Московский государственный технический университет им. Н.Э. Баумана

Факультет «Радиоэлектроника и лазерная техника (РЛ)»

Кафедра «Технология приборостроения (РЛ6)»

Домашнее задание №2,

Лабораторная работа №2

по дисциплине «Цифровые устройства и микропроцессоры»

Выполнил ст. группы РЛ6-69

Лобанов Д.Д.

Преподаватель Семеренко Д.А.

Москва, 2023

Оглавление

Цель

Выполнить передачу данных по параллельному интерфейсу, полученных по последовательному интерфейсу.

Условия

Параллельный интерфейс (запись)

После считывания данных по последовательному интерфейсу их необходимо передать по параллельному интерфейсу. После установления данных на выводах микроконтроллера (параллельный интерфейс) устанавливается лог. 1 на выводе «en». Этот логический сигнал устанавливается в лог 0, после того как данные были считаны, т. е. после перехода уровня сигнала «ready» из лог. 1 в лог 0.

Последовательный интерфейс (чтение)

Считывание данных осуществляется после установления сигнала «d_send» в лог 0. Как показано на рисунке 1 и 2. Длительность лог 0 и лог 1 определяется входами f0 ... f2.

На рисунке 3 показана схема подключения сигналов МК. Частота передачи данных по последовательному интерфейсу определяется установкой соответствующих логических сигналов на входы контроллера f0... f2. Частота передачи данных меняется дискретно от 100Гц до 10кГц (шаг дискретизации выбирается самостоятельно). Предусмотреть возникновение ситуации получения данных по последовательному интерфейсу быстрее, чем их считывание по параллельному.

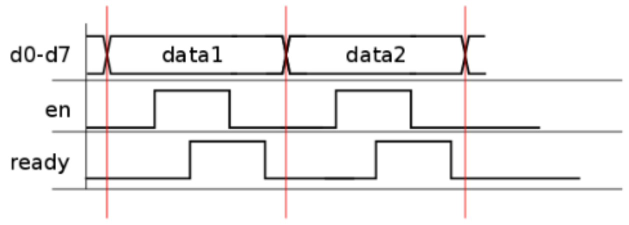


Рисунок 1 – Временные диаграммы установления сигналов на входах микроконтроллера

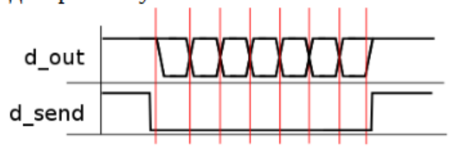


Рисунок 2 – Временные диаграммы установления сигналов на выходах микроконтроллеа

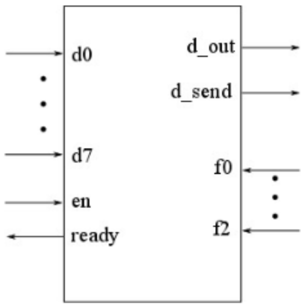
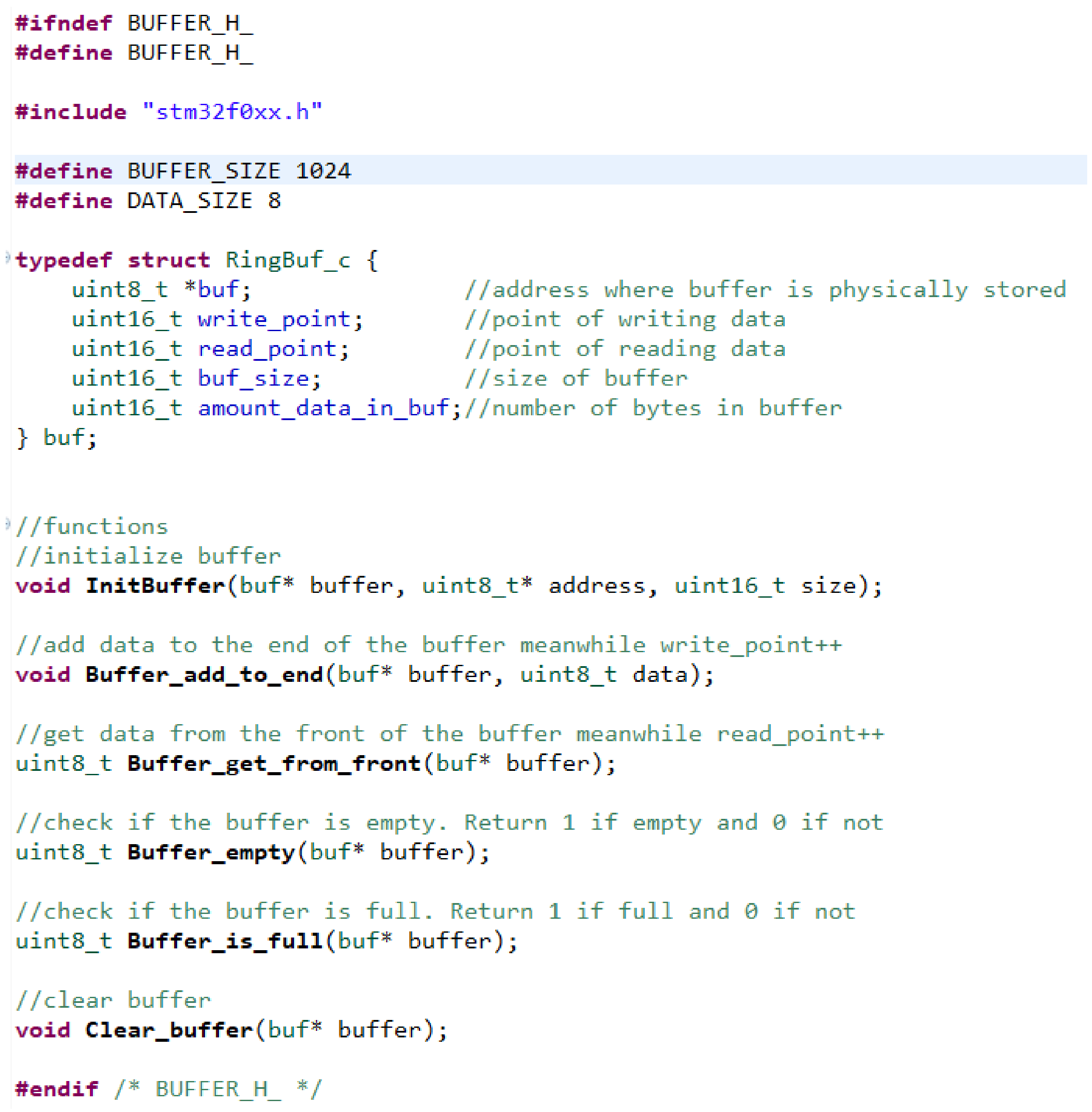


Рисунок 3 – Схема выводов микроконтроллера

buffer.h

Буффер предназначен для временного хранения данных и используется при записи данных во Flash, а также при передаче данных и их получении с другого микроконтроллера.



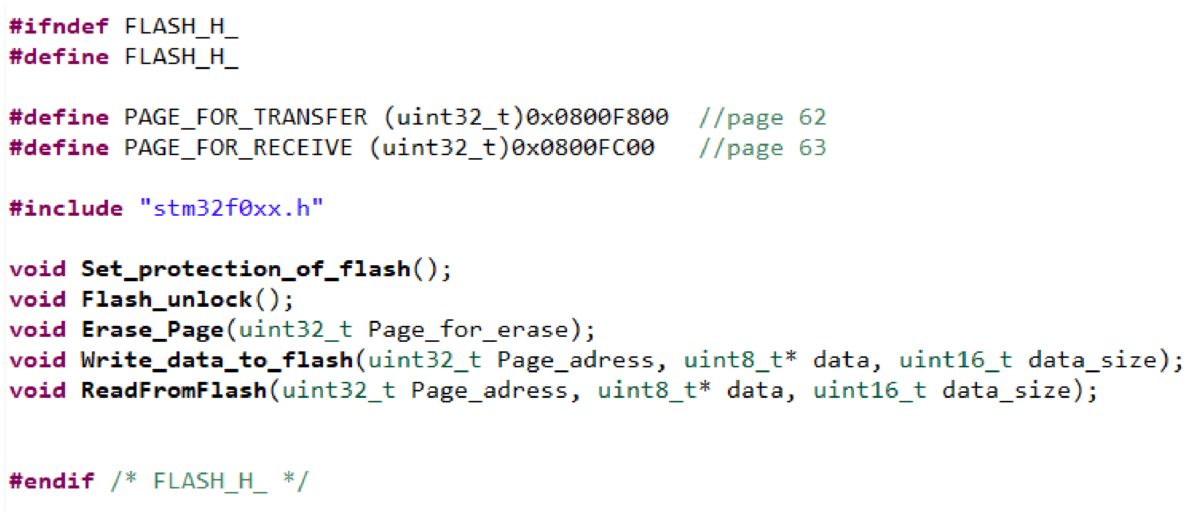
```c
#ifndef BUFFER_H_
#define BUFFER_H_

#include "stm32f0xx.h"

#define BUFFER_SIZE 1024
#define DATA_SIZE 8

typedef struct RingBuf_c {
    uint8_t *buf;               //address where buffer is physically stored
    uint16_t write_point;       //point of writing data
    uint16_t read_point;        //point of reading data
    uint16_t buf_size;          //size of buffer
    uint16_t amount_data_in_buf;//number of bytes in buffer
} buf;


//functions
//initialize buffer
void InitBuffer(buf* buffer, uint8_t* address, uint16_t size);

//add data to the end of the buffer meanwhile write_point++
void Buffer_add_to_end(buf* buffer, uint8_t data);

//get data from the front of the buffer meanwhile read_point++
uint8_t Buffer_get_from_front(buf* buffer);

//check if the buffer is empty. Return 1 if empty and 0 if not
uint8_t Buffer_empty(buf* buffer);

//check if the buffer is full. Return 1 if full and 0 if not
uint8_t Buffer_is_full(buf* buffer);

//clear buffer
void Clear_buffer(buf* buffer);

#endif /* BUFFER_H_ */
```

buffer.c

```c
#include "buffer.h"

//initialize buffer
void InitBuffer(buf* buffer, uint8_t* address, uint16_t size) {
    buffer->buf = address; //set address of buf
    //buffer is empty so read_point and write_point are in the beginning
    buffer->read_point = buffer->write_point = 0;
    //set size of buffer, in our case it is 1024 bytes
    buffer->buf_size = size;
    //buffer is empty so
    buffer->amount_data_in_buf = 0;
}


//add data to the end of the buffer meanwhile write_point++
void Buffer_add_to_end(buf* buffer, uint8_t data) {
    if(buffer->amount_data_in_buf < BUFFER_SIZE) {
        buffer->buf[buffer->write_point] = data; //write data to the end
        buffer->write_point++; //shift write_point
        buffer->amount_data_in_buf++; //increase amount of data on 1

        if(buffer->write_point >= BUFFER_SIZE) // if write point greater than BUFFER_SIZE or equal it
            buffer->write_point = 0;           // then set write_point at the beginning of buffer
    }
}


//get data from the front of the buffer meanwhile read_point++
uint8_t Buffer_get_from_front(buf* buffer) {
    if(buffer->amount_data_in_buf != 0) {//if there are data
        uint8_t data = buffer->buf[buffer->read_point]; //get data
        buffer->read_point++; //shift read_point

        if(buffer->read_point >= BUFFER_SIZE) // if read point greater than BUFFER_SIZE or equal it
            buffer->read_point = 0;           // then set read_point at the beginning of buffer

        buffer->amount_data_in_buf--;
        return data;
    }
    return -1;
}

//check if the buffer is empty. Return 1 if empty and 0 if not
uint8_t Buffer_empty(buf* buffer) {
    return ((buffer->read_point == buffer->write_point) && (buffer->amount_data_in_buf == 0));
}

uint8_t Buffer_is_full(buf* buffer) {
    return (buffer->amount_data_in_buf == BUFFER_SIZE);
}

void Clear_buffer(buf* buffer) {
    buffer->read_point = 0;
    buffer->write_point = 0;
    buffer->amount_data_in_buf = 0;
}
```

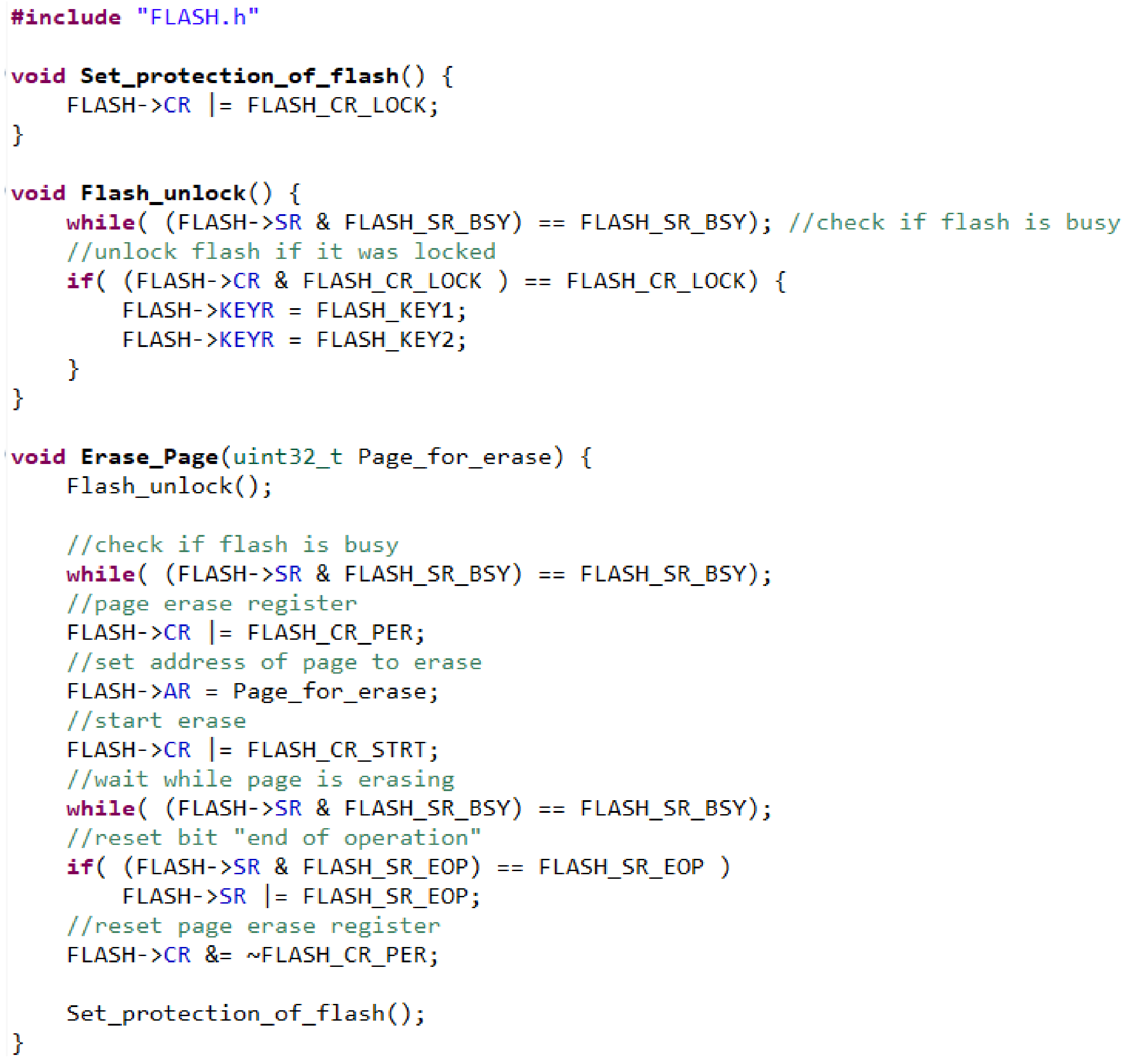
Flash.h

Во Flash записывается передаваемая (получаемая с ПК) и полученная с другого микроконтроллера информация.



```
#ifndef FLASH_H_
#define FLASH_H_

#define PAGE_FOR_TRANSFER (uint32_t)0x0800F800  //page 62
#define PAGE_FOR_RECEIVE (uint32_t)0x0800FC00   //page 63

#include "stm32f0xx.h"

void Set_protection_of_flash();
void Flash_unlock();
void Erase_Page(uint32_t Page_for_erase);
void Write_data_to_flash(uint32_t Page_adress, uint8_t* data, uint16_t data_size);
void ReadFromFlash(uint32_t Page_adress, uint8_t* data, uint16_t data_size);


#endif /* FLASH_H_ */
```

Flash.c

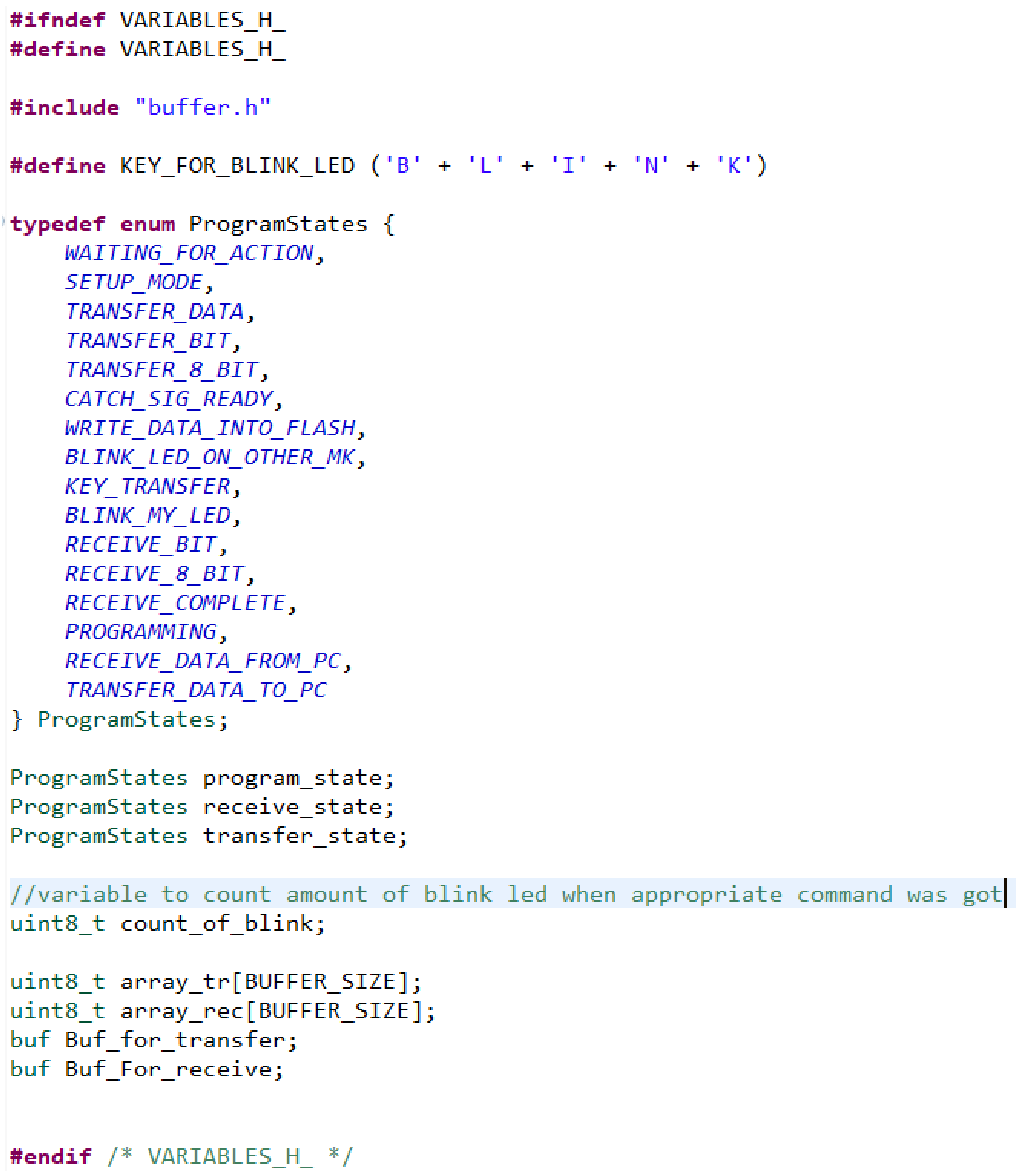


```c
#include "FLASH.h"

void Set_protection_of_flash() {
    FLASH->CR |= FLASH_CR_LOCK;
}

void Flash_unlock() {
    while( (FLASH->SR & FLASH_SR_BSY) == FLASH_SR_BSY); //check if flash is busy
    //unlock flash if it was locked
    if( (FLASH->CR & FLASH_CR_LOCK ) == FLASH_CR_LOCK) {
        FLASH->KEYR = FLASH_KEY1;
        FLASH->KEYR = FLASH_KEY2;
    }
}

void Erase_Page(uint32_t Page_for_erase) {
    Flash_unlock();

    //check if flash is busy
    while( (FLASH->SR & FLASH_SR_BSY) == FLASH_SR_BSY);
    //page erase register
    FLASH->CR |= FLASH_CR_PER;
    //set address of page to erase
    FLASH->AR = Page_for_erase;
    //start erase
    FLASH->CR |= FLASH_CR_STRT;
    //wait while page is erasing
    while( (FLASH->SR & FLASH_SR_BSY) == FLASH_SR_BSY);
    //reset bit "end of operation"
    if( (FLASH->SR & FLASH_SR_EOP) == FLASH_SR_EOP )
        FLASH->SR |= FLASH_SR_EOP;
    //reset page erase register
    FLASH->CR &= ~FLASH_CR_PER;

    Set_protection_of_flash();
}
```

```
void Write_data_to_flash(uint32_t Page_adress, uint8_t* data, uint16_t data_size) {
    Erase_Page(Page_adress);
    Flash_unlock();

    //set programming bit
    FLASH->CR |= FLASH_CR_PG;
    //write data to the flash address, in our case - 1064 byte (1 page)
    uint16_t data_for_write = 0;
    for (int i = 0; i < data_size / 2; i++) {
        data_for_write = 0;
        data_for_write |= data[i * 2];
        data_for_write |= ((data[i * 2 + 1]) << 8);
         *(__IO uint16_t*)(Page_adress + i*2) = data_for_write;
        while( (FLASH->SR & FLASH_SR_BSY) == FLASH_SR_BSY);
    }
    //wait while data is writing
    //reset bit "end of operation"
    if( (FLASH->SR & FLASH_SR_EOP) == FLASH_SR_EOP )
        FLASH->SR |= FLASH_SR_EOP;
    //reset programming bit
    FLASH->CR &= ~FLASH_CR_PG;

    Set_protection_of_flash();
}

void ReadFromFlash(uint32_t Page_adress, uint8_t* data, uint16_t data_size){
    uint16_t data_for_read = 0;
    for(int i = 0; i < data_size / 2; i++) {
      data_for_read = *(uint16_t*)(Page_adress + i*2);
      while( (FLASH->SR & FLASH_SR_BSY) == FLASH_SR_BSY);
      data[i * 2] = data_for_read & 0x00FF;
      data[i * 2 + 1] = (data_for_read >> 8) & 0x00FF;
    }
}
```

Variables.h

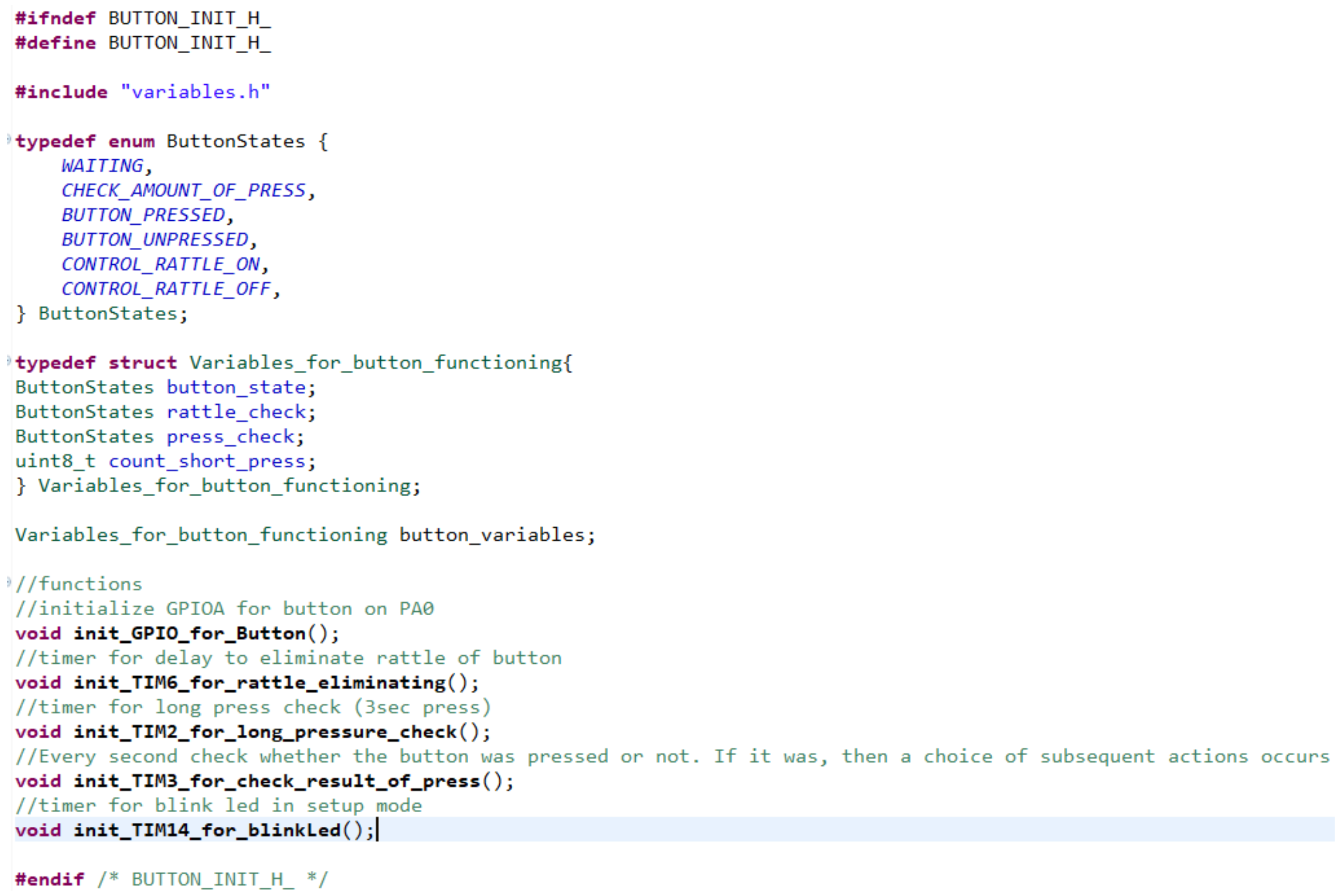
В данном файле размещены переменные, необходимые для отслеживания состояния выполнения программы (флаги), ключ для мигания диодом на другом микроконтроллере, а также буфферы для передаваемых и получаемых данных.



```
#ifndef VARIABLES_H_
#define VARIABLES_H_

#include "buffer.h"

#define KEY_FOR_BLINK_LED ('B' + 'L' + 'I' + 'N' + 'K')

typedef enum ProgramStates {
    WAITING_FOR_ACTION,
    SETUP_MODE,
    TRANSFER_DATA,
    TRANSFER_BIT,
    TRANSFER_8_BIT,
    CATCH_SIG_READY,
    WRITE_DATA_INTO_FLASH,
    BLINK_LED_ON_OTHER_MK,
    KEY_TRANSFER,
    BLINK_MY_LED,
    RECEIVE_BIT,
    RECEIVE_8_BIT,
    RECEIVE_COMPLETE,
    PROGRAMMING,
    RECEIVE_DATA_FROM_PC,
    TRANSFER_DATA_TO_PC
} ProgramStates;

ProgramStates program_state;
ProgramStates receive_state;
ProgramStates transfer_state;

//variable to count amount of blink led when appropriate command was got
uint8_t count_of_blink;

uint8_t array_tr[BUFFER_SIZE];
uint8_t array_rec[BUFFER_SIZE];
buf Buf_for_transfer;
buf Buf_For_receive;


#endif /* VARIABLES_H_ */
```

Button_init.h

В данных файлах размещены переменные для контролирования текущего состояния кнопки и нажатий, а также проинициализирована вся периферия для корректной работы кнопки.



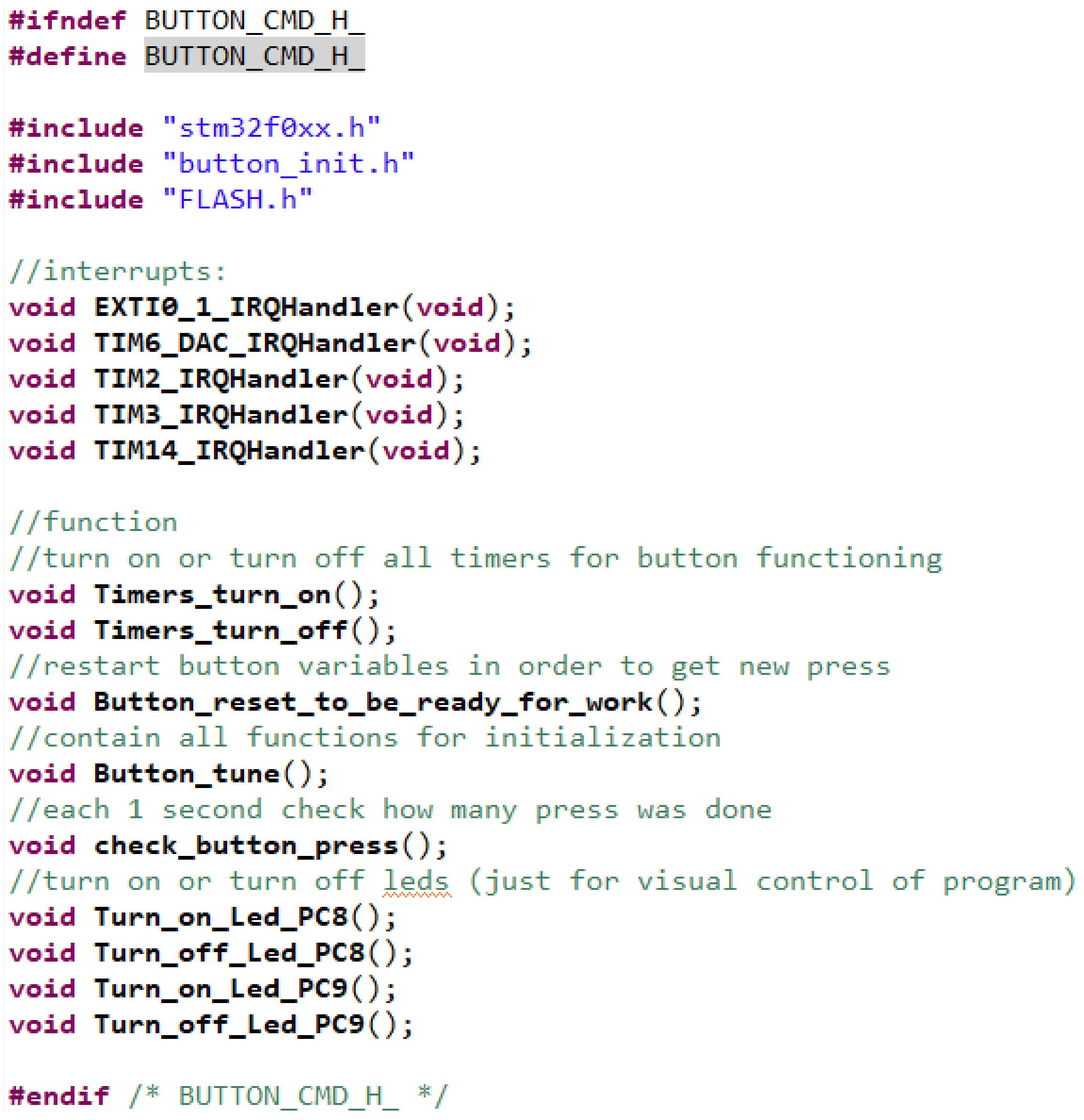
```c
#ifndef BUTTON_INIT_H_
#define BUTTON_INIT_H_

#include "variables.h"

typedef enum ButtonStates {
    WAITING,
    CHECK_AMOUNT_OF_PRESS,
    BUTTON_PRESSED,
    BUTTON_UNPRESSED,
    CONTROL_RATTLE_ON,
    CONTROL_RATTLE_OFF,
} ButtonStates;

typedef struct Variables_for_button_functioning{
ButtonStates button_state;
ButtonStates rattle_check;
ButtonStates press_check;
uint8_t count_short_press;
} Variables_for_button_functioning;

Variables_for_button_functioning button_variables;

//functions
//initialize GPIOA for button on PA0
void init_GPIO_for_Button();
//timer for delay to eliminate rattle of button
void init_TIM6_for_rattle_eliminating();
//timer for long press check (3sec press)
void init_TIM2_for_long_pressure_check();
//Every second check whether the button was pressed or not. If it was, then a choice of subsequent actions occurs
void init_TIM3_for_check_result_of_press();
//timer for blink led in setup mode
void init_TIM14_for_blinkLed();

#endif /* BUTTON_INIT_H_ */
```

Button_init.c

```c
#include "button_init.h"


void init_GPIO_for_Button() {
    RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
    RCC->AHBENR |= RCC_AHBENR_GPIOCEN;
    //input mode on PA0
    GPIOA->MODER &= ~GPIO_MODER_MODER0;
    //output mode for Led PC8 and PC9
    GPIOC->MODER |= GPIO_MODER_MODER8_0 | GPIO_MODER_MODER9_0;
    EXTI->IMR |= EXTI_IMR_IM0;   //interrupt mask register
    EXTI->RTSR |= EXTI_RTSR_RT0; //rising trigger selection
    EXTI->FTSR |= EXTI_FTSR_FT0; //falling trigger selection
    NVIC_EnableIRQ(EXTI0_1_IRQn);//tune NVIC for EXTI0
    NVIC_SetPriority(EXTI0_1_IRQn, 5);
}

void init_TIM6_for_rattle_eliminating () {
    RCC->APB1ENR |= RCC_APB1ENR_TIM6EN;
    TIM6->ARR = 4000; //tune TIM6 for 25ms
    TIM6->PSC = 50;
    //interrupt on
    TIM6->DIER |= TIM_DIER_UIE;
    NVIC_EnableIRQ(TIM6_DAC_IRQn);
    NVIC_SetPriority(TIM6_DAC_IRQn, 5);
    TIM6->CR1 |= TIM_CR1_CEN;
}

void init_TIM2_for_long_pressure_check() {
    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
    TIM2->ARR = 12000; //tune TIM2 for 3sec
    TIM2->PSC = 2000;
    //interrupt on
    TIM2->DIER |= TIM_DIER_UIE;
    NVIC_EnableIRQ(TIM2_IRQn);
    NVIC_SetPriority(TIM2_IRQn, 5);
    TIM2->CR1 |= TIM_CR1_CEN;
}
void init_TIM3_for_check_result_of_press() {
    RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;
    TIM3->ARR = 8000;//tune TIM3 for 1sec
    TIM3->PSC = 1000;
    //interrupt on
    TIM3->DIER |= TIM_DIER_UIE;
    NVIC_EnableIRQ(TIM3_IRQn);
    NVIC_SetPriority(TIM3_IRQn, 5);
    TIM3->CR1 |= TIM_CR1_CEN;
}

void init_TIM14_for_blinkLed() {
    RCC->APB1ENR |= RCC_APB1ENR_TIM14EN;
    TIM14->ARR = 4000;//tune TIM14 for 0.5sec
    TIM14->PSC = 1000;
    //interrupt on
    TIM14->DIER |= TIM_DIER_UIE;
    NVIC_EnableIRQ(TIM14_IRQn);
    NVIC_SetPriority(TIM14_IRQn, 5);
}
```

Button_cmd.h

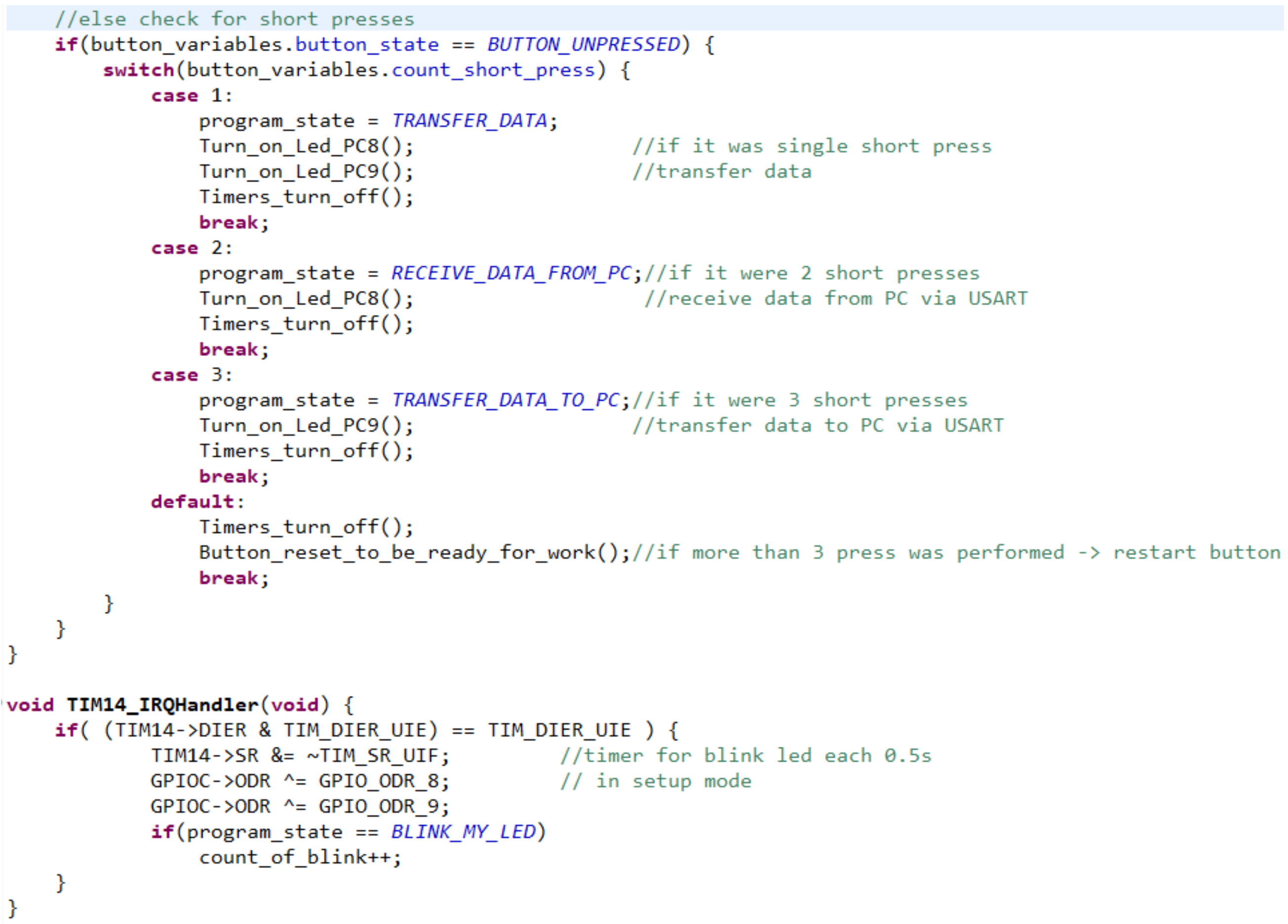
В данных файлах размещены функции для осуществления процесса обработки нажатий и последующих действий, связанных с ними.



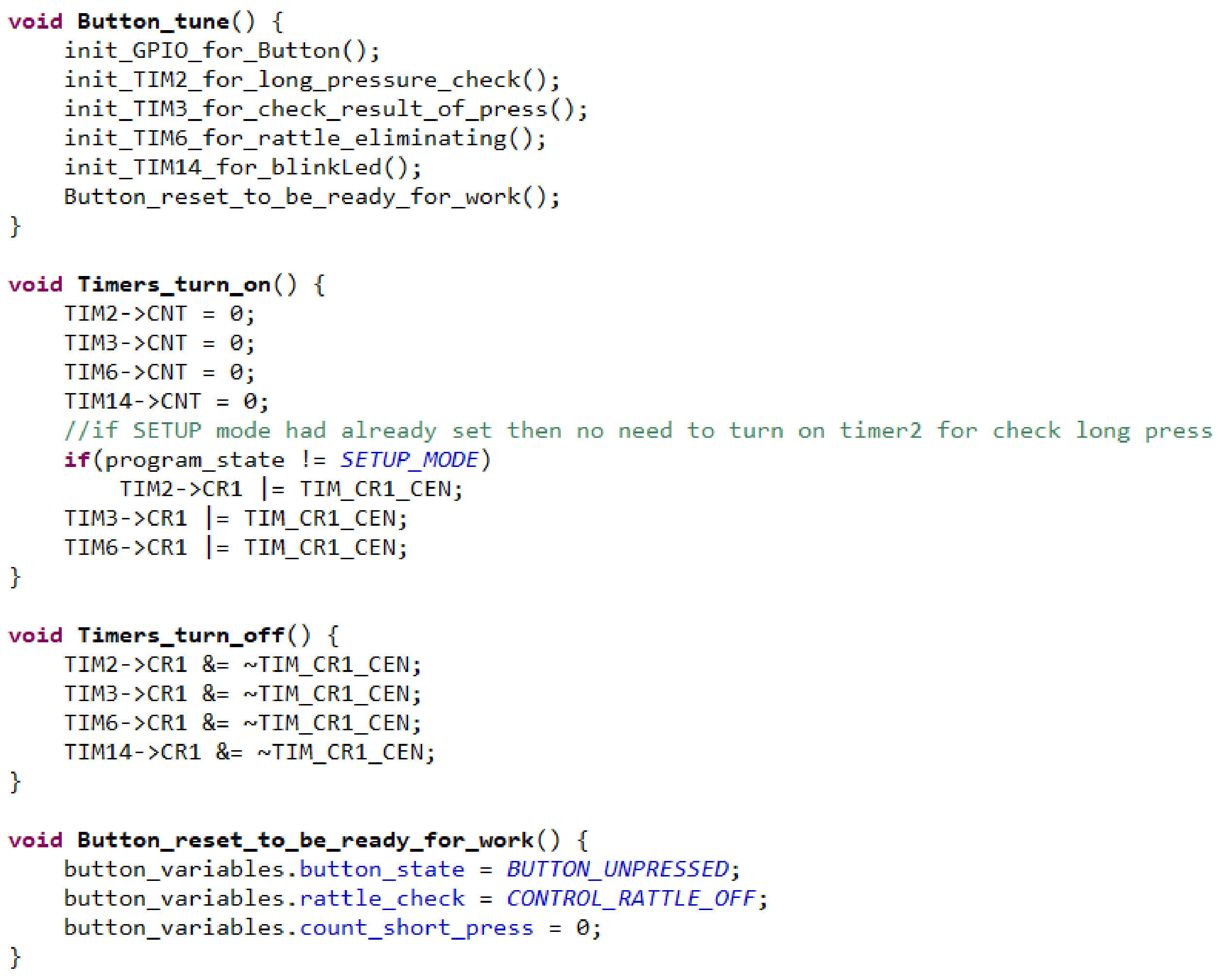
```c
#ifndef BUTTON_CMD_H_
#define BUTTON_CMD_H_

#include "stm32f0xx.h"
#include "button_init.h"
#include "FLASH.h"

//interrupts:
void EXTI0_1_IRQHandler(void);
void TIM6_DAC_IRQHandler(void);
void TIM2_IRQHandler(void);
void TIM3_IRQHandler(void);
void TIM14_IRQHandler(void);

//function
//turn on or turn off all timers for button functioning
void Timers_turn_on();
void Timers_turn_off();
//restart button variables in order to get new press
void Button_reset_to_be_ready_for_work();
//contain all functions for initialization
void Button_tune();
//each 1 second check how many press was done
void check_button_press();
//turn on or turn off leds (just for visual control of program)
void Turn_on_Led_PC8();
void Turn_off_Led_PC8();
void Turn_on_Led_PC9();
void Turn_off_Led_PC9();

#endif /* BUTTON_CMD_H_ */
```

Button_cmd.c

```c
#include <button_cmd.h>

void EXTI0_1_IRQHandler(void) {
    if( (EXTI->PR & EXTI_PR_PIF0) == EXTI_PR_PIF0) {
        EXTI->PR |= EXTI_PR_PIF0;
        //if check for RATTLE or program_state = PROGRAMMING -> return
        if(program_state == PROGRAMMING || button_variables.rattle_check == CONTROL_RATTLE_ON)
            return;
        //if enter then set CONTROL_RATTLE_ON due to eliminate rattle of button
        if(button_variables.button_state == BUTTON_UNPRESSED) {
            button_variables.button_state = BUTTON_PRESSED;     //button is pressed
            button_variables.rattle_check = CONTROL_RATTLE_ON;  //control rattle on
            button_variables.count_short_press++;               //increase number of press
            Timers_turn_on();
            return;
        }
        if(button_variables.button_state == BUTTON_PRESSED) {   //if button was unpressed
            button_variables.button_state = BUTTON_UNPRESSED;   //then it was falling front interrupt
            button_variables.rattle_check = CONTROL_RATTLE_ON;  //control rattle on
            Timers_turn_on();
        }
    }
}

void TIM6_DAC_IRQHandler(void) {
    TIM6->SR &= ~TIM_SR_UIF;
    TIM6->CR1 &= ~TIM_CR1_CEN;
    //if the timer has counted up to 25ms then disable rattle check
    button_variables.rattle_check = CONTROL_RATTLE_OFF;
}

void TIM2_IRQHandler(void) {
    TIM2->SR &= ~TIM_SR_UIF;
    TIM2->CR1 &= ~TIM_CR1_CEN;
    //if the timer has counted up to 3s then set state long press
    if(button_variables.button_state == BUTTON_PRESSED) {
        button_variables.count_short_press = 0;// throw to zero in order to count amount of press
        program_state = SETUP_MODE;            // setup mode is set
        TIM14->CR1 |= TIM_CR1_CEN;             //turn on timer for blink led
    }
}
void TIM3_IRQHandler(void) {
    TIM3->SR &= ~TIM_SR_UIF;
    button_variables.press_check = CHECK_AMOUNT_OF_PRESS;
}

void check_button_press() {
    if(button_variables.press_check != CHECK_AMOUNT_OF_PRESS || button_variables.count_short_press == 0)
        return;
    button_variables.press_check = WAITING;
    //if it was long press, choose between two functions
    if(program_state == SETUP_MODE && button_variables.button_state == BUTTON_UNPRESSED) {
        switch(button_variables.count_short_press) {
        case 1:
            program_state = WRITE_DATA_INTO_FLASH;
            Turn_on_Led_PC8();                  //if it was 1 short press
            Turn_off_Led_PC9();                 //write data into flash
            Timers_turn_off();
            break;
        case 2:
            program_state = BLINK_LED_ON_OTHER_MK;  //if it was 2 short presses
            Turn_off_Led_PC8();                 //blink led on another MK
            Turn_off_Led_PC9();
            Timers_turn_off();
            break;
        default:
            Turn_off_Led_PC8();
            Turn_off_Led_PC9();
            Timers_turn_off();
            Button_reset_to_be_ready_for_work();    //if more than 2 press was performed -> restart button
            break;
        }
        return;
    }
```



```c
    //else check for short presses
    if(button_variables.button_state == BUTTON_UNPRESSED) {
        switch(button_variables.count_short_press) {
            case 1:
                program_state = TRANSFER_DATA;
                Turn_on_Led_PC8();                  //if it was single short press
                Turn_on_Led_PC9();                  //transfer data
                Timers_turn_off();
                break;
            case 2:
                program_state = RECEIVE_DATA_FROM_PC;//if it were 2 short presses
                Turn_on_Led_PC8();                   //receive data from PC via USART
                Timers_turn_off();
                break;
            case 3:
                program_state = TRANSFER_DATA_TO_PC;//if it were 3 short presses
                Turn_on_Led_PC9();                  //transfer data to PC via USART
                Timers_turn_off();
                break;
            default:
                Timers_turn_off();
                Button_reset_to_be_ready_for_work();//if more than 3 press was performed -> restart button
                break;
        }
    }
}

void TIM14_IRQHandler(void) {
    if( (TIM14->DIER & TIM_DIER_UIE) == TIM_DIER_UIE ) {
            TIM14->SR &= ~TIM_SR_UIF;          //timer for blink led each 0.5s
            GPIOC->ODR ^= GPIO_ODR_8;          // in setup mode
            GPIOC->ODR ^= GPIO_ODR_9;
            if(program_state == BLINK_MY_LED)
                count_of_blink++;
    }
}
```



```c
void Button_tune() {
    init_GPIO_for_Button();
    init_TIM2_for_long_pressure_check();
    init_TIM3_for_check_result_of_press();
    init_TIM6_for_rattle_eliminating();
    init_TIM14_for_blinkLed();
    Button_reset_to_be_ready_for_work();
}

void Timers_turn_on() {
    TIM2->CNT = 0;
    TIM3->CNT = 0;
    TIM6->CNT = 0;
    TIM14->CNT = 0;
    //if SETUP mode had already set then no need to turn on timer2 for check long press
    if(program_state != SETUP_MODE)
        TIM2->CR1 |= TIM_CR1_CEN;
    TIM3->CR1 |= TIM_CR1_CEN;
    TIM6->CR1 |= TIM_CR1_CEN;
}

void Timers_turn_off() {
    TIM2->CR1 &= ~TIM_CR1_CEN;
    TIM3->CR1 &= ~TIM_CR1_CEN;
    TIM6->CR1 &= ~TIM_CR1_CEN;
    TIM14->CR1 &= ~TIM_CR1_CEN;
}

void Button_reset_to_be_ready_for_work() {
    button_variables.button_state = BUTTON_UNPRESSED;
    button_variables.rattle_check = CONTROL_RATTLE_OFF;
    button_variables.count_short_press = 0;
}
```

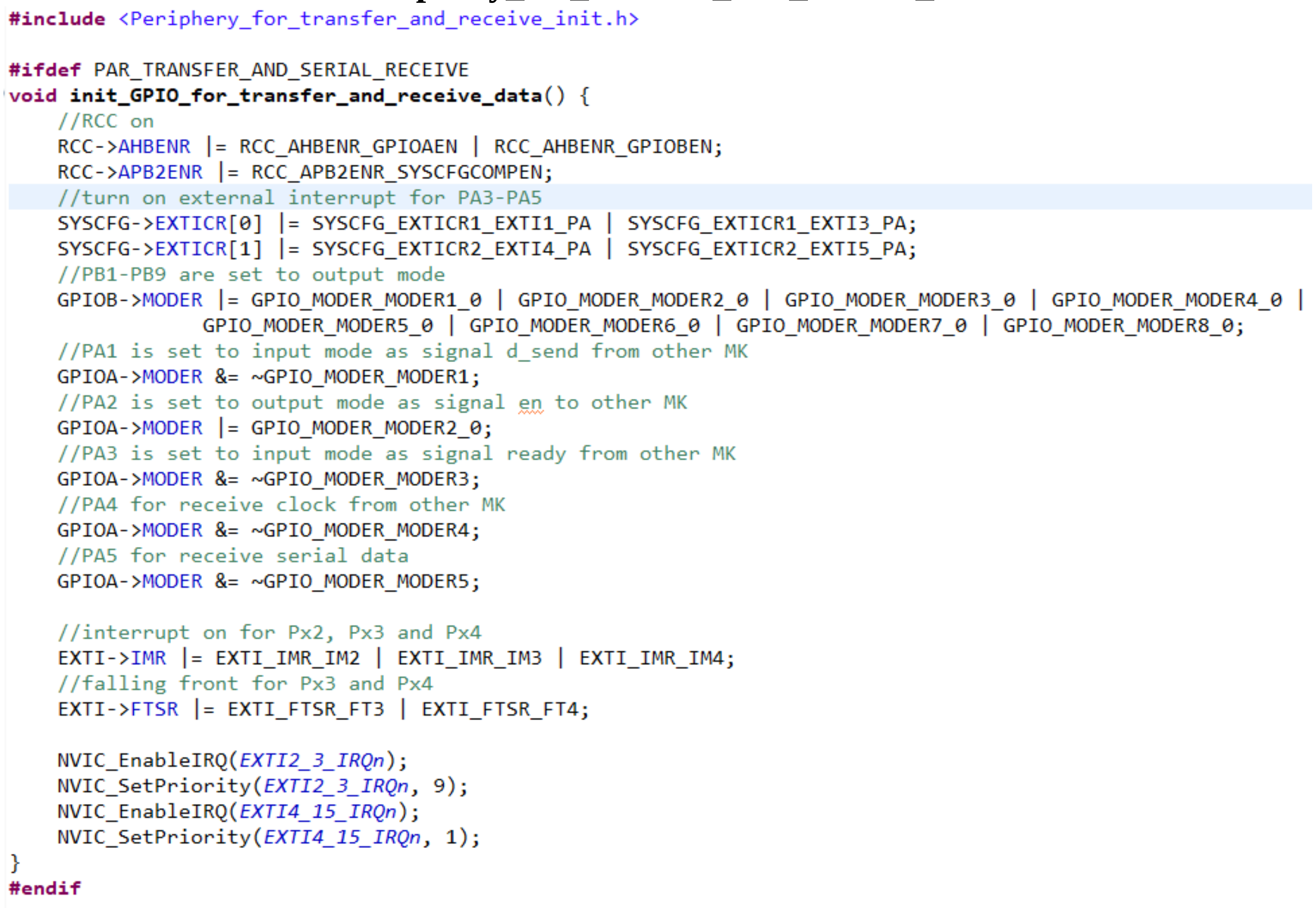
```
void Turn_on_Led_PC8() {
    GPIOC->ODR |= GPIO_ODR_8;
}

void Turn_off_Led_PC8() {
    GPIOC->ODR &= ~GPIO_ODR_8;
}

void Turn_on_Led_PC9(){
    GPIOC->ODR |= GPIO_ODR_9;
}

void Turn_off_Led_PC9(){
    GPIOC->ODR &= ~GPIO_ODR_9;
}
```

Periphery_for_transfer_and_receive_init.h

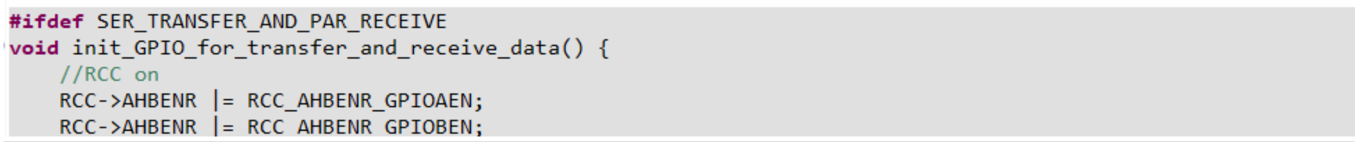
В данный файлах проинициализирована периферия для параллельной передачи и последовательного приёма.

```
#ifndef PERIPHERY_FOR_TRANSFER_AND_RECEIVE_INIT_H_
#define PERIPHERY_FOR_TRANSFER_AND_RECEIVE_INIT_H_

#include "stm32f0xx.h"

#define PAR_TRANSFER_AND_SERIAL_RECEIVE
//#define SER_TRANSFER_AND_PAR_RECEIVE


#ifdef PAR_TRANSFER_AND_SERIAL_RECEIVE
//PA1 - d_send, PA2 - enable, PA3 - ready, PA4 - clock, PA5 - serial data, PB1-PB9 - par data
void init_GPIO_for_transfer_and_receive_data();
#endif

#ifdef SER_TRANSFER_AND_PAR_RECEIVE
//PA1 - d_send, PA2 - enable, PA3 - ready, PA4 - clock, PA5 - serial data, PB1-PB9 - par data
void init_GPIO_for_transfer_and_receive_data();
//timer15 init for generate clock signal
void init_TIM15_for_clock_sig();
#endif

#endif /* PERIPHERY_FOR_TRANSFER_AND_RECEIVE_INIT_H_ */
```

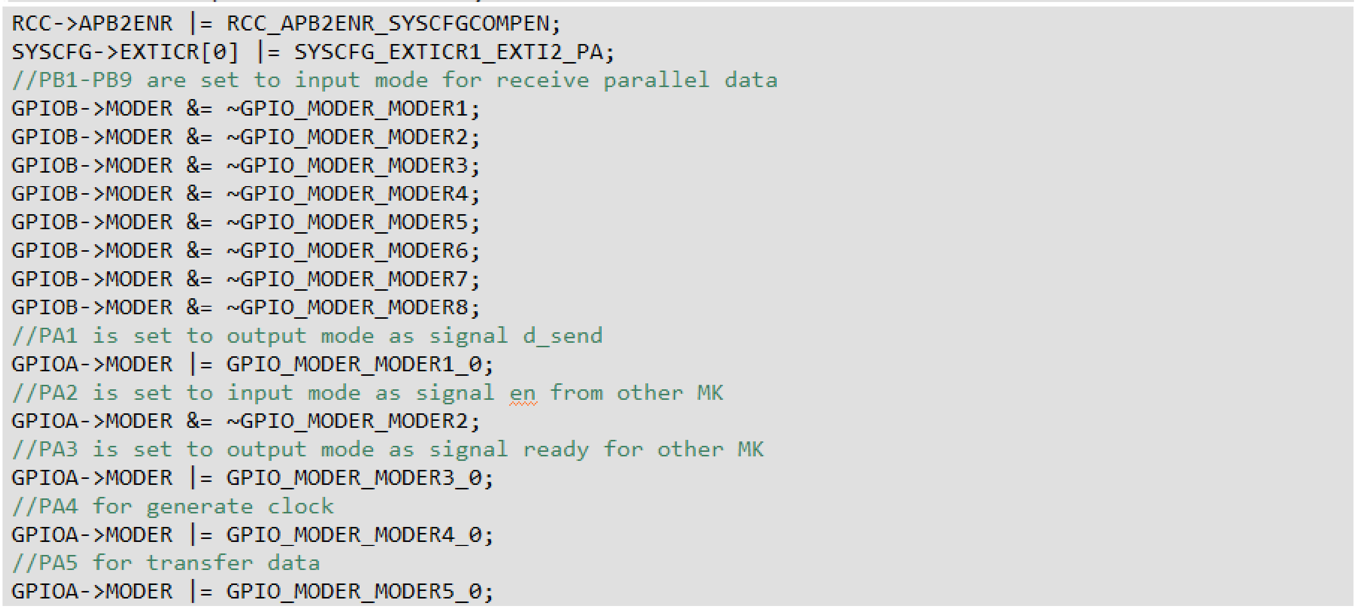
Periphery_for_transfer_and_receive_init.c



```c
#include <Periphery_for_transfer_and_receive_init.h>

#ifdef PAR_TRANSFER_AND_SERIAL_RECEIVE
void init_GPIO_for_transfer_and_receive_data() {
    //RCC on
    RCC->AHBENR |= RCC_AHBENR_GPIOAEN | RCC_AHBENR_GPIOBEN;
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGCOMPEN;
    //turn on external interrupt for PA3-PA5
    SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI1_PA | SYSCFG_EXTICR1_EXTI3_PA;
    SYSCFG->EXTICR[1] |= SYSCFG_EXTICR2_EXTI4_PA | SYSCFG_EXTICR2_EXTI5_PA;
    //PB1-PB9 are set to output mode
    GPIOB->MODER |= GPIO_MODER_MODER1_0 | GPIO_MODER_MODER2_0 | GPIO_MODER_MODER3_0 | GPIO_MODER_MODER4_0 |
            GPIO_MODER_MODER5_0 | GPIO_MODER_MODER6_0 | GPIO_MODER_MODER7_0 | GPIO_MODER_MODER8_0;
    //PA1 is set to input mode as signal d_send from other MK
    GPIOA->MODER &= ~GPIO_MODER_MODER1;
    //PA2 is set to output mode as signal en to other MK
    GPIOA->MODER |= GPIO_MODER_MODER2_0;
    //PA3 is set to input mode as signal ready from other MK
    GPIOA->MODER &= ~GPIO_MODER_MODER3;
    //PA4 for receive clock from other MK
    GPIOA->MODER &= ~GPIO_MODER_MODER4;
    //PA5 for receive serial data
    GPIOA->MODER &= ~GPIO_MODER_MODER5;

    //interrupt on for Px2, Px3 and Px4
    EXTI->IMR |= EXTI_IMR_IM2 | EXTI_IMR_IM3 | EXTI_IMR_IM4;
    //falling front for Px3 and Px4
    EXTI->FTSR |= EXTI_FTSR_FT3 | EXTI_FTSR_FT4;

    NVIC_EnableIRQ(EXTI2_3_IRQn);
    NVIC_SetPriority(EXTI2_3_IRQn, 9);
    NVIC_EnableIRQ(EXTI4_15_IRQn);
    NVIC_SetPriority(EXTI4_15_IRQn, 1);
}
#endif
```



```c
#ifdef SER_TRANSFER_AND_PAR_RECEIVE
void init_GPIO_for_transfer_and_receive_data() {
    //RCC on
    RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
    RCC->AHBENR |= RCC_AHBENR_GPIOBEN;
```



```c
RCC->APB2ENR |= RCC_APB2ENR_SYSCFGCOMPEN;
SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI2_PA;
//PB1-PB9 are set to input mode for receive parallel data
GPIOB->MODER &= ~GPIO_MODER_MODER1;
GPIOB->MODER &= ~GPIO_MODER_MODER2;
GPIOB->MODER &= ~GPIO_MODER_MODER3;
GPIOB->MODER &= ~GPIO_MODER_MODER4;
GPIOB->MODER &= ~GPIO_MODER_MODER5;
GPIOB->MODER &= ~GPIO_MODER_MODER6;
GPIOB->MODER &= ~GPIO_MODER_MODER7;
GPIOB->MODER &= ~GPIO_MODER_MODER8;
//PA1 is set to output mode as signal d_send
GPIOA->MODER |= GPIO_MODER_MODER1_0;
//PA2 is set to input mode as signal en from other MK
GPIOA->MODER &= ~GPIO_MODER_MODER2;
//PA3 is set to output mode as signal ready for other MK
GPIOA->MODER |= GPIO_MODER_MODER3_0;
//PA4 for generate clock
GPIOA->MODER |= GPIO_MODER_MODER4_0;
//PA5 for transfer data
GPIOA->MODER |= GPIO_MODER_MODER5_0;
```

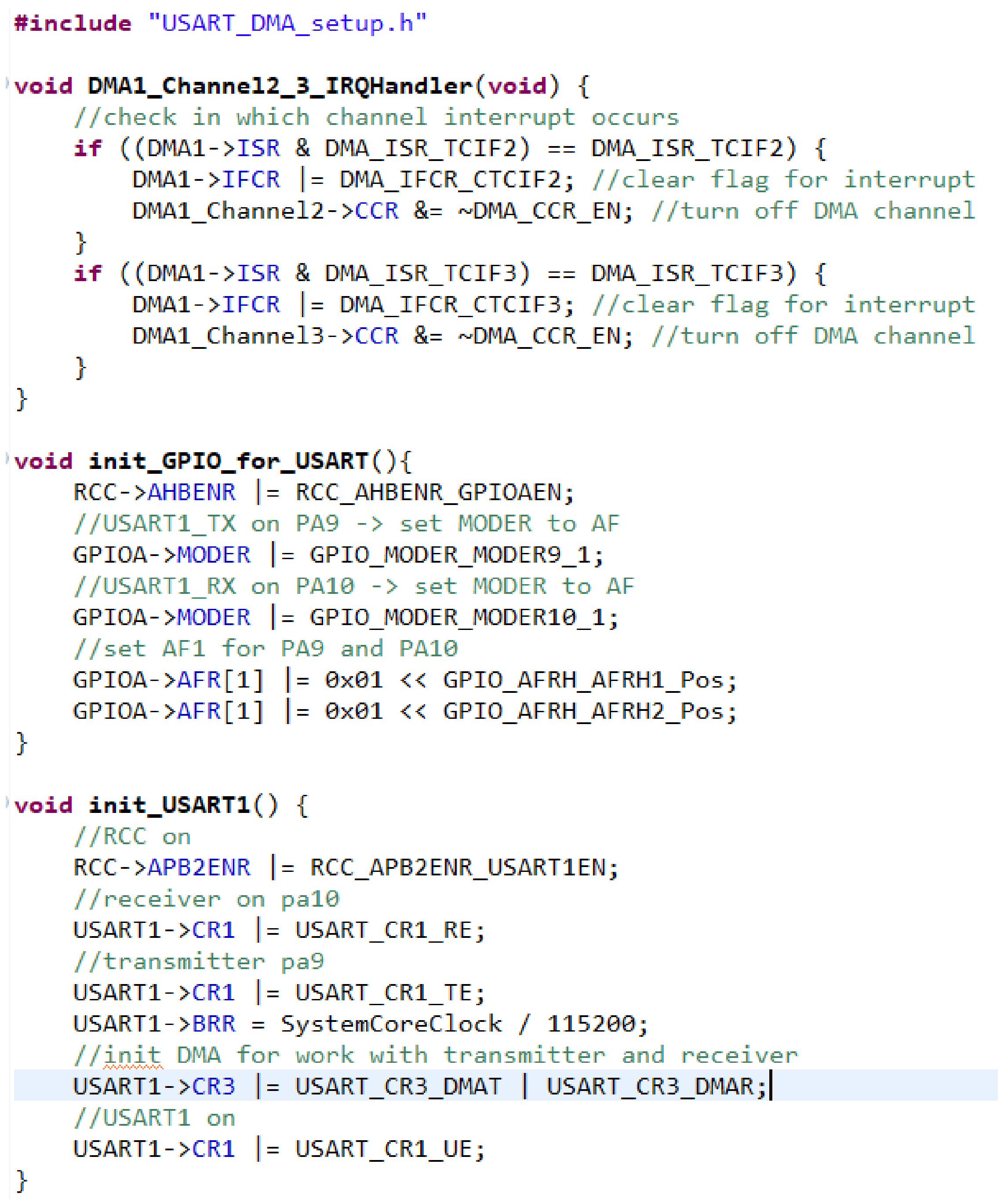
```
    //interrupt on for Px2 and Px4
    EXTI->IMR |= EXTI_IMR_IM2 | EXTI_IMR_IM4;
    //rising front in en sig for read data and clock sig to set data
    EXTI->RTSR |= EXTI_RTSR_RT2 | EXTI_RTSR_RT4;

    //signal d_send initially is set to "1"
    GPIOA->ODR |= GPIO_ODR_1;

    NVIC_EnableIRQ(EXTI4_15_IRQn);
    NVIC_SetPriority(EXTI4_15_IRQn, 1);
    NVIC_EnableIRQ(EXTI2_3_IRQn);
    NVIC_SetPriority(EXTI2_3_IRQn, 9);
}

void init_TIM15_for_clock_sig() {
    //RCC on
    RCC->APB2ENR |= RCC_APB2ENR_TIM15EN;
    //tune TIM14 for 1000 Hz
    TIM15->ARR = 700;
    TIM15->PSC = 4;
    //interrupt on
    TIM15->DIER |= TIM_DIER_UIE;
    NVIC_EnableIRQ(TIM15_IRQn);
    NVIC_SetPriority(TIM15_IRQn, 4);
}
#endif
```

USART_DMA_setup.h

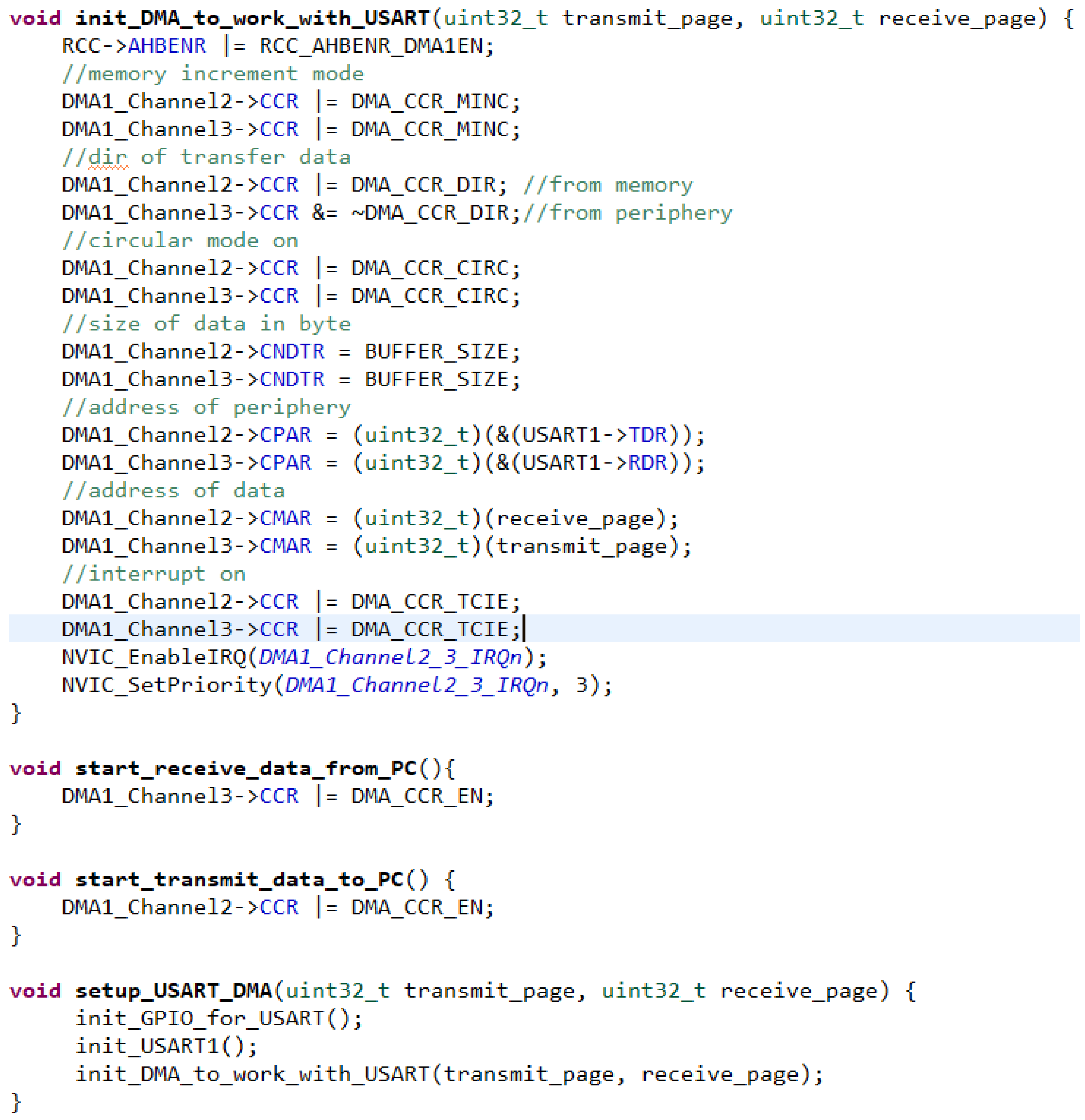
В данных файлах проинициализирована периферия для передачи данных на ПК и их получения с ПК.

```
#ifndef USART_DMA_SETUP_H_
#define USART_DMA_SETUP_H_

#include "stm32f0xx.h"
#include "buffer.h"


void DMA1_Channel2_3_IRQHandler(void);

//initialize GPIO for USART
void init_GPIO_for_USART();
//initialize USART1 for receive data from PC and transfer data to PC
void init_USART1();
//initialize DMA for work with USART and buffers
void init_DMA_to_work_with_USART(uint32_t transmit_page, uint32_t receive_page);
//contain all function above
void setup_USART_DMA(uint32_t transmit_page, uint32_t receive_page);
//enable DMA_channel3 for receive data from PC
void start_receive_data_from_PC();
//enable DMA_channel2 for transfer data to PC
void start_transmit_data_to_PC();

#endif /* USART_DMA_SETUP_H_ */
```

USART_DMA_setup.c



```c
#include "USART_DMA_setup.h"

void DMA1_Channel2_3_IRQHandler(void) {
    //check in which channel interrupt occurs
    if ((DMA1->ISR & DMA_ISR_TCIF2) == DMA_ISR_TCIF2) {
        DMA1->IFCR |= DMA_IFCR_CTCIF2; //clear flag for interrupt
        DMA1_Channel2->CCR &= ~DMA_CCR_EN; //turn off DMA channel
    }
    if ((DMA1->ISR & DMA_ISR_TCIF3) == DMA_ISR_TCIF3) {
        DMA1->IFCR |= DMA_IFCR_CTCIF3; //clear flag for interrupt
        DMA1_Channel3->CCR &= ~DMA_CCR_EN; //turn off DMA channel
    }
}

void init_GPIO_for_USART(){
    RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
    //USART1_TX on PA9 -> set MODER to AF
    GPIOA->MODER |= GPIO_MODER_MODER9_1;
    //USART1_RX on PA10 -> set MODER to AF
    GPIOA->MODER |= GPIO_MODER_MODER10_1;
    //set AF1 for PA9 and PA10
    GPIOA->AFR[1] |= 0x01 << GPIO_AFRH_AFRH1_Pos;
    GPIOA->AFR[1] |= 0x01 << GPIO_AFRH_AFRH2_Pos;
}

void init_USART1() {
    //RCC on
    RCC->APB2ENR |= RCC_APB2ENR_USART1EN;
    //receiver on pa10
    USART1->CR1 |= USART_CR1_RE;
    //transmitter pa9
    USART1->CR1 |= USART_CR1_TE;
    USART1->BRR = SystemCoreClock / 115200;
    //init DMA for work with transmitter and receiver
    USART1->CR3 |= USART_CR3_DMAT | USART_CR3_DMAR;
    //USART1 on
    USART1->CR1 |= USART_CR1_UE;
}
```



```c
void init_DMA_to_work_with_USART(uint32_t transmit_page, uint32_t receive_page) {
    RCC->AHBENR |= RCC_AHBENR_DMA1EN;
    //memory increment mode
    DMA1_Channel2->CCR |= DMA_CCR_MINC;
    DMA1_Channel3->CCR |= DMA_CCR_MINC;
    //dir of transfer data
    DMA1_Channel2->CCR |= DMA_CCR_DIR; //from memory
    DMA1_Channel3->CCR &= ~DMA_CCR_DIR;//from periphery
    //circular mode on
    DMA1_Channel2->CCR |= DMA_CCR_CIRC;
    DMA1_Channel3->CCR |= DMA_CCR_CIRC;
    //size of data in byte
    DMA1_Channel2->CNDTR = BUFFER_SIZE;
    DMA1_Channel3->CNDTR = BUFFER_SIZE;
    //address of periphery
    DMA1_Channel2->CPAR = (uint32_t)(&(USART1->TDR));
    DMA1_Channel3->CPAR = (uint32_t)(&(USART1->RDR));
    //address of data
    DMA1_Channel2->CMAR = (uint32_t)(receive_page);
    DMA1_Channel3->CMAR = (uint32_t)(transmit_page);
    //interrupt on
    DMA1_Channel2->CCR |= DMA_CCR_TCIE;
    DMA1_Channel3->CCR |= DMA_CCR_TCIE;
    NVIC_EnableIRQ(DMA1_Channel2_3_IRQn);
    NVIC_SetPriority(DMA1_Channel2_3_IRQn, 3);
}

void start_receive_data_from_PC(){
    DMA1_Channel3->CCR |= DMA_CCR_EN;
}

void start_transmit_data_to_PC() {
    DMA1_Channel2->CCR |= DMA_CCR_EN;
}

void setup_USART_DMA(uint32_t transmit_page, uint32_t receive_page) {
     init_GPIO_for_USART();
     init_USART1();
     init_DMA_to_work_with_USART(transmit_page, receive_page);
}
```

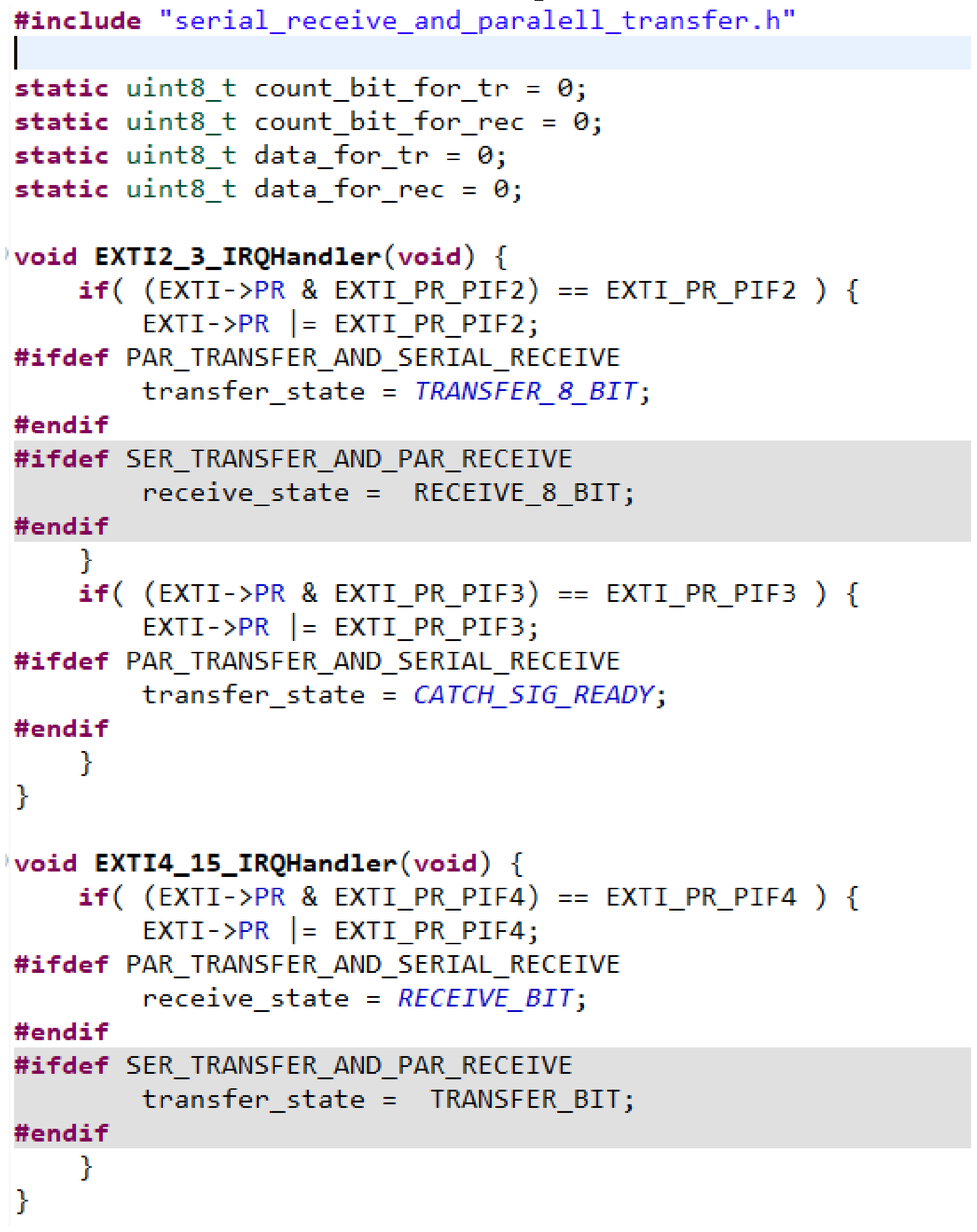
Serial_receive_and_parallel_transfer.h

В данных файлах расположены основные функции для передачи и получения данных, а также исполнения дополнительных функций, таких как мигание диодом на другом МК, получение данных с ПК, передача данных на ПК, запись данных во FLASH.

```c
#ifndef SERIAL_RECEIVE_AND_PARALELL_TRANSFER_H_
#define SERIAL_RECEIVE_AND_PARALELL_TRANSFER_H_

#define AMOUNT_RECEIVE_DATA 8
#include <Periphery_for_transfer_and_receive_init.h>
#include "button_cmd.h"
#include "USART_DMA_setup.h"

#ifdef SER_TRANSFER_AND_PAR_RECEIVE
//timer 15 for generate clock for serial data transfer
void TIM15_IRQHandler(void);
#endif
//falling front of clock generate interrupt in EXTI4 -> flag for read data from GPIO is set here
void EXTI4_15_IRQHandler(void);
//PA2 - signal en from this MK, PA3 - sig ready from other MK. In interrupt EXTI2 flag for transfer data is set
void EXTI2_3_IRQHandler(void);

//functions
//init buffer and write their to flash
void Setup_flash_with_data();
//initialize all peripheries needed to work of program
void Tune_peripheries();
//write data to flash using FLASH.h and FLASH.c
void write_to_flash();
//start program to depend upon which action is chosen
void manage_states();
//during receive data check if first 16 bit was key for blink led on my MK
void check_key_for_blink_led();
//if key is received then blink led 3 times
void blink_led_3_times();
//here actions for transfer 8 bit, receive bit and catch sig ready from other MK
void process_cmd();

#ifdef PAR_TRANSFER_AND_SERIAL_RECEIVE
//retrieve button to beginning state
void Restart_work_after_transfer();
//receive 1 bit from other MK
void receive_bit();
//perform actions if bytes of data are received
void check_byte_receive_complete();
//check if buf with received data is full
void check_receive_complete();
//transfer 8 bits to other MK
void set_data_on_GPIO();
#endif

#ifdef SER_TRANSFER_AND_PAR_RECEIVE
//retrieve button to beginning state
void Restart_work_after_transfer();
//transfer 1 bit on other MK
void transfer_bit();
```

```c
//perform actions if bytes of data are transmitted
void check_byte_transfer_complete();
//receive 8 bits from other MK
void receive_8bit();
//if byte are received -> write in buf ->start to receive new byte of data
void check_byte_receive_complete();
//if all of data are received -> restart work of button
void check_receive_complete();
//if button was pressed 2 times in setup mode -> transfer key for blink led on other mk
void blink_led_on_other_mk();
#endif


#endif /* SERIAL_RECEIVE_AND_PARALELL_TRANSFER_H_ */
```

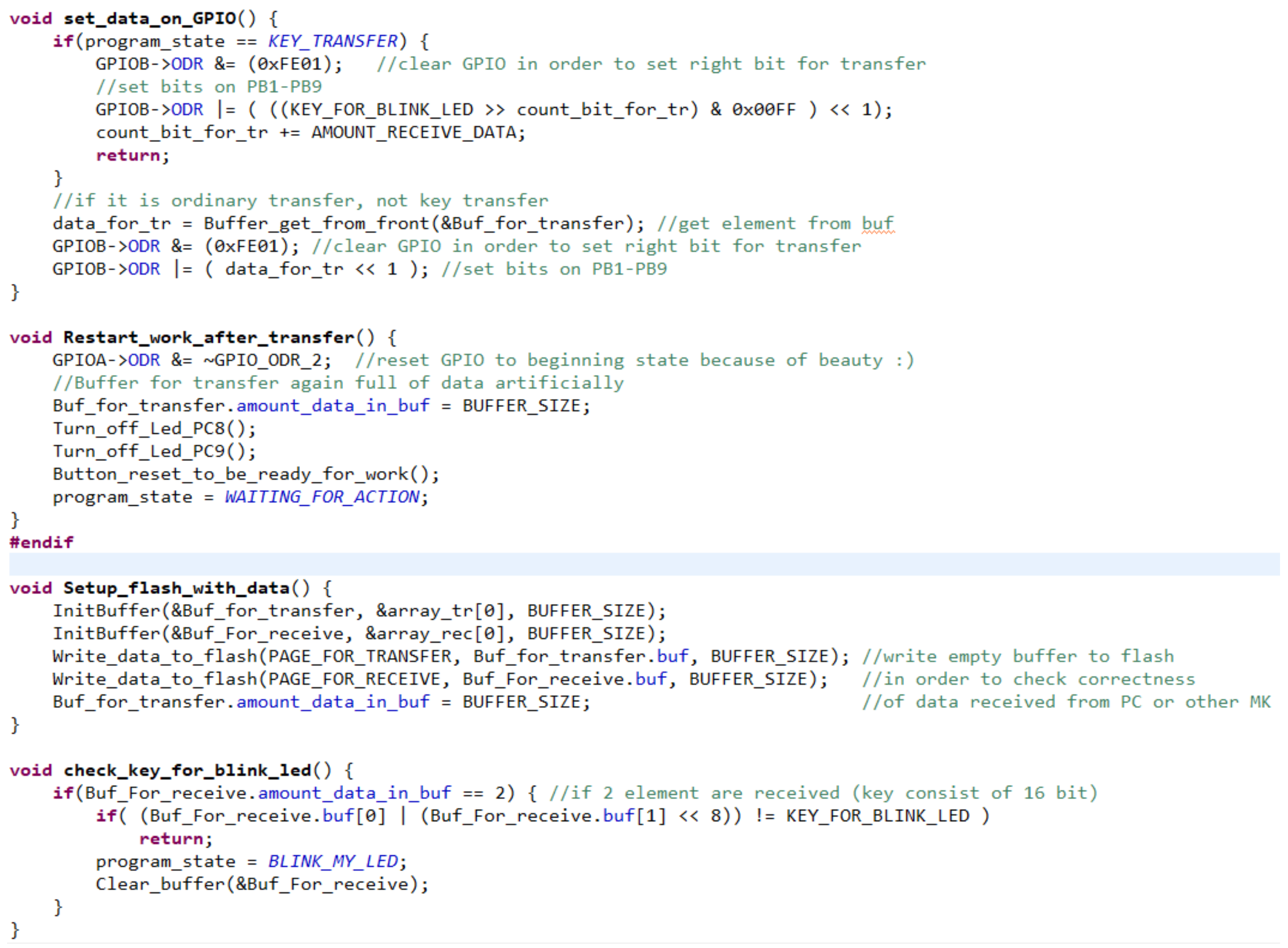
Serial_receive_and_parallel_transfer.c



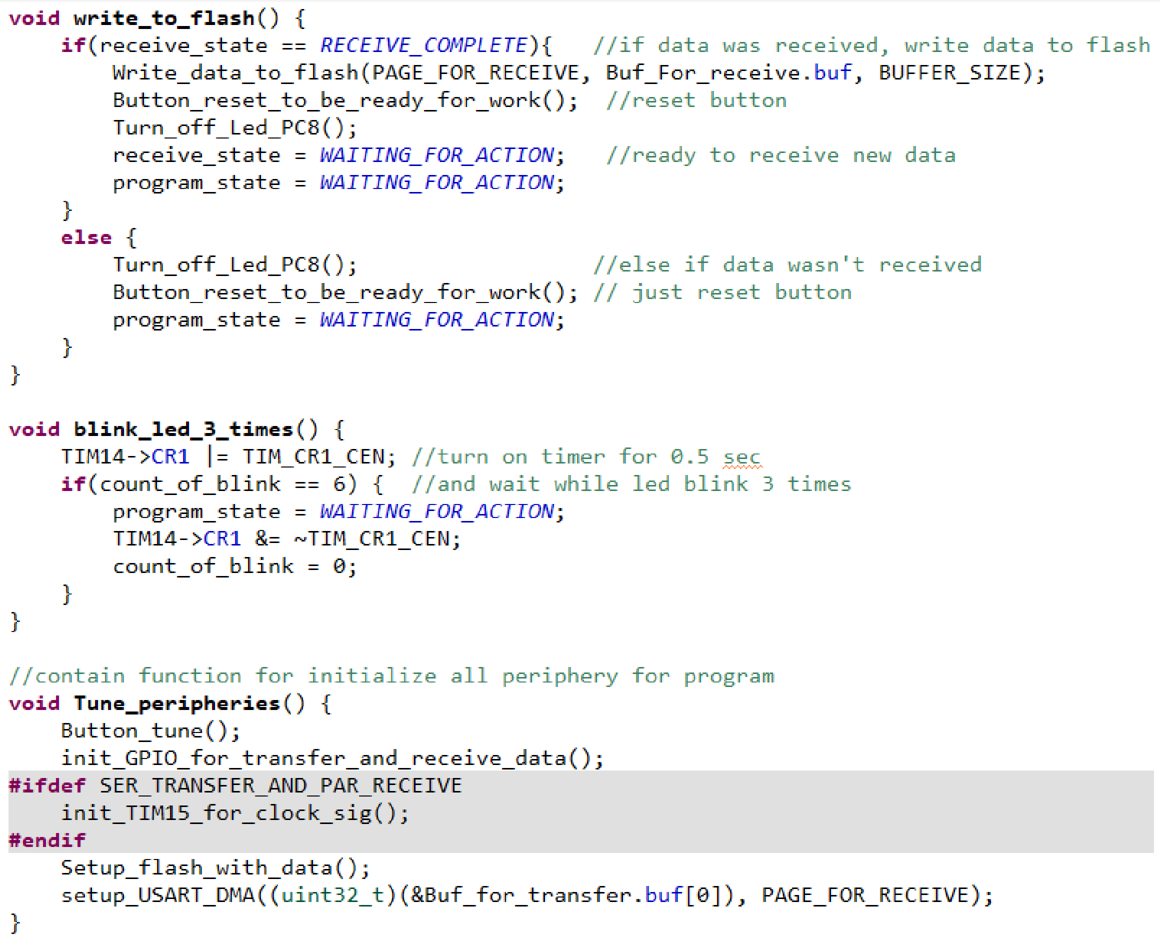
```c
#include "serial_receive_and_paralell_transfer.h"

static uint8_t count_bit_for_tr = 0;
static uint8_t count_bit_for_rec = 0;
static uint8_t data_for_tr = 0;
static uint8_t data_for_rec = 0;

void EXTI2_3_IRQHandler(void) {
    if( (EXTI->PR & EXTI_PR_PIF2) == EXTI_PR_PIF2 ) {
        EXTI->PR |= EXTI_PR_PIF2;
#ifdef PAR_TRANSFER_AND_SERIAL_RECEIVE
        transfer_state = TRANSFER_8_BIT;
#endif
#ifdef SER_TRANSFER_AND_PAR_RECEIVE
        receive_state =  RECEIVE_8_BIT;
#endif
    }
    if( (EXTI->PR & EXTI_PR_PIF3) == EXTI_PR_PIF3 ) {
        EXTI->PR |= EXTI_PR_PIF3;
#ifdef PAR_TRANSFER_AND_SERIAL_RECEIVE
        transfer_state = CATCH_SIG_READY;
#endif
    }
}

void EXTI4_15_IRQHandler(void) {
    if( (EXTI->PR & EXTI_PR_PIF4) == EXTI_PR_PIF4 ) {
        EXTI->PR |= EXTI_PR_PIF4;
#ifdef PAR_TRANSFER_AND_SERIAL_RECEIVE
        receive_state = RECEIVE_BIT;
#endif
#ifdef SER_TRANSFER_AND_PAR_RECEIVE
        transfer_state =  TRANSFER_BIT;
#endif
    }
}
```

```c
#ifdef SER_TRANSFER_AND_PAR_RECEIVE
void TIM15_IRQHandler(void) {    //in interrupts timer15 clock signal is generated
    if( (TIM15->DIER & TIM_DIER_UIE) == TIM_DIER_UIE ) {
        TIM15->SR &= ~TIM_SR_UIF;
        GPIOA->ODR ^= GPIO_ODR_4;//generate clock on PA4
    }
}

void Restart_work_after_transfer() {
    GPIOA->ODR &= ~GPIO_ODR_4;   //generate falling front for receive last bit in other MK
    for(int i = 0; i < 20; i++); //delay in order sig d_send have time to reach other MK
    TIM15->CR1 &= ~TIM_CR1_CEN;  //and then set d_send to "1"
    GPIOA->ODR |= GPIO_ODR_1;    //PA1 is set to 1(d_send signal) as transfer ended
     //Buf for transfer again full of data
    Buf_for_transfer.amount_data_in_buf = BUFFER_SIZE;
    Turn_off_Led_PC8();
    Turn_off_Led_PC9();
    Button_reset_to_be_ready_for_work();
    program_state = WAITING_FOR_ACTION;
}

void transfer_bit() {
    //if count_bit_for_tr = 0 then get byte of data from buffer
    if(count_bit_for_tr == 0)
        data_for_tr = Buffer_get_from_front(&Buf_for_transfer);
    GPIOA->ODR &= ~GPIO_ODR_5; //clear bit for data transfer
    //write 1 bit to PA5, for that get specific bit from byte and shift it to ODR5 (<< 5)
    GPIOA->ODR |= (( (data_for_tr >> count_bit_for_tr) & 0x0001 ) << 5);
    count_bit_for_tr++;
}

void check_byte_transfer_complete() {
    //if byte of data are transfered then throw to zero count_bit_for_tr in order to
    if(count_bit_for_tr == DATA_SIZE) {
        count_bit_for_tr = 0;             //transfer next byte of data
        //if it was last element in transfer buf then transfer complete
        if(Buffer_empty(&Buf_for_transfer))
            Restart_work_after_transfer();
    }
}
```

```
void receive_8bit() {
    data_for_rec |= ((GPIOB->IDR >> 1) &  0x00FF); //receive 8 bit from PB1-PB9
}

void check_byte_receive_complete() {
    //if 8 bit was received write data to buf and throw to zero variables for receive
    Buffer_add_to_end(&Buf_For_receive, data_for_rec);
    count_bit_for_rec = 0;
    data_for_rec = 0;
    check_key_for_blink_led();  //check if first 16 bit was key to blink led
}

void check_receive_complete() {
    if(Buffer_is_full(&Buf_For_receive)) {
        receive_state = RECEIVE_COMPLETE; //if buf for receive is full then receive complete
        Clear_buffer(&Buf_For_receive);   //artificially buf again is empty
    }
}

void blink_led_on_other_mk() {
    GPIOA->ODR &= ~GPIO_ODR_5; //clear bit for data transfer
     //write 1 bit to PA5, for that get specific bit from byte and shift it to ODR5 (<< 5)
    GPIOA->ODR |= (( (KEY_FOR_BLINK_LED >> count_bit_for_tr) & 0x0001 ) << 5);
    count_bit_for_tr++;
    if(count_bit_for_tr == (DATA_SIZE * 2) ) {
        count_bit_for_tr = 0;
        Restart_work_after_transfer();
    }
}

#endif


#ifdef PAR_TRANSFER_AND_SERIAL_RECEIVE
void receive_bit() {
    //get bit from GPIOA->IDR PA5 and write in temp variable
    data_for_rec |= (((GPIOA->IDR >> 5) &  0x0001 ) << count_bit_for_rec);
    count_bit_for_rec++; //increase the number of bits have already received
}

void check_byte_receive_complete() {
    if(count_bit_for_rec == DATA_SIZE) {
        Buffer_add_to_end(&Buf_For_receive, data_for_rec); //if 8 bit are received
        data_for_rec = 0;                                  //then start to receive new bits
        count_bit_for_rec = 0;
        check_key_for_blink_led(); //check if first 16 bit was key to blink led
    }
}

void check_receive_complete() {
    if(Buffer_is_full(&Buf_For_receive)) {  //if buffer is full then all data are received
            receive_state = RECEIVE_COMPLETE;
            Clear_buffer(&Buf_For_receive); //artificially buf again is empty
    }
}
```



```c
void set_data_on_GPIO() {
    if(program_state == KEY_TRANSFER) {
        GPIOB->ODR &= (0xFE01);   //clear GPIO in order to set right bit for transfer
        //set bits on PB1-PB9
        GPIOB->ODR |= ( ((KEY_FOR_BLINK_LED >> count_bit_for_tr) & 0x00FF ) << 1);
        count_bit_for_tr += AMOUNT_RECEIVE_DATA;
        return;
    }
    //if it is ordinary transfer, not key transfer
    data_for_tr = Buffer_get_from_front(&Buf_for_transfer); //get element from buf
    GPIOB->ODR &= (0xFE01); //clear GPIO in order to set right bit for transfer
    GPIOB->ODR |= ( data_for_tr << 1 ); //set bits on PB1-PB9
}

void Restart_work_after_transfer() {
    GPIOA->ODR &= ~GPIO_ODR_2;  //reset GPIO to beginning state because of beauty :)
    //Buffer for transfer again full of data artificially
    Buf_for_transfer.amount_data_in_buf = BUFFER_SIZE;
    Turn_off_Led_PC8();
    Turn_off_Led_PC9();
    Button_reset_to_be_ready_for_work();
    program_state = WAITING_FOR_ACTION;
}
#endif

void Setup_flash_with_data() {
    InitBuffer(&Buf_for_transfer, &array_tr[0], BUFFER_SIZE);
    InitBuffer(&Buf_For_receive, &array_rec[0], BUFFER_SIZE);
    Write_data_to_flash(PAGE_FOR_TRANSFER, Buf_for_transfer.buf, BUFFER_SIZE); //write empty buffer to flash
    Write_data_to_flash(PAGE_FOR_RECEIVE, Buf_For_receive.buf, BUFFER_SIZE);   //in order to check correctness
    Buf_for_transfer.amount_data_in_buf = BUFFER_SIZE;                          //of data received from PC or other MK
}

void check_key_for_blink_led() {
    if(Buf_For_receive.amount_data_in_buf == 2) { //if 2 element are received (key consist of 16 bit)
        if( (Buf_For_receive.buf[0] | (Buf_For_receive.buf[1] << 8)) != KEY_FOR_BLINK_LED )
            return;
        program_state = BLINK_MY_LED;
        Clear_buffer(&Buf_For_receive);
    }
}
```

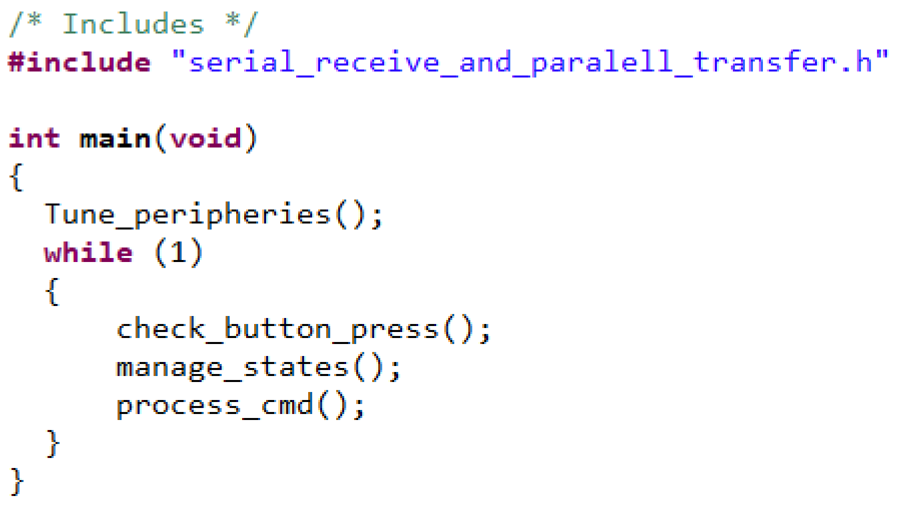


```c
void write_to_flash() {
    if(receive_state == RECEIVE_COMPLETE){    //if data was received, write data to flash
        Write_data_to_flash(PAGE_FOR_RECEIVE, Buf_For_receive.buf, BUFFER_SIZE);
        Button_reset_to_be_ready_for_work();  //reset button
        Turn_off_Led_PC8();
        receive_state = WAITING_FOR_ACTION;   //ready to receive new data
        program_state = WAITING_FOR_ACTION;
    }
    else {
        Turn_off_Led_PC8();                  //else if data wasn't received
        Button_reset_to_be_ready_for_work(); // just reset button
        program_state = WAITING_FOR_ACTION;
    }
}

void blink_led_3_times() {
    TIM14->CR1 |= TIM_CR1_CEN; //turn on timer for 0.5 sec
    if(count_of_blink == 6) {  //and wait while led blink 3 times
        program_state = WAITING_FOR_ACTION;
        TIM14->CR1 &= ~TIM_CR1_CEN;
        count_of_blink = 0;
    }
}

//contain function for initialize all periphery for program
void Tune_peripheries() {
    Button_tune();
    init_GPIO_for_transfer_and_receive_data();
#ifdef SER_TRANSFER_AND_PAR_RECEIVE
    init_TIM15_for_clock_sig();
#endif
    Setup_flash_with_data();
    setup_USART_DMA((uint32_t)(&Buf_for_transfer.buf[0]), PAGE_FOR_RECEIVE);
}
```

```c
void process_cmd() {
#ifdef PAR_TRANSFER_AND_SERIAL_RECEIVE
    if(receive_state == RECEIVE_BIT) {
        receive_state = WAITING_FOR_ACTION;          //change receive state
        if((GPIOA->IDR & GPIO_IDR_1) == GPIO_IDR_1) //if signal d_send "1" so data isn't transfered
            return;                                  //so return because it's fake :(
        receive_bit();                               //else if signal on PA1 (d_send) "0" -> data is transfered
        check_byte_receive_complete();               //so receive this data
        check_receive_complete();
    }

    if(transfer_state == TRANSFER_8_BIT) {       //change transfer state
        transfer_state = WAITING_FOR_ACTION;
        GPIOA->ODR &= ~GPIO_ODR_2;               //set 0 on PA2
        set_data_on_GPIO();                      //set 8 bit on GPIOx
        GPIOA->ODR |= GPIO_ODR_2;                //set 1 on PA2 -> rising front (signal enable) for PA2 other MK
    }

    if(transfer_state == CATCH_SIG_READY) {
        transfer_state = WAITING_FOR_ACTION;
        if( Buffer_empty(&Buf_for_transfer) ) {
            Restart_work_after_transfer(); //if buf empty -> all data are transfered
            return;                        //no need to SWI bit set so restart work and return
        }
        if(program_state == KEY_TRANSFER && count_bit_for_tr == (DATA_SIZE * 2)) {
            count_bit_for_tr = 0;          //if key to blink led on other MK is transfered ->
            Restart_work_after_transfer(); //no need to SWI bit set so restart work and return
            return;
        }
        EXTI->SWIER |= EXTI_SWIER_SWI2; //if falling front came then carry out new 8 bits transfer
    }
#endif

#ifdef SER_TRANSFER_AND_PAR_RECEIVE
    if(receive_state == RECEIVE_8_BIT) {
        receive_state = WAITING_FOR_ACTION;      //change receive state
        GPIOA->ODR |= GPIO_ODR_3;                //set 1 on PA3 (signal ready)
        receive_8bit();
        check_byte_receive_complete();
        check_receive_complete();
        GPIOA->ODR &= ~GPIO_ODR_3;               //set 0 on PA3 (signal ready) providing falling for PA3 other MK
    }
    if(transfer_state == TRANSFER_BIT) {
        transfer_state = WAITING_FOR_ACTION;     //change transfer state
        if(program_state == KEY_TRANSFER) {
            blink_led_on_other_mk();
            return;
        }
        transfer_bit();
        check_byte_transfer_complete();
    }
#endif
}
```

```c
//contain switch in which by using program state actions are chosen
void manage_states() {
    switch(program_state) {
        case TRANSFER_DATA:
            program_state = PROGRAMMING;
            ReadFromFlash(PAGE_FOR_TRANSFER, Buf_for_transfer.buf, BUFFER_SIZE);
#ifdef PAR_TRANSFER_AND_SERIAL_RECEIVE
            EXTI->SWIER |= EXTI_SWIER_SWI2; //call interrupt on EXTI2 and start transfer data
#endif
#ifdef SER_TRANSFER_AND_PAR_RECEIVE
            GPIOA->ODR &= ~GPIO_ODR_1;   //start transfer data so set PA1(d_send) to zero level
            for(int i = 0; i < 20; i++); //delay in order sig d_send have time to reach other MK
            TIM15->CR1 |= TIM_CR1_CEN;   //TIM15 on in order to generate clock
#endif
            break;
        case WRITE_DATA_INTO_FLASH:
            program_state = PROGRAMMING;
            write_to_flash();
            break;
        case BLINK_LED_ON_OTHER_MK:
            program_state = KEY_TRANSFER;
#ifdef PAR_TRANSFER_AND_SERIAL_RECEIVE
            EXTI->SWIER |= EXTI_SWIER_SWI2;
#endif
#ifdef SER_TRANSFER_AND_PAR_RECEIVE
            GPIOA->ODR &= ~GPIO_ODR_1;    //start transfer data so set PA1(d_send) to zero level
            for(int i = 0; i < 20; i++);  //delay in order sig d_send have time to reach other MK
            TIM15->CR1 |= TIM_CR1_CEN;    //TIM15 on in order to generate clock
#endif
            break;
        case BLINK_MY_LED:
            blink_led_3_times();
            break;

        case RECEIVE_DATA_FROM_PC:
            start_receive_data_from_PC();
            while(DMA1_Channel3->CCR & DMA_CCR_EN);  //wait while data receive
            Write_data_to_flash(PAGE_FOR_TRANSFER, Buf_for_transfer.buf, BUFFER_SIZE); //write transfered data to flash
            Turn_off_Led_PC8();
            Button_reset_to_be_ready_for_work();     //reset button
            program_state = WAITING_FOR_ACTION;
            break;
        case TRANSFER_DATA_TO_PC:
            start_transmit_data_to_PC();
            while(DMA1_Channel2->CCR & DMA_CCR_EN);  //wait while data transfer
            Turn_off_Led_PC9();
            Button_reset_to_be_ready_for_work();     //reset button
            program_state = WAITING_FOR_ACTION;
            break;
        default:
            break;
    }
}
```

Main.c



```c
/* Includes */
#include "serial_receive_and_paralell_transfer.h"

int main(void)
{
  Tune_peripheries();
  while (1)
  {
      check_button_press();
      manage_states();
      process_cmd();
  }
}
```