Московский государственный технический университет им. Н.Э. Баумана

Факультет «Радиоэлектроника и лазерная техника (РЛ)»

Кафедра «Технология приборостроения (РЛ6)»

Рубежный контроль №2

по дисциплине «Цифровые устройства и микропроцессоры»

Выполнил ст. группы РЛ6-69

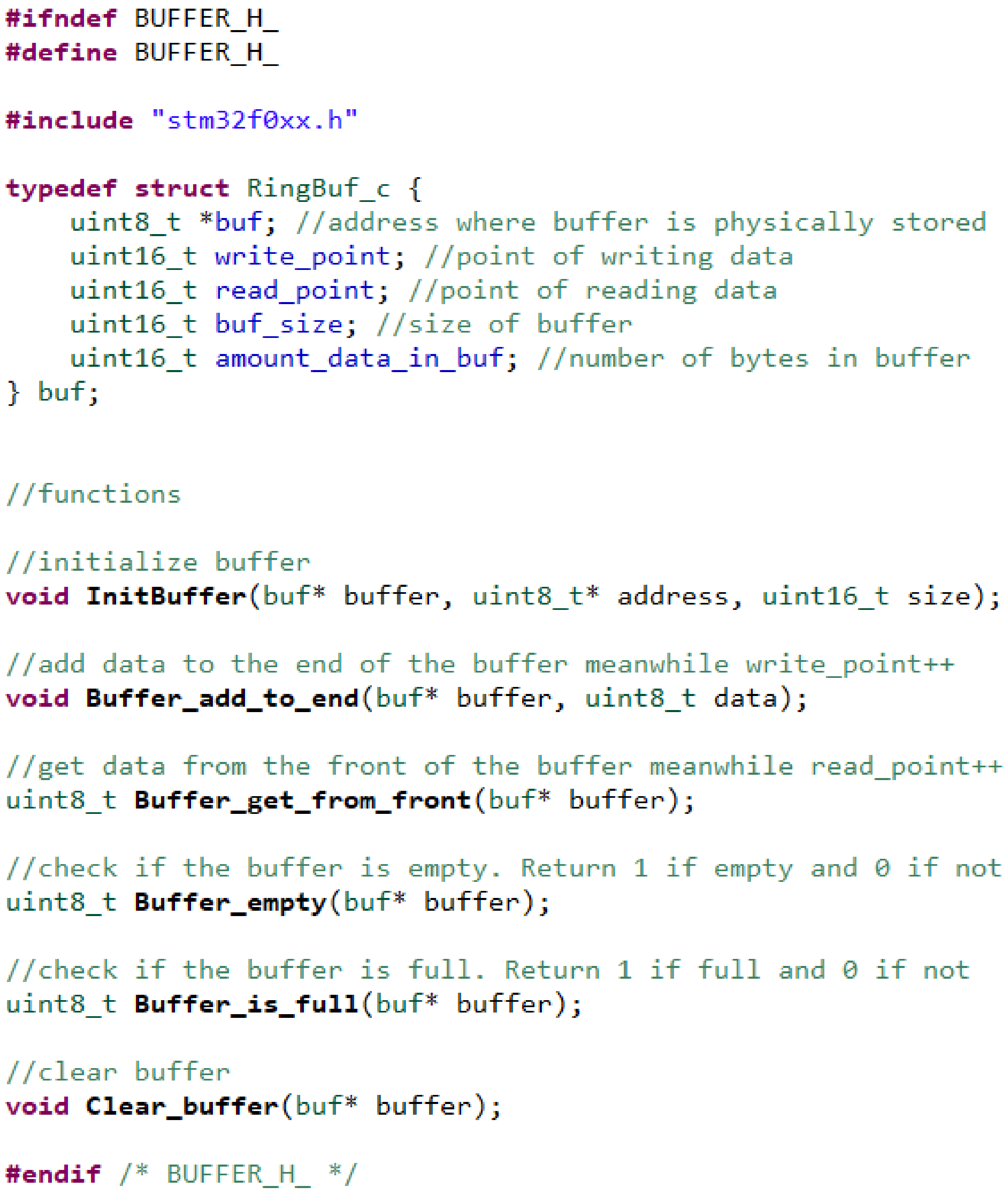
Лобанов Д.Д.

Преподаватель Семеренко Д.А.

Москва, 2023

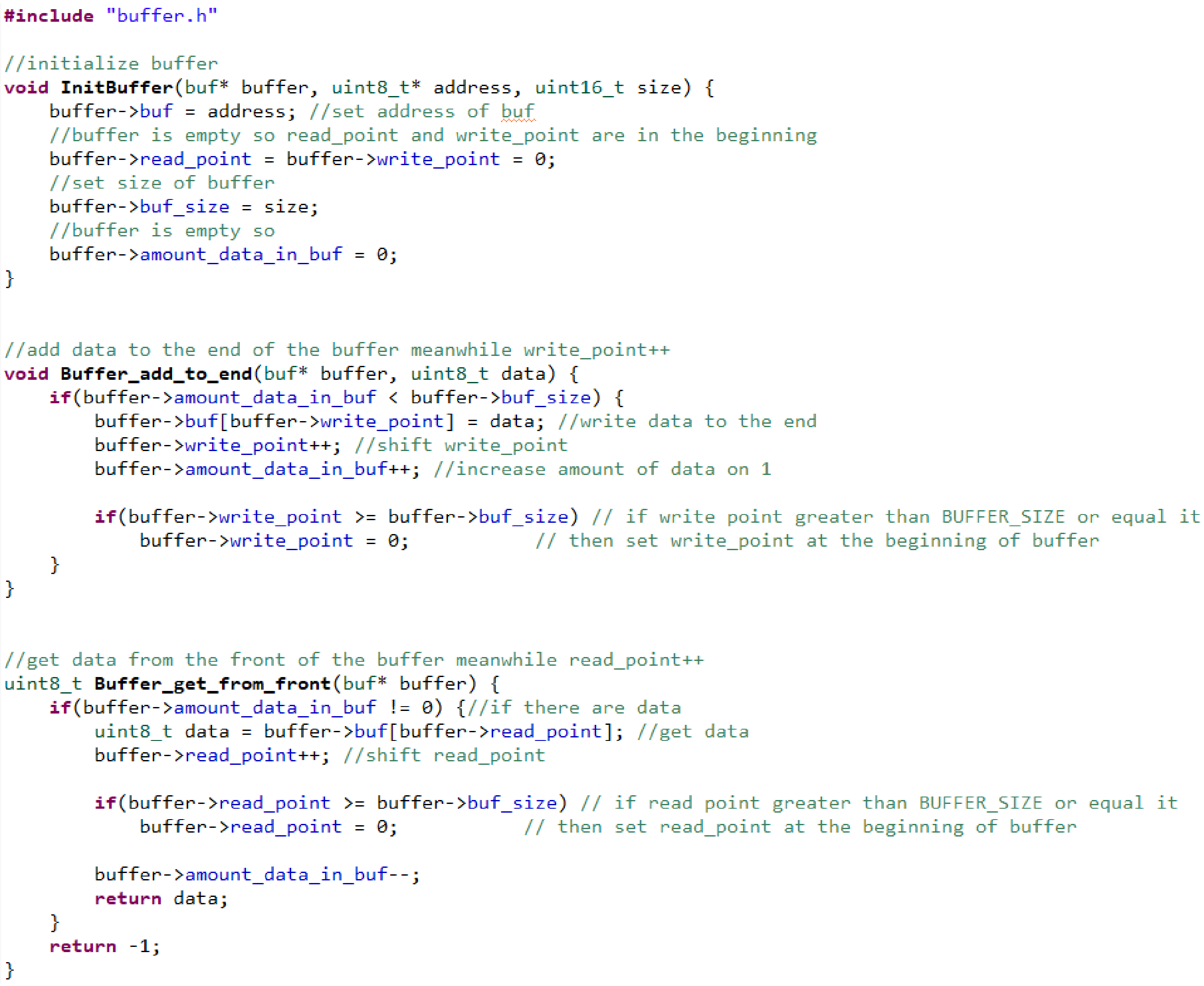
Оглавление

Buffer.h

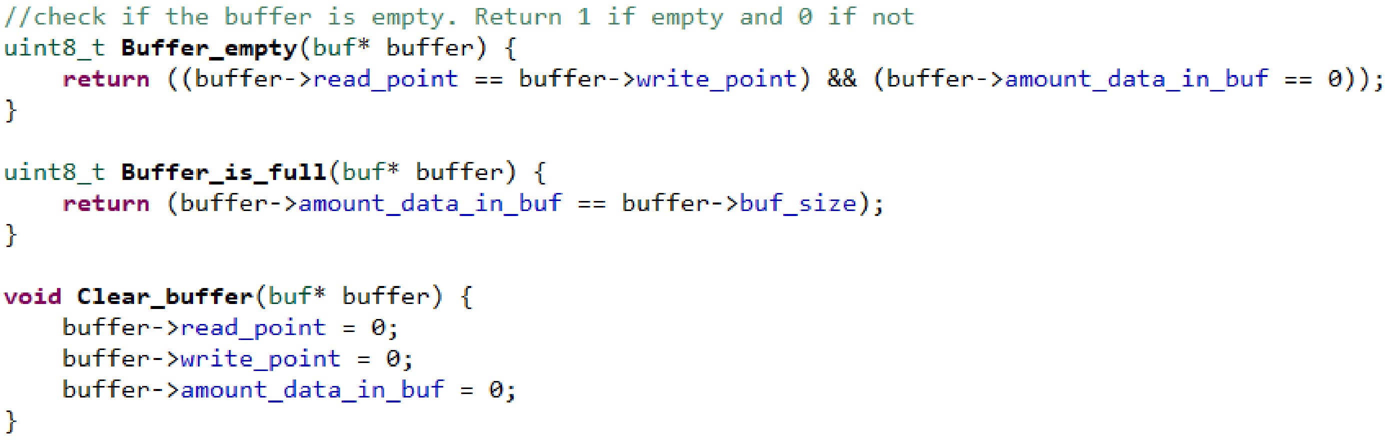


```
#ifndef BUFFER_H_
#define BUFFER_H_

#include "stm32f0xx.h"

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

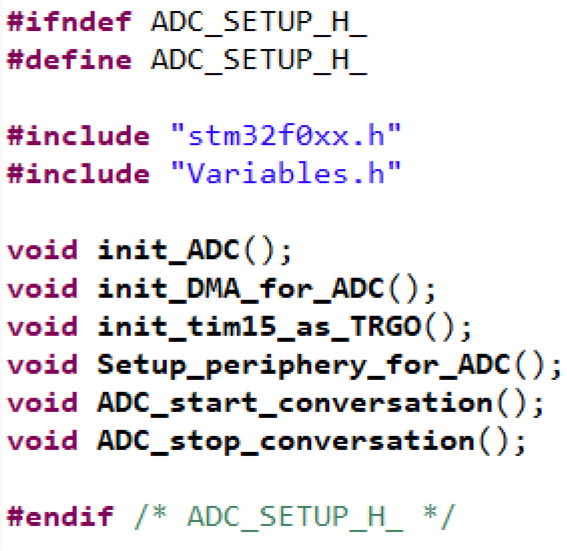


```c
#include "buffer.h"

//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
    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->buf_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->buf_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->buf_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;
}
```



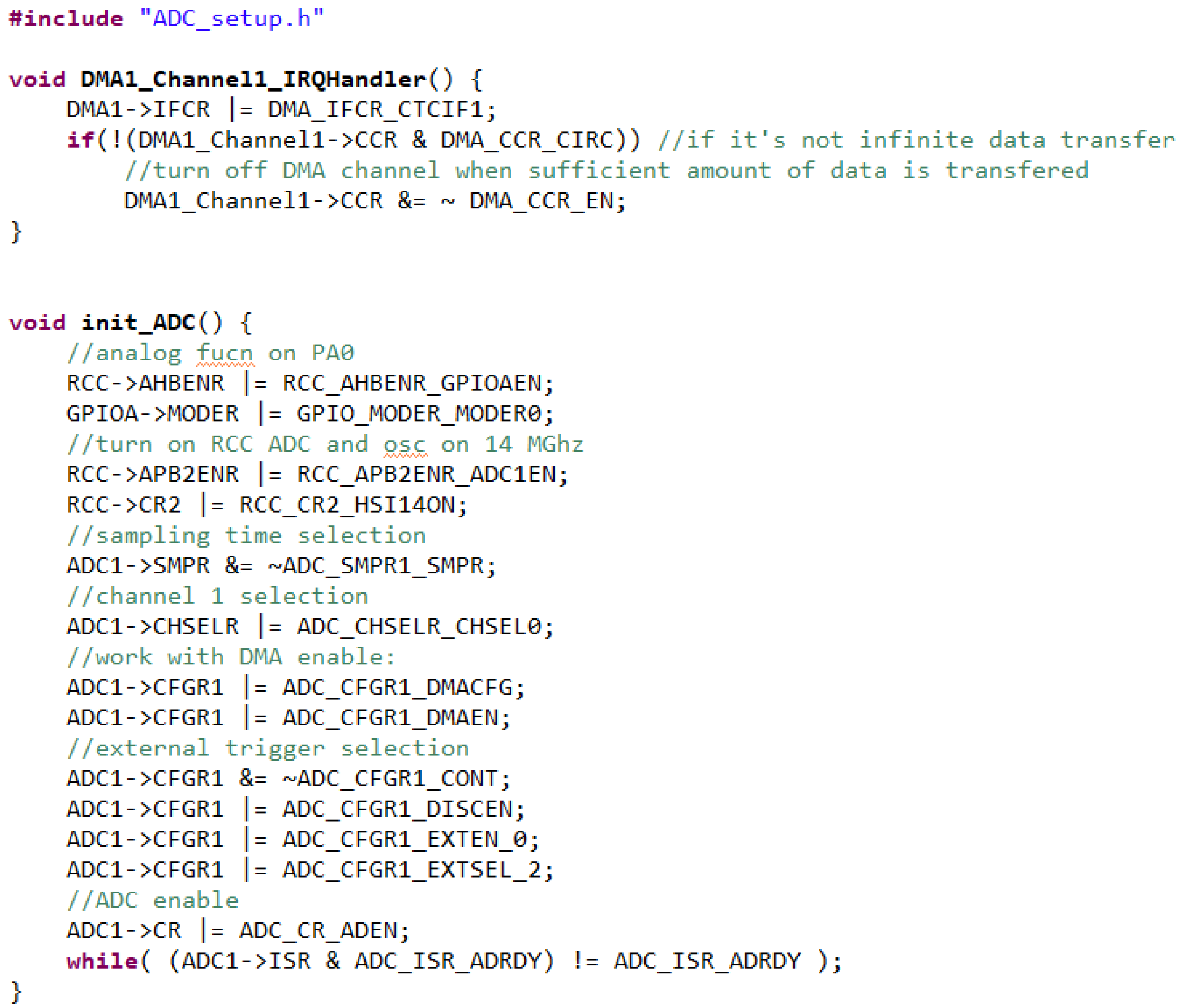
```c
//check if the buffer is empty. Return 1 if empty and 0 if not
uint8_t Buffer_empty(buf* buffer) {
    return ((buffer->read_point == buffer->write_point) && (buffer->amount_data_in_buf == 0));
}

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

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

ADC_setup.h

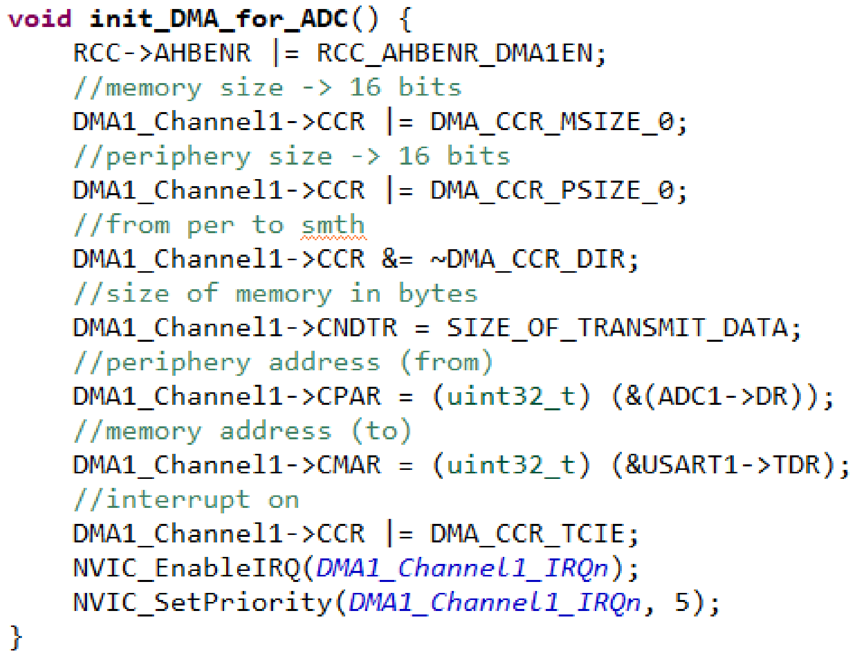


```
#ifndef ADC_SETUP_H_
#define ADC_SETUP_H_

#include "stm32f0xx.h"
#include "Variables.h"

void init_ADC();
void init_DMA_for_ADC();
void init_tim15_as_TRGO();
void Setup_periphery_for_ADC();
void ADC_start_conversation();
void ADC_stop_conversation();

#endif /* ADC_SETUP_H_ */
```

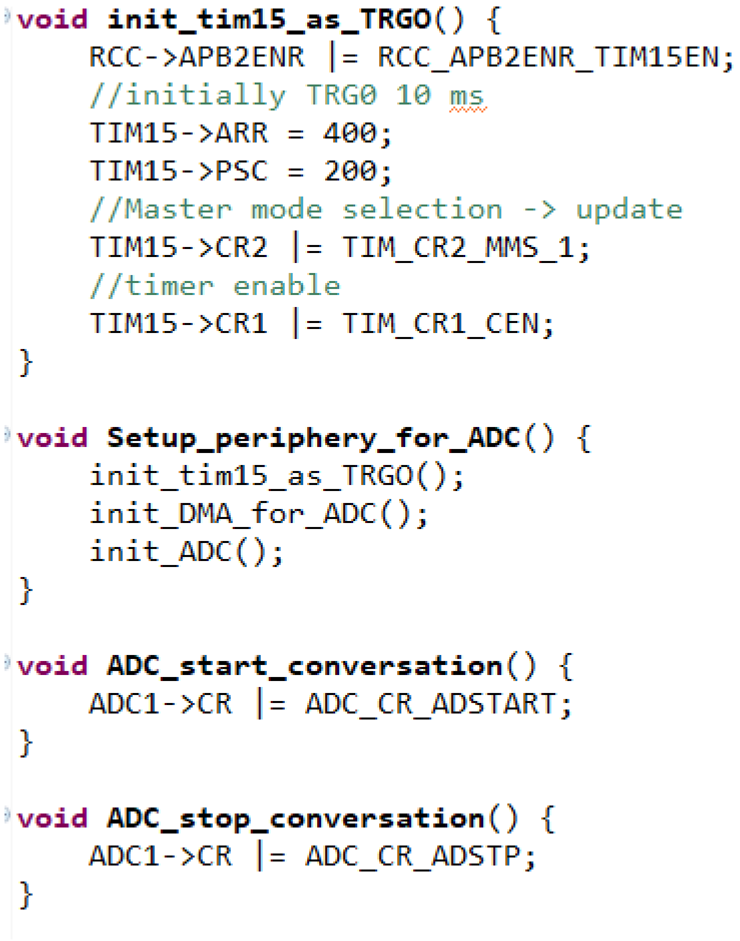
ADC_setup.c



```
#include "ADC_setup.h"

void DMA1_Channel1_IRQHandler() {
    DMA1->IFCR |= DMA_IFCR_CTCIF1;
    if(!(DMA1_Channel1->CCR & DMA_CCR_CIRC)) //if it's not infinite data transfer
        //turn off DMA channel when sufficient amount of data is transfered
        DMA1_Channel1->CCR &= ~ DMA_CCR_EN;
}


void init_ADC() {
    //analog fucn on PA0
    RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
    GPIOA->MODER |= GPIO_MODER_MODER0;
    //turn on RCC ADC and osc on 14 MGhz
    RCC->APB2ENR |= RCC_APB2ENR_ADC1EN;
    RCC->CR2 |= RCC_CR2_HSI14ON;
    //sampling time selection
    ADC1->SMPR &= ~ADC_SMPR1_SMPR;
    //channel 1 selection
    ADC1->CHSELR |= ADC_CHSELR_CHSEL0;
    //work with DMA enable:
    ADC1->CFGR1 |= ADC_CFGR1_DMACFG;
    ADC1->CFGR1 |= ADC_CFGR1_DMAEN;
    //external trigger selection
    ADC1->CFGR1 &= ~ADC_CFGR1_CONT;
    ADC1->CFGR1 |= ADC_CFGR1_DISCEN;
    ADC1->CFGR1 |= ADC_CFGR1_EXTEN_0;
    ADC1->CFGR1 |= ADC_CFGR1_EXTSEL_2;
    //ADC enable
    ADC1->CR |= ADC_CR_ADEN;
    while( (ADC1->ISR & ADC_ISR_ADRDY) != ADC_ISR_ADRDY );
}
```

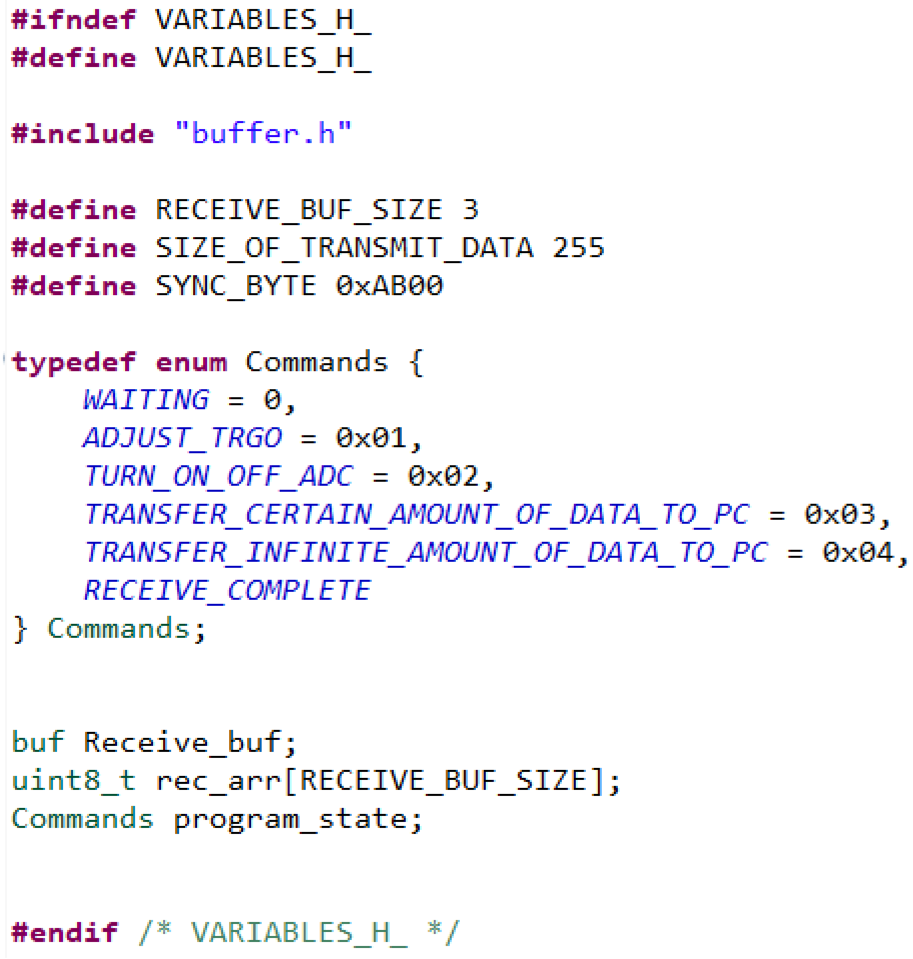


```
void init_DMA_for_ADC() {
    RCC->AHBENR |= RCC_AHBENR_DMA1EN;
    //memory size -> 16 bits
    DMA1_Channel1->CCR |= DMA_CCR_MSIZE_0;
    //periphery size -> 16 bits
    DMA1_Channel1->CCR |= DMA_CCR_PSIZE_0;
    //from per to smth
    DMA1_Channel1->CCR &= ~DMA_CCR_DIR;
    //size of memory in bytes
    DMA1_Channel1->CNDTR = SIZE_OF_TRANSMIT_DATA;
    //periphery address (from)
    DMA1_Channel1->CPAR = (uint32_t) (&(ADC1->DR));
    //memory address (to)
    DMA1_Channel1->CMAR = (uint32_t) (&USART1->TDR);
    //interrupt on
    DMA1_Channel1->CCR |= DMA_CCR_TCIE;
    NVIC_EnableIRQ(DMA1_Channel1_IRQn);
    NVIC_SetPriority(DMA1_Channel1_IRQn, 5);
}
```



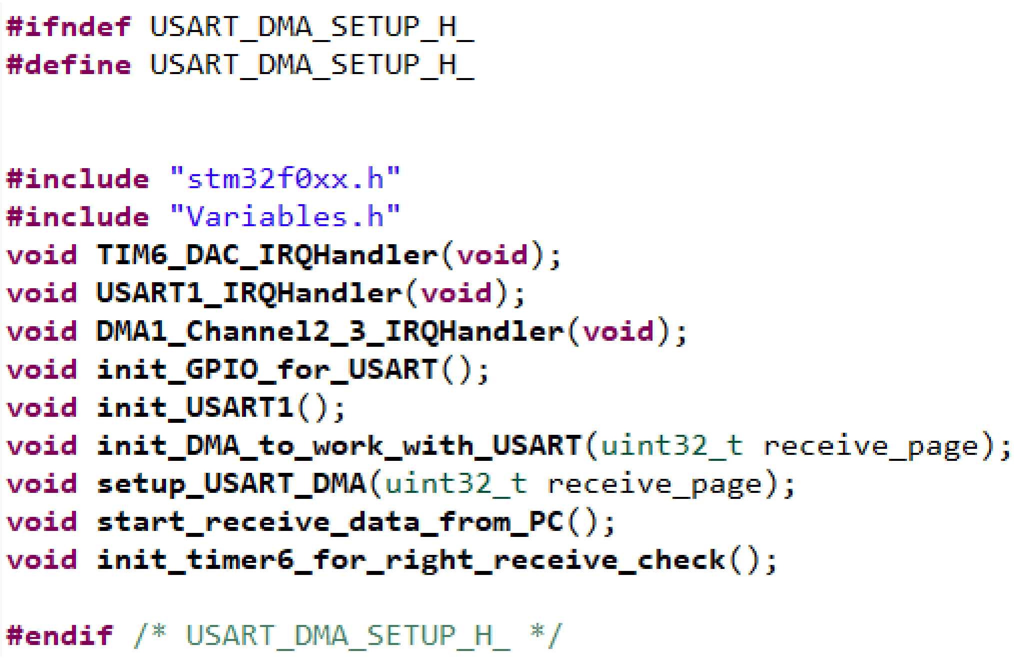
```c
void init_tim15_as_TRGO() {
    RCC->APB2ENR |= RCC_APB2ENR_TIM15EN;
    //initially TRG0 10 ms
    TIM15->ARR = 400;
    TIM15->PSC = 200;
    //Master mode selection -> update
    TIM15->CR2 |= TIM_CR2_MMS_1;
    //timer enable
    TIM15->CR1 |= TIM_CR1_CEN;
}

void Setup_periphery_for_ADC() {
    init_tim15_as_TRGO();
    init_DMA_for_ADC();
    init_ADC();
}

void ADC_start_conversation() {
    ADC1->CR |= ADC_CR_ADSTART;
}

void ADC_stop_conversation() {
    ADC1->CR |= ADC_CR_ADSTP;
}
```

Variables.h



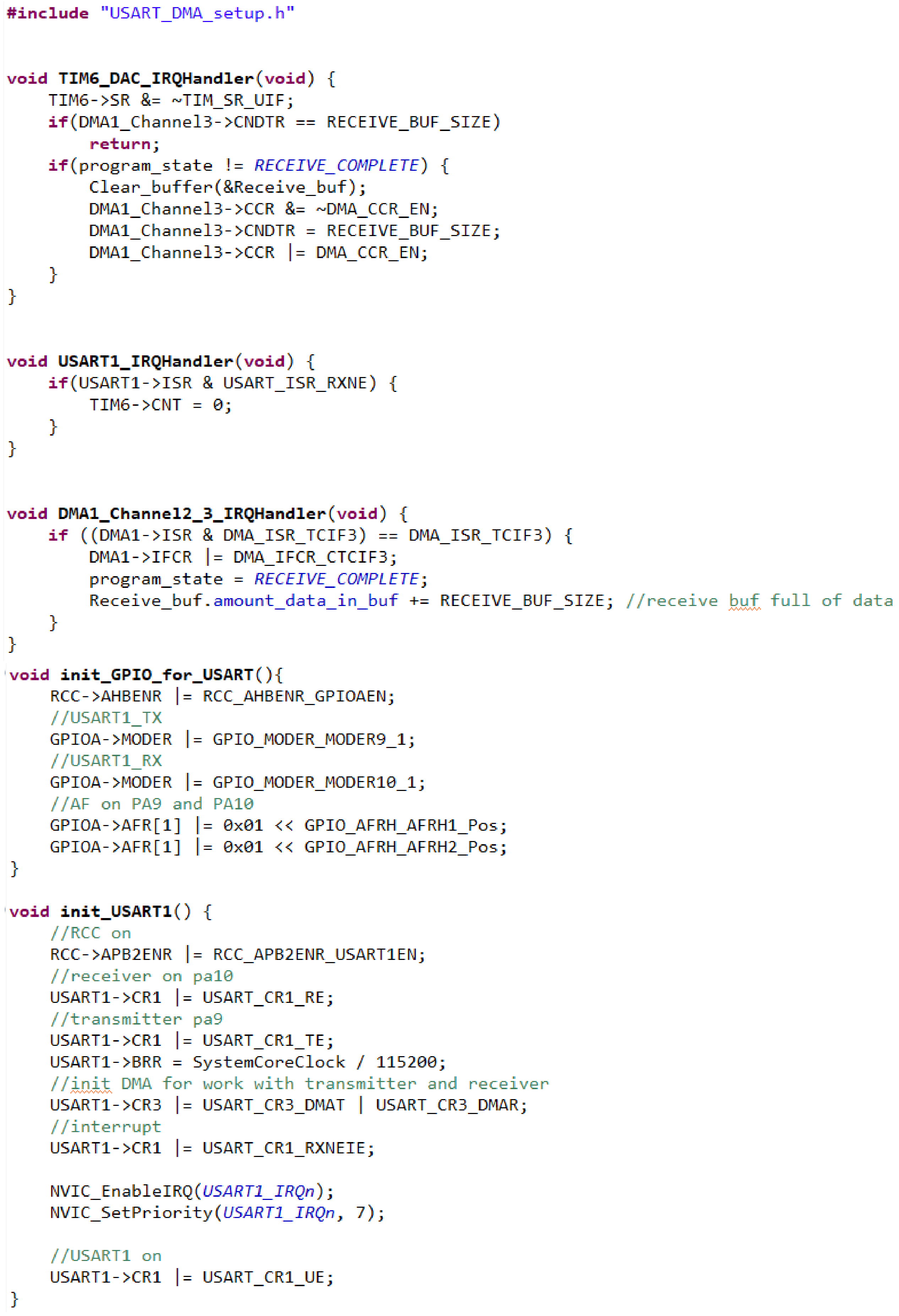
```
#ifndef VARIABLES_H_
#define VARIABLES_H_

#include "buffer.h"

#define RECEIVE_BUF_SIZE 3
#define SIZE_OF_TRANSMIT_DATA 255
#define SYNC_BYTE 0xAB00

typedef enum Commands {
    WAITING = 0,
    ADJUST_TRGO = 0x01,
    TURN_ON_OFF_ADC = 0x02,
    TRANSFER_CERTAIN_AMOUNT_OF_DATA_TO_PC = 0x03,
    TRANSFER_INFINITE_AMOUNT_OF_DATA_TO_PC = 0x04,
    RECEIVE_COMPLETE
} Commands;


buf Receive_buf;
uint8_t rec_arr[RECEIVE_BUF_SIZE];
Commands program_state;


#endif /* VARIABLES_H_ */
```

USART_DMA_setup.h

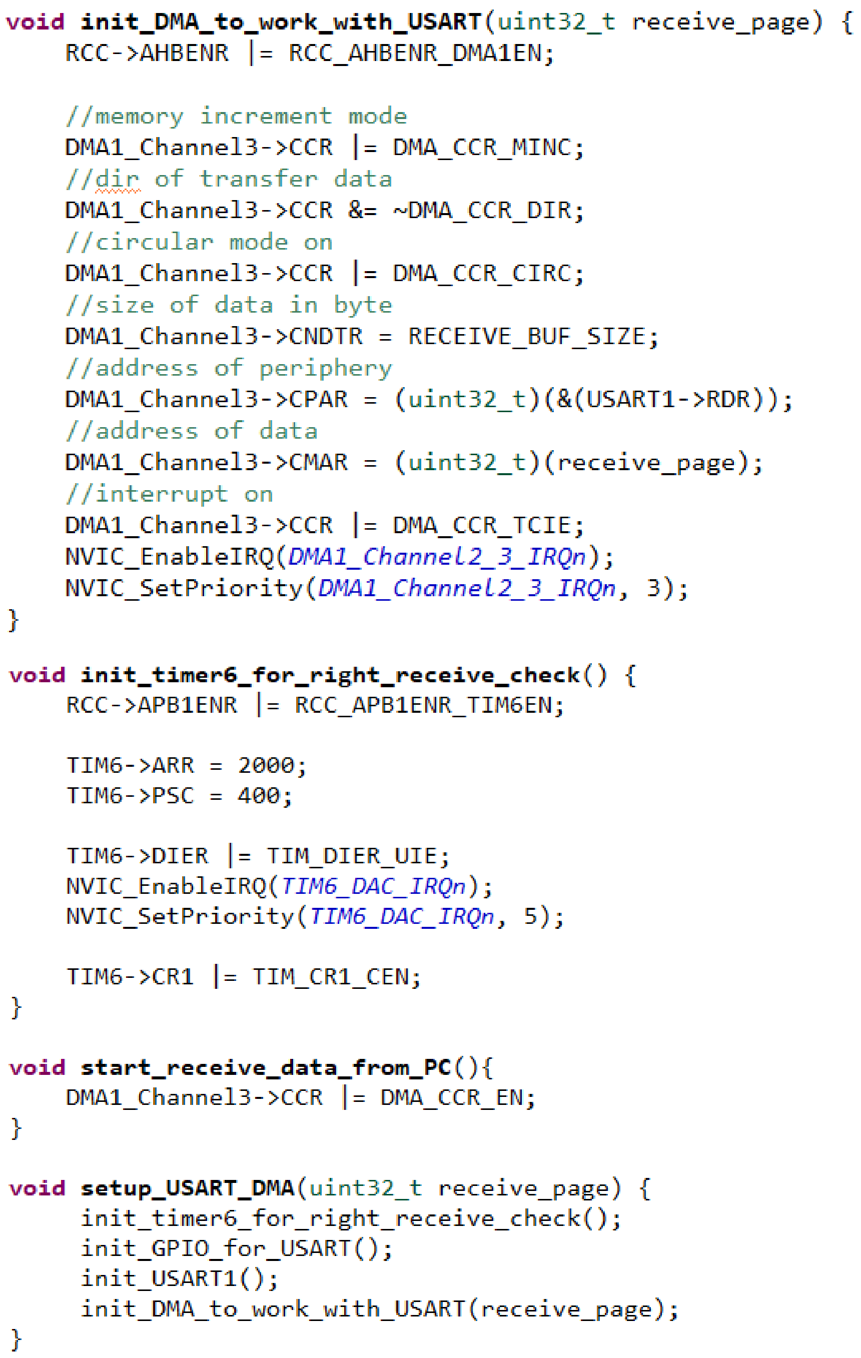


```c
#ifndef USART_DMA_SETUP_H_
#define USART_DMA_SETUP_H_


#include "stm32f0xx.h"
#include "Variables.h"
void TIM6_DAC_IRQHandler(void);
void USART1_IRQHandler(void);
void DMA1_Channel2_3_IRQHandler(void);
void init_GPIO_for_USART();
void init_USART1();
void init_DMA_to_work_with_USART(uint32_t receive_page);
void setup_USART_DMA(uint32_t receive_page);
void start_receive_data_from_PC();
void init_timer6_for_right_receive_check();

#endif /* USART_DMA_SETUP_H_ */
```

USART_DMA_setup.c

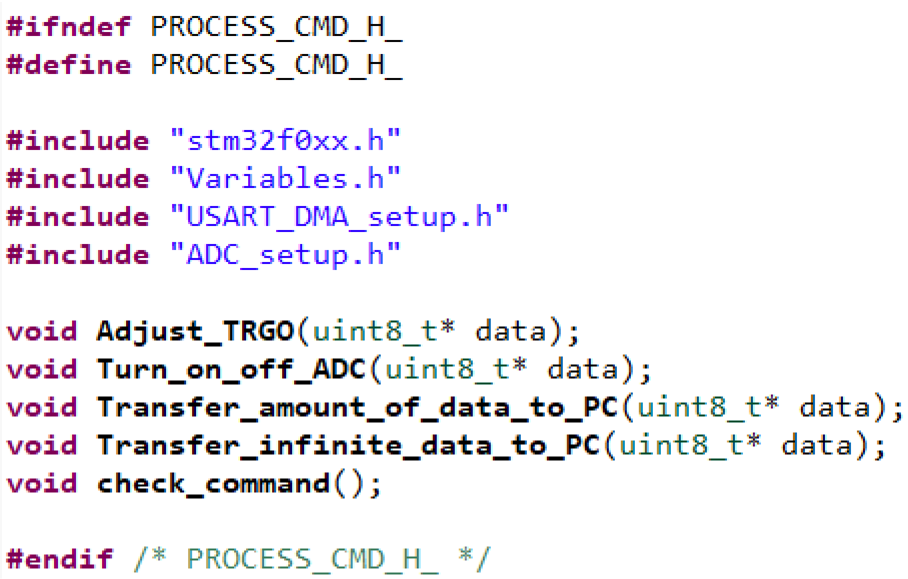


```c
#include "USART_DMA_setup.h"


void TIM6_DAC_IRQHandler(void) {
    TIM6->SR &= ~TIM_SR_UIF;
    if(DMA1_Channel3->CNDTR == RECEIVE_BUF_SIZE)
        return;
    if(program_state != RECEIVE_COMPLETE) {
        Clear_buffer(&Receive_buf);
        DMA1_Channel3->CCR &= ~DMA_CCR_EN;
        DMA1_Channel3->CNDTR = RECEIVE_BUF_SIZE;
        DMA1_Channel3->CCR |= DMA_CCR_EN;
    }
}


void USART1_IRQHandler(void) {
    if(USART1->ISR & USART_ISR_RXNE) {
        TIM6->CNT = 0;
    }
}


void DMA1_Channel2_3_IRQHandler(void) {
    if ((DMA1->ISR & DMA_ISR_TCIF3) == DMA_ISR_TCIF3) {
        DMA1->IFCR |= DMA_IFCR_CTCIF3;
        program_state = RECEIVE_COMPLETE;
        Receive_buf.amount_data_in_buf += RECEIVE_BUF_SIZE; //receive buf full of data
    }
}
void init_GPIO_for_USART(){
    RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
    //USART1_TX
    GPIOA->MODER |= GPIO_MODER_MODER9_1;
    //USART1_RX
    GPIOA->MODER |= GPIO_MODER_MODER10_1;
    //AF on 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;
    //interrupt
    USART1->CR1 |= USART_CR1_RXNEIE;

    NVIC_EnableIRQ(USART1_IRQn);
    NVIC_SetPriority(USART1_IRQn, 7);

    //USART1 on
    USART1->CR1 |= USART_CR1_UE;
}
```



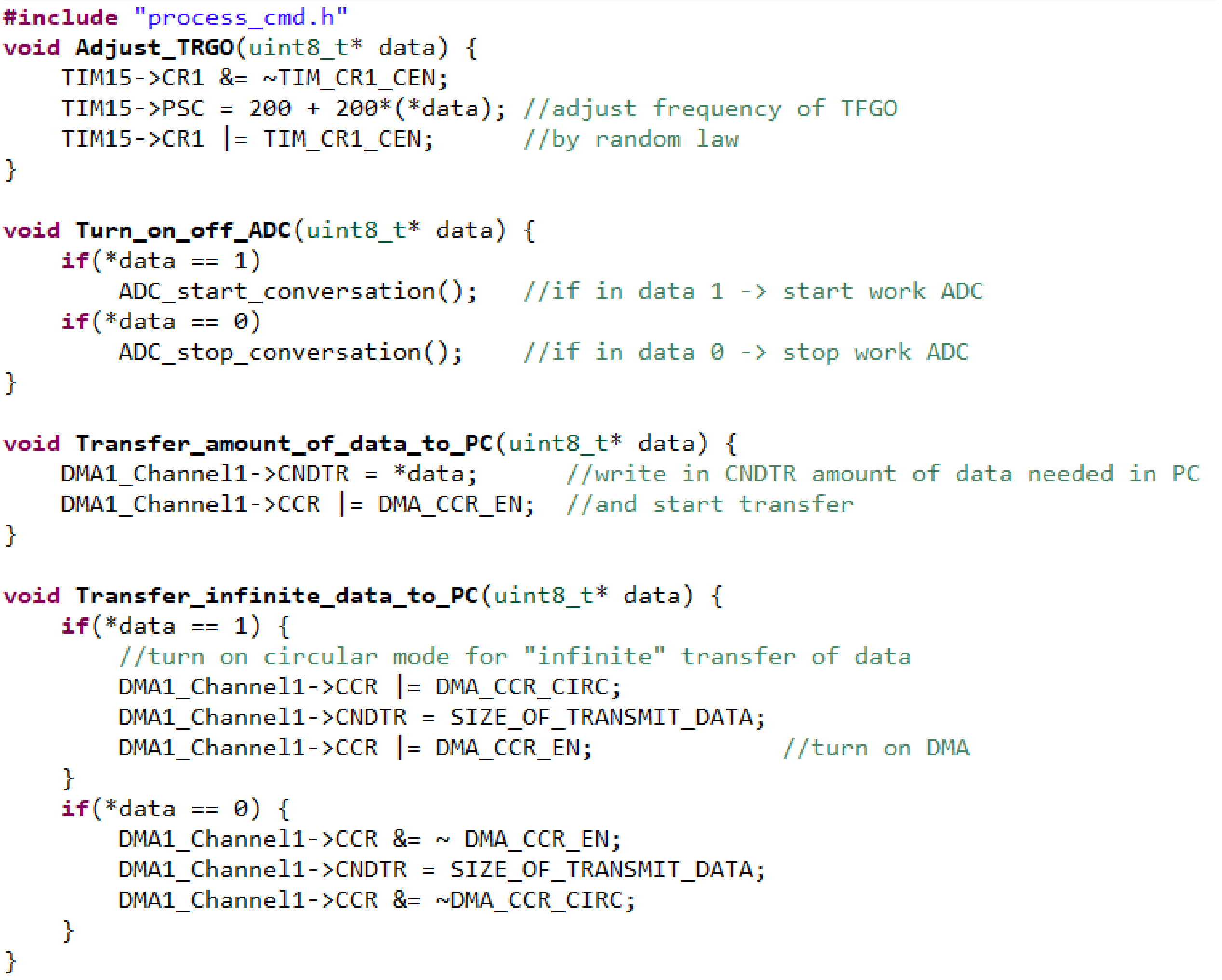
```c
void init_DMA_to_work_with_USART(uint32_t receive_page) {
    RCC->AHBENR |= RCC_AHBENR_DMA1EN;

    //memory increment mode
    DMA1_Channel3->CCR |= DMA_CCR_MINC;
    //dir of transfer data
    DMA1_Channel3->CCR &= ~DMA_CCR_DIR;
    //circular mode on
    DMA1_Channel3->CCR |= DMA_CCR_CIRC;
    //size of data in byte
    DMA1_Channel3->CNDTR = RECEIVE_BUF_SIZE;
    //address of periphery
    DMA1_Channel3->CPAR = (uint32_t)(&(USART1->RDR));
    //address of data
    DMA1_Channel3->CMAR = (uint32_t)(receive_page);
    //interrupt on
    DMA1_Channel3->CCR |= DMA_CCR_TCIE;
    NVIC_EnableIRQ(DMA1_Channel2_3_IRQn);
    NVIC_SetPriority(DMA1_Channel2_3_IRQn, 3);
}

void init_timer6_for_right_receive_check() {
    RCC->APB1ENR |= RCC_APB1ENR_TIM6EN;

    TIM6->ARR = 2000;
    TIM6->PSC = 400;

    TIM6->DIER |= TIM_DIER_UIE;
    NVIC_EnableIRQ(TIM6_DAC_IRQn);
    NVIC_SetPriority(TIM6_DAC_IRQn, 5);

    TIM6->CR1 |= TIM_CR1_CEN;
}

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

void setup_USART_DMA(uint32_t receive_page) {
     init_timer6_for_right_receive_check();
     init_GPIO_for_USART();
     init_USART1();
     init_DMA_to_work_with_USART(receive_page);
}
```

Process_cmd.h

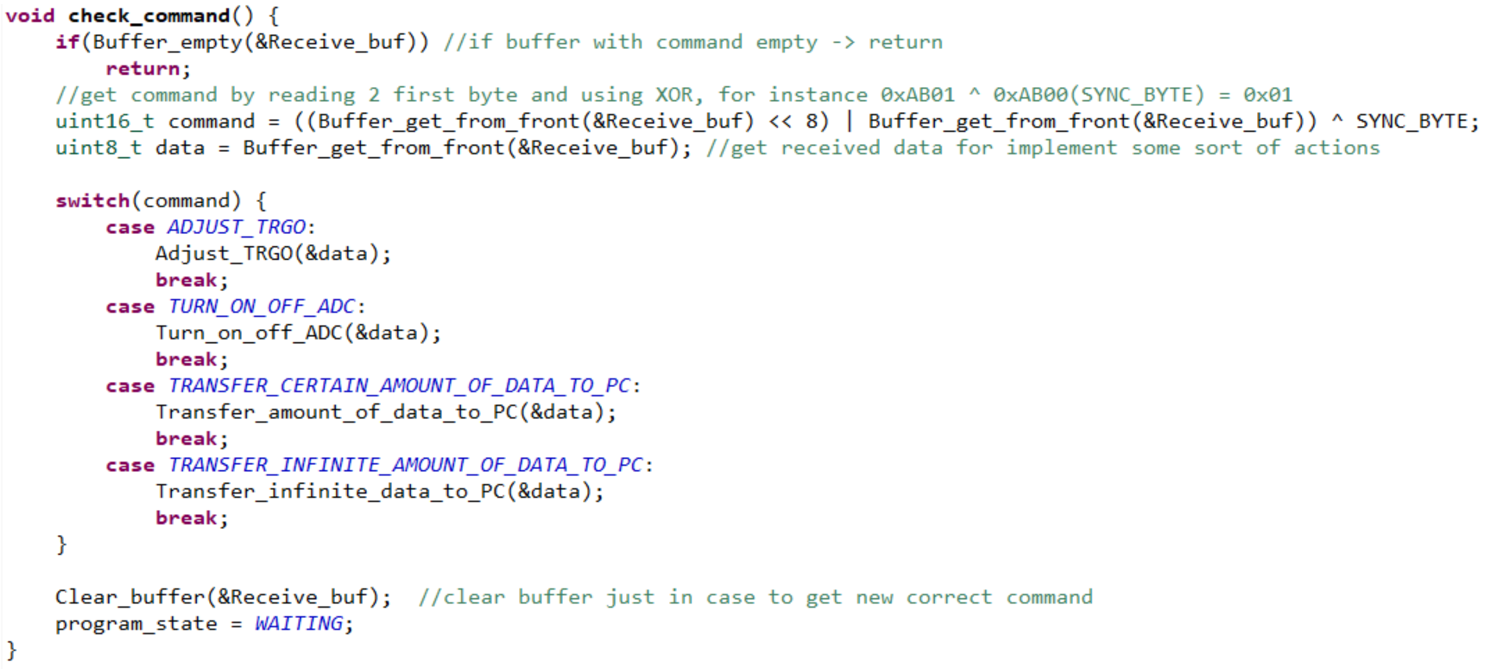


```c
#ifndef PROCESS_CMD_H_
#define PROCESS_CMD_H_

#include "stm32f0xx.h"
#include "Variables.h"
#include "USART_DMA_setup.h"
#include "ADC_setup.h"

void Adjust_TRGO(uint8_t* data);
void Turn_on_off_ADC(uint8_t* data);
void Transfer_amount_of_data_to_PC(uint8_t* data);
void Transfer_infinite_data_to_PC(uint8_t* data);
void check_command();

#endif /* PROCESS_CMD_H_ */
```

Process_cmd.c

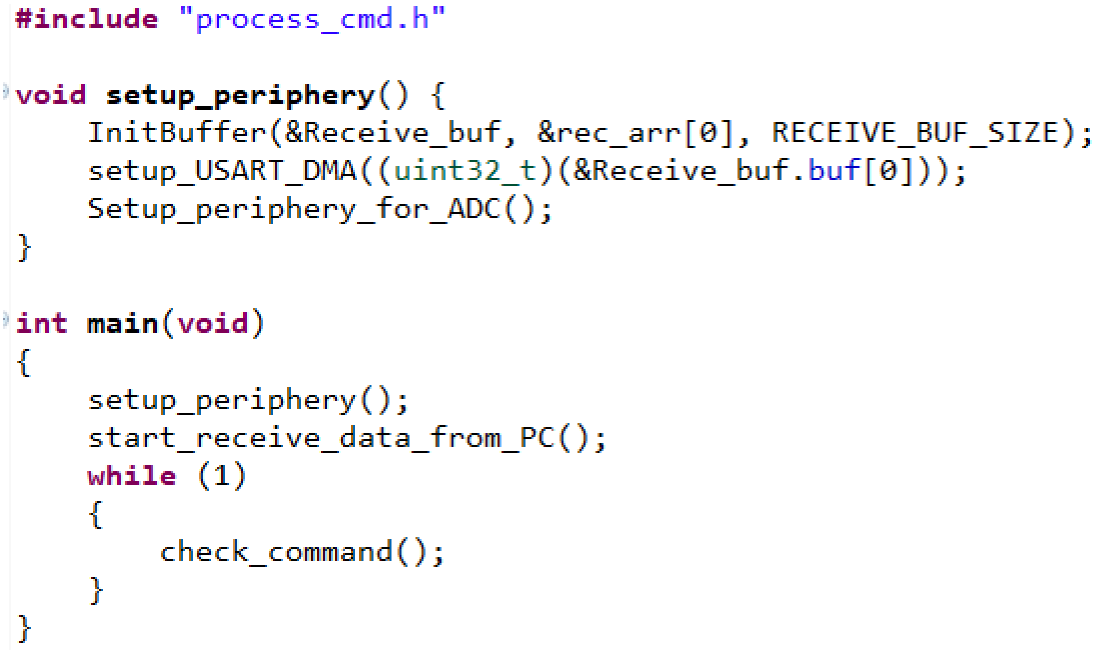


```c
#include "process_cmd.h"
void Adjust_TRGO(uint8_t* data) {
    TIM15->CR1 &= ~TIM_CR1_CEN;
    TIM15->PSC = 200 + 200*(*data); //adjust frequency of TFGO
    TIM15->CR1 |= TIM_CR1_CEN;      //by random law
}

void Turn_on_off_ADC(uint8_t* data) {
    if(*data == 1)
        ADC_start_conversation();   //if in data 1 -> start work ADC
    if(*data == 0)
        ADC_stop_conversation();    //if in data 0 -> stop work ADC
}

void Transfer_amount_of_data_to_PC(uint8_t* data) {
    DMA1_Channel1->CNDTR = *data;      //write in CNDTR amount of data needed in PC
    DMA1_Channel1->CCR |= DMA_CCR_EN;  //and start transfer
}

void Transfer_infinite_data_to_PC(uint8_t* data) {
    if(*data == 1) {
        //turn on circular mode for "infinite" transfer of data
        DMA1_Channel1->CCR |= DMA_CCR_CIRC;
        DMA1_Channel1->CNDTR = SIZE_OF_TRANSMIT_DATA;
        DMA1_Channel1->CCR |= DMA_CCR_EN;              //turn on DMA
    }
    if(*data == 0) {
        DMA1_Channel1->CCR &= ~ DMA_CCR_EN;
        DMA1_Channel1->CNDTR = SIZE_OF_TRANSMIT_DATA;
        DMA1_Channel1->CCR &= ~DMA_CCR_CIRC;
    }
}
```



```c
void check_command() {
    if(Buffer_empty(&Receive_buf)) //if buffer with command empty -> return
        return;
    //get command by reading 2 first byte and using XOR, for instance 0xAB01 ^ 0xAB00(SYNC_BYTE) = 0x01
    uint16_t command = ((Buffer_get_from_front(&Receive_buf) << 8) | Buffer_get_from_front(&Receive_buf)) ^ SYNC_BYTE;
    uint8_t data = Buffer_get_from_front(&Receive_buf); //get received data for implement some sort of actions

    switch(command) {
        case ADJUST_TRGO:
            Adjust_TRGO(&data);
            break;
        case TURN_ON_OFF_ADC:
            Turn_on_off_ADC(&data);
            break;
        case TRANSFER_CERTAIN_AMOUNT_OF_DATA_TO_PC:
            Transfer_amount_of_data_to_PC(&data);
            break;
        case TRANSFER_INFINITE_AMOUNT_OF_DATA_TO_PC:
            Transfer_infinite_data_to_PC(&data);
            break;
    }

    Clear_buffer(&Receive_buf);  //clear buffer just in case to get new correct command
    program_state = WAITING;
}
```

Main.c



```
#include "process_cmd.h"

void setup_periphery() {
    InitBuffer(&Receive_buf, &rec_arr[0], RECEIVE_BUF_SIZE);
    setup_USART_DMA((uint32_t)(&Receive_buf.buf[0]));
    Setup_periphery_for_ADC();
}

int main(void)
{
    setup_periphery();
    start_receive_data_from_PC();
    while (1)
    {
        check_command();
    }
}
```