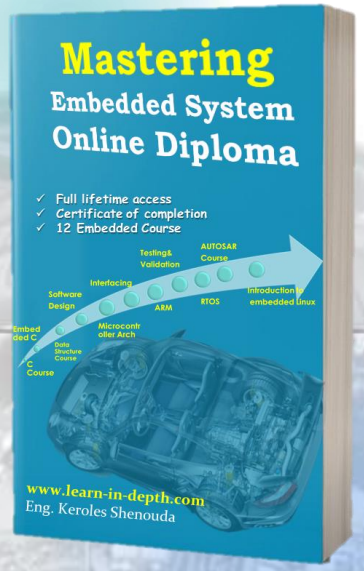
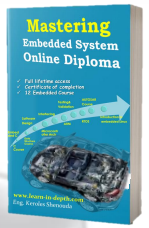
Mastering Embedded System

Online Diploma

- ✓ Full lifetime access
- ✓ Access on Android mobile and PC (Windows)
- ✓ Certificate of completion
- ✓ 12 Embedded Course



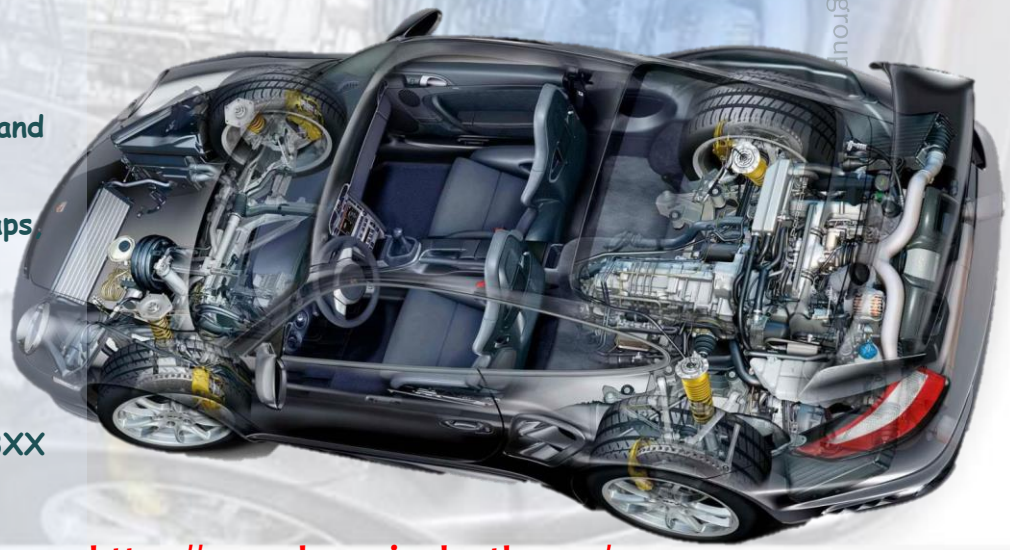


#LEARN_IN DEPTH

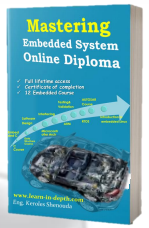
#Be_professional_in_embedded_system

Unit 6 (MCU Fundamentals) . lesson 4 MCU Interrupts

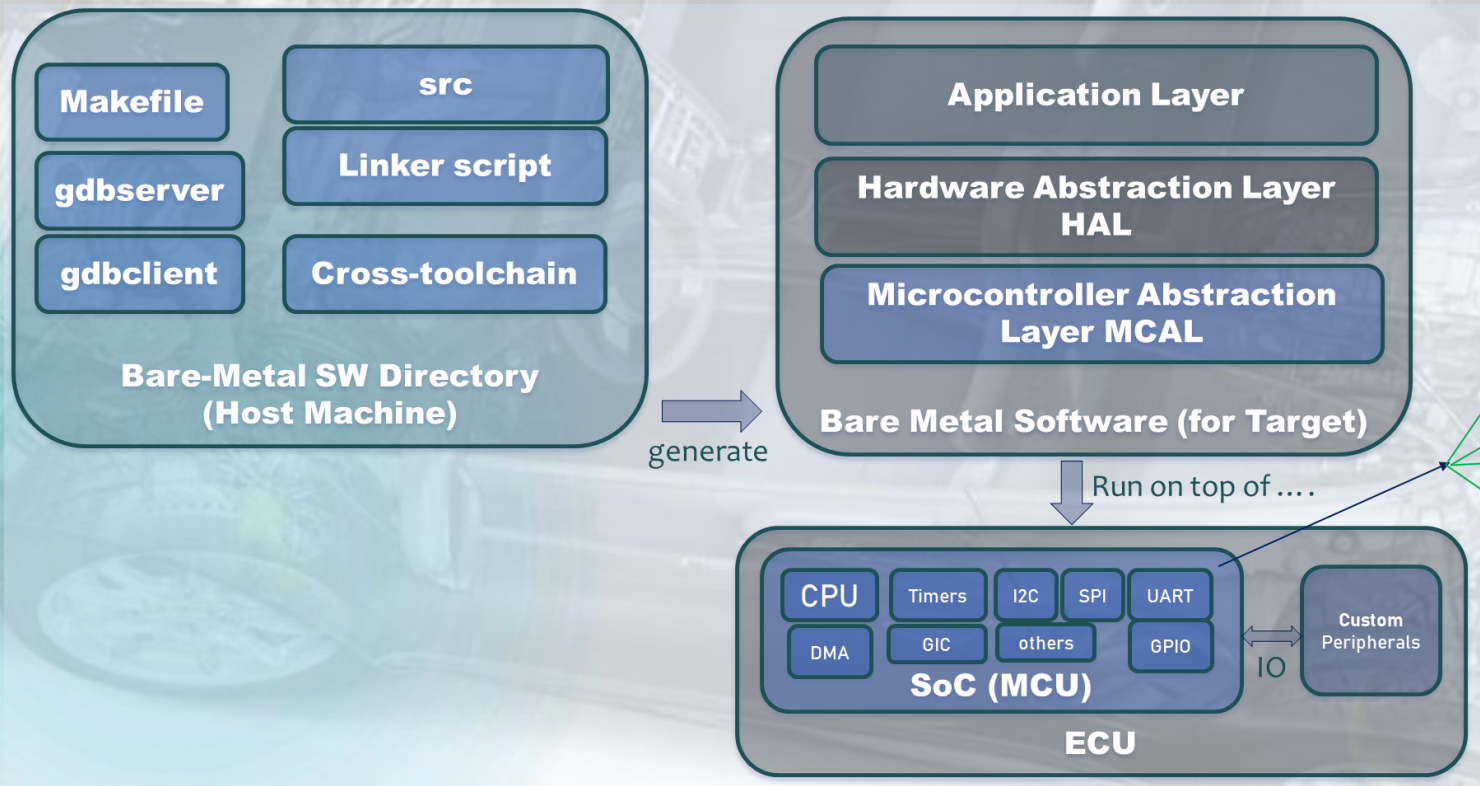
- Introduction to Interrupts
- What is an Interrupt?
- Interrupt HW Signals (inside SoC)
- Interrupt Service Routines (ISR)
- functional attribute: weak and alias for ISR
- Non-Vectored Priority System
- Vectored Arbitration System
- Interrupt vector Table (IVT)
  - Read IVT for different MCUs
- What is the Difference Between Polling And Interrupt ?
- Instruction Cycle with Interrupts
- Deep Dive in Interrupt Processing
- What is interrupt latency?
- Interrupt Example on Atmga32
- Sequential interrupt processing VS Nested interrupt processing
- Lab1: Write a Vector Section in *.s or *.c and add it in *ld linker script
- HW Interrupts / *Exceptions Interrupt* (Traps, *Faults, Aborts and Programmed* exceptions)
- Navigate IVT from STM32F103XX TRM
- Interrupt Controller IC
- Lab2: enable Ext Interrupt for Stm32F103XX
- What is Interrupt Overload?

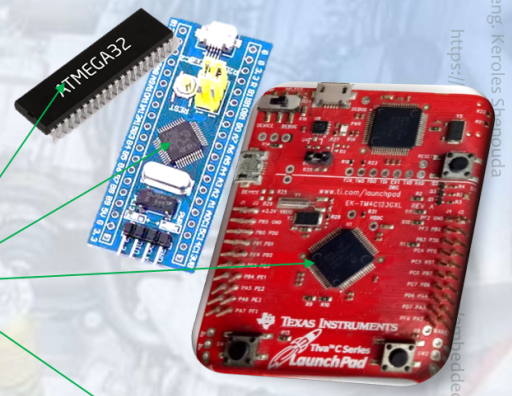


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Big Picture

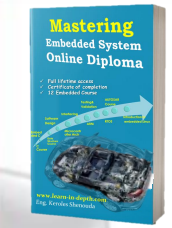




X – SoC

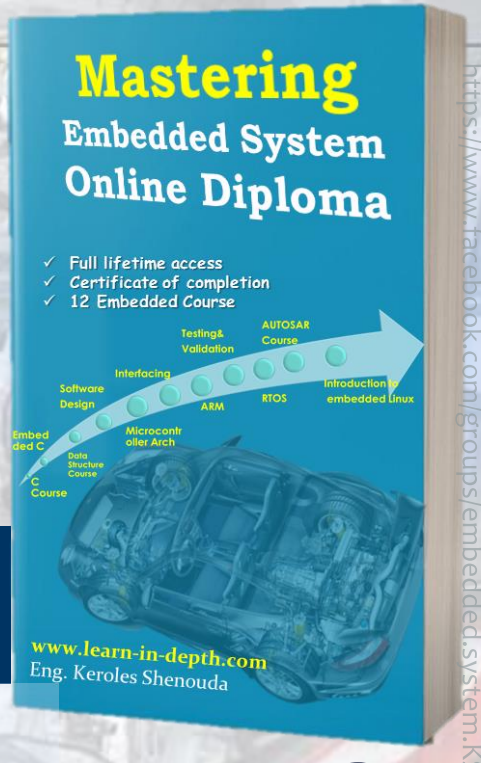
Read Specs and Program X - SoC

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

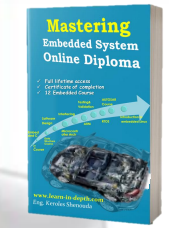
#Be_professional_in_embedded_system



Introduction to Interrupts

LEARN-IN-DEPTH
Be professional in embedded system

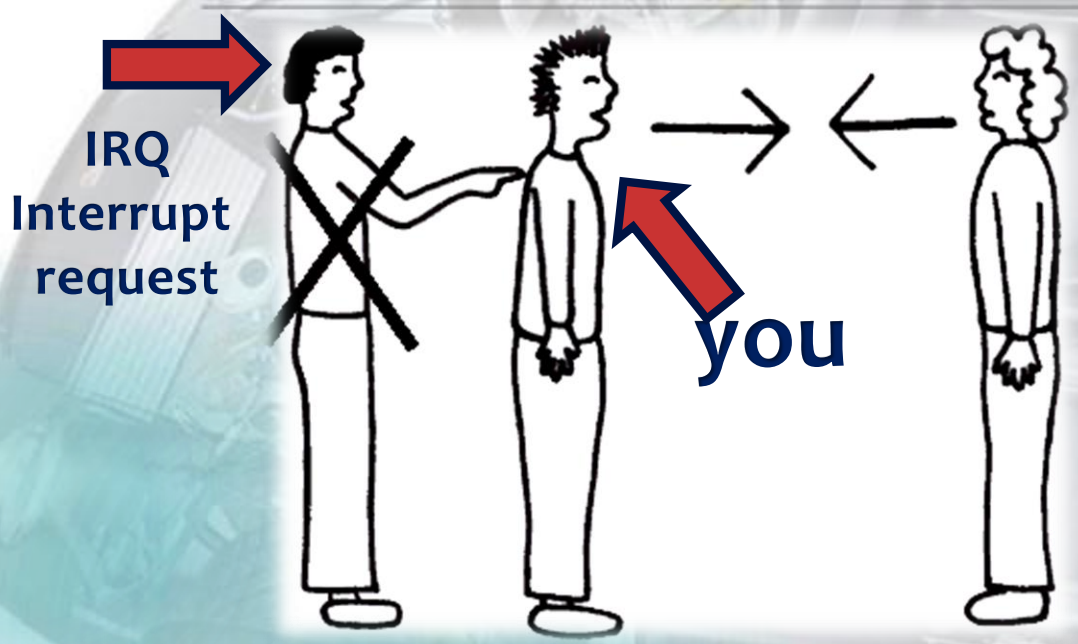
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

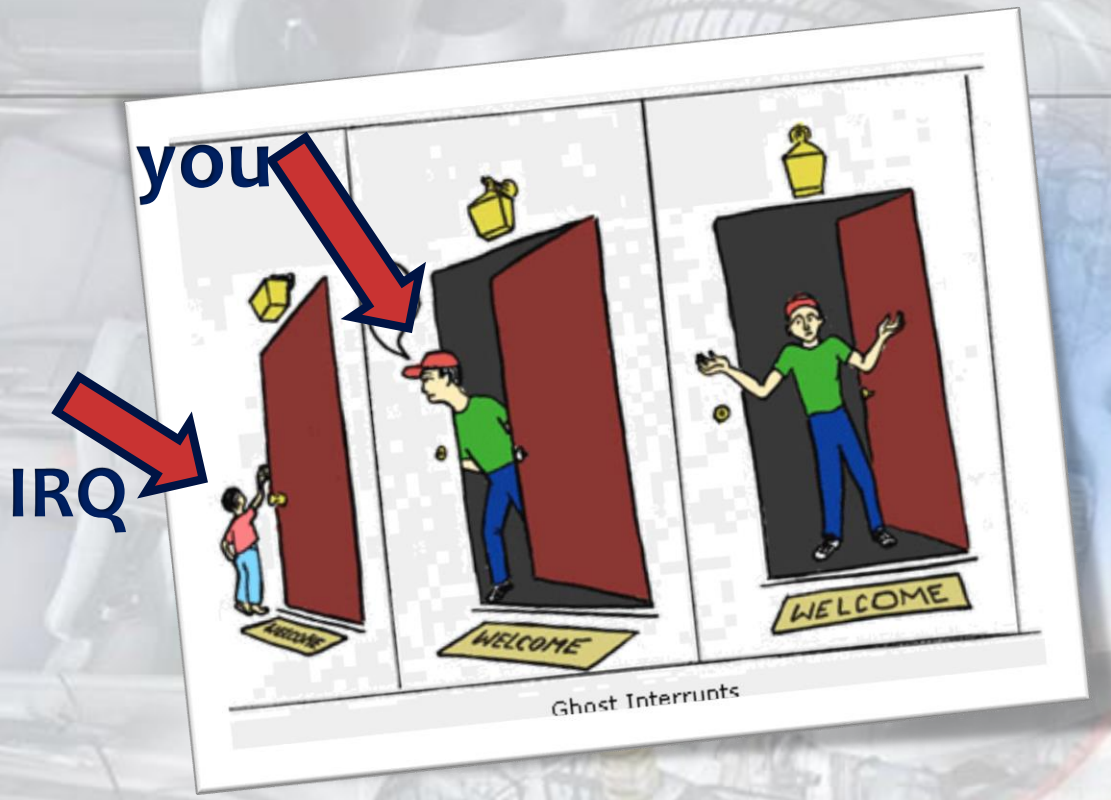


#LEARN_IN DEPTH

#Be_professional_in embedded_system

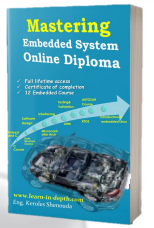
Interrupt definition in social life





Interrupt make you to stop what you are doing and jump to the one who request you

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

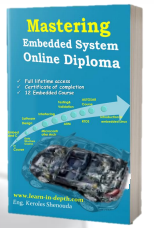


What is an Interrupt?

- An **interrupt** is a **signal** (an "**interrupt request**") generated by some event external to the CPU , which causes **the CPU to stop what it is doing** (stop executing the code it is currently running) **and jump to a separate piece of code designed by the programmer** to deal with the event which generated the interrupt request.

Execute each task sequential

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

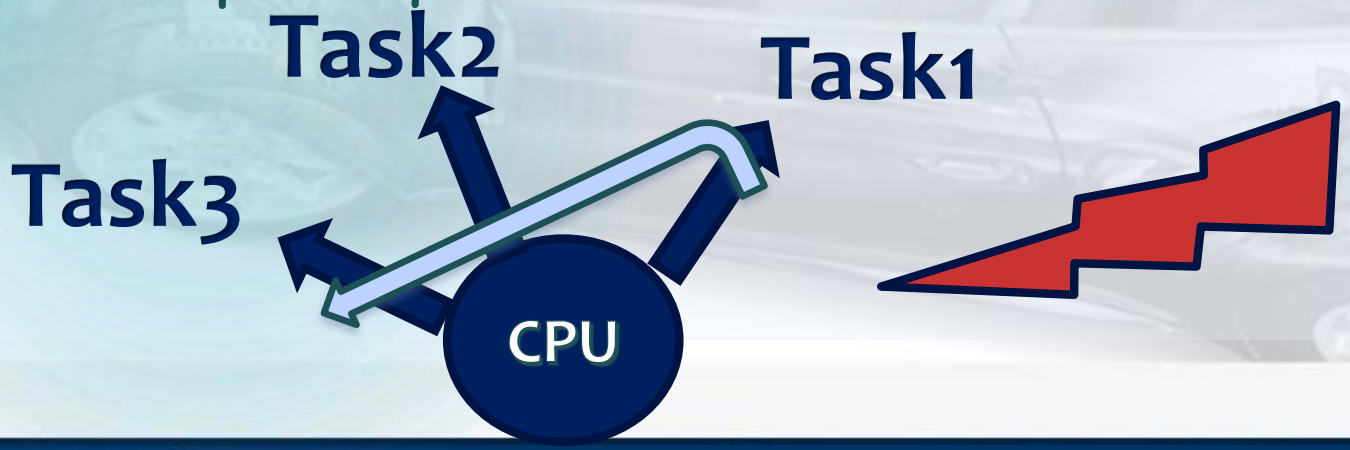


#LEARN_IN DEPTH

#Be_professional_in embedded_system

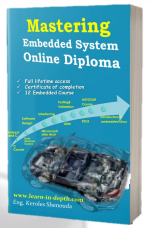
What is an Interrupt?

- An **interrupt** is a **signal** (an "**interrupt request**") generated by some event external to the CPU , which causes **the CPU to stop what it is doing** (stop executing the code it is currently running) **and jump to a separate piece of code designed by the programmer** to deal with the event which generated the interrupt request.



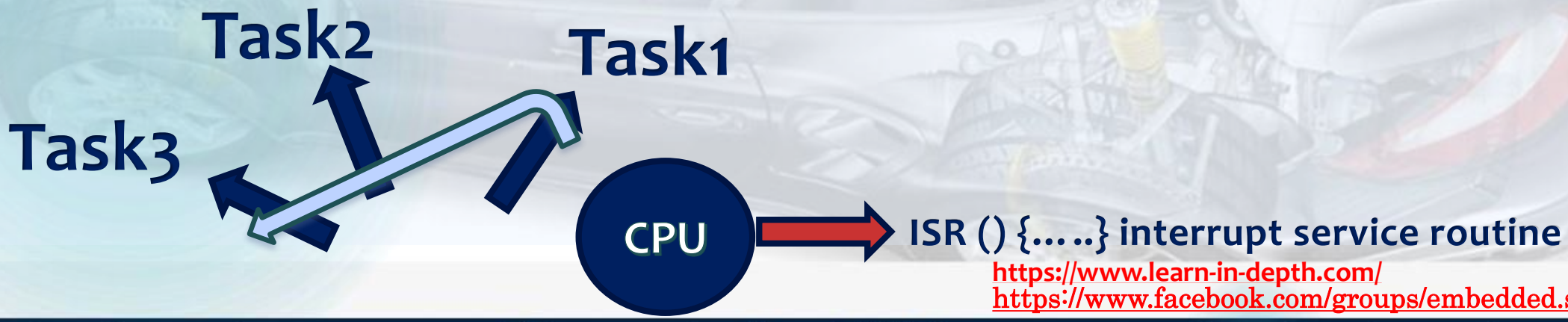
Irq (interrupt request)

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

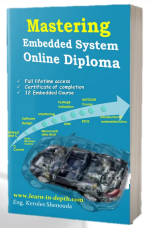


What is an Interrupt?

- An **interrupt** is a **signal** (an "**interrupt request**") generated by some event external to the CPU , which causes **the CPU to stop what it is doing** (stop executing the code it is currently running) **and jump to a separate piece of code designed by the programmer** to deal with the event which generated the interrupt request.

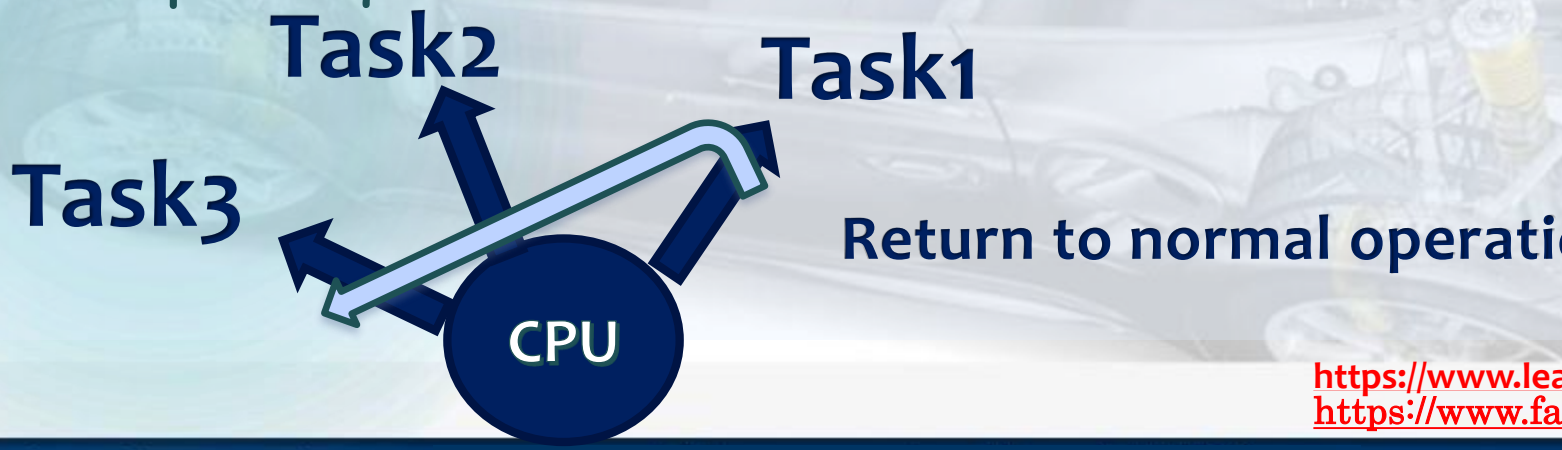


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

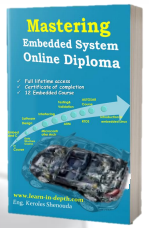


What is an Interrupt?

- An **interrupt** is a **signal** (an "**interrupt request**") generated by some event external to the CPU , which causes **the CPU to stop what it is doing** (stop executing the code it is currently running) **and jump to a separate piece of code designed by the programmer** to deal with the event which generated the interrupt request.



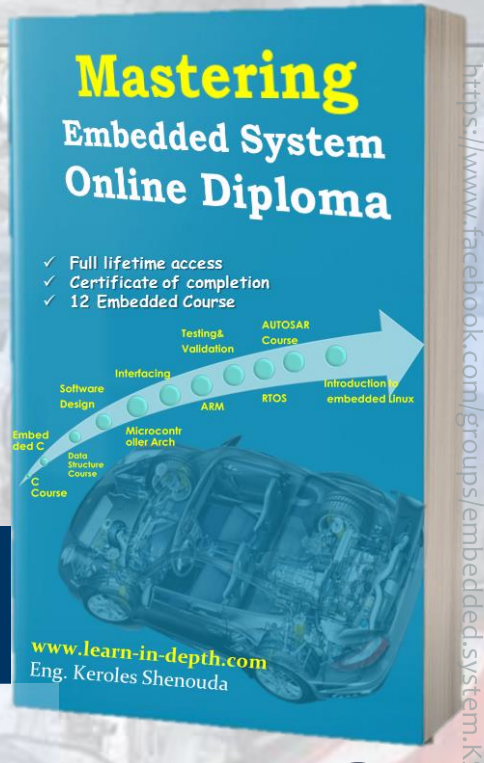
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

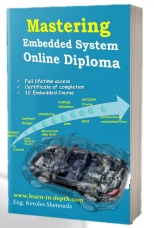
#Be_professional_in_embedded_system

What is an Interrupt?



LEARN-IN-DEPTH
Be professional in embedded system

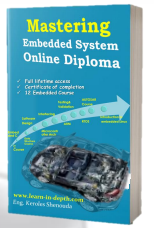
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Interrupt

- An **interrupt** is a **signal** (an "**interrupt request**") generated by some event external to the CPU , which causes **the CPU to stop what it is doing** (stop executing the code it is currently running) **and jump to a separate piece of code designed by the programmer** to deal with the event which generated the interrupt request.
- **This interrupt handling code** is often called an **ISR (interrupt service routine).** When the ISR is finished, it returns to the code that was running prior to the interrupt

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

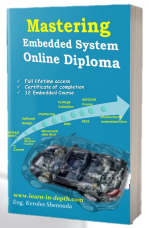
#Be_professional_in embedded_system

Interrupt Service Routine

- For every interrupt, there must be an interrupt service routine (ISR), or **interrupt handler**.
- When an interrupt occurs, the microcontroller runs the interrupt service routine.
- For every interrupt, there is a fixed location in memory that holds the address of its interrupt service routine, ISR.
- The table of memory locations set aside to hold the addresses of ISRs is called as

the Interrupt Vector Table.

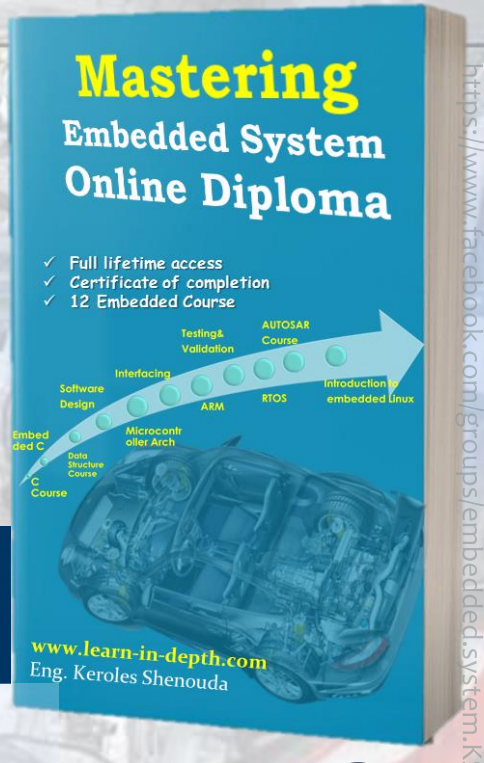
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

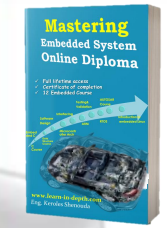
#Be_professional_in_embedded_system

Interrupt HW Signals (inside SoC)



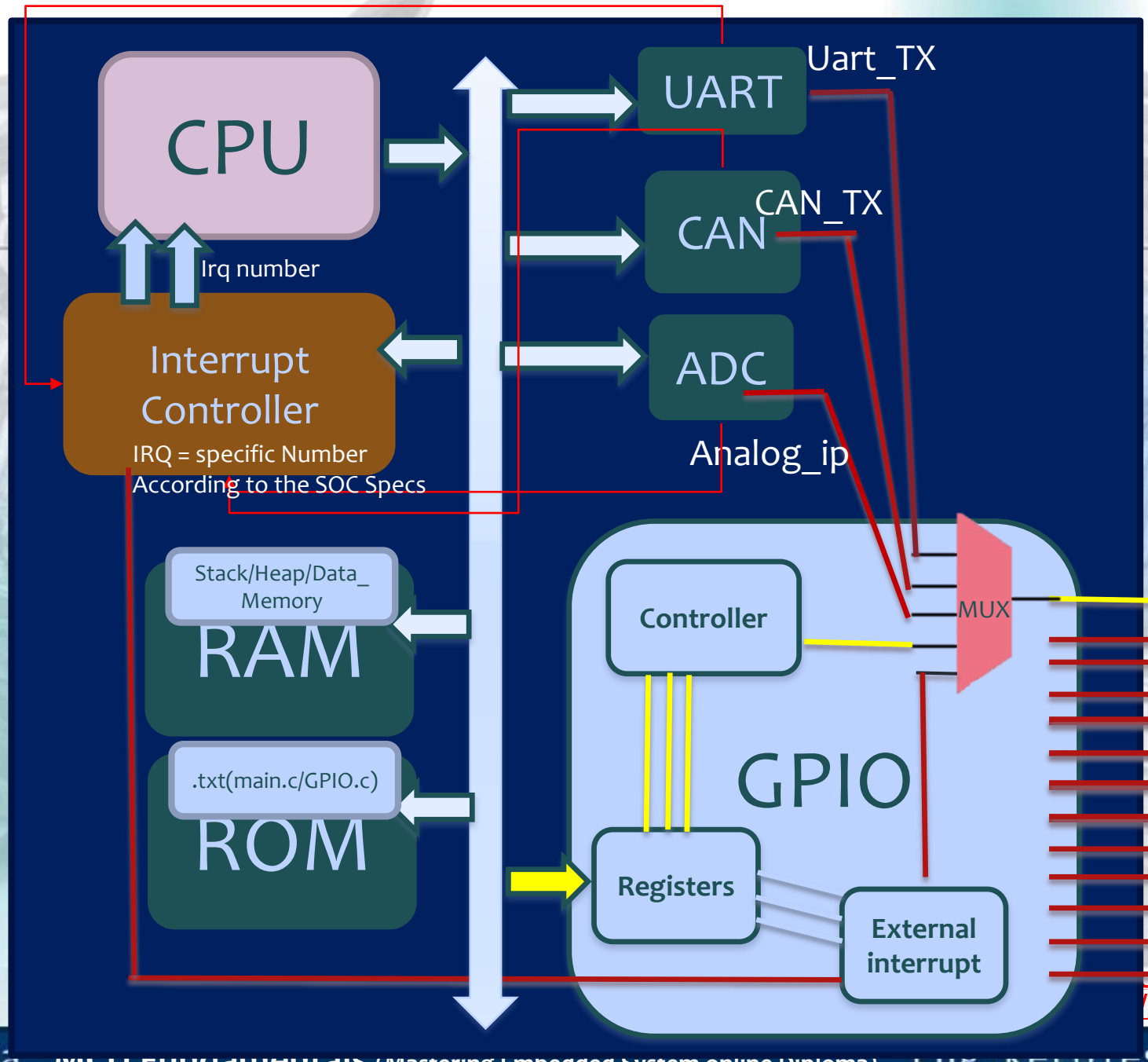
LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



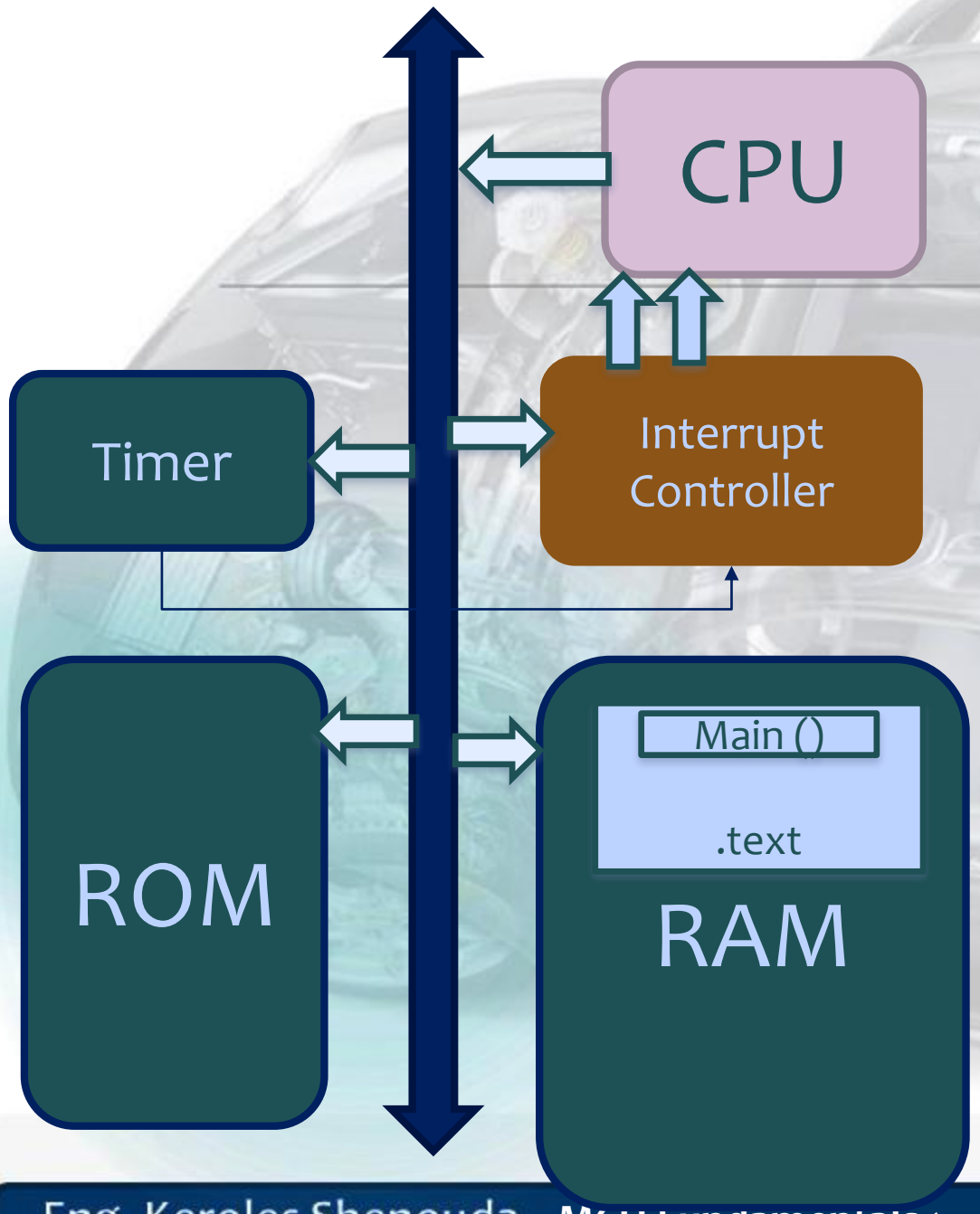
#LEARN_IN DEPTH

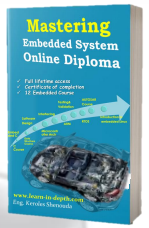
#Be_professional_in embedded_system



/groups/embedded.system.KS/

For Example



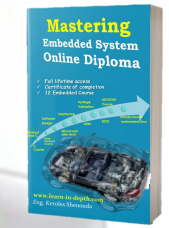


#LEARN_IN DEPTH

#Be_professional_in_embedded_system

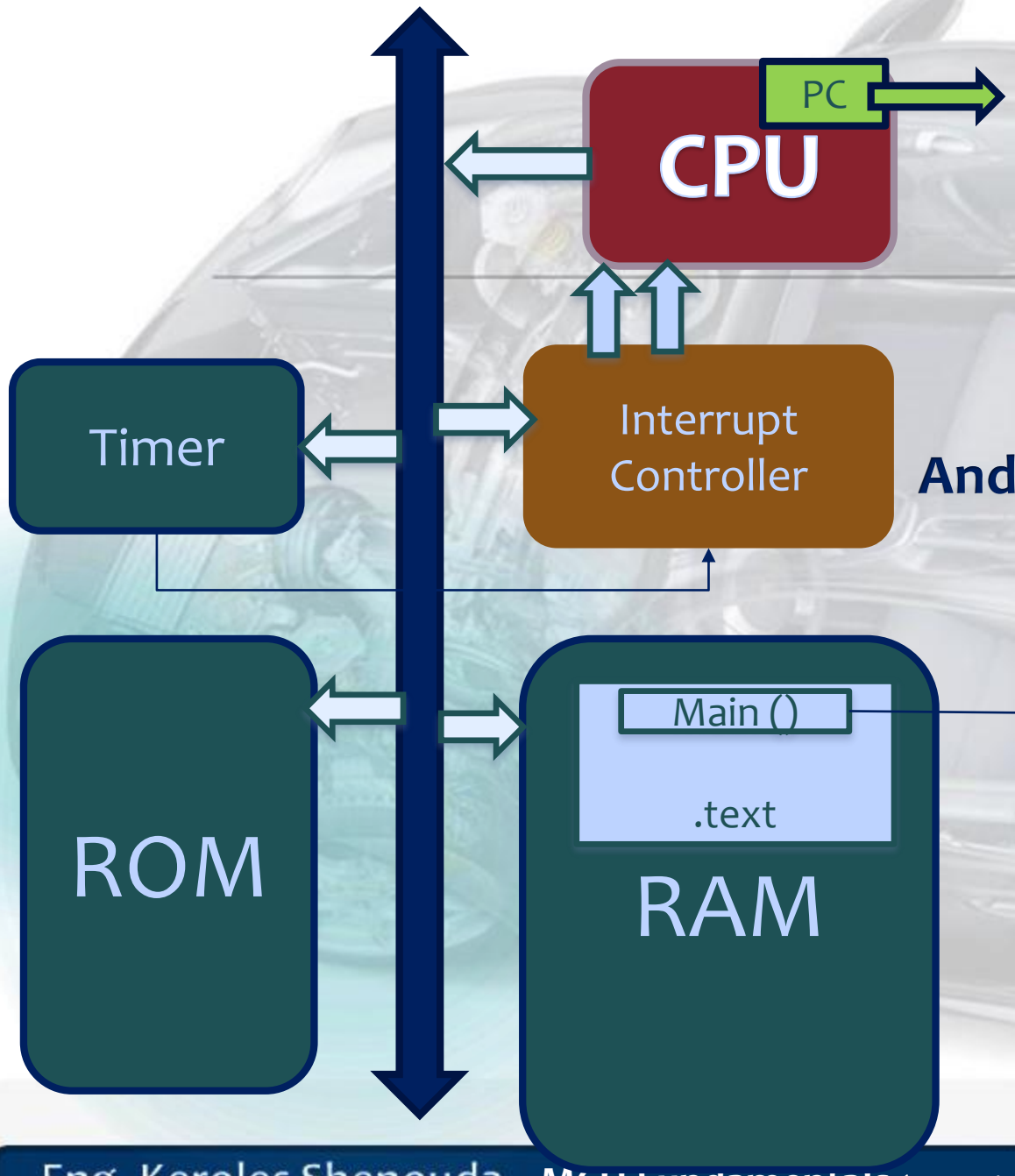
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

For Example



#LEARN_IN DEPTH

#Be_professional_in embedded_system

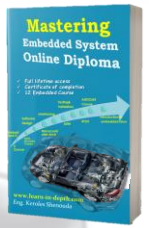


The PC "Program counter "register will
Be indicate to while(1) code
And the CPU will execute the while(1) body forever

```
Main ()
{
Timer(init);
While (1)
{
;
}

}
```

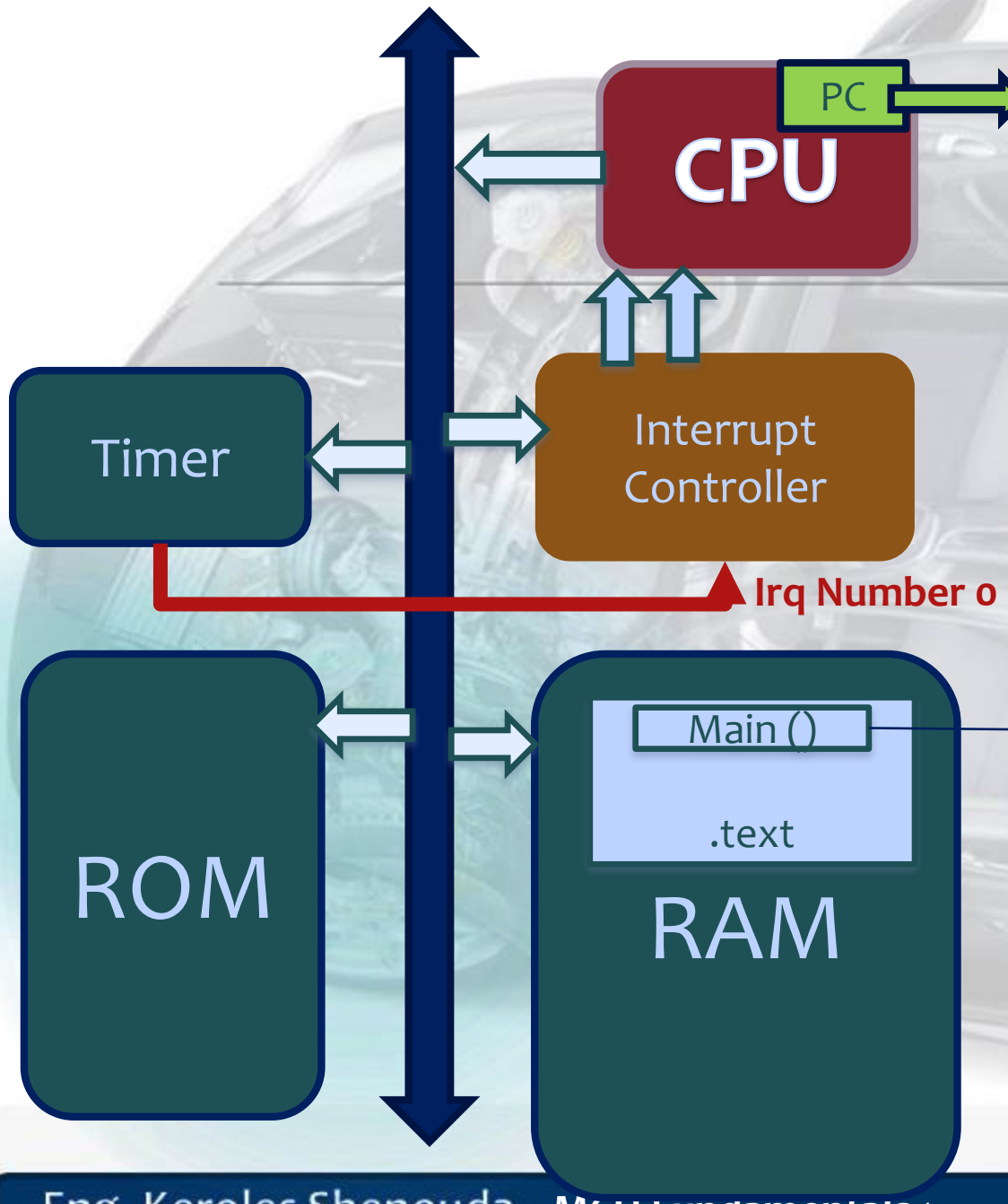
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

For Example



#LEARN_IN DEPTH

#Be_professional_in embedded_system

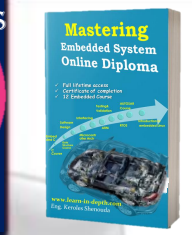


Once the Counter inside Timer reached to the max and Make overflow, the timer will generate irq" interrupt request" To the Interrupt Controller

```
Main ()
{
Timer(init);
While (1)
{
;
}

}
```

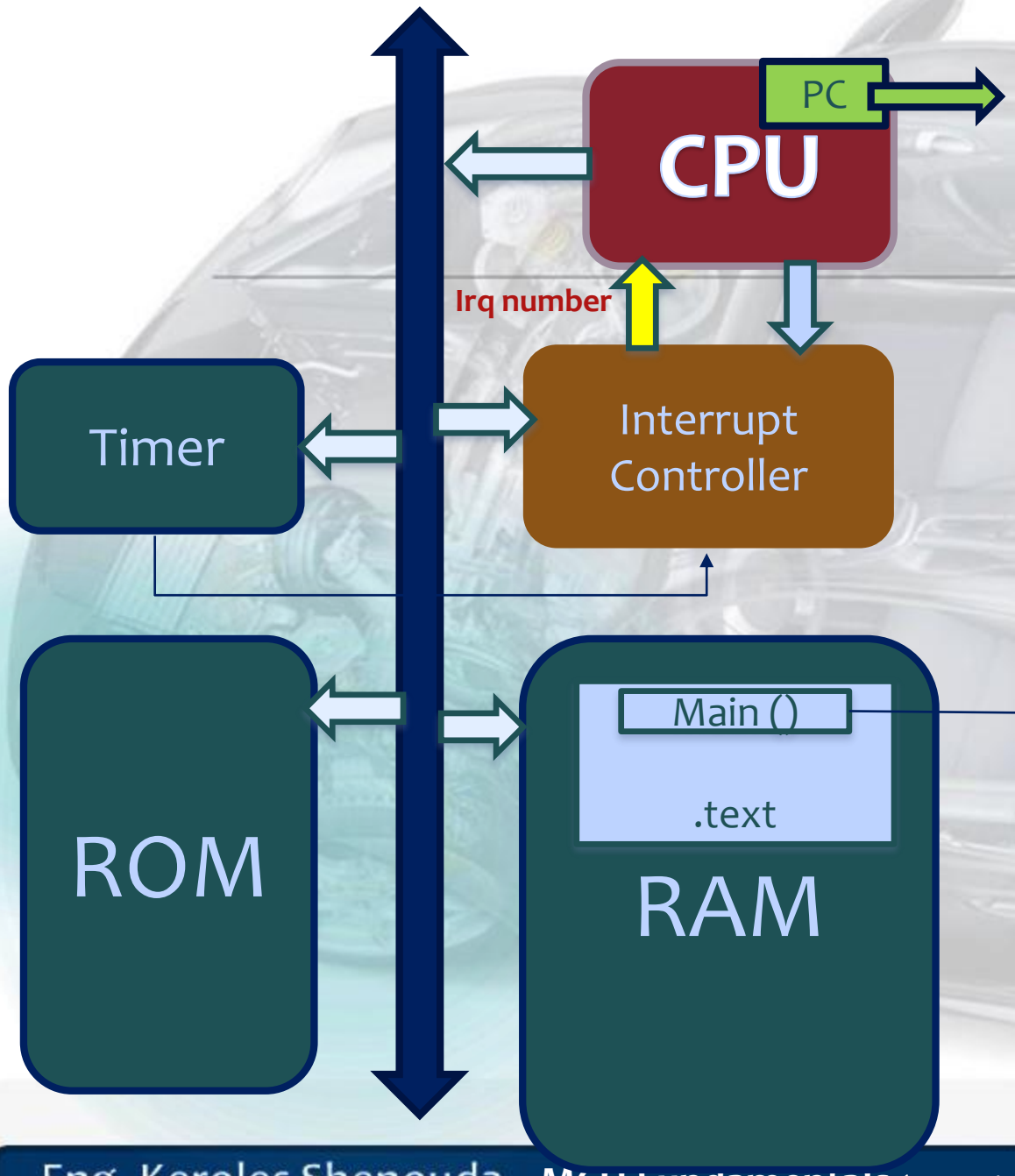
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

For Example



#LEARN_IN DEPTH

#Be_professional_in embedded_system

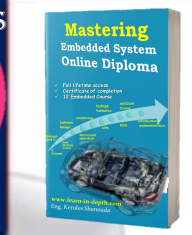


The Interrupt Controller will Decide the (is this interrupt masked or not)and check the priority then generate an interrupt request carry the interrupt Number to the CPU and wait the ACK from the CPU

```
Main ()
{
Timer(init);
While (1)
{
;
}

}
```

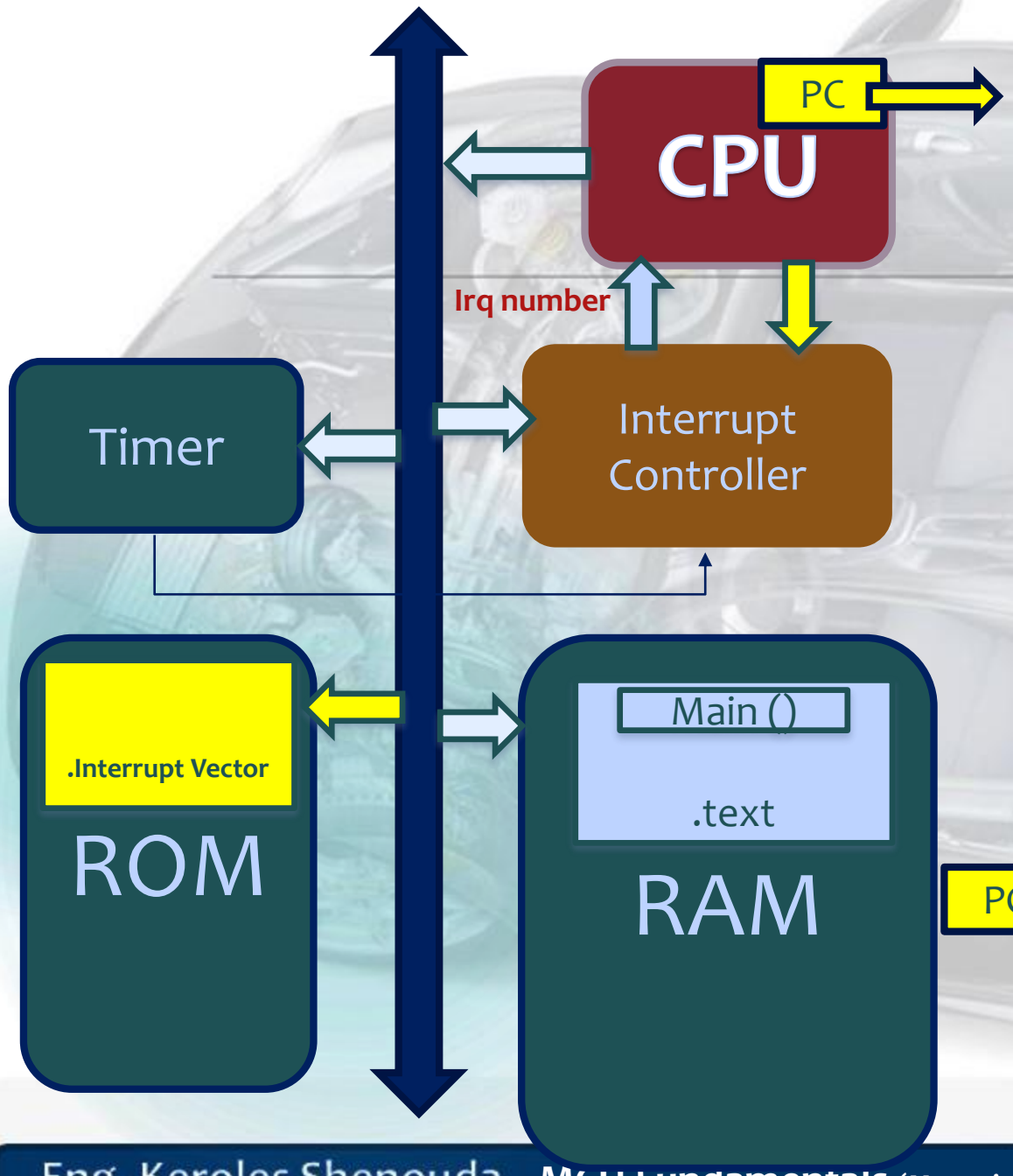
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

For Example



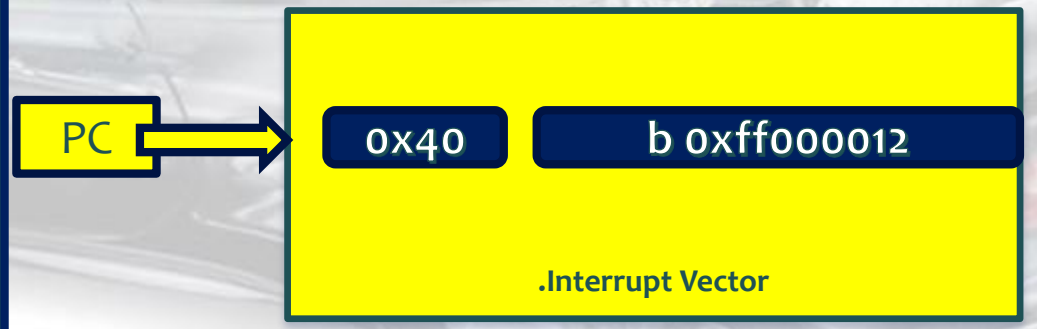
#LEARN_IN DEPTH

#Be_professional_in embedded_system



The CPU will reply by ACK to the Interrupt Controller, then will Jump the Specific Address for interrupt service routing for Timer (ISR_Timer() prototype) according to(IVT "Interrupt Vector Table")on the ROM.

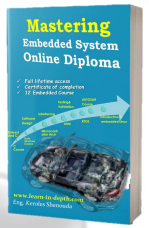
This Address on the ROM in "Interrupt Vector" Section, it is a prototype which carry the address of the definition in RAM



ISR definition

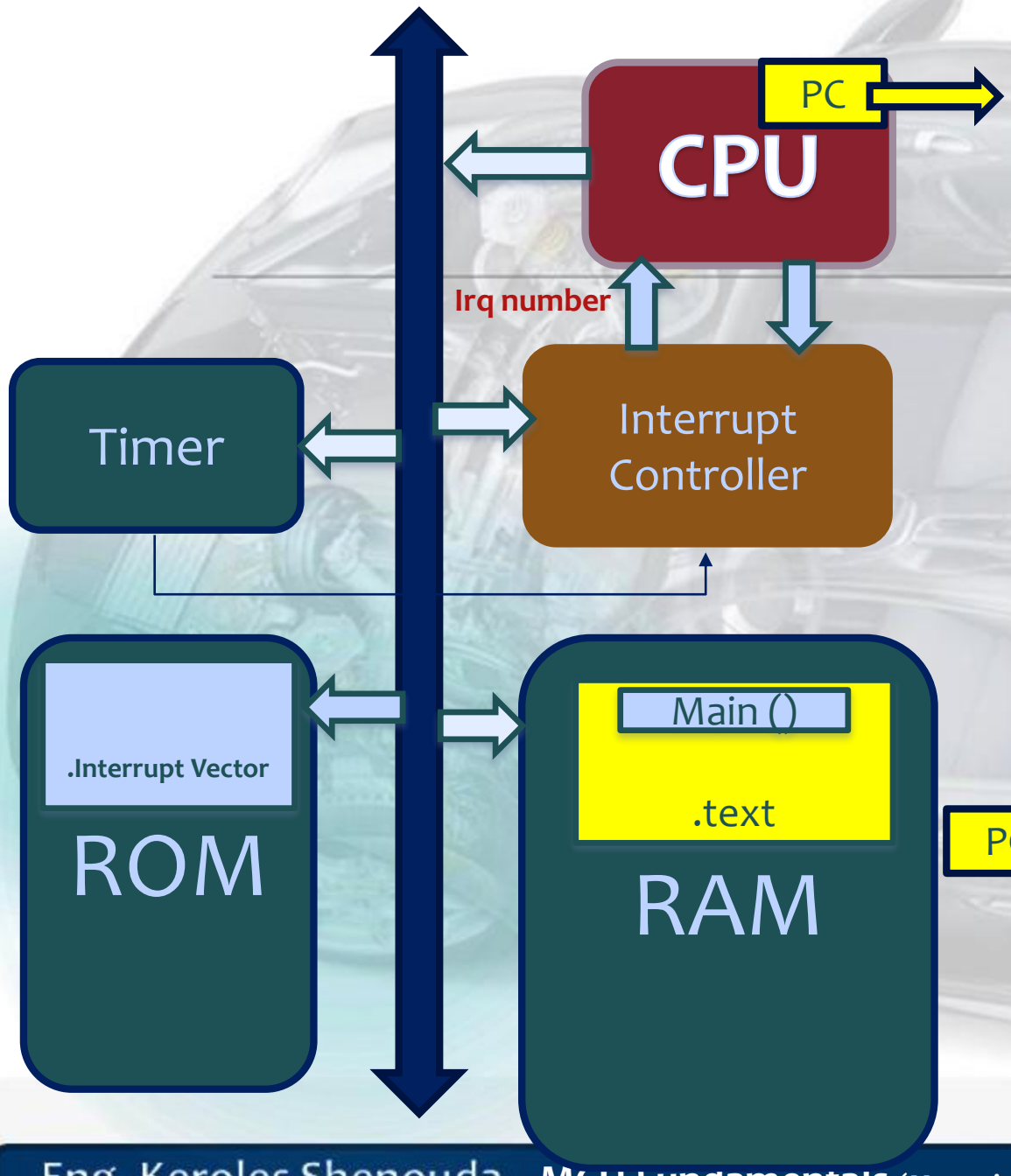
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

For Example



#LEARN_IN DEPTH

#Be_professional_in embedded_system

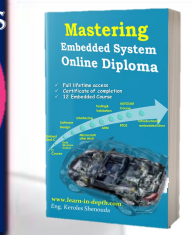


Then the CPU will execute the ISR for Timer Overflow action

ISR definition

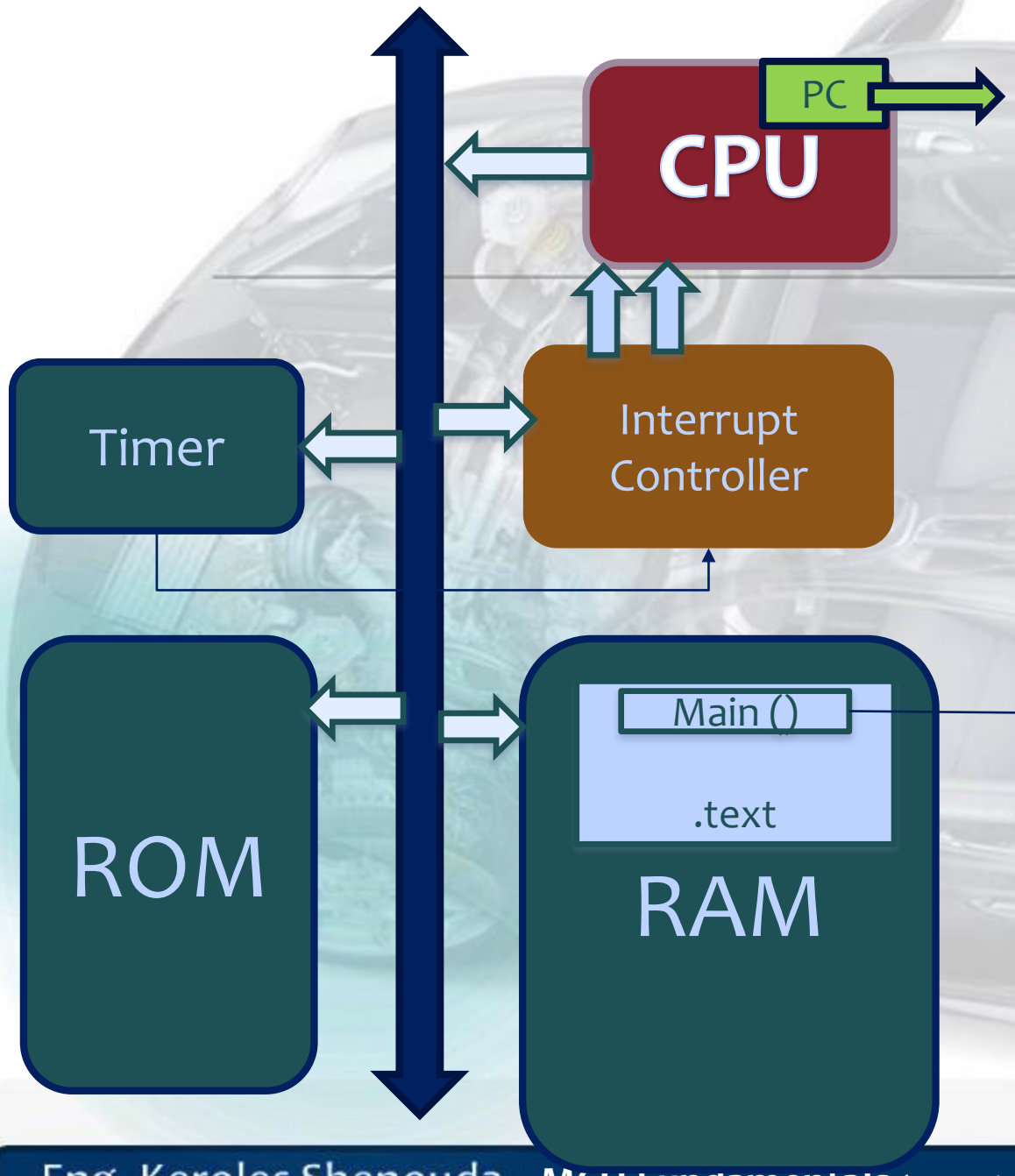
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

For Example



#LEARN_IN DEPTH

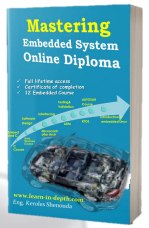
#Be_professional_in embedded_system



After the CPU handling the Interrupt the CPU will come back to the last instruction

```
Main ()
{
Timer(init);
While (1)
{
;
}

}
```

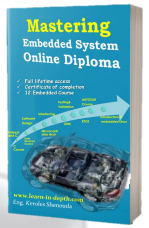
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Conclusion

- An *interrupt* leverages a combination of software and hardware to force the processor to stop its current activity and begin to execute a particular piece of code called an *interrupt service routine* (ISR).
- An ISR responds to a specific event generated by either hardware or software.
- When an ISR completes, the processor automatically resumes the activity that had been halted.
- The halted process continues as if nothing had happened.

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

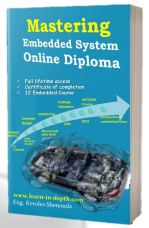


#LEARN_IN DEPTH

#Be_professional_in_embedded_system

An interrupt is simply a hardware-invoked function call.

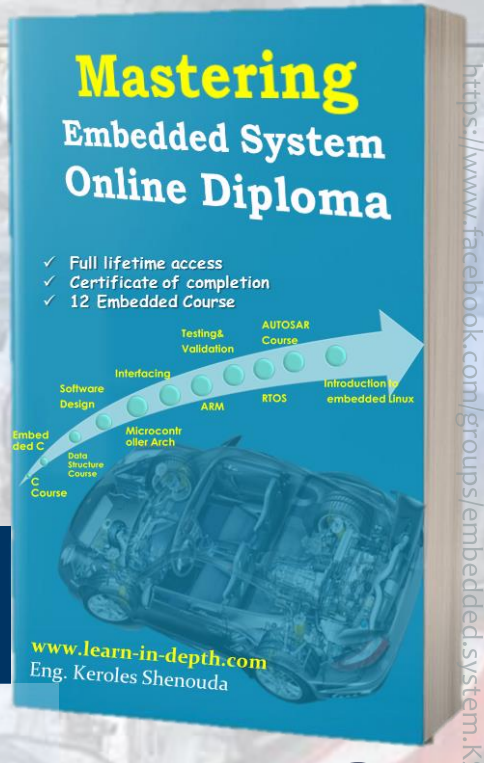
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

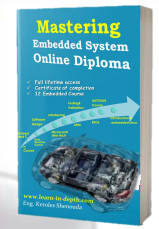
#Be_professional_in_embedded_system

Interrupt Service Routines (ISR)



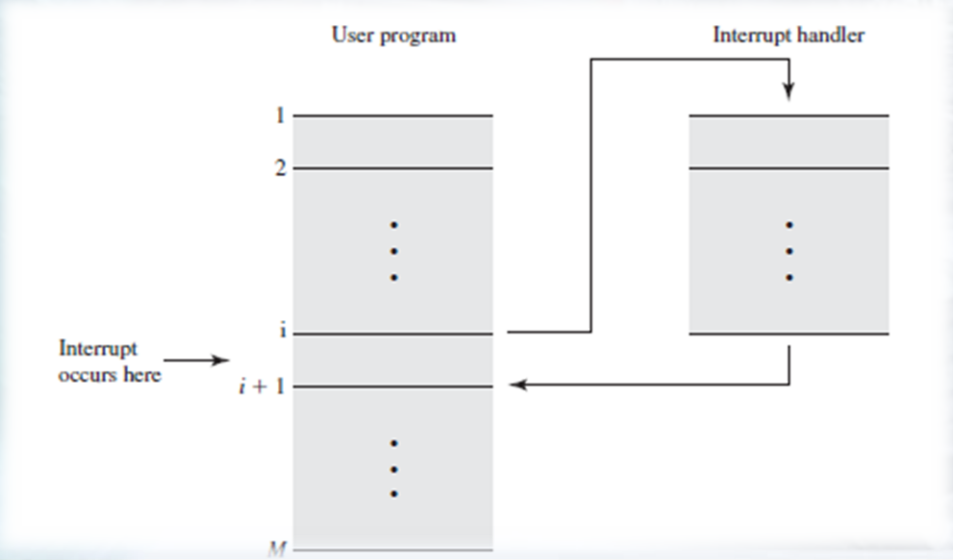
LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

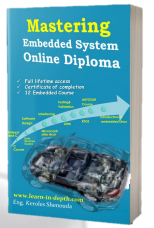


Interrupt Handling

- Code executed by an interrupt is not generally considered part of the main application. Since this code handles the cases where an interrupt occurs, it is called an interrupt handler or an **interrupt service routine**.



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



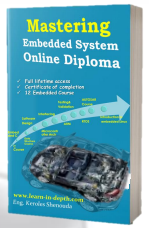
Interrupt Service Routines

- An interrupt service routine (ISR), also called an *interrupt handler*, is a special subroutine that hardware invokes automatically in response to an interrupt.
- Each ISR has a default implementation in the system startup code
- All ISRs are declared as weak in the system startup code.
- The keyword **weak** means that another non-weak subroutine with the same name defined elsewhere can override this one.

C Code	Assembly Code

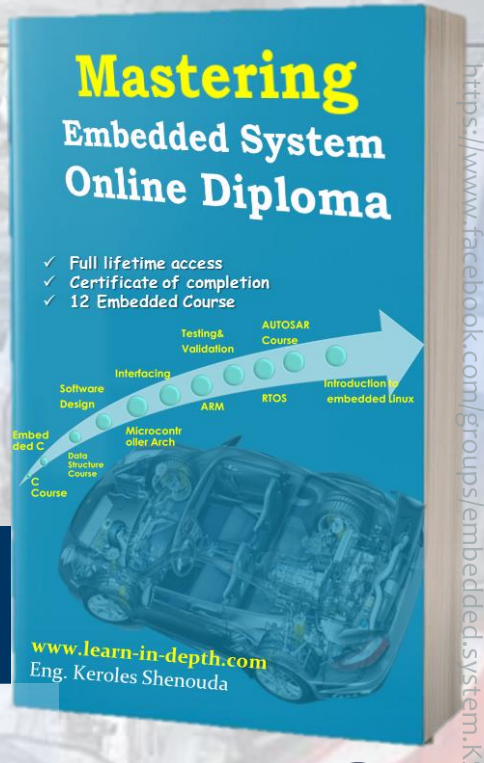
`void SysTick_Handler (void) {`
`  ...`
`}`
	`SysTick_Handler  PROC`
	`  EXPORT  SysTick_Handler`
	`  ...`
	`  ENDP`

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

#Be_professional_in_embedded_system



functional attribute: weak and alias for ISR

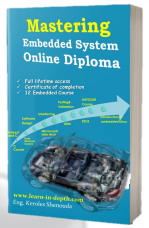
PLEASE REVIEW EMBEDDED C (UNIT3.LESSON3)

LEARN-IN-DEPTH

Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

functional attribute: weak and alias in embedded c



- **weak** The weak attribute causes the declaration to be emitted as **a weak symbol** rather than a **global**.
- This is primarily useful in defining library functions that can be **overridden** in user code, though it can also be used with non-function declarations.
- **alias ("target")** The alias attribute causes the declaration to be emitted as an alias for another symbol, which must be specified.

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Startup.c

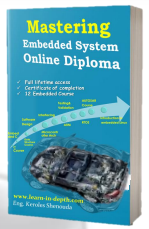
linker_script.ld | startup.c | main.c | startup.s | toggle_led_stm32f103c6.dis | Makefile | log.txt

```
1  //Learn-in-depth
2  //Keroles Shenouda
3  //Mastering_Embedded System online diploma
4  #include <stdint.h>
5  #define STACK_Start_SP 0x20001000
6  extern int main(void);
7
8  void Rest_Handler (void)  ;
9
10 void Default_Handler()
11 {
12   Rest_Handler() ;
13 }
14
15 void NMI_Handler(void) __attribute__ ((weak, alias ("Default_Handler")));;
16 void H_fault_Handler(void) __attribute__ ((weak, alias ("Default_Handler")));;
17 void MM_Fault_Handler(void) __attribute__ ((weak, alias ("Default_Handler")));;
18 void Bus_Fault(void) __attribute__ ((weak, alias ("Default_Handler")));;
19 void Usage_Fault_Handler(void) __attribute__ ((weak, alias ("Default_Handler")));;
20
21 uint32_t vectors[] __attribute__((section(".vectors")))   = {
22 STACK_Start_SP,
23 (uint32_t) &Rest_Handler,
24 (uint32_t) &NMI_Handler ,
25 (uint32_t) &H_fault_Handler ,
26 (uint32_t) &MM_Fault_Handler ,
27 (uint32_t) &Bus_Fault ,
28 (uint32_t) &Usage_Fault_Handler
29 };
30 void Rest_Handler (void)
31 {
32     main();
33 }
34
```

FOLLOW US

Press here

28



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

```
kkhalil@egc-kkhalil-lt MINGW64 /d/courses/new
$ arm-none-eabi-nm.exe learn-in-depth.elf
08000098 W Bus_Fault
08000098 T Default_Handler
08000098 W H_fault_Handler
0800001c T main
08000098 W MM_Fault_Handler
08000098 W NMI_Handler
080000b0 D R_ODR
080000a4 T Rest_Handler
08000098 W Usage_Fault_Handler
08000000 T vectors
```

alias

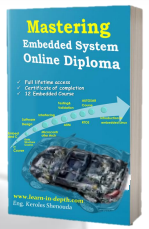
PC → 0x8 → b 0x08000098

.Interrupt Vector

ISR definition

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

We can override on weak symbol in main.c



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

linker_script.ld | startup.c | main.c | startup.s | toggle_led_stm32f103c6.dis | Makefile | log.txt

```
1  //Learn-in-depth
2  //Keroles Shenouda
3  //Mastering_Embedded System online diploma
4  typedef volatile unsigned int vuint32_t ;
5  #include <stdint.h>
6  // register address
7  #define RCC_BASE       0x40021000
8  #define GPIOA_BASE     0x40010800
9  #define RCC_APB2ENR    *(volatile uint32_t *)(RCC_BASE   + 0x18)
10 #define GPIOA_CRH      *(volatile uint32_t *)(GPIOA_BASE + 0x04)
11 #define GPIOA_ODR      *(volatile uint32_t *)(GPIOA_BASE + 0x0C)
12 // bit fields
13 #define RCC_IOPAEN     (1<<2)
14 #define GPIOA13        (1UL<<13)
15
16 extern void NMI_Handler(void)
17 {
18
19 }
20 extern void Bus_Fault(void)
21 {
22
23 }
24
25 typedef union  {
26         vuint32_t     all_fields;
27         struct {
```

```
$ arm-none-eabi-nm.exe learn-in-depth.elf
08000028 T Bus_Fault
080000b0 T Default_Handler
080000b0 W H_fault_Handler
08000034 T main
080000b0 W MM_Fault_Handler
0800001c T NMI_Handler
080000c8 D R_ODR
080000bc T Rest_Handler
080000b0 W Usage_Fault_Hand
08000000 T vectors
```

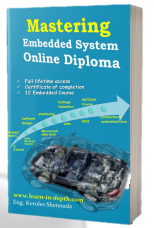
weak

PC → 0x8 | b 0x0800001c

.Interrupt Vector

ISR definition

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



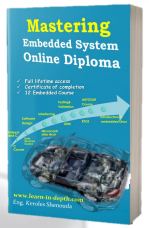
Servicing Interrupts

- There are two general ways in which microcontrollers service interrupts, each

with several variations.

Vectored Arbitration System

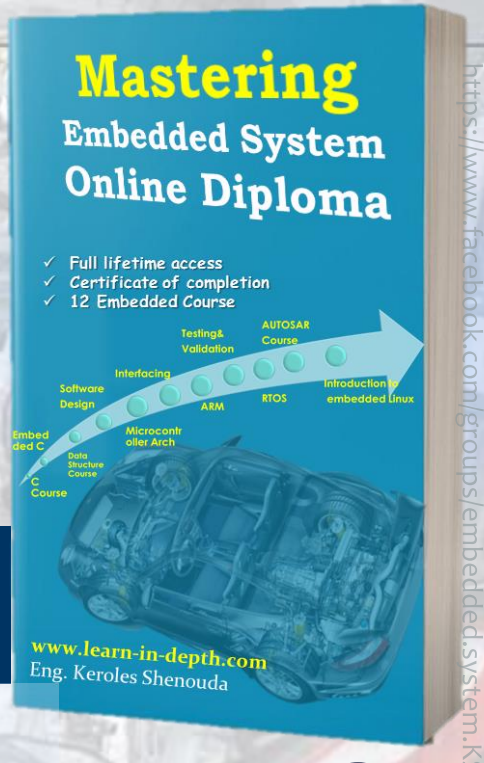
Non-Vectored Priority System



#LEARN_IN DEPTH

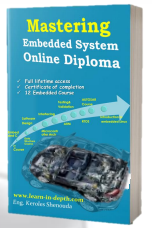
#Be_professional_in_embedded_system

Non-Vectored Priority System



LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Non-Vectored Priority System

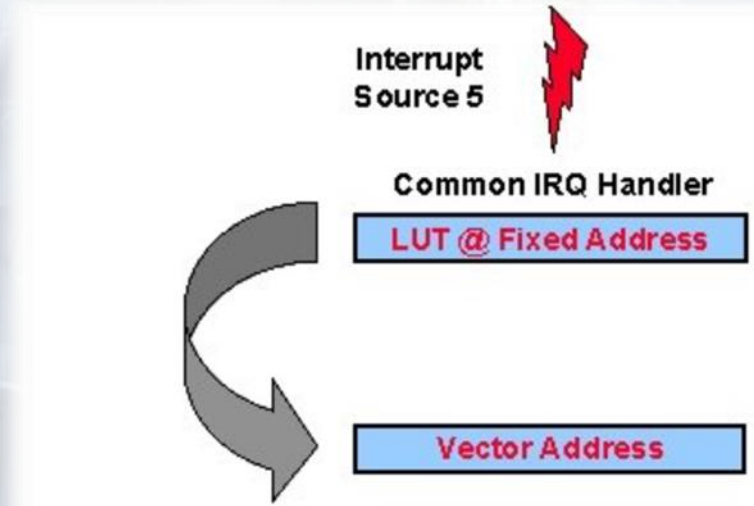
- When an interrupt occurs, **the PC branches to a specific address**.
- At this address the interrupts must **be checked sequentially to determine which one has caused the interrupt.**

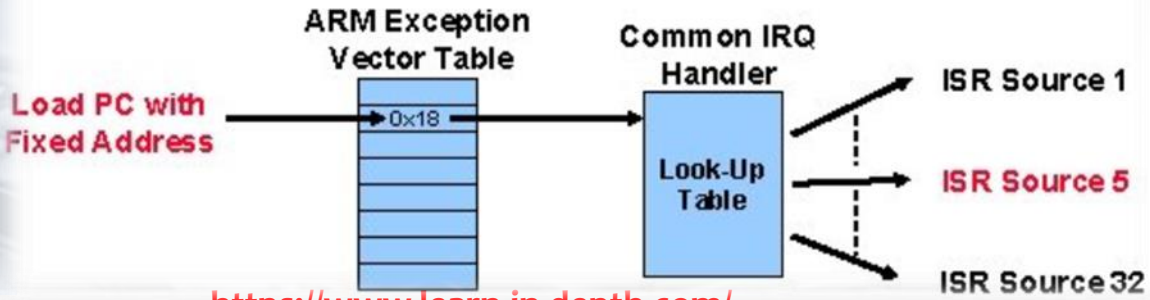
This scheme can be very slow and there can be a large delay between the time

the interrupt occurs and the time it is serviced. However, the programmer can

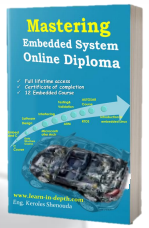
set the interrupt priority and non-vectored interrupts are feasible for

microcontrollers with less than five interrupts.





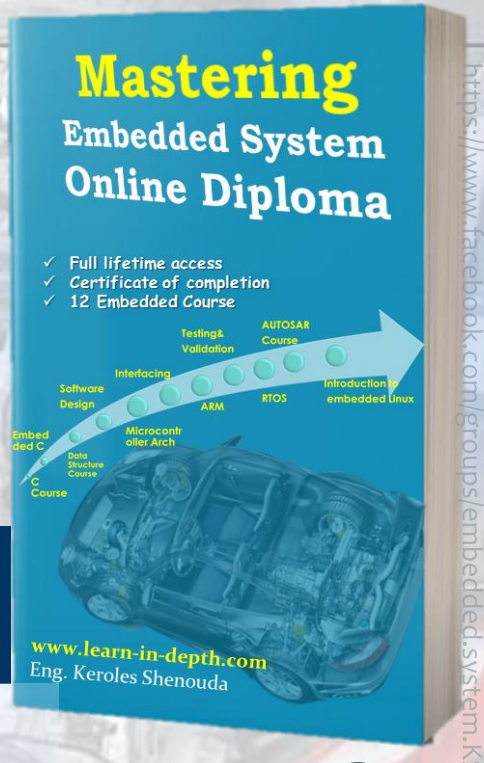
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

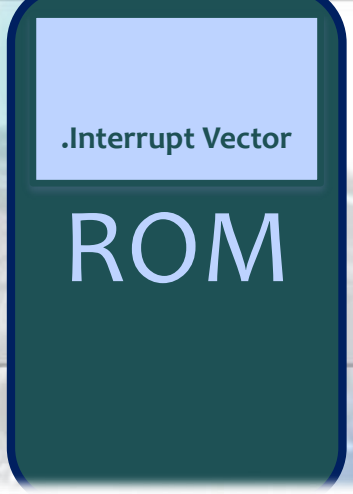
Vectored Arbitration System



LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Vectored Arbitration System



Some machines reserve a portion of program memory for interrupt vectors.

The location of each particular vector in program memory may vary from

processor to processor but it cannot be changed by the programmer.

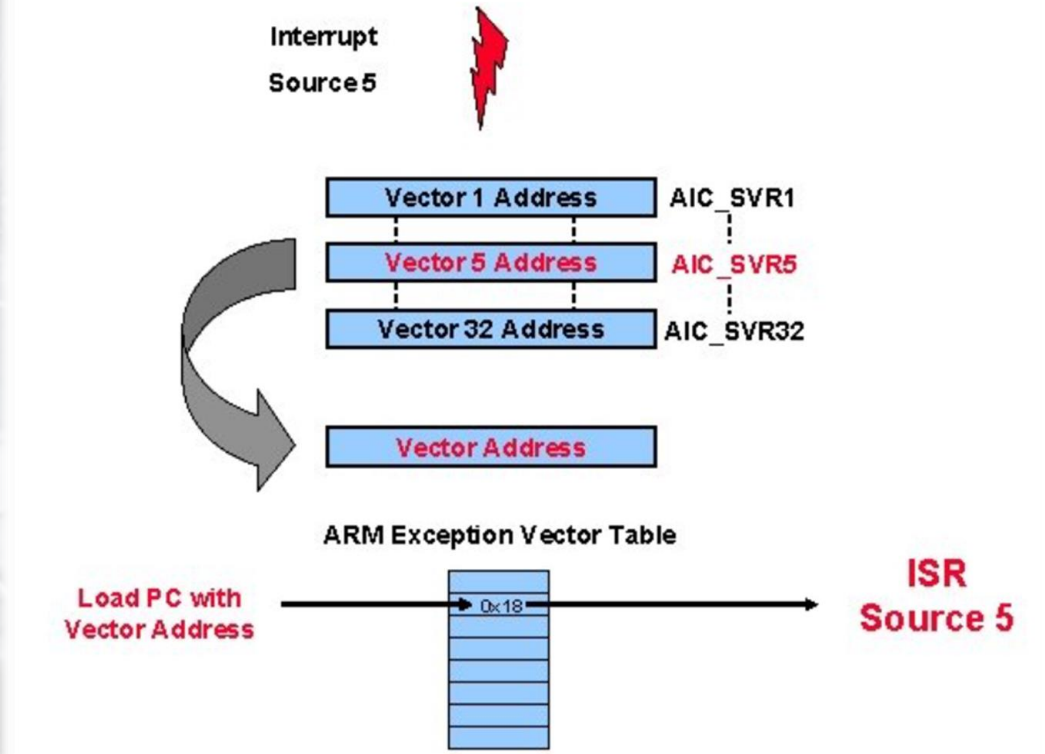
The programmer can only change the data at each vector location.

Each interrupt vector contains the address of that interrupt's service routine.

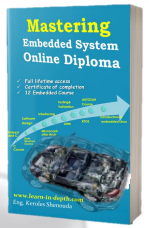
When the compiler allocates program memory for interrupt handlers/ISR,

it places the appropriate address for the handler/ISR in the appropriate interrupt vector.

- To help the compiler you must usually tell it where the interrupt vector for each interrupt is located in program memory



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Interrupt vector Table

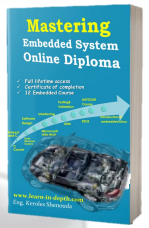
- **interrupt vector** is the memory address of an **interrupt handler**. The interrupt vector for each interrupt provided by the microcontrollers Vendor and should be found inside its datasheet.
- Please note here that the **interrupt vectors are apart of the microcontroller's program memory.** As such when utilizing interrupts this section of memory should be reserved to store pointers to interrupt handlers and not to store regular programs
- The IVT Should contain those informations

IVT Interrupt Vector Table

Irq Number	Vector address	Irq Source

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

IVT-Example interrupt vector Table on ATMEGA 32



15.1. Interrupt Vectors in ATmega32A

Table 15-1 Reset and Interrupt Vectors

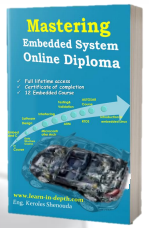
Vector No.	Program Address(2)	Source	Interrupt Definition
1	0x000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	0x002	INT0	External Interrupt Request 0
3	0x004	INT1	External Interrupt Request 1
4	0x006	INT2	External Interrupt Request 2
5	0x008	TIMER2 COMP	Timer/Counter2 Compare Match
6	0x00A	TIMER2 OVF	Timer/Counter2 Overflow
7	0x00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	0x00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	0x010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	0x012	TIMER1 OVF	Timer/Counter1 Overflow
11	0x014	TIMER0 COMP	Timer/Counter0 Compare Match
12	0x016	TIMER0 OVF	Timer/Counter0 Overflow
13	0x018	SPI, STC	SPI Serial Transfer Complete
14	0x01A	USART, RXC	USART, Rx Complete
15	0x01C	USART, UDRE	USART Data Register Empty
16	0x01E	USART, TXC	USART, Tx Complete

Irq Number=6

Vector address = 0x00A

Irq Source= Timer 2 Counter overflow

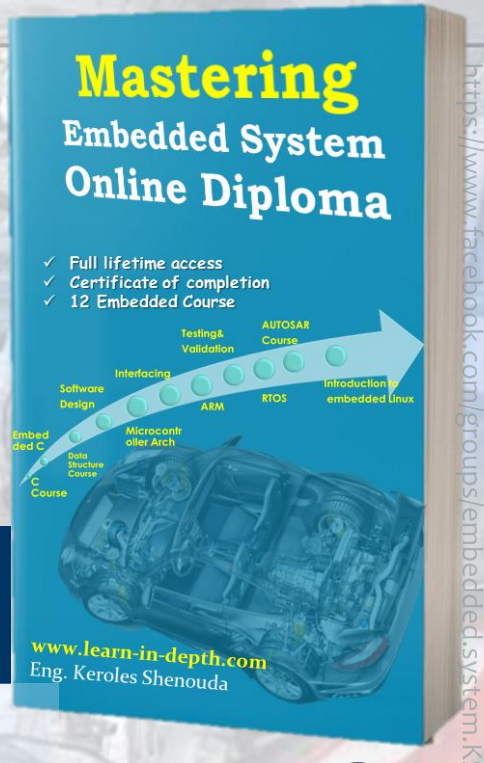
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

Read IVT for different MCUs

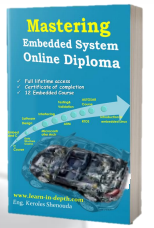


LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

IVT-Example

interrupt vector Table National Semiconductor COP8SAA7



#LEARN_IN DEPTH

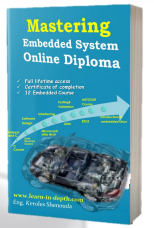
#Be_professional_in embedded_system

Rank	Source	Description	Vector Address *
1	Software	INTR Instruction	0bFE - 0bFF
2	Reserved	Future	0bFC - 0bFD
3	External	G0	0bFA - 0bFB
4	Timer T0	Underflow	0bF8 - 0bF9
5	Timer T1	T1A/Underflow	0bF6 - 0bF7
6	Timer T1	T1B	0bF4 - 0bF5
7	MICROWIRE/PLUS	BUSY Low	0bF2 - 0bF3
8	Reserved	Future	0bF0 - 0bF1
9	Reserved	Future	0bEE - 0bEF
10	Reserved	Future	0bEC - 0bED
11	Reserved	Future	0bEA - 0bEB
12	Reserved	Future	0bE8 - 0bE9
13	Reserved	Future	0bE6 - 0bE7
14	Reserved	Future	0bE4 - 0bE5
15	Port L/Wakeup	Port L Edge	0bE2 - 0bE3
16	Default	VIS Instruction Execution without any interrupts	0bE0 - 0bE1

Question Figure Out The Highlighted Line

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

IVT-Example interrupt vector Table National Semiconductor COP8SAA7



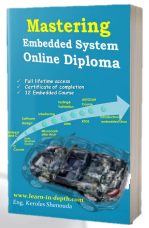
Rank	Source	Description	Vector Address *
1	Software	INTR Instruction	0bFE - 0bFF
2	Reserved	Future	0bFC - 0bFD
3	External	G0	0bFA - 0bFB
4	Timer T0	Underflow	0bF8 - 0bF9
5	Timer T1	T1A/Underflow	0bF6 - 0bF7
6	Timer T1	T1B	0bF4 - 0bF5
7	MICROWIRE/PLUS	BUSY Low	0bF2 - 0bF3
8	Reserved	Future	0bF0 - 0bF1
9	Reserved	Future	0bEE - 0bEF
10	Reserved	Future	0bEC - 0bED
11	Reserved	Future	0bEA - 0bEB
12	Reserved	Future	0bE8 - 0bE9
13	Reserved	Future	0bE6 - 0bE7
14	Reserved	Future	0bE4 - 0bE5
15	Port L/Wakeup	Port L Edge	0bE2 - 0bE3
16	Default	VIS Instruction Execution without any interrupts	0bE0 - 0bE1

Irq Number=4

Vector address = 0xbf8-0bf9

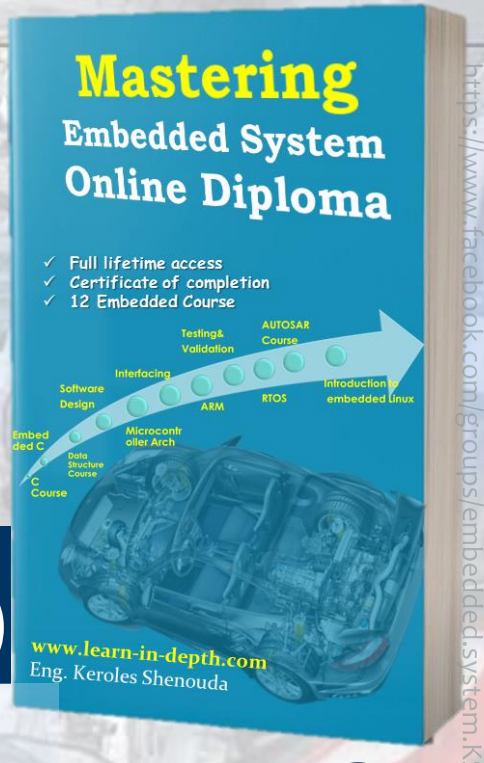
Irq Source= Timer 0 Counter underflow

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

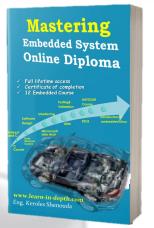


CortexM4 Microcontroller Software Interface Standard (CMSIS)

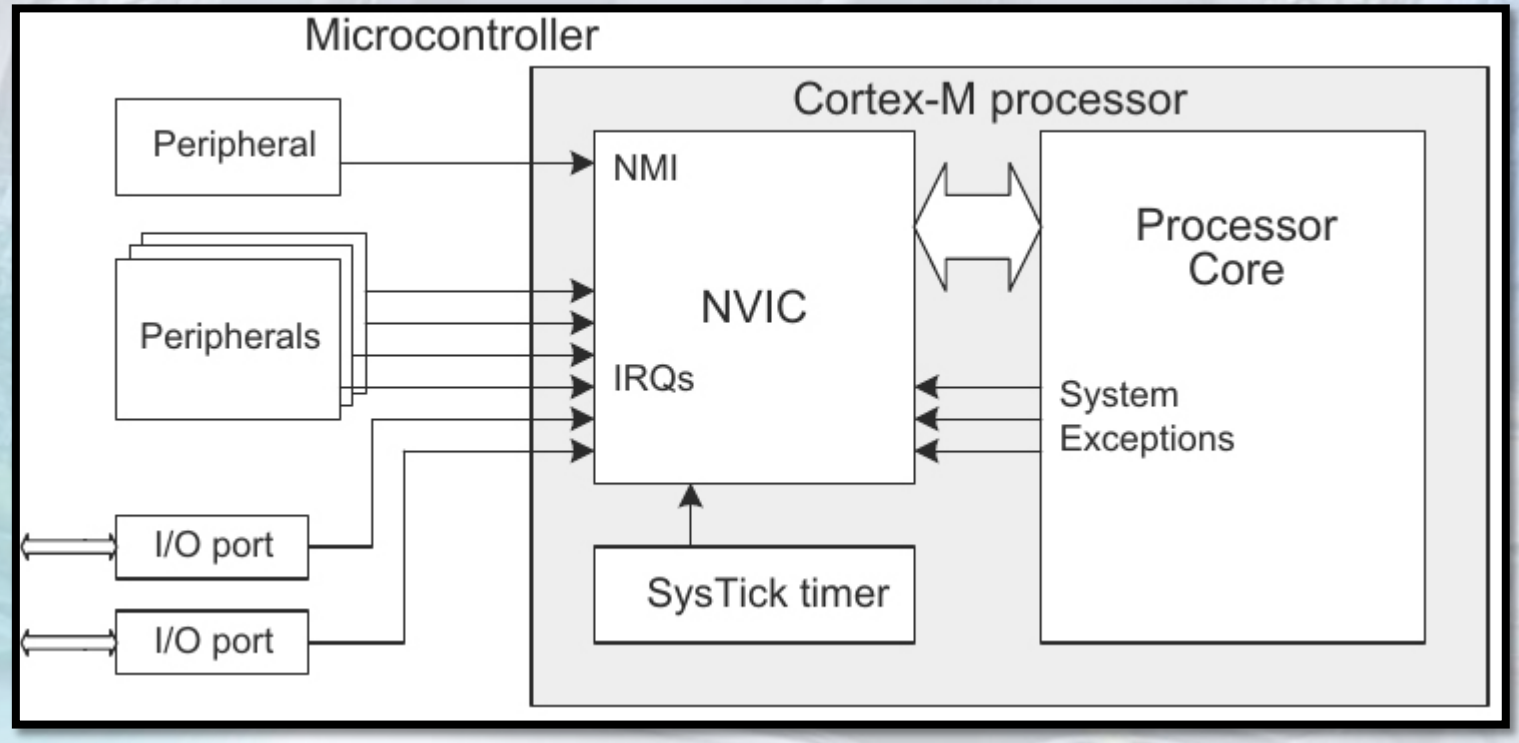
ARM & IVT

LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



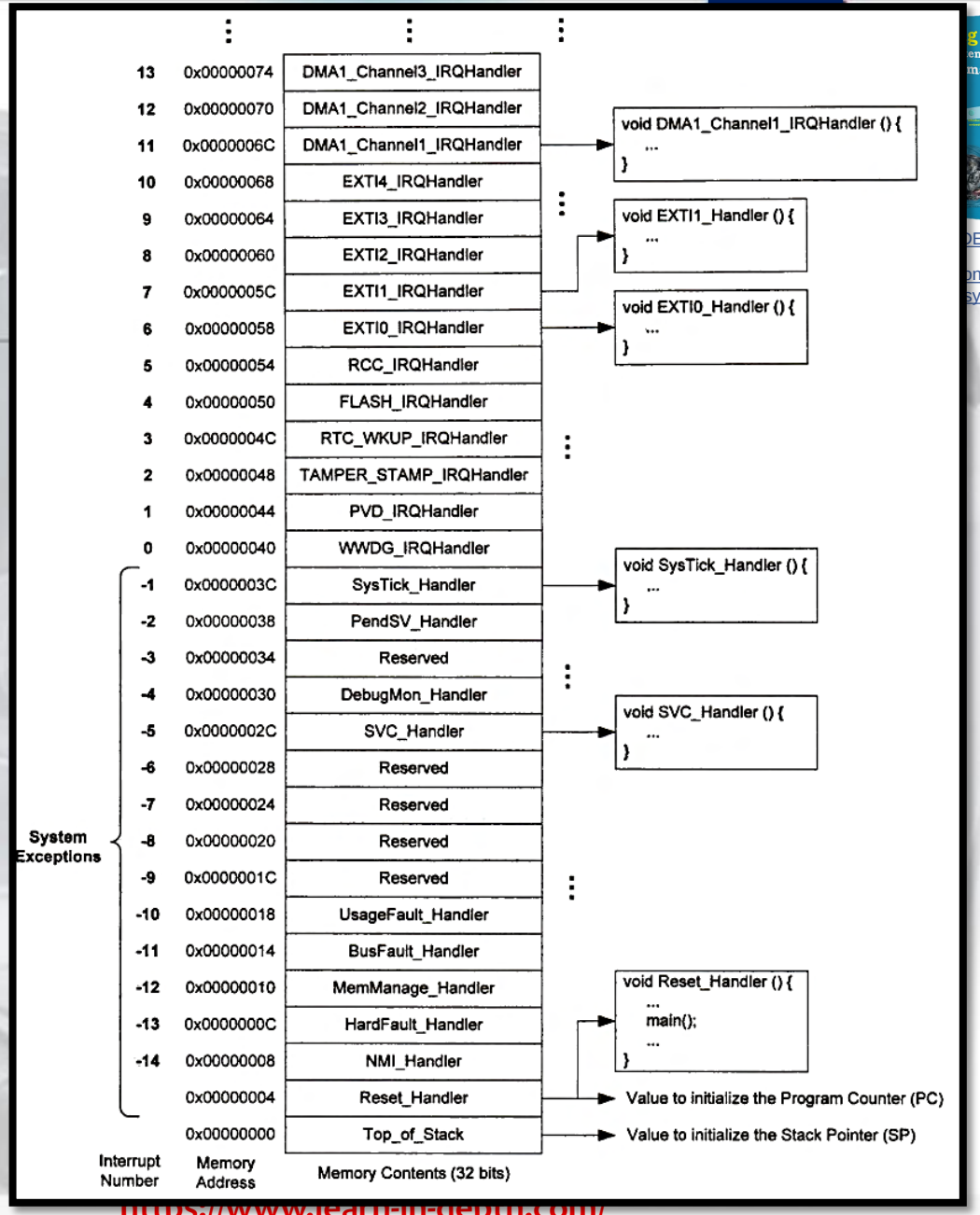
Cortex M



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Cortex-M4

- Cortex-M stores the starting memory address of every ISR in a special array called the *interrupt vector table*
- the ARM **CMSIS** library defines all system exceptions by using **negative values**.
- **CMSIS** stands for Cortex Microcontroller Software Interface Standard
- For a given interrupt number *i* defined in CMSIS, the memory address of its corresponding ISR is located at the (i + 16)th entry in the interrupt vector table.
- *Address of ISR = InterruptVectorTable[i + 15]*



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

IVT-Example interrupt vector Table TiVaC “TM4C1234 SOC”

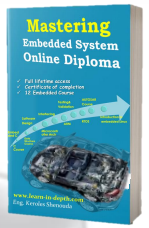
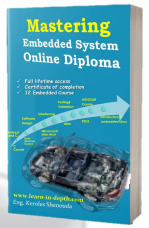


Table 2-9. Interrupts

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
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

Question Figure Out The Highlighted Line

IVT-Example interrupt vector Table TiVaC “TM4C1234 SOC”



Irq Number=21

Vector address = 0x00000054

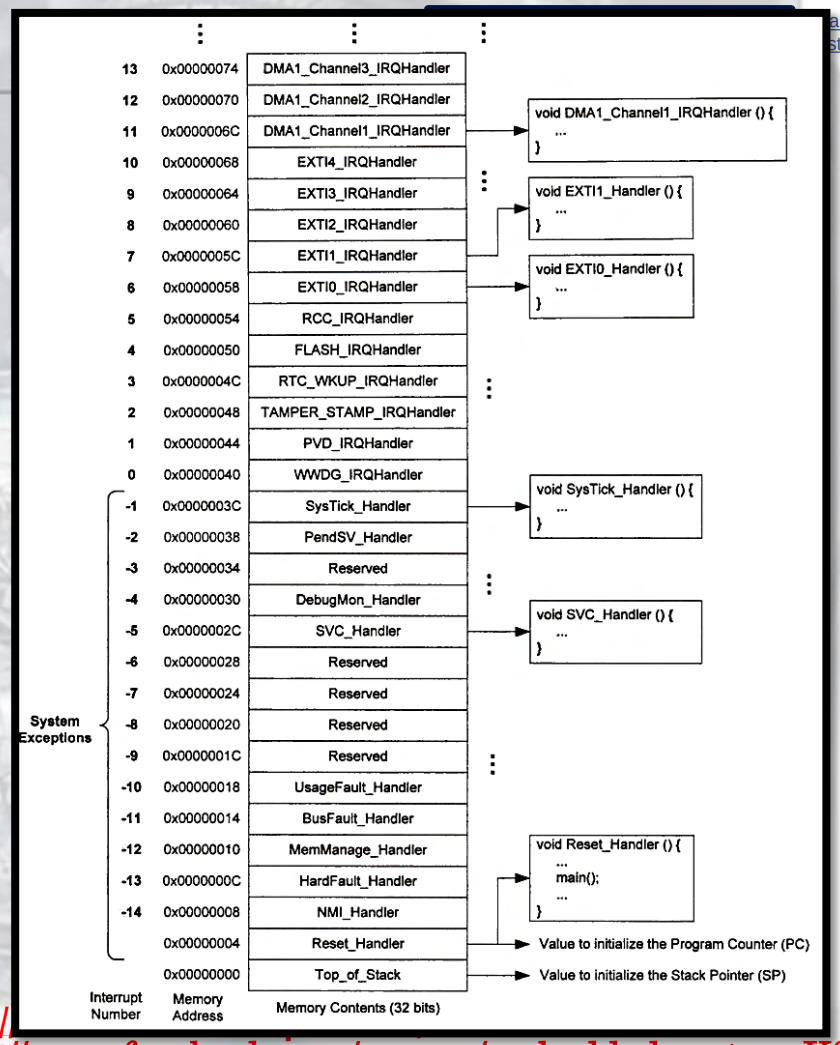
Irq Source= UART0

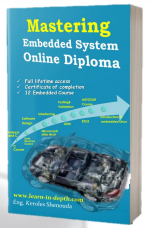
Table 2-9. Interrupts

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
	1	0x0000.0044	GPIO Port B
	2	0x0000.0048	GPIO Port C
	3	0x0000.004C	GPIO Port D
	4	0x0000.0050	GPIO Port E
21	5	0x0000.0054	UART0
		0x0000.0058	UART1
		0x0000.005C	SSI0
		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
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

IVT-Example interrupt vector Table TiVaC "TM4C1234 SOC"

- For example, the interrupt number of SysTick is -1, the memory address of SysTick_Handler can be founding by reading the word stored at the following address.
  Address of SysTick_Handler = Ox00000004 + 4 x (-1+15) = *Ox0000003C*
- The interrupt number of reset is -15. Thus, the memory address of Reset_Handler is
  Address of **Reset_Handler** = Ox00000004 + 4 x (-15 + 15) = Ox00000004

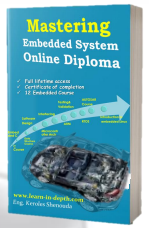




Simply we can Say…..

An interrupt number is used as an index into the interrupt vector table to locate the corresponding interrupt service routine.

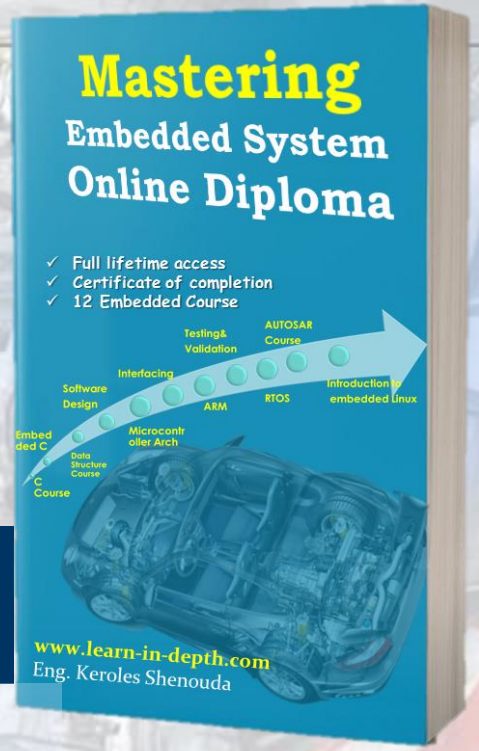
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

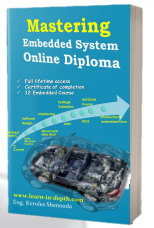
#Be_professional_in_embedded_system

What is the Difference Between Polling And Interrupt ?



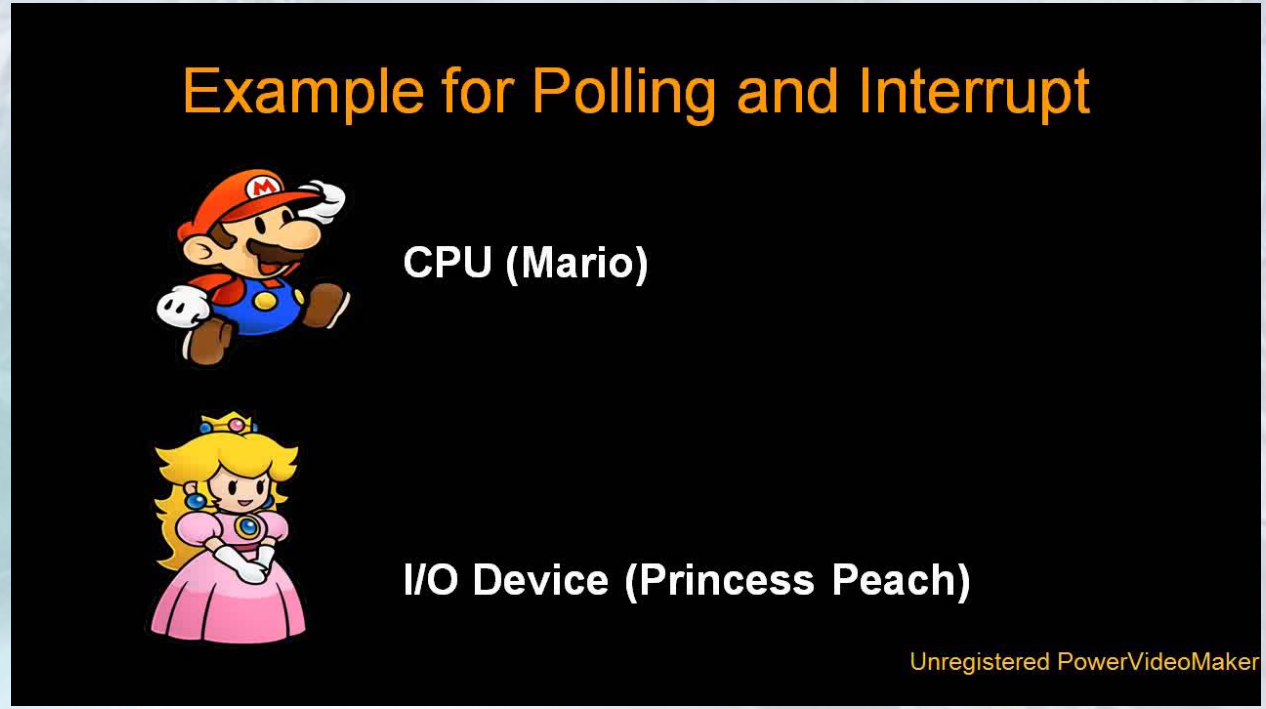
LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

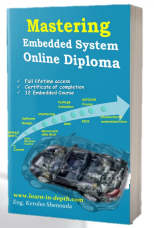


Polling vs. Interrupt (example) Video

- https://www.youtube.com/watch?v=M3nXI_86uLE



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Polling vs. Interrupt

- in **interrupts**, the input from I/O device can arrive at any moment requesting the CPU to process it, **in polling** CPU keeps **asking** the I/O device whether it needs CPU processing.

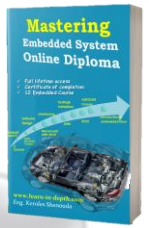
Advantages

Can be efficient if events arrive rapidly

Disadvantages

Takes CPU Time even when no request pending

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

#Be_professional_in embedded_system

Polling

Constantly reading a memory location, in order receive updates of an input value

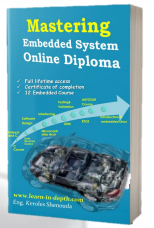
```
#include <avr/io.h>
#include <util/delay.h>
#include <stdint.h>
int main(void){
      unsigned char key_cur,key_pre;
      DDRB=0x0f;
      DDRD=0x00;
      PORTB=0x01;  //turn on the led on the left side as initial state
      PORTD=0xff;  //configurate as pull-up input port
      //waiting for key pressed
      while(1){
            key_pre=key_cur;
            key_cur=PIND&0x1;    //read the key state
            _delay_ms(20); // deltet key jitter
            if(key_cur==0&&key_pre==1) {
            if(PORTB==0x8)
                        PORTB=0x01;
            else
                        PORTB=PORTB<<1;
            }
            //execute another tasks here.
            }
}
```

Polling

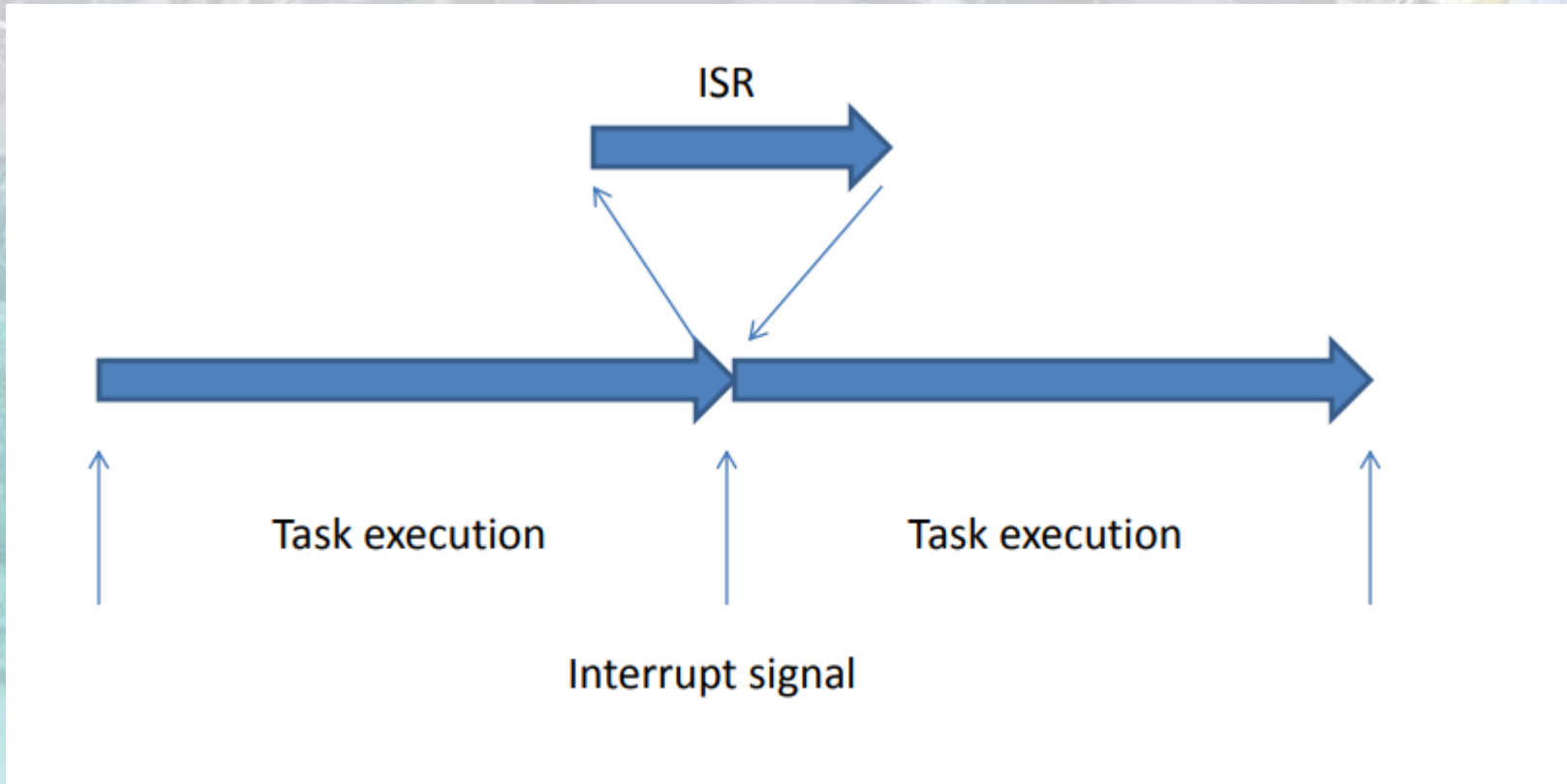
What happens if task need to run for 1 Sec

Miss some action of key pressed……

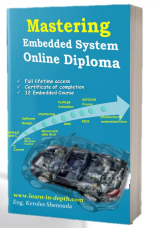
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Interrupt



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Polling Vs. Interrupt

- Polling
  - Ties down the CPU

```
while (true)
{
   if(PIND.2 == 0)
    //do something;
}
```

- Interrupt
  - Efficient CPU use
  - Has priority
  - Can be masked

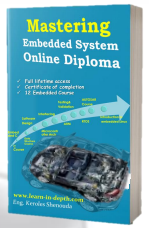
```
void main( )
{
   Do your common task
}

ISR(){ whenever PIND.2 is 0
   then do something}
```

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Polling vs. Interrupt

BASIS FOR COMPARISON	INTERRUPT	POLLING
Basic	Device notify CPU that it needs CPU attention.	CPU constantly checks device status whether it needs CPU's attention.
Mechanism	An interrupt is a hardware mechanism.	Polling is a Protocol.
Servicing	Interrupt handler services the Device.	CPU services the device.
Indication	Interrupt-request line indicates that device needs servicing.	Command-ready bit indicates the device needs servicing.
CPU	CPU is disturbed only when a device needs servicing, which saves CPU cycles.	CPU has to wait and check whether a device needs servicing which wastes lots of CPU cycles.
Occurrence	An interrupt can occur at any time.	CPU polls the devices at regular interval.
Efficiency	Interrupt becomes inefficient when devices keep on interrupting the CPU repeatedly. (Interrupt Overload)	Can be efficient if events arrive rapidly
Example	Let the bell ring then open the door to check who has come.	Constantly keep on opening the door to check whether anybody has come.

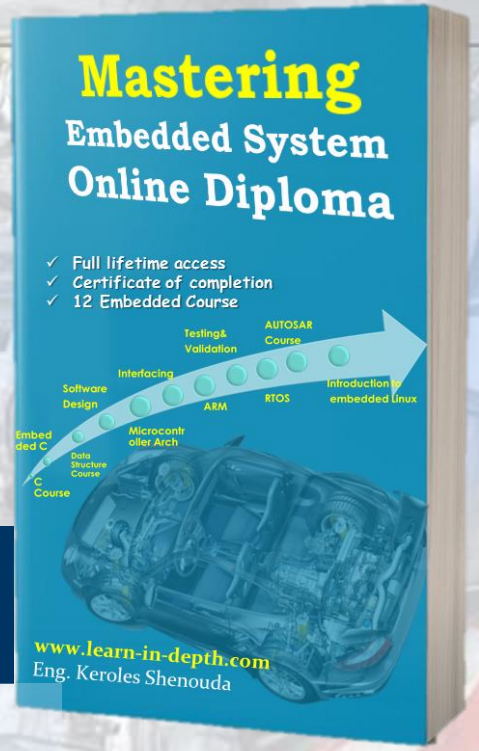
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

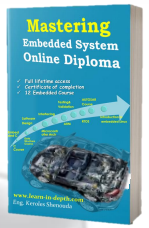
#Be_professional_in_embedded_system

Instruction Cycle with Interrupts



LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

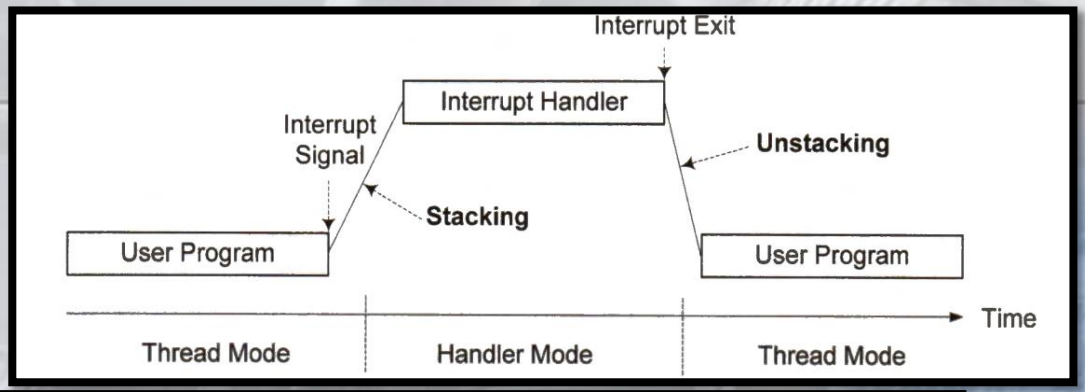


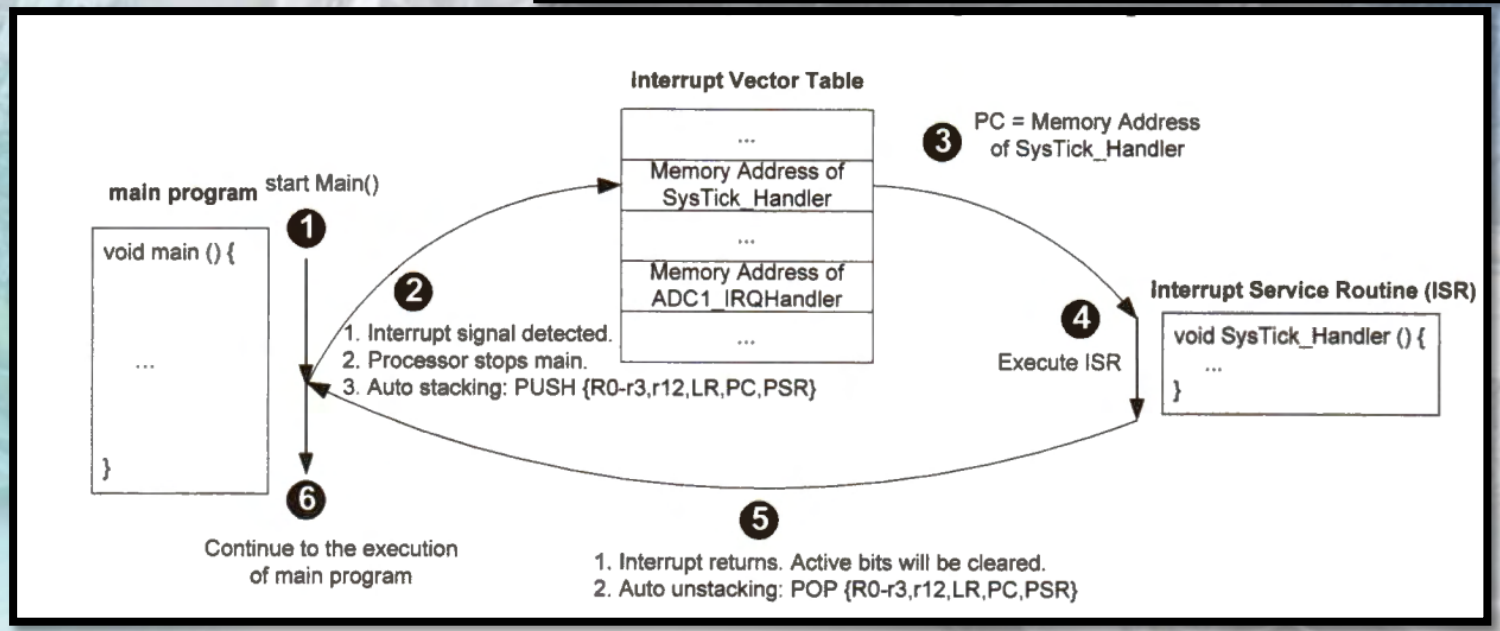
Steps taken in servicing an interrupt

1. The microcontroller **completes** the execution of the current instruction, **clears** the **interrupt status bit** and **stores the address of the next instruction** that should have been executed (the content of the PC) on the stack.
2. **The interrupt vector** of the triggered interrupt is then **loaded** in the **PC** and the microcontroller starts execution from that point up until is reaches a **RETI** instruction.
3. Upon the execution of the **RETI** instruction the address that was stored on the stack in **step 1** is **reloaded** in the **PC** .
4. The microcontroller then start **executing** instructions from that point. That is the point that it **left off when the interrupt was triggered**.

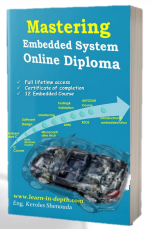
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Instruction Cycle State Diagram, with Interrupts

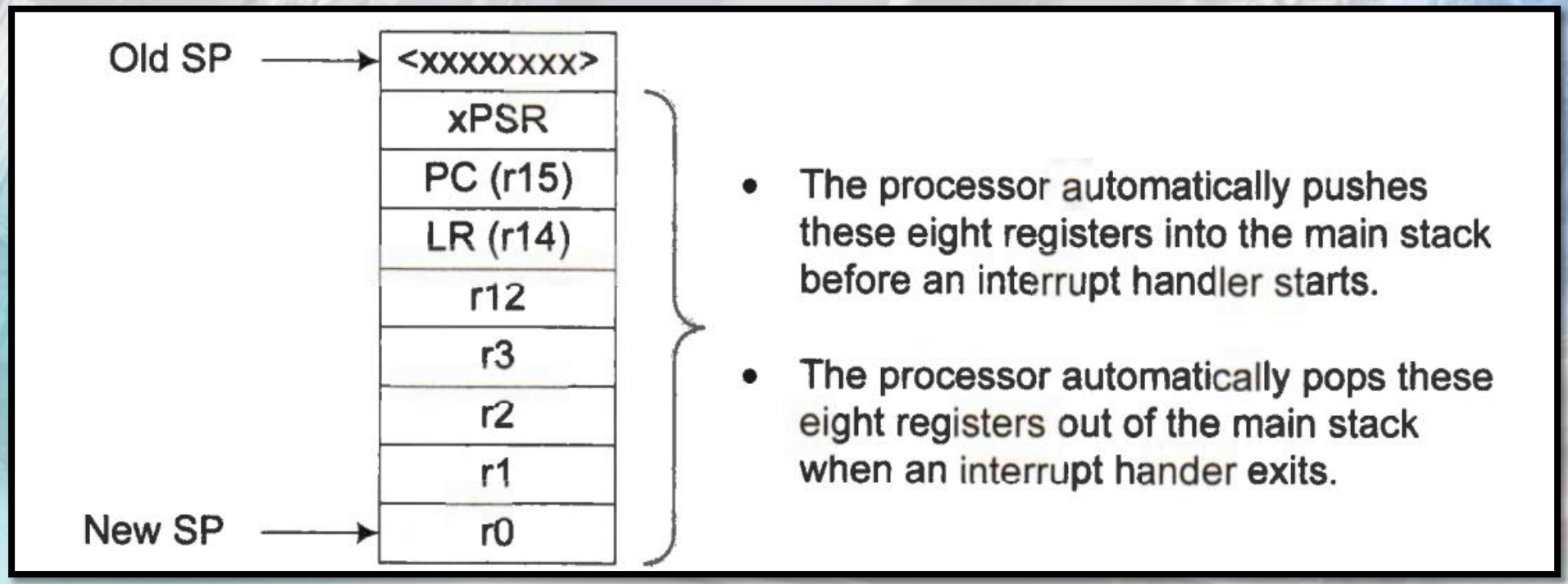




https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

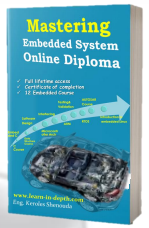


Stacking and unstacking Arm CortexM



- The processor automatically pushes these eight registers into the main stack before an interrupt handler starts.
- The processor automatically pops these eight registers out of the main stack when an interrupt hander exits.

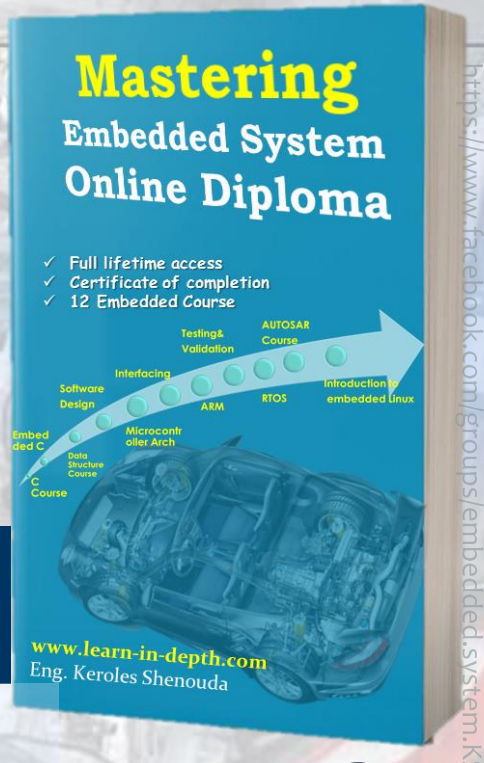
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

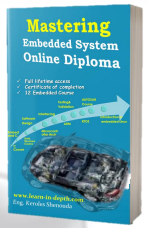
#Be_professional_in_embedded_system

Deep Dive in Interrupt Processing



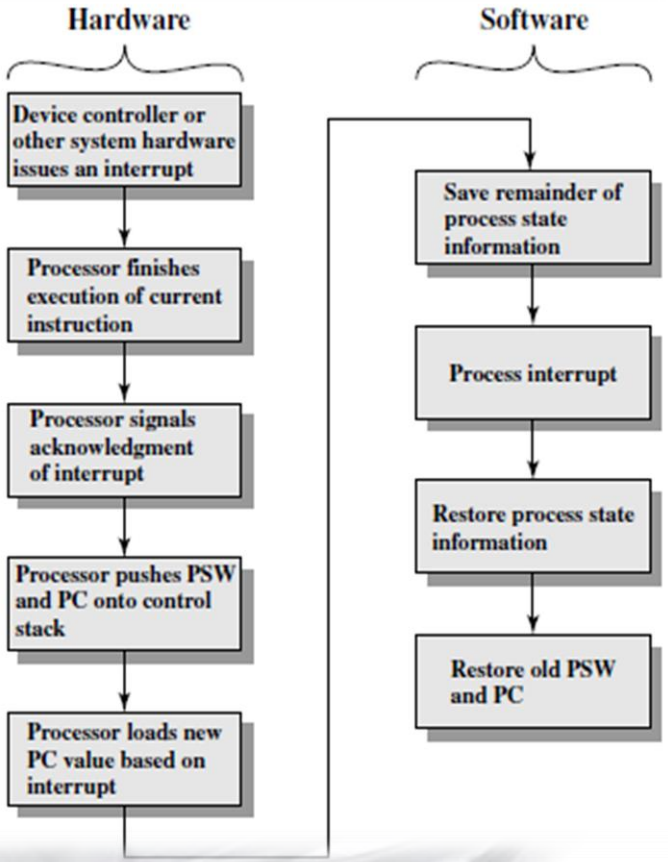
LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Interrupt Processing

1. The **device** issues an interrupt signal to the **processor**.
2. The ***processor*** ***finishes*** execution of the **current instruction** before responding
to the interrupt
3. The ***processor*** tests for an interrupt, determines that there is one, and **sends an acknowledgment signal to the device that issued the interrupt**.
The **acknowledgment** allows the device **to remove its interrupt signal**.
The ***processor*** now needs to prepare to transfer control to the **interrupt routine**. To begin, it needs to save information needed to resume the current program at
the point of interrupt. The minimum information required is (Switch Context)
   (a) the status of the processor, which is contained in a register called the program status word (PSW), and
   (b) (b) the location of the next instruction to be executed. These can be pushed onto the system control stack



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

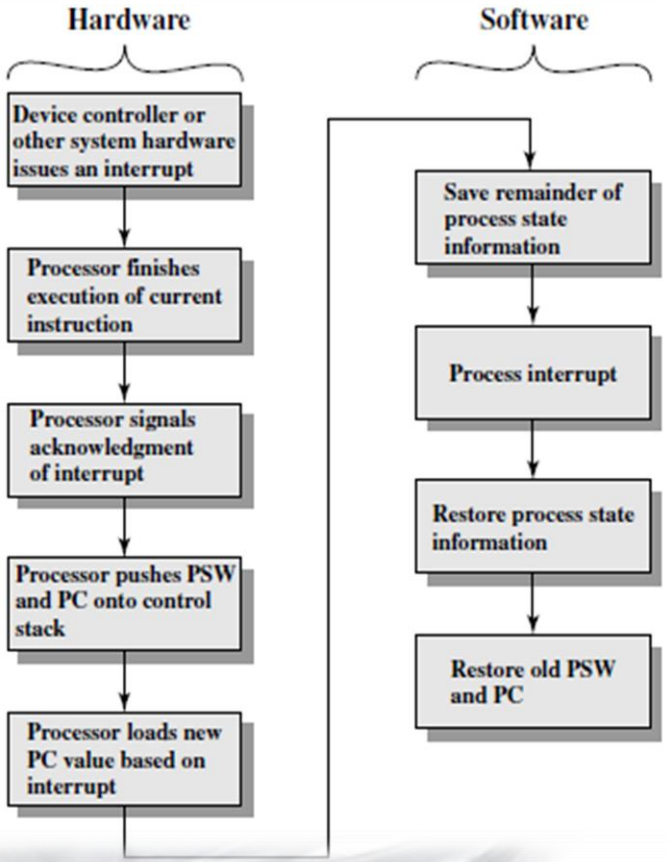
Interrupt Processing

(4) ***The processor*** now loads the ***program counter*** with **the entry location of the interrupt-handling program** **that will respond to this interrupt**. Depending on the computer architecture and operating system design, there may be a single program;

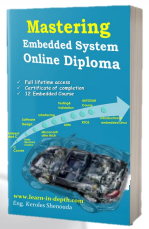
one program for each type of interrupt; or **one program for each device and each type of interrupt**.

If there is more than one interrupt-handling routine, the ***processor*** must **determine which one to invoke**. This information may have been included in the original interrupt signal, or the processor may have to issue a request to the device that issued the interrupt to get a response that contains the needed information.

Once the ***program counter*** has been loaded, the ***processor*** proceeds to the next instruction cycle, which begins with an instruction fetch.

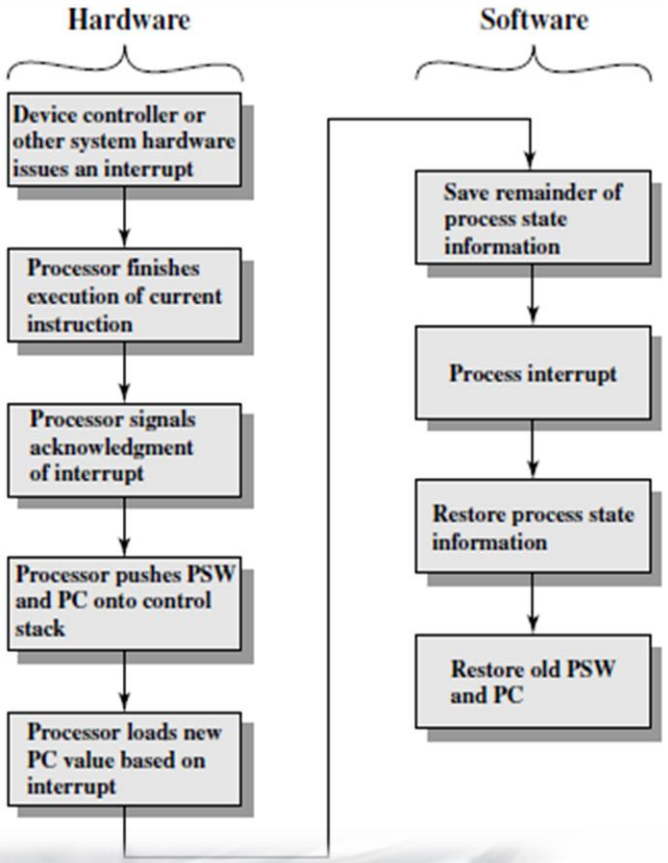


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



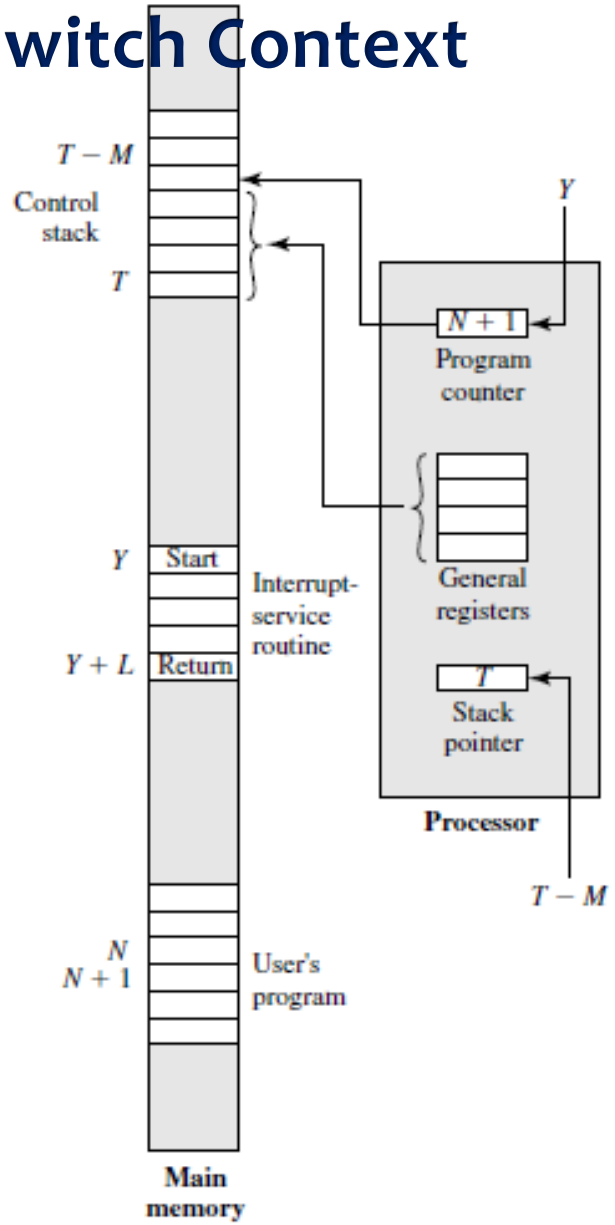
Interrupt Processing

- **(5)** When interrupt processing is complete, the saved register values are retrieved from the stack and restored to the registers (Context restore)
- **(6)** The final act is to restore the PSW and program counter values from the stack. As a result, the next instruction to be executed will be from the previously interrupted program.



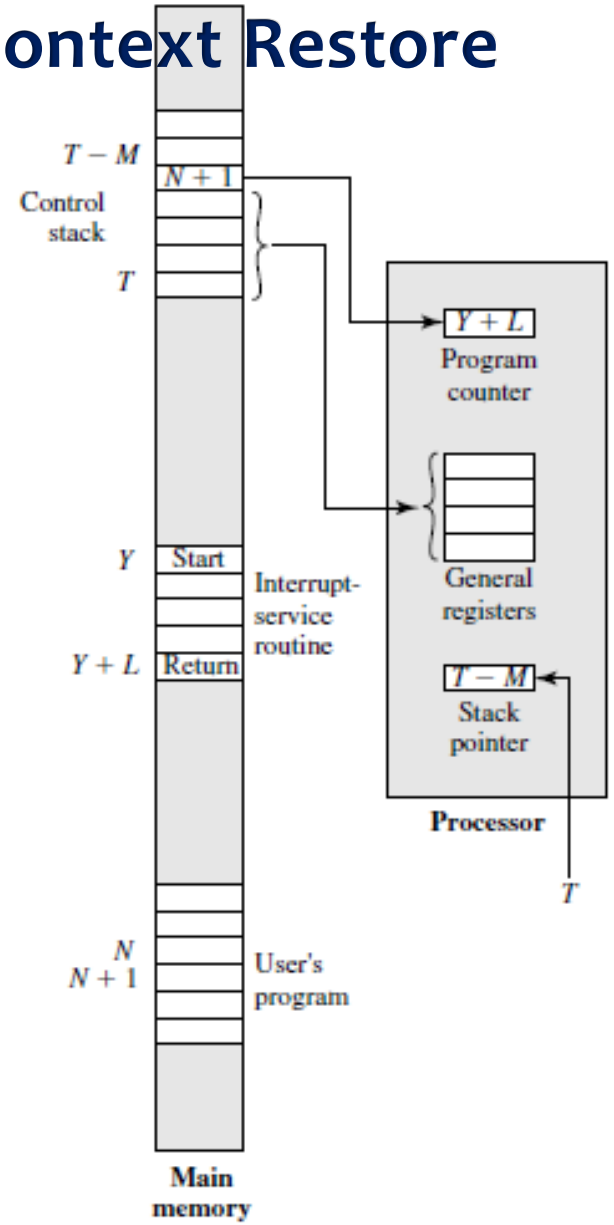
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Switch Context

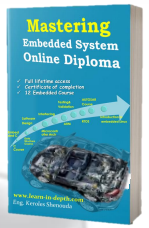


(a) Interrupt occurs after instruction at location N

Context Restore

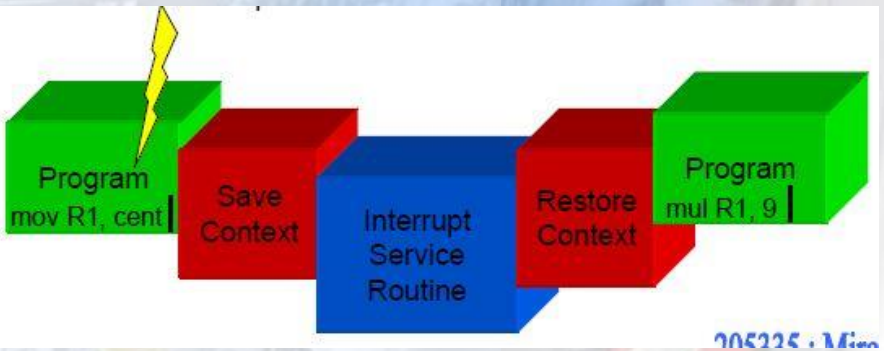


(b) Return from interrupt

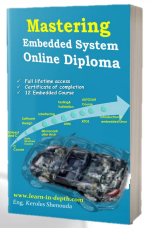


#LEARN_IN DEPTH

#Be_professional_in embedded_system



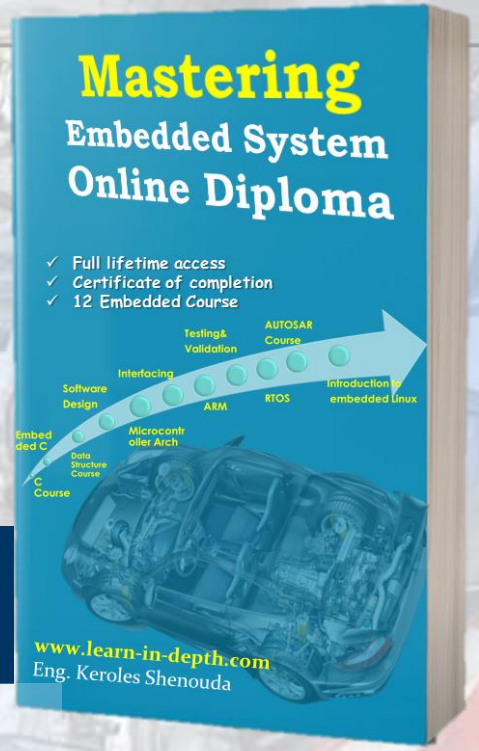
ps://www.learn-in-depth.com/
ps://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

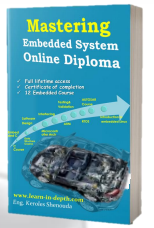
#Be_professional_in_embedded_system

What is interrupt latency?



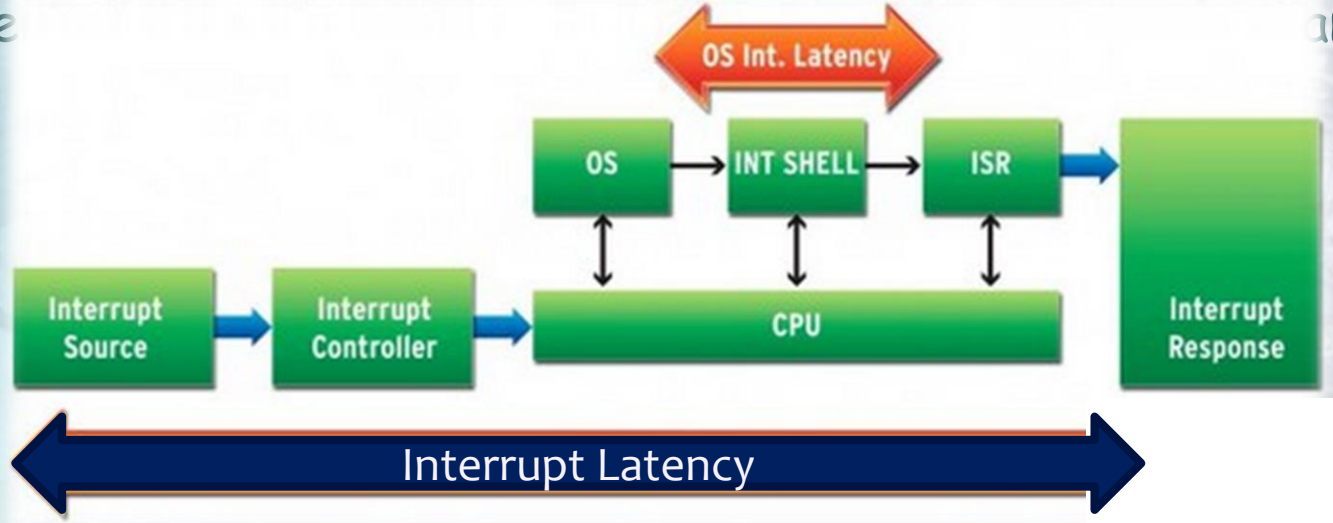
LEARN-IN-DEPTH
Be professional in embedded system

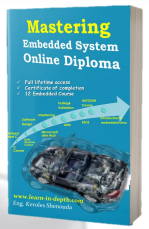
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



What is interrupt latency?

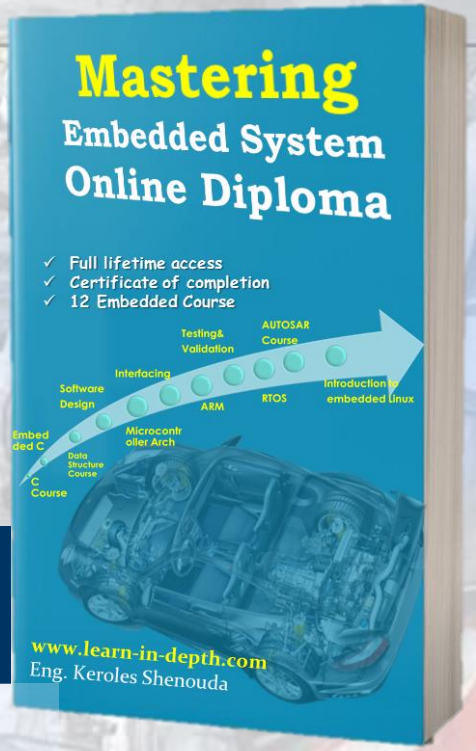
- Interrupt latency refers primarily to the **software interrupt handling latencies**. In other words, the amount of time that elapses from the time that an external **interrupt arrives** at the processor until the time that the **interrupt processing begins**.
- One of the most important aspects of kernel real-time performance is the ability to se mount of time.





#LEARN_IN DEPTH

#Be_professional_in_embedded_system



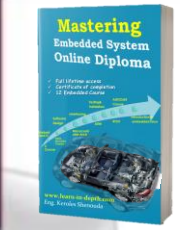
Interrupt Example on Atmga32

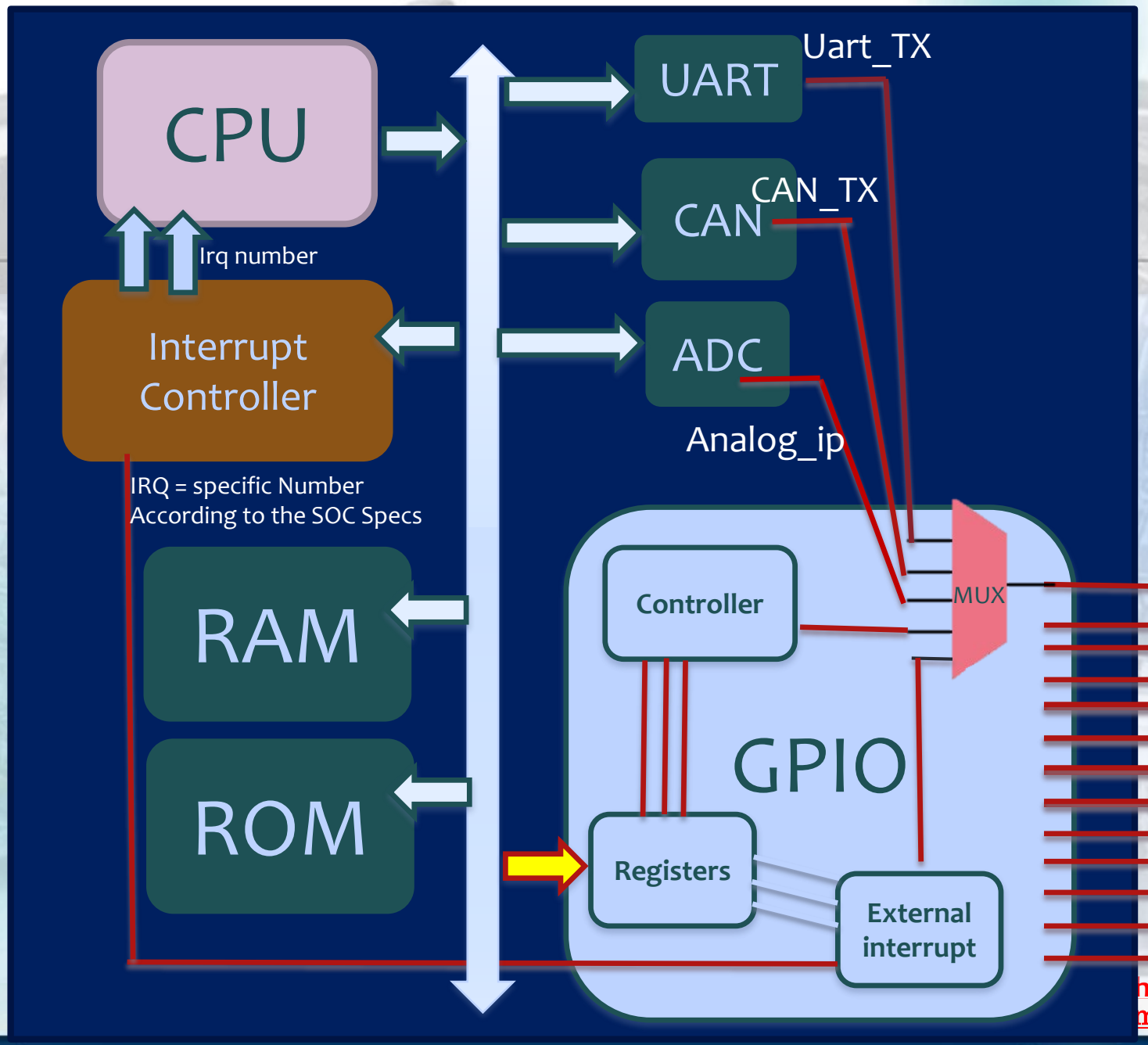
EXTERNAL INTERRUPT

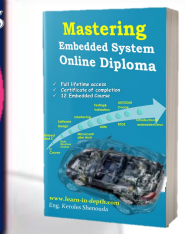
LEARN-IN-DEPTH

Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

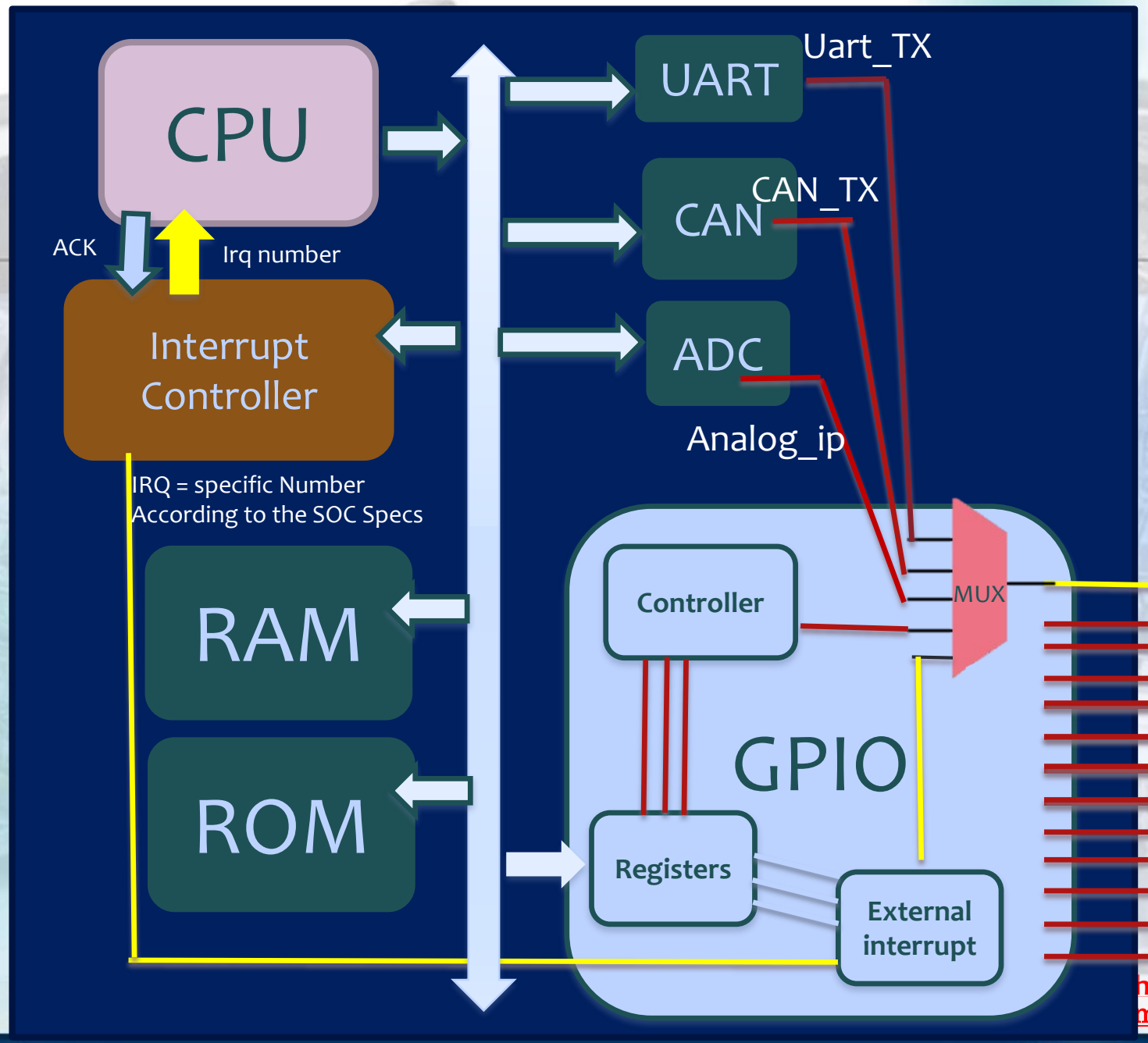






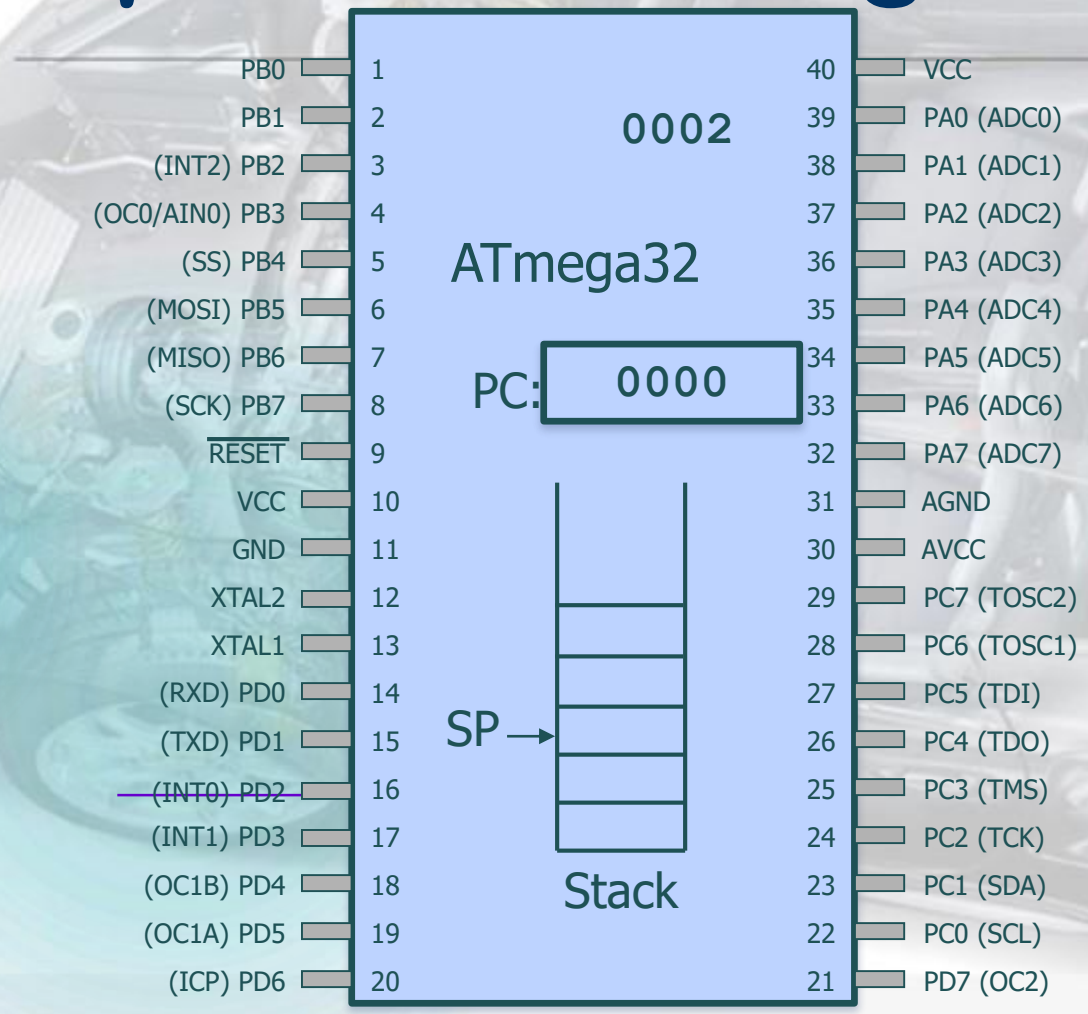
#LEARN_IN DEPTH

#Be_professional_in embedded_system

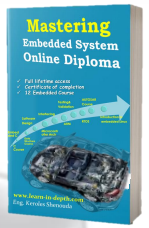


h.com/
m/groups/embedded.system.KS/

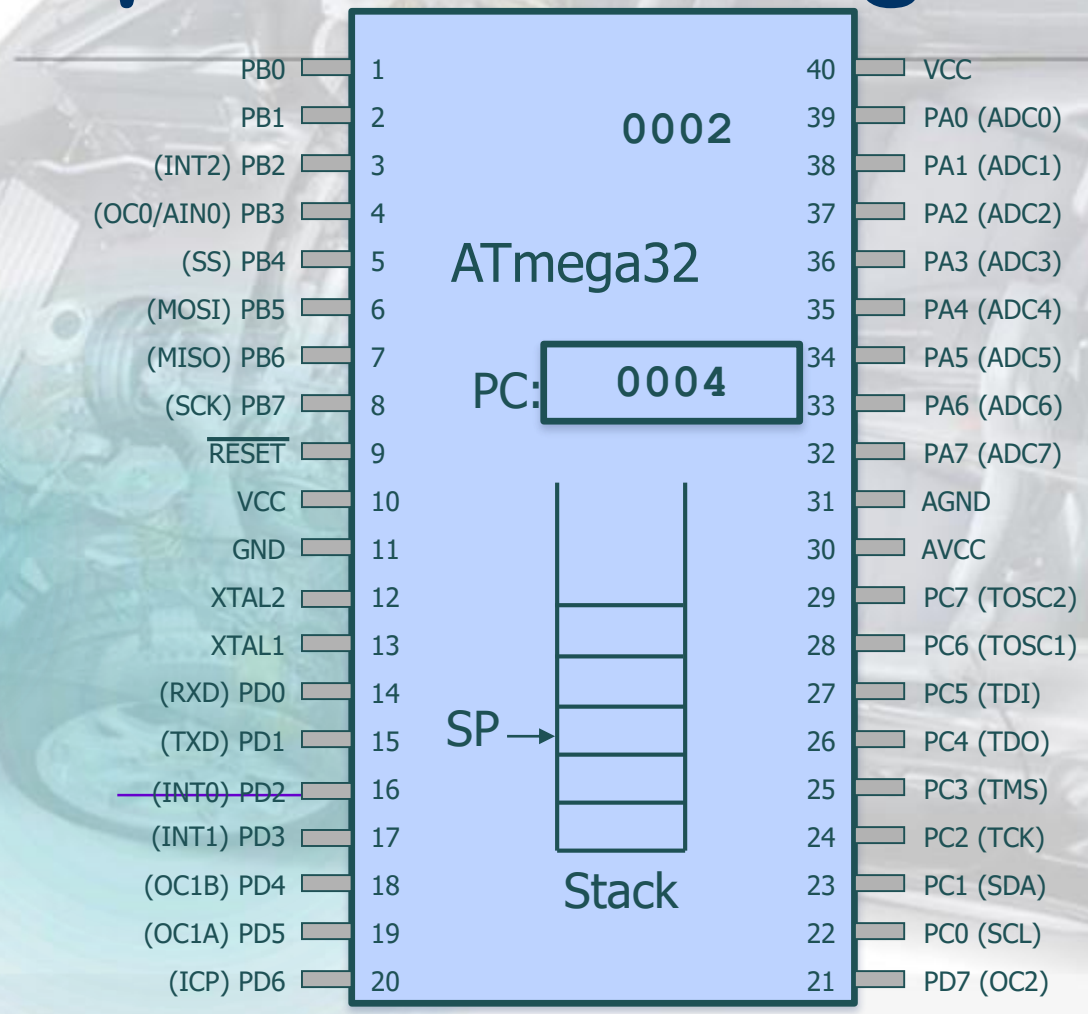
Steps in executing an interrupt



Address	Code
	.INCLUDE "M32DEF.INC"
	.ORG 0 ;location for reset
0000	JMP MAIN
0002	.ORG 0x02 ;location for external INT0
0002	JMP EX0_ISR
0004	MAIN: LDI R20,HIGH(RAMEND)
0005	OUT SPH,R20
0006	LDI R20,LOW(RAMEND)
0007	OUT SPL,R20
0008	SBI DDRC,3 ;PC.3 = output
0009	SBI PORTD,2 ;pull-up activated
000A	LDI R20,1<<INT0 ;Enable INT0
000B	OUT GICR,R20
000C	SEI ;Set I (Enable Interrupts)
000D	LDI R30, 3
000E	LDI R31, 4
000F	ADD R30, R31
0010	HERE:JMP HERE
0012	EX0_ISR:IN R21,PORTC
0013	LDI R22,0x08
0014	EOR R21,R22
0015	OUT PORTC,R21
0016	RETI

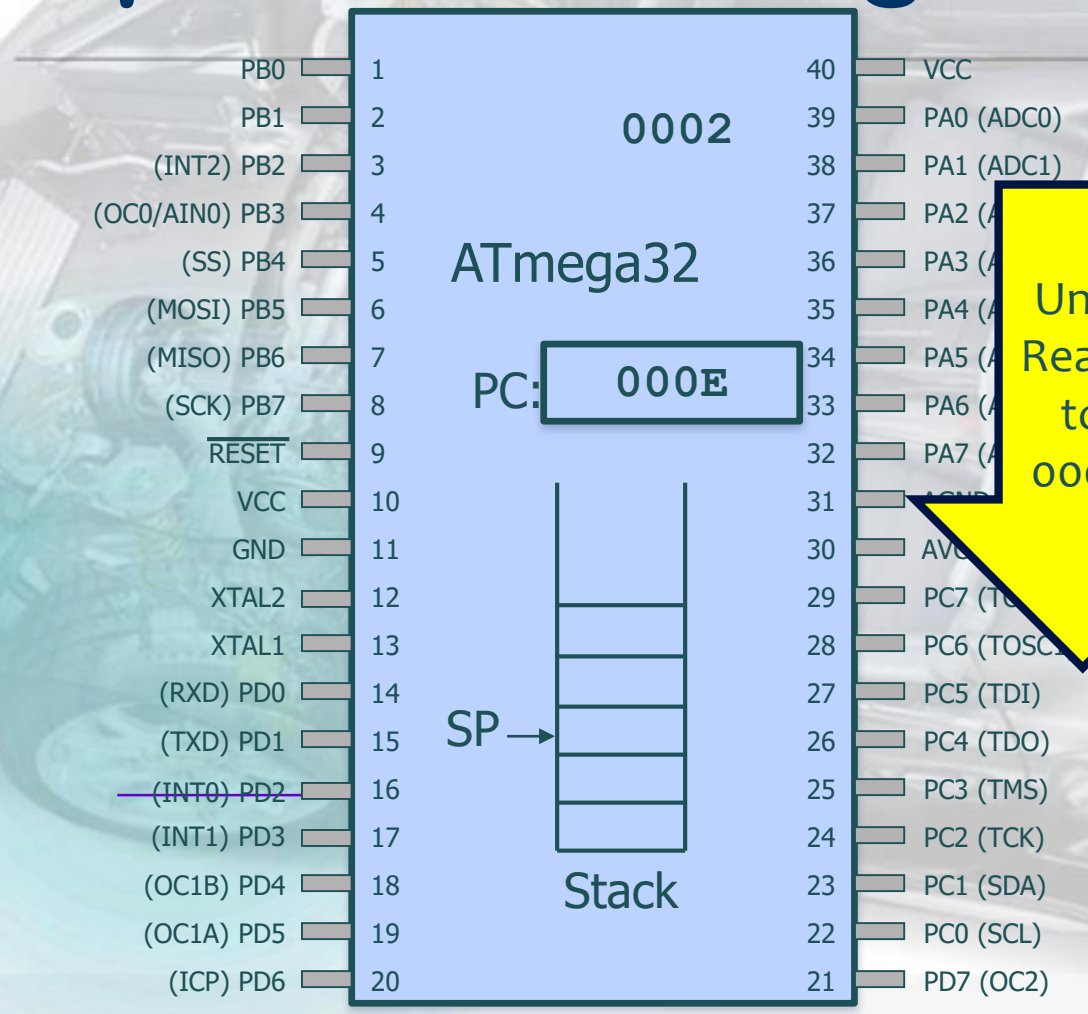


Steps in executing an interrupt



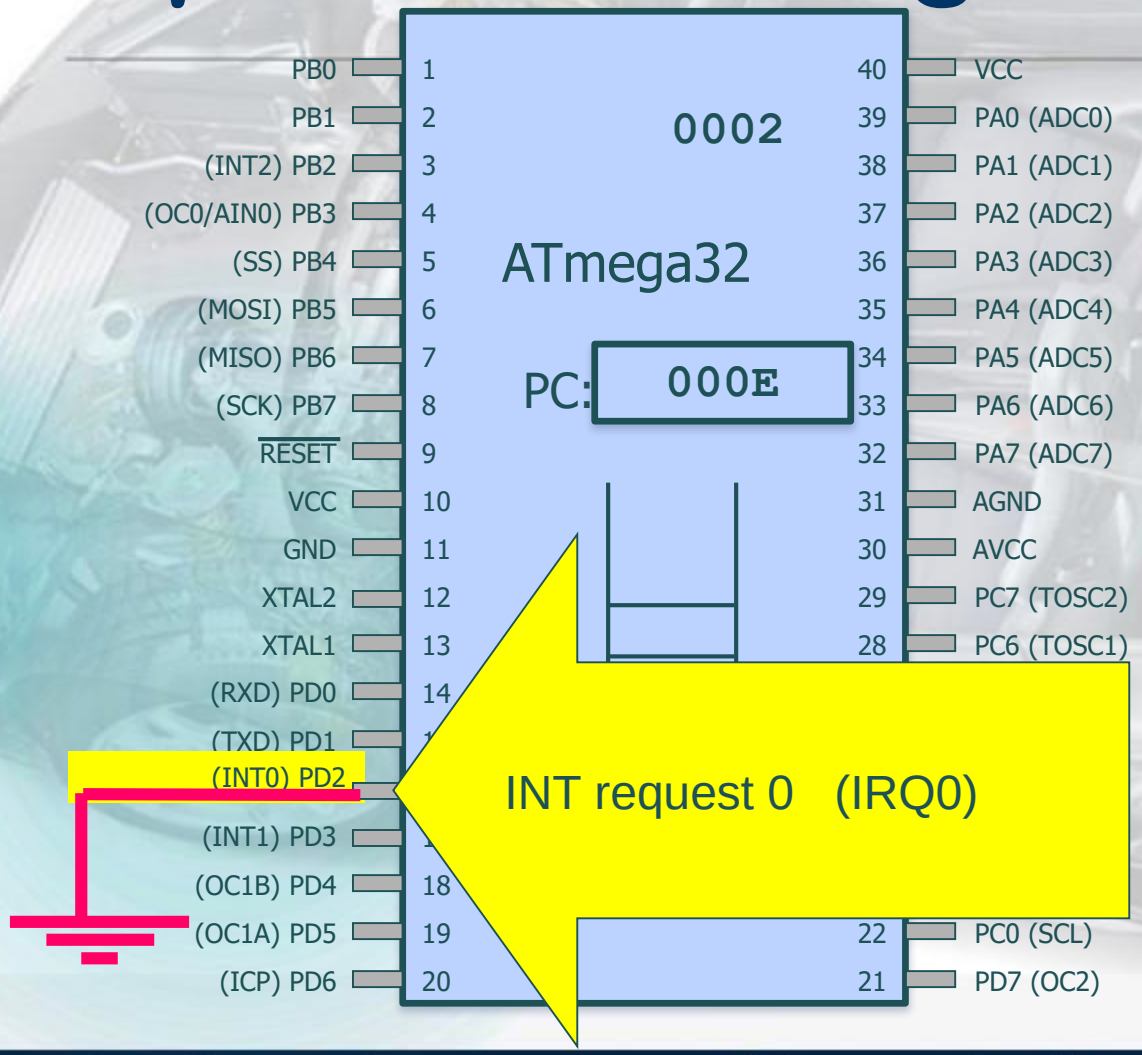
Address	Code
	.INCLUDE "M32DEF.INC"
	.ORG 0 ;location for reset
0000	JMP MAIN
0002	.ORG 0x02 ;location for external INT0
0002	JMP EX0_ISR
0004	MAIN: LDI R20,HIGH(RAMEND)
0005	OUT SPH,R20
0006	LDI R20,LOW(RAMEND)
0007	OUT SPL,R20
0008	SBI DDRC,3 ;PC.3 = output
0009	SBI PORTD,2 ;pull-up activated
000A	LDI R20,1<<INT0 ;Enable INT0
000B	OUT GICR,R20
000C	SEI ;Set I (Enable Interrupts)
000D	LDI R30, 3
000E	LDI R31, 4
000F	ADD R30, R31
0010	HERE:JMP HERE
0012	EX0_ISR:IN R21,PORTC
0013	LDI R22,0x08
0014	EOR R21,R22
0015	OUT PORTC,R21
0016	RETI

Steps in executing an interrupt



Address	Code
	.INCLUDE "M32DEF.INC"
	.ORG 0 ;location for reset
0000	JMP MAIN
0002	.ORG 0x02 ;location for external INT0
0002	JMP EX0_ISR
0004	MAIN: LDI R20,HIGH(RAMEND)
0005	OUT SPH,R20
0006	LDI R20,LOW(RAMEND)
0007	OUT SPL,R20
0008	SBI DDRC,3 ;PC.3 = output
0009	SBI PORTD,2 ;pull-up activated
000A	LDI R20,1<<INT0 ;Enable INT0
000B	OUT GICR,R20
000C	SEI ;Set I (Enable Interrupts)
000D	LDI R30, 3
000E	LDI R31, 4
000F	ADD R30, R31
0010	HERE:JMP HERE
0012	EX0_ISR:IN R21,PORTC
0013	LDI R22,0x08
0014	EOR R21,R22
0015	OUT PORTC,R21
0016	RETI

Steps in executing an interrupt



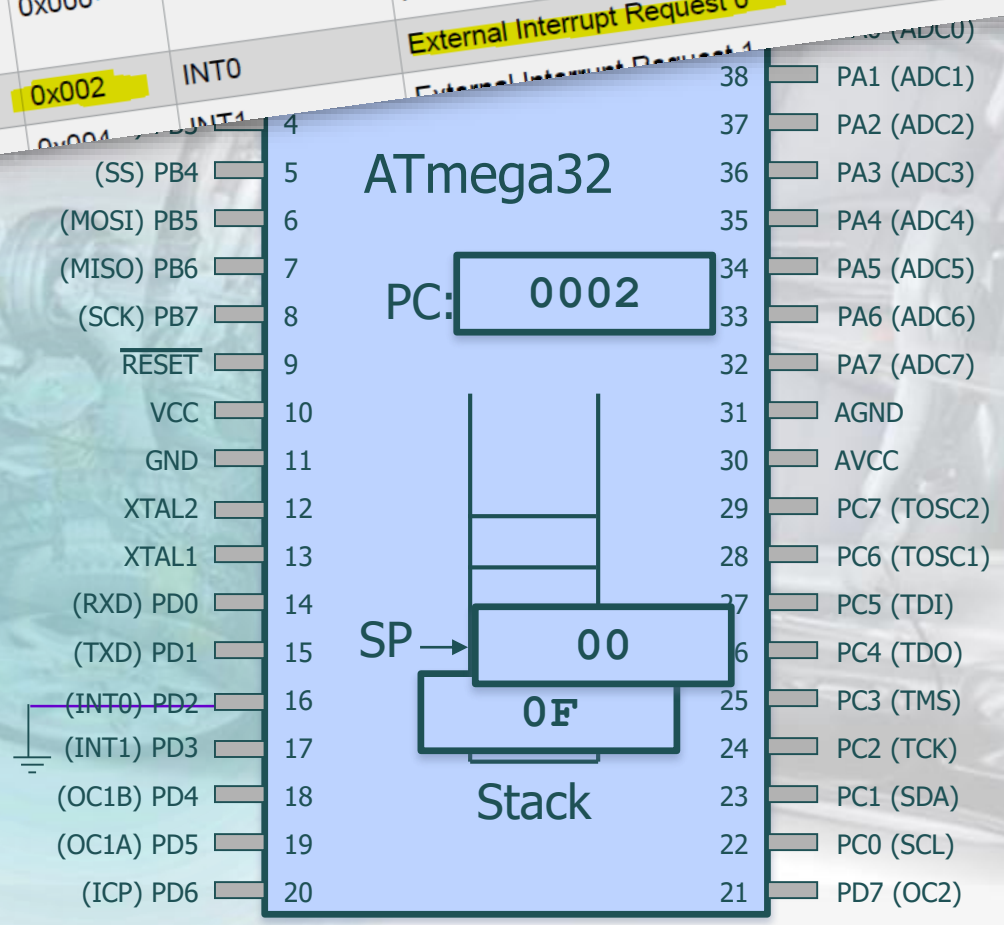
Address	Code
	.INCLUDE "M32DEF.INC"
	.ORG 0 ;location for reset
0000	JMP MAIN
0002	.ORG 0x02 ;location for external INT0
0002	JMP EX0_ISR
0004	MAIN: LDI R20,HIGH(RAMEND)
0005	OUT SPH,R20
0006	LDI R20,LOW(RAMEND)
0007	OUT SPL,R20
0008	SBI DDRC,3 ;PC.3 = output
0009	SBI PORTD,2 ;pull-up activated
000A	LDI R20,1<<INT0 ;Enable INT0
000B	OUT GICR,R20
000C	SEI ;Set I (Enable Interrupts)
000D	LDI R30, 3
000E	LDI R31, 4
000F	ADD R30, R31
0010	HERE:JMP HERE
0012	EX0_ISR:IN R21,PORTC
0013	LDI R22,0x08
0014	EOR R21,R22
0015	OUT PORTC,R21
0016	RETI

Interrupt

Interrupt Vectors in ATmega32A

Table 15-1 Reset and Interrupt Vectors

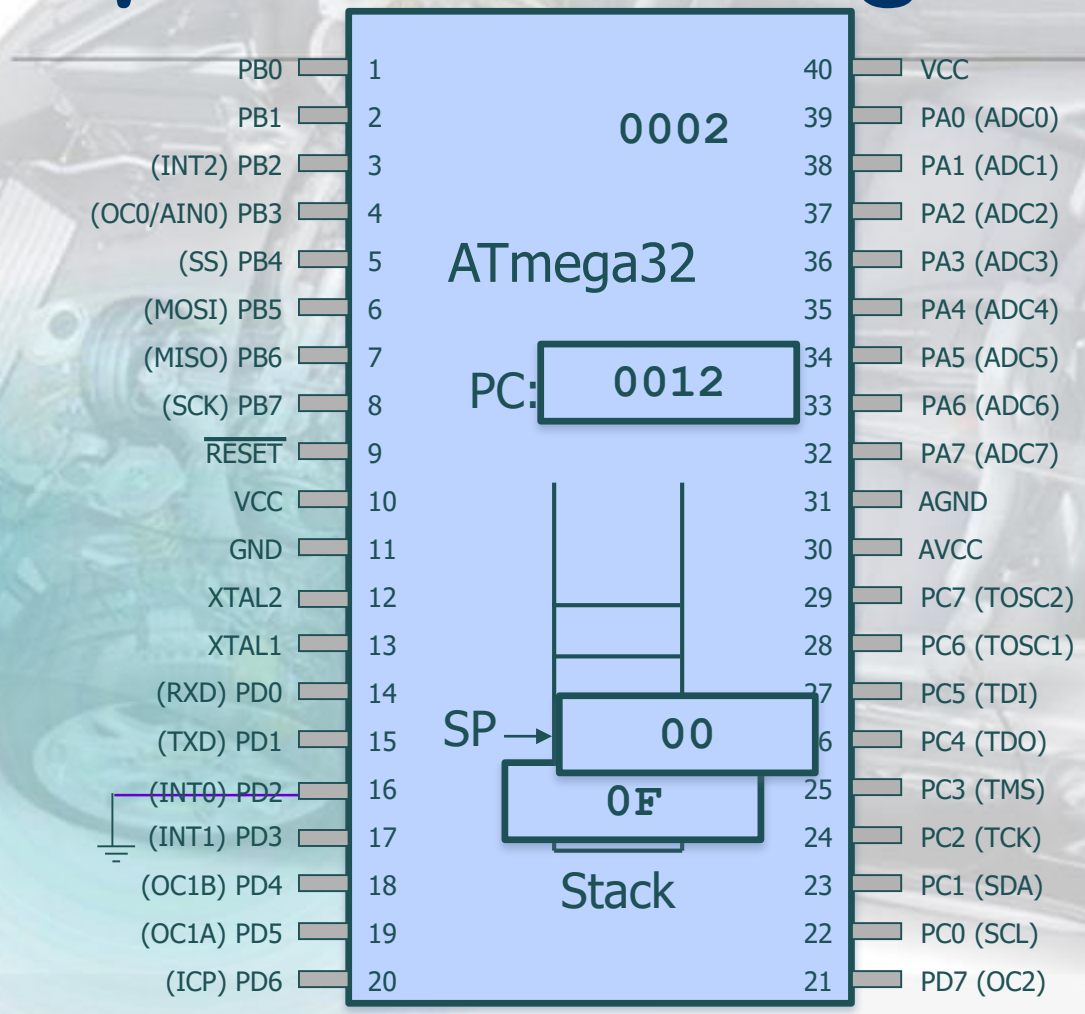
Vector No.	Program Address(2)	Source	Interrupt Definition
1	0x000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	0x002	INT0	External Interrupt Request 0



Vector Address from IVT "Interrupt Vector Table"

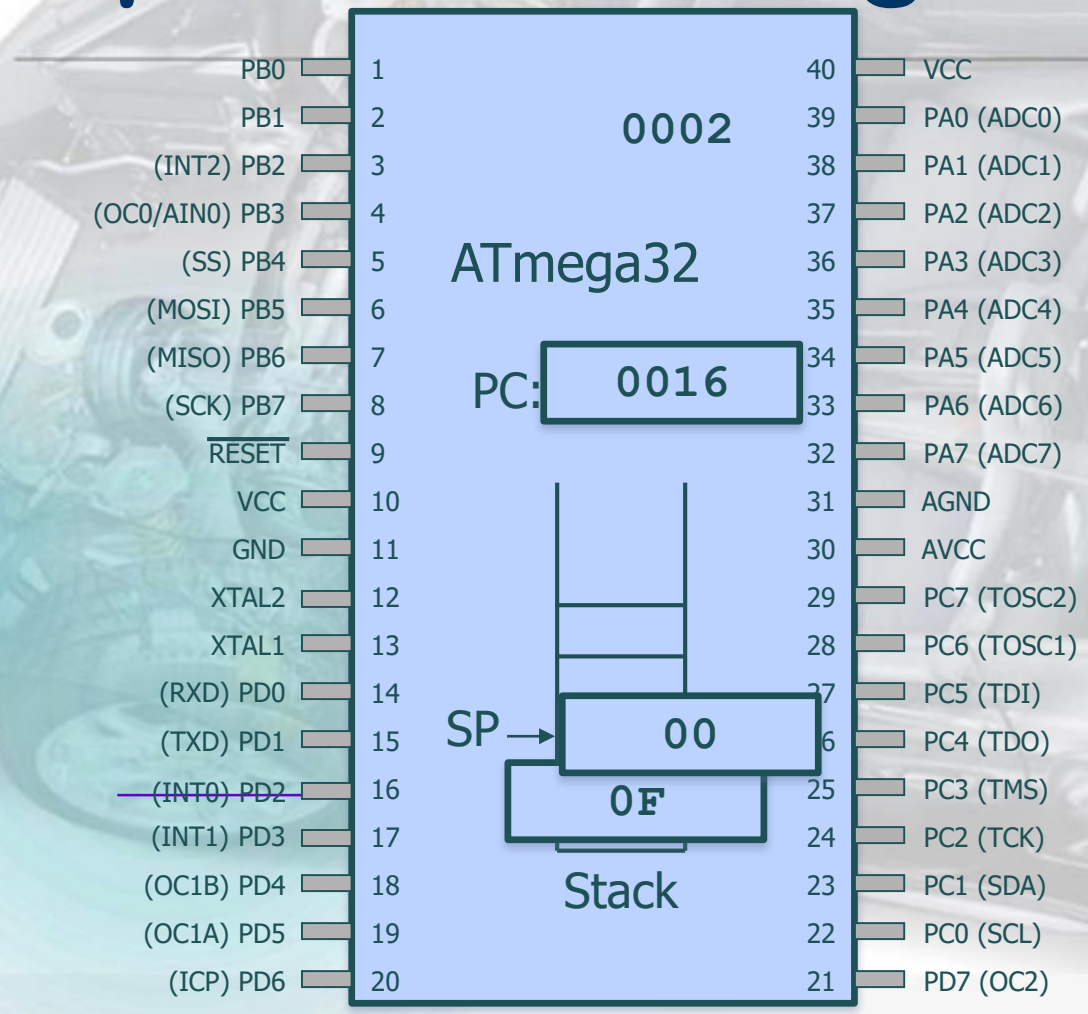
```
                    Code
                    .INCLUDE "M32DEF.INC"
                    .ORG 0          ;location for reset
0000                    JMP     MAIN
0002                .ORG 0x02       ;location for external INT0
0002                    JMP     EX0_ISR
0004                MAIN: LDI   R20,HIGH(RAMEND)
0005                    OUT     SPH,R20
0006                    LDI     R20,LOW(RAMEND)
0007                    OUT     SPL,R20
0008                    SBI     DDRC,3      ;PC.3 = output
0009                    SBI     PORTD,2     ;pull-up activated
000A                    LDI     R20,1<<INT0 ;Enable INT0
000B                    OUT     GICR,R20
000C                    SEI             ;Set I (Enable Interrupts)
000D                    LDI     R30, 3
000E                    LDI     R31, 4
000F                    ADD     R30, R31
0010                HERE:JMP    HERE

0012                EX0_ISR:IN  R21,PORTC
0013                    LDI     R22,0x08
0014                    EOR     R21,R22
0015                    OUT     PORTC,R21
0016                    RETI
```

Steps in executing an interrupt



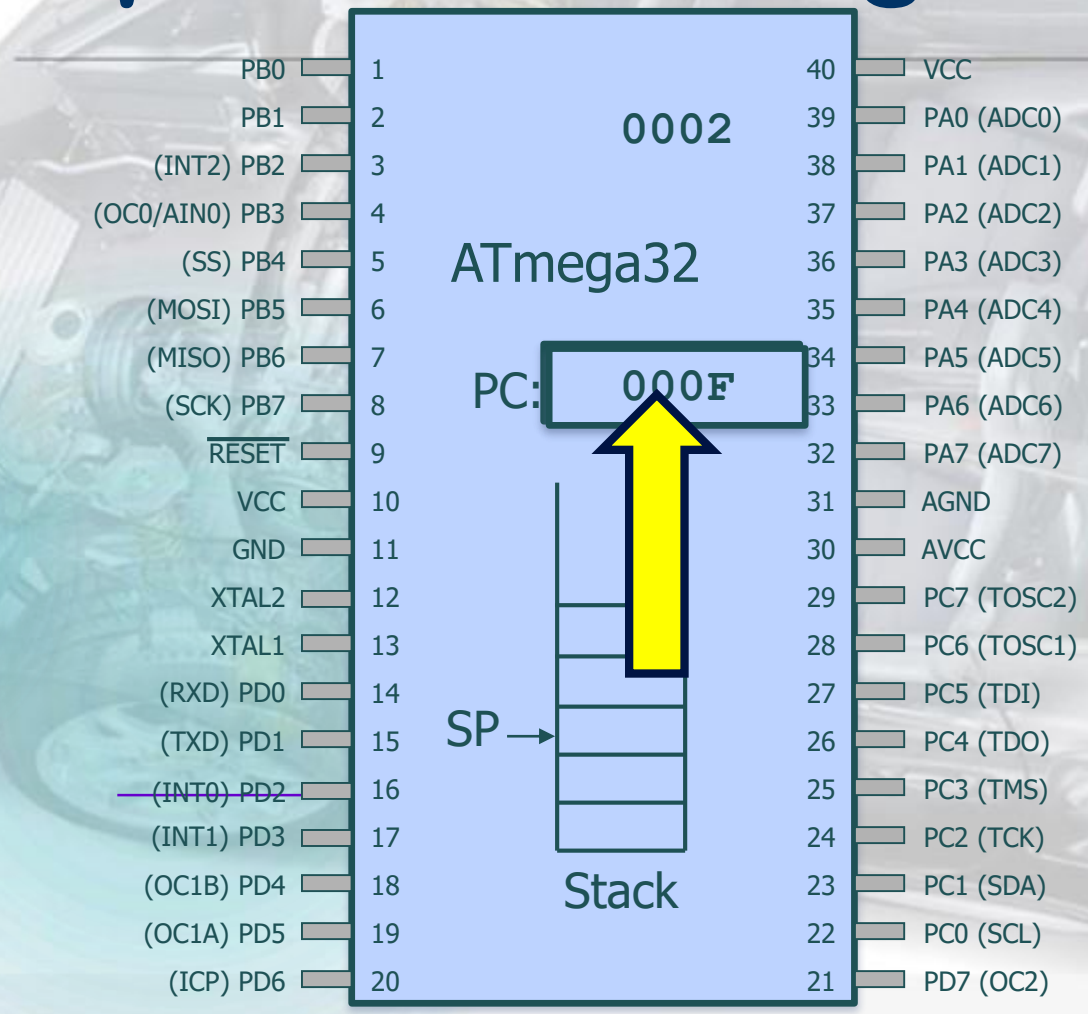
Address	Code
	.INCLUDE "M32DEF.INC"
	.ORG 0 ;location for reset
0000	JMP MAIN
0002	.ORG 0x02 ;location for external INT0
0002	JMP EX0_ISR
0004	MAIN: LDI R20,HIGH(RAMEND)
0005	OUT SPH,R20
0006	LDI R20,LOW(RAMEND)
0007	OUT SPL,R20
0008	SBI DDRC,3 ;PC.3 = output
0009	SBI PORTD,2 ;pull-up activated
000A	LDI R20,1<<INT0 ;Enable INT0
000B	OUT GICR,R20
000C	SEI ;Set I (Enable Interrupts)
000D	LDI R30, 3
000E	LDI R31, 4
000F	ADD R30, R31
0010	HERE:JMP HERE
0012	EX0_ISR:IN R21,PORTC
0013	LDI R22,0x08
0014	EOR R21,R22
0015	OUT PORTC,R21
0016	RETI

Steps in executing an interrupt



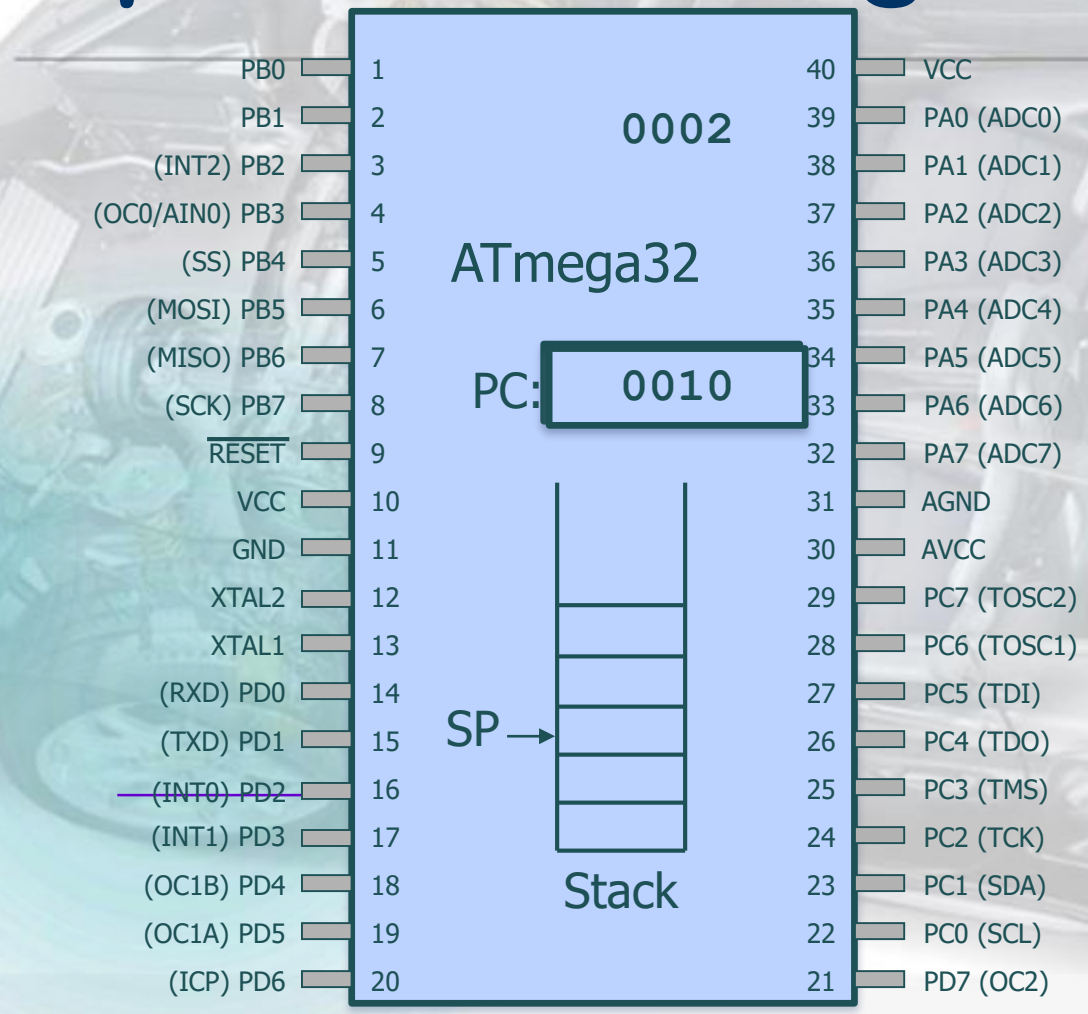
Address	Code
	.INCLUDE "M32DEF.INC"
	.ORG 0 ;location for reset
0000	JMP MAIN
0002	.ORG 0x02 ;location for external INT0
0002	JMP EX0_ISR
0004	MAIN: LDI R20,HIGH(RAMEND)
0005	OUT SPH,R20
0006	LDI R20,LOW(RAMEND)
0007	OUT SPL,R20
0008	SBI DDRC,3 ;PC.3 = output
0009	SBI PORTD,2 ;pull-up activated
000A	LDI R20,1<<INT0 ;Enable INT0
000B	OUT GICR,R20
000C	SEI ;Set I (Enable Interrupts)
000D	LDI R30, 3
000E	LDI R31, 4
000F	ADD R30, R31
0010	HERE:JMP HERE
0012	EX0_ISR:IN R21,PORTC
0013	LDI R22,0x08
0014	EOR R21,R22
0015	OUT PORTC,R21
0016	RETI

Steps in executing an interrupt

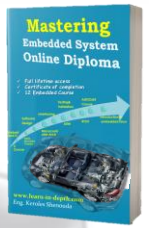


Address	Code
	.INCLUDE "M32DEF.INC"
	.ORG 0 ;location for reset
0000	JMP MAIN
0002	.ORG 0x02 ;location for external INT0
0002	JMP EX0_ISR
0004	MAIN: LDI R20,HIGH(RAMEND)
0005	OUT SPH,R20
0006	LDI R20,LOW(RAMEND)
0007	OUT SPL,R20
0008	SBI DDRC,3 ;PC.3 = output
0009	SBI PORTD,2 ;pull-up activated
000A	LDI R20,1<<INT0 ;Enable INT0
000B	OUT GICR,R20
000C	SEI ;Set I (Enable Interrupts)
000D	LDI R30, 3
000E	LDI R31, 4
000F	ADD R30, R31
0010	HERE:JMP HERE
0012	EX0_ISR:IN R21,PORTC
0013	LDI R22,0x08
0014	EOR R21,R22
0015	OUT PORTC,R21
0016	RETI

Steps in executing an interrupt



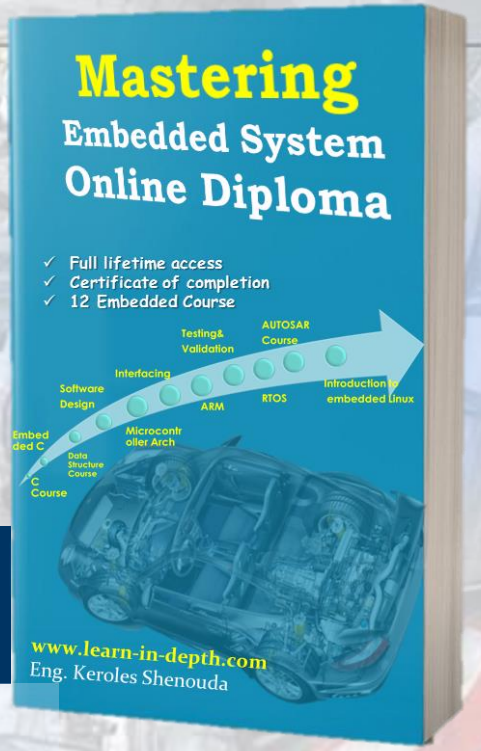
Address	Code
	.INCLUDE "M32DEF.INC"
	.ORG 0 ;location for reset
0000	JMP MAIN
0002	.ORG 0x02 ;location for external INT0
0002	JMP EX0_ISR
0004	MAIN: LDI R20,HIGH(RAMEND)
0005	OUT SPH,R20
0006	LDI R20,LOW(RAMEND)
0007	OUT SPL,R20
0008	SBI DDRC,3 ;PC.3 = output
0009	SBI PORTD,2 ;pull-up activated
000A	LDI R20,1<<INT0 ;Enable INT0
000B	OUT GICR,R20
000C	SEI ;Set I (Enable Interrupts)
000D	LDI R30, 3
000E	LDI R31, 4
000F	ADD R30, R31
0010	HERE:JMP HERE
0012	EX0_ISR:IN R21,PORTC
0013	LDI R22,0x08
0014	EOR R21,R22
0015	OUT PORTC,R21
0016	RETI



#LEARN_IN DEPTH

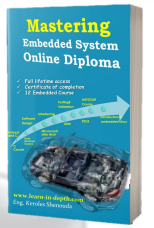
#Be_professional_in_embedded_system

Sequential interrupt processing VS Nested interrupt processing

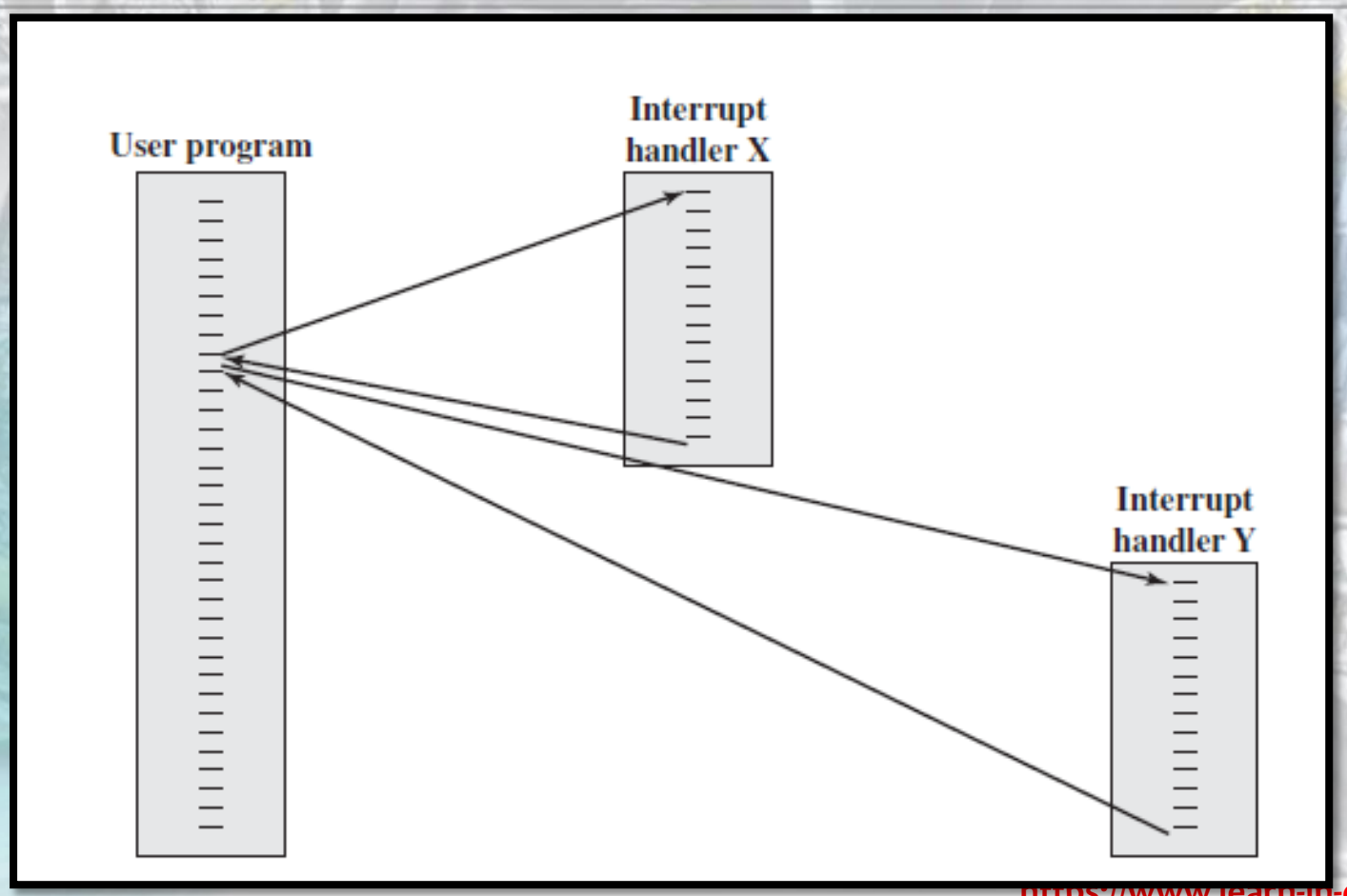


LEARN-IN-DEPTH
Be professional in embedded system

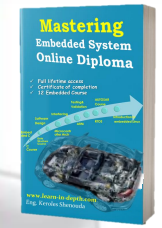
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



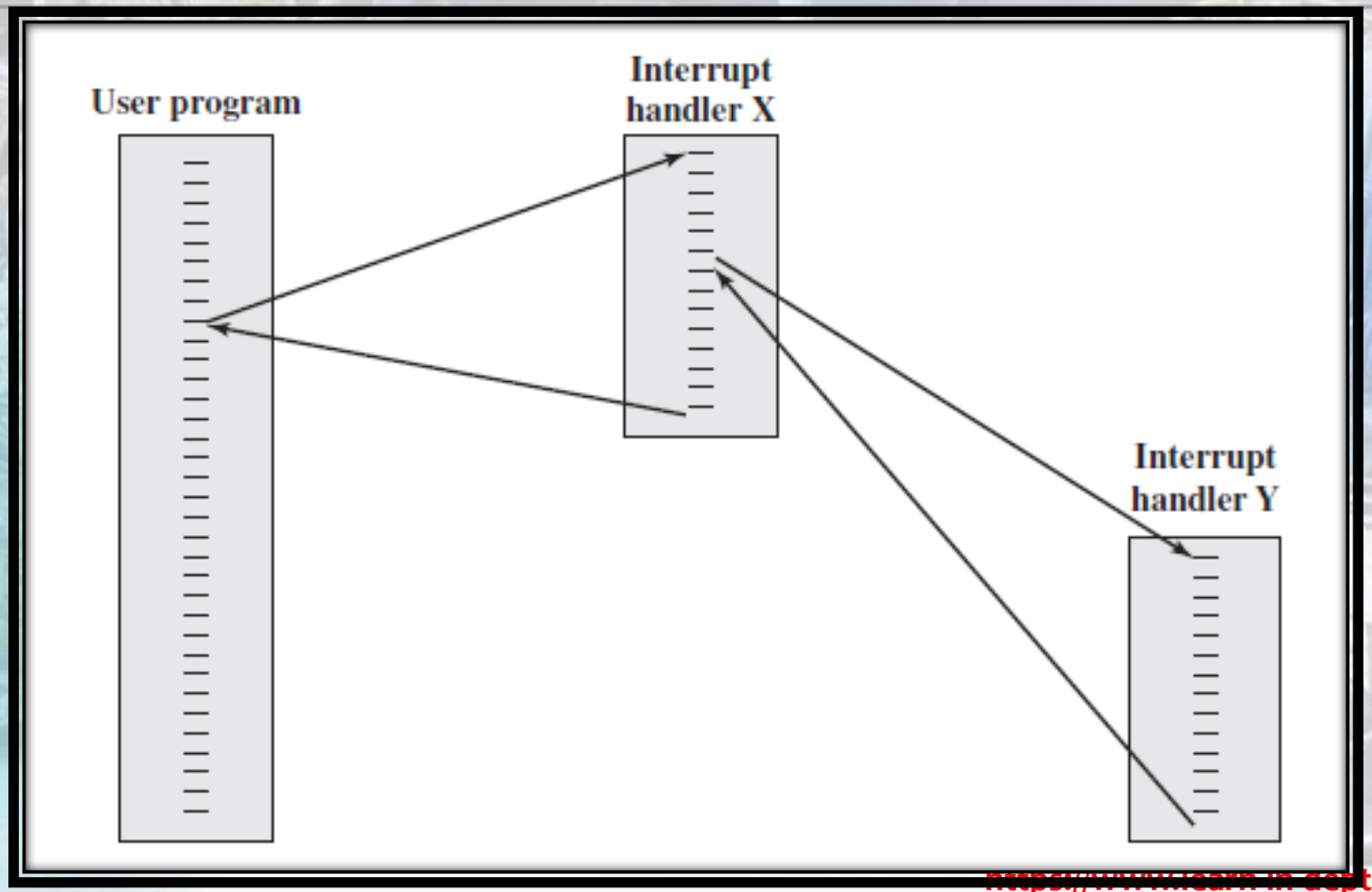
Sequential interrupt processing



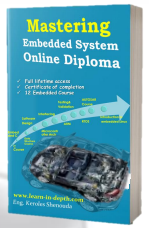
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Nested interrupt processing

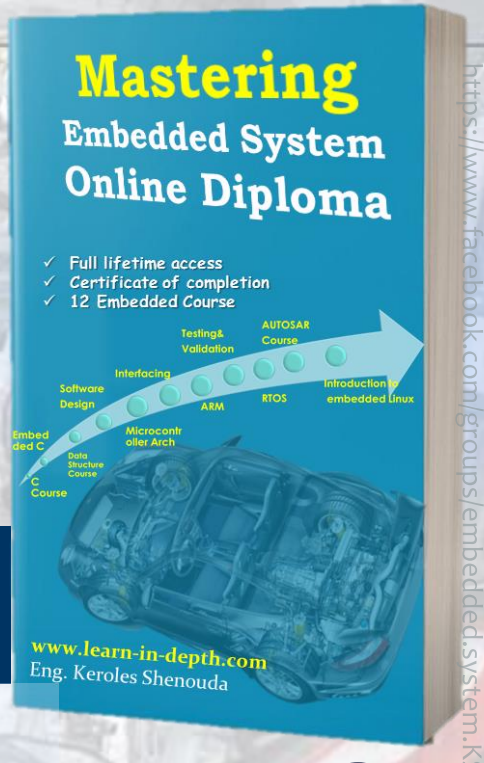


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

#Be_professional_in_embedded_system



Lab1: Write a Vector Section in *.s or *.c and add it in *ld linker script

REVIEW UNIT3 LESSON3/4

LEARN-IN-DEPTH
Be professional in embedded system

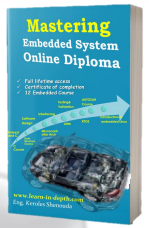
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

What does the Vector Table contain?

- Initial value **of the Stack Pointer**
- The vector table is a section of our flash memory that mostly holds the addresses of various handlers.
- Starting address of the reset handler (Reset handler is the code executed on reset)
- Starting addresses of all other exceptions and interrupts including the NMI handler, Hard fault handler and so on.

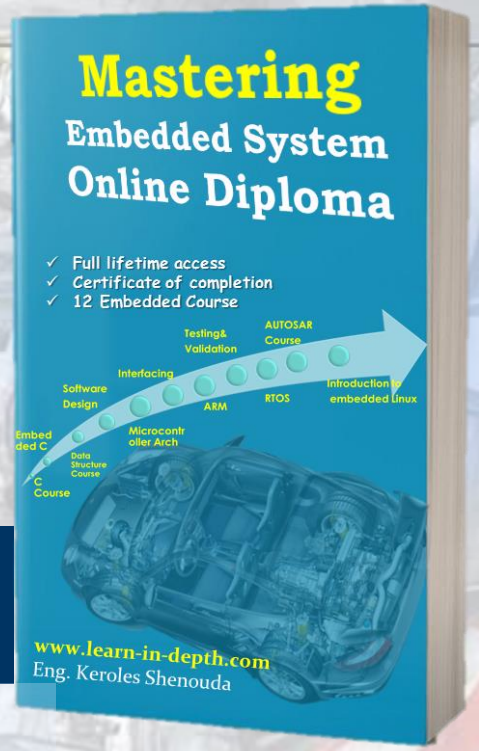
Table 61. Vector table for connectivity line devices

Position	Priority	Type of priority	Acronym	Description	Address
-	-	-	-	Reserved	0x0000_0000
-	-3	fixed	Reset	Reset	0x0000_0004
-	-2	fixed	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.	0x0000_0008
-	-1	fixed	HardFault	All class of fault	0x0000_000C
-	0	settable	MemManage	Memory management	0x0000_0010
-	1	settable	BusFault	Pre-fetch fault, memory access fault	0x0000_0014
-	2	settable	UsageFault	Undefined instruction or illegal state	0x0000_0018
-	-	-	-	Reserved	0x0000_001C - 0x0000_002B
-	3	settable	SVCall	System service call via SWI instruction	0x0000_002C
-	4	settable	Debug Monitor	Debug Monitor	0x0000_0030
-	-	-	-	Reserved	0x0000_0034
-	5	settable	PendSV	Pendable request for system service	0x0000_0038
-	6	settable	SysTick	System tick timer	0x0000_003C
0	7	settable	WWDG	Window Watchdog interrupt	0x0000_0040
1	8	settable	PVD	PVD through EXTI Line detection interrupt	0x0000_0044
2	9	settable	TAMPER	Tamper interrupt	0x0000_0048
3	10	settable	RTC	RTC global interrupt	0x0000_004C
4	11	settable	FLASH	Flash global interrupt	0x0000_0050
5	12	settable	RCC	RCC global interrupt	0x0000_0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060



#LEARN_IN DEPTH

#Be_professional_in_embedded_system



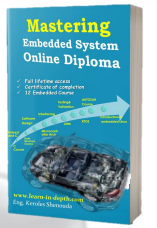
Interrupt types

HW INTERRUPTS

EXCEPTIONS INTERRUPT (TRAPS, FAULTS, ABORTS AND PROGRAMMED EXCEPTIONS)

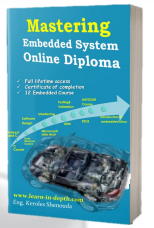
LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Types of Interrupts

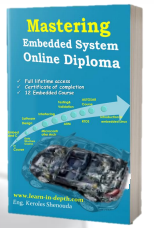
- Asynchronous / Interrupt Hardware
  - From external source, such as I/O device
  - Not related to instruction being executed
- Synchronous (also called *exceptions*)
  - *Processor-detected* exceptions:
    - *Faults* — correctable; offending instruction is *retried*
    - *Traps* — often for debugging; instruction is *not* retried
    - *Aborts* — major error (hardware failure)
  - *Programmed* exceptions:
    - Requests for kernel intervention (software intr/syscalls)



Software Interrupt

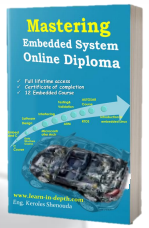
- A **software interrupt** is caused either by **an exceptional condition** or **a special instruction in the instruction set** which causes an interrupt when it is executed by the processor.
- For example, if the processor's arithmetic logic unit runs a command to divide a number by zero, to cause a divide-by-zero exception, thus causing the computer to abandon the calculation or display an error message.
- Software interrupt instructions work similar to subroutine calls.

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



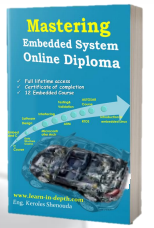
Synchronous (also called *exceptions*) *Faults*

- Instruction would be illegal to execute
- Examples:
  - Writing to a memory segment marked 'read-only'
  - Reading from an unavailable memory segment (on disk)
- Detected *before* incrementing the PC



Traps

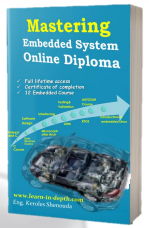
- A CPU might have been programmed to automatically switch control to a 'debugger' program after it has executed an instruction
- That type of situation is known as a 'trap'
- It is activated *after* incrementing the PC



Error Exceptions

- Most error exceptions — divide by zero, invalid operation, illegal memory reference, etc. — translate directly into signals

Asynchronous / Interrupt Hardware

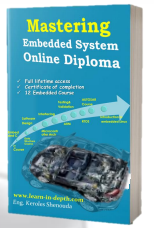


#LEARN_IN DEPTH

#Be_professional_in_embedded_system

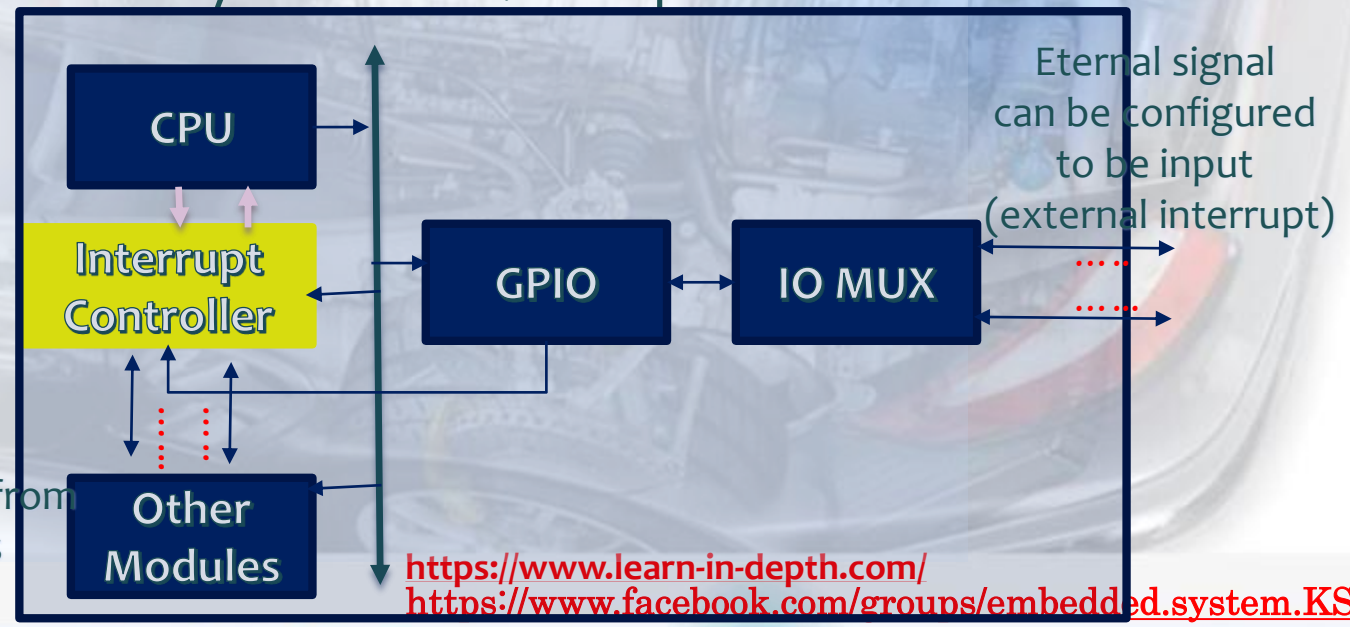
- I/O devices have (unique or shared) *Interrupt Request Lines* (IRQs)
- IRQs are mapped by special hardware to *interrupt vectors*, and passed to the CPU
- This hardware is called a *Programmable Interrupt Controller* (PIC)

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

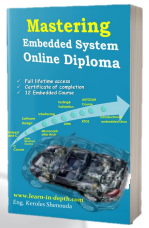


Hardware Interrupt

- A **hardware interrupt** is an **electronic alerting signal sent to the processor from an external device or internal Module**, like a disk controller or an external peripheral. For example, when we press a key on the keyboard or move the mouse, they trigger hardware interrupts which cause the processor to read the keystroke or mouse position.

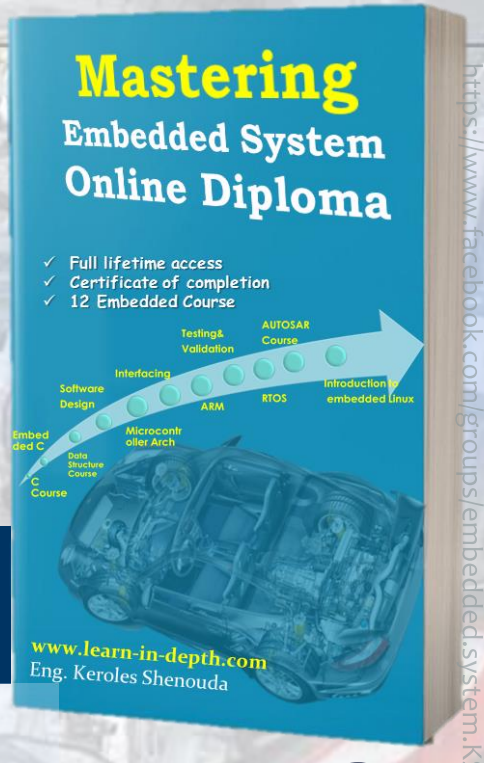


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

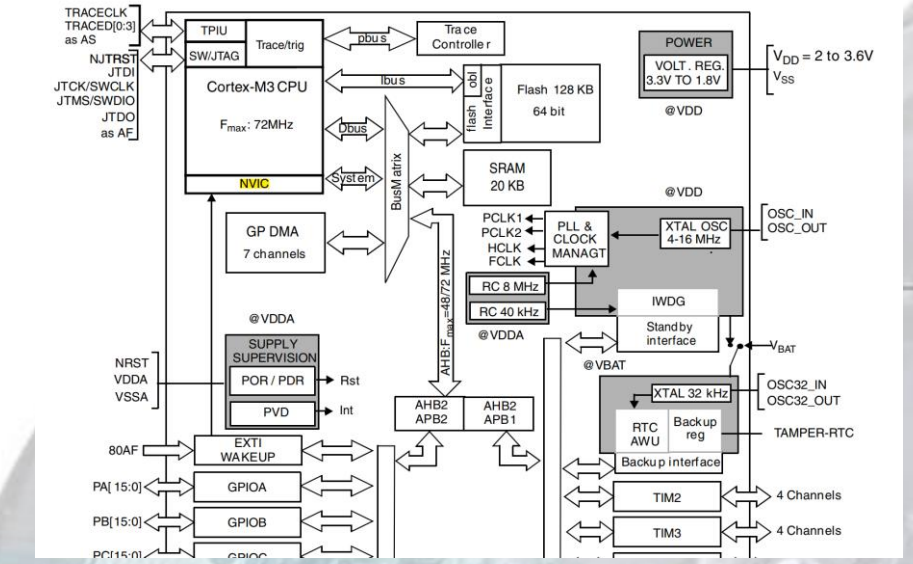
#Be_professional_in_embedded_system



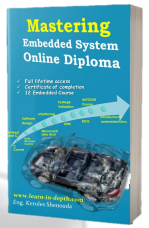
Navigate IVT from STM32F103XX TRM

LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



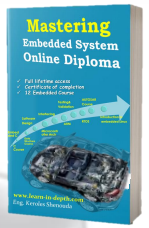
Position	Priority	Type of priority	Acronym	Description	Address
-	-	-	-	Reserved	0x0000_0000
-	-3	fixed	Reset	Reset	0x0000_0004
-	-2	fixed	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.	0x0000_0008
-	-1	fixed	HardFault	All class of fault	0x0000_000C
-	0	settable	MemManage	Memory management	0x0000_0010
-	1	settable	BusFault	Pre-fetch fault, memory access fault	0x0000_0014
-	2	settable	UsageFault	Undefined instruction or illegal state	0x0000_0018
-	-	-	-	Reserved	0x0000_001C - 0x0000_002B
-	3	settable	SVCall	System service call via SWI instruction	0x0000_002C
-	4	settable	Debug Monitor	Debug Monitor	0x0000_0030
-	-	-	-	Reserved	0x0000_0034
-	5	settable	PendSV	Pendable request for system service	0x0000_0038
-	6	settable	SysTick	System tick timer	0x0000_003C
0	7	settable	WWDG	Window Watchdog interrupt	0x0000_0040
1	8	settable	PVD	PVD through EXTI Line detection interrupt	0x0000_0044
2	9	settable	TAMPER	Tamper interrupt	0x0000_0048
3	10	settable	RTC	RTC global interrupt	0x0000_004C
4	11	settable	FLASH	Flash global interrupt	0x0000_0050
5	12	settable	RCC	RCC global interrupt	0x0000_0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000_0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000_0068



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

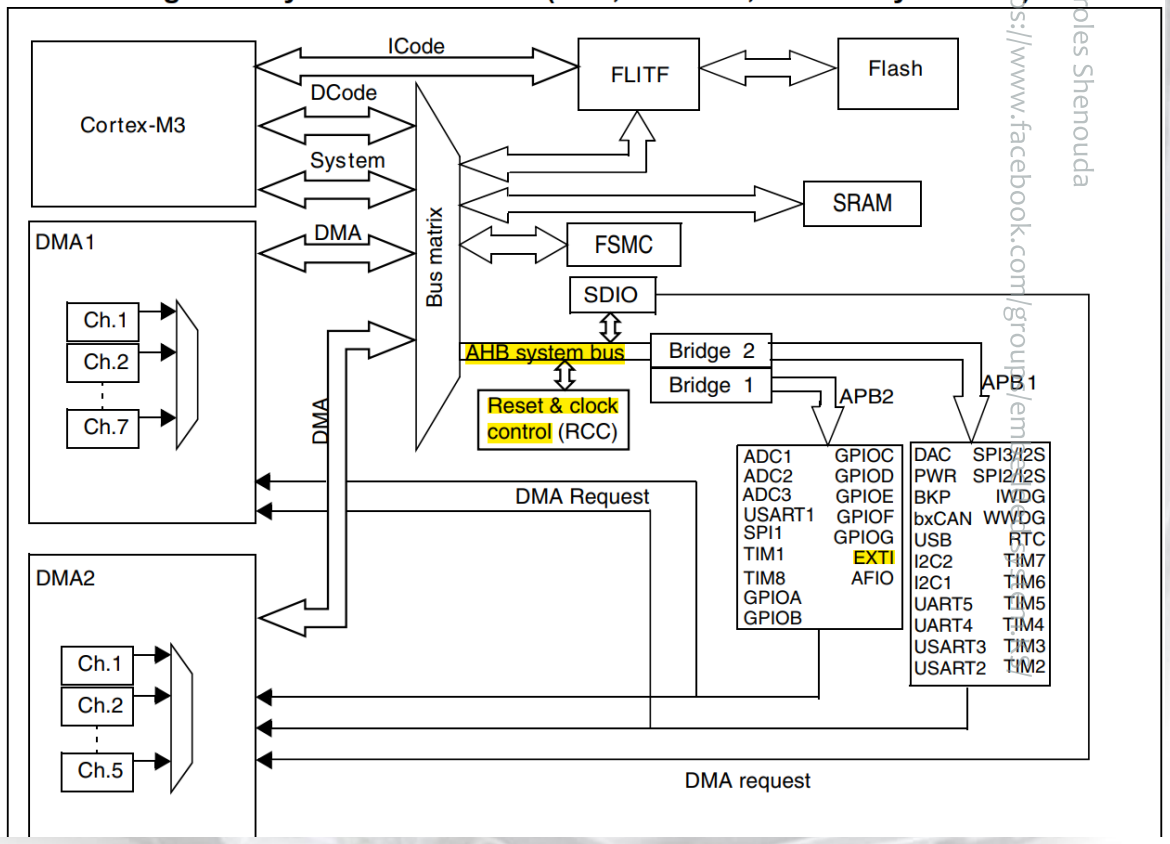
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



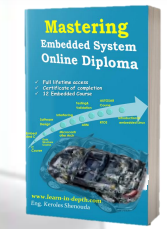
EXTI

0x4001 3000 - 0x4001 33FF	SPI1	APB2	Section 25.5 on page 742
0x4001 2C00 - 0x4001 2FFF	TIM1 timer		Section 14.4.21 on page 363
0x4001 2800 - 0x4001 2BFF	ADC2		Section 11.12.15 on page 252
0x4001 2400 - 0x4001 27FF	ADC1		
0x4001 2000 - 0x4001 23FF	GPIO Port G		Section 9.5 on page 194
0x4001 1C00 - 0x4001 1FFF	GPIO Port F		
0x4001 1800 - 0x4001 1BFF	GPIO Port E		
0x4001 1400 - 0x4001 17FF	GPIO Port D		
0x4001 1000 - 0x4001 13FF	GPIO Port C		
0x4001 0C00 - 0x4001 0FFF	GPIO Port B		
0x4001 0800 - 0x4001 0BFF	GPIO Port A		
0x4001 0400 - 0x4001 07FF	EXTI		Section 10.3.7 on page 214
0x4001 0000 - 0x4001 03FF	AFIO		Section 9.5 on page 194

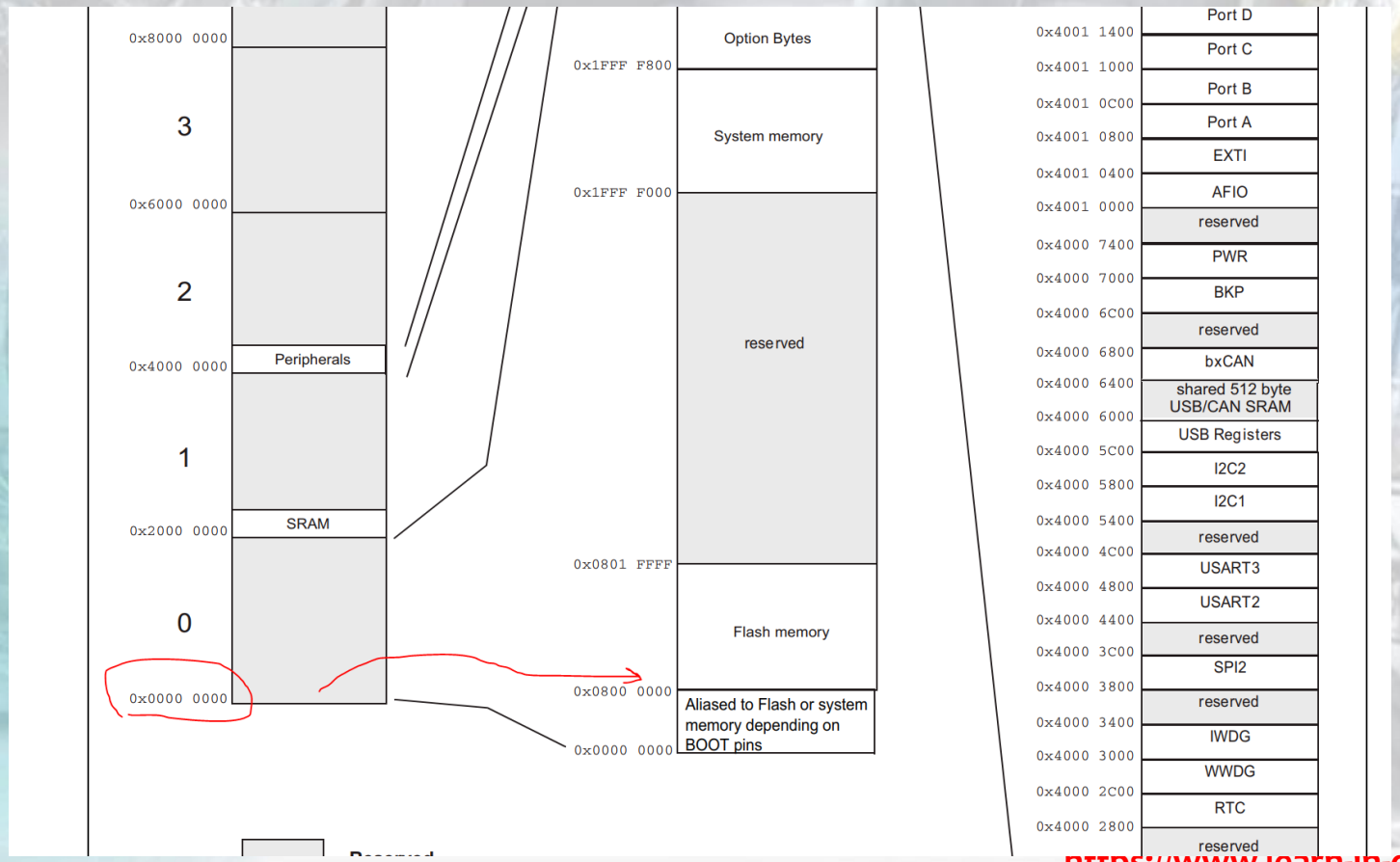
Figure 1. System architecture (low-, medium-, XL-density devices)



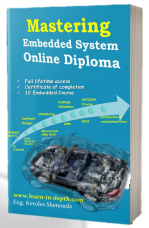
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Memory Location (Based on SoC)

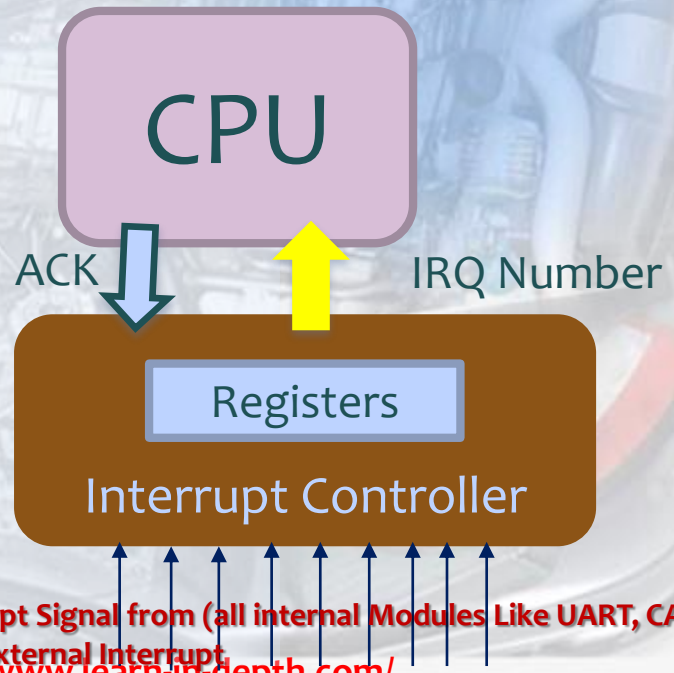


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



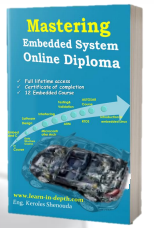
Interrupt Controller IC

- Responsible for telling the CPU when a specific external device wishes to 'interrupt'
  - Needs to tell the CPU *which* one among several devices is the one needing service
- IC translates IRQ to *vector*
  - Raises interrupt to CPU
  - Vector available in register
  - Waits for ack from CPU
- Interrupts can have varying priorities
  - IC also needs to prioritize multiple requests
- Possible to "mask" (disable) interrupts at IC or CPU



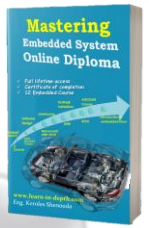
All Interrupt Signal from (all internal Modules Like UART, CAN, I2C ,etc...)
Or from External Interrupt

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



How to make the Interrupt Reach to the CPU ? For example

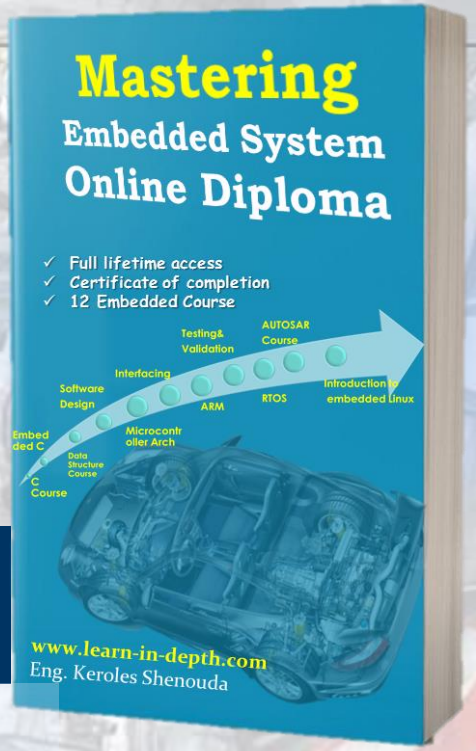
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

Examples On Interrupt Controllers



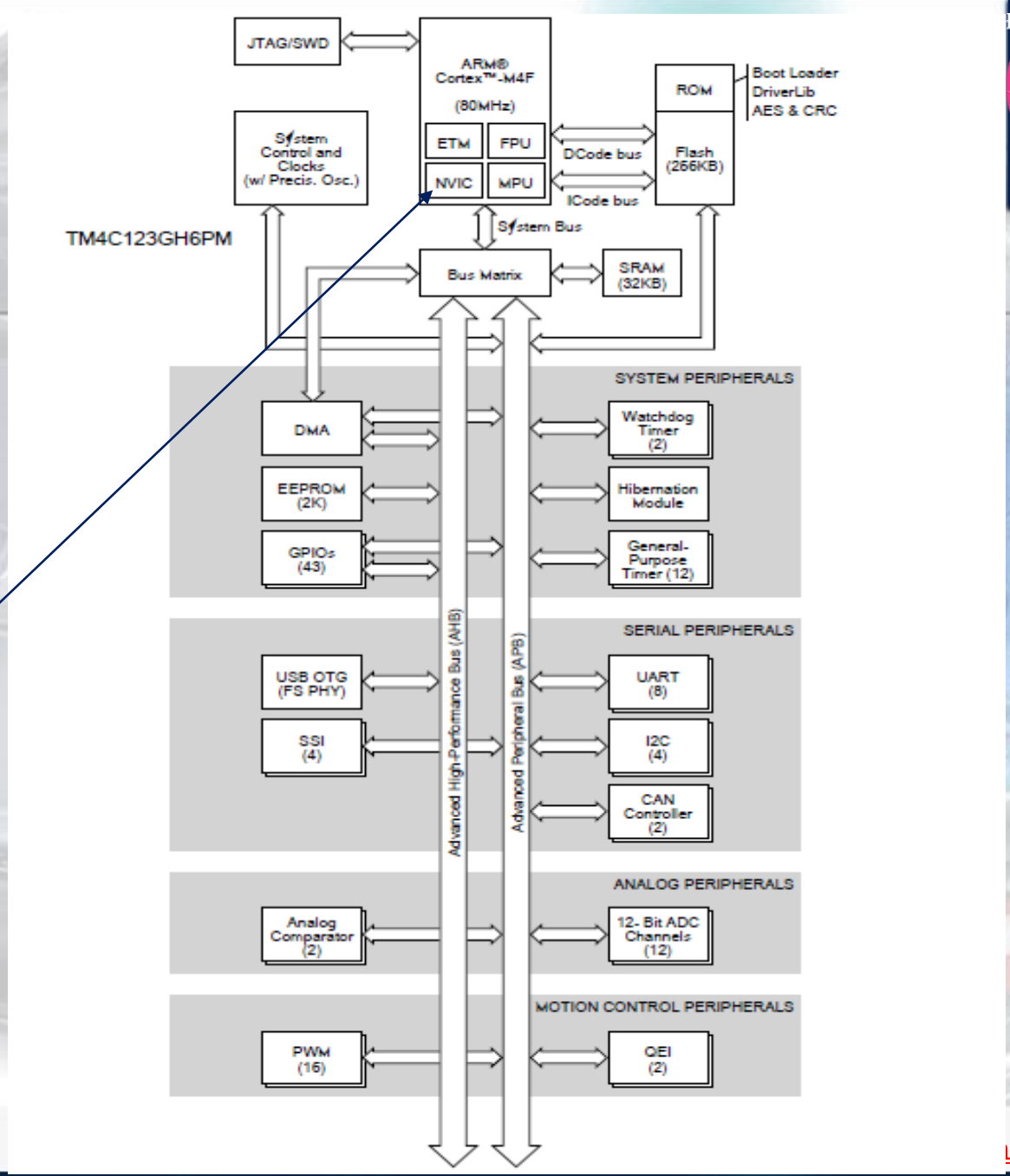
LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

TivaC Cortex-M4 NVIC

“Nested Vectored Interrupt Controller”

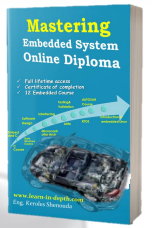
NVIC



FOLLOW US

Press here

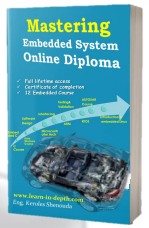
97



#LEARN_IN DEPTH

#Be_professional_in embedded_system

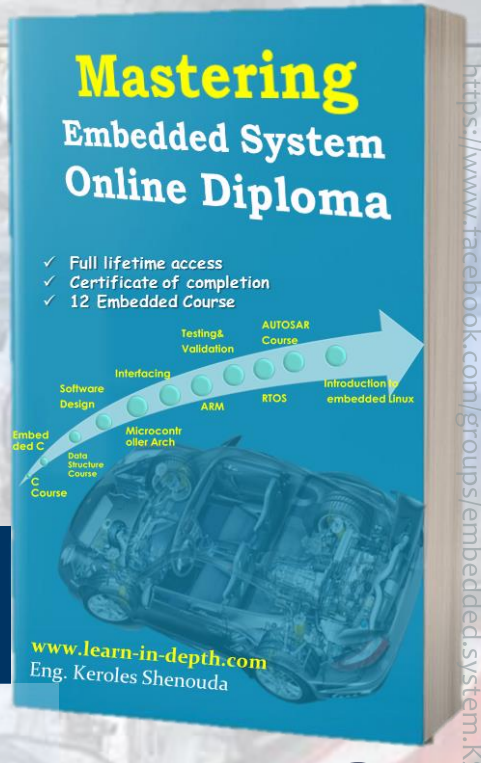
bedded.system.KS/



#LEARN_IN DEPTH

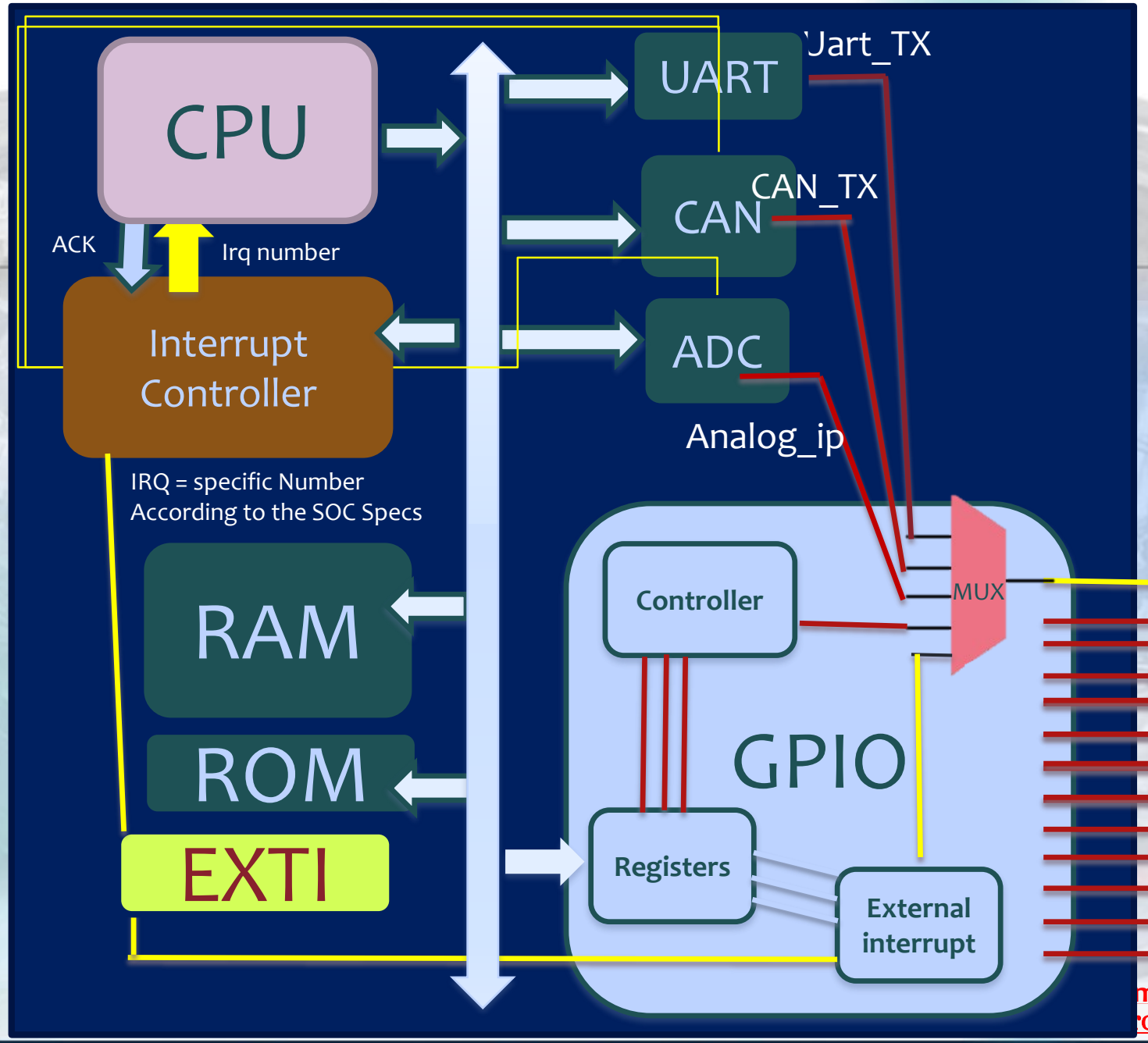
#Be_professional_in_embedded_system

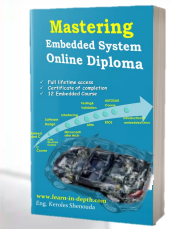
Lab2: enable Ext Interrupt for Stm32F103XX



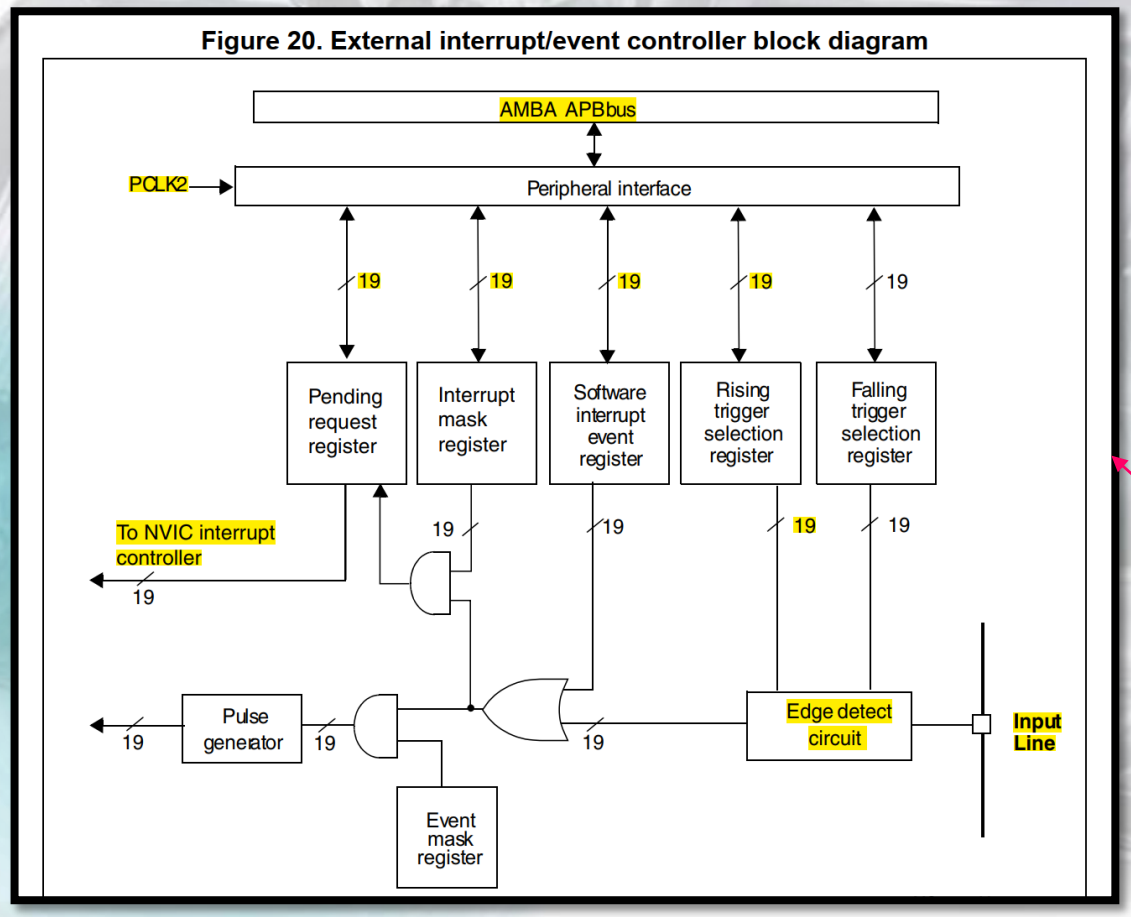
LEARN-IN-DEPTH
Be professional in embedded system

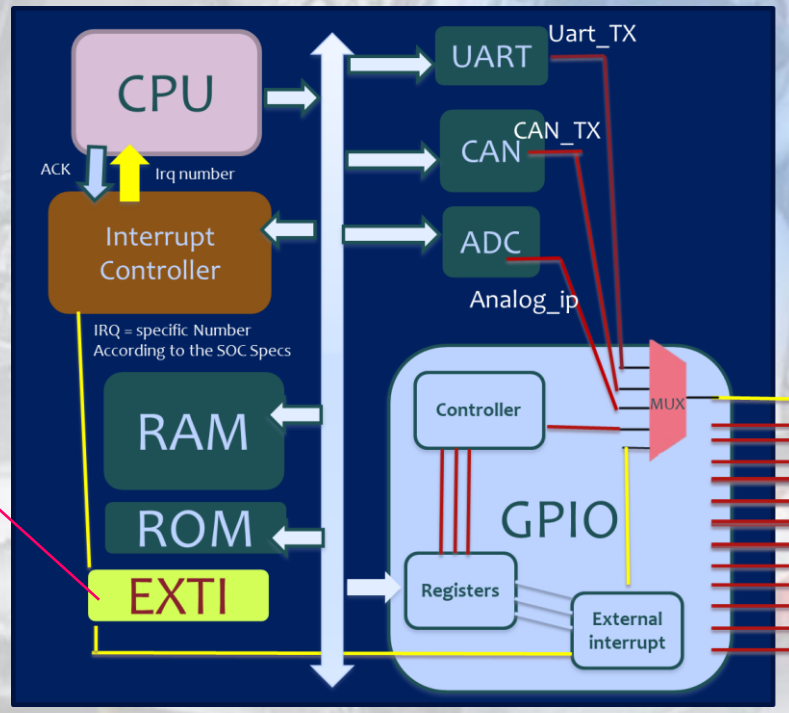
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



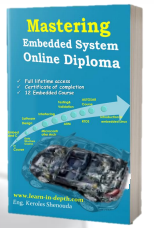


External interrupt/event controller block diagram





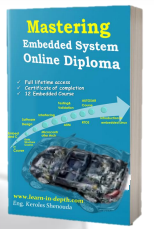
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Ext Interrupt 16

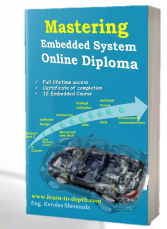
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000_0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000_0068
23	30	settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000_009C
39	46	settable	USART3	USART3 global interrupt	0x0000_00DC
40	47	settable	EXTI15_10	EXTI Line[15:10] interrupts	0x0000_00E0

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

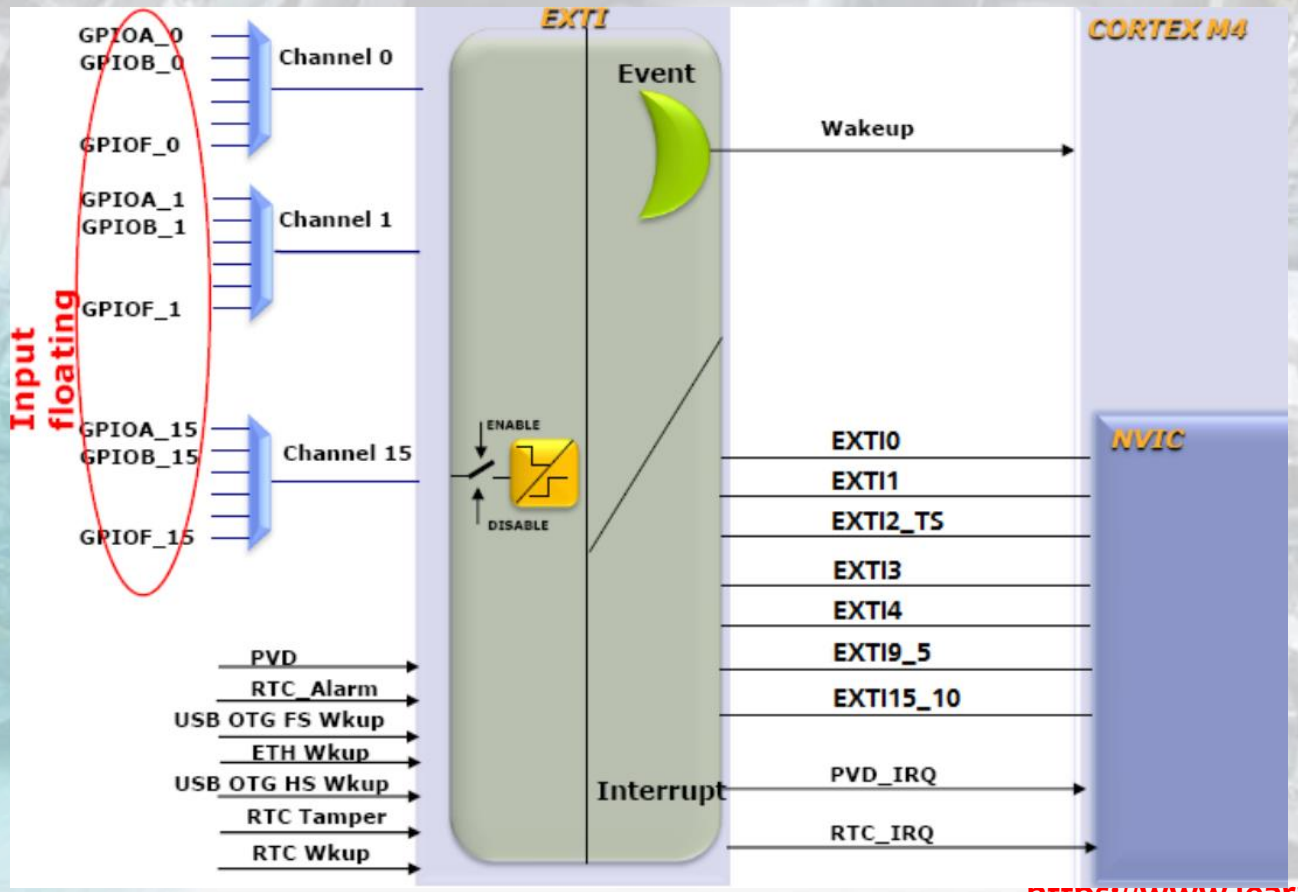


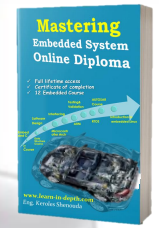
Additional External Interrupt connected to NVIC through EXTI

41	48	settable	RTCAlarm	RTC alarm through EXTI line interrupt	0x0000_00E4
42	49	settable	OTG_FS_WKUP	USB On-The-Go FS Wakeup through EXTI line interrupt	0x0000_00E8
62	69	settable	ETH_WKUP	Ethernet Wakeup through EXTI line interrupt	0x0000_0138
1	8	settable	PVD	PVD through EXTI Line detection interrupt	0x0000_0044

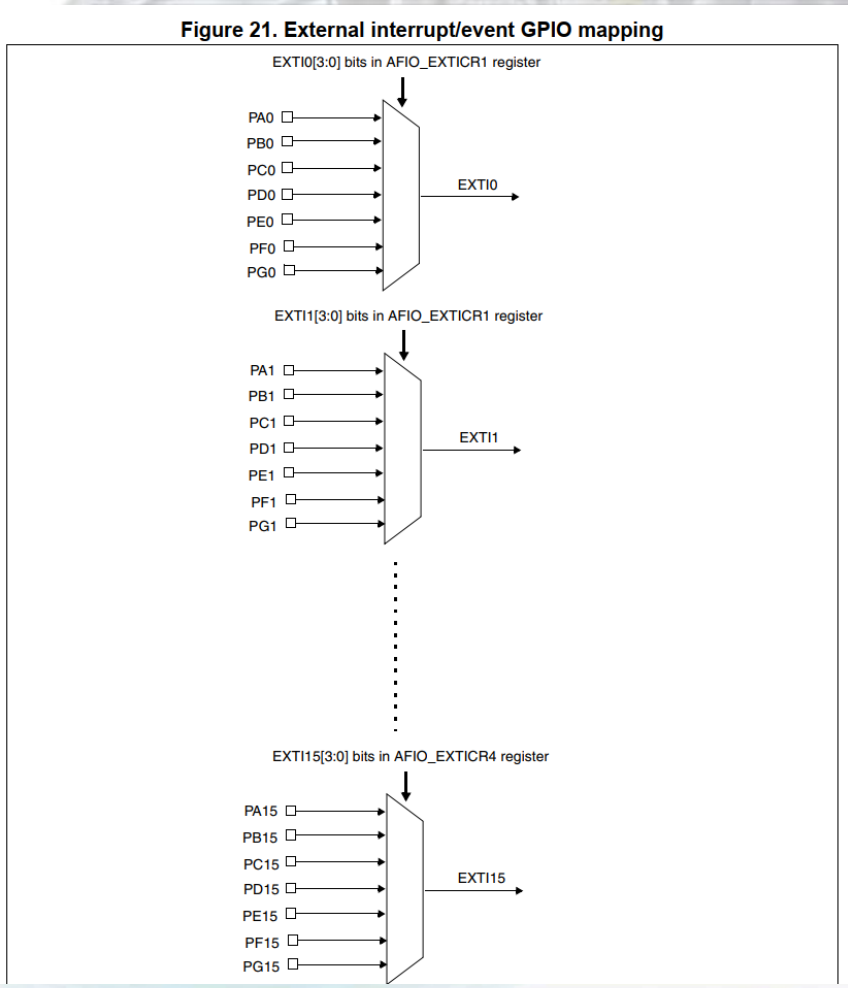


EXTI Module



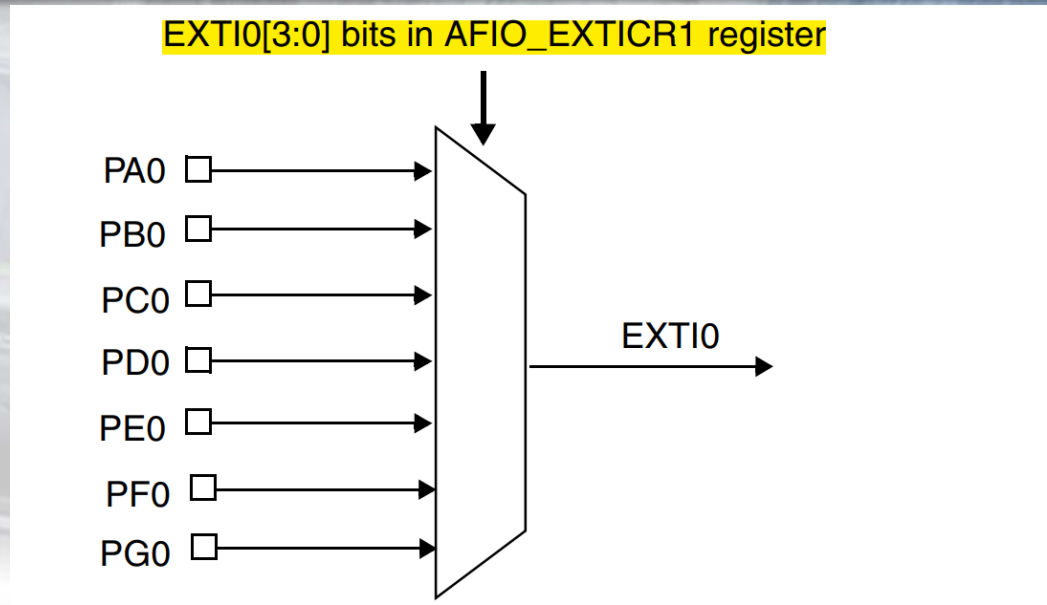


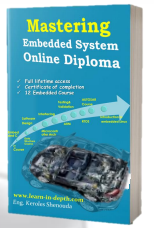
External interrupt/event line mapping



10.2.5 External interrupt/event line mapping

The 112 GPIOs are connected to the 16 external interrupt/event lines in the following manner:

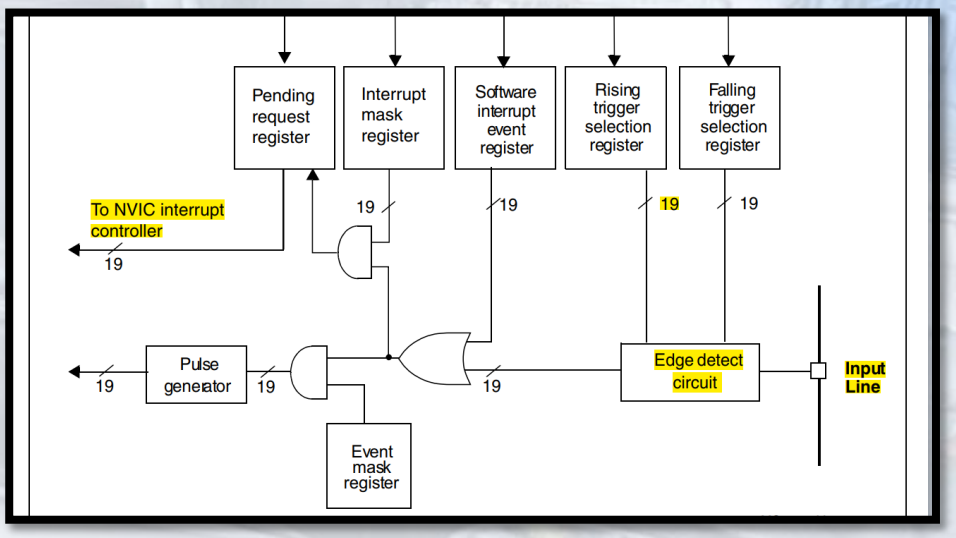




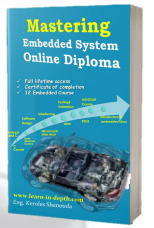
What we need ....

When Pressed in PA0 (GPIO A) -> EXTI0 Line -> NVIC -> CPU

- GPIOA is Configured to be Input Mode
- Enable EXTI0 Line to be Connected with GPIOA0
- Configure the Trigger Detection (Failling/Rising/both) for EXTI0
- Implement the Handler



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

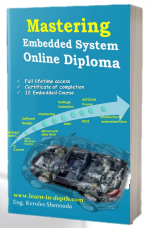


10.2.4 Functional description

To generate the interrupt, the interrupt line should be configured and enabled. This is done by programming the two trigger registers with the desired edge detection and by enabling the interrupt request by writing a '1' to the corresponding bit in the interrupt mask register. When the selected edge occurs on the external interrupt line, an interrupt request is generated. The pending bit corresponding to the interrupt line is also set. This request is reset by writing a '1' in the pending register.

To generate the event, the event line should be configured and enabled. This is done by programming the two trigger registers with the desired edge detection and by enabling the event request by writing a '1' to the corresponding bit in the event mask register. When the selected edge occurs on the event line, an event pulse is generated. The pending bit corresponding to the event line is not set

An interrupt/event request can also be generated by software by writing a '1' in the software interrupt/event register.



Hardware interrupt selection

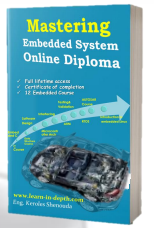
To configure the 20 lines as interrupt sources, use the following procedure:

- Configure the mask bits of the 20 Interrupt lines (EXTI_IMR)
- Configure the Trigger Selection bits of the Interrupt lines (EXTI_RTSR and EXTI_FTSR)
- Configure the enable and mask bits that control the NVIC IRQ channel mapped to the External Interrupt Controller (EXTI) so that an interrupt coming from one of the 20 lines can be correctly acknowledged.

Hardware event selection

To configure the 20 lines as event sources, use the following procedure:

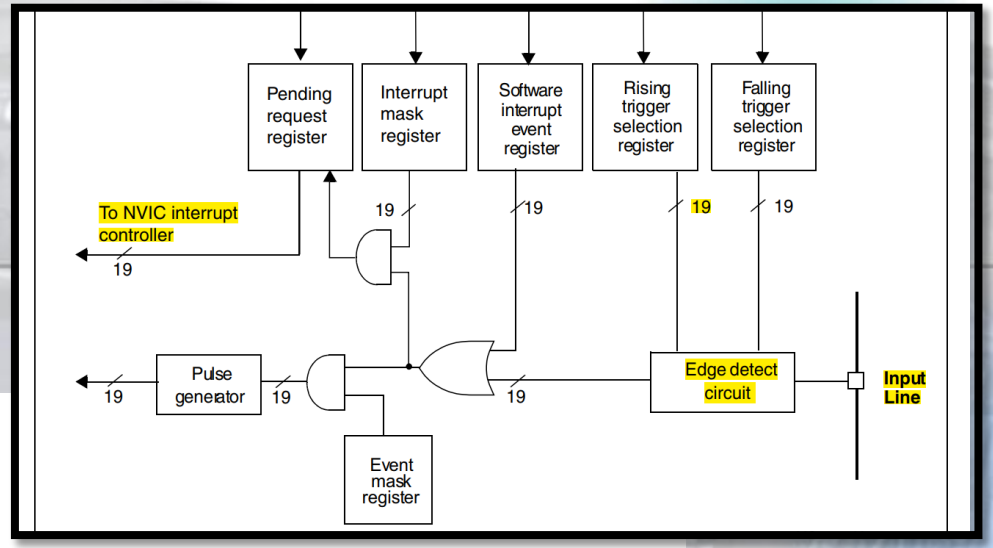
- Configure the mask bits of the 20 Event lines (EXTI_EMR)
- Configure the Trigger Selection bits of the Event lines (EXTI_RTSR and EXTI_FTSR)



Software interrupt/event selection

The 20 lines can be configured as software interrupt/event lines. The following is the procedure to generate a software interrupt.

- Configure the mask bits of the 20 Interrupt/Event lines (EXTI_IMR, EXTI_EMR)
- Set the required bit of the software interrupt register (EXTI_SWIER)



10.3.1 Interrupt mask register (EXTI_IMR)

Address offset: 0x00
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												MR19	MR18	MR17	MR16
												rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

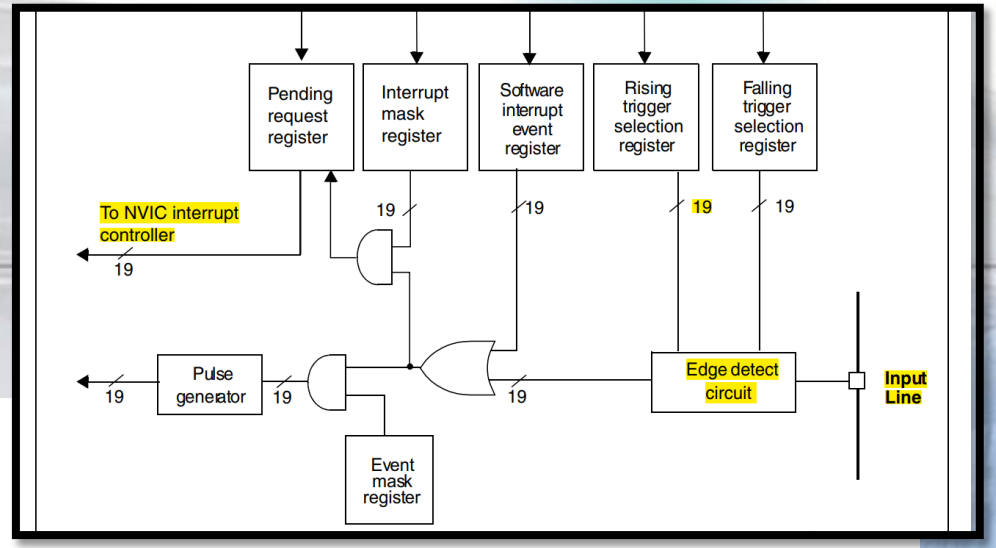
Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **MRx:** Interrupt Mask on line x

0: Interrupt request from Line x is masked
1: Interrupt request from Line x is not masked

Note: Bit 19 is used in connectivity line devices only and is reserved otherwise.

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



10.3.2 Event mask register (EXTI_EMR)

Address offset: 0x04
Reset value: 0x0000 0000

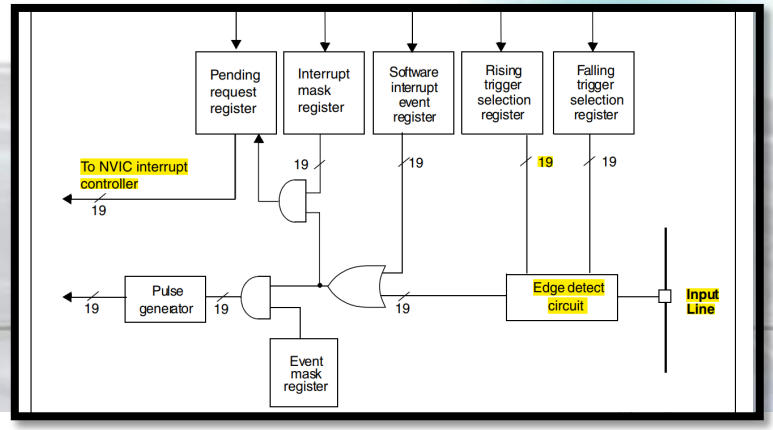
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												MR19	MR18	MR17	MR16
												rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **MRx:** Event mask on line x

0: Event request from Line x is masked
1: Event request from Line x is not masked



10.3.3 Rising trigger selection register (EXTI_RTSR)

Address offset: 0x08
Reset value: 0x0000 0000

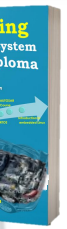
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												TR19	TR18	TR17	TR16
												rw	rw	rw	rw

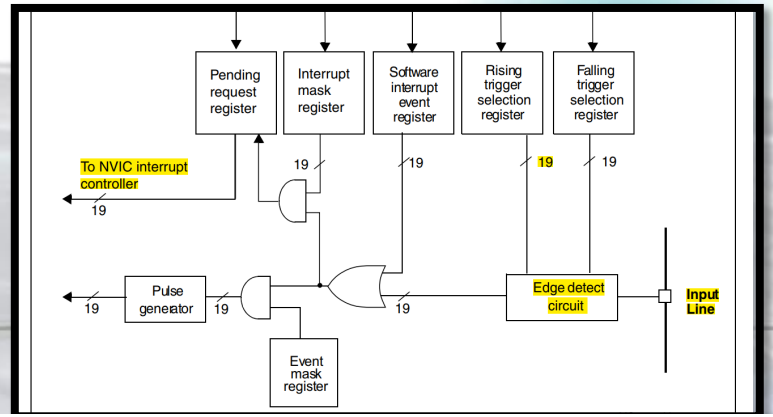
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **TRx:** Rising trigger event configuration bit of line x

0: Rising trigger disabled (for Event and Interrupt) for input line
1: Rising trigger enabled (for Event and Interrupt) for input line.





10.3.4 Falling trigger selection register (EXTI_FTSR)

Address offset: 0x0C
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved												TR19	TR18	TR17	TR16
												rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **TRx:** Falling trigger event configuration bit of line x
0: Falling trigger disabled (for Event and Interrupt) for input line
1: Falling trigger enabled (for Event and Interrupt) for input line.

10.3.5 Software interrupt event register (EXTI_SWIER)

Address offset: 0x10
Reset value: 0x0000 0000

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

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

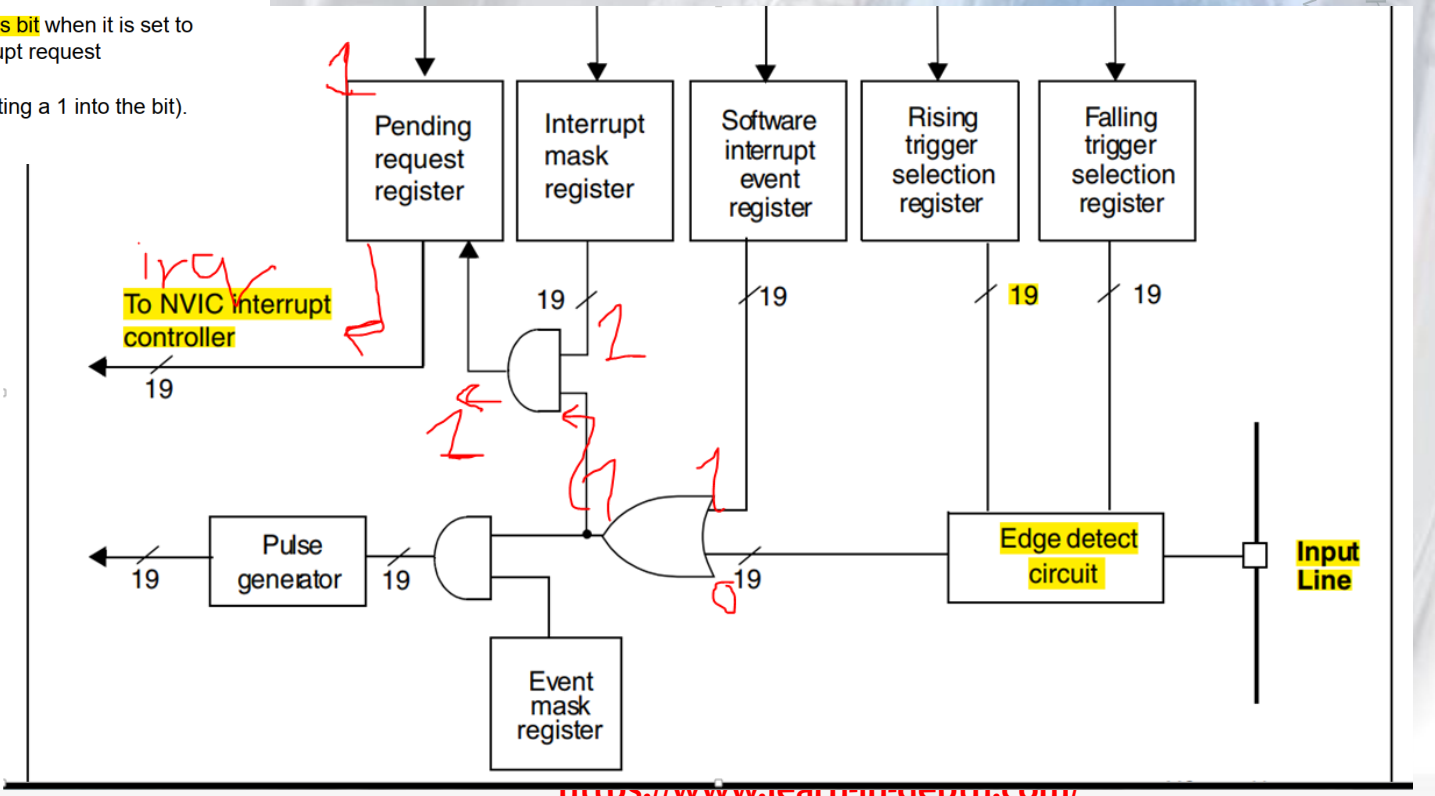
Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **SWIERx:** Software interrupt on line x

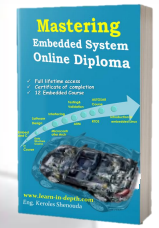
If the interrupt is enabled on this line in the EXTI_IMR, writing a '1' to this bit when it is set to '0' sets the corresponding pending bit in EXTI_PR resulting in an interrupt request generation.

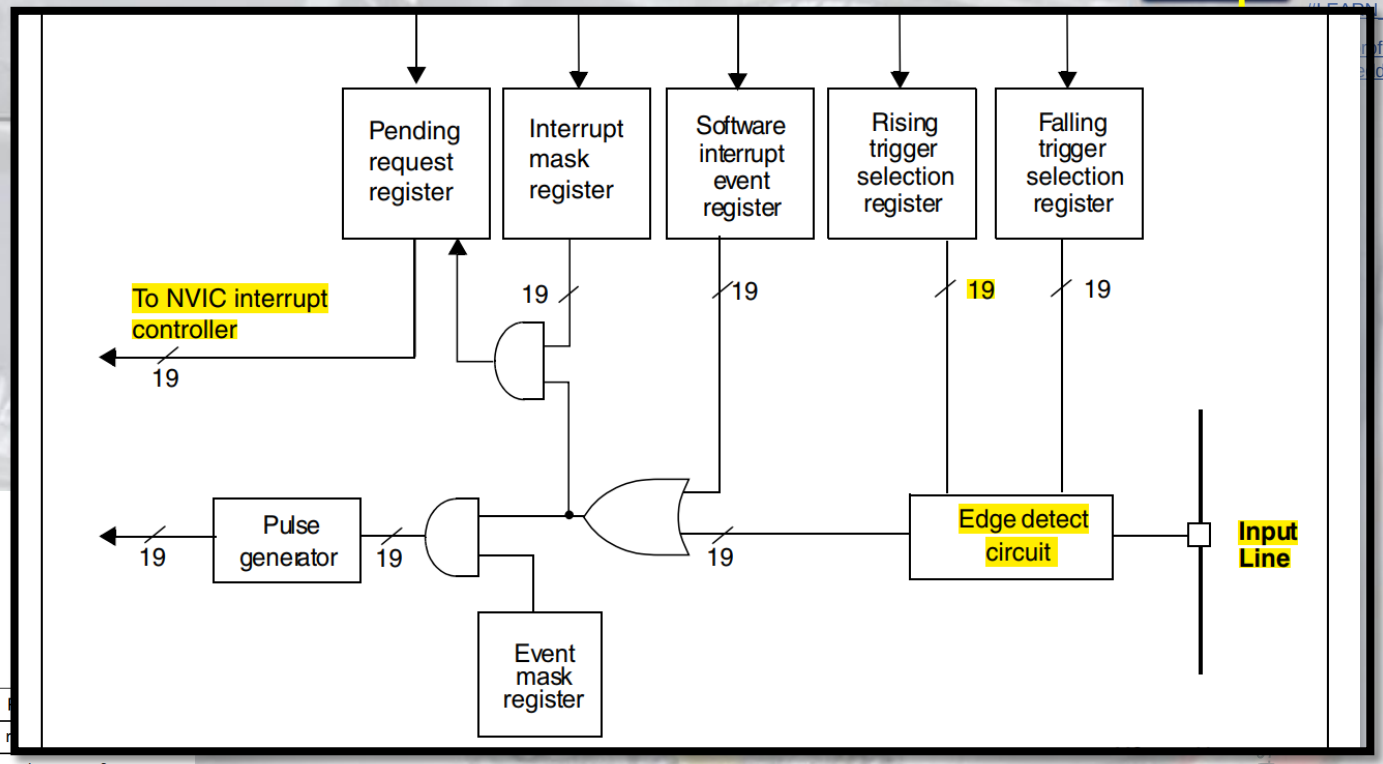
This bit is cleared by clearing the corresponding bit of EXTI_PR (by writing a 1 into the bit).

Note: Bit 19 used in connectivity line devices and is reserved otherwise.



You have to Clear PR in the Handler to avoid infinite interrupt looping





10.3.6 Pending register (EXTI_PR)

Address offset: 0x14
Reset value: undefined

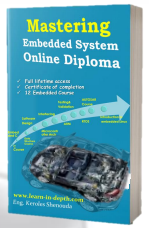
31	30	29	28	27	26	25	24	23	22	21	20	19	18	
Reserved												PR19	PR18	
												rc_w1	rc_w1	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:20 Reserved, must be kept at reset value (0).

Bits 19:0 **PRx:** Pending bit
0: No trigger request occurred
1: selected trigger request occurred
This bit is set when the selected edge event arrives on the external interrupt line. This bit is cleared by writing a '1' into the bit.

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

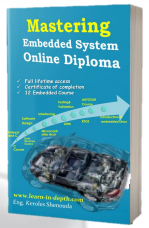


EXT Interrupt Registers

Table 64. External interrupt/event controller register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	EXTI_IMR	Reserved												MR[19:0]																			
	Reset value													0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x04	EXTI_EMR	Reserved												MR[19:0]																			
	Reset value													0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	EXTI_RTSR	Reserved												TR[19:0]																			
	Reset value													0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	EXTI_FTSR	Reserved												TR[19:0]																			
	Reset value													0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	EXTI_SWIER	Reserved												SWIER[19:0]																			
	Reset value													0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x14	EXTI_PR	Reserved												PR[19:0]																			
	Reset value													0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

Generate irq & Mask Register

Interrupt Pending Register =1

EXTI0

IRQ & IntController Mask Register For IRQ17

Check also the Priority

NVIC
Interrupt Controller

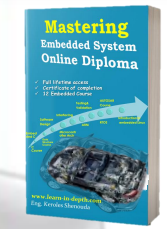
Interrupt Happen

CPU

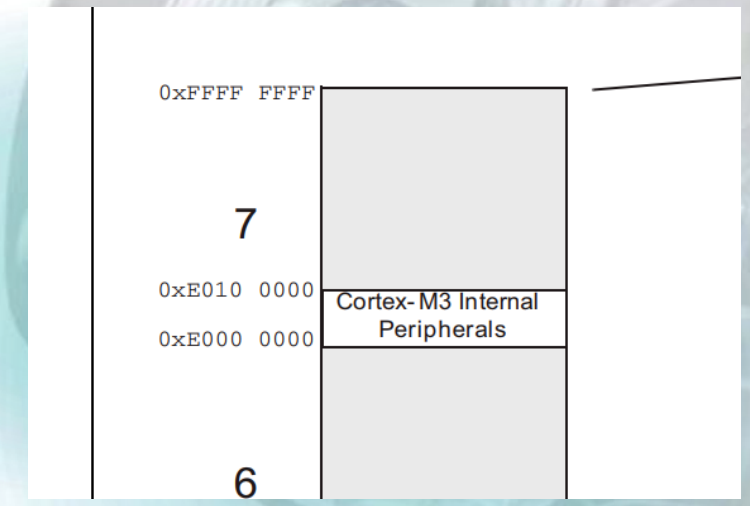
Call ISR

Arm CortexM

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



NVIC Inside Cortex M3



arm Developer

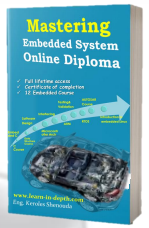
IP Products

Version: 1.0 | Download | Dark | Subscribe | Search within this document

- Back to search
- All Cortex-M3 Documentation
- Cortex-M3 Devices Generic User Guide
  - preface
  - Introduction
  - The Cortex-M3 Processor
  - The Cortex-M3 Instruction Set
  - Cortex-M3 Peripherals
    - About the Cortex-M3 peripherals
    - **Nested Vectored Interrupt Controller**
    - System control block
    - System timer, SysTick
    - Optional Memory Protection Unit

Table 4.2. NVIC register summary

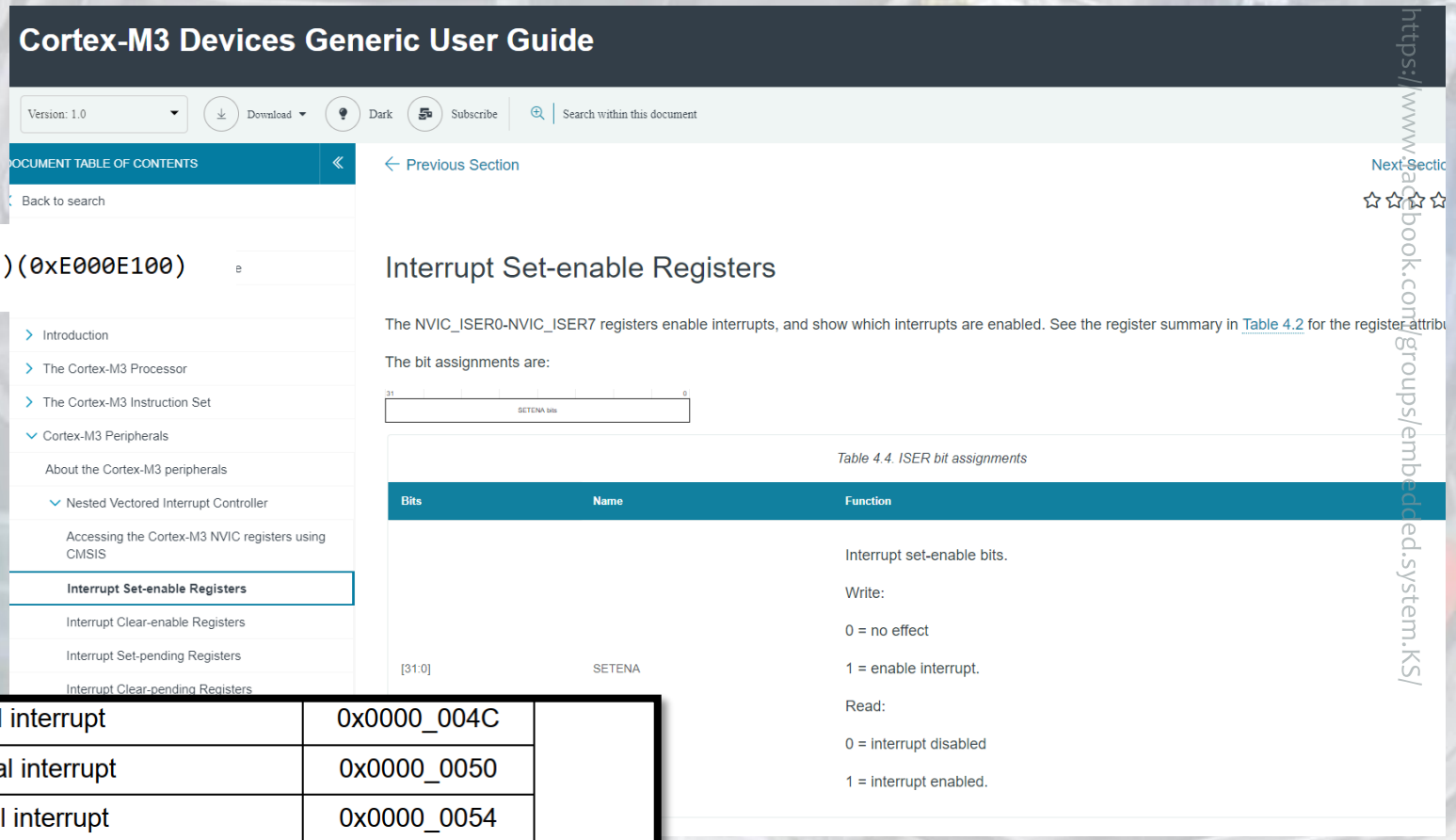
Address	Name	Type	Required privilege	Reset value	Description
0xE000E100 - 0xE000E11C	NVIC_ISER0- NVIC_ISER7	RW	Privileged	0x00000000	Interrupt Set-enable Registers
0XE000E180- 0xE000E19C	NVIC_ICER0- NVIC_ICER7	RW	Privileged	0x00000000	Interrupt Clear-enable Registers
0XE000E200- 0xE000E21C	NVIC_ISPR0- NVIC_ISPR7	RW	Privileged	0x00000000	Interrupt Set-pending Registers
0XE000E280- 0xE000E29C	NVIC_ICPR0- NVIC_ICPR7	RW	Privileged	0x00000000	Interrupt Clear-pending Registers
0XE000E300- 0xE000E31C	NVIC_IABR0- NVIC_IABR7	RW	Privileged	0x00000000	Interrupt Active Bit Registers
0XE000E400- 0xE000E4EF	NVIC_IPR0- NVIC_IPR59	RW	Privileged	0x00000000	Interrupt Priority Registers
0XE000EF00	STIR	WO	Configurable [a]	0x00000000	Software Trigger Interrupt Register



Cortex-M3 Devices Generic User Guide

```
48
49 #define NVIC_ISER0      *(volatile uint32_t *)(0xE000E100)
50
```

```
//Enable NVIC IRQ 6 (EXTI0)
NVIC_ISER0 |= (1<<6) ;
```



Cortex-M3 Devices Generic User Guide

Version: 1.0 | Download | Dark | Subscribe | Search within this document

DOCUMENT TABLE OF CONTENTS

Back to search

- Introduction
- The Cortex-M3 Processor
- The Cortex-M3 Instruction Set
- Cortex-M3 Peripherals
  - About the Cortex-M3 peripherals
  - Nested Vectored Interrupt Controller
    - Accessing the Cortex-M3 NVIC registers using CMSIS
    - **Interrupt Set-enable Registers**
    - Interrupt Clear-enable Registers
    - Interrupt Set-pending Registers
    - Interrupt Clear-pending Registers

← Previous Section Next Sectio

Interrupt Set-enable Registers

The NVIC_ISER0-NVIC_ISER7 registers enable interrupts, and show which interrupts are enabled. See the register summary in Table 4.2 for the register attribu

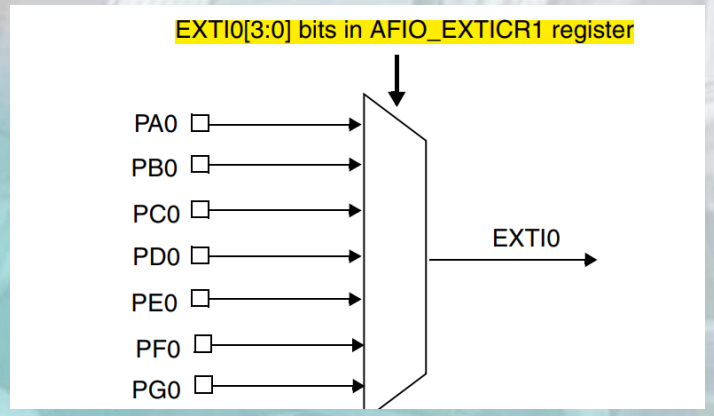
The bit assignments are:

Table 4.4. ISER bit assignments

Bits	Name	Function
[31:0]	SETENA	Interrupt set-enable bits. Write: 0 = no effect 1 = enable interrupt. Read: 0 = interrupt disabled 1 = interrupt enabled.

3	10	settable	RTC	RTC global interrupt	0x0000_004C
4	11	settable	FLASH	Flash global interrupt	0x0000_0050
5	12	settable	RCC	RCC global interrupt	0x0000_0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C

0x4001 0000 - 0x4001 03FF	AFIO		*Section 9.5 on page 194*



9.4.3 External interrupt configuration register 1 (AFIO_EXTICR1)

Address offset: 0x08

Reset value: 0x0000

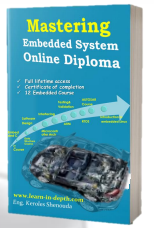
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved

Bits 15:0 **EXTIx[3:0]:** EXTI x configuration (x= 0 to 3)

These bits are written by software to select the source input for EXTIx external interrupt.
Refer to *Section 10.2.5: External interrupt/event line mapping*

0000: PA[x] pin
0001: PB[x] pin
0010: PC[x] pin
0011: PD[x] pin
0100: PE[x] pin
0101: PF[x] pin
0110: PG[x] pin

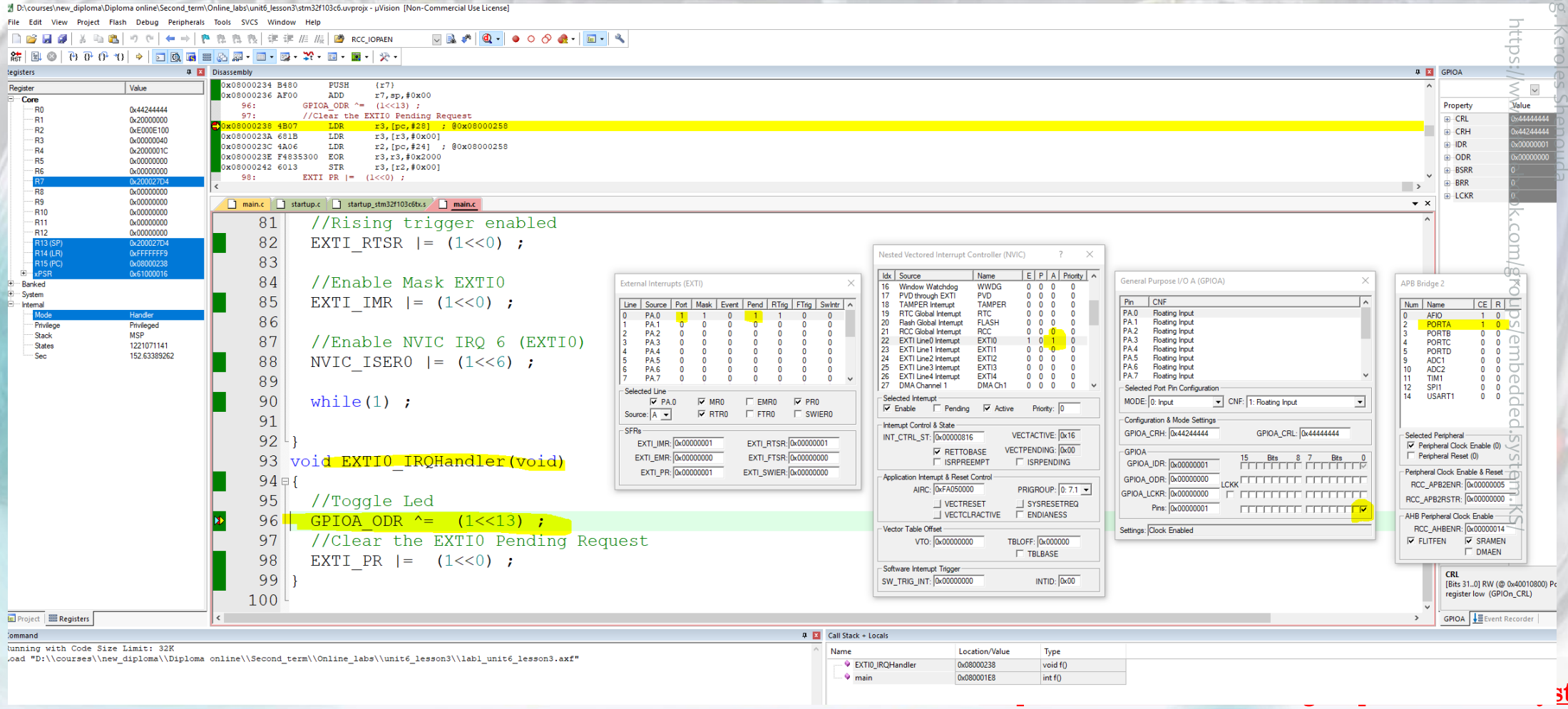


Clock_init

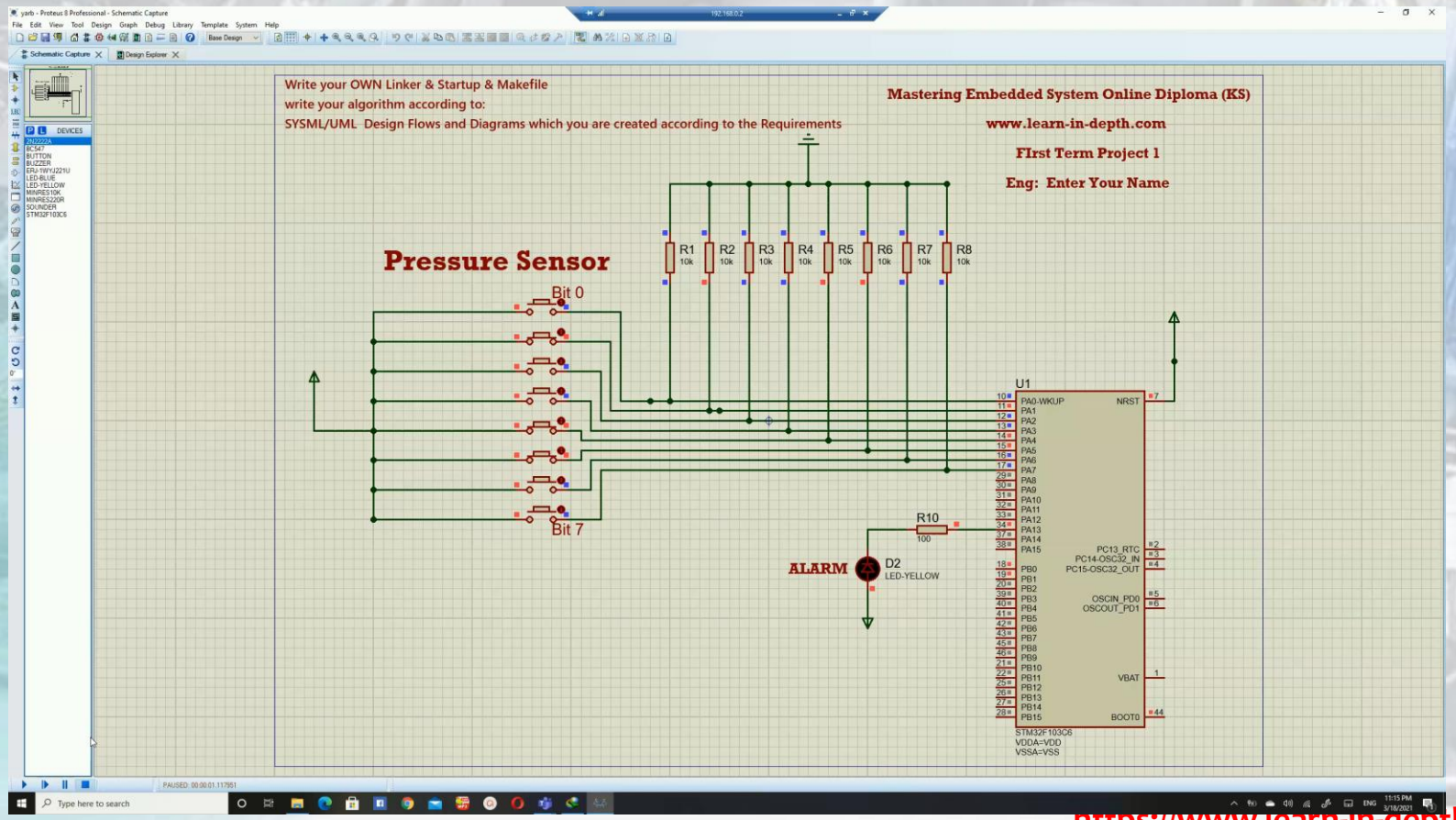
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved										TIM11 EN	TIM10 EN	TIM9 EN	Reserved		
										rw	rw	rw			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC3 EN	USART 1EN	TIM8 EN	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	IOPG EN	IOPF EN	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

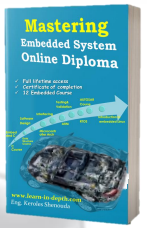
```
}
GPIOA_Pin0_Input_init()
{
    GPIOA_CRL |= (1<<2) ;
}
GPIOA_Pin13_Output_init()
{
    GPIOA_CRH   &= 0xFF0FFFFF;
    GPIOA_CRH   |= 0x00200000;
}
int main(void)
{

    clock_init ();
    GPIOA_Pin0_Input_init ();
    GPIOA_Pin13_Output_init();

    //Select PortA For EXTI0
    AFIO_EXTICR1 = 0x0 ;

    //Rising trigger enabled
    EXTI_RTSR |= (1<<0) ;

    //Enable Mask EXTI0
    EXTI_IMR |= (1<<0) ;

    //Enable NVIC IRQ 6 (EXTI0)
    NVIC_ISER0 |= (1<<6) ;

    while(1) ;

}
void EXTI0_IRQHandler(void)
{
    //Toggle Led
    GPIOA_ODR ^=  (1<<13) ;
    //Clear the EXTI0 Pending Request
    EXTI_PR |=  (1<<0) ;
```

Run It



In Proteus

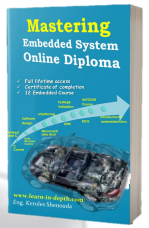




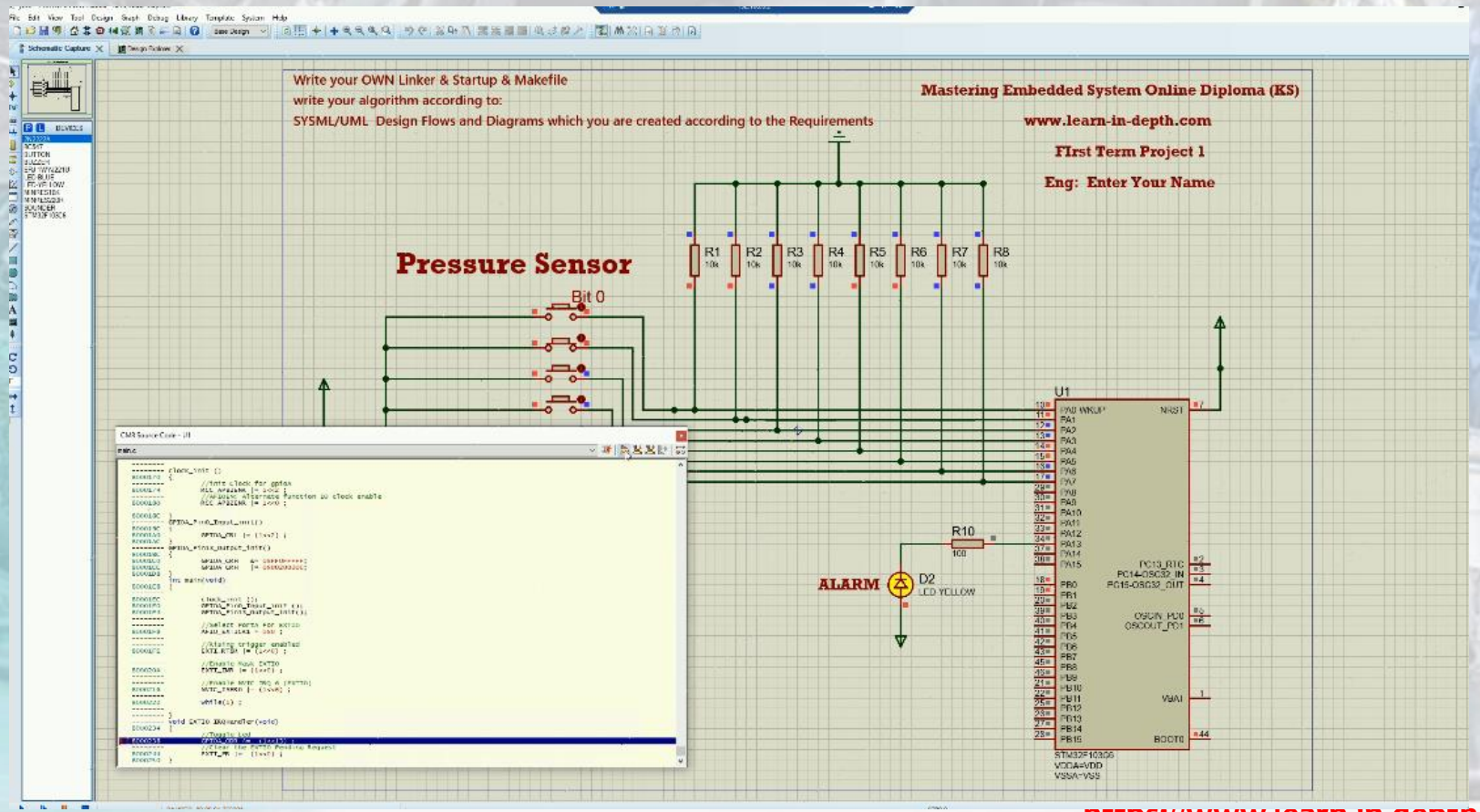
#LEARN_IN DEPTH

#Be_professional_in_embedded_system

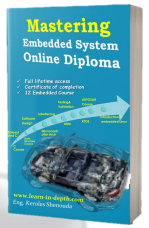
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



In Proteus



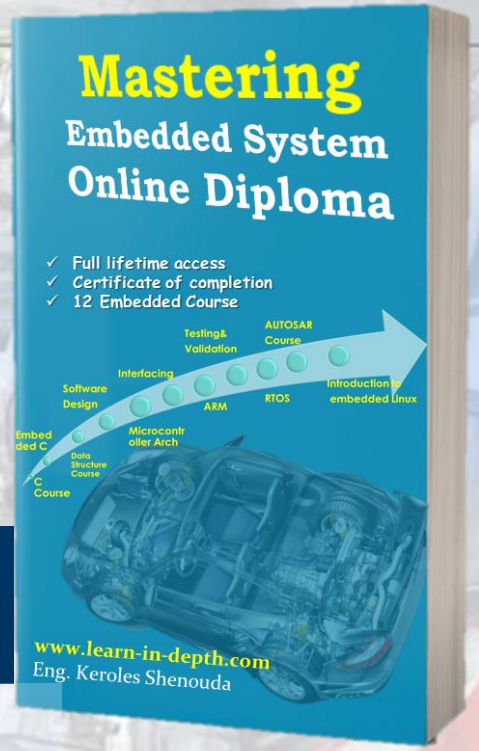
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

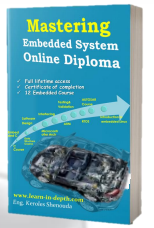
#Be_professional_in_embedded_system

What is Interrupt Overload?



LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



What is Interrupt Overload?

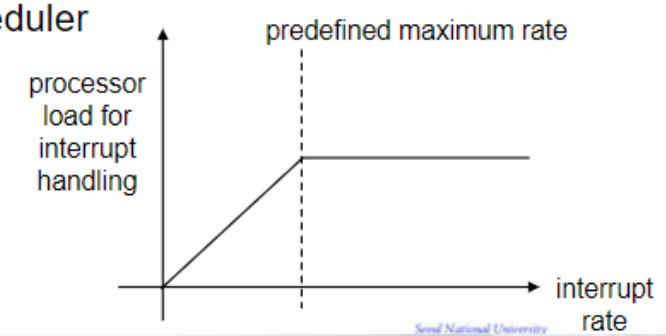
- Interrupt overload: Condition where external interrupts signaled frequently enough to cause other activities running on the processor to be sta

Interrupt overload

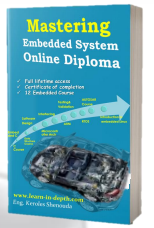
- Too many interrupts that exceed predefined rate.

Solutions for preventing interrupt overload

- Strict software scheduler
- Bursty software scheduler
- Hardware interrupt scheduler

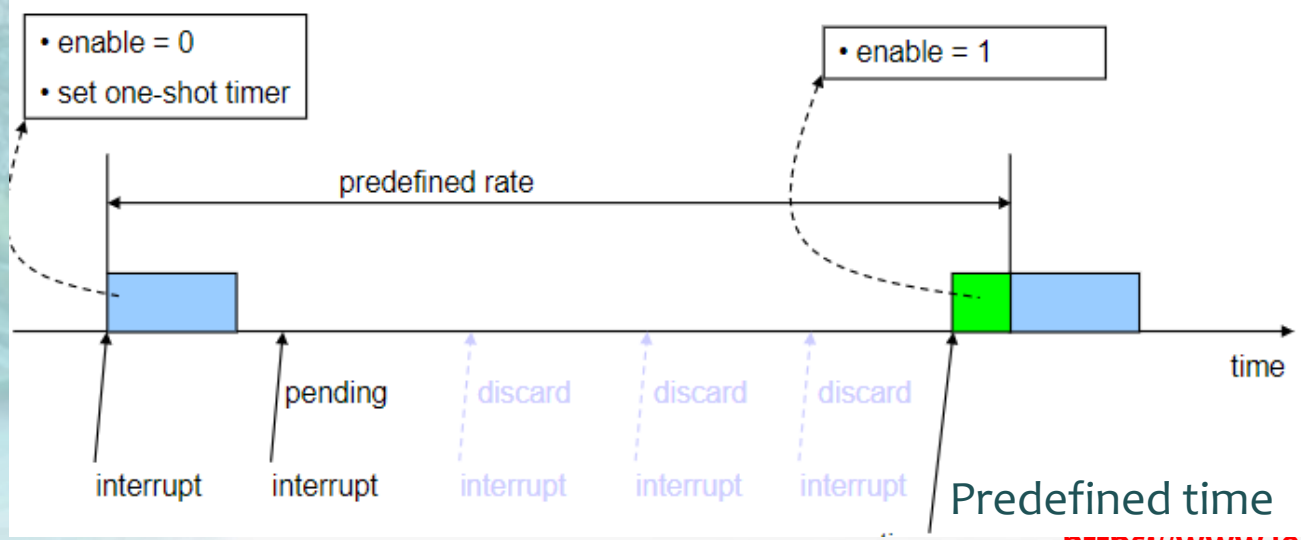


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Solution 1: Strict Software Scheduler

- Interrupt can not be processed faster than a *predefined rate*.
- Modification to interrupt prologue

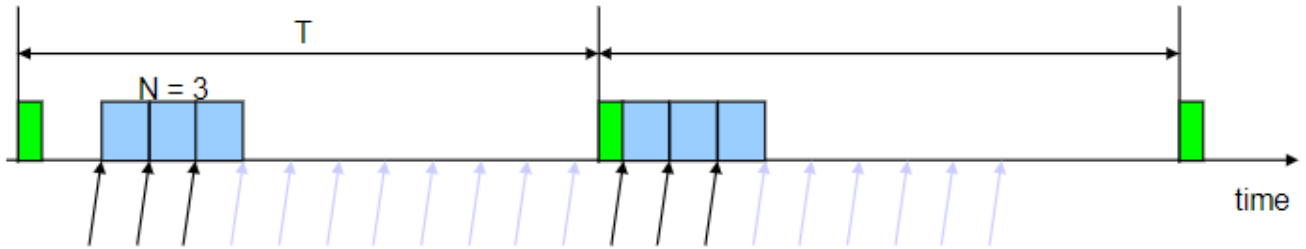


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Solution 2: Bursty Software Scheduler

- Allows maximum N interrupts within T interval.

- *Interrupt handler prologue*
  - counter ++;
  - if counter >= N then enable = off
- *Timer with period T*
  - counter = 0; enable = on



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Hardware Interrupt Schedulate

- This Solution maybe Founded in Advanced Interrupt Controller

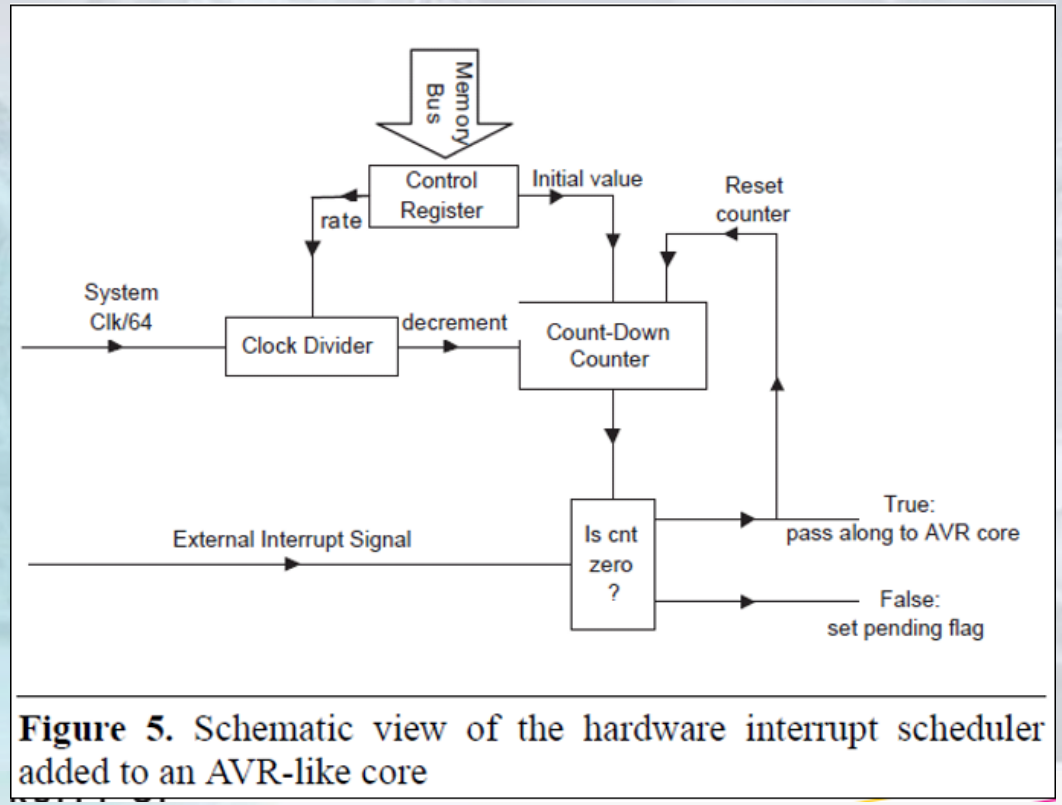
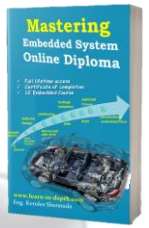


Figure 5. Schematic view of the hardware interrupt scheduler added to an AVR-like core

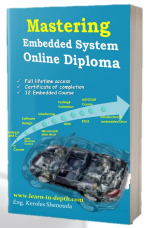
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Conclusion

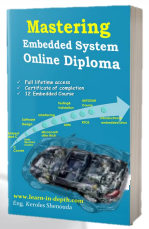
DEFINITIONS

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



DEFINITIONS

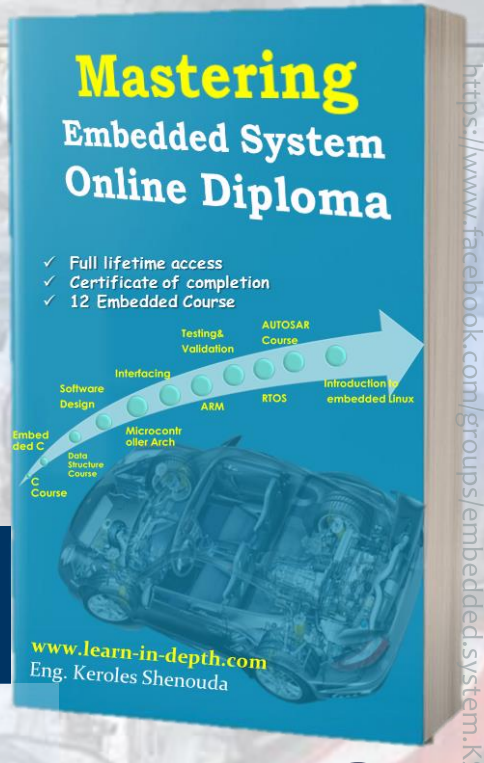
- Interrupt - Hardware-supported asynchronous transfer of control to an
- interrupt vector Interrupt Vector - Dedicated location in memory that specifies address execution jumps to
- Interrupt Handler - Code that is reachable from an interrupt vector
- Interrupt Controller - Peripheral device that manages interrupts for the processor
- Pending - Firing condition met and noticed but interrupt handler has not began to execute
- Interrupt Latency - Time from interrupt's firing condition being met and start of execution of interrupt handler
- Nested Interrupt - Occurs when one interrupt handler preempts another



#LEARN_IN DEPTH

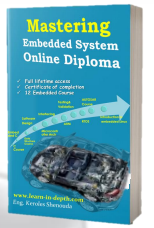
#Be_professional_in_embedded_system

Think In Depth Question



LEARN-IN-DEPTH
Be professional in embedded system

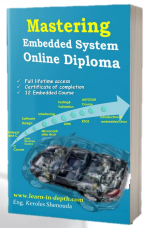
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Think In Depth What Happen…?

```
92 }
93 void EXTI0_IRQHandler(void)
94 {
95     //Toggle Led
96     GPIOA_ODR ^=  (1<<13) ;
97     //Clear the EXTI0 Pending Request
98 //  EXTI_PR |=  (1<<0) ;
99 }
100
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



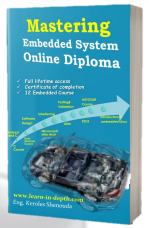
Think In Depth What Happen…?

main.c | startup.c | startup_stm32f103c6tx.s | main.c

```
63  {
64    GPIOA_CRL |= (1<<2) ;
65  }
66  GPIOA_Pin13_Output_init()
67  {
68    GPIOA_CRH   &= 0xFF0FFFFF;
69    GPIOA_CRH   |= 0x00200000;
70  }
71  int main(void)
72  {
73
74    clock_init ();
75    GPIOA_Pin0_Input_init ();
76    GPIOA_Pin13_Output_init();
77
78    //Select PortA For EXTI0
79    AFIO_EXTICR1 = 0x0 ;
80
81    //Rising trigger enabled
82    EXTI_RTSR |= (1<<0) ;
83
84    //Enable Mask EXTI0
85    EXTI_IMR |= (1<<0) ;
86
87    //Enable NVIC IRQ 6 (EXTI0)
88    NVIC_ISER0 |= (1<<6) ;
89
90    while(1) ;
91
92  }
93  void EXTI0_IRQHandler(void)
94  {
95    //Toggle Led
96    GPIOA_ODR ^=  (1<<13) ;
97    //Clear the EXTI0 Pending Request
98  //  EXTI_PR |=  (1<<0) ;
99  }
100
```

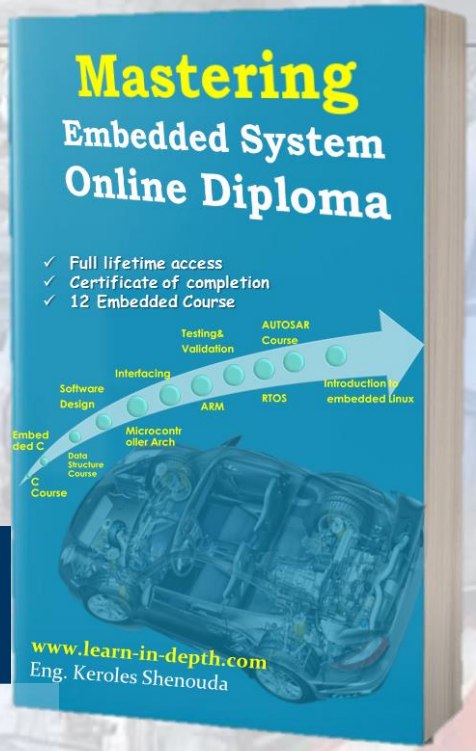
Infinite Loop

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

#Be_professional_in_embedded_system



Lab3: Interrupt Controller On Atmega32

ASSINGMENT (TRY TO SOLVE IT)

WILL BE SOLVED IN SECTION

LEARN-IN-DEPTH

Be professional in embedded system

https://www.learn-in-depth.com/

https://www.facebook.com/groups/embedded.system.KS/

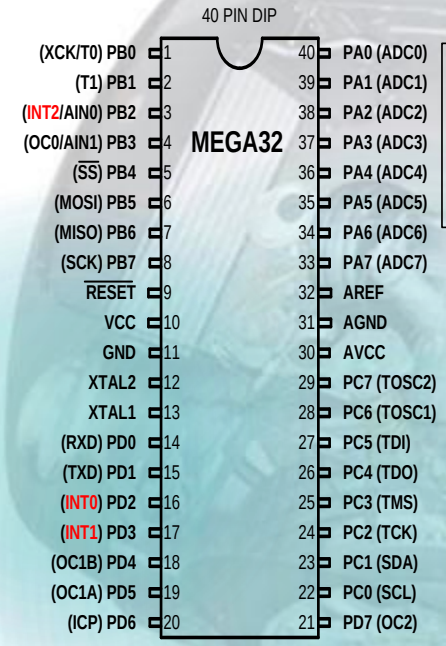
Interrupt Controller

TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0

TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0

GICR	INT1	INT0	INT2	-	-	-	IVSEL	IVCE

SREG	I	T	H	S	V	N	Z	C



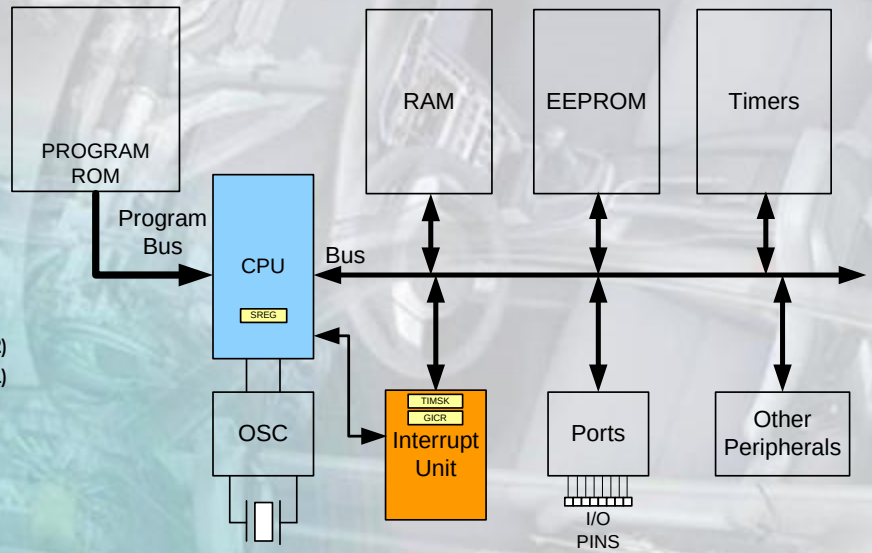
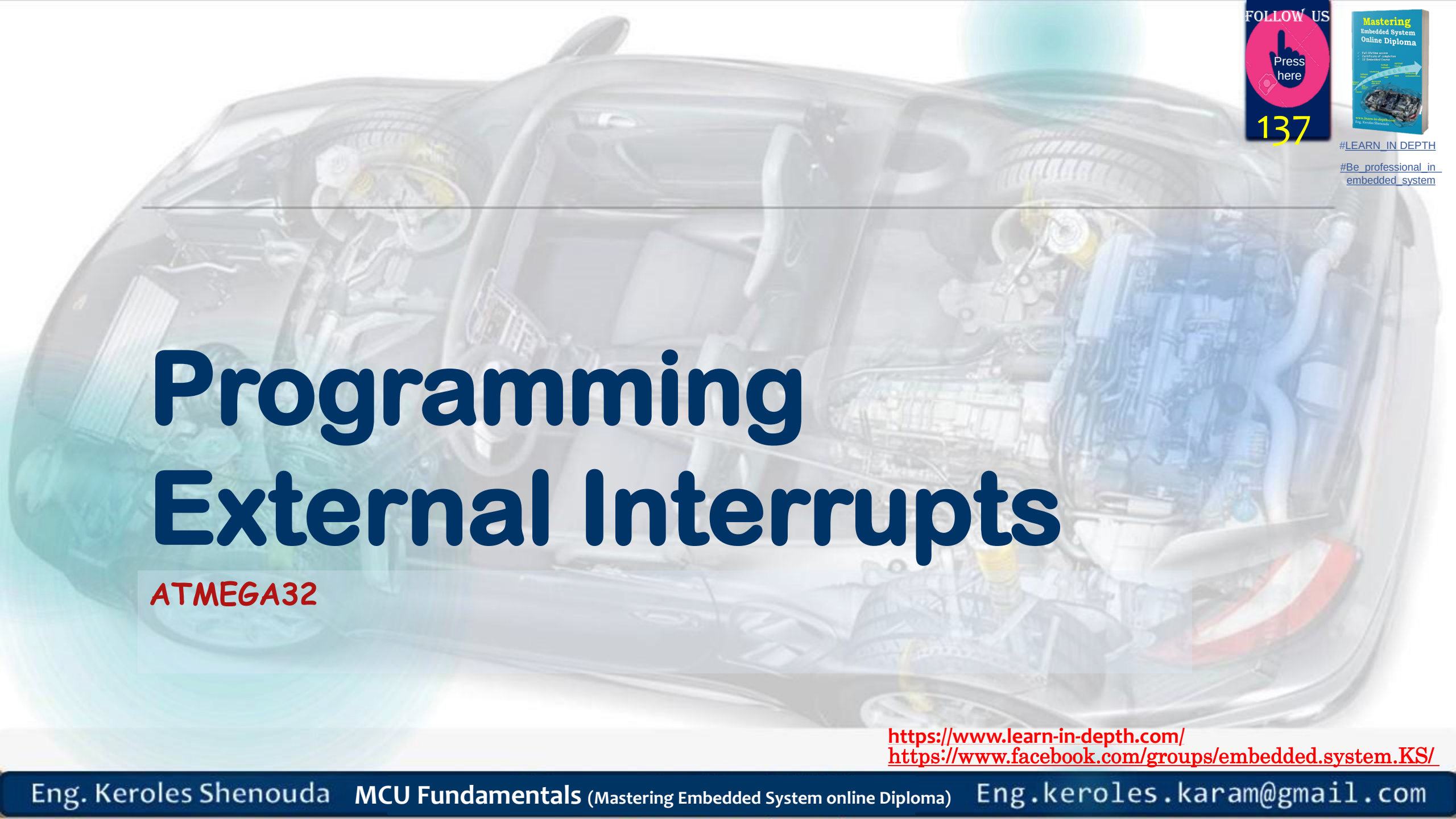


Table 10-1: Interrupt Vector Table for the Mega32

Interrupt	ROM Location (Hex)
Reset	0000
External Interrupt request 0	0002
External Interrupt request 1	0004
External Interrupt request 2	0006
Time/Counter2 Compare Match	0008
Time/Counter2 Overflow	000A
Time/Counter1 Capture Event	000C
Time/Counter1 Compare Match A	000E
Time/Counter1 Compare Match B	0010
Time/Counter1 Overflow	0012
Time/Counter0 Compare Match	0014
Time/Counter0 Overflow	0016
SPI Transfer complete	0018
USART, Receive complete	001A
USART, Data Register Empty	001C
USART, Transmit Complete	001E
ADC Conversion complete	0020
EEPROM ready	0022
Analog Comparator	0024
Two-wire Serial Interface	0026
Store Program Memory Ready	0028



Programming External Interrupts

ATMEGA32

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

16.1.1. MCUCR – MCU Control Register

Name: MCUCR
Offset: 0x35
Reset: 0
Property: When addressing I/O Registers as data space the offset address is 0x55

Bit	7	6	5	4	3	2	1	0
					ISC11	ISC10	ISC01	ISC00
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

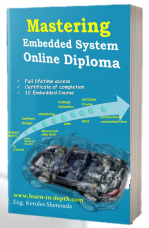
Table 16-1 Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

Table 16-2 Interrupt 0 Sense Control

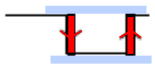
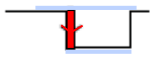
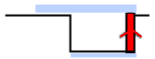
ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

Edge trigger Vs. Level trigger in external interrupts

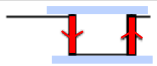
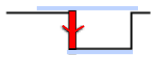


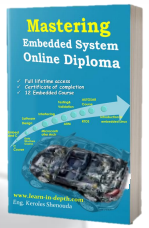
MCUCR	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00

ISC01, ISC00 (Interrupt Sense Control bits) These bits define the level or edge on the external INT0 pin that activates the interrupt, as shown in the following table:

ISC01	ISC00		Description
0	0		The low level of INT0 generates an interrupt request.
0	1		Any logical change on INT0 generates an interrupt request.
1	0		The falling edge of INT0 generates an interrupt request.
1	1		The rising edge of INT0 generates an interrupt request.

ISC11, ISC10 These bits define the level or edge that activates the INT1 pin.

ISC11	ISC10		Description
0	0		The low level of INT1 generates an interrupt request.
0	1		Any logical change on INT1 generates an interrupt request.
1	0		The falling edge of INT1 generates an interrupt request.
1	1		The rising edge of INT1 generates an interrupt request.



16.1.2. MCUCSR – MCU Control and Status Register

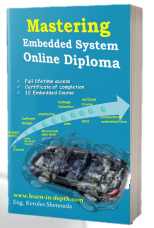
When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these offset addresses.

Name: MCUCSR
Offset: 0x34
Reset: 0
Property: When addressing I/O Registers as data space the offset address is 0x54

Bit	7	6	5	4	3	2	1	0
		ISC2						
Access		R/W						
Reset		0						

Bit 6 – ISC2: ISC2: Interrupt Sense Control 2

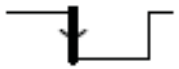
The Asynchronous External Interrupt 2 is activated by the external pin INT2 if the SREG I-bit and the corresponding interrupt mask in GICR are set. If ISC2 is written to zero, a falling edge on INT2 activates the interrupt. If ISC2 is written to one, a rising edge on INT2 activates the interrupt. Edges on INT2 are



Edge trigger Vs. Level trigger (Cont.)

MCUCSR	JTD	ISC2	-	JTRF	WDRF	BORF	EXTRF	PORF

ISC2 This bit defines whether the INT2 interrupt activates on the falling edge or the rising edge.

ISC2		Description
0		The falling edge of INT2 generates an interrupt request.
1		The rising edge of INT2 generates an interrupt request.

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

16.1.3. GICR – General Interrupt Control Register

When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these offset addresses.

Name: GICR
Offset: 0x3B
Reset: 0
Property: When addressing I/O Registers as data space the offset address is 0x5B

Bit	7	6	5	4	3	2	1	0
	INT1	INT0	INT2					
Access	R/W	R/W	R/W					
Reset	0	0	0					

Bit 7 – INT1: External Interrupt Request 1 Enable

When the INT1 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled. The Interrupt Sense Control1 bits 1/0 (ISC11 and ISC10) in the MCU general Control Register (MCUCR) define whether the external interrupt is activated on rising and/or falling edge of the INT1 pin or level sensed. Activity on the pin will cause an interrupt request even if INT1 is configured as an output. The corresponding interrupt of External Interrupt Request 1 is executed from the INT1 Interrupt Vector.

Bit 6 – INT0: External Interrupt Request 0 Enable

When the INT0 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled. The Interrupt Sense Control0 bits 1/0 (ISC01 and ISC00) in the MCU general Control Register (MCUCR) define whether the external interrupt is activated on rising and/or falling edge of the INT0 pin or level sensed. Activity on the pin will cause an interrupt request even if INT0 is configured as an output. The corresponding interrupt of External Interrupt Request 0 is executed from the INT0 Interrupt Vector.

16.1.4. GIFR – General Interrupt Flag Register

When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these offset addresses.

Name: GIFR
Offset: 0x3A
Reset: 0
Property: When addressing I/O Registers as data space the offset address is 0x5A

Bit	7	6	5	4	3	2	1	0
	INTF1	INTF0	INTF2					
Access	R/W	R/W	R/W					
Reset	0	0	0					

Bit 7 – INTF1: External Interrupt Flag 1

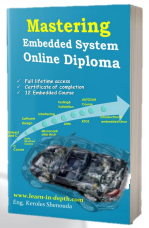
When an event on the INT1 pin triggers an interrupt request, INTF1 becomes set (one). If the I-bit in SREG and the INT1 bit in GICR are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it. This flag is always cleared when INT1 is configured as a level interrupt.

Bit 6 – INTF0: External Interrupt Flag 0

When an event on the INT0 pin triggers an interrupt request, INTF0 becomes set (one). If the I-bit in SREG and the INT0 bit in GICR are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it. This flag is always cleared when INT0 is configured as a level interrupt.

Bit 5 – INTF2: External Interrupt Flag 2

When an event on the INT2 pin triggers an interrupt request, INTF2 becomes set (one). If the I-bit in



10.3.1. SREG – The AVR Status Register

When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these offset addresses.

Name: SREG
Offset: 0x3F
Reset: 0x00
Property: When addressing I/O Registers as data space the offset address is 0x5F

Bit	7	6	5	4	3	2	1	0
	I	T	H	S	V	N	Z	C
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit 7 – I: Global Interrupt Enable

The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt

Interrupt priority

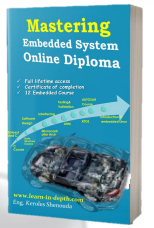


Table 10-1: Interrupt Vector Table for the Mega32

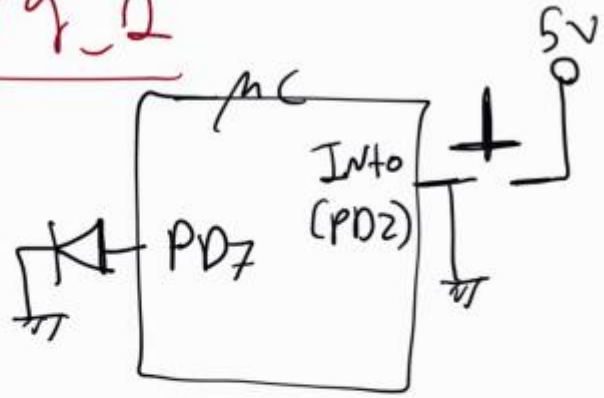
Interrupt	ROM Location (Hex)
Reset	0000
External Interrupt request 0	0002
External Interrupt request 1	0004
External Interrupt request 2	0006
Time/Counter2 Compare Match	0008
Time/Counter2 Overflow	000A
Time/Counter1 Capture Event	000C
Time/Counter1 Compare Match A	000E
Time/Counter1 Compare Match B	0010
Time/Counter1 Overflow	0012
Time/Counter0 Compare Match	0014
Time/Counter0 Overflow	0016
SPI Transfer complete	0018
USART, Receive complete	001A
USART, Data Register Empty	001C
USART, Transmit Complete	001E
ADC Conversion complete	0020
EEPROM ready	0022
Analog Comparator	0024
Two-wire Serial Interface	0026
Store Program Memory Ready	0028

Highest priority

Lowest priority

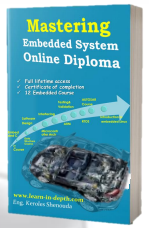
Conclusion by hand writing Simple task

task_irq_2

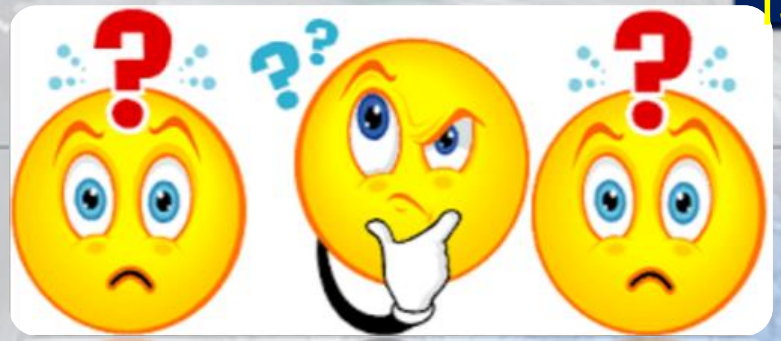



```
#include <avr/interrupt.h>
#  //   <util/delay.h>
//  //   <avr/io.h>
```

```
ISR (INT0_vect) // to
// store ISR at address of INT0
{ PORTD |= (1<<7);
  _delay_ms (200);
  PORTD &= ~(1<<7);
} _delay_ms (200);

int main ( )
{ DDRD |= (1<<7);
  DDRD &= ~(1<<2);
  SREG |= 0x80;
  // enable global irq.

GICR |= 0x40;
// enable INT0

MCUCR |= 0x01;
// Logical Change level
// triggre for INT0

} while(1)
  {
  }
}
```



Interrupt Controller_LAB1

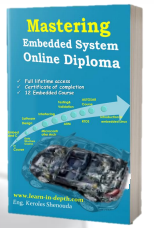


Write C Code using the 3 eternal interrupts

External Interrupt 0 (INT0) - PD2. >> irq occur when "any logical change"
External Interrupt 1 (INT1) - PD3. >> irq occur when "rising edge"
External Interrupt 2 (INT2) - PB2. >> irq occur when "Falling edge"

We have also 3 leds (PD5,6,7) (led0,1,2).
Each interrupt just make the led ON for 1 sec
The main function is always make all the leds off

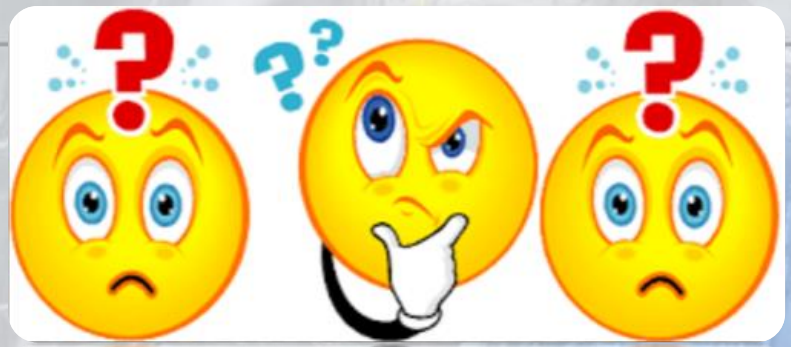
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

Interrupt Controller_LAB1
For example



IRQ0 happen → PC jump to ISR for IRQ0

↓

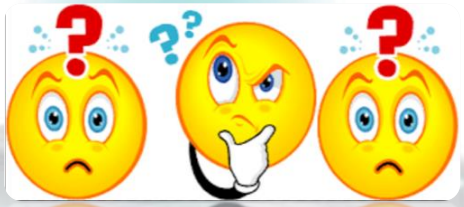
LED 0 ON for 1sec

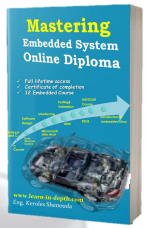
↓

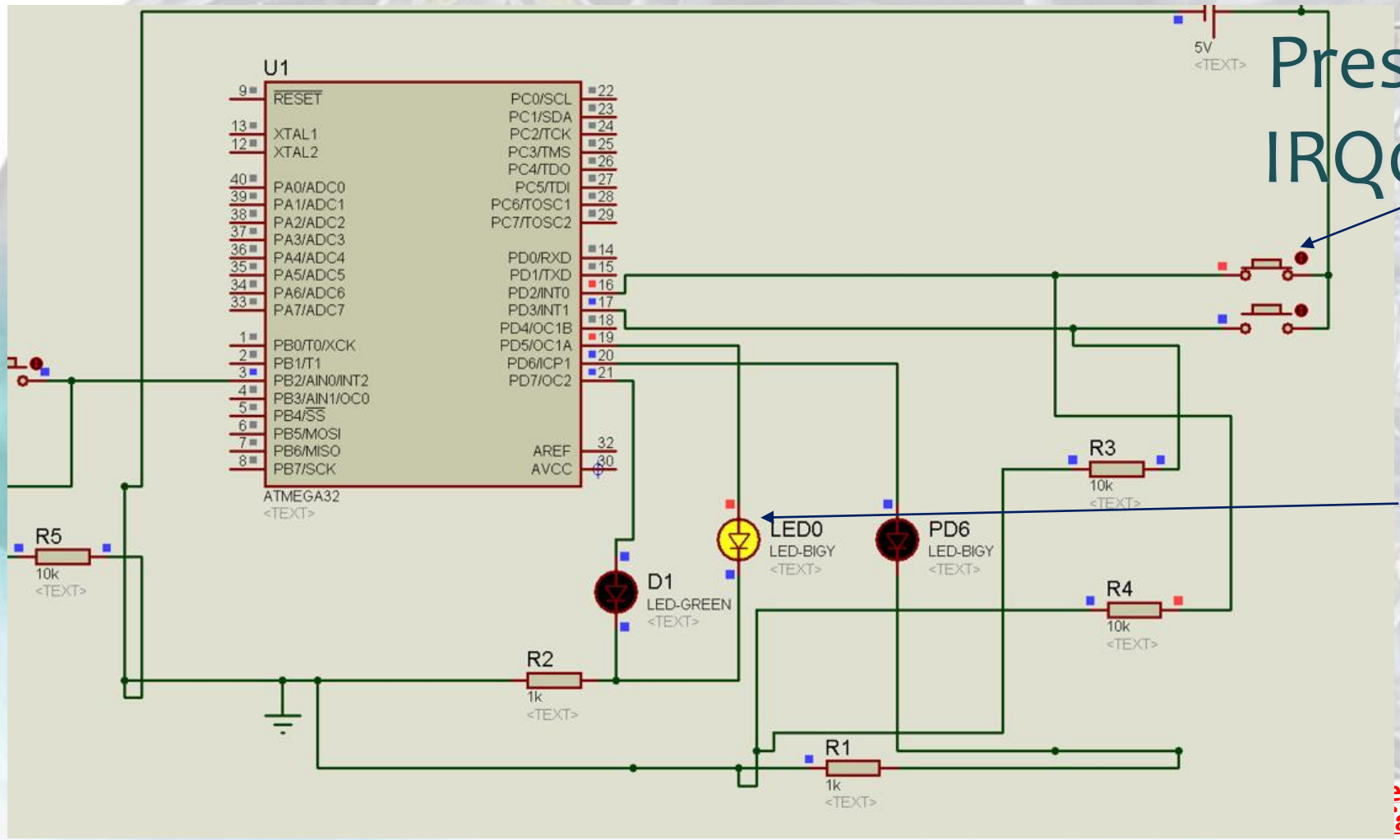
Return to main and all LEDs Being OFF

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Interrupt Controller_LAB1
For example





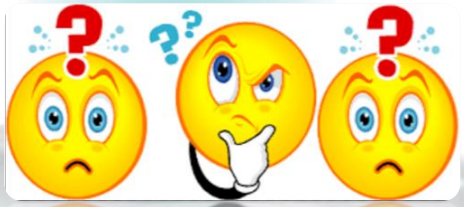


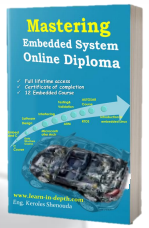
Press on Bush Botton IRQ0 and still pressing (rising edge)

The ledo is on For 1 sec

earn-in-depth.com/
acebook.com/groups/embedded.system.KS/

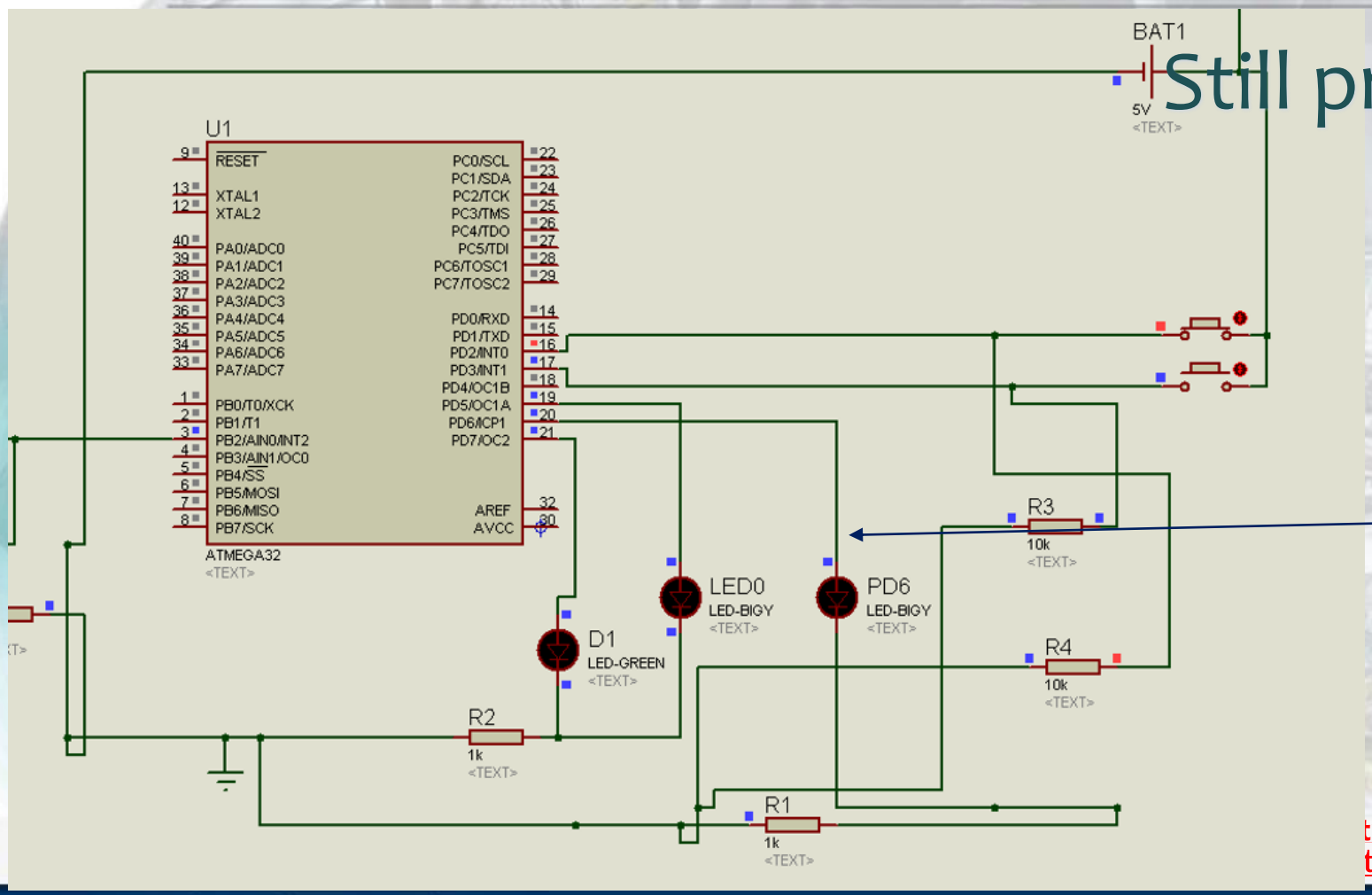
Interrupt Controller_LAB1
For example





#LEARN_IN DEPTH

#Be_professional_in embedded_system

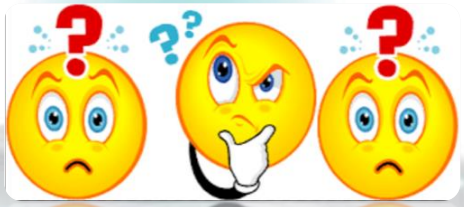


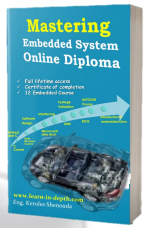
Still pressing on Push Button

The ledo is OFF
After 1 sec

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

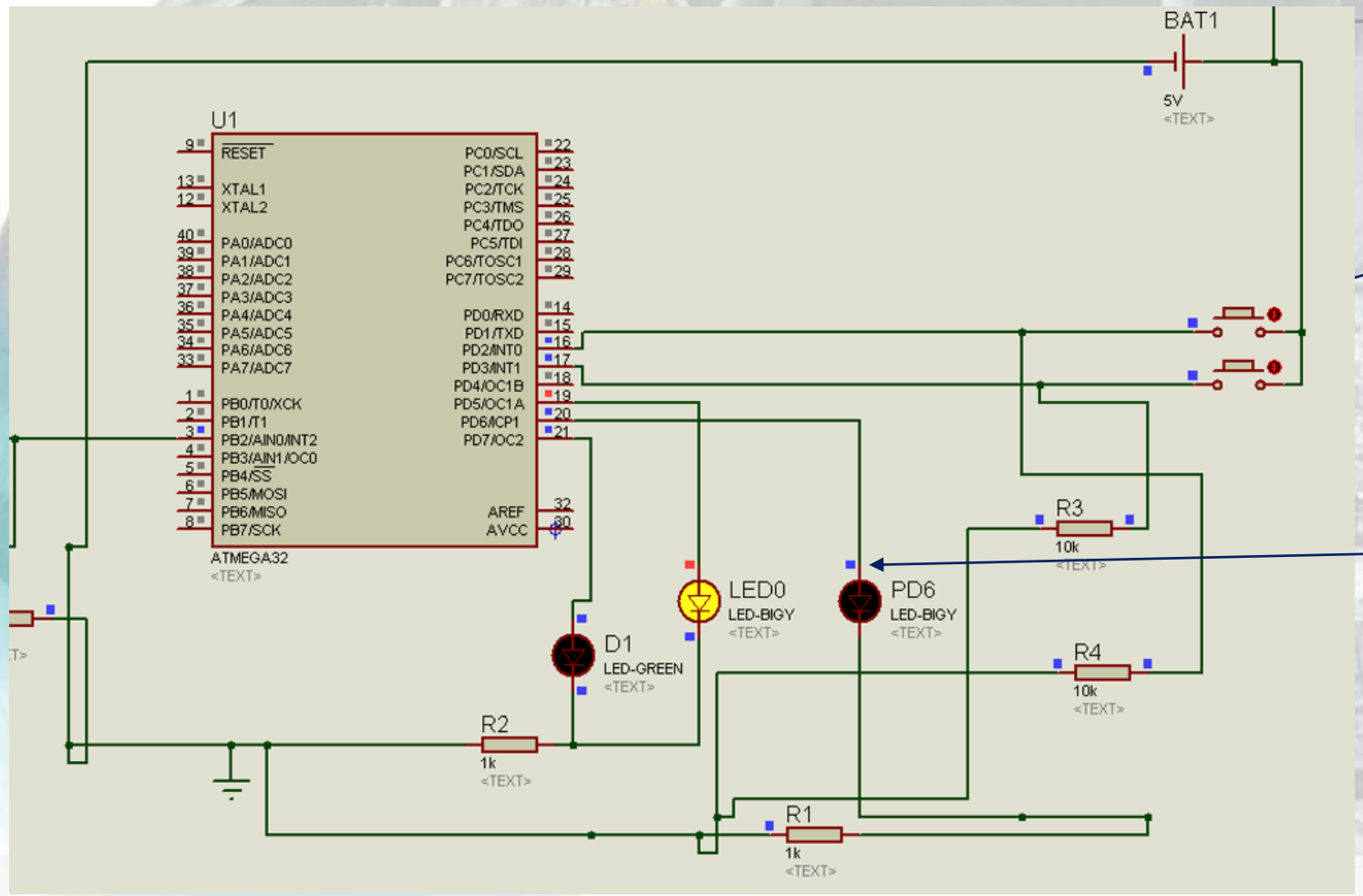
Interrupt Controller_LAB1
For example





#LEARN_IN DEPTH

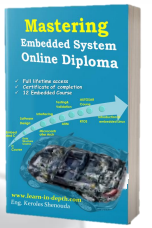
#Be_professional_in_embedded_system



Released so IRQo happen
Again because
(Falling edge)
And you already configured
IRQo to happen if any logical change happen

The ledo is ON
for1 sec

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

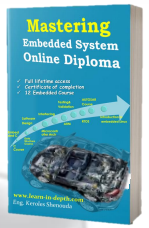
#Be_professional_in_embedded_system

When irq0 is pressed

```
AVR Source Code - U1
main.c
----  /*
----   * main.c
----   *
----   *  Created on: Aug 4, 2017
----   *      Author: Keroles Shenouda
----   *      https://www.facebook.com/groups/embedded.system.
----   */
----
---- #define F_CPU 1000000ul
---- #include "util/delay.h"
---- #include <avr/io.h>
---- #include <avr/interrupt.h>
----
----
---- ISR(INT0_vect)
1B2C {
1B2C PUSH    R1
1B2E PUSH    R0
1B30 IN      R0,$3F
1B32 PUSH    R0
1B34 CLR     R1
1B36 PUSH    R18
1B38 PUSH    R19
1B3A PUSH    R20
1B3C PUSH    R21
1B3E PUSH    R22
1B40 PUSH    R23
1B42 PUSH    R24
1B44 PUSH    R25
1B46 PUSH    R26
1B48 PUSH    R27
1B4A PUSH    R30
1B4C PUSH    R31
1B4E PUSH    R29
1B50 PUSH    R28
1B52 IN      R28,$3D
1B54 IN      R29,$3E
1B56 SBIW    R29:R28,0E
1B58 OUT     $3E,R29
1B5A OUT     $3D,R28
----         //DDRD = 0xff;
1B5C         PORTD |= 1<<5;
1B5C LDI     R26,$32
1B5E LDI     R27,$00
1B60 LDI     R30,$32
1B62 LDI     R31,$00
1B64 LD      R24,Z
1B66 ORI     R24,$20
1B68 ST      X,R24
1B6A LDI     R24,$00
1B6C LDI     R25,$00
1B6E LDI     R26,$7A
1B70 LDI     R27,$44
1B72 ST      Y+11,R24
1B74 ST      Y+12,R25
1B76 ST      Y+13,R26
1B78 ST      Y+14,R27
----         _delay_ms(1000);
1C4E }
1C4E ADIW    R29:R28,14
1C50 OUT     $3E,R29
1C52 OUT     $3D,R28
1C54 POP     R28
1C56 POP     R29
1C58 POP     R31
1C5A POP     R30
1C5C POP     R27
1C5E POP     R26
1C60 POP     R25
1C62 POP     R24
```

```
AVR CPU Registers - U1
PC       INSTRUCTION
1B2C      PUSH R1

SREG     ITHSVNZC  CYCLE COUNT
00       00000000   32800008

R00:80  R08: 00  R16:  00 R24:  81
R01:00  R09: 00  R17:  01 R25:  02
R02:00  R10: 00  R18:  10 R26:  00
R03:00  R11: 00  R19:  27 R27:  00
R04:00  R12: 00  R20:  00 R28:  4D
R05:00  R13: 00  R21:  00 R29:  08
R06:00  R14: 00  R22:  10 R30:  02
R07:00  R15: 00  R23:  27 R31:  08

X:0000  Y: 084D  Z:  0802 S:  084B
```

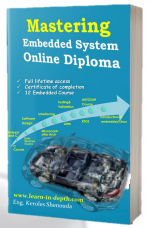
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



References

- https://docs.google.com/viewer?a=v&pid=sites&srcid=ZmtlLnV0bS5teXxyaWR6dWFuLXMtd2Vic2l0ZXxneDo2ODU0NzlkM2JkOTg4MjRk
- http://www.avrprojects.net/index.php/avr-projects/sensors/38-humidity-and-temperature-sensor-dht11?showall=&start=1
- http://www.cse.wustl.edu/~lu/cse467s/slides/dsp.pdf
- http://www.avr-tutorials.com/
- **Microprocessor: ATmega32 (SEE3223-10)**
  http://ridzuan.fke.utm.my/microprocessor-atmega32-see3223-10
- **http://circuitdigest.com/article/what-is-the-difference-between-microprocessor-and-microcontroller**
- **http://cs4hs.cs.pub.ro/wiki/roboticsisfun/chapter2/ch2_7_programming_a_microcontroller**
- Embedded Systems with ARM **Cortex-M Microcontrollers in** Assembly Language and C Dr. Yifeng Zhu Third edition June 2018

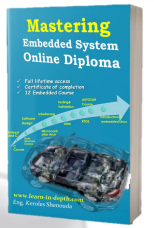
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



References

- http://techdifferences.com/difference-between-interrupt-and-polling-in-os.html
- http://www.bogotobogo.com/Embedded/hardware_interrupt_software_interrupt_latency_irq_vs_fiq.php
- Preventing Interrupt Overload Presented by Jiyong Park Seoul National University, Korea 2005. 2. 22. John Regehr, Usit Duogsaa, School of Computing, University.
- First Steps Embedded Systems Byte Craft Limited reference
- COMPUTER ORGANIZATION AND ARCHITECTURE DESIGNING FOR PERFORMANCE EIGHTH EDITION **William Stallings**
- **Getting Started with the Tiva™ TM4C123G LaunchPad Workshop**

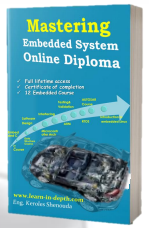
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



References

- Tiva™ TM4C123GH6PM Microcontroller DATA SHEET
- Interrupts and Exceptions COMS W6998 Spring 2010
- THE **AVR MICROCONTROLLER**. AND EMBEDDED SYSTEMS Using Assembly and C. Muhammad Ali **Mazidi**.

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



References

- https://docs.google.com/viewer?a=v&pid=sites&srcid=ZmtlLnV0bS5teXxyaWR6dWFuLXMtd2Vic2l0ZXxneDo2ODU0NzlkM2JkOTg4MjRk
- http://www.avrprojects.net/index.php/avr-projects/sensors/38-humidity-and-temperature-sensor-dht11?showall=&start=1
- http://www.cse.wustl.edu/~lu/cse467s/slides/dsp.pdf
- http://www.avr-tutorials.com/
- Microprocessor: ATmega32 (SEE3223-10) http://ridzuan.fke.utm.my/microprocessor-atmega32-see3223-10
- http://circuitdigest.com/article/what-is-the-difference-between-microprocessor-and-microcontroller
- AVR Microcontroller and Embedded Systems: Using Assembly and C (Pearson Custom Electronics Technology) 1st Edition https://www.amazon.com/AVR-Microcontroller-Embedded-Systems-Electronics/dp/0138003319

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN_IN DEPTH

#Be_professional_in_embedded_system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/