QingKeV4 Processor Manual

V1.00

Overview

The Qingke V4 microprocessor is based on the standard RISC-V instruction set architecture. An in-house developed 32-bit general-purpose MCU microprocessor. The Qingke V4 series include V4A, V4B, V4C and V4F according to different applications and instruction sets. The V4 series supports extension of RV32IMAC instruction set. The V4F microprocessor supports single-precision hardware floating point, that is, supports RV32IMACF extension. The V4B, V4C and V4F microprocessors also support custom extension XW. In addition, they have features of Hardware Push Enable (HPE), Vector Table Free (VTF), reduced 2-wire debug interface, “WFE” command, Physical Memory Protect (PMP), etc.

Features

Feature	Description
Instruction Set Architecture	RV32IMAC[F]
Pipeline	3-level
Bus Interface	AHB
FPU	Supports single-precision floating point
branch Prediction	BHT/BTB/RAS
Interrupt	Supports 256 interrupts including exceptions, supports Vector Table Free
Hardware Push Enable	Supports up to 3-level hardware push enable
Physical Memory Protect	Supports 4 memory protection regions
Low Power Mode	Supports Sleep and Deep-Sleep modes, supports WFI and WFE sleep modes
Expand Instruction Set	Supports half-word and byte operation compressed instructions
Debug	Enhanced 2-wire debug interface, standard RISC-V debug

Chapter 1 Overview

The Qingke V4 series contains V4A, V4B, V4C and V4F microprocessors. And they have some diffrences according to the applications, see Table 1-1 for details.

Table 1-1 Comparison

Feature / Model	Instruction set	Hardware Push Enable	Interrupt Nested	VTF Interrupt Channel	Pipeline	Vector Table Mode	Instruction Extension（XW）	Memory Protection Regions
V4A	RV32IMAC	2	2	4	3	Address/Command	×	4
V4B	RV32IMAC	2	2	4	3	Address/Command	√	4
V4C	RV32IMAC	2	2	4	3	Address/Command	√	4
V4F	RV32IMACF	3	8	4	3	Address/Command	√	4

Note: For the task switching of the operating system, the software stack is generally selected, as the number of software stacks is not limited.

1.1 Instruction set

The Qingke V4 microprocessor follows the standard RISC-V instruction set architecture. For more about the standard, please refer to The RISC-V Instruction Set Manual which can be found on the RISC-V Foundation official website. The RISC-V instruction set architecture is concise, supports modular design, and can be flexibly combined according to different needs. The Qingke V4 microprocessor supports the following instruction set extensions:

- RV32: 32-bit architecture. The bit width of the general-purpose registers is 32-bit.
- I: Supports integer operations, with 32 integer registers
- M: Supports integer multiply and divide instructions
- A: Supports atomic instructions
- C: Supports 16-bit compressed instructions
- F: Supports single-precision floating-point operations, with 32 floating-point registers
- XW: Self-extend 16-bit compressed instruction for byte and halfword operations

Note: 1: The sub-instruction set supported by different models may differ. For details, please see Table 1-1.

2: To further improve code density and expand the XW subset, add the c.lbu/c.lhu/c.sb/c.sh/c.lbusp/ c.lhusp/c.sbsp/c.shsp compressed instructions, which needs to be based on the MRS compiler or the toolchain provided by it.

1.2 Register group

The RV32I provides 32 register groups (x0-x31) and supports “F” extension. The RV32I provides 32 floating-point register groups (f0-f31) in addition to x0-x31. All above registers of the RV32 are 32-bit.
Table 1-2 shows the RISC-V registers and descriptions.

Table 1-2 RISC-V registers

Register	ABI Name	Description	Saver
x0	zero	Hard code 0	-
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	-
x4	tp	Thread pointer	-
x5-7	t0	Temporary registers	Caller
x8	s0/fp	Save register / frame pointer	Callee
x9	s1	Save register	Callee
x10-11	a0-1	Function parameter / return value	Caller
x12-17	a2-7	Function parameter	Caller
x18-27	s2-11	Save registers	Callee
x28-31	t3-6	Temporary registers	Caller
f0-7	ft0-7	Floating-point temporary registers	Caller
f8-9	fs0-1	Floating-point save registers	Callee
f10-11	fa0-1	Floating-point function parameter / return value	Caller
f12-17	fa2-7	Floating-point function parameter	Caller
f18-27	fs2-11	Floating-point save registers	Callee
f28-31	ft8-11	Floating-point temporary registers	Caller

In the above table, “Caller” means that the value of the register is not saved during the called process. “Callee” means that the register is saved during the called process.

1.3 Privileged modes

The standard RISC-V architecture supports 3 privileged modes: machine mode, supervisor mode and user mode, as shown in Table 1-3. The machine mode is a mode that must be implemented, while the other 2 modes are modes that can be implemented optionally. Fore details, please refer to the RISC-V Privileged Architecture.pdf which can be downloaded for fress on the RISC-V Foundation official website.

Table 1-3 RISC-V architecture privileged modes

Code	Name	Abbreviation
0b00	User mode	U
0b01	Supervisor mode	S
0b10	Reserved	Reserved
0b11	Machine mode	M

The Qingke V4 microprocessor supports 2 of the 3 privileged modes.

- Machine mode

  The machine mode has the highest privilege. In this mode, the program can access all the control and status registers (CSR), and can also access all the locks except those locked by the Physical Memory Protection (PMP). Physical address area. The microprocessor is in machine mode by default when powered on. After the program executes mret (machine mode return command) and returns, the MPP bit in the CSR mstatus register (machine mode status register) determines the subsequent mode. If MPP=0b00, the microprocessor exits the machine mode and enters the user mode. If MPP=0x11, the microprocessor is still in machine mode.
- User mode

  The user mode has the lowest privilege. In this mode, the program can only access the limited CSR

registers and the physical address area allowed by PMP. The microprocessor exits user mode and enters machine mode after an exception or interrupt occurs, to handle the exception/interrupt.

1.4 CSR registers

The RISC-V architecture defines some CSR registers to control and record the operating status of the microprocessor. These CSR registers use an internal dedicated 12-bit address encoding space that can be extended to 4096 registers. The CSR[11:10] bits are used to define the read/write permissions of the register. The 0b00, 0b01, 0b10 values indicate read/write, and the 0b11 value indicates read only. The CSR[9:8] bits are used to define the lowest privilege level for register access, please refer to Table 1-3 for the meaning of the values. In addition to the relevant CSR registers defined by the standard, the Qingke V4 microprocessor provides some custom CSR registers to control the enhanced function and record the status. The CSR registers implemented by the microprocessor are detailed in Chapter 8.

Chapter 2 Exception

The Exception mechanism is a mechanism to intercept and handle unusual conditions occurring at run time. The Qingke V4 microprocessor is equipped with an exception response system, which can handle up to 256 exceptions including interrupts. When an exception or interrupt occurs, the microprocessor can respond quickly and handle the exception and interrupt events.

2.1 Exception type

Regardless of an exception/interrupt occurs, the hardware behavior of the microprocessor is consistent. The microprocessor suspends the current program, turns to the exception/interrupt handler, and returns to the previously suspended program after processing is complete. In a broad sense, an interrupt is also an exception. Whether an interrupt or an exception is currently occurring can be known through the machine mode exception cause register (mcause). The mcause[31] is the interrupt field, which is used to indicate whether the unusual condition is an interrupt or an exception. If mcause[31]=1, the unusual condition is an interruption. If mcause[31]=0, the unusual condition is an exception. The mcause[30:0] are the exception code bits, used to indicate the specific cause of the exception or interrupt number, as shown in the following table.

Table 2-1 V4 microprocessor exception codes

Interrupt	Exception Code	Sync/Async	Cause of the unusual condition
1	0-1	-	Reserved
1	2	Exact async	NMI interrupt
1	3-11	-	Reserved
1	12	Exact async	SysTick interrupt
1	13	-	Reserved
1	14	Sync	Software interrupt
1	15	-	Reserved
1	16-255	Exact async	External interrupts 16-255
0	0	Sync	Instruction address not aligned
0	1	Sync	Get instruction access error
0	2	Sync	Illegal instruction
0	3	Sync	Breakpoint
0	4	Sync	Load instruction access memory address not aligned
0	5	Inexact async	Load instruction access memory error
0	6	Sync	Store/AMO instruction access memory address not aligned
0	7	Inexact async	Store/AMO instruction access memory error
0	8	Sync	Environment calls in user mode
0	11	Sync	Environment calls in machine mode

In the above table, “Sync” means that an instruction to be executed can be located exactly, such as an ebreak or ecall instruction, and an entry exception will be triggered every time the instruction is executed. “Async” means that an instruction cannot be located exactly, and the instruction PC value may be different every time an exception occurs. “Exact async” means that the boundary of an instruction can be located exactly after an exception occurs, that is, the state after an instruction is executed, such as an external interrupt. “Inexact

async" means that the boundary of an instruction cannot be located exactly, and it may be a state where an instruction is interrupted after half of the execution, such as an access memory error. It takes time to access the memory. When the microprocessor accesses the memory, it generally does not wait for the end of the access, but continues to execute the instruction. When the memory access error exception occurs again, the microprocessor has already executed the subsequent instructions and cannot locate exactly.

2.2 Exception entry

If for some reason, an exception or interrupt is triggered when the program is running normally. In this case, the hardware behavior of the microprocessor can be summarized as follows:

(1) Pauses the current program flow and turns to execute an exception or interrupt handler.

The entry base address and addressing mode of the exception or interrupt function are defined by the exception entry base address register (mtvec). The mtvec[31:2] bits define the base address of the exception or interrupt function. The mtvec[1:0] bits define the addressing mode of the processing function, and mtvec[0] defines the entry mode of exceptions and interrupts. When mtvec[0]=0, all exceptions and interrupts use a unified entry, in this case, the program turns to the base address defined by mtvec[31:2] and executes it when an exception or interrupt occurs. The specific exception type or interrupt can be known by querying the mcause register and they are processed separately. When mtvec[0]=1, the exception and interrupt use the vector table mode, that is, each exception and interrupt are numbered, and the address offset is based on the interrupt number* 4. The program turns to the base address defined by mtvec[31:2] + interrupt number * 4 and executes it when an exception or interrupt occurs. In vector mode, mtvec[1] defines the identification mode of the vector table. When mtvec[1]=0, the vector table stores an instruction to jump to an exception or interrupt handler, or it can be another instruction. When mtvec[1]=1, the absolute address of the exception handling function is stored in the vector table.

(2) Update CSR registers

The microprocessor automatically updates the relevant CSR registers, including the machine mode exception cause register (mcause), the machine mode exception pointer register (mepc), the machine mode exception value register (mtval), and the machine mode status register (mstatus) when it enters an exception or interrupt.

- Update mcause

As described above, the value reflects the current exception type or interrupt number after the microprocessor enters an exception or interrupt. The value of this register can be read by software to know the cause of the exception or determine the source of the interrupt, see Table 2-1 for details.

- Update mepc

The standard-defined return address of the microprocessor after it exits an exception or interrupt is stored in mepc. Therefore, after an exception or interrupt occurs, the hardware automatically updates the mepc value to the PC value of the current instruction when an exception is encountered, or the PC value of the next pre-executed instruction before the interrupt. After the exception or interrupt processing is over, the microprocessor uses its saved value as the return address, returns to the interrupted location and continues execution.

Note:

1. The mepc register can be read and written. The value of this register can be modified by software to return to the running PC pointer position after modification.

2. When an interrupt occurs, the value of mepc is updated to the PC value of the next unexecuted instruction that is interrupted if mcause[31]=1.
   When an exception occurs, the value of mepc is updated to the PC value of the current unusual instruction if mcause[31]=0. Therefore, when the exception returns at this time, if the value of mepc is returned directly, the program continues to execute the instruction that generated this exception before, and continues to enter the exception. Usually, we can modify the value of mepc to the value of the next unexecuted instruction and then return it after the exception is handled. For example, if an exception occurs due to ecall/ebreak, we only need to modify the value of mepc to mepc+4 (c.ebreak is mepc+2) through software and then return it after the exception is handled, since ecall/ebreak (c.ebreak is 2 bytes) is a 4-byte instruction.

- Update mtval

The hardware automatically updates the value of mtval which cause the exception when the microprocessor enters an exception or an interrupt. This value is generally:

1. When an exception caused by memory access occurs, the hardware stores the address accessed by the memory at the time of the exception into mtval.
2. When an exception caused by an illegal instruction occurs, the hardware stores the instruction code of this instruction into mtval.
3. When an exception caused by a hardware breakpoint occurs, the hardware stores the PC value at the breakpoint into mtval.
4. When other exceptions such as exceptions caused by ebreak and ecall instructions occur, the hardware sets the value of mtval to 0.
5. When the microprocessor enters the interrupt, the hardware sets the value of mtval to 0.

- Update mstatus

The hardware updates some bits in mstatus when the microprocessor enters an exception/interrupt.

1. MPIE is updated to the MIE value before the microprocessor enters an exception/interrupt. And it is used to restore the MIE at the end of the exception/interrupt.
2. MPP is updated to the privileged mode before the microprocessor enters an exception/interrupt. And it is used to restore the previous privileged mode at the end of the exception/interrupt.
3. The Qingke V4 microprocessor supports interrupt nested in machine mode. MIE is not cleared after the microprocessor enters an exception/interrupt.

(3) Update the privileged mode of the microprocessor

The microprocessor privileged mode is updated to machine mode after an exception/interrupt occurs.

2.3 Exception handler

The microprocessor executes the program from the address and mode defined by the mtvec register after it enters an exception/interrupt. When the unified entry is used, the microprocessor takes a jump instruction from the base address defined by mtvec[31:2] according to the value of mtvec[1], or obtains the entry address of the exception/interrupt handle function and executes it instead. In this case, it can be judged whether an exception/interrupt is caused by the value of mcause[31] in the exception/interrupt handle function, and the type and cause of the exception or the interrupt number can be judged by the exception code, and the corresponding processing is performed.

When the offset is based on the (base address + interrupt number * 4), the hardware automatically jumps to the vector table according to the interrupt number to obtain the entry address of the exception/interrupt function, and executes.

2.4 Exception exit

It needs to exit from the service routine at the end of the exception/interrupt handler. The microprocessor exits user mode and enters machine mode after it enters the exception/interrupt, and the processing of the exception/interrupt is also completed in machine mode. When it needs to exit the exception/interrupt, the mret instruction needs to be used to return. At this point, the microprocessor hardware will automatically perform the following actions:

- Restore the PC pointer to the value of the mepc CSR register, that is, the execution starts from the instruction address saved by mepc. It is necessary to pay attention to the mepc offset operation at the end of the exception handler.
- Update the mstatus CSR register. MIE is restored to MPIE. MPP is used to restore the previous privileged mode.

The entire exception response process is shown in Figure 2-1:

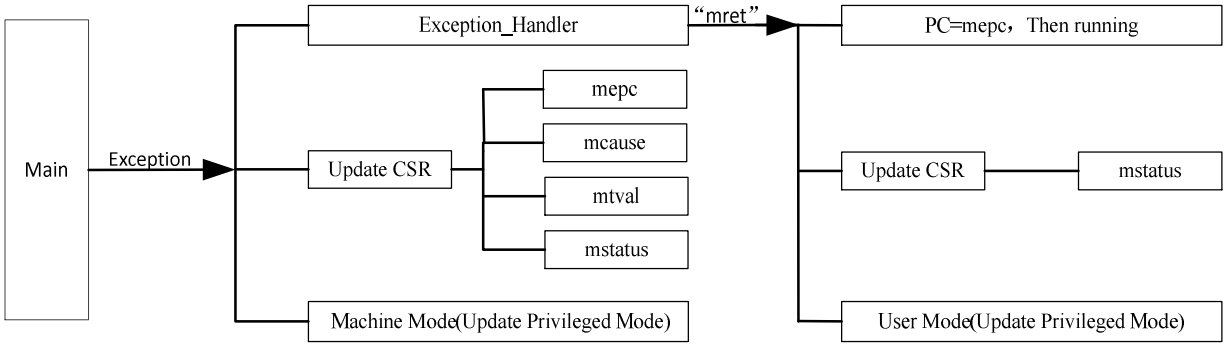

Figure 2-1 Exception response process

Chapter 3 PFIC and interrupt control

The Qingke V4 microprocessor is equipped with a Programmable Fast Interrupt Controller (PFIC) which can manage up to 256 interrupts including exceptions. The first 16 of the 256 interrupts are fixed as internal interrupts of the microprocessor, and the others are external interrupts. So it can extend up to 240 external interrupts.

Main features:

- 240 external interrupts. Each interrupt request has independent trigger and mask control bits, and dedicated status bits.
- Programmable multi-level interrupt nested, the maximum nest depth is 8 levels.
- Fast interrupt entry and exit mechanism, hardware automatic push and recovery, the maximum hardware push depth is 3 levels, with no instruction overhead required.
- Vector Table Free (VTF) interrupt response mechanism, 4 programmable channels that can direct access interrupt vector address

Note: For different types of microprocessors, the interrupt controllers support different maximum nest depths and hardware push depths. For details, please refer to Table 1-1.

The vector table for interrupts and exceptions is shown in Table 3-1.

Table 3-1Exception and interrupt vector table

No.	Priority	Type	Name	Description
0	-	-	-	-
1	-	-	-	-
2	-5	Fixed	NMI	Non-maskable interrupt
3	-4	Fixed	EXC	Exception interrupt
4	-	-	-	-
5	-3	Fixed	ECALL-M	Machine mode callback interrupt
6-7	-	-	-	-
8	-2	Fixed	ECALL-U	User mode callback interrupt
9	-1	Fixed	BREAKPOINT	Breakpoint callback interrupt
10-11	-	-	-	-
12	0	Programmable	SysTick	System timer interrupt
13	-	-	-	-
14	1	Programmable	SWI	Software interrupt
15	-	-	-	-
16-255	2-241	Programmable	External Interrupt	External interrupts 16-255

3.1 PFIC register group

Table 3-2 PFIC registers

Name	Access address	Access	Description	Reset value
PFIC_ISRx	0xE000E000 -0xE000E01C	RO	Interrupt status register x	0x00000000

PFIC_IPRx	0xE000E020 -0xE000E03C	RO	Interrupt pending register x	0x00000000
PFIC_ITHRESDR	0xE000E040	RW	Interrupt priority threshold configure register	0x00000000
PFIC_CFGR	0xE000E048	RW	Interrupt configure register	0x00000000
PFIC_GISR	0xE000E04C	RO	Interrupt global status register	0x00000000
PFIC_VTFIDR	0xE000E050	RW	VTF ID configure register	0x00000000
PFIC_VTFADDRRx	0xE000E060 -0xE000E06C	RW	VTF interrupt x offset address register	0x00000000
PFIC_IENRx	0xE000E100 -0xE000E11C	WO	Interrupt enable register x	0x00000000
PFIC_IRERx	0xE000E180 -0xE000E19C	WO	Interrupt reset enable register x	0x00000000
PFIC_IPSRx	0xE000E200 -0xE000E21C	WO	Interrupt pending set register x	0x00000000
PFIC_IPRRx	0xE000E280 -0xE000E29C	WO	Interrupt pending reset register x	0x00000000
PFIC_IACTRx	0xE000E300 -0xE000E31C	RO	Interrupt activate register x	0x00000000
PFIC_IPRIORx	0xE000E400 -0xE000E43C	RW	Interrupt priority configure register	0x00000000
PFIC_SCTLR	0xE000ED10	RW	System control register	0x00000000

Note: 1. The default value of the PFIC_ISR0 register is 0xC, so NMI and exceptions are always enabled by default.

2. ECALL-M, ECALL-U and BREAKPOINT are EXC conditions, and the status is indicated by the status bit (bit3) of EXC.

3. NMI and EXC supports interrupt pending reset and set operations, but do not support interrupt enable reset or set operations.

4. ECALL-M, ECALL-U and BREAKPOINT do not support interrupt pending reset/set or interrupt enable reset/set operations.

Register descriptions:

Interrupt status register and interrupt pending register (PFIC_ISR<0-7>/PFIC_IPR<0-7>)

Name	Access address	Access	Description	Reset value
PFIC_ISR0	0xE000E000	RO	Interrupts 0-31 status register 32 status bits [n], to indicate the #n interrupt enable status *Note: NMI and EXC are enabled by default.*	0x0000000C
PFIC_ISR1	0xE000E004	RO	Interrupts 32-63 status register 32 status bits	0x00000000
…	…	…	…	…
PFIC_ISR7	0xE000E01C	RO	Interrupts 224-255 status register 32 status bits	0x00000000

PFIC_IPR0	0xE000E020	RO	Interrupts 0-31 pending register 32 status bits [n], to indicate the #n interrupt pending status	0x00000000
PFIC_IPR1	0xE000E024	RO	Interrupts 32-63 pending register 32 status bits	0x00000000
…	…	…	…	…
PFIC_IPR7	0xE000E03C	RO	Interrupts 244-255 pending register 32 status bits	0x00000000

These 2 register groups are used to indicate the interrupt enable status and the interrupt pending status.

Interrupt enable register and interrupt reset register (PFIC_IENR<0-7>/PFIC_IRER<0-7>)

Name	Access address	Access	Description	Reset value
PFIC_IENR0	0xE000E100	WO	Interrupts 0-31 enable register 32 set bits [n], to enable the interrupt #n. *Note: NMI and EXC are enabled by default.*	0x00000000
PFIC_IENR1	0xE000E104	WO	Interrupts 32-63 enable register 32 set bits	0x00000000
…	…	…	…	…
PFIC_IENR7	0xE000E11C	WO	Interrupts 224-255 enable register 32 set bits	0x00000000
-	-	-	-	-
PFIC_IRER0	0xE000E180	WO	Interrupts 0-31 reset register 32 reset bits [n], to reset the interrupt #n. *Note: NMI and EXC do not support it.*	0x00000000
PFIC_IRER1	0xE000E184	WO	Interrupts 32-63 reset register 32 reset bits	0x00000000
…	…	…	…	…
PFIC_IRER7	0xE000E19C	WO	Interrupts 244-255 reset register 32 reset bits	0x00000000

These 2 register groups are used to enable and reset the interrupts.

Interrupt pending set register and interrupt pending reset register (PFIC_IPSR<0-7>/PFIC_IPRR<0-7>)

Name	Access address	Access	Description	Reset value
PFIC_IPSR0	0xE000E200	WO	Interrupts 0-31 pending set register 32 set bits [n], to set the interrupt #n pending *Note: ECALL-M, ECALL-U and BREAKPOINT do not support it.*	0x00000000

PFIC_IPSR1	0xE000E204	WO	Interrupts 32-63 pending set register 32 set bits	0x00000000
…	…	…	…	…
PFIC_IPSR7	0xE000E21C	WO	Interrupts 224-255 pending set register 32 set bits	0x00000000
-	-	-	-	-
PFIC_IPRR0	0xE000E280	WO	Interrupts 0-31 pending reset register 32 reset bits [n], to reset the interrupt #n pending *Note: ECALL-M, ECALL-U and BREAKPOINT do not support it.*	0x00000000
PFIC_IPRR1	0xE000E284	WO	Interrupts 32-63 pending reset register 32 reset bits	0x00000000
…	…	…	…	…
PFIC_IPRR7	0xE000E29C	WO	Interrupts 244-255 pending reset register 32 reset bits	0x00000000

When the microprocessor enables an interrupt, it can be directly set by the interrupt pending register to trigger the interrupt entry. The interrupt pending reset register is used to reset the pending.

Interrupt activate register (PFIC_IACTR<0-7>)

Name	Access address	Access	Description	Reset value
PFIC_IACTR0	0xE000E300	RO	Interrupts 0-31 activate register 32 status bits [n], to indicate that the interrupt #n is activated.	0x00000000
PFIC_IACTR1	0xE000E304	RO	Interrupts 32-63 activate register 32 status bits	0x00000000
…	…	…	…	…
PFIC_IACTR7	0xE000E31C	RO	Interrupts 224-255 activate register 32 status bits	0x00000000

Each interrupt has an active status bit. This bit is set when the microprocessor enters the interrupt, and it is reset by hardware after the mret is returned.

Interrupt priority register and interrupt priority threshold register (PFIC_IPRIOR<0-7>/PFIC_ITHRESDR)

Name	Access address	Access	Description	Reset value
PFIC_IPRIOR0	0xE000E400	RW	Interrupt 0 priority set: [7:4]: Priority control bits No preempt bit if configured as no nesting.	0x00

			The bit7 is the preempt bit if configured as 2-level nesting. The bit7 and bit6 are preempt bits if configured as 4-level nesting. The bit7 to bit5 are preempt bits if configured as 8-level nesting. [3:0]: Reserved, always 0. *Note: The smaller the priority value, the higher the priority. If the interrupts with the same preempt priority are pending at the same time, the interrupt with the higher priority is executed first.*	
PFIC_IPRIOR1	0xE000E401	RW	Interrupt 1 priority set The function is the same as that of PFIC_IPRIOR0.	0x00
PFIC_IPRIOR2	0xE000E402	RW	Interrupt 2 priority set The function is the same as that of PFIC_IPRIOR0.	
…	…	…	…	…
PFIC_IPRIOR254	0xE000E4FE	RW	Interrupt 254 priority set The function is the same as that of PFIC_IPRIOR0	0x00
PFIC_IPRIOR255	0xE000E4FF	RW	Interrupt 255 priority set The function is the same as that of PFIC_IPRIOR0	0x00
-	-	-	-	-
PFIC_ITHRESDR	0xE000E040	RW	Interrupt priority threshold set [31:8]: Reserved, always 0 [7:4]: Priority threshold [3:0] : Reserved, always 0 Note: For an interrupt whose priority value is not less than the threshold, the interrupt service function is not executed when pending. When the value of this register is 0, the threshold register is invalid.	0x00

Interrupt configure register (PFIC_CFGR)

Name	Access address	Access	Description	Reset value
PFIC_CFGR	0xE000E048	RW	Interrupt configure register	0x00000000

Bit definitions:

Bit	Name	Access	Description	Reset value
[31:16]	KEYCODE	WO	Correspond to different target control bits. Only can be modified when the corresponding security access identification data is written. The read value is always 0. KEY1 = 0xFA05. KEY2 = 0xBCAF. KEY3 = 0xBEEF.	0
[15:8]	Reserved	RO	Reserved	0
7	SYSRESET	WO	System reset (KEY3 is written synchronously). Automatically reset. Writing 1 is valid, while writing 0 is invalid. *Note: The functions of it is the same as that of the SYSRESET bit in the PFIC_SCTLR register.*	0
[6:0]	Reserved	RO	Reserved	0

This register of a V4 microprocessor is mainly used for compatibility.

Interrupt global status register (PFIC_GISR)

Name	Access address	Access	Description	Reset value
PFIC_CFGR	0xE000E04C	RW	Interrupt global status register	0x00000000

Bit definitions:

Bit	Name	Access	Description	Reset value
[31:10]	Reserved	RO	Reserved	0
9	GPENDSTA	RO	Whether an interrupt is pending currently: 1: Yes. 0: No.	0
8	GACTSTA	RO	Whether an interrupt is activated currently: 1: Yes. 0: No.	0
[7:0]	NESTSTA	RO	Current interrupt nesting status. Currently supports up to 8 levels of nesting. The maximum hardware push depth is 3 levels. If the nesting depth is set to be greater than 3 levels, the lower three-level interrupts should be configured as hardware push, and other interrupts with high priority should be configured as software push. 0xFF: Level 8 interrupt. 0x7F: Level 7 interrupt. 0x3F: Level 6 inerrupt. 0x1F: Level 5 interrupt. 0x0F: Level 4 interrupt. 0x07: Level 3 interrupt. 0x03: Level 2 interrupt.	0

			0x01: Level 1 interrupt. 0x00: No interrupt. Others: Impossible.	

VTF ID register and VTF address register (PFIC_VTFIDR/PFIC_VTFADDRR<0-3>)

Name	Access address	Access	Description	Reset value
PFIC_VTFIDR	0xE000E050	RW	[31:24]: Numbering of VTF interrupt 3 [23:16]: Numbering of VTF interrupt 2 [15:8]: Numbering of VTF interrupt 1 [7:0]: Numbering of VTF interrupt 0	0x00000000
-	-	-	-	-
PFIC_VTFADDRR0	0xE000E060	RW	[31:1]: Address of VTF interrupt 0, 2-byte aligned [0]: 1: Enable VTF interrupt 0 channel 0: Disable	0x00000000
PFIC_VTFADDRR1	0xE000E064	RW	[31:1]: Address of VTF interrupt 1, 2-byte aligned [0]: 1: Enable VTF interrupt 1 channel 0: Disable	0x00000000
PFIC_VTFADDRR2	0xE000E068	RW	[31:1]: Address of VTF interrupt 2, 2-byte aligned [0]： 1: Enable VTF interrupt 2 channel 0: Disable	0x00000000
PFIC_VTFADDRR3	0xE000E06C	RW	[31:1]: Address of VTF interrupt 3, 2-byte aligned [0]： 1: Enable VTF interrupt 3 channel 0: Disable	0x00000000

System control register (PFIC_SCTLR)

Name	Access address	Access	Description	Reset value
PFIC_SCTLR	0xE000ED10	RW	System control register	0x00000000

Bit definitions:

Bit	Name	Access	Description	Reset value
31	SYSRESET	WO	System reset. Reset automatically. Writing 1 is valid, while writing 0 is invalid. The function is the same as that of PFIC_CFGR	0
[30:6]	Reserved	RO	Reserved	0
5	SETEVENT	WO	Set event. Events that can wake up WFE.	0
4	SEVONPEND	RW	When an event or interrupt pending state	0

			occurs, the system can be woken up by executing the WFE instruction. If the WFE instruction is not executed, the system will be woken up immediately after the next execution of the instruction. 1: The enabled events and all interrupts (including disabled interrupts) can wake up the system. 0: Only the enabled events and enabled interrupts can wake up the system.	
3	WFITOWFE	RW	WFI instruction executed as WFE 1: The following WFI instruction executed as the WFE instruction. 0: No effect.	0
2	SLEEPDEEP	RW	Control the system low-power mode: 1: Deepsleep 0: Sleep	0
1	SLEEPONEXIT	RW	Control the system state after leaving the interrupt service program: 1: The system enters low-power mode. 0: The system enters the main program.	0
0	Reserved	RO	Reserved	0

3.2 Interrupt relevant CSR register

In addition, the following CSR registers greatly affect the handling of interrupts:

Interrupt system control register (INTSYSCR)

Name	CSR address	Access	Description	Reset value
INTSYSCR	0x804	MRW	System control register	0x00000000

Bit definitions:

Bit	Name	Access	Description	Reset value
[31:16]	Reserved	MRO	Reserved	0
[15:8]	PMTSTA	MRO	Preempt bit status indicator: 0x00: No preempt bit in the priority configure bit, no interrupt nested. 0x80: The highest bit in the priority configure bit is the preempt bit. 2-level interrupt nested. 0xC0: The higher 2 bits in the priority configure bit are the preempt bits. 4-level interrupt nested. 0xE0: The higher 3 bits in the priority configure bit are the preempt bits. 8-level interrupt nested.	0
[7:6]	Reserved	MRO	Reserved	0

5	GIHWSTKNEN	MRW1	Global interrupt and hardware push not enable *Note: This bit is often used in real-time operating systems. Setting this bit can disable global interrupt and hardware push and pop when an interrupt switches contexts. The hardware automatically resets this bit when the context switch is completed and the interrupt returns after execution.*	0
4	HWSTKOVEN	MRW	Interrupt enable after hardware push overflows: 0: Global interrupt disabled after hardware push overflows. 1: Interrupt can still be executed after hardware push overflows. *Note: The hardware push depth is 3 levels. When the configured nesting level is greater than 3 levels, it is needed to configure the 3-level interrupts with low priority as hardware push and configure the interrupts with high priority as software push if this bit is set to 1.*	0
[3:2]	PMTCFG	MRW	Interrupt nested depth configure: 0b00: No nested, 0 preempt bit. 0b01: 2 levels nested, 1 preempt bit. 0b10: 4 levels nested, 2 preempt bits. 0b11: 8 levels nested, 3 preempt bits.	0
1	INESTEN	MRW	Interrupt nesting enable: 0: Interrupt nesting disabled. 1: Interrupt nesting enabled.	0
0	HWSTKEN	MRW	Hardware push enable: 0: Hardware push disabled. 1: Hardware push enabled.	0

Machine trap-vector base-address register (mtvec)

Name	CSR address	Access	Description	Reset value
mtvec	0x305	MRW	Machine trap-vector base-address register	0x00000000

Bit definitions:

Bit	Name	Access	Description	Reset value
[31:2]	BASEADDR[31:2]	MRW	Interrupt vector base address	0
1	MODE1	MRW	Interrupt vector identification mode: 0: Identified by jump instruction, limited	0

			range, support non-jump instruction. 1: Identified by absolute address, full range, but must jump.	
0	MODE0	MRW	Interrupt/exception entry address mode selection: 0: Unified entry address. 1: Address offset according to the interrupt number *4.	0

For an MCU with the V4 microprocessor, MODE[1:0]=0b11 is configured by default in the startup file, that is, the vector uses the absolute address of the interrupt function, and the entry of exception/interrupt is offset according to the interrupt number *4.

3.3 Interrupt nested

To support nested interrupts, the interrupt system control register (INTSYSCR, CSR address: 0x804) and the interrupt priority register (PFIC_IPRIOR) are needed. Enable the interrupt nesting function and configure the nested depth via the interrupt system control register (for an MCU with the V4 microprocessor, configure it in the startup file), and configure the priority of the corresponding interrupt. The smaller the priority value, the higher the priority. The smaller the preempt bit value, the higher the preempt priority. If the interrupts with the same preempt priority are pending at the same time, the microprocessor will first respond to the interrupt with the smaller priority value (higher priority).

3.4 Hardware push

The microprocessor stops the current program flow when an exception/interrupt occurs, and turns to execute the exception/interrupt handler. In this case, the field of the current program flow needs to be saved. After the exception/interrupt returns, it is required to continue the stopped program flow after the field is restored. For the V4 microprocessor, the "field" here refers to all the Caller Saved registers in Table 1-2.

The V4 microprocessor supports hardware to automatically save 16 integer Caller Saved registers to the internal stack area in a single cycle. The internal stack area is invisible to users. When an exception/interrupt returns, the hardware automatically restores data from the internal stack area to 16 integer registers in a single cycle. The hardware push stack supports nesting, and the maximum nesting depth is 3 levels. After the hardware push stack overflows, the "field" is saved to the user stack area if the interrupt with higher priority still can be executed.

Set the hardware push stack overflow by setting bit4 in the interrupt system control register (INTSYSCR) to disable the interrupt response when the allowable interrupt nesting depth is greater than the hardware push stack depth, that is, the maximum nesting depth is 3 levels, and the hardware push stack is used. If the interrupt can continue to execute after the hardware push stack overflows, the priority of the interrupt function with hardware push stack needs to be set to the lowest level 3.

The schematic diagram of the microprocessor push stack is shown in the following figure:

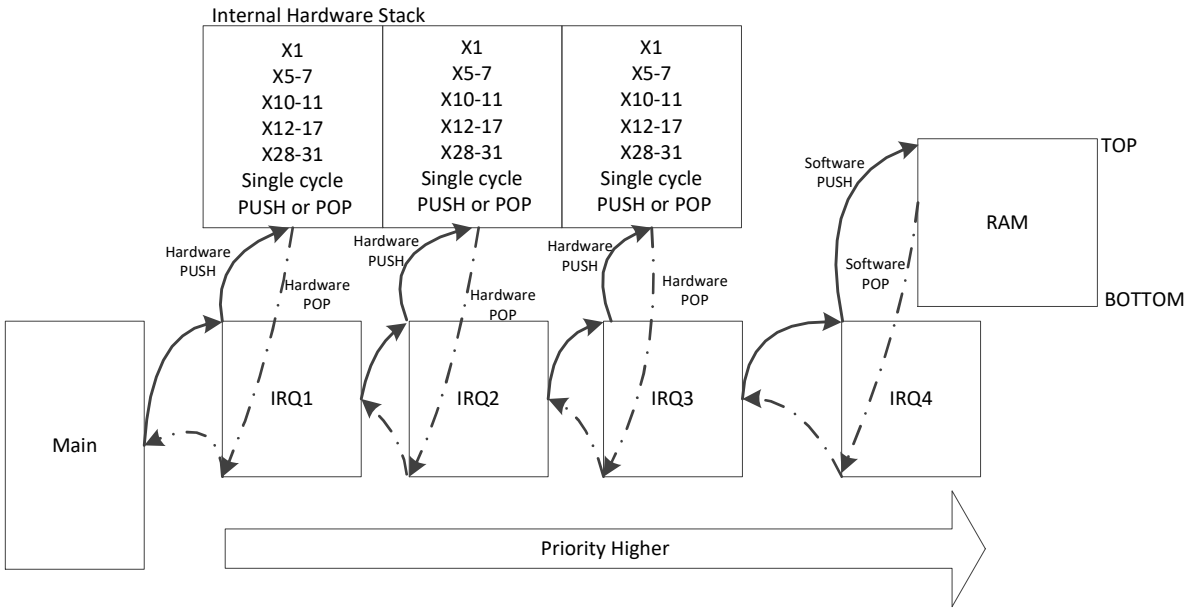

Figure 3-1 Hardware push stack diagram

Note: 1. For different models, the hardware push depths may be different.

2. The interrupt function with hardware push needs to be compiled by MRS or the tool chain provided by it, and it needs to be declared with __attribute__((interrupt("WCH-Interrupt-fast"))).

3. If hardware floating point is used and the interrupt function declared by hardware push stack is selected, the floating point registers are still saved and restored by the compiler, and they are all saved to the user stack area (RAM).

4. The interrupt function with software push is declared with __attribute__((interrupt())).

3.5 Vector table free interrupt

The programmable fast interrupt controller (PFIC) provides 4 VTF (Vector Table Free) interrupt channels that can access interrupt function entry directly without going through the table lookup process of the interrupt vector table.

To normally configure an interrupt function, write its interrupt number into a channel x of the VTF ID register (PFIC_VTFIDR), write its entry address into the corresponding VTF address register (VTFADDRRx) at the same time, and enable the VTF interrupt of this channel. Figure 3-2 shows the response process of PFIC to fast interrupt and VTF interrupt.

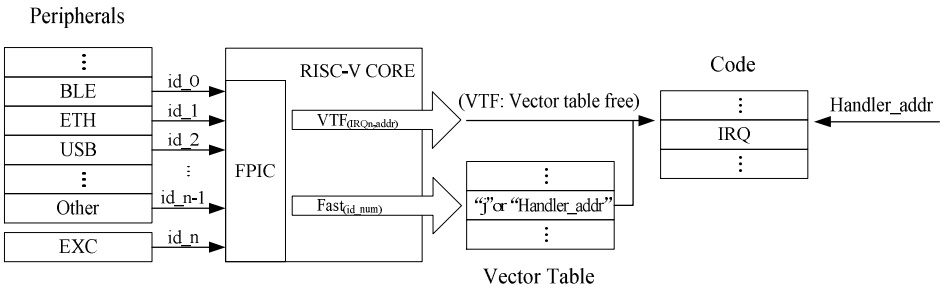

Figure 3-2 Diagram of the programmablr fast interrupt controller

Chapter 4 Physical memory protection (PMP)

To improve system security, the Qingke V4 microprocessor is equipped with a physical memory protection (PMP) module based on the RISC-V architecture standard. It supports access privilege management for up to 4 physical regions. The access privileges include read (R), write (W), and execute (X) attributes, and the minimum length of the protection region can be set to 4 bytes. The PMP module is always valid in user mode, and can be optionally valid in machine mode through the lock (L) attribute.

If the access violates the current privilege restrictions, an exception interrupt will be triggered. The PMP module includes 4 groups of 8-bit configure registers (a group of 32 bits) and 4 groups of address registers. All registers need to be accessed in machine mode using CSR instructions.

Note: The number of protection regions supported by different models of microprocessor PMP may be different, and the number supported by the pmpcfg<n> and pmpaddr<i> registers are also different, see Table 1-1 for details.

4.1 PMP register group

The CSR registers supported by the V4 microprocessor PMP are listed in Table 4-1.

Table 4-1 Register groups of PMP

Name	CSR address	Access	Description	Reset value
pmpcfg0	0x3A0	MRW	PMP configure register 0	0x00000000
pmpaddr0	0x3B0	MRW	PMP address register 0	0x00000000
pmpaddr1	0x3B1	MRW	PMP address register 1	0x00000000
pmpaddr2	0x3B2	MRW	PMP address register 2	0x00000000
pmpaddr3	0x3B3	MRW	PMP address register 3	0x00000000

4.2 pmp<i>cfg register

The pmpcfg<n> is a PMP unit configure register. Each register provides 4 8-bit pmp<i>cfg fields to configure the 4 regions. The pmp<i>cfg represents the configuration value of the region i. The format is shown in Table 4-2.

Table 4-2 pmpcfg0 register

31 24	23 16	15 8	7 0	
pmp3cfg	pmp2cfg	pmp1cfg	pmp0cfg	pmpcfg0

The pmp<i>cfg is used to configure the region I, and the bit definitions are described in Table 4-3.

Table 4-3 pmp<i>cfg register

Bit	Name	Description
7	L	Lock enable. Can be unlocked in machine mode. 0: Unlock. 1: Related registers locked.
[6:5]	-	Reserved
[4:3]	A	Address alignment and protection region selection

		00: OFF (PMP disabled) 01: TOR (top of range) 10: NA4 (naturally aligned 4-byte region) 11: NAPOT ($2^{(G+2)}$ -byte region, G≥1)
2	X	Execute 0: No execute permission 1: Execute permission
1	W	Write 0: No write permission 1: Write permission
0	R	Read 0: No read permission 1: Read permission

4.3 pmpaddr<i> register

The pmpaddr<i> register is used to configure the address of the region i. It encodes the higher 32 bits of a 34-bit physical address for RV32, as shown in Table 4-4. The entire physical address space of the V4 microprocessor is 4G, so the higher 2 bits in this register are not used.

Table 4-4 pmpaddr<i> register

31	0
address[33:2]	

When NAPOT is selected, the low-order bits of the address register are also used to indicate the size of the current protection region, as shown in the table below, where "y" represents a bit of the register.

Table 4-5 Protection region range encoding in PMP configuration and address registers

pmpaddr	pmpcfg.A	Match base address and size
yyyy…yyyy	NA4	“yy…yyyy00” as base address, 4-byte protection region range
yyyy…yyy0	NAPOT	“yy…yyy000” as base address, 8-byte protection region range
yyyy…yy01	NAPOT	“yy…yy0000” as base address, 16-byte protection region range
yyyy…y011	NAPOT	“yy…y00000” as base address, 32-byte protection region range
…	…	…
yyy01…111	NAPOT	“y0…000000” as base address, 2^{31}-byte protection region range
yy011…111	NAPOT	2^{32}-byte protection region range

4.4 Protection mechanism

The X/W/R permission of pmp<i>cfg is used to set the protection permission of the region i. Violating the relevant permission will cause an exception.

(1) Attempting to fetch an instruction in the PMP region without execute permission will cause a fetch instruction access error exception (mcause=1).

(2) Attempting to write data into the PMP region without write permission will cause a store instruction access error exception (mcause=7).

(3) Attempting to read data from the PMP region without read permission will cause a load instruction access error exception (mcause=5).

The A bit in pmp<i>cfg is used to set the protection region range and address alignment of the region I, to perform memory protection for the region i (A_ADDR≤region<i><B_ADDR) (A_ADDR and B_ADDR are required to be 4-byte aligned).

(1) Select NA4 if B_ADDR – A_ADDR == 2^2.

(2) Select NAPOT if B_ADDR – A_ADDR == $2^{(G+2)}$, G≥1, and A_ADDR is $^{(G+2)}$ aligned.

(3) Otherwise select TOP.

Table 4-6 PMP address match

A	Name	Description
0b00	OFF	Null region
0b01	TOR	Top of range: For pmp<i>cfg, $pmpaddr_{i-1}$≤ region<i> <$pmpaddr_i$; $pmpaddr_{i-1}$ = A_ADDR >> 2; $pmpaddr_i$ = B_ADDR >> 2. Note: If the PMP region 0 is configured to TOR (i=0), the page boundary of the protection region is 0 address, that is 0 ≤ addr < pmpaddr0, which in the match range.
0b10	NA4	Naturally aligned 4-byte region: For pmp<i>cfg, 4 bytes with $pmpaddr_i$ as start address $pmpaddr_i$ = A_ADDR >> 2.
0b11	NAPOT	$2^{(G+2)}$ region, G≥1, and A_ADDR is $2^{(G+2)}$ aligned. $pmpaddr_i$ = （(A_ADDR |（$2^{(G+2)}$ -1）） & ~(1<<G+1)）>> 2.

The L bit in pmp<i>cfg is used to lock the PMP entry. The configuration register (pmp<i>cfg) and the address register (pmpaddr<i>) cannot be modified after locked. The pmpaddr<i-1> also cannot be modified if the A bit in pmp<i>cfg is set to TOR. The X/W/R permission defined by pmp<i>cfg also takes effect in machine mode after the L bit is set. While X/W/R is only valid in user mode after the L bit is reset. The L bit is reset only after system reset.

The Qingke V4 microprocessor supports protection of several regions. When the same operation matches several regions at the same time, the region with a smaller number is preferentially matched.

Chapter 5 System tick timer (SysTick)

The Qingke V4 microprocessor is equipped with a 64-bit up and down counter (SysTick). The clock source can be either the system clock or the system clock divided by 8. It provides time base for real-time operating systems, and provides timing and measurement time, etc. The timer involves 6 registers that are mapped to peripheral address space for SysTick control, as shown in Table 5-1.

Table 5-1 SysTick registers

Name	Access address	Description	Reset value
STK_CTLR	0xE000F000	SysTick control register	0x00000000
STK_SR	0xE000F004	SysTick status register	0x00000000
STK_CNTL	0xE000F008	SysTick counter low register	0x00000000
STK_CNTH	0xE000F00C	SysTick counter high register	0x00000000
STK_CMPLR	0xE000F010	SysTick count reload low register	0x00000000
STK_CMPHR	0xE000F014	SysTick count reload high register	0x00000000

SysTick control register (STK_CTLR)

Table 5-2 SysTick control register

Bit	Name	Access	Description	Reset value
31	SWIE	RW	Software interrupt trigger enable (SWI): 1: Software interrupt triggered. 0: Trigger disbled. It needs software to reset after it enters software interrupt, otherwise the interrupt is continuously triggered.	0
[30:6]	Reserved	RO	Reserved	0
5	INIT	W1	Counter initial value update: 1: Updated to 0 when upcounting, while updated to the comparison value when downcounting. 0: No effect.	0
4	MODE	RW	Count mode: 1: Down count. 0: Up count.	0
3	STRE	RW	Auto-reload count enable: 1: Start counting from zero again when counting up to the comparison value. Start counting from the comparison value again when counting down to zero. 0: Continue up/down counting.	0
2	STCLK	RW	Counter clock source selection: 1: HCLK selected as time base. 0: HCLK/8 selected as time base.	0
1	STIE	RW	Counter interrupt enable: 1: Counter interrupt enabled. 0: Counter interrupt disabled.	0

0	STE	RW	SysTick enable 1: SysTick enabled. 0: SysTick disabled, the counter stops counting.	0

SysTick status register (STK_SR)

Table 5-3 SysTick status register

Bit	Name	Access	Description	Reset value
[31:1]	Reserved	RO	Reserved	0
0	CNTIF	RW0	Count value compare flag Write 0 to reset. Writing 1 is invalid. 1: Up count to the comparison value. Down count to zero. 0: Not reach the comparison value.	0

SysTick counter low register (STK_CNTL)

Table 5-4 SysTick counter low register

Bit	Name	Access	Description	Reset value
[31:0]	CNTL	RW	Low-order 32 bits of the current counter value.	0

Note: The STK_CNTL register and the STK_CNTH register constitute a 64-bit system counter.

SysTick counter high register (STK_CNTH)

Table 5-5 SysTick counter high register

Bit	Name	Access	Description	Reset value
[31:0]	CNTH	RW	High-order 32 bits of the current counter value.	0

Note: The STK_CNTL register and the STK_CNTH register constitute a 64-bit system counter.

SysTick count reload low register (STK_CMPLR)

Table 5-6 SysTick comparison value low register

Bit	Name	Access	Description	Reset value
[31:0]	CMPL	RW	Set the low-order 32 bits of the reload counter value	0

Note: The STK_CMPLR register and the STK_CMPHR register constitute a 64-bit counter comparison value.

SysTick count reload high register (STK_CMPHR)

Table 5-7 SysTick comparison value high register

Bit	Name	Access	Description	Reset value
[31:0]	CMPH	RW	Set the high-order 32 bits of the reload counter value	0

Note: The STK_CMPLR register and the STK_CMPHR register constitute a 64-bit counter comparison value.

Chapter 6 Processor low power setting

The Qingke V4 microprocessor supports entering the sleep state through the WFI (Wait For Interrupt) instruction to achieve lower static power consumption. With the PFIC system control register (PFIC_SCTLR), it can implement various sleep modes and WFE instructions.

6.1 Enter sleep

The Qingke V4 microprocessor can enter sleep in two ways, including Wait for Interrupt (WFI) and Wait for Event (WFE). WFI means that it waits for an interrupt to wake up after the microprocessor enters sleep, and then enters the interrupt to execute after waking up. WFE means that it waits for an event to wake up after the microprocessor enters sleep, and then resumes the program flow that was stopped before after waking up.

The standard RISC-V supports WFI. The microprocessor can directly execute the WFI command to enter sleep. The WFITOWFE bit in the system control register PFIC_SCTLR is used to control the following WFI instructions to be treated as WFE, to enter sleep through WFE.

Sleep depth controlled by the SLEEPDEEP bit in PFIC_SCTLR:

- The microprocessor enters sleep when the SLEEPDEEP bit in the PFIC_SCTLR register is reset. The internal unit clocks except SysTick and some wake-up logic can be disabled.
- The microprocessor enters deep-sleep when the SLEEPDEEP bit in the PFIC_SCTLR register is set. All unit clocks can be disabled.

The microprocessor cannot enter either sleep mode when it is in debug mode.

6.2 Wake up from sleep

The Qingke V4 microprocessor can wake up in the following ways when it enters sleep due to WFI and WFE.

- When the microprocessor enters sleep through WFI,
  (1) The microprocessor can be woken up by the interrupt source responded by the interrupt controller. It first executes the interrupt function after it wakes up.
  (2) The debug request can wake up the microprocessor when it is in sleep mode. While the debug request cannot wake up the microprocessor when it is in deep-sleep mode.

- When the microprocessor enters sleep through WFE,
  (1) The microprocessor can be woken up by the internal/external event. In this case, it continues to execute the program after it wakes up, with no need to configure the interrupt controller.
  (2) The microprocessor can be woken up when an interrupt source is enabled and an interrupt is generated. It first executes the interrupt function after it wakes up.
  (3) The microprocessor can be woken up when the SEVONPEND bit in PFIC_SCTLR and the interrupt controller does not enable the interrupt but a new interrupt pending signal is generated (the previously generated pending signal does not take effect). The corresponding interrupt pending flag needs to be reset manually after it wakes up.
  (4) The microprocessor in sleep mode can be woken up by the debug request. While it cannot be woken up by the debug request when it is in deep-sleep mode.

In addition, the SLEEPONEXIT bit in PFIC_SCTLR can be configured to control the status of the microprocessor after it wakes up.

- Set SLEEPONEXIT. The last level trap-return instruction (mret) will trigger the WFI mode sleep.
- Reset SLEEPONEXIT, which takes no effect.

Various MCU products equipped with V4 microprocessors can adopt different sleep modes according to different configurations of PFIC_SCTLR, disable different peripherals and clocks, execute different power management strategies and wake-up methods, and implement various low-power modes.

Chapter 7 Debug support

The Qingke V4 microprocessor contains a hardware debug module, which supports complex debug operations. The debug module can halt and resume the operation of the microprocessor. When the microprocessor is halted, the debug module can access the microprocessor's GPRs, CSRs, memory, external devices, etc. through abstract commands, program buffer deployment instructions, and so on.

The debug module follows the RISC-V External Debug Support Version 0.13.2. The detailed documentation can be downloaded from the Foundation website.

7.1 Debug module

The debug module inside the microprocessor can perform the debug operations issued by the debug host, including:

- Access registers through the debug interface
- Reset, halt and resume the microprocessor through the debug interface
- Read/Write to memory, instruction register and external deviced through the debug interface.
- Deploy several arbitrary instructions through the debug interface
- Set software breakpoints through the debug interface
- Set hardware breakpoints through the debug interface
- Support automatic execution of abstract commands
- Support single-step debug

Note: Hardware breakpoints are only supported by V4C and V4F microprocessors.

The internal registers of the debug module use 7-bit address coding, and the Qingke V4 microprocessor contains the following registers:

Table 7-1 List of debug module registers

Name	Access address	Description
data0	0x04	Data register 0, can be used to store data temporarily
data1	0x05	Data register 1, can be used to store data temporarily
dmcontrol	0x10	Debug module control register
dmstatus	0x11	Debug module status register
hartinfo	0x12	Hart information register
abstractcs	0x16	Abstract command control and status register
command	0x17	Abstract command register
abstractauto	0x18	Abstract command auto executed
progbuf0-7	0x20-0x27	Program buffer registers 0-7
haltsum0	0x40	Halt status register

The debug host can configure the dmcontrol register to halt, resume and reset the microprocessor. The debug module can also be triggered by the command register to generate abstract commands. The RISC-V standard defines 3 abstract command types: access register, fast access, and access memory. The Qingke V4 microprocessor supports 2 of them, but does not support fast access. Access to registers (GPRs, CSRs, FPRs) and sequential access to memories can be implemented through abstract commands.

The debug module internally contains 8 program buffer registers progbuf0-7. The debug host can cache

several instructions (which can be compressed instructions) in the buffer, and can choose to continue executing the instructions in the program buffer registers after executing the abstract command, or directly execute cached instructions. Note that the last instruction in progbufs needs to be an "ebreak" or "c.ebreak" instruction. Access to memory and peripherals can also be achieved through abstract commands and instructions cached in progbufs.

Register description:

Data register 0 (data0)

Table 7-2 data0 register definition

Bit	Name	Access	Description	Reset value
[31:0]	data0	RW	Data register 0, can be used to store data temporarily	0

Data register 1 (data1)

Table 7-3 data1 register definition

Bit	Name	Access	Description	Reset value
[31:0]	data1	RW	Data register 1, can be used to store data temporarily	0

Debug module control register (dmcontrol)

This register can be used to halt, reset and resume the microprocessor. The debug host writes data to the corresponding domain to halt (haltreq), reset (ndmreset), and resume (resumereq). Bit definitions are as shown in the table below.

Table 7-4 dmcontrol register definition

Bit	Name	Access	Description	Reset value
31	haltreq	WO	0: Clear the halt request 1: Send the halt request	0
30	resumereq	W1	0: Invalid 1: Resume the current microprocessor Note: Writing 1 is valid. Reset by hardware after the microprocessor resumes.	0
29	Reserve	RO	Reserved	0
28	ackhavereset	W1	0: Invalid 1: Reset the haverest status bit of the microprocessor	0
[27:2]	Reserve	RO	Reserved	0
1	ndmreset	RW	0: Clear reset 1: Reset the entire system except the debug modle	0
0	dmactive	RW	0: Reset the debug module 1: Debug mosule works normally	0

Debug module status register (dmstatus)

This register is used to indicate the debug module status, which is a read-only register. Bit definitions are shown in the tablr below:

Table 7-5 dmstatus register definition

Bit	Name	Access	Description	Reset value
[31:20]	Reserve	RO	Reserved	0
19	allhavereset	RO	0: Invalid 1: The microprocessor is reset.	0
18	anyhavereset	RO	0: Invalid 1: The microprocessor is reset.	0
17	allresumeack	RO	0: Invalid 1: The microprocessor is resumed.	0
16	anyresumeack	RO	0: Invalid 1: The microprocessor is resumed.	0
[15:14]	Reserve	RO	Reserved	0
13	allavail	RO	0: Invalid 1: The microprocessor is not available.	0
12	anyavail	RO	0: Invalid 1: The microprocessor is not available.	0
11	allrunning	RO	0: Invalid 1: The microprocessor is running.	0
10	anyrunning	RO	0: Invalid 1: The microprocessor is running.	0
9	allhalted	RO	0: Invalid 1: The microprocessor is halted.	0
8	anyhalted	RO	0: Invalid 1: The microprocessor is halted.	0
7	authenticated	RO	0: Authentication is required before the debug module is used. 1: The debug module has passed the authentication.	0x1
[6:4]	Reserve	RO	Reserved	0
[3:0]	version	RO	Version of architecture supported by debug system 2: V0.13	0x2

Hart information register (hartinfo)

This register is used to provide the hart information to the debug host, which is a read-only register. Bit definitions are as shown in the table below:

Table 7-6 hartinfo register definition

Bit	Name	Access	Description	Reset value
[31:24]	Reserve	RO	Reserved	0
[23:20]	nscratch	RO	The number of the supported dscratch registers	0x3
[19:17]	Reserve	RO	Reserved	0
16	dataaccess	RO	0: Data register mapped to CSR address. 1: Data register mapped to memory address.	0x1
[15:12]	datasize	RO	The number of the data registers	0x2
[11:0]	dataaddr	RO	First address of the data registers	x

Abstract command control and status register (abstractcs)

This register is used to indicate the execution of the abstract commands. The debug host can read this register to know whether the last abstract command has been executed and check whether an error is generated and the error type during the execution of the abstract commands. Bit definitions are as shown in the table below:

Table 7-7 abstractcs register definition

Bit	Name	Access	Description	Reset value
[31:29]	Reserve	RO	Reserved	0
[28:24]	progbufsize	RO	To indicate the number of the program buffer registers	0x8
[23:13]	Reserve	RO	Reserved	0
12	busy	RO	0: No abstract command to be executed. 1: It is executing the abstract command. Note: Reset by hardware after execution.	0
11	Reserve	RO	Reserved	0
[10:8]	cmder	RW	Abstract command error type 0: No error 1: Write to the command, abstractcs and abstractauto registers, or read/write to the data and progbuf registers when the abstract command is executed. 2: The current abstract command is not supported. 3: An error occurs when the abstract command is executed. 4: The microprocessor cannot execute the abstract command as it is not halted or as it is not available. 5: Bus error. 6: Parity bit error during communication. 7: Other error. Note: Write 1 to reset.	0
[7:4]	Reserve	RO	Reserved	0
[3:0]	datacount	RO	The number of the data registers.	0x2

Abstract command register (command)

The debug host can write different configuration values to the abstract command register, to access the GPRs, FPRs and CSRs inside the microprocessor and access memory.

When accessing the register, bit definitions of the command register are as shown in the table below:

Table 7-8 Bit definitions of the command register when accessing the register

Bit	Name	Access	Description	Reset value
[31:24]	cmdtype	WO	Abstract command type 0: Access register 1: Fast access (not supported) 2: Access memory	0

23	Reserve	WO	Reserved	0
[22:20]	aarsize	WO	Access register data bit width 0: 8 bits 1: 16 bits 2: 32 bits 3: 64 bits (not supported) 4: 128 bits (not supported) Note: Only 32-bit access is supported when accessing FPRs.	0
19	aarpostincrement	WO	0: No effect 1: Automatically increment the value of regno after accessing the register	0
18	postexec	WO	0: No effect 1: Execute the command in progbuf after executing the abstract command	0
17	transfer	WO	0: Not execute write instruction operations. 1: Execute write instruction operations.	0
16	write	WO	0: Copy data from the specified register to data0 1: Copy data from data0 to the specified register	0
[15:0]	regno	WO	Specify the access register 0x0000-0x0fff: CSRs 0x1000-0x101f: GPRs 0x1020-0x202f: FPRs	0

When accessing the memory, bit definitions of the command register are as shown in the table below:

Table 7-9 Bit definitions of the command register when accessing the memory

Bit	Name	Access	Description	Reset value
[31:24]	cmdtype	WO	Abstract command type 0: Access register 1: Fast access (not supported) 2: Access memory	0
23	aamvirtual	WO	0: Access physical address 1: Access virtual address	0
[22:20]	aamsize	WO	Access memory data bit width 0: 8 bits 1: 16 bits 2: 32 bits 3: 64 bits (not supported) 4: 128 bits (not supported)	0
19	aampostincrement	WO	0: No effect 1: After the memory is accessed successfully, the address stored in the data1 register is incremented according to the number of bytes corresponding to the bit width configured by aamsize aamsize=0, access by byte, add 1 to data1	0

			aamsize=1, access by half-word, add 2 to data1 aamsize=2, access by word, add 4 to data1	
18	postexec	WO	0: No effect 1: Execute the command in progbuf after the abstract command is executed.	0
17	Reserve	RO	Reserved	0
16	write	WO	0: Read data from the address specified by data1 to data0 1: Write the data in data0 to the address specified by data1	0
[15:14]	target-specific	WO	Definition of read and write Write: 0, 1: Directly write to the memory 2: Bit-OR the data in data0 with the data in the memory, and then write the result into the memory (only support access by word) 3: Bit-AND the data in data0 with the data in the memory, and then write the result into the memory (only support access by word) Read: 0, 1, 2, 3: Directly read the memory	0
[13:0]	Reserve	RO	Reserved	

Abstract command automatic executed register (abstractauto)

This register is used to configure the debug module. When reading/writing to progbufx and datax of the debug module, abstract commands can be executed again. Bit definitions are as shown in the table below:

Table 7-10 abstractauto register definition

Bit	Name	Access	Description	Reset value
[31:16]	autoexecprogbuf	RW	If a bit is set to 1, reading/writing to progbufx will cause the abstract command in the command register to be executed again. Note: The V4 microprocessor provides 8 progbuf registers, corresponding to bits [23:16].	0
[15:12]	Reserve	RO	Reserved	0
[11:0]	autoexecdata	RW	If a bit is set to 1, reading/writing to the datax register cause the abstract command in the command register to be executed again. Note: The V4 microprocessor provides 2 data registers, corresponding to bits [1:0].	0

Program buffer register (progbufx)

8 program buffer registers, used to store arbitrary instructions and deploy corresponding operations. Note that the last instruction executed should be "ebreak" or "c.ebreak".

Table 7-11 progbuf register definition

Bit	Name	Access	Description	Reset value
[31:0]	progbuf	RW	Instruction encoding for buffer operations, including compressed instructions	0

Halt status register (haltsum0)

This register is used to indicate that whether the microprocessor is halted. Each bit indicates the halt status of a microprocessor. When there is only one core, only the lowest bit of this register is used.

Table 7-12 haltsum0 register definition

Bit	Name	Access	Description	Reset value
[31:1]	Reserve	RO	Reserved	0
0	haltsum0	RO	0: The microprocessor runs normally 1: The microprocessor is halted	0

In addition to the above registers of the debug module, the debug function also involves some CSRs, mainly including the debug control and status register dcsr and the debug program counter dpc. The registers are described in detail as follows:

Debug control and status register (dcsr)

Table 7-13 dcsr register definition

Bit	Name	Access	Description	Reset value
[31:28]	xdebugver	DRO	0: External debug is not supported. 4: Supports standard external debug. 15: Supports external debug, but does not conform to the specification.	0x4
[27:16]	Reserve	DRO	Reserved	
15	ebreakm	DRW	0: The ebreak instruction in machine mode behaves as described in the privilege documentation. 1: The ebreak command in machine mode can enter debug mode.	0
[14:13]	Reserve	DRO	Reserved	0
12	ebreaku	DRW	0: The ebreak instruction in user mode behaves as described in the privilege documentation. 1: The ebreak command in user mode can enter debug mode.	0
11	stepie	DRW	0: Interrupt disabled under single-step debug 1: Interrupt enabled under single-step debug	0
10	Reserve	DRO	Reserved	0
9	stoptime	DRW	0: System timer runs in debug mode 1 : The system timer stops in debug mode	0
[8:6]	cause	DRO	Cause of entering debug 1: With the ebreak instruction (priority 3) 2: Trigger module (priority 4, the highest) 3: Halt request (priority 1)	0

			4: Single-step debug (priority 0, the lowest) 5: It enters the debug mode after the microprocessor is reset (priority 2) Others: Reserved.	
[5:3]	Reserve	DRO	Reserved	0
2	step	DRW	0: Single-step debug disabled. 1: Single-step debug enabled.	0
[1:0]	prv	DRW	Privileged mode 0: User mode 1: Supervised mode (not supported) 2: Reserved 3: Machine mode Note: Record the privileged mode when entering debug mode, the debugger can modify this value to modify the privileged mode when exiting debug.	0

Debug program counter (dpc)

This register is used to save the address of the next instruction to be executed after the microprocessor enters the debug mode, and the value varies according to the cause of entering the debug mode, and the update rules are also different. The dpc register is described in detail as follows:

Table 7-14 dpc register definition

Bit	Name	Access	Description	Reset value
[31:0]	dpc	DRW	Instruction address	0

The update rules of registers are shown in the following table:：

Table 7-15 dpc update rules

Cause of entering debug	dpc update rules
ebreak	Address of the Ebreak instruction
single step	Address of the instruction next to the current instruction
trigger module	Not supported temporarily
halt request	Address of the next instruction to be executed when entering debug.

7.2 Debug interface

Different from the JTAG interface defined by the standard RISC-V, the Qingke V4 microprocessor provides a 2-wire debug interface and follows the WCH debug interface protocol. The 2-wire interface of the Qingke V4A microprocessor follows the interface protocol V1.0, while the 2-wire interfaces of the Qingke V4B, V4C, and V4F microprocessors follow the interface protocol V1.1. The debug interface is used for the communication between the debug host and the debug module, implementing that the debug host performs read/write operations on the debug module registers. The WCH_Link and open-source schematic diagram and program bin files are available, which can be used for debug of all RISC-V microprocessors.

For specific debug interface protocol, please refer to WCH Debug Protocol Manual.

Chapter 8 Control and status register (CSR)

Some Control and Status Registers (CSRs) are defined in the RISC-V architecture, used to control and record the operating status of the microprocessor. Some CSRs are introduced earlier, this chapter describes in detail the CSR registers implemented by the Qingke V4 microprocessors.

8.1 CSR registers

Table 8-1 List of microprocessor CSRs

Type	Name	CSR address	Access	Description
RISC-V standard CSRs	marchid	0xF12	MRO	Machine architecture ID register
	mimpid	0xF13	MRO	Machine implementation ID register
	mstatus	0x300	MRW	Machine status register
	misa	0x301	MRW	Machine ISA register
	mtvec	0x305	MRW	Machine trap-vector base-address register
	mscratch	0x340	MRW	Machine scratch register
	mepc	0x341	MRW	Machine exception program counter
	mcause	0x342	MRW	Machine cause register
	mtval	0x343	MRW	Machine trap value register
	pmpcfg<i>	0x3A0+i	MRW	Physical memory protection configuration register
	pmpaddr<i>	0x3B0+i	MRW	Physical memory protection address register
	fflags	0x001	URW	Floating-point accrued exception register
	frm	0x002	URW	Floating-point dynamic rounding mode register
	fcsr	0x003	URW	Floating-point control and status register
	dcsr	0x7B0	DRW	Debug control and status register
	dpc	0x7B1	DRW	Debug program counter
	dscratch0	0x7B2	DRW	Debug scratch register 0
	dscratch1	0x7B3	DRW	Debug scratch register 1
Vender-defined CSRs	gintenr	0x800	URW	Global interrupt enable register
	intsyscr	0x804	URW	Interrupt system control register
	corecfgr	0xBC0	MRW	Core configuration register

8.2 RISC-V standard CSRs

Machine architecture ID register (marchid)

This register is a read-only register, used to indicate the hardware architecture ID of the microprocessor, consisting of vender code, architecture code, serial code, and version code. Bit definitions are shown in the table below:

Table 8-2 marchid register definition

Bit	Name	Access	Description	Reset value
31	Reserved	MRO	Reserved	1

[30:26]	Vender0	MRO	Vender ID 0 Always “W”.	0x17
[25:21]	Vender1	MRO	Vender ID 1 Always “C”	0x03
[20:16]	Vender2	MRO	Vender ID 2 Always “H”	0x08
15	Reserved	MRO	Reserved	1
[14:10]	Arch	MRO	Architecture ID Always “V” for RISC-V architecture	0x16
[9:5]	Serial	MRO	Serial ID For Qingke V4 series, always “4”	0x04
[4:0]	Verision	MRO	Version ID Can be “A”, “B”, “C”, “F” or other letters	x

The vender ID and version ID are English letters, while the serial ID is a number. The coding table of letters is shown below:

Table 8-3 Coding table of letters

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Take the Qingke V4F microprocessor as an example, the read value is 0xDC68D886, corresponding to WCH-V4F.

Machine implementation ID register (mimpid)

This register is mainly composed of vender codes. Bit definitions are shown in the table below:

Table 8-4 mimpid register definition

Bit	Name	Access	Description	Reset value
31	Reserved	MRO	Reserved	1
[30:26]	Vender0	MRO	Vender ID 0 Always “W”	0x17
[25:21]	Vender1	MRO	Vender ID 1 Always “C”	0x03
[20:16]	Vender2	MRO	Vender ID 2 Always “H”	0x08
15	Reserved	MRO	Reserved	1
[14:1]	Reserved	MRO	Reserved	0
0	Reserved	MRO	Reserved	1

Machine status register (mstatus)

Some fields of this register have been described earlier. Bit definitions are shown in the table below:

Table 8-5 mstatus register definition

Bit	Name	Access	Description	Reset value
[31:15]	Reserved	MRO	Reserved	0
[14:13]	FS	MRW	Floating-point status	0

			FS: 00 OFF; 01 Initial; 10 Clean; 11 Dirty	
[12:11]	MPP	MRW	Privileged mode before entering interrupt	0
[10:8]	Reserved	MRO	Reserved	0
7	MPIE	MRW	Interrupt enable status before entering interrupt	0
[6:4]	Reserved	MRO	Reserved	0
3	MIE	MRW	Machine interrupt enable	0
[2:0]	Reserved	MRO	Reserved	0

The FS field is used to describe and maintain the floating-point unit state, so this field is only meaningful on the Qingke V4F microprocessor with hardware floating-point function. When its value is 0, the floating-point unit is disabled. In this case, an exception will be triggered if a floating-point instruction is executed. If its value is 1 or 2, this field will be updated to 3 when the floating-point instruction is executed. If the user does not expect to use the hardware floating point function when using the V4F microprocessor, the 2 bits can be manually reset in machine mode to disable the hardware floating point and reduce power consumption.

The MPP field is used to save the privileged mode before entering an exception or interrupt, and restore the privileged mode after exiting an exception or interrupt. MIE is the global interrupt enable bit. The value of MPIE is updated to the value of MIE when entering an exception or interrupt. It should be noted that MIE of the Qingke V4 microprocessor will not be updated to 0 at this time, to ensure that the nested interrupts in machine mode continue to be executed. The microprocessor returns to the machine mode saved by MPP, and MIE returns to the value of MIE when exiting the exception or interrupt.

The Qingke V4 microprocessor supports machine mode and user mode. You can set MPP to 0x3 in the initialization of the startup file, to make the microprocessor only work in machine mode, and it will always remain in machine mode after returning.

Machine ISA register (misa)

This register is used to indicate the architecture of the microprocessor and the supported instruction set extensions. Bit definitions are shown in the table below:

Table 8-6 misa register definition

Bit	Name	Access	Description	Reset value
[31:30]	MXL	MRO	Machine XLEN 1: 32 2: 64 3: 128	1
[29:26]	Reserved	MRO	Reserved	0
[25:0]	Extensions	MRO	Instruction set extensions	x

The MXL is used to indicate the word length of the microprocessor. The Qingke V4 microprocessor is a 32-bit microprocessor, and this field is always 1. The Extensions field is used to indicate the details of the extended instruction set supported by the microprocessor, and each bit represents a type of extension. For detailed description, see table below.

Table 8-7 Instruction set extension description

Bit	Name	Description
0	A	Atomic extension
1	B	Tentatively reserved for Bit-Manipulation extension
2	C	Compressed extension
3	D	Double-precision floating-point extension
4	E	RV32E base ISA
5	F	Single-precision floating-point extension
6	G	Additional standard extensions present
7	H	Hypervisor extension
8	I	RV32I/64I/128I base ISA
9	J	Tentatively reserved for Dynamically Translated Languages extension
10	K	Reserved
11	L	Tentatively reserved for Decimal Floating-Point extension
12	M	Integer Multiply/Divide extension
13	N	User-level interrupts supported
14	O	Reserved
15	P	Tentatively reserved for Packed-SIMD extension
16	Q	Quad-precision floating-point extension
17	R	Reserved
18	S	Supervisor mode implemented
19	T	Tentatively reserved for Transactional Memory extension
20	U	User mode implemented
21	V	Tentatively reserved for Vector extension
22	W	Reserved
23	X	Non-standard extensions present
24	Y	Reserved
25	Z	Reserved

Taking the Qingke V4F microprocessor as an example, the register value is 0x40901125, indicating that the supported instruction set architecture is RV32IMACF, and the non-standard extension X, and has user mode implementation.

Machine trap-vector base-address register（mtvec）

This register is used to store the base address of the exception or interrupt handler, and the lower 2 bits are used to configure the mode and identification method of the vector table, see section 3.2 for details.

Machine scratch register（mscratch）

Table 8-8 mscratch register definition

Bit	Name	Access	Description	Reset value
[31:0]	mscratch	MRW	Data scratch	0

This register is a 32-bit readable and writable register in machine mode, used to temporarily save data. For example, when entering an exception or interrupt handler, the user stack pointer SP is stored into this register, and the interrupt stack pointer is assigned to the SP register. After exiting an exception or exception, the

value of the user stack pointer SP is restored from mscratch, implementing the isolation of the interrupt stack and the user stack.

Machine exception program counter (mepc)

Table 8-9 mepc register definition

Bit	Name	Access	Description	Reset value
[31:0]	mepc	MRW	Exception program counter	0

This register is used to save the program pointer when entering an exception or interrupt, and save the instruction PC pointer before entering the exception when an exception or interrupt is generated. When the exception or interrupt is handeld, mepc is used as the return address for the exception or interrupt return. But it should be noted that:

- When an exception occurs, mepc is updated to the PC value of the current instruction that generates the exception.
- When an interrupt occurs, mepc is updated to the PC value of the next instruction.

So the value of mepc should be modified after the exception is handled and needs to return. For more details, please refer to Chapter 2 Exception.

Machine cause register (mcause)

Table 8-10 mepc register definition

Bit	Name	Access	Description	Reset value
31	Interrupt	MRO	Interrupt 0: Abnormal 1: Interrupt	0
[30:0]	Exception Code	MRO	Exception code, see Table 2-1 for details	0

This register is mainly used to save the cause of the exception or the interrupt ID. The highest bit is the Interrupt field, used to indicate whether an exception or an interrupt is currently occurring. The lower bits are the exception codes, used to indicate the specific cause. For details, please refer to Chapter 2 Exception.

Machine trap value register (mtval)

Table 8-11 mepc register definition

Bit	Name	Access	Description	Reset value
[31:0]	mtval	MRW	Trap value	0

This register is used to save the value which causes the trap. For details on the saved value and time, please refer to Chapter 2 Exception.

PMP configuration register (pmpcfg<i>)

This register is mainly used to configure the physical memory protection unit. Every 8 bits of this register are used to configure the protection of an area. For detailed definitions, please refer to Chapter 4.

PMP address register (pmpaddr<i>)

This register is mainly used to configure the address of the physical memory protection unit. It encodes the upper 32 bits of a 34-bit physical address. Please refer to Chapter 4 for the specific configuration method.

Floating-point control and status register (fcsr)

This register is only provided by microprocessors that support hardware floating point, and is used to configure the rounding mode of floating point calculations and to record floating point exception flags. Bit definitions are shown in the table below:

Table 8-12 fcsr register definition

Bit	Name	Access	Description	Reset value
[31:8]	Reserve	MRO	Reserved	0
[7:5]	FRM	MRW	Floating-point rounding mode	0
4	NV	MRW	Illegal operation exception	0
3	DZ	MRW	Divide by zero exception	0
2	OF	MRW	Overflow exception	0
1	UF	MRW	Underflow exception	0
0	NX	MRW	Non-exact exception	0

It should be noted that it will not trigger the entry of the exception interrupt when the floating-point unit generates an exception, but only set the corresponding flag bit. The flag bit can be reset by software writing 0. The FRM field is used to configure the rounding mode of the floating-point unit. The supported rounding modes are shown in the table below:

Table 8-13 Rounding modes

Code	Rounding mode	Description
000	RNE	Round to nearest value, even values first
001	RTZ	Round towards zero
010	RDN	Round down (towards $-\infty$)
011	RUP	Round up (towards $+\infty$)
100	RMM	Round to nearest value, maximum value first
101	-	Illegal value
110	-	Illegal value
111	-	Dynamic rounding

Floating-point accrued exception register (fflags)

This register exists only in microprocessors that support hardware floating point. It is the exception flag field in fcsr, to facilitate users to read/write exception flags directly and independently using CSR instructions.

Floating-point dynamic rounding mode register (frm)

This register exists only in microprocessors that support hardware floating point. It is the rounding mode field FRM in fcsr, to allow users to directly and independently configure the rounding mode of floating-point calculations using the CSR instructions.

Debug control and status register (dcsr)

This register is used to control and record the running state of the debug mode. Please refer to Section 7.1 for details.

Debug program counter (dpc)

This register is used to save the address of the next instruction to be executed after the microprocessor enters

the debug mode, and its value varies according to the cause of entering the debug mode, and the update rules are also different. Please refer to Section 7.1 for details.

Debug scratch registers (dscratch0-1)
The 2 registers are used to scratch data in debug mode.

Table 8-14 dscratch0-1 register definition

Bit	Name	Access	Description	Reset value
[31:0]	dscratch	DRW	Debug mode data scratch value	0

8.3 User-defined CSRs

Global interrupt enable register (gintenr)
This register is used to control global interrupt enable/disable. Global interrupt enable/disable in machine mode can be controlled by the MIE and MPIE bits in the mstatus register, while the 2 bits cannot work in user mode. The global interrupt enable register gintenr is the mapping of MIE and MPIE in mstatus. In user mode, gintenr can be used to set/reset MIE and MPIE.

Note: Global interrupts do not include non-maskable interrupts NMI or exceptions

Interrupt system control register (intsyscr)
This register is mainly used to configure the functions such as interrupt nested depth and hardware stack. For details, please refer to the relevant description in Section 3.2.

Core configuration register (corecfgr)
This register is mainly used to configure features such as microprocessor pipeline and instruction prediction. Generally, no operation is required. The related MCU products are configured with default values in the startup file.