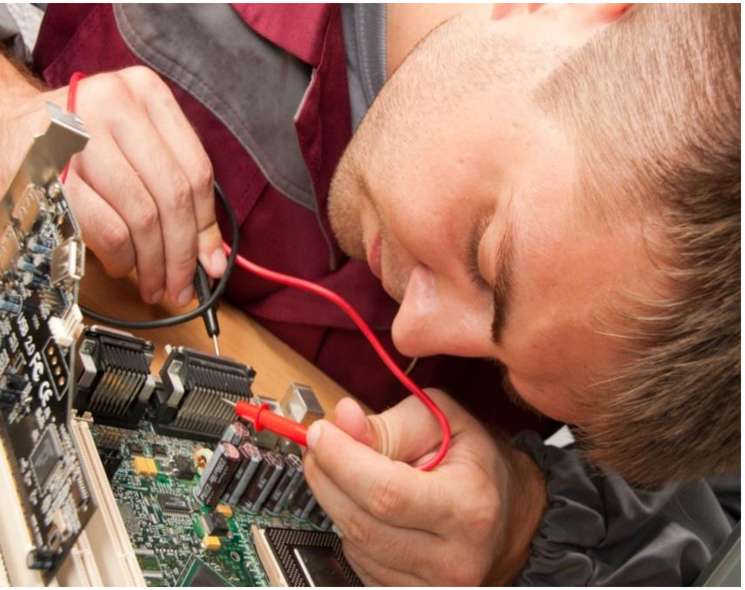


Microcontroller Programming (5)

Gerald Kupris, 05.11.2013

innovativ & lebendig – Bildungsregion DonauWald

Lectures Microcontroller Programming WS2013/14

08.10.2013 Microcontroller, Programming and Debuging Interfaces
15.10.2013 Reading and Writing of Registers
22.10.2013 I/O-Pins, Reading and Writing of Single Bits
29.10.2013 Clock Generation, CPU und Computing Power
05.11.2013 Interrupts
12.11.2013 No lecture !
19.11.2013 Memory
26.11.2013 Timer and PWM, Watchdog Timer
03.12.2013 Analog to Digital Converter
10.12.2013 Serial Interfaces: SPI, IIC and UART
17.12.2013 Additional Explanation of the Freescale Cup Cars
14.01.2014 Project Work on the Freescale Cup Cars
21.01.2014 Project Work on the Freescale Cup Cars

Hands-On Workshops Microcontroller Programming

08.10.2013 Workshop 1: Preparation of the Work Place
15.10.2013 Workshop 2: Loading and Debugging of Programs
22.10.2013 Workshop 3: Using the GPIO Pins
29.10.2013 Workshop 4: Clock Generation and Calculations
05.11.2013 Workshop 5: Interrupts
12.11.2013 No Workshop !
19.11.2013 Workshop 6: Using the Flash Memory
26.11.2013 Workshop 7: Timer and Pulse Width Modulation (PWM)
03.12.2013 Workshop 8: Analog to Digital Conversion
10.12.2013 Workshop 9: Serial Communication
17.12.2013 Project work on the Freescale Cup Cars
14.01.2014 Project work on the Freescale Cup Cars
21.01.2014 Project work on the Freescale Cup Cars

Participation on all workshops is required for admittance to the final project!

Start Time Tuesday: 15:45 p.m.
New Start Time Thursday: 14:45 p.m.

Attention: No Workshop on the 07.11.2013!

Block 1: 08:00 - 09:30
Block 2: 09:45 - 11:15
Block 3: 12:00 - 13:30

3. Semester Bachelor AI (Stand: 12.09.2013)

07.11.2013

Block 4: 14:00 - 15:30
Block 5: 15:45 - 17:15
Block 6: 17:30 - 19:00

	Montag		Dienstag		Mittwoch	Donnerstag		Freitag	
1		Vertiefte Elektrotechnik Bö D 113		Digitaltechnik Bö E 101	SW-Engineering gem. mit MT 5 Jr ITC1-E 104		Messtechnik (bis Mitte Nov) Wu E 101	Einführung GIS LB Zink E 101	
2		Messtechnik (ab Mitte Nov) Bö D 113		Digitaltechnik Praktikum Gruppe 1/2 (14-tägig) LabIng	SW-Engineering gem. mit MT 5 Jr ITC1-E 104	Bezugssysteme und Positionierung LB Reidelstürz ITC1-E 104	Messtechnik (bis Mitte Nov) Wu E 101	Einführung GIS LB Zink E 101	
3		Messtechnik (ab Mitte Nov) Bö A 210		Digitaltechnik Praktikum Gruppe 1/2 (14-tägig) LabIng	SW-Engineering gem. mit MT 5 Jr ITC1-E 103	Bezugssysteme und Positionierung LB Reidelstürz ITC1-E 104		Grundlagen der Raumwis-senschaften LB Reidelstürz /Zink E 101	
4	AWP			Mikrorechnertechnik Vorlesung Ku ITC1-E 104	AWP	Mobile Betriebssysteme Do ITC2-Geoinformatik lab.	~~Vertiefte Elektrotechnik Praktikum~~ ~~Ku~~ ~~ITC1-E 103~~	Grundlagen der Raumwis-senschaften LB Reidelstürz / Zink E 101	
5	AWP			Mikrorechnertechnik Praktikum Ku ITC1-E 104	AWP	Mobile Betriebssysteme Do ITC2-Geoinformatik lab.			

Attention: No Lecture on the 12.11.2013!

Block 1: 08:00 - 09:30
Block 2: 09:45 - 11:15
Block 3: 12:00 - 13:30

3. Semester Bachelor AI (Stand: 12.09.2013)

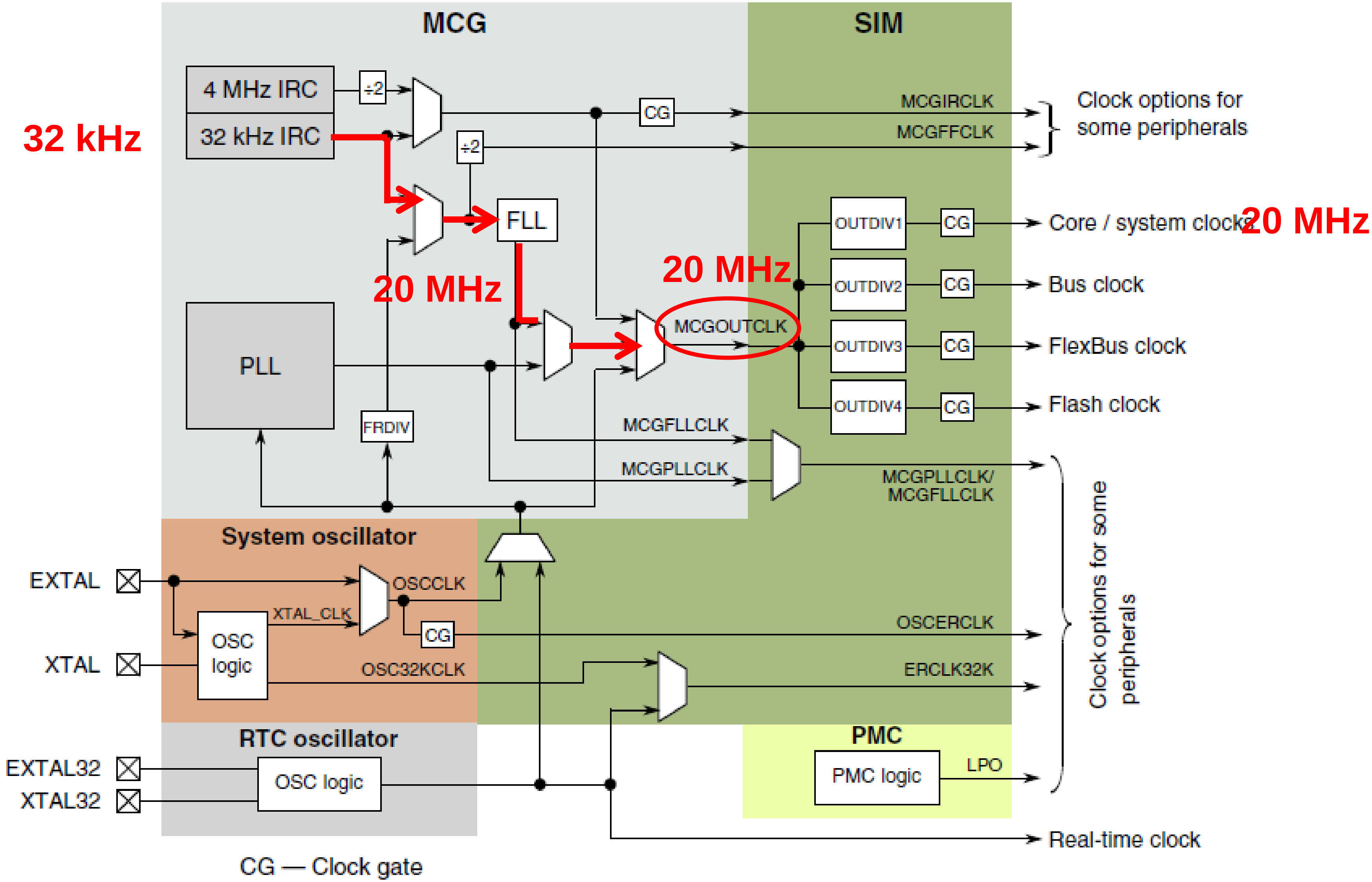
12.11.2013

14.11.2013

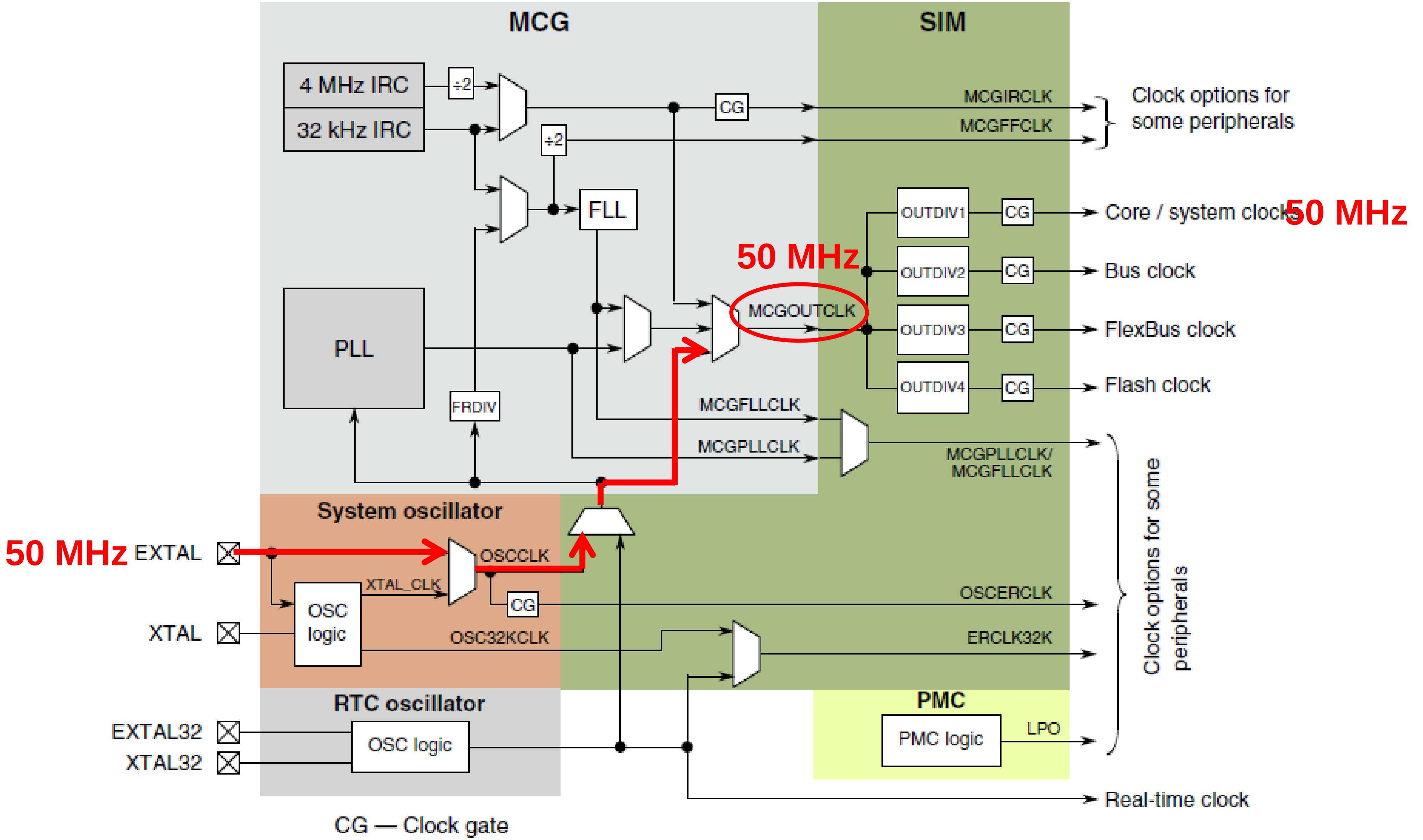
Block 4: 14:00 - 15:30
Block 5: 15:45 - 17:15
Block 6: 17:30 - 19:00

	Montag		Dienstag		Mittwoch		Donnerstag		Freitag	
1		Vertiefte Elektrotechnik Bö D 113		Digitaltechnik Bö E 101	SW-Engineering gem. mit MT 5 Jr ITC1-E 104			Messtechnik (bis Mitte Nov) Wu E 101	Einführung GIS LB Zink E 101	
2		Messtechnik (ab Mitte Nov) Bö D 113		Digitaltechnik Praktikum Gruppe 1/2 (14-tägig) LabIng	SW-Engineering gem. mit MT 5 Jr ITC1-E 104		Bezugssysteme und Positionierung LB Reidelstürz ITC1-E 104	Messtechnik (bis Mitte Nov) Wu E 101	Einführung GIS LB Zink E 101	
3		Messtechnik (ab Mitte Nov) Bö A 210		Digitaltechnik Praktikum Gruppe 1/2 (14-tägig) LabIng	SW-Engineering gem. mit MT 5 Jr ITC1-E 103		Bezugssysteme und Positionierung LB Reidelstürz ITC1-E 104		Grundlagen der Raumwis-senschaften LB Reidelstürz /Zink E 101	
4	AWP			~~Mikrorechnertechnik Vorlesung Ku ITC1-E 104~~	AWP		Mobile Betriebssysteme Do ITC2-Geoinformatik lab.	Vertiefte Elektrotechnik Praktikum Ku ITC1-E 103	Grundlagen der Raumwis-senschaften LB Reidelstürz / Zink E 101	
5	AWP			~~Mikrorechnertechnik Praktikum Ku ITC1-E 104~~	AWP		Mobile Betriebssysteme Do ITC2-Geoinformatik lab.			

Recall: Clock Generation at the Kinetis K60 - 20 MHz



Recall: Clock Generation at the Kinetis K60 - 50 MHz



Clock Generation at the Kinetis K60 - 50 and 100 MHz

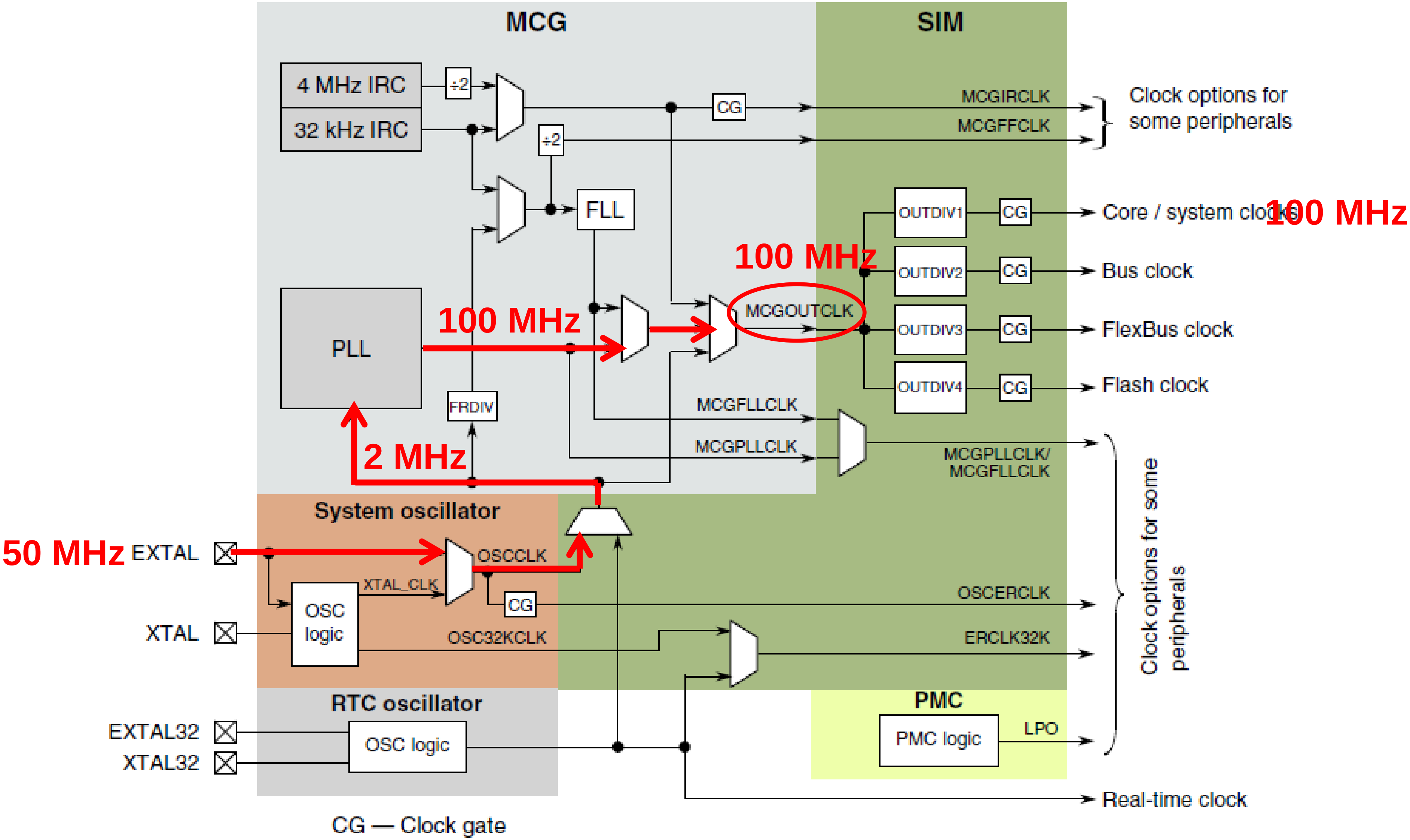
```
// Option 1: Clock 50 MHz
    MCG_C1 |= MCG_C1_CLKS(2); // use external clock


// Option 2: Clock 100 MHz
    MCG_C1 |= MCG_C1_CLKS(2); // use external clock
    MCG_C5 = MCG_C5_PRDIV(24);  // divide by 25
    MCG_C6 = MCG_C6_PLLS_MASK | 0x1A; // PLL factor 50
    MCG_C1 &= ~MCG_C1_CLKS_MASK; // switch to PLL out
```

24.3.1 MCG Control 1 Register (MCG_C1)

Address: MCG_C1 is 4006_4000h base + 0h offset = 4006_4000h

Bit	7	6	5	4	3	2	1	0
Read / Write	CLKS		FRDIV			IREFS	IRCLKEN	IREFSTEN
Reset	0	0	0	0	0	1	0	0

Clock Generation at the Kinetis K60 - 100 MHz



Interrupt

In systems programming, an interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing, the current thread.

The processor responds by suspending its current activities, saving its state, and executing a small program called an interrupt handler (or interrupt service routine, ISR) to deal with the event. This interruption is temporary, and after the interrupt handler finishes, the processor resumes execution of the previous thread.

Hardware interrupts were introduced as a way to reduce wasting the processor's valuable time in polling loops, waiting for external events. They may be implemented in hardware as a distinct system with control lines

Exception

Exception handling is the process of responding to the occurrence, during computation, of exceptions – anomalous or exceptional events requiring special processing – often changing the normal flow of program execution. It is provided by specialized programming language constructs or computer hardware mechanisms.

In general, an exception is handled (resolved) by saving the current state of execution in a predefined place and switching the execution to a specific subroutine known as an exception handler. If exceptions are continuable, the handler may later resume the execution at the original location using the saved information.

Generally, an exception in the superordinate concept of an interrupt. That means, an interrupt can be consideres as a special type of exception.

C language supports various means of error checking, but generally is not considered to support "exception handling.“

Interrupt Vector

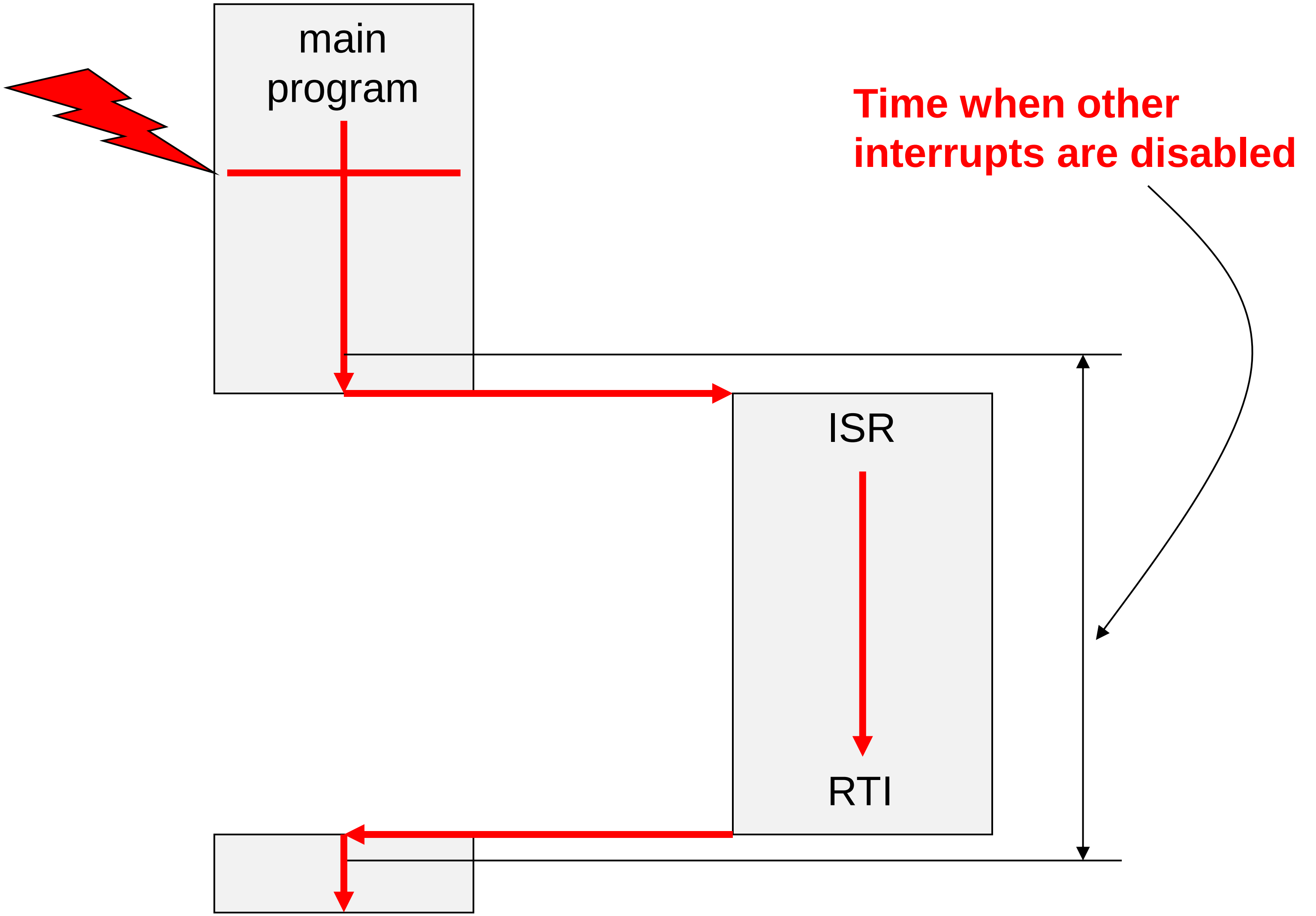
An interrupt vector is the memory address of an interrupt handler, or an index into an array called an interrupt vector table that contains the memory addresses of interrupt handlers.

When an interrupt is generated, the system saves its execution state and begins execution of the interrupt handler at the interrupt vector.

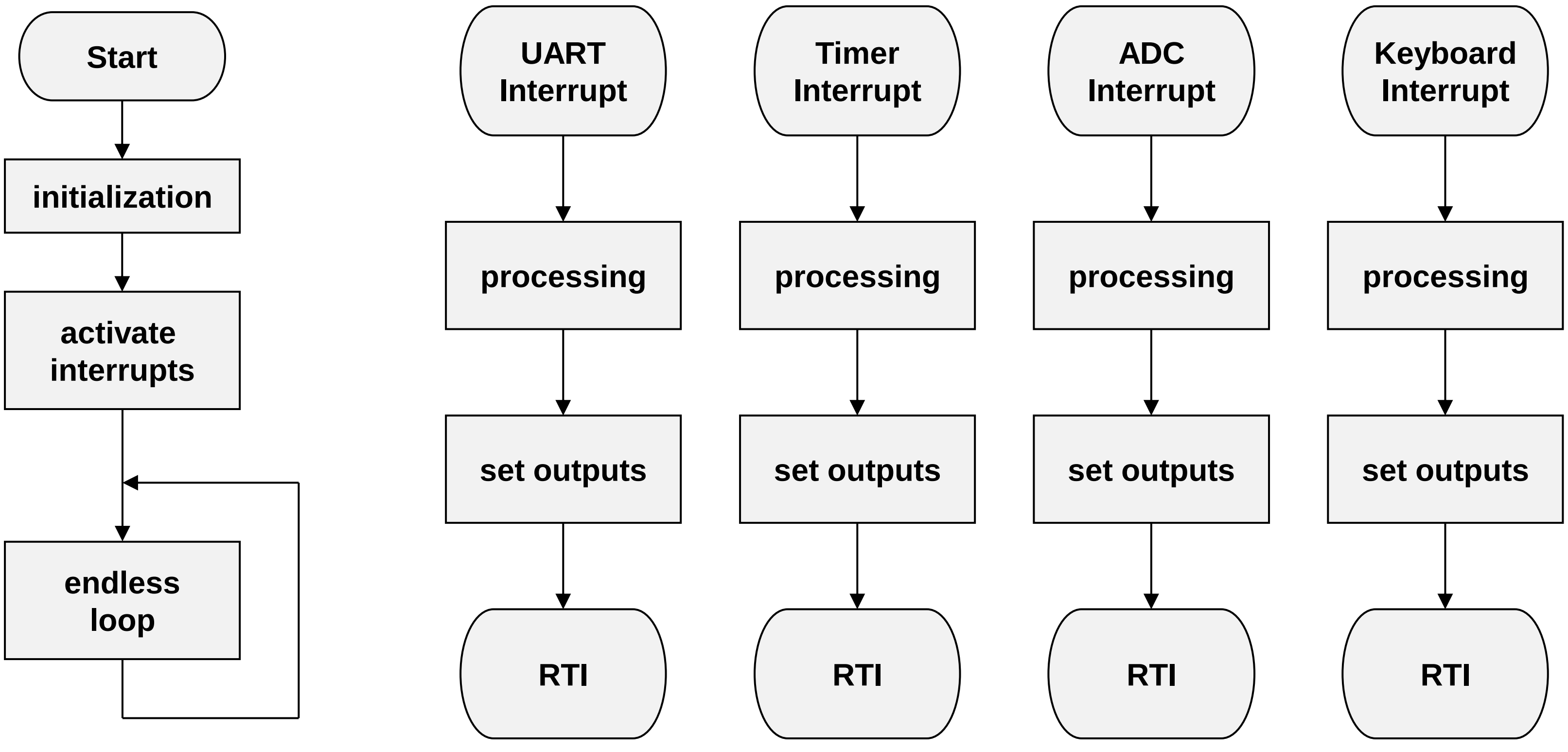
The interrupt vector table is defined **by the hardware** of the microcontroller, so it is necessary to look into the Reference Manual of the microcontroller to see the location of different vectors within the vector table.

The reset vector is a special type of interrupt vector, it is the default location a central processing unit will go to find the first instruction it will execute after a reset. The reset vector is a pointer or address, where the CPU should always begin as soon as it is able to execute instructions.

Sequence of an Interrupt



Program Setup with Interrupts



Short Interrupt Service Routine

The Interrupt Service Routine should be **as short as possible**!

The execution time of the Interrupt Service Routine has to be shorter than the time between the occurrence of the interrupts.

It you do not pay attention to this, interrupts can be „missed“. This can result in failures like:

- loss of data in serial communication
- loss of counting cycles of a timer
- loss of data in A/D converters

These failures are hard to find, since they appear spontaneously.

Long functions like printf(), scanf(), LCD outputs and similair should not be used in an Interrupt Service Routine.

Interrupt Vector Table of ARM Cortex M

vector table

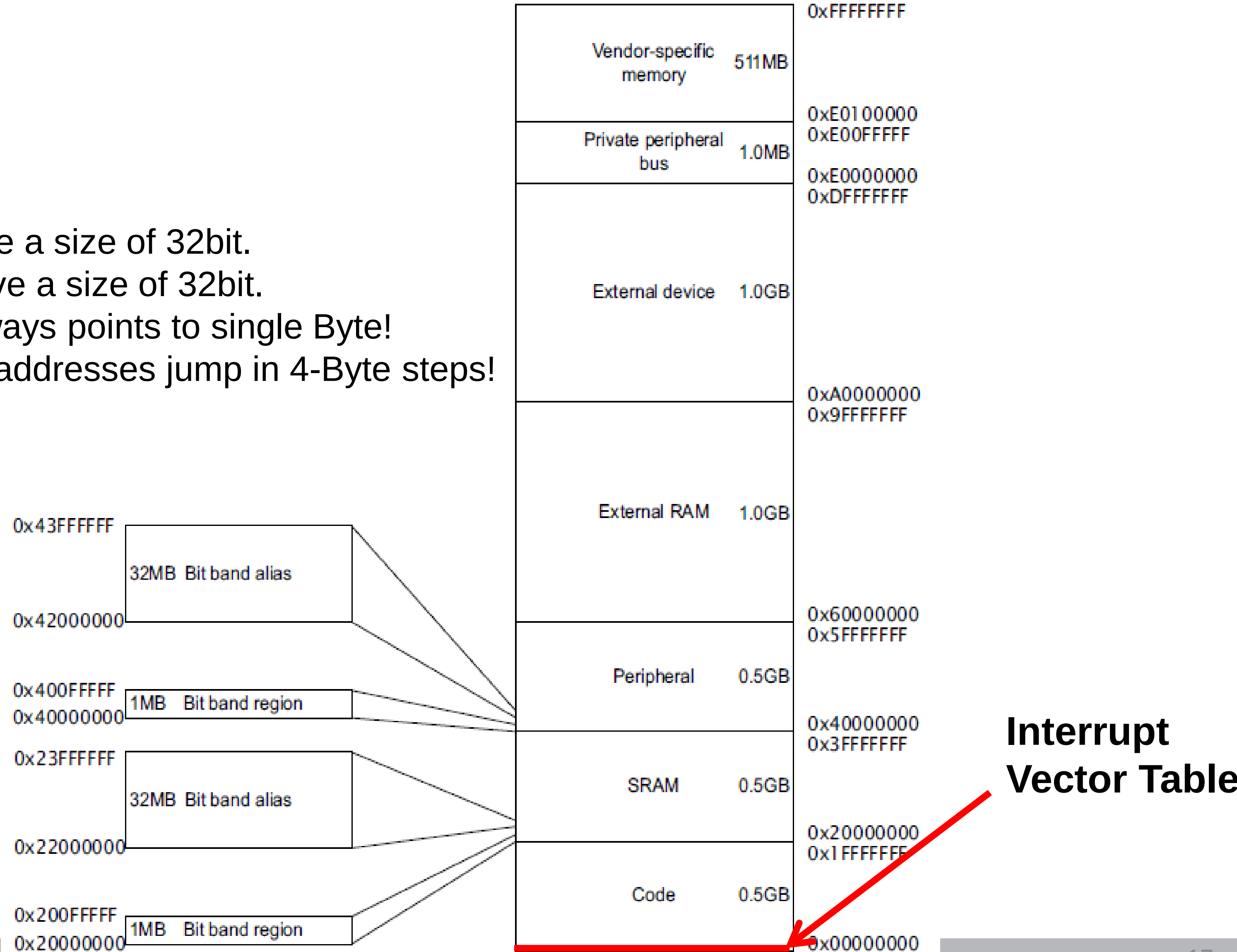
Exception Type	Address Offset	Exception Vector
18–255	0x48–0x3FF	IRQ #2–239
17	0x44	IRQ #1
16	0x40	IRQ #0
15	0x3C	SYSTICK
14	0x38	PendSV
13	0x34	Reserved
12	0x30	Debug Monitor
11	0x2C	SVC
7–10	0x1C–0x28	Reserved
6	0x18	Usage fault
5	0x14	Bus fault
4	0x10	MemManage fault
3	0x0C	Hard fault
2	0x08	NMI
1	0x04	Reset
0	0x00	Starting value of the MSP

up to 104 - 239 external interrupt sources

16 core internal interrupt sources (core exceptions)

Recall: Address Space of ARM Cortex-M3/M4 in general

Please note:
Addresses have a size of 32bit.
Data words have a size of 32bit.
An address always points to single Byte!
Therefore, the addresses jump in 4-Byte steps!

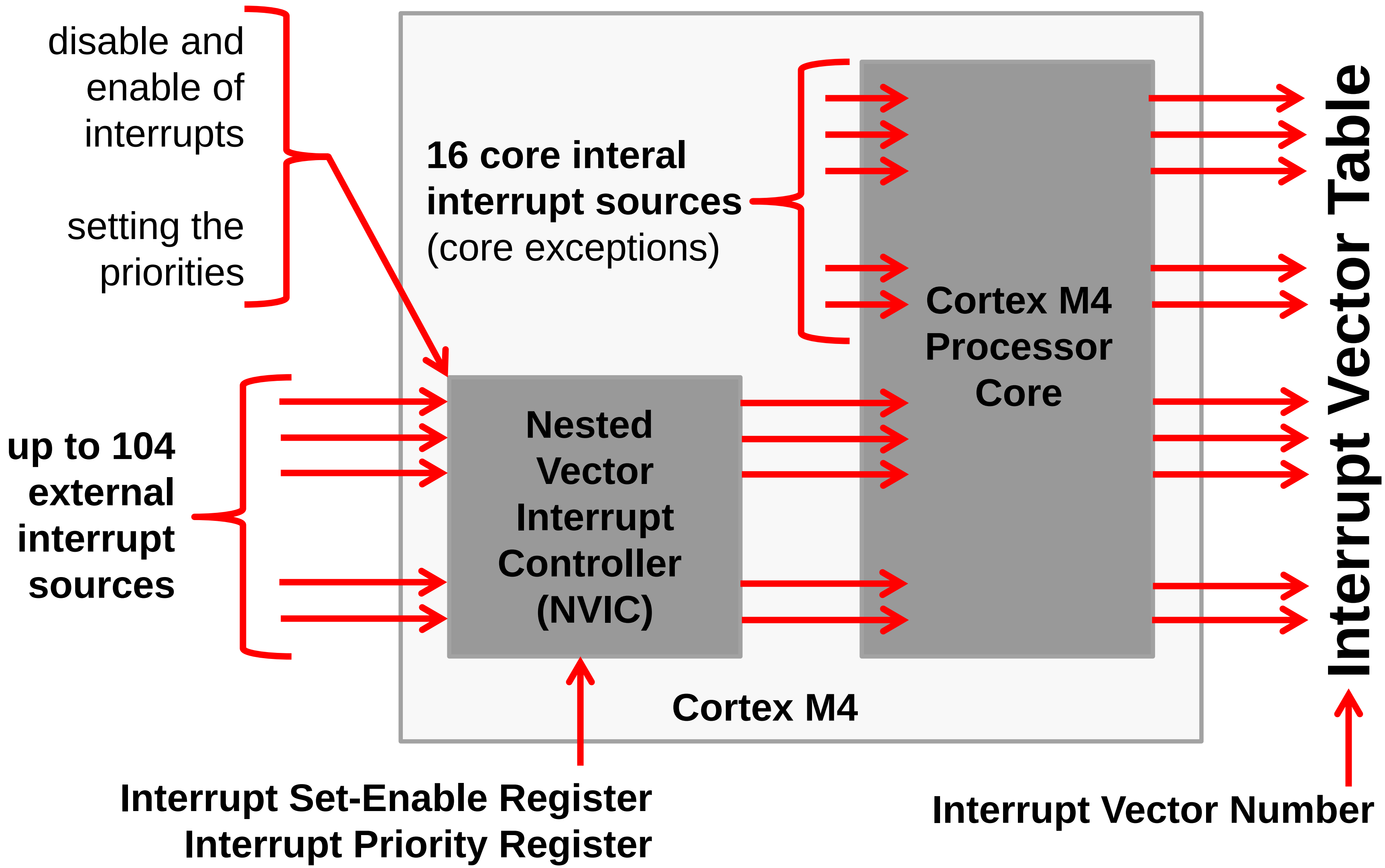


Exceptions of the ARM Cortex M Core

Exception Number	Exception Type	Priority	Function
1	Reset	−3 (Highest)	Reset
2	NMI	−2	Nonmaskable interrupt
3	Hard fault	−1	All classes of fault, when the corresponding fault handler cannot be activated because it is currently disabled or masked by exception masking
4	MemManage	Settable	Memory management fault; caused by MPU violation or invalid accesses (such as an instruction fetch from a nonexecutable region)
5	BusFault	Settable	Error response received from the bus system; caused by an instruction prefetch abort or data access error
6	Usage fault	Settable	Usage fault; typical causes are invalid instructions or invalid state transition attempts (such as trying to switch to ARM state in the Cortex-M3)
7–10	–	–	Reserved
11	SVC	Settable	System service call via SVC instruction
12	Debug monitor	Settable	Debug monitor
13	—	—	Reserved
14	PendSV	Settable	Pendable request for System Service
15	SYSTICK	Settable	System Tick Timer
16–255	IRQ	Settable	IRQ input #0–239

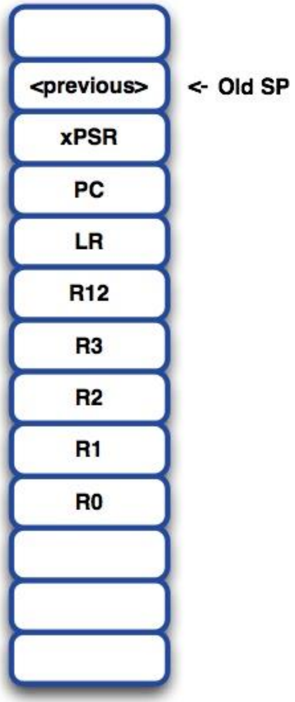
core exceptions

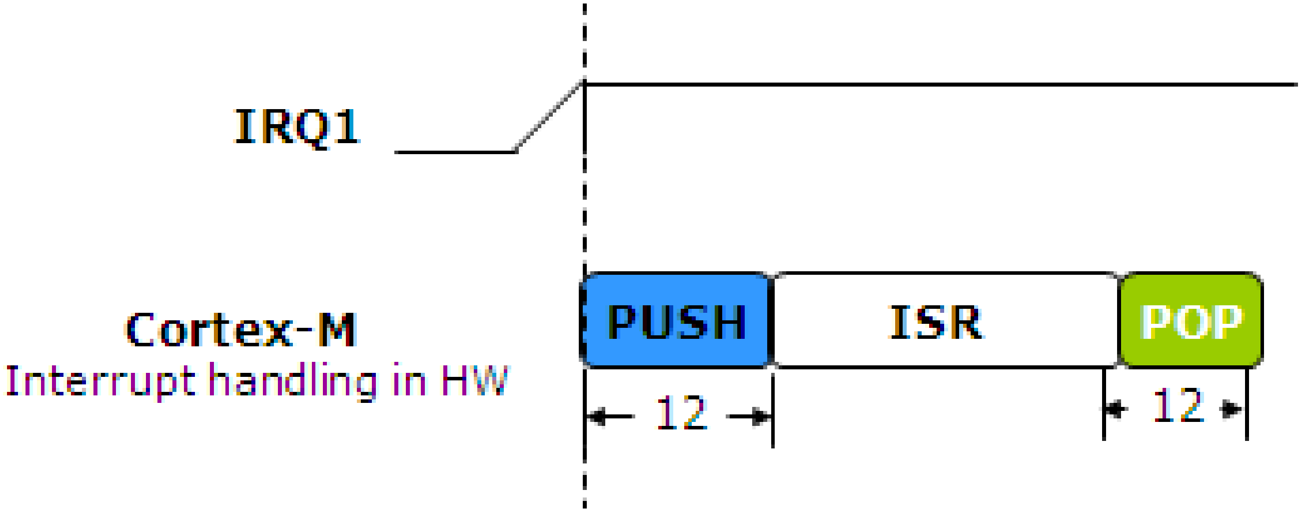
Nested Vector Interrupt Controller (NVIC)



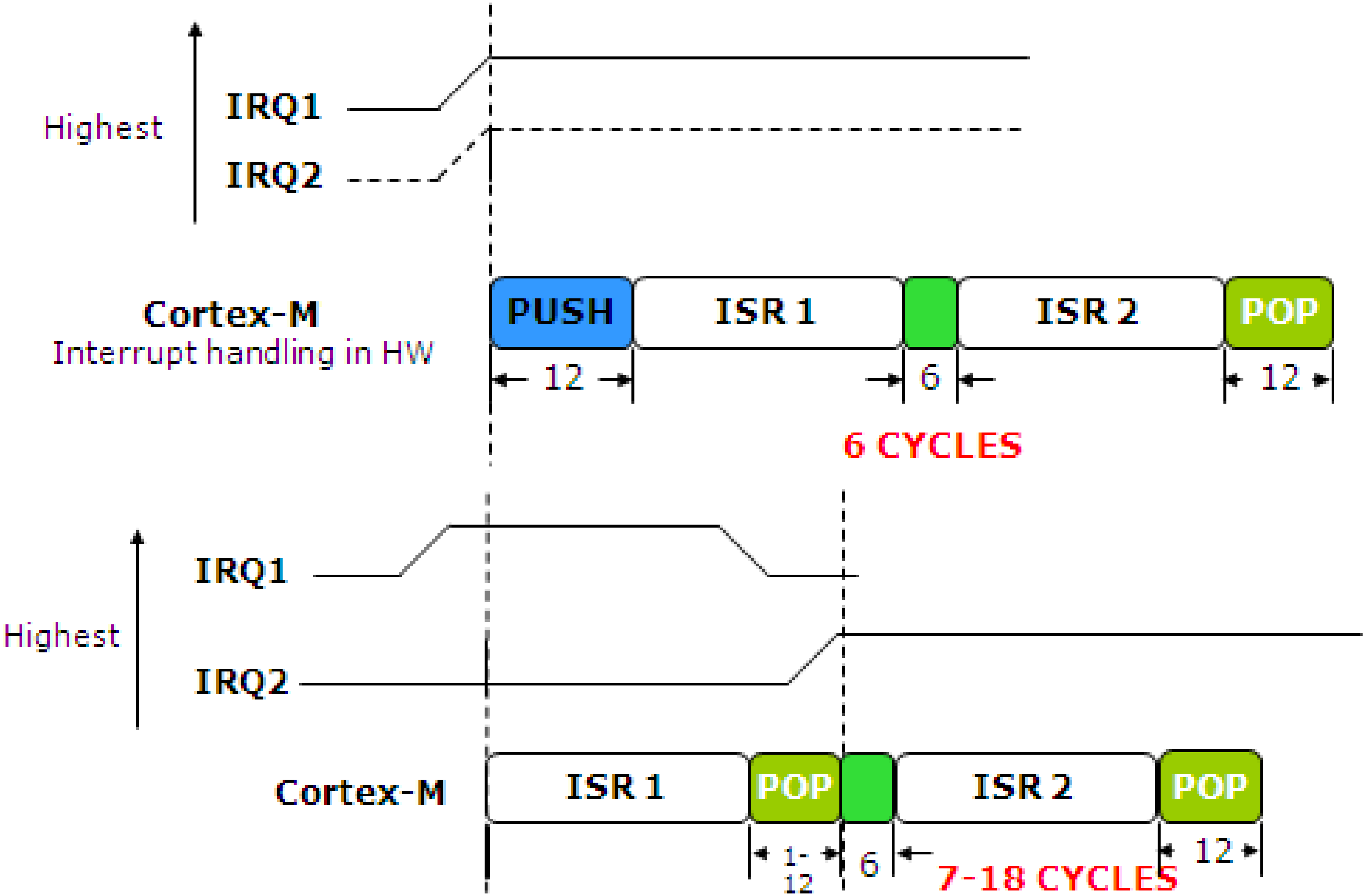
NVIC Operation Exception Entry And Exit

When an interrupt is raised by a peripheral, the NVIC will start the Cortex CPU serving the interrupt. As the Cortex CPU enters its interrupt mode, it will push a set of registers onto the stack. Importantly this is done automatically, so there is no instruction overhead in the application code. While the stack frame is being saved, the starting address of the interrupt service routine is fetched on the instruction bus. Thus the time taken from the interrupt being raised to reaching the first instruction in the interrupt routine is just 12 cycles.

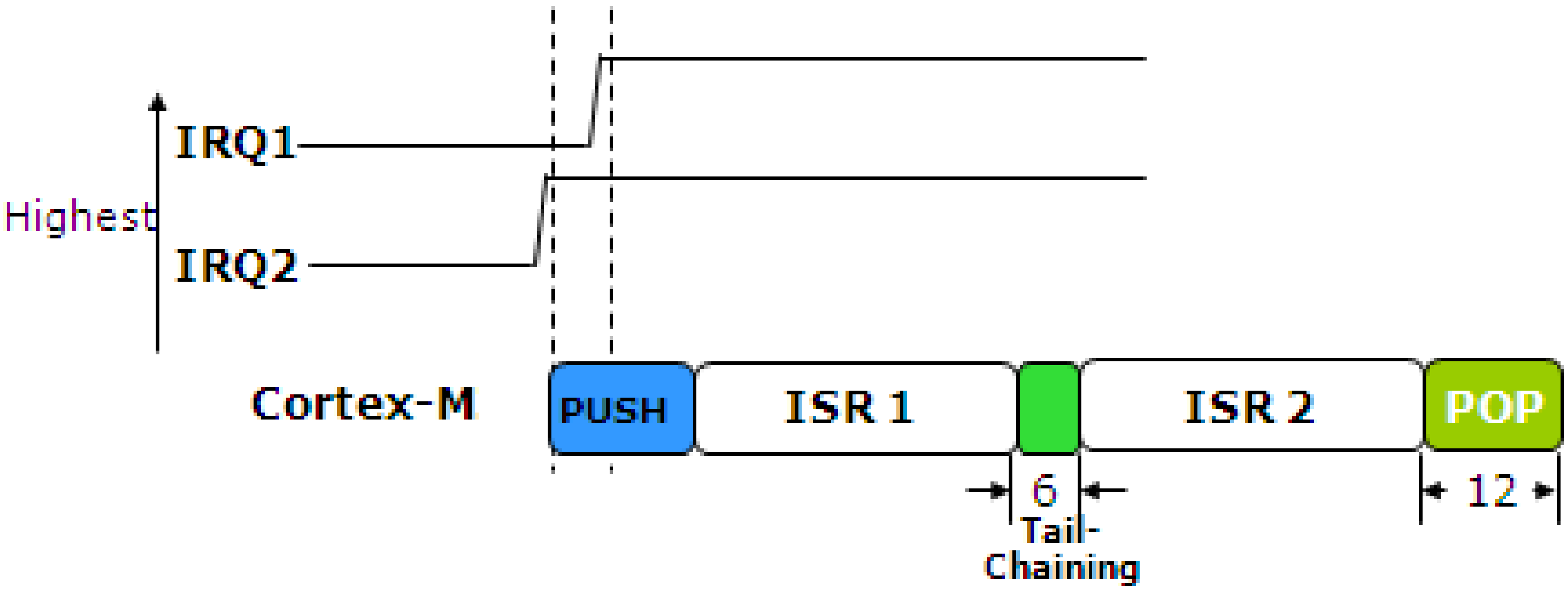




Tail Chaining



Late Arrival of High Priority Interrupt



In a real-time system there may often be a condition where we have started to serve a low priority interrupt, only for a high priority interrupt to be raised.
If this condition occurs during the initial PUSH the NVIC will switch to serve the higher priority interrupt. The stacking continues and there will be a minimum of 6 cycles from the point at which the high priority interrupt is raised, while the new ISR address is fetched.

Cortex Microcontroller Exception Vector Table

No.	Exception Type	Priority	Type of Priority	Descriptions
1	Reset	-3 (Highest)	fixed	Reset
2	NMI	-2	fixed	Non-Maskable Interrupt
3	Hard Fault	-1	fixed	Default fault if other hander not implemented
4	MemManage Fault	0	settable	MPU violation or access to illegal locations
5	Bus Fault	1	settable	Fault if AHB interface receives error
6	Usage Fault	2	settable	Exceptions due to program errors
7-10	Reserved	N.A.	N.A.	
11	SVCall	3	settable	System Service call
12	Debug Monitor	4	settable	Break points, watch points, external debug
13	Reserved	N.A.	N.A.	
14	PendSV	5	settable	Pendable request for System Device
15	SYSTICK	6	settable	System Tick Timer
16	Interrupt #0	7	settable	External Interrupt #0
......	....	....	settable	....
256	Interrupt#240	247	settable	External Interrupt #240

Reset Vector

The reset vector is a special type of interrupt vector, it is the default location a central processing unit will go to find the first instruction it will execute after a reset. The reset vector is a pointer or address, where the CPU should always begin as soon as it is able to execute instructions.

Since directly after the Reset there is no content of RAM, the Reset Vector is always located in nonvolatile memory (ROM).

At ARM Cortex-M the Reset Vector is located at address 0 (0x0000_0000).

“When the internal Reset is removed, the processor begins executing at address 0, which is initially the Reset vector mapped from the Boot Block.”

“On reset, the processor loads the MSP with the value from address 0x0000_0000.”

Reset

Power-on reset (POR)

When power is initially applied to the MCU or when the supply voltage drops below The power-on reset re-arm voltage level (VPOR), the POR circuit causes a POR Reset condition.
Resetting the MCU provides a way to start processing from a known set of initial conditions. System reset begins with the on-chip regulator in full regulation and System clocking generation from an internal reference. When the processor exits reset, it performs the following:

- Reads the start SP (SP_main) from vector-table offset 0
- Reads the start PC from vector-table offset 4
- LR is set to 0xFFFF_FFFF

External pin reset (PIN)

On this device, /RESET is a dedicated pin. This pin is open drain and has an internal pullup device. Asserting /RESET wakes the device from any mode.

Other Possible Scource of Reset

Low-voltage detect (LVD) reset

Computer operating properly (COP) watchdog reset

Low leakage wakeup (LLWU) reset

Multipurpose clock generator loss-of-clock (LOC) reset

Software reset (SW)

Lockup reset (LOCKUP)

EzPort reset

MDM-AP system reset request (via JTAG)

Debug resets

NMI - Non-Maskable Interrupt

Driving the /NMI signal low forces a non-maskable interrupt, if the /NMI function is selected on the corresponding pin.

144 LQFP	144 MAP BGA	Pin Name	Default	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7	EzPort
54	L7	PTA4/ LLWU_P3	NMI_b/ EZP_CS_b	TSI0_CH5	PTA4/ LLWU_P3		FTM0_CH1				NMI_b	EZP_CS_b

The **Miscellaneous Control Module** (MCM) generates two interrupt requests:
- Non-maskable interrupt (NMI)
- Normal interrupt

The MCM's non-maskable interrupt (NMI) is generated, if:
- MCM_ISCR[ETBN] is set, which is caused by
- The Embedded Trace Buffer counter is enabled (MCM_ETBCC[CNTEN] = 1),
- The Embedded Trace Buffer count expires, and
- The response to counter expiration is an NMI (MCM_ETBCC[RSPT] = 10)

MCM Interrupt Status Register (MCM_ISR)

16.2.4 Interrupt status register (MCM_ISR)

Address: MCM_ISR is E008_0000h base + 10h offset = E008_0010h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0												0	NMI	IRQ	0
W														w1c	w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Field	Description
2 NMI	Non-maskable interrupt pending If ETBCC[RSPT] is set to 10b, this bit is set when the ETB counter expires. 0 No pending NMI 1 Due to the ETB counter expiring, an NMI is pending
1 IRQ	Normal interrupt pending If ETBCC[RSPT] is set to 01b, this bit is set when the ETB counter expires. 0 No pending interrupt 1 Due to the ETB counter expiring, a normal interrupt is pending
0 Reserved	This read-only field is reserved and always has the value zero.

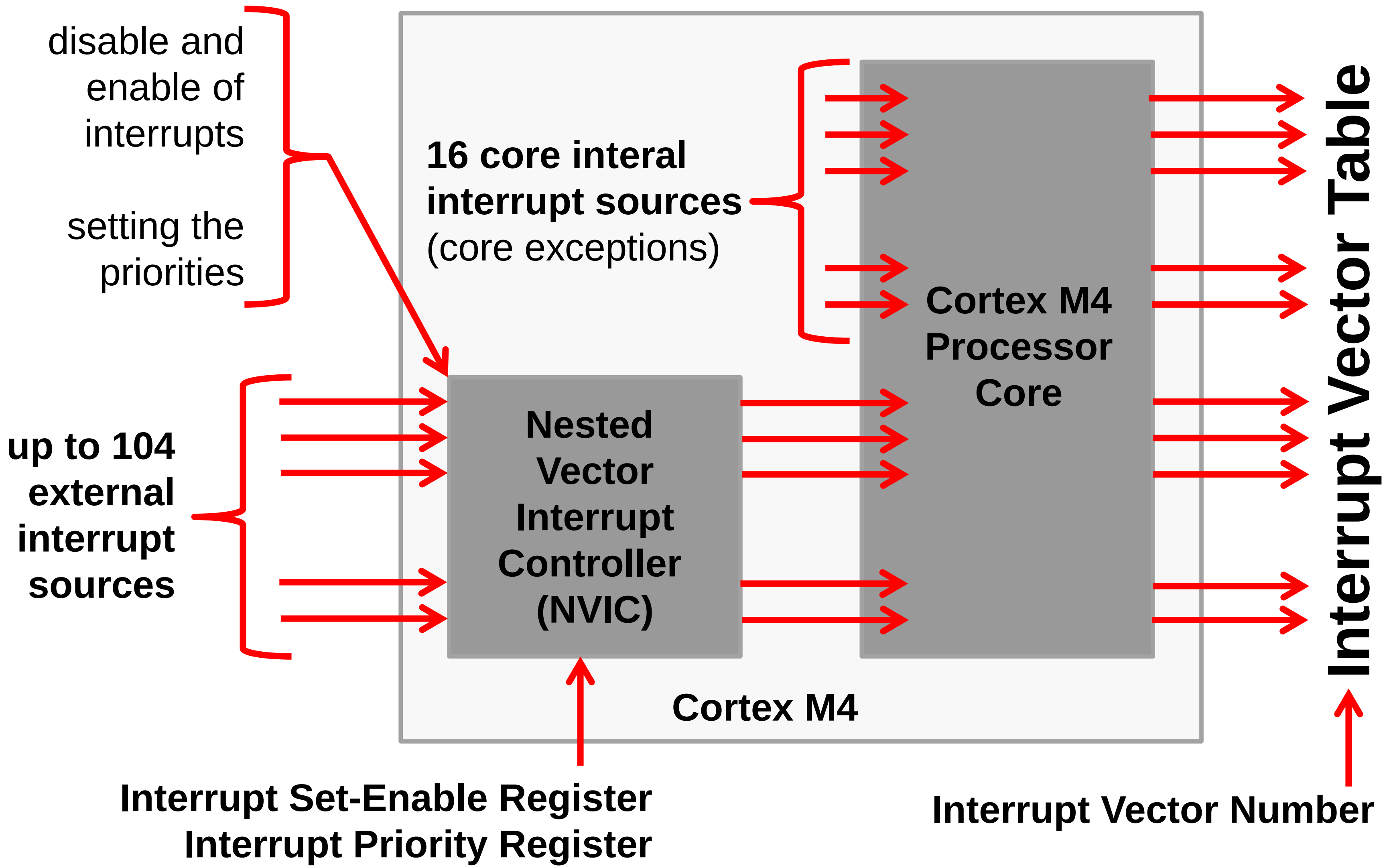
Nested Vector Interrupt Controller (NVIC)

The NVIC is a standard module on the ARM Cortex M series. This module is closely integrated with the core and provides a very low latency for entering an interrupt service routine ISR (12 cycles) and exiting an ISR (12 cycles).

The NVIC in Kinetis K60 provides 16 different interrupt priorities. **Priority 0 is the highest and the lowest is 15.** This can be used to control which interrupt must be serviced. For example, on a motor-control application if a UART and a timer interrupt occur at the same time, serving the timer interrupt that moves the motor is more critical than the UART interrupt that just received a character. In this case, the timer priority must be set higher than the UART.

As its name implies, the NVIC is designed to support **nested** interrupts and on the Kinetis K60 there are 16 levels of priority. The NVIC interrupt structure is designed to be programmed entirely in ‘C’ and does not need any Assembler macros or special non-ANSI directives.

Nested Vector Interrupt Controller (NVIC)



Interrupt Vector Table of Kinetis K60 (Part)

Address	Vector	IRQ[1]	NVIC non-IPR register number [2]	NVIC IPR register number [3]	Source module	Source description
ARM Core System Handler Vectors						
0x0000_0000	0	–	–	–	ARM core	Initial Stack Pointer
0x0000_0004	1	–	–	–	ARM core	Initial Program Counter
0x0000_0008	2	–	–	–	ARM core	Non-maskable Interrupt (NMI)
0x0000_000C	3	–	–	–	ARM core	Hard Fault
0x0000_0010	4	–	–	–	ARM core	MemManage Fault
0x0000_0014	5	–	–	–	ARM core	Bus Fault
0x0000_0018	6	–	–	–	ARM core	Usage Fault
0x0000_001C	7	–	–	–	—	—
0x0000_0020	8	–	–	–	—	—
0x0000_0024	9	–	–	–	—	—
0x0000_0028	10	–	–	–	—	—
0x0000_002C	11	–	–	–	ARM core	Supervisor call (SVCall)
0x0000_0030	12	–	–	–	ARM core	Debug Monitor
0x0000_0034	13	–	–	–	—	—
0x0000_0038	14	–	–	–	ARM core	Pendable request for system service (PendableSrvReq)
0x0000_003C	15	–	–	–	ARM core	System tick timer (SysTick)

Interrupt Vector Table of Kinetis K60 (Part)

Address	Vector	IRQ[1]	NVIC non-IPR register number [2]	NVIC IPR register number [3]	Source module	Source description
Non-Core Vectors						
0x0000_0040	16	0	0	0	DMA	DMA channel 0 transfer complete
0x0000_0044	17	1	0	0	DMA	DMA channel 1 transfer complete
0x0000_0048	18	2	0	0	DMA	DMA channel 2 transfer complete
0x0000_004C	19	3	0	0	DMA	DMA channel 3 transfer complete
0x0000_0050	20	4	0	1	DMA	DMA channel 4 transfer complete
0x0000_0084	33	17	0	4	MCM	Normal interrupt
0x0000_0088	34	18	0	4	Flash memory	Command complete
0x0000_008C	35	19	0	4	Flash memory	Read collision
0x0000_0090	36	20	0	5	Mode Controller	Low-voltage detect, low-voltage warning
0x0000_0094	37	21	0	5	LLWU	Low Leakage Wakeup **NOTE:** The LLWU interrupt must not be masked by the interrupt controller to avoid a scenario where the system does not fully exit stop mode on an LLS recovery.
0x0000_0098	38	22	0	5	WDOG	Watchdog interrupt
0x0000_009C	39	23	0	5	RNG	Randon Number Generator

Interrupt Vector Table of Kinetis K60 (Part)

Address	Vector	IRQ[1]	NVIC non-IPR register number [2]	NVIC IPR register number [3]	Source module	Source description
0x0000_00C4	49	33	1	8	CAN0	Receive Warning
0x0000_00C8	50	34	1	8	CAN0	Wake Up
0x0000_00CC	51	35	1	8	—	—
0x0000_00D0	52	36	1	9	—	—
0x0000_00D4	53	37	1	9	CAN1	OR'ed Message buffer (0-15)
0x0000_00D8	54	38	1	9	CAN1	Bus off
0x0000_00DC	55	39	1	9	CAN1	Error
0x0000_00E0	56	40	1	10	CAN1	Transmit Warning
0x0000_00E4	57	41	1	10	CAN1	Receive Warning
0x0000_00E8	58	42	1	10	CAN1	Wake Up
0x0000_00F4	61	45	1	11	UART0	Single interrupt vector for UART status sources
0x0000_00F8	62	46	1	11	UART0	Single interrupt vector for UART error sources
0x0000_00FC	63	47	1	11	UART1	Single interrupt vector for UART status sources
0x0000_0100	64	48	1	12	UART1	Single interrupt vector for UART error sources

Interrupt Vector Table of Kinetis K60 (Part)

Address	Vector	IRQ[1]	NVIC non-IPR register number [2]	NVIC IPR register number [3]	Source module	Source description
0x0000_01A0	104	88	2	22	Port control module	Pin detect (Port B)
0x0000_01A4	105	89	2	22	Port control module	Pin detect (Port C)
0x0000_01A8	106	90	2	22	Port control module	Pin detect (Port D)
0x0000_01AC	107	91	2	22	Port control module	Pin detect (Port E)
0x0000_01B0	108	92	2	23	—	—
0x0000_01B4	109	93	2	23	—	—
0x0000_01B8	110	94	2	23	Software	Software interrupt[4]

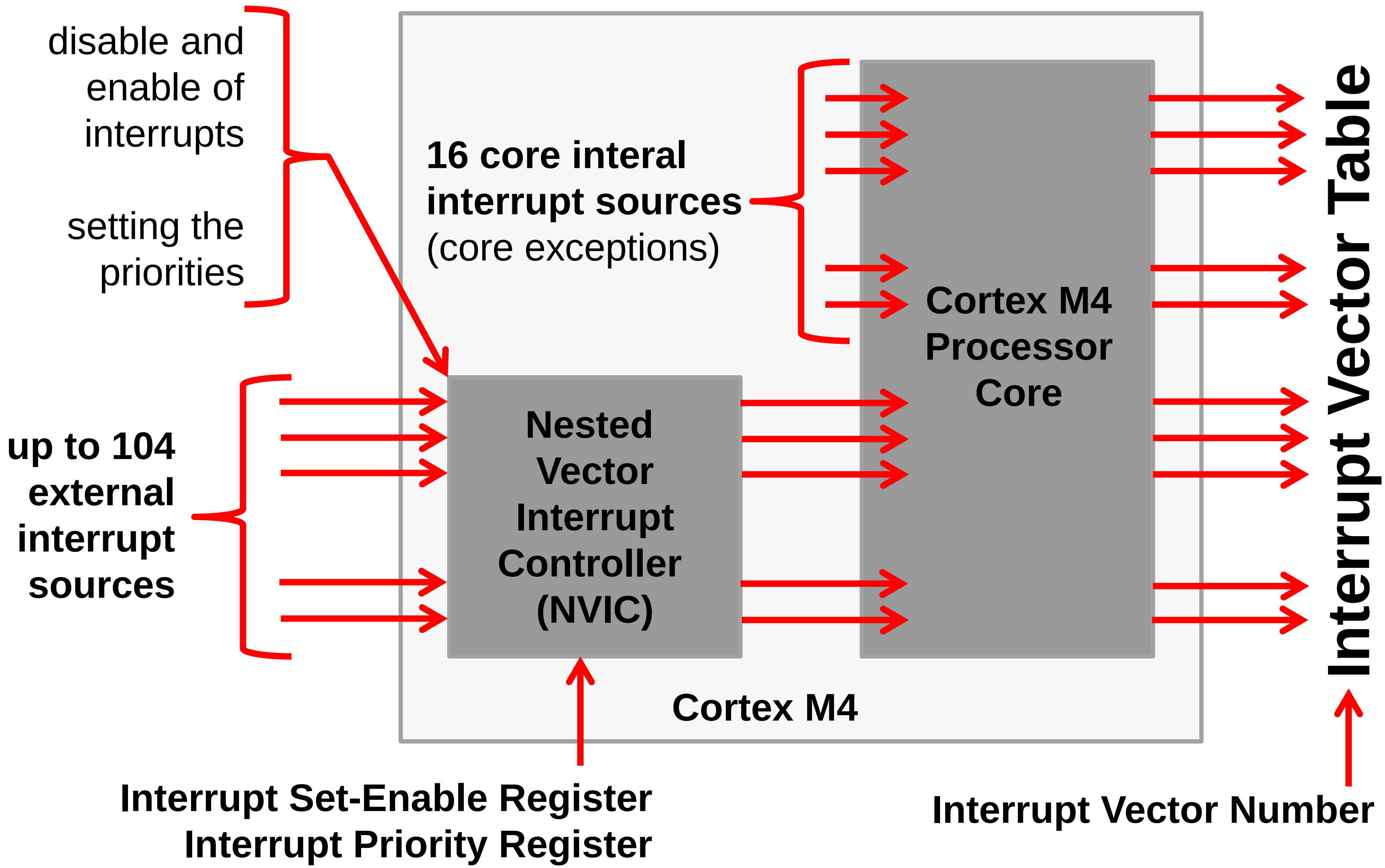
vector in the vector table

IRQ number in the interrupt controller

Interrupt Priority Register IPR

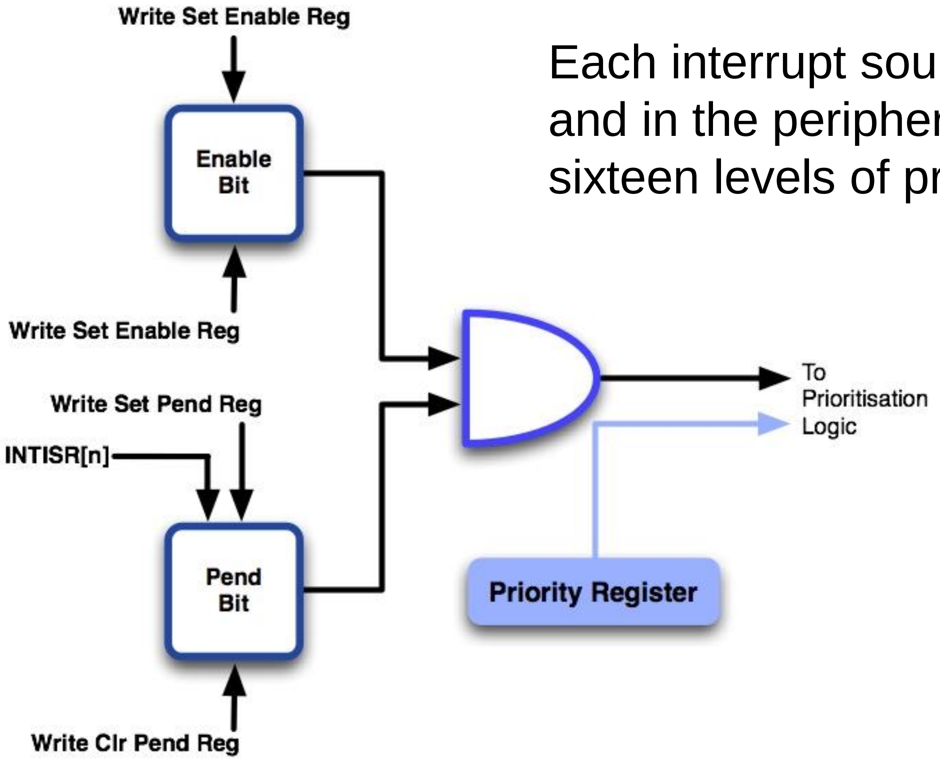
Interrupt Set-Enable Register
Interrupt Clear-Enable Register
Interrupt Set-Pending Register
Interrupt Clear-Pending Register
Active Bit Register

Nested Vector Interrupt Controller (NVIC)



IRQ register blocks

Each of the USER peripherals is controlled by the IRQ register blocks. Each user peripheral has an Interrupt Enable bit. These bits are located across two 32-bit IRQ Set Enable registers. There are matching IRQ Clear Enable registers that are used to disable an interrupt source. The NVIC also includes pending and active registers that allow you to determine the current condition of an interrupt source.



Each interrupt source has an enable bit in the NVIC and in the peripheral. In the Kinetis K60 there are sixteen levels of priority.

NVIC registers

Siehe Cortex™-M3 Technical Reference Manual Seite 8.1

Address	Name	Type	Reset	Description
0xE000E004	ICTR	RO	-	*Interrupt Controller Type Register, ICTR*
0xE000E100 - 0xE000E11C	NVIC_ISER0 - NVIC_ISER7	RW	0x00000000	Interrupt Set-Enable Registers
0xE000E180 - 0E000xE19C	NVIC_ICER0 - NVIC_ICER7	RW	0x00000000	Interrupt Clear-Enable Registers
0xE000E200 - 0xE000E21C	NVIC_ISPR0 - NVIC_ISPR7	RW	0x00000000	Interrupt Set-Pending Registers
0xE000E280 - 0xE000E29C	NVIC_ICPR0 - NVIC_ICPR7	RW	0x00000000	Interrupt Clear-Pending Registers
0xE000E300 - 0xE000E31C	NVIC_IABR0 - NVIC_IABR7	RO	0x00000000	Interrupt Active Bit Register
0xE000E400 - 0xE000E4EC	NVIC_IPR0 - NVIC_IPR59	RW	0x00000000	Interrupt Priority Register

Interrupt Controller Type Register ICTR

Read the Interrupt Controller Type Register to see the number of interrupt lines that the NVIC supports.

The register address, access type, and Reset state are:

Address `0xE000E004`

Access Read-only

Reset state Depends on the number of interrupts defined in this processor implementation.

Figure 8-1 shows the bit assignments of the Interrupt Controller Type Register.

31 … 5	4 … 0
Reserved	INTLINESNUM

Interrupt Set-Enable Registers NVICISER

Use the Interrupt Set-Enable Registers to:
- enable interrupts
- determine which interrupts are currently enabled.

Each bit in the register corresponds to one of 32 interrupts. Setting a bit in the Interrupt Set-Enable Register enables the corresponding interrupt. When the enable bit of a pending interrupt is set, the processor activates the interrupt based on its priority. When the enable bit is clear, asserting its interrupt signal pends the interrupt, but it is not possible to activate the interrupt, regardless of its priority.

Bits	Field	Function
[31:0]	SETENA	Interrupt set enable bits. For write operation: 1 = enable interrupt 0 = no effect. For read operation: 1 = enable interrupt 0 = disable interrupt Writing 0 to a SETENA bit has no effect. Reading the bit returns its current enable state. Reset clears the SETENA fields.

Interrupt Clear-Enable Registers NVICICER

Use the Interrupt Clear-Enable Registers to:
• disable interrupts
• determine which interrupts are currently disabled.

Each bit in the register corresponds to one of the 32 interrupts. Setting an Interrupt Clear-Enable Register bit disables the corresponding interrupt.

Bits	Field	Function
[31:0]	CLRENA	Interrupt clear-enable bits. For write operation: 1 = disable interrupt 0 = no effect. For read operation: 1 = enable interrupt 0 = disable interrupt. Writing 0 to a CLRENA bit has no effect. Reading the bit returns its current enable state.

Interrupt Set-Pending Registers NVICISPR

Use the Interrupt Set-Pending Registers to:
- force interrupts into the pending state
- determine which interrupts are currently pending.

Each bit in the register corresponds to one of the 32 interrupts. Setting an Interrupt Set-Pending Register bit pends the corresponding interrupt.
Clear an Interrupt Set-Pending Register bit by writing a 1 to the corresponding bit in the Interrupt Clear-Pending Register (see *Interrupt Clear-Pending Register*).
Clearing the Interrupt Set-Pending Register bit puts the interrupt into the non-pended state.

Bits	Field	Function
[31:0]	SETPEND	Interrupt set-pending bits: 1 = pend the corresponding interrupt 0 = corresponding interrupt not pending. Writing 0 to a SETPEND bit has no effect. Reading the bit returns its current state.

Interrupt Clear-Pending Registers NVICICPR

Use the Interrupt Clear-Pending Registers to:

- clear pending interrupts
- determine which interrupts are currently pending.

Each bit in the register corresponds to one of the 32 interrupts. Setting an Interrupt Clear-Pending Register bit puts the corresponding pending interrupt in the inactive state.

Bits	Field	Function
[31:0]	CLRPEND	Interrupt clear-pending bits: 1 = clear pending interrupt 0 = do not clear pending interrupt. Writing 0 to a CLRPEND bit has no effect. Reading the bit returns its current state.

Active Bit Registers NVICIABR

Read the Active Bit Register to determine which interrupts are active. Each flag in the register corresponds to one of the 32 interrupts.

Bits	Field	Function
[31:0]	ACTIVE	Interrupt active flags: 1 = interrupt active or pre-empted and stacked 0 = interrupt not active or stacked.

Interrupt Priority Registers NVICIPR

Use the Interrupt Priority Registers to assign a priority from 0 to 255 to each of the available interrupts. 0 is the highest priority, and 255 is the lowest.

The priority registers are stored with the implemented values first. This means that if there are four bits of priority, the priority value is stored in bits [7:4] of the byte.

However, if there are three bits of priority, the priority value is stored in bits [7:5] of the byte. This means that an application can work even if it does not know how many priorities are possible.

3.2.2.1 Interrupt priority levels

This device supports 16 priority levels for interrupts. Therefore, in the NVIC each source in the IPR registers contains 4 bits. For example, IPR0 is shown below:

	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
R	IRQ3				0	0	0	0	IRQ2				0	0	0	0	IRQ1				0	0	0	0	IRQ0				0	0	0	0
W																																

Steps to use an Interrupt

1. **Configure the Peripheral for Interrupt**

2. **Configuration of Interrupt Controller NVIC**
   - activate interrupts
   - set priorities if wanted

3. **Global Interrupt Enable / Disable**

4. **Register the Interrupt in the Vector Table**

5. **Declare Interrupt Service Routine**

1. Configure the Peripheral for Interrupt

11.4.1 Pin Control Register n (PORTx_PCR*n*)

For PCR1 to PCR5 of the port A, bit 0, 1, 6, 8, 9,10 reset to 1; for the PCR0 of the port A, bit 1, 6, 8, 9, 10 reset to 1; in other conditions, all bits reset to 0.

Addresses: 4004_9000h base + 0h offset + (4d × *n*), where *n* = 0d to 31d

Bit	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
R	0							ISF	0				IRQC				LK	0				MUX			0	DSE	ODE	PFE	0	SRE	PE	PS
W								w1c																								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

If the pin is an input:

Interrupt / DMA Request

0000 Interrupt/DMA Request disabled.
0001 DMA Request on rising edge.
0010 DMA Request on falling edge.
0011 DMA Request on either edge.
0100 Reserved.
1000 Interrupt when logic zero.
1001 Interrupt on rising edge.
1010 Interrupt on falling edge.
1011 Interrupt on either edge.
1100 Interrupt when logic one.
Others Reserved.

1. Example: Pin Interrupt

In this example, pin PTA4 is connected to a push button. An interrupt is generated when the button is pressed. A GPIO interrupt is used instead of an NMI interrupt because an edge-sensitive interrupt is preferred versus a level-sensitive interrupt.

This ensures that one interrupt will occur per button press. Interrupts need to be enabled in the ARM core, as described in the NVIC chapter.

```
/* Configure the PTA4 pin for its GPIO function */
PORTA_PCR4 = PORT_PCR_MUX(0x1); // GPIO is alt1 for this pin

/* Configure the PTA4 pin for rising edge interrupts */
PORTA_PCR4 |= PORT_PCR_IRQC(0x9);

/* Initialize the NVIC to enable the specified IRQ */
enable_irq(87);
```

1. Example: FlexTimer Overflow Interrupt

FlexTimer Status and Control Register (FTMx_SC)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0								TOF	TOIE	CPWMS	CLKS		PS		
W									0							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Timer Overflow Interrupt Enable
Enables FTM overflow interrupts.
0 Disable TOF interrupts.
1 Enable TOF interrupts.

```
// FlexTimers Enable Flextimer 0 Overflow Interrupt
FTM0_SC |= FTM_SC_TOIE_MASK;
```

2. Example Configuration of NVIC (1)

Suppose you need to configure the low-power timer (LPTMR) interrupt.

Table 3-5. LPTMR interrupt vector assignment

Address	Vector	IRQ[1]	NVIC non-IPR register number [2]	NVIC IPR register number [3]	Source module	Source description
0x0000_0194	101	85	2	21	Low Power Timer	—

1. Indicates the NVIC's interrupt source number.
2. Indicates the NVIC's ISER, ICER, ISPR, ICPR, and IABR register number used for this IRQ. The equation to calculate this value is: IRQ div 32
3. Indicates the NVIC's IPR register number used for this IRQ. The equation to calculate this value is: IRQ div 4

The NVIC registers you would use to configure the interrupt are:

- NVICISER2
- NVICICER2
- NVICISPR2
- NVICICPR2
- NVICIABR2
- NVICIPR21

Interrupt Set-Enable Register 2
Interrupt Clear-Enable Register 2
Interrupt Set-Pending Register 2
Interrupt Clear-Pending Register 2
Active Bit Register 2
Interrupt Priority Register 21

2. Example Configuration of NVIC (2)

To determine the particular IRQ's bitfield location within these particular registers:

- **NVICISER2, NVICICER2, NVICISPR2, NVICICPR2, NVICIABR2** bit location = IRQ mod 32 = **21**

- **NVICIPR21** bitfield starting location = 8 * (IRQ mod 4) + 4 = 12
  Since the NVICIPR bitfields are 4-bit wide (16 priority levels),
  the NVICIPR21 bitfield range is **12-15**

Therefore, the following bitfield locations are used to configure the LPTMR interrupts:

- NVICISER2[21]
- NVICICER2[21]
- NVICISPR2[21]
- NVICICPR2[21]
- NVICIABR2[21]
- NVICIPR21[15:12]

2. Example Configuration of NVIC (3)

At this point, the interrupt for the LPTMR can be configured:

```
NVICICPR2|=(1<<21); //Clear any pending interrupts on LPTMR
NVICISER2|=(1<<21); //Enable interrupts from LPTMR module
```

Next, set the interrupt priority level. This is application dependent. On Kinetis MCUs there are 16 different priority levels. To set the priority, write to the **NVICIPxx** register, the “xx” represents the IRQ number, in this example, NVICIP85. Note the most significant nibble is used to set-up the priority, the lower nibble is reserved and reads as zero.
The LPTMR example sets the priority to 3:

```
NVICIP85 = 0x30; //Set Priority 3 to the LPTMR module
```

Funktion: void enable_irq (int irq)

```
void enable_irq (int irq)
{
    int div;
    // Make sure that the IRQ is an allowable number. Right now up to 91 is used.
    if (irq > 91)
        printf("\nERR! Invalid IRQ value passed to enable irq function!\n");

    /* Determine which of the NVICISERs corresponds to the irq */
    div = irq/32;

    switch (div)
    {
    case 0x0:
              NVICICPR0 |= 1 << (irq%32);
              NVICISER0 |= 1 << (irq%32);
              break;
    case 0x1:
              NVICICPR1 |= 1 << (irq%32);
              NVICISER1 |= 1 << (irq%32);
              break;
    case 0x2:
              NVICICPR2 |= 1 << (irq%32);
              NVICISER2 |= 1 << (irq%32);
              break;
    }
}
```

3. Global Interrupt Enable / Disable

```
#define EnableInterrupts asm ("CPSIE i");
```

During the initial reset, NVIC is turned off. Therefore, the processor cannot receive any interrupts (except for NMI, Reset interrupt, and hard fault). To turn on the interrupts with configurable priority:

```
asm volatile ("cpsie i");
```

“CPSIE I” is a assembly instruction to enable the priority configurable interrupts. Actually, it’s a shortcut to this longer procedure

```
asm volatile ("MOVS r0, #0\n\
MSR PRIMASK, r0");
```

To turn off the priority configurable interrupts:

```
asm volatile ("cpsid i");
```

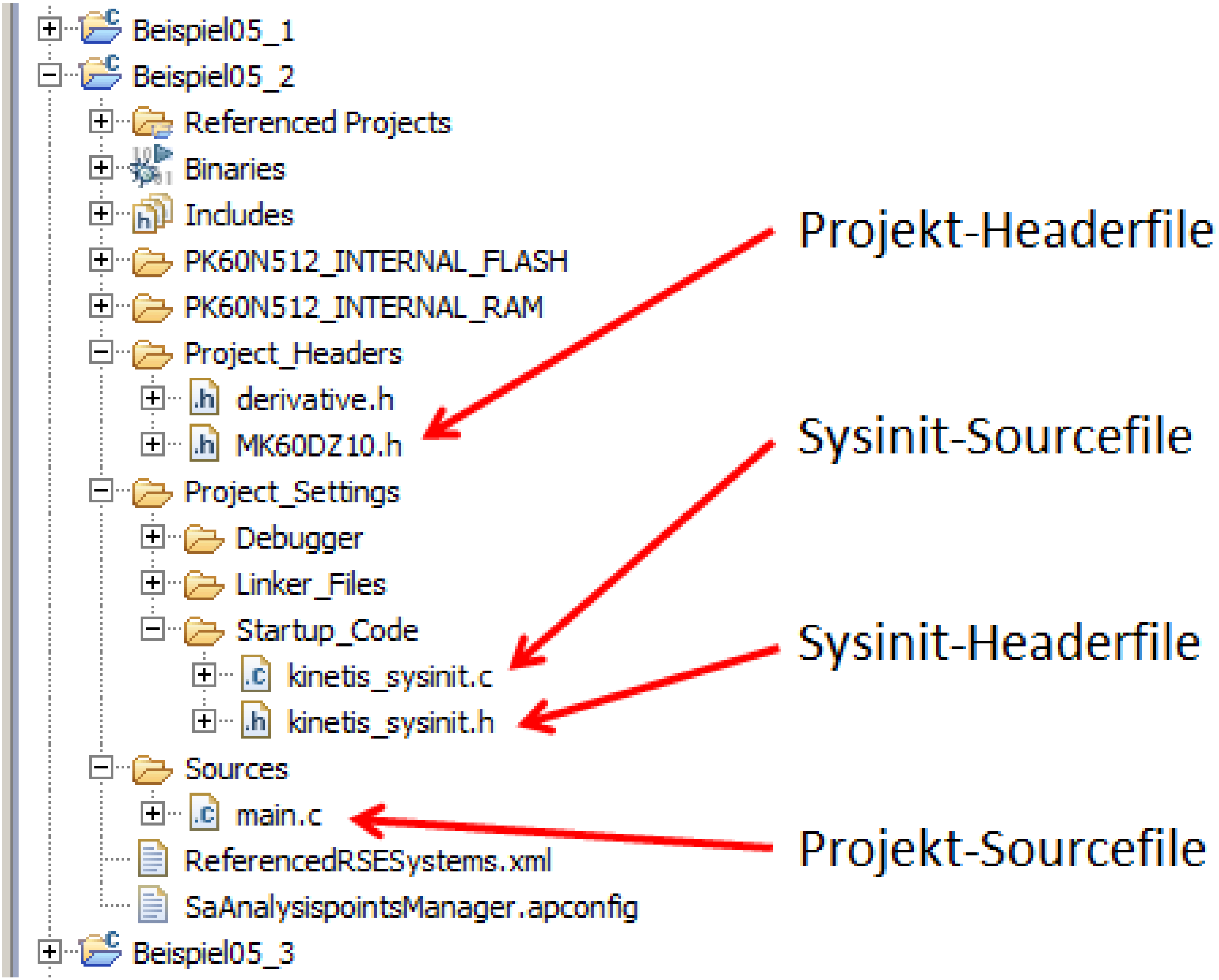
4. Interrupt Vector Table (1)

```
/*
 *    kinetis_sysinit.c - Default init routines for
 *                        Kinetis ARM systems
 *    Copyright © 2010 Freescale semiConductor Inc.
 */

#include "kinetis_sysinit.h"
#include "derivative.h"

typedef void (*const tIsrFunc)(void);

typedef struct {
  uint32_t * __ptr;
  tIsrFunc __fun[119];
} tVectorTable;

extern uint32_t __vector_table[];

...
```

4. Interrupt Vector Table (2)

```
#pragma define_section vectortable ".vectortable" ".vectortable"
static __declspec(vectortable) tVectorTable __vect_table = { /* I
   __SP_INIT,                    /* 0 (0x00000000) (prior: -) */
  {
   (tIsrFunc)__thumb_startup,    /* 1 (0x00000004) (prior: -) */
   (tIsrFunc)isrINT_NMI,         /* 2 (0x00000008) (prior: -2) */
   (tIsrFunc)UNASSIGNED_ISR,     /* 3 (0x0000000C) (prior: -1) */
   (tIsrFunc)UNASSIGNED_ISR,     /* 4 (0x00000010) (prior: -) */

....


   (tIsrFunc)UNASSIGNED_ISR,     /* 102 (0x00000198) (prior: -) */
   (tIsrFunc)porta_isr,          /* 103 (0x0000019C) (prior: -) */
   (tIsrFunc)UNASSIGNED_ISR,     /* 104 (0x000001A0) (prior: -) */
   (tIsrFunc)UNASSIGNED_ISR,     /* 105 (0x000001A4) (prior: -) */
   (tIsrFunc)UNASSIGNED_ISR,     /* 106 (0x000001A8) (prior: -) */
   (tIsrFunc)porte_isr,          /* 107 (0x000001AC) (prior: -) */
   (tIsrFunc)UNASSIGNED_ISR,     /* 108 (0x000001B0) (prior: -) */
```

Sysinit Files in the Project



kinetis_sysinit.c for Freescale Cup Cars

```
(tIsrFunc)UNASSIGNED_ISR,  /* 77 (0x00000134) (prior: -) */
(tIsrFunc)ftm0_isr,        /* 78 (0x00000138) (prior: -) */
(tIsrFunc)UNASSIGNED_ISR,  /* 79 (0x0000013C) (prior: -) */

...

(tIsrFunc)UNASSIGNED_ISR,  /* 102 (0x00000198) (prior: -) */
(tIsrFunc)porta_isr,       /* 103 (0x0000019C) (prior: -) */
(tIsrFunc)UNASSIGNED_ISR,  /* 104 (0x000001A0) (prior: -) */
(tIsrFunc)UNASSIGNED_ISR,  /* 105 (0x000001A4) (prior: -) */
(tIsrFunc)UNASSIGNED_ISR,  /* 106 (0x000001A8) (prior: -) */
(tIsrFunc)porte_isr,       /* 107 (0x000001AC) (prior: -) */
(tIsrFunc)UNASSIGNED_ISR,  /* 108 (0x000001B0) (prior: -) */
```

vector in the
vector table

5. Declare Interrupt Service Routine

Just like any other function in C, the Interrupt Service Routine has to be declared before it can be used. This has to be done in **main** and in **sysinit**.

```
//Function declarations

      void porta_isr(void);
      void porte_isr(void);
      void delay(void);
      void disable_wdog(void);
      void enable_irq(int);
      void pit0_isr(void);
      void pit1_isr(void);
      void adc1_isr(void);
```

5. Interrupt Service Routine in CodeWarrior

```
// ISR for PORTA interrupts
void porta_isr(void)
{
  PORTA_ISFR=0xFFFFFFFF;  //Clear Port A ISR flags
  PIT_LDVAL1 = PIT_LDVAL1 << 1;
  printf("count up\n",PIT_LDVAL1);
  updown = 1;
}


// ISR for PORTE interrupts
void porte_isr(void)
{
  PORTE_ISFR=0xFFFFFFFF;  //Clear Port E ISR flags
  PIT_LDVAL1 = PIT_LDVAL1 >> 1;
  printf("count down\n",PIT_LDVAL1);
  updown = 0;
}
```

5. Interrupt Status Flag Register (PORTx_ISFR)

The **Interrupt Status Flag** register has to be reset within the interrupt!

Addresses: PORTA_ISFR is 4004_9000h base + A0h offset = 4004_90A0h
PORTB_ISFR is 4004_A000h base + A0h offset = 4004_A0A0h
PORTC_ISFR is 4004_B000h base + A0h offset = 4004_B0A0h
PORTD_ISFR is 4004_C000h base + A0h offset = 4004_C0A0h
PORTE_ISFR is 4004_D000h base + A0h offset = 4004_D0A0h

Bit	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
R	ISF																															
W	w1c																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

31–0 ISF	Interrupt Status Flag Each bit in the field indicates the detection of the configured interrupt of the same number as the bit. 0 Configured interrupt has not been detected. 1 Configured interrupt has been detected.

5. Identification of the Interrupt Source

This is done in order to find out, which exactly pin has issued the interrupt.

11.4.1 Pin Control Register n (PORTx_PCR*n*)

For PCR1 to PCR5 of the port A, bit 0, 1, 6, 8, 9,10 reset to 1; for the PCR0 of the port A, bit 1, 6, 8, 9, 10 reset to 1; in other conditions, all bits reset to 0.

Addresses: 4004_9000h base + 0h offset + (4d × *n*), where *n* = 0d to 31d

Bit	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
R	0							ISF	0				IRQC				LK	0				MUX			0	DSE	ODE	PFE	0	SRE	PE	PS
W								w1c																								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PORTx_PCR*n* field descriptions

Field	Description
31–25 Reserved	This read-only field is reserved and always has the value zero.
24 ISF	Interrupt Status Flag The pin interrupt configuration is valid in all digital pin muxing modes. 0 Configured interrupt has not been detected. 1 Configured interrupt has been detected. If pin is configured to generate a DMA request then the corresponding flag will be cleared automatically at the completion of the requested DMA transfer, otherwise the flag remains set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted then flag will set again immediately.

Relocating the Vector Table

Some applications need the vector table to be located in RAM. For example in an RTOS implementation, the vector table needs to be in RAM, this allows the Kernel to install ISRs by modifying the vector table during runtime.

The NVIC provides a simple way to reallocate the vector table, for this purpose the user needs to set up the Vector Table Offset Register (VTOR) with the address offset for the new position. Use the bit TBLBASE[29] to indicate the table is either on RAM with 1 or flash with 0 and the TBLOFF[28:7] to indicate the address offset for the table.

The Cortex-M4 assumes the RAM starts at 0x20000000 and expects the vector table to be stored in that address if the VTOR TBLBASE[29] bit is set. Because the Kinetis MCU family RAM starts at 0x1fff0000, this bit must be cleared.

If the vector table is planned to be stored in RAM, you must the table copy from the flash to RAM. Also note that in some low power modes, a portion of the RAM will not be powered, which can lead to a vector table corruption. In this case, locate the vector table in the flash prior to entering a low power mode.

Literature and Links

http://www.arm.com/products/processors/cortex-m/cortex-m3.php

http://www.arm.com/products/processors/technologies/instruction-set-architectures.php

http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=K60

K60 Sub-Family Reference Manual

Kinetis Peripheral Module Quick Reference

Cortex-M4 Technical Reference Manual

http://www.arm.com/products/processors/cortex-m/index.php

TECHNISCHE HOCHSCHULE DEGGENDORF

Technische Hochschule Deggendorf – Edlmairstr. 6 und 8 – 94469 Deggendorf