Лабораторная работа №8. МИКРОКОНТРОЛЛЕРЫ AT91SAM7

Цель работы:

- знакомство с архитектурой микроконтроллеров SAM7,
- изучение контроллеров ввода-вывода PIO, прерываний AIC, последовательного канала USART и программных примеров, иллюстрирующих их работу,
- программирование контроллеров.

Введение

В системах управления, построенных на основе микроконтроллеров, постоянно появляются новые задачи управления и контроля, неуклонно растет сложность расчетов, и становятся особенно важными требования к скорости вычислений. К современным микроконтроллерам часто предъявляют требования расширить функциональные возможности. Это привело к появлению 32-разрядных микроконтроллерных систем, выполненных на одном кристалле. Широкое распространение получили микроконтроллеры с ядром ARM, производимые многими фирмами: Atmel, Philips, Intel, Analog Device, Siemens и др. К типичным сферам применения микроконтроллеров серии SAM7 относятся: управление приборами, измерительные устройства, системы обеспечения безопасности, регистраторы информации, мобильные телефоны с USB-портом, а также аксессуары ПК.

I. Архитектура и программирование микроконтроллеров AT91SAM7

1. Состав микроконтроллера и назначение устройств

Микроконтроллеры AT91SAM7S256/128 фирмы Atmel поставляются в корпусе LQFP или QFN с 64 выводами. Блок-схема микроконтроллеров приведена на рис. 1, топологические схемы корпусов – на рис. 2.

В состав микроконтроллера входят процессорное ядро, устройства памяти, системный контроллер, периферийные устройства, средства тестирования и отладки.

Процессорное ядро представляет собой RISC-ядро, основанное на архитектуре фон Неймана, с тактовой частотой до 55 МГц и удельной производительностью 0,9 MIPS/МГц. Процессор поддерживает две системы команд: 32-битовую высокоэффективную ARM и 16-битовую с высокой плотностью кода Thumb. Обработка команд осуществляется с помощью трёхуровневого конвейера с уровнями выборки, декодирования и выполнения команд.

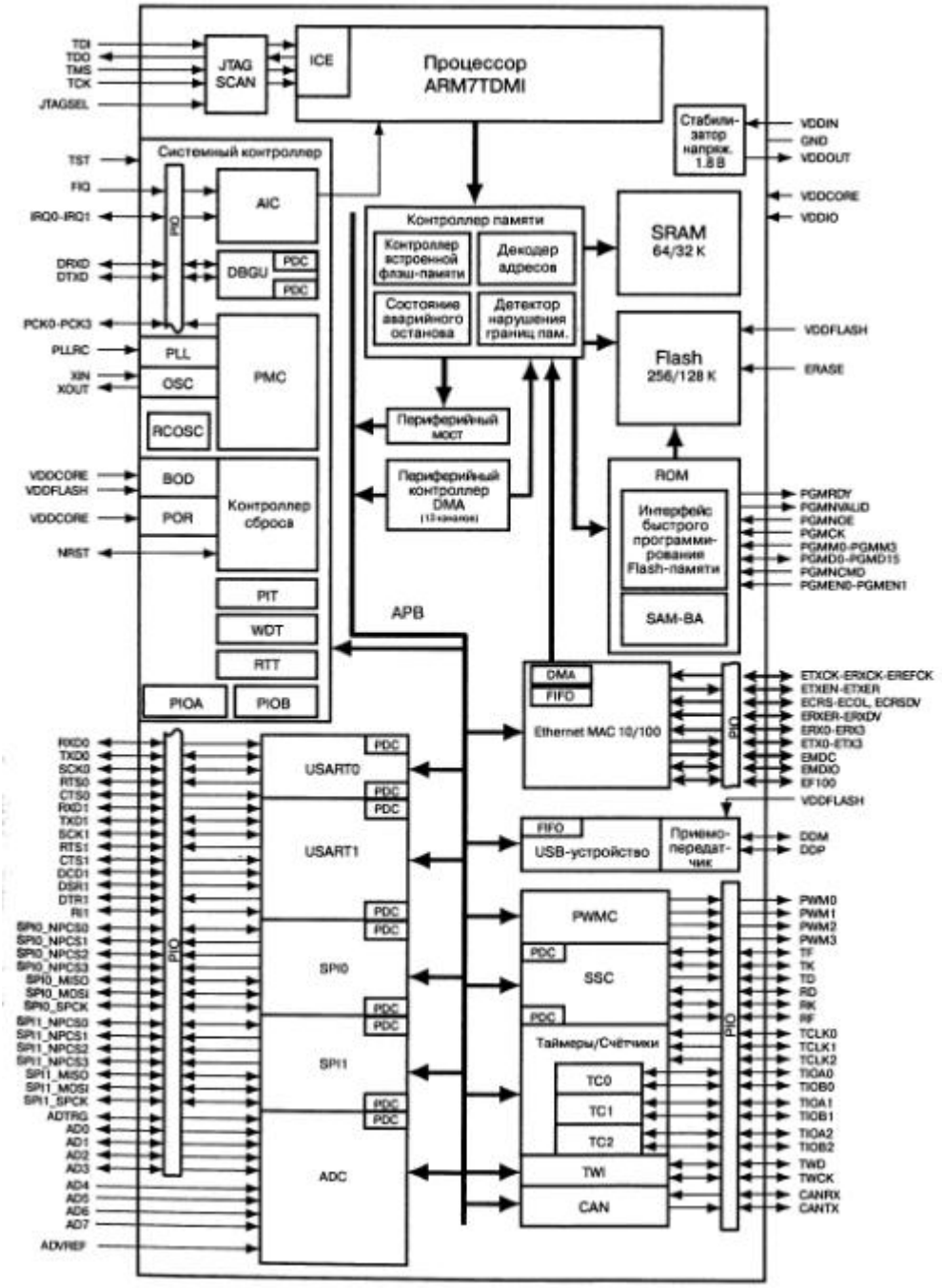


Рис. 1. Блок-схема микроконтроллера AT91SAM7S256/128

Рис. 2. Топологические схемы 64-выводных корпусов LQFP, QFN

Память микроконтроллера представлена тремя типами: Flash, SRAM и ROM.

Flash-память имеет объем 256 Кбайт, включая 1024 страницы по 256 байт в каждой. Flash-память имеет следующие основные характеристики:

- доступ к памяти за один цикл на частоте 30 МГц в самых плохих условиях;
- время программирования страницы, включая автостирание страницы − 6 мс;
- время программирования страницы без автостирания - 3 мс;
- время полного стирания микросхемы - 15 мс;
- ресурс: 10 000 циклов записи, 10-летняя сохранность данных;
- режим защиты содержимого Flash-памяти.

Область Flash-памяти во всех случаях отображается с адреса 0x0010 0000. Она также может быть доступна с адреса 0x0 после сброса и до выполнения команды Remap.

Встроенная Flash-память содержит 256-байтовый буфер записи, доступный через 32-битовый интерфейс.

Быстродействующая память SRAM имеет объем 64 Кбайт. Доступ к памяти SRAM обеспечивается за один цикл в режиме максимального быстродействия. После сброса до выполнения команды переотображения памяти (Remap) область SRAM доступна для программы только с адреса 0x0020 0000. После выполнения команды Remap область SRAM становится доступна с адреса 0x0.

Память ROM содержит заводские программы, FFPI (интерфейс быстрого программирования Flash-памяти) и стартовый загрузчик SAM-BA. Во всех случаях область ROM отображается с адреса 0x30 0000.

Для управления загрузкой из ROM (по умолчанию) или Flash-памяти используется бит общего назначения NVM (GPNVM), Этот бит может быть сброшен или установлен командами «Сбросить бит General-purpose NVM» и «Установить бит General-purpose NVM». Установка бита 2 GPNVM выбирает начальную загрузку из Flash. Подача сигнала на вывод ERASE сбрасывает бит 2 GPNVM и восстанавливает начальную загрузку из ROM.

Контроллер памяти содержит:

• программируемый арбитр шины, обрабатывающий запросы от процессора ARM7TDMI и контроллеров Ethernet MAC, DMA;

• дешифратор адресов, вырабатывающий сигналы выборки для трёх внутренних областей памяти размером по 1 Мбайт и одной области встроенной периферии общим размером 252 Мбайт;

• регистры состояния аварийного останова, сохраняющие источник, тип и все параметры доступа, которые привели к аварийному останову, и обеспечивающие при отладке обнаружение сбойных (некорректных) указателей;

• детектор нарушения границ памяти, выполняющий проверку корректности всех типов доступа к данным и генерирующий аварийный останов при нарушении границ;

• контроллер встроенной Flash-памяти.

Периферийный контроллер DMA обрабатывает перемещение данных между периферийными устройствами и блоками памяти. Контроллер имеет тринадцать каналов (по два для двух модулей USART, два для модуля отладки, два для последовательного синхронного контроллера SSP, по два для двух каналов SPI, один для АЦП).

Встроенные средства тестирования и отладки представлены интегрированным модулем EmbeddedICE (встроенным внутрисхемным эмулятором), модулем отладки DBGU, средствами граничного сканирования всех цифровых выводов в соответствии со стандартом IEEE 1149.1 JTAG.

2. Системный контроллер

Системный контроллер управляет всеми «жизненно важными» функциями МК: прерываниями, тактированием, питанием, заданием временных интервалов, отладкой и сбросом. Все периферийные устройства системного контроллера отображены в область верхних 4 Кбайт адресного пространства МК, расположенную между адресами OxFFFF F000 - OxFFFF FFFF.

Контроллер сброса RSTC содержит узел, вырабатывающий сигнал сброса NRST требуемой длительности при включении питания (POR), и детектор ″провалов″ напряжения питания (BOD).

Узел тактовых частот (рис. 3) AT91SAM7S256/128 имеет в своём составе RC-генератор с низким энергопотреблением, основной генератор и систему ФАПЧ (PLL) со следующими характеристиками:

- частота RC-генератора находится в диапазоне 22...42 кГц;
- частотный диапазон основного генератора составляет 3...20 МГц;
- основной генератор может быть исключён из схемы тактирования;
- выходной частотный диапазон ФАПЧ составляет 80... 200 МГц.

Узел тактовых частот обеспечивает выработку сигналов SLCK, MAINCK и PLLCK, используемых для тактирования узлов МК.

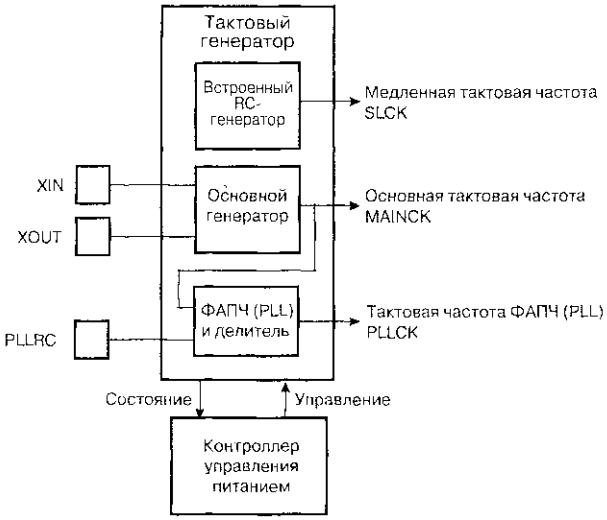


Рис. 3. Блок-схема узла тактовых частот

Контроллер управления питанием PMC. Поскольку мощность по питанию, потребляемая микроконтроллером, зависит от тактовой частоты, то для управления питанием различных узлов МК пользователь имеет возможность выбора (программирования) их тактовых частот. Контроллер PMC использует выходы узла тактовых частот для выработки следующих сигналов:

- тактовая частота процессора (ядра) PCK;
- частота задающего генератора MCK;
- частота синхронизации USB UDPCK;
- все периферийные независимо управляемые синхросигналы;
- четыре программируемых выхода синхросигналов.

Тактовая частота процессора PCK выключается при переходе микроконтроллера в режим пониженного энергопотребления (Idle) для уменьшения потребляемой мощности в ожидании прерывания.

Расширенный контроллер прерываний AIC имеет следующие свойства:

• управляет линиями прерываний nIRQ и nFIQ процессора ARM.

• индивидуально маскируемые векторизованные источники прерываний:

- быстрое прерывание FIQ, зарезервированное для источника 0;

- прерывание IRQ1, резервируеме для системных периферийных устройств (RTT, PIT, EFC, PMC, DBGU и т. д.). Остальные прерывания используются для устройств ввода-вывода (внешние прерывания);

- программируемые на запуск по фронту или уровню сигнала прерывания от внутренних источников;

- программируемые на запуск по переднему или заднему фронту сигнала, или по высокому или низкому уровню сигнала от внешних источников.

• 8-уровневый приоритетный контроллер, который:

- управляет прерыванием nIRQ процессора;

- задает приоритет источников прерываний;

- обеспечивает обслуживание более приоритетных прерываний во время обработки менее приоритетных прерываний.

• векторизация, позволяющая оптимизировать группу подпрограмм обработки прерываний и их выполнение. Имеется один 32-битовый векторный регистр для источников прерываний, через который считывается соответствующий запросу текущий вектор прерывания;

• защищенный режим предоставляет простой способ отладки, предотвращающий выполнение автоматических операций;

• форсирование прерывания до быстрого, позволяющее назначить любому из обычных прерываний режим обработки быстрого прерывания;

• маска прерываний обеспечивающая синхронизацию процессора от событий прерывания.

Периодический интервальный таймер PIT представлен программируемым 20-битовым счётчиком и 12-битовым интервальным счётчиком.

Сторожевой таймер WDT содержит защищенный ключом 12-битовый программируемый счётчик, производящий счёт предварительно делённой частоты SLCK и вырабатывает сигналы сброса или прерывания в системе. Счётчик может быть остановлен, когда процессор находится в режиме отладки или в спящем (Idle) режиме.

Таймер реального времени RTT содержит 32-разрядный счётчик со схемой тревоги, производящий счёт предварительно делённой программируемым 16-разрядным делителем частоты SLCK для компенсации нестабильности.

Каждый из контроллеров параллельного ввода/вывода PIOA, PIOB имеет следующие особенности.

• управляет 32 линиями ввода/вывода.

• полностью программируется путем установки/сброса разрядов регистров.

• мультиплексирует две периферийные функции на одну линию ввода/вывода.

• для каждой линии ввода/вывода:

- формирует прерывание при изменении входного состояния;

- фильтрует импульсные помехи длительностью до половины периода тактовой частоты;
- поддерживает мультидрайверную организацию, разрешая выходы с открытым стоком;
- имеет программируемый, «подтянутый к плюсу питания» резистор;
- регистр состояния данных вывода показывает значение уровня сигнала на выводе.

• синхронный вывод обеспечивает сброс и установку линий ввода/вывода одной командой записи.

3. Периферийные устройства

Каждому периферийному устройству микроконтроллера выделено 16 Кбайт адресного пространства.

Последовательный периферийный интерфейс SPI поддерживает связь с внешними последовательными устройствами памяти типа DataFlash и трехпроводными EEPROM, периферийными устройствами типа АЦП, ЦАП, LCD-контроллерами, CAN-контроллерами и датчиками (сенсорами), внешними сопроцессорами. Четыре выхода выбора микросхемы и внешний декодер позволяют подключить до 15-и внешних периферийных устройств.

В режиме ведущего или ведомого интерфейс можно программировать для обмена 8- или 16-разрядными словами данных с выбранным устройством, задавая частоту и полярность синхросигналов, задержку между сеансами последовательной передачи, а также между синхросигналами и данными, выбирая режим детектирования ошибок.

Двухпроводной интерфейс TWI поддерживает только режим ведущего и совместим со стандартными двухпроводными последовательными устройствами памяти.

Интерфейсы USART0, USART1 обеспечивают прием и передачу (5–9)-битовых данных при полнодуплексном синхронном или асинхронном последовательном обмене данными. Каналы содержат средства управления и поддержки: модема (в USART0), многоточечных режимов с генерацией и детектированием адресов, RS-485, IS07816 протокола T0 или T1 для взаимодействия со смарт-картами, режима IrDA при скоростях до 115,2 Кбит/с. Интерфейсы имеют возможность аппаратного подтверждения установления связи и осуществлять тестовые режимы «местный шлейф», «удалённый шлейф» и «автоматическое эхо».

Последовательный синхронный контроллер SSC обеспечивает последовательную синхронную передачу данных, используемых в аудио- и телекоммуникационных приложениях.

Таймер/счётчик используется для измерения частоты, подсчёта событий, измерения интервалов, генерации импульсов, синхронизации времени задержки, широтно-импульсной

модуляции, счёта на возрастание/убывание. Таймер содержит три 16-разрядных канала с тремя выходами сравнения и двумя входами захвата. Каждый канал может конфигурироваться пользователем и содержит три внешних тактовых входа и пять внутренних, включая два многоцелевых сигнала ввода/вывода. Обслуживают каналы два общих регистра.

Контроллер широтно-импульсной модуляции PWM представлен четырьмя каналами с одним 16-разрядным счётчиком в каждом канале и общим для всех каналов задающим генератором, обеспечивающим генерацию тринадцати различных частот, используя для этого один счётчик по модулю *n*, используемый для генерации одиннадцати значений частоты, и два независимых линейных делителя, подключаемых к выходам счётчика.

Все каналы программируют независимо друг от друга: разрешая или запрещая работу канала, выбирая тактовую частоту, задавая период и длительность импульса с двойной буферизацией, выбирая полярность выходных импульсов и выравнивая по центру или по левой границе.

Контроллер USB содержит полноскоростной (FS) модуль USB V2.0 со скоростью обмена 12 Мбит/с.

Аналого-цифровой преобразователь АЦП содержит:

- 8-канальный преобразователь последовательного приближения с разрешением 10 бит, производительностью 384 KSPS;
- встроенный мультиплексор, который обеспечивает подачу на вход АЦП восьми аналоговых сигналов напряжением до 3,3 В, четыре из которых разделяют выводы микроконтроллера с цифровыми сигналами;
- внешний источник опорного напряжения для обеспечения более высокой точности преобразования при малых входных напряжениях.

Встроенный узел управления преобразователя обеспечивает автоматическое «пробуждение» от триггера запуска и возврат в «спящий» (sleep) режим после завершения преобразований во всех разрешённых каналах.

Более подробную информацию об устройствах микроконтроллера можно найти в /1/.

4. Компиляция и программирование микроконтроллера AT91SAM7

Установка компилятора WinARM

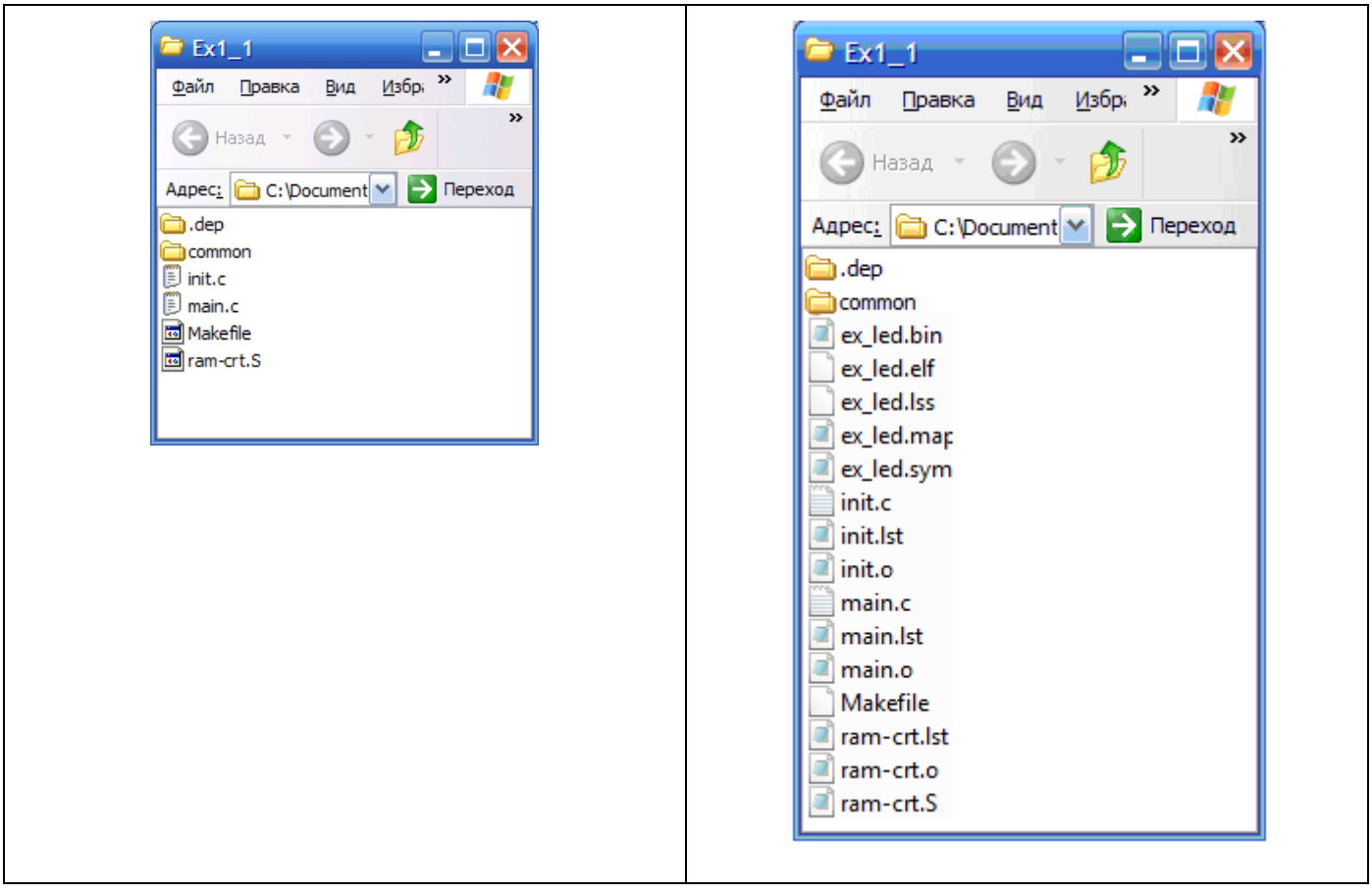
Для написания и последующей компиляции программного кода на языке Си для микроконтроллеров SAM7S используем специализированный редактор и GCC-компилятор WinARM (*http://atmel.argussoft.ru/software/AT91/GNU/winarm/*).

Устанавливаем самораспаковывающийся архив на диск C:/WinARM/. Для того, чтобы системная команда *make* и другие выполнялись из командной строки, в системную переменную PATH добавляем пути: *C:\WinARM\bin; C:\WinARM\utils\bin*.

Структура Makefile и компиляция проекта

Процесс создания программы состоит из написания исходных текстов программы, компиляции и получения *bin*-файла для загрузки в память микроконтроллера и исполнения.

Для компиляции проекта необходимо в папке проекта разместить файлы с исходным кодом и сборочный файл *Makefile*. Минимально необходимое содержимое папки проекта представлено на рис. 4.а. Файл *init.c* содержит начальные установки базовых устройств микроконтроллера (контроллеров управления питанием PMC и сброса RSTC, сторожевого таймера и др.). Сборочный файл содержит директивы и настройки компилятора и линковщика, а также ряд дополнительных директив для самой команды *make*.



а) б)

Рис. 4. Содержимое папки проекта до (а) и после компиляции (б).

Для адаптации исходного файла *Makefile* из программного комплекса WinARM к разрабатываемому проекту необходимо внести изменения в некоторые секции данного файла.

1) Тип памяти для исполняемого файла определяется директивой:

`RUN_MODE = ROM_RUN` для памяти ROM или

`RUN_MODE = RAM_RUN` для памяти SRAM

2) Имя выходного бинарного файла *bin* определяется директивой:

```
TARGET = ex1
```

3) Список исходных файлов на языке Си указывают в секции:

```
SRC  = init.c
SRC += main.c
```

4) Исходные программы на ассемблере, которые должны компилироваться в режиме ARM, указываются директивой:

```
ASRCARM  = ram-crt.S [5]
```

5) В общей папке *Common* располагаются файлы с описаниями платы фирмы-производителя Olimex, микроконтроллеров семейства SAM7 и его периферийных устройств и стандартные определения, используемые при написании программ (Board.h, at91s.h, AT91SAM7S256.h, lib_AT91SAM7S256.h, SAM7-P256.h, project.h и др.). Доступ к папке описан в *Makefile* директивами *EXTRAINCDIRS = common/, EXTRA_LIBDIRS = common/*.

Для выполнения компиляции открываем командную строку: *Пуск-Выполнить-cmd*. Используя команду *cd,* выполняем переход в папку проекта. Применив команду *make,* выполняем компиляцию. Если на этой стадии обнаружены ошибки или необходимо исправить исходный код проекта, рекомендуется очистить папку проекта от результатов предыдущей компиляции, чтобы предотвратить ошибки, связанные с использованием файлов старых версий. Для этого в папке проекта выполним команду *make clean*. После выполнения команды все временные файлы будут удалены из папки проекта. Сделав исправления, повторяем команду *make*. При отсутствии ошибок можно переходить к следующему этапу – загрузке программы в память микроконтроллера. Папка проекта после компиляции имеет вид (рис. 4,б).

Удобным инструментом при подготовке программ является текстовый редактор Notepad Programmer (NP) из пакета WinARM. После запуска программы можно создать проект и добавить в него исходные файлы программных модулей. В папку проекта следует поместить *Makefile.* Открыв его в окне редактора NP, вносим изменения в секции файла (с директивами TARGET=, SRC=, RUN_MODE=, ASRCARM=) и сохраняем. Вышеупомянутая папка *Common* помещается в папку проекта.

Для того, чтобы выполнить компиляцию, не выходя из окна программы NP, следует предварительно выполнить дополнительную настройку программы NP. Для этого необходимо открыть окно *Options*, выполнив команды меню *Tools\Options* и выбрать слева строку *Tools*. Выбрав вверху поле (None-Global Tools), добавляем копкой *Add* две команды **make *all*** и **make *clean***, указав ссылки на общую программу C:\WinArm\utils\bin\make.exe и параметр *clean* во втором случае, и путь к папке проекта. Открыв вкладку *Consol I/O*, включаем флажок *Capture Output* для вывода результатов компиляции в окно вывода PN. Завершив настройку, получим в меню *Tools* две команды, одну – для обработки *Makefile* (компиляции проекта), вторую – для удаления из папки проекта файлов предыдущей компиляции. Выполнив компиляцию (*Tools\make all*) через окно сообщений можно найти обнаруженные при компиляции ошибки и исправить их в окне редактора. При отсутствии ошибок можно приступить к проверке работы программы.

Установка загрузчика

Используем инсталлятор Install AT91-ISP v1.11.exe системы управления внутренним программатором (In-system Programmer, ISP) для микроконтроллеров на базе AT91SAM7 и AT91SAM9 (http://www.atmel.com). Запускаем инсталлятор, следуя установочным инструкциям. AT91-ISP включает:

- SAM-BA;
- SAM-PROG;
- две библиотеки на базе AT91Boot_DLL: AT91Boot_DLL.dll и AT91Boot_TCL.dll, что позволяет использовать AT91Boot_DLL.dll в командной строке TCL (TCL shell - TCLSH);
- документацию по AT91-ISP;
- примеры проектов OLE_MFC и OLE_without_MFC Visual C++ 6.0, демонстрирующие использование AT91Boot_DLL при разработке собственных ISP приложений;
- CAN_TCLSH-пример, показывающий примеры Flash-программирования через CAN;
- примеры TCL-скриптов.

«Помощник» начального загрузчика SAM-BA

«Помощник» начального загрузчика SAM-BA − вызываемая по умолчанию заводская загрузочная программа, которая обеспечивает простой способ программирования встроенной Flash-памяти. SAM-BA поддерживает последовательную передачу данных через DBGU или порт USB устройства.

Связь через DBGU поддерживает широкую номенклатуру тактовых частот кварцевых резонаторов от 3 до 20 МГц с помощью программного автоматического определения частоты. Связь через порт USB возможна только с кварцевым резонатором с частотой 18,432 МГц.

Начальный загрузчик SAM-BA имеет графический интерфейс пользователя SAM-BA (GUI). Этот интерфейс представляет собой компьютерную программу, свободно распространяемую производителем МК. Код начального загрузчика размещается в ROM и отображён в памяти с адреса 0x0 при сброшенном бите GPNVM.

Библиотека AT91Boot_DLL

AT91Boot_DLL подключается к микроконтроллерам на основе AT91SAM через USB-порт, последовательный порт или JTAG с использованием SAM-ICE или JLink JTAG переходника.

В зависимости от того, какой порт выбран для связи, плата должна быть в следующем состоянии:

- при использовании порта USB или последовательного порта DBGU на плате должен быть запущен загрузчик SAM-BA (Boot Assistant);
- при использовании CAN-порта на плате должен быть запущен загрузчик SAM-BA Boot4CAN;
- при использовании JTAG через SAM-ICE или JLink плата может быть в неопределенном состоянии. В этом случае конфигурирование платы (PLL и др.) возлагается на пользователя.

Принцип работы библиотеки AT91Boot_DLL прост.

1. Сканирование всех устройств, подключенных к компьютеру.
2. Установка связи с выбранным устройством.
3. Выполнение всех требуемых действий, таких как запись во Flash-память.
4. Разрыв связи.

Загрузка программы в память микроконтроллера

Рассмотрим последние этапы создания программы – загрузку и выполнение *bin*-файла. Когда библиотека установлена, можно подключить устройство к компьютеру. Используем USB-кабель. Компьютер обнаруживает новое оборудование и появляется предложение его установить. Соглашаемся на автоматический поиск подходящего драйвера. Если AT91-ISP был установлен, подходящий драйвер будет найден, так как он поставляется с инсталлятором AT91-ISP.

Только после того, как устройство подключено к компьютеру и определено в системе, запускаем программу SAM-BA (GUI) из пакета AT91-ISP. Эта программа поможет нам загрузить *bin*-файл во FLASH или SRAM память устройства.

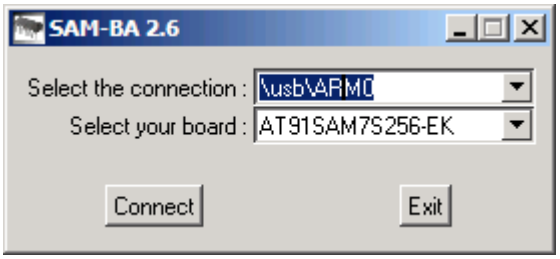


Рис. 5. Соединение с платой через SAM-BA

В окне типа соединения (рис. 5) появляется \usb\ARMx, в строке типа платы выбираем нашу модель. Если USB-подключения нет в списке возможных, плата не определилась в системе, драйвер не установлен или программа запущена раньше подключения устройства.

Список соединений содержит дополнительно записи:

″\jlink\ARMx″, выбираемую, если устройство подключено через SAM-ICE/JLink,

″COMx″, если устройство подключено через COM-порт.

При работе с платой мы используем USB-подключение. В этом случае питание платы осуществляется от шины USB хоста.

Выбираем память (Flash/SRAM), в которую хотим загрузить *bin*-файл (рис.6). Файл должен быть специально подготовлен для загрузки в соответствующую память. Тип памяти, для которой компилируется программа, указывается в Makefile.

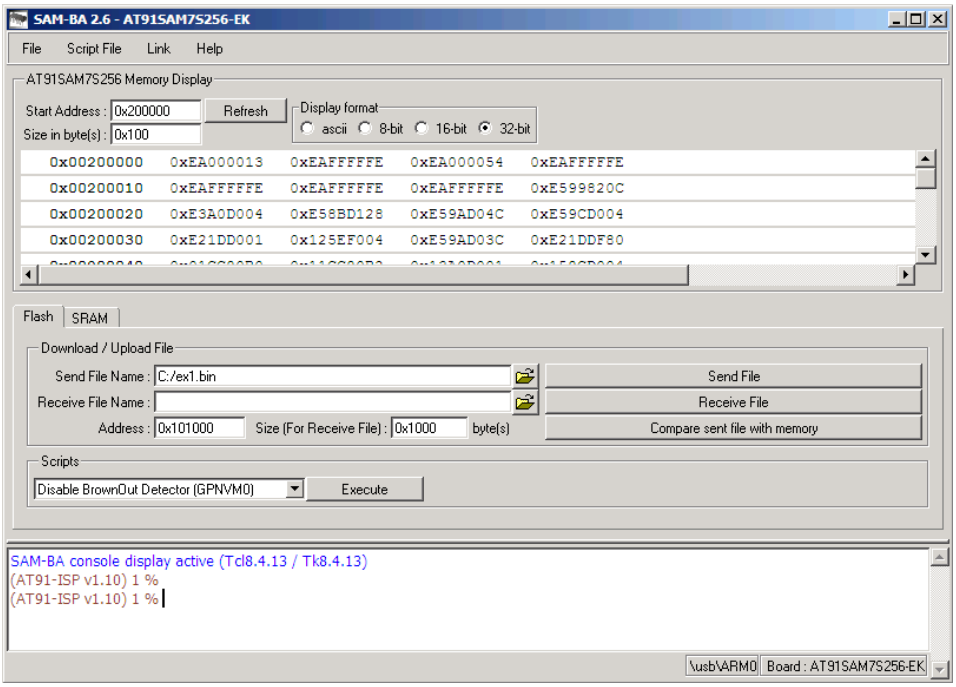


Рис. 6. Главное окно SAM-BA

Далее выбираем *bin*-файл, который будем загружать. В поле Address указываем адрес по которому хотим загрузить исполняемый код. Нужно помнить, что в памяти в двух первых секторах уже хранится ISP загрузчик для SAM-BA (о нем ниже), благодаря которому мы сейчас связываемся с платой. Поэтому его лучше не трогать и загружать программу в свободную область памяти, так как процедура восстановления SAM-BA Boot Assistant в памяти контроллера нежелательна к частому применению. Size - размер обрабатываемого кода, важен при процедуре получения кода из платы.

При тестировании программ PIO использовались адреса Flash 0x101000, в остальных случаях − адреса SRAM 0x202000.

Нажимаем кнопку *Send File* и получаем предложение переписать защитные биты, от чего отказываемся. Нажимаем кнопку *Compare sent file with memory* для проверки соответствия отправленного файла и памяти платы. Полученное сообщение о соответствии означает удачную загрузку (без ошибок). Теперь код находится в памяти платы и готов к запуску.

Запуск программного кода

Не выходя из предыдущего окна, введем в командную строку TCL команду с начальным адресом программы:

TCL_Go $target(handle) 0x202000

Если программа не имеет возврата в SAM-BA Boot Assistant, связь с программой будет потеряна, плата будет выполнять программу независимо, повторный запуск с компьютера невозможен. Для повторного запуска необходимо, закрыв программу SAM-BA, выполнить на плате сброс (RESET) и запустить SAM-BA (GUI) на компьютере заново.

Восстановление загрузчика

Одной из причин, по которым плата может не соединяться с компьютером, может быть затирание ISP SAM-BA загрузчика в памяти микроконтроллера. Для его восстановления необходимо отключить питание платы, замкнуть джампер TST, подать питание на плату и выждать 10 с. За это время происходит перезапись кода начального загрузчика из ROM во Flash-память микроконтроллера. Затем, снова отключив питание платы, нужно разомкнуть джампер TST и вновь подать питание на плату. ISP SAM-BA загрузчик загружен в два первых сектора Flash и готов к использованию. Теперь можно использовать SAM-BA (GUI) для связи с SAM-BA загрузчиком (Boot Assistant). Подробно работа SAM-BA описана в SAM-BA User Guide, доступном в режиме "on-line".

II. Примеры проектов для работы с периферийными устройствами

1. Контроллер ввода-вывода PIO

В состав микроконтроллера AT91SAM7S256 входит контроллер ввода/вывода PIO, который выполняет базовые функции ввода/вывода.

Контроллер PIO управляет 32-мя полностью программируемыми линиями ввода/вывода. Каждая линия ввода/вывода может быть определена как линия общего назначения, либо как линия, управляемая одним из встроенных периферийных устройств, и связана с соответствующим битом каждого 32- разрядного управляющего регистра.

Контроллер PIO позволяет:

- отслеживать изменение состояния каждой линии ввода/вывода;
- подавлять дребезг, не воспринимая внешние импульсы длительностью меньше половины периода сигнала синхронизации;
- управлять подтягивающими резисторами каждой линии ввода/вывода.

1.1. Функциональное описание контроллера

Логическая схема управления линиями ввода/вывода контроллером PIO представлена на рис. 7. Она описывает работу лишь одной из 32-х линий ввода/вывода, к остальным линиям можно обратиться, изменив индексы регистров на соответствующий номер линии.

Управление подтягивающими резисторами. У каждой линии есть встроенный подтягивающий резистор сопротивлением около 10 кОм, который может быть подключен к плюсу питания или отключен путем записи 1 в соответствующий разряд регистров PIO_PUER (Pull-up Enable Register) и PIO_PUDR (Pull-up Disable register). Запись 1 в эти регистры приводит к изменению содержимого регистра PIO_PUSR (Pull-up Status Register). Если разряд регистра PIO_PUSR содержит 1, подтягивающий резистор соответствующей линии подключен; если 0, резистор отключен. После сброса микроконтроллера все подтягивающие резисторы отключены (чтение регистра PIO_PUSR возвращает значение 0x0).

Управление функциональным назначением выводов. Так как каждый вывод может мультиплексировать несколько функций, то для выбора источника данных функций используются регистры управления PIO_PER (PIO Enable Register) и PIO_PDR (PIO Disable Register). Установка какого-либо бита в регистрах PIO_PER и PIO_PDR приводит к изменению содержимого регистра PIO_PSR (PIO Status Register). Нулевое значение бита в регистре PIO_PSR указывает на то, что соответствующий вывод будет контролироваться одним из встроенных периферийных устройств, при этом выбор устройства определяется регистром PIO_ABSR (AB Select Register). Если же бит в регистре PIO_PSR установлен в 1, то соответствующий вывод будет контролироваться контроллером PIO. При сбросе микроконтроллера в регистр PIO_PSR записываются все единицы (за исключением некоторых случаев, оговоренных производителем), то есть выводы настраиваются на режим управления от контроллера PIO.

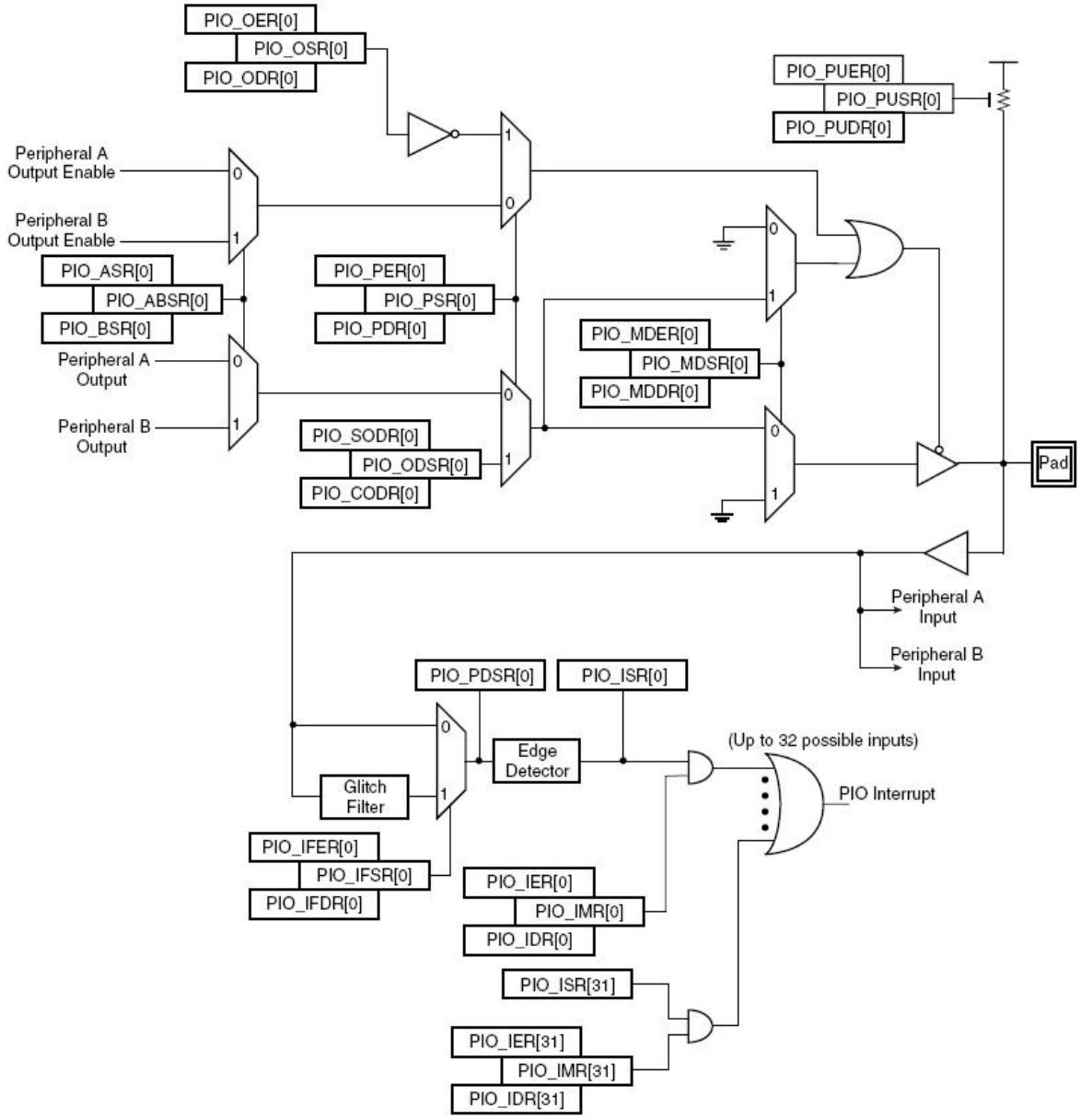


Рис. 7. Логическая схема управления линией ввода/вывода

Управление выходами. Если линия ввода/вывода настроена на какую-либо периферийную функцию (соответствующий бит в регистре PIO_PSR установлен в 0), то выбор периферийного устройства определяет содержимое регистра PIO_ABSR.

Если же линия ввода/вывода контролируется контроллером PIO, то появляется возможность определить, будет она управляемой или нет. Это можно сделать путем установки соответствующего выбранной линии бита в регистрах PIO_OER (Output Enable Register) и PIO_ODR (Output Disable Register). В результате изменяется содержимое регистра PIO_OSR (Output Status Register). Если бит регистра PIO_OSR сброшен в 0, то соответствующая линия

ввода/вывода используется для ввода информации; если бит установлен в 1, линия управляется контроллером PIO.

Уровень сигнала на линии (высокий или низкий) может быть задан путем записи 1 в соответствующий бит регистров PIO_SODR (Set Output Data Register) и PIO_CODR (Clear Output Data Register). Эти операции приводят к записи 1 или 0 в аналогичный бит регистра PIO_ODSR (Output Data Status Register), который непосредственно задает уровни сигналов на линиях ввода/вывода.

Входы. Текущий уровень сигнала, установленный на линиях ввода/вывода, может быть получен с помощью регистра PIO_PDSR (Pin Data Status Register). Этот регистр содержит информацию о значениях выводов независимо от того, как они сконфигурированы (на ввод, на управление контроллером PIO или одним из периферийных устройств). Заметим, чтение состояний линий ввода/вывода может осуществляться, только если разрешена подача на контроллер PIO синхроимпульсов от контроллера управления питанием PMC.

Фильтрация помех. Каждую линию ввода/вывода, независимо от других, можно настроить на подавление помех. Когда схема подавления помех включена, входные импульсы длительностью меньше 1/2 такта сигнала синхронизации MCK не воспринимаются. Воспринимаются лишь те импульсы на входе, длительность которых больше одного такта MCK. Для импульсов длительностью от 1/2 до 1 такта MCK возникает неопределенность, т.е. они могут быть восприняты или проигнорированы в зависимости от точного момента их появления.

Схема подавления помех управляется с помощью регистров PIO_IFER (Input Filter Enable Register), PIO_IFDR (Input Filter Disable Register) и PIO_IFSR (Input Filter Status Register). Запись в регистры PIO_IFER и PIO_IFDR приводит к установке в 1 или 0 соответствующих битов регистра PIO_IFSR, которые непосредственно включают или отключают схемы подавления помех каждой из линий ввода/вывода.

Подавление помех учитывается лишь при чтении регистра PIO_PDSR и при формировании признака прерывания на линии ввода/вывода. Для работы фильтров помех необходимо подать на контроллер PIO сигналы синхронизации от контроллера PMC.

Генерация сигнала прерывания по изменению уровня на линии. Контроллер PIO может быть настроен на генерацию прерывания при обнаружении изменения уровня сигнала на какой-либо из линий ввода/вывода. Для разрешения или запрета генерации прерывания используются регистры разрешения PIO_IER (Interrupt Enable Register) и запрещения прерываний PIO_IDR (Interrupt Disable Register), которые устанавливают или сбрасывают соответствующие биты регистра маски прерываний PIO_IMR (Interrupt Mask Register). Изменение уровня сигнала можно обнаружить

путем сопоставления двух состояний на входе, для этого следует разрешить подачу синхроимпульсов МСК на контроллер PIO. Разрешение прерываний возможно при настройке линии на ввод, управляемый контроллером PIO, либо назначенной для периферийной функции.

При обнаружении изменения уровня на одной из линий ввода/вывода устанавливается соответствующий бит регистра PIO_ISR (Interrupt Status Register). Если для этой линии разрешены прерывания, т.е. установлен соответствующий бит в регистре PIO_IMR, то генерируется сигнал прерывания. При этом прерывания от всех 32-х линий ввода/вывода объединяются по ИЛИ и затем подаются на контролер прерываний AIC. При чтении содержимого регистра PIO_ISR автоматически сбрасываются все сигналы прерываний.

1.2. Программирование контроллера ввода-вывода

Функции вывода контроллера PIO

Для демонстрации базовых функций вывода контроллера PIO подготовим программу, которая осуществляет мигание одного из индикаторов на плате. Перед началом работы с индикаторами необходимо настроить соответствующие линии ввода/вывода контроллера PIO. Для этого их нужно подключить, настроить на вывод, а также отключить подтягивающие резисторы. Для того, чтобы увидеть мигание индикатора, необходимо в программу ввести процедуру временной задержки, например на 80000 тактов. Текст основной программы, представляющий содержимое файла *main.c,* представлен в листинге 1. Первая строки программы подключает базовую библиотеку определений для микроконтроллера (AT91SAM7S256.h), определенную производителем микроконтроллера (в данном случае описание фирмы Atmel) с именами и адресами регистров модулей. Вторая строка подключает описание SAM7-P256.h платы с установленным микроконтроллером и небольшой периферией, выпущенной фирмой Olimex (www.olimex.com). Далее описаны процедуры задержки *delay*, инициализации индикаторов *led_config* и главная функция *main.* В Makefile помимо главного (main.c) src-файла указан дополнительный (init.c).

Листинг 1. *Программирование контроллера PIO при выводе данных*

```
//============================================================================//
//          Мигание индикаторов.
//============================================================================//
#include "AT91SAM7S256.h" // библиотека определений для AT91SAM7S256
#include "SAM7-P256.h"    // библиотека определений для периферии SAM7S-P256

void delay (int i);                       //формирование задержки
static void led_config (void);            //конфигурирование индикаторов

//процедура задержки-------------------------------------------------------------//
void delay (int n)
{
```

```
      int i;
      for (i=0; i<=n; i++)
      {
            asm("nop");
      }
}
//предварительная настройка индикаторов-----------------------------------------//
static void led_config (void)
{
      volatile AT91PS_PIO pPIO = AT91C_BASE_PIOA;  //указатель на контроллер PIO
      pPIO->PIO_PER = LED_MASK;   //разрешение работы индикаторов (выводы 18, 17)
      pPIO->PIO_OER = LED_MASK;  //выводы 18, 17 как выходы
      pPIO->PIO_PPUDR = LED_MASK;//отключение подтягивающих резисторов
      pPIO->PIO_CODR = LED2;     //включение индикатора 2 (вывод 17)
      pPIO->PIO_SODR = LED1;     //выключение индикатора 1 (вывод 18)
}

//главная процедура-------------------------------------------------------------//
int main()
{
      volatile AT91PS_PIO pPIO = AT91C_BASE_PIOA;//указатель на контроллер PIO
      led_config ();               //вызов процедуры настройки индикаторов
      while (1)                    //бесконечный цикл
      {
            pPIO->PIO_CODR = LED1;  //включение индикатора 1
            delay (80000);          //задержка
            pPIO->PIO_SODR = LED1;  //выключение индикатора 1
            delay (80000);          //задержка
      }
      return 0;
}
```

Задание 1. Изменить программу переключения светодиодов LED1 и LED2 так, чтобы частота переключения LED1 вдвое превышала частоту переключения LED2.

Обработка нажатия кнопок

Для демонстрации функции ввода данных контроллером PIO, подготовим программу, которая обрабатывает нажатие кнопок. Программа отслеживает нажатие кнопок SW1 и SW2 на плате SAM7-P256. При этом по нажатию кнопки SW1 скорость мигания индикатора LED1 будет увеличиваться, а по нажатию кнопки SW2 − уменьшаться. Текст программы представлен в листинге 2.

Листинг 2. *Программа обработки нажатия кнопок*

```
//===========================================================================//
//    Мигание индикаторов с регулировкой скорости мигания
//===========================================================================//

#include "AT91SAM7S256.h" // библиотека определений для AT91SAM7S256
#include "SAM7-P256.h"    // библиотека определений для периферии SAM7-P256

void delay (int i);                //формирование задержки
static void wate (void);           //процедура проверки нажатия кнопки
static void button_config (void);  //конфигурирование кнопок
static void led_config (void);     //конфигурирование индикаторов
```

```
int speed;                    //величина задержки для регулирования скорости мигания

//процедура формирования задержки-----------------------------------------------//
void delay (int n)
{
      int i;

      for (i=0; i<=n; i++)
      {
            asm("nop");
      }
}

//процедура изменения скорости мигания при нажатия кнопок ----------------------//
static void wait (void)
{
      unsigned int pio_pdsr;  //регистр состояния выводов контроллера PIO (PDSR)
      volatile AT91PS_PIO pPIO = AT91C_BASE_PIOA;//указатель на контроллер PIO

      pio_pdsr=pPIO->PIO_PDSR;  //текущее состояние выводов 0-31
      if ((pio_pdsr&SW1) == 0)
      {                          //если нажата кнопка SW1,
            speed=speed-20000;  // уменьшаем время паузы
      }
      else
      if ((pio_pdsr&SW2) == 0)
      {                          //если нажата кнопка SW2,
            speed=speed+20000;  // увеличиваем время паузы
      }
}

//процедура настройки кнопок----------------------------------------------------//
static void button_config (void)
{
      volatile AT91PS_PIO pPIO = AT91C_BASE_PIOA; //указатель на контроллер PIO
      volatile AT91PS_PMC pPMC = AT91C_BASE_PMC;  //указатель на контроллер PMC

      pPMC->PMC_PCER = (1<<AT91C_ID_PIOA); //разрешение синхронизации контроллера PIO
      pPIO->PIO_PER = SW_MASK;             //подключение кнопок (выводы 19, 20)
      pPIO->PIO_ODR = SW_MASK;             //выводы 19, 20 для ввода информации
      pPIO->PIO_IFER = SW_MASK;            //включение схемы подавления дребезга
}

//процедура настройки индикаторов-----------------------------------------------//
static void led_config (void)
{
      volatile AT91PS_PIO pPIO = AT91C_BASE_PIOA;  //указатель на контроллер PIO

      pPIO->PIO_PER = LED_MASK;           //подключение индикаторов (выводы 17, 18)
      pPIO->PIO_OER = LED_MASK;           //выводы 17, 18 для вывода информации
      pPIO->PIO_PPUDR = LED_MASK;         //отключение подтягивающих резисторов
      pPIO->PIO_CODR = LED2;              //включение индикатора 2
      pPIO->PIO_SODR = LED1;              //выключение индикатора 1
}

//главная процедура-------------------------------------------------------------//
int main()
{
```

```
      volatile AT91PS_PIO pPIO = AT91C_BASE_PIOA; //указатель на контроллер PIO

      button_config ();
      led_config ();
      speed = 80000;

      while (1)                          //бесконечный цикл
      {
            pPIO->PIO_CODR = LED1;       //включение LED1
            wait ();                     //проверка кнопок
            delay (speed);               // и задержка
            pPIO->PIO_SODR = LED1;       //выключение LED1
            wait ();                     //проверка кнопок
            delay (speed);               // и задержка
      }
      return 0;
}
```

Проверка нажатия кнопки осуществляется путем мониторинга состояния регистра PIO_PDSR.

Задание 2. Изменить программу, включив в нее проверку значения *speed* и, в случае выхода значения за границы выбранного диапазона, зажигая индикатор LED2.

2. Контроллер прерываний AIC

Расширенный контроллер прерываний AIC (Advanced Interrupt Controller) представляет собой векторизованный контроллер с восемью уровнями приоритета, позволяющий обрабатывать до 32-х индивидуально маскируемых источников прерываний.

2.1. Функциональное описание контроллера

Контроллер прерываний AIC управляет двумя входами процессора ARM7: быстрых прерываний nFIQ (Fast Interrupt Request) и стандартных прерываний nIRQ (Standard Interrupt Request). На входы контроллера AIC поступают сигналы либо от внутренних периферийных устройств, либо сигналы IRQ и FIQ от внешних выводов (рис. 8).

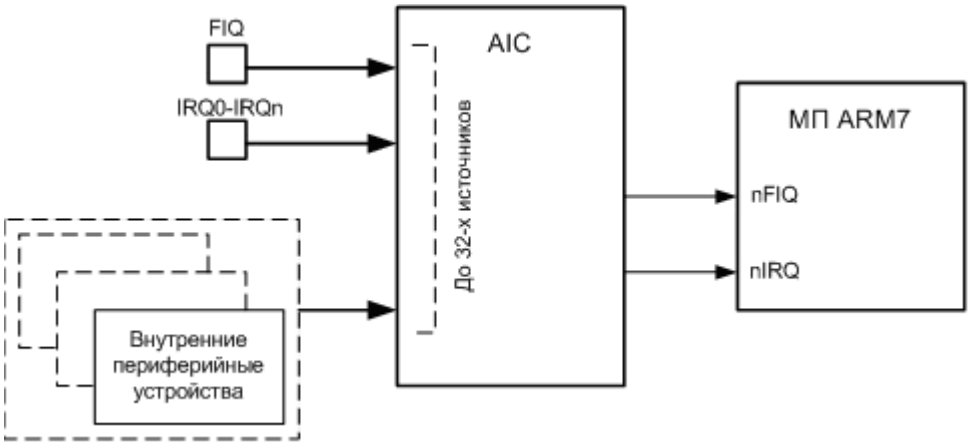


Рис. 8. Взаимодействие AIC и ARM7

8 - уровневая схема приоритетов позволяет пользователю задать приоритет для каждого источника прерываний. Таким образом, прерывания с большим приоритетом будут выполнены, даже если в этот момент уже началась обработка прерывания с меньшим приоритетом.

Внутренние прерывания могут быть запрограммированы на срабатывание по определенному уровню сигнала от источника, либо по изменению уровня. Внешние прерывания могут быть настроены на срабатывание по фронту или спаду, по высокому или по низкому уровню сигнала от источника.

Режим Fast Forcing контроллера AIC позволяет перенаправить все прерывания на обработку по типу быстрых прерываний.

Управление источником прерываний

Контроллер прерываний AIC позволяет независимо запрограммировать каждый источник прерывания. Поле SRCTYPE соответствующего регистра режима источника AIC_SMRn (Source Mode Register) определяет условие срабатывания прерывания для каждого из источников.

Внутренние источники прерываний могут быть настроены на чувствительность к уровню сигнала или перепаду. При этом конкретный уровень или перепад не играют роли. Внешние же источники можно настроить на чувствительность к высокому или низкому уровню сигнала, либо к положительному или отрицательному перепаду напряжения. Признак срабатывания прерывания от источников, работающих по перепаду, может быть установлен или сброшен с помощью регистров AIC_ISCR и AIC_ICCR. Запись в эти регистры для источников, чувствительных к уровню сигнала, не дает никакого эффекта. Операция записи в регистр AIC_ICCR используется для сброса признака срабатывания прерывания в контроллере AIC, запись в регистр AIC_ISCR – для установки признака.

Любое прерывание может быть разрешено или запрещено с помощью управляющих регистров AIC_IECR, AIC_IDCR (Interrupt Enable/Disable Command Register). При этом маска прерываний находится в регистре AIC_IMR (Interrupt Mask Register).

Контроллер прерываний AIC автоматически сбрасывает признак прерывания при чтении регистра вектора прерывания AIC_IVR (Interrupt Vector Register). При автоматическом сбросе запрещается прерывание, форсированное режимом Fast Forcing до FIQ прерывания. Автоматический сброс прерывания FIQ0 выполняется при чтении регистра AIC_FVR (Fast Vector Register). Схемы управления внутренними и внешними прерываниями представлены на рис. 9.

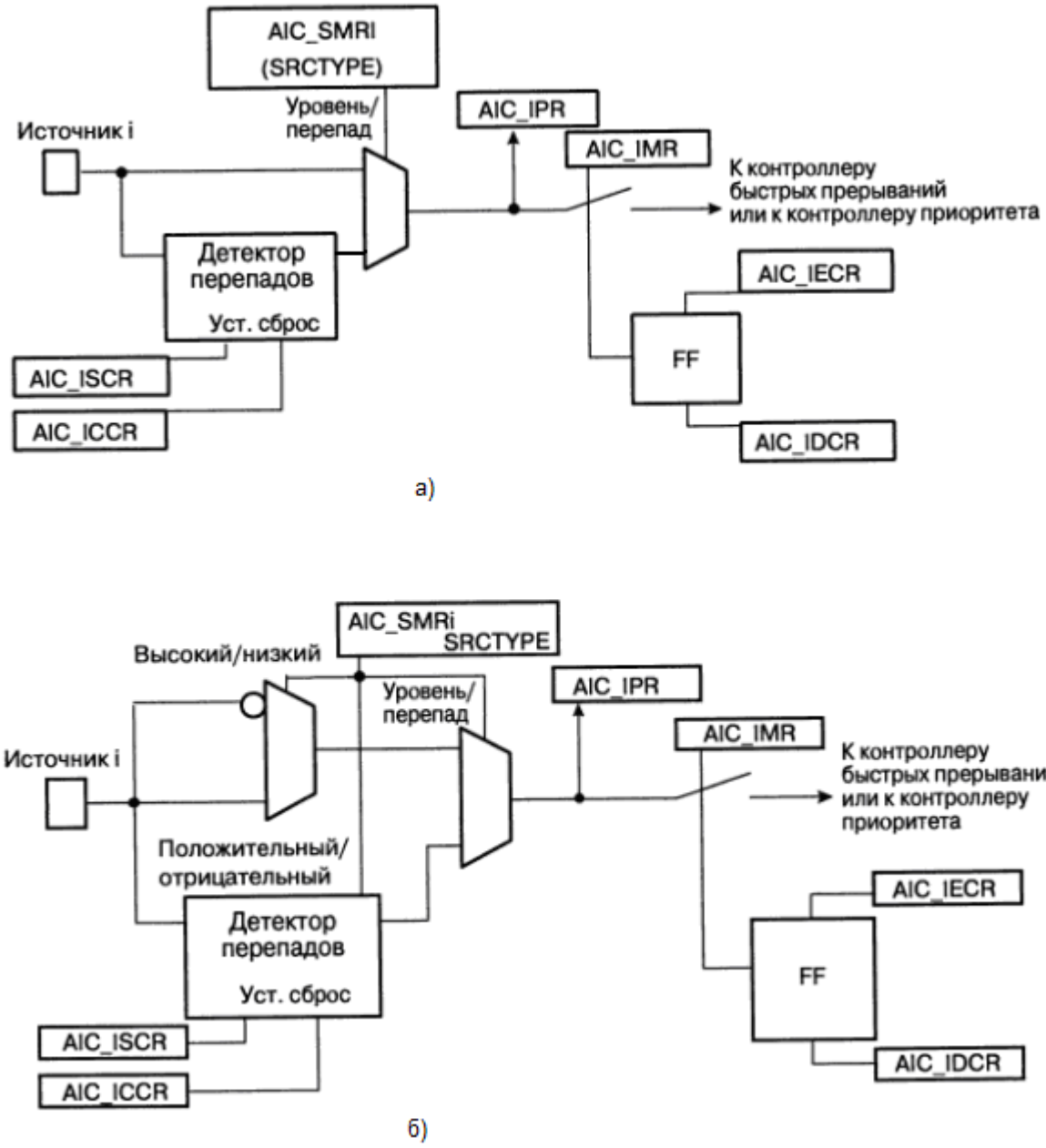


Рис. 9. Схемы управления внутренними (а) и внешними (б) прерываниями

Обычные IRQ прерывания

Под обычными прерываниями понимаются IRQ прерывания от источников с 1 по 31, т.е. (IRQ1 – IRQ31).

Контроллер приоритетов. Каждому источнику прерываний можно задать уровень приоритета от 0 до 7, это осуществляется путем записи требуемого приоритета в поле PRIOR регистра AIC_SMRn. Уровень 7 имеет высший приоритет, уровень 0 – низший.

Контроллер позволяет обрабатывать прерывания более высоких уровней во время обработки прерывания низкого уровня. Если во время обработки некоторого прерывания, возникает другое

прерывание с тем же приоритетом, контроллер сначала завершит обработку первого прерывания и, лишь, затем перейдет к обработке второго. Если во время обработки сработает прерывание с более высоким приоритетом, то контроллер сохранит текущее состояние процессора и перейдет на обработку нового прерывания, а после его завершения вернется к обработке первого прерывания.

Когда прерывание более высокого уровня прерывает обработку прерывания низкого уровня, происходит повторная активация линии nIRQ процессора. Если прерывания в этот момент разрешены на уровне ядра, то обработка текущего прерывания (с меньшим приоритетом) прерывается. При этом прерывание с большим приоритетом считывает свой вектор прерывания из регистра AIC_IVR. Это в свою очередь приводит к сохранению в аппаратном стеке номера первоначального прерывания и его приоритета. Тем самым обеспечивается возможность возврата к обработке первоначального прерывания по окончании обработки прерывания более высокого приоритета. Контроллер прерываний AIC имеет в своем составе 8-уровневый аппаратный стек, поддерживая обработку прерываний до 8 уровней вложенности.

Адреса обработчиков прерываний для определенных источников (невекторизованных) могут быть записаны в специальные регистры AIC_SVRn (n=1,..,31). Когда процессор считывает значение регистра AIC_IVR, то возвращается адрес обработчика прерывания, находящийся в регистре AIC_SVRn, соответствующем текущему прерыванию.

При этом логика AIC дает возможность выполнить переход на обработчик прерывания одной командой. Так как регистр AIC_IVR отображен по адресу 0xFFFF F100, то с учетом запаздывания обработки прерывания, во время которого адрес программного счетчика увеличивается на 8 (PC = 0x18 + 8 = 0x20), доступ к регистру AIC_IVR осуществляется помощью команды LDR PC, [PC,# -0xF20], размещаемой по адресу вектора прерываний ARM 0x0000 0018.

При возникновении прерывания процессор переходит по вектору данного прерывания к обработчику с помощью инструкции: LDR PC, [PC, # – 0xF20]. Выполняя данную инструкцию, процессор загружает в программный счетчик PC вектор прерывания, передавая управление обработчику. Схема обработки векторного прерывания приведена на рис. 10.

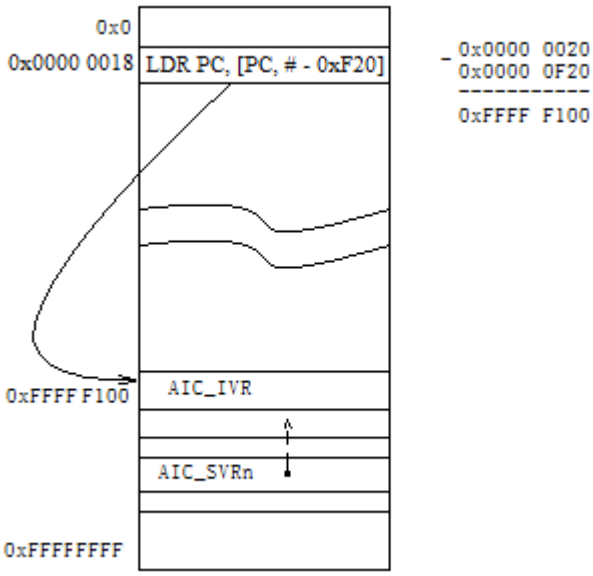


Рис. 10. Схема обработки прерывания IRQ

Обработка прерывания

Для работы с IRQ прерываниями следует выполнить некоторые предварительные действия:

- настроить должным образом контроллер прерываний AIC,
- загрузить в регистр AIC_SVR вектор прерывания и разрешить прерывания от заданного источника;
- задать инструкцию перехода к обработчику: LDR PC, [PC, # – 0xF20].

При наступлении прерывания, если бит микропроцессора I в регистре CPSR установлен в 0, выполняется следующая последовательность действий.

1) Процессор переходит в режим IRQ, содержимое регистра CPSR запоминается в регистре SPSR_irq, текущее значение программного счетчика сохраняется в регистре R14_irq, в PC загружается адрес 0x18. При записи текущего значения PC впоследствии при возврате в исходную программу его необходимо уменьшить на 4, чтобы начать выполнение той инструкции, на которой прервалось выполнение основного кода.
2) При выполнении инструкции по адресу 0x18 в программный счетчик загружается значение регистра AIC_IVR. Помимо этого:
    - устанавливается признак обработки прерывания в регистре AIC_IPR (Interrupt Pending Register) и разрешаются прерывания более высокого уровня;
    - формируется сигнал nIRQ, поступающий в процессор;
    - автоматически очищается признак срабатывания прерывания по чувствительности к перепаду;

- сохраняется содержимое регистров (SPSR_irq, LR) в текущем уровне стека;
- возвращается значение вектора прерывания из регистра AIC_SVRn.

3) Предыдущий шаг предполагает переход к обработчику прерывания, чему должно предшествовать сохранение регистров R14_irq и SPSR_irq. Регистр связи LR необходимо уменьшить на 4, если предполагается его непосредственная загрузка в PC после окончания работы обработчика. Это также можно сделать командой: SUBS PC, LR, #4.
4) Для работы с прерываниями более высокого приоритета сбрасывают в 0 бит I регистра CPSR.
5) Выполняется непосредственно обработчик прерывания. По окончании обработки прерывания необходимо произвести запись в регистр AIC_EOICR (End Of Interrupt Command Register), чтобы сообщить контроллеру AIC о том, что обработка текущего прерывания закончен.
6) Устанавливается бит I регистра CPSR для маскирования прерывания перед выходом из обработчика, после чего можно восстановить значения регистров из стека, регистров PC и CPSR для передачи управления основной программе.

FIQ прерывание

Единственным источником FIQ прерывания является источник 0, если не используется режим Fast Forcing. На плате SAM7-P256 источником FIQ прерывания является кнопка SW1, которая может работать напрямую или через контроллер PIO.

Управление FIQ прерыванием. Прерывание FIQ не обрабатывается контроллером приоритетов. Режим обработки прерывания задается с помощью регистра AIC_SMR0, при этом поле PRIOR данного регистра не используется. Поле SRCTYPE регистра AIC_SMR0 позволяет запрограммировать обработку по фронту, спаду, по высокому или по низкому уровню сигнала.

Запись значения 0x1 в регистры AIC_IECR и AIC_IDCR соответственно разрешает или запрещает FIQ прерывание. Бит 0 регистра маски AIC_IMR определяет, разрешено или запрещено FIQ прерывание.

Вектор FIQ прерывания должен быть записан в регистр AIC_SVR0. Значение, записанное в него, возвращается при чтении процессором регистра AIC_FVR по адресу 0xFFFF F104. Таким образом, как и в случае с IRQ прерываниями, имеется возможность перехода к обработчику прерывания с помощью всего лишь одной инструкции: LDR PC, [PC, # – 0xF20].

При этом процессор загружает значение регистра AIC_FVR в программный счетчик, т.е. передает управление обработчику прерывания. Также, если FIQ прерывание настроено на перепад, происходит автоматический сброс сигнала прерывания.

Обработка FIQ прерывания. Перед началом обработки FIQ прерывания следует выполнить ряд действий:

- настроить контроллер AIC;
- загрузить адрес обработчика прерывания в регистр AIC_SVR0;
- разрешить FIQ прерывание;
- задать инструкцию перехода к обработчику: LDR PC, [PC, # – 0xF20].

При поступлении сигнала на вход nFIQ и если бит F регистра CPSR равен 0, выполняются следующие действия.

1) Процессор ARM входит в режим FIQ прерываний, регистр CPSR сохраняется в регистре SPSR_fiq, текущее значение программного счетчика PC сохраняется в регистре связи R14_fiq, в PC загружается значение 0x1C.
2) При выполнении соответствующей инструкции по адресу 0x1C происходит загрузка в PC содержимого регистра AIC_FVR.
3) Затем происходит непосредственное выполнение программы обработчика прерывания. Регистры R8−R13 сохранять не обязательно, так как в режиме FIQ имеются свои регистры R8_ fiq −R13_fiq, остальные регистры R0−R7 должны быть сохранены.
4) После окончания работы обработчика необходимо восстановить регистры PC и CPSR. Содержимое регистра связи LR перед загрузкой его в PC необходимо уменьшить на 4. Для выполнения указанных действий используется команда: SUBS PC, LR, #4.

Режим Fast Forcing

Режим Fast Forcing – это специальный режим работы контроллера AIC, при котором все IRQ прерывания перенаправляются на генерацию FIQ-прерывания.

Данный режим можно разрешить или запретить с помощью регистров AIC_FFER и AIC_FFDR (Fast Forcing Enable/Disable Register). Запись данных в эти регистры приводит к соответствующим изменениям в регистре AIC_FFSR (Fast Forcing Status Register), который непосредственно управляет режимом Fast Forcing каждого источника прерывания.

Если данный режим отключен, то обработка прерываний происходит так, как это было описано выше. Если режим Fast Forcing включен, то источник прерывания не может вызвать срабатывания обычного прерывания. Вместо этого формируется прерывание FIQ в обход

контроллера приоритетов. При этом происходит передача сигнала по линии nFIQ в процессор. Режим Fast Forcing не предполагает установку бита прерывания от источника 0 в регистре AIC_IPR.

При срабатывании какого-либо прерывания независимо от источника через регистр AIC_FVR читается содержимое регистра AIC_SVR0. Чтение регистра AIC_FVR не приводит к автоматическому сбросу прерывания. Необходимо вручную сбросить прерывание путем записи в регистр AIC_ICCR. При использовании режима Fast Forcing источник 0 остается полноценным источником FIQ-прерываний и работает, как описано ранее.

2.2. Программирование контроллера прерываний

Обработка IRQ прерываний

Для знакомства с работой IRQ прерываний подготовим программу, в которой IRQ прерывание генерируется от нажатия кнопки. При этом каждый раз при нажатии кнопки либо выключается, либо включается один из индикаторов. В программе, которая будет работать с IRQ прерываниями, необходимо предусмотреть:

- предварительную настройку контроллера AIC, кнопок и индикаторов;
- корректный вход и выход из обработчика прерывания.

Предварительная настройка контроллера AIC заключается в указании вектора прерывания, режима обработки прерывания и разрешении прерывания. Для обеспечения корректного входа и выхода из обработчика прерывания, а также для разрешения и запрещения вложенных прерываний используем библиотечные функции ISR_ENTRY(), ISR_EXIT(), ISR_ENABLE_NEST(), ISR_DISABLE_NEST() из подключаемой библиотеки «interrupt_utils.h». Пример программы работы с IRQ прерываниями представлен в листинге 3. Загрузите программу и проверьте ее работу на плате.

Листинг 3. *Программа обработки IRQ прерываний*

```
//=========================================================================//
//          Обработка IRQ прерывания
//=========================================================================//
#include "AT91SAM7S256.h"     // библиотека определений для AT91SAM7S256
#include "SAM7-P256.h"        // библиотека определений для периферии SAM7S-P256
#include "interrupt_utils.h"  // библиотека для работы с прерываниями

#define ERAM (1)              // память RAM

#ifdef ERAM                   // директивы компилятору, если вызываемая процедура
#define ATTR RAMFUNC          // обработки прерывания находится в памяти RAM
#else
#define ATTR
#endif
```

```
#if 0
#define IENABLE                 // Макрос входа во вложенное прерывание
  __asm { MRS     LR, SPSR      }
  __asm { STMFD   SP!, {LR}     }
  __asm { MSR     CPSR_c, #0x1F }
  __asm { STMFD   SP!, {LR}     }
#define IDISABLE                // Макрос выхода из вложенного прерывания
  __asm { LDMFD   SP!, {LR}     }
  __asm { MSR     CPSR_c, #0x92 }
  __asm { LDMFD   SP!, {LR}     }
  __asm { MSR     SPSR_cxsf, LR }
#endif

//процедура обработки IRQ прерывания----------------------------------------------//
void NACKEDFUNC ATTR irq_int (void)
{
      AT91PS_PIO pPIO = AT91C_BASE_PIOA;  // указатель на контроллер PIO
      AT91PS_AIC pAIC = AT91C_BASE_AIC;   // указатель на контроллер AIC

      ISR_ENTRY();         // действия при входе в обработчик прерывания
      ISR_ENABLE_NEST();   // разрешение прерываний с более высоким приоритетом
      if((pPIO->PIO_ISR & SW2) == SW2)    // проверка источника IRQ прерывания
      {
      if ((pPIO->PIO_PDSR & SW2) == SW2)  // выполнение при отпускании кнопки SW2
      {
            if ((pPIO->PIO_PDSR & LED1) == 0)   // если индикатор LED1 горел,
                  pPIO->PIO_SODR = LED1;        //  то его надо выключить,
            else                                // иначе,
            if ((pPIO->PIO_PDSR & LED1) == LED1)// если LED1 не горел,
                  pPIO->PIO_CODR = LED1;        //  то его надо включить
      }
      }
      ISR_DISABLE_NEST();     // запрещение прерываний с более высоким приоритетом
      pAIC->AIC_EOICR = 0;    // сигнализация об окончании обработки прерывания
      ISR_EXIT();             // возврат в основную процедуру
}

//предварительная настройка кнопок и контроллера прерываний AIC -------------------//
static void button_ AIC_config(void)
{
      volatile AT91PS_PIO pPIO = AT91C_BASE_PIOA; // указатель на контроллер PIO
      volatile AT91PS_AIC pAIC = AT91C_BASE_AIC;  // указатель на контроллер AIC
      volatile AT91PS_PMC pPMC = AT91C_BASE_PMC;  // указатель на контроллер PMC

      pPMC->PMC_PCER = (1<<AT91C_ID_PIOA);
      pPIO->PIO_ODR = SW2_MASK;             // вывод PIN20 работает на ввод (SW2)
      pPIO->PIO_PER = SW2_MASK;             // подключение кнопки SW2
      pPIO->PIO_IFER = SW2_MASK;            // подключение схемы подавления дребезга

      pAIC->AIC_IDCR = (1<<AT91C_ID_PIOA); // запрещение прерываний от PIO
      pAIC->AIC_SVR[AT91C_ID_PIOA] = (unsigned long) irq_int;// вектор IRQ прерывания
      // режим обработки прерывания
      pAIC->AIC_SMR[AT91C_ID_PIOA]= AT91C_AIC_SRCTYPE_EXT_NEGATIVE_EDGE|
                                    AT91C_AIC_PRIOR_LOWEST;
      pAIC->AIC_ICCR = (1<<AT91C_ID_PIOA);  // сброс бита прерывания от PIO
        pPIO->PIO_IER = SW2_MASK;             // разрешение прерывания от кнопки SW2
      pAIC->AIC_IECR = (1<<AT91C_ID_PIOA);  // разрешение прерываний от PIO
}
```

```
//предварительная настройка индикаторов-----------------------------------------//
static void led_config(void)
{
      volatile AT91PS_PIO pPIO = AT91C_BASE_PIOA; // указатель на контроллер PIO

      pPIO->PIO_OER = LED_MASK;     // выводы 17, 18 как выходы
      pPIO->PIO_PER = LED_MASK;     // подключение индикаторов
      pPIO->PIO_PPUDR = LED_MASK;   // отключение резисторов для индикаторов
      pPIO->PIO_SODR = LED1;        // выключение индикатора LED1
      pPIO->PIO_CODR = LED2;        // включение индикатора LED2
}

//главная процедура-------------------------------------------------------------//
int main()
{
      button_ AIC_config();         // настройка PIO и AIC
      led_config();                 // настройка индикаторов
      while (1)                     // бесконечный цикл ожидания прерывания
      {
      ;
      }
}
```

В данной программе есть ряд особенностей. Во-первых, в обработчике прерываний используется регистр PIO_ISR для проверки состояния кнопки и регистр PIO_PDSR для определения отпускания кнопки. Это необходимо делать, несмотря на то, что срабатывание прерывания настроено на отрицательный перепад сигнала.

Во-вторых, перед использованием контроллера AIC необходимо во время низкоуровневой инициализации микроконтроллера восемь раз произвести запись в регистр AIC_EOICR (такие действия рекомендуются производителем). Для этого в файл инициализации `init.c` добавлена строка: `for (i = 0; i < 8 ; i++){ pAIC->AIC_EOICR = 0;}`

Обработка FIQ прерываний

Для знакомства с FIQ прерываниями можно реализовать аналогичную программу обработки прерывания от нажатия кнопки. Однако есть некоторые особенности:

- при обработке FIQ прерываний нет необходимости разрешать и запрещать прерывания с более высоким приоритетом;
- нет необходимости проверять, была ли нажата нужная кнопка или нет, так как единственным источником FIQ прерываний является кнопка SW1 (если не используется режим Fast Forcing);
- также нет необходимости в конце обработчика прерывания производить запись в регистр AIC_EOICR.

Тексты замещающих функций для обработки прерываний в режиме *FIQ* представлены в листинге 4. Загрузите программу с замещающими функциями на плату и проверьте работу программы.

Листинг 4. *Функции NACKEDFUNC ATTR fiq_int и button_ AIC_config в режиме FIQ*

```
//обработчик FIQ прерывания-------------------------------------------------------//
void NACKEDFUNC ATTR fiq_int (void)
{
      AT91PS_PIO pPIO = AT91C_BASE_PIOA;        // указатель на контроллер PIO

      ISR_ENTRY();    // вспомогательные действия при входе в обработчик прерывания
      if ((pPIO->PIO_PDSR & LED1) == 0)         // если индикатор LED1 горит,
            pPIO->PIO_SODR = LED1;              //  то его надо выключить,
      else                                      // иначе,
      if ((pPIO->PIO_PDSR & LED1) == LED1)      // если индикатор LED1 не горит,
            pPIO->PIO_CODR = LED1;              //  то его надо включить
      ISR_EXIT();     // возврат в основную процедуру
}

//предварительная настройка кнопок и контроллера прерываний AIC -------------------//
static void button_ AIC_config(void)
{
      volatile AT91PS_PIO pPIO = AT91C_BASE_PIOA;// указатель на контроллер PIO
      volatile AT91PS_AIC pAIC = AT91C_BASE_AIC; // указатель на контроллер AIC
      volatile AT91PS_PMC pPMC = AT91C_BASE_PMC; // указатель на контроллер PMC

      pPMC->PMC_PCER = (1<<AT91C_ID_PIOA);
      pPIO->PIO_ODR = SW1_MASK;                 // вывод PIN19 работает на ввод (SW1)
      pPIO->PIO_PER = SW1_MASK;                 // подключение кнопки SW1

      pAIC->AIC_IDCR = (1<<AT91C_ID_FIQ);       // запрещение FIQ прерывания

      pAIC->AIC_SVR[AT91C_ID_FIQ] = (unsigned long) fiq_int;// вектор FIQ прерывания
      // режим обработки FIQ
      pAIC->AIC_SMR[AT91C_ID_FIQ] = AT91C_AIC_SRCTYPE_EXT_NEGATIVE_EDGE | 0;
      pAIC->AIC_ICCR = (1<<AT91C_ID_FIQ);       // установка в 0 бита FIQ прерывания
      pAIC->AIC_IECR = (1<<AT91C_ID_FIQ);       // разрешение FIQ прерывания
}
```

Прерывание в режима Fast Forcing

При использовании режима Fast Forcing структура обработчика прерывания остается такой же, как и при работе с FIQ прерываниями. Также при настройке контроллера AIC необходимо разрешить режим Fast Forcing для контроллера PIO, так как через него будет мультиплексироваться сигнал от кнопки SW2. Тексты замещающих функций для обработки прерываний в режиме Fast Forcing представлены в листинге 5.

Загрузите программу с замещающими функциями на плату и проверьте работу программы.

Листинг 5. *Функции NACKEDFUNC ATTR fiq_int и button_ AIC_config в режиме Fast Forcing*

```
//обработчик прерывания Fast Forcing----------------------------------------------//
void NACKEDFUNC ATTR fiq_int (void)
```

```
{
      AT91PS_PIO pPIO = AT91C_BASE_PIOA;        // указатель на контроллер PIO
      AT91PS_AIC pAIC = AT91C_BASE_AIC;         // указатель на контроллер AIC

      ISR_ENTRY();      //действия при входе в обработчик прерывания
      if((pPIO->PIO_ISR & SW2) == SW2)          // если была нажата кнопка SW2,то
      {
            if ((pPIO->PIO_PDSR & LED1) == 0)     // если индикатор LED1 горит,
                  pPIO->PIO_SODR = LED1;          // его надо выключить,
            else                                  // иначе,
            if ((pPIO->PIO_PDSR & LED1) == LED1) // если индикатор LED1 погашен,
                  pPIO->PIO_CODR = LED1;          // его надо включить
      }
      ISR_EXIT();       // возврат в основную процедуру
}

//предварительная настройка кнопок и контроллера прерываний AIC -------------------//
static void button_ AIC_config(void)
{
      volatile AT91PS_PIO pPIO = AT91C_BASE_PIOA; // указатель на контроллер PIO
      volatile AT91PS_AIC pAIC = AT91C_BASE_AIC;  // указатель на контроллер AIC
      volatile AT91PS_PMC pPMC = AT91C_BASE_PMC;  // указатель на контроллер PMC

      pPMC->PMC_PCER = (1<<AT91C_ID_PIOA);
      pPIO->PIO_ODR = SW2_MASK;                 // вывод PIN20 работает на ввод (SW2)
      pPIO->PIO_PER = SW2_MASK;                 // подключение кнопки SW2

      pAIC->AIC_IDCR = (1<<AT91C_ID_PIOA);      // запрещение прерываний от PIO
      pAIC->AIC_SVR[AT91C_ID_FIQ] = (unsigned long) fiq_int;// вектор прерывания
      // режим обработки прерывания
      pAIC->AIC_SMR[AT91C_ID_PIOA] = AT91C_AIC_SRCTYPE_EXT_NEGATIVE_EDGE | 0;
      pAIC->AIC_FFER = (1<<AT91C_ID_PIOA); // разрешение режима Fast Forcing
      pAIC->AIC_ICCR = (1<<AT91C_ID_PIOA); // установка в 0 бита прерывания от PIO
      pPIO->PIO_IER = SW2_MASK;             // разрешение прерывания от кнопки SW2
      pAIC->AIC_IECR = (1<<AT91C_ID_PIOA); // разрешение прерываний от PIO
}
```

3. Универсальный синхронный/асинхронный приемопередатчик

Универсальный синхронный/асинхронный приемопередатчик (USART) реализует один полнодуплексный последовательный канал связи. Формат передаваемых данных (длину поля данных, паритет, число стоповых битов и др.) можно настраивать для поддержки разнообразных стандартов приема/передачи: RS-232, RS-485, IrDa, ISO7816. Приемник позволяет обнаружить ошибки четности, формата кадров и переполнения, а также обрабатывать кадры данных переменной длины и осуществлять связь с медленными устройствами.

3.1. Функциональное описание USART

Модуль USART (рис.11) поддерживает работу с контроллером прямого доступа к памяти, что позволяет организовать управление процессом передачи или приема без участия процессора.

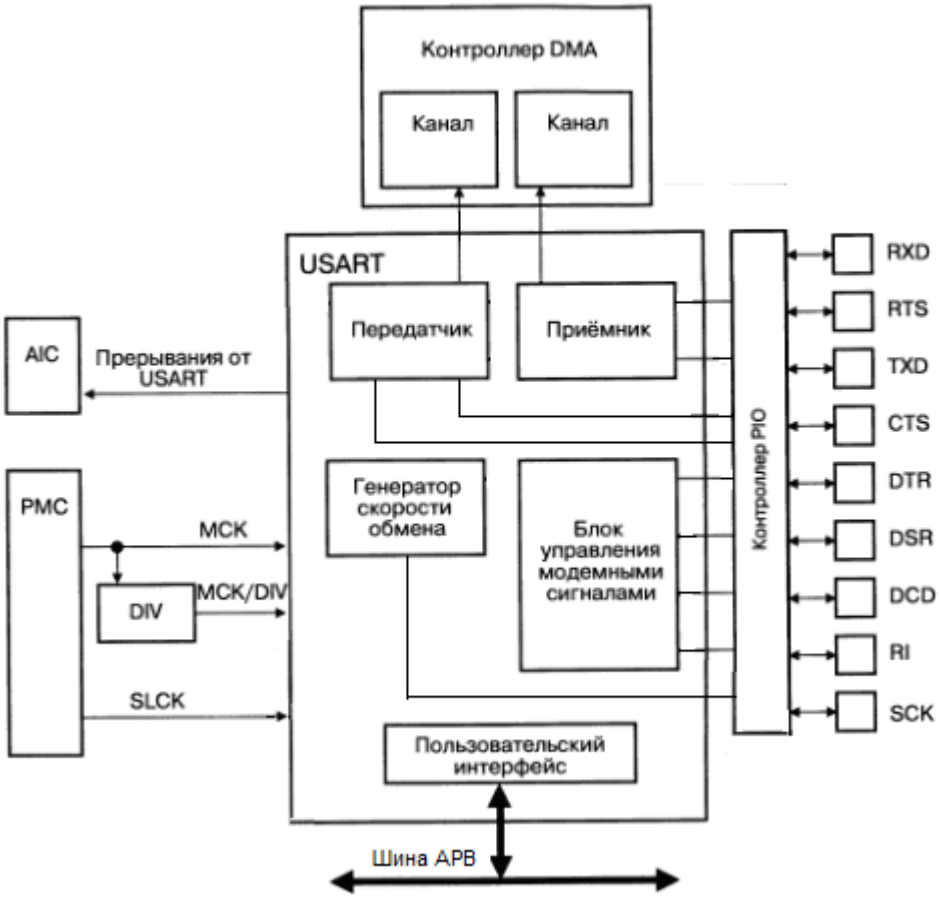


Рис. 11. Блок-схема USART

При работе с USART необходимо, чтобы все используемые им выводы были настроены на выполнение соответствующих функций. Для этого требуется настроить контроллер PIO. Также необходимо разрешить подачу синхроимпульсов на USART путем настройки контроллера управления питанием PMC. При использовании прерываний от USART необходимо, чтобы контроллер прерываний AIC был предварительно сконфигурирован. При этом не рекомендуется использовать линию прерывания из USART в режиме реакции на перепад.

Модуль USART поддерживает несколько последовательных синхронных или асинхронных способов передачи данных:

- (5–9)-разрядный полнодуплексный асинхронный и синхронный режимы передачи:
  - начиная с младших или старших разрядов;
  - используя 1, 1,5 или 2 стоповых бита в асинхронном режиме и 1 или 2 бита в синхронном режиме;
  - контроль по четности, нечетности, по маркеру, по пробелу, без проверки;
  - 8- или 16-кратная частота выборки при приеме;
  - возможность аппаратного подтверждения установления связи;

- управление сигналами модема;
- RS-485 с сигналом управления драйвером;
- ISO7816, T0 или T1 протоколы для взаимодействия со смарт-картами;
- передача посредством ИК-излучения (IrDa);
- тестовые режимы «автоматическое эхо», «местный шлейф», «удаленный шлейф».

Контроллер скорости передачи (Baud Rate Generator) выдает периодические тактовые сигналы на приемник и передатчик USART. Источник тактовых импульсов можно выбрать с помощью поля USCLKS регистра US_MR (Mode Register):

если USCLKS = 00, выбирается сигнал MCK;

если USCLKS = 01, MCK подается на делитель, который обычно имеет коэффициент деления, равный 8 (это зависит от модели микроконтроллера). На выходе делителя вырабатывается сигнал MCK/8;

если USCLKS = 11, используются внешние тактовые импульсы, поступающие на вход SCK. Если в качестве источника тактовых импульсов выбран внешний источник, его частота должна быть меньше частоты MCK по крайней мере в 4,5 раза.

Контроллер скорости передачи содержит 16-разрядный счетчик-делитель, который можно программировать с помощью поля CD регистра US_BRGR. Если CD=0, тактовые импульсы не генерируются. При CD=1 делитель не используется. Если USART настроен на асинхронный режим работы, то тактовый сигнал сначала делится на значение, записанное в поле CD регистра US_BRGR.

Затем полученный тактовый сигнал подается на приемник и передатчик, а также на делитель на 8 или 16 в зависимости от значения бита OVER в регистре US_MR. На выходе делителя получаем сигнал Baud Rate Clock.

Скорость передачи (Baud rate) определяется по формуле

$$\text{Baudrate} = \text{SelectedClock}/(8(2 - \text{OVER})/\text{CD},$$

где SelectedClock – частота тактового сигнала, выбранная с помощью поля USCKLS.

После сброса микроконтроллера приемник и передатчик отключены. Чтобы их подключить, нужно установить биты RXEN и TXEN регистра US_CR (Control Register). Эти действия можно выполнять одновременно или отдельно для приемника и передатчика.

Путем установки битов RSTRX и RSTTX регистра US_CR можно осуществить программный сброс приемника и передатчика USART. При программном сбросе так же, как и при аппаратном, связь немедленно прекращается. Помимо этого имеется возможность независимого отключения

приемника и передатчика, при котором передача (или прием) данных прерывается только после того, как будет обработан текущий байт и байт, находящийся в регистре US_THR (Transmit Holding Register). Для отключения передатчика следует установить бит TXDIS (для приемника – RXDIS) в регистре US_CR.

Передача данных

Передатчик работает одинаково в синхронном и асинхронном режиме. При каждом спаде тактового сигнала происходит последовательная передача на линию TXD стартового бита, до девяти битов данных, необязательный бит паритета и до двух стоповых битов.

Количество битов данных указывается в полях CHRL и MODE9 регистра US_MR. Если MODE9=1, то для данных отводится девять битов.

Длина поля данных зависит от значения CHRL: CHRL = 00 – 5 битов, CHRL = 01 – 6 битов, CHRL = 10 – 7 битов, CHRL = 11 – 8 битов.

Тип бита паритета указывается в поле PAR того же регистра.

Порядок передачи битов назначается полем MSBF регистра US_MR. Если MSBF=1, то передача начинается со старшего разряда, если MSBF=0 − с младшего.

Количество стоповых битов задается полем NBSTOP (табл.1).

Таблица 1. Количество стоповых битов в асинхронном и синхронном режимах.

NBSTOP	Асинхронный режим	Синхронный режим
00	1 стоповый бит	1 стоповый бит
01	1,5 стоповых битов	Зарезервировано
10	2 стоповых бита	2 стоповых бита
11	Зарезервировано	Зарезервировано

Для передачи символа его необходимо записать в регистр US_THR (Transmit Holding Register). Передатчик имеет два бита статуса в регистре US_CSR (Channel Status Register): бит TXRDY (Transmitter Ready), который показывает, что US_THR пуст, и бит TXEMPTY, который показывает, что все символы, записанные в US_THR, были обработаны. Когда завершается передача очередного символа, содержимое US_THR переписывается в сдвиговый регистр передатчика, регистр US_THR опустошается и устанавливается бит TXRDY.

Запись символа в регистр US_THR в то время, когда бит TXRDY=0, приводит к потере находящегося в US_THR символа.

Прием данных

Если USART настроен на асинхронный режим работы (SYNC=0), приемник дискретизирует линию RXD. Увеличение частоты тактового сигнала (дискретизация) в 8 или 16 раз по сравнению

со скоростью передачи задается битом OVER регистра US_MR. Если изменение скорости осуществляется в 16 раз (OVER=0), то стартовый бит обнаруживается, начиная с 8-го такта. Затем через каждые 16 тактов происходит прием битов данных, бита паритета и стоповых битов. При изменении скорости в 8 раз (OVER=1) стартовый бит может быть принят через 4 такта сэмплированного сигнала, а остальные биты принимаются через каждые 8 тактов.

В синхронном режиме (SYNC=1) приемник обрабатывает сигнал RXD по каждому фронту тактового сигнала Baud Rate Clock. Если обнаружен низкий уровень на линии при ожидании приема данных, то это свидетельствует о приходе стартового бита. Затем обрабатываются биты данных, бит паритета и стоповые биты, после чего наступает режим ожидания очередного стартового бита. Синхронный режим работы обеспечивает высокоскоростную передачу информации. Конфигурационные биты и поля аналогичны асинхронному режиму.

Когда прием символа завершается, он передается в регистр US_RHR (Receive Holding Register). При этом устанавливается бит RXRDY в регистре US_CSR (Channel Status Register). Если прием символа завершается при ранее установленном бите RXRDY, то устанавливается в 1 бит переполнения OVRE. При этом последний принятый символ помещается в регистр US_RHR, замещая хранящуюся там информацию. Бит OVRE сбрасывается при установке в 1 бита RSTSTA в регистре US_CR.

3.2. Программирование USART

Знакомство с основными функциями USART можно провести на примере аппаратной аутентификации. Суть ее заключается в следующем. В диалоговом режиме у пользователя запрашивается пароль. После ввода пароля по нажатию клавиши ENTER осуществляется процесс аутентификации пользователя. Если пароль был введен правильно, на плате SAM7-P256 загорается зеленый индикатор, если неправильно – желтый. При этом в терминал выводится сообщение об успешной или неудачной попытке аутентификации.

Программа использует библиотеку, в которой реализованы элементарные функции по приему/передаче символов и строк.

Авторизация пользователя с помощью разработанных функций возлагается на главный модуль программы. Для считывания введенного пароля необходимо организовать цикл, в котором происходит посимвольный прием данных, вводимых пользователем. В конце принятой строки добавляется признак конца строки `'\D'`. В листинге 6 представлен текст главного модуля программы аутентификации.

Листинг 6. *Главный модуль программы аутентификации (main.c)*

```
//=========================================================================//
```

```
//       Обмен информацией по USART0
//=============================================================================//

#include "AT91SAM7S256.h"        // библиотека определений для AT91SAM7S256
#include "SAM7-P256.h"           // библиотека определений для периферии SAM7-P256
#include "usart.h"               // библиотека функций для работы с USART

//настройка индикаторов-----------------------------------------------------//
void led_config(void)
{
   AT91PS_PIO pPIO = AT91C_BASE_PIOA;        // указатель на контроллер PIO

   pPIO->PIO_PER = LED_MASK;                 // подключение индикаторов к PIO
   pPIO->PIO_OER = LED_MASK;                 // настройка выводов на вывод
   pPIO->PIO_PPUDR = LED_MASK;               // отключение резисторов
   pPIO->PIO_SODR = LED_MASK;                // выключение индикаторов
}

//процедура формирования задержки-------------------------------------------//
void delay()
{
   int i;                                    // счетчик
   for (i = 0; i < 24000000; i++)            // цикл задержки
         asm("nop");
}

//главная функция-----------------------------------------------------------//
int main()
{
   AT91PS_PIO pPIO = AT91C_BASE_PIOA;        // указатель на контроллер PIO
   AT91PS_USART pUSART = AT91C_BASE_US0;     // указатель на USART0
   unsigned char     passw[]="";             // буфер для вводимого пароля
   unsigned char     psw_ok[5]="asdf";       // правильный пароль
   unsigned int      i=0x0;            // вспомогательный счетчик
   unsigned char     ch;               // вспомогательная переменная

   led_config();                       // настройка индикаторов
   InitUSART0()

   while (1)                           // бесконечный цикл
   {
         i=0x0;                        // обнуление счетчика
         *passw="";                    // очистка буфера
         write_str_USART0("\nEnter the password:\n"); //вывод приглашения на ввод
         ch=read_char_USART0();        // чтение первого символа вводимого пароля
         while (ch != 0xD)             // если не ENTER, то
         {
               passw[i]=ch;            // добавление очередного символа в буфер
               write_char_USART0(ch);  // вывод введенного символа
               i++;                    // переход к следующему символу
               ch=read_char_USART0();  // чтение следующего символа
         }
         passw[i]='\D';                // добавление признака конца строки
         ch=psw_ok[i];
         while ((passw[i] == ch) & (I != -1)) // проверка пароля
         {
               i--;
               ch=psw_ok[i];
         }
```

```
            pPIO->PIO_SODR = LED_MASK;    // выключение индикаторов
            if (i == -1)                  // при совпадении паролей
            {
                  pPIO->PIO_CODR = LED1;  // включить зеленый индикатор
                  write_str_USART0("\nThe password is correct
                                    (green LED must be turned on).\n");
            }
            else                          // при несовпадении паролей
            {
                  pPIO->PIO_CODR = LED2;  // включить желтый индикатор
                  write_str_USART0("\nError (yellow LED must be turned on).\n");
            }
            delay();                // задержка перед новой попыткой ввода пароля
      }
      return 0;
}
```

Для тестирования программы необходим терминал RS-232. В качестве него можно использовать встроенные средства Windows или специализированные программы.

Задание 3. Настройте программу для проверки пароля из 8-и символов. Проверьте работу программы.

Оформление отчета

Отчет должен содержать:

- схему контроллера PIO, тексты программ ввода-вывода согласно заданиям 1 и 2 и диаграммы выходных сигналов (включения/выключения светодиодов),
- тексты программ для работы с внешними прерываниями IRQ, FIQ,
- модифицированную программу для последовательного канала,
- наблюдаемые результаты при выполнении каждой программы.

Требования при защите: уметь отвечать на вопросы по текстам программ на языке Си, основным способам обработки внешних прерываний, работе контроллеров PIO и USART.

Контрольные вопросы

1. Охарактеризовать состав микроконтроллера SAM7 и входящие в него устройства системного контроллера и периферийные устройства.
2. Как работает контроллер ввода-вывода PIO?
3. Какие виды внешних прерываний обрабатывает контроллер прерываний?
4. Опишите работу канала USART.

Рекомендуемая литература

1. Хартов В.Я. Введение в архитектуру микроконтроллеров ARM7 семейства AT91SAM7. – М.: Издательство МГТУ им. Н.Э. Баумана, 2015 − 124 с. (Портал ebooks.bmstu.ru).