7 ТИПОВЫЕ АРХИТЕКТУРНЫЕ РЕШЕНИЯ ПОДСИСТЕМЫ ОБСЛУЖИВАНИЯ ПРЕРЫВАНИЙ МК ОБЩЕГО НАЗНАЧЕНИЯ

7.1 Общие вопросы

Подсистема обслуживания прерываний входит в состав практически всех семейств МК классов «*mainstream*» и «*high performance*», а также большинства семейств класса «*cost-sensitive*» отсутствует только в модельных рядах простейших МК данного класса (см., например, рис. 1.1). Назначение данной подсистемы - управление обслуживанием **запросов на прерывания** от периферийных устройств или ядра МК.

Прерыванием (*Interrupt*) называют приостановку выполнения основной программы по запросу от какого-либо ПУ (резидентного или внешнего по отношению к МК), а также от ядра МК, с выполнением подпрограммы обслуживания соответствующего запроса (*Interrupt Service Routine, ISR*) [3, 9]. Каждый из запросов на прерывание (*Interrupt Requests, IRQ*), в свою очередь, генерируется по наступлении определенного **события** (*Event*), т. е. некоторого явления, имевшего место в каком-либо функциональном узле или блоке МК, распознаваемого ЦП и / или ПУ МК и могущего вызвать определенные действия с их стороны, в том числе генерацию запроса на прерывание [9]. Типовым примером события является поступление очередного слова данных в регистр приемника какого-либо из блоков стандартного цифрового интерфейса МК, допустим, *USART* (см. раздел 11), которое, при определенных условиях, может вызвать прерывание основной программы, с переходом к подпрограмме считывания и анализа принятых данных.

В зависимости от источников запросов на прерывания, они подразделяются на следующие основные типы [3, 9]:

- прерывания от внешних по отношению к ЦП источников (например, АЦП, таймеров, внешних по отношению к МК устройств и т. п.), называемые **асинхронными** или **внешними**; первое из перечисленных названий происходит от того, что запросы на прерывания данного типа генерируются независимо от процесса выполнения программы и не «привязаны» во времени к сигналу синхронизации ЦП;

- прерывания, инициируемые некоторыми нештатными событиями, возникшими в ЦП при выполнении программного кода (например, делением на ноль или переполнением стека), и

называемые **синхронными** или **внутренними**, а также **исключениями**;

- прерывания, **инициируемые ПО**, называемые **программными** и фактически представляющие собой вызов некоторой подпрограммы.

С точки зрения практического применения МК, интерес представляют, в первую очередь, асинхронные прерывания. Они используются для реакции на штатные, но не «подконтрольные» ПО МК события. В вышеприведенном примере данные на приемник блока *USART* поступают, как правило, с внешнего по отношению к МК источника (обычно – с передатчика блока *USART* другого МК). Следовательно, моменты их поступления заранее «не известны» ПО МК. Для их корректного считывания, в принципе, могут быть применены два подхода.

Первый из них основывается на обслуживании соответствующего события (в данном конкретном примере – считывании регистра приемника), инициируемом ПО МК, и состоит в следующем:

- ПО МК в цикле постоянно опрашивает расположенный в регистре статуса *USART* бит («флаг») признака поступления данных в регистр приемника и их готовности для считывания;

- по установке указанного бита в активное состояние осуществляются считывание принятых данных и необходимые операции по их обработке, после чего бит готовности данных сбрасывается программно, и ПО переходит к ожиданию поступления очередных данных, с циклическим опросом бита готовности.

Второй подход состоит в использовании механизма прерываний и реализуется следующим образом:

- ЦП выполняет основную программу контроля и управления объектом;

- *USART*, по поступлении данных в регистр его приемника, генерирует запрос на прерывание по событию «Готовность принятых данных для считывания»;

- ЦП, по получении запроса, приостанавливает (прерывает) выполнение основной программы и переходит к подпрограмме

обслуживания прерывания по вышеуказанному событию (считыванию и обработке принятых данных);

- по завершении подпрограммы обслуживания прерывания, ЦП возобновляет выполнение основной программы.

Нетрудно увидеть, что при обслуживании событий, моменты наступления которых «не подконтрольны» ПО МК, использование механизма прерываний обеспечивает значительно более эффективную работу ЦП, т. к.:

- при использовании первого из вышеописанных подходов ЦП должен постоянно отслеживать момент наступления соответствующего события, и не может выполнять никаких других задач;

- с другой стороны, при использовании механизма прерывания по событиям, подлежащим обслуживанию, в промежутках между ними ЦП может выполнять основную программу, и загружать его задачей отслеживания моментов их наступления нет необходимости, поскольку о соответствующем событии оповестит запрос на прерывание по нему.

Благодаря вышеуказанному преимуществу, прерывания широко используются как в МК, так и в средствах вычислительной техники в целом.

Таким образом, общими характеристиками механизма прерываний являются следующие:

- его рационально применять для реакции на события, моменты наступления которых не регулярны и не подконтрольны ПО МК;

- выполнение процедуры обслуживания некоторого события по прерыванию инициируется аппаратно, источником данного события (а не ПО МК); исключение составляют программные прерывания;

- собственно процедура обслуживания выполняется под управлением ПО (конкретно – подпрограммы обработки соответствующего запроса на прерывание).

7.2 Основные требования к подсистеме обслуживания прерываний и общие принципы их удовлетворения

На основании общих положений, изложенных в подразделе 7.1, могут быть сформулированы следующие базовые требования к подсистеме обслуживания прерываний МК [3, 9]:

- подсистема должна обеспечивать распознавание события, являющегося источником запроса на прерывание (в дальнейшем, для краткости – источника прерывания) и автоматический переход к подпрограмме обслуживания соответствующего запроса;
- должно обеспечиваться **приоритетное** обслуживание прерываний, т. е. первоочередная обработка запросов, вызванных событиями, более важными с точки зрения архитектуры МК или с точки зрения выполняемой им задачи (в общем случае – с приостановкой обслуживания запросов от менее приоритетных источников);
- при переходе к подпрограмме обслуживания запроса на прерывание должно быть обеспечено сохранение данных, необходимых для возврата основную программу (как минимум – адреса возврата, т. е. адреса команды, с которой должно быть возобновлено выполнение основной программы);
- по выполнении подпрограммы обслуживания запроса на прерывание должно быть обеспечено возобновление работы основной программы с того адреса, на котором произошла приостановка ее выполнения, вызванная прерыванием;
- должна быть обеспечена возможность индивидуального программного разрешения или запрета (**маскирования**) каждого из прерываний, кроме так называемых немаскируемых прерываний (см. пункт 7.2.4).

7.2.1. Распознавание источников запросов на прерывания и общие принципы их обслуживания

В современных МК (и средствах вычислительной техники в целом) применяется **векторная** система прерываний [3]. Ее сущность состоит в следующем. Каждому источнику прерывания

присваивается определенный номер – вектор. В свою очередь, каждому вектору ставится в соответствие определенный адрес, по которому в памяти программ должна размещаться начальная команда подпрограммы обслуживания прерывания (*ISR*) от соответствующего источника. По получении от него запроса на прерывание ЦП автоматически переходит к команде, размещенной по указанному адресу. Таблица соответствия начальных адресов *ISR* векторам прерываний называется **таблицей векторов прерываний** и определяется архитектурой конкретного семейства / подсемейства / модельного ряда МК.

В качестве примера на рис. 7.1 представлен фрагмент такой таблицы МК 1887ВЕ7Т (аналог *ATmega128*) [8].

Номер вектора	Адрес памяти программ	Источник	Условие возникновения прерывания
1	$0000	RESET	Внешний сброс, сброс при подаче питания, сброс при снижении питания, сброс от сторожевого таймера и сброс через JTAG-интерфейс
2	$0002	INT0	Запрос на внешнее прерывание 0
3	$0004	INT1	Запрос на внешнее прерывание 1
4	$0006	INT2	Запрос на внешнее прерывание 2
5	$0008	INT3	Запрос на внешнее прерывание 3
6	$000A	INT4	Запрос на внешнее прерывание 4
7	$000C	INT5	Запрос на внешнее прерывание 5
8	$000E	INT6	Запрос на внешнее прерывание 6
9	$0010	INT7	Запрос на внешнее прерывание 7
10	$0012	TIMER2 COMP	Срабатывание компаратора таймера/счетчика 2
11	$0014	TIMER2 OVF	Переполнение таймера/счетчика 2
12	$0016	TIMER1 CAPT	Захват таймером/счетчиком 1
13	$0018	TIMER1 COMPA	Срабатывание компаратора А таймера/счетчика 1
14	$001A	TIMER1 COMPB	Срабатывание компаратора В таймера/счетчика 1
15	$001C	TIMER1 OVF	Переполнение таймера/счетчика 1
16	$001E	TIMER0 COMP	Срабатывание компаратора таймера/счетчика 0
17	$0020	TIMER0 OVF	Переполнение таймера/счетчика 0
18	$0022	SPI, STC	Завершение последовательной передачи данных интерфейсом SPI
19	$0024	USART0, RX	Завершение приема USART 0
20	$0026	USART0, UDRE	Регистр данных USART 0 свободен
21	$0028	USART0, TX	Завершение передачи USART 0
22	$002A	ADC	Завершение преобразования АЦП

Рис. 7.1. Фрагмент таблицы векторов прерываний МК 1887ВЕ7Т (*ATmega128*) [8]

Например, прерыванию по событию «Завершение приема *USART0*» (т. е. по поступлении в регистр приемника *USART0* очередного слова данных) присвоен номер вектора 19, а начальная

команда подпрограммы обслуживания данного прерывания должна быть размещена по адресу *$0024* (т. е. *0x0024*).

Из представленного на рис. 7.1 фрагмента таблицы векторов прерываний можно заметить, что под начальные команды каждой из *ISR* выделено всего по два адреса (по два 16-битовых слова ПП). Естественно, никакую сколько-нибудь полноценную *ISR* в таком адресном пространстве разместить невозможно. Поэтому на практике по данным адресам размещается только команда безусловного перехода к собственно *ISR*, располагаемой в другом месте ПП. Ниже представлен фрагмент типового программного кода на языке ассемблера семейства *AVR* (см. табл. 2.7), располагаемого по адресам ПП, выделенным под начальные команды *ISR* МК 1887ВЕ7Т (*ATmega128*) [8] (см. также рис. 7.1).

```
Адрес   Команда      Операнд          Комментарий

$0000     jmp          RESET      ;Переход на обработку сброса
$0002     jmp        EXT_INT0     ;Переход на обработку запроса IRQ0
$0004     jmp        EXT_INT1     ;Переход на обработку запроса IRQ1
$0006     jmp        EXT_INT2     ;Переход на обработку запроса IRQ2
$0008     jmp        EXT_INT3     ;Переход на обработку запроса IRQ3
$000A     jmp        EXT_INT4     ;Переход на обработку запроса IRQ4
$000C     jmp        EXT_INT5     ;Переход на обработку запроса IRQ5
$000E     jmp        EXT_INT6     ;Переход на обработку запроса IRQ6
$0010     jmp        EXT_INT7     ;Переход на обработку запроса IRQ7
$0012     jmp        TIM2_COMP    ;Переход на обработку при срабатывании
                                  ;компаратора таймера 2
$0014     jmp        TIM2_OVF     ;Переход на обработку при переполнении
                                  ;таймера 2
$0016     jmp        TIM1_CAPT    ;Переход на обработку при захвате фронта
                                  ;таймером  1
$0018     jmp        TIM1_COMPA   ;Переход на обработку при срабатывании
                                  ;компаратора A таймера 1
$001A     jmp        TIM1_COMPB   ;Переход на обработку при срабатывании
                                  ;компаратора B таймера 1
```

Распознавание источника прерывания и формирование запроса на его обслуживание (*IRQ*), поступающего на ЦП, осуществляются **автоматически**, на **аппаратном** уровне (без участия ПО), специальным функциональным блоком, который на рис. 1.3 назван блоком прерываний, на рис. 1.4 – *NVIC*, а на рис. 1.5 – контроллером прерываний. По получении указанного запроса ЦП приостанавливает выполнение основной программы и также автоматически, без участия ПО переходит к подпрограмме обслуживания прерывания от соответствующего источника (более

подробное описание процедуры перехода приведено в пункте 7.2.2). При этом подпрограмма обслуживания прерывания разрабатывается как часть прикладного ПО МК, а адрес ее начальной команды должен соответствовать таблице векторов прерываний (см., например, рис. 7.1). Большинство современных IDE обеспечивают данное требование автоматически, при компиляции программы.

Следует отметить, что, кроме вышеописанной векторной, известна также система распознавания источников запросов на прерывания и их обслуживания, называемая **обзорной** или **опросной** [3] и применявшаяся на ранних этапах развития электронной вычислительной техники. Данная система характеризуется наличием одной общей подпрограммы обслуживания прерываний, запускаемой по поступлении запроса от любого из их источников, имеющихся в составе вычислительной системы. После запуска данной подпрограммы ею осуществляется проверка состояний сигналов запроса всех источников прерываний, с последующим переходом к процедуре обслуживания того из них, сигнал запроса от которого находится в активном состоянии. Если в нем находятся сигналы запроса от нескольких источников, их обслуживание осуществляется в соответствии с уровнем их приоритета (в 1-ю очередь – наиболее приоритетного).

Нетрудно увидеть, что обзорная система характеризуется значительно менее эффективным использованием процессорного времени, чем векторная. В настоящее время она практически не применяется в качестве основной (как альтернатива векторной системе). Однако, ее элементы используются и в архитектуре современных МК, как дополнение к векторной системе, если условием генерации некоторого запроса на прерывание от какого-либо блока МК может служить одно из нескольких событий. В таких ситуациях проверка, какое именно из них вызвало прерывание, реализуется в *ISR*, путем опроса битов регистра (регистров) состояния соответствующего функционального блока (см., например, рис. 5.2 и пояснения к нему, а также подпункты 7.3.2.16 и 7.3.2.22).

7.2.2. Приоритетное обслуживание прерываний. Способы назначения приоритетов

7.2.2.1. Приоритетное обслуживание прерываний является одним из базовых требований к подсистеме их обработки [3, 9]. Как видно из рис. 7.1, число источников прерываний даже у относительно простых МК равно нескольким десяткам. Поэтому, во избежание конфликтов при обслуживании запросов от них, должна быть установлена его очередность, что осуществляется присвоением каждому источнику запроса на прерывание определенного **приоритета**. Известно несколько способов назначения приоритетов (см. подпункт 7.2.2.2), но при любом из них **не существует** 2-х источников запросов на прерывание с одинаковым приоритетом.

В случае одновременного поступления запросов на прерывание от 2-х или нескольких источников, в первую очередь обслуживается прерывание с наивысшим приоритетом; остальные запросы фиксируются, «ставятся в очередь» и обслуживаются в порядке убывания их приоритета.

Различают **относительное** и **абсолютное** обслуживание прерываний [3]. Если любой запрос на прерывание, поступивший во время выполнения подпрограммы обслуживания ранее поступившего запроса, обрабатывается только после ее завершения, независимо от приоритета источника запроса, такой способ обслуживания прерываний называется **относительным**. Если же по поступлении запроса от более приоритетного источника выполнение подпрограммы обслуживания прерывания приостанавливается, и возобновляется только после обработки более приоритетного запроса (см. рис. 7.2), данный способ обслуживания прерываний называется **абсолютным**.

В частности, при применении МК по их основному назначению (контроль и управление техническими объектами) во многих случаях бывает желательно, чтобы во время выполнения подпрограммы обслуживания прерывания от некоторого источника ЦП не реагировал ни на какие другие запросы. Поэтому в ряде семейств МК (в частности, *AVR* [6, 8]), по умолчанию, применяется **относительное** обслуживание прерываний. При переходе к

подпрограмме обработки прерывания происходит автоматический сброс бита глобального разрешения прерываний; в семействе *AVR* им является бит *I* в регистре статуса (см. рис. 2.24). Восстановление активного состояния данного бита происходит, также автоматически, при выполнении команды возврата из подпрограммы обслуживания прерывания, *RETI* (см. табл. 2.7). Однако, если разработчик конкретного устройства на МК считает необходимым, чтобы ЦП при обслуживании прерываний от каких-либо источников реагировал на запросы от других источников, активное состояние бита глобального разрешения прерываний может быть восстановлено программно, командой его установки, располагаемой в начале подпрограммы обслуживания соответствующего прерывания.

7.2.2.2. Известны следующие основные способы назначения приоритетов запросов на прерывания [3, 9]:

- **статическая одноуровневая приоритезация**;
- **многоуровневая квазистатическая приоритезация** (пояснения см. далее);
- **динамическая приоритезация**, наиболее распространенным вариантом которой является **циклическая одноуровневая приоритезация** (см. далее);
- **комбинация** вышеперечисленных способов назначения приоритета.

Статическая одноуровневая приоритезация характерна для относительно несложных МК, с общим количеством источников прерываний, не превышающим 20 – 30. Например, на ней базируется обслуживание прерываний в МК подсемейств *ATmega* и *ATtiny* семейства *AVR* [6 – 8]. Ее принцип поясняет рис. 7.2.

Приоритет каждого из источников прерываний при статической одноуровневой приоритезации однозначно определяется номером вектора соответствующего прерывания (см. рис. 7.1): чем меньше номер вектора, тем выше приоритет. Например, из источников прерываний, представленных в приведенном на рис. 7.1 фрагменте таблицы векторов, наивысшим приоритетом обладает сброс (что естественно), наименьшим – прерывание, генерируемое по завершении цикла преобразования АЦП. При этом приоритет

прерывания по переполнении 2-го таймера выше, чем прерывания по переполнении 1-го таймера и т. п.

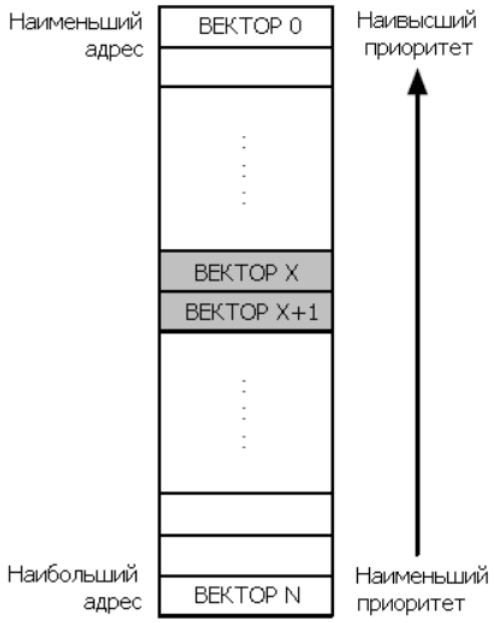


Рис. 7.2. Принцип статической одноуровневой приоритезации прерываний [43]

Номера векторов, а значит, и приоритеты задаются при разработке архитектуры МК, и не могут быть изменены. Разбиение источников прерываний на группы по уровням приоритета при **одноуровневой** приоритезации **отсутствует**. Обслуживание всех источников осуществляется в соответствии с единой «шкалой приоритетов».

Многоуровневая квазистатическая приоритезация применяется, в основном, в семействах МК класса «*high performance*», с числом источников прерываний более 30 – 40. При таком их количестве рациональным является их разбиение на группы по степени важности при решении конкретной задачи, т. е. назначение каждому из источников прерывания некоторого уровня важности (приоритетности), например, высокого, среднего или низкого. Любой из источников прерываний, включенный в группу с более высоким уровнем приоритетности, обладает б**о**льшим приоритетом, чем каждый из источников, отнесенных к любой из групп с меньшим уровнем приоритетности. В свою очередь, в

пределах каждой группы источников прерываний с приоритетностью одинакового уровня, приоритеты распределяются, например, в соответствии с номерами векторов, назначенными при разработке архитектуры МК: чем меньше номер вектора, тем выше приоритет источника прерываний в пределах группы. Возможны и более сложные варианты (см. подпункты 7.3.2.7 – 7.3.2.9). Как правило, распределение источников прерываний по группам осуществляется ПО МК, и, в принципе, может изменяться в процессе работы МК; соответственно, при этом изменяются и приоритеты запросов от этих источников в пределах подсистемы обслуживания прерываний в целом. Поэтому данный способ приоритезации корректнее назвать не статическим, а квазистатическим.

В качестве относительно простого примера использования многоуровневой квазистатической приоритезации можно привести подсистему обслуживания прерываний МК семейства *ATxmega* [44]. В одном из РСФ регистровой модели каждого из ПУ МК данного семейства имеется программно доступное 2-битовое поле, посредством которого назначается уровень приоритетности прерываний от соответствующего ПУ: 00 – прерывания запрещены; 01, 10 и 11 – соответственно низкий, средний и высокий уровень приоритетности. В свою очередь, в пределах группы источников прерываний с приоритетностью одинакового уровня, приоритеты распределяются в соответствии с таблицей векторов прерываний; ее фрагмент для МК подсемейства *XMEGA E* приведен на рис. 7.3. В пределах группы с низким уровнем приоритетности возможен также циклический вариант динамической приоритезации (см. далее).

Предположим, например, что прерываниям от порта *A* присвоен уровень приоритетности 11 («высокий»), а от портов *C* и *D* – 10 («средний»). При этом:

- приоритет прерываний от порта *A* будет выше, чем прерываний от портов *C* и *D*;

- в соответствии с таблицей, приведенной на рис. 7.3, приоритет прерываний от порта *C* выше, чем от порта *D*.

Аналогичный, но несколько более сложный способ приоритезации, с разбиением источников прерываний по степени приоритетности не только на группы, но и на подгруппы,

используется в МК семейства *ARM Cortex-Mx* (подробнее – см. пункт 7.3.2).

Program address (base address)	Source	Interrupt description
0x0000	RESET	
0x0002	OSCF_INT_vect	Crystal oscillator failure and PLL lock failure interrupt vector (NMI)
0x0004	PORTR_INT_vect	Port R Interrupt vector
0x0006	EDMA_INT_base	EDMA Controller Interrupt base
0x000E	RTC_INT_base	Real time counter interrupt base
0x0012	PORTC_INT_vect	Port C interrupt vector
0x0014	TWIC_INT_base	Two-wire interface on Port C interrupt base
0x0018	TCC4_INT_base	Timer/counter 4 on port C interrupt base
0x0024	TCC5_INT_base	Timer/counter 5 on port C interrupt base
0x002C	SPIC_INT_vect	SPI on port C interrupt vector
0x002E	USARTC0_INT_base	USART 0 on port C interrupt base
0x0034	NVM_INT_base	Non-Volatile Memory interrupt base
0x0038	XCL_INT_base	XCL (programmable logic) module interrupt base
0x003C	PORTA_INT_vect	Port A interrupt vector
0x003E	ACA_INT_base	Analog comparator on Port A interrupt base
0x0044	ADCA_INT_base	Analog to digital converter on Port A interrupt base
0x0046	PORTD_INT_vect	Port D interrupt vector
0x0048	TCD5_INT_base	Timer/counter 5 on port D interrupt base
0x0050	USARTD0_INT_base	USART 0 on port D interrupt base

Рис. 7.3. Фрагмент таблицы векторов прерываний МК подсемейства *XMEGA E* семейства *ATxmega* [45]

Динамическая приоритезация сравнительно мало распространена в МК общего назначения. Из ее возможных вариантов на практике обычно используется **циклическая приоритезация** (в англоязычных источниках называемая *Round Robin*). Она опционально может применяться, например, в МК семейства *ATxmega*, но только в пределах группы источников прерываний с уровнем приоритетности 01 («низким») [43]. При этом, как указано ранее, на высоком и среднем уровнях приоритетности используется только статическая / квазистатическая приоритезация. На низком уровне возможен выбор статической или циклической приоритезации, в зависимости от состояния бита *RREN*

(*Round Robin Enable*) в регистре управления контроллером прерываний. Принцип циклической приоритезации поясняется рис. 7.4 и состоит в следующем: по обслуживании прерывания от некоторого источника ему присваивается **наименьший** приоритет (образно говоря – тот, кого только что обслужили, пусть становится в конец очереди, другие тоже хотят). Выбор циклической приоритезации рационален при большом количестве источников прерываний в группе с низким уровнем приоритетности и высокой интенсивности генерации запросов на прерывания в пределах данной группы. В таких случаях при статической приоритезации могут возникнуть ситуации, когда запросы от источников с б**о**льшим (в пределах группы) приоритетом обрабатываются часто, а до других, с меньшим приоритетом, очередь никогда не дойдет. Циклическая приоритезация позволяет избежать подобных ситуаций, благодаря присвоению наименьшего приоритета источнику последнего обслуженного прерывания (см. рис. 7.4).

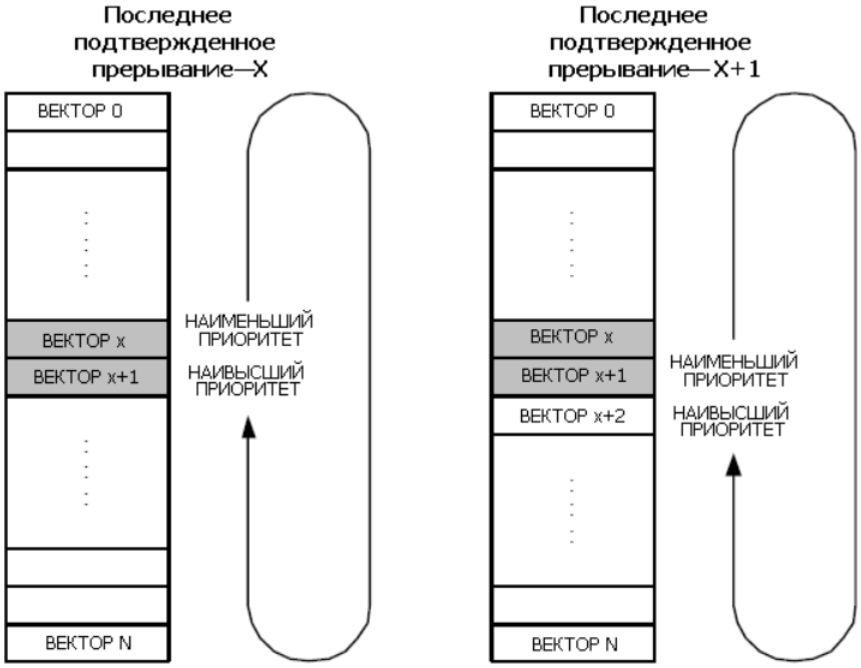


Рис. 7.4. Принцип циклической одноуровневой приоритезации прерываний [43]

Следует отметить, что в МК подсемейств *ATtiny* и *ATmega* циклическая приоритезация не применяется, ввиду относительно небольшого количества источников прерываний (тем более –

используемых при решении конкретных задач). Также не применяется данный способ приоритезации в МК семейства *ARM Cortex-Mx*, благодаря возможности избежать вышеописанных ситуаций за счет рационального распределения источников прерываний на группы и подгруппы по уровню приоритетности (см. пункт 7.3.2).

Интересно отметить, что циклическая приоритезация опционально использовалась в подсистеме прерываний ранних поколений ПК семейства *IBM PC*.

7.2.3. Обеспечение возврата в основную программу по выполнении подпрограммы обслуживания прерывания. Реализация вложенных прерываний

Как указано в основных требованиях к подсистеме прерываний (см. начало текущего подраздела), по выполнении подпрограммы обслуживания запроса на прерывание должно быть обеспечено **возобновление** работы основной программы с того адреса, на котором произошла приостановка ее выполнения, вызванная прерыванием. Для этого используется механизм сохранения необходимых для возврата данных (как минимум – адреса возврата) в **стеке**, с извлечением их из стека по завершении обслуживания запроса на прерывание [3], аналогично процедурам перехода к выполнению подпрограмм и возврата из них, описываемым рис. 2.2. Тот же принцип используется при обслуживании **вложенных** прерываний, т. е. при получении запроса от источника с более высоким приоритетом во время обслуживания прерывания с более низким приоритетом.

Вышесказанное иллюстрирует рис. 7.5. На нем представлена схема обслуживания запроса на прерывание $IRQn$ от некоторого n-го источника, во время выполнения подпрограммы обработки которого поступает запрос $IRQk$ от k-го источника с более высоким приоритетом. Полагается, что оба запроса не **маскированы** (см. пункт 7.2.4) и используется **абсолютный** способ обслуживания прерываний (см. подпункт 7.2.2.1).

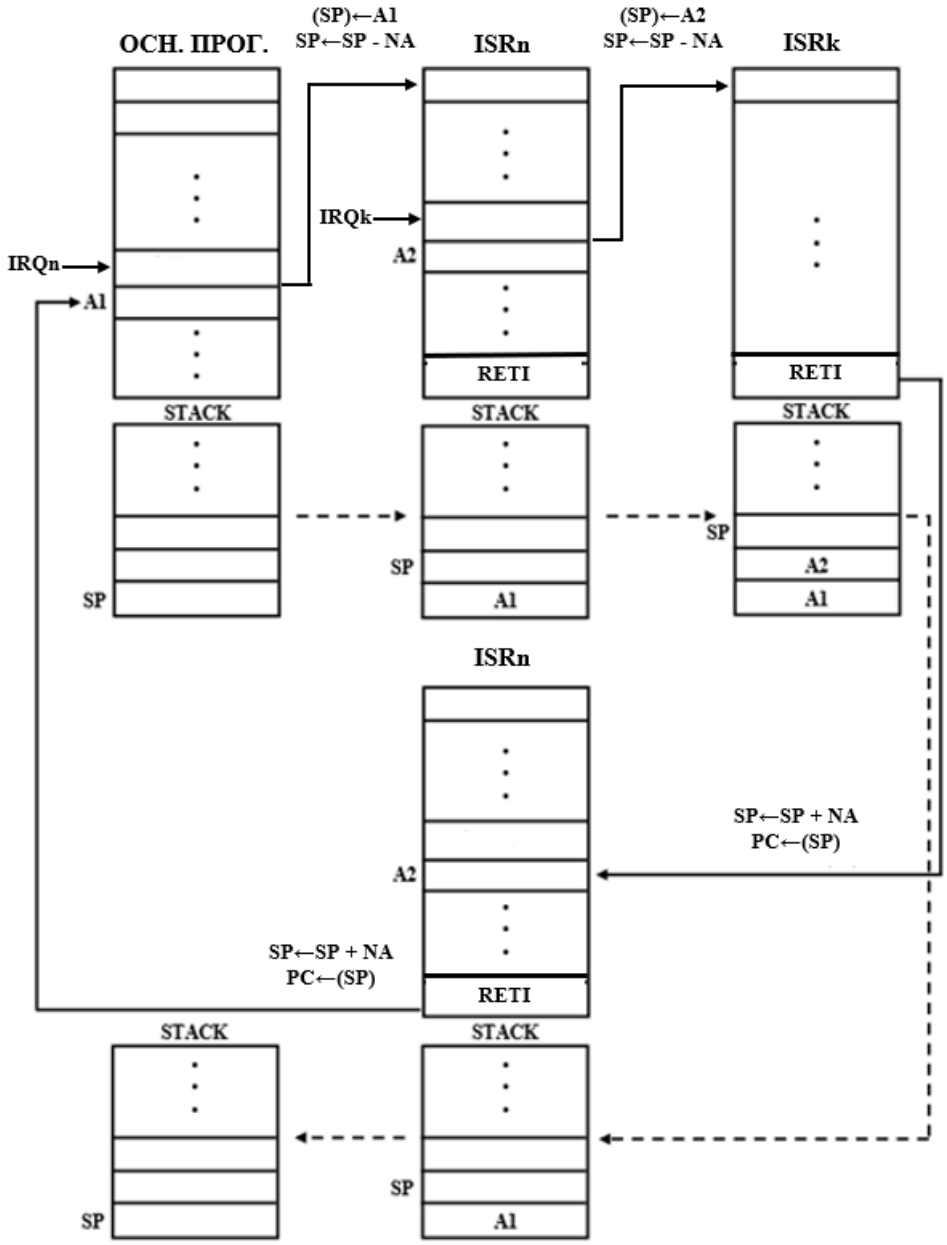


Рис. 7.5. Схема обслуживания запросов на прерывание (см. пояснения в тексте)

Как и на рис. 2.2, *STACK* – содержимое стека на соответствующем этапе вычислительного процесса; сплошными стрелками показан ход выполнения программы, а пунктирными – порядок изменения содержимого стека. Предполагается, что стек заполняется, начиная с максимального из адресов области памяти, выделенной под него. Также предполагается, что при переходе к подпрограмме обслуживания прерывания в стеке автоматически сохраняется только адрес возврата, что имеет место в большинстве семейств МК классов «*cost-effective*» и «*mainstream*», в т. ч. *AVR* (в

ряде семейств класса «*high performance*» сохраняется также содержимое регистра статуса и ряда других регистров).

По поступлении запроса $IRQn$ и завершении выполнения команды, во время которого он поступил, выполняются следующие действия (см. рис. 7.5):

- выполнение основной программы приостанавливается;

- в стек помещается адрес *A1* команды, с которой необходимо возобновить выполнение основной программы по завершении обслуживания запроса $IRQn$;

- затем происходит переход к подпрограмме обслуживания указанного запроса, обозначенной на рис. 7.5 как $ISRn$.

При этом все перечисленные операции выполняются **автоматически**, на аппаратном уровне, без участия ПО.

Предположим теперь, что во время выполнения подпрограммы обслуживания запроса $IRQn$ поступил запрос на прерывание $IRQk$ от k-го источника, с более высоким приоритетом, чем у n-го. При этом (см. рис. 7.5):

- приостанавливается выполнение подпрограммы $ISRn$;

- в стек, «поверх» адреса *A1* помещается адрес *A2* команды, с которой необходимо возобновить выполнение подпрограммы $ISRn$ по завершении обслуживания запроса $IRQk$;

- затем происходит переход к подпрограмме обслуживания указанного запроса от k-го источника, обозначенной на рис. 7.2 как $ISRk$.

Каждая из подпрограмм обслуживания запросов на прерывания (и $ISRk$, и $ISRn$) должна заканчиваться специальной командой возврата из прерывания. На рис. 7.5 она условно обозначена как *RETI*, таков же ее синтаксис в системе команд МК семейства *AVR* (см. табл. 2.7); в системах команд других семейств он может быть иным. В большинстве *IDE* при разработке ПО МК нет необходимости вводить данную команду в программный код на языке *C*, она автоматически включается в каждую из подпрограмм обслуживания прерываний при компиляции.

По окончании выполнения подпрограммы обслуживания более приоритетного прерывания ($ISRk$), по завершающей ее команде *RETI* в программный счетчик с вершины стека загружается адрес *A2*

возврата в подпрограмму $ISRn$. В свою очередь, *после* окончания ее выполнения, по завершающей ее команде $RETI$ в программный счетчик с вершины стека загружается адрес $A1$ возврата в основную программу, и возобновляется ее выполнение, приостановленное прерываниями (см. рис. 7.5).

В общем случае, если во время выполнения подпрограммы $ISRk$ обслуживания прерывания от k-го источника поступит не маскированный запрос от некоторого i-го источника с более высоким приоритетом, произойдет переход к подпрограмме $ISRi$ его обслуживания (с приостановкой выполнения $ISRk$ и сохранением в стеке адреса возврата в нее). По выполнении подпрограммы $ISRi$ данный адрес будет загружен в программный счетчик, осуществится возврат к $ISRk$ и ее завершение. после чего аналогичным образом возобновится выполнение подпрограммы $ISRn$ и, по его окончании – возврат в основную программу.

Глубиной вложенности прерываний (или **глубиной прерываний**) [3] называют максимальное число запросов на прерывания с возрастающим от запроса к запросу уровнем приоритета, которые могут быть обслужены от приостановки выполнения основной программы по поступлении первого из них и до возврата в нее по обслуживании всех запросов. При **относительном** обслуживании прерываний (см. подпункт 7.2.2.1) глубина вложенности прерываний, очевидно, равна 1. При **абсолютном** обслуживании прерываний она теоретически ограничена только емкостью (глубиной) стека, определяющей максимальное число адресов возврата, которое может в нем храниться (см. рис.7.5). На практике, однако, могут существовать дополнительные ограничения на глубину вложенности прерываний, зависящие от архитектуры конкретного семейства МК.

Следует также отметить, что архитектурой многих семейств МК не предусмотрено автоматическое сохранение никаких данных при переходе к ISR, кроме адреса возврата. При этом не подлежащие потере данные (например, содержимое каких-либо РОН или РСФ и т. п.) могут быть сохранены включением в ISR программного модуля их записи в определенную область ПД перед выполнением основной

части *ISR*, а также программного модуля их выгрузки из ПД по выполнении основной части *ISR* (см. пример в подпункте 7.3.1.5).

Необходимо в заключение отметить, что важными параметрами подсистемы обслуживания прерываний МК (особенно при решении задач, время выполнения которых критично) являются **время отклика** на запрос прерывания и **время возврата** из подпрограммы его обслуживания. **Первое** из них, по определению, равно длительности интервала времени от поступления запроса на прерывание до начала выполнения подпрограммы его обслуживания. **Второй** из вышеназванных параметров фактически представляет собой время выполнения команды возврата из подпрограммы обслуживания прерывания. Время отклика и время возврата выражаются в периодах синхросигнала ЦП, и указываются в технической документации на соответствующее семейство / подсемейство МК (см. подпункты 7.3.1.8 и 7.3.2.24).

7.2.4. Разрешение и запрет прерываний

7.2.4.1. Возможность индивидуального программного разрешения или запрета (**маскирования**) каждого из прерываний, кроме так называемых немаскируемых прерываний (см. далее) является еще одним базовым требованием к подсистеме обслуживания прерываний [3]. Архитектурой большинства семейств МК обеспечивается возможность [6, 9]:

- глобального разрешения / запрета прерываний, кроме так называемых немаскируемых (см. далее);
- выборочного разрешения прерываний по событиям, мониторинг которых необходим в процессе работы МК в составе проектируемого устройства / системы;
- временного запрета (маскирования) прерываний по событиям, на которые ЦП не должен реагировать при определенных состояниях объекта контроля и управления (например, на открывание двери автомобиля при выключенной охранной сигнализации), с последующим разрешением прерываний при выходе объекта из соответствующего состояния (например, по включении сигнализации).

7.2.4.2. Архитектура всех семейств МК предусматривает наличие так называемых **немаскируемых** прерываний (*Non-Maskable Interrupts, NMI*), которые не могут быть запрещены программно. Как минимум одно немаскируемое прерывание, **сброс** (*Reset*) присутствует во всех семействах всех классов МК, даже в модельных рядах, в которых подсистема обслуживания прерываний от ПУ отсутствует (например, в *PIC*-МК младшего подсемейства). Из прерываний МК семейства *AVR* немаскируемым также является только сброс; прерывание по нему обслуживается и при нулевом состоянии бита глобального разрешения прерываний. В МК семейства *ARM Cortex-Mx* существует несколько источников немаскируемых прерываний (подробнее – см. подпункт 7.3.2.3).

Следует отметить, что «по умолчанию» (после сброса МК, в т. ч. по включении питания), как правило, бывают запрещены (**замаскированы**) все прерывания, кроме немаскируемых [6, 9].

7.4.2.3. Глобальное разрешение / запрет прерываний осуществляется установкой / сбросом определенного бита в одном из РСФ. Например, в МК семейства *AVR* данная функция реализуется установкой / сбросом бита глобального разрешения прерываний (*I*) в регистре статуса (см. рис. 2.24). Состояние данного бита «по умолчанию», например, после сброса – нулевое, т. е. все прерывания, кроме немаскируемых, запрещены. Возможность глобального разрешения / запрета прерываний имеется также в МК семейства *ARM Cortex-Mx* (см. подпункт 7.3.2.19).

7.4.2.4. Для **выборочного** разрешения / запрета прерываний по отдельным событиям, в определенном РСФ каждого из ПУ выделяются специальные доступные для записи биты разрешения / запрета прерывания по каждому из событий, которое может быть источником прерывания от данного ПУ. Разрешение осуществляется установкой соответствующего бита в единицу, запрет – его сбросом в ноль.

В качестве примера на рис. 7.6 приведен формат регистра маскирования прерываний таймера / счетчика 3 МК 1887ВЕ7Т (*ATmega128*); в регистровой модели блока таймеров данного МК ему присвоено имя *ETIMSK* [8]. Заметим, что начальное состояние всех

его битов при сбросе, в т. ч. по включении питания – нулевое, т. е. по умолчанию все прерывания запрещены (см. выше).

Например, разрешение прерывания по переполнении таймера / счетчика 3 осуществляется установкой в единицу 2-го бита (*TOIE3*) регистра маскирования, запрет прерывания по данному событию – сбросом бита *TOIE3* в ноль.

Бит	7	6	5	4	3	2	1	0
	-	-	TICIE3	OCIE3A	OCIE3B	TOIE3	OCIE3C	OCIE1C
Чтение/ Запись	Ч	Ч	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З
Начальное значение	0	0	0	0	0	0	0	0

Рис. 7.6. Формат регистра маскирования прерываний таймера / счетчика 3 (*ETIMSK*) МК 1887ВЕ7Т (*ATmega128*) [8]

В МК семейства *AVR* для разрешения прерываний по какому-либо событию необходимо установить в единицу как бит маскирования прерываний по соответствующему событию, так и бит *I* глобального разрешения прерываний, например:

```
/*
Установка в единицу бита разрешения прерываний по переполнении таймера / счетчика 3 (см. рис. 7.6)
*/
ETIMSK = ETIMSK|0b00000100;

//Установка в единицу бита глобального разрешения прерываний
sei();
```

Аналогично реализуется индивидуальное разрешение и запрет прерываний по отдельным событиям практически во всех современных семействах МК.

7.2.4.5. Важно отметить, что архитектура практически всех семейств МК позволяет контролировать, имело ли место какое-либо из событий, которое могло бы вызвать прерывание, если собственно прерывание по данному событию запрещено. Для этого в одном из регистров каждого из блоков, являющихся источниками прерываний, выделяются специальные биты признаков (флагов) всех событий, вызывающих прерывания от соответствующего блока.

Установка флага в активное, как правило, единичное состояние служит признаком соответствующего события (за некоторыми исключениями, см., например, подпункт 7.3.1.7). В качестве примера на рис. 7.7 представлен формат регистра флагов прерываний таймера / счетчика 3 МК 1887ВЕ7Т (*ATmega128*) [8]. Сравнивая форматы регистров маскирования и флагов прерываний (см. рис. 7.6 и 7.7), нетрудно заметить, что для каждого из событий, служащих источниками прерываний от таймера / счетчика 3, в регистре флагов выделен специальный бит (флаг), активное (единичное) состояние которого служит признаком наступления соответствующего события. Например, при переполнении 3-го таймера / счетчика автоматически устанавливается в единицу бит *TOV3*.

Бит	7	6	5	4	3	2	1	0
	-	-	ICF3	OCF3A	OCF3B	TOV3	OCF3C	OCF1C
Чтение/ Запись	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З	Ч/З
Начальное значение	0	0	0	0	0	0	0	0

Рис. 7.7. Формат регистра флагов прерываний таймера / счетчика 3 МК 1887ВЕ7Т (*ATmega128*) [8]

В МК семейства *AVR*, если разрешено прерывание по некоторому событию, бит его признака автоматически сбрасывается при переходе к подпрограмме обслуживания прерывания. Если прерывание запрещено – бит сбрасывается программно, записью **единицы** в соответствующий разряд регистра флагов. В МК семейства *ARM Cortex-Mx* бит признака события **не сбрасывается** при переходе к подпрограмме прерывания по соответствующему событию; сброс данного бита осуществляется записью в него **нуля** или **единицы** (подробнее – см. подпункт 7.3.2.16).

Если ПУ вырабатывает запрос на прерывание только по одному – двум событиям, под биты разрешения прерываний и под флаги прерываний обычно не выделяются специальные регистры; данные биты располагаются в одном из регистров управления и статуса ПУ. Например, у МК 1887ВЕ7Т (*ATmega128*) бит разрешения прерывания по окончании цикла преобразования встроенного АЦП (*ADIE*) и бит признака окончания цикла преобразования (*ADIF*)

располагаются в регистре *A* управления и состояния АЦП (*ADCSRA*), являясь его 3-м и 4-м битами соответственно.

По состоянию бита признака наступления того или иного события ПО может осуществлять его мониторинг, если механизм прерываний по каким-либо причинам не применяется. Ниже представлен программный фрагмент запуска цикла преобразования АЦП МК *ATmega128* и считывания результата преобразования по его завершении, определяемому опросом бита признака данного события – бита *ADIF* регистра *ADCSRA* (см. выше).

```
/*
Запуск цикла преобразования установкой в единицу 6-го бита (AD Start Conversion, ADSC) регистра ADCSRA
*/
ADCSRA = ADCSRA|0b01000000;

/*
Ожидание, пока 4-й бит (ADIF) регистра ADCSRA не установится в единицу, что является признаком окончания цикла преобразования и готовности его результата для считывания
*/
while((ADCSRA&0b00010000)==0);

/*
Сброс бита ADIF записью в него единицы
*/
ADCSRA = ADCSRA|0b00010000;

/*
Считывание старших 8-и битов результата преобразования и присвоение их значения переменной x
*/
x = ADCH;
```

7.3 Типовые примеры архитектурных решений и программирования подсистемы обслуживания прерываний

В качестве типовых примеров рассмотрим архитектуру подсистемы обслуживания прерываний МК семейств *AVR* и *ARM Cortex-Mx*.

7.3.1. Подсистема обслуживания прерываний МК семейства *AVR*

Базовые архитектурные решения подсистемы обслуживания прерываний МК семейства *AVR* типичны для относительно простых МК класса «*mainstream*». В целом, они рассмотрены в разделе 7.2 в качестве типовых примеров архитектурных решений подсистемы обслуживания прерываний в общем. В данном пункте необходимо только их систематизировать и дополнить рядом деталей [8].

7.3.1.1. Подсистема обслуживания прерываний МК семейства *AVR*, как и практически всех современных семейств МК, использует **векторный** принцип обслуживания (см. пункт 7.2.1). Пример таблицы векторов прерываний МК данного семейства приведен на рис. 7.1.

Запрос на прерывание обслуживается только **по выполнении** ЦП текущей команды.

Необходимо отметить, что у большинства МК подсемейства ATmega начальные адреса подпрограмм обслуживания прерываний могут быть перемещены из области, указанной в таблице векторов прерываний (см., например, рис. 7.1) в начало загрузочного сектора. После перемещения, начальные команды подпрограмм обслуживания прерываний должны размещаться по адресам, равным сумме начального адреса загрузочного сектора и указанного в таблице векторов прерываний (см. рис. 7.1) адреса, с которого должна начинаться подпрограмма обслуживания соответствующего прерывания в отсутствие перемещения.

Процедура перемещения реализуется программно. Ее порядок и примеры программных кодов ее реализации представляются в технических описаниях (datasheet) МК (см., например, [8]).

7.3.1.2. В подсистеме обслуживания прерываний МК семейства *AVR* применена **одноуровневая статическая** приоритезация (см. пункт 7.2.2, в т. ч. рис. 7.2). Разбиение источников прерываний на группы по уровням приоритета отсутствует. Приоритет прерывания однозначно определяется номером его вектора, и тем выше, чем меньше данный номер. Приоритеты не могут изменяться ПО МК.

7.3.1.3. По умолчанию, в МК семейства *AVR* применяется **относительное** обслуживание запросов на прерывание (см. пункт 7.2.3). По переходу к подпрограмме обслуживания прерываний бит глобального разрешения прерываний (бит *I* регистра статуса, см. рис. 2.24) сбрасывается, и во время ее выполнения ЦП не реагирует ни на какие другие запросы на прерывание, кроме сброса. Восстановление активного состояния указанного бита происходит при выполнении команды возврата из подпрограммы обслуживания прерывания, *RETI*. Однако, при необходимости, активное состояние бита глобального разрешения прерываний может быть восстановлено командой, располагаемой в начале подпрограммы обслуживания прерывания.

МК семейства *AVR* поддерживают **очередь** запросов на прерывания. Если вызывающие их события происходят при сброшенном бите глобального разрешения прерываний, признаки возникновения данных событий (см. подпункт 7.3.1.7) остаются в активном состоянии до установки указанного бита в единичное состояние. После этого запросы обслуживаются в порядке их приоритета (если прерывания по ним разрешены).

7.3.1.4. В МК семейства *AVR* при переходе к подпрограмме обслуживания прерывания в стеке сохраняется только адрес возврата, выгружаемый из стека при выполнении команды *RETI*. Если имеется необходимость сохранения других данных (например, содержимого регистра статуса, каких-либо РОН и т. п.), оно должно быть предусмотрено при разработке подпрограммы обслуживания прерывания и осуществляться непосредственно при входе в нее; извлечение сохраненных данных из стека при этом должно выполняться перед выходом из подпрограммы (см. подпункт 7.3.1.5).

7.3.1.5. Ниже представлен простой пример оформления подпрограммы обслуживания прерывания в *IDE Atmel Studio 7*, с сохранением в стеке содержимых регистра статуса и РОН *R16* при входе в подпрограмму и их восстановлением перед возвратом из нее в основную программу. Модель МК – *ATmega128*. Событием, вызывающим прерывание, в данном примере является завершение передачи блоком *USART0* (см. рис. 7.1). Сохранение и

восстановление содержимых регистра статуса и РОН *R16* оформлены в виде ассемблерных вставок (см. табл. 2.7).

```
/*
Пример подпрограммы обслуживания прерывания по завершении передачи блоком USART0 МК ATmega128
*/
ISR (USART0_TX_vect)
{
        //Сохранение содержимых регистра статуса и РОН R16 в стеке
         asm volatile(
        //Загрузка содержимого РОН R16 в стек
        "push r16\n\t"

        /*
        Загрузка в РОН R16 содержимого РСФ с адресом 0x5f, т. е. регистра статуса
        */
        "lds r16,0x5f\n\t"
        //Загрузка содержимого РОН R16 в стек
        "push r16\n\t"
        );
        //Основной код подпрограммы
        ……
        //Восстановление содержимых регистра статуса и РОН R16
        asm volatile(
        /*
        Извлечение сохраненного содержимого регистра статуса из стека в РОН R16
        */
        “pop r16\n\t”
        //Пересылка содержимого РОН R16 в регистр статуса
        "sts 0x5f,r16\n\t"
        //Извлечение из стека сохраненного содержимого РОН R16
        "pop r16\n\t"
        );
}
```

Заголовок подпрограммы оформляется в соответствии с «подсказками» *IDE*, и должен содержать идентификатор источника прерывания в соответствии с таблицей векторов (см. рис. 7.1).

Размещение подпрограммы в ПП по адресам, выделенным для процедуры обслуживания запроса на прерывание по данному событию, происходит автоматически, при компиляции кода, на основании параметров заголовка подпрограммы. Заметим, что в представленном коде отсутствует команда возврата из прерывания (*RETI*); она также добавляется при компиляции.

Необходимо также отметить, что корректная компиляция и работа вышеприведенной подпрограммы возможны только при включении в заголовок содержащего ее программного модуля следующих директив:

```
//Подключение библиотеки моделей ПУ МК семейства AVR
#include <avr/io.h>
/*
Подключение библиотеки стандартных функций обработки прерываний МК семейства AVR
*/
#include <avr/interrupt.h>
```

7.3.1.6. В МК семейства *AVR* все прерывания, за исключением сброса, являются **маскируемыми**, причем существует возможность как индивидуального программного разрешения или запрета каждого из прерываний, кроме сброса, так и запрета всех прерываний, кроме немаскируемых. В определенном РСФ каждого из ПУ МК выделены специальные биты разрешения / запрета прерывания по каждому из событий, которое может быть источником прерывания. Разрешение осуществляется установкой соответствующего бита в единицу, запрет – его сбросом в ноль. Глобальное разрешение / запрет осуществляются установкой / сбросом бита *I* в регистре статуса (см. рис.2.24). Для разрешения прерываний по какому-либо событию необходимо установить в единицу как бит маскирования прерываний по соответствующему событию, так и бит *I* глобального разрешения прерываний (см. пример в пункте 7.2.4).

7.3.1.7. В МК семейства *AVR*, как и во всех современных семействах МК, при возникновении каждого из событий, которое может вызвать прерывание (за исключением внешних прерываний

по уровню сигнала [8]), устанавливается в единицу бит (флаг) признака соответствующего события в некотором РСФ. По состоянию бита признака наступления того или иного события ПО может осуществлять его мониторинг, если механизм прерываний по каким-либо причинам не применяется (см. программный фрагмент запуска цикла преобразования АЦП МК *ATmega128* и считывания результата преобразования по его завершении, приведенный в пункте 7.2.4). Флаги признаков событий сбрасываются при переходе к подпрограмме обслуживания прерывания; также они могут быть сброшены записью **единицы** в соответствующий бит признака.

7.3.1.8. Минимальные значения **времени отклика** на запрос прерывания и **времени возврата** из подпрограммы обслуживания прерывания в МК семейства *AVR* равны 4-м периодам синхросигнала МК [6, 8]. При этом необходимо отметить, что подпрограмма обслуживания прерывания практически всегда начинается с команды безусловного перехода к области реального расположения подпрограммы (см. пункт 7.2.1). Время выполнения данной команды равно 3-м периодам синхросигнала МК [8]. Таким образом, фактическое значение времени отклика на запрос прерывания – не менее 7-ми тактов.

7.3.2. Подсистема обслуживания прерываний МК семейства *ARM Cortex-Mx*

Подсистема обслуживания прерываний МК семейства *ARM Cortex-Mx*, в целом, реализована по общим принципам, описанным в подразделе 7.2, но несколько сложнее, чем, например, у семейства *AVR*, ввиду большего количества источников прерываний и более развитой архитектуры семейства *ARM Cortex-Mx*. Рассмотрим основные характеристики и особенности данной подсистемы; большинство применяемых в ней архитектурных решений используется и в других семействах МК класса «*high performance*».

В документации на МК семейства *ARM Cortex-Mx* прерывания называются **исключениями** (*Exceptions*). В свою очередь, к **собственно прерываниям** (*Interrupts*) в данном семействе МК относят исключения, источниками которых являются резидентные

ПУ МК, а также внешние по отношению к нему устройства (см. рис. 7.8). В дальнейшем, во избежание терминологической путаницы, будем использовать термин «прерывание» для всех видов исключений.

7.3.2.1. Источниками прерываний в МК семейства *ARM Cortex-Mx* являются [9]:

- запросы на прерывания (*IRQ*), формируемые встроенными ПУ МК, а также внешними по отношению к нему устройствами;
- системные исключения, генерируемые ЦП;
- запросы на прерывание от системного таймера;
- сообщения о сбоях или отказах генератора *HSE*, формируемые блоком *CSS* (см. подпункт 4.4.2.6) и вызывающие немаскируемое прерывание (*NMI*);
- сброс МК.

7.3.2.2. Подсистема обслуживания прерываний МК семейства *ARM Cortex-Mx*, как и всех современных семейств МК, использует **векторный** принцип обслуживания (см. пункт 7.2.1). В качестве примера на рис. 7.8 представлены начало и окончание таблицы векторов прерываний МК модельного ряда *STM32F10xxx* [13] подсемейства *ARM Cortex-M3*. Полностью приводить здесь таблицу векторов прерываний представляется излишним, т. к. представленных на рис. 7.8 фрагментов достаточно для пояснения принципа ее организации и описания состава источников прерываний.

Атрибут «*fixed*» в столбце «*Type of priority*» означает, что приоритет соответствующего прерывания является фиксированным и не может быть изменен, а атрибут «*settable*» - что приоритет может быть задан разработчиком, в зависимости от специфики решаемой задачи (см. подпункты 7.3.2.7 – 7.3.2.14). В столбце «*Priority*» для каждого прерывания указано значение приоритета, присвоенные «по умолчанию»; чем оно меньше, тем выше приоритет. В столбце «*Address*» указан начальный адрес подпрограммы обслуживания соответствующего прерывания.

Прерываниям, инициируемым встроенными ПУ МК, а также внешними по отношению к нему устройствами, соответствуют строки таблицы векторов с отсутствием цветовой заливки.

Position	Priority	Type of priority	Acronym	Description	Address
-	-	-	-	Reserved	0x0000_0000
-	-3	fixed	Reset	Reset	0x0000_0004
-	-2	fixed	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.	0x0000_0008
-	-1	fixed	HardFault	All class of fault	0x0000_000C
-	0	settable	MemManage	Memory management	0x0000_0010
-	1	settable	BusFault	Prefetch fault, memory access fault	0x0000_0014
-	2	settable	UsageFault	Undefined instruction or illegal state	0x0000_0018
-	-	-	-	Reserved	0x0000_001C - 0x0000_002B
-	3	settable	SVCall	System service call via SWI instruction	0x0000_002C
-	4	settable	Debug Monitor	Debug Monitor	0x0000_0030
-	-	-	-	Reserved	0x0000_0034
-	5	settable	PendSV	Pendable request for system service	0x0000_0038
-	6	settable	SysTick	System tick timer	0x0000_003C
0	7	settable	WWDG	Window watchdog interrupt	0x0000_0040
1	8	settable	PVD	PVD through EXTI Line detection interrupt	0x0000_0044
2	9	settable	TAMPER	Tamper interrupt	0x0000_0048
3	10	settable	RTC	RTC global interrupt	0x0000_004C
4	11	settable	FLASH	Flash global interrupt	0x0000_0050
5	12	settable	RCC	RCC global interrupt	0x0000_0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000_0064
...					
47	54	settable	ADC3	ADC3 global interrupt	0x0000_00FC
48	55	settable	FSMC	FSMC global interrupt	0x0000_0100
49	56	settable	SDIO	SDIO global interrupt	0x0000_0104
50	57	settable	TIM5	TIM5 global interrupt	0x0000_0108
51	58	settable	SPI3	SPI3 global interrupt	0x0000_010C
52	59	settable	UART4	UART4 global interrupt	0x0000_0110
53	60	settable	UART5	UART5 global interrupt	0x0000_0114
54	61	settable	TIM6	TIM6 global interrupt	0x0000_0118
55	62	settable	TIM7	TIM7 global interrupt	0x0000_011C
56	63	settable	DMA2_Channel1	DMA2 Channel1 global interrupt	0x0000_0120
57	64	settable	DMA2_Channel2	DMA2 Channel2 global interrupt	0x0000_0124
58	65	settable	DMA2_Channel3	DMA2 Channel3 global interrupt	0x0000_0128
59	66	settable	DMA2_Channel4_5	DMA2 Channel4 and DMA2 Channel5 global interrupts	0x0000_012C

Рис. 7.8. Начало и окончание таблицы векторов прерываний МК модельного ряда *STM32F10xxx* [13]
(подробности см. в тексте)

7.3.2.3. Три прерывания с фиксированным (и наивысшим) приоритетом реализуются практически во всех подсемействах / модельных рядах МК семейства *ARM Cortex-Mx*. К ним относятся (см. рис. 7.8, а также [9]):

- *Reset* (Сброс); обладает наивысшим приоритетом; не может быть запрещен (замаскирован);

- *NMI* (*Non-Maskable Interrupt*, немаскируемое прерывание с наивысшим после сброса приоритетом), основным источником которого в МК семейства *ARM Cortex-Mx* является запрос на прерывание, генерируемый блоком *CSS* при сбое или отказе генератора *HSE* (см. подпункт 4.4.2.6);

- *Hard Fault* («тяжелый» отказ); возникает при ошибке во время обработки прерывания по отказу (системному исключению) или если такое прерывание не может быть обслужено каким-либо другим способом; например, обращение к недопустимой ячейке памяти вызывает прерывание *Hard Fault*, если прерывание по сбою шины (*Bus Fault*) запрещено.

Приоритет прерывания *Hard Fault* фиксированный, он ниже, чем у сброса и *NMI*, но выше, чем у любого из прерываний с конфигурируемым приоритетом.

7.3.2.4. Также к генерируемым ЦП системным исключениям относятся (см. рис. 7.8, а также [9]):

- *Memory Management Fault* (сбой системы управления памятью); возникает при нарушении правил доступа к некоторой области памяти (см. подпункт 2.5.8.10);

- *Bus Fault* (сбой шины); данное исключение генерируется, если возникает ошибка при выборке команды или данных по шине доступа к ПП или, соответственно, к ПД;

- *Usage Fault* (программная ошибка); имеет место, например, при попытке выполнения «не известной» ЦП команды; при обращении к слову или полуслову (см. подпункт 2.5.8.4) по некорректному адресу или при некорректном значении операнда (к примеру, при попытке деления на ноль);

- *SVCCall* (вызов супервизора); инициируется командой *SVC* (*Supervisor Call*) и используется в операционных системах (ОС) реального времени для получения доступа к ядру ОС и к драйверам устройств;

- *Debug Monitor* (прерывание от монитора отладки); генерируется при работе ЦП в отладочном режиме, если происходят определенные события процесса отладки;

- *PendSV* (*Pendable Request for System Service*, отложенный запрос системной службы); отличается от *SVCCall* (см. выше) тем, что может быть отложено при выполнении ОС более приоритетных задач, в то время как обслуживание исключения *SVCCall* осуществляется сразу после выполнения команды *SVC*.

Необходимо отметить, что в МК подсемейств *ARM Cortex-M0/0+* генерация системных исключений *Memory Management Fault, Bus Fault, Usage Fault* и *Debug Monitor* не реализована.

7.3.2.5. Запросы на прерывание от системного таймера (*SysTick*) могут быть использованы при решении задач, требующих «привязки» некоторых действий к определенным моментам времени. В принципе, для этого может быть применен любой из таймеров, входящих в состав МК. Однако, в составе ядра практически всех МК семейства *ARM Cortex-Mx* имеется специально предназначенный для подобных целей системный таймер, называемый *SysTick timer* (*STK*).

Основой *STK* является 24-битовый вычитающий счетчик с автоматической перезагрузкой содержимым специального, программно-доступного регистра перезагрузки по достижении счетчиком нулевого значения. Тактирование *STK*, по умолчанию, осуществляется синхросигналом с частотой, равной 1/8 частоты системного сигнала синхронизации МК (см., например, рис. 4.1), с опциональной возможностью программного выбора внешнего опорного синхросигнала (например, в МК модельного ряда *STM32F0*) или непосредственно системного синхросигнала (в большинстве модельных рядов подсемейств *ARM Cortex-M3* и *ARM Cortex-M4*) в качестве источника тактирования.

Работа *STK*, а также генерация им запросов на прерывания могут быть программно разрешены или запрещены. Если они разрешены, *STK* вырабатывает запросы на прерывание по каждом достижении его содержимым нулевого значения, тем самым «сообщая» ЦП об истечении очередного интервала времени длительностью $(RELOAD + 1)/f_{STK}$, где $RELOAD$ – значение содержимого регистра перезагрузки, f_{STK} – частота тактовых импульсов *STK*.

Конфигурирование *STK* (в т. ч. разрешение / запрет его работы) осуществляется РСФ, программно-доступными только на привилегированном уровне.

Более подробное описание архитектуры *STK* представлено в источниках [16], [18] и [19].

7.3.2.6. Из потенциальных источников прерываний МК семейства *ARM Cortex-Mx* «рядовыми» разработчиками чаще всего используются прерывания от встроенных ПУ и от внешних по отношению к МК устройств, с позициями от 0 и выше в таблице векторов прерываний (см. рис. 7.8). Запросы данных прерываний обозначаются $IRQk$ (где k – номер позиции в таблице векторов). Все данные прерывания являются маскируемыми. Их приоритеты могут назначаться разработчиком, в зависимости от степени важности того или иного источника прерывания при решении конкретной задачи. В общем случае, распределение приоритетов осуществляется по принципу многоуровневой квазистатической приоритезации (см. пункт 7.2.2).

7.3.2.7. По умолчанию (после сброса) система приоритетов прерываний МК семейства *ARM Cortex-Mx* является одноуровневой (см. рис. 7.8), с назначенными при разработке архитектуры МК значениями приоритетов. При этом чем меньше значение приоритета, присвоенные «по умолчанию», тем выше приоритет. Однако пользовательским ПО может быть осуществлено распределение приоритетов прерываний по **группам** и **подгруппам**, с их обслуживанием по следующим правилам:

- выполнение подпрограммы обслуживания прерывания может быть приостановлено только при поступлении запроса от источника, входящего в группу с более высоким приоритетом, чем у группы, к которой относится источник обслуживаемого прерывания;

- если при выполнении подпрограммы обслуживания прерывания от некоторого источника поступают запросы от источников прерываний, входящих в ту же группу, но в другие подгруппы, они ставятся в очередь, и по завершении подпрограммы, выполнявшейся в момент их поступления, обслуживаются в порядке убывания их **субприоритета**, т. е. приоритетов подгрупп, в которые они включены (первым обслуживается запрос от источника с наивысшим субприоритетом и т. д.);

- при поступлении запросов на прерывания от источников, входящих в одну и ту же группу и подгруппу, они также ставятся в

очередь и обслуживаются в порядке возрастания их номеров (*Positions*) в таблице векторов (см. рис. 7.8).

Наглядные иллюстрации вышеизложенных правил представлены в [9] (см. рис. 7.9 и 7.10).

7.3.2.8. На рис. 7.9 приведен пример временны**х** диаграмм процесса обслуживания 3-х прерываний от источников, входящих в 3 различные группы по уровню приоритета:

- приоритет группы, в которую входит источник *C*, выше, чем у групп, к которым отнесены источники *A* и *B*;
- приоритет группы, в которую входит источник *B*, выше, чем у группы, к которой отнесен источник *A*.

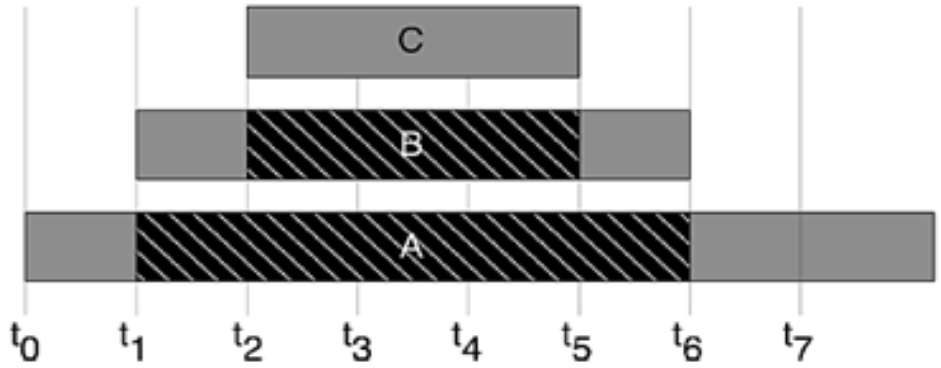


Рис. 7.9. Пример временных диаграмм обслуживания в МК семейства *ARM Cortex-Mx* прерываний от источников, входящих в группы с различными уровнями приоритета (пояснения – в тексте)

Более светлой заливкой обозначены интервалы времени, в течение которых выполняется подпрограмма обслуживания прерывания от соответствующего источника; более темной заливкой – интервалы, в течение которых выполнение данной подпрограммы приостановлено.

В момент времени t_0 поступил запрос на прерывание от источника *A*. Предполагается, что в данный момент не поступило никаких других запросов, и никакие запросы не находятся в очереди на обслуживание. Поэтому ЦП переходит к выполнению подпрограммы обслуживания прерывания от источника *A*.

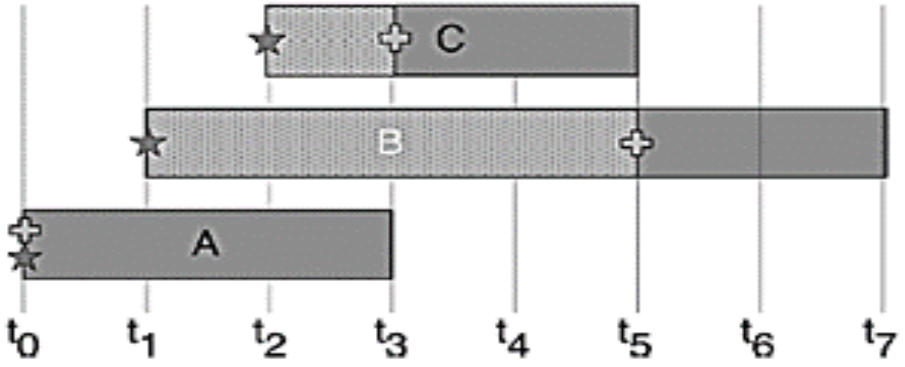
В момент времени t_1 поступает запрос от источника *B*, входящего в группу с более высоким приоритетом, чем та, к которой относится источник *A*. Обслуживание прерывания от источника *A* приостанавливается, и ЦП переходит к подпрограмме обслуживания прерывания от источника *B*. Во время ее выполнения (момент t_2)

поступает запрос на прерывание от источника *C*, входящего в группу с более высоким приоритетом, чем у группы, в которую включен источник *B*. Происходит приостановка обслуживания прерывания от источника *B* и переход к подпрограмме обслуживания прерывания от источника *C*. По ее выполнении (момент времени t_5) возобновляется обслуживание прерывания от источника B, по завершении которого (момент времени t_6) происходит возврат к подпрограмме обслуживания прерывания от источника *A* и ее завершение.

Напомним, что при приостановке выполнения подпрограмм адрес возврата сохраняется в стеке, и извлекается из него при возобновлении работы подпрограммы (см. рис. 7.5).

7.3.2.9. На рис. 7.10 представлен пример временны**х** диаграмм обслуживания прерываний от источников, входящих в одну и ту же группу по приоритету, но в подгруппы с различным субприоритетом. Предполагается, что уровни субприоритета прерываний от источников A и C одинаковы, но выше, чем у прерываний от источника B.

Более светлым цветом заливки обозначены интервалы времени, в течение которых запросы на прерывания находятся в **отложенном** состоянии (т. е. в очереди), более темным – интервалы, в течение которых выполняются подпрограммы обслуживания прерываний.



★ - моменты генерации запросов на прерывания
✚ - моменты начала обслуживания запросов

Рис. 7.10. Пример временных диаграмм обслуживания в МК семейства *ARM Cortex-Mx* прерываний от источников, входящих в одну и ту же группу по уровню приоритета (пояснения – в тексте)

В момент времени t_0 поступает запрос на прерывание от источника *A*, и начинается выполнение подпрограммы его обслуживания, во время которого, в моменты t_1 и t_2, поступают запросы от источников *B* и *C* соответственно. Поскольку эти источники входят в ту же группу по уровню приоритета, что и источник *A*, их запросы не приостанавливают подпрограмму обслуживания прерывания от него, а ставятся в очередь. По выполнении данной подпрограммы (момент времени t_3) вначале обслуживается прерывание от источника *C*, как обладающего более высоким субприоритетом (несмотря на то что запрос от него поступил позднее, чем от источника *B*). После выполнения подпрограммы обслуживания прерывания от источника C (момент времени t_5) обслуживается запрос от источника *B*.

Ситуация, аналогичная представленной на рис. 7.10, будет иметь место, если прерывания от источников A, B и C обладают одинаковым приоритетом и субприоритетом, но номер вектора прерывания (*Position* в таблице на рис. 7.8) от источника *C* меньше, чем от источника *B*.

7.3.2.10. Число групп источников прерываний по уровню приоритета и число подгрупп в их составе задается битовым полем ***PRIGROUP*** регистра ***SCB_AIRCR*** [16, 18], входящего в состав регистров системного назначения *Private peripheral bus* (см. рис. 2.25) и программно-доступного на **привилегированном** уровне работы ЦП. В регистре *SCB_AIRCR* МК подсемейств ***ARM Cortex-M3*** и ***ARM Cortex-M4*** разрядность поля *PRIGROUP* равна 3-м битам. Например:

- при *PRIGROUP* = 011 число групп равно 16-ти, разбиение на подгруппы отсутствует;

- при *PRIGROUP* = 101 число групп равно 4-м, в каждой из них выделяется по 4 подгруппы;

- при *PRIGROUP* = 111 разбиение на группы отсутствует; источники прерывания разбиваются на 16 подгрупп, с правилами обслуживания, иллюстрируемыми рис. 7.10 (см. также пояснения к нему);

подробнее – см. [16] и [18].

Архитектура МК подсемейства ***ARM Cortex-M0*** не предоставляет возможность разбиения групп источников прерываний на подгруппы, и в их регистре *SCB_AIRCR* поле *PRIGROUP* **отсутствует** [19].

7.3.2.11. Распределение приоритетов запросов ***IRQk*** (см. подпункт 7.3.2.6) по группам и подгруппам (при их наличии) осуществляется посредством регистров ***NVIC_IPRx*** (*Interrupt Priority Registers*) [16, 18, 19], входящих в состав контроллера прерываний (*NVIC*), см., например, рис. 1.4. Данные регистры программно-доступны только на **привилегированном** уровне работы. Их разрядность равна 32-м битам, а их количество зависит от подсемейства, к которому принадлежит МК. Например, контроллер прерываний МК подсемейства *ARM Cortex-M0* включает в себя 8 регистров *NVIC_IPRx* (*NVIC_IPR0 – NVIC_IPR7*), подсемейства *ARM Cortex-M3* – 21, а подсемейства *ARM Cortex-M4* – 60 регистров *NVIC_IPRx* [16, 18, 19].

В регистрах *NVIC_IPRx* для назначения группы и подгруппы приоритета каждого из запросов $IRQk$ выделяется 1 байт. Он располагается в регистре с номером $x = \lfloor k/4 \rfloor$ (где $\lfloor \cdot \rfloor$ - символ округления до ближайшего меньшего целого), занимая в нем биты с номерами от $8 \times (k - 4 \times x)$ до $8 \times (k - 4 \times x) + 7$. Например, группа и подгруппа приоритета прерывания $IRQ57$ от 5-го таймера МК модельного ряда *STM32F10xxx* (см. рис. 7.8) должны быть указаны в битах с 8-го по 15-й регистра *NVIC_IPR14*. При этом число битов соответствующего байта, реально задействованных для указания группы и подгруппы приоритета каждого из прерываний, в большинстве подсемейств МК меньше 8-и. Например, в регистрах *NVIC_IPRx* МК подсемейств *ARM Cortex-M3* и *ARM Cortex-M4* для данной цели используются только старшие 4 бита [16, 18], а в регистрах *NVIC_IPRx* МК подсемейства *ARM Cortex-M0* – только старшие 2 бита [19] каждого из байтов; состояния остальных битов игнорируются.

7.3.2.12. Формат битового поля, используемого для указания группы и подгруппы приоритета, назначаемых какому-либо источнику прерываний (см. выше), определяется общим количеством групп и подгрупп, задаваемым полем *PRIGROUP* регистра *SCB_AIRCR* (см. пункт 7.3.2.10) [16, 18, 19]:

- если количество групп и количество подгрупп не равны нулю, то старшие $log_2 n$ бит (где n – число групп) задают номер группы, а младшие $log_2 m$ бит (где m – число подгрупп в пределах группы) – номер подгруппы;

- если разбиение групп на подгруппы отсутствует (что, в частности, имеет место в МК подсемейства *ARM Cortex-M0* [19] – все биты задают номер группы);

- если отсутствует разбиение источников прерываний на группы по приоритету – все биты задают номер подгруппы.

Например, если в битовое поле *PRIGROUP* регистра *SCB_AIRCR* МК модельного ряда *STM32F10xxx* записан двоичный код 101, что соответствует разбиению источников прерываний на 4 группы, с 4-мя подгруппами в пределах каждой из них [18], старшие 2 бита каждого из байтов регистра *NVIC_IPRx* определяют номер группы, назначаемый соответствующему источнику прерываний, а следующие за ними по старшинству 2 бита – номер подгруппы в пределах группы. В частности, при этом 14-й и 15-й биты регистра *NVIC_IPR14* будут задавать номер группы по приоритету, назначаемый прерываниям от 5-го таймера, а 12-й и 13-й биты того же регистра – номер подгруппы в пределах данной группы (см. пример в подпункте 7.3.2.11).

Для разработчиков, предпочитающих программирование на уровне *HAL* (см. подпункт 2.6.1.6), в [9] представлены примеры распределения источников прерываний по группам и подгруппам приоритетности, разработанные на данном уровне.

7.3.2.13. Напомним, что, чем **выше** номер группы (подгруппы), тем **ниже** приоритет (подприоритет). Например, если в МК модельного ряда *STM32F10xxx* (см. рис. 7.8) используемые для решения некоторой задачи источники прерываний распределены по группам и подгруппам приоритета следующим образом:

- прерывания от внешних источников *EXTI0 – EXTI3* включены в 0-ю группу, причем прерываниям *EXTI0* и *EXTI1* назначен 0-й номер подгруппы, а *EXTI2* и *EXTI3* – 1-й;

- прерывания от 5-го, 6-го и 7-го таймеров включены в 0-ю подгруппу 1-ой группы;

то:

- прерывания *EXTI0 – EXTI3* могут приостанавливать выполнение подпрограмм обслуживания прерываний от 5-го и 6-го таймеров;

- при генерации запроса любого из прерываний *EXTI0 – EXTI3* во время выполнения подпрограммы обслуживания какого-либо другого прерывания той же группы работа данной подпрограммы не

будет приостановлена; запрос будет поставлен в очередь и обслужен по завершении выполняемой подпрограммы (см. рис. 7.10);

- очередность обслуживания запросов на прерывания *EXTI0 – EXTI3* следующая: в первую очередь обслуживаются прерывания 0-й подгруппы (*EXTI0* и *EXTI1*), во вторую – 1-ой подгруппы (*EXTI2* и *EXTI3*), в пределах 0-й подгруппы первым обслуживается прерывание *EXTI0*, а 1-ой подгруппы – *EXTI2*, номера векторов которых меньше, чем у *EXTI1* и *EXTI3* соответственно;

- аналогичным образом, при генерации запроса на прерывание от 5-го, 6-го или 7-го таймера во время выполнения подпрограммы обслуживания прерывания от другого таймера той же группы приоритета работа подпрограммы не приостанавливается; запросы ставятся в очередь и, по выполнении данной подпрограммы, обслуживаются в порядке возрастания позиции в таблице векторов прерываний (поскольку они включены в одну и ту же подгруппу);

- порядок обслуживания запросов на прерывания, находящихся в очереди, не зависит от времени их поступления, а определяется номерами подгруппы и позиции в таблице векторов прерывания.

7.3.2.14. Доступные для назначения пользователем приоритеты системных исключений, генерируемых ЦП (отмеченных атрибутом *settable* в таблице векторов на рис. 7.8), а также приоритет прерываний от системного таймера задаются посредством регистров *SCB_SHPRx* (*System handler priority registers*) [16, 18, 19], входящих в состав регистров системного назначения *Private peripheral bus* (см. рис. 2.25) и программно-доступных на **привилегированном** уровне работы ЦП. В МК подсемейств *ARM Cortex-M3* и *ARM Cortex-M4* число таких регистров равно 3-м (*SCB_SHPR1 – SCB_SHPR3*) [16, 18], подсемейства *ARM Cortex-M0* – 2-м (*SCB_SHPR2* и *SCB_SHPR3*) [19]. Например, в МК подсемейства *ARM Cortex-M3* [18]:

- приоритеты системных исключений *Memory Management Fault, Bus Fault* и *Usage Fault* задаются битами 4…7, 12…15 и 20…23 соответственно регистра *SCB_SHPR1*;

- приоритет исключения *SVCCall* назначается битами 28…31 регистра *SCB_SHPR2*;

- приоритет исключения *PendSV* задается битами 20…23 регистра *SCB_SHPR3*;

- приоритет прерывания от системного таймера (*SysTick*) назначается битами 28…31 регистра *SCB_SHPR3*.

Более подробно вопросы назначения приоритетов системных исключений и прерываний от системного таймера освещены в [16, 18, 19].

7.3.2.15. Необходимо остановиться на следующем существенном отличии подсистем прерываний МК семейств *ARM Cortex-Mx* и *AVR*. В семействе *AVR* каждой позиции в таблице векторов прерываний соответствует **определенное событие** в одном из функциональных блоков МК или на каком-либо его внешнем выводе (см. рис. 7.1). В семействе *ARM Cortex-Mx* большинству позиций в таблице векторов соответствуют прерывания, каждое из которых может быть вызвано **различными событиями** в определенном блоке МК. Например, прерывание *UART4 global interrupt* (глобальное прерывание от 4-го универсального асинхронного приемопередатчика) МК модельного ряда *STM32F10xxx* (см. рис. 7.8) может быть вызвано следующими событиями [13]:

- завершением передачи очередного кадра;
- приемом очередного кадра;
- ошибкой в принятом кадре;
- рядом других событий (см. [13]).

Если прерывания по данным событиям разрешены, любое их них сгенерирует *UART4 global interrupt* и вызовет одну и ту же подпрограмму обслуживания. Аналогичным образом, *ADC3 global interrupt* может быть инициирован любым вызывающим прерывание событием в 3-м АЦП, *TIM5 global interrupt* – любым событием, являющимся источником прерывания в 5-м таймере и т. п.

Такая особенность подсистемы прерываний МК семейства *ARM Cortex-Mx* (характерная и для ряда других семейств класса «*high performance*») обусловлена сложностью или невозможностью назначения отдельной позиции в таблице векторов прерываний каждому из событий, потенциально являющихся их источниками, ввиду большого количества таких событий. Оно, в свою очередь, обусловлено развитой системой ПУ и реализуемых ими функций.

Однако, при обслуживании прерывания, источником которого может быть несколько событий, должна быть обеспечена возможность выяснения, **какое именно** из них вызвало прерывание (от чего, естественно, зависит алгоритм его обслуживания). Она реализуется за счет наличия битов признаков (флагов) всех событий, которые могли бы вызвать данное прерывание, обычно

располагаемых в регистре статуса функционального блока МК, являющегося источником соответствующего прерывания (см. также пункт 7.2.4). Например, в состав всех блоков *UART* и *USART* МК модельного ряда *STM32F10xxx* входит регистр статуса (*USART_SR*), единичное состояние 5-го бита которого является признаком наличия принятого кадра в регистре данных приемника, 6-го бита – признаком завершения передачи кадра и т. д. (подробнее – см. [13]). Следовательно, конкретное событие, вызвавшее прерывание, выявляется путем **опроса** битов признаков всех событий, которые могли бы быть его источниками.

Следует отметить, что прерывание по каждому из событий может быть **запрещено** (замаскировано), см. пункт 7.2.4, что осуществляется обнулением бита разрешения прерывания по соответствующему событию. Данные биты располагаются в одном из регистров управления функциональным блоком, являющимся источником событий. Например, в блоках *UART* и *USART* МК модельного ряда *STM32F10xxx* [13] таким регистром является *USART_CR1*, 5-й бит которого является битом разрешения / запрета прерываний по наличию принятого кадра в регистре данных приемника, 6-й бит – прерываний по завершении передачи кадра и т. д. (подробнее – см. [13]).

Также возможен запрет и обслуживания запроса $IRQk$ **в целом** (подробнее см. подпункт 7.3.2.18).

Таким образом, в МК семейства *ARM Cortex-Mx*, если разрешена генерация некоторого запроса на прерывание по 2-м и более событиям, в подпрограмму его обслуживания в обязательном порядке должен быть включен опрос признаков данных событий, с целью корректного ее выполнения. См. примеры 2 и 3 в подпункте 7.3.2.22.

7.3.2.16. Примечание. Из изложенного в подпункте 7.3.2.15 следует, что система прерываний МК семейства *ARM Cortex-Mx* является комбинированной, **векторно-обзорной**. При обслуживании запросов от большинства источников прерываний:

- идентификация функционального блока МК, вызвавшего прерывание, и переход к подпрограмме его обслуживания осуществляются векторным способом, на аппаратном уровне;

- идентификация конкретного события, вызвавшего прерывание, осуществляется выполняемым подпрограммой его обслуживания

обзором (опросом) признаков событий, которые могут вызвать прерывание от соответствующего блока.

Существуют и исключения: генерация ряда запросов на прерывание происходит по конкретному событию в конкретном функциональном блоке МК [13, 14].

Следует заметить, что из-за необходимости опроса признаков событий, вызвавших прерывание в подпрограмме его обслуживания, они **не сбрасываются** при входе в нее, в отличие от флагов прерываний МК семейства *AVR*. Однако, их сброс в нулевое состояние, по крайней мере, перед выходом из подпрограммы, **необходим**; рекомендуется сброс непосредственно после проверки признака [9]. Если сброс не выполнить, ЦП будет автоматически возобновлять выполнение подпрограммы обслуживания прерывания после выхода из нее. Сброс осуществляется записью **нуля** или **единицы** (в зависимости от модельного ряда МК, а также от конкретного источника запроса на прерывание) в соответствующий бит признака; см. примеры, приведенные в подпункте 7.3.2.22. Особенности процедуры сброса того или иного бита признака необходимо уточнять по *Reference Manual* на соответствующий модельный ряд МК.

Для **повышения скорости** обслуживания запросов на прерывания рекомендуется запретить их генерацию по событиям, наступление или не наступление которых не важно при решении конкретной задачи. При этом, естественно, исключается необходимость проверки признаков данных событий в подпрограмме обслуживания прерывания.

7.3.2.17. Следует также остановиться на реализации **внешних прерываний** в МК семейства *ARM Cortex-Mx*. Под ними подразумеваются прерывания от внешних по отношению к МК источников, инициируемые сигналами запросов, поступающими на ПВВ МК.

В то время как в МК семейства *AVR* сигналы запросов прерываний от внешних источников могут подаваться только на определенные выводы МК, сконфигурированные на выполнение альтернативной функции «Вход внешнего прерывания» [6, 8], в МК семейства *ARM Cortex-Mx* приемниками таких сигналов могут служить все выводы всех ПВВ. Однако, подсистема прерываний МК семейства *ARM Cortex-Mx* предусматривает только 16 событий, являющихся источниками прерываний от внешних источников:

поступление сигнала запроса прерывания на 0-й, 1-й, … 15-й вывод **одного** из ПВВ, выбранного при конфигурировании блока внешних прерываний (напомним, что разрядность ПВВ МК семейства *ARM Cortex-Mx* равна 16-ти битам). Данные события обозначаются как *EXTI0, EXTI1, … EXTI15*. При этом возможна инициализация события *EXTI0* по сигналу на 0-м выводе, например, порта *A*, а события *EXTI1* – на 1-м выводе, к примеру, порта *C* и т. п. Однако, инициализация события *EXTIx* сигналами на выводах с номером *x* двух или нескольких различных ПВВ **невозможна**.

На рис. 7.11 в качестве примера показана структурная схема формирования сигналов, служащих признаками событий *EXTIx*, в МК модельных рядов *STM32F405xx/07xx* и *STM32F415xx/17xx* [14].

Сигналы, инициирующие события *EXTIx*, поступают с выходов мультиплексоров, информационные входы которых подключены к выводам с номером *x* каждого из ПВВ, входящих в состав МК. Управление мультиплексорами осуществляется 4-битовыми кодовыми комбинациями в регистрах *EXTICRn* контроллера системной конфигурации (*System configuration controller, SYSCFG*). Например, если в битовое поле *EXTI0* регистра *SYSCFG_EXTICR1* записан код 0000, в поле *EXTI1* данного регистра – код 0011, а в поле *EXTI15* регистра *SYSCFG_EXTICR4* – код 0101, событие *EXTI0* инициируется сигналом на 0-м выводе ПВВ *A*, событие *EXTI1* – сигналом на 1-м выводе ПВВ *D*, а событие *EXTI15* – сигналом на 15-м выводе ПВВ *F* [14].

По аналогичным схемам реализуется формирование сигналов, инициирующих события *EXTIx*, в других подсемействах / модельных рядах МК семейства *ARM Cortex-Mx* (см., например, [13] и [15]). При этом, например, в МК модельного ряда *STM32F10xxx* [13] регистры *EXTICRn* входят в состав регистровой модели ПВВ.

Конфигурирование выводов ПВВ, используемых для внешних прерываний, на выполнение функции «Вход внешнего прерывания» как альтернативной не требуется. Однако, данные выводы должны быть сконфигурированы на работу в режиме входов, «плавающих» или с подтяжкой к питанию или к общей шине (см. пункты 6.3.1 – 6.3.3), в зависимости от схемотехники источников прерываний.

События *EXTIx* могут вызываться фронтом или спадом сигнала на выводе с номером *x* ПВВ, выбранного посредством регистра *EXTICRn* (см. выше), а также как фронтом, так и спадом на данном выводе, в зависимости от состояния битов с номером *x* в регистрах

Rising trigger selection register (EXTI_RTSR) и *Falling trigger selection register (EXTI_FTSR)* контроллера внешних прерываний.

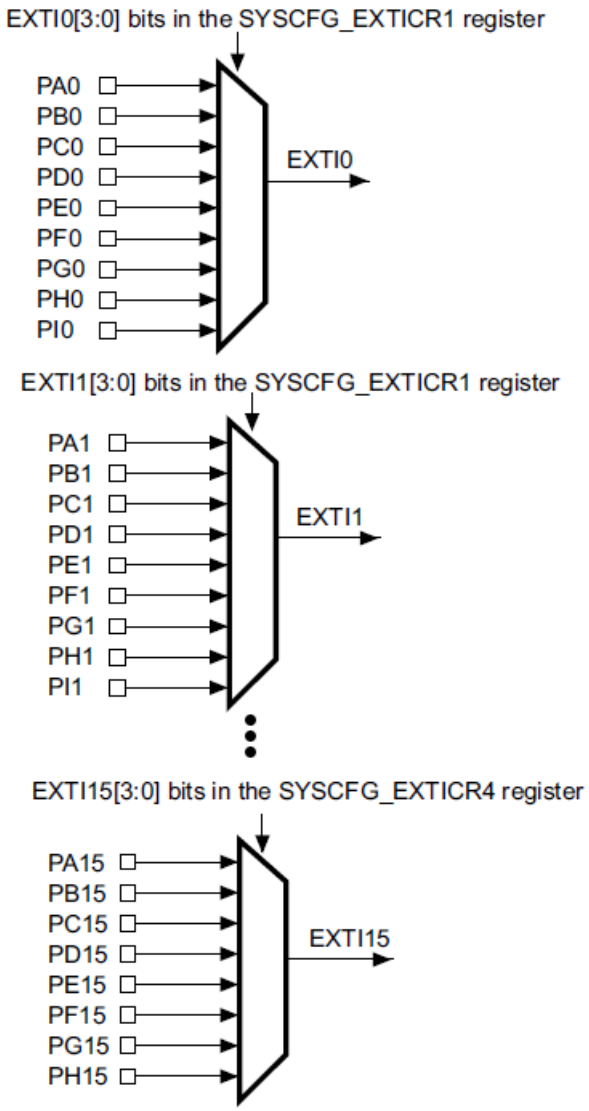


Рис. 7.11. Структурная схема формирования сигналов – признаков событий *EXTIx* в МК модельных рядов *STM32F405xx/07xx* и *STM32F415xx/17xx* [14]

Необходимо отметить, что запрос на внешнее прерывание с **одним и тем** же номером и с одной и той же подпрограммой его обслуживания может быть вызван **различными** событиями *EXTIx*. Например, в таблице векторов прерываний МК модельных рядов *STM32F405xx/07xx* и *STM32F415xx/17xx* [14] отдельные позиции с номерами от 6 до 10 выделены только для прерываний по событиям *EXTI0*, *EXTI1*, *EXTI2*, *EXTI3* и *EXTI4* соответственно. По событиям *EXTI5 – EXTI9* генерируется запрос на одно и то же прерывание (*EXTI9_5*), с номером позиции 23; события *EXTI10 – EXTI15* также

вызывают одно и то же прерывание (*EXTI15_10*), с номером позиции 40. Поэтому в подпрограммах обслуживания внешних прерываний, вызываемых несколькими событиями, должен быть обеспечен **опрос** их признаков, для идентификации конкретного события, инициировавшего прерывание (аналогично подпрограммам обслуживания прерываний от встроенных ПУ МК, см. подпункт 7.3.2.15). В контроллере внешних прерываний МК семейства *ARM Cortex-Mx* признаком события *EXTIx* служит единичное состояние бита с номером x в регистре *EXTI_PR* (*Pending Register*). Как и признаки других событий, вызывающих прерывания, биты регистра *EXTI_PR* должны быть сброшены в нулевое состояние, по крайней мере, перед выходом из подпрограммы, во избежание ее повторного запуска по ее завершении (см. подпункт 7.3.2.16, а также примеры, приведенные в подпункте 7.3.2.22). Обратим внимание на то, что сброс битов регистра *EXTI_PR* осуществляется записью в них **единицы**.

Как и во всех других функциональных блоках МК, в контроллере внешних прерываний имеется возможность их **маскирования** (запрета). Оно осуществляется посредством регистра *EXTI_IMR* (*Interrupt mask register*). Разрешение или запрет прерывания по событию *EXTIx* осуществляется записью соответственно единицы или нуля в бит с номером x данного регистра. При этом, естественно, исключается необходимость проверки признаков событий, прерывания по которым замаскированы, в подпрограммах их обслуживания (см. также подпункт 7.3.2.16).

7.3.2.18. Ввиду изложенных в подпунктах 7.3.2.15 – 7.3.2.17 особенностей подсистемы прерываний МК семейства *ARM Cortex-Mx*, обслуживание прерываний по некоторым событиям, генерирующим запрос $IRQk$, возможно только при **2-х условиях**:

- установке в единицу битов разрешения генерации запроса $IRQk$ по соответствующим событиям, располагаемых в одном из регистров управления функциональным блоком МК, являющимся источником запроса $IRQk$ (АЦП, таймером, *USART* и т. п.);

- установке в единицу бита разрешения прерывания по запросу $IRQk$ в регистре *NVIC_ISERx* (*Interrupt Set Enable Register*) с номером $x = \lfloor k/32 \rfloor$, при этом номер данного бита в регистре *NVIC_ISERx* равен $k - 32x$; регистры *NVIC_ISERx* входят в состав контроллера

прерываний (*NVIC*) и доступны на **привилегированном** уровне работы МК;
причем желателен именно вышеприведенный порядок установки битов разрешения.

См. также примеры, представленные в подпункте 7.3.2.22.

7.3.2.19. Необходимо также отметить, что в МК семейства *ARM Cortex-Mx* возможно маскирование (см. рис. 2.30 и пояснения к нему):

- всех прерываний с конфигурируемым приоритетом, осуществляемое записью единицы в нулевой бит регистра *PRIMASK*;
- всех прерываний, за исключением *NMI* (см. подпункт 7.3.2.3), установкой в единичное состояние нулевого бита регистра *FAULTMASK*.

«По умолчанию» состояния указанных битов – нулевые, т. е. глобальный запрет прерываний отсутствует (в отличие от МК семейства *AVR*).

Также в МК семейства *ARM Cortex-Mx* возможно маскирование всех прерываний со значением приоритета, равным или большим указанного в битах с 4-го по 7-й регистра *BASEPRI* (см. рис. 2.30 и пояснения к нему). Напомним, что чем выше значение приоритета, тем ниже его уровень. По умолчанию, состояния всех данных битов – нулевые; при этом возможно обслуживание прерываний с любым значением приоритета (при соблюдении условий, указанных в подпункте 7.3.2.18).

В заключение необходимо отметить, что в МК семейства *ARM Cortex-Mx* замаскированные прерывания **не ставятся в очередь** [9]. Поэтому не рекомендуется глобальное или групповое маскирование прерываний на длительное время, т. к. при этом имеется вероятность потери запросов, важных для выполнения конкретной задачи [9].

7.3.2.20. В отличие от МК семейства *AVR*, в МК семейства *ARM Cortex-Mx* при переходе к подпрограмме обслуживания прерывания в стеке автоматически сохраняется содержимое не только программного счетчика, но также регистра статуса, регистра связи, РОН *R0 – R3* и *R12* (см. рис. 2.30 и пояснения к нему) [16, 18, 19], что позволяет использовать перечисленные регистры в процессе выполнения подпрограммы. При выходе из нее их содержимое восстанавливается (выгружается из стека).

Если имеется необходимость сохранения в стеке содержимого каких-либо других РОН и / или РСФ, оно должно быть реализовано

в подпрограмме обслуживания прерывания, как и восстановление содержимого соответствующих регистров (посредством его выгрузки из стека) по завершении подпрограммы.

7.3.2.21. В *IDE* ПО МК семейства *ARM Cortex-Mx* подпрограммы обслуживания прерываний, как правило, оформляются в виде *C*-функций типа *void* с именами, закрепленными за ними во входящих в состав *IDE* файлах описания соответствующих моделей / модельных рядов МК. Как правило, имена указанных функций имеют вид: *IRQk_name_IRQHandler*, где *IRQk_name* – имя, присвоенное запросу *IRQk* в соответствии с таблицей векторов прерываний конкретной модели / модельного ряда МК. Например, функциям обслуживания запросов на прерывания МК модельного ряда *STM32F10xxx* (см. рис. 7.8) в большинстве *IDE* присвоены следующие имена:

WWDG_IRQHandler;
PVD_IRQHandler;
TAMPER_IRQHandler;
RTC_IRQHandler;
FLASH_IRQHandler;
RCC_IRQHandler;
EXTI0_IRQHandler;
EXTI1_IRQHandler;
EXTI2_IRQHandler;
EXTI3_IRQHandler;
EXTI4_IRQHandler;

и т. д. Тем не менее, принятые в используемой *IDE* имена функций обслуживания запросов на прерывания необходимо уточнять в процессе разработки ПО МК, руководствуясь документацией на *IDE* и ее «подсказками».

Размещение подпрограмм в ПП по адресам, выделенным для процедуры обслуживания соответствующих прерываний, происходит автоматически, при компиляции кода. Команды возврата из подпрограмм также добавляются при компиляции.

7.3.2.22. Далее представлены типовые примеры конфигурирования функциональных блоков МК семейства *ARM Cortex-Mx* на генерацию прерываний по определенным событиям, а также оформления подпрограмм обслуживания данных прерываний.

Уровень программирования – *CMSIS,* представление операндов – числовое (см. подпункт 2.6.1.14 и Приложение В).

Во всех примерах для упрощения предполагается, что для решения конкретной задачи используется только один из запросов *IRQk*.

Пример 1. Внешнее прерывание по фронту сигнала на 2-м выводе ПВВ *B* МК *STM32F407VG* модельного ряда *STM32F405xx/07xx* [14].

```
//Подключение файла описания МК модельного ряда stm32f4xx
#include "stm32f4xx.h"
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*
Объявление подпрограммы конфигурирования контроллера внешних прерываний и подпрограммы конфигурирования ПВВ B
*/
void extint_conf(void);
void gpio_extint(void);
//Объявление других подпрограмм. Объявление переменных
.
.
.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Подпрограмма конфигурирования контроллера внешних прерываний
void extint_conf(void)
{
/*
Разрешение тактирования контроллера системной конфигурации (SYSCFG) установкой в единицу 14-го бита регистра RCC_APB2ENR разрешения / запрета тактирования функциональных блоков домена APB2 (см. раздел 1.3).
*/
RCC->APB2ENR |=0x00004000;
/*
Назначение ПВВ, являющегося источником внешнего прерывания EXTI2: запись двоичного кода 0001 (что соответствует порту B) в биты с 8-го по 11-й регистра SYSCFG_EXTICR1 контроллера системной конфигурации. В IDE данному регистру назначено имя EXTICR[0] (в соответствии с «подсказкой» IDE).
*/
SYSCFG->EXTICR[0] = 0x00000100;
```

```
/*
```
Демаскирование внешнего прерывания по 2-му выводу выбранного ПВВ (в данном примере ПВВ *B*), установкой в единицу 2-го бита регистра *EXTI_IMR* контроллера внешних прерываний.
```
*/
EXTI->IMR |= 0x00000004;
/*
```
Установка в единицу 2-го бита регистра *EXTI_RTSR* (*Rising trigger selection register*), для указания, что генерация запроса на внешнее прерывание по 2-му выводу выбранного ПВВ должна осуществляться по **фронту** (перепаду из 0 в 1) сигнала на данном выводе.
```
*/
EXTI->RTSR |= 0x00000004;
/*
```
Разрешение обслуживания запроса на прерывание *EXTI2,* занимающего 8-ю позицию в таблице векторов прерываний [14], установкой в единицу 8-го бита регистра *NVIC_ISER0* (см. подпункт 7.3.2.18).
```
*/
NVIC->ISER[0] |= 0x00000100;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*
```
Подпрограмма конфигурирования ПВВ *B*, 2-й вывод которого служит для приема запросов на прерывания
```
*/
void gpio_extint(void)
{
/*
```
Разрешение тактирования ПВВ *B* записью единицы в 1-й бит регистра *RCC_AHB1ENR* разрешения / запрета тактирования функциональных блоков домена *AHB1* (см. раздел 1.3). Параметры конфигурации 2-го вывода оставлены «по умолчанию» («плавающий» цифровой вход, без подтяжек, скорость переключения низкая).
```
*/
RCC->AHB1ENR |= 0x00000002;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Подпрограмма обслуживания прерывания EXTI2
void EXTI2_IRQHandler(void)
{
    /*
    Сброс признака поступления запроса на прерывание *EXTI2* записью единицы во 2-й бит регистра *EXTI_PR* (см. подпункт 7.3.2.17).
    */
    EXTI->PR |= 0x00000004;
```

```
        /*
        Опрос признаков событий, вызвавших прерывание, отсутствует, т. к. прерывание EXTI2 может быть вызвано только одним событием.
        */
        //Основной код подпрограммы
        .
        .
        .
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Коды других объявленных подпрограмм
.
.
.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Код основной программы
int main(void)
{
//Конфигурирование ПВВ B
gpio_extint();
/*
Конфигурирование других функциональных блоков МК, кроме контроллера внешних прерываний
*/
.
.
.
/*
Конфигурирование контроллера внешних прерываний. Желательно осуществлять его после конфигурирования всех остальных используемых функциональных блоков. В противном случае прерывания будут разрешены при еще не сконфигурированных узлах и блоках МК, используемых при решении конкретной задачи, что может привести к некорректной реакции МК на прерывания.
*/
extint_conf();
//Основной программный цикл
.
.
.
}
```

Пример 2. Внешние прерывания по фронту и по спаду сигналов на 0-м и 1-м выводах ПВВ *A* МК *STM32F030F4* модельного ряда *STM32F030xx* [15].

```
//Подключение файла описания МК модельного ряда stm32f0xx
#include "stm32f0xx.h"
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*
Объявление подпрограммы конфигурирования контроллера внешних прерываний и подпрограммы конфигурирования ПВВ A
*/
void extint_conf(void);
void gpio_extint(void);
//Объявление других подпрограмм. Объявление переменных
.
.
.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Подпрограмма конфигурирования контроллера внешних прерываний
void extint_conf(void)
{
/*
Выбор ПВВ, являющегося источником внешних прерываний, в данном примере не требуется, т. к. им служит ПВВ A, назначаемый источником «по умолчанию». Демаскирование внешних прерываний по 0-му и 1-му выводу ПВВ A, установкой в единицу 0-го и 1-го битов регистра EXTI_IMR контроллера внешних прерываний.
*/
EXTI->IMR |= 0x00000003;
/*
 Установка в единицу 0-го и 1-го битов регистров EXTI_RTSR (Rising trigger selection register) и EXTI_FTSR (Falling trigger selection register), для указания, что генерация запроса на внешнее прерывание по 0-му и по 1-му выводам выбранного ПВВ должна осуществляться как по фронту (перепаду из 0 в 1), так и по спаду (перепаду из 1 в 0) сигналов на данных выводах.
*/
EXTI->RTSR |= 0x00000003;
EXTI->FTSR |= 0x00000003;
/*
Разрешение обслуживания запроса на прерывание EXTI0_1, занимающего 5-ю позицию в таблице векторов прерываний [15], установкой в единицу 5-го бита регистра NVIC_ISER0 (см. подпункт 7.3.2.18).
Данное прерывание, как следует из его названия, может быть вызвано 2-мя событиями: поступлением сигнала запроса на 0-й или на 1-й вывод выбранного ПВВ (в данном примере – ПВВ A)
*/
NVIC->ISER[0] |= 0x00000020;
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*
```
Подпрограмма конфигурирования ПВВ *A*, 0-й и 1-й выводы которого служит для приема сигналов запросов на прерывание
```
*/
void gpio_extint(void)
{
/*
```
Разрешение тактирования ПВВ *A* записью единицы в 17-й бит регистра *RCC_AHBENR* разрешения / запрета тактирования функциональных блоков домена *AHB* (см. раздел 1.3). Параметры конфигурации 0-го и 1-го выводов оставлены «по умолчанию» («плавающий» цифровой вход, без подтяжек, скорость переключения низкая).
```
*/
RCC->AHBENR |= 0x00020000;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*
```
Подпрограмма обслуживания прерывания EXTI0_1, вызываемого 2-мя событиями: поступлением сигнала запроса на 0-й или на 1-й вывод выбранного ПВВ (в данном примере – ПВВ *A*)
```
*/
void EXTI0_1_IRQHandler(void)
{
    /*
    ```
    Если запрос поступил на 0-й вывод ПВВ *A* (в единичном состоянии 0-й бит регистра *EXTI_PR*) – выполнение программного фрагмента обслуживания запроса по 0-му выводу
    ```
    */
    if(((EXTI->PR)&0x00000001)!=0x00000000)
    {
    /*
    ```
    Сброс 0-го бита регистра *EXTI_PR* записью в него единицы во избежание многократного выполнения подпрограммы (см. подпункт 7.3.2.16)
    ```
    */
    EXTI->PR |= 0x00000001;
    //Программный фрагмент обслуживания запроса по 0-му выводу ПВВ *A*
    .
    .
    .
    }
```

```
        /*
        Если запрос поступил на 1-й вывод ПВВ A (в единичном состоянии 1-й
        бит регистра EXTI_PR) – выполнение программного фрагмента
        обслуживания запроса по 1-му выводу
        */
        if(((EXTI->PR)&0x00000002)!=0x00000000)
        {
         /*
        Сброс 1-го бита регистра EXTI_PR записью в него единицы во
        избежание многократного выполнения подпрограммы
        */
        EXTI->PR |= 0x00000002;
        //Программный фрагмент обслуживания запроса по 1-му выводу ПВВ A
        .
        .
        .
        }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Коды других объявленных подпрограмм
.
.
.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Код основной программы
int main(void)
{
//Конфигурирование ПВВ A
gpio_extint();
/*
Конфигурирование других функциональных блоков МК, кроме контроллера
внешних прерываний
*/

.
.
.
/*
Конфигурирование контроллера внешних прерываний (см. комментарий к
вызову подпрограммы extint_conf() в Примере 1).
*/
extint_conf();
```

```
//Основной программный цикл
.
.
.
}
```

Пример 3. *USART1 global interrupt* МК *STM32F103C8* модельного ряда *STM32F103xx* [13]. Предполагается, что источниками прерывания могут быть два события - «Наличие принятого кадра в регистре данных приемника» (*RXNE*) и «Завершение передачи кадра» (*TC*). Их признаками являются единичные состояния соответственно 5-го и 6-го бита регистра статуса *USART1*. Прерывания по всем другим событиям *USART1* запрещены.

```
//Подключение файла описания МК модельного ряда stm32f10x
#include "stm32f10x.h"
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Объявление подпрограммы конфигурирования USART1
*/
void usart1_conf(void);
//Объявление других подпрограмм. Объявление переменных
.
.
.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Подпрограмма конфигурирования USART1
void usart1_conf(void)
{
/*
Разрешение тактирования USART1 записью единицы в 14-й бит регистра RCC_APB2ENR разрешения / запрета тактирования функциональных блоков домена APB2 (см. раздел 1.3).
*/
RCC->APB2ENR |= 0x00004000;

/*
Разрешение работы USART1 на прием и на передачу установкой в единицу 2-го бита (RE) и 3-го бита (TE) 1-го регистра управления USART1.
Разрешение прерываний от USART1 по событиям «Наличие принятого кадра в регистре данных приемника» и «Завершение передачи кадра» установкой в единицу 5-го бита (RXNEIE) и 6-го бита (TCIE) 1-го регистра управления USART1.
*/
```

```
USART1->CR1 |= 0x0000006C;
/*
```
Запись в *Baud Rate Register* числа 26,0, соответствующего скорости обмена данными, равной 19200 бит / с при тактовой частоте домена *APB2*, равной 8 МГц.
```
*/
USART1->BRR=0x01A0;
/*
```
Все остальные параметры конфигурации *USART1* оставлены «по умолчанию»
```
*/
/*
```
Разрешение обслуживания прерывания *USART1 global interrupt,* занимающего 37-ю позицию в таблице векторов прерываний [13], установкой в единицу 5-го бита регистра *NVIC_ISER1* (см. подпункт 7.3.2.18).
```
*/
NVIC->ISER[1] |= 0x00000020;
/*
```
Разрешение работы *USART1* установкой в единицу 13-го бита (*UE, USART Enable*) в его 1-м регистре управления
```
*/
USART1->CR1 |= 0x00002000;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Подпрограмма обслуживания прерывания USART1 global interrupt
void USART1_IRQHandler(void)
{
  /*
  Если в единичном состоянии находится 5-й бит (*RXNE*) регистра статуса *USART1* – обработка прерывания по событию «Наличие принятого кадра в регистре данных приемника»
  */
  if (((USART1->SR) & (0x00000020)) == 0x00000020)
  {
   // Сброс 5-го бита регистра статуса USART1 записью в него нуля [13].
   USART1->SR &= 0xFFFFFFDF;


   /*
   Код обработки прерывания по событию «Наличие принятого кадра в регистре данных приемника»
   */
   .
   .
   .
  }
```

```
    /*
    Если в единичном состоянии находится 6-й бит регистра статуса USART1 –
    обработка прерывания по событию «Завершение передачи кадра»
    */
    if (((USART1->SR) & (0x00000040)) == 0x00000040)
    {
      // Сброс 6-го бита регистра статуса USART1 записью в него нуля [13].
      USART1->SR &= 0xFFFFFFBF;
       /*
      Код обработки прерывания по событию «Завершение передачи кадра»
       */
      .
      .
      .
     }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Коды других объявленных подпрограмм
.
.
.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Код основной программы
int main(void)
{
/*
Конфигурирование функциональных блоков МК, кроме USART1
*/
.
.
.
/*
Конфигурирование USART1 (см. комментарий к вызову подпрограммы
extint_conf() в Примере 1).
*/
usart1_conf();
//Основной программный цикл
.
.
.
}
```

Пример 4. Прерывание по обновлении содержимого счетчика 2-го таймера МК *STM32F103C8* модельного ряда *STM32F103xx* [13].

При однонаправленном счете «вверх» обновление происходит по достижении счетчиком максимального значения, в качестве которого служит содержимое регистра *ARR* (*Auto Reload Register*); при обновлении счетчик сбрасывается в ноль. При однонаправленном счете «вниз» обновление происходит по достижении счетчиком нулевого значения; при обновлении в счетчик загружается содержимое регистра *ARR*. Подробнее – см. раздел 9.

```
//Подключение файла описания МК модельного ряда stm32f10x
#include "stm32f10x.h"
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Объявление подпрограммы конфигурирования 2-го таймера
*/
void timer2_conf(void);
//Объявление других подпрограмм. Объявление переменных
.
.
.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Подпрограмма конфигурирования 2-го таймера
void timer2_conf(void)
{
/*
Разрешение тактирования таймера 2 записью единицы в 0-й бит регистра RCC_APB1ENR разрешения / запрета тактирования функциональных блоков домена APB1 (см. раздел 1.3).
*/
RCC->APB1ENR |= 0x00000001;
/*
Задание параметров конфигурации таймера 2 (режима работы, коэффициента деления предделителя, содержимого ARR и т. п., см. раздел 9).
*/
.
.
.

/*
Разрешение прерываний по обновлении таймера 2 записью единицы в 0-й бит (UIE, Update interrupt enable) регистра TIM2 DMA/Interrupt enable register
*/
TIM2->DIER |= 0x00000001;
```

```
/*
```
Разрешение обслуживания прерывания *TIM2 global interrupt,* занимающего 28-ю позицию в таблице векторов прерываний [13], установкой в единицу 28-го бита регистра *NVIC_ISER0* (см. подпункт 7.3.2.18).
```
*/
NVIC->ISER[0] |= 0x10000000;
/*
```
Разрешение работы 2-го таймера установкой в единицу 0-го бита (*CEN, Counter Enable*) в его 1-м регистре управления
```
*/
TIM2->CR1 |= 0x00000001;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Подпрограмма обработки прерывания TIM2 global interrupt
void TIM2_IRQHandler(void)
{
/*
```
Проверка того, какое событие вызвало прерывание, не производится, т. к. в данном примере разрешены прерывания только по обновлению таймера.
```
*/
/*
```
Сброс признака обновления 2-го таймера записью **нуля** [13] в 0-й бит (*UIF, Update interrupt flag*) его регистра статуса. См. также подпункт 7.3.2.16.
```
*/
TIM2->SR = (TIM2->SR)&0xFFFFFFFE;
//Код обработки прерывания
.
.
.
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Коды других объявленных подпрограмм
.
.
.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Код основной программы
int main(void)
{
/*
```
Конфигурирование функциональных блоков МК, кроме 2-го таймера
```
*/
.
.
.
```

```
/*
Конфигурирование 2-го таймера (см. комментарий к вызову подпрограммы
extint_conf() в Примере 1).
*/
timer2_conf();
//Основной программный цикл
.
.
.
}
```

7.3.2.23. Для читателей, предпочитающих программирование на уровне *HAL*, в [9] представлено несколько типовых примеров программирования подсистемы прерываний МК семейства *ARM Cortex-Mx* на данном уровне.

7.3.2.24. В МК семейства *ARM Cortex-Mx* значения **времени отклика** на запрос прерывания и **времени возврата** из подпрограммы обслуживания прерывания зависят от подсемейства, к которому принадлежит МК [9]. Для МК подсемейства *ARM Cortex-M0* они равны 15-ти тактам ЦП, *ARM Cortex-M0+* - 16-ти тактам, других подсемейств – 12-ти тактам [9].

7.4 Выводы по разделу 7

7.4.1. Прерыванием называют приостановку выполнения основной программы по запросу от какого-либо ПУ (резидентного или внешнего по отношению к МК), а также от ядра МК, с выполнением подпрограммы обслуживания соответствующего запроса. Каждый из запросов на прерывание, в свою очередь, генерируется по наступлении определенного **события**, т. е. некоторого явления, имевшего место в каком-либо функциональном узле или блоке МК, распознаваемого ЦП и / или ПУ МК и могущего вызвать определенные действия с их стороны, в том числе генерацию запроса на прерывание.

Как правило, прерывания рационально применять для реакции на события, моменты наступления которых не регулярны и не подконтрольны ПО МК.

7.4.2. В зависимости от источников запросов на прерывания, они подразделяются на следующие основные типы:

- прерывания от внешних по отношению к ЦП источников (например, АЦП, таймеров, внешних по отношению к МК устройств и т. п.), называемые **асинхронными** или **внешними**;

- прерывания, инициируемые некоторыми нештатными событиями, возникшими в ЦП при выполнении программного кода (например, делением на ноль или переполнением стека), и называемые **синхронными** или **внутренними**, а также **исключениями**;

- прерывания, инициируемые ПО, называемые **программными**.

7.4.3. Общими характеристиками механизма прерываний являются следующие:

- выполнение процедуры обслуживания некоторого события по прерыванию, как правило, инициируется аппаратно, источником данного события (а не ПО МК);

- собственно процедура обслуживания выполняется под управлением ПО (конкретно – подпрограммы обработки соответствующего запроса на прерывание).

7.4.4. Все подсистемы обслуживания прерываний МК и других средств ВТ **обеспечивают**:

- распознавание события, являющегося источником запроса на прерывание и автоматический переход к подпрограмме обслуживания соответствующего запроса;

- приоритетное обслуживание прерываний, т. е. первоочередная обработка запросов, вызванных событиями, более важными с точки зрения архитектуры МК или с точки зрения выполняемой им задачи (в общем случае – с приостановкой обслуживания запросов от менее приоритетных источников);

- сохранение данных, необходимых для возврата основную программу (как минимум – адреса возврата) при переходе к подпрограмме обслуживания запроса на прерывание;

- возобновление работы основной программы с того адреса, на котором произошла приостановка ее выполнения, вызванная прерыванием, по выполнении подпрограммы его обслуживания;

- возможность индивидуального программного разрешения или запрета (маскирования) каждого из прерываний, кроме так называемых немаскируемых прерываний.

7.4.5. Распознавание источников запросов на прерывания и их обслуживание в современных МК (и средствах ВТ в целом) базируется на **векторном** способе обработки прерываний. Каждому источнику прерывания присваивается определенный номер – вектор. В свою очередь, каждому вектору ставится в соответствие определенный адрес, по которому в памяти программ должна размещаться начальная команда подпрограммы обслуживания прерывания (*ISR*) от соответствующего источника. По получении от него запроса на прерывание ЦП автоматически переходит к команде, размещенной по указанному адресу. Распознавание источника прерывания и формирование запроса на его обслуживание (*IRQ*), поступающего на ЦП, осуществляются на аппаратном уровне (без участия ПО), контроллером прерываний.

Таблица соответствия начальных адресов *ISR* векторам прерываний называется **таблицей векторов прерываний** и определяется архитектурой конкретного семейства / подсемейства / модельного ряда МК (см. рис. 7.1 и 7.8).

7.4.6. Запрос на прерывание может быть вызван несколькими различными событиями в одном и том же функциональном блоке МК, что характерно, в частности, для МК семейства *ARM Cortex-Mx* (см. подпункт 7.3.2.15). Такие прерывания обрабатываются комбинированным, **векторно-обзорным** (**векторно-опросным**) способом:

- идентификация функционального блока МК, вызвавшего прерывание, и переход к подпрограмме его обслуживания осуществляются векторным способом, на аппаратном уровне;

- идентификация конкретного события, вызвавшего прерывание, осуществляется выполняемым подпрограммой его обслуживания обзором (опросом) битов признаков событий, которые могут вызвать прерывание от соответствующего блока; данные биты располагаются в одном из его регистров (обычно – в регистре статуса).

7.4.7. Приоритетное обслуживание прерываний осуществляется присвоением каждому источнику запроса на прерывание определенного приоритета. Не существует 2-х источников запросов на прерывание с одинаковым приоритетом. В случае одновременного поступления запросов на прерывание от 2-х или

нескольких источников, в первую очередь обслуживается прерывание с наивысшим приоритетом; остальные запросы фиксируются, «ставятся в очередь» и обслуживаются в порядке убывания их приоритета.

7.4.8. Различают **относительное** и **абсолютное** приоритетное обслуживание прерываний. Если любой запрос на прерывание, поступивший во время выполнения подпрограммы обслуживания ранее поступившего запроса, обрабатывается только после ее завершения, независимо от приоритета источника запроса, такой способ обслуживания прерываний называется **относительным**. Если же по поступлении запроса от более приоритетного источника выполнение подпрограммы обслуживания прерывания приостанавливается, и возобновляется только после обработки более приоритетного запроса, данный способ обслуживания прерываний называется **абсолютным**. В частности, в МК семейства AVR «по умолчанию» используется относительный способ обслуживания прерываний, с возможностью программной перенастройки на абсолютный способ (см. подпункт 7.2.2.1). Для МК семейства *ARM Cortex-Mx* характерно сочетание абсолютного и относительного способов: абсолютный способ используется при обслуживании запросов от источников прерываний, относящихся к различным группам по приоритету, относительный – в пределах одной группы (см. подпункты 7.3.2.7 – 7.3.2.9).

7.4.9. Известны следующие основные способы **назначения приоритетов** запросов на прерывания [3, 9]:

- статическая одноуровневая приоритезация;
- многоуровневая квазистатическая приоритезация (пояснения см. далее);
- динамическая приоритезация, наиболее распространенным вариантом которой является циклическая одноуровневая приоритезация (см. далее);
- комбинация вышеперечисленных способов назначения приоритета.

7.4.10. Статическая одноуровневая приоритезация характерна для относительно несложных МК, с общим количеством источников прерываний, не превышающим 20 – 30. Например, на

ней базируется обслуживание прерываний в МК подсемейств *ATmega* и *ATtiny* семейства *AVR*. Ее принцип поясняет рис. 7.2. Приоритет каждого из источников прерываний однозначно определяется номером вектора соответствующего прерывания (см. рис. 7.1 и 7.2): чем меньше номер вектора, тем выше приоритет. Номера векторов, а значит, и приоритеты задаются при разработке архитектуры МК, и не могут быть изменены. Разбиение источников прерываний на группы по уровням приоритета при **одноуровневой** приоритезации **отсутствует**. Обслуживание всех источников осуществляется в соответствии с единой «шкалой приоритетов».

7.4.11. Многоуровневая квазистатическая приоритезация применяется, в основном, в семействах МК с числом источников прерываний более 30 – 40. При таком их количестве рациональным является их разбиение на группы по степени важности при решении конкретной задачи, т. е. назначение каждому из источников прерывания некоторого уровня важности (приоритетности). Любой из источников прерываний, включенный в группу с более высоким уровнем приоритетности, обладает б**о**льшим приоритетом, чем каждый из источников, отнесенных к любой из групп с меньшим уровнем приоритетности. В свою очередь, каждая группа источников прерываний с приоритетностью одинакового уровня может быть разбита на подгруппы (см. подпункты 7.3.2.6 – 7.3.2.9). В пределах подгруппы (в отсутствие разбиения на подгруппы – в пределах группы) приоритеты распределяются в соответствии с номерами векторов, назначенными при разработке архитектуры МК: чем меньше номер вектора, тем выше приоритет источника прерываний в соответствующей группе / подгруппе. Как правило, распределение источников прерываний по группам / подгруппам осуществляется ПО МК, и, в принципе, может изменяться в процессе работы МК. Соответственно, при этом изменяются и приоритеты запросов от этих источников в пределах подсистемы обслуживания прерываний в целом. Поэтому данный способ приоритезации корректнее назвать не статическим, а квазистатическим.

Конкретные примеры многоуровневой квазистатической приоритезации описаны в подпунктах 7.2.2.2 и 7.3.2.6 – 7.3.2.10.

7.4.12. **Динамическая приоритезация** сравнительно мало распространена в МК общего назначения. Из ее возможных вариантов на практике обычно используется **циклическая приоритезация** (*Round Robin*). Ее принцип поясняет рис. 7.4 (см. также пояснения к нему).

7.4.13. При переходе к подпрограмме обслуживания прерывания, необходимые для возврата в основную программу данные (как минимум – адрес возврата), **сохраняются в стеке,** с извлечением их из него по завершении обслуживания прерывания. Тем самым обеспечивается корректное **возобновление** работы основной программы, приостановленной прерыванием. Тот же принцип используется при обслуживании **вложенных** прерываний, т. е. при получении запроса от источника с более высоким приоритетом во время обслуживания прерывания с более низким приоритетом. См. рис. 7.5 и пояснения к нему.

В относительно простых МК класса «*mainstream*», например, семейства *AVR*, при переходе к подпрограмме обслуживания прерывания в стеке сохраняется только адрес возврата. В МК с более развитой архитектурой, например, семейства *ARM Cortex-Mx* в стеке также сохраняется содержимое регистра статуса и ряда РОН (см. подпункт 7.3.2.20). При выходе из подпрограммы их содержимое восстанавливается (выгружается из стека). Если для корректного возобновления работы основной программы существует необходимость запоминания в стеке содержимого каких-либо регистров, сохранение которых не обеспечивается архитектурой МК, оно должно быть предусмотрено при разработке подпрограммы обслуживания прерывания (см. пример в подпункте 7.3.1.5).

7.4.14. Важными параметрами подсистемы обслуживания прерываний МК (особенно при решении задач, время выполнения которых критично) являются **время отклика** на запрос прерывания и **время возврата** из подпрограммы его обслуживания. **Первое** из них, по определению, равно длительности интервала времени от поступления запроса на прерывание до начала выполнения подпрограммы его обслуживания. **Второй** из вышеназванных параметров фактически представляет собой время выполнения команды возврата из подпрограммы обслуживания прерывания. Время отклика и время возврата выражаются в периодах синхросигнала ЦП, и указываются в технической документации на

соответствующее семейство / подсемейство МК (см. подпункты 7.3.1.8 и 7.3.2.24).

7.4.15. Глубиной вложенности прерываний (или **глубиной прерываний**) называют максимальное число запросов на прерывания с возрастающим от запроса к запросу уровнем приоритета, которые могут быть обслужены от приостановки выполнения основной программы по поступлении первого из них и до возврата в нее по обслуживании всех запросов. При **относительном** обслуживании прерываний (см. подпункт 7.2.2.1) глубина вложенности прерываний, очевидно, равна 1. При **абсолютном** обслуживании прерываний она теоретически ограничена только емкостью (глубиной) стека, определяющей максимальное число адресов возврата, которое может в нем храниться (см. рис.7.5). На практике, однако, могут существовать дополнительные ограничения на глубину вложенности прерываний, зависящие от архитектуры конкретного семейства МК.

7.4.16. Архитектурой большинства семейств МК обеспечивается возможность:

- глобального разрешения / запрета прерываний, кроме так называемых немаскируемых;
- выборочного разрешения прерываний по событиям, мониторинг которых необходим в процессе работы МК в составе проектируемого устройства / системы;
- временного запрета (маскирования) прерываний по событиям, на которые ЦП не должен реагировать при определенных состояниях объекта контроля и управления, с последующим разрешением прерываний при выходе объекта из соответствующего состояния.

Архитектура всех семейств МК предусматривает наличие так называемых **немаскируемых** прерываний (*Non-Maskable Interrupts, NMI*), которые не могут быть запрещены программно. Как минимум одно немаскируемое прерывание, **сброс** (*Reset*) присутствует во всех семействах всех классов МК. В МК семейства *ARM Cortex-Mx* существует несколько источников немаскируемых прерываний (подробнее – см. подпункт 7.3.2.3).

7.4.17. Глобальное разрешение / запрет прерываний осуществляется установкой / сбросом определенного бита в одном из РСФ. Например, в МК семейства *AVR* данная функция реализуется установкой / сбросом бита глобального разрешения прерываний (*I*) в регистре статуса (см. рис. 2.24). Состояние данного бита «по умолчанию», например, после сброса – нулевое, т. е. все прерывания, кроме немаскируемых, запрещены. В МК семейства *ARM Cortex-Mx* глобальный запрет / разрешение прерываний осуществляются установкой / сбросом нулевого бита регистров *PRIMASK* и *FAULTMASK* (см. подпункт 7.3.2.19). «По умолчанию» состояния указанных битов – нулевые, т. е. глобальный запрет прерываний отсутствует (в отличие от МК семейства *AVR*).

В ряде семейств МК, например, *ARM Cortex-Mx*, замаскированные прерывания **не ставятся в очередь**. Поэтому не рекомендуется глобальное маскирование прерываний на длительное время, т. к. при этом имеется вероятность потери запросов, важных для выполнения конкретной задачи.

7.4.18. Для **выборочного** разрешения / запрета прерываний по отдельным событиям, в определенном РСФ каждого из ПУ выделяются специальные доступные для записи биты разрешения / запрета прерывания по каждому из событий, которое может быть источником прерывания от данного ПУ (см., например, рис. 7.6). Разрешение осуществляется установкой соответствующего бита в единицу, запрет – его сбросом в ноль.

В семействах МК, в которых каждый запрос на прерывание вызывается определенным событием, для разрешения прерывания по нему необходимо установить в единицу бит маскирования прерываний по данному событию, а также бит глобального разрешения прерываний (если оно предусмотрено архитектурой соответствующего семейства МК); см. пример в подпункте 7.2.4.4.

В семействах МК, в которых запрос на прерывание может быть вызван несколькими различными событиями в одном и том же функциональном блоке МК, условиями разрешения прерывания по какому-либо из них являются:

- установка в единицу бита разрешения генерации запроса на прерывание по соответствующему событию, располагаемого в одном из регистров управления функциональным блоком МК, являющимся источником запроса (АЦП, таймером, *USART* и т. п.);

- установке в единицу бита разрешения прерывания по запросу соответствующему запросу, обычно располагаемого в одном из регистров системного назначения;

причем желателен именно вышеприведенный порядок установки битов разрешения. См. также примеры, представленные в подпункте 7.3.2.22.

7.4.19. Архитектура практически всех семейств МК позволяет контролировать, имело ли место какое-либо из событий, которое могло бы вызвать прерывание, если собственно прерывание по данному событию запрещено. Для этого в одном из регистров каждого из блоков, являющихся источниками прерываний, выделяются специальные биты признаков (флагов) всех событий, вызывающих прерывания от соответствующего блока (см., например, рис. 7.7).

В семействах МК, в которых каждый запрос на прерывание вызывается определенным событием (например, в МК семейства *AVR*), данные биты, как правило, сбрасываются при переходе к подпрограмме обслуживания прерывания по соответствующему событию. Если же данное прерывание запрещено, биты признаков могут быть использованы для контроля пользовательской программой наличия или отсутствия соответствующих событий. Их сброс при этом также осуществляется программно. См. пример в подпункте 7.2.4.5.

В семействах МК, в которых запрос на прерывание может быть вызван несколькими различными событиями в одном и том же функциональном блоке МК, биты признаков событий не сбрасываются при переходе к подпрограмме обслуживания прерывания, и по их состояниям определяется событие, вызвавшее прерывание (см. примеры 2 и 3 в подпункте 7.3.2.22). Они же могут быть использованы для контроля наличия / отсутствия соответствующих событий, если прерывания по ним запрещены. В подпрограмме обслуживания прерывания, после анализа битов признаков, должен быть выполнен их сброс, во избежание ее многократного выполнения (см. подпункт 7.3.2.22).

7.4.20. В современных *IDE* ПО МК подпрограммы обслуживания прерываний оформляются в виде *C*-функций с именами, закрепленными за ними во входящих в состав *IDE* файлах описания соответствующих семейств / модельных рядов МК (см. примеры в подпунктах 7.3.1.5 и 7.3.2.22). В процессе разработки ПО

имена данных функций необходимо уточнять, руководствуясь документацией на *IDE* и ее «подсказками».

Размещение подпрограмм в ПП по адресам, выделенным для процедуры обслуживания соответствующих прерываний, происходит автоматически, при компиляции кода. Команды возврата из подпрограмм также добавляются при компиляции.

7.4.21. Примеры структурно-архитектурных решений подсистемы обслуживания прерываний, приведенные в подразделе 7.3, являются типовыми для МК общего назначения в целом.