EE414 Embedded Systems

Ch 6. Interfacing

Part 3/4: Interrupt & Timer

Byung Kook Kim
School of Electrical Engineering
Korea Advanced Institute of Science and Technology

Overview

Part C. Interrupt and Timer

- 6.5 Interrupts in AM3359
- 6.6 Exceptions in ARM
- 6.7 Timers & Counters
- 6.8 Timer & Clock in AM3359

- 6.9 Direct memory access
- 6.10 Arbitration
- 6.11 Multilevel Bus Architectures

6.5 Interrupts in AM3359

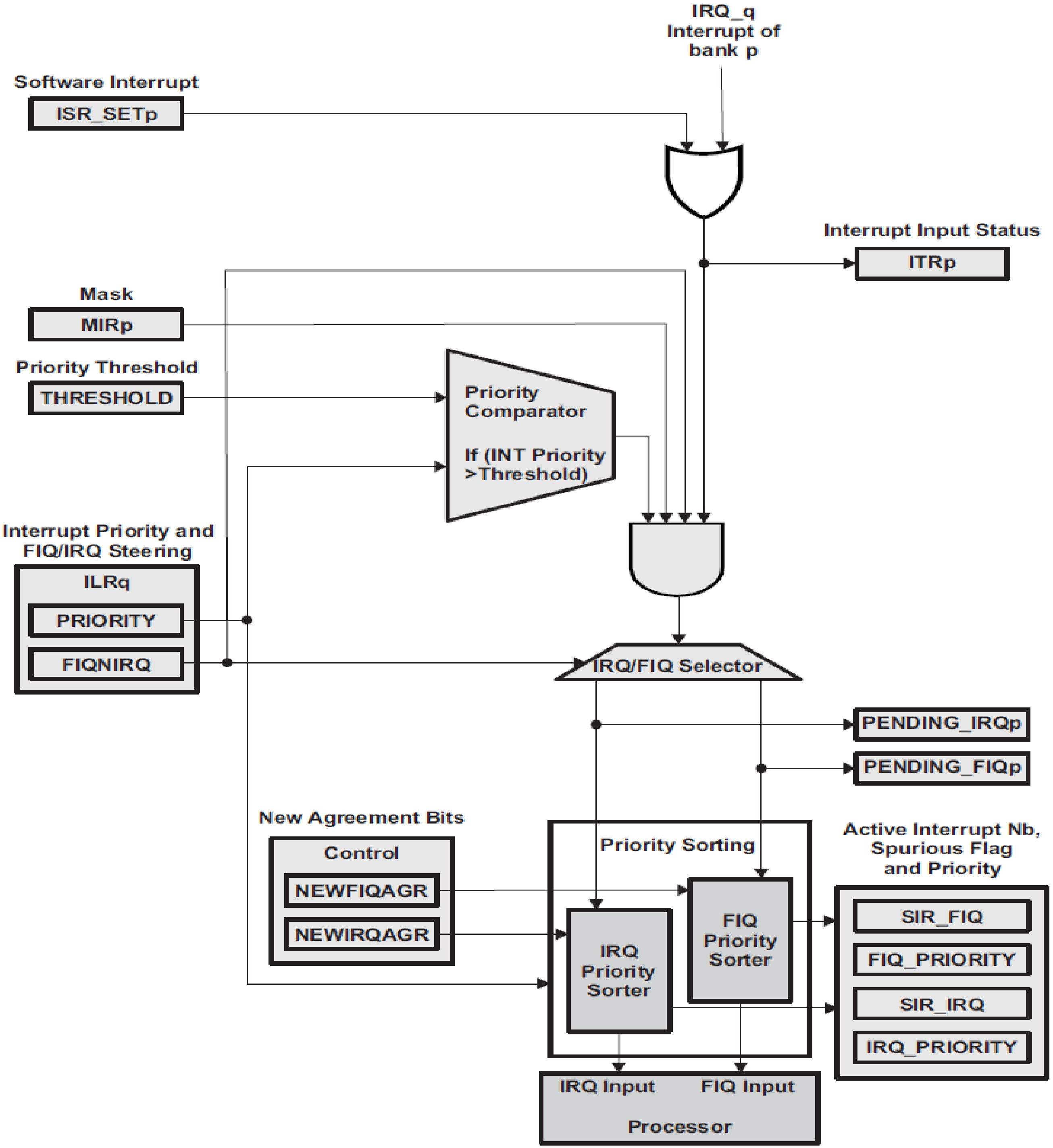
- Overview
  - ARM supports two types of interrupts:
    - Fast interrupt requests (FIQs).
    - Interrupt requests (IRQs).
  - Interrupt sources
    - Internal on-chip peripherals
      - All on-chip interrupts are *enabled*, *masked*, and routed to the core FIQ or IRQ.
    - External I/O devices
      - Interrupt via GPIO

Interrupts in AM3359 (II)

Interrupt Controller [TRM p. 176]

- **The Host ARM Interrupt Controller (AINTC)** is responsible for prioritizing all service requests from the system peripherals and generating either nIRQ or nFIQ to the host.
- The type of the interrupt (nIRQ or nFIQ) and the priority of the interrupt inputs are programmable.
- The AINTC interfaces to the ARM processor via the AXI port through an AXI2OCP bridge and runs at half the processor speed.
- It has the capability to handle up to 128 requests which can be steered/prioritized as A8 nFIQ or nIRQ interrupt requests.
- The general features of the AINTC are:
  - Up to 128 level sensitive interrupts inputs
  - Individual priority for each interrupt input
  - Each interrupt can be steered to nFIQ or nIRQ
  - Independent priority sorting for nFIQ and nIRQ

Interrupt Controller Block Diagram of AM3359

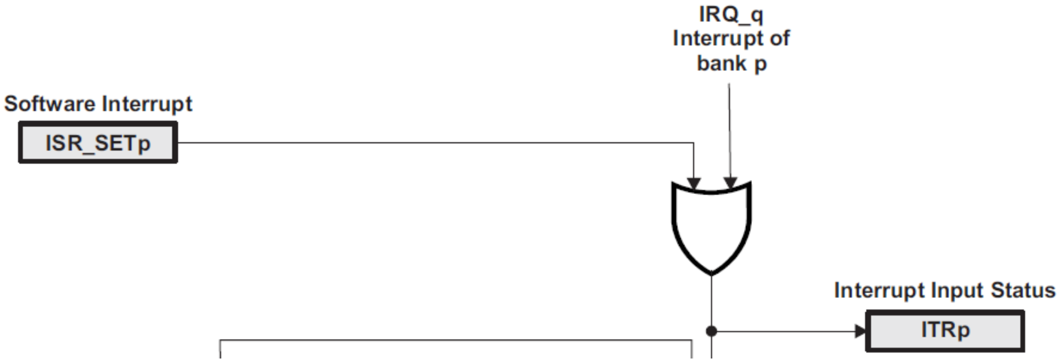


- Mask
- Priority
- Priority sorting

Interrupt Processing in AM3359

Input Selection

- The INTC supports only level-sensitive incoming interrupt detection.
- A peripheral asserting an interrupt maintains it until software has handled the interrupt and instructed the peripheral to deassert the interrupt.

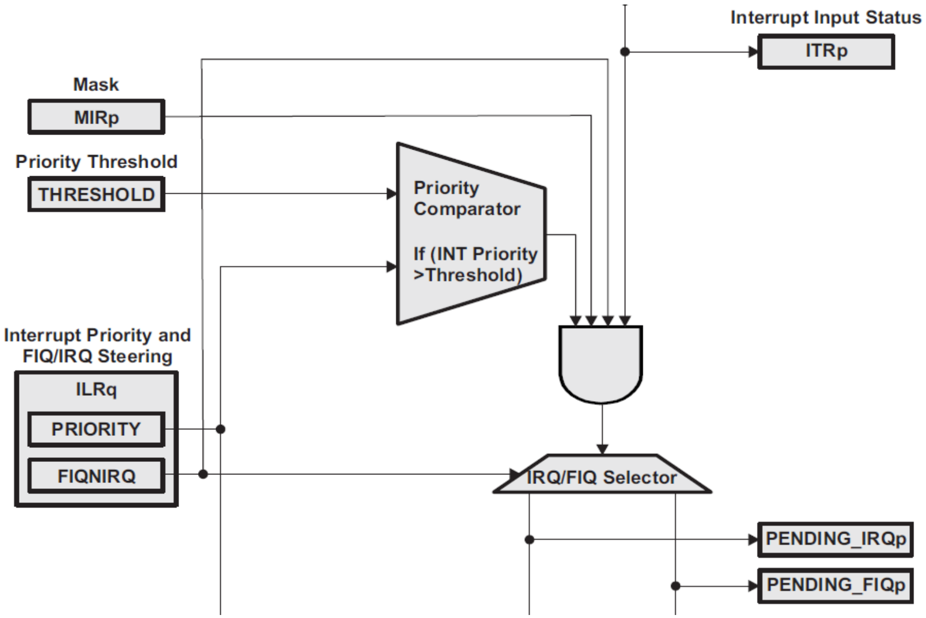


- A software interrupt is generated if the corresponding bit in the MPU_INTC.INTC_ISR_SETn register is set (register bank number: n = [0,1,2,3] for the MPU subsystem INTC, 128 incoming interrupt lines are supported).
- The software interrupt clears when the corresponding bit in the MPU_INTC.INTC_ISR_CLEARn register is written. Typical use of this feature is software debugging.

Interrupt Processing in AM3359 (II)

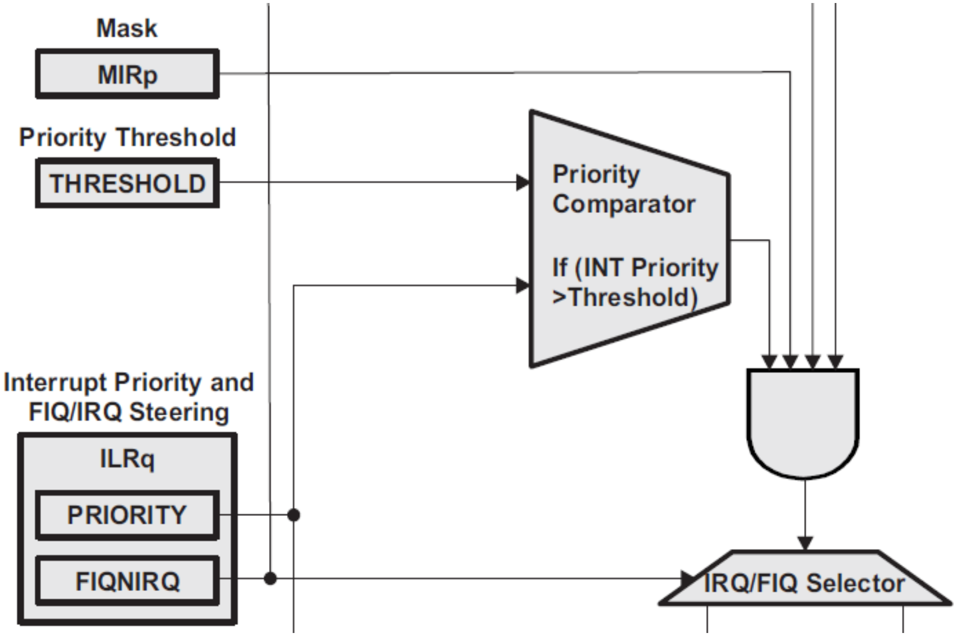
Individual Masking

- Detection of interrupts on each incoming interrupt line can be enabled or disabled independently by the MPU_INTC.INTC_MIRn interrupt mask register.
- In response to an unmasked incoming interrupt, the INTC can generate one of two types of interrupt requests to the processor:
  - IRQ: low-priority interrupt request
  - FIQ: fast interrupt request
- The type of interrupt request is determined by the MPU_INTC.INTC_ILRm[0] FIQNIRQ bit (m= [0,127]).



- The current incoming interrupt status before masking is readable from the MPU_INTC.INTC_ITRn register.
- After masking and IRQ/FIQ selection, and before priority sorting is done, the interrupt status is readable from the MPU_INTC.INTC_PENDING_IRQn and MPU_INTC.INTC_PENDING_FIQn registers.

Interrupt Processing in AM3359 (III)

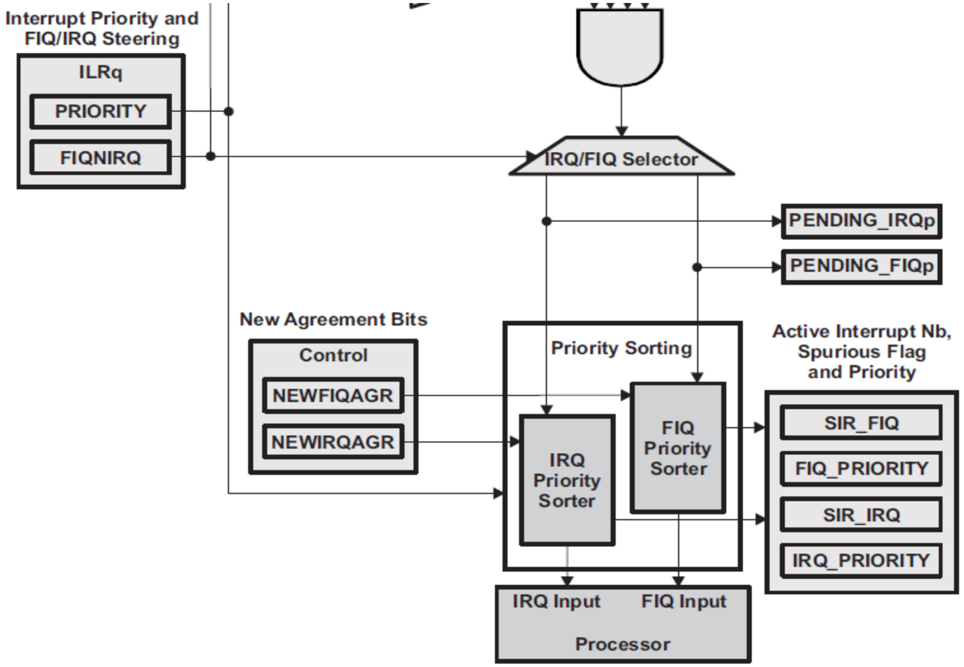


Priority Masking

- To enable faster processing of high-priority interrupts, a programmable priority masking threshold is provided (the MPU_INTC.INTC_THRESHOLD[7:0] PRIORITYTHRESHOLD field).
- This priority threshold allows preemption by higher priority interrupts; *all interrupts of lower or equal priority than the threshold are masked.*
- However, *priority 0 can never be masked by this threshold (NMI).*

- PRIORITY and PRIORITYTHRESHOLD fields values can be set between 0x0 and 0x7F; 0x0 is the highest priority and 0x7F is the lowest priority.
- When priority masking is not necessary, a priority threshold value of 0xFF disables the priority threshold mechanism. This value is also the reset default for backward compatibility with previous versions of the INTC.

Interrupt Processing in AM3359 (IV)



Priority Sorting

- Both the priority level and the interrupt request type are configured by the MPU_INTC.INTC_ILRm register.
- When one or more unmasked incoming interrupts are detected, the INTC separates between IRQ and FIQ using the corresponding MPU_INTC.INTC_ILRm[0] FIQNIRQ bit. The result is placed in INTC_PENDING_IRQn or INTC_PENDING_FIQn.
- Priority sorting for IRQ and FIQ can execute in parallel. Each IRQ/FIQ priority sorter determines the highest priority interrupt number.
- Each priority number is placed in the corresponding MPU_INTC.INTC_SIR_IRQ[6:0] ACTIVEIRQ field or MPU_INTC.INTC_SIR_FIQ[6:0] ACTIVEFIQ field.
- Once the interrupting peripheral device has been serviced and the incoming interrupt deasserted, the user must write to the appropriate NEWIRQAGR or NEWFIQAGR bit in MPU_INTC.INTC_CONTROL register to indicate to the INTC the interrupt has been handled.

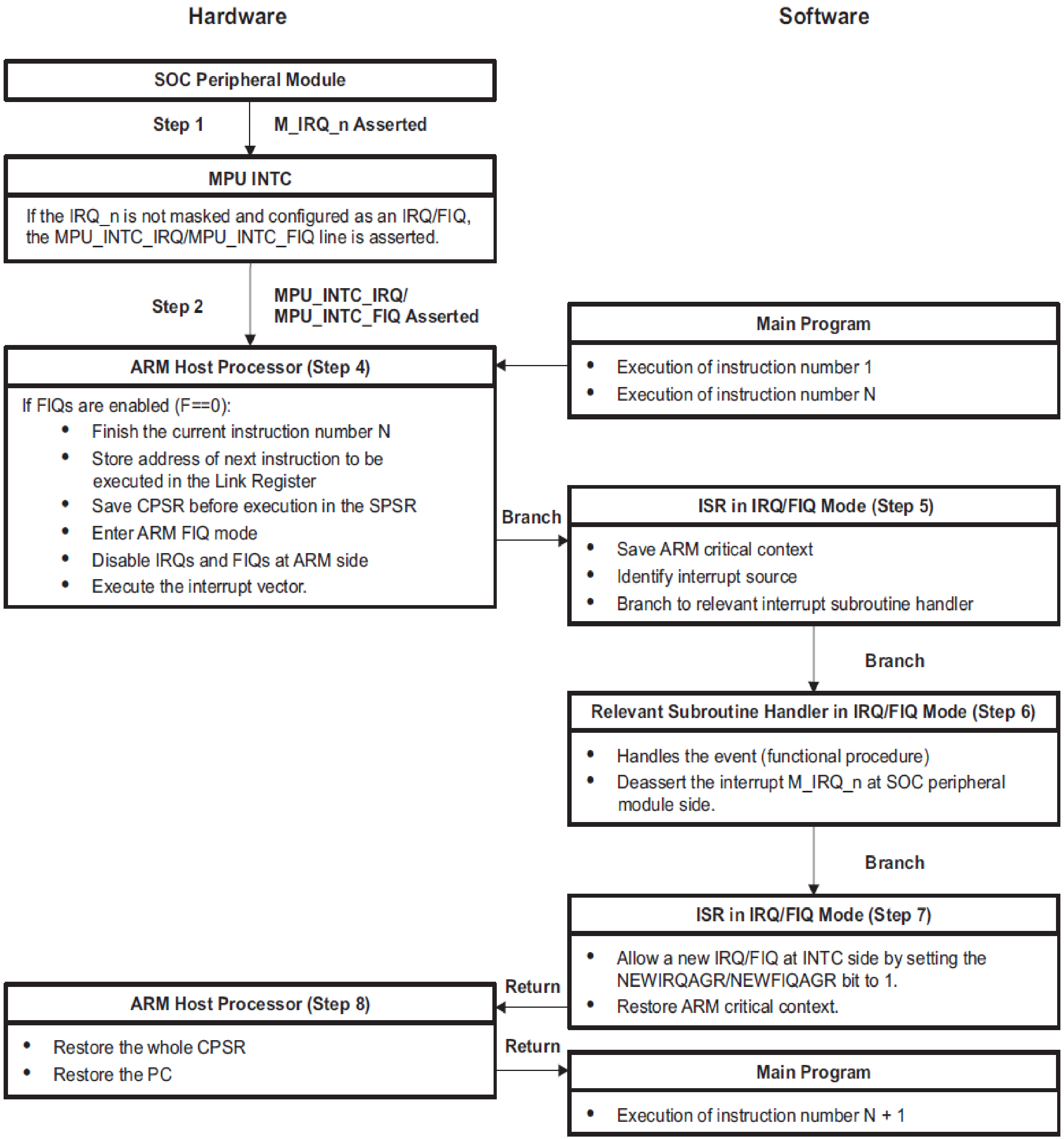
Interrupt Processing in AM3359 (V)

Interrupt Handling

- The IRQ/FIQ interrupt generation takes four to six INTC functional clock cycles depending on MPU_INTC.INTC_IDLE[1] TURBO bit.
- The FIQ/IRQ priority sorting takes 10 functional clock cycles.
- The minimum delay between the interrupt request being generated and the interrupt service routine being executed is such that priority sorting always completes before the MPU_INTC.INTC_SIR_IRQ or MPU_INTC.INTC_SIR_FIQ register is read.

IRQ/FIQ Processing Sequence

- ARM ASM language programming



Hardware

Software

SOC Peripheral Module

Step 1 — M_IRQ_n Asserted

MPU INTC

If the IRQ_n is not masked and configured as an IRQ/FIQ, the MPU_INTC_IRQ/MPU_INTC_FIQ line is asserted.

Step 2 — MPU_INTC_IRQ/ MPU_INTC_FIQ Asserted

Main Program

- Execution of instruction number 1
- Execution of instruction number N

ARM Host Processor (Step 4)

If FIQs are enabled (F==0):

- Finish the current instruction number N
- Store address of next instruction to be executed in the Link Register
- Save CPSR before execution in the SPSR
- Enter ARM FIQ mode
- Disable IRQs and FIQs at ARM side
- Execute the interrupt vector.

Branch

ISR in IRQ/FIQ Mode (Step 5)

- Save ARM critical context
- Identify interrupt source
- Branch to relevant interrupt subroutine handler

Branch

Relevant Subroutine Handler in IRQ/FIQ Mode (Step 6)

- Handles the event (functional procedure)
- Deassert the interrupt M_IRQ_n at SOC peripheral module side.

Branch

ISR in IRQ/FIQ Mode (Step 7)

- Allow a new IRQ/FIQ at INTC side by setting the NEWIRQAGR/NEWFIQAGR bit to 1.
- Restore ARM critical context.

Return

ARM Host Processor (Step 8)

- Restore the whole CPSR
- Restore the PC

Return

Main Program

- Execution of instruction number N + 1

ARM Cortex-A8 Interrupts

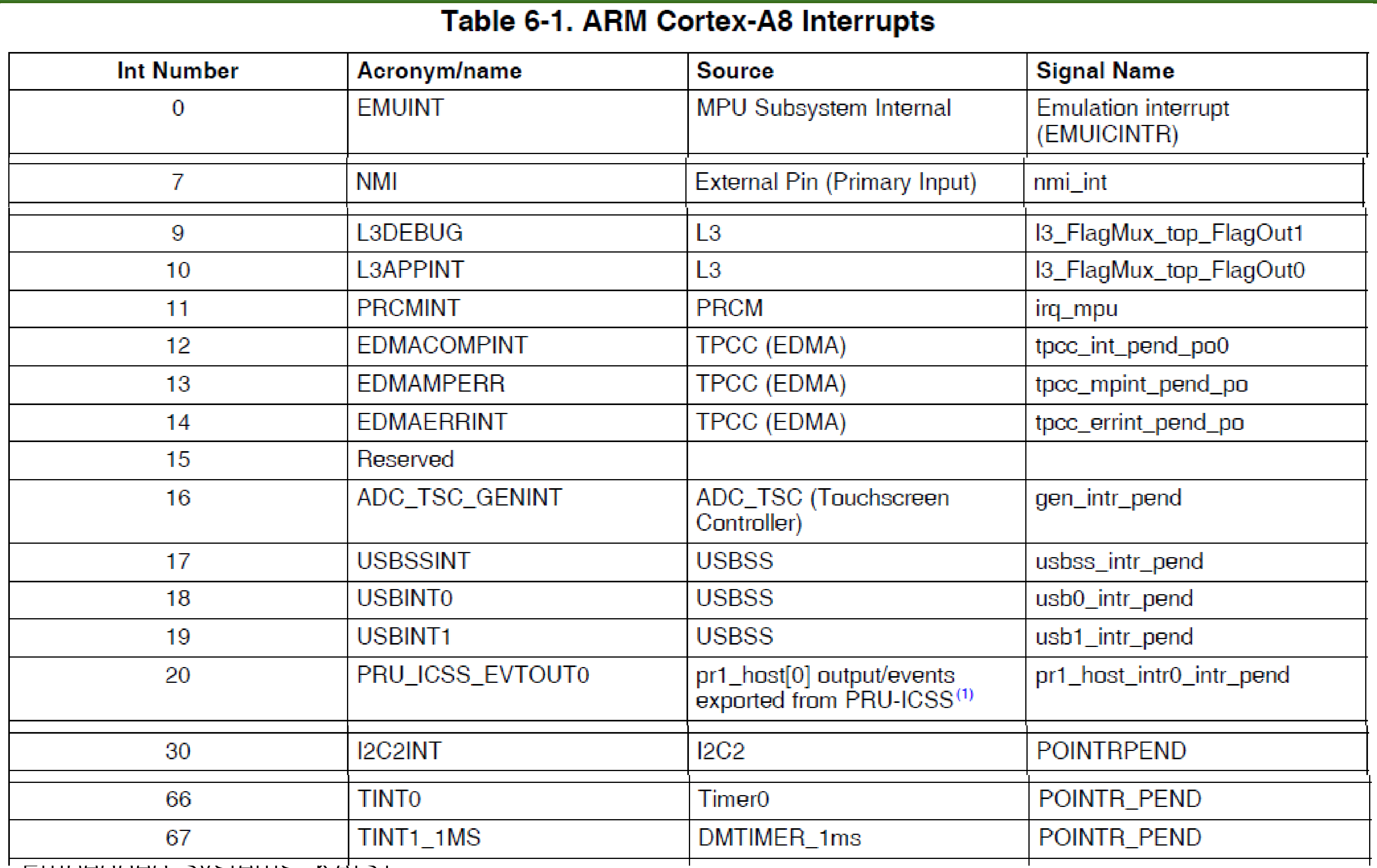


Table 6-1. ARM Cortex-A8 Interrupts

Int Number	Acronym/name	Source	Signal Name
0	EMUINT	MPU Subsystem Internal	Emulation interrupt (EMUICINTR)
7	NMI	External Pin (Primary Input)	nmi_int
9	L3DEBUG	L3	l3_FlagMux_top_FlagOut1
10	L3APPINT	L3	l3_FlagMux_top_FlagOut0
11	PRCMINT	PRCM	irq_mpu
12	EDMACOMPINT	TPCC (EDMA)	tpcc_int_pend_po0
13	EDMAMPERR	TPCC (EDMA)	tpcc_mpint_pend_po
14	EDMAERRINT	TPCC (EDMA)	tpcc_errint_pend_po
15	Reserved		
16	ADC_TSC_GENINT	ADC_TSC (Touchscreen Controller)	gen_intr_pend
17	USBSSINT	USBSS	usbss_intr_pend
18	USBINT0	USBSS	usb0_intr_pend
19	USBINT1	USBSS	usb1_intr_pend
20	PRU_ICSS_EVTOUT0	pr1_host[0] output/events exported from PRU-ICSS[1]	pr1_host_intr0_intr_pend
30	I2C2INT	I2C2	POINTRPEND
66	TINT0	Timer0	POINTR_PEND
67	TINT1_1MS	DMTIMER_1ms	POINTR_PEND

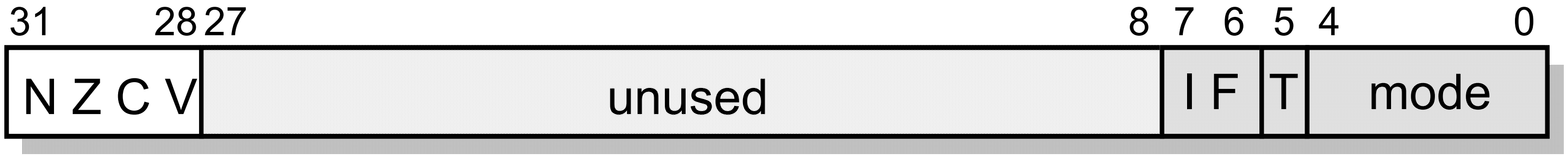
6.6 Exceptions in ARM

- **Exceptions**
  - Internally detected error.
  - Exceptions are synchronous with instructions but unpredictable.
  - Build exception mechanism on top of interrupt mechanism.
  - Exceptions are usually prioritized and vectorized.

- Trap (software interrupt):
  - An exception generated by an instruction.
    - Call supervisor mode.
  - ARM uses SWI instruction for traps.

Exceptions in ARM (II)

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



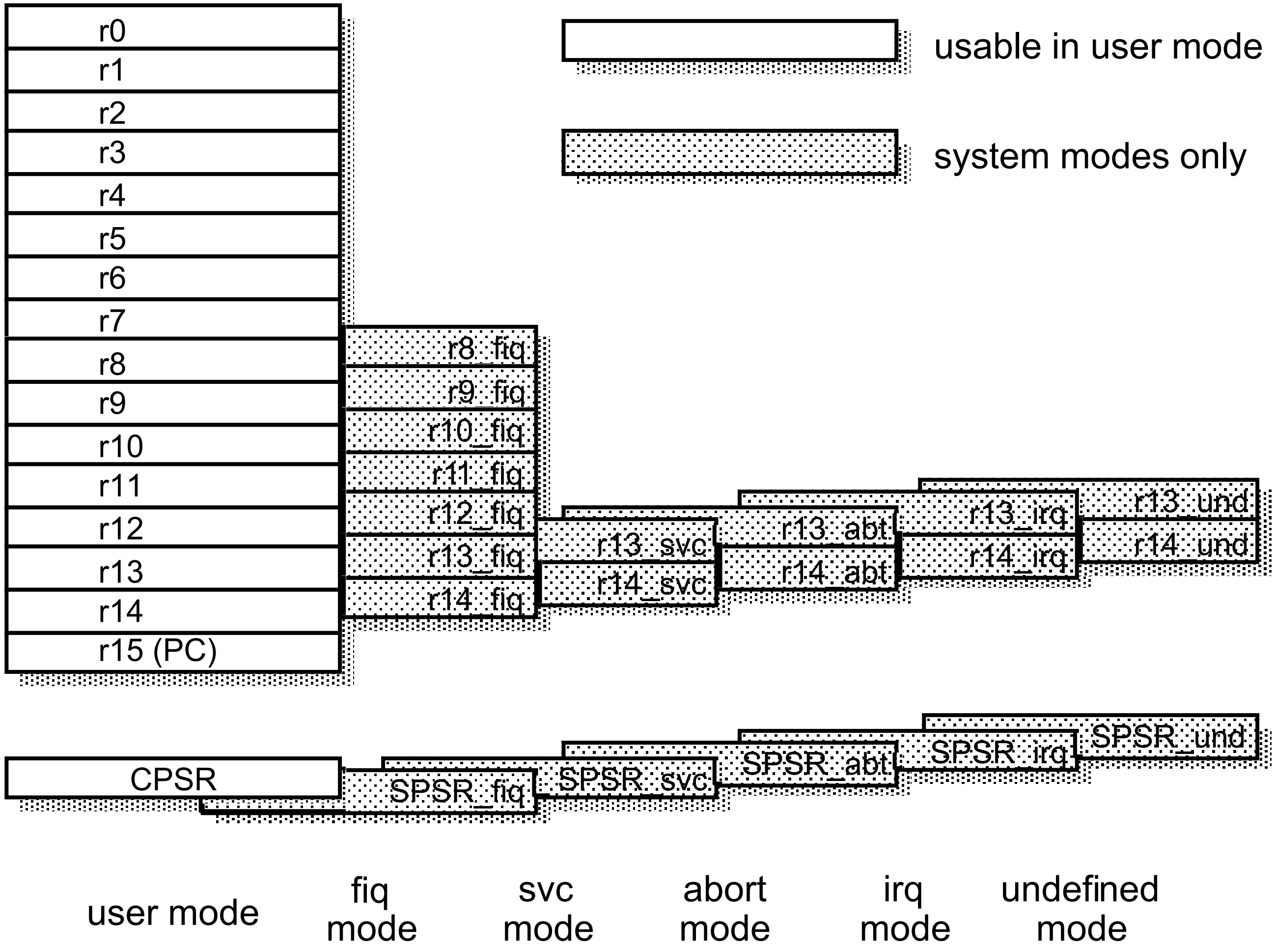
CPSR[4:0]	Mode	Use	Registers
10000	User	Normal user code	user
10001	FIQ	Processing fast interrupts	_fiq
10010	IRQ	Processing standard interrupts	_irq
10011	SVC	Processing software interrupts (SWIs)	_svc
10111	Abort	Processing memory faults	_abt
11011	Undef	Handling undefined instruction traps	_und
11111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

Exceptions in ARM (III)

- **Privileged mode ← → User 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.

The ARM Programmer’s Model

- Visible registers in an ARM processor

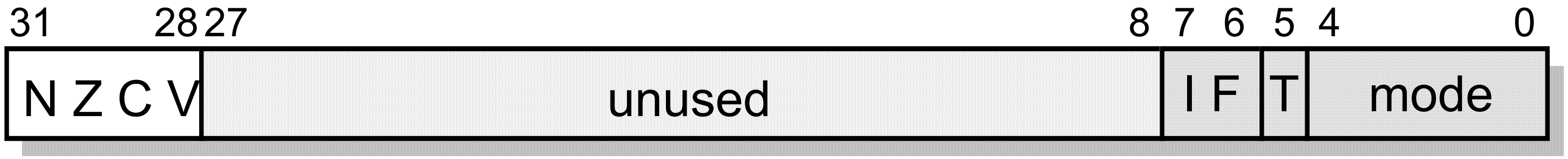


Exceptions in ARM (IV)

- **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 (V)

- **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 (Return Addr, Link Register) 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 (VI)

- 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

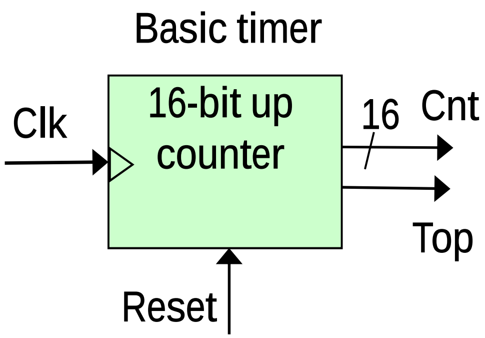
- Registers usage
  - Two registers of the privileged mode saves
    - Return address (Link register): r14
    - Stack pointer: r13
  - The stack pointer may be used to save other user registers so that they can be used by the exception handler.
  - FIQ mode have additional private registers (r8 to r12)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

Exceptions in ARM (VII)

- **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 (Indivisible).
      - MOVS PC, R14

6.7 Timers & Counters [Text 4.2]

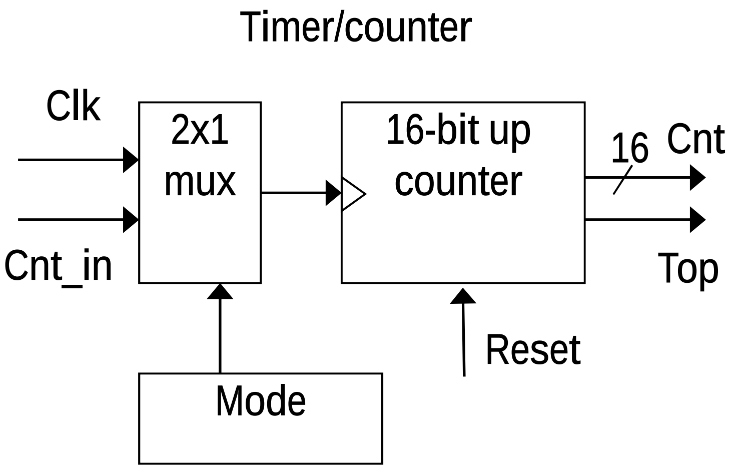
- **Timer:** measures time intervals
  - To generate timed output events
    - e.g., hold traffic light green for 10 s
  - To measure input events
    - e.g., measure a car's speed



- Based on counting clock pulses
    - E.g., let Clk period be 10 ns
    - And we count 20,000 Clk pulses
    - Then 200 microseconds have passed
    - 16-bit counter would count up to
      65,535*10 ns = 655.35 microsec.,
      resolution = 10 ns
    - Top: indicates top count reached, wrap-around

Counters

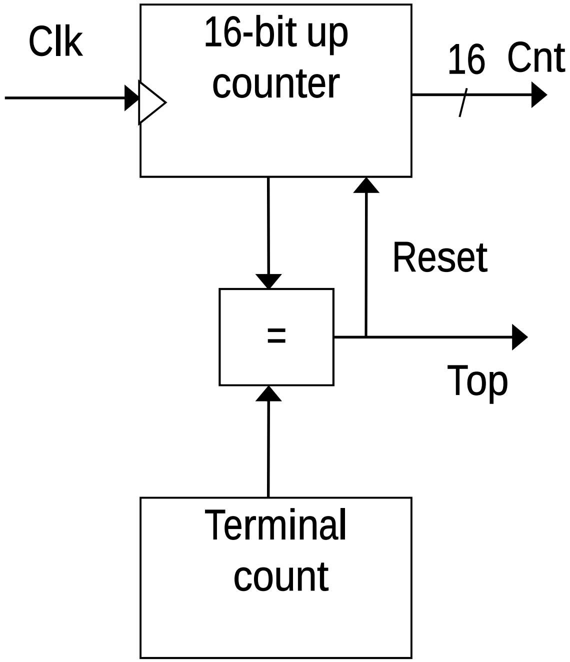
- **Counter:** like a timer, but counts pulses on a general input signal rather than clock
  - e.g., count cars passing over a sensor
  - Can often configure device as *either* a timer or counter



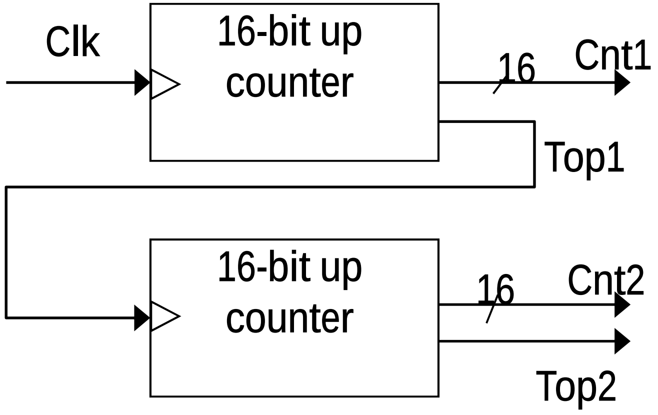
Other Timer Structures

- Interval timer →
  - Indicates when desired time interval has passed
  - We set terminal count to desired interval
    - *Number of clock cycles = Desired time interval / Clock period*
- Cascaded counters →
- Prescaler (Below)
  - Divides clock
  - Increases range, decreases resolution

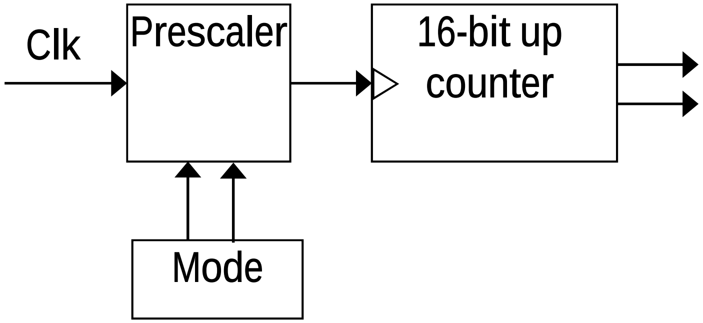
Timer with a terminal count



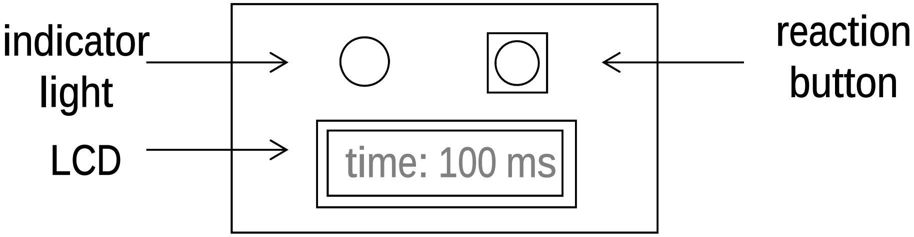
16/32-bit timer



Time with prescaler



Example: Reaction Timer

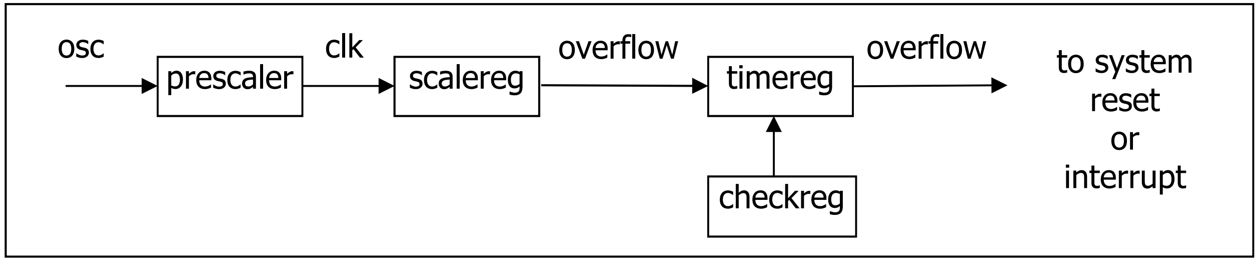


- Measure time between turning light on and user pushing button
  - 16-bit timer, clk period is 83.33 ns, counter increments every 6 cycles
  - Resolution = 6*83.33=0.5 microsec.
  - Range = 65535*0.5 microseconds = 32.77 milliseconds
  - Want program to 1 count millisec., so initialize counter to 65535 – 1000/0.5 = 63535

```
/* main.c */

#define MS_INIT       63535
void main(void){
    int count_milliseconds = 0;

    configure timer mode
    set Cnt to MS_INIT

    wait a random amount of time
    turn on indicator light
    start timer

    while (user has not pushed reaction
button){
      if(Top) {
         stop timer
         set Cnt to MS_INIT
         start timer
         reset Top
         count_milliseconds++;
      }
   }
   turn light off
   printf(“time:  %i ms“,
count_milliseconds);
}
```

Watchdog Timer

- Must reset timer every X time unit, else timer generates a signal
- Common use: detect failure, self-reset
- Another use: timeouts
  - e.g., ATM machine
  - 16-bit timer, 2 microsec. resolution
  - *timereg* value = $2*(2^{16}-1)–X = 131070–X$
  - For 2 min., X = 120,000 microsec.



```
/* main.c */

main(){
  wait until card inserted
  call watchdog_reset_routine

  while(transaction in progress){
    if(button pressed){
      perform corresponding action
      call watchdog_reset_routine
    }

/* if watchdog_reset_routine not called
every < 2 minutes,
interrupt_service_routine is called */
}
```

```
watchdog_reset_routine(){
/* checkreg is set so we can load value into
timereg.  Zero is loaded into scalereg and
11070 is loaded into timereg */

  checkreg = 1
  scalereg = 0
  timereg  = 11070
}

void interrupt_service_routine(){
  eject card
  reset screen
}
```

6.8 Timer & Clock in AM3359

Real-Time Clock (RTC) [Datasheet]

- Real-Time Date (Day/Month/Year/Day of Week) and Time (Hours/Minutes/Seconds)
- Internal 32.768-kHz Oscillator, RTC Logic and 1.1-V Internal LDO
- Independent Power-on-Reset (RTC_PWRONRSTn) Input
- Dedicated Input Pin (EXT_WAKEUP) for External Wake Events
- Programmable Alarm Can be Used to Generate Internal Interrupts to the PRCM (for Wake Up) or Cortex-A8 (for Event Notification)
- Programmable Alarm Can be Used with External Output (PMIC_POWER_EN) to Enable the Power Management IC to Restore Non-RTC Power Domains.

Real-Time Clock in AM3359 (II)

Real Time Clock Subsystem (RTC_SS) [TRM p. 3808]

- The real-time clock is a precise timer which can generate interrupts on intervals specified by the user.
  - Interrupts can occur every second, minute, hour, or day.
  - The clock itself can track the passage of real time for durations of several years, provided it has a sufficient power source the whole time.
- The basic purpose for the RTC is to keep time of day.
  - The other equally important purpose of RTC is for Digital Rights management.
  - The final purpose of RTC is to wake the rest of chip up from a power down state.
- Alarms are available to interrupt the CPU at a particular time, or at periodic time intervals, such as once per minute or once per day.

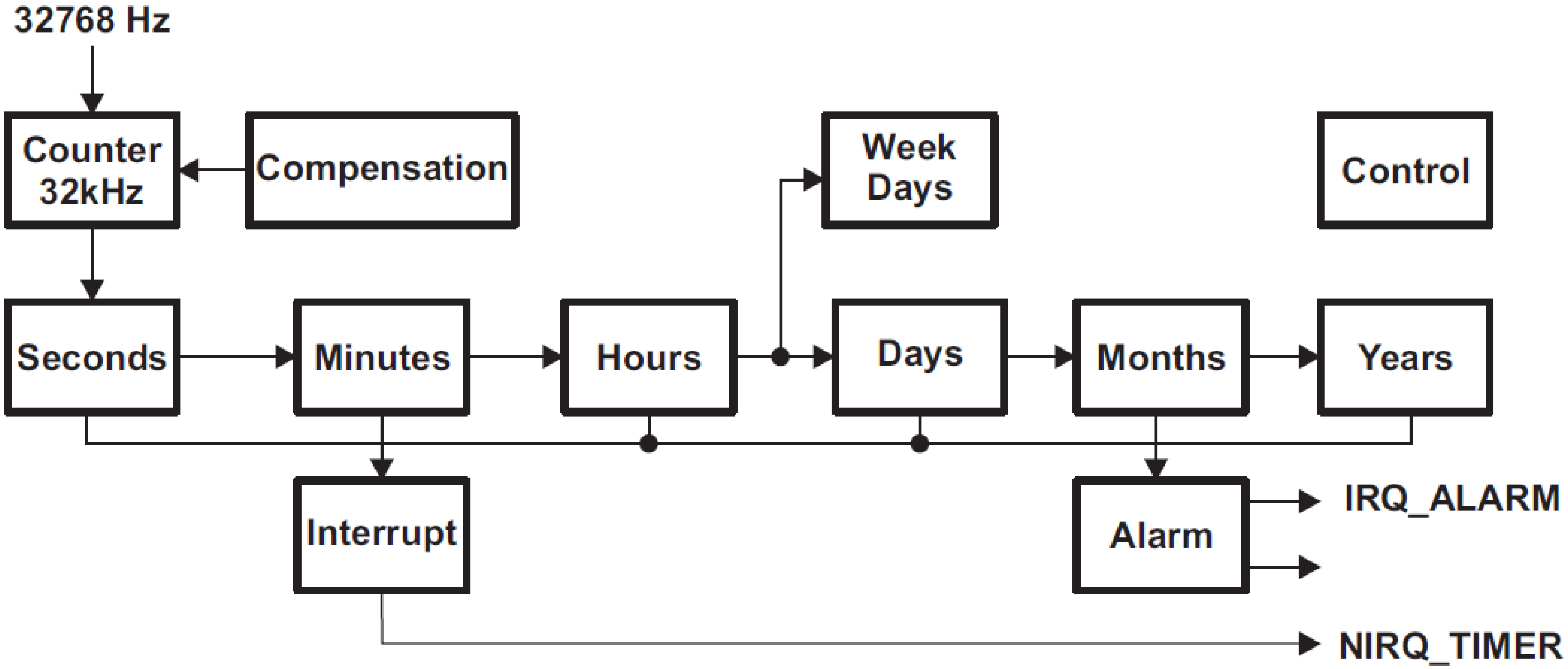
Real-Time Clock in AM3359 (III)

Features of RTC_SS

- 100-year calendar (xx00 to xx99)
- Counts seconds, minutes, hours, day of the week, date, month, and year with leap year compensation
- Binary-coded-decimal (BCD) representation of time, calendar, and alarm
- 12-hour clock mode (with AM and PM) or 24-hour clock mode
- Alarm interrupt
- Periodic interrupt
- Single interrupt to the CPU
- Supports external 32.768-kHz crystal or external clock source of the same frequency.

Real-Time Clock in AM3359 (IV)

- **RTC Functional Block Diagram**



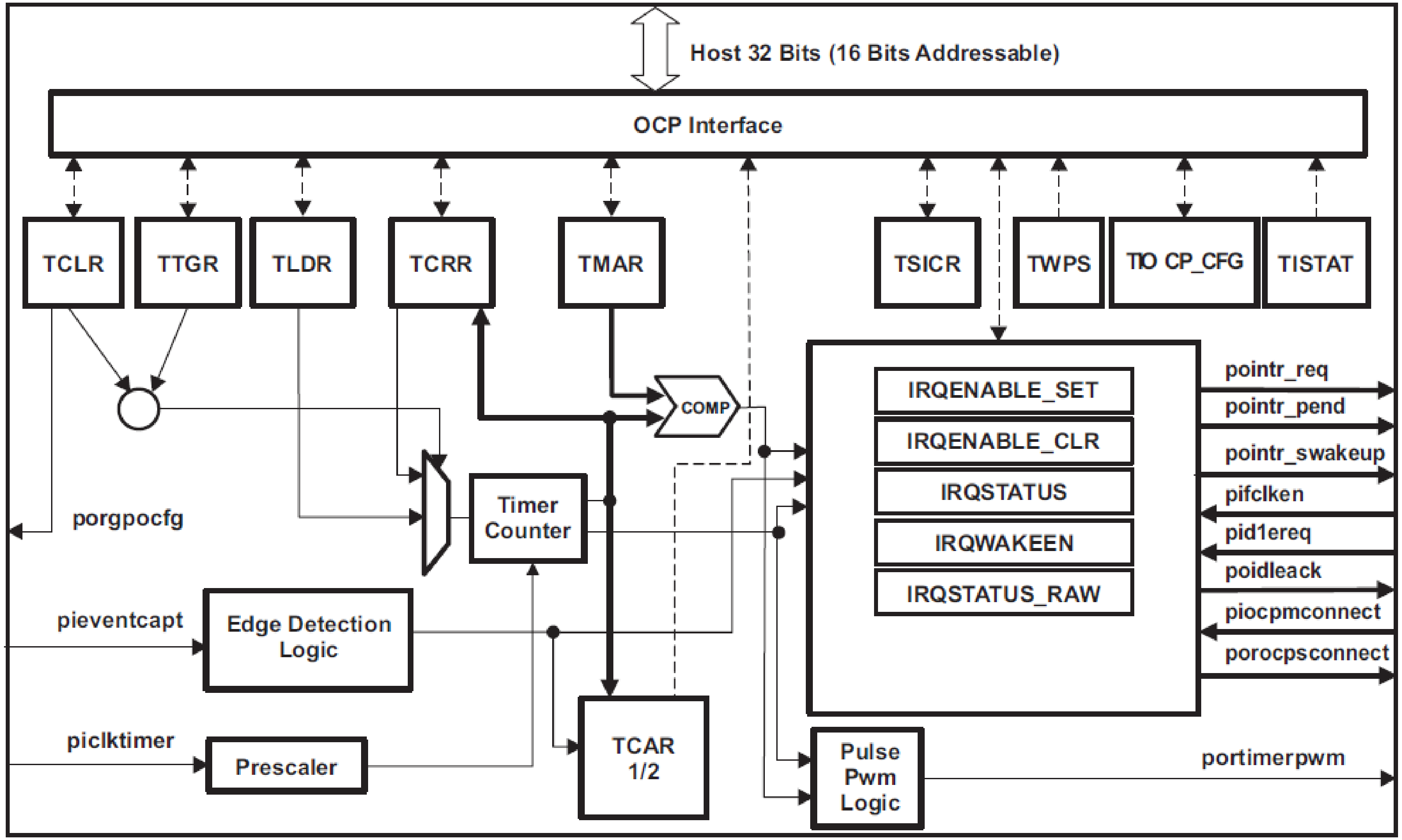
DM Timer in AM3359

DM Timer Overview [TRM p. 3741]

- The timer module contains a free running upward counter with auto reload capability on overflow.
  - The timer counter can be read and written in real-time (while counting).
  - The timer module includes compare logic to allow an interrupt event on a programmable counter matching value.
- A dedicated output signal can be pulsed or toggled on overflow and match event.
- A dedicated input signal is used to trigger automatic timer counter capture and interrupt event, on programmable input signal transition.
- A programmable clock divider (prescaler) allows reduction of the timer input clock frequency.
- All internal timer interrupt sources are merged in one module interrupt line and one wake-up line.
  - Each internal interrupt sources can be independently enabled/disabled.
- This module is controllable through the OCP peripheral bus.

DM Timer in AM3359 (II)

Timer Block Diagram



DM Timer in AM3359 (III)

Functional Description

- The general-purpose timer is an upward counter. It supports 3 functional modes:
  - Timer mode
  - Capture mode
  - Compare mode
- By default, after core reset, the capture and compare modes are disabled.

DM Timer in AM3359 (IV)

Timer Mode Functionality

- The timer is an upward counter that can be started and stopped at any time through the Timer Control Register (TCLR ST bit).
- The Timer Counter Register (TCRR) can be loaded when stopped or on the fly (while counting).
  - TCRR can be loaded directly by a TCRR Write access with the new timer value.
  - The timer counter register TCRR value can be read when stopped or captured on the fly by a TCRR Read access.
- In the one shot mode (TCLR AR bit = 0), the counter is stopped after counting overflow (counter value remains at zero).
  - When the auto-reload mode is enabled (TCLR AR bit = 1), the TCRR is reloaded with the Timer Load Register (TLDR) value after a counting overflow.
- An interrupt can be issued on overflow if the overflow interrupt enable bit is set in the timer Interrupt Enable Register (IRQENABLE_SET OVF_IT_FLAG bit = 1).
- Linux kernel contains DM timer driver.

References

- Interrupt Controller, Real-Time clock, DM timer in AM3359
  - AM3359 Datasheet, AM335x ARM Cortex-A8 Microprocessors (MPUs) (Rev. D), http://www.ti.com/lit/ds/sprs717d/sprs717d.pdf
  - Technical Reference Manual - AM335x ARM Cortex-A8 Microprocessors (MPUs) Technical Reference Manual (Rev. F), http://www.ti.com/lit/ug/spruh73f/spruh73f.pdf
- Exceptions in ARM
  - Steve Furber, "ARM system-on-chip architecture", Addison Wesley, 2000.