
ARM Exception Handling and Vectored Interrupt Controller

1

Exceptions

An exception arises whenever the normal flow of program has to be halted temporarily

- due to the occurrence of an event that needs the immediate attention of the processor

Can be of two general types:

- Interrupts due to the occurrence of events that are usually asynchronous in nature, benign, and anticipated (e.g., timer, push button)
- Occurrences of unexpected events that may be problematic (e.g., memory access error), but can also be mitigated by proper software design

2

ARM Exceptions

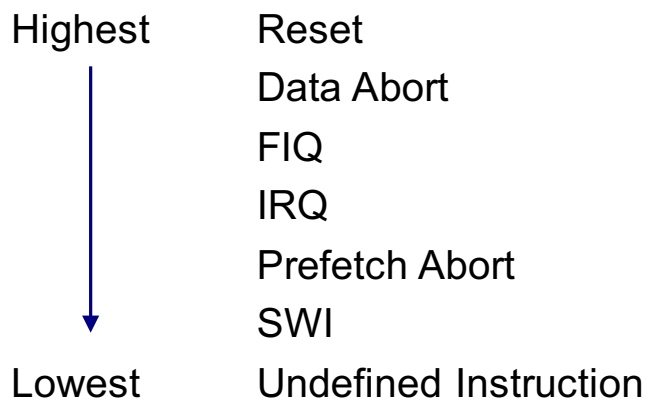
ARM defines seven possible types of exceptions that can occur:

- Reset
- Interrupt request IRQ
- Fast interrupt request FIQ
- Software interrupt SWI
- Data abort
- Prefetch abort
- Undefined instruction

3

Exception Priority

Exceptions are prioritized in the event where multiple exceptions occur simultaneously.



4

Reset Exception

The reset exception is most commonly encountered during the application of power to the system.

- But may also be caused by manual application of the reset signal to the processor pin.

The ARM® processor puts the value 0x00000000 on its address bus to fetch the instruction code from this memory location. This is called the **RESET VECTOR**

- Typically, a branch instruction is found in this location and takes the processor to the boot-up routine.

Reset Exception (cont'd)

Any program that was running prior to occurrence of the Reset exception will not be resumed.

- Hence, no saving of the register contents is needed.

On reset, processor operates in the 32-bit ARM state and Supervisor mode

- allow the initialization of the processor hardware

Data Abort

The Data Abort exception is typically issued by the system memory controller in response to an invalid data access:

- accessing a reserved area or a non-existing page
- executing an unaligned memory access
- writing to the ROM address space

Can be of the following:

- System origin: To support the virtual memory system
- Software origin: Improper C-structures, stray pointer, code ported from a different processor architecture.
- Hardware origin: Improper memory system design, failure of marginal hardware design or component.

7

Prefetch Abort

The Prefetch Abort exception occurs when the processor tries to fetch an instruction from an invalid address.

For a system with the memory management unit (MMU), this could be just a page fault.

- MMU will perform a page swap and the processor can retry the access.
- i.e. a virtual memory system design

For a system without MMU, this should be considered a fatal error since it cannot continue with no instruction.

8

Interrupt Request

The ARM core provides two interrupt lines that when triggered, causes (hardware) interrupts to occur

1.nFIQ: fast interrupt request

2.nIRQ: normal lower priority interrupt request

These interrupts are usually of a non-error origin, most likely issued by a peripheral that requests for immediate attention. Examples include:

- network port receiving a data
- timer expired
- external signal source (like a push button)

nIRQ is used for a non critical event and of a lower priority

9

SWI

A software interrupt (instead of hardware based) that is generated by a program when executing the instruction "SWI".

- An intention interrupt that is associated with a specific routine number.

Example:

Allow a user program to "call" the OS through System Call.

- When SWI is executed, the processor changes from the User mode to the Supervisor mode (and disables IRQ interrupts).
- The exception handler then executes the task based on the SWI number.

10

Undefined Instruction

The undefined code can be of genuine error (e.g. accidental execution of data), or intentional design (e.g. to extend the instruction set).

Example:

Floating-point instructions not supported by the ARM core will trigger an undefined exception when encountered.

- An exception handler can emulate the floating-point operation by executing a program routine and returning the result to the main code.

11

Fast Interrupts Request

The FIQ has higher handling priority than the IRQ and is designed to have much reduced interrupt latency.

This is achieved by:

1) assigning the vector address of FIQ to be at the top of the exceptions' vector table.

FIQ exception handler can be placed at the vector address to remove the need to perform the branch.

2) providing more banked registers (R8_FIQ to R12_FIQ) to further reduce the overhead that would otherwise be spent saving the content of the registers.

12

Exception Vector Table

Each exception will cause the processor to branch to a specific memory location – known collectively as the exception vector table.

<u>Exception</u>		<u>Vector Address</u>
Reset	-	0x00
Undefined Instruction	-	0x04
SWI (or SVC)	-	0x08
Prefetch Abort	-	0x0C
Data Abort	-	0x10
(Reserved)	-	0x14
IRQ	-	0x18
FIQ	-	0x1C


13

Banked Registers

System and User	FIQ	Supervisor	Abort	IRQ	Undefined
r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7
r8	r8_fiq	r8	r8	r8	r8
r9	r9_fiq	r9	r9	r9	r9
r10	r10_fiq	r10	r10	r10	r10
r11	r11_fiq	r11	r11	r11	r11
r12	r12_fiq	r12	r12	r12	r12
r13	r13_fiq	r13_svc	r13_abt	r13_irq	r13_und
r14	r14_fiq	r14_svc	r14_abt	r14_irq	r14_und
r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)

ARM-state program status registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

 = banked register

14

Exception Entrance

When an exception occurs, the processor will automatically perform a sequence of events before jumping to the exception vector address

- Copy CPSR to SPSR_<new-mode>
 - Set appropriate bits in CPSR
 - Change to a new exception mode (bit 0 to bit 4)
 - Switch to the ARM state (bit 5)
 - Disable the IRQ interrupt
 - Disable FIQ if new-mode = Reset and FIQ
 - Store the return address in LR_<new-mode>
 - Change the PC to the appropriate vector address
- The processor then jumps to the vector address.

15

Exception Exit

When the handling routine completes its execution, the processor will typically resume with the original main program

- provided the exception is not caused by an irrecoverable system error (e.g., hardware failure).

The processor can exit the exception two ways:

- use of a data-processing instruction with the S-bit set, and the PC as the destination
- use of the Load Multiple with Restore CPSR instruction (LDM)

16

Exception Exit

Just before exiting the handler, the processor will

- restore the CPSR with value saved in SPSR_<new-mode>
- restore the PC with value save in LR_<new-mode>

Depending on the exception type, the processor then returns to either

- the instruction that caused the exception
- or
- its next instruction

17

Vector Table Instruction

Because there are only 32 bits of memory space in between the vector address

- it is not possible to have the actual handler code residing in the vector table
- except for FIQ, which is located at the top of the vector table

Hence, only one 32-bit instruction code is allowed in each vector location, which can be of the following type

- BRANCH: jump to the exception handler (located within the 32 MB range)
- LDR PC: nIRQ's auto-vectoring to the ISR

18

Exception Handlers

When an exception occurs, the processor will branch to its vector address, and eventually proceed to its exception handler

- the actual code that is responsible to perform the necessary operation for the particular exception

The handler will use its own set of banked registers

- and save the content of the other shared (non-banked) registers on the stack

Depending on the exception type, the handler

- could be short and simple, or long and complicated
- typically returns to the main code eventually
- completely bypass using 'auto-vectoring'

19

Reset Handler

This is typical due to power-up reset, and hence will perform the various initializations as required.

Example:

- Set up the vector table
- Initialize the SP
- Initialize the SDRAM controller
- Initialize all critical registers and peripheral devices
- Enable interrupts
- Change the processor state and mode

20

Undefined Instruction Handler

An undefined instruction handler is most commonly used to further 'extend' the instruction set of the processor.

Example:

The undefined instruction could be that intended for a coprocessor that is not integrated or functioning.

- The handler can execute the instruction in software, hence emulating the hardware.

21

Abort Handler

When the abort exception is encountered, the handler can either

- try to "fix" the error if possible, and return to re-execute the instruction that caused the exception (E.g. for page fault, MMU could swap in the correct page and the processor will retry the access)

or

- report the error and stop further program execution

22

SWI Handler

Software interrupt is commonly used by a program to make a system call to the operating system.

- i.e. to request for service identified by the associated service number
- The processor will be forced to enter the Supervisor mode

The handler must extract the service number from the SWI instruction.

- Use the effective address to calculate through the LR (which contains the subsequent address following the SWI code in the main program)

23

nFIQ Interrupt Handler

Fast Interrupt will disable the normal IRQ

- its handler should be placed immediately at the FIQ vector table
- allows very fast entry to the handler

Seven banked registers are also made available in the FIQ_mode

- the handler only needs to save the other general purpose register as needed

24

nIRQ Interrupt Handler

- The most commonly encountered type of exception
- Typically triggered by peripherals and external signal sources
- The handler needs to further identify the source of the interrupt
 - before it can execute the specific Interrupt Service Routine (ISR)
- For a system that uses
 - chained-interrupt, the processor polls each of the interrupt source
 - standard interrupt controller, the processor checks a device bitmap register in the controller
- But it is more common to use a **Vector Interrupt Controller** in a typical ARM system

25

Vector Interrupt Controller

Vector interrupt controllers can further perform the following:

- prioritize the multiple interrupts sources
 - enable nested interrupt implementation
- directly supply the address of the service routine (ISR) for the specific interrupt directly to the processor
 - typically through a specific register location

26

Case Study: AT91RM9200 Advanced Interrupt (AIC)

27

Sharing of Interrupt Signals

ARM processor has defined seven exception vectors

- but only two are for interrupt signals: nFIQ and nIRQ
- have to be shared by peripherals and external sources that need to interrupt the processor

In AT91RM9200, sharing is supported through a vector interrupt controller

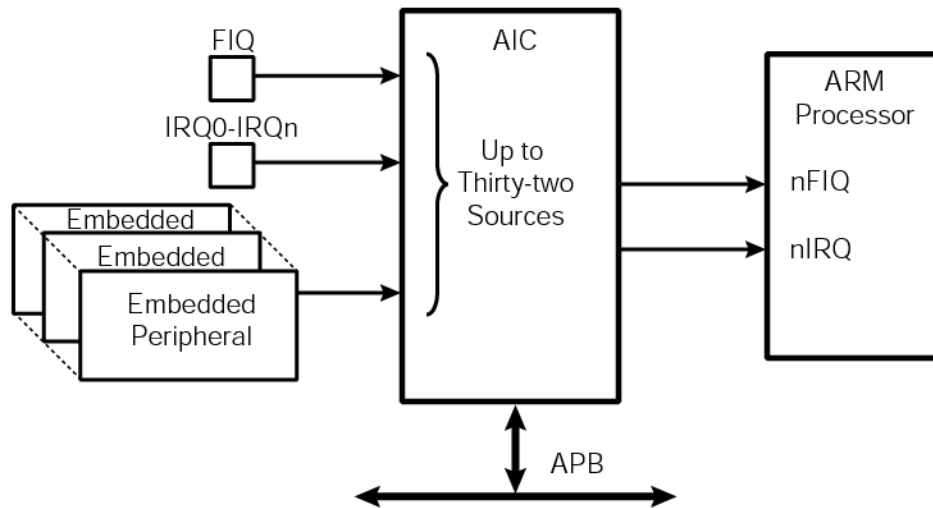
- Advanced Interrupt Controller (AIC)

AIC identifies and supplies the (vector) address of the ISR to the processor upon the assertion of an interrupt

- supports up to 32 individually maskable interrupt signals

28

AIC Block Diagram



29

Interrupt Lines Connection

In AT91RM9200

- nFIQ is not shared
 - used by the FIQ input only
- nIRQ is shared by 31 interrupt sources
 - 24 for embedded peripherals (e.g., Timer/Counter, USART)
 - Seven for external interrupt sources

Each source is identified by a Peripheral ID

- that matches with the corresponding bit position in the AIC 32-bit command registers

Example: Setting bit 6 of the AIC's Interrupt Enable command register is equivalent to enabling the interrupt for peripheral #6.

30

AT91RM9200's Peripheral ID

The following table list the various peripheral IDs defined for AT91RM9200

Peripheral ID	Peripheral Mnemonic	Peripheral Name	External Interrupt
0	AIC	Advanced Interrupt Controller	FIQ
1	SYSIRQ	System Timer, WDT	
2–5	PIOA–PIOD	Parallel I/O Controller A to D	
6–9	US0–US3	USART 0 to 3	
10	MCI	Multimedia Card Interface	
11	UDP	USB Device Port	

31

Peripheral ID (cont'd)

Peripheral ID	Peripheral Mnemonic	Peripheral Name	External Interrupt
12	TWI	Two-Wire Interface	
13	SPI	Serial Peripheral Interface	
14–16	SSC0–SSC2	Synchronous Serial Controller 0 to 2	
17–22	TC0 –TC5	Timer/Counter 0 to 5	
23	UHP	USB Host Port	
24	EMAC	Ethernet MAC	
25–31	AIC	Advanced Interrupt Controller	IRQ0–IRQ6

32

AIC Basic Operation

The AIC is preprogrammed with the address of each enabled interrupt source.

- stored in the corresponding source Vector Register (AIC_SVR)

On receiving an interrupt from one of the peripheral:

- The AIC will assert the corresponding interrupt line of the ARM processor:
 - nFIQ for peripheral #0
 - nIRQ for peripherals #1 to #31
 - The AIC also copies the ISR address of the asserted interrupt (in the AIC_SVR) to its Interrupt Vector Register (AIC_IVR), located at the fixed location 0xFFFFF100.
- nFIQ or nIRQ exception handler then
- reads AIC_IVR to retrieve the address of the ISR
 - jumps to the ISR location and begins execution

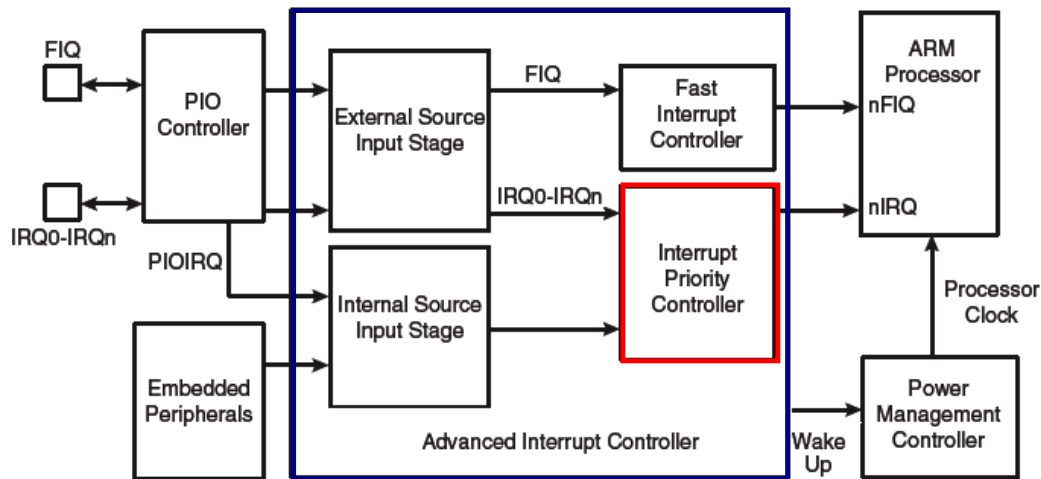
33

Prioritization Support

- AIC contains an internal interrupt priority controller to support interrupt prioritization
 - handles the priority of the interrupt sources 1 to 31 (nFIQ has no priority setting since it is not shared)
 - up to eight different programmable priority levels
- When the nIRQ is asserted and the processor is reading AIC_IVR
 - the priority controller checks all pending interrupts, and select the one with highest priority to supply the ISR address
- For equal priority pending interrupts
 - the peripheral with the lowest ID is selected first

34

AIC Interrupt Priority Controller



35

Nested Interrupt Support

The AIC priority controller also supports interrupt nesting

- allows higher priority interrupt to be handled during the service of lower priority interrupt

This requires the ISR of the lower interrupt to re-enable the nIRQ interrupt before it starts execution.

When a higher priority interrupt occurs

- the current interrupt number and its priority level are saved in an embedded hardware stack

They are restored when the higher priority interrupt servicing is finished

- indicated by the writing of the AIC's End of Interrupt Command Register (AIC_EOICR) by the exiting ISR

36

AIC Initialization

Operation of AIC involves the initializations of the relevant registers

1. AIC Source Mode Register (AIC_SMRx)

- 32 of them; one for each peripheral.
- Use for setting the interrupt priority level and active state (high/low level and +ve/–ve edge sensitive)

2. AIC Source Vector Register (AIC_SVRx)

- 31 of them; one for each peripheral.
- Store the (vector) address of the ISR of each peripheral.
- nFIQ has its own vector register: AIC_FVR

37

AIC Initialization (cont'd)

3. AIC Interrupt Enable Command Register (AIC_IECR) and AIC Interrupt Disable Command Register (AIC_IDCR)

- Enable and Disable of individual interrupt source to support the maskable interrupt

4. AIC Interrupt Clear Command Register (AIC_ICCR) and AIC Interrupt Set Command Register (AIC_ISCR)

- Used only for edge-triggered interrupt
- After an edge-triggered interrupt has occurred, clearing is needed to reset the hardware circuit and reactivate the interrupt
- AIC performs an automatic clear function when the AIC_IVR is read by the processor

38

Interrupt Auto-Vectoring

A branch in one single instruction to the ISR of the interrupting source

- completely bypasses the nIRQ exception handler

Operation principle:

- Put a 'LDR PC' instruction at the nIRQ vector address, to directly load the Program Counter with the AIC_IVR content, which cause a direct jump to the ISR in next clock cycle.
- nIRQ vector address = 0x00000018

39

Interrupt Auto-Vectoring (cont'd)

A branch in one single instruction to the ISR of the interrupting source

- completely bypasses the nIRQ exception handler

Operation principle:

Put a 'LDR PC' instruction at the nIRQ vector address

- which directly loads the Program Counter with the AIC_IVR content
- and causes a direct jump to the ISR in the next clock cycle.

40

Interrupt Auto-Vectoring Calculation

nIRQ vector address = 0x0000 0018

AIC_IVR address = 0xFFFF F100

Instruction to use: `LDR PC, [PC, #offset]`

Offset calculation: Include processor pipeline (+0x08)

$$(0x00000018 + 0x08) + \text{\#offset} = 0xFFFFF100$$

$$(0x00000020) + \text{\#offset} = 0xFFFFF100$$

$$\Rightarrow \text{\#offset} = 0xFFFFF30$$

But a 32-bit number exceeds the 12-bit operand storage size of the LDR instruction. (Recall that ARM uses fixed length 32-bit instructions when in the ARM state)

41

Interrupt Auto-Vectoring

0xFFFFF100 can be alternatively derived from

$$0x00000020 - 0xF20 = 0xFFFFF100$$

$$\Rightarrow \text{\#offset} = -0xF20$$

Instruction to be placed at 0x00000018:

`LDR PC, [PC, #-0xF20]`

Auto-vectoring is usually not used for OS-based system

- The OS's exception handler usually first check the source of the interrupt before deciding whether to executing the ISR

42

Case Study: AT91RM9200 Memory Remapping

43

AT91RM9200 Booting

ARM processor

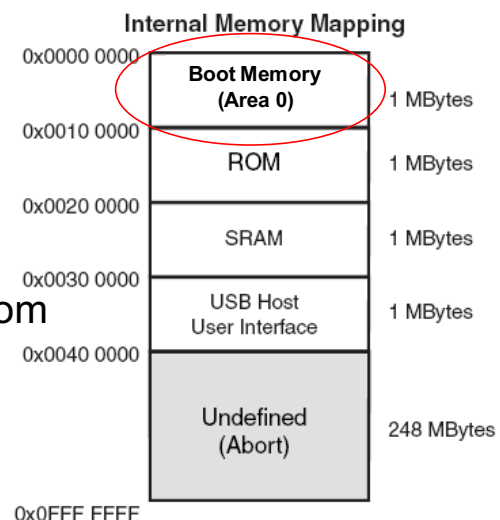
- always boots up at memory location 0x00000000
- the first 32 bytes contain the exception vectors of the ARM processor

So where do these values originate from?

- depend on the boot up condition

For the AT91RM9200, it can be from

- external (Flash) memory
- internal ROM
- internal SRAM



44

Boot Mode Selection

Booting up of the AT91RM9200 processor is

- determined by the Boot Mode Select (BMS) option
- selection is made by the BMS pin sampled at Reset
- BMS = 1
Boot program in on-chip ROM
- BMS = 0

16-bit non-volatile memory (e.g., parallel Flash)
connected to external chip select zero (NCS0)

(On the ARM9 board, these are done through the JP1 setting)

45

Program Execution

While it is possible to run a program from external Flash memory directly

- it is more common to copy the program code to the volatile RAM (SDRAM and SRAM)
- which is faster, and consumes less power (with internal SRAM)

However, memory location 0x00000000 is needed for the exception vector address

- is occupied by the Flash memory

46

Memory Remap

We need to 'remove' the Flash memory from its boot-up address space

- how is this possible without the use of hardware circuitry?

Use the memory remapping concept:

- switch the Flash memory out of the boot-up address space
- swap in the SRAM memory to the vector address space

This is performed through software instruction.

47

AT91RM9200 Remapping

Before the execution of the Remap command, immediately after power-up reset, the internal (boot) memory Area 0 is mapped to

- external Flash memory (for BMS = 0) or on-chip ROM (for BMS = 1)

After the execution of the Remap command (@ MC_RCR)

- internal SRAM (16 KB for AT91RM9200)

BMS State	Before Remap		After Remap
	1	0	X
Internal Memory Area 0	Internal ROM	External Memory Area 0	Internal SRAM

Before the remap, the SRAM must first be filled with the proper exception vector address (see the laboratory exercise on IRQ).

48