Mark	

Team name:	*A1*		
Homework number:	*HOMEWORK 9*		
Due date:	03/12/2023		
Contribution	NO	Partial	Full
Pietro Monti			*x*
Alessia Moretto			*x*
Francesco Pallotto			*x*
Alessandro Perna			*x*
Ludovico Ventura			*x*
Notes:			

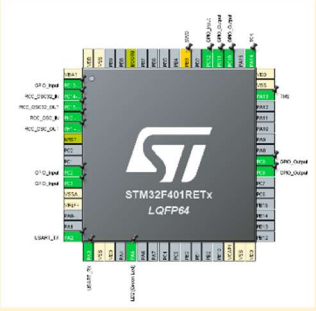
Project name	Keyboard Readout		
Not done	Partially done (major problems)	Partially done (minor problems)	Completed
			x

Explanation:
We successfully completed the homework.

1. Introduction:

- The aim of the provided code is to detect the pressure of a key in a 4x4 matrix keypad. The primary objective is to implement a robust mechanism for reading user inputs from the keypad, ensuring accuracy in key detection while mitigating issues such as contact bouncing.
- The code introduces a comprehensive approach to keypad interfacing by incorporating features like debouncing to eliminate signal fluctuations caused by mechanical switches. Additionally, it implements functionality for detecting both standard key presses and long presses, enhancing the versatility of user interaction.

1.1 Project Settings:
The pinout configurations is set as in the image:



The appropriate column pins are set as **GPIO_Outputs**, while the rows are set as **GPIO_Inputs.**

2.1 Defines & Macros

This part defines important constants used in the code. These constants play a crucial role in configuring the keypad and timers, ensuring versatility and generalization/reusability of the code:

```
#define COL_BANK GPIOC
#define ROW_BANK GPIOC
#define COL_NUM 4
#define ROW_NUM 4
#define DEBOUNCE_DELAY_MS 5
#define COLUMN_CHECK_DELAY_MS 5
#define LONG_PRESS_DURATION_MS 1000
```

- **COL_BANK and ROW_BANK**: Specifies the GPIO banks used for the columns and rows of the keypad matrix. In this code, both are set to GPIOC.
- **COL_NUM and ROW_NUM**: Defines the number of columns and rows in the keypad matrix. This code is designed for a 4x4 matrix, so COL_NUM is 4, and ROW_NUM is 4.
- **DEBOUNCE_DELAY_MS**: Specifies the duration (in milliseconds) for which the system waits for a key press to stabilize. This delay helps in mitigating contact bouncing issues common in mechanical switches.**(*)**
- **COLUMN_CHECK_DELAY_MS**: Sets the delay (in milliseconds) between successive column scans. It determines the rate at which the code iterates through the end of the scan of a column and the next one.
- **LONG_PRESS_DURATION_MS**: Defines the duration (in milliseconds) required for a key press to be considered a long press. If a key is held down for at least this duration, it is treated as a long press and therefore printed out.**(**)**

```
#define SET_START_TIMER(htim, delay) do { \
  __HAL_TIM_SET_AUTORELOAD(htim, delay*10); \
  __HAL_TIM_CLEAR_FLAG(htim, TIM_FLAG_UPDATE); \
  HAL_TIM_Base_Start_IT(htim); \
} while(0)
```

- **SET_START_TIMER:** Macro used to start a given timer in interrupt mode for a specified period of time (delay), multiplied by 10 since the prescaler of the timer is 8400-1. It also clears the flag of such timer. In the main function, TIM2 is started in this manner.

```
SET_START_TIMER(&htim2, COLUMN_CHECK_DELAY_MS);
```

(*)Note: The debounce time is particularly crucial in correctly reading the value of the second row, which is connected to GPIOC Pin 13: this pin is connected to a pull-up network, and requires enough time after key pressure for its value to be pulled down. The selected **DEBOUNCE_DELAY_MS** intrinsically solves this issue.

(**)Note: This setting can be changed to control how long a key needs to be pressed for it to count as a valid input. If set it to a value smaller then **DEBOUNCE_DELAY_MS**, or to 0 for simplicity, it'll just register a quick press without requiring a long press from the user, so a press of any duration. The system still handles any bouncing issues in this scenario.

2.2 Variables

Declares several global variables crucial for managing the keypad state and indices:

```
const uint16_t keymap[ROW_NUM][COL_NUM] = {
  {0x30, 0x31, 0x32, 0x33}, // {'0', '1', '2', '3'}
  {0x34, 0x35, 0x36, 0x37}, // {'4', '5', '6', '7'}
  {0x38, 0x39, 0x41, 0x42}, // {'8', '9', 'A', 'B'}
  {0x43, 0x44, 0x45, 0x46}  // {'C', 'D', 'E', 'F'}
};

const uint16_t rows_pin[ROW_NUM] = {GPIO_PIN_12, GPIO_PIN_13, GPIO_PIN_2, GPIO_PIN_3};
const uint16_t columns_pin[COL_NUM] = {GPIO_PIN_8, GPIO_PIN_9, GPIO_PIN_10, GPIO_PIN_11};

int keyboard_state[ROW_NUM][COL_NUM];
int keyboard_state_old[ROW_NUM][COL_NUM];

int col_idx = 0;
int row_idx = 0;
```

- **keymap:** A 4x4 matrix representing the characters associated with each key in the keypad. The characters are the hexadecimal representation of the ASCII code.
- **rows_pin & columns_pin:** Arrays containing the GPIO pins corresponding to the rows and columns of the keypad matrix(rows are in the physical order of the port).
- **keyboard_state & keyboard_state_old:** 2D arrays storing the current and previous state of each key in the matrix. These arrays are used for detecting key transitions.
- **col_idx & row_idx:** Variables representing the current column and row indices during keypad scanning.

3.1 TIM2 Timer Callback

The implementation of **HAL_TIM_PeriodElapsedCallback** associated with TIM2 has the purpose of triggering the column scanning process:

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
  if (htim==&htim2){
    scan_column();
  }
}
```

- When the arbitrarily chosen delay expires, the function **scan_column** is called, starting the detection of a key press.

3.2 Private Functions

3.2.1 Keypad Column Scanning: “void scan_column(void)”

```
void scan_column(void) {

  // Stop TIM2
  HAL_TIM_Base_Stop_IT(&htim2);

  // Copy last state of the keyboard
  memcpy(keyboard_state_old, keyboard_state, sizeof(keyboard_state));

  // Set column to scan
  HAL_GPIO_WritePin(COL_BANK, columns_pin[col_idx], GPIO_PIN_SET);

  // Scan through all the rows
  for (int row_idx = 0; row_idx < ROW_NUM; row_idx++) {
    debounce(DEBOUNCE_DELAY_MS);
    check_and_transmit_key(row_idx, col_idx);
  }

  // Reset column scanned
  HAL_GPIO_WritePin(COL_BANK, columns_pin[col_idx], GPIO_PIN_RESET);

  // Update column index
  col_idx = (col_idx < COL_NUM) ? col_idx + 1 : 0;

  // Restart TIM2
  SET_START_TIMER(&htim2, COLUMN_CHECK_DELAY_MS);

}

```

The **scan_column** function orchestrates the scanning of keypad columns, incorporating debouncing and key press detection:

- **Timer Stop and State Copying:** Temporarily halts TIM2**(***)** and copies the current keyboard state to allow for the comparison needed for key press detection.
- **Column Activation:** Sets the specified column for scanning by activating the corresponding GPIO pin.
- **Row-wise Scanning:** Iterates through all rows, invoking the **debounce** and **check_and_transmit_key** functions for each key in the active column.
- **Column Reset and Index Update:** Resets the scanned column and updates the column index, ensuring a comprehensive scan across the keypad.
- **Timer Restart:** Restarts TIM2, initiating the next round of periodic column scanning.

(***)Note: TIM2 is halted to optimize press duration measurement and minimize false detections. Instead of continuously scanning columns at set intervals, the timer is stopped while processing rows within each column. This ensures precise key press duration measurement, eliminating the reliance on users holding a key down for a specific consecutive read count (dictated by multiples of the time required to scan all 4 columns). Pausing column scans allows for more frequent checks of the button state, reducing the chance of a button being released and re-pressed between scans.

3.2.2 Debouncing Mechanism: “void debounce(int debounce_delay_ms)”

```c
void debounce(int debounce_delay_ms) {
  SET_START_TIMER(&htim3, debounce_delay_ms);
  while(__HAL_TIM_GET_FLAG(&htim3, TIM_FLAG_UPDATE) == RESET){}
  HAL_TIM_Base_Stop_IT(&htim3);
}
```

The **debounce** function is responsible for implementing a simple yet effective debouncing mechanism to filter out noise in the keypad signals:

- **Initialization with Timer:** Initiates a timer (TIM3) with a specified delay (**debounce_delay_ms**). This timer is used to introduce a controlled delay before the function proceeds, allowing for the stabilization of the keypad signal.
- **Timer Wait Loop:** Utilizes a loop that continues until the timer's update flag is set, indicating the completion of the delay. This loop ensures that the program remains in a wait state until the specified debounce delay elapses.
- **Timer Stop:** Stops the timer once the delay is completed, concluding the debounce process.

3.2.3 Key Press Detection & Transmission: “void check_and_transmit_key(int row_idx, int col_idx)”

The **check_and_transmit_key** function is responsible for updating the keyboard state, detecting key presses, and transmitting corresponding UART data:

```c
void check_and_transmit_key(int row_idx, int col_idx) {
  if (!HAL_GPIO_ReadPin(ROW_BANK, rows_pin[row_idx])) {
      keyboard_state[row_idx][col_idx] = 1;
      if (LONG_PRESS_DURATION_MS > DEBOUNCE_DELAY_MS) {
          SET_START_TIMER(&htim3, (LONG_PRESS_DURATION_MS - DEBOUNCE_DELAY_MS));
          while (__HAL_TIM_GET_FLAG(&htim3, TIM_FLAG_UPDATE) == RESET) {
              if (HAL_GPIO_ReadPin(ROW_BANK, rows_pin[row_idx])) {
                  keyboard_state[row_idx][col_idx] = 0;
              }
          }
        HAL_TIM_Base_Stop_IT(&htim3);
      }
  } else {
      keyboard_state[row_idx][col_idx] = 0;
  }
  if (keyboard_state[row_idx][col_idx] == 1 && keyboard_state_old[row_idx][col_idx] == 0) {
      uint16_t key = keymap[row_idx][col_idx];
      HAL_UART_Transmit(&huart2, (uint8_t*) &key, sizeof(key), 10);
  }
}
```

- **Condition for Key Press:** The function initiates by checking if the corresponding row pin is low, indicating a key press. If true, it proceeds to temporarily assert the **keyboard_state** array for the specific key to 1. If not, the specific key is set to 0.
- **Press duration with Timer:** If the key is pressed, the function starts a timer (TIM3) to measure the duration of the key press, taking into account the intrinsic delay of the debounce mechanism (**DEBOUNCE_DELAY_MS**).
- **Conditional Check Loop:** While the timer is active, the function enters a loop, monitoring the row pin. If the row pin returns to a high state (indicating key release) before the timer elapses, it signifies an early release of that key, and the key state is reset to 0. At the end of such loop, the TIM3 is stopped.
- **UART Transmission:** If the key state transitions from 0 to 1 and the previous state was 0, indicating a newly pressed key, the function proceeds to transmit the corresponding character over UART.

Professor comments: