```
/*
MODBUS RTU slave (USART)
04 READ INPUT REGISTER
https://www.modbustools.com/modbus.html#function04

Step 7.
- USART2 interrupt, ok
- wrong slave address, ok
- read 7 bytes from usartx, ok
- crc check, ok
- reads sensor, ok
- gives resonse to master, ok
- Reads ADC and sends data to master
- USART1
*/

/* Includes */


#include "stm32l1xx.h"
#define HSI_VALUE    ((uint32_t)16000000)
#include "nucleo152start.h"
#include <stdint.h>
#include <stdbool.h>

/* Private typedef */
/* Private define  */
/* Private macro */
/* Private variables */
/* Private function prototypes */
/* Private functions */
#define SLAVE_ADDRESS    0x01
#define FUNCTION_CODE    0x04
#define REGISTER_NUM    2
#define RESPONSE_BYTES   (REGISTER_NUM * 2)


#define LED_PIN         GPIO_ODR_ODR_5
#define TX_EN_PIN       GPIO_ODR_ODR_5  // Same PA5 controls MAX3485 RE/DE


volatile uint8_t modbus_flag = 0;
int sensor_values[REGISTER_NUM] = {0};  // 0: Temperature, 1: Humidity


void USART1_Init(void);
void USART1_write(uint8_t data);
uint8_t USART1_read(void);
void delay_Ms(int delay);
```

```
void delay_Us(int delay);
void delay_us(unsigned long delay);
void read_dht22_humidity_and_temperature(int *hum, int *temp);
void respond_with_sensor_data(int *values);
void wrong_slave_address(void);
unsigned short CRC16(const uint8_t *data, uint16_t length);

/**
**==========================================================================
====
**
**  Abstract: main program
**
**==========================================================================
====
*/
int main(void) {
   __disable_irq();
   USART1_Init();
   SetSysClock();
   SystemCoreClockUpdate();

   USART1->CR1 |= USART_CR1_RXNEIE;
   NVIC_EnableIRQ(USART1_IRQn);
   __enable_irq();

   // PA5 = TX_EN or LED
   RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
   GPIOA->MODER &= ~(3 << (5 * 2));
   GPIOA->MODER |= (1 << (5 * 2));

   uint8_t frame[8] = {0};

   while (1) {
      if (modbus_flag == 1) {
         frame[0] = SLAVE_ADDRESS;
         for (int i = 1; i < 8; i++) frame[i] = USART1_read();

         uint16_t crc_calc = CRC16(frame, 6);
         if (((crc_calc & 0xFF) == frame[6]) && ((crc_calc >> 8) == frame[7])) {
            uint16_t start_addr = (frame[2] << 8) | frame[3];
            uint16_t quantity = (frame[4] << 8) | frame[5];

            if (quantity > REGISTER_NUM || frame[1] != FUNCTION_CODE) {
               wrong_slave_address();
               continue;
            }
```

```
            read_dht22_humidity_and_temperature(&sensor_values[1], &sensor_values[0]);  // Hum, Temp
            respond_with_sensor_data(sensor_values);
          }
          modbus_flag = 0;
          USART1->CR1 |= USART_CR1_RXNEIE;

       } else if (modbus_flag == 2) {
          wrong_slave_address();
       }
    }
}

void USART1_IRQHandler(void) {
    if (USART1->SR & USART_SR_RXNE) {
       uint8_t addr = USART1->DR;
       modbus_flag = (addr == SLAVE_ADDRESS) ? 1 : 2;
       USART1->CR1 &= ~USART_CR1_RXNEIE;
    }
}

void USART1_Init(void) {
    RCC->APB2ENR |= RCC_APB2ENR_USART1EN;
    RCC->AHBENR |= RCC_AHBENR_GPIOAEN;

    GPIOA->AFR[1] &= ~(0xFF << 4);   // PA9, PA10
    GPIOA->AFR[1] |= (7 << 4) | (7 << 8);
    GPIOA->MODER &= ~(0xF << 18);
    GPIOA->MODER |= (2 << 18) | (2 << 20);

    USART1->BRR = 0x0D05; // 9600 baud @ 32MHz
    USART1->CR1 = USART_CR1_TE | USART_CR1_RE | USART_CR1_UE;
}

void USART1_write(uint8_t data) {
    while (!(USART1->SR & USART_SR_TXE));
    USART1->DR = data;
}

uint8_t USART1_read(void) {
    while (!(USART1->SR & USART_SR_RXNE));
    return USART1->DR;
}

void respond_with_sensor_data(int *values) {
    GPIOA->ODR |= TX_EN_PIN;

    uint8_t response[RESPONSE_BYTES + 5] = {0};
    response[0] = SLAVE_ADDRESS;
```

```
    response[1] = FUNCTION_CODE;
    response[2] = RESPONSE_BYTES;

    for (int i = 0; i < REGISTER_NUM; i++) {
        response[3 + i * 2] = (values[i] >> 8) & 0xFF;
        response[4 + i * 2] = values[i] & 0xFF;
    }

    uint16_t crc = CRC16(response, RESPONSE_BYTES + 3);
    response[RESPONSE_BYTES + 3] = crc & 0xFF;
    response[RESPONSE_BYTES + 4] = (crc >> 8) & 0xFF;

    for (int i = 0; i < RESPONSE_BYTES + 5; i++) {
        USART1_write(response[i]);
    }

    GPIOA->ODR &= ~TX_EN_PIN;
}

void wrong_slave_address(void) {
    USART1->CR1 &= ~USART_CR1_RE;
    delay_Ms(10);
    USART1->CR1 |= USART_CR1_RE | USART_CR1_RXNEIE;
    modbus_flag = 0;
}

void delay_Ms(int delay) {
    for (int i = 0; i < delay * 2460; ++i) __NOP();
}

void delay_Us(int delay) {
    for (int i = 0; i < delay * 2; ++i) __NOP();  // Approx 0.5us per iteration
}

void delay_us(unsigned long delay) {
    RCC->APB2ENR |= RCC_APB2ENR_TIM11EN;
    TIM11->PSC = 31;  // 32MHz / 32 = 1MHz (1us tick)
    TIM11->ARR = 1;
    TIM11->CNT = 0;
    TIM11->CR1 = 1;
    for (unsigned long i = 0; i < delay; ++i) {
        while (!(TIM11->SR & 1));
        TIM11->SR &= ~1;
        TIM11->CNT = 0;
    }
    TIM11->CR1 = 0;
}

unsigned short CRC16(const uint8_t *data, uint16_t length) {
```

```
    uint16_t crc = 0xFFFF;
    while (length--) {
        crc ^= *data++;
        for (uint8_t i = 0; i < 8; ++i) {
            crc = (crc & 1) ? (crc >> 1) ^ 0xA001 : crc >> 1;
        }
    }
    return crc;
}

void read_dht22_humidity_and_temperature(int *hum, int *temp) {
    uint32_t humidity = 0, temperature = 0, mask = 0x80000000;
    RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
    GPIOA->MODER |= (1 << (6 * 2));
    GPIOA->ODR |= (1 << 6);
    delay_Ms(10);
    GPIOA->ODR &= ~(1 << 6);
    delay_Ms(1);
    GPIOA->ODR |= (1 << 6);
    GPIOA->MODER &= ~(3 << (6 * 2));

    while (GPIOA->IDR & (1 << 6));
    while (!(GPIOA->IDR & (1 << 6)));
    while (GPIOA->IDR & (1 << 6));

    for (int i = 0; i < 32; i++) {
        while (!(GPIOA->IDR & (1 << 6)));
        delay_Us(35);
        if (GPIOA->IDR & (1 << 6)) {
            if (i < 16) humidity |= (mask >> 16);
            else temperature |= mask;
        }
        mask >>= 1;
        while (GPIOA->IDR & (1 << 6));
    }

    *hum = (int)
```