|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ **Информатика и системы управления**

КАФЕДРА **Компьютерные системы и сети (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***к курсовой работе***

***по дисциплине «Микропроцессорные системы»***

***на тему:***

**Многоканальная регистрирующая МК-система**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-73Б |  |  | А.А. Бушев |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Руководитель |  |  |  | В.Я. Хартов |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |

*2022 г.*

ЛИСТ ЗАДАНИЯ

**РЕФЕРАТ**

РПЗ 56 страниц, 14 рисунков, 9 таблиц, 12 источников, 4 приложения.

МИКРОКОНТРОЛЛЕР, СИСТЕМА, РЕГИСТРАЦИЯ, ПИТАНИЕ

Объектом разработки является многоканальная регистрирующая МК-система.

Цель работы – создание функционального устройства ограниченной сложности, модель устройства и разработка необходимой документации на объект разработки.

Поставленная цель достигается посредством использования Keil MDK 5.

В процессе работы над курсовым проектом решаются следующие задачи: выбор МК и протокола обмена данных, создание функциональной и принципиальной схем системы, расчёт потребляемой мощности устройства, разработка алгоритма управления и соответствующей программы МК, а также написание сопутствующей документации.

**Содержание**

[Введение 6](#_Toc17521)

[1 Конструкторская часть 7](#_Toc32637)

[1.1 Анализ требований и принцип работы системы 7](#_Toc13792)

[1.2 Проектирование функциональной схемы 8](#_Toc22358)

[1.2.1 Микроконтроллер MDR32F9Q2I 8](#_Toc29569)

[1.2.2 Используемые элементы 13](#_Toc29191)

[1.2.3 Распределение портов 15](#_Toc16206)

[1.2.4 Организация памяти 17](#_Toc30495)

[1.2.5 Пульт оператора 23](#_Toc16941)

[1.2.6 Приём команд от смартфона 24](#_Toc21813)

[1.2.7 Настройка канала передачи 24](#_Toc7285)

[1.2.8 Батарейный домен и часы реального времени 31](#_Toc28380)

[1.2.9 LCD дисплей 32](#_Toc17316)

[1.2.10 Генератор таковых импульсов 37](#_Toc21510)

[1.2.11 Построение функциональной схемы 37](#_Toc24226)

[1.3 Проектирование принципиальной схемы 37](#_Toc5041)

[1.3.1 Разъем программатора 37](#_Toc7501)

[1.3.2 Подключение цепи питания 39](#_Toc20437)

[1.3.3 Расчёт сопротивления резисторов 39](#_Toc11543)

[1.3.4 Расчёт потребляемой мощности 40](#_Toc19948)

[1.3.5 Построение принципиальной схемы 41](#_Toc15370)

[1.4 Алгоритмы работы системы 43](#_Toc20619)

[1.4.1 Функция Main и модули 43](#_Toc17513)

[1.4.2 Цифровой автомат обработки команд 47](#_Toc10974)

[2 Технологическая часть 50](#_Toc28351)

[2.1 Отладка и тестирование программы 51](#_Toc17682)

[2.2 Способы программирования МК 52](#_Toc1319)

[Заключение 53](#_Toc16618)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 54](#_Toc2574)

[Приложение А Текст программы 56](#_Toc10253)

[Приложение Б Графическая часть 90](#_Toc21948)

[Приложение В Перечень элементов 91](#_Toc6619)

[Приложение Г Принципиальная схема отладочной платы 92](#_Toc31077)

**ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ**

МК – микроконтроллер.

ТЗ – техническое задание.

Keil MDK 5 — пакет программ для автоматизированного программирования (САПР) электронных схем.

ASCII – таблица кодировки символов.

UART – Universal asynchronous receiver/transmitter – последовательный универсальный синхронный/асинхронный приемопередатчик.

Bluetooth — производственная спецификация беспроводных персональных сетей (Wireless personal area network, WPAN).

LCD — экран на основе жидких кристаллов.

ПЗУ — постоянное запоминающее устройство.

JTAG — специализированный аппаратный интерфейс на базе стандарта IEEE 1149.1.

ЖК — жидкокристаллический.

**Введение**

В данной работе производится разработка многоканальной регистрирующей МК-системы.

В процессе выполнения работы проведён анализ технического задания,

создана концепция устройства, разработаны электрические схемы, построен алгоритм и управляющая программа для МК, выполнено интерактивное моделирование устройства.

Система состоит из МК, LCD дисплея, пульта оператора, Bluetooth-модуля, и 6 модулей реле. Система автоматически отправляет собранные данные по беспроводному подключению. Также можно управлять каждым модулем реле как через меню, так и дистанционно через Bluetooth.

Актуальность разрабатываемой системы заключается в том, что концепция «умных домов» интенсивно совершенствуется и это невозможно без регистрирующих систем. Регистрирующая система должна позволять управлять работой разного рода устройств и собирать информацию об использовании этих устройств.

**1 Конструкторская часть**

**1.1 Анализ требований и принцип работы системы**

Исходя из требований, изложенных в техническом задании, можно сделать вывод, что задачей работы системы является управление питанием и регистрация времени работы шести электроприборов.

Системы должно позволять включать и выключать электроприборы посредством пульта оператора или смартфона, подключённого к системе посредством Bluetooth.

Система должна регистрировать время работы электроприборов и записывать зарегистрированные данные в ПЗУ МК посредством пульта оператора.

Система должна отправлять зарегистрированные данные на смартфон, подключённый к системе посредством Bluetooth, каждые двенадцать часов в полночь и полдень.

Система должна позволять отправлять текущие зарегистрированные данные на смартфон, подключённый к системе посредством Bluetooth, посредством пульта оператора. Система должна позволять сохранять текущие зарегистрированные данные в ПЗУ МК посредством пульта оператора. Система должна сохранять в оперативной памяти МК зарегистрированные за предыдущий период работы данные.

Система должна позволять изменять текущее время на часах реального времени посредством пульта оператора. Изменение времени считается концом текущего периода работы, что означает, что система должна отправить текущие зарегистрированные данные на смартфон, подключённый к системе посредством Bluetooth, и сохранить в оперативной памяти МК текущие зарегистрированные данные как зарегистрированные за предыдущий период времени данные.

Система должна позволять запросить текущие зарегистрированные данные и зарегистрированные за предыдущий период работы данные посредством пульта оператора и смартфона, подключённого к системе посредством Bluetooth.

Работа МК разделяется на следующие этапы:

1. Инициализация
2. Обработка ввода с пульта управления
3. Обработка ввода с UART
4. Обновление регистрируемых данных
5. Обновление меню, выводимого на LCD дисплей

Этапы 2–5 выполняются в бесконечном цикле.

Обработка ввода с пульта оператора происходит, когда считанный ввод отличается от считанного на предыдущей итерации.

Взаимодействие с электроприборами происходит посредством реле.

Этапы 4–5 выполняются каждую секунду, отмеренную часами реального времени.

Разработанная структурная схема системы представлена на рисунке 1.

**1.2 Проектирование функциональной схемы**

В этом разделе приведено функциональное описание работы системы и проектирование функциональной схемы.

**1.2.1 Микроконтроллер MDR32F9Q2I**

В этом разделе приведено функциональное описание работы системы и проектирование функциональной схемы.

Основным элементом разрабатываемого устройства является микроконтроллер (МК). В соответствии с техническим заданием имеется на выбор два семейства МК:

* AVR – это серия МК, разработанная компанией Atmel;
* «Миландр» – это серия МК на основе ARM ядра.

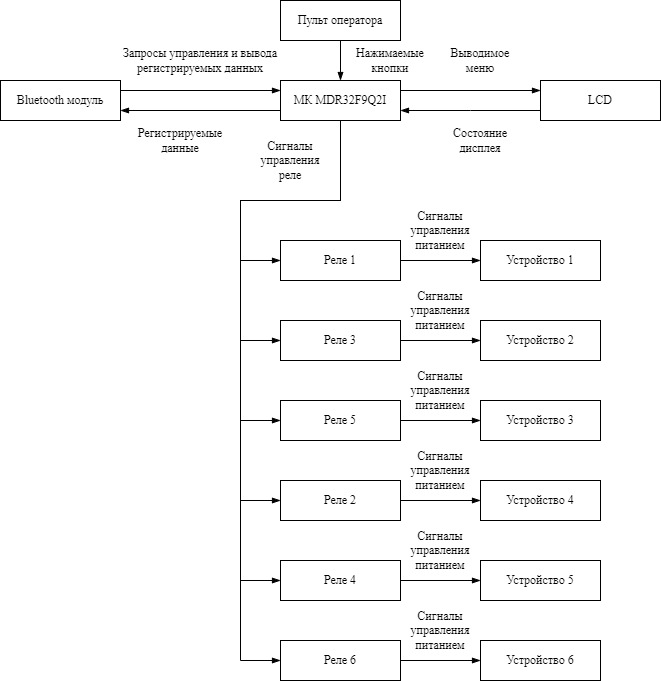


Рисунок 1 – Структурная схема системы

Сравнение семейств показано в таблице 1[1][2].

Было выбрано семейство «Миландр», так как рассмотренные семейства сопоставимы, а компания «Миландр» является отечественной компанией.

Выбран МК MDR32F9Q2I, обладающий всем необходимым функционалом для реализации проекта:

* интерфейс JTAG для программирования МК;
* интерфейс UART для обмена данными;
* 32 КБ ОЗУ;
* 128 Кбайт FLASH памяти;
* частота работы до 80 Мгц.

Таблица 1 – Сравнение семейств МК

|  |  |  |
| --- | --- | --- |
| Критерий | AVR | «Миландр» |
| Разрядность | 8/32 бит | 8/32 бит |
| Интерфейсы | UART, USART, SPI, I2C, иногда CAN, USB, Ethernet | UART, USART, LIN, I2C, SPI, CAN, USB, Ethernet, I2S, DSP, SAI, IrDA |
| Скорость | 1 такт на инструкцию | 1 такт на инструкцию |
| Память | Flash, SRAM, EEPROM | Flash, SDRAM, EEPROM |
| Энергопо-требление | Низкое | Низкое |
| Объем FLASH памяти | До 256 Кб | До 192 Кб |
| Производители | Atmel Microchip | «Миландр» |

Структурная схема МК показана на рисунке 2 и УГО на рисунке 3[3].

Он обладает следующими характеристиками[3]:

Ядро:

* ядро ARM Cortex-M3 ревизии 2.0, 32-разрядная RISC-архитектура, тактовая частота до 80 МГц, производительность 1,25 DMIPS/МГц (Dhrystone 2.1) при нулевой задержке памяти;
* блок аппаратной защиты памяти MPU;
* умножение за один цикл, аппаратная реализация деления.

Память:

* встроенная энергонезависимая Flash-память программ размером 128 Кбайт;
* встроенное ОЗУ размером 32 Кбайт;
* контроллер внешней шины с поддержкой микросхем памяти СОЗУ, ПЗУ, NAND Flash.

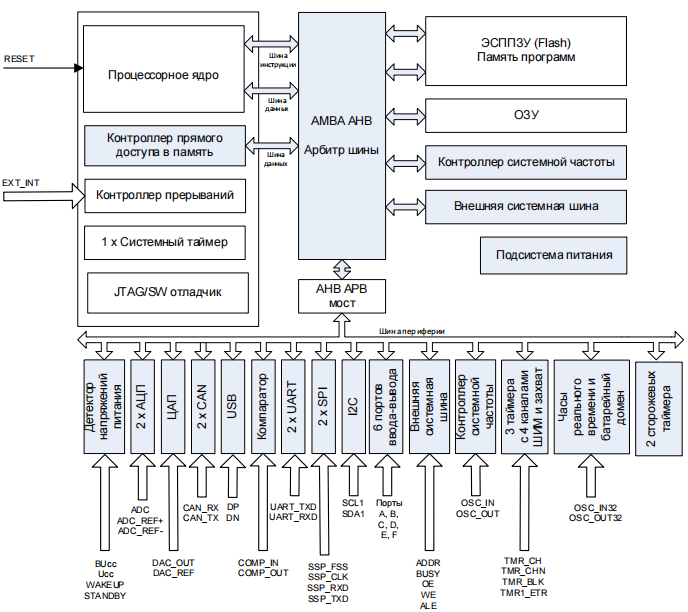


Рисунок 2 – Структурная схема МК MDR32F9Q2I

Питание и тактовая частота:

* внешнее питание 2,2 ÷ 3,6 В;
* встроенный регулируемый стабилизатор напряжения на 1,8 В для питания ядра;
* встроенные схемы контроля питания;
* встроенный домен с батарейным питанием;
* встроенные подстраиваемые RC-генераторы 8 МГц и 40 кГц;
* внешние кварцевые резонаторы на 2 ÷ 16 МГц и 32 кГц;
* встроенный умножитель тактовой частоты PLL для ядра;
* встроенный умножитель тактовой частоты PLL для USB.

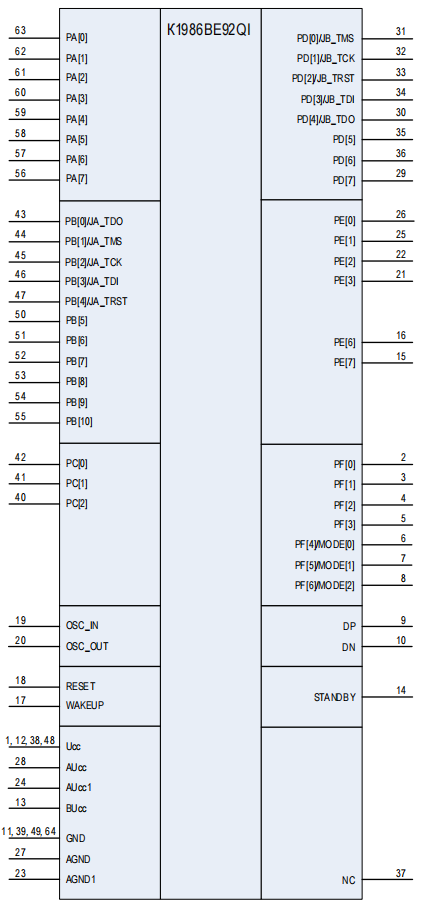


Рисунок 3 – УГО МК MDR32F9Q2I

Режим пониженного энергопотребления:

* режимы Sleep, Deep Sleep и Standby;
* батарейный домен с часами реального времени и регистрами аварийного сохранения.

Аналоговые модули:

* два 12-разрядных АЦП (8 каналов);
* температурный датчик;
* 12-разрядный ЦАП;
* встроенный компаратор.

Периферия:

* контроллер DMA с функциями передачи Периферия-Память, Память-Память;
* два контроллера CAN интерфейса;
* контроллер USB интерфейса с функциями работы Device и Host;
* контроллеры интерфейсов UART, SPI, I2C;
* три 16-разрядных таймер-счетчика с функциями ШИМ и регистрации событий;
* 43 пользовательских линий ввода-вывода.

Отладочные интерфейсы:

* последовательные интерфейсы SWD и JTAG.

**1.2.2 Используемые элементы**

Для функционирования системы в МК MDR32F9Q2I задействованы не все элементы его архитектуры. Выделим и опишем те, что используются во время функционирования системы[3].

Порты А, В, C, D, E, F – назначения каждого из них описано в разделе *1.2.3*.

Указатель стека – используется для работы со стеком, при вызове подпрограмм. В коде они присутствуют.

SRAM – статическая память МК, где хранятся объявленные переменные.

Регистры общего назначения – предназначены для хранения операндов арифметико-логических операций, а также адресов или отдельных компонентов адресов ячеек памяти.

АЛУ – блок процессора, который под управлением устройства управления служит для выполнения арифметических и логических преобразований над данными.

JTAG – последовательный интерфейс для прошивки и отладки МК.

Программный счётчик – используется для указания следующей команды выполняемой программы.

Память Flash – память МК, в котором хранится загруженная в него программа.

Регистры команды – содержит исполняемую в текущий момент (или следующий) команду, то есть команду, адресуемую счётчиком команд.

Декодер – блок, выделяющий код операции и операнды команды, а затем вызывающий микропрограмму, которая исполняет данную команду.

Сигналы управления – синхронизируют обработку данных.

Логика программирования – устанавливает логику того, как программа будет вшита в МК.

Управление синхронизацией и сбросом (MCU CTRL. & Timing) – в этом блоке обрабатываются тактовые сигналы и принимается сигнал сброса.

Прерывания – контроллер прерываний обрабатывает внешние прерывания и прерывания от периферийных устройств МК (таймеров, портов ввода/вывода). В проекте используется прерывание при переполнении счётчика.

UART – через этот интерфейс в МК передаётся информация из ПЭВМ. В регистр UART информация попадает через порт PD0 (RxD).

Генератор – генератор тактовых импульсов. Необходим для синхронизации работы МК.

Часы реального времени и батарейный домен – Часы реального времени позволяют организовать механизм отсчёта времени в кристалле, в том числе при отключении основного источника питания..Блок батарейного домена предназначен для обеспечения функций часов реального времени и сохранения некоторого набора пользовательских данных при отключении основного источника питания.

**1.2.3 Распределение портов**

МК MDR32F9Q2I содержит пять портов – А, В, С, D, E и F. Опишем назначение тех, что используются в данной системе для её функционирования.

Порт А:

* РА0 – LCD: линия 0 параллельной шины данных (DB0);
* РА1 – LCD: линия 1 параллельной шины данных (DB1);
* РА2 – LCD: линия 2 параллельной шины данных (DB2);
* РА3 – LCD: линия 3 параллельной шины данных (DB3);
* РА4 – LCD: линия 4 параллельной шины данных (DB4);
* РА5 – LCD: линия 5 параллельной шины данных (DB5);
* РА6 – на него поступает информация по протоколу UART от модуля Bluetooth. Используется для получения команд со смартфона, подключённого к системе посредством Bluetooth (PA6\_RX);
* РА7 – с него поступает информация по протоколу UART на модуль Bluetooth. Используется для отправки регистрируемых данных и ошибок при обработке команд, поступающих на порт PA6 (PA7\_TX).

Порт В:

* РВ0 – управление реле 1. Если поступает логическая «1», то реле замыкается, если поступает логический «0», то реле открывается;
* РВ1 – управление реле 2. Если поступает логическая «1», то реле замыкается, если поступает логический «0», то реле открывается;
* РВ2 – управление реле 3. Если поступает логическая «1», то реле замыкается, если поступает логический «0», то реле открывается;
* РВ3 – управление реле 4. Если поступает логическая «1», то реле замыкается, если поступает логический «0», то реле открывается;
* РВ4 – управление реле 5. Если поступает логическая «1», то реле замыкается, если поступает логический «0», то реле открывается;
* РВ5 – на него поступает сигнал о нажатии кнопки «UP» пульта оператора. Если поступает логический «0», то значит, что кнопка нажата. Используется для взаимодействия с меню;
* РВ7 – LCD: выбор первого сегмента дисплея. Дисплей с разрешением 128x64 обладает двумя сегментами, отвечающих за 1–64 и 65–128 столбца соответственно (E1);
* РВ8 – LCD: выбор второго сегмента дисплея. Дисплей с разрешением 128x64 обладает двумя сегментами, отвечающих за 1–64 и 65–128 столбца соответственно (E2);
* РВ9 – LCD: сброс при подаче логического «0» (RES);
* РВ10 – LCD: «1» — чтение данных / «0» — запись данных (R/W).

Порт C:

* РC0 – LCD: выбор регистра получателя информации: «1» —регистр данных, «0» — регистр инструкций (A0);
* PC1 – LCD: разрешающий сигнал. Активный уровень сигнала — логическая «1»;
* PC2 – на него поступает сигнал о нажатии кнопки «SEL» пульта оператора. Если поступает логический «0», то значит, что кнопка нажата. Используется для взаимодействия с меню.

Порт D:

* РD0 – JTAG-B: Test Mode Select — сигнал управления TAP-контроллером (TMS);
* РD1 – JTAG-B: Test Clock — тактовый сигнал (TCK);
* РD2 – JTAG-B: Test Reset — сброс, активный уровень сигнала — «0» (nTRST);
* РD3 – JTAG-B: Test Data Input — сигнал данных на вход, данные задвигаются по переднему фронту TCK (TDI);
* РD4 – JTAG-B: Test Data Output — выход последовательных данных JTAG, выдвигаются по обратному фронту TCK, должен находиться в третьем состоянии — «Z» — когда данные не передаются (TDO);
* PD5 – управление реле 6. Если поступает логическая «1», то реле замыкается, если поступает логический «0», то реле открывается.

Порт E:

* РE1 – на него поступает сигнал о нажатии кнопки «DN» пульта оператора. Если поступает логический «0», то значит, что кнопка нажата. Используется для взаимодействия с меню.

Порт F:

* РF2 – LCD: линия 6 параллельной шины данных (DB6);
* PF3 – LCD: линия 7 параллельной шины данных (DB7).

**1.2.4 Организация памяти**

Процессорное ядро имеет три системных шины:

* I Code – шина выборки инструкций;
* D Code – шина выборки данных, расположенных в коде программы;
* S Bus – шина выборки данных, расположенных в области ОЗУ.

Также в микросхеме реализован контроллер прямого доступа в память (DMA), который осуществляет выборку через шину DMA Bus.

Все адресное пространство микросхемы едино и имеет максимальный объем 4 Гбайт. В данное адресное пространство отображаются различные модули памяти и периферии.

Структурная схема организации памяти МК MDR32F9Q2I показана на рисунке 4.

Карта распределения основных областей памяти МК MDR32F9Q2I показана на рисунке 5.

**Секция CODE**

1. Область BOOT ROM предназначена для хранения программы запуска микросхемы; в ходе выполнения этой программы определяется режим запуска основной программы или переход в режим программирования микросхемы.

2. Область EEPROM PROGRAM — основная область энергонезависимой памяти программы, доступной для перепрограммирования пользователем. Память предназначена для хранения основной рабочей программы.

3 Область EXTERNAL BUS — Область отображения внешней системной шины в адресное пространство области программы. Предназначена для хранения кода программ во внешних микросхемах памяти, подсоединённых к внешней системной шине.

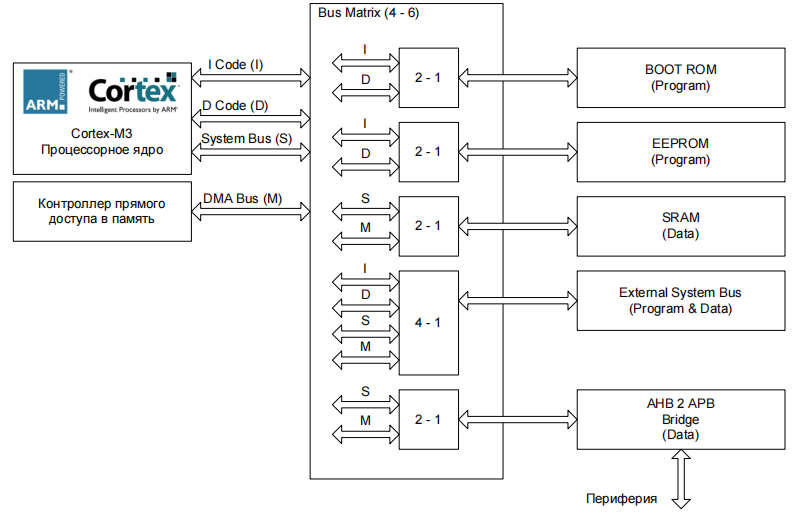


Рисунок 4 – Структурная схема организации памяти

**Секция DATA**

1. Область Internal SRAM (Data) — основная область ОЗУ, предназначенная для хранения данных программы. В данной область также располагаются стек (stack) и «куча» (heap) программы. Адресные диапазоны стека и «кучи» задаются пользователем при написании программы.

2. Область BIT BAND REGION TO SRAM (Data) — виртуальная область памяти данных, предназначенная для осуществления побитного доступа к области Internal SRAM. Работа с BIT BAND REGION позволяет осуществлять операции «Установка бита» и «Сброс Бита» одной инструкцией «Чтение-Модификация-Запись».

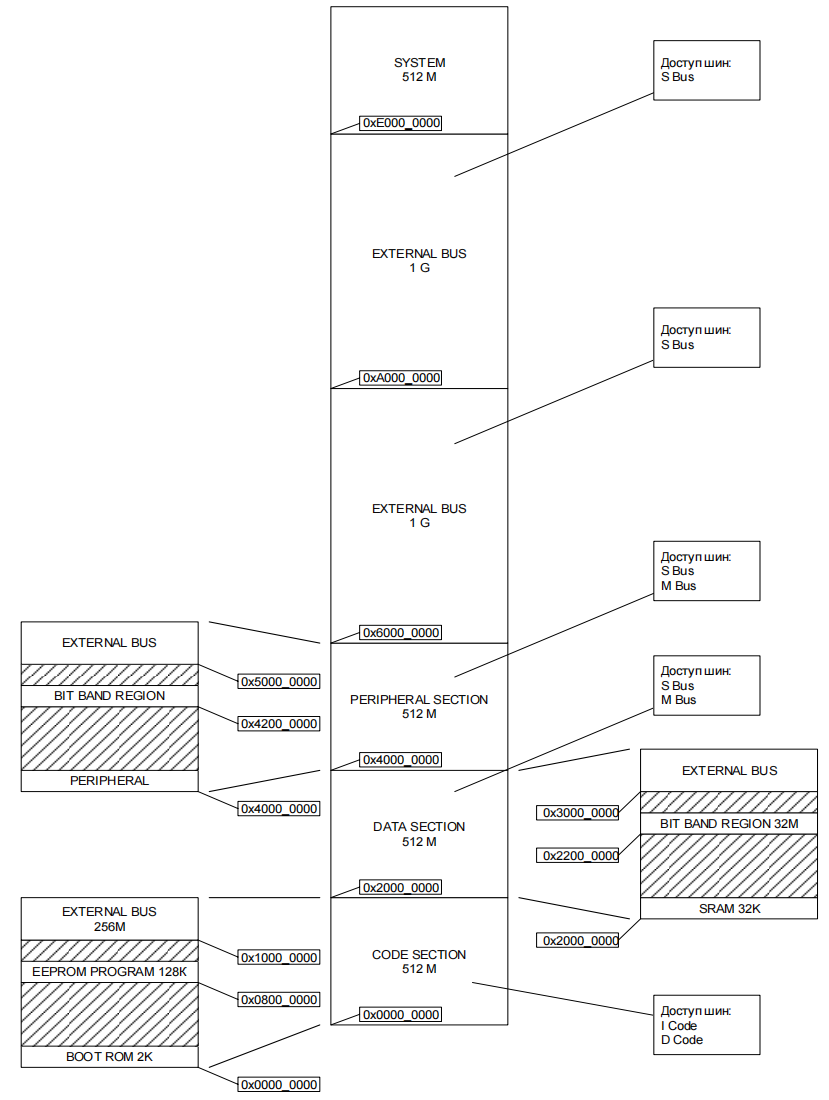


Рисунок 5 – Карта распределения основных областей памяти

Запись слова в область alias-региона имеет тот же эффект, что и операция чтение-модификация-запись для целевого бита в области bit-band региона. Бит [0] записываемого значения в область alias-региона определяет значение, которое будет записано в целевой бит в области bit-band региона. Запись значения с установленным битом [0] записывает «1» в целевой бит bit-band региона, а запись значения с очищенным битом [0] записывает «0» в целевой бит bit-band региона. Биты [31:1] записываемого слова в область alias-региона не влияют на целевой бит bit-band региона. Запись 0x01 имеет тот же эффект, что и запись 0xFF. Запись 0x00 имеет тот же эффект, что и запись 0xFE. При чтении слова из области alias-региона будет возвращено значение 0x01 или 0x00. Биты [31:1] равны нулю. Значение 0x01 указывает, что целевой бит в области bit-band региона установлен. Значение 0x00 указывает, что целевой бит в области bit-band региона сброшен.

3. Область EXTERNAL BUS — область отображения внешней системной шины в адресное пространство области данных. Предназначена для хранения данных во внешних микросхемах памяти, подсоединённых к внешней системной шине.

**Секция PERIPHERAL**

1. Область PERIPHERAL (Data) — область отображения регистров периферии в общее адресное пространство памяти.

2. Область BIT BAND REGION TO PERIPHERAL (Data) —Виртуальная область памяти данных, предназначенная для осуществления побитного доступа к области PERIPHERAL. Работа с BIT BAND REGION позволяет осуществлять операции «Установка бита» и «Сброс Бита» одной инструкцией «Чтение-Модификация-Запись».

3. Область EXTERNAL BUS — область отображения внешней системной шины в адресное пространство области периферии. Предназначена для хранения данных во внешних микросхемах памяти или для работы с периферийными устройствами, подсоединёнными к внешней системной шине.

**Секция EXTERNAL RAM**

Область EXTERNAL BUS — область отображения внешней системной шины в адресное пространство области внешней памяти и периферии. Эта секция предназначена для хранения данных во внешних микросхемах памяти или для работы с периферийными устройствами, подсоединёнными к внешней системной шине.

**Секция SYSTEM**

Предназначена для отображения системных регистров ядра и системной периферии.

**Блок BUS MATRIX**

Блок BUS MATRIX предназначен для переключения системных шин I Code, D Code, System Bus и DMA Bus между различными областями памяти. Переключение производится автоматически на основании адреса запроса каждой конкретной шины. Если адреса запросов не пересекаются, то они могут быть выполнены одновременно. Если адреса запросов пересекаются, то они выполняются в порядке приоритета. Приоритеты обращений заданы аппаратно. Наивысшим приоритетом обладает запрос по шине SystemBus, затем следует запрос D Code, затем I Code и наименьшим приоритетом обладает запрос DMA Bus. Если два запроса пришли одновременно, то выполняется запрос с большим приоритетом. Запрос с меньшим приоритетом задерживается до окончания выполнения запроса с большим приоритетом. При переключении между шинами возникает дополнительная задержка в один цикл. Если запросы идут непосредственно друг за другом, то дополнительных задержек не возникает.

**Память BOOT ROM**

Память области BOOT ROM реализована в виде MASK ROM, c занесением информации одним из технологических слоёв при изготовлении кристалла микросхемы. Скорость доступа к памяти BOOT ROM – один цикл системной частоты.

**Память EEPROM**

Память области EEPROM реализована в виде перепрограммируемой энергонезависимой памяти. Скорость доступа к памяти EEPROM – порядка 40 нс. При работе микросхемы на скорости до 100 МГц скорость доступа к памяти может составлять до 5 циклов системной частоты. При последовательной выборке за счёт упреждающего чтения задержка может быть сокращена до одного цикла системной частоты.

**Память SRAM**

Память области SRAM реализована в виде блока статической памяти. Скорость доступа к памяти SRAM – один цикл системной частоты.

**1.2.5 Пульт оператора**

Пульт оператора представляет из себя кнопочный джойстик, состоящий из пяти кнопок. Кнопки подключены к земле, поэтому при их замыкании на соответствующих выводах появляется логический «0». Используется только три кнопки: кнопка «вверх» («UP»), кнопка выбора («SEL») и кнопка «вниз» («DN»).

Пульт оператора используется для переключения и выбора пунктов меню. Меню представляет собой список строк, на каждую из которых можно переключиться и выбрать. При выборе пункта меню может открыться под меню или выполнится какое-то действие.

**1.2.6 Приём команд от смартфона**

Приём данных от смартфона происходит через Bluetooth-модуль HC-06 — интегральная схема, преобразующая сигналы последовательного порта UART в радиоволны.

UART — асинхронный интерфейс, который преобразует передаваемые данные в последовательный вид так, чтобы было возможно передать их по одной физической цифровой линии другому аналогичному устройству. Представляет собой логическую схему, с одной стороны подключённую к шине вычислительного устройства, а с другой имеющую два или более выводов для внешнего соединения. Используется для передачи данных через последовательный порт компьютера.

Скорость обмена данных через интерфейс UART для Bluetooth-модуля HC-06 по умолчанию составляет 9600 бод. HC-06 подключён к смартфону через Bluetooth соединение.

Назначение выводов HC-06 описано в таблице 2[4].

Таблица 2 — Назначение выводов HC-06

|  |  |  |  |
| --- | --- | --- | --- |
| Номер | Имя | Тип | Описание |
| 1 | RXD | I | Ввод данных через интерфейс UART |
| 2 | TXD | O | Вывод данных через интерфейс UART |
| 3 | GND | – | Земля |
| 4 | VCC | – | Напряжение питания, подключение к внешнему источнику питания 5 В |

**1.2.7 Настройка канала передачи**

Для передачи информации в МК используется последовательный интерфейс UART (первый из двух контроллеров — UART1). Контроллер содержит независимые буферы приёма (16x12) и передачи (16x8) типа FIFO (First In First Out – первый вошёл, первый вышел), что позволяет снизить интенсивность прерываний центрального процессора. Допускается использование нецелых коэффициентов деления частоты.

Для корректной работы необходимо настроить регистры управления MDR\_UART1->IBRD, MDR\_UART1->FBRD, MDR\_UART1->LCR\_H, MDR\_UART1->CR, MDR\_UART1->IFLS [3].

MDR\_UART1->IBRD — Целая часть делителя скорости обмена данными. MDR\_UART1->FBRD — Дробная часть делителя скорости обмена данными.

Коэффициент деления для формирования скорости передачи данных состоит из 22 бит, при этом 16 бит выделено для представления его целой части, а 6 бит – дробной части [3]. Целая часть коэффициента деления записывается в 16-битный регистр UARTIBRD. Шестиразрядная дробная часть записывается в регистр UARTFBRD. Значение коэффициента деления связано с содержимым указанных регистров следующим образом:

Коэффициент деления ,

где BRD\_I – целая часть коэффициента деления, BRD\_F – дробная часть коэффициента деления.

Шестибитное значение, записываемое в регистр UARTFBRD, вычисляется путём выделения дробной части требуемого коэффициента деления, умножения её на 64 (то есть на , где n – ширина регистра UARTFBRD) и округления до ближайшего целого числа

,

где integer – операция отсечения дробной части числа, n = 6.

Пусть требуемая скорость передачи данных составляет 9600 бод, частота тактового сигнала UARTCLK равна 8 МГц. Тогда

Коэффициент деления .

Таким образом, , .

Следовательно,

,

.

Реальное значение коэффициента деления . Реальная скорость передачи данных бод. Ошибка установки скорости .

MDR\_UART1->LCR\_H — Регистр управления линией. Биты регистра LCR\_H показаны в таблице 3. При сбросе все биты регистра UARTLCR\_H обнуляются.

Содержимое регистров UARTLCR\_H, UARTIBRD и UARTFBRD совместно образует общий 30-разрядный регистр UARTLCR, который обновляется по стробу, формируемому при записи в UARTLCR\_H. Таким образом, для того, чтобы изменение параметров коэффициента деления частоты обмена данными вступило в силу, после изменения значения регистров UARTIBRD и/или UARTFBRD необходимо осуществить запись данных в регистр UARTLCR\_H.

Перечислим значения для битов, которые используются при работе системы. Передача бита чётности при работе системы не предусматривается, поэтому PEN = 0. Предполагается передача одного стопового бита, поэтому STP2 = 0. Предполагается передача байт, поэтому WLEN = 0b11. Предполагается буферизация, поэтому FEN = 1. Предполагается передача одного стопового бита, поэтому STP2 = 0.

MDR\_UART1->IFLS — Регистр порога прерывания по заполнению буфера FIFO. Биты регистра IFLS показаны в таблице 4.

Перечислим значения для битов, которые используются при работе системы. Выберем 1/4 заполненность буферов приёма, поэтому RXIFLSEL = b001. Выберем 7/8 заполненность буферов отправки, поэтому TXIFLSEL = b100.

Таблица 3 — Биты регистра UARTLCR\_H.

|  |  |  |
| --- | --- | --- |
| Номер бита | Функци-ональное имя бита | Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений |
| 7 | SPS | Передача бита чётности с фиксированным значением.  0 – запрещена;  1 – на месте бита чётности передаётся инверсное значение бита EPS, оно же проверяется при приёме данных. (При EPS=0 на месте бита чётности передаётся 1, при EPS=1 – передаётся 0).  Значение бита SPS не играет роли в случае, если битом PEN формирование и проверка бита чётности запрещён |
| 6...5 | WLEN | Длина слова – количество передаваемых или принимаемых информационных бит в кадре:  0b11 – 8 бит  0b10 – 7 бит  0b01 – 6 бит  0b00 – 5 бит |
| 4 | FEN | Разрешение работы буфера FIFO приёмника и передатчика.  0 – запрещено;  1 – разрешено |
| 3 | STP2 | Режим передачи двух стоповых бит.  0 – один стоповый бит;  1 – два стоповых бита.  Приёмник не проверяет наличие дополнительного стопового бита в кадре |
| 2 | EPS | Чётность/нечётность.  0 – бит чётности дополняет количество единиц в информационной части кадра до нечётного;  1 – до чётного числа.  Значение бита EPS не играет роли в случае, если битом PEN формирование и проверка бита чётности запрещена |
| 1 | PEN | Разрешение проверки чётности.  0 – кадр не содержит бита чётности;  1 – бит чётности передаётся в кадре и проверяется при приёме данных |
| 0 | BRK | Разрыв линии. Если этот бит установлен в 1, то по завершении передачи текущего символа на выходе UARTTXD устанавливается низкий уровень сигнала. Для правильного выполнения этой операции программное обеспечение должно обеспечить передачу сигнала разрыва в течение, как минимум, времени передачи двух информационных кадров. В нормальном режиме функционирования бит должен быть установлен в 0 |

Таблица 4 — Биты регистра UARTIFLS.

|  |  |  |
| --- | --- | --- |
| Номер бита | Функци-ональное имя бита | Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений |
| 5...3 | RXIFLSEL | Порог прерывания по заполнению буфера приёмника:  b000 = буфер заполнен на 1/8  b001 = буфер заполнен на 1/4  b010 = буфер заполнен на 1/2  b011 = буфер заполнен на 3/4  b100 = буфер заполнен на 7/8  b101-b111 = резерв |
| 2...0 | TXIFLSEL | Порог прерывания по заполнению буфера передатчика:  b000 = буфер заполнен на 1/8  b001 = буфер заполнен на 1/4  b010 = буфер заполнен на 1/2  b011 = буфер заполнен на 3/4  b100 = буфер заполнен на 7/8  b101-b111 = резерв  Также стоит помнить, что в случае, если сдвиговый регистр передатчика пуст, то слово, записанное в FIFO, будет сразу же переписано в сдвиговый регистр. Следовательно, для генерирования события прерывания от передатчика блока UART необходимо произвести запись в FIFO такого количества слов, которое превысит установленный порог хотя бы на одно слово с учётом описанного случая |

MDR\_UART1->CR — Регистр управления. Биты регистра CR показаны в таблице 5. После сброса все биты регистра управления, за исключением бит 9 и 8 устанавливаются в нулевое состояние. Биты 9 и 8 устанавливаются в единичное состояние.

Перечислим значения для битов, которые используются при работе системы. Во время работы системы будет использовать и приём, и отправка данных через интерфейс UART, поэтому RXE = 1, TXE = 1.

Для запуска работы UART контроллера необходимо установить бит UARTEN = 1

Таблица 5 — Биты регистра UARTCR.

|  |  |  |
| --- | --- | --- |
| Номер бита | Функци-ональное имя бита | Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений |
| 15 | CTSen | Разрешение управления потоком данных по CTS.  1 – разрешено, данные передаются в линию только при активном значении сигнала nUARTCTS. |
| 14 | RTSen | Разрешение управления потоком данных по RTS.  1 – разрешено. Запрос данных от внешнего устройства осуществляется только при наличии свободного места в буфере FIFO приёмника |
| 13 | Out2 | Инверсия сигнала на линии состояния модема nUARTOut2. В режиме оконечного оборудования (DTE) эта линия может использоваться в качестве линии «сигнал вызова» (RI) |
| 12 | Out1 | Инверсия сигнала на линии состояния модема nUARTOut1. В режиме оконечного оборудования (DTE) эта линия может использоваться в качестве линии «обнаружен информационный сигнал» (DCD) |
| 11 | RTS | Инверсия сигнала на линии состояния модема nUARTRTS |
| 10 | DTR | Инверсия сигнала на линии состояния модема nUARTDTR |
| 9 | RXE | Приём разрешён. Установка бита в 1 разрешает работу приёмника. Приём данных осуществляется либо по интерфейсу асинхронного последовательного обмена, либо по интерфейсу ИК обмена SIR, в зависимости от значения бита SIREN.  В случае перевода приемопередатчика в запрещённое состояние в ходе приёма данных, он завершает приём текущего символа перед остановкой |
| 8 | TXE | Передача разрешена. Установка бита в 1 разрешает работу передатчика. Передача осуществляется либо по интерфейсу  асинхронного последовательного обмена, либо по интерфейсу ИК обмена SIR, в зависимости от значения бита SIREN.  В случае перевода приемопередатчик в запрещённое состояние в ходе передачи данных, он завершает передачу текущего символа: перед остановкой |

Таблица 5 — продолжение.

|  |  |  |
| --- | --- | --- |
| 7 | LBE | 0 – запрещено;  1 – шлейф разрешён.  В режиме разрешённого шлейфа:  Если установлены бит SIREN=1 и бит регистра управления тестированием UARTTCR SIRTEST=1, то сигнал с выхода кодека nSIROUT инвертируется и подаётся на вход кодека SIRIN. Бит SIRTEST устанавливается в 1 для того, чтобы вывести устройство из полудуплексного режима, характерного для интерфейса SIR. После окончания тестирования по шлейфу бит SIRTEST должен быть установлен в 0.  Если бит SIRTEST=0, то выходная линия передатчика UARTTXD коммутируется на вход приемника UARTRXD.  Как в режиме SIR, так и в режиме UART, выходные линии состояния модема коммутируются на соответствующие входные линии.  После сброса бит устанавливается в 0 |
| 2 | SIRPL | Выбор режима ИК обмена с пониженным энергопотреблением:  0 – длительность импульсов данных равна 3/16 длительности передачи бита;  1 – длительность импульсов данных равна трём тактам сигнала IrLPBaud16 вне зависимости от выбранной скорости передачи данных. Выбор этого режима снижает энергопотребление, однако может привести к уменьшению дальности связи |
| 1 | SIREN | Разрешение работы кодека ИК передачи данных IrDA SIR:  0 – запрещено. Сигнал nSIROUT находится в низком состоянии, данные на входе SIRIN не обрабатываются.  1 – разрешено. Данные передаются на выход nSIROUT и принимаются с входа SIRIN. Линия UARTTXD находится в высоком состоянии. Данные на входе UARTRXD и линиях состояния модема не обрабатываются.  В случае, если UARTEN=0 значение бита не играет роли |
| 0 | UARTEN | Разрешение работы приемопередатчика:  0 – работа запрещена. Перед остановкой завершается прием и/или передача обрабатываемого в текущий момент символа.  1 – работа разрешена. Производится обмен данными либо по линиям асинхронного обмена, либо по линиям ИК обмена SIR, в зависимости от состояния бита SIREN |

**1.2.8 Батарейный домен и часы реального времени**

Блок батарейного домена предназначен для обеспечения функций часов реального времени и сохранения некоторого набора пользовательских данных при отключении основного источника питания. При снижении питания UCC в блоке SW происходит автоматическое переключение питания BDUCC c UCC на BUCC. Если на BUCC имеется отдельный источник питания (батарейка), то батарейный домен остаётся включённым и может выполнять свои функции.

Батарейный домен используется для настройки часов реального времени (хранит ключевое значение для определения того, что часы реального времени уже были настроены) и для поддержания непрерывной работы часов реального времени, которые работают на основе внешнего опорного тактового сигнала с частотой 32,768 Гц.

Часы реального времени позволяют организовать механизм отсчёта времени в кристалле, в том числе при отключении основного источника питания. Включение часов реального времени осуществляется битом RTCEN. В качестве источника тактовой частоты часов реального времени может выступать генератор LSI, или осциллятор LSE, или HSE, или HSI c дополнительным делителем до 256 (HSE и HSI формируются в блоке управления тактовыми частотами и могут быть выбраны только при наличии питания DUCC, LSI может быть выбран при наличии питания UCC, LSE может быть выбран при наличии UCC или BUCC). Выбор между источниками осуществляется битами RTCSEL. При возможном отключении основного источника питания UCC в качестве источника тактовой частоты должен использоваться осциллятор LSE, так как он также имеет питание BDUCC. Биты управления осциллятором LSE расположены в батарейном домене и таким образом при отключении основного питания они не сбрасываются.

Для калибровки тактовой частоты внутренним регистром RTC\_20 используются биты CAL[6:0]. Значение CAL определяет, какое число тактов из 220 будет замаскировано.

Таким образом, с помощью бит CAL[6:0] производится замедление хода часов. Изменение значения бит CAL может быть осуществлено в ходе работы часов реального времени.

Регистр RTC\_DIV выступает в роли 20-битного предварительного делителя входной тактовой частоты, таким образом, чтобы на его выходе была тактовая частота в 1 Гц. Для задания коэффициента деления регистра RTC\_DIV используется регистр RTC\_PRL.

Регистр RTC\_CNT предназначен для отсчёта времени в секундах и работает на выходной частоте делителя RTC\_DIV.

Батарейный домен имеет 16 встроенных 32-х разрядных регистров аварийного сохранения. 16-й и 15-й регистры служат для хранения бит управления батарейным доменом, оставшиеся 14 регистров могут быть использованы разработчиком программы.

Структурная блок-схема батарейного домена и часов реального времени показана на рисунке 6.

Описание регистров блока батарейного домена показаны в таблице 6.

**1.2.9 LCD дисплей**

Для отображения меню используется жидкокристаллический (ЖК) дисплей МТ-12864J-2YLG-3V0 (производство компании «МЭЛТ», г. Москва), расположенный на отладочной плате[5].

Модуль содержит ОЗУ для хранения данных, выводимых на ЖКИ, размером 64х64х2 бит (по 64х64 бит на каждый кристалл). Для выбора нужного кристалла используются выводы E1,E2. ОЗУ разбито на 8 страниц размером по 64х8 бит каждая. Каждой светящейся точке на ЖКИ соответствует логическая «1» в ячейке ОЗУ модуля.

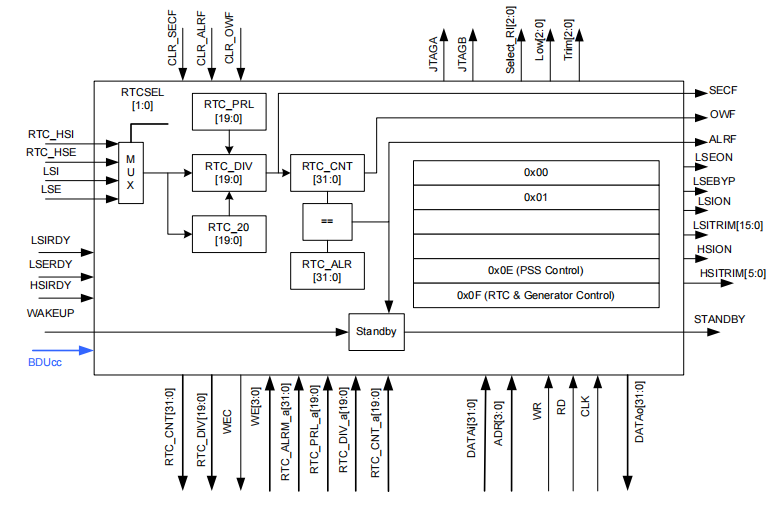


Рисунок 6 – Структурная блок-схема батарейного домена и часов реального времени

Таблица 6 – Описание регистров блока батарейного домена.

|  |  |  |
| --- | --- | --- |
| Базовый адрес | Название | Описание |
| 0x400D\_8000 | MDR\_BKP | Контроллер батарейного домена и часов реального времени |
| Смещение |  |  |
| 0x00…0x34 | REG\_00…REG\_0D | Регистр аварийного сохранения 0…13 |
| 0x38 | REG\_0E | Регистр MDR\_BKP->REG\_0E аварийного сохранения 14 |
| 0x3C | REG\_0F | Регистр MDR\_BKP->REG\_0F аварийного сохранения 15 и управления блоками RTC, LSE, LSI и HSI |
| 0x40 | RTC\_CNT | Регистр MDR\_BKP->RTC\_CNT основного счётчика часов реального времени |
| 0x44 | RTC\_DIV | Регистр MDR\_BKP->RTC\_DIV предварительного делителя основного счётчика |
| 0x48 | RTC\_PRL | Регистр MDR\_BKP->RTC\_PRL основания счета предварительного делителя |
| 0x4C | RTC\_ALRM | Регистр MDR\_BKP->RTC\_ALRM значения для сравнения основного счётчика и выработки сигнала ALRF |
| 0x50 | RTC\_CS | Регистр MDR\_BKP->RTC\_CS управления и состояния флагов часов реального времени |

Чтение (запись) информации из (в) модуль осуществляется по страницам (64х8 бит или 64х1 байт). Каждая страница представлена как 64 байта. Для чтения или записи байта данных по произвольному адресу необходимо предварительно установить страницу ОЗУ и установить адрес внутри страницы ОЗУ. Это осуществляется командами «Set Page» и «Set Address» соответственно. После этого можно прочитать или записать байт данных. В режиме чтения данных после команд «Set Page» и «Set Address» необходимо однократно выполнить «пустую» операцию чтения, результат которой не использовать.

Модуль поддерживает непрерывную последовательность операций чтения или записи: после чтения (записи) одного байта счётчик адреса автоматически увеличивается на 1 и модуль готов к новой операции чтения (записи) по следующему адресу без предварительной установки страницы ОЗУ и адреса. Счётчик адреса считает только внутри одной страницы. При достижении адреса 63 следующим значением счетчика будет 0 и т.д.

Между любыми двумя передачами данных или команд, необходимо ожидать сброса флага BUSY в регистре состояния того кристалла, к которому будет обращение.

Программируемые внешние выводы модуля LCD МТ-12864J показаны в таблице 7[6].

Блок схема LCD МТ-12864J показана на рисунке 7.

Временные диаграммы протокола обмена с LCD MT-12864J показаны на рисунке 8.

Описание команд модуля LCD МТ-12864J показаны в таблице 8.

Таблица 7 – Программируемые внешние выводы модуля LCD МТ-12864J.

|  |  |  |
| --- | --- | --- |
| Вывод | Обозначение | Назначение |
| 1 | UCC | Питание модуля (цифровая часть) |
| 2 | GND | Общий вывод (0В) |
| 3 | U0 | Вход питания ЖК панели |
| 4–11 | DB0–DB7 | Шина данных |
| 12 | E1 | Выбор кристалла 1 |
| 13 | E2 | Выбор кристалла 2 |
| 14 | RES | Сброс (начальная установка) |
| 15 | R/W | Выбор: Чтение/Запись |
| 16 | A0 | Выбор: Команды/Данные |
| 17 | E | Стробирование данных |
| 18 | UEE | Выход DC$DC преобразователя |
| 19 | A | + питания подсветки |
| 20 | K | - питания подсветки |

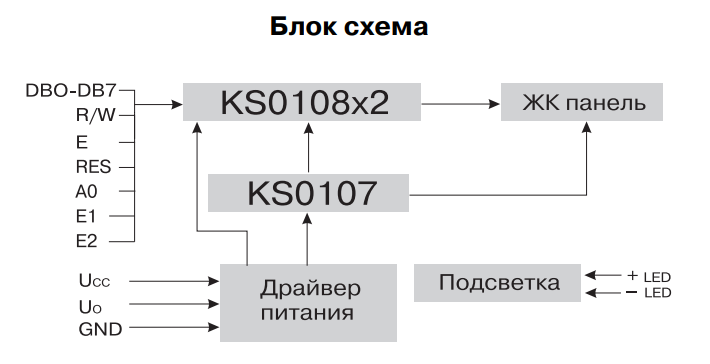


Рисунок 7 – Блок схема LCD MT-12864J

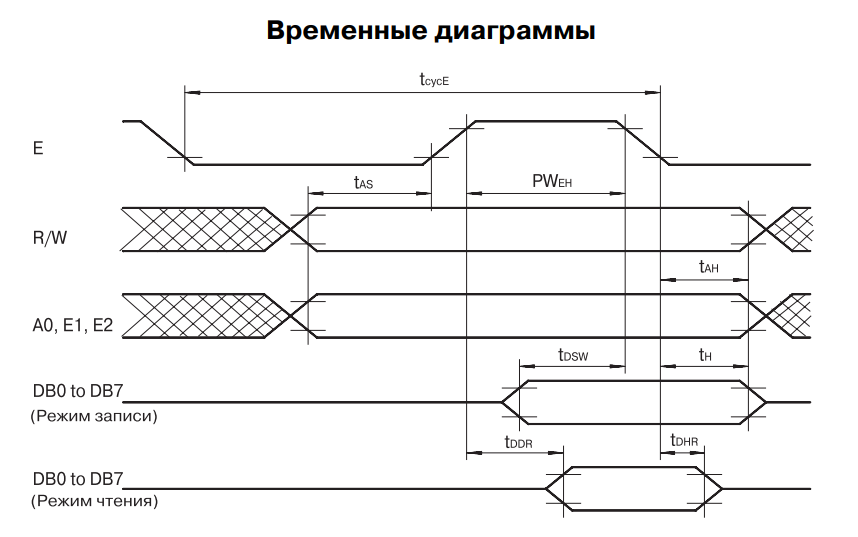
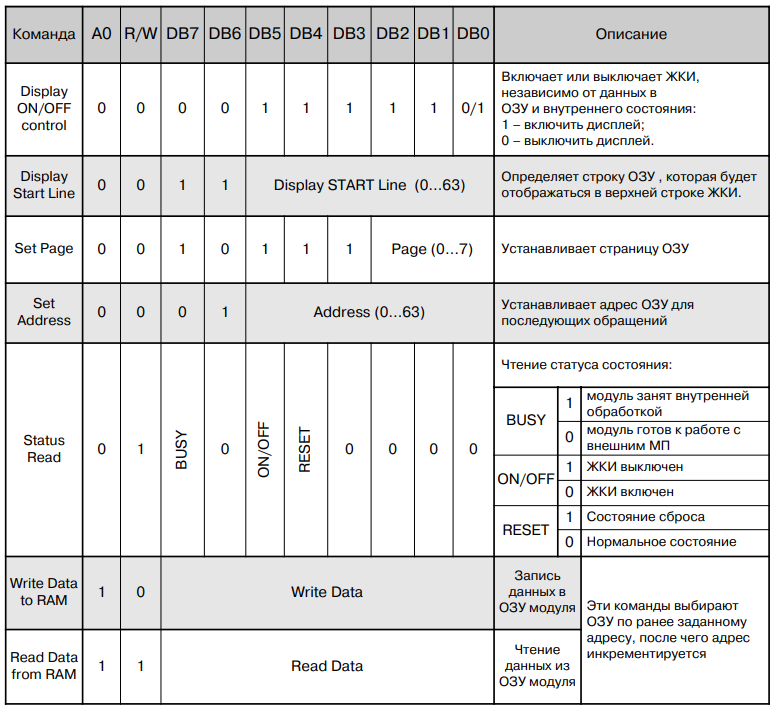


Рисунок 8 – Временные диаграммы протокола обмена с LCD MT-12864J

Таблица 8 – Описание команд модуля LCD МТ-12864J.



**1.2.10 Генератор таковых импульсов**

Для работы МК необходим тактовый генератор. Внутри МК есть встроенный высокочастотный резонатор HSI, обеспечивающий частоту работы 8 МГц. На отладочной плате[5] расположен внешний высокочастотный резонатор HSE, обеспечивающий частоту работы 8 МГц. Для стабильности внешний высокочастотный резонатор подключён к выводам OSC\_IN, OSC\_OUT МК соответственно[3](Приложение Г).

Для работы часов реального времени нужен низкочастотный резонатор, обеспечивающий частоту работы 32,768 Гц. На отладочной плате[5] расположен внешний низкочастотный резонатор LSE, который подключён к портам PE6, PE7 МК соответственно[3](Приложение Г).

**1.2.11 Построение функциональной схемы**

На основе всех вышеописанных сведений была спроектирована функциональная схема разрабатываемой системы, показанная на рисунке 9 и в приложении Б[7][8].

**1.3 Проектирование принципиальной схемы**

**1.3.1 Разъем программатора**

Для программирования МК используется программатор, для его подключения необходим специальный разъём. Будет использован разъем IDC-20. Подключение программатора осуществляется при помощи интерфейса JTAG, под что на МК MDR32F9Q2I задействован порт РD. На принципиальной схеме, которая показана в разделе *1.3.5*, условное обозначение – XP2.

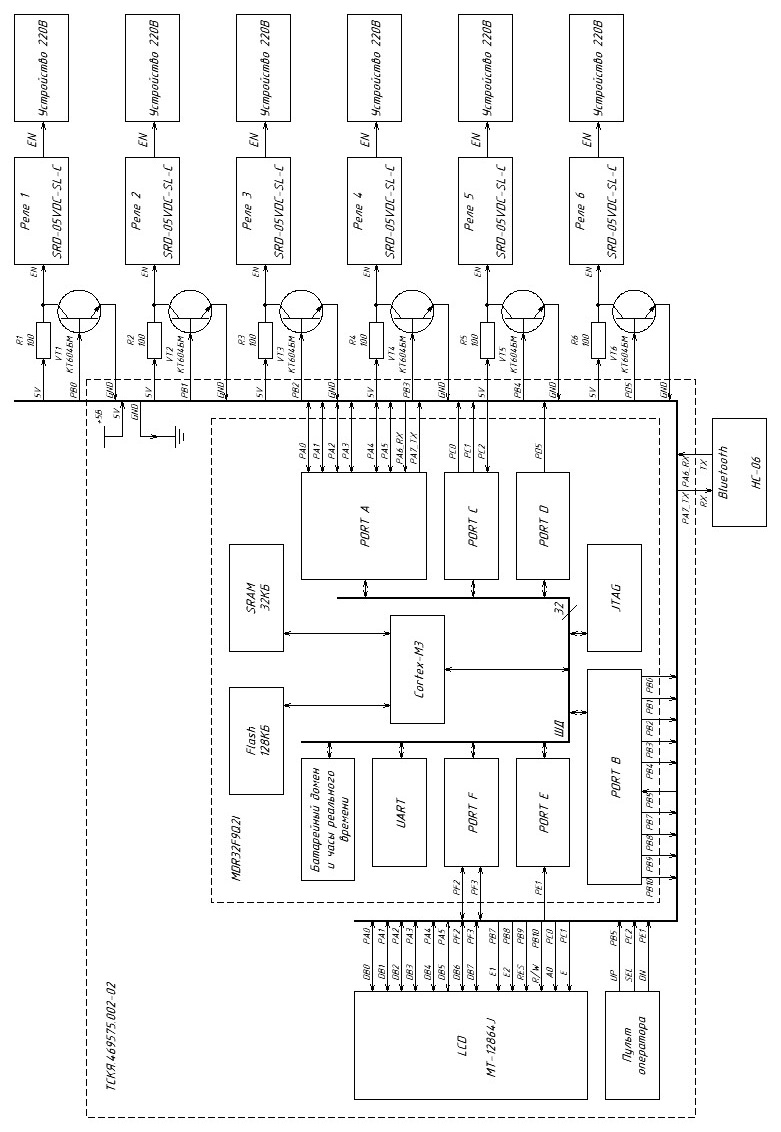


Рисунок 9 – Функциональная схема

Он имеет следующие разъёмы для подключения к МК:

* TMS: Test Mode Select — сигнал управления TAP-контроллером;
* TCK: Test Clock — тактовый сигнал;
* nTRST: Test Reset — сброс, активный уровень сигнала — «0»;
* TDI: Test Data Input — сигнал данных на вход, данные задвигаются по переднему фронту TCK;
* TDO: Test Data Output — выход последовательных данных JTAG, выдвигаются по обратному фронту TCK, должен находиться в третьем состоянии — «Z» — когда данные не передаются.

**1.3.2 Подключение цепи питания**

Для работы схемы, на неё необходимо подать напряжение, для этого будет использован блок питания МТ-ИЭС2-050200, имеющий следующие технические характеристики:

* выходной ток: 2 А;
* выходное напряжение: 5 В;
* входное напряжение: 176–264 В;
* пусковой ток: 30 А.

Для подключения питания к схеме, будет использована MF-2x2MRA вилка на плату угловая 4.2мм. На принципиальной схеме, которая показана в разделе *1.3.5*, условное обозначение – XP1.

**1.3.3 Расчёт сопротивления резисторов**

Резисторы по своей сути предназначаются, чтобы уменьшить значение тока, подаваемого на устройство. В системе резисторы подключены с одной стороны к питанию в 5 вольт, а с другой стороны к разрешающему выводу реле и коллектору транзистора.

Активным сигналом для замыкания реле SRD-05VDC-SL-C является сигнал низкого уровня. Выходной ток для замыкания реле до 75 мА, поэтому решено ограничить ток до 50 мА[9][10].

Определим сопротивление резистора, который предназначен для ограничения величины тока, протекающего через транзисторный ключ на основе транзистора КТ604БМ[11].

= 88 Ом

где U — напряжение питания на резисторе,

I — сила протекающего тока,

Uvt — падение напряжение на переходе коллектор-эмиттер.

Выбран будет резистор с номиналом 100 Ом – CF-100.

Тогда ток через транзисторный ключ будет = 0,044 А

**1.3.4 Расчёт потребляемой мощности**

Потребляемая мощность – это мощность, потребляемая интегральной схемой, которая работает в заданном режиме соответствующего источника питания.

Чтобы рассчитать суммарную мощность, рассчитаем мощность каждого элемента. На все микросхемы подаётся напряжение +5В. Мощность, потребляемая один устройством, в статическом режиме, рассчитывается формулой:

Р = U \* I

где U – напряжение питания (В);

I– ток потребления микросхемы (мА).

Также в схеме присутствуют резисторы и транзисторы. Мощность для резисторов рассчитывается по формуле:

Р = I2 \* R

где R – сопротивление резистора;

I – ток, проходящий через резистор.

Для транзисторов по формуле:

Р = U \* I

где U – напряжение коллектор-эмиттер;

I – ток коллектора (он был рассчитан в разделе *1.3.3*).

Расчёт потребляемого напряжения для каждой микросхемы показан в таблице 9.

Таблица 9 – Потребляемая мощность

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Микросхема | Ток потребления, мА | Потребляемая мощность, мВт | Количе-  ство устройств | Суммарная потребляемая мощность, мВт |
| MDR32F9Q2I | 40 | 200 | 1 | 200 |
| MT–12864J | 4 | 20 | 1 | 20 |
| SRD-05-VDC-SL-C | 20 | 100 | 8 | 800 |
| HC-06 | 45 | 225 | 1 | 225 |

Также в схеме используются 6 резисторов CF-100 с номиналом 100 Ом и 6 транзисторов КТ604БМ.

Pсуммарная = РMDR32F9Q2I + РMT–12864J + РSRD-05-VDC-SL-C + РHC-06 + Ррезисторов + Ртранзисторов = 200 + 20 + 800 + 225 + 6\*194 + 6\*26 = 2565 мВт

Суммарная потребляемая мощность системы равна 2565 мВт = 2,565 Вт.

**1.3.5 Построение принципиальной схемы**

На основе всех вышеописанных сведений была спроектирована принципиальная схема разрабатываемой системы, показанная на рисунке 10 и в приложении Б[7][8].

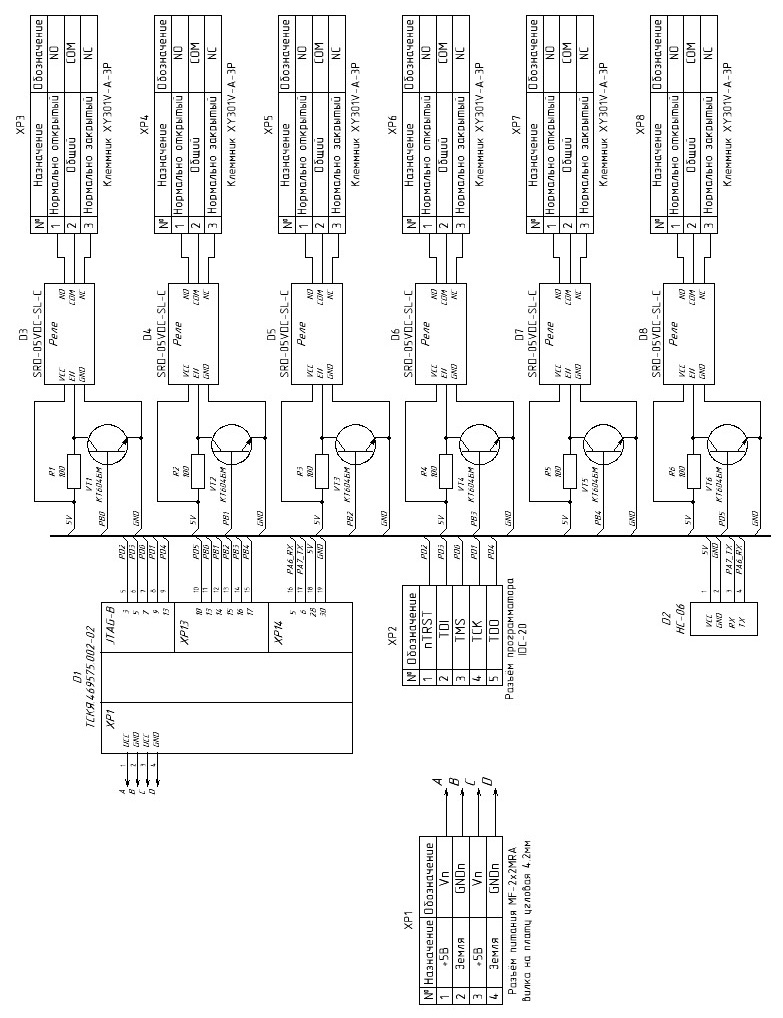


Рисунок 10 – Принципиальная схема

**1.4 Алгоритмы работы системы**

**1.4.1 Функция Main и модули**

Работа начинается с функции main, из которой вызываются все остальные функции. Сначала происходит настройка обработчиков для прерываний UART и часов реального времени. Обработчики — это процедуры, в общем случае реализованные в модулях, отличных от тех, в которых обработчики используются. Затем вызывается процедура App\_init модуля App, который отвечает за общесистемные функции, координируя работу остальных модулей. Модуль App также отвечает за отображение главного меню системы и обработку переключения между разными меню.

В процедуре App\_init вызывается процедура Demo\_init модуля Demo, отвечающей за настройку тактирования портов ввода-вывода, часов реального времени, батарейного домена, контроллера ПЗУ и контроллера UART, а также за настройку работы портов ввода-вывода. Модуль Demo использует библиотечный модуль работы с LCD и пультом оператора. Эти модули были взяты из демонстрационного проекта[12] и адаптированы под текущий проект.

Затем в процедуре App\_init вызывается процедура Time\_init модуля Time, отвечающего за инициализацию, настройку часов реального времени, за предоставление меню изменения текущего времени системы, за обработку конца дня, за программный доступ к текущему времени системы в секундах. Процедура Time\_init выполняет инициализацию обратного вызова возврата из меню и обратного вызова изменения оператором системных часов; обратные вызовы передаются из модуля App. Процедура Time\_init проверяет наличие первого запуска системы с помощью регистра аварийного сохранения из батарейного домена. При первом запуске системное время устанавливается равным 0 и происходит запись ключевого значения в регистр аварийного сохранения. Процедура Time\_init увеличивает счётчик дней, проведённых системой без основного питания, в соответствии с количеством секунд, накопленных в счётчике часов реального времени; устанавливает корректное текущее системное время.

Затем в процедуре App\_init происходит установка стандартных действий меню, реализованных в модуле App, для модуля Pwr с целью уменьшить дублирование кода программ. Модуль Pwr отвечает за включение и выключение внешних устройств, подключённых к реле; сбор статистики работы устройств; формирование строки статистики; формирование меню для взаимодействия оператора с устройствами и отслеживания их статистики. Процедура Pwr\_init инициализирует порты ввода-вывода для открытия всех подключённых реле, инициализирует обратные вызовы возврата из меню и обратный вызов обновления состояния одного из устройств.

Затем в процедуре App\_init вызывается процедура Uart\_init модуля Uart, который отвечает за программную реализацию кольцевых буферов байт для обмена данными через интерфейс Uart. Процедура Uart\_init инициализирует индексы начала и конца занятой области массива нулями (означает, что буфер пуст), инициализирует контроллер UART в соответствии с разделом *1.2.7*. Модуль Uart предоставляет программный интерфейс работы с аппаратными буферами контроллера UART. Данный модуль предоставляет процедуру Uart\_sendData заполнения аппаратного буфера отправки данными из программного буфера записи и процедуру Uart\_receiveData программного буфера чтения данными из аппаратного буфера приёма. Процедуры Uart\_sendData и Uart\_receiveData имеют программную защиту от прерываний: если прерывание вызвало одну из процедур, когда основной цикл программ уже выполняет данную процедуру, то прерывание выйдет из процедуры. Защита нужна для сохранения целостности программных буферов; реализовано через статическую переменную внутри процедуры (значение флага остаётся неизменным между вызовами процедур).

Затем вызывается процедура Eeprom\_init модуля Eeprom, который отвечает за предоставление массива свободной для изменения области памяти, расположенной в ПЗУ, а также за интерфейс чтения/записи в ПЗУ. Процедура Eeprom\_init устанавливает задержку в 1 такт для гарантии, что контроллер ПЗУ успеет считать или записать данные. Модуль Eeprom гарантирует, что при записи в ПЗУ не будут повреждены коды программ.

Затем происходит инициализация главного меню, реализованного модулем App, и история переходов между меню. В процедуре App\_init выполняется проверка валидности текущего меню (в данном случае главного меню), т.е. история переходов не должна быть длиннее заданной константы и каждое меню в истории переходов должно быть валидным (используется функция проверки из модуля Menu). Модуль Menu предоставляет программный интерфейс для создания и отображения меню списочного типа. Каждое меню обладает заголовком и имеет список (набор строк) пунктов меню. Между пунктами меню можно переключаться вверх и вниз (текущий пункт выделен) и выбирать каждый из пунктов. Для каждого пункта меню может быть задано подменю, на которое произойдёт переключение в случае выбора пункта меню. Для каждого пункта меню может быть задано действие (процедура), которое выполнится перед переключение на под меню, если таковое имеется, или выполнится без переключения. Например, в меню изменения системного времени можно выбрать пункт меню, отвечающий за установку часа/минуты/секунды и будет выполнена замена обработчиков кнопок «вверх» и «вниз» для выбора количества часов/минут/секунд. Модуль меню предоставляет функцию проверки валидности меню и функций визуального переключения между пунктами меню, а также функцию выделения одного из пунктов меню или снятия выделения с одного из пунктов меню.

Модуль Lcd является реализацией шаблона проектирования «фасад» для библиотечного модуля LCD, взятого из демонстрационного проекта[12]. Данный модуль предоставляет адаптированные функции работы с ЖК-дисплеем как с набором из пяти строк разного форматирования.

После инициализации системы в функции Main вызовется процедура App\_run, которая запускает главный цикл системы. В главном цикле системы выполняется считывание данных с пульта оператора и выполнения соответствующего действия. Обработчики ввода с пульта оператора определяются каждым меню отдельно. Например, в главном меню обработчики ввода с пульта оператора выполняются переключение между пунктами меню. После обработки ввода с пульта оператора происходит циклический вызов процедуры обновления состояния системы и чтение ввода с пульта оператора. Выход из цикла происходит тогда, когда считанный ввод с пульта оператора отличается от предыдущего считанного ввода с пульта оператора. Гарантируется хотя бы одно обновление состояния системы на каждой итерации главного цикла системы.

Процедура обновления состояния системы вызывает процедуру обработки команд, принятых через интерфейс UART. После проверяется флаг на обновление меню, который выставляется по прерыванию от часов реального времени каждую секунду, и если флаг установлен, то вызывается процедура обновления текущего меню (в модуле Pwr эта процедура также используется для обновления статистики по устройствам) и флаг сбрасывается.

В модуле App реализовано меню для сохранения текущей статистики по каждому устройству в ПЗУ МК.

Схема алгоритма функции Main и процедуры инициализации системы показана на рисунке 11.

Схема алгоритма главного цикла системы показана на рисунке 12.

Схема алгоритма обновления состояния системы и обновления статистики показана на рисунке 13.

Схема алгоритма отправки данных показана на рисунке 14.

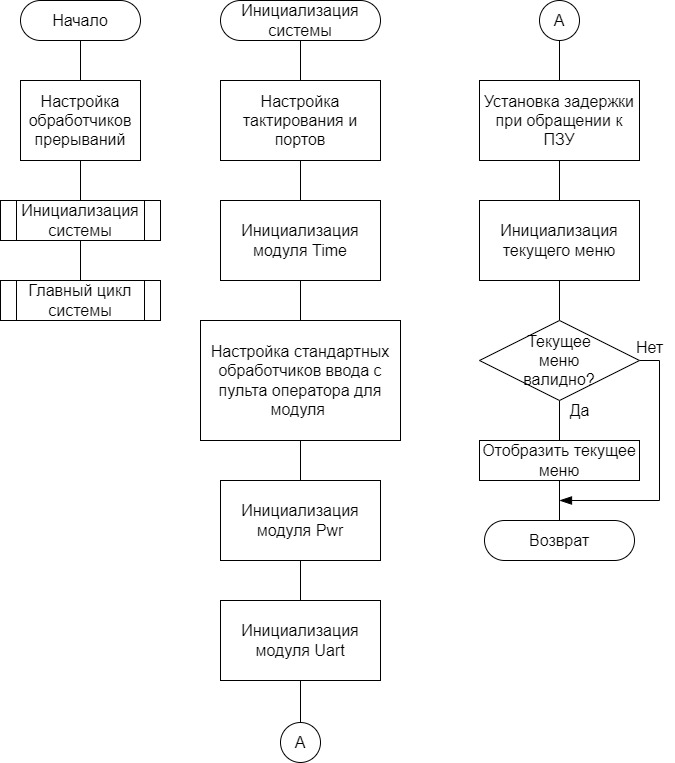


Рисунок 11 — Схема алгоритма функции Main и процедуры инициализации системы

**1.4.2 Цифровой автомат обработки команд**

Обработка команд полученных через интерфейс UART представляет собой примитивный конечный цифровой автомат, позволяющий включить или выключить внешнее устройство и получить статистику по всем внешним устройствам. Система отправит возможные варианты правильных команд, если были получены неизвестные символы; если команда состоит из нескольких символов, то при ошибке необходимо ввести команду с начала.

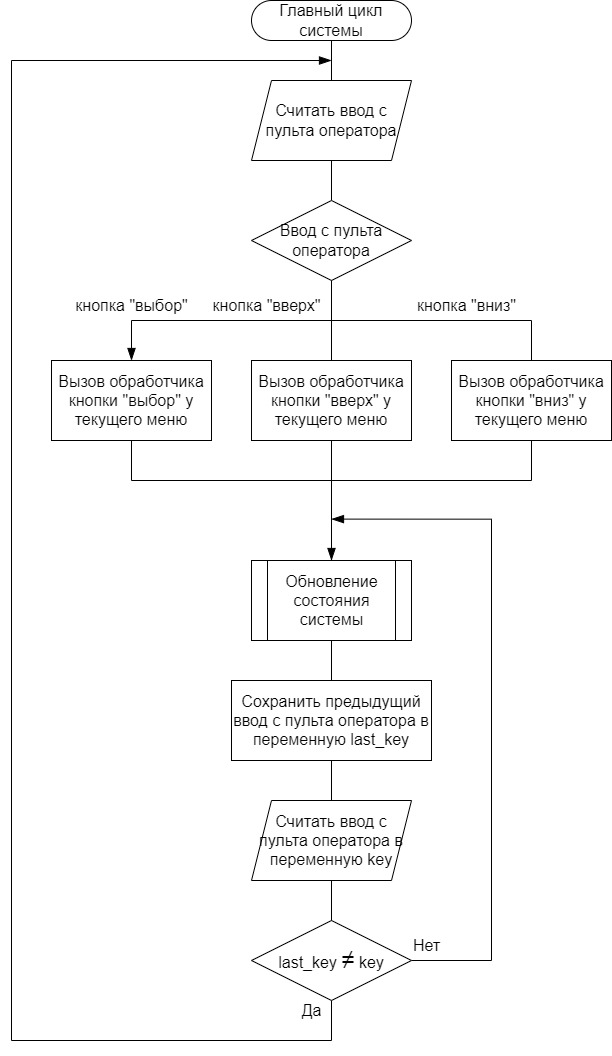


Рисунок 12 — Схема алгоритма главного цикла системы

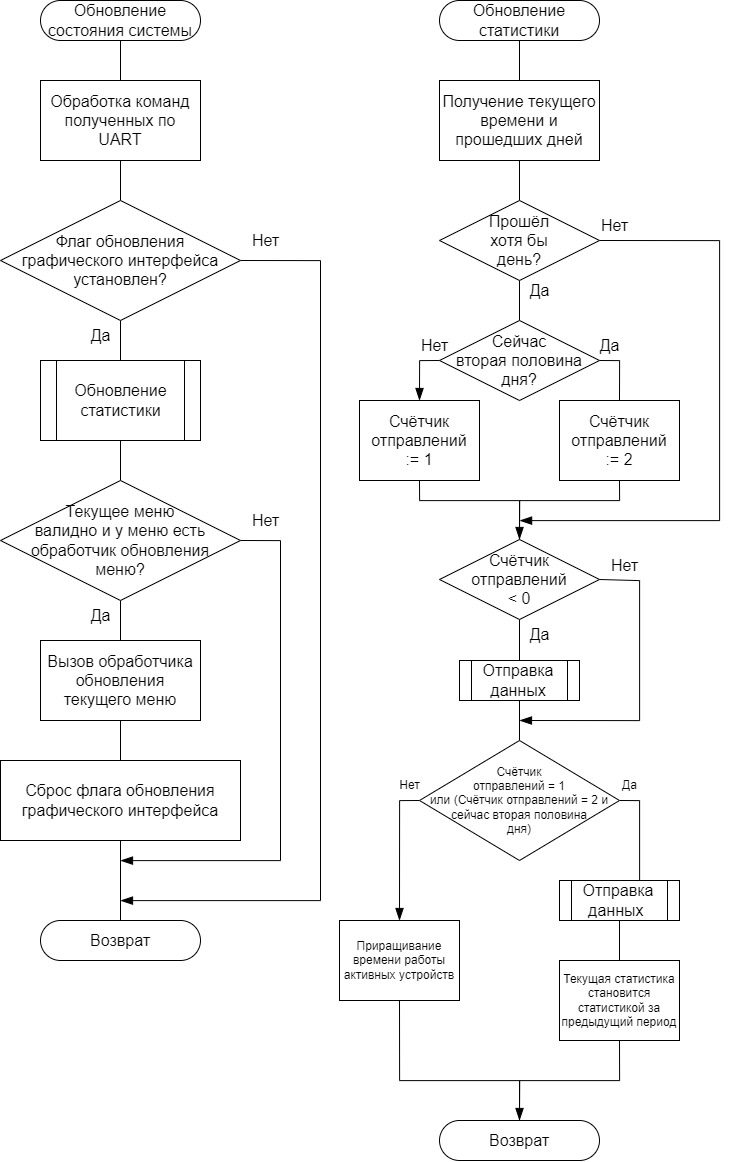


Рисунок 13 — Схема алгоритма обновления состояния системы и обновления статистики

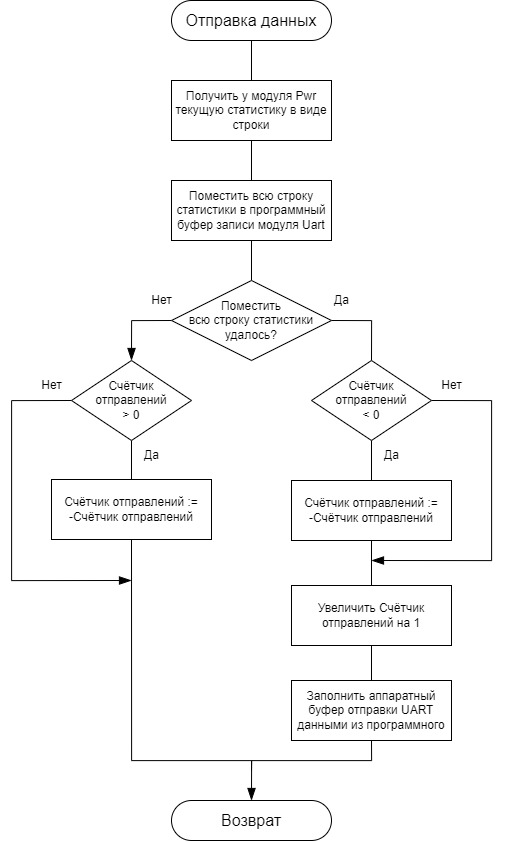


Рисунок 14 — Схема алгоритма отправки данных

Схема цифрового автомата обработки команд представлена рисунке 15.

**2 Технологическая часть**

Для реализации работы системы была написана программа на языке Си, после загруженная в МК. Отладка проводилась в программе Keil MDK 5 с использованием отладочной платы[5].

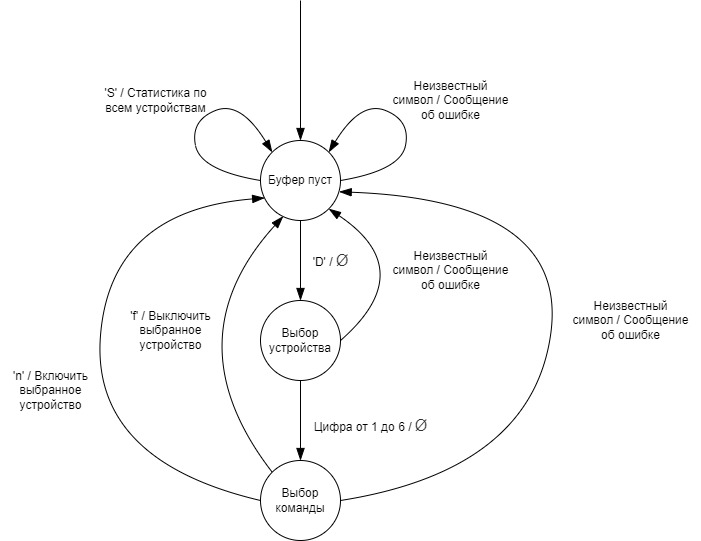


Рисунок 15 — Схема цифрового автомата обработки команд

**2.1 Отладка и тестирование программы**

Программа была отлажена с использованием приложения Keil MDK 5 и отладочной платы. Keil MDK 5 — это комплексное решение для разработки программного обеспечения для микроконтроллеров на базе Arm. Оно включает в себя все компоненты, необходимые для создания, сборки и отладки встроенных приложений.

При написании кода были использованы следующие библиотеки:

⎯MDR32FxQI\_rst\_clk.h – это стандартная библиотека, которая объяснить компилятору как работать с периферийный блоком формирования тактовой частоты, какие периферийные блоки требуют тактирования и какие генераторы тактовой частоты существуют;

⎯MDR32FxQI\_port.h – это стандартная библиотека, которая объяснит компилятору какие порты ввода/вывода есть у микроконтроллера, как они обозначены и на что они способны;

⎯stdlib.h – это стандартная библиотека C общего назначения, включающая в себя функции, занимающиеся распределением памяти, управлением процессами, преобразованием и др.;

⎯MDR32FxQI\_uart.h – это стандартная библиотека, которая объяснит компилятору какие контроллеры UART есть у микроконтроллера, как они обозначены и на что они способны.

⎯MDR32FxQI\_eeprom.h – это библиотека, которая позволяет работать с ПЗУ микроконтроллера, стирать и программировать Flash память;

⎯MDR32FxQI\_bkp.h – это библиотека, которая позволяет работать с батарейным доменом микроконтроллера и часами реального времени;

После компиляции создаётся файл с расширением “.hex”, объем которого равен 40,4 килобайта – столько занимает скомпилированная программа.

По итогу отладки и тестирования, результатом стала функционирующая система, работающая в соответствии с ТЗ.

**2.2 Способы программирования МК**

После написания и тестирования кода в программе, в который все – это виртуальная модель, идёт этап загрузки файла (с расширением hex – бинарный файл) в микроконтроллер. Это может выполнятся с помощью отладочного интерфейса JTAG[3]: Программирование МК происходит через программатор и одноимённый разъем, о котором было рассказано в разделе *1.3.1*.

Прошивка проходит по интерфейсу JTAG, для работы программатора нужно 4 контакта и питание (достаточно только земли, чтобы уравнять потенциалы земель программатора и устройства):

* TMS: Test Mode Select — сигнал управления TAP-контроллером;
* TCK: Test Clock — тактовый сигнал;
* TDI: Test Data Input — сигнал данных на вход, данные задвигаются по переднему фронту TCK;
* TDO: Test Data Output — выход последовательных данных JTAG, выдвигаются по обратному фронту TCK, должен находиться в третьем состоянии — «Z» — когда данные не передаются;
* GND – земля;
* VCC – питание.

Передача по JTAG осуществляется в полнодуплексном режиме, по одному биту за такт в каждую сторону. По возрастающему фронту сигнала TCK тестируемое устройство считывает очередной бит с линии TDI, а по спадающему – выдаёт следующий бит на линию TDO.

В МК передаётся бинарный файл с расширением “.hex” с скомпилированной программой.

**Заключение**

В результате выполнения курсовой работы был создан проект — многоканальная регистрирующая МК-система. Система работает на основе МК серии «Миландр» — К1986ВЕ92QI. Устройство разработано в соответствии с ТЗ.

В процессе работы над курсовой работой была разработана схема электрическая функциональная и принципиальная, спецификация и документация к устройству. Исходный код программы, написанный на языке С, отлажен и протестирован при помощи интегрированной среды разработки Keil MDK 5 и отладочной платы.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Сравнение отечественных МК : [Электронный ресурс] — URL: <https://chipinfo.pro/mcu/ruscomparison.shtml> (дата обращения: 30.11.2022)

2. AVR против STM32 : [Электронный ресурс] — URL: <http://www.avrki.ru/articles/content/avr_stm/> (дата обращения: 30.11.2022)

3. К1986ВЕ92QI — характеристики, документация : [Электронный ресурс] — URL: <https://ic.milandr.ru/products/mikrokontrollery_i_protsessory/32_razryadnye_mikrokontrollery/k1986ve92qi/> (дата обращения: 30.11.2022)

4. Документация на модуль HC-06 [Электронный ресурс]. — URL: <https://www.olimex.com/Products/Components/RF/BLUETOOTH-SERIAL-HC-06/resources/hc06.pdf> (дата обращения: 30.11.2022)

5. Отладочный комплект для микроконтроллера К1986ВЕ92QI : [Электронный ресурс] — URL: <https://ic.milandr.ru/products/programmno_otladochnye_sredstva/otladochnye_komplekty/otladochnyy-komplekt-dlya-mikrokontrollera-k1986ve92qi/> (дата обращения: 30.11.2022)

6. Документация на жидкокристаллический модуль MT–12864J [Электронный ресурс]. — URL: <http://files.amperka.ru/datasheets/MT-12864J.pdf> (дата обращения: 30.11.2022)

7. ГОСТ 2.743-91 ЕСКД ОБОЗНАЧЕНИЯ БУКВЕННО-ЦИФРОВЫЕ В ЭЛЕКТРИЧЕСКИХ СХЕМАХ [Электронный ресурс]. — URL: <https://docs.cntd.ru/document/1200001985> (дата обращения: 30.11.2022)

8. ГОСТ 2.721-74 ЕСКД ОБОЗНАЧЕНИЯ УСЛОВНЫЕ ГРАФИЧЕСКИЕ В СХЕМАХ [Электронный ресурс]. — URL: <https://docs.cntd.ru/document/1200007058> (дата обращения: 30.11.2022)

9. Транзисторный ключ \* diodov.net | Электроника для всех [Электронный ресурс]. — URL: <https://diodov.net/tranzistornyj-klyuch/> (дата обращения: 30.11.2022)

10. Модуль реле SRD-05-VDC-SL-C [Электронный ресурс]. — URL: <https://iarduino.ru/shop/Expansion-payments/rele-elektromehanicheskoe-do-250v-10-a-4--kanal-5v.html> (дата обращения: 30.11.2022)

11. КТ604БМ, Транзистор NPN средней мощности [Электронный ресурс]. — URL: <https://www.chipdip.ru/product/kt604bm> (дата обращения: 30.11.2022)

12. Демонстрационные проекты для МК К1986ВЕ92QI [Электронный ресурс]. — URL: <https://support.milandr.ru/base/primenenie/programmirovanie-32-razryadnykh-mk/osobennosti-primeneniya-know-how/48536/> (дата обращения: 30.11.2022)

**Приложение А**Текст программы

Проект Keil MDK 5 расположен по адресу <https://github.com/AntonBush/pwr_mng/tree/master/src>.

Объем исполняемого кода – 1250 строк.

Main.c:

#include "joystick.h"

#include "module\_app.h"

#include "module\_uart.h"

#include "module\_interrupt.h"

int main(void)

{

Interrupt\_UartReceiveHandler = Uart\_receiveData;

Interrupt\_UartSendHandler = Uart\_sendData;

Interrupt\_UartLineStateHandler = NULL;

Interrupt\_RtcSecondHandler = App\_rtcSecondHandle;

App\_init();

App\_run();

}

module\_app.h:

#ifndef MODULE\_APP

#define MODULE\_APP

void App\_init(void);

void App\_run(void);

void App\_rtcSecondHandle(void);

#endif // MODULE\_APP

module\_app.c:

#include "module\_app.h"

#include "module\_demo.h"

#include "module\_menu.h"

#include "module\_pwr.h"

#include "module\_time.h"

#include "module\_interrupt.h"

#include "module\_uart.h"

#include "module\_eeprom.h"

#include "joystick.h"

#include <stddef.h>

#include <string.h>

// Private function prototypes

static void App\_up(void);

static void App\_select(void);

static void App\_down(void);

static void App\_update(void);

static void App\_updateGui(void);

static void App\_upProc(void);

static void App\_selectProc(void);

static void App\_downProc(void);

static void App\_returnProc(void);

static void App\_sendStatsProc(void);

static bool App\_isValidCurrentMenu(void);

static void App\_sendStats(void);

static void App\_updateStats(void);

static void App\_resetStats(uint32\_t old\_time);

static void App\_resolveCommands(void);

static void App\_eepromRead13(void);

static void App\_eepromRead46(void);

static void App\_eepromUpdateCurrent(void);

// Variables

#define APP\_\_MAX\_MENU\_LEVELS 4

static Menu\_Menu \*App\_CurrentMenu;

static Menu\_Menu \*App\_PreviousMenus[APP\_\_MAX\_MENU\_LEVELS];

static size\_t App\_MenuLevel;

static volatile bool App\_UpdateGuiSoon = FALSE;

static uint8\_t App\_EepromWorktimeStrs[][18] = {

{'0', ':', ' ', '.', '.', 'h', '.', '.', 'm', '.', '.', 's', '\0'},

{'0', ':', ' ', '.', '.', 'h', '.', '.', 'm', '.', '.', 's', '\0'},

{'0', ':', ' ', '.', '.', 'h', '.', '.', 'm', '.', '.', 's', '\0'}};

// d: 0

// h: 3, 4

// m: 6, 7

// s: 9, 10

/\* EEPROM menu \*/

static Menu\_MenuItem App\_Eeprom16MenuItems[] = {{App\_EepromWorktimeStrs[0], NULL, App\_eepromUpdateCurrent},

{App\_EepromWorktimeStrs[1], NULL, App\_eepromUpdateCurrent},

{App\_EepromWorktimeStrs[2], NULL, App\_eepromUpdateCurrent},

{"Return", NULL, App\_returnProc}};

static Menu\_Menu App\_Eeprom16Menu = {"EEPROM devices",

App\_Eeprom16MenuItems,

UTILITY\_\_COUNT\_OF(App\_Eeprom16MenuItems),

0,

App\_upProc,

App\_selectProc,

App\_downProc,

NULL};

static Menu\_MenuItem App\_EepromMenuItems[] = {{"Devices 1-3", &App\_Eeprom16Menu, App\_eepromRead13},

{"Devices 4-6", &App\_Eeprom16Menu, App\_eepromRead46},

{"Return", NULL, App\_returnProc}};

static Menu\_Menu App\_EepromMenu = {"EEPROM devices",

App\_EepromMenuItems,

UTILITY\_\_COUNT\_OF(App\_EepromMenuItems),

0,

App\_upProc,

App\_selectProc,

App\_downProc,

NULL};

/\* Main menu \*/

static Menu\_MenuItem App\_MainMenuItems[] = {{"Devices", &Pwr\_DeviceMenu, NULL},

{"SetTime", &Time\_SetTimeMenu, NULL},

{"Force Send", NULL, App\_sendStatsProc},

{"EEPROM", &App\_EepromMenu, NULL}};

static Menu\_Menu App\_MainMenu = {"Main menu",

App\_MainMenuItems,

UTILITY\_\_COUNT\_OF(App\_MainMenuItems),

0,

App\_upProc,

App\_selectProc,

App\_downProc,

NULL};

static int App\_SendStatsCounter = 1;

// Public functions

void App\_init(void)

{

Demo\_init();

Time\_init(App\_returnProc, App\_resetStats);

Pwr\_StdUpProc = App\_upProc;

Pwr\_StdSelectProc = App\_selectProc;

Pwr\_StdDownProc = App\_downProc;

Pwr\_init(App\_returnProc, App\_updateStats);

Uart\_init();

Eeprom\_init();

App\_CurrentMenu = &App\_MainMenu;

App\_MenuLevel = 0;

App\_PreviousMenus[App\_MenuLevel] = App\_CurrentMenu;

if (App\_isValidCurrentMenu()) Menu\_displayMenu(App\_CurrentMenu);

}

void App\_run(void)

{

uint32\_t key = NOKEY;

uint32\_t last\_key = NOKEY;

while (TRUE) {

switch (key) {

case SEL:

App\_select();

break;

case UP:

App\_up();

break;

case DOWN:

App\_down();

break;

}

do {

App\_update();

last\_key = key;

key = GetKey();

} while (key == last\_key);

}

}

void App\_rtcSecondHandle(void)

{

uint32\_t time;

App\_updateGui();

/\* If counter is equal to 86400: one day was elapsed \*/

time = Time\_getRawTime();

if (time == 24 \* 60 \* 60) {

Time\_setRawTime(0);

Time\_addPassedDay();

}

}

// Private functions

void App\_up(void)

{

if (!App\_isValidCurrentMenu() || App\_CurrentMenu->up == NULL) return;

App\_CurrentMenu->up();

}

void App\_select(void)

{

if (!App\_isValidCurrentMenu() || App\_CurrentMenu->select == NULL) return;

App\_CurrentMenu->select();

}

void App\_down(void)

{

if (!App\_isValidCurrentMenu() || App\_CurrentMenu->down == NULL) return;

App\_CurrentMenu->down();

}

void App\_update(void)

{

App\_resolveCommands();

if (App\_UpdateGuiSoon) {

App\_updateStats();

if (!App\_isValidCurrentMenu() || App\_CurrentMenu->update == NULL) return;

App\_CurrentMenu->update();

App\_UpdateGuiSoon = FALSE;

}

}

void App\_updateGui(void)

{

App\_UpdateGuiSoon = TRUE;

}

// Common procedures for menu interaction

void App\_upProc(void)

{

if (!App\_isValidCurrentMenu()) return;

Menu\_selectUpperItem(App\_CurrentMenu);

}

void App\_selectProc(void)

{

Menu\_MenuItem \*menu\_item;

if (!App\_isValidCurrentMenu()) return;

menu\_item = &App\_CurrentMenu->items[App\_CurrentMenu->item\_index];

if (menu\_item->proc != NULL) {

menu\_item->proc();

}

if (App\_MenuLevel < APP\_\_MAX\_MENU\_LEVELS - 1 && menu\_item->submenu != NULL) {

App\_CurrentMenu = menu\_item->submenu;

Menu\_displayMenu(App\_CurrentMenu);

++App\_MenuLevel;

App\_PreviousMenus[App\_MenuLevel] = App\_CurrentMenu;

}

}

void App\_downProc(void)

{

if (!App\_isValidCurrentMenu()) return;

Menu\_selectLowerItem(App\_CurrentMenu);

}

void App\_returnProc(void)

{

if (App\_MenuLevel == 0) return;

App\_CurrentMenu = App\_PreviousMenus[App\_MenuLevel - 1];

--App\_MenuLevel;

Menu\_displayMenu(App\_CurrentMenu);

}

void App\_sendStatsProc(void)

{

Uart\_putString(Pwr\_getStats());

Uart\_sendData();

}

bool App\_isValidCurrentMenu(void)

{

size\_t i;

if (App\_CurrentMenu == NULL || APP\_\_MAX\_MENU\_LEVELS <= App\_MenuLevel) return FALSE;

for (i = 0; i <= App\_MenuLevel; ++i) {

if (App\_PreviousMenus[i] == NULL || !Menu\_isValidMenu(App\_PreviousMenus[i])) return FALSE;

}

return TRUE;

}

void App\_sendStats(void)

{

if (Uart\_putWholeString(Pwr\_getStats())) {

if (App\_SendStatsCounter < 0) {

App\_SendStatsCounter = -App\_SendStatsCounter + 1;

} else {

++App\_SendStatsCounter;

}

Uart\_sendData();

} else if (0 < App\_SendStatsCounter) {

App\_SendStatsCounter = -App\_SendStatsCounter;

}

}

#define APP\_\_TWELVE\_HOURS (12 \* 60 \* 60)

void App\_updateStats(void)

{

uint32\_t time, n\_passed\_days;

time = Time\_getRawTime();

n\_passed\_days = Time\_getNPassedDays();

if (0 < n\_passed\_days) {

if (APP\_\_TWELVE\_HOURS <= time) {

App\_SendStatsCounter = 2;

} else {

App\_SendStatsCounter = 1;

}

}

if (App\_SendStatsCounter < 0) {

App\_sendStats();

}

if ((App\_SendStatsCounter == 1) || (App\_SendStatsCounter == 2 && APP\_\_TWELVE\_HOURS <= time)) {

App\_sendStats();

Pwr\_checkPoint(time);

} else {

Pwr\_updateStats(time);

}

}

void App\_resetStats(uint32\_t old\_time)

{

uint32\_t new\_time = Time\_getRawTime();

if (APP\_\_TWELVE\_HOURS <= new\_time) {

App\_SendStatsCounter = 2;

} else {

App\_SendStatsCounter = 1;

}

Pwr\_updateStats(old\_time);

Pwr\_resetStats(new\_time);

}

typedef enum {

App\_ResolveCommandState\_init,

App\_ResolveCommandState\_device,

App\_ResolveCommandState\_device\_x

} App\_ResolveCommandState;

// S - send stats

// D0n - turn Device 0 oN

// other - clear buffer

void App\_resolveCommands(void)

{

static App\_ResolveCommandState state = App\_ResolveCommandState\_init;

static unsigned int chosen\_device;

Uart\_MaybeReceivedChar maybe;

Uart\_receiveData();

while (maybe = Uart\_getChar(), maybe.received) {

uint8\_t ch = maybe.received\_data.ch;

if (maybe.received\_data.error != UART\_\_RECIEVE\_ERROR\_NO\_ERROR) {

continue;

}

if (state == App\_ResolveCommandState\_init) {

if (ch == 'S') {

App\_sendStatsProc();

} else if (ch == 'D') {

state = App\_ResolveCommandState\_device;

} else {

Uart\_putString("Unknown command: ");

Uart\_putChar(ch);

Uart\_putString("\r\nAvailable: 'S' - send stats, 'D' - choose device\r\n");

}

} else if (state == App\_ResolveCommandState\_device) {

if ('0' < ch && ch < '7') {

chosen\_device = ch - '1';

state = App\_ResolveCommandState\_device\_x;

} else {

state = App\_ResolveCommandState\_init;

Uart\_putString("Unknown device number: ");

Uart\_putChar(ch);

Uart\_putString("\r\nAvailable: [1, 6]\r\n");

}

} else if (state == App\_ResolveCommandState\_device\_x) {

int old\_device = Pwr\_currentDevice();

Pwr\_setCurrentDevice(chosen\_device);

if (ch == 'n') {

Pwr\_turnDeviceOn();

} else if (ch == 'f') {

Pwr\_turnDeviceOff();

} else {

Uart\_putString("Unknown turn command: ");

Uart\_putChar(ch);

Uart\_putString("\r\nAvailable: 'n' - on, 'f' - off\r\n");

}

state = App\_ResolveCommandState\_init;

Pwr\_setCurrentDevice(old\_device);

}

}

Uart\_sendData();

}

// h: 3, 4

// m: 6, 7

// s: 9, 10

static void App\_eepromRead(int x)

{

int i;

int device\_index;

int data\_index;

for (i = 0, device\_index = x + 1, data\_index = 6 \* x;

i < 3;

++i, ++device\_index) {

App\_EepromWorktimeStrs[i][0] = '0' + device\_index;

// h

App\_EepromWorktimeStrs[i][3] = Eeprom\_readByte(data\_index++);

App\_EepromWorktimeStrs[i][4] = Eeprom\_readByte(data\_index++);

// m

App\_EepromWorktimeStrs[i][6] = Eeprom\_readByte(data\_index++);

App\_EepromWorktimeStrs[i][7] = Eeprom\_readByte(data\_index++);

// s

App\_EepromWorktimeStrs[i][9] = Eeprom\_readByte(data\_index++);

App\_EepromWorktimeStrs[i][10] = Eeprom\_readByte(data\_index++);

}

}

void App\_eepromRead13(void)

{

App\_eepromRead(0);

}

void App\_eepromRead46(void)

{

App\_eepromRead(3);

}

void App\_eepromUpdateCurrent(void)

{

int old\_device;

Time\_TimeEdit time;

int device\_index;

int data\_index;

int str\_index = App\_Eeprom16Menu.item\_index;

device\_index = App\_EepromWorktimeStrs[str\_index][0] - '0' - 1;

old\_device = Pwr\_currentDevice();

Pwr\_setCurrentDevice(device\_index);

time = Pwr\_currentDeviceTimeEdit();

Pwr\_setCurrentDevice(old\_device);

data\_index = 6 \* device\_index;

Eeprom\_writeByte(App\_EepromWorktimeStrs[str\_index][3] = Utility\_uintToChar(time.hour / 10 % 10), data\_index++);

Eeprom\_writeByte(App\_EepromWorktimeStrs[str\_index][4] = Utility\_uintToChar(time.hour % 10), data\_index++);

// m

Eeprom\_writeByte(App\_EepromWorktimeStrs[str\_index][6] = Utility\_uintToChar(time.minute / 10 % 10), data\_index++);

Eeprom\_writeByte(App\_EepromWorktimeStrs[str\_index][7] = Utility\_uintToChar(time.minute % 10), data\_index++);

// s

Eeprom\_writeByte(App\_EepromWorktimeStrs[str\_index][9] = Utility\_uintToChar(time.second / 10 % 10), data\_index++);

Eeprom\_writeByte(App\_EepromWorktimeStrs[str\_index][10] = Utility\_uintToChar(time.second % 10), data\_index++);

Menu\_displayMenu(App\_CurrentMenu);

}

module\_menu.h:

#ifndef MODULE\_MENU

#define MODULE\_MENU

#include "module\_utility.h"

#include "module\_lcd.h"

#include <stddef.h>

typedef struct Menu\_Menu Menu\_Menu;

typedef struct

{

uint8\_t \*title;

struct Menu\_Menu \*submenu;

Utility\_Procedure \*proc;

} Menu\_MenuItem;

struct Menu\_Menu {

uint8\_t \*title;

Menu\_MenuItem \*items;

size\_t n\_items;

size\_t item\_index;

Utility\_Procedure \*up;

Utility\_Procedure \*select;

Utility\_Procedure \*down;

Utility\_Procedure \*update;

};

void Menu\_highlightLine(Lcd\_Line line);

void Menu\_displayMenu(Menu\_Menu \*menu);

void Menu\_selectUpperItem(Menu\_Menu \*menu);

void Menu\_selectLowerItem(Menu\_Menu \*menu);

int Menu\_isValidMenu(Menu\_Menu \*menu);

#endif // MODULE\_MENU

module\_menu.c:

#include "module\_menu.h"

#include "module\_lcd.h"

void Menu\_highlightLine(Lcd\_Line line)

{

Lcd\_displayString(line,

" ",

LCD\_\_STYLE\_NO\_OVERRIDE | LCD\_\_STYLE\_HIGHLIGHT);

}

int Menu\_isValidMenu(Menu\_Menu \*menu)

{

return !(menu == NULL || menu->items == NULL || menu->n\_items <= menu->item\_index);

}

void Menu\_displayMenu(Menu\_Menu \*menu)

{

size\_t i;

Menu\_MenuItem \*menu\_item;

if (!Menu\_isValidMenu(menu)) return;

Lcd\_clear();

/\* Display menu header \*/

Lcd\_displayString(Lcd\_Line\_line0, menu->title, LCD\_\_STYLE\_NO\_OVERRIDE);

Lcd\_displayLine(Lcd\_Line\_line1 - 3);

/\* Display menu items \*/

for (i = 0; i < menu->n\_items; ++i) {

menu\_item = menu->items + i;

Lcd\_displayString(Lcd\_Lines[i + 1],

menu\_item->title,

LCD\_\_STYLE\_NO\_OVERRIDE);

if (i == menu->item\_index) {

Menu\_highlightLine(Lcd\_Lines[i + 1]);

}

}

}

void Menu\_selectUpperItem(Menu\_Menu \*menu)

{

if (!Menu\_isValidMenu(menu)) return;

/\* Display current menu item as non-selected \*/

Menu\_highlightLine(Lcd\_Lines[menu->item\_index + 1]);

/\* Determine new menu item (iteratively) \*/

if (menu->item\_index > 0) {

--(menu->item\_index);

} else {

menu->item\_index = menu->n\_items - 1;

}

/\* Display new menu item as selected \*/

Menu\_highlightLine(Lcd\_Lines[menu->item\_index + 1]);

}

void Menu\_selectLowerItem(Menu\_Menu \*menu)

{

if (!Menu\_isValidMenu(menu)) return;

/\* Display current menu item as non-selected \*/

Menu\_highlightLine(Lcd\_Lines[menu->item\_index + 1]);

/\* Determine new menu item (iteratively) \*/

if (menu->item\_index >= (menu->n\_items - 1)) {

menu->item\_index = 0;

} else {

++(menu->item\_index);

}

/\* Display new menu item as selected \*/

Menu\_highlightLine(Lcd\_Lines[menu->item\_index + 1]);

}

module\_demo.h:

#ifndef MODULE\_DEMOINIT

#define MODULE\_DEMOINIT

void Demo\_init(void);

#endif // MODULE\_DEMOINIT

module\_demo.c:

#include "module\_demo.h"

#include "module\_pwr.h"

#include "lcd.h"

#include "joystick.h"

#include "MDR32FxQI\_rst\_clk.h" // Milandr::Drivers:RST\_CLK

#include "MDR32FxQI\_port.h" // Milandr::Drivers:PORT

#include "MDR32FxQI\_bkp.h" // Milandr::Drivers:BKP

#define ALL\_PORTS\_CLK (RST\_CLK\_PCLK\_PORTA | RST\_CLK\_PCLK\_PORTB | \

RST\_CLK\_PCLK\_PORTC | RST\_CLK\_PCLK\_PORTD | \

RST\_CLK\_PCLK\_PORTE | RST\_CLK\_PCLK\_PORTF)

static void Demo\_configureClock(void);

static void Demo\_configureRtc(void);

void Demo\_init(void)

{

PORT\_InitTypeDef PortInitStructure;

PORT\_StructInit(&PortInitStructure);

Demo\_configureClock();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LCD Initialization \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Configure LCD\_DATA\_PORT for data transfer to/from LCD \*/

PortInitStructure.PORT\_Pin = LCD\_DATA\_BAS\_8\_0;

PortInitStructure.PORT\_FUNC = PORT\_FUNC\_PORT;

PortInitStructure.PORT\_OE = PORT\_OE\_IN;

PortInitStructure.PORT\_SPEED = PORT\_SPEED\_SLOW;

PortInitStructure.PORT\_MODE = PORT\_MODE\_DIGITAL;

PORT\_Init(LCD\_DATA\_PORT\_0, &PortInitStructure);

PortInitStructure.PORT\_Pin = LCD\_DATA\_BAS\_8\_1;

PORT\_Init(LCD\_DATA\_PORT\_1, &PortInitStructure);

/\* Configure LCD\_RD\_WR\_PORT for read/write control \*/

/\* Switch LCD into data output mode \*/

PORT\_SetBits(LCD\_RD\_WR\_PORT, LCD\_RD\_WR\_PIN);

PortInitStructure.PORT\_Pin = LCD\_RD\_WR\_PIN;

PortInitStructure.PORT\_OE = PORT\_OE\_OUT;

PORT\_Init(LCD\_RD\_WR\_PORT, &PortInitStructure);

/\* Configure LCD\_CLOCK\_PORT for CLOCK signal control\*/

/\* Set LCD CLOCK signal to its initial value (0) \*/

PORT\_ResetBits(LCD\_CLOCK\_PORT, LCD\_CLOCK\_PIN);

PortInitStructure.PORT\_Pin = LCD\_CLOCK\_PIN;

PortInitStructure.PORT\_SPEED = PORT\_SPEED\_FAST;

PORT\_Init(LCD\_CLOCK\_PORT, &PortInitStructure);

/\* Configure LCD\_CRYSTAL\_PORT for LCD crystal control \*/

/\* De-select both LCD crystals\*/

PORT\_ResetBits(LCD\_CRYSTAL\_PORT, LCD\_CRYSTAL\_PINs);

PortInitStructure.PORT\_Pin = LCD\_CRYSTAL\_PINs;

PortInitStructure.PORT\_SPEED = PORT\_SPEED\_SLOW;

PORT\_Init(LCD\_CRYSTAL\_PORT, &PortInitStructure);

/\* Configure LCD\_CMD\_DATA\_PORT for data/command mode switching \*/

PortInitStructure.PORT\_Pin = LCD\_CMD\_DATA\_PIN;

PORT\_Init(LCD\_CMD\_DATA\_PORT, &PortInitStructure);

/\* Configure LCD\_RESET\_PORT for RESET signal control \*/

/\* Zeroing LCD RES signal (initial state) \*/

PORT\_ResetBits(LCD\_RESET\_PORT, LCD\_RESET\_PIN);

PortInitStructure.PORT\_Pin = LCD\_RESET\_PIN;

PortInitStructure.PORT\_SPEED = PORT\_SPEED\_FAST;

PORT\_Init(LCD\_RESET\_PORT, &PortInitStructure);

LCD\_INIT();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Joystick Initialization \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Configure SEL\_PORT for input to handle joystick event SEL \*/

PortInitStructure.PORT\_Pin = SEL\_PIN;

PortInitStructure.PORT\_OE = PORT\_OE\_IN;

PortInitStructure.PORT\_FUNC = PORT\_FUNC\_PORT;

PortInitStructure.PORT\_SPEED = PORT\_SPEED\_SLOW;

PORT\_Init(SEL\_PORT, &PortInitStructure);

/\* Configure UP\_PORT for input to handle joystick event UP \*/

PortInitStructure.PORT\_Pin = UP\_PIN;

PORT\_Init(UP\_PORT, &PortInitStructure);

/\* Configure DOWN\_PORT for input to handle joystick event DOWN \*/

PortInitStructure.PORT\_Pin = DOWN\_PIN;

PORT\_Init(DOWN\_PORT, &PortInitStructure);

/\* Configure LEFT\_PORT for input to handle joystick event LEFT \*/

PortInitStructure.PORT\_Pin = LEFT\_PIN;

PORT\_Init(LEFT\_PORT, &PortInitStructure);

/\* Configure RIGHT\_PORT for input to handle joystick event RIGHT \*/

PortInitStructure.PORT\_Pin = RIGHT\_PIN;

PORT\_Init(RIGHT\_PORT, &PortInitStructure);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PWR Initialization \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

PORT\_StructInit(&PortInitStructure);

PortInitStructure.PORT\_FUNC = PORT\_FUNC\_PORT;

PortInitStructure.PORT\_OE = PORT\_OE\_OUT;

PortInitStructure.PORT\_SPEED = PORT\_SPEED\_SLOW;

PortInitStructure.PORT\_MODE = PORT\_MODE\_DIGITAL;

PortInitStructure.PORT\_Pin = PWR\_\_DEVICE\_1\_PIN;

PORT\_Init(PWR\_\_DEVICE\_1\_PORT, &PortInitStructure);

PortInitStructure.PORT\_Pin = PWR\_\_DEVICE\_2\_PIN;

PORT\_Init(PWR\_\_DEVICE\_2\_PORT, &PortInitStructure);

PortInitStructure.PORT\_Pin = PWR\_\_DEVICE\_3\_PIN;

PORT\_Init(PWR\_\_DEVICE\_3\_PORT, &PortInitStructure);

PortInitStructure.PORT\_Pin = PWR\_\_DEVICE\_4\_PIN;

PORT\_Init(PWR\_\_DEVICE\_4\_PORT, &PortInitStructure);

PortInitStructure.PORT\_Pin = PWR\_\_DEVICE\_5\_PIN;

PORT\_Init(PWR\_\_DEVICE\_5\_PORT, &PortInitStructure);

PortInitStructure.PORT\_Pin = PWR\_\_DEVICE\_6\_PIN;

PORT\_Init(PWR\_\_DEVICE\_6\_PORT, &PortInitStructure);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Time Initialization \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

PORT\_StructInit(&PortInitStructure);

PortInitStructure.PORT\_Pin = PORT\_Pin\_6 | PORT\_Pin\_7;

PortInitStructure.PORT\_MODE = PORT\_MODE\_ANALOG;

PORT\_Init(MDR\_PORTE, &PortInitStructure);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Uart1 Initialization \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Enable the RTCHSE clock on UART1 \*/

RST\_CLK\_PCLKcmd(RST\_CLK\_PCLK\_UART1, ENABLE);

PORT\_StructInit(&PortInitStructure);

PortInitStructure.PORT\_SPEED = PORT\_SPEED\_MAXFAST;

PortInitStructure.PORT\_MODE = PORT\_MODE\_DIGITAL;

PortInitStructure.PORT\_FUNC = PORT\_FUNC\_OVERRID;

PortInitStructure.PORT\_Pin = PORT\_Pin\_6;

PortInitStructure.PORT\_OE = PORT\_OE\_IN;

PORT\_Init(MDR\_PORTA, &PortInitStructure);

PortInitStructure.PORT\_Pin = PORT\_Pin\_7;

PortInitStructure.PORT\_OE = PORT\_OE\_OUT;

PORT\_Init(MDR\_PORTA, &PortInitStructure);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Demo\_configureRtc();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

RST\_CLK\_PCLKcmd(RST\_CLK\_PCLK\_EEPROM, ENABLE);

}

static void Demo\_configureClock(void)

{

/\* Configure CPU\_PLL clock \*/

RST\_CLK\_CPU\_PLLconfig(RST\_CLK\_CPU\_PLLsrcHSIdiv1, 0);

/\* Enables the RTCHSE clock on all ports \*/

RST\_CLK\_PCLKcmd(ALL\_PORTS\_CLK, ENABLE);

}

static void Demo\_configureRtc(void)

{

/\* Enables the HSE clock for BKP control \*/

RST\_CLK\_PCLKcmd(RST\_CLK\_PCLK\_BKP, ENABLE);

/\* Configure LSE as RTC clock source \*/

RST\_CLK\_LSEconfig(RST\_CLK\_LSE\_ON);

/\* Wait till LSE is ready \*/

while (RST\_CLK\_LSEstatus() != SUCCESS) {

}

/\* Select the RTC Clock Source \*/

BKP\_RTCclkSource(BKP\_RTC\_LSEclk);

/\* Wait until last write operation on RTC registers has finished \*/

BKP\_RTC\_WaitForUpdate();

/\* Sets the RTC prescaler \*/

BKP\_RTC\_SetPrescaler(RTC\_PRESCALER\_VALUE);

/\* Wait until last write operation on RTC registers has finished \*/

BKP\_RTC\_WaitForUpdate();

/\* Sets the RTC calibrator \*/

BKP\_RTC\_Calibration(RTC\_CalibratorValue);

/\* Wait until last write operation on RTC registers has finished \*/

BKP\_RTC\_WaitForUpdate();

/\* Enable the RTC Clock \*/

BKP\_RTC\_WorkPermit(ENABLE);

/\* Enable the Second interrupt \*/

BKP\_RTC\_ITConfig(BKP\_RTC\_IT\_SECF, ENABLE);

NVIC\_EnableIRQ(BACKUP\_IRQn);

}

module\_pwr.h:

#ifndef MODULE\_PWR

#define MODULE\_PWR

#include "module\_menu.h"

#include "module\_time.h"

#include "MDR32FxQI\_port.h" // Milandr::Drivers:PORT

#define PWR\_\_N\_DEVICES 6

#define PWR\_\_DEVICE\_1\_PORT MDR\_PORTB

#define PWR\_\_DEVICE\_1\_PIN PORT\_Pin\_0

#define PWR\_\_DEVICE\_2\_PORT MDR\_PORTB

#define PWR\_\_DEVICE\_2\_PIN PORT\_Pin\_1

#define PWR\_\_DEVICE\_3\_PORT MDR\_PORTB

#define PWR\_\_DEVICE\_3\_PIN PORT\_Pin\_2

#define PWR\_\_DEVICE\_4\_PORT MDR\_PORTB

#define PWR\_\_DEVICE\_4\_PIN PORT\_Pin\_3

#define PWR\_\_DEVICE\_5\_PORT MDR\_PORTB

#define PWR\_\_DEVICE\_5\_PIN PORT\_Pin\_4

#define PWR\_\_DEVICE\_6\_PORT MDR\_PORTD

#define PWR\_\_DEVICE\_6\_PIN PORT\_Pin\_5

extern Utility\_Procedure \*Pwr\_StdUpProc;

extern Utility\_Procedure \*Pwr\_StdSelectProc;

extern Utility\_Procedure \*Pwr\_StdDownProc;

extern Menu\_Menu Pwr\_DeviceMenu;

void Pwr\_init(Utility\_Procedure \*return\_proc, Utility\_Procedure \*update);

int Pwr\_currentDevice(void);

Time\_TimeEdit Pwr\_currentDeviceTimeEdit(void);

void Pwr\_setCurrentDevice(int device);

void Pwr\_turnDeviceOn(void);

void Pwr\_turnDeviceOff(void);

uint8\_t \*Pwr\_getStats(void);

void Pwr\_checkPoint(uint32\_t time);

void Pwr\_updateStats(uint32\_t time);

void Pwr\_resetStats(uint32\_t time);

#endif // MODULE\_PWR

module\_pwr.c:

#include "module\_pwr.h"

#include "module\_pwrsts.h"

#include "module\_pwrmng.h"

#include "module\_time.h"

#include "module\_utility.h"

// Pwr

// Pwr typedefs

typedef enum {

Pwr\_DeviceState\_off = 0,

Pwr\_DeviceState\_on = !Pwr\_DeviceState\_off

} Pwr\_DeviceState;

typedef struct

{

Pwr\_DeviceState state;

unsigned int last\_update;

unsigned int worktime;

unsigned int previous\_worktime;

MDR\_PORT\_TypeDef \*port;

uint32\_t pin;

} Pwr\_Device;

// Pwr private function prototypes

static void Pwr\_updateGuiStr(void);

static void Pwr\_stdUpProc(void);

static void Pwr\_stdSelectProc(void);

static void Pwr\_stdDownProc(void);

static void Pwr\_setDevice1Proc(void);

static void Pwr\_setDevice2Proc(void);

static void Pwr\_setDevice3Proc(void);

static void Pwr\_setDevice4Proc(void);

static void Pwr\_setDevice5Proc(void);

static void Pwr\_setDevice6Proc(void);

static void Pwr\_updateGuiProc(void);

static void Pwr\_update(void);

static void Pwr\_turnDevice(Pwr\_DeviceState state);

static void Pwr\_toggleProc(void);

// Variables

static Pwr\_Device Pwr\_Devices[PWR\_\_N\_DEVICES] = {{Pwr\_DeviceState\_off, 0, 0, 0, PWR\_\_DEVICE\_1\_PORT, PWR\_\_DEVICE\_1\_PIN},

{Pwr\_DeviceState\_off, 0, 0, 0, PWR\_\_DEVICE\_2\_PORT, PWR\_\_DEVICE\_2\_PIN},

{Pwr\_DeviceState\_off, 0, 0, 0, PWR\_\_DEVICE\_3\_PORT, PWR\_\_DEVICE\_3\_PIN},

{Pwr\_DeviceState\_off, 0, 0, 0, PWR\_\_DEVICE\_4\_PORT, PWR\_\_DEVICE\_4\_PIN},

{Pwr\_DeviceState\_off, 0, 0, 0, PWR\_\_DEVICE\_5\_PORT, PWR\_\_DEVICE\_5\_PIN},

{Pwr\_DeviceState\_off, 0, 0, 0, PWR\_\_DEVICE\_6\_PORT, PWR\_\_DEVICE\_6\_PIN}};

static size\_t Pwr\_DeviceIndex = 0;

static uint8\_t Pwr\_DeviceStr[] = {'D', 'e', 'v', 'i', 'c', 'e', ' ', '0', '\0'};

static size\_t Pwr\_DeviceStrLength = UTILITY\_\_COUNT\_OF(Pwr\_DeviceStr);

static uint8\_t Pwr\_StateStr[] = {'S', 't', 'a', 't', 'e', ' ', 'o', 'f', 'f', '\0'};

static size\_t Pwr\_StateStrLength = UTILITY\_\_COUNT\_OF(Pwr\_StateStr);

static uint8\_t Pwr\_WorktimeStr[] = {'W', 'o', 'r', 'k', 'T', ':', ' ', '.', '.', 'h', '.', '.', 'm', '.', '.', 's', ' ', '\0'};

// h: 7, 8

// m: 10, 11

// s: 13, 14

// static size\_t Pwrsts\_WorktimeStrLength = UTILITY\_\_COUNT\_OF(Pwrsts\_WorktimeStr);

static uint8\_t Pwr\_PreviousWorktimeStr[] = {'P', 'W', 'o', 'r', 'k', 'T', ':', ' ', '.', '.', 'h', '.', '.', 'm', '.', '.', 's', '\0'};

// h: 8, 9

// m: 11, 12

// s: 14, 15

// static size\_t Pwrsts\_PreviousWorktimeStrLength = UTILITY\_\_COUNT\_OF(Pwrsts\_PreviousWorktimeStr);

Utility\_Procedure \*Pwr\_StdUpProc = NULL;

Utility\_Procedure \*Pwr\_StdSelectProc = NULL;

Utility\_Procedure \*Pwr\_StdDownProc = NULL;

static Menu\_MenuItem Pwr\_MainDeviceMenuItems[] = {{Pwr\_DeviceStr, NULL, NULL},

{Pwr\_StateStr, NULL, Pwr\_toggleProc},

{Pwr\_WorktimeStr, NULL, NULL},

{Pwr\_PreviousWorktimeStr, NULL, NULL},

{"Return", NULL, NULL}};

static Menu\_Menu Pwr\_MainDeviceMenu = {"Device info",

Pwr\_MainDeviceMenuItems,

UTILITY\_\_COUNT\_OF(Pwr\_MainDeviceMenuItems),

0,

Pwr\_stdUpProc,

Pwr\_stdSelectProc,

Pwr\_stdDownProc,

Pwr\_updateGuiProc};

static Menu\_MenuItem Pwr\_Device13MenuItems[] = {{"Device 1", &Pwr\_MainDeviceMenu, Pwr\_setDevice1Proc},

{"Device 2", &Pwr\_MainDeviceMenu, Pwr\_setDevice2Proc},

{"Device 3", &Pwr\_MainDeviceMenu, Pwr\_setDevice3Proc},

{"Return", NULL, NULL}};

static Menu\_Menu Pwr\_Device13Menu = {"Devices 1-3",

Pwr\_Device13MenuItems,

UTILITY\_\_COUNT\_OF(Pwr\_Device13MenuItems),

0,

Pwr\_stdUpProc,

Pwr\_stdSelectProc,

Pwr\_stdDownProc,

NULL};

static Menu\_MenuItem Pwr\_Device46MenuItems[] = {{"Device 4", &Pwr\_MainDeviceMenu, Pwr\_setDevice4Proc},

{"Device 5", &Pwr\_MainDeviceMenu, Pwr\_setDevice5Proc},

{"Device 6", &Pwr\_MainDeviceMenu, Pwr\_setDevice6Proc},

{"Return", NULL, NULL}};

static Menu\_Menu Pwr\_Device46Menu = {"Devices 4-6",

Pwr\_Device46MenuItems,

UTILITY\_\_COUNT\_OF(Pwr\_Device46MenuItems),

0,

Pwr\_stdUpProc,

Pwr\_stdSelectProc,

Pwr\_stdDownProc,

NULL};

static Menu\_MenuItem Pwr\_DeviceMenuItems[] = {{"Devices 1-3", &Pwr\_Device13Menu, NULL},

{"Devices 4-6", &Pwr\_Device46Menu, NULL},

{"Return", NULL, NULL}};

Menu\_Menu Pwr\_DeviceMenu = {"Devices",

Pwr\_DeviceMenuItems,

UTILITY\_\_COUNT\_OF(Pwr\_DeviceMenuItems),

0,

Pwr\_stdUpProc,

Pwr\_stdSelectProc,

Pwr\_stdDownProc,

NULL};

static Utility\_Procedure \*Pwr\_Update = NULL;

// Pwr public functions

void Pwr\_init(Utility\_Procedure \*return\_proc, Utility\_Procedure \*update)

{

size\_t i;

for (i = 0; i < PWR\_\_N\_DEVICES; ++i) {

Pwr\_Device \*device = Pwr\_Devices + i;

PORT\_ResetBits(device->port, device->pin);

}

Pwr\_MainDeviceMenuItems[UTILITY\_\_COUNT\_OF(Pwr\_MainDeviceMenuItems) - 1].proc = return\_proc;

Pwr\_Device13MenuItems[UTILITY\_\_COUNT\_OF(Pwr\_Device13MenuItems) - 1].proc = return\_proc;

Pwr\_Device46MenuItems[UTILITY\_\_COUNT\_OF(Pwr\_Device46MenuItems) - 1].proc = return\_proc;

Pwr\_DeviceMenuItems[UTILITY\_\_COUNT\_OF(Pwr\_DeviceMenuItems) - 1].proc = return\_proc;

Pwr\_Update = update;

}

int Pwr\_currentDevice(void)

{

return Pwr\_DeviceIndex;

}

Time\_TimeEdit Pwr\_currentDeviceTimeEdit(void)

{

return Time\_timeEdit((Pwr\_Devices + Pwr\_DeviceIndex)->worktime);

}

void Pwr\_setCurrentDevice(int device)

{

if (device < 0) {

Pwr\_DeviceIndex = 0;

} else if (PWR\_\_N\_DEVICES <= device) {

Pwr\_DeviceIndex = PWR\_\_N\_DEVICES - 1;

} else {

Pwr\_DeviceIndex = device;

}

}

void Pwr\_turnDeviceOn(void)

{

Pwr\_turnDevice(Pwr\_DeviceState\_on);

Pwr\_update();

}

void Pwr\_turnDeviceOff(void)

{

Pwr\_update();

Pwr\_turnDevice(Pwr\_DeviceState\_off);

}

uint8\_t \*Pwr\_getStats(void)

{

static uint8\_t stats[] = {'D', '0', '-', 'f',

'\r', '\n',

'w', '0', '0', ':', '0', '0', ':', '0', '0',

'\r', '\n',

'p', '0', '0', ':', '0', '0', ':', '0', '0',

'\r', '\n',

'D', '1', '-', 'f',

'\r', '\n',

'w', '0', '0', ':', '0', '0', ':', '0', '0',

'\r', '\n',

'p', '0', '0', ':', '0', '0', ':', '0', '0',

'\r', '\n',

'D', '2', '-', 'f',

'\r', '\n',

'w', '0', '0', ':', '0', '0', ':', '0', '0',

'\r', '\n',

'p', '0', '0', ':', '0', '0', ':', '0', '0',

'\r', '\n',

'D', '3', '-', 'f',

'\r', '\n',

'w', '0', '0', ':', '0', '0', ':', '0', '0',

'\r', '\n',

'p', '0', '0', ':', '0', '0', ':', '0', '0',

'\r', '\n',

'D', '4', '-', 'f',

'\r', '\n',

'w', '0', '0', ':', '0', '0', ':', '0', '0',

'\r', '\n',

'p', '0', '0', ':', '0', '0', ':', '0', '0',

'\r', '\n',

'D', '5', '-', 'f',

'\r', '\n',

'w', '0', '0', ':', '0', '0', ':', '0', '0',

'\r', '\n',

'p', '0', '0', ':', '0', '0', ':', '0', '0',

'\r', '\n',

'D', '6', '-', 'f',

'\r', '\n',

'w', '0', '0', ':', '0', '0', ':', '0', '0',

'\r', '\n',

'p', '0', '0', ':', '0', '0', ':', '0', '0',

'\r', '\n',

'D', '7', '-', 'f',

'\r', '\n',

'w', '0', '0', ':', '0', '0', ':', '0', '0',

'\r', '\n',

'p', '0', '0', ':', '0', '0', ':', '0', '0',

'\r', '\n',

'\0'};

unsigned int i, base\_i;

Time\_TimeEdit edit;

Pwr\_Device \*device;

for (i = 0; i < PWR\_\_N\_DEVICES; ++i) {

base\_i = i \* 28;

device = Pwr\_Devices + i;

stats[base\_i + 3] = (device->state) ? 'n' : 'f';

edit = Time\_timeEdit(device->worktime);

stats[base\_i + 7] = Utility\_intToChar(edit.hour / 10);

stats[base\_i + 8] = Utility\_intToChar(edit.hour % 10);

stats[base\_i + 10] = Utility\_intToChar(edit.minute / 10);

stats[base\_i + 11] = Utility\_intToChar(edit.minute % 10);

stats[base\_i + 13] = Utility\_intToChar(edit.second / 10);

stats[base\_i + 14] = Utility\_intToChar(edit.second % 10);

edit = Time\_timeEdit(device->previous\_worktime);

stats[base\_i + 18] = Utility\_intToChar(edit.hour / 10);

stats[base\_i + 19] = Utility\_intToChar(edit.hour % 10);

stats[base\_i + 21] = Utility\_intToChar(edit.minute / 10);

stats[base\_i + 22] = Utility\_intToChar(edit.minute % 10);

stats[base\_i + 24] = Utility\_intToChar(edit.second / 10);

stats[base\_i + 25] = Utility\_intToChar(edit.second % 10);

}

return stats;

}

void Pwr\_checkPoint(uint32\_t time)

{

unsigned int i;

for (i = 0; i < PWR\_\_N\_DEVICES; ++i) {

Pwr\_Device \*device = Pwr\_Devices + i;

device->previous\_worktime = device->worktime;

device->worktime = 0;

device->last\_update = time;

}

}

void Pwr\_updateStats(uint32\_t time)

{

unsigned int i;

for (i = 0; i < PWR\_\_N\_DEVICES; ++i) {

Pwr\_Device \*device = Pwr\_Devices + i;

if (device->state) {

device->worktime += time - device->last\_update;

}

device->last\_update = time;

}

}

void Pwr\_resetStats(uint32\_t time)

{

unsigned int i;

for (i = 0; i < PWR\_\_N\_DEVICES; ++i) {

Pwr\_Device \*device = Pwr\_Devices + i;

device->last\_update = time;

}

}

// Pwr private functions

void Pwr\_updateGuiStr(void)

{

unsigned int worktime = Pwr\_Devices[Pwr\_DeviceIndex].worktime;

unsigned int previous\_worktime = Pwr\_Devices[Pwr\_DeviceIndex].previous\_worktime;

Time\_TimeEdit edit;

Pwr\_DeviceStr[Pwr\_DeviceStrLength - 2] = Utility\_intToChar(Pwr\_DeviceIndex + 1);

Pwr\_StateStr[Pwr\_StateStrLength - 3] = Pwr\_Devices[Pwr\_DeviceIndex].state ? 'n' : 'f';

Pwr\_StateStr[Pwr\_StateStrLength - 2] = Pwr\_Devices[Pwr\_DeviceIndex].state ? ' ' : 'f';

edit = Time\_timeEdit(worktime);

// hours

Pwr\_WorktimeStr[7] = Utility\_intToChar(edit.hour / 10);

Pwr\_WorktimeStr[8] = Utility\_intToChar(edit.hour % 10);

// minutes

Pwr\_WorktimeStr[10] = Utility\_intToChar(edit.minute / 10);

Pwr\_WorktimeStr[11] = Utility\_intToChar(edit.minute % 10);

// seconds

Pwr\_WorktimeStr[13] = Utility\_intToChar(edit.second / 10);

Pwr\_WorktimeStr[14] = Utility\_intToChar(edit.second % 10);

edit = Time\_timeEdit(previous\_worktime);

// hours

Pwr\_PreviousWorktimeStr[8] = Utility\_intToChar(edit.hour / 10);

Pwr\_PreviousWorktimeStr[9] = Utility\_intToChar(edit.hour % 10);

// minutes

Pwr\_PreviousWorktimeStr[11] = Utility\_intToChar(edit.minute / 10);

Pwr\_PreviousWorktimeStr[12] = Utility\_intToChar(edit.minute % 10);

// seconds

Pwr\_PreviousWorktimeStr[14] = Utility\_intToChar(edit.second / 10);

Pwr\_PreviousWorktimeStr[15] = Utility\_intToChar(edit.second % 10);

}

void Pwr\_stdUpProc(void)

{

if (Pwr\_StdUpProc != NULL) {

Pwr\_StdUpProc();

}

}

void Pwr\_stdSelectProc(void)

{

if (Pwr\_StdSelectProc != NULL) {

Pwr\_StdSelectProc();

}

}

void Pwr\_stdDownProc(void)

{

if (Pwr\_StdDownProc != NULL) {

Pwr\_StdDownProc();

}

}

void Pwr\_setDevice1Proc(void)

{

Pwr\_setCurrentDevice(0);

Pwr\_updateGuiStr();

}

void Pwr\_setDevice2Proc(void)

{

Pwr\_setCurrentDevice(1);

Pwr\_updateGuiStr();

}

void Pwr\_setDevice3Proc(void)

{

Pwr\_setCurrentDevice(2);

Pwr\_updateGuiStr();

}

void Pwr\_setDevice4Proc(void)

{

Pwr\_setCurrentDevice(3);

Pwr\_updateGuiStr();

}

void Pwr\_setDevice5Proc(void)

{

Pwr\_setCurrentDevice(4);

Pwr\_updateGuiStr();

}

void Pwr\_setDevice6Proc(void)

{

Pwr\_setCurrentDevice(5);

Pwr\_updateGuiStr();

}

void Pwr\_updateGuiProc(void)

{

size\_t index = Pwr\_MainDeviceMenu.item\_index;

Pwr\_updateGuiStr();

Lcd\_displayStringSh(6, Lcd\_Line\_line2, Pwr\_StateStr, (index == 1 ? LCD\_\_STYLE\_HIGHLIGHT : LCD\_\_STYLE\_NO\_STYLE));

Lcd\_displayStringSh(7, Lcd\_Line\_line3, Pwr\_WorktimeStr, (index == 2 ? LCD\_\_STYLE\_HIGHLIGHT : LCD\_\_STYLE\_NO\_STYLE));

Lcd\_displayStringSh(8, Lcd\_Line\_line4, Pwr\_PreviousWorktimeStr, (index == 3 ? LCD\_\_STYLE\_HIGHLIGHT : LCD\_\_STYLE\_NO\_STYLE));

}

void Pwr\_update(void)

{

if (Pwr\_Update != NULL) {

Pwr\_Update();

}

}

void Pwr\_turnDevice(Pwr\_DeviceState state)

{

Pwr\_Device \*device = Pwr\_Devices + Pwr\_DeviceIndex;

if (state) {

PORT\_SetBits(device->port, device->pin);

} else {

PORT\_ResetBits(device->port, device->pin);

}

device->state = state;

}

void Pwr\_toggleProc(void)

{

if (Pwr\_Devices[Pwr\_DeviceIndex].state) {

Pwr\_turnDeviceOff();

} else {

Pwr\_turnDeviceOn();

}

Pwr\_updateGuiStr();

Lcd\_displayStringSh(6, Lcd\_Line\_line2, Pwr\_StateStr, LCD\_\_STYLE\_HIGHLIGHT);

}

module\_time.h:

#ifndef MODULE\_TIME

#define MODULE\_TIME

#include "module\_menu.h"

typedef struct

{

unsigned int hour;

unsigned int minute;

unsigned int second;

} Time\_TimeEdit;

extern Menu\_Menu Time\_SetTimeMenu;

Time\_TimeEdit Time\_timeEdit(uint32\_t raw\_time);

void Time\_init(Utility\_Procedure \*return\_proc, Utility\_ProcedureUint32 \*time\_set\_callback);

uint32\_t Time\_getRawTime(void);

Time\_TimeEdit Time\_getTime(void);

void Time\_setRawTime(uint32\_t time);

void Time\_setTime(Time\_TimeEdit time);

uint32\_t Time\_getNPassedDays(void);

void Time\_addPassedDay(void);

#endif // MODULE\_TIME

module\_time.c:

#include "module\_time.h"

#include "MDR32FxQI\_bkp.h" // Milandr::Drivers:BKP

#include "MDR32FxQI\_rst\_clk.h" // Milandr::Drivers:RST\_CLK

// Private functions prototypes

static void Time\_upProc(void);

static void Time\_selectProc(void);

static void Time\_downProc(void);

static void Time\_updateProc(void);

static void Time\_highlightItem(void);

static void Time\_changeTimeProc(void);

static void Time\_saveProc(void);

static void Time\_returnProc(void);

static void Time\_increaseProc(void);

static void Time\_reduceProc(void);

static void Time\_updateEdit(void);

static void Time\_updateGuiStr(void);

// Variables

static Time\_TimeEdit Time\_Time = {0, 0, 0};

static uint8\_t Time\_HourStr[] = {'0', '0', 'h', '\0'};

// static size\_t Time\_HourStrLength = UTILITY\_\_COUNT\_OF(Time\_HourStr);

static uint8\_t Time\_MinuteStr[] = {'0', '0', 'm', '\0'};

// static size\_t Time\_MinuteStrLength = UTILITY\_\_COUNT\_OF(Time\_MinuteStr);

static uint8\_t Time\_SecondStr[] = {'0', '0', 's', '\0'};

// static size\_t Time\_SecondStrStrLength = UTILITY\_\_COUNT\_OF(Time\_SecondStr);

static uint8\_t Time\_TimeStr[] = {'.', '.', 'h', '.', '.', 'm', '.', '.', 's', '\0'};

// static size\_t Time\_TimeStrLength = UTILITY\_\_COUNT\_OF(Time\_TimeStr);

static uint32\_t Time\_NPassedDays = 0;

#define TIME\_\_RETURN\_ITEM\_INDEX 4

static Menu\_MenuItem Time\_SetTimeMenuItems[] = {{Time\_HourStr, NULL, Time\_changeTimeProc}, {Time\_MinuteStr, NULL, Time\_changeTimeProc}, {Time\_SecondStr, NULL, Time\_changeTimeProc}, {"Save", NULL, Time\_saveProc}, {Time\_TimeStr, NULL, NULL}};

Menu\_Menu Time\_SetTimeMenu = {"Set time", Time\_SetTimeMenuItems, UTILITY\_\_COUNT\_OF(Time\_SetTimeMenuItems), 0, Time\_upProc, Time\_selectProc, Time\_downProc, Time\_updateProc};

static Utility\_ProcedureUint32 \*Time\_TimeSetCallback = NULL;

// Public functions

Time\_TimeEdit Time\_timeEdit(uint32\_t raw\_time)

{

Time\_TimeEdit time;

time.hour = raw\_time / 3600;

time.minute = raw\_time / 60 % 60;

time.second = raw\_time % 60;

return time;

}

void Time\_init(Utility\_Procedure \*return\_proc, Utility\_ProcedureUint32 \*time\_set\_callback)

{

unsigned int i;

uint32\_t tmp;

Time\_SetTimeMenuItems[TIME\_\_RETURN\_ITEM\_INDEX].proc = return\_proc;

Time\_TimeSetCallback = time\_set\_callback;

// if backup was reset (first run)

if (MDR\_BKP->REG\_00 != 0x1234) {

Time\_setRawTime(0);

MDR\_BKP->REG\_00 = 0x1234;

/\* Wait until last write operation on RTC registers has finished \*/

BKP\_RTC\_WaitForUpdate();

} else if (tmp = Time\_getRawTime(), tmp / 86400 != 0) {

for (i = 0; i < tmp / 86400; ++i) {

Time\_addPassedDay();

}

Time\_setRawTime(tmp % 86400);

}

}

uint32\_t Time\_getRawTime(void)

{

return BKP\_RTC\_GetCounter();

}

Time\_TimeEdit Time\_getTime(void)

{

return Time\_timeEdit(Time\_getRawTime());

}

void Time\_setRawTime(uint32\_t time)

{

uint32\_t old\_time = Time\_getRawTime();

/\* Wait until last write operation on RTC registers has finished \*/

BKP\_RTC\_WaitForUpdate();

/\* Set the RTC counter value \*/

BKP\_RTC\_SetCounter(time);

/\* Wait until last write operation on RTC registers has finished \*/

BKP\_RTC\_WaitForUpdate();

if (Time\_TimeSetCallback != NULL) {

Time\_TimeSetCallback(old\_time);

}

}

void Time\_setTime(Time\_TimeEdit time)

{

if ((23 < time.hour) || (59 < time.minute) || (59 < time.second)) return;

Time\_setRawTime(time.hour \* 3600 + time.minute \* 60 + time.second);

Time\_updateGuiStr();

}

uint32\_t Time\_getNPassedDays(void)

{

uint32\_t tmp = Time\_NPassedDays;

Time\_NPassedDays = 0;

return tmp;

}

void Time\_addPassedDay(void)

{

++Time\_NPassedDays;

}

// Private functions

void Time\_upProc(void)

{

Menu\_selectUpperItem(&Time\_SetTimeMenu);

}

void Time\_selectProc(void)

{

size\_t index = Time\_SetTimeMenu.item\_index;

Utility\_Procedure \*proc = Time\_SetTimeMenu.items[index].proc;

if (proc != NULL) {

proc();

}

}

void Time\_downProc(void)

{

Menu\_selectLowerItem(&Time\_SetTimeMenu);

}

void Time\_updateProc(void)

{

uint8\_t style = LCD\_\_STYLE\_NO\_STYLE;

Time\_TimeEdit time = Time\_getTime();

Time\_TimeStr[0] = '0' + (time.hour / 10 % 10);

Time\_TimeStr[1] = '0' + (time.hour % 10);

Time\_TimeStr[3] = '0' + (time.minute / 10 % 10);

Time\_TimeStr[4] = '0' + (time.minute % 10);

Time\_TimeStr[6] = '0' + (time.second / 10 % 10);

Time\_TimeStr[7] = '0' + (time.second % 10);

if (Time\_SetTimeMenu.item\_index == TIME\_\_RETURN\_ITEM\_INDEX) {

style |= LCD\_\_STYLE\_HIGHLIGHT;

}

Lcd\_displayString(Lcd\_Line\_line5, Time\_TimeStr, style);

}

void Time\_highlightItem(void)

{

int8\_t style = LCD\_\_STYLE\_NO\_OVERRIDE | LCD\_\_STYLE\_HIGHLIGHT;

size\_t index = Time\_SetTimeMenu.item\_index;

switch (index) {

case 0:

case 1:

case 2:

Lcd\_displayString(Lcd\_Lines[index + 1], " ", style);

break;

}

}

void Time\_changeTimeProc(void)

{

Time\_SetTimeMenu.up = Time\_increaseProc;

Time\_SetTimeMenu.select = Time\_returnProc;

Time\_SetTimeMenu.down = Time\_reduceProc;

Time\_highlightItem();

}

void Time\_saveProc(void)

{

Time\_setTime(Time\_Time);

}

void Time\_returnProc(void)

{

Time\_SetTimeMenu.up = Time\_upProc;

Time\_SetTimeMenu.select = Time\_selectProc;

Time\_SetTimeMenu.down = Time\_downProc;

Time\_highlightItem();

}

void Time\_increaseProc(void)

{

switch (Time\_SetTimeMenu.item\_index) {

case 0:

Time\_Time.hour = (Time\_Time.hour + 1) % 24;

break;

case 1:

Time\_Time.minute = (Time\_Time.minute + 1) % 60;

break;

case 2:

Time\_Time.second = (Time\_Time.second + 1) % 60;

break;

default:

return;

}

Time\_updateEdit();

}

void Time\_reduceProc(void)

{

switch (Time\_SetTimeMenu.item\_index) {

case 0:

Time\_Time.hour = (Time\_Time.hour == 0) ? 23 : Time\_Time.hour - 1;

break;

case 1:

Time\_Time.minute = (Time\_Time.minute == 0) ? 59 : Time\_Time.minute - 1;

break;

case 2:

Time\_Time.second = (Time\_Time.second == 0) ? 59 : Time\_Time.second - 1;

break;

}

Time\_updateEdit();

}

void Time\_updateEdit(void)

{

Time\_updateGuiStr();

switch (Time\_SetTimeMenu.item\_index) {

case 0:

Lcd\_displayString(Lcd\_Line\_line1, Time\_HourStr, LCD\_\_STYLE\_NO\_STYLE);

break;

case 1:

Lcd\_displayString(Lcd\_Line\_line2, Time\_MinuteStr, LCD\_\_STYLE\_NO\_STYLE);

break;

case 2:

Lcd\_displayString(Lcd\_Line\_line3, Time\_SecondStr, LCD\_\_STYLE\_NO\_STYLE);

break;

}

}

void Time\_updateGuiStr(void)

{

Time\_HourStr[0] = '0' + (Time\_Time.hour / 10);

Time\_HourStr[1] = '0' + (Time\_Time.hour % 10);

Time\_MinuteStr[0] = '0' + (Time\_Time.minute / 10);

Time\_MinuteStr[1] = '0' + (Time\_Time.minute % 10);

Time\_SecondStr[0] = '0' + (Time\_Time.second / 10);

Time\_SecondStr[1] = '0' + (Time\_Time.second % 10);

}

module\_interrupt.h:

#ifndef MODULE\_INTERRUPT

#define MODULE\_INTERRUPT

#include "module\_utility.h"

#include "MDR32FxQI\_config.h" // Milandr::Device:Startup

#define UART\_LINE\_STATE\_EVENTS (UART\_IT\_OE | UART\_IT\_BE | UART\_IT\_PE | UART\_IT\_FE | \

UART\_IT\_DSR | UART\_IT\_DCD | UART\_IT\_RI)

extern Utility\_Procedure \*Interrupt\_UartReceiveHandler;

extern Utility\_Procedure \*Interrupt\_UartSendHandler;

extern Utility\_ProcedureUint32 \*Interrupt\_UartLineStateHandler;

extern Utility\_Procedure \*Interrupt\_RtcSecondHandler;

#endif // MODULE\_INTERRUPT

module\_interrupt.c:

#include "module\_interrupt.h"

#include "MDR32FxQI\_bkp.h" // Milandr::Drivers:BKP

#include "MDR32FxQI\_uart.h" // Milandr::Drivers:UART

#include <stddef.h>

// Private function prototypes

static void Interrupt\_UartXHandler(MDR\_UART\_TypeDef \*uart\_x);

// Variables

Utility\_Procedure \*Interrupt\_UartReceiveHandler = NULL;

Utility\_Procedure \*Interrupt\_UartSendHandler = NULL;

Utility\_ProcedureUint32 \*Interrupt\_UartLineStateHandler = NULL;

Utility\_Procedure \*Interrupt\_RtcSecondHandler = NULL;

// Public functions

void UART1\_IRQHandler(void)

{

Interrupt\_UartXHandler(MDR\_UART1);

}

void UART2\_IRQHandler(void)

{

Interrupt\_UartXHandler(MDR\_UART2);

}

void BACKUP\_IRQHandler(void)

{

if (BKP\_RTC\_GetFlagStatus(BKP\_RTC\_FLAG\_SECF) == SET) {

BKP\_RTC\_ITConfig(BKP\_RTC\_IT\_SECF, DISABLE);

if (Interrupt\_RtcSecondHandler != NULL) {

Interrupt\_RtcSecondHandler();

}

BKP\_RTC\_ITConfig(BKP\_RTC\_IT\_SECF, ENABLE);

}

}

// Private functions

void Interrupt\_UartXHandler(MDR\_UART\_TypeDef \*uart\_x)

{

uint32\_t tmp\_ris = uart\_x->RIS;

/\* Clear all pending bits except for UART\_IT\_RX \*/

uart\_x->ICR |= (tmp\_ris & (~UART\_IT\_RX));

/\* Receive handler \*/

if ((tmp\_ris & UART\_IT\_RX) && Interrupt\_UartReceiveHandler != NULL) {

Interrupt\_UartReceiveHandler();

}

/\* Transmit handler \*/

if ((tmp\_ris & UART\_IT\_TX) && Interrupt\_UartSendHandler != NULL) {

Interrupt\_UartSendHandler();

}

/\* Break/error handler \*/

if (Interrupt\_UartLineStateHandler != 0) {

tmp\_ris &= UART\_LINE\_STATE\_EVENTS;

if (tmp\_ris) {

Interrupt\_UartLineStateHandler(tmp\_ris);

}

}

}

module\_uart.h:

#ifndef MODULE\_UART

#define MODULE\_UART

#include "types.h"

#include <stddef.h>

#define UART\_\_RECIEVE\_ERROR\_NO\_ERROR 0

#define UART\_\_RECIEVE\_ERROR\_FRAME 1

#define UART\_\_RECIEVE\_ERROR\_PARITY 2

#define UART\_\_RECIEVE\_ERROR\_BREAK\_LINE 4

#define UART\_\_RECIEVE\_ERROR\_OVERFLOW 8

typedef struct

{

uint8\_t ch;

uint8\_t error;

} Uart\_ReceivedChar;

typedef struct

{

Uart\_ReceivedChar received\_data;

bool received;

} Uart\_MaybeReceivedChar;

void Uart\_init(void);

// buffer operations

Uart\_MaybeReceivedChar Uart\_getChar(void);

// str[str\_length].ch must contain '\0' character

// return number of received chars

int Uart\_getString(Uart\_ReceivedChar \*str);

// return TRUE if successfully

bool Uart\_putChar(uint8\_t ch);

// param 'str' must contain '\0' character

// return number of received chars

int Uart\_putString(uint8\_t \*str);

// if it is impossible to put whole string

// buffer will not be changed

bool Uart\_putWholeString(uint8\_t \*str);

// uart buffer operations

void Uart\_receiveData(void);

void Uart\_sendData(void);

#endif // MODULE\_UART

module\_uart.c:

module\_lcd.h:

#ifndef MODULE\_LCD

#define MODULE\_LCD

#include "text.h"

#define LCD\_\_STYLE\_NO\_STYLE 0

#define LCD\_\_STYLE\_HIGHLIGHT 1

#define LCD\_\_STYLE\_OVERRIDE\_LINE 2

#define LCD\_\_STYLE\_NO\_OVERRIDE 4

#define LCD\_\_N\_LINES 6

typedef enum {

Lcd\_Line\_line0 = 0,

Lcd\_Line\_line1 = LineMessage1,

Lcd\_Line\_line2 = LineMessage2,

Lcd\_Line\_line3 = LineMessage3,

Lcd\_Line\_line4 = LineMessage4,

Lcd\_Line\_line5 = LineMessage5,

} Lcd\_Line;

extern const Lcd\_Line \*Lcd\_Lines;

void Lcd\_displayStringX(uint32\_t x, Lcd\_Line line, ucint8\_t \*str, int style);

void Lcd\_displayStringSh(uint32\_t shift, Lcd\_Line line, ucint8\_t \*str, int style);

void Lcd\_displayString(Lcd\_Line line, ucint8\_t \*str, int style);

void Lcd\_displayLine(uint8\_t y);

void Lcd\_clear(void);

#endif // MODULE\_LCD

module\_lcd.c:

#include "module\_lcd.h"

#include "lcd.h"

#include "gl.h"

#include <string.h>

static const Lcd\_Line Lcd\_LineArray[] = {Lcd\_Line\_line0,

Lcd\_Line\_line1,

Lcd\_Line\_line2,

Lcd\_Line\_line3,

Lcd\_Line\_line4,

Lcd\_Line\_line5};

const Lcd\_Line \*Lcd\_Lines = Lcd\_LineArray;

void Lcd\_displayStringX(uint32\_t x, Lcd\_Line line, ucint8\_t \*str, int style)

{

uint32\_t i, tmp;

bool highlight = FALSE;

bool override\_line = FALSE;

bool no\_override = FALSE;

if (style & LCD\_\_STYLE\_HIGHLIGHT) {

highlight = TRUE;

}

if (style & LCD\_\_STYLE\_OVERRIDE\_LINE) {

override\_line = TRUE;

} else if (style & LCD\_\_STYLE\_NO\_OVERRIDE) {

no\_override = TRUE;

}

if (!no\_override) {

CurrentMethod = MET\_AND;

if (override\_line) {

LCD\_PUTS(0, (int8\_t)line, " ");

} else {

for (tmp = x, i = 0; str[i] != '\0'; tmp += CurrentFont->Width, ++i) {

LCD\_PUTC(tmp, (uint8\_t)line, ' ');

}

}

}

CurrentMethod = highlight ? MET\_NOT\_XOR : MET\_OR;

LCD\_PUTS(x, (uint8\_t)line, str);

}

void Lcd\_displayStringSh(uint32\_t shift, Lcd\_Line line, ucint8\_t \*str, int style)

{

uint32\_t x;

uint32\_t shift\_x;

uint32\_t str\_length = strlen((const char \*)str);

if (str\_length <= shift) {

return;

}

x = (MAX\_X - (CurrentFont->Width \* str\_length)) / 2;

shift\_x = CurrentFont->Width \* shift;

x += shift\_x;

Lcd\_displayStringX(x, line, str + shift, style);

}

void Lcd\_displayString(Lcd\_Line line, ucint8\_t \*str, int style)

{

uint32\_t x = (MAX\_X - (CurrentFont->Width \* strlen((const char \*)str))) / 2;

Lcd\_displayStringX(x, line, str, style);

}

void Lcd\_displayLine(uint8\_t y)

{

CurrentMethod = MET\_OR;

LCD\_Line(0, y, MAX\_X, y);

}

void Lcd\_clear(void)

{

LCD\_CLS();

}

module\_eeprom.h:

#ifndef MODULE\_EEPROM

#define MODULE\_EEPROM

#include "types.h"

#include "stddef.h"

void Eeprom\_init(void);

size\_t Eeprom\_nBytes(void);

uint8\_t Eeprom\_readByte(size\_t index);

bool Eeprom\_writeByte(uint8\_t byte, size\_t index);

#endif // MODULE\_EEPROM

module\_eeprom.c:

#include "module\_eeprom.h"

#include "module\_utility.h"

#include "MDR32FxQI\_eeprom.h" // Milandr::Drivers:EEPROM

#include "MDR32FxQI\_rst\_clk.h" // Milandr::Drivers:RST\_CLK

static uint32\_t Eeprom\_Storage[64] \_\_attribute\_\_((section("READONLY\_MEMORY\_SECTION"))) = {

'0', '1', '2', '3', '4', '5', '6', '7',

'0', '1', '2', '3', '4', '5', '6', '7',

'0', '1', '2', '3', '4', '5', '6', '7',

'0', '1', '2', '3', '4', '5', '6', '7',

'0', '1', '2', '3', '4', '5', '6', '7',

'0', '1', '2', '3', '4', '5', '6', '7',

'0', '1', '2', '3', '4', '5', '6', '7',

'0', '1', '2', '3', '4', '5', '6', '7',

};

void Eeprom\_init(void)

{

EEPROM\_SetLatency(EEPROM\_Latency\_1);

}

size\_t Eeprom\_nBytes(void)

{

return UTILITY\_\_COUNT\_OF(Eeprom\_Storage);

}

uint8\_t Eeprom\_readByte(size\_t index)

{

uint32\_t old\_mask;

uint8\_t byte;

if (Eeprom\_nBytes() <= index) {

return 0;

}

old\_mask = \_\_get\_PRIMASK();

\_\_set\_PRIMASK(1);

byte = EEPROM\_ReadWord((uint32\_t)(Eeprom\_Storage + index), EEPROM\_Main\_Bank\_Select);

\_\_set\_PRIMASK(old\_mask);

return byte;

}

bool Eeprom\_writeByte(uint8\_t byte, size\_t index)

{

uint32\_t old\_mask;

if (Eeprom\_nBytes() <= index) {

return FALSE;

}

old\_mask = \_\_get\_PRIMASK();

\_\_set\_PRIMASK(1);

EEPROM\_ProgramWord((uint32\_t)(Eeprom\_Storage + index), EEPROM\_Main\_Bank\_Select, (uint32\_t)byte);

\_\_set\_PRIMASK(old\_mask);

return TRUE;

}

module\_utility.h:

#ifndef MODULE\_UTILITY

#define MODULE\_UTILITY

#include "types.h"

#define UTILITY\_\_COUNT\_OF(a) (sizeof(a) / sizeof(\*(a)))

typedef void(Utility\_Procedure)(void);

typedef void(Utility\_ProcedureUint32)(uint32\_t);

char Utility\_intToChar(int digit);

char Utility\_uintToChar(unsigned int digit);

#endif // MODULE\_UTILITY

module\_utility.c:

#include "module\_utility.h"

char Utility\_intToChar(int digit)

{

if (digit < 0 || 9 < digit) {

return '?';

}

return '0' + digit;

}

char Utility\_uintToChar(unsigned int digit)

{

if (9 < digit) {

return '?';

}

return '0' + digit;

}

**Приложение Б**Графическая часть

На 2 листах

Электрическая схема функциональная

Электрическая схема принципиальная

**Приложение В**Перечень элементов

На 1 листе

**Приложение Г**Принципиальная схема отладочной платы

На 2 листах