Buttons and interrupt controller

R. Ferrero, P. Bernardi

Politecnico di Torino

Dipartimento di Automatica e Informatica (DAUIN)

Torino - Italy

Available resources

- User manual
- Sample project
- Schematic

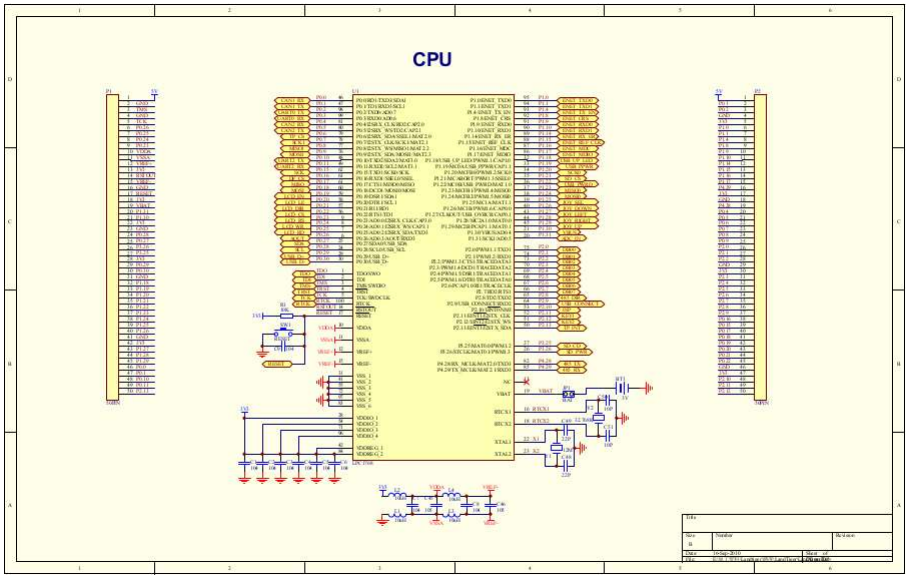
UM10360

LPC176x/5x User manual

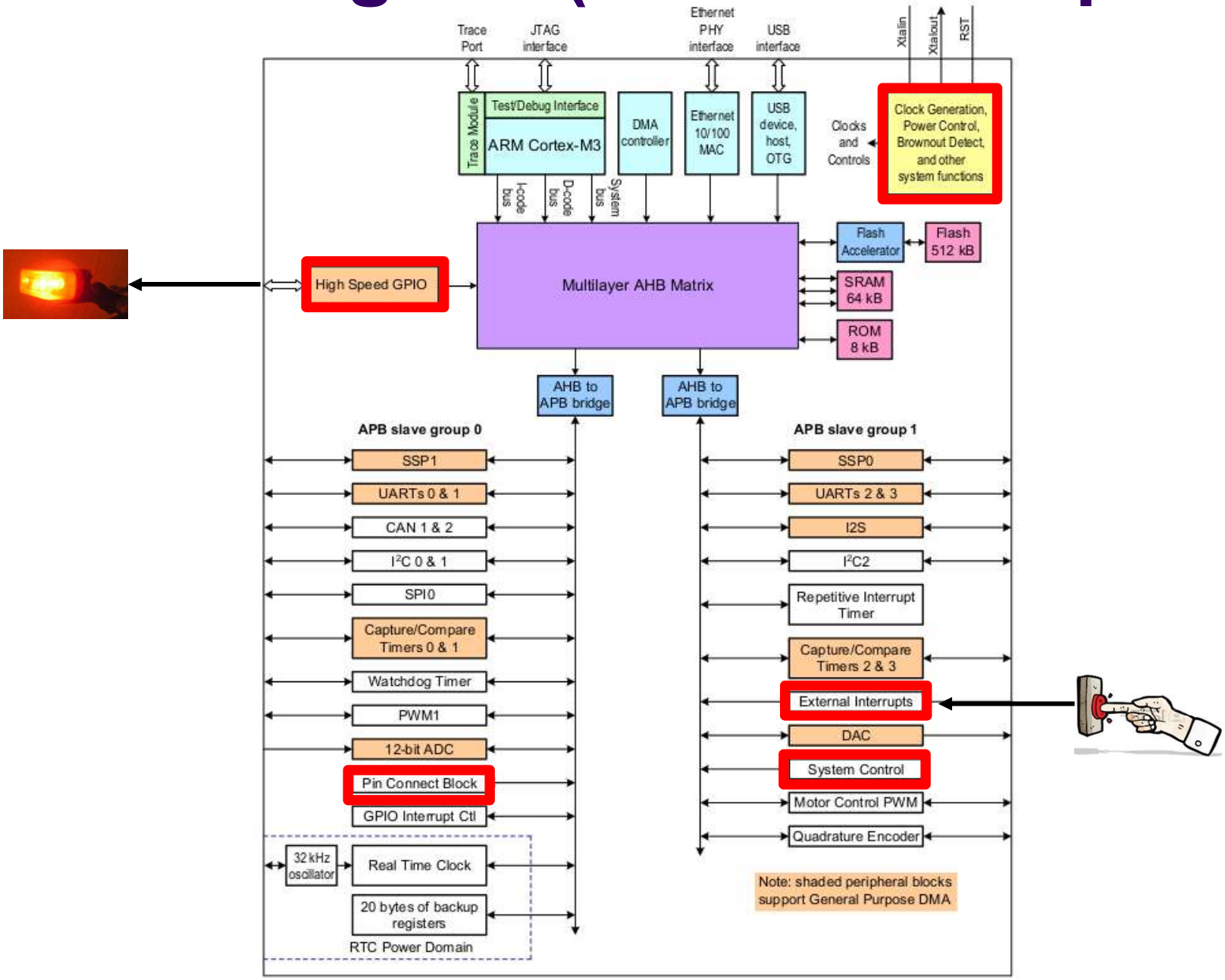
Rev. 4. 1 — 19 December 2016 User manual

Document information

Info	Content
Keywords	LPC1769, LPC1768, LPC1767, LPC1766, LPC1765, LPC1764, LPC1763, LPC1759, LPC1758, LPC1756, LPC1754, LPC1752, LPC1751, ARM, ARM Cortex-M3, 32-bit, USB, Ethernet, CAN, I2S, Microcontroller
Abstract	LPC176x/5x user manual



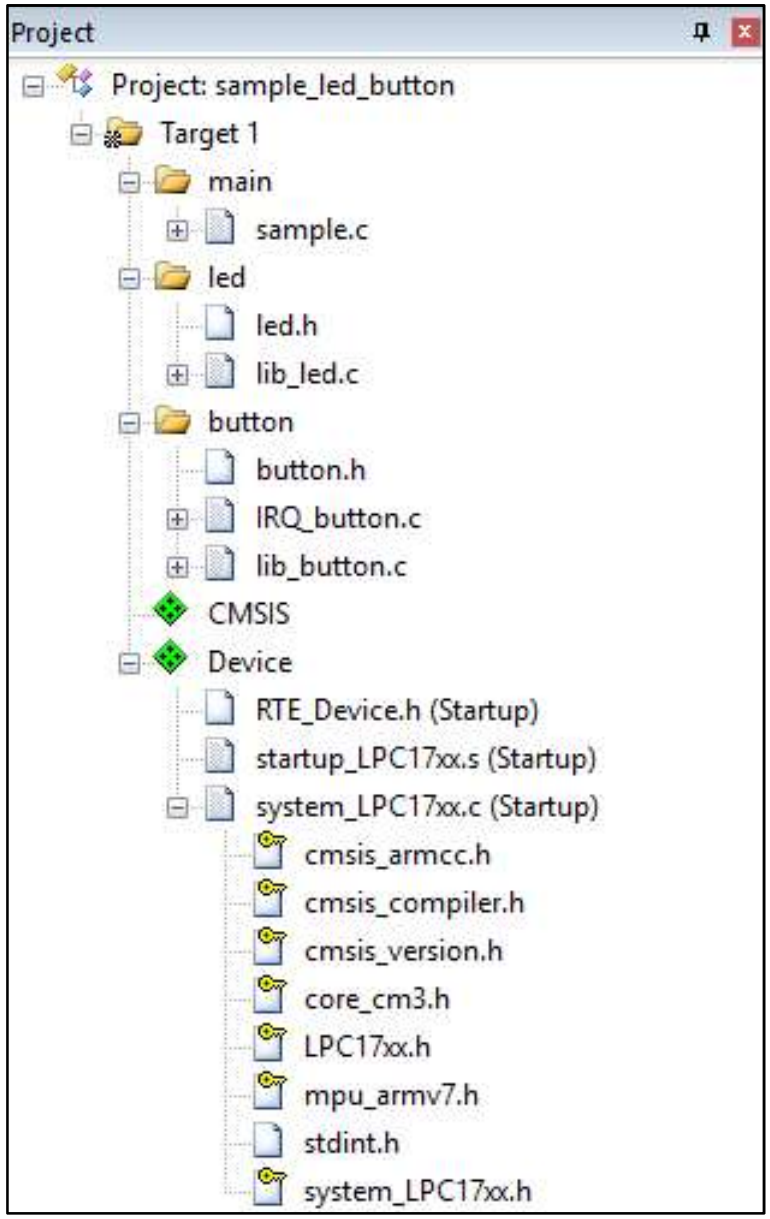
Block diagram (user manual p. 10)



Creation of sample project

- Add two new groups beside the default one
  - Right click on Target 1 -> Add Group
- Optional: rename groups as *main*, *led*, *button*
  - Right click on a group -> Manage Project items
- Copy sample.c, led and button folder in the project folder; add files to corresponding group
  - Right click on a group -> Add Existing Files
- Options for target  -> Debug
  - Dialog DLL: DARMP1.DLL (Simulator) or TARMP1.DLL (ULINK2/ME Cortex Debugger)
  - Parameter: -pLPC1768 (both)

Project view



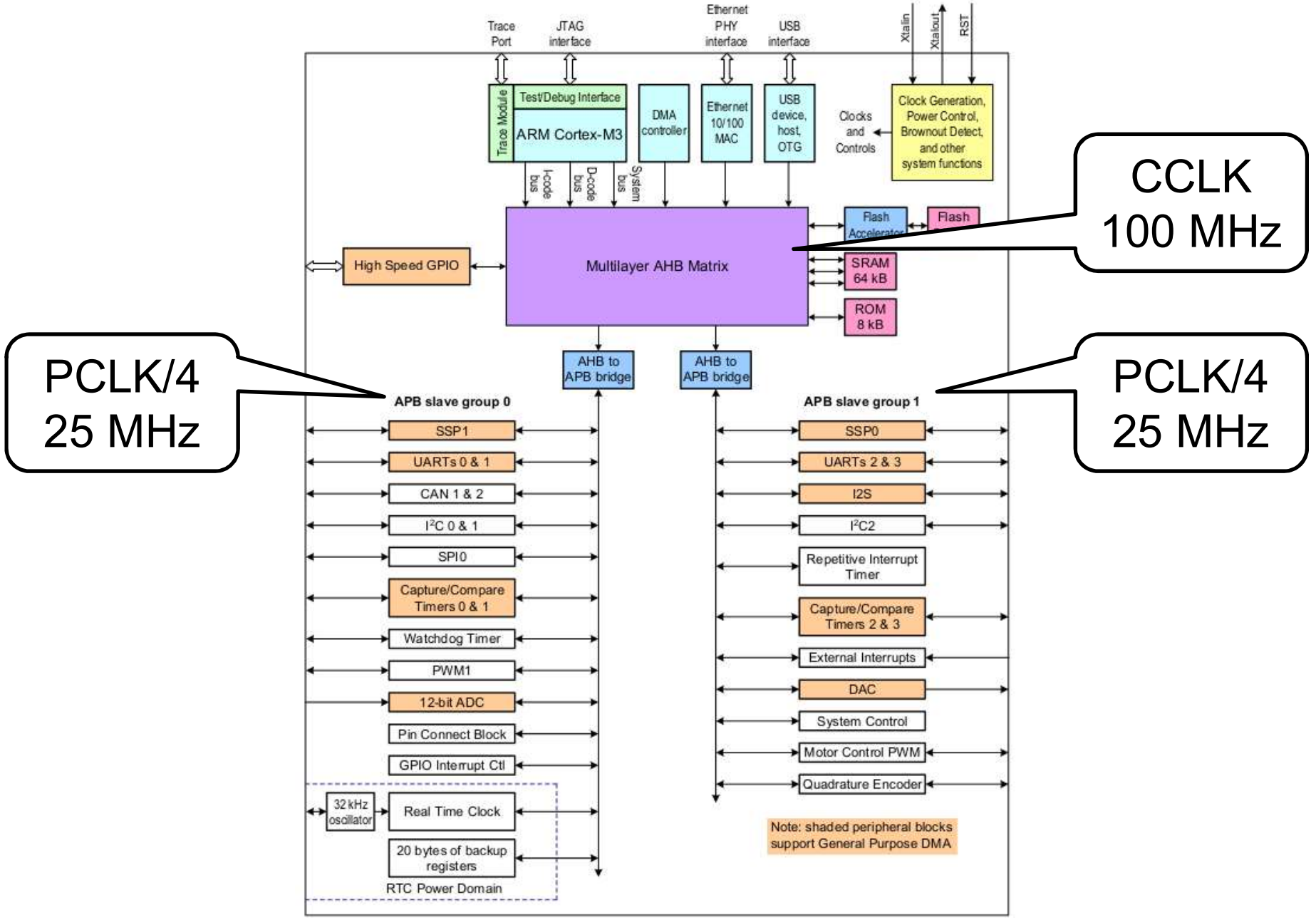
Files in sample project

- startup_LPC17xx.s
- system_LPC17xx.c
  - *SystemInit()* function called by reset_handler
- sample.c
- led group
- button group
  - lib_button.c: inizialitation of buttons and NVIC
  - IRQ_button.c: handlers for external interrupts
- system libraries
  - lpc17xx.h
  - core_cm3.h

Files in sample project

- startup_LPC17xx.s
- **system_LPC17xx.c**
  - ***SystemInit()* function called by reset_handler**
- sample.c
- led group
- button group
  - lib_button.c: inizialitation of buttons and NVIC
  - IRQ_button.c: handlers for external interrupts
- system libraries
  - lpc17xx.h
  - core_cm3.h

Clock configuration in the project



Clock selection (page 31)

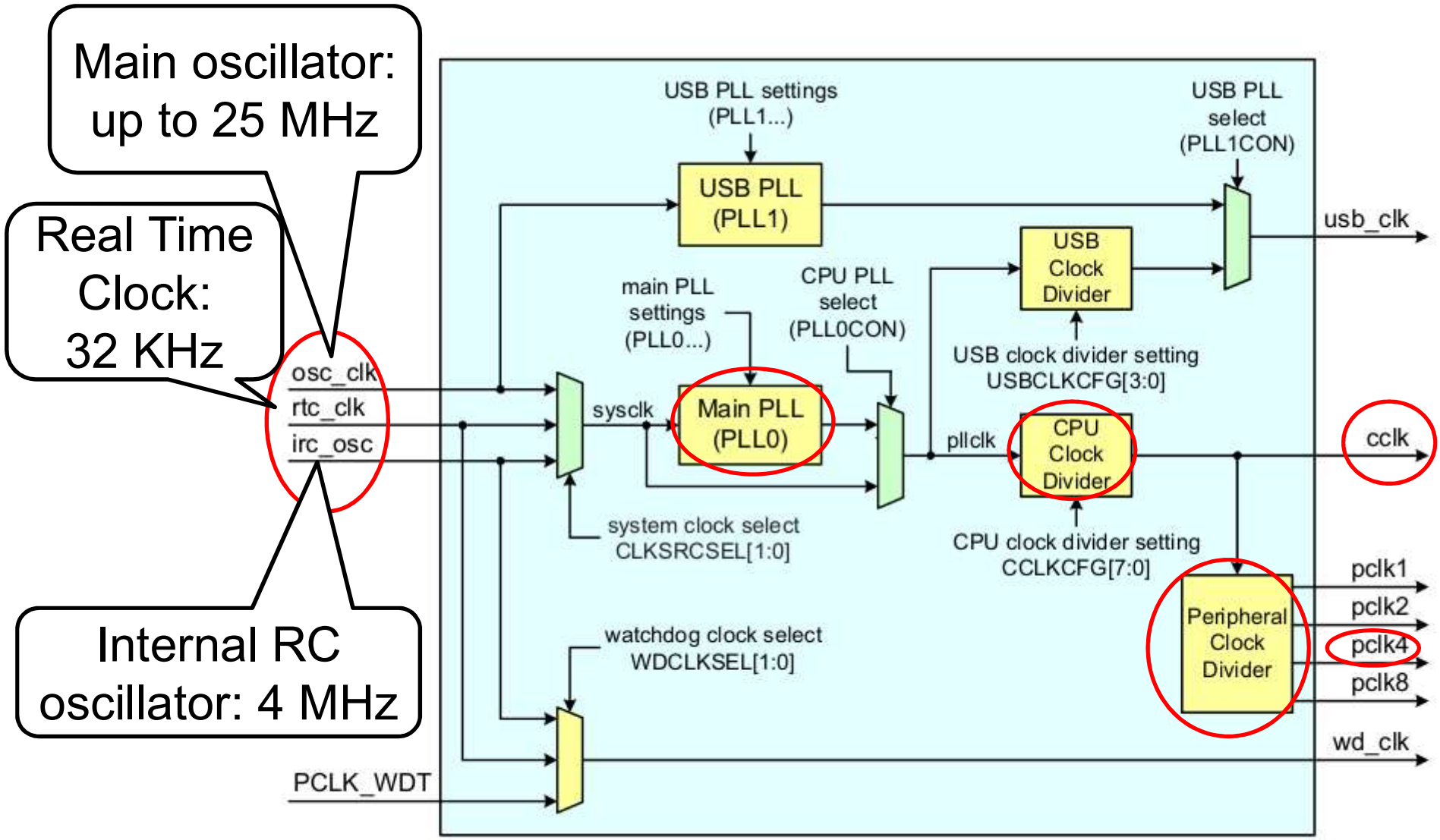
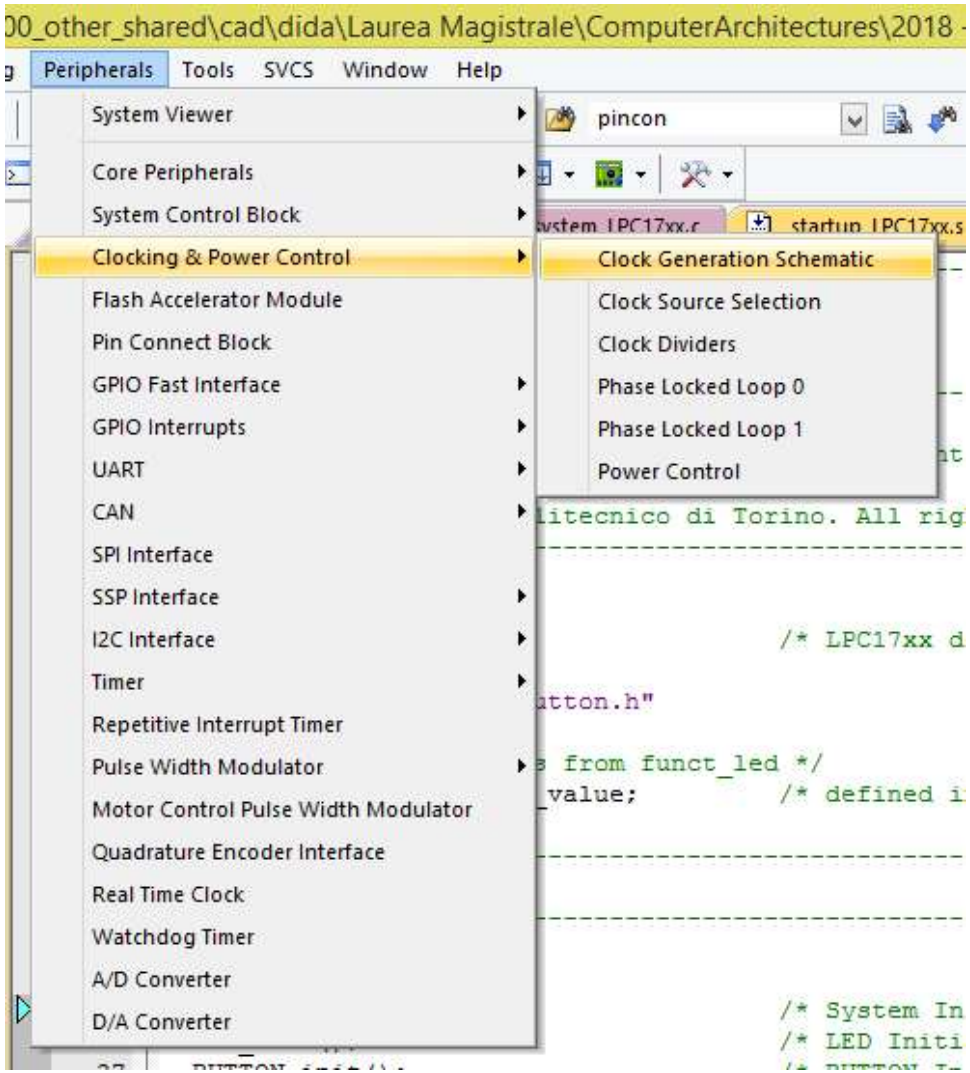
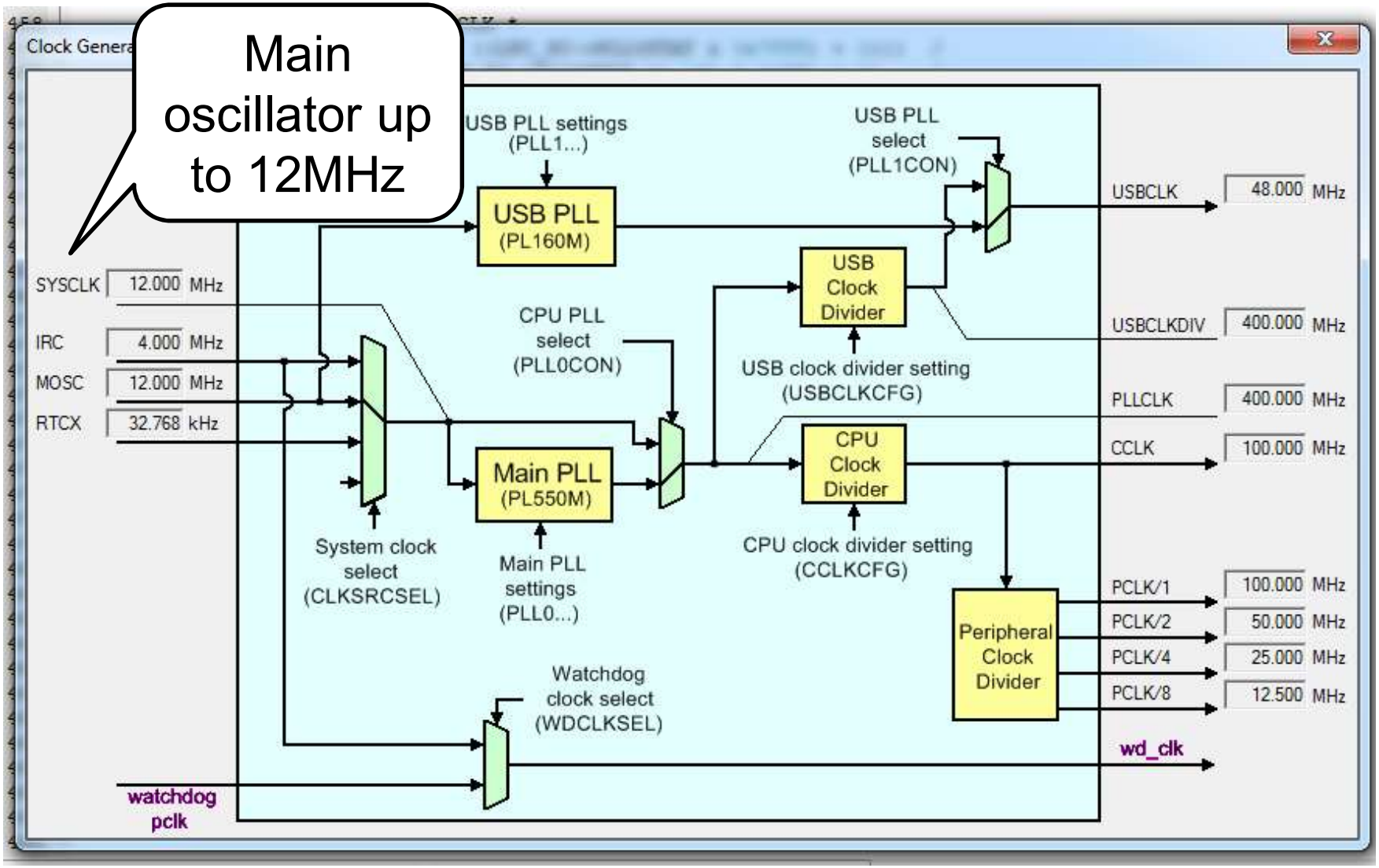


Fig 7. Clock generation for the LPC176x/5x

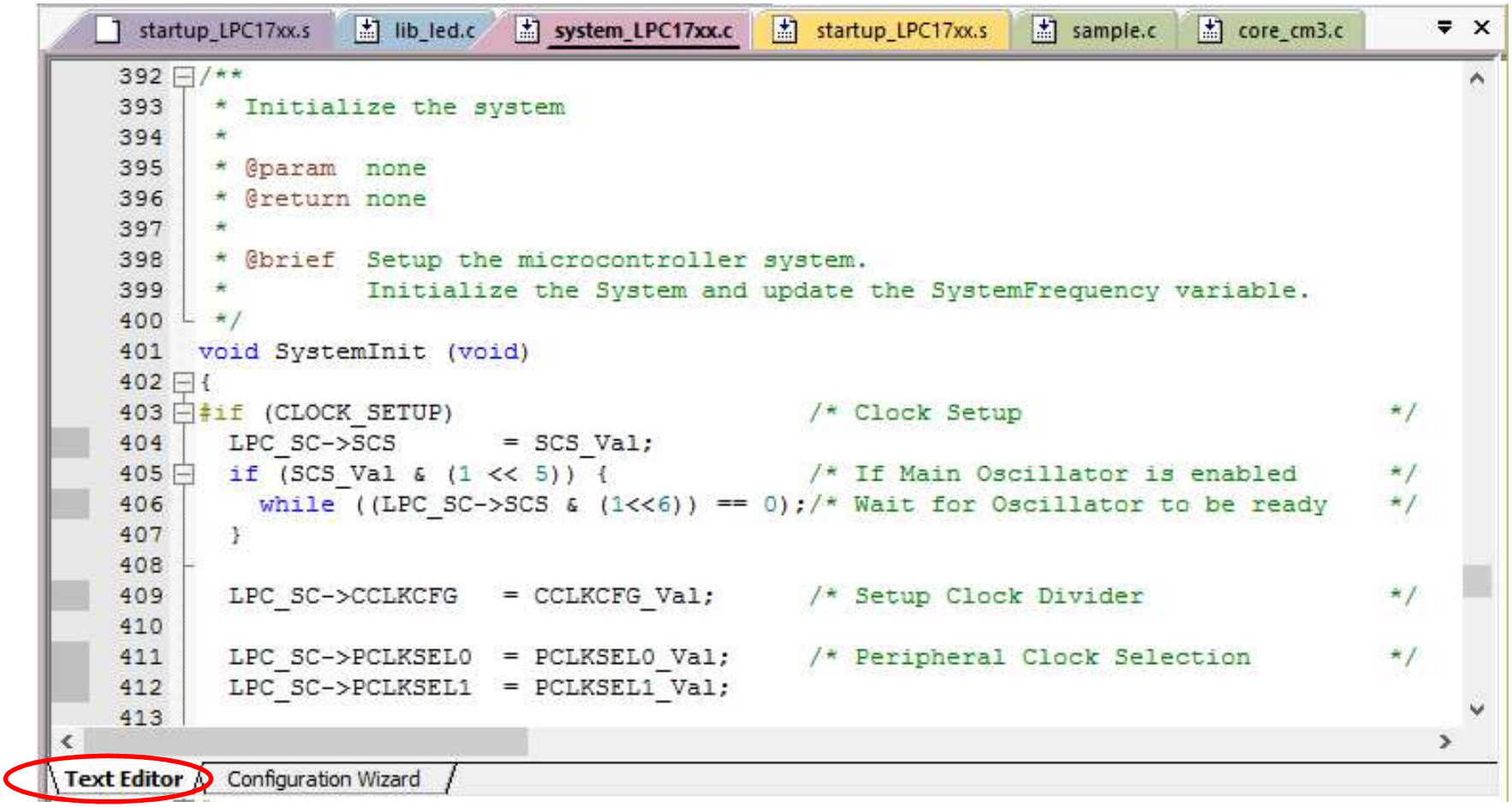
Debug view


00_other_shared\cad\dida\Laurea Magistrale\ComputerArchitectures\2018
Peripherals
Tools
SVCS
Window
Help
System Viewer
Core Peripherals
System Control Block
Clocking & Power Control
Flash Accelerator Module
Pin Connect Block
GPIO Fast Interface
GPIO Interrupts
UART
CAN
SPI Interface
SSP Interface
I2C Interface
Timer
Repetitive Interrupt Timer
Pulse Width Modulator
Motor Control Pulse Width Modulator
Quadrature Encoder Interface
Real Time Clock
Watchdog Timer
A/D Converter
D/A Converter
Clock Generation Schematic
Clock Source Selection
Clock Dividers
Phase Locked Loop 0
Phase Locked Loop 1
Power Control
pincon
litecnico di Torino. All rig
/* LPC17xx d
tton.h"
from funct_led */
value;
/* defined i
/* System In
/* LED Initi

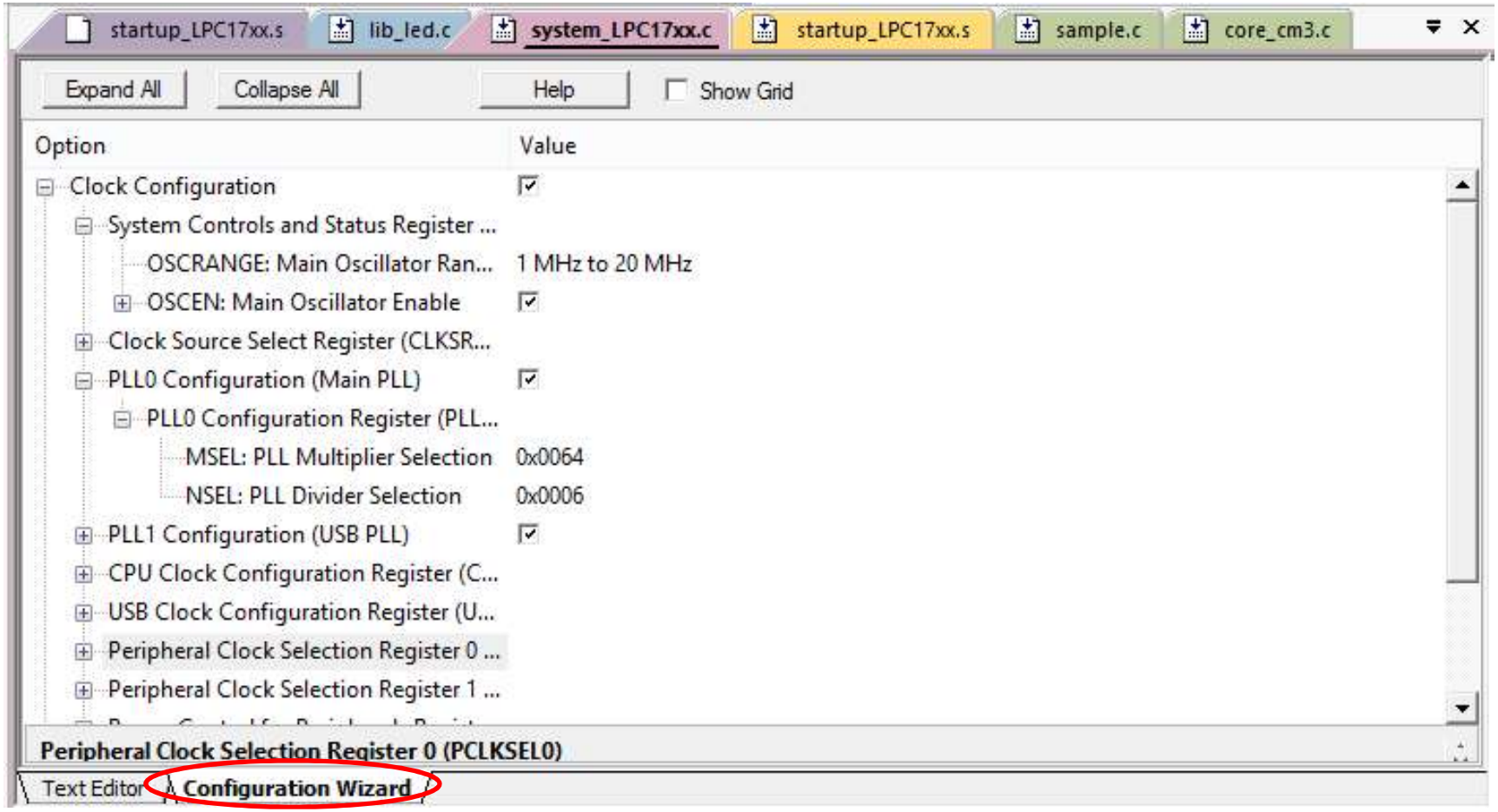
Clock values after *SystemInit()*



system_LPC17xx.c: text view



system_LPC17xx.c: wizard



Configuration Wizard Annotations

- Configuration Wizard Annotations consist of annotation items and annotation modifiers.
- They create GUI-like elements in IDEs for configuration files.
- The GUI-like approach makes it easier for the user to check and adapt configuration files to the application needs.

Configuration Wizard Annotations

- The Configuration Wizard section must begin within the first 100 lines of code and must start with:

  // <<< Use Configuration Wizard in Context Menu >>>

- The optional end of the Configuration Wizard section is :

  // <<< end of configuration section >>>

- Annotations are written as comments in the code: it must start with a double backslash (//).
- By default, it is the next code symbol that follows the annotation to be modified.
- It is possible to add a “skip-value” to omits a number of code symbols. This overwrites the previous rule.
- A descriptive text can be added to items.

Configuration Wizard Annotations

- <h> : *Heading.* Creates a header section. All items and options enclosed by <h> and </h> belong to one group and can be expanded. This entry makes no changes to code symbols. It is just used to group other items and modifiers.
- <e>* : *Heading with Enable.* Creates a header section with a checkbox to enabled or disabled all items and options enclosed by <e> and </e>.
- <e.*i*>* : *Heading with Enable*: modifies a specific bit (*i*) (example: <e.4> - changes bit 4 of a value).
- <i> : Tooltip help

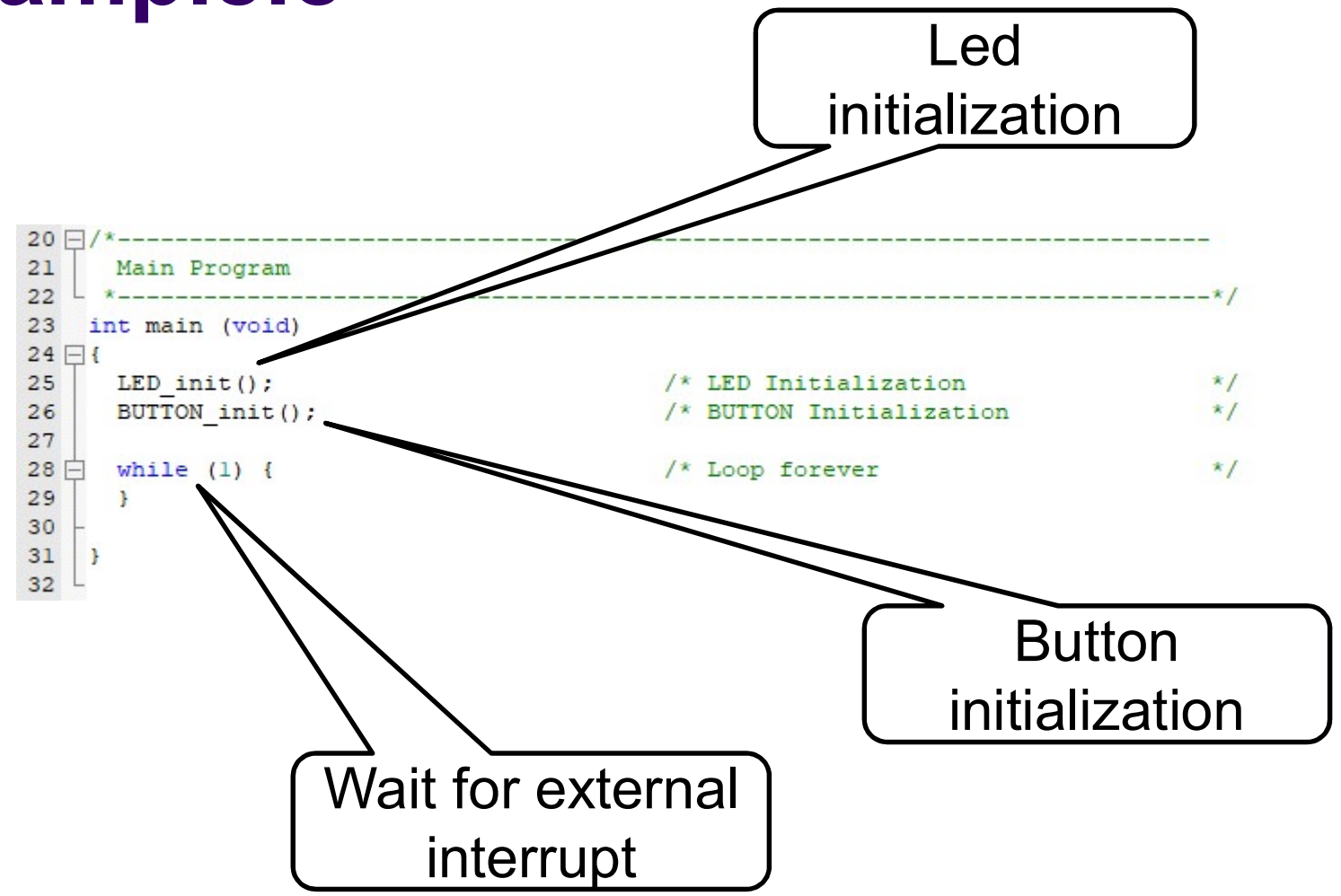
Configuration Wizard Annotations

- <o>* : Option with selection or number entry.
  - // <o>Round-Robin Timeout [ticks] <1-1000>
  - The example creates an option with the text *Round-Robin Timeout [ticks]* and a field to enter values that can range between [1..1000].
- <o.*x*..*y*>* : Option Modify a range of bits. (example: <o.4..5> - bit 4 to 5).
  - // <o.0..15>Language ID <0x0000-0xFCFF>
- <o*i*> *:* Skip *i* items. Can be applied to all annotation items marked with a *

Files in sample project

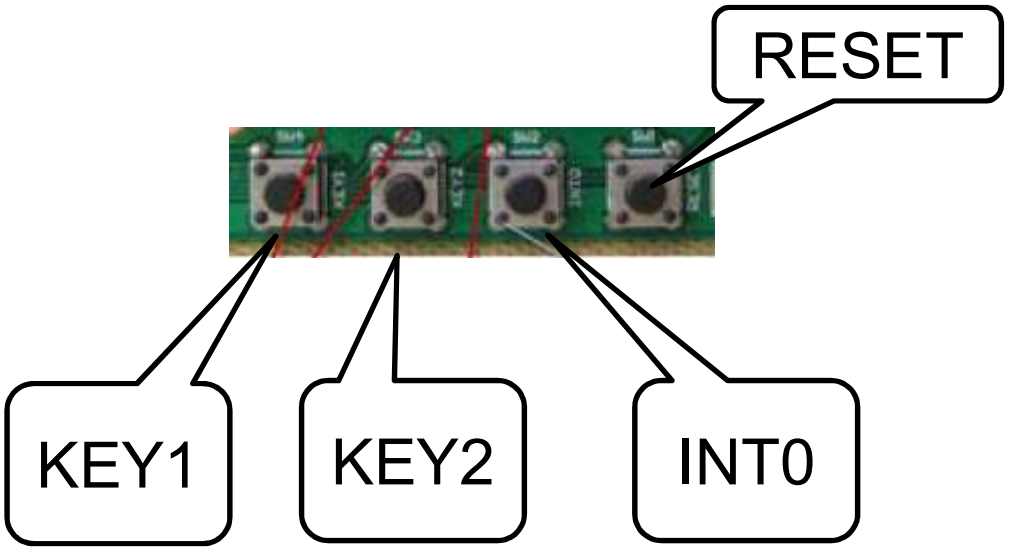
- startup_LPC17xx.s
- system_LPC17xx.c
  - *SystemInit()* function called by reset_handler
- **sample.c**
- led group
- button group
  - lib_button.c: inizialitation of buttons and NVIC
  - IRQ_button.c: handlers for external interrupts
- system libraries
  - lpc17xx.h
  - core_cm3.h

sample.c


Led initialization
20 /*----------
21 Main Program
22 *----------*/
23 int main (void)
24 {
25 LED_init(); /* LED Initialization */
26 BUTTON_init(); /* BUTTON Initialization */
27
28 while (1) { /* Loop forever */
29 }
30
31 }
32
Button initialization
Wait for external interrupt

Sample project: functionality

- Pression of button INT0 -> switch on LD11
- Pression of button KEY1 -> switch on LD10
- Pression of button KEY2 -> switch off LD10 and LD11

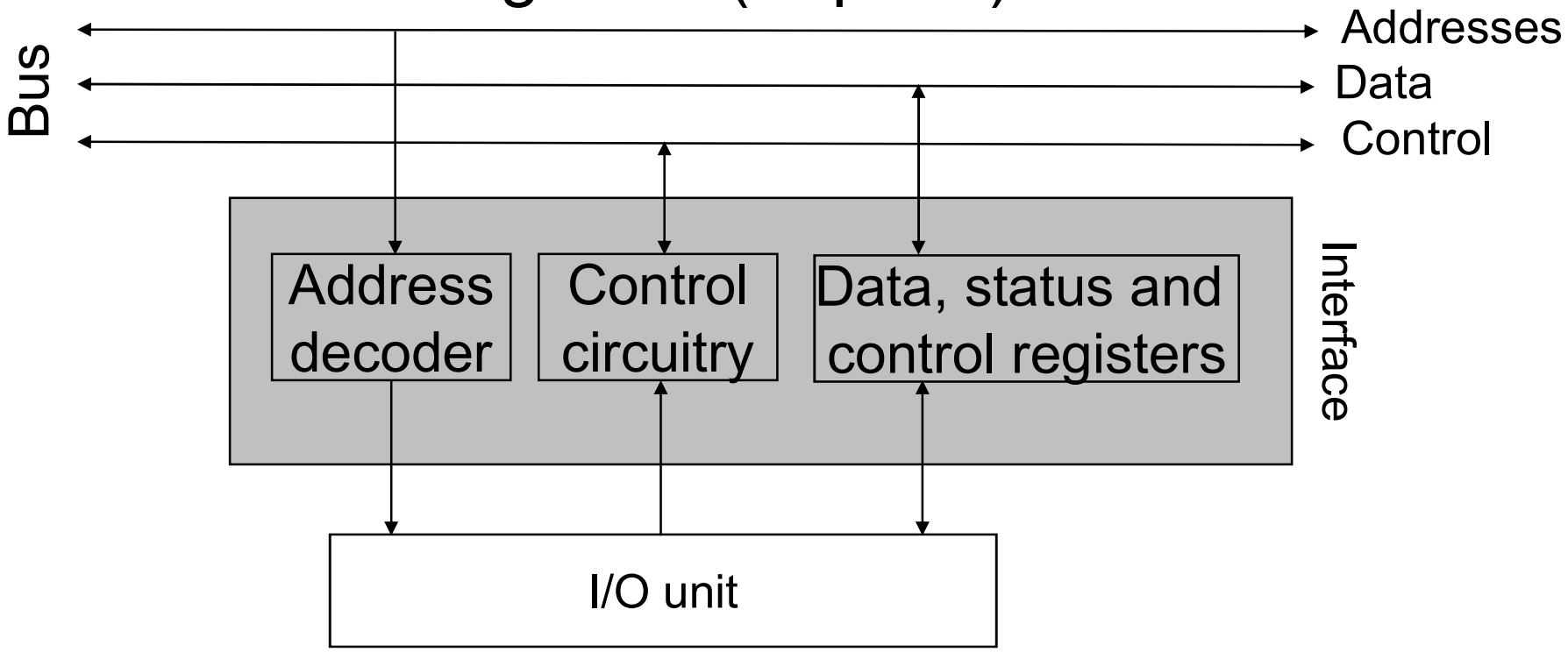


Input/Output system management

- One of the main functions of computer systems is to interact and exchange information with the outside world.
- This function is performed through the Input/Output (*I/O) subsystem*, consisting of:
  - Input/Output devices (*peripherals*)
  - Units for controlling these devices (*interfaces*)
  - Software for their management.

I/O interface

- Communication between the processor (or memory) and the device occurs through the interface registers (or ports).



Registers addressing

- Each register is associated with an address and it is accessible via the system bus.
- Memory-mapped I/O
  - address space is shared with memory
  - same instructions and addressing methods used to access the memory.
- Isolated I/O (or I/O-mapped I/O)
  - separated addressing spaces (activated by signals)
  - appropriate instructions to access I/O ports.

访问I/O端口的适当指令

Synchronization

CPU must synchronize with I/O devices since their operating speeds are different. Two ways:

1. Polling or Programmed I/O
   - the CPU repetitively checks if the external device is ready for I/O operations by testing status registers (best practice) or data registers.
2. Interrupt
   - the device sends an asynchronous signal to CPU
   - the CPU stops running the current program
   - it executes an *Interrupt Service Routine* (ISR)
   - current program continues at the end of ISR.

Polling Vs. interrupt

	Polling	Interrupt
Implemen-tation	easy: software cycle	hardware requirements
Latency	high,depending on check frequency	low: asynchronous signal sent by the device
Nested requests	difficult to be managed	managed with priorities
Efficiency	poor: CPU wastes time in software cycle, with trivial tests	the system can enter in idle mode, and wakes up with interrupt request

Peripheral identification 外围设备识别

Different peripherals can require an interrupt, with different service procedures to be called.

1. Multiple interrupt lines 多中断线法
   - each device is connected to a different signal.

每个设备连接的信号不同

2. Polling 查询
   - when the CPU receives the request, it scans the state of all peripherals to identify the requester.

当CPU收到一个请求的时候，扫描所有的外部设备状态来识别请求者

3. Vectorized interrupt
   - requests are managed by the *interrupt controller*.

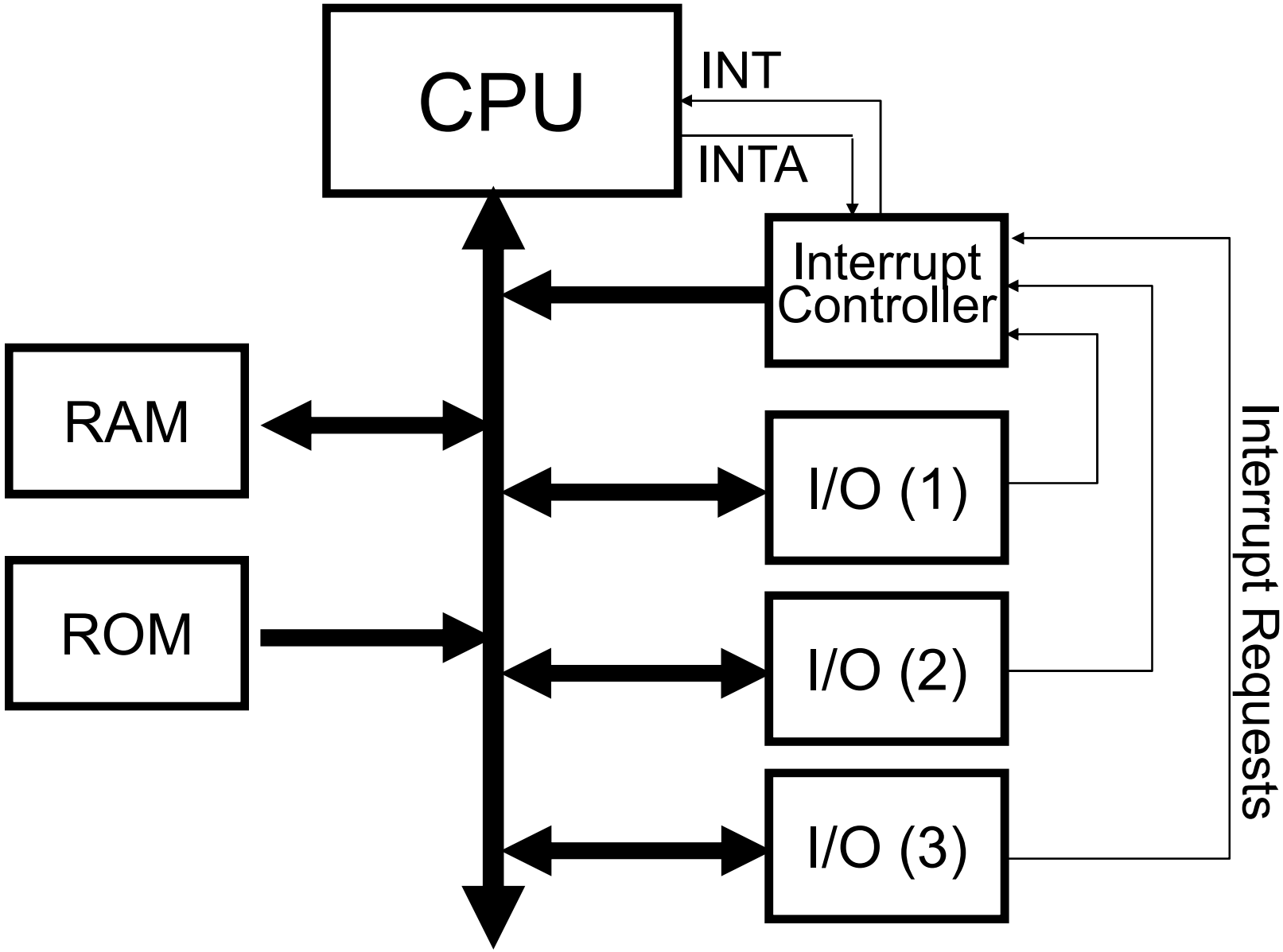
中断控制器管理着请求

Nested Vectored Interrupt Controller

内嵌向量中断控制器

- NVIC receives all requests and it drives one signal towards the CPU. NVIC接收所有请求，并把信号发送给CPU
- When the CPU is ready to serve a request:
  - it sends an Acknowledge Interrupt signal
  - the Interrupt Controller puts the identification code of the peripheral on the data bus 中断控制器把外围设备识别码放到数据总线上
  - the CPU uses the identification code as an index to access the Interrupt Vector Table (IVT)
  - the CPU reads the address of service procedure.

Interrupt controller



Interrupt Vector Table

- For each type of interrupt, it contains the address of its Interrupt Service Routine.
- There are 35 external interrupts in the Cortex-M3.

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^2C0 Interrupt Pending set. See functional description for bit 0.
11	ISP_I2C1	I^2C1 Interrupt Pending set. See functional description for bit 0.
12	ISP_I2C2	I^2C2 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^2S 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.

System setup for interrupts

引导时间：配置中断控制器

- Boot time: configure interrupt controller
  - Enable interrupt sources
  - Set priority of every source (optional)
  - Specify a flag variable as semaphore (optional)

将标志变量设为信号量

- Run time: execute interrupt service routine

运行时间：执行中断服务例程

  - Acknowledge: clear the flag of active interrupt
  - Preserve `r4-r8`, `r10-r11` (ABI AAPCS)
  - Communicate via shared global variables.

通过共享全局变量来沟通

Files in sample projects

- startup_LPC17xx.s
- system_LPC17xx.c
  - *SystemInit()* function called by reset_handler
- sample.c
- led group
- button group
  - lib_button.c: inizialitation of buttons and NVIC
  - IRQ_button.c: handlers for external interrupts
- **system libraries**
  - **lpc17xx.h**
  - core_cm3.h

lpc17xx.h: addressing memory

1) Constants are defined at C language level for addressing main blocks of memory.

```
915 /*****************************************************
916 /*                          Peripheral memory map
917 /*****************************************************
918 /* Base addresses
919 #define LPC_FLASH_BASE        (0x00000000UL)
920 #define LPC_RAM_BASE          (0x10000000UL)
921 #ifdef __LPC17XX_REV00
922 #define LPC_AHBRAM0_BASE      (0x20000000UL)
923 #define LPC_AHBRAM1_BASE      (0x20004000UL)
924 #else
925 #define LPC_AHBRAM0_BASE      (0x2007C000UL)
926 #define LPC_AHBRAM1_BASE      (0x20080000UL)
927 #endif
928 #define LPC_GPIO_BASE         (0x2009C000UL)
929 #define LPC_APB0_BASE         (0x40000000UL)
930 #define LPC_APB1_BASE         (0x40080000UL)
931 #define LPC_AHB_BASE          (0x50000000UL)
932 #define LPC_CM3_BASE          (0xE0000000UL)
```

Memory map (page 14)

Table 3. LPC176x/5x memory usage and details

Address range	General Use	Address range details and description	
0x0000 0000 to 0x1FFF FFFF	On-chip non-volatile memory	0x0000 0000 - 0x0007 FFFF	For devices with 512 kB of flash memory.
		0x0000 0000 - 0x0003 FFFF	For devices with 256 kB of flash memory.
		0x0000 0000 - 0x0001 FFFF	For devices with 128 kB of flash memory.
		0x0000 0000 - 0x0000 FFFF	For devices with 64 kB of flash memory.
		0x0000 0000 - 0x0000 7FFF	For devices with 32 kB of flash memory.
	On-chip SRAM	0x1000 0000 - 0x1000 7FFF	For devices with 32 kB of local SRAM.
		0x1000 0000 - 0x1000 3FFF	For devices with 16 kB of local SRAM.
		0x1000 0000 - 0x1000 1FFF	For devices with 8 kB of local SRAM.
	Boot ROM	0x1FFF 0000 - 0x1FFF 1FFF	8 kB Boot ROM with flash services.
0x2000 0000 to 0x3FFF FFFF	On-chip SRAM (typically used for peripheral data)	0x2007 C000 - 0x2007 FFFF	AHB SRAM - bank 0 (16 kB), present on devices with 32 kB or 64 kB of total SRAM.
		0x2008 0000 - 0x2008 3FFF	AHB SRAM - bank 1 (16 kB), present on devices with 64 kB of total SRAM.
	GPIO	0x2009 C000 - 0x2009 FFFF	GPIO.
0x4000 0000 to 0x5FFF FFFF	APB Peripherals	0x4000 0000 - 0x4007 FFFF	APB0 Peripherals, up to 32 peripheral blocks, 16 kB each.
		0x4008 0000 - 0x400F FFFF	APB1 Peripherals, up to 32 peripheral blocks, 16 kB each.
	AHB peripherals	0x5000 0000 - 0x501F FFFF	DMA Controller, Ethernet interface, and USB interface.
0xE000 0000 to 0xE00F FFFF	Cortex-M3 Private Peripheral Bus	0xE000 0000 - 0xE00F FFFF	Cortex-M3 related functions, includes the NVIC and System Tick Timer.

lpc17xx.h: addressing elements

2) A start address is defined for every element, based on main blocks of memory.

```
955  /* APB1 peripherals
956  #define LPC_SSP0_BASE            (LPC_APB1_BASE + 0x08000)
957  #define LPC_DAC_BASE             (LPC_APB1_BASE + 0x0C000)
958  #define LPC_TIM2_BASE            (LPC_APB1_BASE + 0x10000)
959  #define LPC_TIM3_BASE            (LPC_APB1_BASE + 0x14000)
960  #define LPC_UART2_BASE           (LPC_APB1_BASE + 0x18000)
961  #define LPC_UART3_BASE           (LPC_APB1_BASE + 0x1C000)
962  #define LPC_I2C2_BASE            (LPC_APB1_BASE + 0x20000)
963  #define LPC_I2S_BASE             (LPC_APB1_BASE + 0x28000)
964  #define LPC_RIT_BASE             (LPC_APB1_BASE + 0x30000)
965  #define LPC_MCPWM_BASE           (LPC_APB1_BASE + 0x38000)
966  #define LPC_QEI_BASE             (LPC_APB1_BASE + 0x3C000)
967  #define LPC_SC_BASE              (LPC_APB1_BASE + 0x7C000)
968
```

APB1 peripheral addresses (page 17)

Table 5. APB1 peripherals and base addresses

APB1 peripheral	Base address	Peripheral name
0	0x4008 0000	reserved
1	0x4008 4000	reserved
2	0x4008 8000	SSP0
3	0x4008 C000	DAC
4	0x4009 0000	Timer 2
5	0x4009 4000	Timer 3
6	0x4009 8000	UART2
7	0x4009 C000	UART3
8	0x400A 0000	I^2C2
9	0x400A 4000	reserved
10	0x400A 8000	I^2S
11	0x400A C000	reserved
12	0x400B 0000	Repetitive interrupt timer
13	0x400B 4000	reserved
14	0x400B 8000	Motor control PWM
15	0x400B C000	Quadrature Encoder Interface
16 to 30	0x400C 0000 to 0x400F 8000	reserved
31	0x400F C000	System control

System control (page 19)

对于非特殊设备，系统控制块包括系统特点和控制寄存器

- The system control block includes system features and control registers for functions not related to specific peripheral devices:
  - reset source identification 重置资源识别
  - brown-out detection 低功率欠压检测
  - external interrupt inputs
  - system controls and status, related to main oscillator. 系统控制和状态，与主要振荡器相关
- Each function has its own registers (if needed)
  - unneeded bits are defined as reserved in order to allow future expansion.

每个功能有它自己的寄存器

不需要的位被预留，用来允许将来的扩展

lpc17xx.h: addressing registers

3) A structured list is defined for every element, in order to access its registers.

对于每个元素定义了一个结构表，用来访问它的寄存器

```
118 /*------------- System Control (SC) ------------------------------------------*/
119 /** @brief System Control (SC) register structure definition */
120 typedef struct
121 {
122   __IO uint32_t FLASHCFG;              /* Flash Accelerator Module          */
123        uint32_t RESERVED0[31];
124   __IO uint32_t PLL0CON;               /* Clocking and Power Control        */
125   __IO uint32_t PLL0CFG;
126   __I  uint32_t PLL0STAT;
127   __O  uint32_t PLL0FEED;
128        uint32_t RESERVED1[4];
129   __IO uint32_t PLL1CON;
130   __IO uint32_t PLL1CFG;
131   __I  uint32_t PLL1STAT;
132   __O  uint32_t PLL1FEED;
133        uint32_t RESERVED2[4];
134   __IO uint32_t PCON;
135   __IO uint32_t PCONP;
136        uint32_t RESERVED3[15];
137   __IO uint32_t CCLKCFG;
138   __IO uint32_t USBCLKCFG;
139   __IO uint32_t CLKSRCSEL;
140   __IO uint32_t CANSLEEPCLR;
141   __IO uint32_t CANWAKEFLAGS;
142        uint32_t RESERVED4[10];
143   __IO uint32_t EXTINT;                /* External Interrupts               */
144        uint32_t RESERVED5;
145   __IO uint32_t EXTMODE;
146   __IO uint32_t EXTPOLAR;
147        uint32_t RESERVED6[12];
148   __IO uint32_t RSID;                  /* Reset                             */
149        uint32_t RESERVED7[7];
150   __IO uint32_t SCS;                   /* Syscon Miscellaneous Registers    */
151   __IO uint32_t IRCTRIM;               /* Clock Dividers                    */
152   __IO uint32_t PCLKSEL0;
153   __IO uint32_t PCLKSEL1;
154        uint32_t RESERVED8[4];
155   __IO uint32_t USBIntSt;              /* USB Device/OTG Interrupt Register */
156   __IO uint32_t DMAREQSEL;
157   __IO uint32_t CLKOUTCFG;             /* Clock Output Configuration        */
158 } LPC_SC_TypeDef;
```

System control registers (page 26)

Table 9. External Interrupt registers

Name	Description	Access	Reset value[1]	Address
EXTINT	The External Interrupt Flag Register contains interrupt flags for EINT0, EINT1, EINT2 and EINT3. See Table 10.	R/W	0x00	0x400F C140
EXTMODE	The External Interrupt Mode Register controls whether each pin is edge- or level-sensitive. See Table 11.	R/W	0x00	0x400F C148
EXTPOLAR	The External Interrupt Polarity Register controls which level or edge on each pin will cause an interrupt. See Table 12.	R/W	0x00	0x400F C14C

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

```
118 /*------------- System Control (SC)
119 /** @brief System Control (SC) regis
120 typedef struct
121 {
122   __IO uint32_t FLASHCFG;          base
123        uint32_t RESERVED0[31];     +124
124   __IO uint32_t PLL0CON;           +4
125   __IO uint32_t PLL0CFG;           +4
126   __I  uint32_t PLL0STAT;          +4
127   __O  uint32_t PLL0FEED;          +4
128        uint32_t RESERVED1[4];      +16
129   __IO uint32_t PLL1CON;           +4
130   __IO uint32_t PLL1CFG;           +4
131   __I  uint32_t PLL1STAT;          +4
132   __O  uint32_t PLL1FEED;          +4
133        uint32_t RESERVED2[4];      +16
134   __IO uint32_t PCON;              +4
135   __IO uint32_t PCONP;             +4
136        uint32_t RESERVED3[15];     +60
137   __IO uint32_t CCLKCFG;           +4
138   __IO uint32_t USBCLKCFG;         +4
139   __IO uint32_t CLKSRCSEL;         +4
140   __IO uint32_t CANSLEEPCLR;       +4
141   __IO uint32_t CANWAKEFLAGS;      +4
142        uint32_t RESERVED4[10];     +40
143   __IO uint32_t EXTINT;            +4
144        uint32_t RESERVED5;
145   __IO uint32_t EXTMODE;           .....
146   __IO uint32_t EXTPOLAR;
147        uint32_t RESERVED6[12];
148   __IO uint32_t RSID;
149        uint32_t RESERVED7[7];
150   __IO uint32_t SCS;
151   __IO uint32_t IRCTRIM;
152   __IO uint32_t PCLKSEL0;
153   __IO uint32_t PCLKSEL1;
154        uint32_t RESERVED8[4];
155   __IO uint32_t USBIntSt;
156   __IO uint32_t DMAREQSEL;
157   __IO uint32_t CLKOUTCFG;
158 } LPC_SC_TypeDef;
```

lpc17xx.h: accessing symbols

```
989 /*****************************************************************************/
990 /*                         Peripheral declaration                            */
991 /*****************************************************************************/
992 #define LPC_SC              ((LPC_SC_TypeDef        *) LPC_SC_BASE      )
993 #define LPC_GPIO0           ((LPC_GPIO_TypeDef      *) LPC_GPIO0_BASE   )
994 #define LPC_GPIO1           ((LPC_GPIO_TypeDef      *) LPC_GPIO1_BASE   )
995 #define LPC_GPIO2           ((LPC_GPIO_TypeDef      *) LPC_GPIO2_BASE   )
996 #define LPC_GPIO3           ((LPC_GPIO_TypeDef      *) LPC_GPIO3_BASE   )
997 #define LPC_GPIO4           ((LPC_GPIO_TypeDef      *) LPC_GPIO4_BASE   )
998 #define LPC_WDT             ((LPC_WDT_TypeDef       *) LPC_WDT_BASE     )
999
1000
1001
1002
1003
1004 #define LPC_UART0           ((LPC_UART_TypeDef      *) LPC_UART0_BASE   )
1005 #define LPC_UART1           ((LPC_UART1_TypeDef     *) LPC_UART1_BASE   )
1006 #define LPC_UART2           ((LPC_UART_TypeDef      *) LPC_UART2_BASE   )
1007 #define LPC_UART3           ((LPC_UART_TypeDef      *) LPC_UART3_BASE   )
1008 #define LPC_PWM1            ((LPC_PWM_TypeDef       *) LPC_PWM1_BASE    )
1009 #define LPC_I2C0            ((LPC_I2C_TypeDef       *) LPC_I2C0_BASE    )
1010 #define LPC_I2C1            ((LPC_I2C_TypeDef       *) LPC_I2C1_BASE    )
1011 #define LPC_I2C2            ((LPC_I2C_TypeDef       *) LPC_I2C2_BASE    )
1012 #define LPC_I2S             ((LPC_I2S_TypeDef       *) LPC_I2S_BASE     )
1013 #define LPC_SPI             ((LPC_SPI_TypeDef       *) LPC_SPI_BASE     )
1014 #define LPC_RTC             ((LPC_RTC_TypeDef       *) LPC_RTC_BASE     )
1015 #define LPC_GPIOINT         ((LPC_GPIOINT_TypeDef   *) LPC_GPIOINT_BASE )
1016 #define LPC_PINCON          ((LPC_PINCON_TypeDef    *) LPC_PINCON_BASE  )
1017 #define LPC_SSP0            ((LPC_SSP_TypeDef       *) LPC_SSP0_BASE    )
```

Accessible name from C code

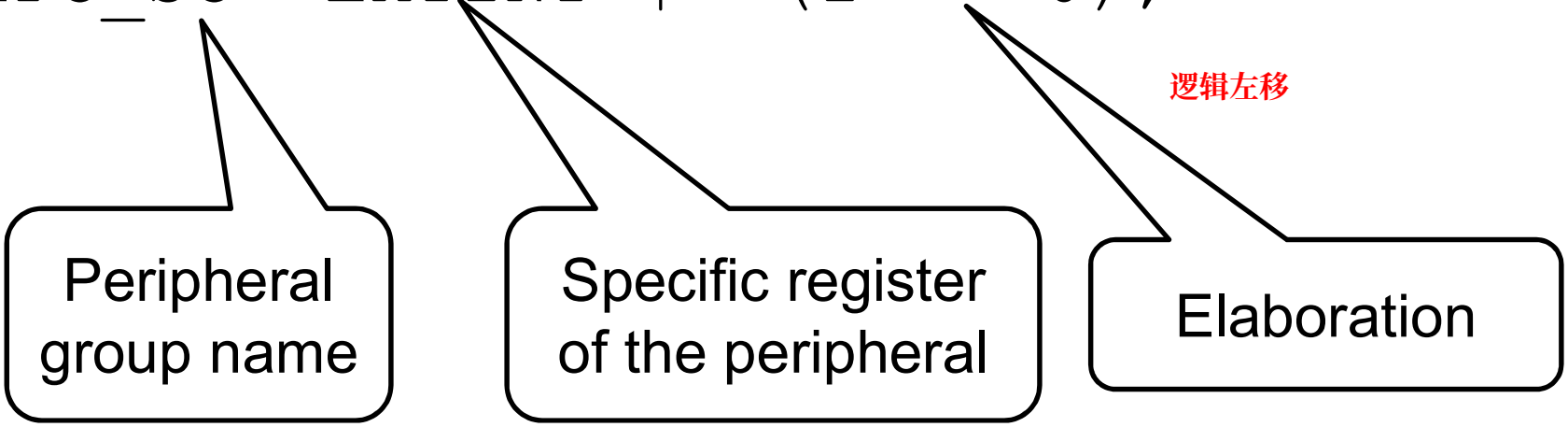
Cast as a pointer to the relative type

Address of every specific SoC resource

Use of definitions in lpc17xx.h

- Symbols defined in lpc17xx.h simplify addressing peripheral registers in the project.
- Example in lib_led.c:

```
LPC_SC->EXTINT |= (1 << 0);
```



Files in sample project

- startup_LPC17xx.s
- system_LPC17xx.c
  - *SystemInit()* function called by reset_handler
- sample.c
- led group
- button group
  - lib_button.c: inizialitation of buttons and NVIC
  - IRQ_button.c: handlers for external interrupts
- **system libraries**
  - lpc17xx.h
  - **core_cm3.h**

核心硬件寻址

core_cm3.h: addressing core hardware

1) Constants are defined for addressing main core blocks.

常量被定义用来寻址主要核心块

2) A start address is defined for every core element, based on main core blocks.

- It is the same mechanism used in lpc17xx.h for memory mapping of board elements.

```
1374 /* Memory mapping of Core Hardware */
1375 #define SCS_BASE            (0xE000E000UL)                    /*!< System Control Space Base Address */
1376 #define ITM_BASE            (0xE0000000UL)                    /*!< ITM Base Address */
1377 #define DWT_BASE            (0xE0001000UL)                    /*!< DWT Base Address */
1378 #define TPI_BASE            (0xE0040000UL)                    /*!< TPI Base Address */
1379 #define CoreDebug_BASE      (0xE000EDF0UL)                    /*!< Core Debug Base Address */
1380 #define SysTick_BASE        (SCS_BASE +  0x0010UL)            /*!< SysTick Base Address */
1381 #define NVIC_BASE           (SCS_BASE +  0x0100UL)            /*!< NVIC Base Address */
1382 #define SCB_BASE            (SCS_BASE +  0x0D00UL)            /*!< System Control Block Base Address */
```

core_cm3.h: addressing registers

3) A structured list is defined for every core element, in order to access its registers.

```
337 /**
338   \brief  Structure type to access the Nested Vectored Interrupt Controller (NVIC).
339  */
340 typedef struct
341 {
342   __IOM uint32_t ISER[8U];               /*!< Offset: 0x000 (R/W)  Interrupt Set Enable Register */
343         uint32_t RESERVED0[24U];
344   __IOM uint32_t ICER[8U];               /*!< Offset: 0x080 (R/W)  Interrupt Clear Enable Register */
345         uint32_t RESERVED1[24U];
346   __IOM uint32_t ISPR[8U];               /*!< Offset: 0x100 (R/W)  Interrupt Set Pending Register */
347         uint32_t RESERVED2[24U];
348   __IOM uint32_t ICPR[8U];               /*!< Offset: 0x180 (R/W)  Interrupt Clear Pending Register */
349         uint32_t RESERVED3[24U];
350   __IOM uint32_t IABR[8U];               /*!< Offset: 0x200 (R/W)  Interrupt Active bit Register */
351         uint32_t RESERVED4[56U];
352   __IOM uint8_t  IP[240U];               /*!< Offset: 0x300 (R/W)  Interrupt Priority Register (8Bit wide) */
353         uint32_t RESERVED5[644U];
354   __OM  uint32_t STIR;                   /*!< Offset: 0xE00 ( /W)  Software Trigger Interrupt Register */
355 } NVIC_Type;
356
```

NVIC registers (page 78)

Table 51. NVIC register map

Name	Description	Access	Reset value	Address
ISER0 to ISER1	Interrupt Set-Enable Registers. These 2 registers allow enabling interrupts and reading back the interrupt enables for specific peripheral functions.	RW	0	ISER0 - 0xE000 E100 ISER1 - 0xE000 E104
ICER0 to ICER1	Interrupt Clear-Enable Registers. These 2 registers allow disabling interrupts and reading back the interrupt enables for specific peripheral functions.	RW	0	ICER0 - 0xE000 E180 ICER1 - 0xE000 E184
ISPR0 to ISPR1	Interrupt Set-Pending Registers. These 2 registers allow changing the interrupt state to pending and reading back the interrupt pending state for specific peripheral functions.	RW	0	ISPR0 - 0xE000 E200 ISPR1 - 0xE000 E204
ICPR0 to ICPR1	Interrupt Clear-Pending Registers. These 2 registers allow changing the interrupt state to not pending and reading back the interrupt pending state for specific peripheral functions.	RW	0	ICPR0 - 0xE000 E280 ICPR1 - 0xE000 E284
IABR0 to IABR1	Interrupt Active Bit Registers. These 2 registers allow reading the current interrupt active state for specific peripheral functions.	RO	0	IABR0 - 0xE000 E300 IABR1 - 0xE000 E304
IPR0 to IPR8	Interrupt Priority Registers. These 9 registers allow assigning a priority to each interrupt. Each register contains the 5-bit priority fields for 4 interrupts.	RW	0	IPR0 - 0xE000 E400 IPR1 - 0xE000 E404 IPR2 - 0xE000 E408 IPR3 - 0xE000 E40C IPR4 - 0xE000 E410 IPR5 - 0xE000 E414 IPR6 - 0xE000 E418 IPR7 - 0xE000 E41C IPR8 - 0xE000 E420
STIR	Software Trigger Interrupt Register. This register allows software to generate an interrupt.	WO	0	STIR - 0xE000 EF00

```
typedef struct
{
  __IOM uint32_t ISER[8U];          base
        uint32_t RESERVED0[24U];    +32
  __IOM uint32_t ICER[8U];          +96
        uint32_t RESERVED1[24U];    +32
  __IOM uint32_t ISPR[8U];          +96
        uint32_t RESERVED2[24U];    +32
  __IOM uint32_t ICPR[8U];          +96
        uint32_t RESERVED3[24U];    +32
  __IOM uint32_t IABR[8U];          +96
        uint32_t RESERVED4[56U];    +32
  __IOM uint8_t  IP[240U];          +224
        uint32_t RESERVED5[644U];...
  __OM  uint32_t STIR;
}  NVIC_Type;
```

core_cm3.h: accessing symbols

```
1384 #define SCnSCB              ((SCnSCB_Type    *)     SCS_BASE      )   /*!< System control Register not in SCB */
1385 #define SCB                 ((SCB_Type       *)     SCB_BASE      )   /*!< SCB configuration struct */
1386 #define SysTick             ((SysTick_Type   *)     SysTick_BASE  )   /*!< SysTick configuration struct */
1387 #define NVIC                ((NVIC_Type      *)     NVIC_BASE     )   /*!< NVIC configuration struct */
1388 #define ITM                 ((ITM_Type       *)     ITM_BASE      )   /*!< ITM configuration struct */
1389 #define DWT                 ((DWT_Type       *)     DWT_BASE      )   /*!< DWT configuration struct */
1390 #define TPI                 ((TPI_Type       *)     TPI_BASE      )   /*!< TPI configuration struct */
1391 #define CoreDebug           ((CoreDebug_Type *)     CoreDebug_BASE)   /*!< Core Debug configuration struct */
```

Accessible name from C code

Cast as a pointer to the relative type

Address of every specific SoC resource

Use of definitions in core_cm3.h

- Usually core registers are not directly addressed in the project.
- Symbols defined in core_cm3.h are used in this file only, for implementing functions.
- Then, functions are invoked in the project.

然后在项目中调用函数

```
1497 /**
1498   \brief   Enable Interrupt
1499   \details Enables a device specific interrupt in the NVIC interrupt controller.
1500   \param [in]      IRQn  Device specific interrupt number.
1501   \note    IRQn must not be negative.
1502  */
1503 __STATIC_INLINE void __NVIC_EnableIRQ(IRQn_Type IRQn)
1504 {
1505   if ((int32_t)(IRQn) >= 0)
1506   {
1507     NVIC->ISER[(((uint32_t)IRQn) >> 5UL)] = (uint32_t)(1UL << (((uint32_t)IRQn) & 0x1FUL));
1508   }
1509 }
```

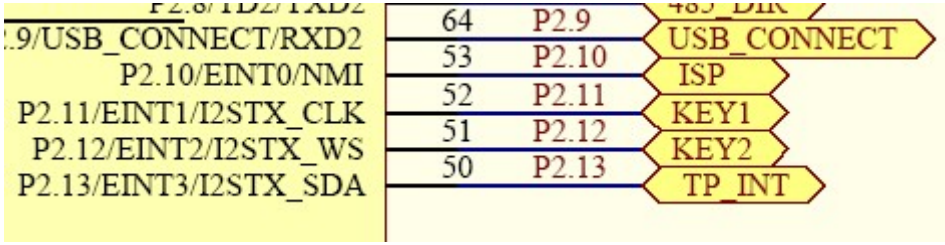
Files in sample project

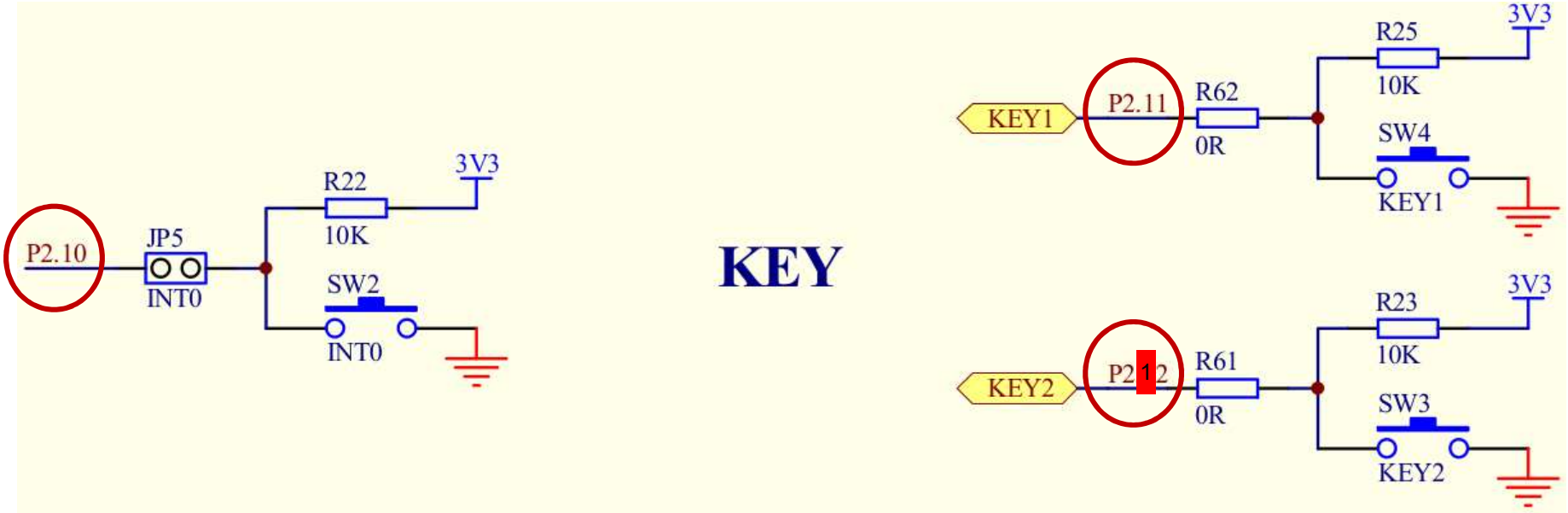
- startup_LPC17xx.s
- system_LPC17xx.c
  - *SystemInit()* function called by reset_handler
- sample.c
- led group
- **button group**
  - **lib_button.c: inizialitation of buttons and NVIC**
  - IRQ_button.c: handlers for external interrupts
- system libraries
  - lpc17xx.h
  - core_cm3.h

Buttons

- INT0 -> p2.10
- KEY1 -> p2.11
- KEY2 -> p2.12

- released = 1
- pressed = 0

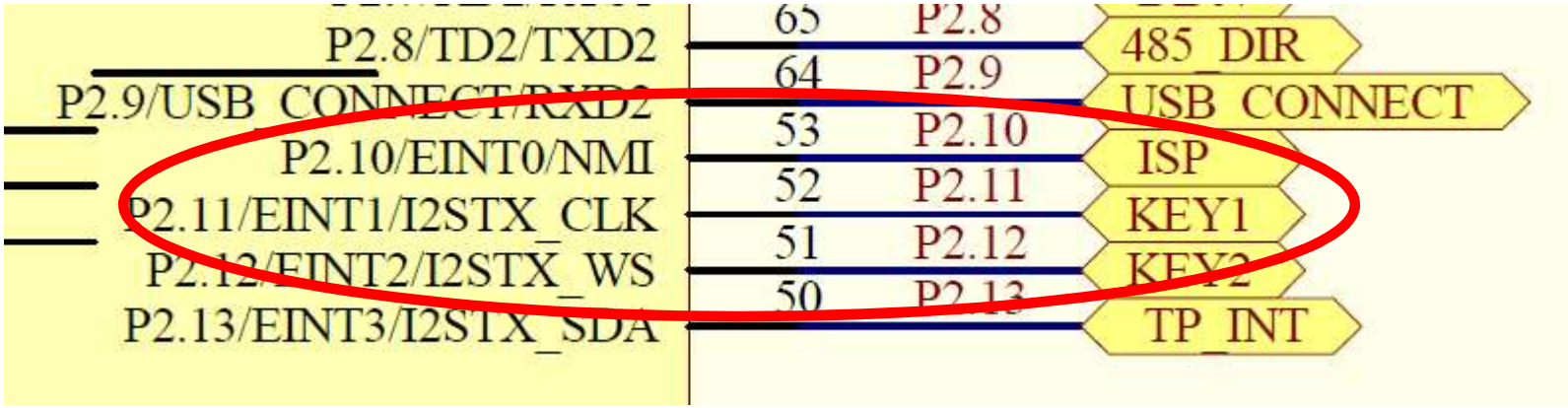




引脚选择和方向

Pin selection and direction

- After discovering the interested CPU pins, you have to provide additional information to the SoC:
  在发现了相关的CPU PIN脚之后，你必须提供额外的信息到SoC：
  - the pin functionality
    PIN脚的功能
  - the pin direction (input or output)
    pin脚的方向（输入或者输出）



Pin connect block (pag. 119)

Table 84. Pin function select register 4 (PINSEL4 - address 0x4002 C010) bit description

PINSEL4	Pin name	Function when 00	Function when 01	Function when 10	Function when 11	Reset value
1:0	P2.0	GPIO Port 2.0	PWM1.1	TXD1	Reserved	00
3:2	P2.1	GPIO Port 2.1	PWM1.2	RXD1	Reserved	00
5:4	P2.2	GPIO Port 2.2	PWM1.3	CTS1	Reserved [2]	00
7:6	P2.3	GPIO Port 2.3	PWM1.4	DCD1	Reserved [2]	00
9:8	P2.4	GPIO Port 2.4	PWM1.5	DSR1	Reserved [2]	00
11:10	P2.5	GPIO Port 2.5	PWM1.6	DTR1	Reserved [2]	00
13:12	P2.6	GPIO Port 2.6	PCAP1.0	RI1	Reserved [2]	00
15:14	P2.7	GPIO Port 2.7	RD2	RTS1	Reserved	00
17:16	P2.8	GPIO Port 2.8	TD2	TXD2	ENET_MDC	00
19:18	P2.9	GPIO Port 2.9	USB_CONNECT	RXD2	ENET_MDIO	00
21:20	P2.10	GPIO Port 2.10	$\overline{\text{EINT0}}$	NMI	Reserved	00
23:22	P2.11 [1]	GPIO Port 2.11	$\overline{\text{EINT1}}$	Reserved	I2STX_CLK	00
25:24	P2.12 [1]	GPIO Port 2.12	$\overline{\text{EINT2}}$	Reserved	I2STX_WS	00
27:26	P2.13 [1]	GPIO Port 2.13	$\overline{\text{EINT3}}$	Reserved	I2STX_SDA	00
31:28	-	Reserved	Reserved	Reserved	Reserved	0

External interrupt inputs (page 26)

Table 9. External Interrupt registers

Name	Description	Access	Reset value[1]	Address
EXTINT	The External Interrupt Flag Register contains interrupt flags for EINT0, EINT1, EINT2 and EINT3. See Table 10.	R/W	0x00	0x400F C140
EXTMODE	The External Interrupt Mode Register controls whether each pin is edge- or level-sensitive. See Table 11.	R/W	0x00	0x400F C148
EXTPOLAR	The External Interrupt Polarity Register controls which level or edge on each pin will cause an interrupt. See Table 12.	R/W	0x00	0x400F C14C

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

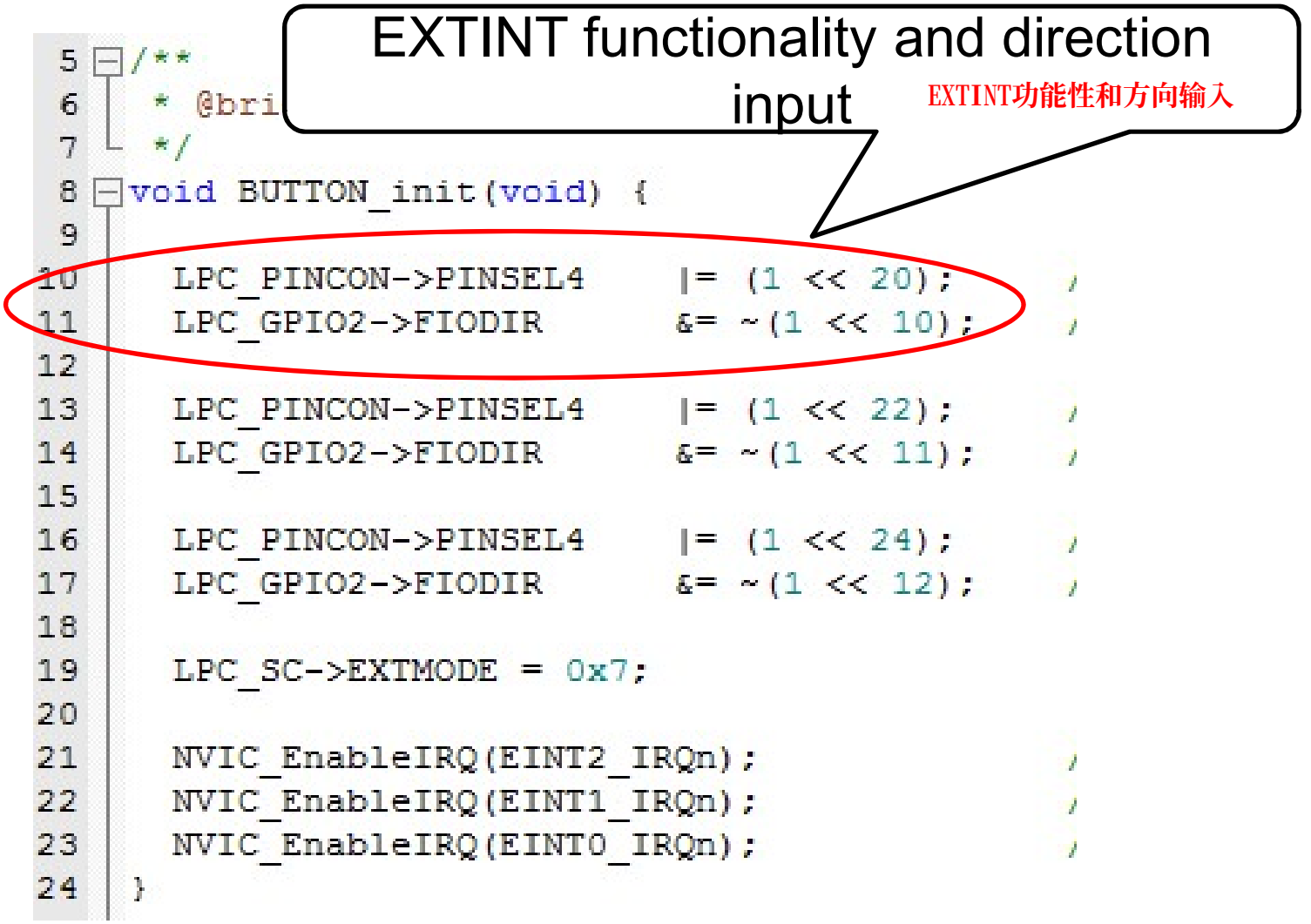
Table 11. External Interrupt Mode register (EXTMODE - address 0x400F C148) bit description

Bit	Symbol	Value	Description	Reset value
0	EXTMODE0	0	Level-sensitivity is selected for $\overline{\text{EINT0}}$.	0
		1	$\overline{\text{EINT0}}$ is edge sensitive.	

Table 12. External Interrupt Polarity register (EXTPOLAR - address 0x400F C14C) bit description

Bit	Symbol	Value	Description	Reset value
0	EXTPOLAR0	0	$\overline{\text{EINT0}}$ is low-active or falling-edge sensitive (depending on EXTMODE0).	0
		1	$\overline{\text{EINT0}}$ is high-active or rising-edge sensitive (depending on EXTMODE0).	

Direction and EXTINT configuration




```
5  /**
6   * @bri
7   */
8  void BUTTON_init(void) {
9
10    LPC_PINCON->PINSEL4    |= (1 << 20);
11    LPC_GPIO2->FIODIR      &= ~(1 << 10);
12
13    LPC_PINCON->PINSEL4    |= (1 << 22);
14    LPC_GPIO2->FIODIR      &= ~(1 << 11);
15
16    LPC_PINCON->PINSEL4    |= (1 << 24);
17    LPC_GPIO2->FIODIR      &= ~(1 << 12);
18
19    LPC_SC->EXTMODE = 0x7;
20
21    NVIC_EnableIRQ(EINT2_IRQn);
22    NVIC_EnableIRQ(EINT1_IRQn);
23    NVIC_EnableIRQ(EINT0_IRQn);
24 }
```

PinSel setup

```
 5 /**
 6  * @brief  Function that initializes Button
 7  */
 8 void BUTTON_init(void) {
 9
10   LPC_PINCON->PINSEL4    |= (1 << 20);     /
11   LPC_GPIO2->FIODIR      &= ~(1 << 10);    /
12
13   LPC_PI
14   LPC_GP
15
16   LPC_PINCON->PIN          = (1 << 24);    /
17   LPC_GPIO2->FIODIR        &= ~(1 << 12);  /
18
19   LPC_SC->EXTMODE = 0x7;
20
21   NVIC_EnableIRQ(EINT2_IRQn);              /
22   NVIC_EnableIRQ(EINT1_IRQn);              /
23   NVIC_EnableIRQ(EINT0_IRQn);              /
24 }
```

EXTINT pins mode: edge sensitive

边缘敏感

lib_button.c – setup PinSel, direction and ExtINT config

```
 5 /**
 6  * @brief  Function that initializes Button
 7  */
 8 void BUTTON_init(void) {
 9
10   LPC_PINCON->PINSEL4    |= (1 << 20);
11   LPC_GPIO2->FIODIR      &= ~(1 << 10);
12
13   LPC_PINCON->PINSEL4    |= (1 << 22);
14   L
15
16   L
17   L
18
19   LPC_SC->EX             = 0x7;
20
21   NVIC_EnableIRQ(EINT2_IRQn);
22   NVIC_EnableIRQ(EINT1_IRQn);
23   NVIC_EnableIRQ(EINT0_IRQn);
24 }
```

Nested Vectored Interrupt Controller (NVIC) selective enable of external interrupts

外部中断选择启动

Files in sample project

- startup_LPC17xx.s
- system_LPC17xx.c
  - *SystemInit()* function called by reset_handler
- sample.c
- led group
- **button group**
  - lib_button.c: inizialitation of buttons and NVIC
  - **IRQ_button.c: handlers for external interrupts**
- system libraries
  - lpc17xx.h
  - core_cm3.h

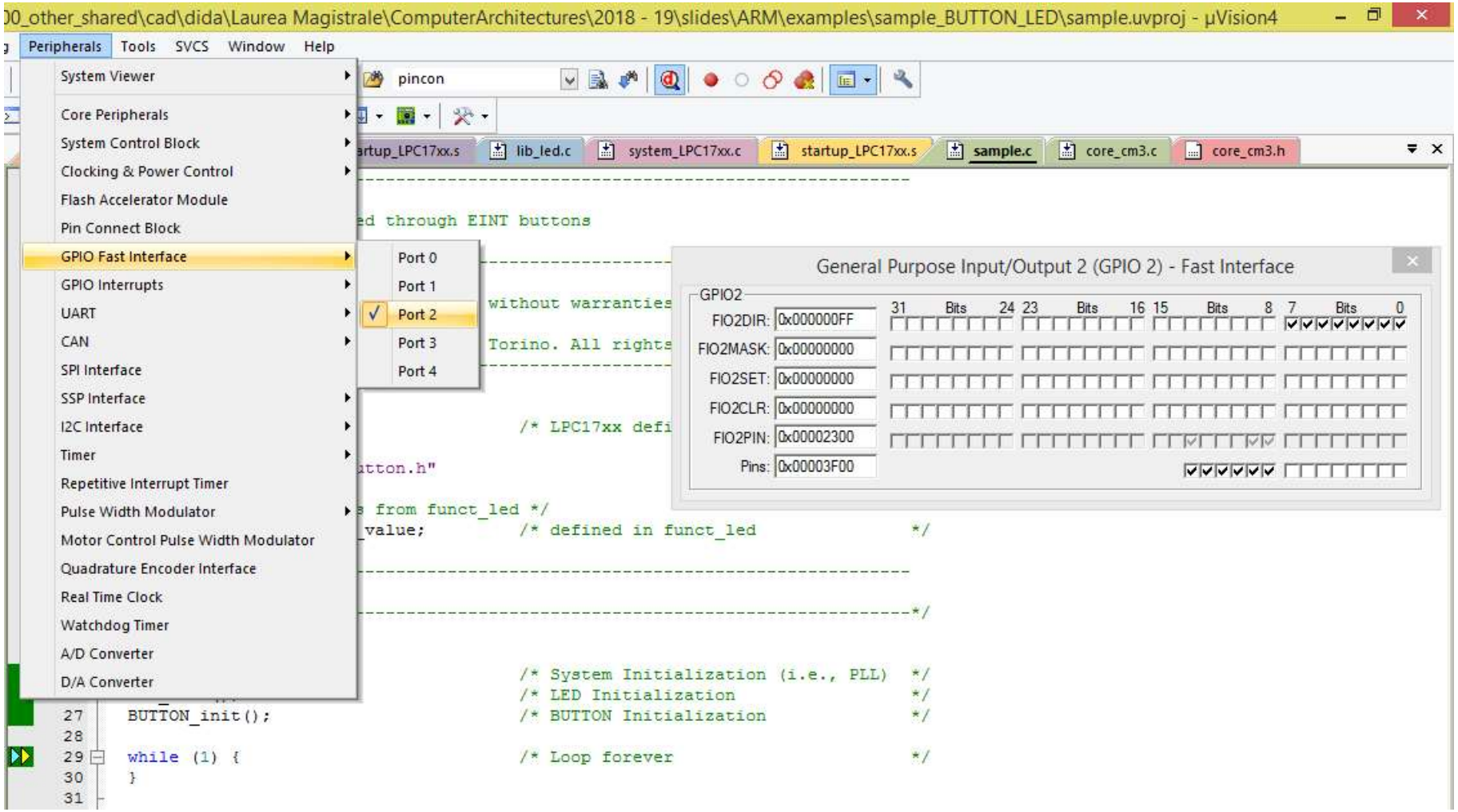
Interruption handler and IVT

```
 6  void EINT0_IRQHandler (void)
 7  {
 8    LED_On(0);
 9    LPC_SC->EXTINT |= (1 << 0);     /* clear pending interrupt
10  }
```

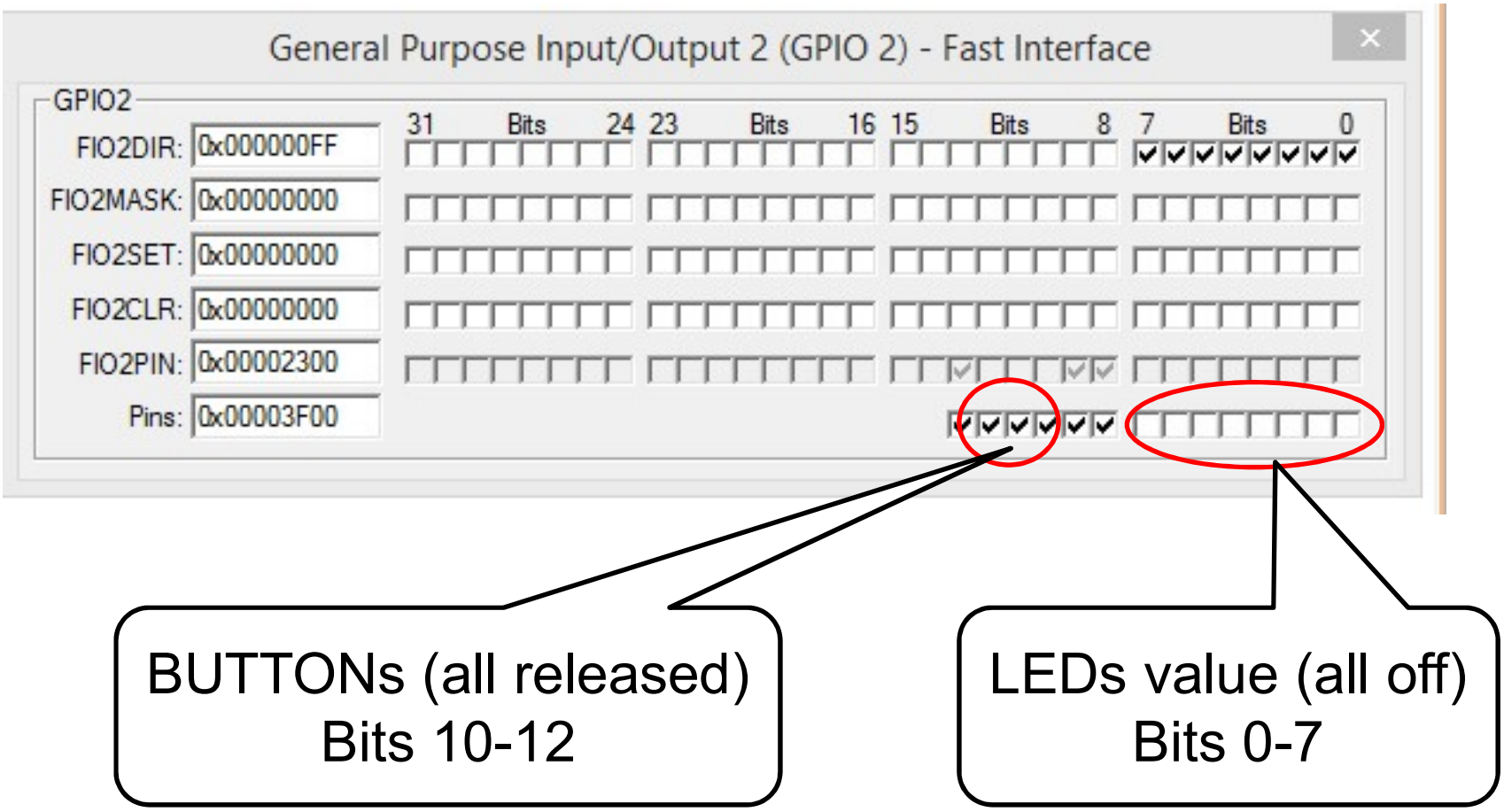
```
52                AREA    RESET, DATA, READONLY
53                EXPORT  __Vectors
54
55  __Vectors     DCD     __initial_sp              ; Top of Stack
56                DCD     Reset_Handler             ; Reset Handler
57                DCD     NMI_Handler               ; NMI Handler
58                DCD     HardFault_Handler         ; Hard Fault Handler
59                DCD     MemManage_Handler         ; MPU Fault Handler
60                DCD     BusFault_Handler          ; Bus Fault Handler
61                DCD     UsageFault_Handler        ; Usage Fault Handler
62                DCD     0                         ; Reserved
63                DCD     EINT0_IRQHandler          ; 34: External Interrupt 0
64                DCD     EINT1_IRQHandler          ; 35: External Interrupt 1
65                DCD     EINT2_IRQHandler          ; 36: External Interrupt 2
66                DCD     EINT3_IRQHandler          ; 37: External Interrupt 3
67                DCD     _                         ; 38: Reserved
68                DCD     0                         ; Reserved
69                DCD     PendSV_Handler            ; PendSV Handler
70                DCD     SysTick_Handler           ; SysTick Handler
71
72                ; External Interrupts
73                DCD     WDT_IRQHandler            ; 16: Watchdog Timer
74                DCD     TIMER0_IRQHandler         ; 17: Timer0
75                DCD     TIMER1_IRQHandler         ; 18: Timer1
```

触发一个外部中断

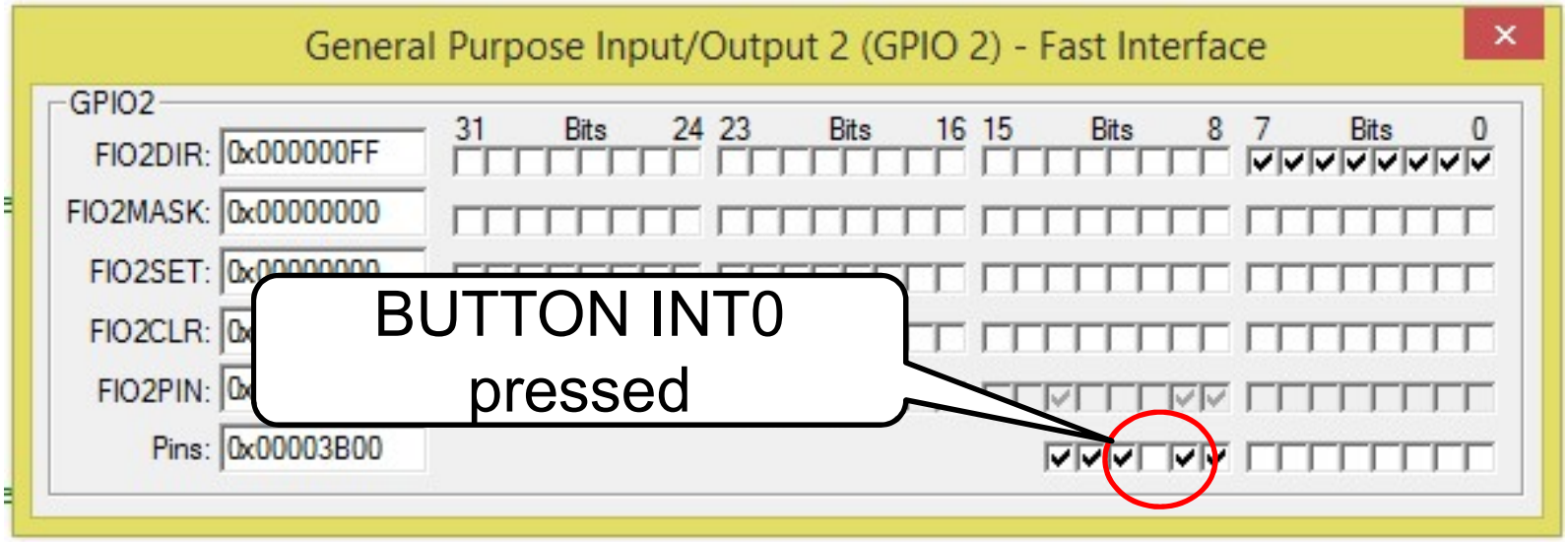
Triggering an external interruption



GPIO mask


General Purpose Input/Output 2 (GPIO 2) - Fast Interface
GPIO2
FIO2DIR: 0x000000FF
FIO2MASK: 0x00000000
FIO2SET: 0x00000000
FIO2CLR: 0x00000000
FIO2PIN: 0x00002300
Pins: 0x00003F00
31 Bits 24 23 Bits 16 15 Bits 8 7 Bits 0
BUTTONs (all released)
Bits 10-12
LEDs value (all off)
Bits 0-7

Triggering an external interruption




```
5
6   void EINT0_IRQHandler (void)
7   {
8     LED_On(0);
9     LPC_SC->EXTINT |= (1 << 0);     /* clear pending in
10  }
11
```

Peripheral priorities

外围设备优先级

- In complex systems, it is often necessary to assign a priority to each peripheral, and to ensure that the service procedure of an interrupt is only interrupted by an interrupt request coming from a lower priority device.
- In some cases, priorities assigned to peripherals can change over time.
- The interrupt controller manages the priorities.

在一个复杂的系统中，他通常给每一个外围设备分配一个优先级，并且确保中断服务进程只能被低优先级设备请求打断。

在某些情况下，优先级会随着时间改变

Simultaneous requests

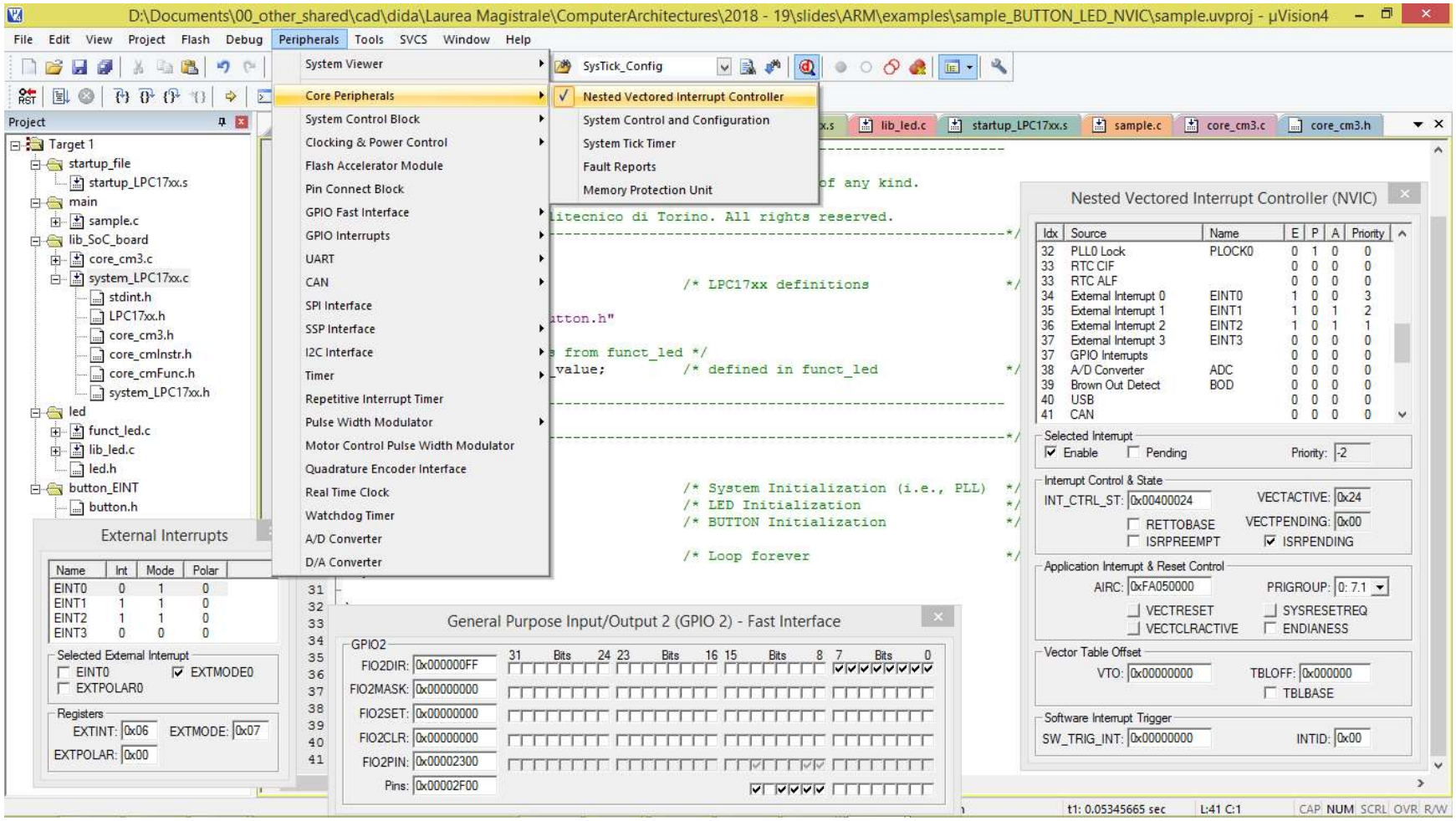
并发请求

同时中断请求，最高优先级的设备必须首先被服务

- With simultaneous interrupt requests, the highest priority device must be served first.
- In case of multiple interrupt lines, the processor solves the problem. 在多中断线程中，processor解决这个问题
- In case of polling, the peripherals are served in the same order as they are polled. 对于缓冲池，根据入池的顺序被服务
- In case of vectorized interrupt, the Interrupt Controller decides which is the first peripheral to be served, and provides the CPU with its ID code.

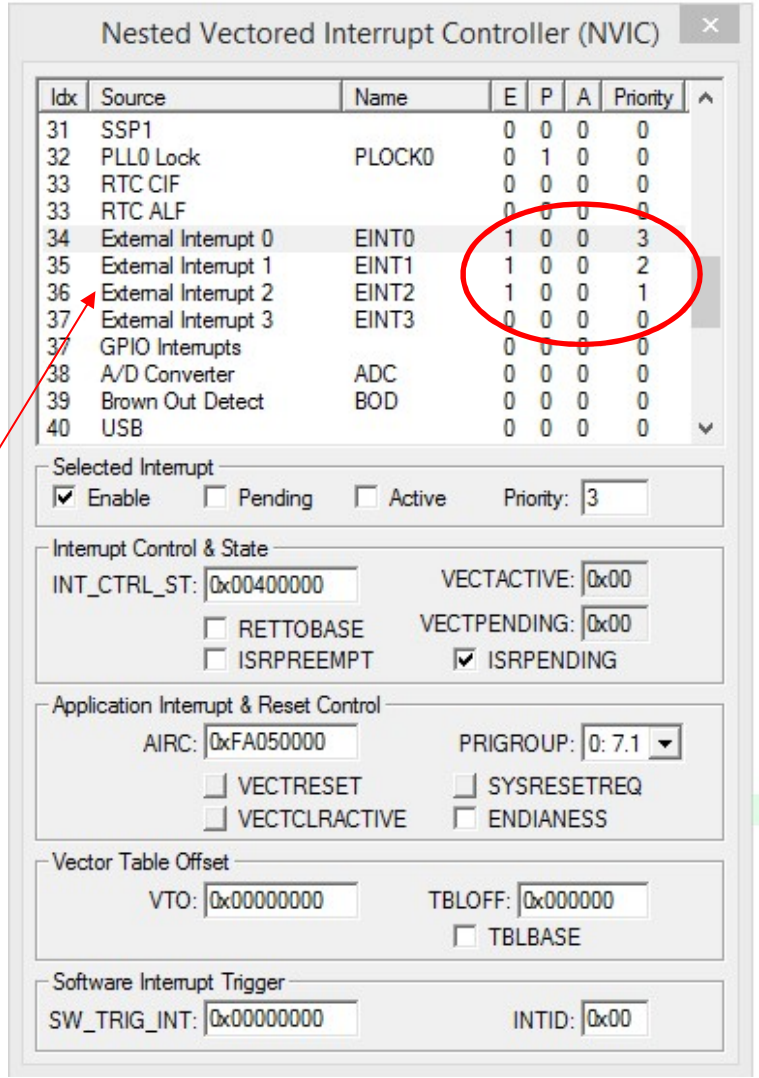
对于向量中断，中断控制器决定，并且提供给CPU它的设备码

Experimenting priority



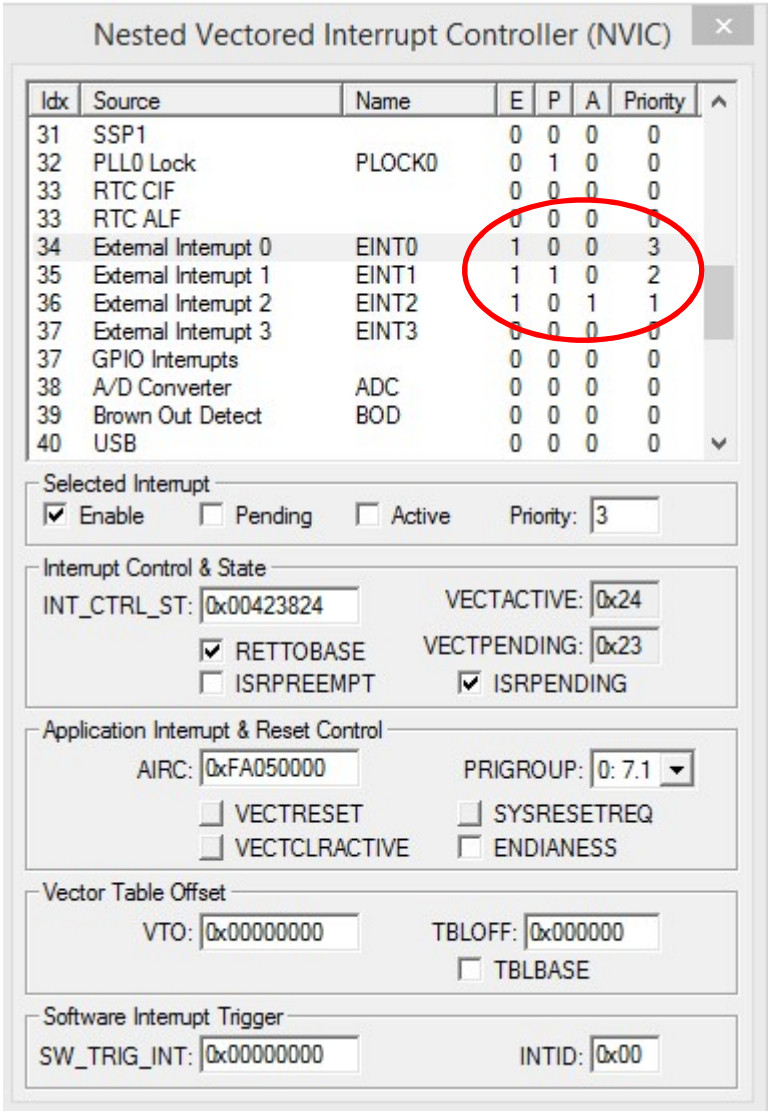
Boot

```
 8  void BUTTON_init(void) {
 9
10    LPC_PINCON->PINSEL4    |= (1 << 20);
11    LPC_GPIO2->FIODIR      &= ~(1 << 10);
12
13    LPC_PINCON->PINSEL4    |= (1 << 22);
14    LPC_GPIO2->FIODIR      &= ~(1 << 11);
15
16    LPC_PINCON->PINSEL4    |= (1 << 24);
17    LPC_GPIO2->FIODIR      &= ~(1 << 12);
18
19    LPC_SC->EXTMODE = 0x7;
20
21    NVIC_EnableIRQ(EINT2_IRQn);
22    NVIC_SetPriority(EINT2_IRQn, 1);
23    NVIC_EnableIRQ(EINT1_IRQn);
24    NVIC_SetPriority(EINT1_IRQn, 2);
25    NVIC_EnableIRQ(EINT0_IRQn);
26    NVIC_SetPriority(EINT0_IRQn, 3);
27  }
28
```



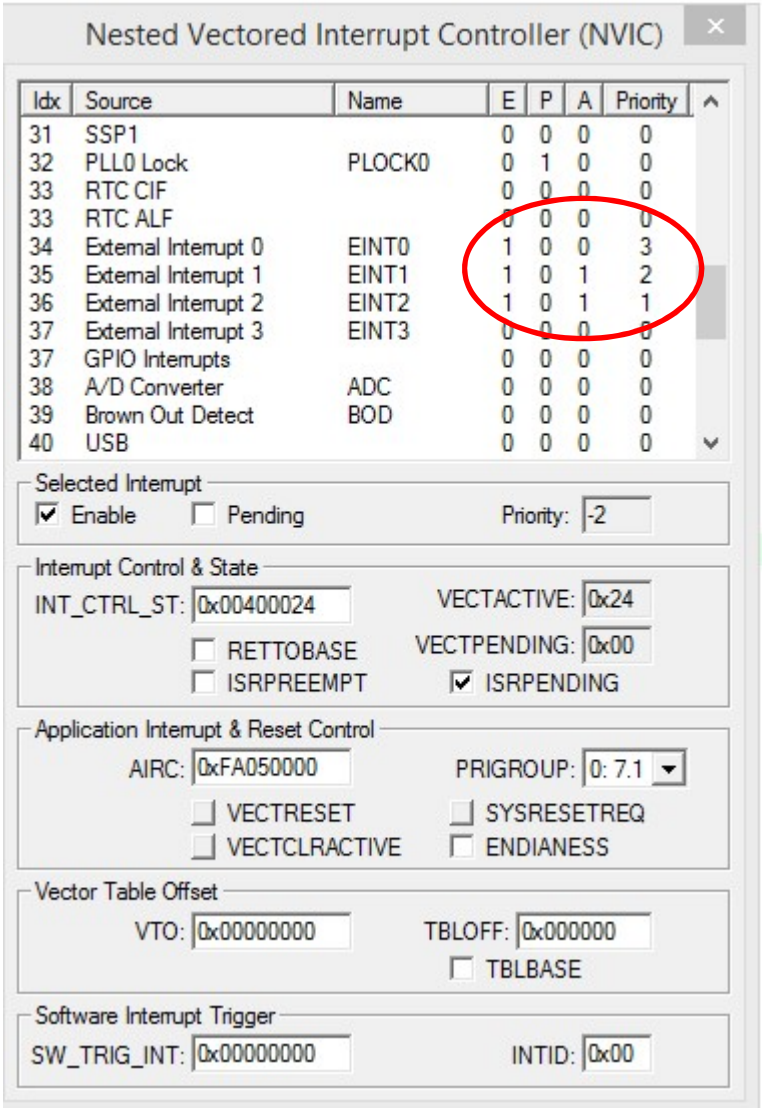
Runtime: case 1

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.

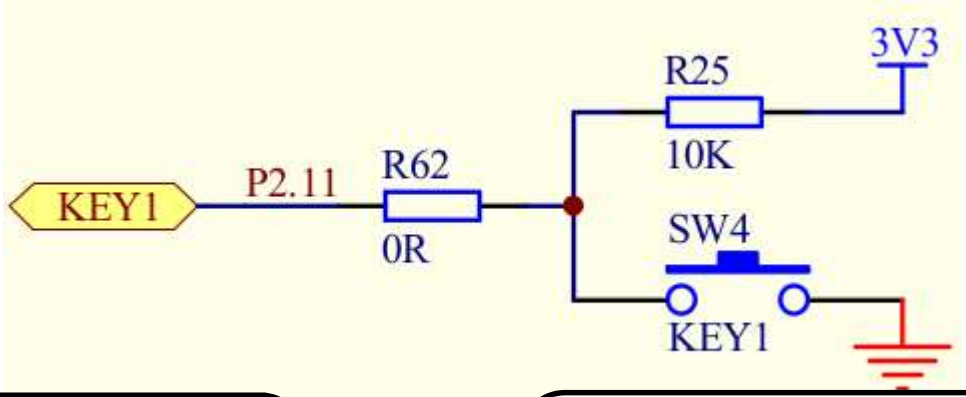


Runtime: case 2

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.



Button bouncing



When pressed, the button drives the logic value ‘0’

When released, the button drives the logic value ‘1’

- ideal

- real