

Инструкция по использованию
микропрограмм созданных на основе
Open Robotics Firmware Architecture
версии: 0.7.5

ревизия документа: 12:08fd2bb55829

Оглавление

1	Введение	3
2	Сборка микропрограммы	4
3	Режим serialgate	5
4	Режим подчиненного I²C	7
5	Драйвера устройств	8
5.1	Драйвер интроспекции	9
5.2	Драйвер портов Ввода/Вывода	9
5.3	Драйвера модельных сервоприводов	10
5.4	Драйвер управления моторами	11
5.5	Драйвер АЦП	11
5.6	Драйвер SPI	12
6	Примеры	13
6.1	Работа с драйвером интроспекции	13

Глава 1

Введение

Зачем нужна эта архитектура, какие задачи она решает:

- Единый формат обращения к любым устройствам низкого уровня, использованным в составе робота. В качестве такого формата принят протокол I²C. Это сделано для того, чтобы прозрачно интегрировать в систему уже имеющиеся устройства для шины I²C (компасы, гироскопы, акселерометры, УЗ-дальномеры и другие).
- Возможность быстрой сборки прошивки для каждого контроллера как в варианте OR-шлюз-контроллера, так и в варианте подчиненного контроллера на шине RoboBus[I²C].

Глава 2

Сборка микропрограммы

ORFA написана на языке Си под микроконтроллеры семейства AVR. Стандартные драйвера привязаны к модулям [Open Robotics](#), но Вы можете модифицировать микропрограмму под свои нужды, так как это ПО с открытыми исходными кодами, распространяемое по условиям лицензии MIT (см. файл LICENSE в дистрибутиве).

Для сборки вам потребуется компилятор AVR-GCC (GNU Compiler Collection), система сборки GNU Make и стандартные утилиты POSIX. Для OS Windows подойдет набор WinAVR.

Если вы хотите, чтобы ваши исправления были внесены в основной дистрибутив, то вам также потребуется система контроля версий Mercurial и немного навыков работы с ней (описание работы с этой системой выходит за рамки этого руководства).

Результат сборки зависит от файла конфигурации `local_config.mk`. Пример с описанием опций можно посмотреть в каталоге `doc` дистрибутива.

Основные настройки:

- PLATFORM – задает под какой модуль собирается программа (по умолчанию OR-AVR-M32-D)
- BAUD – задает скорость последовательного порта (по умолчанию автоопределение)

Сборка:

1. `$ cp ./doc/local_config.mk ./`
2. `$ edit ./local_config.mk`
3. `$ make`

Глава 3

Режим serialgate

В этом режиме используется библиотека serialgate, которая работает в качестве шлюза последовательный порт → I²C Master.

Начиная с версии ORFA 0.7.5 есть автоопределение скорости последовательного порта. Для его работы, после старта контроллера нужно подать 8 раз символ 0x0d (или '\r' в Си) до подачи каких-либо команд, после чего нужно сделать задержку не менее чем на 20 мс. После этого ORFA должна быть готова принимать команды. Последовательность символов 0x0d игнорируется парсером serialgate.

Обмен идет в текстовом режиме, каждая команда заканчивается символом перевода строки '\n' (LF) или '\r' (CR). Символ перевода строки без команды игнорируется, таким образом можно заканчивать команды последовательностью CR LF (будет разобрано как команда + пустая команда). Байты конвертируются в шестнадцатеричную форму. Учитывайте, что внутренний буфер может содержать не больше 128 символов, т.е. длина вашего запроса должна быть не более 128 символов. Это особенно важно при составлении мультискан запросов.

Таблица 3.1: Список команд

Название	Запрос	Ответ	Комментарий
Get protocol version	V	V1.1	
Clear I2C bus	X	X	
Set local address	L<addr>	L<addr>	addr – uint8
Set bus speed (freq)	C<freq>	C<freq>	freq – uint16
I ² C write	S<adr+w><data>P	SW(A)+P	A – Ack
I ² C read	S<adr+r><count>P	SR<adata>P	adata – uint8 array
Ошибка	–	ERROR <errtype>	errtype – uint8

В таблице нет мультискан-I²C запроса, регулярное выражение для

такого запроса выглядит так: (S<adr+w><data>|S<adr+r><count>)+P , а
ответ: (SW(A)+|SR<adata>)+P

В случае какой-либо ошибки при разборе или выполнении запроса
будет возвращен ее код.

Таблица 3.2: Коды ошибок

Код	Тип
1	Внутренняя ошибка парсера (на практике встречаться не должна)
2	Неизвестная команда
3	Не шестнадцатеричный символ в байте
4	Ошибочный данные
5	Потерян символ 'P' из I ² C запроса
6	Нет подтверждения адреса I ² C (NAck)
7	Нет подтверждения байта I ² C (NAck)

Глава 4

Режим подчиненного I²S

Глава 5

Драйвера устройств

Доступ к драйверам происходит через I²C регистры. Метод работы с регистрами похож на работу с I²C EEPROM микросхемами, с той лишь разницей, что здесь в общем случае нет автоинкремента адреса. Принятый порядок байт – **Big Endian** (также известный как Network Byte Order).

Процедура записи регистра: в фрейме записи I²C нужно сначала передать адрес регистра, а за тем данные. Пример запись в регистр 0x20 данных 0x001112 – S 10 20 00 11 12 P

Процедура чтения регистра: сначала нужно записать адрес регистра, а за тем прочитать N байт. Пример чтение из регистра 0x20 3 байта – S 10 20 S 11 03 P

Таблица 5.1: Стандартные драйвера

UID	Платформа	Драйвер
0x0000	Все	Драйвер интроспекции
0x0001	Все	Драйвер шины SPI
0x0020	OR-AVR-M32-D	Драйвер портов BB (RoboGPIO)
0x0021	OR-AVR-M128-S	Драйвер портов BB (RoboGPIO)
0x0030	OR-AVR-M128-S	Драйвер серв
0x0031	OR-AVR-M32-D	Драйвер серв
0x0040	Все	Драйвер АЦП (RoboGPIO)
0x0060	OR-AVR-M32-D	Драйвер моторов (RoboMD2)

5.1 Драйвер интроспекции

Этот драйвер предоставляет информацию обо всех включенных драйверах. Показывает идентификационный номер (UID), версию, стартовый адрес регистра и количество регистров.

За этим драйвером закреплен $UID = 0x0000$ и он всегда находится на регистре $0x00$.

Есть два вида запроса:

1. Получить количество всех драйверов (включая сам драйвер интроспекции)
Записать в регистр $0x00$ байт $0x00$, а потом прочесть 1 байт.
2. Получить информацию о драйвере №N
Записать в регистр $0x00$ байт N, а потом прочесть 6 байт.
Поддерживается автоинкремент номера драйвера. Например если записать номер 1, а потом прочесть 12 байт, а не 6, то вы получите данные о 1-ом и 2-ом драйвере. Но нужно обязательно учитывать размер буфера в 64 байт и количество драйверов. Попытка получить информацию о несуществующем драйвере приведет к ошибке, будет отдана информация о первом драйвере из списка. [См. пример чтения на псевдоязыке.](#)

Таблица 5.2: Структура информации о драйвере

Байт	Описание
1, 2	UID
3	Старшая часть версии
4	Младшая часть версии
5	Стартовый адрес регистра
6	Количество занятых регистров

5.2 Драйвер портов Ввода/Вывода

Драйвер позволяет управлять состоянием портов контроллера в режимах цифровой логической вход или выход.

Количество управляемых портов N: $N = 4$ для драйвера с $UID = 0x20$, $N = 2$ для драйвера с $UID = 0x21$.

Количество регистров, используемых драйвером равно $N \cdot 2$. Обозначим R – начальный регистр драйвера, а $R+n$ - $(R+n)$ -й регистр.

Первые N регистров, это чтение/запись значения, оставшиеся N регистров – запись режима порта. Каждой из 8 линий порта соответствует 1 бит. То есть если у контроллера 2 порта PORTA, PORTF, тогда для задания 2-му и 3-му линиям порта PORTF режима работы „цифровой вход“, а остальным битам этого порта режим „цифровой выход“ нужно в последний регистр драйвера R+3 отправить 0xF3 (т.е. 11110011 в двоичном представлении – 2-й и 3-й биты выставлены в 0). При работе в режиме входа запись в значение порта включает или выключает подтягивающие резисторы контроллера на соответствующих линиях.

5.3 Драйвера модельных сервоприводов

Предназначение – выработка ШИМ сигнала для управления модельными сервоприводами подключенными к портам RoboGPIO или RoboServo, в зависимости от контроллера. В этом драйвере всегда 2 регистра. Обозначим начальный регистр этого драйвера – RS, второй регистр SP. Регистр RS зарезервирован для обратной совместимости с драйвером версии 1.0.

Количество управляемых сервоприводов: 16 для драйвера с UID = 0x31, 32 для драйвера с UID = 0x30.

Модельные (RC) сервоприводы управляются прямоугольными сигналами частотой 50 Гц и скважность (временем, которое на линии высокое напряжение) от 1 до 2 мс. Поэтому не зная характеристик конкретного подключенного сервопривода часто приходится говорить о положении сервопривода, указывая лишь скважность управляющего сигнала. Например, 1200 мкс, это означает что если сервопривод поворачивается на угол $\pm 75^\circ$ при управляющих сигналах 1 мс и 2 мс соответственно, тогда 1200 мкс соответствует примерно:

$$-75^\circ + \frac{2 \cdot 75^\circ (1200 - 1000)}{2000 - 1000} = -45^\circ$$

ШИМ сигнал вырабатываемый драйвером имеет следующие характеристики:

- Период – 20 мс
- Скважность – от 500 до 2500 мкс

Для управления в регистр SP записывается N наборов [номер серво (1 байт)][скважность (2 байта)], но нужно учитывать, что размер буфера приема позволяет передать за раз только 16 положений.

Для включения режима серво RoboGPIO (UID = 0x0031) у версии драйвера 1.0 нужно было записать в регистр RS битовую маску, а сейчас достаточно установить скважность не равную 0-ю.

5.4 Драйвер управления моторами

UID = 0x0060

Содержит 4-е однобайтных регистра:

1. PWM1 – ШИМ 1-ого мотора (0 – 255)
2. PWM2 – ШИМ 2-ого мотора
3. DIR1 – направление 1-ого мотора (0/1 – вперед/назад)
4. DIR2 – направление 2-ого мотора

Есть возможность установить все регистры за один запрос, передав подряд PWM1, PWM2, DIR1, DIR2 в первый регистр. Например если записать в регистр PWM1 0x80600001, то мы установим PWM1 = 0x80, PWM2 = 0x60, DIR1 = 0, DIR2 = 1.

5.5 Драйвер АЦП

Драйвер позволяющий управлять блоком Аналого-Цифрового Преобразователя (далее АЦП).

Количество регистров, используемых драйвером - 2. Обозначим RC - первый регистр драйвера, RD - второй регистр.

Регистр RC - конфигурационный, в него пишутся 2 байта всегда:

1. Настройки АЦП
 - (a) биты 0..1
 - i. 00 – External reference
 - ii. 01 – AVCC
 - iii. 10, 11 – Internal reference
 - (b) бит 2 – режим точности АЦП
 - i. 0 – 8-битный
 - ii. 1 – 11-битный

2. Маска включаемых в АЦП каналов (8 бит – соотв. 8 линиям порта А)

Второй регистр для чтения результатов - из него читаются 8 байт, если режим 8-битный или 16 байт, если режим 16-битный. Можно читать меньше, но тогда будем знать состояние первых нескольких линий.

ВНИМАНИЕ! Чтение линий АЦП происходит в автоматическом циклическом режиме, поэтому после конфигурирования АЦП необходимо сделать паузу в 10 мс перед чтением значений.

5.6 Драйвер SPI

Глава 6

Примеры

Примеры даны на Си-подобном псевдоязыке.

6.1 Работа с драйвером интроспекции

```
/* introspection register always zero */
#define INTRO 0x00

/* Calculated for 128 char / 64 byte ORFA's cbf_t
 * · size of one pkg: 6 byte (12 chars)
 * · reply header and end (SWAASR ... P\n): 8 chars
 */
#define MAX_PER_PKG 10
#define PKG_LEN 6

drvlist orfa_get_drivers(int16_t address)
{
    drvlist head;
    orfareply reply;
    uint8_t count;
    vector buf = [INTRO, 0];

    /* get drivers count {{{ */
    orfa_tr_start();
    orfa_write(address, buf, 2);
    orfa_read(address, 1);
    orfa_tr_commit();
}
```

```

reply = orfa_read_reply(); /* SW */
reply = orfa_read_reply(); /* SR */

count = reply->data[0]
/* }}} */

uint8_t max_cnt = count / MAX_PER_PKG;
uint8_t min_cnt = count % MAX_PER_PKG;
uint8_t min_num = MAX_PER_PKG * max_cnt + 1;
uint8_t reply_cnt = max_cnt*2 + (min_cnt)? 2:0;

for (int i=0; i<max_cnt; i++) {
    buf[1] = MAX_PER_PKG * i + 1;

    orfa_tr_start();
    orfa_write(address, buf, 2);
    orfa_read(address, MAX_PER_PKG * PKG_LEN);
    orfa_tr_commit();
}

/* read last elements {{{ */
buf[1] = min_num;

orfa_tr_start();
orfa_write(address, buf, 2);
orfa_read(address, min_cnt * PKG_LEN);
orfa_tr_commit();
/* }}} */

for (int i=0; i < reply_cnt; i++) {
    reply = orfa_read_reply();

    if (reply->type == OR_READ) {
        continue;
    }
    if (reply->size % 6) {
        cerr << "get_drivers: wrong read reply size: " << reply...=>
...>size << endl;
        continue;
    }
}

```

```

    for (int pi=0; pi < reply->size; pi += 6) {
        head->append(
            uid = (reply->data[pi+0] << 8) | (reply->data[pi...    =>
...+1]),
            major_version = reply->data[pi+2],
            minor_version = reply->data[pi+3],
            start_addr = reply->data[pi+4],
            count = reply->data[pi+5]);
        }
    }

    return head;
}

```