Проектирование схем микроэлектронных устройств в Proteus с использованием внешней памяти.
Часть 1

Татьяна Колесникова (beluikluk@gmail.com)

В статье подробно описана подготовка карты памяти (форматирование и создание образа) для управления ею через микроконтроллер в Proteus. Приведены примеры моделирования схем, имитирующих подключение внешней памяти MMC (MultiMediaCard) к микроконтроллеру ATmega32, компиляция программы инициализации которого выполнена в CodeVisionAVR. Описан программный способ создания файлов, записи информации во внешнюю память, её чтения и отображения на экране терминала и буквенно-цифрового дисплея.

Введение

Карты памяти применяют в качестве носителя информации в таких устройствах, как смартфоны, цифровые фотоаппараты, видеокамеры, персональные компьютеры и др. Они подходят для расширения памяти в электронных системах и создания интерфейса обмена информацией через шину SPI, которая присутствует во многих микроконтроллерах, в частности и в микроконтроллерах AVR семейства Mega. Протокол SPI позволяет вести обмен данными на высокой скорости, задействовав при этом минимальное количество выводов микроконтроллера.

При проектировании устройства обмена информацией с картой памяти работающего под управлением микроконтроллера AVR написание программы инициализации и её компиляцию удобно выполнить с помощью CodeVisionAVR 3.12 (интегрированной среды разработки программного обеспечения для микроконтроллеров семейства AVR фирмы Atmel, которая имеет в своём составе компилятор языка С для AVR). CodeVisionAVR поддерживает все базовые конструкции языка С, которые используются при написании программ (алфавит, константы, идентификаторы, комментарии) и разрешены архитектурой AVR, с некоторыми добавленными характеристиками, реализующими преимущество специфики архитектуры AVR. Используя специальные директивы, в любом месте программы можно включить ассемблерный код. В CodeVisionAVR имеется набор команд управления буквенно-цифровыми и графическими дисплеями, а также библиотеки функций работы с файлами. Программные средства позволяют напрямую обращаться к регистрам микроконтроллера и управлять состоянием линий портов.

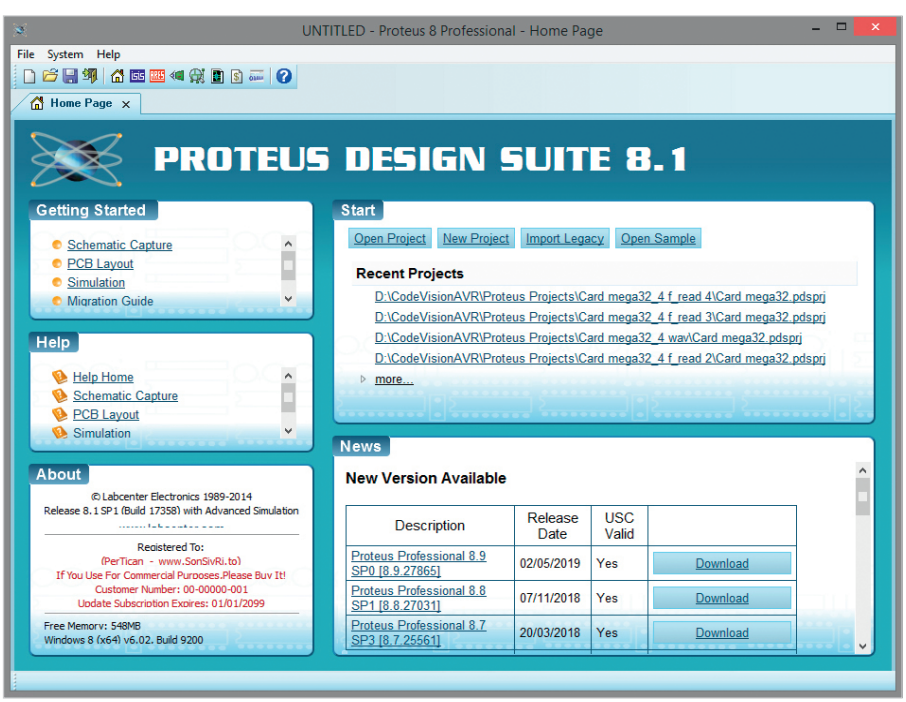

Рис. 1. Стартовое окно программы Proteus

Проектирование схемы электрической принципиальной в Proteus

Для проектирования устройства обмена данными и моделирования его работы удобно использовать программную среду Proteus, библиотека компонентов которой содержит как аналоговые, так и цифровые компоненты, а также устройства вывода информации и микроконтроллеры с возможностью их программирования.

Если написание программного кода управления электронной системой предполагается выполнить в CodeVisionAVR, то проект схемы электрической принципиальной, в котором присутствует карта памяти, буквенно-цифровой дисплей, терминал и микроконтроллер создают без использования мастера – при помощи кнопки ISIS (Schematic Capture) верхней панели инструментов Proteus (см. рис. 1). В результате будет открыта новая вкладка «Schematic Capture», в рабочем поле которой и будет выполняться разработка схемы. В нашем примере вывод считанной с карты памяти информации (содержимого текстового файла) выполним на экран терминала и буквенно-цифрового дисплея, для чего добавим эти устройства в рабочую область схемного редактора.

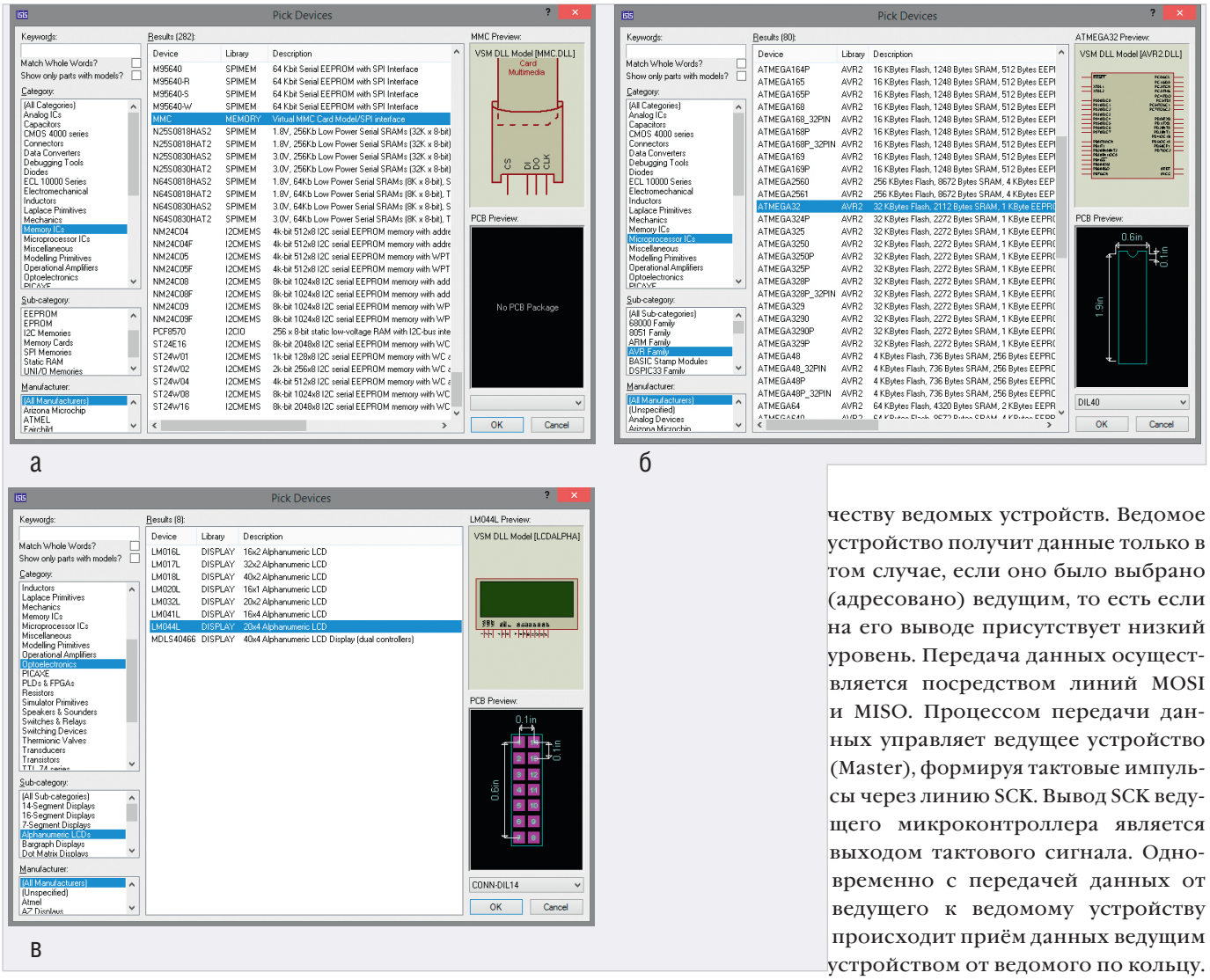

Рис. 2. Раздел: Memory Cards библиотеки Memory ICs (а), AVR Family библиотеки Microprocessor ICs (б), Alphanumeric LCDs библиотеки Optoelectronics (в)

В Proteus внешний накопитель представлен картой формата MMC, которая находится в разделе «Memory Cards» библиотеки «Memory ICs» (см. рис. 2а). Выводы карты MMC имеют следующее назначение: CS – выбор карты, DI – вход данных для записи в карту, DO – выход данных для чтения из карты, CLK – синхроимпульсы шины SPI.

Рассмотрим работу с картой памяти на примере её подключения к 8-битному микроконтроллеру ATmega32, который имеет следующие аппаратные характеристики: Flash-память программ объёмом 32 Кбайт, ОЗУ объёмом 2 Кбайт, EEPROM-память данных объёмом 1 Кбайт, количество контактов ввода/вывода – 32, тактовая частота 0–16 МГц, два 8-битных (T0, T2) и один 16-битный таймер/счётчик, интерфейсные модули USART, SPI, TWI, 8-канальный 10-битный АЦП, интерфейс JTAG. В Proteus микросхема ATmega32 находится в разделе «AVR Family» библиотеки «Microprocessor ICs» (см. рис. 2б).

При обмене данными по интерфейсу SPI микроконтроллер AVR может работать как ведущий (режим Master) либо как ведомый (режим Slave). Связь между устройствами осуществляется с помощью следующих линий портов ввода/вывода общего назначения микроконтроллера:

- MOSI – выход данных для ведущего или вход данных для ведомого устройства;
- MISO – вход данных для ведущего или выход данных для ведомого устройства;
- SCK – сигнал общей синхронизации интерфейса;
- $\overline{SS}$ – выбор ведомого устройства.

Ведущее устройство формирует один или несколько сигналов SS (slave select) для выбора ведомых устройств. При этом количество формируемых сигналов соответствует количеству ведомых устройств. Ведомое устройство получит данные только в том случае, если оно было выбрано (адресовано) ведущим, то есть если на его выводе присутствует низкий уровень. Передача данных осуществляется посредством линий MOSI и MISO. Процессом передачи данных управляет ведущее устройство (Master), формируя тактовые импульсы через линию SCK. Вывод SCK ведущего микроконтроллера является выходом тактового сигнала. Одновременно с передачей данных от ведущего к ведомому устройству происходит приём данных ведущим устройством от ведомого по кольцу. Таким образом, за один полный цикл сдвига всех разрядов регистра происходит обмен данными между двумя устройствами.

Выбор компонентов из базы данных для последующего их размещения в рабочей области программы выполняют в окне «Pick Devices», которое открывают командой контекстного меню Place/Component/From Libraries или нажатием кнопки P на панели «DEVICES» (по умолчанию панель расположена в левой части программы и содержит список имеющихся в проекте компонентов). Открывают панель «DEVICES» нажатием кнопки «Component Mode» на левой панели инструментов редактора ISIS.

Для добавления микросхемы микроконтроллера в рабочее поле проекта в левой верхней части окна «Pick Devices» в поле «Category» щелчком левой кнопки мыши выбирают из списка библиотеку «Microprocessor ICs». Пакет «Microprocessor ICs» позволяет включать в эмуляцию смешанной схемы

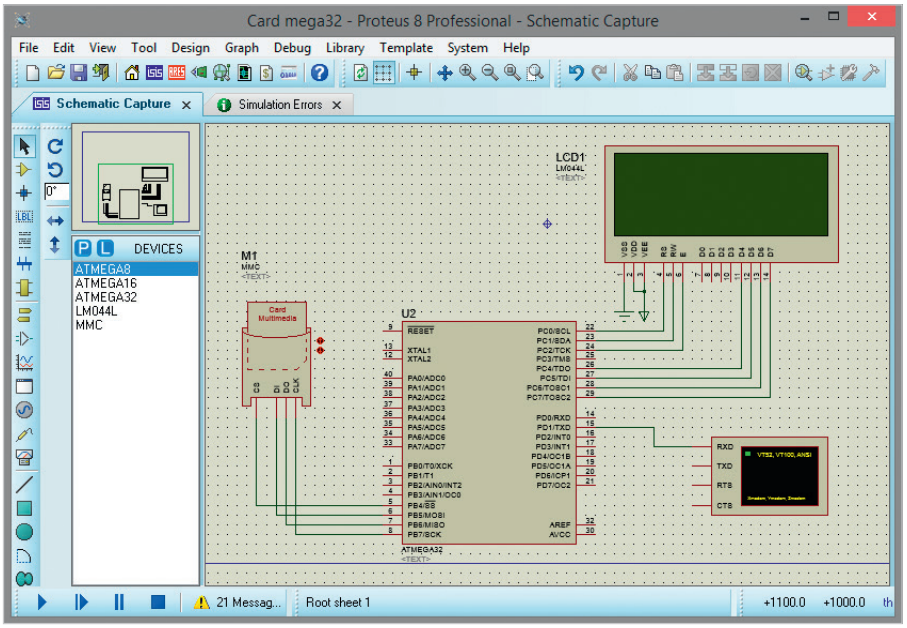

Рис. 3. Сопряжение микроконтроллера ATmega32 с картой памяти и устройствами вывода информации в рабочей области редактора ISIS программы Proteus

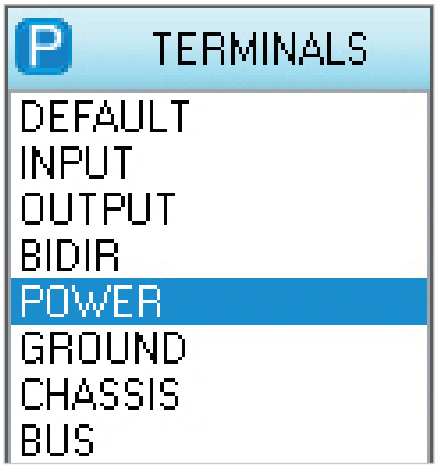

Рис. 4. Панель TERMINALS

определённые микроконтроллеры с возможностью написания и отладки программного кода. В поле «Subcategory» таким же способом задают семейство микроконтроллеров выбранной библиотеки (в нашем примере AVR Family). Все компоненты семейства отображаются в поле «Results». В поле «Manufacturer» выбирают производителя микроконтроллера. Если производитель не имеет значения – указывают значение «All Manufacturers». Для размещения микроконтроллера на схеме нажимают на кнопку OK, после чего окно «Pick Devices» будет закрыто, а символ компонента прикреплён к курсору мыши, при помощи которого его помещают в нужное место на схеме щелчком левой кнопки мыши.

Аналогичным образом добавим в рабочее поле проекта карту памяти MMC и микросхему буквенно-цифрового дисплея LM044L, которая находится в разделе «Alphanumeric LCDs» библиотеки «Optoelectronics» (см. рис. 2в).

Микросхема LM044L имеет 14 контактов [3], назначение которых следующее:

- Vss – GND;
- Vdd – напряжение питания +5 В;
- Vee – напряжение регулировки контрастности от 0 до +5 В (настройка контрастности отображаемых на дисплее символов);
- RS – выбор регистра данных DR (RS - 1) или команд IR (RS – 0);
- RW – выбор операции чтения (RW = 1) или записи (RW = 0);
- E – линия синхронизации;
- D0...D7 – шина данных/команд.

Микросхема LM044L может работать в двух режимах:

- 8-разрядном (для обмена информацией используются выводы D0...D7);
- 4-разрядном (для обмена информацией используются выводы D4...D7).

В представленном примере вывод данных на экран дисплея разрешением 20 символов на 4 строки выполнен в 4-разрядном режиме.

Для подключения микросхемы LM044L к схеме управления используется параллельная синхронная шина данных/команд (D0...D7), вывод выбора операции чтения/записи (RW), вывод выбора регистра данных/команд (RS) и вывод синхронизации (E). Подсоединим выводы модуля дисплея D4...D7 к выводам PC4...PC7, а выводы RS, RW и E к выводам PC0...PC2 микроконтроллера ATmega32 так, как показано на рис. 3. Выводы Vss и Vdd подключим к «земле» и напряжению +5 В соответственно. На вывод Vee подаётся напряжение контрастности (от 0 до +5В). На практике этот вывод подключают к питанию через подстроечный резистор, который позволяет плавно регулировать контрастность отображения символов на дисплее.

Символы «земли» и питания добавляют в схему, выбрав на панели «TERMINALS» (см. рис. 4) строки «GROUND» и «POWER». Панель открывают нажатием кнопки «Terminals Mode» на левой панели редактора ISIS.

Выбор линий портов микроконтроллера для подключения к указанным выводам дисплея выполняется разработчиком произвольно. В окне свойств дисплея (окно открывают двойным щелчком левой кнопки мыши после его выделения на схеме) в поле «Advanced Properties» из выпадающего списка выбирают пункт «Clock Frequency» (тактовая частота) (см. рис. 5а), значение которой должно совпадать с частотой работы микроконтроллера (в нашем примере 2 МГц).

Выводы CS, DI, DO, CLK карты памяти подсоединим к выводам PB4...PB7 микроконтроллера так, как показано на рис. 3. Для работы с картой в окне её свойств (окно открывают путём выделения левой кнопкой мыши карты на схеме, вызова правой кнопкой мыши контекстного меню и выбора в нём команды «Edit Properties») указывают следующие параметры (см. рис. 5б):

- Size of media (MB) – объём данных;
- Card Image File – путь к файлу образа карты;
- Require SPI init sequence – необходимость инициализации SPI интерфейса.

Если образ находится в одном каталоге со схемным проектом Proteus, то указывают только его имя с расширением .mmc. На схеме подключение карты в слот выполняют щелчком левой кнопки мыши по её компоненту.

В окне свойств микроконтроллера (окно открывают двойным щелчком левой кнопки мыши после его выделения на схеме) указывают путь к hex-файлу на диске компьютера (поле «Program File») и значение частоты (поле «CKSEL Fuses») – в нашем примере 2 МГц (см. рис. 5в).

Виртуальный терминал в рабочее поле проекта добавляют выбором левой кнопкой мыши на панели «INSTRUMENTS» (см. рис. 6) строки с названием «VIRTUAL TERMINAL».

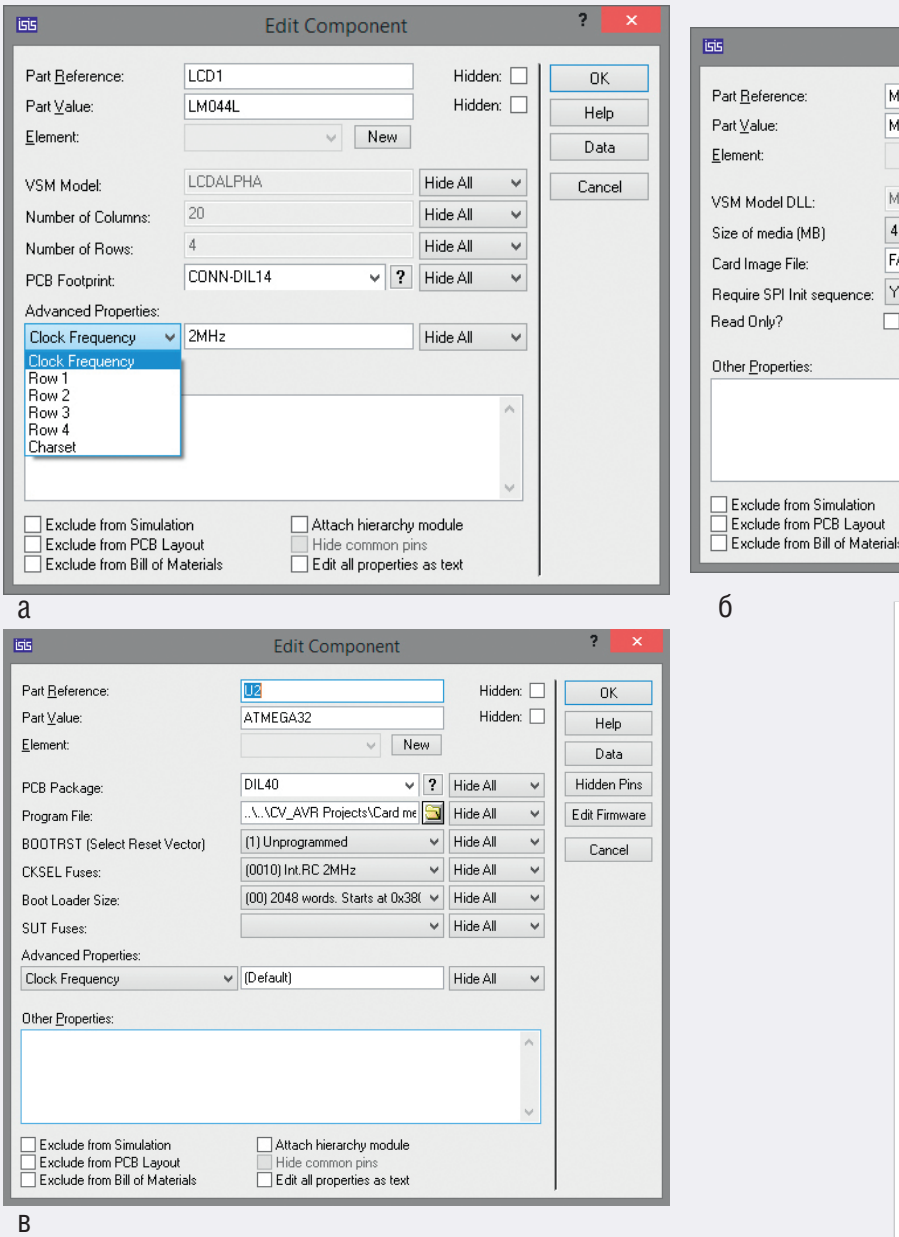

Рис. 5. Окно свойств: микросхемы LM044L (а), карты памяти (б), микроконтроллера ATmega32 (в)

Рис. 6. Выбор виртуального терминала на панели INSTRUMENTS программы Proteus

Панель «INSTRUMENTS» открывают кнопкой «Instruments Mode» левой панели редактора ISIS. Передачу данных на экран терминала выполняют по интерфейсу USART. Выводы микроконтроллера, используемые модулем USART, являются линиями ввода/вывода общего назначения. В микроконтроллере ATmega32 модулем USART используются линии PD0 (RXD) – вход USART, PD1 (TXD) – выход USART, PB0 (XCK) – вход/выход внешнего тактового сигнала USART. В нашем примере подсоединим вывод TXD микроконтроллера к выводу RXD виртуального терминала для передачи данных на его экран.

По умолчанию в редакторе ISIS для отображения текста на экране терминала установлен западноевропейский шрифт. Чтобы установить шрифт с поддержкой кириллицы (см. рис. 7), запускают симуляцию схемы, щелчком правой кнопки мыши в области открывшегося окна терминала вызывают контекстное меню и выбирают в нём пункт «Set Font», В результате откроется окно настройки шрифта, где задают шрифт (поле «Шрифт») – в нашем примере Courier New, стиль шрифта (поле «Начертание»), размер шрифта (поле «Размер») и нажимают кнопку OK.

После создания схемы, подключения всех приборов и настройки их параметров переходят к следующему этапу разработки – написанию программного кода управления устройством в CodeVisionAVR. В результате его компиляции (при условии отсутствия в коде ошибок) на диске компьютера будет получен hex-файл, путь к которому указывают в окне свойств микроконтроллера в Proteus.

Завершающим этапом работы в Proteus является запуск процесса моделирования схемы в редакторе ISIS, который выполняют кнопкой «Run the simulation», расположенной в левом нижнем углу окна редактора или командой основного меню «Debug/Run Simulation». Временную приостановку процесса симуляции выполняют кнопкой «Pause the simulation, or start up at time 0 if stopped» (кнопка находится в левом нижнем углу окна редактора). Останавливают моделирование кнопкой «Stop the simulation».

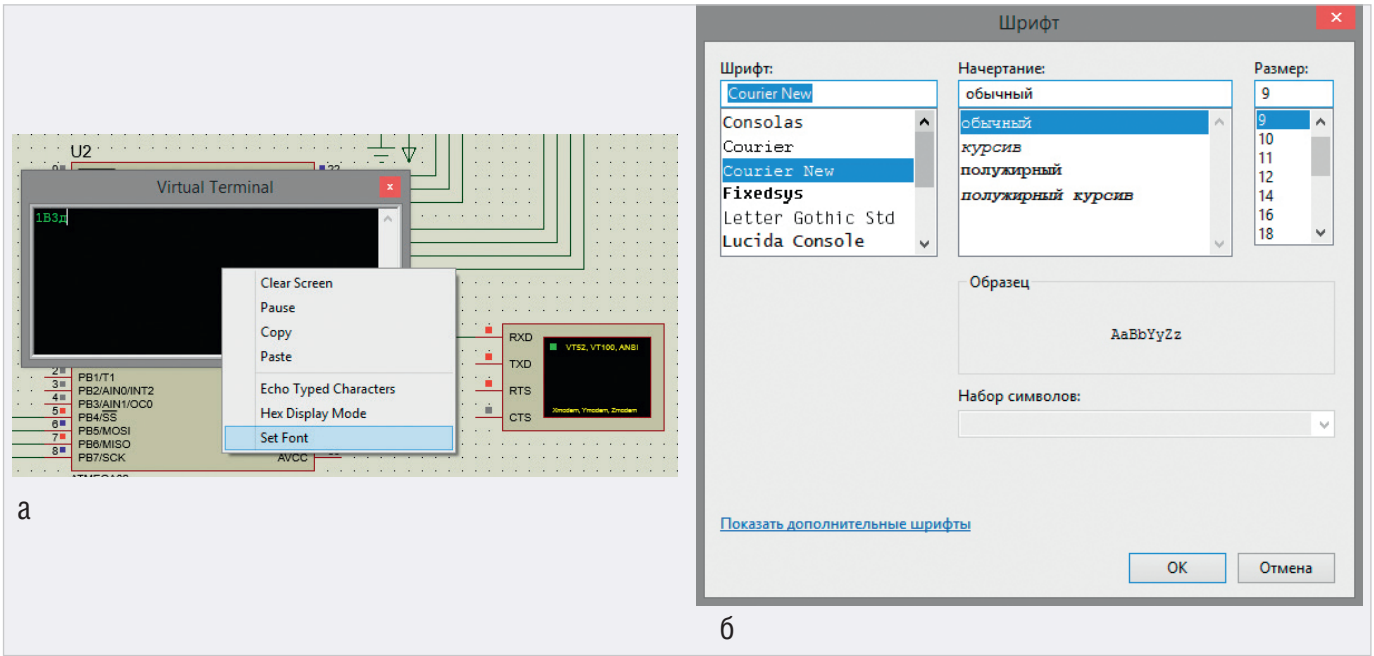

Рис. 7. Установка шрифта с поддержкой кириллицы: выбор команды Set Font в контекстном меню, открытом из области окна терминала (а), окно настройки шрифта (б)

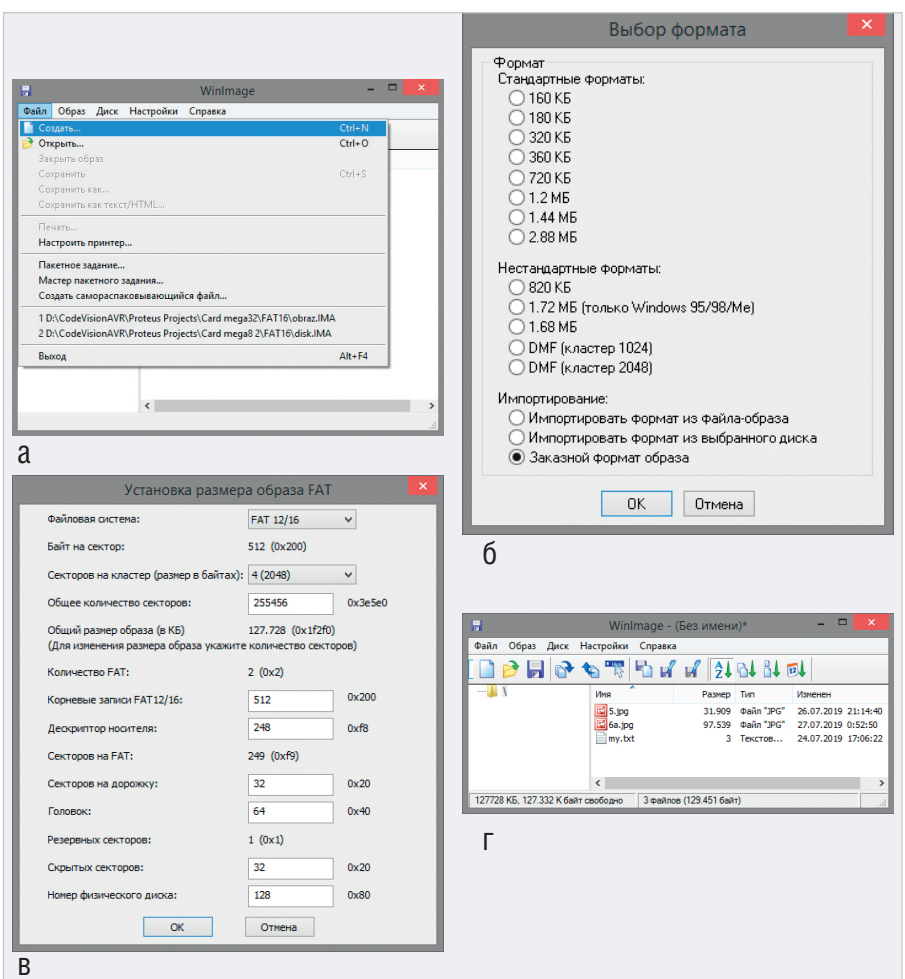

Рис. 8. Создание образа карты памяти в программе WinImage: меню «Файл» программы WinImage (а), выбор формата образа (б), выбор файловой системы и размера пространства для записи файлов (в), добавление файлов в образ (г)

Создание образа карты памяти

Для работы с картой памяти в Proteus в окне её свойств указывают имя образа с расширением. Образ содержит информацию о файлах карты и их содержимом. Для его создания можно воспользоваться программой WinImage (в нашем примере версия 9.0), в составе которой имеются средства создания, чтения и редактирования образа диска (точной копии физического диска или его раздела, которая сохраняет его структуру) и файловой системы. С помощью WinImage можно просматривать содержание образа, извлекать необходимые файлы и добавлять новые. После запуска программы в основном меню «Файл» указывают пункт «Создать» (см. рис. 8а). В открывшемся окне «Выбор формата» в поле «Импортирование» устанавливают переключатель в позицию «Заказной формат образа» (см. рис. 8б) и нажимают кнопку ОК. Для работы с носителем информации в операционной системе (создания, чтения, редактирования файлов) его нужно отформатировать с учётом определённой файловой системы. С этой целью в следующем окне «Установка размера образа FAT» (см. рис. 8в) указывают файловую систему (поле «Файловая система») – в нашем примере это значение FAT 12/16, количество секторов на кластер (поле «Секторов на кластер (размер в байтах)») – в нашем примере это значение 4 (2048), общее количество секторов, от которого зависит размер пространства для записи файлов (поле «Общее количество секторов»), – в нашем примере это значение 255456, что соответствует размеру 127,7 Кбайт (этого достаточно для реализации нашего

примера), и нажимают на кнопку ОК. Добавление файлов в образ выполняют их перемещением левой кнопкой мыши из каталога на диске компьютера в пустое поле редактора WinImage (см. рис. 8г), в статусной строке в нижней левой части которого отображается объём свободного пространства образа в Кбайт.

Сохранение образа на диск компьютера выполняют командой «Файл / Сохранить как» основного меню WinImage, после чего редактор можно закрыть. Среди доступных для сохранения (.imz, .ima, .vfd) отсутствует расширение .mmc, именно то, которое требуется для работы с картой памяти в Proteus. В таком случае образ на диск компьютера сохраняют в формате .ima, а затем изменяют его расширение на .mmc в любом файловом менеджере (например, в Total Commander). Теперь, указав в Proteus в окне свойств карты памяти путь к размещённому на диске компьютера образу, загруженные в него файлы можно читать через микроконтроллер.

Для создания с помощью программы инициализации микроконтроллера файлов на карте памяти и записи в них в Proteus в поле «Card Image File» окна свойств карты путь к образу в формате .ima прописывают вручную (без использования кнопки Open). Если воспользоваться стандартным интерфейсом выбора образа, то файл с расширением .ima на диске компьютера будет не виден.

Более примитивный способ создания образа, который также имеет право на существование, заключается в создании на диске компьютера с помощью редактора Блокнот текстового файла, записи в него текстовых данных и сохранения файла с расширением .mmc. В этом случае через микроконтроллер можно считать записанную текстовую информацию.

Создание программного кода в CodeVisionAVR

Формирование программного кода в CodeVisionAVR выполняют при помощи автоматического генератора CodeWizardAVR или вручную с нуля, используя синтаксис языка программирования С и функции стандартных библиотек программы. Удобство применения генератора состоит в быстром получении кода выполнения функций инициализации микроконтроллера и его портов ввода/вывода, аналогового компаратора, таймеров/счётчиков, интерфейса UART и SPI, буквенно-цифровых и графических дисплеев и др. Однако в процессе работы мастера формируется достаточно объёмный код, который впоследствии приходится редактировать.

После создания командой основного меню «File/New/Project» нового проекта в CodeVisionAVR открывается окно генератора кода CodeWizardAVR, где задают параметры микроконтроллера, его внутренних ресурсов и используемых в схеме периферийных устройств. В нашем примере это тип и частота работы микроконтроллера (вкладка «Chip Settings» – см. рис. 9а), опции модуля USART (вкладка «USART Settings» – см. рис. 9б), портов ввода/вывода микроконтроллера (вкладка «Ports Settings» – см. рис. 9в), буквенно-цифрового дисплея (вкладка «Alphanumeric LCD Settings» – см. рис. 9г), интерфейса SPI (см. рис. 9д). Важно, чтобы значение тактовой частоты микроконтроллера, указанное в поле Clock вкладки «Chip Settings», совпадало со значением в поле «CKSEL Fuses» его окна свойств в Proteus (в нашем примере это 2 MHz).

На вкладке «Alphanumeric LCD Settings» задают разрешение поддержки буквенно-цифрового дисплея (поле «Enable Alphanumeric LCD Support»), тип контроллера (поле «Controller Type», в нашем примере – HD44780) и количество символов в строке (поле «Character/Line», в нашем примере – 20). В поле «Connections» настраивают параметры подключения микроконтроллера (порт и номер вывода) к микросхеме дисплея, работающего в 4-разрядном режиме – в нашем примере 0, 1 и 2 биты порта PC микроконтроллера подключены к выводам RS, RD и E дисплея, 4...7 биты порта PC микроконтроллера подключены к выводам D4...D7 дисплея. Если предполагается, что буквенно-цифровой дисплей будет работать в 8-разрядном режиме и написание кода программы управления будет вестись самостоятельно (так как стандартной библиотеки для этого режима в CodeVisionAVR нет), то поле «Connections» можно не заполнять.

На вкладке «Ports Settings» для каждого отдельного порта микроконтроллера отведена своя закладка, где в поле «Data Direction» щелчком левой кнопки мыши выбирают одно из значений битов порта: Out (линия порта работает на вывод данных), In (приём данных). В нашем примере для битов Bit 0...Bit 7 портов Port D и Port C укажем значение Out, для бита Bit 6 порта Port B – In, для битов Bit 4, Bit 5, Bit 7 порта Port B – Out. Значения битов неиспользуемых выводов микроконтроллера не важны, поэтому их можно оставить по умолчанию.

На вкладке «USART Settings» определяют следующие параметры USART:

- Receiver – активизация приёмника USART;
- RxInterrupt – выбор режима работы приёмника путём снятия (режим опроса) или установки (режим прерывания) флажка. В режиме прерывания доступна опция «Receiver Buffer», которая определяет размер буфера приёмника;
- Transmitter – активизация передатчика USART;
- TxInterrupt – выбор режима работы передатчика путём снятия (режим опроса) или установки (режим прерывания) флажка. В режиме прерывания доступна опция «Transmitter Buffer», которая определяет размер буфера передатчика;
- Baud Rate – скорость передачи в режимах приёмника и передатчика;
- Baud Rate Error – ошибка скорости передачи (вычисляется автоматически);
- Communication Parameters – установка параметров связи (Data – количество битов данных, Stop – количество стоповых битов, Parity – чётность);
- Mode – выбор режима связи: Asynchronous (асинхронный), Sync. Master UCPOL=0 (синхронный ведущий со сброшенным битом UCPOL регистра UCSRC), Sync. Master UCPOL=1 (синхронный ведущий с установленным битом UCPOL регистра UCSRC), Sync. Slave UCPOL=0 (синхронный ведомый со сброшенным битом UCPOL регистра UCSRC), Sync. Slave UCPOL=1 (синхронный ведомый с установленным битом UCPOL регистра UCSRC).

На вкладке SPI Settings определяют следующие параметры SPI:

- SPI Enabled – активизация работы SPI-интерфейса;
- SPI Interrupt – разрешение прерывания от SPI-интерфейса;
- Clock Rate x2 – удвоение тактовой частоты SPI-интерфейса;
- SPI Mode – режим работы SPI;

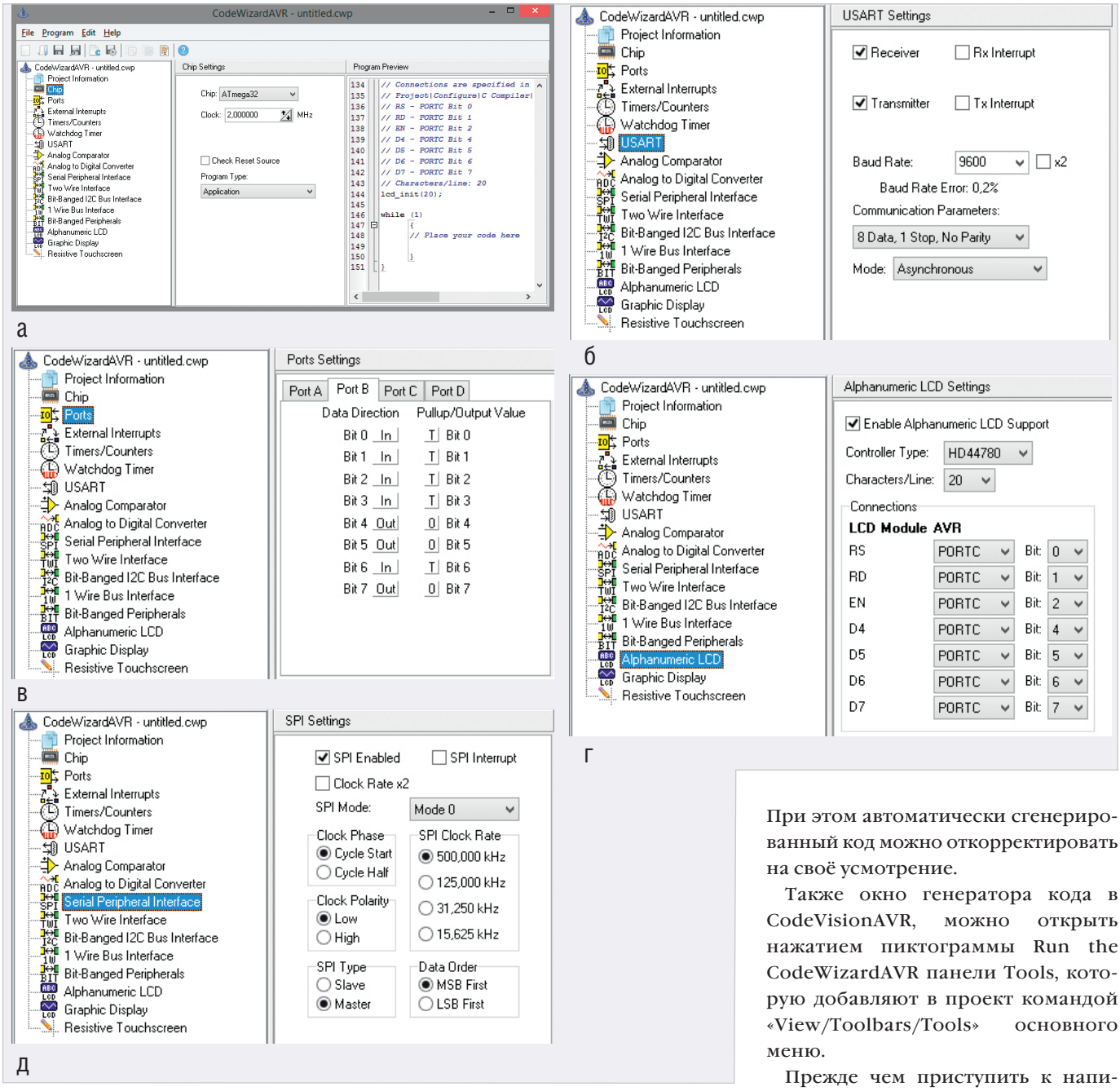

Рис. 9. Настройка в окне CodeWizardAVR параметров: микроконтроллера (а), модуля USART (б), портов ввода/вывода микроконтроллера (в), буквенно-цифрового дисплея (г), интерфейса SPI (д)

- SPI Clock Rate – выбор тактовой частоты для последовательной передачи данных по SPI-интерфейсу;
- Clock Phase – выбор позиции фронта стробирующего сигнала относительно бита данных: Cycle Start (начало цикла), Cycle Half (половина цикла);
- Clock Polarity – уровень полярности тактовых импульсов: Low (низкий), High (высокий);
- SPI Type – выбор роли микроконтроллера в SPI-интерфейсе: Slave – ведомый, Master – ведущий;
- Data Order – порядок данных при последовательной передаче: MSB First (первый старший байт), LSB First (первый младший байт).

Предварительный просмотр кода программы, который генерируют командой «Program/Generate» основного меню CodeWizardAVR, выполняют в поле «Program Preview». После настройки параметров и генерации кода командой «Program/Generate, Save and Exit» основного меню или пиктограммой «Generate program, save and exit» верхней панели инструментов окно CodeWizardAVR автоматически будет закрыто. Полученный код отобразится в окне кода CodeVisionAVR, где и будет вестись дальнейшее написание программы.

При этом автоматически сгенерированный код можно откорректировать на своё усмотрение.

Также окно генератора кода в CodeVisionAVR, можно открыть нажатием пиктограммы Run the CodeWizardAVR панели Tools, которую добавляют в проект командой «View/Toolbars/Tools» основного меню.

Прежде чем приступить к написанию программного кода в CodeVisionAVR подключим поддержку карты памяти MMC и увеличим размер стека, для чего с помощью команды Project/Configure основного меню откроем окно «Configure Project», перейдём на вкладку «C Compiler», на которой перейдём на вкладку «Libraries», где откроем вкладку «MMC/SD/SD HC Card» и установим флажок в чекбоксе «Enable MMC/SD/SD HC Card and FAT Support» (разрешить поддержку карт памяти и файловой системы FAT) (см. рис. 10а). На вкладке «C Compiler» откроем вкладку «Code Generation» (см. рис. 10б) и в поле «Data Stack Size» укажем размер стека в байтах – для компиляции кода в нашем примере значения 1840 будет достаточно.

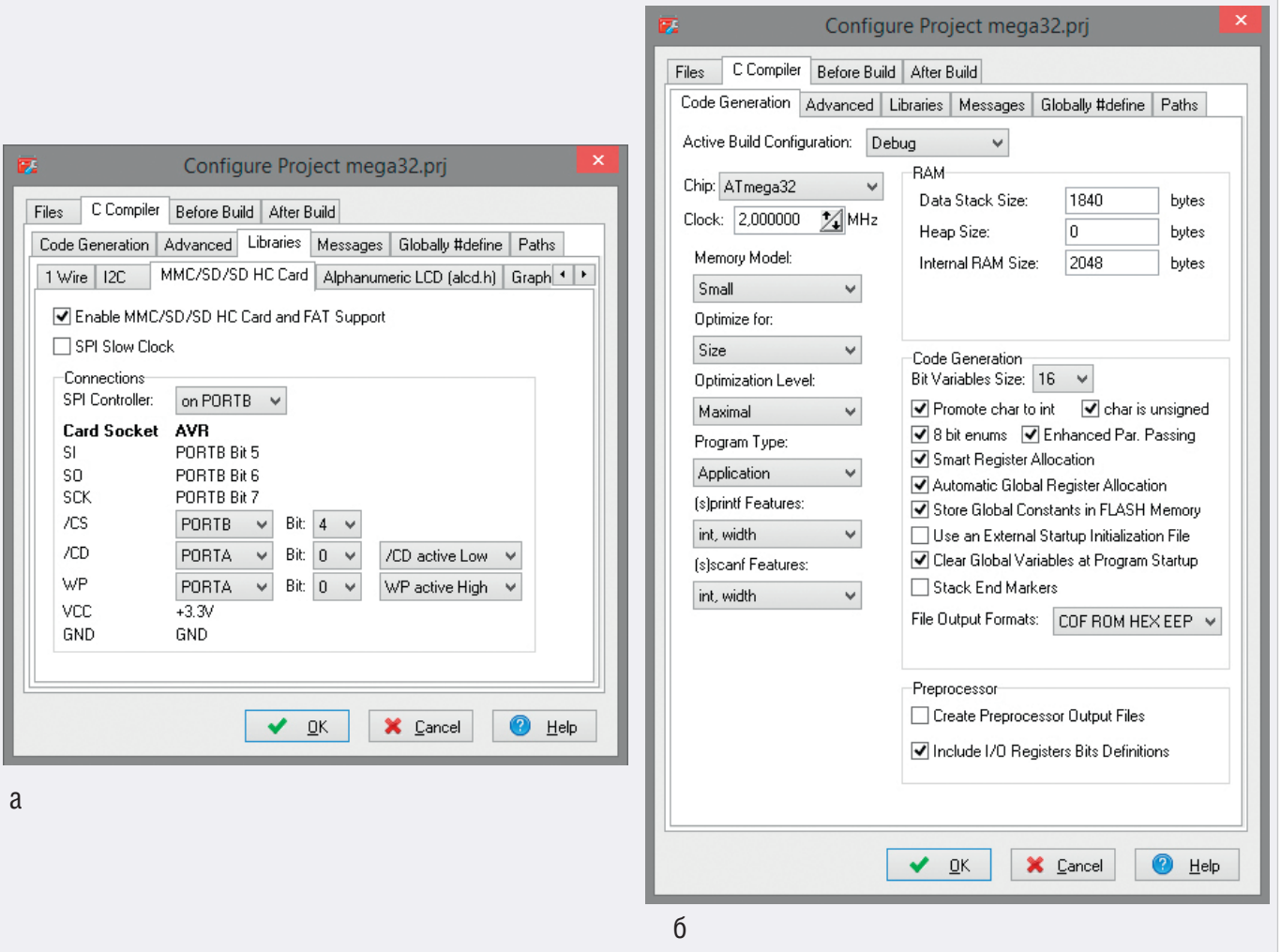

Рис. 10. Окно Configure Project: вкладка MMC/SD/SD HC Card (а), вкладка Code Generation (б)

Применение функций библиотеки sdcard.h для чтения и записи данных во внешнюю память

Для работы с картами памяти MMC/SD/SD HC, отформатированными в FAT12, FAT16 или FAT32 в CodeVisionAVR предусмотрены библиотеки sdcard.h и ff.h. С помощью функций disk_read и disk_write библиотеки sdcard.h удобно выполнить чтение и запись текстовых данных во внешнюю память, когда образ карты памяти создан с помощью редактора Блокнот. В коде программы функция чтения информации имеет следующий формат записи: disk_read (unsigned char drv, unsigned char* buff, unsigned long sector, unsigned char count), где drv – это номер устройства (нумерация начинается с 0), buff – переменная для записи массива считанных данных, sector – номер блока, с которого начнётся чтение, count – количество блоков для чтения (нумерация начинается с 1). Функция записи имеет следующий формат: disk_write (unsigned char drv, unsigned char* buff, unsigned long sector, unsigned char count), где drv – это номер устройства, buff – переменная, в которой хранятся данные для записи, sector – номер блока, с которого начнётся запись, count – количество блоков для записи. Чтение и запись информации во внешнюю память выполняют блоками объёмом 512 байт.

Рассмотрим работу с функциями disk_read и disk_write на конкретном примере, для чего создадим в Блокноте новый текстовый документ, запишем в него следующий текст: «I like CodeVisionAVR!», сохраним файл с расширением *.mmc на диск компьютера в одном каталоге с проектом Proteus и укажем его имя в поле Card Image File в окне свойств карты памяти.

Для записи информации во внешнюю память, её чтения и отображения на экране терминала и буквенно-цифрового дисплея напишем программу инициализации микроконтроллера на языке С с применением стандартных функций CodeVisionAVR. Текст программы приведён в листинге 1.

Введём текст программы в окне кода CodeVisionAVR и запустим командой основного меню «Project/Build All» компиляцию, по окончании которой выдаётся отчёт о наличии ошибок в коде программы (см. рис. 11). Если ошибки не обнаружены, на диске компьютера будет создан hex-файл для записи в микроконтроллер.

Теперь перейдём в Proteus и в окне свойств микросхемы ATmega32 в поле «Program File» укажем путь к файлу прошивки на диске компьютера. Командой основного меню «Debug / Run Simulation» запустим в Proteus моделирование собранной схемы (результат представлен на рис. 12) и проанализируем её работу.

После запуска программа инициализации микроконтроллера проверяет подключение карты памяти с помощью функции disk_initialize(0), где в скобках указан номер подключаемого устройства (при успешном выполнении функ-

Листинг 1

```
include <mega32.h> // Подключение заголовочных файлов
#include <alcd.h> // в которых содержатся
#include <stdio.h> // прототипы функций
#include <delay.h>
#include <sdcard.h>

#define F_CPU 2000000 // Рабочая частота микроконтроллера
#define BAUD 9600L // Скорость обмена данными
#define UBRRL_value (F_CPU/(BAUD*16))-1 // Согласно заданной скорости
// подсчитываем значение для регистра UBRR

void init_USART() { // Функция инициализации USART
UBRRL = UBRRL_value; // Младшие 8 бит UBRRL_value
UBRRH = UBRRL_value >> 8; // Старшие 8 бит UBRRL_value
UCSRB = (1<<TXEN); // Бит разрешения передачи
UCSRC = (1<< UCSZ0)|(1<< UCSZ1); } // Устанавливаем формат 8 бит данных

void send_UART(char value) {
while(!( UCSRA & (1 << UDRE))); // Ожидаем когда очистится буфер передачи
UDR = value; } // Помещаем данные в буфер, начинаем передачу

interrupt [TIM1_COMPA] void timer1_compa_isr(void)
{disk_timerproc();} // Вызов функции синхронизации

void main(void) // Основная функция программы
{
unsigned char Buff[512];
unsigned char Buff2[512]="Чтение информации с карты памяти и ее вывод на экран терминала";
unsigned char Buff3[512];
int i;

// Инициализация портов микроконтроллера

// Port A
DDRA=(0<<DDA7) | (0<<DDA6) | (0<<DDA5) | (0<<DDA4) | (0<<DDA3) |
(0<<DDA2) | (0<<DDA1) | (0<<DDA0);
PORTA=(0<<PORTA7) | (0<<PORTA6) | (0<<PORTA5) | (0<<PORTA4) |
(0<<PORTA3) | (0<<PORTA2) | (0<<PORTA1) | (0<<PORTA0);

// Port B
DDRB=(1<<DDB7) | (0<<DDB6) | (1<<DDB5) | (1<<DDB4) | (0<<DDB3) |
(0<<DDB2) | (0<<DDB1) | (0<<DDB0);
PORTB=(0<<PORTB7) | (0<<PORTB6) | (0<<PORTB5) | (0<<PORTB4) |
(0<<PORTB3) | (0<<PORTB2) | (0<<PORTB1) | (0<<PORTB0);

// Port C
DDRC=(1<<DDC7) | (1<<DDC6) | (1<<DDC5) | (1<<DDC4) | (1<<DDC3) |
(1<<DDC2) | (1<<DDC1) | (1<<DDC0);
PORTC=(0<<PORTC7) | (0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) |
(0<<PORTC3) | (0<<PORTC2) | (0<<PORTC1) | (0<<PORTC0);

// Port D
DDRD=(1<<DDD7) | (1<<DDD6) | (1<<DDD5) | (1<<DDD4) | (1<<DDD3) |
(1<<DDD2) | (1<<DDD1) | (1<<DDD0);
PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) | (0<<PORTD4) |
(0<<PORTD3) | (0<<PORTD2) | (0<<PORTD1) | (0<<PORTD0);

// Инициализация таймера

TCCR1A=0x00;
TCCR1B=0x0D;
TCNT1H=0x00;
TCNT1L=0x00;
OCR1AH=0x00;
OCR1AL=0x4E;
TIMSK=0x10;

lcd_init(20); // Инициализация дисплея
#asm("sei")

// Проверка подключения карты памяти
if((disk_initialize(0))==0) // Если карта подключена
{lcd_puts("OK");} // Вывод на экран дисплея сообщения OK
else // иначе
{lcd_puts("Karta ne podkluchena");} // Вывод сообщения об отсутствии подключения карты

init_USART(); // Инициализация USART
disk_read (0, Buff, 0, 1); // Чтение с карты памяти текстового блока
// и его запись в переменную Buff
delay_ms(500); // Пауза длительностью 500 мс
lcd_gotoxy(0,1); // Установка курсора в первую позицию второй строки дисплея
for (i=0;i<33;i++)
{send_UART(Buff[i]); // Вывод считанной с карты памяти информации на экран терминала
lcd_putchar(Buff[i]);} // Вывод считанной с карты памяти информации на экран дисплея
delay_ms(500);
disk_write (0, Buff2, 0, 1); // Запись на карту памяти блока информации
// из переменной Buff2
disk_read (0, Buff3, 0, 1); // Чтение информации с карты памяти
// и её запись в переменную Buff3
delay_ms(500);
printf("\n\r"); // Переход на новую строку на экране терминала
for (i=0;i<64;i++)
{ send_UART(Buff3[i]); }} // Вывод считанной с карты памяти информации
// на экран терминала
```

ция возвращает 0). Если карта не подключена, на экран дисплея выводится предупреждающее сообщение «Karta ne podkluchena». Когда карта подключена, на экран дисплея выводится сообщение «OK», выполняется инициализация интерфейса USART для вывода данных на экран терминала, затем чтение одного текстового блока объёмом 512 байт (четвёртый параметр функции чтения count = 1), начиная с первого сектора (третий параметр функции чтения sector = 0), и запись считанных данных в переменную Buff.

После паузы длительностью 500 мс с помощью функции lcd_gotoxy(0,1); курсор дисплея устанавливается в первую позицию второй строки и начинается посимвольный вывод в цикле for считанной с карты памяти информации на экран терминала (функция send_UART(Buff[i]);) и дисплея (функция lcd_putchar(Buff[i]);). Запись на карту памяти массива данных из переменной Buff2, начиная с первого сектора (третий параметр функции записи sector = 0), выполняется с помощью функции disk_write (0, Buff2, 0, 1); после паузы длительностью 500 мс. В результате уже имеющаяся в файле образа MMC.mmc информация будет перезаписана (см. рис. 13). Теперь с помощью функции disk_read (0, Buff3, 0, 1); считаем записанную информацию с карты памяти и после паузы длительностью 500 мс выведем её посимвольно с помощью функции send_UART(Buff3[i]); с новой строки на экран терминала.

Обмен информацией между микроконтроллером и картой памяти выполняется через линии PB4...PB7 микроконтроллера. Для работы с буквенно-цифровым дисплеем задействованы линии PC0...PC2 и PC4...PC7 микроконтроллера, линия PD1 используется для последовательного вывода информации на экран терминала.

Применение функций библиотеки ff.h для чтения и записи данных во внешнюю память

В библиотеке ff.h определены функции работы с картой памяти, отформатированной в файловой системе FAT, среди которых следующие:

- f_open(FIL* fp, const char* path, unsigned char mode) – функция создания нового или открытия уже имеющего файла, где fp – указатель на структуру данных фай-

ла, path – номер устройства и имя файла для создания/открытия, mode – режим работы с файлом. Аргумент path имеет формат записи 0:/file.txt. Значения аргумента mode: FA_READ – чтение данных из файла, FA_WRITE – запись данных в файл, FA_OPEN_EXISTING – открытие уже имеющегося на карте файла, FA_OPEN_ALWAYS – создание нового или открытие существующего файла, FA_CREATE_NEW – создание нового файла, FA_CREATE_ALWAYS – создание нового файла, если файл с указанным именем уже существует, то он будет перезаписан. Допускается комбинация представленных значений. Например, запись FA_OPEN_EXISTING | FA_READ означает открытие уже существующего файла для чтения. Функция возвращает следующие значения:

- FR_OK – успешное выполнение функции;
- FR_NO_FILE – не удалось найти файл;
- FR_NO_PATH – путь к файлу не существует;
- FR_INVALID_NAME – неверное имя файла;
- FR_INVALID_DRIVE – неверный номер устройства;
- FR_EXIST – создание файла невозможно, так как файл с таким именем уже существует;
- FR_DENIED – в доступе отказано по одной из следующих причин: попытка открыть файл только для чтения в режиме записи, невозможно создать файл, потому что файл с таким именем уже существует или отсутствует свободное место на карте;
- FR_NOT_READY – доступ к карте памяти невозможен из-за отсутствия её подключения или по другой причине;
- FR_WRITE_PROTECTED – создание файла или его открытие в режиме записи невозможно, поскольку носитель защищён от записи;
- FR_DISK_ERR – ошибка доступа к карте памяти;
- FR_INT_ERR – внутренняя ошибка карты памяти или файловой системы FAT;
- FR_NOT_ENABLED – логический диск не был смонтирован с помощью функции f_mount;
- FR_NO_FILESYSTEM – на физическом носителе отсутствует корректный раздел FAT.

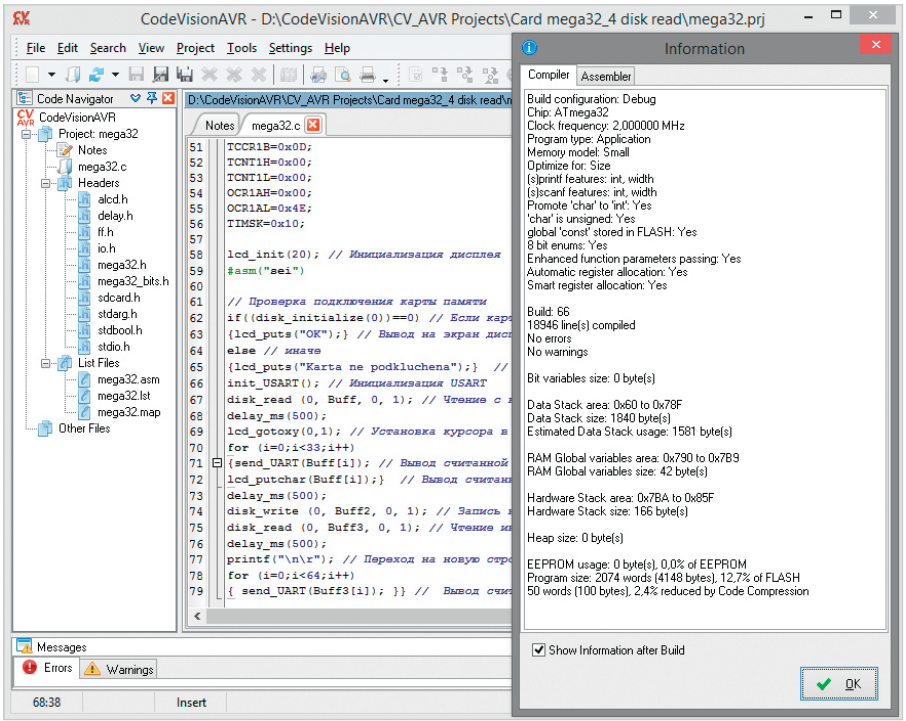

Рис. 11. Результат компиляции программного кода в CodeVisionAVR

- f_close (FIL* fp) – функция закрытия файла (если измененный файл не закрыть, данные могут быть утеряны). Параметр fp – указатель на структуру данных файла. Функция возвращает следующие значения:
  - FR_OK – успешное выполнение функции;
  - FR_NOT_READY – доступ к карте памяти невозможен из-за отсутствия её подключения или по другой причине;
  - FR_DISK_ERR – ошибка доступа к карте памяти;

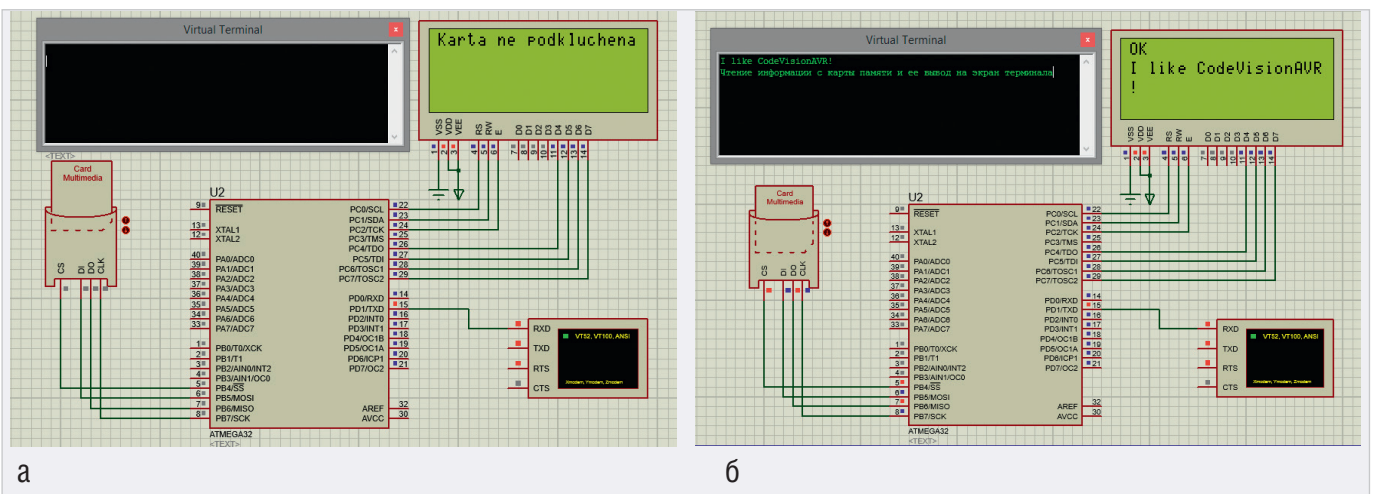

Рис. 12. Результат работы программы обмена данными с картой памяти, образ которой создан с помощью редактора Блокнот: карта памяти не подключена (а), чтение и запись информации на карту памяти и её вывод на экран терминала и дисплея (б)

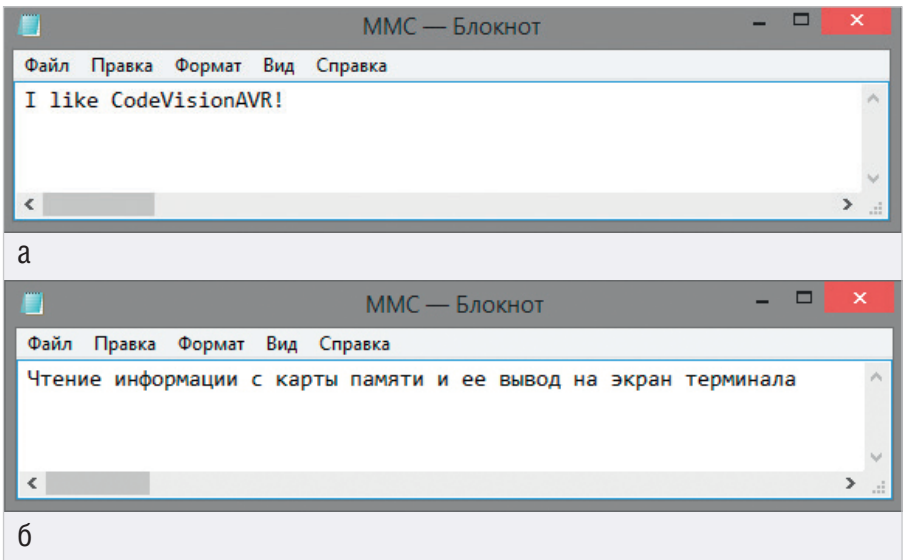

Рис. 13. Файл MMC.mmc: до (а) и после (б) записи информации на карту памяти

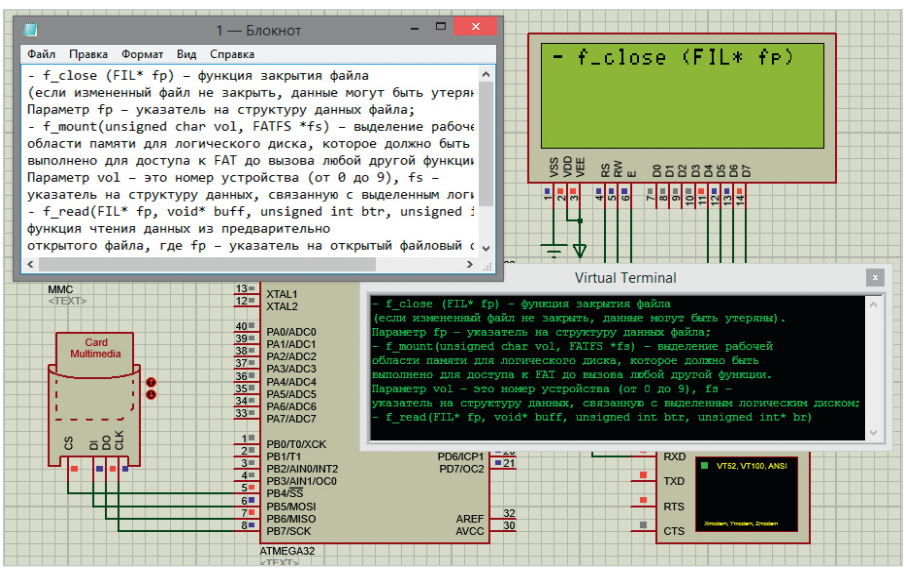

Рис. 14. Результат работы программы чтения текстовых данных из файла 1.txt, размещённого на карте памяти, образ которой создан с помощью WinImage и преобразован в формат .mmc (считанные данные отображаются на экране терминала и буквенно-цифрового дисплея)

- FR_INT_ERR – внутренняя ошибка карты памяти или файловой системы FAT;
- FR_INVALID_OBJECT – файл не был открыт с помощью функции f_open.

- f_mount(unsigned char vol, FATFS *fs) – выделение рабочей области памяти для логического диска, которое должно быть выполнено для доступа к FAT до вызова любой другой функции. Параметр vol – это номер устройства (от 0 до 9), fs – указатель на структуру данных, связанную с выделенным логическим диском. Значения, которые возвращает функция:
  - FR_OK – успешное выполнение функции;
  - FR_INVALID_DRIVE – неверный номер устройства.
- f_write(FIL* fp, const void* buff, unsigned int btw, unsigned int* bw) – функция записи данных в предварительно открытый файл, где fp – указатель на открытый файловый объект, buff – переменная, в которой хранятся данные для записи, btw – количество байтов для записи, bw – количество реально записанных байтов. Значения, которые возвращает функция:
  - FR_OK – успешное выполнение функции;
  - FR_DENIED – в доступе к файлу отказано, так как он открыт только для чтения;
  - FR_NOT_READY – доступ к карте памяти невозможен из-за отсутствия её подключения или по другой причине;
  - FR_DISK_ERR – ошибка доступа к карте памяти;

НОВОСТИ МИРА

В 2030 году объём мирового рынка робототехники превысит $160 млрд

Общий объём мирового рынка робототехники достигнет $160-260 млрд в 2030 году, говорится в исследовании консалтинговой компании The Boston Consulting Group (BCG).

Компания при этом оценивает объём мирового рынка робототехники в 2020 году в $25 млрд. Также отмечается, что в 2030 году профессиональные сервисные роботы с объёмом рынка от $90 до $170 млрд намного опередят обычных промышленных роботов и коботов с объёмом рынка от $40 до $50 млрд.

В исследовании в том числе рассматриваются три возможных сценария развития робототехники на ближайшие 10 лет, которые определяются такими факторами, как готовность технологий и спрос. Первый сценарий связан с ростом числа индивидуальных решений и отсутствием новых вариантов массового использования, что ведёт к проектированию целевых роботизированных систем, начальные цены на которые довольно высоки.

Робот как стандартное устройство автоматизации – в этом сценарии на рынке будет лидировать несколько решений, которые легко установить, настроить и интегрировать. Примером может быть робот-курьер, автономный робот-сборщик или робот для зарядки электромобилей. Как правило, варианты массового использования в этой категории не отличаются особой сложностью, – добавили в сообщении.

Третий возможный сценарий – «Мир Google», при котором прорыв в области ИИ, адаптивности и связанности приведёт к появлению целого ряда интеллектуальных модулей, способных обрабатывать сложные и динамические ситуации. Так программное обеспечение будет ключевым фактором успеха, а доминировать будут компании, занимающиеся крупномасштабными разработками.

industry-hunter.com

Листинг 2

```
#include <mega32.h> // Подключение заголовочных файлов
#include <alcd.h> // в которых содержатся
#include <stdio.h> // прототипы функций
#include <delay.h>
#include <ff.h>

#define F_CPU 2000000 // Рабочая частота микроконтроллера
#define BAUD 9600L // Скорость обмена данными
#define UBRRL_value (F_CPU/(BAUD*16))-1 // Согласно заданной скорости
// подсчитываем значение для регистра UBRR

void init_USART() { // Функция инициализации USART
UBRRL = UBRRL_value; // Младшие 8 бит UBRRL_value
UBRRH = UBRRL_value >> 8; // Старшие 8 бит UBRRL_value
UCSRB = (1<<TXEN); // Бит разрешения передачи
UCSRC = (1<< UCSZ0)|(1<< UCSZ1); } // Устанавливаем формат 8 бит данных

void send_UART(char value) {
while(!( UCSRA & (1 << UDRE))); // Ожидаем когда очистится буфер передачи
UDR = value; } // Помещаем данные в буфер, начинаем передачу

interrupt [TIM1_COMPA] void timer1_compa_isr(void)
{disk_timerproc();} // Вызов функции синхронизации

void main(void) // Основная функция программы
{
FATFS fat; // Выделение рабочей области памяти для логического диска
FIL file; // Указатель на структуру данных файла
unsigned char Buff[542]; // Переменная для записи считанных данных
// из файла 1.txt карты памяти
unsigned int br; // Число прочитанных из файла 1.txt байтов
int i;

// Инициализация портов микроконтроллера
// Port A
DDRA=(0<<DDA7) | (0<<DDA6) | (0<<DDA5) | (0<<DDA4) | (0<<DDA3) |
(0<<DDA2) | (0<<DDA1) | (0<<DDA0);
PORTA=(0<<PORTA7) | (0<<PORTA6) | (0<<PORTA5) | (0<<PORTA4) |
(0<<PORTA3) | (0<<PORTA2) | (0<<PORTA1) | (0<<PORTA0);

// Port B
DDRB=(1<<DDB7) | (0<<DDB6) | (1<<DDB5) | (1<<DDB4) | (0<<DDB3) |
(0<<DDB2) | (0<<DDB1) | (0<<DDB0);
PORTB=(0<<PORTB7) | (0<<PORTB6) | (0<<PORTB5) | (0<<PORTB4) |
(0<<PORTB3) | (0<<PORTB2) | (0<<PORTB1) | (0<<PORTB0);

// Port C
DDRC=(1<<DDC7) | (1<<DDC6) | (1<<DDC5) | (1<<DDC4) | (1<<DDC3) |
(1<<DDC2) | (1<<DDC1) | (1<<DDC0);
PORTC=(0<<PORTC7) | (0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) |
(0<<PORTC3) | (0<<PORTC2) | (0<<PORTC1) | (0<<PORTC0);

// Port D
DDRD=(1<<DDD7) | (1<<DDD6) | (1<<DDD5) | (1<<DDD4) | (1<<DDD3) |
(1<<DDD2) | (1<<DDD1) | (1<<DDD0);
PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) | (0<<PORTD4) |
(0<<PORTD3) | (0<<PORTD2) | (0<<PORTD1) | (0<<PORTD0);

// Инициализация таймера
TCCR1A=0x00;
TCCR1B=0x0D;
TCNT1H=0x00;
TCNT1L=0x00;
OCR1AH=0x00;
OCR1AL=0x4E;
TIMSK=0x10;

lcd_init(20); // Инициализация дисплея
#asm("sei")
delay_ms(200);

init_USART(); // Инициализация USART
delay_ms(200);

f_mount(0, &fat); // Выделение рабочей области памяти для логического раздела
f_open(&file,"0:/1.txt", FA_OPEN_EXISTING | FA_READ); // Открываем файл 1.txt
// только для чтения
f_read(&file, Buff, sizeof(Buff), &br); // Читаем в переменную Buff данные из файла 1.txt
f_close(&file); // Закрываем файл 1.txt
delay_ms(50); // Пауза длительностью 50 мс
lcd_gotoxy(0,0); // Установка курсора в первую позицию первой строки дисплея
for (i=0;i<br;i++)
{send_UART(Buff[i]);} // Вывод считанных из файла 1.txt карты памяти данных
// на экран терминала
for (i=0;i<19;i++)
{lcd_putchar(Buff[i]);} // Вывод на экран дисплея 19 символов считанных
// из файла 1.txt карты памяти
}
```

- FR_INT_ERR – внутренняя ошибка карты памяти или файловой системы FAT;
- FR_INVALID_OBJECT – файл не был открыт с помощью функции f_open.

- f_read(FIL* fp, void* buff, unsigned int btr, unsigned int* br) – функция чтения данных из предварительно открытого файла, где fp – указатель на открытый файловый объект, buff – переменная для записи считанных данных, btr – количество байтов для чтения, br – количество реально прочитанных байтов. Значения, которые возвращает функция:
  - FR_OK – успешное выполнение функции;
  - FR_DENIED – в доступе к файлу отказано, так как он открыт только для записи;
  - FR_NOT_READY – доступ к карте памяти невозможен из-за отсутствия её подключения или по другой причине;
  - FR_DISK_ERR – ошибка доступа к карте памяти;
  - FR_INT_ERR – внутренняя ошибка карты памяти или файловой системы FAT;
  - FR_INVALID_OBJECT – файл не был открыт с помощью функции f_open.

Рассмотрим работу с функцией чтения данных из внешней памяти на конкретном примере, для чего создадим с помощью программы WinImage образ карты памяти, добавим в образ файл с расширением *.txt, который содержит блок текстовых данных, сохраним образ в папке FAT16 в каталоге с проектом Proteus, изменим его расширение на .mmc и укажем путь к файлу образа и его имя в поле Card Image File в окне свойств карты памяти. Для чтения данных из размещённого во внешней памяти файла и их отображения на экране терминала и буквенно-цифрового дисплея напишем программу инициализации микроконтроллера на языке С с применением стандартных функций CodeVisionAVR.

Текст программы приведён в листинге 2.

Введем текст программы в окне кода CodeVisionAVR и запустим компиляцию. После чего перейдём в Proteus и в окне свойств микросхемы ATmega32 укажем путь к файлу прошивки на диске компьютера. Запустим моделирование собранной схемы, результат которого представлен на рис. 14, и проанализируем её работу.

После запуска симуляции происходит монтирование карты памяти, а затем открытие размещённого на ней файла 1.txt для чтения из него в переменную Buff фрагмента текстовых данных. Объём реально прочитанных данных в байтах записывается в переменную br и может отличаться от заданного. После закрытия файла выполняется посимвольный вывод в цикле for всего блока считанных данных на экран терминала. На экран дисплея посимвольно выводятся в цикле for 19 символов из файла 1.txt.

Литература:

1. ISIS Help, Labcenter Electronics, 2014.
2. CodeVisionAVR Help, HP InfoTech, 2014.
3. HD44780U (LCD-II) (Dot Matrix Liquid Crystal Display Controller/Driver). Hitachi, Ltd. 1998.