ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

Согласовано

Профессор департамента
программной инженерии
факультета компьютерных наук
канд. техн. наук

_____ Гринкруг Е. М.

”____”_____ 2017 г

Утверждаю

Академический руководитель
образовательной программы
«Программная инженерия»
профессор департамента программной
инженерии канд. техн. наук

_____ Шилов В. В.

”____”_____ 2017 г

ПРОГРАММАТОР МИКРОКОНТРОЛЛЕРОВ PIC НА ОСНОВЕ ORANGE PI LITE

Пояснительная записка

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.509000 81 01-1

Студент группы БПИ 151 НИУ ВШЭ
_____ Абрамов А.М.
”____”_____ 2017 г

2017

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

УТВЕРЖДЕНО
RU.17701729.509000 81 01-1

ПРОГРАММАТОР МИКРОКОНТРОЛЛЕРОВ PIC НА ОСНОВЕ ORANGE PI LITE

Пояснительная записка

RU.17701729.509000 81 01-1

Листов 35

2017

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

Содержание

1 Введение **3**
1.1 Наименование 3
1.2 Краткая характеристика 3
1.3 Документы, на основании которых ведется разработка 3

2 Назначение разработки **4**
2.1 Функциональное назначение 4
2.2 Эксплутационное назначение 4

3 Технические характеристики **5**
3.1 Постановка задачи на разработку программы 5
3.2 Описание алгоритма и функционирования программы 5
3.2.1 Выбор алгоритма 5
3.2.2 Основные определения и структуры данных 6
3.2.3 Описание алгоритма 10
3.3 Метод организации входных и выходных данных 18
3.3.1 Описание метода входных и выходных данных 18
3.4 Выбор состава технических средств 19
3.4.1 Состав технических и програмных средств 19

4 Технико-экономические показатели **20**
4.1 Оринтировочная экономическая эффективность 20
4.2 Экономические преимущества разработки 20

5 Источники, используемые при разработке **21**
5.1 Список используемой литературы 21

6 Приложение 1. Терминология **22**
6.1 Терминология 22

7 Приложение 2. Формат INTEL HEX8M (.hex) **23**
7.1 Формат INTEL HEX8M (.hex) 23

8 Приложение 3. Описание классов **24**

8.1 Файл rpp.c 24
8.1.1 Функции 26
8.1.2 Переменные 32
8.2 Структура picmicro 33
8.2.1 Подробное описание 33
8.3 Структура picmemory 33
8.3.1 Подробное описание 34

1. Введение

1.1. Наименование

Наименование: «Программатор микроконтроллеров PIC на основе Orange PI Lite».
Наименование на английском: «Programmer for PIC Microcontrollers Based on Orange PI Lite».

1.2. Краткая характеристика

Цель работы - реализовать программатор для микроконтроллеров PIC серии 16F на тонком клиенте Orange PI Lite. В задачи работы входит расчет и инженерия электронной схемы для программирования, написание программы для управления этой схемой. Электронная схема предоставляет возможность подключить микроконтроллер PIC серии 16F к тонкому клиенту Orange Pi Lite, управлять уровнями вольтажа на 5В и на 3В и возможность проверить процесс программирования на светодиодах. Программа предоставляет пользователю командный и графичексий интерфейсы, чтение файлов INTEL HEX8M, возможность записать файлы программы в программную и EEPROM память микроконтроллера. В состав работы также входит создание демонстрационных исходных данных (файлов) для данного программатора и микроконтроллеров серии 16F.

Файл программы в формате INTEL HEX8M, удовлетворяющий требованиям входных данных, может быть получен в результате компиляции исходного кода одним из компиляторов для микроконтроллеров серии PIC 16F. Обычно для разработки используются пакеты предоставляющие интегрированную среду разработки. Например пакет MPLAB X (https://www.microchip.com/, разработчик: организация Microchip Ltd.)

1.3. Документы, на основании которых ведется разработка

Разработка программы ведется на основании приказа №6.18.1-02/1112-19 от 11.12.2016 «Об утверждении тем, руководителей курсовых работ студентов образовательной программы Программная инженерия факультета компьютерных наук» в соответствии с учебным планом подготовки бакалавров по направлению «Программная инженерия», факультета Компьютерных наук, Национального исследовательского университета «Высшая школа экономики»

2. Назначение разработки

2.1. Функциональное назначение

Функциональным назначением программы и электронной схемы является предоставление пользователю возможности загрузить программу из файла INTEL HEX8M (.hex), проинтерпретировать полученную информацию, проверить ее на наличие ошибок, стереть программную память и EEPROM память имикроконтроллера, записать прочитанные данные из файла в программную память микроконтроллера, записать новые данных в EEPROM память микроконтроллера без стирания программной памяти.

2.2. Эксплутационное назначение

Программа и электронная схема предназначена для работы на тонком кленте Orange Pi Lite с операционной системой семейства Linux. Программа и схема могут использоваться в учебных целях для демонстации основных компонентов необходимых для прошивки микроконтроллера. Они предоставляют новое направление использования тонкого клиента Orange Pi Lite. Ими может воспользоваться любой человек, желающий запрограммировать микроконтроллер, не имеющий на руках официального программатора, но у которого есть Orange Pi Lite. Данная программа и электронаня схема могут использоваться в качестве дешевой, простой и быстрой алтернативы к покупке официального программатора.

3. Технические характеристики

3.1. Постановка задачи на разработку программы

Цель работы - реализовать программатор для микроконтроллеров PIC серии 16F на тонком клиенте Orange PI Lite.

Задачи работы:

1. Чтение данных из формата INTEL HEX8M для хранения программы прошивки.
2. Возможность отдельной записи EEPROM памяти, не стирая програмную память микроконтроллера.
3. Поддержка 3 линеек микроконтроллеров серии 16F: 627A / 628A / 648A.
4. Проверка входного файла на корректность.
5. Графический интерфейс для оперирования программой.
6. Интерфейс командной строки для оперирования программой.
7. Повышаюший переходник с 3.3В на 5В для взаимодействия с микроконтроллером.
8. Схемотехника для платы которая позволяет подключить микроконтроллер к тонкому клиенту Orange Pi Lite.
9. Завершенные, работающие схемы на макетной плате.
10. Схемы разводки макетной платы для подключения микроконтроллера к Orange Pi Lite.

3.2. Описание алгоритма и функционирования программы

3.2.1. Выбор алгоритма

Различные подходы для программирования (или прошивки) микроконтроллеров варьируются в зависимости от кампании производиля. Данная курсовой работы нацелена на создание программатора для определенной серии и линейки микроконтроллеров определенного производителя. Микроконтроллер - микросхема, предназначенная для управления электронными устройствами. Микроконтроллер сочетает на одном кристалле функции микропроцессора, а также и функции периферийных устройств, содержит ОЗУ и ПЗУ. Это однокристальный компьютер, способный выполнять относительно простые задачи. Имеет смысл упомянуть две большие компании производящие микроконтроллеры общего назначения. А иммено кампанию Microchip, производящую микроконтроллеры PIC и компанию Atmel чьи микроконтроллеры ATmega легли в основу Arduino.

Программирование PIC16F627A/628A/648A производится с помощью серийного (последовательного) метода. Серийный режим позволяет PIC16F627A/628A/648A быть запрограммированым с изпользованием лишь 5 ножек микроконтроллера (или 6 ножек при режиме низковольтного программирования) уже будучи встроенным в систему пользователя. Это предоставляет большую гибкость в процессе программирования (позволяет пользователю более свободно выбирать «место» и «время» для программирования).

Вольтаж режима программирования определяет если будет использован низковольтный или высоковольтный режим. Использование низковольтного режима позволяет программировать PIC имея в доступноси только источники питания от 2.0В до 5.0В, но требует дополнительной ножки микроконтроллера. Использование только высоковольтных режимов позволяет переопределить ножку PGM микроконтроллера под пользовательские нужды, но требует наличия источника питания на 12В. В данной работе используется низковолтный режим поскольку тонкий клиент Orange Pi Lite не имеет возможности предоставить 12В питание.

Режим программирования для PIC16F627A/628A/648A позволяет программировать ячейки программной памяти, ячейки памяти данных, конфигурациионное слово, а также специальные 7 ячеек, которые используются для храненния ID устройства.

Команды программирования и их операнды в режиме программирования определяют обмен информацией с PIC. Операнды и команды передаются в микроконтроллер и/или обратно через серийные кабели.

3.2.2. Основные определения и структуры данных

Ячейки памяти в данном микроконтроллере есть как 14, так и 8 битные ячейки. На PIC16F627A/628A/648A реализованна Гарвардская архитектура с отдельными шинами для инструкций и данных, что позволяет 14-разрядным инструкциям работать с 8-разрядными данными.

Пространство програмной памяти отведенное пользователю простирается от 0x0000 до 0x1FFF. В режиме программирования, пространство программной памяти простирается от 0x0000 до 0x3FFF, с первой половиной (от 0x0000-0x1FFF), которая отведена программной памяти, и второй половиной (0x2000-0x3FFF), которая отведена конфигурационной памяти. Все другие адреса в конфигурационной памяти PIC зарезервированы и не могут быть запрограммированы пользователем.

В пространство конфигурационной памяти (адреса 0x2000 - 0x2007) можно попасть через передачу в PIC специальной команды «Загрузить данные для конфигурационной памяти». Только адреса 0x2000-0x200F конфигурационного пространства памяти физически реализованы. Однако, только ячейки 0x2000 вплоть до 0x2007 доступны для программирования. Остальные ячейки зарезервированны. Переход по адресу за пределами 0x200F будет физически осуществлять доступ к пользовательской памяти.

Пространство ПЗУ памяти простирается от 0x00 до 0xFF и находится отдельно от пространства программной памяти и пространства оперативной памяти. Для ПЗУ реализуются только нижние 128 байт для устройств PIC16F627A/628A, в то время как для PIC16F648A реализуются все 256 байт. Программирование ПЗУ памяти данных использует тот же программный счетчик что и для программирования конфигурационной и програмной памяти, однако только нижние биты декодируется и используются. Поэтому перед программированием ПЗУ необходимо чтобы программный счетчик указывал на 0x0000 или 0x2000.

Создание структуры для представления памяти микроконтроллера. Ключевым элементом в представлении памяти микроконтроллера является массив типа uint16_t размером с память PIC. В массиве хранится пользовательская программа непосред-

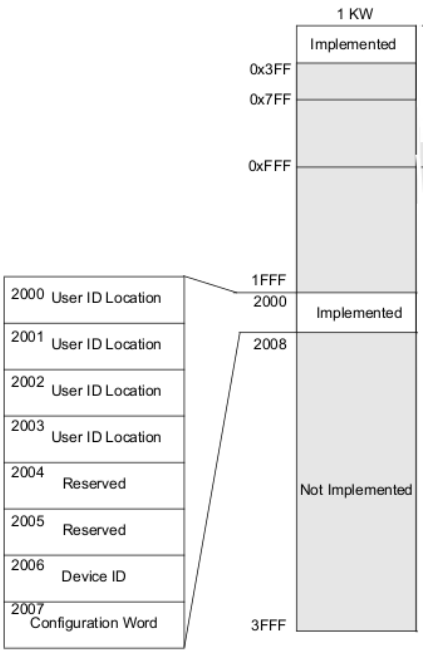

Рис. 1: Карта памяти микроконтроллера. Белым выделены области для пользовательского кода.

ственно перед её побитовой передачей на микроконтроллер. Из 16 битов отведенных под тип uint16_t для хранения данных и команд используются только нижние 8 и 14 байт соответственно.

Для того чтобы ускорить процесс прошивки микроконтроллера вводятся переменные program_memory_used_cells, и program_memory_max_used_address. Они обозначают общее колличество задействованных программой ячеек памяти, и самый высокий адрес в програмной памяти. Таким образом можно заметить условия при которых можно досрочно закончить программирование програмной памяти PIC и перейти к следующей стадии программирования.

```
struct picmemory {
    uint16_t  program_memory_used_cells;
    uint16_t  program_memory_max_used_address;

    uint8_t   has_configuration_data;
    uint8_t   has_eeprom_data;

    uint16_t *data;     /* 14-bit and 8-bit data */
    uint8_t  *filled;   /* 1 if this cell is used */
};
```

Внутренний програмный счетчик может увеличиваться от 0x0000 до конца реализованной програмной памяти 0x03FF (или 0x07FF, или 0x0FFF в зависимости от модели PIC) после чего он вновь обернется на адресс 0x0000. Для включения высоких бит програмного счетчика и перехода к программированию конфигурационной памяти, необходимо послать специальную команду. Послее нее програмный счетчик будет оперировать в

пространстве от 0x2000 до 0x3FFF (по достижении 0x3FFF он будет вновь оборачиваться на 0x2000). Единственным способом сбросить верхние биты програмного счетчика вновь на 0x0000 это покинуть и повторно войти в режим программирования.

Ячейки для ID пользователя отображаются на адреса [0x2000 : 0x2003] и являются частью конфигурционной памяти. Пользователь может хранить идентификационные данные (идентификатор пользователя) в четырех локациях для идентификатора пользователя. Этот идентификатор всегда можно считать корректно, даже если будет включена защита кода.

Питание в режиме программирования. Для PIC16F627A/628A/648A требуется один источник питания с VDD (2.0V до 5.5V) и VPP от 12В до 14В, или же VPP от 4.5В до 5.5В, при использовании низковольтного программирования. Оба источника должны иметь разрешение как минимум в 0,25В. В данной работе используется режим низковольтного программирования.

Команды программирования в режиме программирования, обмен информацией с PIC определяеться набором команд и их операндами которые передаются в микроконтроллер и/или обратно через серийные кабели. Общая форма для всех последовательностей команд состоит из 6-битовой команды и условно 16-битного слова данных. И команды и слова данных передаются начиная с наименее значимого бита (LSB first). Для других серий и линеек типа PIC набор команд может отличаться.

Command	Mapping (MSb ... LSb)						Data
Load Configuration	X	X	0	0	0	0	0, data (14), 0
Load Data for Program Memory	X	X	0	0	1	0	0, data (14), 0
Load Data for Data Memory	X	X	0	0	1	1	0, data (8), zero (6), 0
Increment Address	X	X	0	1	1	0	
Read Data from Program Memory	X	X	0	1	0	0	0, data (14), 0
Read Data from Data Memory	X	X	0	1	0	1	0, data (8), zero (6), 0
Begin Programming Only Cycle	X	0	1	0	0	0	
Bulk Erase Program Memory	X	X	1	0	0	1	
Bulk Erase Data Memory	X	X	1	0	1	1	

Рис. 2: Последовательности 6-разрядных команд для PIC16F627A/628A/648A

I/O Выходы используемые для программирования. Положительный входной сигнал на ножке RB4 называемой PGM вводит микроконтроллер в режим низковольтного программирования (если данная опция была включена в конфигурационном слове микроконтроллера). Ножка RB7 называемая DATA, настраивается как вход и используется для по-битовой передачи данных программых в микроконтроллер. Ножка RB6 называемая CLOCK, также настраивается на прием входного сигнала и используется для синхронизации состояния напряжения на ножке RB7. Во время падения напряжения на ножке RB6, микроконтроллер считает следующий бит с ножки RB7. Ножка MCLR/Vpp изпользуется для выбора режима программирования. В PIC16F627A/628A/648A, высокое напряжение для работы с ячейками памяти генерируется автоматически. Для активации режима программирования, необходимо применить высокое напряжение ко входу MCLR. Поскольку MCLR используется на уровне источника, это означает, что MCLR не тянет какиой-либо значительный ток. Ножка VDD предоставляет 5В необходимые для стабильной работы микроконтроллера в штатном режиме. Ножка VSS определяет напряжение на уровне земли.

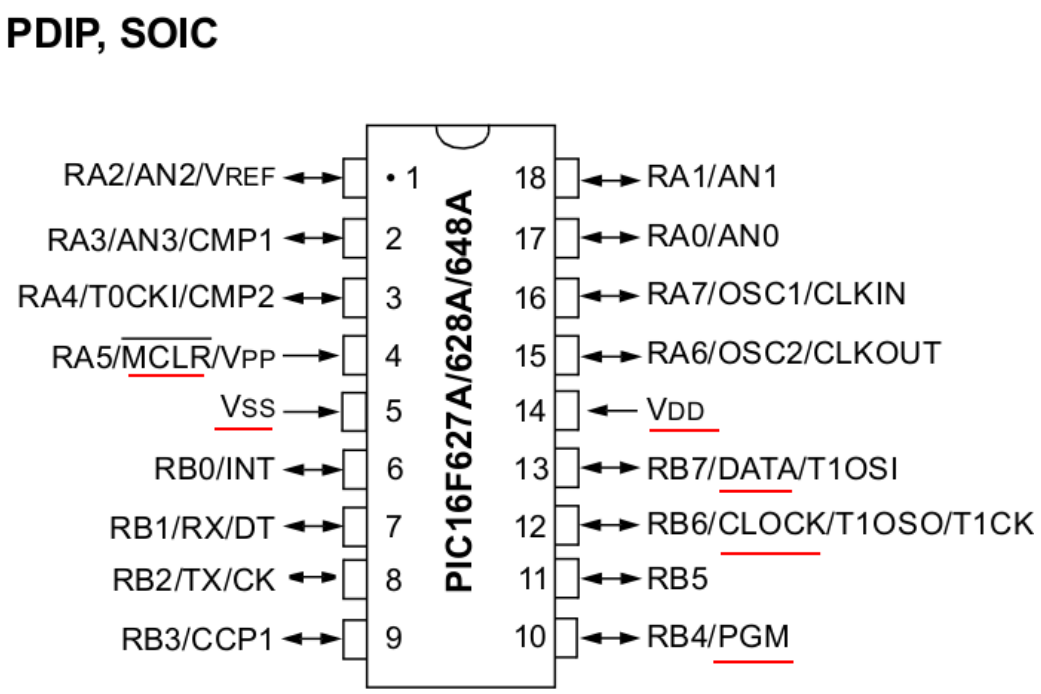

Рис. 3: I/O выходы необходимые для программирования в серийном режиме

Абстрагирование программатора от конкретной модели микроконтроллера семейства PIC делается с помощью определения дополнительной структуры в которой для каждой конкретной модели хранится информация о карте памяти, о командах программирования, о задержках необходимых между сигнальными последовательностями.

```
struct picmicro {
    uint16_t device_id;
    char     name[16];
    size_t   program_memory_size;
    size_t   data_memory_size;

    int program_cycle_time;     /* in microseconds */   // T_PROG
    int eeprom_program_cycle_time;  // T_DPROG
    int bulk_erase_cycle_time;      // T_ERA

    uint8_t load_configuration_cmd;
    uint8_t load_data_for_program_memory_cmd;
    uint8_t load_data_for_data_memory_cmd;
    uint8_t read_data_from_program_memory_cmd;
    uint8_t read_data_from_data_memory_cmd;
    uint8_t increment_address_cmd;
    uint8_t begin_erase_programming_cycle_cmd;
    uint8_t begin_programming_only_cycle_cmd;
    uint8_t bulk_erase_program_memory_cmd;
    uint8_t bulk_erase_data_memory_cmd;
};
```

Для микроконтроллеров PIC16F627A/628A/648A используется нижеследующее

определение экземпляра данной структуры.

```
const struct picmicro pic16f628a = {
    /* General */
    .device_id =                        0x1060,
    .name =                            "pic16f628a",
    .program_memory_size =                0x800,
    .data_memory_size =                  128,

    /* Time intervals in microseconds */
    .program_cycle_time =                4000,
    .eeprom_program_cycle_time =            6000,
    .bulk_erase_cycle_time =             6000,

    /* Commands */
    .load_configuration_cmd =              0x00,
    .load_data_for_program_memory_cmd =      0x02,
    .load_data_for_data_memory_cmd =       0x03,
    .read_data_from_program_memory_cmd =     0x04,
    .read_data_from_data_memory_cmd =      0x05,
    .increment_address_cmd =             0x06,
    .begin_erase_programming_cycle_cmd =     0xFF,
    .begin_programming_only_cycle_cmd =      0x08,
    .bulk_erase_program_memory_cmd =       0x09,
    .bulk_erase_data_memory_cmd =          0x0B
};
```

3.2.3. Описание алгоритма

Чтение файла с программой пользователя, это первый шаг программатора. Следующим шагом является проверка входного файла на корректность. Для этого используются особенности формата INTEL HEX8M. В частности приведенные в конце строки проверочные суммы и адреса для данных.

```
1 :02000000FE2FD1
2 :10001000FF3FFF3FFF3FFF3FFF3FFF3FFF3FFF3FD0
3 :100FD40007308312031390F0085018316031385D1
4 :100FE400831203138601831603138601FF308312D1
5 :0C0FF4008500FF308600F82F8301EA2FF3
6 :02400E00D0FFE1
7 :00000001FF
```

Рис. 4: Пример входного файла к программатору

По мере чтения входного файла в соответствующий адрес в массиве структуры «picmemory» вставляются значения из входного файла. Поскольку на тонком клиенте Orange Pi Lite установлен little-endian процессор компании Allwinner модели H3, а в формате INTEL HEX8M данные записываются в порядке MSB, то при вставке в массив необходимо поменять местами верхний и нижний байты.

В цикле происходит прочтение входного файла и создание структуры для представления памяти микроконтроллера, а также подсчет проверочных значений.

```
for (i = 0; i < byte_count / 2; i++) {
```

```
        nread = sscanf(&line[9+4*i], "%4hx", &data);
        uint16_t pic_data = swap_uint16(data);
        if (nread != 1) {
                fprintf(stderr, "Error: cannot read data.\n");
                free_picmemory(&pm);
                return NULL;
        }
        if (debug)
                fprintf(stderr, "  data        = 0x%04X (file) = 0x%04X (micro)\n", data, pic_data);
        checksum_calculated += (data >> 8) & 0xFF;
        checksum_calculated += data & 0xFF;

        if (address + i < 0x2000) {
                pm->program_memory_used_cells       += 1;
                pm->program_memory_max_used_address  = address + i;
        } else if (0x2000 <= address + i && address + i < 0x2008)
                pm->has_configuration_data = 1;
        else if (address + i >= 0x2100)
                pm->has_eeprom_data = 1;

        pm->data[address + i]   = pic_data;
        pm->filled[address + i] = 1;
}
```

Программатор работает через посылание последовательности команд и данных, введенных в серийном режиме в котором бит на линии данных загоняется в микроконтроллер на падающем фронте напряжения на линии часов. Команда + данные, вводятся последовательно, через линию часы и линию данных, которые с аппаратной точки зрения являются входными линиями изпользующими триггеры Шмитта для различения напряжения 0 или 1.

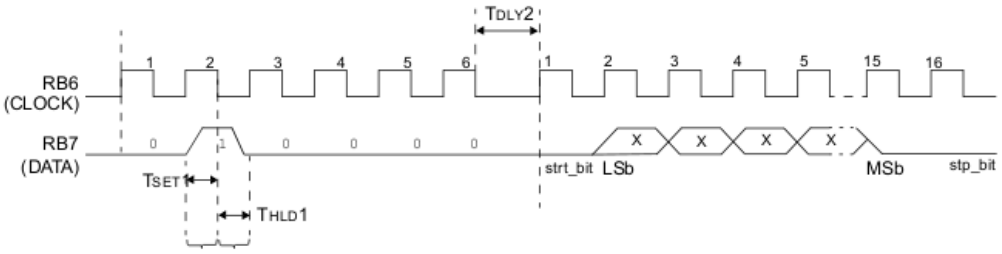

Рис. 5: Временные интервалы для побитовой передачи команды «Загрузка данных в програмную память»

Минимальные время установки и удержания приписываются каждому сигнал на ножке данных (описанные в таблице ниже, вместе с ограничением по напряжению) по отношению к падающему фронту напражения на линии часов. Командам на чтение и запись, которые требуют передачи данных связанных с ними, требуется минимальная задержка между передачей команды и передачей данных.

AC/DC Characteristics	**Standard Operating Conditions (unless otherwise stated)** Operating Temperature: 0°C ≤ TA ≤ +70°C Operating Voltage: 4.5V ≤ VDD ≤ 5.5V					
Characteristics	**Sym**	**Min**	**Typ**	**Max**	**Units**	**Conditions/ Comments**
General						
VDD level for word operations, program memory	VDD	2.0	—	5.5	V	
VDD level for word operations, data memory	VDD	2.0	—	5.5	V	
VDD level for Bulk Erase operations, program and data memory	VDD	4.5	—	5.5	V	
High voltage on $\overline{\text{MCLR}}$	VIHH	10.0	—	13.5	V	
$\overline{\text{MCLR}}$ rise time (VSS to VIHH) for Programming mode entry	TVHHR	—	—	1.0	µs	
Hold time after $\overline{\text{MCLR}}$↑	TPPDP	5	—	—	µs	
Hold time after LVP↑	TLVPP	5	—	—	µs	
(CLOCK, DATA) input high level	VIH1	0.8 VDD	—	—	V	Schmitt Trigger input
(CLOCK, DATA) input low level	VIL1	—	—	0.2 VDD	V	Schmitt Trigger input
CLOCK, DATA setup time before $\overline{\text{MCLR}}$↑	TSET0	100	—	—	ns	
Hold time after VDD↑	THLD0	5	—	—	µs	
Serial Program/Verify						
Data in setup time before clock↓	TSET1	100	—	—	ns	
Data in hold time after clock↓	THLD1	100	—	—	ns	
Data input not driven to next clock input (delay required between command/data or command/ command)	TDLY1	1.0	—	—	µs	
Delay between clock↓ to clock↑ of next command or data	TDLY2	1.0	—	—	µs	
Clock↑ to data out valid (during read data)	TDLY3	—	—	80	ns	
Programming cycle time	TPROG	—	—	4	ms	
Data EEPROM Programming cycle time	TDPROG	—	—	6	ms	
Bulk Erase cycle time	TERA	—	—	6	ms	

Рис. 6: Спецификация AC/DC для поднятия/опускания и удержания линий данных и часов

Алгоритм записи данных в память программы приведен в нижеследующем графе.

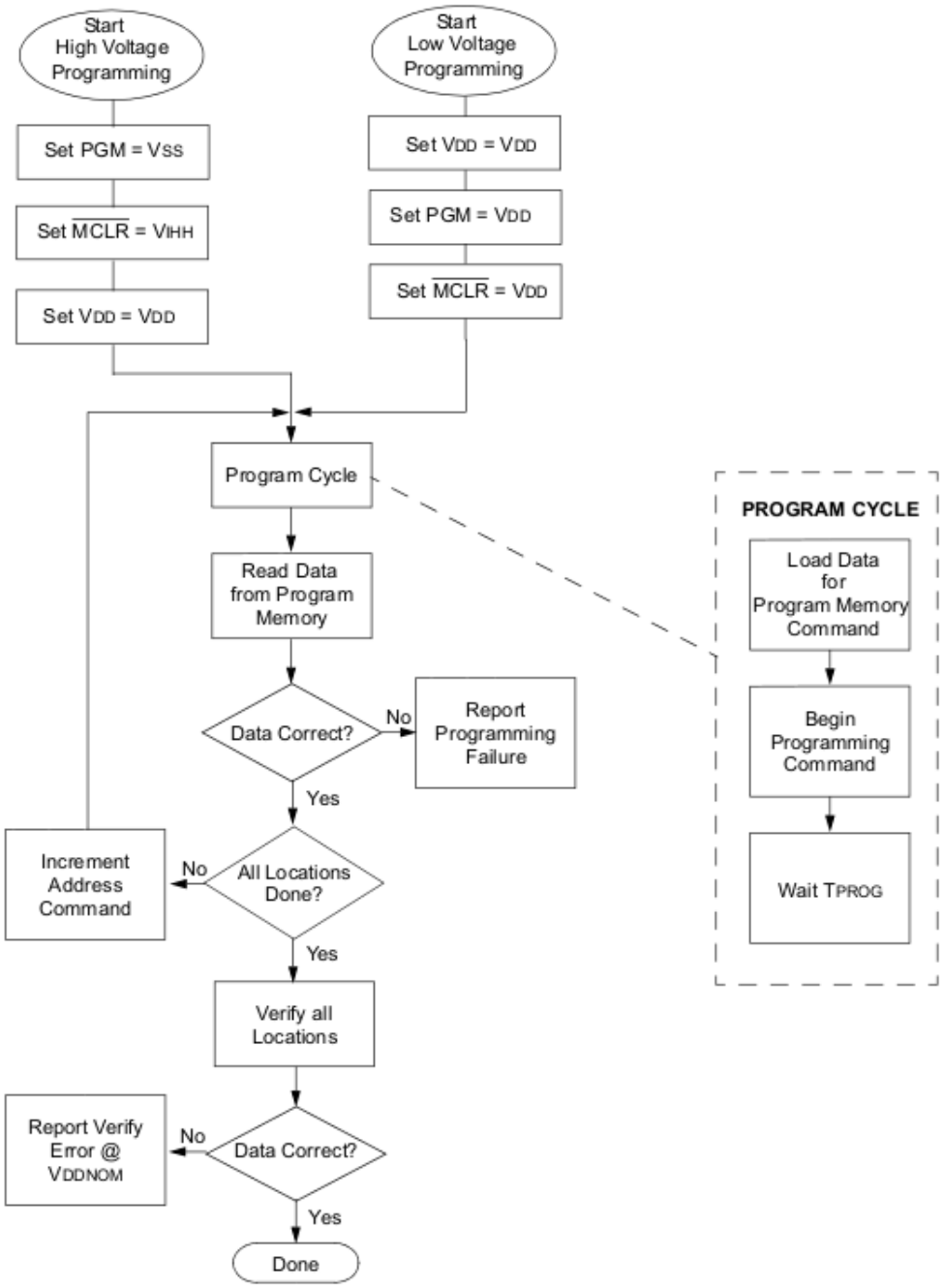

Рис. 7: Граф алгоритма записи данных в програмную память микроконтроллера

После создания карты памяти которая должна быть загруженна в микроконтроллер, следующим шагом является непосредственная передача этих данных в PIC. Для этого необходимо получить доступ к GPIO выходам тонкого клиента Orange Pi Lite. Со стороны Orange Pi Lite надо было посмотреть на поставляемую с ним схемотехнику платы чтобы определить какие выходы PGIO наиболее подходят для работы программатора. Сигналы

передаваемые на ножки микроконтроллера имеют строгие временные рамки, и для того что бы в них вписаться необходимо использовать механизм «memory mapped file» для включения и отключения сигналов нв выходах GPIO на уровне процессора.

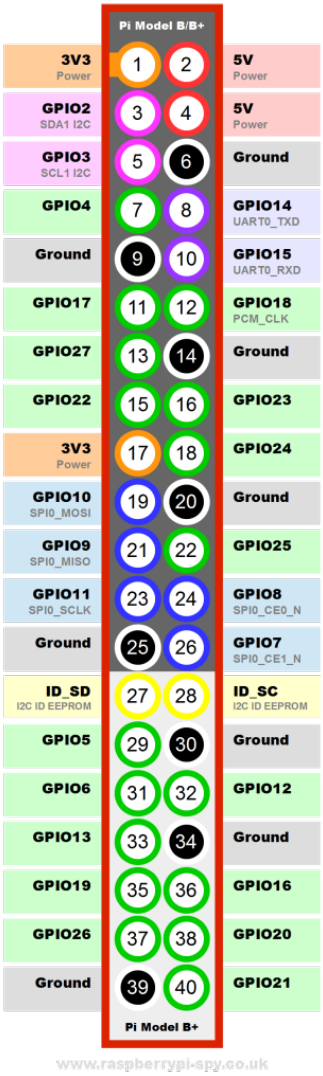

Рис. 8: Разъем GPIO выходов модели Rasberry Pi B+ которая таже используется и на Orange Pi Lite

Посмотрев на схемотехнику для выходов GPIO, решено было выбрать 5 последовательных выходов 29, 31, 33, 35, 37 которые подсоедены к процессору H3 соответственно на ножках 7, 8, 9, 10, 20 порта A.

Из схемотехники ниже понятно что ножки PA7, PA8, PA9, PA10, PA20 подключаются к порту A процессора и выходят на GPIO.

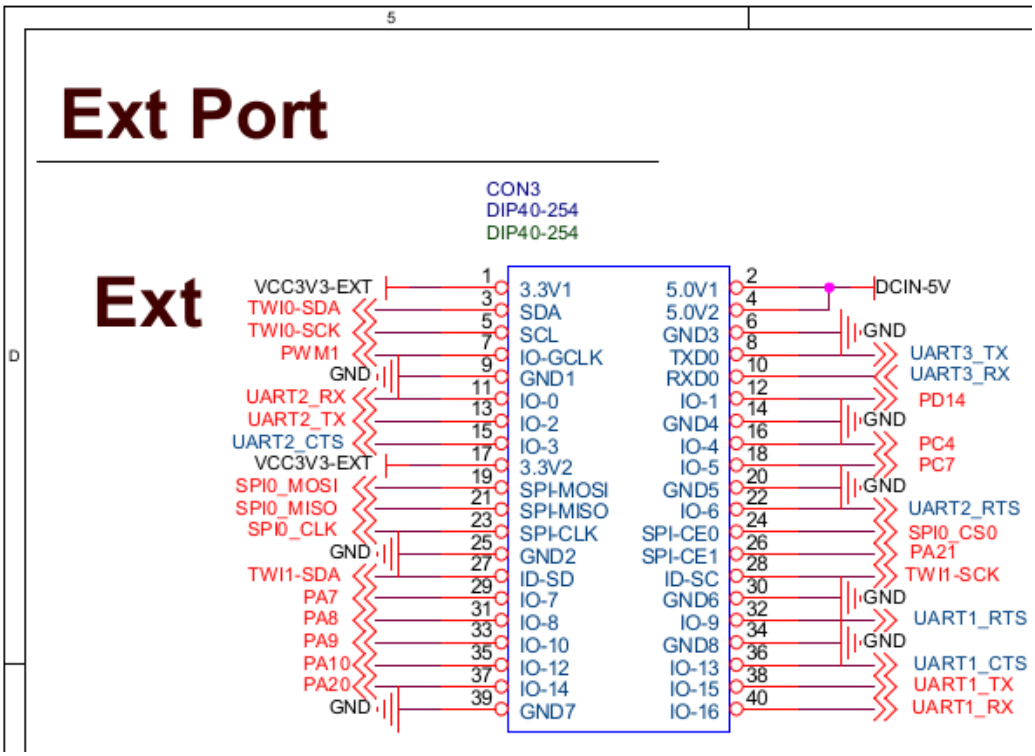

Рис. 9: Схемотехника для выходов GPIO Orange Pi Lite с процессором H3

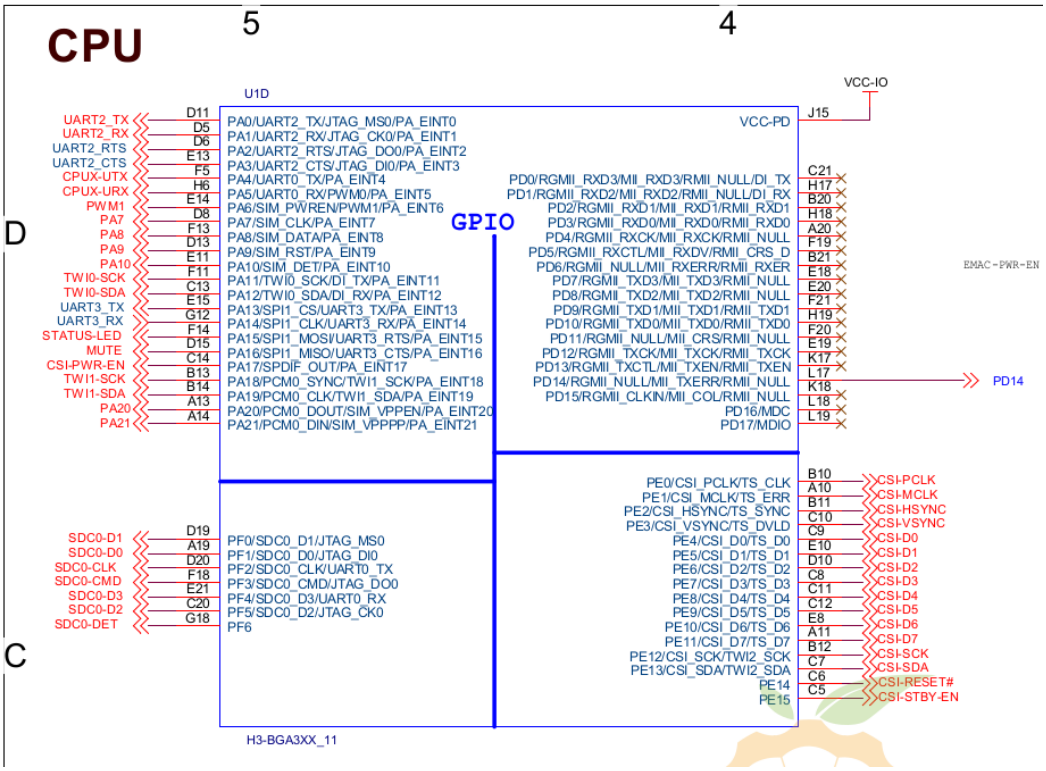

Рис. 10: Схемотехника для процессора H3 Orange Pi Lite

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.509000 81 01-1				
Инв. №подл.	Подп. и дата	Взам. инв. №	Инв. №дубл.	Подп. и дата

Получение доступа к выходам GPIO, осуществляется через манипуляцию регистрами отвечающими за порт A напрямую. Для этого делается маппинг регистров настройки и состояния порта A в адресоное пространство программы. Значения для адресов регистров и их размера берутся из руководства инженера к процессору H3. Значения адресов всех регистров влияющих на состояние порта A вписанны в код программы. Далее приведена краткая сводка констант в которые были записаны необходимые адреса.

```
// AllWinner H3 datasheet, page 86, page 316
uint32_t CCU_BASE = 0x01C20000ul;       // Port controller
uint32_t PIO_OFS =  0x00000800ul;       // GPIO offset

// AllWinner H3 datasheet, page 86, block size is 1024 = 1K = 0x400 bytes
uint32_t PIO_MAP_LEN = 0x2000;          // Port controller end (0x01C20BFF) - Port controller start (0x01C20800)

// AllWinner H3 datasheet, page 316
uint32_t PA_CFG0_OFS = 0x00000000ul;    //  Port A Configure Register 0
uint32_t PA_CFG1_OFS = 0x00000004ul;    //  Port A Configure Register 1
uint32_t PA_CFG2_OFS = 0x00000008ul;    //  Port A Configure Register 2
uint32_t PA_DAT_OFS  = 0x00000010ul;    //  Port A Data Register
```

Следом были написанны функции для манипулирования этими регистрами. Ниже приведены фрагменты функций которые ипользуются чтобы выставить состояние ножки GPIO на выход (output) и чтобы менять напряжение на дфнной ножке с 0В до 3.3В. Для регистров бит 0 - это наименее значимый бит.

Ниже приведены фрагменты функций которые ипользуются чтобы выставить состояние ножки GPIO на выход (output) и чтобы менять напряжение на данной ножке с 0В до 3.3В. Для регистров бит 0 - это наименее значимый бит.

```
// получение указателя на адрес по которому хранится регистр
volatile uint32_t* get_data_reg(int pin)
{
    volatile uint32_t* data_reg = (volatile uint32_t * )(gpio + PA_DAT_OFS);
    return data_reg;
}

// создает новое измененное значение для записи в регистр
uint32_t modify_data(uint32_t data, int pin, int value)
{
    uint32_t new_data;
    if (value) {
        new_data = data | (1U << pin);
    }
    else {
        new_data = data & ~(1U << pin);
    }
    return new_data;
}

// выставляет на ножке номер "pin" значение напряжения 0 или 1.
void gpio_wr(int pin, int value) {
    volatile uint32_t* data_reg = get_data_reg(pin);
    uint32_t data = *data_reg;
    uint32_t new_data = modify_data(data, pin, value);
    *(data_reg) = new_data;
}
```

```
// модифицирует копию текущих настроек чтобы ножка номер "pin" оказалась настроена как ВЫХОД
// возвращает новое значение которое должно быть записано в регистр
uint32_t modify_cfg_output(uint32_t cfg, int pin)
{
    /* Write value 001 to position of pin, each pin is z001 ('z' is dont care) */
    uint32_t new_cfg = cfg & ~(6U << 4 * (pin % 8));
    uint32_t new_new_cfg = new_cfg | (1U << 4 * (pin % 8));
    return new_new_cfg;
}


// модифицирует регистр настроек порта A чтобы ножка номер "pin" оказалась настроена как ВЫХОД
// Pin is one of PA7, PA8, PA9, PA10, PA20
void gpio_output(int pin)
{
    volatile uint32_t* cfg_reg = get_cfg_reg(pin);
    uint32_t cfg = *cfg_reg;
    uint32_t new_cfg = modify_cfg_output(cfg, pin);
    *(cfg_reg) = new_cfg;
}


// модифицирует копию текущих настроек чтобы ножка номер "pin" оказалась настроена как ВХОД
// возвращает новое значение которое должно быть записано в регистр
uint32_t modify_cfg_input(uint32_t cfg, int pin)
{
    /* Write value 000 to position of pin, each pin is z000 ('z' is dont care) */
    uint32_t new_cfg = cfg & ~(7U << 4 * (pin % 8));
    return new_cfg;
}


// модифицирует регистр настроек порта A чтобы ножка номер "pin" оказалась настроена как ВХОД
// Pin is one of PA7, PA8, PA9, PA10, PA20
void gpio_input(int pin)
{
    volatile uint32_t* cfg_reg = get_cfg_reg(pin);
    uint32_t cfg = *cfg_reg;
    uint32_t new_cfg = modify_cfg_input(cfg, pin);
    *(cfg_reg) = new_cfg;
}
```

Последним шагом произходящим на уровне программы необходимо пройтись по созданной карте памяти микроконтроллера и посылая в PIC команд и их операнды через GPIO передать в микророконтроллер необходимую информацию.

```
// Bulk erase the chip, and then write contents of the .hex file to the PIC
void pic_write(const struct picmicro *pic, char *infile, int debug)
{
    uint16_t addr;
    struct picmemory *pm;

    pm = read_inhx16(infile, debug);

    /* Turn pic on. Give supply power. */
    pic_enter_lvp();

    /* Bulk erase the chip first */
```

```
    pic_bulk_erase(pic, debug);

    /* Write Program Memory */
    for (addr = 0; addr <= pm->program_memory_max_used_address; addr++) {
        if (pm->filled[addr]) {
            pic_send_cmd(pic->load_data_for_program_memory_cmd);
            pic_load_data(pm->data[addr]);
            // trigger
            pic_send_cmd(pic->begin_programming_only_cycle_cmd);
            usleep(pic->program_cycle_time);
        }
        pic_send_cmd(pic->increment_address_cmd);
    }

    /* Write Configuration Memory */
    if (pm->has_configuration_data) {
        pic_send_cmd(pic->load_configuration_cmd);
        pic_load_data(pm->data[0x2000]);
        for (addr = 0x2000; addr < 0x2008; addr++) {
            if ((addr <= 0x2003 || addr == 0x2007) && pm->filled[addr]) {
                pic_send_cmd(pic->load_data_for_program_memory_cmd);
                pic_load_data(pm->data[addr]);
                // trigger
                pic_send_cmd(pic->begin_programming_only_cycle_cmd);
                usleep(pic->program_cycle_time);
            }
            pic_send_cmd(pic->increment_address_cmd);
        }
    }

    pic_exit_lvp();
    free_picmemory(&pm);
}
```

3.3. Метод организации входных и выходных данных

3.3.1. Описание метода входных и выходных данных

Входными данными для работы программатора являются скомпилированный файл программы, микроконтроллер подключенный к плате, а также (для обеспечения взаимодействия с пользователем) клавиатура и/или мышь. Входной файл данных может быть созданн в любой среде разработки и любым компилятором поддершивающим формат INTEL HEX8M. Примером такой среды разработки является MPLAB X (https://www.microchip.com/, разработчик: коммерческая организация Microchip Ltd.).

1. Из-за огромного количества серий микроконтроллеров поддерживать их все не представляется возможным. Поэтому программатор работает только с микроконтроллерами PIC серии 16F, конкретно с линейками 627A / 628A / 648A.
2. Входной файл программы должен соответствовать формату INTEL HEX8M. По сравнению с двумя другими часто встречающимися форматами INTEL HEX8S, INTEL HEX32, данный формат наиболее оптимально подходит под серию 16F. В силу того что память 14-битных микроконтроллеров не превышает 64 килобайт (здесь подходит формат HEX32) и програмное слово не нуждаеться в разбиении на высокий и

низкий байт как в 16-битных микроконтроллерах (здесь подходит формат HEX8S).

3. Пользователь имеет возможность модифицировать следующие входные данные в процессе работы программы в условиях графического интерфейса и перед запуском программы в командной строке:
   (a) Указать что требуется запись EEPROM памети без модификации програмной памяти микроконтроллера.
   (b) Указать что требуется проверить входной файл на ошибки.
   (c) Указать что требуется записать входной файл в програмную память и в EEPROM память микроконтроллера.
   (d) Поменять уровень колличества сообщений выводимих программой пользователю.
   (e) Отменить процесс программирования.

Выходными данными для программатора является запрограммированный микроконтроллер, данные на экране и индикатор программирования на плате программатора.

3.4. Выбор состава технических средств

3.4.1. Состав технических и програмных средств

Для оптимальной работы приложения необходимы следующие системные требования:

1. Тонкий клиент Orange Pi Lite, оснащенный:
   (a) Обязательно процессор Allwinner H3 с тактовой частотой 1 гигагерц (ГГц) или выше;
   (b) 0.5 ГБ оперативной памяти (ОЗУ);
   (c) 0.5 ГБ свободного места на жестком диске;
   (d) Периферия: выход GPIO типа Rasberry Pi B+
2. Опционально: Компьютер для удаленного доступа к Orange Pi Lite, оснащенный:
   (a) Обязательно 64-разрядный (x64) процессор с тактовой частотой 1 гигагерц (ГГц) или выше;
   (b) 1 ГБ оперативной памяти (ОЗУ);
   (c) 1.5 ГБ свободного места на жестком диске;
   (d) Wifi модулем (если Orange Pi Lite подключен к сети, то можно воспользоваться и стандартным Ethernet портом) или TTL переходником для подключения к тонкому клиенту Orange Pi Lite.
3. Монитор
4. Мышь
5. Клавиатура

4. Технико-экономические показатели

4.1. Оринтировочная экономическая эффективность

Оринтировочная экономическая эффективность не рассчитывается.

4.2. Экономические преимущества разработки

Существующими коммерчискими аналогами данного приложения являются серийные программаторы комрании Microchip Ltd. В силу того что схематехника и код данного программатора распростроняется бесплатно, экономически выгодным аналогом к нему являются известный программатор K150 (подключающийся к компьютеру через USB), а также разнообразные схемы основанные на портах LDP (для принтера) и COM портах. Однако схематехника K150 в несколько раз сложнее, к тому же для его работы требуются два уже заранее запрограммированных микроконтроллера, поэтому собрать его в домашних условиях невозможно (требуется приобрести 2 уже запрограммированных микроконтроллера). Варианты основанные на портах LDP и COM также не подходят поскольку эти потры морально устарели и все реже и реже присутствую на современных компьютерах. Данная разработка позволяет использовать совремный и актуальный тонкий клиент Orange Pi Lite в качестве программатора микроконтроллеров, к тому же все части для сборки программатора стоят гораздо дешевле чем покупка официального прибора от Microchip, и наконец они предельно просты и позволяют сборку программатора в домашних условиях.

5. Источники, используемые при разработке

5.1. Список используемой литературы

1. DS40044G PIC16F627A/628A/648A Data Sheet
   Microchip Ltd.
   www.microchip.com

2. DS33014L MPASM Assembler, MPLINK Object Linker, MPLIB Object Librarian User's Guide
   Microchip Ltd.
   www.microchip.com

3. DS33023A PICmicro Mid-Range MCU Family Reference Manual
   Microchip Ltd.
   www.microchip.com

4. DS41196G PIC16F627A/628A/648A EEPROM Memory Programming Specification
   Microchip Ltd.
   www.microchip.com

5. Schematics Index ORANGE PI Lite Ver 1.1
   Orange Pi
   www.orangepi.com

6. Version 1.2 Allwinner H3 Datasheet Quad-Core OTT Box Processor
   Allwinner Ltd.
   www.allwinnertech.com

6. Приложение 1. Терминология

6.1. Терминология

EEPROM электрически стираемое перепрограммируемое ПЗУ (ЭСППЗУ), один из видов энергонезависимой памяти (таких, как PROM и EPROM). Память такого типа может стираться и заполняться данными до миллиона раз.

Архитектура набора команд (англ. instruction set architecture, ISA) часть архитектуры компьютера, определяющая программируемую часть ядра микропроцессора. На этом уровне определяются реализованные в микропроцессоре конкретного типа

Язык ассемблера (англ. assembly language) машинно-ориентированный язык низкого уровня с командами, не всегда соответствующими командам машины, который может обеспечить дополнительные возможности вроде макрокоманд.

MPLAB интегрированная среда разработки, представляющая собой набор программных продуктов, предназначенная для облегчения процесса создания, редактирования и отладки программ для микроконтроллеров семейства PIC, производимых компанией Microchip Technology. Среда разработки состоит из отдельных приложений, связанных друг с другом и включает в себя компилятор с языка ассемблер, текстовый редактор, программный симулятор и средства работы над проектами, также среда позволяет использовать компилятор с языка C.

контрольный таймер, англ. Watchdog timer аппаратно реализованная схема контроля над зависанием системы. Представляет собой таймер, который периодически сбрасывается контролируемой системой. Если сброса не произошло в течение некоторого интервала времени, происходит принудительная перезагрузка системы. В некоторых случаях сторожевой таймер может посылать системе сигнал на перезагрузку («мягкая» перезагрузка), в других же — перезагрузка происходит аппаратно (замыканием сигнального провода RST или подобного ему).

Внутрисхемное программирование (англ. In-System Programming, сокр. ISP) технология программирования электронных компонентов (ПЛИС, микроконтроллеры и т. п.), позволяющая программировать компонент, уже установленный в устройство. До появления этой технологии компоненты программировались перед установкой в устройство, для их перепрограммирования требовалось их извлечение из устройства.

Универсальный асинхронный приёмопередатчик (англ. UART) узел вычислительных устройств, предназначенный для организации связи с другими цифровыми устройствами. Преобразует передаваемые данные в последовательный вид так, чтобы было возможно передать их по одной физической цифровой линии другому аналогичному устройству. Метод преобразования хорошо стандартизован и широко применяется в компьютерной технике (особенно во встраиваемых устройствах и системах на кристалле (SoC)).

7. Приложение 2. Формат INTEL HEX8M (.hex)

7.1. Формат INTEL HEX8M (.hex)

INTEL HEX8M — это формат, разработанный для обмена скомпилированным кодом но не в бинарном а в текстовом формате. В формате указывается связь между адресом и байтовым значением, которое должно находиться по этому адресу.

Формат включает в себя подсчет проверочной суммы, котоая записывается последним байтом в конце строки.

Ниже приведен пример описания простой прогаммы для мигания светодиодом на микроконтроллере PIC16F628A в данном формате:

```
:10000000000000000000000000000000000000F0
:0400100000000000EC
:1000320000002800400068000A800E800C80028016D
:100042006801A9018901EA01280208026A02BF02C5
:10005200E002E80228036803BF03E803C8030804B8
:1000620008040804030443050306E807E807FF0839
:06007200FF08FF08190A57
:00000001FF
```

8. Приложение 3. Описание классов

8.1. Файл rpp.c

Структуры данных

- struct **picmemory**
  Structure for the pic memory map.
- struct **picmicro**
  Structure representing the various settings that are used in programming a specific model of the microcontroller.

Макросы

- #define **_BSD_SOURCE**
- #define **PIC_VDD** (7) /∗ Power ∗/
- #define **PIC_DATA** (8) /∗ Output ∗/
- #define **PIC_CLK** (9) /∗ Output ∗/
- #define **PIC_PGM** (10) /∗ Output ∗/
- #define **PIC_MCLR** (20) /∗ Output ∗/
- #define **DELAY** (40) /∗ microseconds ∗/
- #define **PICMEMSIZE** (0x2100 + 0x80)
- #define **PIC16F628A** 0x1060
- #define **GPIO_IN**(g) **gpio_input**(g)
- #define **GPIO_OUT**(g) **gpio_output**(g)
- #define **GPIO_SET**(g) **gpio_wr**(g, 0)
- #define **GPIO_CLR**(g) **gpio_wr**(g, 1)

Функции

- struct **picmemory** ∗ **read_inhx16** (char ∗infile, int debug)
  Read a file in Intel HEX 16-bit format and return a pointer to the picmemory struct on success, or NULL on error.
- bool **test_bit** (uint32_t n, int bit)
  Tests if a bit is set.
- void **print_hex** (uint32_t n)
  Prints a number in hex format to stdout.
- int **gpio_rd** (long pin)
  Reads input from GPIO pin.
- volatile uint32_t ∗ **get_data_reg** (int pin)
  Returns a pointer to the DATA register for a particular pin.
- uint32_t **modify_data** (uint32_t data, int pin, int value)
  Modifies an existing data value to toggle or clear a pin output.
- void **gpio_wr** (int pin, int value)
  Writes a value to the GPIO pin.
- volatile uint32_t ∗ **get_cfg_reg** (int pin)

Returns the address for the CONFIG register for a specific pin.

- uint32_t **modify_cfg_output** (uint32_t cfg, int pin)

  Modifies the current CONFIG register value to change pin settings to OUTPUT.
- void **gpio_output** (int pin)

  Changes GPIO pin settings into OUTPUT mode.
- uint32_t **modify_cfg_input** (uint32_t cfg, int pin)

  Modifies the current CONFIG register value to change pin settings to INPUT.
- void **gpio_input** (int pin)

  Changes GPIO pin settings into INPUT mode.
- void **setup_io** ()

  Set up a memory regions to access GPIO.
- void **close_io** ()

  Closes GPIO connections and releases memory regions.
- void **free_picmemory** (struct **picmemory** **ppm)

  Frees picmemory structure.
- void **pic_send_cmd** (uint8_t cmd)

  Send a 6-bit command to the PIC.
- void **pic_load_data** (uint16_t data)

  Load 14-bit data to the PIC (start bit, 14-bit data, stop bit.
- void **pic_enter_lvp** ()

  Put pic into low voltage programming mode.
- void **pic_exit_lvp** ()

  Exit from voltage programming mode and turn off the chip power.
- void **pic_bulk_erase** (const struct **picmicro** *pic, int debug)

  Bulk erase the chip.
- void **pic_write** (const struct **picmicro** *pic, char *infile, int debug)

  Bulk erase the chip, and then write contents of the .hex file to the PIC.
- void **usage** (void)

  Prints the usage guide for the program.
- int **main** (int argc, char *argv[])

  Main entry function.
- uint16_t **swap_uint16** (uint16_t val)

  Swapps the lower and higher bytes of the unint16_t.

Переменные

- uint32_t **CCU_BASE** = 0x01C20000ul
- uint32_t **PIO_OFS** = 0x00000800ul
- uint32_t **PIO_MAP_LEN** = 0x2000
- uint32_t **PA_CFG0_OFS** = 0x00000000ul
- uint32_t **PA_CFG1_OFS** = 0x00000004ul
- uint32_t **PA_CFG2_OFS** = 0x00000008ul
- uint32_t **PA_CFG3_OFS** = 0x0000000Cul
- uint32_t **PA_DAT_OFS** = 0x00000010ul

- const struct **picmicro pic16f628a**

  Instance of picmicro structure.
- const struct **picmicro** * **piclist** [ ] = {& **pic16f628a**, NULL}
- int **mem_fd**

  Memory file descriptor.
- volatile char * **gpio**

  Pointer to the base adress of GPIO module in the Allwinner H3 processor.

8.1.1. Функции

8.1.1.1 free_picmemory() void free_picmemory (struct picmemory ** ppm)

Frees picmemory structure.

Аргументы

ppm	structure to free

8.1.1.2 get_cfg_reg() volatile uint32_t* get_cfg_reg (int pin)

Returns the address for the CONFIG register for a specific pin.

Аргументы

pin	Number of the pin.

Возвращает

Pointer to the CONFIG register for this particular pin.

8.1.1.3 get_data_reg() volatile uint32_t* get_data_reg (int pin)

Returns a pointer to the DATA register for a particular pin.

Аргументы

pin	Pin to get the register address for.

Возвращает

Pointer to DATA register for the pin.

8.1.1.4 gpio_input() void gpio_input (int pin)

Changes GPIO pin settings into INPUT mode.

Pin is one of PA7, PA8, PA9, PA10, PA20

Аргументы

pin	Number of the pin.

8.1.1.5 gpio_output() void gpio_output (int pin)

Changes GPIO pin settings into OUTPUT mode.

Pin is one of PA7, PA8, PA9, PA10, PA20

Аргументы

pin	Number of the pin.

8.1.1.6 gpio_rd() int gpio_rd (long pin)

Reads input from GPIO pin.

Аргументы

pin	To read voltage from.

Возвращает

0 or 1

8.1.1.7 gpio_wr() void gpio_wr (

int pin,
int value)

Writes a value to the GPIO pin.

Аргументы

pin	Number of the pin.
value	0 or 1

8.1.1.8 main() int main (

int argc,
char * argv[])

Main entry function.

Аргументы

argc	
argv[]	

Возвращает

8.1.1.9 modify_cfg_input() uint32_t modify_cfg_input (

uint32_t cfg,
int pin)

Modifies the current CONFIG register value to change pin settings to INPUT.

Аргументы

cfg	Current cfg register value.
pin	Number of the pin.

Возвращает

New value that should be writtten to the config register for this pin.

8.1.1.10 modify_cfg_output()

```
uint32_t modify_cfg_output (
    uint32_t cfg,
    int pin )
```

Modifies the current CONFIG register value to change pin settings to OUTPUT.

Аргументы

cfg	Current cfg register value.
pin	Number of the pin.

Возвращает

New value that should be writtten to the config register for this pin.

8.1.1.11 modify_data()

```
uint32_t modify_data (
    uint32_t data,
    int pin,
    int value )
```

Modifies an existing data value to toggle or clear a pin output.

Аргументы

data	Existing DATA register value.
pin	Pin number to modify.
value	0 or 1

Возвращает

new value that should be written to the register.

8.1.1.12 pic_bulk_erase()

```
void pic_bulk_erase (
    const struct  picmicro * pic,
    int debug )
```

Bulk erase the chip.

Аргументы

pic	Settings to use while erasing.
debug	1 if you want debug info printed.

8.1.1.13 pic_load_data()

void pic_load_data (
uint16_t data)

Load 14-bit data to the PIC (start bit, 14-bit data, stop bit.
lsb first)

Аргументы

data	Data to be loaded

8.1.1.14 pic_send_cmd()

void pic_send_cmd (
uint8_t cmd)

Send a 6-bit command to the PIC.

Аргументы

cmd	Hex code of the command

8.1.1.15 pic_write()

void pic_write (
const struct picmicro * pic,
char * infile,
int debug)

Bulk erase the chip, and then write contents of the .hex file to the PIC.

Аргументы

pic	Settings for the pic to be programmed.
infile	input file name
debug	1 if you want debug info printed

8.1.1.16 print_hex()

void print_hex (
uint32_t n)

Prints a number in hex format to stdout.

Аргументы

n	Number to print.

8.1.1.17 read_inhx16()

struct picmemory * read_inhx16 (
char * infile,
int debug)

Read a file in Intel HEX 16-bit format and return a pointer to the picmemory struct on success, or NULL on error.

Аргументы

infile	Input file name
debug	1 if you want debug info printed

Возвращает

Structure for the pic memory map

8.1.1.18 setup_io()

void setup_io ( )

Set up a memory regions to access GPIO.

Memory map Allwinner H3 processor registers.

8.1.1.19 swap_uint16()

uint16_t swap_uint16 (
uint16_t val)

Swapps the lower and higher bytes of the unint16_t.

Аргументы

val	Int16 to swap

Возвращает

Swapped value

8.1.1.20 test_bit()

bool test_bit (

uint32_t n,
int bit)

Tests if a bit is set.

Аргументы

n	Number to test in
bit	Number of the bit to test

Возвращает

True - bit is set. False - bit is cleared.

8.1.2. Переменные

8.1.2.1 mem_fd int mem_fd

Memory file descriptor.

Used to memory map Alwinner H3 processor registers.

8.1.2.2 pic16f628a const struct picmicro pic16f628a

Инициализатор

```
= {
    .device_id =                    PIC16F628A,
    .name =                        "pic16f628a",
    .program_memory_size =             0x800,
    .data_memory_size =               128,

    .program_cycle_time =             4000,
    .eeprom_program_cycle_time =      6000,
    .bulk_erase_cycle_time =          6000,

    .load_configuration_cmd =          0x00,
    .load_data_for_program_memory_cmd =  0x02,
    .load_data_for_data_memory_cmd =   0x03,
    .read_data_from_program_memory_cmd = 0x04,
    .read_data_from_data_memory_cmd =  0x05,
    .increment_address_cmd =           0x06,
    .begin_erase_programming_cycle_cmd = 0xFF,
    .begin_programming_only_cycle_cmd =  0x08,
    .bulk_erase_program_memory_cmd =   0x09,
    .bulk_erase_data_memory_cmd =      0x0B
}
```

Instance of picmicro structure.

Contains constants for PIC16f628a/627a/648a

8.2. Структура picmicro

Structure representing the various settings that are used in programming a specific model of the microcontroller.

Поля данных

- uint16_t **device_id**
- char **name** [16]
- size_t **program_memory_size**
- size_t **data_memory_size**
- int **program_cycle_time**
- int **eeprom_program_cycle_time**
- int **bulk_erase_cycle_time**
- uint8_t **load_configuration_cmd**
- uint8_t **load_data_for_program_memory_cmd**
- uint8_t **load_data_for_data_memory_cmd**
- uint8_t **read_data_from_program_memory_cmd**
- uint8_t **read_data_from_data_memory_cmd**
- uint8_t **increment_address_cmd**
- uint8_t **begin_erase_programming_cycle_cmd**
- uint8_t **begin_programming_only_cycle_cmd**
- uint8_t **bulk_erase_program_memory_cmd**
- uint8_t **bulk_erase_data_memory_cmd**

8.2.1. Подробное описание

Structure representing the various settings that are used in programming a specific model of the microcontroller.

Объявления и описания членов структуры находятся в файле:

- **rpp.c**

8.3. Структура picmemory

Structure for the pic memory map.

Поля данных

- uint16_t **program_memory_used_cells**
- uint16_t **program_memory_max_used_address**
- uint8_t **has_configuration_data**
- uint8_t **has_eeprom_data**
- uint16_t * **data**
- uint8_t * **filled**

8.3.1. Подробное описание

Structure for the pic memory map.

Объявления и описания членов структуры находятся в файле:

- **rpp.c**

Лист регистрации изменений

Изм.	Номера листов (страниц)				Всего листов (страниц) в докум.	№ докум.	Входящий № сопроводительного докум. и дата	Подпись	Дата
	измененных	замененных	новых	аннулированных					