Научно-практический электронный журнал
Оригинальные исследования (ОРИС), *ISSN 2222-0402*
www.ores.su
original-research.ru

УДК 004.4

РАЗРАБОТКА И РЕАЛИЗАЦИЯ АЛГОРИТМА ВЗАИМОДЕЙСТВИЯ ДВУХ МИКРОКОНТРОЛЛЕРОВ ТИПА STM32 ПРИ ПРОВЕДЕНИИ АВТОМАТИЧЕСКИХ ПРОВЕРОК В РАМКАХ СТЕНДОВЫХ ИСПЫТАНИЙ[1]

Кыргыс Алдын-Херел Хеймер-оолович,
специалитет, Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет), кафедра РКТ-1 (системы автоматического управления), Москва, email: sepak1193@mail.ru

Морозов Кирилл Сергеевич,
специалитет, Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет), кафедра РКТ-1 (системы автоматического управления), Москва, email: kirillmorozov3108@gmail.com

Аннотация

В статье рассматривается подход к разработке программно-аппаратного комплекса для проведения автоматических стендовых испытаний. Основное внимание уделено архитектуре на основе интерфейсной платы с двумя микроконтроллерами STM32 и алгоритму их взаимодействия для управления испытательным оборудованием, сбора данных и коммуникации с верхнеуровневым ПО на ПК.

Ключевые слова: микроконтроллер, архитектура, протокол, STM32, плата интерфейсов, обмен, испытания, алгоритм.

DEVELOPMENT AND IMPLEMENTATION OF AN ALGORITHM FOR THE INTERACTION OF TWO STM32 TYPE MICROCONTROLLERS DURING AUTOMATED BENCH TESTS

Kyrgys Aldyn-Kherel Kheimer-oolovich,
Specialist, Bauman Moscow State Technical University (National Research University), RST-1 Department (automatic control systems), Moscow, email: sepak1193@mail.ru

[1] Научный руководитель: Деменев Дмитрий Андреевич – Начальник отдела проектной деятельностью студентов Студенческого научно-технического центра МГТУ им. Н. Э. Баумана, заместитель декана факультета Ракетно-космическая техника МГТУ им. Н. Э. Баумана, старший преподаватель кафедр ФН-7 (Электротехника и промышленная электроника), ИУ-1 (Системы автоматического управления), Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет), Москва, email: demenev@bmstu.ru
Scientific supervisor: Demenev Dmitry Andreevich – Head of the Department of Student Project Activities at the Bauman Moscow State Technical University Student Scientific and Technical Center, Deputy Dean of the Faculty of Rocket and Space Technology at Bauman Moscow State Technical University, Senior Lecturer at the departments FN-7 (Electrical Engineering and Industrial Electronics), IU-1 (Automatic Control Systems), Bauman Moscow State Technical University (National Research University), Moscow, email: demenev@bmstu.ru

Morozov Kirill Sergeevich,
Specialist, Bauman Moscow State Technical University (National Research University), RST-1 Department (automatic control systems), Moscow, email: kirillmorozov3108@gmail.com

ABSTRACT

The article considers an approach to the development of a software and hardware complex for conducting automatic bench tests. The main focus is on the architecture based on an interface card with two STM32 microcontrollers and the algorithm of their interaction for controlling test equipment, data collection and communication with top-level software on a PC.

Keywords: microcontroller, architecture, protocol, STM32, interface card, exchange, testing, algorithm.

Введение

Современные стендовые испытания сложных электронных систем, таких как бортовое оборудование космических аппаратов или авионика, требуют высокой степени автоматизации. Данный процесс обеспечивает повторяемость тестовых процедур, минимизацию операторских ошибок и сокращение длительности тестового цикла.

Однако, использование единого микроконтроллера (МК) для решения всех задач типичного испытательного стенда — управления исполнительными механизмами, генерации тестовых сигналов, точного измерения аналоговых величин и обеспечения высокоскоростного обмена с верхнеуровневым программным обеспечением (ПО) на персональном компьютере (ПК) — ведет к ряду проблем.

Перегрузка МК: Задачи, критичные ко времени могут прерываться сетевым обменом, что приводит к потере точности измерений.

Сложность кода: Прошивка становится монолитной, сложной для отладки, тестирования и модификации.

Низкая надежность: Ошибка в одном модуле может привести к критическим последствиям во всей системе, включая управление особо важными цепями.

Цель данного исследования — разработать и реализовать алгоритм взаимодействия между двумя МК, обеспечивающий выполнение автоматических проверок по командам от ПК в рамках стендовых испытаний.

Задача исследования – привести к минимуму участие человека при проведении стендовых испытаний сложных электронных систем.

Задача автоматизации в данном случае предполагается к выполнению с учетом следующих особенностей:

Наличие аналого-цифрового преобразователя (АЦП);

Наличие цифро-аналогового преобразователя (ЦАП);

Наличие входных сигналов приборов;

Наличие выходных сигналов приборов.

Предлагаемое нами решение предполагает функциональную декомпозицию задач между двумя МК на общей плате. Это позволяет эффективно распределить нагрузку, изолировать критические участки кода и создать модульную, масштабируемую систему.

Особенности микроконтроллера STM32F429ZI

Для реализации системы управления стендовыми испытаниями был выбран микроконтроллер STM32F429ZI, относящийся к высокопроизводительному семейству STM32F4 компании STMicroelectronics. Ключевым фактором выбора послужила его архитектура на базе ядра ARM Cortex-M4 с модулем вычислений с плавающей запятой (FPU). Наличие FPU имеет критическое значение для задач, связанных с обработкой данных в реальном времени, таких как быстрое преобразование сырых данных АЦП в инженерные величины (напряжения, токи) с применением калибровочных коэффициентов и цифровых фильтров, без существенной нагрузки на CPU.[1]

Основные преимущества STM32:

Оптимальное соотношение производительности и стоимости;

Удобство использования;

Большой выбор сред разработки;

Высокая вычислительная эффективность;

Развитая инфраструктура для отладки и диагностики.

Функциональная схема стенда испытаний

Предлагаемая функциональная схема стенда представляет собой систему, имеющую трёхуровневую архитектуру, где верхним уровнем является ПК, средним – два микроконтроллера типа STM32, а нижним – испытываемое оборудование или прибор. Схема стенда испытаний приведена на рисунке 1.

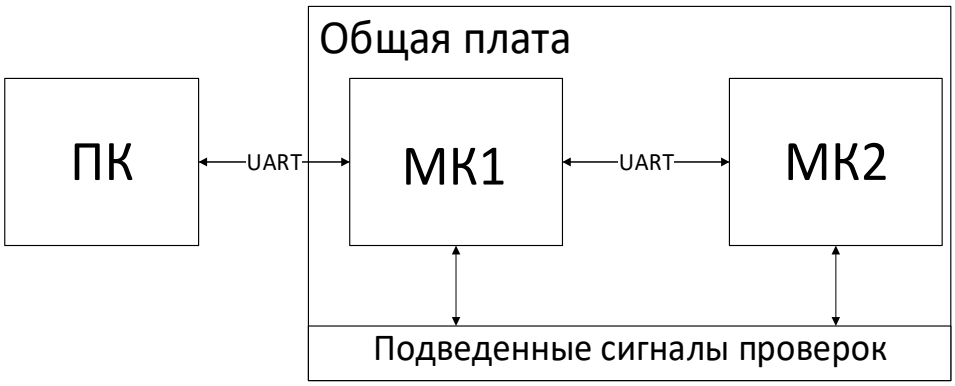

Рисунок 1 – Схема взаимодействия ПК, ПИ и МК

Верхний уровень (ПК): на персональном компьютере работает специализированное программное обеспечение.

Его задачи:

Предоставление интерфейса оператору;

Формирование и отправка сценариев испытаний;

Визуализация процесса испытаний в реальном времени;

Сбор, хранение и последующий анализ протоколов испытаний;

Коммуникация с интерфейсной платой по высокоскоростному каналу.

Общая плата: это аппаратный модуль, содержащий два микроконтроллера STM32.

МК1 (Контроллер управления и коммуникаций): выполняет роль связующего звена между ПК и остальной частью стенда. Его основные функции:

Прием и дешифровку команд от ПК;

Управление МК2 путем отправки ему заданий;

Первичная обработка и агрегация данных от МК2;

Частичная работа с аппаратурой стенда

Отправка отчетов о статусе и результатах испытаний на ПК.

МК2 (Контроллер ввода/вывода и измерений): отвечает за непосредственную работу с аппаратурой стенда. Его функции:

Управление силовыми и сигнальными реле, ключами (формирование необходимых цепей испытываемого устройства);

Генерация аналоговых и цифровых тестовых сигналов (ШИМ, ЦАП);

Точное измерение аналоговых сигналов (напряжения, токи) с помощью АЦП;

Считывание цифровых линий.

Теоретической основой для изучения архитектуры и принципов работы микроконтроллеров послужили работы А.Б. Белова [2], а также Б.В. Бородина и И.И. Шагурина [3].

Особенности микроконтроллера STM32F429ZI изучены по данным технической документации производителя [4, 5].

Подведенные сигналы проверок: сигналы испытываемого объекта, подключенного к выходам общей платы.

Связь между МК1 и МК2 организуется по одной из внутренних периферийных шин STM32, что обеспечивает высокоскоростной и надежный обмен данными.

Данный испытательный стенд предусматривает формирование как цифровых, так и аналоговых сигналов. Поскольку их конфигурация может изменяться в зависимости от решаемой задачи, ключевое значение имеет унифицированная основа системы. Далее будет описана её неизменяемая часть, которая обеспечивает гибкость при испытаниях различных устройств.

Описание протокола обмена

В целях реализации информационного обмена был разработан протокол, удовлетворяющий требованиям надежности, простоты реализации и целостности данных.

Выбранный протокол предполагает следующие структуры информационных посылок:

Пакет данных от ПК к прибору

На рисунке 2 представлена схема пакета данных, отправляемая из ПК в проверяемый прибор.

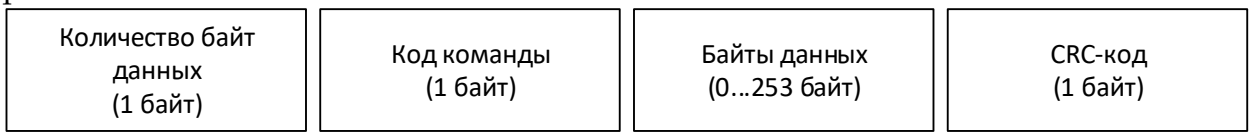

Рисунок 2 – Пакет данных от ПК к прибору

В расчете контрольной суммы (CRC-код) участвуют все байты пакета, включая код команды и количество байт данных.

Ответ от прибора к ПК

После приема командного пакета прибор ВСЕГДА отправляет в ПК статус корректности принятой команды (и параметров к ней, если они есть).

Если принятая команда корректна, то прибор отрабатывает её и отправляет в ПК код статуса отработки команды (1 байт), что показано на рисунке 3. В случае, если команда не корректна, прибор ничего не выполняет и отправляет в ПК только код статуса анализа команды (1 байт), этот случай изображен в виде схемы на рисунке 4.

Если команда требует передачи данных и при этом распознана корректно (то есть принята к исполнению), то сразу после отработки команды прибор передает в ПК код статуса отработки команды (1 байт без обвязки) и сразу за ним байты данных в виде пакета данных:

Первый байт, содержит кол-во байт в пакете, включая самого себя.

Непосредственно сами байты данных.

Последний байт - CRC-код, в расчете которого участвуют только байты пакета данных и не участвуют оба байта с кодами статусов.

Данный вариант ответа прибора изображен на рисунке 5.

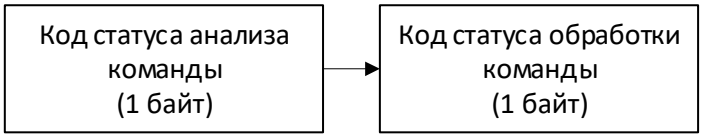

Рисунок 3 – Ответ прибора, если команда и ее параметры правильные, а передача данных не требуется

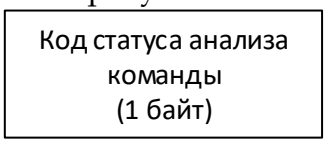

Рисунок 4 – Ответ прибора, если команда или ее параметры неправильные

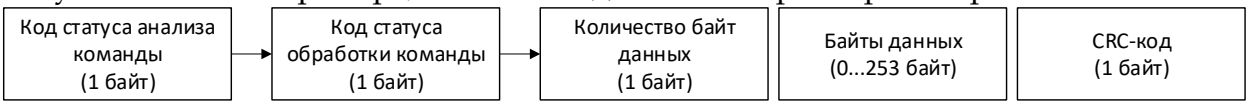

Рисунок 5 – Ответ прибора, если команда и ее параметры правильные и требуется передача данных

На основе этого протокола обмена и функциональной схемы взаимодействия, описанной выше, нами был разработан алгоритм взаимодействия двух МК STM32F429ZI, представленный блок-схеме на рисунке 6.

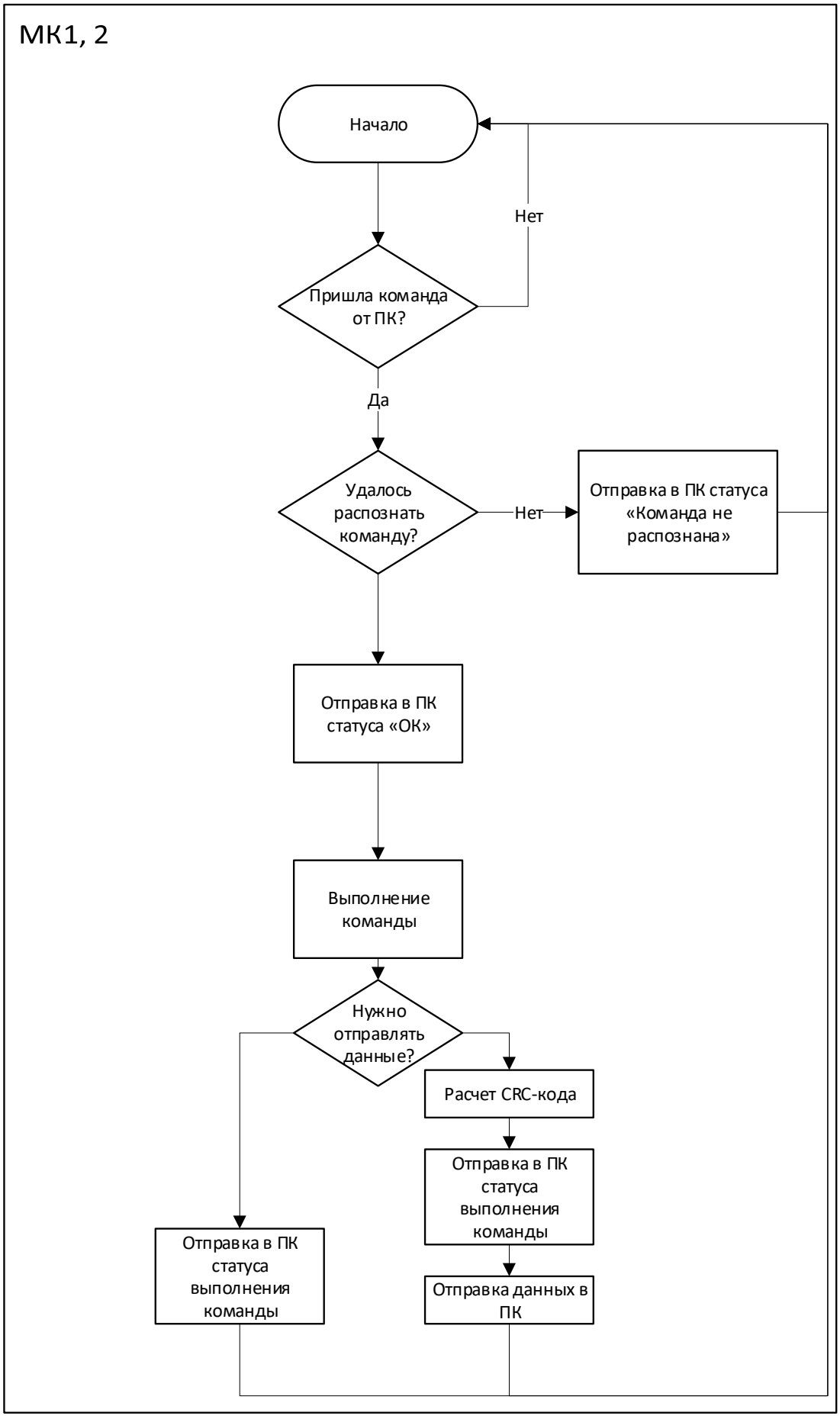

Рисунок 6 – Блок-схема алгоритма взаимодействия двух МК и ПК

Разработанный алгоритм взаимодействия реализует конечный автомат с основной цикличной обработкой команд. Его ключевые особенности:

Ожидание команды: оба МК в основном цикле постоянно опрашивают свои буферы связи (с ПК для МК1, с МК1 для МК2) на предмет поступления новых пакетов.

Верификация: при получении пакета первым делом проверяется его целостность с помощью контрольной суммы. Некорректный пакет игнорируется, отправителю передается код ошибки.

Диспетчеризация: после успешной верификации код команды ищется в таблице команд. Это гибкое решение, позволяющее легко добавлять новые функции в систему.

Выполнение: при нахождении команды вызывается соответствующая функция-обработчик. Для МК1 это часто функции, связанные с коммуникацией (запрос данных у МК2, отправка отчета ПК). Для МК2 — это функции непосредственного управления периферией (установка уровня на ЦАП, чтение АЦП, переключение реле).

Обратная связь: Система всегда предоставляет обратную связь. Даже в случае неизвестной команды отправитель получает соответствующий код статуса.

При разработке алгоритма была использована методика проектирования программного обеспечения, предложенная Е.А. Микриным [6].

Реализация представленного алгоритма была выполнена на языке С в среде STM32CubeIDE с применением библиотеки HAL. Особенности и методы работы с данной библиотекой приведены в соответствующем руководстве пользователя [7]. Текст программы приведен в приложении.

Заключение

В ходе работы была успешно решена задача декомпозиции функций испытательного стенда между двумя микроконтроллерами STM32. Разработанные аппаратная платформа и программный алгоритм взаимодействия показали свою высокую эффективность в реальных условиях.

Достигнута высокая надежность: критичный код управления аппаратурой (МК2) изолирован от потенциально нестабильного сетевого обмена (МК1). Наличие таймера на каждом МК гарантирует восстановление работы при сбоях.

Обеспечена точность измерений: МК2, не отвлекаясь на задачи коммуникации с ПК, выполняет измерения АЦП и генерацию сигналов с максимальной временной предсказуемостью.

Создана модульная система: архитектура "ведущий-ведомый" и табличный диспетчер команд позволяют легко добавлять новую функциональность, модифицируя код лишь одного из контроллеров.

Упрощены разработка и отладка: разделение кода позволило вести параллельную разработку и тестирование модулей МК1 и МК2.

Конфигурация и настройки микроконтроллеров варьируются в зависимости от конкретной решаемой задачи, что требует генерации соответствующих сигналов и изменения их параметров. Ключевым принципом является то, что эти изменения не затрагивают единые протоколы обмена данными: как между персональным компьютером и ведущим контроллером (МК-мастером), так и между самими микроконтроллерами. Это разделение обеспечивает гибкость проведения испытаний и позволяет легко адаптировать стенд под новые тестовые сценарии без переработки базовой коммуникационной логики.

Предложенный подход является универсальным и может быть успешно применен для автоматизации испытаний широкого спектра электронных устройств и систем, где предъявляются высокие требования к надежности, точности и масштабируемости.

Список литературы:

1. Морозов К.С., Кыргыс А.Х., Деменев Д.А. Архитектура, классификация и интерфейсы современных микроконтроллеров: сравнительный анализ STM32, AVR

и ESP / Научный альманах 2025 № 5-2. 46с. [Электронный ресурс]. – Режим доступа: https://ukonf.com/doc/na.2025.05.02.pdf (дата обращения: 02.10.2025)
2. Белов А.Б. Конструирование устройств на микроконтроллерах / Наука и Техника, 2005. 255 с.
3. Бродин Б.В., Шагурин И.И. Микроконтроллеры: Справочник. М.: ЭКОМ, 1999. 395 с.
4. User manual for the STM32 microcontroller [Электронный ресурс]. – Режим доступа: https://www.st.com/resource/en/user_manual/dm00244518-stm32-nucleo144-boards-mb1137-stmicroelectronics.pdf (дата обращения: 02.10.2025)
5. Technical parameters of the STM32F4 microcontroller [Электронный ресурс]. – Режим доступа: https://www.st.com/en/microcontrollers-microprocessors/stm32f4-series.html (дата обращения: 02.10.2025)
6. Е.А. Микрин Бортовые комплексы управления космическими аппаратами и проектирование их программного обеспечения/ М.: Издательство МГТУ им. Н.Э. Баумана, 2003. - 8, 9, 39с.
7. User manual Description of STM32F1 HAL and low-layer drivers [Электронный ресурс]. – Режим доступа: https://www.st.com/resource/en/user_manual/dm00154093-description-of-stm32f1-hal-and-lowlayer-drivers-stmicroelectronics.pdf (дата обращения: 02.10.2025)

References:

1. Morozov K.S., Kyrgys A.K., Demenev D.A. Architectura, classificatsia i interfeicy sovremennikh microcontrollerov: sravnitelniy analis STM32, AVR i ESP [Architecture, classification, and interfaces of modern microcontrollers: a comparative analysis of STM32, AVR and ESP] / Science Almanac 2025 № 5-2. 46p. [Elektronnyi resurs]. – Available at: https://ukonf.com/doc/na.2025.05.02.pdf (accessed: 02.10.2025) (in Russian)
2. Belov A.B. Konstruirovanie ustroistv na mikrokontrollerakh [Designing devices on microcontrollers], Nauka i Tekhnika, 2005. 255 p. (in Russian)
3. Brodin B.V., Shagurin I.I. Mikrokontrollery: Spravochnik [Microcontrollers: A reference book] Moscow: EKOM, 1999. 395 p. (in Russian)
4. User manual for the STM32 microcontroller [Elektronnyi resurs] – Available at: https://www.st.com/resource/en/user_manual/dm00244518-stm32-nucleo144-boards-mb1137-stmicroelectronics.pdf (accessed: 02.10.2025)
5. Technical parameters of the STM32F4 microcontroller [Elektronnyi resurs] – Available at: https://www.st.com/en/microcontrollers-microprocessors/stm32f4-series.html (accessed: 02.10.2025)
6. E.A. Mikrin Bortovie kompleksy upravleniya kosmicheskimi apparatami i proektirovanie ikh programmnogo obespecheniya [On-board spacecraft control systems and their software design]: M.: Publishing House of Bauman Moscow State Technical University, 2003. - 8, 9, 39p. (in Russian)
7. User manual Description of STM32F1 HAL and low-layer drivers [Elektronnyi resurs] – Available at: https://www.st.com/resource/en/user_manual/dm00154093-description-of-stm32f1-hal-and-lowlayer-drivers-stmicroelectronics.pdf (accessed: 02.10.2025)

Приложение:
Код разработанных программ для микроконтроллеров
Программный код в файле «main.c»

```
/* USER CODE BEGIN Header */
/**
 ******************************************************************************
 * @file           : main.c
 * @brief          : Main program body
 ******************************************************************************
 * @attention
 *
 * Copyright (c) 2023 STMicroelectronics.
 * All rights reserved.
 *
 * This software is licensed under terms that can be found in the LICENSE file
 * in the root directory of this software component.
 * If no LICENSE file comes with this software, it is provided AS-IS.
 *
 ******************************************************************************
 */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "hw_config.h"
#include "uart_init.h"
#include "gpio_init.h"
#include "type_new.h"
#include "main_cpu_2.h"
#include "stm32f4xx_it.h"
#include "dac_init.h"
#include "adc_init.h"
/* USER CODE END Includes */
/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
/* USER CODE END PTD */
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define  CPU1_PROG_VERSION      0x0200
/* USER CODE END PD */
/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
/* USER CODE END PM */
/* Private variables ---------------------------------------------------------*/
/* USER CODE BEGIN PV */
extern ADC_HandleTypeDef hadc1;
extern ADC_HandleTypeDef hadc3;
```

```
extern DAC_HandleTypeDef hdac;
extern IWDG_HandleTypeDef hiwdg;
extern TIM_HandleTypeDef htim8;
extern TIM_HandleTypeDef htim10;
extern UART_HandleTypeDef huart4;
extern UART_HandleTypeDef huart7;
extern UART_HandleTypeDef huart8;
extern UART_HandleTypeDef huart2;
extern UART_HandleTypeDef huart3;
static BYTE *gBufCmd;
WORD SetBit_1[5] = {0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF};
volatile static BYTE CPU1_CRASH_StopFlag = 0;
/* USER CODE END PV */
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
/* USER CODE BEGIN PFP */
void  command_1 (void);
void  command_2 (void);
void  command_3 (void);
/* USER CODE END PFP */
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
const T_COMMAND Command[] = {
  {CMD_TEST_1,                command_1},
  { CMD_TEST_2,                command_2},
  { CMD_TEST_3,                command_3}
};
/* USER CODE END 0 */
/**
 * @brief  The application entry point.
 * @retval int
 */
int main(void)
{
  /* USER CODE BEGIN 1 */
WORD idx;
BOOL bFindCmd;
void (*ptr_func)(void);
  /* USER CODE END 1 */
  /* MCU Configuration--------------------------------------------------------*/
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();
  /* USER CODE BEGIN Init */
  if (CheckCPU() == CPU_1){
          Init_GPIO(CPU_1);
   }
   else {
          Main_CPU_2();
```

```
  }
/* USER CODE END Init */
/* Configure the system clock */
SystemClock_Config();
/* USER CODE BEGIN SysInit */
if (CheckCPU() == CPU_2) MX_UART4_Init();
MX_UART7_Init();
MX_UART8_Init();
MX_USART2_UART_Init();
MX_USART3_UART_Init();
MX_DAC_Init();
MX_TIM8_Init();
MX_TIM10_Init();
/* USER CODE END SysInit */
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_IWDG_Init();
MX_UART4_Init();
MX_UART7_Init();
MX_UART8_Init();
MX_USART2_UART_Init();
MX_USART3_UART_Init();
MX_ADC3_Init();
MX_DAC_Init();
MX_TIM8_Init();
MX_TIM10_Init();
MX_ADC1_Init();
/* USER CODE BEGIN 2 */
__HAL_UART_ENABLE_IT(&huart2, UART_IT_RXNE);
__HAL_UART_ENABLE_IT(&huart3, UART_IT_RXNE);
__HAL_UART_ENABLE_IT(&huart4, UART_IT_RXNE);
__HAL_UART_ENABLE_IT(&huart7, UART_IT_RXNE);
__HAL_UART_ENABLE_IT(&huart8, UART_IT_RXNE);
HAL_TIM_PWM_Start(&htim8, TIM_CHANNEL_1);
HAL_TIM_PWM_Start(&htim10, TIM_CHANNEL_1);
HAL_DAC_Start(&hdac, DAC_CHANNEL_1);
gBufCmd = (BYTE *)SetBit_1;
/* USER CODE END 2 */
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
        if (!CPU1_CRASH_StopFlag){
            RESET_CRASH();
        }
        gBufCmd = GetCommandUART();
        if (gBufCmd == 0)
           continue;
        // ищем команду в списке
```

```
                bFindCmd = FALSE;
                for (idx = 0; idx < NUMELEM(Command); idx++) {
                   if (Command[idx].code == gBufCmd[1]) {
                     bFindCmd = TRUE;
                     break;
                   }
                }
                if (bFindCmd) {
                    SendUART_Byte(UART_MAIN, UART_RES_OK);
                    ptr_func = Command[idx].ptrFunc;
                    if (ptr_func)
                        ptr_func();
                }
                // команда не найдена
                else {
                       SendUART_Byte(UART_MAIN, UART_RES_UNKNOWN);
                }
            /* USER CODE END WHILE */
            /* USER CODE BEGIN 3 */
          }
          /* USER CODE END 3 */
        }
        /**
          * @brief System Clock Configuration
          * @retval None
          */
        void SystemClock_Config(void)
        {
          RCC_OscInitTypeDef RCC_OscInitStruct = {0};
          RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
          /** Configure the main internal regulator output voltage
          */
          __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);
          /** Initializes the RCC Oscillators according to the specified parameters
          * in the RCC_OscInitTypeDef structure.
          */
          RCC_OscInitStruct.OscillatorType                                    =
RCC_OSCILLATORTYPE_LSI|RCC_OSCILLATORTYPE_HSE;
          RCC_OscInitStruct.HSEState = RCC_HSE_BYPASS;
          RCC_OscInitStruct.LSIState = RCC_LSI_ON;
          RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
          RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
          RCC_OscInitStruct.PLL.PLLM = 8;
          RCC_OscInitStruct.PLL.PLLN = 256;
          RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
          RCC_OscInitStruct.PLL.PLLQ = 6;
          if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
```

```
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType                                                    =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                    |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV4;
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4) != HAL_OK)
  {
    Error_Handler();
  }
}
/*
*******************************************************************************
  Пример команды 1
*******************************************************************************
*/
void command_1 (void)
{
}
/*
*******************************************************************************
  Пример команды 2
*******************************************************************************
*/
void command_2(void)
{
}
/*
*******************************************************************************
  Пример команды 3
*******************************************************************************
*/
void command_3(void)
{
}
/*
static void MX_GPIO_Init(void)
{
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */
  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOF_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
```

```
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}
/* USER CODE BEGIN 4 */
/* USER CODE END 4 */
/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}
#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Программный код в файле «main_cpu_2.c»

```
/*
 * main_cpu_2.c
*  Created on: 19 февр. 2024 г.
 *      Author: mvide
 */
#include "hw_config.h"
#include "gpio_init.h"
#include "uart_init.h"
```

```
#include "main_cpu_2.h"
#define  CPU2_PROG_VERSION     0x0200
/*
*****************************************************************************
    П Р О Т О Т И П Ы   Ф У Н К Ц И Й
*****************************************************************************
*/
void  command_4 (void);
void  command_5 (void);
void  command_6 (void);
// external variables
extern IWDG_HandleTypeDef hiwdg;
extern TIM_HandleTypeDef htim8;
extern TIM_HandleTypeDef htim10;
extern UART_HandleTypeDef huart4;
extern UART_HandleTypeDef huart7;
extern UART_HandleTypeDef huart8;
extern UART_HandleTypeDef huart2;
extern UART_HandleTypeDef huart3;
/*
*****************************************************************************
    Г Л О Б А Л Ь Н Ы Е   П Е Р Е М Е Н Н Ы Е
*****************************************************************************
*/
static BYTE *_gBufCmd;
//===== Таблица команд
const T_COMMAND Command[] = {
  {CMD_TEST_4,                command_4},
  { CMD_TEST_5,                command_5},
  { CMD_TEST_6,                command_6}
};volatile static BYTE CPU2_CRASH_StopFlag = 0;
/*
=============================================================================
=========
                    M A I N   C P U 2
=============================================================================
======
*/
void Main_CPU_2 (void)
{
  WORD idx;
  BOOL bFindCmd;
  void (*ptr_func)(void);
  Init_GPIO(CPU_2);        // Инициализация всех ног CPU2
  MX_IWDG_Init();
  MX_TIM8_Init();
  MX_TIM10_Init();
  if (CheckCPU() == CPU_2) MX_UART4_Init();
  MX_UART7_Init();
```

```
  MX_UART8_Init();
  MX_USART2_UART_Init();
  MX_USART3_UART_Init();
  /* USER CODE BEGIN 2 */
  __HAL_UART_ENABLE_IT(&huart2, UART_IT_RXNE);
  __HAL_UART_ENABLE_IT(&huart3, UART_IT_RXNE);
  __HAL_UART_ENABLE_IT(&huart4, UART_IT_RXNE);
  __HAL_UART_ENABLE_IT(&huart7, UART_IT_RXNE);
  __HAL_UART_ENABLE_IT(&huart8, UART_IT_RXNE);
  HAL_TIM_PWM_Start(&htim8, TIM_CHANNEL_1);
  HAL_TIM_PWM_Start(&htim10, TIM_CHANNEL_1);
  while (1)
   {
                if (!CPU2_CRASH_StopFlag){
                    RESET_CRASH();
                }
                _gBufCmd = GetDataCPU1();
                if (_gBufCmd == 0)
                   continue;
                // ищем команду в списке
                bFindCmd = FALSE;
                for (idx = 0; idx < NUMELEM(_Command); idx++) {
                   if (_Command[idx].code == _gBufCmd[1]) {
                     bFindCmd = TRUE;
                     break;
                   }
                }
                if (bFindCmd) {
                     SendUART_Byte(UART_CPU2_1, UART_RES_OK);
                     ptr_func = _Command[idx].ptrFunc;
                     if (ptr_func)
                         ptr_func();
                }
                // команда не найдена
                else {
                        SendUART_Byte(UART_CPU2_1, UART_RES_CMD_2);
                }
     /* USER CODE END WHILE */
     /* USER CODE BEGIN 3 */
   }
}
/*
*********************************************************************************
  Пример команды 4
*********************************************************************************
*/
void  command_4(void)
{
}
```

```
/*
*********************************************************************************
  Пример команды 5
*********************************************************************************
*/
void command_5 (void)
{
}
/*
*********************************************************************************
  Пример команды 6
*********************************************************************************
*/
void command_6 (void)
{
}
```

программный код в файле «uart_init.c»

```
#include "stm32f4xx_hal_conf.h"
#define UART_INIT_GLOBALS
#include "uart_init.h"
UART_HandleTypeDef huart4;
UART_HandleTypeDef huart7;
UART_HandleTypeDef huart8;
UART_HandleTypeDef huart2;
UART_HandleTypeDef huart3;
/*
 * определение номера контроллера
 */
T_NUM_CPU CheckCPU (void)
{
  GPIO_InitTypeDef GPIO_InitStruct;
  static T_NUM_CPU num = CPU_NONE;
  if (num == 0) {
  __HAL_RCC_GPIOC_CLK_ENABLE();
     GPIO_InitStruct.Pin = GPIO_PIN_9;
     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
     GPIO_InitStruct.Pull = GPIO_PULLDOWN;
     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
     if (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_9) == GPIO_PIN_RESET)
       num = CPU_1;
     else
       num = CPU_2;
  }
  return(num);
}
// определение контрольной суммы
BYTE Crc8 (BYTE *buf, WORD len)
{
```

```
  BYTE crc = 0x00;
  BYTE i;
  while (len--) {
    crc ^= *buf++;
    for (i = 0; i < 8; i++)
      crc = crc & 1 ? (crc >> 1) ^ 0x8C : crc >> 1;
  }
  return(crc);
}
/*
*****************************************************************************
  Запись в кольцевой буфер
*****************************************************************************
*/
void PutBufferUART (BYTE data, T_BUF_UART *buf)
{
  if (buf->Count < SIZE_BUF_UART) {
    buf->buf[buf->Tail] = data;
    buf->Tail++;
    buf->Count++;
    if (buf->Tail >= SIZE_BUF_UART)
      buf->Tail = 0;
  }
}
/*
*****************************************************************************
  Чтение из кольцевого буфера самого старого данного
*****************************************************************************
*/
BYTE GetBufferUART (T_BUF_UART *buf)
{
  BYTE data;
  if (buf->Count) {
    data = buf->buf[buf->Head];
    buf->Count--;
    buf->Head++;
    if (buf->Head >= SIZE_BUF_UART)
      buf->Head = 0;
    return(data);
  }
  return(0);
}
/*
*****************************************************************************
  проверка кольцевого буфера
*****************************************************************************
*/
BOOL IsBufferEmptyUART (T_BUF_UART *buf)
{
```

```
  return(buf->Tail == buf->Head);
}
WORD GetNumBytesUART (T_BUF_UART *buf)
{
  return(buf->Count);
}
void ClearBufferUART (T_BUF_UART *buf)
{
  buf->Count = 0;
  buf->Tail = 0;
  buf->Head = 0;
}
/**
 * @brief UART4 Initialization Function
 * @param None
 * @retval None
 */
void MX_UART4_Init(void)
{
 /* USER CODE BEGIN UART4_Init 0 */
 /* USER CODE END UART4_Init 0 */
 /* USER CODE BEGIN UART4_Init 1 */
 /* USER CODE END UART4_Init 1 */
 huart4.Instance = UART4;
 huart4.Init.BaudRate = 115200;
 huart4.Init.WordLength = UART_WORDLENGTH_8B;
 huart4.Init.StopBits = UART_STOPBITS_2;
 huart4.Init.Parity = UART_PARITY_NONE;
 huart4.Init.Mode = UART_MODE_TX_RX;
 huart4.Init.HwFlowCtl = UART_HWCONTROL_NONE;
 huart4.Init.OverSampling = UART_OVERSAMPLING_16;
 if (HAL_UART_Init(&huart4) != HAL_OK)
 {
  Error_Handler();
 }
 /* USER CODE BEGIN UART4_Init 2 */
 /* USER CODE END UART4_Init 2 */
/**
 * @brief UART7 Initialization Function
 * @param None
 * @retval None
 */
void MX_UART7_Init(void)
{
 /* USER CODE BEGIN UART7_Init 0 */
 /* USER CODE END UART7_Init 0 */
 /* USER CODE BEGIN UART7_Init 1 */
 /* USER CODE END UART7_Init 1 */
 huart7.Instance = UART7;
```

```
  huart7.Init.BaudRate = 115200;
  huart7.Init.WordLength = UART_WORDLENGTH_8B;
  huart7.Init.StopBits = UART_STOPBITS_2;
  huart7.Init.Parity = UART_PARITY_NONE;
  huart7.Init.Mode = UART_MODE_TX_RX;
  huart7.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart7.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart7) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN UART7_Init 2 */
  /* USER CODE END UART7_Init 2 */
}
/**
  * @brief UART8 Initialization Function
  * @param None
  * @retval None
  */
void MX_UART8_Init(void)
{
  /* USER CODE BEGIN UART8_Init 0 */
  /* USER CODE END UART8_Init 0 */
  /* USER CODE BEGIN UART8_Init 1 */
  /* USER CODE END UART8_Init 1 */
  huart8.Instance = UART8;
  huart8.Init.BaudRate = 115200;
  huart8.Init.WordLength = UART_WORDLENGTH_8B;
  huart8.Init.StopBits = UART_STOPBITS_2;
  huart8.Init.Parity = UART_PARITY_NONE;
  huart8.Init.Mode = UART_MODE_TX_RX;
  huart8.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart8.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart8) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN UART8_Init 2 */
  /* USER CODE END UART8_Init 2 */
}
/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
void MX_USART2_UART_Init(void)
{
  /* USER CODE BEGIN USART2_Init 0 */
  /* USER CODE END USART2_Init 0 */
```

```
  /* USER CODE BEGIN USART2_Init 1 */
  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_2;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */
  /* USER CODE END USART2_Init 2 */
}
/**
  * @brief USART3 Initialization Function
  * @param None
  * @retval None
  */
void MX_USART3_UART_Init(void)
{
  /* USER CODE BEGIN USART3_Init 0 */
  /* USER CODE END USART3_Init 0 */
  /* USER CODE BEGIN USART3_Init 1 */
  /* USER CODE END USART3_Init 1 */
  huart3.Instance = USART3;
  huart3.Init.BaudRate = 115200;
  huart3.Init.WordLength = UART_WORDLENGTH_8B;
  huart3.Init.StopBits = UART_STOPBITS_2;
  huart3.Init.Parity = UART_PARITY_NONE;
  huart3.Init.Mode = UART_MODE_TX_RX;
  huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart3.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart3) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART3_Init 2 */
  /* USER CODE END USART3_Init 2 */
}
/*
*********************************************************************************
   ожидание и прием данных от ПК
   1-й байт длина данных
   1 задний CRC8
*********************************************************************************
```

```
*/
BYTE *GetCommandUART (void)
{
  static BYTE buf[SZ_MAX_UART];
  BYTE crc;
  DWORD tm;
  WORD idx = 0;
  WORD len;
  BYTE res_time_out, res_crc;
  HAL_StatusTypeDef check;
  HAL_UART_StateTypeDef check1;

  if (CheckCPU() != CPU_1) {
    return(0);
  }

  //ждем 1-й байт
  if (IsBufferEmptyUART(&gBufUART_Main))
    return(0);
  tm = GetTimeSys();
  len = GetBufferUART(&gBufUART_Main);
  buf[idx++] = len--;
  //принять остальные
  while (len) {
    if (!IsBufferEmptyUART(&gBufUART_Main)) {
      buf[idx++] = GetBufferUART(&gBufUART_Main);
      len--;
    }
    if (GetTimeSys() - tm > 30) {     //тайм-аут
          res_time_out=UART_RES_TIMEOUT;
          check = HAL_UART_Transmit_IT(&huart2, &res_time_out , 1);
      return(0);
    }
  }
  crc = Crc8(buf, buf[0]);
  ClearBufferUART(&gBufUART_Main);
  if (crc) {
  res_crc=UART_RES_CRC;
  HAL_UART_Transmit_IT(&huart2, &res_crc , 1);
    return(0);
  }
  return(buf);
}
/*
*****************************************************************************
  ожидание и прием пакета от CPU 1
  1-й байт длина данных
  1 задний CRC8
*****************************************************************************
```

```
*/
BYTE *GetDataCPU1 (void)
{
  static BYTE buf[SZ_MAX_UART];
  BYTE crc;
  DWORD tm;
  WORD idx = 0;
  WORD len;
  BYTE res_time_out, res_crc;
  if (CheckCPU() != CPU_2) {
    return(0);
  }
  //ждем 1-й байт
  if (IsBufferEmptyUART(&gBufUART_CPU1))
    return(0);
  tm = GetTimeSys();
  len = GetBufferUART(&gBufUART_CPU1);
  buf[idx++] = len--;
  //принять остальные
  while (len) {
    if (!IsBufferEmptyUART(&gBufUART_CPU1)) {
      buf[idx++] = GetBufferUART(&gBufUART_CPU1);
      len--;
    }
    if (GetTimeSys() - tm > 30) {      //тайм-аут
          res_time_out=UART_RES_TIMEOUT;
          HAL_UART_Transmit_IT(&huart2, &res_time_out , 1);
      return(0);
    }
  }
  crc = Crc8(buf, buf[0]);
  if (crc) {
  res_crc=UART_RES_CRC;
  HAL_UART_Transmit_IT(&huart2, &res_crc , 1);
    return(0);
  }
  return(buf);
}
/*
*****************************************************************************
  ожидание и прием num данных от CPU2 в буфер buf
  возращает:
  UART_RES_OK
  UART_RES_TIMEOUT
*****************************************************************************
*/
BYTE GetDataCPU2 (BYTE *buf, WORD num)
{
  DWORD tm;
```

```
  WORD idx = 0;
  if (CheckCPU() != CPU_1) {
    return(0);
  }
  tm = GetTimeSys();
  while (num) {
    if (!IsBufferEmptyUART(&gBufUART_CPU2)) {
      buf[idx++] = GetBufferUART(&gBufUART_CPU2);
      num--;
    }
    if (GetTimeSys() - tm > 50) {
      return(UART_RES_TIMEOUT);
    }
  }
  return(UART_RES_OK);
}
/*
*****************************************************************************
  передать 1 байт
*****************************************************************************
*/
void SendUART_Byte (UART_HandleTypeDef *uart, BYTE data)
{
HAL_UART_Transmit(uart, &data, 1, 100);
}
/*
*****************************************************************************
  передать массив данных
*****************************************************************************
*/
void SendUART_Data (UART_HandleTypeDef *uart, BYTE *buf, WORD num)
{
WORD i, n;
  n = 0;
  for (i = 0; i < num; i++) {
        HAL_UART_Transmit(uart, &buf[i], 1, 100);
    if (n++ == 20) {
      n = 0;
      RESET_CRASH();
    }
  }
}
/*
*****************************************************************************
  передать пакет данных в ПК
*****************************************************************************
*/
void SendMainUART_Packet (BYTE *buf, WORD sz)
{
```

```
  BYTE pkt[5500];
  if (CheckCPU() != CPU_1) {
    return;
  }
  pkt[0] = sz + 2;
  for (int i = 0; i < sz; i++)
    pkt[i + 1] = buf[i];
  pkt[sz + 1] = Crc8(pkt, sz + 1);
  SendUART_Data(UART_MAIN, pkt, sz + 2);
  ClearBufferUART(&gBufUART_Main);
}
/*
*****************************************************************************
  передать пакет данных CPU1==>CPU2
*****************************************************************************
*/
BYTE SendCPU2_Packet (BYTE *buf, WORD sz)
{
  DWORD tm, tm1;
  static BYTE pkt[200];
  if (CheckCPU() != CPU_1) {
    return(0);
  }
  pkt[0] = sz + 2;
  for (int i = 0; i < sz; i++)
    pkt[i + 1] = buf[i];
  pkt[sz + 1] = Crc8(pkt, sz + 1);
  ClearBufferUART(&gBufUART_CPU2);
  SendUART_Data(UART_CPU1_2, pkt, sz + 2);
  //ждать подтверждения от CPU2
  tm = GetTimeSys();
  while (IsBufferEmptyUART(&gBufUART_CPU2)) {
    tm1 = GetTimeSys();
    if (tm1 - tm > 30)      //тайм-аут
      return(UART_RES_TIMEOUT);
  }
  return(GetBufferUART(&gBufUART_CPU2));
}
```

программный код в файле «stm32f4xx_it.c»

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f4xx_it.h"
#include "uart_init.h"
#include "main_cpu_2.h"
/* Private includes ----------------------------------------------------------*/
/* External variables --------------------------------------------------------*/
extern UART_HandleTypeDef huart4;
extern UART_HandleTypeDef huart7;
extern UART_HandleTypeDef huart8;
```

```
extern UART_HandleTypeDef huart2;
extern UART_HandleTypeDef huart3;
/* Private variables ---------------------------------------------------------*/
/* USER CODE BEGIN PV */
volatile static DWORD TimeDelay;
volatile static DWORD TimeSys;
/* USER CODE END PV */
/**
 * @brief This function handles System tick timer.
 */
void SysTick_Handler(void)
{
 /* USER CODE BEGIN SysTick_IRQn 0 */
TimeSys++;
if (TimeDelay)
    TimeDelay--;
 /* USER CODE END SysTick_IRQn 0 */
 HAL_IncTick();
 /* USER CODE BEGIN SysTick_IRQn 1 */
 /* USER CODE END SysTick_IRQn 1 */
}
/**
 * @brief This function handles USART2 global interrupt.
 */
void USART2_IRQHandler(void)
{
 /* USER CODE BEGIN USART2_IRQn 0 */
if (CheckCPU()==CPU_1) {
            if ((USART2->SR & USART_SR_RXNE) != RESET){
                PutBufferUART((uint16_t)(USART2->DR & (uint16_t)0x00FF),
&gBufUART_CPU2);
            }
        }
        else {
            if ((USART2->SR & USART_SR_RXNE) != RESET){
                                PutBufferUART((uint16_t)(USART2->DR &
(uint16_t)0x00FF), &gBufUART_CPU2);
                            }
        }
 /* USER CODE END USART2_IRQn 0 */
 HAL_UART_IRQHandler(&huart2);
 /* USER CODE BEGIN USART2_IRQn 1 */
 /* USER CODE END USART2_IRQn 1 */
}
/**
 * @brief This function handles USART3 global interrupt.
 */
void USART3_IRQHandler(void)
{
```

```
  /* USER CODE BEGIN USART3_IRQn 0 */
 static BYTE data = 0xFF;
          if (CheckCPU() == CPU_1) {
            if ((USART3->SR & USART_SR_RXNE) != RESET) {
              PutBufferUART((uint16_t)(USART3->DR & (uint16_t)0x00FF),
&gBufUART_MMI_OUT);
            }
          }
          else {
            if ((USART3->SR & USART_SR_RXNE) != RESET) {
              data = (uint16_t)(USART3->DR & (uint16_t)0x00FF);
              PutBufferUART(data, &gBufUART_TX_2);
              while (HAL_UART_Transmit_IT(&huart2, &data, 1)!=HAL_BUSY);
              if (data == 0x00) {
           //       EndOfWrapTestRunning = FALSE;
              }
            }
          }
  /* USER CODE END USART3_IRQn 0 */
  HAL_UART_IRQHandler(&huart3);
  /* USER CODE BEGIN USART3_IRQn 1 */
  /* USER CODE END USART3_IRQn 1 */
 }
 /**
  * @brief This function handles UART4 global interrupt.
  */
 void UART4_IRQHandler(void)
 {
  /* USER CODE BEGIN UART4_IRQn 0 */
 static BYTE data = 0xFF;
          if ((UART4->SR & USART_SR_RXNE) != RESET) {
            data = (uint16_t)(UART4->DR & (uint16_t)0x00FF);
            PutBufferUART(data, &gBufUART_TX_3);
            while (HAL_UART_Transmit_IT(&huart2, &data, 1)!=HAL_BUSY);
            if (data == 0x00) {
          //      EndOfWrapTestRunning = FALSE;
            }
          }
  /* USER CODE END UART4_IRQn 0 */
  HAL_UART_IRQHandler(&huart4);
  /* USER CODE BEGIN UART4_IRQn 1 */
  /* USER CODE END UART4_IRQn 1 */
 }
 /**
  * @brief This function handles UART7 global interrupt.
  */
 void UART7_IRQHandler(void)
 {
  /* USER CODE BEGIN UART7_IRQn 0 */
```

```
	static BYTE data = 0xFF;
		if (CheckCPU() == CPU_1) {
		  if ((UART7->SR & USART_SR_RXNE) != RESET) {
		      data = (uint8_t)(UART7->DR & (uint8_t)0x00FF);
		      PutBufferUART(data, &gBufUART_Main);
		    //	PutBufferUART((uint16_t)(UART7->DR & (uint16_t)0x00FF), &gBufUART_Main);
		  }
		}
		/* else {
		  if ((UART7->SR & USART_SR_RXNE) != RESET) {
		    data = (uint16_t)(UART7->DR & (uint16_t)0x00FF);
		    PutBufferUART(data, &gBufUART_TX_4);
		    while (HAL_UART_Transmit_IT(&huart2, &data, 1)!=HAL_BUSY);
		    if (data == 0x00) {
		      EndOfWrapTestRunning = FALSE;
		    }
		  }
		}*/
	  /* USER CODE END UART7_IRQn 0 */
	  HAL_UART_IRQHandler(&huart7);
	  /* USER CODE BEGIN UART7_IRQn 1 */
	  /* USER CODE END UART7_IRQn 1 */
	}
	/**
	  * @brief This function handles UART8 global interrupt.
	  */
	void UART8_IRQHandler(void)
	{
	  /* USER CODE BEGIN UART8_IRQn 0 */
	static BYTE data = 0xFF;
		if (CheckCPU() == CPU_1) {
		  if ((UART8->SR & USART_SR_RXNE) != RESET) {
		    PutBufferUART((uint16_t)(UART8->DR & (uint16_t)0x00FF), &gBufUART_MMI_IN);
		  }
		}
		else {
		  if ((UART8->SR & USART_SR_RXNE) != RESET) {
		    data = (uint16_t)(UART8->DR & (uint16_t)0x00FF);
		    PutBufferUART(data, &gBufUART_TX_1);
		    while (HAL_UART_Transmit_IT(&huart2, &data, 1)!=HAL_BUSY);
		    if (data == 0x00) {
		  //	EndOfWrapTestRunning = FALSE;
		    }
		  }
		}
	  /* USER CODE END UART8_IRQn 0 */
	  HAL_UART_IRQHandler(&huart8);
```

```
  /* USER CODE BEGIN UART8_IRQn 1 */
  /* USER CODE END UART8_IRQn 1 */
}
/* USER CODE BEGIN 1 */
/*
*******************************************************************************
    прочитать счетчик тайм-аут
*******************************************************************************
*/
DWORD GetTimeSys (void)
{
  return(TimeSys);
}
```

программный код в файле «hw_config.h»

```
#if !defined(__HW_CONFIG_H__)
#define __HW_CONFIG_H__
#include "type_new.h"
//===== структура команд
typedef struct {
  BYTE     code;        // код команды
  void (*ptrFunc)(void);  // указатель на функцию обработки
} T_COMMAND;
typedef enum {
  CPU_NONE = 0,
  CPU_1 = 0x10,
  CPU_2
} T_NUM_CPU;
//CPU_1
#define UART_MAIN    UART7
#define UART_CPU1_2  USART2
#define UART_MMI_OUT USART3
#define UART_MMI_IN  UART7
//CPU_2
#define UART_CPU2_1  USART2
#define UART_TX_1    UART8
#define UART_TX_2    USART3
#define UART_TX_3    UART4
#define UART_TX_4    UART7
typedef struct {
  volatile BYTE buf[SIZE_BUF_UART];
  volatile WORD Count;
  volatile WORD Tail;
  volatile WORD Head;
} T_BUF_UART;
#endif
```