Exception and Interrupt Handling in ARM

Seminar Course:
Architectures and Design Methods for Embedded Systems

Author: Ahmed Fathy Abdelrazek (Infotech Master Student)
Advisor: Dominik Lücke
Contents

- Introducing ARM
- Exceptions
- Interrupts
- Interrupt handling schemes
- Summary
Introducing ARM

- **Modes of operation**

  - ARM processor has 7 modes of operation.
  - Switching between modes can be done manually through modifying the mode bits in the CPSR register.
  - Most application programs execute in user mode
  - Non user modes (called privileged modes) are entered to serve interrupts or exceptions
  - The system mode is special mode for accessing protected resources. It don’t use registers used by exception handlers, so it can’t be corrupted by any exception handler error!!!
## Introducing ARM

### Modes of operation

<table>
<thead>
<tr>
<th>Processor Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User (usr)</td>
<td>Normal program execution mode</td>
</tr>
<tr>
<td>FIQ (fiq)</td>
<td>Fast data processing mode</td>
</tr>
<tr>
<td>IRQ (irq)</td>
<td>For general purpose interrupts</td>
</tr>
<tr>
<td>Supervisor (svc)</td>
<td>A protected mode for the operating system</td>
</tr>
<tr>
<td>Abort (abt)</td>
<td>When data or instruction fetch is aborted</td>
</tr>
<tr>
<td>Undefined (und)</td>
<td>For undefined instructions</td>
</tr>
<tr>
<td>System (sys)</td>
<td>Privileged mode for OS Tasks</td>
</tr>
</tbody>
</table>

Switching between these modes requires saving/loading register values.
Introducing ARM

- **ARM register set**
  - ARM processor has 37 32-bit registers.
  - 31 registers are general purpose registers.
  - 6 registers are control registers.
  - Registers are named from R0 to R16 with some registers banked in different modes.
  - R13 is the stack pointer *SP* (Banked).
  - R14 is subroutine link register *LR* (Banked).
  - R15 is program counter *PC*.
  - R16 is current program status register *CPSR* (Banked).
Introducing ARM

- **ARM register set**

More banked registers, so context switching is faster
Contents

- Introducing ARM
- Exceptions
- Interrupts
- Interrupt handling schemes
- Summary
Exceptions

What is an exception?
An exception is any condition that needs to halt normal execution of the instructions

Examples
• Resetting ARM core
• Failure of fetching instructions
• HWI
• SWI
Exceptions

Exceptions and modes

Each exception causes the ARM core to enter a specific mode.

<table>
<thead>
<tr>
<th>Exception</th>
<th>Mode</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast Interrupt Request</td>
<td>FIQ</td>
<td>Fast interrupt handling</td>
</tr>
<tr>
<td>Interrupt Request</td>
<td>IRQ</td>
<td>Normal interrupt handling</td>
</tr>
<tr>
<td>SWI and RESET</td>
<td>SVC</td>
<td>Protected mode for OS</td>
</tr>
<tr>
<td>Pre-fetch or data abort</td>
<td>ABT</td>
<td>Memory protection handling</td>
</tr>
<tr>
<td>Undefined Instruction</td>
<td>UND</td>
<td>SW emulation of HW coprocessors</td>
</tr>
</tbody>
</table>
Exceptions

- Vector table

It is a table of addresses that the ARM core branches to when an exception is raised and there is always branching instructions that direct the core to the ISR.

At this place in memory, we find a branching instruction

\texttt{ldr \ pc, [pc, \_IRQ\_handler\_offset]}
Exceptions

- Exception priorities

  decide which of the currently raised exceptions is more important

  Decide if the exception handler itself can be interrupted during execution or not?

Both are caused by an instruction entering the execution stage of the ARM instruction pipeline

<table>
<thead>
<tr>
<th>Exception</th>
<th>Priority</th>
<th>I bit</th>
<th>F bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Data Abort</td>
<td>2</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>FIQ</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>IRQ</td>
<td>4</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>Prefetch abort</td>
<td>5</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>SWI</td>
<td>6</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>Undefined instruction</td>
<td>6</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>
Exceptions

- Link Register Offset

This register is used to return the PC to the appropriate place in the interrupted task since this is not always the old PC value. It is modified depending on the type of exception.

The PC has advanced beyond the instruction causing the exception. Upon exit of the prefetch abort exception handler, software must re-load the PC back one instruction from the PC saved at the time of the exception.

<table>
<thead>
<tr>
<th>Exception</th>
<th>Returning Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>None</td>
</tr>
<tr>
<td>Data Abort</td>
<td>LR-8</td>
</tr>
<tr>
<td>FIQ, IRQ, prefetch Abort</td>
<td>LR-4</td>
</tr>
<tr>
<td>SWI, Undefined Instruction</td>
<td>LR</td>
</tr>
</tbody>
</table>
Exceptions

- **Entering exception handler**
  1. Save the address of the next instruction in the appropriate Link Register \( LR \).
  2. Copy \( CPSR \) to the \( SPSR \) of new mode.
  3. Change the mode by modifying bits in \( CPSR \).
  4. Fetch next instruction from the vector table.

- **Leaving exception handler**
  1. Move the Link Register \( LR \) (minus an offset) to the \( PC \).
  2. Copy \( SPSR \) back to \( CPSR \), this will automatically changes the mode back to the previous one.
  3. Clear the interrupt disable flags (if they were set).
Contents

- Introducing ARM
- Exceptions
- Interrupts
- Interrupt handling schemes
- Summary
Interrupts

- Assigning interrupts

It is up to the system designer who can decide which HW peripheral can produce which interrupt.

**But** system designers have adopted a standard design for assigning interrupts:

- SWI are used to call privileged OS routines.
- IRQ are assigned to general purpose interrupts like periodic timers.
- FIQ is reserved for one single interrupt source that requires fast response time.
Interrupts

Interrupt latency

It is the interval of time from an external interrupt signal being raised to the first fetch of an instruction of the ISR of the raised interrupt signal.

System architects try to achieve two main goals:

• To handle multiple interrupts simultaneously.
• To minimize the interrupt latency.

And this can be done by 2 methods:

• allow nested interrupt handling
• give priorities to different interrupt sources
Interrupts

Enabling and disabling Interrupt

This is done by modifying the **CPSR**, this is done using only 3 ARM instruction:

- **MRS** To read **CPSR**
- **MSR** To store in **CPSR**
- **BIC** Bit clear instruction
- **ORR** OR instruction

<table>
<thead>
<tr>
<th>Enabling an IRQ/FIQ Interrupt</th>
<th>Disabling an IRQ/FIQ Interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRS r1, cpsr</td>
<td>MRS r1, cpsr</td>
</tr>
<tr>
<td>BIC r1, r1, #0x80/0x40</td>
<td>ORR r1, r1, #0x80/0x40</td>
</tr>
<tr>
<td>MSR cpsr_c, r1</td>
<td>MSR cpsr_c, r1</td>
</tr>
</tbody>
</table>
Interrupts

Interrupt stack

Stacks are needed extensively for context switching between different modes when interrupts are raised.

The design of the exception stack depends on two factors:
• OS Requirements.
• Target hardware.

A good stack design tries to avoid stack overflow because it cause instability in embedded systems.
Interrupts

- Interrupt stack

Two design decisions need to be made for the stacks:
- The location
- The size

The benefit of this layout is that the vector table remains untouched if a stack overflow occurred!!
Contents

- Introducing ARM
- Exceptions
- Interrupts
- Interrupt handling schemes
- Summary
Non-nested interrupt handling scheme

- This is the simplest interrupt handler.
- Interrupts are disabled until control is returned back to the interrupted task.
- One interrupt can be served at a time.
- Not suitable for complex embedded systems.
Interrupt handling schemes

- **Nested interrupt handling scheme (1)**

  - Handling more than one interrupt at a time is possible by enabling interrupts before fully serving the current interrupt.

  - Latency is improved.

  - System is more complex.

  - No difference between interrupts by priorities, so normal interrupts can block critical interrupts.
Interrupt handling schemes

- Nested interrupt handling scheme (2)

  - The handler tests a flag that is updated by the ISR

  - Re-enabling interrupts requires switching out of current interrupt mode to either SVC or system mode.

  - Context switch involves emptying the IRQ stack into reserved blocks of memory on SVC stack called stack frames.
Interrupt handling schemes

- Prioritized simple interrupt handling
  - associate a priority level with a particular interrupt source.
  - Handling prioritization can be done by means of software or hardware.
  - When an interrupt signal is raised, a fixed amount of comparisons is done.
    - So the interrupt latency is deterministic.
    - But this could be considered a disadvantage!!
Interrupt handling schemes

Other schemes

There are some other schemes, which are actually modifications to the previous schemes as follows:

• “Re-entrant interrupt handler”: re-enable interrupts earlier and support priorities, so the latency is reduced.

• “Prioritized standard interrupt handler”: arranges priorities in a special way to reduce the time needed to decide on which interrupt will be handled.

• “Prioritized grouped interrupt handler”: groups some interrupts into subset which has a priority level, this is good for large amount of interrupt sources.
Contents

- Introducing ARM
- Exceptions
- Interrupts
- Interrupt handling schemes
- Summary
Summary

- Availability of different modes of operation in ARM helps in exception handling in a structured way.

- Context switching is one of the main issues affecting interrupt latency, and this is resolved in ARM FIQ mode by increasing number of banked registers.

- We can’t decide on one interrupt handling scheme to be used as a standard in all systems, it depends on the nature of the system:
  - What type of interrupts are there?
  - How many interrupts are there?
Thanks For listening,
Waiting for questions