Инструкция по использованию микропрограмм созданных на основе Open Robotics Firmware Architecture

версии: 0.8.0

ревизия документа: 19:b205cd99e03f

Оглавление

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 I^2C. Это сделано для того, чтобы прозрачно интегрировать в систему уже имеющиеся устройства для шины I^2C (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 (I^2C).

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. длинна вашего запроса должна быть не более 128 символов. Это особенно важно при составлении мультистарт 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
I²C write	`S<adr+w><data>P`	`SW(A)+P`	A – Ack
I²C read	`S<adr+r><count>P`	`SR<adata>P`	adata – uint8 array
Ошибка	–	`ERROR <errtype>`	errtype – uint8

В таблице нет мультистарт-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	Ошибочные данные
5	Потерян символ 'P' из I^2C запроса
6	Нет подтверждения адреса I^2C (NAck)
7	Нет подтверждения байта I^2C (NAck)

Глава 4

Режим подчиненного I^2C

Раздел в разработке

Глава 5

Адапторы устройств

Доступ к устройствам происходит через I^2C регистры адапторов [1]. Метод работы с регистрами похож на работу с I^2C 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 I^2C 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 адапторы

UID	**Платформа**	**Адаптер**
0x0000	Все	интроспекции
0x0001	Все	шины SPI
0x0020	OR-AVR-M32-D	портов BB (RoboGPIO)
0x0021	OR-AVR-M128-S OR-AVR-M128-DS	портов BB (RoboGPIO)
0x0030	OR-AVR-M128-S	серв
0x0031	OR-AVR-M32-D	серв
0x0032	OR-AVR-M128-DS	серв
0x0040	Все	АЦП (RoboGPIO)
0x0060	OR-AVR-M32-D	моторов (RoboMD2)

[1] В предыдущих версиях адапторы назывались драйверами т.к. реализация низкоуровневой работы была в них. Сейчас низкоуровневая работа вынесена в HAL

5.1 Адаптер интроспекции

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

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

Есть два вида запроса:

1. Получить количество записей
   Записать в регистр 0x00 байт 0x00, а потом прочитать 1 байт.

2. Получить информацию о адаптере №N
   Записать в регистр 0x00 байт N, а потом прочитать 6 байт.

   Поддерживается автоинкремент номера адаптера. Например если записать номер 1, а потом прочитать 12 байт, а не 6, то вы получите данные о 1-ом и 2-ом адаптере. Но нужно обязательно учитывать размер буфера в 64 байт и количество адаптеров.

   Попытка получить информацию о несуществующем адаптере приведет к ошибке, будет отдана информация о первом драйвере из списка.

   Чтение количества адаптеров **обязательно**! Так как во время этого запроса происходит пересчет адаптеров. Попытка получить информацию до чтения кол-ва приведет к ошибке – будут получены нули.

   См. пример чтения на псевдоязыке.

Таблица 5.2: Структура информации о адаптере

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

5.2 Адаптер портов Ввода/Вывода

Адаптер позволяет управлять состоянием портов контроллера в режимах цифровой логический вход или выход.

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

Количество регистров: $N \cdot 2$. Обозначим R – начальный регистр, 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 Адаптер модельных сервоприводов

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

Таблица 5.3: Количество управляемых сервоприводов

UID	**Количество**
0x0030	32
0x0031	16
0x0032	16

Модельные (RC) сервоприводы управляются прямоугольными сигналами частотой 50 Гц и скважностью (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 Адаптер 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 Адаптер АЦП

Позволяющий управлять блоком Аналого-Цифрового Преобразователя (далее АЦП).

Количество регистров: 2. Обозначим RC - первый регистр, RD - второй регистр.

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

1. Настройки АЦП
   (a) биты 0..1 – Источник опорного напряжения

i. 00 – Внешний (напряжение питание МК)
ii. 01 – AVCC (вход опоры, 3,3 В)
iii. 10, 11 – Внутренний (2,54 В)

(b) бит 2 – режим точности АЦП

i. 0 – 8-битный
ii. 1 – 10-битный

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

Второй регистр для чтения результата. В него нужно записать номер канала с которого начнется чтение. После считывания одного байта в 8-и битном, или 2-х – в 10-и битном режиме номер канала автоматически увеличивается (а после седьмого канала сбрасывается в ноль). Это позволяет прочитать сразу несколько линий идущих подряд.

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

См. пример чтения на псевдоязыке.

5.6 Адаптер SPI

Временно удален из дистрибутива т.к. не подерживался.
Ожидается возврат в версии 0.8.3

Глава 6

Уровень абстракции от аппаратного обеспечения (HAL)

Обеспечивает низкоуровневые операции. Предоставляет единый прогамм-ный интерфейс (API) для различных контроллеров Open Robotics.

В версии 0.8.0 HAL поддерживает платы:

1. OR-AVR-M32-D
2. OR-AVR-M128-D
3. OR-AVR-M128-DS

HAL также возможно использовать отдельно от ORFA. Для этого вам нужно:

1. добавить в ваш Makefile
   ```
   ORFA = ../orfa
   HAL = adc servo motor
   PLATFORM = OR_AVR_M128_DS
   CFLAGS += -DOR_AVR_M128_DS $(INCLUDE_DIRS)
   ```
2. подключить `hal/resolve.mk` в ваш Makefile
   ```
   include $(ORFA)/hal/resolve.mk
   ```

После чего в `$SRC` будут добавлены исходные файлы.
Смотри пример использования HAL'а а также пример в `orfa/doc/examples/hal-servo`

Глава 7

Примеры

Примеры даны на Си-подобном псевдоязыке. Также в качестве примера может быть использована библиотека liborfa (Си).

7.1 Работа с адаптером 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;
}
```

7.2 Работа с адаптером АЦП

```
enum adc_ref {
    REF_EXTERNAL = 0x0,
    REF_AVCC     = 0x1,
    REF_INTERNAL = 0x2
};

struct adc_conf {
    enum adc_ref ref; /* refrence */
    unsigned char precision; /* 8/10 */
    unsigned char enabled_pins; /* bit mask */
    int16_t address; /* device addr */
    uint8_t start_addr; /* adapter first register */
};

void orfa_adc_cfg(struct adc_conf *conf, enum adc_ref ref,
        unsigned char precision, unsigned char enabled_pins)
{
    uint8_t cfgmask = ref | (precision == 10)? 0x4 : 0x0;
    conf->ref = ref;
    conf->precision = precision;
    conf->enabled_pins = enabled_pins;

    vector buf = [conf->start_addr, cfgmask, enabled_pins];
```

```
    orfa_write(conf->address, buf, 3);
    reply = orfa_read_reply();
}

int orfa_adc_read_ch(struct adc_conf *conf, unsigned char channel)
{
    vector buf = [conf->start_addr + 1, channel];

    orfa_tr_start();
    orfa_write(conf->address, buf, 2);
    orfa_read(conf->address, (conf->precision == 10)? 2 : 1);
    orfa_tr_commit();

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

    int data = reply->data[0];
    if (conf->precision == 10) {
        data <<= 8;
        data |= reply->data[1];
    }

    return data;
}
```

7.3 Пример использования HAL отдельно от ORFA

```
# -*- makefile -*-
# HAL Test Makefile

target = hal-test

ORFA = ../orfa
PLATFORM = OR_AVR_M128_DS
HAL = servo
MCU_FLAGS = -mmcu=atmega128 -DF_CPU=7372800UL
```

```
CROSS_COMPILE_GCC = avr-
CROSS_COMPILE_BIN = avr-

CC = $(CROSS_COMPILE_GCC)gcc
LD = $(CROSS_COMPILE_BIN)ld
OBJCOPY = $(CROSS_COMPILE_BIN)objcopy
OBJDUMP = $(CROSS_COMPILE_BIN)objdump
SIZE = $(CROSS_COMPILE_BIN)size

INCLUDE_DIRS =

CFLAGS = -std=gnu99 -I${ORFA} $(INCLUDE_DIRS) $(MCU_FLAGS)
LDFLAGS = -Wl,--relax -Wl,--gc-sections

DEFINES = -D$(PLATFORM)

SRC = hal-test.c

OBJS = $(patsubst %.c,%.o,$(SRC))

all: $(target).hex
    $(SIZE) $(target).elf

$(target).hex: $(target).elf
    $(OBJCOPY) -j .text -j .data -O ihex $(target).elf $(target).hex
    chmod -x $(target).hex $(target).elf

$(target).elf: $(OBJS)
    $(CC) $(CFLAGS) -o $(target).elf $(OBJS)

ram_size: $(target).elf
    readelf -s $< | grep OBJECT | awk '{ SUM += $$3 } END { print SUM }'

%.o: %.c
    $(CC) $(DEFINES) $(CFLAGS) -c -o $@ $<

clean:
    rm -f $(OBJS) \
        $(target).hex $(target).elf

force: clean all
```

include ${ORFA}/hal/resolve.mk

```
/* HAL Test
 * (test servo)
 */

#include <stdint.h>
#include <stdio.h>

#include <util/delay.h>

#include "hal/servo.h"

/** Main
 */
int main(void)
{
    servo_init();
    asm volatile ("sei");
    for(;;) {
        servo_set_position(0, 1300);
        _delay_ms(250);
        _delay_ms(250);
        servo_set_position(0, 1700);
        _delay_ms(250);
        _delay_ms(250);
    }

    return 0;
}
```