank = 1;
119 sConfig.SamplingTime = ADC_SAMPLETIME_28CYCLES;
120 sConfig.Offset = 0;
121
122 if(HAL_ADC_ConfigChannel(&AdcHandle, &sConfig) != HAL_OK)
123 {
124     /* Channel Configuration Error */
125     Error_Handler();
126 }
127
128 /*##-3- Start the conversion process and enable interrupt #####
129 if(HAL_ADC_Start_DMA(&AdcHandle, (uint32_t*)&uhADCxConvertedValue, 1) != HAL_OK)
130 {
131     /* Start Conversation Error */
132     Error_Handler();
133 }
```


步骤和 ADC_RegularConversion_Interrupt 相同。步骤 3 中调用的是 HAL_ADC_Start_DMA()函数，转换完成后转换值会通过 DMA 传输到变量 uhADCxConvertedValue 中，并产生一次转换完成中断。

```
223 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* AdcHandle)
224 {
225     /* Turn LED1 on: Transfer process is correct */
226     BSP_LED_On(LED1);
227 }
```

在本例中，ADC 转换完成中断回调函数没有作实质性的数据处理。实际应用中，可在此添加对转换值的处理代码。

4. ADC_TriggerMode

演示用 TIM 触发 ADC 转换的使用。

```
90  /*##-1- TIM8 Peripheral Configuration #####
91  TIM_Config();
92
93  /*##-2- Configure the ADC3 peripheral #####
94  ADC_Config();
95
96  /*##-3- Start the conversion process and enable interrupt #
97  if(HAL_ADC_Start_IT(&AdcHandle) != HAL_OK)
98  {
99      /* Start Conversation Error */
100     Error_Handler();
101 }
102
103 /*##-4- TIM8 counter enable #####
104 if(HAL_TIM_Base_Start(&htim) != HAL_OK)
105 {
106     /* Counter Enable Error */
107     Error_Handler();
108 }
109
110 /* Infinite loop */
111 while (1)
112 {
113 }
114 }
```

步骤 1 配置 TIM8，设置 UPDATE 为 OutputTrigger 信号。

步骤 2 配置 ADC，设置 T8_TRGO 为转换触发信号。

步骤 3 启动 ADC 转换，并使能中断。

步骤 4 启动 TIM8。

运行时，每次 TIM8 产生更新信号，就触发一次 ADC 转换，并产生一次中断。

```
277 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* AdcHandle)
278 {
279     /* Get the converted value of regular channel */
280     uhADCxConvertedValue = HAL_ADC_GetValue(AdcHandle);
281 }
```

在 ADC 转换完成中断回调函数中，获取转换值。

5. ADC_InjectedConversion_Interrupt

演示 ADC 注入组转换的使用。

```
88  /*##-1- Configure the ADC peripheral #####*/
89  ADC_Config();
90
91  /*##-2- Start the conversion process and enable interrupt for regular channel #*/
92  if(HAL_ADC_Start_IT(&AdcHandle) != HAL_OK)
93  {
94      /* Start Conversation Error */
95      Error_Handler();
96  }
```

步骤 1 配置 ADC。在 ADC_Config()函数中，还配置了一个规则通道和一个注入通道。

步骤 2 是启动规则通道的转换，并使能中断。

```

98  /*##-3- Wait one second before starting injected conversion #####*/
99  HAL_Delay(1000);
100
101  /*##-4- Start the conversion process and enable interrupt for injected channel */
102  if(HAL_ADCEx_InjectedStart_IT(&AdcHandle) != HAL_OK)
103  {
104      /* Start Conversation Error */
105      Error_Handler();
106  }
107
108  /* Infinite loop */
109  while (1)
110  {
111  }
112 }

```

步骤 3 是延时 1 秒钟。

步骤 4 启动注入通道的转换，并使能中断。

```

257 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* AdcHandle)
258 {
259     /* Get the converted value of regular channel */
260     uhADCxConvertedRegValue = HAL_ADC_GetValue(AdcHandle);
261 }
270 void HAL_ADCEx_InjectedConvCpltCallback(ADC_HandleTypeDef* AdcHandle)
271 {
272     /* Get the converted value of injected channel */
273     uhADCxConvertedInjValue = HAL_ADCEx_InjectedGetValue(AdcHandle, ADC_INJECTED_RANK_1);
274 }

```

规则通道和注入通道的转换值，分别在不同的中断函数中获取。

6. ADC_DualModeInterleaved

演示 ADC1 和 ADC2 交错采样的应用。

```

90  /*##-1- Configure ADC1 and ADC2 peripherals #####*/
91  ADC_Config();
92
93  /*##-2- Enable ADC2 #####*/
94  if(HAL_ADC_Start(&AdcHandle2) != HAL_OK)
95  {
96      /* Start Error */
97      Error_Handler();
98  }
99
100  /*##-3- Start ADC1 and ADC2 multimode conversion process and enable DMA #####*/
101  if(HAL_ADCEx_MultiModeStart_DMA(&AdcHandle1, (uint32_t*)&uhADCDualConvertedValue, 1) != HAL_OK)
102  {
103      /* Start Error */
104      Error_Handler();
105  }
106
107  /* Infinite loop */
108  while (1)
109  {
110  }

```

基本步骤很简单。步骤 1 配置 ADC1 和 ADC2 外设。

```

267  /*##-5- Configure Multimode #####*/
268  mode.Mode = ADC_DUALMODE_INTERL;
269  mode.DMAAccessMode = ADC_DMAACCESSMODE_3;
270  mode.TwoSamplingDelay = ADC_TWOSAMPLINGDELAY_6CYCLES;
271  if(HAL_ADCEx_MultiModeConfigChannel(&AdcHandle1, &mode) != HAL_OK)
272  {
273      /* Channel Configuration Error */
274      Error_Handler();
275  }
276 }

```

在 ADC_Config() 函数中，除了给 ADC1 和 ADC2 分别配置了一个规则通道，还有一个重要步骤：多 ADC 模式配置，即多个 ADC 联合采样，这里配置为双 ADC 交错模式。

步骤 2 启动 ADC2。

步骤 3 启动 ADC1 和 ADC2 多模转换，使能 DMA 并使能中断。

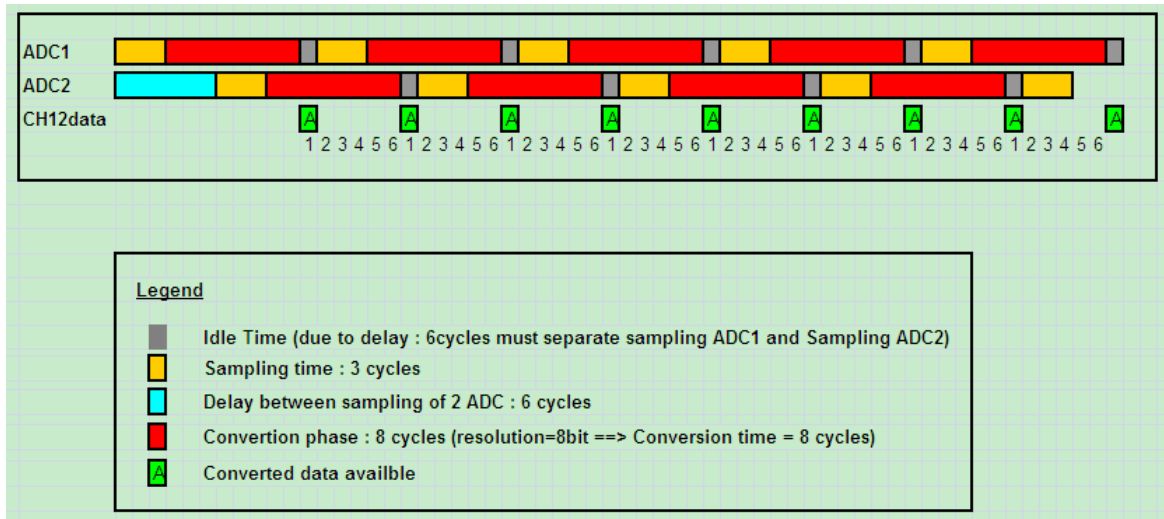
```

285 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
286 {
287     /* Turn LED1 on: Transfer process is correct */
288     BSP_LED_On(LED1);
289 }

```

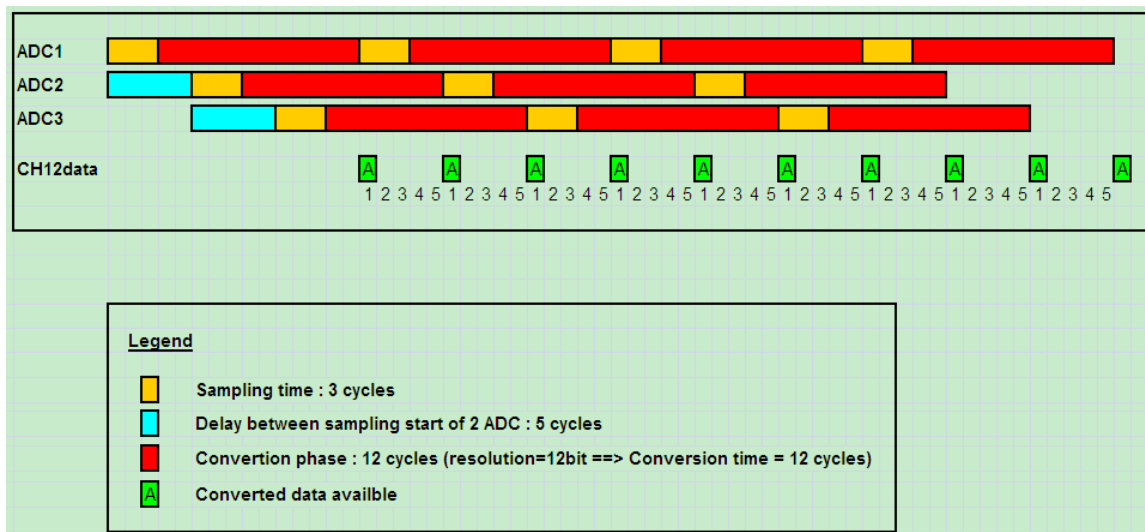
在本例的中断回调函数中，没有作数据处理。

本例程中 ADC1 和 ADC2 联合工作的过程，可以在..\ADC\ADC_TripleModeInterleaved 文件夹的 simulation.xls 文件中看到示意图。



7. ADC_TripleModeInterleaved

该例程和 ADC_DualModeInterleaved 类似，只是增加了 ADC3。其工作过程如下图所示。



注意，ADC_Config()函数中，步骤 7 配置通道时用的模式参数为 ADC_DUALMODE_INTERL，即双模交错。本例要演示的是 3 个 ADC 交错采样，因此应为 ADC_TRIPLEMODE_INTERL。本人为进行实测验证，在使用本例做参考时，请先进行验证。

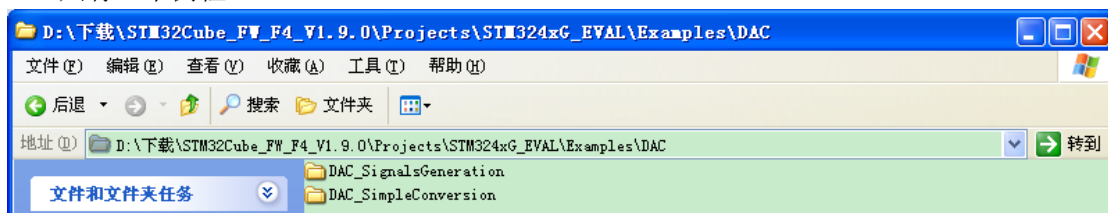
```

305 /*##-7- Configure Multimode #####
306 mode.Mode = ADC_DUALMODE_INTERL;
307 mode.DMAAccessMode = ADC_DMAACCESSMODE_2;
308 mode.TwoSamplingDelay = ADC_TWOSAMPLINGDELAY_5CYCLES;
309
310 if(HAL_ADCEx_MultiModeConfigChannel(&AdcHandle1, &mode) != HAL_OK)
311 {
312     /* Multimode Configuration Error */
313     Error_Handler();
314 }
315 }

```

第五章： DAC

DAC 只有 2 个例程。



1. DAC_SimpleConversion

演示最基本的 DAC 转换的使用。

```
84  /*##-1- Configure the DAC peripheral #####*/
85  DacHandle.Instance = DACx;
86
87  if(HAL_DAC_Init(&DacHandle) != HAL_OK)
88  {
89      /* Initialization Error */
90      Error_Handler();
91  }
92
93  /*##-2- Configure DAC channel1 #####*/
94  sConfig.DAC_Trigger = DAC_TRIGGER_NONE;
95  sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_DISABLE;
96
97  if(HAL_DAC_ConfigChannel(&DacHandle, &sConfig, DACx_CHANNEL) != HAL_OK)
98  {
99      /* Channel configuration Error */
100     Error_Handler();
101 }
```

步骤 1 配置 DAC 外设，步骤 2 配置 DAC 通道。

```
103 /*##-3- Set DAC Channel1 DHR register #####*/
104 if(HAL_DAC_SetValue(&DacHandle, DACx_CHANNEL, DAC_ALIGN_8B_R, 0xFF) != HAL_OK)
105 {
106     /* Setting value Error */
107     Error_Handler();
108 }
109
110 /*##-4- Enable DAC Channel1 #####*/
111 if(HAL_DAC_Start(&DacHandle, DACx_CHANNEL) != HAL_OK)
112 {
113     /* Start Error */
114     Error_Handler();
115 }
116
117 /* Infinite loop */
118 while (1)
119 {
120 }
121 }
```

步骤 3 设置转换数据，步骤 4 启动 DAC 转换。

2. DAC_SignalsGeneration

演示用 DAC 产生信号。

该例程中包含了两种信号的产生方法，一种是使用硬件直接产生三角波，另一种是通过 TIM+DMA+ADC 的实现阶梯波。后一种是产生波形的通用方法。

```

94  /*##-1- Configure the DAC peripheral #####
95  DacHandle.Instance = DAC;
96
97  /*##-2- Configure the TIM peripheral #####
98  TIM6_Config();
99
100  /* Infinite loop */
101  while (1)
102  {
103      /* If the Key is pressed */
104      if (ubKeyPressed != RESET)
105      {
106          HAL_DAC_DeInit(&DacHandle);
107
108          /* select waves forms according to the Key Button status */
109          if (ubSelectedWavesForm == 1)
110          {
111              /* The triangle wave has been selected */
112
113              /* Triangle Wave generator -----
114              DAC_Ch1_TriangleConfig();
115          }
116          else
117          {
118              /* The escalator wave has been selected */
119
120              /* Escalator Wave generator -----
121              DAC_Ch1_EscalatorConfig();
122          }
123
124          ubKeyPressed = RESET;
125      }
126  }
127  }

```

本例程中，用按键选择输出的波形。两种波形都需要 TIM 触发 DAC 的转换。

DAC_Ch1_TriangleConfig()函数的关键步骤，就是调用 HAL_DACEx_TriangleWaveGenerate()进行三角波发生配置。

```

255  /*##-3- DAC channel2 Triangle Wave generation configuration #####*/
256  if(HAL_DACEx_TriangleWaveGenerate(&DacHandle, DACx_CHANNEL1, DAC_TRIANGLEAMPLITUDE_1023) != HAL_OK)
257  {
258      /* Triangle wave generation Error */
259      Error_Handler();
260  }

```

DAC_Ch1_EscalatorConfig()函数才是本例程的重点。

```

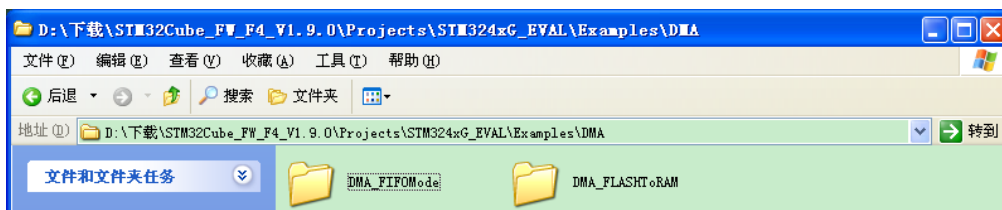
204  static void DAC_Ch1_EscalatorConfig(void)
205  {
206      /*##-1- Initialize the DAC peripheral #####*/
207      if(HAL_DAC_Init(&DacHandle) != HAL_OK)
208      {
209          /* Initialization Error */
210          Error_Handler();
211      }
212
213      /*##-1- DAC channel1 Configuration #####*/
214      sConfig.DAC_Trigger = DAC_TRIGGER_T6_TRGO;
215      sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE;
216
217      if(HAL_DAC_ConfigChannel(&DacHandle, &sConfig, DACx_CHANNEL1) != HAL_OK)
218      {
219          /* Channel configuration Error */
220          Error_Handler();
221      }
222
223      /*##-2- Enable DAC Channel1 and associated DMA #####*/
224      if(HAL_DAC_Start_DMA(&DacHandle, DACx_CHANNEL1, (uint32_t*)aEscalator8bit, 6, DAC_ALIGN_8B_R) != HAL_OK)
225      {
226          /* Start DMA Error */
227          Error_Handler();
228      }
229  }

```

通过配置 TIM6 的 TRGO 作为 DAC 转换触发信号，并配置 DMA 传输，在触发转换时就循环将 aEscalator8bit[]的元素传递到 DAC 转换输出。

第六章： DMA

DMA 只有 2 个例程，都是 MemToMem 的应用，和外设相关的 DMA 例程放在了外设例程集文件夹中。



1. DMA_FLASHToRAM

演示用 DMA 将数据从 FLASH 传输至 RAM。关键代码是 DMA_Config()函数的最后几个步骤。

```
148  /*##-4- Select Callbacks functions called after Transfer complete and Transfer error */
149  DmaHandle.XferCpltCallback = TransferComplete;
150  DmaHandle.XferErrorCallback = TransferError;
151
152  /*##-5- Initialize the DMA stream #####*/
153  if (HAL_DMA_Init(&DmaHandle) != HAL_OK)
154  {
155      /* Initialization Error */
156      Error_Handler();
157  }
158
159  /*##-6- Configure NVIC for DMA transfer complete/error interrupts #####*/
160  /* Set Interrupt Group Priority */
161  HAL_NVIC_SetPriority(DMA_STREAM_IRQ, 0, 0);
162
163  /* Enable the DMA STREAM global Interrupt */
164  HAL_NVIC_EnableIRQ(DMA_STREAM_IRQ);
165
166  /*##-7- Start the DMA transfer using the interrupt mode #####*/
167  /* Configure the source, destination and buffer size DMA fields and Start DMA Stream transfer */
168  /* Enable All the DMA interrupts */
169  if (HAL_DMA_Start_IT(&DmaHandle, (uint32_t)&aSRC_Const_Buffer, (uint32_t)&aDST_Buffer, BUFFER_SIZE)
170  {
171      /* Transfer Error */
172      Error_Handler();
173  }
174 }
```

要特别注意的是，步骤 4 和步骤 7 是需要用户添加的代码，DMA_Config()函数的其他代码都可由 CubeMX 配置生成。

步骤 4 是指定回调函数，DMA 的回调函数有 4 个，可查看 stm32f4xx_hal_dma.h 中 DMA_HandleTypeDef 结构体定义。

```
void (* XferCpltCallback)( struct __DMA_HandleTypeDef * hdma);
void (* XferHalfCpltCallback)( struct __DMA_HandleTypeDef * hdma);
void (* XferM1CpltCallback)( struct __DMA_HandleTypeDef * hdma);
void (* XferErrorCallback)( struct __DMA_HandleTypeDef * hdma);
```

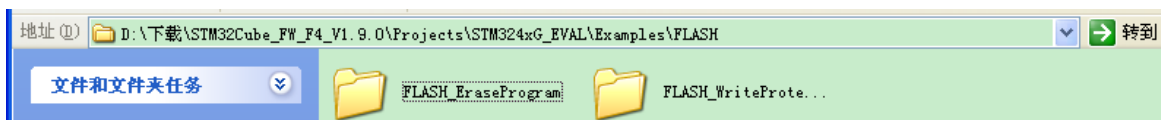
步骤 7 设置源和目标指针、传输大小，启动 DMA 传输，并使能中断。

2. DMA_FIFOmode

该例程和 DMA_FLASHToRAM 功能完全相同，只是在配置的时候使能了 FIFO。在 CubeMX 配置 DMA 的 MemToMem 模式时，默认使能 FIFO，且无法禁止。

第七章： FLASH

FLASH 只有 2 个例程，下面只分析最常用的擦写应用例程。



FLASH_EraseProgram

演示 FLASH 的擦除和写入操作。看 main 函数：

```
95  /* Unlock the Flash to enable the flash control register access */
96  HAL_FLASH_Unlock();
97
98  /* Erase the user Flash area
99   (area defined by FLASH_USER_START_ADDR and FLASH_USER_END_ADDR)
100
101   /* Get the 1st sector to erase */
102   FirstSector = GetSector(FLASH_USER_START_ADDR);
103   /* Get the number of sector to erase from 1st sector*/
104   NbOfSectors = GetSector(FLASH_USER_END_ADDR) - FirstSector + 1;
105
106   /* Fill EraseInit structure*/
107   EraseInitStruct.TypeErase = FLASH_TYPEERASE_SECTORS;
108   EraseInitStruct.VoltageRange = FLASH_VOLTAGE_RANGE_3;
109   EraseInitStruct.Sector = FirstSector;
110   EraseInitStruct.NbSectors = NbOfSectors;
111   if(HAL_FLASHEx_Erase(&EraseInitStruct, &SectorError) != HAL_OK)
112   {
113     /*
114     Error occurred while sector erase.
115     User can add here some code to deal with this error.
116     SectorError will contain the faulty sector and then to know t
117     user can call function 'HAL_FLASH_GetError()'
118     */
119     while (1)
120     {
121       BSP_LED_On(LED3);
122     }
123   }
```

步骤 1 是解锁 FLASH，这是擦除或写入操作 FLASH 的必要步骤。

步骤 2 是擦除，HAL_FLASHEx_Erase()函数使用了一个结构体，该结构体指定了擦除的起始扇区号、扇区数量等参数。

```
125  /* Note: If an erase operation in Flash memory also
126   you have to make sure that these data are rewritt
127   execution. If this cannot be done safely, it is r
128   DCRST and ICRST bits in the FLASH_CR register. */
129   __HAL_FLASH_DATA_CACHE_DISABLE();
130   __HAL_FLASH_INSTRUCTION_CACHE_DISABLE();
131
132   __HAL_FLASH_DATA_CACHE_RESET();
133   __HAL_FLASH_INSTRUCTION_CACHE_RESET();
134
135   __HAL_FLASH_INSTRUCTION_CACHE_ENABLE();
136   __HAL_FLASH_DATA_CACHE_ENABLE();
```

步骤 3 是涉及到缓存操作安全的，阅读注释说明即可。如果没有涉及到缓存，可以忽略。

```
138  /* Program the user Flash area word by word
139   (area defined by FLASH_USER_START_ADDR and FLASH_USER_END_ADDR) *****/
140
141   Address = FLASH_USER_START_ADDR;
142
143   while (Address < FLASH_USER_END_ADDR)
144   {
145     if (HAL_FLASH_Program(FLASH_TYPEPROGRAM_WORD, Address, DATA_32) == HAL_OK)
146     {
147       Address = Address + 4;
148     }
149     else
150     {
151       /* Error occurred while writing data in Flash memory.
152       User can add here some code to deal with this error */
153       while (1)
154       {
155         BSP_LED_On(LED3);
156       }
157     }
158   }
159
160  /* Lock the Flash to disable the flash control register access (recommended
161   to protect the FLASH memory against possible unwanted operation) *****/
162  HAL_FLASH_Lock();
163
```

步骤 4 是按 Word 写入 FLASH。这里写入的数据都是 DATA_32，其值是 0x12345678。

步骤 5 是锁定 FLASH。

```

164 /* Check if the programmed data is OK
165     MemoryProgramStatus = 0: data programmed correctly
166     MemoryProgramStatus != 0: number of words not programmed correctly */
167 Address = FLASH_USER_START_ADDR;
168 MemoryProgramStatus = 0;
169
170 while (Address < FLASH_USER_END_ADDR)
171 {
172     data32 = *((__IO uint32_t*)Address);
173
174     if (data32 != DATA_32)
175     {
176         MemoryProgramStatus++;
177     }
178
179     Address = Address + 4;
180 }
181
182 /* Check if there is an issue to program data */
183 if (MemoryProgramStatus == 0)
184 {
185     /* No error detected. Switch on LED1 */
186     BSP_LED_On(LED1);
187 }
188 else
189 {
190     /* Error detected. Switch on LED2 */
191     BSP_LED_On(LED2);
192 }
193
194 /* Infinite loop */
195 while (1)
196 {
197 }
198 }

```

步骤 6 是回读数据，验证写入是否正确。在应用中，此步骤是可选的。

要特别注意 GetSector() 函数，其功能是获取某个 FLASH 地址所在扇区号。该函数用了很多个 if...else 语句来实现，是因为 STM32F4xx 的 FLASH 扇区大小不一致。可以查看 F4 的资料，或者直接看本例的 main.h 文件：

```

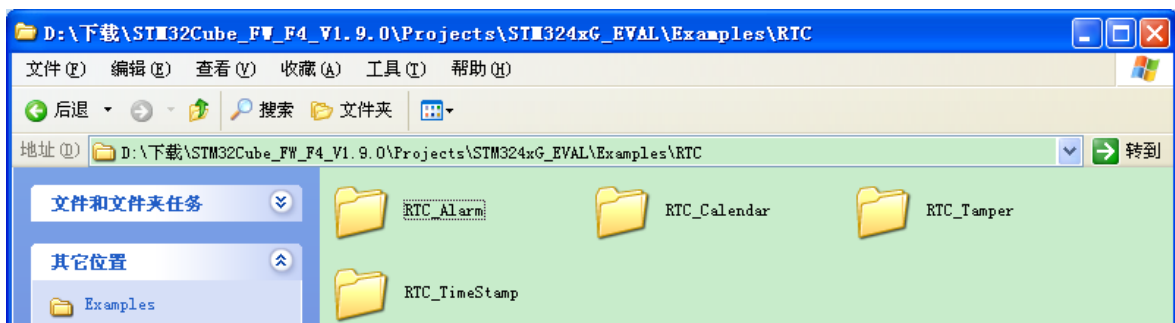
48 /* Base address of the Flash sectors */
49 #define ADDR_FLASH_SECTOR_0 ((uint32_t)0x08000000) /* Base @ of Sector 0, 16 Kbytes */
50 #define ADDR_FLASH_SECTOR_1 ((uint32_t)0x08004000) /* Base @ of Sector 1, 16 Kbytes */
51 #define ADDR_FLASH_SECTOR_2 ((uint32_t)0x08008000) /* Base @ of Sector 2, 16 Kbytes */
52 #define ADDR_FLASH_SECTOR_3 ((uint32_t)0x0800C000) /* Base @ of Sector 3, 16 Kbytes */
53 #define ADDR_FLASH_SECTOR_4 ((uint32_t)0x08010000) /* Base @ of Sector 4, 64 Kbytes */
54 #define ADDR_FLASH_SECTOR_5 ((uint32_t)0x08020000) /* Base @ of Sector 5, 128 Kbytes */
55 #define ADDR_FLASH_SECTOR_6 ((uint32_t)0x08040000) /* Base @ of Sector 6, 128 Kbytes */
56 #define ADDR_FLASH_SECTOR_7 ((uint32_t)0x08060000) /* Base @ of Sector 7, 128 Kbytes */
57 #define ADDR_FLASH_SECTOR_8 ((uint32_t)0x08080000) /* Base @ of Sector 8, 128 Kbytes */
58 #define ADDR_FLASH_SECTOR_9 ((uint32_t)0x080A0000) /* Base @ of Sector 9, 128 Kbytes */
59 #define ADDR_FLASH_SECTOR_10 ((uint32_t)0x080C0000) /* Base @ of Sector 10, 128 Kbytes */
60 #define ADDR_FLASH_SECTOR_11 ((uint32_t)0x080E0000) /* Base @ of Sector 11, 128 Kbytes */

```

F4 把最大 1M 字节的 FLASH 划成了不均等的 12 个扇区，使用的时候一定要注意。

第八章： RTC

RTC 有 4 个例程，下面只分析最常用的日历例程。



RTC_Calendar

演示如何使用 RTC 实现一个时钟/日历。看 main 函数：

```
96  /*##-1- Configure the RTC peripheral #####*/
97  RtcHandle.Instance = RTC;
98
99  /* Configure RTC prescaler and RTC data registers */
100 /* RTC configured as follow:
101    - Hour Format      = Format 24
102    - Asynch Prediv    = Value according to source clock
103    - Synch Prediv     = Value according to source clock
104    - OutPut           = Output Disable
105    - OutPutPolarity   = High Polarity
106    - OutPutType       = Open Drain */
107  RtcHandle.Init.HourFormat = RTC_HOURFORMAT_24;
108  RtcHandle.Init.AsynchPrediv = RTC_ASYNC_PREDIV;
109  RtcHandle.Init.SynchPrediv = RTC_SYNC_PREDIV;
110  RtcHandle.Init.OutPut = RTC_OUTPUT_DISABLE;
111  RtcHandle.Init.OutPutPolarity = RTC_OUTPUT_POLARITY_HIGH;
112  RtcHandle.Init.OutPutType = RTC_OUTPUT_TYPE_OPENDRAIN;
113
114  if(HAL_RTC_Init(&RtcHandle) != HAL_OK)
115  {
116      /* Initialization Error */
117      Error_Handler();
118  }
```

步骤 1 配置 RTC 外设。(注意：这里并没有设置日期和时间。)

```
120  /*##-2- Check if Data stored in BackUp register0: No Need to reconfigure RTC*/
121  /* Read the Back Up Register 0 Data */
122  if(HAL_RTCEX_BKUPRead(&RtcHandle, RTC_BKP_DR0) != 0x32F2)
123  {
124      /* Configure RTC Calendar */
125      RTC_CalendarConfig();
126  }
127  else
128  {
129      /* Check if the Power On Reset flag is set */
130      if(__HAL_RCC_GET_FLAG(RCC_FLAG_PORRST) != RESET)
131      {
132          /* Turn on LED2: Power on reset occurred */
133          BSP_LED_On(LED2);
134      }
135      /* Check if Pin Reset flag is set */
136      if(__HAL_RCC_GET_FLAG(RCC_FLAG_PINRST) != RESET)
137      {
138          /* Turn on LED4: External reset occurred */
139          BSP_LED_On(LED4);
140      }
141      /* Clear source Reset Flag */
142      __HAL_RCC_CLEAR_RESET_FLAGS();
143  }
144
145  /* Infinite loop */
146  while (1)
147  {
148      /*##-3- Display the updated Time and Date #####*/
149      RTC_CalendarShow(aShowTime, aShowDate);
150  }
151 }
```

步骤 2 检测备份寄存器 RTC_BKP_DR0 是否为 0x32F2。

如果不是，说明 RTC 已经被复位过了，即当前 RTC 的值是不可信的，要重新设置 RTC 时间。如果是，则只清除 RESET_FLAG。

这里的 0x32F2 没有特殊含义，只是在 RTC_CalendarConfig()函数中，写入 RTC_BKP_DR0 的值也是 0x32F2。只要这两个值一致且不等于 0 即可。

步骤 3 就是读取 RTC 时间，并显示或者打印输出。

```

239 static void RTC_CalendarConfig(void)
240 {
241     RTC_DateTypeDef sdatestructure;
242     RTC_TimeTypeDef stimestructure;
243
244     /*##-1- Configure the Date #####*/
245     /* Set Date: Tuesday February 18th 2014 */
246     sdatestructure.Year = 0x14;
247     sdatestructure.Month = RTC_MONTH_FEBRUARY;
248     sdatestructure.Date = 0x18;
249     sdatestructure.WeekDay = RTC_WEEKDAY_TUESDAY;
250
251     if(HAL_RTC_SetDate(&RtcHandle,&sdatestructure,RTC_FORMAT_BCD) != HAL_OK)
252     {
253         /* Initialization Error */
254         Error_Handler();
255     }
256
257     /*##-2- Configure the Time #####*/
258     /* Set Time: 02:00:00 */
259     stimestructure.Hours = 0x02;
260     stimestructure.Minutes = 0x00;
261     stimestructure.Seconds = 0x00;
262     stimestructure.TimeFormat = RTC_HOURFORMAT12_AM;
263     stimestructure.DayLightSaving = RTC_DAYLIGHTSAVING_NONE;
264     stimestructure.StoreOperation = RTC_STOREOPERATION_RESET;
265
266     if(HAL_RTC_SetTime(&RtcHandle,&stimestructure,RTC_FORMAT_BCD) != HAL_OK)
267     {
268         /* Initialization Error */
269         Error_Handler();
270     }
271
272     /*##-3- Writes a data in a RTC Backup data Register0 #####*/
273     HAL_RTCEx_BKUPWrite(&RtcHandle,RTC_BKP_DR0,0x32F2);
274 }

```

RTC_CalendarConfig()函数就是 RTC 时间重置函数。

HAL 库对日期和时间设置分开用两个函数进行设置。设置日期用 HAL_RTC_SetDate(), 设置时间用 HAL_RTC_SetTime()。

RTC_CalendarConfig()函数的第三步是向 RTC_BKP_DR0 备份寄存器写入 0x32F2。

注意：用 CubeMX 配置生成代码时，main 函数中的步骤 1 和 RTC_CalendarConfig()函数步骤 1、2 是被放在 MX_RTC_Init()函数中的。这样就会造成，每次上电都进行日期和时间的重置。这是不符合实际应用要求的。

所以，应该参考本例程的做法，将这两部分分开处理，利用备份寄存器判断是否需要重置 RTC。

第九章： 其它例程简介

..\CRC\CRC_Example

演示如何使用 STM32 的 CRC 模块计算 CRC。

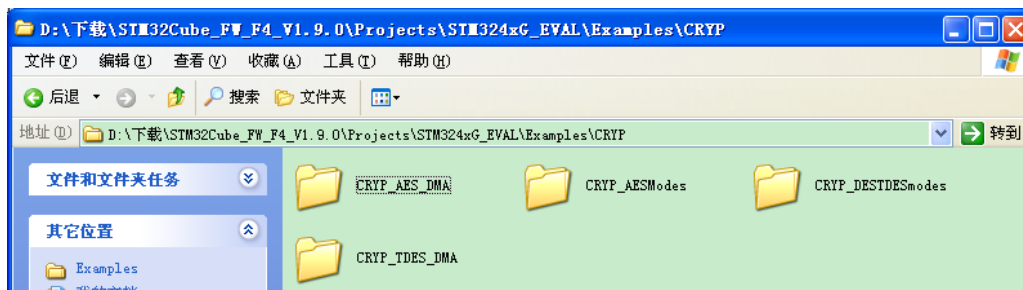
```

117 /*##-1- Configure the CRC peripheral #####*/
118 CrcHandle.Instance = CRC;
119
120 if(HAL_CRC_Init(&CrcHandle) != HAL_OK)
121 {
122     /* Initialization Error */
123     Error_Handler();
124 }
125
126 /*##-2- Compute the CRC of "aDataBuffer" #####*/
127 uwCRCValue = HAL_CRC_Accumulate(&CrcHandle, (uint32_t *)aDataBuffer, BUFFER_SIZE);
128

```

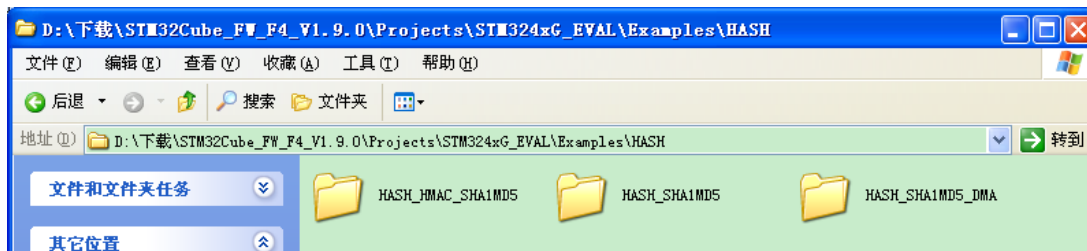
只需要两个步骤。

..\CRYPT



这 4 个例程演示了 DES、TDES 和 AES 三种加密/解密算法的使用方法。

..\HASH



这 3 个例程演示了几种 HASH 算法的使用方法。

..\IWDG\IWDG_Example

独立看门狗的使用例程。

..\WWDG\WWDG_Example

窗口看门狗的使用例程。

..\RNG\RNG_MultiRNG

演示使用硬件模块生成随机数。

..\SMARTCARD\SMARTCARD_T0

演示了智能卡的应用，使用的是 T0 通讯协议。所用外设是 UART。因为程序包含了 T0 通讯协议处理，所以看上去比较复杂。

STM32Cube 提供了 smartcard 的底层驱动，即 stm32f4xx_hal_smartcard.c 和.h 文件，这属于 HAL 库。本例中 ST 还提供了 smartcard 的 T0 协议支持文件，即 smartcard.c 和.h 文件，这两个文件不属于 HAL 库。



S.D.Lu 于 深圳
2016 年 9 月