Программное обеспечения для лаборатории организации ЭВМ

Тестовые драйверы и инструментальное обеспечение для работы с учебным стендом SDK-1.1

Научно-образовательное направления № 2
«Встроенные вычислительные системы»

Руководство пользователя

ОГЛАВЛЕНИЕ

ОГЛАВЛЕНИЕ **1**

НАЗНАЧЕНИЕ **2**

СОСТАВ ПРОЕКТА **2**

ИНСТАЛЛЯЦИЯ ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ **2**

Требования к платформе разработки 2
Порядок установки Eclipse 2
Порядок установки Cygwin 3
Компиляция и загрузка тестов 5

ИСПОЛЬЗОВАНИЕ ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ **5**

Использование компилятора SDCC 5
Использование make 5
Пример makefile *6*
Использование Eclipse 7
Программатор Flash для ADuC812 (DL) 13
Использование программы M3P для загрузки тестов 14

ТЕСТОВЫЕ ДРАЙВЕРЫ ДЛЯ УЧЕБНОГО СТЕНДА SDK-1.1 ..... **17**

Последовательный интерфейс 17
Работа с последовательным каналом по опросу *19*
Описание драйвера *20*
Управление светодиодными индикаторами 20
Описание драйвера *21*
Регистры ПЛИС 21
Регистр клавиатуры KB *21*
Регистр шины данных ЖКИ DATA_IND *22*
Регистр данных параллельного порта EXT_LO *22*
Регистр данных параллельного порта EXT_HI *22*
Регистр управления ENA *23*
Регистр управления ЖКИ C_IND *23*
Регистр управления светодиодами SV *24*
Драйвер доступа к регистрам ПЛИС *24*
Драйвер дискретных портов *25*
Проблемы, часто возникающие при доступе к регистрам ПЛИС *25*
Аналого-цифровой преобразователь 26
Передаточная функция АЦП *26*
SFR регистры АЦП *26*
Частота тактирования *28*
Режимы работы *28*
Описание драйвера *29*

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-1.1» состоит из двух частей. В каталоге EXAMPLE находятся исходные тексты драйверов для учебного стенда SDK-1.1.

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

В каталоге HW_TEST находятся исходные тексты комплексного теста учебного стенда SDK-1.1, 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 SDCC v2.7.0 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

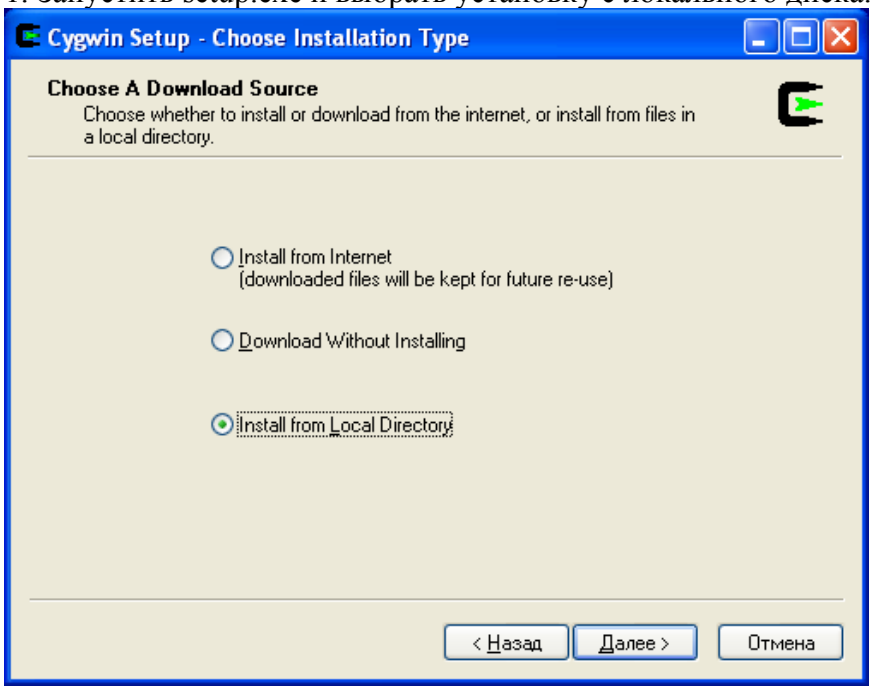
3. Установить CDT из каталога "Zylin embedded CDT". Перед установкой убедиться, что предыдущие дистрибутивы CDT удалены. Установка заключается в простом копировании содержимого каталогов в каталог Eclipse.

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

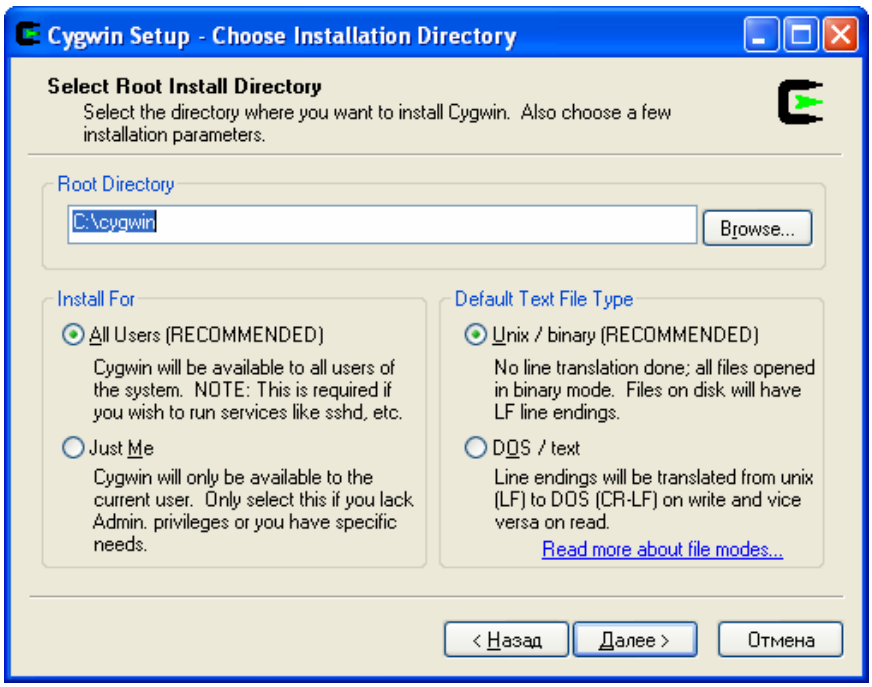
- Linux - embeddedcdt-linux-gtk-20070424.zip
  zylincdt-20070424.zip

Порядок установки Cygwin

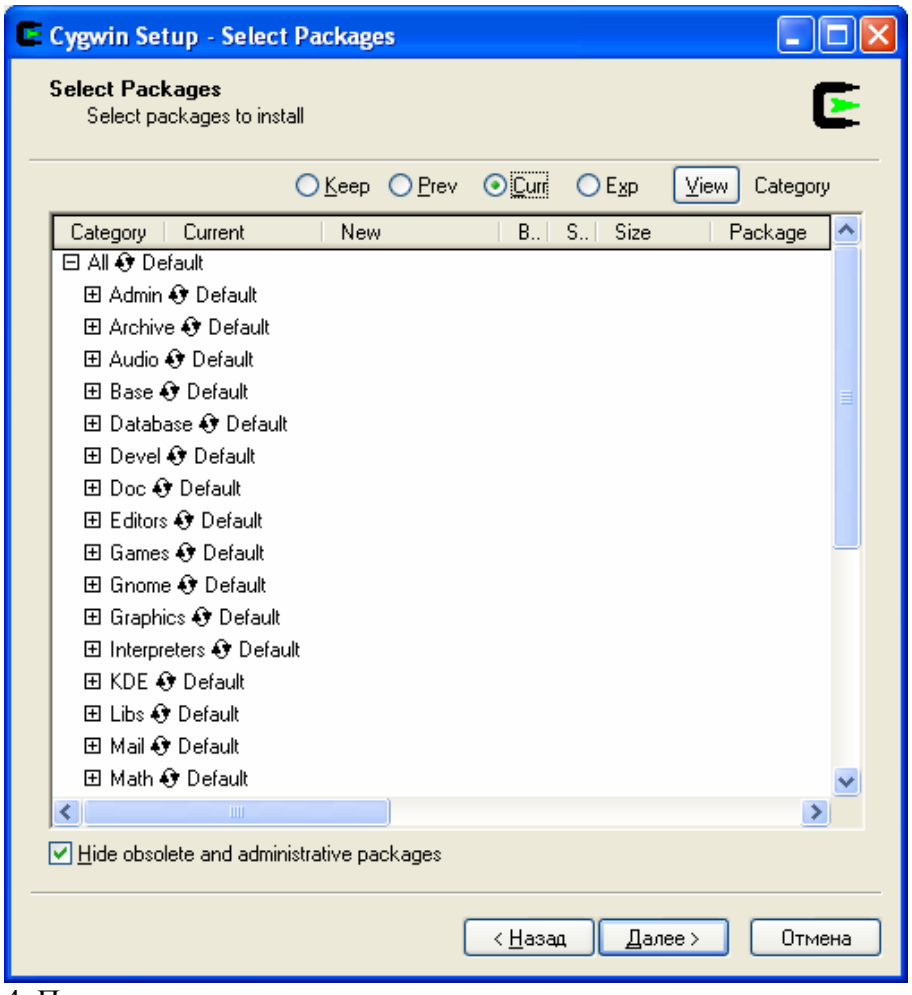
1. Запустить setup.exe и выбрать установку с локального диска.



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 SDCC

Ключи компилятора

Ключ	Описание
-I	Каталог заголовочных файлов
-c	Компилировать в объектный файл

Ключи линкера

Ключ	Описание
--code-loc	Расположение памяти программ
--xram-loc	Расположение внешней памяти данных
-- stack-loc	Расположение стека

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.

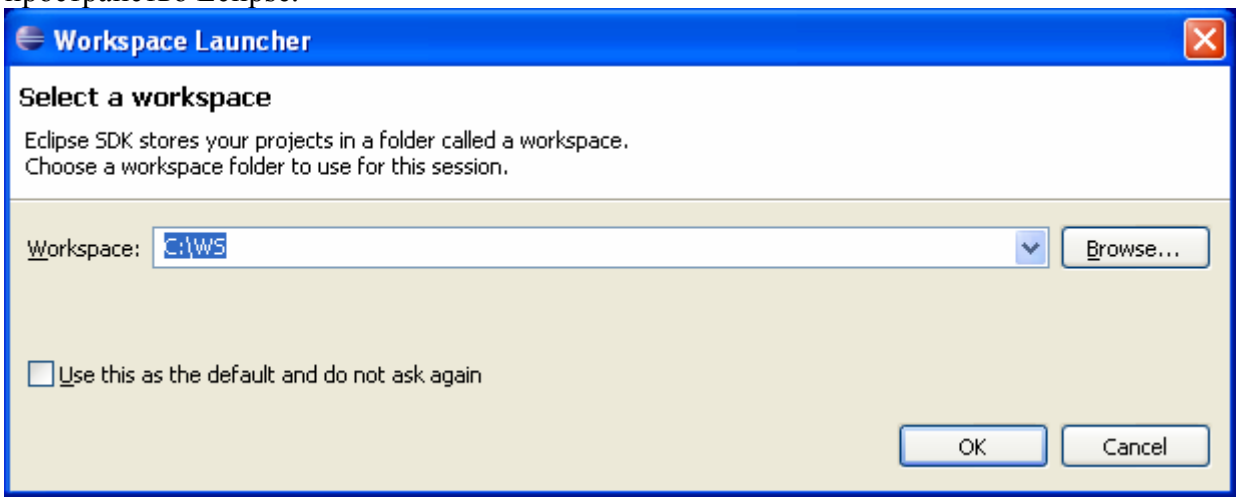
Пример makefile

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

NAME    = hw_test

# Настройки компилятора и линкера

CC      = sdcc
CFLAGS  = -I./INCLUDE -c
LFLAGS  = --code-loc 0x2100 --xram-loc 0x6000 --stack-loc 0x80

# 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  = com1
COMLOG   = $(COMPORT)_log.txt
BAUD     = 9600

# Каталоги с исходными текстами

HW_TEST_SRC_DIR = SRC
# ------------------------------------------------------------------------------

all: hw_test

clean:
        -rm -f  $(NAME).hex \
                        $(NAME).bin \
                        $(NAME).map \
                        $(NAME).mem \
                        $(NAME).lnk \
                        pm3p_*.txt \
                        $(TARBALL).gz \
                        $(HW_TEST_SRC_DIR)/*.asm \
                        $(HW_TEST_SRC_DIR)/*.rel \
                        $(HW_TEST_SRC_DIR)/*.rst \
                        $(HW_TEST_SRC_DIR)/*.sym \
                        $(HW_TEST_SRC_DIR)/*.lst

load:
        $(M3P) lfile load.m3p


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

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

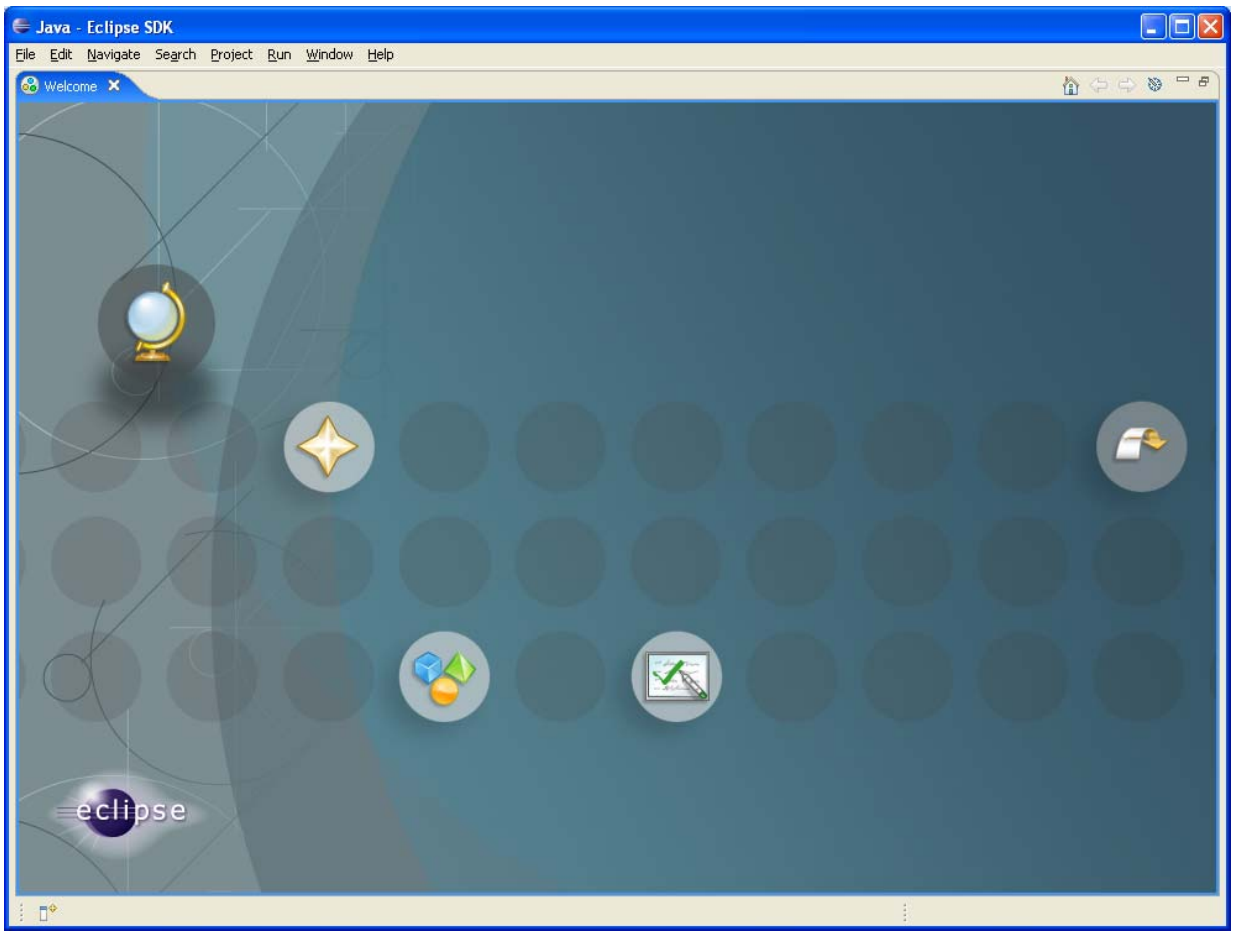
```
HW_TEST_SRC = \
$(HW_TEST_SRC_DIR)/adc.c \
$(HW_TEST_SRC_DIR)/dac.c \
$(HW_TEST_SRC_DIR)/din_dout.c \
$(HW_TEST_SRC_DIR)/eeprom.c \
$(HW_TEST_SRC_DIR)/i2c.c \
$(HW_TEST_SRC_DIR)/led.c \
$(HW_TEST_SRC_DIR)/main.c \
$(HW_TEST_SRC_DIR)/max.c \
$(HW_TEST_SRC_DIR)/rtc.c \
$(HW_TEST_SRC_DIR)/sio.c \
$(HW_TEST_SRC_DIR)/test_dac.c \
$(HW_TEST_SRC_DIR)/test_din.c \
$(HW_TEST_SRC_DIR)/test_e.c \
$(HW_TEST_SRC_DIR)/test_led.c \
$(HW_TEST_SRC_DIR)/test_rtc.c

HW_TEST_OBJ = $(HW_TEST_SRC:.c=.rel)


hw_test : $(HW_TEST_OBJ) makefile
        $(CC) $(HW_TEST_OBJ) -o hw_test.hex $(LFLAGS)
        $(M3P) hb166 hw_test.hex hw_test.bin bye


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

Использование Eclipse

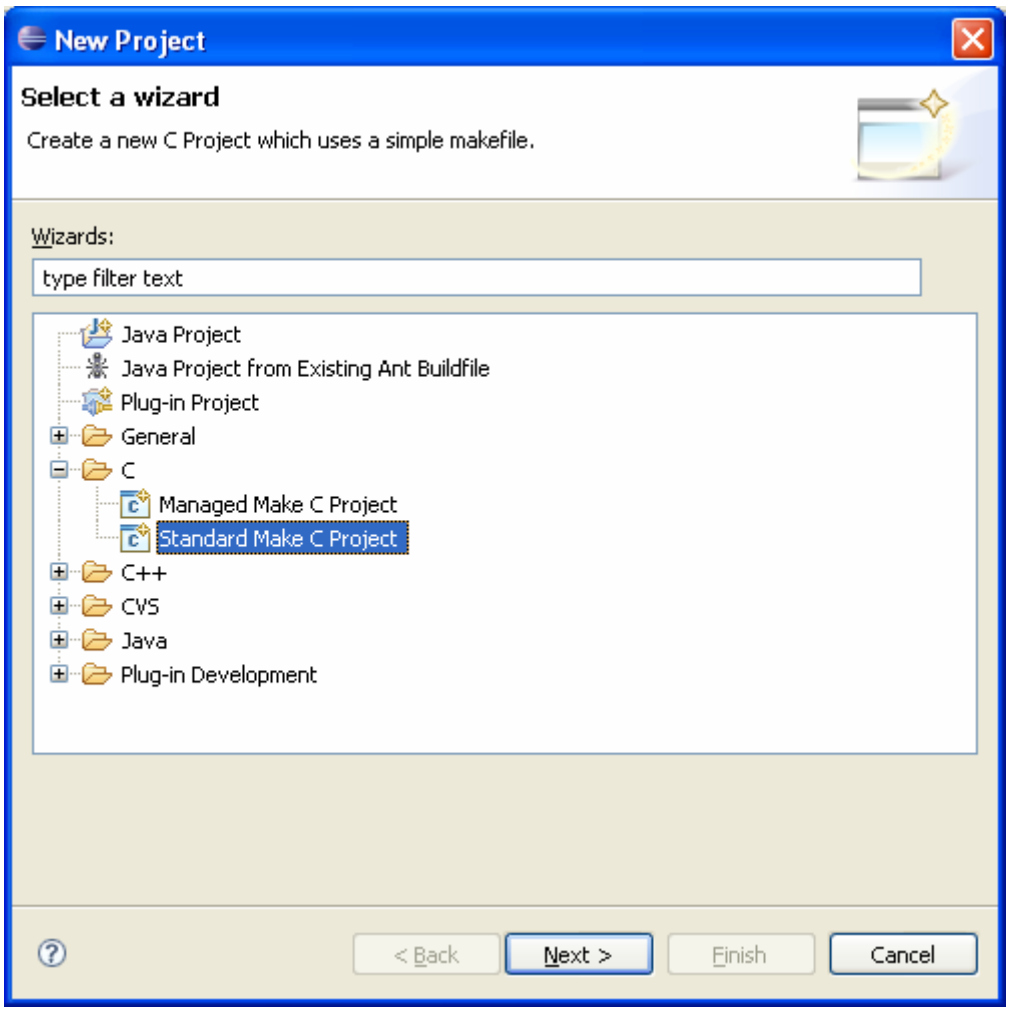
После запуска укажите расположение каталога, в котором будет хранится рабочее пространство Eclipse.



После запуска, окно принимает следующий вид:

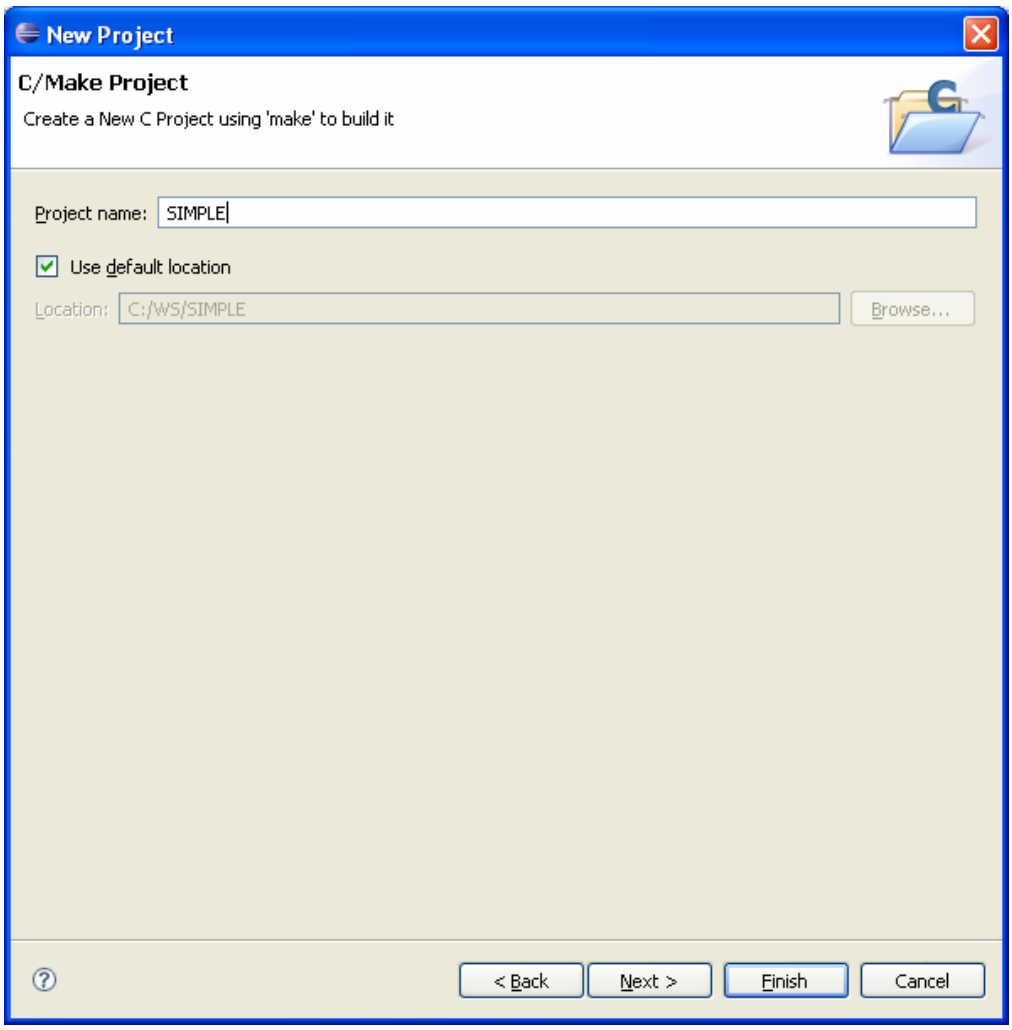


Создайте новый проект. Для этого выберите пункт меню File/New/Project

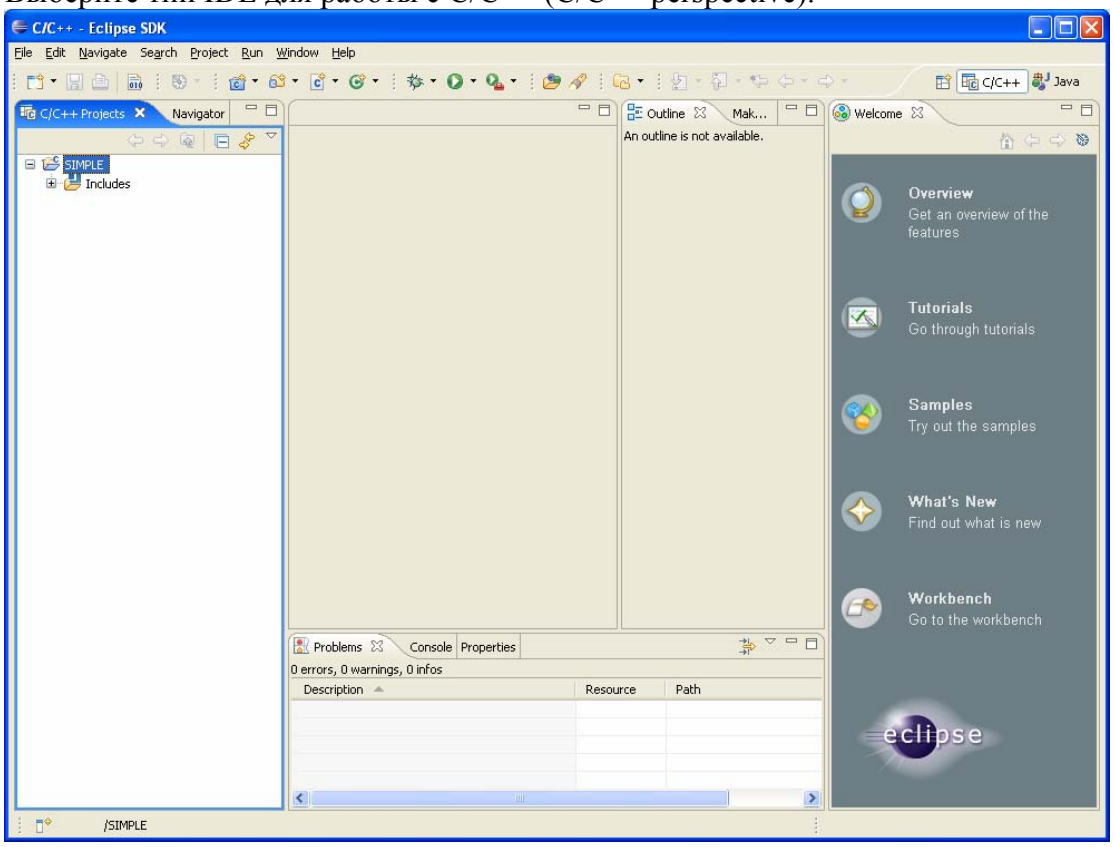


Выберите «Standard Make C Project».

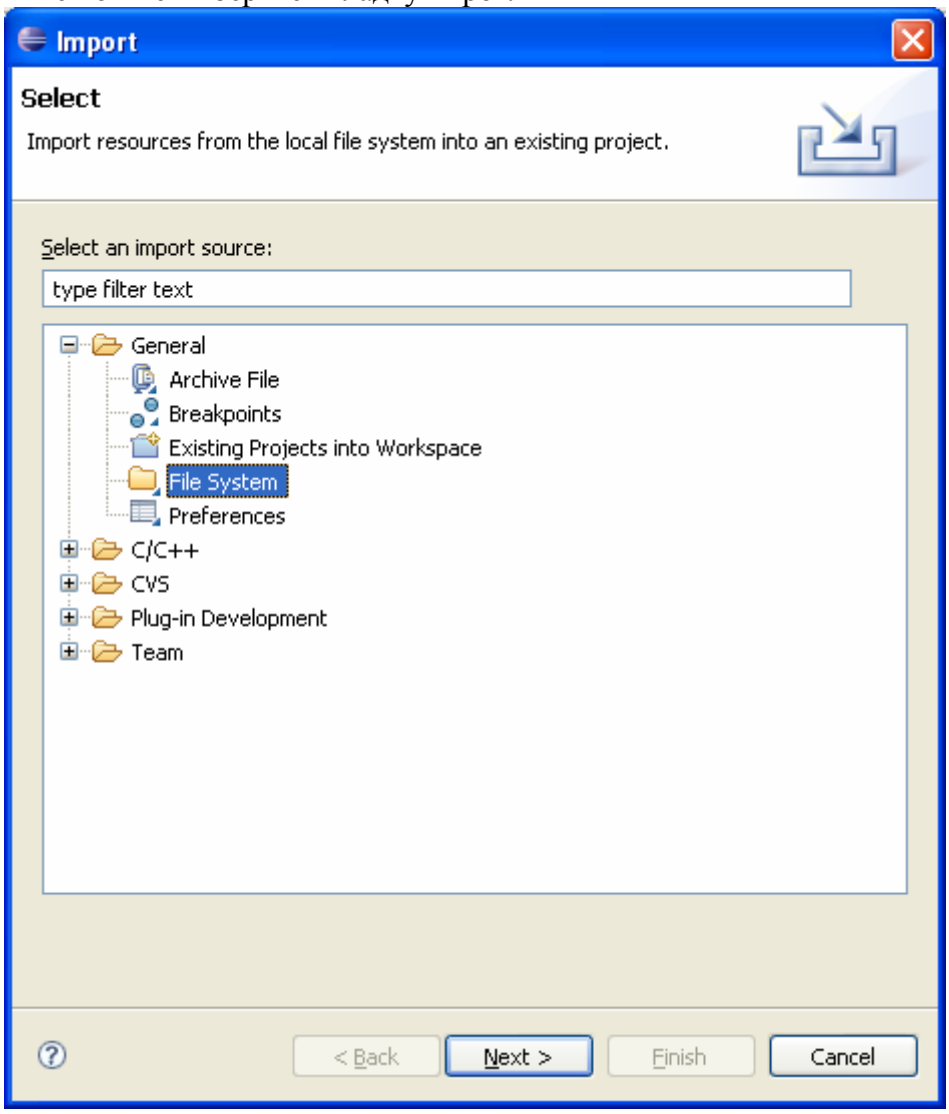
Укажите имя проекта (например, SIMPLE).



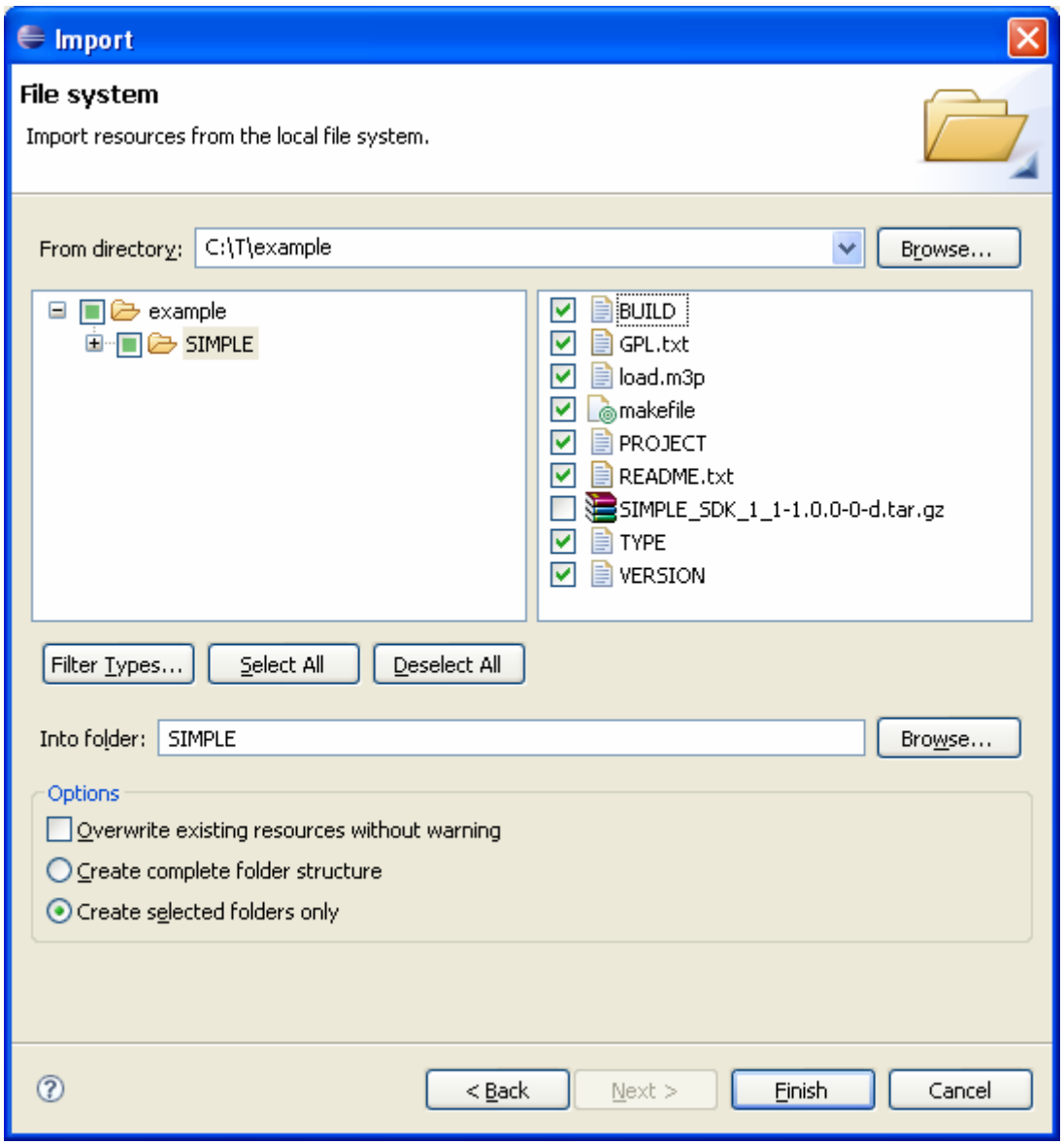
Выберите тип IDE для работы с C/C++ (C/C++ perspective).



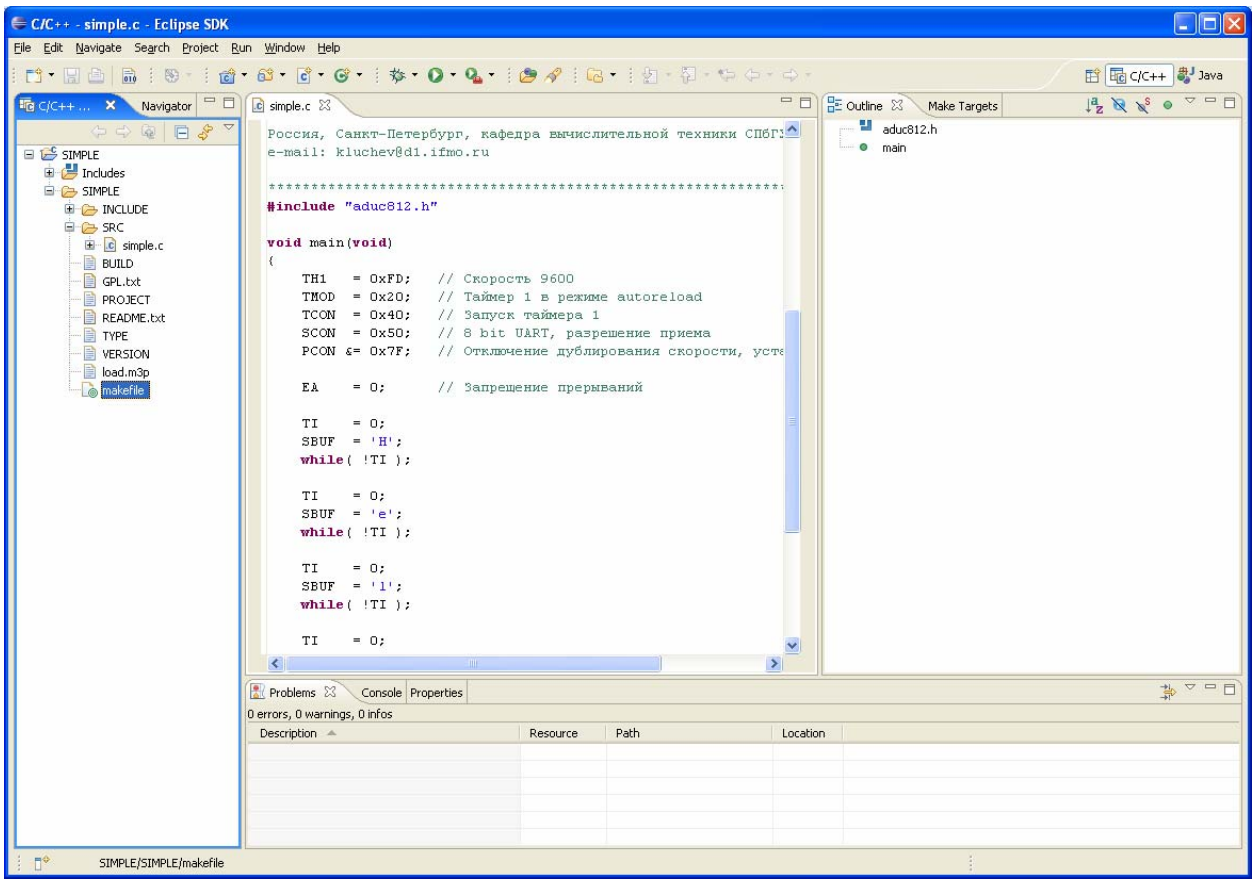
В меню File выберите вкладку import



Выберите файлы, которые собираетесь использовать в своем проекте.



После импорта проекта проект будет иметь следующий вид:



Программатор Flash для ADuC812 (DL)

Программное обеспечение (ПО) программирования Flash-памяти ADuC812 – это модифицированная версия программного обеспечения ADuC Downloader Version 2.07, свободно распространяемого фирмой Analog Devices. Изменение, внесенное в программу, заключалось в соответствующем конфигурировании COM-порта для обеспечения питания оптически развязанного от остальной части платы порта SDK-1.1.

При запуске программа dl.exe полностью стирает Flash-память ADuC812 и записывает туда образ программы, содержащейся в HEX-файле, имя которого указывается в командной строке. Протокол загрузки HEX-файла во Flash-память микроконтроллера детально описан в документе MicroConverter™ Technical Note – uC004, прилагаемом к настоящему руководству в виде pdf-файла.

Формат командной строки. Командная строка dl.exe имеет следующие параметры:

- /C:*X*, где X – номер COM-порта (1 или 2). Если параметр не указан, по умолчанию принимается 1.
- /F:*n.n*, где n.n – частота, на которой работает микроконтроллер, в МГц (по умолчанию 11.0592). Так как микроконтроллер ADuC812, установленный на стенде SDK-1.1, работает от кварца 11.0592 МГц, этот параметр можно не указывать.
- /D – указывает программе не стирать память данных (RAM). Этот параметр не является обязательным.
- /R или /R:*XXXX*, где XXXX – адрес в HEX-формате. Указывает загрузчику запустить передаваемую программу либо с определенного адреса, либо с адреса по умолчанию (FF00h). Этот параметр необходимо пропустить.
- *filename.hex* – единственный обязательный параметр: имя файла с программой в HEX-формате.

Замечание. Для загрузки программы во Flash-память необходимо установить перемычку JP1 на плате SDK-1.1 (находится слева от клавиатуры примерно на уровне клавиши «1»). После этого необходимо нажать кнопку сброса или включить питание (если стенд был отключен) и запустить программу dl.exe, указав необходимые параметры в командной строке. Затем необходимо снять перемычку и нажать кнопку сброса – записанная во Flash-память программа начнет работать.

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).

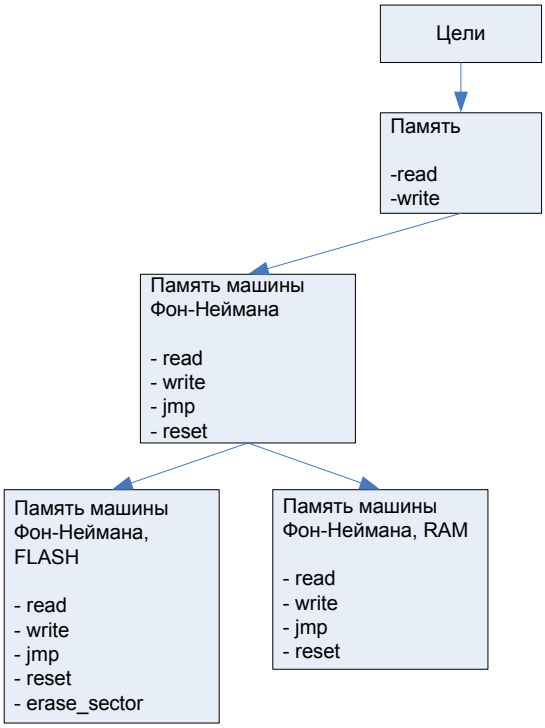


Рисунок 1 Иерархия целей

При необходимости, можно расширять и настраивать протокол для различных приложений. Можно предложить две основных стратегии развития РМ3Р:

- 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.

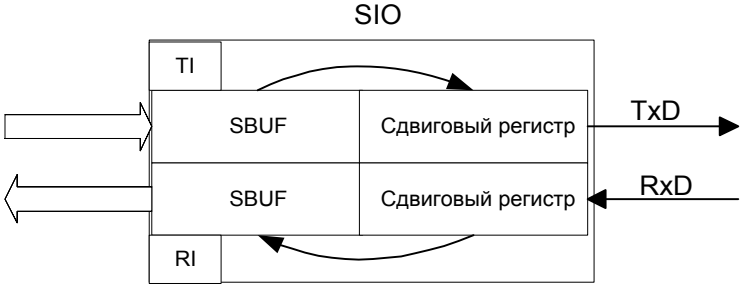
```
 terminateonerror
  9600 openchannel com1

: 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-1.1

Последовательный интерфейс

Последовательный порт в контроллерах MCS51 позволяет осуществлять последовательный дуплексный ввод-вывод в синхронном и асинхронном режимах с разными скоростями обмена. Помимо обычного ввода-вывода, в нем предусмотрена аппаратная поддержка взаимодействия нескольких микроконтроллеров.

Схематически контроллер порта можно представить так:



Как видно из рисунка, регистр SBUF, доступный пользователю для чтения и записи, есть на самом деле два регистра, в один из которых можно только записывать, а из другого – читать. Контроллер позволяет дуплексный обмен данными, т.е. одновременно может передавать и принимать информацию по линиям TxD (P3.1) и RxD (P3.0) соответственно.

Передача инициируется записью в SBUF байта данных. Этот байт переписывается в сдвиговый регистр, из которого пересылается бит за битом. Как только байт будет переслан целиком[1], устанавливается флаг TI, сигнализирующий о том, что контроллер готов к передаче очередного байта.

Прием осуществляется в обратном порядке: после начала процесса приема принятые биты данных последовательно сдвигаются в сдвиговом регистре, пока не будет принято их установленное количество, затем содержимое сдвигового регистра переписывается в SBUF и устанавливается флаг RI. После этого возможен прием следующей последовательности бит. Необходимо отметить, что запись принятого байта из сдвигового регистра в SBUF происходит только при условии, что RI=0, поэтому при заборе пользовательской программой очередного байта из SBUF ей необходимо сбрасывать этот флаг, иначе принятый в сдвиговый регистр, но не записанный в SBUF, следующий байт будет безвозвратно утерян.

Контроллер последовательного порта может работать в одном из четырех режимов (один синхронный и три асинхронных), различающихся скоростями обмена (бод[2]) и количеством передаваемых/принимаемых бит:

- Режим 0 (синхронный): данные передаются и принимаются через RxD, TxD является синхронизирующим (выдает импульсы сдвига). Скорость в этом режиме фиксирована (1/12 частоты тактового генератора).
- Режим 1 (8 бит данных, асинхронный, переменная скорость). Передаются 10 бит: старт-бит, 8 бит данных (SBUF) и стоп-бит.
- Режим 2 (9 бит данных, асинхронный, фиксированная скорость). Передаются 11 бит: старт-бит, 8 бит (SBUF), бит TB8/RB8 (посылка/прием соответственно) и 1 стоп-бит. Биты TB8 и RB8 содержатся в регистре SCON (биты 3 и 2 соответственно),

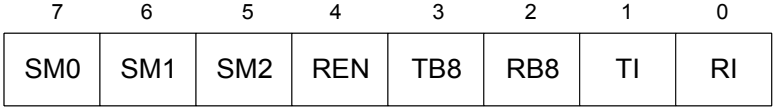
[1] В действительности, как только будет переслано установленное количество бит, зависящее от режима работы контроллера, о которых речь пойдет ниже.

[2] Линии TxD и RxD могут находиться всего в двух состояниях, поэтому скорость обмена в бодах соответствует скорости в бит/с.

первый устанавливается программно, а второй содержит 9-й бит принятой комбинации. С помощью них можно организовать, например, контроль четности или обмен с двумя стоп-битами.

- Режим 3 (9 бит данных, асинхронный, переменная скорость). То же, что и режим 2, только скорость обмена переменная (см. ниже).

Режим контроллера, а также другие параметры его работы задаются в регистре SCON (098h).



Бит	Описание
SM0 SM1 0 0 0 1 1 0 1 1	(Serial Mode) Режим 0 Режим 1 Режим 2 Режим 3
SM2	Включает поддержку взаимодействия нескольких ОКМЭВМ. Если SM2=1, то при приеме в режимах 2 и 3 RI не устанавливается в том случае, если принятый 9-й бит (RB8) равен 0. При приеме в режиме 1 RI не устанавливается, если не был принят правильный стоп-бит. В режиме 0 SM2 должен быть равен 0.
REN	(Receiver Enable) Если установлен, то разрешен прием данных. Устанавливается и сбрасывается программно.
TB8	(Transmitter Bit 8) 9-й бит посылаемых данных в режимах 2 и 3. Устанавливается и сбрасывается программно.
RB8	(Receiver Bit 8) 9-й бит принимаемых данных в режимах 2 и 3. В режиме 1, если SM2=0, в RB8 записывается принятый стоп-бит (точнее то, что было принято в момент приема стоп-бита). В режиме 0 не используется.
TI	(Transmitter Interrupt) Флаг завершения посылки. Устанавливается аппаратно по завершении посылки 8-го бита данных в режиме 0 или стоп-бита в других режимах. При ES = 1 (бит 4 регистра IEN0(0A8h)) происходит прерывание №4 (вектор 023h), когда TI устанавливается контроллером в 1 (подробнее об этом см. в следующем разделе). Должен быть сброшен программно.
RI	(Receiver Interrupt) Флаг завершения приема. Устанавливается аппаратно по завершении приема 8-го бита данных в режиме 0 или стоп-бита в других режимах. При ES = 1 (бит 4 регистра IEN0(0A8h)) происходит прерывание №4 (вектор 023h), когда RI устанавливается контроллером в 1 (подробнее об этом см. в следующем разделе). Должен быть сброшен программно.

Как было сказано выше, скорости обмена в разных режимах различаются. В двух из них скорости фиксированы, а в двух других переменные. Рассмотрим подробнее способы задания скорости обмена в каждом из режимов.

- Режим 0. Фиксированная скорость обмена. В этом режиме скорость вычисляется следующим образом:

$$baudrate = \frac{Core\_Clk}{12}$$, где $Core\_Clk$ есть внутренняя тактовая частота МП.

- Режим 2. Скорость обмена зависит от значения бита SMOD (старший бит в регистре PCON(087h)). Если SMOD=0 (по умолчанию), то скорость определяется как 1/64 тактовой частоты. SMOD=1 удваивает это значение. В общем случае:

$$baudrate = \frac{2^{SMOD} \times f_{osc}}{64}.$$

- Режимы 1 и 3. В этих режимах порт можно синхронизировать от двух источников: от встроенного генератора импульсов и от таймера 1. Источник синхронизации определяется значением бита BD (старший бит регистра

ADCON(0D8h)): если BD=1, то используется встроенный генератор импульсов, в противном случае используется таймер 1.
Встроенный генератор импульсов делит тактовую частоту МП на 2500 и, если SMOD=1 (см. выше), то полученное значение удваивается. Например, при тактовой частоте МП 12MHz можно получить стандартные скорости 12MHz/2500 = 4800 бод и (12MHz/2500)*2 = 9600 бод. В общем случае: $baudrate = \frac{2^{SMOD} \times f_{osc}}{2500}$.

При использовании таймера 1 скорость обмена определяется временем переполнения таймера и значением бита SMOD:

$$baudrate = \frac{2^{\mathrm{SMOD}} \times \mathrm{TM1OVrate}}{32},$$

где TM1OVrate есть время переполнения таймера 1, зависящее от режима его работы. Таймер 1 может быть установлен как «таймер» или как «счетчик», в любом режиме работы. Прерывание от таймера при этом обычно запрещается. Чаще всего таймер 1 устанавливают как «таймер» в режиме 2 («auto-reload»): для этого старшая тетрада регистра TMOD(089h) должна равняться 0010b. При таком использовании после каждого переполнения регистра таймера TL1 в него загружается значение из регистра TH1. Скорость обмена в этом случае вычисляется по формуле:

$$baudrate = \frac{2^{SMOD}}{32} \times \frac{f_{osc}}{12 \times (256 - \mathrm{TH1})}.$$

В следующей таблице приведены способы задания некоторых стандартных скоростей обмена.

Скорость	f_{osc}, MHz	SMOD	Таймер 1		
			C/T[3]	Режим	TH1
62.5 Кбод	12.0	1	0	2	FF
19200 бод	11.059	1	0	2	FD
9600 бод	11.059	0	0	2	FD
4800 бод	11.059	0	0	2	FA
2400 бод	11.059	0	0	2	F4
1200 бод	11.059	0	0	2	E8
110 бод	6.0	0	0	2	72
110 бод	12.0	0	0	1	FEEB[4]

Работа с последовательным каналом по опросу

Простейшим из способов организации последовательного обмена является управление им из прикладной программы. То есть, если требуется переслать байт, то: 1) сбрасывается TI; 2) в SBUF записывается нужный байт данных; и 3) ожидается, пока TI не будет установлен контроллером. С приемом здесь сложнее: нужно постоянно проверять флаг RI и, если он установлен, то читать принятый байт из SBUF и сбрасывать RI. Такой способ удобен, когда разработчику четко известно, когда произойдет прием данных и когда его завершать. Неудобен он тем, что время выполнения самой программы напрямую зависит от скорости обмена. В самом деле, чем медленнее скорость, тем дольше по времени цикл ожидания готовности к приему/посылке следующего байта.

[3] Бит 6 регистра TMOD

[4] В данном случае работа таймера происходит в режиме 1 (16bit) и значение FEEBh загружается из обработчика прерывания программно: FEh в TH1, EBh в TL1. Так можно достичь очень низких скоростей даже при большой тактовой частоте МП.

Описание драйвера

void init_sio(unsigned char speed)

Инициализирует последовательный канал на заданной скорости. Использует таймер 1

Вход: char speed - скорость. Задается константами, описанными в заголовочном файле sio.h
bit sdouble - дублирование скорости: 0 - не дублировать скорость, заданную аргументом speed; 1 - дублировать.
Выход: нет
Результат: нет

unsigned char rsiostat(void)

Возвращает ненулевое значение, если буфер приема не пуст

Вход: нет
Выход: нет
Результат: 0 - буфер приема пуст;
1 - был принят символ

void wsio(unsigned char c)

Отправляет символ по последовательному каналу

Вход: unsigned char c - символ, который нужно отправить
Выход: нет
Результат: нет

unsigned char rsio(void)

Дожидается приема символа из последовательного канала и возвращает его.

Вход: нет
Выход: нет
Результат: принятый символ

void type(char * str)

Выводит ASCIIZ-строку в последовательный канал

Вход: char *str - указатель на строку
Выход: нет
Результат: нет

Управление светодиодными индикаторами

Таблица 1. Регистр управления светодиодами

Адрес	Регистр	Доступ	Назначение
080007H	SV	W	Регистр управления светодиодами.

Описание драйвера

void led(unsigned char n, unsigned char on)

Управление одним светодиодом

Вход: n - порядковый номер светодиода (от 0 до 7)
on - 1 - зажигает, 0 гасит светодиод
Выход: нет
Результат: нет
Описание: Производится доступ к регистру расширителя портов SV с помощью функции write_max. Состояние светодиодов хранится в регистре old_led.

void leds(unsigned char on)

Зажигание линейки светодиодов

Вход: on - управление светодиодами. Каждый бит переменной отвечает за один светодиод: 1 - зажигает, 0 гасит светодиод
Выход: нет
Результат: нет
Описание: Производится доступ к регистру расширителя портов SV с помощью функции write_max. Состояние светодиодов хранится в регистре old_led.

Регистры ПЛИС

Таблица 2. Перечень регистров ПЛИС.

Адрес	Регистр	Доступ	Назначение
080000H	KB	R/W	Регистр клавиатуры.
080001H	DATA_IND	R/W	Регистр шины данных ЖКИ.
080002H	EXT_LO	R/W	Регистр данных параллельного порта (разряды 0..7).
080003H	EXT_HI	R/W	Регистр данных параллельного порта (разряды 8..15).
080004H	ENA	W	Регистр управления портами ввода-вывода, звуком, сигналом INT0 и прерыванием от клавиатуры.
080006H	C_IND	W	Регистр управления ЖКИ.
080007H	SV	W	Регистр управления светодиодами.

Регистр клавиатуры KB

Адрес 080000H. Значение после сброса xxxx1111B (полная конфигурация) и xxxx0000B (упрощенная конфигурация).

7	6	5	4	3	2	1	0
R	R	R	R	R/W	R/W	R/W	R/W
ROW				COL			

Рисунок 2. Регистр клавиатуры KB.

Таблица 3. Назначение битов регистра KB.

Биты	Поле	Описание
0..3	COL	Поле предназначено для сканирования клавиатуры (колонки матрицы). Сканирование производится посредством записи логического «0» в один из

		разрядов поля. В полной конфигурации ПЛИС из этой тетрады считывается записанное ранее в нее значение. В упрощенной конфигурации всегда возвращается «0».
4..7	ROW	Поле предназначено для считывания данных с клавиатурной матрицы (строки). Если ни одна из кнопок в строке не нажата, все биты поля ROW содержат логические «1». Если кнопка нажата и на ее колонку подан логический «0», то в поле ROW также появится логический «0».

Данный регистр можно использовать для определения типа конфигурации. Если при старте программы записать в тетраду COL регистра KB ненулевое значение, а затем его считать, то в полной конфигурации будет считано записанное значение, а в упрощенной — нули.

Регистр шины данных ЖКИ DATA_IND

Адрес 080001H. Значение после сброса 00000000B.

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
D7	D6	D5	D4	D3	D2	D1	D0

Рисунок 3. Регистр шины данных ЖКИ DATA_IND.

Таблица 4. Назначение битов регистра DATA_IND.

Биты	Поле	Описание
0..7	D0..D7	Регистр DATA_IND позволяет устанавливать данные на шине данных ЖКИ и считывать их оттуда. Для организации взаимодействия с ЖКИ (формирования временных диаграмм чтения и записи) необходимо использование регистра C_IND.

Регистр данных параллельного порта EXT_LO

Адрес 080002H. Значение после сброса 00000000B.

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
D7	D6	D5	D4	D3	D2	D1	D0

Рисунок 4. Регистр данных параллельного порта EXT_LO.

Таблица 5. Назначение битов регистра EXT_LO.

Биты	Поле	Описание
0..7	D0..D7	Регистр EXT_LO позволяет считывать и записывать биты 0..7 параллельного порта. Для того, чтобы данные из регистра попали на выход, необходимо установить бит EN_LO в логическую «1» (см. регистр ENA). Для чтения данных необходимо установить этот бит в логический «0».

Регистр данных параллельного порта EXT_HI

Адрес 080003H. Значение после сброса 00000000B.

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
D7	D6	D5	D4	D3	D2	D1	D0

Рисунок 5. Регистр данных параллельного порта EXT_HI.

Таблица 6. Назначение битов регистра EXT_HI.

Биты	Поле	Описание
0..7	D0..D7	Регистр EXT_HI позволяет считывать и записывать биты 8..15 параллельного порта. Для того, чтобы данные из регистра попали на выход, необходимо

		установить бит EN_LO или EN_HI в логическую «1» (см. регистр ENA). Для чтения данных необходимо установить этот бит в логический «0».

Регистр управления ENA

Адрес 080004H. Значение после сброса x0100000B.

7	6	5	4	3	2	1	0
-	W	W	W	W	W	W	W
-	KB	INT0	SND2	SND1	SND0	EN_HI	EN_LO

Рисунок 6. Регистр управления ENA.

Таблица 7. Назначение битов регистра ENA.

Биты	Поле	Описание
0	EN_LO	В полной конфигурации бит EN_LO нужен для управления младшими 8 разрядами (биты 0..7) 16-разрядного порта ввода-вывода. Если записать в EN_LO логический «0», то порт ввода-вывода переводится в Z-состояние и появляется возможность чтения данных из EXT_LO. При записи в данный бит логической «1» порт переключается на вывод и данные, записанные в регистр EXT_LO, попадают на выход порта ввода-вывода. В упрощенной конфигурации этот бит управляет всеми 16 разрядами порта ввода-вывода. Если записать в EN_LO логический «0», то весь порт ввода-вывода переводится в Z-состояние и появляется возможность чтения данных из регистров EXT_LO и EXT_HI. При записи в данный бит логической «1» порт переключается на вывод и данные, записанные в регистры EXT_LO и EXT_HI, попадают на выход порта ввода-вывода.
1	EN_HI	В полной конфигурации бит EN_HI нужен для управления старшими 8 разрядами (биты 8..15) 16-разрядного порта ввода-вывода. Если записать в EN_HI логический «0», то порт ввода-вывода переводится в Z-состояние и появляется возможность чтения данных из EXT_HI. При записи в данный бит логической «1» порт переключается на вывод и данные, записанные в регистр EXT_HI, попадают на выход порта ввода-вывода. В упрощенной конфигурации бит EN_HI не влияет на функционирование порта ввода-вывода. Все управление портом производится битом EN_LO.
2..4	SND0-SND2	Выход звукового ЦАП. Задает уровень напряжения на динамике. Позволяет формировать звуковые сигналы различной тональности и громкости.
5	INT0	При записи логического «0» в этот бит на вход INT0 ADuC812 также попадает логический «0». Бит можно использовать для формирования внешнего прерывания для микроконтроллера.
6	KB	В полной конфигурации при записи логического «0» прерывание от клавиатуры запрещается. Если бит установлен в «1», то прерывание от клавиатуры разрешено. В упрощенной конфигурации бит KB всегда равен нулю, т.е. прерывание клавиатуры запрещено.

Регистр управления ЖКИ C_IND

Адрес 080006H. Значение после сброса xxxxx010B.

7	6	5	4	3	2	1	0
-	-	-	-	-	W	W	W
-	-	-	-	-	RS	RW	E

Рисунок 7. Регистр управления ЖКИ C_IND.

Таблица 8. Назначение битов регистра C_IND.

Биты	Поле	Описание
0	E	Бит управления входом «E» ЖКИ. Наличие положительного импульса на входе «E» позволяет зафиксировать данные на шине ЖКИ (данные, сигналы RW и RS к этому моменту должны быть уже установлены).
1	RW	Бит переключения шины данных ЖКИ на чтение или запись. 0 – запись, 1 –

		чтение.
2	RS	Бит переключения режимов команды/данные ЖКИ. 1 – данные, 0 – команды.

Регистр управления светодиодами SV

Адрес 080007H. Значение после сброса 00000000B.

7	6	5	4	3	2	1	0
W	W	W	W	W	W	W	W
D7	D6	D5	D4	D3	D2	D1	D0

Рисунок 8. Регистр управления светодиодами SV.

Таблица 9. Назначение битов регистра SV.

Биты	Поле	Описание
0..7	D0..D7	Биты управления светодиодами. Подача логической «1» зажигает светодиоды.

Драйвер доступа к регистрам ПЛИС

Для доступа к регистрам ПЛИС нужно переключить страничный регистр DPP на 8 страницу памяти. Адреса регистров внутри страницы находятся в диапазоне от 0 до 7. Доступ к регистрам возможен через указатель: `unsigned char xdata *regnum`
Ниже приведен пример функций для доступа к регистрам ПЛИС.

```
#define MAXBASE     8      // Страница памяти, в которую отображаются регистры ПЛИС

/*
WriteMAX      - Запись байта в регистр ПЛИС
regnum        - адрес регистра ПЛИС
val                - записываемое значение
результат     - нет
*/

void WriteMax (unsigned char xdata *regnum, unsigned char val)
{
   unsigned char oldDPP = DPP;

   DPP = MAXBASE;
   *regnum = val;
   DPP = oldDPP;
}

/*
ReadMAX       - Чтение байта из регистра ПЛИС
regnum        - адрес регистра ПЛИС
результат     - прочитанное значение
*/

unsigned char ReadMax(unsigned char xdata *regnum)
{
   unsigned char oldDPP = DPP;
   unsigned char val = 0;

   DPP = MAXBASE;
   val = *regnum;
   DPP = oldDPP;
   return val;
}
```

Драйвер дискретных портов

unsigned char get_din (unsigned char n)

Чтение дискретного порта ввода

Вход: n - порядковый номер дискретного порта (от 0 до 15)
Выход: нет
Результат: нет

unsigned short get_dins (void)

Чтение дискретных портов EXT_HI, EXT_LO

Вход: нет

Выход: нет
Результат: возвращает значение дискретных портов EXT_LO и EXT_HI

void set_dout (unsigned short value)

запись дискретных портов EXT_HI, EXT_LO

Вход: value - 16-ти разрядное значение портов EXT_HI, EXT_LO

Выход: нет
Результат: нет

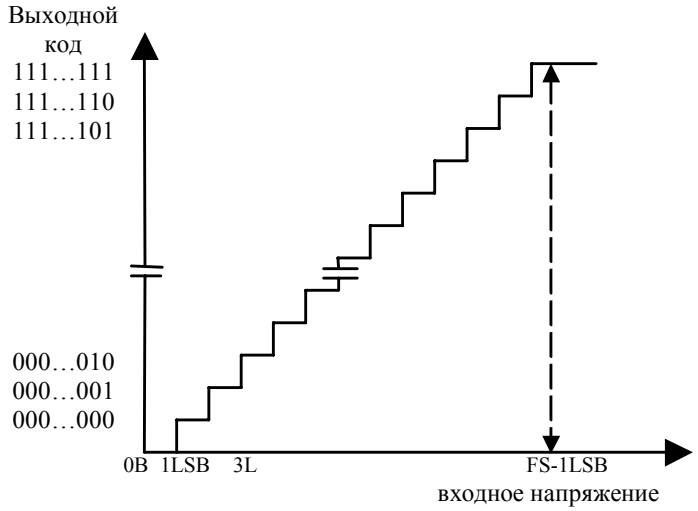
Проблемы, часто возникающие при доступе к регистрам ПЛИС

Необходимо помнить, что при переключении страниц становятся недоступными все данные, размещенные в странице 0.
Для того, чтобы избежать проблем со страничным регистром DPP, нужно использовать специальные функции для доступа к ПЛИС, которые перед началом работы с регистрами ПЛИС будут запоминать старое значение страничного регистра, а по окончании работы возвращать его обратно.
Нужно следить, чтобы передаваемые в регистры ПЛИС значения хранились во внутренней памяти микроконтроллера (DATA, IDATA). Убедиться, что передаваемая информация не содержится во внешней памяти контроллера (XDATA), достаточно легко: так как для доступа к внешней памяти в микроконтроллерах семейства C51 используется регистр DPTR, нужно просто просмотреть листинг программы и убедиться в том, что для доступа к переменным компилятор не использует DPTR.

Аналого-цифровой преобразователь

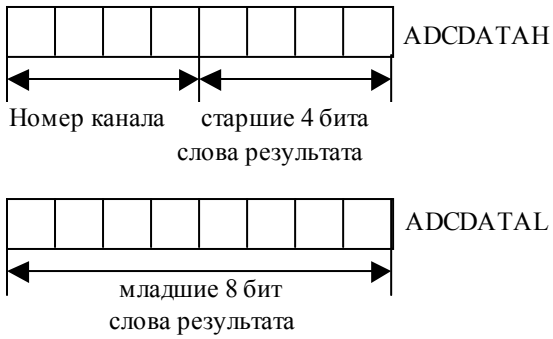
Передаточная функция АЦП

Диапазон входных напряжений АЦП составляет от 0 до $+V_{REF}$. Для этого диапазона смена кодов происходит посередине соответствующего кванта (т.е. 1.2 LSB, 3/2 LSB, 5/2 LSB, …, FS –3/2 LSB). Выходной код – прямая в двоичном коде с 1LSB=FS/4096 или 2.5/4096=0.61 Мв при V_{REF} =2.5 В. Идеальная передаточная функция показана на рисунке.



SFR регистры АЦП

Управление и настройка АЦП осуществляется при помощи 3 SFR регистров. Результат преобразования в 12 битном формате записывается ADCDATAH/L. В первые четыре бита регистра ADCDATAH записывается биты выбора канала. Формат слова результата следующий:



ADCCON1 (адрес SFR EFh) – регистр управляет преобразованием, временем переключения, режимом преобразования, токопотреблением устройства.

7-6	MD1 MD0	Два бита выбирают режима АЦП MD1 MD0 Режим 0 0 дежурный 0 1 нормальный 1 0 дежурный, если не выполняется цикл преобразования

		1 1 холостой, если не выполняется цикл преобразования
5-4	CLK1 CLK0	Биты деления тактовой частоты. Устанавливают коэффициент деления основной частоты, используемой для генерирования частоты работы АЦП CLK1 CLK0 Делитель 0 0 1 0 1 2 1 0 4 1 1 8
3-2	AQ1 AQ0	Биты задержки переключения выбирают время, необходимое для перезарядки УВХ при переключении мультиплексора. AQ1 AQ0 Число тактов задержки 0 0 1 0 1 2 1 0 3 1 1 4
1	T2C	Бит запуска преобразования от Таймера 2. Если бит установлен, то сигнал переполнения Таймера 2 используется для АЦП
0	EXC	Бит разрешения внешнего запуска. Если бит установлен, то разрешен внешний запуск от низкого уровня на контакте 23 (CONVST). Активный низкий уровень должен поддерживаться на контакте не менее 100 нс

ADCCON2 (адрес SFR D8h) – регистр управляет режимом преобразования и выбором канала.

7	ADCI	Бит прерывания АЦП. Устанавливается аппаратно в конце каждого цикла преобразования или в конце передачи блока данных в режиме прямого доступа к памяти. Сбрасывается аппаратно после перехода процессора на процедуру обработки прерывания.
6	DMA	Бит разрешения прямого доступа к памяти. Устанавливается пользователем для начала операции передачи в режиме прямого доступа к памяти.
5	CCONV	Бит разрешения циклического преобразования**.** В этом режиме АЦП после того как преобразование закончилось, начинает следующие преобразование. Параметры работы АЦП должны быть заранее сконфигурированы.
4	SCONV	Бит однократного преобразования Устанавливается пользователем для инициализации однократного цикла преобразования. После того, как цикл завершился, этот бит автоматически сбрасывается в 0

3-0	CS3 CS2 CS1 CS0	Биты выбора канала Позволяют пользователю программно выбирать номер канала, для которого будет осуществляться преобразование. Для режима прямого доступа номер канала будет зависеть от идентификатора во внешней памяти. CS3 CS2 CS1 CS0 Номера входных каналов 0 0 0 0 0 0 0 0 1 1 0 0 1 0 2 0 0 1 1 3 0 1 0 0 4 0 1 0 1 5 0 1 1 0 6 0 1 1 1 7 1 0 0 0 Температурный сенсор 1 1 1 1 Остановка КПД (DMA)

ADCCON3 (адрес SFR F5h) – регистр дает пользователю доступ к флагу занятости АЦП.

7	BUSY	Флаг занятости ЦАП (только для чтения). Устанавливается аппаратно на время цикла преобразования или калибровки. Автоматически сбрасывается ядром в конце преобразования или калибровки.
6-0		Биты зарезервированы для дальнейшего использования

Частота тактирования

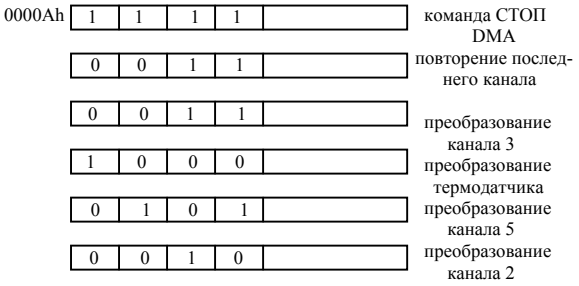
Для корректной работы АЦП частота тактирования должна находится между 400кГц и 4Гц, а оптимальными являются частоты от 400кГц до 3МГц. Время преобразования составляет 15 тактовых интервалов плюс 1 интервал для синхронизации, плюс задержка переключения. Например, если задержка переключения 4 тактовых интервала, то время одного преобразования будет составлять 20 тактовых интервалов. Таким образом, итоговая частота работы АЦП может быть вычислена по следующей формуле:
Частота=част. такт. АЦП/((16+задержка переключения)*делитель чстоты).

Режимы работы

Встроенный АЦП сконструирован таким образом, что может осуществлять выборки каждые 5 мс (частота 200КГц). Таким образом, процессору за 5мс необходимо прочитать значение выборки и записать его во внешнюю память, если он не успеет этого сделать, то значение выборки будет потеряно. Если управление осуществляется прерыванием, то процессору еще надо будет перейти на процедуру обработки прерывания. Для приложений, в которых процессор не может поддерживать такую высокую скорость обработки прерываний, и предназначен режим прямого доступа к памяти (DMA- Direct Memory Access).
Для установки этого режима надо выставить 6 бит (DMA) в регистре ADCCON2. Это приведет к тому, что результаты преобразования будут записываться во внешнюю статическую память, минуя ядро процессора.
Для работы в этом режиме необходимо произвести следующие настройки:

1. АЦП переводится в дежурный режим сбросом старших двух бит регистра ADCCON1.

2. Устанавливается адрес начала области памяти, куда будут записываться результаты работы АЦП. Для этого адрес начала области памяти записывается в регистры DMAL, DMAH, DMAP. При этом сначала заполняется DMAL, затем DMAH, а затем DMAP.
3. Размечается внешняя память. В старшие четыре бита каждого второго байта внешней памяти, начиная с адреса, записанного в регистры DMA, записывается идентификатор канала. Поскольку в этом режиме АЦП не завис от ядра, необходимо завершить разметку памяти стоповой командой. Это делается дублированием идентификатора последнего преобразуемого канала, за которым в следующем слове следует стоповая последовательность 1111. Типичная последовательность разметки памяти:



4. DMA инициализируется записью в SFR регистры АЦП следующей последовательности
   а. В регистре ADCCON1 устанавливается бит разрешения режима DMA.
   б. Производятся настройки в регистре ADCCON1, устанавливается тактирование от Таймера 2 или от внешнего триггера.
   с. Инициируется преобразование. Это делается началом единичного/циклического преобразования (АЦП переводится в рабочий режим) и запуском Таймера 2 или подачей сигналов на внешний триггер.
   Когда DMA преобразование завершается, аппаратно устанавливается флаг прерывания, а внешняя SRAM содержит результаты преобразования. Необходимо отметить, что в последние два слова размеченной памяти (содержащие стоп команду и дубликат последнего преобразуемого канала) не записывается никаких результатов.

Описание драйвера

void init_adc(void)

Инициализация АЦП.

Вход: нет
Выход: нет
Результат: нет
Описание:

float get_voltage(unsigned char channel)

Получение значения напряжения на одном из входов АЦП SDK-1.1

Вход: channel - номер входа (канала) АЦП
Выход: нет
Результат: значение напряжения в вольтах.

Описание: Производится запуск одиночного преобразования напряжения на одном из каналов АЦП.

void init_dac(bit mode)

Инициализация ЦАП.

Вход: mode - режим (_8BIT/_12BIT)
Выход: нет
Результат: нет
Описание: Прекращает работу ЦАП, устанавливает режим (8-битный/12-битный), обнуляет регистры данных DACxH,DACxL. Затем включает оба канала.

void switch_dac(bit channel, bit state)

Включение/выключение канала ЦАП.

Вход: channel - номер канала (DAC0/DAC1)
state - включить (ON) или выключить (OFF)
Выход: нет
Результат: нет

void set_voltage(float v, bit channel)

Подача нужного напряжения на один из каналов ЦАП.

Вход: v - значение напряжения в вольтах
channel - номер канала (DAC0/DAC1)
Выход: нет
Результат: нет
Описание: Подает напряжение на выбранный канал ЦАП. Предполагается, что канал включен.