Embedded Systems

Ch 9
Interrupt and Timer

Byung Kook Kim
Dept of EECS
Korea Advanced Institute of Science and Technology

Overview

- *1. Interrupts and Exceptions*
- *2. Interrupt Controller in Xscale*
- *3. Exceptions in ARM*
- *4. Real-Time Clock in Xscale*
- *5. OS Timer in Xscale*

1. Interrupts and Exceptions

- **What is an interrupt?**
  - Analogy in everyday life:
    - Phone call while reading a book
  - *An event either from an internal or external source where*
    - *a processor will stop its current processing thread,*
    - *switch to a different instruction sequence, and then*
    - *resume its current processing.*

Interrupts and Exceptions (II)

- Notes on interrupt
  - The most important aspect of any embedded system design.
    - Potential for many problems when debugging a system.
  - Affect the overall design and structure of the system.
    - Allows the designer to split software into two types: background and foreground.
    - Can be developed in modular fashion.
  - *Interrupt service routine (ISR)*
    - Data transfer code associated with device hardware which generates the interrupt.
    - Background processes.

Interrupts and Exceptions (III)

- **Interrupt sources**
  - Internal interrupts
    - Generated by on-chip peripherals
    - Connection for interrupt is already made
  - External interrupts
    - Generated by external peripherals of the processor
    - Provided through external pins of the processor, which can be driven by external peripherals.
  - Exceptions
    - Extended to cover any event that causes the processor to change to a service routines.
    - Normally coupled with a change in processor's mode.
  - Software interrupts
    - Provide an interface to other mode (privileged mode) or software (such as operating system).
    - Parameter passing via registers.
  - Non-maskable interrupt (NMI)
    - External interrupt that cannot be masked out.
    - Highest priority. (~Fire!)

Interrupts and Exceptions (IV)

- **Recognizing an interrupt**
  - Edge triggered
    - Low-to-high or high-to-low transition
  - Level triggered
    - Low or high level
    - Sampled on regular basis (clock or every instruction)
    - Minimum pulse width may be defined.
  - Maintaining the interrupt
    - Mostly should maintain the interrupt until explicitly serviced and the source told to remove it.
  - Internal queuing
    - Any subsequent interrupts that have the same level will be maintained after the first one has been serviced and its signal removed.

Interrupts and Exceptions (V)

- **The interrupt mechanism**
  - To hold essential data:
    - Stack: MC68000 or 80x86
    - Special internal registers: RISC processors
  - At the instruction boundary, the processor must save certain state information to allow resuming the current process.
    - Copy of condition code register
    - Program counter
    - Return address
    - The register set
  - Get the location of ISR to service the interrupt
    - Vector table in memory (fixed location)
    - Appropriate vector can be supplied by the peripheral or pre-assigned.
  - Starts the code within ISR until it reaches a return from interrupt instruction.
  - Processor reloads the status information and processing continues.

Interrupts and Exceptions (VI)

- **Fast interrupts**
  - Very fast servicing at the expense of several restrictions.
    - Without saving the processor context
  - Ex: DSP56000
    - The external interrupt is synchronized with the processor clock for two successive clock cycles
    - The processor fetches two instructions from the appropriate vector location and executes them.
  - Once completed, the program counter simply carries on as if nothing has happened.
  - Adv:
    - No stack frame building: Fast
  - Disadv:
    - The size of ISR and resource restricted.
    - Usually assign a couple of address registers for the fast interrupt routine.

Interrupts and Exceptions (VII)

- **Interrupt Controllers**
  - Provides a large number of interrupt pins that can be allocated to many external devices.
    - At least eight and higher numbers can be supported by cascading two or more controllers together (**Expandability**)
    - Ex: Intel 8259 Programmable Interrupt Controller
      - IBM PC: Two 8259's cascaded to give 15 interrupt levels.
  - Orders the interrupt pins in a priority level so that high level interrupt will inhibit a lower level interrupt (**Interrupt priority**)
  - May provide registers for each interrupt pin which contain the vector number to be used during an **interrupt acknowledge** cycle.
    - Allow peripherals without vector capability.
  - Can provide **interrupt masking**
    - Allow the system software to decide when an interrupt is allowed.

Interrupts and Exceptions (VIII)

- **Interrupt Latency**
  - *Time taken by the processor from recognition of the interrupt to the start of the ISR.*
    - Defines several aspects of an embedded system with reference to its ability to respond to real-time events.
  - Sources of interrupt latency
    - The time taken to recognize the interrupt
    - The time taken by the CPU to complete the current instruction
      - CISC: Dep. on instruction (Ex: FP divide)
      - RISC: 1 or 2 clocks
    - The time for the CPU to perform a context switch
      - CISC: block store to the stack
      - RISC: simply switching registers (register windowing or shadowing)
    - The time to fetch the interrupt vector
      - Cache miss?
    - The time taken to start the interrupt service routine execution.

Interrupts and Exceptions (IX)

- **Do's and Don'ts**
  - Always expect the unexpected interrupt
    - Always include a generic handler for all unused/unexpected exceptions.
  - Don't expect too much from an interrupt
    - Don't overload the system with too many interrupts or put too much into the ISR.
  - Use handshaking
    - Do not remove the interrupt until explicitly acknowledged.
  - Control resource sharing
    - Variables used in normal program and ISR.
  - Beware false interrupts
    - Noise and other factors
  - Controlling interrupt levels
    - NO priority inversion
  - Controlling stacks
    - Prevent stack from overflowing.

2. Interrupts in Xscale

- **Introduction**
  - All on-chip interrupts are enabled, masked, and routed to the core FIQ or IRQ.
    - Each interrupt is enabled or disabled at the source through an interrupt mask bit.
    - Generally, all interrupt bits in a unit are ORed together and present a single value to the interrupt controller.
    - Each interrupt goes through the Interrupt Controller Mask Register and then the Interrupt Controller Level Register directs the interrupt into either the IRQ or FIQ.
  - If an interrupt is taken, the software may read the Interrupt Controller Pending Register to identify the source.
    - After it identifies the interrupt source, software is responsible for servicing the interrupt and clearing it in the source unit before exiting the service routine.

Interrupts in Xscale (II)

- **The interrupt controller**
  - provides masking capability for all interrupt sources and
  - generates either an FIQ or IRQ processor interrupt.

- **Two-level structure**
  - The first level:
    - identifies the interrupts from all the enabled and unmasked interrupt sources in the Interrupt Controller Mask Register (ICMR).
  - The second level:
    - uses registers contained in the source device (the device generating the first-level interrupt bit).
    - gives additional information about the interrupt
    - used inside the interrupt service routine.
    - Multiple second-level interrupts are OR'ed to produce a first-level interrupt bit.

Interrupts in Xscale (III)

- Registers for first level interrupt control
  - Interrupt Controller Pending Register (ICPR)
    - identifies all the active interrupts within the system
  - Interrupt Controller IRQ Pending Register (ICIP)
    - contains the interrupts from all sources that can generate an IRQ interrupt.
  - Interrupt Controller FIQ Pending Register (ICFP)
    - contains the interrupts from all sources that can generate an FIQ interrupt.
  - The Interrupt Controller Level Register (ICLR)
    - programmed to send interrupts to the ICIP to generate an IRQ or FIQ.

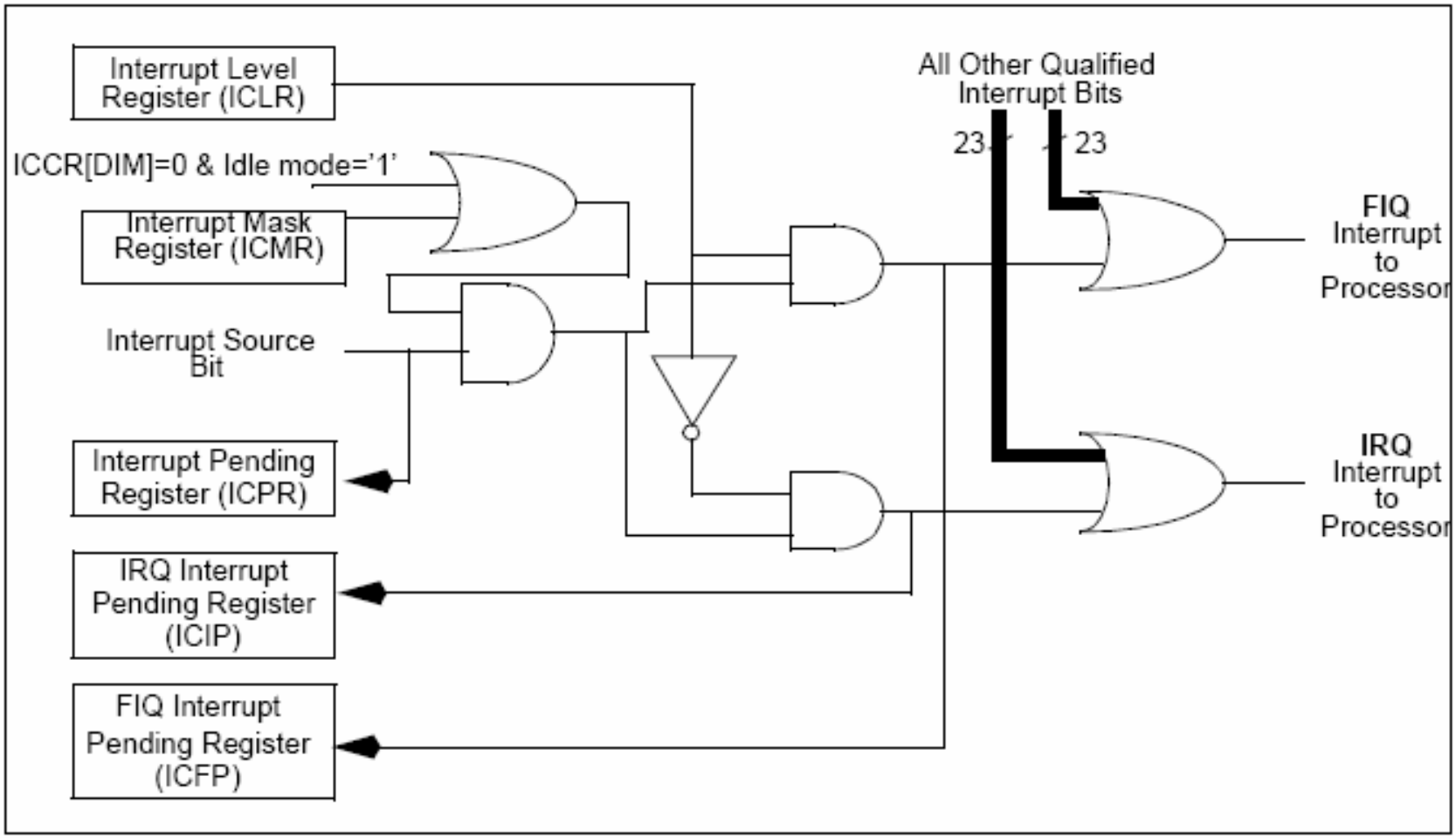
Interrupts in Xscale (IV)

- Interrupt source identification
  - In most cases, the root cause of an interrupt can be determined by reading two register locations:
    - the ICIP for an IRQ interrupt or
    - the ICFP for an FIQ interrupt.
  - Then read the status register within that device to find the exact function requesting service.

- Idle Mask
  - When the ICCR[DIM] bit is zero (Reset state)
    - Interrupt Controller Control Register [Disable Idle Mask]
    - The Interrupt Mask Register is ignored during Idle mode, and all enabled interrupts cause the processor to exit from idle mode.
  - Otherwise (ICCR[DIM]=one)
    - only unmasked interrupts cause the processor to exit from idle mode.

Interrupt in Xscale (V)

- **Interrupt Controller Block Diagram**



Interrupt in Xscale (VI)

- **Interrupt Controller Register Definitions**
  - Interrupt Controller Mask register (ICMR)
    - 0 – Pending interrupt is masked from becoming active (interrupts are NOT sent to CPU or Power Manager).
    - 1 – Pending interrupt is allowed to become active (interrupts are sent to CPU and Power Manager).
  - Interrupt Controller Level register (ICLR)
    - 0 – Interrupt routed to IRQ interrupt input.
    - 1 – Interrupt routed to FIQ interrupt input.
  - Interrupt Controller Control register (ICCR)
    - Disable Idle mask (bit 0):
    - 0 – All enabled interrupts bring the processor out of idle mode.
    - 1 – Only enabled and unmasked (as defined in the ICMR) bring the processor out of idle mode.

Interrupt in Xscale (VII)

- **Interrupt Controller Register Definitions (II)**
  - Interrupt Controller IRQ Pending register (ICIP)
    - IRQ Pending x (where x = 8 through 14 and 17 through 31):
    - 0 – IRQ NOT requested by any enabled source.
    - 1 – IRQ requested by an enabled source.
  - Interrupt Controller FIQ Pending register (ICFP)
    - FIQ Pending x (where x = 8 through 14 and 17 through 31):
    - 0 – FIQ NOT requested by any enabled source.
    - 1 – FIQ requested by an enabled source.

Interrupt in Xscale (VIII)

- **Interrupt Controller Register Definitions (III)**
  - Interrupt Controller Pending register (ICPR)
    - IS31: RTC Alarm Match Register Interrupt Pending
      - 1 – Interrupt pending due to RTC Alarm Match Register.
    - IS30: RTC HZ Clock Tick
    - IS29: OS Timer Match Register 3
    - IS28: OS Timer Match Register 2
    - IS27: OS Timer Match Register 1
    - IS26: OS Timer Match Register 0
    - IS25: DMA Channel Service Request
    - IS24: SSP Service Request
    - IS23: MMC Status/Error Detection
    - IS22: FFUART Transmit/Receive/Erro
    - IS21: BTUART Transmit/Receive/Error
    - IS20: STUART Transmit/Receive/Error
    - IS19: ICP Transmit/Receive/Error
    - IS18: I2C Service Request
    - IS17: LCD Controller Service Request
    - IS16: Network SSP Service Request
    - IS14: AC97
    - IS13: I2S
    - IS12: Performance Monitoring Unit (PMU)
    - IS11: USB Service
    - IS10 GPIO[84:2] Edge Detect
    - IS9: GPIO[1] Edge Detect
    - IS8: GPIO[0] Edge Detect
    - IS7: Hardware UART Service Request Interrupt Pending.

3. Exceptions in ARM

- Operating modes of ARM
  - **User mode**: Most programs
  - **Privileged mode**: Handle exceptions and supervisor calls
- Current operating mode
  - Defined by CPSR[4:0]

CPSR[4:0]	Mode	Use	Registers
10000	User	Normal user code	user
10001	FIQ	Processing fast interrupts	_fiq
10010	IRQ	Processing standard interrupts	_irq
10011	SVC	Processing software interrupts (SWIs)	_svc
10111	Abort	Processing memory faults	_abt
11011	Undef	Handling undefined instruction traps	_und
11111	System	Running privileged operating system tasks	user

Exceptions in ARM (II)

- **Privileged mode**
  - Can only be entered through controlled mechanism
  - Suitable memory protection
    - Fully protected operating system to be built.
  - SPSRs
    - Each privileged mode has associated with Saved Program Status Register (SPSR), except system mode.
    - Used to save the content of CPSR when entering privileged mode.
    - If the privileged mode to be re-entrant, the SPSR must be copied into a general register and saved.

Exceptions in ARM (III)

- **ARM exception groups**
  - Exceptions generated as a direct effect of executing an instruction
    - Software interrupts
    - Undefined instructions (including coprocessor instructions when requested but absent)
    - Prefetch aborts: Memory fault during fetch
  - Exceptions generated as a side-effect of an instruction
    - Data aborts: Memory fault during a load or store data access
  - Exceptions generated externally, unrelated to the instruction flow
    - Reset
    - IRQ (Interrupt Request)
    - FIQ (Fast Interrupt Request)

Exceptions in ARM (IV)

- **Exception entry**
  - ARM completes the current instruction
  - Changes the operating mode to the particular exception
  - Saves the address of the instruction following the exception entry instruction in r14 of the new mode
  - Save the old value of CPSR in the SPSR of the new mode
  - Disables IRQs by setting bit 7 of CPSR. Disables FIQs by setting bit 6 of CPSR (for FIQ exception)
  - Force the PC to begin executing at the relevant vector address.
    - Normally the vector address will contain a branch to the relevant routine.

Exceptions in ARM (V)

- Exception vector address

Exception	Mode	Vector address
Reset	SVC	0x00000000
Undefined instruction	UND	0x00000004
Software interrupt (SWI)	SVC	0x00000008
Prefetch abort (instruction fetch memory fault)	Abort	0x0000000C
Data abort (data access memory fault)	Abort	0x00000010
IRQ (normal interrupt)	IRQ	0x00000018
FIQ (fast interrupt)	FIQ	0x0000001C

- Registers usage
  - Two registers of the privileged mode saves
    - Return address: r14
    - Stack pointer: r13
  - The stack pointer may be used to save other user registers so that they can be used by the exception handler.
  - FIQ mode have additional private registers (r8 to r12) to give better performance (by avoiding the need to save user registers).

Exceptions in ARM (VI)

- **Exception return**
  - Once the exception has been handled, the user task is normally returned
  - Sequence
    - Any modified user registers must be restored from the handler's stack
    - The CPSR must be restored from the appropriate SPSR
    - The PC must be changed back to the relevant instruction address in the user instruction stream.
    - Last two steps should be performed atomically.

4. Real-Time Clock in Xscale

- **Operation**
  - Provides a general-purpose real-time reference for your design.
    - Set to be a 1 Hz output and is utilized as a system time keeper.
    - Alarm feature that enables an interrupt or a wake up event when the RTC output clock increments to a pre-set value.
  - The RTC Counter register (RCNR)
    - initialized to zero after a hardware reset or a watchdog reset.
    - a free running counter that starts incrementing the count value after the deassertion of reset.
    - The counter is incremented one 32kHz cycle after the rising edge of the Hz clock.
      - Since the high phase of the 1 Hz clock is one 32kHz cycle wide, it appears to increment on the falling edge of the 1 Hz clock.
    - Set this counter to the desired value. The value of the counter is unaffected by transitions into and out of Sleep or Idle mode.

Real-Time Clock in Xscale (II)

- Alarm function
  - RTC Alarm register (RTAR)
    - may be programmed with a value that is compared against the RCNR.
    - One 32-kHz cycle after each rising edge of the HZ clock, the counter is incremented and then compared to the RTAR.
    - If the values match, and the enable bit is set, then the RTC Status register (RTSR) alarm match bit (RTSR[AL]) is set.
    - This status bit is also routed to the interrupt controller and may be unmasked in the interrupt controller to generate a processor interrupt.
- The HZ clock
  - generated by dividing one of two selectable clock sources
    - The first source: the output of the 3.6864 MHz crystal oscillator further divided by 112 to approximately 32.914 kHz.
    - The other source: the optional 32.768 kHz crystal oscillator output itself.

5. OS Timer in Xscale

- **Operating System (OS) Timer**
  - The processor contains a 32-bit OS timer that is clocked by the 3.6864 MHz oscillator.
  - The Operating System Count register (OSCR) is a free running up-counter.
  - The OS timer also contains four 32-bit match registers (OSMR3, OSMR2, OSMR1, OSMR0).
  - Developers can read and write to each register. When the value in the OSCR is equal to the value within any of the match registers, and the interrupt enable bit is set, the corresponding bit in the OSSR is set.
  - These bits are also routed to the interrupt controller where they can be programmed to cause an interrupt.
  - OSMR3 also serves as a watchdog match register that resets the processor when a match occurs provided the OS Timer Watchdog Match Enable Register (OWER) is set.

OS Timer (II)

- **Watchdog Timer Operation**
  - The OSMR3 can also be used as a watchdog compare register, enabled by setting OWER[0].
  - When a compare against this register occurs and the watchdog is enabled, reset is applied to the processor and most internal states are cleared.
    - Internal reset is asserted for 256 processor clocks and then removed, allowing the processor to boot.

- Watchdog procedure suggested
  1. The current value of the counter is read.
  2. An offset is then added to the read value. This offset corresponds to the amount of time before the next time-out (care must be taken to account for counter wraparound).
  3. The updated value is written back to OSMR3.
  - The OS code must repeat this procedure periodically before each match occurs. If a match occurs, the OS timer asserts a reset to the processor.

OS Timer (III)

- **OS Timer Register Definitions**
  - **OS Timer Match Register 0-3 (OSMRx)**
    - Compared against the OSCR after every rising edge of the 3.6864 MHz clock.
    - All four registers are identical, except for location.
  - **OS Timer Interrupt Enable Register (OIER)**
    - contains four enable bits that indicate whether a match between one of the match registers and the OS timer counter sets a status bit in the OSSR.
      - E3: Interrupt enable channel 3.
      - 1 – A match between OSMR3 and the OS Timer asserts OSSR[M3].
  - **OS Timer Watchdog Match Enable Register (OWER)**
    - contains a single control bit (bit 0) that enables the watchdog function.
    - can only be cleared by one of the reset functions such as, hardware reset, sleep reset, watchdog reset, and GPIO reset.
      - WME Watchdog Match Enable
      - 1 – OSMR3 match causes a reset of the processor.

OS Timer (IV)

- **OS Timer Register Definitions (II)**
  - **OS Timer Count Register (OSCR)**
    - 32-bit counter that increments on rising edges of the 3.6864 MHz clock.
    - This counter can be read or written at any time.
  - **OS Timer Status Register (OSSR)**
    - contains status bits that indicate a match has occurred between any of the four match registers and the OSCR.
    - These bits are set when the match event occurs (following the rising edge of the 3.6864 MHz clock) and the corresponding interrupt enable bit is set in the OIER.
    - The OSSR bits are cleared by writing a one to the proper bit position.
      - M3 Match status channel 3. If OIER[3] is set then
      - 0 – OSMR[3] has NOT matched the OS timer counter since last being cleared.
      - 1 – OSMR[3] has matched the OS timer counter.

References

- Interrupts and exceptions
  - Steve Heath, “Embedded Systems Design”, Newnes, 2003.
- Interrupt Controller, Real-Time clock, OS timer in Xscale
  - PXA255 Developer’s Manual, http://developer.intel.com.
- Exceptions in ARM
  - Steve Furber, “ARM system-on-chip architecture”, Addison Wesley, 2000.