Embedded System and Microcomputer Principle

LAB2 General-purpose Input/Output

2024 Fall
wangq9@mail.sustech.edu.cn

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

CONTENTS

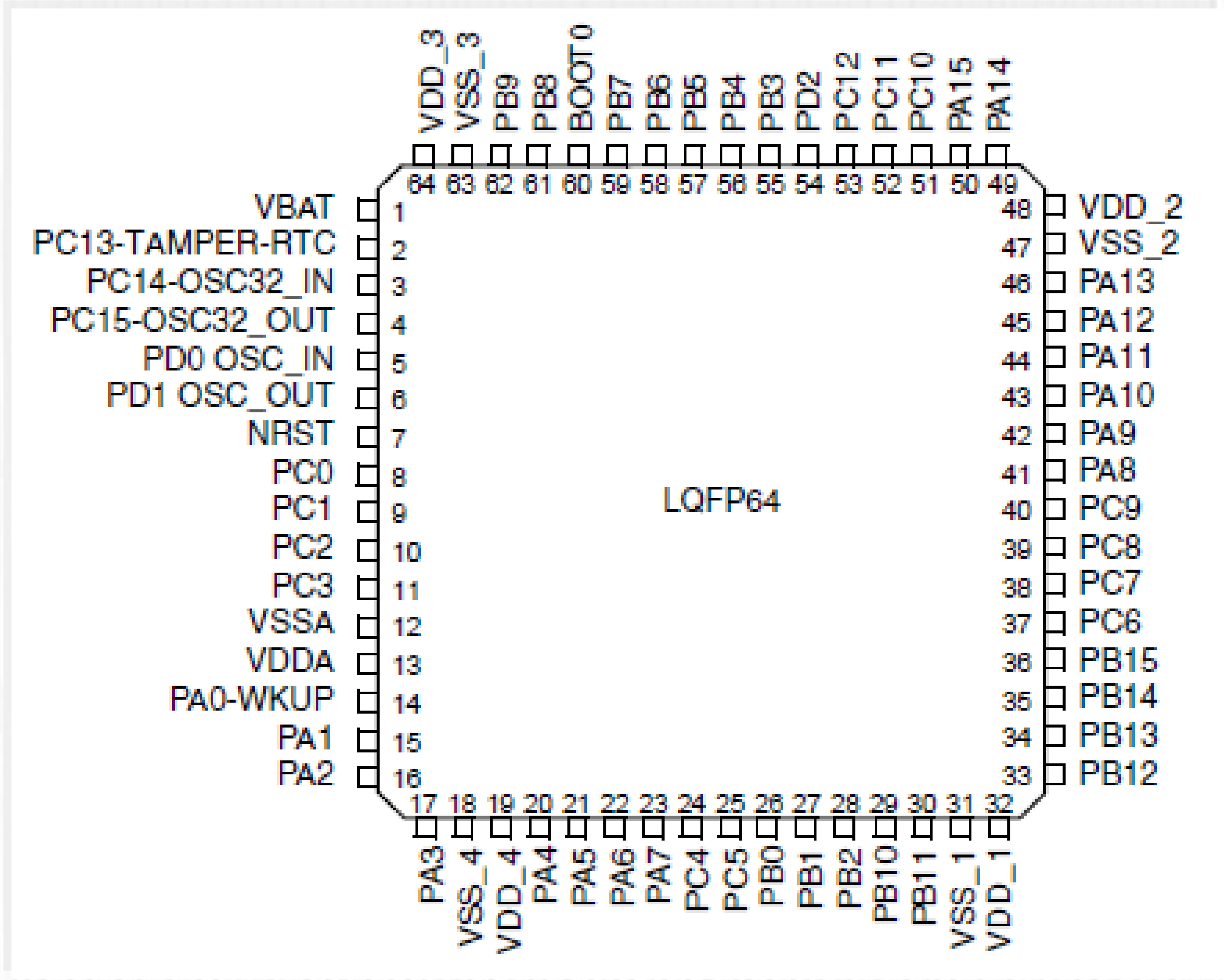
南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

01

GPIO Function Description

1. GPIO Function Description

- There are 4 groups of I/O in STM32F103RCT6
- 51 I/O ports
  - GPIOA0~A15
  - GPIOB0~B15
  - GPIOC0~C15
  - GPIOD0~D2
  - 16*3 + 3 = 51
- 47 I/O ports are available on MiniSTM32 board
  - Two crystal oscillators occupy four pins



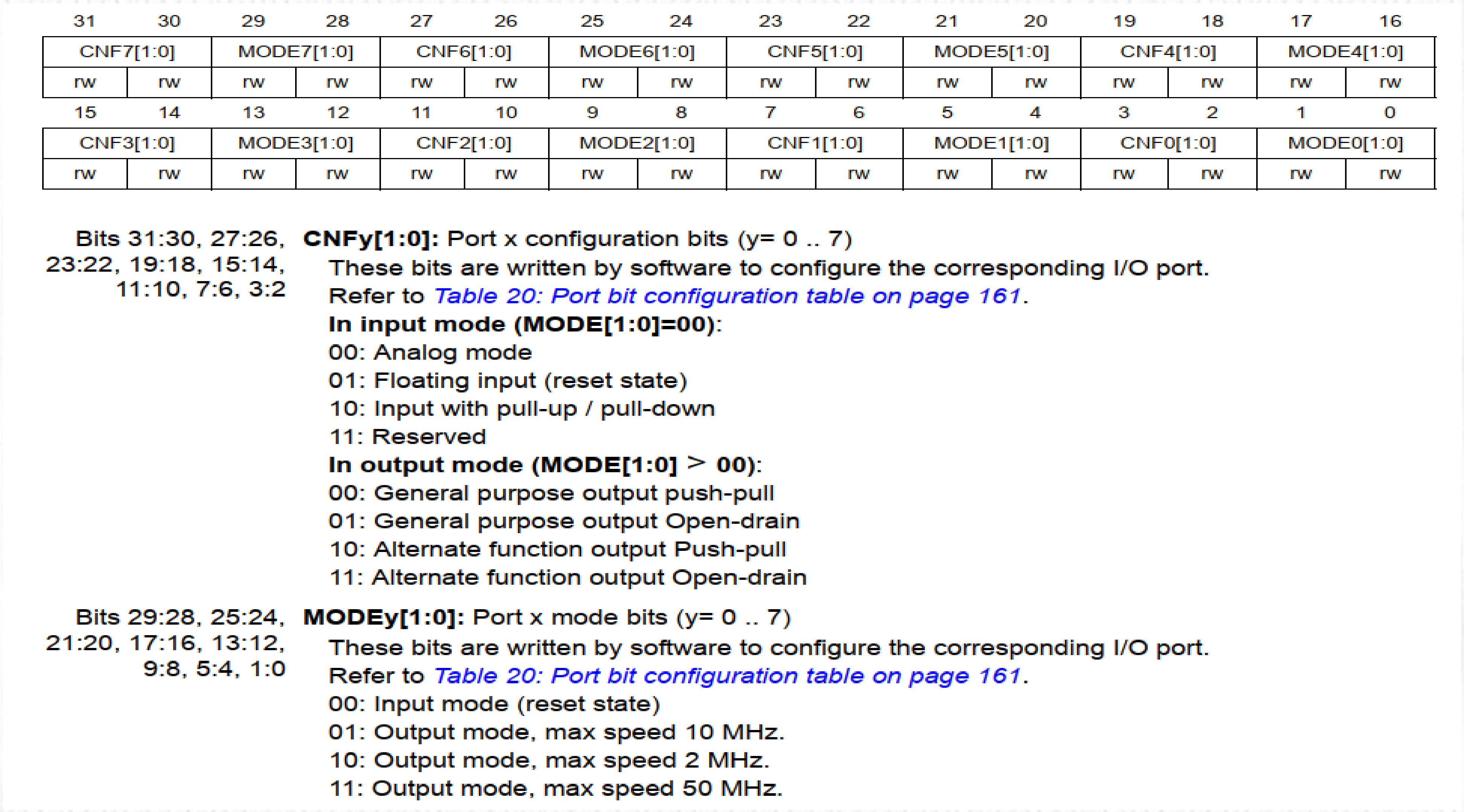
1. GPIO Function Description

- Each GPIO port in STM32 can be individually configured by software in 8 modes
  - Input floating
  - Input pull-up
  - Input pull-down
  - Analog
  - Output open-drain with pull-up or pull-down capability
  - Output push-pull with pull-up or pull-down capability
  - Alternate function push-pull with pull-up or pull-down capability
  - Alternate function open-drain with pull-up or pull-down capability
- More about GPIO and the corresponding feature
  - https://blog.stratifylabs.co/device/2013-10-21-Understanding-Microcontroller-Pin-Input-Output-Modes/
  - http://www.openedv.com/posts/list/21980.htm

1. GPIO Function Description

- Each group GPIO ports has 7 registers
  - two 32-bit configuration registers (GPIOx_CRL, GPIOx_CRH)
    - GPIOx_CRL: Port configuration register low
    - GPIOx_CRH: Port configuration register high
  - two 32-bit data registers (GPIOx_IDR, GPIOx_ODR)
    - GPIOx_IDR: Port input data register
    - GPIOx_ODR: Port output data register
  - a 32-bit set/reset register (GPIOx_BSRR)
  - a 16-bit reset register (GPIOx_BRR)
  - a 32-bit locking register (GPIOx_LCKR)
- Each I/O port bit is freely programmable, however the I/O port registers have to be accessed as 32-bit words (half-word or byte accesses are not allowed)

1. GPIO Function Description -- GPIOx_CRL



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[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
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2 **CNFy[1:0]:** Port x configuration bits (y= 0 .. 7)

These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 161*.
In input mode (MODE[1:0]=00):
00: Analog mode
01: Floating input (reset state)
10: Input with pull-up / pull-down
11: Reserved
In output mode (MODE[1:0] > 00):
00: General purpose output push-pull
01: General purpose output Open-drain
10: Alternate function output Push-pull
11: Alternate function output Open-drain

Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0 **MODEy[1:0]:** Port x mode bits (y= 0 .. 7)

These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 161*.
00: Input mode (reset state)
01: Output mode, max speed 10 MHz.
10: Output mode, max speed 2 MHz.
11: Output mode, max speed 50 MHz.

1. GPIO Function Description -- GPIOx_CRL

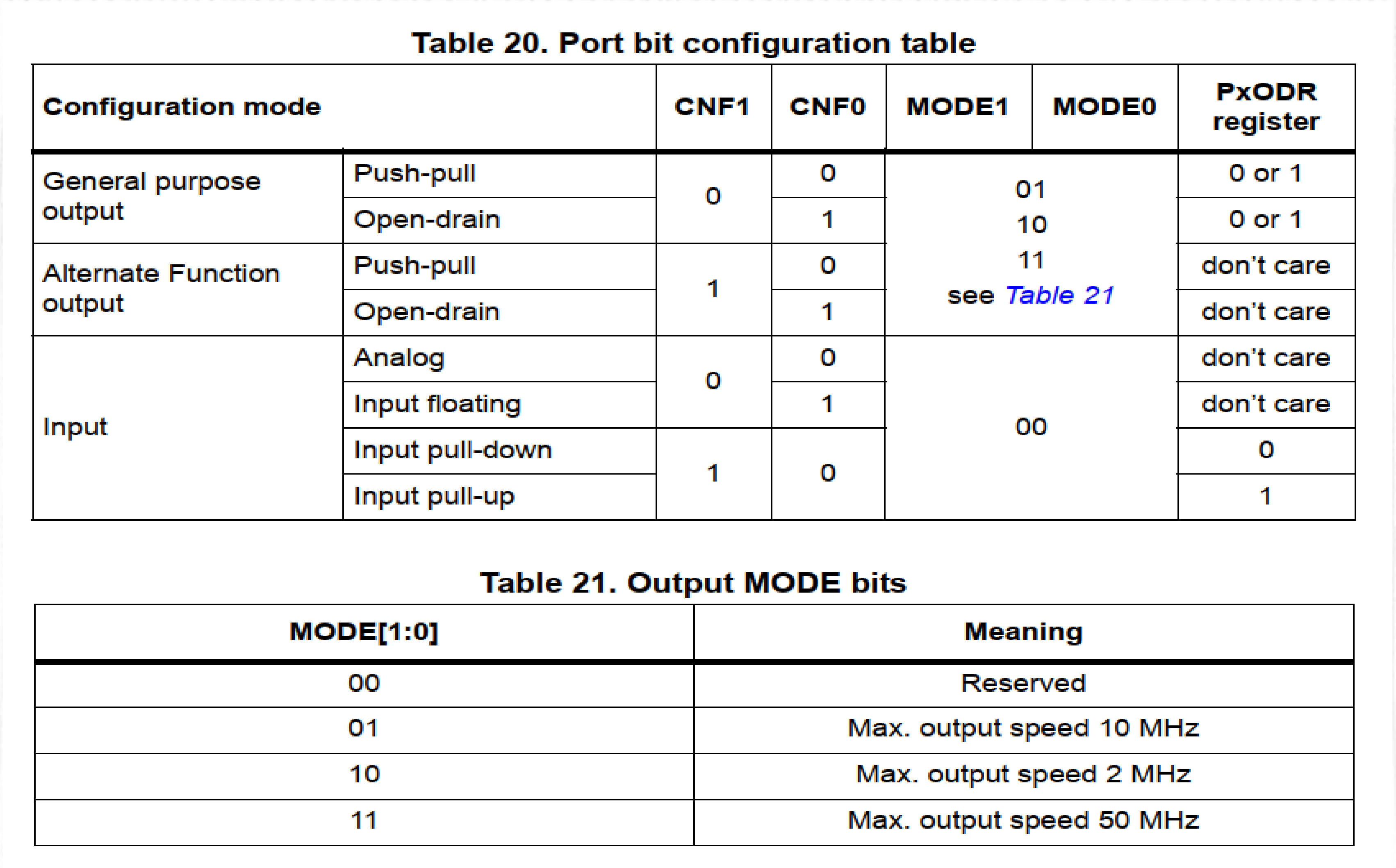


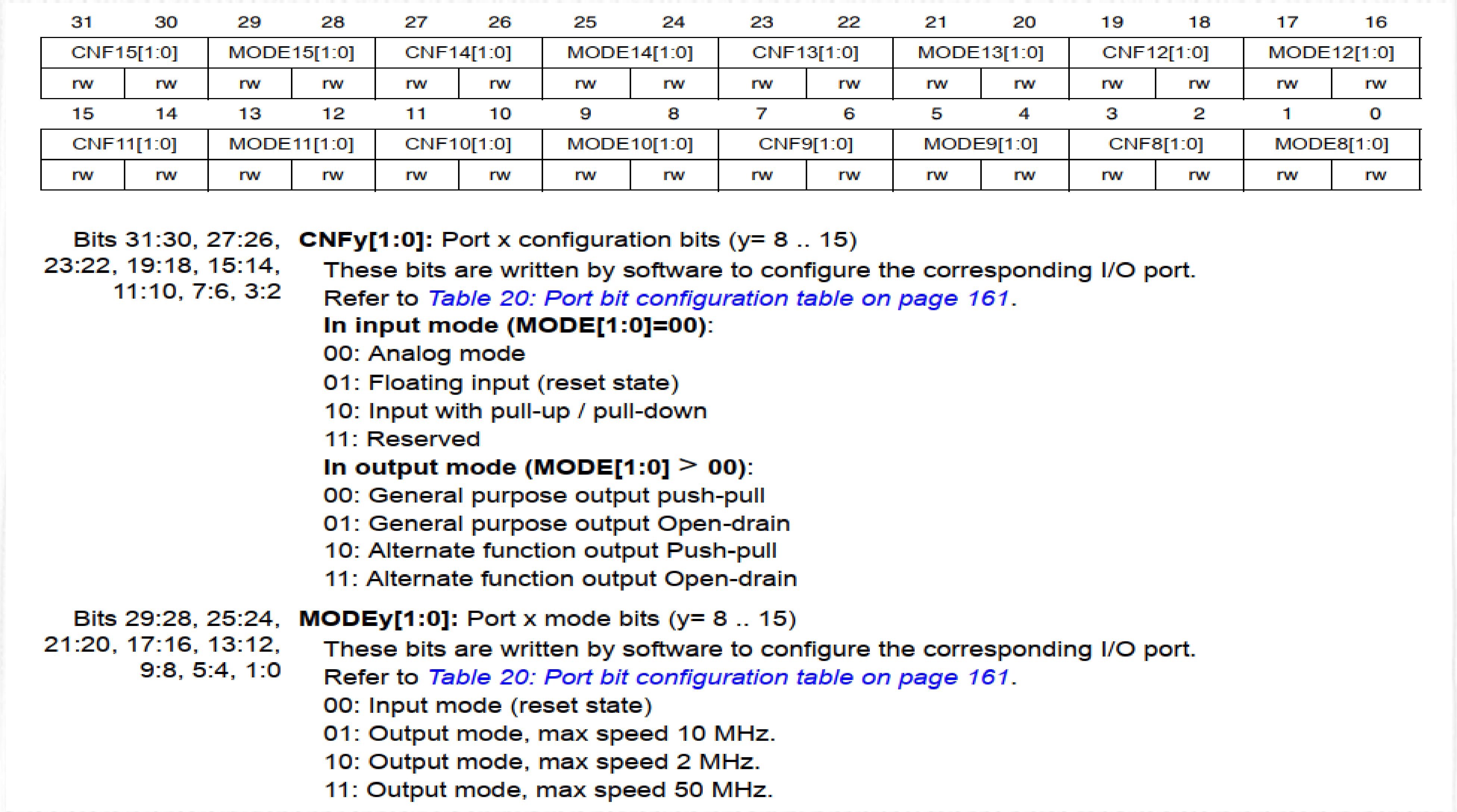
Table 20. Port bit configuration table

Configuration mode		CNF1	CNF0	MODE1	MODE0	PxODR register
General purpose output	Push-pull	0	0	01 10 11 see *Table 21*		0 or 1
	Open-drain		1			0 or 1
Alternate Function output	Push-pull	1	0			don't care
	Open-drain		1			don't care
Input	Analog	0	0	00		don't care
	Input floating		1			don't care
	Input pull-down	1	0			0
	Input pull-up					1

Table 21. Output MODE bits

MODE[1:0]	Meaning
00	Reserved
01	Max. output speed 10 MHz
10	Max. output speed 2 MHz
11	Max. output speed 50 MHz

1. GPIO Function Description -- GPIOx_CRH



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[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
CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2 **CNFy[1:0]:** Port x configuration bits (y= 8 .. 15)

These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 161*.

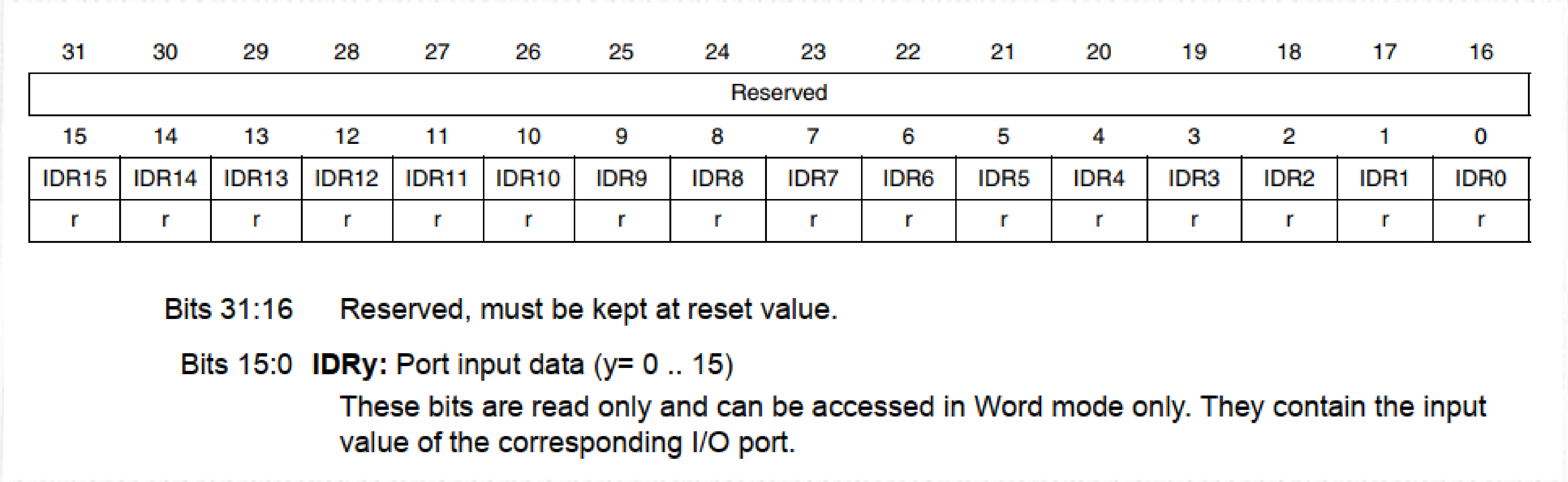
In input mode (MODE[1:0]=00):
00: Analog mode
01: Floating input (reset state)
10: Input with pull-up / pull-down
11: Reserved

In output mode (MODE[1:0] > 00):
00: General purpose output push-pull
01: General purpose output Open-drain
10: Alternate function output Push-pull
11: Alternate function output Open-drain

Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0 **MODEy[1:0]:** Port x mode bits (y= 8 .. 15)

These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 161*.
00: Input mode (reset state)
01: Output mode, max speed 10 MHz.
10: Output mode, max speed 2 MHz.
11: Output mode, max speed 50 MHz.

1. GPIO Function Description
-- GPIOx_IDR



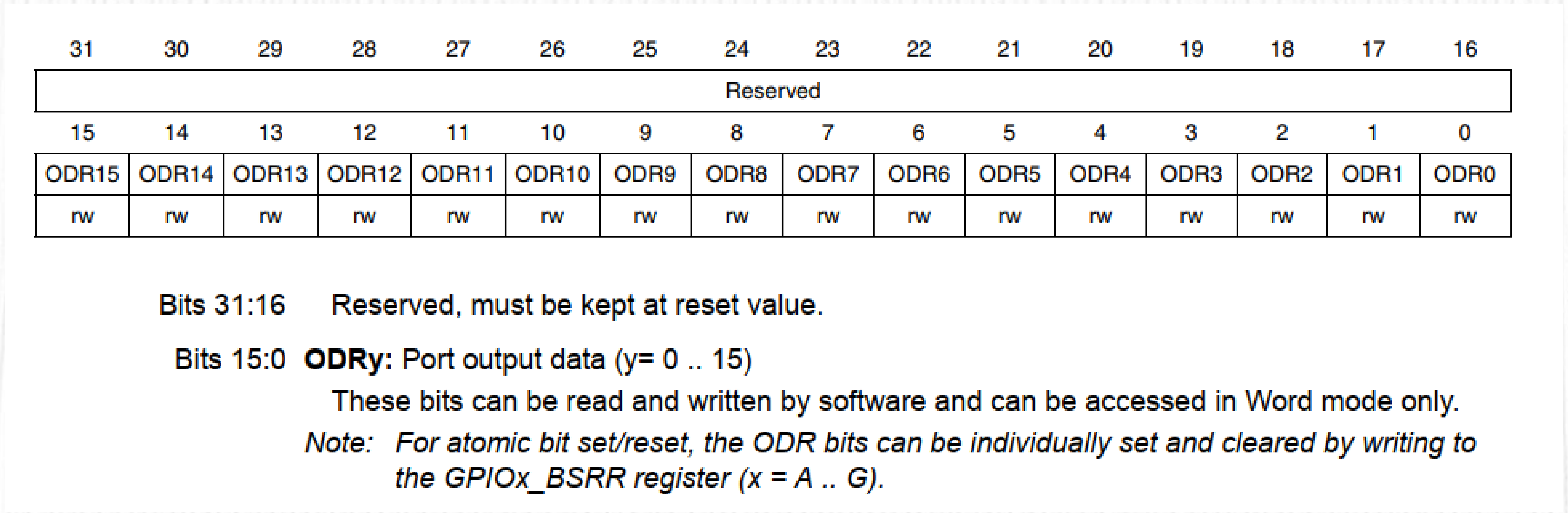
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

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.

1. GPIO Function Description -- GPIOx_ODR



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

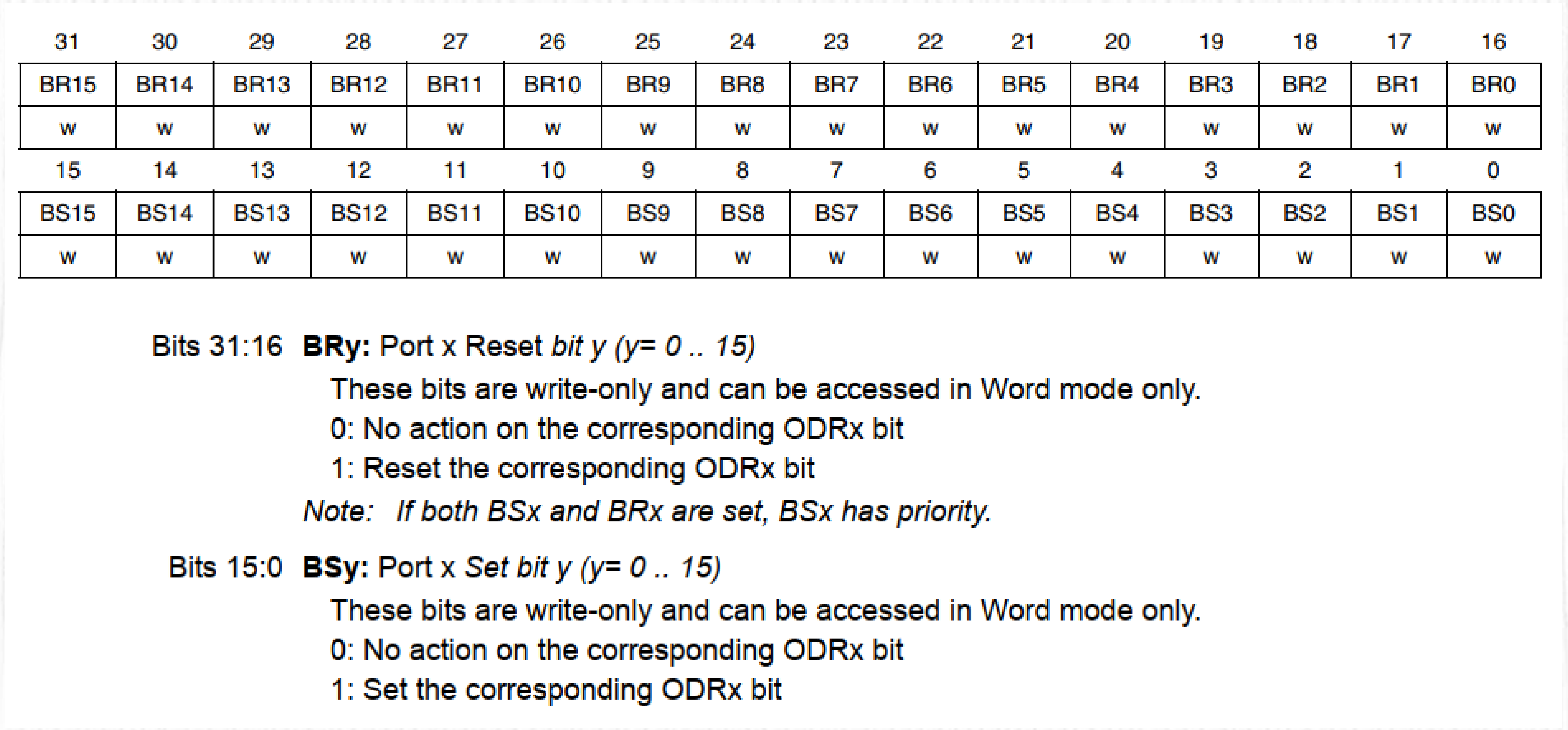
Bits 31:16 Reserved, must be kept at reset value.

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

These bits can be read and written by software and can be accessed in Word mode only.

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

1. GPIO Function Description -- GPIOx_BSRR



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 **BRy:** Port x Reset *bit y (y= 0 .. 15)*

These bits are write-only and can be accessed in Word mode only.

0: No action on the corresponding ODRx bit

1: Reset the corresponding ODRx bit

Note: If both BSx and BRx are set, BSx has priority.

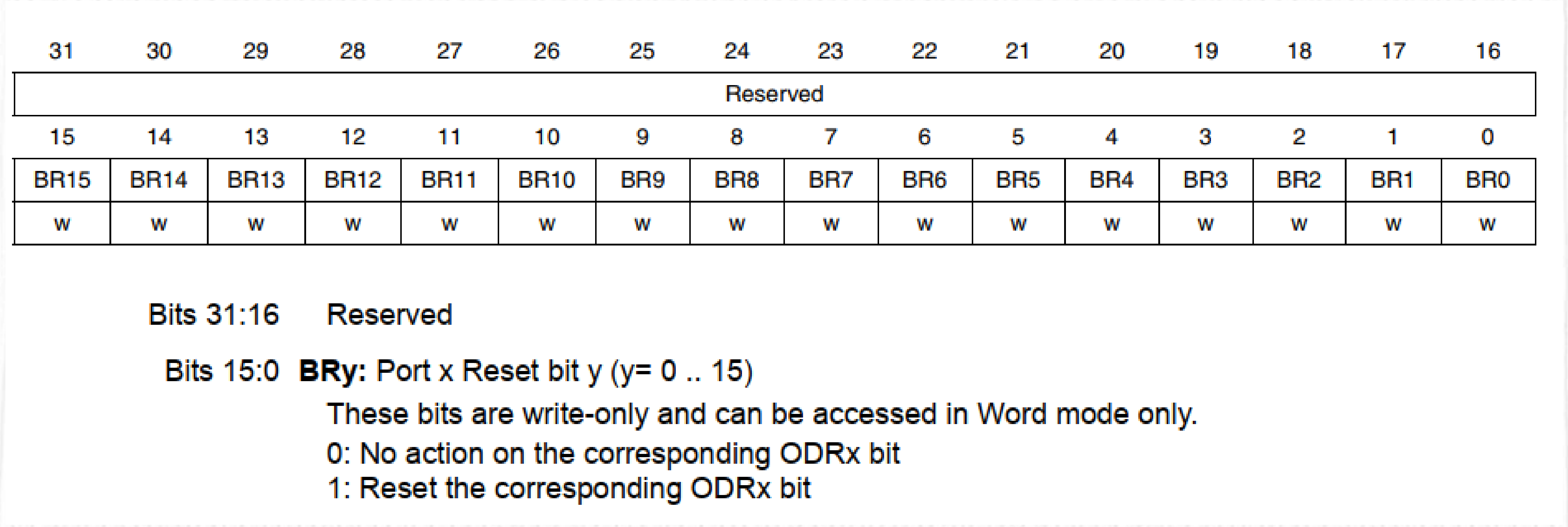
Bits 15:0 **BSy:** Port x *Set bit y (y= 0 .. 15)*

These bits are write-only and can be accessed in Word mode only.

0: No action on the corresponding ODRx bit

1: Set the corresponding ODRx bit

1. GPIO Function Description
-- GPIOx_BRR



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
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 Reserved

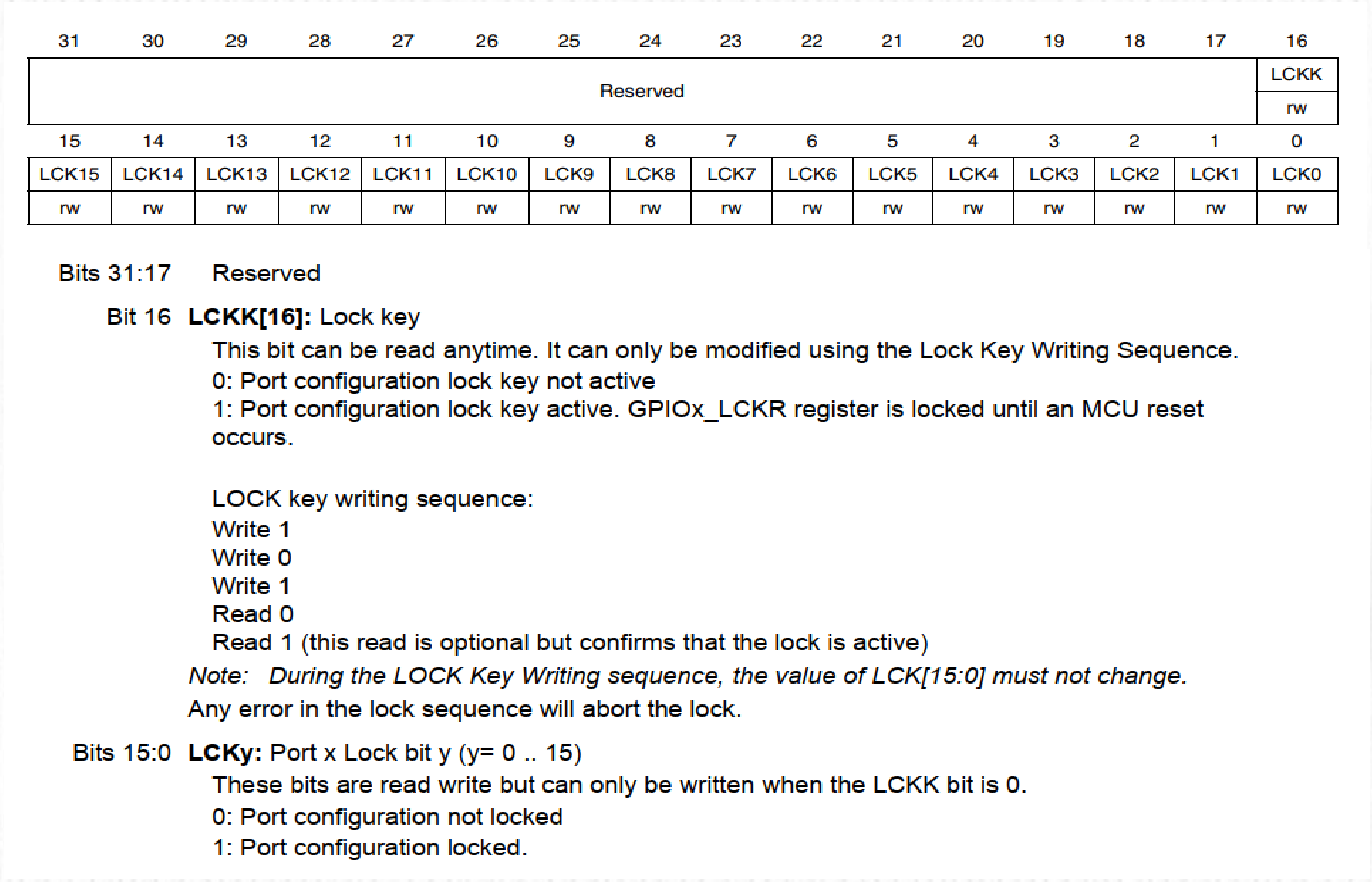
Bits 15:0 **BRy:** Port x Reset bit y (y= 0 .. 15)

These bits are write-only and can be accessed in Word mode only.

0: No action on the corresponding ODRx bit

1: Reset the corresponding ODRx bit

1. GPIO Function Description -- GPIOx_LCKR



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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LCK15	LCK14	LCK13	LCK12	LCK11	LCK10	LCK9	LCK8	LCK7	LCK6	LCK5	LCK4	LCK3	LCK2	LCK1	LCK0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:17 Reserved

Bit 16 **LCKK[16]:** Lock key

This bit can be read anytime. It can only be modified using the Lock Key Writing Sequence.

0: Port configuration lock key not active

1: Port configuration lock key active. GPIOx_LCKR register is locked until an MCU reset occurs.

LOCK key writing sequence:

Write 1

Write 0

Write 1

Read 0

Read 1 (this read is optional but confirms that the lock is active)

Note: During the LOCK Key Writing sequence, the value of LCK[15:0] must not change.

Any error in the lock sequence will abort the lock.

Bits 15:0 **LCKy:** Port x Lock bit y (y= 0 .. 15)

These bits are read write but can only be written when the LCKK bit is 0.

0: Port configuration not locked

1: Port configuration locked.

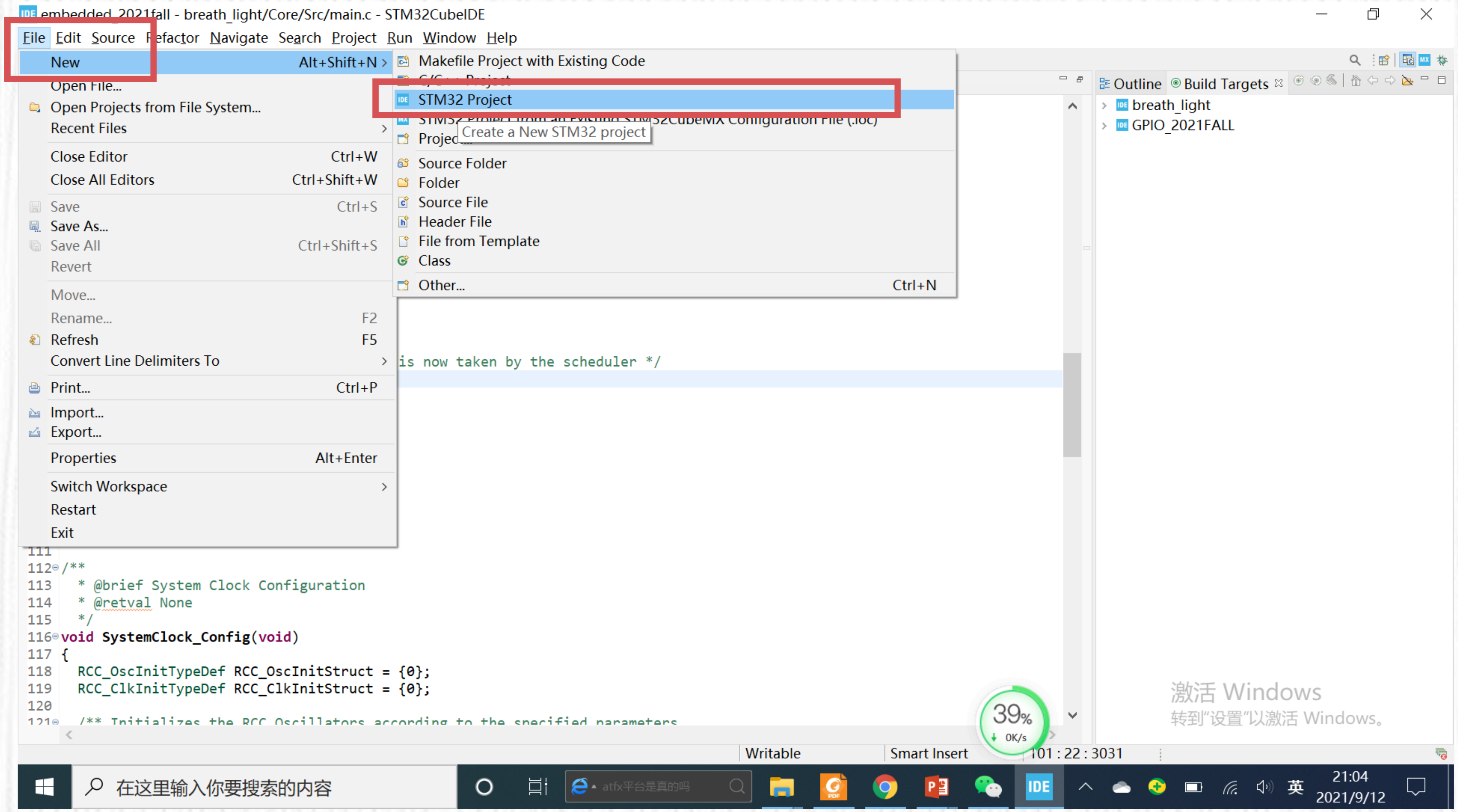
南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

02

How to create a new project

2. How to create a new project

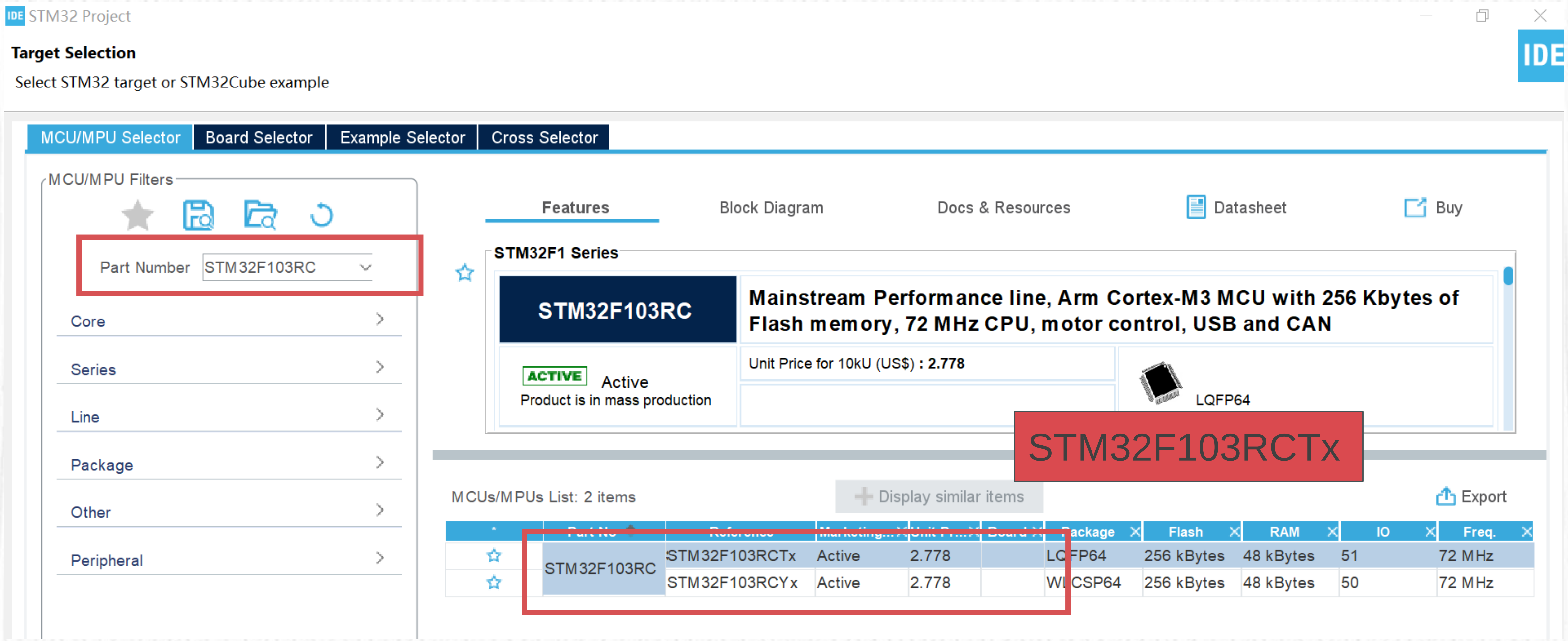
- Create a new STM32 project


embedded_2021fall - breath_light/Core/Src/main.c - STM32CubeIDE

```
is now taken by the scheduler */

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
```

2. How to create a new project

- Target selection -- STM32F103RCTx



STM32 Project

Target Selection

Select STM32 target or STM32Cube example

STM32F1 Series

STM32F103RC

Mainstream Performance line, Arm Cortex-M3 MCU with 256 Kbytes of Flash memory, 72 MHz CPU, motor control, USB and CAN

ACTIVE Active
Product is in mass production

Unit Price for 10kU (US$) : **2.778**

LQFP64

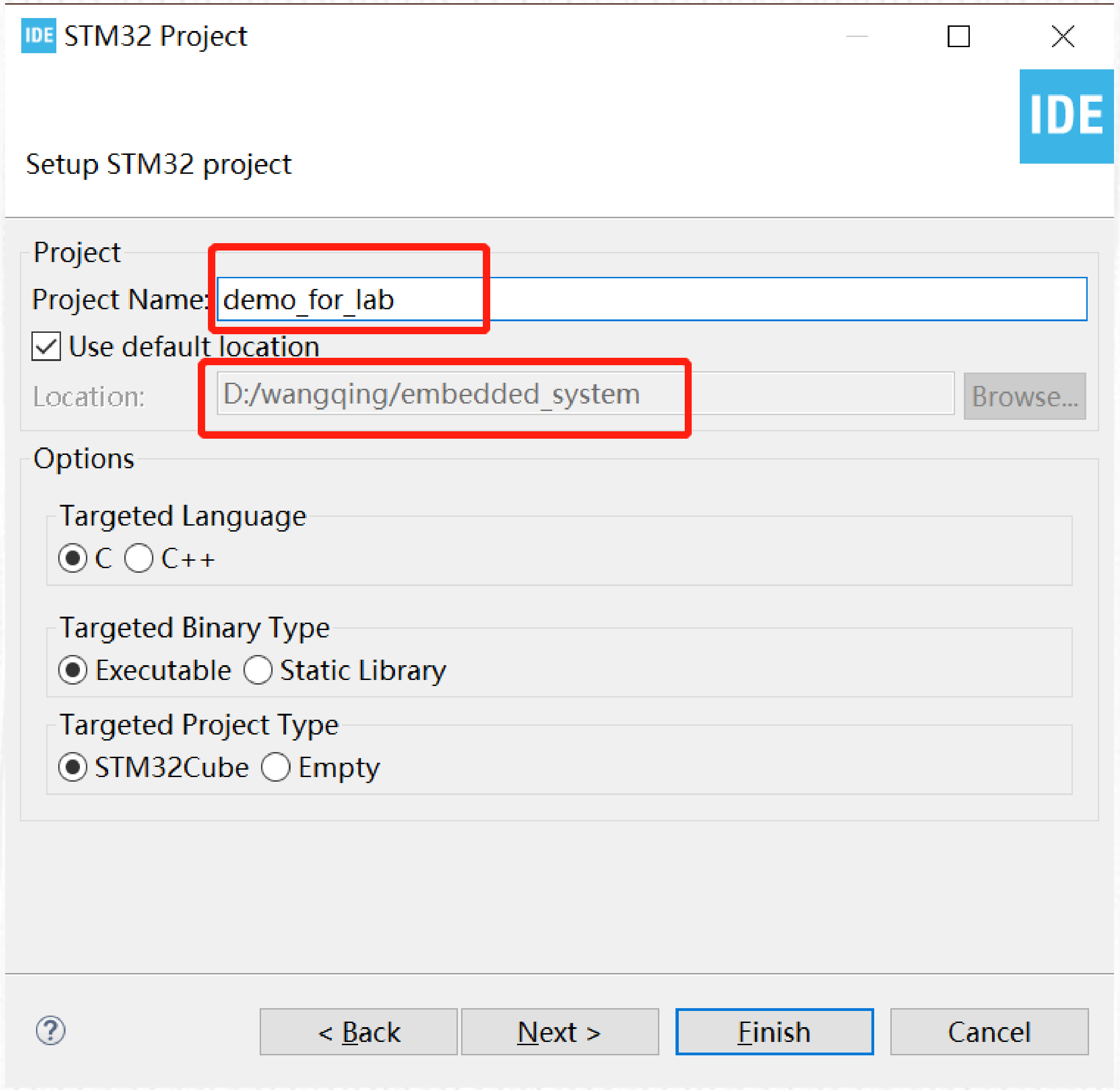
STM32F103RCTx

MCUs/MPUs List: 2 items

*	Part No	Reference	Marketing...	Unit P...	Board	Package	Flash	RAM	IO	Freq.
☆	STM32F103RC	STM32F103RCTx	Active	2.778		LQFP64	256 kBytes	48 kBytes	51	72 MHz
☆		STM32F103RCYx	Active	2.778		WLCSP64	256 kBytes	48 kBytes	50	72 MHz

2. How to create a new project

- Enter the project name – **only ASCII characters**
- Keep other options as default


STM32 Project

Setup STM32 project

Project

Project Name: demo_for_lab

[✓] Use default location

Location: D:/wangqing/embedded_system

Options

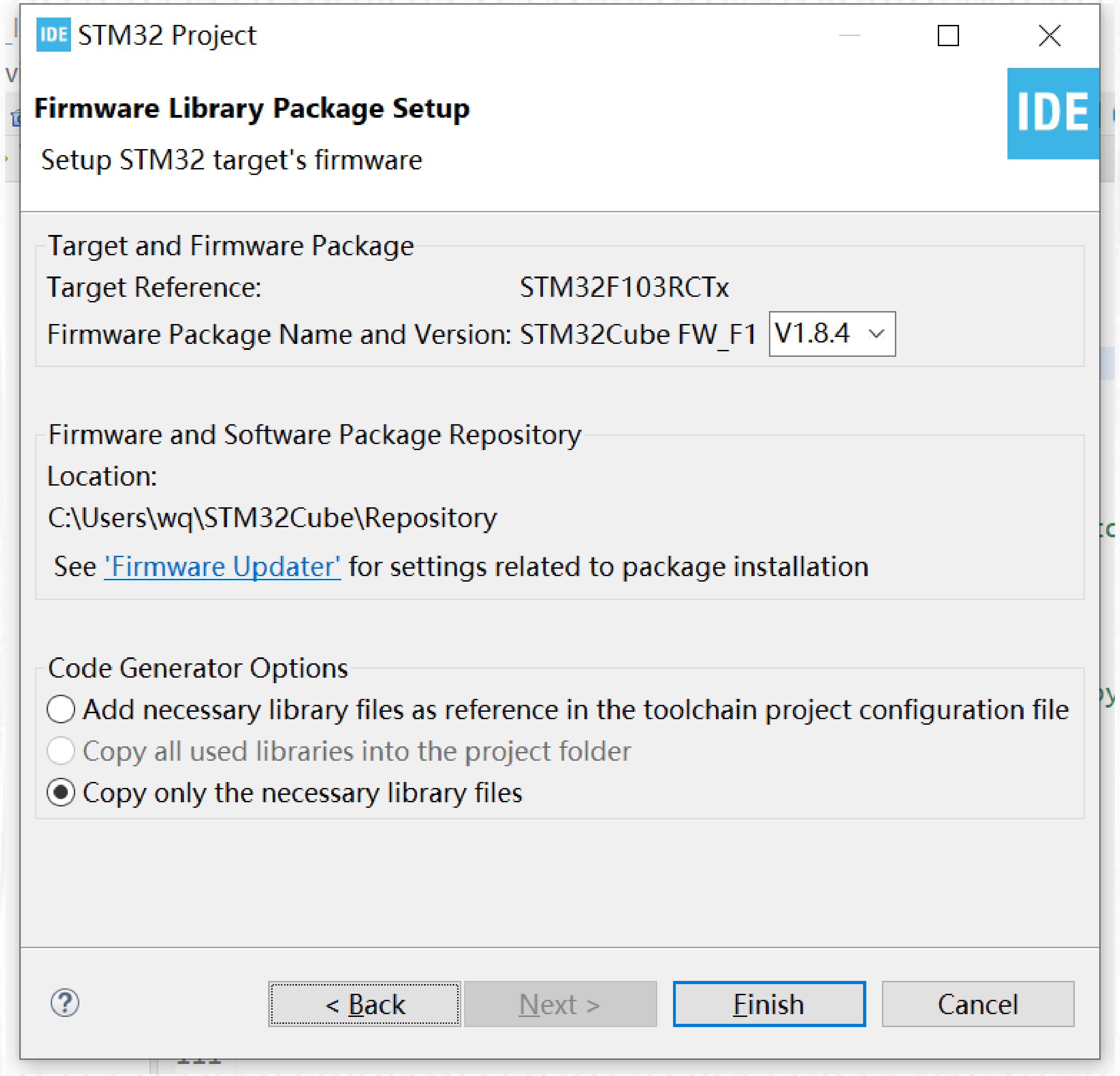
Targeted Language
[● C] ○ C++

Targeted Binary Type
[● Executable] ○ Static Library

Targeted Project Type
[● STM32Cube] ○ Empty

2. How to create a new project

- Check the project information
- Click ***Finish***



STM32 Project

Firmware Library Package Setup

Setup STM32 target's firmware

Target and Firmware Package

Target Reference: STM32F103RCTx

Firmware Package Name and Version: STM32Cube FW_F1 V1.8.4

Firmware and Software Package Repository

Location:

C:\Users\wq\STM32Cube\Repository

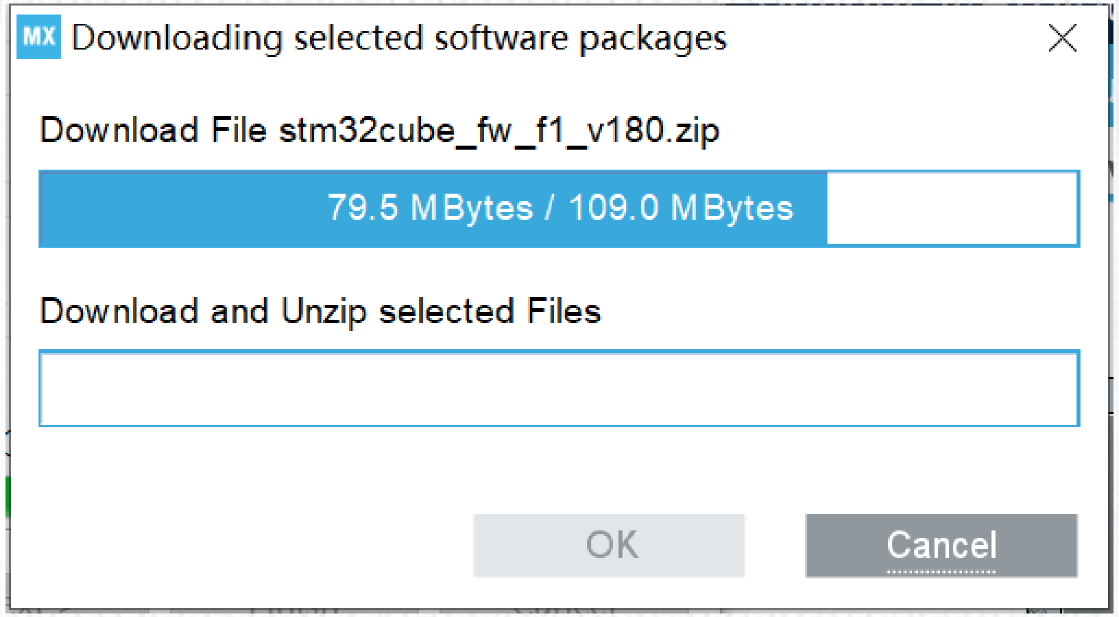
See 'Firmware Updater' for settings related to package installation

Code Generator Options

- ○ Add necessary library files as reference in the toolchain project configuration file
- ○ Copy all used libraries into the project folder
- ● Copy only the necessary library files

2. How to create a new project

- Download software packages



Downloading selected software packages

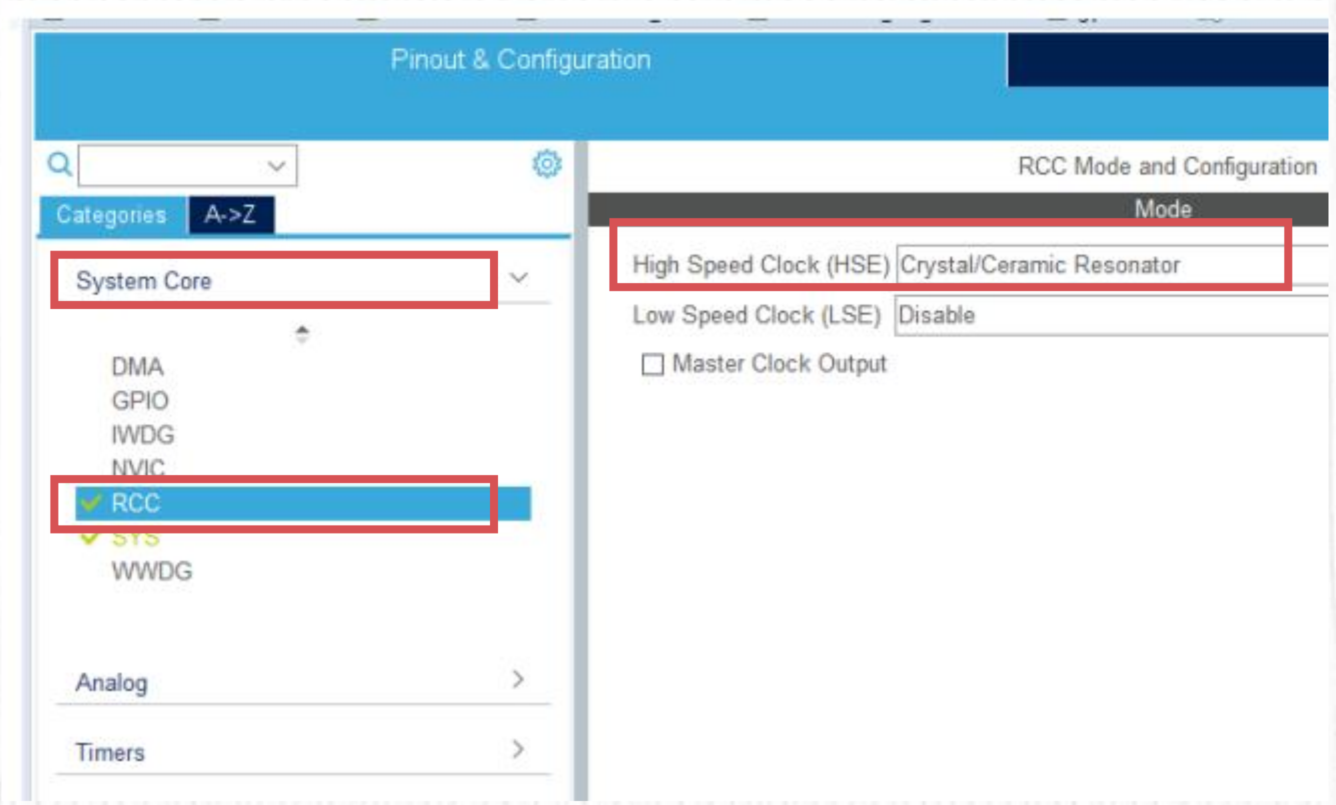
Download File stm32cube_fw_f1_v180.zip

79.5 MBytes / 109.0 MBytes

Download and Unzip selected Files

2. How to create a new project

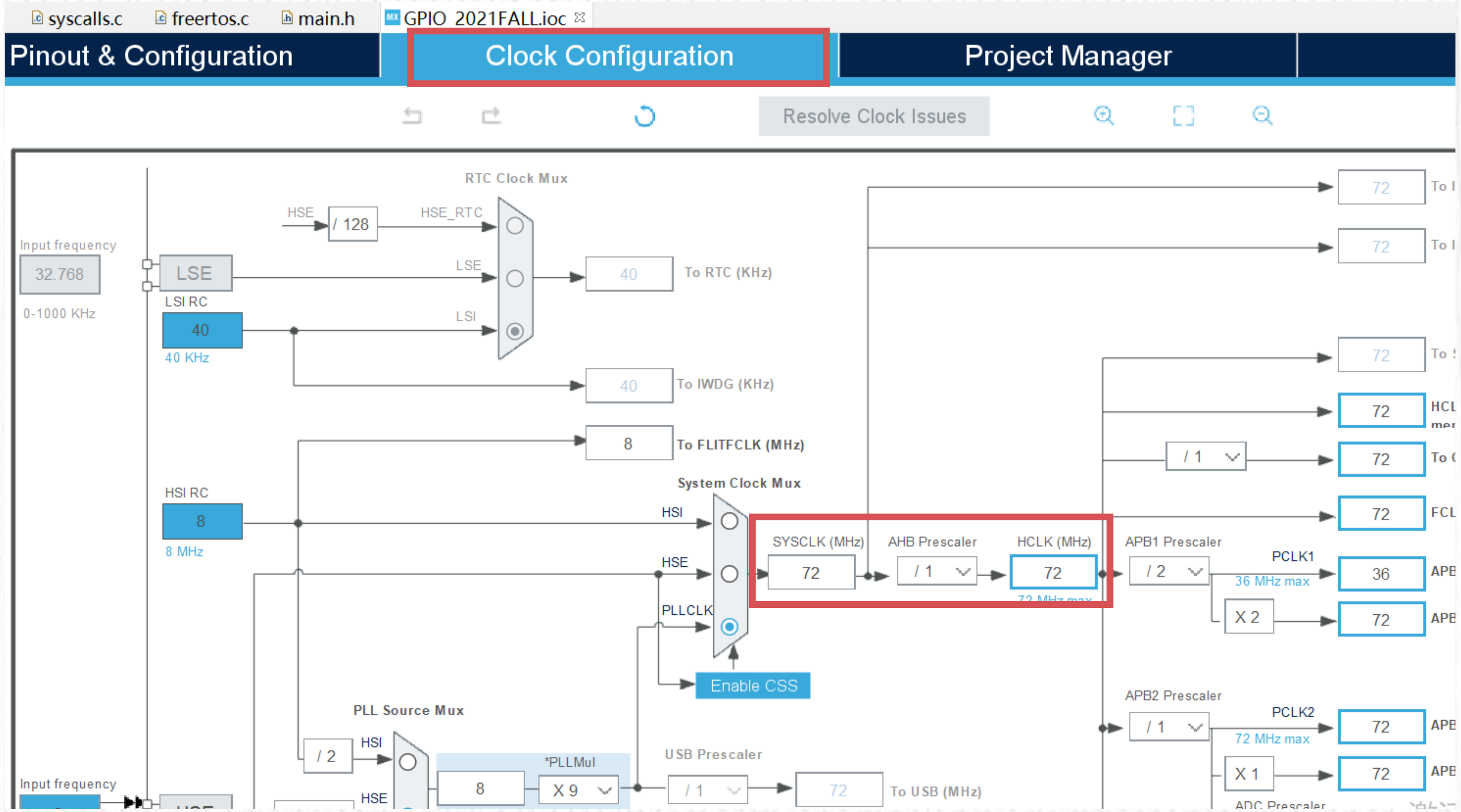
- RCC and Clock Configuration
- There are four kinds of clock sources in STM32: HSE clock, HSI clock, LSE clock and LSI clock (HS for high speed, LS for low speed, E for external and I for internal)
- Only HSE and HSI clock can used to driven the SYSCLK
- Most of the case we use an external crystal oscillator or ceramic resonator (HSE) to drive the SYSCLK because it is more accurate



High Speed Clock (HSE) Crystal/Ceramic Resonator
Low Speed Clock (LSE) Disable
☐ Master Clock Output

2. How to create a new project

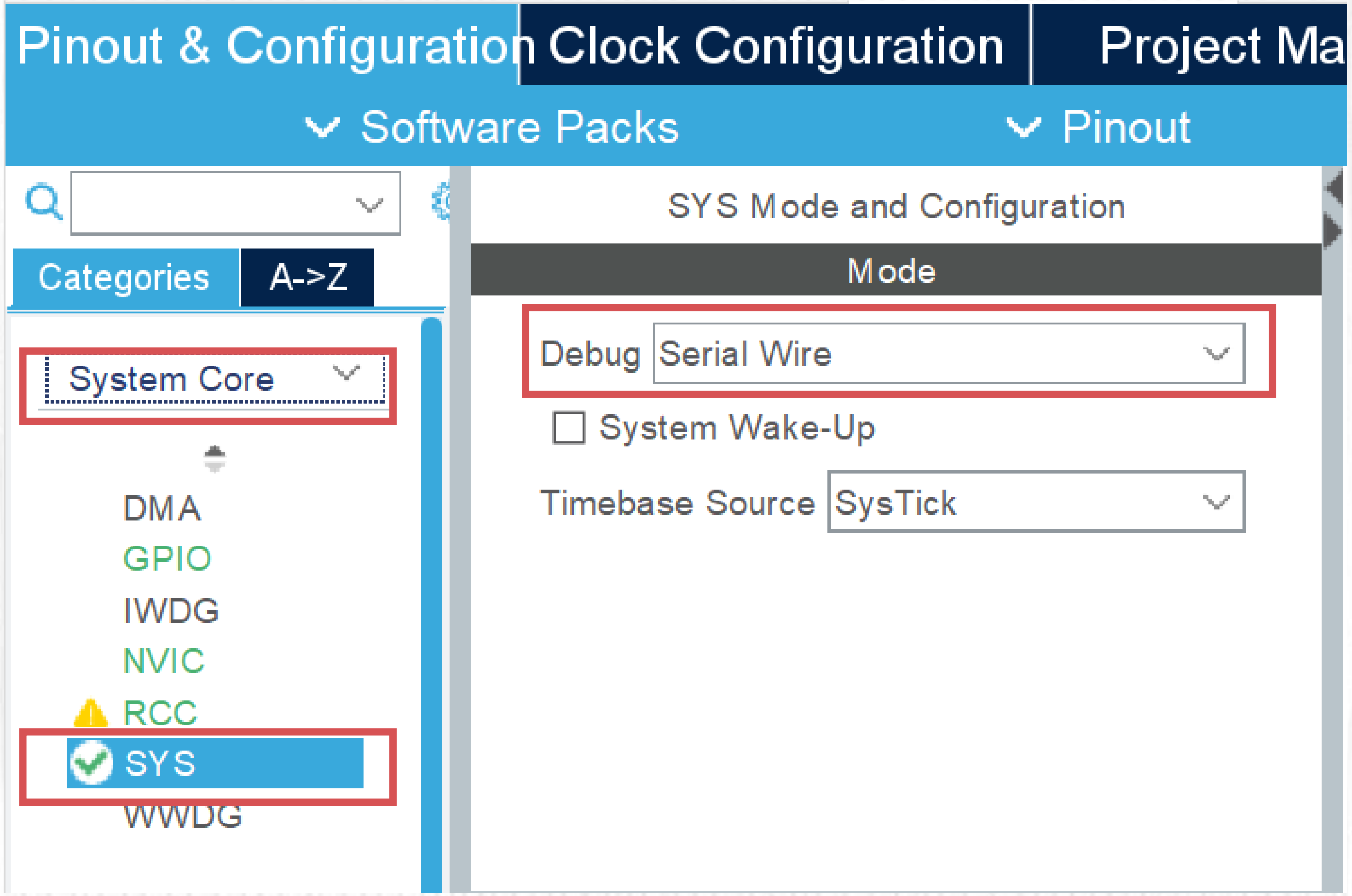
- Clock Configuration
  - Change to **clock configuration**
  - Set SYSCLK as 72M (maximum)



2. How to create a new project

- SYS Mode and Configuration

- Back to **Pinout & Configuration -> Categories -> System Core -> SYS**
- Use Serial Wire(SW) as debug wire



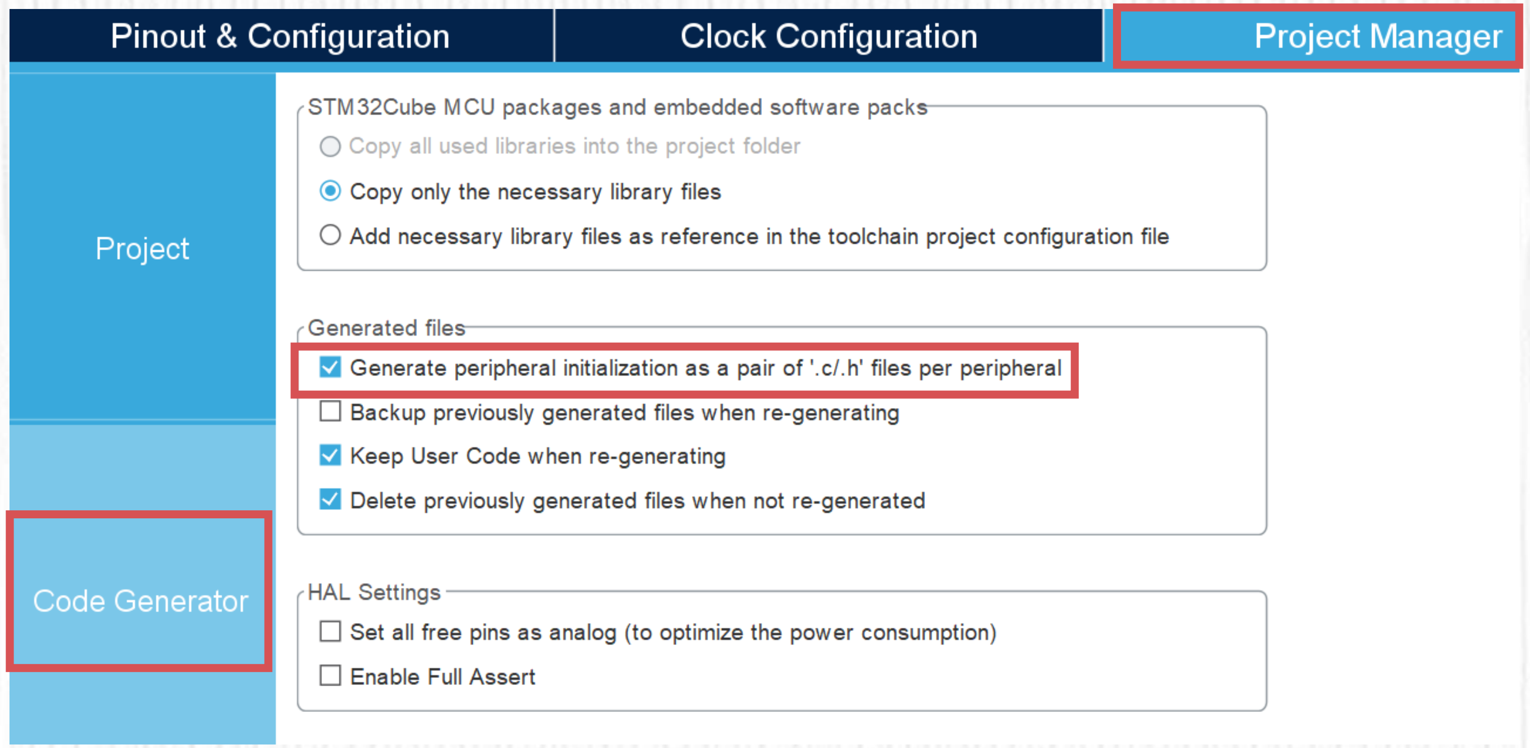
Debug Serial Wire

☐ System Wake-Up

Timebase Source SysTick

2. How to create a new project

- Project Management
  - Change to **Project Manager -> Code Generator**
  - Check up ***Generate peripheral initialization as a pair of '.c/.h' files per peripheral***, which will make the codes more modular



STM32Cube MCU packages and embedded software packs

- ( ) Copy all used libraries into the project folder
- (●) Copy only the necessary library files
- ( ) Add necessary library files as reference in the toolchain project configuration file

Generated files

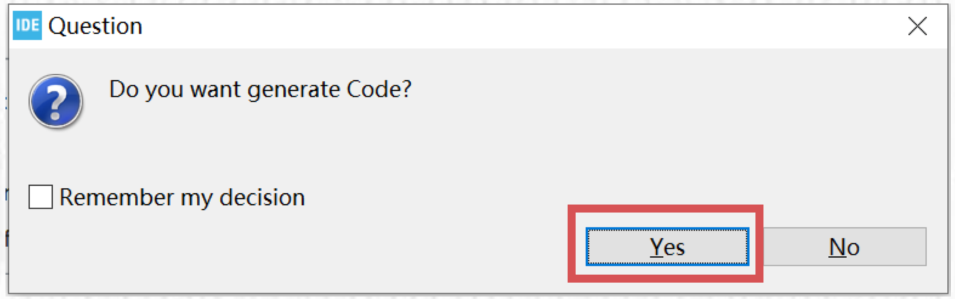
- ☑ Generate peripheral initialization as a pair of '.c/.h' files per peripheral
- ☐ Backup previously generated files when re-generating
- ☑ Keep User Code when re-generating
- ☑ Delete previously generated files when not re-generated

HAL Settings

- ☐ Set all free pins as analog (to optimize the power consumption)
- ☐ Enable Full Assert

2. How to create a new project

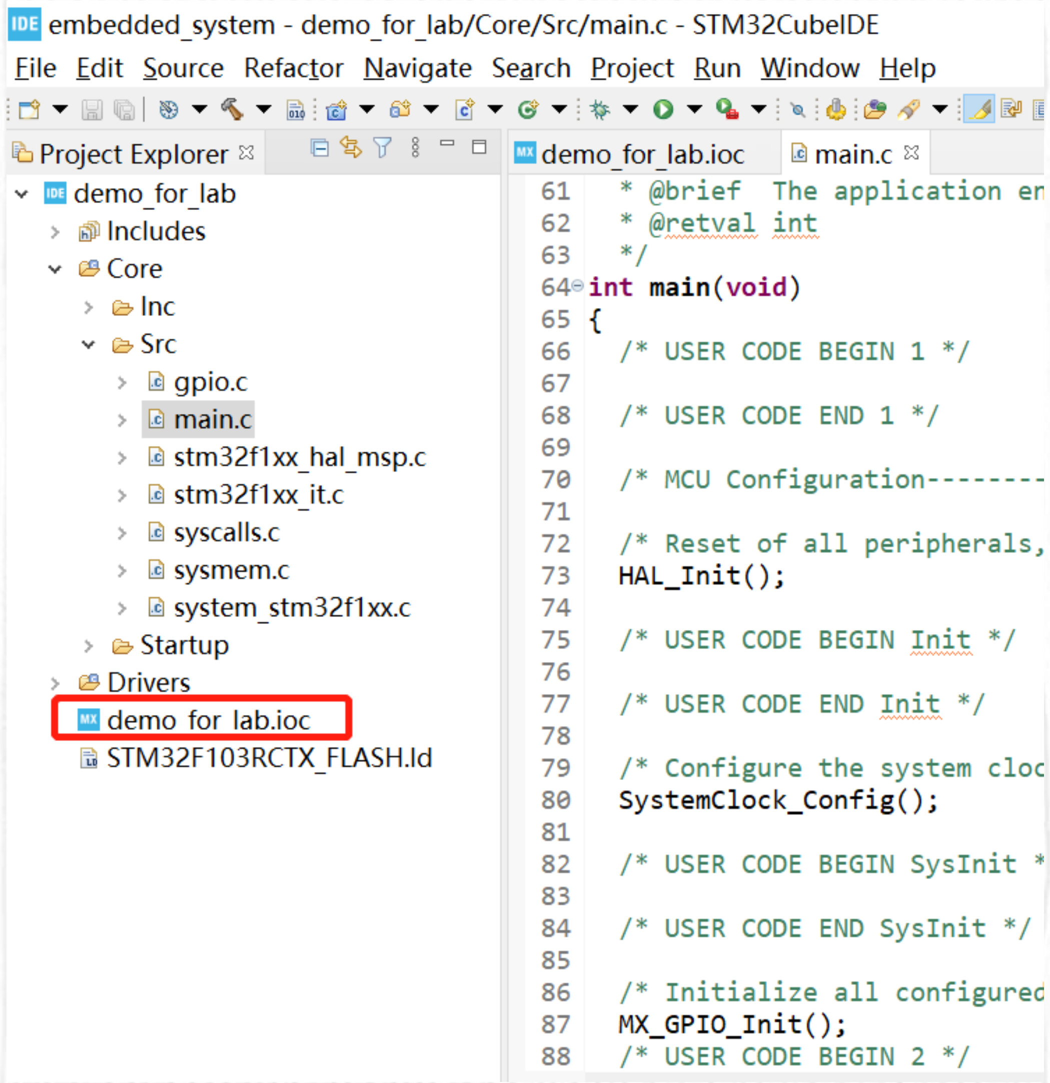
- Project Management
- Save the configuration information and start a new project


Question

Do you want generate Code?

☐ Remember my decision

Yes No


embedded_system - demo_for_lab/Core/Src/main.c - STM32CubeIDE

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer

```
demo_for_lab
  Includes
  Core
    Inc
    Src
      gpio.c
      main.c
      stm32f1xx_hal_msp.c
      stm32f1xx_it.c
      syscalls.c
      sysmem.c
      system_stm32f1xx.c
    Startup
  Drivers
  demo_for_lab.ioc
  STM32F103RCTX_FLASH.ld
```

demo_for_lab.ioc main.c

```c
  * @brief  The application er
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------

  /* Reset of all peripherals,
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system cloc
  SystemClock_Config();

  /* USER CODE BEGIN SysInit *

  /* USER CODE END SysInit */

  /* Initialize all configured
  MX_GPIO_Init();
  /* USER CODE BEGIN 2 */
```

03

Programming using registers

3. Programming using registers

- Registers are a special type of memory within a CMU that can control various functions of the CMU, including various states of the kernel and peripherals.
- Registers are required to implement various controls over CMUs.
- Register resources are precious and typically use one or several bits to control a function.
- STM32 has hundreds of internal registers, which are 32-bit.
- The STM32 register can be divided into two major categories
  - Core registers
  - Peripheral device registers

3. Programming using registers

Category	Register	Description
Core registers	Kernel related registers	Including R0~R15, Xpsr, and special function register
	Interrupt control registers	Including NVIC and SCB related registers. NVIC includes ISER, ICER, ISPR, IP, etc.; SCB includes VTOR, AIRCR, SCR, etc.
	SysTick registers	Including four registers: CTRL, LOAD, VAL, and CALIB
	Memory protection register	Optional function, not available in STM32F103
	System debug registers	ETM, ITM, DWT, IPIU and other related registers
Peripheral device registers		Registers for various peripherals such as GPIO, UART, IIC, SPI, TIM, DMA, ADC, DAC, RTC, IWDG, WWDG, PWR, CAN, USB…

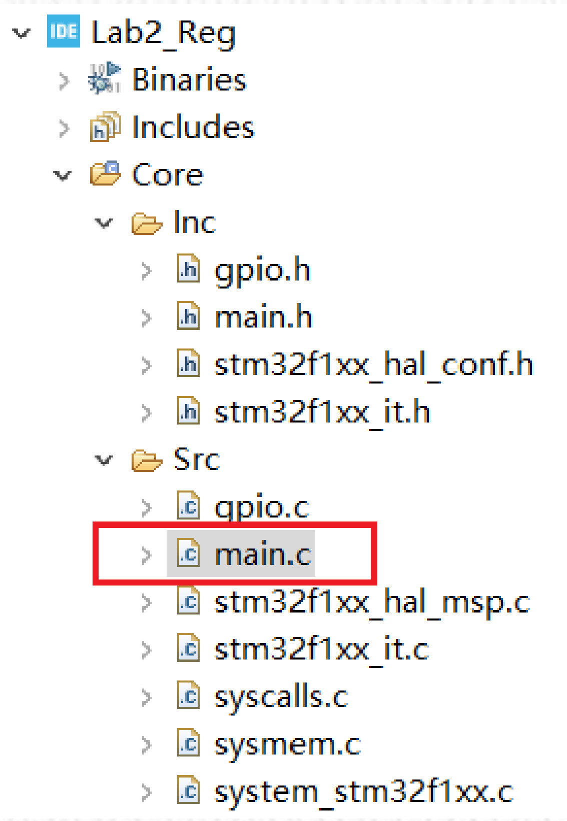
- Generally, interrupt control registers and SysTick registers are concerned during the kernel registers.
- Peripheral registers vary depending on different peripherals.

3. Programming using registers

- Codes for implementation can be added to *main. c* file or the component code file.
- Taking the ODR register of GPIOB as an example
  - The address is： 0X40010C0C
  - The assignment can be written as:

    (*(unsigned int *))(0X40010C0C) = 0XFFFF;
  - We have assigned the GPIOB ->ODR register with a value of 0XFFFF, indicating that all outputs of IO ports (16 IO ports) of GPIOB are high levels.
  - The same function can also be written as：

    GPIOB -> ODR = 0XFFFF;



```
Lab2_Reg
  Binaries
  Includes
  Core
    Inc
      gpio.h
      main.h
      stm32f1xx_hal_conf.h
      stm32f1xx_it.h
    Src
      gpio.c
      main.c
      stm32f1xx_hal_msp.c
      stm32f1xx_it.c
      syscalls.c
      sysmem.c
      system_stm32f1xx.c
```

3. Programming using registers

- Bitwise operators

Operator	&	~	|	^	<<	>>
Operation	AND	NOT	OR	XOR	Left shift	Right shift

- Setting values for specified bits without changing the values of others

```
GPIOA->CRL &= 0XFFFFFFBF; /* Clear bit6 (starting from 0) to 0 */
GPIOA->CRL |= 0X00000040; /* Set the value of bit6 to 1 without changing other bits */
```

- Using shift operation to improve code readability

```
SysTick->CTRL |= 1 << 1; /* Set the value of bit1 to 1 */ /* better readability*/
SysTick->CTRL |= 0X0002; /* Set the value of bit1 to 1 */
```

- Using bitwise NOT to clear one or more bits

```
SysTick->CTRL &= ~(1 << 0) ; /* close SYSTICK */ /* better readability*/
SysTick->CTRL &= 0XFFFFFFFE; /* close SYSTICK */
```

- Using bitwise XOR to control the flipping of one or more bit states

```
GPIOB->ODR ^= 1 << 5;
```

04

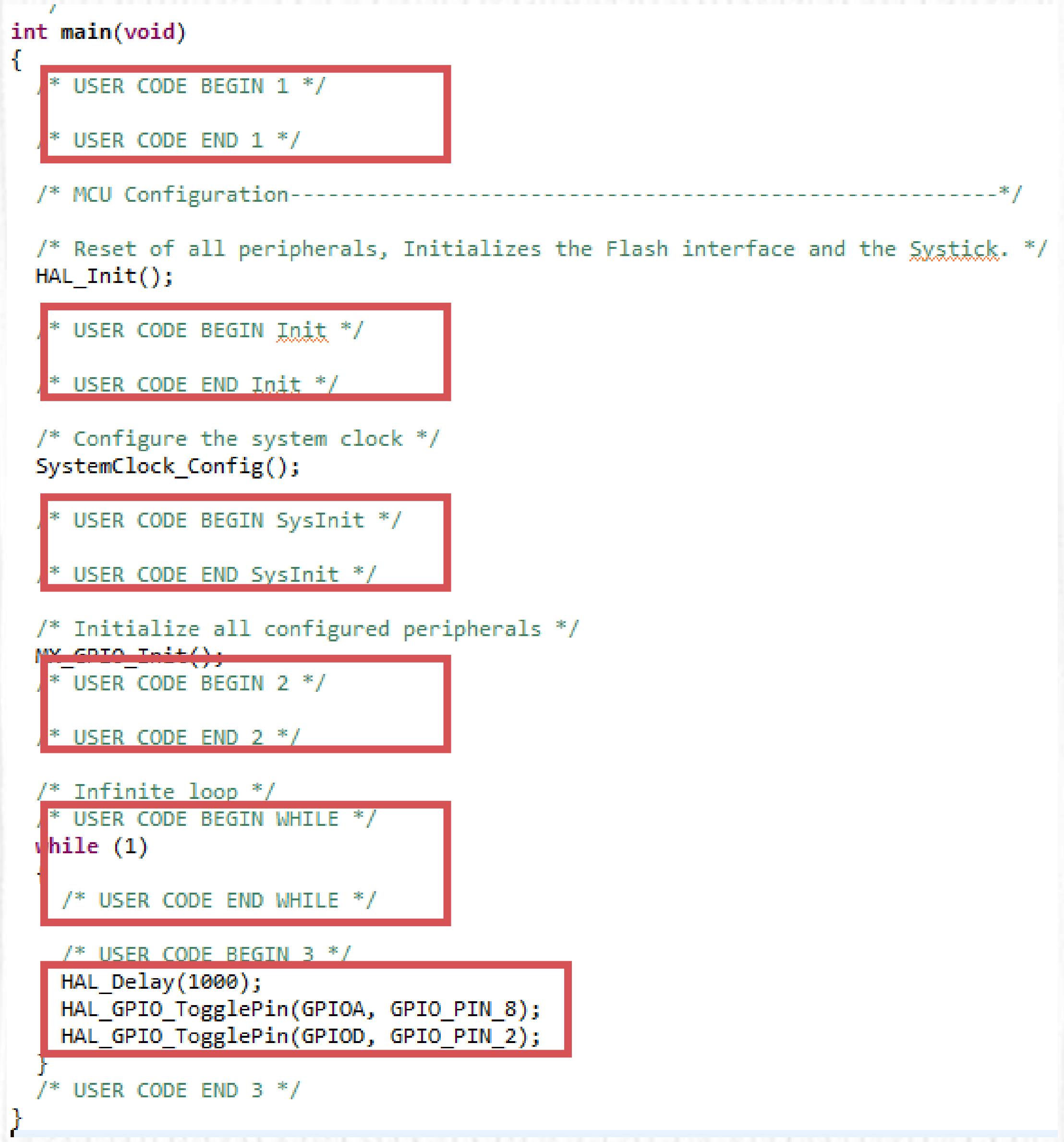
Programming using HAL APIs

4. Programming using HAL APIs

- HAL(Hardware Abstract Layer) library
- Keep updating, bugs will be changed at the next version
- Rapid development, work with cube tool and generate code with one click
- It's convenient to replace the chip and transplant it. You don't have to think about what special registers this chip has. The manufacturer has made it for you
- Learn more about HAL
  - https://bbs.21ic.com/icview-2512392-1-1.html?_dsign=133c8287
  - https://www.jianshu.com/p/c6809c2bcb4f?from=timeline

4. Programming using HAL APIs

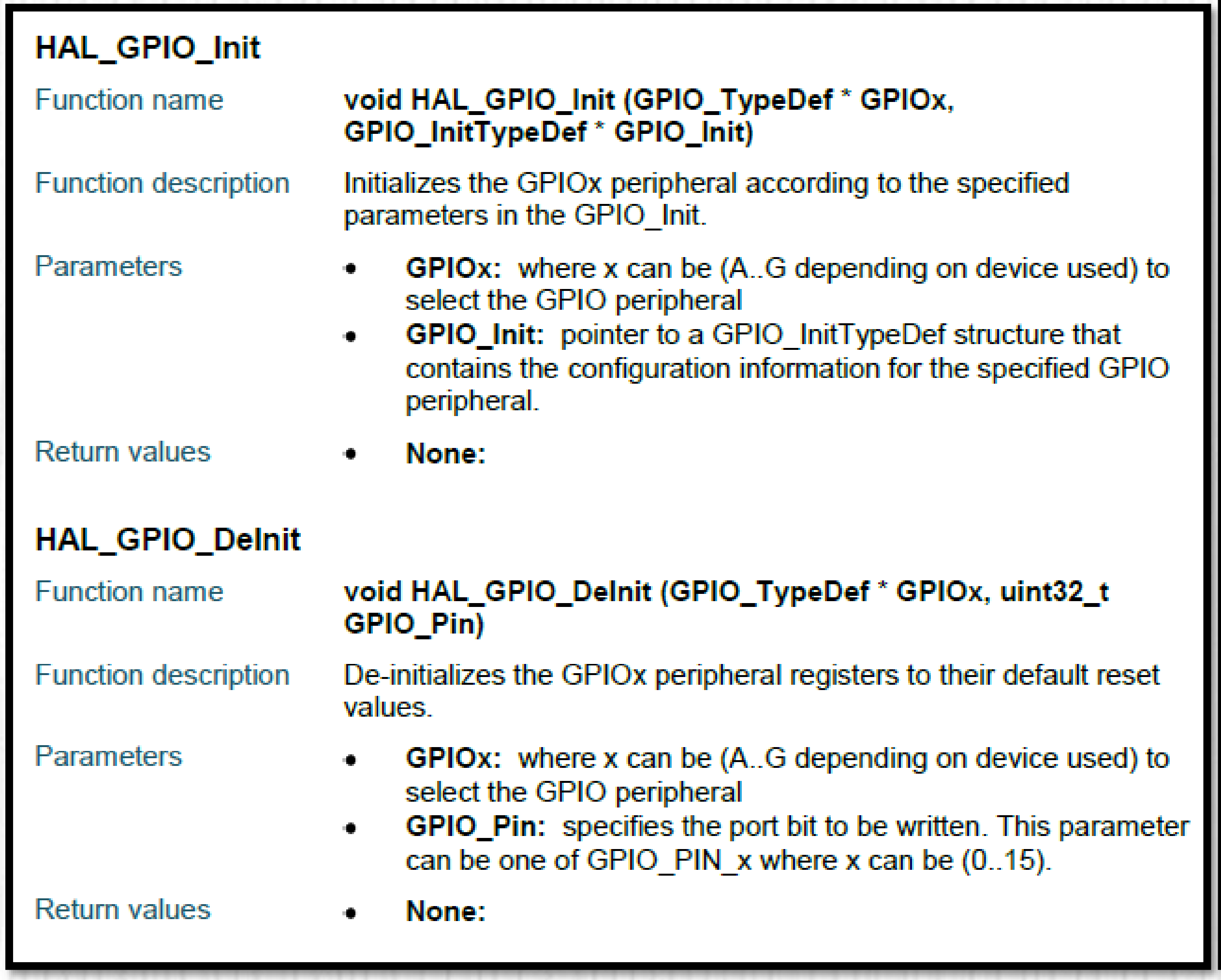
- STM32CubeIDE has generated codes for us according our configuration. The last thing we need to do is flash the LED
- Remember to put our own codes **into the USER CODE comment block**, otherwise, STM32CubeIDE will overwrite them.



```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    HAL_Delay(1000);
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_8);
    HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_2);
  }
  /* USER CODE END 3 */
}
```

4. Programming using HAL APIs

- Functions used frequently

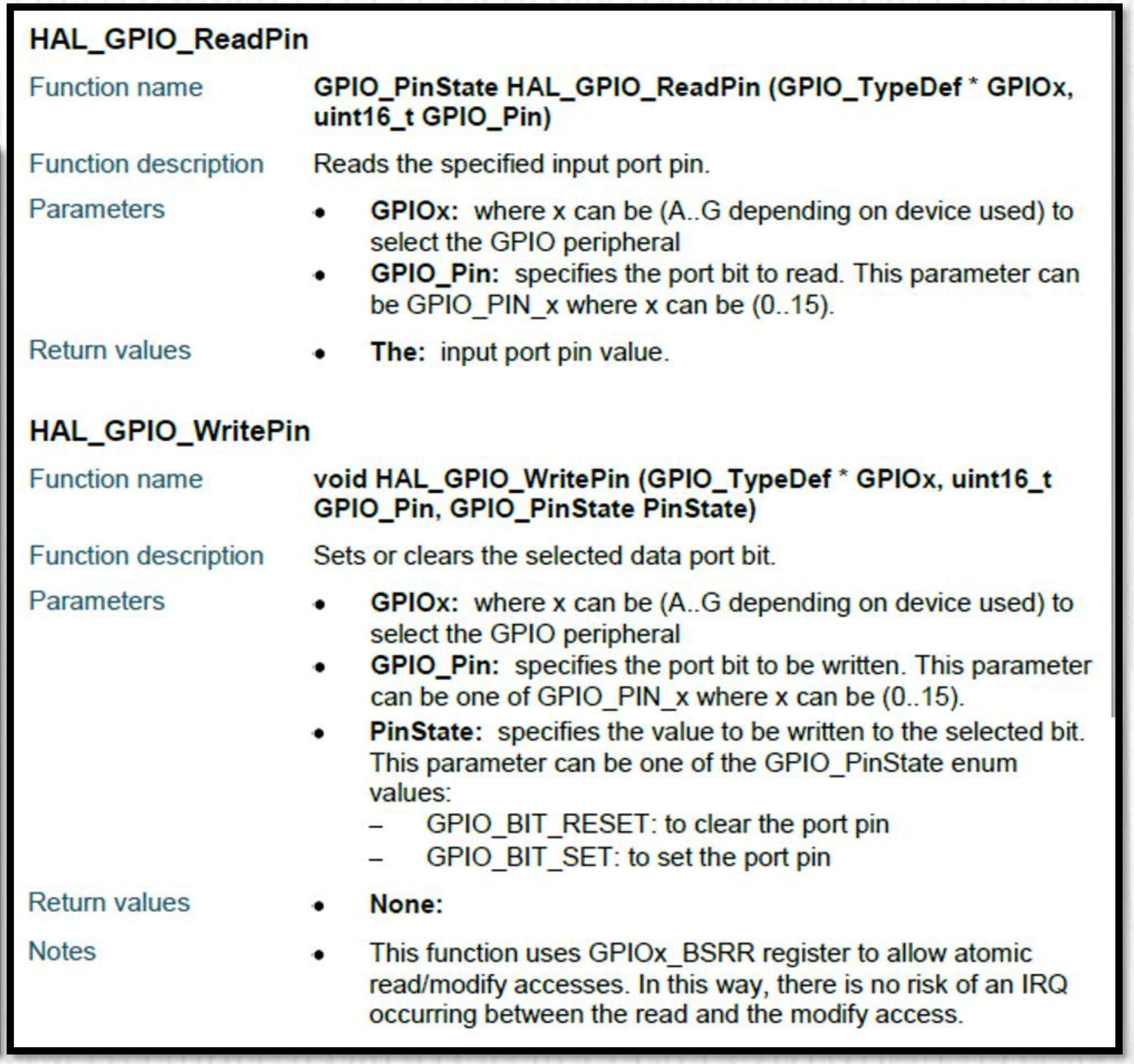


HAL_GPIO_Init

Function name	**void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_Init)**
Function description	Initializes the GPIOx peripheral according to the specified parameters in the GPIO_Init.
Parameters	• **GPIOx:** where x can be (A..G depending on device used) to select the GPIO peripheral • **GPIO_Init:** pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.
Return values	• **None:**

HAL_GPIO_DeInit

Function name	**void HAL_GPIO_DeInit (GPIO_TypeDef * GPIOx, uint32_t GPIO_Pin)**
Function description	De-initializes the GPIOx peripheral registers to their default reset values.
Parameters	• **GPIOx:** where x can be (A..G depending on device used) to select the GPIO peripheral • **GPIO_Pin:** specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15).
Return values	• **None:**



HAL_GPIO_ReadPin

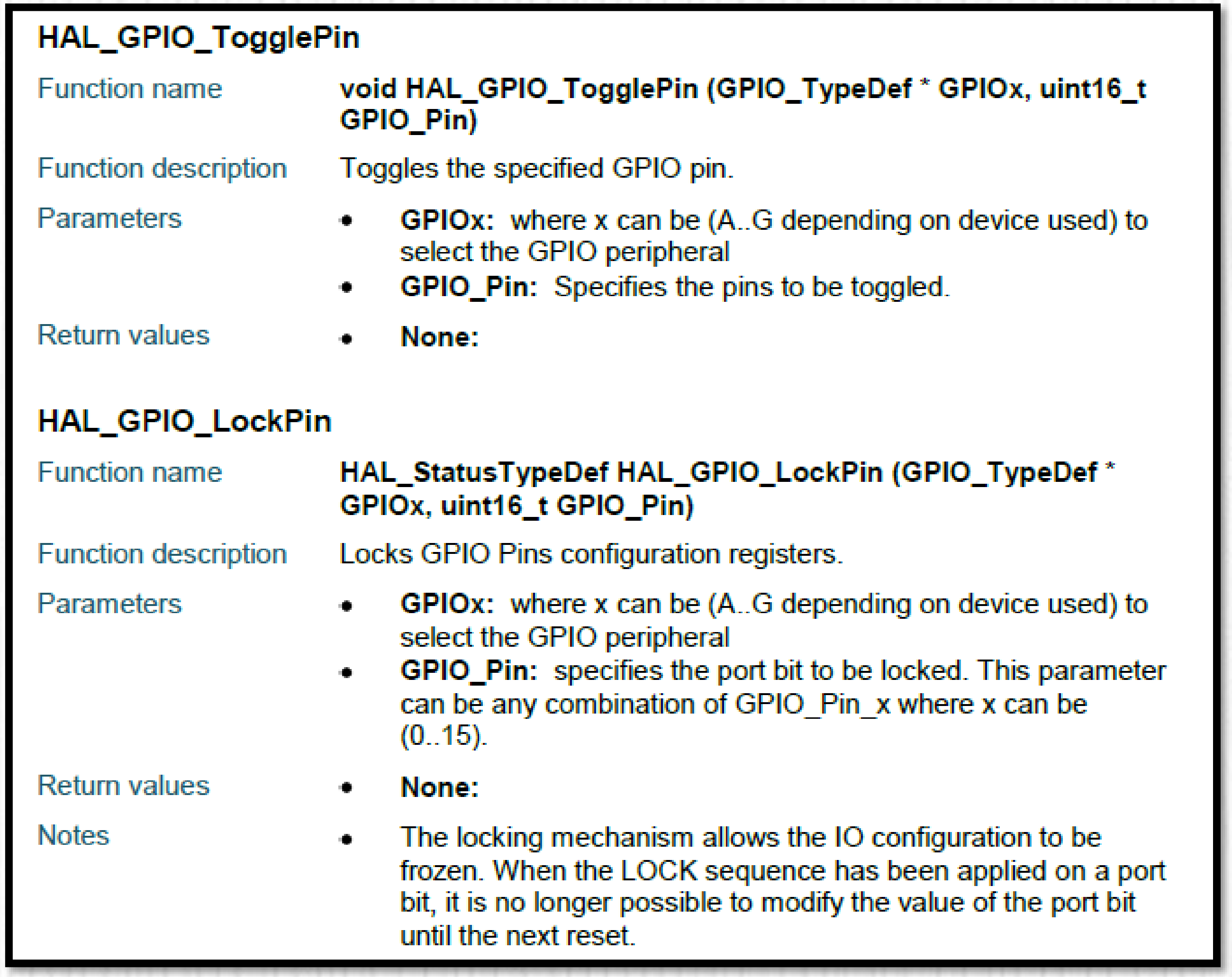
Function name	**GPIO_PinState HAL_GPIO_ReadPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)**
Function description	Reads the specified input port pin.
Parameters	• **GPIOx:** where x can be (A..G depending on device used) to select the GPIO peripheral • **GPIO_Pin:** specifies the port bit to read. This parameter can be GPIO_PIN_x where x can be (0..15).
Return values	• **The:** input port pin value.

HAL_GPIO_WritePin

Function name	**void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)**
Function description	Sets or clears the selected data port bit.
Parameters	• **GPIOx:** where x can be (A..G depending on device used) to select the GPIO peripheral • **GPIO_Pin:** specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15). • **PinState:** specifies the value to be written to the selected bit. This parameter can be one of the GPIO_PinState enum values: – GPIO_BIT_RESET: to clear the port pin – GPIO_BIT_SET: to set the port pin
Return values	• **None:**
Notes	• This function uses GPIOx_BSRR register to allow atomic read/modify accesses. In this way, there is no risk of an IRQ occurring between the read and the modify access.

4. Programming using HAL APIs

- Functions used frequently

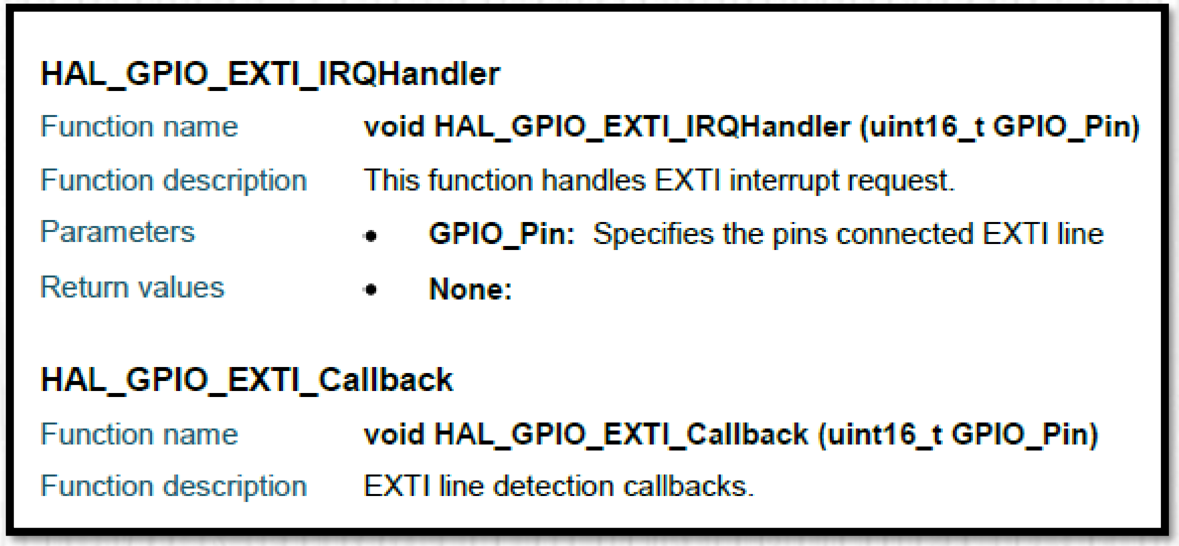


HAL_GPIO_TogglePin

Function name	**void HAL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)**
Function description	Toggles the specified GPIO pin.
Parameters	• **GPIOx:** where x can be (A..G depending on device used) to select the GPIO peripheral • **GPIO_Pin:** Specifies the pins to be toggled.
Return values	• **None:**

HAL_GPIO_LockPin

Function name	**HAL_StatusTypeDef HAL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)**
Function description	Locks GPIO Pins configuration registers.
Parameters	• **GPIOx:** where x can be (A..G depending on device used) to select the GPIO peripheral • **GPIO_Pin:** specifies the port bit to be locked. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).
Return values	• **None:**
Notes	• The locking mechanism allows the IO configuration to be frozen. When the LOCK sequence has been applied on a port bit, it is no longer possible to modify the value of the port bit until the next reset.



HAL_GPIO_EXTI_IRQHandler

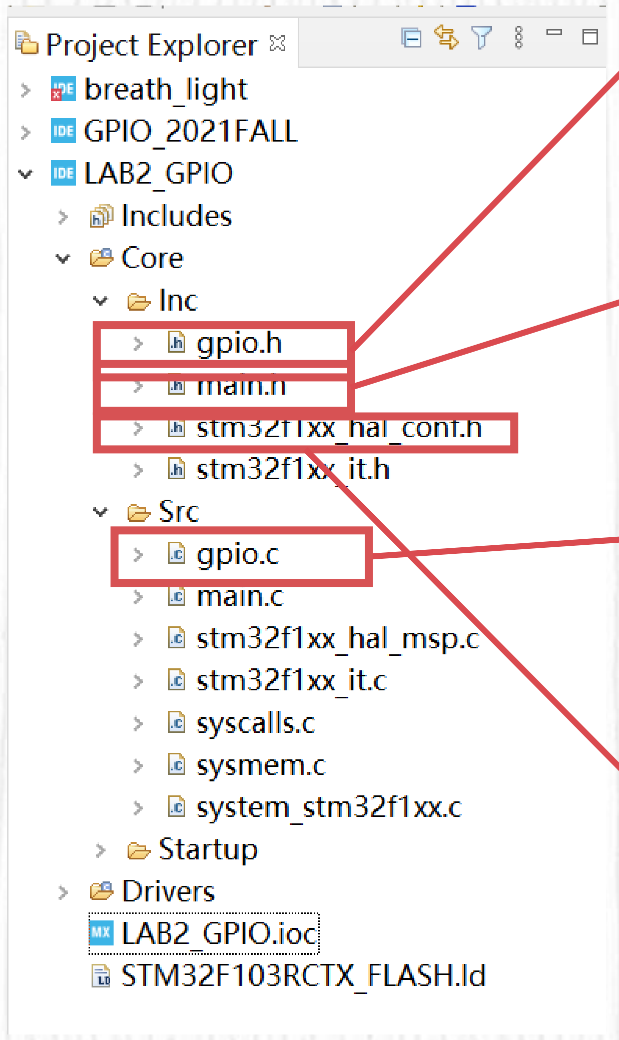
Function name	**void HAL_GPIO_EXTI_IRQHandler (uint16_t GPIO_Pin)**
Function description	This function handles EXTI interrupt request.
Parameters	• **GPIO_Pin:** Specifies the pins connected EXTI line
Return values	• **None:**

HAL_GPIO_EXTI_Callback

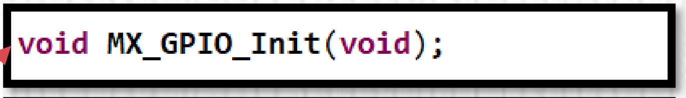
Function name	**void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)**
Function description	EXTI line detection callbacks.

4. Programming using HAL APIs

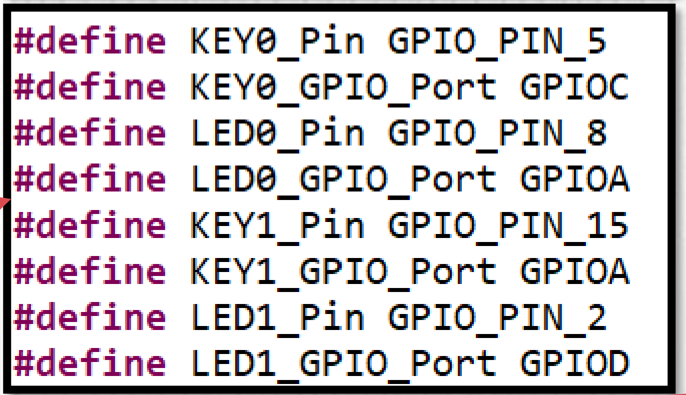
- The generated files and codes



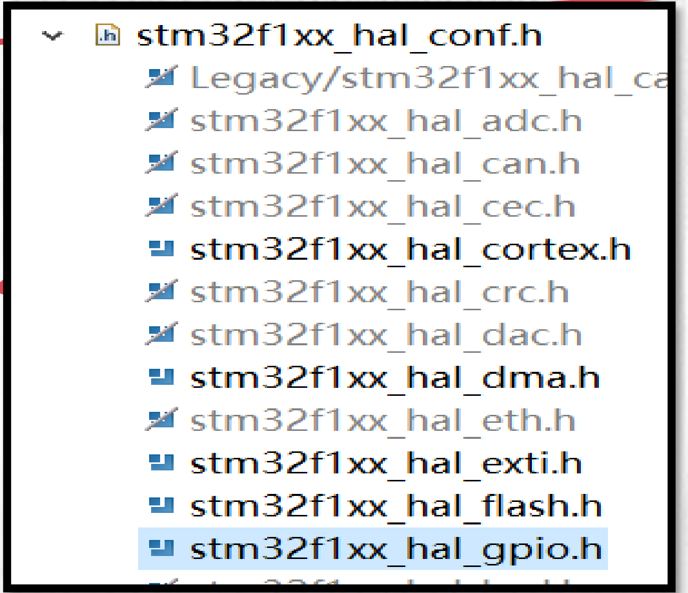
```
Project Explorer
> breath_light
> GPIO_2021FALL
v LAB2_GPIO
  > Includes
  v Core
    v Inc
      > gpio.h
      > main.h
      > stm32f1xx_hal_conf.h
      > stm32f1xx_it.h
    v Src
      > gpio.c
      > main.c
      > stm32f1xx_hal_msp.c
      > stm32f1xx_it.c
      > syscalls.c
      > sysmem.c
      > system_stm32f1xx.c
    > Startup
  > Drivers
  LAB2_GPIO.ioc
  STM32F103RCTX_FLASH.ld
```



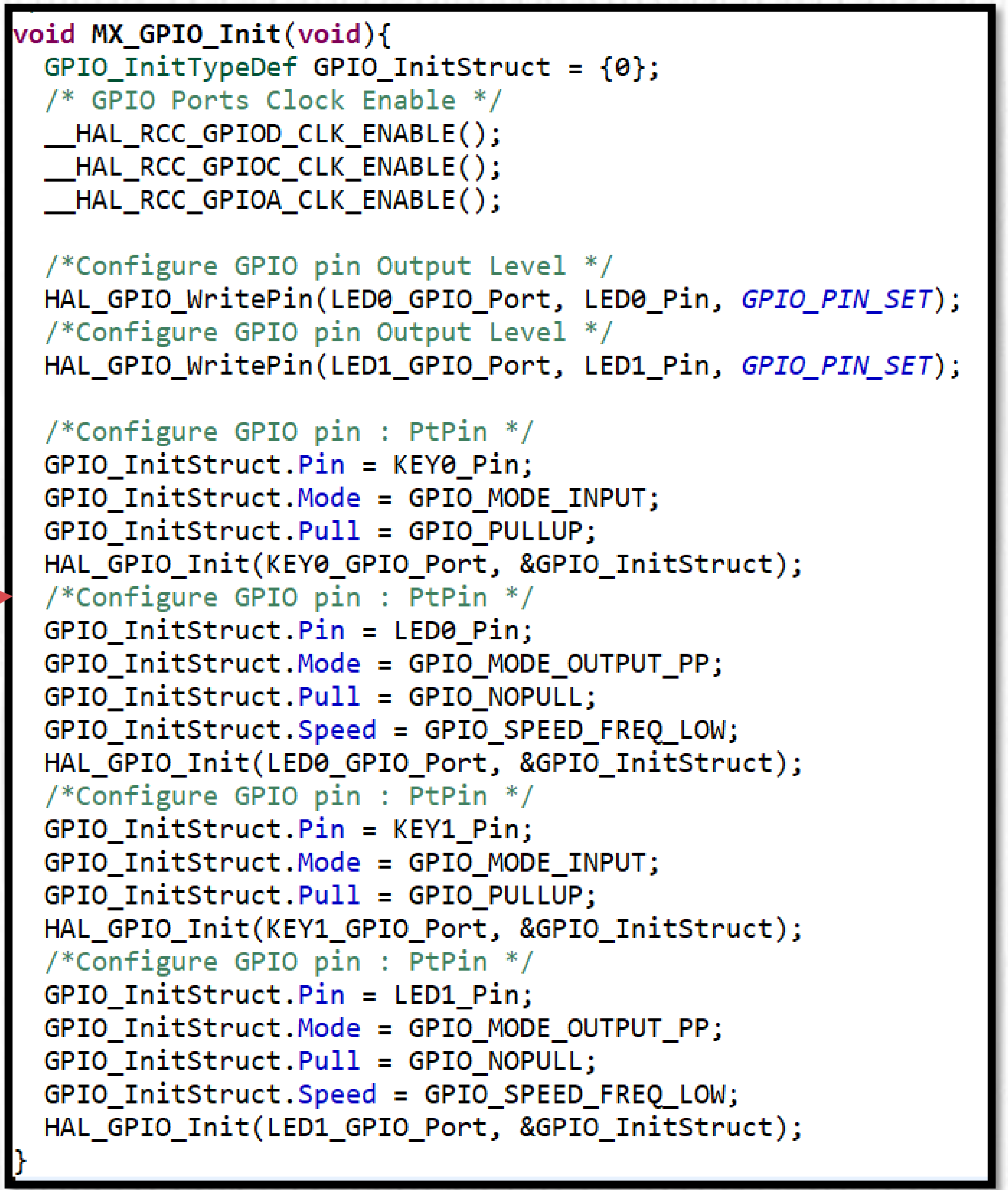
```c
void MX_GPIO_Init(void);
```



```c
#define KEY0_Pin GPIO_PIN_5
#define KEY0_GPIO_Port GPIOC
#define LED0_Pin GPIO_PIN_8
#define LED0_GPIO_Port GPIOA
#define KEY1_Pin GPIO_PIN_15
#define KEY1_GPIO_Port GPIOA
#define LED1_Pin GPIO_PIN_2
#define LED1_GPIO_Port GPIOD
```



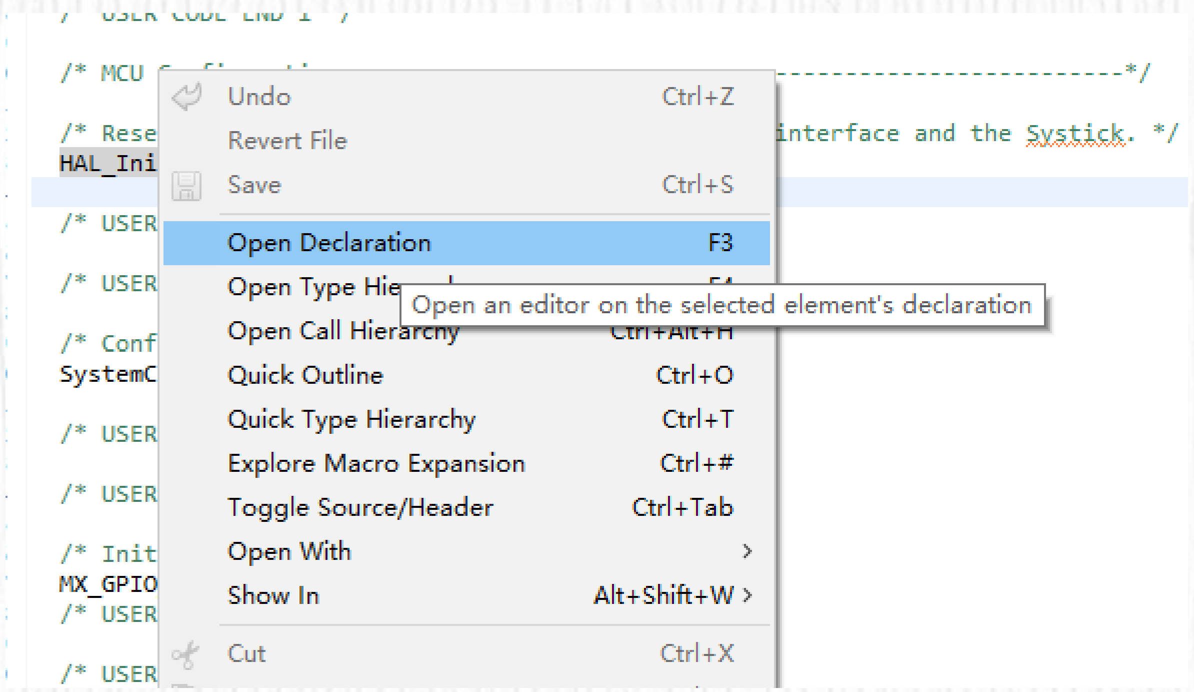
```
v stm32f1xx_hal_conf.h
    Legacy/stm32f1xx_hal_ca
    stm32f1xx_hal_adc.h
    stm32f1xx_hal_can.h
    stm32f1xx_hal_cec.h
    stm32f1xx_hal_cortex.h
    stm32f1xx_hal_crc.h
    stm32f1xx_hal_dac.h
    stm32f1xx_hal_dma.h
    stm32f1xx_hal_eth.h
    stm32f1xx_hal_exti.h
    stm32f1xx_hal_flash.h
    stm32f1xx_hal_gpio.h
```



```c
void MX_GPIO_Init(void){
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LED0_GPIO_Port, LED0_Pin, GPIO_PIN_SET);
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_SET);

  /*Configure GPIO pin : PtPin */
  GPIO_InitStruct.Pin = KEY0_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(KEY0_GPIO_Port, &GPIO_InitStruct);
  /*Configure GPIO pin : PtPin */
  GPIO_InitStruct.Pin = LED0_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LED0_GPIO_Port, &GPIO_InitStruct);
  /*Configure GPIO pin : PtPin */
  GPIO_InitStruct.Pin = KEY1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(KEY1_GPIO_Port, &GPIO_InitStruct);
  /*Configure GPIO pin : PtPin */
  GPIO_InitStruct.Pin = LED1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LED1_GPIO_Port, &GPIO_InitStruct);
}
```

4. Programming using HAL APIs

- Use right click menu ***"Open Declaration"*** to check the definition of functions or structures

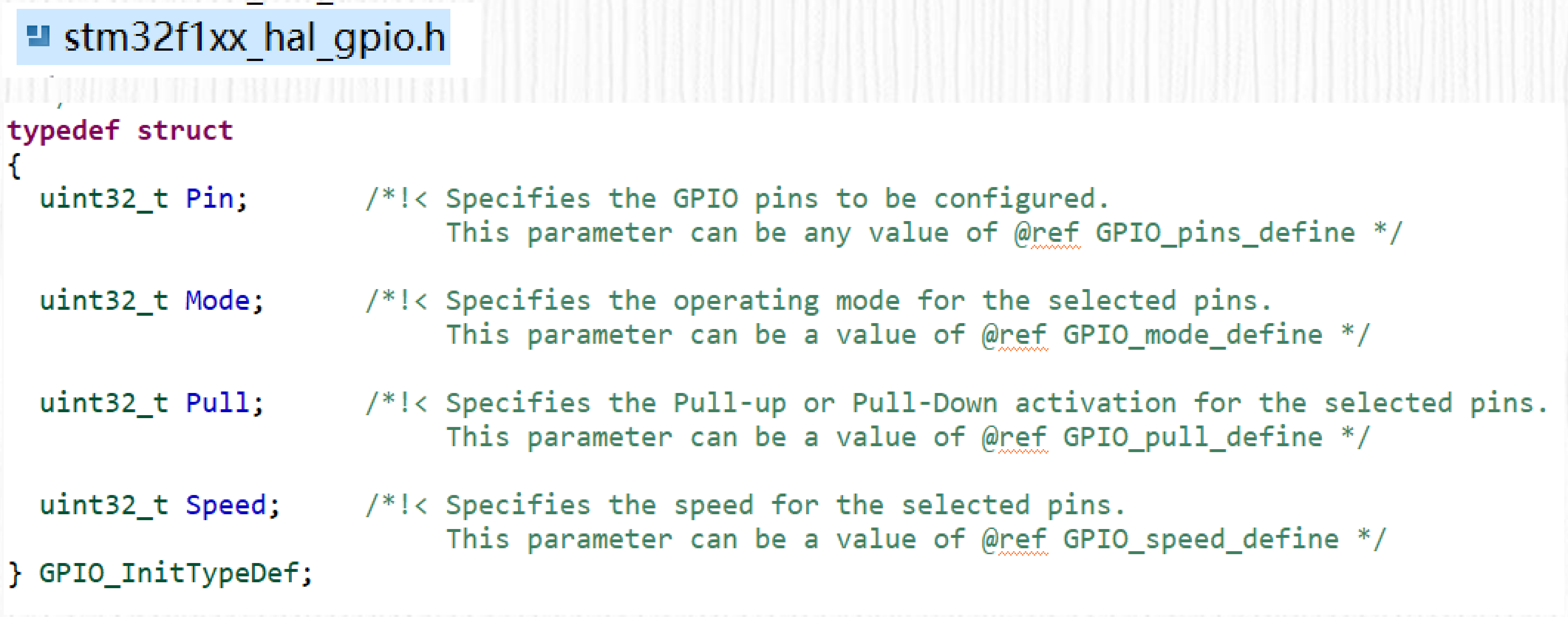


```
/* MCU Configuration---------------------------------------------------------*/

/* Rese                                       interface and the Systick. */
HAL_Ini

/* USER

/* USER

/* Conf
SystemC

/* USER

/* USER

/* Init
MX_GPIO
/* USER

/* USER
```

Undo Ctrl+Z
Revert File
Save Ctrl+S
Open Declaration F3
Open Type Hierarchy F4
Open Call Hierarchy Ctrl+Alt+H
Quick Outline Ctrl+O
Quick Type Hierarchy Ctrl+T
Explore Macro Expansion Ctrl+#
Toggle Source/Header Ctrl+Tab
Open With
Show In Alt+Shift+W
Cut Ctrl+X
Open an editor on the selected element's declaration

4. Programming using HAL APIs

- The structure used frequently

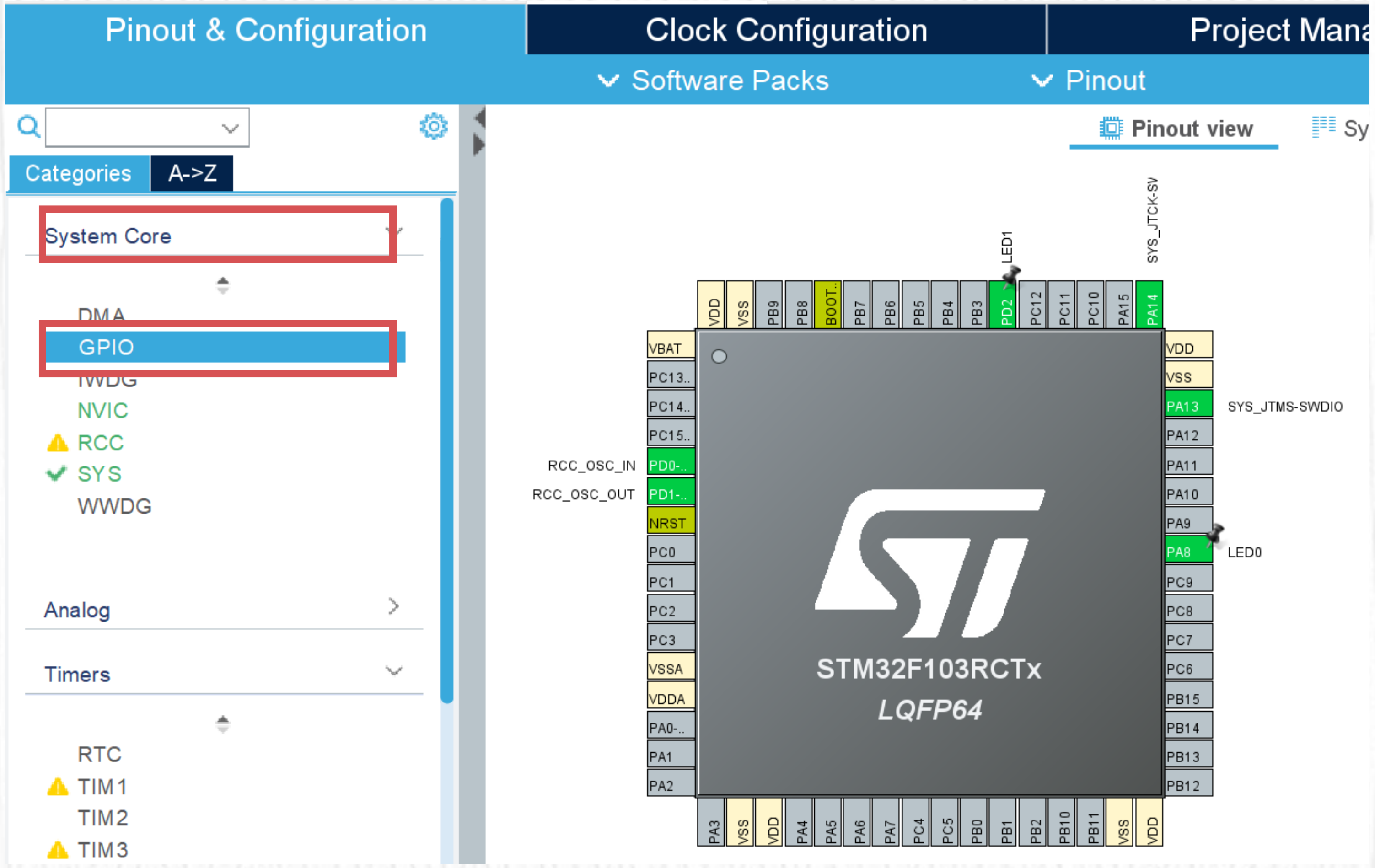


stm32f1xx_hal_gpio.h

```
typedef struct
{
  uint32_t Pin;       /*!< Specifies the GPIO pins to be configured.
                           This parameter can be any value of @ref GPIO_pins_define */

  uint32_t Mode;      /*!< Specifies the operating mode for the selected pins.
                           This parameter can be a value of @ref GPIO_mode_define */

  uint32_t Pull;      /*!< Specifies the Pull-up or Pull-Down activation for the selected pins.
                           This parameter can be a value of @ref GPIO_pull_define */

  uint32_t Speed;     /*!< Specifies the speed for the selected pins.
                           This parameter can be a value of @ref GPIO_speed_define */
} GPIO_InitTypeDef;
```

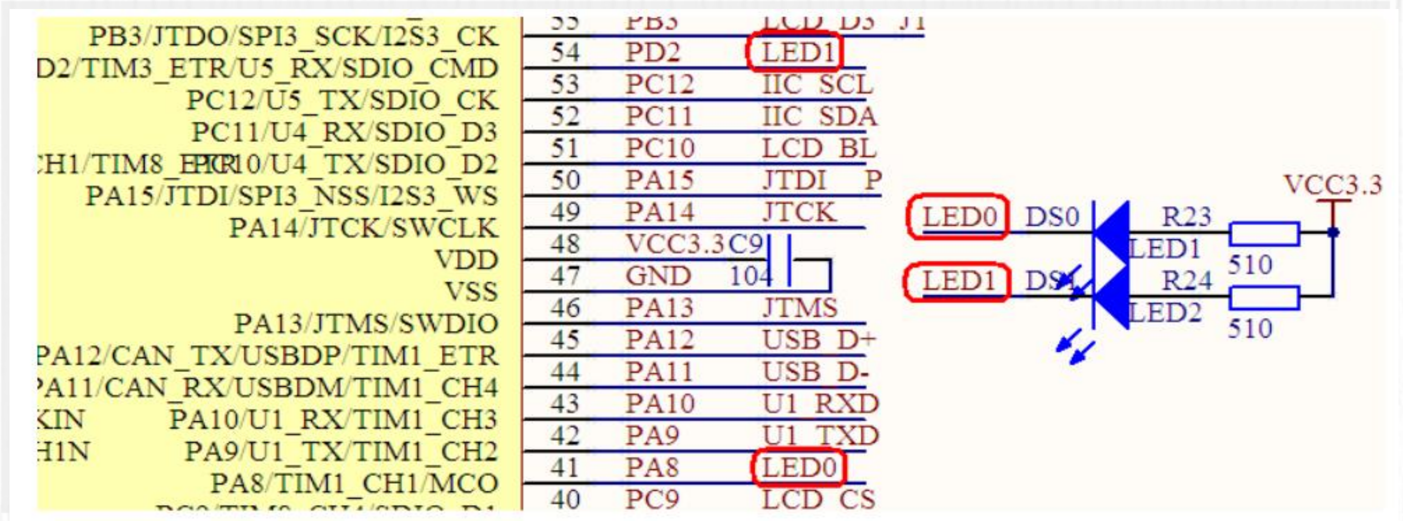
4. Programming using HAL APIs - configure external devices

- GPIO Configuration
  - Still in **Pinout & Configuration -> Categories -> System Core -> GPIO**
  - Set PA8 as GPIO_Output, and rename as LED0
  - Set PD2 as GPIO_Output, and rename as LED1



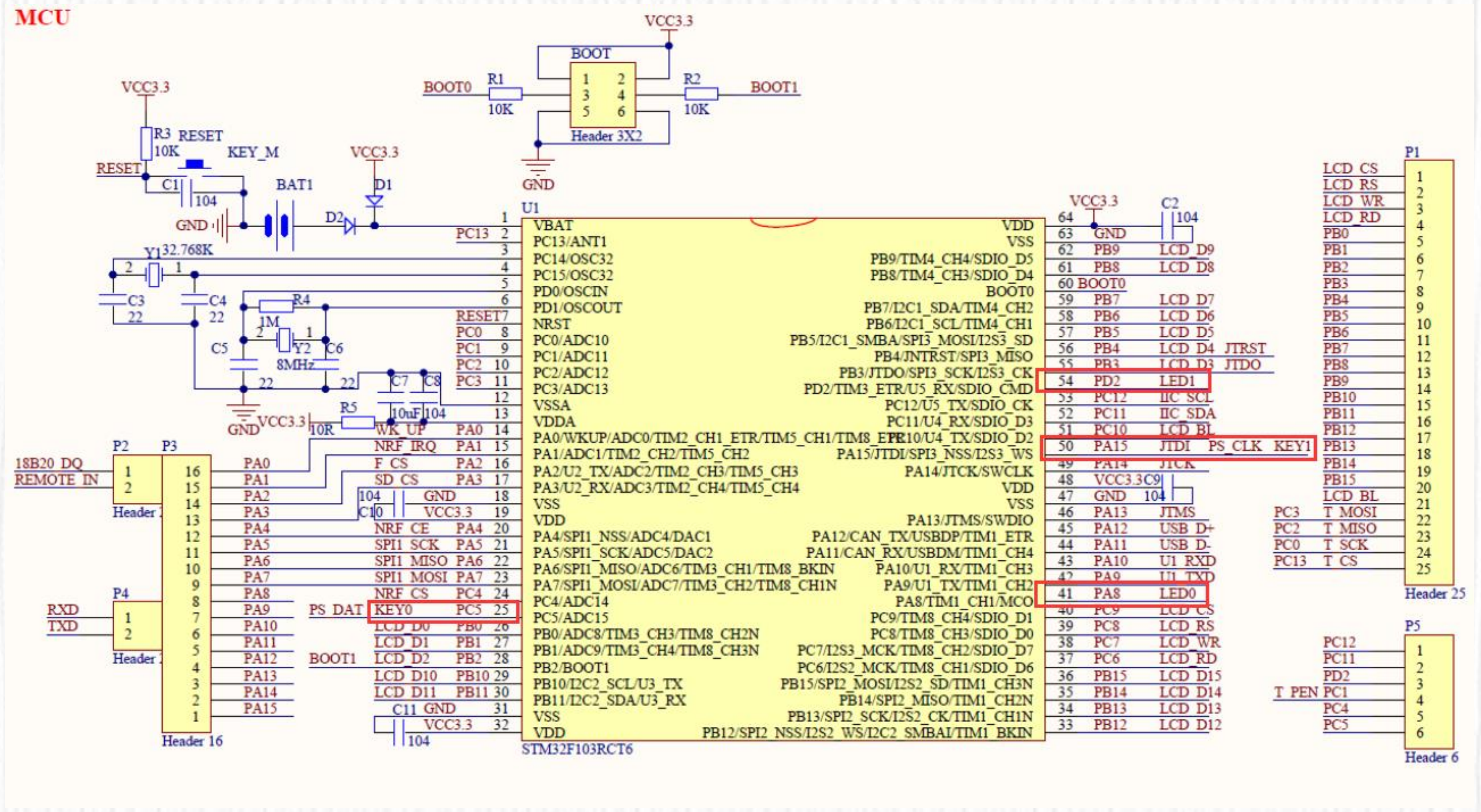
4. Programming using HAL APIs
- configure external devices

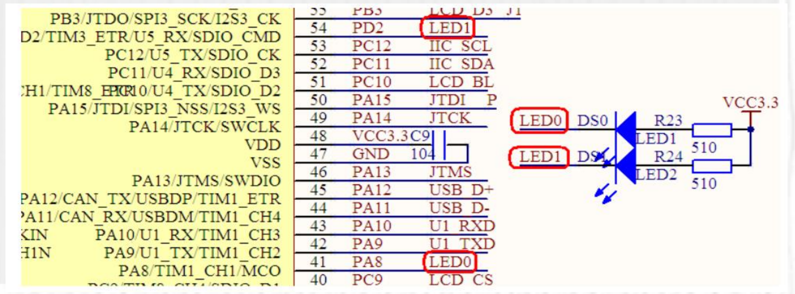
- When PA8 is low, LED0 will light.
- Otherwise, LED0 will extinct.

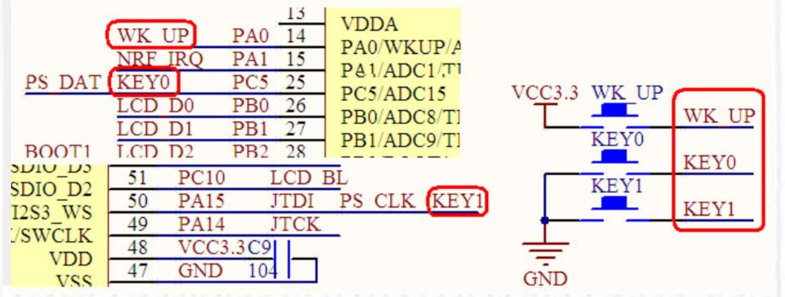


4. Programming using HAL APIs

- GPIO schematic

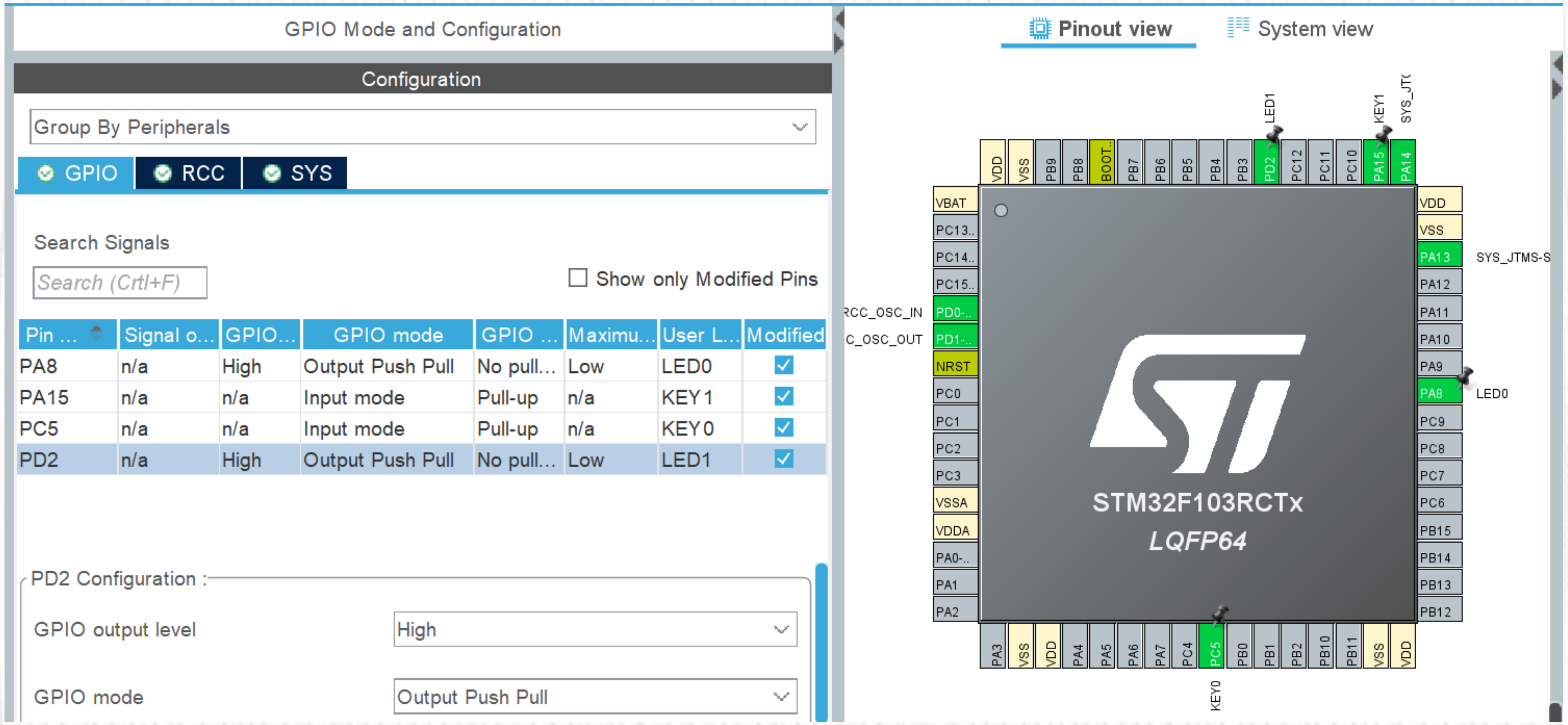






4. Programming using HAL APIs

- GPIO Configuration
  - Find the pins connected to KEY0, KEY1, LED0 and LED1, which are PC5, PA15, PA8 and PD2



GPIO Mode and Configuration

Configuration

Group By Peripherals

GPIO | RCC | SYS

Search Signals

Search (Crtl+F)

☐ Show only Modified Pins

Pin ...	Signal o...	GPIO...	GPIO mode	GPIO ...	Maximu...	User L...	Modified
PA8	n/a	High	Output Push Pull	No pull...	Low	LED0	✓
PA15	n/a	n/a	Input mode	Pull-up	n/a	KEY1	✓
PC5	n/a	n/a	Input mode	Pull-up	n/a	KEY0	✓
PD2	n/a	High	Output Push Pull	No pull...	Low	LED1	✓

PD2 Configuration :

GPIO output level: High

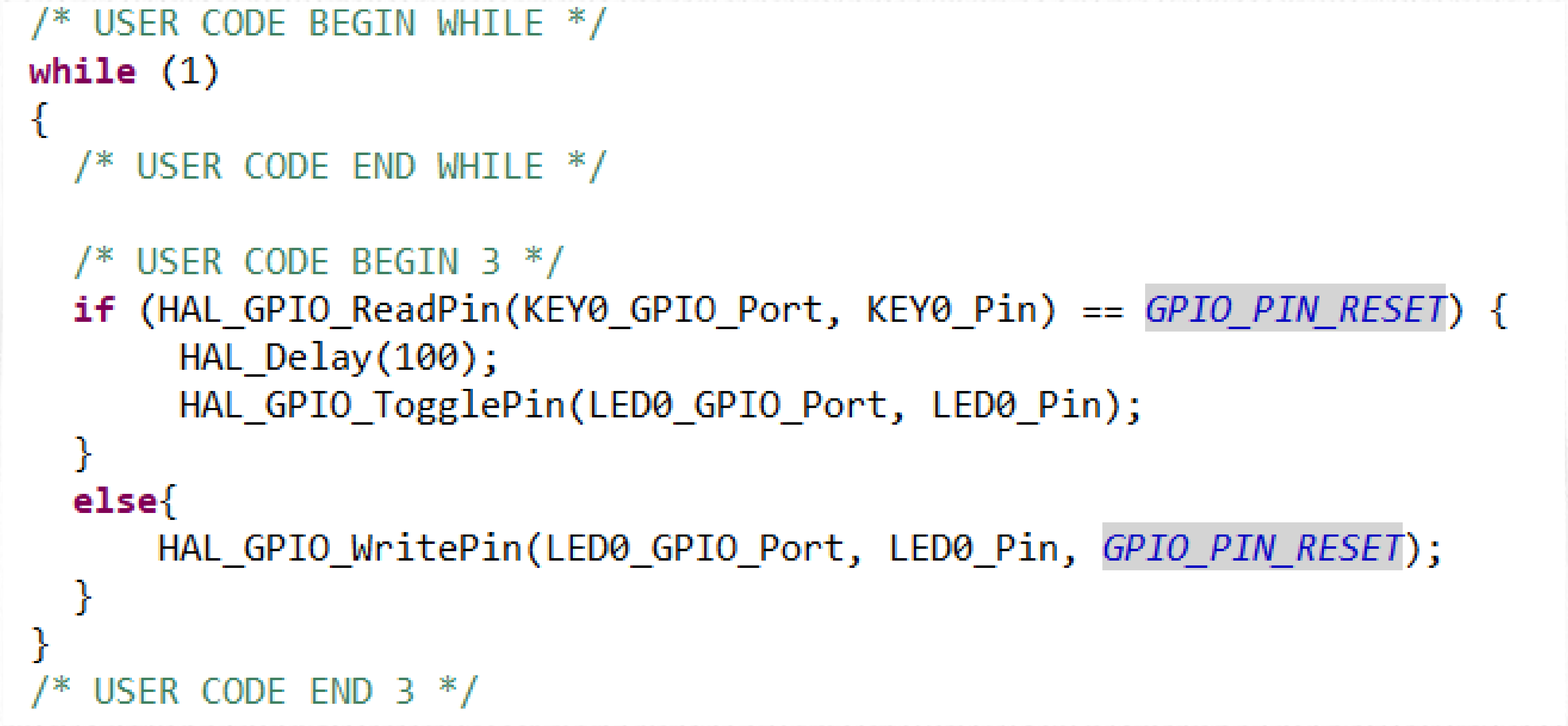
GPIO mode: Output Push Pull

4. Programming using HAL APIs

- Our goal in this lab
  - On the MiniSTM32 board, use KEY0(PC5) to control LED0(PA8).
- Control mode 1
  - As soon as the board is powered on, LED0 is on.
  - When KEY0 is pressed, LED0 starts to flash.
  - When KEY0 is released, LED0 stops flashing and remains on.
- Control mode 2
  - As soon as the board is powered on, LED0 is on.
  - When KEY0 is pressed, LED0 is off.
  - When KEY0 is released, LED0 remains on.

4. Programming using HAL APIs

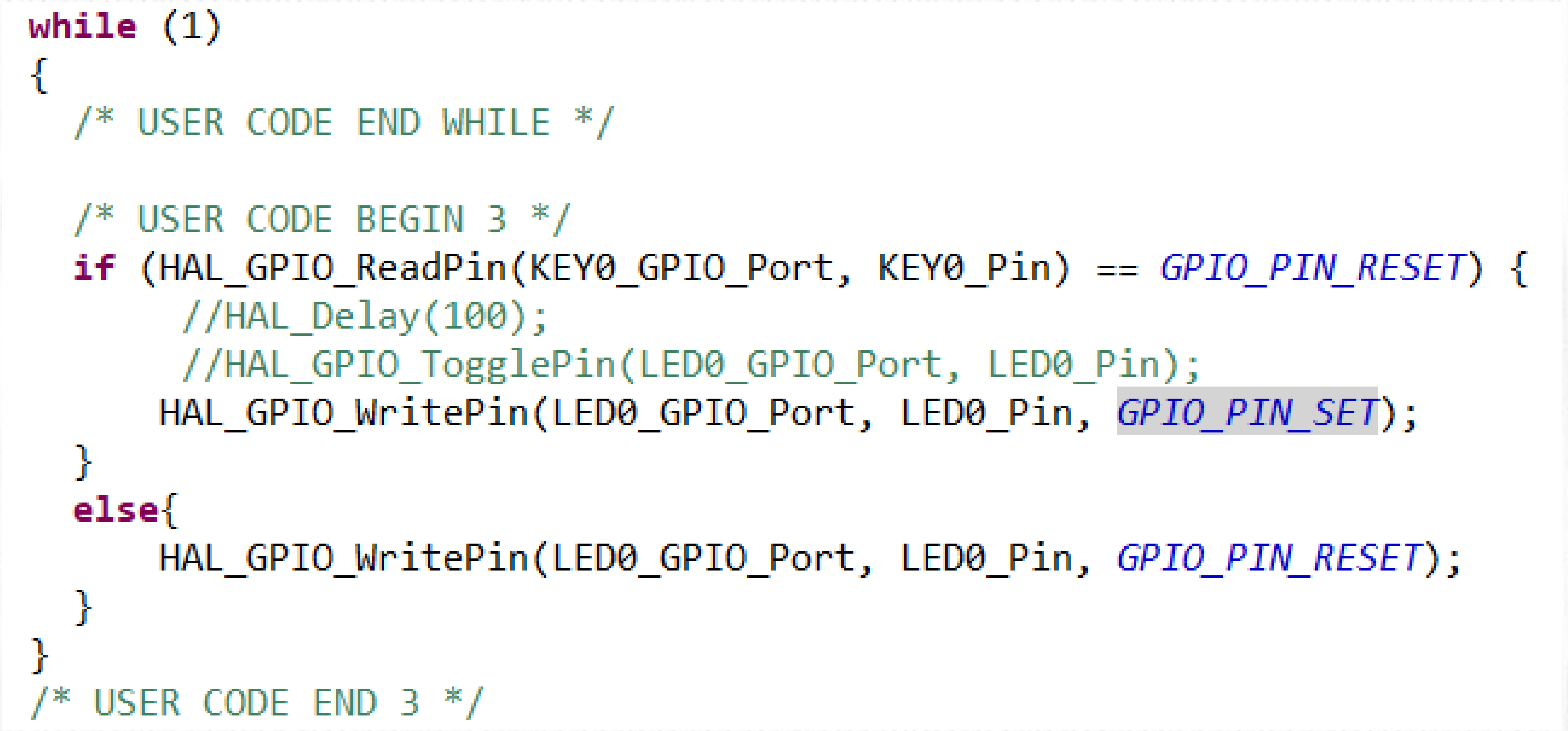
- Add our codes in main.c (control mode 1)



```c
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
  if (HAL_GPIO_ReadPin(KEY0_GPIO_Port, KEY0_Pin) == GPIO_PIN_RESET) {
       HAL_Delay(100);
       HAL_GPIO_TogglePin(LED0_GPIO_Port, LED0_Pin);
  }
  else{
      HAL_GPIO_WritePin(LED0_GPIO_Port, LED0_Pin, GPIO_PIN_RESET);
  }
}
/* USER CODE END 3 */
```

4. Programming using HAL APIs

- Add our codes in main.c (control mode 2)



```c
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
  if (HAL_GPIO_ReadPin(KEY0_GPIO_Port, KEY0_Pin) == GPIO_PIN_RESET) {
       //HAL_Delay(100);
       //HAL_GPIO_TogglePin(LED0_GPIO_Port, LED0_Pin);
      HAL_GPIO_WritePin(LED0_GPIO_Port, LED0_Pin, GPIO_PIN_SET);
  }
  else{
      HAL_GPIO_WritePin(LED0_GPIO_Port, LED0_Pin, GPIO_PIN_RESET);
  }
}
/* USER CODE END 3 */
```

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

05

Practice

5. Practice

- 1. Create a project named Lab2_1_SID, configure the clock and sys mode (RCC and SYS), program with registers to implement one of the control modes shown in next page. (HAL delay function is not allowed to use in register programming.)
- 2. Create a new project named Lab2_2_SID, configure the clock, sys mode and GPIO, then program with HALs, to implement one of the control modes in next page.

5. Practice - Control modes

- Our goal in this practice
  - On the MiniSTM32 board, use KEY1(PA15) to control LED1(PD2).
- Control mode 1
  - As soon as the board is powered on, LED1 is on.
  - When KEY1 is pressed, LED1 starts to flash once per second.
  - When KEY1 is released, LED1 stops flashing and remains on.
- Control mode 2
  - As soon as the board is powered on, LED1 is on.
  - When KEY1 is pressed, LED1 is off.
  - When KEY1 is released, LED1 remains on.