

ВОПРОСЫ ПО ДИСЦИПЛИНЕ МИКРОПРОЦЕССОРНЫЕ СИСТЕМЫ

1. Последовательные интерфейсы. Основные понятия: однопроводная и дифференциальная передача; топологии шин.
2. Интерфейс RS-232: физический уровень. Схемотехнические аспекты применения: преобразование логических уровней напряжения; гальваническое разделение.
3. Интерфейс RS-232: управление потоком данных и квитирование; примеры форматов кадров передаваемых данных.
4. Интерфейс I2C: общая характеристика; описание шины; формат первого байта, передаваемого ведущим устройством; примеры сеансов передачи данных.
5. Интерфейс SPI: общая характеристика, описание шины, форматы передачи данных.
6. Применение языка Си для кодирования встраиваемого ПО. Расширения языка C51: типы и модели памяти, декларации переменных и РСФ; подпрограммы обработки прерываний; встроенные функции.
7. Схемы алгоритмов и программ. Символы. Правила применения символов и выполнения схем.
8. Диаграммы состояний. Базовые понятия и условные графические обозначения диаграммы состояний. Составное состояние и подсостояние.
9. *Диаграммы состояний. Построение диаграмм состояний и переход от диаграмм состояний к кодированию программы.*
10. Место разработки ПО в разработке измерительного устройства в целом. Обобщенные схемы программ измерительных устройств с микроконтроллером. Структура программы. Нисходящее и восходящее проектирование программ. Метод расширения ядра.
11. Основные понятия о тестировании и отладке программ: определение тестирования, теста и отладки. Виды ошибок. Принципы построения тестов. Методы построения тестов, основанные на принципе «белого ящика».
12. Основные понятия о тестировании и отладке программ: определение тестирования, теста и отладки. Методы проверки правильности программ. Принципы построения тестов. Методы построения тестов, основанные на принципе «черного ящика».
13. Процессор Cortex-M3: режимы работы и уровни привилегий выполнения программы; рабочие состояния; основной стек и стек процесса; действия после сброса.
14. Процессор Cortex-M3: карта памяти; побитовый доступ; обращение к невыровненным данным.
15. Процессор Cortex-M3: состояния и типы исключений; обработка исключений; приоритеты исключений; контроллер вложенных векторных прерываний (NVIC); системный таймер.
16. Процессор Cortex-M3: режимы пониженного энергопотребления.
17. Стандарт CMSIS (стандарт программного интерфейса микроконтроллеров с ядром Cortex-M).

1. Последовательные интерфейсы. Основные понятия: однопроводная и дифференциальная передача; топологии шин.

Последовательный интерфейс — используется лишь одна сигнальная линия, и биты группы передаются друг за другом по очереди; на каждый из них отводится свой квант времени (битовый интервал). Примеры: последовательный коммуникационный порт (COM-порт), последовательные шины USB и FireWire, PCI Express, интерфейсы локальных и глобальных сетей.

По способу организации передачи данных различают однопроводные (single-ended) и дифференциальные (differential) интерфейсы.

Основным недостатком однопроводных систем является их низкая помехоустойчивость. Из-за наводок на общий провод возможен сдвиг уровней сигналов, приводящий к ошибкам. При передаче на расстояния порядка нескольких метров начинает оказывать влияние индуктивность и емкость проводов.

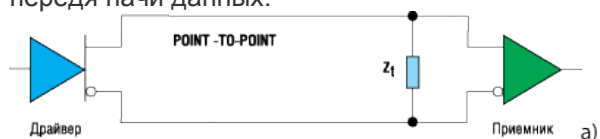
Преодолеть указанные недостатки удастся в дифференциальных системах

Для балансной дифференциальной передачи данных используется пара проводов. На приемном конце линии вычисляется разность между сигналами. Заметим, что такой способ передачи данных пригоден не только для цифровых, но и для аналоговых линий. Ясно, что при дифференциальной передаче удастся в значительной мере подавить синфазную помеху. Отсюда следует основное достоинство дифференциальных протоколов — высокая помехоустойчивость. Недаром один из самых распространенных протоколов в промышленных компьютерах — RS-485 построен по дифференциальной схеме.

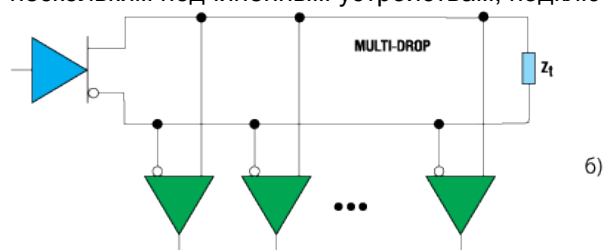
Недостатком дифференциальных схем является их относительно высокая стоимость, а также сложности при выполнении парных согласованных каскадов передатчиков и приемников.

Топология шин

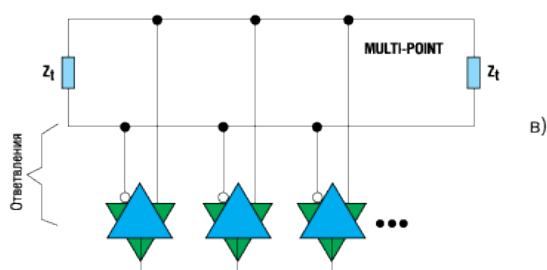
Точка-точка применяется для построения последовательных или параллельных быстродействующих шин передачи данных.



Многоточечная шина. Используется, когда одно ведущее устройство передает одну и ту же информацию нескольким подчиненным устройствам, подключенным к общей шине.



Многоточечная шина — моноканал



2. Интерфейс RS-232: физический уровень. Схемотехнические аспекты применения: преобразование логических уровней напряжения; гальваническое разделение.

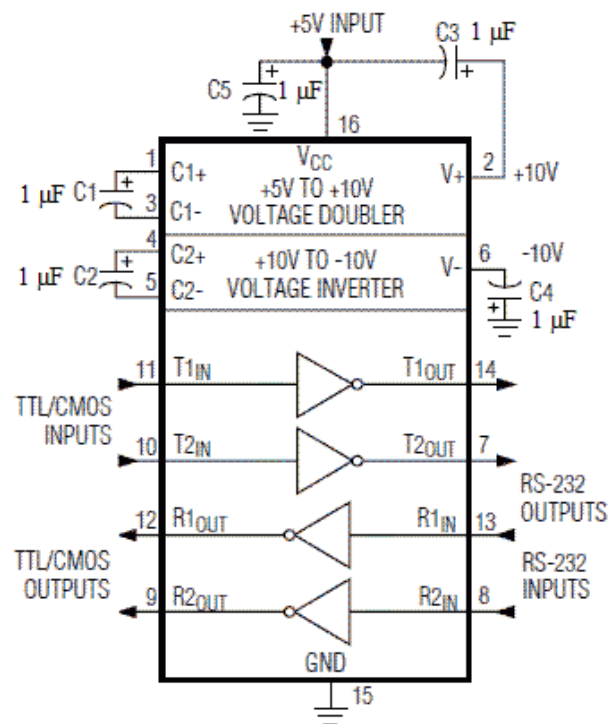
RS-232 — проводной дуплексный интерфейс. Метод передачи данных аналогичен асинхронному последовательному интерфейсу.

Для электрического согласования линий RS-232 и стандартной цифровой логики UART выпускается большая номенклатура микросхем драйверов, например MAX232.

Обычно RS-232 развязывают по токовой петле.

Плюсом токовой петли является то, что она легко развязывается оптикой, ведь светодиод питается током.

Когда подаем единицу на вход, то она зажигает светодиод, транзистор оптопары открывается и пускает ток в петлю. Это ток зажигает светодиод во второй оптопаре, ее транзистор открывается и прижимает линию к земле. Линия при этом получается инвертирующей.



3. Интерфейс RS-232: управление потоком данных и квитирование; примеры форматов кадров передаваемых данных.

Управление потоком данных (Flow Control):

- Аппаратный. Сигналы «готов/занят» передаются по отдельным физическим линиям связи. Основные линии: RTS (Request to send) - запрос на передачу; CTS (Ready for sending) - свободен для передачи.

- Программный. Сигнал «готов/занят» взводится и сбрасывается вставкой в поток данных специальной уникальной последовательности (XOn/XOff).

Пример кадра :

Modbus RTU :

Start	Address	Function	Data	CRC	End
3.5 Char time	8 Bit	8 Bit	N * 8Bit	16 Bit	3.5 Char time

Проверка целостности осуществляется с помощью CRC.

- Квитирование (Handshaking - подтверждение), передаётся порциями (кадрами). Программный флажок «готов/занят» взводится и сбрасывается специальными соглашениями. Пример: контроль потока в протоколе TCP методом скользящего окна.

4. Интерфейс I2C: общая характеристика; описание шины; формат первого байта, передаваемого ведущим устройством; примеры сеансов передачи данных.

I²C (Inter-Integrated Circuit) – последовательная шина данных для связи интегральных схем, использующая две двунаправленные линии связи (SDA и SCL). Используется для соединения низкоскоростных периферийных компонентов с материнской платой, встраиваемыми системами и мобильными телефонами.

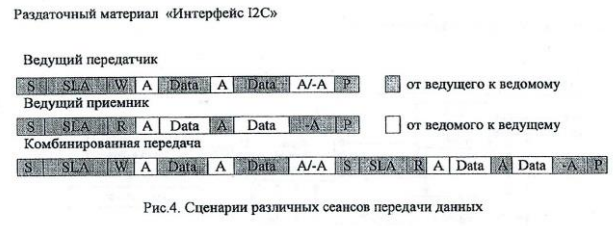
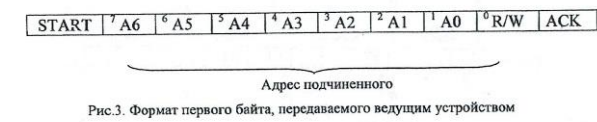
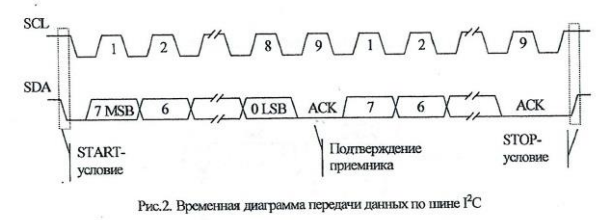
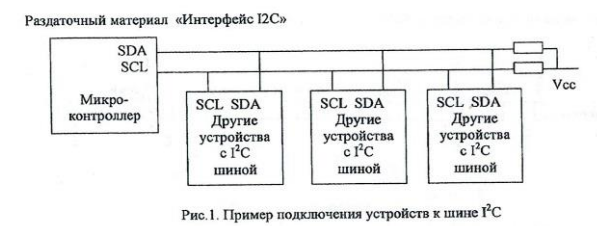


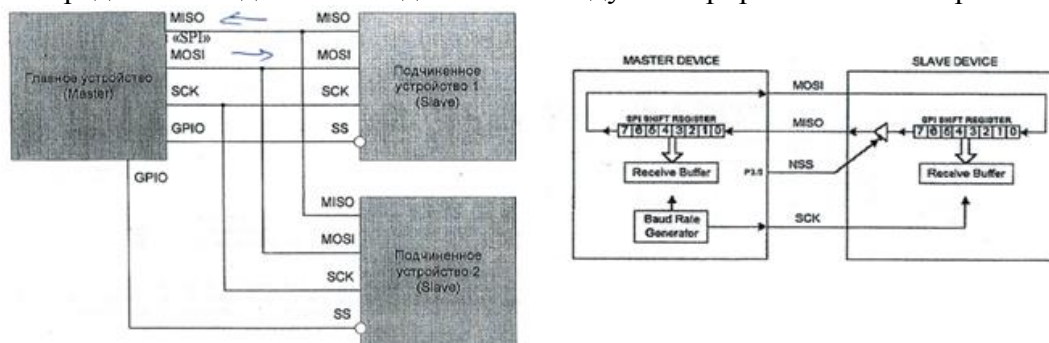
Таблица 1. Специальные адреса шины I²C и их назначение

Адрес подчиненного	Бит R/W	Назначение
0000 000	0	General call address – адрес общего вызова
0000 000	1	Start byte – признак начала активного обмена
0000 001	X	CBUS address – выбор устройства, удовлетворяющих передаче в формате шины CBUS
0000 010	X	Зарезервирован для устройств иных шин
0000 011	X	Зарезервирован
0000 1XX	X	Код главного устройства в высокоскоростном режиме
1111 1XX	X	Зарезервирован
1111 0XX	X	Признак 10-битовой адресации

Процедура обмена начинается с того, что ведущий формирует состояние СТАРТ: генерирует переход сигнала линии SDA из ВЫСОКОГО состояния в НИЗКОЕ при ВЫСОКОМ уровне на линии SCL. Этот переход воспринимается всеми устройствами, подключенными к шине, как признак начала процедуры обмена. Генерация синхросигнала — это всегда обязанность ведущего; каждый ведущий генерирует свой собственный сигнал синхронизации при пересылке данных по шине. Процедура обмена завершается тем, что ведущий формирует состояние СТОП — переход состояния линии SDA из низкого состояния в ВЫСОКОЕ при ВЫСОКОМ состоянии линии SCL. Состояния СТАРТ и СТОП всегда вырабатываются ведущим.

5. Интерфейс SPI: общая характеристика, описание шины, форматы передачи данных.

SPI интерфейс предназначен для обмена данными между интегрированными микросхемами.



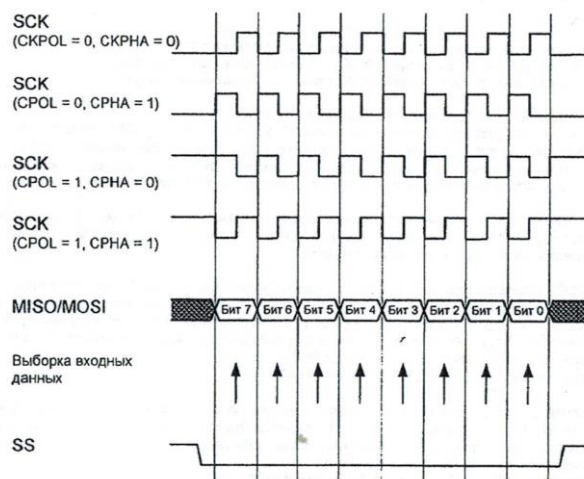
Назначение линий SPI-шины

Линия	Описание
MISO (Master In/Slave Out)	Вход данных главного устройства и выход данных подчиненного устройства.
MOSI (Master Out/Slave In)	Выход данных главного устройства и вход данных подчиненного устройства.
SCK (Serial Clock)	Синхросигнал передачи последовательных данных. В течение каждого периода принимается и передается 1 бит данных.
SS (Slave Select)	Выбор подчиненного устройства. Так как главное устройство – это обычно микроконтроллер, то для выбора «подчиненного» используются линии порта общего назначения (GPIO - General Purpose Input/Output).

Приём и передача идут одновременно (но сдвинуты на пол такта).

Описание параметров CPOL и CPHA.

Параметр	Описание
CPOL	Полярность SCK в неактивном состоянии: 0 – на линии SCK низкий уровень («0»); 1 – на линии SCK высокий уровень («1»).
CPHA	Фаза синхросигнала: 0 – входные данные выбираются по первому фронту (срезу) в пределах периода SCK; 1 – входные данные выбираются по второму фронту (срезу) в пределах периода SCK.



Временные диаграммы синхросигнала и данных в зависимости от CPOL и CPHA

6. Применение языка Си для кодирования встраиваемого ПО. Расширения языка C51: типы и модели памяти, декларации переменных и РСФ; подпрограммы обработки прерываний; встроенные функции.

Класс хранения: область действия и время жизни

Область видимости – часть программы, в которой переменную можно использовать.

Классы хранения:

- Автоматический (auto)
- Регистровый (register)
- Внутренний статический (static) – внутри функции
- Внешний (extern)
- Внешний статический (static) - вне функции

Класс хранения	Внешний	Внешний статический	Аргументы функции	Автоматический	Регистровый	Внутренний статический
Область действия	программа	модуль	функция	блок	блок	блок
Время жизни	программа	программа	функция	блок	блок	программа

Модели памяти – определяет какой тип памяти используется:

SMALL(малая) – внутренне ОЗУ

COMPACT(компактная) и LARGE(большая) – внешнее ОЗУ

Явно декларируемые типы памяти:

<тип данных> <тип памяти> <имя переменной> [=значение]

Тип памяти:

code – в ПЗУ

data – в ОЗУ

idata – косвенная адресация

bdata, pdata – битовая память

xdata – внешнее ОЗУ

Пример: uint8_t data x;

Новые типы данных:

Тип	Размер в битах	Размер в байтах	Диапазон значений
bit	1		0 и 1
sbit	1		0 и 1
sfr	8	1	0 – 255
sfr16	16	2	0 – 65535

sbit, sfr, sfr16 – регистры специальных функций

Примеры:

int bdata ibase;

sbit bit0 = ibase^0; //не присваивание, а декларация

sfr P0 = 0x80; //порт 0, адрес 80H

sfr16 T2 = 0xCC; // таймер 2: T2L=0CCh, T2H=0h

Объявление функций:

<возвращаемый тип> <имя функции> ([список формальных параметров]) [{small | comact | large}] [reentrant]

[interrupt n] [using k]

reentrant – допускается рекурсия

interrupt n – функция – обработчик прерывания, где n – номер прерывания (в даташите)

using k – указание на использование банка регистров k

Пример: void ISR_TC0(void) interrupt 1 using 2 {...}

Размещение переменных по абсолютному адресу:

<тип данных> [<тип памяти>] <имя переменной> _at_ <константа>

Пример: float x_at_ 50;

Встроенные функции:


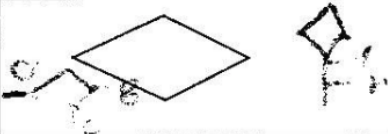
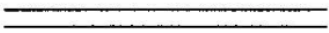

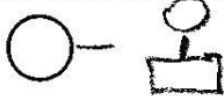
Циклические сдвиги			Команда NOP	Проверка бита и его сброс
crol (влево)	_irol_	_lrol_	_nop_	_testbit_
cror (вправо)	_iror_	_lror_		
char – 8 разряд- ный сдвиг	int – 16 разряд- ный	long – 32 разряд- ный		

Прототипы этих функции содержит заголовочный файл <intrins.h>

7. Схемы алгоритмов и программ. Символы. Правила применения символов и выполнения схем.

Схемы алгоритмов и программ - тип схем (графических моделей), описывающих алгоритмы или процессы, в которых отдельные шаги изображаются в виде блоков различной формы, соединенных между собой линиями, указывающими направление последовательности.

Символы:

Наименование	Обозначение	Функция
Процесс		Отображает обработку данных любого вида (выполнение операции или группы операций, в результате которых изменяется значение, форма представления или расположение данных).
Решение		Выбирает направления выполнения алгоритма или программы в зависимости от условий, определенных внутри этого символа.
Параллельные действия		Отображает синхронизацию двух или более параллельных операций.
Предопределенный процесс		Отображает предопределенный процесс, состоящий из одной или нескольких операций или шагов программы, которые определены в другом месте (в подпрограмме, модуле).
Терминатор		Начало, конец, прерывание процесса обработки данных или выполнения программы.
Линия потока		Отображает поток данных или управления.
Комментарий		Используется для добавления комментариев в целях объяснения или примечаний.
Соединитель		Используется для обрыва линии потока и продолжения ее в другом месте.

Правила применения символов и выполнения схем:

**** Правила применения символов**

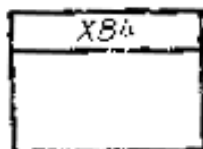
* Надписи внутри блока действия выполняются сверху вниз и слева направо.

* Если текст внутри символа, превышает его размеры, использовать символ комментария.

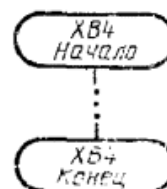
* Может использоваться подробное представление, обозначается символом с полосой для процесса или данных. Указывает, что в этом же комплекте документации в другом месте имеется более подробное представление.

В качестве первого и последнего символа подробного представления используется символ указателя конца. Первый символ указателя конца должен содержать ссылку, которая имеется также в символе с полосой.

Символ с полосой



Подробное представление



**** Правила выполнения соединений**

- * Направление потока (линии) слева направо и сверху вниз считается стандартным. Когда необходимо внести большую ясность в схему, на линиях используются стрелки.
- * В схемах следует избегать пересечения линий. Пересекающиеся линии не имеют логической связи между собой.



- * Две или более входящие линии могут объединяться в одну исходящую линию. Место объединения должно быть смещено.

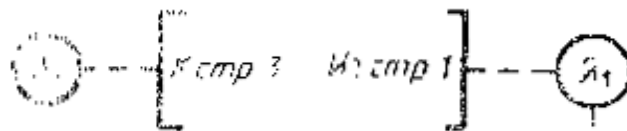


- * Линии должны подходить к символу слева, либо сверху, а исходить справа, либо снизу. * При необходимости линии в схемах разрывать. Соединитель в начале разрыва называется внешним соединителем, а соединитель в конце разрыва - внутренним соединителем.

- * Ссылки к страницам приводятся совместно с комментарием для их соединителей.

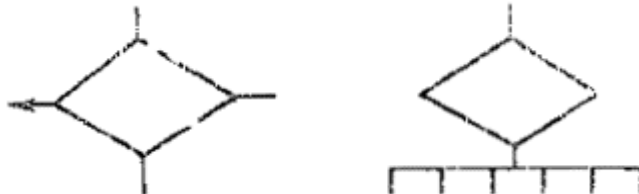
Внешний соединитель

Внутренний соединитель

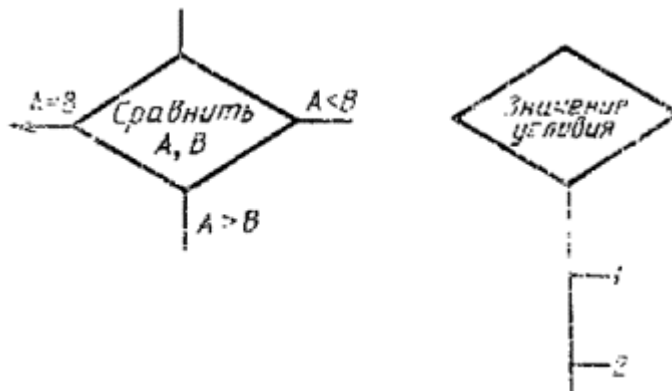


**** Специальные условные обозначения**

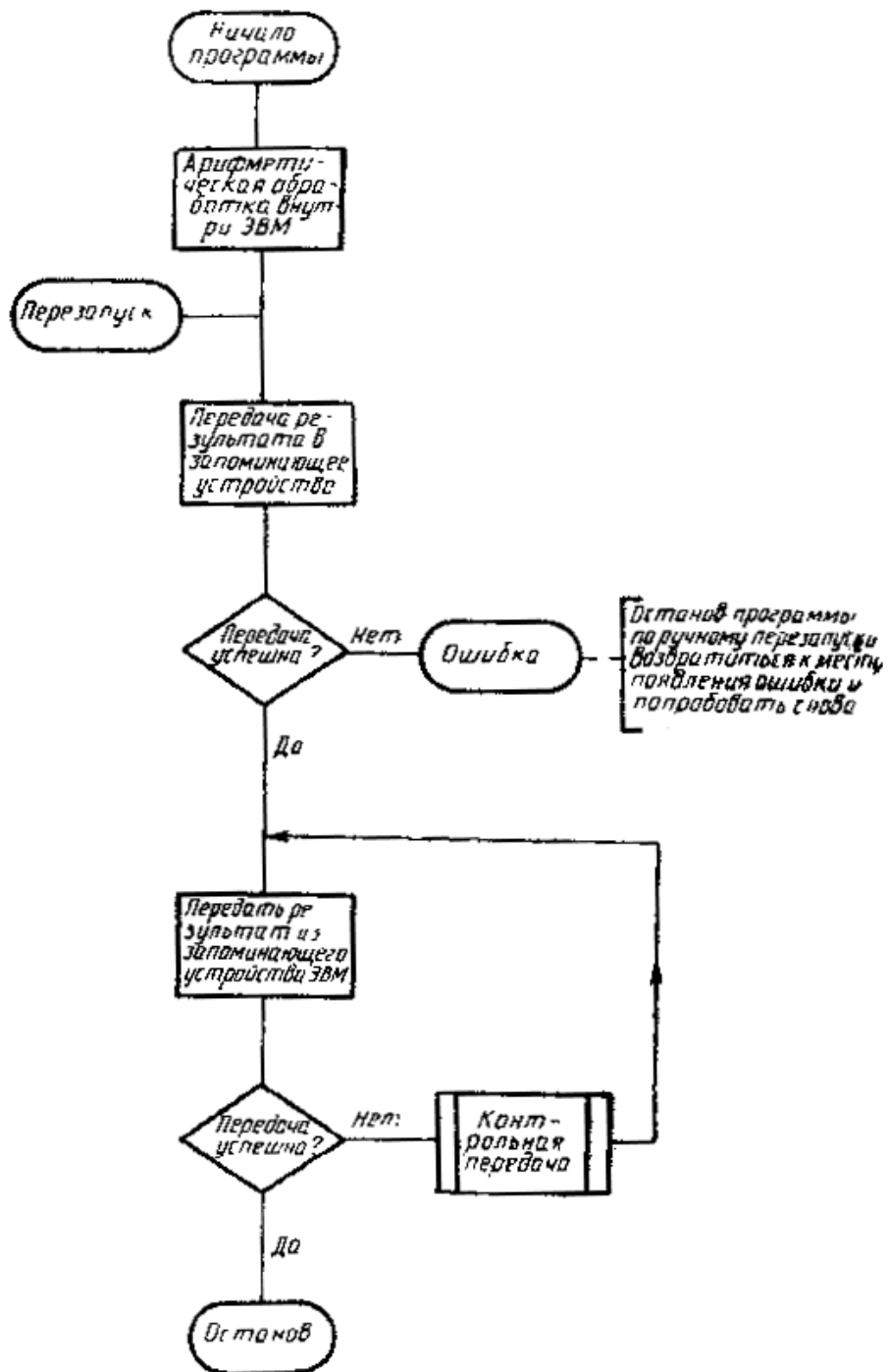
- * Несколько выходов из символа следует показывать:



- * Каждый выход должен сопровождаться условием.



Пример схемы алгоритма:



8. Диаграммы состояний. Базовые понятия и условные графические обозначения диаграммы состояний. Составное состояние и подсостояние.

Диаграмма состояний — ориентированный граф для конечного автомата.

Диаграмма состояний включает:

- 1) набор состояний;
- 2) событие, которое вызовет переход из 1 сост. в другое;
- 3) действия, которые происх. в результате изменения.

1 Состояние — период в жизни автомата.

2 Метка действия / выражение действия.

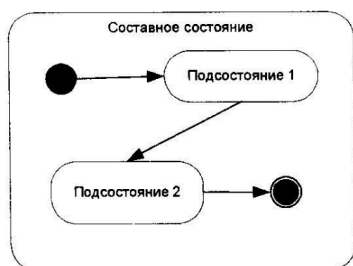
3 Переход — изменение сост. сис-мы.

Есть 2 типа перехода:

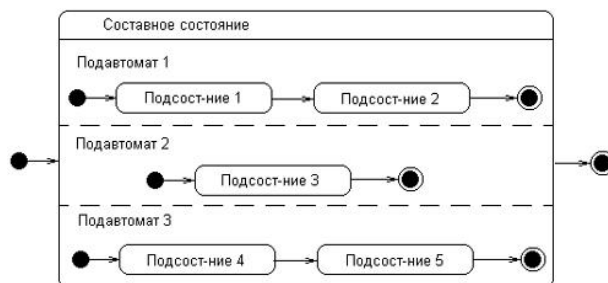
- * триггерный — условный;
- * не триггерный — безусловный;

Составное состояние — сложное состояние, состоящее из других вложенных в него состояний (подсостояний).

*Последовательные подсостояния



* Параллельные подсостояния:



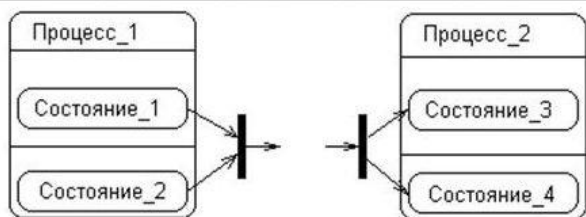
Исторические состояния — исп. для того из последовательных подсостояний, которое было текущим в момент выхода из составного состояния.

Н — недавнее истор. состояние;

Н* - давнее истор. сост.

Переход в историч. состояние снова активизирует подсостояние, которое было активно до перехода.

Переход между || подсостояниями:



Метка	Описание
entry	Указывает на то, что следующее за ней выражение действия должно быть выполнено <u>в момент входа в данное состояние (входное действие)</u> .
exit	Указывает на то, что следующее за ней выражение действия должно быть выполнено <u>в момент выхода в данное состояние (выходное действие)</u> .
do	Специфицирует некоторую деятельность (ду-деятельность), которая выполняется <u>в течение всего времени, пока объект находится в данном состоянии</u> , или до тех пор, пока не будет выполнено условие ее окончания, специфицируемое в вычислительной процедуре.
xxx	Во всех остальных случаях метка действия обозначает событие, которое запускает соответствующее выражение действия. Эти события называются <u>внутренними переходами</u> . Выход из состояния и повторный вход в него не происходят. Это означает, что действия входа и выхода не выполняются. Внутренние переходы удобны для моделирования действий прерывания, не требующих изменения состояния, например при подсчете количества каких-либо событий или выдаче на экран справочной информации[6].

10. Место разработки ПО в разработке измерительного устройства в целом. Обобщенные схемы программ измерительных устройств с микроконтроллером. Структура программы. Нисходящее и восходящее проектирование программ. Метод расширения ядра.

Firm ware – прог-аппар средства, встроенные прог, «защитные прог» (в ПЗУ)

Embedded software – встроен прог обеспеч, прог обесп, зашитое в ПЗУ.

Разработчики встроен ПО сод ряд специф этапов, обусловленных налич как аппарат, так и прог частей. Осн особен и проблема такого рода разработки – оптим комплексирование и интегрир как аппарат, так и прог компонент в едином устр.

Структура программы:

Еще одна классич графич форма предст программы – структура программы. Вкл опред всех модулей прог, их иерархии и сопряж между ними. Графически структура прог изобр в виде иерархич схемы. Чтобы созд эту сх следует начать с вершины и идти вниз. Необх, чтобы первый модуль в прог упр выполн ост модулей.

Схемы иерархии не показывают потока дан, порядка исполнения или моментов и частоты активиз кажд модуля. Располож модулей на зад ур-не не опред порядок их исполн. Схемы иерархии показ подчиненность модулей. Кажд модуль активиз вышестоящим и закончив свою работу, возвращ управл вызвавшему модулю. Такое вертик упр происх по след правилам:

- Модуль должен возвращ упр тому модулю, кот его вызвал
- М может вызв другой модуль уровнем ниже. Он не м. вызв мод своего ур или выше

Проектирование структуры программы

- Нисходящее проектирование

Прогр разрабат ,начин с сист уровня, путем послед замены общих положений конкр прог. Нисх проектир вып в след порядке:

1. Разраб и тестир общая управл прог, кот выз главн подпрограммы. Ненаписанная подпрог замен прог заглушками.
2. Вып детализация каждой прог заглушки. На кажд этапе, где заглушка замен работающей прог должны вып отладка и тестир
3. Тестируется вся с-ма в целом

- Восходящее проектирование

Осн на выделении неск достаточно крупных модулей, реализ некоторые ф в общей программе. Каждый модуль при восход проектировании автономно програм, тестир и отлаживается. После этого отд мод объедин в подс-мы с пом упр модуля, в кот опред послед-ть вызово мод, ввод-вывод и контроль дан и результ-в. В свою очередь подсист затем объедин в более сложн с-мы и в общий прог комплекс, кот подверг комплексной отладке с провер правильности межмодульных связей.

Метод расширенного ядра

Метод восход програм, при кот осн влияние удаляется выявлению множ-ва вспомог мод, а не опред ф всей прог в целом.

11. Основные понятия о тестировании и отладке программ: определение тестирования, теста и отладки. Виды ошибок. Принципы построения тестов. Метод построения тестов основанный на принципе «белого ящика».

Тестирование-процесс использующий программу с целью обнаружения ошибок.

Тест-набор входных данных использующихся при проверке.

Отладка-процесс локализации и исправления ошибок.

Ошибки:

1. Логические
 - 1.1 Алгоритма
 - 1.2 Семантические
2. Синтаксические

классификация методов проверки правильности программ



Проектирование теста

Методологии, основанные на принципе черного ящика:

- Эквивалентное разбиение
- Анализ граничных значений
- Метод функциональных диаграмм
- Предположение об ошибке

Принцип построения тестов

1. Тестирование по отношению к спецификации (черный ящик)
2. Тестирование по отношению к тексту программы (метод белого ящика)

Методологии, основанные на принципе белого ящика:

- ☐ Покрытие операторов
- ☐ Покрытие решений
- ☐ Покрытие условий
- ☐ Покрытие решений/условий
- ☐ Комбинаторное покрытие условий

12. Основные понятия о тестировании и отладке программ: определение тестирования, теста и отладки. Виды ошибок. Принципы построения тестов. Метод построения тестов основанный на принципе «черного ящика».

Тестирование-процесс использующий программу с целью обнаружения ошибок.

Тест-набор входных данных использующихся при проверке.

Отладка-процесс локализации и исправления ошибок.

Ошибки:

2. Логические

1.1Алгоритма

1.2Семантические

2.Синтаксические

Принцип построения тестов

1.Тестирование по отношению к спецификации (черный ящик)

2.Тестирование по отношению к тексту программы (метод белого ящика)

Методологии, основанные на принципе белого ящика:

☐ Покрытие операторов

☐ Покрытие решений

☐ Покрытие условий

☐ Покрытие решений/условий

☐ Комбинаторное покрытие условий

13. Процессор Cortex-M3: режимы работы и уровни привилегий выполнения программы; рабочие состояния; основной стек и стек процесса; действия после сброса.

3.3. Режимы работы

Процессор Cortex-M3 поддерживает два режима работы и два уровня доступа к коду программы (Рис. 3.6).

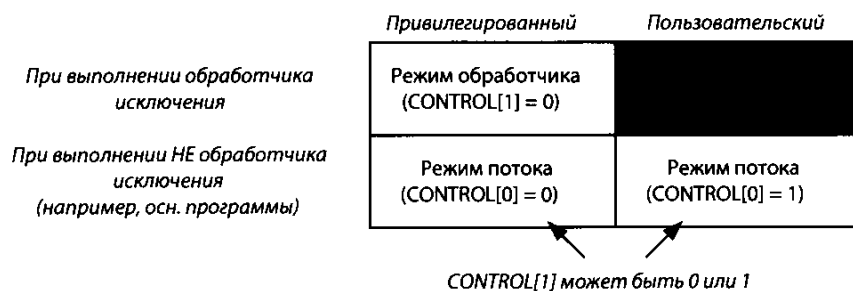


Рис. 3.6. Режимы работы и уровни доступа к коду процессора Cortex-M3.

При работе процессора в режиме потока он может находиться как на привилегированном, так и на пользовательском уровне, тогда как обработка исключительных ситуаций всегда осуществляется на привилегированном уровне. После сброса процессор находится в режиме потока с правами привилегированного доступа к коду.

На пользовательском уровне доступа (режим потока) обращения к пространству управления системой (System Control Space — SCS) — области памяти, содержащей регистры конфигурации и компоненты отладки, — заблокированы. Более того, на этом уровне запрещено даже использование команд, обращающихся к регистрам специального назначения (таких как MSR, за исключением обращений к регистру APSR). Если программа, выполняющаяся на пользовательском уровне доступа, попытается обратиться к пространству SCS или к регистрам специального назначения, то будет сгенерировано исключение отказа.

Программное обеспечение, работающее на привилегированном уровне, может переключиться на пользовательский уровень, используя регистр управления. При возникновении исключительной ситуации процессор всегда переключается на привилегированный уровень, а при выходе из обработчика исключения — возвращается на исходный. Пользовательская программа не может сама переключиться на привилегированный уровень, выполнив запись в регистр управления. Она должна воспользоваться обработчиком исключения, который запрограммирует регистр CONTROL так, чтобы процессор переключился на привилегированный уровень при возврате в режим потока (Рис. 3.7).

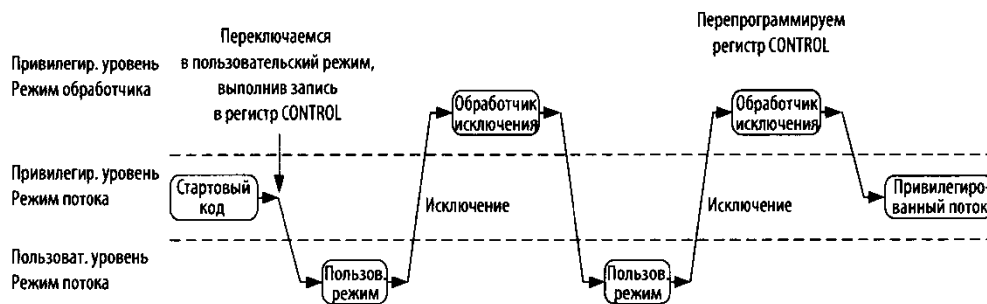


Рис. 3.7. Переключение режимов работы процессора.

Поддержка процессором двух уровней доступа обеспечивает более безопасную и надёжную архитектуру. Например, при некорректной работе пользовательской программы она никогда не сможет повредить содержимое регистров управления контроллера прерываний. А при наличии модуля MPU можно вообще заблокировать пользовательским программам доступ к областям памяти, используемым привилегированными процессами.

В простых приложениях нет нужды разделять привилегированный и пользовательский уровни доступа. В таких случаях пользовательский уровень доступа не используется и, соответственно, программирование регистра управления не требуется.

Вы можете отделить стек пользовательского приложения от стека ядра операционной системы, чтобы предотвратить крах системы из-за некорректной работы со стеком в пользовательской программе. В этом случае пользовательская программа, работающая в режиме потока, будет использовать указатель PSP, а обработчики исключений — указатель MSP. Переключение между указателями стеков осуществляется автоматически при входе в обработчики и при выходе из них (см. подраздел 3.6.3). Более подробно этот вопрос рассматривается в Главе 8.

Режим работы процессора и уровень доступа к коду определяются регистром управления. Если 0-й бит регистра управления сброшен в 0, то при генерации исключения режим процессора будет изменён (Рис. 3.8 и 3.9).

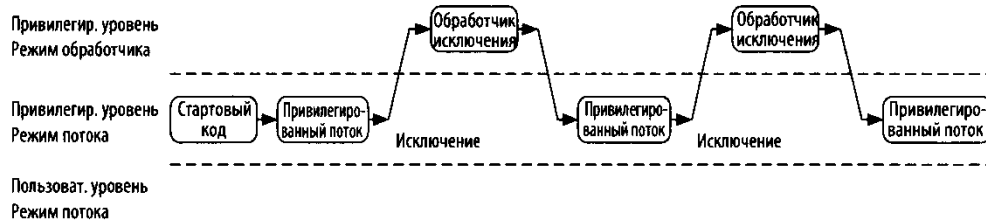


Рис. 3.8. Простая программа, не использующая пользовательский уровень доступа в режиме потока.

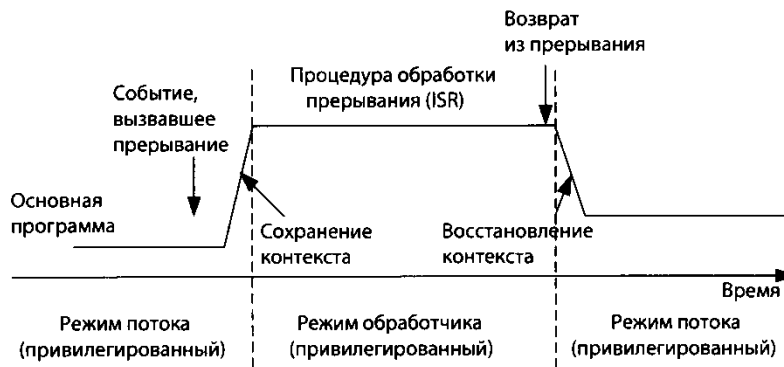


Рис. 3.9. Переключение режима работы процессора в прерывании.

Если 0-й бит регистра управления установлен в 1, то при генерации исключения будут изменены и режим процессора, и уровень доступа (Рис. 3.10).

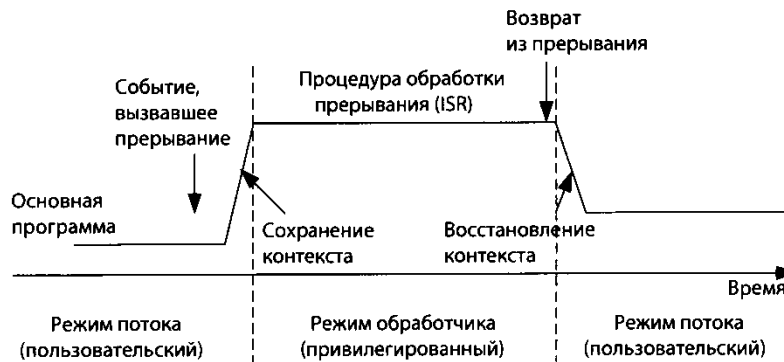


Рис. 3.10. Переключение режима работы процессора и уровня доступа в прерывании.

Запись в 0-й бит регистра управления допускается только на привилегированном уровне (см. Рис. 2.5). Чтобы программа, выполняющаяся на пользовательском уровне, смогла переключиться в привилегированное состояние, она должна сгенерировать прерывание (например, посредством команды вызова супервизора SVC) и изменить бит CONTROL[0] в обработке прерывания.

3.4. Исключения и прерывания

Процессор Cortex-M3 поддерживает определённое число исключений, в том числе фиксированное количество системных исключений и ряд прерываний, обычно называемых IRQ. Количество входов прерываний в микроконтроллерах с процессором Cortex-M3 зависит от конкретной реализации. Прерывания, генерируемые периферийными устройствами процессора, за исключением системного таймера, тоже подключаются к входам прерываний. Как правило, используется 16 или 32 входа прерываний. Однако вы вполне можете встретить микроконтроллеры, имеющие большее (или меньшее) число прерываний.

Помимо входов обычных прерываний, в процессоре также имеется вход немаскируемого прерывания (NMI). Фактическое использование немаскируемого прерывания зависит от конкретного микроконтроллера или системы на кристалле. В большинстве случаев вход NMI может быть подключён к сторожевому таймеру или блоку контроля напряжения питания, который сообщит процессору о падении напряжения ниже заданного уровня. Немаскируемое прерывание может быть активировано в любой момент времени, в том числе сразу же после выхода процессора из состояния сброса.

Перечень исключений, поддерживаемых процессором Cortex-M3, приведён в Табл. 3.4. Некоторые системные исключения предназначены для обработки отказов процессора и могут генерироваться при возникновении различных нештатных ситуаций. В контроллере NVIC также имеется несколько регистров состояния отказов, с помощью которых обработчики исключений могут определить причину возникновения исключительной ситуации.

Более подробно исключения процессора Cortex-M3 рассматриваются в Главах 7...9.

Таблица 3.4. Типы исключений в процессоре Cortex-M3

Номер исключения	Тип исключения	Приоритет	Описание
1	Reset	-3 (наивысший)	Сброс
2	NMI	-2	Немаскируемое прерывание
3	Hard Fault	-1	Любой отказ, если соответствующий обработчик не может быть запущен (исключение в данный момент запрещено или маскировано)
4	MemManage Fault	Программируемый	Отказ системы управления памятью; нарушение правил доступа, заданных модулем MPU, или обращение по некорректному адресу (например, при выборке команды из области памяти, не предназначенной для хранения исполняемого кода)
5	Bus Fault	Программируемый	Отказ шины; возникает при отказе предвыборки команды или при ошибке доступа к данным
6	Usage Fault	Программируемый	Отказ программы; обычно возникает при попытке исполнить неверную команду или переключиться в недопустимое состояние (скажем, переключить процессор Cortex-M3 в состояние ARM)
7...10	—	—	Зарезервировано
11	SVCall	Программируемый	Вызов супервизора посредством команды SVC
12	Debug monitor	Программируемый	Исключение монитора отладки
13	—	—	Зарезервировано
14	PendSV	Программируемый	Запрос системной службы
15	SYSTICK	Программируемый	Системный таймер
16...255	IRQ	Программируемый	Вход прерывания №0...239

3.5. Таблица векторов

При возникновении события, генерирующего исключение (если данное исключение разрешено), запускается обработчик соответствующего исключения. Для определения начального адреса обработчика предназначена таблица векторов. *Таблица векторов* представляет собой массив 32-битных значений, расположенных в системной области памяти, каждое из которых является начальным адресом обработчика одного конкретного исключения. Положение таблицы векторов можно изменять — оно определяется регистром смещения таблицы векторов VTOR контроллера NVIC. После сброса этот регистр содержит нулевое значение, поэтому при включении процессора таблица векторов располагается, начиная с адреса 0x00 (см. Табл. 3.5).

Таблица 3.5. Таблица векторов после сброса

Номер исключения	Смещение	Вектор обработчика исключения
18...255	0x48...0x3FF	IRQ №2...239
17	0x44	IRQ №1
16	0x40	IRQ №0
15	0x3C	SYSTICK
14	0x38	PendSV
13	0x34	Зарезервировано
12	0x30	Debug monitor
11	0x2C	SVCall
7...10	0x1C...0x28	Зарезервировано
6	0x18	Usage Fault
5	0x14	Bus Fault
4	0x10	MemManage Fault
3	0x0C	Hard Fault
2	0x08	NMI
1	0x04	Сброс
0	0x00	Начальное значение MSP

К примеру, если событие сброса является исключением с номером 1, то адрес вектора сброса равен 1×4 (каждое слово содержит 4 байта), т.е. 0x00000004. Вектор NMI (номер 2) расположен по адресу $2 \times 4 = 0x00000008$. Адрес 0x00000000 используется для хранения начального значения основного указателя стека.

Младший бит каждого вектора указывает, в каком состоянии должен выполняться соответствующий обработчик. Поскольку процессор Cortex-M3 поддерживает только команды Thumb, младшие биты всех векторов исключений должны быть установлены в 1.

3.6. Стек

Процессор Cortex-M3 поддерживает обычные программно-управляемые операции загрузки в стек и извлечения из стека. Кроме того, операции загрузки в стек и извлечения из стека выполняются автоматически при входе и выходе из

обработчика исключения/прерывания. В данном разделе мы рассмотрим программную работу со стеком (стековые операции при обработке исключений рассматриваются в Главе 9).

3.6.1. Основные стековые операции

В общем случае стековые операции представляют собой операции записи или чтения памяти по адресу, определяемому указателем стека. Содержимое регистров сохраняется в стеке посредством операции загрузки в стек и может быть восстановлено позже посредством операции извлечения из стека. При выполнении этих операций указатель стека изменяется автоматически таким образом, чтобы многократная загрузка данных в стек не привела к стиранию ранее сохранённых данных.

Стек предназначен для временного сохранения содержимого регистров в памяти и последующего его восстановления после завершения некоторой задачи. При обычном использовании стека каждой операции загрузки должна соответствовать своя операция извлечения, причём адрес, используемый в этих операциях, должен быть одним и тем же (Рис. 3.11). При выполнении команд PUSH/POP указатель стека инкрементируется и декрементируется автоматически.

Основная программа

```

...
; R0  X, R1  Y, R2  Z
BL   function1      Подпрограмма

function1
    PUSH    {R0} ; Сохраняем R0 в стеке (SP инкрементируется)
    PUSH    {R1} ; Сохраняем R1 в стеке (SP инкрементируется)
    PUSH    {R2} ; Сохраняем R2 в стеке (SP инкрементируется)
    ... ; Выполняем задачу (R0, R1 и R2
        ; можно изменять)
    POP     {R2} ; Восстанавливаем R2 (SP декрементируется)
    POP     {R1} ; Восстанавливаем R1 (SP декрементируется)
    POP     {R0} ; Восстанавливаем R0 (SP декрементируется)
    BX      LR ; Возврат

; Вернулись в основную программу
; R0  X, R1  Y, R2  Z
... ; Следующие команды

```

Рис. 3.11. Основные стековые операции: одна операция — один регистр.

При возврате управления в основную программу содержимое регистров R0...R2 будет таким же, что и до вызова подпрограммы. Обратите внимание на порядок выполнения операций: операции извлечения из стека выполняются в обратном порядке по сравнению с операциями загрузки в стек.

Данный процесс можно упростить, так как команды PUSH и POP поддерживают сохранение и восстановление нескольких регистров. В этом случае порядок

регистров, восстанавливаемых из стека, автоматически изменяется процессором на обратный (Рис. 3.12).

Основная программа

```

...
; R0 X, R1 Y, R2 Z
BL function1

```

Подпрограмма

```

function1
PUSH {R0 R2} ; Сохраняем R0, R1, R2 в стеке
... ; Выполняем задачу (R0, R1 и R2
; можно изменять)
POP {R0 R2} ; Восстанавливаем R0, R1, R2
BX LR ; Возвращаемся

```

; Вернулись в основную программу
; R0 X, R1 Y, R2 Z
... ; Следующие команды

Рис. 3.12. Основные стековые операции: одна операция — несколько регистров.

Вы также можете объединить операцию возврата из подпрограммы с операцией извлечения из стека. Для этого необходимо сохранить содержимое регистра LR в стеке и извлечь его в счётчик команд в конце подпрограммы (Рис. 3.13).

Основная программа

```

...
; R0 X, R1 Y, R2 Z
BL function1

```

Подпрограмма

```

function1
PUSH {R0 R2, LR} ; Сохраняем регистры,
; включая регистр связи
... ; Выполняем задачу (R0, R1 и R2
; можно изменять)
POP {R0 R2, PC} ; Восстанавливаем регистры
; и возвращаемся

```

; Вернулись в основную программу
; R0 X, R1 Y, R2 Z
... ; Следующие команды

Рис. 3.13. Основные стековые операции: объединение операции извлечения из стека с операцией возврата.

3.6.2. Реализация стека в процессоре Cortex-M3

В процессоре Cortex-M3 используется модель «полного» убывающего стека. Указатель стека указывает на последнее значение, помещённое в стек, и инкрементируется перед выполнением новой операции загрузки в стек. В качестве примера на Рис. 3.14 показано выполнение команды `PUSH {R0}`.

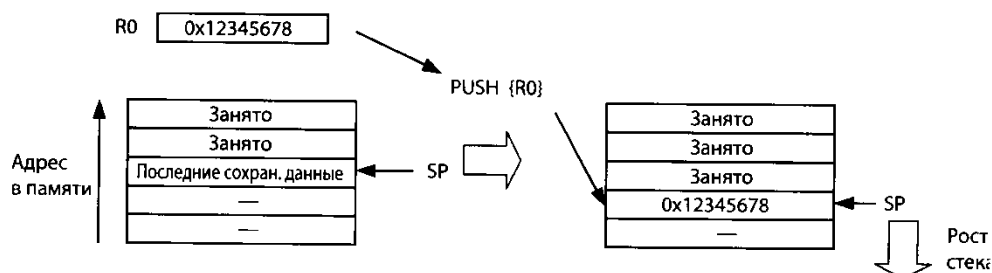


Рис. 3.14. Реализация операции загрузки в стек.

Во время операции извлечения из стека данные считываются из памяти по адресу, определяемому указателем стека, после чего указатель инкрементируется. Содержимое ячейки памяти не изменяется, однако оно будет перезаписано при выполнении последующей операции загрузки в стек (Рис. 3.15).

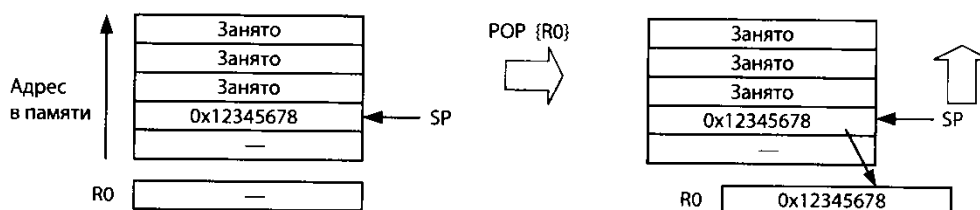


Рис. 3.15. Реализация операции извлечения из стека.

Поскольку при выполнении каждой операции загрузки в стек и извлечения из стека осуществляется пересылка 4 байт данных (каждый регистр содержит одно слово, или 4 байт), значение указателя стека уменьшается/увеличивается на 4 или, при одновременном сохранении/восстановлении нескольких регистров, на величину, кратную четырём.

В процессоре Cortex-M3 в качестве указателя стека используется регистр R13. При возникновении прерывания некоторые регистры будут автоматически сохранены в стеке. При возврате из обработчика прерывания эти регистры также автоматически будут восстановлены из стека, а значение указателя стека изменится соответствующим образом.

3.6.3. Два стека процессора Cortex-M3

Как уже было сказано, в процессоре Cortex-M3 имеется два указателя стека: основной MSP и дополнительный PSP. Используемый в настоящий момент указатель определяется битом 1 регистра управления (далее этот бит обозначается как CONTROL[1]).

Если бит CONTROL[1] сброшен в 0, то указатель MSP используется в обоих режимах работы процессора (Рис. 3.16). В этом случае и основная программа, и обработчики исключений задействуют под стек одну и ту же область памяти. Такое поведение используется по умолчанию после включения процессора.

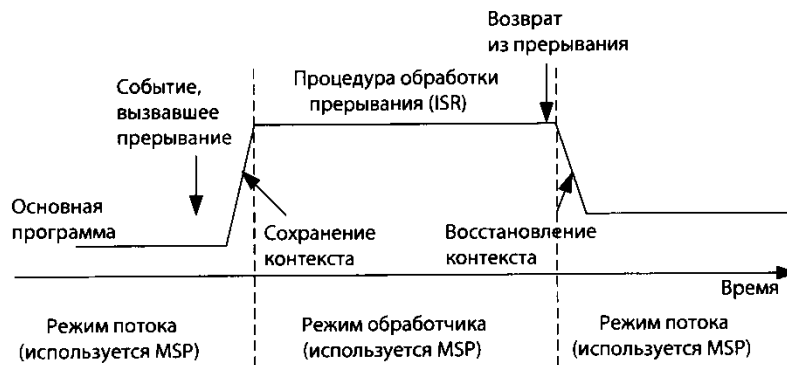


Рис. 3.16. $CONTROL[1] = 0$; основной стек используется и в режиме потока, и в режиме обработчика.

Если бит $CONTROL[1]$ установлен в 1, то в режиме потока используется указатель PSP (**Рис. 3.17**). В этом случае основная программа и обработчики прерываний задействуют под стеки разные области памяти. Это позволяет предотвратить порчу стека, используемого операционной системой, при наличии ошибок в пользовательской программе (предполагается, что пользовательская программа выполняется только в режиме потока, а ядро ОС выполняется в режиме обработчика).

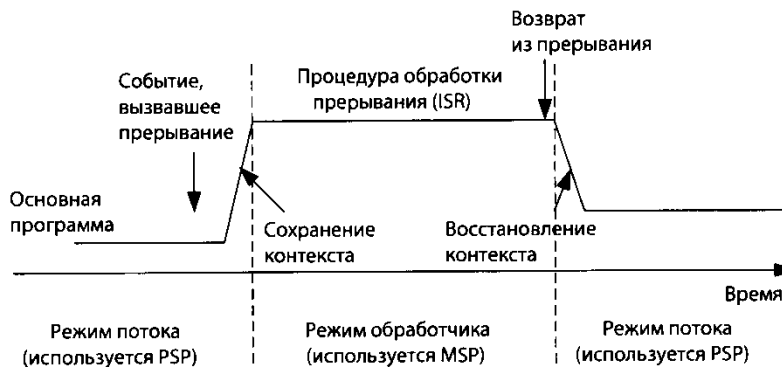


Рис. 3.17. $CONTROL[1] = 1$; в режиме потока используется стек процесса, а в режиме обработчика — основной стек.

Заметьте, что в этом случае при автоматическом сохранении и восстановлении регистров будет использоваться дополнительный стек, тогда как при стековых операциях внутри обработчиков — основной стек.

В процессоре предусмотрена возможность непосредственной записи/чтения указателей MSP и PSP, независимо от того, на какой из них ссылается в данный момент регистр R13. При работе на привилегированном уровне для доступа к содержимому указателей MSP и PSP можно использовать следующие функции:

```
x = __get_MSP(); // Чтение значения MSP
__set_MSP(x);    // Установка значения MSP
x = __get_PSP(); // Чтение значения PSP
```

```
__set_PSP(x);    // Установка значения PSP
```

Вообще говоря, изменять значение текущего указателя стека внутри Си-функций крайне не рекомендуется, поскольку в стеке могут храниться локальные переменные. Для обращения к указателям стека из ассемблерного кода можно использовать уже известные нам команды MRS и MSR:

```
MRS R0, MSP ; Чтение основного указателя стека в R0
MSR MSP, R0 ; Запись R0 в основной указатель стека
MRS R0, PSP ; Чтение указателя стека процесса в R0
MSR PSP, R0 ; Запись R0 в указатель стека процесса
```

Считав значение PSP посредством команды MRS, ОС может прочитать данные, помещённые в стек пользовательской программой (скажем, содержимое регистров перед вызовом супервизора командой SVC). Более того, ОС может изменить значение указателя PSP, например при переключении контекста в многозадачной системе.

3.7. Цикл сброса

После выхода процессора из состояния сброса он считывает из памяти два 32-битных значения (Рис. 3.18):

- по адресу 0x00000000 — начальное значение R13 (указателя стека);
- по адресу 0x00000004 — вектор сброса (адрес, с которого начинается выполнение программы; младший бит значения должен быть установлен в 1 для указания состояния Thumb).

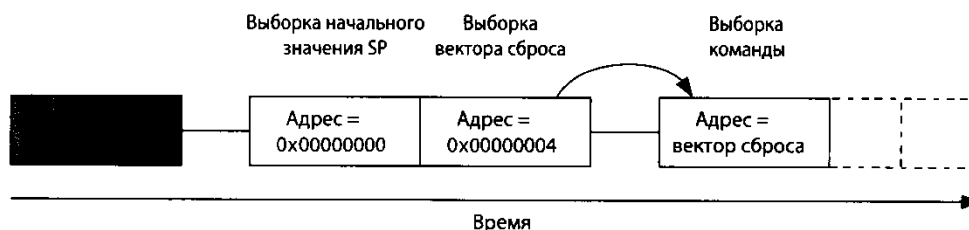


Рис. 3.18. Цикл сброса.

Эта последовательность отличается от поведения традиционных процессоров ARM. Предыдущие процессоры компании ARM выполняли программный код, начиная с нулевого адреса. Более того, таблица векторов предыдущих процессоров ARM содержала команды (вы должны были поместить в таблицу команду перехода на начало обработчика исключения, расположенного в каком-нибудь другом месте).

В процессоре же Cortex-M3 в самом начале карты памяти размещается начальное значение MSP, после него располагается таблица векторов (позже, во время выполнения программы, таблица векторов может быть перемещена в другое место). К тому же в таблице векторов содержатся не команды перехода, а значения адресов. Первым вектором в таблице является вектор сброса, который представляет собой второе значение, считываемое процессором при выходе из состояния сброса.

Поскольку в процессоре Cortex-M3 реализована модель «полного» убывающего стека (указатель стека декрементируется перед загрузкой значения в стек), на-

14. Процессор Cortex M3 :карта памяти; побитовый доступ; обращение к не выровненным данным;

Процессоры на базе ядра Cortex-M3 имеют простую, фиксированную карту памяти с максимальным адресуемым пространством вплоть до 4 гигабайт, с predetermined, заранее выделенными адресными массивами для памяти программ, ОЗУ, устройствами внешней памяти или периферийными устройствами, а также встроенной периферией (рис. 1). Также имеется специальная область памяти, которая содержит адреса, зарезервированные производителем.

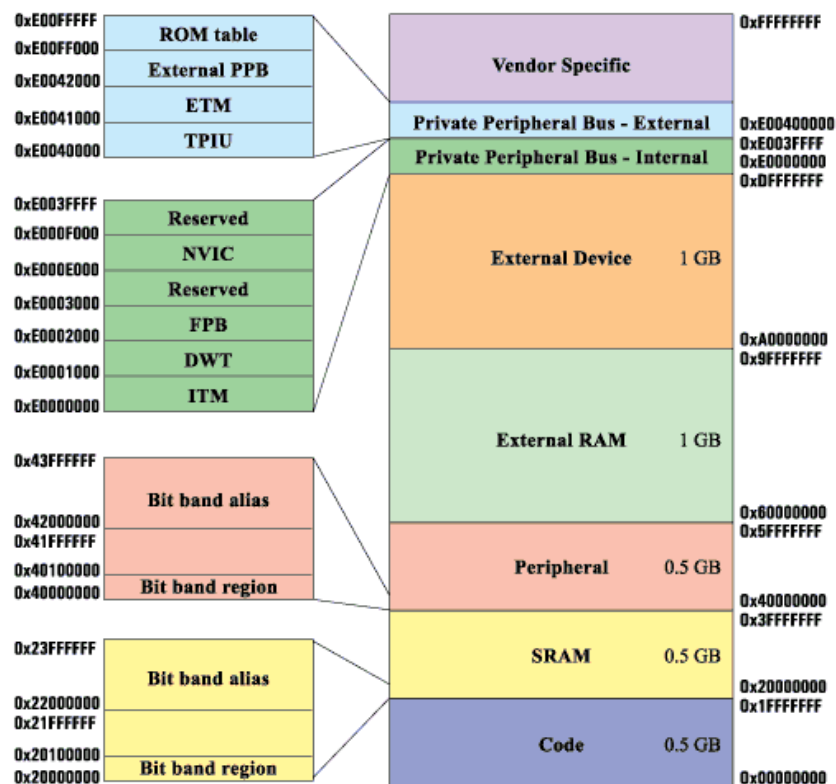


Рис. 1. Карта памяти процессоров с архитектурой Cortex-M3

Процессоры на базе ядра CortexM3 обеспечивают прямой доступ к битам данных при помощи довольно простого механизма (рис. 2). Заметим, что операции с битами являются элементарными и не могут быть прерванными другими операциями, в том числе и прерываниями.

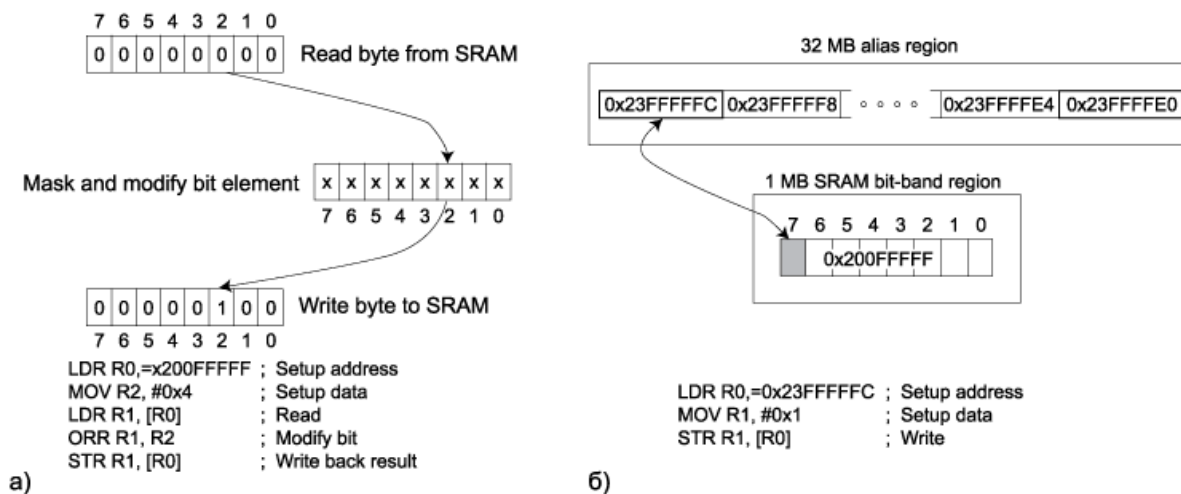
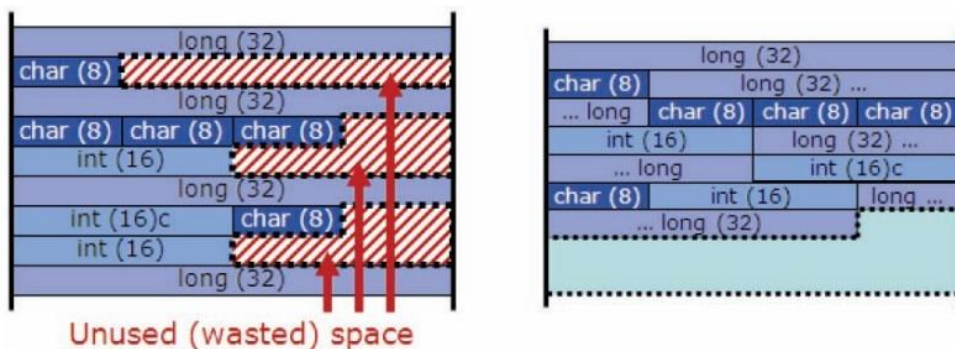


Рис. 2. Сравнение механизма битовых операций в традиционной архитектуре ARM7 (a) и Cortex-M3 (б)

Традиционные процессоры на базе архитектуры ARM7 поддерживают только доступ к выровненным данным, что подразумевает, что сохраняемые и вычитываемые данные должны быть выровнены по границе

слова. Процессоры же на базе архитектуры Cortex-M3 позволяют обращаться к невыровненным данным и сводят к минимуму временные задержки, связанные с доступом к данным. В случае, если происходит обращение к невыровненным данным, это обращение разбивается на несколько параллельных обращений к выровненным данным, но этот процесс является прозрачным для программы пользователя, поскольку происходит автоматически внутри ядра.

ЦПУ Cortex имеет режимы адресации для слов, полуслов и байт, но он может осуществлять и не выровненный доступ к памяти. Это дает компоновщику дополнительную свободу при размещении данных программы в памяти. За счет поддержки битовой сегментации, ЦПУ Cortex может упаковывать программные флаги в переменные формата полуслова и слова и не использовать отдельный байт для каждого флага.



Cortex-M3 может осуществлять невыровненный доступ к памяти, что увеличивает эффективность использования SRAM.

Перевод к рисунку: *Unused (wasted) space – Неиспользуемое (потеряное) пространство*

15. Процессор Cortex M3: состояния и типы исключений; обработка исключений; приоритеты исключений; контроллер вложенных векторов прерываний(NVIC); системный таймер

Состояние исключений

Каждое прерывание может находится в одном из 4-х состояний:

- Неактивное/Inactive
- Отложенное/Pending
- Активное/Active
- Активное и отложенное/Active and pending (состояние возникает, когда во время обработки прерывания наступило прерывание от того же источника).

Типы исключений

Таблица векторов прерываний Cortex находится в низу адресного пространства. Первым адресом таблицы векторов прерывания является 0x00000004. Первые четыре байта используются для хранения начального адреса указателя стека.

№	Тип Исключения	Приоритет	Тип Приоритета	Описание
1	Reset	-3 (Высший)	фиксирован	Сброс
2	NMI	-2	Фиксирован	Немаскируемое прерывание
3	Hard Fault	-1	фиксирован	Ошибка по умолчанию, если другие пункты неприменимы
4	MemManage Fault	0	задается	Нарушение МПУ или обращение к запрещенной области
5	Bus Fault	1	задается	Интерфейс АНВ принял ошибку
6	Usage Fault	2	задается	Исключение из-за программной ошибки
7-10	Зарезервировано	недоступно		
11	SVCall	3	задается	Вызов Системного Сервиса
12	Dedug Monitor	4	задается	Точки останова, точки просмотра, внешняя отладка
13	Зарезервировано	Недоступно		
14	PendSV	5	задается	«Подвешиваемый» запрос для Системного Устройства
15	SYSTICK	6	задается	Таймер System Tick
16	Прерывание #0	7	задается	Внешнее прерываний №0
.....	задается
256	Прерывание #240	247	задается	Внешнее прерывание №240

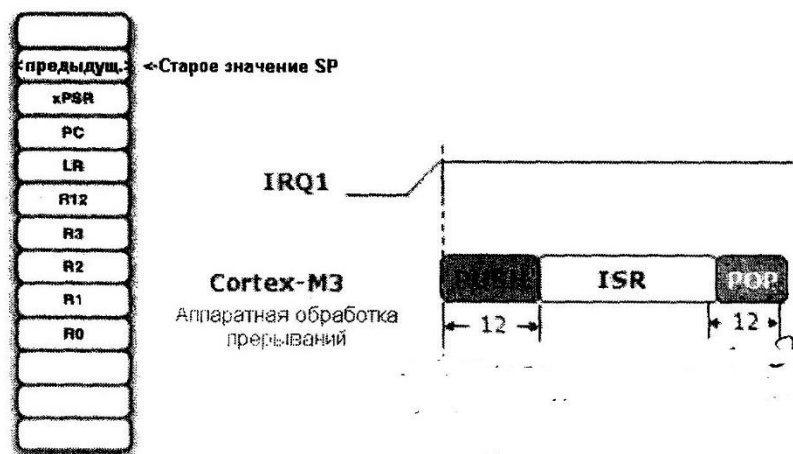


Рис. 6. КВВП (NVIC) реагирует на обработку прерывания с задержкой всего лишь 12 циклов. В них входит выполнение микрокода, который автоматически помещает набор регистров в стек

NVIC реагирует на прерывание с задержкой только 12 циклов. За это время подпрограмма микрокода автоматически сохраняет значения набора регистров в стеке.

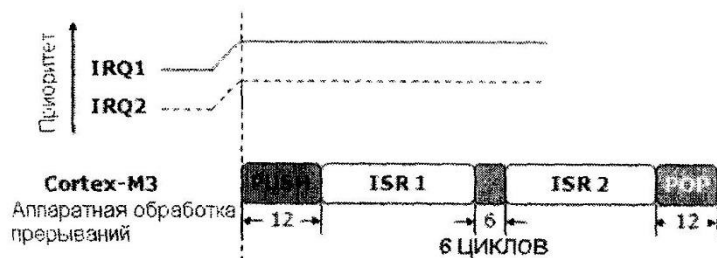


Рис. 7. Несколько прерываний обрабатываются непрерывно с исключением внутренних операций над стеком, поэтому задержка после завершения обработки одного прерывания и перед началом обработки следующего минимальна

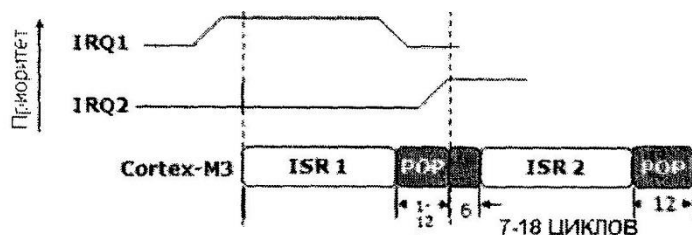


Рис. 8. Переход к обработке низкоприоритетного прерывания, которое возникает при выходе из текущего прерывания, выполняется с задержкой 7-18 циклов

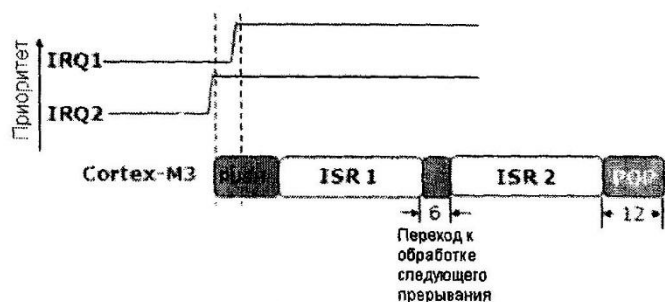


Рис. 9. Высокоприоритетное прерывание будет обработано первым, даже если оно возникнет уже на фазе перехода к обработке низкоприоритетного прерывания, при этом, дополнительные операции над стеком будут исключены

Таймер SysTick – это 24-битный автоматически перезагружающийся таймер в составе процессора Cortex-M3. Он предназначен для отсчета интервалов. Операционной Системы Реального Времени. Таймер SysTick содержит три регистра. Текущее значение и значение перезагрузки должны устанавливаться вместе с периодом счета. Регистр управления и статуса содержит бит ENABLE для запуска таймера и бит TICKINT для включения линии прерываний.

16. Процессор Cortex M3: режимы пониженного энергопотребления

Назначение битов регистра ядра SCR

SEVONPEND (Send Event on Pending bit) (Считать запрос IRQ событием пробуждения)

Когда событие или прерывание ставят флаг запроса, то это пробуждает процессор из состояния WFE. Если процессор не ждет этого события, то оно регистрируется и имеет эффект на то, что следует после состояния WFE. Процессор также просыпается при выполнении инструкции SEV или от внешнего события.

0: Только разрешенные прерывания или события могут разбудить процессор.

1: Разрешенные события и все прерывания, включая запрещенные, могут разбудить процессор

SLEEPDEEP
Управляет, будет ли процессор использовать простой сон или глубокий сон при входе в режим малого потребления:

0: Sleep

1: Deep sleep. (Глубокий сон)

SLEEPONEXIT

Конфигурация режима сон-по-выходу, при возврате из обработчика прерывания в режим Thread.

Установка этого бита в '1' дает возможность приложению, управляемому только от прерываний, избежать возвращения в пустую функцию main() .

0: Нет засыпания при возврате в режим потока(Thread).

1: Вход в режим сна или глубокого сна по возвращению из процедуры обработки прерывания.

Дополнительные возможности снижения энергопотребления реализуются в микроконтроллерах.

Например STM 32F0xxx имеют три режима с низким потреблением

Sleep Mode дежурный режим, режим ожидания (синхросигнал МП ключен ,вся периферия, включая периферию ядра Cortex-M3)

Stop mode-все синхросигналы выключены

Standby mode-резервный режим (1.8 область обесточена)

10.4.4. Использование стандарта CMSIS

Поскольку стандарт CMSIS, условно говоря, «внедрён» в библиотеки драйверов устройств, то для его использования в проекте не требуется предпринимать никаких специальных мер. Для каждого микроконтроллера производитель предоставляет заголовочный файл `<device>.h`, который, в свою очередь, содержит директивы вставки дополнительных заголовочных файлов, требуемых библиотекой драйвера устройства, в том числе и заголовочный файл уровня доступа к периферии ядра, разработанный компанией ARM (Рис. 10.8).

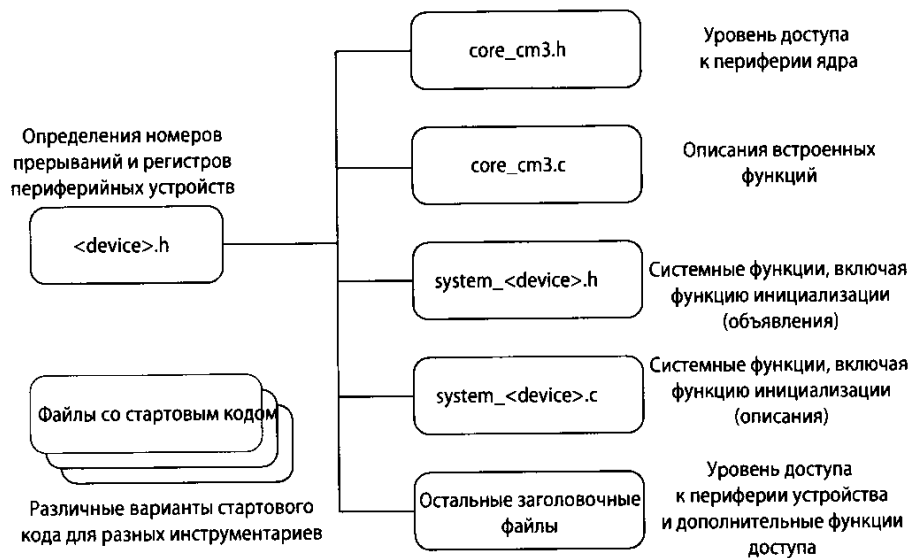


Рис. 10.8. Файлы CMSIS.

Файл `core_cm3.h` содержит определения регистров периферийных устройств и описания функций доступа к периферии процессора Cortex-M3, такой как контроллер NVIC, регистры управления системой и регистры системного таймера SYSTICK. Файл `core_cm3.h` также содержит объявления встраиваемых функций стандарта CMSIS, позволяющих использовать команды процессора, которые не могут быть сгенерированы при использовании стандартных конструкций языка. Кроме того, этот файл содержит описание функции для вывода отладочной информации через модуль ITM.

Следует заметить, что имена некоторых встроенных функций CMSIS могут совпасть с именами встроенных функций компилятора, однако функции CMSIS не зависят от используемого компилятора.

Файл `core_cm3.c` содержит описания встроенных функций CMSIS, которые не могут быть реализованы в файле `core_cm3.h` в виде макроопределений.

Файл `system_<device>.h` содержит объявление, а файл `system_<device>.c` — описание функции `SystemInit()`, предназначенной для инициализации системы (реализация этой функции зависит от конкретного микроконтроллера).

В самом файле `<device>.h` содержатся определения номеров прерываний и регистров периферийных устройств конкретного устройства.

Кроме того, CMSIS-совместимые драйверы устройств также содержат стартовый код (с таблицей векторов) для всех поддерживаемых компиляторов, а также CMSIS-версии встроенных функций, что позволяет встроенному ПО использовать все возможности ядра независимо от применяемого компилятора.

Примеры использования стандарта CMSIS можно найти на сайтах производителей микроконтроллеров. Подобные примеры также можно найти в самих библиотеках драйверов. Кроме того, вы можете загрузить с сайта www.onarm.com пакет CMSIS, содержащий примеры и документацию. В этом же пакете можно найти документацию на функции общего назначения.

Простейший пример использования стандарта CMSIS при разработке собственного приложения показан на **Рис. 10.9**. Чтобы воспользоваться функциями CMSIS для конфигурирования прерываний и исключений, необходимо задействовать константы, определённые в файле `<device>.h`. Значения указанных констант отличаются от номеров исключений, используемых регистрами ядра, например регистром IPSR. В CMSIS для системных исключений используются отрицательные значения, а для прерываний периферии — положительные.

```
#include "vendor_device.h" // Например,
// lm3s_cmsis.h для микроконтроллеров Texas Instruments
// LPC17xx.h для микроконтроллеров NXP
// stm32f10x.h для микроконтроллеров ST

void main(void) {
    SystemInit();
    ...
    NVIC_SetPriority(UART1_IRQn, 0x0);
    NVIC_EnableIRQ(UART1_IRQn);
    ...
}

void UART1_IRQHandler {
    ...
}

void SysTick_Handler(void) {
    ...
}
```

Унифицированное имя функции инициализации системы (начиная с версии 1.30 CMSIS эта функция вызывается из стартового кода)

Настройка контроллера NVIC посредством функций доступа к ядру

Номера прерываний, определённые в `<device.h>`

Имена обработчиков прерываний зависят от конкретного устройства и определяются в файле со стартовым кодом

Имена обработчиков системных исключений одинаковы для всех микроконтроллеров семейства Cortex

Рис. 10.9. Пример использования CMSIS.

Чтобы создать переносимое приложение, вы должны использовать для обращения к процессору и периферийным устройствам функции доступа к ядру и функции промежуточного ПО соответственно. Это позволит вам перенести приложения с одного микроконтроллера на другой с минимальными усилиями.

Более подробно CMSIS-функции, как общего назначения, так и встроенные, описаны в Приложении Ж.

10.4.5. Выгода от использования CMSIS

И всё же, что стандарт CMSIS даёт конечному пользователю?

Прежде всего, стандарт CMSIS значительно улучшает переносимость и увеличивает возможность повторного использования программного кода. Следование этому стандарту не только облегчает переход между разными микроконтроллерами с ядром Cortex-M3, но и позволяет ускорить процесс переноса программного обеспечения с процессора Cortex-M3 на другие процессоры линейки Cortex-M, тем самым сокращая время вывода продукта на рынок.

Для разработчиков встраиваемых ОС и промежуточного ПО выгода от использования стандарта CMSIS гораздо существеннее. Применение этого стандарта позволит обеспечить совместимость их продукции с драйверами устройств для микроконтроллеров самых разных производителей, в том числе и для тех устройств, которые пока существуют только на бумаге (Рис. 10.10). Если не использовать CMSIS, то поставщики ПО должны либо предоставлять небольшую библиотеку функций для работы с ядром Cortex-M3, либо разрабатывать множество конфигураций для своей продукции, позволяющих ей работать с библиотеками от различных производителей микроконтроллеров.

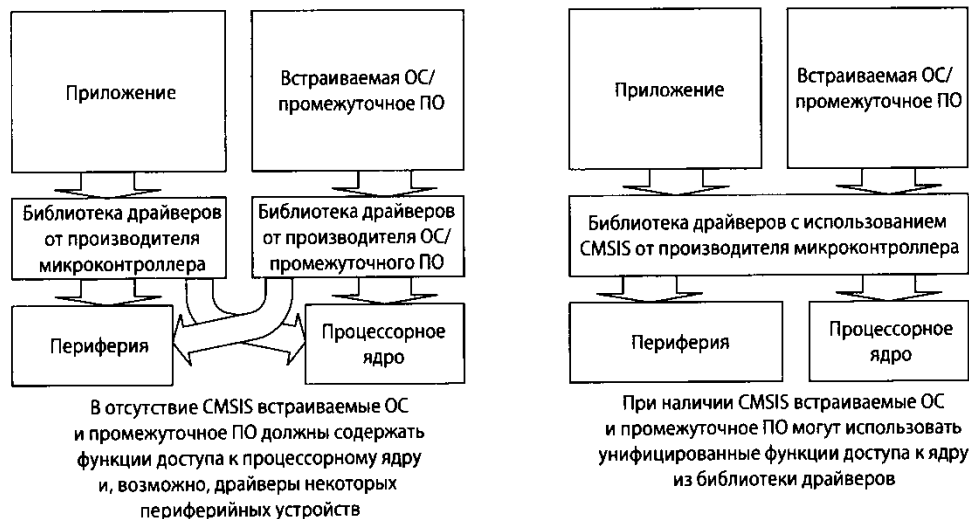


Рис. 10.10. Стандарт CMSIS как средство против дублирования кода.

Стандарт CMSIS очень нетребователен к памяти (необходимо менее 1 Кбайт для всех функций доступа к ядру и несколько байтов ОЗУ). Он также позволяет избежать дублирования кода драйвера периферии ядра при повторном использовании кода из другого проекта.

Поскольку стандарт CMSIS поддерживается многими разработчиками компиляторов, для компиляции встраиваемого ПО можно использовать самые разные компиляторы. Это позволяет разрабатывать встраиваемые ОС и промежуточное ПО, которые были бы независимы как от производителей микроконтроллеров, так и от производителей средств разработки. До появления стандарта