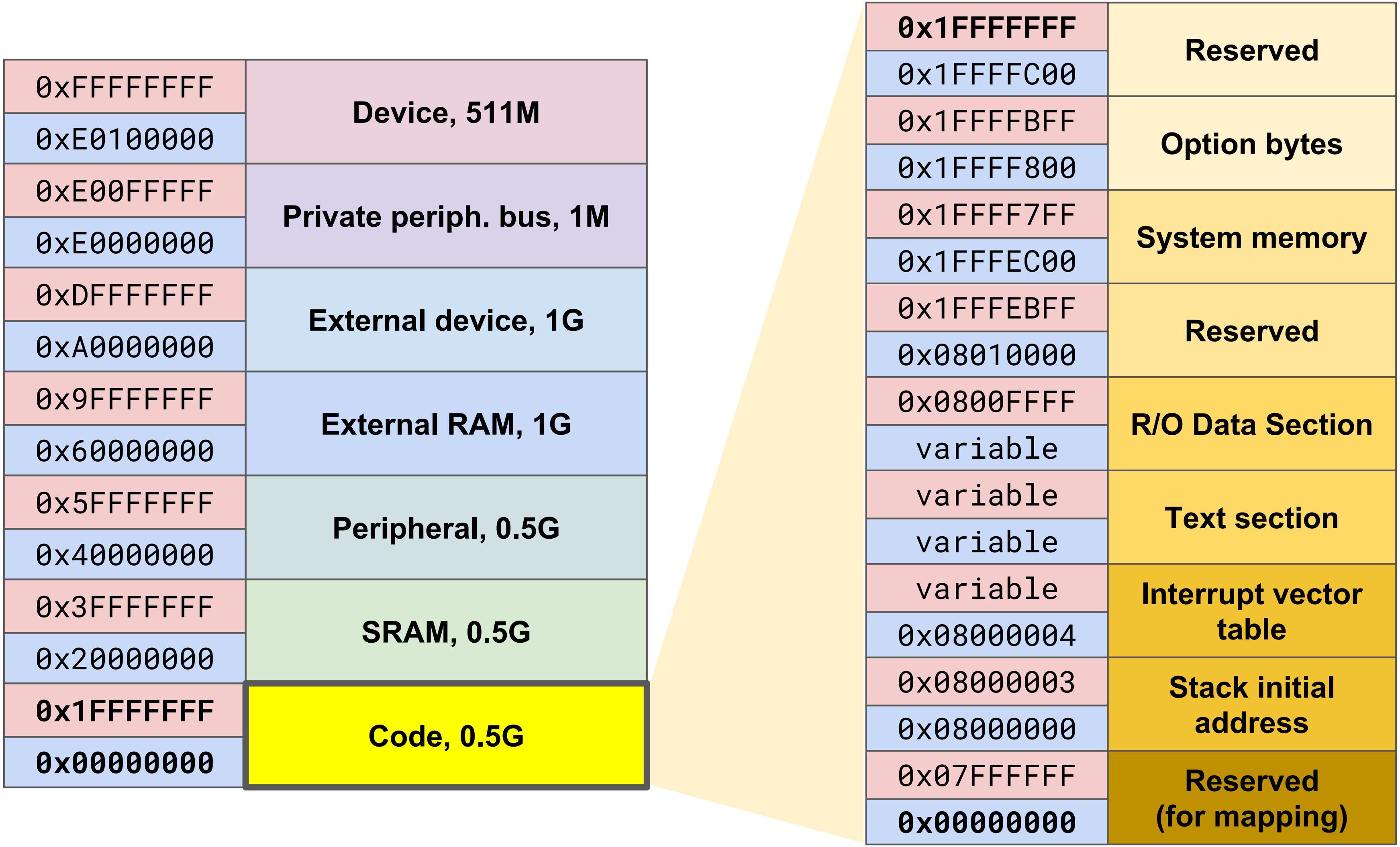
Introduction to embedded programming on STM32

NVIC, GPIO, EXTI

Skoltech, 2019

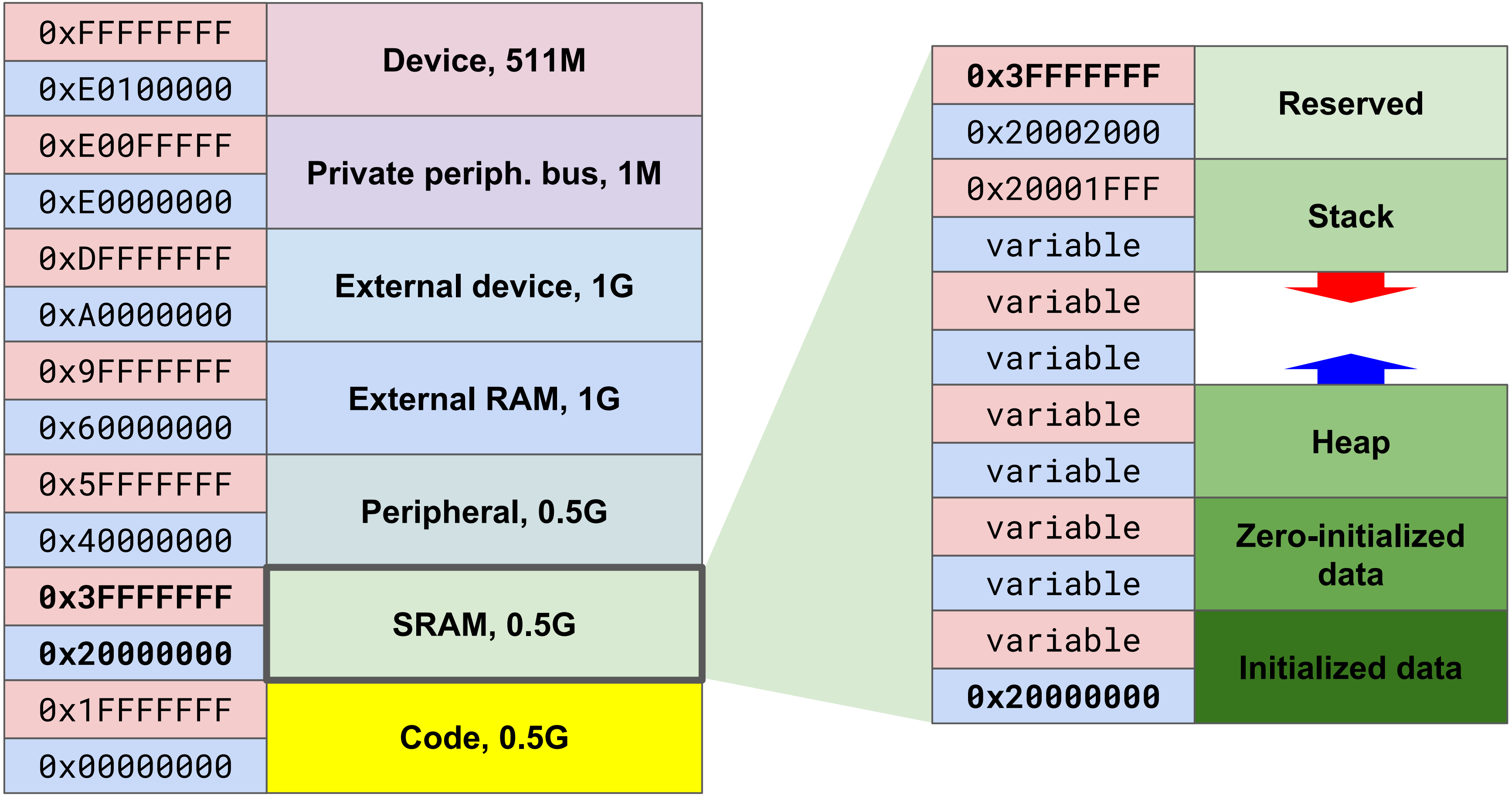
Memory Map

0xFFFFFFFF	**Device, 511M**
0xE0100000	
0xE00FFFFF	**Private periph. bus, 1M**
0xE0000000	
0xDFFFFFFF	**External device, 1G**
0xA0000000	
0x9FFFFFFF	**External RAM, 1G**
0x60000000	
0x5FFFFFFF	**Peripheral, 0.5G**
0x40000000	
0x3FFFFFFF	**SRAM, 0.5G**
0x20000000	
0x1FFFFFFF	**Code, 0.5G**
0x00000000	



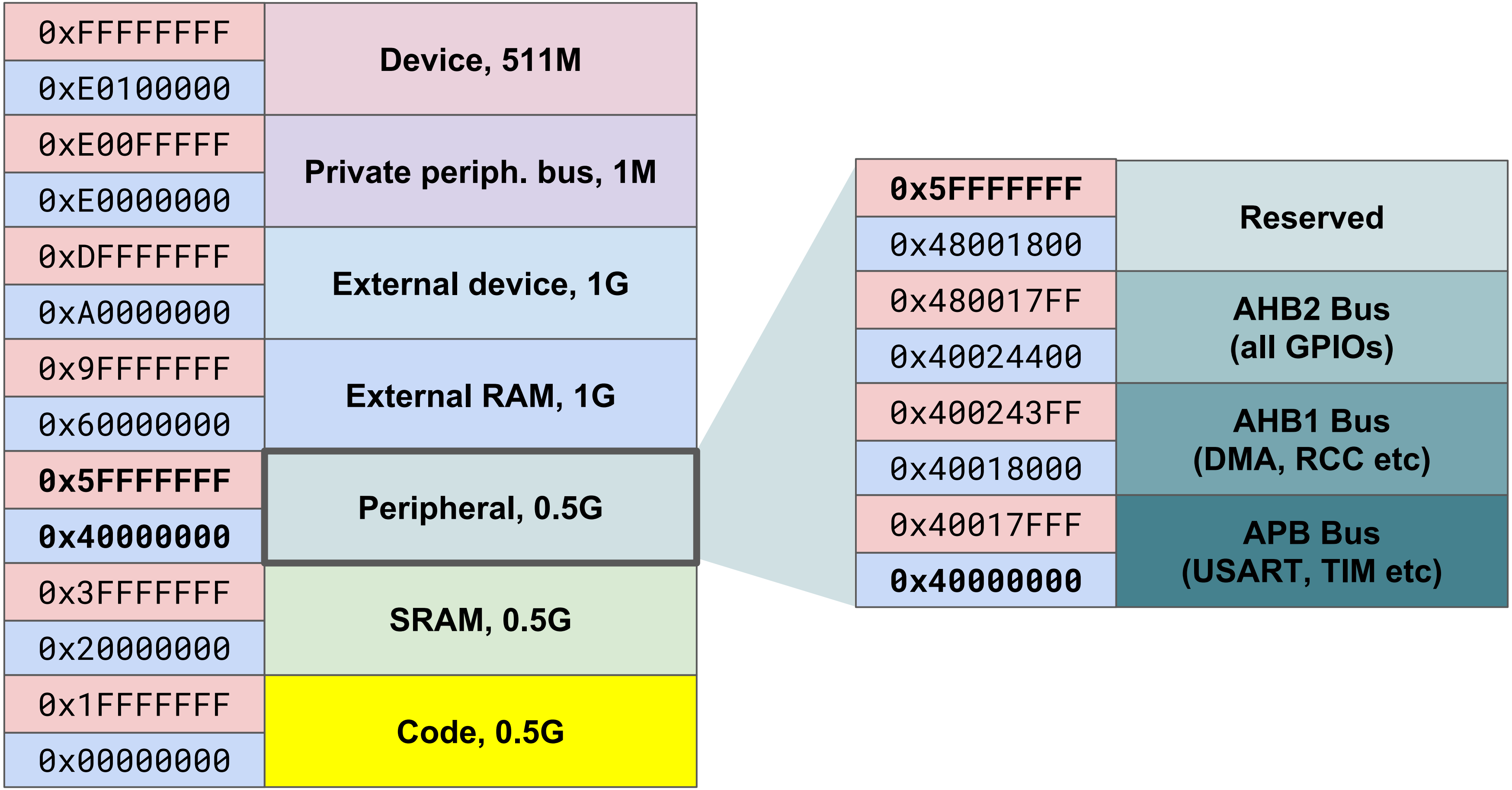
0xFFFFFFFF	Device, 511M
0xE0100000	
0xE00FFFFF	Private periph. bus, 1M
0xE0000000	
0xDFFFFFFF	External device, 1G
0xA0000000	
0x9FFFFFFF	External RAM, 1G
0x60000000	
0x5FFFFFFF	Peripheral, 0.5G
0x40000000	
0x3FFFFFFF	SRAM, 0.5G
0x20000000	
0x1FFFFFFF	**Code, 0.5G**
0x00000000	

0x1FFFFFFF	Reserved
0x1FFFFC00	
0x1FFFFBFF	Option bytes
0x1FFFF800	
0x1FFFF7FF	System memory
0x1FFFEC00	
0x1FFFEBFF	Reserved
0x08010000	
0x0800FFFF	R/O Data Section
variable	
variable	Text section
variable	
variable	Interrupt vector table
0x08000004	
0x08000003	Stack initial address
0x08000000	
0x07FFFFFF	Reserved (for mapping)
0x00000000	



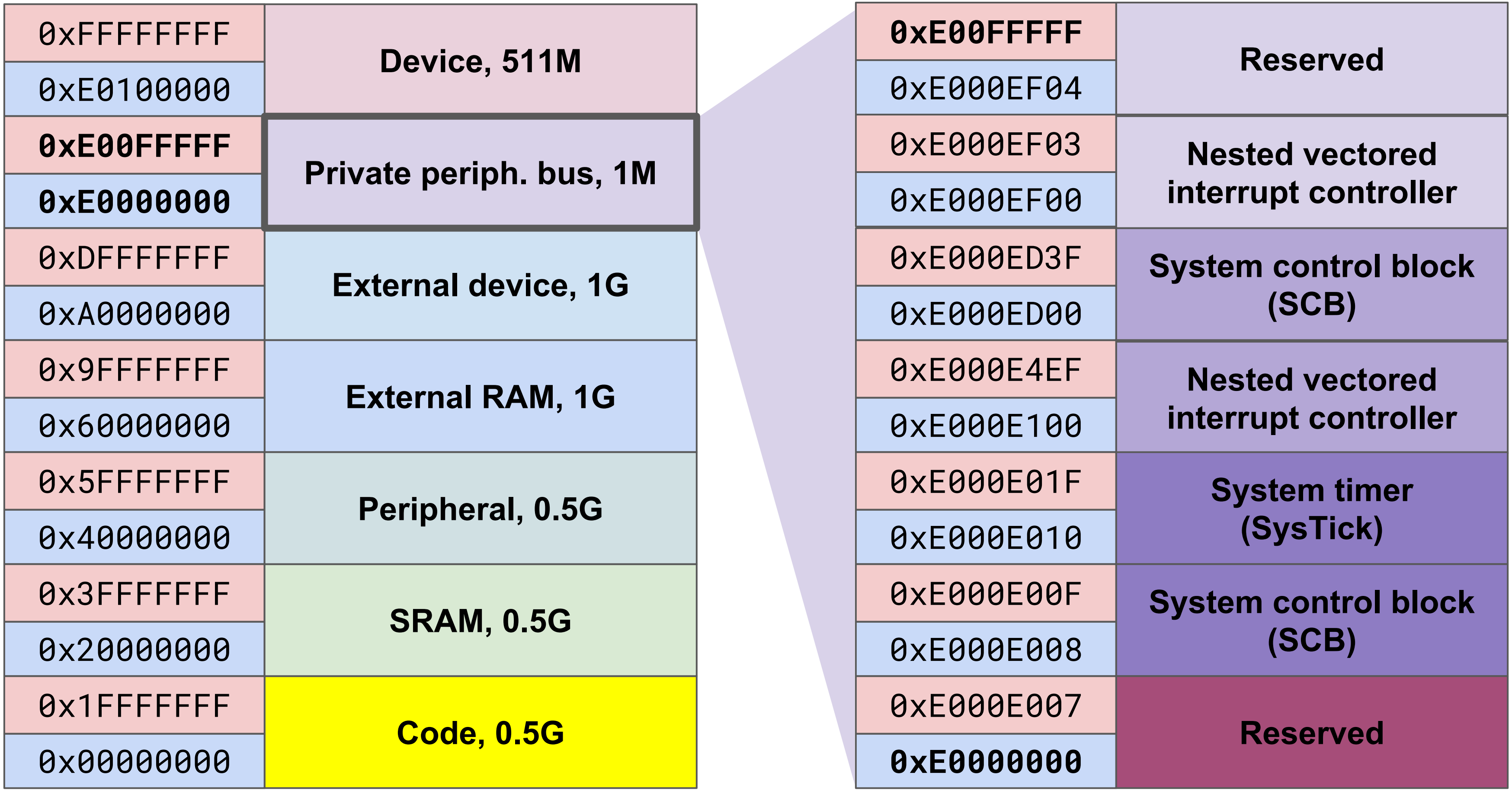
0xFFFFFFFF	Device, 511M
0xE0100000	
0xE00FFFFF	Private periph. bus, 1M
0xE0000000	
0xDFFFFFFF	External device, 1G
0xA0000000	
0x9FFFFFFF	External RAM, 1G
0x60000000	
0x5FFFFFFF	Peripheral, 0.5G
0x40000000	
0x3FFFFFFF	SRAM, 0.5G
0x20000000	
0x1FFFFFFF	Code, 0.5G
0x00000000	

0x3FFFFFFF	Reserved
0x20002000	
0x20001FFF	Stack
variable	
variable	
variable	
variable	Heap
variable	
variable	Zero-initialized data
variable	
variable	Initialized data
0x20000000	



0xFFFFFFFF	Device, 511M
0xE0100000	
0xE00FFFFF	Private periph. bus, 1M
0xE0000000	
0xDFFFFFFF	External device, 1G
0xA0000000	
0x9FFFFFFF	External RAM, 1G
0x60000000	
0x5FFFFFFF	Peripheral, 0.5G
0x40000000	
0x3FFFFFFF	SRAM, 0.5G
0x20000000	
0x1FFFFFFF	Code, 0.5G
0x00000000	

0x5FFFFFFF	Reserved
0x48001800	
0x480017FF	AHB2 Bus (all GPIOs)
0x40024400	
0x400243FF	AHB1 Bus (DMA, RCC etc)
0x40018000	
0x40017FFF	APB Bus (USART, TIM etc)
0x40000000	



0xFFFFFFFF 0xE0100000	Device, 511M
0xE00FFFFF **0xE0000000**	Private periph. bus, 1M
0xDFFFFFFF 0xA0000000	External device, 1G
0x9FFFFFFF 0x60000000	External RAM, 1G
0x5FFFFFFF 0x40000000	Peripheral, 0.5G
0x3FFFFFFF 0x20000000	SRAM, 0.5G
0x1FFFFFFF 0x00000000	Code, 0.5G

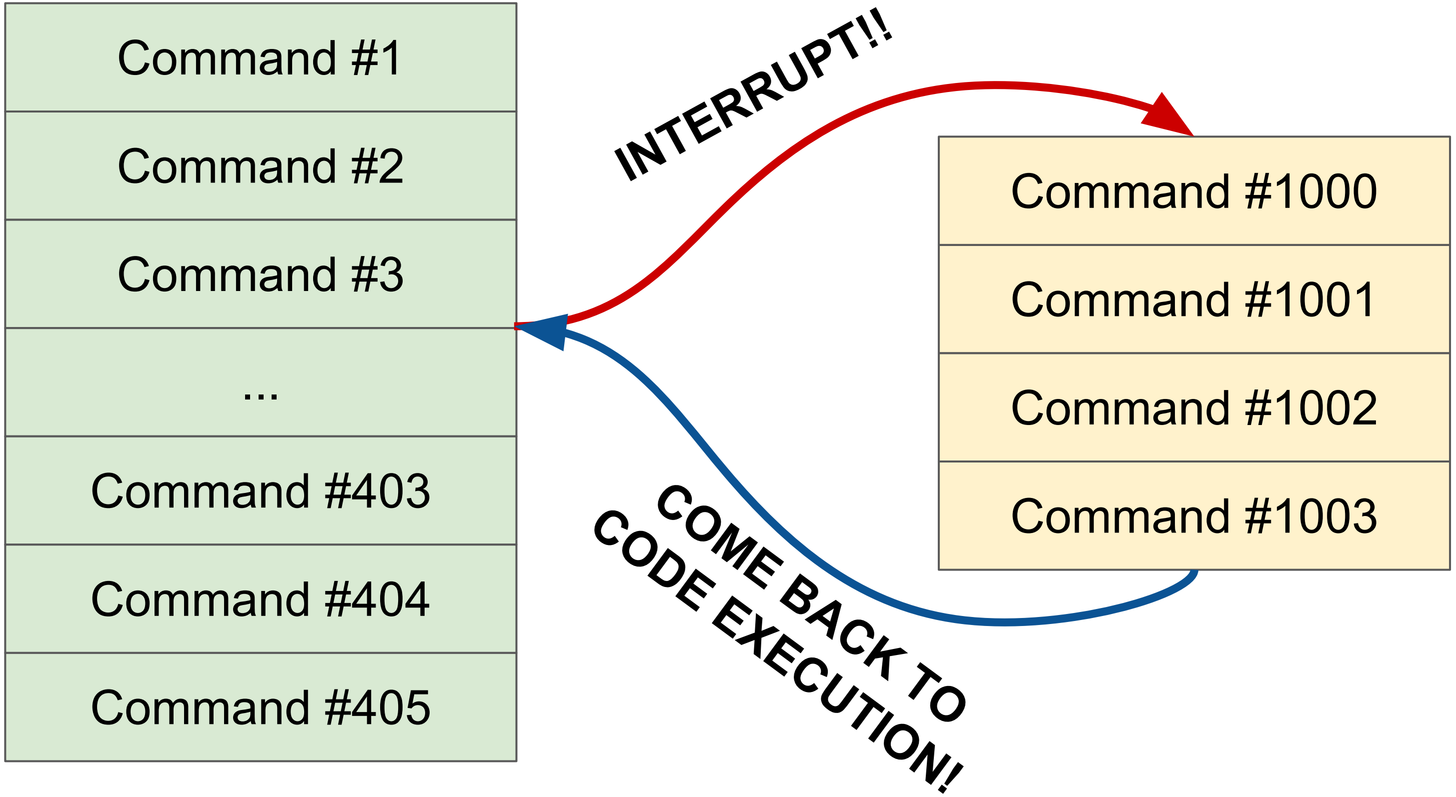
0xE00FFFFF 0xE000EF04	Reserved
0xE000EF03 0xE000EF00	Nested vectored interrupt controller
0xE000ED3F 0xE000ED00	System control block (SCB)
0xE000E4EF 0xE000E100	Nested vectored interrupt controller
0xE000E01F 0xE000E010	System timer (SysTick)
0xE000E00F 0xE000E008	System control block (SCB)
0xE000E007 **0xE0000000**	Reserved

Why would we need interrupts?

There are *two* main approaches:

- **Polling** - pick up the phone every second to check if you are getting a call
- **Interrupts** - work on whatever you need, pick up the phone once it rings

Interrupts concept



Command #1

Command #2

Command #3

...

Command #403

Command #404

Command #405

INTERRUPT!!

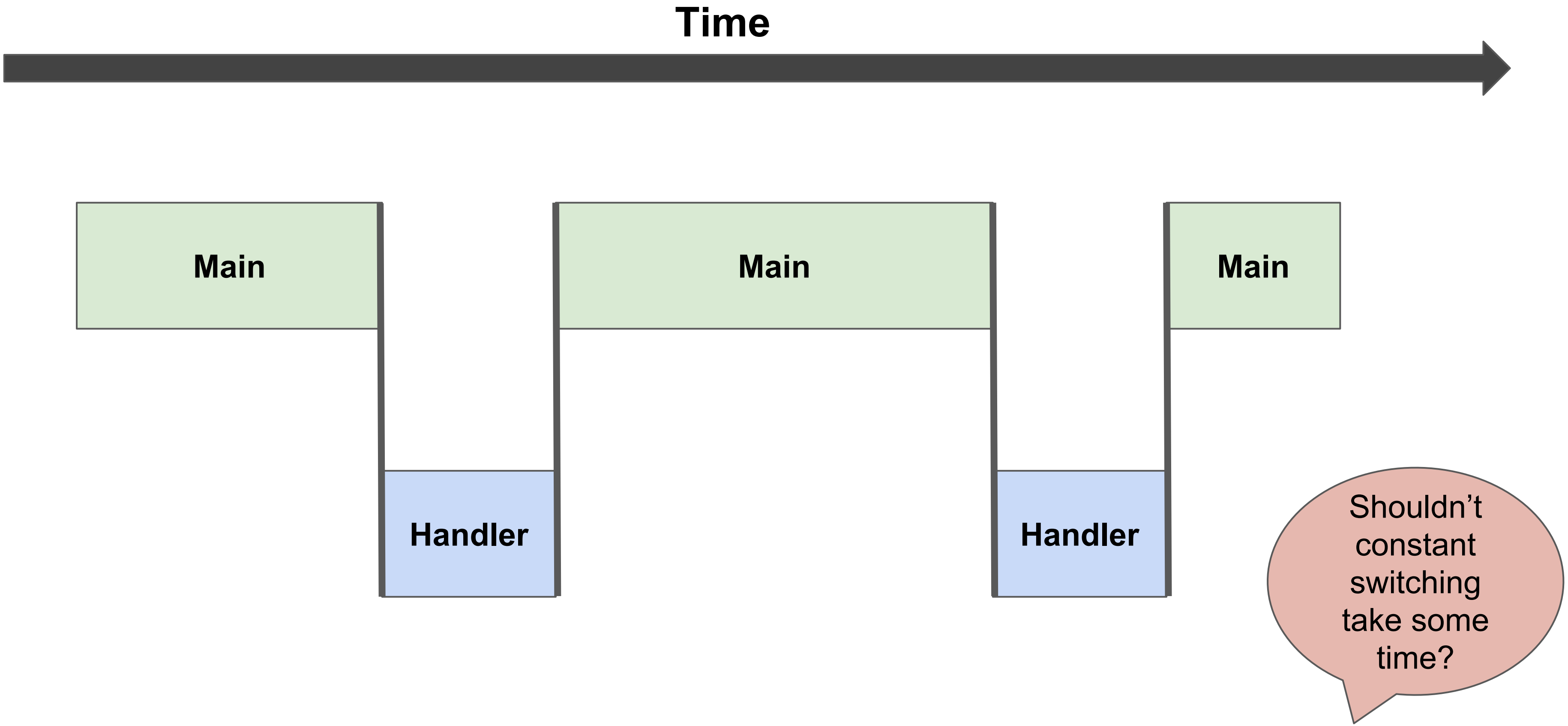
Command #1000

Command #1001

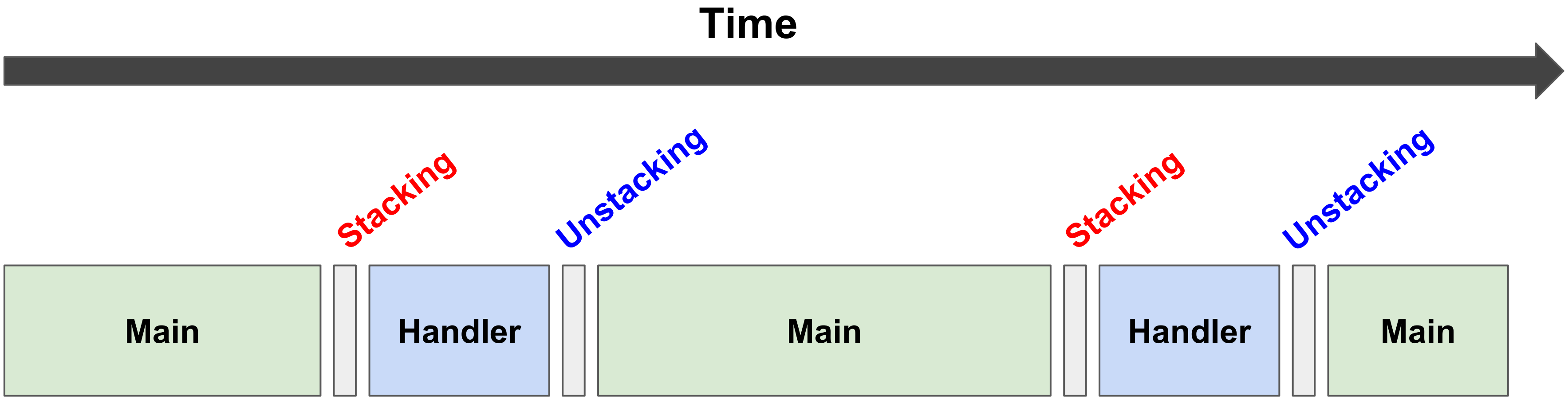
Command #1002

Command #1003

COME BACK TO CODE EXECUTION!



Shouldn’t constant switching take some time?



Time

Main

Stacking

Handler

Unstacking

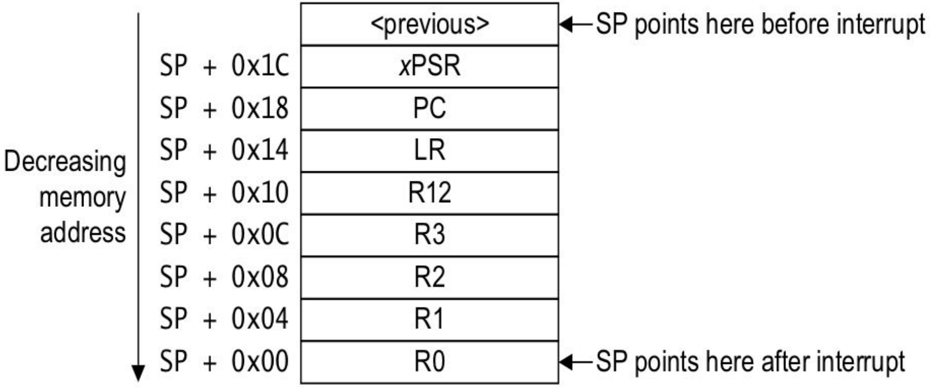
Main

Stacking

Handler

Unstacking

Main



	<previous>
SP + 0x1C	*x*PSR
SP + 0x18	PC
SP + 0x14	LR
SP + 0x10	R12
SP + 0x0C	R3
SP + 0x08	R2
SP + 0x04	R1
SP + 0x00	R0

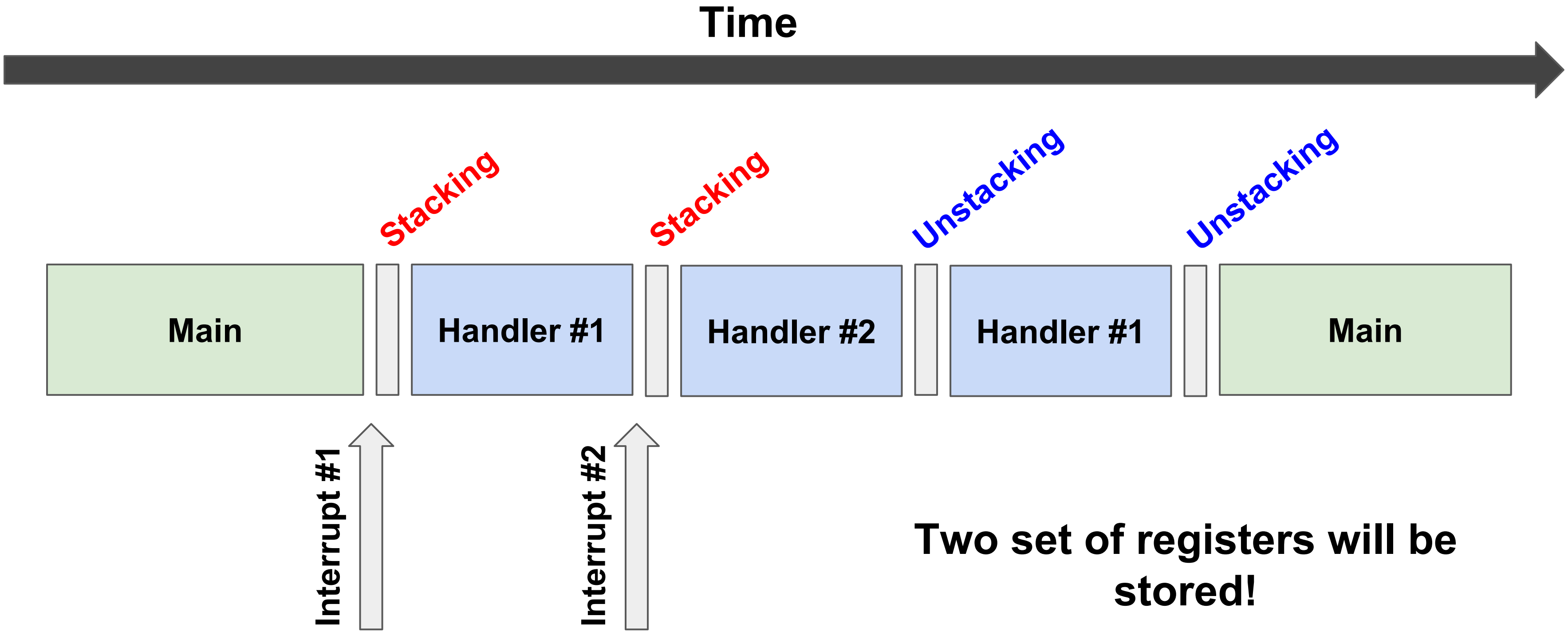
SP points here before interrupt

SP points here after interrupt

Decreasing memory address

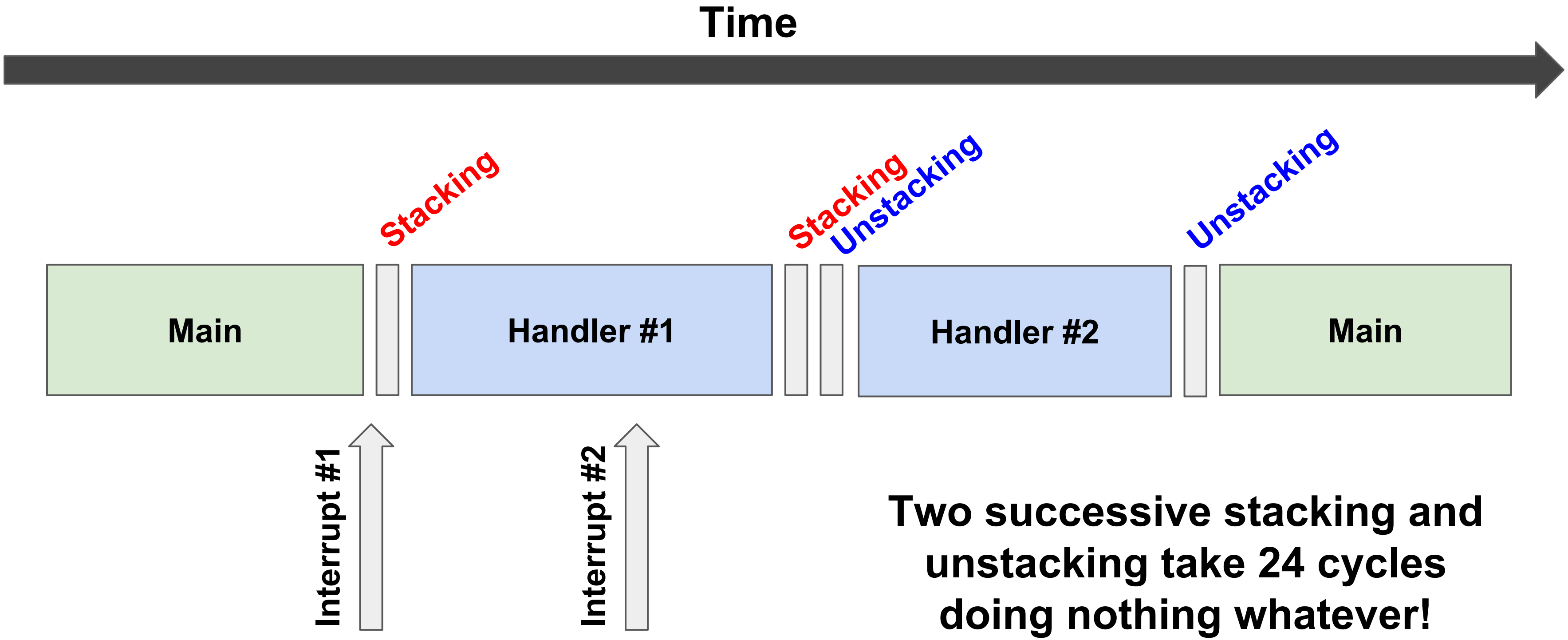
Each stacking/unstacking procedure takes 12 cycles and atomic!

Nested interrupts. Preemption. Higher urgency



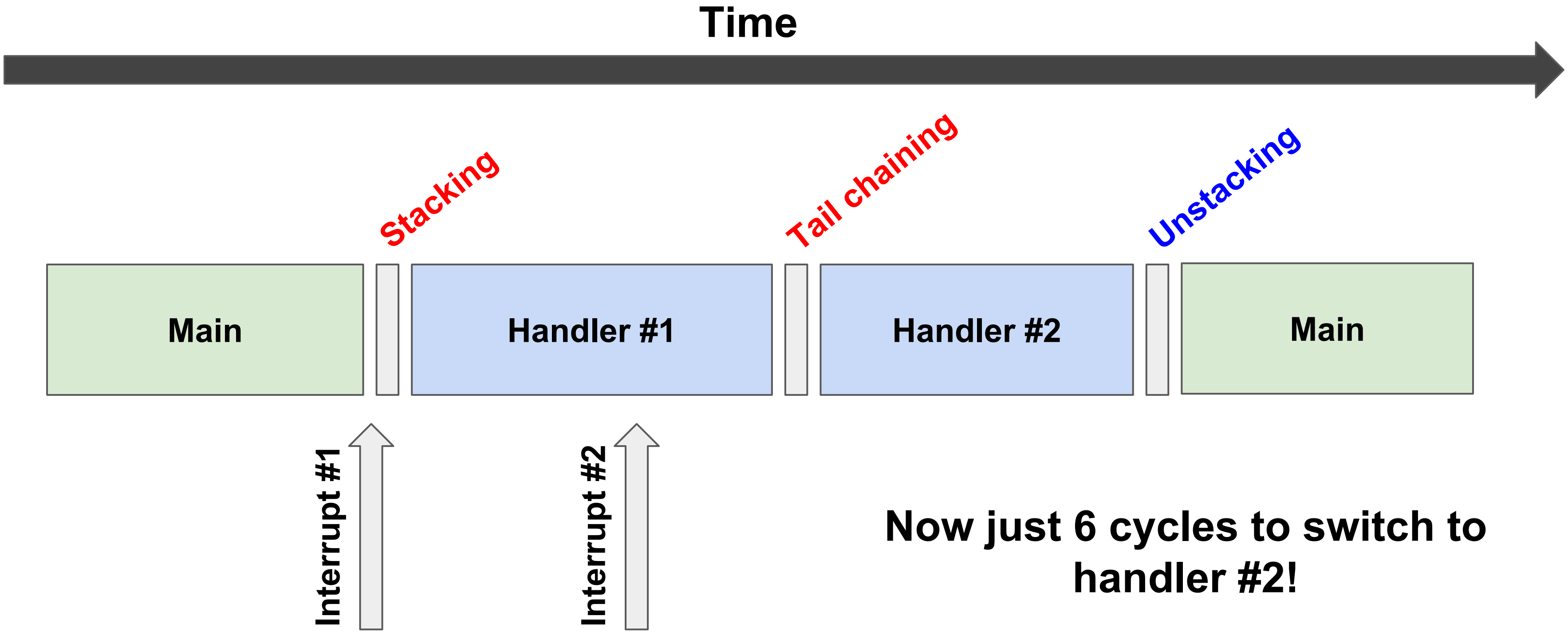
Two set of registers will be stored!

Nested interrupts. Lower urgency



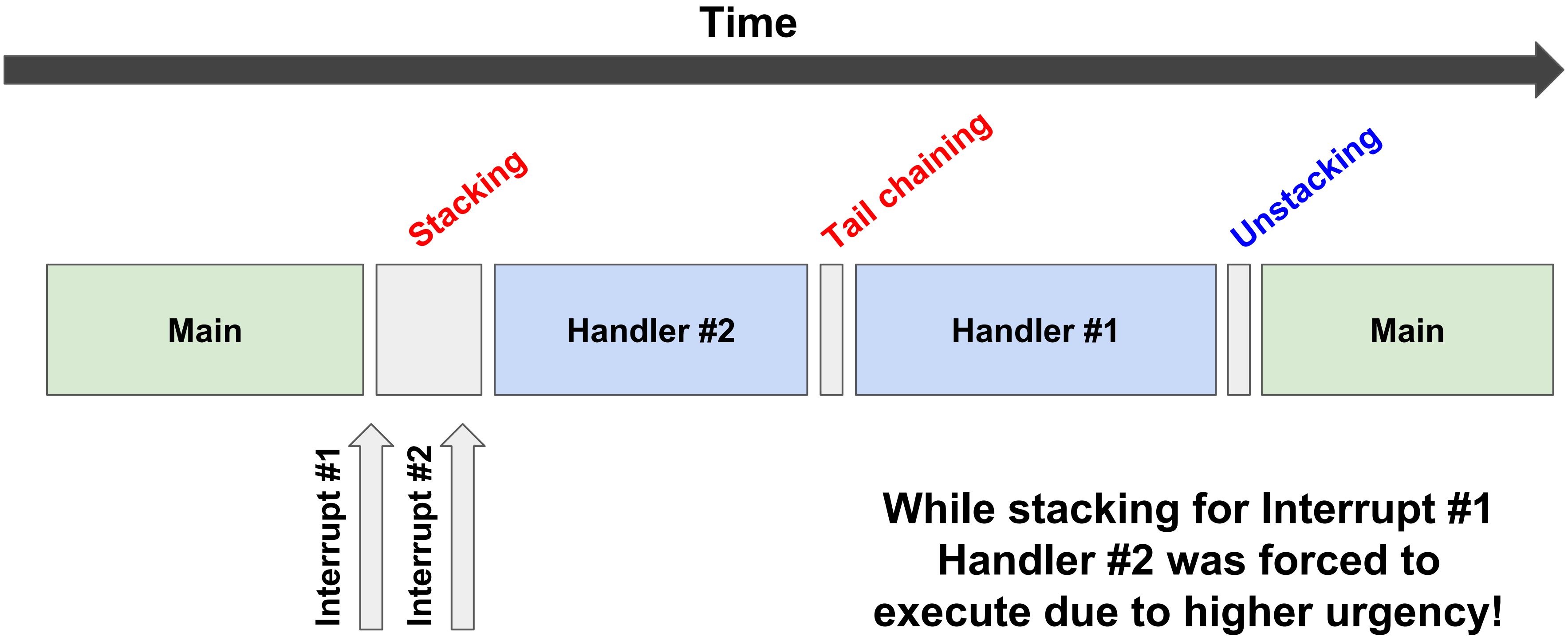
Two successive stacking and unstacking take 24 cycles doing nothing whatever!

Nested interrupts. Tail-chaining



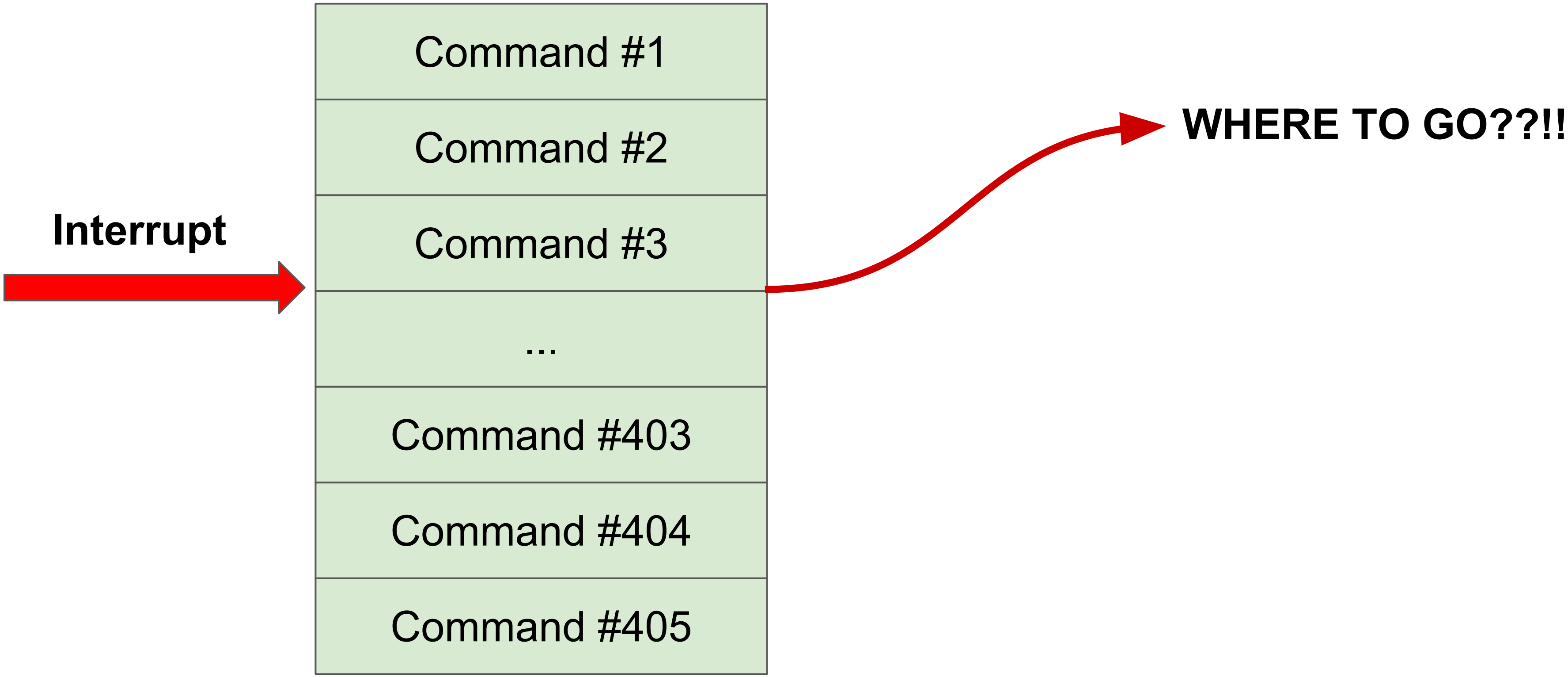
Now just 6 cycles to switch to handler #2!

Nested interrupts. Late-arriving



While stacking for Interrupt #1 Handler #2 was forced to execute due to higher urgency!

Interrupt. Vector table



Interrupt

Command #1

Command #2

Command #3

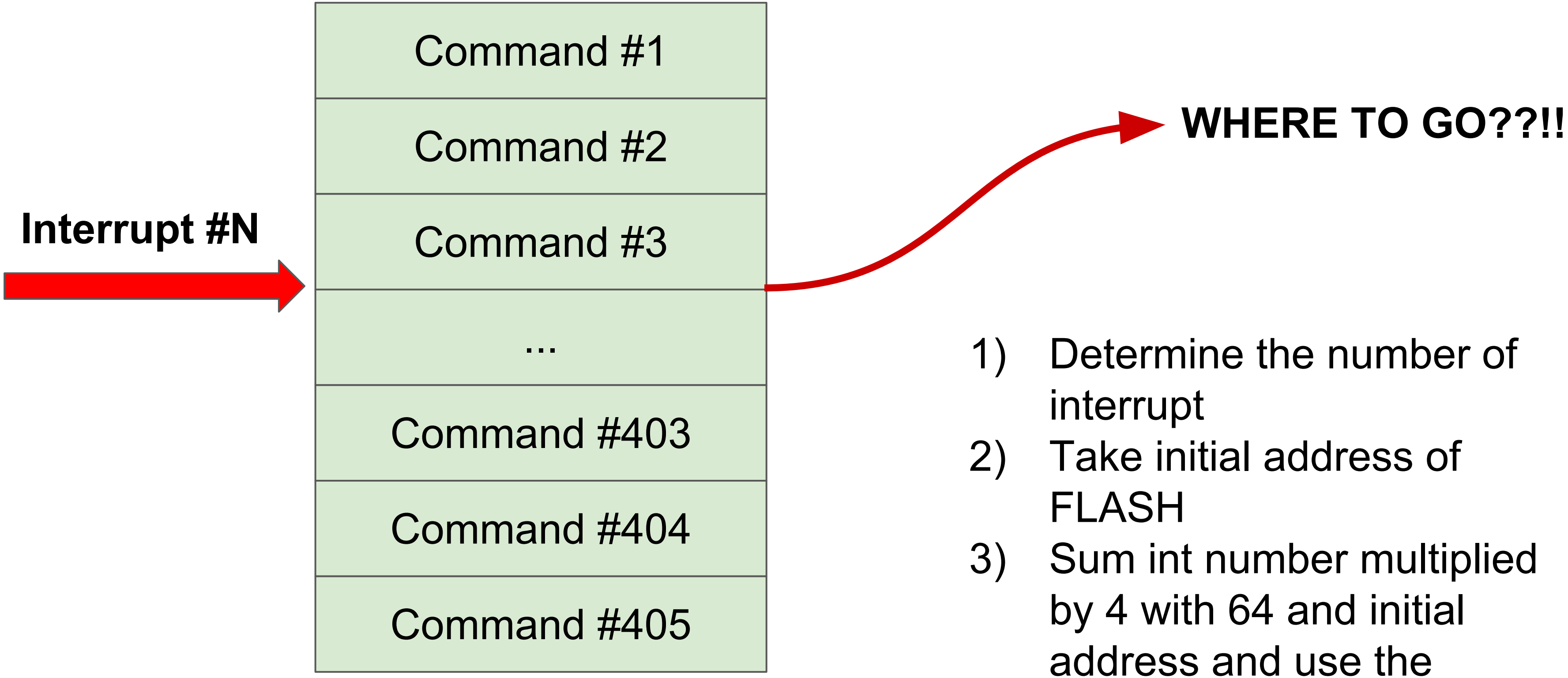
...

Command #403

Command #404

Command #405

WHERE TO GO??!!



Interrupt #N

Command #1

Command #2

Command #3

...

Command #403

Command #404

Command #405

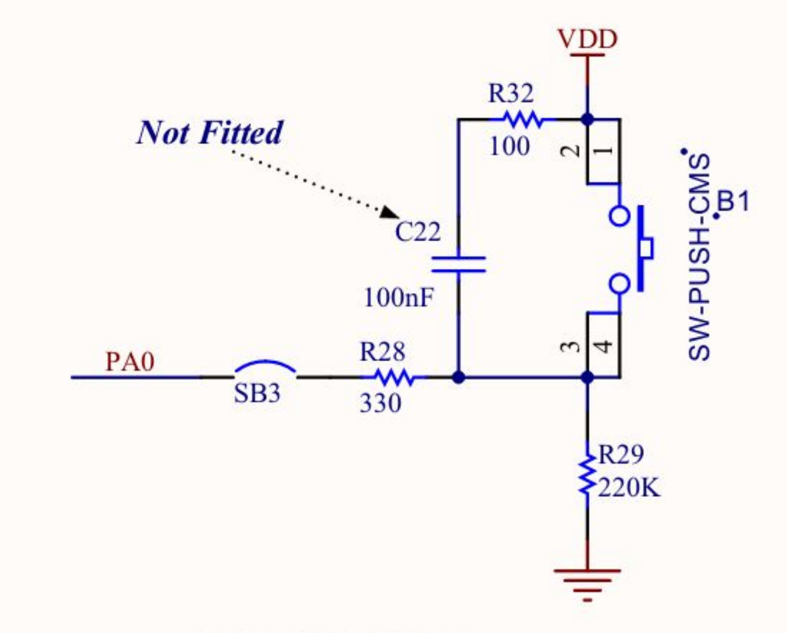
WHERE TO GO??!!

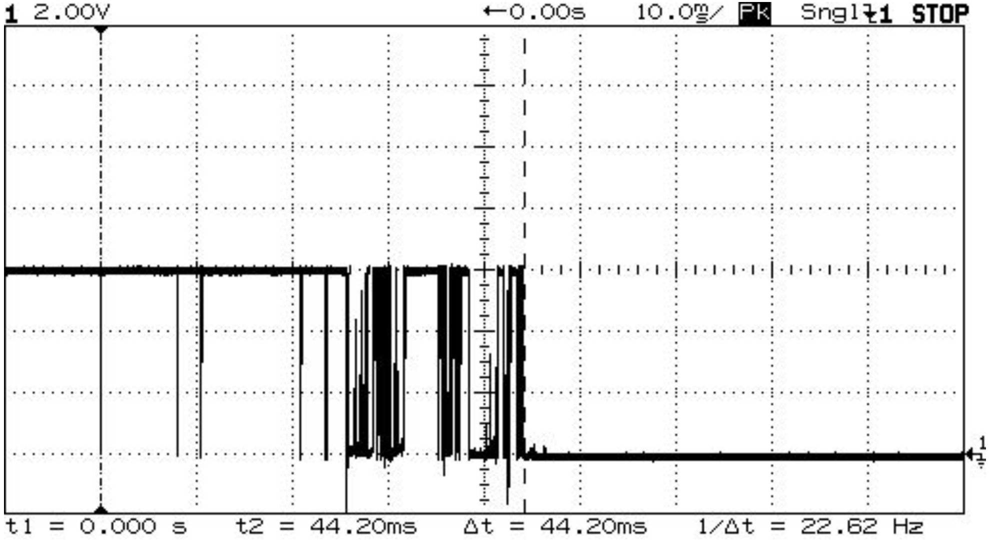
1) Determine the number of interrupt
2) Take initial address of FLASH
3) Sum int number multiplied by 4 with 64 and initial address and use the resulted address to jump

Nested vectored interrupt controller (NVIC)

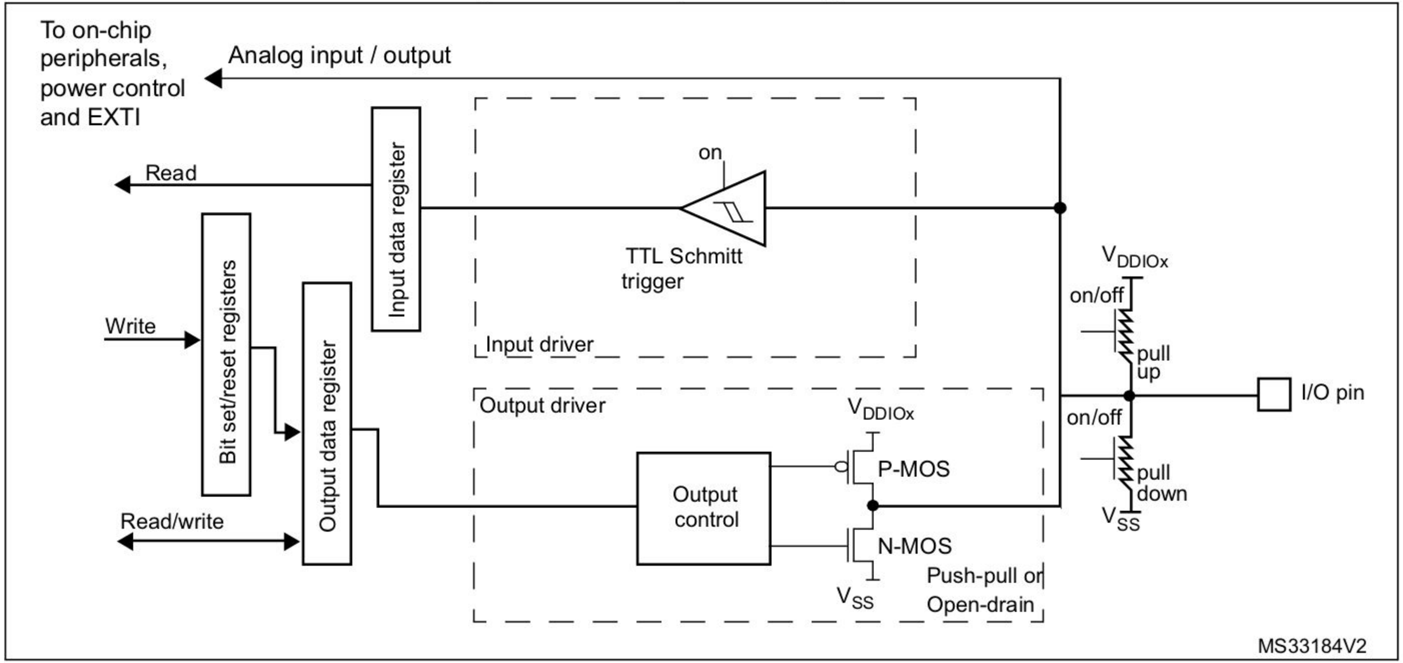
- void NVIC_EnableIRQ(IRQn_Type IRQn)
- void NVIC_DisableIRQ(IRQn_Type IRQn)
- void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)
- uint32_t NVIC_GetPriority(IRQn_Type IRQn)

Inputs-Outputs



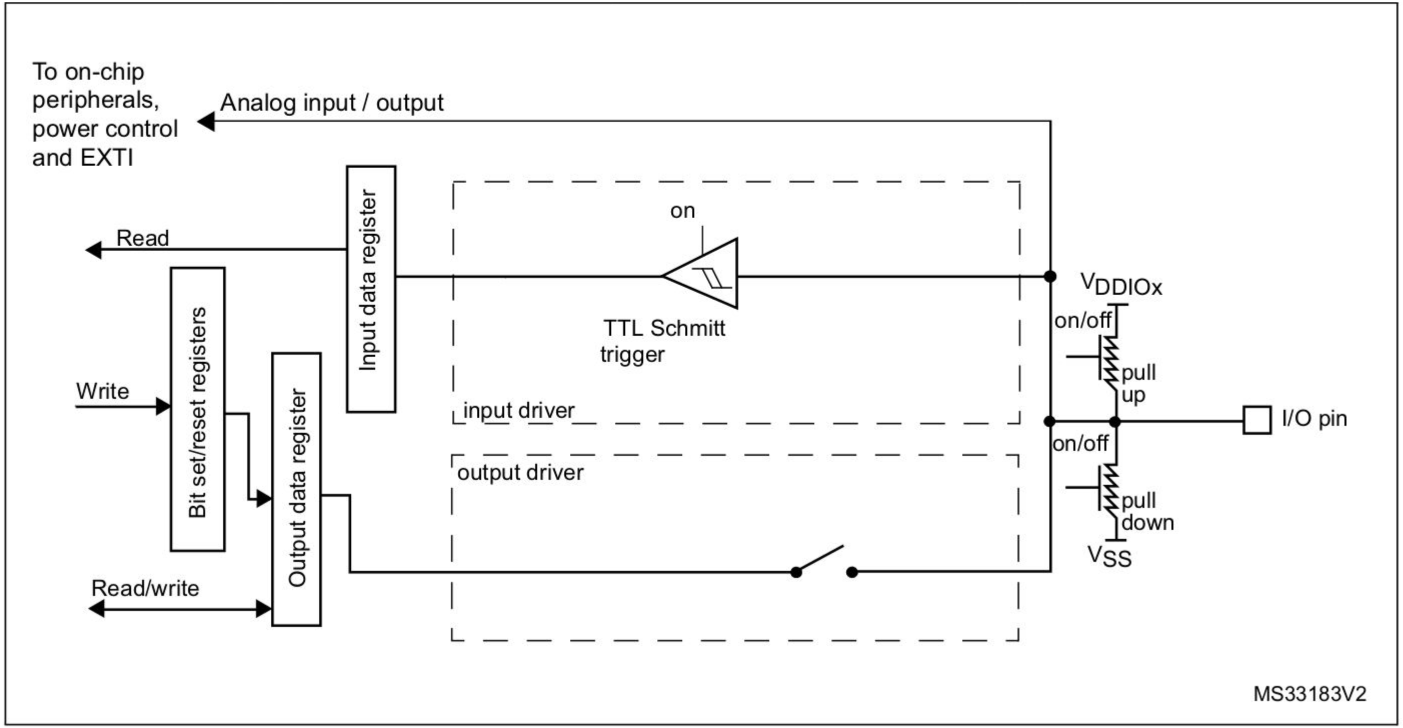


GPIO. Output configuration



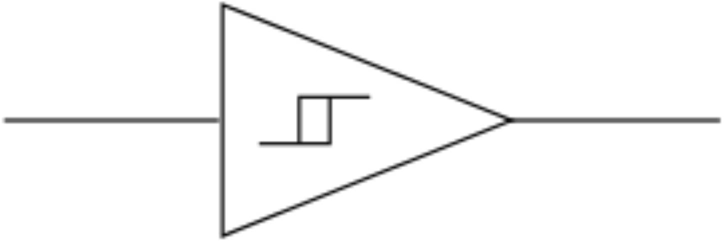
- GPIOx_MODER (Input, Output, Alternate function, Analog)
- GPIOx_OTYPER (Push-Pull : 1 -> P-MOS, Open-Drain : 1 -> Hi-Z)

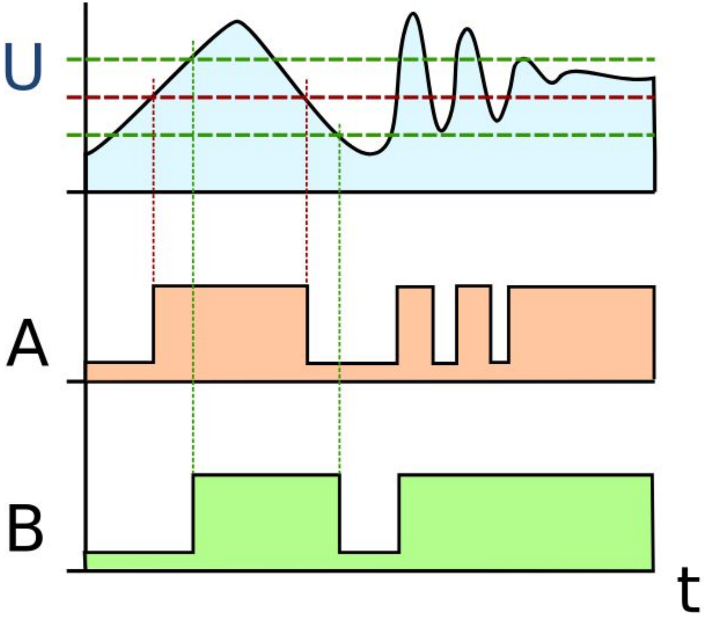
GPIO. Input configuration



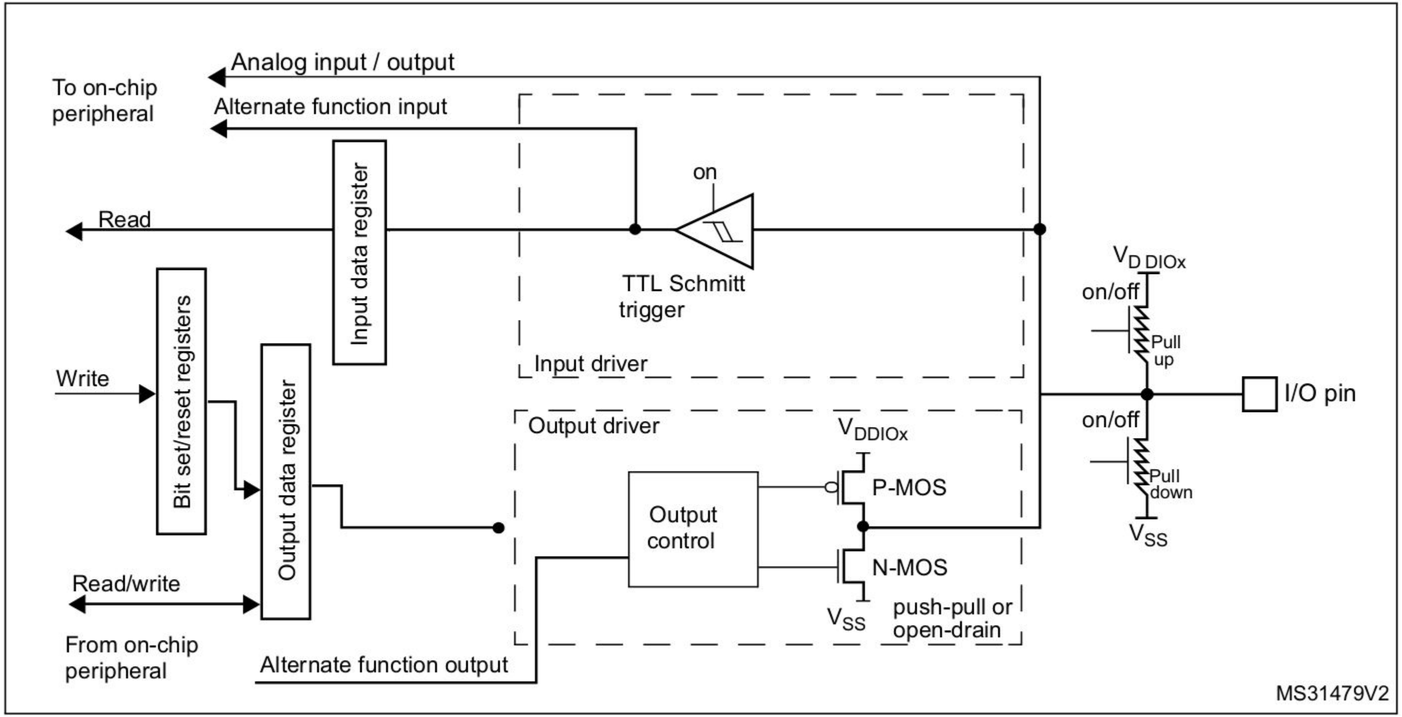
- GPIOx_PUPDR (none, pull-up, pull-down)
- GPIOx_IDR

GPIO. Input configuration. Schmitt trigger



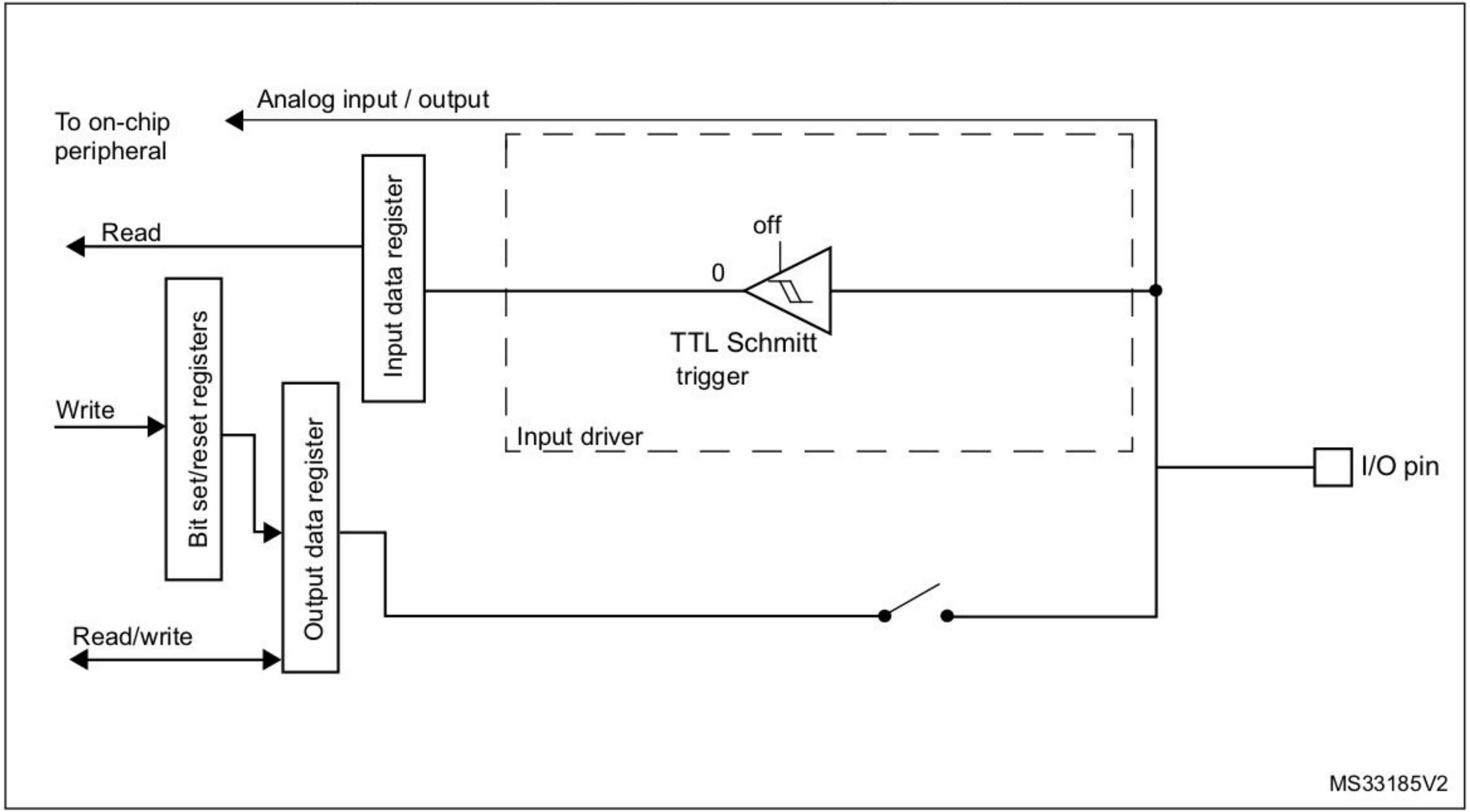


GPIO. Alternate function configuration



- GPIOx_AFRL (0 - 7 pins)
- GPIOx_AFRH (8 - 15 pins)

GPIO. Analog configuration



GPIO. LL main subroutine. Part 1

- LL_GPIO_SetPinMode(GPIOx, LL_GPIO_PIN_x, LL_GPIO_MODE_x)
    - LL_GPIO_MODE_INPUT
    - LL_GPIO_MODE_OUTPUT
    - LL_GPIO_MODE_ALTERNATE
    - LL_GPIO_MODE_ANALOG
- LL_GPIO_SetPinOutputType(GPIOx, LL_GPIO_PIN_x, LL_GPIO_OUTPUT_x)
    - LL_GPIO_OUTPUT_PUSHPULL
    - LL_GPIO_OUTPUT_OPENDRAIN
- LL_GPIO_SetPinPull(GPIOx, LL_GPIO_PIN_x, LL_GPIO_PULL_x)
    - LL_GPIO_PULL_NO
    - LL_GPIO_PULL_UP
    - LL_GPIO_PULL_DOWN
- LL_GPIO_SetAFPin_0_7(GPIOx, LL_GPIO_PIN_x, LL_GPIO_AF_x)
- LL_GPIO_SetAFPin_8_15(GPIOx, LL_GPIO_PIN_x, LL_GPIO_AF_x)

GPIO. Alternate function configuration

GPIO_AF_0	EVENTOUT, SWDIO, SWCLK, MCO, CEC, CRS, IR, SPI1, SPI2, TIM1, TIM3, TIM14, TIM15, TIM16, TIM17, TSC, USART1, USART2, USART3, USART4, USART8, CAN
GPIO_AF_1	TIM3, TIM15, USART{1-8}, IR, CEC, EVENTOUT, I2C1, I2C2, TSC, SPI1, SPI2
GPIO_AF_2	TIM2, TIM1, TIM16, TIM17, EVENTOUT, USART{5-8}
GPIO_AF_3	TSC, I2C1, TIM15, EVENTOUT
GPIO_AF_4	TIM14, USART{3-5}, CRS, CAN, I2C1
GPIO_AF_5	TIM15, TIM16, TIM17, SPI2, I2C2, MCO, USART6
GPIO_AF_6	EVENTOUT
GPIO_AF_7	COMP1, COMP2

GPIO. LL main subroutine. Part 2

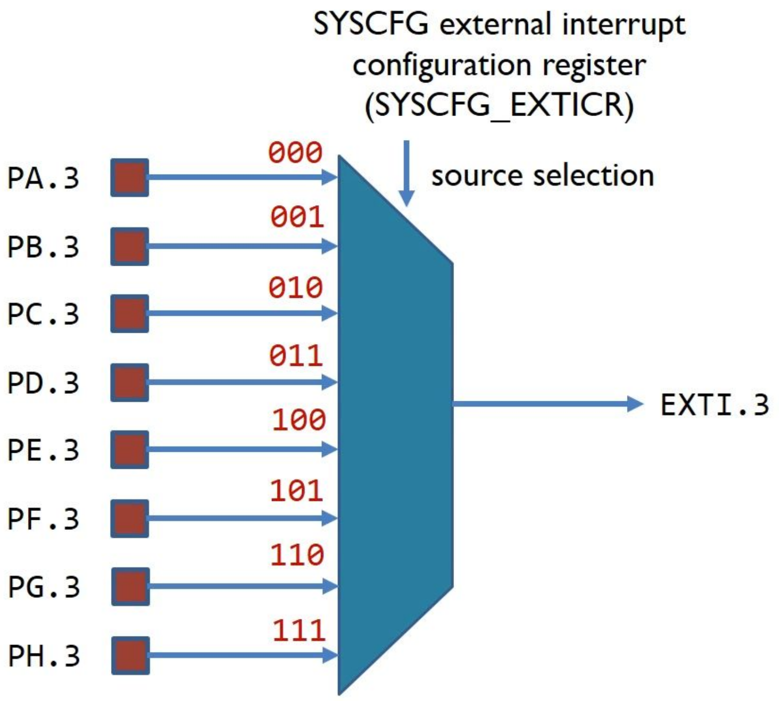
- LL_GPIO_ReadInputPort(GPIOx) -> IDR
- LL_GPIO_IsInputPinSet(GPIOx, LL_GPIO_PIN_x)
- LL_GPIO_WriteOutputPort(GPIOx, PortValue) -> ODR
- LL_GPIO_ReadOutputPort(GPIOx)
- LL_GPIO_IsOutputPinSet(GPIOx, LL_GPIO_PIN_x)
- LL_GPIO_SetOutputPin(GPIOx, LL_GPIO_PIN_x) -> BSRR
- LL_GPIO_ResetOutputPin (GPIOx, LL_GPIO_PIN_x) -> BRR
- LL_GPIO_TogglePin(GPIOx, LL_GPIO_PIN_x) -> RMW(ODR)

Small coding practice!

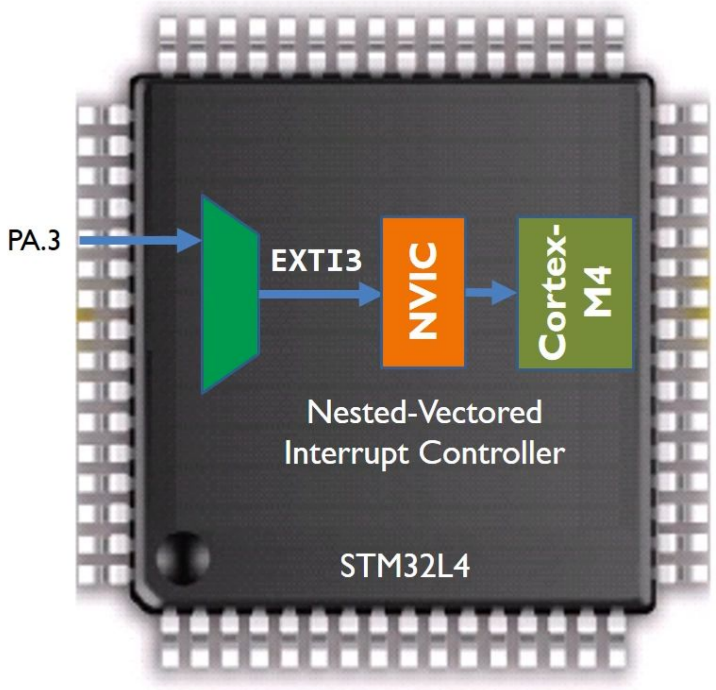
EXTI. Extended interrupts and events controller

- Supports generation of up to 32 event/interrupt requests
- Independent mask on each event/interrupt line
- Independent trigger for external event/interrupt line
- Dedicated status bit for external interrupt line
- Emulation for all the external event requests

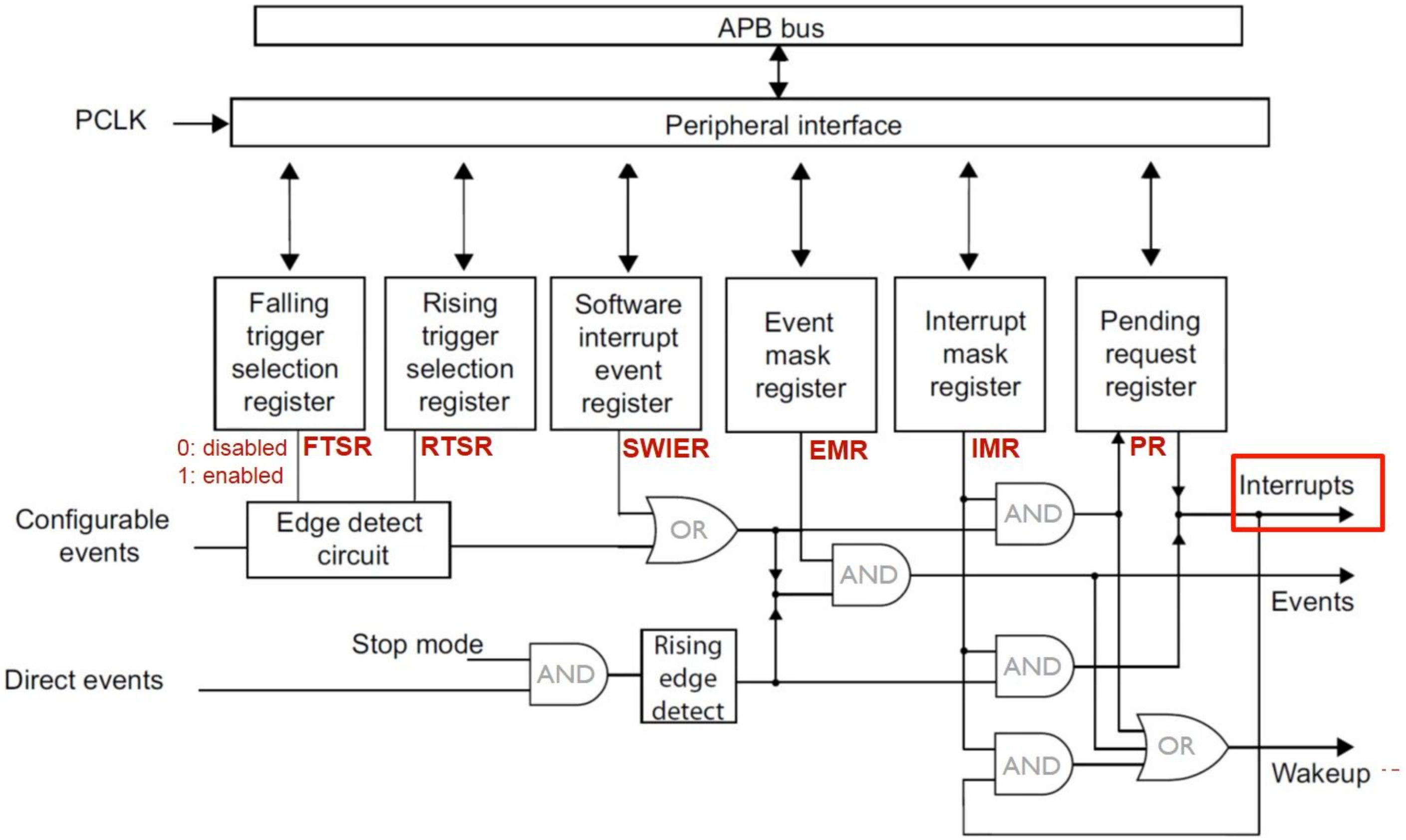
EXTI. Basic diagram



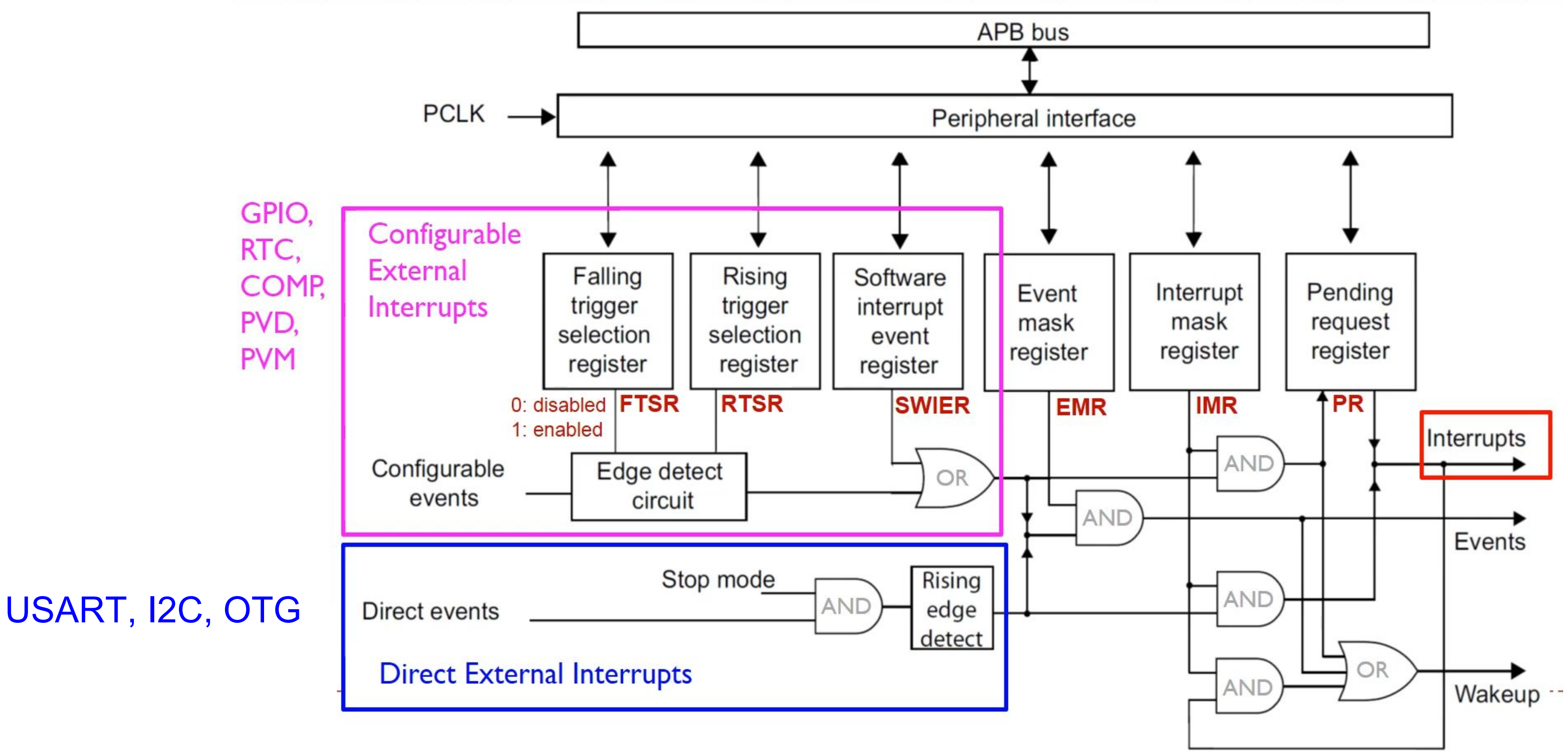
Select pin x from Port y as EXTIx



EXTI. Block diagram



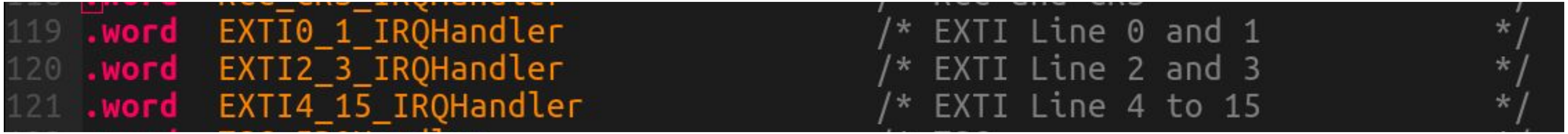
EXTI. Block diagram



EXTI. Main LL subroutines

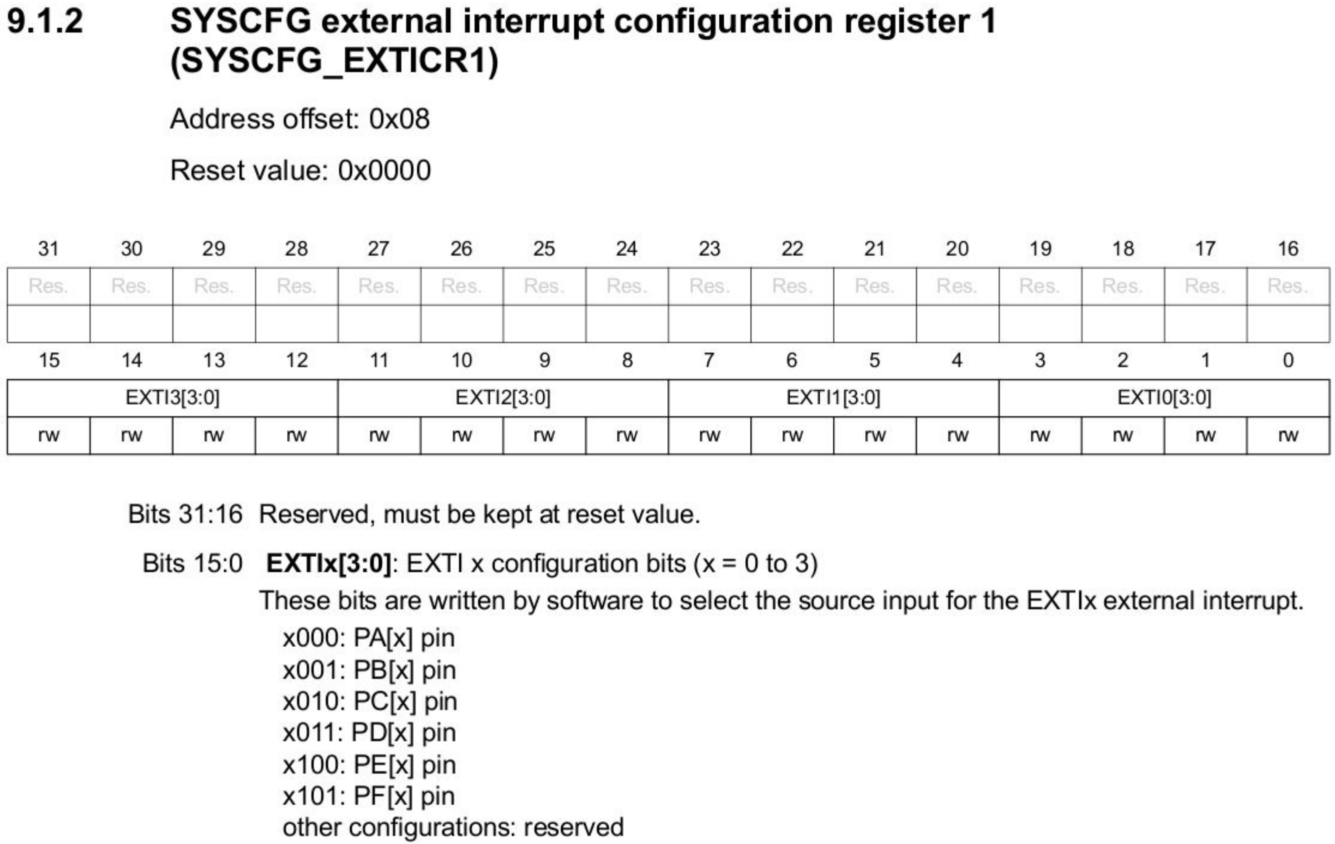
- LL_EXTI_EnableIT_0_31(LL_EXTI_LINE_x)
- LL_EXTI_EnableEvent_0_31(LL_EXTI_LINE_x)
- LL_EXTI_EnableRisingTrig_0_31(LL_EXTI_LINE_x)
- LL_EXTI_EnableFallingTrig_0_31(LL_EXTI_LINE_x)
- LL_EXTI_GenerateSWI_0_31*(LL_EXTI_LINE_x)*
- LL_EXTI_ClearFlag_0_31(LL_EXTI_LINE_x)
- LL_EXTI_IsActiveFlag_0_31(LL_EXTI_LINE_x)

EXTI handlers in STM32F051



```
.word  EXTI0_1_IRQHandler                     /* EXTI Line 0 and 1             */
.word  EXTI2_3_IRQHandler                     /* EXTI Line 2 and 3             */
.word  EXTI4_15_IRQHandler                    /* EXTI Line 4 to 15             */
```

System configuration controller (SYSCFG)



9.1.2 SYSCFG external interrupt configuration register 1 (SYSCFG_EXTICR1)

Address offset: 0x08

Reset value: 0x0000

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

Bits 31:16 Reserved, must be kept at reset value.

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

These bits are written by software to select the source input for the EXTIx external interrupt.

x000: PA[x] pin
x001: PB[x] pin
x010: PC[x] pin
x011: PD[x] pin
x100: PE[x] pin
x101: PF[x] pin
other configurations: reserved

Small coding practice!