**УТВЕРЖДЕН**

**ЮКСУ.91264-01 12 01-УД**

Пакет поддержки программирования микроконтроллера

**(ППП МК)**

Руководство программиста

ЮКСУ.91264-01 33 01

**МН**

**Листов 30**

**2022**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| **Инв № подл** | **Подпись и дата** | **Взам инв №** | **Инв № дубл** | **Подпись и дата** |

**Аннотация**

Настоящий документ содержит руководящие указания для программиста по применению программы "Пакет поддержки программирования микроконтроллера" (ППП МК).

**Содержание**

1 Назначение и условия применения программы 4

2 Характеристика программы, входные и выходные данные. Сообщения 4

3 Создание прикладной программы. Пошаговое руководство 5

3.1 Описание демонстрационной задачи 5

3.2 Описание аппаратуры 5

3.3 План реализации проекта 8

3.4 Подготовка рабочего места 8

3.5 Создание нового проекта на базе примера 8

3.6 Шаг 1 – вывод в UART0 9

3.7 Шаг 2 – работа с I2C EEPROM 12

3.8 Шаг 3 – консоль 16

3.9 Шаг 4 – реализация команд eeget и eeset 22

# 1 Назначение и условия применения программы

* 1. Назначение программы и документа

Программа «Пакет поддержки программирования микроконтроллера» (далее – программа или ППП МК) предназначена для применения в качестве базового инструмента создания прикладных программ микроконтроллера К5500ВК018 (Комдив-МК) и программируемых логических контроллеров (ПЛК) на его основе.

Данное Руководство программиста иллюстрирует процесс разработки прикладной задачи. Представлено пошаговое руководство для программистов, приступающих к созданию прикладных программ для микроконтроллера К5500ВК018 (Комдив-МК) с применением ППП МК.

1.2 Условия применения

1.2.1 Необходимые для применения ППП МК аппаратные и программные средства описаны в документе Описание применения (ЮКСУ.91264-01 31 01).

1.2.2 Данное руководство составлено применительно к отладочной плате Багет-ПЛК1-01 (RBM\_C1K5500VK018).

# 2 Характеристика программы, входные и выходные данные. Сообщения

2.1 Программа ППП МК представляет собой базовое средство для разработки программ для микроконтроллера и ПЛК на его основе, позволяя создавать полноценные встраиваемые приложения. Компактность ППП МК и создаваемых целевых программ позволяет применять ее в устройствах без внешней оперативной памяти, с использованием только встроенного ОЗУ микроконтроллера объемом от нескольких десятков килобайт.

Программа и созданные на ее основе прикладные программы функционирует без использования среды какой-либо операционной системы (ОС).

Программа ориентирована на класс микроконтроллеров с одним микропроцессорным ядром и достаточной производительностью (десятки мегагерц частоты микропроцессорного ядра) и/или аппаратные средства распараллеливания (каналы DMA, контроллеры коммуникационных шин).

Программа поддерживает дисциплину кооперативной многозадачности.

2.2 Входных и выходных данных программа не имеет.

2.3 Сообщения программы отсутствуют.

# 3 Создание прикладной программы. Пошаговое руководство

## 3.1 Описание демонстрационной задачи

Требуется разработать редактор i2c EEPROM M24C32 (32-Kbit serial I²C bus EEPROM, STMicroelectronics), позволяющий просматривать и редактировать записанные в EEPROM данные.

Необходимо предусмотреть командный интерфейс пользователя, включающий несколько вводимых с консоли команд.

Далее приведена спецификация команд.

3.1.1 **eeget**

eeget <address> <count> - чтение <count> байт из EEPROM, начиная с адреса <address>. Данные выводятся в шестнадцатеричном виде через пробел с форматированием (переводом строки) после некоего числа байт (для примера взять 32).

3.1.2 **eeset**

eeset <address> <byte1> … <byteN> - запись байт из командной строки в EEPROM, начиная с адреса <address>. Данные допустимо указывать как в шестнадцатеричном виде (указывая префикс 0x), так и в десятичном.

3.1.3 **help**

help – вывод краткой информации по командам.

## 3.2 Описание аппаратуры

В качестве аппаратной части для целей настоящего руководства будет использована плата Багет-ПЛК1-01 (RBM\_C1K5500VK018) (рис. 1). В соответствии с конструкцией платы будут определены используемые в разработке внутренние устройства (последовательный порт, шина I2C, EEPROM), а также способ подключения и взаимодействия с платой.

Плата RBM\_C1K5500VK018 – отладочная плата, созданная и распространяемая НИИСИ РАН в помощь разработчикам программно-аппаратных решений на микроконтроллере Комдив-МК. Плата выпускается без корпуса, имеет соединительные интерфейсы для подключения внешних компонентов распространенных форм-факторов, а также набор готовых компонентов (светодиодов, микросхем ПЗУ и т.д.) расположенных непосредственно на плате и доступных для отладочных задач. Внешний вид платы приведен на рис. 1.

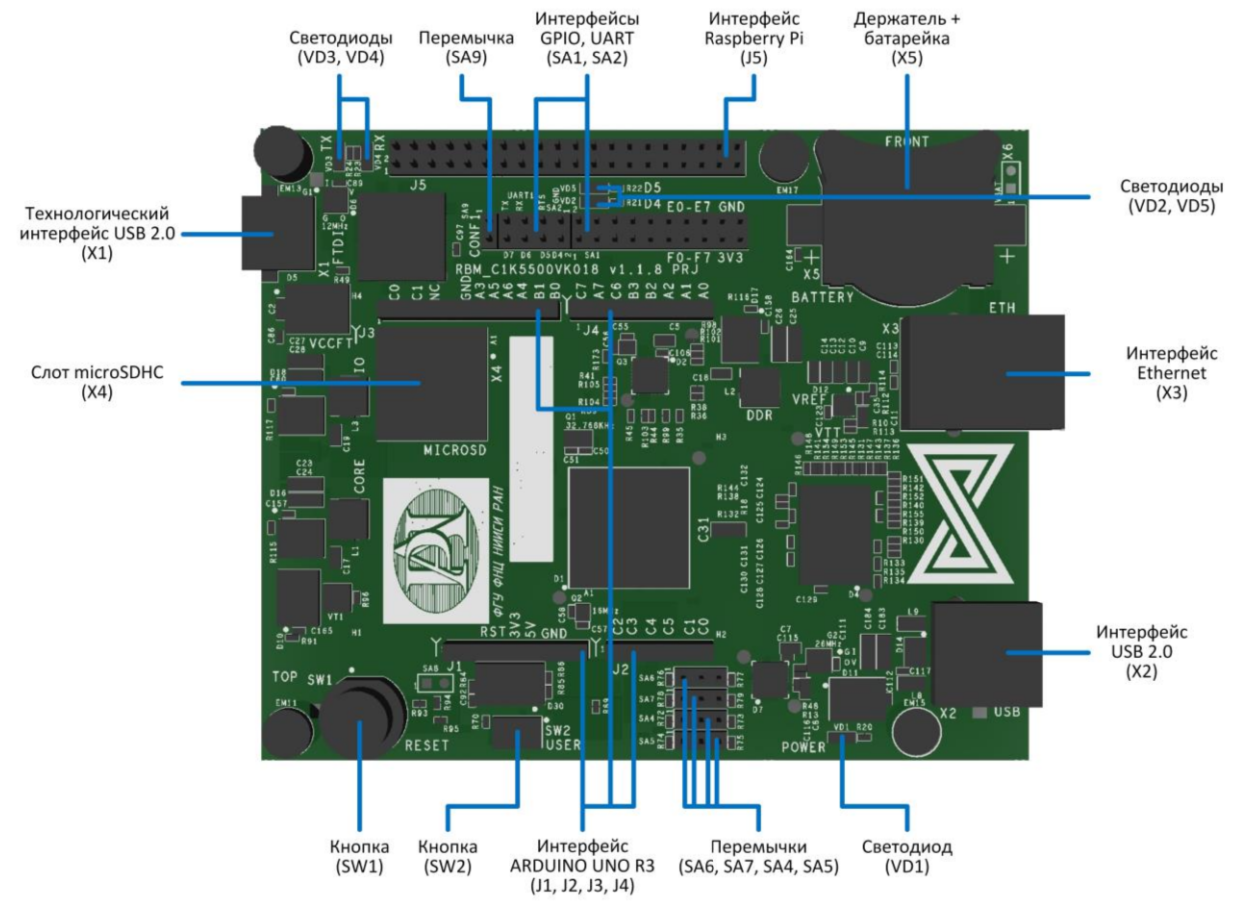


Рис. 1. Плата Багет-ПЛК1-01

Плата включает большое количество интерфейсов и компонентов, с их полной спецификацией можно ознакомиться в ее руководстве по эксплуатации (ЮКСУ.421457.002-01РЭ), здесь же будут приведены только выдержки, требующиеся в контексте реализуемого проекта.

В частности,

* Подключение платы к инструментальному ПК (рабочей станции) осуществляется через технологический USB порт, соединенный в свою очередь с контроллером FT2232H. Этот контроллер обеспечивает для инструментального ПК доступ к последовательному порту UART0 микроконтроллера Комдив-МК и к шине QSPI, к которой подключена ППЗУ, хранящая программное обеспечение микроконтроллера.
* На плате имеется микросхема ПЗУ m24c32, можно использовать ее для целей настоящего демонстрационного проекта.

На рис. 2 приведена выдержка из блок-схемы платы.

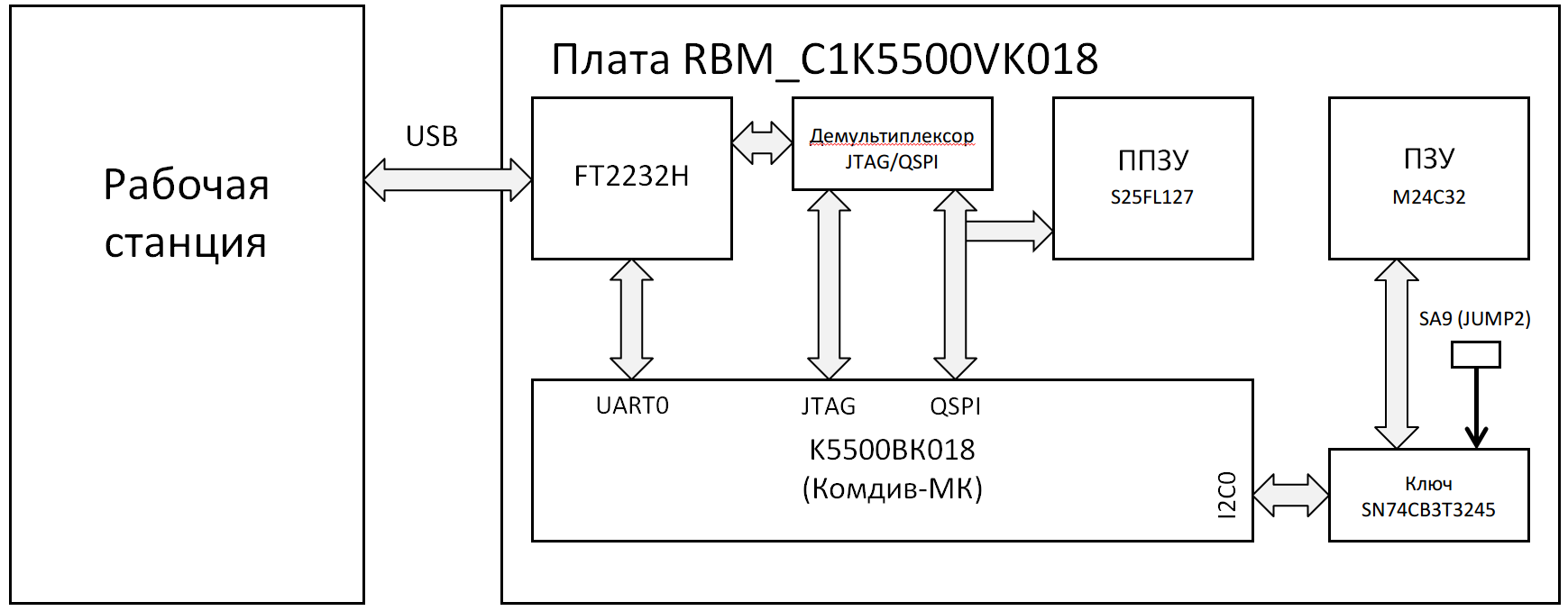


Рис. 2. Компоненты платы Багет-ПЛК1-01 и подключение к инструментальному ПК (рабочей станции)

Наличие контроллера FT2232H позволяет загружать программное обеспечение (в настоящем документе – сборку демонстрационного проекта) утилитой flashrom (см. Описание применения ППП МК ЮКСУ.91264-01 31 01) в QSPI ПЗУ S25FL127.

Также наличие данного контроллера и подключение к нему интерфейса последовательного порта UART0 позволяет использовать его для пользовательского интерфейса. Для инструментального ПК с ОС Linux для этого может быть использована программа **minicom**:

$ minicom -b 115200 -D /dev/ttyUSB1

(где /dev/ttyUSB1 – устройство в ОС Linux, соответствующее подключению платы).

На самой плате RBM\_C1K5500VK018 необходимо осуществить конфигурирование:

* Установить перемычку SA4 в положение, соответствующее использованию внутренней памяти микроконтроллера;
* Установить перемычку SA9 в положение CONF1, при котором ПЗУ m24c32 будет доступна микроконтроллеру на шине I2C0 (см. рис 2).

Несмотря на необходимость учета специфики аппаратуры, демонстрационный проект может быть реализован и на иной аппаратуре с Комдив-МК, или даже на ином микроконтроллере, поддерживаемом ППП МК. Далее по тексту будут даваться уточнения относительно параметров, относящихся к плате RBM\_C1K5500VK018 и возможных отличий при использовании других плат или схем подключения.

## 3.3 План реализации проекта

Проект будет разрабатываться на базе демонстрационного примера – программы для управления индикацией через I2C-GPIO-расширитель. Пример разобран в документе Описание применения (ЮКСУ.91264-01 31 01).

Последовательность шагов:

- подготовка рабочего места (установка компилятора, утилиты flashrom, скрипта для записи программы на плату, подключение платы);

- создание нового проекта на базе примера, компиляция;

- разработка программы вывода данных в UART0 (добавляем код, прошивка выводит Hello World на экран при старте);

- расширение возможностей программы: взаимодействие с I2C EEPROM (добавляем драйвер M24C32 и выводим содержимое EEPROM – допустим, первые 32 байта – на экран при старте);

- расширение возможностей программы: консоль для ввода команд (добавляем консоль и примитивный command processor, вводим команды и получаем на экран текстовое сообщение по любой команде);

- завершение реализации созданием полной программы с eeget, eeset и help;

- ограничения, выводы.

## 3.4 Подготовка рабочего места

- установить кросс-компилятор (например, из ЮКСУ.91264-01 12 02);

- установить программу записи во флеш-память (flashrom, например, из ЮКСУ.91264-01 12 02);

- установить ППП МК.

## 3.5 Создание нового проекта на базе примера

Создаем в каталоге **apps** подкаталог «**eeprom\_editor**» и копируем туда полностью содержимое подкаталога **demo\_pca**:

/

|-- apps

| |-- eeprom\_editor

| |-- src

| |-- komdiv\_board

| |-- main.c

| |-- config.h

| |-- Makefile

## 3.6 Шаг 1 – вывод в UART0

3.6.1 UART0 в Комдив-МК не имеет аппаратной возможности работать с DMA. Для разрабатываемой простой демонстрационной программы UART может работать в самом простом режиме, без использования прерываний. Для этого достаточно добавить в функцию init\_board() (реализована в файле **main.c**) инициализацию в следующем виде:

|  |
| --- |
| kmk\_serial\_init(0, KMK\_ADDR(UART0\_BASE), 0, 0, 0, 0, 0, 0, 0, 0);  \_\_system\_serial\_configure(\_\_system\_serial\_get(0), 115200, 8, 0, 1); |

Объявление функции kmk\_serial\_init() находится в файле **arch/kmk/kmk\_common.h**, а объявление \_\_system\_serial\_configure() – в файле **arch/arch.h**. Оба эти заголовочных файла уже подключены в **main.c**.

Чтобы данный код можно было собрать, также потребуется добавить в **config.h** определение:

|  |
| --- |
| #define \_\_ARCH\_WITH\_SERIAL 1 |

Далее можно выполнить **make** и, если все сделано правильно, проект успешно соберется (будут созданы файлы форматов .bin и .elf). Скрипты сборки ППП МК самостоятельно добавят в сборку драйвер последовательного порта kmk\_serial и необходимые ему библиотечные функции.

3.6.2 Для иллюстративности добавим в проект использование прерываний. Прерывания будут задействованы только для передаваемых данных (TX), так как для принимаемых данных (RX) потребуется более сложная схема с установкой коллбэка, а это выходит за рамки текущей разрабатываемой программы.

*Примечание*: **Callback** (англ. *call* — вызов, англ. *back* — обратный) или **функция обратного вызова** в программировании — передача исполняемого кода в качестве одного из параметров другого кода. Обратный вызов позволяет в функции исполнять код, который задаётся в аргументах при её вызове. Этот код может быть определён в других контекстах программного кода и быть недоступным для прямого вызова из этой функции

В файле **main.c**:

1. Добавляем объявление функции линии аппаратного прерывания (ее реализация находится в файле **kmk\_irqs.S**; она сохраняет в стеке все регистры и тем самым обеспечивает возможность вызвать обработчик прерывания на Си)

void kmk\_my\_int2(void);

2. В самое начало функции init\_board добавляем установку вектора прерываний ни линию INT2. Можно выбрать любую линию от 0 до 5, но т.к. в примере уже есть реализация kmk\_my\_int2, используем INT2:

kmk\_set\_int\_vector(2, &kmk\_my\_int2);

После нее добавляем установку обработчика последовательного порта на прерывание от источника IRQ\_UART0:

int\_connect(IRQ\_UART0, kmk\_serial\_irq, 0);

Функция int\_connect осуществляет помещение обработчика от прерывания от определенного источника (IRQ – 48 источников прерываний для микроконтроллера Комдив-МК) в таблицу маршрутизатора прерываний. В третьем параметре int\_connect указывается значение, которое будет передаваться в обработчик прерываний. Сигнатура его функции определена как

|  |
| --- |
| void(\*routine)(unsigned char) |

и передаваемый в нее параметр позволяет использовать один и тот же код обработчика для множества однотипных устройств.

В данном случае на запрос прерывания IRQ\_UART0 будет вызываться обработчик kmk\_serial\_irq, которому будет передаваться значение 0, означающее номер последовательного порта.

3. Меняем ранее добавленную инициализацию UART0 на следующую:

kmk\_serial\_init(0, KMK\_ADDR(UART0\_BASE), KMK\_SERIAL\_FLAG\_IRQ\_TX,  
 IRQ\_UART0, 0, 0, 0, 0, 0, 0);

\_\_system\_serial\_configure(\_\_system\_serial\_get(0), 115200, 8, 0, 1);

kmk\_init\_int\_line(IRQ\_UART0, 2, 0);

Здесь в kmk\_serial\_init добавлен флаг «использовать прерывание в процессе передачи данных» и указан номер запроса прерывания – IRQ\_UART0. Ниже добавлен вызов функции kmk\_init\_int\_line, активирующей в контроллере прерываний линию IRQ\_UART0 и указывающей контроллеру прерываний формировать для таких запросов прерывание процессора INT2. Третий параметр – 0 – определяет приоритет (0 – низший приоритет) для таких запросов.

Итого, для настройки прерываний было сделано

- установка на обработчик прерывания процессора INT2 функции kmk\_my\_int2, сохраняющей регистры и передающей управление в маршрутизатор прерываний;

- в таблицу маршрутизатора прерываний на запрос IRQ\_UART0 помещен обработчик от драйвера последовательного порта kmk\_serial\_irq, который будет вызываться с параметром 0 (номером UART);

- запросы IRQ\_UART0 активированы в контроллере прерываний и сопоставлены с линией INT2 процессора.

Т.е. **IRQ\_UART0** ---> **kmk\_serial\_irq(0)**;

4. В файл **app.c** в список подключаемых хедеров добавляем

#include "../../../system/serial.h"

#include <string.h>

А в конец функции app\_initialize() –

char \*pcHelloWorld = "Hello, world!\r\n";

void \*pS = \_\_system\_serial\_get(0);

unsigned uLen = strlen(pcHelloWorld);

if (\_\_system\_serial\_write(pS, pcHelloWorld, &uLen) == 1) {

while(!\_\_system\_serial\_ready(pS, SERIAL\_OPERATION\_TX)) {};

}

Собираем образ прошивки командой **make**, прошиваем в плату, и перезагружаем.

Если в подключенном терминале присутствует

|  |
| --- |
| **Hello, world!** |

значит, UART0 успешно проинициализирован и работоспособен.

## 3.7 Шаг 2 – работа с I2C EEPROM

Следующим шагом в реализации проекта будет обращение к EEPROM. На рис. 2 приведен фрагмент схемы платы Багет-ПЛК1-01 (RBM\_C1K5500VK018), отображающий размещенную на плате микросхему EEPROM:

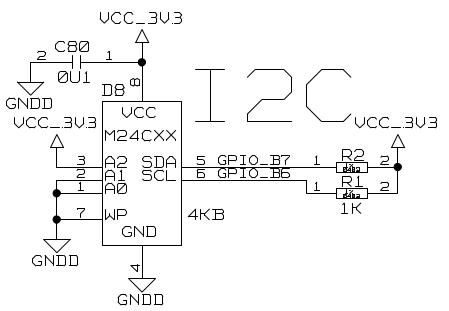


Рис. 3. Фрагмент схемы платы

Для выполняемого проекта здесь важна следующая информация:

* подключение выполнено на выводы B6/B7, в соответствии со спецификацией микроконтроллера Комдив-МК, это I2C0;
* микросхема EEPROM здесь (из спецификации платы RBM\_C1K5500VK018) - M24C32. В соответствии с ее спецификацией подключение контактов A0-A2 устанавливает адрес 0x54 (7 бит);
* линия WP EEPROM соединена с землей, что разрешает операции записи в EEPROM.

**Внимание! Для платы RBM\_C1K5500VK018 работа с I2C0 требует установки соединителя (джампера) SA9 (CONF1)! См. Указания по применению платы.**

*Примечание: Если работа над проектом ведется на другой плате или подключение EEPROM выполнено не так, как на рисунке выше, необходимо будет вносить правки в приводимый ниже код; по возможности для этого будут даваться пояснения.*

Проект, взятый за основу (пример работы с I2C-GPIO расширителем), уже содержит код для инициализации I2C0 в файле **main.c**:

// I2C 0

GPIO\_SetPinMode(GPIOB\_BASE, 6, GPIO\_MODE\_ALT1);

GPIO\_SetPinMode(GPIOB\_BASE, 7, GPIO\_MODE\_ALT1);

kmk\_i2c\_init(0, KMK\_ADDR(I2C0\_BASE), 400000);

В случае, если подключение осуществлено на другой канал I2C, необходимо будет отредактировать номера линий GPIO и проинициализировать другой контроллер I2C, например, I2C1.

Добавим в проект драйвер для микросхем EEPROM M24x – m24c04. (Код драйвера входит в состав поставки ППП МК).

Файлы **m24c04.c** и **m24c04.h** размещаются в подкаталоге **devices**:

/

|-- apps

| |-- eeprom\_editor

| |-- src

| |-- komdiv\_board

| |-- main.c

| |-- config.h

| |-- Makefile

|

|-- arch

| |--kmk

|

|-- **devices**

| |--m24c04.c

| |--m24c04.h

|

|-- system

И не требуют отдельного подключения к проекту, скрипты сборки ППП МК подключат необходимый код автоматически.

**Для информации:** драйвер m24c04 рассчитан на работу в условиях общего доступа к шине I2C с другими компонентами, поэтому содержит набор вызовов для получения доступа к шине и ее освобождению. Реализация механизма предоставления доступа в таком случае производится в отдельном компоненте – менеджере ресурсов. Для текущего проекта, где на шине находится только EEPROM, менеджер ресурсов не требуется, однако его заголовочный файл **core/res\_manager.h** требуется для использования структуры данных **i2c\_res\_ctx**.

Далее добавляем в проект непосредственно прикладной код. Для этого создадим в каталоге **app** следующие файлы:

**editor.h:**

#ifndef \_\_EDITOR\_PROJECT\_H\_\_

#define \_\_EDITOR\_PROJECT\_H\_\_

unsigned char dump\_eeprom(void);

#endif /\* \_\_EDITOR\_PROJECT\_H\_\_ \*/

Здесь мы объявляем одну функцию, которая осуществит вывод в демонстрационных целях.

**editor.c**:

#include "editor.h"

#include "../../../arch/arch.h"

#include "../../../core/res\_manager.h"

#include "../../../system/serial.h"

#include "../../../devices/m24c04.h"

#include <string.h>

#define EEPROM\_CONFIG\_ADDR 0x54

#define EEPROM\_CONFIG\_ADDR\_WIDTH 16

// dump data in HEX with max 32 bytes per column

static void out\_buffer(unsigned char \*pbBuf, unsigned uLen)

{

unsigned char pbLNBuf[4] = { '\r', '\n', 0, 0 }; // \r\nXX

unsigned char pbSPBuf[3] = { ' ', 0, 0 }; // \_XX

unsigned char ucMaxLen = 32, ucFirst = 1, \*pStart, \*pValue;

unsigned x, uSendLen;

void \*pS = \_\_system\_serial\_get(0);

for(x = 0; x < uLen; x++) {

if (ucFirst) {

pValue = pbLNBuf + 2;

uSendLen = (x)? 4 : 2;

pStart = (x)? pbLNBuf : pbLNBuf + 2;

ucFirst = 0;

} else {

pValue = pbSPBuf + 1;

uSendLen = 3;

pStart = pbSPBuf;

ucFirst = ((x % ucMaxLen) == (ucMaxLen - 1))? 1 : 0;

}

pValue[0] = (pbBuf[x] >> 4);

pValue[0] += (pValue[0] > 9)? 87 : 48;

pValue[1] = (pbBuf[x] & 0xF);

pValue[1] += (pValue[1] > 9)? 87 : 48;

if (\_\_system\_serial\_write(pS, pStart, &uSendLen) == 1) {

while(!\_\_system\_serial\_ready(pS, SERIAL\_OPERATION\_TX)) {};

}

}

}

unsigned char dump\_eeprom(void)

{

res\_i2c\_ctx i2c\_res;

eeprom\_m24\_ctx e\_ctx;

eeprom\_m24\_io e\_io;

unsigned char pbBuf[64], ucRc;

memset(pbBuf, 0, sizeof(pbBuf));

i2c\_res.ucChannel = 0;

i2c\_res.pResI2C = i2c\_get(0);

eeprom\_m24\_prepare\_context(&e\_ctx, EEPROM\_CONFIG\_ADDR, &i2c\_res, EEPROM\_CONFIG\_ADDR\_WIDTH, 0);

eeprom\_m24\_prepare\_io(&e\_io, pbBuf, sizeof(pbBuf), 0);

if ((ucRc = eeprom\_m24\_start\_read(&e\_ctx, &e\_io)) == 0) {

while((ucRc = eeprom\_m24\_poll(&e\_ctx)) == 0) { \_\_asm("nop"); };

if (ucRc == 1) {

out\_buffer(pbBuf, sizeof(pbBuf));

return 1;

}

}

return 0;

}

Здесь реализованы две функции: out\_buffer() осуществляет форматированный побайтовый вывод на экран содержимого переданного в нее буфера, а dump\_eeprom() осуществляет чтение первых 64 байт из EEPROM и выводит их при помощи out\_buffer. В случае успеха dump\_eeprom возвращает 1.

Также следует обратить внимание на объявляемые в начале файла

#define EEPROM\_CONFIG\_ADDR 0x54

#define EEPROM\_CONFIG\_ADDR\_WIDTH 16

Как было указано выше, для платы RBM\_C1K5500VK018 адрес EEPROM на шине I2C0 – 0x54, также из спецификации M24C32 определяется, что адресация ячеек памяти внутри EEPROM осуществляется двубайтным (16-битным) адресом. Эти данные необходимы для инициализации драйвера.

**editor.c** необходимо добавить в **Makefile**

SRC += app/editor.c

А в **app.c** – в начало файла добавить

#include "editor.h"

А в конец функции app\_initialize

if (dump\_eeprom() != 1) {

unsigned uLen = 7;

if (\_\_system\_serial\_write(pS, "ERROR\r\n", &uLen) == 1) {

while(!\_\_system\_serial\_ready(pS, SERIAL\_OPERATION\_TX)) {};

}

}

Здесь видно, что если функция dump\_eeprom завершится с ошибкой, в терминал будет выдано ERROR.

Собираем образ прошивки командой **make**, прошиваем в плату, и перезагружаем.

Если в подключенном терминале присутствует

|  |
| --- |
| **Hello, world!**  **ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff**  **ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff** |

значит, микросхема EEPROM на плате успешно доступна и работоспособна. (ff – значения, которые содержит новая микросхема EEPROM. Если в вашем случае в нее что-то ранее записывалось, то на экране будут присутствовать другие значения)

## 3.8 Шаг 3 – консоль

Следующим шагом будет расширение работы с UART и организация не только вывода, но и ввода – консоли, воспринимающей несколько команд.

Подключим в проект простую консоль – **console.c/h**. Ее код поставляется в ППП МК.

Консоль, реализованная в **console.c**, предназначена для работы в системах с кооперативной многозадачностью, в том числе имеет в интерфейсе функцию console\_tick для получения квантов времени для выполнения внутренних процессов.

Файлы **console.c** и **console.h** размещены в подкаталоге **system** в каталоге проекта. Консоль работает с последовательными портами при помощи универсальных команд, реализация которых находится в файле **serial.c**, располагающимся там же:

/

|-- apps

| |-- eeprom\_editor

| |-- src

| |-- komdiv\_board

| |-- main.c

| |-- config.h

| |-- Makefile

|

|-- arch

| |--kmk

|

|-- **system**

| |--console.c

| |--console.h

| |- serial.c

| |--serial.h

Эти файлы не требуют отдельного подключения к проекту, скрипты сборки ППП МК подключат необходимый код автоматически.

Алгоритм работы консоли следующий: она обеспечивает отправку данных в подключенный к ней последовательный порт (с использованием промежуточного буфера), а также отслеживает вводимые через этот последовательный порт символы, предоставляя простые средства редактирования (удаление ошибочно введенных символов клавишей Backspace). Когда в потоке символов определяется символ ввода (Enter), введенные данные передаются в обработчик.

Соответственно, необходимо создать компонент, который будет принимать данные с консоли и запускать команды. Реализуем эту задачу в файле **editor.c** посредством двух функций: непосредственно обработчика и функции, которая будет принимать кванты времени для выполнения запущенной задачи (так как в перспективе задача требует работы с EEPROM, которая предполагает отложенные действия).

Дополнение файла **editor.h** декларациями новых функций (и подключением **console.h**, так как функция-обработчик принимает ссылку на консоль, с которой осуществлен ввод):

#ifndef \_\_EDITOR\_PROJECT\_H\_\_

#define \_\_EDITOR\_PROJECT\_H\_\_

#include "../../../system/console.h"

int editor\_get\_command(void \*pCompContext,

console\_ctx \*pConsole,

unsigned char \*pbBuffer,

unsigned uSize);

int editor\_tick(void);

unsigned char dump\_eeprom(void);

#endif /\* \_\_EDITOR\_PROJECT\_H\_\_ \*/

На данном этапе обеспечим обработку команд следующим образом:

**help** – выводим строку помощи («Commands: help, eeget, eeset»)

**eeget** – выводим строку «EEGET invoked!»

**eeset** – выводим строку «EESET invoked!»

На все иные команды (в том числе пустую строку) выводим «Unrecognized command. Use "help" to check.»

Код, добавляемый в **editor.c**:

typedef struct process\_ctx\_\_ {

unsigned char ucProcessing;

unsigned char ucState;

console\_ctx \*pConsole;

unsigned char \*pbCurrentOut;

unsigned uCurrentOutLen;

} process\_ctx;

#define PROCESSING\_POSTMESSAGE 1

#define PROCESSING\_EEGET 2

#define PROCESSING\_EESET 3

static process\_ctx gctx = { .ucProcessing = 0, .ucState = 0,

.pConsole = 0, .pbCurrentOut = 0,

.uCurrentOutLen = 0 };

const char \*pcMsgBadCommand =

"Unrecognized command. Use \"help\" to check.\r\n";

const char \*pcMsgHelp = "Commands: help, eeget, eeset\r\n";

const char \*pcRN = "\r\n";

// forwards

static int is\_space(char c);

static int extract\_token(unsigned char \*pbBuffer,

unsigned short usLen,

unsigned short \*pusStart,

unsigned short \*pusFinish);

static void one\_message(process\_ctx \*pCtx, const char \*pcMessage);

static int command\_eeget(process\_ctx \*pCtx, unsigned char \*pbBuffer,

unsigned short usLen);

static int command\_eeget\_cont(process\_ctx \*pCtx);

static int command\_eeset(process\_ctx \*pCtx, unsigned char \*pbBuffer,

unsigned short usLen);

static int command\_eeset\_cont(process\_ctx \*pCtx);

static int is\_space(char c)

{

return ((c == ' ') || (c == '\t'));

}

static int extract\_token(unsigned char \*pbBuffer,

unsigned short usLen,

unsigned short \*pusStart,

unsigned short \*pusFinish)

{

unsigned short u;

for(u = 0; u < usLen; u++) {

if (!is\_space((char)(\*(pbBuffer + u)))) {

\*pusStart = u;

for(; u < usLen; u++) {

if (is\_space((char)(\*(pbBuffer + u)))) {

\*pusFinish = u;

return 1;

}

}

\*pusFinish = usLen;

return 1;

}

}

return 0;

}

static int check\_token\_by\_options(unsigned char \*pbToken,

unsigned short usLen,

char \*\*ppOptions,

unsigned char ucOptionsCount,

unsigned char \*pusResult)

{

unsigned char x, y;

for(x = 0; x < ucOptionsCount; x++) {

unsigned char \*pB = (unsigned char \*)(ppOptions[x]);

for(y = 0; y < usLen; y++) {

if (pB[y] != pbToken[y]) break;

}

if ((y == usLen) && (pB[y] == 0)) {

\*pusResult = x;

return 1;

}

}

return 0;

}

static void one\_message(process\_ctx \*pCtx, const char \*pcMessage)

{

pCtx->ucProcessing = PROCESSING\_POSTMESSAGE;

pCtx->ucState = 1;

pCtx->pbCurrentOut = (unsigned char \*)pcMessage;

pCtx->uCurrentOutLen = strlen(pcMessage);

}

static int command\_eeget(process\_ctx \*pCtx, unsigned char \*pbBuffer,

unsigned short usLen)

{

one\_message(pCtx, "EEGET invoked!\r\n");

return 1;

}

static int command\_eeget\_cont(process\_ctx \*pCtx)

{

return 1;

}

static int command\_eeset(process\_ctx \*pCtx, unsigned char \*pbBuffer,

unsigned short usLen)

{

one\_message(pCtx, "EESET invoked!\r\n");

return 1;

}

static int command\_eeset\_cont(process\_ctx \*pCtx)

{

return 1;

}

int editor\_get\_command(void \*pCompContext, console\_ctx \*pConsole,

unsigned char \*pbBuffer, unsigned uSize)

{

int rc = 1;

unsigned char ucOption;

unsigned short usS, usF = 0;

char \*ppCommands[] = { "help", "eeset", "eeget" };

process\_ctx \*pCtx = &gctx;

if (pCtx->ucProcessing) {

return 0; // busy

} else {

gctx.pConsole = pConsole;

if ((!extract\_token(pbBuffer, uSize, &usS, &usF)) ||

(!check\_token\_by\_options(pbBuffer + usS, usF - usS, ppCommands, (sizeof(ppCommands) / sizeof(char \*)), &ucOption))) {

one\_message(pCtx, pcMsgBadCommand);

return 1;

}

switch(ucOption) {

case 0: one\_message(pCtx, pcMsgHelp); break;

case 1: rc = command\_eeset(pCtx, pbBuffer + usF, uSize - usF); break;

case 2: rc = command\_eeget(pCtx, pbBuffer + usF, uSize - usF); break;

}

if (rc == 0) rc = 1;

}

return rc;

}

int editor\_tick(void)

{

int rc = 1;

process\_ctx \*pCtx = &gctx;

switch(pCtx->ucProcessing) {

case PROCESSING\_POSTMESSAGE:

// task means completed when response is pushed to console (of if is is just impossible)

if ((!(pCtx->pConsole)) || (\_\_console\_print\_pb(pCtx->pConsole, pCtx->pbCurrentOut, pCtx->uCurrentOutLen))) pCtx->ucProcessing = 0;

break;

case PROCESSING\_EEGET:

rc = command\_eeget\_cont(pCtx);

break;

case PROCESSING\_EESET:

rc = command\_eeset\_cont(pCtx);

break;

default: rc = 0;

}

return rc;

}

В данном коде осуществляется:

1 Описание структуры-контекста выполняемой команды: контекст предполагает выполнение «вывода в консоль текстовой строки» (PROCESSING\_POSTMESSAGE) и команды PROCESSING\_EEGET и PROCESSING\_EESET – пока они будут делать то же самое – выводить простые строки, но сразу выделим их, чтобы на следующем этапе добавить в них нужные обработчики.

2 Разбор строки, переданной в функцию editor\_get\_command с заполнением соответствующим образом полей структуры контекста.

3 Выполнение команд из функции editor\_tick, предназначенной для получения временных квантов.

И, наконец, добавление вызовов реализованных выше функций (а также консоли) в файл **app.c**:

В начало файла:

static console\_ctx cs;

В функцию app\_initialize:

\_\_console\_init(&cs, editor\_get\_command, 0);

\_\_console\_attach\_serial(&cs, \_\_system\_serial\_get(0));

(вызов dump\_eeprom, добавленный туда на предыдущем этапе, больше не используется и его можно убрать).

В функцию app\_cycle\_tick:

\_\_console\_tick(&cs);

editor\_tick();

Собираем образ прошивки командой make, прошиваем в плату, и перезагружаем.

Если все реализовано правильно… вводим команды через консоль, получаем ответы:

**help**

**Commands: help, eeget, eeset**

**eeget**

**EEGET invoked!**

**eegett**

**Unrecognized command. Use "help" to check.**

**eeset**

**EESET invoked!**

**Unrecognized command. Use "help" to check.**

## 3.9 Шаг 4 – реализация команд eeget и eeset

Дальнейшие шаги в принципе уже не являются специфичными для микроконтроллера КомдивМК, но в них будет показана реализация компонента, ориентированного на выполнение в режиме кооперативной многозадачности.

Все изменения будут производиться только в файле **editor.c**.

Итак, для завершения проекта необходимо:

- реализовать парсеры команд, извлекающие из строки параметры в соответствии с заданием;

- реализовать конечные автоматы для работы с драйвером EEPROM.

Также необходимо будет доработать реализованную на шаге 2 функцию вывода буфера в консоль: во-первых, она не должна содержать ожидания в цикле, во-вторых, вывод будет осуществляться через интерфейс консоли, а не напрямую в последовательный порт.

static unsigned out\_buffer\_to\_console(console\_ctx \*pConsole,

unsigned char \*pbBuf,

unsigned uPos,

unsigned uLen)

{

unsigned char pbOutBuffer[32]; // will push up to 8 symbols per chunk

unsigned char \*p = pbOutBuffer;

unsigned char ucMaxLen = 32;

unsigned x;

pbBuf += uPos;

for(x = 0; x < 8; x++) {

if ((uPos + x) == uLen) break;

\*p = (pbBuf[x] >> 4); \*p += (\*p > 9)? 87 : 48; p++;

\*p = (pbBuf[x] & 0xF); \*p += (\*p > 9)? 87 : 48; p++;

if (((uPos + x + 1) == uLen) || (((uPos + x) % ucMaxLen) == (ucMaxLen - 1))) {

\*p = '\r'; p++;

\*p = '\n'; p++;

} else {

\*p = ' '; p++;

}

}

return ((x > 0) && (\_\_console\_print\_pb(pConsole, pbOutBuffer, p - pbOutBuffer) == 1))? x : 0;

}

Функция формирует шестнадцатеричное представление байтов в буфере pbOutBuffer и передает его в консоль. Консоль содержит внутренний буфер, куда функция \_\_console\_print\_pb осуществляет копирование. Если места в буфере нет, функция возвратит 0 и операцию можно будет попробовать повторить позже.

Через каждые 32 байта осуществляется добавление в вывод символов «\r\n» для перевода строки.

Отредактируем структуру контекста командного обработчика и добавим объявления новых функций:

#define COMP\_PROC\_BUFFER\_SIZE 255

typedef struct process\_ctx\_\_ {

unsigned char ucProcessing;

unsigned char ucState;

console\_ctx \*pConsole;

unsigned char \*pbCurrentOut;

unsigned uCurrentOutLen;

unsigned long ulAddress;

unsigned short usCount;

unsigned short usPrinted;

unsigned char pbBuffer[COMP\_PROC\_BUFFER\_SIZE];

res\_i2c\_ctx i2c\_res;

eeprom\_m24\_ctx e\_ctx;

eeprom\_m24\_io e\_io;

} process\_ctx;

#define PROCESSING\_POSTMESSAGE 1

#define PROCESSING\_EEGET 2

#define PROCESSING\_EESET 3

static process\_ctx gctx = { .ucProcessing = 0, .ucState = 0,

.pConsole = 0, .pbCurrentOut = 0,

.uCurrentOutLen = 0, .ulAddress = 0,

.usCount = 0, .usPrinted = 0 };

const char \*pcMsgBadCommand =

"Unrecognized command. Use \"help\" to check.\r\n";

const char \*pcMsgHelp = "Commands: help, eeget, eeset\r\n";

const char \*pcMsgError = "ERROR\r\n";

char pcErrorExtractArgument[] = "Error extract argument ";

char pcBadArgument[] = "Bad argument ";

const char \*pcRN = "\r\n";

// forwards

static int is\_space(char c);

static int extract\_token(unsigned char \*pbBuffer,

unsigned short usLen,

unsigned short \*pusStart,

unsigned short \*pusFinish);

static int check\_token\_by\_options(unsigned char \*pbToken,

unsigned short usLen,

char \*\*ppOptions,

unsigned char ucOptionsCount,

unsigned char \*pusResult);

static int check\_token\_digit(unsigned char \*pbToken,

unsigned short usLen,

long \*plResult);

static int error\_extract\_argument(process\_ctx \*pCtx, unsigned char ucArg);

static int bad\_argument(process\_ctx \*pCtx, unsigned char ucArg);

И их реализации: разбор извлеченного из строки параметра как числа, вывод числа в десятичном виде (понадобится в сообщениях об ошибках) и сами сообщения об ошибках: недостаточность параметров и ошибка в параметре:

static int check\_token\_digit(unsigned char \*pbToken, unsigned short usLen, long \*plResult)

{

unsigned char x, state = 0, minus = 0;

long l = 0;

for(x = 0; x < usLen; x++) {

unsigned char c = pbToken[x];

switch(state) {

case 0: // first symbol. "-", "0x" etc are vaild here

if (c == '-') {

minus = 1; state = 10; break;

} else

if (c == '0') {

state = 1; break;

} else

if ((c > '0') && (c <= '9')) {

l = (c - 48); state = 10; break;

}

return 0;

case 1: /\* 'x' means we will parse hexadecimal, digit means we're parsing just decimal \*/

if (c == 'x') {

state = 16; break;

} else

if ((c >= '0') && (c <= '9')) {

l = (c - 48); state = 10; break;

}

return 0;

case 10: // parsing decimal

if ((c >= '0') && (c <= '9')) {

l = (l \* 10) + (long)(c - 48); break;

}

return 0;

case 16: // parsing hexadecimal

if ((c >= '0') && (c <= '9')) {

l = (l << 4) + (long)(c - 48); break;

} else

if ((c >= 'A') && (c <= 'F')) {

l = (l << 4) + (long)(c - 55); break;

} else

if ((c >= 'a') && (c <= 'f')) {

l = (l << 4) + (long)(c - 87); break;

}

return 0;

}

}

\*plResult = (minus)? (-l) : l;

return 1;

}

static unsigned char dump\_decimal(char \*pBuf, unsigned char ucValue)

{

unsigned char ucAmount = 1;

if (ucValue >= 100) {

\*pBuf = (ucValue / 100) + 48;

pBuf++; ucAmount++; ucValue %= 100;

}

if (ucValue >= 10) {

\*pBuf = (ucValue / 10) + 48;

pBuf++; ucAmount++; ucValue %= 10;

}

\*pBuf = ucValue + 48;

return ucAmount;

}

static int error\_extract\_argument(process\_ctx \*pCtx, unsigned char ucArg)

{

unsigned uSize = sizeof(pcErrorExtractArgument) - 5;

uSize += dump\_decimal(pcErrorExtractArgument + uSize, ucArg);

pcErrorExtractArgument[uSize++] = '\r';

pcErrorExtractArgument[uSize++] = '\n';

pCtx->ucProcessing = PROCESSING\_POSTMESSAGE;

pCtx->pbCurrentOut = pcErrorExtractArgument;

pCtx->uCurrentOutLen = uSize;

return 1;

}

static int bad\_argument(process\_ctx \*pCtx, unsigned char ucArg)

{

unsigned uSize = sizeof(pcBadArgument) - 5;

uSize += dump\_decimal(pcBadArgument + uSize, ucArg);

pcErrorExtractArgument[uSize++] = '\r';

pcErrorExtractArgument[uSize++] = '\n';

pCtx->ucProcessing = PROCESSING\_POSTMESSAGE;

pCtx->pbCurrentOut = pcErrorExtractArgument;

pCtx->uCurrentOutLen = uSize;

return 1;

}

#define TOKENIZE\_DIGIT(lVal, ucArgNum) \

pbP += usF; if (usF > usL) return error\_extract\_argument(pCtx, ucArgNum); \

usL -= usF; \

if (!extract\_token(pbP, usL, &usS, &usF)) return error\_extract\_argument(pCtx, ucArgNum); \

if (!check\_token\_digit(pbP + usS, usF - usS, &lVal)) return error\_extract\_argument(pCtx, ucArgNum)

Далее остается только реализовать команды в виде конечных автоматов.

**eeget**:

static int command\_eeget(process\_ctx \*pCtx, unsigned char \*pbBuffer,

unsigned short usLen)

{

long lTmp;

unsigned long ulAddress;

unsigned uCount;

unsigned char \*pbP = pbBuffer;

unsigned short usL = usLen, usS, usF = 0;

TOKENIZE\_DIGIT(lTmp, 1);

if (lTmp < 0) return bad\_argument(pCtx, 1);

ulAddress = lTmp;

TOKENIZE\_DIGIT(lTmp, 2);

if ((lTmp < 0) || (lTmp > 65535)) return bad\_argument(pCtx, 2);

uCount = lTmp;

pCtx->ucProcessing = PROCESSING\_EEGET;

pCtx->ucState = 0;

pCtx->ulAddress = ulAddress;

pCtx->usCount = uCount;

return command\_eeget\_cont(pCtx);

}

static int command\_eeget\_cont(process\_ctx \*pCtx)

{

unsigned char ucRc;

unsigned u;

switch(pCtx->ucState) {

case 0:

pCtx->i2c\_res.ucChannel = 0;

pCtx->i2c\_res.pResI2C = i2c\_get(0);

eeprom\_m24\_prepare\_context(&(pCtx->e\_ctx), EEPROM\_CONFIG\_ADDR, &(pCtx->i2c\_res), EEPROM\_CONFIG\_ADDR\_WIDTH, 0);

eeprom\_m24\_prepare\_io(&(pCtx->e\_io), pCtx->pbBuffer, pCtx->usCount, pCtx->ulAddress);

pCtx->ucState++;

case 1:

if ((ucRc = eeprom\_m24\_start\_read(&(pCtx->e\_ctx), &(pCtx->e\_io))) != 0) {

one\_message(pCtx, pcMsgError);

return 1;

}

pCtx->ucState++;

case 2:

ucRc = eeprom\_m24\_poll(&(pCtx->e\_ctx));

if (!ucRc) break;

if (ucRc != 1) {

one\_message(pCtx, pcMsgError);

return 1;

}

pCtx->ucState++;

pCtx->usPrinted = 0;

case 3:

u = out\_buffer\_to\_console(pCtx->pConsole, pCtx->pbBuffer, pCtx->usPrinted, pCtx->usCount);

pCtx->usPrinted += u;

if (pCtx->usPrinted >= pCtx->usCount) pCtx->ucProcessing = 0;

break;

}

return 1;

}

**eeset**:

static int command\_eeset(process\_ctx \*pCtx, unsigned char \*pbBuffer,

unsigned short usLen)

{

long lTmp;

unsigned long ulAddress;

unsigned uCount = 0;

unsigned char x,

xMax = sizeof(pCtx->pbBuffer),

\*pbP = pbBuffer;

unsigned short usL = usLen, usS, usF = 0;

TOKENIZE\_DIGIT(lTmp, 1);

if (lTmp < 0) return bad\_argument(pCtx, 1);

ulAddress = lTmp;

for(x = 0; x < xMax; x++) {

pbP += usF; if (usF > usL) break;

usL -= usF;

if (!extract\_token(pbP, usL, &usS, &usF)) break;

if (!check\_token\_digit(pbP + usS, usF - usS, &lTmp)) return error\_extract\_argument(pCtx, 2 + x); // arguments from 3

if ((lTmp < 0) || (lTmp > 255)) return bad\_argument(pCtx, 2 + x);

pCtx->pbBuffer[uCount++] = (lTmp & 0xFF);

}

if (x == 0) {

return error\_extract\_argument(pCtx, 2);

}

pCtx->ucProcessing = PROCESSING\_EESET;

pCtx->ucState = 0;

pCtx->ulAddress = ulAddress;

pCtx->usCount = x;

return command\_eeset\_cont(pCtx);

}

static int command\_eeset\_cont(process\_ctx \*pCtx)

{

unsigned char ucRc;

unsigned u;

switch(pCtx->ucState) {

case 0:

pCtx->i2c\_res.ucChannel = 0;

pCtx->i2c\_res.pResI2C = i2c\_get(0);

eeprom\_m24\_prepare\_context(&(pCtx->e\_ctx), EEPROM\_CONFIG\_ADDR, &(pCtx->i2c\_res), EEPROM\_CONFIG\_ADDR\_WIDTH, 0);

eeprom\_m24\_prepare\_io(&(pCtx->e\_io), pCtx->pbBuffer, pCtx->usCount, pCtx->ulAddress);

pCtx->ucState++;

case 1:

if ((ucRc = eeprom\_m24\_start\_write(&(pCtx->e\_ctx), &(pCtx->e\_io))) != 0) {

one\_message(pCtx, pcMsgError);

return 1;

}

pCtx->ucState++;

case 2:

ucRc = eeprom\_m24\_poll(&(pCtx->e\_ctx));

if (!ucRc) break;

if (ucRc != 1) {

one\_message(pCtx, pcMsgError);

return 1;

}

one\_message(pCtx, "OK\r\n");

break;

}

return 1;

}

Собираем образ прошивки командой **make**, прошиваем в плату, и перезагружаем.

Для проверки можно выполнить:

1 Чтение из EEPROM исходной информации (для примера читаем 10 байт с адреса 0):

**eeget 0 10**

**ff ff ff ff ff ff ff ff ff ff**

2 Запишем несколько значений (с адреса 1 запишем значения 1, 2, 3, 4, 5):

**eeset 1 1 2 3 4 5**

**OK**

3 Прочитаем те же 10 байт, что и на шаге 1:

**eeget 0 10**

**ff 01 02 03 04 05 ff ff ff ff**

Далее можно выполнить перезагрузку платы или выключить и включить ее питание – EEPROM будет содержать модифицированные данные

**eeget 0 10**

**ff 01 02 03 04 05 ff ff ff ff**

Лист регистрации изменений

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Номера листов (страниц) | | | |  |  |  |  |  |
| Изм. | изме-ненных | заме-нен-ных | новых | анну-лиро-ванных | Всего листов (стра-ниц) в доку-менте | Номер документа | Входящий номер сопрово-  дительного документа и дата | Под-пись | Дата |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |