# QingKeV2 Microprocessor Manual

V1.3

## Overview

QingKe V2 series microprocessor is a 32-bit general-purpose MCU microprocessor based on the standard RISC-V instruction set RV32I subset RV32E, with only 16 general-purpose registers, half of RV32I, and a more streamlined structure for deep embedded scenarios. The V2 series supports standard RV32EC instruction extensions, with the V2C supporting hardware multiplication. In addition to custom XW extensions, Hardware Prologue/Epilogue (HPE), Vector Table Free (VTF), a more streamlined 1-wire/2-wire serial debug interface (SDI), and support for "WFE" instructions.

## Features

| Features | Description |
|---|---|
| Instruction Set Architecture (ISA) | RV32EmC |
| Pipeline | 2 |
| Branch prediction | Static branch prediction |
| Interrupt | Supports a total of 256 interrupts including exceptions, and supports VTF |
| Hardware Prologue/Epilogue (HPE) | Supports 2 levels of HPE |
| Low-power consumption mode | Supports Sleep and Deep sleep modes, and support WFI and WFE sleep methods |
| Extended Instruction Set | Supports half-word and byte operation compression instructions |
| Debug | 1-wire/2-wire SDI, standard RISC-V debug |

# Chapter 1 Overview

QingKe V2 series microprocessors based on RISC-V architecture in line with the RV32EC subset of 32-bit MCU microprocessors, only 16 general-purpose registers, a more streamlined structure, suitable for area and power consumption requirements of deep embedded scenarios, its main features are shown in Table 1-1 below.

Table 1-1 Overview of QingKe V2 microprocessor

| Feature / Model | ISA | HPE number of levels | Interruptions nesting number of levels | VTF number of channels | Pipeline | Vector table mode | Extended Instruction (XW) | Number of memory protection areas |
|---|---|---|---|---|---|---|---|---|
| V2A | RV32EC | 2 | 2 | 2 | 2 | Address/ Instruction | √ | × |
| V2C | RV32EmC | 2 | 2 | 2 | 2 | Address/ Instruction | √ | × |

*Note: OS task switching generally uses stack push, which are not limited in number of levels.*

## 1.1 Instruction Set

QingKe V2 series microprocessors follow the standard RV32EC Instruction Set Architecture (ISA). Detailed documentation of the standard can be found in "The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 2.2" on the RISC-V International website. The RISC-V instruction set has a simple architecture and supports a modular design, allowing for flexible combinations based on different needs. The following instruction set extensions are supported by QingKe V2 series microprocessors.

- RV32: 32-bit architecture, general-purpose register bit width of 32 bits
- E: RV32I subset, only 16 general-purpose registers supported
- C: Supports 16-bit compression instruction
- XW: 16-bit compression instruction for self-extending byte and half-word operations
- m: hardware multiplication, i.e., Zmmul extension.

*Note: 1. To further improve code density, extend the XW subset by adding the following compression directives c.lbu/c.lhu/c.sb/c.sh/c.lbusp/ c.lhusp/c.sbsp/c.shsp, use based on the MRS compiler or the toolchain it provides.*

*2. The m-extension in V2C only includes hardware multiplication instructions, i.e. Zmmul extension, the use of which needs to be based on the MRS compiler or the toolchain it provides.*

## 1.2 Register Set

RV32E is a subset of RV32I, which has only half of its registers. That is, there are 16 register sets from x0-x15. Table 1-2 below lists the registers of RV32E and their descriptions.

Table 1-2 RV32E registers

| Register | ABI Name | Description | Storer |
|---|---|---|---|
| x0 | zero | Hardcoded 0 | - |
| x1 | ra | Return address | Caller |
| x2 | sp | Stack pointer | Callee |
| x3 | gp | Global pointer | - |

| | | | |
|---|---|---|---|
| x4 | tp | Thread pointer | - |
| x5-7 | t0-2 | Temporary register | Caller |
| x8 | s0/fp | Save register/frame pointer | Callee |
| x9 | s1 | Save register | Callee |
| x10-11 | a0-1 | Function parameters/return values | Caller |
| x12-15 | a2-5 | Function parameters | Caller |

The Caller attribute in the above table means that the called procedure does not save the register value, and the Callee attribute means that the called procedure saves the register.

## 1.3 Privilege Mode

The standard RISC-V architecture includes three privileged modes: Machine mode, Supervisor mode, and User mode, as shown in Table 1-3 below. The machine mode is a mandatory mode, and the other modes are optional modes. For details, you can refer to "The RISC-V Instruction Set Manual Volume II: Privileged Architecture", which is available for free download from the RISC-V International website.

Table 1-3 RISC-V architecture privilege mode

| Code | Name | Abbreviations |
|---|---|---|
| 0b00 | User Mode | U |
| 0b01 | Supervisor Model | S |
| 0b10 | Reserved | Reserved |
| 0b11 | Machine mode | M |

The Machine mode has the highest privileges, and the program can access all Control and Status Registers (CSR) in this mode, and there is no Physical Memory Protection (PMP) unit designed inside QingKe V2. The MPP bit in the CSR register mstatus (Machine Mode Status Register) is 0b11 by default, i.e. it is always running in Machine mode.

## 1.4 CSR Register

A series of CSR registers are defined in the RISC-V architecture to control and record the operating state of the microprocessor. These CSRs can be extended by 4096 registers using an internal dedicated 12-bit address coding space. And use the high two CSR[11:10] to define the read/write permission of this register, 0b00, 0b01, 0b10 for read/write allowed and 0b11 for read only. Use CSR[9:8] two bits to define the lowest privilege level that can access this register, and the value corresponds to the privilege mode defined in Table 1-3. In addition to the standard definition of the relevant CSR registers, QingKe V2 series microprocessors are extended with some custom CSR registers for control and status logging of enhanced functions. The CSR registers implemented by the microprocessor are detailed in Chapter 7.

# Chapter 2 Exception

Exception mechanism, which is a mechanism to intercept and handle "unusual operation events". QingKe V2 series microprocessors are equipped with an exception response system that can handle up to 256 exceptions including interrupts. When an exception or interruption occurs, the microprocessor can quickly respond and handle the exception and interruption events.

## 2.1 Exception Types

The hardware behavior of the microprocessor is the same whether an exception or an interrupt occurs. The microprocessor suspends the current program, moves to the exception or interrupt handler, and returns to the previously suspended program when processing is complete. Broadly speaking, interrupts are also part of exceptions. Whether exactly the current occurrence is an interrupt or an exception can be viewed through the Machine mode exception cause register mcause. The mcause[31] is the interrupt field, which is used to indicate whether the cause of the exception is an interrupt or an exception. mcause[31]=1 means interrupt, mcause[31]=0 means exception. mcause[30:0] is the exception code, which is used to indicate the specific cause of the exception or the interrupt number, as shown in the following table.

Table 2-1 V2 microprocessor exception codes

| Interrupt | Exception codes | Synchronous / Asynchronous | Reason for exception |
|---|---|---|---|
| 1 | 0-1 | - | Reserved |
| 1 | 2 | Precise asynchronous | NMI interrupts |
| 1 | 3-11 | - | Reserved |
| 1 | 12 | Precise asynchronous | SysTick interrupts |
| 1 | 13 | - | Reserved |
| 1 | 14 | Synchronous | Software interrupts |
| 1 | 15 | - | Reserved |
| 1 | 16-255 | Precise asynchronous | External interrupt 16-255 |
| 0 | 0 | Synchronous | Instruction address misalignment |
| 0 | 1 | Synchronous | Fetch command access error |
| 0 | 2 | Synchronous | Illegal instructions |
| 0 | 3 | Synchronous | Breakpoints |
| 0 | 4 | Synchronous | Load instruction access address misalignment |
| 0 | 5 | Non- precision asynchronous | Load command access error |
| 0 | 6 | Synchronous | Store/AMO instruction access address misalignment |
| 0 | 7 | Non-precision asynchronous | Store/AMO command access error |
| 0 | 8 | Synchronous | Environment call in User mode (not supported in V2) |
| 0 | 11 | Synchronous | Environment call in Machine mode |

'Synchronous' in the table means that an instruction can be located exactly where it is executed, such as an ebreak or ecall instruction, and each execution of that instruction will trigger an exception. 'Asynchronous' means that it is not possible to pinpoint an instruction, and the instruction PC value may be different each time an exception occurs. 'Precise asynchronous' means that an exception can be located exactly at the boundary of an instruction, i.e., the state after the execution of an instruction, such as an external interrupt.

'Non- precision asynchronous' means that the boundary of an instruction cannot be precisely located, and may be the state after an instruction has been interrupted halfway through execution, such as a memory access error. Access to memory takes time, and the microprocessor usually does not wait for the end of the access when accessing memory, but continues to execute the instruction, when the access error exception occurs again, the microprocessor has already executed the subsequent instructions, and cannot be precisely located.

## 2.2 Entering Exception

When the program is in the process of normal operation, if for some reason, triggered into an exception or interrupt. The hardware behavior of the microprocessor at this point can be summarized as follows.

(1) Suspend the current program flow and move to the execution of exception or interrupt handling functions.

The entry base address and addressing mode of the exception or interrupt function are defined by the exception entry base address register mtvec. mtvec[31:2] defines the base address of the exception or interrupt function. mtvec[1:0] defines the addressing mode of the handler function, where mtvec[0] defines the entry mode of the exception and interrupt. when mtvec[0]=0, all exceptions and interrupts use When mtvec[0]=0, all exceptions and interrupts use a unified entry, i.e., when an exception or interrupt occurs, it turns to the base address defined by mtvec[31:2] for execution. When mtvec[0]=1, exceptions and interrupts use vector table mode, i.e., each exception and interrupt is numbered, and the address is shifted according to interrupt number*4, and when an exception or interrupt occurs, it is shifted to the base address defined by mtvec[31:2] + interrupt number*4 for execution. The vector mode mtvec[1] defines the identification mode of the vector table. When mtvec[1]=0, the instruction stored at the vector table is an instruction to jump to the exception or interrupt handling function, or it can be another instruction; when mtvec[1]=1, the absolute address of the exception handling function is stored at the vector table. It should be noted that the vector table base address needs to be 1KB aligned in the QingKe V2 microprocessor.

(2) Update CSR register

When an exception or interrupt is entered, the microprocessor automatically updates the relevant CSR registers, including the Machine mode exception cause register mcause, the Machine mode exception pointer register mepc, and the Machine mode status register mstatus.

- Update mcause

As mentioned before, after entering an exception or interrupt, its value reflects the current exception type or interrupt number, and the software can read this register value to check the cause of the exception or determine the source of the interrupt, as detailed in Table 2-1.

- Update mepc

The standard definition of the return address of the microprocessor after exiting an exception or interrupt is stored in mepc. So when an exception or interrupt occurs, the hardware automatically updates the mepc value to the current instruction PC value when the exception is encountered, or the next pre-executed instruction PC value before the interrupt. After the exception or interrupt is processed, the microprocessor uses its saved value as the return address to return to the location of the interrupt to continue execution.

However, it is worth noting that.

1. mepc is a readable and writable register, and the software can also modify the value for the purpose of modifying the location of the PC pointer running after the return.
2. When an interrupt occurs, i.e., when the exception cause register mcause[31]=1, the value of mepc is

updated to the PC value of the next unexecuted instruction at the time of the interrupt.

And when an exception occurs, that is, when the exception cause register mcause[31]=0, the value of mepc is updated to the instruction PC value of the current exception. So at this time when the exception returns, if we return directly using the value of mepc, we still continue to execute the instruction that generated the exception before, and at this time, we will continue to enter the exception. Usually, after we handle the exception, we can modify the value of mepc to the value of the next unexecuted instruction and then return. For example, we cause an exception due to ecall/ebreak, and after handling the exception, since ecall/ebreak (c.ebreak is 2 bytes) is a 4-byte instruction, we only need the software to modify the value of mepc to mepc+4 (c.ebreak is mepc+2) and then return.

- Update mstatus

Upon entering exceptions and interrupts, the hardware updates certain bits in mstatus.

1. MPIE is updated to the MIE value before entering the exception or interrupt, and MPIE is used to restore the MIE after the exception and interrupt are over.
2. MPP bit is used to save the privileged mode before entering the exception or interrupt, QingKe V2 only supports Machine mode, solid it keeps 0b11 unchanged.
3. QingKe V2 microprocessor supports interrupt nesting in Machine mode, and the MIE will not be cleared before entering the last level of exceptions and interrupts.
4. The out-stack flag MPOP is updated to the current exception or interrupt out stack flag, and MPPOP is updated to MPOP.

## 2.3 Exception Handling Functions

Upon entering an exception or interrupt, the microprocessor executes the program from the address and mode defined by the mtvec register. When using the unified entry, the microprocessor takes a jump instruction from the base address defined by mtvec[31:2] based on the value of mtvec[1], or gets the exception and interrupt handling function entry address and goes to execute it instead. At this time, the exception and interrupt handling function can determine whether the cause is an exception or interrupt based on the value of mcause[31], and the type and cause of the exception or the corresponding interrupt can be judged by the exception code and handled accordingly.

When using the base address + interrupt number *4 for offset, the hardware automatically jumps to the vector table to get the entry address of the exception or interrupt function based on the interrupt number and jumps to execute it.

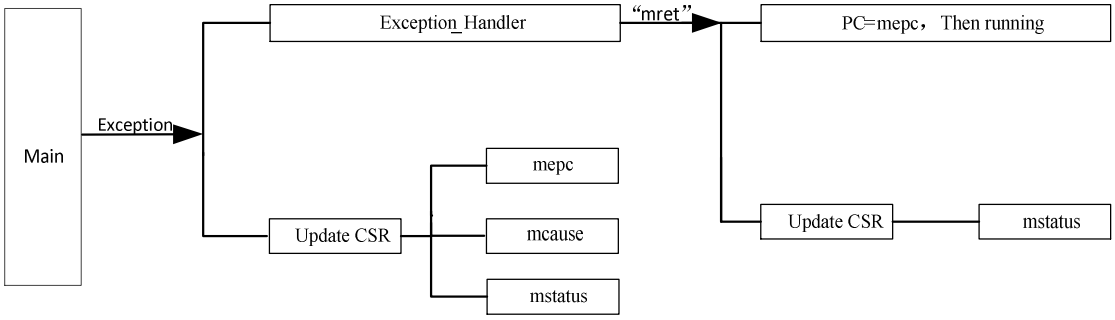
## 2.4 Exception Exit

After the exception or interrupt handler is completed, it is necessary to exit from the service program. After entering exceptions and interrupts, the microprocessor enters Machine mode from User mode, and the processing of exceptions and interrupts is also completed in Machine mode. When it is necessary to exit exceptions and interrupts, it is necessary to use the mret instruction to return. At this time, the microprocessor hardware will automatically perform the following operations.

- The PC pointer is restored to the value of CSR register mepc, i.e., execution starts at the instruction address saved by mepc. It is necessary to pay attention to the offset operation of mepc after the exception handling is completed.
- Update the CSR register mstatus, MIE is restored to MPIE and MPOP is restored to MPPOP.

The entire exception response process can be described by the following Figure 2-1.

Figure 2-1 Exception response process diagram



# Chapter 3 PFIC and Interrupt Control

QingKe V2 microprocessor is designed with a Programmable Fast Interrupt Controller (PFIC) that can manage up to 256 interrupts including exceptions. The first 16 of them are fixed as internal interrupts of the microprocessor, and the rest are external interrupts, i.e. the maximum number of external interrupts can be extended to 240. Its main features are as follows.

- 240 external interrupts, each interrupt request has independent trigger and mask control bits, with dedicated status bits
- Programmable interrupt priority, supports 2 levels of nesting
- Special fast interrupt in/out mechanism, hardware automatic stacking and recovery, maximum HPE depth of 2 levels
- Vector Table Free (VTF) interrupt response mechanism, 2-channel programmable direct access to interrupt vector addresses

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

Table 3-1 Exception and interrupt vector table

| Number | Priority | Type | Name | Description |
|---|---|---|---|---|
| 0 | - | - | - | - |
| 1 | - | - | - | - |
| 2 | -2 | Fixed | NMI | Non-maskable interrupt |
| 3 | -1 | Fixed | EXC | Exception interrupt |
| 4-11 | - | - | - | - |
| 12 | 0 | Programmable | SysTick | System timer interrupt |
| 13 | - | - | - | - |
| 14 | 1 | Programmable | SWI | Software interrupt |
| 15 | - | - | - | - |
| 16-255 | 2-241 | Programmable | External interrupt | External interrupt 16-255 |

## 3.1 PFIC Register Set

Table 3-2 PFIC Registers

| Name | Access address | Access | Description | Reset value |
|---|---|---|---|---|
| PFIC_ISRx | 0xE000E000<br>-0xE000E01C | RO | Interrupt enable status register x | 0x00000000 |
| PFIC_IPRx | 0xE000E020<br>-0xE000E03C | RO | Interrupt pending status register x | 0x00000000 |
| PFIC_ITHRESDR | 0xE000E040 | RW | Interrupt priority threshold configuration register | 0x00000000 |
| PFIC_CFGR | 0xE000E048 | RW | Interrupt configuration register | 0x00000000 |
| PFIC_GISR | 0xE000E04C | RO | Interrupt global status register | 0x00000000 |
| PFIC_VTFIDR | 0xE000E050 | RW | VTF ID configuration register | 0x00000000 |
| PFIC_VTFADDRRx | 0xE000E060<br>-0xE000E06C | RW | VTF x offset address register | 0x00000000 |

| | | | | |
|---|---|---|---|---|
| PFIC_IENRx | 0xE000E100<br>-0xE000E11C | WO | Interrupt enable setting register x | 0x00000000 |
| PFIC_IRERx | 0xE000E180<br>-0xE000E19C | WO | Interrupt enable clear register x | 0x00000000 |
| PFIC_IPSRx | 0xE000E200<br>-0xE000E21C | WO | Interrupt pending setting register x | 0x00000000 |
| PFIC_IPRRx | 0xE000E280<br>-0xE000E29C | WO | Interrupt pending clear register x | 0x00000000 |
| PFIC_IACTRx | 0xE000E300<br>-0xE000E31C | RO | Interrupt activation status register x | 0x00000000 |
| PFIC_IPRIORx | 0xE000E400<br>-0xE000E43C | RW | Interrupt priority configuration register | 0x00000000 |
| PFIC_SCTLR | 0xE000ED10 | RW | System control register | 0x00000000 |

*Note: 1. The default value of PFIC_ISR0 register is 0xC, which means that NMI and exception are always enabled by default.*

*2. NMI and EXC support interrupt pending clear and setting operation, but not interrupt enable clear and setting operation.*

Each register is described as follows.

**Interrupt Enable Status and Interrupt Pending Status Registers (PFIC_ISR<0-7>/PFIC_IPR<0-7>)**

| Name | Access address | Access | Description | Reset value |
|---|---|---|---|---|
| PFIC_ISR0 | 0xE000E000 | RO | Interrupt 0-31 enable status register, a total of 32 status bits [n], indicating #n interrupt enable status<br>*Note: NMI and EXC are enabled by default* | 0x0000000C |
| PFIC_ISR1 | 0xE000E004 | RO | Interrupt 32-63 enable status register, total 32 status bits | 0x00000000 |
| … | … | … | … | … |
| PFIC_ISR7 | 0xE000E01C | RO | Interrupt 224-255 enable status register, total 32 status bits | 0x00000000 |
| PFIC_IPR0 | 0xE000E020 | RO | Interrupt 0-31 pending status register, a total of 32 status bits [n], indicating the pending status of interrupt #n | 0x00000000 |
| PFIC_IPR1 | 0xE000E024 | RO | Interrupt 32-63 pending status registers, 32 status bits in total | 0x00000000 |
| … | … | … | … | … |
| PFIC_IPR7 | 0xE000E03C | RO | Interrupt 244-255 pending status register, 32 status bits in total | 0x00000000 |

2 sets of registers are used to enable and de-enable the corresponding interrupts.

**Interrupt Enable Setting and Clear Registers (PFIC_IENR<0-7>/PFIC_IRER<0-7>)**

| Name | Access address | Access | Description | Reset value |
|---|---|---|---|---|
| PFIC_IENR0 | 0xE000E100 | WO | Interrupt 0-31 enable setting register, a total of 32 setting bits [n], for interrupt #n enable setting<br>*Note: NMI and EXC are enabled by default* | 0x00000000 |
| PFIC_IENR1 | 0xE000E104 | WO | Interrupt 32-63 enable setting register, total 32 setting bits | 0x00000000 |
| … | … | … | … | … |
| PFIC_IENR7 | 0xE000E11C | WO | Interrupt 224-255 enable setting register, total 32 setting bits | 0x00000000 |
| - | - | - | - | - |
| PFIC_IRER0 | 0xE000E180 | WO | Interrupt 0-31 enable clear register, a total of 32 clear bits [n], for interrupt #n enable clear<br>*Note: NMI and EXC cannot be operated* | 0x00000000 |
| PFIC_IRER1 | 0xE000E184 | WO | Interrupt 32-63 enable clear register, total 32 clear bits | 0x00000000 |
| … | … | … | … | … |
| PFIC_IRER7 | 0xE000E19C | WO | Interrupt 244-255 enable clear register, total 32 clear bits | 0x00000000 |

2 sets of registers are used to enable and de-enable the corresponding interrupts.

*Note: When using registers to mask any interrupt or using CSR registers to mask global interrupts, add a 'fence.i' instruction to synchronize between core control state and interrupt enable state.*

**Interrupt Pending Setting and Clear Registers (PFIC_IPSR<0-7>/PFIC_IPRR<0-7>)**

| Name | Access address | Access | Description | Reset value |
|---|---|---|---|---|
| PFIC_IPSR0 | 0xE000E200 | WO | Interrupt 0-31 pending setting register, 32 setting bits [n], for interrupt #n pending setting | 0x00000000 |
| PFIC_IPSR1 | 0xE000E204 | WO | Interrupt 32-63 pending setup register, total 32 setup bits | 0x00000000 |
| … | … | … | … | … |
| PFIC_IPSR7 | 0xE000E21C | WO | Interrupt 224-255 pending setting register, 32 setting bits in total | 0x00000000 |
| - | - | - | - | - |
| PFIC_IPRR0 | 0xE000E280 | WO | Interrupt 0-31 pending clear register, a total of 32 clear bits [n], for interrupt #n pending clear | 0x00000000 |
| PFIC_IPRR1 | 0xE000E284 | WO | Interrupt 32-63 pending clear register, total 32 clear bits | 0x00000000 |
| … | … | … | … | … |
| PFIC_IPRR7 | 0xE000E29C | WO | Interrupt 244-255 pending clear register, | 0x00000000 |

| | | | | |
|---|---|---|---|---|
| | | | total 32 clear bits | |

When the microprocessor enables an interrupt, it can be set directly through the interrupt pending register to trigger into the interrupt. Use the interrupt pending clear register to clear the pending trigger.

**Interrupt Activation Status Register (PFIC_IACTR<0-7>)**

| Name | Access address | Access | Description | Reset value |
|---|---|---|---|---|
| PFIC_IACTR0 | 0xE000E300 | RO | Interrupt 0-31 activates the status register with 32 status bits [n], indicating that interrupt #n is being executed | 0x00000000 |
| PFIC_IACTR1 | 0xE000E304 | RO | Interrupt 32-63 activation status registers, 32 status bits in total | 0x00000000 |
| … | … | … | … | … |
| PFIC_IACTR7 | 0xE000E31C | RO | Interrupt 224-255 activation status register, total 32 status bits | 0x00000000 |

Each interrupt has an active status bit that is set up when the interrupt is entered and cleared by hardware when mret returns.

**Interrupt Priority and Priority Threshold Registers (PFIC_IPRIOR<0-7>/PFIC_ITHRESDR)**

| Name | Access address | Access | Description | Reset value |
|---|---|---|---|---|
| PFIC_IPRIOR0 | 0xE000E400 | RW | Interrupt 0 priority configuration. [7:6]: Priority control bits<br>If the configuration is not nested, no preemption bit<br>If nesting is configured, bit7 is the preempted bit.<br>[5:0]: Reserved, fixed to 0<br>*Note: The smaller the priority value, the higher the priority. If the same preemption priority interrupt hangs at the same time, the interrupt with the higher priority will be executed first.* | 0x00 |
| PFIC_IPRIOR1 | 0xE000E401 | RW | Interrupt 1 priority setting, same function as PFIC_IPRIOR0 | 0x00 |
| PFIC_IPRIOR2 | 0xE000E402 | RW | Interrupt 2 priority setting, same function as PFIC_IPRIOR0 | |
| … | … | … | … | … |
| PFIC_IPRIOR254 | 0xE000E4FE | RW | Interrupt 254 priority setting, same function as PFIC_IPRIOR0 | 0x00 |
| PFIC_IPRIOR255 | 0xE000E4FF | RW | Interrupt 255 priority setting, same function as PFIC_IPRIOR0 | 0x00 |

| - | - | - | - | - |
|---|---|---|---|---|
| PFIC_ITHRESDR | 0xE000E040 | RW | Interrupt priority threshold setting<br>[31:8]: Reserved, fixed to 0<br>[7:6]: Priority threshold<br>[5:0]: Reserved, fixed to 0<br>*Note: For interrupts with priority value ⩾ threshold, the interrupt service function is not executed when a hang occurs, and when this register is 0, it means the threshold register is invalid.* | 0x00 |

**Interrupt Configuration Register (PFIC_CFGR)**

| Name | Access address | Access | Description | Reset value |
|---|---|---|---|---|
| PFIC_CFGR | 0xE000E048 | RW | Interrupt configuration register | 0x00000000 |

Its folks are defined as.

| Bit | Name | Access | Description | Reset value |
|---|---|---|---|---|
| [31:16] | KEYCODE | WO | Corresponding to different target control bits, the corresponding security access identification data needs to be written simultaneously in order to be modified, and the readout data is fixed to 0.<br>KEY1 = 0xFA05;<br>KEY2 = 0xBCAF;<br>KEY3 = 0xBEEF。 | 0 |
| [15:8] | Reserved | RO | Reserved | 0 |
| 7 | SYSRESET | WO | System reset (simultaneous writing to KEY3). Auto clear 0.<br>Writing 1 is valid, writing 0 is invalid.<br>*Note: Same function as the PFIC_SCTLR register SYSRESET bit.* | 0 |
| [6:0] | Reserved | RO | Reserved | 0 |

V2 series microprocessor This register is mainly used for compatible

**Interrupt Global Status Register (PFIC_GISR)**

| Name | Access address | Access | Description | Reset value |
|---|---|---|---|---|
| PFIC_CFGR | 0xE000E04C | RO | Interrupt global status register | 0x00000000 |

Its folks are defined as.

| Bit | Name | Access | Description | Reset value |
|---|---|---|---|---|
| [31:10] | Reserved | RO | Reserved | 0 |
| 9 | GPENDSTA | RO | Whether an interrupt is currently pending. | 0 |

| | | | | |
|---|---|---|---|---|
| | | | 1: Yes; 0: No. | |
| 8 | GACTSTA | RO | Whether an interrupt is currently being executed.<br>1: Yes; 0: No. | 0 |
| [7:0] | NESTSTA | RO | Current interrupt nesting status.<br>0x03: in level 2 interrupt.<br>0x01: in level 1 interrupt.<br>0x00: no interrupts occur.<br>Other: Impossible situation. | 0 |

**VTF ID and Address Registers (PFIC_VTFIDR/PFIC_VTFADDRR<0-1>)**

| Name | Access address | Access | Description | Reset value |
|---|---|---|---|---|
| PFIC_VTFIDR | 0xE000E050 | RW | [15:8]: number of VTF 1<br>[7:0]: number of VTF 0 | 0x00000000 |
| - | - | - | - | - |
| PFIC_VTFADDRR0 | 0xE000E060 | RW | [31:1]: VTF 0 address, two-byte alignment<br>[0]:<br>1: Enable VTF 0 channel<br>0: Close | 0x00000000 |
| PFIC_VTFADDRR1 | 0xE000E064 | RW | [31:1]: VTF 1 address, two-byte alignment<br>[0]:<br>1: Enable VTF 1 channel<br>0: Close | 0x00000000 |

**System Control Register (PFIC_SCTLR)**

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

Each of them is defined as follows.

| Bit | Name | Access | Description | Reset value |
|---|---|---|---|---|
| 31 | SYSRESET | WO | System reset, clear 0 automatically. write 1 valid, write 0 invalid, same effect as PFIC_CFGR register | 0 |
| [30:6] | Reserved | RO | Reserved | 0 |
| 5 | SETEVENT | WO | Set the event to wake up the WFE case. | 0 |
| 4 | SEVONPEND | RW | When an event occurs or interrupts a pending state, the system can be woken up from after the WFE instruction, or if the WFE instruction is not executed, the system will be woken up immediately after the next execution of the instruction.<br>1: Enabled events and all interrupts (including unenabled interrupts) can | 0 |

| | | | | |
|---|---|---|---|---|
| | | | wake up the system.<br>0: Only enabled events and enabled interrupts can wake up the system. | |
| 3 | WFITOWFE | RW | Execute the WFI command as if it were a WFE.<br>1: treat the subsequent WFI instruction as a WFE instruction.<br>0: No effect. | 0 |
| 2 | SLEEPDEEP | RW | Low power mode of the control system.<br>1: deepsleep 0: sleep | 0 |
| 1 | SLEEPONEXIT | RW | System status after control leaves the interrupt service program.<br>1: The system enters low-power mode.<br>0: The system enters the main program. | 0 |
| 0 | Reserved | RO | Reserved | 0 |

## 3.2 Interrupt-related CSR Registers

In addition, the following CSR registers also have a significant impact on the processing of interrupts.

**Interrupt System Control Register (INTSYSCR)**

| Name | CSR Address | Access | Description | Reset value |
|---|---|---|---|---|
| INTSYSCR | 0x804 | MRW | Interrupt system control register | 0x00000000 |

Its folks are defined as.

| Bit | Name | Access | Description | Reset value |
|---|---|---|---|---|
| [31:3] | Reserved | MRO | Reserved | 0 |
| 2 | EABIEN | MRW | EABI enable.<br>0: EABI off.<br>1: EABI enabled. | 0 |
| 1 | INESTEN | MRW | Interrupt nesting enable.<br>0: Interrupt nesting function off.<br>1: Interrupt nesting function enabled. | 0 |
| 0 | HWSTKEN | MRW | HPE enable.<br>0: HPE function off.<br>1: HPE function enabled. | 0 |

**Machine Mode Exception Base Address Register (mtvec)**

| Name | CSR Address | Access | Description | Reset value |
|---|---|---|---|---|
| mtvec | 0x305 | MRW | Exception base address register | 0x00000000 |

Its folks are defined as.

| Bit | Name | Access | Description | Reset value |
|---|---|---|---|---|
| [31:2] | BASEADDR[31:2] | MRW | The interrupt vector table base address, which needs to be 1KB aligned. | 0 |
| 1 | MODE1 | MRW | Interrupt vector table identifies patterns.<br>0: Identification by jump instruction, limited range, support for non-jump instructions.<br>1: Identify by absolute address, support full range, but must jump. | 0 |
| 0 | MODE0 | MRW | Interrupt or exception entry address mode selection.<br>0: Use of the uniform entry address.<br>1: Address offset based on interrupt number *4. | 0 |

For MCU of V2 series microprocessors, MODE[1:0]=11 is configured in the startup file by default, i.e. the vector table uses the absolute address of the interrupt function and the entry of the exception or interrupt is offset according to the interrupt number *4.

## 3.3 Interrupt Nesting

In conjunction with the interrupt system control register INTSYSCR (CSR address: 0x804) and the interrupt priority register PFIC_IPRIOR, nesting of interrupts can be allowed to occur. Enable nesting in the Interrupt System Control Register (configured in the startup file for V2 series MCUs) and configure the priority of the corresponding interrupt. The smaller the priority value, the higher the priority. The smaller the value of the preemption bit, the higher the preemption priority. If there are interrupts hanging at the same time under the same preemption priority, the microprocessor responds to the interrupt with the lower priority value (higher priority) first.

## 3.4 Hardware Prologue/Epilogue (HPE)

When an exception or interrupt occurs, the microprocessor stops the current program flow and shifts to the execution of the exception or interrupt handling function, the site of the current program flow needs to be saved. After the exception or interrupt returns, it is necessary to restore the site and continue the execution of the stopped program flow. For V2 series microprocessors, the "site" here refers to all the Caller Saved registers in Table 1-2.

V2 series microprocessors, support hardware to automatically save 10 of the shaped Caller Saved registers to the user stack area, and when an exception or interrupt returns, the hardware automatically restores data from the user stack area to the 10 shaped registers. HPE supports nesting, and the maximum nesting depth is 2 levels. Suppose the value of SP at the beginning of the stack is N. When the stack is inserted, the SP value is automatically offset by 48 bytes and the SP value is updated to N-48. The changes inside the stack after the stack is inserted are shown in the table below.

Table 3-3 V2 processor stack entry order

| SP Address | Register | Description |
|---|---|---|
| Old SP value(N-0) | - | The contents of the stack already pressed in |
| N-4 | x1 | Automatic saving of x1 values |

| | | |
|---|---|---|
| N-8 | x5 | Automatic saving of x5 values |
| N-12 | x6 | Automatic saving of x6 values |
| N-16 | x7 | Automatic saving of x7 values |
| N-20 | x10 | Automatic saving of x10 values |
| N-24 | x11 | Automatic saving of x11 values |
| N-28 | x12 | Automatic saving of x12 values |
| N-32 | x13 | Automatic saving of x13 values |
| N-36 | x14 | Automatic saving of x14 values |
| N-40 | x15 | Automatic saving of x15 values |
| … | … | … |
| New SP value(N-48) | - | SP value after entering the stack |

*Note: 1. Interrupt functions using the HPE need to be compiled using MRS or its provided toolchain and the interrupt function needs to be declared with __attribute__((interrupt("WCH-Interrupt-fast"))).*

*2. The interrupt function using stack push is declared by __attribute__((interrupt())).*

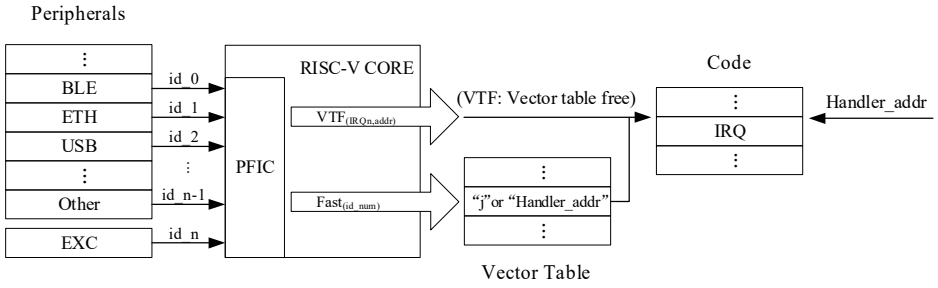
## 3.5 Vector Table Free (VTF)

The Programmable Fast Interrupt Controller (PFIC) provides two VTF channels, i.e., direct access to the interrupt function entry without going through the interrupt vector table lookup process.

While configuring an interrupt function normally, write its interrupt number to one of the channels x of the VTF ID register PFIC_VTFIDR, and write its entry address to the corresponding VTF address register VTFADDRRx, and enable the VTF for that channel.

The PFIC responds to fast interrupts and VTF as shown in Figure 3-2 below.

Figure 3-2 Schematic diagram of programmable fast interrupt controller




## 3.6 Tail-chaining and Late Arrivals

QingKe V2 series HPE saves registers to the user stack area and supports priority configuration and interrupt nesting. When the tail-chaining phenomenon occurs, i.e., when responding to a high-priority interrupt, there is a low-priority interrupt hanging at the same time. When the high-priority interrupt is executed, the stack out of the high-priority middle end is omitted and the stack in operation before executing the pending interrupt is performed to reduce the interrupt latency.

Similarly, when the processor is processing the stacking operation of a low-priority interrupt and receives a late request for a high-priority interrupt, the high-priority interrupt does not repeat the stacking operation and directly follows the previous operation.

# Chapter 4 System Timer (SysTick)

QingKe V2 series microprocessor is designed with a 32-bit plus counter (SysTick) inside, and its clock source can be the system clock or 8 divisions of the system clock. It can provide time base for real time operating system, provide timing, measure time, etc. The timer involves four registers and maps to the peripheral address space for controlling the SysTick, as shown in Table 4-1 below.

Table 4-1 SysTick register list

| Name | Access address | Description | Reset value |
|---|---|---|---|
| STK_CTLR | 0xE000F000 | System count control register | 0x00000000 |
| STK_SR | 0xE000F004 | System count status register | 0x00000000 |
| STK_CNTR | 0xE000F008 | System counter register | 0x00000000 |
| STK_CMPR | 0xE000F010 | System count comparison value register | 0x00000000 |

Each register is described in detail as follows.

**System Count Control Register (STK_CTLR)**

Table 4-2 SysTick Control Registers

| Bit | Name | Access | Description | Reset value |
|---|---|---|---|---|
| 31 | SWIE | RW | Software interrupt trigger enable (SWI).<br>1: Triggering software interrupts.<br>0: Turn off the trigger.<br>After entering software interrupt, software clear 0 is required, otherwise it is continuously triggered. | 0 |
| [30:4] | Reserved | RO | Reserved | 0 |
| 3 | STRE | RW | Auto-reload Count enable bit.<br>1: Count up to the comparison value and start counting from 0 again<br>0: Continue counting up. | 0 |
| 2 | STCLK | RW | Counter clock source selection bit.<br>1: HCLK for time base.<br>0: HCLK/8 for time base. | 0 |
| 1 | STIE | RW | Counter interrupt enable control bit.<br>1: Enabling counter interrupts.<br>0: Turn off the counter interrupt. | 0 |
| 0 | STE | RW | System counter enable control bit.<br>1: Start the system counter STK.<br>0: Turn off the system counter STK and the counter stops counting. | 0 |

**System Count Status Register (STK_SR)**

Table 4-3 SysTick Status Register

| Bit | Name | Access | Description | Reset value |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| [31:1] | Reserved | RO | Reserved | 0 |
| 0 | CNTIF | RW0 | Counting value comparison flag, write 0 clear, write 1 invalid:<br>1: upward counting to reach the comparison value.<br>0: The comparison value is not reached. | 0 |

**System Counter Value Register (STK_CNTR)**

| Bit | Name | Access | Description | Reset value |
|---|---|---|---|---|
| [31:0] | CNTR | RW | Current counter count value. | 0 |

**System Count Comparison Value Register (STK_CMPR)**

| Bit | Name | Access | Description | Reset value |
|---|---|---|---|---|
| [31:0] | CMPR | RW | Set counter comparison value | 0 |

# Chapter 5 Processor Low-power Settings

QingKe V2 series microprocessors support sleep state via WFI (Wait For Interrupt) instruction to achieve low static power consumption. Combined with PFIC's system control register (PFIC_SCTLR), it can implement various Sleep modes and WFE commands

## 5.1 Enter Sleep

QingKe V2 series microprocessors can go to sleep in two ways, Wait for Interrupt (WFI) and Wait For Event (WFE).The WFI method means that the microprocessor goes to sleep, waits for an interrupt to wake up, and then wakes up to the corresponding interrupt to execute. The WFE method means that the microprocessor goes to sleep, waits for an event to wake up, and wakes up to continue executing the previously stopped program flow.

The standard RISC-V supports WFI instruction, and the WFI command can be executed directly to enter sleep by WFI method. For the WFE method, the WFITOWFE bit in the system control register PFIC_SCTLR is used to control the subsequent WFI commands as WFE processing to achieve the WFE method to enter sleep.

The depth of sleep is controlled according to the SLEEPDEEP bit in PFIC_SCTLR.

- If the SLEEPDEEP in PFIC_SCTLR register is cleared to zero, the microprocessor enters Sleep mode and the internal unit clock is allowed to be turned off except for SysTick and part of the wake-up logic.
- If SLEEPDEEP in the PFIC_SCTLR register is set, the microprocessor enters Deep sleep mode and all cell clocks are allowed to be turned off.

When the microprocessor is in Debug mode, it is not possible to enter any kind of Sleep mode.

## 5.2 Sleep Wakeup

QingKe V2 series microprocessors can be woken up after sleep due to WFI and WFE in the following ways.

- After the WFI method goes to sleep, it can be awakened by

(1) The microprocessor can be woken up by the interrupt source responded by the interrupt controller. After waking up, the microprocessor executes the interrupt function first.

(2) Enter Sleep mode, debug request can make the microprocessor wake up and enter deep sleep, debug request cannot wake up the microprocessor.

- After the WFE method goes to sleep, the microprocessor can be woken up by the following.

(1) Internal or external events, when there is no need to configure the interrupt controller, wake up and continue to execute the program.

(2) If an interrupt source is enabled, the microprocessor is woken up when an interrupt is generated, and after waking up, the microprocessor executes the interrupt function first.

(3) If the SEVONPEND bit in PFIC_SCTLR is configured, the interrupt controller does not enable the interrupt under, but when a new interrupt pending signal is generated (the previously generated pending signal does not take effect), it can also make the microprocessor wake up, and the corresponding interrupt pending flag needs to be cleared manually after waking up.

(4) Enter Sleep mode debug request can make the microprocessor wake up and enter deep sleep, debug request cannot wake up the microprocessor.

In addition, the state of the microprocessor after wake-up can be controlled by configuring the SLEEPONEXIT bit in PFIC_SCTLR.

- SLEEPONEXIT is set and the last level interrupt return instruction (mret) will trigger the WFI mode sleep.

- SLEEPONEXIT is cleared with no effect.

Various MCU products equipped with V2 series microprocessors can adopt different sleep modes, turn off different peripherals and clocks, implement different power management policies and wake-up methods according to different configurations of PFIC_SCTLR, and realize various low-power modes.

# Chapter 6 Debug Support

QingKe V2 series microprocessors include a hardware debug module that supports complex debugging operations. When the microprocessor is suspended, the debug module can access the microprocessor's GPRs, CSRs, Memory, external devices, etc. through abstract commands, program buffer deployment instructions, etc. The debug module can suspend and resume the microprocessor's operation.

The debug module follows the RISC-V External Debug Support Version0.13.2 specification, detailed documentation can be downloaded from RISC-V International website.

## 6.1 Debug Module

The debug module inside the microprocessor, capable of performing debug operations issued by the debug host, includes.

- Access to registers through the debug interface
- Reset, suspend and resume the microprocessor through the debug interface
- Read and write memory, instruction registers and external devices through the debug interface
- Deploy multiple arbitrary instructions through the debug interface
- Set software breakpoints through the debug interface
- Support single-step debugging

The internal registers of the debugging module use a 7-bit address code, and the following registers are implemented inside QingKe V2 series microprocessors.

Table 6-1 Debug module register List

| Name | Access address | Description |
|---|---|---|
| data0 | 0x04 | Data register 0, can be used for temporary storage of data |
| data1 | 0x05 | Data register 1, can be used for temporary storage of data |
| dmcontrol | 0x10 | Debug module control register |
| dmstatus | 0x11 | Debug module status register |
| hartinfo | 0x12 | Microprocessor status register |
| abstractcs | 0x16 | Abstract command status register |
| command | 0x17 | Abstract command register |
| progbuf0-7 | 0x20-0x27 | Instruction cache registers 0-7 |
| haltsum0 | 0x40 | Pause status register |

The debug host can control the microprocessor's suspend, resume, reset, etc. by configuring the dmcontrol register. The RISC-V standard defines three types of abstract commands: access registers, fast access, and access memory. QingKe V2 microprocessor supports register (GPRs, CSRs, FPRs) access through abstract commands.

The debug module implements eight instruction cache registers progbuf0-7, and the debug host can cache multiple instructions (which can be compressed instructions) to the buffer, and can choose to continue to execute the instructions in the instruction cache registers after executing the abstract command or execute the

cached instructions directly. It should be noted that if the instruction in progbufs is less than 32 bytes, the last instruction needs to be an "ebreak" or "c.ebreak" instruction, and if the instruction fills 32 bytes, the debug module automatically adds an "ebreak" instruction. The debug host can access the abstract command and the instructions cached in the progbufs, and also the storage, peripherals, etc.

Each register is described in detail as follows.

**Data Register 0 (data0)**

Table 6-2 data0 register definition

| Bit | Name | Access | Description | Reset Value |
|---|---|---|---|---|
| [31:0] | data0 | RW | Data register 0, used for temporary storage of data | 0 |

**Data Register 1 (data1)**

Table 6-3 data1 register definition

| Bit | Name | Access | Description | Reset Value |
|---|---|---|---|---|
| [31:0] | data1 | RW | Data register 1, used for temporary storage of data | 0 |

**Debug Module Control Register (dmcontrol)**

This register controls the pause, reset, and resume of the microprocessor. Debug host write data to the corresponding field to achieve pause (haltreq), reset (ndmreset), resume (resumereq). You describe into the following.

Table 6-4 dmcontrol register definition

| Bit | Name | Access | Description | Reset Value |
|---|---|---|---|---|
| 31 | haltreq | WO | 0: Clear the pause request<br>1: Send a pause request | 0 |
| 30 | resumereq | W1 | 0: Invalid<br>1: Restore the current microprocessor<br>*Note: Write 1 is valid and the hardware is cleared after the microprocessor is recovered* | 0 |
| 29 | Reserved | RO | Reserved | 0 |
| 28 | ackhavereset | W1 | 0: Invalid<br>1: Clear the haverest status bit of the microprocessor | 0 |
| [27:2] | Reserved | RO | Reserved | 0 |
| 1 | ndmreset | RW | 0: Clear reset<br>1: Reset the entire system other than the debug module | 0 |
| 0 | dmactive | RW | 0: Reset debug module<br>1: Debug module works properly | 0 |

**Debug Module Status Register (dmstatus)**

This register is used to indicate the status of the debug module and is a read-only register with the following description of each bit.

Table 6-5 dmstatus register definition

| Bit | Name | Access | Description | Reset Value |
|---|---|---|---|---|
| [31:20] | Reserved | RO | Reserved | 0 |
| 19 | allhavereset | RO | 0: Invalid<br>1: Microprocessor reset | 0 |
| 18 | anyhavereset | RO | 0: Invalid<br>1: Microprocessor reset | 0 |
| 17 | allresumeack | RO | 0: Invalid<br>1: Microprocessor reset | 0 |
| 16 | anyresumeack | RO | 0: Invalid<br>1: Microprocessor reset | 0 |
| [15:14] | Reserved | RO | Reserved | 0 |
| 13 | allavail | RO | 0: Invalid<br>1: Microprocessor is not available | 0 |
| 12 | anyavail | RO | 0: Invalid<br>1: Microprocessor is not available | 0 |
| 11 | allrunning | RO | 0: Invalid<br>1: Microprocessor is running | 0 |
| 10 | anyrunning | RO | 0: Invalid<br>1: Microprocessor is running | 0 |
| 9 | allhalted | RO | 0: Invalid<br>1: Microprocessor is in suspension | 0 |
| 8 | anyhalted | RO | 0: Invalid<br>1: Microprocessor out of suspension | 0 |
| 7 | authenticated | RO | 0: Authentication is required before using the debug module<br>1: The debugging module has been certified | 0x1 |
| [6:4] | Reserved | RO | Reserved | 0 |
| [3:0] | version | RO | Debugging system support architecture version<br>0010: V0.13 | 0x2 |

**Microprocessor Status Register (hartinfo)**

This register is used to provide information about the microprocessor to the debug host and is a read-only register with each bit described as follows.

Table 6-6 hartinfo register definition

| Bit | Name | Access | Description | Reset Value |
|---|---|---|---|---|
| [31:24] | Reserved | RO | Reserved | 0 |
| [23:20] | nscratch | RO | Number of dscratch registers supported | 0x2 |
| [19:17] | Reserved | RO | Reserved | 0 |
| 16 | dataaccess | RO | 0: Data register is mapped to CSR address<br>1: Data register is mapped to memory address | 0x1 |
| [15:12] | datasize | RO | Number of data registers | 0x2 |
| [11:0] | dataaddr | RO | Data register data0 offset address, the base address is 0xe0000000 | 0x0f4 |

**Abstract Command Control and Status Registers (abstractcs)**

This register is used to indicate the execution of the abstract command. The debug host can read this register to know whether the last abstract command is executed or not, and can check whether an error is generated during the execution of the abstract command and the type of the error, which is described in detail as follows.

Table 6-7 abstractcs register definitions

| Bit | Name | Access | Description | Reset Value |
|---|---|---|---|---|
| [31:29] | Reserved | RO | Reserved | 0 |
| [28:24] | progbufsize | RO | Indicates the number of program buffer program cache registers | 0x8 |
| [23:13] | Reserved | RO | Reserved | 0 |
| 12 | busy | RO | 0: No abstract command is executing<br>1: There are abstract commands being executed<br>*Note: After execution, the hardware is cleared.* | 0 |
| 11 | Reserved | RO | Reserved | 0 |
| [10:8] | cmder | RW | Abstract command error type<br>000: No error<br>001: abstract command execution to write to command, abstractcs, abstractauto registers or read and write to data and progbuf registers<br>010: Does not support current abstract command<br>011: Execution of abstract command with exception<br>100: The microprocessor is not suspended or unavailable and cannot execute abstract commands<br>101: Bus error<br>110: Parity bit error during communication<br>111: Other errors<br>*Note: For bit writing 1 is used to clear the zero.* | 0 |
| [7:4] | Reserved | RO | Reserved | 0 |
| [3:0] | datacount | RO | Number of data registers | 0x2 |

**Abstract Command Register(command)**

The debug host can access the GPRs, FPRs, and CSRs registers inside the microprocessor by writing different configuration values in the abstract command registers.

When accessing the registers, the command register bits are defined as follows.

Table 6-8 Definition of command register when accessing registers

| Bit | Name | Access | Description | Reset Value |
|---|---|---|---|---|
| [31:24] | cmdtype | WO | Abstract command type<br>0: Access register<br>1: Quick access (not supported)<br>2: Access to memory (not supported) | 0 |
| 23 | Reserved | WO | Reserved | 0 |
| [22:20] | aarsize | WO | Access register data bit width<br>000: 8-bit<br>001: 16-bit | 0 |

| | | | | |
|---|---|---|---|---|
| | | | 010: 32-bit<br>011: 64-bit (not supported)<br>100: 128-bit (not supported)<br>*Note: When accessing floating-point registers FPRs, only 32-bit access is supported.* | |
| 19 | aarpostincrement | WO | 0: No effect<br>1: Automatically increase the value of regno after accessing the register | 0 |
| 18 | postexec | WO | 0: No effect<br>1：Execute the abstract command and then execute the command in progbuf | 0 |
| 17 | transfer | WO | 0: Do not execute the operation specified by write<br>1: Execute the manipulation specified by write | 0 |
| 16 | write | WO | 0: Copy data from the specified register to data0<br>1: Copy data from data0 register to the specified register | 0 |
| [15:0] | regno | WO | Specify access registers<br>0x0000-0x0fff are CSRs<br>0x1000-0x101f are GPRs<br>0x1020-0x103f are FPRs | 0 |

**Instruction Cache Register (progbufx)**

This register is used to store any instruction, deploy the corresponding operation, including 8, need to pay attention to the last execution needs to be "ebreak" or "c.ebreak".

Table 6-9 progbuf register definition

| Bit | Name | Access | Description | Reset Value |
|---|---|---|---|---|
| [31:0] | progbuf | RW | Instruction encoding for cache operations, which may include compression instructions | 0 |

**Pause Status Register (haltsum0)**

This register is used to indicate whether the microprocessor is suspended or not. Each bit indicates the suspended status of a microprocessor, and when there is only one core, only the lowest bit of this register is used to indicate it.

Table 6-10 haltsum0 register definition

| Bit | Name | Access | Description | Reset Value |
|---|---|---|---|---|
| [31:1] | Reserved | RO | Reserved | 0 |
| 0 | haltsum0 | RO | 0: Microprocessor operates normally<br>1: Microprocessor stop | 0 |

In addition to the above-mentioned registers of the debug module, the debug function also involves some CSR registers, mainly the debug control and status register dcsr and the debug instruction pointer dpc, which are described in detail as follows.

**Debug Control and Status Register (dcsr)**

Table 6-11 dcsr register definition

| Bit | Name | Access | Description | Reset Value |
|---|---|---|---|---|
| [31:28] | xdebugver | DRO | 0000: External debugging is not supported<br>0100: Support standard external debugging<br>1111: External debugging is supported, but does not meet the specification | 0x4 |
| [27:16] | Reserved | DRO | Reserved | |
| 15 | ebreakm | DRW | 0: The ebreak command in machine mode behaves as described in the privilege file<br>1: The ebreak command in machine mode can enter debug mode | 0 |
| [14:13] | Reserved | DRO | Reserved | 0 |
| 12 | ebreaku | DRW | 0: The ebreak command in user mode behaves as described in the privilege file<br>1: The ebreak command in user mode can enter debug mode | 0 |
| 11 | stepie | DRW | 0: Interrupts are disabled under single-step debugging<br>1: Enable interrupts under single-step debugging | 0 |
| 10 | Reserved | DRO | Reserved | 0 |
| 9 | stoptime | DRW | 0: System timer running in Debug mode<br>1: System timer stop in Debug mode | 0 |
| [8:6] | cause | DRO | Reasons for entering debugging<br>001: Entering debugging in the form of ebreak command (priority 3)<br>010: Entering debugging in the form of trigger module (priority 4, the highest)<br>011: Entering debugging in the form of pause request (priority 1)<br>100: debugging in the form of single-step debugging (priority 0, the lowest)<br>101: enter debug mode directly after microprocessor reset (priority 2)<br>Others: Reserved | 0 |
| [5:3] | Reserved | DRO | Reserved | 0 |
| 2 | step | DRW | 0: Turn off single-step debugging<br>1: Enable single-step debugging | 0 |
| [1:0] | prv | DRW | Privilege mode<br>00: User mode<br>01: Supervisor mode (not supported)<br>10: Reserved<br>11: Machine mode<br>*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 Mode Program Pointer (dpc)**

This register is used to store the address of the next instruction to be executed after the microprocessor enters debug mode, and its value is updated with different rules depending on the reason for entering debug. dpc register is described in detail as follows.

Table 6-12 dpc register definitions

| Bit | Name | Access | Description | Reset Value |
|---|---|---|---|---|
| [31:0] | dpc | DRW | Command Address | 0 |

The rules for updating the registers are shown in the following table.

Table 6-13 dpc update rules

| Enter the debugging method | dpc Update rules |
|---|---|
| ebreak | Address of the Ebreak instruction |
| single step | Instruction address of the next instruction of the current instruction |
| trigger module | Temporarily not supported |
| halt request | Address of the next instruction to be executed when entering Debug |

## 6.2 Debug Interface

Different from the standard RISC-V defined JTAG interface, QingKe V2 series microprocessors use 1/2-wire debugging interface, of which the V2A only supports a 1-wire interface, following the WCH debugging interface protocol. The debug interface is responsible for the communication between the debug host and the debug module, and realizes the read/write operation of the debug host to the debug module registers. WCH designed WCH_Link and open source its schematic and program binary files, which can be used for debugging all microprocessors of RISC-V architecture.

Refer to WCH Debug Protocol Manual for specific debug interface protocols.

# Chapter 7 CSR Register List

The RISC-V architecture defines a number of Control and Status Registers (CSRs) for controlling and recording the operating status of the microprocessor. Some of the CSRs have been introduced in the previous section, and this chapter will detail the CSR registers implemented in the QingKe V2 series microprocessors.

## 7.1 CSR Register List

Table 7-1 List of Microprocessor CSR Registers

| Type | Name | CSR Address | Access | Description |
|---|---|---|---|---|
| RISC-V Standard CSR | marchid | 0xF12 | MRO | Architecture number register |
| | mimpid | 0xF13 | MRO | Hardware implementation numbering register |
| | mstatus | 0x300 | MRW | Status register |
| | misa | 0x301 | MRW | Hardware instruction set register |
| | mtvec | 0x305 | MRW | Exception base address register |
| | mscratch | 0x340 | MRW | Machine mode staging register |
| | mepc | 0x341 | MRW | Exception program pointer register |
| | mcause | 0x342 | MRW | Exception cause register |
| | dcsr | 0x7B0 | DRW | Debug control and status registers |
| | dpc | 0x7B1 | DRW | Debug mode program pointer register |
| | dscratch0 | 0x7B2 | DRW | Debug mode staging register 0 |
| | dscratch1 | 0x7B3 | DRW | Debug mode staging register 1 |
| Vendor-defined CSRs | intsyscr | 0x804 | URW | Interrupt system control register |

## 7.2 RISC-V Standard CSR Registers

**Architecture Number Register (marchid)**

This register is a read-only register to indicate the current microprocessor hardware architecture number, which is mainly composed of vendor code, architecture code, series code, and version code. Each of them is defined as follows.

Table 7-2 marchid register definition

| Bit | Name | Access | Description | Reset Value |
|---|---|---|---|---|
| 31 | Reserved | MRO | Reserved | 1 |
| [30:26] | Vender0 | MRO | Manufacturer code 0<br>Fixed to the letter "W" code | 0x17 |
| [25:21] | Vender1 | MRO | Manufacturer code1<br>Fixed to the letter "C" code | 0x03 |
| [20:16] | Vender2 | MRO | Manufacturer code 2<br>Fixed to the letter "H" code | 0x08 |
| 15 | Reserved | MRO | Reserved | 1 |
| [14:10] | Arch | MRO | Architecture Code | 0x16 |

| | | | | |
|---|---|---|---|---|
| | | | RISC-V architecture is fixed to the letter "V" code | |
| [9:5] | Serial | MRO | Series Code<br>QingKe V2 series, fixed to the number "4" | 0x04 |
| [4:0] | Verision | MRO | Version Code<br>Can be the version "A", "B", "C", "F" and other letters of the code | x |

The manufacturer number and version number are alphabetic, and the series number is numeric. The coding table of letters is shown in the following table.

Table 7-3 Alphabetic Mapping Table

| 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 |

For example, QingKe V2A microprocessor, read the value of this register as: 0xDC68D841, which corresponds to WCH-V2A.

**Hardware Implementation Numbering Register (mimpid)**

This register is mainly composed of vendor codes, each of which is defined as follows.

Table 7-4 mimpid register definition

| Bit | Name | Access | Description | Reset Value |
|---|---|---|---|---|
| 31 | Reserved | MRO | Reserved | 1 |
| [30:26] | Vender0 | MRO | Manufacturer code 0<br>Fixed to the letter "W" code | 0x17 |
| [25:21] | Vender1 | MRO | Manufacturer code1<br>Fixed to the letter "C" code | 0x03 |
| [20:16] | Vender2 | MRO | Manufacturer code 2<br>Fixed to the letter "H" code | 0x08 |
| 15 | Reserved | MRO | Reserved | 1 |
| [14:1] | Reserved | MRO | Reserved | 0 |
| 0 | Reserved | MRO | Reserved | 1 |

**Machine Mode Status Register (mstatus)**

This register has been partially described in the previous section, and its folks are positioned as follows.

Table 7-5 mstatus register definition

| Bit | Name | Access | Description | Reset Value |
|---|---|---|---|---|
| [31:25] | Reserved | MRO | Reserved | 0 |
| 24 | MPPOP | MRW | Whether the current subactive interrupt needs to come out of the stack | 0 |
| 23 | MPOP | MRW | Whether the current active interrupt needs to come out of the stack | 0 |
| [22:13] | Reserved | MRO | Reserved | 0 |
| [12:11] | MPP | MRW | Privileged mode before entering break | 0 |
| [10:8] | Reserved | MRO | Reserved | 0 |
| 7 | MPIE | MRW | Interrupt enable state before entering interrupt | 0 |
| [6:4] | Reserved | MRO | Reserved | 0 |
| 3 | MIE | MRW | Machine mode interrupt enable | 0 |

| | | | | |
|---|---|---|---|---|
| [2:0] | Reserved | MRO | Reserved | 0 |

The MPOP field is used to control and indicate that the currently active interrupt needs to be out of the stack when it is 1, and does not need to be out of the stack when it is 0. The MPPOP field is used to control and indicate that the next active interrupt needs to be out of the stack when it is 1, and does not need to be out of the stack when it is 0. Normally, when nesting occurs, MPPOP is updated to MPOP and MPOP is updated to the current interrupt out stack flag. When nesting returns, MPOP is restored to MPPOP.

The MPP field is used to save the privileged mode before entering an exception or interrupt, and to restore the privileged mode after exiting an exception or interrupt, QingKe V2 microprocessor only supports Machine mode, although the MPP update mechanism is like the above process, but the value in V2 may only be 0b11. MIE is the global interrupt enable bit, when entering an exception or interrupt, the value of MPIE is updated to MIE value, it should be noted that it should be noted that in the QingKe V2 series microprocessors, MIE is not updated to 0 at this time before entering the last level of interrupt to ensure that the interrupt nesting in Machine mode continues to be executed. After exiting the exception or interrupt, the microprocessor reverts to the Machine mode saved by the MPP and the MPIE reverts to the value of MIE.

**Hardware Instruction Set Register (misa)**

This register is used to indicate the architecture of the microprocessor and the supported instruction set extensions, each of which is described as follows.

Table 7-6 misa register definition

| Bit | Name | Access | Description | Reset Value |
|---|---|---|---|---|
| [31:30] | MXL | MRO | Machine word length<br>1:32<br>2:64<br>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, QingKe V2 are 32-bit microprocessors, the domain is fixed to 1. Extensions are used to indicate that the microprocessor supports extended instruction set details, each indicates a class of extensions, its detailed description is shown in the following table.

Table 7-7 Instruction Set Extension Details

| 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 |

For example, for QingKe V2A microprocessor, the register value is 0x40800014, which means that the supported instruction set architecture is RV32EC, as well as the non-standard extension X, and there is no User mode implementation.

**Machine Mode Exception Base Address Register (mtvec)**

This register is used to store the base address of the exception or interrupt handler and the lower two bits are used to configure the mode and identification method of the vector table as described in Section 3.2.

**Machine Mode Staging Register (mscratch)**

Table 7-8 mscratch register definitions

| Bit | Name | Access | Description | Reset Value |
|---|---|---|---|---|
| [31:0] | mscratch | MRW | Data storage | 0 |

This register is a 32-bit readable and writable register in Machine mode for temporary data storage. For example, when entering an exception or interrupt handler, the user stack pointer SP is stored in this register and the interrupt stack pointer is assigned to the SP register. After exiting the exception or interrupt, restore the value of user stack pointer SP from mscratch. That is, the interrupt stack and user stack can be isolated.

**Machine Mode Exception Program Pointer Register (mepc)**

Table 7-9 mepc register definitions

| Bit | Name | Access | Description | Reset Value |
|---|---|---|---|---|
| [31:0] | mepc | MRW | Exception procedure pointer | 0 |

This register is used to save the program pointer when entering an exception or interrupt. It is used to save the instruction PC pointer before entering an exception when an exception or interrupt is generated, and mepc is used as the return address when the exception or interrupt is handled and used for exception or interrupt return. However, it is important to note that.

- When an exception occurs, mepc is updated to the PC value of the instruction currently generating the exception.
- When an interrupt occurs, mepc is updated to the PC value of the next instruction.

When you need to return an exception after processing the exception, you should pay attention to modifying

the value of the MEPC, and more details can be found in Chapter 2, Exception.

**Machine Mode Exception Cause Register (mcause)**

Table 7-10 mcause register definition

| Bit | Name | Access | Description | Reset Value |
|---|---|---|---|---|
| 31 | Interrupt | MRW | Interrupt indication field<br>0: Exception<br>1: Interruption | 0 |
| [30:0] | Exception Code | MRW | Exception codes, see Table 2-1 for details | 0 |

This register is mainly used to store the cause of the exception or the interrupt number of the interrupt. Its highest bit is the Interrupt field, which is used to indicate whether the current occurrence is an exception or an interrupt. The lower bit is the exception code, which is used to indicate the specific cause. Its details can be found in Chapter 2, Exception.

**Debug Control and Status Register (dcsr)**

This register is used to control and record the operation status of Debug mode, refer to section 6.1 for detailed description.

**Debug Mode Program Pointer Register (dpc)**

This register is used to store the address of the next instruction to be executed after the microprocessor enters debug mode, and its value is updated with different rules depending on the reason for entering debug. Refer to Section 6.1 for detailed description.

**Debug Mode Staging Register (dscratch0-1)**

This group of registers is used for temporary storage of data in Debug mode.

Table 7-11 dscratch0-1 register definitions

| Bit | Name | Access | Description | Reset Value |
|---|---|---|---|---|
| [31:0] | dscratch | DRW | Debug mode data staging value | 0 |

## 7.3 User-defined CSR Registers

**Interrupt System Control Register (intsyscr)**

This register is mainly used to configure the interrupt nesting depth, HPE, EABI enable and other related functions, see the relevant description in section 3.2.