Interrupt Controller

Paolo Bernardi

Input/Output system management

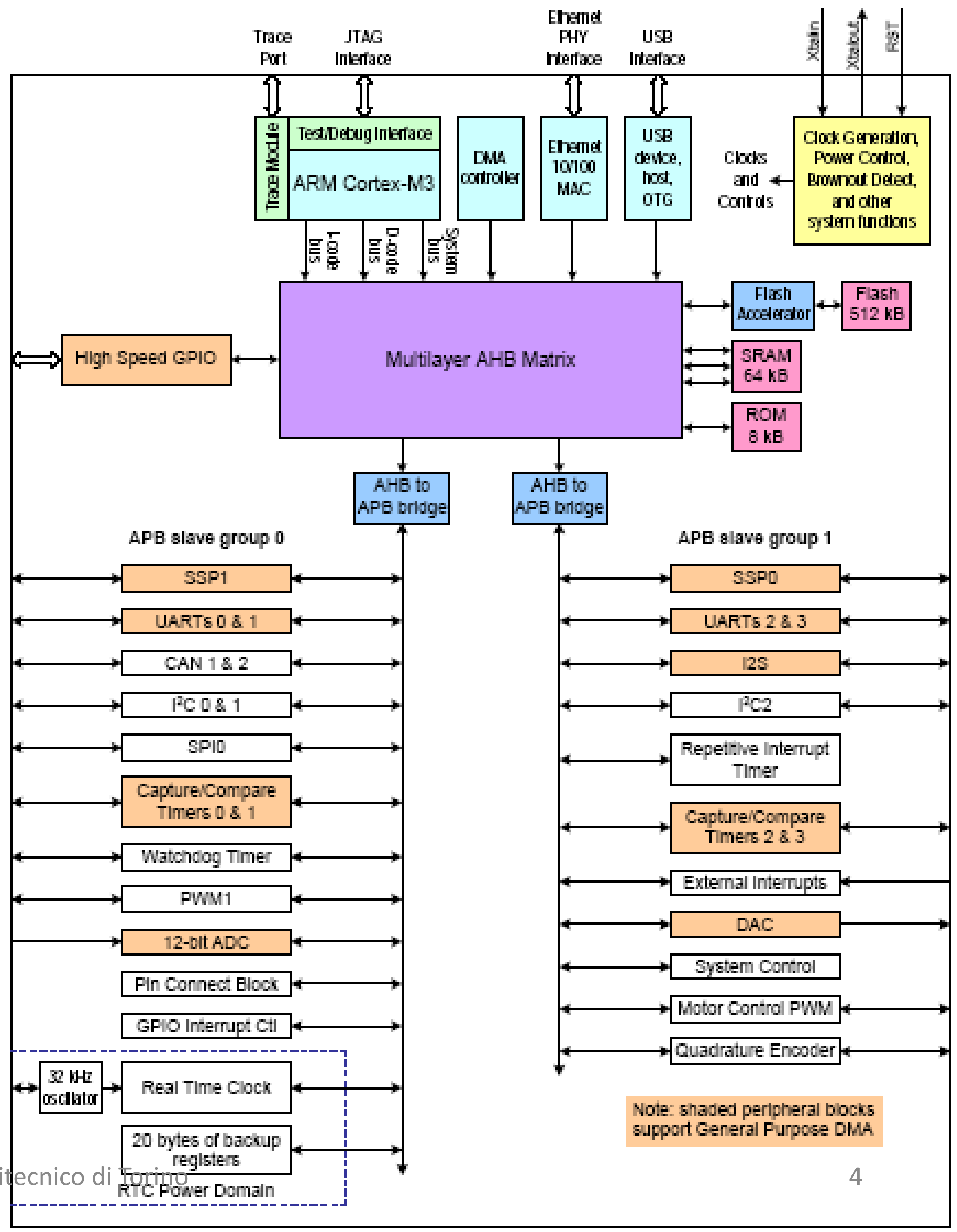
- The main function of a Input/Output (I/O) system is to exchange information with the external world.
- An I/O system needs to be controlled by the CPU which has to incercept service requests
  - For example if a new data is received by a peripheral core
- There are 2 main methods used to manage service requests:
  - <u>Polling</u>
  - <u>Interrupt.</u>

System event categories

- Respond to infrequent but important events
  - Alarm conditions like low battery power (i.e., NMI)
  - Error conditions (i.e., USAGE FAULT)
- I/O synchronization
  - Trigger interrupt when signal on a port changes
- Periodic interrupts
  - Generated by the timer at a regular rate
  - Systick timer can generate interrupt when it hits zero
  - Reload value + frequency determine interrupt rate
- Data acquisition samples ADC

Block diagram NXP LPC1768

- I/O system composed of several peripheral cores
  - Serial ports – UARTS
  - 12-bit ADC / DAC
  - GPIO
  - Timers
  - Other communication protocols like
    - I2C
    - SPI/SSP



Polling

- Polling is the process where the computer or controlling device waits for an external device to check for its readiness or state
  - Checking status registers (best practice)
  - Checking data registers.
- The polling is often implemented as a software cycle
  - Performing a predefined sequence of checks at regular time
  - A scheduling can be defined to access peripheral cores more or less frequently
- If the polled core needs to be handled, the CPU moves from the polling loop to the handler of the specific event.
- Main characteristics
  - The most of the time is spent in the software cycle (disadvantage – power inefficient)
  - Easy to implement (advantage)
  - High latency in the handling (disadvantage – low performance)
  - Difficult management of nested requests (disadvantage – very low performance)

Interrupt

- Peripheral devices are directly interacting with the CPU,
  - CPU is no more implementing a sw polling loop,
  - Idle mode can be entered,
  - The system wakes up as soon as a peripheral core is requesting a service
- When a request is received, the CPU needs to recognize the source of request in order to execute the proper handler
- Current architectures implements a Vectored Interrupt management method
  - Based on the Interrupt Vector Table (IVT)
  - The CPU collaborates with an external device called Interrupt Controller

System setup for interrupt mode

- Things you must do in programming a system to use interrupts
- BOOT TIME
    - Initialize data structures
        - counters, pointers
        - Eventually specify a flag variable that may interrupt (semaphores)
    - Configure Interrupt Controller
        - Enable interrupt sources
        - Set priority of every source

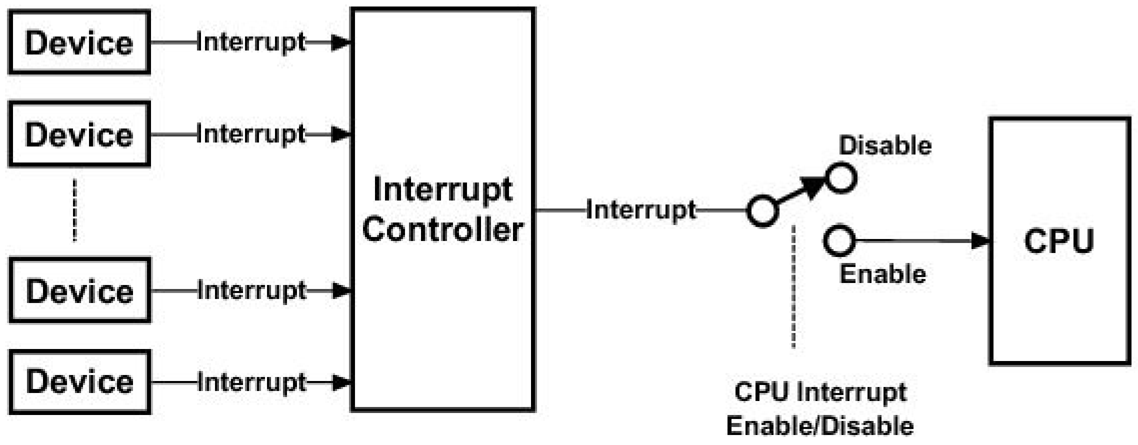
System setup for interrupt mode (II)

- Things you must do in every interrupt service routine
- RUNTIME
  - Acknowledge
    - Clear the flags that indicate the interrupt is active
    - Can be done in different parts of the interrupt service routine
  - Maintain contents of R4-R8,R10-R11 (ABI AAPCS)
  - Communicate via shared global variables.

May be important for nesting interruptions

Interrupt controller

- In computing, an interrupt controller is a device that is used to combine several sources of interrupt onto one or more CPU lines, while allowing priority levels to be assigned to its interrupt outputs
- Manages interrupt signals received from devices by combining multiple interrupts into a single interrupt output.



Nested Vectored Interrupt Controller (NVIC) (UM pg. 73)

- The Nested Vectored Interrupt Controller (NVIC) is an integral part of the Cortex-M3.
- The tight coupling to the CPU allows for low interrupt latency and efficient processing of late arriving interrupts
- It manages 35 possible external interrupt.

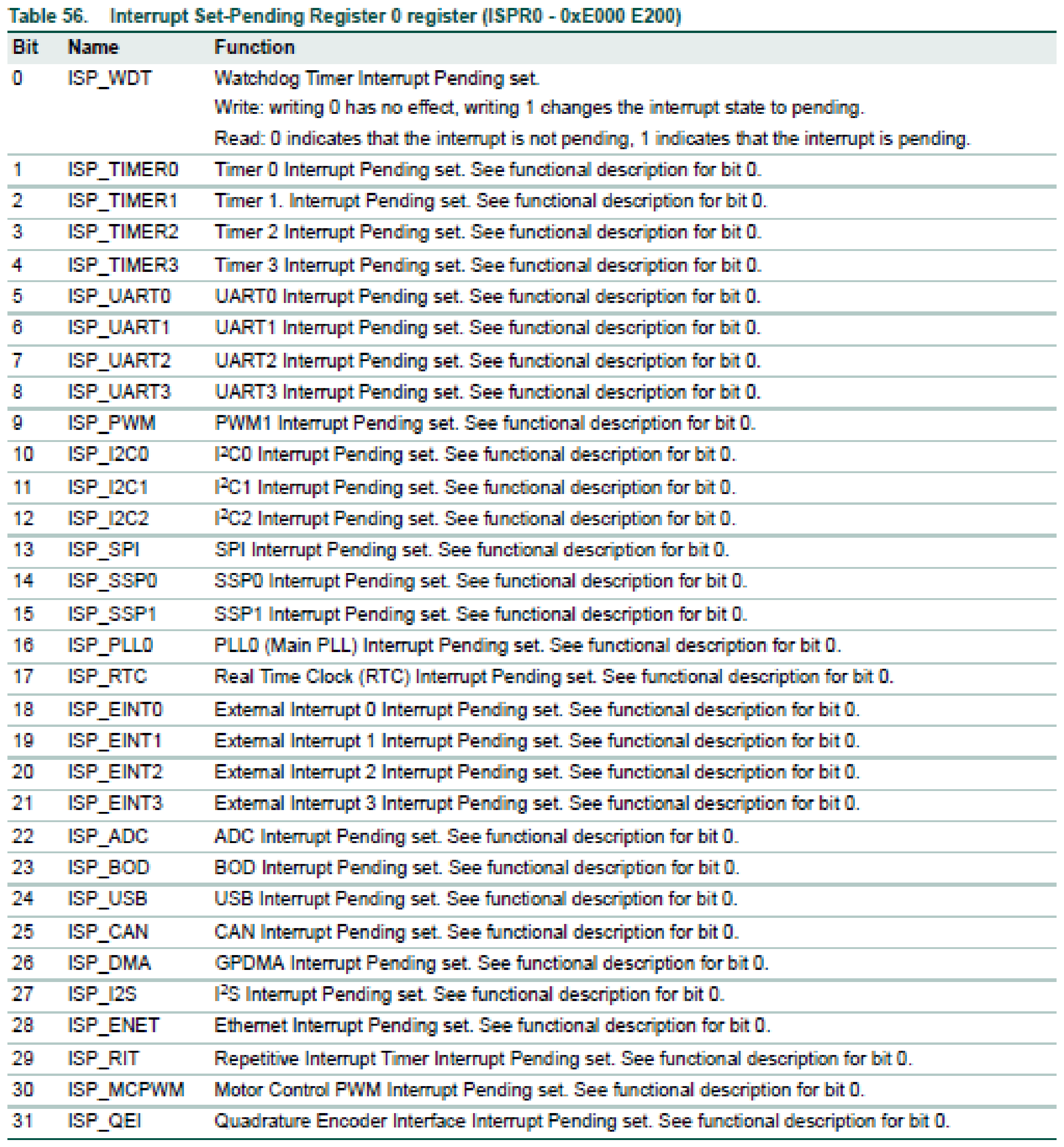
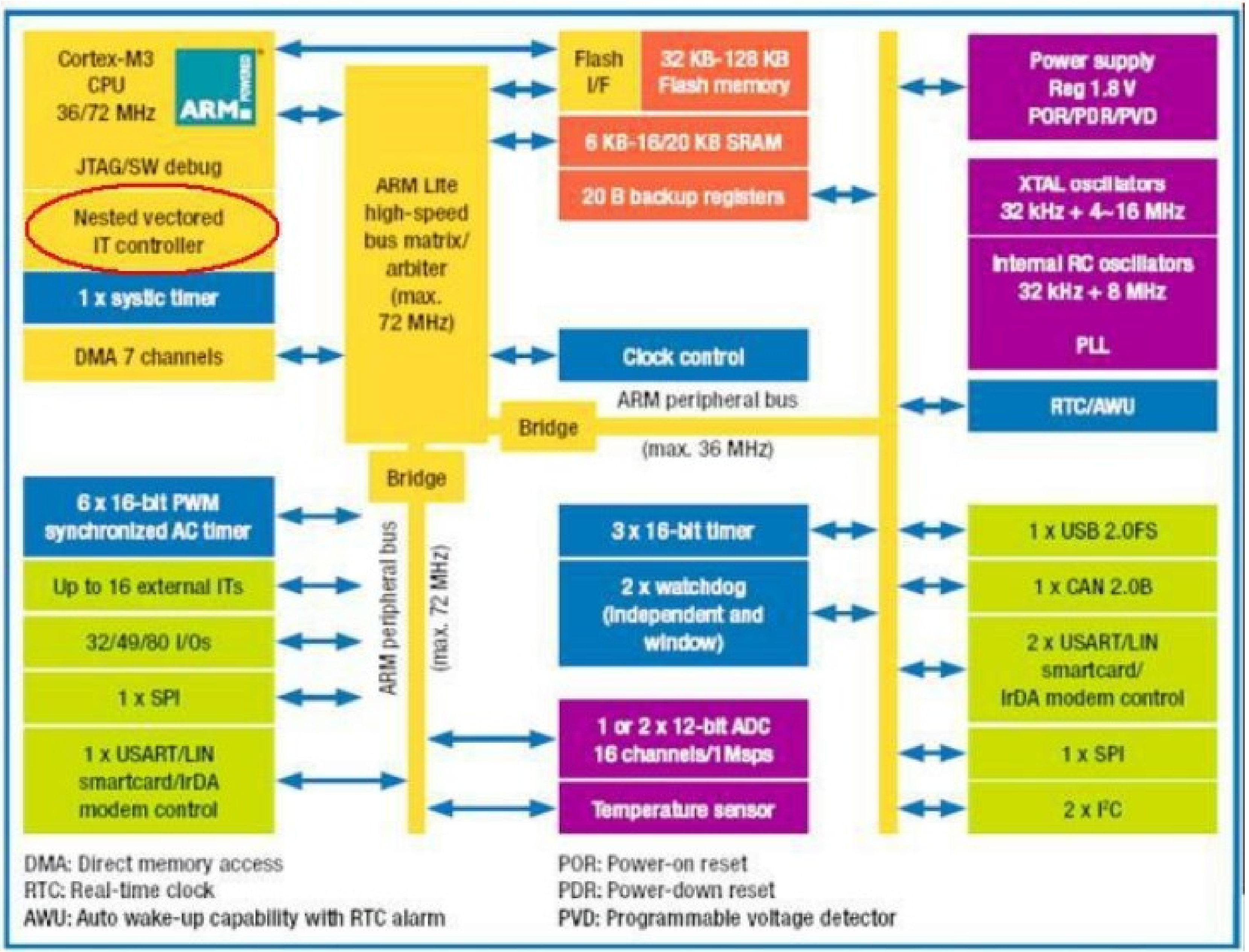


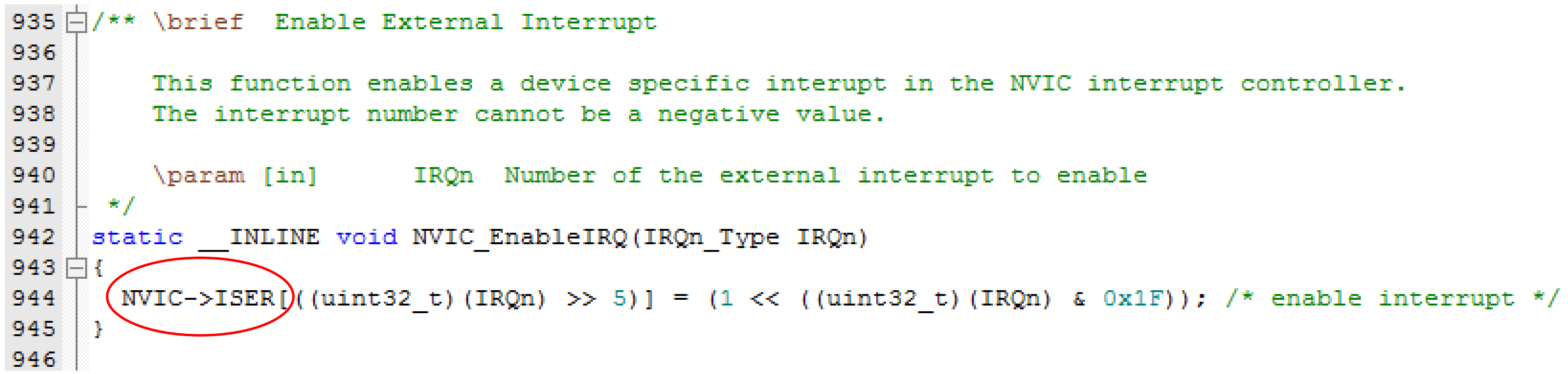
Table 56. Interrupt Set-Pending Register 0 register (ISPR0 - 0xE000 E200)

Bit	Name	Function
0	ISP_WDT	Watchdog Timer Interrupt Pending set. Write: writing 0 has no effect, writing 1 changes the interrupt state to pending. Read: 0 indicates that the interrupt is not pending, 1 indicates that the interrupt is pending.
1	ISP_TIMER0	Timer 0 Interrupt Pending set. See functional description for bit 0.
2	ISP_TIMER1	Timer 1. Interrupt Pending set. See functional description for bit 0.
3	ISP_TIMER2	Timer 2 Interrupt Pending set. See functional description for bit 0.
4	ISP_TIMER3	Timer 3 Interrupt Pending set. See functional description for bit 0.
5	ISP_UART0	UART0 Interrupt Pending set. See functional description for bit 0.
6	ISP_UART1	UART1 Interrupt Pending set. See functional description for bit 0.
7	ISP_UART2	UART2 Interrupt Pending set. See functional description for bit 0.
8	ISP_UART3	UART3 Interrupt Pending set. See functional description for bit 0.
9	ISP_PWM	PWM1 Interrupt Pending set. See functional description for bit 0.
10	ISP_I2C0	I^{2}C0 Interrupt Pending set. See functional description for bit 0.
11	ISP_I2C1	I^{2}C1 Interrupt Pending set. See functional description for bit 0.
12	ISP_I2C2	I^{2}C2 Interrupt Pending set. See functional description for bit 0.
13	ISP_SPI	SPI Interrupt Pending set. See functional description for bit 0.
14	ISP_SSP0	SSP0 Interrupt Pending set. See functional description for bit 0.
15	ISP_SSP1	SSP1 Interrupt Pending set. See functional description for bit 0.
16	ISP_PLL0	PLL0 (Main PLL) Interrupt Pending set. See functional description for bit 0.
17	ISP_RTC	Real Time Clock (RTC) Interrupt Pending set. See functional description for bit 0.
18	ISP_EINT0	External Interrupt 0 Interrupt Pending set. See functional description for bit 0.
19	ISP_EINT1	External Interrupt 1 Interrupt Pending set. See functional description for bit 0.
20	ISP_EINT2	External Interrupt 2 Interrupt Pending set. See functional description for bit 0.
21	ISP_EINT3	External Interrupt 3 Interrupt Pending set. See functional description for bit 0.
22	ISP_ADC	ADC Interrupt Pending set. See functional description for bit 0.
23	ISP_BOD	BOD Interrupt Pending set. See functional description for bit 0.
24	ISP_USB	USB Interrupt Pending set. See functional description for bit 0.
25	ISP_CAN	CAN Interrupt Pending set. See functional description for bit 0.
26	ISP_DMA	GPDMA Interrupt Pending set. See functional description for bit 0.
27	ISP_I2S	I^{2}S Interrupt Pending set. See functional description for bit 0.
28	ISP_ENET	Ethernet Interrupt Pending set. See functional description for bit 0.
29	ISP_RIT	Repetitive Interrupt Timer Interrupt Pending set. See functional description for bit 0.
30	ISP_MCPWM	Motor Control PWM Interrupt Pending set. See functional description for bit 0.
31	ISP_QEI	Quadrature Encoder Interface Interrupt Pending set. See functional description for bit 0.

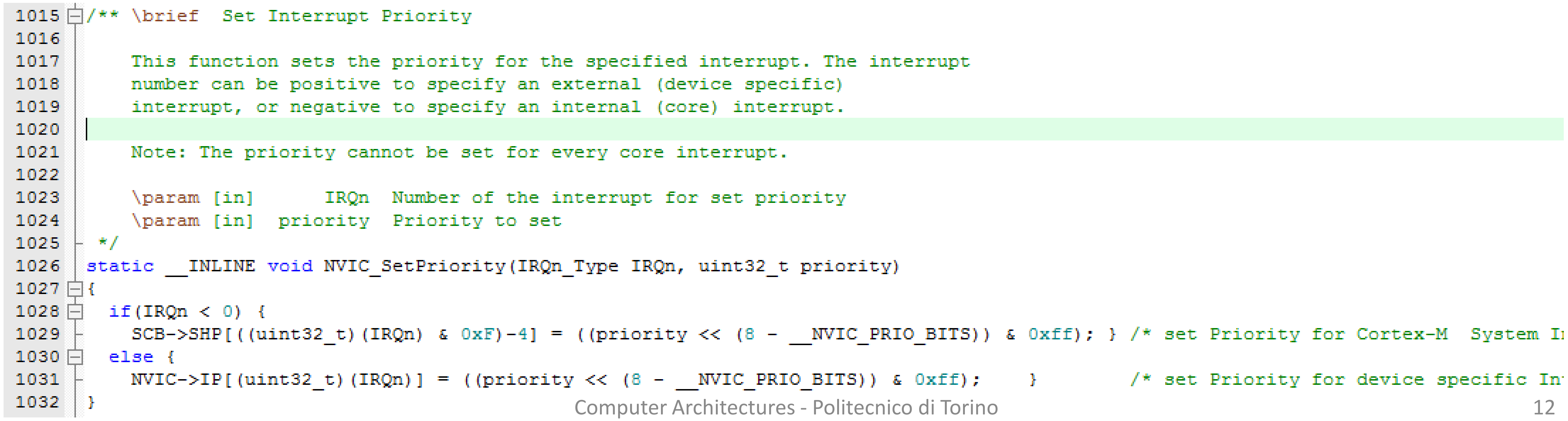


Cortex-M3 CPU 36/72 MHz
ARM
JTAG/SW debug
Nested vectored IT controller
1 x systic timer
DMA 7 channels
ARM Lite high-speed bus matrix/ arbiter (max. 72 MHz)
Flash I/F
32 KB-128 KB Flash memory
6 KB-16/20 KB SRAM
20 B backup registers
Clock control
ARM peripheral bus
(max. 36 MHz)
Bridge
Bridge
ARM peripheral bus
(max. 72 MHz)
Power supply Reg 1.8 V POR/PDR/PVD
XTAL oscillators 32 kHz + 4~16 MHz
Internal RC oscillators 32 kHz + 8 MHz
PLL
RTC/AWU
6 x 16-bit PWM synchronized AC timer
Up to 16 external ITs
32/49/80 I/Os
1 x SPI
1 x USART/LIN smartcard/IrDA modem control
3 x 16-bit timer
2 x watchdog (independent and window)
1 or 2 x 12-bit ADC 16 channels/1 Msps
Temperature sensor
1 x USB 2.0FS
1 x CAN 2.0B
2 x USART/LIN smartcard/ IrDA modem control
1 x SPI
2 x I²C
DMA: Direct memory access
RTC: Real-time clock
AWU: Auto wake-up capability with RTC alarm
POR: Power-on reset
PDR: Power-down reset
PVD: Programmable voltage detector

Library functions in core_cm3.h



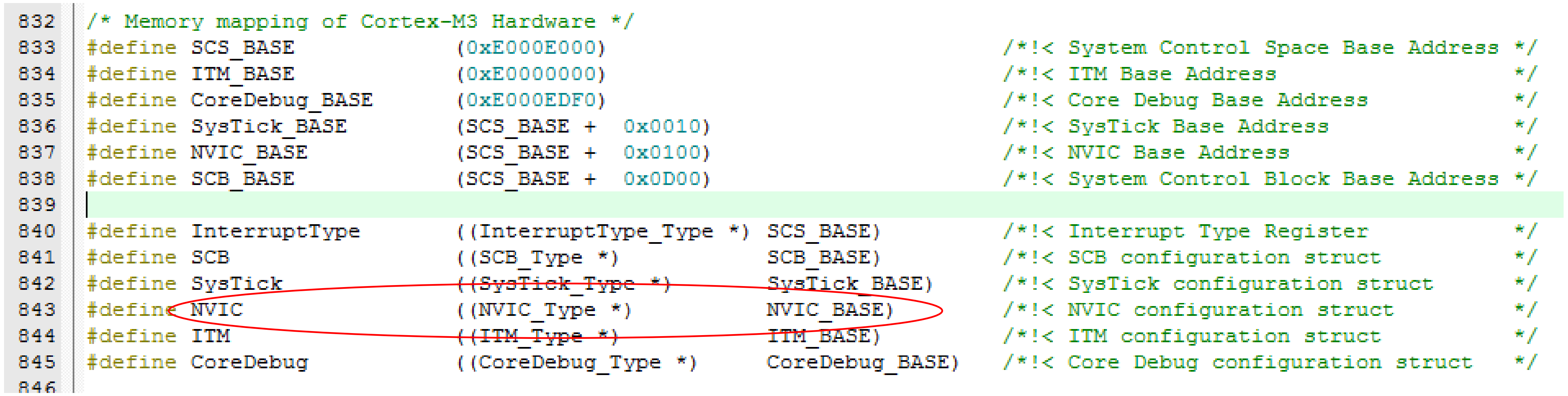
```
/** \brief  Enable External Interrupt

    This function enables a device specific interrupt in the NVIC interrupt controller.
    The interrupt number cannot be a negative value.

    \param [in]      IRQn  Number of the external interrupt to enable
 */
static __INLINE void NVIC_EnableIRQ(IRQn_Type IRQn)
{
  NVIC->ISER[((uint32_t)(IRQn) >> 5)] = (1 << ((uint32_t)(IRQn) & 0x1F)); /* enable interrupt */
}

```



```
/** \brief  Set Interrupt Priority

    This function sets the priority for the specified interrupt. The interrupt
    number can be positive to specify an external (device specific)
    interrupt, or negative to specify an internal (core) interrupt.

    Note: The priority cannot be set for every core interrupt.

    \param [in]      IRQn  Number of the interrupt for set priority
    \param [in]  priority  Priority to set
 */
static __INLINE void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)
{
  if(IRQn < 0) {
    SCB->SHP[((uint32_t)(IRQn) & 0xF)-4] = ((priority << (8 - __NVIC_PRIO_BITS)) & 0xff); } /* set Priority for Cortex-M  System I
  else {
    NVIC->IP[(uint32_t)(IRQn)] = ((priority << (8 - __NVIC_PRIO_BITS)) & 0xff);    }        /* set Priority for device specific In
}
```

NVIC Constant and addresses

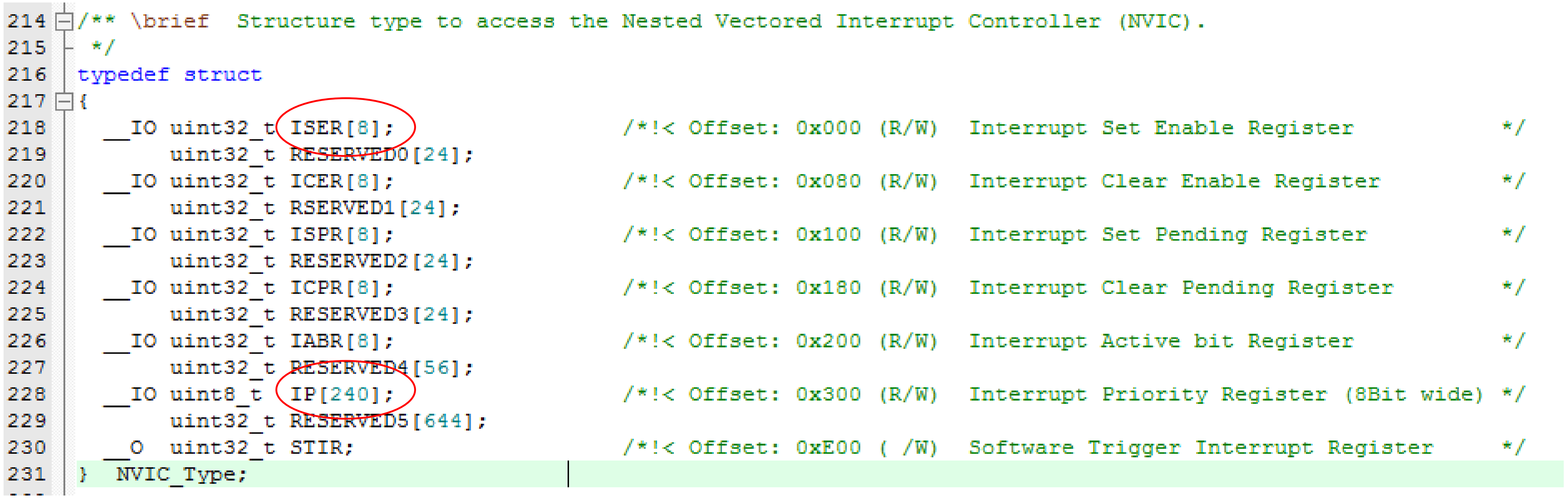
- core_cm3.h



```
/* Memory mapping of Cortex-M3 Hardware */
#define SCS_BASE            (0xE000E000)                                /*!< System Control Space Base Address */
#define ITM_BASE            (0xE0000000)                                /*!< ITM Base Address                  */
#define CoreDebug_BASE      (0xE000EDF0)                                /*!< Core Debug Base Address           */
#define SysTick_BASE        (SCS_BASE +  0x0010)                        /*!< SysTick Base Address              */
#define NVIC_BASE           (SCS_BASE +  0x0100)                        /*!< NVIC Base Address                 */
#define SCB_BASE            (SCS_BASE +  0x0D00)                        /*!< System Control Block Base Address */

#define InterruptType       ((InterruptType_Type *) SCS_BASE)           /*!< Interrupt Type Register           */
#define SCB                 ((SCB_Type *)           SCB_BASE)           /*!< SCB configuration struct          */
#define SysTick             ((SysTick_Type *)       SysTick_BASE)       /*!< SysTick configuration struct      */
#define NVIC                ((NVIC_Type *)          NVIC_BASE)          /*!< NVIC configuration struct         */
#define ITM                 ((ITM_Type *)           ITM_BASE)           /*!< ITM configuration struct          */
#define CoreDebug           ((CoreDebug_Type *)     CoreDebug_BASE)     /*!< Core Debug configuration struct   */
```

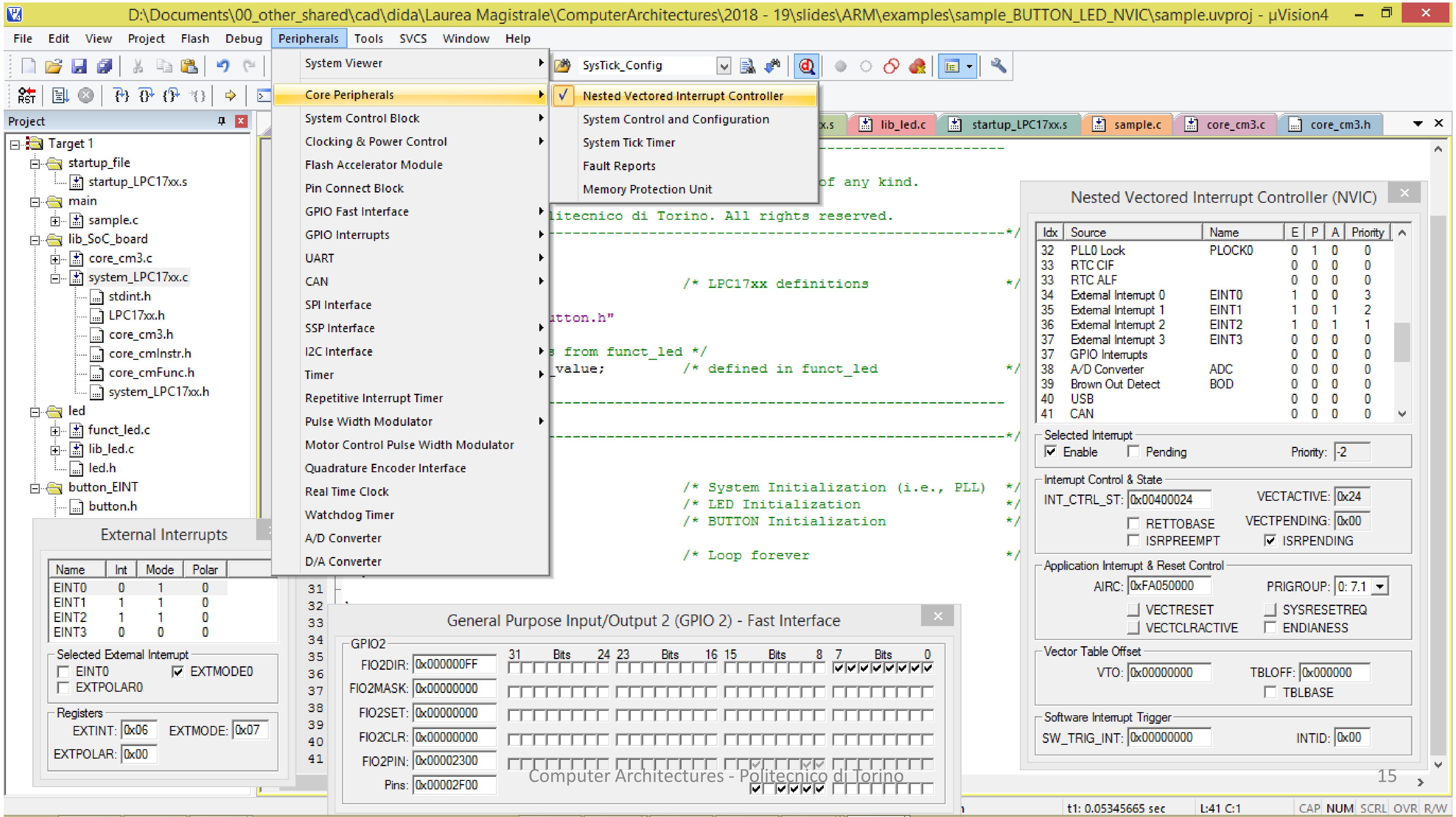
NVIC Constant and addresses

- core_cm3.h



```
/** \brief  Structure type to access the Nested Vectored Interrupt Controller (NVIC).
 */
typedef struct
{
  __IO uint32_t ISER[8];                 /*!< Offset: 0x000  (R/W)  Interrupt Set Enable Register           */
       uint32_t RESERVED0[24];
  __IO uint32_t ICER[8];                 /*!< Offset: 0x080  (R/W)  Interrupt Clear Enable Register         */
       uint32_t RSERVED1[24];
  __IO uint32_t ISPR[8];                 /*!< Offset: 0x100  (R/W)  Interrupt Set Pending Register          */
       uint32_t RESERVED2[24];
  __IO uint32_t ICPR[8];                 /*!< Offset: 0x180  (R/W)  Interrupt Clear Pending Register        */
       uint32_t RESERVED3[24];
  __IO uint32_t IABR[8];                 /*!< Offset: 0x200  (R/W)  Interrupt Active bit Register           */
       uint32_t RESERVED4[56];
  __IO uint8_t  IP[240];                 /*!< Offset: 0x300  (R/W)  Interrupt Priority Register (8Bit wide) */
       uint32_t RESERVED5[644];
  __O  uint32_t STIR;                    /*!< Offset: 0xE00  ( /W)  Software Trigger Interrupt Register     */
}  NVIC_Type;
```

Experiment priority and nested interruptions



```
---------------------------
of any kind.
litecnico di Torino. All rights reserved.
-----------------------------------------------------------*/
                    /* LPC17xx definitions                    */
utton.h"
s from funct_led */
_value;            /* defined in funct_led                   */
-----------------------------------------------------------
-----------------------------------------------------------*/
                    /* System Initialization (i.e., PLL)      */
                    /* LED Initialization                     */
                    /* BUTTON Initialization                  */
                    /* Loop forever                           */
```

Nested Vectored Interrupt Controller (NVIC)

Idx	Source	Name	E	P	A	Priority
32	PLL0 Lock	PLOCK0	0	1	0	0
33	RTC CIF		0	0	0	0
33	RTC ALF		0	0	0	0
34	External Interrupt 0	EINT0	1	0	0	3
35	External Interrupt 1	EINT1	1	0	1	2
36	External Interrupt 2	EINT2	1	0	1	1
37	External Interrupt 3	EINT3	0	0	0	0
37	GPIO Interrupts		0	0	0	0
38	A/D Converter	ADC	0	0	0	0
39	Brown Out Detect	BOD	0	0	0	0
40	USB		0	0	0	0
41	CAN		0	0	0	0

Selected Interrupt: ✓ Enable ☐ Pending Priority: -2

Interrupt Control & State
INT_CTRL_ST: 0x00400024 VECTACTIVE: 0x24
☐ RETTOBASE VECTPENDING: 0x00
☐ ISRPREEMPT ✓ ISRPENDING

Application Interrupt & Reset Control
AIRC: 0xFA050000 PRIGROUP: 0: 7.1
☐ VECTRESET ☐ SYSRESETREQ
☐ VECTCLRACTIVE ☐ ENDIANESS

Vector Table Offset
VTO: 0x00000000 TBLOFF: 0x000000
☐ TBLBASE

Software Interrupt Trigger
SW_TRIG_INT: 0x00000000 INTID: 0x00

External Interrupts

Name	Int	Mode	Polar
EINT0	0	1	0
EINT1	1	1	0
EINT2	1	1	0
EINT3	0	0	0

Selected External Interrupt
☐ EINT0 ✓ EXTMODE0
☐ EXTPOLAR0

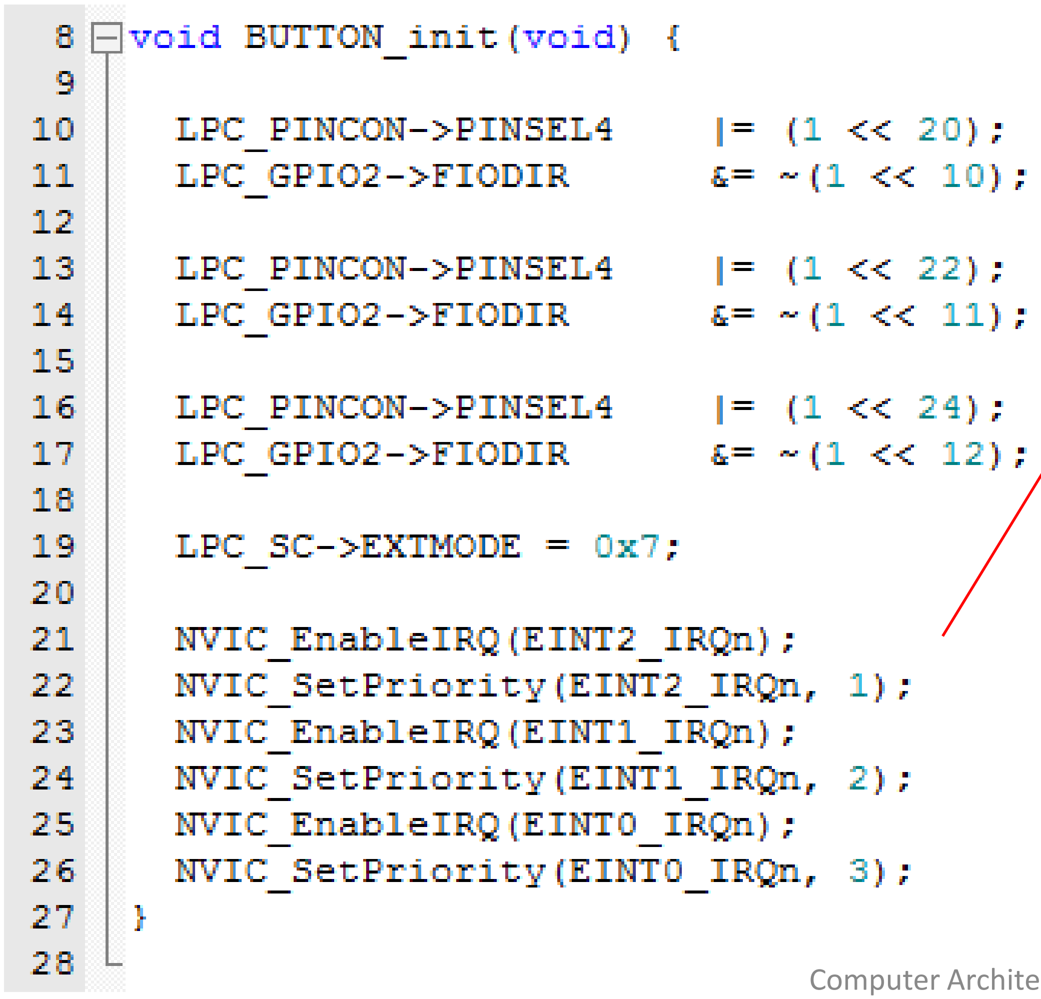
Registers
EXTINT: 0x06 EXTMODE: 0x07
EXTPOLAR: 0x00

General Purpose Input/Output 2 (GPIO 2) - Fast Interface

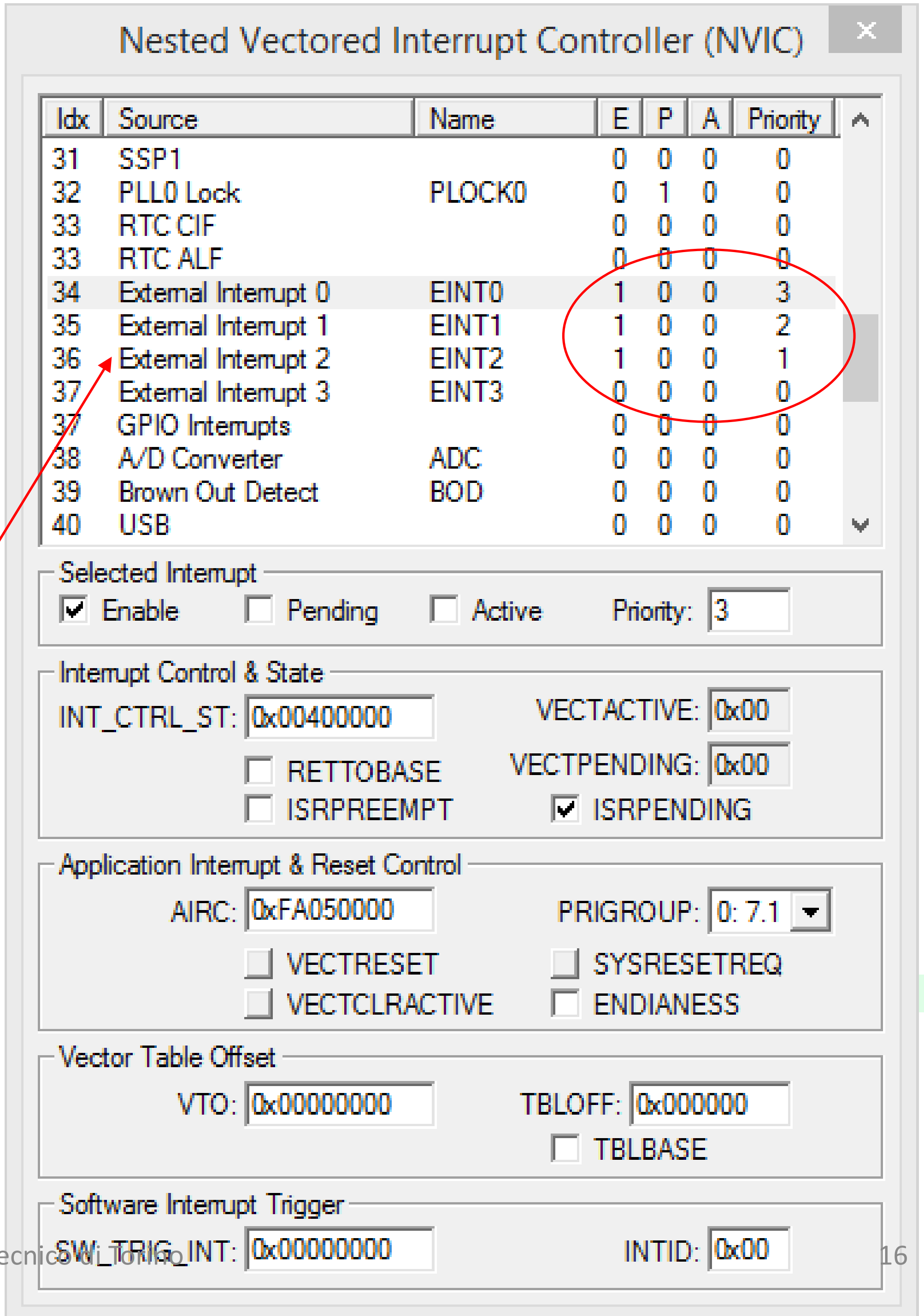
GPIO2
FIO2DIR: 0x000000FF
FIO2MASK: 0x00000000
FIO2SET: 0x00000000
FIO2CLR: 0x00000000
FIO2PIN: 0x00002300
Pins: 0x00002F00

t1: 0.05345665 sec L:41 C:1 CAP NUM SCRL OVR R/W

BOOT



```c
void BUTTON_init(void) {

    LPC_PINCON->PINSEL4    |= (1 << 20);
    LPC_GPIO2->FIODIR      &= ~(1 << 10);

    LPC_PINCON->PINSEL4    |= (1 << 22);
    LPC_GPIO2->FIODIR      &= ~(1 << 11);

    LPC_PINCON->PINSEL4    |= (1 << 24);
    LPC_GPIO2->FIODIR      &= ~(1 << 12);

    LPC_SC->EXTMODE = 0x7;

    NVIC_EnableIRQ(EINT2_IRQn);
    NVIC_SetPriority(EINT2_IRQn, 1);
    NVIC_EnableIRQ(EINT1_IRQn);
    NVIC_SetPriority(EINT1_IRQn, 2);
    NVIC_EnableIRQ(EINT0_IRQn);
    NVIC_SetPriority(EINT0_IRQn, 3);
}
```



Nested Vectored Interrupt Controller (NVIC)

Idx	Source	Name	E	P	A	Priority
31	SSP1		0	0	0	0
32	PLL0 Lock	PLOCK0	0	1	0	0
33	RTC CIF		0	0	0	0
33	RTC ALF		0	0	0	0
34	External Interrupt 0	EINT0	1	0	0	3
35	External Interrupt 1	EINT1	1	0	0	2
36	External Interrupt 2	EINT2	1	0	0	1
37	External Interrupt 3	EINT3	0	0	0	0
37	GPIO Interrupts		0	0	0	0
38	A/D Converter	ADC	0	0	0	0
39	Brown Out Detect	BOD	0	0	0	0
40	USB		0	0	0	0

Selected Interrupt
- [✓] Enable
- [ ] Pending
- [ ] Active
- Priority: 3

Interrupt Control & State
- INT_CTRL_ST: 0x00400000
- VECTACTIVE: 0x00
- VECTPENDING: 0x00
- [ ] RETTOBASE
- [ ] ISRPREEMPT
- [✓] ISRPENDING

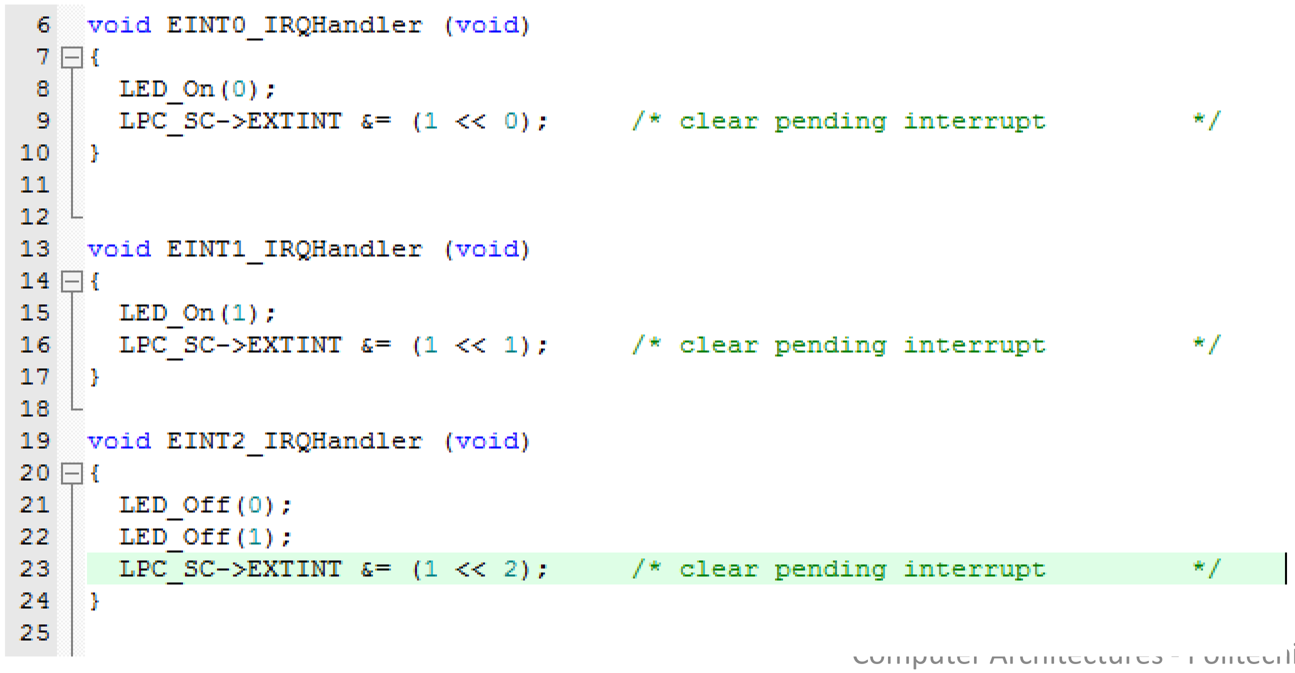
Application Interrupt & Reset Control
- AIRC: 0xFA050000
- PRIGROUP: 0: 7.1
- [ ] VECTRESET
- [ ] SYSRESETREQ
- [ ] VECTCLRACTIVE
- [ ] ENDIANESS

Vector Table Offset
- VTO: 0x00000000
- TBLOFF: 0x000000
- [ ] TBLBASE

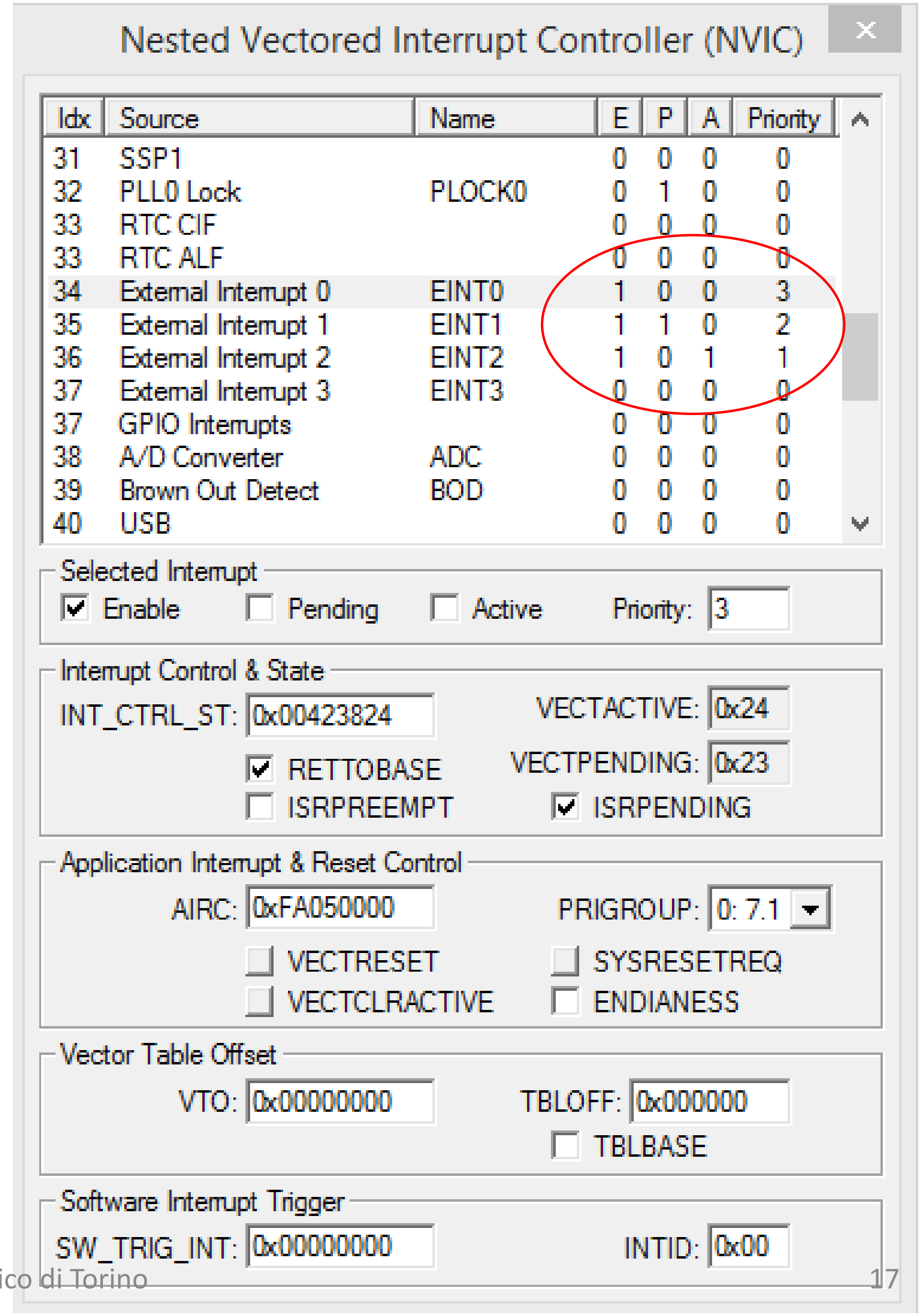
Software Interrupt Trigger
- SW_TRIG_INT: 0x00000000
- INTID: 0x00

RUNTIME (1)

- CASE 1)
  1. The EINT2 interrupt is taken and being served
  2. The EINT1 interrupt (with lower priority) is taken
  3. EINT1 is pending and will be served only when EINT2 is fully handled



```
void EINT0_IRQHandler (void)
{
  LED_On(0);
  LPC_SC->EXTINT &= (1 << 0);     /* clear pending interrupt         */
}


void EINT1_IRQHandler (void)
{
  LED_On(1);
  LPC_SC->EXTINT &= (1 << 1);     /* clear pending interrupt         */
}

void EINT2_IRQHandler (void)
{
  LED_Off(0);
  LED_Off(1);
  LPC_SC->EXTINT &= (1 << 2);     /* clear pending interrupt         */
}

```



Nested Vectored Interrupt Controller (NVIC)

Idx	Source	Name	E	P	A	Priority
31	SSP1		0	0	0	0
32	PLL0 Lock	PLOCK0	0	1	0	0
33	RTC CIF		0	0	0	0
33	RTC ALF		0	0	0	0
34	External Interrupt 0	EINT0	1	0	0	3
35	External Interrupt 1	EINT1	1	1	0	2
36	External Interrupt 2	EINT2	1	0	1	1
37	External Interrupt 3	EINT3	0	0	0	0
37	GPIO Interrupts		0	0	0	0
38	A/D Converter	ADC	0	0	0	0
39	Brown Out Detect	BOD	0	0	0	0
40	USB		0	0	0	0

Selected Interrupt
[✓] Enable [ ] Pending [ ] Active Priority: 3

Interrupt Control & State
INT_CTRL_ST: 0x00423824 VECTACTIVE: 0x24
[✓] RETTOBASE VECTPENDING: 0x23
[ ] ISRPREEMPT [✓] ISRPENDING

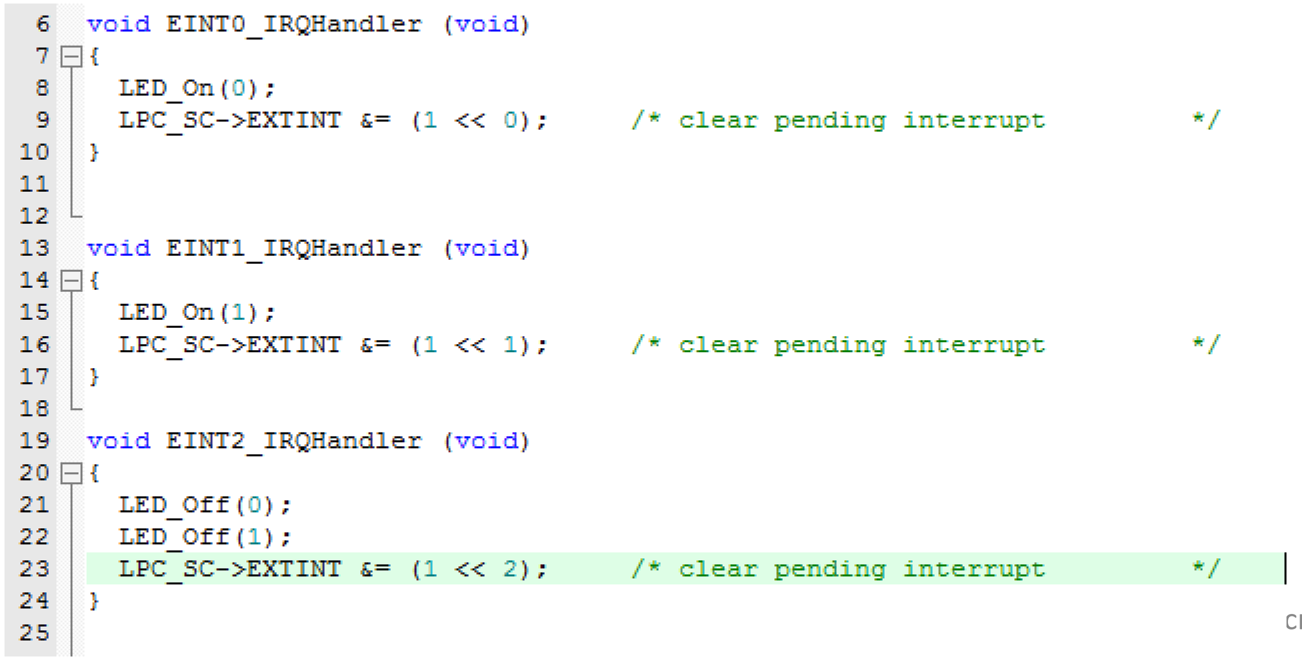
Application Interrupt & Reset Control
AIRC: 0xFA050000 PRIGROUP: 0: 7.1
[ ] VECTRESET [ ] SYSRESETREQ
[ ] VECTCLRACTIVE [ ] ENDIANESS

Vector Table Offset
VTO: 0x00000000 TBLOFF: 0x000000
[ ] TBLBASE

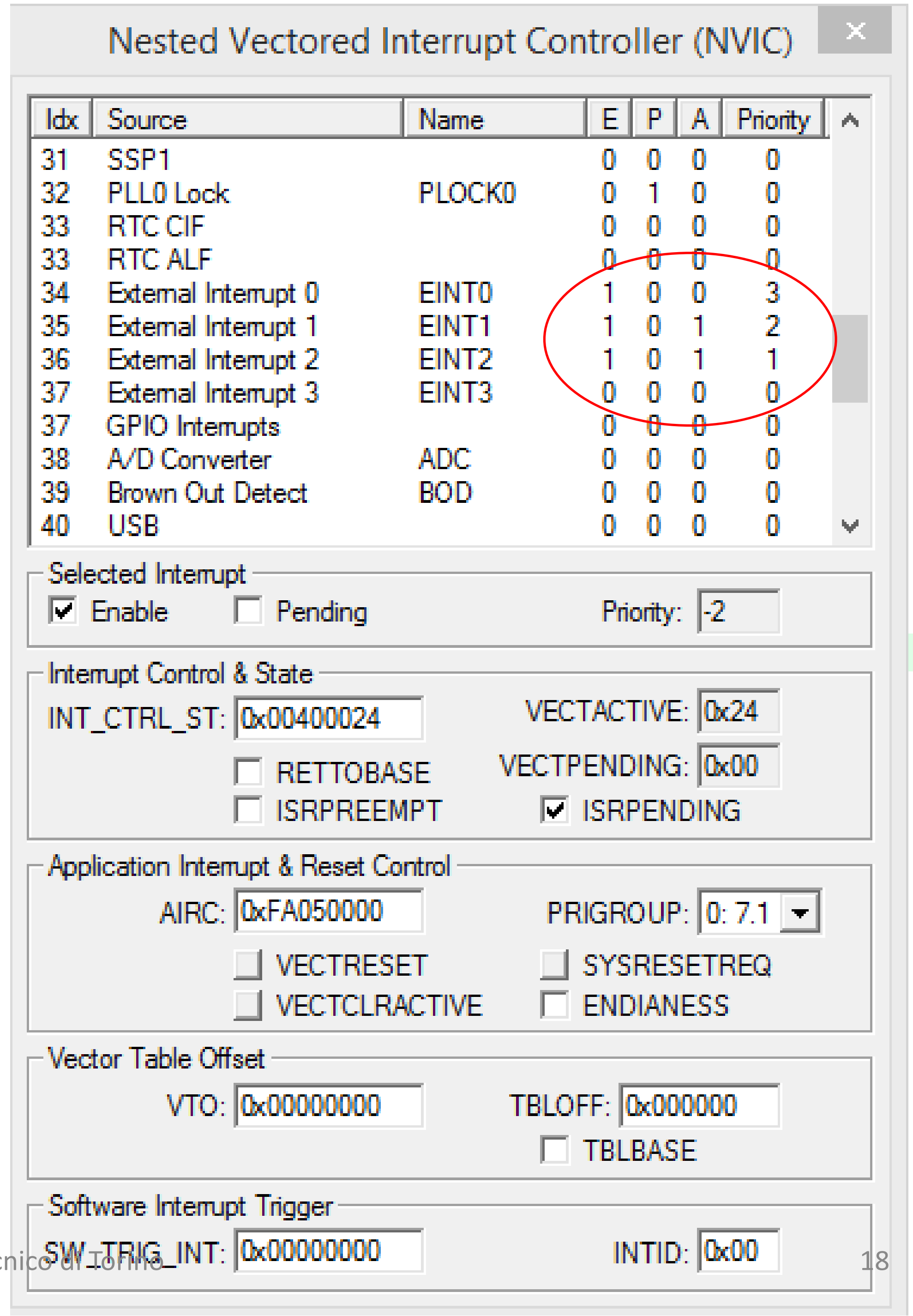
Software Interrupt Trigger
SW_TRIG_INT: 0x00000000 INTID: 0x00

RUNTIME (2)

- CASE 2)
  1. The EINT1 interrupt is taken and being served
  2. The EINT2 interrupt (with higher priority) is taken
  3. EINT1 is suspended and completed only when EINT2 is handled



```
void EINT0_IRQHandler (void)
{
  LED_On(0);
  LPC_SC->EXTINT &= (1 << 0);     /* clear pending interrupt         */
}


void EINT1_IRQHandler (void)
{
  LED_On(1);
  LPC_SC->EXTINT &= (1 << 1);     /* clear pending interrupt         */
}

void EINT2_IRQHandler (void)
{
  LED_Off(0);
  LED_Off(1);
  LPC_SC->EXTINT &= (1 << 2);     /* clear pending interrupt         */
}

```



Nested Vectored Interrupt Controller (NVIC)

Idx	Source	Name	E	P	A	Priority
31	SSP1		0	0	0	0
32	PLL0 Lock	PLOCK0	0	1	0	0
33	RTC CIF		0	0	0	0
33	RTC ALF		0	0	0	0
34	External Interrupt 0	EINT0	1	0	0	3
35	External Interrupt 1	EINT1	1	0	1	2
36	External Interrupt 2	EINT2	1	0	1	1
37	External Interrupt 3	EINT3	0	0	0	0
37	GPIO Interrupts		0	0	0	0
38	A/D Converter	ADC	0	0	0	0
39	Brown Out Detect	BOD	0	0	0	0
40	USB		0	0	0	0

Selected Interrupt
[✓] Enable [ ] Pending Priority: -2

Interrupt Control & State
INT_CTRL_ST: 0x00400024 VECTACTIVE: 0x24
[ ] RETTOBASE VECTPENDING: 0x00
[ ] ISRPREEMPT [✓] ISRPENDING

Application Interrupt & Reset Control
AIRC: 0xFA050000 PRIGROUP: 0: 7.1
[ ] VECTRESET [ ] SYSRESETREQ
[ ] VECTCLRACTIVE [ ] ENDIANESS

Vector Table Offset
VTO: 0x00000000 TBLOFF: 0x000000
[ ] TBLBASE

Software Interrupt Trigger
SW_TRIG_INT: 0x00000000 INTID: 0x00