Программное обеспечения для лаборатории организации ЭВМ

Тестовые драйверы и инструментальное обеспечение для работы с учебным стендом SDK-2.0

Научно-образовательное направления № 2
«Встроенные вычислительные системы»

Руководство пользователя

ОГЛАВЛЕНИЕ

Назначение

Программное обеспечение предназначено для обеспечения лабораторных, курсовых, и выпускных квалификационных работ проводимых в лаборатории организации ЭВМ научно-образовательного направления № 2 «Встроенные вычислительные системы». Целью создания программного обеспечения является инструментальная поддержка лабораторных, курсовых, выпускных работ, а также исследований в области встроенных вычислительных систем.

Состав проекта

Проект «Тестовые драйверы и инструментальное обеспечение для работы с учебным стендом SDK-2.0» состоит из двух частей. В каталоге EXAMPLE находятся исходные тексты драйверов для учебного стенда SDK-2.0.

Каталог	Описание
SIMPLE	Простейшая программа на языке С для стенда SDK-1.1
SERIAL	Пример драйвера последовательного канала
LED	Пример драйвера светодиодных индикаторов
DIN_DOUT	Пример драйвера дискретных портов ввода-вывода
DAC_ADC	Пример драйверов ЦАП-АЦП
EEPROM	Пример драйвера EEPROM
RTC	Пример драйвера RTC
CAN	Пример драйвера CAN 2.0
ETHERNET	Пример драйвера Ethernet
IEEE802.15.4	Пример драйвера IEEE 802.15.4

В каталоге HW_TEST находятся исходные тексты комплексного теста учебного стенда SDK-2.0, содержащего все перечисленные выше драйверы.

Инсталляция инструментальных средств

Требования к платформе разработки

Сборку тестов можно производить в среде Linux или Win32/cygwin.
Для сборки проекта необходимы следующие компоненты:

- компилятор GNUARM GCC v4.0.2 или старше для вашей платформы;
- инструментальная программа (G)M3P;
- пакет cygwin;
- IDE Eclipse и CDT;

Для архивации проекта вам понадобятся утилиты tar и gzip.

Порядок установки Eclipse

1. Установить виртуальную машину Java из каталога JavaRuntime.

- Win32 - jre-6u1-windows-i586-p.exe
- Linux - jre-6u1-linux-i586-rpm.bin

2. Установить Eclipse SDK.

- Win32 - eclipse-SDK-3.2.2-win32.zip
- Linux - eclipse-SDK-3.2.2-linux-gtk.tar.gz

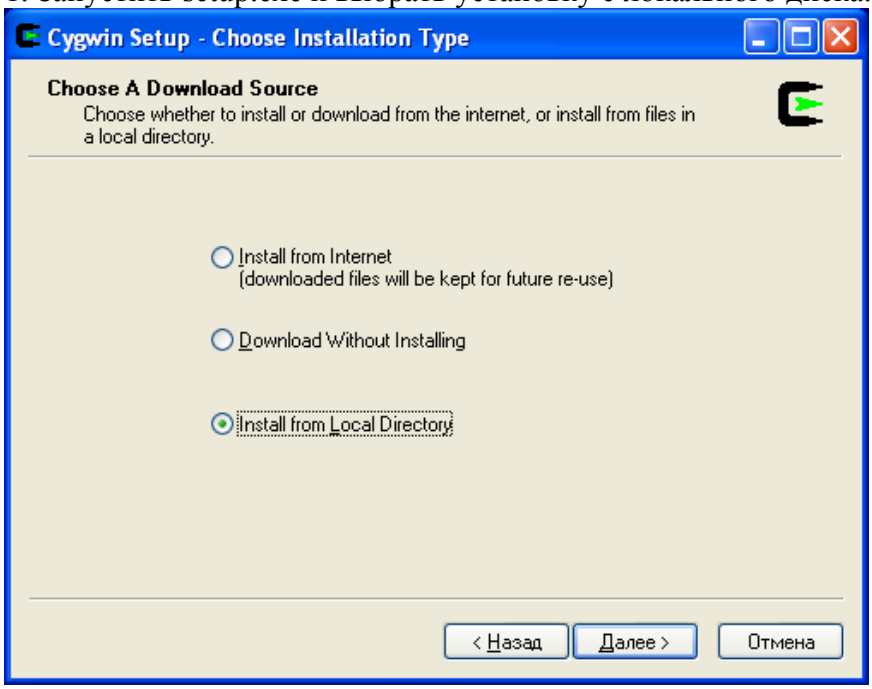
3. Установить CDT из каталога "Zylin embedded CDT". Перед установкой убедиться, что предыдущие дистрибутивы CDT удалены. Установка заключается в простом копировании содержимого каталогов в каталог Eclipse.

- Win32 - embeddedcdt-20070424.zip
  zylincdt-20070424.zip

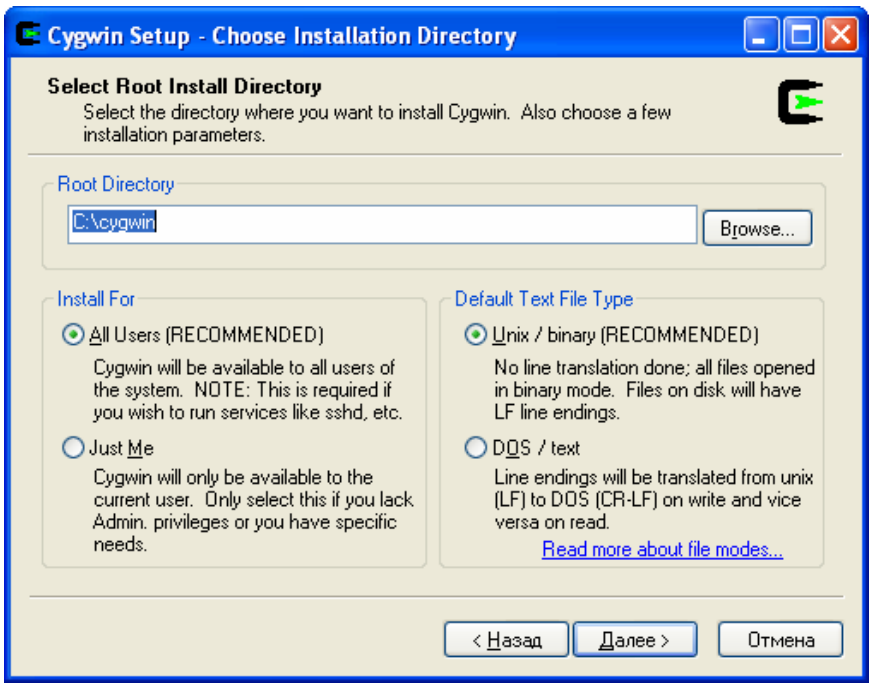
- Linux - embeddedcdt-linux-gtk-20070424.zip
  zylincdt-20070424.zip

Порядок установки Cygwin

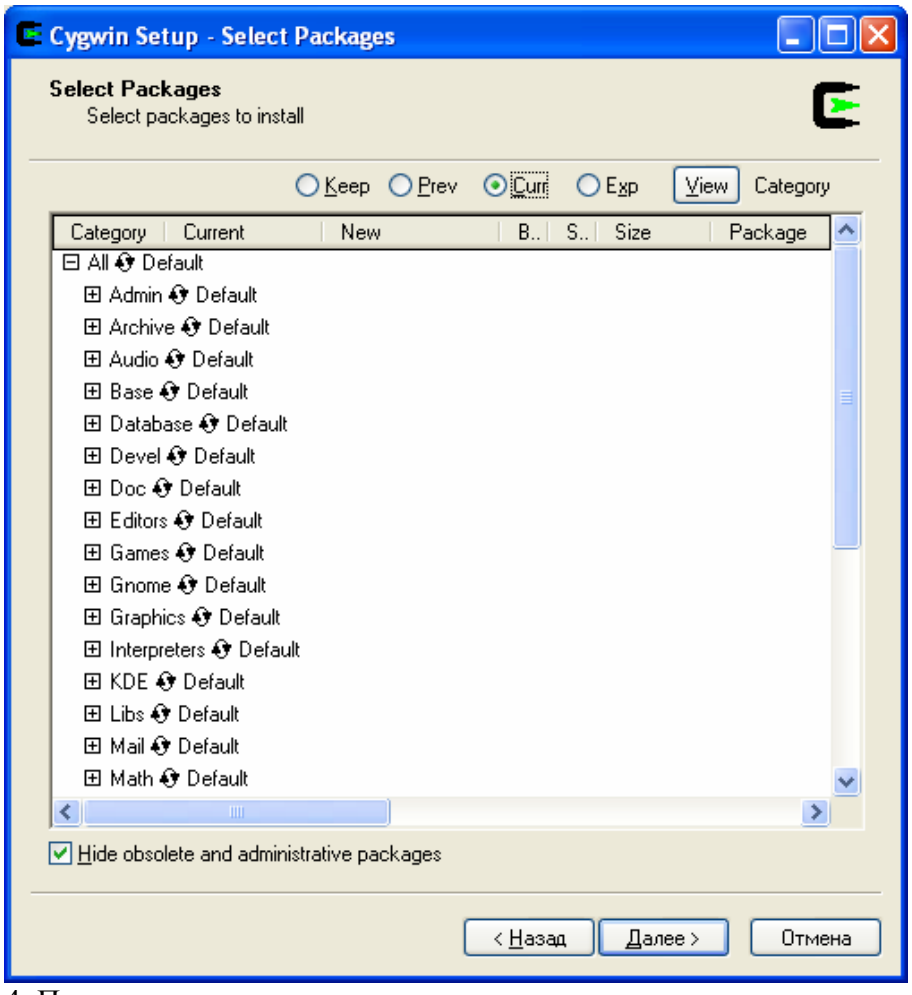
1. Запустить setup.exe и выбрать установку с локального диска.



2. Выбрать каталог для установки



3. Поставить метки Install напротив следующих групп:
   - Archive
   - Devel
   - Libs
   - Web



4. После установки поставить переменные окружения
   c:\cygwin\bin;c:\cygwin\usr\local\bin

Компиляция и загрузка тестов

Для сборки системы используйте утилиту GNU make.

make - сборка проекта
make clean - очистка проекта
make load - загрузка проекта в учебный стенд
make term - вызов эмулятора терминала
make dist - подготовка архива tar.gz

Использование инструментальных средств

Использование компилятора GCC

Ключи компилятора

Ключ	Описание
-I	Добавляет каталог 'директория' в начало списка каталогов, используемых для поиска заголовочных файлов. Ее можно использовать для подмены системных заголовочных файлов, подставляя ваши собственные версии, поскольку эти директории просматриваются до директорий системных заголовочных файлов. Если вы используете более чем одну опцию '-I', директории просматриваются в порядке слева направо; стандартные системные директории идут после.
-c	Компилировать или ассемблировать исходные файлы, но не линковать. Стадия ликовки просто не выполняется. Конечный вывод происходит в форме объектного файла для каждого исходного файла. По умолчанию, имя объектного файла делается из имени исходного файла заменой суффикса '.c', '.i', '.s', и.т.д. на '.o'. Нераспознанные входные файлы, не требующие компиляции или ассемблирования, игнорируются.
-fno-common	Размешает даже неинициализированные глобальные переменные в секции bss объектного файла, вместо генерации их в качестве общих блоков.Это имеет эффект в том, что если одна и та же переменныя объявлена (без extern) в двух различных единицах компиляции, вы получети ошибку при их линковке. Единственной причиной, по которой это может быть полезно, является проверка того, что программа будет работать на других системах, которые всегда работают таким образом.
-O0	Не оптимизировать.
-g	Добавляет явные глобальные обьявления. Это значит вставлять явные обьявления в начало каждого исходного файла для каждой функции, которая вызывается в нем, и не была обьявлена. Эти обьявления предшествуют первому описанию функции, которое содержит вызов к необъявленной функции. Эта опция применима только к protoize.

Ключи линкера

Ключ	Описание
-nostartfiles	Не использует стандартные системные файлы начального запуска

	при линковке. Стандартные системные библиотеки используются как обычно, если не указано `-nostdlib' или `-nodefaultlibs'.
-Wl, -Map=test_eth.map	Передает опцию -Map в качестве опции линкеру. Если 'опция' содержит запятые, она расщепляется запятыми на многочисленные опции. –Map – задает имя MAP файла.
-Tlinker.ld	Указывает на командный файл линкера.
-lc	Ищет при линковке библиотеку с именем 'библиотека'. Есть различие в том, где в комадной строке вы записываете эту опцию; линкер ищет обрабатываемые библиотеки и объектные файлы в порядке, в котором они указаны. Таким образом, `foo.o -lz bar.o' ищет библиотеку `z' после файла `foo.o', но перед `bar.o'. Если `bar.o' ссылается на функции в `z', эти функции не могут быть загружены. Линкер просматривает стандартный список каталогов в поиске библиотеки, который, фактически, является файлом с именем `libбиблиотека.a'. Затем линкер использует этот файл так, как будто бы он был точно специфицирован по имени. Директории, в которых ищет линкер, включают несколько стандартных системных каталогов, плюс любые каталоги, которые вы определяете с помощью `-L'. Обычно файлы, обнаруженные этим способом являются библиотечными файлами - архивными файлами, чьи элементы - объектные файлы. Линкер обрабатывает архивный файл, просматривая его в поиске элементов, которые определяют символы, на которые были ссылки, но которые до сих пор не определялись. Но, если оказывается, что обнаруженный файл - обычный объектный файл, то он линкуется в обычном порядке. Единственное различие между использованием опции `-l' и указанием имени файла в том, что `-l' добавляет к 'библиотеке' `lib' и `.a' и ищет в нескольких директориях.

Использование make

Утилита `make` автоматически определяет какие части большой программы должны быть перекомпилированы, и выполняет необходимые для этого действия.

Простой make-файл состоит из "правил" (rules) следующего вида:

```
цель ... : пререквизит ...
        команда
        ...
        ...
```

Обычно, **цель (target)** представляет собой имя файла, который генерируется в процессе работы утилиты `make`. Примером могут служить объектные и исполняемый файлы собираемой программы. Цель также может быть именем некоторого действия, которое нужно выполнить (например, `` `clean' `` - очистить).

Пререквизит (prerequisite) - это файл, который используется как исходдные данные для порождения цели. Очень часто цель зависит сразу от нескольких файлов.

Команда - это действие, выполняемое утилитой `make`. В правиле может содержаться несколько команд - каждая на свое собственной строке. **Важное замечание:** строки, содержащие команды обязательно должны начинаться с символа табуляции! Это - "грабли", на которые наступают многие начинающие пользователи.

Обычно, команды находятся в правилах с пререквизитами и служат для создания файла-цели, если какой-нибудь из пререквизитов был модефицирован. Однако, правило, имеющее команды, не обязательно должно иметь пререквизиты. Например, правило с целью `clean' ("очистка"), содержащее команды удаления, может не иметь пререквизитов.

Правило (rule) описывает, когда и каким образом следует обновлять файлы, указанные в нем в качестве цели. Для создания или обновления цели, `make` исполняет указанные в правиле команды, используя пререквизиты в качестве исходных данных. Правило также может описывать каким образом должно выполняться некоторое действие.

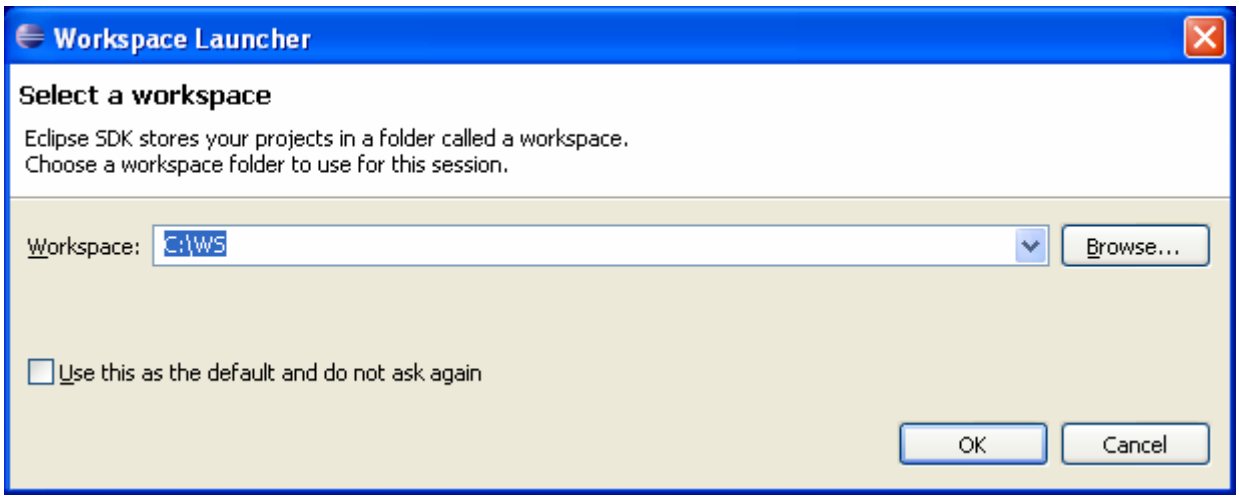
Пример makefile

```
# ------------------------------------------------------------------------------
# Имя проекта

NAME   = test_rtc

# Настройки компилятора и линкера
# LD      = arm-elf-ld -v


CC      = arm-elf-gcc
LD            = arm-elf-gcc
AR      = arm-elf-ar
AS      = arm-elf-as
CP      = arm-elf-objcopy
OD            = arm-elf-objdump

CFLAGS  = -I./INCLUDE -c -fno-common -O0 -g
AFLAGS  = -ahls -mapcs-32
LFLAGS  = -nostartfiles -Wl,-Map=test_rtc.map -Tlinker.ld -lc -lgcc
CPFLAGS = -O ihex
ODFLAGS= -x --syms

# Настройки системы автоинкремента версии сборки

PROJECT  = $(shell cat PROJECT)
VERSION  = $(shell cat VERSION)
BUILD    = $(shell cat BUILD)
TYPE     = $(shell cat TYPE)

PROJNAME = ${PROJECT}-${VERSION}-${BUILD}-${TYPE}
TARBALL  = ${PROJNAME}.tar

# Настройки M3P

M3P            = m3p
COMPORT  = com5
COMLOG   = $(COMPORT)_log.txt
BAUD     = 57600

# Каталоги с исходными текстами

SRC_DIR = SRC

# ------------------------------------------------------------------------------

all: test_rtc

clean:
```

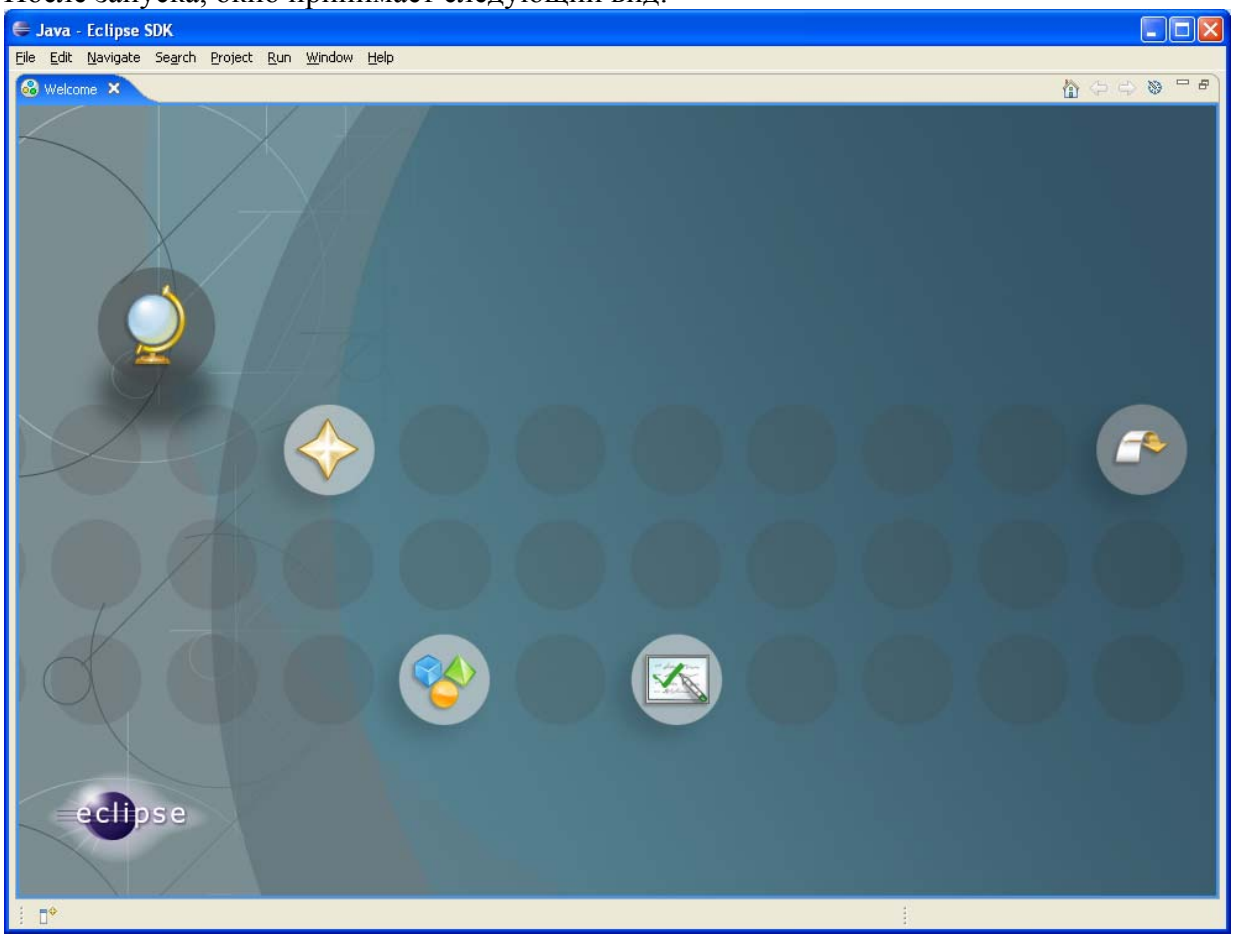
```
	-rm -f $(NAME).hex
	-rm -f $(NAME).dmp
	-rm -f $(NAME).map
	-rm -f $(NAME).out
	-rm -f pm3p_*.txt
	-rm -f com?_log.txt
	-rm -f $(TARBALL).gz
	-rm -f $(SRC_DIR)/*.asm
	-rm -f $(SRC_DIR)/*.o
	-rm -f $(SRC_DIR)/*.rst
	-rm -f $(SRC_DIR)/*.sym
	-rm -f $(SRC_DIR)/*.lst


term:
	$(M3P) echo $(COMLOG) $(BAUD)  openchannel $(COMPORT) +echo 6 term -echo bye

dist:
	tar cvf $(TARBALL) *
	gzip $(TARBALL)


PROJECT_SRC = \
$(SRC_DIR)/rtc.c \
$(SRC_DIR)/init.c \
$(SRC_DIR)/interrupt.c \
$(SRC_DIR)/serial.c \
$(SRC_DIR)/stdio_low.c \
$(SRC_DIR)/systimer.c \
$(SRC_DIR)/supervisor.c \
$(SRC_DIR)/crc8.c \
$(SRC_DIR)/i2c.c \
$(SRC_DIR)/wdt.c \
$(SRC_DIR)/test_rtc.c


PROJECT_OBJ = $(PROJECT_SRC:.c=.o)


test_rtc: test_rtc.out
	$(CP) $(CPFLAGS) test_rtc.out test_rtc.hex
	$(OD) $(ODFLAGS) test_rtc.out > test_rtc.dmp


test_rtc.out : $(PROJECT_OBJ) makefile linker.ld
	$(AS) $(AFLAGS) -o $(SRC_DIR)/crt.o $(SRC_DIR)/crt.s > $(SRC_DIR)/crt.lst
	$(LD) $(LFLAGS) $(SRC_DIR)/crt.o $(PROJECT_OBJ)   -o test_rtc.out


$(PROJECT_OBJ) : %.o : %.c makefile
	$(CC) $(CFLAGS) $< -o $@
```

Использование Eclipse

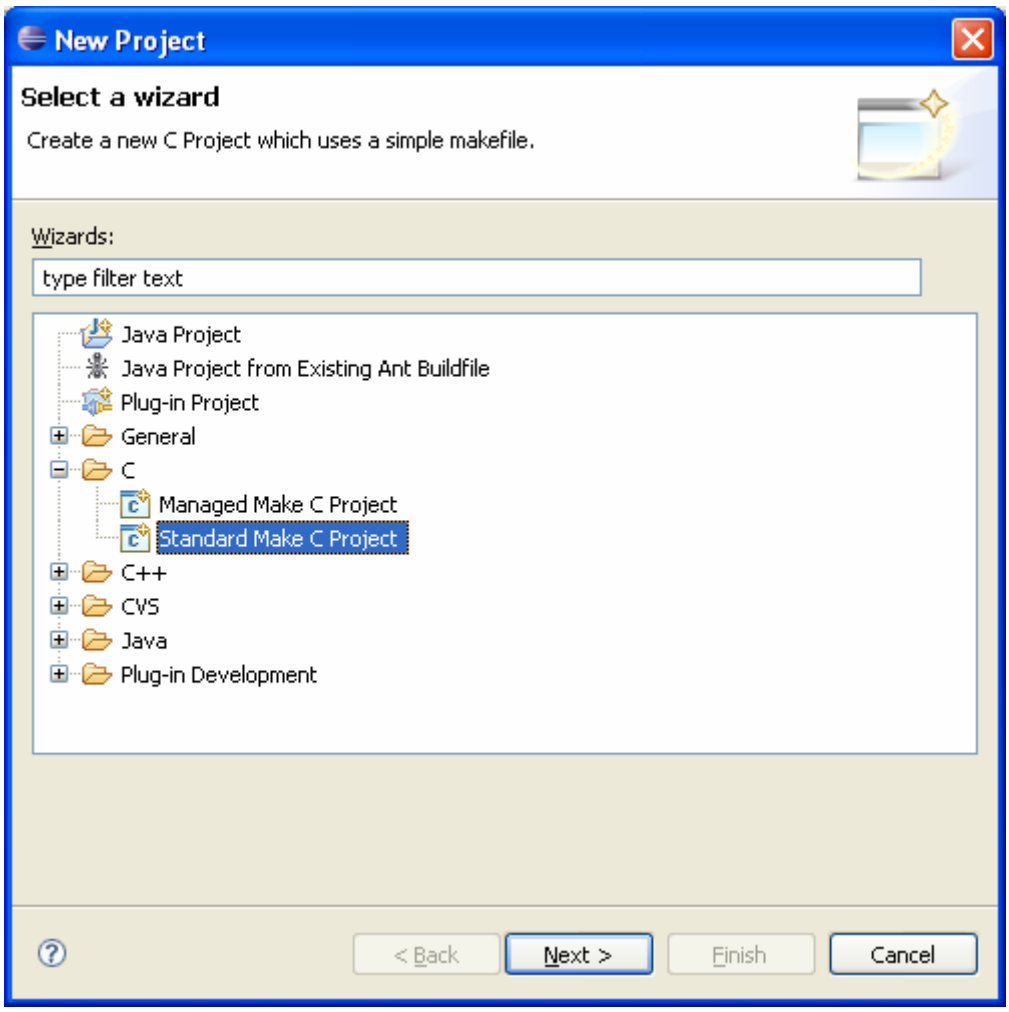
После запуска укажите расположение каталога, в котором будет хранится рабочее пространство Eclipse.



После запуска, окно принимает следующий вид:

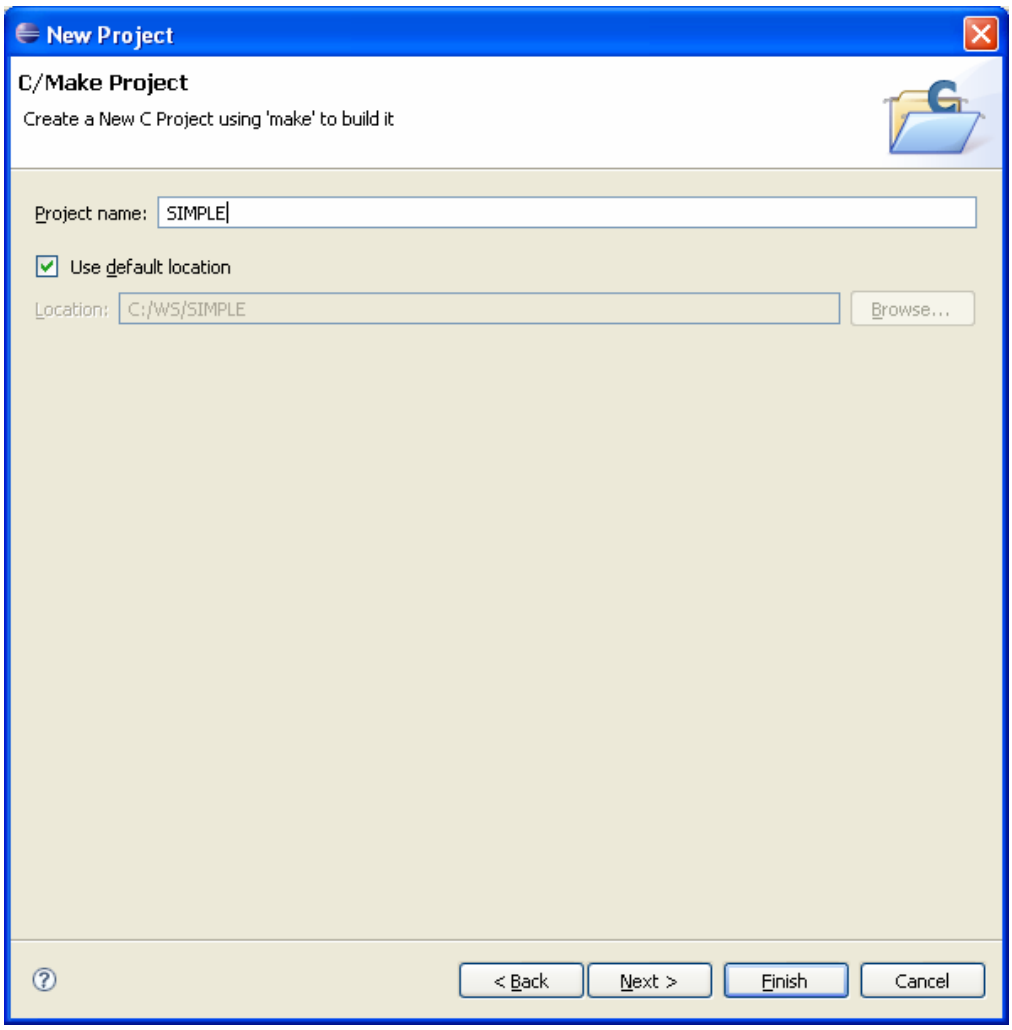


Создайте новый проект. Для этого выберите пункт меню File/New/Project

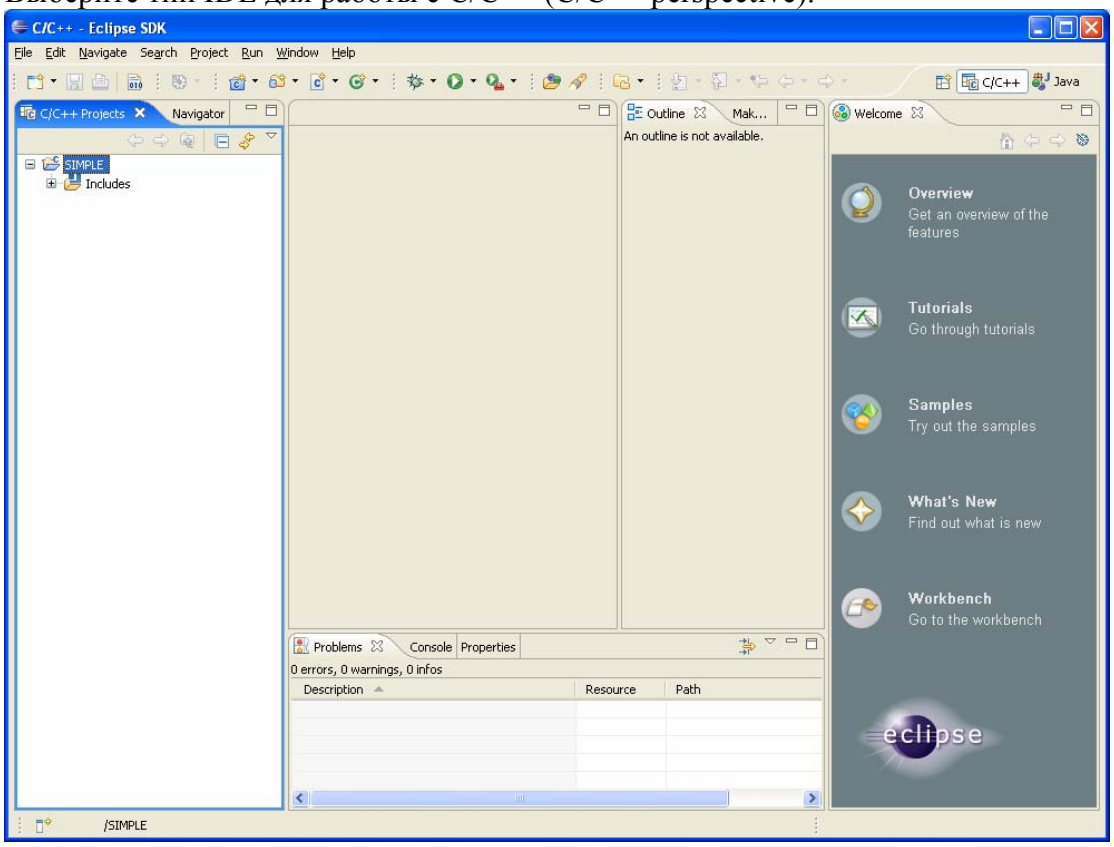


Выберите «Standard Make C Project».

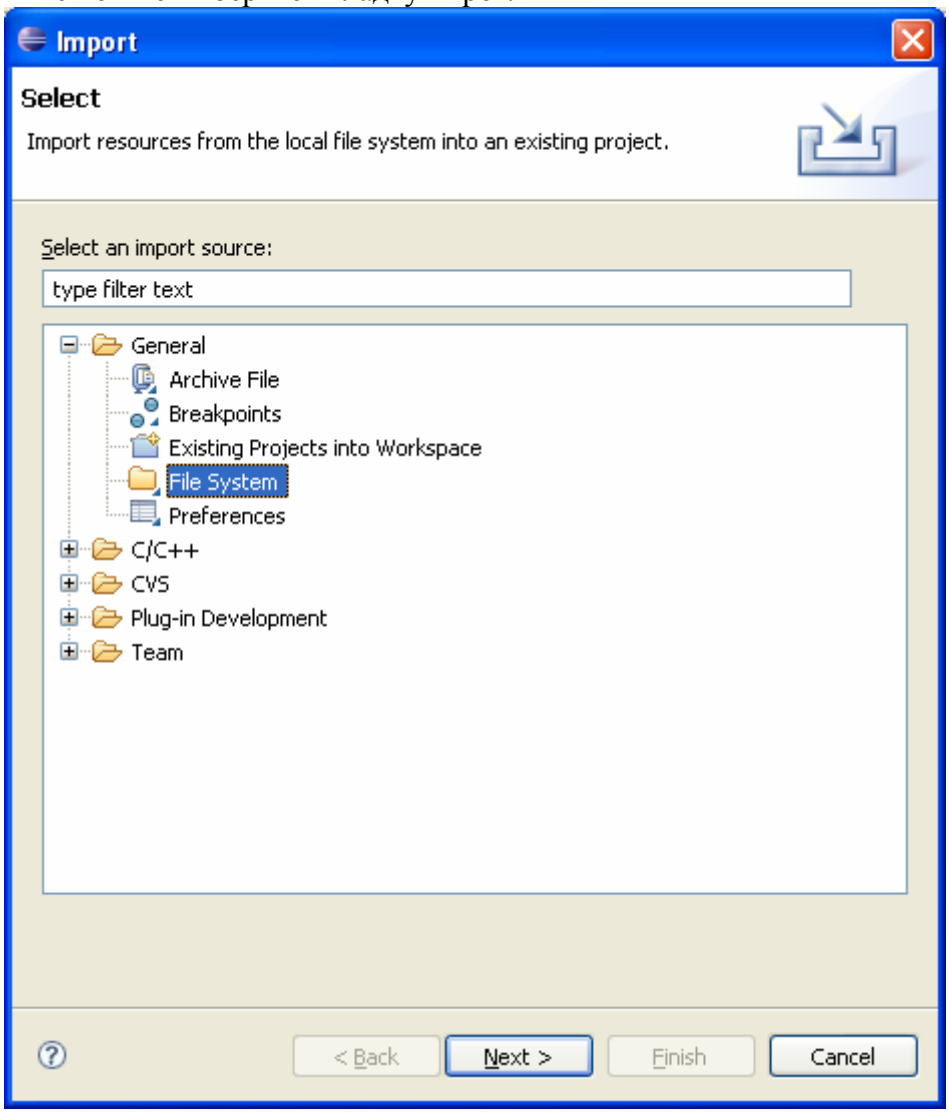
Укажите имя проекта (например, SIMPLE).



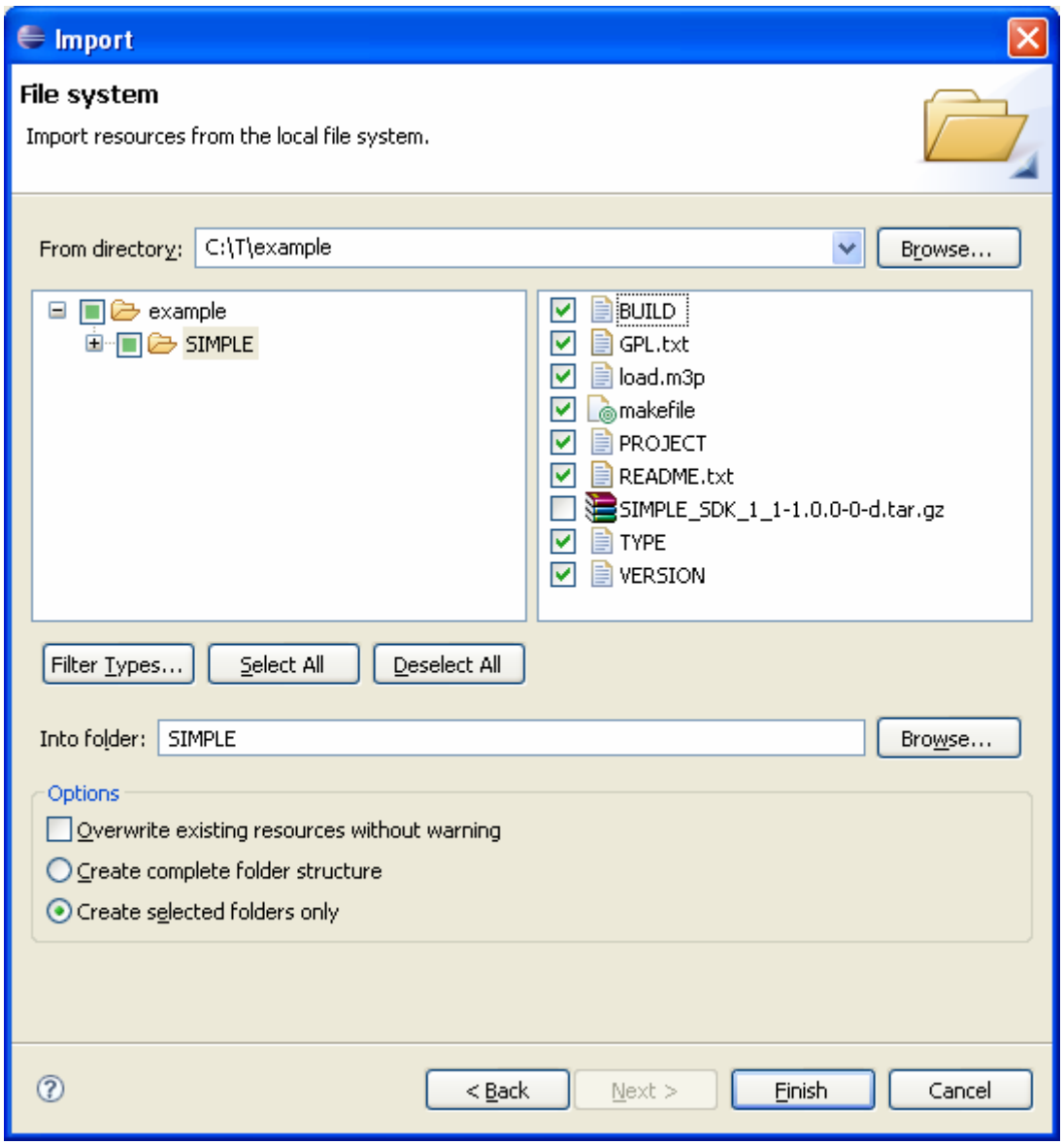
Выберите тип IDE для работы с C/C++ (C/C++ perspective).



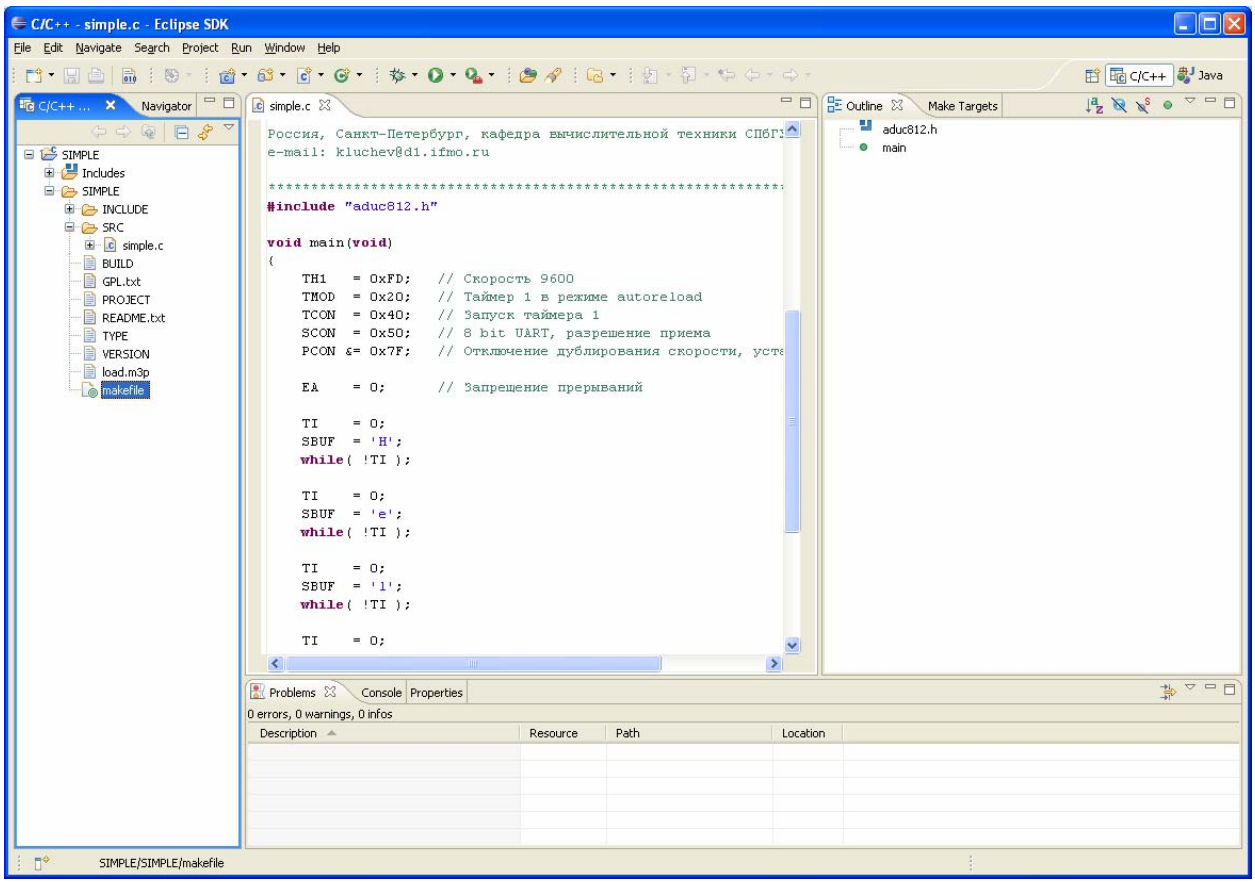
В меню File выберите вкладку import



Выберите файлы, которые собираетесь использовать в своем проекте.



После импорта проекта проект будет иметь следующий вид:



Программатор Flash для Philips LPC2000

Для доставки исполняемого модуля в стенд SDK-2.0 необходима специальная программа LPC2000 Flash Utility, поставляемая бесплатно фирмой Philips. Для загрузки модуля необходимо:

1. Установить перемычку ISP mode и перезапустить SDK-2.0;
2. Выбрать файл с вашим загрузочным модулем в формате HEX;
3. Выбрать тип микроконтроллера (LPC2292);
4. Нажать кнопку Read Device ID и убедиться в том, что связь со стендом работает;
5. Нажать кнопку Upload to Flash;
6. Дождаться окончания загрузки и снять перемычку ISP mode;
7. Перезапустить стенд SDK-2.0

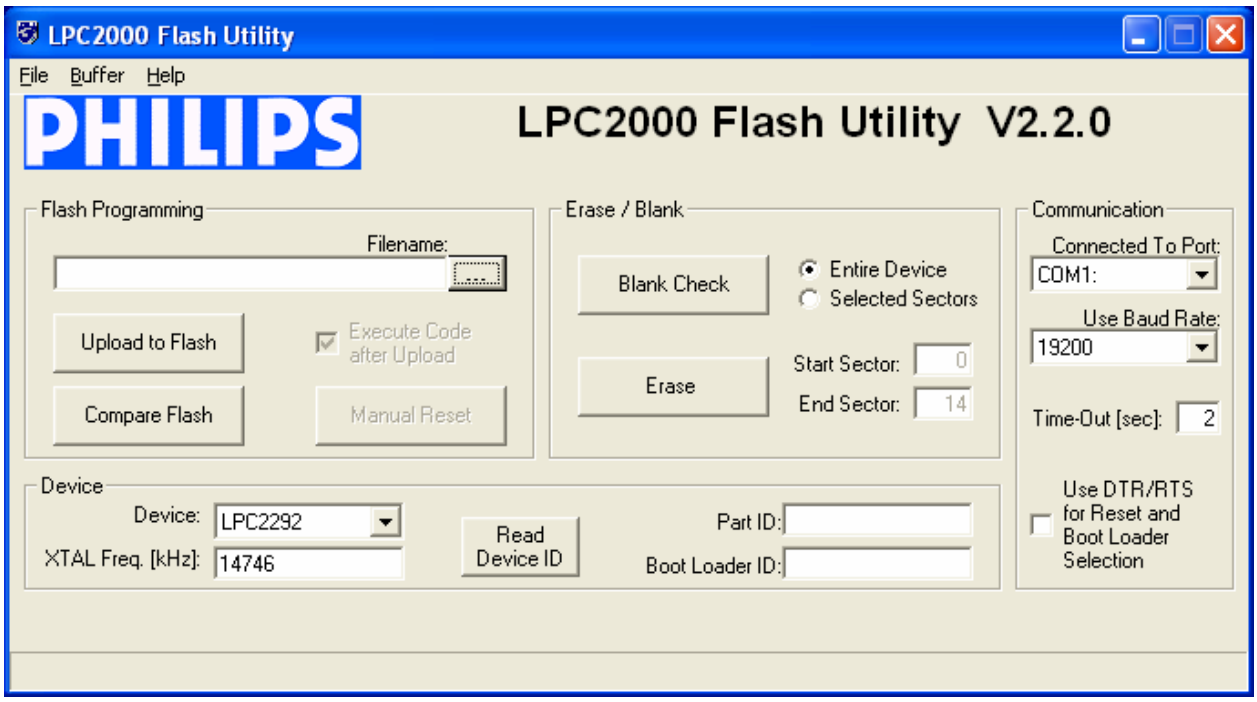


Рисунок 1 Внешний вид программы LPC2000 Flash Utility

Использование программы M3P для загрузки тестов

Инструментальная система M3P предназначена для решения следующего ряда задач:

1. Отладки, тестирование и внутрисистемного программирования встроенных систем;
2. Интеграции инструментальных средств в единую систему;
3. Связывания разнородных инструментальных средств посредством языка сценариев.

В качестве языка сценариев в M3P используется FORTH. В качестве базового стандарта языка использован стандарт FORTH83.

Протокол PM3P поддерживает модель взаимодействия MASTER-SLAVE, при этом роль подчиненного устройства отводится целевой системе. Канальный уровень PM3P обеспечивает надежную доставку данных. Так как PM3P сделан по принципу «ЗАПРОС-ОТВЕТ», каждому запросу сделанному мастером на канальном уровне соответствует ответ целевой системы. При отсутствии ответа, производится ряд перезапросов.
Прикладной уровень обеспечивает работу с так называемыми целями(target). С точки зрения прикладного уровня протокола, целевая система представлена в виде множества целей (не более 256). «Цель» можно трактовать как виртуальную машину с некоторым набором команд, поставляемых вместе с данными через прикладной уровень протокола обмена. Множество команд каждой из целей может отличаться друг от друга, но возможны и пересечения. Например, для целей типа ПАМЯТЬ (RAM, FLASH) естественными являются команды записи данных (write), чтения данных (read), а если память является частью машины Фон-Неймана, то естественной выглядит команда передачи управления (jmp) или сброса (reset). Для специфических видов памяти (например, таких как FLASH), может быть реализована команда стирания (erasepart).

В настоящей спецификации зафиксировано несколько основных целей (FLASH, RAM, LOOPBACK).

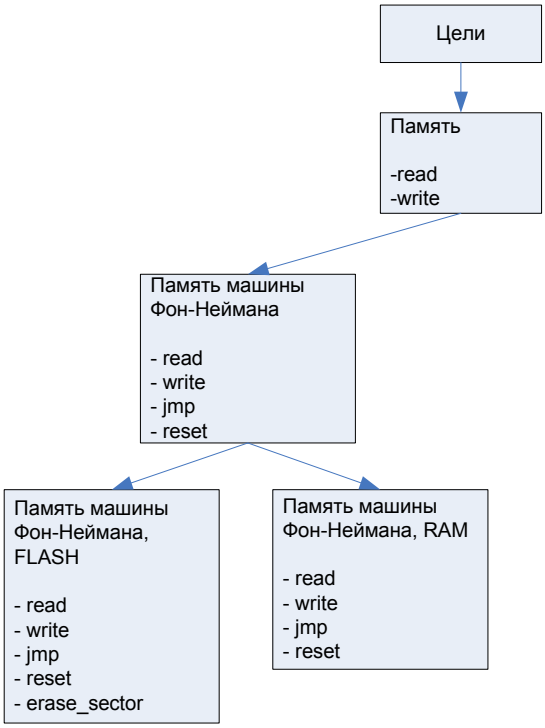


Рисунок 2 Иерархия целей

При необходимости, можно расширять и настраивать протокол для различных приложений. Можно предложить две основных стратегии развития PM3P:

- введение новых целей;
- использование адресной селекции, с выделением специальных областей адресного пространства имеющейся цели.

Первый вариант развития предполагает доработку спецификации и может привести к «взрыву» несовместимых версий реализаций. Этот путь развития необходимо использовать при работе с целями, радикально не совпадающими по своей идеологии построения с уже имеющимися в спецификации.

Второй вариант, хотя и не затрагивает спецификацию и системную часть реализации протокола, несколько менее удобен, из-за необходимости реализации адресной селекции на прикладном уровне в целевой системе.

Команда	Назначение
T_FLASH	Переключение системы для работы с FLASH-памятью.
T_RAM	Переключение системы для работы с RAM.
erasepart	Стирание сектора.
eraseall	Стирание всей микросхемы FLASH
write	Запись файла в цель.
Read	Чтение файла из цели.
Set_target	Установка номера цели (0.255)
Wait_m	Ожидание готовности резидентной части системы к работе.
Port_mode	Режим открытия последовательного порта. При port_mode = 2 происходит сброс APM3000r2 и блокировка COM0.

Пример скрипта для загрузки.

```
 terminateonerror
  9600 openchannel com1

: wait

    cr cr
    ." Включите питание и нажмите кнопку RESET на стенде SDK." cr cr
    ." Ожидание перезапуска... "

    begin rsio dup emit 109 == until

    ." Ok" cr cr
;

    wait

    T_RAM

    0x2100       write       hw_test.bin
    0x2100       jmp

    0 term

bye
```

Тестовые драйверы для учебного стенда SDK-2.0

Последовательный интерфейс

В LPC2292 находится два контроллера последовательного канала: UART0 и UART1. Основные возможности:

- Входные и выходные очереди на 16 байт.
- Совместимость с классическим контроллером последовательного канала 16550 на уровне регистров.
- Настраиваемый уровень контроля заполнения входной очереди 1,4,8 и14 байт.
- Встроенный генератор тактовой частоты.

UART0 и UART1 практически идентичны, за одним исключением: UART1 содержит набор модемных сигналов для организации аппаратного квитирования.

UxDLL, регистр делителя частоты, младшая часть

В регистры UxDLL и UxDLM записывается младшая и старшая часть делителя частоты, позволяющего задавать скорость обмена по последовательному каналу. Для доступа к регистру бит DLAB (Divisor Latch Access Bit) должен быть установлен в 1. Значение делителя частоты D высчитывается по следующей формуле:

D = Fosc/(16 * V), где Fosc = $12*10^6$ - тактовая частота
V – скорость обмена в битах в секунду.

UxDLM, регистр делителя частоты, старшая часть

Для доступа к регистру бит DLAB (Divisor Latch Access Bit) должен быть установлен в 1.

Регистр UxLCR, регистр управления линией

Биты	Функция	Описание
1:0	Количество битов данных	00 – 5 бит 01 – 6 бит 10 – 7 бит 11 – 8 бит
2	Количество стоп битов	0 – 1 стоп бит 1 – 2 стоп бита (1.5 бита, если количество битов данных 5 бит)
3	Контроль четности	0 – выключен 1 – включен
5:4	Тип контроля четности	00 – нечетный 01 - четный
6	Управление BREAK	0 – нет BREAK 1 – установить BREAK
7	DLAB	0 – доступ к делителю запрещен 1 – доступ к делителю разрешен[1]

UxLSR, регистр статуса

Биты	Функция	Описание
0	Данные готовы	1, если принят байт
1	Ошибка переполнения	1, если в приемный буфер пришел очередной байт, а старый байт еще не прочитан.
2	Ошибка четности	1, если принят байт с некорректным битом четности
3	Ошибка формата	1, если принят байт с некорректным форматом посылки
4	Обнаружен BREAK	1, если на линии состояние BREAK
5	Регистр хранения передатчика пуст	1, если регистр пуст
6	Регистр сдвига передатчика пуст	1, если регистр пуст
7	Ошибка входной очереди	Байт содержащийся в регистре данных принят с ошибкой (четности, формата или переполнения).

UxRBR, регистр данных, чтение

Регистр данных, чтение. Бит DLAB (Divisor Latch Access Bit) должен быть установлен в 0. Данный регистр доступен только для чтения.

UxTHR, регистр данных, запись

Регистр данных, запись. Бит DLAB (Divisor Latch Access Bit) должен быть установлен в 0. Данный регистр доступен только для записи.

[1] Бит DLAB понадобился, так как делитель частоты и буфер данных находятся по одному и тому же адресу

Пример программы

Инициализация

```
static void init_serial0( void )
{
//    PINSEL0 = 0x00000005; // RxD0, TxD0

    U0LCR   = 0x83;
    U0DLL   = 78;          // X = Fosc / (16 * 9600)
    U0DLM   = 0x00;
    U0LCR   = 0x03;        // DLAB = 0

}
```

Прием байта

```
unsigned char rsio0 (void)
{
    while ( ! ( U0LSR & 0x01 ) );

    return ( U0RBR );
}
```

Передача байта

```
void wsio0 (unsigned char ch)
{
    while ( ! ( U0LSR & 0x20 ) );

    U0THR = ch;
}
```

Системный супервизор, RTC и EEPROM

Системный супервизор (Processor Companion) фирмы Ramtron является многофункциональным устройством содержащим в себе следующие элементы:

- Схема сброса;
- Монитор питания;
- Часы реального времени;
- Счетчик событий;
- Энергонезависимую память FRAM;
- Сторожевой таймер;
- Уникальный серийный номер.

Системный супервизор взаимодействует с процессором через интерфейс I^2C. Устройство работает при питании от 2.7 до 5.5В Представители семейства FM31xxx отличаются размером энергонезависимой памяти: 4 кб, 16 кб, 64 кб и 256 кб. Быстрота записи и неограниченное число циклов записи делают возможным использование данной памяти как расширенного ОЗУ или для энергонезависимого хранения данных. Данная память обладает большей степенью энергонезависимости, чем память с батарейным резервированием питания.
Схема сброса предназначена для формирования качественного сигнала RESET после включения питания, после нажатия кнопки RESET (JP1) или после выключения питания. Проблема состоит в том, что при старте контроллера после включения питания или при

выключении питания возможны различные переходные процессы, могущие привести к некорректному исполнении программ или порче содержимого ОЗУ. Схема сброса обеспечивает формирование сигнала RESET на время, достаточное для окончания всех переходных процессов.
В устройстве есть возможность хранить 64-разрядный код (серийный номер) с возможностью блокировки записи для сохранения его неизменности.
Программируемый монитор питания позволяет вырабатывать сигнал RESET при понижения напряжения питания ниже установленного порога на 100 мс. Имеется флаг для индикации источника сброса.
Сторожевой таймер является счетчиком с программируемым периодом и сигналом сброса. При переполнении счетчика сторожевого таймера вырабатывается сигнал RESET. Интервал времени программируемый и может меняться в пределах от 100 мс до 3 сек.
Часы реального времени представляют информацию о времени и дате в двоично-десятичном формате. Часы могут непрерывно питаться от внешней батареи или конденсатора в качестве резервного источника. Часы используют кварцевый резонатор 32768 Гц и имеют режим калибровки для программной настройки точности хода.
Энергонезависимая память FRAM является перепрограммируемым электрически стираемым постоянным запоминающим устройством. Объем памяти FRAM, установленной в стенде SDK-2.0, составляет от 512 байт до 256 Кбайт. Количество циклов записи и чтения не ограничено, время хранения информации – до 10 лет.
Счетчик событий позволяет считать импульсы с помощью двух 16-ти разрядных энергонезависимых счетчиков событий, подсчитывающих число нарастающих или спадающих фронтов на соответствующих входах.

Основные возможности

Высокая степень интеграции для замены нескольких устройств
- Последовательная энергонезависимая память
- Часы реального времени (ЧРВ)
- Формирование сигнала сброса при понижении напряжения
- Сторожевой таймер
- Упреждающая сигнализация о нарушении питания/NMI
- Два 16-разрядных счетчика событий
- Последовательное число с блокировкой записи

Сегнетоэлектрическое энергонезависимое ОЗУ
- Версии различной емкости памяти: 4 кб, 16 кб, 64 кб и 256 кб
- Неограниченное количество циклов чтение/запись
- 10 летний срок хранения информации
- Запись без задержки (NoDelay™)

Часы-календарь реального времени
- Ток потребления от резервного источника до 1 мкА
- Представление реального времени в диапазоне от секунд до столетия в двоично-десятичном формате
- Учет високосности до 2099 г.
- Использует стандартный кварцевый резонатор 32768 Гц (6 пф)
- Программная калибровка
- Поддержка батареи или конденсатора в качестве резервного источника

Функции совместной работы с процессором системы
- Формирование сигнала сброса с активным низким уровнем при снижении уровня VDD и переполнении сторожевого таймера
- Программируемый порог VDD для формирования сброса

- Фильтрация и защита от дребезга ручного сброса
- Программируемый сторожевой таймер
- Два счетчика событий фиксируют системные вторжения или другие события
- Компаратор для простоты генерации упреждающего сигнала о снижении нестабилизированной точки питания -64-разр. программируемый последовательный код с защитой от записи

Быстродействующий двухпроводной последовательный интерфейс
- Максимальная тактовая частота последовательной шины до 1 МГц
- Поддержка тактовых частот предшествующих устройств 100 кГц и 400 кГц
- Выводы для задания адреса устройства позволяют использовать до 4 устройств на одной шине
- Управление ЧРВ, супервизором через двухпроводной интерфейс

Простота использования
- Работа при питании 2.7…5.5В
- Малогабаритный 14-выв. корпус SOIC
- Малый рабочий ток
- Рабочая температура -40°С…+85°С

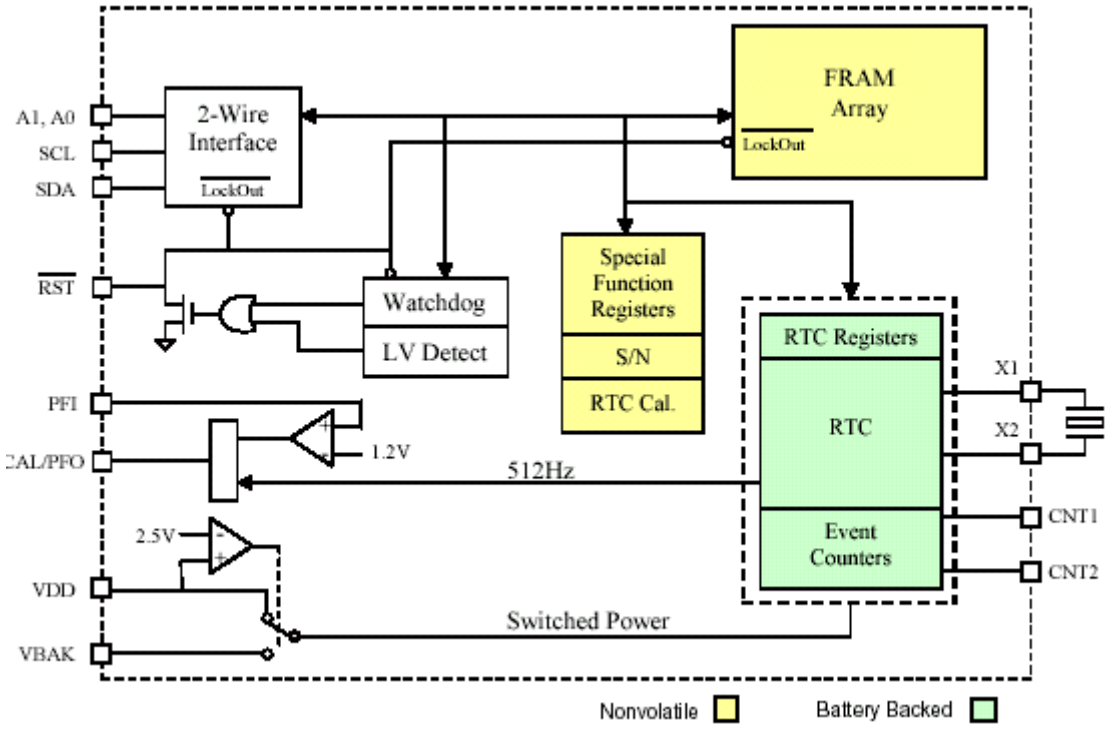


Рисунок 3 Структурная схема FM31xxx

Описание регистров

Все устройства FM31xxx доступны как набор 8-разрядных регистров.

Address	Data							
	D7	D6	D5	D4	D3	D2	D1	D0
18h	Serial Number Byte 7							
17h	Serial Number Byte 6							
16h	Serial Number Byte 5							
15h	Serial Number Byte 4							
14h	Serial Number Byte 3							
13h	Serial Number Byte 2							
12h	Serial Number Byte 1							
11h	Serial Number Byte 0							
10h	Counter 2 MSB							
0Fh	Counter 2 LSB							
0Eh	Counter 1 MSB							
0Dh	Counter 1 LSB							
0Ch					RC	CC	C2P	C1P
0Bh	SNL	-	-	WP1	WP0	VBC	VTP1	VTP0
0Ah	WDE	-	-	WDT4	WDT3	WDT2	WDT1	WDT0
09h	WTR	POR	LB	-	WR3	WR2	WR1	WR0
08h	10 years				years			
07h	0	0	0	10 mo	months			
06h	0	0	10 date		date			
05h	0	0	0	0	0	day		
04h	0	0	10 hours		hours			
03h	0	10 minutes			minutes			
02h	0	10 seconds			seconds			
01h	/OSCEN	reserved	CALS	CAL4	CAL3	CAL2	CAL1	CAL0
00h	reserved	CF	reserved	reserved	reserved	CAL	W	R

Рисунок 4 Карта памяти FM31xxx

Таблица 1 Значения по умолчанию

Адрес	Hex значение
18h	0x00
17h	0x00
16h	0x00
15h	0x00
14h	0x00
13h	0x00
12h	0x00
11h	0x00
0Bh	0x00
0Ah	0xlF
0lh	0x80

Описание выводов

Таблица 2 Описание выводов

Наимено-вание вывода	Описание
CNT1, CNT2	Входы счетчиков событий
A0, A1	Входы задания адреса устройства
CAL/PFO	Выход калибровки часов/предупреждения о нарушении питания
/RST	Вход/выход сброса
PFI	Вход компаратора упреждающего контроля понижения питания
X1, X2	Выводы подключения кварцевого резонатора
SDA	Ввод-вывод последовательных данных
SCL	Вход тактирования последовательной связи
VBAK	Резервное батарейное питание
VDD	Напряжение питания
VSS	Общий

Адреса I2C

С точки зрения интерфейса I2C, FM31xxx представлен как два устройства с различными адресами.

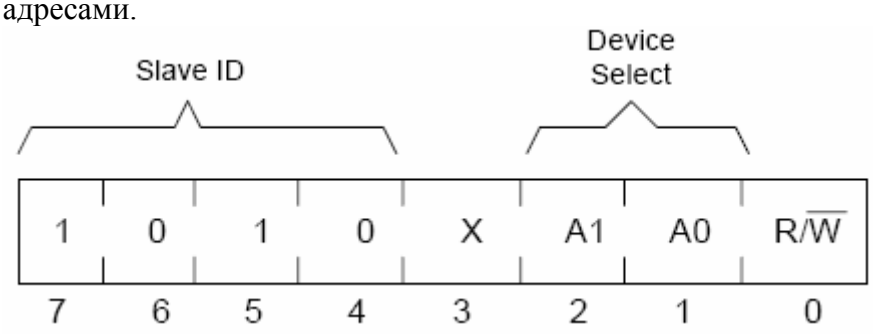


Рисунок 5 I2C адрес памяти энергонезависимой памяти

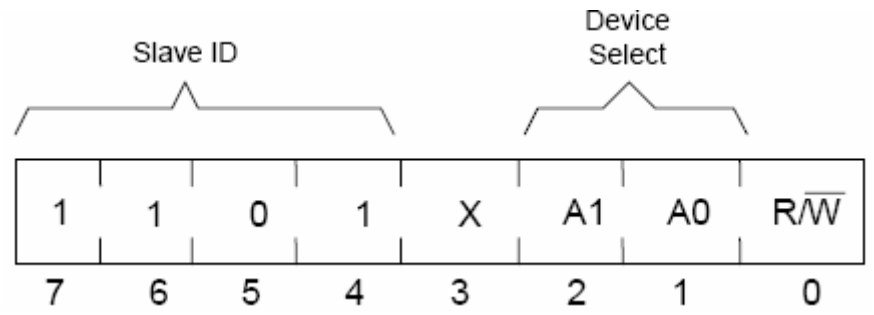


Рисунок 6 I2C адрес регистров

Взаимодействие с энергонезависимой памятью

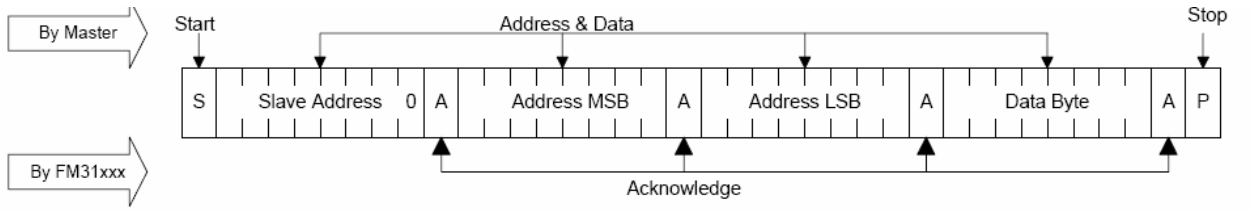


Рисунок 7 Запись одиночного байта в FRAM

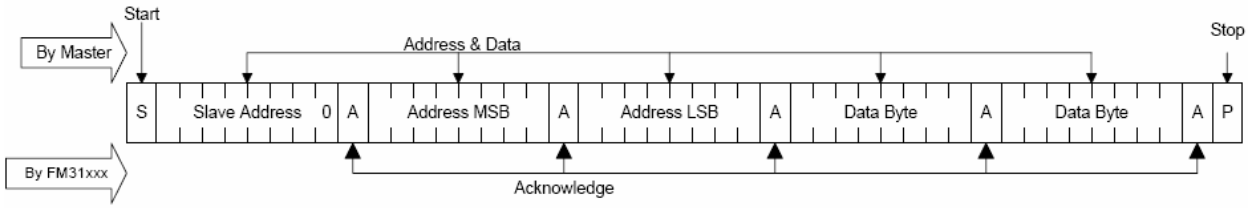


Рисунок 8 Запись нескольких байтов в FRAM

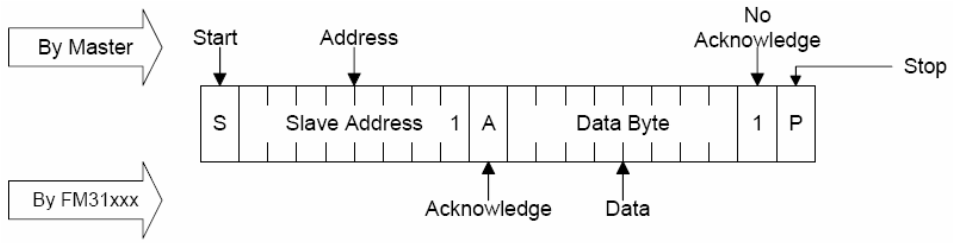


Рисунок 9 Чтение FRAM с текущего места

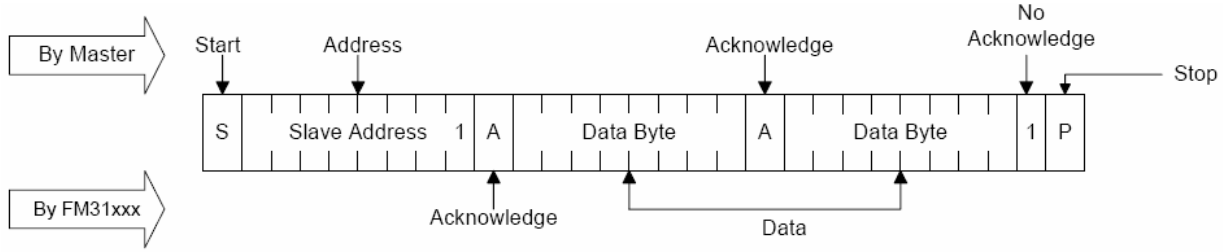


Рисунок 10 Чтение памяти с указанного адреса

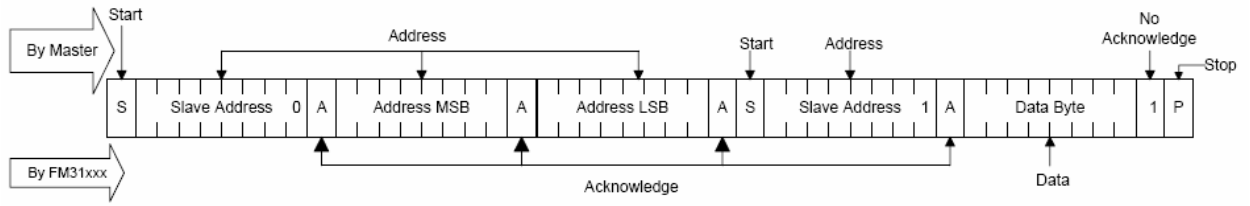


Рисунок 11 Случайный доступ к памяти

Тип		Первый байт адреса									Второй байт адреса						
FM31256		X	A14	A13	A12	A11	A10	A9	A8		A7		A5	A4	A3	A2	A1	A0
FM3164		X	X	X	A12	A11	Al0	A9	A8		A7		A5	A4	A3	A2	A1	A0
FM3116		X	X	X	X	X	Al0	A9	A8		A7		A5	A4	A3	A2	A1	A0
FM3104		X	X	X	X	X	X	X	A8		A7		A5	A4	A3	A2	A1	A0

Рисунок 12 Двухбайтовый адрес FRAM

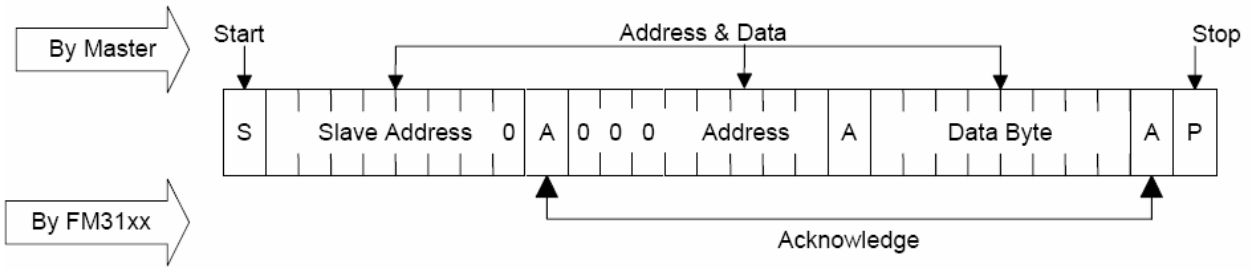


Рисунок 13 Запись байта в регистр

Драйвер мастера I2C для LPC2292

Инициализация

```
void init_i2c( void )
{
    I2CONCLR = 0xFF;


    I2SCLL=15;        //speed at 400 kHz for a VPB Clock Divider = 4 (12 MHz)
    I2SCLH=15;
    I2CONSET = 0x40;  //Active Master Mode on I2C bus
}
```

Установка адреса

```
int send_slave_addr(unsigned char addr)
{
    if( ! ( addr & 0x01 ) ) //test if it's a master adress
    {
        I2CONSET = STA;

        while( I2STAT != 0x08 );    //Set and wait the start

        I2DAT    = addr; // Charge slave Address
        I2CONCLR = SIC | STAC; // Clear i2c interrupt bit to send the data

        while( I2STAT != 0x18 && !( I2CONSET & SI ) );    //wait the ACK
    }
    else            //it's a slave adress
    {
        I2CONSET = STA;
        I2CONCLR = SIC;

        while( I2STAT != 0x10 && !( I2CONSET & SI ) );
        //Set and wait the start

        I2DAT    = addr; // Charge slave Address
        I2CONCLR = SIC | STAC; // Clear i2c interrupt bit to send the data

        while( I2STAT != 0x40 && !( I2CONSET & SI));    //wait the ACK
    }

    return OK;
}
```

Прием байта

```
int read_i2c( unsigned char * data)
{
int res = OK;

    I2CONCLR = SIC;                          //ACK from the master
    while( I2STAT != 0x50 && !( I2CONSET & SI ) ); //wait the data

    // Read the data...
    *data = I2DAT;
    return res;
}
```

Передача байта

```
int write_i2c(unsigned char data)
{
    I2DAT    = data; // Charge Data
    I2CONCLR = SIC; // Clear i2c interrupt bit to send the data
```

```
    while( I2STAT !=0x28 && !( I2CONSET & SI ) );    //wait the ACK

    return OK;
}
```

Формирование стоп состояния

```
int stop_i2c( void )
{
volatile unsigned long i;

    I2CONCLR = SIC;
    I2CONSET = STO;
    while( ( I2CONSET & STO ) );
    for( i = 0; i < 10000; i++);
    return OK;
}
```

АЦП микроконтроллера LPC2292

Основные характеристики

- 10-битное последовательное преобразование
- 8 мультиплексируемых каналов
- Режим энергосбережения
- Диапазон измерений от 0 до 3В
- Время 10-битного преобразования >= 2.44 мкс
- Burst-режим преобразования для одного или нескольких каналов

Основным источником тактовых импульсов АЦП является линия VPB. АЦП имеет программно-управляемый делитель частоты (максимальная рабочая частота АЦП – 4,5МГц). Преобразование с максимальной точностью выполняется за 11 тактов.

Управляющий регистр АЦП (ADCR)

Биты	Название	Описание
7:0	SEL	Выбор оцифровываемых аналоговых входов. В программно-управляемом режиме может быть выбран только один вход. В аппаратном режиме может быть выбрано любое количество входов. Нулевое значение этих битов соответствует значению 0x01.
15:8	CLKDIV	Частота VPB (PCLK) делится на (это значение + 1) чтобы получить частоту работы АЦП, которая должна быть меньше или равна 4,5МГц.
16	BURST	Если этот бит, равен нулю, производится программно-управляемое преобразование, которое требует 11 тактов. Если этот бит, равен единице, АЦП производит периодическое преобразование со скоростью указанной в CLKS.
19:17	CLKS	Количество тактов, затрачиваемое на преобразование в Burst-режиме, и количество точных битов результата. 000 = 11 тактов/10 бит, 001 = 10 тактов/9 бит, …, 111 = 4 такта/3 бита.
21	PDN	1: АЦП включено 0: АЦП выключено
23:22	TEST 1:0	Эти биты используются для тестирования устройства. (00 =

		нормальный режим работы)
26:24	START	Если бит BURST равен нулю, эти биты указывают событие, по которому начинается преобразование. 000: не начинать преобразование 001: начать преобразование немедленно 010: начать преобразование по перепаду указанному в бите EDGE на линии P0.16/EINT0/MAT0.2/CAP0.2 011: начать преобразование по перепаду указанному в бите EDGE на линии P0.22/TD3/CAP0.0/MAT0.0 100: начать преобразование по перепаду указанному в бите EDGE на линии MAT0.1 101: начать преобразование по перепаду указанному в бите EDGE на линии MAT0.3 110: начать преобразование по перепаду указанному в бите EDGE на линии MAT1.0 111: начать преобразование по перепаду указанному в бите EDGE на линии MAT1.1
27	EDGE	Эти биты используются, только если в поле START находится значение из интервала 010-111. 0: начало преобразования по отрицательному перепаду выбранного CAP/MAT сигнала 1: начало преобразования по положительному перепаду выбранного CAP/MAT сигнала

Регистр данных АЦП (ADDR)

Биты	Название	Описание
5:0		Эти биты всегда читаются как ноль. Они предназначены для совместимости с высокоточными АЦП в будущем.
15:6	V/V_{3A}	Когда бит DONE установлен в единицу, это поле содержит двоичное значение, представляющее напряжение на линии Ain выбранной полем SEL, деленное на напряжение на линии VddA. Ноль в этом поле показывает, что напряжение на линии Ain меньше, равно или близко к V_{SSA}. Значение 0x3FF указывает, что напряжение на линии Ain больше, равно или близко к V_{3A}.
23:16		Эти биты всегда читаются как ноль.
26:24	CHN	Эти биты содержат номер канала, из которого было проведено преобразование.
29:27		Эти биты всегда читаются как ноль. В будущем они могут использоваться для расширения поля CHN для совместимости с АЦП с большим количеством каналов.
30	OVERUN	Этот бит устанавливается в единицу в Burst-режиме, если результаты одного или боле преобразований были потеряны и перезаписаны до того, как преобразование сформировало результат в младших битах.
31	DONE	Этот бит устанавливается в единицу, когда АЦП заканчивает преобразование. Бит очищается, когда считывается этот регистр (ADDR) или производится запись в регистр ADCR. Если запись в регистр ADCR произошла во время выполнения преобразования, этот бит устанавливается и начинается новое преобразование.

Прерывания АЦП

Когда бит DONE устанавливается в единицу, в VIC (Vectored Interrupt Controller) поступает сигнал прерывания от АЦП. В программе можно настроить бит "разрешения прерывания" в VIC. Бит DONE сбрасывается после чтения регистра ADDR.

Пример драйвера АЦП

```
/*---------------------------------------------------------------------------
Инициализация АЦП
---------------------------------------------------------------------------*/
void init_adc()
{
	ADCR = 0x01010300;
}

/*---------------------------------------------------------------------------
Включить АЦП
channel - канал (от 0 до 7)
---------------------------------------------------------------------------*/
void start_adc( unsigned char channel )
{
	if ( channel > 7 ) channel = 0;
	ADCR &= ~0x000000FF;
	ADCR |= ( 1 << channel );
	ADCR |= 0x00200000; //бит PDN = 1
}

/*---------------------------------------------------------------------------
Выключить АЦП
---------------------------------------------------------------------------*/
void stop_adc( void )
{
	ADCR &= 0xFFDFFFFF; //бит PDN = 0
}

/*---------------------------------------------------------------------------
Функция чтения преобразованного значения АЦП
---------------------------------------------------------------------------*/
unsigned short read_adc( void )
{
	unsigned long reg;

	while ( !( ( reg = ADDR ) & 0x80000000 ) );

	return ( unsigned short )( ( reg & 0x0000FFFF ) >> 6 );
}
```

Пример драйвера ЦАП

```
/*-----------------------------------------------------------------------------
Инициализация PWM
-----------------------------------------------------------------------------*/
void init_pwm( void )
{
        //P0.07 = PWM2
        PINSEL0 &= 0xFFFFBFFF;
        PINSEL0 |= 0x00008000;
        //P0.07 = Output
        IO0DIR |= 0x00000080;
        //P0.21 = PWM5
        PINSEL1 &= 0xFFFFF7FF;
        PINSEL1 |= 0x00000400;
        //P0.21 = Output
        IO0DIR |= 0x00200000;

        PWMTCR = 0x00000000;
        PWMTC  = 0x00000000;
        PWMPC  = 0x00000000;
        PWMPR  = 0x00000000;
        PWMMR0 = PWM_MAX;
        PWMLER = 0x00000001;
        //Reset on PWMMR0
        PWMMCR = 0x00000002;
        PWMTCR = 0x00000008;
}

/*-----------------------------------------------------------------------------
Включение генерации PWM
-----------------------------------------------------------------------------*/
void start_pwm( void )
{
        PWMTCR |= 0x00000001;
}

/*-----------------------------------------------------------------------------
Выключение генерации PWM
-----------------------------------------------------------------------------*/
void stop_pwm( void )
{
        PWMTCR &= ~0x00000001;
}

/*-----------------------------------------------------------------------------
Установка значения PWM
-----------------------------------------------------------------------------*/
void set_pwm( unsigned char channel, unsigned long val )
{
        switch ( channel )
        {
        case 2:
                //Enable PWMx output
                PWMPCR = 0x00000400;
                PWMMR2 = val;
                PWMLER = 0x00000004;
                break;
        case 5:
                //Enable PWMx output
                PWMPCR = 0x00002000;
                PWMMR5 = val;
                PWMLER = 0x00000020;
                break;
        default:
                break;
        }
}
```

Контроллер Ethernet LAN91C111

Основные характеристики LAN91C111:

- Две скорости обмена – 10/100 Мбит/сек.
- Совместимость с IEEE 802.3/802.3u - 100BASE-TX/10BASE-T
- Автоопределение скорости 10/100 и типа связи: полный дуплекс/полудуплекс.
- Не требует внешних фильтров.
- Поддержка работы в режиме полного дуплекса.
- 8 Кбайт встроенной памяти для организации входной и выходной очереди пакетов.
- Расширенные возможности по управлению энергопотреблением.
- Возможность подключения внешнего конфигурационного EEPROM с последовательным интерфейсом.
- 8, 16 и 32 разрядный режим работы с системной шиной.
- Внутренний 32 разрядный интерфейс к буферу пакетов.

Структурная схема

Внутри, контроллер LAN91C111 состоит из двух блоков. Блок Ethernet MAC отвечает за прием и формирование пакетов Ethernet. Блок PHY отвечает за формирование сигналов на линиях физического уровня интерфейса Ethernet. Блок PHY соединятся с блоком Ethernet MAC через интерфейс IEEE 802.3 MII (Media Independed Interface).

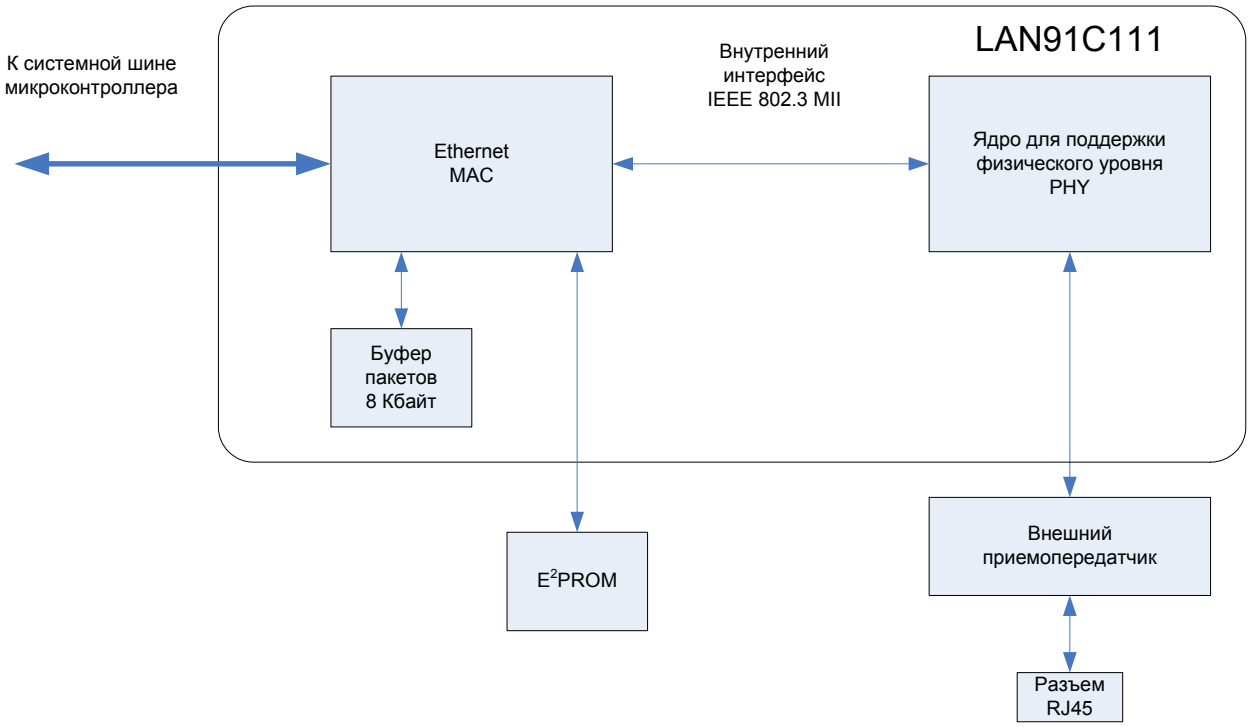


Рисунок 14 Структура контроллера Ethernet LAN91C111

Для увеличения производительности контроллера, к блоку MAC подключен буфер пакетов размеров 8 Кбайт. Увеличение производительности достигается за счет того, что внутренняя шина данных 32 разрядная, а внешний интерфейс для связи с микроконтроллером может быть как 32 разрядным, так и 8 или 16 разрядным. Опционально, к контроллеру Ethernet может подключаться энергонезависимая память для хранения различных настроек, таких например как MAC адрес контроллера. К блоку PHY подключается внешний приемопередатчик для обеспечения подключения к шине Ethernet.

Доступ к регистрам физического уровня

Доступ к блоку PHY осуществляется посредством интерфейса IEEE 802.3 MII. Этот интерфейс относится к классу последовательных синхронных интерфейсов.

Процедуры доступа к блоку PHY осуществляются через регистр Management Interface.

```
// Чтение бита из MII
static int mii_read( void )
{
int res;

    ETH_BANK = 3;

    ETH_MGMT = 0x3330;
    ETH_MGMT = 0x3334;

    res      = ETH_MGMT;

    ETH_MGMT = 0x3330;

    if( res & 0x0002 )
        return 1;
    else
        return 0;
}

// Запись бита в MII или перевод MII на вход (Z)
static void mii_write( unsigned short value )
{
    ETH_BASE = 3;

    switch( value )
    {
    case 0: // '0'
        ETH_MGMT = 0x3338;
        ETH_MGMT = 0x333C;
        ETH_MGMT = 0x3338; break;

    case 1: // '1'
        ETH_MGMT = 0x3339;
        ETH_MGMT = 0x333D;
        ETH_MGMT = 0x3339; break;
    case 2: // 'Z'
        ETH_MGMT = 0x3330;
        ETH_MGMT = 0x3334;
        ETH_MGMT = 0x3330; break;
    }
}

// Начало посылки через MII
static void mii_start( unsigned char mode, unsigned char addr )
{
int i;

    ETH_BANK = 3;

    // Синхропоследовательность (преамбула) 32 * '1'
    for( i = 0; i < 32; i++ )
    {
        ETH_MGMT = 0x3339;
        ETH_MGMT = 0x333D;
    }


    // Старт-бит '01'

    mii_write( 0 );
    mii_write( 1 );


    // Операция (read/write)

    if( mode == MII_WRITE )
    {
```

```
        mii_write( 0 );
        mii_write( 1 );
    }
    else
    {
        mii_write( 1 );
        mii_write( 0 );
    }

    // Адрес устройства PHY (для 91C111 = 0)

    for( i = 0; i < 5; i++ )
        mii_write( 0 );

    // Адрес регистра

    for( i = 0; i < 5; i++ )
    {
        if( addr & 0x10 )
            mii_write( 1 );
        else
            mii_write( 0 );

        addr <<= 1;
    }

    // Переключение направления интерфейса

    if( mode == MII_WRITE )
    {
        mii_write( 1 );
        mii_write( 0 );
    }
    else
    {
        mii_write( 2 );
    }
}
```

Функции доступа к регистрам блока PHY построены на базе представленных выше функций.

```
/**-------------------------------------------------------------------------------
                            read_phy_reg
-------------------------------------------------------------------------------
Чтение регистра PHY

Вход:       addr - адрес регистра PHY
Выход:      нет
Результат:  прочитанные данные
Описание:
Пример:
-------------------------------------------------------------------------------*/

unsigned short read_phy_reg( unsigned char addr )
{
int i;
unsigned short data;

    mii_start( MII_READ, addr );

    ETH_BANK = 3;

    data = 0;

    for( i = 0; i < 16; i++ )
    {
        data <<= 1;

        if( mii_read() )
            data |= 1;
    }

    mii_write( 2 );

    return data;
}
```

```
/**-----------------------------------------------------------------------------
                            write_phy_reg
-----------------------------------------------------------------------------
Запись в регистр PHY

Вход:       addr - адрес регистра PHY
            data - записываемые данные
Выход:      нет
Результат:  нет
Описание:
Пример:
-----------------------------------------------------------------------------*/


void write_phy_reg( unsigned char addr, unsigned short data )
{
int i;

    mii_start( MII_WRITE, addr );

    ETH_BASE = 3;

    for( i = 0; i < 16; i++ )
    {
        if( data & 0x8000 )
            mii_write( 1 );
        else
            mii_write( 0 );

        data <<= 1;
    }

    ETH_MGMT = 0x3330;

}
```

Регистр управления – адрес 0

RST	LPBK	SPEED	ANEG_EN	PDN	MII_DIS	ANEG_RST	DPLX
RW, SC	RW	RW	RW	RW	RW	RW. SC	RW
0	0	1	1	0	1	0	0

COLST	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
RW	RW	RW	RW	RW	RW	RW	RW
0	0	0	0	0	0	0	0

RST – запись «1» в этот бит приводит к сбросу блока PHY. Регистр автоматически сбрасывается в «0» после окончания процедуры сброса. Любая запись в регистр во время сброса будет игнорирована. Время операции сброса занимает 50 мс и на это время необходимо задержать все операции с контроллером.

LPBK – бит LOOPBACK для переключения блока PHY в режим проверки. После записи «1» в этот регистр вход замыкается на выход.

SPEED – выбор скорости. Если режим автоопределения выключен, то запись «1» приводит к переключению на скорость 100 Мбит/сек, а «0» на скорость 10 Мбит/сек.

ANEG_EN – автоопределение. Если записать «1» в этот бит, то контроллер будет сам определять скорость и тип обмена, а состояние битов SPEED и DPLX будут игнорироваться.

PDN – power down. Запись «1» приводит к выключению блока PHY для экономии энергии. В этом режиме доступен только интерфейс MII.

MII_DIS – выключение интерфейса MII. После сигнала RESET интерфейс MII заблокирован и все команды MII игнорируются блоком PHY. Для обеспечения доступа к блоку PHY необходимо записать в этот бит «0».

ANEG_RST – из этого бита читается «0», если блок PHY в данный момент не поддерживает режим автоопределения. Если этот бит равен «1», то можно сбросить процессы автоопределения в исходное состояние путем записи «1» в этот бит.

DPLX – запись «1» в этот бит приводит к переключению в режим полного дуплекса.

COLST – запись «1» в этот бит приводит к запуску теста коллизий.

Регистры MAC уровня

	BANK0	BANK1	BANK2	BANK3
0	TCR	CONFIG	MMU COMMAND	MT0-1
2	EPH STATUS	BASE	PNR	MT2-3
4	RCR	IA0-1	FIFO PORTS	MT4-5
6	COUNTER	IA2-3	POINTER	MT6-7
8	MIR	IA4-5	DATA	MGMT
A	RPCR	GENERAL	DATA	REVISION
C	RESERVED	CONTROL	INTERRUPT	ERCV
E	BANK	BANK	BANK	BANK

Рисунок 15 Регистры MAC уровня

BANK- Регистр выбора банка

Смещение 0xE.

HIGH BYTE	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
	0	0	1	1	0	0	1	1
LOW BYTE						BS2	BS1	BS0
	X	X	X	X	X	0	0	0

BS2..BS0 – номер банка регистров.

Банк 0, TCR - регистр управления передачей

Смещение 0.

HIGH BYTE	SWFDUP	Reserved	EPH LOOP	STP SQET	FDUPLX	MON_ CSN	Reserved	NOCRC
	0	0	0	0	0	0	0	0
LOW BYTE	PAD_EN	Reserved	Reserved	Reserved	Reserved	FORCOL	LOOP	TXENA
	0	0	0	0	0	0	0	0

SWFDUP – включение работы с коммутатором в режиме полного дуплекса. В этом режиме запрещен анализ коллизий т.к. при работе с сетевым коммутатором в полном дуплексе коллизий не может быть. При установке этого бита FDUPLX и MON_CSN не используются.

EPH LOOP – тестовый режим. Отправленные данные возвращаются на собственный приемник.

STP SQET – завершение передачи при наличии ошибки SQET.

FDUPLX – выключение проверки заголовка принимаемого пакета. В режиме полудуплекса этот бит должен быть установлен «0», т.к. иначе мы будет принимать пакеты отправленные нашим контроллером.

MON_CSN – когда этот бит установлен, контроллер проверяет наличие несущей во время передачи.

NOCRC – не добавлять CRC к пакету при передаче. Позволяет программному обеспечению самому добавлять CRC к передаваемому пакету.

PAD_EN – если установить этот бит, то контроллер будет автоматически удлинять короткие пакеты до длины 64 байта.

FORCOL – (Force a Collision) искусственно вызывает коллизию, при TXENA = 1. Бит управляется программно.

LOOP – тестовый режим.

TXENA – разрешение передачи.

Банк 0, RCR - регистр управления приемом

Смещение 4

HIGH BYTE	SOFT RST	FILT CAR	ABORT_ENB	Reserved	Reserved	Reserved	STRIP CRC	RXEN
	0	0	0	0	0	0	0	0
LOW BYTE	Reserved	Reserved	Reserved	Reserved	Reserved	ALMUL	PRMS	RX_ ABORT
	0	0	0	0	0	0	0	0

SOFT RST – программно-управляемый сброс. Начинается после записи «1» и заканчивается после записи «0».

FILT CAR – включение фильтра несущей.

ABORT_ENB – разрешение прерывания приема при наличии коллизии.

STRIP CRC – удаляет CRC из принятых пакетов.

RXEN – включение приема пакетов.

ALMUL – разрешает прием широковещательных (multicast) пакетов.

PRMS – включение приема всех пакетов (Promiscuous mode).

RX_ABORT – этот бит устанавливается, если принятый пакет длиннее 2 килобайт. Бит равен 0 после сброса. Сбрасывается программно.

Банк 0, RPCR - регистр управления приемом на физическом уровне

Смещение 0xA

HIGH BYTE	Reserved	Reserved	SPEED	DPLX	ANEG	Reserved	Reserved	Reserved
	0	0	0	0	0	0	0	0
LOW BYTE	LS2A	LS1A	LS0A	LS2B	LS1B	LS0B	Reserved	Reserved
	0	0	0	0	0	0	0	0

SPEED – выбор скорости обмена. Этот бит имеет смысл, если ANEG = 0.

DPLX – включение полного дуплекса. Этот бит имеет смысл, если ANEG = 0.

ANEG – автоопределение скорости обмена и типа дуплекса.

LSxx – биты выбора источников сигнала для светодиодов LEDA и LEDB.

LS2A	LS1A	LS0A	LED SELECT SIGNAL – LEDA
0	0	0	nPLED3+ nPLED0 – Logical OR of 100Mbps Link detected 10Mbps Link detected (default)
0	0	1	Reserved
0	1	0	nPLED0 - 10Mbps Link detected
0	1	1	nPLED1 - Full Duplex Mode enabled
1	0	0	nPLED2 - Transmit or Receive packet occurred
1	0	1	nPLED3 - 100Mbps Link detected
1	1	0	nPLED4 - Receive packet occurred
1	1	1	nPLED5 - Transmit packet occurred

LS2B	LS1B	LS0B	LED SELECT SIGNAL – LEDB
0	0	0	nPLED3+ nPLED0 – Logical OR of 100Mbps Link detected 10Mbps Link detected (default)
0	0	1	Reserved
0	1	0	nPLED0 - 10Mbps Link detected
0	1	1	nPLED1 – Full Duplex Mode enabled
1	0	0	nPLED2 – Transmit or Receive packet occurred
1	0	1	nPLED3 - 100Mbps Link detected
1	1	0	nPLED4 - Receive packet occurred
1	1	1	nPLED5 - Transmit packet occurred

Банк 1, CONFIG - регистр конфигурации

Смещение – 0.

HIGH BYTE	EPH Power EN	Reserved	Reserved	NO WAIT	Reserved	GPCNTRL	EXT PHY	Reserved
	1	0	1	0	0	0	0	0
LOW BYTE	Reserved	Reserved	Reserved			Reserved	Reserved	Reserved
	1	0	1	1	0	0	0	1

EPH Power EN – управление энергопотреблением EPH. Если значение бита = «0», то EPH переходит в режим пониженного энергопотребления.

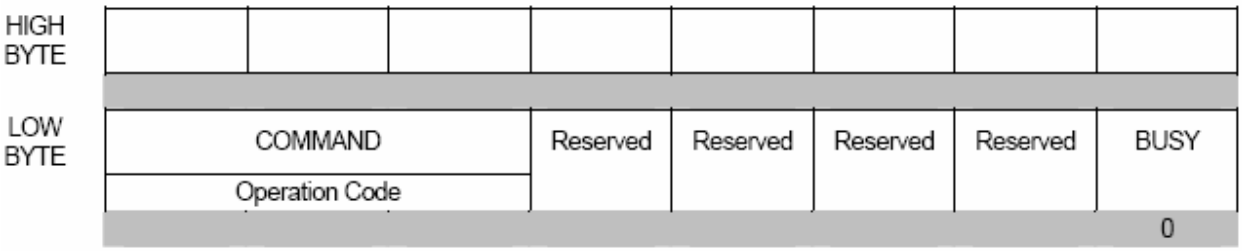
NO WAIT - если установлена «1» не требуются дополнительные такты неготовности.

GPCNTR – бит управляет дискретным портом вывода общего назначения.

EXT PHY – бит переключает контроллер на использование внешнего блока PHY.

Банк 2, MMU COMMAND - регистр команд MMU

Смещение 0



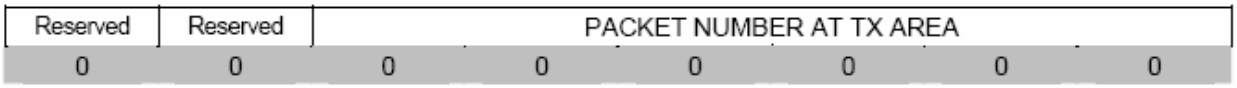
COMMAND – код команды.

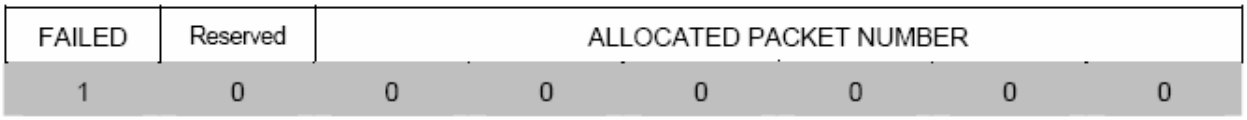
Код	Описание
0	NOOP – нет операции
1	Выделить память в очереди для передачи
2	Сбросить MMU в исходное состояние
3	Убрать пакет с вершины приемной очереди
4	Аналогично команде 3.
5	Освобождает страницы памяти, выделенные для пакета, номер которого указан в регистре PACKET NUMBER REGISTER.
6	Выбрать и передать пакет с номером указанным в регистре PACKET NUMBER REGISTER.
7	Сброс очереди на передачу.

BUSY – равен «1» пока исполняется команда.

Банк 2, регистр номера пакета

Смещение 2

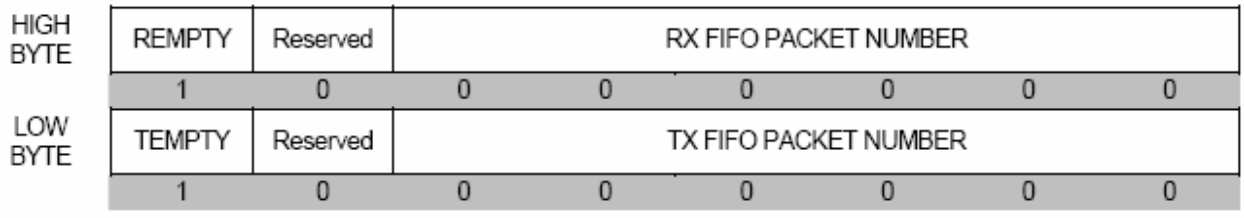




FAILED – бит равен «0», если выделение памяти прошло успешно.

Банк 2, FIFO PORTS - регистр кольцевых буферов

Смещение 4



REMPTY – приемной очереди нет принятых пакетов.

RX FIFO PACKET NUMBER – номер пакета находящегося в конце приемного буфера. Значение имеет смысл если бит REMPTY = 0.

TEMPTY – очередь пакетов на передачу пуста.

TX FIFO PACKET NUMBER – номер пакета находящегося в конце буфера передачи. Значение имеет смысл если бит TEMPTY = 0.

Банк 2, POINTER - регистр указатель

Смещение 6

Регистр предназначен для установки указателя в памяти очередей для приема и передачи.

HIGH BYTE	RCV	AUTO INCR.	READ	ETEN	NOT EMPTY	POINTER HIGH		
	0	0	0	0	0	0	0	0
LOW BYTE	POINTER LOW							
	0	0	0	0	0	0	0	0

RCV – «1» обращение к приемной очереди, «0» - к передающей.
AUTOINCR – включение автоматического инкремента адреса.
READ – определяет тип операции: «1» - чтение, «0» - запись.
ETEN – включение определения преждевременной передачи пакета (когда пакет еще не до конца сформирован в памяти контроллера).
NOT EMPTY – бит равен «1», если буфер передачи не пуст.

Банк 2, DATA - регистр данных

Смещение 8

DATA HIGH							
X	X	X	X	X	X	X	X
DATA LOW							
X	X	X	X	X	X	X	X

Банк 2, INTERRUPT - регистр статуса прерываний

Смещение 0xC, только чтение

MDINT	ERCV INT	EPH INT	RX_OVRN INT	ALLOC INT	TX EMPTY INT	TX INT	RCV INT
0	0	0	0	0	1	0	0

Смещение 0xC, только запись

MDINT	ERCV INT		RX_OVRN INT		TX EMPTY INT	TX INT	

Смещение 0xD, чтение и запись

MDINT MASK	ERCV INT MASK	EPH INT MASK	RX_OVRN INT MASK	ALLOC INT MASK	TX EMPTY INT MASK	TX INT MASK	RCV INT MASK
0	0	0	0	0	0	0	0

MDINT – устанавливается в «1», если один из битов PHY меняет свое значение: LNKFAIL, LOSSSYNC, CWRD, SSD, ESD, PROL, JAB, SPDDET, DPLXDET
ERCV INT – устанавливается в «1», когда размер принимаемого пакета превышает значение, установленное в регистре ERCV TRESHOLD (банк 3, смещение 0xC).
EPH INT – устанавливается в «1», когда секция «протокол» в заголовке пакета не соответствует заданному значению.
RX_OVRN INT – устанавливается в нескольких случаях: нет места в приемном буфере, пакет слишком большой, установлен бит RCV DISCRD в регистре ERCV.
ALLOC INT – успешное выполнение команды выделения памяти для передачи (MMU).
TX_EMPTY INT – очередь пакетов на передачу пуста.
TX INT – устанавливается если пакет успешно передан или в случае ряда ошибок: TXUNRN, SQET, LOST CARR, LAT COL, 16COL.
RCV INT – принят пакет.

Банк 3, регистр MMI интерфейса

HIGH BYTE	Reserved	MSK_ CRS100	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
	0	0	1	1	0	0	1	1
LOW BYTE	Reserved				MDOE	MCLK	MDI	MDO
	0	0	1	1	0	0	MDI Pin	0

MDO – управление выводом передачи.
MDI – вход.
MCLK – вывод синхронизации.
MDOE – вывод управления передачей (Output Enable).

Сброс контроллера

Сброс производится установкой бита RESET контроллера в логическую «1» на время большее 100 нс. Во время сброса все регистры контроллера переводятся в исходное состояние указанное в документации.

Инициализация блока PHY

При инициализации блока PHY необходимо включить внутренний блок PHY путем записи соответствующего значения в регистр конфигурации.

Инициализация приема и передачи

Необходимо инициализировать регистры управления приемом и передачей, а также регистр управления приемом физического уровня.

```
/**-----------------------------------------------------------------------------
                            init_eth
-----------------------------------------------------------------------------
Инициализация контроллера ethernet

Вход:       mode - режим рыботы драйвера
Выход:      нет
Результат:  нет
Описание:
Пример:
-----------------------------------------------------------------------------*/

void init_eth( unsigned long mode )
{
volatile long i;

    // Сброс контроллера
    IO3SET_bit.P3_29 = 1;

    for( i = 0; i < 50000; i++ );    // задержка

    IO3CLR_bit.P3_29 = 1;

    // Инициализация PHY
    init_eth_phy();

    for( i = 0; i < 50000; i++ );    // задержка

    write_phy_reg( PHY_CONTROL, 0x3000 );


    ETH_BANK = 0; ETH_RCR       = 0x0102;
    ETH_BANK = 0; ETH_TCR       = 0x0001;
    ETH_BANK = 0; ETH_RPCR      = 0x2800;
}
```

Чтение пакета

```
int read_eth_pack( unsigned char * buf, unsigned short * len )
{
unsigned short wtmp, n, i, k;
unsigned short interrupt_status, fifo_ports;

        ETH_BANK = 2; interrupt_status = ETH_INTERRUPT;
        ETH_BANK = 2; fifo_ports       = ETH_FIFO_PORTS;

        if(  ( interrupt_status & 0x0001 ) &&
```

```
            !( fifo_ports       & 0x8000 ) )
        {
            ETH_BANK = 2; ETH_POINTER = 0xE000;
            ETH_BANK = 2; n           = ETH_DATA0 & 0x7FF;

            n -= 4;

            *len = n;

            if( n%2 )
                n = n >> 1  + 1;
            else
                n = n >> 1;

            k = 0;

            for( i = 0; i < n; i++ )
            {
                ETH_BANK = 2; wtmp = ETH_DATA0;

                buf[k++] = wtmp;
                buf[k++] = wtmp >> 8;
            }

            ETH_BANK = 2; ETH_MMU_COMMAND = 0x0080; // Убираем переданный пакет из очереди

            return OK;
        }
        else
            return ERROR;
}
```

Запись пакета

```
int write_eth_pack( unsigned char * buf, unsigned short len )
{
unsigned short wtmp, n, i, k, busy;
unsigned short interrupt_status, allocate_number;
unsigned long t1;


        ETH_BANK = 2; ETH_MMU_COMMAND = 0x0020; // выделяем память для пакета

        // Ждем когда память под пакет выделится

        t1 = clock();

        for( ;; )
        {
            ETH_BANK = 2; interrupt_status = ETH_INTERRUPT;

            if( interrupt_status & 0x0008 ) break;

            if( dtime( t1 ) > ETH_ALLOCATE_TO )
            {
#if ETH_DEBUG
                printf("write_eth_pack(): error of allocation memory for transmitting\n");
#endif
                return ERROR;
            }
        }

        ETH_BANK = 2; allocate_number = ETH_PNR >> 8;
        ETH_BANK = 2; ETH_PNR         = allocate_number;

        ETH_BANK = 2; ETH_POINTER     = 0x4000;

        ETH_BANK = 2; ETH_DATA0 = 0;        // status
        ETH_BANK = 2; ETH_DATA0 = len + 6;  // packet len

        n = len;

        if( n%2 )
            n = n >> 1  + 1;
        else
            n = n >> 1;
```

```
        k = 0;

        for( i = 0; i < n; i++ )
        {

            wtmp =  buf[ k + 1 ]; wtmp <<= 8;
            wtmp += buf[ k ];

            k += 2;


            ETH_BANK = 2; ETH_DATA0 = wtmp;

        }

        ETH_BANK = 2; ETH_TCR = 0x0001; // TXENA = 1;

// Ждем, когда флаг Busy установится в 0

        while( 1 )
        {
            ETH_BANK = 2; busy = ETH_MMU_COMMAND;

            if( busy & 0x0001 ) break;

            if( dtime( t1 ) > ETH_TRANSMIT_TO )
            {
#if ETH_DEBUG
                printf("write_eth_pack(): TO of transmitting\n");
#endif
                return ERROR;
            }
        }

// Сбрасываем последний переданный пакет

        ETH_BANK = 2; ETH_MMU_COMMAND = 0x00C0;

        return OK;
}
```

Драйвер CAN для микроконтроллера LPC2292

В микроконтроллере LPC2292 два контроллера CAN.

```
// Определение двух CAN-контроллеров
#define CAN0    0
#define CAN1    1

// Определение ошибок (возвращаемое функцией значение)
#define OK          0
#define ERROR       1

#define VIC_CANRX1_bit (1 << VIC_CANRX1)     // Формирование маски прерывания (прием по CAN0)
#define VIC_CANRX2_bit (1 << VIC_CANRX2)     // Формирование маски прерывания (прием по CAN1)
#define VIC_CANERR_bit (1 << VIC_CANERR)     // Формирование маски прерывания (ошибки в CAN)

#define MAX_CAN_PORT        2        // Количество контроллеров CAN
#define MAX_CAN_PIPE_SIZE   16       // Размер буфера (очереди)

// Определение типа данных для хранения сообщений CAN
typedef struct
{
        unsigned int frame;    // Биты 16..19: код длины данных (DLC)
                               // Бит 30 устанавливается, как бит запроса удаленной передачи (RTR)
                               // Бит 31: устанавливается, если это сообщение расширенного формата
                               //(29-битный идентификатор)
        unsigned int id;       // Идентификатор сообщения CAN (11-битный или 29-битный)
        unsigned int data_a;          // Поле данных сообщения CAN (0-3 байты)
```

```
        unsigned int data_b;          // Поле данных сообщения CAN (4-7 байты)
} CANMSG;

// Определение типа данных, описывающего буфер (очередь) сообщений на прием
typedef struct {
    CANMSG         msg[ MAX_CAN_PIPE_SIZE ];        // Очередь сообщений
    unsigned short start;     // "Голова" очереди (очередь кольцевая)
    unsigned short size;      // Размер активной очереди (т. е. количество сообщений, которые
                              //были не использованы); "хвост" = start+size

}CANPIPE;

// Таблица со смещениями до регистров контроллеров CAN1 и CAN2
static unsigned long offset_table[2] = { 0x00000000L, 0x00001000L };

static char error_handler_install = 0;    // =0, если обработчик ошибок CAN не установлен

static int can_baud[2];

static CANPIPE can_pipe[ MAX_CAN_PORT ];    // Очереди сообщений для всех контроллеров CAN


/*-----------------------------------------------------------------------------
                            init_can
-----------------------------------------------------------------------------
Инициализация CAN

Вход:  interface – тип CAN (0 или 1),
       baud – скорость передачи по каналу
Выход:      нет
Результат:  ERROR или OK
-----------------------------------------------------------------------------*/

int init_can( int interface, int baud )
{
        unsigned int *pSFR;   // указатель на регистры SFR
        unsigned int offset;  // смещение, добавляемое к pSFR

        if( OK != check_can_interface( interface ) ) return ERROR;

        init_can_pipe();

        AFMR = 0x00000002;


        can_baud[ interface ] = baud;

        offset = offset_table[ interface ];

        pSFR = (unsigned int *) &C1MOD + offset;     // Выбрать регистр режима работы
        *pSFR = 1;     // Установить режим сброса (перезагрузки)

        pSFR = (unsigned int *) &C1IER + offset;     // Выбрать регистр разрешения прерываний
        *pSFR = 0;     // Запретить все прерывания

        pSFR = (unsigned int *) &C1GSR + offset;     // Выбрать глобальный регистр статуса
        *pSFR = 0;     // Очистить глобальный регистр статуса

        pSFR = (unsigned int *) &C1BTR + offset; // Выбрать регистр временных характеристик шины
        *pSFR = baud;  // Установить время передачи бита

        switch ( interface )
        {
                case CAN0:

                        VICIntSelect &= ~VIC_CANRX1_bit;     // Тип прерывания IRQ
                        // Адрес обработчика прерываний
                        VICVectAddr1 = (unsigned int)&CANAll_CANISR_Rx1;
                        VICVectCntl1 = 0x20 | VIC_CANRX1;    // Включить векторное прерывание
                        VICIntEnable = VIC_CANRX1_bit;       // Разрешить прерывание
                        break;

                case CAN1:

                        VICIntSelect &= ~VIC_CANRX2_bit;     // Тип прерывания IRQ
                        // Адрес обработчика прерываний
                        VICVectAddr2 = (unsigned int)&CANAll_CANISR_Rx2;
                        VICVectCntl2 = 0x20 | VIC_CANRX2;    // Включить векторное прерывание
                        VICIntEnable = VIC_CANRX2_bit;       // Разрешить прерывание
                        break;
```

```
                default: return ERROR;
        }


        if ( error_handler_install == 0 )
        {

                VICIntSelect &= ~VIC_CANERR_bit;      // Тип прерывания IRQ
                // Адрес обработчика прерываний
                VICVectAddr5 = (unsigned int)&CANAll_CANISR_Err;
                VICVectCntl5 = 0x20 | VIC_CANERR;     // Включить векторное прерывание
                VICIntEnable = VIC_CANERR_bit;        // Разрешить прерывание

                error_handler_install = 1;
        }

        pSFR = (unsigned int *) &C1IER + offset;      // Выбрать регистр разрешения прерываний
        pSFR = 0x1;     // Разрешить прерывание на прием

        // Установка нормального режима работы CAN
        pSFR = (unsigned int *) &C1MOD + offset;      // Выбрать регистр режима работы
        pSFR = 0;       // Режим работы контроллера CAN (выполняет действия)

        return OK;
}

/*-----------------------------------------------------------------------------
                              write_can
-----------------------------------------------------------------------------
Передача сообщений по CAN

Вход:   interface – тип CAN (0 или 1),
        pTransmitBuf – адрес буфера передачи
Выход:          нет
Результат:      ERROR или OK
-----------------------------------------------------------------------------*/

int write_can_packet ( int interface, CANMSG *pTransmitBuf )
{
        unsigned int *pAddr;
        unsigned int *pCandata;
        unsigned int offset;


        if ( OK != check_can_interface( interface ) ) return ERROR;

        offset = offset_table[ interface ];

        pAddr = (unsigned int *) &C1SR + offset;      // Выбрать регистр статуса
        // Проверка: доступен ли канал передачи
        if (!(*pAddr & 0x00000004L))
        {
                // Канал передачи не доступен
                return ERROR;
        }

        // Запись DLC, RTR и FF для передаваемого сообщения
        pAddr = (unsigned int *) &C1TFI1 + offset;   // Регистр информации о кадре передачи
        *pAddr = pTransmitBuf->frame & 0xC00F0000L; // Установка DLC (8 байт), RTR и FF
                                                    //(расширенный формат сообщения)

        // Запись идентификатора сообщения в регистр идентификатора передачи
        pAddr++;
        *pAddr = pTransmitBuf->id;

        // Запись первых 4 байтов сообщения в регистр1 передаваемых данных
        pCandata = (unsigned int *) &(pTransmitBuf->data_a);
        pAddr++;
        *pAddr = *pCandata;

        // Запись вторых 4 байтов сообщения в регистр2 передаваемых данных
        pCandata++;
        pAddr++;
        *pAddr = *pCandata;

        // Работа с регистром команд
        pAddr = (unsigned int *) &C1CMR + offset;    // Выбрать регистр команд
        *pAddr = 0x30; // Выбрать Self Transmission Request и буфер1 для передачи
```

```
        return OK;
}

/*-----------------------------------------------------------------------------
                            read_can
-----------------------------------------------------------------------------
Прием сообщений из CAN

Вход:   interface – тип CAN (0 или 1),
        pReceiveBuf – адрес буфера приема

Выход:          нет
Результат:      ERROR или OK
-----------------------------------------------------------------------------*/

int read_can_packet ( int interface, CANMSG *pReceiveBuf )
{
        if ( OK != check_can_interface( interface ) ) return ERROR;

        return can_pipe_read( interface, pReceiveBuf );
}

/*-----------------------------------------------------------------------------
                            check_busoff
-----------------------------------------------------------------------------
Проверка нахождения контроллера в состоянии BusOff

Вход:   interface – тип CAN (0 или 1)

Выход:          нет
Результат:      ERROR или OK
-----------------------------------------------------------------------------*/

int check_busoff( int interface )
{
        unsigned int *pSFR;
        unsigned int offset;

        if ( OK != check_can_interface( interface ) ) return ERROR;

        __disable_interrupt();        // Запретить прерывания

        offset = offset_table[ interface ];

        pSFR = (unsigned int *) &C1GSR + offset;     // Выбрать глобальный регистр статуса

        // Проверка: если контроллер находится в состоянии BusOff, то инициализировать CAN
        if ( *pSFR & 0x80 ) init_can( interface, can_baud[interface] );

        __enable_interrupt();         // Разрешить прерывания

        return OK;

}

/*-----------------------------------------------------------------------------
                            check_can_interface
-----------------------------------------------------------------------------
Проверка корректности обращения к CAN (по номеру).

Вход:   interface – тип CAN (0 или 1)

Выход:          нет
Результат:      ERROR или OK
-----------------------------------------------------------------------------*/

static int check_can_interface( int interface )
{
        if( interface > CAN1 ) return ERROR;
        else return OK;
}

/*-----------------------------------------------------------------------------
                            Interrupt_Service_Routine
-----------------------------------------------------------------------------
Обработчик прерывания по ошибкам приема/передачи по CAN
-----------------------------------------------------------------------------*/
static void CANAll_CANISR_Err ( void )
{
        // Проверка на переполнение счетчика ошибок передачи (> 255)
```

```
        if ( C1SR_bit.BS )
        {
                C1MOD = 0;
        }

        if ( C2SR_bit.BS )
        {
                C2MOD = 0;
        }

        VICVectAddr = 0xFFFFFFFFL;    // Подтверждение прерывания
}

/*-----------------------------------------------------------------------------
                            Interrupt_Service_Routine
-----------------------------------------------------------------------------
Обработчик прерывания по приему сообщений CAN0.
-----------------------------------------------------------------------------*/

static void CANAll_CANISR_Rx1 ( void )
{
        CANMSG msg;

        msg.frame  = C1RFS;   // Чтение характеристик принятого сообщения (DLC, RTR, FF,...)
        msg.id     = C1RID;   // Чтение идентификатора принятого сообщения
        msg.data_a = C1RDA;   // Чтение первой половины сообщения (0-4 байта)
        msg.data_b = C1RDB;   // Чтение второй половины сообщения (5-7 байта)

        // Запись сообщения в очередь первого контроллера CAN
        can_pipe_write( CAN0, &msg ) ;

        C1CMR = 0x04;  // Освобождение буфера приема
        VICVectAddr = 0xFFFFFFFFL;    // Подтверждение прерывания
}

/*-----------------------------------------------------------------------------
                            Interrupt_Service_Routine
-----------------------------------------------------------------------------
Обработчик прерывания по приему сообщений CAN1.
-----------------------------------------------------------------------------*/

static void CANAll_CANISR_Rx2 ( void )
{
        CANMSG msg;

        msg.frame  = C2RFS;   // Чтение характеристик принятого сообщения (DLC, RTR, FF,...)
        msg.id     = C2RID;   // Чтение идентификатора принятого сообщения
        msg.data_a = C2RDA;   // Чтение первой половины сообщения (0-4 байта)
        msg.data_b = C2RDB;   // Чтение второй половины сообщения (5-7 байта)

        // Запись сообщения в очередь второго контроллера CAN
        can_pipe_write( CAN1, &msg )  ;

        C2CMR = 0x04;  // Освобождение буфера приема
        VICVectAddr = 0xFFFFFFFFL;    // Подтверждение прерывания
}

/*-----------------------------------------------------------------------------
                            can_pipe_write
-----------------------------------------------------------------------------
Запись сообщений в очередь CAN (по номеру).

Вход:   interface – тип CAN (0 или 1),
        msg – записываемое сообщение

Выход:          нет
Результат:      ERROR или OK
-----------------------------------------------------------------------------*/

static int can_pipe_write( int interface, CANMSG * msg )
{
        unsigned short index;

        if ( OK != check_can_interface( interface ) ) return ERROR;

        // Проверка: размер активной очереди не должен быть больше максимального размера очереди
        if ( can_pipe[ interface].size < MAX_CAN_PIPE_SIZE )
        {
                // Определение номера свободного элемента очереди для записи нового сообщения
                index = ( can_pipe[ interface].start + can_pipe[ interface].size) & (
```

```
        MAX_CAN_PIPE_SIZE - 1 );

                can_pipe[ interface].msg[ index ] = *msg;    // Запись сообщения в очередь

                can_pipe[ interface].size++;         // Увеличение размера активной очереди

                return OK;
        }
        else return ERROR;
}

/*-----------------------------------------------------------------------------
                            can_pipe_read
-----------------------------------------------------------------------------
Чтение сообщения из очереди CAN (по номеру).

Вход:   interface – тип CAN (0 или 1)

Выход:          msg – прочитанное сообщение (принятое по CAN)
Результат:      ERROR или OK
-----------------------------------------------------------------------------*/

static int can_pipe_read( int interface, CANMSG * msg )
{

        if( OK != check_can_interface( interface ) ) return ERROR;

        // Чтение возможно только тогда, когда еще не все сообщения были прочитаны из нее
        if ( can_pipe[ interface ].size == 0 ) return ERROR;

        // Чтение сообщения
        *msg = can_pipe[ interface ].msg[ can_pipe[ interface].start ];

        can_pipe[ interface ].start++;       // Смещение "головы" на одну позицию
        can_pipe[ interface ].start &= ( MAX_CAN_PIPE_SIZE - 1 );
        can_pipe[ interface ].size--;        // Уменьшение размера активной очереди

        return OK;
}

/*-----------------------------------------------------------------------------
                            init_can_pipe
-----------------------------------------------------------------------------
Инициализация очередей контроллеров CAN.

Вход:   нет

Выход:  нет
Результат:      нет
-----------------------------------------------------------------------------*/

static void init_can_pipe( void )
{
        int i;

        for( i = 0; i < MAX_CAN_PORT; i++ )
        {
                can_pipe[i].start = 0;
                can_pipe[i].size  = 0;
        }
}
```