SCHEDA e MCU
La scheda è una Nucleo F767 ZI, con microcontrollore (MCU) STM32F767ZI basato su ARM Cortex M7, fino a 216 MHz di clock, package LQFP144: ciò significa che i pins del nostro MCU sono in tutto 144 (STM32F767xx datasheet, package LQFP144). Ci sono 107 pin mappati su corrispondenti GPIO, e che hanno infatti come nomi principali:

PA0-15
PB0-15
PC0-15
PD0-15
PE0-15
PF0-15
PG0-15
PH0-1

Notare le che porte GPIO nei microcontrollori ST sono indicate con le lettere da A a K, e sono anche chiamate PA..PK. Il nostro MCU arriva fino a PH.
Gli altri pin sono power (27), reset state (2), boot (1) - non tornano i conti!

PERIFERICHE
Dall'header file stm32f767xx.h (e dai diagrammi contenuti nei manuali) facciamo una tabellina delle periferiche rispetto ai bus sui quali sono connessi:

Periferiche su bus AHB1:
GPIOA..K, CRC, RCC, FLASH, DMA1..2, DMA2D (CHROM-ART), ETH

Periferiche su bus AHB2:
DCMI, JPEG, RNG

Periferiche su bus APB1:
PWR, IWDG, WWDG, LPTIM1, TIM2..7, TIM12..14, USART2..3, UART4..5, UART7..8, SPI2..3, I2C1..4, CAN1..3, SPDIFRX, HDMI-CEC, DAC

Periferiche su bus APB2:
EXTI, SDMMC1..2, TIM1, TIM8, TIM9..11, USART1, USART6, SPI1, SPI4..6, SAI1..2, DFSDM1, MDIOS, ADC1..3, SYSCFG, LTDC

GPIO OUTPUT: LED

Iniziamo con i led; facciamoli lampeggiare tutti e tre contemporaneamente con frequenza 0.5 Hz (ossia, i led devono stare accesi per 1 sec, e spenti per 1 sec). Dall'user manual delle schede STM32 Nucleo-144 UM1974 (en.DM00244518.pdf), abbiamo (sezione 6.5) che LD1 è connesso a PB0 o PA5 a seconda dei solder button SB120 e SB119, ma di default la scheda viene configurata con i solder button che connettono LD1 a PB0; il led LD2 è connesso a PB7; e il led LD3 a PB14. Questi sono dei pin del processore. Per attivarli il pin deve pilotare il GPIO.

Configurazione pin GPIO: sotto device configuration tool selezionare System Core > GPIO, e a quel punto PB0, PB7 e PB14. L'output GPIO può essere impostato:

- push-pull (sempre a bassa impedenza, impone l'output alto / basso) o open-drain (l'output può essere basso / alta impedenza, oppure alto / alta impedenza)
- pull-up, pull-down o nessuno dei due. Nell'open drain si può usare una resistenza di pull-up o pull-down interna, oppure usarne una esterna; dal momento che l'open drain si usa quando

si collegano tanti output GPIO in parallelo per fare un bus wired (N)OR, almeno uno deve avere una resistenza di pull-up o pull-down. Anche nel push-pull si può selezionare se attivare una resistenza di pull-up, pull-down o nessuna delle due; probabilmente servono per limitare la corrente sul pin di output nel caso il carico sia troppo "sensibile", ma non è chiaro.

I default vanno bene, ossia devono essere:

- GPIO mode: output push-pull (occorre pilotare il led sia in accensione che in spegnimento)
- GPIO pull-up / pull-down: no pull-up and no pull-down

Codice generato:

main.c:

```
static void MX_GPIO_Init(void)
{
...
HAL_GPIO_WritePin(GPIOB, LD3_Pin|LD2_Pin|LD1_Pin, GPIO_PIN_RESET);
```

dove:

stm32f767xx.h (tra i driver del CMSIS):

```
#define GPIOB               ((GPIO_TypeDef *) GPIOB_BASE)
#define GPIOB_BASE            (AHB1PERIPH_BASE + 0x0400UL)
#define AHB1PERIPH_BASE       (PERIPH_BASE + 0x00020000UL)
#define PERIPH_BASE            0x40000000UL /*!< Base address of : AHB/ABP
Peripherals                                                 */

typedef struct
{
  __IO uint32_t MODER;    /*!< GPIO port mode register,               Address
offset: 0x00      */
  __IO uint32_t OTYPER;   /*!< GPIO port output type register,        Address
offset: 0x04      */
  __IO uint32_t OSPEEDR;  /*!< GPIO port output speed register,       Address
offset: 0x08      */
  __IO uint32_t PUPDR;    /*!< GPIO port pull-up/pull-down register,  Address
offset: 0x0C      */
  __IO uint32_t IDR;      /*!< GPIO port input data register,         Address
offset: 0x10      */
  __IO uint32_t ODR;      /*!< GPIO port output data register,        Address
offset: 0x14      */
  __IO uint32_t BSRR;     /*!< GPIO port bit set/reset register,      Address
offset: 0x18      */
  __IO uint32_t LCKR;     /*!< GPIO port configuration lock register, Address
offset: 0x1C      */
  __IO uint32_t AFR[2];   /*!< GPIO alternate function registers,     Address
offset: 0x20-0x24 */
} GPIO_TypeDef;
```

stm32f7xx_hal_gpio.c (tra i driver dell'HAL):

```
typedef enum
{
  GPIO_PIN_RESET = 0,
  GPIO_PIN_SET
}GPIO_PinState;

#define GPIO_PIN_0                 ((uint16_t)0x0001U)  /* Pin 0 selected    */
...
#define GPIO_PIN_15                ((uint16_t)0x8000U)  /* Pin 15 selected   */
```

```
void HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState
PinState)
{
  /* Check the parameters */
  assert_param(IS_GPIO_PIN(GPIO_Pin));
  assert_param(IS_GPIO_PIN_ACTION(PinState));

  if(PinState != GPIO_PIN_RESET)
  {
    GPIOx->BSRR = GPIO_Pin;
  }
  else
  {
    GPIOx->BSRR = (uint32_t)GPIO_Pin << 16;
  }
}
```

main.h

```
#define LD1_Pin GPIO_PIN_0
#define LD2_Pin GPIO_PIN_7
#define LD3_Pin GPIO_PIN_14
```

Codice:

main.c:

```
int main(void)
{
...
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_0 | GPIO_PIN_7 | GPIO_PIN_14);
    HAL_Delay(1000);
  /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Alternativa: usare ciclo di attesa attiva; durata calcolata approssimativamente dal clock del processore, che secondo il clock tree è di 96 MHz. Dal momento che il Cortex M7 è dual-issued, ossia esegue due istruzioni per ciclo, otteniamo 192 MIPS. Il ciclo ha 3 istruzioni, quindi otteniamo 64,000,000 di ripetizioni del ciclo, in esadecimale 0x3D09000. Però le costanti immediate nel caso dei processori ARM sono 8 bit + rotazione di 4 bit (https://alisdair.mcdiarmid.org/arm-immediate-value-encoding/) per formare un valore di 32 bit. Il più simile è 0x3D00000.

main.c:

```
...
/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
#define PAUSE_CIRCA_ONE_SECOND \
    asm volatile("mov r0, #0x3D00000\n" \
                 "lbl%=:\n" \
                 "sub r0, r0, #1\n" \
                 "cmp r0, #0\n" \
                 "bne lbl%=" : : : "cc", "r0");
                 /* o anche: "bne .-6" : : : "cc", "r0"); */
/* USER CODE END PM */
...
int main(void)
{
...
  /* USER CODE BEGIN WHILE */
```

```
  while (1)
  {
        HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_0 | GPIO_PIN_7 | GPIO_PIN_14);
        PAUSE_CIRCA_ONE_SECOND;
   /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
...
```

Provare a cambiare:

```
  while (1)
  {
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0 | GPIO_PIN_7 | GPIO_PIN_14,
GPIO_PIN_SET);
        PAUSE_CIRCA_ONE_SECOND;
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0 | GPIO_PIN_7 | GPIO_PIN_14,
GPIO_PIN_RESET);
        PAUSE_CIRCA_ONE_SECOND;
  /* USER CODE BEGIN 3 */
  }
```

oppure:

```
  while (1)
  {
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0 | GPIO_PIN_7 | GPIO_PIN_14,
GPIO_PIN_SET);
        PAUSE_CIRCA_ONE_SECOND;
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0 | GPIO_PIN_7 | GPIO_PIN_14,
GPIO_PIN_RESET);
        PAUSE_CIRCA_ONE_SECOND;
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0 | GPIO_PIN_7 | GPIO_PIN_14,
GPIO_PIN_SET);
        PAUSE_CIRCA_ONE_SECOND;
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0 | GPIO_PIN_7 | GPIO_PIN_14,
GPIO_PIN_RESET);
        PAUSE_CIRCA_ONE_SECOND;
  /* USER CODE BEGIN 3 */
  }
```

oppure aggiungere:

```
/* USER CODE BEGIN Includes */
#include <core_cm7.h>
/* USER CODE END Includes */
...
  /* USER CODE BEGIN SysInit */
  SCB_DisableICache();
  /* USER CODE END SysInit */
```