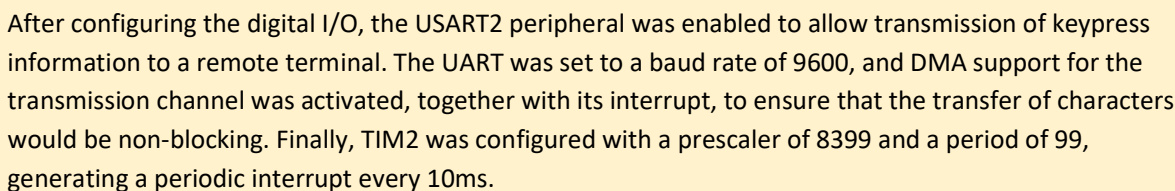


| | |
|------|-----|
| Mark | /11 |
|------|-----|

| | | | |
|-------------------|-------------|---------|------|
| Team name: | A14 | | |
| Homework number: | HOMEWORK 10 | | |
| Due date: | 23/11/25 | | |
| | | | |
| 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 10 | | |
| Not done | Partially done (major problems) | Partially done (minor problems) | Completed |
| | | | x |
| Project 1: Keyboard readout To interface a 4x4 matrix keypad with the STM32 board, the GPIO pins corresponding to the keypad rows and columns were first configured in the .ioc file. The four input lines, connected to PC2, PC3, PC12 and PC13, were set as GPIO inputs, while PC8, PC9, PC10 and PC11 were selected as GPIO outputs to drive the keypad columns. | | | |



Inside the code, a simple structure was created to store, for every key in the 4x4 matrix, its current and previous logic levels. This allows the detection of a new press only when a transition from high to low is observed, avoiding repeated transmissions while the key remains held down. The keypad character arrangement was stored in a bidimensional array representing all 16 keys, and a column index was maintained to keep track of which column was currently being driven. A small character buffer was used to format the symbol that would later be transmitted by UART through DMA.

```

/* Private typedef -----*/
/* USER CODE BEGIN PTD */
typedef struct {
    GPIO_PinState current;
    GPIO_PinState previous;
} State;
/* USER CODE END PTD */

/* Private define -----*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -----*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables -----*/
TIM_HandleTypeDef htim2;

UART_HandleTypeDef huart2;
DMA_HandleTypeDef hdma_usart2_tx;

/* USER CODE BEGIN PV */
uint8_t col = 0;

int flag_columnactivated=0;
int transmission_completed=1;

char keyboard[4][4] = {{'D', 'C', 'B', 'A'},
    {'#', '9', '6', '3'},
    {'0', '8', '5', '2'},
    {'*', '7', '4', '1'}
};

char caract[64];

State keyState[4][4] = {GPIO_PIN_SET};

/* USER CODE END PV */

```

At the beginning of the code the first column was activated and the timer was started so that each timer interrupt would execute the scanning routine.

```

/* USER CODE BEGIN 2 */
activateColumn(0);
HAL_TIM_Base_Start_IT(&htim2);
/* USER CODE END 2 */

```

```

/* USER CODE BEGIN 4 */
void activateColumn(uint8_t col) {
    GPIO_PinState activation_columnpin_matrix[4][4] = {{GPIO_PIN_SET, GPIO_PIN_RESET, GPIO_PIN_RESET, GPIO_PIN_RESET},
        {GPIO_PIN_RESET, GPIO_PIN_SET, GPIO_PIN_RESET, GPIO_PIN_RESET},
        {GPIO_PIN_RESET, GPIO_PIN_RESET, GPIO_PIN_SET, GPIO_PIN_RESET},
        {GPIO_PIN_RESET, GPIO_PIN_RESET, GPIO_PIN_RESET, GPIO_PIN_SET}
    };

    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_11, activation_columnpin_matrix[col][0]);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_10, activation_columnpin_matrix[col][1]);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_9, activation_columnpin_matrix[col][2]);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, activation_columnpin_matrix[col][3]);
    flag_columnactivated=1;
}

```

During each timer callback, the software reads the state of the four row pins associated with the active column. If one of the rows is found in the low state while its stored state indicates that the key was previously high, the system identifies this as a new keypress. The corresponding character from the keypad matrix is placed into the transmission buffer, and a DMA transfer is started through HAL_UART_Transmit_DMA. A flag is used to ensure that a new character is sent only when the previous

transmission has completed, and this flag is set again inside the UART transmit completion callback. After checking all four rows and updating the stored states, the column index is incremented and the next column is activated so that the following timer interrupt will scan it in turn.

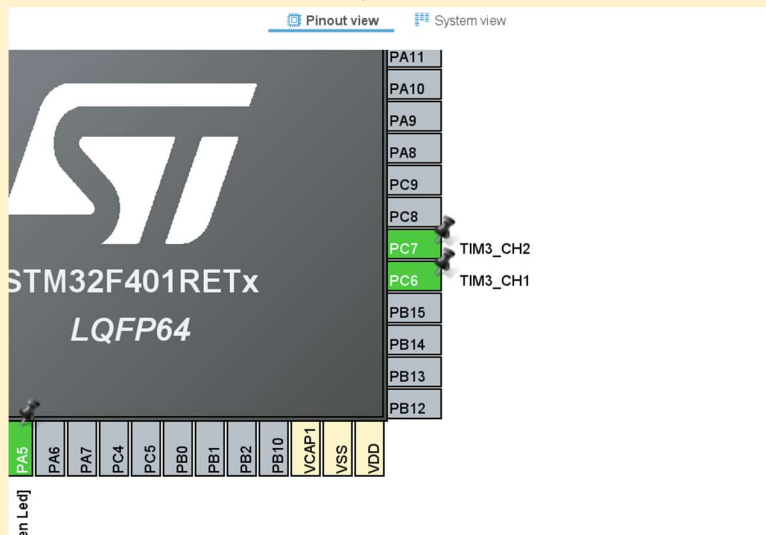
```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if (flag_columnactivated && transmission_completed){
        GPIO_PinState r[4]; // read the rows' values
        r[0] = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_3);
        r[1] = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_2);
        r[2] = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13);
        r[3] = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_12);
        for (int i = 0; i < 4; i++) {
            if (r[i] == GPIO_PIN_RESET && keyState[col][i].current == GPIO_PIN_RESET && keyState[col][i].previous == GPIO_PIN_SET) {
                int length = snprintf(charact, sizeof(charact), "%c\n", keyboard[col][i]);
                transmission_completed=0;
                HAL_UART_Transmit_DMA(&huart2, charact, length);
            }
            keyState[col][i].previous = keyState[col][i].current;
            keyState[col][i].current = r[i];
        }
        col = (col + 1) % 4;
        activateColumn(col);
    }
}

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart) {
    transmission_completed = 1;
}
//-----END OF FILE-----
```

The project worked as intended, providing a reliable and responsive interface between the keypad and the remote terminal.

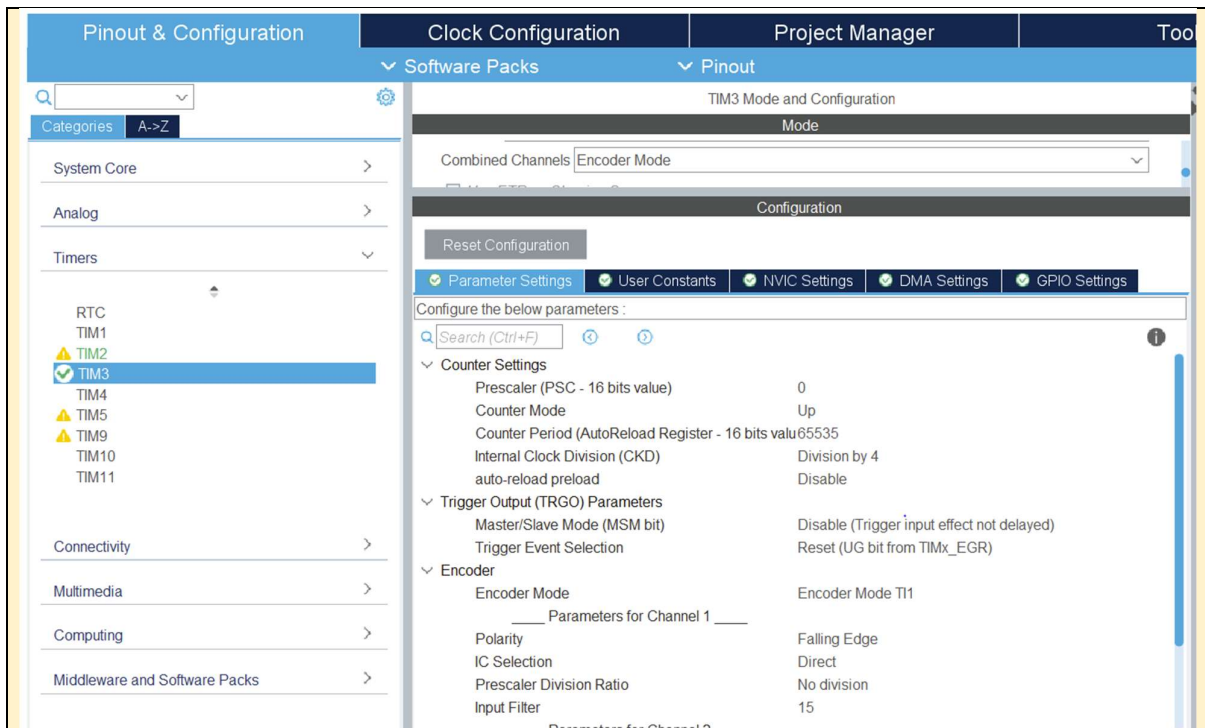
Project 2: Encoder readout

We first identified the encoder pins (PA6 and PA7) and enabled them in TIM3_CH3 mode:

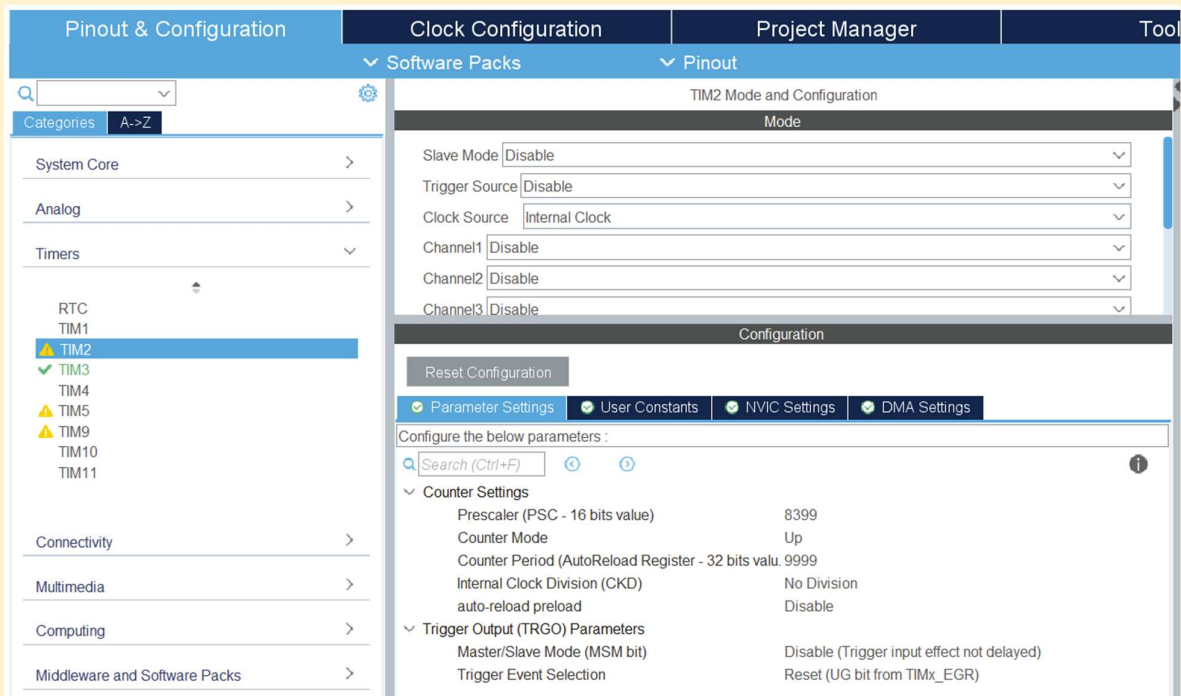


Next, we configured two timers.

TIM3 was set to operate in encoder mode, with the input filter set to 15 (corresponding to the maximum digital filtering level, which requires eight consecutive stable samples to validate a signal transition) and Encoder Mode TI1:



TIM2 was configured to generate an interrupt every second, and its global interrupt was enabled to allow the corresponding callback function to execute:



We also enabled a DMA request and enabled the required interrupts to allow transmission of the encoder data to the terminal.

In the main function we started both timers:

```
133 /* USER CODE BEGIN 2 */
134 HAL_TIM_Encoder_Start(&htim3, TIM_CHANNEL_ALL);
135 HAL_TIM_Base_Start_IT(&htim2);
136
137 /* USER CODE END 2 */
```

In the TIM2 callback function, executed once per second, we read the value of the TIM3 counter into the variable `current_count`, previously defined as a 16-bit unsigned integer. We then computed the delta by subtracting the previously stored counter value (also a 16-bit unsigned global variable). This difference was cast to a signed 16-bit integer to correctly interpret overflow and underflow events.

The resulting delta was then converted to revolutions per minute by dividing by 24, which corresponds to the number of counts per full revolution in TI1 encoder mode. Finally, the delta, the computed RPM value, and the rotation direction were transmitted to the terminal using UART with DMA.

```
69 /* USER CODE BEGIN 0 */
70 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
71 {
72     if(htim == &htim2)
73     {
74         current_count = __HAL_TIM_GET_COUNTER(&htim3);
75         delta = (int16_t)(current_count - old_count);
76         old_count = current_count;
77
78         float rpms = delta * (60.0f / 24.0f);
79
80         char direction;
81         if(delta > 0)
82         {
83             direction = '+'; //clockwise
84         } else if(delta < 0){
85             direction = '-'; //anti-clockwise
86         } else {
87             direction = '0'; //no movement
88         }
89
90         int length = snprintf(string, sizeof(string), "Delta: %d, %.1f rpm, Direction: %c\r\n", (int)delta, rpms, direction);
91         HAL_UART_Transmit_DMA(&huart2, string, length);
92     }
93 }
94 /* USER CODE END 0 */
```

The code was then tested on the board and confirmed to work as intended:

```
Delta: 0, 0.0 rpm, Direction: 0
Delta: 0, 0.0 rpm, Direction: 0
Delta: 0, 0.0 rpm, Direction: 0
Delta: 0, 0.0 rpm, Direction: 0
Delta: 0, 0.0 rpm, Direction: 0
Delta: -1, -2.5 rpm, Direction: -
Delta: 0, 0.0 rpm, Direction: 0
Delta: 0, 0.0 rpm, Direction: 0
Delta: 0, 0.0 rpm, Direction: 0
Delta: -2, -5.0 rpm, Direction: -
Delta: -8, -20.0 rpm, Direction: -
Delta: 0, 0.0 rpm, Direction: 0
Delta: 0, 0.0 rpm, Direction: 0
Delta: 3, 7.5 rpm, Direction: +
Delta: 5, 12.5 rpm, Direction: +
Delta: 2, 5.0 rpm, Direction: +
Delta: 4, 10.0 rpm, Direction: +
Delta: 0, 0.0 rpm, Direction: 0
Delta: 0, 0.0 rpm, Direction: 0
```

We also tested the implementation using controlled counter values to verify its correct behaviour in both overflow and underflow conditions:

```
old_count = 65530;  
current_count = 4;  
delta = (int16_t)(current_count - old_count);
```

```
Delta: 10, 25.0 rpm, Direction: +
```

```
old_count = 5;  
current_count = 65534;  
delta = (int16_t)(current_count - old_count);
```

```
Delta: -7, -17.5 rpm, Direction: -
```

This approach fails if the counter change between samples exceeds half the timer range (32,768 counts), because when the unsigned difference is cast into a signed 16-bit value any unsigned delta > 32,767 becomes a negative two's-complement number, producing an incorrect direction and value.

```
old_count = 5;  
current_count = 40005;  
delta = (int16_t)(current_count - old_count);
```

```
Delta: -25536, -63840.0 rpm, Direction: -
```

This issue is not particularly relevant in our application, as a counter change of that magnitude is physically unrealistic for the operating speeds of the encoder.

Professor comments: