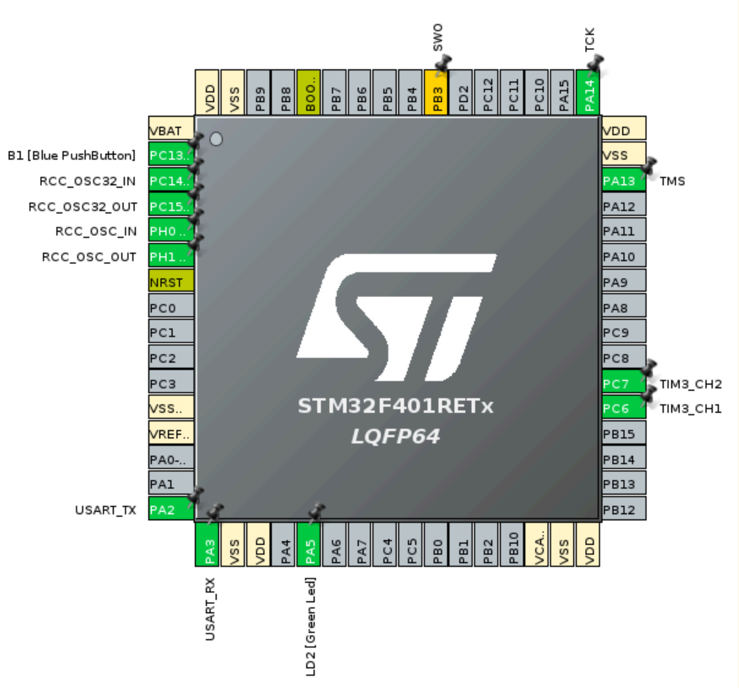
Mark	**1/11**

Team name:	*B5*		
Homework number:	*HOMEWORK 10*		
Due date:	3/12/24		
Contribution	NO	Partial	Full
Marenghi Manuela			*x*
Fellegara Tommaso			*x*
Giammusso Samuele			*x*
Cattani Luca			*x*
Csata Dániel			*x*
Notes: none			

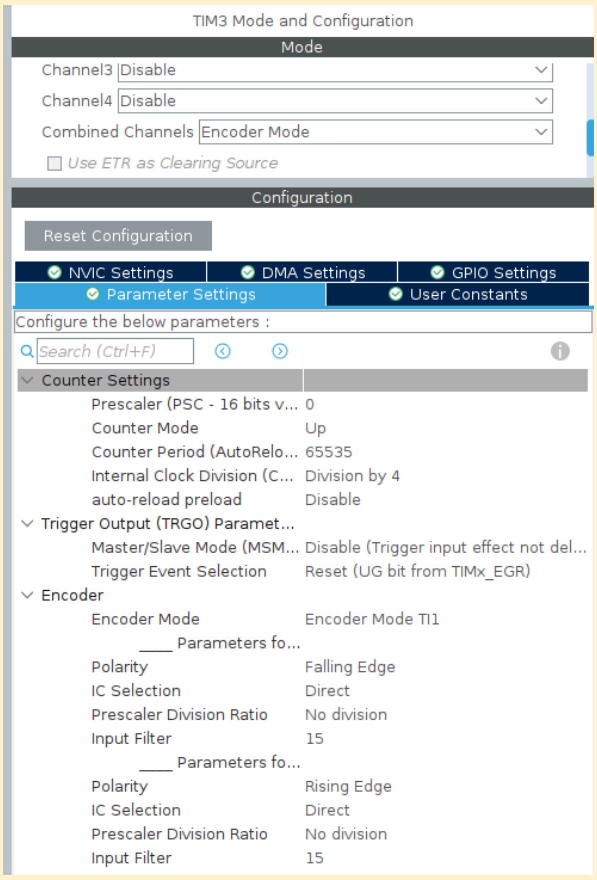
Project name	Test		
Not done	Partially done (major problems)	Partially done (minor problems)	Completed
			x

Encoder project:

- Setup the .ioc file:
  - TIM3:
    - Set the pins PC7 to TIM3_CH2 and PC6 to TIM3_CH1 that are the dedicated pins for the encoder;



- Set the Combined Channels the Encoder Mode;
- Set Internal Clock Division to ‘divided by 4’ in Parameter Settings;
- Set input FIlter to 15 in Encoder section of Parameter Settings for both channels to avoid hardware debouncing;
- Set the polarity for channel 1 to Falling edge
- Set the polarity for channel 2 to Rising edge



TIM3 Mode and Configuration

Mode

Channel3 Disable

Channel4 Disable

Combined Channels Encoder Mode

□ Use ETR as Clearing Source

Configuration

Reset Configuration

NVIC Settings | DMA Settings | GPIO Settings | Parameter Settings | User Constants

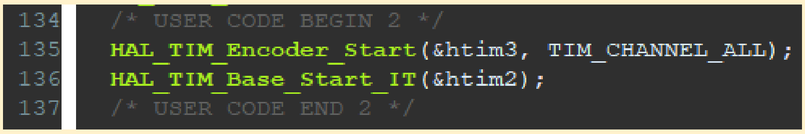
Configure the below parameters :

Search (Ctrl+F)

Counter Settings	
Prescaler (PSC - 16 bits v...	0
Counter Mode	Up
Counter Period (AutoRelo...	65535
Internal Clock Division (C...	Division by 4
auto-reload preload	Disable
Trigger Output (TRGO) Paramet...	
Master/Slave Mode (MSM...	Disable (Trigger input effect not del...
Trigger Event Selection	Reset (UG bit from TIMx_EGR)
Encoder	
Encoder Mode	Encoder Mode TI1
____ Parameters fo...	
Polarity	Falling Edge
IC Selection	Direct
Prescaler Division Ratio	No division
Input Filter	15
____ Parameters fo...	
Polarity	Rising Edge
IC Selection	Direct
Prescaler Division Ratio	No division
Input Filter	15

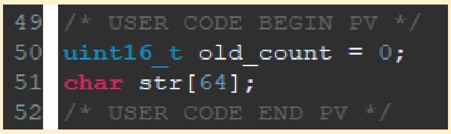
    - TIM2:
        - Set the Clock Source as Internal Clock;
        - We've decided to send every 1 second so we set Prescaler to 8400-1 and Counter Period to 10000-1;
        - Enable the global interrupt in the NVIC Settings;
    - UART:
        - Enable the global interrupt and the DMA;

- How we implemented the code:
    - In the main(), we initialize the timer and the encoder using functions HAL_TIM_Encoder_Start(&htim3, TIM_CHANNEL_ALL) where we specified to use both channels and HAL_TIM_Base_Start_IT(&htim2) to enable the interrupt mode;

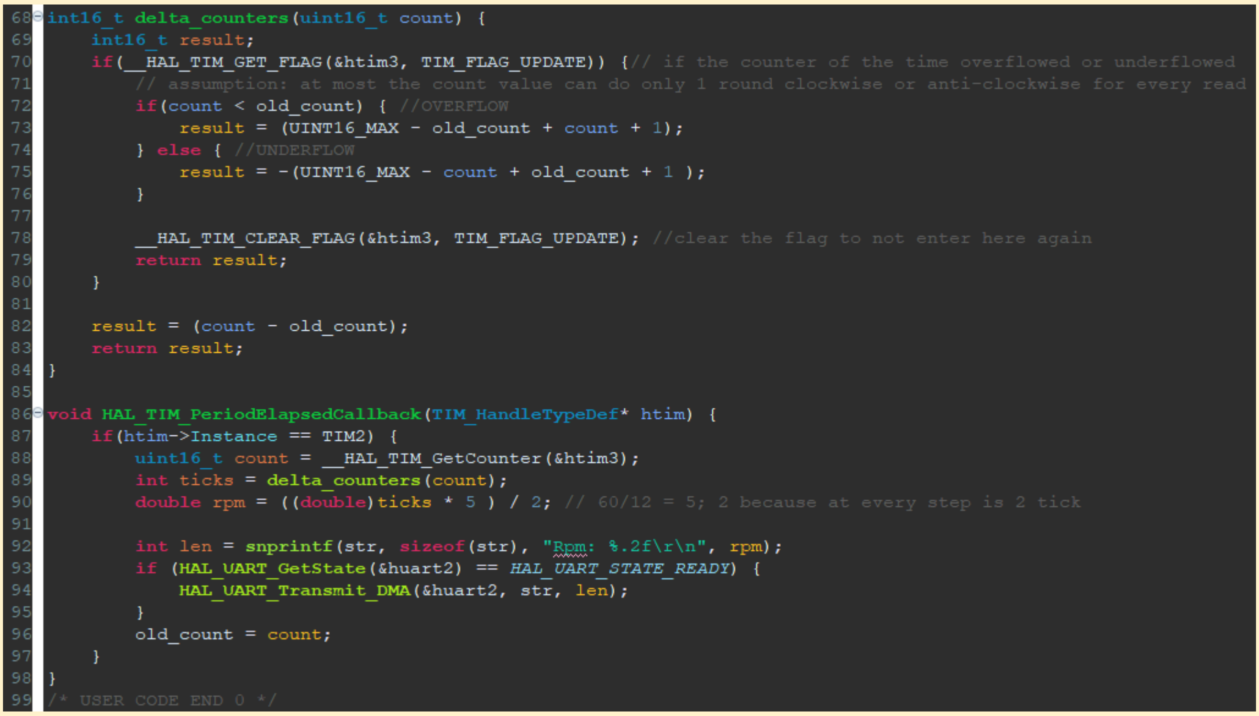


```
  /* USER CODE BEGIN 2 */
  HAL_TIM_Encoder_Start(&htim3, TIM_CHANNEL_ALL);
  HAL_TIM_Base_Start_IT(&htim2);
  /* USER CODE END 2 */
```

- Periodically computing the RPM (every second):
    - We implemented the *HAL_TIM_PeriodElapsedCallback* function to read the counter value and convert it to RPM, in particular:
        - We read the counter value via the *__HAL_TIM_GetCounter* macro
        - Compute the number of ticks the encoder has been turned taking overflow and underflow into account (via the *delta_counters* function)
        - Since it takes 12 ticks to make a full rotation, we have to take the number of ticks, divide by 12 and multiply by 60 to get the rotations per minute (i.e., multiply by 5). Then we divide it by 2 because every step is 2 ticks.
        - The result is then used to write a string which is transmitted via DMA using the UART interface.
- Overflow and underflow (*delta_counters* function):
    - The counter is a 16 bit unsigned integer value, which can overflow once the counter goes over UINT16_MAX (65'535), and can underflow if it goes below 0.
    - The flag TIM_FLAG_UPDATE of timer 3 becomes '1' when the timer over/under-flow.
    - For the Overflow we sum: the distance between the older value to the maximum +the new value of the counter + 1(because there is 1 bit that is causing the overflowing)
    - For the Underflow is handled similarly, except the sign is changed (since we are traveling counterclockwise) and old_count and count roles are switched.
    - If it's not under/over-flowing, then simply calculate the result with the difference between the new and the old value of the counter.

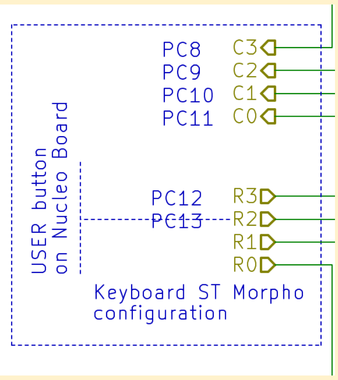


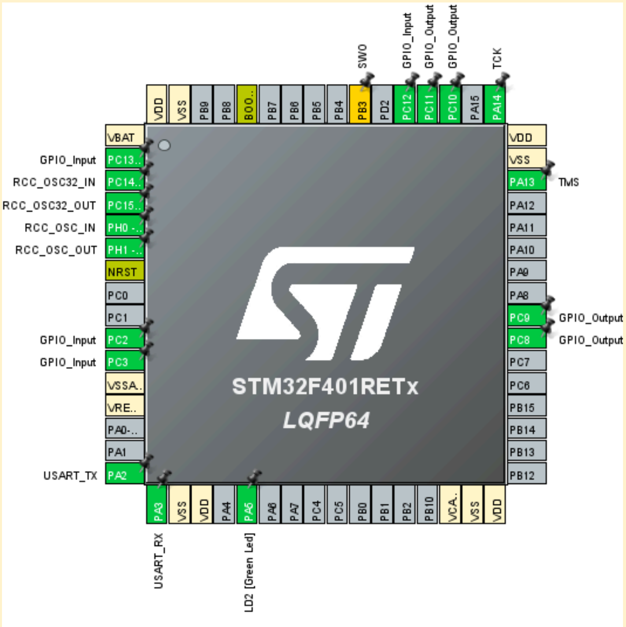
```c
/* USER CODE BEGIN PV */
uint16_t old_count = 0;
char str[64];
/* USER CODE END PV */
```



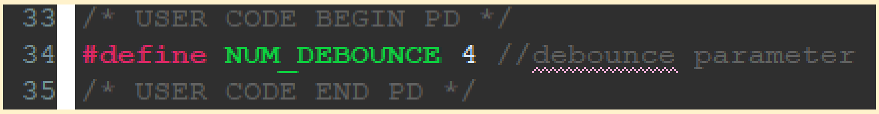
```c
int16_t delta_counters(uint16_t count) {
    int16_t result;
    if(__HAL_TIM_GET_FLAG(&htim3, TIM_FLAG_UPDATE)) {// if the counter of the time overflowed or underflowed
        // assumption: at most the count value can do only 1 round clockwise or anti-clockwise for every read
        if(count < old_count) { //OVERFLOW
            result = (UINT16_MAX - old_count + count + 1);
        } else { //UNDERFLOW
            result = -(UINT16_MAX - count + old_count + 1 );
        }

        __HAL_TIM_CLEAR_FLAG(&htim3, TIM_FLAG_UPDATE); //clear the flag to not enter here again
        return result;
    }

    result = (count - old_count);
    return result;
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim) {
    if(htim->Instance == TIM2) {
        uint16_t count = __HAL_TIM_GetCounter(&htim3);
        int ticks = delta_counters(count);
        double rpm = ((double)ticks * 5 ) / 2; // 60/12 = 5; 2 because at every step is 2 tick

        int len = snprintf(str, sizeof(str), "Rpm: %.2f\r\n", rpm);
        if (HAL_UART_GetState(&huart2) == HAL_UART_STATE_READY) {
            HAL_UART_Transmit_DMA(&huart2, str, len);
        }
        old_count = count;
    }
}
/* USER CODE END 0 */
```

Keyboard Project

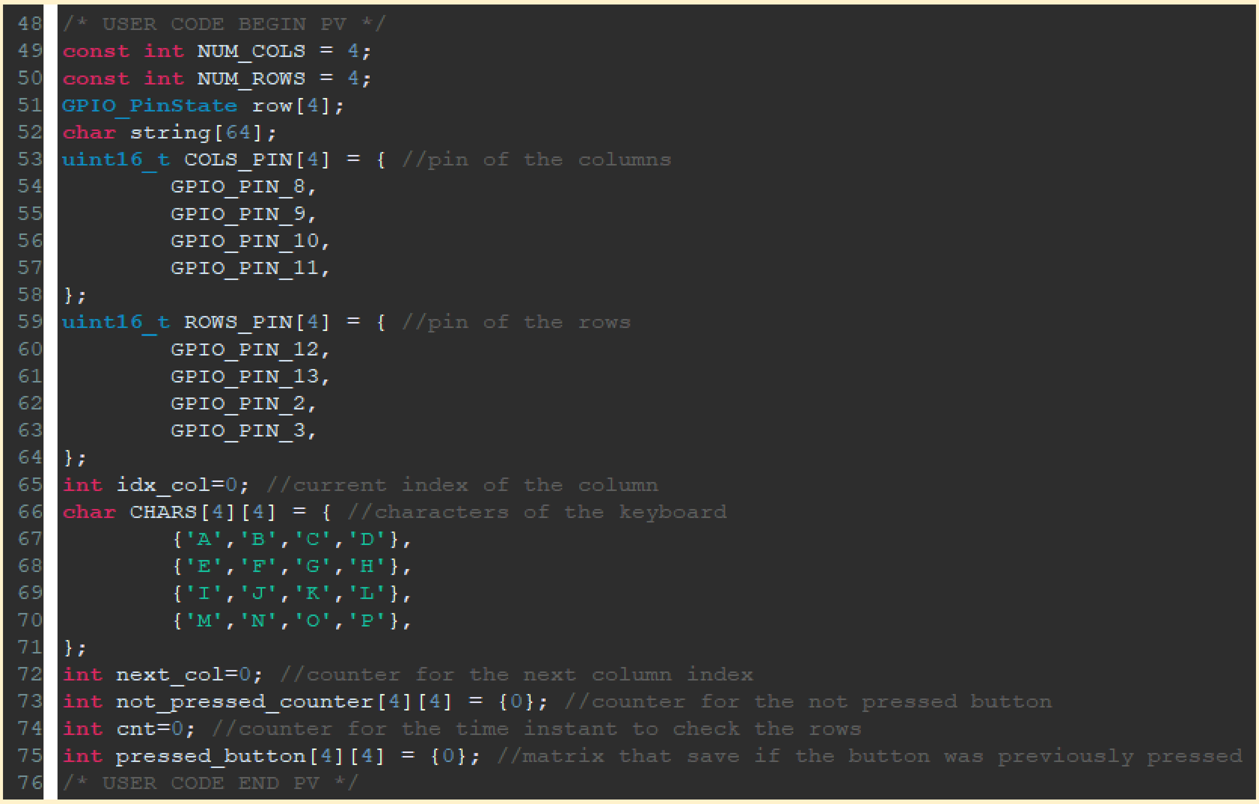
- Setup of the .ioc file:
  - In the GUI we set the pins of the columns (8-9-10-11) to GPIO_Output and the pins of the rows (12-13-2-3) to GPIO_Input
  - 

  - 

  - TIM2:
    - Set the Clock Source as Internal Clock;
    - We've decided to trigger the interrupt every 1 ms so we set Prescaler to 8400-1 and Counter Period to 10-1;
    - Enable the global interrupt in the NVIC Settings;
  - UART:
    - Enable the global interrupt and the DMA;
- How we implemented the code:
  - we defined this define and private variables, as commented in the code below:

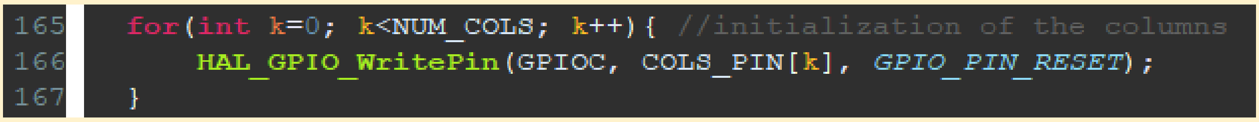


```
/* USER CODE BEGIN PD */
#define NUM_DEBOUNCE 4 //debounce parameter
/* USER CODE END PD */
```



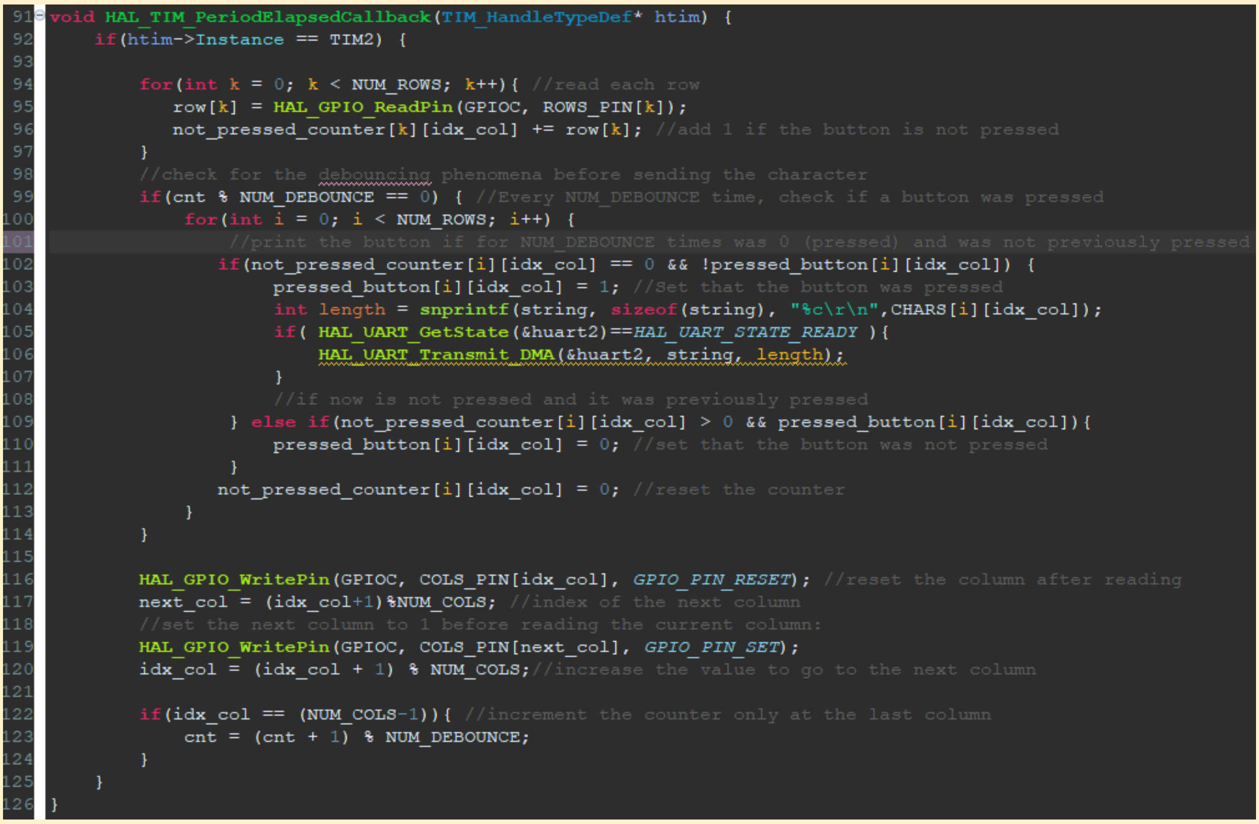
```c
/* USER CODE BEGIN PV */
const int NUM_COLS = 4;
const int NUM_ROWS = 4;
GPIO_PinState row[4];
char string[64];
uint16_t COLS_PIN[4] = { //pin of the columns
        GPIO_PIN_8,
        GPIO_PIN_9,
        GPIO_PIN_10,
        GPIO_PIN_11,
};
uint16_t ROWS_PIN[4] = { //pin of the rows
        GPIO_PIN_12,
        GPIO_PIN_13,
        GPIO_PIN_2,
        GPIO_PIN_3,
};
int idx_col=0; //current index of the column
char CHARS[4][4] = { //characters of the keyboard
        {'A','B','C','D'},
        {'E','F','G','H'},
        {'I','J','K','L'},
        {'M','N','O','P'},
};
int next_col=0; //counter for the next column index
int not_pressed_counter[4][4] = {0}; //counter for the not pressed button
int cnt=0; //counter for the time instant to check the rows
int pressed_button[4][4] = {0}; //matrix that save if the button was previously pressed
/* USER CODE END PV */
```

- in the main, we initialize the columns to RESET:



```c
    for(int k=0; k<NUM_COLS; k++){ //initialization of the columns
        HAL_GPIO_WritePin(GPIOC, COLS_PIN[k], GPIO_PIN_RESET);
    }
```

- the timer callback function is redefined as shown below:
    - even if it was not strictly necessary, we decided to implement a debouncing algorithm: we wait for 4 callbacks before validating an input, meaning that the duration is 1 ms (which is the timer callback) * 4 (which is NUM_DEBOUNCE) = 4ms



```c
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim) {
    if(htim->Instance == TIM2) {

        for(int k = 0; k < NUM_ROWS; k++){ //read each row
           row[k] = HAL_GPIO_ReadPin(GPIOC, ROWS_PIN[k]);
           not_pressed_counter[k][idx_col] += row[k]; //add 1 if the button is not pressed
        }
        //check for the debouncing phenomena before sending the character
        if(cnt % NUM_DEBOUNCE == 0) { //Every NUM_DEBOUNCE time, check if a button was pressed
            for(int i = 0; i < NUM_ROWS; i++) {
                //print the button if for NUM_DEBOUNCE times was 0 (pressed) and was not previously pressed
               if(not_pressed_counter[i][idx_col] == 0 && !pressed_button[i][idx_col]) {
                    pressed_button[i][idx_col] = 1; //Set that the button was pressed
                    int length = snprintf(string, sizeof(string), "%c\r\n",CHARS[i][idx_col]);
                    if( HAL_UART_GetState(&huart2)==HAL_UART_STATE_READY ){
                        HAL_UART_Transmit_DMA(&huart2, string, length);
                    }
                    //if now is not pressed and it was previously pressed
                } else if(not_pressed_counter[i][idx_col] > 0 && pressed_button[i][idx_col]){
                    pressed_button[i][idx_col] = 0; //set that the button was not pressed
                }
               not_pressed_counter[i][idx_col] = 0; //reset the counter
            }
        }

        HAL_GPIO_WritePin(GPIOC, COLS_PIN[idx_col], GPIO_PIN_RESET); //reset the column after reading
        next_col = (idx_col+1)%NUM_COLS; //index of the next column
        //set the next column to 1 before reading the current column:
        HAL_GPIO_WritePin(GPIOC, COLS_PIN[next_col], GPIO_PIN_SET);
        idx_col = (idx_col + 1) % NUM_COLS;//increase the value to go to the next column

        if(idx_col == (NUM_COLS-1)){ //increment the counter only at the last column
            cnt = (cnt + 1) % NUM_DEBOUNCE;
        }
    }
}
```

Professor comments: