Interrupt Basics, Edge-Triggered Interrupts and SysTick Interrupt

By Dr. Jonathan Valvano
Modified by Dr. Min He
Updated by David Mahakian

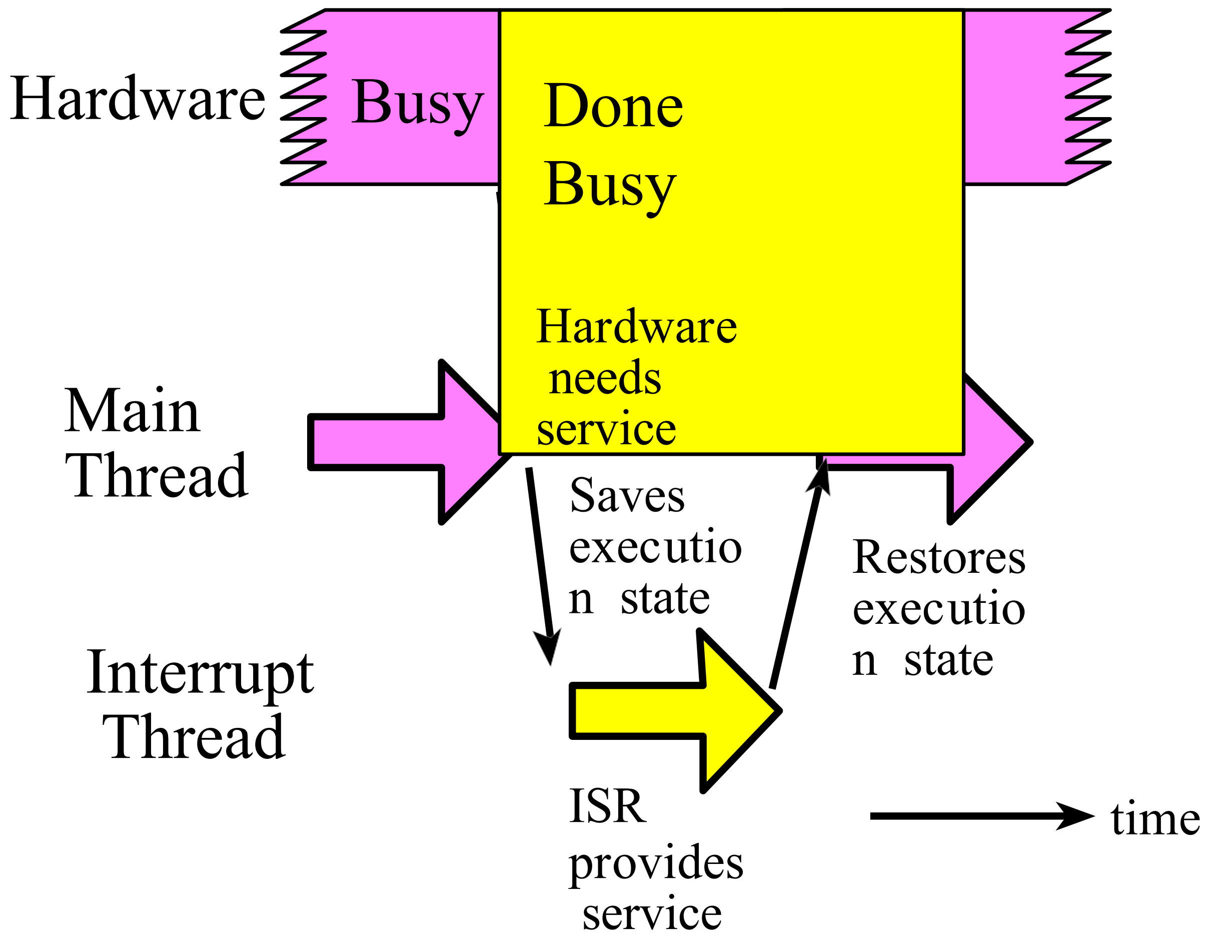
Agenda

- ❑Interrupt: review concept
- ❑Interrupt Processing
- ❑Common interrupt components
- ❑ARM Cortex M4 interrupt programming: arm, enable, trigger, vector, priority, acknowledge.
- ❑Edge Triggered Interrupts
- ❑SysTick Periodic Interrupt

Interrupts

- An **interrupt** is the automatic transfer of software execution in response to a hardware event (**trigger**) that is asynchronous with current software execution.
  - external I/O device (like a keyboard or printer) or
  - an internal event (like an op code fault, or a periodic timer.)
- Occurs when the hardware needs or can service (busy to done state transition)

Interrupt Processing



A **thread** is defined as the path of action of software as it executes.
A hardware event is called a **trigger.**

Common Interrupt Components

- The ability for the hardware to request action from computer
- The ability for the computer to determine the source: polled vs. vectored
- The ability for the computer to acknowledge the interrupt

Polled vs. Vectored Interrupts

- **vectored** interrupt system
  - separate connections per device
    = separate vector address per device
- **polled** interrupt system
  - interrupt software must poll each device
    - look for the device that requested the interrupt
- Most interrupts on the TM4C microcontrollers are vectored, but there are some triggers that share the same vector. Ex: GPIO Ports.

NVIC on the ARM Cortex-M Processor

- **Exceptions** include resets, software interrupts and hardware interrupts.
- **NVIC** (Nested Vectored Interrupt Controller) : control all exceptions
- Each exception has an associated 32-bit **vector** that points to the memory location where the ISR that handles the exception is located.
- Vectors are stored in **ROM** at the beginning of memory. See startup.s.

Interrupt Vector Table: Table 2-9 in Datasheet

Vector address	Number	IRQ	ISR name in Startup.s	NVIC	Priority bits
0x00000038	14	-2	PendSV_Handler	NVIC_SYS_PRI3_R	23 – 21
0x0000003C	15	-1	SysTick_Handler	NVIC_SYS_PRI3_R	31 – 29
0x00000040	16	0	GPIOPortA_Handler	NVIC_PRI0_R	7 – 5
0x00000044	17	1	GPIOPortB_Handler	NVIC_PRI0_R	15 – 13
0x00000048	18	2	GPIOPortC_Handler	NVIC_PRI0_R	23 – 21
0x0000004C	19	3	GPIOPortD_Handler	NVIC_PRI0_R	31 – 29
0x00000050	20	4	GPIOPortE_Handler	NVIC_PRI1_R	7 – 5
0x00000054	21	5	UART0_Handler	NVIC_PRI1_R	15 – 13
0x00000058	22	6	UART1_Handler	NVIC_PRI1_R	23 – 21
0x0000005C	23	7	SSI0_Handler	NVIC_PRI1_R	31 – 29
0x00000060	24	8	I2C0_Handler	NVIC_PRI2_R	7 – 5
0x00000064	25	9	PWMFault_Handler	NVIC_PRI2_R	15 – 13
0x00000068	26	10	PWM0_Handler	NVIC_PRI2_R	23 – 21
0x0000006C	27	11	PWM1_Handler	NVIC_PRI2_R	31 – 29
0x00000070	28	12	PWM2_Handler	NVIC_PRI3_R	7 – 5
0x00000074	29	13	Quadrature0_Handler	NVIC_PRI3_R	15 – 13
0x00000078	30	14	ADC0_Handler	NVIC_PRI3_R	23 – 21
0x0000007C	31	15	ADC1_Handler	NVIC_PRI3_R	31 – 29
0x00000080	32	16	ADC2_Handler	NVIC_PRI4_R	7 – 5
0x00000084	33	17	ADC3_Handler	NVIC_PRI4_R	15 – 13
0x00000088	34	18	WDT_Handler	NVIC_PRI4_R	23 – 21
0x0000008C	35	19	Timer0A_Handler	NVIC_PRI4_R	31 – 29
0x00000090	36	20	Timer0B_Handler	NVIC_PRI5_R	7 – 5
0x00000094	37	21	Timer1A_Handler	NVIC_PRI5_R	15 – 13
0x00000098	38	22	Timer1B_Handler	NVIC_PRI5_R	23 – 21
0x0000009C	39	23	Timer2A_Handler	NVIC_PRI5_R	31 – 29
0x000000A0	40	24	Timer2B_Handler	NVIC_PRI6_R	7 – 5
0x000000A4	41	25	Comp0_Handler	NVIC_PRI6_R	15 – 13
0x000000A8	42	26	Comp1_Handler	NVIC_PRI6_R	23 – 21
0x000000AC	43	27	Comp2_Handler	NVIC_PRI6_R	31 – 29
0x000000B0	44	28	SysCtl_Handler	NVIC_PRI7_R	7 – 5
0x000000B4	45	29	FlashCtl_Handler	NVIC_PRI7_R	15 – 13
0x000000B8	46	30	GPIOPortF_Handler	NVIC_PRI7_R	23 – 21
0x000000BC	47	31	GPIOPortG_Handler	NVIC_PRI7_R	31 – 29
0x000000C0	48	32	GPIOPortH_Handler	NVIC_PRI8_R	7 – 5
0x000000C4	49	33	UART2_Handler	NVIC_PRI8_R	15 – 13
0x000000C8	50	34	SSI1_Handler	NVIC_PRI8_R	23 – 21
0x000000CC	51	35	Timer3A_Handler	NVIC_PRI8_R	31 – 29
0x000000D0	52	36	Timer3B_Handler	NVIC_PRI9_R	7 – 5
0x000000D4	53	37	I2C1_Handler	NVIC_PRI9_R	15 – 13
0x000000D8	54	38	Quadrature1_Handler	NVIC_PRI9_R	23 – 21
0x000000DC	55	39	CAN0_Handler	NVIC_PRI9_R	31 – 29
0x000000E0	56	40	CAN1_Handler	NVIC_PRI10_R	7 – 5
0x000000E4	57	41	CAN2_Handler	NVIC_PRI10_R	15 – 13
0x000000E8	58	42	Ethernet_Handler	NVIC_PRI10_R	23 – 21
0x000000EC	59	43	Hibernate_Handler	NVIC_PRI10_R	31 – 29
0x000000F0	60	44	USB0_Handler	NVIC_PRI11_R	7 – 5
0x000000F4	61	45	PWM3_Handler	NVIC_PRI11_R	15 – 13
0x000000F8	62	46	uDMA_Handler	NVIC_PRI11_R	23 – 21
0x000000FC	63	47	uDMA_Error	NVIC_PRI11_R	31 – 29

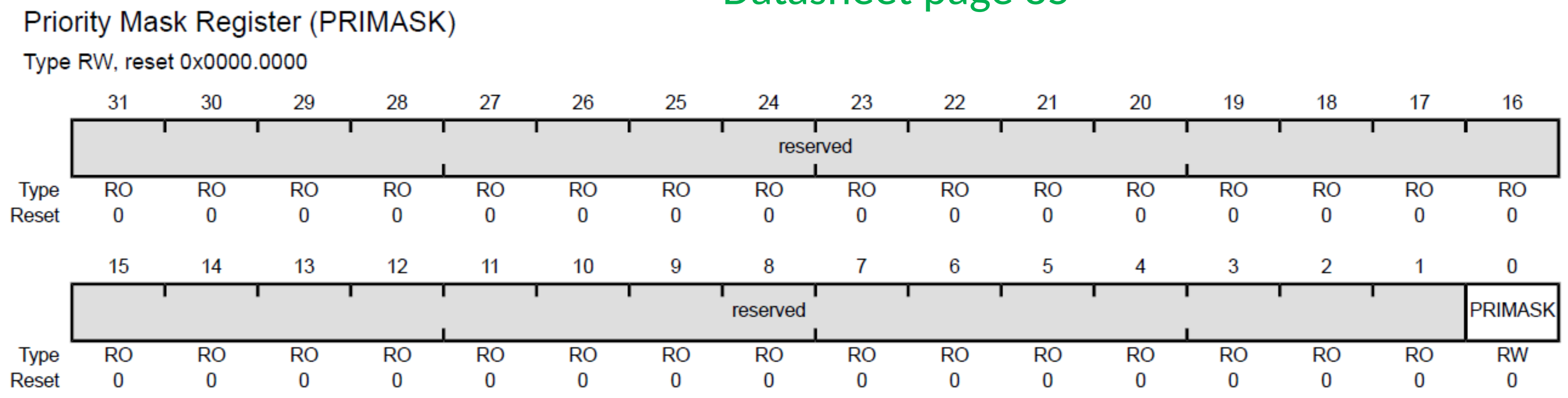
ARM Cortex-M Interrupts Software Controls

- Each potential interrupt source has a separate **arm** bit
  - Set for those devices from which it wishes to accept interrupts,
  - Deactivate in those devices from which interrupts are not allowed
- Each potential interrupt source has a separate **flag** bit
  - hardware sets the flag when it wishes to request an interrupt
  - software clears the flag in ISR to signify it is processing the request
- Interrupt **enable** conditions in processor
  - Global interrupt enable bit, I, in **PRIMASK** register: In C, we enable and disable interrupts by calling the functions **EnableInterrupts()** and **DisableInterrupts()** respectively.
  - Priority level, **BASEPRI**, of allowed interrupts (0 = all)

Priority Mask Register (PRIMASK)

- This register prevents activation of all exceptions with programmable priority.
- PRIMASK bit: bit 0, we call it bit I.
  - 1: means most interrupts and exceptions are not allowed, which we define as **disabled**.
  - 0: means interrupts are allowed, which we define as **enabled**.

*Datasheet page 85


Priority Mask Register (PRIMASK)

Type RW, reset 0x0000.0000

	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

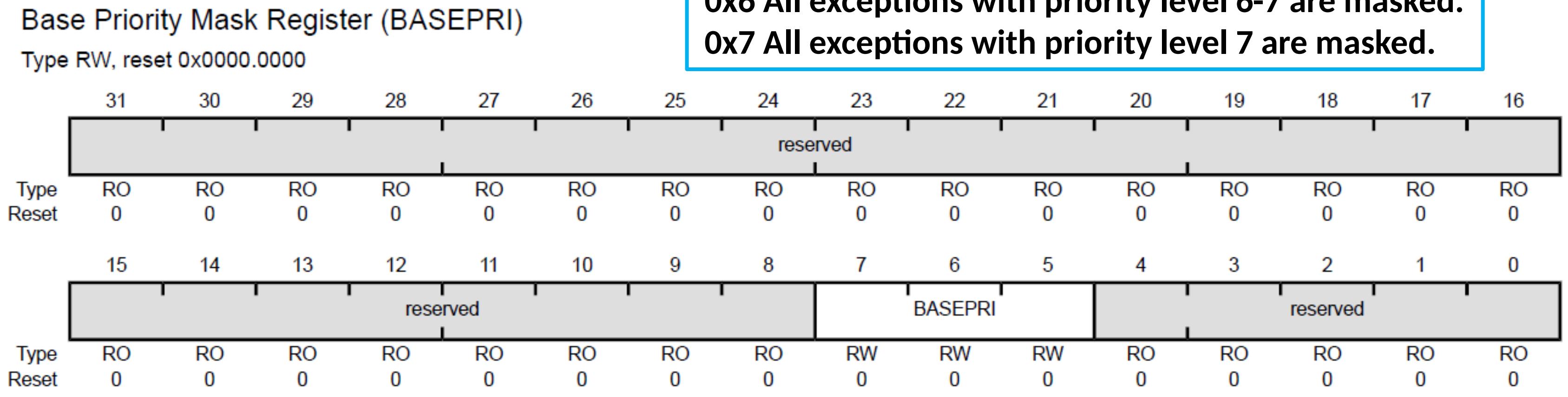
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved															PRIMASK
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Base Priority Mask Register (BASEPRI)

- ❑ Defines the minimum priority for exception processing.
- ❑ When **BASEPRI** is set to a nonzero value, it prevents the activation of all exceptions with the same or lower priority level as the **BASEPRI** value.
- ❑ Exceptions should be disabled when they might impact the timing of critical tasks.

Value	**Description**
0x0	**All exceptions are unmasked(default setting).**
0x1	**All exceptions with priority level 1-7 are masked.**
0x2	**All exceptions with priority level 2-7 are masked.**
0x3	**All exceptions with priority level 3-7 are masked.**
0x4	**All exceptions with priority level 4-7 are masked.**
0x5	**All exceptions with priority level 5-7 are masked.**
0x6	**All exceptions with priority level 6-7 are masked.**
0x7	**All exceptions with priority level 7 are masked.**

*Datasheet page 87


Base Priority Mask Register (BASEPRI)

Type RW, reset 0x0000.0000

	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								BASEPRI			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

Global Interrupt Control

- Enable interrupt globally by **clear** the I bit in register PRIMASK
  - By default it is set, meaning interrupt is globally disabled.
  - Functions to enable/disable interrupt globally:
    EnableInterrupts(), DisableInterrupts()
    Defined in startup.c

- Set global interrupt priority at **BASEPRI register**
  - By default it is set to 0 to allow all exceptions
  - 8 levels: 0 to 7
  - Interrupts with priority < BASEPRI can interrupt
    lower number => higher priority

Interrupts Source Controls

- Separate **arm** bit per interrupt source
  - armed = can trigger interrupt
  - disarmed = will not trigger interrupt
- Separate **flag** bit per interrupt source
  - Set when hardware requests interrupt
  - Interrupt handler must clear flag

Interrupt Conditions

- Five conditions must be true simultaneously for an interrupt to occur:
  1. Arm: control bit for each possible source is set
  2. NVIC enable
  3. Enable: interrupts globally enabled (I=0 in PRIMASK)
  4. Level: interrupt level must be less than BASEPRI
  5. Trigger: hardware action sets source-specific flag
- Interrupt remains **pending** if trigger is set but any other condition is not true
  - Interrupt serviced once all conditions become true
- Need to **acknowledge** interrupt
  - Clear trigger flag in ISR or will get endless interrupts!

Interrupt Initialization

- Things you must do to initialize an interrupt
  - Initialize data structures (counters, pointers)
  - Arm (specify a flag may interrupt)
  - Configure NVIC
    - Enable interrupt (NVIC_EN0_R)
    - Set priority (e.g., NVIC_PRI1_R)
  - Enable Interrupts
    - Assembly code **`CPSIE I`**
    - C code **`EnableInterrupts();`**

NVIC_Enx_R

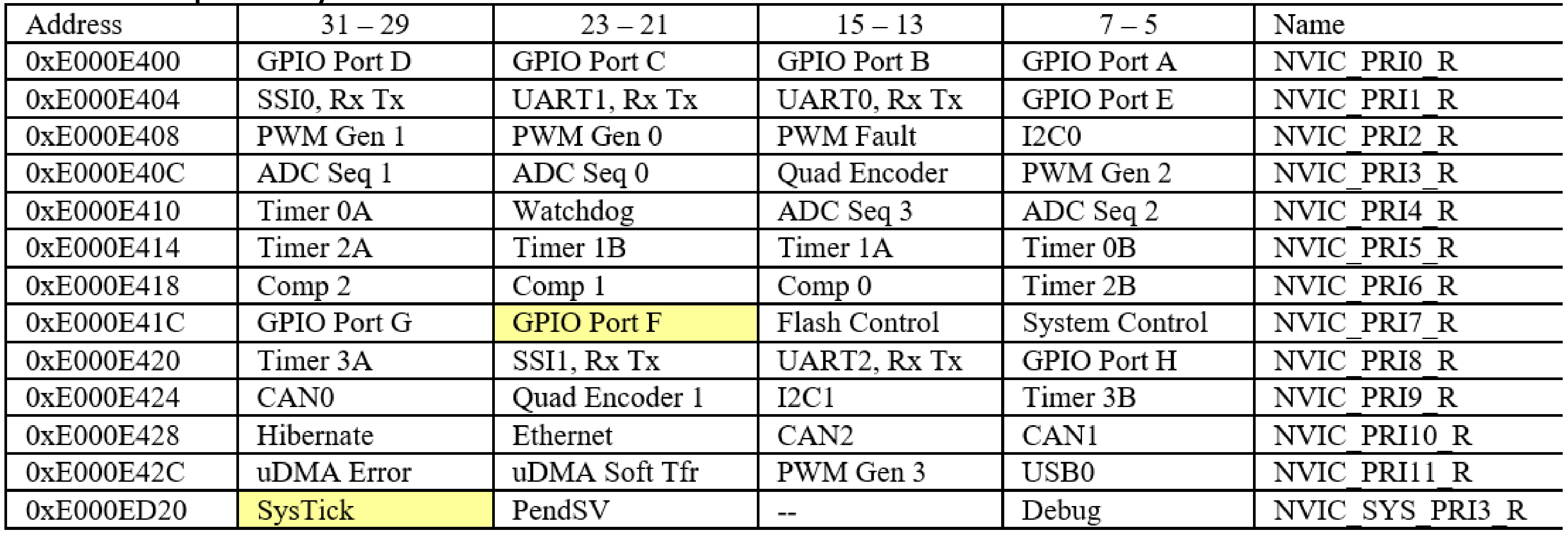
- There are five enable registers defining 139 interrupt enable bits.
- Each one is 32 bits, takes care of 32 interrupts:
  - **NVIC_EN0_R** controls the IRQ numbers 0 to 31.
  - **NVIC_EN1_R** controls the IRQ numbers 32 to 63.
  - **NVIC_EN2_R** controls the IRQ numbers 64 to 95.
  - **NVIC_EN3_R** controls the IRQ numbers 96 to 127.
  - **NVIC_EN4_R** controls the IRQ numbers 128 to 138.

Observation: There are many interrupt sources, but an effective system will use only a few.

Address	31	30	29-7	6	5	4	3	2	1	0	Name
0xE000E100	G	F	…	UART1	UART0	E	D	C	B	A	NVIC_EN0_R
0xE000E104			…						UART2	H	NVIC_EN1_R

Priority Registers: NVIC_PRIx_R/ NVIC_SYS_PRIx_R

- ❑ Two types of priority registers:
  - ❖ NVIC_**PRI**x_R: Interrupt **PRI**ority PRI0 - PRI34
  - ❖ NVIC_**SYS_PRI**x_R: **Sys**tem Handler **Pri**ority
- ❑ Each register contains an 8-bit priority field for four devices.
- ❑ Each interrupt source gets 8 bits to define priority, but only top 3 bits are used in TM4C124.
- ❑ Priority level 0 to 7, with 0 being the highest priority.
- ❑ A higher priority interrupt can suspend the execution of a lower priority ISR.



Address	31 – 29	23 – 21	15 – 13	7 – 5	Name
0xE000E400	GPIO Port D	GPIO Port C	GPIO Port B	GPIO Port A	NVIC_PRI0_R
0xE000E404	SSI0, Rx Tx	UART1, Rx Tx	UART0, Rx Tx	GPIO Port E	NVIC_PRI1_R
0xE000E408	PWM Gen 1	PWM Gen 0	PWM Fault	I2C0	NVIC_PRI2_R
0xE000E40C	ADC Seq 1	ADC Seq 0	Quad Encoder	PWM Gen 2	NVIC_PRI3_R
0xE000E410	Timer 0A	Watchdog	ADC Seq 3	ADC Seq 2	NVIC_PRI4_R
0xE000E414	Timer 2A	Timer 1B	Timer 1A	Timer 0B	NVIC_PRI5_R
0xE000E418	Comp 2	Comp 1	Comp 0	Timer 2B	NVIC_PRI6_R
0xE000E41C	GPIO Port G	GPIO Port F	Flash Control	System Control	NVIC_PRI7_R
0xE000E420	Timer 3A	SSI1, Rx Tx	UART2, Rx Tx	GPIO Port H	NVIC_PRI8_R
0xE000E424	CAN0	Quad Encoder 1	I2C1	Timer 3B	NVIC_PRI9_R
0xE000E428	Hibernate	Ethernet	CAN2	CAN1	NVIC_PRI10_R
0xE000E42C	uDMA Error	uDMA Soft Tfr	PWM Gen 3	USB0	NVIC_PRI11_R
0xE000ED20	SysTick	PendSV	--	Debug	NVIC_SYS_PRI3_R

Edge Triggered Interrupts

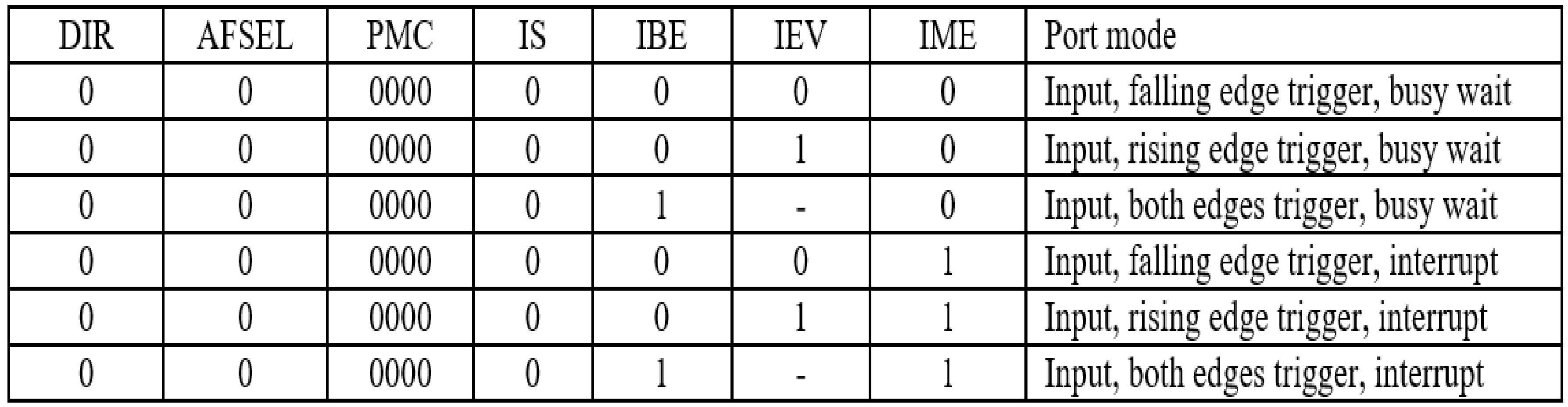
Edge-Triggered Interrupt

- ARM Cortex M4 GPIO ports support two types of interrupts:
  - Edge-detect(triggered) interrupt: corresponding bit will be set at GPIORIS register when required edge is detected.
  - Level-detect interrupt: interrupt signal much be held until serviced.
- Support three types of edge triggers:
  - rising edge (low to high)
  - falling edge (high to low)
  - both edges
- Any TM4C123 GPIO pin can be configured to generate edge interrupt when configuring as standard digital GPIO input.

Set Up Edge Triggered Interrupts

- Five conditions must be simultaneously true for an edge-triggered interrupt to be requested:
  1. The trigger flag bit is set at **RIS** register by hardware, software clears it at initialization.
  2. The arm bit (IME) in **IM** register is set.
  3. The priority level of the edge-triggered interrupt must be less than **BASEPRI**
  4. The edge-triggered interrupt must be enabled in the **NVIC_EN0_R**
  5. The I bit, bit 0 of the special register **PRIMASK**, is 0

Edge-triggered Interrupt Modes

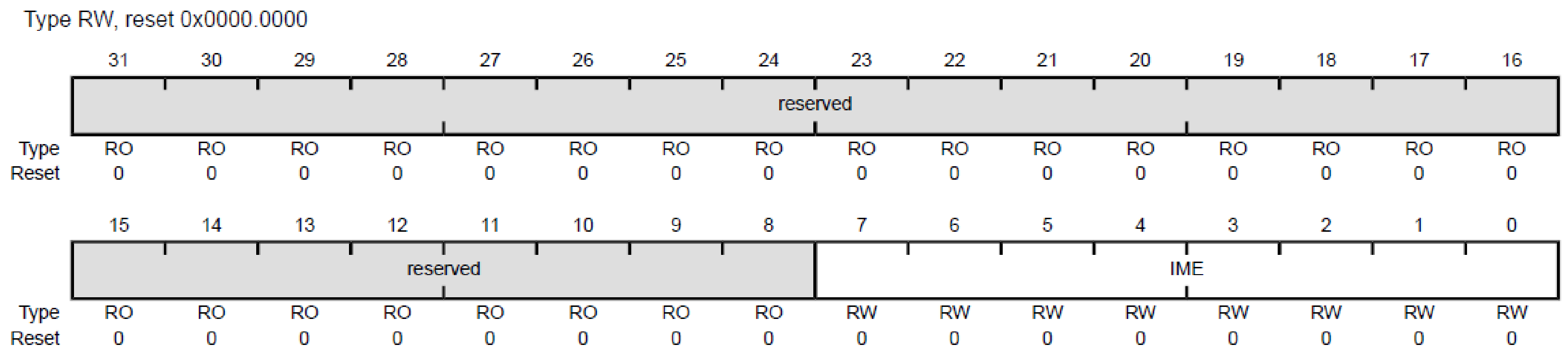


DIR	AFSEL	PMC	IS	IBE	IEV	IME	Port mode
0	0	0000	0	0	0	0	Input, falling edge trigger, busy wait
0	0	0000	0	0	1	0	Input, rising edge trigger, busy wait
0	0	0000	0	1	-	0	Input, both edges trigger, busy wait
0	0	0000	0	0	0	1	Input, falling edge trigger, interrupt
0	0	0000	0	0	1	1	Input, rising edge trigger, interrupt
0	0	0000	0	1	-	1	Input, both edges trigger, interrupt

set IME=1 to arm interrupt.
Datasheet: table 10-4.

Interrupt Mask Register (GPIOIM) GPIO_PORTx_IM_R

Setting a bit in the **GPIOIM** register allows interrupts that are generated by the corresponding pin to be sent to the interrupt controller on the combined interrupt signal.



Type RW, reset 0x0000.0000

	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

Edge/Level Interrupt Registers

- GPIOIS / GPIO_PORTx_IS_R [**I**nterrupt **S**ense]
  - 0 = edge triggered, 1 = level triggered
- GPIOIBE / GPIO_PORTx_IBE_R [**I**nterrupt **B**oth **E**dges]
  - 0 = use GPIOIEV, 1 = both edges
- GPIOIEV / GPIO_PORTx_IEV_R [**I**nterrupt **Ev**ent]
  - GPIOIS = 0: 0 = falling edge trigger, 1 = rising edge trigger
  - GPIOIS = 1: 0 = low level trigger, 1 = high level trigger

The Role of ICR/RIS Registers

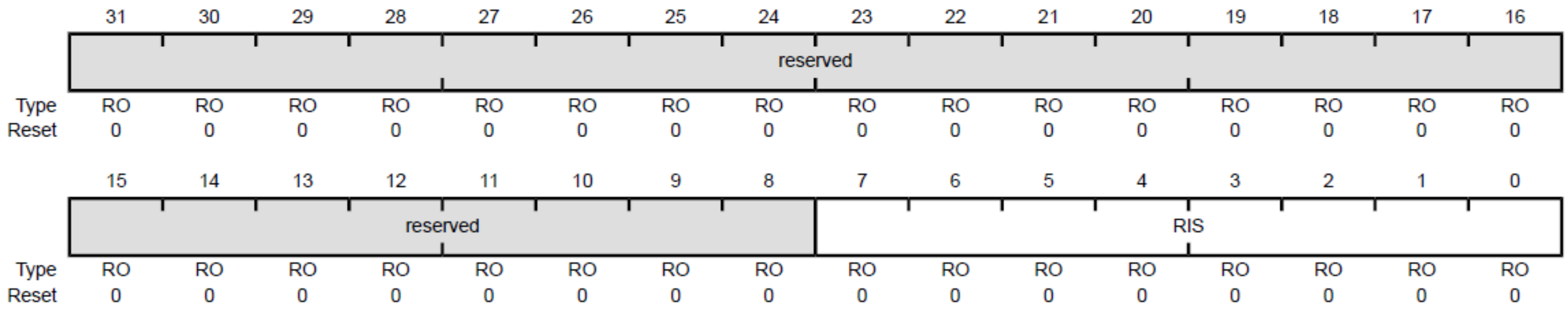
- RIS:
  - holds the trigger flags,
  - set by hardware,
  - cleared by software writing **1** to corresponding bits in ICR register in the ISR.
- ICR:
  - interrupt clear register.
  - It is write-only: use an "=" instead of "|=" for writing "1" to certain bits.
  - Writing **ones to ICR** register will **clear** corresponding bits in the **RIS register**.
  - Writing zeros to the ICR register has no effect.

Raw Interrupt Status Register (GPIORIS): GPIO_PORTx_RIS_R

- ❑ 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 interrupt controller.
- ❑ For a GPIO edge-detect interrupt, the RIS bit in the **GPIORIS** register is cleared by writing a '1' to the corresponding bit in the **GPIO Interrupt Clear (GPIOICR)** register.
- ❑ Value Description
  - ➢ 0: An interrupt condition has not occurred on the corresponding pin.
  - ➢ 1: An interrupt condition has occurred on the corresponding pin.
- ❑Code Example:

```
if (GPIO_PORTF_RIS_R & 0x01) { // switch 2 is pressed
  GPIO_PORTF_ICR_R = 0x01;      // acknowledge flag 0
}
```

*Datasheet page 668.



	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

Address	7	6	5	4	3	2	1	0	Name
$4000.43FC	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	GPIO_PORTA_DATA_R
$4000.4400	DIR	DIR	DIR	DIR	DIR	DIR	DIR	DIR	GPIO_PORTA_DIR_R
$4000.4404	IS	IS	IS	IS	IS	IS	IS	IS	GPIO_PORTA_IS_R
$4000.4408	IBE	IBE	IBE	IBE	IBE	IBE	IBE	IBE	GPIO_PORTA_IBE_R
$4000.440C	IEV	IEV	IEV	IEV	IEV	IEV	IEV	IEV	GPIO_PORTA_IEV_R
$4000.4410	IME	IME	IME	IME	IME	IME	IME	IME	GPIO_PORTA_IM_R
$4000.4414	RIS	RIS	RIS	RIS	RIS	RIS	RIS	RIS	GPIO_PORTA_RIS_R
$4000.4418	MIS	MIS	MIS	MIS	MIS	MIS	MIS	MIS	GPIO_PORTA_MIS_R
$4000.441C	ICR	ICR	ICR	ICR	ICR	ICR	ICR	ICR	GPIO_PORTA_ICR_R
$4000.4420	SEL	SEL	SEL	SEL	SEL	SEL	SEL	SEL	GPIO_PORTA_AFSEL_R
$4000.4500	DRV2	DRV2	DRV2	DRV2	DRV2	DRV2	DRV2	DRV2	GPIO_PORTA_DR2R_R
$4000.4504	DRV4	DRV4	DRV4	DRV4	DRV4	DRV4	DRV4	DRV4	GPIO_PORTA_DR4R_R
$4000.4508	DRV8	DRV8	DRV8	DRV8	DRV8	DRV8	DRV8	DRV8	GPIO_PORTA_DR8R_R
$4000.450C	ODE	ODE	ODE	ODE	ODE	ODE	ODE	ODE	GPIO_PORTA_ODR_R
$4000.4510	PUE	PUE	PUE	PUE	PUE	PUE	PUE	PUE	GPIO_PORTA_PUR_R
$4000.4514	PDE	PDE	PDE	PDE	PDE	PDE	PDE	PDE	GPIO_PORTA_PDR_R
$4000.4518	SLR	SLR	SLR	SLR	SLR	SLR	SLR	SLR	GPIO_PORTA_SLR_R
$4000.451C	DEN	DEN	DEN	DEN	DEN	DEN	DEN	DEN	GPIO_PORTA_DEN_R
$4000.4524	CR	CR	CR	CR	CR	CR	CR	CR	GPIO_PORTA_CR_R
$4000.4528	AMSEL	AMSEL	AMSEL	AMSEL	AMSEL	AMSEL	AMSEL	AMSEL	GPIO_PORTA_AMSEL_R

	31-28	27-24	23-20	19-16	15-12	11-8	7-4	3-0	
$4000.452C	PMC7	PMC6	PMC5	PMC4	PMC3	PMC2	PMC1	PMC0	GPIO_PORTA_PCTL_R
$4000.4520	LOCK (32 bits)								GPIO_PORTA_LOCK_R

Setting Interrupt Priority

Example: Set GPIO Port F Interrupt to Priority 5

NVIC_PRI7_R = (NVIC_PRI7_R&0xFF1FFFF) | 0x00A00000;

- **Each NVIC_PRIx_R / NVIC_SYS_PRIx_R register has 4 8-bit fields in it**
- **From the table on slide 17 we see that the GPIO Port F interrupt priority bits are in NVIC_PRI7_R bits 23-16.**

31 – 29	23 – 21	15 – 13	7 – 5	Name
GPIO Port G	GPIO Port F	Flash Control	System Control	NVIC_PRI7_R

31-24 23-16 15-8 7-0

- **We clear the existing bits in 23-16 by ANDing the bits we want to clear with 0. (ANDing with 1 does not change the value)**

F F **1** **F** F F F F

- **If we want priority 5, we must set the top 3 bits of 23-16 to 5 in hex: 101**

101x xxxx

- **We set the other 5 bits we aren't using to 0 and convert to hex**
- **We put 0s in the other fields that we aren't using**

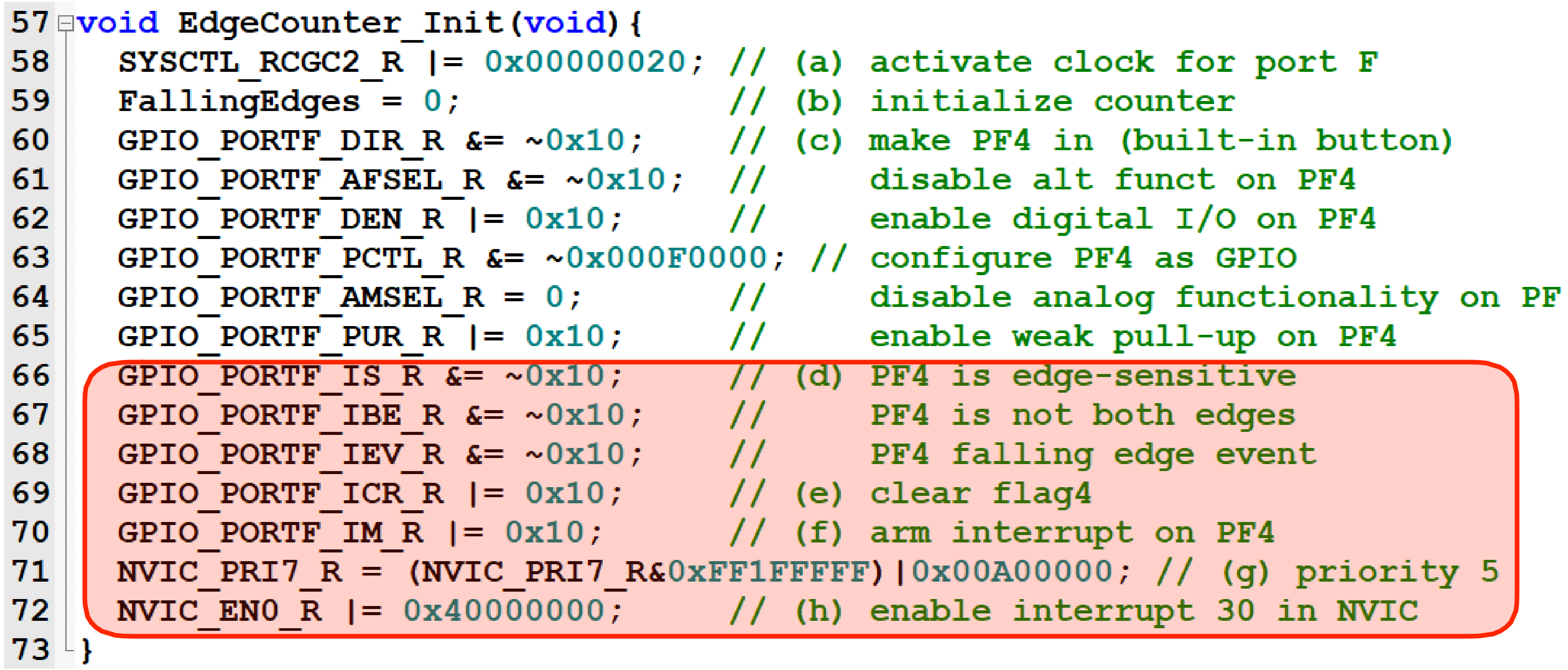
1010 0000

A **0**

0 0 **A** **0** 0 0 0
0

NVIC_PRI7_R = (NVIC_PRI7_R&0xFF1FFFF) | 0x00A00000;

EdgeInterrupt.c: EdgeCounter_Init()



```c
void EdgeCounter_Init(void){
  SYSCTL_RCGC2_R |= 0x00000020; // (a) activate clock for port F
  FallingEdges = 0;             // (b) initialize counter
  GPIO_PORTF_DIR_R &= ~0x10;    // (c) make PF4 in (built-in button)
  GPIO_PORTF_AFSEL_R &= ~0x10;  //     disable alt funct on PF4
  GPIO_PORTF_DEN_R |= 0x10;     //     enable digital I/O on PF4
  GPIO_PORTF_PCTL_R &= ~0x000F0000; // configure PF4 as GPIO
  GPIO_PORTF_AMSEL_R = 0;       //     disable analog functionality on PF
  GPIO_PORTF_PUR_R |= 0x10;     //     enable weak pull-up on PF4
  GPIO_PORTF_IS_R &= ~0x10;     // (d) PF4 is edge-sensitive
  GPIO_PORTF_IBE_R &= ~0x10;    //     PF4 is not both edges
  GPIO_PORTF_IEV_R &= ~0x10;    //     PF4 falling edge event
  GPIO_PORTF_ICR_R |= 0x10;     // (e) clear flag4
  GPIO_PORTF_IM_R |= 0x10;      // (f) arm interrupt on PF4
  NVIC_PRI7_R = (NVIC_PRI7_R&0xFF1FFFFF)|0x00A00000; // (g) priority 5
  NVIC_EN0_R |= 0x40000000;     // (h) enable interrupt 30 in NVIC
}
```

Interrupt Service Routine (ISR)

- Things you must do in every interrupt service routine
  - Acknowledge
    - clear flag that requested the interrupt
    - SysTick is exception; automatic acknowledge
  - Communicate via shared global variables

EdgeInterrupt.c: ISR for Port F Edge-triggered Interrupt

```
void GPIOPortF_Handler(void){
    GPIO_PORTF_ICR_R = 0x10;        // acknowledge flag4
FallingEdges = FallingEdges + 1;
}
```

SysTick Periodic Interrupt

SysTick Periodic Interrupt

- The SysTick timer is a simple way to create a periodic interrupt that is requested on a fixed time basis.
- When the SysTick counter goes from 1 to 0, the **Count** flag in the **NVIC_ST_CTRL_R** register is set, triggering an interrupt.See datasheet P.139.
- On the next clock, the **CURRENT** is loaded with the **RELOAD** value.
- If the **RELOAD** value is *n*, then it rolls over every *n*+1 counts.
- Notice there is **no explicit software step** in the ISR to clear the **COUNT** flag.

Setup SysTick Registers

In Main:

NVIC_ST_CTRL_R = 0;   // disable SysTick during setup
NVIC_ST_RELOAD_R = period-1;// reload value
NVIC_ST_CURRENT_R = 0;   // any write to current clears it
NVIC_SYS_PRI3_R=(NVIC_SYS_PRI3_R&0x1FFFFFFF)|
0x40000000;//priority 2
NVIC_ST_CTRL_R = 0x07; // enable SysTick with core clock & interrupts
EnableInterrupts();// enable interrupts after all initialization is finished

In ISR: No need to acknowledge. SysTick is the only interrupt on the TM4C that has an automatic acknowledge.

Address	31-24	23-17	16	15-3	2	1	0	Name
$E000E010	0	0	COUNT	0	CLK_SRC	INTEN	ENABLE	NVIC_ST_CTRL_R
$E000E014	0	24-bit RELOAD value						NVIC_ST_RELOAD_R
$E000E018	0	24-bit CURRENT value of SysTick counter						NVIC_ST_CURRENT_R

Address	31-29	28-24	23-21	20-8	7-5	4-0	Name
$E000ED20	TICK	0	PENDSV	0	DEBUG	0	NVIC_SYS_PRI3_R

** We must set **CLK_SRC**=1, because **CLK_SRC**=0 external clock mode is not implemented on the TM4C family.

C12_PeriodicSysTickInts

```
volatile unsigned long Counts = 0;
void SysTick_Init(unsigned long period){
     NVIC_ST_CTRL_R = 0;               // disable SysTick during
     setup  NVIC_ST_RELOAD_R = period-1;// reload value
     NVIC_ST_CURRENT_R = 0;            // any write to current
     clears it
     // priority 2
    NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|
    0x40000000;
     // enable SysTick with core clock and interrupts
      NVIC_ST_CTRL_R = 0x07;
     EnableInterrupts();
}
// Interrupt service routine: Executed every 62.5ns*(period)
void SysTick_Handler(void){
     GPIO_PORTF_DATA_R ^= 0x04;            // toggle
     PF2  Counts = Counts + 1;
}
```

Interrupt Service Routine (ISR)

- Things you must do in every interrupt service routine
  - Acknowledge
    - clear flag that requested the interrupt
    - SysTick is exception; automatic acknowledge
  - Communicate via shared global variables

EdgeInterrupt.c: ISR for Port F Edge-triggered Interrupt

```
void GPIOPortF_Handler(void){
   GPIO_PORTF_ICR_R = 0x10;      // acknowledge flag4
FallingEdges = FallingEdges + 1;
}
```

C12_PeriodicSysTickInts

```
volatile unsigned long Counts = 0;
void SysTick_Init(unsigned long period){
    NVIC_ST_CTRL_R = 0;                  // disable SysTick during
    setup  NVIC_ST_RELOAD_R = period-1;// reload value
    NVIC_ST_CURRENT_R = 0;               // any write to current
    clears it
    // priority 2
   NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|
   0x40000000;
    // enable SysTick with core clock and interrupts
     NVIC_ST_CTRL_R = 0x07;
    EnableInterrupts();
}
// Interrupt service routine: Executed every 62.5ns*(period)
void SysTick_Handler(void){
    GPIO_PORTF_DATA_R ^= 0x04;              // toggle
    PF2  Counts = Counts + 1;
}
```

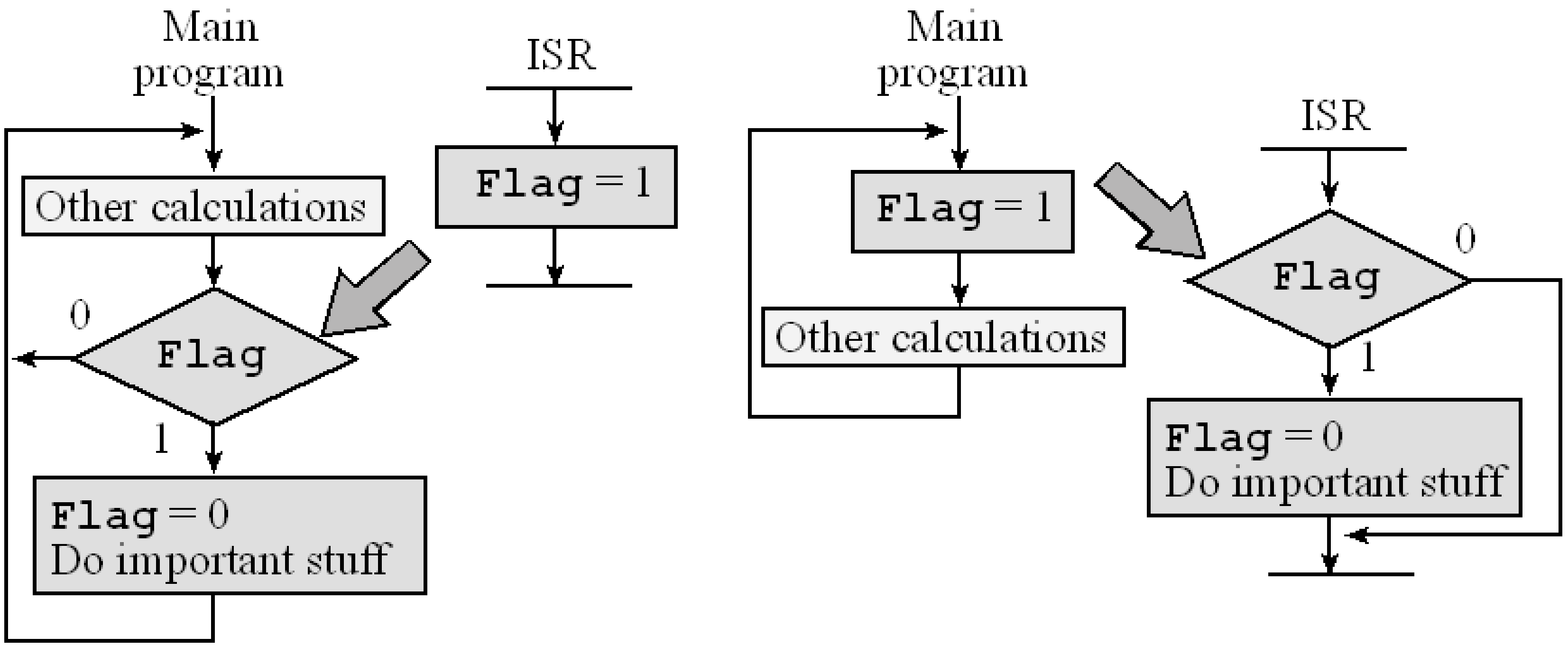
ISR: To Do & Not To Do

- Do:
  - Acknowledge: software clear the interrupt flag
  - Make sure it is **Short**
- Not to Do:
  - Delay loops, long nested if-else, long switch
  - ISR execution longer than time between triggers

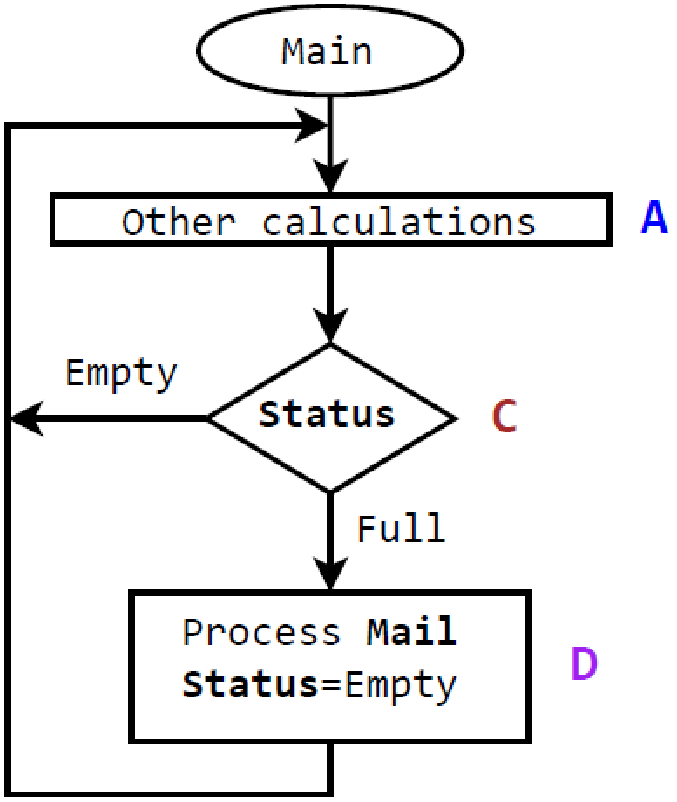
Inter-Thread Communication

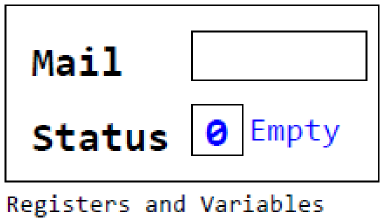
- Main thread vs. background thread
- Three ways to communicate
  - Flag
  - Mailbox
  - Buffers, such as FIFO

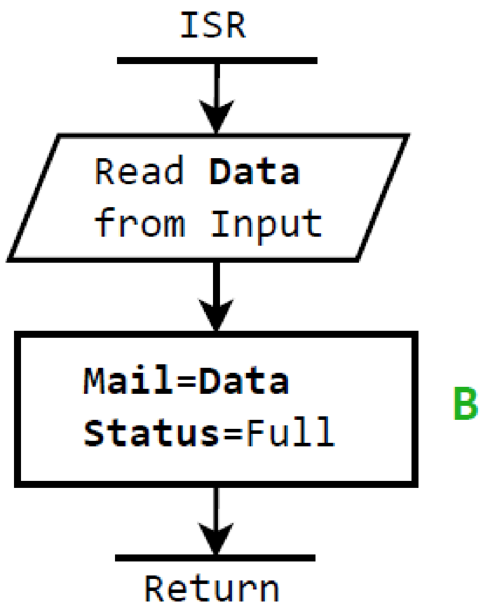
Inter-Thread Communication
Flag



Inter-Thread Communication
Mailbox







References

- ❑ Textbook Chapter 9: 9.1 – 9.6
- ❑ Example Projects:

  EdgeInterrupt

  PeriodicSysTickInts