HT32 MCU GPIO User's Guide

D/N: AN0681EN

Introduction

The HT32 MCU's General Purpose Input/Output pins (GPIO) along with the Alternate Function Input/Output Control Unit (AFIO) provide a range of different pin control interfaces. These MCU pins can be configured through these interfaces to communicate with external devices via digital or analog methods. The HT32 MCUs provide a wide array of I/O alternative options to expand the GPIO's flexibility to provide a range of alternative functions. Additionally, the External Interrupt/Event Controller (EXTI) can detect signal states via configurations and quickly execute external interrupt/event handling tasks according to the required conditions.

This application note will introduce the HT32 MCU GPIOs, AFIO and EXTI. It will provide a detailed explanation of registers and API usage for each function, as well as characteristics for the related configuration limitations. The application note is aimed at helping users quickly understand these features, and will also discuss considerations for I/O design, assisting users to complete their relevant circuit design efficiently.

Terms and Abbreviations:

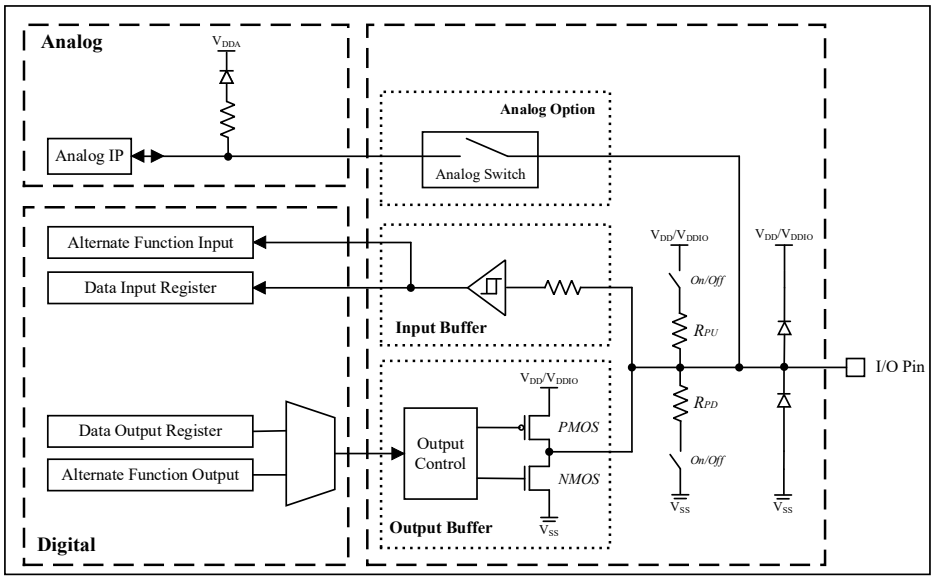
- AFIO: Alternate Function Input / Output Control Unit
- CKCU: Clock Control Unit
- ESD: Electrostatic Discharge
- EFT: Electrical Fast Transient
- EXTI: External Interrupt / Event Controller
- GPIO: General Purpose Input / Output
- NVIC: Nested Vectored Interrupt Controller
- POC: Power On Control

GPIO/AFIO Basic Introduction

This section provides a basic introduction of the GPIOs and the AFIO to assist users to quickly understand these HT32 MCU functions.

GPIO Basic Introduction

The General Purpose Inputs/Outputs are abbreviated as GPIOs. These pins can be controlled and configured by the user using the application program to meet the user's relevant application requirements. The HT32 MCU simplified GPIO architecture diagram is shown in Figure 1.



Note: V_{DDIO} is only supported by some MCUs, refer to the corresponding Datasheet for details.

Figure 1. HT32 MCU Simplified GPIO Architecture Diagram

GPIO pins are usually classified in the format "Pxn", where "x" stands for the port letter (such as A, B, C, etc.) and "n" stands for the pin number (such as 0, 1, 2, etc.). The number of GPIO pins in each HT32 MCU for different package types can be found in the corresponding datasheet. Taking the HT32F52352 64-pin LQFP package as an example, there are up to 51 General Purpose I/O ports, GPIOs named PA0 ~ PA15, PB0 ~ PB8, PB10 ~ PB15, PC0 ~ PC15 and PD0 ~ PD3. Each of the GPIO Ports has a series of related control and configuration registers to implement the following characteristics.

- Input/output direction control
- Schmitt Trigger Input function enable control
- Input weak pull-up/pull-down control
- Output push-pull/Open-Drain enable control
- Output set/reset control
- Output drive current selection
- External interrupt with programmable trigger edge – using the EXTI configuration registers
- Analog input/output configurations – using the AFIO configuration registers

- Alternate function input/output configurations - using the AFIO configuration registers
- Port configuration lock

Each pin in the HT32 MCUs has its own type, such as power supply [P], digital I/O pin [I/O], digital & analog I/O pin [AI/O], etc. The HT32 MCU GPIO architecture list is summarized in Table 1. Refer to the "Pin Assignment" in the corresponding datasheet for pin type confirmation and planning.

Name	Abbreviation	Definition
I/O Type	P	Power Supply
	I	Input
	O	Output
	A	Analog Port
	V_{DD}(1)	V_{DD} Domain
	V_{DDIO}(2)	V_{DDIO} Domain
	BK(1) (3)	Backup Domain
I/O Structure	33V	3.3V Tolerant
	5V	5V Tolerant
	PU(4)	Pull-Up
	PD(4)	Pull-Down

Table 1. HT32 MCU GPIO Architecture List

Note:

1. The V_{DD}/Backup Domain I/O has the ability to ensure stable operation when the MCU core voltage power domain is not supplied. For example, an I/O can still maintain an output High or Low state in the Power-Down mode. If the MCU enters the Power-Down mode correctly, the system information can be saved using the V_{DD} or Backup Domain registers and it can be accessed again when the MCU core voltage power domain is once again restored.
2. For an MCU with a V_{DDIO} power pin, there are I/O pins that can be powered independently, here the V_{DDIO} voltage can be different from V_{DD}.
3. Backup Domain I/O pins have drive current limitations. Taking the HT32F52352 specification as an example, which has a specification of less than 2mA@V_{DD}=3.3V. The other general I/O pins can be set to have various drive current. Refer to the "Special GPIO Configurations and Characteristics" below and the corresponding device datasheet for details.
4. For the HT32 MCUs, typical GPIO pins are usually set to "Input Disable Floating Mode" by default while the specific GPIO pins might default to Input Enable with a pull-up or pull-down resistor. Refer to the "Special GPIO Configuration and Characteristics" section below for details.

The pin assignment and status of each package can be checked in the device datasheet. Figure 2 shows the pin assignment for the HT32F52352 with a 33-pin QFN package type. This figure shows the power supply, I/O pin types and corresponding pins, which helps users to quickly plan and design how to use the MCU pins.

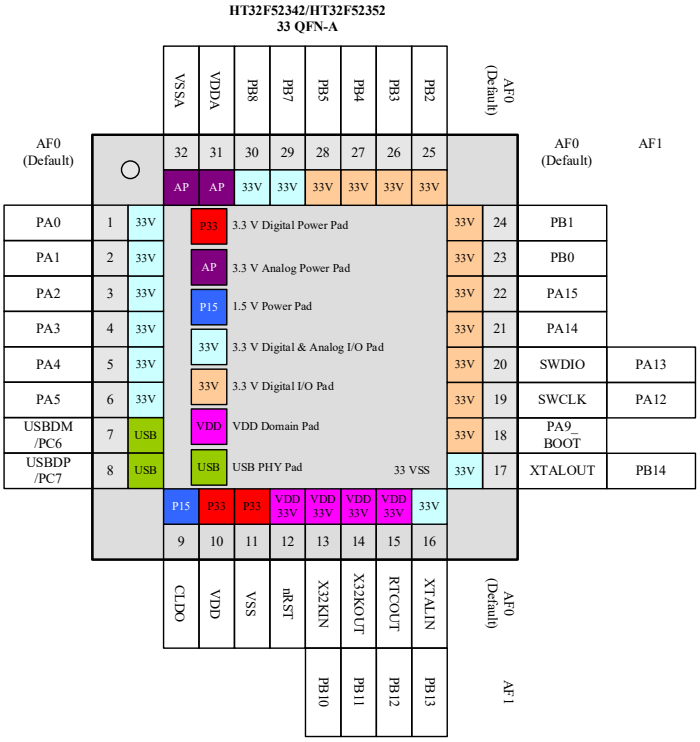


Figure 2. HT32F52352 33-pin QFN Pin Assignment - Refer to Datasheet

Taking the HT32F52352 GPIO as an example, the GPIOs can be divided into the following types according to their voltage domain and digital/analog type:

- Digital I/O = "I/O" (3.3 V Digital I/O Pad)

  For example: PA9 (PA9_BOOT), SWCLK, SWDIO and so on.

- Digital I/O (V_{DD} or V_{BK} Domain) = "I/O (V_{DD}) or I/O (BK)" (V_{DD} or Backup Domain Pad)

  For example: PB12 (RTCOUT), nRST and so on.

- Digital & Analog I/O = "AI/O" (3.3V Digital & Analog I/O Pad)

  For example: PA0, PB13 (XTALIN), PB14 (XTALOUT) and so on.

- Digital & Analog I/O (V_{DD} or V_{BK} Domain) = "AI/O (V_{DD}) or AI/O (BK)" (V_{DD} or Backup Domain Pad)

  For example: PB10 (X32KIN), PB11 (X32KOUT) and so on.

Because the HT32 MCU design and package types are different, the pin classification may not be entirely the same across different MCUs and package types. To view the pin description of the MCU being used, refer to the pin description table in the pin assignment section of the corresponding device datasheet. Taking the HT32F52352 as an example, the pin description table is shown in figure 3, from which the I/O related information can be determined, such as Type, I/O Structure, Output Driving and so on.

Table 4. Pin Description

Pin Number			Pin Name	Type (Note1)	I/O Structure (Note2)	Output Driving	Description
64LQFP	48LQFP	33QFN					Default Function (AF0)
1	1	1	PA0	AI/O	33V	4/8/12/16 mA	PA0
2	2	2	PA1	AI/O	33V	4/8/12/16 mA	PA1
3	3	3	PA2	AI/O	33V	4/8/12/16 mA	PA2
4	4	4	PA3	AI/O	33V	4/8/12/16 mA	PA3
5	5	5	PA4	AI/O	33V	4/8/12/16 mA	PA4
6	6	6	PA5	AI/O	33V	4/8/12/16 mA	PA5
7	7		PA6	AI/O	33V	4/8/12/16 mA	PA6
8	8		PA7	AI/O	33V	4/8/12/16 mA	PA7
9	—		VDD_4	P	—	—	Voltage for digital I/O
10	—		VSS_4	P	—	—	Ground reference for digital I/O
11	9		PC4	AI/O	33V	4/8/12/16 mA	PC4
12	10		PC5	AI/O	33V	4/8/12/16 mA	PC5
13	—		PC8	AI/O	33V	4/8/12/16 mA	PC8
14	—		PC9	AI/O	33V	4/8/12/16 mA	PC9
15	11	7	PC6	I/O	33V	4/8/12/16 mA	PC6
15	11	7	USBDM	AI/O	—	—	USB Differential data bus conforming to the Universal Serial Bus standard.
16	12	8	USBDP	AI/O	—	—	USB Differential data bus conforming to the Universal Serial Bus standard.
16	12	8	PC7	I/O	33V	4/8/12/16 mA	PC7
17	13	9	CLDO	P	—	—	Core power LDO 1.5 V output. It is recommended to connect a 2.2 µF capacitor as close as possible between this pin and VSS_1.
18	14	10	VDD_1	P	—	—	Voltage for digital I/O
19	15	11	VSS_1	P	—	—	Ground reference for digital I/O
20	16	12	nRST	I(BK)	33V_PU	—	External reset pin and external wakeup pin in the Power-Down mode
21	17		VBAT	P	—	—	Battery power input for the backup domain
22	18	13	PB10(4)	AI/O(BK)	33V	< 2 mA	X32KIN
23	19	14	PB11(4)	AI/O(BK)	33V	< 2 mA	X32KOUT
24	20	15	PB12(4)	I/O(BK)	33V	< 2 mA	RTCOUT

Figure 3. HT32F52352 Pin Description - Refer to the Datasheet

The electrical characteristics of the HT32 MCU I/Os can be obtained from the device datasheet. Taking the HT32F52352 as an example, the Electrical Characteristics chapter describes multiple characteristics, such as Absolute Maximum Ratings, Recommended DC Operating Conditions, etc., as shown in Figure 4. Refer to the following "GPIO" section for detailed GPIO information.

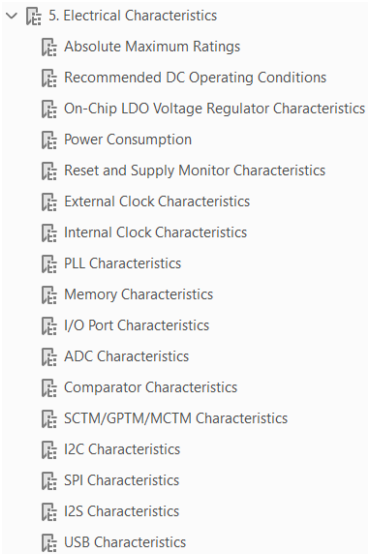


Figure 4. HT32F52352 Electrical Characteristics List - Refer to the Datasheet

AFIO Basic Introduction

In order to expand the flexibility of the GPIOs or the usage of the peripheral functions, each I/O pin can be configured to have different alternative functions such as GPIO, UART and ADC functions by programming the GPxCFGLR or GPxCFGHR registers, where x represents the different port name: A, B, C, etc. According to the resource and application requirements, suitable pin-out locations can be selected using the AFIO mechanism.

Taking the HT32F52352's AFIO as an example, the characteristics are as follows:

- APB slave interface for register access
- EXTI source selection
- Configurable pin function for each GPIO - up to sixteen alternative functions on each pin
- AFIO lock mechanism

The AFIO block diagram is shown in Figure 5.

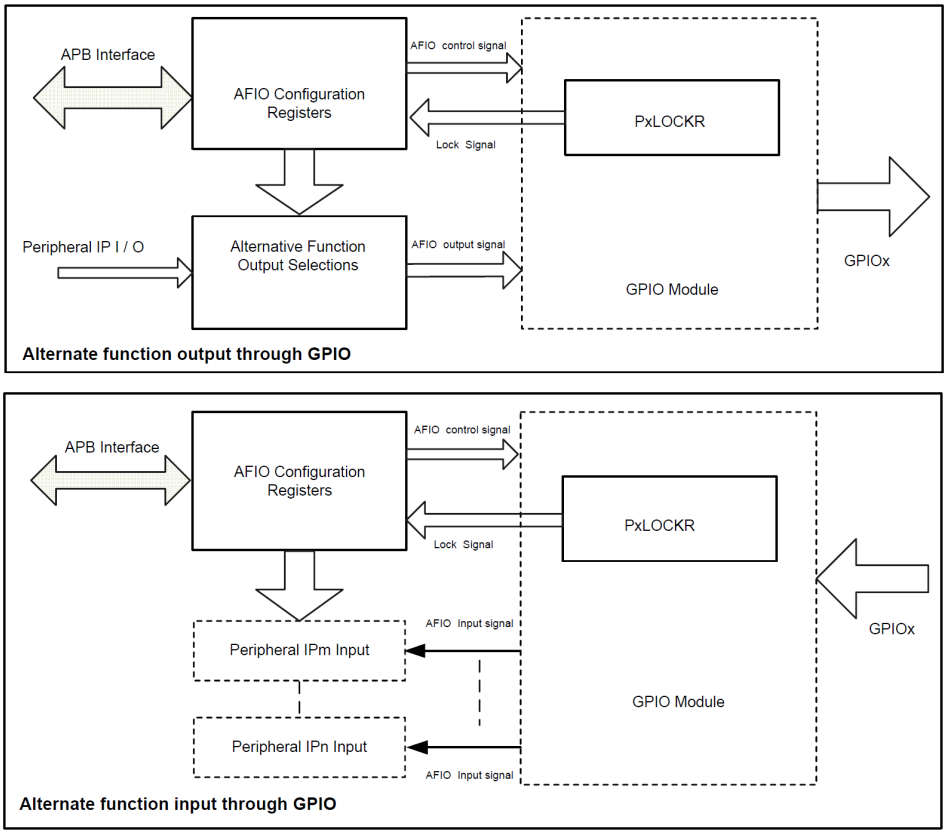


Figure 5. AFIO Block Diagram

Taking the HT32F52352 as an example to illustrate the alternative functions. Up to sixteen alternative functions can be chosen for each I/O pad by setting the PxCFGn[3:0] field in the GPxCFGLR or GPxCFGHR registers (x = A, B, C, etc. n = 0, 1, 2, etc.) The following description shows the setting of the PxCFGn[3:0] field to select 16 alternative functions (AF0 to AF15).

- PxCFGn[3:0] = 0000: The default function (after a reset, AF0)
- PxCFGn[3:0] = 0001: Alternative Function 1 (AF1)

- PxCFGn[3:0] = 0010: Alternative Function 2 (AF2)

…

- PxCFGn[3:0] = 1110: Alternative Function 14 (AF14)
- PxCFGn[3:0] = 1111: Alternative Function 15 (AF15)

The HT32F52352 's Alternative Function Mapping Example Table is shown in Figure 6. From the table, it can be seen that if AF2 is selected, the alternative function is ADC, while if AF5 is selected, the alternative function is SPI, etc. If the N/A item is selected, which means that the alternative function is not available, this pin will be defined as default alternative function (AF0).

Note: A complete table of alternative functions corresponding to the I/O can be obtained from the datasheet of each MCU, as shown in Figure 7.

Refer to the AFIO section below for a detailed AFIO description.

AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
System Default	GPIO	ADC	CMP	MCTM /GPTM	SPI	USART /UART	I²C	SCI	EBI	I²S	N/A	N/A	SCTM	N/A	System Other

Figure 6. HT32F52352 AFIO Selection for Peripheral Map - Refer to the User Manual

Package			Alternate Function Mapping														
			AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
64 LQFP	48 LQFP	33 QFN	System Default	GPIO	ADC	CMP	MCTM /GPTM	SPI	USART /UART	I²C	SCI	EBI	I2S	N/A	N/A	SCTM	N/A	System Other
1	1	1	PA0		ADC_IN0		GT1_CH0	SPI1_SCK	USR0_RTS	I2C1_SCL	SCI0_CLK		I2S_WS					
2	2	2	PA1		ADC_IN1		GT1_CH1	SPI1_MOSI	USR0_CTS	I2C1_SDA	SCI0_DIO		I2S_BCLK					
3	3	3	PA2		ADC_IN2		GT1_CH2	SPI1_MISO	USR0_TX				I2S_SDO					
4	4	4	PA3		ADC_IN3		GT1_CH3	SPI1_SEL	USR0_RX				I2S_SDI					
5	5	5	PA4		ADC_IN4		GT0_CH0	SPI0_SCK	USR1_TX	I2C0_SCL	SCI1_CLK							
6	6	6	PA5		ADC_IN5		GT0_CH1	SPI0_MOSI	USR1_RX	I2C0_SDA	SCI1_DIO							
7	7		PA6		ADC_IN6		GT0_CH2	SPI0_MISO	USR1_RTS		SCI1_DET							
8	8		PA7		ADC_IN7		GT0_CH3	SPI0_SEL	USR1_CTS				I2S_MCLK					
9			VDD_4															
10			VSS_4															
11	9		PC4		ADC_IN8		GT0_CH0	SPI1_SEL	UR0_TX	I2C1_SCL		EBI_A19				SCTM0		
12	10		PC5		ADC_IN9		GT0_CH1	SPI1_SCK	UR0_RX	I2C1_SDA		EBI_A20				SCTM1		
13			PC8		ADC_IN10		GT0_CH2	SPI1_MOSI				EBI_A0						
14			PC9		ADC_IN11		GT0_CH3	SPI1_MISO				EBI_A1						

Figure 7. HT32F52352 AFIO Pin Assignment - Refer to the Datasheet

Additionally, the GPIO pins also can be selected as EXTI trigger sources by configuring the EXTInPIN[3:0] field in the AFIO ESSRn register to trigger an interrupt or event. Refer to the chapter "I/O External Interrupt/Event Controller (EXTI)" below for more information about the EXTI.

GPIO

This chapter will introduce the HT32 MCU GPIO functions in detail, including the GPIO modes, registers, APIs, configuration priorities and limitations. It also introduces the GPIO Pin Lock and special GPIO features, allowing users to gain a better understanding of the HT32 MCU GPIO functions.

GPIO Mode Introduction

The GPIO modes are mainly divided into two types: input and output, while the breakdown of GPIO modes can be divided into types: Input, Output, Push-Pull, Open-Drain, Pull-Up, Pull-Down. The following provides an explanation of each.

Input / Output

As the name implies, the most important General Purpose Input/Output function is that of input/output communication with the external environment. In other words, it is able to communicate by connecting to external devices through the I/O pins.

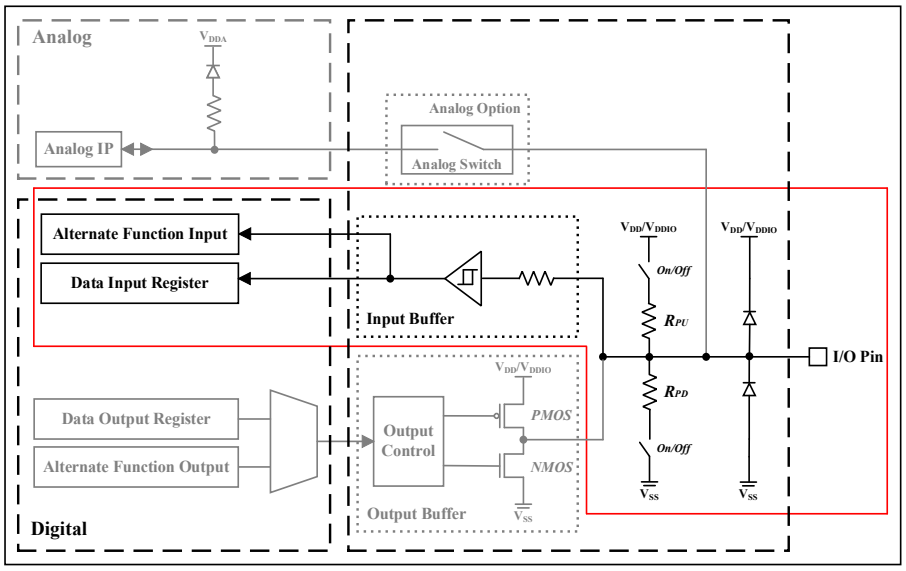
- Input

  When the HT32 MCU I/O pins are configured as inputs, one of the following three options must be selected. The Input area follows the HT32 MCU simplified GPIO Architecture Diagram and the Input area can be seen from the red box section in Figure 8:

  - Internal Pull-Up Input: The HT32 MCU uses a pull-up resistor to ensure that the input signal has a well-defined logic level when it is floating.
  - Internal Pull-Down Input: The HT32 MCU uses a pull-down resistor to ensure that the input signal has a well-defined logic level when it is floating.
  - Floating Input: The I/O voltage level will follow the external signal. When there is no external signal, the Schmitt trigger will switch randomly between logic levels caused by external noise. This behavior will increase overall power consumption.

  Therefore, when the I/O pins are configured as inputs, the I/Os have the following characteristics:

  - The Output Buffer is turned off.
  - The Schmitt Trigger will automatically turn on when the input is enabled and automatically turn off when the input is disabled.
  - The pull-up (R_{PU}) and pull-down (R_{PD}) resistors are configured according to the PxPUR and PxPDR register, where x represents the different port name: A, B, C, etc.
  - The input data register samples the signal on the I/O pin every AHB Clock Cycle.
  - The I/O status can be obtained by reading the PxDINR register, where x represents a different port name: A, B, C, etc.



Note: V_{DDIO} is only supported by some MCUs, refer to the corresponding Datasheet for details.

Figure 8. HT32 MCU Simple GPIO Architecture Diagram - Input Area

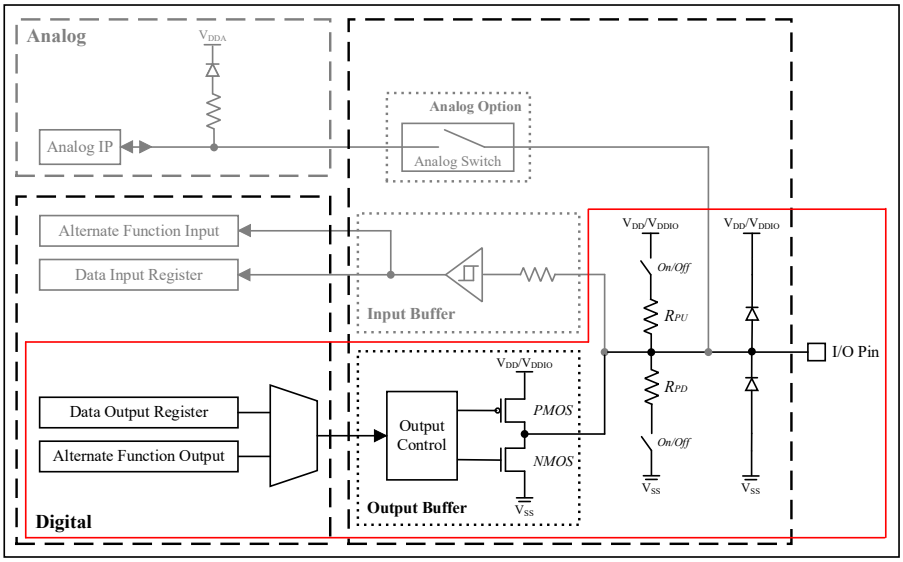
- Output

  When the HT32 MCU I/O pins are configured as outputs, the Output Buffer must be set to either Push-Pull or Open-Drain output. Refer to the "Push-Pull" / "Open-Drain" chapter for details. The output area follows the HT32 MCU simplified GPIO Architecture Diagram and the output area can be seen from the red box section of Figure 9:

  Therefore, when the I/O pins are configured as outputs, the I/Os have the following characteristics:

  - The output buffer can be configured to be in a Push-Pull or Open-Drain mode.
  - The output drive current is configured according to the PxDRVR register, where x is the different port name: A, B, C, etc.

  Note: The configurable GPIO Port output driving current capability may vary depending on the MCU and pin. Detailed information about this can be obtained from the Pin Description table in the corresponding device datasheet. For example, in the HT32F52352, most of the I/Os can be set to have a 4/8/12/16mA sink/source current, while the I/Os in the Backup Domain only have lower output driving current.



Note: V_{DDIO} is only supported by some MCUs, refer to the corresponding Datasheet for details.

Figure 9. HT32 MCU Simplified GPIO Architecture Diagram - Output Area

Push-Pull / Open-Drain

The following will introduce the Push-Pull and Open-Drain output configurations. The logic state for an output can be divided into "High" and "Low". When the output is driven to a low state, the I/O pin will absorb current from the load, this is called a sink current. When the output is driven to a high state, the I/O pin provides current to the load, this is called a source current. The next section will explain the specific behaviors when the output is driven to High/Low states with the "Push-Pull" and "Open-Drain" options.

- Push-Pull

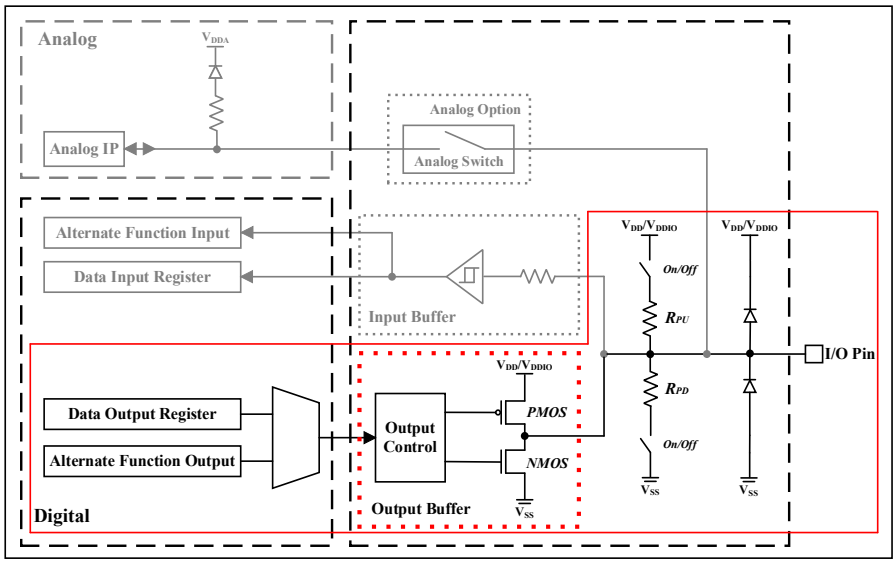
  When a HT32 MCU I/O is configured as a Push-Pull type, the Output Buffer is shown within the red dotted line in Figure 10. Here the Output Buffer uses two transistors, one PMOS and one NMOS. It is only when each transistor is turned on that the output is driven to the appropriate level, where the output voltage is determined by the V_{DD} or V_{DDIO} power supply.

  - When the output is driven to a high state, the top PMOS transistor is turned on.
  - When the output is driven to a low state, the lower NMOS transistor is turned on.

  Note: The output data is set through the PxDOUTR register, where x represents the different port name: A, B, C, etc.

  Writing a "0" to the relevant bit to enable the NMOS transistor forces the I/O pin to be connected to GND.

  Writing a "1" to the relevant bit to enable the PMOS transistor forces the I/O pin to be connected to V_{DD} or V_{DDIO}.



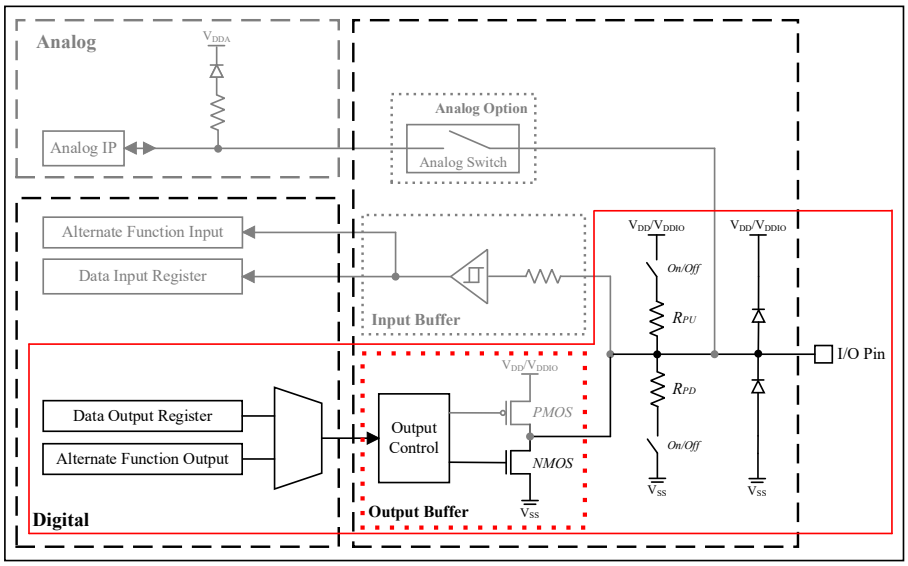
Note: V_{DDIO} is only supported by some MCUs, refer to the corresponding Datasheet for details.

Figure 10. HT32 MCU Simplified GPIO Architecture Diagram - Output Area - Push-Pull

- Open-Drain

  When a HT32 MCU I/O is configured as an Open-Drain, the Output Buffer is shown within the red dotted line in Figure 11. Here the Output Buffer uses an NMOS transistor and is composed of Output Control and an NMOS transistor. Since the Open-Drain structure does not have a PMOS transistor connected to V_{DD} or V_{DDIO} as in the Push-Pull structure, it is necessary to connect a pull-up resistor, $R_{Pull-Up}$, to the output, otherwise the pin will be in a floating state when a high output condition is setup. Examples which use an Open-Drain structure would be I^2C, SMbus, etc.

  - When the output is driven to a high state, it is necessary to connect a pull-up resistor ($R_{Pull-Up}$) in the circuit. The driving capability of the high level and the level conversion speed is determined by the external pull-up resistor value. Generally speaking, the smaller the resistance value of the pull-up resistor, the faster the level conversion speed, but will come at a cost of higher power consumption.
  - When the output drive is driven to a low state, the lower NMOS transistor is turned on to connect the I/O pin to GND, thereby generating a low level signal.



Note: V_{DDIO} is only supported by some MCUs, refer to the corresponding Datasheet for details.

Figure 11. HT32 MCU Simplified GPIO Architecture Diagram - Output Area Open-Drain

Pull-Up / Pull-Down

The following will introduce the pull-up / pull-down resistor. There are three states for the HT32 MCU GPIOs, High impedance, High (V_{DD} or V_{DDIO}) and Low (GND). "High impedance" can be written as "Hi-Z", which means that the I/O is in a high impedance state. Here, the I/O pin may be high or low, which depends on the current usage environment. This I/O pin will be connected to an external device, but it will not affect the normal output of the external device. When the I/O pin is not connected to an external device, it is necessary for it to be given a definite state, which can be maintained at a reference level by a pull-up or pull-down resistor (*1) to prevent erroneous operations. Pull-Up / Pull-Down (*2) is the method used to manage this state.

Note:

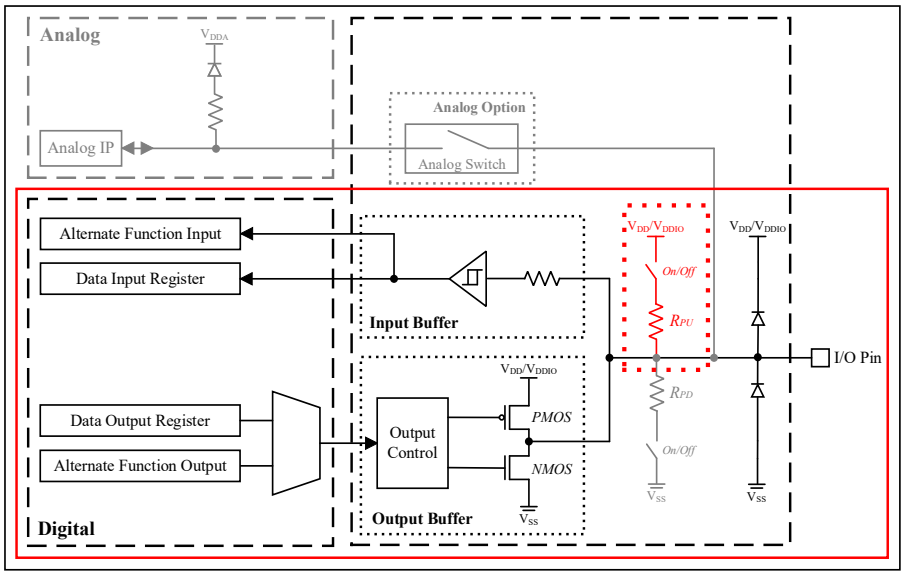
*1. The specifications of the HT32 MCU internal pull-up and pull-down resistance values may be different. Refer to the R_{PU} and R_{PD} value in the corresponding datasheet for the internal pull-up and pull-down parameters of the MCU used. Taking the HT32F52352 as an example, the internal pull-up and pull-down resistor parameters are "46kΩ" at 3.3V.

*2. When the pull-up and pull-down functions are both enabled, the pull-up function will have the higher priority and therefore the pull-down function will be blocked and disabled.

- Pull-Up

  When the HT32 MCU I/O is configured as a Pull-Up, as shown in the red dashed part of Figure 12, then when the pull-up is turned on, this indicates that the I/O default level is high (V_{DD} or V_{DDIO}).

  - The pull-up resistor can be configured through the PxPUR register, where x represents the different port name: A, B, C, etc.



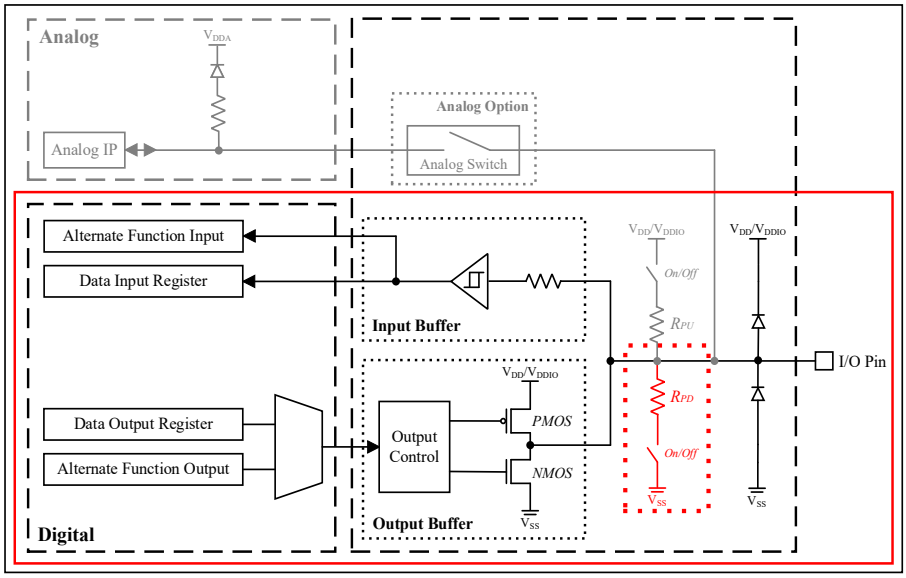
Note: V_{DDIO} is only supported by some MCUs, refer to the corresponding Datasheet for details.

Figure 12. HT32 MCU Simplified GPIO Architecture Diagram - Pull-up

- Pull-Down

  When the HT32 MCU I/O is configured as aPull-down, as shown within the red dashed line in Figure 13, this indicates that the I/O default level is low (GND) once the Pull-down is enabled.

  - The pull-down resistor can be configured through the PxPDR register, where x represents the different port name: A, B, C, etc.



Note: V_{DDIO} is only supported by some MCUs, refer to the corresponding Datasheet for details.

Figure 13. HT32 MCU Simplified GPIO Architecture Diagram - Pull-Down

GPIO Register Introduction

This chapter will introduce the HT32 MCU GPIO registers. These registers are introduced according to their six different GPIO modes described above.

Note: The specification of different MCUs may be different and the available register resources may also be different. The registers introduced in this chapter are based on the HT32 HT32F52352 specifications. In actual applications, refer to the specific device datasheet for accurate register information and configuration methods.

- Input/Output Related Registers

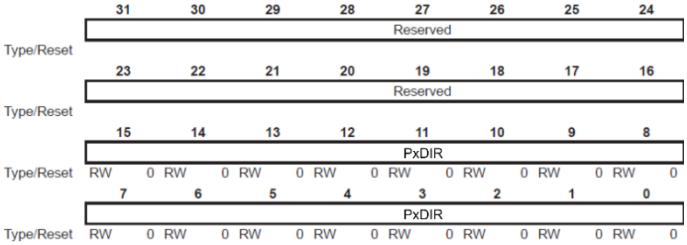
Register	PxDIRCR Register (Port x Data Direction Control Register, where x = A, B, C, etc.)
Descriptions	This register is used to control the direction of the GPIO Port x pin as input or output
Bitmap	
Bits	[15:0]
Field	PxDIRn
Descriptions	GPIO Port x Pin n Direction Control Bits (n=0~15) 0: Pin n is input mode 1: Pin n is output mode

Table 2. PxDIRCR Register

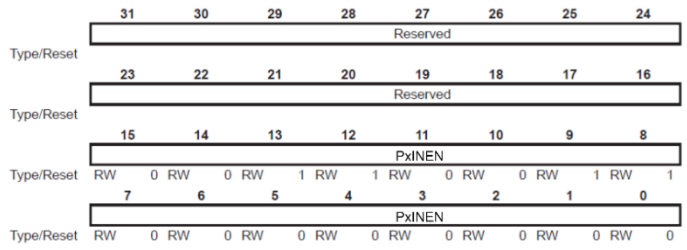
Register	PxINER Register (Port x Input Function Enable Control Register, where x = A, B, C, etc.)
Descriptions	This register is used to enable or disable the GPIO Port x input function
Bitmap	
Bits	[15:0]
Field	PxINENn
Descriptions	GPIO Port x Pin n Input Enable Control Bits (n=0~15) 0: Pin n input function is disabled 1: Pin n input function is enabled Note: When the pin n input function is disabled, the input Schmitt trigger will be turned off and the Schmitt trigger output will remain in a zero state.

Table 3. PxINER Register

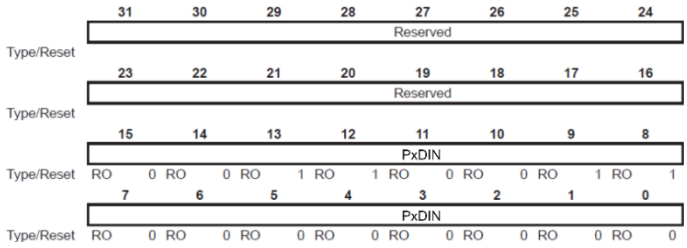
Register	PxDINR Register (Port x Data Input Register, where x = A, B, C, etc.)
Descriptions	This register specifies the GPIO Port x input data
Bitmap	
Bits	[15:0]
Field	PxDINn
Descriptions	GPIO Port x Pin n Data Input Bits (n=0~15) 0: The input data of pin n is 0 1: The input data of pin n is 1

Table 4. PxDINR Register

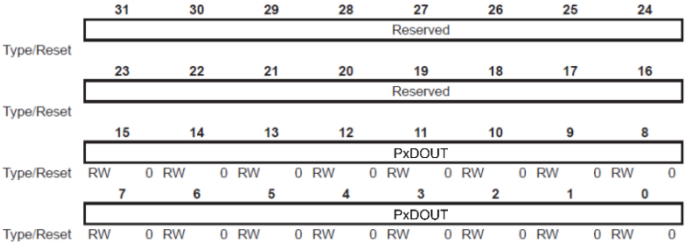
Register	PxDOUTR Register (Port x Output Data Register, where x = A, B, C, etc.)
Descriptions	This register specifies the GPIO Port x output data
Bitmap	
Bits	[15:0]
Field	PxDOUTn
Descriptions	GPIO Port x Pin n Data Output Bits (n=0~15) 0: Data to be output on pin n is 0 1: Data to be output on pin n is 1

Table 5. PxDOUTR Register

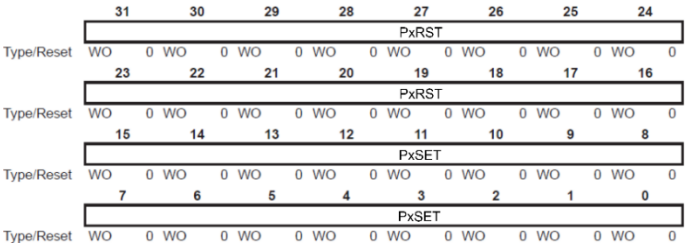
Register	PxSRR Register (Port x Output Set/Reset Control Register, where x = A, B, C, etc.)	
Descriptions	This register is used to set or reset the corresponding bit of the GPIO Port x output data	
Bitmap		
Bits	[31:16]	[15:0]
Field	PxRSTn	PxSETn
Descriptions	GPIO Port x Pin n Output Reset Control Bits (n=0~15) 0: No effect on the PxDOUTn bit 1: Reset the PxDOUTn bit	GPIO Port x Pin n Output Set Control Bits (n=0~15) 0: No effect on the PxDOUTn bit 1: Set the PxDOUTn bit
	Note: The function enabled by the PxSETn bit has the higher priority if both the PxSETn and PxRSTn bits are set at the same time.	

Table 6. PxSRR Register

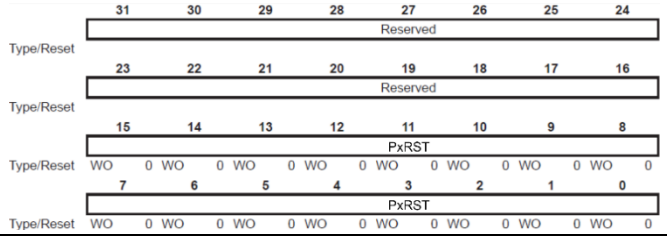
Register	PxRR Register (Port x Output Reset Register, where x = A, B, C, etc.)
Descriptions	This register is used to reset the corresponding bit of the GPIO Port x output data
Bitmap	
Bits	[15:0]
Field	PxRSTn
Descriptions	GPIO Port x Pin n Output Reset Bits (n=0~15) 0: No effect on the PxDOUTn bit 1: Reset the PxDOUTn bit Note: The function enabled by the PxSETn bit has the higher priority if both the PxSETn and PxRSTn bits are set at the same time.

Table 7. PxRR Register

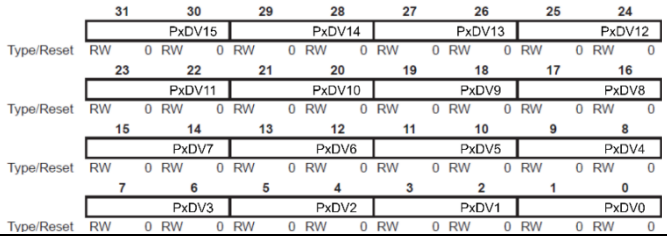
Register	PxDRVR Register (Port x Output Current Drive Selection Register, where x = A, B, C, etc.)
Descriptions	This register specifies the GPIO Port x output driving current
Bitmap	
Bits	[31:0]
Field	PxDVn[1:0]
Descriptions	GPIO Port x Pin n Output Current Drive Selection Control Bits (n=0~15) 00: 4 mA source/sink current 01: 8 mA source/sink current 10: 12 mA source/sink current 11: 16 mA source/sink current

Table 8. PxDRVR Register

- Push-Pull/Open-Drain Related Register

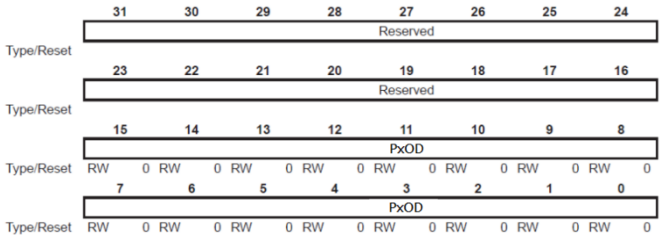
Register	PxODR Register (Port x Open-Drain Selection Register, where x = A, B, C, etc.)
Descriptions	This register is used to enable or disable the GPIO Port x Open-Drain function
Bitmap	
Bits	[15:0]
Field	PxODn
Descriptions	GPIO Port x Pin n Open-Drain Selection Control Bits (n=0~15) 0: Pin n Open-Drain output is disabled. The output type is a Push-Pull output. 1: Pin n Open-Drain output is enabled. The output type is an Open-Drain output.

Table 9. PxODR Register

- Pull-Up/Pull-Down Related Register

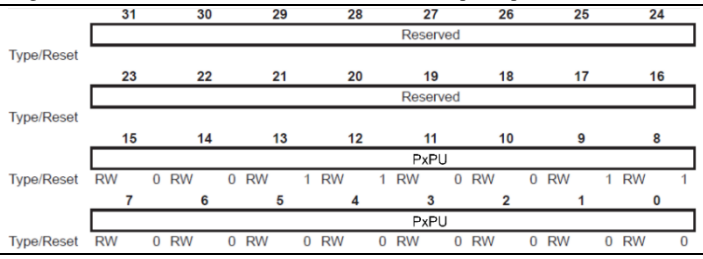
Register	PxPUR Register (Port x Pull-Up Selection Register, where x = A, B, C, etc.)
Descriptions	This register is used to enable or disable the GPIO Port x pull-up function
Bitmap	
Bits	[15:0]
Field	PxPUn
Descriptions	GPIO Port x Pin n Pull-Up Selection Control Bits (n=0~15) 0: Pin n pull-up function is disabled 1: Pin n pull-up function is enabled Note: When the pull-up and pull-down functions are both enabled, the pull-up function will have the higher priority and therefore the pull-down function will be disabled

Table 10. PxPUR Register

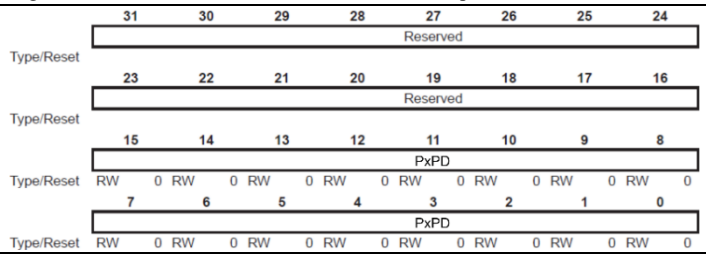
Register	PxPDR Register (Port x Pull-Down Selection Register, where x = A, B, C, etc.)
Descriptions	This register is used to enable or disable the GPIO Port x pull-down function
Bitmap	
Bits	[15:0]
Field	PxPDn
Descriptions	GPIO Port x Pin n Pull-Down Selection Control Bits (n=0~15) 0: Pin n pull-down function is disabled 1: Pin n pull-down function is enabled Note: When the pull-up and pull-down functions are both enabled, the pull-up function will have the higher priority and therefore the pull-down function will be disabled.

Table 11. PxPDR Register

GPIO API Description

This chapter will introduce the GPIO API classification settings for the six GPIO modes. Through this chapter, users can quickly understand the GPIO API function and usage. Table 12 provides a GPIO API summary. The detailed description of each API will be described later.

Classification GPIO Registers	API	Descriptions
Input/Output		
PxDIRCR	void GPIO_DirectionConfig()	Set the direction of the specified GPIO pin
PxINER	void GPIO_InputConfig()	Enable or disable the specified GPIO pin input control
PxDINR	FlagStatus GPIO_ReadInBit()	Obtain the specified GPIO pin input data
	u16 GPIO_ReadInData()	Obtain the specified GPIO Port input data
Input/Output		
PxDOUTR	FlagStatus GPIO_ReadOutBit()	Obtain the specified GPIO pin output data
	u16 GPIO_ReadOutData()	Obtain the specified GPIO Port output data
	void GPIO_WriteOutData()	Set the specified GPIO Port output data

Classification GPIO Registers	API	Descriptions
PxSRR\PxRR	void GPIO_SetOutBits()	Set the specified GPIO pin output data bit to 1
	void GPIO_ClearOutBits()	Clear the specified GPIO pin output data bit
	void GPIO_WriteOutBits()	Set the specified GPIO pin output data bit to 1 or clear it
PxDRVR	void GPIO_DriveConfig()	Select the specified GPIO pin drive current
Push-Pull/Open-Drain		
PxODR	void GPIO_OpenDrainConfig()	Enable or disable the GPIO Port Open-Drain function
Pull-Up/Pull-Down		
PxPUR\PxPDR	void GPIO_PullResistorConfig()	Configure the specified GPIO pin pull-up/pull-down resistor
DeInit API		
-	void GPIO_DeInit()	De-initialise the GPIO Port peripheral registers to the default reset values

Table 12. GPIO API Summary Table

- Input/Output Related API

  - PxDIRCR Register

API	void GPIO_DirectionConfig (HT_GPIO_TypeDef* HT_GPIOx, u16 GPIO_PIN_nBITMAP, GPIO_DIR_Enum GPIO_DIR_INorOUT)	
Function	Set the specified GPIO pin direction	
Input	HT_GPIOx	Where HT_GPIOx is the GPIO selected from the GPIO peripherals. Note: This parameter can have one of the following values: HT_GPIOA, HT_GPIOB...
	GPIO_PIN_nBITMAP	Port x Pin n Note: This parameter can be any combination of the following values: GPIO_PIN_x (x=0~15)
	GPIO_DIR_INorOUT	Port direction Note: This parameter can have one of the following values: GPIO_DIR_IN, GPIO_DIR_OUT

Table 13. GPIO API - GPIO_DirectionConfig()

  - PxINER Register

API	void GPIO_InputConfig (HT_GPIO_TypeDef* HT_GPIOx, u16 GPIO_PIN_nBITMAP, ControlStatus Cmd)	
Function	Enable or disable the specified GPIO pin input control	
Input	HT_GPIOx	Where HT_GPIOx is the GPIO selected from the GPIO peripherals. Note: This parameter can have one of the following values: HT_GPIOA, HT_GPIOB...
	GPIO_PIN_nBITMAP	Port x Pin n Note: This parameter can be any combination of the following values: GPIO_PIN_x (x=0~15)
	Cmd	Enable or disable Note: This parameter can have one of the following values: ENABLE, DISABLE

Table 14. GPIO API - GPIO_InputConfig()

  - PxDINR Register

API	FlagStatus GPIO_ReadInBit (HT_GPIO_TypeDef* HT_GPIOx, u16 GPIO_PIN_n)	
Function	Obtain the specified GPIO pin input data	
Input	HT_GPIOx	Where HT_GPIOx is the GPIO selected from the GPIO peripherals. Note: This parameter can have one of the following values: HT_GPIOA, HT_GPIOB...
	GPIO_PIN_n	Port x Pin n Note: This parameter can be any combination of the following values: GPIO_PIN_x (x=0~15)
Output	SET	The input data of the specified pin is 1
	RESET	The input data of the specified pin is 0

Table 15. GPIO API - GPIO_ReadInBit()

API	u16 GPIO_ReadInData (HT_GPIO_TypeDef* HT_GPIOx)	
Function	Obtain the specified GPIO pin output data	
Input	HT_GPIOx	Where HT_GPIOx is the GPIO selected from the GPIO peripherals. Note: This parameter can have one of the following values: HT_GPIOA, HT_GPIOB...
Output	uValue	Input data register value

Table 16. GPIO API - GPIO_ReadInData()

➢ PxDOUTR Register

API	FlagStatus GPIO_ReadOutBit (HT_GPIO_TypeDef* HT_GPIOx, u16 GPIO_PIN_n)	
Function	Obtain the specified GPIO pin output data	
Input	HT_GPIOx	Where HT_GPIOx is the GPIO selected from the GPIO peripherals. Note: This parameter can have one of the following values: HT_GPIOA, HT_GPIOB...
	GPIO_PIN_n	Port x Pin n Note: This parameter can be any combination of the following values: GPIO_PIN_x (x=0~15)
Output	SET	The input data of the specified pin is 1
	RESET	The input data of the specified pin is 0

Table 1. GPIO API - GPIO_ReadOutBit()

API	u16 GPIO_ReadOutData (HT_GPIO_TypeDef* HT_GPIOx, u16 Data)	
Function	Obtain the specified GPIO Port output data	
Input	HT_GPIOx	Where HT_GPIOx is the GPIO selected from the GPIO peripherals. Note: This parameter can have one of the following values: HT_GPIOA, HT_GPIOB...
Output	uValue	Input data register value

Table 2. GPIO API - GPIO_ReadOutData ()

API	void GPIO_WriteOutData (HT_GPIO_TypeDef* HT_GPIOx, u16 Data)	
Function	Set the specified GPIO Port output data	
Input	HT_GPIOx	Where HT_GPIOx is the GPIO selected from the GPIO peripherals. Note: This parameter can have one of the following values: HT_GPIOA, HT_GPIOB…
	Data	The data to be written into the GPIO Port data register

Table 3. GPIO API - GPIO_WriteOutData()

➢ PxSRR\PxRR Register

API	void GPIO_SetOutBits (HT_GPIO_TypeDef* HT_GPIOx, u16 GPIO_PIN_nBITMAP)	
Function	Set the specified GPIO pin output data bit to 1	
Input	HT_GPIOx	Where HT_GPIOx is the GPIO selected from the GPIO peripherals. Note: This parameter can have one of the following values: HT_GPIOA, HT_GPIOB...
	GPIO_PIN_nBITMAP	Port x Pin n Note: This parameter can be any combination of the following values: GPIO_PIN_x (x=0~15)

Table 20. GPIO API - GPIO_SetOutBits()

API	void GPIO_ClearOutBits (HT_GPIO_TypeDef* HT_GPIOx, u16 GPIO_PIN_nBITMAP)	
Function	Clear the specified GPIO pin output data bit	
Input	HT_GPIOx	Where HT_GPIOx is the GPIO selected from the GPIO peripherals. Note: This parameter can have one of the following values: HT_GPIOA, HT_GPIOB...
	GPIO_PIN_nBITMAP	Port x Pin n Note: This parameter can be any combination of the following values: GPIO_PIN_x (x=0~15)

Table 4. GPIO API - GPIO_ClearOutBits()

API	void GPIO_WriteOutBits (HT_GPIO_TypeDef* HT_GPIOx, u16 GPIO_PIN_nBITMAP, FlagStatus Status)	
Function	Set the specified GPIO pin output data bit 1 or clear it	
Input	HT_GPIOx	Where HT_GPIOx is the GPIO selected from the GPIO peripherals. Note: This parameter can have one of the following values: HT_GPIOA, HT_GPIOB…
	GPIO_PIN_nBITMAP	Port x Pin n Note: This parameter can be any combination of the following values: GPIO_PIN_x (x=0~15)
	Status	Set or reset Note: This parameter can have one of the following values: SET, RESET

Table 5. GPIO API - GPIO_WriteOutBits ()

➢ PxDRV Register

API	void GPIO_DriveConfig (HT GPIO TypeDef* HT GPIOx, u16 GPIO PIN nBITMAP, GPIO DV Enum GPIO_DV_nMA)	
Function	Select the specified GPIO pin drive current	
Input	HT_GPIOx	Where HT_GPIOx is the GPIO selected from the GPIO peripherals. Note: This parameter can have one of the following values: HT_GPIOA, HT_GPIOB…
	GPIO_PIN_nBITMAP	Port x Pin n Note: This parameter can be any combination of the following values: GPIO_PIN_x (x=0~15)
	GPIO_DV_nMA	Select the output drive current, depending on the device used, different devices or different types of pins may offer different output drive current selections Note: This parameter can have one of the following values: GPIO_DV_4MA: Select an output drive current of 4mA GPIO_DV_8MA: Select an output drive current of 8mA GPIO_DV_12MA: Select an output drive current of 12mA GPIO_DV_16MA: Select an output drive current of 16mA

Table 23. GPIO API—GPIO_DriveConfig ()

- Push-Pull/Open-Drain Related API

➢ PxODR Rregister

API	void GPIO_OpenDrainConfig (HT_GPIO_TypeDef* HT_GPIOx, u16 GPIO_PIN_nBITMAP, ControlStatus Cmd)	
Function	Enable or disable the specified GPIO pin Open-Drain function.	
Input	HT_GPIOx	Where HT_GPIOx is the GPIO selected from the GPIO peripherals. Note: This parameter can have one of the following values: HT_GPIOA, HT_GPIOB…
	GPIO_PIN_nBITMAP	Port x Pin n Note: This parameter can be any combination of the following values: GPIO_PIN_x (x=0~15)
	Cmd	Enable or disable Note: This parameter can have one of the following values: ENABLE, DISABLE

Table 24. GPIO API—GPIO_OpenDrainConfig()

- Pull-Up/Pull-Down Related API
  - PxPUR\PxPDR register

API	void GPIO_PullResistorConfig (HT_GPIO_TypeDef* HT_GPIOx, u16 GPIO_PIN_nBITMAP, GPIO_PR_Enum GPIO_PR_x)	
Function	Configure the specified GPIO pin pull-up/down resistor	
Input	HT_GPIOx	Where HT_GPIOx is the GPIO selected from the GPIO peripherals. Note: This parameter can have one of the following values: HT_GPIOA, HT_GPIOB…
	GPIO_PIN_nBITMAP	Port x Pin n Note: This parameter can be any combination of the following values: GPIO_PIN_x (x=0~15)
	GPIO_PR_x	Select the pull-up resistor Note: This parameter can have one of the following values: GPIO_PR_UP: Pin with internal pull-up resistor GPIO_PR_DOWN: Pin with internal pull-down resistor GPIO_PR_DISABLE: Pin without pull resistor

Table 25. GPIO API—GPIO_PullResistorConfig()

- DeInit API

API	void GPIO_DeInit (HT_GPIO_TypeDef* HT_GPIOx)	
Function	De-initialise the GPIO Port peripheral register to the default reset value	
Input	HT_GPIOx	Where HT_GPIOx is the GPIO selected from the GPIO peripherals. Note: This parameter can have one of the following values: HT_GPIOA, HT_GPIOB…

Table 26. GPIO API—GPIO_DeInit ()

GPIO API Usage Examples

This chapter provides some GPIO API usage examples, categorized by the functions they set and listed in the corresponding API usage syntax. This will assist users to quickly understand the GPIO API usage methods.

- Enable Clock: Enable the AFIO and Port A clocks in the CKCU, as shown in the table below.

  Note: Before setting the GPIOs, it is necessary to enable the CKCU Clock for the GPIO Port. If Register Access is executed without enabling the clock, it will cause a hard fault on the HT32 MCU.

Program Example	`{` `  CKCU_PeripClockConfig_TypeDef CKCUClock = {{0}};` `  CKCUClock.Bit.AFIO = 1;` `  CKCUClock.Bit.PA = 1;` `  CKCU_PeripClockConfig(CKCUClock, ENABLE);` `}`

Table 27. GPIO API Usage Example—Enable Clock

- Input+Pull-Up/Pull-Down+Read: Configure the GPIO pin to be an input, pull up/down settings and read the I/O status, as shown in the table below.

Example Setting	1. AFIO mode: GPIO Mode 2. Pin direction: Input 3. Pull resistor: PA0 set to pull-down; PA1 set to pull-up 4. Input function: Enable 5. Read PA0 & PA1 I/O

Program Example	{ FlagStatus PA0_Status = RESET; FlagStatus PA1_Status = RESET; /* Configure PA0, PA1 AFIO mode of GPIO */ AFIO_GPxConfig(GPIO_PA, AFIO_PIN_0, AFIO_FUN_GPIO); AFIO_GPxConfig(GPIO_PA, AFIO_PIN_1, AFIO_FUN_GPIO); /* Configure GPIO direction of input pins */ GPIO_DirectionConfig(HT_GPIOA, GPIO_PIN_0, GPIO_DIR_IN); GPIO_DirectionConfig(HT_GPIOA, GPIO_PIN_1, GPIO_DIR_IN); /* Configre PA0 pin as pull-down */ GPIO_PullResistorConfig(HT_GPIOA, GPIO_PIN_0, GPIO_PR_DOWN); /* Configre PA1 pin as pull-up */ GPIO_PullResistorConfig(HT_GPIOA, GPIO_PIN_1, GPIO_PR_UP); /* Configure Input function enable */ GPIO_InputConfig(HT_GPIOA, GPIO_PIN_0, ENABLE); GPIO_InputConfig(HT_GPIOA, GPIO_PIN_1, ENABLE); /* Read PA0 and store it in PA0_Status variable, PA1 and store it in PA1_Status variable */ PA0_Status = GPIO_ReadInBit(HT_GPIOA, GPIO_PIN_0); PA1_Status = GPIO_ReadInBit(HT_GPIOA, GPIO_PIN_1); }

Table 28. GPIO API Usage Example—Input + Pull-Up / Pull-Down + Read

- Output High/Low (Push-Pull/Open-Drain): Configure the GPIO pin to be an output, Push-Pull/Open-Drain setting, output High/Low, as shown in the table below.

Example Setting	1. AFIO mode: GPIO Mode 2. Open-Drain function: PA2 is configured to Open-Drain (ENABLE); PA3 is configured to Push-Pull (DISABLE). Note: Since PA2 is configured to Open-Drain, an external pull-up resistor is required. 3. Pin direction: Output 4. Set PA2 output High, PA3 output Low
Program Example	{ /* Configure PA2, PA3 AFIO mode of GPIO */ AFIO_GPxConfig(GPIO_PA, AFIO_PIN_2, AFIO_FUN_GPIO); AFIO_GPxConfig(GPIO_PA, AFIO_PIN_3, AFIO_FUN_GPIO); /* Configure PA2 pin as open-drain */ GPIO_OpenDrainConfig(HT_GPIOA, GPIO_PIN_2, ENABLE); /* PA3 pin as push-pull */ GPIO_OpenDrainConfig(HT_GPIOA, GPIO_PIN_3, DISABLE); /* Configure GPIO direction of output pins */ GPIO_DirectionConfig(HT_GPIOA, GPIO_PIN_2, GPIO_DIR_OUT); GPIO_DirectionConfig(HT_GPIOA, GPIO_PIN_3, GPIO_DIR_OUT); /* Configure data to be output on PA2 is 1 */ **/* Notice that because PA2 is Open-Drain, an external pull-up resistor is required */** GPIO_WriteOutBits(HT_GPIOA, GPIO_PIN_2, SET); /* Configure data to be output on PA3 is 0 */ GPIO_WriteOutBits(HT_GPIOA, GPIO_PIN_3, RESET); }

Table 29. GPIO API Usage Example—Output High/Low - Push-Pull/Open-Drain

GPIO Configuration Priorities and Limitation

This chapter describes the priority and related limitations of the GPIO settings, to help users to quickly understand the priority and limitation relationships between settings. The table shows the relationship between various modes and their respective limitations.

- Pull-Up & Pull-Down Priority Rules
  - Priority: Pull-Up > Pull-Down
  - If the Pull-up and Pull-down functions are both enabled on the same pin at the same time, the pull-up function has the higher priority while the pull-down function will have no effect.
- Pull-Up & Pull-Down Limitation
  - When an I/O pin is configured as a GPIO input or AFIO input mode, the Pull-up and Pull-down resistors on that pin can be controlled by modifying the PxPUR and PxPDR registers.
  - When an I/O pin is configured as a GPIO Output or AFIO Output mode, the internalpull-up and pull-down resistors function on that pin will be disabled.
- Open-Drain Limitation and Special Conditions
  - The Open-Drain function is applicable to the GPIO Output or AFIO Output modes. When an I/O pin is configured to be in either of these modes, the Open-Drain feature can be enabled through the PxODR register.
  - When the I/O pin is configured to the AFIO Mode 7 (I^2C mode), the Open-Drain will be forced by the AFIO and cannot be controlled by the PxODR register.
- Driver Current Description
  - The Driver Current is used to set the I/O pin Sink/Source Current capability, which can be used in a GPIO Output and AFIO Output (Digital) modes.
  - Most I/O pins can be configured to provide sink current/source currents of 4mA/8mA/ 12mA/16mA, while the I/O pins in the Backup Domain only have a lower current driving capability. Refer to the Pin Description table in the device datasheet for details about the I/O output drive current capability.

	GPIO Input	GPIO Output	AFIO Input	AFIO Output
Pull-Up / Pull-Down	V	X	V	X
Open-Drain	N/A	V	N/A	V (Digital)
Driver Current	N/A	V	N/A	V (Digital)

Table 30. Mode and Limitation Relationship Table

GPIO Pin Lock

The main purpose of the GPIO Pin Lock function is used to enhance the security and stability of the system. The function allows certain I/O pin configurations to be locked to prevent system errors or external attacks from changing these critical configurations.

The GPIO offers a lock function to lock the port until a reset event occurs. In applications, the GPIO Pin Lock prevents I/O Settings from being inadvertently changed by software to avoid unintended GPIO operations. The PxLOCKR registers are used to lock the GPIO Port and lock control options. Refer to the table below for a PxLOCKR register brief description.

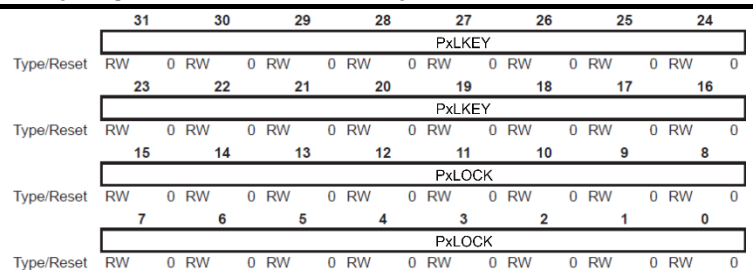
Register	PxLOCKR Register (Port x Lock Register, x = A, B, C, etc.)	
Descriptions	This register specifies the GPIO Port x lock configuration	
Bitmap		
Bits	[31:16]	[15:0]
Field	PxLKEY	PxLOCKn
Descriptions	GPIO Port x Lock Key 0x5FA0: Port x Lock function is enable Others: Port x Lock function is disable	GPIO Port x Pin n Lock Control Bits (n=0~15) 0: Port x Pin n is not locked 1: Port x Pin n is locked
	Note: The PxLOCKR can only be written once which means that PxLKEY and PxLOCKn (lock control bit) should be written together and cannot be changed until a system reset or a GPIO Port x reset occurs.	

Table 31. PxLOCKR Register

To lock the Port x function, a value 0x5FA0 should be written into the PxLKEY field in this register. To execute a successful write operation on this lock register, the value written into the PxLKEY field must be 0x5FA0. If the value written into this field is not equal to 0x5FA0, any write operations to the PxLOCKR register will be ineffective. The PxLOCKn bits are used to lock the configurations of the corresponding GPIO pins when the correct Lock Key "0x5FA0"is applied to the PALKEY field (Note).

The result of a read operation on the PxLKEY field returns the GPIO Port x Lock Status which indicates whether the GPIO Port x is locked or not. If the read value of the PxLKEY field is 0, this indicates that the GPIO Port x Lock function is disabled. Otherwise, it indicates that the GPIO Port x Lock function is enabled as the read value is equal to 1.

Note: The locked configuration includes the GPIO setting related registers such as "PxDIRn, PxINENn, PxPUn, PxPDn, PxODn, and PxDVn" and the GPxCFGHR or GPxCFGLR field which is used to configure the alternative function of the associated GPIO pin.

The GPIO Pin Lock related APIs are shown in Tables 32~34. Refer to the following program code and its results for an actual GPIO Pin Lock example.

Function	void GPIO_PinLock (HT_GPIO_TypeDef* HT_GPIOx, u16 GPIO_PIN_nBITMAP)	
Function	Configure the specified GPIO pin lock function	
Input	HT_GPIOx	Where HT_GPIOx is the GPIO selected from the GPIO peripherals. Note: This parameter can have one of the following values: HT_GPIOA, HT_GPIOB…
	GPIO_PIN_nBITMAP	Port x Pin n Note: This parameter can be any combination of the following values: GPIO_PIN_x (x=0~15)

Table 32. API - GPIO_PinLock()

Function	bool GPIO_IsPortLocked (HT_GPIO_TypeDef* HT_GPIOx)	
Function	Obtain the specified GPIO Port lock status	
Input	HT_GPIOx	Where HT_GPIOx is the GPIO selected from the GPIO peripherals. Note: This parameter can have one of the following values: HT_GPIOA, HT_GPIOB…
Output	TRUE	GPIO Port lock function is enabled
	FALSE	GPIO Port lock function is disabled

Table 33. API - GPIO_IsPortLocked()

Function	bool GPIO_IsPinLocked (HT_GPIO_TypeDef* HT_GPIOx, u16 GPIO_PIN_n)	
Function	Obtain the specified GPIO pin lock status	
Input	HT_GPIOx	Where HT_GPIOx is the GPIO selected from the GPIO peripherals. Note: This parameter can have one of the following values: HT_GPIOA, HT_GPIOB…
	GPIO_PIN_n	Port x Pin n Note: This parameter can be any combination of the following values: GPIO_PIN_x (x=0~15)
Output	TRUE	GPIO pin lock function is enabled
	FALSE	GPIO pin lock function is disabled

Table 34. API - GPIO_IsPinLocked()

The following is an example to describe how to set the HT32 MCU GPIO Pin Lock function and its result.

Refer to the HT32 MCU Firmware Library GPIO Pin Lock example.

✧ relevant Include, Function, Variable as follows

```
/* Includes ------------------------------------------------------------------*/
#include "ht32.h"
/* Private function prototypes -----------------------------------------------*/
void GPIO_Configuration(void);
void GPIO_LockStatus(void);
void GPIO_LockPinTest(void);
void GPIO_OtherPinTest(void);
```

Table 35. Pin Lock Example - Related Include and Function

✧ The main function includes RETARGET setting function → GPIO setting function → Locks PA1 → Lock state reading function → Lock pin testing function (PA1) → Unlock pin testing function (PA2)). Refer to the following for functions detailed description.

```
/* Global functions ----------------------------------------------------------------------------*/
int main(void)
{
  RETARGET_Configuration();
  GPIO_Configuration();
  GPIO_PinLock(HT_GPIOA, GPIO_PIN_1);
  GPIO_LockStatus();
  GPIO_LockPinTest();
  GPIO_OtherPinTest();
  while (1);
}
```

Table 36. Pin Lock Example - main ()

✧ Refer to Table 37 for GPIO_Configuration() instruction.

Function	void GPIO_Configuration (void)
Function Description	Configure GPIO (PA1 and PA2)
Implement Setting	AFIO mode: GPIO Mode Default value: RESET Pull resistance: Pull-Disable Output drive current: 8mA Pin direction: Output
Program Example	void GPIO_Configuration(void) { { /* Enable peripheral clock */ CKCU_PeripClockConfig_TypeDef CKCUClock = {{ 0 }}; CKCUClock.Bit.AFIO = 1; CKCUClock.Bit.PA = 1; CKCU_PeripClockConfig(CKCUClock, ENABLE); } { /* Configure GPIO as output mode */ /* Configure AFIO mode as GPIO */ AFIO_GPxConfig(GPIO_PA, AFIO_PIN_1, AFIO_FUN_GPIO); AFIO_GPxConfig(GPIO_PA, AFIO_PIN_2, AFIO_FUN_GPIO); /* Default value RESET/SET */ GPIO_WriteOutBits(HT_GPIOA, GPIO_PIN_1, RESET); GPIO_WriteOutBits(HT_GPIOA, GPIO_PIN_2, RESET); /* Configure GPIO pull resistor */ GPIO_PullResistorConfig(HT_GPIOA, GPIO_PIN_1, GPIO_PR_DISABLE); GPIO_PullResistorConfig(HT_GPIOA, GPIO_PIN_2, GPIO_PR_DISABLE); /* Configure GPIO Output Driving Current */ GPIO_DriveConfig(HT_GPIOA, GPIO_PIN_1, GPIO_DV_8MA); GPIO_DriveConfig(HT_GPIOA, GPIO_PIN_2, GPIO_DV_8MA); /* Configure GPIO direction as output */ GPIO_DirectionConfig(HT_GPIOA, GPIO_PIN_1, GPIO_DIR_OUT); GPIO_DirectionConfig(HT_GPIOA, GPIO_PIN_2, GPIO_DIR_OUT); } }

Table 37. Pin Lock Example—GPIO_Configuration()

✧ The GPIO_PinLock(HT_GPIOA, GPIO_PIN_1) function can be used to lock GPIO PA1. Refer to table 32 for the API description.

✧ Refer to Table 38 for the GPIO_LockStatus() description. Since the RETARGET function is used, the result information can be confirmed by the terminal software using the HT32 Starter Kit RETARGET to specify the UART pin.

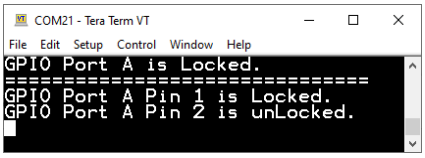
Function	void GPIO_LockStatus (void)
Function Description	Determine the specified GPIO Port and Pin lock state (PA1 and PA2)
Example Content	void GPIO_LockStatus(void) { /* Port A Lock Status */ if (GPIO_IsPortLocked(HT_GPIOA) == TRUE) { printf("GPIO Port A is Locked. \r\n"); } else { printf("GPIO Port A is unLocked. \r\n"); } printf("================================\r\n"); /* Port A Pin 1 & 2 Lock Status */ if (GPIO_IsPinLocked(HT_GPIOA, GPIO_PIN_1) == TRUE) { printf("GPIO Port A Pin 1 is Locked. \r\n"); } else { printf("GPIO Port A Pin 1 is unLocked. \r\n"); } if (GPIO_IsPinLocked(HT_GPIOA, GPIO_PIN_2) == TRUE) { printf("GPIO Port A Pin 2 is Locked. \r\n"); } else { printf("GPIO Port A Pin 2 is unLocked. \r\n"); } }
Results	GPIO Port A locked state: Locked. GPIO Port A pin 1 locked state: Locked. (Locking method: GPIO_PinLock(HT_GPIOA, GPIO_PIN_1)) GPIO Port A pin 2 locked state: unLocked. 

Table 38. Pin Lock Example—GPIO_LockStatus()

✧ Refer to Table 39 for GPIO_LockPinTest() description.

Function	void GPIO_LockPinTest (void)
Function Description	Lock pin test (PA1). It is invalid to change the configuration of the PA1 GPIO since PA1 is locked.
Function Description	After changing the GPIO configuration, PA1 is locked, therefore the GPIO configuration change is invalid and the configuration status remains unchanged: ➢ AFIO mode: GPIO Mode ➢ Default: RESET ➢ Pull resistor: Pull-Disable ➢ Output drive current: 8mA ➢ Direction: Output

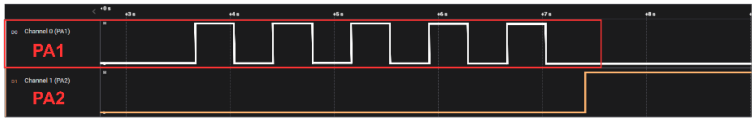
Function	void GPIO_LockPinTest (void)
Example Contents	void GPIO_LockPinTest(void) { vu32 i, j; **/* Change PA1 GPIO Configuration is invalid, because PA1 is locked */** /* PA1: (Output → Output) Mode, (Pull-Disable → Pull-Disable) */ GPIO_PullResistorConfig(HT_GPIOA, GPIO_PIN_1, GPIO_PR_UP); GPIO_DirectionConfig(HT_GPIOA, GPIO_PIN_1, GPIO_DIR_IN); /* Toggle PA1 ten times */ for (j = 0; j < 10; j++) { HT_GPIOA->DOUTR ^= 1 << 1; /* PA1 Pin Lock in Output Mode, so toggle valid. */ for (i = 0; i < 1000000; i++); } }
Results	PA1 has toggled ten times. Since PA1 is locked, changing the configuration is invalid. (PA1 is still in output mode, so it can be toggled) PA1: (Output→Output) Mode, (Pull-Disable→Pull-Disable) 

Table 39: Pin Lock Example—GPIO_LockPinTest()

✧ Refer to Table 40 for the GPIO_OtherPinTest() description.

Function	void GPIO_OtherPinTest(void)
Function Description	Unlocked pin test (PA2). It is valid to change the PA2 GPIO configuration since PA2 is unlocked.
Function Description	After changing the GPIO configuration, changing the PA2 GPIO configuration is valid, because PA2 is unlocked (pull resistance and direction change): ➢ AFIO mode: GPIO Mode ➢ Default value: RESET ➢ Pull resistor: Pull-up ➢ Output drive current: 8mA ➢ Direction: Input
Example Contents	void GPIO_OtherPinTest(void) { vu32 i, j; **/* Change PA2 GPIO Configuration is valid, because PA2 is unlocked */** /* PA2: (Output → Input) Mode, (Pull-Disable → Pull-Up) */ GPIO_PullResistorConfig(HT_GPIOA, GPIO_PIN_2, GPIO_PR_UP); GPIO_DirectionConfig(HT_GPIOA, GPIO_PIN_2, GPIO_DIR_IN); /* PA2 keeps the high level */ for (j = 0; j < 10; j++) { HT_GPIOA->DOUTR ^= 1 << 2; /* PA2 Input Mode, so toggle invalid. */ for (i = 0; i < 1000000; i++); } }
Results	PA2 changed to keep a high level, since PA2 is unlocked, the changed configuration is valid. (PA2 is changed from an output mode to an input mode, and the internal pull-up is set, so it cannot be toggled and the I/O status is High). PA2：(Output→Input) Mode, (Pull-Disable→Pull-Up) 

Table 40. Pin Lock Example—GPIO_UnLockPinTest()

Special GPIO Configuration and Characteristics

This chapter describes the special GPIO characteristics to help users can quickly understand which type of I/O or which I/O belongs to the special GPIO category.

- Boot & Debug Pin Default Configuration

  After an MCU reset, most of the GPIO Ports are configured into input disable floating mode, i.e. input disabled without pull-up/pull-down resistors. For some specific GPIO pins, such as the Boot and Debug pins, the pull-up function will be enabled by default. Taking the HT32F52352 as an example, the Boot and Debug pins include PA8, PA9_BOOT, SWCLK, SWDIO pins. The default configurations for these specific GPIO pins are as follows.

  - PA8: Input enable with internal pull-up
  - PA9_BOOT: Input enable with internal pull-up
  - SWCLK: Input enable with internal pull-up
  - SWDIO: Input enable with internal pull-up

- nRST Pin Internally Integrated Pull-up

  The HT32 MCU's nRST pin typically integrates an internal weak pull-up resistor (33V_PU) by default. Refer to the device datasheet for the internal weak pull-up resistance actual value. Taking the HT32F52352 as an example, the nRST pin internal weak pull-up resistance is about 46kΩ.

Pin Number			Pin Name	Type (Note1)	I/O Structure (Note2)	Output Driving	Description
64LQFP	48LQFP	33QFN					Default Function (AF0)
20	16	12	nRST	I(BK)	33V_PU	—	External reset pin and external wakeup pin in the Power-Down mode

Figure 14. HT32F52352 nRST Pin Description -Refer to the Datasheet

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
R_{PU}	Internal pull-up resistor	3.3 V I/O	—	46	—	kΩ
R_{PD}	Internal pull-down resistor	3.3 V I/O	—	46	—	kΩ

Figure 15. HT32F52352 Internal Pull-Up/Down Resistor -Refer to the Datasheet

- Backup Domain I/O Driving Current Capability Limitation (e.g. RTCOUT and X32KIN pin, etc.)

  Taking the HT32F52352 as an example, pins such as PB12 (RTCOUT), PB10 (X32KIN) and PB11 (X32KOUT) belong to the Backup Domain I/O. Their current drive capability is weaker than for other I/Os.

  Taking the HT32F52352 specification as an example, the Backup Domain I/O pins have the following driving capability limitations:

  - Source Current Limitation: < 2mA @ V_{DD}=3.3V
  - Typical Sink Current: 4/8mA @ V_{DD}=3.3V

- USB Related I/O Characteristics

  For the HT32 MCUs which have a USB interface, such as the HT32F52352 and HT32F12366 devices, when the USB is suspended or when the USB has not been initialised after the MCU has powered up, the USB PHY has a default weak pull-up on the D+ line and a weak pull-down on the D- line. Here, the $R_{Pull\text{-}Up}$ and $R_{Pull\text{-}Down}$ values are approximately 120kΩ.

Note: Due to the USB PHY condition, some problems may be caused due to external/internal circuits, such as abnormal power consumption. To solve the internal weak pull-up/down function, there is an API - USBD_DisableDefaultPull(), which can be used to disable the default pull-up resistor function on the USB D+ and D- pins. Calling this API after enabling the USB interface to turn off the weak pull resistor function may be helpful for some problems, but it will only eliminate the DC Path for power consumption problem and transform it into only USB PHY / SIE power consumption, therefore it is not of significant benefit with regards to power consumption.

Function	void USBD_DisableDefaultPull(void)
Function	Disables the default pull-resistor function on D+ and D-.

Table 41. API - USBD_DisableDefaultPull()

AFIO

This chapter will introduce the HT32 MCU AFIO, including the AFIO registers, APIs, usage examples and the I/O default AFIO mode. This will allow users to have a better understanding of the HT32 MCU AFIO functions.

AFIO Function Description

For the HT32 MCU GPIOs, users can configure each pin to have an alternative function input/output for various peripherals such as a communication interface (SPI, USART/UART, I^2C, etc.), timer (GPTM, SCTM, MCTM, etc.), ADC, DAC, CMP, etc.

The alternative functions of the specified GPIO pin can be configured using the following registers.

- GPxCFGLR - Configure the alternative function on port x pin n (x = A, B, C, etc, n = 0 ~ 7)
- GPxCFGHR - Configure the alternative function on port x pin n (x = A, B, C, etc, n = 0 ~ 7)

Refer to the "AFIO Register Introduction" section in this document for the register description. Refer to the corresponding section in the device datasheet for specific alternative functions supported by each MCU GPIO pin.

AFIO Register Description

This chapter will introducethe HT32 MCU AFIO registers. Since different MCUs may have different specifications and the available register resources might also differ, the following explanation of the AFIO registers is based on the HT32F52352 specification as an example.

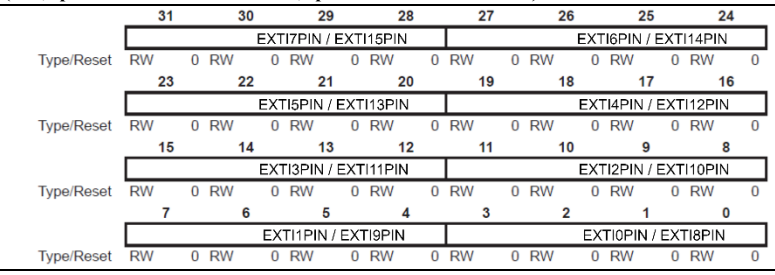
Register	ESSRx Register (EXTI Source Selection Register x, x = 0 or 1)
Descriptions	This register specifies the I/O selection of EXTIn **(x=0, specifies EXTI0 ~ EXTI7x=1, specifies EXTI8 ~ EXTI15)**
Bitmap	
Bits	[31:0]
Field	EXTInPIN[3:0]
Descriptions	EXTIn Pin Selection (n=0~15) 0000: PA Bit n is selected as EXTIn source signal 0001: PB Bit n is selected as EXTIn source signal 0010: PC Bit n is selected as EXTIn source signal 0011: PD Bit n is selected as EXTIn source signal Others: Reserved Note: Since not all GPIO pins are available in all products and package types, refer to the datasheet pin assignment section for detailed pin information. The EXTInPIN[3:0] field setting is invalid when the corresponding pin is not available.

Table 42. ESSRx Register

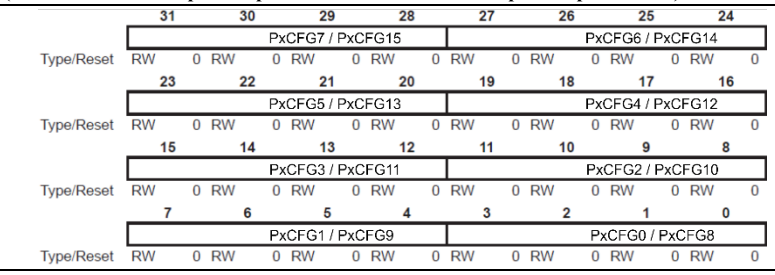
Register	GPxCFGLR/GPxCFGHR Register (GPIO x Configuration Register, x = A, B, C, etc.)
Descriptions	This register specifies the alternate function of GPIO Port x Pin n (n=0~15) **(where GPxCFGLR specifies pins 0 to 7 and GPxCFGHR specifies pins 8 to 15)**
Bitmap	
Bits	[31:0]
Field	PxCFGn[3:0]
Descriptions	Port x Pin n Alternate function selection (n=0~15) 0000: Port x pin n is selected as AF0. 0001: Port x pin n is selected as AF1. … 1110: Port x pin n is selected as AF14 1111: Port x pin n is selected as AF15 Note: If the pin is selected as an unavailable item which is noted as "N/A" item in the "Alternate Function Mapping" table in the device datasheet, this pin will be defined as the default alternate function(AF0).

Table 43. GPxCFGR register (GPxCFGLR, GPxCFGHR)

AFIO API Description

This chapter will introduce the AFIO API. In this chapter, users can quickly understand the AFIO API function and usage. Table 44 shows the AFIO API summary table. The detailed description of each API will be described later.

Classification AFIO Registers	API	Descriptions
GPxCFGR	**void AFIO_GPxConfig()**	Set the specified GPIO pin alternate mode
ESSRx	**void AFIO_EXTISourceConfig()**	Select the GPIO pin to be used as the EXTI channel
DeInit API		
-	**void AFIO_DeInit()**	De-initialise the AFIO port peripheral registers to the default reset values

Table 44. AFIO API Summary Table

- GPxCFGR Register (GPxCFGLR, GPxCFGHR)

Function	void AFIO_GPxConfig (u32 GPIO_Px, u32 AFIO_PIN_n, AFIO_MODE_Enum AFIO_MODE_n)	
Function	Set the specified GPIO pin alternate mode	
Input	GPIO_Px	GPIO Port setting Note: This parameter can have one of the following values: GPIO_Px (x=A, B, C, etc.)
	AFIO_PIN_n	Port x Pin n Note: This parameter can be any combination of the following values: AFIO_PIN_x(x=0~15)
	AFIO_MODE_n	Alternate mode Note: This parameter can have one of the following values: ========== Configure the value of AFIO_MODE_Enum typedef as follows: AFIO_MODE_DEFAULT: Default I/O function AFIO_MODE_1: Alternate function 1 AFIO_MODE_2: Alternate function 2 … AFIO_MODE_14: Alternate function 14 AFIO_MODE_15: Alternate function 15 --- The values using the FUN macro mode (Note) are as follows: AFIO_FUN_DEFAULT AFIO_FUN_GPIO AFIO_FUN_ADC0 … FUN macro refer to ht32fXxxxx_gpio.h in the Firmware Library ==========

Table 45. AFIO API - AFIO_GPxConfig()

Note: It is recommended that designers give priority to using the FUN Macro mode when configuring the AFIO, because the AFIO Mode planning for each MCU may be different. The FUN Macro mode can be set for the alternative function.

As shown below (e.g. setting PA1, where AFIO Mode 1 is GPIO Mode), for example, to configure the HT32F52352 PA1 pin to the GPIO Mode (which corresponds to AFIO Mode 1), designers can use the following code:

Priority recommendation setting: AFIO_GPxConfig(GPIO_PA, AFIO_PIN_1, AFIO_FUN_GPIO);

→ Equal to setting: AFIO_GPxConfig(GPIO_PA, AFIO_PIN_1, AFIO_MODE_1);

- ESSRx Register (x=0, 1)

Function	void AFIO_EXTISourceConfig (u32 GPIO_PIN_NUM_n, u32 GPIO_Px)	
Function	Select the GPIO pin to be used as an EXTI channel	
Input	GPIO_PIN_NUM_n	Specify the GPIO pins to be configured Note: This parameter can have one of the following values: GPIO_PIN_NUM_n(n=0, 1, 2…)
	GPIO_Px	GPIO Port setting Note: This parameter can have one of the following values: GPIO_Px(x=A, B, C, etc.)

Table 46. AFIO API - AFIO_EXTISourceConfig()

- DeInit API

Function	void AFIO_DeInit (void)
Function	De-initialise the AFIO port peripheral registers to the default reset values

Table 47. AFIO API - AFIO_DeInit()

AFIO API Usage Examples

This section provides a brief description of the AFIO API usage examples, allowing users to gain a better understanding of how to use the AFIO API through the following content.

- AFIO_GPxConfig(): Refer to Table 45 for this API description, which is used to set the alternative mode for the specified GPIO pin. The AFIO clock in the CKCU must first be enabled before using this API.

Program Examples	{ /* Enable peripheral clock */ CKCU_PeripClockConfig_TypeDef CKCUClock = {{ 0 }}; CKCUClock.Bit.AFIO = 1; CKCU_PeripClockConfig(CKCUClock, ENABLE); }

Table 48. Enable CKCU Clocking for AFIO

When the AFIO alternative mode is required to be switched to a GPIO or ADC function, the AFIO_GPxConfig() API is configured as shown in Table 49 below.

Note: Taking the HT32F52352 as an example, there are two ways to set the AFIO_GPxConfig() API.

- Use the AFIO_MODE_Enum typedef way: as shown in the commented example (//AFIO_GPxConfig(...)), where GPIO mode is set to AFIO_MODE_1 and ADC mode is AFIO_ MODE_2.
- Use the FUN Macro way (recommended way, refer to Table 45 for description): The setting is shown in red.

Program Examples	/* Configure GPIO Port A Pin 1 AFIO mode as GPIO function */ **//AFIO_GPxConfig(GPIO_PA, AFIO_PIN_1, AFIO_MODE_1);** **AFIO_GPxConfig(GPIO_PA, AFIO_PIN_1, AFIO_FUN_GPIO);** /* Configure GPIO Port A Pin 1 AFIO mode as ADC function */ **//AFIO_GPxConfig(GPIO_PA, AFIO_PIN_1, AFIO_MODE_2);** **AFIO_GPxConfig(GPIO_PA, AFIO_PIN_1, AFIO_FUN_ADC0);**

Table 49. AFIO Switching Example - AFIO_GPxConfig()

- AFIO_EXTISourceConfig(): This API is used to select the specified GPIO pin as the EXTI trigger source. For example, using GPIO PA1 as the EXTI1 trigger source, the settings are shown in Table 50 below. Then refer to the "EXTI/NVIC API Usage Examples" section for detailed EXTI setting examples.

Program Examples	/* Configure GPIO Port A Pin 1 as EXTI1 Trigger Source */ AFIO_EXTISourceConfig(GPIO_PIN_NUM_1, GPIO_PA);

Table 50. EXTI Trigger Source Setting Examples - AFIO_EXTISourceConfig()

I/O Default AFIO Mode

Taking the HT32F52352 MCU as an example, the I/O peripheral AFIO of the device provides 16 function options, while the default function of the system is the AFIO Mode 0 (AF0). Most of I/Os default state are GPIO functions (for AFIO MODE 1 - GPIO function). Table 51 lists I/Os that do not have GPIO Port functions in the default state of the HT32F52352. Table 51 summarizes the I/Os in the HT32F52352 that do not default to GPIO functions. Users can refer to the AFIO chapter if required to switch these I/Os to GPIOs or other alternate functions.

AF0	AF1
X32KIN	PB10
X32KOUT	PB11
RTCOUT	PB12
XTALIN	PB13
XTALOUT	PB14
SWCLK	PA12
SWDIO	PA13

Table 51. HT32F52352 Default Non-GPIO I/O Corresponding Lists (AF0 | AF1)

Note:

1. Since different MCUs have different package types, their pin alternative functions might also differ. Refer to the Pin Assignment table in the device datasheet for a detailed pin alternative functional description.
2. Although the I/O may default to the AFIO MODE 0, which is the GPIO function, it is recommended that users still execute the procedure of switching the I/O to AFIO MODE 1 - GPIO to quickly understand the I/O planning situation. This can be achieved by using the AFIO_GPxConfig() API code. For example to configure PA1 as a GPIO function, the following code can be used:

```
AFIO_GPxConfig(GPIO_PA, AFIO_PIN_1, AFIO_FUN_GPIO);
```

I/O External Interrupt/Event Controller - EXTI

This chapter will introduce the HT32 EXTI, including the EXTI registers, setting up a GPIO as an external interrupt Pin along with the external interrupt settings.

EXTI/NVIC Introduction

The HT32F52352 External Interrupt/Event Controller, EXTI, comprises 16 edge detectors which can generate a wake-up event or interrupt request independently.

In the interrupt mode there are five trigger types. These are low level, high level, negative edge, positive edge and both edges. These are selected using the SRCnTYPE field in the EXTICFGRn (n=0~15) register. In the wake-up event mode, the wake-up event polarity can be configured by setting the EXTIWAKUPPOLR register. If the EVWUPIEN bit in the EXTIWAKUPCR Register is set, the EVWUP interrupt can be generated when the associated wake-up event occurs and the corresponding EXTI wake-up enable bit is set. The overall EXTI block diagram is shown in Figure 16.

Note:

1. Each EXTI line can also be masked independently.
2. Since different MCUs have different specifications, the available resources for EXTI might also differ. The following EXTI explanation is based on the HT32F52352 specification.

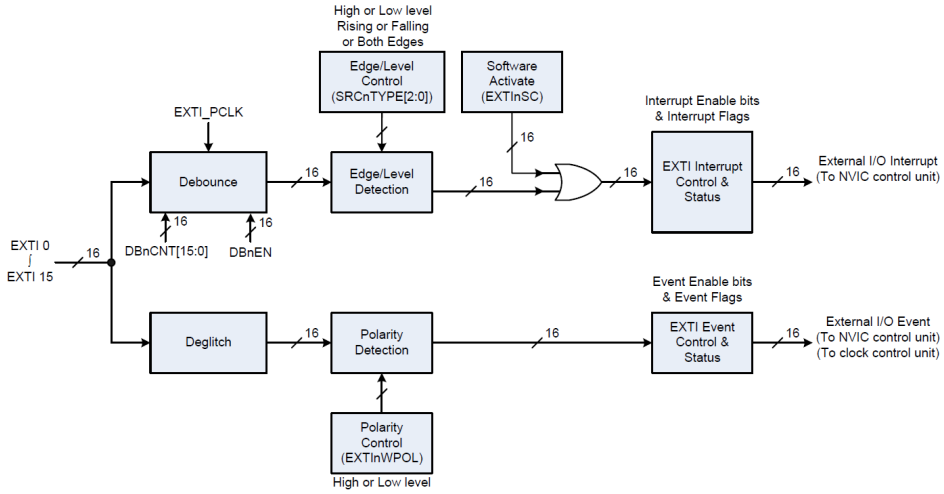


Figure 16. HT32F52352 EXTI Block Diagram

As shown in Figure 16, it can be seen that the EXTI Interrupt/Event Control & Status ultimately outputs to the NVIC (Nested Vectored Interrupt Controller) Interrupt Controller. The NVIC is part of the Cortex-Mx core and is responsible for managing the Cortex-Mx vector interrupts. Therefore, it can be used to enable the NVIC interrupts, set the interrupt priorities and so on, which is important for users whose applications require interrupts. Refer to the "EXTI/NVIC API Description" and "EXTI/NVIC API Usage Examples" for a related API description and usage examples.

Regarding the EXTI input channels, all GPIO pins can be selected as an EXTI trigger source. Taking the HT32F52352 as an example, there are a total of 16 EXTI input channels, channels 0~15. The GPIO pins are connected to the 16 EXTI channels according to the pin numbers as shown in Figure 17. As can be seen from the following figure, the GPIO pins are grouped into EXTI channels according to their pin numbers. Therefore, for the HT3F52352, there can be up to 16 external interrupts requests from the GPIO pins (Note).

Note: Since different MCUs have different specifications, the EXTI channel numbers might also differ. Refer to the corresponding device datasheet for detailed EXTI usage.

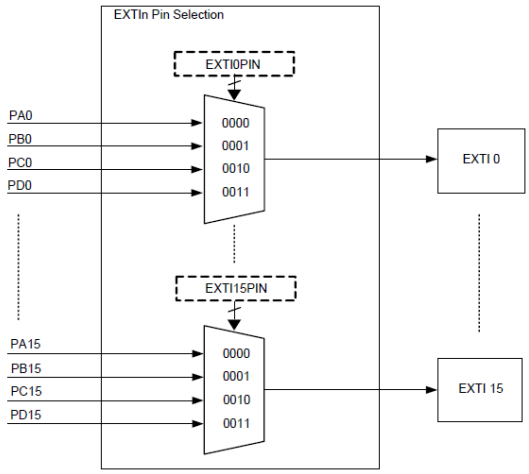


Figure 17. HT32F52352 EXTI Input Channel Selection

The EXTI interrupt debounce function is shown in Figure 18. As shown in the figure, because the low pulse duration is shorter than the setup debounce time, an EXTI interrupt will not be generated. When the low pulse duration is greater than the debounce time, it will generate an EXTI interrupt request effectively. The EXTI debounce function can be configured by setting the DBnEN bit in the EXTIn Interrupt Configuration Register EXTICFGRn (n=0~15). This will enable the corresponding pin de-bounce function and configure the DBnCNT field in the EXTICFGRn so as to select an appropriate application de-bounce time.

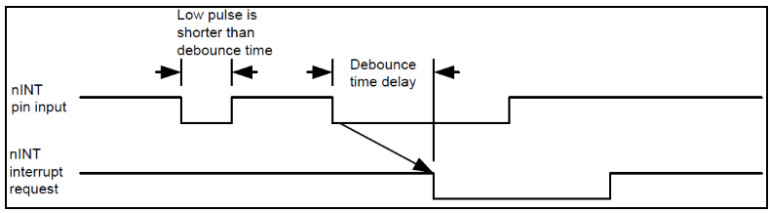
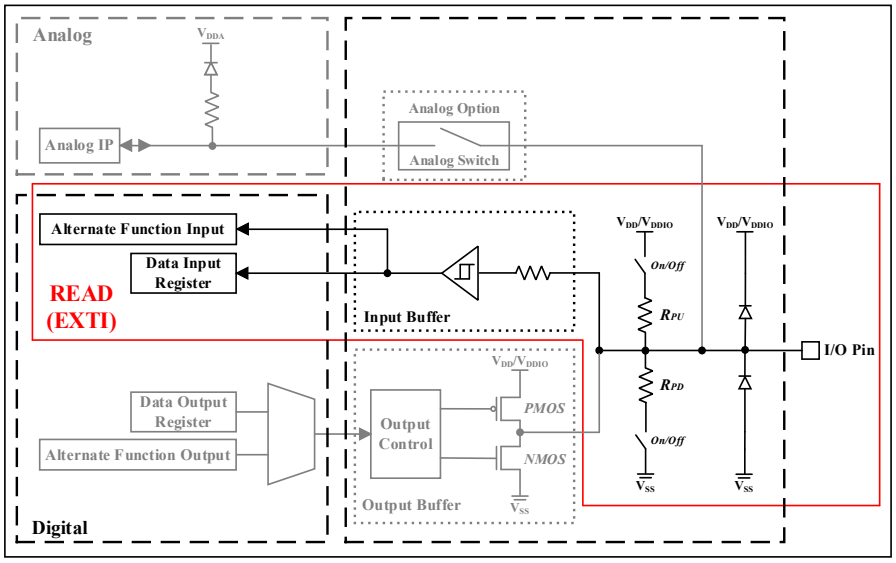


Figure 18. EXTI Interrupt Debounce Function Schematic Diagram

The following are important EXTI characteristic considerations:

(1) Using the EXTI requires enabling the Input Enable function.

As shown in the red box in the figure below, the EXTI function is to be added to the "Figure 1-HT32 MCU Simplified GPIO Architecture Diagram" position. The EXTI function requires that the I/O status is obtained by reading the Data Input Register. Therefore, to use the EXTI, the Input Enable function needs to be enabled. After setting the interrupt trigger condition and interrupt processing program in this way, the system will respond when an external interrupt event occurs. The Input Enable function needs to be set through the PxINER register. Refer to Table 3 for a register description and Table 14 for the API description.



Note: V_{DDIO} is only supported by some MCUs, refer to the corresponding Datasheet for details.

Figure 19. HT32 MCU Simplified GPIO Architecture Diagram - READ (EXTI) in Red Box

(2) Interrupt trigger condition flag. An edge trigger requires writing a "1" to the corresponding flag bit to clear it, while level triggering does not require this:

From the previous section, there are five trigger types, which are classified as follows based on edge triggering and level triggering.

- **Edge trigger**

  Positive edge, Negative edge, Both edges

- **Level trigger**

  High level, Low level

If an edge trigger EXTI interrupt type is selected, when an edge trigger interrupt event occurs, the corresponding edge trigger flag will be set. The flag should be cleared by writing a "1" to the corresponding bits in the EXTIEDGEFLGR and EXTIEDGESR registers. Refer to Table 54 and Table 55 for details of these two registers. If a level trigger EXTI interrupt type is selected, there will be no trigger condition flag set when a level trigger interrupt event occurs, so there is no need to clear the flag.

EXTI Register Introduction

This chapter introduces the HT32 MCU EXTI registers. As different MCUs have different specifications, the available register resources might also differ. In this chapter, the following register introduction is based on the HT32F52352.

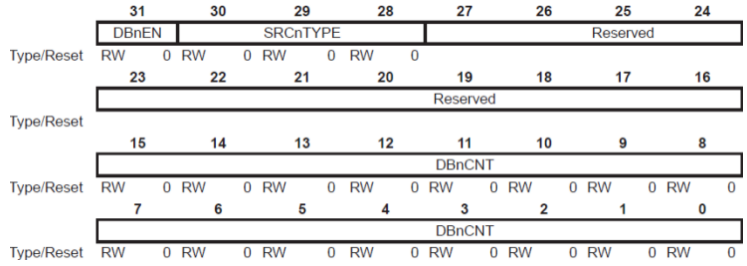
Register	EXTICFGRn Register - EXTI Interrupt Configuration Register n, n = 0~15		
Descriptions	This register is used to specify the debounce function and select the trigger type		
Bitmap			
Bits	[31]	[30:28]	[15:0]
Field	DBnEN	SRCnTYPE[2:0]	DBnCNT
Descriptions	**EXTIn** De-bounce Circuit Enable Bit 0: De-bounce circuit is disabled 1: De-bounce circuit is enabled	EXTIn Interrupt Source Trigger Type 000: Low-level Triggered 001: High-level Triggered 010: Negative-edge Triggered 011: Positive-edge Triggered 1xx: Both-edge Triggered x: don't care	EXTIn De-bounce Counter The de-bounce time is DBnCNT APB(EXTI_PCLK) The clock cycle should be long enough to take effect on the input signal.

Table 52. EXTICFGRn Register

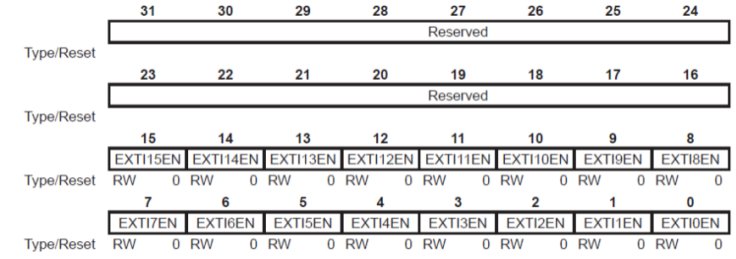
Register	EXTICR Register (EXTI Interrupt Control Register)
Descriptions	This register is used to control the EXTI interrupt
Bitmap	
Bits	[15:0]
Field	EXTInEN
Descriptions	EXTIn Interrupt Enable Bit (n=0~15) 0: EXTI line n interrupt is disabled 1: EXTI line n interrupt is enabled

Table 53. EXTICR Register

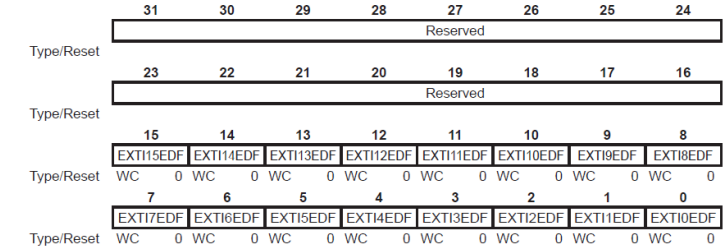
Register	EXTIEDGEFLGR Register (EXTI Interrupt Edge Flag Register)
Descriptions	This register is used to indicate if an EXTI edge has been detected
Bitmap	
Bits	[15:0]
Field	EXTInEDF
Descriptions	EXTIn Both Edge Detection Flag (n=0~15) 0: No edge is detected 1: Positive or negative edge is detected Note: This bit is set by the hardware circuitry when a positive or negative edge is detected on the corresponding EXTI line. The software should write 1 to clear it.

Table 54. EXTIEDGEFLGR Register

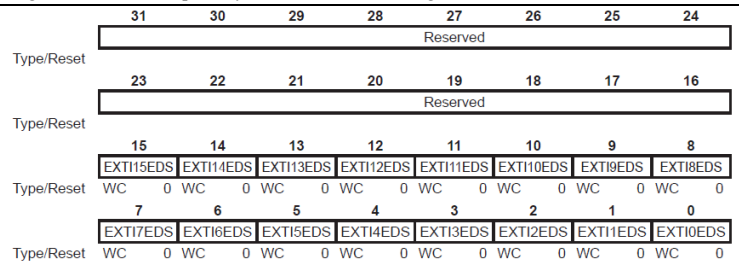
Register	EXTIEDGESR Register (EXTI Interrupt Edge Status Register)
Descriptions	This register indicates the polarity of a detected EXTI edge
Bitmap	
Bits	[15:0]
Field	EXTInEDS
Descriptions	EXTIn Both Edge Detection Status (n=0~15) 0: Negative edge is detected 1: Positive edge is detected Note: The software should write 1 to clear it.

Table 55. EXTIEDGESR Register

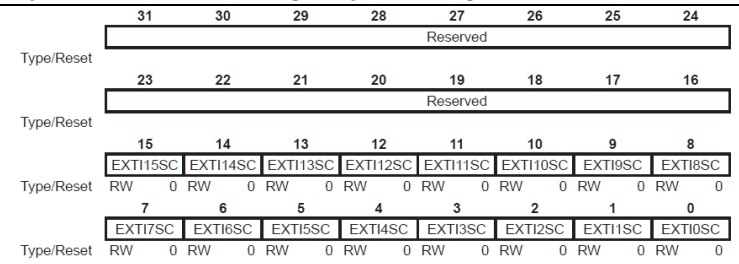
Register	EXTISSCR Register (EXTI Interrupt Software Set Command Register)
Descriptions	This register is used to enable the corresponding EXTI interrupt
Bitmap	
Bits	[15:0]
Field	EXTInSC
Descriptions	EXTIn Software Set Command (n=0~15) 0: The corresponding EXTI interrupt is disabled 1: The corresponding EXTI interrupt is enabled

Table 56. EXTISSCR Register

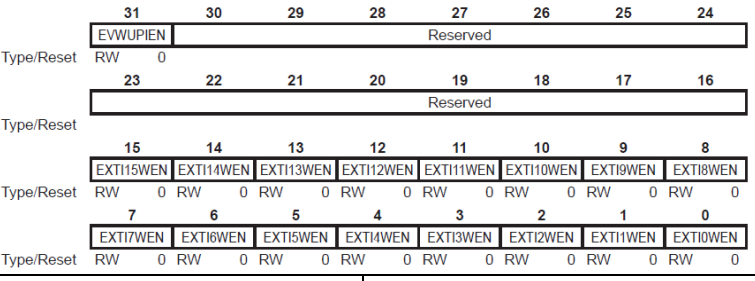
Register	EXTIWAKUPCR Register (EXTI Interrupt Wakeup Control Register)	
Descriptions	This register is used to control the EXTI interrupt and wakeup function	
Bitmap		
Bits	[31]	[15:0]
Field	EVWUPIEN	EXTInWEN
Descriptions	EXTI Event Wakeup Interrupt Enable Bit 0: Disable EVWUP interrupt 1: Enable EVWUP interrupt	EXTIn Wakeup Enable Bit (n=0~15) 0: Power saving mode wakeup is disabled 1: Power saving mode wakeup is enabled

Table 57. EXTIWAKUPCR Register

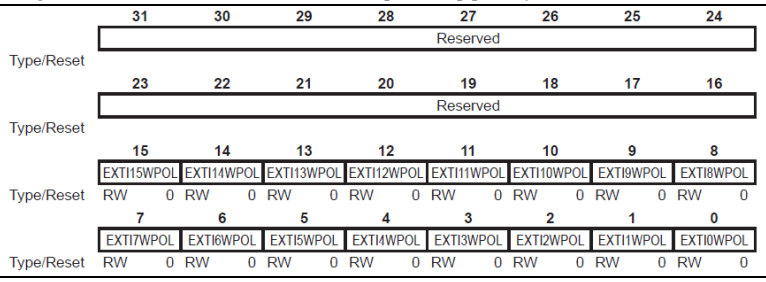
Register	EXTIWAKUPPOLR Register (EXTI Interrupt Wakeup Polarity Register)
Descriptions	This register is used to select the EXTI line interrupt wakeup polarity
Bitmap	
Bits	[15:0]
Field	EXTInWPOL
Descriptions	EXTIn Wakeup Polarity (n=0~15) 0: EXTIn wakeup is high level active 1: EXTIn wakeup is low level active

Table 58. EXTIWAKUPPOLR Register

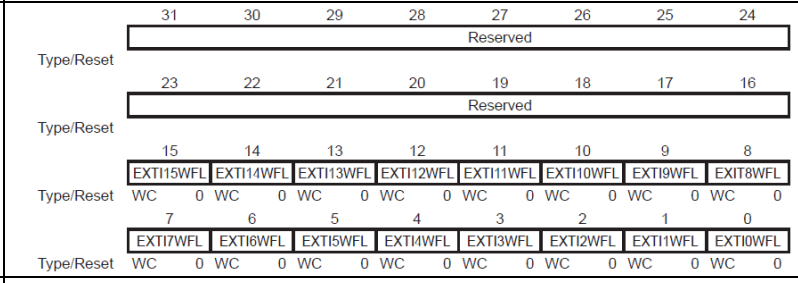
Register	EXTIWAKUPFLG Register (EXTI Interrupt Wakeup Flag Register)
Descriptions	This register is the EXTI interrupt wake flag register
Bitmap	
Bits	[15:0]
Field	EXTInWFL
Descriptions	EXTIn Wakeup Flag (n=0~15) 0: No wakeup occurs 1: System is woken up by EXTIn Note: The software should write 1 to reset it.

Table 59. EXTIWAKUPFLG Register

EXTI/NVIC API Description

This chapter introduces the EXTI API classification. Through this chapter, users can quickly understand the EXTI API description and usage.

EXTI API Data Type Description

The EXTI basic configuration structure (EXTI_InitTypeDef), including Channel, Debounce, DebounceCnt and IntType, is composed as follows:

Variable Name	Types	Description
EXTI_Channel	u32	EXTI channel n (n=0~15)
EXTI_Debounce	EXTI_Deb_TypeDef	EXTI debounce function switch (0 is off, 1 is on)
EXTI_DebounceCnt	u16	Debounce counter setting
EXTI_IntType	EXTI_Interrupt_TypeDef	EXTI interrupt trigger type

Table 60. EXTI Basic Configuration Structure

EXTI API Basic Configuration

Before using the API function, the main program needs to perform an EXTI basic configuration. Table 61 illustrates the EXTI basic configuration. In this example, the EXTI channel 1 is enabled, the debounce function is enabled and the debounce count is configured to 0xFFFF. The interrupt trigger type is a negative edge type. Users can refer to Table 61 for the required settings.

Program Example	{ /* Configure EXTI Channel n as rising edge trigger */ /* !!! NOTICE !!! Notice that the local variable (structure) did not have an initial value. Please confirm that there are no missing members in the parameter settings below in this function. */ EXTI_InitTypeDef EXTI_InitStruct; EXTI_InitStruct.EXTI_Channel = EXTI_CHANNEL_1; EXTI_InitStruct.EXTI_Debounce = EXTI_DEBOUNCE_ENABLE; EXTI_InitStruct.EXTI_DebounceCnt = 0xFFFF; EXTI_InitStruct.EXTI_IntType = EXTI_NEGATIVE_EDGE; EXTI_Init(&EXTI_InitStruct); }

Table 61. EXTI Basic Configuration Example

EXTI API Description

This section will discuss the EXTI API classification allowing users to quickly understand the EXTI API function and its usage. Table 62 shows the EXTI API summary table. The detailed description of each API will be described later.

Classification EXTI Register	API	Description
EXTICR	void EXTI_IntConfig()	Enable or disable the specified EXTI channel x interrupt
EXTIEDGEFLGR	FlagStatus EXTI_GetEdgeFlag()	Obtain the specified EXTI channel x edge flag
EXTIEDGEFLGR & EXTIEDGESR	FlagStatus EXTI_GetEdgeStatus()	Obtain the specified EXTI channel x edge state
	void EXTI_ClearEdgeFlag()	Clear the specified EXTI channel x edge flag
EXTISSCR	void EXTI_SWIntCmd()	Enable or disable specified EXTI channel x interrupt through software
	FlagStatus EXTI_GetSWCmdStatus()	Obtain the specified EXTI channel x software command bit
EXTIWAKUPCR	void EXTI_WakeupEventIntConfig()	Enable or disable EXTI channel x event wakeup interrupt
EXTIWAKUPCR & EXTIWAKUPPOLR	void EXTI_WakeupEventConfig()	Configure the EXTI channel x event wake-up function
EXTIWAKUPFLG	FlagStatus EXTI_GetWakeupFlagStatus()	Obtain the specified EXTI channel x wake-up flag
	void EXTI_ClearWakeupFlag()	Clear the specified EXTI channel x wake-up flag
Init / DeInit API		
-	void EXTI_Init()	Initialise the EXTI peripheral
-	void EXTI_DeInit()	De-initialise the EXTI peripheral

Table 62. EXTI API Summary Table

- Init / DeInit

Function	void EXTI_Init (EXTI_InitTypeDef* EXTI_InitStruct)	
Function	Initialise the EXTI peripheral	
Input	EXTI_InitStruct	A pointer to the EXTI_InitTypeDef structure

Table 63. EXTI API - EXTI_Init()

Function	void EXTI_DeInit (u32 EXTI_Channel)	
Function	De-initialise the EXTI peripheral	
Input	EXTI_Channel	EXTI channel (0~15) Note: This parameter can have one of the following values: EXTI_CHANNEL_0 EXTI_CHANNEL_1 … EXTI_CHANNEL_14 EXTI_CHANNEL_15

Table 64. EXTI API - EXTI_DeInit()

- EXTICR Register

Function	void EXTI_IntConfig (u32 EXTI_Channel, ControlStatus NewState)	
Function	Enable or disable the specified EXTI channel x interrupt	
Input	EXTI_Channel	EXTI channel (0~15) Note: This parameter can have one of the following values: EXTI_CHANNEL_0 EXTI_CHANNEL_1 … EXTI_CHANNEL_14 EXTI_CHANNEL_15
	NewState	Enable or disable Note: This parameter can have one of the following values: ENABLE, DISABLE

Table 65. EXTI API - EXTI_IntConfig()

- EXTIEDGEFLGR Register

Function	FlagStatus EXTI_GetEdgeFlag (u32 EXTI_Channel)	
Function	Obtain the specified EXTI channel x edge flag	
Input	EXTI_Channel	EXTI channel (0~15) Note: This parameter can have one of the following values: EXTI_CHANNEL_0 EXTI_CHANNEL_1 … EXTI_CHANNEL_14 EXTI_CHANNEL_15
Output	SET	Detected EXTI channel x edge state (Rising or falling edge)
	RESET	No edge state detected

Table 66. EXTI API - EXTI_GetEdgeFlag()

- EXTIEDGEFLGR & EXTIEDGESR Register

Function	FlagStatus EXTI_GetEdgeStatus (u32 EXTI_Channel, u32 EXTI_Edge)	
Function	Obtain the specified EXTI channel x edge state	
Input	EXTI_Channel	EXTI channel (0~15) Note: This parameter can have one of the following values: EXTI_CHANNEL_0 EXTI_CHANNEL_1 … EXTI_CHANNEL_14 EXTI_CHANNEL_15
	EXTI_Edge	Monitored edge state Note: This parameter can have one of the following values: EXTI_EDGE_POSITIVE EXTI_EDGE_NEGATIVE
Output	SET	Detected monitored EXTI channel x edge state
	RESET	The edge state of the detected EXTI channel x has not been detected

Table 67. EXTI API - EXTI_GetEdgeStatus()

Function	void EXTI_ClearEdgeFlag (u32 EXTI_Channel)	
Function	Clear the specified EXTI channel x edge flag	
Input	EXTI_Channel	EXTI channel (0~15) Note: This parameter can have one of the following values: EXTI_CHANNEL_0 EXTI_CHANNEL_1 … EXTI_CHANNEL_14 EXTI_CHANNEL_15

Table 68. EXTI API - EXTI_ClearEdgeFlag()

- EXTISSCR Register

Function	void EXTI_SWIntCmd (u32 EXTI_Channel, ControlStatus NewState)	
Function	Enable or disable specified EXTI channel x interrupt through software	
Input	EXTI_Channel	EXTI channel (0~15) Note: This parameter can have one of the following values: EXTI_CHANNEL_0 EXTI_CHANNEL_1 … EXTI_CHANNEL_14 EXTI_CHANNEL_15
	NewState	Enable or disable Note: This parameter can have one of the following values: ENABLE, DISABLE

Table 69. EXTI API - EXTI_SWIntCmd()

Function	FlagStatus EXTI_GetSWCmdStatus (u32 EXTI_Channel)	
Function	Obtain the specified EXTI channel x software command bit	
Input	EXTI_Channel	EXTI channel (0~15) Note: This parameter can have one of the following values: EXTI_CHANNEL_0 EXTI_CHANNEL_1 … EXTI_CHANNEL_14 EXTI_CHANNEL_15
Output	SET	EXTI channel x software command bit - enable state
	RESET	EXTI channel x software command bit - disable state

Table 70. EXTI API - EXTI_GetSWCmdStatus ()

- EXTIWAKUPCR

Function	void EXTI_WakeupEventIntConfig (ControlStatus NewState)	
Function	Enable or disable EXTI channel x event wakeup interrupt	
Input	NewState	Enable or disable Note: This parameter can have one of the following values: ENABLE, DISABLE

Table 71. EXTI API - EXTI_WakeupEventIntConfig()

- EXTIWAKUPCR & EXTIWAKUPPOLR Register

Function	void EXTI_WakeupEventConfig (u32 EXTI_Channel, u8 EXTI_WakeUpType, ControlStatus NewState)	
Function	Configure the EXTI channel x event wake-up function	
Input	EXTI_Channel	EXTI channel (0~15) Note: This parameter can have one of the following values: EXTI_CHANNEL_0 EXTI_CHANNEL_1 … EXTI_CHANNEL_14 EXTI_CHANNEL_15
	EXTI_WakeUpType	Determines the signal trigger type that triggers the EXTI interrupt Note: This parameter can have one of the following values: EXTI_WAKEUP_HIGH_LEVEL EXTI_WAKEUP_LOW_LEVEL
	NewState	Enable or disable Note: This parameter can have one of the following values: ENABLE, DISABLE

Table 72. EXTI API - EXTI_WakeupEventConfig()

- EXTIWAKUPFLG Register

Function	FlagStatus EXTI_GetWakeupFlagStatus (u32 EXTI_Channel)	
Function	Obtain the specified EXTI channel x wake-up flag	
Input	EXTI_Channel	EXTI channel (0~15) Note: This parameter can have one of the following values: EXTI_CHANNEL_0 EXTI_CHANNEL_1 … EXTI_CHANNEL_14 EXTI_CHANNEL_15
Output	SET	EXTI channel x wake-up flag - set state
	RESET	EXTI channel x wake-up flag - reset state

Table 73. EXTI API - EXTI_GetWakeupFlagStatus()

Function	void EXTI_ClearWakeupFlag (u32 EXTI_Channel)	
Function	Clear the specified EXTI channel x wake-up flag	
Input	EXTI_Channel	EXTI channel (0~15) Note: This parameter can have one of the following values: EXTI_CHANNEL_0 EXTI_CHANNEL_1 … EXTI_CHANNEL_14 EXTI_CHANNEL_15

Table 74. EXTI API - EXTI_ClearWakeupFlag()

NVIC API Description

Common NVIC API description: This section will introduce the NVIC API classification. Here, users can quickly understand the NVIC API function and usage. Table 75 shows the NVIC API summary table. The detailed description of each API will be described later.

Classification	API	Description
Enable/disable External Interrupt	**void NVIC_EnableIRQ()**	Enable the device specific interrupt in the NVIC interrupt controller
	void NVIC_DisableIRQ()	Disable the device specific interrupt in the NVIC interrupt controller
	uint32_t NVIC_GetEnableIRQ()	Return the device specific interrupt enable status from the NVIC interrupt controller
Set/Obtain Interrupt Priority Level	**void NVIC_SetPriority()**	Set the device specific interrupt priority
	uint32_t NVIC_GetPriority()	Read the device specific interrupt priority
Set/Obtain Pending Interrupt	**void NVIC_SetPendingIRQ()**	Set the device specific interrupt pending bit in the NVIC set pending register
	uint32_t NVIC_GetPendingIRQ()	Read the NVIC pending register and return the pending bit for the device specific interrupt
	void NVIC_ClearPendingIRQ()	Clear the device specific interrupt pending bit in the NVIC clear pending register

Table 75. NVIC API Summary Table

- Enable/Disable External Interrupt Related API

Function	**__STATIC_INLINE void __NVIC_EnableIRQ (IRQn_Type IRQn)**	
Macro Define	**#define NVIC_EnableIRQ __NVIC_EnableIRQ**	
Function	Enable the device specific interrupt in the NVIC interrupt controller	
Input	IRQn	Device specific interrupt number Note: IRQn can't be negative

Table 76. NVIC API - NVIC_EnableIRQ()

Function	**__STATIC_INLINE void __NVIC_DisableIRQ (IRQn_Type IRQn)**	
Macro Define	**#define NVIC_DisableIRQ __NVIC_DisableIRQ**	
Function	Disable the device specific interrupt in the NVIC interrupt controller	
Input	IRQn	Device specific interrupt number Note: IRQn cannot be negative

Table 77. NVIC API - NVIC_DisableIRQ()

Function	**__STATIC_INLINE uint32_t __NVIC_GetEnableIRQ (IRQn_Type IRQn)**	
Macro Define	**#define NVIC_GetEnableIRQ __NVIC_GetEnableIRQ**	
Function	Return the device specific interrupt enable status from the NVIC interrupt controller	
Input	IRQn	Device specific interrupt number Note: IRQn cannot be negative
Output	0	Interrupt is disabled
	1	Interrupt is enabled

Table 78. NVIC API - NVIC_GetEnableIRQ()

- Set/Obtain Interrupt Priority related API

Function	**__STATIC_INLINE void __NVIC_SetPriority (IRQn_Type IRQn, uint32_t priority)**	
Macro Define	**#define NVIC_SetPriority __NVIC_SetPriority**	
Function	Set the device specific interrupt priority	
Input	IRQn	Device specific interrupt number Note: IRQn cannot be negative
	Priority	Priority to be set

Table 79. NVIC API - NVIC_SetPriority()

Function	**__STATIC_INLINE uint32_t __NVIC_GetPriority (IRQn_Type IRQn)**	
Macro Define	**#define NVIC_GetPriority __NVIC_GetPriority**	
Function	Read the device specific interrupt priority	
Input	IRQn	Device specific interrupt number Note: IRQn cannot be negative
Output	Interruption priority	

Table 80. NVIC API - NVIC_GetPriority()

- Set/et Pending Interrupt Related APIs

Function	__STATIC_INLINE void __NVIC_SetPendingIRQ (IRQn_Type IRQn)	
Macro Define	#define NVIC_SetPendingIRQ __NVIC_SetPendingIRQ	
Function	Set the device specific interrupt pending bit in the NVIC set pending register	
Input	IRQn	Device specific interrupt number Note: IRQn cannot be negative

Table 81. NVIC API - NVIC_SetPendingIRQ()

Function	__STATIC_INLINE uint32_t __NVIC_GetPendingIRQ (IRQn_Type IRQn)	
Macro Define	#define NVIC_GetPendingIRQ __NVIC_GetPendingIRQ	
Function	Read the NVIC pending register and return the pending bit for the device specific interrupt	
Input	IRQn	Device specific interrupt number Note: IRQn cannot be negative
Output	0	Interrupt status: Not pending
	1	Interrupt status: pending

Table 82. NVIC API - NVIC_GetPendingIRQ()

Function	__STATIC_INLINE void __NVIC_ClearPendingIRQ (IRQn_Type IRQn)	
Macro Define	#define NVIC_ClearPendingIRQ __NVIC_ClearPendingIRQ	
Function	Clear the device specific interrupt pending bit in the NVIC clear pending register	
Input	IRQn	Device specific interrupt number Note: IRQn cannot be negative

Table 83. NVIC API - NVIC_ClearPendingIRQ()

EXTI/NVIC API Usage Examples

In this chapter, the EXTI/NVIC API is illustrated through a simple modifications of the HT32 MCU Firmware Library example. This allows users to gain a better understanding of the EXTI/NVIC API usage through the example programs.

Refer to the HT32 MCU Firmware Library EXTI example for modification.

- The contents related to Include, Function and Variable are shown in the following Table 84.

```
/* Includes ------------------------------------------------------------------------------------------------------*/
#include "ht32.h"
#include "ht32_board.h"

/* Private function prototypes -----------------------------------------------------------------------------------*/
void EXTI_Configuration(void);
void EXTI_MainRoutine(void);

/* Global variables ----------------------------------------------------------------------------------------------*/
vu32 guEXTIState[1];
```

Table 84. EXTI/NVIC API Usage Examples - Related Include, Function and Variable

- The main function includes HT_LED1 initialisation (Note) →CKCU setup function→EXTI setup function →EXTI main loop test function, refer to the following content for a detailed function description.

  Note: HT_LED1 is a pin definition for quick use of the HT32 Starter Kit pins. For different HT32 MCUs, the pins used are not entirely the same. Refer to ht32fxxxxx_sk.h to confirm the pin settings (xxxxxx is the Starter Kit IC device).

```
/* Global functions ------------------------------------------------------------------------------------------*/
int main(void)
{
  HT32F_DVB_LEDInit(HT_LED1); /* HT_LED1 Init */
  CKCU_Configuration(); /* System Related configuration */
  /* Configure PA1 pin as the input function and enable exti relate function */
  EXTI_Configuration();
  while (1)
  {
    EXTI_MainRoutine();
  }
}
```

Table 85. EXTI/NVIC API Usage Examples - main function

- CKCU_Configuration(): The I/O port and EXTI clocks in the CKCU must first be enabled before setting the EXTI. In this section, the example uses PA1, therefore, it is necessary to enable the CKCU clock for the AFIO, Port A, and EXTI. The description and the sample program code are shown in Table 86.

Function	void CKCU_Configuration (void)
Function Description	Set the system clock (Enable AFIO, Port A and EXT clocks)
Program Examples	void CKCU_Configuration(void) { CKCU_PeripClockConfig_TypeDef CKCUClock = {{ 0 }}; CKCUClock.Bit.AFIO = 1; CKCUClock.Bit.EXTI = 1; CKCUClock.Bit.PA = 1; CKCU_PeripClockConfig(CKCUClock, ENABLE); }

Table 86. EXTI/NVIC API Usage Examples - CKCU_Configuration()

- EXTI_Configuration(): Set the EXTI I/O (PA1) input function and related settings as shown in Table 87.

Function	void EXTI_Configuration (void)
Function Description	EXTI port configuration
Implementation Settings	● AFIO mode: GPIO Mode ● Input Function: Enable ● Pull Resistance: Pull-Disable ● Pin direction: Input ● EXTI trigger source pin: PA1 ● EXTI_InitStruct ➢ EXTI_Channel: CHANNEL_1 ➢ EXTI_Debounce: EXTI_DEBOUNCE_DISABLE ➢ EXTI_DebounceCnt: 0 ➢ EXTI_IntType: EXTI_POSITIVE_EDGE (Positive edge trigger) ● EXTI interrupt setting: Channel 1 enable ● NVIC interrupt setting (note): EXTI1_IRQn

Function	void EXTI_Configuration (void)
Program Examples	void EXTI_Configuration(void) { /* Configure AFIO mode of input pins */ AFIO_GPxConfig(GPIO_PA, AFIO_PIN_1, AFIO_FUN_GPIO); /* Enable GPIO Input Function */ GPIO_InputConfig(HT_GPIOA, GPIO_PIN_1, ENABLE); /* Configure GPIO pull resistor of input pins */ GPIO_PullResistorConfig(HT_GPIOA, GPIO_PIN_1, GPIO_PR_DISABLE); /* Select Port as EXTI Trigger Source */ AFIO_EXTISourceConfig(GPIO_PIN_NUM_1, GPIO_PA); { /* Configure EXTI Channel n as rising edge trigger */ /* !!! NOTICE !!! Notice that the local variable (structure) did not have an initial value. Please confirm that there are no missing members in the parameter settings below in this function. */ EXTI_InitTypeDef EXTI_InitStruct; EXTI_InitStruct.EXTI_Channel = EXTI_CHANNEL_1; EXTI_InitStruct.EXTI_Debounce = EXTI_DEBOUNCE_DISABLE; EXTI_InitStruct.EXTI_DebounceCnt = 0; EXTI_InitStruct.EXTI_IntType = EXTI_POSITIVE_EDGE; EXTI_Init(&EXTI_InitStruct); } /* Enable EXTI & NVIC line Interrupt */ EXTI_IntConfig(EXTI_CHANNEL_1, ENABLE); NVIC_EnableIRQ(EXTI1_IRQn); }

Table 87. EXTI/NVIC API Usage Examples - EXTI_Configuration()

Note: Other related settings of the NVIC depend on different application requirements, such as setting the priority with NVIC_SetPriority(…) and obtaining the specific interrupt enable status with NVIC_GetEnableIRQ(…). Refer to the "EXTI/NVIC API Description" section for a detailed NVIC API description.

The EXTI_MainRoutine() function is described below. Before describing this function, it should be noted that when using the EXTI function, it is necessary to program the function content for this EXTI IRQ. The content may include clearing the edge flag bit, and the actions that are required to be taken after entering the EXTI, etc.

Note: For this interrupt execution function, it is recommended that the HT32 devices program the content in the file, which is specifically for the IRQ handling functions.

This section uses PA1 EXTI, therefore it is required to add EXTI0_1_IRQHandler(). The description is shown in the table below.

- EXTI0_1_IRQHandler(): EXTI 0~1 interrupt processing function, program examples and results are described in Table 88.

Function	void EXTI0_1_IRQHandler (void)
Function Description	EXTI 0~1 interrupt processing function
Program Examples	extern vu32 guEXTIState[1]; void EXTI0_1_IRQHandler(void) { if (EXTI_GetEdgeFlag(EXTI_CHANNEL_1)) { EXTI_ClearEdgeFlag(EXTI_CHANNEL_1); guEXTIState[0] = TRUE; } }
Results	The device enters the EXTI IRQ handler after being triggered by a positive edge, it checks whether the edge trigger flag bit is set. ➔Clear the edge trigger flag and change the guEXTIState state to TRUE. After exiting the EXTI IRQ handler, when entering EXTI_MainRoutine(), perform the subsequent actions. Refer to Table 89

Table 88. EXTI/NVIC API Usage Examples - EXTI0_1_IRQHandler()

- EXTI_MainRoutine(): EXTI main loop test function, refer to the following content for a program example and result description.

Function	void EXTI_MainRoutine (void)
Function Description	EXTI main loop test
Program Examples	void EXTI_MainRoutine(void) { if (guEXTIState[0] == TRUE) { guEXTIState[0] = FALSE; HT32F_DVB_LEDToggle(HT_LED1); } }
Results	When the guEXTIState state is judged to be TRUE ➔Reset the guEXTIState status to FALSE and toggle the HT_LED1. There will be two states: 1. PA1 positive edge trigger➔HT_LED1 Toggle (trigger EXTI interrupt) 2. PA1 no positive edge trigger➔ HT_LED1 remains unchanged 

Table 89. EXTI/NVIC API Usage Example - EXTI_MainRoutine()

I/O Design Considerations

This chapter describes categorised I/O considerations which will allow users to prevent potential problems or quickly resolve problems related to their I/O design.

Unused I/O Recommended Connection Methods

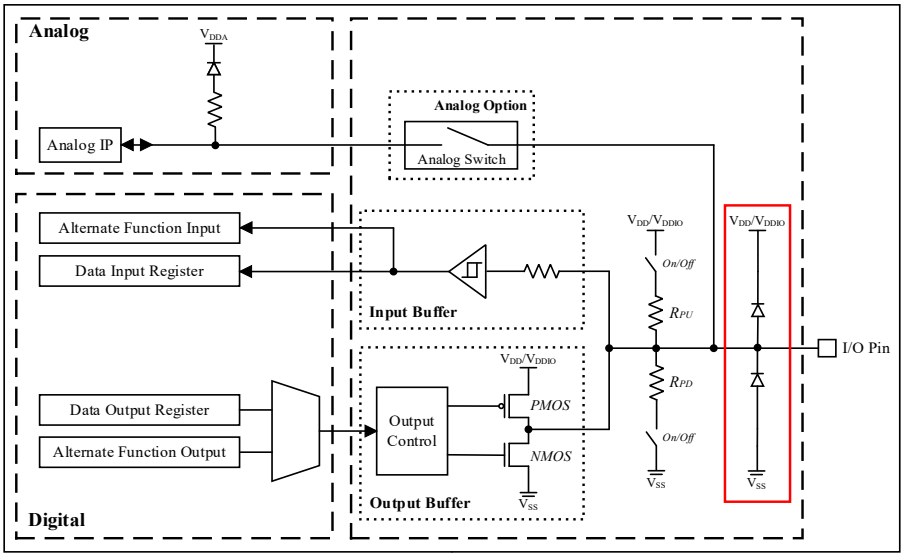
According to specific application demands, the MCU I/O resources may not be fully used. For unused I/O pins, it is recommended not to leave them floating. It is recommended that users consider the following ways to enhance the EFT and ESD circuit capabilities.

- Connect unused I/O pins to GND.
- Set unused I/O pins to the Push-Pull mode and output low using firmware.

ESD Diode Description and Consideration

The HT32 MCU GPIO architecture uses ESD protection diodes as shown in the red box in Figure 20. Therefore, when designing the circuit, special attention should be paid to avoid a situation where a high voltage (e.g., 3.3V) device is connected to the circuit when V_{DD} or V_{DDIO} is not powered-up. This may cause the ESD protection diodes to conduct. This situation will lead to the MCU and the overall circuit power supply conditions behaving as expected, resulting in abnormal operation.

Solution: For example, unexpected situations can be avoided by connecting a level shifter or isolation circuit.



Note: V_{DDIO} is only supported by some MCUs, refer to the corresponding Datasheet for details.

Figure 20. HT32 MCU Simplified GPIO Architecture - ESD Protection Diode in Red Box

IOPOC Control with I/O Power-Up Pulse

During the V_{DD} power-up period and before the MCU core voltage has been established, its internal register status will be unknown, making it impossible to correctly control the I/O outputs. Therefore, the I/O Power On Control (IOPOC) will have to be used to avoid unexpected output stage states (I/O output high or low).

The IOPOC starts to operate when V_{DD} reaches a certain voltage (about 0.8 V) and detects the MCU core voltage. Before the core voltage has stabilised, the IOPOC will isolate the I/O Cell control signals from the MCU core and control the I/O output stage PMOS/NMOS transistors to be in an Off state. It will only release control to the GPIO related registers to control the I/O once the MCU core voltage has been detected to have reached the set voltage.

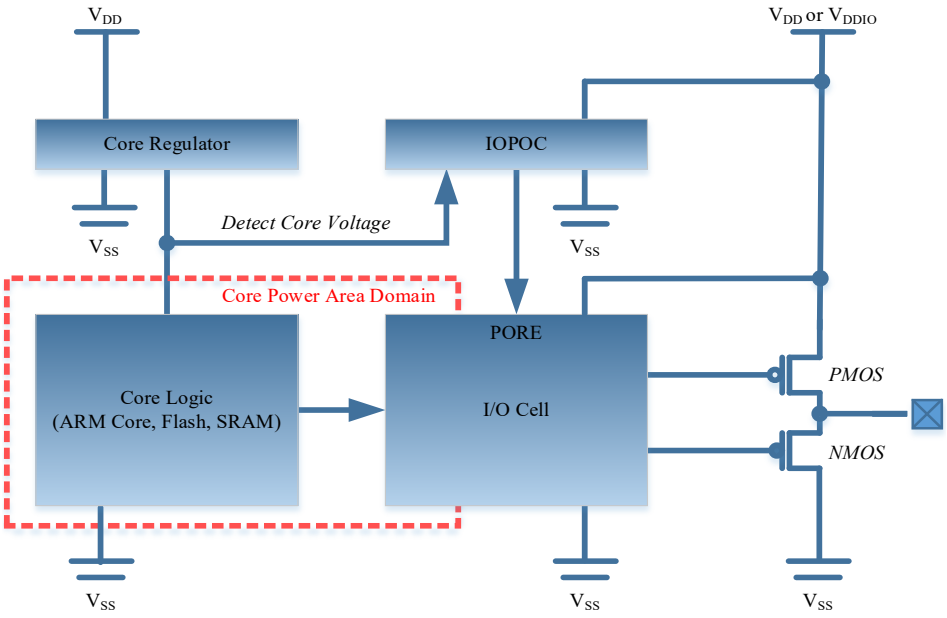


Figure 21. HT32 MCU Internal Design Simplified Diagram

A point to consider is during the V_{DD} power-up process and before the IOPOC starts operating and the HT32 MCU core voltage has been established. If the unknown signal on the PMOS Gate terminal of the output stage happens to be at a logic level"0", and at this time, as long as the PMOS Source terminal voltage is higher than the PMOS Gate terminal voltage by 0.8V (PMOS conduction threshold), then the V_{DD} voltage will be directly applied to the I/O pad due to the output stage PMOS being turned on. Therefore, the I/O Pad may have a short pulse during the V_{DD} power-up time. The possibility of this happening depends upon the rise rate of the V_{DD} voltage. If this pulse creates problems for the application circuit adding a small value capacitor to the I/O port may filter out the pulse.

- Suggested Solution

  Connect a 1~10nF capacitor between GND and the I/O terminal to filter out any short-term surges that may occur during power-on.

GPIO Recommended Configuration in the Low-Power Mode

In order to avoid additional power consumption caused by unknown states or DC Paths, it is recommended that all I/O pins must have a fixed level (V_{DD} or GND) when entering the Low-Power mode.

Practically, the GPIO pin configuration recommendations are related to the GPIO direction, which is described in detail as follows:

- GPIO Output

  When a GPIO is configured as an output pin and connected to an external device (external device input), it is recommended to use the Push-Pull mode to establish the pin state. If a pull-up/pull-down resistor is connected externally, it is necessary to pay attention to the GPIO output state to avoid causing a DC Path from V_{DD} to GND, or temporarily switch the GPIO to the input mode (not driving) when entering the Low-Power mode.

- GPIO Input

  When GPIO is configured as an input pin and connected to an external device (external device output or bus), the external device has to provide a valid level, either V_{DD} or GND.

  If the external device does not provide a valid level, the internal GPIO pin pull-up or pull-down can be enabled to force the I/O pin to a fixed level.

nRST Pin cannot be used as an Enable Pin

In order to achieve the lowest power consumption, the application may design-in an enable pin to enable the device to enter the low power consumption mode.

Since the MCU internal circuit operation is controlled by the nRST pin, when nRST=Low, the MCU is not operating in the lowest power consumption condition, therefore the nRST cannot be used as an enable pin.

When the application requires low power consumption, the nRST can be released to enter a low-power mode (Sleep, Deep-Sleep or Power-Down) by the firmware to have a lower power consumption.

External Interrupt Pin Selection Issues

For external interrupt settings, pins with the same number will be classified into the same external interrupt request channel. For example, PA0, PB0, PC0 and PD0 all use EXTI channel 0 as an external interrupt request channel, as shown in Figure 22. Therefore, at any time, only one pin with the same number can be used as an external interrupt pin for all ports, otherwise it will cause conflicts in the EXTI channels.

Refer to the chapter "I/O External Interrupt/Event Controller (EXTI)" for detailed EXTI usage.

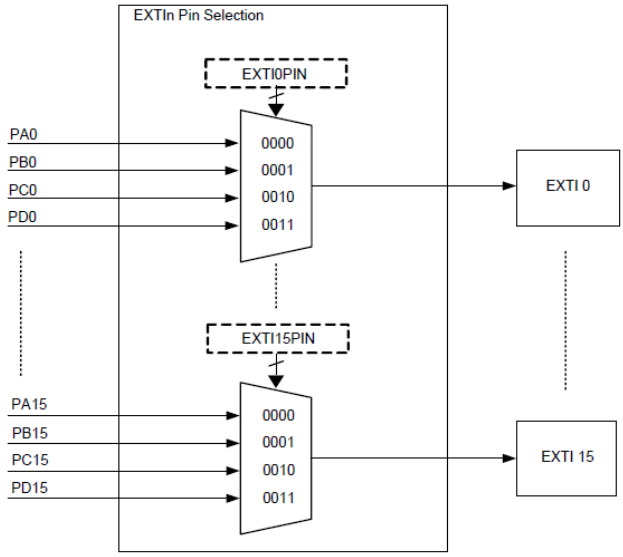


Figure 22. HT32F52352 EXTI Input Channel Selection - Refer to the User Manual

Conclusion

This application note has introduced the HT32 MCU GPIOs, AFIO, EXTI/NVIC in detail, and has given information on I/O design considerations. Users can quickly become familiar with the related use of HT32 MCU GPIOs using the information provided in this application note.

Reference Files

For more information, refer to the Holtek official https://www.holtek.com.

Version and Modification Information

Date	Author	Issue	Modification
2023.11.01	蔡期育	V1.00	First Version

Disclaimer

All information, trademarks, logos, graphics, videos, audio clips, links and other items appearing on this website ('Information') are for reference only and is subject to change at any time without prior notice and at the discretion of Holtek Semiconductor Inc. and its relevant companies (hereinafter 'Holtek', 'the company', 'us', 'we' or 'our'). Whilst Holtek endeavors to ensure the accuracy of the Information on this website, no express or implied warranty is given by Holtek to the accuracy of the Information. Holtek will bear no responsibility for any incorrectness or leakage.

Holtek will not be liable for any damages (including but not limited to computer virus, system problems or data loss) whatsoever arising in using or in connection with the use of this website by any party. There may be links in this area, which allow you to visit the websites of other companies. These websites are not controlled by Holtek. Holtek will bear no responsibility and no guarantee to whatsoever Information displayed at such sites. Hyperlinks to other websites are at your own risk.

Limitation of Liability

In no event shall Holtek Limited be liable to any other party for any loss or damage whatsoever or howsoever caused directly or indirectly in connection with your access to or use of this website, the content thereon or any goods, materials or services.

Governing Law

The Disclaimer contained in the website shall be governed by and interpreted in accordance with the laws of the Republic of China. Users will submit to the non-exclusive jurisdiction of the Republic of China courts.

Update of Disclaimer

Holtek reserves the right to update the Disclaimer at any time with or without prior notice, all changes are effective immediately upon posting to the website.