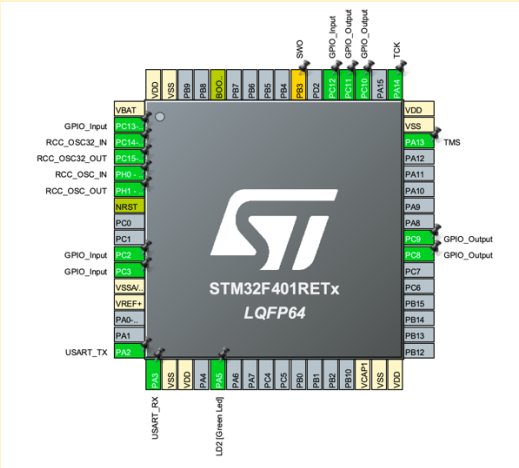
Mark	/11

Team name:	*A1*		
Homework number:	*HOMEWORK 10*		
Due date:	03/12/24		
Contribution	NO	Partial	Full
Piombo			*x*
Fumagalli			*x*
Pierfederici			*x*
Zenoni			*x*
Ferraro			*x*
Notes:			

Project name	Keyboard + Encoder		
Not done	Partially done (major problems)	Partially done (minor problems)	Completed
			x

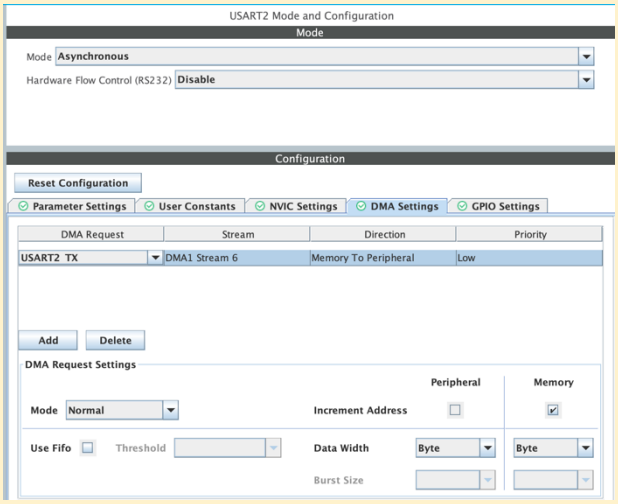
Part 1:

In the ".ioc" we set the I/O pins to use the keyboard and we checked the proper configuration of the UART pins.



We also set TIM2 to rise an interrupt every 5ms to change the keyboard's column we are reading from.

Additionally, we set the USART2 in DMA mode to send the characters to the PC, enabling also its interrupt.



Looking at the schematic we defined the keyboard pins:

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

#define ROW_0 GPIOC,GPIO_PIN_3
#define ROW_1 GPIOC,GPIO_PIN_2
#define ROW_2 GPIOC,GPIO_PIN_13
#define ROW_3 GPIOC,GPIO_PIN_12

#define COL_0 GPIOC,GPIO_PIN_11
#define COL_1 GPIOC,GPIO_PIN_10
#define COL_2 GPIOC,GPIO_PIN_9
#define COL_3 GPIOC,GPIO_PIN_8

#define NUM_OF_SAMPLES 2
#define ROW_LENGTH 4
#define COLUMN_LENGTH 4

/* USER CODE END PD */
```

We choose to sample the buttons only twice because it’s enough to correctly debounce.

These are our variables:

```
/* USER CODE BEGIN PV */

uint8_t sample_index = 0;   //index of the samples collected in order to debounce
uint8_t col_index = 0;      //index of the number of the column
uint8_t row_index = 0;      //index of the number of the row

uint8_t btn = 0;    //button state (used also for debouncing) - if btn = 0 -> button is pressed

uint8_t i = 0;
uint8_t j = 0;
uint8_t k = 0;

uint8_t rx_data[NUM_OF_SAMPLES][COLUMN_LENGTH][ROW_LENGTH];     //buttons state - 3rd dimension stores the different samples to debounce

char out_matrix[COLUMN_LENGTH][ROW_LENGTH] = {
        {'F', 'E', 'D', 'C'},
        {'B', 'A', '9', '8'},
        {'7', '6', '5', '4'},
        {'3', '2', '1', '0'}
};

uint8_t old_btn[COLUMN_LENGTH][ROW_LENGTH] = {  //previous state of the buttons
        {1, 1, 1, 1},
        {1, 1, 1, 1},
        {1, 1, 1, 1},
        {1, 1, 1, 1},
};

/* USER CODE END PV */
```

In the main function we initialized “rx_data” matrix with all ones (buttons not pressed).
Then we set the first column and we started TIM2 in interrupt mode.

```
  /* USER CODE BEGIN 2 */

  for (i = 0; i < NUM_OF_SAMPLES; i++) {
      for (j = 0; j < COLUMN_LENGTH; j++) {
          for (k = 0; k < ROW_LENGTH; k++) {
              rx_data[i][j][k] = 1;
          }
      }
  }

  HAL_GPIO_WritePin(COL_0, GPIO_PIN_SET); //drive the first column

  __HAL_TIM_CLEAR_IT(&htim2, TIM_IT_UPDATE);    //clear interrupt request BEFORE enabling tim interrupt
  HAL_TIM_Base_Start_IT(&htim2);                //start TIM2 in interrupt mode

  /* USER CODE END 2 */
```

In the timer’s callback, for each row of the selected column we read the button’s state, we debounce it and we send the corresponding value in case the button is not kept pressed from the previous reading.
Here’s an example for the first row:

```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef *htim) {  //5ms elapsed
    if (htim == &htim2) {
        row_index = 0;

        //ROW_0
        rx_data[sample_index][row_index][col_index] = HAL_GPIO_ReadPin(ROW_0);  //save the btn state
        //debouncing: check if the btn is stably zero (check if many samples are = to 0)
        for (i = 0, btn = 0; i < NUM_OF_SAMPLES; i++)
            btn = btn || rx_data[i][row_index][col_index];

        //check if button already pressed: print only if previously not pressed, now pressed
        if (!btn && old_btn[row_index][col_index])
            HAL_UART_Transmit_DMA(&huart2, &out_matrix[row_index][col_index], 1);

        old_btn[row_index][col_index] = btn; //save the state for the next check
```

This is repeated for all rows incrementing “row_index”.

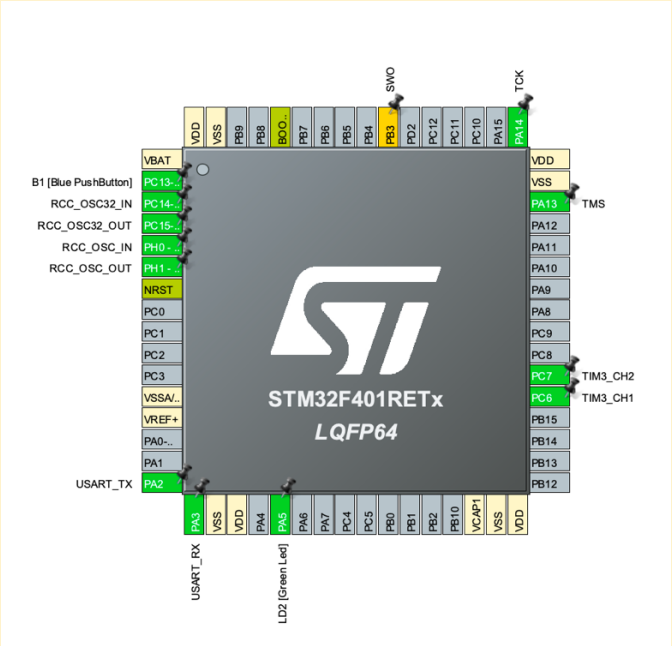
In the end, we increment “col_index” and if necessary “sample_index”.

Then we set the proper column and we reset the others for the next reading.

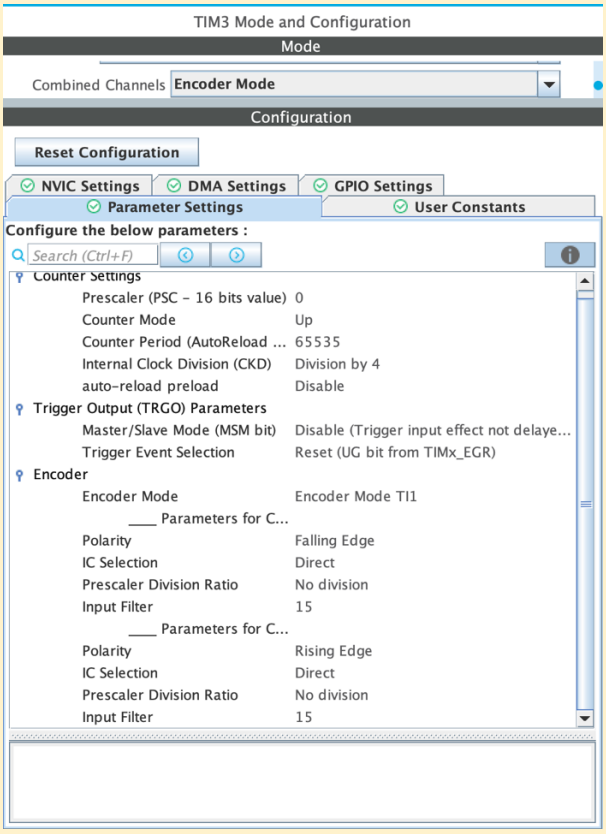
```
        col_index++;
        if (col_index == ROW_LENGTH) {
            col_index = 0;
            sample_index++; //need to acquire new sample
            if (sample_index == NUM_OF_SAMPLES)
                sample_index = 0;
        }

        //columns drive manager
        HAL_GPIO_WritePin(COL_0, (col_index == 0) ? GPIO_PIN_SET : GPIO_PIN_RESET);
        HAL_GPIO_WritePin(COL_1, (col_index == 1) ? GPIO_PIN_SET : GPIO_PIN_RESET);
        HAL_GPIO_WritePin(COL_2, (col_index == 2) ? GPIO_PIN_SET : GPIO_PIN_RESET);
        HAL_GPIO_WritePin(COL_3, (col_index == 3) ? GPIO_PIN_SET : GPIO_PIN_RESET);
    }
}

/* USER CODE END 0 */
```

Part 2:

In the ".ioc" we set PC6 and PC7 to use TIM3 in Encoder mode and we checked the proper configuration of the UART pins.



Then we enabled TIM3 in Encoder mode TI1 with the proper polarity and setting the timer's digital filter at 15.



TIM2 is configured in interrupt mode to enter in the ISR each second.

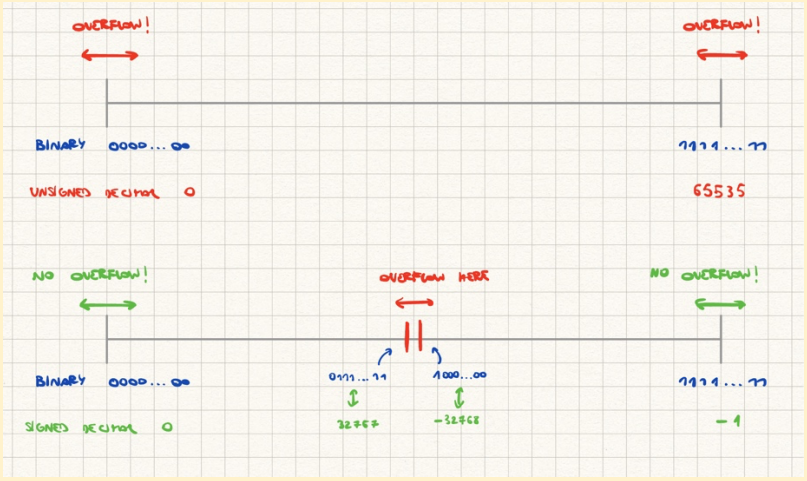
We enabled USART2 in DMA mode as in the previous project.

These are our variables and our defines:

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

#define STR_LEN 32
#define FULL_CYCLE_COUNT 24.0
#define SEC_TO_MIN 60

/* USER CODE END PD */
```

```
/* USER CODE BEGIN PV */

//signed to solve overflow/underflow problem of timer counter
int16_t counter = 0;
int16_t old_counter = 0;
int16_t delta = 0;

float rpm_value = 0;

int length = 0;
char string[STR_LEN];

/* USER CODE END PV */
```

We collect the values from the timer's counter as signed integer to solve the overflow/underflow problem.



As shown in the drawing: if we used unsigned integers we easily face an underflow problem when the delta is calculated around 0/65535, while using signed integers (cpl2 format) this problem is moved in the middle of the counter range, which is hardly reached in our application.

In the main function we start our timers in their corresponding modes.

```
  /* USER CODE BEGIN 2 */

  HAL_TIM_Encoder_Start(&htim3, TIM_CHANNEL_ALL);   //start TIM3 in encoder mode
  __HAL_TIM_CLEAR_IT(&htim2, TIM_IT_UPDATE);        //clear interrupt request BEFORE enabling tim interrupt
  HAL_TIM_Base_Start_IT(&htim2);                    //start TIM2 in interrupt mode

  /* USER CODE END 2 */
```

In the timer's ISR we acquire the new counter value and compute the difference with respect to the old one. Then we convert it into displacement dividing by "FULL_CYCLE_COUNT"; the displacement value corresponds to the rps one because we acquire each second (rps = displacement/1s).

Finally rpm are obtained by multiplying by 60, and the value is sent to the PC.

```
84 void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef *htim) {  //timer 2 callback called every second
85
86     if (htim == &htim2) {
87
88         counter = __HAL_TIM_GET_COUNTER(&htim3);
89         delta = counter - old_counter;
90         old_counter = counter;
91
92         rpm_value = (delta/FULL_CYCLE_COUNT)*SEC_TO_MIN;    //convert in rpm
93
94         int length = snprintf(string, sizeof(string), "Speed: %.1frpm\n", rpm_value);
95         HAL_UART_Transmit_DMA(&huart2, string, length);
96     }
97 }
98
99 /* USER CODE END 0 */
```