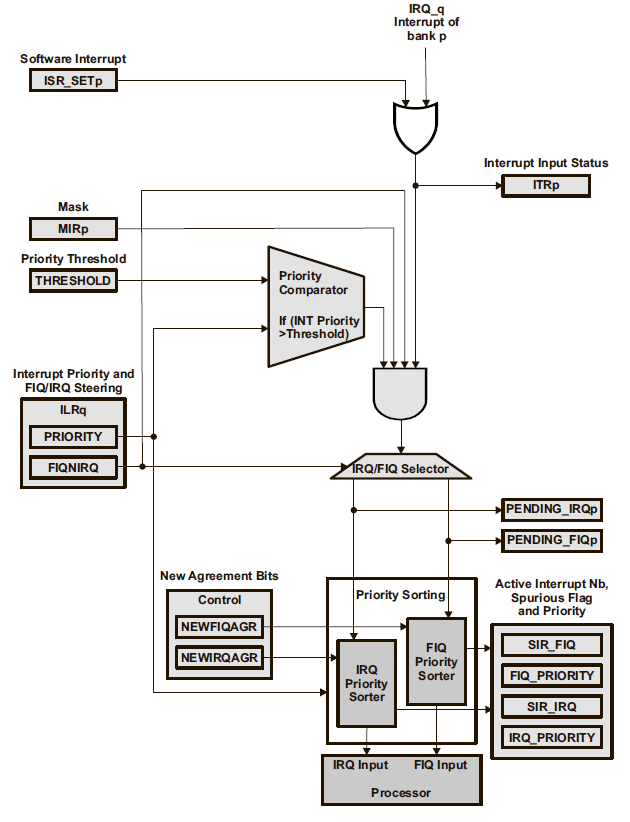
**6.1 Функциональное описание**

Контроллер прерываний обрабатывает входящие прерывания путем маскирования и сортировки по приоритету для получения сигналов прерывания процессора, к которому он подключен. На рис. 6-1 показано представление обработки прерывания верхнего уровня.

ПРИМЕЧАНИЕ. FIQ недоступен на устройствах общего назначения (GP).

**Рисунок 6-1. Блок-схема контроллера прерываний**



**6.1.1 Обработка прерываний**

**6.1.1.1Выбор входа**

**INTC** поддерживает только обнаружение входящих прерываний с учетом уровня. Периферийное устройство, утверждающее прерывание поддерживает его до тех пор, пока программное обеспечение не обработает прерывание и не даст указание периферийному устройству снять прерывание.

Программное прерывание генерируется, если установлен соответствующий бит в регистре **MPU\_INTC.INTC\_ISR\_SETn** (номер банка регистров: n = [0,1,2,3] для подсистемы **MPU INTC**, 128 входящих линий прерывания поддерживается). Программное прерывание сбрасывается, когда соответствующий бит в регистр **MPU\_INTC.INTC\_ISR\_CLEARn** записывается. Обычно эта функция используется для отладки программного обеспечения.

**6.1.1.2 Маскировка**

**6.1.1.2.1 Индивидуальное маскирование**

Обнаружение прерываний на каждой входящей линии прерывания может быть включено или отключено независимо от регистра **MPU\_INTC.INTC\_MIRn**  маски прерывания. В ответ на немаскированное входящее прерывание **INTC** может генерировать один из двух типов запросов прерывания к процессору:

-  IRQ: низкоприоритетный запрос на прерывание.

-  FIQ: запрос быстрого прерывания (недоступно на устройствах общего назначения (GP))

Тип запроса прерывания определяется битом **MPU\_INTC.INTC\_ILRm[0] FIQNIRQ (m=[0,127]).** Текущий статус входящего прерывания до маскирования можно прочитать из регистра **MPU\_INTC.INTC\_ITRn**. После маскирования и выбора **IRQ/FIQ** и до выполнения сортировки по приоритету статус прерывания доступен для чтения. из регистров **MPU\_INTC.INTC\_PENDING\_IRQn** и **MPU\_INTC.INTC\_PENDING\_FIQn.**

**6.1.1.2.2 Маскировка приоритета**

Для более быстрой обработки высокоприоритетных прерываний используется программируемый порог маскирования приоритета. (поле **MPU\_INTC.INTC\_THRESHOLD[7:0] PRIORITYTHRESHOLD**). Этот порог приоритета позволяет вытеснение прерываниями с более высоким приоритетом; все прерывания с более низким или равным приоритетом, чем порог маскируются. Однако приоритет 0 никогда не может быть замаскирован этим порогом; порог приоритета **0** обрабатывается так же, как приоритет **1**. Значения полей **PRIORITY** и **PRIORITYTHRESHOLD** могут быть установлены между **0x0** и **0x7F**; **0x0** — самый высокий приоритет, **0x7F** — самый низкий. Когда маскирование приоритета не нужно, пороговое значение приоритета **0xFF** отключает механизм порога приоритета. Это значение также сброс по умолчанию для обратной совместимости с предыдущими версиями **INTC**.

**6.1.1.3 Сортировка по приоритету**

Каждой входящей линии прерывания назначается уровень приоритета (**0** — самый высокий). Как уровень приоритета, так и тип запроса прерывания настраивается регистром **MPU\_INTC.INTC\_ILRm**. Если более одного входящее прерывание с одинаковым уровнем приоритета и типом запроса на прерывание происходит одновременно, номерное прерывание обслуживается первым. При обнаружении одного или нескольких немаскированных входящих прерываний **INTC** разделяет **IRQ** и **FIQ**, используя соответствующий бит **MPU\_INTC.INTC\_ILRm[0] FIQNIRQ**. Результат помещается в **INTC\_PENDING\_IRQn** или **INTC\_PENDING\_FIQn**, если в настоящее время нет других прерываний. В процессе обработки **INTC** активирует **IRQ/FIQ** и начинает вычисление приоритета. Сортировка по приоритету для **IRQ** и **FIQ** может выполняться параллельно. Каждый сортировщик приоритетов **IRQ/FIQ** определяет номер прерывания с наивысшим приоритетом. И этот номер помещается в соответствующее поле **MPU\_INTC.INTC\_SIR\_IRQ[6:0] ACTIVEIRQ** или поле **MPU\_INTC.INTC\_SIR\_FIQ[6:0] ACTIVEFIQ**. Значение сохраняется до тех пор, пока не будет установлен бит **MPU\_INTC.INTC\_CONTROL NEWIRQAGR** или **NEWFIQAGR**. Как только периферийное устройство, сгенерировавшее прерывание, было обслужено и входящее прерывание снято, пользователь должен написать в соответствующий **NEWIRQAGR** или **NEWFIQAGR**, чтобы указать **INTC**, что прерывание было обработано. Если есть какие-либо в ожидании немаскированных входящих прерываний для этого типа запроса на прерывание, **INTC** перезапускает соответствующий сортировщик приоритетов; в противном случае линия прерывания **IRQ** или **FIQ** сбрасывается.

**6.1.2 Защита регистра**

Если бит **MPU\_INTC.INTC\_PROTECTION[0] PROTECTION** установлен, доступ к регистрам **INTC** запрещен. Ограничен привилегированным режимом. Доступ к регистру **MPU\_INTC.INTC\_PROTECTION** всегда ограничен в привилегированный режим. Для получения дополнительной информации см. Раздел 6.6.1.7, регистр **INTC\_PROTECTION (смещение = 4Ch) [сброс = 0h].**

**6.1.3 Энергосбережение модуля**

**INTC** обеспечивает функцию автоматического бездействия в своих трех доменах тактирования:

- Интерфейсное тактирование

- Функциональное тактирование

- Тактирование синхронизатора

Режим энергосбережения с автоматическим бездействием интерфейсного тактирования включен, если  в **MPU\_INTC.INTC\_SYSCONFIG[0]** бит **AUTOIDLE** установлен в **1**. Когда этот режим включен и на  интерфейсе шины нет активности, интерфейсноге тактирование отключаются внутри модуля, что снижает энергопотребление. Когда появляется активность на шине тактирование интерфейса перезапускаютя без каких-либо задержек. После сброса этот режим по умолчанию отключен. Режим энергосбережения функционального тактирования с автоматическим холостым ходом включается, если **MPU\_INTC.INTC\_IDLE[0]** бит **FUNCIDLE** установлен в **0**. Когда этот режим включен и нет активного прерывания (прерывание **IRQ** или **FIQ** обрабатывается или генерируется) или нет ожидающего входящего прерывания, функциональное тактирование отключено внутри модуля, что снижает энергопотребление. Когда обнаруживается новое немаскированное входящее прерывание,функциональное тактирование перезапускаются, и **INTC** обрабатывает прерывание. Если этот режим отключен, задержка прерывания уменьшается на один цикл. После сброса этот режим включен по умолчанию. Тактирование синхронизатора позволяет обрабатывать внешние асинхронные прерывания. Ресинхронизированные до того, как они будут замаскированы. Входное тактирование синхронизатора имеет автоматический режим энергосбережения в режиме холостого хода. Режим включен, если бит **MPU\_INTC.INTC\_IDLE[1] TURBO** установлен в **1**. Если включен режим автоматического холостого хода, мощность в режиме ожидания снижается, но задержка прерывания **IRQ** или **FIQ** увеличивается с четырех до шести функциональных тактовых циклов. Эта функция может быть включена динамически в соответствии с требованиями устройства. После сброса этот режим отключен по умолчанию.

**6.1.4 Обработка ошибок**

Следующие обращения вызовут ошибку:

- Нарушение привилегий (попытка доступа к регистру **PROTECTION** в пользовательском режиме или любому регистру в пользовательском режиме).

если бит защиты установлен)

- Неподдерживаемые команды

Следующие обращения не вызовут никаких ошибок:

- Доступ к недекодированному адресу

- Запись в регистр только для чтения

**6.1.5 Обработка прерываний**

Генерация прерывания **IRQ/FIQ** занимает четыре такта **INTC** (плюс-минус один цикл), если **MPU\_INTC.INTC\_IDLE[1]** бит **TURBO** установлен в **0**. Если бит **TURBO** установлен в 1, генерация прерывания занимает шесть тактов, но энергопотребление снижается во время ожидания прерывания. Эти задержки могут быть уменьшено на один цикл за счет отключения автоматического бездействия функционального тактирования (установлен бит **MPU\_INTC.INTC\_IDLE[0] FUNCIDLE** в **1**), но энергопотребление увеличивается, поэтому выгода минимальна. Чтобы свести к минимуму задержку прерывания при возникновении немаскированного прерывания, генерируется прерывание **IRQ** или **FIQ**  до завершения приоритетной сортировки. Сортировка по приоритету занимает **10** функциональных тактов, что меньше минимального количество циклов, необходимое для переключения **MPU** на контекст прерывания после получения события  **IRQ** или **FIQ**. Любое чтение регистра **MPU\_INTC.INTC\_SIR\_IRQ** или **MPU\_INTC.INTC\_SIR\_FIQ** во время процесса сортировки по приоритетам останавливается до завершения этого процесса и обновления соответствующего регистра. Тем не менее задержка между генерацией запроса на прерывание и выполнением процедуры обслуживания прерывания такая, что сортировка по приоритету всегда завершается до  чтения регистров **MPU\_INTC.INTC\_SIR\_IRQ** или  **MPU\_INTC.INTC\_SIR\_FIQ** .

**6.2 Базовая модель программирования**

**6.2.1 Последовательность инициализации**

1. Запрограммируйте регистр **MPU\_INTC.INTC\_SYSCONFIG:** При необходимости включите тактирование интерфейса автоматического стробирования установкой бита **AUTOIDLE**.

2. Запрограммируйте регистр **MPU\_INTC.INTC\_IDLE**: При необходимости отключите функциональное тактирование автоматическго стробирования   или

включите автозапуск синхронизатора, установив соответственно бит **FUNCIDLE** или бит **TURBO**.

3. Запрограммируйте регистр **MPU\_INTC.INTC\_ILRm** для каждой линии прерывания: Назначьте уровень приоритета и установите бит **FIQNIRQ** для прерывания **FIQ** (по умолчанию прерывания отображаются как **IRQ**, а приоритет равен **0x0** [самый высокий]).

4. Запрограммируйте регистр **MPU\_INTC.INTC\_MIRn**: Разрешить прерывания (по умолчанию все линии прерываний

под маской).

**ПРИМЕЧАНИЕ**:Программирование регистров **MPU\_INTC.INTC\_MIRn**, **MPU\_INTC.INTC\_MIR\_SETn** и **MPU\_INTC.INTC\_MIR\_CLEARn** нужно для облегчения маскирования, для обратной совместимости желателно писать непосредственно в регистр **MPU\_INTC.INTC\_MIRn**.

**6.2.2 Последовательность обработки INTC**

После настройки регистров **INTC\_MIRn** и **INTC\_ILRm** для включения и назначения приоритетов входящим прерываниям, прерывание обрабатывается, как описано в следующих подразделах. Обработка последовательности **IRQ** и **FIQ**  очень похожи, различия для последовательности **FIQ** показаны после символа '/' в коде ниже.

1. Принимается одно или несколько немаскированных входящих прерываний (сигналы **M\_IRQ\_n**) и выводятся **IRQ** или **FIQ**. (**IRQ**/**FIQ**) в настоящее время не заявлены.

2. Если бит **INTC\_ILRm[0] FIQNIRQ** сброшен в **0**, генерируется выходной сигнал **MPU\_INTC\_IRQ**. Если бит **FIQNIRQ** установлен в **1**, генерируется выходной сигнал **MPU\_INTC\_FIQ**.

3. **INTC** выполняет сортировку по приоритету и обновляет **INTC\_SIR\_IRQ[6:0] ACTIVEIRQ**./**INTC\_SIR\_FIQ[6:0]** поле **ACTIVEFIQ** с текущим номером прерывания.

4. Во время сортировки по приоритету, если **IRQ**/**FIQ** включен на стороне хост-процессора, хост-процессор автоматически сохраняет текущий контекст и выполняет **ISR** следующим образом. Хост-процессор ARM автоматически выполняет следующие действия в псевдокоде:

5. ISR сохраняет оставшийся контекст, идентифицирует источник прерывания, считывая **ACTIVEIRQ**/**ACTIVEFIQ** и переходит к соответствующему обработчику подпрограммы следующим образом:

***ВНИМАНИЕКод на шагах 5 и 7 представляет собой ассемблерный код, совместимый с ARM архитектуры V6 и V7. Этот код разработан для набора инструментов CODE COMPOSER STUDIO. Это черновая версия, тестировалась только на emulated environment.***

;INTC\_SIR\_IRQ/INTC\_SIR\_FIQ register address

INTC\_SIR\_IRQ\_ADDR/INTC\_SIR\_FIQ\_ADDR .word 0x48200040/0x48200044

; ACTIVEIRQ bit field mask to get only the bit field

ACTIVEIRQ\_MASK .equ 0x7F

\_IRQ\_ISR/\_FIQ\_ISR:

; Save the critical context

STMFD SP!, {R0-R12, LR} ; Save working registers and the Link register

MRS R11, SPSR ; Save the SPSR into R11

; Get the number of the highest priority active IRQ/FIQ

LDR R10, INTC\_SIR\_IRQ\_ADDR/INTC\_SIR\_FIQ\_ADDR

LDR R10, [R10] ; Get the INTC\_SIR\_IRQ/INTC\_SIR\_FIQ register

AND R10, R10, #ACTIVEIRQ\_MASK ; Apply the mask to get the active IRQ number

; Jump to relevant subroutine handler

LDR PC, [PC, R10, lsl #2] ; PC base address points this instruction + 8

NOP ; To index the table by the PC

; Table of handler start addresses

.word IRQ0handler ;For IRQ0 of BANK0

.word IRQ1handler

.word IRQ2handler

LR = PC + 4 /\* return link \*/

SPSR = CPSR /\* Save CPSR before execution \*/

CPSR[5] = 0 /\* Execute in ARM state \*/

CPSR[7] = 1 /\* Disable IRQ \*/

CPSR[8] = 1 /\* Disable Imprecise Data Aborts \*/

CPSR[9] = CP15\_reg1\_EEbit /\* Endianness on exception entry \*/

if interrupt == IRQ then

CPSR[4:0] = 0b10010 /\* Enter IRQ mode \*/

if high vectors configured then

PC = 0xFFFF0018

else

PC = 0x00000018 /\* execute interrupt vector \*/

else if interrupt == FIQ then

CPSR[4:0] = 0b10001 /\* Enter FIQ mode \*/

CPSR[6] = 1 /\* Disable FIQ \*/

if high vectors configured then

PC = 0xFFFF001C

else

PC = 0x0000001C /\* execute interrupt vector \*/

endif

6. Обработчик подпрограммы выполняет код, специфичный для периферийного устройства, генерирующего прерывание, путем обработки события и снятие условия прерывания на периферийной стороне.

; IRQ0 subroutine

IRQ0handler:

; Save working registers

STMFD SP!, {R0-R1}

; Now read-modify-write the peripheral module status register

; to de-assert the M\_IRQ\_0 interrupt signal

; De-Assert the peripheral interrupt

MOV R0, #0x7 ; Mask for 3 flags

LDR R1, MODULE0\_STATUS\_REG\_ADDR ; Get the address of the module Status Register

STR R0, [R1] ; Clear the 3 flags

; Restore working registers LDMFD SP!, {R0-R1}

; Jump to the end part of the ISR

B IRQ\_ISR\_end/FIQ\_ISR\_end

7. После возврата из подпрограммы **ISR** устанавливает бит **NEWIRQAGR**/**NEWFIQAGR**, чтобы включить обработку последующих ожидающих запросов **IRQ**/**FIQ** и восстановить контекст **ARM** в следующем коде.

Поскольку записи размещаются на шине Interconnect, чтобы убедиться, что предыдущие записи выполнены перед включением **IRQ**/**FIQ** используется барьер синхронизации данных. Эта операция гарантирует, что линия **IRQ**/**FIQ** сбрасывается перед включением **IRQ**/**FIQ**. После этого **INTC** обрабатывает любые другие ожидающие прерывания или сбрасывает сигнал **IRQ**/**FIQ**, если прерывания нет.

; INTC\_CONTROL register address

INTC\_CONTROL\_ADDR .word 0x48200048;

NEWIRQAGR/NEWFIQAGR bit mask to set only the NEWIRQAGR/NEWFIQAGR bit

NEWIRQAGR\_MASK/NEWFIQAGR\_MASK .equ 0x01/0x02

IRQ\_ISR\_end/FIQ\_ISR\_end:

; Allow new IRQs/FIQs at INTC side

; The INTC\_CONTROL register is a write only register so no need to write back others bits

MOV R0, #NEWIRQAGR\_MASK/NEWFIQAGR\_MASK ; Get the NEWIRQAGR/NEWFIQAGR bit position

LDR R1, INTC\_CONTROL\_ADDR

STR R0, [R1] ; Write the NEWIRQAGR/NEWFIQAGR bit to allow new IRQs/FIQ

; Data Synchronization Barrier

MOV R0, #0

MCR P15, #0, R0, C7, C10, #4

; restore critical context

MSR SPSR, R11 ; Restore the SPSR from R11

LDMFD SP!, {R0-R12, LR} ; Restore working registers and Link register

; Return after ha

8. После возврата **ISR ARM** автоматически восстанавливает свой контекст следующим образом:

CPSR = SPSR

PC = LR

На рис. 6-2 показана последовательность обработки **IRQ**/**FIQ** от периферийного модуля исходного устройства до прерывание основной программы. Механизм сортировки по приоритету замораживается во время последовательности обработки прерывания. Если условие прерывания происходит в течение этого времени, прерывание не теряется. Он отсортируется, когда будет установлен бит **EWIRQAGR**/**NEWFIQAGR**.(реактивирована приоритетная сортировка).

**Рисунок 6-2. Последовательность обработки IRQ/FIQ**

**Железо**

**Софт**

**Шаг 1 назначение M\_IRQ**

Периферийный модуль SOC

**MPU INTC**

Если IRQ\_n не замаскирован и настроен как IRQ/FIQ, устанавливается линия MPU\_INTC\_IRQ/MPU\_INTC\_FIQ .

**Шаг 2 назначение MPU\_INTC\_IRQ/MPU\_INTC\_FIQ**

**Главная программа(main())**

**ARM Host Processor (Шаг 4)**

- Выполнение инструкции номер 1

- Выполнение инструкции номер N

Если FIQ включены (F==0):

- Завершить текущую инструкцию номер N

-Сохранить адрес следующей инструкции выполняемой в Link Register

- Сохраните CPSR перед выполнением в SPSR

- Войдите в режим ARM FIQ

- Отключить IRQ и FIQ на стороне ARM

- Выполнить вектор прерывания.

**Branch**

- Сохранить критический контекст ARM

- Определить источник прерывания

- Переход к соответствующему обработчику подпрограммы прерывания

**Прерывание IRQ/FIQ (Шаг 5)**

**Branch**

**Соответствующий обработчик прерывания IRQ/FIQ (Шаг 6)**

- Обрабатывает событие (функциональная процедура)

- Сбрасывает прерывание M\_IRQ\_n на стороне периферии SOC модуля.

**Branch**

- разрешает новоеIRQ/FIQ на стороне INTC, установкой NEWIRQAGR/NEWFIQAGR бит в 1.

- восстанавливает критический контекст ARM.

**Return**

- Выполнение инструкции номер N + 1

**Главная программа(main())**

**Return**

- Восстановить всю CPSR

- Восстановить PC

**Прерывание IRQ/FIQ (Шаг 7)**

**ARM Host Processor (Шаг 8)**

**6.2.3 Последовательность упреждающей обработки INTC**

Упреждающие прерывания, также называемые вложенными прерываниями, могут уменьшить задержки для прерываний с более высоким приоритетом. Упреждающая **ISR** может быть приостановлена прерыванием с более высоким приоритетом. Таким образом, прерывание с более высоким приоритетом может быть обработано немедленно. Вложенные прерывания следует использовать осторожно, чтобы избежать использования поврежденных данных. Программисту необходимо сохранить поврежденные регистры и включить **IRQ** или **FIQ** на стороне ARM. Последовательности обработки **IRQ** и **FIQ** очень похожи, различия для последовательности **FIQ** показаны после символа '/' в приведенном ниже коде. Чтобы включить вытеснение **IRQ/FIQ IRQ/FIQ** с более высоким приоритетом, программисты могут следовать этой процедуре, чтобы написать **ISR** в начале **IRQ/FIQ ISR**:

1. Сохраните регистры критического контекста **ARM**.

2. Сохраните поле **INTC\_THRESHOLD PRIORITYTHRESHOLD** перед его изменением.

3. Прочитайте приоритет активного прерывания в **INTC\_IRQ\_PRIORITY IRQPRIORITY/INTC\_FIQ\_PRIORITY. FIQPRIORITY** и запишите его в поле **PRIORITYTHRESHOLD(1)**.

4. Прочитайте номер активного прерывания в **INTC\_SIR\_IRQ[6:0] ACTIVEIRQ/INTC\_SIR\_FIQ[6:0]**. Поле **ACTIVEFIQ** для идентификации источника прерывания.

5. Запишите 1 в соответствующие биты **INTC\_CONTROL NEWIRQAGR** и (2) **NEWFIQAGR** во время прерывани которое все еще обрабатывается, чтобы разрешить вытеснение только прерываний с более высоким приоритетом.

6. Поскольку записи размещаются на шине Interconnect, чтобы убедиться, что предыдущие записи выполнены перед включением **IRQ/FIQ** используется барьер синхронизации данных. Эта операция гарантирует, что **IRQ** линия сбрасывается до включения **IRQ/FIQ**.

7. Включите **IRQ/FIQ** на стороне **ARM**.

8. Перейти к соответствующему обработчику подпрограммы.

Ниже показано как это сделать в коде:

***ВНИМАНИЕ***

***Код представляет собой ассемблерный код, совместимый с ARM архитектуры V6 и V7. Этот код разработан для набора инструментов CODE COMPOSER STUDIO. Это черновая версия, тестировалась только на emulated environment.***

; bit field mask to get only the bit field

ACTIVEPRIO\_MASK .equ 0x7F

\_IRQ\_ISR:

; Step 1 : Save the critical context

STMFD SP!, {R0-R12, LR} ; Save working registers

MRS R11, SPSR ; Save the SPSR into R11

; Step 2 : Save the INTC\_THRESHOLD register into R12

LDR R0, INTC\_THRESHOLD\_ADDR

LDR R12, [R0]

(1) The priority-

threshold mechanism is enabled automatically when writing a priority in the range of 0x00 to

0x3F. Writing a value of 0xFF (reset default) disables the priority-

threshold mechanism. Values between 0x3F and 0xFF must not be used. When the hardware-

priority threshold is in use, the priorities of interrupts selected as FIQ or IRQ become linked

otherwise, they are independent. When they are linked, all FIQ priorities must be set higher than

all IRQ priorities to maintain the relative priority of FIQ over IRQ.

(2) When handling FIQs using the priority-

threshold mechanism, both NEWFIQAGR and NEWIRQAGR bits must be written at the same time to ensure

that the new priority threshold is applied while an IRQ sort is in progress. This IRQ will not

have been seen by the ARM, as it will have been masked on entry to the FIQ ISR. However, the

source of the IRQ remains active and it is finally processed when the priority threshold falls to

a priority sufficiently low to allow it to be processed. The precaution of writing to New FIQ

Agreement is not required during an IRQ ISR, as FIQ sorting is not affected (provided all FIQ

priorities are higher than all IRQ priorities).

; Step 3 : Get the priority of the highest priority active IRQ

LDR R1, INTC\_IRQ\_PRIORITY\_ADDR/INTC\_FIQ\_PRIORITY\_ADDR

LDR R1, [R1] ; Get the INTC\_IRQ\_PRIORITY/INTC\_FIQ\_PRIORITY register

AND R1, R1, #ACTIVEPRIO\_MASK ; Apply the mask to get the priority of the IRQ

STR R1, [R0] ; Write it to the INTC\_THRESHOLD register

; Step 4 : Get the number of the highest priority active IRQ

LDR R10, INTC\_SIR\_IRQ\_ADDR/INTC\_SIR\_FIQ\_ADDR

LDR R10, [R10] ; Get the INTC\_SIR\_IRQ/INTC\_SIR\_FIQ register

AND R10, R10, #ACTIVEIRQ\_MASK ; Apply the mask to get the active IRQ number

; Step 5 : Allow new IRQs and FIQs at INTC side

MOV R0, #0x1/0x3 ; Get the NEWIRQAGR and NEWFIQAGR bit position

LDR R1, INTC\_CONTROL\_ADDR

STR R0, [R1] ; Write the NEWIRQAGR and NEWFIQAGR bit

; Step 6 : Data Synchronization Barrier

MOV R0, #0

MCR P15, #0, R0, C7, C10, #4

; Step 7 : Read-modify-write the CPSR to enable IRQs/FIQs at ARM side

MRS R0, CPSR ; Read the status register

BIC R0, R0, #0x80/0x40 ; Clear the I/F bit

MSR CPSR, R0 ; Write it back to enable IRQs

; Step 8 : Jump to relevant subroutine handler

LDR PC, [PC, R10, lsl #2] ; PC base address points this instruction + 8

NOP ; To index the table by the PC

; Table of handler start addresses

.word IRQ0handler ;IRQ0 BANK0

.word IRQ1handler

.word IRQ2handler

После возврата соответствующего дескриптора подпрограммы **IRQ/FIQ**:

1. Отключите **IRQ/FIQ** на стороне **ARM**.

2. Восстановите поле **INTC\_THRESHOLD PRIORITYTHRESHOLD**.

3. Восстановите регистры критического контекста **ARM**.

В следующем примере кода показаны три предыдущих шага:

***ВНИМАНИЕ***

***Код представляет собой ассемблерный код, совместимый с ARM архитектуры V6 и V7. Этот код разработан для набора инструментов CODE COMPOSER STUDIO. Это черновая версия, тестировалась только на emulated environment.***

IRQ\_ISR\_end:

; Step 1 : Read-modify-write the CPSR to disable IRQs/FIQs at ARM side

MRS R0, CPSR ; Read the CPSR

ORR R0, R0, #0x80/0x40 ; Set the I/F bit

MSR CPSR, R0 ; Write it back to disable IRQs

; Step 2 : Restore the INTC\_THRESHOLD register from R12

LDR R0, INTC\_THRESHOLD\_ADDR

STR R12, [R0]

; Step 3 : Restore critical context

MSR SPSR, R11 ; Restore the SPSR from R11

LDMFD SP!, {R0-R12, LR} ; Restore working registers and Link register

; Return after handling the interrupt

SUBS PC, LR, #4

Рисунок 6-3 показывает вложенную последовательность обработки IRQ/FIQ из периферийного модуля исходного устройства к прерыванию основной программы.

**Рисунок 6-3. Вложенная последовательность обработки IRQ/FIQ**