

Interrupts & Exception

TM4C123GH6PM Launchpad

Dr. Munesh Singh

Indian Institute of Information Technology Design and Manufacturing
Kancheepuram, Chennai, Tamil Nadu

March 2, 2020



Interrupt vs Polling

- Generally there are two methods by which a peripheral device can receive a service from micro controller
 - Polling
 - Interrupt
- **Polling:** In polling method a micro controller continuously monitors the status of a device.
- **Example:** UART code you can see that we are continuously monitoring the bit 5 of UART0_FR_R register
- **Interrupt:** Interrupt the same thing is serviced by a microcontroller without continuously monitoring the status.

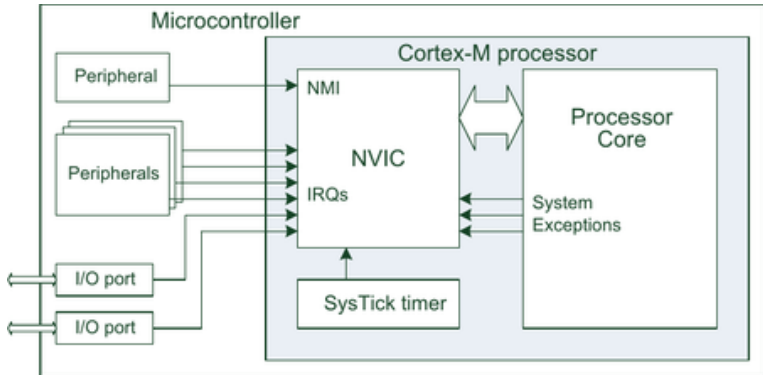


Interrupt service routine (ISR)

- For every interrupt there must be a program associated with it.
- When an interrupt occurs this program is executed to perform certain service for the interrupt.
- This program is commonly referred to as an interrupt service routine (ISR) or interrupt handler.
- When an interrupt occurs, the CPU runs the interrupt service routine.
- Now the question is, how the ISR gets executed?
- In the ARM CPU there are pins that are associated with hardware interrupts.



- They are input signals into the CPU. When the signals are triggered, CPU pushes the PC register onto the stack and loads the PC register with the address of the interrupt service routine.



Interrupt Vector Table

- Since there is a program (ISR) associated with every interrupt and this program resides in memory (RAM or ROM).
- There must be a look-up table to hold the addresses of these ISRs.
- This look-up table is called interrupt vector table.
- In the ARM, the **lowest 1024 bytes** ($256 * 4 = 1024$) of memory space are set aside for the interrupt vector table and must not be used for any other function
- Of the 256 interrupts, some are used for software interrupts and some are for hardware IRQ interrupts.



Interrupt and Exception assignments in ARM Cortex-M

- Interrupts on the Cortex-M are controlled by the **Nested Vectored Interrupt Controller (NVIC)**.
- Each exception has an associated 32-bit vector that points to the memory location where the ISR that handles the exception is located.
- The NVIC of the ARM Cortex-M has room for the **total of 255 interrupts and exceptions**.
- The interrupt numbers are also referred to as INT type (**or INT #**) in which the type can be from **1 to 255** or **0x01 to 0xFF**.
- The NVIC in ARM Cortex-M assigns the **first 15 interrupts for internal use**.
- The memory locations **0 to 3** are used to store the value to be loaded into the **stack pointer** when the device is coming out of reset



Interrupt #	Interrupt	Memory Location	Priority Level
0	Stack Pointer Initial Value	0x00000000	
1	Reset	0x00000004	-3 Highest
2	NMI	0x00000008	-2
3	Hard Fault	0x0000000C	-1
4	Memory Management Fault	0x00000010	Programmable
5	Bus Fault	0x00000014	Programmable
6	Usage Fault (undefined instructions, divide by zero, unaligned memory access,)	0x00000018	Programmable
7	Reserved	0x0000001C	Programmable
8	Reserved	0x00000020	Programmable
9	Reserved	0x00000024	Programmable
10	Reserved	0x00000028	Programmable
11	SVCall	0x0000002C	Programmable
12	Debug Monitor	0x00000030	Programmable
13	Reserved	0x00000034	Programmable
14	PendSV	0x00000038	Programmable
15	SysTick	0x0000003C	Programmable
16	IRQ for peripherals	0x00000040	Programmable
17	IRQ for peripherals	0x00000044	Programmable
...
255	IRQ for peripherals	0x000003FC	Programmable

Interrupt Vector Table for ARM Cortex-M.



- An ISR can be launched as a result of an event at the peripheral devices such as **timer timeout or analog-to-digital converter (ADC) conversion complete**.
- The largest number of the interrupts in the ARM Cortex-M belong to above type category.
- that ARM Cortex-M NVIC has set aside the **first 15 interrupts (INT 1 to INT 15)** for internal use and exceptions
- The **Reset, NMI, undefined instructions**, and so on are part of this group of exceptions.
- Many of the **INT 16 to INT 255** are used by the chip manufacturer to be assigned to **various peripherals such as timers, ADC, Serial COM, external hardware interrupts**, and so on
- Each peripheral device has a group of special function registers that must be used to access the device for configuration



Interrupt Priority for ARM Cortex-M

- All exceptions and interrupts in the Cortex-M4 system have certain priority levels, either **maskable (software)** or **unmaskable sources (Hardware)**.
- Most **maskable interrupts** have **programmable priority levels**, but all **Non-Maskable Interrupts (NMIs)** have **fixed priority levels**.
- The **NVIC** performs a comparison between the priority level of current exception or interrupt and the priority level of the new coming exception/interrupt.
- In the ARM Cortex-M4 system, the interrupt priority levels are controlled by the **Interrupt Priority Registers**.
- Each priority register can use **3 bits, 4 bits, or 8 bits** to cover all priority levels used in the priority control system.



- Devices within the Tiva family support up to **154 interrupt sources and 8 priority levels**, which means that 3 bits are used in the priority register in the TM4C123GH6PM MCU.
- To activate an interrupt source we need to set its priority and enable that source in the NVIC.
- lists some of the interrupt sources available on the TM4C family of micro controllers.
 - Interrupt numbers **0 to 15** contain the **faults**
 - **Software interrupt and SysTick**; these interrupts will be handled differently from interrupts 16 and up



Vector address	Vector (Exception) Number	Interrupt # (IRQ)	Address (Priority Register)	Interrupt Source	Priority Register (Tivaware Name)	Priority Bits
0x00000038	14	-2	0xE000.ED20	PendSV	NVIC_SYS_PRI3_R	23 - 21
0x0000003C	15	-1	0xE000.ED20	SysTick	NVIC_SYS_PRI3_R	31 - 29
0x00000040	16	0	0xE000.E400	GPIO Port A	NVIC_PRI0_R	7 - 5
0x00000044	17	1	0xE000.E400	GPIO Port B	NVIC_PRI0_R	15 - 13
0x00000048	18	2	0xE000.E400	GPIO Port C	NVIC_PRI0_R	23 - 21
0x0000004C	19	3	0xE000.E400	GPIO Port D	NVIC_PRI0_R	31 - 29
0x00000050	20	4	0xE000.E404	GPIO Port E	NVIC_PRI1_R	7 - 5
0x00000054	21	5	0xE000.E404	UART0, Rx Tx	NVIC_PRI1_R	15 - 13
0x00000058	22	6	0xE000.E404	UART1, Rx Tx	NVIC_PRI1_R	23 - 21
0x0000005C	23	7	0xE000.E404	SSI0, Rx Tx	NVIC_PRI1_R	31 - 29
0x00000060	24	8	0xE000.E408	I2C0	NVIC_PRI2_R	7 - 5
0x00000064	25	9	0xE000.E408	PWM Fault	NVIC_PRI2_R	15 - 13
0x00000068	26	10	0xE000.E408	PWM Gen 0	NVIC_PRI2_R	23 - 21
0x0000006C	27	11	0xE000.E408	PWM Gen 1	NVIC_PRI2_R	31 - 29
0x00000070	28	12	0xE000.E40C	PWM0 Gen 2	NVIC_PRI3_R	7 - 5
0x00000074	29	13	0xE000.E40C	Quad Encoder 0	NVIC_PRI3_R	15 - 13
0x00000078	30	14	0xE000.E40C	ADC Seq 0	NVIC_PRI3_R	23 - 21
0x0000007C	29	15	0xE000.E40C	ADC Seq 1	NVIC_PRI3_R	31 - 29
0x00000080	32	16	0xE000.E410	ADC Seq 2	NVIC_PRI4_R	7 - 5
0x00000084	33	17	0xE000.E410	ADC Seq 3	NVIC_PRI4_R	15 - 13
0x00000088	34	18	0xE000.E400	Watchdog	NVIC_PRI4_R	23 - 21
0x0000008C	35	19	0xE000.E410	Timer 0A	NVIC_PRI4_R	31 - 29
0x00000090	36	20	0xE000.E414	Timer 0B	NVIC_PRI5_R	7 - 5
0x00000094	37	21	0xE000.E414	Timer 1A	NVIC_PRI5_R	15 - 13
0x00000098	38	22	0xE000.E414	Timer 1B	NVIC_PRI5_R	23 - 21
0x0000009C	39	23	0xE000.E414	Timer 2A	NVIC_PRI5_R	31 - 29
0x000000A0	40	24	0xE000.E418	Timer 2B	NVIC_PRI6_R	7 - 5
0x000000A4	41	25	0xE000.E418	Comp 0	NVIC_PRI6_R	15 - 13
0x000000A8	42	26	0xE000.E418	Comp 1	NVIC_PRI6_R	23 - 21



0x000000A8	42	26	0xE000.E418	Comp 1	NVIC_PRI6_R	23 - 21
0x000000AC	43	27	0xE000.E418	Comp 2	NVIC_PRI6_R	31 - 29
0x000000B0	44	28	0xE000.E41C	System Control	NVIC_PRI7_R	7 - 5
0x000000B4	45	29	0xE000.E41C	Flash Control	NVIC_PRI7_R	15 - 13
0x000000B8	46	30	0xE000.E41C	GPIO Port F	NVIC_PRI7_R	23 - 21
0x000000BC	47	31	0xE000.E41C	GPIO Port G	NVIC_PRI7_R	31 - 29
0x000000C0	48	32	0xE000.E420	GPIO Port H	NVIC_PRI8_R	7 - 5
0x000000C4	49	33	0xE000.E420	UART2, Rx Tx	NVIC_PRI8_R	15 - 13
0x000000C8	50	34	0xE000.E420	SSI1, Rx Tx	NVIC_PRI8_R	23 - 21
0x000000CC	51	35	0xE000.E420	Timer 3A	NVIC_PRI8_R	31 - 29
0x000000D0	52	36	0xE000.E424	Timer 3B	NVIC_PRI9_R	7 - 5
0x000000D4	53	37	0xE000.E424	I2C1	NVIC_PRI9_R	15 - 13
0x000000D8	54	38	0xE000.E424	Quad Encoder 1	NVIC_PRI9_R	23 - 21
0x000000DC	55	39	0xE000.E424	CAN0	NVIC_PRI9_R	31 - 29
0x000000E0	56	40	0xE000.E428	CAN1	NVIC_PRI10_R	7 - 5
0x000000E4	57	41	0xE000.E428	CAN2	NVIC_PRI10_R	15 - 13
0x000000E8	58	42	0xE000.E428	Ethernet	NVIC_PRI10_R	23 - 21
0x000000EC	59	43	0xE000.E428	Hibernate	NVIC_PRI10_R	31 - 29
0x000000F0	60	44	0xE000.E42C	USB0	NVIC_PRI11_R	7 - 5
0x000000F4	61	45	0xE000.E42C	PWM Gen 3	NVIC_PRI11_R	15 - 13
0x000000F8	62	46	0xE000.E42C	uDMA Soft Tfr	NVIC_PRI11_R	23 - 21
0x000000FC	63	47	0xE000.E42C	uDMA Error	NVIC_PRI11_R	31 - 29

The relationship between vectors and NVIC definitions.



- The following five conditions must be true for an interrupt to be generated:
- 1 **Device arm:** Each potential interrupt trigger has a separate arm bit that the software can activate or deactivate.
 - The software will set the arm bits for those devices from which it wishes to accept interrupts
 - deactivate the arm bits within those devices from which interrupts are not to be allowed.
 - 2 **NVIC enable:** For most devices there is a enable bit in the NVIC that must be set (periodic SysTick interrupts are an exception, having no NVIC enable).
 - 3 **Global enable:** Bit 0 of the special register **PRIMASK** is the interrupt mask bit, I
 - If this bit is 1 most interrupts and exceptions are not allowed, which we will define as disabled
 - If the bit is 0, then interrupts are allowed, which we will define as enabled.



① Interrupt priority level must be higher than current level executing:

- The BASEPRI register prevents interrupts with lower priority interrupts, but allows higher priority interrupts.
- For example if the software sets the **BASEPRI to 3**, then requests with level 0, 1, and 2 can interrupt, while requests at levels 3 and higher will be postponed.
- The software can also specify the priority level of each interrupt request.
- If **BASEPRI is zero**, then the priority feature is disabled and all interrupts are allowed.

② Hardware event trigger:

- Hardware triggers are bits in the GPIO_PORTx_RIS_R register that are set on rising or falling edges of digital input pins.

For an interrupt to occur, these five conditions must be simultaneously true but can occur in any order.



GPIO Port Interrupt Programming

- Interrupt numbers **16 to 255** are assigned to the peripherals.
- The **INT (IRQ) 46** is assigned to the **GPIO Port of F**.
- Although **PortF has 8 pins**, we have only one interrupt assigned to the entire PortF.
- In other words, when any of the PortF pins trigger an interrupt, they all go to the same address location in the interrupt vector table.
- It is the job of our **Interrupt Service Routine (ISR or interrupt Handler)** to find out which pin caused the interrupt.



Interrupt trigger point

- When an input pin is connected to an external device to be used for interrupt, we have 5 choices for trigger point
 - ① low-level trigger (active Low level),
 - ② high-level trigger (active High level),
 - ③ rising-edge trigger (positive-edge going from Low to High),
 - ④ falling-Edge trigger (negative-edge going from High to Low),
 - ⑤ Both edge (rising and falling) trigger.
- Upon Reset, all the interrupts are disabled. To enable any interrupt:
 - Enable the interrupt for a specific peripheral module.
 - Enable the interrupts at the NVIC module.
 - Enable the interrupt globally



GPIO Interrupt Control Registers

GPIO Register	Tiware Name	Each Bit Value (Lowest 8-Bit) and Each Pin Function
GPIOIS	GPIO_PORTx_IS_R	<u>Interrupt sense register</u> Determines level or edge triggered 0: Detect an edge (edge-sensitive) on the pin, 1: Detect a level (level-sensitive) on the pin.
GPIOIBE	GPIO_PORTx_IBE_R	0: Interrupt is controlled by GPIOIEV, 1: Both edges on the corresponding pin trigger an interrupt
GPIOIEV	GPIO_PORTx_IEV_R	<u>GPIO Interrupt Event Register</u> Determines the detecting edges or levels. 0: A falling edge or a LOW level, 1: A rising edge or a HIGH level triggers an interrupt
GPIOIM	GPIO_PORTx_IM_R	<u>GPIO Interrupt Mask Register</u> Masks (disables) or unmask (enable) an interrupt. 0: Interrupt is masked (disabled), 1: Interrupt is unmasked (enabled).
GPIORIS	GPIO_PORTx_IS_R	<u>GPIO Raw Interrupt Status Register</u> Indicates the raw interrupt status for a pin. 0: No interrupt occurred on the pin, 1: An interrupt is occurred on the pin. For the edge-triggered interrupts, write a 1 to the pin to clear that interrupt. For level-triggered interrupt, no action is needed.
GPIOMIS	GPIO_PORTx_MIS_R	<u>GPIO Masked Interrupt Status Register</u> Indicates the state of the interrupt. 0: No interrupt occurred or the pin has been masked, 1: An interrupt has been occurred.
GPIOICR	GPIO_PORTx_ICR_R	<u>GPIO Interrupt Clear Register</u> Clears an edge-triggered interrupt. 0: No action, 1: The corresponded edge-triggered interrupt is cleared.

Table: The bit values and functions for GPIO interrupt controls.



- All of these registers are 32-bit, but only lowest 8 bits are used.
- Each bit corresponds to each pin in the selected GPIO Port: bit 0 is for pin 0, bit 1 is for pin 1, and so on.
- The above table shows the bit values and their functions for these registers
- Before any exception or interrupt can be applied to any pin on any GPIO Port
- All GPIO pins on selected GPIO Port should be initialized and configured via related GPIO registers.



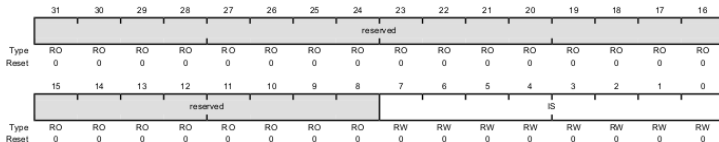
GPIO Interrupt Control Registers Description

- GPIO Interrupt Sense (GPIOIS) register to decide the level or edge.
- GPIOIS register we need to indicate which level or edge
- GPIO Interrupt Event (GPIOIEV) to decide low-level, high-level, falling, or rising-edge.

GPIO Interrupt Sense (GPIOIS)

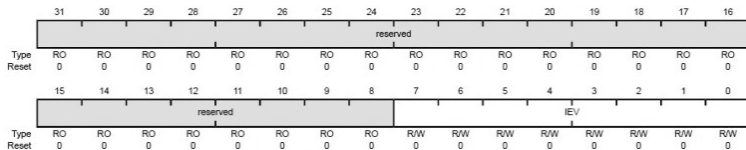
GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
GPIO Port F (APB) base: 0x4002.5000
GPIO Port F (AHB) base: 0x4005.D000
Offset 0x404
Type RW, reset 0x0000.0000

Bit	Bit Name	Description
7-0	IS	<u>GPIO Interrupt Sense</u> 0: The edge on the corresponding pin is detected (edge-sensitive). 1: The level on the corresponding pin is detected (level-sensitive).



GPIO Interrupt Control Registers Description

- GPIO Interrupt Event Register (GPIOIEV)
- Since we need a rising edge triggered interrupt at PF4, so writing 1 to the respective bit field will do the same for us.

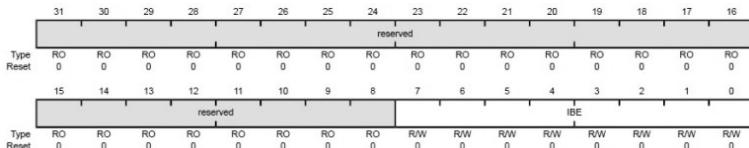


GPIOIS	GPIOIEV	
(interrupt sense)	(Interrupt Event)	
0	0	Falling edge
0	1	Rising edge
1	0	Low level
1	1	High level



GPIO Interrupt Control Registers Description

- **GPIO Interrupt Both Edge (GPIOIBE)**
- Setting the corresponding bit field enables the interrupts for both edges i.e. rising edge and falling edge
- PF4 to 0 will allow us to use this register in conjunction with the GPIOIEV register.



Bit Bit Name

Description

GPIO Interrupt Both Edges

7-0 IBE

0: Interrupt generation is controlled by the GPIO Interrupt Event (GPIOIEV) register
1: Both edges on the corresponding pin trigger an interrupt



GPIO Interrupt Control Registers Description

● GPIO Interrupt Clear Register (GPIOICR)

- The corresponding bit fields in this register clears the interrupt for the respective pin.
- It is critical that the interrupt handler clears the interrupt flag before returning from interrupt handler.
- Otherwise the interrupt appears as if it is still pending and the interrupt handler will be executed again and again forever and the program hangs.

GPIO Interrupt Clear (GPIOICR)

GPIO Port A (APB) base: 0x4000.4000

GPIO Port A (AHB) base: 0x4005.8000

GPIO Port B (APB) base: 0x4000.5000

GPIO Port B (AHB) base: 0x4005.9000

GPIO Port C (APB) base: 0x4000.6000

GPIO Port C (AHB) base: 0x4005.A000

GPIO Port D (APB) base: 0x4000.7000

GPIO Port D (AHB) base: 0x4005.B000

GPIO Port E (APB) base: 0x4002.4000

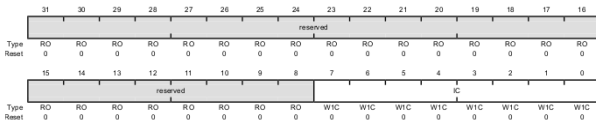
GPIO Port E (AHB) base: 0x4002.C000

GPIO Port F (APB) base: 0x4002.5000

GPIO Port F (AHB) base: 0x4005.D000

Offset 0x41C

Type W1C, reset 0x0000.0000



GPIO Interrupt Control Registers Description

- **GPIO Interrupt Mask Register (GPIOIM)**
- We need to enable the interrupt capability of a given peripheral at the module level.
- The lower 8 bits of this register is used to enable the interrupt capability of each pin of the I/O port
- To enable the interrupts for PF0 and PF4 pins, we will need the following:

```
GPIO_PORTF_IM_R |= 0x11; /* unmask interrupt */
```

GPIO Interrupt Mask (GPIOIM)

GPIO Port A (APB) base: 0x4000.4000

GPIO Port A (AHB) base: 0x4005.8000

GPIO Port B (APB) base: 0x4000.5000

GPIO Port B (AHB) base: 0x4005.9000

GPIO Port C (APB) base: 0x4000.6000

GPIO Port C (AHB) base: 0x4005.A000

GPIO Port D (APB) base: 0x4000.7000

GPIO Port D (AHB) base: 0x4005.B000

GPIO Port E (APB) base: 0x4002.4000

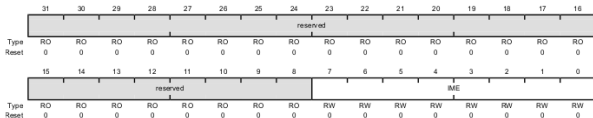
GPIO Port E (AHB) base: 0x4005.C000

GPIO Port F (APB) base: 0x4002.5000

GPIO Port F (AHB) base: 0x4005.D000

Offset 0x10

Type RW, reset 0x0000.0000



GPIO Interrupt Control Registers Description

- **GPIO Raw Interrupt Status (GPIORIS)**
- For edge-detect interrupts, this bit is cleared by writing a 1 to the corresponding bit in the GPIOICR register.
- For a GPIO level-detect interrupt, the bit is cleared when the level is deasserted.

GPIO Raw Interrupt Status (GPIORIS)

GPIO Port A (APB) base: 0x4000.4000

GPIO Port A (AHB) base: 0x4005.8000

GPIO Port B (APB) base: 0x4000.5000

GPIO Port B (AHB) base: 0x4005.9000

GPIO Port C (APB) base: 0x4000.6000

GPIO Port C (AHB) base: 0x4005.A000

GPIO Port D (APB) base: 0x4000.7000

GPIO Port D (AHB) base: 0x4005.B000

GPIO Port E (APB) base: 0x4002.4000

GPIO Port E (AHB) base: 0x4005.C000

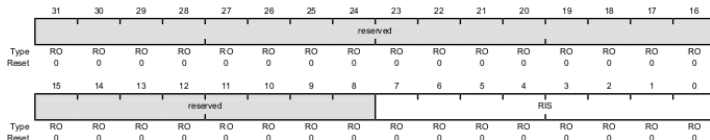
GPIO Port F (APB) base: 0x4002.5000

GPIO Port F (AHB) base: 0x4005.D000

Offset 0x14

Type RO, reset 0x0000.0000

Bit	Bit Name	Description
7-0	RIS	<u>GPIO Interrupt Raw Status</u> 0: An interrupt condition has not occurred on the corresponding pin. 1: An interrupt condition has occurred on the corresponding pin.



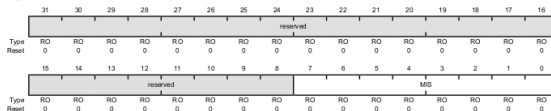
GPIO Interrupt Control Registers Description

- **GPIO Masked Interrupt Status (GPIOMIS)**
- For edge-detect interrupts, this bit is cleared by writing a 1 to the corresponding bit in the GPIOICR register.
- For a GPIO level-detect interrupt, the bit is cleared when the level is deasserted.

GPIO Masked Interrupt Status (GPIOMIS)

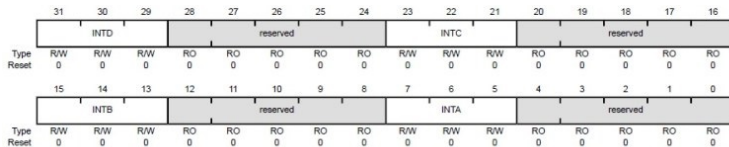
GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4000.8000
GPIO Port E (AHB) base: 0x4005.C000
GPIO Port F (APB) base: 0x4002.5000
GPIO Port F (AHB) base: 0x4005.D000
Offset 0x4 18
Type RO, reset 0x0000.0000

Bit	Bit Name	Description
7-0	MIS	<u>GPIO Masked Interrupt Status</u> 0: An interrupt condition on the corresponding pin is masked or has not occurred. 1: An interrupt condition on the corresponding pin has triggered an interrupt to the interrupt controller.



GPIO Interrupt Control Registers Description

• NVIC Interrupt Priority Register



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 0	PWM0 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

Table 7.6: Most popular priority registers used in the TM4C123GH6PM NVIC.

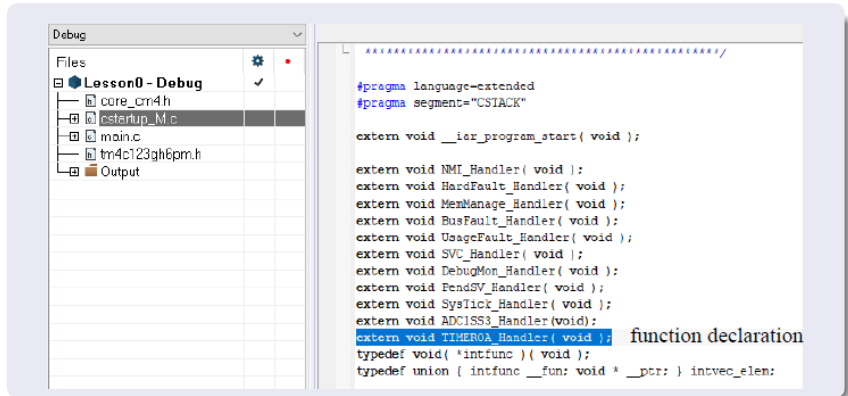


Interrupt Vector Table Entry in cstartup_M

- Initialization steps for interrupt:
 - In **cstartup_M** file we add the declaration of interrupt function
 - Add the function name in the interrupt table with correct order
 - Order of the interrupt get in the interrupt vector table
 - Lastly, function definition.



Function Addition in cstartup M.c vector file



Debug

Files

Lesson0 - Debug

- core_cm4.h
- cstartup_M.c
- main.c
- tm4c123gh6pm.h
- Output

```
/////////////////////////////////////////////////////////////////

#pragma language-extended
#pragma segment="CSTACK"

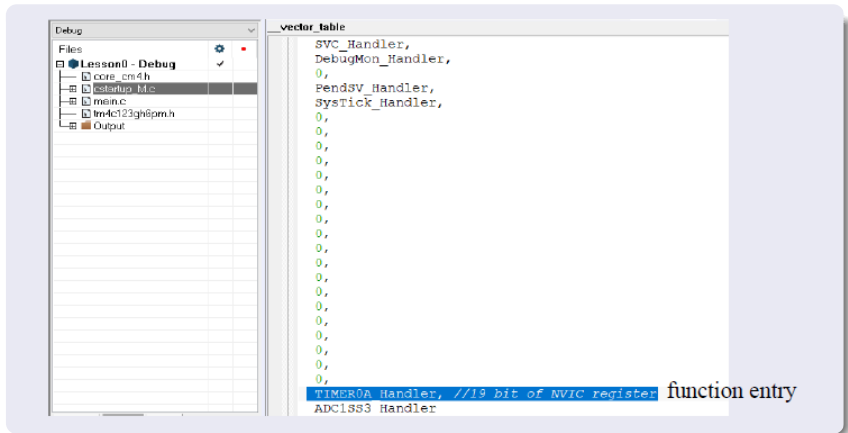
extern void __iar_program_start( void );

extern void NMI_Handler( void );
extern void HardFault_Handler( void );
extern void MemManage_Handler( void );
extern void BusFault_Handler( void );
extern void UsageFault_Handler( void );
extern void SVC_Handler( void );
extern void DebugMon_Handler( void );
extern void PendSV_Handler( void );
extern void SysTick_Handler( void );
extern void ADC1SS3_Handler(void);
extern void TIMERA0_Handler( void );
typedef void( *intfunc )( void );
typedef union { intfunc __fun; void * __ptr; } intvec_elem;
```

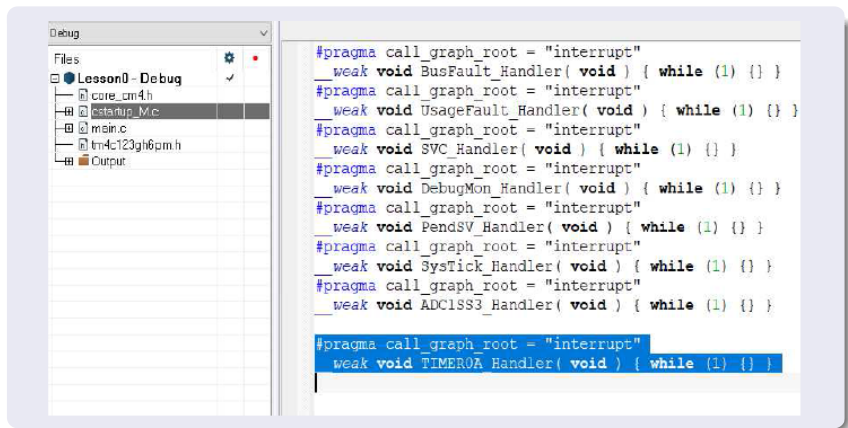
function declaration



Function Addition in cstartup M.c vector file



Function Addition in cstartup M.c vector file



Debug

Files

- Lesson0 - Debug
 - core_cm4.h
 - cstartup_M.c
 - main.c
 - tm4c123gh6pm.h
 - Output

```
#pragma call_graph_root = "interrupt"
__weak void BusFault_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void UsageFault_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void SVC_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void DebugMon_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void PendSV_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void SysTick_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void ADC1SS3_Handler( void ) { while (1) {} }

#pragma call_graph_root = "interrupt"
__weak void TIMER0A_Handler( void ) { while (1) {} }
```



Timing Interrupt for LED Control

```
#include "tm4c123gh6pm.h"
#define CG *((volatile unsigned int *)0x400FE608) // clock gating
#define DIR *((volatile unsigned int *)0x40025400) // direction register (b011110)->portF
#define DEN *((volatile unsigned int *)0x4002551C) //digital enable (b11111)->portF
#define DATAF *((unsigned int *)0x400253FC) //Data register of port F
#define DATABITF ((volatile unsigned int *)0x40025000) //Data register of port F

void TIMER0A_Handler( void )
{
    TIMER0_ICR_R |= (1<<0);
    DATAF ^= (1<<2);
}

int main() {
    CG=0x20;
    SYSCCTL_RCGCTIMER_R=(1U<<0); //clock gating on timer circuit
    TIMER0_CTL_R &= ~(1<<0); // disable the timer0
    TIMER0_CFG_R= 0x00000000; // configure resiter for periodic
    TIMER0_TAMR_R=(0x2<<0); // periodic mode timer
    TIMER0_TAMR_R&= ~(1<<4); // configure down counter
    TIMER0_TAILR_R= 0x00F42400; // set the value 16000000
    TIMER0_IMR_R=(1<<0); // enable the interrupt for timer0
    NVIC_EN0_R |= (1<<19); // Enter the entry in NVIC Register
    TIMER0_CTL_R |= (1<<0); // enable the timer0

    DIR=((1U<<1)|(1U<<2)|(1U<<3)) // Set the direction of GPIOF Port
    DEN=((1U<<1)|(1U<<2)|(1U<<3)) //Enable Digital Function of GPIOF Port
    while(1) {
    }
    return 0;
}
```



Analog Control the LED Using Potentiometer

```
#include "lm4f120h5qr.h"
volatile static uint32_t adcResult = 0;
void ADC1SS3_Handler(void){
    adcResult = ADC1->SSFIFO3;
    ADC1->ISC = (1<<3); }
int main()
{
    SYSCTL->RCGCADC = (1<<1); //Step 1
    SYSCTL->RCGCGPIO = (1<<4)|(1<<5); //Step 2
    GPIOE->DIR &= ~(1<<1);
    GPIOF->DEN = 0xFF; //Enable digital functions for the corresponding pin
    GPIOF->AFSEL = 0x00; //Disable alternate functions, we are using digital
    GPIOF->DIR = 0xFF; //Setting the pins makes them an output
    GPIOF->DATA = (1<<1);
    GPIOE->AFSEL = (1<<1); //Step 3
    GPIOE->DEN &= ~(1<<1); //Step 4
    GPIOE->AMSEL = (1<<1); //Step 5
    ADC1->ACTSS &= ~(1<<3); //Step 1 sample sequenser setting
    ADC1->EMUX = (0xF<<12); //Step 2
    ADC1->SSMUX3 = 2; //Step 3
    ADC1->SSCTL3 = 0x6; //Step 4
    ADC1->IM = (1<<3); //Step 5
    ADC1->ACTSS |= (1<<3); //Step 6
    ADC1->ISC = (1<<3);
    NVIC_EnableIRQ(ADC1SS3_IRQn);
    while(1) {
        if(adcResult > 2047){
            GPIOF->DATA |= (1<<1);}
```


Function Addition in cstartup M.c vector file

`_vector_table`

`0,
TIMER0A_Handler,`

`extern void TIMER0A_Handler(void);`

`extern void ADC1SS3_Handler(void);`

`#pragma call_graph_root = "interrupt"`

`_weak void ADC1SS3_Handler(void) { while (1) {} }`

`0, //50
ADC1SS3_Handler`



Thank You

