ARM Exception Handling

System Exceptions, Interrupts

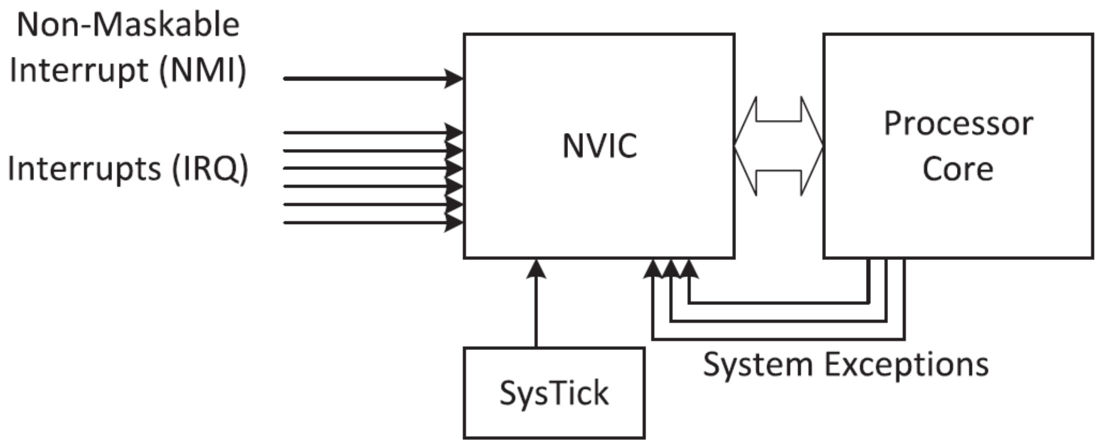
What are Exceptions?

- **Exceptions** are events that can cause changes to program flow.
- When an exception occurs, program suspends its current task and branches off to the **Exception Handler**.
- Upon execution of the Exception Handler, the ARM CPU then resumes normal program execution.

- Exceptions include …
  - Hardware interrupts:
    - Triggered from a peripheral (timer, UART, …)
    - Triggered externally (GPIO)
  - CPU resets, like power dip (NMI) condition
  - Software interrupts, like memory fault.

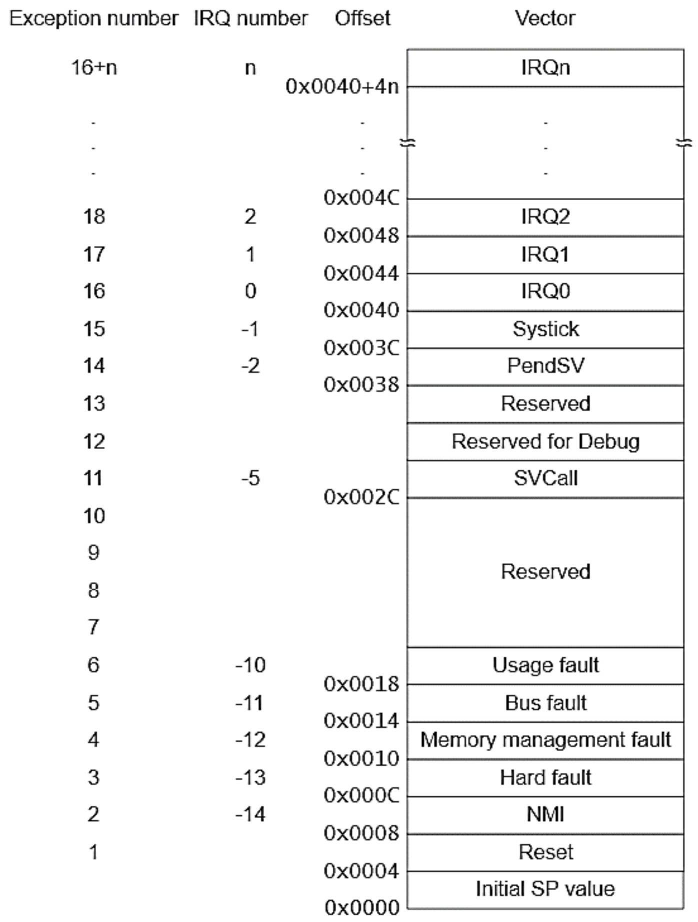
What are Exceptions?

- Interrupts on the ARM CPU are gated to and controlled by the **NVIC**.
- NVIC prioritizes and handles all exceptions in Handler mode.



NVIC Exception Handling

- Each exception source has an **exception number**:
  - 1–15 : system exceptions,
  - ≥16 : interrupts.
- Each exception has a 32-bit vector (address) entry in the **Vector Table** that points to the memory location of the associated Interrupt Service Routine (ISR).
- When an exception occurs,
  - Processor state is automatically stored in the stack, and automatically restored from the stack at the end of the ISR.
  - Interrupt vector is fetched in parallel from the Vector Table. *(Recall the vector table from previous lecture. See table at right).*



Exception number	IRQ number	Offset	Vector
16+n	n	0x0040+4n	IRQn
.		.	.
.		.	.
.		. 0x004C	.
18	2	0x0048	IRQ2
17	1	0x0044	IRQ1
16	0	0x0040	IRQ0
15	-1	0x003C	Systick
14	-2	0x0038	PendSV
13			Reserved
12			Reserved for Debug
11	-5	0x002C	SVCall
10			Reserved
9			
8			
7			
6	-10	0x0018	Usage fault
5	-11	0x0014	Bus fault
4	-12	0x0010	Memory management fault
3	-13	0x000C	Hard fault
2	-14	0x0008	NMI
1		0x0004	Reset
		0x0000	Initial SP value

Vector Table

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf), pg102

NVIC Exception Handling

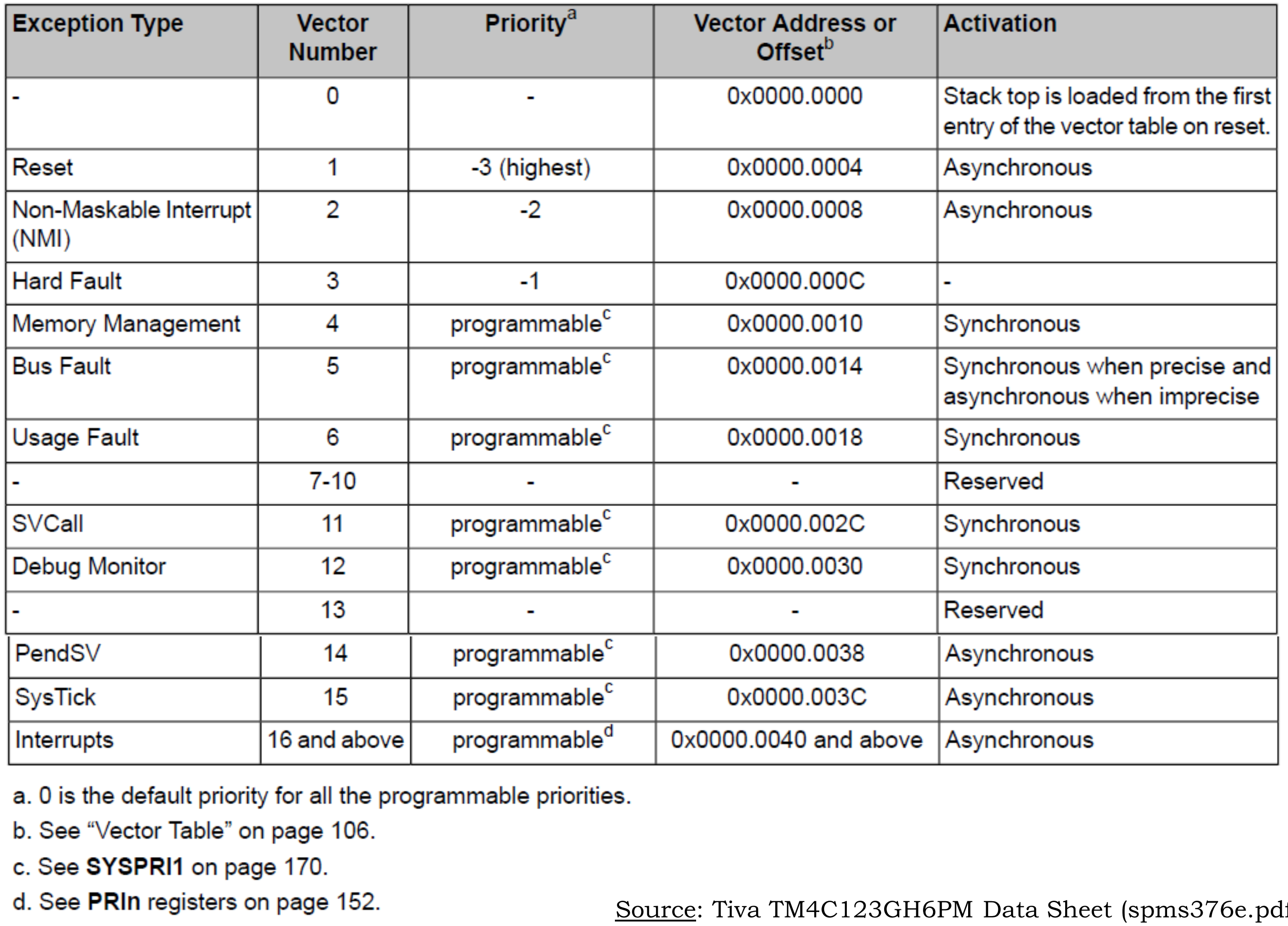
- The following are Exceptions defined in the ARM Tiva system. The ones in **Blue** has priorities that are user-programmable:
  - **System Exceptions** (total of 10):
    - Reset,
    - Non-Maskable Interrupt (NMI),
    - Hard Fault,
    - Memory Management,
    - Bus Fault,
    - Usage Fault,
    - SVCall,
    - Debug Monitor,
    - PendSV,
    - SysTick.
  - **Interrupts** (total of 78):
    - Interrupt numbers 0 to 138 (61 are reserved) : → (78 + 61 = 139).

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf), pg102

System Exceptions

TM4C123GH6PM7

System Exceptions: Summary Table



Exception Type	Vector Number	Priority[a]	Vector Address or Offset[b]	Activation
-	0	-	0x0000.0000	Stack top is loaded from the first entry of the vector table on reset.
Reset	1	-3 (highest)	0x0000.0004	Asynchronous
Non-Maskable Interrupt (NMI)	2	-2	0x0000.0008	Asynchronous
Hard Fault	3	-1	0x0000.000C	-
Memory Management	4	programmable[c]	0x0000.0010	Synchronous
Bus Fault	5	programmable[c]	0x0000.0014	Synchronous when precise and asynchronous when imprecise
Usage Fault	6	programmable[c]	0x0000.0018	Synchronous
-	7-10	-	-	Reserved
SVCall	11	programmable[c]	0x0000.002C	Synchronous
Debug Monitor	12	programmable[c]	0x0000.0030	Synchronous
-	13	-	-	Reserved
PendSV	14	programmable[c]	0x0000.0038	Asynchronous
SysTick	15	programmable[c]	0x0000.003C	Asynchronous
Interrupts	16 and above	programmable[d]	0x0000.0040 and above	Asynchronous

a. 0 is the default priority for all the programmable priorities.

b. See "Vector Table" on page 106.

c. See **SYSPRI1** on page 170.

d. See **PRIn** registers on page 152.

<u>Source</u>: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf), pg103

Types of System Exceptions

- **System Exceptions** include the following:
  - **Reset**:
    - due to power-up or warm reset (like Reset button).
    - Priority: -3 (highest) *(not user-programmable)*.
  - **NMI**: Non-Maskable Interrupt.
    - Can be triggered by HW or SW.
    - Through HW: through the NMI pin/signal.
    - Through SW: through the NMISET bit in Interrupt Control and State (INTCTRL) register.
    - Priority: -2 *(not user-programmable)*.
  - **Hard Fault**:
    - Due to error occurring during exception handling.
    - Priority: -1 *(not use-programmable)*.
  - **Memory Management**:
    - Due to memory access violation or mis-match.
    - Priority: user-programmable.

Exceptions in blue have user programmable priority.

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf), pg102 (Exceptions), pg160 (INTCTRL Reg)

Types of Exceptions

- **System Exceptions** include the following *(continued)*:
  - **Bus Fault**:
    - Memory-related fault due to instruction or data memory transaction.
    - Example: detection of memory pre-fetch or memory access faults on memory bus. faults.
    - Priority: user-programmable.
  - **Usage Fault**:
    - Occurs due to errors in instruction execution.
    - Example: invalid instructions, illegal unaligned memory access.
    - Priority: user-programmable.
  - **SVCall** (Supervisor Call):
    - Exception triggered by an SVC instruction.
    - In OS environment, SVC used to access kernel functions/ device drivers.
    - Priority: user-programmable.

Exceptions in blue have user programmable priority.

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf), pg102

Exceptions

- **System Exceptions** include the following *(continued)*:
  - **Debug Monitor**:
    - Caused by debug monitor function in ARM system.
    - Priority: user-programmable.
  - **PendSV**:
    - System-level service for context switching (used by OS).
    - Triggered through the INTCTRL register.
    - Priority: user-programmable.
  - **SysTick**:
    - Generated when SysTick timer when it reach 0 & when it is enabled to generate interrupt.
    - SW can also generate SysTick exception through the INTCTRL register.
    - Priority: user-programmable.

Note: Reset, NMI, Hard Fault priorities are non-user programmable. The rest are.

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf), pg102

Setting Exception & Interrupt Priority

Q: How do we set System Exception Priorities?

- **System Exception priorities** are set through the System Handler Priority registers – SYSPRI1, SYSPRI2 & SYSPRI3.
- Registers are part of the NVIC registers.
  - *Look into NVIC.h file (in Lab 1 template program on SysTick Timer) & look for the register definitions.*

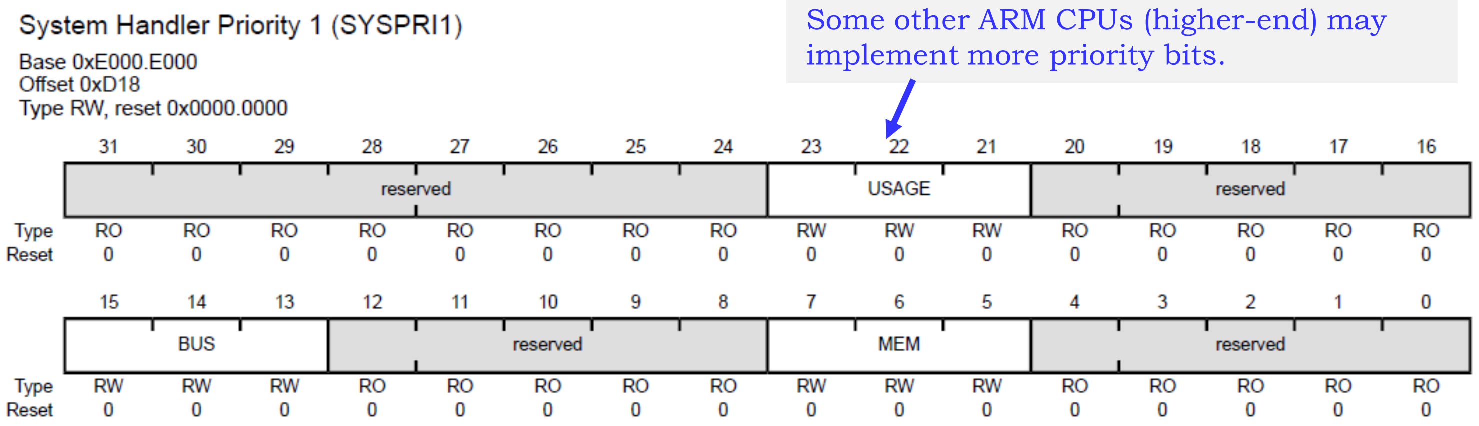
Also, note the following: *(covered later …)*

- **Interrupt priorities** are set through the Interrupt Priority registers – PRI0 to PRI34.

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf), pg170, 152

System Handler Priority 1 (SYSPRI1)

Note: Only the **upper 3-bits** of the 8-bit field is used.
Some other ARM CPUs (higher-end) may implement more priority bits.



System Handler Priority 1 (SYSPRI1)

Base 0xE000.E000
Offset 0xD18
Type RW, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved								USAGE			reserved				
Type	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

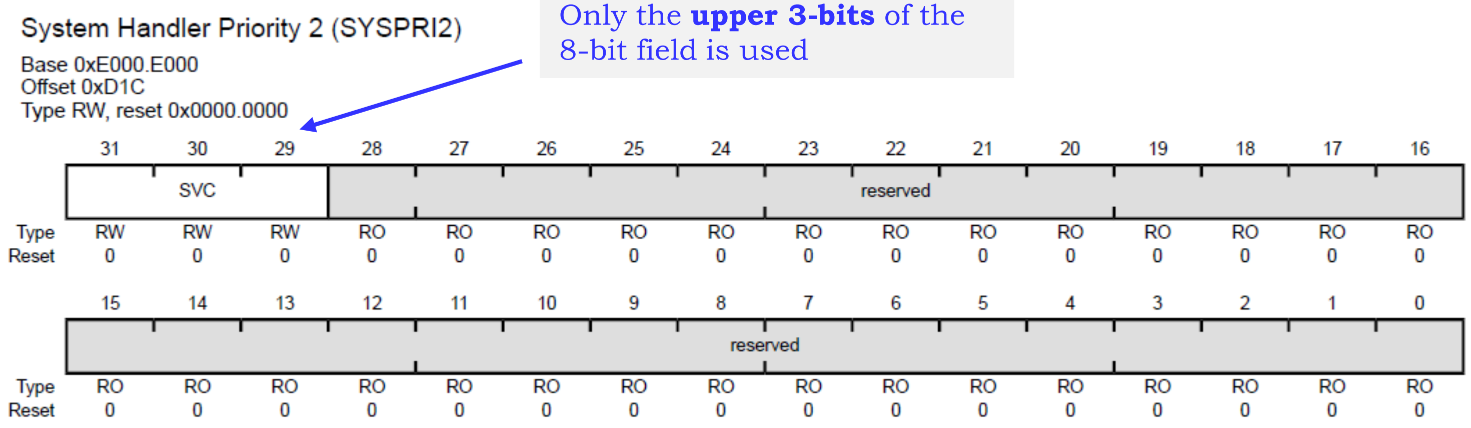
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	BUS			reserved					MEM			reserved				
Type	RW	RW	RW	RO	RO	RO	RO	RO	RW	RW	RW	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

USAGE	RW	0x0	Usage Fault Priority This field configures the priority level of the usage fault. Configurable priority values are in the range 0-7, with lower values having higher priority.
BUS	RW	0x0	Bus Fault Priority This field configures the priority level of the bus fault. Configurable priority values are in the range 0-7, with lower values having higher priority.
MEM	RW	0x0	Memory Management Fault Priority This field configures the priority level of the memory management fault. Configurable priority values are in the range 0-7, with lower values having higher priority.

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf), pg170

System Handler Priority 2 (SYSPRI2)

Only the **upper 3-bits** of the 8-bit field is used



System Handler Priority 2 (SYSPRI2)

Base 0xE000.E000
Offset 0xD1C
Type RW, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		SVC								reserved						
Type	RW	RW	RW	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

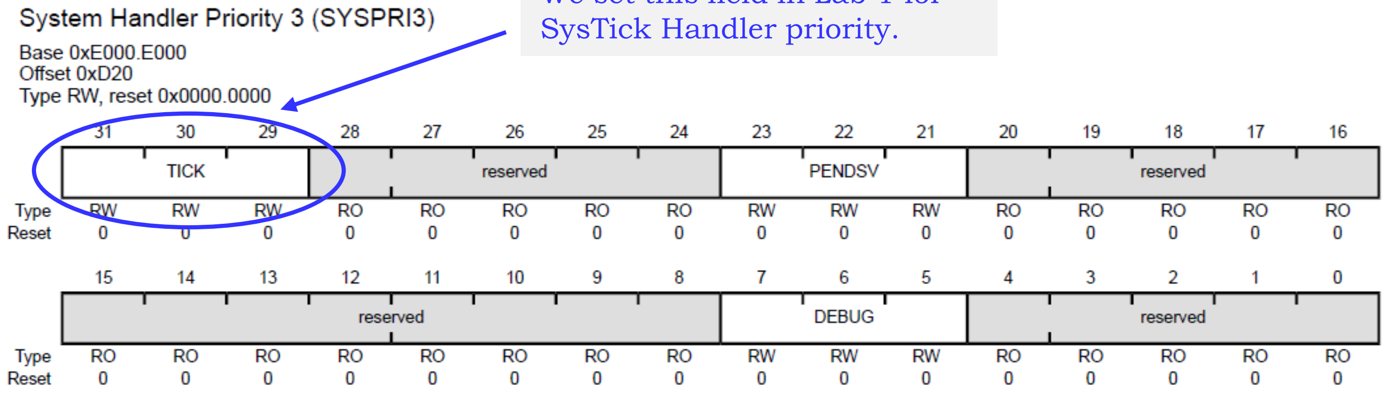
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								reserved								
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit/Field	Name	Type	Reset	Description
31:29	SVC	RW	0x0	SVCall Priority This field configures the priority level of SVCall. Configurable priority values are in the range 0-7, with lower values having higher priority.

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf), pg171

System Handler Priority 3 (SYSPRI3)

We set this field in Lab 1 for SysTick Handler priority.



System Handler Priority 3 (SYSPRI3)

Base 0xE000.E000
Offset 0xD20
Type RW, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	TICK			reserved					PENDSV			reserved				
Type	RW	RW	RW	RO	RO	RO	RO	RO	RW	RW	RW	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								DEBUG			reserved				
Type	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit/Field	Name	Type	Reset	Description
31:29	TICK	RW	0x0	SysTick Exception Priority This field configures the priority level of the SysTick exception. Configurable priority values are in the range 0-7, with lower values having higher priority.
23:21	PENDSV	RW	0x0	PendSV Priority This field configures the priority level of PendSV. Configurable priority values are in the range 0-7, with lower values having higher priority.
7:5	DEBUG	RW	0x0	Debug Priority This field configures the priority level of Debug. Configurable priority values are in the range 0-7, with lower values having higher priority.

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf), pg172

System Exception Priorities

Q: How many Priority Levels can be set for System Exceptions?

- Each priority field is made up of 3 bits.
- $2^3 = 8$ => 8 priority levels, from 0 to 7.
- 0 has the highest priority.
- 7 has lowest priority.

Interrupts

TM4C123GH6PM7

Interrupt Basics

- Interrupts allow **asynchronous** (*means, it can happen at any time during program execution*) transfer of program execution when triggered by a hardware event.
- Trigger or Events can be <u>internal</u> or <u>external</u>.
  - Internal events include bus faults, memory faults, SysTick timer, …
  - External events include UART data receive, switch status (GPIO), …

Polling versus Interrupts Analogy

- **Polling**: You pick up the phone every few minutes to check if you have a call coming through.

- **Interrupt**: You do whatever you need to do (get on with life), gets notified when a call comes through.
  When there is a call, you take the call and then goes back to do whatever you want to do.

Polling versus Interrupts

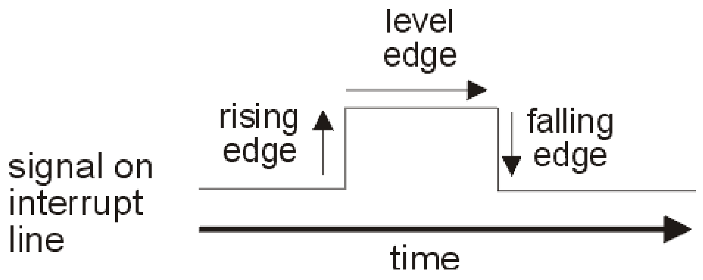
```
/** Polling method **/

void main()
{
  while (1)
  {
    read_SW1(); // read status
    if (SW1_pressed)
    {
      LED_on();
    }

    /* other program codes */

  {
}
```

```
/** Interrupt method **/

void main() {
  while (1)
  {

     /* program codes */

  }
}


/* Interrupt Handler for SW1 */
GPIO_handler
{
  LED_on();
  exit;
}
```

Interrupt Types

- HW Interrupts can be **level-triggered** or **edge-triggered** (*also called pulse triggered*).
- All interrupts are **latched**.

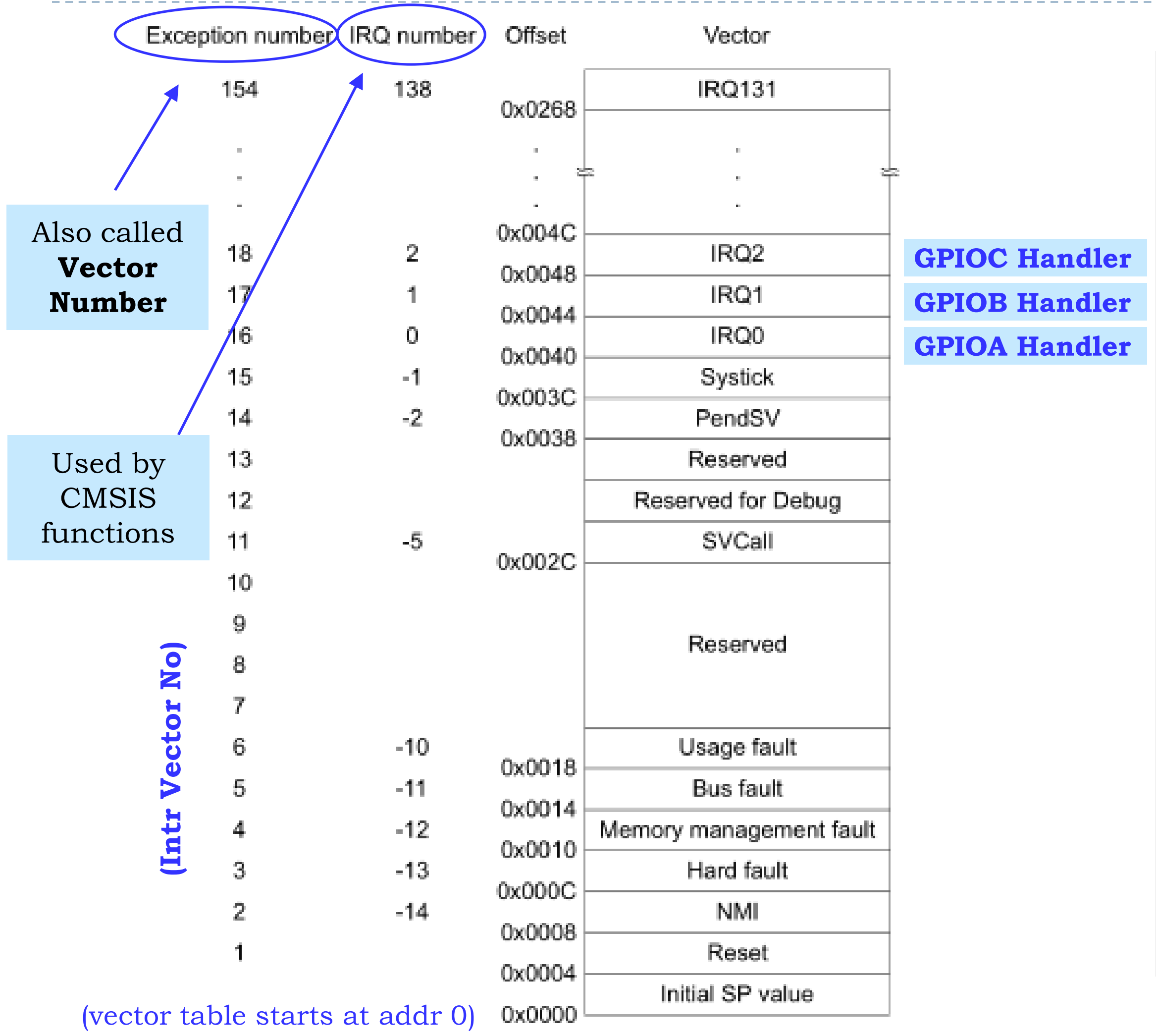
- **Level Interrupts:**
  - As long as the interrupt signal is held active, interrupts are generated.
  - If interrupt signal is not removed, the interrupt becomes pending again (repeated interrupts!).
  - A peripheral can thus assert the interrupt signal until it no longer needs servicing.
- **Edge Interrupts:**
  - Can be rising or falling edge triggered interrupts.
  - Interrupt is sampled synchronously on the rising edge of the processor clock.
  - Peripheral requesting interrupt must assert interrupt for at least one clock cycle.



Pending & Active Interrupts

- Once an Interrupt is successfully detected, the interrupt status is **Pending** & waiting to be serviced by the CPU.
- A Pending Interrupt changes to **Active** status when the **ISR** is being serviced.
- ISR then writes to the corresponding **Interrupt Clear** bit to clear the Interrupt.
- Once cleared, the next interrupt can be triggered.

Interrupt Vector Table



Exception number	IRQ number	Offset	Vector
154	138	0x0268	IRQ131
. . .		. . . 0x004C	. . .
18	2	0x0048	IRQ2
17	1	0x0044	IRQ1
16	0	0x0040	IRQ0
15	-1	0x003C	Systick
14	-2	0x0038	PendSV
13			Reserved
12			Reserved for Debug
11	-5	0x002C	SVCall
10			Reserved
9			
8			
7			
6	-10	0x0018	Usage fault
5	-11	0x0014	Bus fault
4	-12	0x0010	Memory management fault
3	-13	0x000C	Hard fault
2	-14	0x0008	NMI
1		0x0004	Reset
		0x0000	Initial SP value

An **Interrupt Vector Table** contains the starting addresses (Vectors) of the Interrupt Service Routines (ISR) or Handlers.

Each vector is assigned an **IRQ number**.
Example,
GPIOB has IRQ number 1.
GPIOC has IRQ number 2.

Vector example:
Contents at address 0x0040 points to the start address of GPIOA ISR (GPIOA_Handler())

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf, pg107)

Interrupt Vector & Addresses

Translation between <u>Interrupt Number</u> & <u>Vector Table address</u>:

Address (in vector table) which holds the address of ISR for interrupt number n is given by,

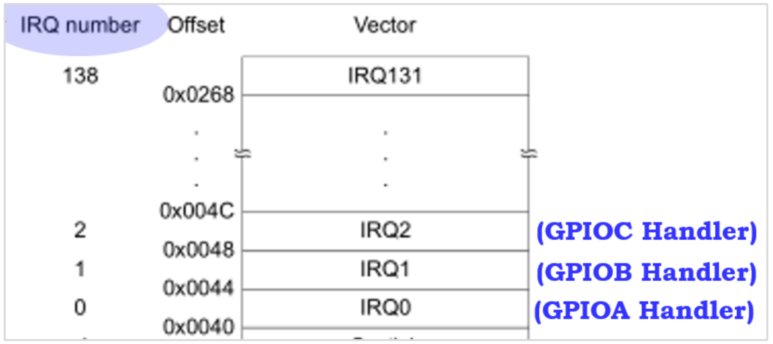
$$Address\ in\ vector\ table = 64 + (n \times 4)$$

<u>Example 1</u>:

- GPIOA has interrupt number = 0
- Address of pointer to GPIOA ISR = 64 = 0x40.

<u>Example 2</u>:

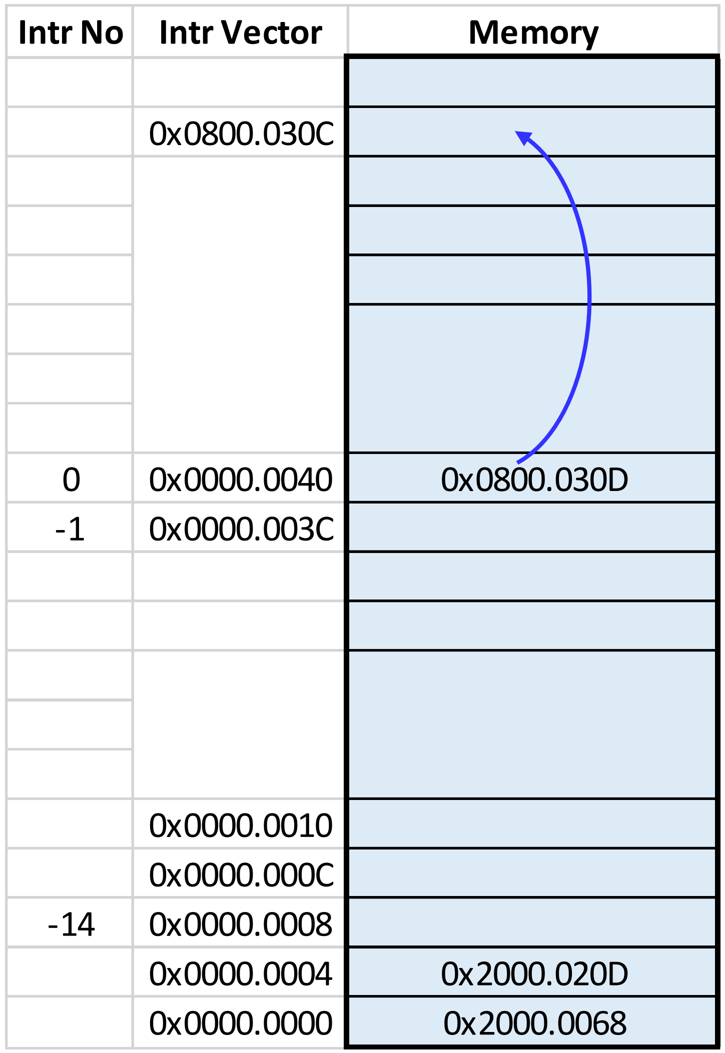
- GPIOC has interrupt number = 2
- Address of pointer to GPIOA ISR = 64 + (2 × 4) = 0x48.



<u>Example 3</u>:

- GPIOF has interrupt number = 30
- Address of pointer to GPIOF ISR = 64 + (30 x 4) = 0xB8.

Ex: Interrupt Vector & Addresses



Intr No	Intr Vector	Memory
	0x0800.030C	
0	0x0000.0040	0x0800.030D
-1	0x0000.003C	
	0x0000.0010	
	0x0000.000C	
-14	0x0000.0008	
	0x0000.0004	0x2000.020D
	0x0000.0000	0x2000.0068

(GPIOA Handler)

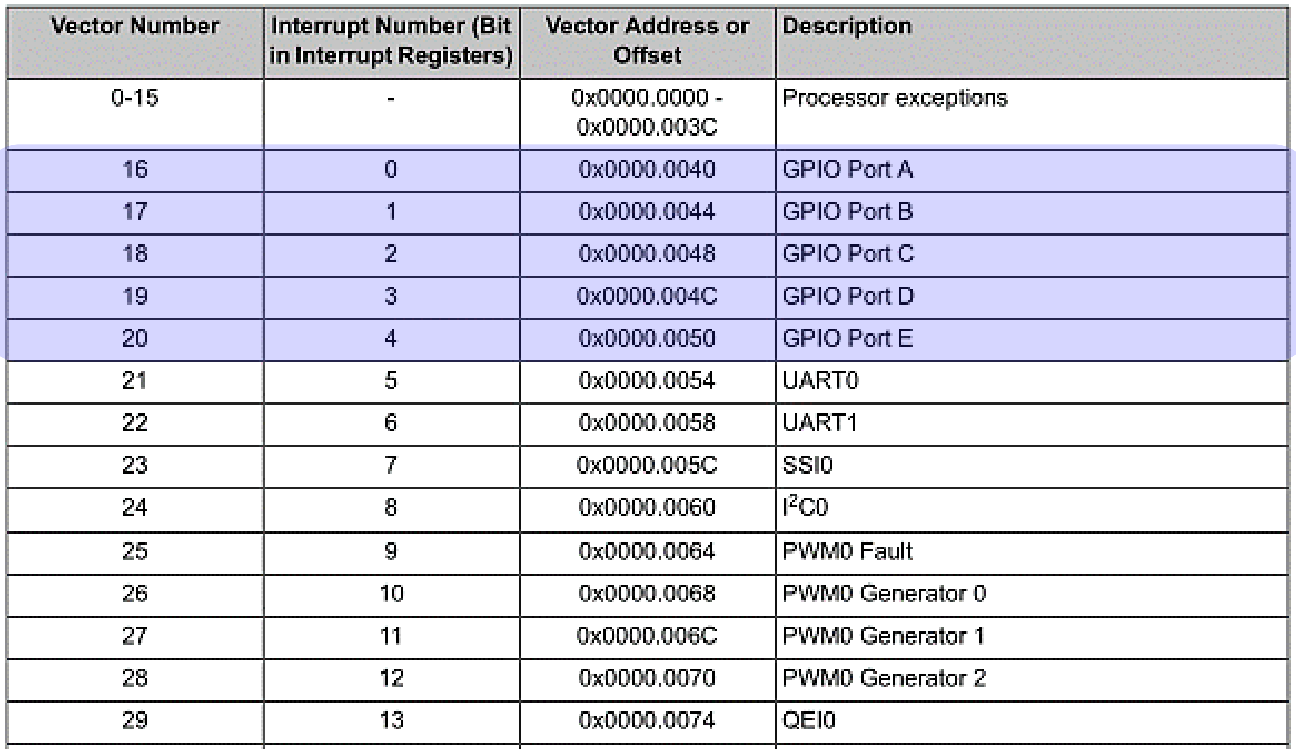
```
void GPIOA_Handler()
{  . . .  }
```

GPIOA Intr Number = 0
Vector Address = 0x40

(Address of GPIOA Handler)

Pointer to NMI_Handler()
Pointer to Reset_Handler()
MSP

Interrupt Vectors [1/5]

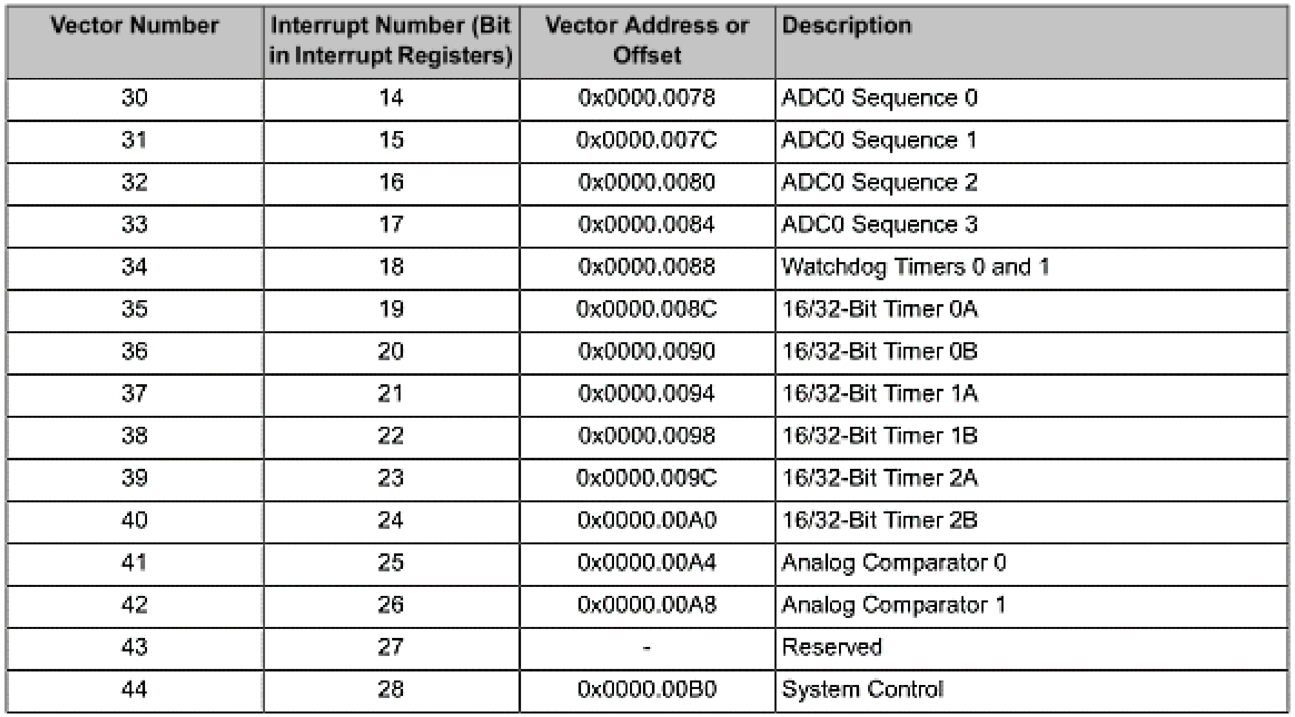


Vector Number	Interrupt Number (Bit in Interrupt Registers)	Vector Address or Offset	Description
0-15	-	0x0000.0000 - 0x0000.003C	Processor exceptions
16	0	0x0000.0040	GPIO Port A
17	1	0x0000.0044	GPIO Port B
18	2	0x0000.0048	GPIO Port C
19	3	0x0000.004C	GPIO Port D
20	4	0x0000.0050	GPIO Port E
21	5	0x0000.0054	UART0
22	6	0x0000.0058	UART1
23	7	0x0000.005C	SSI0
24	8	0x0000.0060	I^2C0
25	9	0x0000.0064	PWM0 Fault
26	10	0x0000.0068	PWM0 Generator 0
27	11	0x0000.006C	PWM0 Generator 1
28	12	0x0000.0070	PWM0 Generator 2
29	13	0x0000.0074	QEI0

• Interrupt Vectors Number	: 0 to 154
• Interrupt Numbers	: 0 to 138 (***used by NVIC registers***)
• IRQs	: IRQ0 to IRQ131 (7 reserved)

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf, pg104)

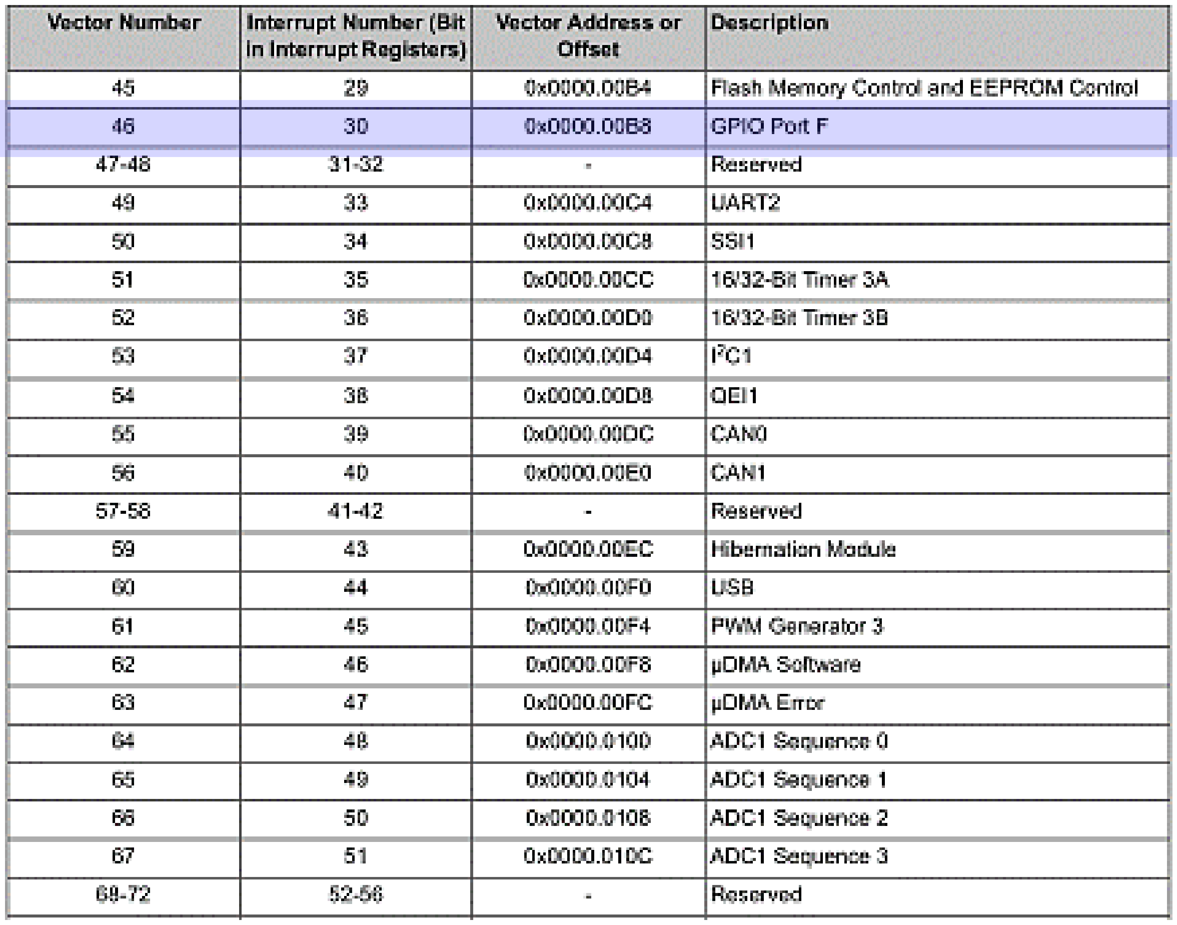
Interrupt Vectors [2/5]



Vector Number	Interrupt Number (Bit in Interrupt Registers)	Vector Address or Offset	Description
30	14	0x0000.0078	ADC0 Sequence 0
31	15	0x0000.007C	ADC0 Sequence 1
32	16	0x0000.0080	ADC0 Sequence 2
33	17	0x0000.0084	ADC0 Sequence 3
34	18	0x0000.0088	Watchdog Timers 0 and 1
35	19	0x0000.008C	16/32-Bit Timer 0A
36	20	0x0000.0090	16/32-Bit Timer 0B
37	21	0x0000.0094	16/32-Bit Timer 1A
38	22	0x0000.0098	16/32-Bit Timer 1B
39	23	0x0000.009C	16/32-Bit Timer 2A
40	24	0x0000.00A0	16/32-Bit Timer 2B
41	25	0x0000.00A4	Analog Comparator 0
42	26	0x0000.00A8	Analog Comparator 1
43	27	-	Reserved
44	28	0x0000.00B0	System Control

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf, pg104)

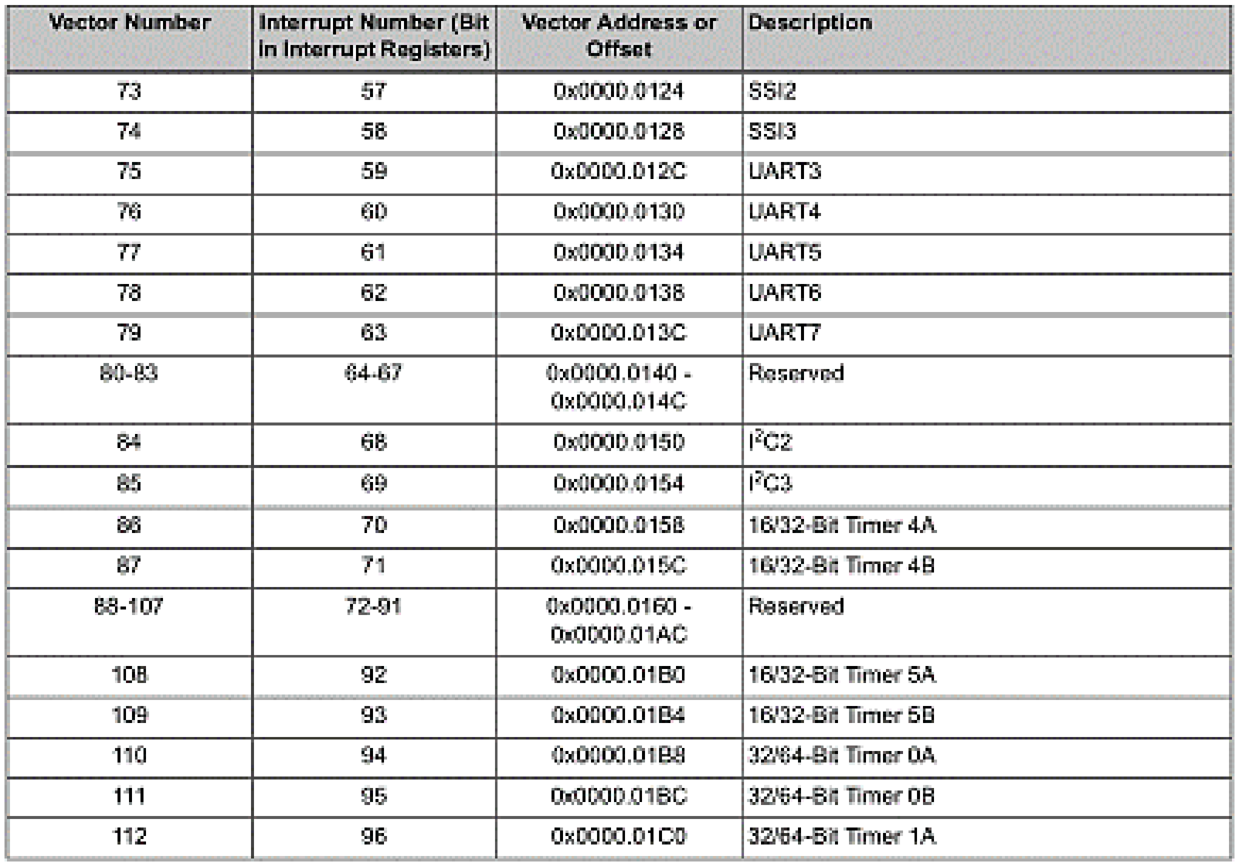
Interrupt Vectors [3/5]



Vector Number	Interrupt Number (Bit in Interrupt Registers)	Vector Address or Offset	Description
45	29	0x0000.00B4	Flash Memory Control and EEPROM Control
46	30	0x0000.00B8	GPIO Port F
47-48	31-32	-	Reserved
49	33	0x0000.00C4	UART2
50	34	0x0000.00C8	SSI1
51	35	0x0000.00CC	16/32-Bit Timer 3A
52	36	0x0000.00D0	16/32-Bit Timer 3B
53	37	0x0000.00D4	I^{2}C1
54	38	0x0000.00D8	QEI1
55	39	0x0000.00DC	CAN0
56	40	0x0000.00E0	CAN1
57-58	41-42	-	Reserved
59	43	0x0000.00EC	Hibernation Module
60	44	0x0000.00F0	USB
61	45	0x0000.00F4	PWM Generator 3
62	46	0x0000.00F8	μDMA Software
63	47	0x0000.00FC	μDMA Error
64	48	0x0000.0100	ADC1 Sequence 0
65	49	0x0000.0104	ADC1 Sequence 1
66	50	0x0000.0108	ADC1 Sequence 2
67	51	0x0000.010C	ADC1 Sequence 3
68-72	52-56	-	Reserved

Source: Tiva TM4C123GH6PM
Data Sheet (spms376e.pdf, pg104)

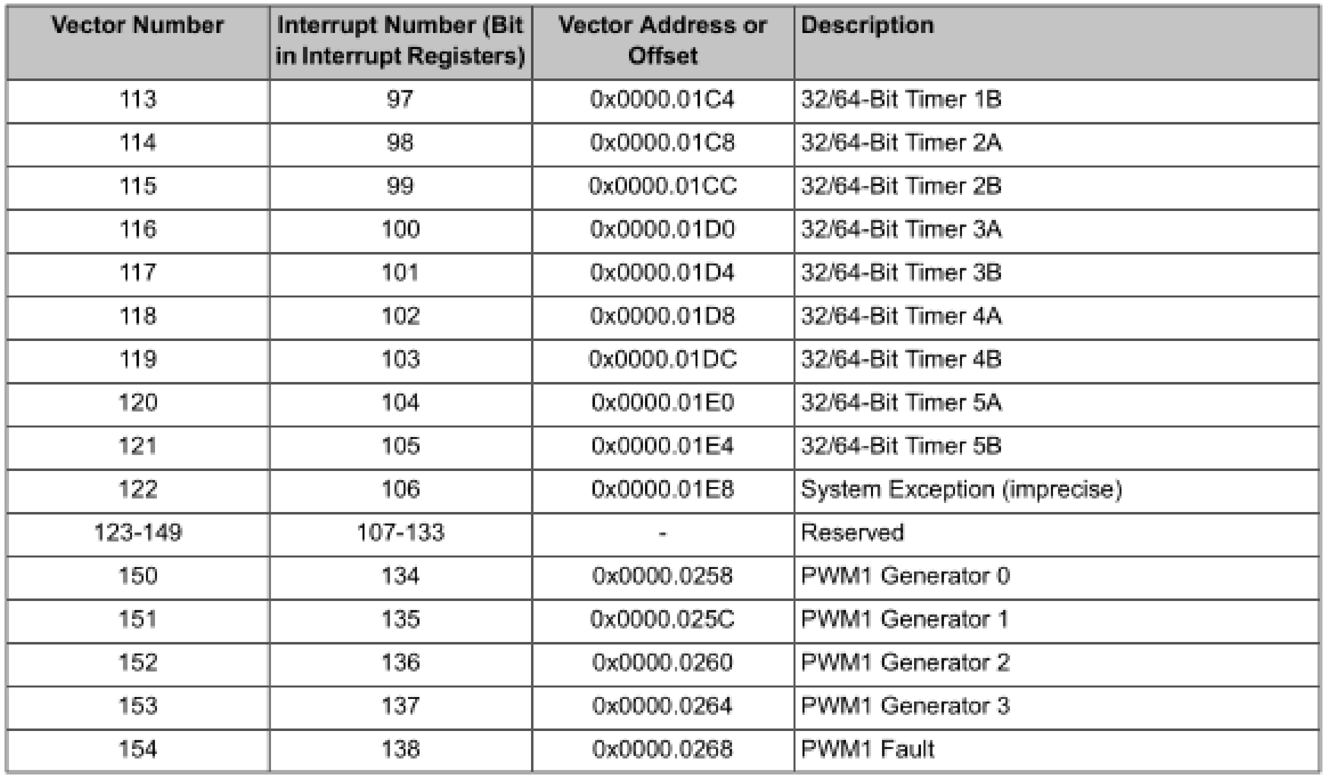
Interrupt Vectors [4/5]



Vector Number	Interrupt Number (Bit in Interrupt Registers)	Vector Address or Offset	Description
73	57	0x0000.0124	SSI2
74	58	0x0000.0128	SSI3
75	59	0x0000.012C	UART3
76	60	0x0000.0130	UART4
77	61	0x0000.0134	UART5
78	62	0x0000.0138	UART6
79	63	0x0000.013C	UART7
80-83	64-67	0x0000.0140 - 0x0000.014C	Reserved
84	68	0x0000.0150	I^{2}C2
85	69	0x0000.0154	I^{2}C3
86	70	0x0000.0158	16/32-Bit Timer 4A
87	71	0x0000.015C	16/32-Bit Timer 4B
88-107	72-91	0x0000.0160 - 0x0000.01AC	Reserved
108	92	0x0000.01B0	16/32-Bit Timer 5A
109	93	0x0000.01B4	16/32-Bit Timer 5B
110	94	0x0000.01B8	32/64-Bit Timer 0A
111	95	0x0000.01BC	32/64-Bit Timer 0B
112	96	0x0000.01C0	32/64-Bit Timer 1A

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf, pg104)

Interrupt Vectors [5/5]



Vector Number	Interrupt Number (Bit in Interrupt Registers)	Vector Address or Offset	Description
113	97	0x0000.01C4	32/64-Bit Timer 1B
114	98	0x0000.01C8	32/64-Bit Timer 2A
115	99	0x0000.01CC	32/64-Bit Timer 2B
116	100	0x0000.01D0	32/64-Bit Timer 3A
117	101	0x0000.01D4	32/64-Bit Timer 3B
118	102	0x0000.01D8	32/64-Bit Timer 4A
119	103	0x0000.01DC	32/64-Bit Timer 4B
120	104	0x0000.01E0	32/64-Bit Timer 5A
121	105	0x0000.01E4	32/64-Bit Timer 5B
122	106	0x0000.01E8	System Exception (imprecise)
123-149	107-133	-	Reserved
150	134	0x0000.0258	PWM1 Generator 0
151	135	0x0000.025C	PWM1 Generator 1
152	136	0x0000.0260	PWM1 Generator 2
153	137	0x0000.0264	PWM1 Generator 3
154	138	0x0000.0268	PWM1 Fault

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf, pg104)

Interrupt Vectors (startup_TM4C123.s) [1/2]

```
DCD     __initial_sp          ; Top of Stack
DCD     Reset_Handler         ; Reset Handler
DCD     NMI_Handler           ; NMI Handler
DCD     HardFault_Handler     ; Hard Fault Handler
DCD     MemManage_Handler     ; MPU Fault Handler
DCD     BusFault_Handler      ; Bus Fault Handler
DCD     UsageFault_Handler    ; Usage Fault Handler
DCD     0                     ; Reserved
DCD     0                     ; Reserved
DCD     0                     ; Reserved
DCD     0                     ; Reserved
DCD     SVC_Handler           ; SVCall Handler
DCD     DebugMon_Handler      ; Debug Monitor Handler
DCD     0                     ; Reserved
DCD     PendSV_Handler        ; PendSV Handler
DCD     SysTick_Handler       ; SysTick Handler

; External Interrupts

DCD     GPIOA_Handler         ;   0: GPIO Port A
DCD     GPIOB_Handler         ;   1: GPIO Port B
DCD     GPIOC_Handler         ;   2: GPIO Port C
DCD     GPIOD_Handler         ;   3: GPIO Port D
DCD     GPIOE_Handler         ;   4: GPIO Port E
DCD     UART0_Handler         ;   5: UART0 Rx and Tx
DCD     UART1_Handler         ;   6: UART1 Rx and Tx
DCD     SSI0_Handler          ;   7: SSI0 Rx and Tx
DCD     I2C0_Handler          ;   8: I2C0 Master & Slave
DCD     PMW0_FAULT_Handler    ;   9: PWM Fault
DCD     PWM0_0_Handler        ;  10: PWM Generator 0
```

- Example of interrupt vectors defined in *startup.s* file.
- Address points to the function that handles the exception.
- Contents at 0x0000.0000 points to top of the main stack.
- Contents at 0x0000.0004 points to address of *Reset_Handler()* which is executed upon CPU reset.
  - *Reset_Handler()* initializes global variables and then calls the *main()* program.

- From table, we can see that the start address for ISR handlers are at:
  - 0x0000.0040 for GPIOA.
  - 0x0000.0044 for GPIOB.
  - 0x0000.0048 for GPIOC.
  - 0x0000.004C for GPIOD.
  - 0x0000.0050 for GPIOE.

Interrupt Vectors **(startup_TM4C123.s)** [2/2]

```
DCD     TIMER0A_Handler   ;  19: Timer 0 subtimer A
DCD     TIMER0B_Handler   ;  20: Timer 0 subtimer B
DCD     TIMER1A_Handler   ;  21: Timer 1 subtimer A
DCD     TIMER1B_Handler   ;  22: Timer 1 subtimer B
DCD     TIMER2A_Handler   ;  23: Timer 2 subtimer A
DCD     TIMER2B_Handler   ;  24: Timer 2 subtimer B
DCD     COMP0_Handler     ;  25: Analog Comparator 0
DCD     COMP1_Handler     ;  26: Analog Comparator 1
DCD     COMP2_Handler     ;  27: Analog Comparator 2
DCD     SYSCTL_Handler    ;  28: System Control (PLL, OSC, BO)
DCD     FLASH_Handler     ;  29: FLASH Control
DCD     GPIOF_Handler     ;  30: GPIO Port F
DCD     GPIOG_Handler     ;  31: GPIO Port G
DCD     GPIOH_Handler     ;  32: GPIO Port H
DCD     UART2_Handler     ;  33: UART2 Rx and Tx
 CD     SSI1_Handler      ;  34: SSI1 Rx and Tx
```

← Points to start address of GPIOF_Handler( )

- **`GPIOF_Handler()`** is the interrupt handler that handles interrupts from SW1 (PF04) & SW2 (PF0) on the Tiva LaunchPad.

- RGB LEDs are also connected to Port F (PF1, PF2, PF3).

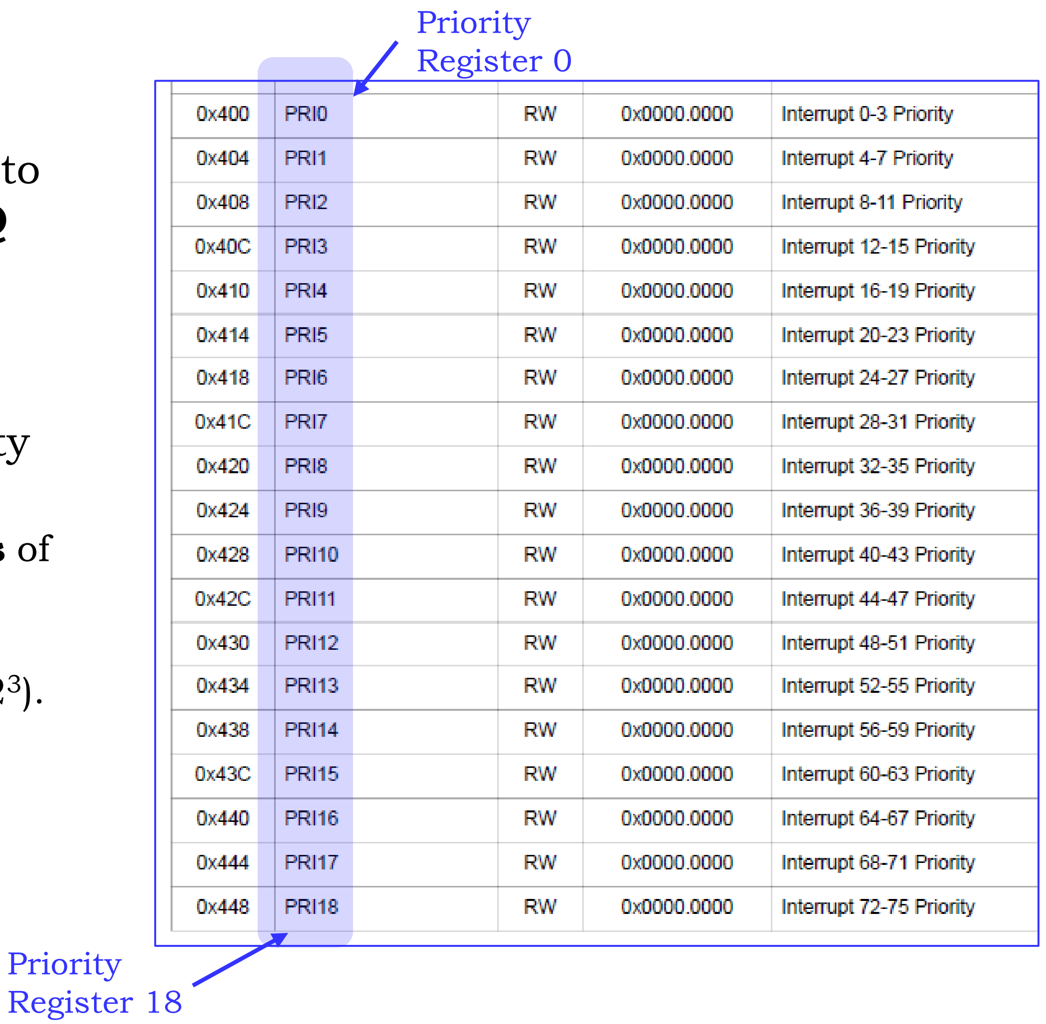
Interrupt Priority

TM4C123GH6PM7

NVIC Interrupt Priority Registers

- Total of **35** Interrupt Priority Registers (PRI0 to PRI34) for **IRQ 0** to **IRQ 138**.

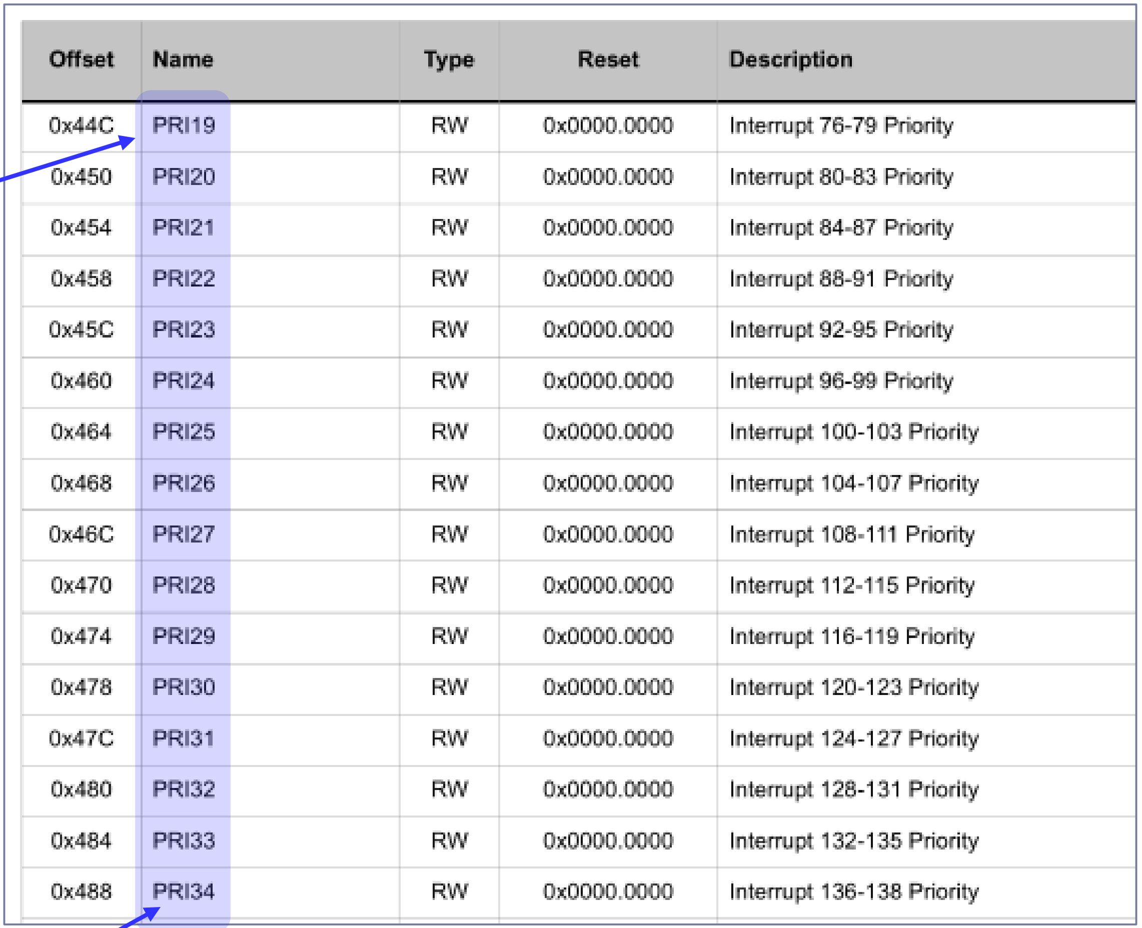
- Each **Priority Register** contains an 8-bit priority field for **4** devices.
    - Only the **upper 3-bits** of the 8-bit field is used. Rest of bits = 0.
    - 8 possible priorities (2^3).
    - 0 is highest priority.
    - 7 is lowest priority.


Priority Register 0
Priority Register 18

0x400	PRI0	RW	0x0000.0000	Interrupt 0-3 Priority
0x404	PRI1	RW	0x0000.0000	Interrupt 4-7 Priority
0x408	PRI2	RW	0x0000.0000	Interrupt 8-11 Priority
0x40C	PRI3	RW	0x0000.0000	Interrupt 12-15 Priority
0x410	PRI4	RW	0x0000.0000	Interrupt 16-19 Priority
0x414	PRI5	RW	0x0000.0000	Interrupt 20-23 Priority
0x418	PRI6	RW	0x0000.0000	Interrupt 24-27 Priority
0x41C	PRI7	RW	0x0000.0000	Interrupt 28-31 Priority
0x420	PRI8	RW	0x0000.0000	Interrupt 32-35 Priority
0x424	PRI9	RW	0x0000.0000	Interrupt 36-39 Priority
0x428	PRI10	RW	0x0000.0000	Interrupt 40-43 Priority
0x42C	PRI11	RW	0x0000.0000	Interrupt 44-47 Priority
0x430	PRI12	RW	0x0000.0000	Interrupt 48-51 Priority
0x434	PRI13	RW	0x0000.0000	Interrupt 52-55 Priority
0x438	PRI14	RW	0x0000.0000	Interrupt 56-59 Priority
0x43C	PRI15	RW	0x0000.0000	Interrupt 60-63 Priority
0x440	PRI16	RW	0x0000.0000	Interrupt 64-67 Priority
0x444	PRI17	RW	0x0000.0000	Interrupt 68-71 Priority
0x448	PRI18	RW	0x0000.0000	Interrupt 72-75 Priority

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf), pg135, 154

NVIC Interrupt Priority Registers



Offset	Name	Type	Reset	Description
0x44C	PRI19	RW	0x0000.0000	Interrupt 76-79 Priority
0x450	PRI20	RW	0x0000.0000	Interrupt 80-83 Priority
0x454	PRI21	RW	0x0000.0000	Interrupt 84-87 Priority
0x458	PRI22	RW	0x0000.0000	Interrupt 88-91 Priority
0x45C	PRI23	RW	0x0000.0000	Interrupt 92-95 Priority
0x460	PRI24	RW	0x0000.0000	Interrupt 96-99 Priority
0x464	PRI25	RW	0x0000.0000	Interrupt 100-103 Priority
0x468	PRI26	RW	0x0000.0000	Interrupt 104-107 Priority
0x46C	PRI27	RW	0x0000.0000	Interrupt 108-111 Priority
0x470	PRI28	RW	0x0000.0000	Interrupt 112-115 Priority
0x474	PRI29	RW	0x0000.0000	Interrupt 116-119 Priority
0x478	PRI30	RW	0x0000.0000	Interrupt 120-123 Priority
0x47C	PRI31	RW	0x0000.0000	Interrupt 124-127 Priority
0x480	PRI32	RW	0x0000.0000	Interrupt 128-131 Priority
0x484	PRI33	RW	0x0000.0000	Interrupt 132-135 Priority
0x488	PRI34	RW	0x0000.0000	Interrupt 136-138 Priority

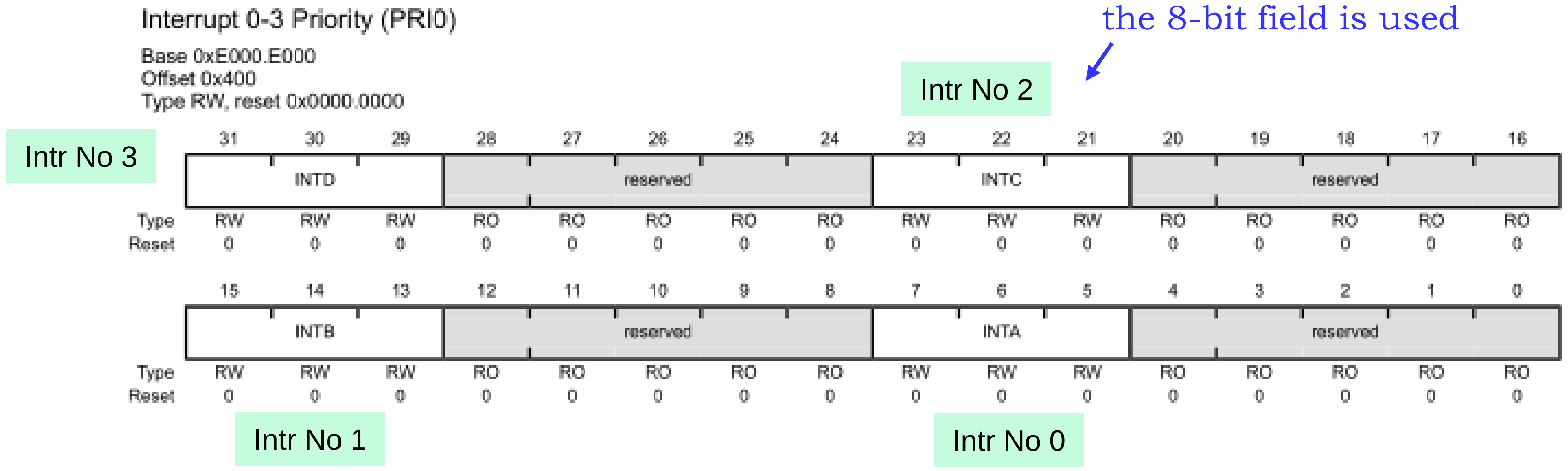
Priority Register 19

Priority Register 34

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf), pg136

Interrupt Priority Register (PRI0)

Only the **upper 3-bits** of the 8-bit field is used



Interrupt 0-3 Priority (PRI0)

Base 0xE000.E000
Offset 0x400
Type RW, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		INTD				reserved				INTC				reserved		
Type	RW	RW	RW	RO	RO	RO	RO	RO	RW	RW	RW	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		INTB				reserved				INTA				reserved		
Type	RW	RW	RW	RO	RO	RO	RO	RO	RW	RW	RW	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Intr No 3
Intr No 2
Intr No 1
Intr No 0

- Each register holds the priority bits for **4 interrupt sources**.
- Interrupt priority is defined through a 3-bit field – priority 0 to 7. *(0 is highest priority)*

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf), pg153

How to Set NVIC Interrupt Priority ….

```
void NVIC_SetPriority (IRQn_Type IRQn,uint32_t priority )
```

Parameters:

IRQn	: Interrupt Number
priority	: Priority to set

Description:

Sets the priority for the interrupt specified by IRQn.
IRQn can specify any device specific interrupt, or processor exception.
The priority specifies the interrupt priority value, whereby lower values indicate a higher priority.
The default priority is 0 for every interrupt which is the highest possible priority.

CMSIS function defined in 'core_cm.h'.

Example:

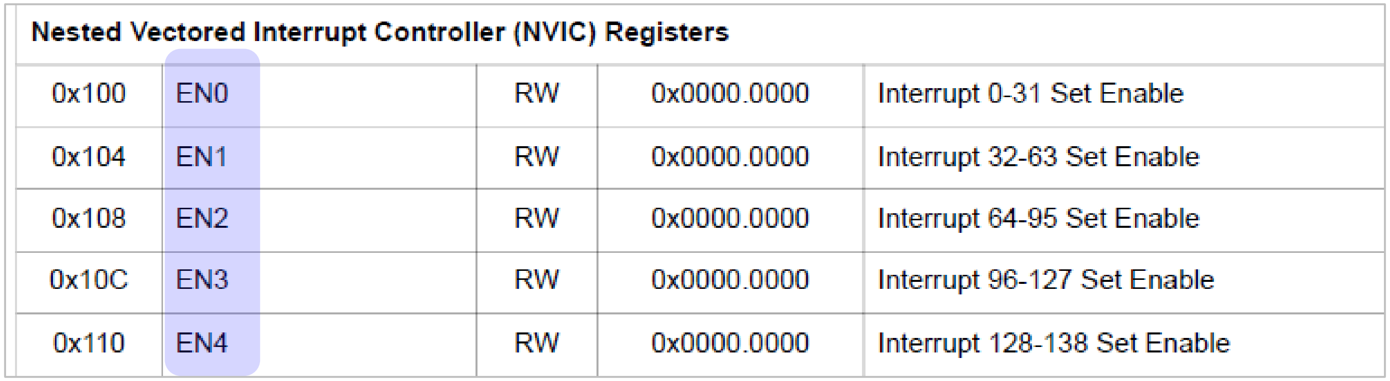
```
NVIC_SetPriority (GPIOF_IRQn, 5); /* set GPIOF intr priority to 5 */
```

Source:https://www.keil.com/pack/doc/CMSIS/Core/html/group__NVIC__gr.html#details

Enable & Disable Interrupts

TM4C123GH6PM7

NVIC Interrupt Enable Registers

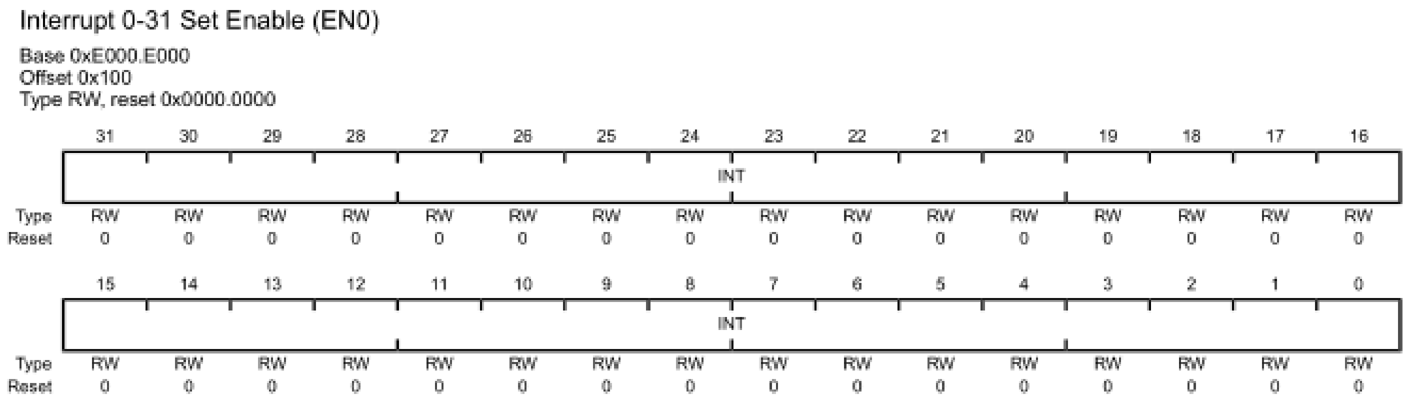


Nested Vectored Interrupt Controller (NVIC) Registers				
0x100	EN0	RW	0x0000.0000	Interrupt 0-31 Set Enable
0x104	EN1	RW	0x0000.0000	Interrupt 32-63 Set Enable
0x108	EN2	RW	0x0000.0000	Interrupt 64-95 Set Enable
0x10C	EN3	RW	0x0000.0000	Interrupt 96-127 Set Enable
0x110	EN4	RW	0x0000.0000	Interrupt 128-138 Set Enable

- Total of 5 enable registers: **EN0**, **EN1**, **EN2**, **EN3**, **EN4**. (for 138 interrupt numbers). Each EN*n* register is 32-bits wide.
  - **EN0** controls Interrupt Number 0 to 31 (Intr Vector 16 to 47).
  - **EN1** controls Interrupt Number 32 to 63 (Intr Vector 48 to 79).
  - **EN2** controls Interrupt Number 64 to 95 (Intr Vector 80 to 111).
  - **EN3** controls Interrupt Number 96 to 127 (Intr Vector 112 to 143).
  - **EN4** controls Interrupt Number 128 to 138 (Intr Vector 144 to 154).
- A bit set to ‘1’ enables the interrupt source.
- Note that the SysTick interrupt do NOT have an Intr enable register.

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf, pg134)

Interrupt Set Enable Register (EN0)


Interrupt 0-31 Set Enable (EN0)

Base 0xE000.E000
Offset 0x100
Type RW, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	INT															
Type	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	INT															
Type	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- **EN0** register enables Interrupt Numbers 0 to 31.
- Read operation:
  - If bit = ‘1’, corresponding interrupt is enabled.
  - If bit = ‘0’, corresponding interrupt is disabled.
- Write operation:
  - Writing a ‘1’ to a bit enables the corresponding interrupt.
  - Writing a ‘0’ has no effect.

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf, pg142)

Enable Interrupt through NVIC

```
void NVIC_EnableIRQ(IRQn_Type IRQn)
```

Parameters:

`IRQn` : Interrupt Number

Description:

This function enables the specified device specific interrupt IRQn.
IRQn cannot be a negative value.

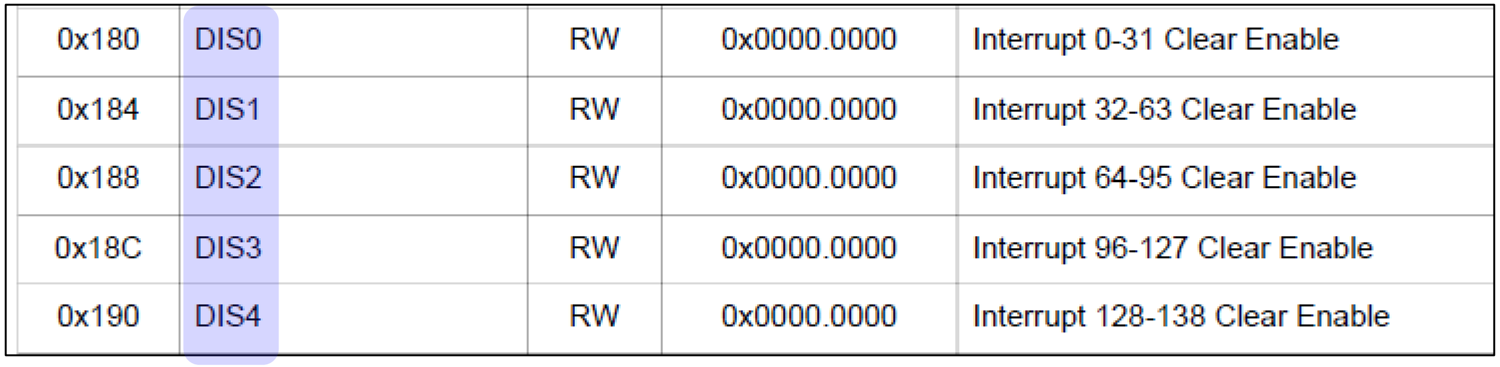
CMSIS function defined in 'core_cm.h'.

Example:

```
NVIC_EnableIRQ (GPIOF_IRQn); /* enable GPIOF interrupt */
```

Source:https://www.keil.com/pack/doc/CMSIS/Core/html/group__NVIC__gr.html#details

Interrupt Disable Registers

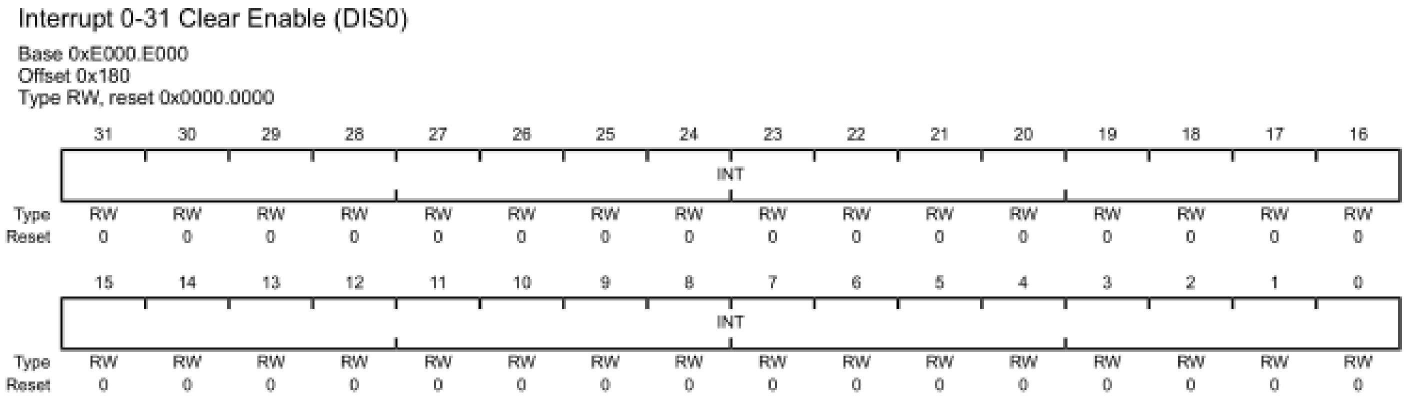


0x180	DIS0	RW	0x0000.0000	Interrupt 0-31 Clear Enable
0x184	DIS1	RW	0x0000.0000	Interrupt 32-63 Clear Enable
0x188	DIS2	RW	0x0000.0000	Interrupt 64-95 Clear Enable
0x18C	DIS3	RW	0x0000.0000	Interrupt 96-127 Clear Enable
0x190	DIS4	RW	0x0000.0000	Interrupt 128-138 Clear Enable

- Total of **5** disable registers: **DIS0**, **DIS1**, **DIS2**, **DIS3**, **DIS4**. Each register is 32-bits wide.
  - **DIS0** controls Interrupt Number 0 to 31 (Intr Vector 16 to 47).
  - **DIS1** controls Interrupt Number 32 to 63 (Intr Vector 48 to 79).
  - **DIS2** controls Interrupt Number 64 to 95 (Intr Vector 80 to 111).
  - **DIS3** controls Interrupt Number 96 to 127 (Intr Vector 112 to 143).
  - **DIS4** controls Interrupt Number 128 to 138 (Intr Vector 144 to 154).
- A bit set to ‘1’ disables the interrupt source.

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf, pg 134)

Interrupt Disable Register (DIS0)


Interrupt 0-31 Clear Enable (DIS0)

Base 0xE000.E000
Offset 0x180
Type RW, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	INT															
Type	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	INT															
Type	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- **DIS0** register enables Interrupt Numbers 0 to 31.
- Read operation:
  - If bit = ‘1’, corresponding interrupt is enabled.
  - If bit = ‘0’, corresponding interrupt is disabled.
- Write operation:
  - Writing a ‘1’ to a bit disables the corresponding interrupt (EN*n* register).
  - Writing a ‘0’ has no effect.

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf, pg144)

Disable Interrupt through NVIC

```
void NVIC_DisableIRQ(IRQn_Type IRQn)
```

Parameters:

`IRQn` : Interrupt Number

Description:

This function disables the specified device specific interrupt IRQn. IRQn cannot be a negative value.

CMSIS function defined in ‘core_cm.h’.

Example:

```
NVIC_DisableIRQ (GPIOF_IRQn); /* disable GPIOF interrupt */
```

Source:https://www.keil.com/pack/doc/CMSIS/Core/html/group__NVIC__gr.html#details

File: *NVIC.h (in Lab 1)*

```
/* System Handler Priority Registers   */
#define NVIC_SYS_PRI1_R         (*((volatile uint32_t *)0xE000ED18))
#define NVIC_SYS_PRI2_R         (*((volatile uint32_t *)0xE000ED1C))
#define NVIC_SYS_PRI3_R         (*((volatile uint32_t *)0xE000ED20))

/* Interrupt Priority Registers   */
#define NVIC_PRI0_R             (*((volatile uint32_t *)0xE000E400))
#define NVIC_PRI1_R             (*((volatile uint32_t *)0xE000E404))
#define NVIC_PRI2_R             (*((volatile uint32_t *)0xE000E408))
#define NVIC_PRI3_R             (*((volatile uint32_t *)0xE000E40C))
#define NVIC_PRI4_R             (*((volatile uint32_t *)0xE000E410))
#define NVIC_PRI5_R             (*((volatile uint32_t *)0xE000E414))
#define NVIC_PRI6_R             (*((volatile uint32_t *)0xE000E418))
#define NVIC_PRI7_R             (*((volatile uint32_t *)0xE000E41C))
#define NVIC_PRI8_R             (*((volatile uint32_t *)0xE000E420))
#define NVIC_PRI9_R             (*((volatile uint32_t *)0xE000E424))
#define NVIC_PRI10_R            (*((volatile uint32_t *)0xE000E428))
#define NVIC_PRI11_R            (*((volatile uint32_t *)0xE000E42C))
#define NVIC_PRI12_R            (*((volatile uint32_t *)0xE000E430))
#define NVIC_PRI13_R            (*((volatile uint32_t *)0xE000E434))
#define NVIC_PRI14_R            (*((volatile uint32_t *)0xE000E438))
#define NVIC_PRI15_R            (*((volatile uint32_t *)0xE000E43C))
#define NVIC_PRI16_R            (*((volatile uint32_t *)0xE000E440))
#define NVIC_PRI17_R            (*((volatile uint32_t *)0xE000E444))
#define NVIC_PRI18_R            (*((volatile uint32_t *)0xE000E448))
#define NVIC_PRI19_R            (*((volatile uint32_t *)0xE000E44C))
#define NVIC_PRI20_R            (*((volatile uint32_t *)0xE000E450))
```

File: *NVIC.h*

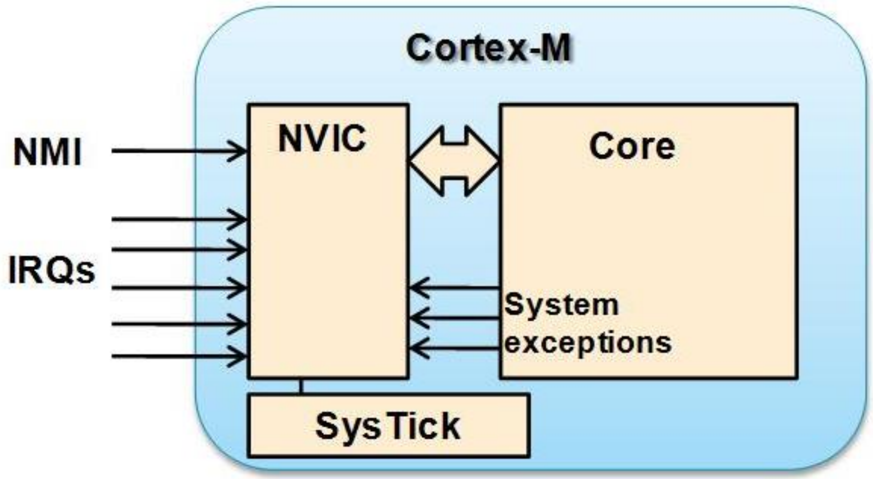
```
#define NVIC_PRI21_R            (*((volatile uint32_t *)0xE000E454))
#define NVIC_PRI22_R            (*((volatile uint32_t *)0xE000E458))
#define NVIC_PRI23_R            (*((volatile uint32_t *)0xE000E45C))
#define NVIC_PRI24_R            (*((volatile uint32_t *)0xE000E460))
#define NVIC_PRI25_R            (*((volatile uint32_t *)0xE000E464))
#define NVIC_PRI26_R            (*((volatile uint32_t *)0xE000E468))
#define NVIC_PRI27_R            (*((volatile uint32_t *)0xE000E46C))
#define NVIC_PRI28_R            (*((volatile uint32_t *)0xE000E470))
#define NVIC_PRI29_R            (*((volatile uint32_t *)0xE000E474))
#define NVIC_PRI30_R            (*((volatile uint32_t *)0xE000E478))
#define NVIC_PRI31_R            (*((volatile uint32_t *)0xE000E47C))
#define NVIC_PRI32_R            (*((volatile uint32_t *)0xE000E480))
#define NVIC_PRI33_R            (*((volatile uint32_t *)0xE000E484))
#define NVIC_PRI34_R            (*((volatile uint32_t *)0xE000E488))

/* Enable & Disable Interrupts  */
#define NVIC_EN0_R              (*((volatile uint32_t *)0xE000E100))
#define NVIC_EN1_R              (*((volatile uint32_t *)0xE000E104))
#define NVIC_EN2_R              (*((volatile uint32_t *)0xE000E108))
#define NVIC_EN3_R              (*((volatile uint32_t *)0xE000E10C))
#define NVIC_EN4_R              (*((volatile uint32_t *)0xE000E110))
#define NVIC_DIS0_R             (*((volatile uint32_t *)0xE000E180))
#define NVIC_DIS1_R             (*((volatile uint32_t *)0xE000E184))
#define NVIC_DIS2_R             (*((volatile uint32_t *)0xE000E188))
#define NVIC_DIS3_R             (*((volatile uint32_t *)0xE000E18C))
#define NVIC_DIS4_R             (*((volatile uint32_t *)0xE000E190))
```

GPIO Interrupts

GPIOIS, GPIOIBE, GPIOIEV, GPIOIM, GPIOMIS, GPIORIS, GPIOICR registers

GPIO Interrupts

- The interrupt mechanism of each GPIO port are controlled by a set of **7** registers.
- Functions of the registers include:
  - select the source of the interrupt,
  - Set interrupt trigger polarity & edge properties.
- When one or more GPIOs cause an interrupt, a <u>single</u> interrupt output is sent to the NVIC for the entire GPIO port.
- NVIC prioritizes & routes ALL interrupts to the CPU.

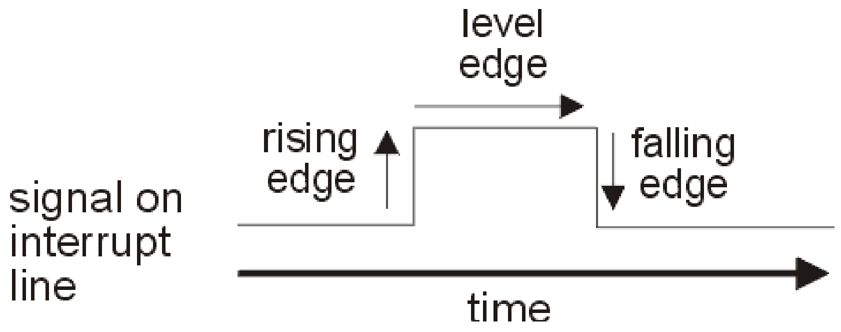


GPIO Interrupt Registers

- Set of 7 GPIO interrupt registers serves various functions:
  - Configure type of interrupt:
    - Interrupt Sense register **(GPIOIS)**
      - Level or Edge interrupt.
    - Interrupt Both Edges register **(GPIOIBE)**.
    - Interrupt Event register **(GPIOIEV)**.
  - Mask off an interrupt, if needed:
    - Interrupt Mask register **(GPIOIM)**.
  - Check which interrupt(s) had occurred:
    - Masked Interrupt Status register **(GPIOMIS)**.
    - Raw Interrupt Status register **(GPIORIS)**.
  - Clear an interrupt after it had occurred:
    - Interrupt Clear register **(GPIOICR)**.

GPIO Edge & Level Interrupts

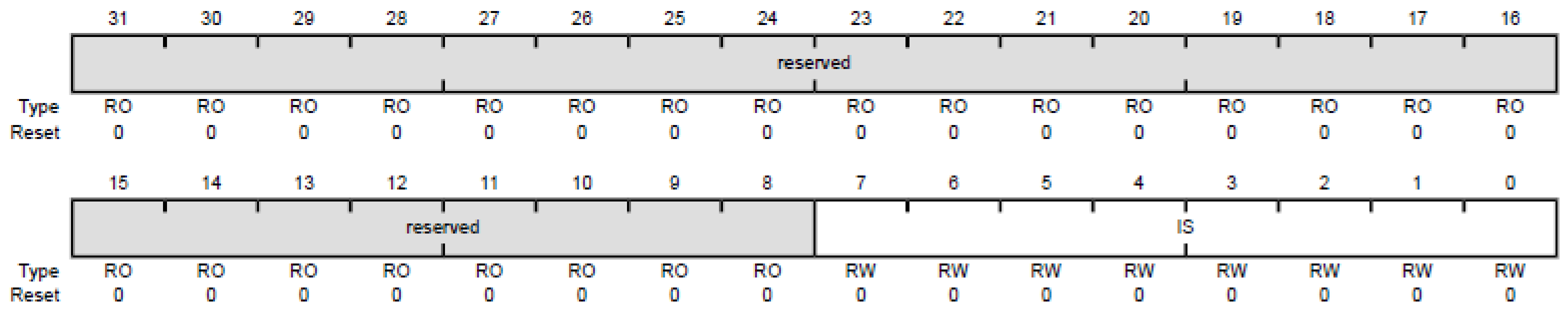
- Interrupts may be Edge-triggered or Level-sensitive.
- **Edge-triggered**:
  - Interrupts can be triggered on rising or falling clock edges or both.
  - SW program must clear the interrupt to enable any further interrupts.
- **Level-sensitive**:
  - External interrupt source must hold the level constant for it to be recognized.



GPIO Edge & Level Interrupts

- 3 registers define the **Edge** or **Level** sense that causes interrupts:
  - GPIO Interrupt Sense **(GPIOIS)** register.
  - GPIO Interrupt Both Edges **(GPIOIBE)** register.
  - GPIO Interrupt Event **(GPIOIEV)** register.

Interrupt Sense Register (GPIOIS)



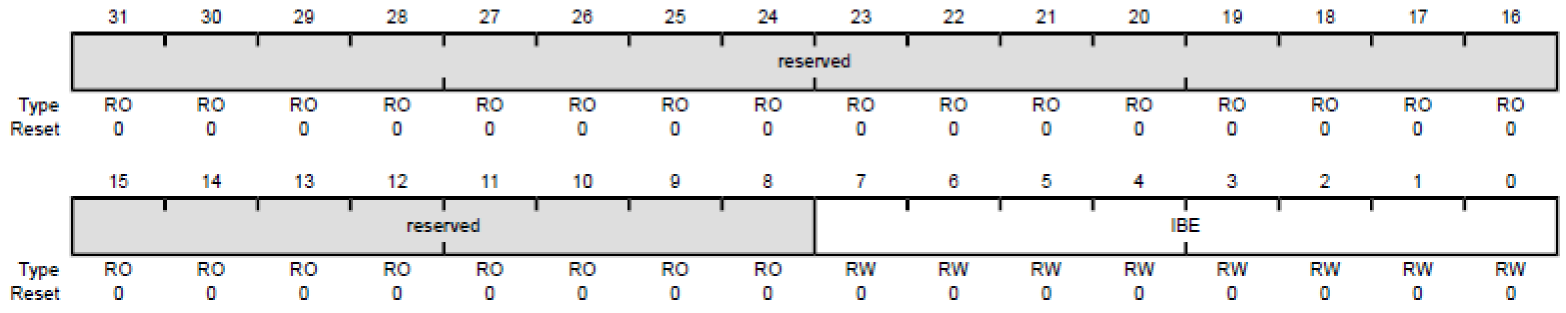
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								IS							
Type	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Setting a bit in the **GPIOIS** register configures the corresponding pin to detect levels, while clearing a bit configures the corresponding pin to detect edges.
- All bits are cleared by a reset.
- **IS** bit:
  - bit = ‘1’ : interrupt configured to be **level-sensitive**.
  - bit = ‘0’ : interrupt configured to be **edge-sensitive**.

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf, pg144)

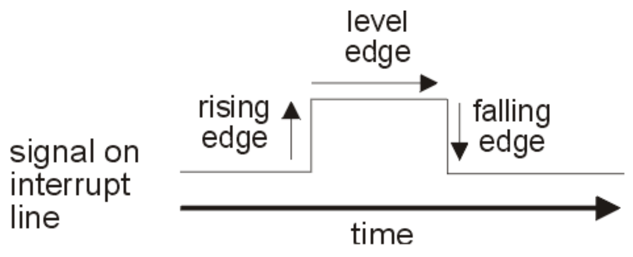
Interrupt Both Edges Register
(GPIOIBE)



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

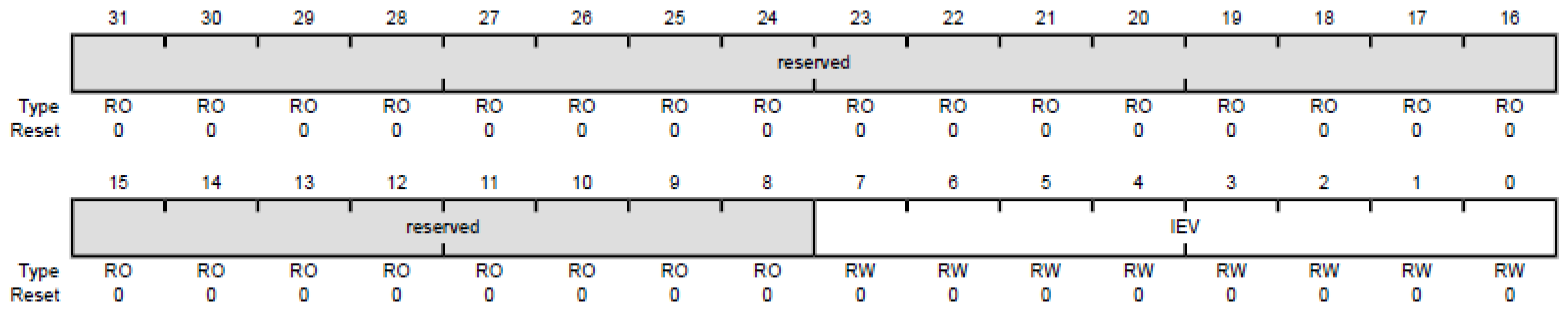
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								IBE							
Type	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- **GPIOIBE** register allows both edges to cause interrupts.
- **IBE bit:**
  - bit = ‘1’: both edges causes interrupt to be triggered.
  - bit = ‘0’: interrupt is controlled by the GPIOIEV register.



Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf, pg144)

Interrupt Event Register (GPIOIEV)



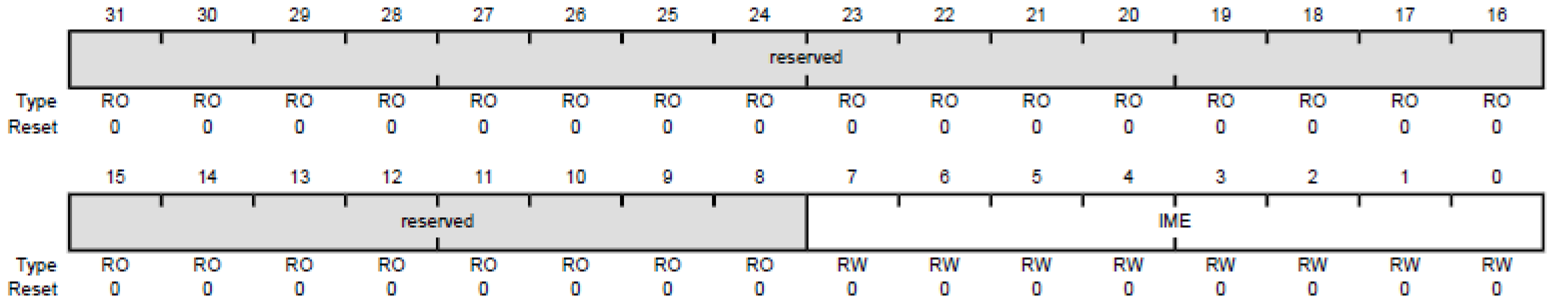
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								IEV							
Type	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Setting a bit in the **GPIOIEV** register configures the corresponding pin to detect rising edges or high levels, depending on the corresponding bit value in the **GPIO Interrupt Sense (GPIOIS)** register.
- Clearing a bit configures the pin to detect falling edges or low levels, depending on the corresponding bit value in the **GPIOIS** register. All bits are cleared by a reset.
- **IEV bits:**
  - bit = ‘1’: detect rising edge or high level at interrupt pin.
  - bit = ‘0’: detect falling edge or low level at interrupt pin.

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf, pg144)

Interrupt Mask Register (GPIOIM)



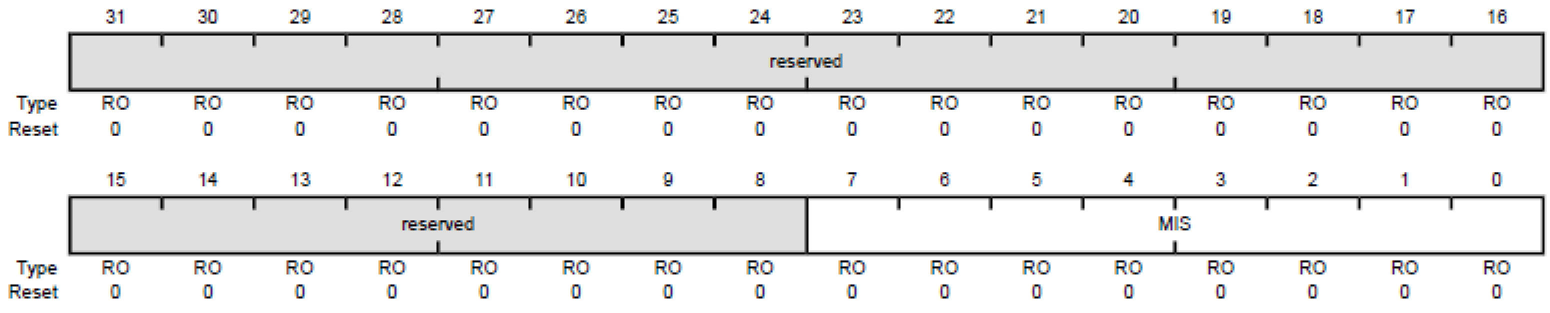
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								IME							
Type	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Setting a bit in the **GPIOIM** register allows interrupts that are generated by the corresponding pin to be sent to the NVIC.
- Clearing a bit prevents an interrupt on the corresponding pin from being sent to the NVIC.
- All bits are cleared by a reset.
- **IME bits:**
  - bit = ‘1’: allows interrupt is sent to NVIC.
  - bit = ‘0’: interrupt is masked (not sent to NVIC).

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf, pg144)

Masked Interrupt Status Register
(GPIOMIS)



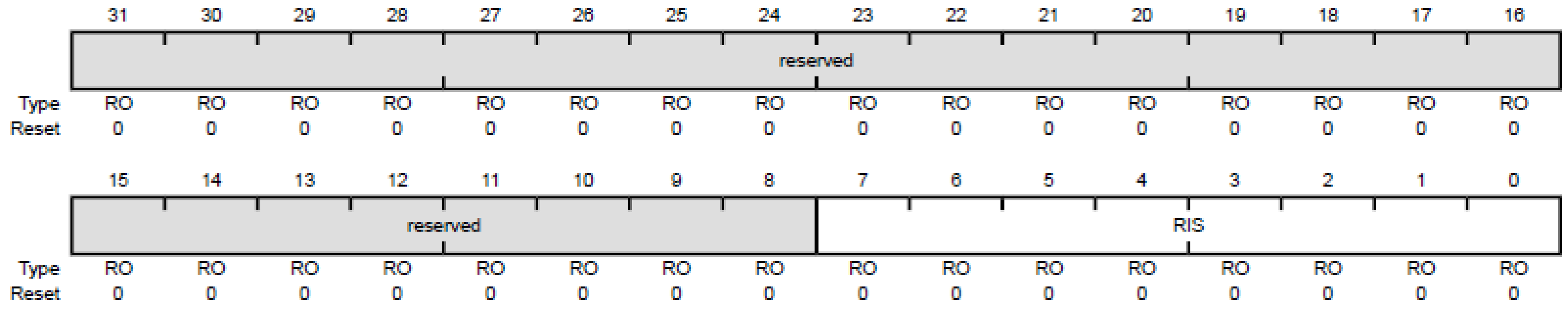
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								MIS							
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- If a bit is set in this register, the corresponding interrupt has triggered an interrupt to the NVIC.
- If a bit is cleared, either no interrupt has been generated, or the interrupt is masked.
- **MIS bits:**
  - bit = ‘1’: interrupt is triggered.
  - bit = ‘0’: no interrupt triggered.

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf, pg144)

Raw Interrupt Status Register
(GPIORIS)



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								RIS							
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- A bit in this register is set when an interrupt condition occurs on the corresponding GPIO pin.
- If the corresponding bit in the **GPIO Interrupt Mask (GPIOIM)** register is set, the interrupt is sent to the NVIC.
- Bits read as ‘0’ indicate that corresponding input pins have not initiated an interrupt.
- **RIS bits**:
  - bit = ‘1’ : interrupt has occurred for corresponding GPIO pin.
  - bit = ‘0’ : interrupt has not occurred for corresponding GPIO pin.
- The corresponding **GPIOMIS** bit reflects the masked value of the RIS bit.

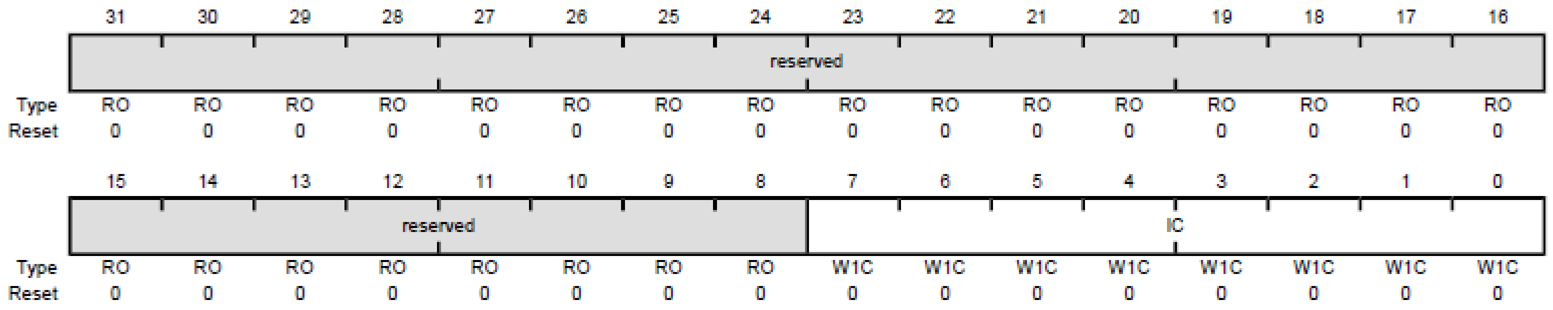
Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf, pg144)

GPIOMIS versus GPIORIS Registers

What is the difference between usage of **GPIOMIS** & **GPIORIS** registers?

- **GPIOMIS** register shows only interrupt status/conditions that are allowed to be passed to the NVIC.
- **GPIORIS** register indicates that a GPIO pin meets the conditions for an interrupt, but has not necessarily been sent to the NVIC.

Interrupt Clear Register (GPIOICR)



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								IC							
Type	RO	RO	RO	RO	RO	RO	RO	RO	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- For edge-detect interrupts, writing a 1 to the **IC** bit in the **GPIOICR** register clears the corresponding bit in the **GPIORIS** & **GPIOMIS** registers.
- If the interrupt is a level-detect, the **IC** bit in this register has no effect.
- Writing a 0 to any of the bits in the **GPIOICR** register has no effect.
- **IC bits:**
  - bit = ‘1’: interrupt is cleared.
  - bit = ‘0’: interrupt unaffected.

Source: Tiva TM4C123GH6PM Data Sheet (spms376e.pdf, pg144)

Interrupt Handler & Programming

`GPIO`*`n`*`_Handler()`

C Program Example

- The following example shows a programming example for the following:
  - GPIO initialization for SW1 and SW2 for interrupts.
    - SW1 is connected to GPIOF bit 4.
    - SW2 is connected to GPIOF bit 0.
  - Interrupt Service Routine.
- Program is not meant to be complete.

Main Program **(main.c)**

```
static volatile BOOL       g_bSystemTick = FALSE;
void GPIOF_IRQHandler( uint32_t Status );

int main()
{
  BSPInit(); /* in BSP.c   */
  SysTick_Config(SystemCoreClock/10);
  SystemCoreClockUpdate();
  IRQ_Init(); /* in IRQ.c */

  for(;;){
    if( FALSE != g_bSystemTick )
    {
      g_bSystemTick = FALSE;
      /* other C statements   */
    }
  }
}

void SysTick_Handler( void )
{
  SysTemTick = TRUE;
  /* other C statements */
}
```

Type Definitions **(`Hal.h`)**

```
/** macros & definitions      **/
#define PF_SW1          4U
#define PF_SW2          0U

#define BIT( x )        (1U<<(x))
```

Port Initialization **(Hal.c)**

```
 void Port_Init( void )
{
   /* enable GPIO port and clock to Port F (pg 340)     */
   SYSCTL->RCGCGPIO |= SYSCTL_RCGCGPIO_R5; /* enable CLOCK  to Port F */

   /* Wait for GPIOF to be ready */
   while( 0 == (SYSCTL->PRGPIO & SYSCTL_PRGPIO_R5)){};

   GPIOF->LOCK = GPIO_LOCK_KEY;      /* unlock Port F (pg 684)   */
   GPIOF->CR |= BIT(PF_SW2) | BIT(PF_SW1) );

   /** Add C statements to initialize GPIOs for button SW1 and SW2   **/
   /** See previous week's lecture example                           **/

   /** enable interrupts for SW1 and SW2                             **/
   /** intr mask reg - mask interrups first, enable after configured **/
   GPIOF->IM &= ~( BIT(PF_SW1) | BIT(PF_SW2) );  /* mask intrs           */
   GPIOF->IBE &= ~( BIT(PF_SW1) | BIT(PF_SW2) ); /* disable both edges    */
   GPIOF->IS &= ~( BIT(PF_SW1) | BIT(PF_SW2) );  /* edge sensitive        */
   GPIOF->IEV &= ~( BIT(PF_SW1) | BIT(PF_SW2) ); /* detect falling edges  */
   GPIOF->IM |= BIT(PF_SW1) | BIT(PF_SW2);       /* enable intrs in mask
}
```

Interrupt Handler **(IRQ.c)**

```
/* initialize NVIC  */
void IRQ_Init( void )
{
   NVIC_SetPriority (GPIOF_IRQn, 5);
   NVIC_EnableIRQ( GPIOF_IRQn);
}

/* ISR: interrupt handler for PORT F interrupt  */
void GPIOF_Handler(void)
{
   uint32_t status = GPIOF->RIS; /* read & store interrupt status */
   GPIOF_Button_IRQHandler (status);  /* ISR function in main.c   */
}
```

Main Program (main.c)

```
void GPIOF_Button_IRQHandler( uint32_t Status )
{
   /* check if it is SW2 (PF0) intr  */
   if( 0 != (Status & BIT(PF_SW2 ) ))
   {
      GPIOF->ICR = BIT(PF_SW2); /* clear intr bit  */
      /* add C statements to process SW2 intr */

   }

   /* check if it is SW1 (PF4) interrupt   */
   if( 0 != (Status & BIT(PF_SW1) ))
   {
      GPIOF->ICR = BIT(PF_SW1); /* clear intr bit  */
      /* add C statements to process SW1 intr */
   }
}
```

Clearing Interrupt

- When should we clear interrupt status within the ISR?
  - After a write to the ICR register, it may take several system clock cycles before the NVIC will see the interrupt source deassert.
  - If interrupt clear is done at the end of the ISR, it is possible for the ISR to complete and the NVIC still sees the interrupt as being asserted (.. a race condition depending on code timing, sequence, other interrupts, …).
    - It could cause the ISR to be re-entered erroneously.
  - In practice, we should therefore clear the interrupt status at the beginning of the ISR.

Review Questions

1. Which System Exceptions had priorities that are preset?
2. How do we set the System Exception priorities that are user-programmable?
3. What is the difference between edge-triggered and level-sensitive interrupts?
4. “Interrupts are asynchronous”. What do you understand by this statement?
5. What is the Interrupt Number for GPIO port C? What is the corresponding address in the Interrupt Vector Table that contains the pointer to the ISR?
6. If there are multiple interrupts occurring at the same time, how are these presented to the ARM CPU?
7. How many levels of interrupt priority are supported by the NVIC? Explain how the number of levels comes about.
8. How many GPIO Interrupt registers are there?
9. Which register do we read in order to identify the source of a GPIO interrupt?
10. Which register would you use to clear an interrupt?
11. Why is it necessary to clear an interrupt in an ISR?
12. Why should an ISR be kept to execute as short a time as possible?
13. What does this CMSIS function do?

```
NVIC_SetPriority (GPIOC_IRQn, 5);
```