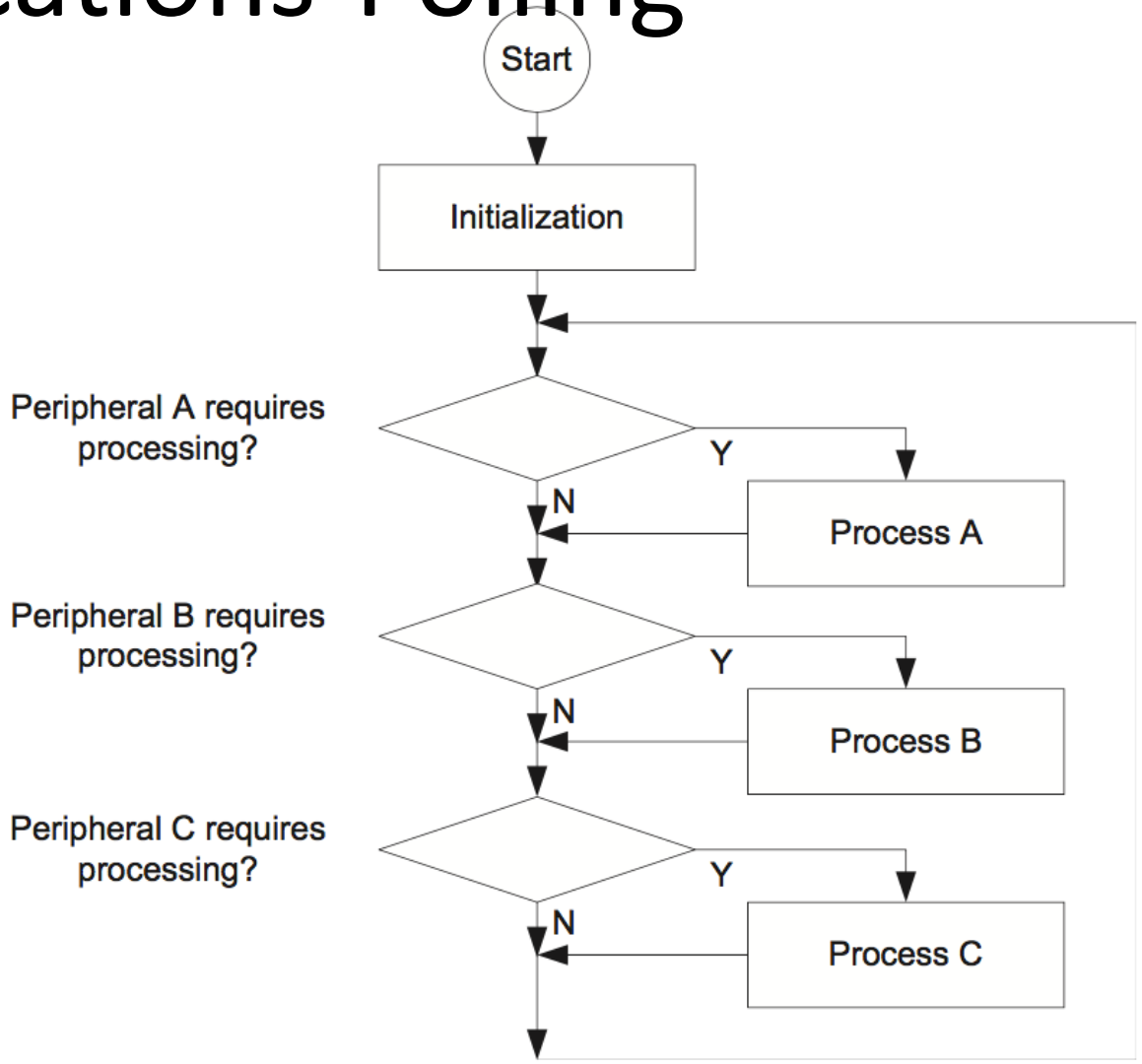
Interrupt and Exception

Why We Need Exceptions and Interrupts?

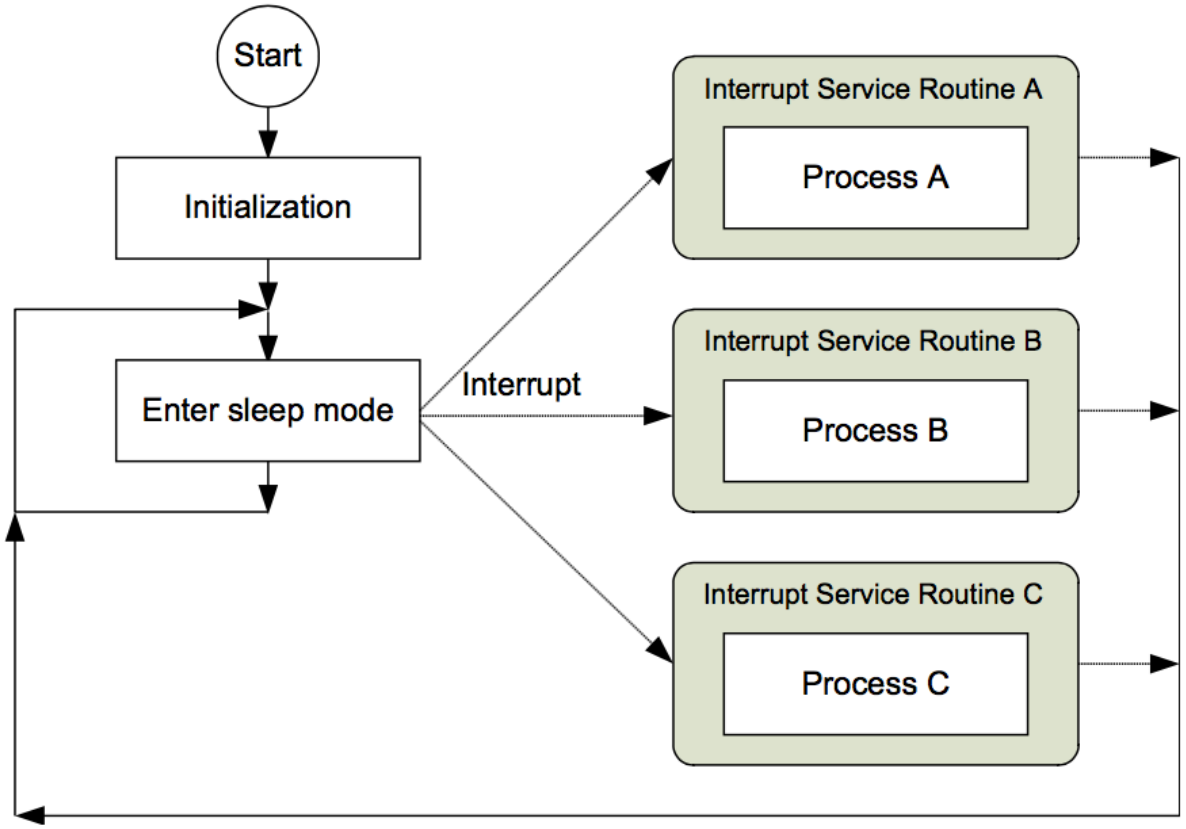
- How could you write a program handling
  - User inputs
  - Display (including animation)
  - Timeout (including timer)
  - Music
  - Network
  - Incoming calls and LINE messages
- at the same time?

- Polling-based and sequential programing is not enough

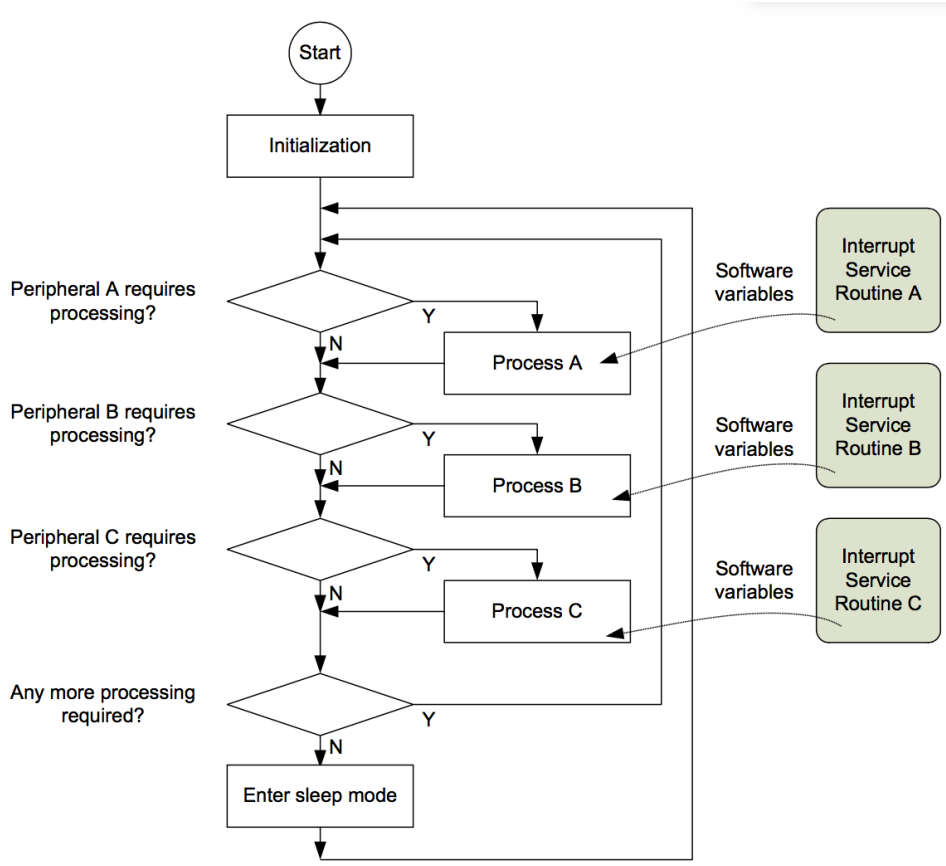
Applications-Polling



Application-Interrupt Driven



Application-Combined

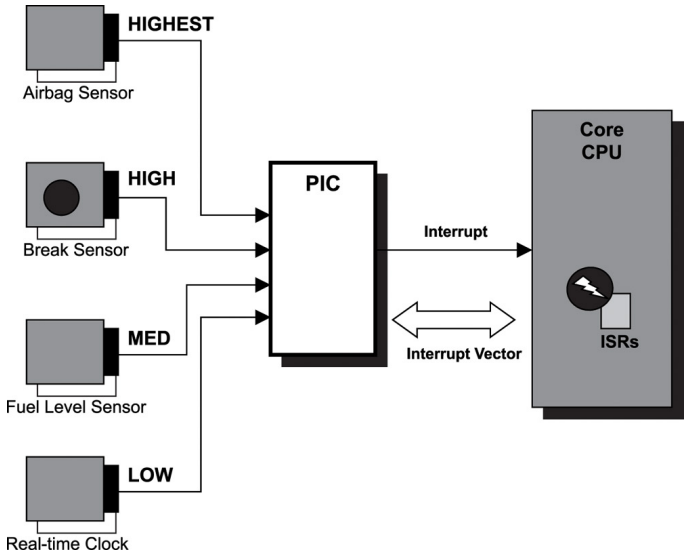


Definitions

- Exceptions are events that cause changes in program flow control outside a normal code sequence
- When it happens, the program that is currently executing is suspended, and a piece of code associated with the event (the exception handler) is executed
- The events could either be external or internal
- When an event is from an external source, it is commonly known as an interrupt or interrupt request (IRQ)

How Exceptions and Interrupts Work?

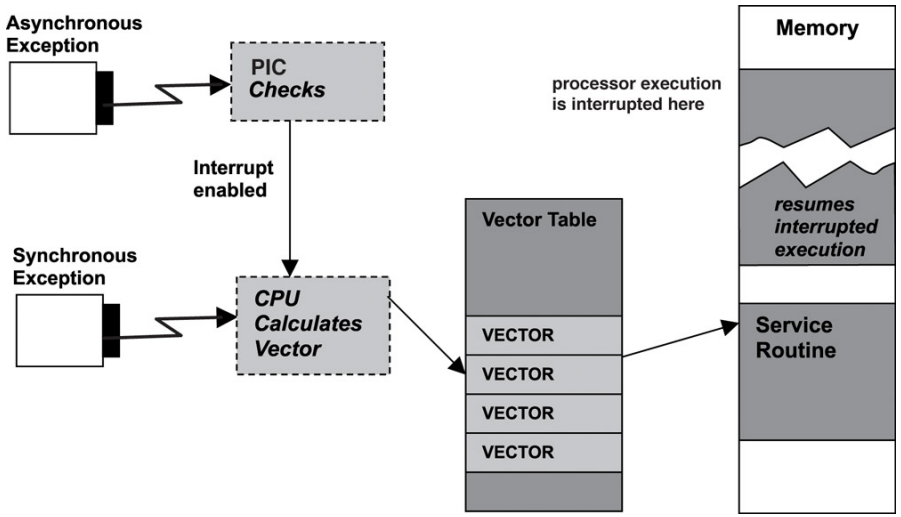
- How does exceptions and interrupts work ?
  - Programmable interrupt controller



Source: Qing Li "real-time concepts for embedded systems"

Exceptions and interrupts (Cont.)

- Processing general exceptions (Cont.)
  - Loading and invoking exception handlers

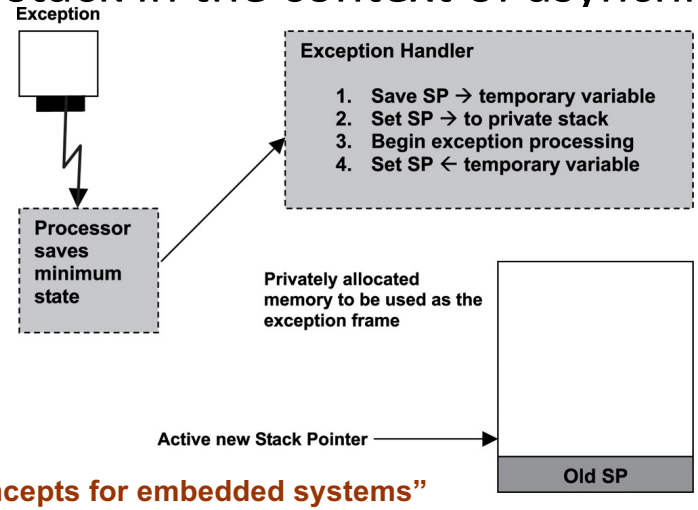


  - Nested exceptions and stack overflow

Source: Qing Li "real-time concepts for embedded systems"

Exceptions and interrupts (Cont.)

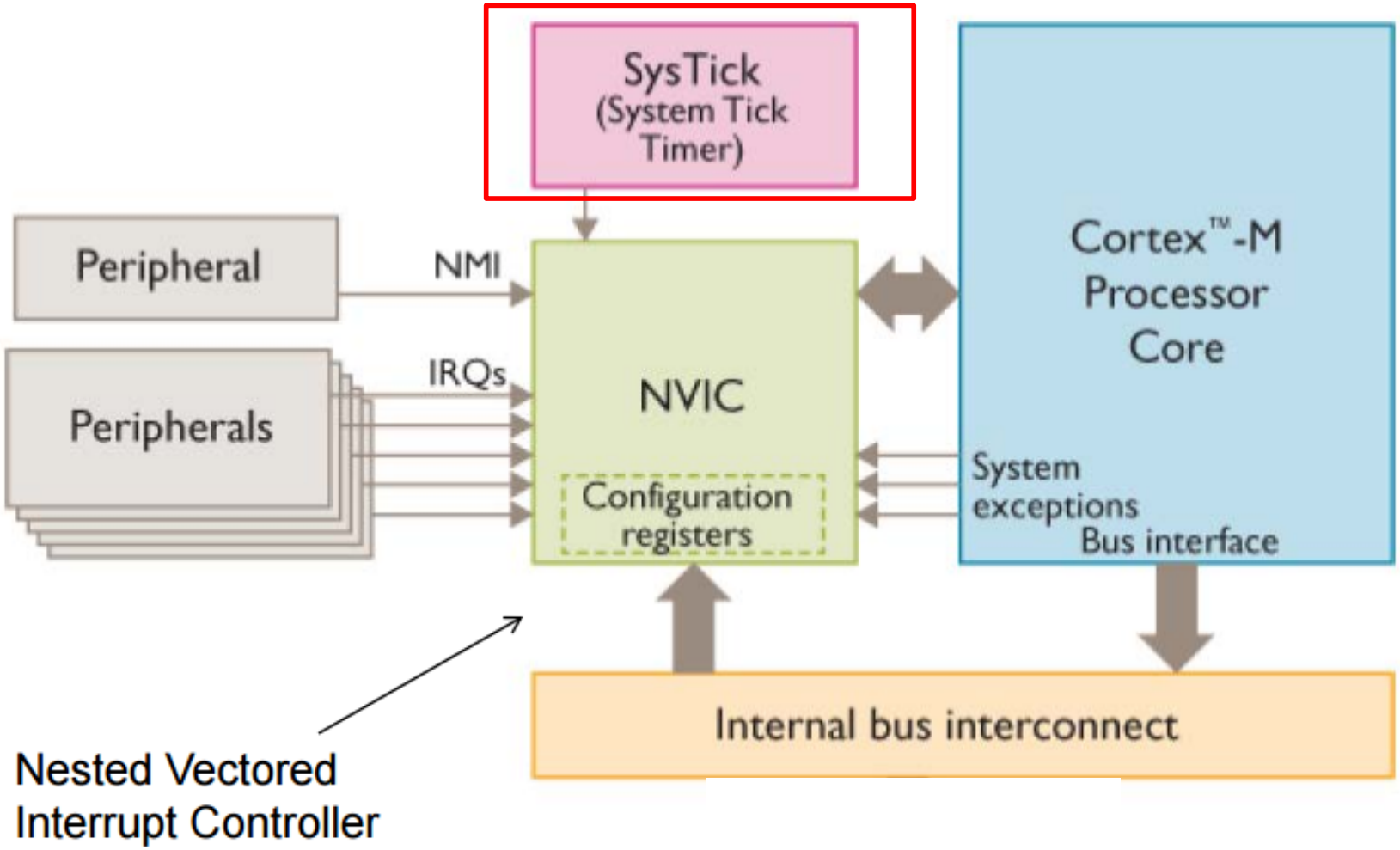
- Exception handlers
  - Exception frame
    - The exception frame is also called the interrupt stack in the context of asynchronous exceptions.



Source: Qing Li "real-time concepts for embedded systems"

Cortex-M structure

Also exception



Exception

Table 4.9 Exception Types

Exception Number	CMSIS Interrupt Number	Exception Type	Priority	Function
1	—	Reset	−3 (Highest)	Reset
2	−14	NMI	−2	Non-Maskable interrupt
3	−13	HardFault	−1	All classes of fault, when the corresponding fault handler cannot be activated because it is currently disabled or masked by exception masking
4	−12	MemManage	Settable	Memory Management fault; caused by MPU violation or invalid accesses (such as an instruction fetch from a non-executable region)
5	−11	BusFault	Settable	Error response received from the bus system; caused by an instruction prefetch abort or data access error
6	−10	Usage fault	Settable	Usage fault; typical causes are invalid instructions or invalid state transition attempts (such as trying to switch to ARM state in the Cortex-M3)
7–10	—	—	—	Reserved
11	−5	SVC	Settable	Supervisor Call via SVC instruction
12	−4	Debug monitor	Settable	Debug monitor – for software based debug (often not used)
13	—	—	—	Reserved
14	−2	PendSV	Settable	Pendable request for System Service
15	−1	SYSTICK	Settable	System Tick Timer
16–255	0–239	IRQ	Settable	IRQ input #0–239

Interrupt Request(IRQ)

```
/******  STM32 specific Interrupt Numbers **********************************************************************
  WWDG_IRQn                   = 0,      /*!< Window WatchDog Interrupt
  PVD_PVM_IRQn                = 1,      /*!< PVD/PVM1/PVM2/PVM3/PVM4 through EXTI Line detection Interrupts
  TAMP_STAMP_IRQn             = 2,      /*!< Tamper and TimeStamp interrupts through the EXTI line
  RTC_WKUP_IRQn               = 3,      /*!< RTC Wakeup interrupt through the EXTI line
  FLASH_IRQn                  = 4,      /*!< FLASH global Interrupt
  RCC_IRQn                    = 5,      /*!< RCC global Interrupt
  EXTI0_IRQn                  = 6,      /*!< EXTI Line0 Interrupt
  EXTI1_IRQn                  = 7,      /*!< EXTI Line1 Interrupt
  EXTI2_IRQn                  = 8,      /*!< EXTI Line2 Interrupt
  EXTI3_IRQn                  = 9,      /*!< EXTI Line3 Interrupt
  EXTI4_IRQn                  = 10,     /*!< EXTI Line4 Interrupt
  DMA1_Channel1_IRQn          = 11,     /*!< DMA1 Channel 1 global Interrupt
  DMA1_Channel2_IRQn          = 12,     /*!< DMA1 Channel 2 global Interrupt
  DMA1_Channel3_IRQn          = 13,     /*!< DMA1 Channel 3 global Interrupt
  DMA1_Channel4_IRQn          = 14,     /*!< DMA1 Channel 4 global Interrupt
  DMA1_Channel5_IRQn          = 15,     /*!< DMA1 Channel 5 global Interrupt
  DMA1_Channel6_IRQn          = 16,     /*!< DMA1 Channel 6 global Interrupt
  DMA1_Channel7_IRQn          = 17,     /*!< DMA1 Channel 7 global Interrupt
  ADC1_2_IRQn                 = 18,     /*!< ADC1, ADC2 SAR global Interrupts
  CAN1_TX_IRQn                = 19,     /*!< CAN1 TX Interrupt
  CAN1_RX0_IRQn               = 20,     /*!< CAN1 RX0 Interrupt
  CAN1_RX1_IRQn               = 21,     /*!< CAN1 RX1 Interrupt
  CAN1_SCE_IRQn               = 22,     /*!< CAN1 SCE Interrupt
  EXTI9_5_IRQn                = 23,     /*!< External Line[9:5] Interrupts
  TIM1_BRK_TIM15_IRQn         = 24,     /*!< TIM1 Break interrupt and TIM15 global interrupt
  TIM1_UP_TIM16_IRQn          = 25,     /*!< TIM1 Update Interrupt and TIM16 global interrupt
  TIM1_TRG_COM_TIM17_IRQn     = 26,     /*!< TIM1 Trigger and Commutation Interrupt and TIM17 global interrup
  TIM1_CC_IRQn                = 27,     /*!< TIM1 Capture Compare Interrupt
  TIM2_IRQn                   = 28,     /*!< TIM2 global Interrupt
  TIM3_IRQn                   = 29,     /*!< TIM3 global Interrupt
```

11.3 Interrupt and exception vectors

Table 42. STM32L4x6 vector table

Position	Priority	Type of priority	Acronym	Description	Address
-	-	-	-	Reserved	0x0000 0000
-	-3	fixed	Reset	Reset	0x0000 0004
-	-2	fixed	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.	0x0000 0008
-	-1	fixed	HardFault	All classes of fault	0x0000 000C
-	0	settable	MemManage	Memory management	0x0000 0010
-	1	settable	BusFault	Pre-fetch fault, memory access fault	0x0000 0014
-	2	settable	UsageFault	Undefined instruction or illegal state	0x0000 0018
-	-	-	-	Reserved	0x0000 001C - 0x0000 0028
-	3	settable	SVCall	System service call via SWI instruction	0x0000 002C
-	4	settable	Debug	Monitor	0x0000 0030
-	-	-	-	Reserved	0x0000 0034
-	5	settable	PendSV	Pendable request for system service	0x0000 0038
-	6	settable	SysTick	System tick timer	0x0000 003C
0	7	settable	WWDG	Window Watchdog interrupt	0x0000 0040
1	8	settable	PVD_PVM	PVD/PVM1/PVM2/PVM3/PVM4 through EXTI lines 16/35/36/37/38 interrupts	0x0000 0044
2	9	settable	RTC_TAMP_STAMP /CSS_LSE	RTC Tamper or TimeStamp /CSS on LSE through EXTI line 19 interrupts	0x0000 0048
3	10	settable	RTC_WKUP	RTC Wakeup timer through EXTI line 20 interrupt	0x0000 004C

4	11	settable	FLASH	Flash global interrupt	0x0000 0050
5	12	settable	RCC	RCC global interrupt	0x0000 005C
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000 005C
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000 005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000 0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000 0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000 0068
11	18	settable	DMA1_CH1	DMA1 channel 1 interrupt	0x0000 006C
12	19	settable	DMA1_CH2	DMA1 channel 2 interrupt	0x0000 0070
13	20	settable	DMA1_CH3	DMA1 channel 3 interrupt	0x0000 0074
14	21	settable	DMA1_CH4	DMA1 channel 4 interrupt	0x0000 0078
15	22	settable	DMA1_CH5	DMA1 channel 5 interrupt	0x0000 007C
16	23	settable	DMA1_CH6	DMA1 channel 6 interrupt	0x0000 0080
17	24	settable	DMA1_CH7	DMA1 channel 7 interrupt	0x0000 0084
18	25	settable	ADC1_2	ADC1 and ADC2 global interrupt	0x0000 0088
19	26	settable	CAN1_TX	CAN1_TX interrupts	0x0000 008C
20	27	settable	CAN1_RX0	CAN1_RX0 interrupts	0x0000 0090
21	28	settable	CAN1_RX1	CAN1_RX1 interrupt	0x0000 0094
22	29	settable	CAN1_SCE	CAN1_SCE interrupt	0x0000 0098
23	30	settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000 009C
24	31	settable	TIM1_BRK/TIM15	TIM1 Break/TIM15 global interrupts	0x0000 00A0
25	32	settable	TIM1_UP/TIM16	TIM1 Update/TIM16 global interrupts	0x0000 00A4
26	33	settable	TIM1_TRG_COM /TIM17	TIM1 trigger and commutation/TIM17 interrupts	0x0000 00A8
27	34	settable	TIM1_CC	TIM1 capture compare interrupt	0x0000 00AC
28	35	settable	TIM2	TIM2 global interrupt	0x0000 00B0
29	36	settable	TIM3	TIM3 global interrupt	0x0000 00B4
30	37	settable	TIM4	TIM4 global interrupt	0x0000 00B8

What Are those Interrupt/Exceptions?

- NMI
  - non-recoverable errors (corruption in system memory such as parity and ECC errors)
- Hard fault
  - unknown opcode
- SVC and PendSVC
  - OS system calls
- SysTick
  - Timer
- Interrupt #x
  - interrupts

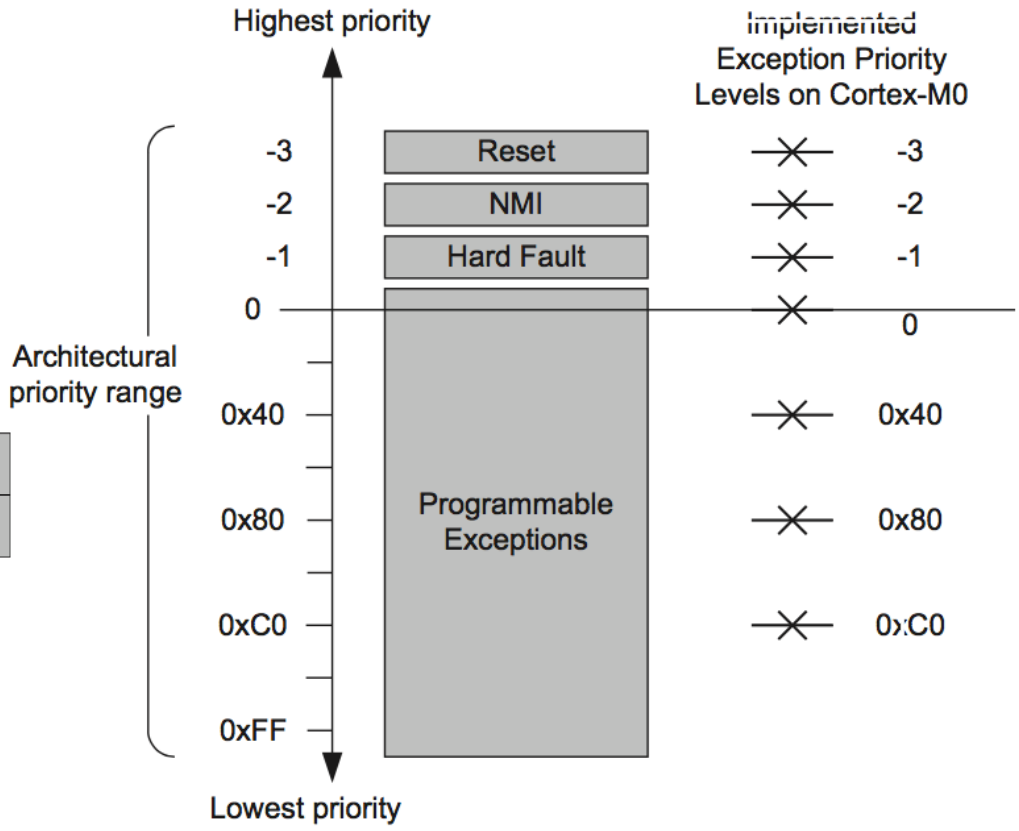
ARM M0 Interrupts

- 1 to 32 interrupts
- Interrupt signals could be connected from on-chip peripherals or from an external source via the I/O port
- External interrupts need to be enabled before being used
- If an interrupt is not enabled, or if the processor is already running another exception handler with same or higher priority, the interrupt request will be stored in a pending status register
- The pended interrupt request can be triggered when the priority level allows. For example, when a higher-priority interrupt handler has been completed and returned
- NVIC can accept interrupt request signals in the form of a high logic level, as well as an interrupt pulse (with a minimum of one clock cycle)

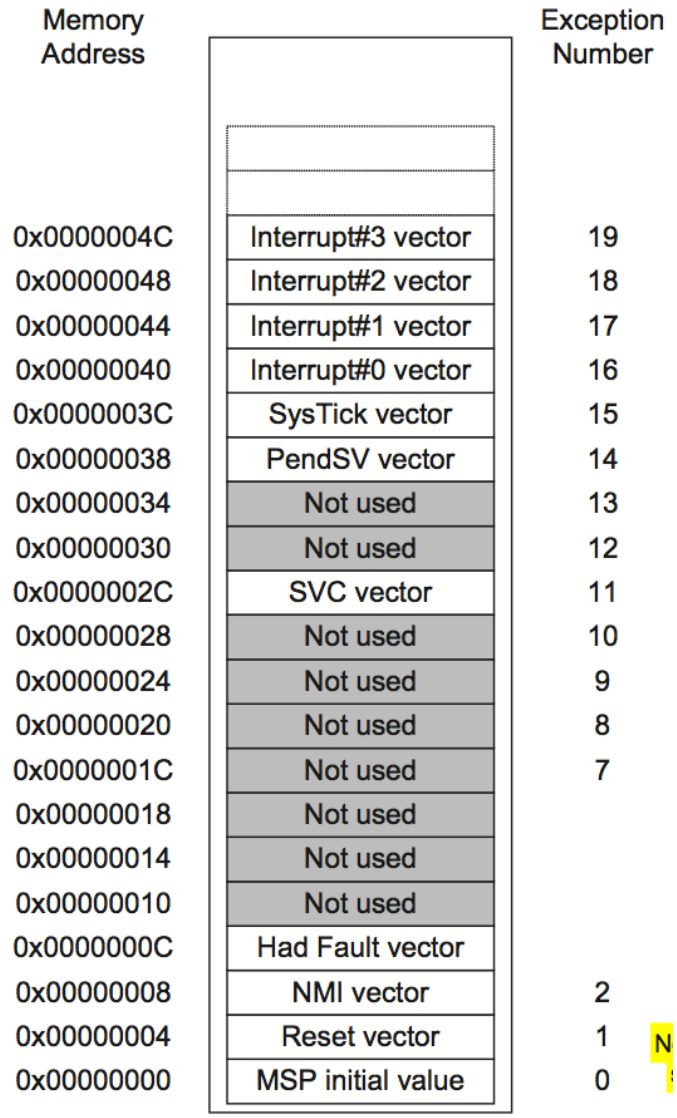
Exception Priority

- Why and How
- Each exception has a priority level

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Implemented		Not implemented, read as zero					



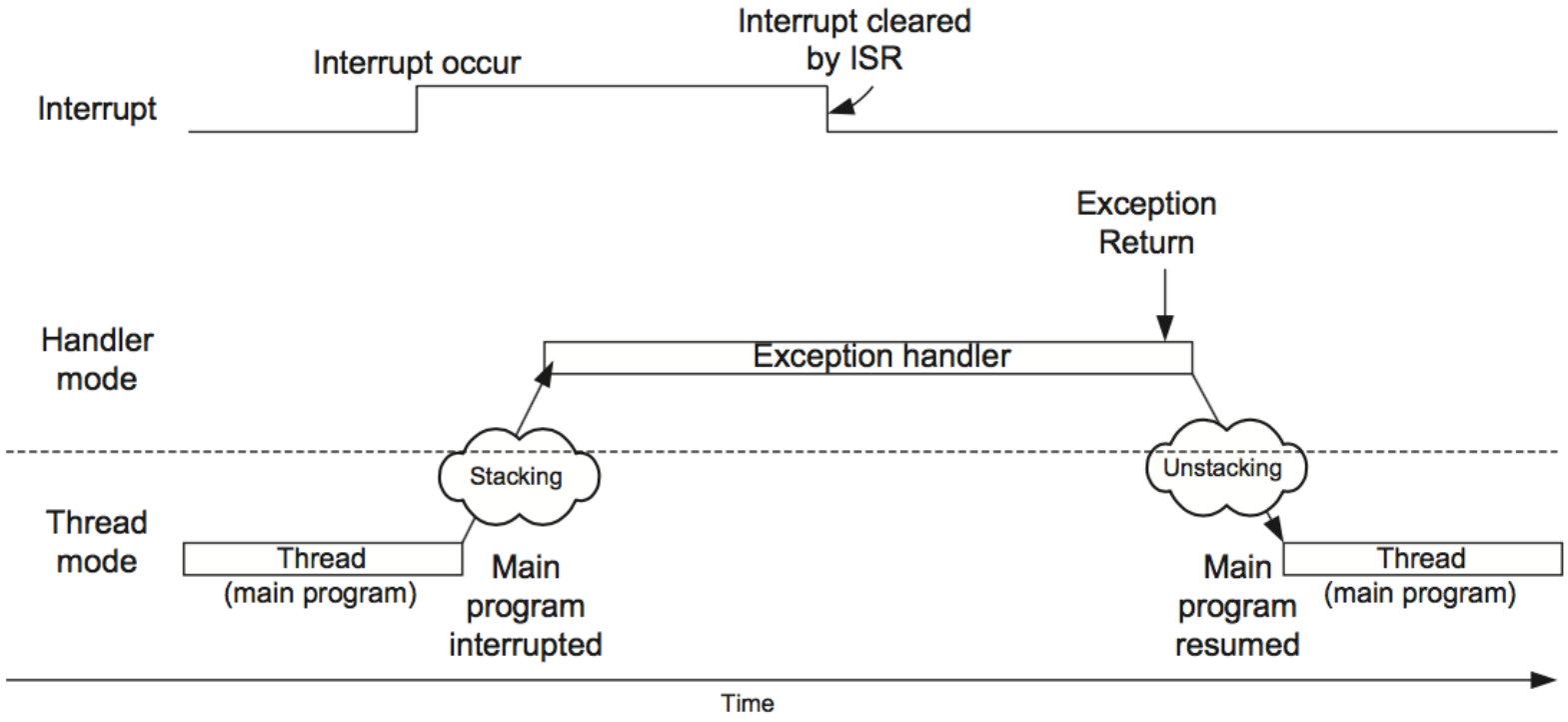
Vector Table



Exception Sequence Overview

- The processor accepts an exception if the following conditions are satisfied:
  - For interrupt and SysTick interrupt requests, the interrupt has to be enabled
  - The processor is not running an exception handler of the same or a higher priority
  - The exception is not blocked by the PRIMASK interrupt masking register

Exception Sequence Overview (Cont.)



Stacking and Unstacking

- To allow an interrupted program to be resumed correctly
- Different processor architectures have different ways to do this
- M0 processor, the architecture uses a mixture of automatic hardware arrangement and, only if necessary, additional software steps for saving and restoring processor status
- Some of the registers in the register banks (R0 to R3, R12, R14), the return address (PC), and the Program Status Register (xPSR) are pushed to the current active stack memory automatically
- The Link Register (LR/R14) is then updated to a special value to be used during exception return
- Exception vector is automatically located and the exception handler starts to execute

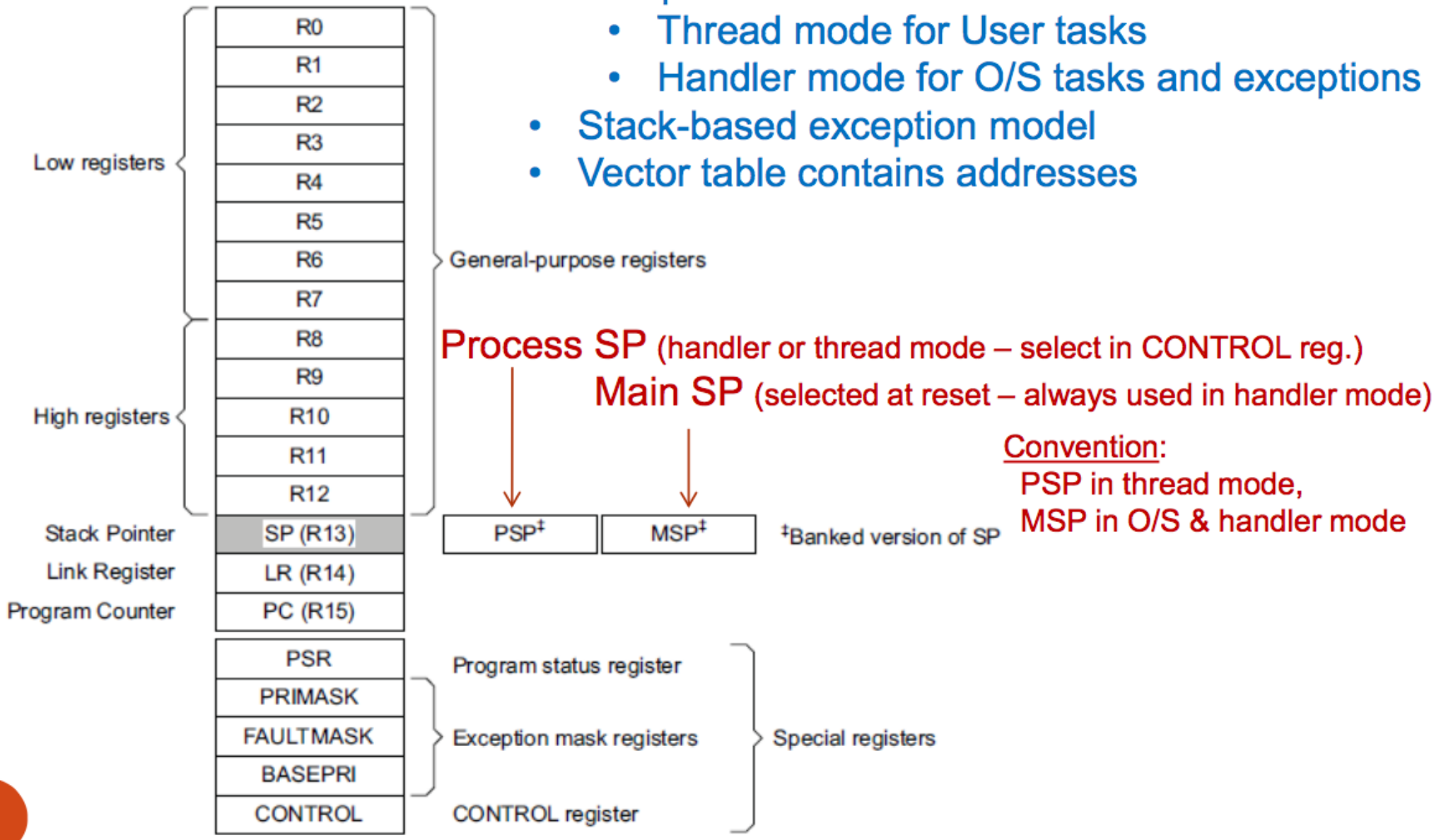
Stacking and Unstacking (Cont.)

- At the end of the exception handling process, the exception handler executes a return using the special value (EXC_RETURN, previously generated in LR) to trigger the exception return mechanism
- The processor checks to determine if there is any other exception to be serviced. If not, the register values previously stored on the stack memory are restored and the interrupted program is resumed
- The registers not saved by the automatic stacking process will have to be saved and restored by software in the exception handler
- Two different instructions can be used for exception return:
  - `BX <Reg>; Load a register value into PC (e.g., "BX LR") and`
  - `POP {<Reg1>,< Reg2>,..,PC}; POP instruction with PC being one of the registers being updated`

- *How about functional calls?*

Cortex CPU core registers

- Two processor modes:
  - Thread mode for User tasks
  - Handler mode for O/S tasks and exceptions
- Stack-based exception model
- Vector table contains addresses

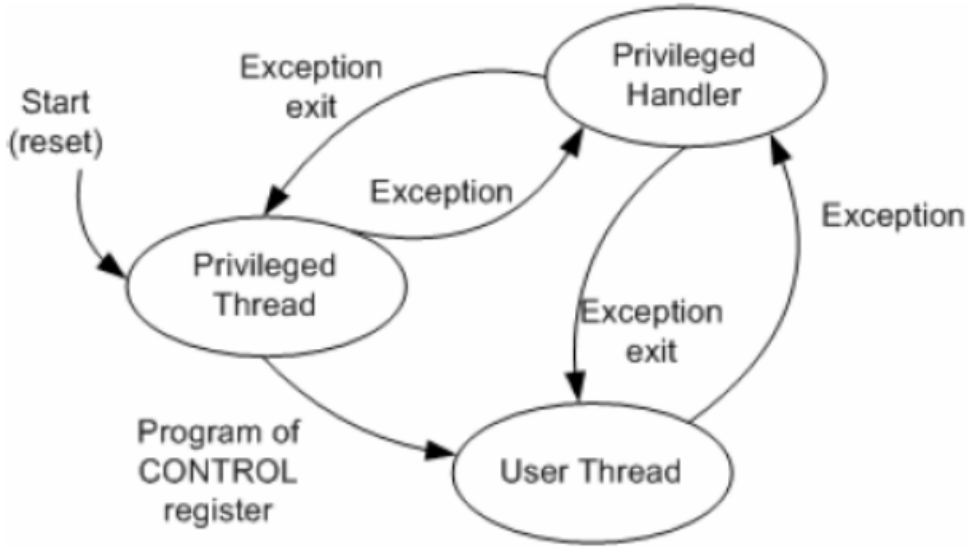


Process SP (handler or thread mode – select in CONTROL reg.)

Main SP (selected at reset – always used in handler mode)

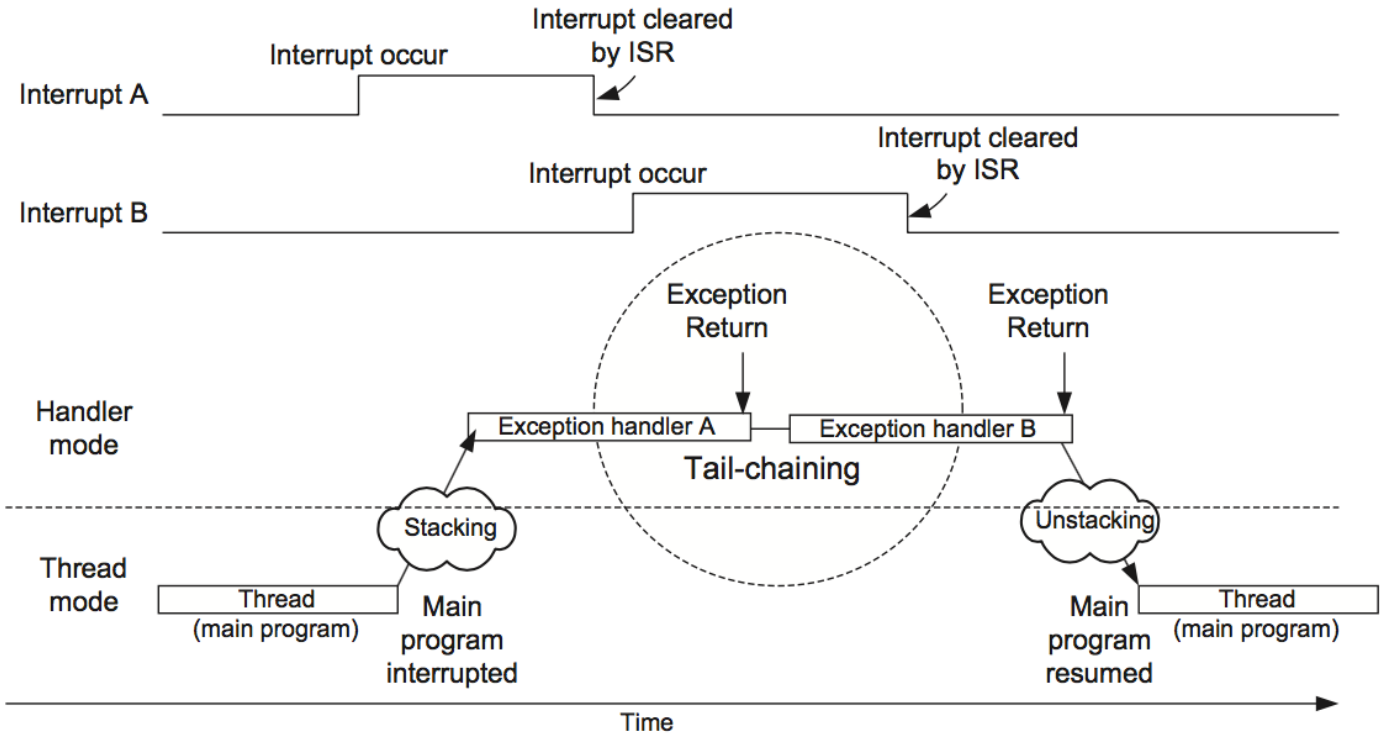
Convention:
PSP in thread mode,
MSP in O/S & handler mode

Cortex-M4 processor operating modes



- Thread mode – normal processing
- Handler mode – interrupt/exception processing
- Privilege levels = User and Privileged
  - Supports basic “security” & memory access protection
  - Supervisor/operating system usually privileged

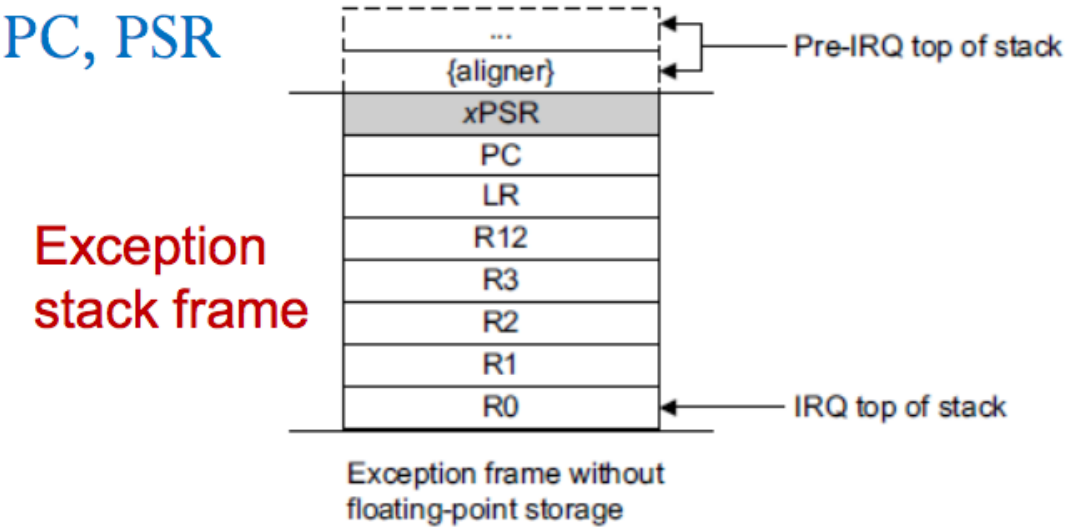
Nested Interrupts



Cortex-M4 interrupts/exceptions

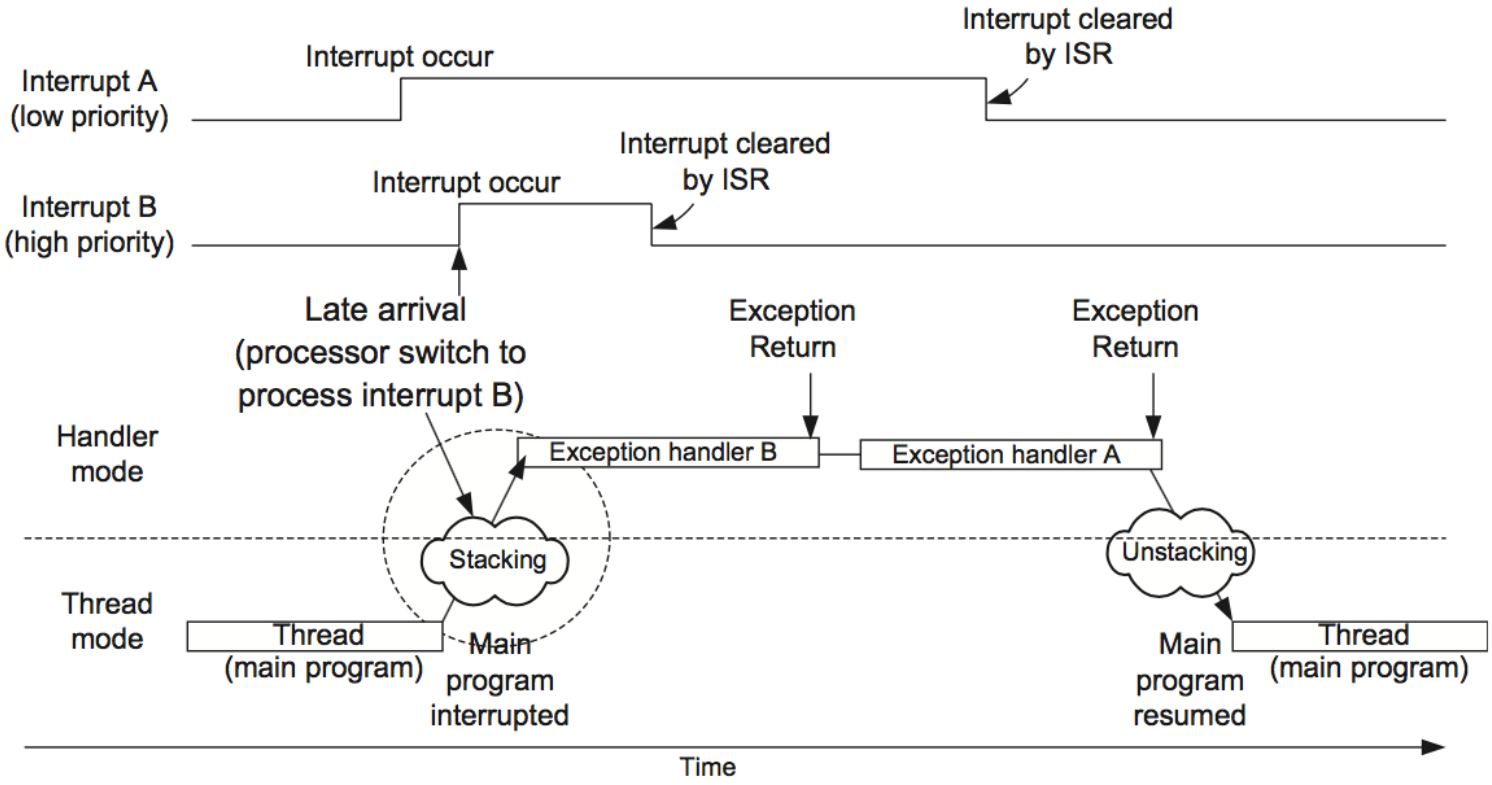
- Interrupts/exceptions managed by *Nested Vectored Interrupt Controller* (NVIC)
- CPU state/context (subset of registers) *saved on the stack*

  R0-R3, R12, LR, PC, PSR



- PC loaded from a vector table, located at 0x0000_0000
  - Vector fetched (Flash memory) while saving state (SRAM)
  - Typical latency = 12 cycles

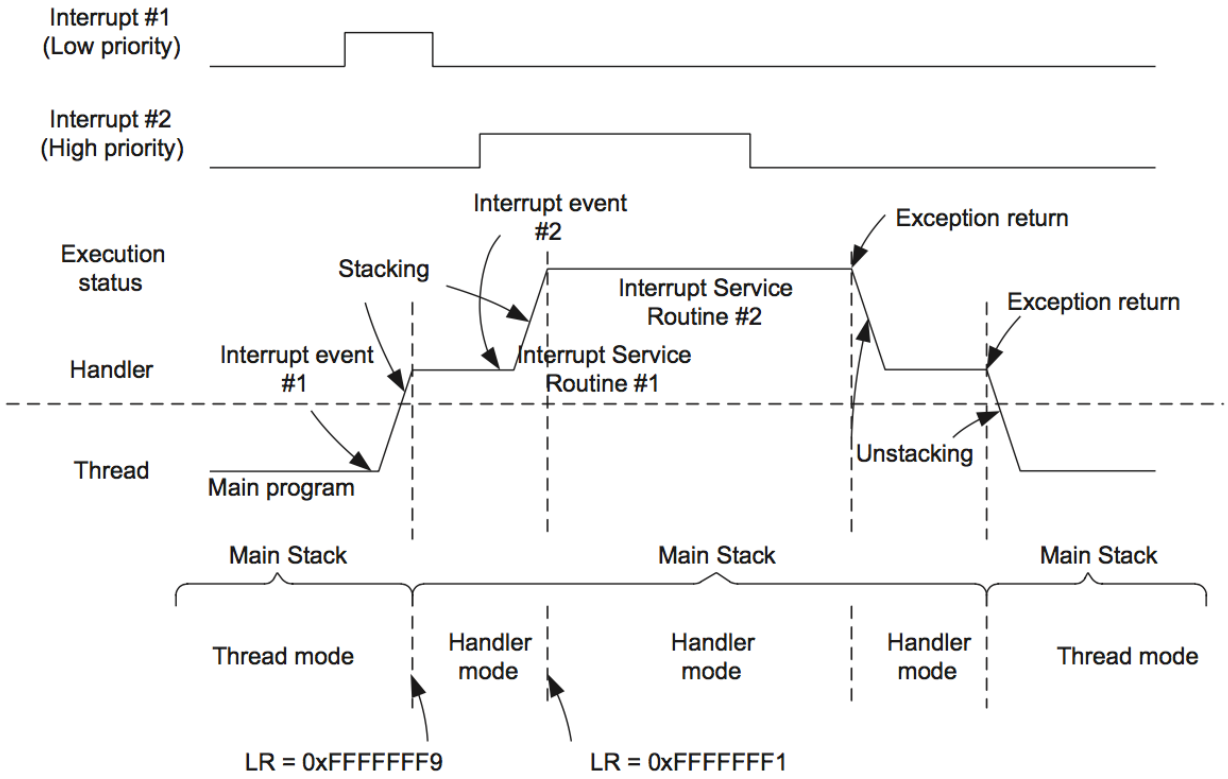
Nested Interrupts (Cont.)



Nested Interrupts (Cont.)

Table 8.3: Valid EXC_RETURN Value

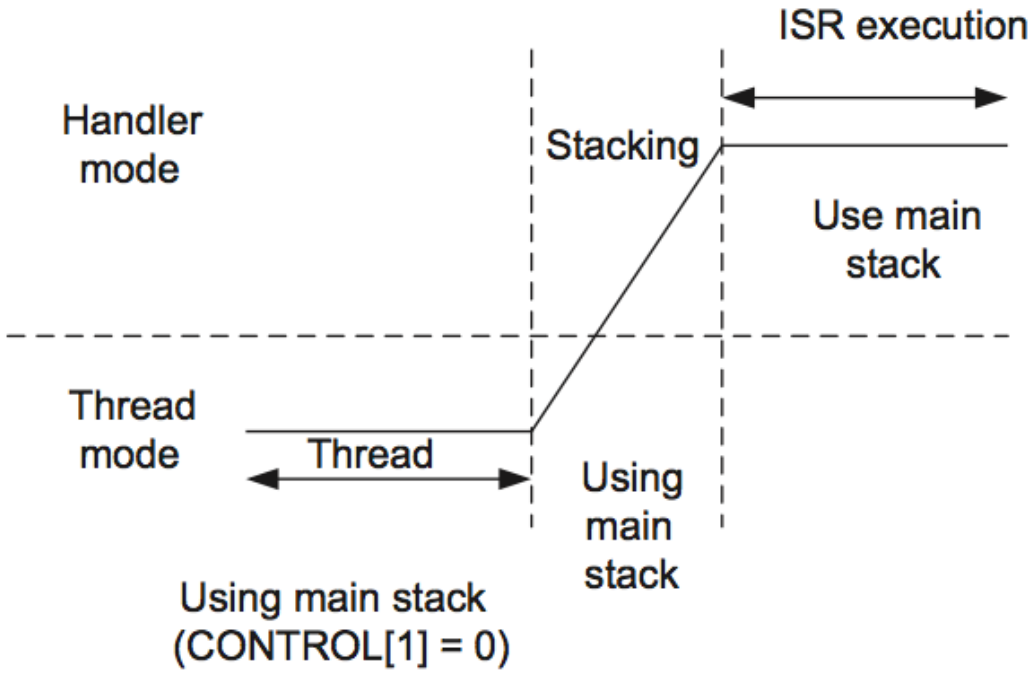
EXC_RETURN	Condition
0xFFFFFFF1	Return to Handler mode (nested exception case)
0xFFFFFFF9	Return to Thread mode and use the main stack for return
0xFFFFFFFD	Return to Thread mode and use the process stack for return



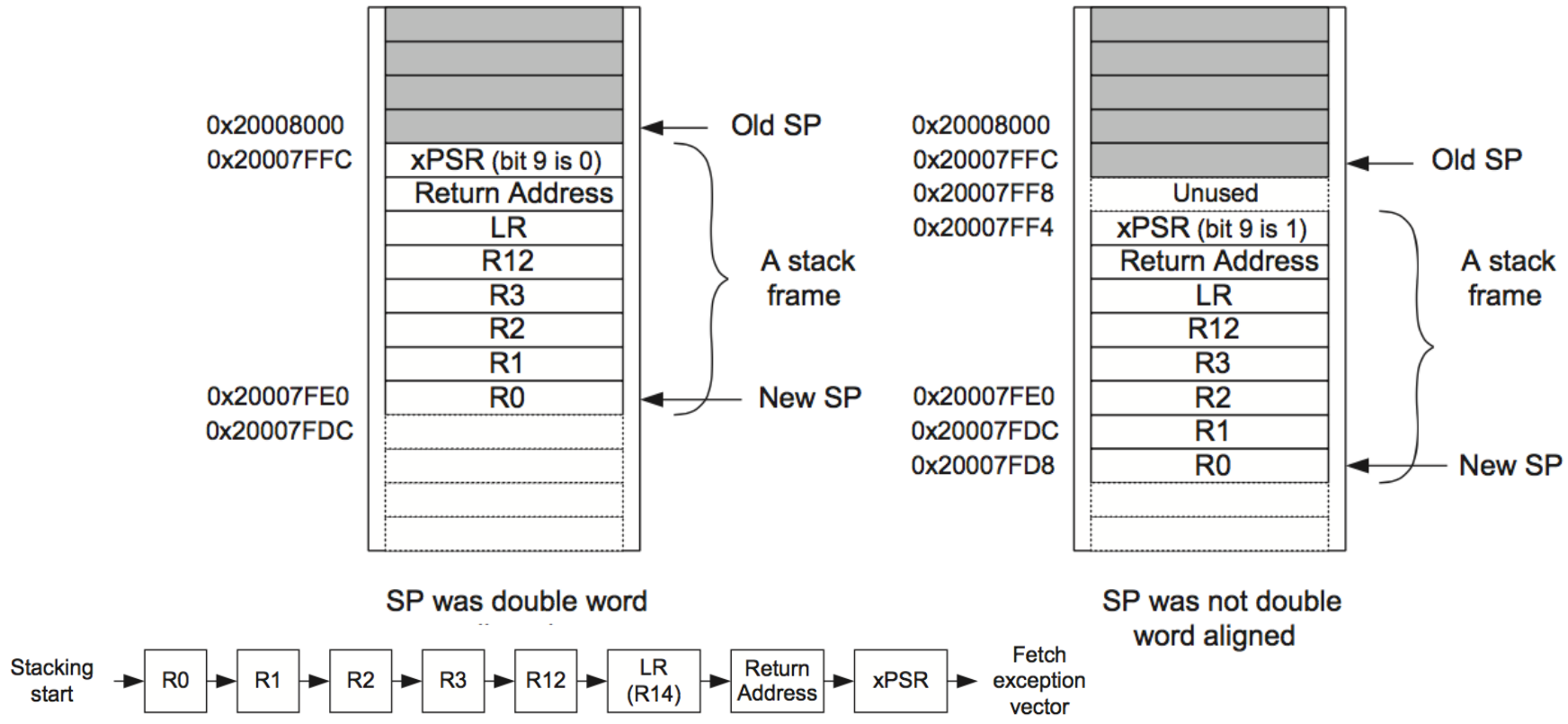
Details of Exception Entry Sequence

- When an exception takes place, a number of things happen:
  - The stack pointer stacks and updates
  - The processor fetches the vector and updates it to the PC
  - The registers update (LR, IPSR, NVIC registers)

Stacking

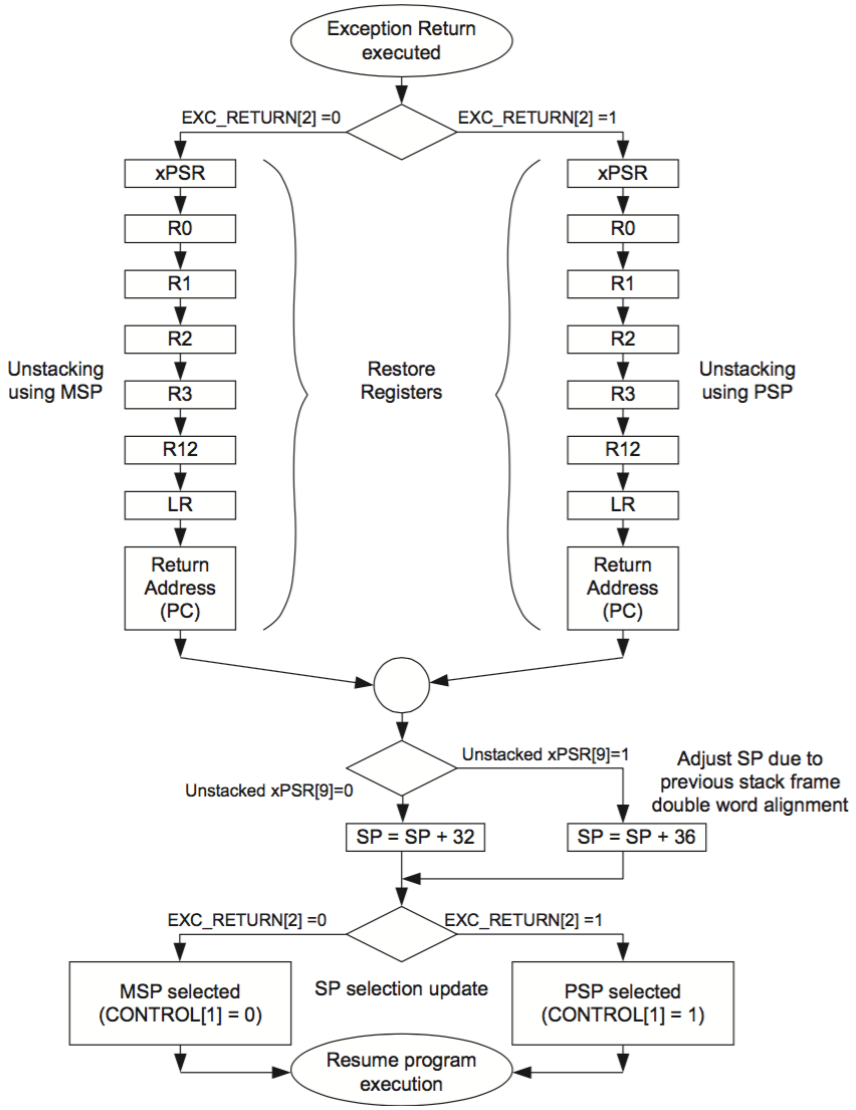

ISR execution
Handler mode
Stacking
Use main stack
Thread mode
Thread
Using main stack
Using main stack (CONTROL[1] = 0)

Double-word Aligned



Details of Exception Exit Sequence

- When an exception return instruction is executed (loading of EXC_RETURN into PC by POP or BX instruction), the exception exit sequence begins. This includes the following steps:
  - Unstacking of registers
  - Fetching and executing from the restored return address


Exception Return executed
EXC_RETURN[2] =0
EXC_RETURN[2] =1
xPSR
R0
R1
R2
R3
R12
LR
Return Address (PC)
Unstacking using MSP
Restore Registers
Unstacking using PSP
Unstacked xPSR[9]=0
Unstacked xPSR[9]=1
Adjust SP due to previous stack frame double word alignment
SP = SP + 32
SP = SP + 36
MSP selected (CONTROL[1] = 0)
SP selection update
PSP selected (CONTROL[1] = 1)
Resume program execution

Interrupt Control and System Control

- The NVIC features include the following:
  - Flexible interrupt management include enable/disable, priority configurations
  - Hardware nested interrupt support
  - Vectored exception entry
  - Interrupt masking

Interrupt Enable Set and Clear Register

Address	Name	Type	Reset Value	Descriptions
0xE000E100	SETENA	R/W	0x00000000	Set enable for interrupt 0 to 31. Write 1 to set bit to 1, write 0 has no effect. Bit[0] for Interrupt #0 (exception #16) Bit[1] for Interrupt #1 (exception #17) … Bit[31] for Interrupt #31 (exception #47) Read value indicates the current enable status
0xE000E180	CLRENA	R/W	0x00000000	Clear enable for interrupt 0 to 31. Write 1 to clear bit to 0, write 0 has no effect. Bit[0] for Interrupt #0 (exception #16) … Bit[31] for Interrupt #31 (exception #47) Read value indicates the current enable status

Interrupt Enable Set and Clear Register

- Enable

```
LDR    R0,=0xE000E100 ;  Setup address in R0
MOVS   R1,#0x4        ;  interrupt #2
STR    R1,[ R0]       ;  write to set interrupt enable
```

- Disable

```
LDR    R0,=0xE000E180;  Setup address in R0
MOVS   R1,#0x4       ;  interrupt #2
STR    R1,[ R0]      ;  write to clear interrupt enable
```

Interrupt Pending and Clear Pending

Address	Name	Type	Reset Value	Descriptions
0xE000E200	SETPEND	R/W	0x00000000	Set pending for interrupt 0 to 31. Write 1 to set bit to 1, write 0 has no effect. Bit[0] for Interrupt #0 (exception #16) Bit[1] for Interrupt #1 (exception #17) … Bit[31] for Interrupt #31 (exception #47) Read value indicates the current pending status
0xE000E280	CLRPEND	R/W	0x00000000	Clear pending for interrupt 0 to 31. Write 1 to clear bit to 0, write 0 has no effect. Bit[0] for Interrupt #0 (exception #16) … Bit[31] for Interrupt #31 (exception #47) Read value indicates the current pending status

Interrupt Pending Set

- The Interrupt Pending status register allows an interrupt to be triggered by software
- If the interrupt is already enabled, no higher-priority exception handler is running, and no interrupt masking is set, then the interrupt service routine will be carried out almost immediately

```
LDR    R0,=0xE000E100 ; Setup address in R0
MOVS   R1,#0x4        ; interrupt #2
STR    R1,[ R0]       ; write to set interrupt enable
LDR    R0,=0xE000E200 ; Setup address in R0
STR    R1,[ R0]       ; write to set pending status
```

Interrupt Pending Clear

- When an interrupt-generating peripheral is being reprogrammed, we can disable the interrupt for this peripheral, reprogram its control registers, and clear the interrupt pending status before re- enabling the peripheral

```
LDR     R0,=0xE000E280 ; Setup address in R0
MOVS    R1,#0x4        ; interrupt #2
STR     R1,[ R0]       ; write to clear pending status
```

Interrupt Priority Level

Bit	31 30	24	23 22	16	15 14	8	7 6	0
0xE000E41C	31		30		29		28	
0xE000E418	27		26		25		24	
0xE000E414	23		22		21		20	
0xE000E410	19		18		17		16	
0xE000E40C	15		14		13		12	
0xE000E408	11		10		9		8	
0xE000E404	7		6		5		4	
0xE000E400	IRQ 3		IRQ 2		IRQ 1		IRQ 0	

Interrupt Priority Level

Address	Name	Type	Reset Value	Descriptions
0xE000E400	IPR0	R/W	0x00000000	Priority level for interrupt 0 to 3 [31:30] Interrupt priority 3 [23:22] Interrupt priority 2 [15:14] Interrupt priority 1 [7:6] Interrupt priority 0
0xE000E404	IPR1	R/W	0x00000000	Priority level for interrupt 4 to 7 [31:30] Interrupt priority 7 [23:22] Interrupt priority 6 [15:14] Interrupt priority 5 [7:6] Interrupt priority 4
0xE000E408	IPR2	R/W	0x00000000	Priority level for interrupt 8 to 11 [31:30] Interrupt priority 11 [23:22] Interrupt priority 10 [15:14] Interrupt priority 9 [7:6] Interrupt priority 8
0xE000E40C	IPR3	R/W	0x00000000	Priority level for interrupt 12 to 15 [31:30] Interrupt priority 15 [23:22] Interrupt priority 14 [15:14] Interrupt priority 13 [7:6] Interrupt priority 12
0xE000E410	IPR4	R/W	0x00000000	Priority level for interrupt 16 to 19
0xE000E414	IPR5	R/W	0x00000000	Priority level for interrupt 20 to 23
0xE000E418	IPR6	R/W	0x00000000	Priority level for interrupt 24 to 27
0xE000E41C	IPR7	R/W	0x00000000	Priority level for interrupt 28 to 31

Interrupt Priority Level Set

- the priority level register will access four of them in one go, if we only want to change one of them, we need to read back the whole

```
LDR     R0,=0xE000E400; Setup address in R0
LDR     R1,[ R0]      ; Get PRIORITY0
MOVS    R2, #0xFF     ; Byte mask
LSLS    R2, R2, #16   ; Shift mask to interrupt #2's position
BICS    R1, R1, R2    ; R1 = R1 AND (NOT(0x00FF0000))
MOVS    R2, #0xC0     ; New value for priority level
LSLS    R2, R2, #16   ; Shift left by 16 bits
ORRS    R1, R1, R2    ; Put new priority level
STR     R1,[ R0]      ; write back value
```

Interrupt Enable and Disable

```
    ;---------------------
    ; Enable IRQ
    ; - input R0 : IRQ number. E.g., IRQ#0 = 0
    ALIGN
nvic_set_enable FUNCTION
    PUSH    { R1, R2}
    LDR     R1,=0xE000E100 ; NVIC SETENA
    MOVS    R2, #1
    LSLS    R2, R2, R0
    STR     R2,[ R1]
    POP     { R1, R2}
    BX      LR    ; Return
    ENDFUNC
    ;---------------------
    ; Disable IRQ
    ; - input R0 : IRQ number. E.g., IRQ#0 = 0
    ALIGN
nvic_clr_enable FUNCTION
    PUSH    { R1, R2}
    LDR     R1,=0xE000E180 ; NVIC CLRENA
    MOVS    R2, #1
    LSLS    R2, R2, R0
    STR     R2,[ R1]
    POP     { R1, R2}
    BX      LR    ; Return
    ENDFUNC
    ;---------------------
```

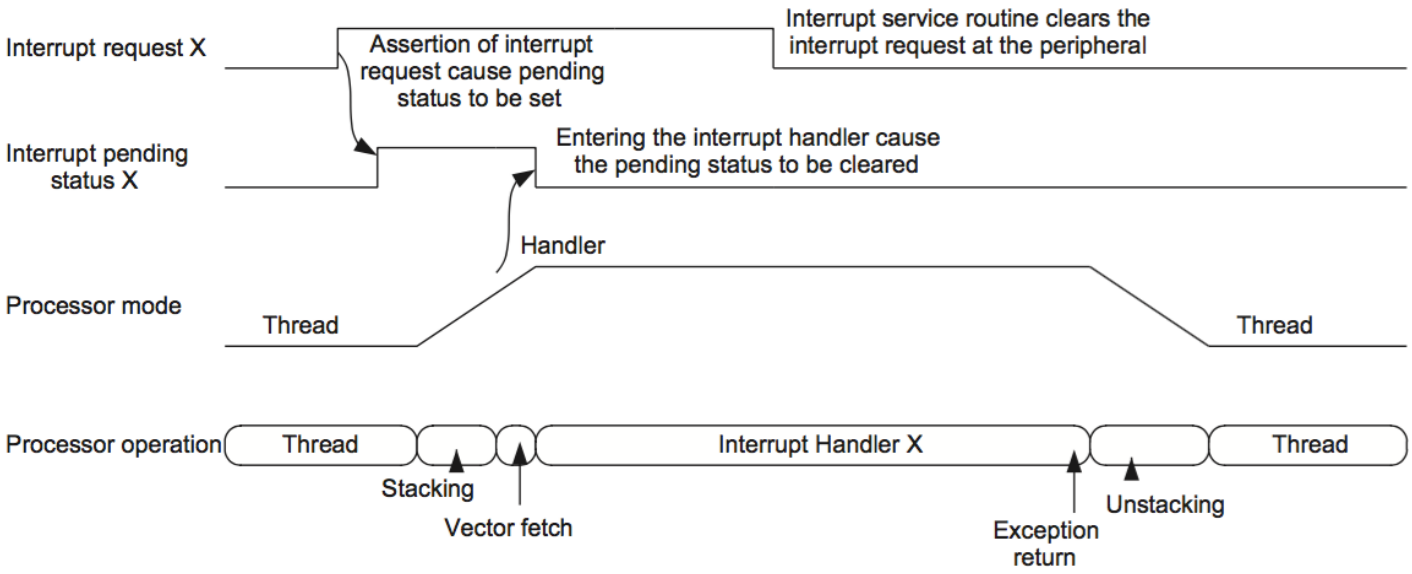
Set and Clear Interrupt Pending Status

```
    ;---------------------
    ; Set IRQ Pending status
    ; - input R0 : IRQ number. E.g., IRQ#0 = 0
    ALIGN
nvic_set_pending FUNCTION
    PUSH    { R1, R2}
    LDR     R1,=0xE000E200 ; NVIC SETPEND
    MOVS    R2, #1
    LSLS    R2, R2, R0
    STR     R2, [ R1]
    POP     { R1, R2}
    BX      LR    ; Return
    ENDFUNC
    ;---------------------
    ; Clear IRQ Pending
    ; - input R0 : IRQ number. E.g., IRQ#0 = 0
    ALIGN
nvic_clr_pending FUNCTION
    PUSH    {R1, R2}
    LDR     R1,=0xE000E280 ; NVIC CLRPEND
    MOVS    R2, #1
    LSLS    R2, R2, R0
    STR     R2, [ R1]
    POP     {R1, R2}
    BX      LR    ; Return
    ENDFUNC
    ;---------------------
```

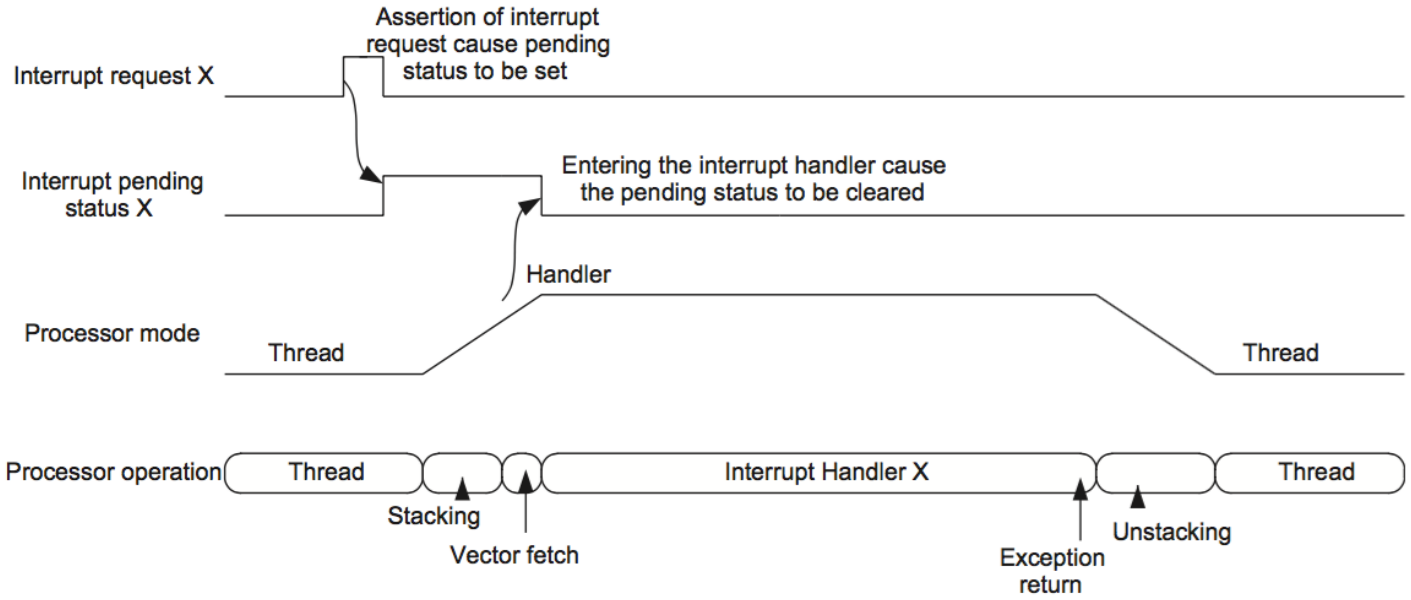
Setting up Interrupt Priority Level

```
    ;--------------------
    ; Set interrupt priority
    ; - input R0 : IRQ number. E.g., IRQ#0 = 0
    ; - input R1 : Priority level
    ALIGN
nvic_set_priority FUNCTION
    PUSH    {R2-R5}
    LDR     R2,=0xE000E400 ; NVIC Interrupt Priority #0
    MOV     R3, R0  ; Make a copy of IRQ number
    MOVS    R4, #3  ; clear lowest two bit of IRQ number
    BICS    R3, R4
    ADDS    R2, R3  ; address of priority register in R2
    ANDS    R4, R0  ; byte number (0 to 3) in priority register
    LSLS    R4, R4, #3 ; Number of bits to shift for priority & mask
    MOVS    R5, #0xFF  ; byte mask
    LSLS    R5, R5, R4 ; byte mask shift to right location
    MOVS    R3, R1
    LSLS    R3, R3, R4 ; Priority shift to right location
    LDR     R4,[R2]    ; Read existing priority level
    BICS    R4, R5     ; Clear existing priority value
    ORRS    R4, R3     ; Set new level
    STR     R4,[R2]    ; Write back
    POP     {R2-R5}
    BX      LR         ; Return
    ENDFUNC
    ;--------------------
```

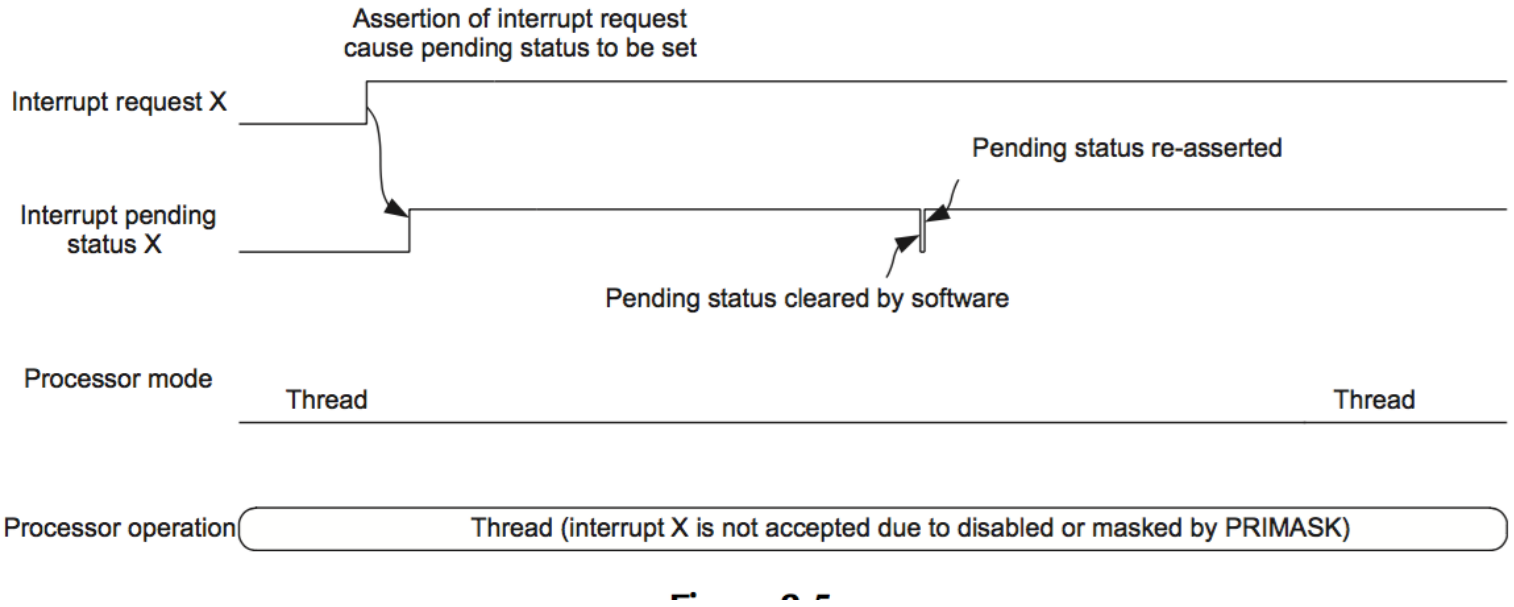
Simple Interrupt Process



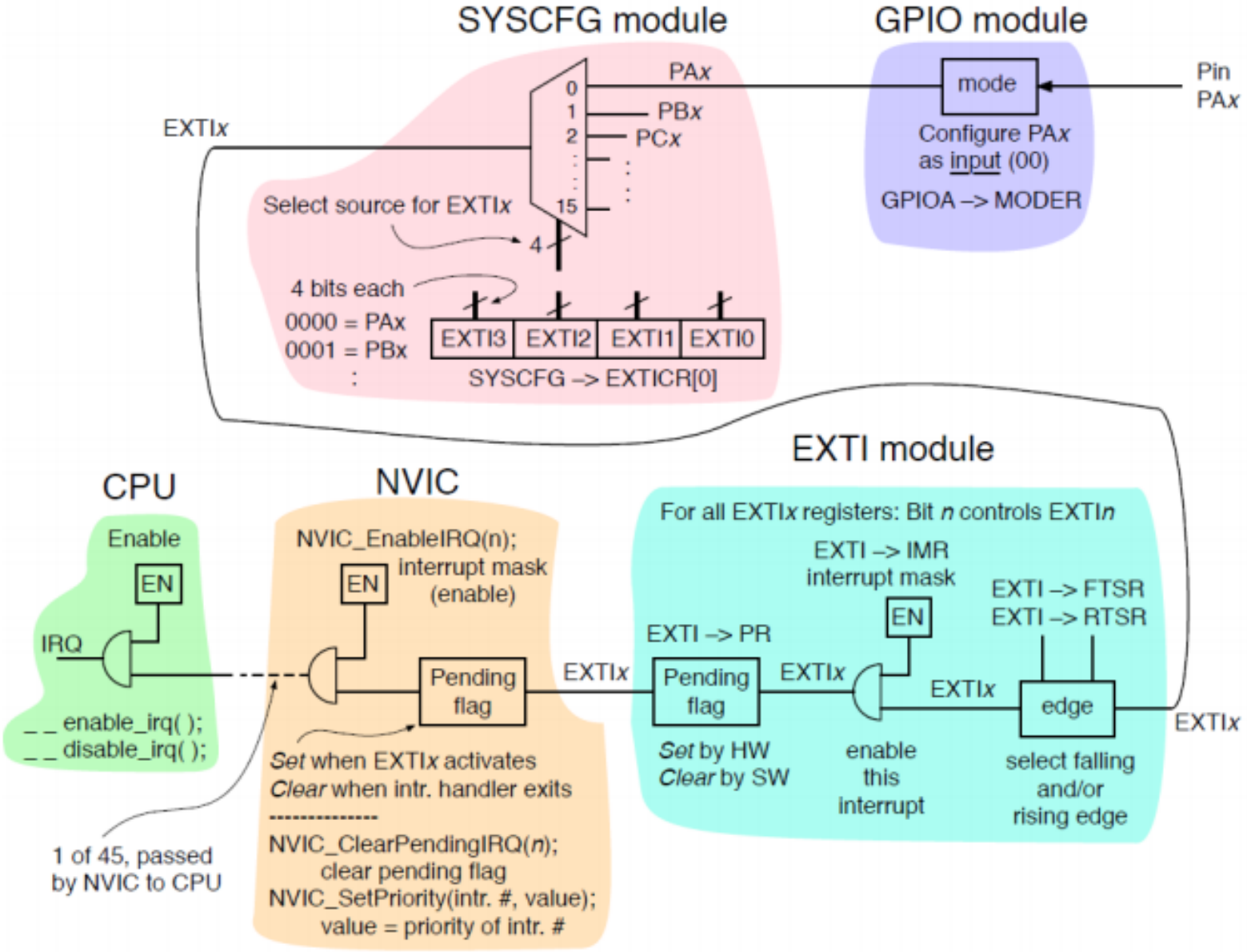
Pulsed Interrupt Activation

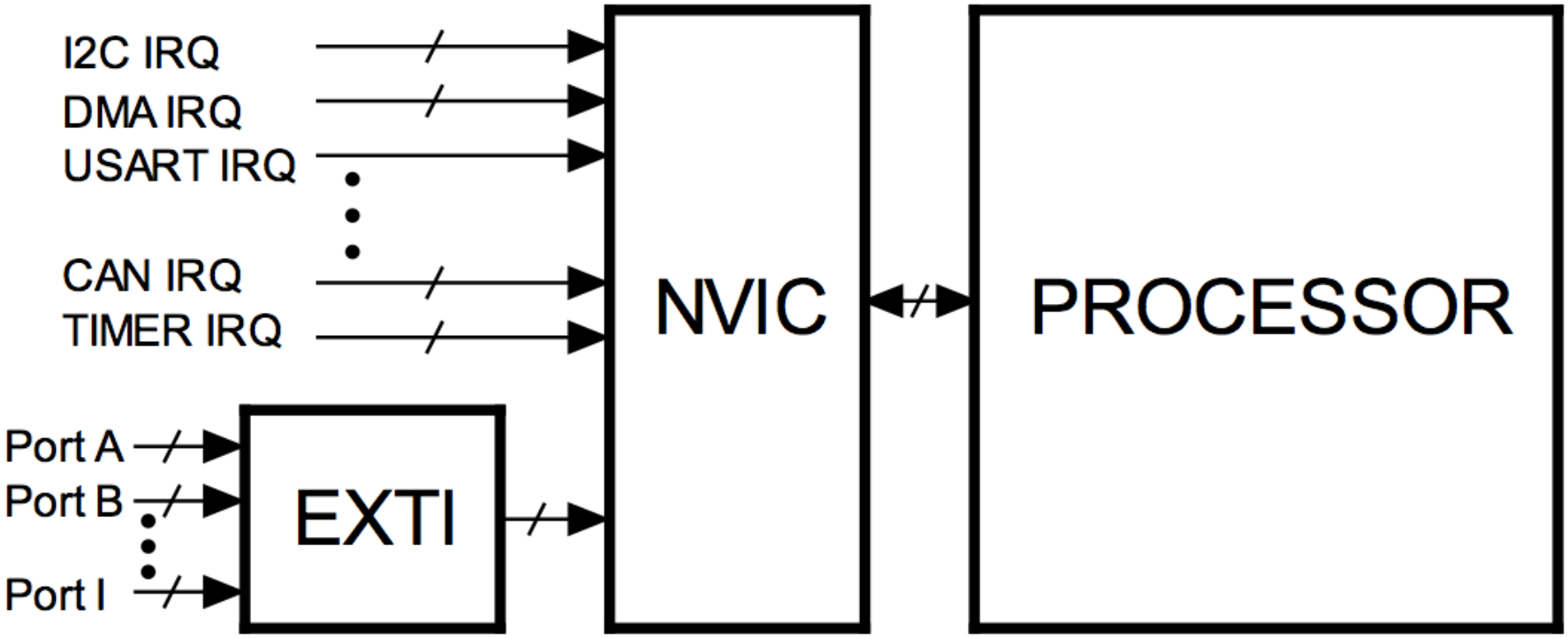


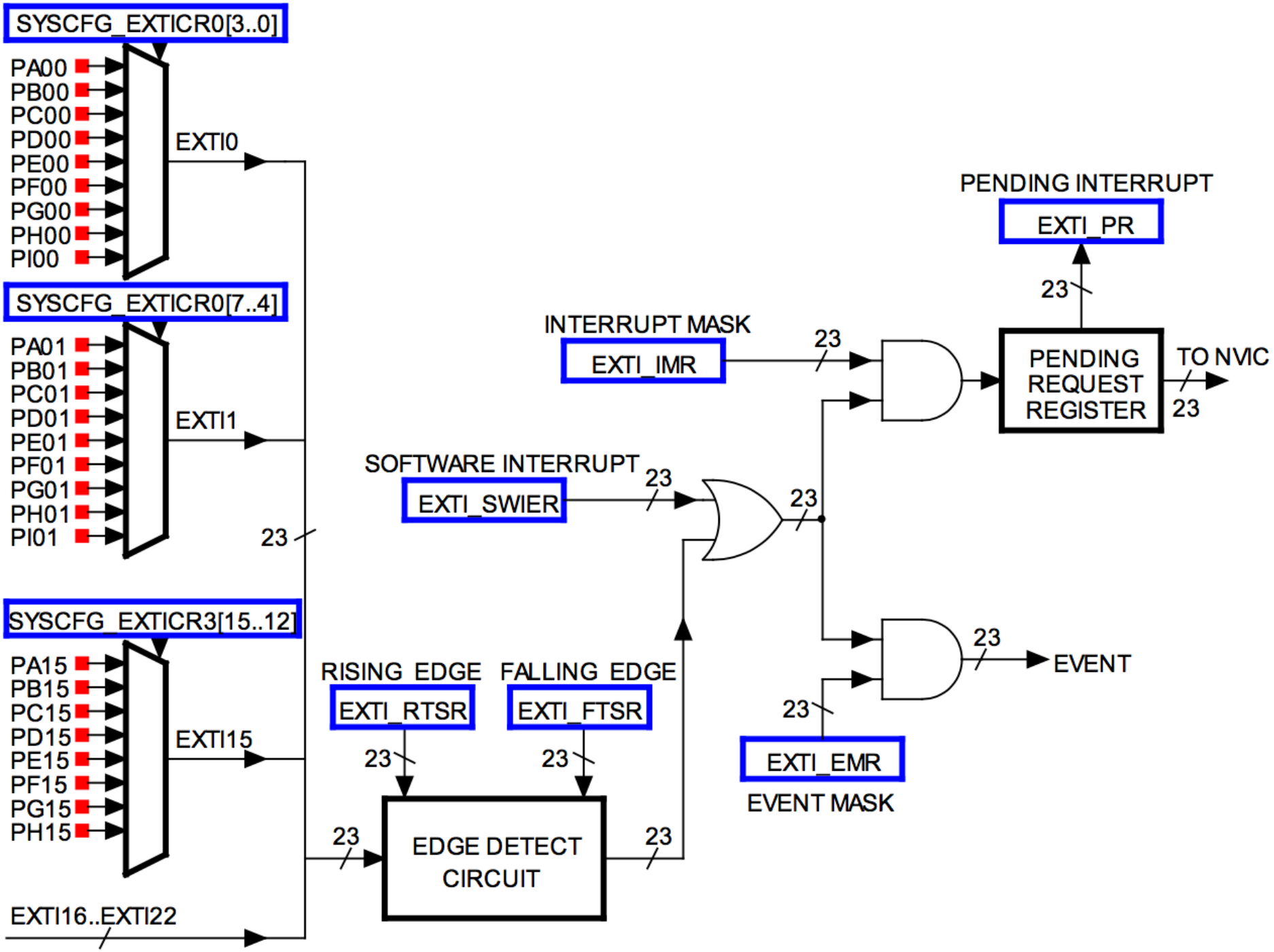
Interrupt Pending Status Reasserted



Signal flow/setup for External Interrupt EXTI*x*, *x* = 0...15




I2C IRQ
DMA IRQ
USART IRQ
CAN IRQ
TIMER IRQ
Port A
Port B
Port I
EXTI
NVIC
PROCESSOR


SYSCFG_EXTICR0[3..0]
PA00
PB00
PC00
PD00
PE00
PF00
PG00
PH00
PI00
EXTI0
SYSCFG_EXTICR0[7..4]
PA01
PB01
PC01
PD01
PE01
PF01
PG01
PH01
PI01
EXTI1
23
SYSCFG_EXTICR3[15..12]
PA15
PB15
PC15
PD15
PE15
PF15
PG15
PH15
EXTI15
EXTI16..EXTI22
RISING EDGE
EXTI_RTSR
FALLING EDGE
EXTI_FTSR
23
23
23
EDGE DETECT CIRCUIT
23
SOFTWARE INTERRUPT
EXTI_SWIER
23
23
INTERRUPT MASK
EXTI_IMR
23
PENDING INTERRUPT
EXTI_PR
23
PENDING REQUEST REGISTER
TO NVIC
23
23
EVENT
23
EXTI_EMR
EVENT MASK

System configuration controller (SYSCFG)

SYSCFG main features

The STM32L4x6 devices feature a set of configuration registers. The main purposes of the system configuration controller are the following:

- Remapping memory areas
- Managing the external interrupt line connection to the GPIOs
- Managing robustness feature
- Setting SRAM2 write protection and software erase
- Configuring FPU interrupts
- Enabling the firewall
- Enabling /disabling I^2C Fast-mode Plus driving capability on some I/Os and voltage booster for I/Os analog switches.

6.4.21 APB2 peripheral clock enable register (RCC_APB2ENR)

Address: 0x60

Reset value: 0x0000 0000

Access: word, half-word and byte access

Note: When the peripheral clock is not active, the peripheral registers read or write access is not supported.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	DFSDM1EN	Res.	SAI2EN	SAI1EN	Res.	Res.	TIM17EN	TIM16EN	TIM15EN
							rw		rw	rw			rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	USART1EN	TIM8EN	SPI1EN	TIM1EN	SDMMC1EN	Res.	Res.	FWEN	Res.	Res.	Res.	Res.	Res.	Res.	SYSCFGEN
	rw	rw	rw	rw	rw			rs							rw

8.2.3 SYSCFG external interrupt configuration register 1 (SYSCFG_EXTICR1)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	EXTI3[2:0]			Res	EXTI2[2:0]			Res	EXTI1[2:0]			Res	EXTI0[2:0]		
	rw	rw	rw		rw	rw	rw		rw	rw	rw		rw	rw	rw

Bits 31:15 Reserved, must be kept at reset value.

Bits 14:12 **EXTI3[2:0]**: EXTI 3 configuration bits

These bits are written by software to select the source input for the EXTI3 external interrupt.

000: PA[3] pin
001: PB[3] pin
010: PC[3] pin
011: PD[3] pin
100: PE[3] pin
101: PF[3] pin
110: PG[3] pin
111: Reserved

Bit 11 Reserved, must be kept at reset value.

EXTI

Extended interrupts and events controller (EXTI)

Introduction

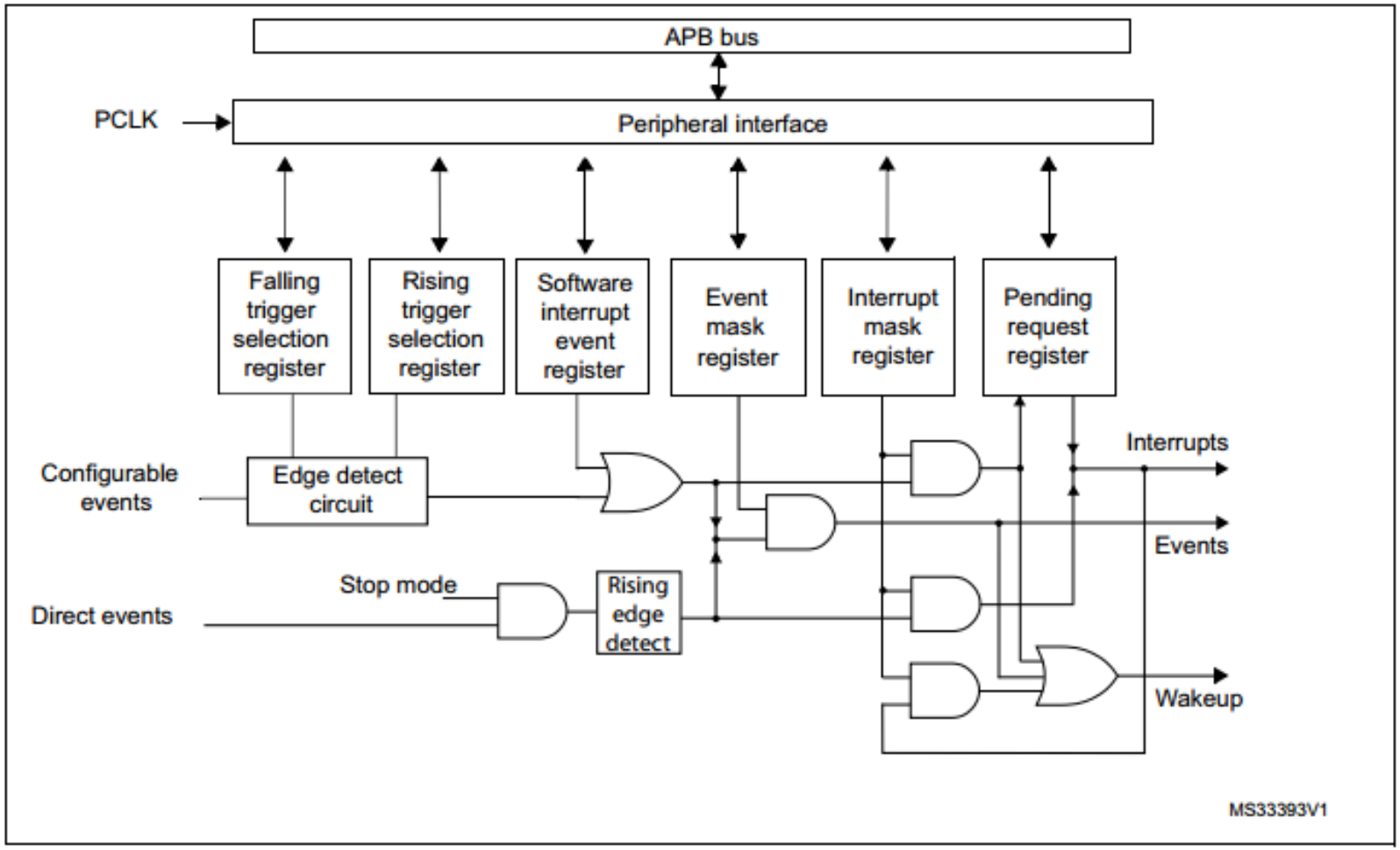
The EXTI main features are as follows:

- Generation of up to 40 event/interrupt requests
  - 26 configurable lines
  - 14 direct lines
- Independent mask on each event/interrupt line
- Configurable rising or falling edge (configurable lines only)
- Dedicated status bit (configurable lines only)
- Emulation of event/interrupt requests (configurable lines only)

12.3.1 EXTI block diagram

The extended interrupt/event block diagram is shown on *Figure 26*.

Figure 26. Configurable interrupt/event block diagram



12.3.4 Hardware interrupt selection

To configure a line as an interrupt source, use the following procedure:

1. Configure the corresponding mask bit in the EXTI_IMR register.
2. Configure the Trigger Selection bits of the Interrupt line (EXTI_RTSR and EXTI_FTSR).
3. Configure the enable and mask bits that control the NVIC IRQ channel mapped to the EXTI so that an interrupt coming from one of the EXTI lines can be correctly acknowledged.

Note: The direct lines do not require any EXTI configuration.

12.3.5 Hardware event selection

To configure a line as an event source, use the following procedure:

1. Configure the corresponding mask bit in the EXTI_EMR register.
2. Configure the Trigger Selection bits of the Event line (EXTI_RTSR and EXTI_FTSR).

12.3.6 Software interrupt/event selection

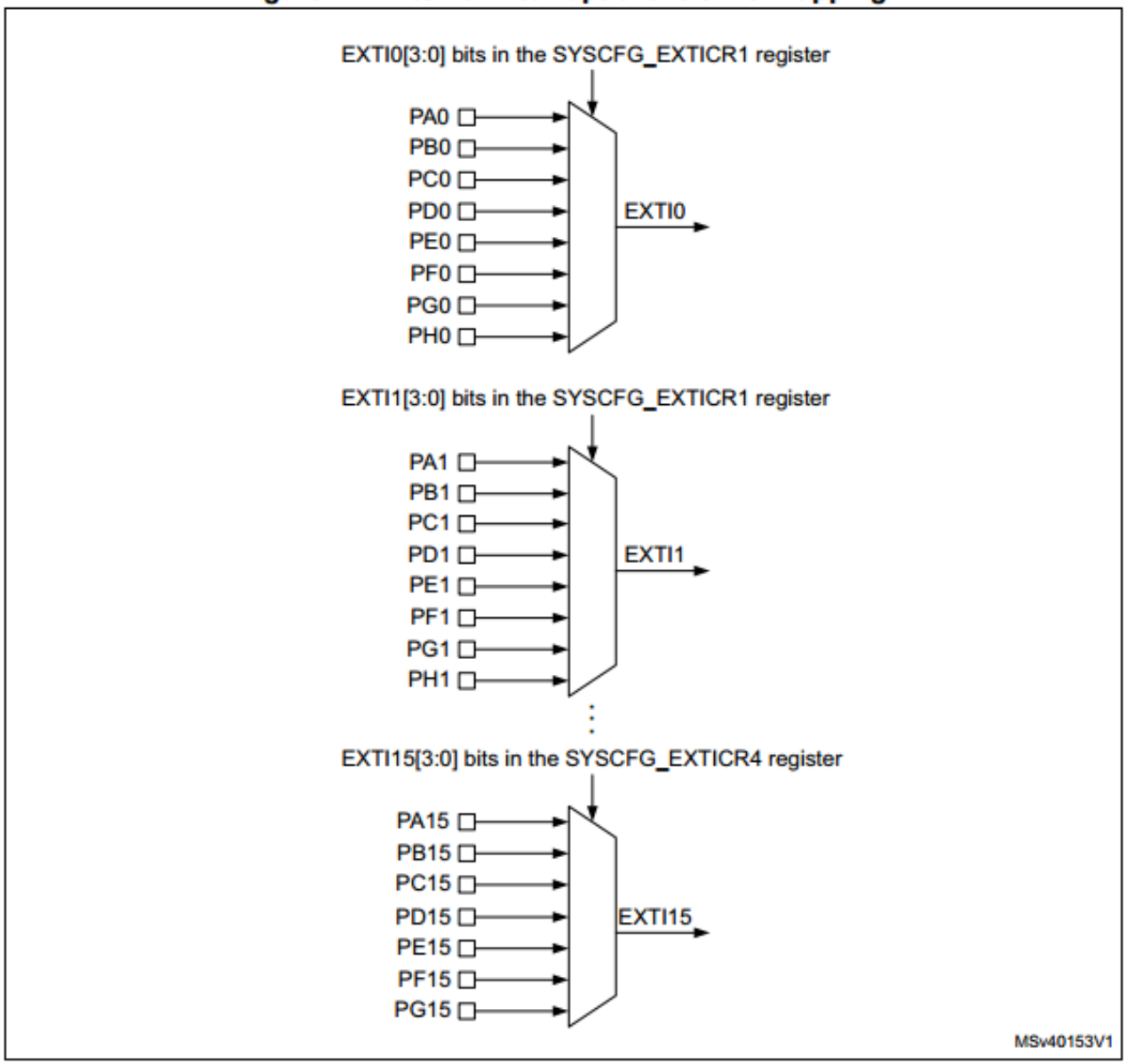
Any of the configurable lines can be configured as a software interrupt/event line. The procedure to generate a software interrupt is as follows:

1. Configure the corresponding mask bit (EXTI_IMR, EXTI_EMR).
2. Set the required bit of the software interrupt register (EXTI_SWIER).

12.4 EXTI interrupt/event line mapping

In the STM32L4x6, 40 interrupt/event lines are available. The GPIOs are connected to 16 configurable interrupt/event lines (see *Figure 27*).

Figure 27. External interrupt/event GPIO mapping



EXTI line	Line source[1]	Line type
0-15	GPIO	configurable
16	PVD	configurable
17	OTG_FS wakeup event[2]	direct
18	RTC alarms	configurable
19	RTC tamper or timestamp or CSS_LSE	configurable
20	RTC wakeup timer	configurable
21	COMP1 output	configurable
22	COMP2 output	configurable
23	I2C1 wakeup[2]	direct
24	I2C2 wakeup[2]	direct
25	I2C3 wakeup	direct
26	USART1 wakeup[2]	direct
27	USART2 wakeup[2]	direct
28	USART3 wakeup[2]	direct
29	UART4 wakeup[2]	direct
30	UART5 wakeup[2]	direct
31	LPUART1 wakeup	direct
32	LPTIM1	direct
33	LPTIM2[2]	direct
34	SWPMI1 wakeup[2]	direct
35	PVM1 wakeup	configurable
36	PVM2 wakeup	configurable
37	PVM3 wakeup	configurable
38	PVM4 wakeup	configurable
39	LCD wakeup	direct

12.5.1 Interrupt mask register 1 (EXTI_IMR1)

Address offset: 0x00

Reset value: 0xFF82 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IM31	IM30	IM29	IM28	IM27	IM26	IM25	IM24	IM23	IM22	IM21	IM20	IM19	IM18	IM17	IM16
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IM15	IM14	IM13	IM12	IM11	IM10	IM9	IM8	IM7	IM6	IM5	IM4	IM3	IM2	IM1	IM0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **IMx:** Interrupt Mask on line x (x = 31 to 0)

0: Interrupt request from Line x is masked

1: Interrupt request from Line x is not masked

Note: The reset value for the direct lines (line 17, lines from 23 to 34, line 39) is set to '1' in order to enable the interrupt by default.

12.5.2 Event mask register 1 (EXTI_EMR1)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EM31	EM30	EM29	EM28	EM27	EM26	EM25	EM24	EM23	EM22	EM21	EM20	EM19	EM18	EM17	EM16
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EM15	EM14	EM13	EM12	EM11	EM10	EM9	EM8	EM7	EM6	EM5	EM4	EM3	EM2	EM1	EM0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **EMx:** Event mask on line x (x = 31 to 0)

0: Event request from line x is masked

1: Event request from line x is not masked

12.5.3 Rising trigger selection register 1 (EXTI_RTSR1)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RT22	RT21	RT20	RT19	RT18	Res.	RT16
									rw	rw	rw	rw	rw		rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RT15	RT14	RT13	RT12	RT11	RT10	RT9	RT8	RT7	RT6	RT5	RT4	RT3	RT2	RT1	RT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:23 Reserved, must be kept at reset value.

Bits 22:18 **RTx:** Rising trigger event configuration bit of line x (x = 22 to 18)
0: Rising trigger disabled (for Event and Interrupt) for input line
1: Rising trigger enabled (for Event and Interrupt) for input line

Bit 17 Reserved, must be kept at reset value.

Bits 16:0 **RTx:** Rising trigger event configuration bit of line x (x = 16 to 0)
0: Rising trigger disabled (for Event and Interrupt) for input line
1: Rising trigger enabled (for Event and Interrupt) for input line

Note: The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set.

Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

12.5.4 Falling trigger selection register 1 (EXTI_FTSR1)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	FT22	FT21	FT20	FT19	FT18	Res.	FT16
									rw	rw	rw	rw	rw		rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FT15	FT14	FT13	FT12	FT11	FT10	FT9	FT8	FT7	FT6	FT5	FT4	FT3	FT2	FT1	FT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 22:18 **FTx:** Falling trigger event configuration bit of line x (x = 22 to 18)
0: Falling trigger disabled (for Event and Interrupt) for input line
1: Falling trigger enabled (for Event and Interrupt) for input line

Bit 17 Reserved, must be kept at reset value.

Bits 16:0 **FTx:** Falling trigger event configuration bit of line x (x = 16 to 0)
0: Falling trigger disabled (for Event and Interrupt) for input line
1: Falling trigger enabled (for Event and Interrupt) for input line

Note: The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a falling edge on a configurable interrupt line occurs during a write operation to the EXTI_FTSR register, the pending bit is not set.

Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.

12.5.5 Software interrupt event register 1 (EXTI_SWIER1)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SWI 22	SWI 21	SWI 20	SWI 19	SWI 18	Res.	SWI 16
									rw	rw	rw	rw	rw		rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWI 15	SWI 14	SWI 13	SWI 12	SWI 11	SWI 10	SWI 9	SWI 8	SWI 7	SWI 6	SWI 5	SWI 4	SWI 3	SWI 2	SWI 1	SWI 0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:23 Reserved, must be kept at reset value.

Bits 22: 18 **SWIx:** Software interrupt on line x (x = 22 o 18)

If the interrupt is enabled on this line in the EXTI_IMR, writing a '1' to this bit when it is at '0' sets the corresponding pending bit in EXTI_PR resulting in an interrupt request generation.

This bit is cleared by clearing the corresponding bit in the EXTI_PR register (by writing a ‘1' into the bit).

Bit 17 Reserved, must be kept at reset value.

Bits 16:0 **SWIx:** Software interrupt on line x (x = 16 to 0)

If the interrupt is enabled on this line in the EXTI_IMR, writing a '1' to this bit when it is at '0' sets the corresponding pending bit in EXTI_PR resulting in an interrupt request generation.

This bit is cleared by clearing the corresponding bit of EXTI_PR (by writing a ‘1' into the bit).

12.5.6 Pending register 1 (EXTI_PR1)

Address offset: 0x14

Reset value: undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PIF22	PIF21	PIF20	PIF19	PIF18	Res.	PIF16
									rc_w1	rc_w1	rc_w1	rc_w1	rc_w1		rc_w1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PIF15	PIF14	PIF13	PIF12	PIF11	PIF10	PIF9	PIF8	PIF7	PIF6	PIF5	PIF4	PIF3	PIF2	PIF1	PIF0
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:23 Reserved, must be kept at reset value.

Bits 22:18 **PIFx:** Pending interrupt flag on line x (x = 22 to 18)

0: No trigger request occurred

1: Selected trigger request occurred

This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a ‘1’ to the bit.

Bit 17 Reserved, must be kept at reset value.

Bits 16:0 **PIFx:** Pending interrupt flag on line x (x = 16 to 0)

0: No trigger request occurred

1: Selected trigger request occurred

This bit is set when the selected edge event arrives on the interrupt line. This bit is cleared by writing a ‘1’ to the bit.

Nested Vectored Interrupt Controller (NVIC)

NVIC main features：

- Supports up to 240 interrupts but 82 maskable interrupt channels in stm32l4
- Low-latency exception and interrupt handling
- Power management control
- Implementation of System Control Registers

All interrupts including the core exceptions are managed by the NVIC.

All interrupts including the core exceptions are managed the NVIC.

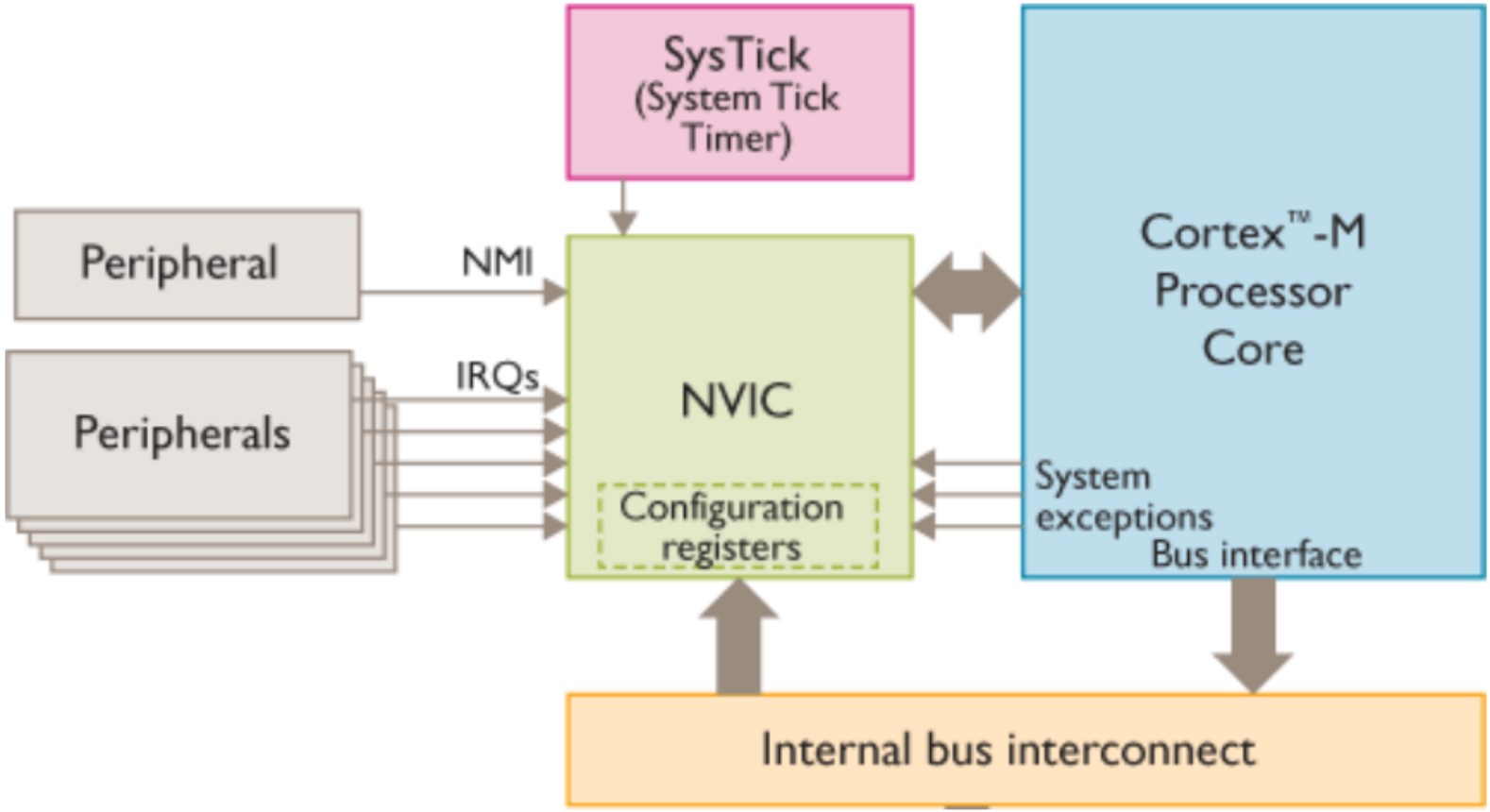
Hardware and software control of interrupts

The Cortex-M4 latches all interrupts. A peripheral interrupt becomes pending for one of the following reasons:

- the NVIC detects that the interrupt signal is HIGH and the interrupt is not active
- the NVIC detects a rising edge on the interrupt signal
- software writes to the corresponding interrupt set-pending register bit

Nested Vectored Interrupt Controller

- NVIC manages and prioritizes external interrupts in Cortex-M
  - 82 IRQ sources from STM32F4xx peripherals
- NVIC interrupts CPU with IRQ# of highest-priority IRQ signal
  - CPU uses IRQ# to access the vector table & get intr. handler start address



NVIC registers (one bit for each IRQ#)

- NVIC_ISERx/NVIC_ICERx
  - Each IRQ has its own **enable** bit within NVIC
  - Interrupt Set/Clear Enable Register
  - 1 = Set (enable) interrupt/Clear (disable) interrupt
- NVIC_ISPRx/NVIC_ICPRx
  - Interrupt Set/Clear Pending Register
  - Read 1 from ISPR if interrupt in pending state
  - Write 1 to set interrupt to pending or clear from pending state
- NVIC_IABRx – Interrupt Active Bit Register
  - Read 1 if interrupt in active state

EnableK

PendK

PriorityK

x = 0..7 for each register type, with 32 bits per register, to support up to 240 IRQs (82 in STM32F4xx)

- Each bit controls one interrupt, identified by its IRQ# (0..239)
- Register# x = IRQ# DIV 32
- Bit n in the register = IRQ# MOD 32

NVIC registers (continued)

- NVIC_IPRx (x=0..59) – Interrupt Priority Registers
  - Supports up to 240 interrupts: 0..239 (82 in STM32F4)
  - 8-bit priority field for each interrupts (4-bit field in STM32F4)
    - 4 priority values per register (STM32F4 – upper 4 bits of each byte)
    - 0 = highest priority
    - Register# x = IRQ# DIV 4
    - Byte offset within the register = IRQ# MOD 4
    - Ex. IRQ85:
      - 85/4 = 21 with remainder 1 (register 21, byte offset 1)
        Write priority<<8 to NVIC_IPR2
      - 85/32 = 2 with remainder 21: write 1<<21 to NVIC_SER2

- STIR – Software Trigger Interrupt Register
  - Write IRQ# (0..239) to trigger that interrupt from software
  - Unprivileged access to this register enabled in system control register (SCR)

```
typedef struct
{
  __IOM uint32_t ISER[8U];               /*!< Offset: 0x000 (R/W)  Interrupt Set Enable Register */
        uint32_t RESERVED0[24U];
  __IOM uint32_t ICER[8U];               /*!< Offset: 0x080 (R/W)  Interrupt Clear Enable Register */
        uint32_t RSERVED1[24U];
  __IOM uint32_t ISPR[8U];               /*!< Offset: 0x100 (R/W)  Interrupt Set Pending Register */
        uint32_t RESERVED2[24U];
  __IOM uint32_t ICPR[8U];               /*!< Offset: 0x180 (R/W)  Interrupt Clear Pending Register */
        uint32_t RESERVED3[24U];
  __IOM uint32_t IABR[8U];               /*!< Offset: 0x200 (R/W)  Interrupt Active bit Register */
        uint32_t RESERVED4[56U];
  __IOM uint8_t  IP[240U];               /*!< Offset: 0x300 (R/W)  Interrupt Priority Register (8Bit wide) */
        uint32_t RESERVED5[644U];
  __OM  uint32_t STIR;                   /*!< Offset: 0xE00 ( /W)  Software Trigger Interrupt Register */
}  NVIC_Type;
```

In startup_stm32l476xx.S

```
.word   WWDG_IRQHandler
.word   PVD_PVM_IRQHandler
.word   TAMP_STAMP_IRQHandler
.word   RTC_WKUP_IRQHandler
.word   FLASH_IRQHandler
.word   RCC_IRQHandler
.word   EXTI0_IRQHandler
.word   EXTI1_IRQHandler
.word   EXTI2_IRQHandler
.word   EXTI3_IRQHandler
.word   EXTI4_IRQHandler
.word   DMA1_Channel1_IRQHandler
.word   DMA1_Channel2_IRQHandler
.word   DMA1_Channel3_IRQHandler
.word   DMA1_Channel4_IRQHandler
.word   DMA1_Channel5_IRQHandler
.word   DMA1_Channel6_IRQHandler
.word   DMA1_Channel7_IRQHandler
.word   ADC1_2_IRQHandler
.word   CAN1_TX_IRQHandler
.word   CAN1_RX0_IRQHandler
.word   CAN1_RX1_IRQHandler
.word   CAN1_SCE_IRQHandler
.word   EXTI9_5_IRQHandler
.word   TIM1_BRK_TIM15_IRQHandler
.word   TIM1_UP_TIM16_IRQHandler
.word   TIM1_TRG_COM_TIM17_IRQHandler
.word   TIM1_CC_IRQHandler
.word   TIM2_IRQHandler
.word   TIM3_IRQHandler
```

SysTick Timer

24-bit count down SysTick timer

Address	Name	Type	Required privilege	Reset value	Description
0xE000E010	SYST_CSR	RW	Privileged	a	*SysTick Control and Status Register*
0xE000E014	SYST_RVR	RW	Privileged	Unknown	*SysTick Reload Value Register*
0xE000E018	SYST_CVR	RW	Privileged	Unknown	*SysTick Current Value Register*
0xE000E01C	SYST_CALIB	RO	Privileged	-[a]	*SysTick Calibration Value Register*

SysTick Control and Status Register

The SysTick SYST_CSR register enables the SysTick features. The register resets to 0x00000000, or to 0x00000004

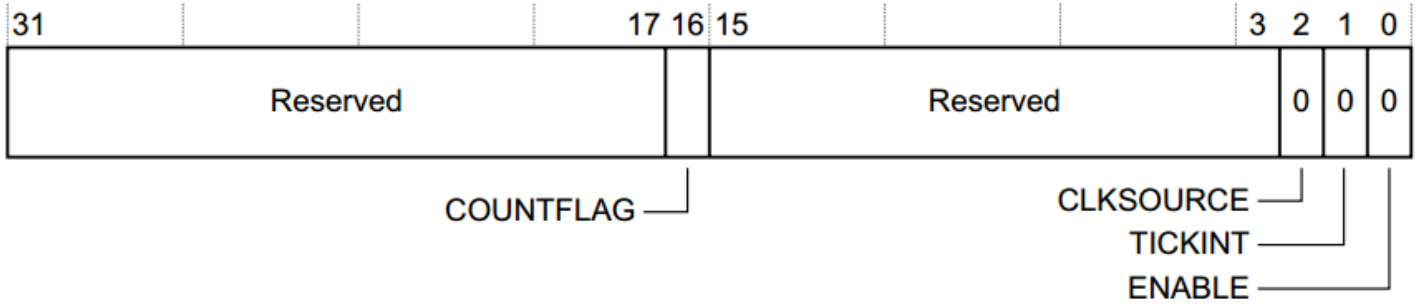


Table 4-33 SysTick SYST_CSR register bit assignments

Bits	Name	Function
[31:17]	-	Reserved.
[16]	COUNTFLAG	Returns 1 if timer counted to 0 since last time this was read.
[15:3]	-	Reserved.

Bits	Name	Function
[2]	CLKSOURCE	Indicates the clock source: 0 = external clock 1 = processor clock.
[1]	TICKINT	Enables SysTick exception request: 0 = counting down to zero does not assert the SysTick exception request 1 = counting down to zero asserts the SysTick exception request. Software can use COUNTFLAG to determine if SysTick has ever counted to zero.
[0]	ENABLE	Enables the counter: 0 = counter disabled 1 = counter enabled.

When ENABLE is set to 1, the counter loads the RELOAD value from the SYST_RVR register and then counts down. On reaching 0, it sets the COUNTFLAG to 1 and optionally asserts the SysTick depending on the value of TICKINT. It then loads the RELOAD value again, and begins counting.

SysTick Reload Value Register

The SYST_RVR register specifies the start value to load into the SYST_CVR register

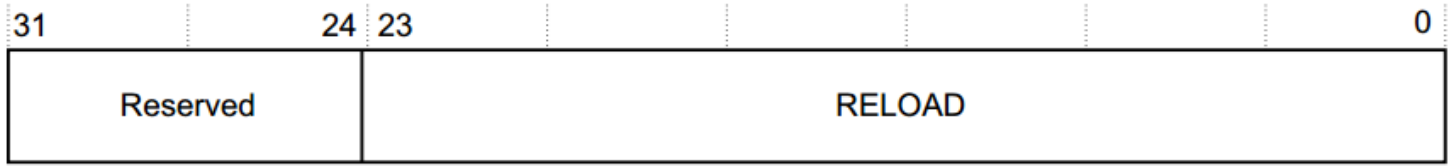


Table 4-34 SYST_RVR register bit assignments

Bits	Name	Function
[31:24]	-	Reserved.
[23:0]	RELOAD	Value to load into the SYST_CVR register when the counter is enabled and when it reaches 0, see *Calculating the RELOAD value*.

Calculating the RELOAD value

The RELOAD value can be any value in the range `0x00000001-0x00FFFFFF`. A start value of 0 is possible, but has no effect because the SysTick exception request and COUNTFLAG are activated when counting from 1 to 0.

The RELOAD value is calculated according to its use. For example, to generate a multi-shot timer with a period of N processor clock cycles, use a RELOAD value of N-1. If the SysTick interrupt is required every 100 clock pulses, set RELOAD to 99.

SysTick Current Value Register

The SYST_CVR register contains the current value of the SysTick counter.

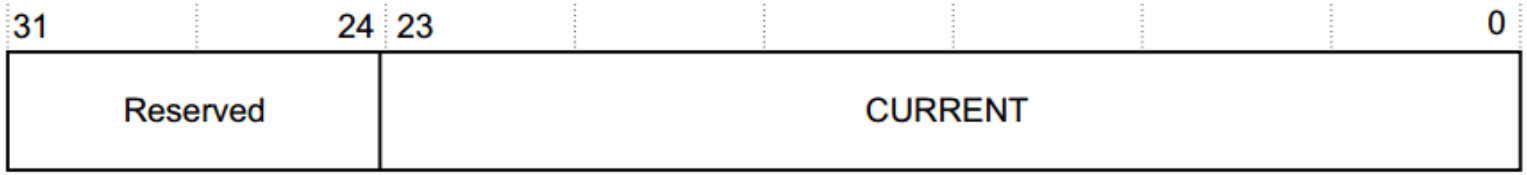


Table 4-35 SYST_CVR register bit assignments

Bits	Name	Function
[31:24]	-	Reserved.
[23:0]	CURRENT	Reads return the current value of the SysTick counter. A write of any value clears the field to 0, and also clears the SYST_CSR COUNTFLAG bit to 0.

System control block

The *System Control Block* (SCB) provides system implementation information, and system control. This includes configuration, control, and reporting of the system exceptions. The system control block registers are:

Table 4-12 Summary of the system control block registers

Address	Name	Type	Required privilege	Reset value	Description
0xE000E008	ACTLR	RW	Privileged	0x00000000	*Auxiliary Control Register*
0xE000ED00	CPUID	RO	Privileged	0x410FC240	*CPUID Base Register* on page 4-13
0xE000ED04	ICSR	RW[a]	Privileged	0x00000000	*Interrupt Control and State Register* on page 4-13
0xE000ED08	VTOR	RW	Privileged	0x00000000	*Vector Table Offset Register* on page 4-16
0xE000ED0C	AIRCR	RW[a]	Privileged	0xFA050000	*Application Interrupt and Reset Control Register* on page 4-16
0xE000ED10	SCR	RW	Privileged	0x00000000	*System Control Register* on page 4-19
0xE000ED14	CCR	RW	Privileged	0x00000200	*Configuration and Control Register* on page 4-19
0xE000ED18	SHPR1	RW	Privileged	0x00000000	*System Handler Priority Register 1* on page 4-21
0xE000ED1C	SHPR2	RW	Privileged	0x00000000	*System Handler Priority Register 2* on page 4-22
0xE000ED20	SHPR3	RW	Privileged	0x00000000	*System Handler Priority Register 3* on page 4-22
0xE000ED24	SHCRS	RW	Privileged	0x00000000	*System Handler Control and State Register* on page 4-23
0xE000ED28	CFSR	RW	Privileged	0x00000000	*Configurable Fault Status Register* on page 4-24
0xE000ED28	MMSR[b]	RW	Privileged	0x00	*MemManage Fault Status Register* on page 4-25
0xE000ED29	BFSR[b]	RW	Privileged	0x00	*BusFault Status Register* on page 4-26
0xE000ED2A	UFSR[b]	RW	Privileged	0x0000	*UsageFault Status Register* on page 4-28
0xE000ED2C	HFSR	RW	Privileged	0x00000000	*HardFault Status Register* on page 4-30
0xE000ED34	MMAR	RW	Privileged	Unknown	*MemManage Fault Address Register* on page 4-30
0xE000ED38	BFAR	RW	Privileged	Unknown	*BusFault Address Register* on page 4-31
0xE000ED3C	AFSR	RW	Privileged	0x00000000	*Auxiliary Fault Status Register* on page 4-31

Interrupt Control and State Register

The ICSR:

- provides:
  - a set-pending bit for the *Non-Maskable Interrupt* (NMI) exception
  - set-pending and clear-pending bits for the PendSV and SysTick exceptions
- indicates:
  - the exception number of the exception being processed
  - whether there are preempted active exceptions
  - the exception number of the highest priority pending exception
  - whether any interrupts are pending.

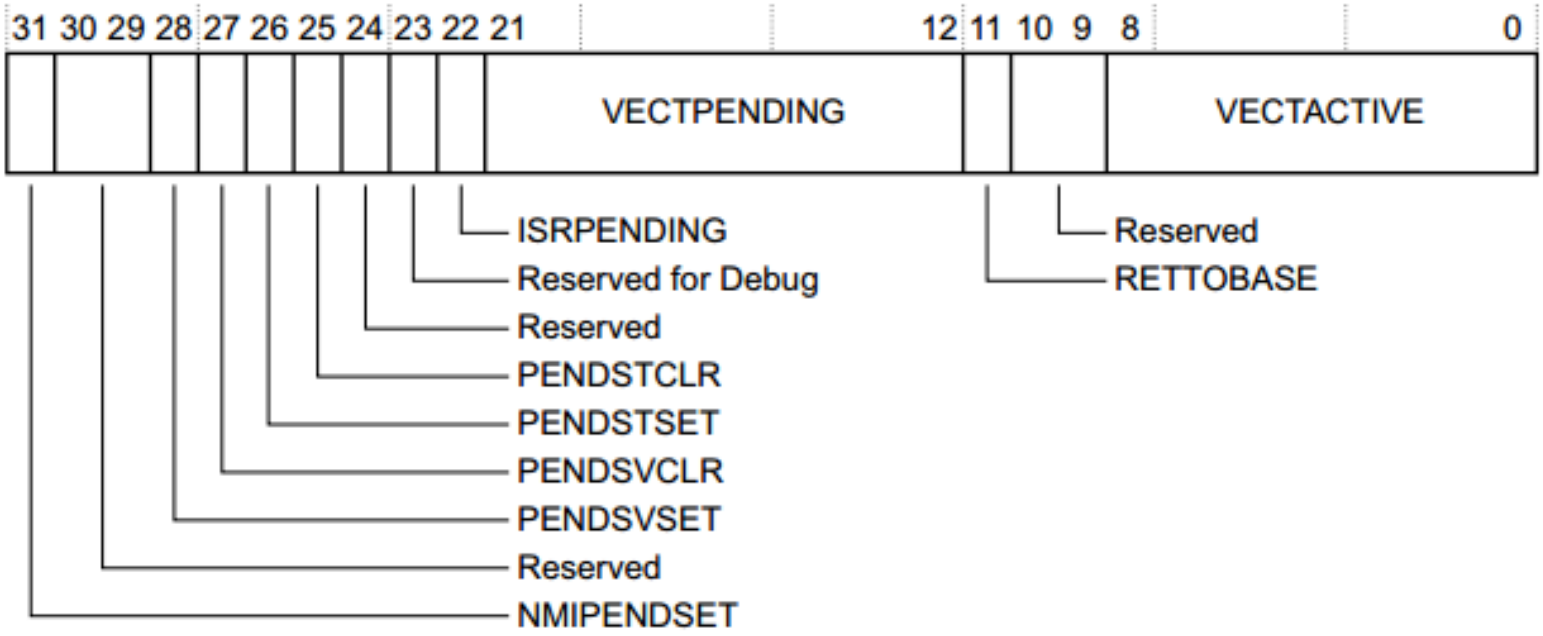


Table 4-15 ICSR bit assignments

Bits	Name	Type	Function
[31]	NMIPENDSET	RW	NMI set-pending bit. Write: 0 = no effect 1 = changes NMI exception state to pending. Read: 0 = NMI exception is not pending 1 = NMI exception is pending. Because NMI is the highest-priority exception, normally the processor enter the NMI exception handler as soon as it registers a write of 1 to this bit, and entering the handler clears this bit to 0. A read of this bit by the NMI exception handler returns 1 only if the NMI signal is reasserted while the processor is executing that handler.
[30:29]	-	-	Reserved.
[28]	PENDSVSET	RW	PendSV set-pending bit. Write: 0 = no effect 1 = changes PendSV exception state to pending. Read: 0 = PendSV exception is not pending 1 = PendSV exception is pending. Writing 1 to this bit is the only way to set the PendSV exception state to pending.
[27]	PENDSVCLR	WO	PendSV clear-pending bit. Write: 0 = no effect 1 = removes the pending state from the PendSV exception.
[26]	PENDSTSET	RW	SysTick exception set-pending bit. Write: 0 = no effect 1 = changes SysTick exception state to pending. Read: 0 = SysTick exception is not pending 1 = SysTick exception is pending.

Bits	Name	Type	Function
[25]	PENDSTCLR	WO	SysTick exception clear-pending bit. Write: 0 = no effect 1 = removes the pending state from the SysTick exception. This bit is WO. On a register read its value is Unknown.
[24]	-	-	Reserved.
[23]	Reserved for Debug use	RO	This bit is reserved for Debug use and reads-as-zero when the processor is not in Debug.
[22]	ISRPENDING	RO	Interrupt pending flag, excluding NMI and Faults: 0 = interrupt not pending 1 = interrupt pending.
[21:18]	-	-	Reserved.
[17:12]	VECTPENDING	RO	Indicates the exception number of the highest priority pending enabled exception: 0 = no pending exceptions Nonzero = the exception number of the highest priority pending enabled exception. The value indicated by this field includes the effect of the BASEPRI and FAULTMASK registers, but not any effect of the PRIMASK register.
[11]	RETTOBASE	RO	Indicates whether there are preempted active exceptions: 0 = there are preempted active exceptions to execute 1 = there are no active exceptions, or the currently-executing exception is the only active exception.
[10:9]	-	-	Reserved.
[8:0]	VECTACTIVE[a]	RO	Contains the active exception number: 0 = Thread mode Nonzero = The exception number[a] of the currently active exception. **Note** Subtract 16 from this value to obtain the CMSIS IRQ number required to index into the Interrupt Clear-Enable, Set-Enable, Clear-Pending, Set-Pending, or Priority Registers, see Table 2-5 on page 2-6.

a. This is the same value as IPSR bits[8:0], see *Interrupt Program Status Register* on page 2-6.

When you write to the ICSR, the effect is Unpredictable if you:

- write 1 to the PENDSVSET bit and write 1 to the PENDSVCLR bit
- write 1 to the PENDSTSET bit and write 1 to the PENDSTCLR bit.

System Handler Priority Registers

System Handler Priority Register 1

The bit assignments are:

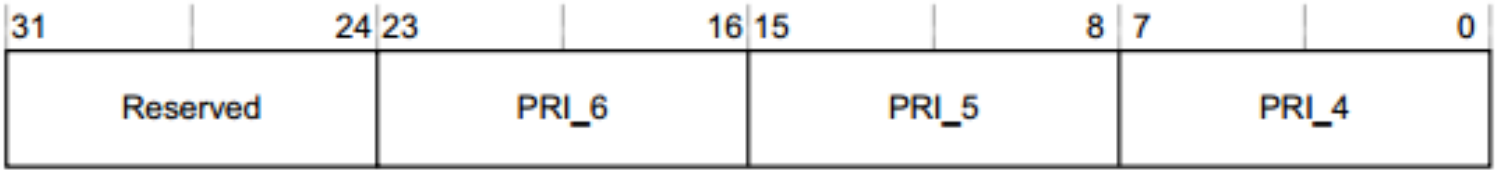


Table 4-21 SHPR1 register bit assignments

Bits	Name	Function
[31:24]	PRI_7	Reserved.
[23:16]	PRI_6	Priority of system handler 6, UsageFault
[15:8]	PRI_5	Priority of system handler 5, BusFault
[7:0]	PRI_4	Priority of system handler 4, MemManage

System Handler Priority Register 2

The bit assignments are:

Table 4-22 SHPR2 register bit assignments

Bits	Name	Function
[31:24]	PRI_11	Priority of system handler 11, SVCall
[23:0]	-	Reserved.

System Handler Priority Register 3

The bit assignments are:

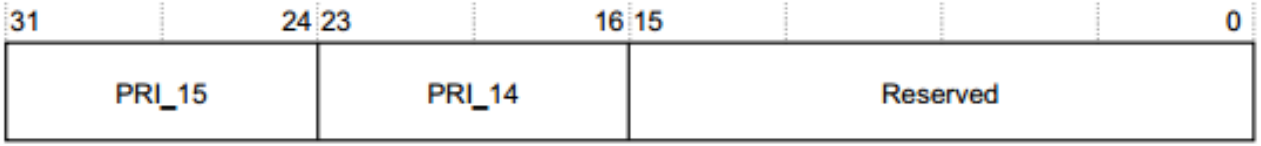


Table 4-23 SHPR3 register bit assignments

Bits	Name	Function
[31:24]	PRI_15	Priority of system handler 15, SysTick exception
[23:16]	PRI_14	Priority of system handler 14, PendSV
[15:0]	-	Reserved.

SysTick usage hints and tips

Some implementations stop all the processor clock signals during deep sleep mode. If this happens, the SysTick counter stops.

Ensure software uses aligned word accesses to access the SysTick registers.

The SysTick counter reload and current value are not initialized by hardware. This means the correct initialization sequence for the SysTick counter is:

1. Program reload value.
2. Clear current value.
3. Program Control and Status register.

4. (optional) Set SysTick timer priority .

```
.word   _estack
.word   Reset_Handler
.word   NMI_Handler
.word   HardFault_Handler
.word   MemManage_Handler
.word   BusFault_Handler
.word   UsageFault_Handler
.word   0
.word   0
.word   0
.word   0
.word   SVC_Handler
.word   DebugMon_Handler
.word   0
.word   PendSV_Handler
.word   SysTick Handler
```

Lab 8.1

- 設定clock source為HIS的SysTick timer，利用SysTick timer中斷機制，控制LED使它3秒暗一次(timer 3秒 interrupt一次)。

```
int main(){
        SystemClock_Config_source();
        SysTick_Config(…);
        init_GPIO();
while(1){}}
```

```
void SysTick_Handler(void){
//reset val
//LED control
}
```

```
void SystemClock_Config_source(void){
// turn on HSI16 oscillator
//check HSI16 ready
//SYSCLK divide by 16.
// Use HSI16 as system clock
}

void SysTick_Config(uint32_t ticks){
// set reload register
// Load the SysTick Counter Value
// CLKSOURCE processor clock
}
```

Lab 8.2

- 設定keypad的input腳為外部中斷的輸入源，當沒按任何按鍵時，LED(1顆)會保持在亮的狀態，當按下按鍵時會觸發程式進入handler，LED會在handler亮、暗(各0.5秒)輸入的次數，離開handler後LED會回到一開始的狀態。

Lab 8.3

- 利用SysTick timer、User button和蜂鳴器設計一個簡單的鬧鐘，先透過keypad輸入計時鬧鐘時間，每一個數字代表設定幾秒(2為2秒)，當輸入為0時則沒反應，繼續等待下次輸入，時間輸入完畢後，Systick timer會開始計時，而當時間到後，蜂鳴器便會響起(利用delay讓蜂鳴器發出聲音，頻率自訂)直到使用者按下User button後才會停止發出聲音並回到等待使用者輸入狀態，注意timer開始計時到使用者關閉蜂鳴器的期間，keypad不會有任何作用。

Reference

http://www.st.com/content/ccc/resource/technical/document/reference_manual/02/35/09/0c/4f/f7/40/03/DM00083560.pdf/files/DM00083560.pdf/jcr:content/translations/en.DM00083560.pdf

http://infocenter.arm.com/help/topic/com.arm.doc.dui0553a/DUI0553A_cortex_m4_dgug.pdf

http://www.eng.auburn.edu/~nelson/courses/elec5260_6260/slides/ARM%20STM32F407%20Interrupts.pdf