Mark	A

Team name:	*A1*		
Homework number:	*HOMEWORK 7*		
Due date:	19/11/2023		
Contribution	NO	Partial	Full
Monti Pietro			*x*
Moretto Alessia			*x*
Pallotto Francesco			*x*
Perna Alessandro			*x*
Ventura Ludovico			*x*
Notes:			

Project name	Write a single letter and Alternate between letters		
Not done	Partially done (major problems)	Partially done (minor problems)	Completed
			x

Explanation:
We successfully completed the homework.

Part 7a:

At first we set timer 2 at 250Hz, with the prescaler to 8400-1 and the counter period to 40-1.

We configured PA5 pin to SPI1_SCK and PA7 pin to SPI1_MOSI to be able to exploit the SPI interface. Regarding SPI1_MISO and CS ports, it was not necessary to touch them, since the former is not used in this project, whereas the latter is soldered to ground on our board, so it's always active. The SPI1 is set with a prescaler of 4 and it operates in Transmit Only Master mode.
Then, we adjusted the PB6 pin as GPIO_Output in order to control the RCLK.

Before starting timer 2 in interrupt mode, we store all the letter picture elements in a 16 bits buffer to be sent by SPI1. Column values are placed in MSB position, whereas row ones in LSB.

```
for (int i = 0; i < 5; i++) {
    buffer[i] = (uint16_t)((letter[i].column << 8) | (letter[i].row & 0xFF));
}
```

We used the following callback to turn on the LEDs in the selected positions in one column of the LED matrix each time the timer is triggered. To do this, we transmit the data stored in buffer[position] with the SPI1 to the board LED matrix.

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    if (htim==&htim2){
        HAL_SPI_Transmit(&hspi1,(uint8_t*)&buffer[position], 2, timeout_ms);
        HAL_GPIO_WritePin(GPIOB,GPIO_PIN_6, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB,GPIO_PIN_6, GPIO_PIN_RESET);
        if(position < 5){
            position++;
        } else position = 0;
    }
}
```

After that we set the GPIOB_PIN_6, which represents the RCLK, to 1 and then we reset it: these two operations are necessary to pass the focus from the first shift register in the LED matrix, which contains

data which are updating, so they are not ready to be transmitted, to the second one, whose content must be constant in order to display correct and coherent data on the LED matrix.
We used a counter variable to select the column. When we reach the last one, we reset the counter.

Part 7b:

We initiated the board configuration following the same setup as the previous project (7a).

Subsequently, we adjusted the SPI1 configuration, enabling DMA with memory-to-peripheral direction (SPI1_TX). Additionally, we activated a second Timer (TIM3) as an internal clock, setting the Prescaler to 8400-1 and the Auto-Reload Register to 10000-1, resulting in a clock frequency of 1Hz. Finally, we enabled TIM3 and SPI1 global interrupts from the NVIC table.

In the C code, within the main function, we defined the pixels of the letters A and W as a matrix, utilizing a struct variable described earlier.

```
/* USER CODE BEGIN 1 */
matrix letter_A[5] = {{16,31},{8,36},{4,68},{2,36},{1,31}};
matrix letter_W[5] = {{16,126},{8,1},{4,14},{2,1},{1,126}};
uint8_t temp_letter_switch = letter_switch;
/* USER CODE END 1 */
```

Prior to the while(1) loop, we started TIM2 and TIM3 in interrupt mode.

The TIM2 callback now manages the DMA transmission of one LED column (variable position) of the letter from the board memory through the SPI1. Simultaneously, TIM3 oversees a variable used to determine which of the two letters will be sent to the LED matrix.
Every second, TIM3 changes the letter displayed by the LED matrix.

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    if (htim==&htim2){              //TIM2 sends the data to spi
        HAL_SPI_Transmit_DMA(&hspi1,(uint8_t*)&buffer[position], 2);
    } else if(htim==&htim3){        //TIM3 sets the switch flag of the letters(operation is executed in main)
        letter_switch = !letter_switch;
    }
}
```

The HAL_SPI_TxCpltCallback sets and resets the RLCK pin, cycling through the columns of the letter. The fundamental principle of this code is that the SPI sends information one column at a time. Once sent, we prepare the SPI for the next transmission and switch to the next column.

```
void HAL_SPI_TxCpltCallback(SPI_HandleTypeDef *hspi){
    // Set & Reset the RCLK Pin
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_6, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_6, GPIO_PIN_RESET);
    // Update position index
    if(++position >= 5)
        position = 0;
}
```

In the main while(1) function, we manage the alternation of the two letters, when TIM3 ticks, by checking the variable letter_switch. Temporarily halting the SPI1 transmission, we copy the new desired

letter into the transmission buffer and then restart the SPI operation.

```
while (1)
{
    // temp_letter_switch is a flag by triggering TIM3
    if (temp_letter_switch != letter_switch){
        // Pause the SPI transmission
        HAL_SPI_DMAPause(&hspi1);

        // Change the letter data in the buffer
        buffer_cpy(buffer, (letter_switch == 1) ? letter_A : letter_W);
        temp_letter_switch = letter_switch;
        position = 0;

        // Resume the SPI transmission
        HAL_SPI_DMAResume(&hspi1);
    }
    /* USER CODE END WHILE */
```

The “buffer_cpy” function is defined by us as follows, copying all columns of one matrix into another.

```
void buffer_cpy(uint16_t *buffer, matrix *to_buffer) {
    // Place all elements in a buffer to be sent by SPI
    for (int i = 0; i < 5; i++) {
            buffer[i] = (uint16_t)((to_buffer[i].column << 8) | (to_buffer[i].row & 0xFF));
    }
}
```

Professor comments:

Very good. The code is very well written.