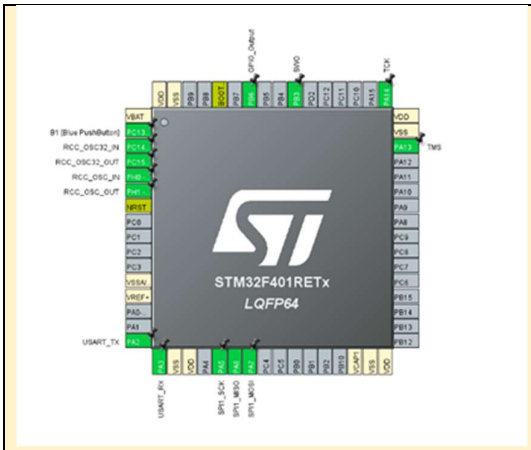
Mark	/11

Team name:	*A14*		
Homework number:	*HOMEWORK 09*		
Due date:	16/11/25		
Contribution	NO	Partial	Full
Mattia Di Mauro			*x*
Francesca Biondi			*x*
Lorenzo Castelli			*x*
Carmen Maria Niro			*x*
Pietro Albrigi			*x*
Notes: none			

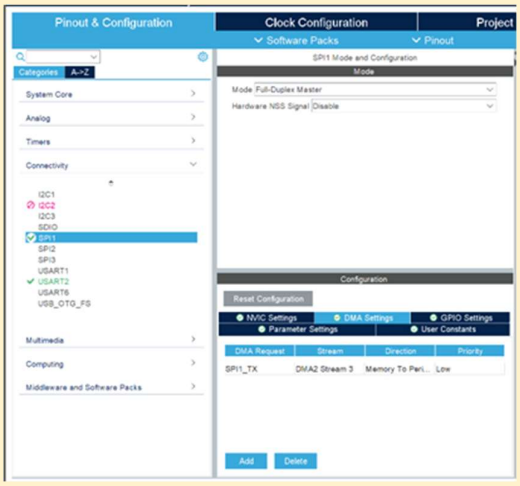
Project name	HOMEWORK 9		
Not done	Partially done (major problems)	Partially done (minor problems)	Completed
			x

The goal of this project was to transmit a letter to an LED matrix using SPI and a timer interrupt, alternating between two letters over time. The implementation required the configuration of GPIO pins, SPI communication, DMA transfers and two timers operating at different frequencies to handle the refresh of the display and the switching between characters.

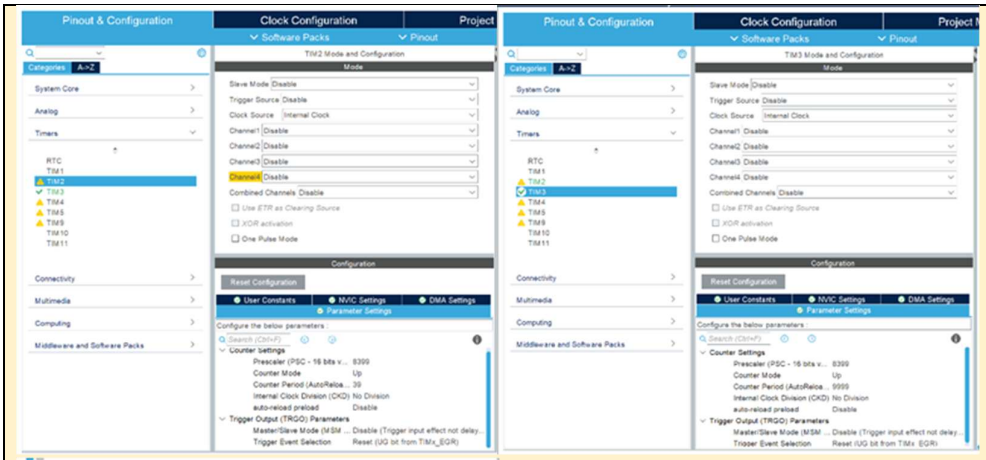
In the CubeMX configuration, the pin PB6 was set as a general-purpose output. This pin corresponds to the RCLK signal, which acts as a latch for the LED matrix driver: after each SPI transmission, a short pulse on this pin updates the outputs of the matrix, making the new data visible. The pins PA5, PA6 and PA7 were instead configured as SPI1 interface lines, specifically for the SCK, MISO and MOSI functions. This configuration allows the STM32 to send serial data to the LED matrix driver.



The SPI1 peripheral was set in master mode with a prescaler value of four. The SPI transmission was enabled in DMA mode, allowing data to be sent automatically without involving the CPU.



Two timers were used to coordinate the operation of the system. Timer 2 was configured to generate an interrupt every 4 milliseconds, determining the refresh rate of the matrix, while Timer 3 generated an interrupt every 1 second, used to alternate between two different characters. The priority of Timer 3 was set to 1 to ensure accurate and regular display updates.



In the source code, the definitions #define RCLK_GPIO_Port GPIOB and #define RCLK_Pin GPIO_PIN_6 identify the port and pin used for the latch pulse.

```
/* USER CODE BEGIN PD */

#define RCLK_GPIO_Port GPIOB
#define RCLK_Pin       GPIO_PIN_6

/* USER CODE END PD */
```

The characters to be displayed were defined as five-byte arrays, each byte representing one of the five columns of a 5×7 LED character matrix. Specifically, two arrays named FONT_A and FONT_B were created to store the binary patterns for the letters “A” and “B”. Each bit in these arrays corresponds to a single LED in a column, with the value 1 turning the LED on and 0 turning it off. Another array, called COL, specifies which column of the matrix is currently active. By rapidly cycling through these columns, the human eye perceives the full character thanks to persistence of vision, even though only one column is illuminated at a time.

```c
/* USER CODE BEGIN PV */

uint8_t FONT_A[5] = {
  0b1111100,
  0b0010010,
  0b0010001,
  0b0010010,
  0b1111100
};

uint8_t FONT_B[5] = {
  0b1111111,
  0b1001001,
  0b1001001,
  0b1001001,
  0b0110110
};

uint8_t COL[5] = {
  0b00000001,
  0b00000010,
  0b00000100,
  0b00001000,
  0b00010000
};

uint8_t tx2[2];
uint8_t col_idx=0;
int font=0;

/* USER CODE END PV */
```

A dedicated function named RCLK_Pulse() was implemented to manage the latching mechanism. This function generates a short pulse on the RCLK pin by toggling it high and then low. Every time a new pair of bytes is transmitted over SPI, this pulse transfers the data from the shift registers to the output registers of the LED driver, updating the LEDs that are currently active.

```c
void RCLK_Pulse(void)
{
  HAL_GPIO_WritePin(RCLK_GPIO_Port, RCLK_Pin, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(RCLK_GPIO_Port, RCLK_Pin, GPIO_PIN_SET);
  HAL_GPIO_WritePin(RCLK_GPIO_Port, RCLK_Pin, GPIO_PIN_RESET);
}
```

The SPI communication was handled using DMA, and the completion of each transmission triggered the callback function HAL_SPI_TxCpltCallback(). Within this callback, the RCLK pulse function was called to latch the transmitted data, and the column index variable was incremented modulo five to move to the next column. In this way, each SPI transmission corresponds to the update of one column of the currently displayed letter.

```c
void HAL_SPI_TxCpltCallback(SPI_HandleTypeDef *hspi)
{
  if (hspi==&hspi1)
  {
    RCLK_Pulse();
    col_idx=(col_idx+1)%5;
  }
}
```

The overall behavior of the system was controlled through the timer interrupt callback HAL_TIM_PeriodElapsedCallback(). When the interrupt was generated by Timer 2, the program prepared the data for the next transmission. Depending on the value of a variable named font, the system selected either the letter “A” or the letter “B”. The corresponding column data was copied from the chosen font array into a two-byte buffer, with the first byte representing the LED pattern and the second byte selecting the active column. This buffer was then transmitted through SPI using the DMA function HAL_SPI_Transmit_DMA(&hspi1, tx2, 2). The DMA mechanism automatically handled the transfer, and upon completion, the SPI callback was invoked to latch the data and proceed to the next column.

```
/* USER CODE BEGIN 4 */

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if (htim == &htim2)
  {
      if(font)
      {
          tx2[0] = FONT_B[col_idx];
          tx2[1] = COL[col_idx];
      }
      else
      {
          tx2[0] = FONT_A[col_idx];
          tx2[1] = COL[col_idx];
      }



    HAL_SPI_Transmit_DMA(&hspi1, tx2, 2);
  }

  if(htim==&htim3)
    {
        if(font)
        {
            font=0;
        }
        else
        {
            font=1;
        }
    }
}

/* USER CODE END 4 */
```

When the interrupt was generated by Timer 3, the font variable was toggled between 0 and 1, effectively alternating the displayed character every second. This mechanism produced the visual effect of switching between two letters, such as “A” and “B”, on the LED matrix.

The code was then tested on the board and confirmed to work as expected.

Professor comments: