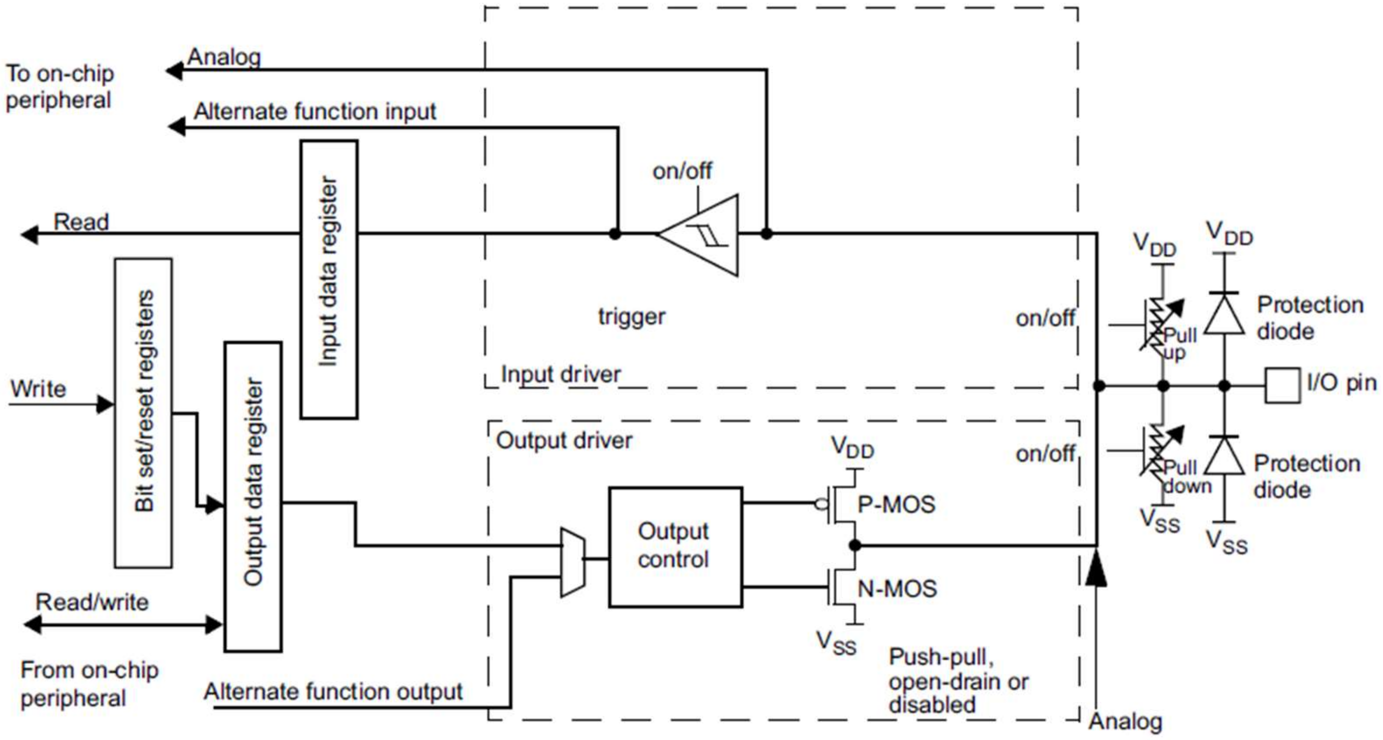
PART 5: Embedded Programming STM32

GPIOs, basics in electronics (push-pull/Open Drain), polling vs interrupt, Configure GPIO as Interrupt

GPIOs overview

- ✓ GPIO stands for General Purpose Input/Output
- ✓ It is part of the peripherals registers
- ✓ It is the peripheral that controls the pins allowing to communicate with the external world
- ✓ The connection of the pins with the CPU is done through a PORT

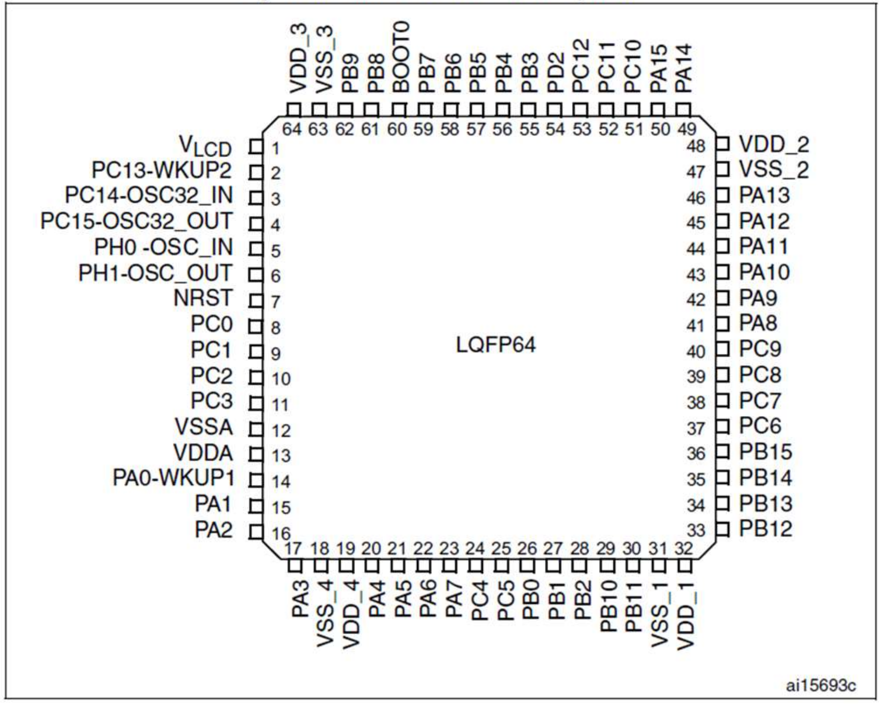
Standard I/O Port Bit Schematic



Pinout STM32L152RE

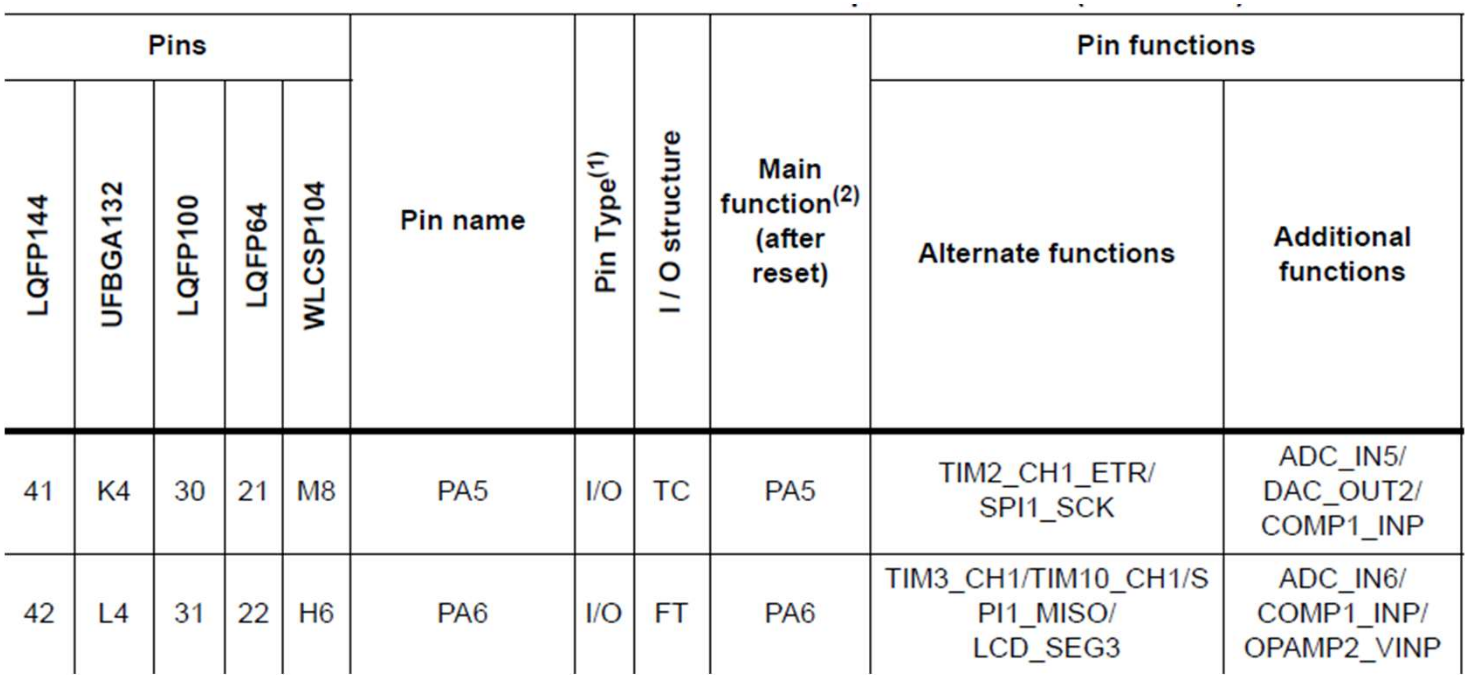
- ✓ The datasheet gives the pin out of the component

Figure 6. STM32L15xRE LQFP64 pinout



- ✓ Most pins can have several functions
- ✓ Pins related to power have a unique function (VDD, VSS, VDDIO…)
- ✓ Most GPIOS pins are coupled to another function i.e. PC2 (GPIO) and Timer 1 channel 2

Multiple Usage of Pin



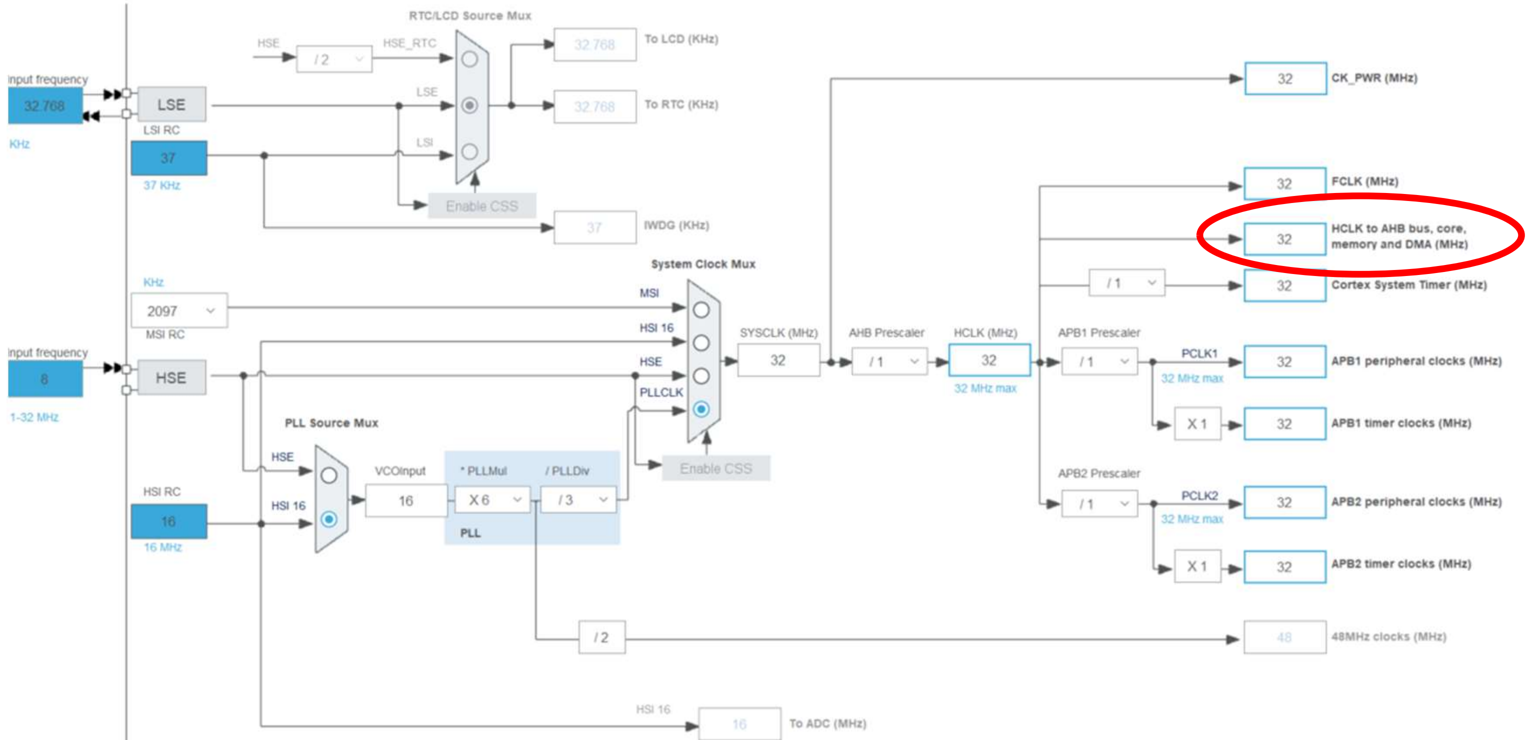
Pins					Pin name	Pin Type[1]	I / O structure	Main function[2] (after reset)	Pin functions	
LQFP144	UFBGA132	LQFP100	LQFP64	WLCSP104					Alternate functions	Additional functions
41	K4	30	21	M8	PA5	I/O	TC	PA5	TIM2_CH1_ETR/ SPI1_SCK	ADC_IN5/ DAC_OUT2/ COMP1_INP
42	L4	31	22	H6	PA6	I/O	FT	PA6	TIM3_CH1/TIM10_CH1/SPI1_MISO/ LCD_SEG3	ADC_IN6/ COMP1_INP/ OPAMP2_VINP

IO 5 of port 1 can be used as GPIO, or channel 1 Timer 2, or Clock of peripheral SPI1 or input 5 of ADC…

Port

- ✓ Each pin is connected to a port.
- ✓ A port can be connected to up to 16 pins.
- ✓ A port is recognized by a letter and the number is the pin number of the port
- ✓ PC2 means: Input/Output 2 of the port C
- ✓ A port is a peripheral

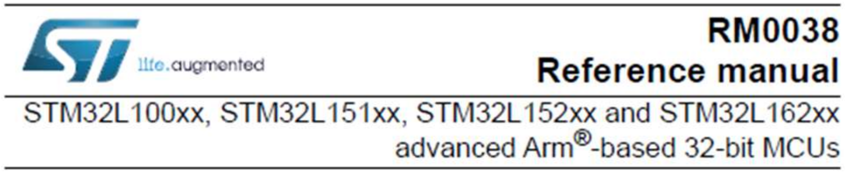
Clock



Each Peripheral must be clocked. 3 main buses are inside the STM32l152; AHB, APB1 and APB2. GPIOs are clocked by the AHB bus

Basic Questions for a Pin

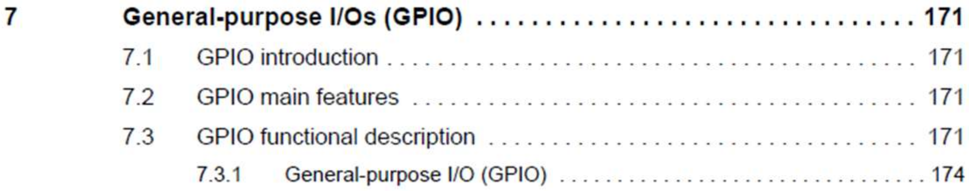
- ✓ Receive or transmit data?.
- ✓ Where to store data in receiving or transmitting mode?
- ✓ Physical configuration
- ✓ All the information is available in the Reference Manual



RM0038

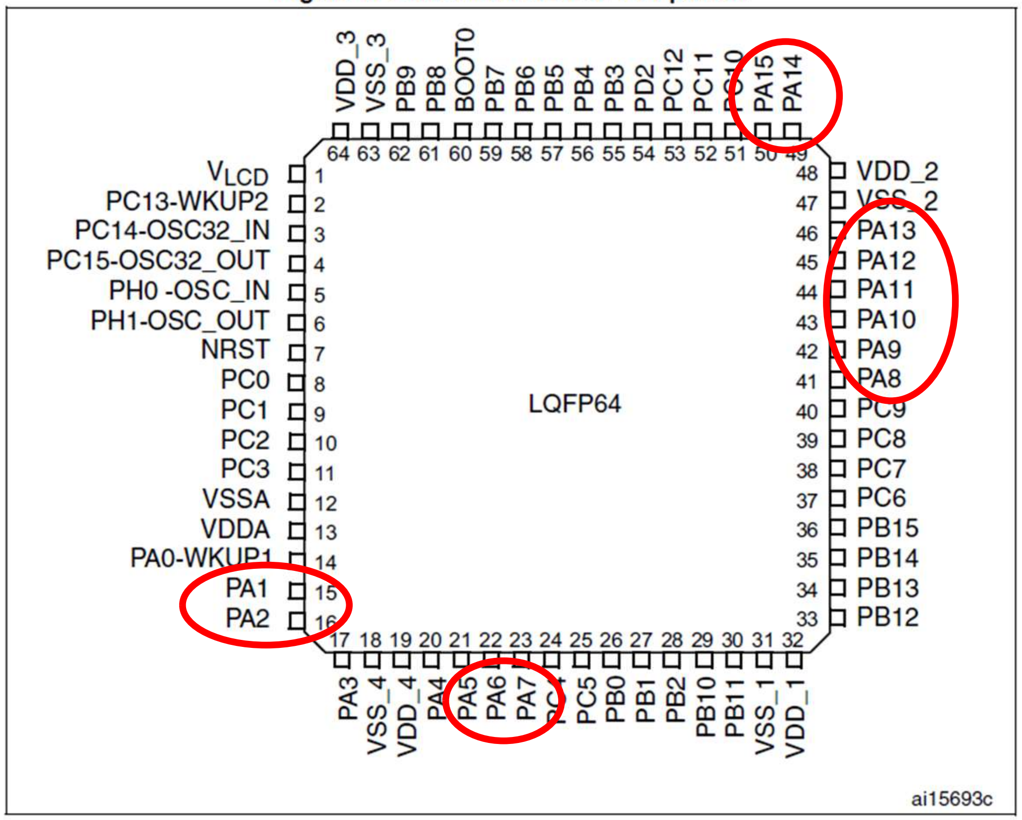
Reference manual

STM32L100xx, STM32L151xx, STM32L152xx and STM32L162xx advanced Arm®-based 32-bit MCUs



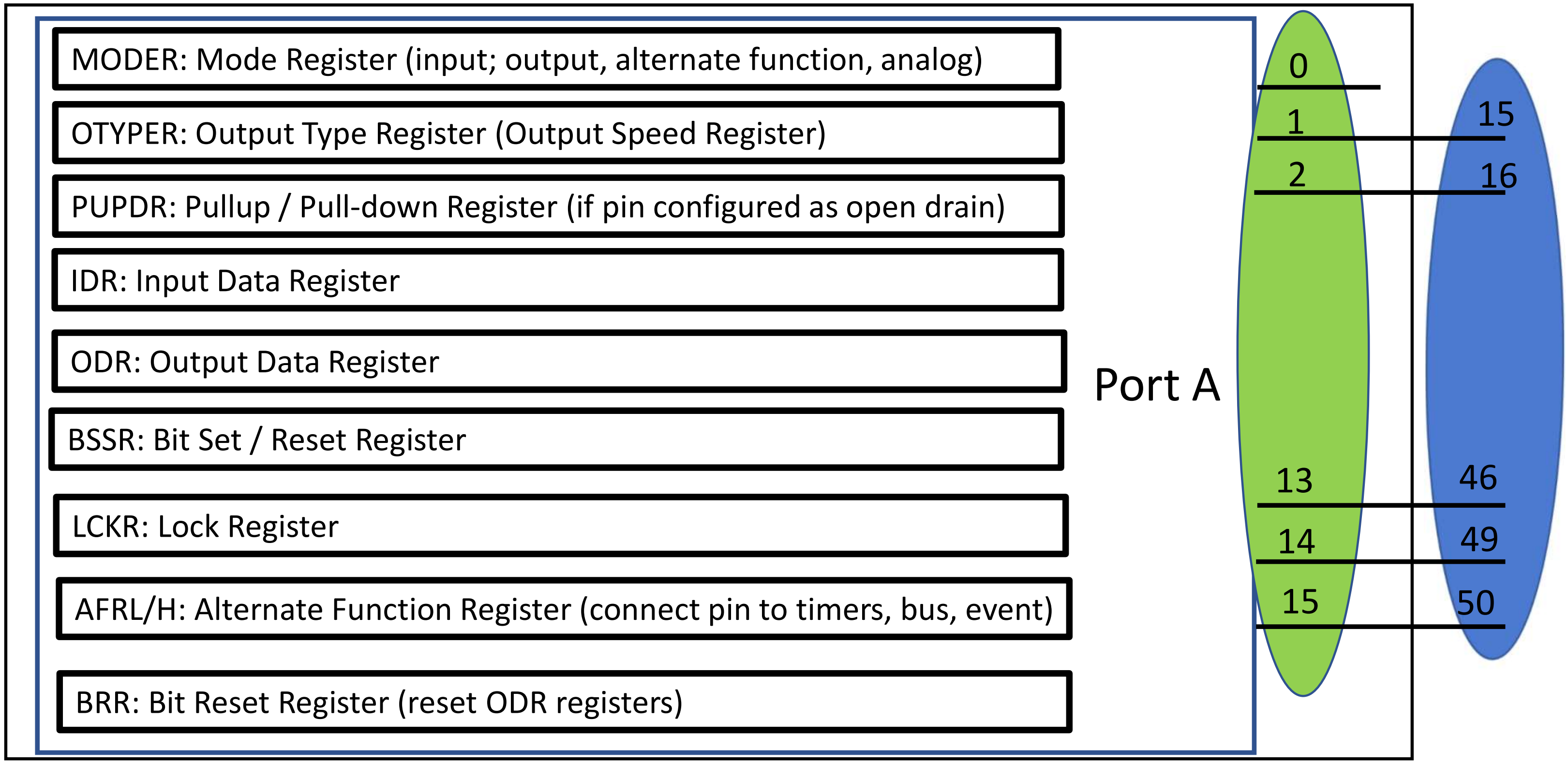
STM32L152RE Pin Out

Figure 6. STM32L15xRE LQFP64 pinout



- ✓ A port is referenced by a letter: A, B, C…
- ✓ Each port has up to 16 IOs (Input/Output)
- ✓ Each IO is connected to a pin

Port IO’s and Pin Numbers



MODER: Mode Register (input; output, alternate function, analog)

OTYPER: Output Type Register (Output Speed Register)

PUPDR: Pullup / Pull-down Register (if pin configured as open drain)

IDR: Input Data Register

ODR: Output Data Register

BSSR: Bit Set / Reset Register

LCKR: Lock Register

AFRL/H: Alternate Function Register (connect pin to timers, bus, event)

BRR: Bit Reset Register (reset ODR registers)

Each port has 10* Registers

*AFRL/H are 2 registers

Port Register Roles

MODER: Mode Register (input; output, alternate function, analog)

OTYPER: Output Type Register (Output Speed Register)

PUPDR: Pullup / Pull-down register (if pin configured as open drain)

IDR: Input Data Register

ODR: Output Data Register

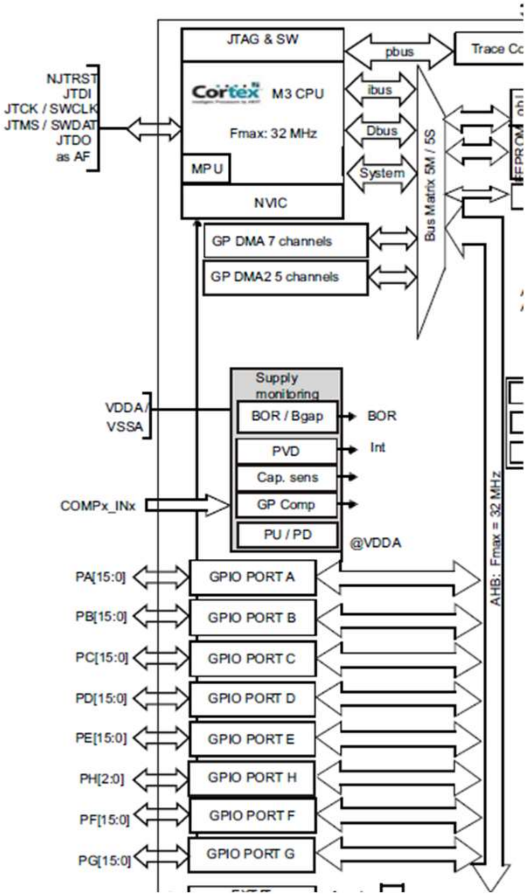
BSSR: Bit Set / Reset Register

LCKR: Lock Register

AFRL/H: Alternate Function Register (connect pin to timers, bus, event)

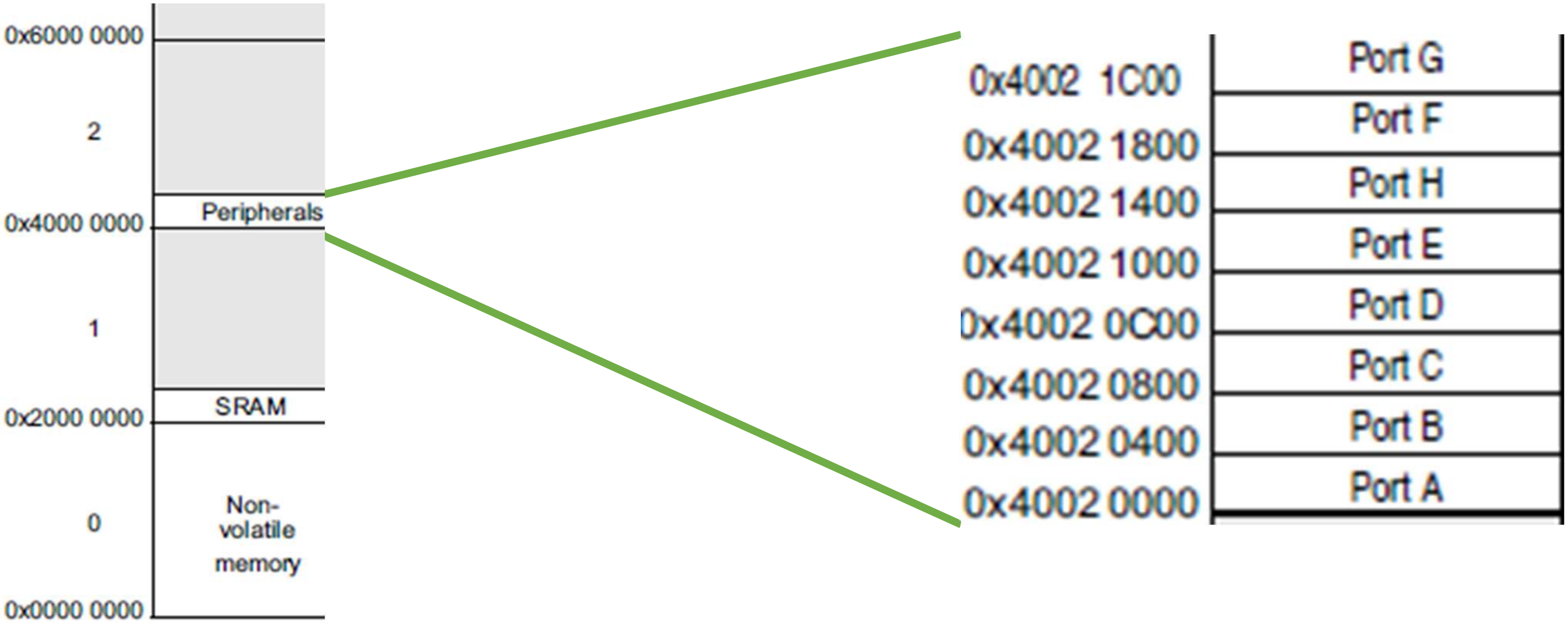
BRR: Bit Reset Register (reset ODR registers)

Ports in STM32L152



- ✓ STM32L152 has 8 GPIO ports.
- ✓ Each port has up of 16 IOs
- ✓ The number of GPIOs is limited by the package (pin numbers) and not necessarily the silicon
- ✓ STM32L152RE is a 64 pins LQFP package (Low Profile Quad Flat Package)
  - 3 ports
    - Port A : 15 IOs
    - Port B : 13 IOs
    - Port C : 11 IOs

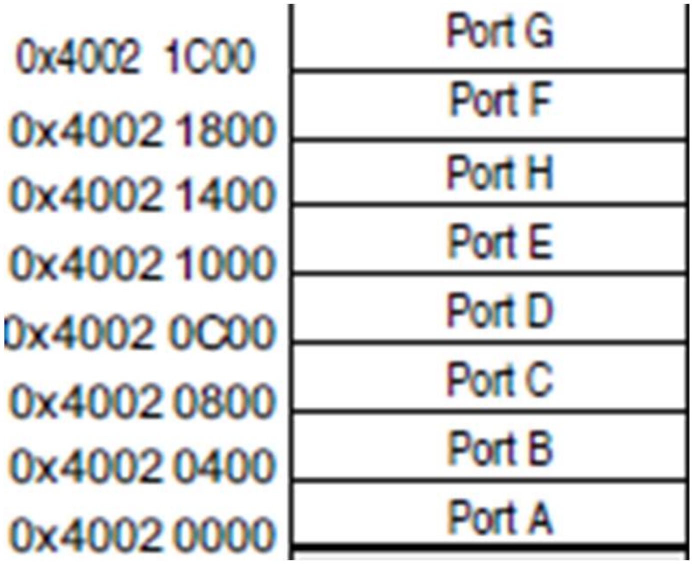
GPIOs Memory Mapping



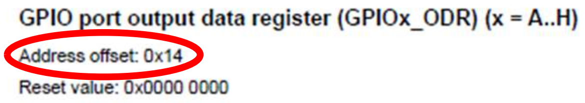
0x4002 1C00	Port G
0x4002 1800	Port F
0x4002 1400	Port H
0x4002 1000	Port E
0x4002 0C00	Port D
0x4002 0800	Port C
0x4002 0400	Port B
0x4002 0000	Port A

How To Calculate Addresses

Base Address given in Datasheet + Offset given in Reference Manual = Register Address



0x4002 1C00	Port G
0x4002 1800	Port F
0x4002 1400	Port H
0x4002 1000	Port E
0x4002 0C00	Port D
0x4002 0800	Port C
0x4002 0400	Port B
0x4002 0000	Port A



GPIO port output data register (GPIOx_ODR) (x = A..H)

Address offset: 0x14

Reset value: 0x0000 0000

0x4002 0814

Example of ODR register in port C

Port Addresses

Adresses Port A

GPIOA_MODER:	0x4002 0000
GPIOA_OTYPER:	0x4002 0004
GPIOA_OSPEEDR:	0x4002 0008
GPIOA_PUPDR:	0x4002 000C
GPIOA_IDR:	0x4002 0010
GPIOA_ODR :	0x4002 0014
GPIOA_BSRR :	0x4002 0018
GPIOA_LCKR :	0x4002 001C
GPIOA_AFRL :	0x4002 0020
GPIOA_AFRH :	0x4002 0024
GPIOA_BRR :	0x4002 0028

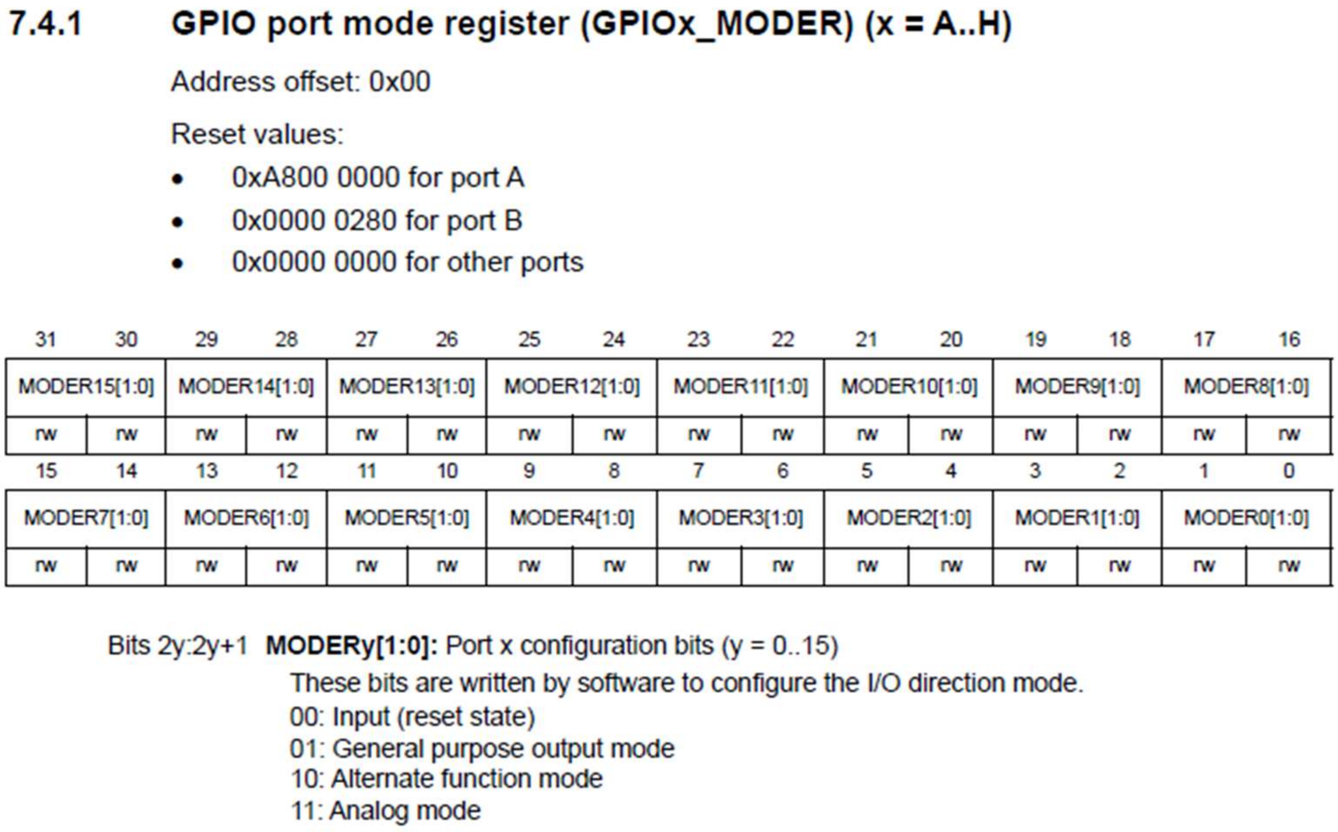
Adresses Port B

GPIOB_MODER:	0x4002 0400
GPIOB_OTYPER:	0x4002 0404
GPIOB_OSPEEDR:	0x4002 0408
GPIOB_PUPDR:	0x4002 040C
GPIOB_IDR:	0x4002 0410
GPIOB_ODR :	0x4002 0414
GPIOB_BSRR :	0x4002 0418
GPIOB_LCKR :	0x4002 041C
GPIOB_AFRL :	0x4002 0420
GPIOB_AFRH :	0x4002 0424
GPIOB_BRR :	0x4002 0428

Adresses Port C

GPIOC_MODER:	0x4002 0800
GPIOC_OTYPER:	0x4002 0804
GPIOC_OSPEEDR:	0x4002 0808
GPIOC_PUPDR:	0x4002 080C
GPIOC_IDR:	0x4002 0810
GPIOC_ODR :	0x4002 0814
GPIOC_BSRR :	0x4002 0818
GPIOC_LCKR :	0x4002 081C
GPIOC_AFRL :	0x4002 0820
GPIOC_AFRH :	0x4002 0824
GPIOC_BRR :	0x4002 0828

GPIO port Mode Register



7.4.1 GPIO port mode register (GPIOx_MODER) (x = A..H)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

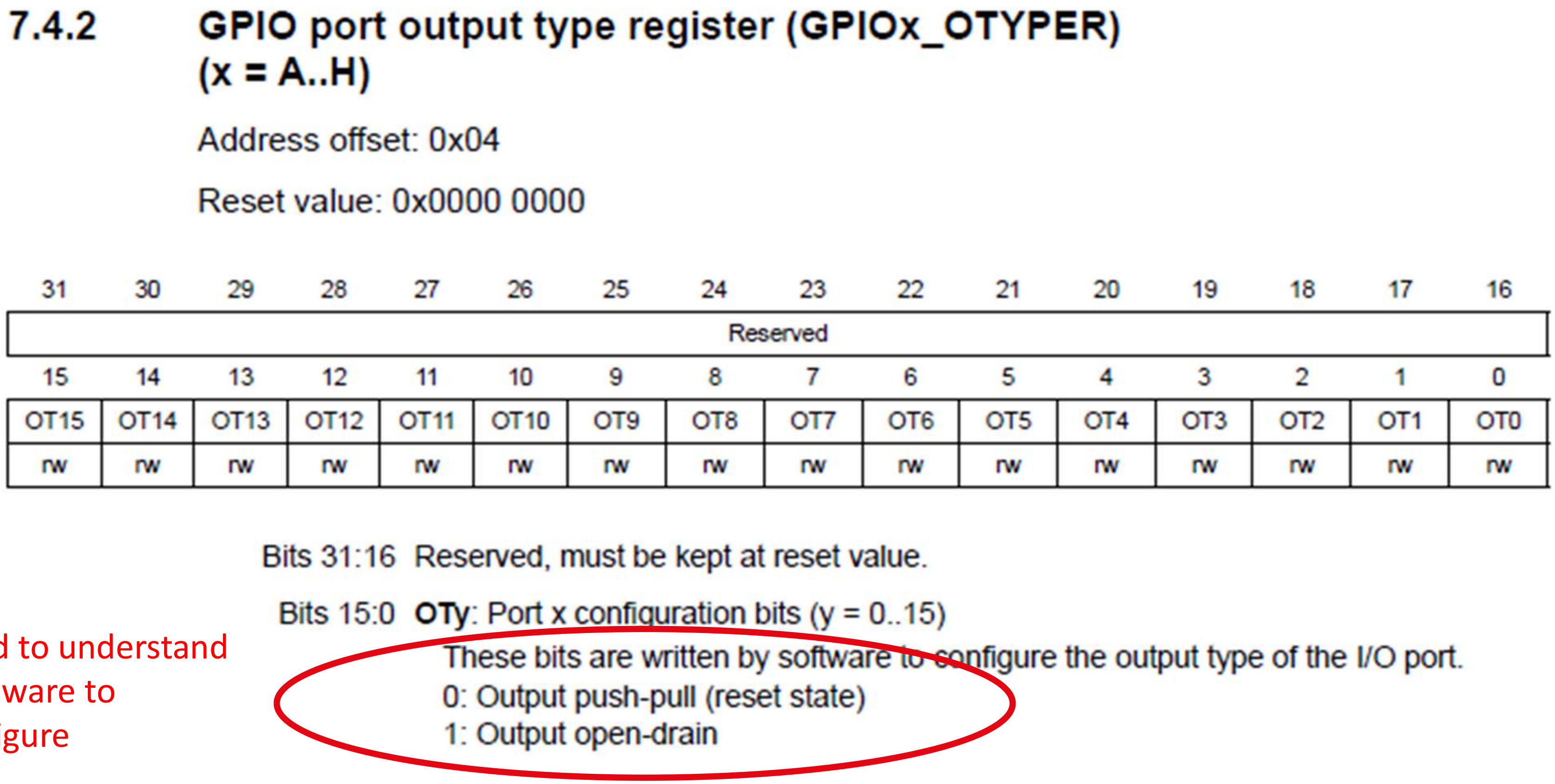
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

GPIO port Output Type Register



7.4.2 GPIO port output type register (GPIOx_OTYPER) (x = A..H)

Address offset: 0x04

Reset value: 0x0000 0000

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
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 **OTy**: Port x configuration bits (y = 0..15)

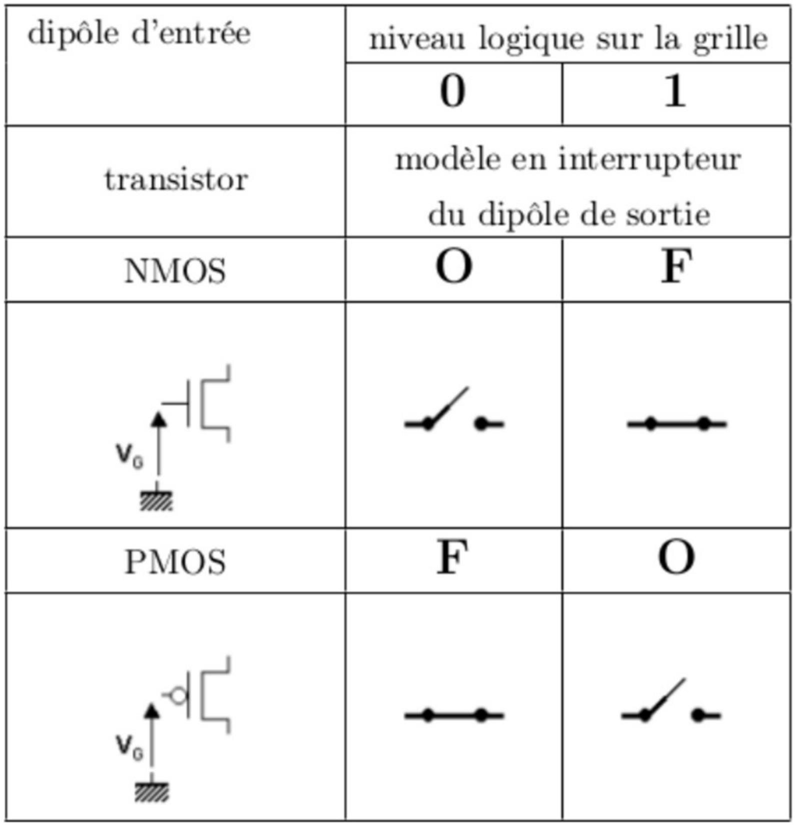
These bits are written by software to configure the output type of the I/O port.

0: Output push-pull (reset state)

1: Output open-drain

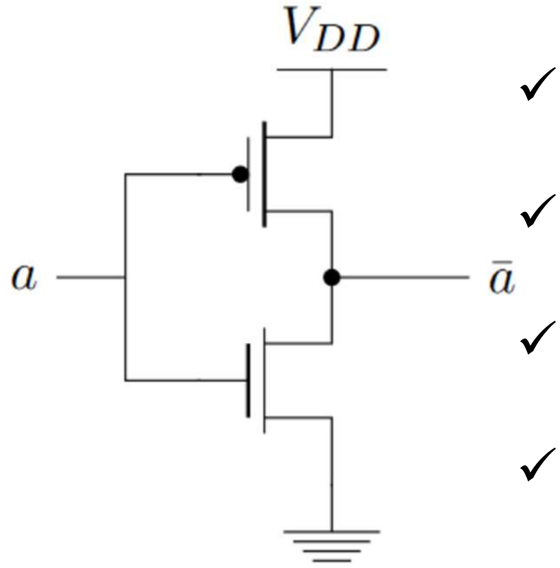
Need to understand Hardware to configure

MOS Basics Summary



dipôle d'entrée	niveau logique sur la grille	
	0	1
transistor	modèle en interrupteur du dipôle de sortie	
NMOS	O	F
PMOS	F	O

Push-Pull Configuration

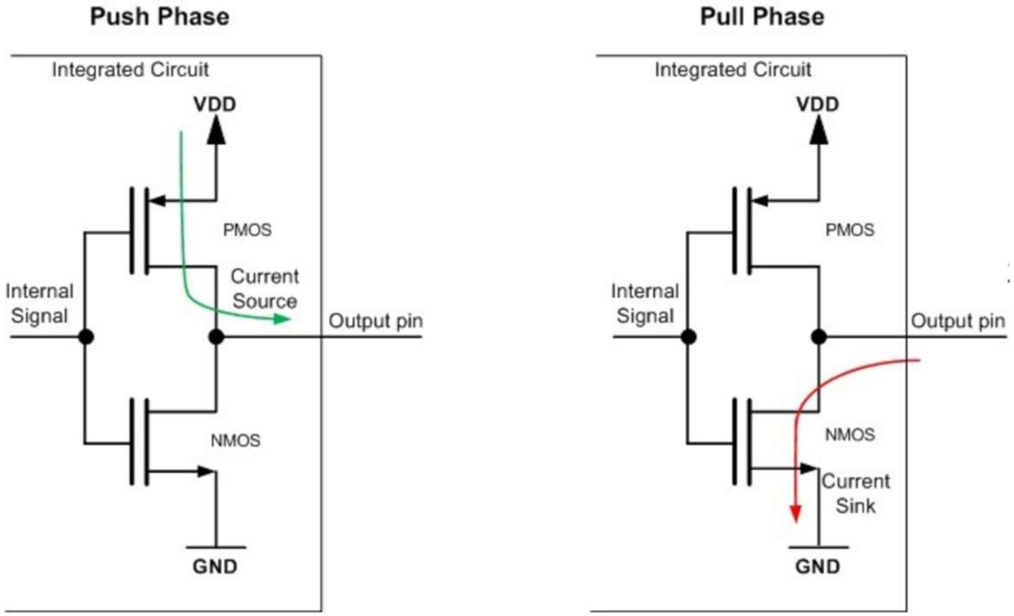


- ✓ When the Lower transistor is ON, the upper transistor is OFF
- ✓ When the Lower transistor is OFF, the upper transistor is ON
- ✓ No other external voltage can/must be applied
- ✓ Can switch fast and consumes current only during switching
- ✓ Configuration used to drive an LED or a motor

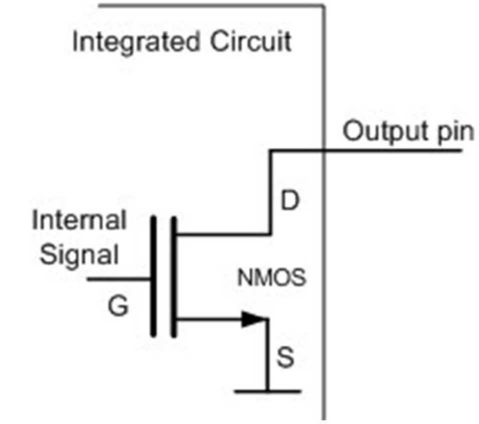
MOS Transistor Basics

La logique CMOS

Ou Complementary MOS : utilise minimum un couple canal N et canal P pour une entrée logique.



Open Drain Configuration



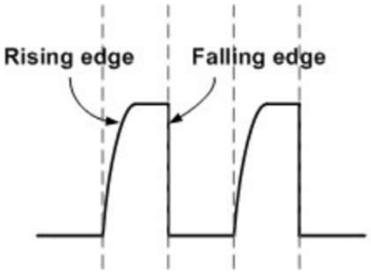
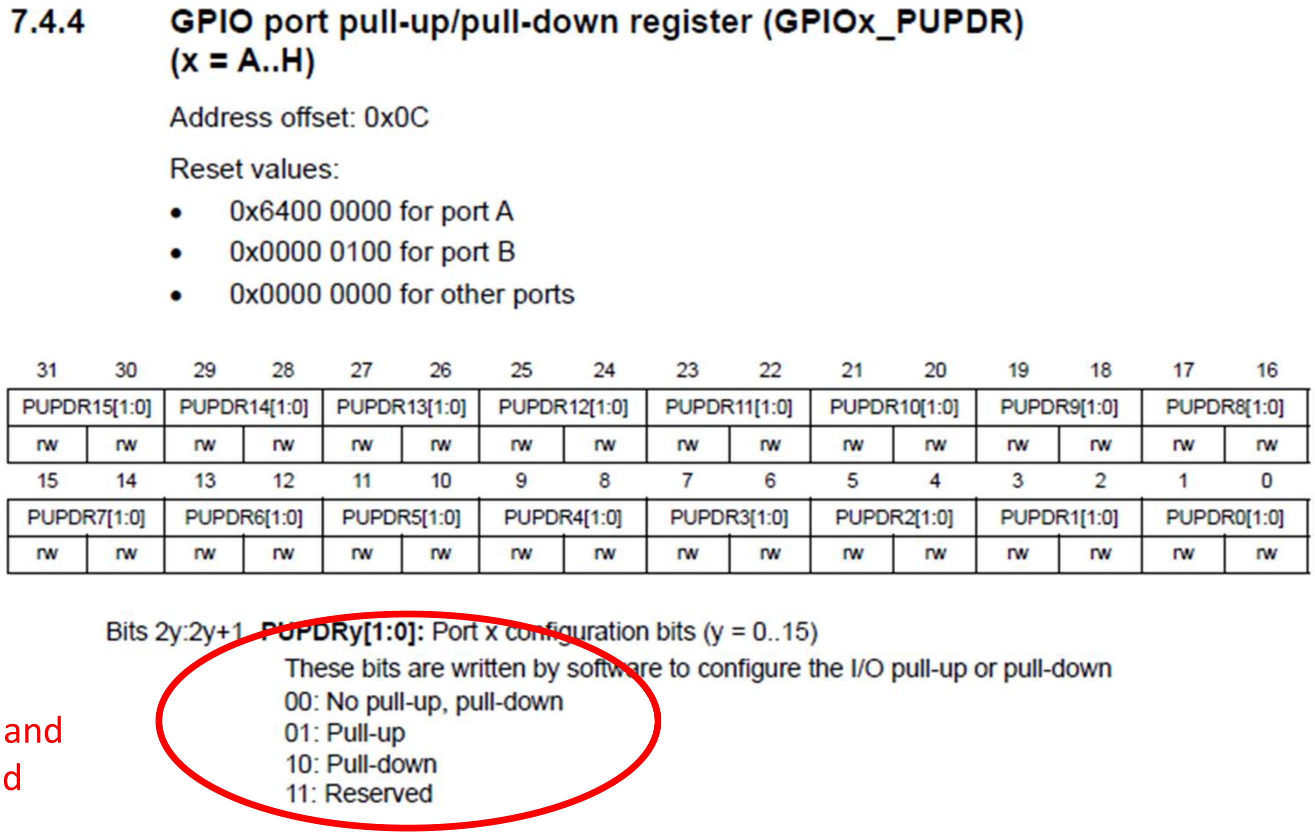


Fig. 1 Edges of square wave signal generated by an open drain output

- ✓ When the Lower transistor is ON, the output is grounded
- ✓ When the upper transistor is OFF, the output is pulled up to Vc thanks to a pull up resistor
- ✓ It is possible to connect various output pins to create an ‘OR’ structure
- ✓ Current consumption through pull up resistor when output is grounded
- ✓ Switch is low

GPIO port Pull-up/Pull-down Register



7.4.4 GPIO port pull-up/pull-down register (GPIOx_PUPDR) (x = A..H)

Address offset: 0x0C

Reset values:

- 0x6400 0000 for port A
- 0x0000 0100 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **PUPDRy[1:0]:** Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O pull-up or pull-down

00: No pull-up, pull-down

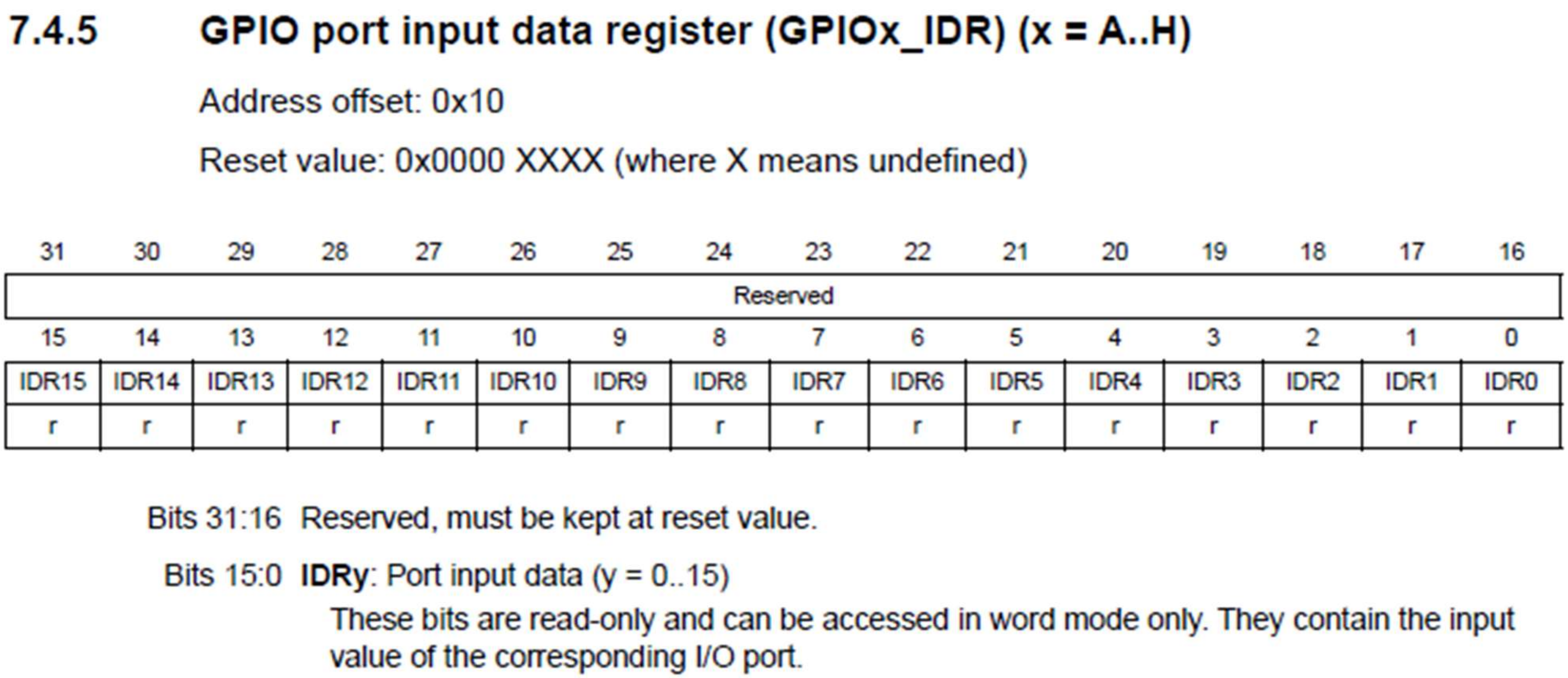
01: Pull-up

10: Pull-down

11: Reserved

Depends on Hardware. In embedded programming, understanding of software and hardware isssues is required

GPIO Input Data Register



7.4.5 GPIO port input data register (GPIOx_IDR) (x = A..H)

Address offset: 0x10

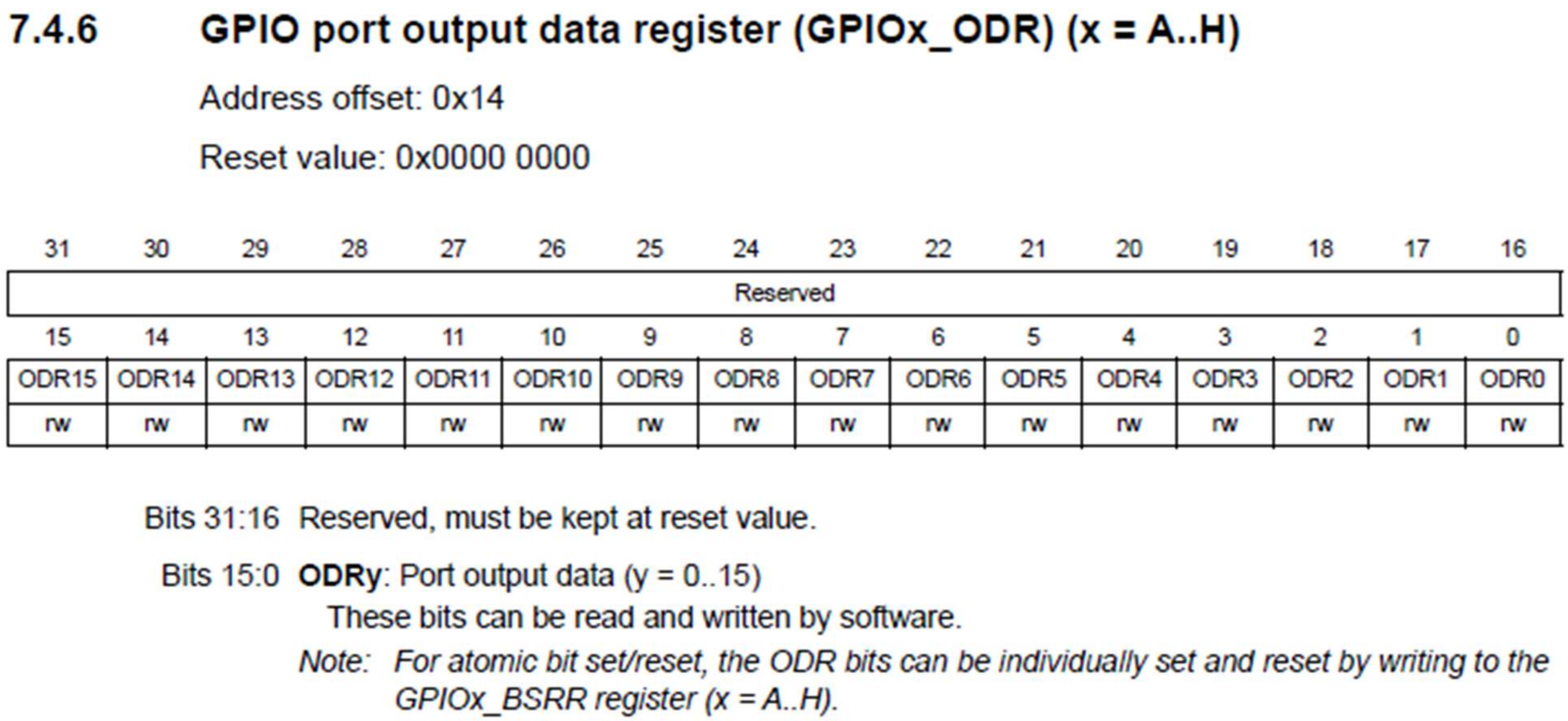
Reset value: 0x0000 XXXX (where X means undefined)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

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

Bits 15:0 **IDRy**: Port input data (y = 0..15)
These bits are read-only and can be accessed in word mode only. They contain the input value of the corresponding I/O port.

GPIO Output Data Register



7.4.6 GPIO port output data register (GPIOx_ODR) (x = A..H)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

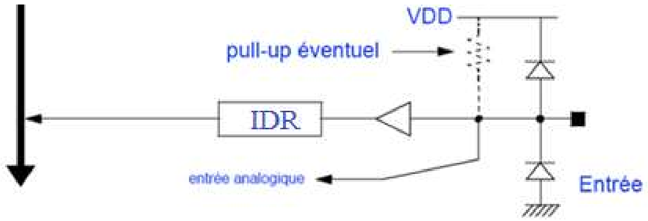
Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy**: Port output data (y = 0..15)

These bits can be read and written by software.

Note: For atomic bit set/reset, the ODR bits can be individually set and reset by writing to the GPIOx_BSRR register (x = A..H).

High Impedance



Imagine the input impedance was ridiculously low, say 1Ω. You'd have to somehow supply 5A to bring the input up to 5V. 5A is a gigantic amount of electricity. You could use it to keep nice and warm but your batteries wouldn't last very long.

If the input impedance is too high, say 100MΩ, then you'd need only 50nA to get 5V. This would make the input far too sensitive - it would pick up enough airborne electromagnetic noise to reach 5V then go back to 0V in an erratic and confusing manner.

A high impedance is in the MΩ range as a trade-off

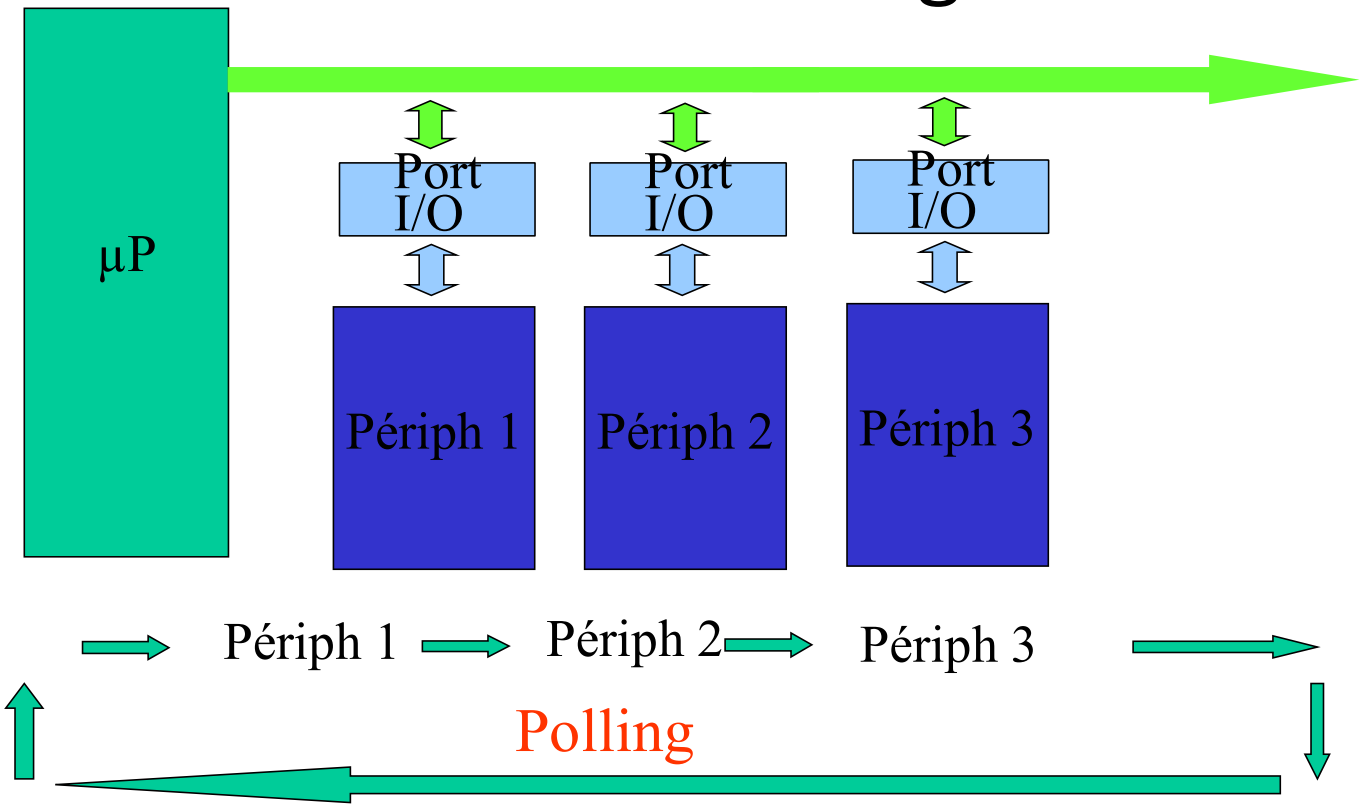
How to Read Input Data

A microcontroller communicates with the external world through its pins. When receiving information, the issue is :

How to know when the input register must be read? 2 solutions:

- ✓ Polling (or continuous scanning)
  - A loop checks regularly if new data is available
- ✓ Interrupt
  - A signal is sent to the CPU to let him know that new data is available

Polling



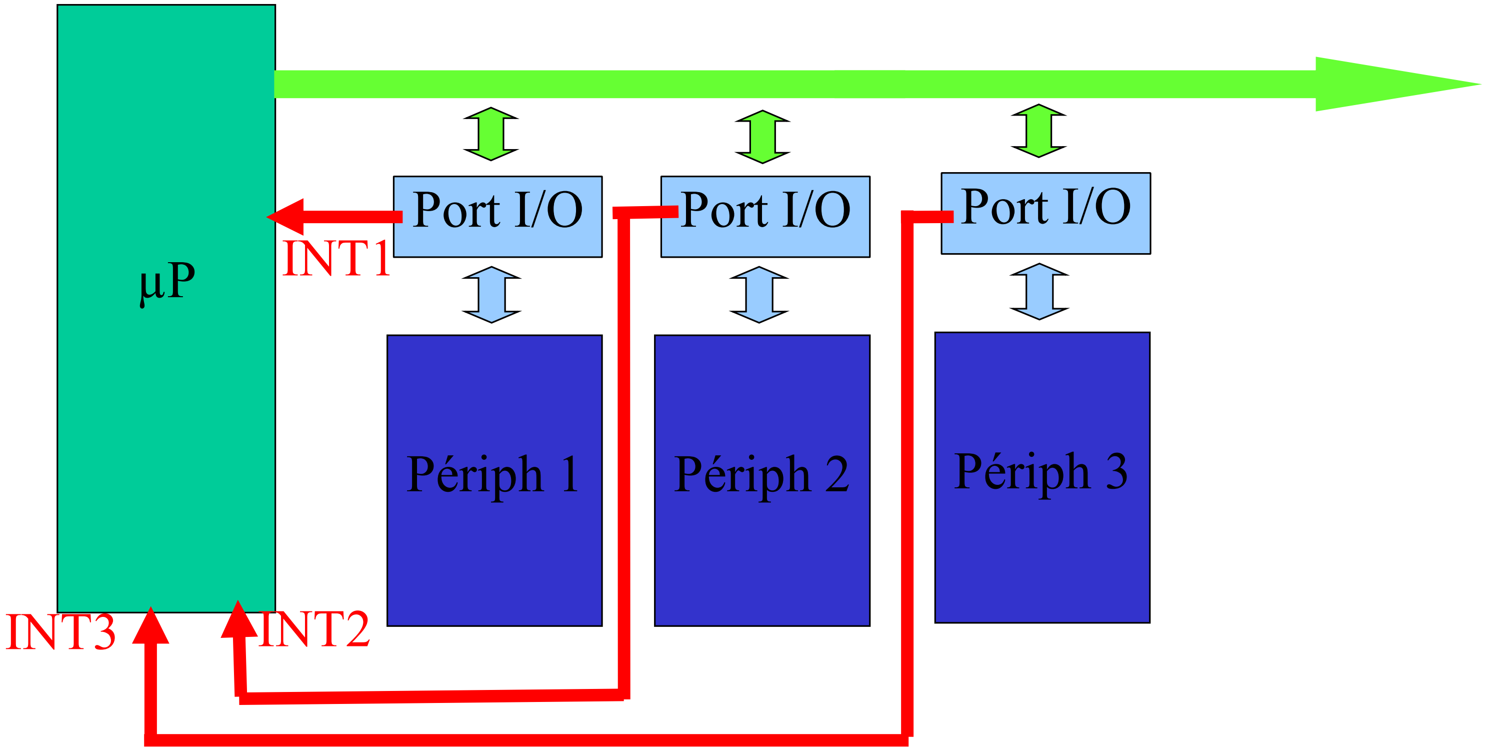
Need to program polling

Issues with Polling

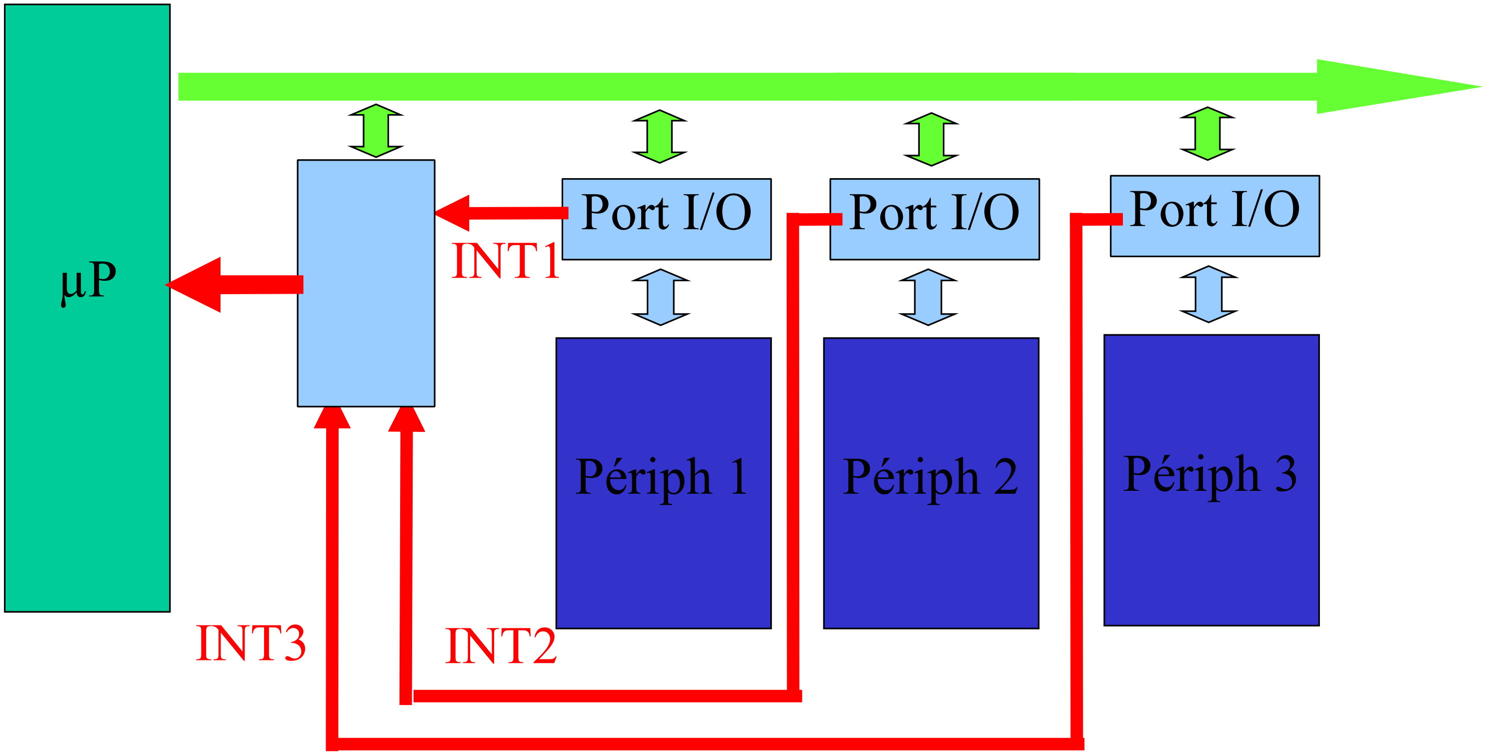
- ✓ Polling wastes a lot of ressources.
- ✓ The µController might be woken up regularly for nothing → waste of energy
- ✓ The µController takes time to monitor the IDR and could have more important routine to process
- ✓ Do you have the most updated data?

In this case Interrupt might be a preferred solution

Interrupts Principle



Interrupts Principle



The number of available interrupts in a µController is limited and interrupts can be grouped

Interrupts Vs Polling

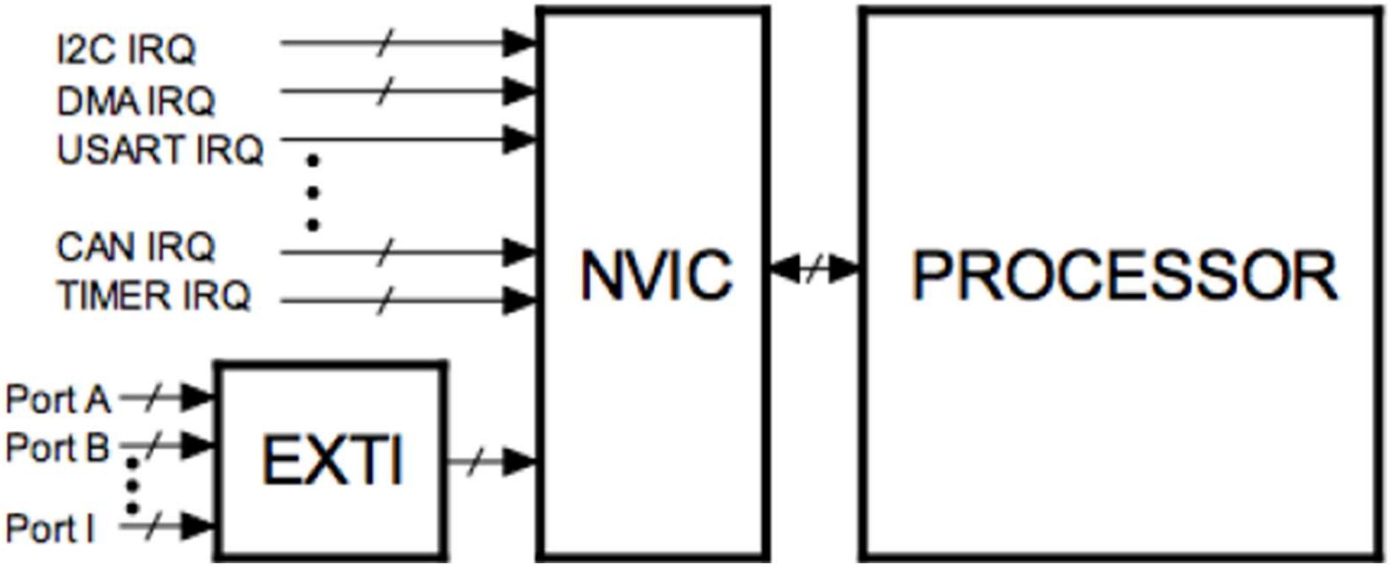
Interrupts might be a good choice when:

- Events occur infrequently
- There are long intervals between two event
- The main program has nothing else to do and the MCU can be put into sleep mode
- There are short impulses which could be missed otherwise
- Event are generated by hardware i.e; there no bouncing effects or spikes

Polling might be preferred when:

- The operator is human
- No precise timing is necessary
- Impulses are long
- The signla is noisy
- There is something else to do in the main program but not too much
- Required by the industry
- Complexity with interrupt priorities

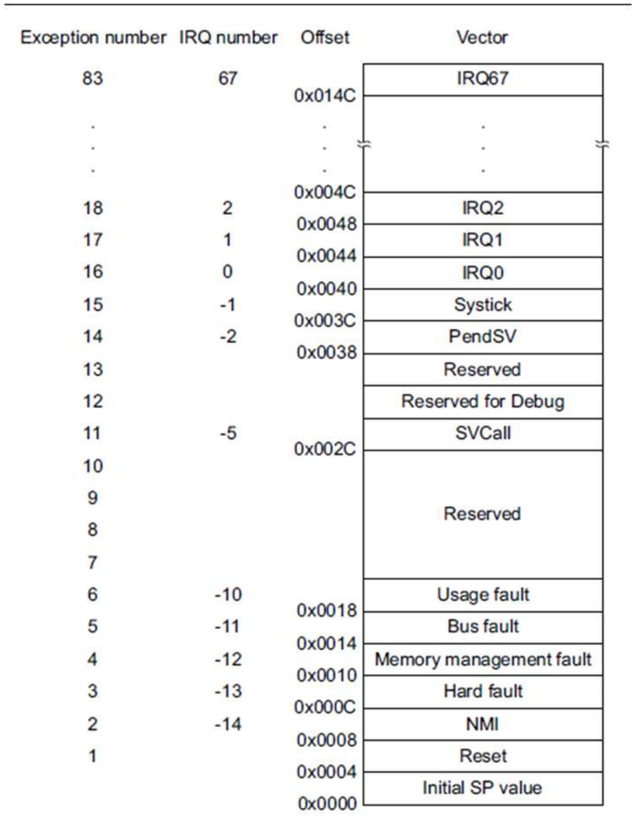
Processing of Interrupt Request



Interrupts are managed by the NVIC (Nested Vector Interrupt Controller)

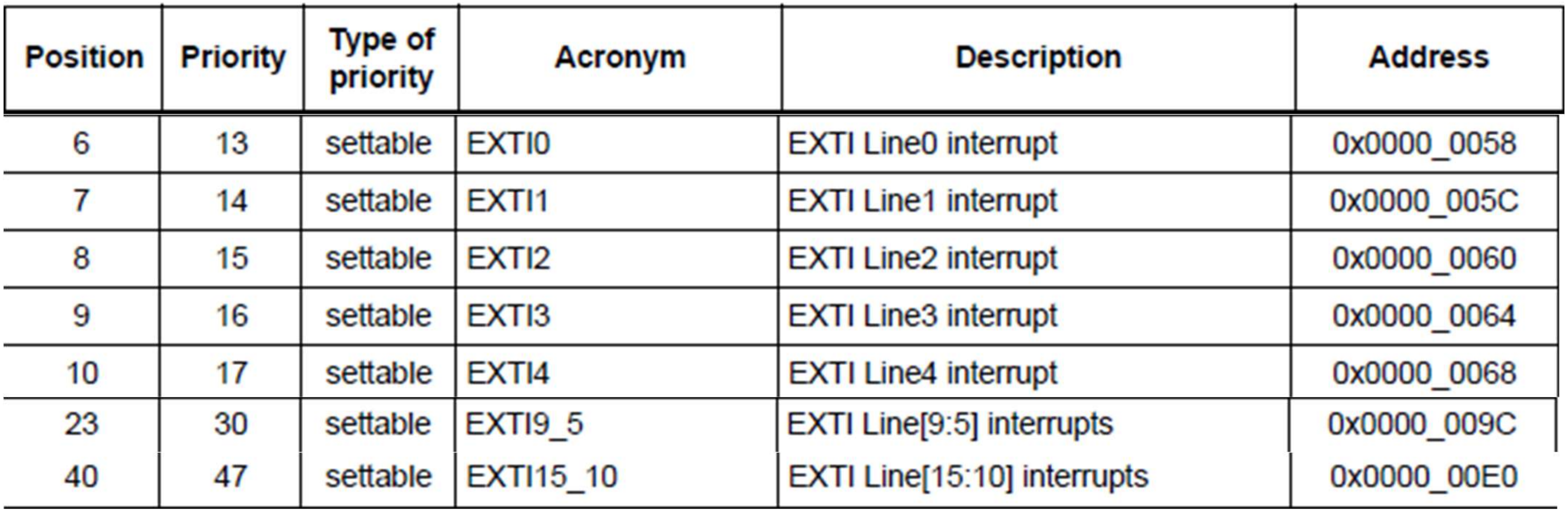
Exception Vector Table

- ✓ All the exceptions (including interrupts) are organized as a matrix called Vector Table.
- ✓ Each exception has a fixed address
- ✓ When a microcontroller powers up, the exception reset at address 0x4 is called
- ✓ The number of interrupts depends on the microcontroller (Cortex M can have up to 256 exceptions)



Exception number	IRQ number	Offset	Vector
83	67	0x014C	IRQ67
. . .		. . . 0x004C	. . .
18	2	0x0048	IRQ2
17	1	0x0044	IRQ1
16	0	0x0040	IRQ0
15	-1	0x003C	Systick
14	-2	0x0038	PendSV
13			Reserved
12			Reserved for Debug
11	-5	0x002C	SVCall
10			Reserved
9			
8			
7			
6	-10	0x0018	Usage fault
5	-11	0x0014	Bus fault
4	-12	0x0010	Memory management fault
3	-13	0x000C	Hard fault
2	-14	0x0008	NMI
1		0x0004	Reset
		0x0000	Initial SP value

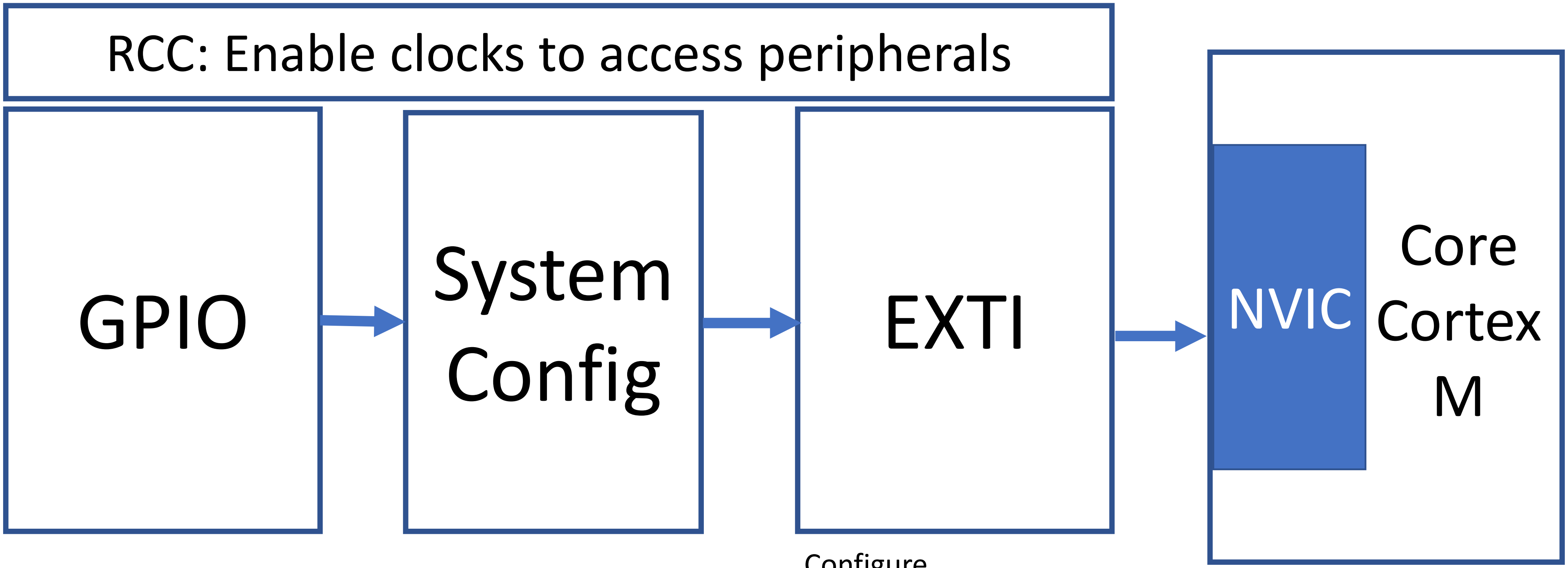
External Interrupt Memory Location



Position	Priority	Type of priority	Acronym	Description	Address
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000_0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000_0068
23	30	settable	EXTI9_5	EXTI Line[9:5] interrupts	0x0000_009C
40	47	settable	EXTI15_10	EXTI Line[15:10] interrupts	0x0000_00E0

GPIO must be connected to EXTI to generate an interrupt

Path to Configure Pin Interrupt



Configure Hardware pin

- Input Mode
- Pull up/down

Select which pin is connected to which EXTI

Configure interrup detection:

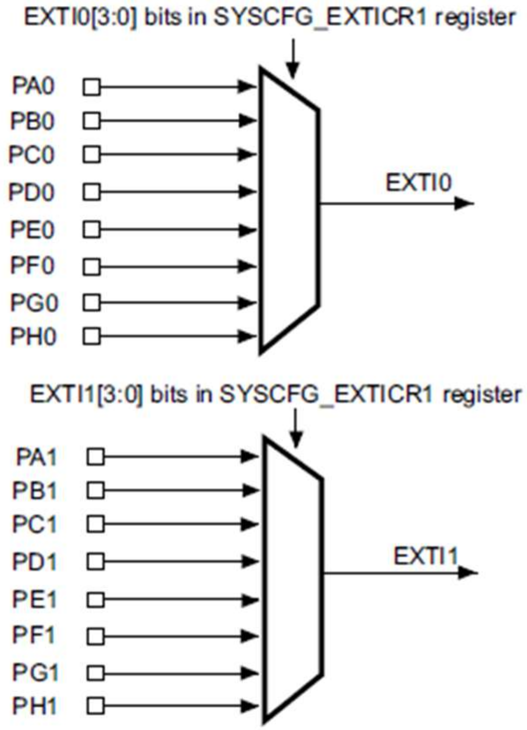
- Polarity
- Mask interrupts

Process Interrupt: Enable/disable interrupts

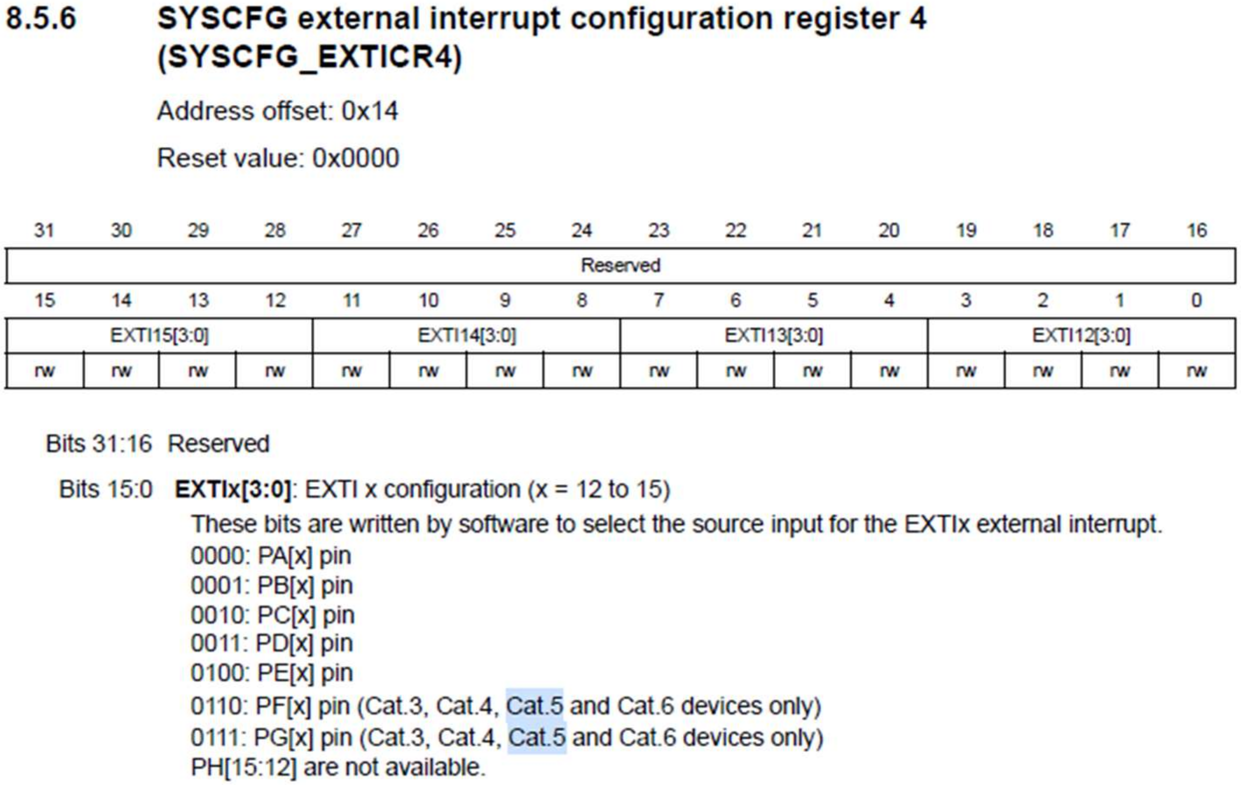
Schematic Pin to EXTI

- ✓ The reference manual shows a schematic of the External Interrupt GPIO mapping
- ✓ It shows which GPIO, SYSCFG and EXTI peripherals are involved

Figure 33. External interrupt/event GPIO mapping


EXTI0[3:0] bits in SYSCFG_EXTICR1 register
PA0
PB0
PC0
PD0
PE0
PF0
PG0
PH0
EXTI0
EXTI1[3:0] bits in SYSCFG_EXTICR1 register
PA1
PB1
PC1
PD1
PE1
PF1
PG1
PH1
EXTI1

System Config



8.5.6 SYSCFG external interrupt configuration register 4 (SYSCFG_EXTICR4)

Address offset: 0x14

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
EXTI15[3:0]				EXTI14[3:0]				EXTI13[3:0]				EXTI12[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved

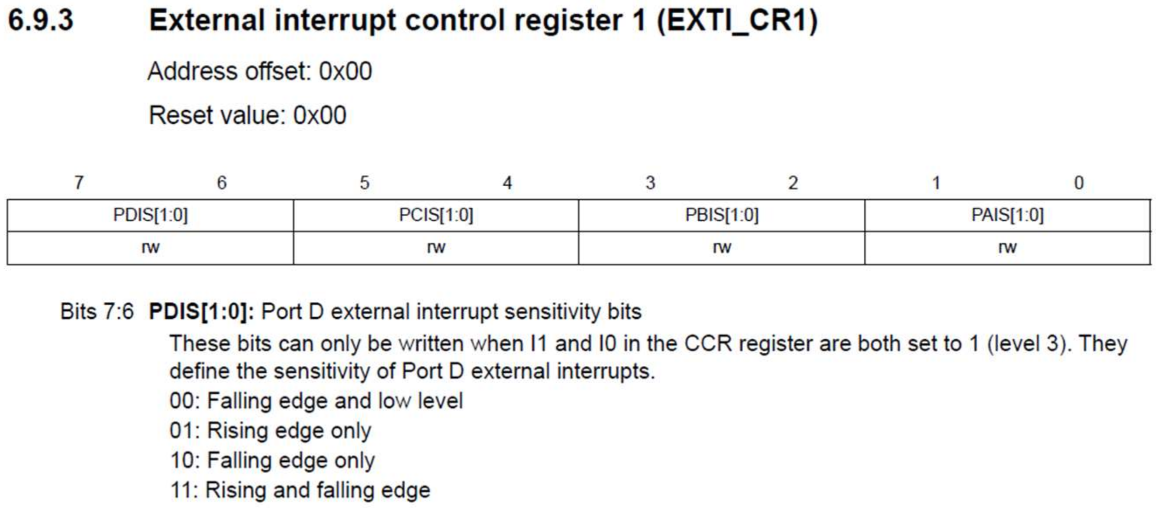
Bits 15:0 **EXTIx[3:0]**: EXTI x configuration (x = 12 to 15)

These bits are written by software to select the source input for the EXTIx external interrupt.
0000: PA[x] pin
0001: PB[x] pin
0010: PC[x] pin
0011: PD[x] pin
0100: PE[x] pin
0110: PF[x] pin (Cat.3, Cat.4, Cat.5 and Cat.6 devices only)
0111: PG[x] pin (Cat.3, Cat.4, Cat.5 and Cat.6 devices only)
PH[15:12] are not available.

To configure an interrupt on PC13, b0010 = 0x2 must be written on bits 4…7

EXTIx

- ✓ A given port number is the source of an interrupt
- ✓ All the same pin numbers trigger the same interrupt (in other words: PA0, PB0, PC0, PD0… can trigger an interrupt through EXTI0
- ✓ EXTIx configures the sensitivity level to trigger an interrupt



6.9.3 External interrupt control register 1 (EXTI_CR1)

Address offset: 0x00

Reset value: 0x00

7	6	5	4	3	2	1	0
PDIS[1:0]		PCIS[1:0]		PBIS[1:0]		PAIS[1:0]	
rw		rw		rw		rw	

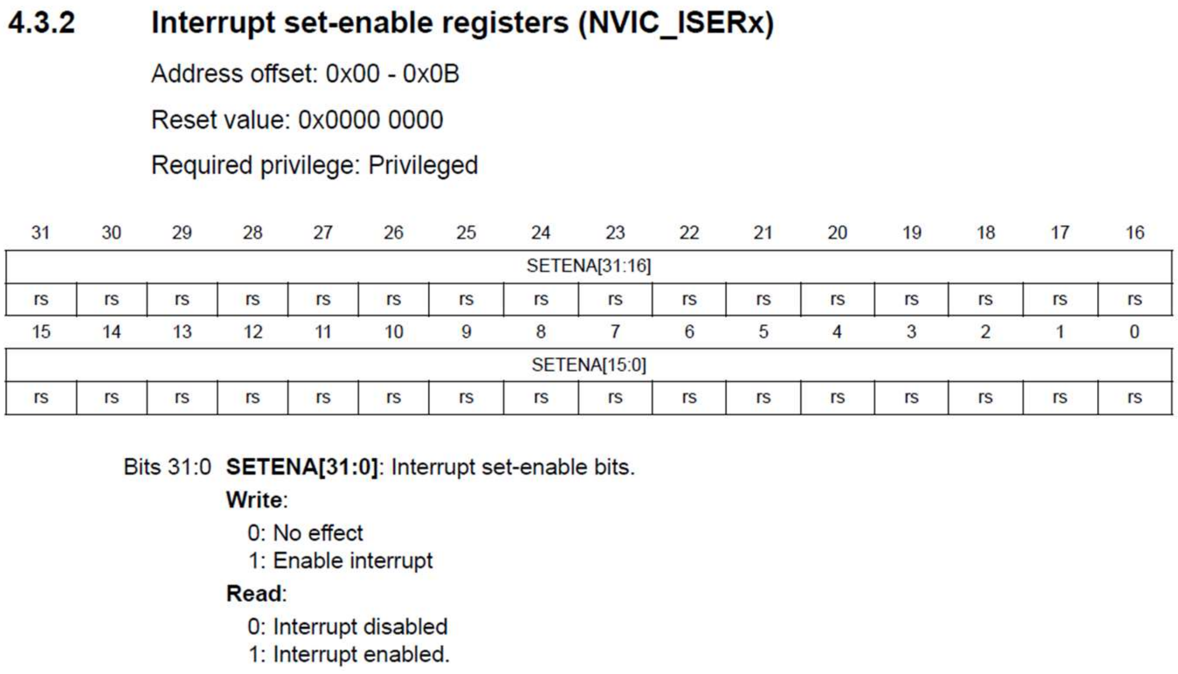
Bits 7:6 **PDIS[1:0]:** Port D external interrupt sensitivity bits

These bits can only be written when I1 and I0 in the CCR register are both set to 1 (level 3). They define the sensitivity of Port D external interrupts.

00: Falling edge and low level
01: Rising edge only
10: Falling edge only
11: Rising and falling edge

Enable Interrupt NVIC

- ✓ NVIC stands for Nested Vector Interrupt Controller
- ✓ It contains registers to enable/disable interrupts and to set priority for each interrupt
- ✓ The NVIC contains various ISER registers. The bit to set is calculated with the interrupt number



4.3.2 Interrupt set-enable registers (NVIC_ISERx)

Address offset: 0x00 - 0x0B

Reset value: 0x0000 0000

Required privilege: Privileged

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

Bits 31:0 **SETENA[31:0]**: Interrupt set-enable bits.

Write:

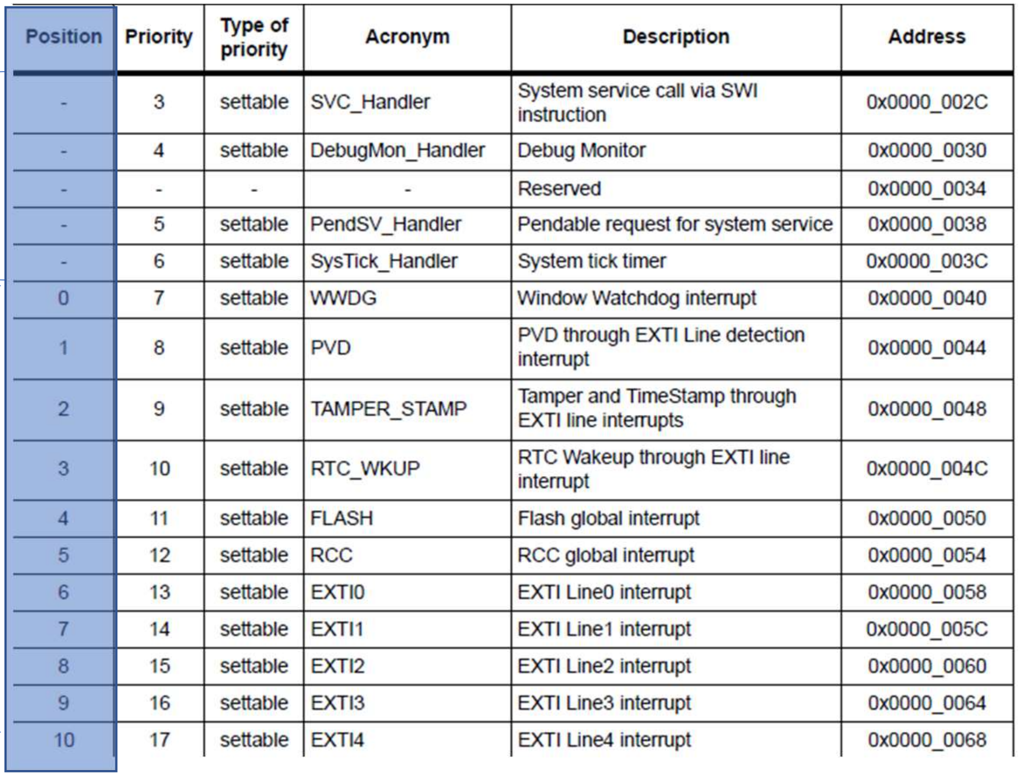
0: No effect
1: Enable interrupt

Read:

0: Interrupt disabled
1: Interrupt enabled.

Position for ISER Register

the position of the interrupt corresponds to the bit in the register ISER



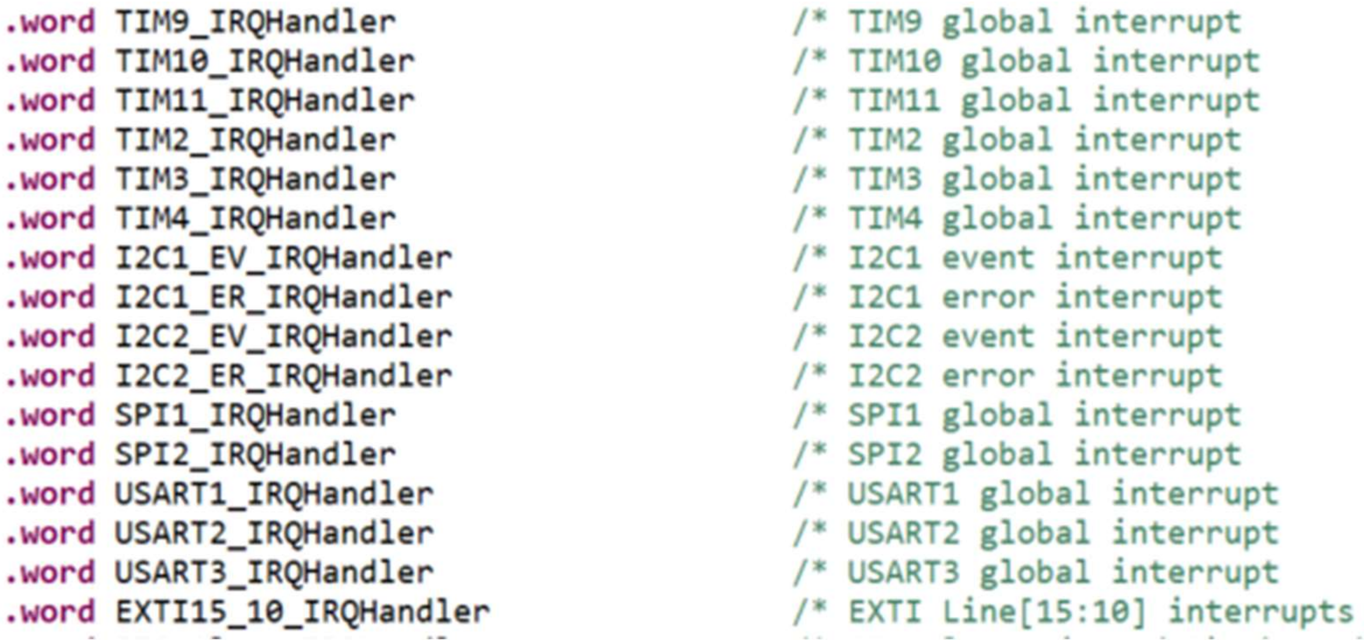
Position	Priority	Type of priority	Acronym	Description	Address
-	3	settable	SVC_Handler	System service call via SWI instruction	0x0000_002C
-	4	settable	DebugMon_Handler	Debug Monitor	0x0000_0030
-	-	-	-	Reserved	0x0000_0034
-	5	settable	PendSV_Handler	Pendable request for system service	0x0000_0038
-	6	settable	SysTick_Handler	System tick timer	0x0000_003C
0	7	settable	WWDG	Window Watchdog interrupt	0x0000_0040
1	8	settable	PVD	PVD through EXTI Line detection interrupt	0x0000_0044
2	9	settable	TAMPER_STAMP	Tamper and TimeStamp through EXTI line interrupts	0x0000_0048
3	10	settable	RTC_WKUP	RTC Wakeup through EXTI line interrupt	0x0000_004C
4	11	settable	FLASH	Flash global interrupt	0x0000_0050
5	12	settable	RCC	RCC global interrupt	0x0000_0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000_0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000_0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000_0068

Exceptions

Interrupts

List of Interrupts

- ✓ In order to be compliant with CMSIS directives from ARM and portability, the name of interrupts are fixed and cannot be changed.
- ✓ The vector table is declared at the end of the start-up file

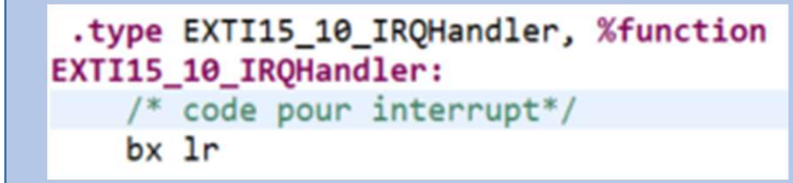


```
.word TIM9_IRQHandler                         /* TIM9 global interrupt
.word TIM10_IRQHandler                        /* TIM10 global interrupt
.word TIM11_IRQHandler                        /* TIM11 global interrupt
.word TIM2_IRQHandler                         /* TIM2 global interrupt
.word TIM3_IRQHandler                         /* TIM3 global interrupt
.word TIM4_IRQHandler                         /* TIM4 global interrupt
.word I2C1_EV_IRQHandler                      /* I2C1 event interrupt
.word I2C1_ER_IRQHandler                      /* I2C1 error interrupt
.word I2C2_EV_IRQHandler                      /* I2C2 event interrupt
.word I2C2_ER_IRQHandler                      /* I2C2 error interrupt
.word SPI1_IRQHandler                         /* SPI1 global interrupt
.word SPI2_IRQHandler                         /* SPI2 global interrupt
.word USART1_IRQHandler                       /* USART1 global interrupt
.word USART2_IRQHandler                       /* USART2 global interrupt
.word USART3_IRQHandler                       /* USART3 global interrupt
.word EXTI15_10_IRQHandler                    /* EXTI Line[15:10] interrupts
```

- ✓ An interrupt on the pin PC13 must call a routine called EXTI15-10_IRQHandler

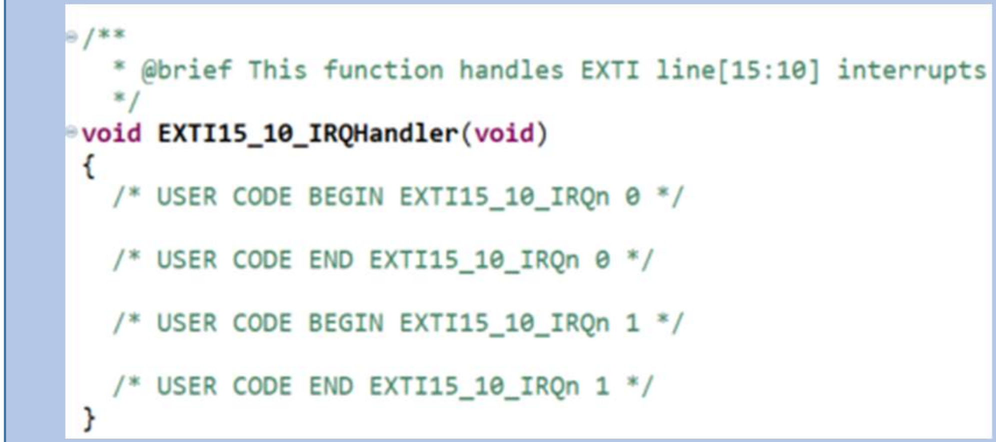
How to Program Interrupts

Code in Assembly language



```
  .type EXTI15_10_IRQHandler, %function
EXTI15_10_IRQHandler:
    /* code pour interrupt*/
    bx lr
```

Code in C



```
/**
  * @brief This function handles EXTI line[15:10] interrupts
  */
void EXTI15_10_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI15_10_IRQn 0 */

  /* USER CODE END EXTI15_10_IRQn 0 */

  /* USER CODE BEGIN EXTI15_10_IRQn 1 */

  /* USER CODE END EXTI15_10_IRQn 1 */
}
```