TIMER
Cerchiamo di far lampeggiare i led programmando esplicitamente un timer.

I timer e il clock sono un argomento complesso. Secondo UM1974, sez. 6.8, di default l'high speed clock esterno (HSE) ad 8 MHz proviene dall'output MCO (microcontroller clock output) di ST/LINK (il debugger). Secondo il clock tree di default prodotto da CubeMX è questo che viene propagato, ed arriva a SYSCLK (96 MHz) -> HCLK (96 MHz, dovrebbe essere il clock della CPU), APB1 peripherals (48 MHz), APB1 timer (96 MHz), APB2 periperhal (96 MHz), APB2 timer (96 MHz).

- Advanced: TIM1, TIM8 (16 bit)
- General-purpose: TIM2, TIM3, TIM4, TIM5 (16/32 bit), TIM9, TIM10, TIM11, TIM12, TIM13, TIM14 (16 bit)
- Basic: TIM6, TIM7 (16 bit)

A quali bus sono attaccati i timer? Vedendo stm32f767xx.h:

```
#define TIM6                ((TIM_TypeDef *) TIM6_BASE)
```

Andando più su nel file:
```
/*!< APB1 peripherals */
#define TIM2_BASE             (APB1PERIPH_BASE + 0x0000UL)
#define TIM3_BASE             (APB1PERIPH_BASE + 0x0400UL)
#define TIM4_BASE             (APB1PERIPH_BASE + 0x0800UL)
#define TIM5_BASE             (APB1PERIPH_BASE + 0x0C00UL)
#define TIM6_BASE             (APB1PERIPH_BASE + 0x1000UL)
#define TIM7_BASE             (APB1PERIPH_BASE + 0x1400UL)
#define TIM12_BASE            (APB1PERIPH_BASE + 0x1800UL)
#define TIM13_BASE            (APB1PERIPH_BASE + 0x1C00UL)
#define TIM14_BASE            (APB1PERIPH_BASE + 0x2000UL)
#define LPTIM1_BASE           (APB1PERIPH_BASE + 0x2400UL)
#define RTC_BASE              (APB1PERIPH_BASE + 0x2800UL)
...

/*!< APB2 peripherals */
#define TIM1_BASE             (APB2PERIPH_BASE + 0x0000UL)
#define TIM8_BASE             (APB2PERIPH_BASE + 0x0400UL)
...
#define TIM9_BASE             (APB2PERIPH_BASE + 0x4000UL)
#define TIM10_BASE            (APB2PERIPH_BASE + 0x4400UL)
#define TIM11_BASE            (APB2PERIPH_BASE + 0x4800UL)
...
```

molto chiaro!

Il seguente codice è tratto da Carmine Noviello; se vogliamo lampeggiare con la solita frequenza di 0.5 Hz usiamo la formula (pag. 320 eq. 1):

$$UpdateFreq = \frac{TimerFreq}{(Prescaler + 1)(Period + 1)} \quad ;$$

se vogliamo un UpdateEvent ogni secondo, ossia a frequenza di 1 Hz, abbiamo 1 Hz = $96 \cdot 10^6$ Hz /((Prescaler + 1) (Period + 1)). Tenendo conto che il prescaler è a 16 bit, e quindi non può avere valori più alti di 65536, prendiamo ad esempio Prescaler + 1 = $32 \cdot 10^3$ e Period + 1 = $3 \cdot 10^3$; otteniamo $96 \cdot 10^6$ / ($32 \cdot 10^3 \cdot 3 \cdot 10^3$) = $96 \cdot 10^6$ / ($96 \cdot 10^6$) = 1. Quindi: Prescaler = 31999 e Period = 2999. Sotto CubeMX nel device configuration tool basta selezionare Timers > TIM6 e

configurare il timer attivandolo ed impostando il valore di prescaler e period. Altre configurazioni utili:

- Autoreload preload: determina se, una volta che modifichiamo il valore del registro autoreload (modifica che avviene su un registro shadow), questo viene immediatamente caricato nel registro autoreload oppure caricato solo al successivo overflow.
- One pulse mode: il timer in one pulse mode non è periodico, ma si disattiva dopo il primo overflow.
- Counter mode: può essere up (da zero al valore di period) o down (da period a zero); nel caso dei timer basic può solo essere up.

Per il momento gestiamo il timer in polling. CubeMX genera questo codice:

stm32f7xx_hal_conf.h (incluso in stm32f7xx_hal.h, a sua volta incluso in main.h):

```
/* #define HAL_SPI_MODULE_ENABLED   */
#define HAL_TIM_MODULE_ENABLED
#define HAL_UART_MODULE_ENABLED
...

#ifdef HAL_SPI_MODULE_ENABLED
 #include "stm32f7xx_hal_spi.h"
#endif /* HAL_SPI_MODULE_ENABLED */

#ifdef HAL_TIM_MODULE_ENABLED
 #include "stm32f7xx_hal_tim.h"
#endif /* HAL_TIM_MODULE_ENABLED */

#ifdef HAL_UART_MODULE_ENABLED
 #include "stm32f7xx_hal_uart.h"
#endif /* HAL_UART_MODULE_ENABLED */

...
```

stm32f7xx_hal_msp.c:

```
void HAL_TIM_Base_MspInit(TIM_HandleTypeDef* htim_base)
{
  if(htim_base->Instance==TIM6)
  {
  /* USER CODE BEGIN TIM6_MspInit 0 */

  /* USER CODE END TIM6_MspInit 0 */
    /* Peripheral clock enable */
    __HAL_RCC_TIM6_CLK_ENABLE();
  /* USER CODE BEGIN TIM6_MspInit 1 */

  /* USER CODE END TIM6_MspInit 1 */
  }

}

void HAL_TIM_Base_MspDeInit(TIM_HandleTypeDef* htim_base)
{
  if(htim_base->Instance==TIM6)
  {
  /* USER CODE BEGIN TIM6_MspDeInit 0 */

  /* USER CODE END TIM6_MspDeInit 0 */
    /* Peripheral clock disable */
    __HAL_RCC_TIM6_CLK_DISABLE();
  /* USER CODE BEGIN TIM6_MspDeInit 1 */

  /* USER CODE END TIM6_MspDeInit 1 */
  }
```

```
}
```

main.c:

```
...
/* Private variables ---------------------------------------------------------*/
...
TIM_HandleTypeDef htim6;
...

int main(void)
{
  ...
  MX_TIM6_Init();
  ...
}

static void MX_TIM6_Init(void)
{

  /* USER CODE BEGIN TIM6_Init 0 */

  /* USER CODE END TIM6_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM6_Init 1 */

  /* USER CODE END TIM6_Init 1 */
  htim6.Instance = TIM6;
  htim6.Init.Prescaler = 31999;
  htim6.Init.CounterMode = TIM_COUNTERMODE_UP; /* modalità contatore, da zero a
2999 */
  htim6.Init.Period = 2999;
  htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim6) != HAL_OK) /* notare che HAL_TIM_Base_Init
invoca HAL_TIM_Base_MspInit */
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM6_Init 2 */

  /* USER CODE END TIM6_Init 2 */

}
```

A questo punto dobbiamo scrivere il codice custom nel main. Lo stato di overflow è determinato dal bit 0 del registro TIM6_SR (sez. 28.4.4 RM0410), pertanto:

main.c:

```
...
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define UIF 0x0001
/* USER CODE END PD */

...
```

```
int main(void)
{
  ...
  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start(&htim6); /* questo dobbiamo metterlo noi per avviare il
timer, CubeMX non lo fa! */
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
    if (htim6.Instance->SR & UIF) {
      /* overflow */
      HAL_GPIO_TogglePin(GPIOB, LD1_Pin | LD2_Pin | LD3_Pin);
      htim6.Instance->SR &= ~UIF;
    }
  }
  /* USER CODE END 3 */
}
```

INTERRUPT
Ora modifichiamo il blinky basato sul timer per gestire gli overflow con interrupt: occorre solo discutere

- come attivare gli interrupt all'overflow del timer;
- come agganciare del codice all'interrupt del timer.

Per attivare gli interrupt da CubeMX basta selezionare un opportuno flag nel device configuration tool per Timers > TIM6 (NVIC settings, enabled). Il codice generato da CubeMX cambia così:

stm32f7xx_hal_msp.c:

```
void HAL_TIM_Base_MspInit(TIM_HandleTypeDef* htim_base)
{
  if(htim_base->Instance==TIM6)
  {
  /* USER CODE BEGIN TIM6_MspInit 0 */

  /* USER CODE END TIM6_MspInit 0 */
    /* Peripheral clock enable */
    __HAL_RCC_TIM6_CLK_ENABLE();
    /* TIM6 interrupt Init */
    HAL_NVIC_SetPriority(TIM6_DAC_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(TIM6_DAC_IRQn);
  /* USER CODE BEGIN TIM6_MspInit 1 */

  /* USER CODE END TIM6_MspInit 1 */
  }

}

void HAL_TIM_Base_MspDeInit(TIM_HandleTypeDef* htim_base)
{
  if(htim_base->Instance==TIM6)
  {
  /* USER CODE BEGIN TIM6_MspDeInit 0 */

  /* USER CODE END TIM6_MspDeInit 0 */
```

```
    /* Peripheral clock disable */
    __HAL_RCC_TIM6_CLK_DISABLE();

    /* TIM6 interrupt DeInit */
    HAL_NVIC_DisableIRQ(TIM6_DAC_IRQn);
  /* USER CODE BEGIN TIM6_MspDeInit 1 */

  /* USER CODE END TIM6_MspDeInit 1 */
  }

}
```

main.c:

```
...
/* Private variables ---------------------------------------------------------*/
...
TIM_HandleTypeDef htim6;
...

int main(void)
{
  ...
  MX_TIM6_Init();

  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start_IT(&htim6); /* questo dobbiamo metterlo noi per avviare il
timer, il suffisso _IT lo avvia con interrupt */
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    /* notare che il corpo del ciclo while è vuoto! */
  }
  /* USER CODE END 3 */
}
```

Per agganciare il codice all'interrupt, il meccanismo per registrare le proprie funzioni è diverso sia da quello usato nell' 8051 (sintassi speciale int) sia da quello della registrazione di callback con puntatori a funzione. L'idea è che lo HAL mette già a disposizione delle funzioni agganciate agli interrupt che non fanno nulla. Tali funzioni sono annotate dall'annotazione __weak: questa annotazione fa si che se da qualche altra parte viene definita una funzione con lo stesso nome, la seconda definizione fa l'overriding della prima.

Su Carmine Noviello viene spiegato tutto nel dettaglio: occorre agganciare un gestore di interrupt fornito dall'HAL apposta per gestire l'interrupt dei timer, e che fa il dispatching su delle callback opportune a seconda dei tipi di eventi generati dai diversi tipi di timer. Questa si chiama HAL_TIM_IRQHandler ed è definita in stm32f7xx_hal_tim.h. Fatto questo occorre agganciare la callback per l'evento di elapsing del periodo del timer con il nostro codice custom, funzione che si chiama HAL_TIM_PeriodElapsedCallback (vedere 64.2.11 / 64.2.13 UM1905).

Codice di Carmine Noviello:

main.c:

```
void TIM6_DAC_IRQHandler(void) /* nome funzione aggiornato all'ultima versione
dell'HAL per la scheda di laboratorio */
{
  HAL_TIM_IRQHandler(&htim6);
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
  /* questo e' l'unico vero codice applicativo */
  if (htim->Instance == TIM6) {
    HAL_GPIO_TogglePin(GPIOB, LD1_Pin | LD2_Pin | LD3_Pin);
  }
}
```

Se invece generiamo il codice con CubeMX dobbiamo solo inserire il codice della callback:

main.c:

```
...
/* USER CODE BEGIN 4 */

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
  if (htim->Instance == TIM6) {
    HAL_GPIO_TogglePin(GPIOB, LD1_Pin | LD2_Pin | LD3_Pin);
  }
}


/* USER CODE END 4 */
...
```

perché l'aggancio del gestore di interrupt standard che fa il dispatching alle callback viene generato automaticamente da CubeMX come segue:

stm32f7xx_it.c: /* interrupt service routines */

```
...
/* External variables --------------------------------------------------------*/
extern TIM_HandleTypeDef htim6; /* dichiarato in main.c */
...
void TIM6_DAC_IRQHandler(void)
{
  /* USER CODE BEGIN TIM6_DAC_IRQn 0 */

  /* USER CODE END TIM6_DAC_IRQn 0 */
  HAL_TIM_IRQHandler(&htim6);
  /* USER CODE BEGIN TIM6_DAC_IRQn 1 */

  /* USER CODE END TIM6_DAC_IRQn 1 */
}
```

Quindi, in estrema sintesi, con CubeMX quello che dobbiamo fare è:

1. Configurare il timer tra i devices dell'IOC: attivarlo, impostare il valore di prescaler e period, attivare l'interrupt;
2. Generare il codice del progetto;
3. Aggiungere lo start del timer nel main;
4. Aggiungere la callback per l'evento di timer elapsed che fa il toggle dei led.

Molto semplice!!!