芯来科技
NUCLEI

Bumblebee Processor Core Instruction Architecture Manual

revise history

versio n number	Revision date	Revised chapter	Revised content
1.0	2019/6/21	N/A	Initial version

table of Contents

List of forms

1. Bumblebee kernel instruction set and CSR introduction

The Bumblebee Processor Core, or Bumblebee core, is a commercial custom made by Nuclei System Technology in conjunction with Gigadevice for its general purpose MCU products for IoT or other ultra-low power scenarios. RISC-V processor core.

For an introduction to the hardware features of the Bumblebee kernel, see the Bumblebee Kernel Concise Data Sheet. This article provides a detailed introduction to the supported instruction architecture.

Note: The Bumblebee core used for this

MCU is jointly developed by Nuclei System Technology and Taiwanese Andes Technology, and Nuclei System Technology provides authorization and technical support services.At present, Nuclei System Technology can license the mass-proven N200 series of ultra-low-power commercial processor cores, as well as research a variety of high-performance embedded processor series, and provide customers with customized services.

1.1. Introduction to the risc-v instruction set

The standard RISC-V instruction set document version followed by the Bumblebee kernel is: "Instruction Set Document Version 2.2" (riscv-spec-v2.2.pdf).Users can register on the RISC-V Foundation website to follow and download the full text (free of charge) (**https://riscv.org/specifications/**)。

In addition to the risc-v ″instruction set document version 2.2″ English original text, users can also refer to the Chinese book ″Teach you to design cpu - risc-v processor articles″ appendix a, appendix c ~ g part, its use of popular The easy-to-understand Chinese systematically explains the risc-v instruction set standard.

The RISC-V instruction set is based on a modular design that can be flexibly combined according to configuration. The Bumblebee kernel supports the following modular instruction set:

- Rv32 architecture: 32-bit address space, general-purpose register width 32 bits.
- i: Supports 32 general integer registers.
- m: support integer multiplication and division instructions
- c: Supports compression instructions with a code length of 16 bits to increase code density.
- a: Support atomic operation instructions.

According to the risc-v architecture naming convention, the combination of the above subset of instructions can be expressed as rv32imac.

1.3. Csr register

Some control and status registers (CSRs) are defined in the RISC-V architecture to configure or record the operational status of some processor cores. The CSR register is a register internal to the processor core and uses its proprietary 12-bit address encoding space.See Chapter 7 for details.

2. Introduction to the Bumblebee kernel privilege architecture

2.1. General introduction

The standard RISC-V privilege schema document version followed by the Bumblebee kernel is: "Privilege Architecture Document Version 1.10" (riscv-privileged-v1.10.pdf).Users can register on the RISC-V Foundation website to follow and download the full text (free of charge) (**https://riscv.org/specifications/**)。

In addition to the risc-v "privileged architecture document version 1.10" English original text, users can also refer to the Chinese book "Hands to teach you design cpu - risc-v processor articles" appendix a, appendix c ~ g part, its use of popular The easy-to-understand Chinese systematically explains the risc-v privilege architecture standard.

2.2. Privilege Modes

The Bumblebee kernel supports two privileged modes (Privilege Modes):

- Machine Mode is a required mode, and the code of this Privilege Mode is 0x3.
- User Mode is a configurable mode, and the code for this Privilege Mode is 0x0.

2.2.1. Machine Mode (Machine Mode)

The key points of the Bumblebee kernel about Machine Mode are as follows:

- After the processor core is reset, it defaults to Machine Mode.
- In Machine Mode, the program has access to all CSR registers.

2.2.2. User Mode

The key points of the Bumblebee kernel about User Mode are as follows:

- Only User Mode qualified CSR registers can be accessed in User Mode, see Section 7.3 for details.

2.2.3. Machine Sub-Mode

The Machine Mode of the Bumblebee kernel may be in four different states, called the machine submode.

(Machine Sub-Mode):

- Normal machine mode (the encoding of the Machine Sub-Mode is 0x0):
  - After the processor core is reset, it is in this submode. If no exception, nmi, or interrupt is generated after the processor is reset, it will continue to operate normally in this mode.
- Exception handling mode (the encoding of the Machine Sub-Mode is 0x2):
  - The processor core is in this state after responding to an exception.
  - See Chapter 3 for details on the exception mechanism.
- NMI processing mode (the encoding of the Machine Sub-Mode is 0x3):
  - The processor core is in this state in response to the nmi.
  - See Chapter 4 for details on the nmi mechanism.
- Interrupt processing mode (the encoding of the Machine Sub-Mode is 0x1):
  - The processor core is in this state after responding to the interrupt.
  - See Chapter 5 for details on the interrupt mechanism.

The Machine Sub-Mode currently in the processor core is reflected in the TYP field of the CSR register msubm, so software can read the currently in the Machine Sub-Mode by reading this CSR register. For details on the msubm register, see Section 7.5.3.

Note: In the RISC-V architecture, entry exceptions, NMIs, or interrupts are also collectively referred to as traps.

2.2.4. Mode view

The key points of the processor mode view are as follows:

- According to the architecture definition of RISC-V, the current Machine Mode or User Mode of the processor is not reflected in any software-visible registers (the processor core maintains a hardware register that is invisible to the software), so the software program cannot read it. Look at any of the registers to see the current Machine Mode or User Mode.
- The four machine sub-modes of the Bumblebee kernel are reflected in the CSR register msubm

In the TYP domain, the software can view the currently in the Machine Sub-Mode by reading this CSR register.

2.2.5. Machine Mode to User Mode Switch

The mret instruction can be executed directly in Machine Mode. Switching from Machine Mode to User Mode is only possible

The execution of the mret instruction occurs. Since the Mode Mode may be in four different states as described in Section 2.2.32.2.3,

Introduced as follows:

- If in normal machine mode, the hardware behavior of executing the mret instruction is the same as executing the mret instruction in exception handling mode, see Section 3.5

设置

for details.

- Therefore, if you want to switch from Machine Mode to User Mode in normal machine mode, you need to modify the value of MTP field in mstatus first, and then execute the mret instruction to achieve the effect of mode switching.A typical program code snippet looks like this:

```
/* Switch Machine sub-mode to User mode */
Li t0, MSTATUS_MPP // The value of MSTATUS_MPP is 0x00001800, which corresponds to the MPP bit field of mstatus.
                         // See Section 7.4.7 for bitfield details for mstatus.
csrc mstatus,  t0       // Clear the MPP bit field of the
    mstatus register to 0 la t0, 1f    // Assign the PC address
    where the previous label 1 is located to t0 csrw mepc, t0 //
    Assign the value of t0 to the CSR register mepc
    mret                 // Executing the mret instruction will switch the mode to User Mode and execute from the
    previous label 1
                         // program (label 1 is the position of the next instruction of mret)
1:                       // position of tag 1
```

- If the hardware behavior of the mret instruction is executed in exception handling mode, see Section 3.5 for details.

  - In general, the mret instruction is used to exit from exception handling mode to the mode before entering the exception.

  - If you explicitly want to exit from Machine Mode to User Mode (or normal machine mode), you need to modify the value of mstatus' MPP field first, and then execute the mret instruction to achieve the mode switch effect.

- If the hardware behavior of the mret instruction is executed in interrupt processing mode, see Section 5.7 for details.

  - In general, the mret instruction is used to exit from interrupt processing mode to the mode before entering the interrupt.

  - If you explicitly want to exit from Machine Mode to User Mode (or normal machine mode), you need to modify the value of mstatus' MPP field first, and then execute the mret instruction to achieve the mode switch effect.

- If the hardware behavior of the mret instruction is executed in NMI processing mode, see Section 4.4 for details.

  - In general, the mret instruction is used to exit from NMI processing mode to normal machine mode.

  - If you explicitly want to exit from Machine Mode to User Mode (or normal machine mode), you need to modify the value of mstatus' MPP field first, and

then execute the mret instruction to achieve the mode switch effect.

note:

- Executing the mret instruction directly in User Mode will generate an Illegal Instruction exception

2.2.6. User Mode to Machine Mode Switch

Switching from the User Mode to the Machine Mode of the Bumblebee kernel can only occur via exceptions, response interrupts, or NMI:

- The response exception enters the exception handling mode. See Section 3.4 for details.
  - Note: The software can force the ecall exception handler by calling the ecall instruction.
- The interrupt processing mode is entered in response to the interrupt. See Section 5.6 for details.
- In response to nmi enter the nmi processing mode. See section 4.3 for details.

2.2.7. Interrupt, exception, nmi nesting

Interrupts and exceptions can nest themselves, and nmi can't nest itself:

- In nmi processing mode, if nmi occurs again, the new nmi will be masked, so nmi cannot self-nesert, see section 4.6 for details.

- In exception handling mode, if an exception occurs again, this is an abnormal nesting situation, see Section 3.73.7 for information.

Details.

- In interrupt processing mode, if an interrupt occurs again, this is an interrupt nesting situation, see Section 5.15.11.

Solve the details.

Interrupts, exceptions, and nmis may also nest with each other, as follows:

- When an exception occurs in the interrupt processing mode, the exception processing mode is entered.
- If an exception occurs in the nmi processing mode, the exception handling mode is entered.
- When nmi occurs in the interrupt processing mode, it enters the nmi processing mode.
- When nmi occurs in the exception handling mode, it enters the nmi processing mode.
- Note: In nmi and exception mode, the default interrupt bit is automatically turned off by the hardware, so it will not respond to the interrupt.

In order to be able to guarantee that the exception and the NMI can be restored to the previous state (Recoverable) after they are nested with each other,

The Bumblebee kernel implements a

″Two Levels of NMI/Exception State

Save Stacks" technique, see Section 4.6 for more details.

2.3. Physical memory protection (pmp)

Since the Bumblebee core is a low-power core for the microcontroller domain, it does not support virtual address management units.
(Memory Management Unit), so all address access operations are physical addresses used. In order to isolate and protect permissions based on different memory physical address ranges and different Privilege Modes, the RISC-V architecture standard defines a

physical memory protection mechanism

(Physical Memory Protection (PMP) unit.

Note: The Bumblebee kernel does not support PMP units.

3. Bumblebee kernel exception mechanism introduction

3.1. Abnormal overview

Exception mechanism, that is, the processor core suddenly encounters an abnormal event in the process of sequentially executing the program instruction stream and aborts execution of the current program, and then processes the exception instead. The main points are as follows:

- The "exceptional thing" that the processor encounters is called an exception.An exception is caused by an internal event in the processor or an event in the execution of the program, such as a hardware failure, a program failure, or the execution of a special system service instruction. In short, it is an internal cause.

- After the exception occurs, the processor enters the exception service handler.

3.2. Abnormal shielding

The exception specified in the risc-v architecture cannot be masked, which means that once an exception occurs, the processor will stop the current operation and enter the exception handling mode.

3.3. Abnormal priority

There may be multiple instances of exceptions in the processor core, so exceptions also have priority. The priority of the exception is asTable 3-1

设置

~~Table 3-1~~ As shown in the figure, the smaller the exception number, the higher the priority of the exception. 设置

3.4. Enter exception handling mode

When entering an exception, the hardware behavior of the Bumblebee kernel can be briefly described as follows. Note that the following hardware behavior is done simultaneously in one clock cycle:

- The execution of the current program flow is stopped and the execution begins with the PC address defined by the CSR register mtvec.
- Update the relevant csr registers, which are the following registers:
  - mcause（Machine Cause Register）
  - mepc（Machine Exception Program Counter）
  - mtval（Machine Trap Value Register ）
  - mstatus（Machine Status Register）
- Update the Privilege Mode of the processor core and the Machine Sub-Mode.

The overall process of abnormal

responseFigure 3-1 Figure 3-1 Shown.

设置

设置

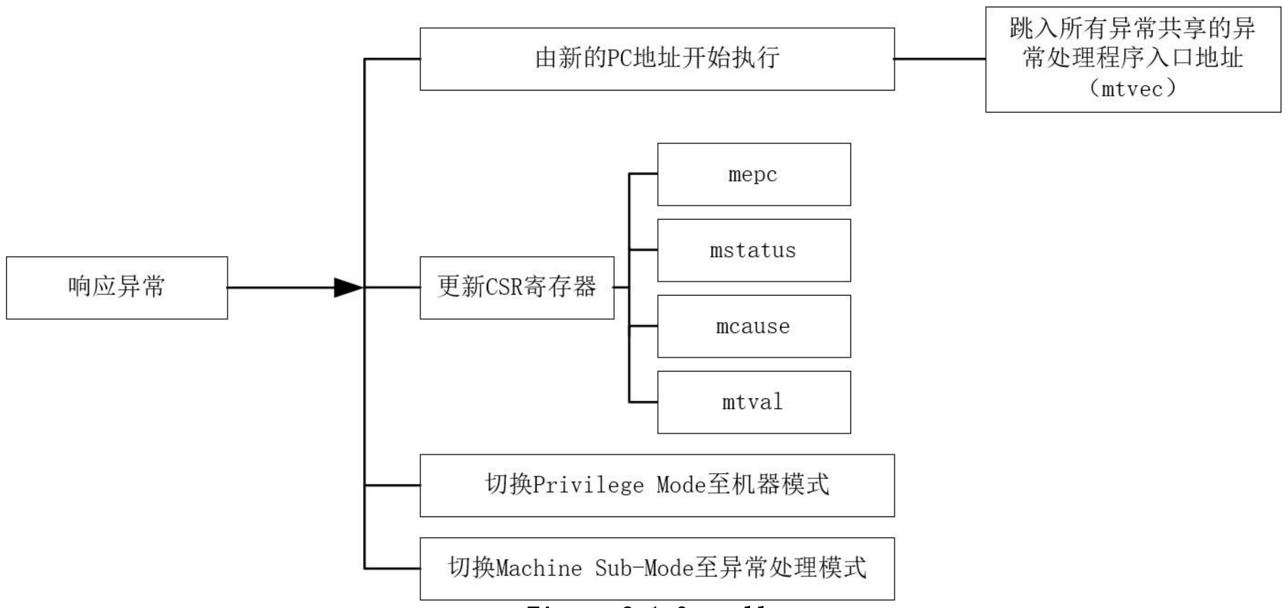

响应异常
由新的PC地址开始执行
跳入所有异常共享的异常处理程序入口地址（mtvec）
更新CSR寄存器
mepc
mstatus
mcause
mtval
切换Privilege Mode至机器模式
切换Machine Sub-Mode至异常处理模式

Figure 3-1 Overall process of abnormal response

These will be detailed below.

3.4.1. Execute from the PC address defined by mtvec

The PC address that the Bumblebee kernel jumps after encountering an exception is specified by the CSR register mtvec.

The mtvec register is a readable

and writable CSR register, so the

software can programmatically change the value. The detailed format of the mtvec register is shown in Table 7-3 in Table 7-3.

3.4.2. Update CSR register mcause

When the Bumblebee kernel enters an exception, the CSR register mcause is updated (hardware automatically) to reflect the current type of exception. The software can read this register to query the specific cause of the exception.

The detailed format of the mcause

register is shown in Table 7-6, Table 7-6, where the lower 5 bits are the exception number field, which is used to indicate various types.

The same type of exception, such asTable 3-1 Table 3-1 Shown.

设置

设置

Table 3-1 Exception Code in the mcause Register

Exception number （**Exception Code**）	Exception and interrupt type	Synchronous Asynchronous	description
0	Instruction address misaligned	Synchronize	The instruction pc address is not aligned. Note: This exception type is configured with the "c" extension It is not possible to have a subset of instructions in the processor.
1	Instruction access error (Instruction access fault）	Synchronize	Fetch instruction fetch error.
2	Illegal instruction (Illegal instruction）	Synchronize	Illegal order.
3	Breakpoint	Synchronize	The RISC-V architecture defines the EBREAK instruction, which occurs when the processor executes the instruction and enters the exception service routine.This instruction is often used by the debugger (Debugger). Break point
4	Read memory address misaligned	Synchronize	The Load instruction fetch address is not aligned. Note: The Bumblebee kernel does not support data memory read and write operations with unaligned addresses, so This exception is raised when the access address is not aligned.
5	Read memory access error (Load access fault）	Inexact asynchronous	Load instruction fetch error.
6	Write memory and amo address are not aligned （Store/AMO address misaligned）	Synchronize	The Store or AMO instruction fetch address is not aligned.Note: The Bumblebee kernel does not support data memory read and write operations with unaligned addresses, so this happens when the access address is not aligned. often.
7	Write memory and amo access error （Store/AMO access fault）	Inexact asynchronous	Store or AMO instruction fetch error.

8	User mode environment call （Environment call from U-mode）	Synchronize	The ecall instruction is executed in User Mode. The RISC-V architecture defines the ecall instruction, which occurs when the processor executes the instruction. Into the exception service program. This instruction is often used by software to force entry into the exception mode.
11	Machine mode environment call （Environment call from M-mode）	Synchronize	Execute the ecall command in Machine Mode. The RISC-V architecture defines an ecall instruction that, when executed by the processor, will cause an exception to enter the exception service routine. This instruction is often used by software Use, forcibly enter the abnormal mode.

3.4.3. Update CSR register mepc

The return address of the Bumblebee kernel when exiting the exception is determined by the CSR register mepc (Machine Exception Program Counter) Save. When an exception is entered, the hardware will automatically update the value of the mepc register, which will be used as the return address for the exit exception. After the exception is over, it can use its saved PC value to return to the program point that was previously stopped.

note:

- When an exception occurs, the value of the exception return address mepc is updated to the instruction PC where the exception occurred.
- Although the mepc register is automatically updated by hardware when an exception occurs, the mepc register itself is a readable and writable register, so software can also write to it directly to modify its value.

3.4.4. Update CSR register mtval

When the Bumblebee kernel enters an exception, the hardware will automatically update the CSR register mtval (Machine Trap Value Register) to reflect the memory access address or instruction encoding that caused the current exception:

- If an exception is caused by a memory access, such as an exception caused by a hardware breakpoint, instruction fetch, or memory read and write, the address of the memory access is updated to the mtval register.

- If the exception is caused by an illegal instruction, the instruction code of the instruction is updated to the mtval register.

3.4.5. Update the CSR register mstatus

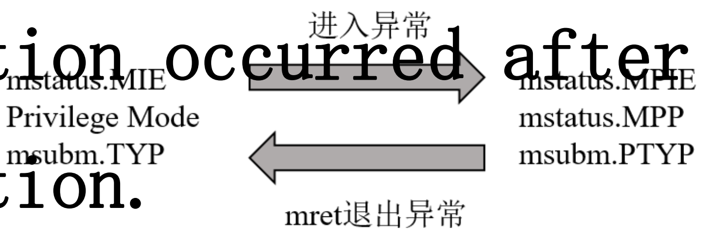
The detailed format of the mstatus register is shown in Table 7-2, Table 7-2. When the Bumblebee kernel enters an exception, the hardware will automatically update some fields of the CSR register mstatus (Machine Status Register):

- The value of the mstatus.MPIE field is updated to the value of the mstatus.MIE field before the exception occurred, as described in Sections 8.2 and 8.2. 设置

  Shown. The role of the mstatus.MPIE field is to use the value of mstatus.MPIE to recover the mstatus.MIE value before the exception occurred after the exception.

- The value of the mstatus.MIE field is updated to 0 (meaning that the interrupt is globally closed after entering the exception service routine, all interrupts will be masked and not responding).
- The value of the mstatus.MPP field is updated to the Privilege Mode before the exception occurred, as described in Sections 8.2 and 8.2.

Show.The role of the mstatus.MPP field is to use the value of mstatus.MPP to recover the Privilege Mode before the exception occurred after the exception.



3.4.6. Update Privilege Mode

Exceptions need to be handled in Machine Mode. When entering an exception, the processor kernel's Privilege Mode

Updated to machine mode.

3.4.7. Update Machine Sub-Mode

The Machine Sub-Mode of the Bumblebee kernel is reflected in real time in the msubm.TYP field of the CSR register. When an exception is entered, the Machine Sub-Mode of the processor core is updated to the exception handling mode, so:

- The value of the msubm.PTYP field of the CSR register is updated to the Machine Sub-Mode before the exception occurred.

  (The value of the msubm.TYP field), as described in Sections 8.2 and 8.2. The role of the msubm.PTYP domain is in the exception

  设置

  After the end, you can use the value of msubm.PTYP to recover the Machine Sub-Mode value before the exception occurred.

- The value of the msubm.TYP field of the CSR register is updated to "Exception Handling Mode" as described in Sections 8.2 and 8.2. 设置

It is shown that the current mode is already in the "abnormal processing mode".

Figure 3-2 Changes to the csr register when entering/exiting an exception

3.5. Exit exception handling mode

After the program completes the exception handling, it eventually needs to exit from the exception service.

Since the exception handling is in Machine Mode, the software must use the mret instruction when exiting the exception. The hardware behavior of the processor after executing the mret instruction is as follows.Note that the following hardware behavior is done simultaneously in one clock cycle:

- The execution of the current program flow is stopped and the execution begins with the PC address defined by the CSR register mepc.
- Update the CSR register mstatus (Machine Status Register) as shown in Sections 8.2 and 8.2, and

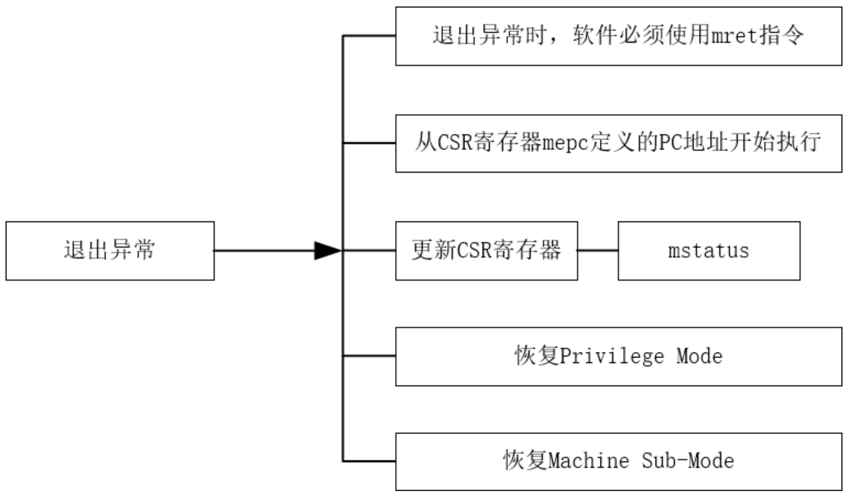
  Update the Privilege Mode of the processor core and the Machine Sub-Mode.

The overall process of exiting an anomalyFigure 3-3 Figure 3-3 Show.

设置

设置

设置



退出异常
退出异常时，软件必须使用mret指令
从CSR寄存器mepc定义的PC地址开始执行
更新CSR寄存器
mstatus
恢复Privilege Mode
恢复Machine Sub-Mode

Figure 3-3 Exiting the abnormal overall process

These will be detailed below.

3.5.1. Execute from the PC address defined by mepc

When an exception is entered, the mepc register is updated simultaneously to reflect the instruction PC value at which the exception was encountered. Through this mechanism, it means that after the execution of the mret instruction, the processor returns to the PC address of the instruction that encountered the exception at that time, so that the program stream that was previously aborted can be executed.

Note: You may need to use software to update the value of mepc before exiting the exception. For example, if an exception is generated by ecall or ebreak, the value of mepc is updated

to ecall or ebreak to command its own PC.So if the exception returns, if you use it directly
The mepc saved PC value as the return address.will jump back to the ecall or ebreak instruction, causing an infinite loop (execution)
The ecall or ebreak instruction causes the exception to be re-entered.The correct way is to change the mepc to the next instruction in the exception handler. Since ecall/ebreak is now a 4-byte instruction, rewrite the setting mepc=mepc+4.

3.5.2. Update the CSR register mstatus

The detailed format of the mstatus register is shown in Table 7-2 in Table 7-2. The hardware will automatically update the CSR after executing the mret instruction

Some fields of the register mstatus:

- The value of the mstatus.MIE field is restored to the value of the current mstatus.MPIE.
- The value of the current mstatus.MPIE field is updated to 1.
- The updated values of the mstatus.MPP domain are divided into the following two cases:
  - When user mode U-mode is configured, mstatus.MPP is updated to 0x0.
  - When user mode U-mode is not configured, mstatus.MPP is updated to 0x11.

When entering an exception, the value of mstatus.MPIE was updated to the mstatus.MIE value before the exception occurred, such as the section and the first

Section 8.2 and Section 8.2. After the

mret instruction is executed, the value of the mstatus.MIE field is restored to the value of mstatus.MPIE.

设置

Through this mechanism, it means that after the mret instruction is executed, the mstatus.MIE value of the processor is restored to the value before the exception occurred.

(Assuming the previous mstatus.MIE value is 1, it means the interrupt was re-opened globally).

3.5.3. Update Privilege Mode

When entering an exception, the value of mstatus.MPP was updated to the Privilege Mode before the exception occurred, while the mret was executed.

After the instruction, the processor's Privilege Mode is restored to the value of mstatus.MPP, as described in Sections 8.2 and 8.2. 设置

Through this mechanism, the processor is guaranteed to return to the Privilege Mode of the processor before the exception occurred.

3.5.4. Update Machine Sub-Mode

The Machine Sub-Mode of the Bumblebee kernel is reflected in real time in the msubm.TYP field of the CSR register. Executing

After the mret instruction, the hardware will automatically restore the processor's Machine Sub-Mode to the value of the msubm.PTYP field:

- When an exception is entered, the value of the msubm.PTYP field is updated to the Machine Sub-Mode value before the exception occurred.After exiting the exception with the mret instruction, the hardware restores the value of the processor Machine Sub-Mode to

The value of the msubm.PTYP field is shown in Figure 4-2 Figure 4-2.Through this mechanism, it means that after exiting the exception, the processor's Machine Sub-Mode is restored to the Machine Sub-Mode before the exception occurred.

3.6. Exception service program

When the processor enters an exception, it starts executing a new program from the PC address defined by the mtvec register. The program is usually an exception service program, and the program can also decide to jump further by querying the exception number in the mcause (Exception Code). Specific exception service procedures.For example, when the value in the program query mcause is 0x2, it is known that the exception is an illegal instruction

error.
(Illegal Instruction), so you can jump
further to the illegal instruction
error exception service subroutine.

Note: Since there is no hardware to automatically save and restore the context in the entry exception and exit exception mechanism, the software needs to explicitly use the instructions (written in assembly language) for context saving and recovery.Please understand this in conjunction with a complete exception service code example for the mcu chip.

3.7. Abnormal nesting

The Bumblebee kernel supports two levels of NMI/Exception State Save Stacks. See Section 4.6 for more details.

4. Introduction to the Bumblebee kernel NMI mechanism

4.1. Nmi overview

NMI (Non-Maskable Interrupt) is a special input signal of the processor core, often used to indicate system-level emergency errors (such as external hardware failures, etc.).After encountering the NMI, the processor core should immediately abort execution of the current program and instead process the NMI error.

4.2. Nmi shielding

The NMI in the Bumblebee kernel cannot be masked, which means that once the NMI occurs, the processor will stop the current operation and process the NMI.

4.3. Enter nmi processing mode

When entering NMI processing mode, the hardware behavior of the Bumblebee kernel can be briefly described as follows.Note that the following hardware behavior is done simultaneously in one clock cycle:

- The execution of the current program flow is stopped and the execution begins with the PC address defined by the CSR register mnvec.
- Update the relevant csr registers, which are the following registers:

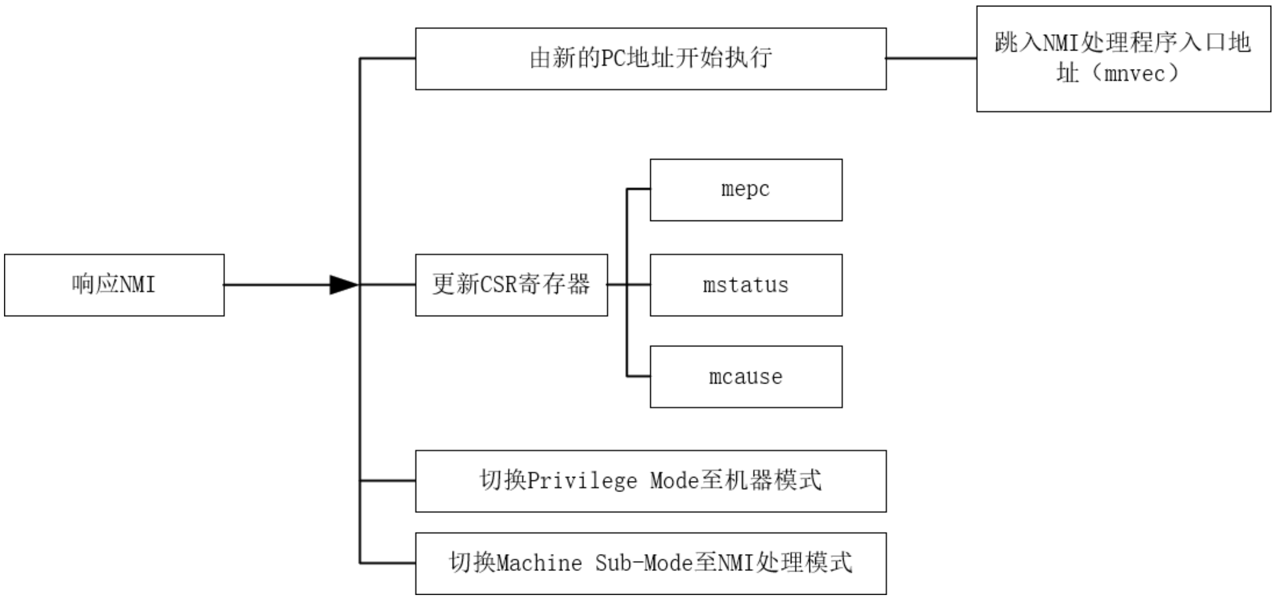
- mepc（Machine Exception Program Counter ）
- mstatus（Machine Status Register）
- mcause（Machine Cause Register）

■ Update the Privilege Mode of the processor core and the Machine Sub-Mode.

The overall process of nmi response is shown in Figure 4-1 Figure 4-1.

设置

设置



响应NMI
由新的PC地址开始执行
跳入NMI处理程序入口地址（mnvec）
更新CSR寄存器
mepc
mstatus
mcause
切换Privilege Mode至机器模式
切换Machine Sub-Mode至NMI处理模式

Figure 4-1 The overall process of nmi response

These will be detailed below.

4.3.1. Execute from the PC address defined by mnvec

The PC address that the Bumblebee kernel jumps into after encountering the NMI is specified by the CSR register mnvec. The values of the mnvec register are as follows:

- When mmisc_ctl[9]=1, the value of the mnvec register is equal to mtvec, ie the NMI has the same Trap as the exception.

  Entrance address.

- When mmisc_ctl[9]=0, the value of the mnvec register is equal to reset_vector, and the reset_vector is the processor

  The value of the PC after reset.

4.3.2. Update CSR register mepc

The return address of the Bumblebee kernel when exiting the NMI is determined by the CSR register mepc (Machine Exception Program

Counter) Save.Upon entering the NMI, the hardware will automatically update the value of the mepc register, which will be used as the exit NMI
The return address, after the end

of nmi, can use its saved pc

value to return to the program

point that was previously

stopped.note:

- When the NMI occurs, the NMI return address mepc is pointed to the next instruction that has not yet been executed (because the instruction at NMI has been executed correctly). Then, after exiting the NMI, the program will return to the previous program point and re-execute from the next instruction.
- Although the mepc register is automatically updated by hardware when the NMI occurs, the mepc register itself is a readable and writable register, so the software can also write this register directly to modify its value.

4.3.3. Update CSR register mcause

The detailed format of the mcause register is shown in Table 7-6, Table 7-6. When the Bumblebee kernel enters the NMI, the hardware automatically saves the current trap ID to mcause to indicate the cause of the trap. Interrupts, exceptions, and NMIs have their own special Trap IDs.

The NMI Trap ID has the following two values:

- When mmisc_ctl[9]=1, the NMI Trap ID is 0xfff.

- When mmisc_ctl[9]=0, the NMI Trap ID is 0x1.

By assigning a specific Trap ID to each Trap, you can identify the cause of the Trap. The software can design a specific handler to process the Trap based on the cause of the Trap.

4.3.4. Update the CSR register mstatus

The detailed format of the mstatus register is shown in Table 7-2, Table 7-2. When the Bumblebee kernel enters the NMI, the hardware will automatically update some fields of the CSR register mstatus:

- The value of the mstatus.MPIE field is updated to the value of the mstatus.MIE field before the NMI occurs, such asFigure 4-2 Figure 4-2 Shown.

  The role of the mstatus.MPIE field is to recover the NMI using the value of mstatus.MPIE after the end

of the NMI.
The mstatus.MIE value before the occurrence.

- The value of the mstatus.MIE field is updated to 0 (meaning that the interrupt is globally closed after entering the NMI server and all interrupts will be masked and not responding).
- The value of the mstatus.MPP field is updated to the Privilege Mode before the NMI occurs.When the role of the mstatus.MPP field is saved, after the NMI is finished, the value of mstatus.MPP can be used to recover the Privilege Mode before the NMI occurs.

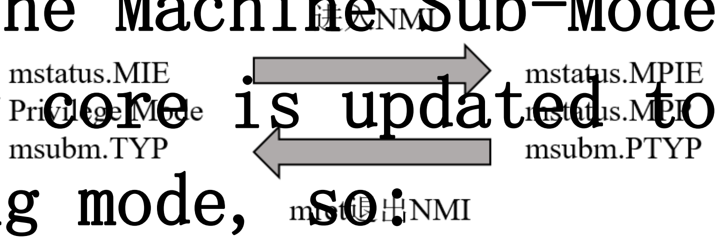
4.3.5. Update Privilege Mode

NMI processing is done in Machine Mode, so when entering NMI, the processor kernel's privileged mode (Privilege Mode) switches to machine mode.

4.3.6. Update Machine Sub-Mode

The Machine Sub-Mode of the Bumblebee kernel is reflected in real time in the msubm.TYP field of the CSR register. Entering

At NMI, the Machine Sub-Mode of the processor core is updated to NMI processing mode, so:



- The value of the msubm.PTYP field of the CSR register is updated to the Machine Sub-Mode before the NMI occurs.

(the value of the msubm.TYP field), such asFigure 4-2 Figure 4-2 Shown. The role of the msubm.PTYP domain is at the end of the NMI After that, you can use the value of msubm.PTYP to recover the Machine Sub-Mode value before the NMI occurred.

- The value of the msubm.TYP field of the CSR register is updated to "NMI processing mode" such asFigure 4-2 Figure 4-2 As shown

The real-time reflection of the

current mode is already ″nmi processing mode″.

设置

Figure 4-2 Changes to the csr register when entering/exiting nmi

4.4. Exit nmi processing mode

When the program finishes nmi processing, it finally needs to exit from the nmi server and return to the main program.

Since the NMI processing is in Machine Mode, the software must use the mret instruction when exiting the NMI. The hardware behavior of the processor after executing the mret instruction is as follows.Note that the following hardware behavior is done simultaneously in one clock cycle:

- The execution of the current program flow is stopped and the execution begins with the PC address defined by the CSR register mepc.
- Update the CSR register mstatus.
- Update Privilege Mode and Machine Sub-Mode.

The overall process of exiting nmi is as followsFigure 4-3 Figure 4-3 所示。

设置

设置

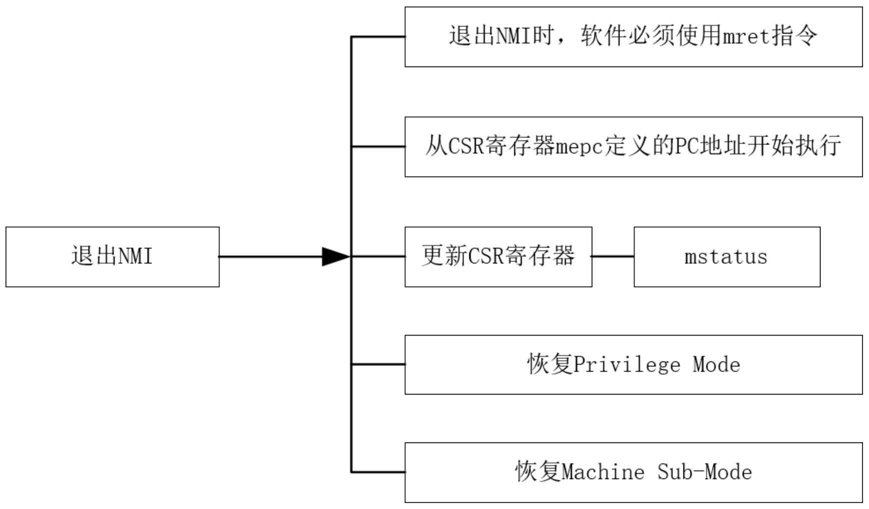

退出NMI
退出NMI时，软件必须使用mret指令
从CSR寄存器mepc定义的PC地址开始执行
更新CSR寄存器
mstatus
恢复Privilege Mode
恢复Machine Sub-Mode

Figure 4-3 Exiting the overall process of nmi

These will be detailed below.

4.4.1. Execute from the PC address defined by mepc

Upon entering the NMI, the mepc register is updated simultaneously to reflect the PC value of the next instruction that encountered the NMI at that time. Through this mechanism, it means that after the execution of the mret instruction, the processor returns to the PC address of the next instruction that encountered the NMI at that time, so that the program stream that was previously aborted can be executed.

4.4.2. Update the CSR register mstatus

The detailed format of the mstatus register is shown in Table 7-2, Table 7-2. After executing the mret instruction, the hardware will automatically update the CSR.

Register mstatus some fields:

- The value of the mstatus.MIE field is restored to the value of the current mstatus.MPIE.
- The value of the mstatus.MPIE field is updated to 1.
- The updated values of the mstatus.MPP domain are divided into the following two cases:
  - When user mode U-mode is configured, mstatus.MPP is updated to 0x0.

- When user mode U-mode is not configured, mstatus.MPP is updated to 0x11.

Upon entering the NMI, the value of mstatus.MPIE was updated to the mstatus.MIE value before the NMI occurred. After the mret instruction is executed, the value of mstatus.MIE is restored to the value of mstatus.MPIE, such asFigure 4-2 Figure 4-2 Shown. Through this mechanism, it means that after the mret instruction is executed, the processor's mstatus.MIE value is restored to the value before the

NMI occurred (assuming the previous mstatus.MIE value is 1, it means that the interrupt is re-opened globally).

4.4.3. Update Privilege Mode

When entering the NMI, the value of mstatus.MPP was updated to the Privilege Mode before the NMI occurred, while it was executing.

After the mret instruction, the processor's Privilege Mode is restored to the value of mstatus.MPP, such asFigure 4-2 Figure 4-2 Shown. This mechanism ensures that the processor returns to the Privilege Mode of the processor before the NMI occurred.

4.4.4. Update Machine Sub-Mode

The Machine Sub-Mode of the Bumblebee kernel is reflected in real time in the msubm.TYP field of the CSR register. Executing

After the mret instruction, the hardware will automatically restore the processor's Machine Sub-Mode to the value of the msubm.PTYP field:

- When entering the NMI, the value of the msubm.PTYP field was updated to the Machine Sub-Mode before the NMI occurred.

  value. After exiting the NMI with

the mret instruction, the hardware restores the value of the processor Machine Sub-Mode to The value of the msubm.PTYP field, such asFigure 4-2 Figure 4-2 Shown. This mechanism means that after exiting the NMI, the processor's Machine Sub-Mode is restored to the Machine Sub-Mode before the NMI occurred.

4.5. Nmi service program

When the processor enters the NMI, it begins to execute the new program from the PC address defined by the mnvec register, which is usually

Nmi service program.

Note: Since there is no hardware to automatically save and restore the context in the nmi and exit nmi mechanisms, the software needs to explicitly use the instructions (written in assembly language) for context saving and recovery.Please combine a complete mcu chip

The nmi server code example understands it.

The Bumblebee kernel is customized likeFigure 4-4 Figure 4-4 Two Levels of NMI/Exception State Save Stacks, which can save up to three levels of NMI/Exception processor state, can achieve secondary recoverable NMI/Exception nesting.

Note: Because the NMI's response is masked on the hardware when the processor is in the NMI state, the NMI cannot self-nesert. The Nmir/Exception Nesting of the

Bumblebee kernel supports only the following three nestings:

- Nmi nesting exception
- Exception nesting exception
- Abnormal nesting nmi

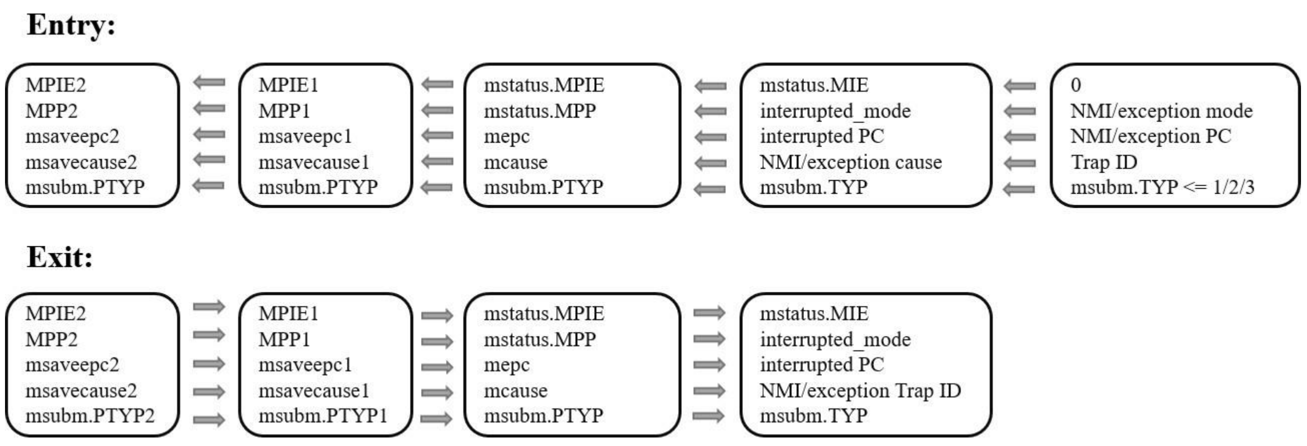


Figure 4-4 Schematic diagram of the two-stage NMI/Exception state stack mechanism of the Bumblebee kernel

4.6.1. Enter nmi/exception nesting

In response to NMI and exceptions, the hardware behavior of the Bumblebee kernel is asFigure 4-4 Figure 4-4 As shown, it can be briefly described as follows.

- Stop executing the current program flow and start executing from the new pc address.

    - If the response is an exception, it is executed from the PC address stored by mtvec.

- If the response is NMI, it starts from the PC address stored by mnvec.

- Update the relevant csr registers, which are the following registers and their associated fields:

  - Mepc: Record the PC before the current NMI/exception, and recover the NMI/PC before the exception occurs from the mepc when exiting the NMI/Exception.

  - Msaveepc1: The first-level NMI/exception state stack records the PC before the first-level nested NMI/exception (NMI/Exception nested by the current NMI/Exception), that is, the mepc value before the current NMI/Exception occurs, and exits. The value of mepc can be recovered from msaveepc1 when NMI/Exception.

  - Msaveepc2: The second-level NMI/Exception State Stack records the second-level nested NMI/Exception (is nested by the first level NMI/Exception Nested NMI/Exception) The PC before the occurrence, that is, the current NMI/msaveepc1 before the occurrence of the abnormality The value of msaveepc1 can be recovered from msaveepc2 when exiting NMI/Exception.

  - mstatus:

- Mpie: Record the current mie before the nmi/abnormal occurrence.
- MPP: Record the Privilege Mode before the current NMI/Exception occurred.

- msavestatus:
  - Mpie1: The first-level nmi/abnormal state stack records the mie before the first-level nested nmi/abnormality, that is, the mpie before the current nmi/abnormality, and the value of mpie can be restored from mpie1 when exiting the nmi/exception.
  - Mpie2: The second-level nmi/abnormal state stack records the mie before the second-level nested nmi/abnormality, that is, the mpi1 before the current nmi/exception occurs, and the value of mpie1 can be recovered from mpie2 when exiting nmi/anomaly.
  - MPP1: The first-level NMI/abnormal state stack records the first-level nested NMI/Privilege Mode before the occurrence of the exception, that is, the MPP before the current NMI/exception occurs, and can be recovered from the MPP1 when the NMI/Exception is exited.

    The value of mpp.
  - MPP2: The second-level NMI/Exception State Stack records the second-level nested NMI/Privilege Mode before the occurrence of the exception, that is, the current NMI/MPP1 before the exception occurs, and the MMI2 can be recovered when the NMI/Exception is exited.

    The value of mpp1.

- Mcause: Records the cause of the current NMI/Exception.
- Msavecause1: The first-level NMI/Exception Status Stack records the first-level nested NMI/Exception Cause.
- Msavecause2: The second-level NMI/Exception Status Stack records the second-level nested NMI/Exception Cause.

- msubm:
  - TYP: Records the current NMI/exception Trap type.
  - PTYP: records the type of trap that the processor is in before the current NMI/exception occurs.
  - PTYP1: The first-level NMI/abnormal state stack records the Machine Sub Mode before the first-level nested NMI/abnormality, that is, the current NMI/PTYP before the abnormality occurs. When the NMI/Exception is exited, the value of PTYP can be restored from PTYP1.
  - PTYP2: The second-level NMI/abnormal state stack records the second-level nested NMI/Machine Sub Mode before the exception occurs, that is, the current NMI/PTYP1 before the abnormality occurs, and the value of PTYP1 can be restored from the PTYP2 when the NMI/abnormal is exited.

- NMI/Exception Handling is done in Machine Mode, so when entering NMI/Exception, the Privilege Mode of the processor core switches to Machine mode.

4.6.2. Exit nmi/exception nesting

After the program completes the NMI/Exception handling, it finally needs to exit from the NMI/Exception Service program and return to the superior NMI/Exception or the main program. Before exiting, it is necessary to restore the processor state from the relevant registers. This is done by the mret instruction, and the processor executes. The hardware behavior after the mret instruction is as followsFigure 4-4 Figure 4-4Can be briefly described as follows.

- The execution of the current program flow is stopped and the execution begins with the PC address defined by the CSR register mepc.
- Update the relevant csr registers, which are the following registers and their associated fields:

- Mepc(Machine Exception Program Counter): Reverts to the PC stored in the first level of nested NMI/Exception in msaveepc1.
- Msaveepc1: first-level NMI/exception state stack, mret occurs from second-level NMI/exception state stack

  Msaveepc2 restores the register msaveepc1 value, which is restored to the second level of nesting stored in msaveepc2

Nm / pc before the occurrence of an abnormality.

- mstatus（Machine Status Register）
  - Mpie: Reverts to mie before the first-level nested nmi/exception stored in mpie1.
  - MPP: Reverts to the first level of nested NMI/Privilege Mode before the exception occurred in MPP1.

- msavestatus:
  - MPIE1: First level NMI/Exception state stack, mret occurs from second level NMI/Exception state stack

    MPIE2 restores the value of the register field msavestatus.MPIE1, which is restored to the MIE before the second-level nested NMI/exception stored in MPIE2.
  - MPP1: first-level NMI/exception state stack, mret occurs from second-level NMI/exception state stack

MPP2 restores the value of the register field msavestatus.MPP1, which is restored to the Privilege Mode before the second-level nested NMI/exception stored in MPP2.

- Mcause(Machine Cause Register): Revert to the first level of nesting stored in msavecause1

The cause of nmi/abnormality.

- Msavecause1: first-level NMI/exception state stack, mret occurs from second-level NMI/exception state stack

Msavecause2 restores the value of register msavecause1, which is the reason for reverting to the second-level nested

NMI/exception stored in msavecause2.

- msubm（Machine Sub-Mode Register）
  - ◆ TYP: Reverts to the Trap type of the processor before the current NMI/Exception occurred in msubm.PTYP.
  - ◆ PTYP: Revert to the first-level nested NMI/Exception occurrence pre-processor stored in msubm.PTYP1

    Trap type.
  - ◆ PTYP1: First level NMI/Exception Status Stack, mret occurs from the second level NMI/Exception Status Stack

    The value of the PTYP2 recovery register field msubm.PTYP1 is restored to the Trap type of the second-level nested NMI/pre-existing processor stored in msubm.PTYP2.

- Update the processor's Privilege Mode based on the value of the mstatus.MPP field.

5. Bumblebee kernel interrupt mechanism introduction

5.1. Interrupt overview

Interrupt mechanism, that is, the processor core is suddenly interrupted by other requests during the execution of the program instruction flow, and the execution of the current program is aborted, and then other things are processed, after waiting for other things to be processed, and then Go back to the point where the program was interrupted before continuing to execute the previous program instruction stream.

Some basic points of knowledge of the

interruption are as follows:

- The "other request" interrupted by the processor is called the Interrupt Request. The source of the "other request" is called the interrupt source. The interrupt source usually comes from outside the kernel. It can also be internal to the core (becoming an internal interrupt source).
- The "other thing" that the processor goes to handle is called the Interrupt Service Routine (ISR).
- Interrupt processing is a normal mechanism, not an error situation.After the processor receives the interrupt request, it needs to save the scene of the current program, which is referred to as “save site”.After processing the interrupt service routine, the processor needs to restore the previous site, thereby continuing to execute the previously interrupted program, referred to as "recovery site."
- There may be multiple interrupt sources that simultaneously initiate requests to the processor, and these interrupt sources need to be arbitrated to select which interrupt source is prioritized.This situation is called "interrupt arbitration", and different interrupts can be assigned levels and priorities to facilitate arbitration, so there is a concept of "interrupt level" and "interrupt priority".

5.2. Interrupt controller eclic

As described in Section 7.4.13, the Bumblebee kernel supports "default interrupt mode" and "ECLIC interrupt mode" through different configurations of the software. It is recommended to use ″ECLIC interrupt mode″. This article only introduces ″ECLIC interrupt mode″.

The Bumblebee kernel implements an ″Improved Core Local Interrupt (Enhanced Core Local Interrupt) Controller, ECLIC) can be used to manage multiple interrupt sources. All types of interrupts in the Bumblebee kernel (except for debug interrupts) are managed by ECLIC. For details on ECLIC, see Section 6.2. About Bumblebee Kernel See Section 5.3 for an introduction to all supported interrupt types.

5.3. Interrupt type

The types of interrupts supported by the Bumblebee kernel are asFigure 5-1 Figure 5-1 Shown in .

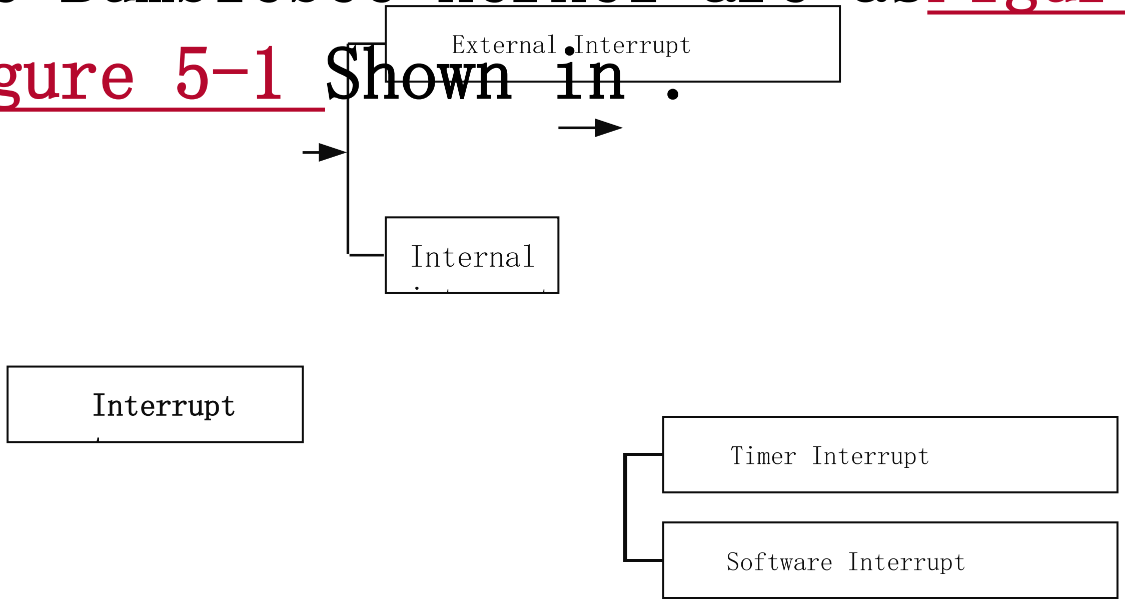


Figure 5-1 Schematic diagram of the interrupt type

These will be detailed below.

5.3.1. External Interrupt

An external interrupt is an interrupt

from outside the processor core. External interrupts allow users to connect to external interrupt sources, such as external devices

Interrupts generated by uart, gpio, etc

Note: The Bumblebee core supports multiple external interrupt sources, all of which are managed by ECLIC.

5.3.2. Internal interrupt

The Bumblebee kernel has several kernel-specific internal interrupts, namely:

- Software Interrupt
- Timer Interrupt

Note: The internal interrupts of the Bumblebee kernel are also managed by ECLIC.

5.3.2.1 The main points of the software interrupt software interruption are as follows:

- The Bumblebee kernel implements a TIMER unit, and a msip register is defined in the TIMER unit to generate software interrupts. See Section 6.1.6 for details.
- Note: Software interrupts are also managed by eclic.

5.3.2.2 The main points of the timer interrupt timer interrupt are as follows:

- The Bumblebee kernel implements a TIMER unit, and a timer is defined in the TIMER unit to generate a timer interrupt. See Section 6.1.5 for details.
- Note: Timer interrupts are also managed by eclic.

5.3.2.3 Memory access error interrupt

The main points of the "memory access error exception" conversion are as follows:

- When the Bumblebee kernel encounters a "memory access error exception", it does not generate an exception, but is instead converted into a corresponding internal interrupt, treated as an interrupt.

5.4. Interrupt mask

5.4.1. Interrupt global mask

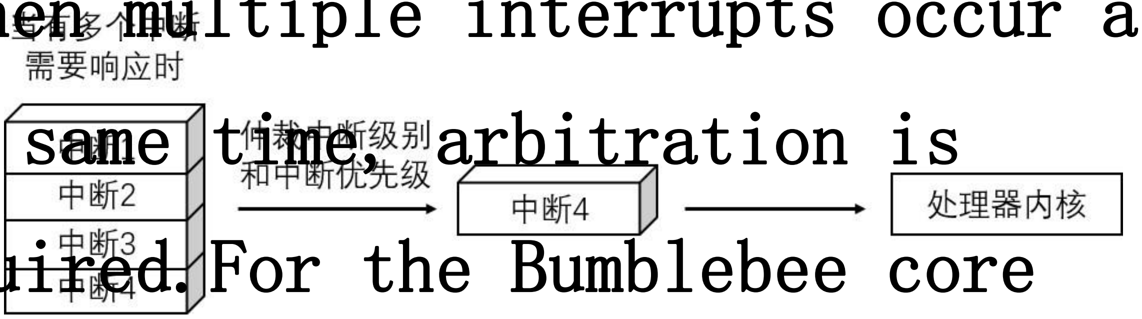
The Bumblebee kernel interrupt can be masked, and the MIE field of the CSR register mstatus controls the global enable of the interrupt. See Section 7.4.8 for details.

5.4.2. Interrupt source separately shielded

For different interrupt sources, eclic assigns its own interrupt enable register to each interrupt source. The user can configure the eclic registers to manage the masking of each interrupt source. See Section 6.2.6 for details.

5.5. Interrupt level, priority and arbitration

When multiple interrupts occur at the same time, arbitration is required. For the Bumblebee core processor, ECLIC manages all interrupts in a unified manner. ECLIC assigns its own interrupt level and priority register to each interrupt source, which can be configured by the user.



The eclic register manages the level and priority of each interrupt source.
When multiple interrupts occur

simultaneously, eclic arbitrates the level.

And the highest priority interrupt, such asFigure 5-2 Figure 5-2 Shown in . See Section 6.2.9 for details.

设置

设置

Figure 5-2 Schematic diagram of interrupt arbitration

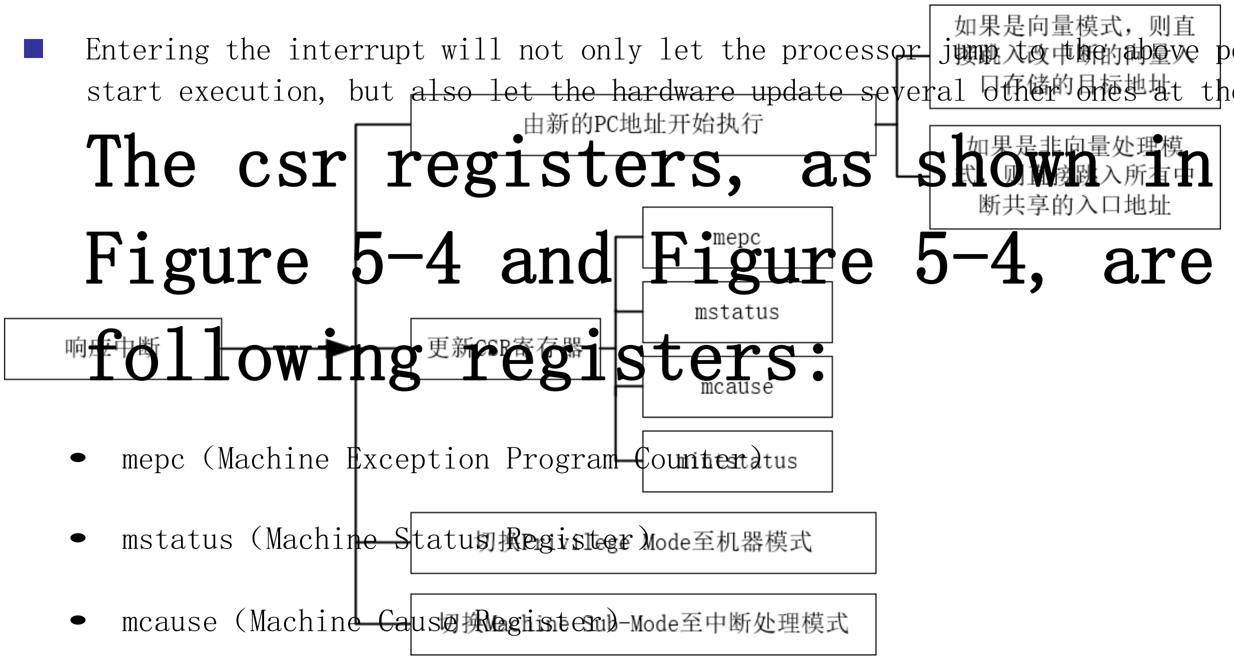
5.6. Enter interrupt processing mode

The hardware behavior of the Bumblebee kernel can be briefly described as follows in response to an interrupt. Note that the following

hardware behavior is done simultaneously in one clock cycle:

- Stop executing the current program flow and start executing from the new pc address.
- Entering the interrupt will not only let the processor jump to the above pc address to start execution, but also let the hardware update several other ones at the same time.

The csr registers, as shown in Figure 5-4 and Figure 5-4, are the following registers:


响应中断
更新CSR寄存器
由新的PC地址开始执行
如果是向量模式，则直接跳入该中断的向量入口存储的目标地址
如果是非向量处理模式，则直接跳入所有中断共享的入口地址
mepc
mstatus
mcause
mintstatus
切换Privilege Mode至机器模式
切换Machine Sub-Mode至中断处理模式

- mepc（Machine Exception Program Counter）
- mstatus（Machine Status Register）
- mcause（Machine Cause Register）
- mintstatus （Machine Interrupt Status Register）

- In addition to this, entering the interrupt also updates the Privilege Mode of the processor core and the Machine Sub-Mode.
- The overall processFigure 5-3 Figure 5-3 Shown in .

Figure 5-3 Overall response to the interrupt

These will be detailed below.

5.6.1. Execute from the new pc address

Each interrupt source of ECLIC can be set to vector or non-vector processing (via the shv field of the register clicintattr[i]). The main points are as follows:

- If configured as a vector processing mode, after the interrupt is responded by the processor core, the processor jumps directly into the target address stored in the vector table entry of the interrupt.For a detailed description of the interrupt vector table, see Section 5.8. For a detailed description of the vector processing mode, see Section 5.13.2.

- If configured as a non-vector processing mode, the processor jumps directly into the interrupt after the interrupt is processed by the processor core.

Interrupt the shared entry address.For a detailed description of the interrupt non-vector processing mode, see Section 5.13.1.

5.6.2. Update Privilege Mode

Upon entering the interrupt, the processor kernel's Privilege Mode is updated to Machine Mode.

5.6.3. Update Machine Sub-Mode

The Machine Sub-Mode of the

Bumblebee kernel is reflected in

real time in the msubm.TYP field of the CSR register. Upon entering the interrupt, the Machine Sub-Mode of the processor core is updated to the interrupt handling mode, so:

- The value of the msubm.PTYP field of the CSR register is updated to the Machine Sub-Mode (msubm.TYP field before the interrupt occurs).

  The value), as shown in Figure 5-4 Figure 5-4. The role of the msubm.PTYP field is to be able to use after the end of the interrupt.

  Sett

  ing

  The value of msubm.PTYP recovers the Machine Sub-Mode value before the interrupt occurred.

  Sett

  ing

- The value of the msubm.TYP field of the CSR register is updated to "interrupt processing mode", as shown in Figure 5-4 Figure 5-4.

  Setting

  The current mode is already

reflected in the "interrupt processing mode".

Sett

ing

5.6.4. Update CSR register mepc

The return address of the Bumblebee kernel exit interrupt is specified by the CSR register mepc. Upon entering the interrupt, the hardware will automatically update the value of the mepc register, which will be used as the return address of the exit interrupt. After the interrupt is completed, it can use its saved PC value to return to the program point that was previously stopped.

note:

- When an interrupt occurs, the interrupt return address mepc is pointed to an instruction that failed to complete execution due to the occurrence of the interrupt. Then, after exiting the interrupt, the program will return to the previous program point and re-execute from the unexecuted instruction stored by mepc.
- Although the mepc register is automatically updated by hardware when an interrupt occurs, the mepc register itself is a readable and writable register, so software can also write to it directly to modify its value.

5.6.5. Update the CSR registers mcause and mstatus

The detailed format of the mcause register is shown in Table 7-6, Table 7-6.CFF hosting when the Bumblebee kernel enters the interrupt

Sett

ing

Settin

The mcause is updated simultaneously (hardware automatically), such asFigure 5-4 Figure 5-4 As shown, the details are as follows:

- After the current interrupt is responded, a mechanism is needed to record the current id number of the interrupt source.
  - When the Bumblebee core enters the interrupt, the CSR register mcause.EXCCODE field is updated to reflect the ID number of the ECLIC interrupt source for the current response, so the software can read the register to query the specific ID of the interrupt source.
- The current interrupt is responded, possibly interrupting the interrupt that was previously being processed (the interrupt level is relatively low, so it can be interrupted), and a mechanism is needed to record the Interrupt Levels of the interrupted interrupt.
  - When the Bumblebee core enters the interrupt, the CSR register mcause.MPIL field is updated to reflect the interrupted interrupt level (the value of the mintstatus.MIL field).The purpose of the mcause.MPIL field is to use the value of mcause.MPIL to recover the mintstatus.MIL value before the interrupt occurred after the end of the interrupt.
- After the current interrupt is responded, a mechanism is needed to record the interrupt global enable state and privileged mode before responding to the interrupt.
  - When the Bumblebee core enters the interrupt, the value of the mstatus.MPIE field in the CSR register is updated to the global enable state of the interrupt before the interrupt occurred (the value of the mstatus.MIE field).The value of the mstatus.MIE field is updated to 0.

    (It means that the interrupt is globally closed after entering the interrupt service routine, all interrupts will be masked and not responding).

- When the Bumblebee kernel enters the interrupt, the processor's current privilege mode (Privilege Mode) switches to Machine Mode, and the value of the CSR register mstatus.MPP field is updated to the Privilege Mode before the interrupt occurs.

■ If the interrupt of the current response is in vector processing mode, the processor will jump directly into the target address stored in the vector table entry of the interrupt after responding to the interrupt.For a detailed description of the interrupt vector processing mode, see Section 5.13.2.In terms of hardware implementation, the processor needs to be divided into two steps. The first step is to take the stored target address from the interrupt vector table, and then jump to the target address in the second step.Then, in the first step of the memory access operation of "removing the stored target address from the interrupt vector table", a memory access error may occur, and a mechanism is needed to record such a special memory access error.

- When the Bumblebee core enters an interrupt, if the interrupt is in vector processing mode, the CSR register

The value of the mcause.minhv field is updated to 1 until the "two-step" operation described above is completely successful.

The value of the mcause.minhv field is cleared to 0.Assuming a memory access error occurs midway, the final processor will have an Instruction access fault and the mcause.minhv field has a value of 1 (not cleared).

- Note: The values of the mstatus.MPIE and mstatus.MPP fields are mirrored with the values of the mcause.MPIE and mcause.MPP fields, ie, under normal circumstances, the value of the mstatus.MPIE field and the value of the mcause.MPIE field are always Is completely one

As such, the value of the mstatus.MPP field is always exactly the same as the value of the mcause.MPP field.

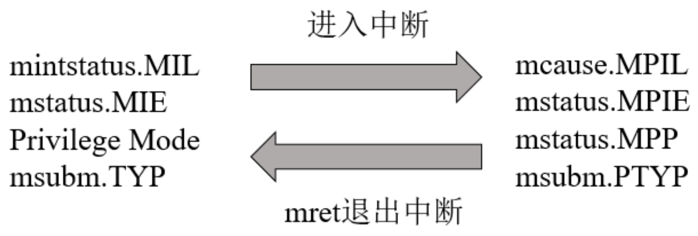


Figure 5-4 Changes to the csr register when entering/exiting an interrupt

5.7. Exit interrupt processing mode

After the program completes the interrupt processing, it finally needs to exit from the interrupt service routine and return to the main program. Since the interrupt processing is in Machine Mode, the software must use the mret instruction when exiting the interrupt. The hardware behavior of the processor after executing the mret instruction is as follows. Note that the following hardware behavior is done simultaneously in one clock cycle:

- The execution of the current program flow is stopped and the execution begins with the PC address defined by the CSR register mepc.
- Executing the mret instruction will not only cause the processor to jump to the above PC address to start execution, but also let the hardware update several other CSR registers at the same time, such asFigure 5-4 Figure 5-4 As shown, they are the following registers:
  - mstatus（Machine Status Register）
  - mcause（Machine Cause Register）
  - mintstatus（Machine Interrupt Status Register）
- In addition to this, entering the interrupt also updates the Privilege Mode of the processor core and the Machine Sub-Mode.

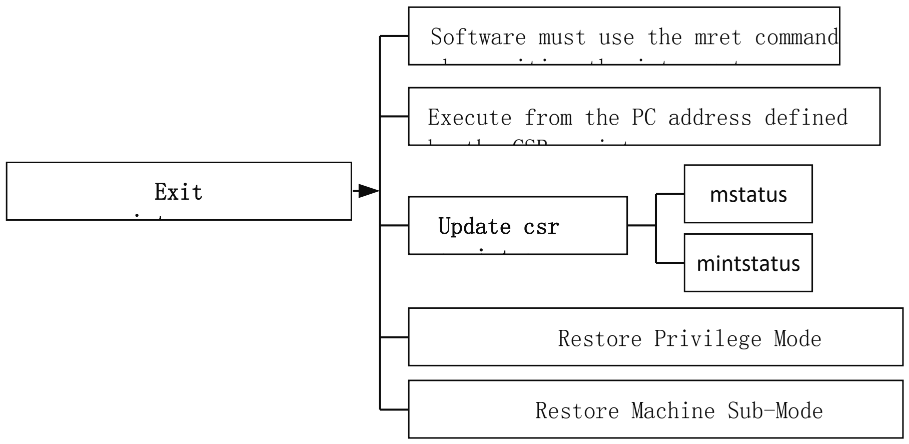


Figure 5-5 Overall process of exiting the interrupt

These will be detailed below.

5.7.1. Execute from the PC address defined by mepc

Upon entering the interrupt, the mepc register is updated simultaneously to reflect the PC value at the time the interrupt was encountered. The software must exit the interrupt using the mret instruction. After executing the mret instruction, the processor will resume execution from the pc address defined by mepc. Through this mechanism, it means that after the execution of the mret instruction, the processor returns to the PC address when the interrupt was encountered, so that the program stream that was previously aborted can be executed.

5.7.2. Update the CSR registers mcause and mstatus

The detailed format of the mcause

register is shown in Table 7-6, Table 7-6. After executing the mret instruction, the hardware will automatically update some fields of the CSR register mcause:

- Upon entering the interrupt, the value of mcause.MPIL was updated to the mintstatus.MIL value before the interrupt occurred.After exiting the interrupt with the mret instruction, the hardware restores the value of mintstatus.MIL to the value of mcause.MPIL.Through this mechanism, it means that after the exit interrupt, the processor's mintstatus.MIL value is restored to the value before the interrupt occurred.
- Upon entering the interrupt, the value of mcause.MPIE was updated to the mstatus.MIE value before the interrupt occurred.And use After the mret instruction exits the interrupt, the hardware will restore the mstatus.MIE value to mcause.MPIE after executing the mret instruction Value, such asFigure 5-4 Figure 5-4 Shown.Through this mechanism, it means that after the interrupt is interrupted

the processor’s mstatus.MIE

The value is restored to the value before the interrupt occurred.

- Upon entering the interrupt, the value of mcause.MPP was updated to the Privilege Mode before the interrupt occurred.After exiting the interrupt with the mret instruction, the hardware restores the processor privilege mode (Privilege Mode) to

The value of mcause.MPP, such asFigure 5-4 Figure 5-4

Shown.Through this mechanism, it means that after the exit interrupt, the processor's privilege mode (Privilege Mode) is restored to the mode before the interrupt occurred.

- Note: The values of the mstatus.MPIE and mstatus.MPP fields are mirrored with the values of the mcause.MPIE and mcause.MPP fields, ie, under normal circumstances, the value of the mstatus.MPIE field and the value of the mcause.MPIE field are always It is exactly the same, the value of the mstatus.MPP field is always exactly the same as the value of the mcause.MPP field.

5.7.3. Update Privilege Mode

After executing the mret instruction, the hardware will automatically update the processor's Privilege Mode to the value of the mcause.MPP field:

- Upon entering the interrupt, the value of mcause.MPP was updated to the Privilege Mode before the interrupt occurred.After exiting the interrupt with the mret instruction, the hardware restores the processor privilege mode (Privilege Mode) to

  The value of mcause.MPP.Through this mechanism, it means that after the exit interrupt, the processor's privilege mode (Privilege Mode) is restored to the mode before the interrupt occurred.

5.7.4. Update Machine Sub-Mode

The Machine Sub-Mode of the Bumblebee kernel is reflected in real time in the msubm.TYP field of the CSR register.Executing

After the mret instruction, the hardware will automatically restore the processor's Machine Sub-Mode to the value of the msubm.PTYP field:

- When an interrupt is entered, the value of the msubm.PTYP field is updated to the Machine Sub-Mode value before the interrupt occurred.After exiting the interrupt with the mret instruction, the hardware restores the value of the processor Machine Sub-Mode to

  The value of the msubm.PTYP field, such asFigure 5-4 Figure 5-4 Shown.This mechanism means that after the exit interrupt, the processor's Machine Sub-Mode is restored to the Machine Sub-Mode before the interrupt occurred.

5.8. Interrupt vector table

Such asFigure 5-6 Figure 5-6 As shown in the figure, the interrupt vector table refers to a continuous address space opened in the memory, the address

Set

ing

Each word of the space (Word) is used to store the Interrupt Service Routine corresponding to each interrupt source of the ECLIC (Interrupt Service Routine,

Se

tting

Isr) The pc address of the function.

The start address of the interrupt vector table is specified by the CSR register mtvt. The mtvt register can usually be set to the beginning of the entire code segment.

The role of the interrupt vector table is very important. When the processor responds to an interrupt source, whether the interrupt is in vector processing mode or non-vector processing mode, the hardware will eventually jump to the corresponding address by querying the pc address stored in the interrupt vector table. In the interrupt service routine function, see Section 5.13 for more details.

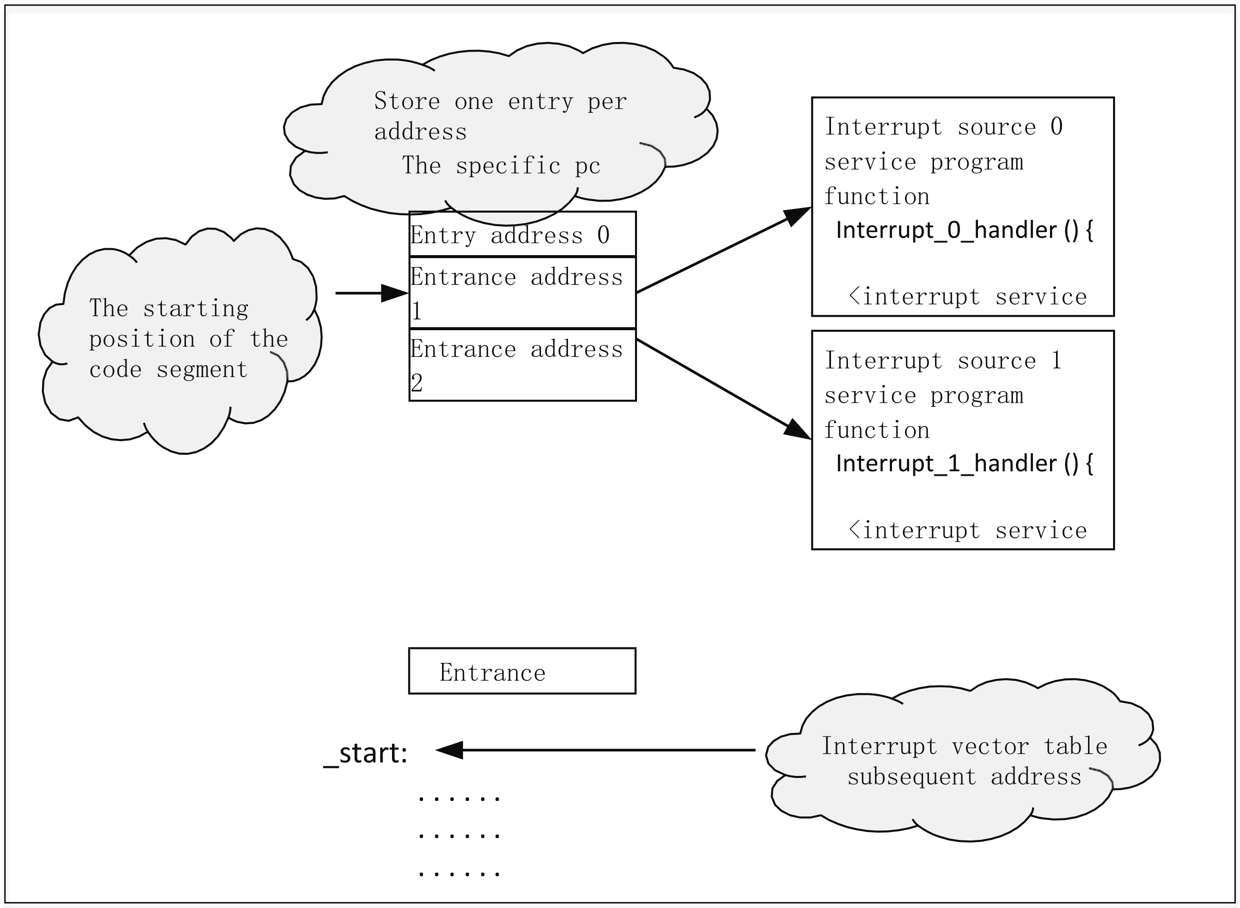

Store one entry per address
The specific pc
The starting position of the code segment
Entry address 0
Entrance address 1
Entrance address 2
Interrupt source 0 service program function
Interrupt_0_handler () {
<interrupt service
Interrupt source 1 service program function
Interrupt_1_handler () {
<interrupt service
Entrance
_start:
......
......
......
Interrupt vector table subsequent address

Figure 5-6 Schematic diagram of the interrupt vector table

5.9. Context save and restore of incoming and outgoing interrupts

The risc-v architecture processor does not have hardware to automatically save and restore context (general purpose registers) operations when entering and exiting the interrupt processing mode, so software is required to explicitly use (in assembly language) instructions for context saving and recovery. Depending on whether the interrupt is in vector processing mode or non-vector processing mode, the content involved in context saving and recovery will vary, see

First **5.13** Learn more about the section.

5.10. Interrupt response delay

The concept of interrupt response delay usually refers to the instruction consumed from ″external interrupt source pull-up″ to ″the first instruction in the Interrupt Service Routine (ISR) that the processor actually starts executing the interrupt source″. The number of cycles. Therefore, the interrupt response delay usually includes the periodic overhead of the following aspects:

- The overhead of the processor core to jump after responding to the interrupt
- The cycle overhead spent by the processor core to save the context
- The overhead that the processor core jumps into the Interrupt Service Routine (ISR).

The interrupt response delay will vary depending on whether the interrupt is in vector processing mode or non-vector processing mode, see section **5.13**

Learn more about the section.

5.11. Interrupt nesting

While the processor core is processing an interrupt, there may be a new interrupt request of a higher level. The processor can abort the current interrupt service routine and start responding to the new interrupt and execute its ″interrupt service routine″. So, the interrupt nesting is formed (that is, the previous interrupt has not responded yet, and the new interrupt is started again), and the nested hierarchy can have many layers.

TakeFigure 5-7 Figure 5-7 An example of this is:

Setting
Settin

- Assuming the processor is processing a timer interrupt and suddenly another button 1 interrupt is coming (level is higher than the timer interrupt), the processor will pause processing the timer interrupt and begin processing the button 1 interrupt.

- But suddenly another button 2 interrupt comes (level is higher than button 1 interrupt), then the processor will pause

  When the button 1 is interrupted, the button 2 interrupt is processed.

- After that, no other higher-level interrupts will come, the button 2 interrupt will not be interrupted, the processor can successfully complete the interrupt of the button 2, and then return to the button 1 interrupt handler, complete the button 1 interrupt Processing.
- After completing the processing of the button 1 interrupt, the processor will return to the timer interrupt handler to complete the timer interrupt processing.

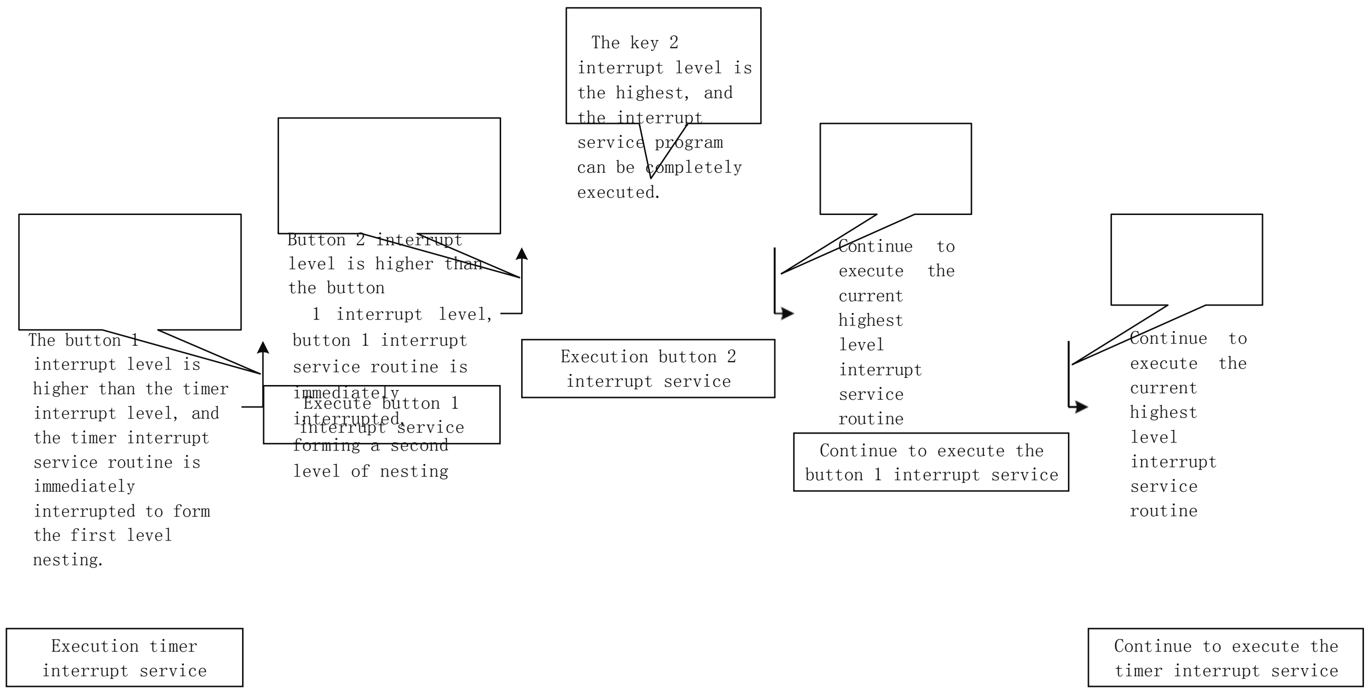

The key 2 interrupt level is the highest, and the interrupt service program can be completely executed.
Button 2 interrupt level is higher than the button 1 interrupt level, button 1 interrupt service routine is immediately interrupted, forming a second level of nesting
The button 1 interrupt level is higher than the timer interrupt level, and the timer interrupt service routine is immediately interrupted to form the first level nesting.
Execute button 1 interrupt service
Execution button 2 interrupt service
Continue to execute the current highest level interrupt service routine
Continue to execute the button 1 interrupt service
Continue to execute the current highest level interrupt service routine
Execution timer interrupt service
Continue to execute the timer interrupt service

Figure 5-7 Schematic diagram of interrupt nesting

Note: Assuming that the new interrupt request has a lower priority (or the same) as the interrupt level being processed, the processor should not respond to this new interrupt request. The processor must complete the current interrupt service routine before considering the new response. Interrupt request (since the level of the new interrupt request is not higher than the interrupt level currently being processed).See Section 6.2.9 for more information on interrupt level settings.

In the Bumblebee kernel, depending on whether the interrupt is in vector processing mode or non-vector processing mode, the support method for interrupt nesting will vary, see section **5.13** Learn more about the section.

5.12. Interrupted biting

While the processor core is processing an interrupt, a new interrupt request may be coming, but the ″new interrupt level″ is lower than or equal to ″the interrupt level currently being processed″, so the new interrupt cannot interrupt the current interrupt. Interrupt (so does not form a nest).

After the processor completes the

current interrupt, it is theoretically necessary to restore the context, then exit the interrupt back to the main application, then re-respond to the new interrupt, and in response to the new interrupt, the context needs to be saved again. Therefore, there is a back-to-back "recovery context" and "save context" operation. If this back-to-back "recovery context" and "save context" are omitted, it is called "medium".

Broken tail bite, such asFigure 5-8

Figure 5-8 As shown, it is obvious that interrupting the tail bite can speed up the back-to-back processing speed of multiple interruptions.

Settin
Settin

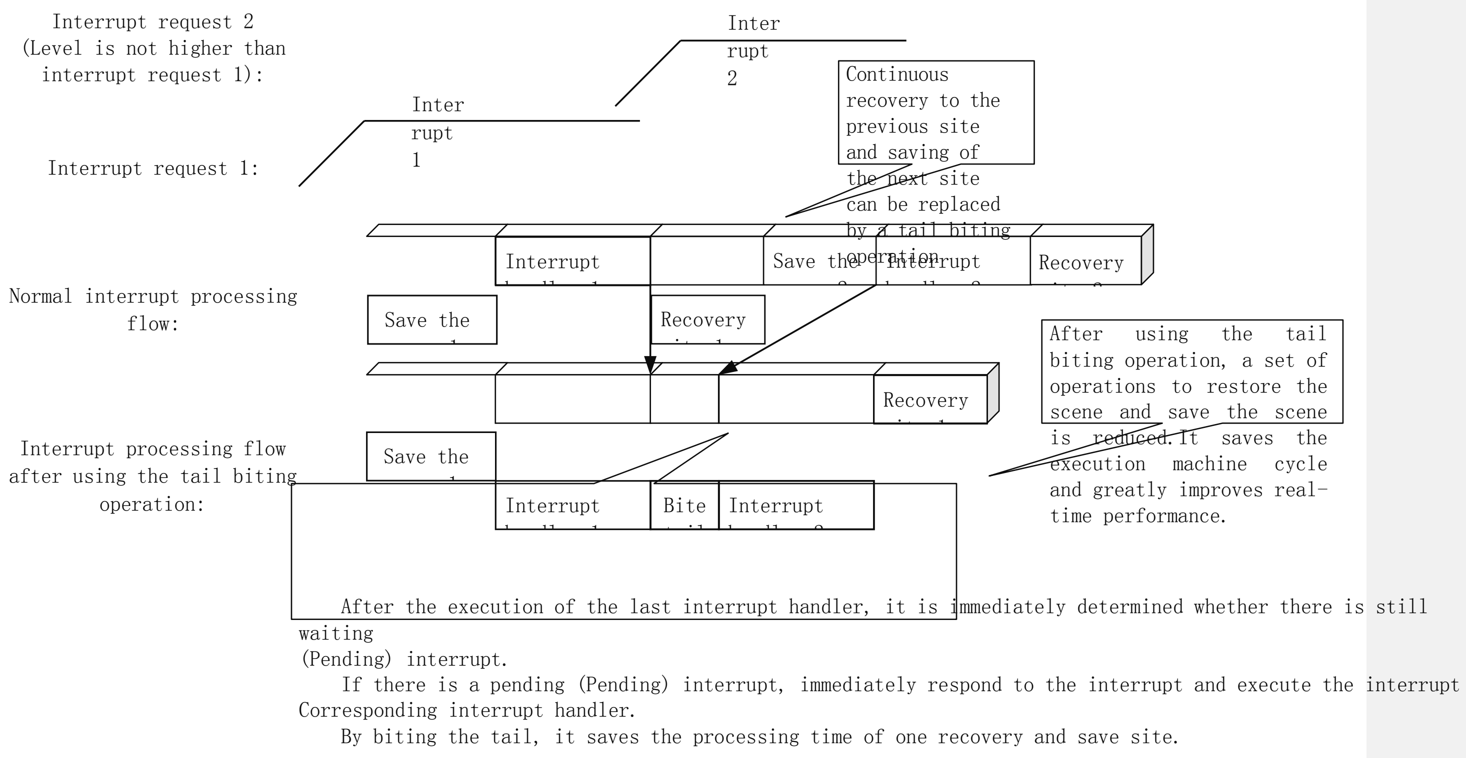

Interrupt request 2
(Level is not higher than interrupt request 1):
Inter rupt 2
Interrupt request 1:
Inter rupt 1
Continuous recovery to the previous site and saving of the next site can be replaced by a tail biting operation
Normal interrupt processing flow:
Interrupt
Save the
Interrupt
Recovery
Save the
Recovery
After using the tail biting operation, a set of operations to restore the scene and save the scene is reduced. It saves the execution machine cycle and greatly improves real-time performance.
Recovery
Interrupt processing flow after using the tail biting operation:
Save the
Interrupt
Bite
Interrupt
After the execution of the last interrupt handler, it is immediately determined whether there is still waiting
(Pending) interrupt.
If there is a pending (Pending) interrupt, immediately respond to the interrupt and execute the interrupt Corresponding interrupt handler.
By biting the tail, it saves the processing time of one recovery and save site.

Figure 5-8 Schematic diagram of interrupt bite

In the Bumblebee kernel, interrupt biting is only supported in non-vector processing mode, see Section 5.13.1.3 for more details.

5.13. Interrupted vector processing mode and non-vector processing mode

As described in Section 6.2.2062.10, each interrupt source of eclic can be set to vector or non-vector processing (by sending

ing Set

The shv domain of the memory clicintattr[i], the vector processing mode and the non-vector processing mode have large differences, respectively, as follows.

5.13.1. Non-vector processing mode

5.13.1.1 Features and delays of non-vector processing modes

If configured as a non-vector processing mode, the interrupt is interrupted by the processor core and the processor jumps directly into the

entry address of all non-vector interrupt shares. The entry address can be set by software:

- If the least significant bit of the configuration CSR register mtvt2 is 0 (power-on reset default), the entry address of all non-vector interrupt shares is specified by the value of the CSR register mtvec (ignoring the value of the lowest 2 bits).Since the value of the mtvec register also specifies the entry address of the exception, it means that in this case, the exception and all non-vector interrupts share the entry point.

site.

- If the least significant bit of the configuration CSR register mtvt2 is 1, then the entry address shared by all non-vector interrupts is specified by the value of the CSR register mtvt2 (ignoring the value of the lowest 2 bits). In order for the interrupt to be responded and processed as fast as possible, it is recommended to set the least significant bit of the CSR register mtvt2 to 1, ie, specify a separate entry address for all non-vector interrupts by mtvt2, and the exception entry address (by The value of mtvec is specified) completely separated.

After entering the entry address of all non-vector interrupt shares, the processor will begin executing a common piece of software code, such asFigure

5-9 ~~Figure 5-9~~ In the example shown, the contents of this software code are usually as follows

- First save the CSR registers mepc, mcause, msubm into the stack. These CSR registers are saved to ensure that subsequent interrupt nesting is functional, because the new interrupt response will overwrite the values of mepc, mcause, and msubm, so they need to be saved to the stack first.

- Save several general-purpose registers (the context of the processor) onto the stack.

- Then execute a special instruction "csrrw ra, CSR_JALMNXTI, ra". If there is no interruption waiting

(Pending), the instruction is

equivalent to a Nop instruction does nothing; if there is an interrupt waiting (Pending), after executing the instruction, the processor will:

- The target address stored in the Vector Table Entry of the interrupt is directly jumped into the Interrupt Service Routine (ISR) of the interrupt source.
- While jumping into the interrupt service routine, the hardware also turns on the global enable of the interrupt, that is, sets the MIE field of the mstatus register to 1.After the interrupt global enable is enabled, the new interrupt can be responded to achieve the effect of interrupt nesting.
- The "csrrw ra, CSR_JALMNXTI, ra" command also reaches JAL while jumping into the interrupt service routine.

  (Jump and Link) effect, the hardware also updates the value of the Link register as the return address of the PC itself of the instruction as a function call.Therefore, after returning from the interrupt service routine function, it will return to the "csrrw ra, CSR_JALMNXTI, ra" instruction to re-execute, and re-determine whether there is still an interrupt waiting (Pending), thereby achieving the effect of interrupting the tail bite.

- At the end of the interrupt service routine, you also need to add the corresponding recovery context pop operation.And before the CSR registers mepc, mcause, msubm are out of the stack, the global enable of the interrupt needs to be turned off again to ensure the mepc, The atomicity of mcause, msubm recovery operations.(not interrupted by new interrupts).

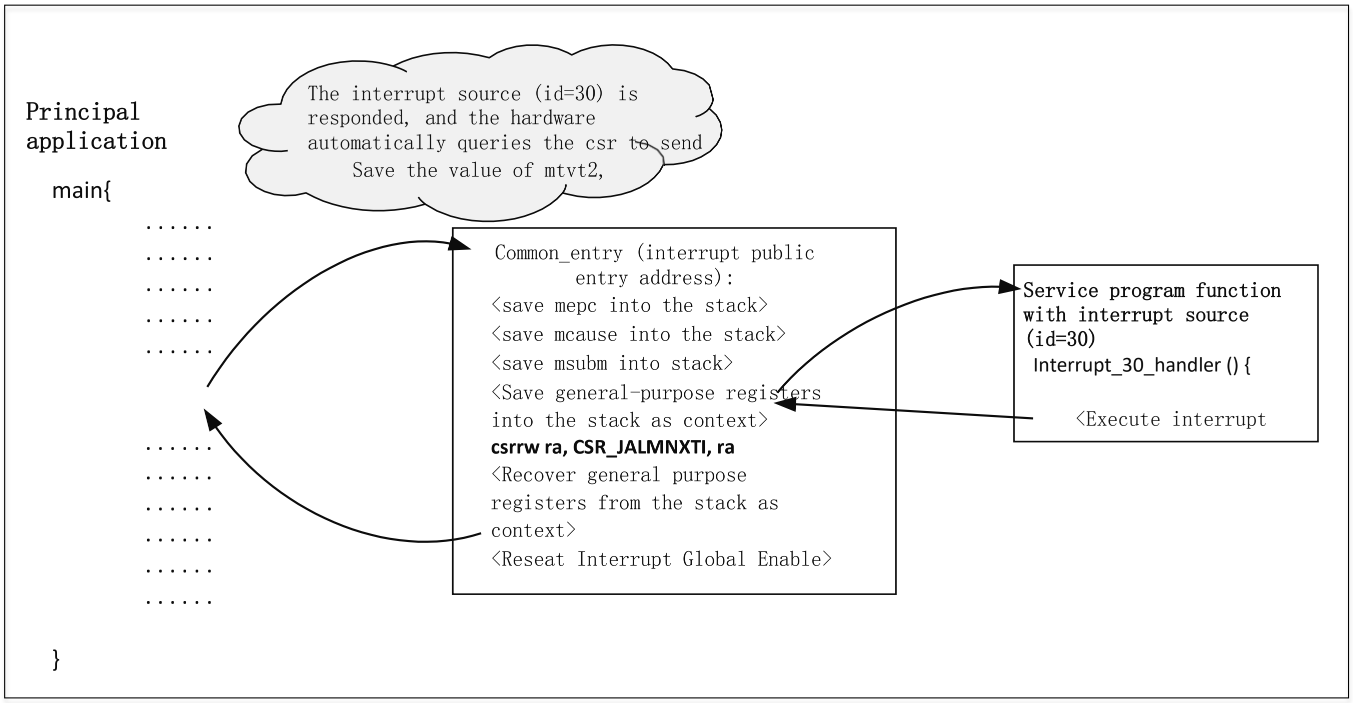

Principal application
main{
......
}
The interrupt source (id=30) is responded, and the hardware automatically queries the csr to send Save the value of mtvt2,
Common_entry (interrupt public entry address):
<save mepc into the stack>
<save mcause into the stack>
<save msubm into stack>
<Save general-purpose registers into the stack as context>
csrrw ra, CSR_JALMNXTI, ra
<Recover general purpose registers from the stack as context>
<Reseat Interrupt Global Enable>
Service program function with interrupt source (id=30)
Interrupt_30_handler () {
<Execute interrupt

Figure 5-9 Example of interrupted non-vector processing mode (always nesting is always supported)

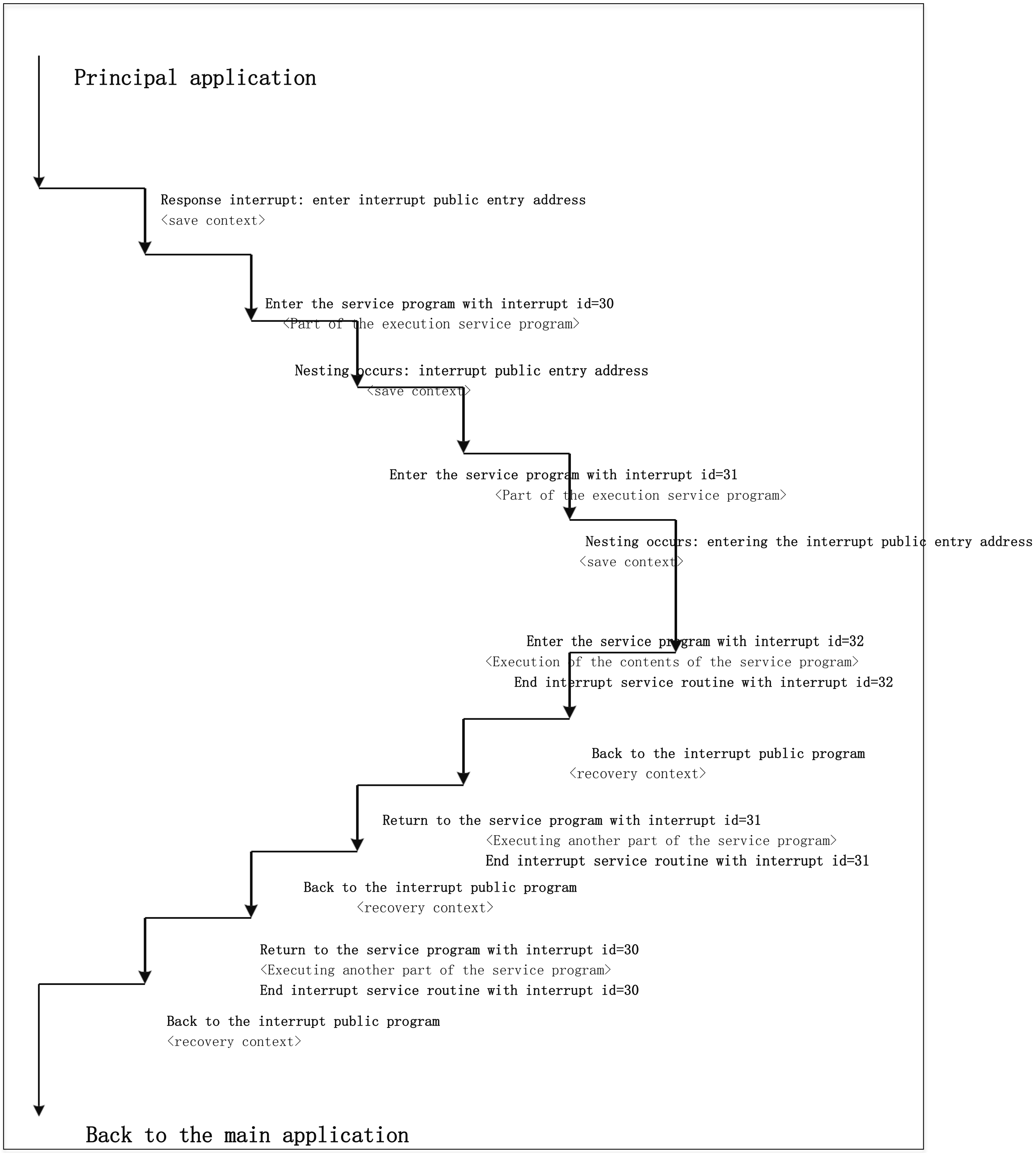
Since the processor needs to execute a common software code for context preservation before jumping to the interrupt service routine in the non-vector processing mode, the first instruction in the interrupt service routine is executed from the interrupt source to the processor. Need to experience the following clock cycle overhead:

- The overhead of the processor core responding to a jump after an interrupt. Ideally about 4 clock cycles.
- The processor core saves the overhead of the CSR registers mepc, mcause, and msubm into the stack.
- The periodic cost of the processor core saving the context. If it is the architecture of rv32e, you need to save 8 general-purpose registers. If it is the architecture of rv32i, you need to save 16 general-purpose registers.
- The overhead that the processor core jumps into the Interrupt Service Routine (ISR). Ideally, it takes about 5 clock cycles.

Interrupt nesting in non-vector processing mode

As mentioned above, non-vector processing mode can always support interrupt nesting, such asFigure 5-

10 Figure 5-10 The example shown in the example: Assume that the three interrupt sources 30, 31, 32 come in succession, and "level of interrupt source 32" > "level of interrupt source 31" > "level of interrupt source 30", then later Interrupts interrupt interrupts that were previously processed to form interrupt nesting.



Principal application
Response interrupt: enter interrupt public entry address
<save context>
Enter the service program with interrupt id=30
<Part of the execution service program>
Nesting occurs: interrupt public entry address
<save context>
Enter the service program with interrupt id=31
<Part of the execution service program>
Nesting occurs: entering the interrupt public entry address
<save context>
Enter the service program with interrupt id=32
<Execution of the contents of the service program>
End interrupt service routine with interrupt id=32
Back to the interrupt public program
<recovery context>
Return to the service program with interrupt id=31
<Executing another part of the service program>
End interrupt service routine with interrupt id=31
Back to the interrupt public program
<recovery context>
Return to the service program with interrupt id=30
<Executing another part of the service program>
End interrupt service routine with interrupt id=30
Back to the interrupt public program
<recovery context>
Back to the main application

Figure 5-10 Three successive (non-vector processing mode) interrupts form a nest

5.13.1.3 Interrupt bite in non-vector processing mode

For non-vector processing mode interrupts, the ″interruption bite″ saves significant time (saving one back-to-back save context) since the processor has to save and restore the context before jumping into and out of the interrupt service routine. And recovery context).

As mentioned above, in all common code segments shared by non-vector interrupts, the ″csrrw ra, CSR_JALMNXTI, ra″ instruction also achieves the effect of JAL (Jump and Link) while jumping into the interrupt service routine, and the hardware is updated simultaneously. The value of the Link register is the return address of the PC itself of the instruction as a function call. Therefore, after returning from the interrupt service routine function, it will return to the "csrrw ra, CSR_JALMNXTI, ra" instruction to re-execute, and re-determine whether there is still an interrupt waiting (Pending), thereby achieving the effect of interrupting the tail bite.

Such asFigure 5-11 Figure 5-11 Example shown in the following: Assume that the three interrupt sources 30, 29, and 28 come in succession, and ″level of interrupt source 30″ >= ″level of interrupt source 29″ >= ″level of interrupt source 28″, then Subsequent interrupts will not interrupt the interrupt that was being processed before (no interrupt nesting will be formed), but will be placed in the Pending state. When the interrupt source 30 is completed

After processing, interrupt processing of interrupt source 29 will be started directly, eliminating the intermediate "recovery context" and "save context" procedures.

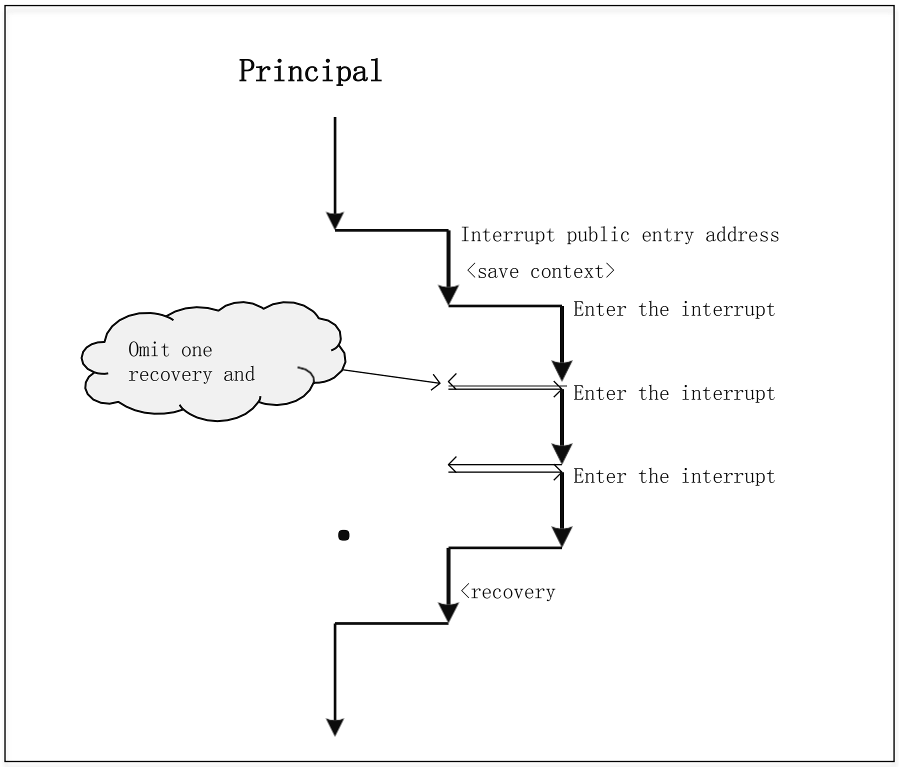


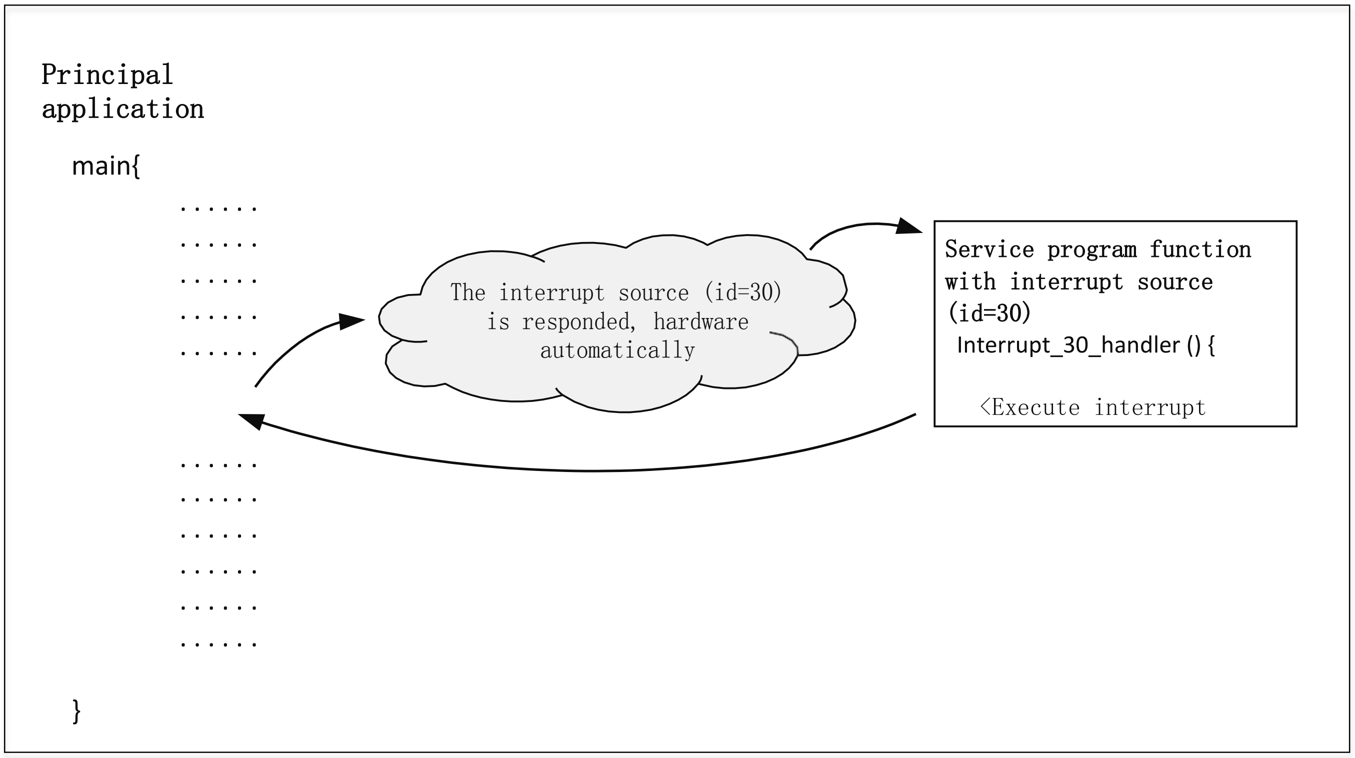
Figure 5-11 Schematic diagram of interrupt bite

5.13.2. Vector processing mode

5.13.2.1 The characteristics and delay of the vector processing mode

If configured as a vector processing mode, after the interrupt is responded by the processor core, the processor directly jumps to the target address stored in the Vector Table Entry of the interrupt, that is, the interrupt service routine of the interrupt source (Interrupt Service) Routine, ISR), such as Figure 5-12

Figure 5-12 The example shown in .



Principal application
main{
......
The interrupt source (id=30) is responded, hardware automatically
Service program function with interrupt source (id=30)
Interrupt_30_handler () {
<Execute interrupt
}

Figure 5-12 Example of vector processing mode for interrupts

The vector processing mode has the following characteristics:

- In the vector processing mode, the processor jumps directly to the interrupt service routine and does not save the context. Therefore, the interrupt response delay is very short, from the interrupt source to the first instruction in the interrupt service routine. Basically, only hardware is required to perform table lookup and jump time overhead, ideally about 6 clock cycles.
- For vector service mode interrupt service routine functions, special interrupt ((interrupt)) must be used to modify the interrupt service routine function.

- In the vector processing mode, since the processor does not save the context before jumping into the interrupt service routine, the interrupt service routine function itself cannot theoretically call the subfunction (ie, must be a Leaf Function).
  - If the interrupt service routine function accidentally calls another subfunction (not a Leaf Function), it will cause a function error if it is not processed. In order to avoid this accidental error situation, as long as a special __Attribute ((interrupt)) to modify the interrupt service program function, then the compiler will automatically determine, when the compiler finds that the function calls other sub-functions, it will automatically insert a piece of code for context preservation. Note: In this case,

although the correctness of the function is guaranteed, the overhead caused by saving the context will actually increase the response delay of the interrupt (equivalent to the non-vector mode) and cause the expansion of the code size. .Therefore, in practice, if the vector processing mode is used, it is not recommended to call other sub-functions in the vector processing mode interrupt service routine function.

- In the vector processing mode, the processor does not perform any special processing before jumping into the interrupt service routine, and since the processor core responds to the interrupt, the MIE field in the mstatus register will be automatically updated by the hardware.

0 (meaning that the interrupt is globally closed and cannot respond to new interrupts). Therefore, the vector processing mode does not support interrupt nesting by default. In order to achieve the vector processing mode and interrupt the nesting effect, such asFigure 5-13 Figure 5-13 As shown in the figure, you need to add a special push operation at the beginning of the interrupt service routine:

- First save the CSR registers mepc, mcause, msubm into the stack. These CSR registers are saved to ensure that subsequent interrupt nesting functions correctly, because the new interrupt response will overwrite mepc, mcause,

The value of msubm, so you need to save them to the stack first.

- Re-enable the global enable of the interrupt, that is, set the MIE field of the mstatus register to 1.After the interrupt global enable is enabled, the new interrupt can be responded to achieve the effect of interrupt nesting.
- At the end of the interrupt service routine, you also need to add the corresponding recovery context pop operation.And before the CSR registers mepc, mcause, msubm are out of the stack, the global enable of the interrupt needs to be turned off again to ensure the mepc,

The atomicity of mcause, msubm recovery operations (not interrupted by new interrupts).

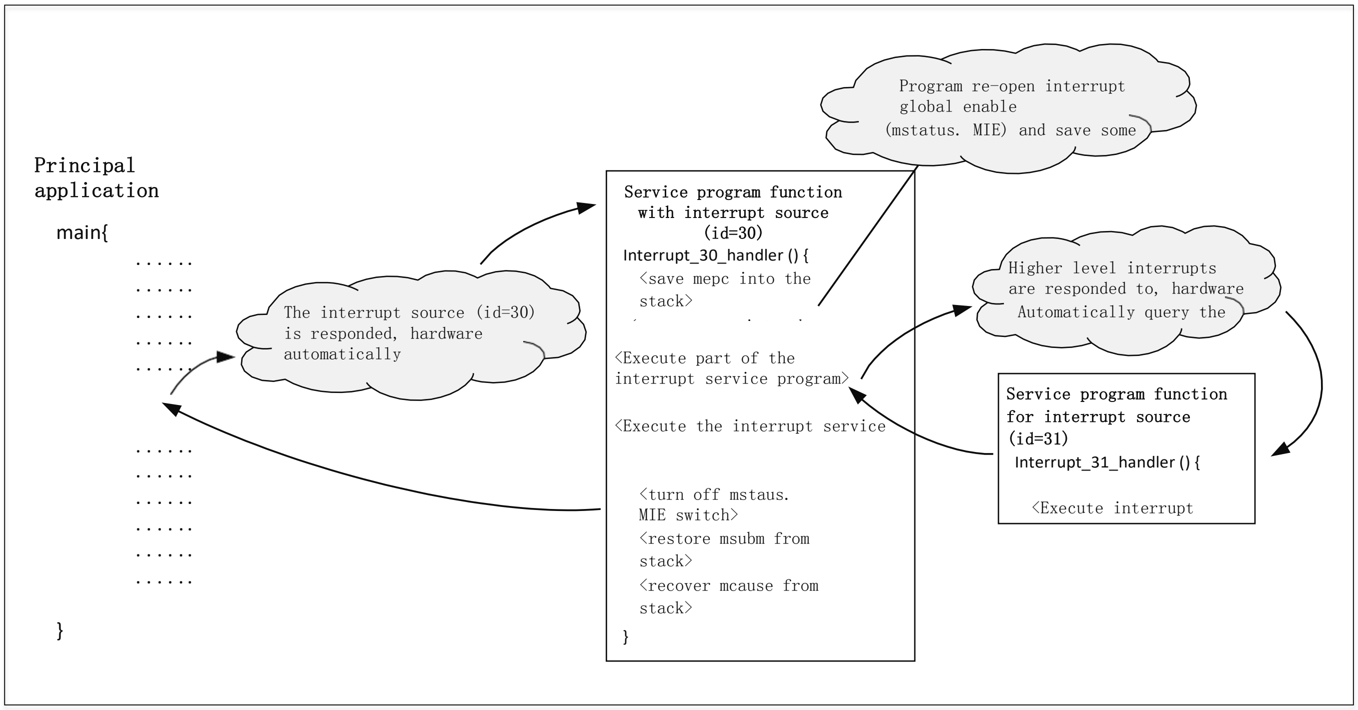

Principal application
main{
......
}
The interrupt source (id=30) is responded, hardware automatically
Service program function with interrupt source (id=30)
Interrupt_30_handler () {
<save mepc into the stack>
<Execute part of the interrupt service program>
<Execute the interrupt service
<turn off mstaus. MIE switch>
<restore msubm from stack>
<recover mcause from stack>
}
Program re-open interrupt global enable (mstatus. MIE) and save some
Higher level interrupts are responded to, hardware Automatically query the
Service program function for interrupt source (id=31)
Interrupt_31_handler () {
<Execute interrupt

Figure 5-13 Example of vector processing mode for interrupts (support interrupt nesting)

5.13.2.2 Vector processing mode interrupt nesting

As mentioned above, interrupts in vector processing mode can also support interrupt nesting after special processing, such asFigure 5-14 Figure 5-14 The example shown in the example: Assume that the three interrupt sources 30, 31, 32 come in succession, and ″level of interrupt source 32″ > ″level of interrupt source 31″ > ″level of interrupt source 30″, then later Interrupts interrupt interrupts that were previously processed to form interrupt nesting.

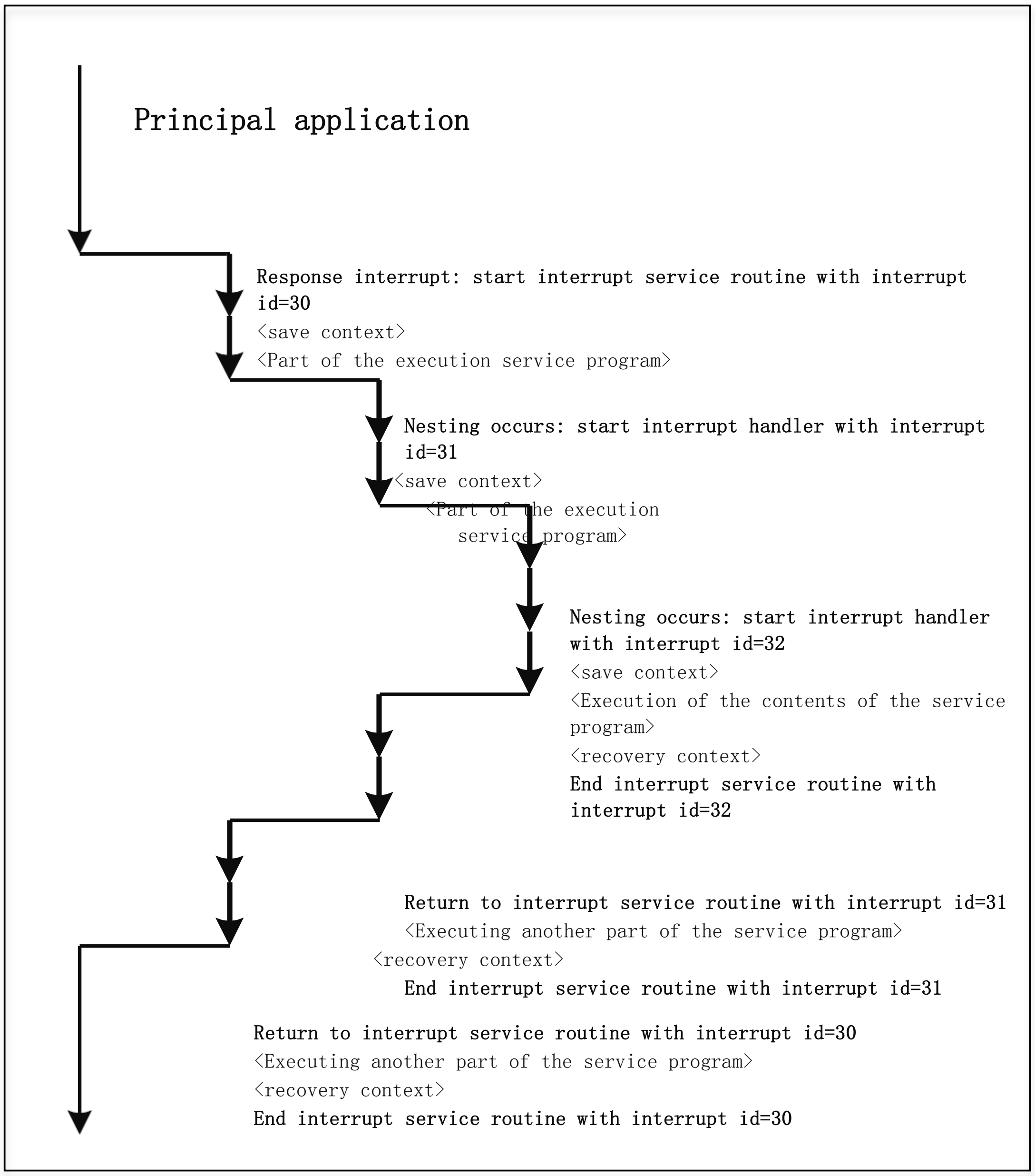

Principal application
Response interrupt: start interrupt service routine with interrupt id=30
<save context>
<Part of the execution service program>
Nesting occurs: start interrupt handler with interrupt id=31
<save context>
<Part of the execution service program>
Nesting occurs: start interrupt handler with interrupt id=32
<save context>
<Execution of the contents of the service program>
<recovery context>
End interrupt service routine with interrupt id=32
Return to interrupt service routine with interrupt id=31
<Executing another part of the service program>
<recovery context>
End interrupt service routine with interrupt id=31
Return to interrupt service routine with interrupt id=30
<Executing another part of the service program>
<recovery context>
End interrupt service routine with interrupt id=30

Figure 5-14 Three successive (vector processing mode) interrupts form a nest

5.13.2.3 Vector processing mode interrupt bite

For the vector processing mode interrupt, since the processor does not save the context before jumping into the interrupt service routine, the meaning of ″interrupt biting″ is not significant. Therefore, the vector processing mode is interrupted without ″interruption″. The ability to bite the tail.

6. Introduction to Bumblebee Core TIMER and ECLIC

6.1. Timer introduction

6.1.1. Introduction to timer

Timer Unit (TIMER), which is mainly used to generate timer interrupts in the Bumblebee kernel (Timer Interrupt) and Software Interrupt.See Sections 5.3.2.1 and 5.3.2.2 for details on timer interrupts and software interrupts.

6.1.2. Timer register

Timer is a unit of memory address mapping:

- The base address of the TIMER unit in the Bumblebee kernel is described in the Bumblebee Kernel Concise Data Sheet.
- Timer unit address and address offsetTable 6-1 Table 6-1 Shown in .

Table 6-1 Memory Map Addresses of the timer Register

Intra-module offset address	Read and write properties	Register name	Reset default	Functional description
0x0	Readable and writable	mtime_lo	0x00000000	Reflects the lower 32-bit value of timer mtime, see section 6.1.3 Learn more about it in detail.
0x4	Readable and writable	mtime_hi	0x00000000	Reflects the high 32-bit value of the timer mtime, see section 6.1.3 Learn more about it in detail.
0x8	Readable and writable	mtimecmp_lo	0xFFFFFFFF	Configure the timer comparison value mtimecmp to be lower 32 bits, see See section 6.1.5 Learn more about it in detail.
0xC	Readable and writable	mtimecmp_hi	0xFFFFFFFF	Configure the timer comparison value mtimecmp high 32 bits, see See section 6.1.5 Learn more about it in detail.
0xFF8	Readable and writable	mstop	0x00000000	Control the pause of the timer, see section 6.1.4 Learn about it in detail Detailed introduction.
0xFFC	Readable and writable	msip	0x00000000	Generate software interrupts, see section 6.1.6 Learn more about it Shao.

note:

- TIMER's registers only support aligned read and write accesses with a size of (Size).
- The register range of TIMER is 0x00 ~ 0xFF, and the value in the address other than the one listed in the above table is constant 0.

The function and use of each register are described in detail below.

6.1.3. Timing through the mtime register

The timer can be used for real time timing. The main points are as follows

- A 64-bit mtime register is implemented in TIMER, concatenated by {mtime_hi, mtime_lo}, which reflects the value of the 64-bit timer. The timer increments according to the low-speed input beat signal. The timer is turned on by default, so it will always count.
- In the Bumblebee kernel, the self-incrementing frequency of this counter is controlled by the processor's input signal mtime_toggle_a, see the document Bumblebee Core Concise Data Sheet for details on this input signal.

6.1.4. Pause timer through mstop register

Since the timer's timer is powered on, it will continue to increment by default, in order to turn off this timer in some special cases.

Counting, an mstop register is implemented in TIMER. Such as Table 6-2

Table 6-2 As shown, the mstop register

has only the lowest bit.

设置

As a valid bit, this valid bit acts directly as a timer's pause control signal, so software can set the mstop register

设

置

1 to pause the timer.

Table 6-2 Bit field of register mstop

doma in name	Bit	Attribut es	Reset value	desc ript ion
Reserved	7:1	Read only, write ignore	N/A	Unused field, value is constant 0
TIMESTOP	0	Readable and writable	0	Control the timer to run or pause. If the value of this field is 1, the timer pauses counting. Otherwise normal auto increment.

6.1.5. Generate timer interrupts via the mtime and mtimecmp registers

The timer can be used to generate a timer interrupt, the main points are as follows:

- A 64-bit mtimecmp register is implemented in TIMER, which is composed of {mtimecmp_hi, mtimecmp_lo}. This register is used as a comparison value of the timer, assuming that the value mtime of the timer is greater than or equal to

The value of mtimecmp generates a timer interrupt. The software can clear the timer interrupt by overwriting the value of mtimecmp or mtime (so that mtimecmp is greater than the value of mtime).

Note: The timer interrupt is connected to the eclic unit for unified management. For details on eclic, please refer to 6.2~~6.~~2 Section.

6.1.6. Generate software interrupts via msip register

TIMER can be used to generate software interrupts. A msip register is implemented in TIMER, such asTable 6-3 Table 6-3 As shown in the msip register, only the least significant bit is a valid bit, which is directly interrupted as a software, so:

- Software write generates a software interrupt by writing a 1 to the msip register;
- Software can clear the software interrupt by writing a 0 to the msip register.

Note: Software interrupts are connected to the eclic unit for unified management. For details on eclic, see section **6.2** Section.

Table 6-3 Bit field of register msip

doma in name	Bit	Attributes	Reset value	description
Reserved	7:1	Read only, write ignore	N/A	Unused field, value is constant 0
MSIP	0	Readable and writable	0	This field is used to generate software interrupts

6.2. Eclic introduction

The Bumblebee kernel supports ″improved kernel interrupt controllers optimized from the RISC-V standard CLIC

(Enhanced Core Local Interrupt

Controller, ECLIC)″, used to

manage all interrupt sources.

Note:

- Eclic serves only one processor core and is private to the processor core.
- The software programming model of eclic is also backward compatible with the standard clic.

6.2.1. Introduction to eclic

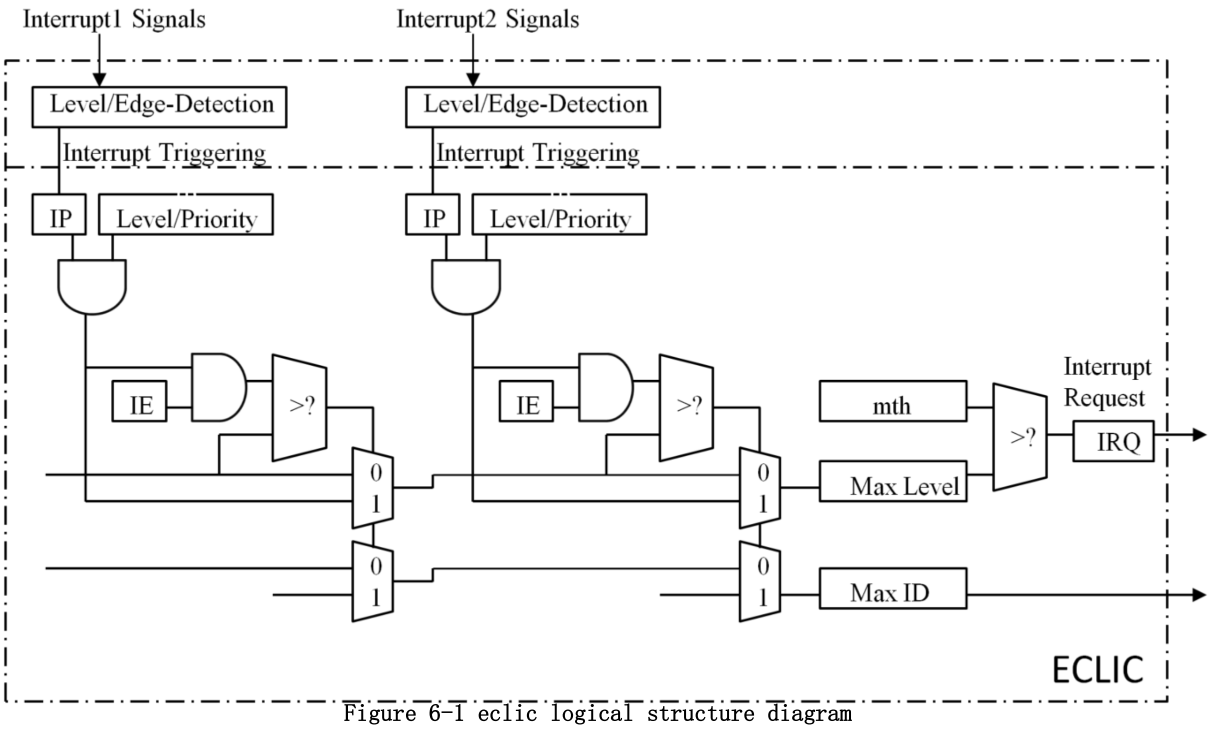


Figure 6-1 eclic logical structure diagram

Eclic is used to arbitrate, send requests, and support interrupt nesting for multiple internal and external interrupt sources. Eclic register

Such asTable 6-5 Table 6-5 Said logical structure such asFigure 6-1 Figure 6-1 As shown, the related concepts are as follows:

- Eclic interrupt target
- Eclic interrupt source
- Eclic interrupt source number
- Eclic register
- Eclic interrupt source enable bit
- Eclic interrupt source wait flag
- Level or edge attribute of the eclic interrupt source
- Eclic interrupt source level and priority

设置

color,

设置

color,

设置

color,

设置

color,

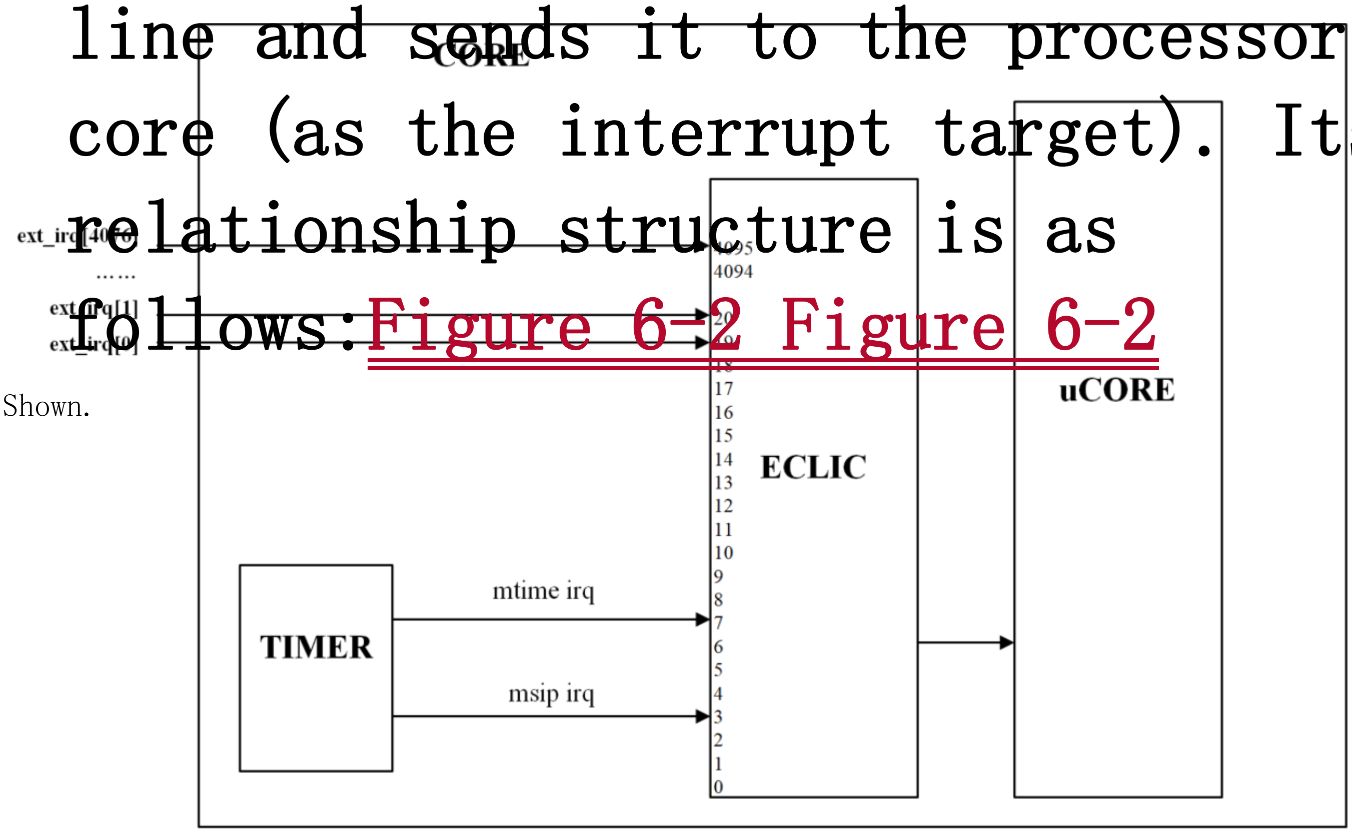
- Vector or non-vector processing of eclic interrupt sources
- Eclic interrupt target threshold level
- Arclic interrupt arbitration mechanism
- The response, nesting, and

tailing mechanisms of the eclic

interrupt are detailed below.

6.2.2. Eclic interrupt target

The eclic unit generates an interrupt line and sends it to the processor core (as the interrupt target). Its relationship structure is as follows:Figure 6-2 Figure 6-2



Shown.

设置

设置

Figure 6-2 eclic relationship structure diagram

6.2.3. Eclic interrupt source

Such as Figure 6-2 Figure 6-2 As shown, ~~ECLIC theoretically supports up to 4096 interrupt sources (interrupt) from the programming model.~~

Source). ECLIC defines the following characteristics and parameters for each interrupt source:

- Number (id)
- Enable bit (ie)
- Waiting flag bit (ip)
- Level or Edge-Triggered
- Level and Priority
- Vector or Non-Vector Mode is described below.

6.2.4. Eclic interrupt source number (id)

Eclic assigns a unique number (id) to each interrupt source. For example, if the hardware implementation of an eclic really supports 4096 ids, the id should be 0 to 4095. note:

- In the Bumblebee kernel, interrupts with interrupt ID numbers 0 through 18 are reserved as kernel-specific internal interrupts.
- The interrupt source id assigned to the normal external interrupt starts at 19 and can

be used by the user to connect to an external interrupt source.

Detailed introductionTable 6-4 Table 6-4 Shown in .

设置

设置

Table 6-4 eclic interrupt source number and assignment

Eclic interrupt number	Features	Interrupt source introduction
0	*Reserved*	*The interrupt is not used by the Bumblebee kernel.*
1	*Reserved*	*The interrupt is not used by the Bumblebee kernel.*
2	*Reserved*	*The interrupt is not used by the Bumblebee kernel.*
3	Software interruption	A software interrupt generated by the TIMER unit of the Bumblebee kernel.
4	*Reserved*	*The interrupt is not used by the Bumblebee kernel.*
5	*Reserved*	*The interrupt is not used by the Bumblebee kernel.*
6	*Reserved*	*The interrupt is not used by the Bumblebee kernel.*
7	Timer interrupt	The timer generated by the TIMER unit of the Bumblebee kernel Broken.
8	*Reserved*	*The interrupt is not used by the Bumblebee kernel.*

9	*Reserved*	*The interrupt is not used by the Bumblebee kernel.*
10	*Reserved*	*The interrupt is not used by the Bumblebee kernel.*
11	*Reserved*	*The interrupt is not used by the Bumblebee kernel.*
12	*Reserved*	*The interrupt is not used by the Bumblebee kernel.*
13	*Reserved*	*The interrupt is not used by the Bumblebee kernel.*
14	*Reserved*	*The interrupt is not used by the Bumblebee kernel.*
15	*Reserved*	*The interrupt is not used by the Bumblebee kernel.*
16	*Reserved*	*The interrupt is not used by the Bumblebee kernel.*
17	Memory access error interrupt	Bumblebee kernel memory access error turned into internal Interrupted.
18	*Reserved*	*The interrupt is not used by the Bumblebee kernel.*
19 ~ 4095	External Interrupt	Normal external interrupts are available for user connections.note: ■ Although ECLIC supports up to 4096 interrupt sources from the programming model, the actual number of interrupt sources supported by the hardware is reflected in the information register clicinfo.NUM_INTERRUPT.

6.2.5. Eclic register

Eclic is a unit of memory address mapping:

- The base address of the ECLIC unit in the Bumblebee kernel is described in the Bumblebee Kernel Concise Data Sheet.
- Register and address offsets in the eclic unitTable 6-5 Table 6-5 Shown in .

设置 color,

设置 color,

Table 6-5 Intra-cell address offset of the eclic register

	Attr ibut	name	width

	es		
0x0000	Readable and writable	cliccfg	8 digits
0x0004	Read only, write ignore	clicinfo	32 bit
0x000b	Readable and writable	mth	8 digits
0x1000+4*i	Readable and writable	clicintip[i]	8 digits
0x1001+4*i	Readable and writable	clicintie[i]	8 digits
0x1002+4*i	Readable and writable	clicintattr[i]	8 digits
0x1003+4*i	Readable and writable	clicintctl[i]	8 digits
note: ■ The above i represents the ID number of the interrupt, and the register with the [i] suffix indicates that there is a separate register for each interrupt source. ■ ECLIC's registers support aligned read and write accesses of byte, half-word, or word. ■ Writes to the above "read only" registers are ignored, but no bus error exceptions are generated.			

- The actual eclic may not be configured with 4096 interrupt sources, so the value of the corresponding register of the non-existing interrupt source is constant 0.

The individual registers are described in detail below.

6.2.5.1 Register cliccfg

The cliccfg register is a global configuration register. Software can rewrite this register to configure several global parameters. For details on the specific bit field, seeTable 6-6 Table 6-6 Shown in .

Table 6-6 Bit field of register cliccfg

domain name	Bit	Attributes	Reset value	description
Reserved	7:5	Read only, write ignore	N/A	Unused field, value is constant 0
nlbits	4:1	Readable and writable	0	Bit used to specify the Level field in the clicintctl[i] register

				Number, see section 6.2.9 Learn more about it in detail.
Reserved	0	Read only, write ignore	N/A	Unused field, value is constant 1

6.2.5.2 Register clicinfo

The clicinfo register is a global information register that software can read to view several global parameters

For details on the bit field, seeTable 6-7 Table 6-7 Shown in .

设置

设置

Table 6-7 Bit field of register clicinfo

domain name	Bit	Attributes	Reset value	description
Reserved	31:25	Read only, write ignore	N/A	Unused field, value is constant 0
CLICINTCTLBITS	24:21	Read only, write ignore	N/A	The number of bits used to specify the valid bits in the clicintctl[i] register, see section 6.2.9 Learn more about it in detail.
VERSION	20:13	Read only, write ignore	N/A	Hardware implementation version number
NUM_INTERRUPT	12:0	Read only, write ignore	N/A	Number of interrupt sources supported by hardware

6.2.5.1 Register mth

The mth register is the threshold level register of the interrupt target. Software can rewrite the register to configure the threshold level of the interrupt target. For details of the specific bit field, seeTable 6-8 Table 6-8 Shown in .

Table 6-8 Bit field of register mth

domain name	Bit	Attributes	Reset value	description
mth	7:0	Readable and writable	N/A	Threshold level register for interrupt target, see section 6.2.11 Learn more about it Shao.

6.2.5.2 Register clicintip[i]

The clicintip[i] register is the wait flag register of the interrupt source. For details of the specific bit field, seeTable 6-9 Table 6-9 Central

示。

Table 6-9 Bit field of register clicintip[i]

domain name	Bit	Attributes	Reset value	description
Reserved	7:1	Read only, write ignore	N/A	Unused field, value is constant 0
IP	0	Readable and writable	0	Waiting flag bit of the interrupt source, see section 6.2.7 Learn more about it in detail.

6.2.5.3 Register clicintie[i]

The clicintie[i] register is the enable register of the interrupt source. For details of the specific bit field, seeTable 6-10 Table 6-10 Shown in .

设置

设置

Table 6-10 Bit field of register clicintip[i]

domain name	Bit	Attributes	Reset value	description
Reserved	7:1	Read only, write ignore	N/A	Unused field, value is constant 0
IE	0	Readable and writable	0	The enable bit of the interrupt source, see section 6.2.6 Learn more about it in detail.

6.2.5.4 Register clicintattr[i]

The clicintattr[i] register is the attribute register of the interrupt source. Software can rewrite several registers of the interrupt source by rewriting this register.

Sex, see the specific bit field information. Table 6-11 Table 6-11 Shown in .

设置

设置

Table 6-11 Bit fields of the register clicintattr[i]

doma in name	Bit	Attribut es	Reset value	desc ript ion

Reserved	7:6	Read only, write ignore	N/A	Unused field, value is constant 3
Reserved	5:3	Read only, write ignore	N/A	Unused field, value is constant 0
trig	2:1	Readable and writable	0	Specify the level or edge attribute of the interrupt source, See section 6.2.8 Learn more about it in detail.
shv	0	Readable and writable	0	Specify whether the interrupt source uses vector processing mode or non-vector processing mode, see section 6.2.10 Learn more about it in detail.

6.2.5.5 Register clicintctl[i]

The clicintctl[i] register is the control register for the interrupt source. Software can override this register to configure the level of the interrupt source.

(Level) and Priority (Priority), the Level and Priority fields are dynamically allocated according to the value of cliccfg.nlbits, see section **6.2.9** Learn more about it in detail.

6.2.6. Eclic interrupt source enable bit (ie)

Such asFigure 6-2 Figure 6-2 As shown, eclic assigns an interrupt enable bit (ie) to each interrupt source, reflected in the register.

设置

In clicintie[i].IE, its functions are as follows:

设

置

- The clicintie[i] register of each interrupt source is a readable and writable register of the memory address map so that software can program it.
- If the clicintie[i] register is programmed to 0, it means that this interrupt source is masked.
- If the clicintie[i] register is programmed to be 1, it means that this interrupt source is turned on.

6.2.7. Eclic interrupt source wait flag (ip)

Such asFigure 6-2 Figure 6-2 As shown, eclic assigns an interrupt wait flag (ip) to each interrupt source, reflected in the registration.

设置

In clicintip[i].IP, its function is as follows:

设

置

- If the ip bit of an interrupt source is high, it indicates that the interrupt source is triggered.The trigger condition of the interrupt source depends on whether it is a level-triggered or edge-triggered attribute, see section 6.2.8 Detailed introduction of the section.
- The ip bit software of the interrupt source is readable and writable. The behavior of the software to write the ip bit depends on whether it is a level-triggered or edge-triggered attribute. 6.2.8 Detailed introduction of the section.
- For edge-triggered interrupt sources, their ip may also have hardware self-clearing behavior, see section 6.2.8 Detailed introduction of the section.

6.2.8. Level or Edge-Triggered of the ECLIC interrupt source

Such asFigure 6-2 Figure 6-2 As shown, each interrupt source of eclic can be set to level trigger or edge triggered attributes (through

The trig field of the register clicintattr[i], the main points are as follows:

设置

设置

设

- When clicintattr[i].trig[0] = 0, set the interrupt attribute to a level-triggered interrupt:
  - If the interrupt source is configured for level triggering, the ip bit of the interrupt source will reflect the level of the interrupt source in real time.
  - If the interrupt source is configured as a level trigger, since the ip bit of the interrupt source reflects the level value of the interrupt source in real time, the software write operation of the interrupt ip bit is ignored, that is, the software cannot be set by the write operation or Clear the value of the ip bit.If the software needs to clear the interrupt, it can only be done by clearing the final source of the interrupt.
- When clicintattr[i].trig[0] = 1 and clicintattr[i].trig[1] = 0, set the interrupt attribute to a rising edge triggered interrupt:
  - If the interrupt source is configured as a rising edge trigger, when eclic detects the rising edge of the interrupt source, the interrupt source is triggered in eclic, and the ip bit of the interrupt source is set high.
  - If the interrupt source is configured for a rising edge trigger, software

writes to the interrupt ip bit, that is, the software can set or clear the value of the ip bit by a write operation.

- Note: For the rising edge triggered interrupt, in order to improve the efficiency of interrupt processing, when the interrupt is responded and the processor core jumps into the Interrupt Service Routines (ISR), the ECLIC hardware will automatically clear the interrupt. The interrupted IP bit eliminates the need to software clear the IP bit of the interrupt within the ISR.

■ When clicintattr[i].trig[0] = 1 and clicintattr[i].trig[1] = 1, set the interrupt attribute to the interrupt triggered by the falling edge:

- If the interrupt source is configured as a falling edge trigger, when eclic detects the falling edge of the interrupt source, the interrupt source is triggered in eclic and the ip bit of the interrupt source is set high.
- If the interrupt source is configured as a falling edge trigger, the software write to the interrupt ip bit will take effect, ie, the software can set or clear the value of the ip bit by a write operation.
- Note: For the interrupt triggered by the falling edge, in order to improve the efficiency of the interrupt processing, when the interrupt is responded and the processor core jumps into the Interrupt Service Routines (ISR), the ECLIC hardware will automatically clear the interrupt. The interrupted IP bit eliminates the need to software clear the IP bit of the interrupt within the ISR.

6.2.9. ECLIC interrupt source level and priority (Level and Priority)

设置

Such asFigure 6-2 Figure 6-2 As shown, each interrupt source of eclic can be set to a specific level and priority (via register

设置

Clicintctl[i]), the main points are as follows:

- The clicintctl[i] register of each interrupt source is theoretically 8 bits wide, and the real number of bits that the hardware actually implements is

  The clicinfoCTLBITS field of the clicinfo register is specified.For example, if the value of the clicinfo.CLICINTCTLBITS field is 6, it means that only the upper 6 bits of the clicintctl[i] register are true valid bits, and the lowest 2 bits are constant 1, such asFigure 6-3 Figure 6-3 The example in the example.

  - Note: The value of the clicintctlbits field is a read-only fixed constant that cannot be overwritten by software.The theoretically reasonable range is 2 <= clicintctlbits <= 8. The actual actual value is determined by the hardware implementation of the processor core.

- In the valid bit of the clicintctl[i] register, there are two dynamic fields, which are used to specify the level of the interrupt source.

(Level) and Priority (Priority).The width of the Level field is specified by the nlbits field of the cliccfg register.For example, if the value of the cliccfg.nlbits field is 4, it means that the upper 4 bits of the valid bit of the clicintctl[i] register are the Level field, and the other lower significant bits are the Priority field, such asFigure 6-3 Figure 6-3 The example in the example.

- Note: The value of the cliccfg.nlbits field is a readable and writable field that can be programmed by the software.

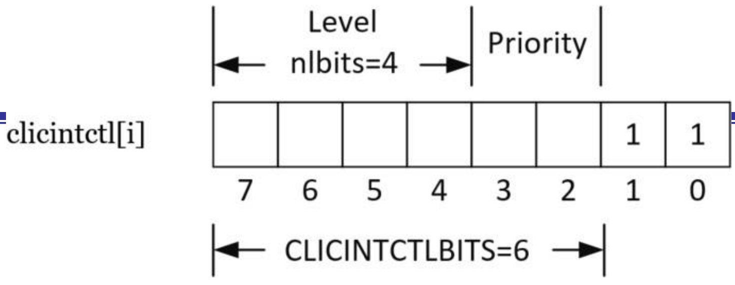


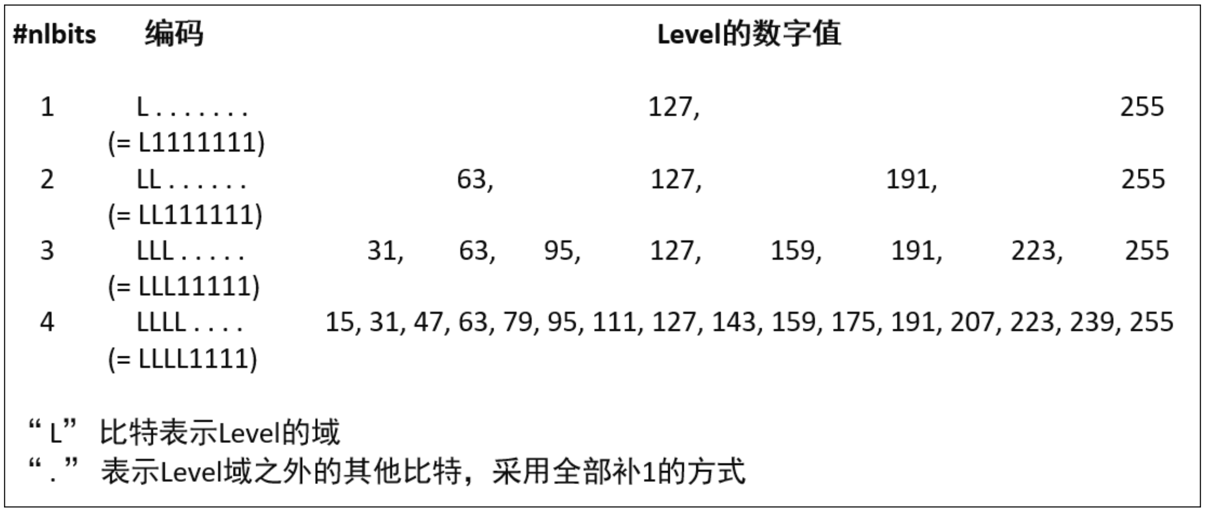
Figure 6-3 Example format of the register clicintctl[i]

- The main points related to the level of the interrupt source are as follows:
  - The digital values of Level are interpreted in a left-aligned manner, and the low bits except the effective bit width (specified by cliccfg.nlbits) are all filled with a constant constant of 1, such asFigure 6-4 Figure 6-4 The example in the example.
    - Note: If cliccfg.nlbits > clicinfo.CLICINTCTLBITS, it means that the number of bits indicated by nlbits exceeds the valid bit of the clicintctl[i] register, and the excess bits are all filled with the complement constant 1.
    - Note: If cliccfg.nlbits = 0, the numeric value of Level will be considered a fixed 255.Such asFigure 6-5 ~~Figure 6-5~~ The example in the example.
  - The higher the numeric value of Level, the higher its level. Note:

◆ High-level interrupts can interrupt low-level interrupt processing, resulting in interrupt nesting, see Section 5.11

Detailed introduction of the section.

◆ Multiple interrupts are waiting at the same time (IP bit is high). ECLIC needs to arbitrate to determine which interrupt is sent to the kernel for processing. The arbitration needs to refer to the Level

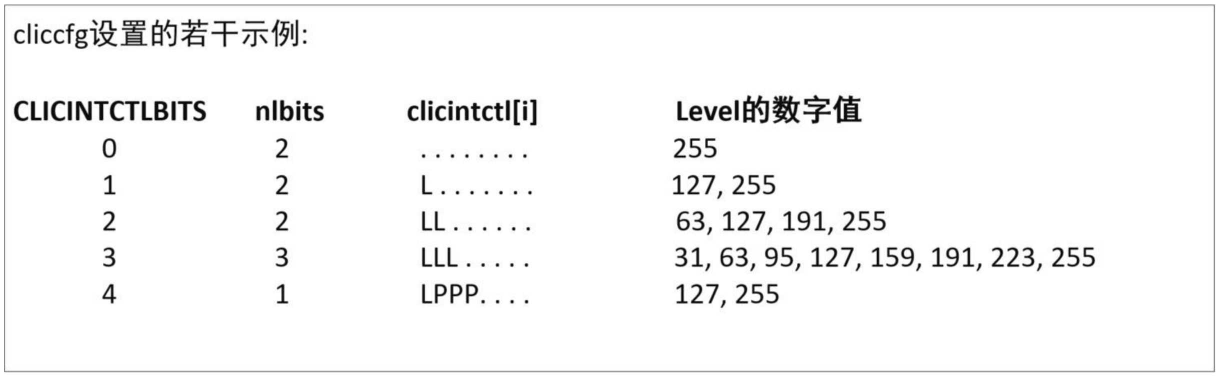


#nlbits	编码	Level的数字值
1	L . . . . (= L1111111)	127, 255
2	LL . . . . . . (= LL111111)	63, 127, 191, 255
3	LLL . . . . . (= LLL11111)	31, 63, 95, 127, 159, 191, 223, 255
4	LLLL . . . . (= LLLL1111)	15, 31, 47, 63, 79, 95, 111, 127, 143, 159, 175, 191, 207, 223, 239, 255

“L” 比特表示Level的域
“.” 表示Level域之外的其他比特，采用全部补1的方式

value of each interrupt source.See the detailed description in Section 5.5.

Figure 6-4 How to interpret the digital value of Level



cliccfg设置的若干示例:

CLICINTCTLBITS	nlbits	clicintctl[i]	Level的数字值
0	2	. . . .	255
1	2	L	127, 255
2	2	LL	63, 127, 191, 255
3	3	LLL	31, 63, 95, 127, 159, 191, 223, 255
4	1	LPPP. . . .	127, 255

Figure 6-5 Several examples of cliccfg settings

- The main points related to the priority of the interrupt source are as follows:
  - The digital value of Priority is also interpreted in a left-aligned manner, effective bit width

    The lower bits except (clicinfo.CLICINTCTLBITS - cliccfg.nlbits) are all filled with the complement constant 1.
  - The higher the numeric value of Priority, the higher its priority. Note:

- The priority of the interrupt does not participate in the judgment of interrupt nesting, that is, whether the interrupt can be nested and interrupted.

The value of (Priority) does not matter, but is related to the value of the level of the interrupt level.

- When multiple interrupts are simultaneously Pending, ECLIC needs to arbitrate to determine which interrupt is sent to the core for processing. The arbitration needs to refer to the Priority digital value of each interrupt source. See section 6.2.12 Detailed introduction of the section.

6.2.10. Vector or Non-Vector Mode of ECLIC Interrupt Sources

Each interrupt source of ECLIC can be set to vector or non-vector processing (via the shv field of the register clicintattr[i]). The main points are as follows:

- If configured as a vector processing mode, after the interrupt is responded by the processor core, the processor jumps directly into the target address stored in the vector table entry of the interrupt. For a detailed description of the interrupt vector processing mode, see Section 5.13.

- If configured to be a non-vector processing mode, then after the interrupt is responded by the processor core, the processor jumps directly into the entry address of all interrupt shares. For a detailed description of the interrupt non-vector processing mode, see Section 5.13.

6.2.11. Eclic interrupt target threshold level

Such as Figure 6-1 Figure 6-1 As shown in the figure, ECLIC can set the threshold level (mth) of a specific interrupt target, the main points are as follows:

设置 color,

- The mth register is a complete 8-bit register, all bits are readable and writable, software can write this register configuration

  设置

  Target threshold. Note: This threshold characterizes a level value.

  col

  or,

- The "level value" of the interrupt that ECLIC finally arbitrates is only higher than the value in the "mth register", and the interrupt can be sent to the processor core.

6.2.12. Arclic interrupt arbitration mechanism

Such as Figure 6-2 Figure 6-2 As shown, the principle that eclic arbitrates all of its interrupt sources is as follows:

设置 color,

- Only sources of interruption that meet all of the following conditions can participate in arbitration:

  设置 color,

  - The enable bit of the interrupt source (clicintie[i] register) must be 1.
  - The wait flag bit (clicintip[i] register) of the interrupt source must be 1.

- The rules for arbitration from all sources of interruption involved in arbitration are:

  - First, determine the level (Level), the larger the level value of the interrupt source, the higher the arbitration priority.
  - If the Levels are equal, then the Priority is determined. The larger the Priority number, the higher the arbitration priority.
  - If both Level and Priority are equal, the judgment interrupt ID is judged again, and the interrupt source with the larger interrupt ID has higher arbitration priority.

- If the level value of the interrupt source that was last arbitrated is higher than the threshold level (mth) of the interrupt target, a final interrupt request is generated, pulling the interrupt request signal to the processor core high.

6.2.13. Eclic interrupt response, nesting, tail biting mechanism

After the eclic interrupt request is sent to the processor core, the processor core will respond to it. Through eclic and kernel collaboration, you can support interrupt nesting, fast tail biting and other mechanisms. See Section 5.6, Section 5.11, Section 5.12 for details.

7. Bumblebee kernel CSR register introduction

7.1. Bumblebee Core CSR Register Overview

Some control and status registers (CSRs) are defined in the RISC-V architecture to configure or log the status of some operations. The CSR register is a register internal to the processor core and uses its proprietary 12-bit address encoding space.

7.2. Bumblebee kernel CSR register list

A list of CSR registers supported by the Bumblebee kernel, such asTable 7-1 Table 7-1 As shown, including the crisr of the risc-v standard

设置

Register (RV32IMAC architecture supports Machine Mode and User Mode related) and Bumblebee kernel custom expansion 设置

The csr register of the show.

Table 7-1 List of CSR Registers Supported by the Bumblebee Kernel

Types of	Csr address	Read and write properties	name	Full name
Risc-v standard csr （**Machine Mode**）	0xF11	MRO	mvendorid	Commercial Supplier Number Register (Machine Vendor ID Register)
	0xF12	MRO	marchid	Architecture Number Register (Machine Architecture ID Register)
	0xF13	MRO	mimpid	Hardware Implementation Number Register (Machine Implementation ID Register)
	0xF14	MRO	mhartid	Hart number register (Hart ID Register)
	0x300	MRW	mstatus	Exception handling status register
	0x301	MRO	misa	Instruction Set Architecture Register (Machine ISA) Register)
	0x304	MRW	mie	Local interrupt mask control register (Machine Interrupt Enable Register)
	0x305	MRW	mtvec	Exception entry base address register
	0x307	MRW	mtvt	The base address of the eclic interrupt vector table
	0x340	MRW	mscratch	Temporary register (Machine Scratch Register)
	0x341	MRW	mepc	Machine Exception Program Counter
	0x342	MRW	mcause	Abnormal Cause Register (Machine Cause Register)
	0x343	MRW	mtval	Outlier register (Machine Trap Value) Register)

	0x344	MRW	mip	Interrupt wait register (Machine Interrupt) Pending Register)
	0x345	MRW	mnxti	Standard registers are used to enable interrupts, processing the next one Break and return the handler entry address of the next interrupt
	0x346	MRO	mintstatus	Standard register is used to save the current interrupt level
	0x348	MRW	mscratchcsw	Standard registers are used to exchange when privileged mode changes Mscratch and destination register values
	0x349	MRW	mscratchcswl	Standard registers are used to exchange when the interrupt level changes Mscratch and destination register values
	0xB00	MRW	mcycle	Lower 32 bits of Cycle counter
	0xB80	MRW	mcycleh	The upper 32 bits of the cycle counter (Upper 32 bits of Cycle counter)
	0xB02	MRW	minstret	Complete the lower 32 bits of the instruction counter (Lower 32 bits of Instructions-retired counter)
	0xB82	MRW	minstreth	Complete the 32 bits of Instructions-retired counter
Risc-v standard csr **(User Mode)**	0xC00	URO	cycle	Read-only copy of the mcycle register Note: Whether this register is readable in User Mode is controlled by the CY bit field of the CSR register mcounteren, see section 7.4.29~~7.4.29~~ Understanding Details.
	0xC01	URO	time	Read-only copy of the mtime register Note: Is this register readable by the TMR register mcounteren in User Mode? Special domain to control, see section 7.4.29~~7.4.29~~ Festival Explain its details.
	0xC02	URO	instret	Read-only copy of the minstret register Note: Whether this register is readable in User Mode is controlled by the IR bit field of the CSR register mcounteren, see section 7.4.29~~7.4.29~~ Understanding Details.

设置

设置

设置

设置

设置

设置

	0x7d0	MRW	mmisc_ctl	Custom registers are used to control the handler of nmi Entry address
	0x7d6	MRW	msavestatus	Custom registers are used to hold mstatus values
	0x7d7	MRW	msaveepc1	Custom registers are used to save the first level nested nmi Or abnormal mepc
	0x7d8	MRW	msavecause1	Custom registers are used to save the first level nested nmi Or unusual mcause
	0x7d9	MRW	msaveepc2	Custom registers are used to hold the second level nested nmi Or abnormal mepc
	0x7da	MRW	msavecause2	Custom registers are used to hold the second level nested nmi Or unusual mcause
	0x7eb	MRW	pushmsubm	Custom registers are used to store the value of msubm in the heap Stack address space
	0x7ec	MRW	mtvt2	Custom registers are used to set non-vector interrupt handling Mode interrupt entry address
	0x7ed	MRW	jalmnxti	The custom register is used to enable the ECLIC interrupt. The read operation of this register can process the next interrupt and return the entry address of the next interrupt handler. Jump to this address.
	0x7ee	MRW	pushmcause	Custom registers are used to store the value of mcause in the stack address space
	0x7ef	MRW	pushmepc	Custom registers are used to store the value of mepc on the stack Address space
	0x811	MRW	sleepvalue	Wfi sleep mode register
	0x812	MRW	txevt	Send Event Register
	0x810	MRW	wfe	Wait for Event Control Register

note:

- MRW means Machine Mode Readable/Writeable
- MRO stands for Machine Mode Read-Only
- URW means User Mode Readable/Writeable
- URO stands for User Mode Read-Only

7.3. Access to the Bumblebee kernel's CSR register

The Bumblebee kernel has access to the CSR registers as follows:

- Whether in Machine Mode or User Mode:
  - If you read or write to a non-existing CSR register address range, an Illegal Instruction is generated.

    Exce
    ptio
    n.

- In Machine Mode:
  - Reading and writing to the csr register of the mrw or urw attribute is fine.

- Reading the csr register of the mro or uro attribute is fine.
- If you write to the CSR register of the MRO or URO attribute, an Illegal Instruction is generated.

Exception.

- In User Mode:
  - Reading and writing the csr register of the urw attribute is fine.
  - Reading the csr register of the uro attribute is fine.
    - ◆ Note: For the cycle, cycleh, time, timeh, instret, and intreth registers of the URO attribute, its readability is also controlled by the relevant bit field of mcounteren, see 7.4.29 Learn more about it.
  - If you write to the CSR register of the URO attribute, an Illegal Instruction Exception is generated.
  - If you read or write to the CSR register of the MRW or MRO attribute, an Illegal Instruction Exception is generated.

7.4. RISC-V standard CSR supported by the Bumblebee kernel

This section describes the Bumblebee kernel-defined CSR registers (the RV32IMAC architecture supports Machine Mode and User).

Mode related).

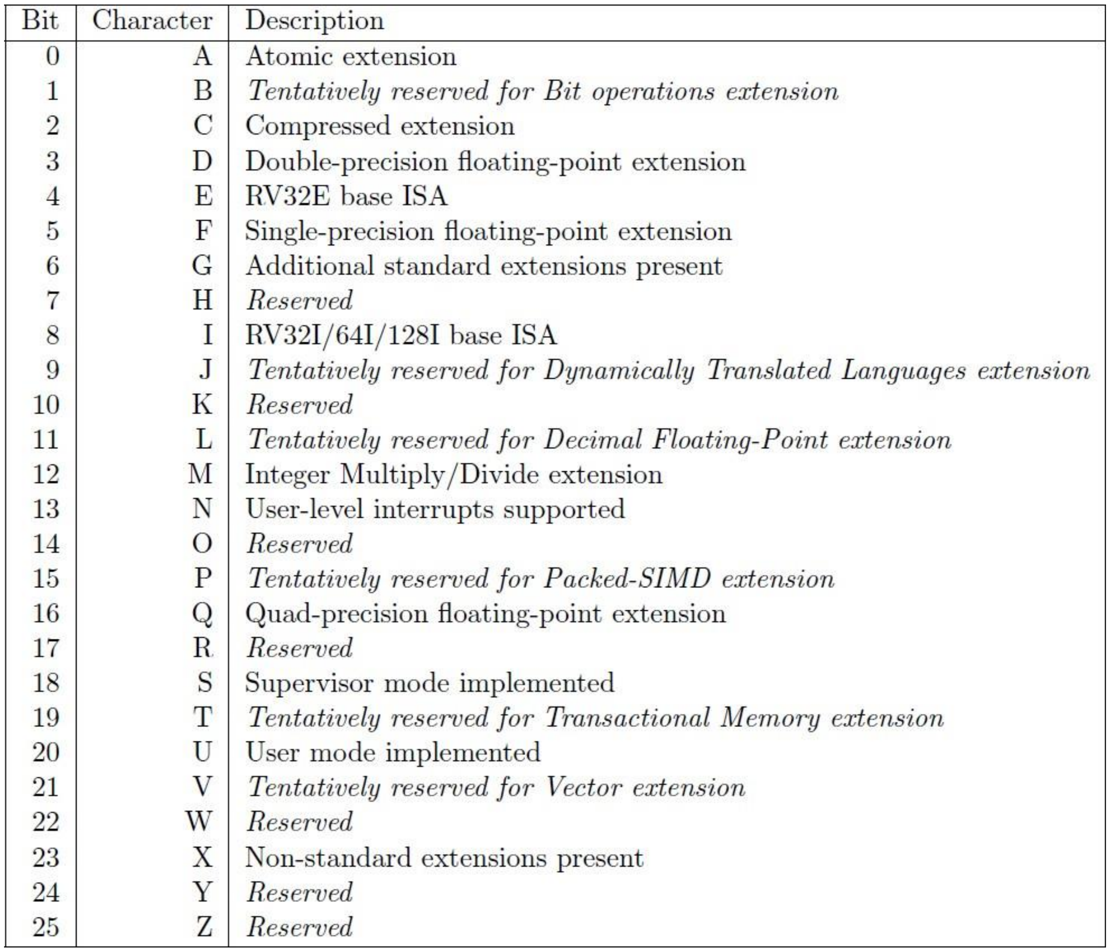
7.4.1. misa

The misa register is used to indicate the architectural characteristics supported by the current processor.

The highest two bits of the misa register are used to indicate the number of architecture bits supported by the current processor:

- If the highest two-bit value is 1, it means that it is currently a 32-bit architecture (rv32).
- If the highest two-bit value is 2, it means that the current 64-bit architecture (rv64).
- If the highest two-bit value is 3, it means that the current 128-bit architecture (rv128).

The lower 26 bits of the misa register are used to indicate a subset of different modular instructions in the RISC-V ISA supported by the current processor, each



Bit	Character	Description
0	A	Atomic extension
1	B	*Tentatively reserved for Bit operations 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	*Reserved*
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*

A representation of a modular instruction set such asFigure 7-1 Figure 7-1 Shown in . The other unused bit fields of this register are constant 0.

Figure 7-1 A subset of the modular instructions represented by the lower 26 bits of the misa register

Note: The misa register is defined as a readable and writable register in the RISC-V architecture document, allowing some processors to be designed to dynamically configure certain features. However, in the implementation of the Bumblebee kernel, the misa register is a read-only register that constantly reflects the ISA modular subset supported by different processor cores.

7.4.2. mie

The control bit of the mie register in ECLIC interrupt mode does not work, and reads mie returns all 0s.

7.4.3. mvendorid

This register is a read-only register that reflects the commercial vendor number (Vendor ID) of the processor core. If the value of this register is 0, this register is not implemented.

7.4.4. marchid

This register is a read-only

register that reflects the hardware implementation microarchitecture ID of the processor core. If the value of this register is 0, this register is not implemented.

7.4.5. mimpid

This register is a read-only register that reflects the hardware implementation number (Implementation ID) of the processor core. If the value of this register is 0, this register is not implemented.

7.4.6. mhartid

This register is a read-only register that reflects the current Hart number (Hart ID).

Hart (meaning "Hardware Thread") means a hardware thread. Multiple hardware threads may be implemented in a single processor core, such as hardware hyper-threading technology.

Each set of threads has its own independent register group and other context resources. However, most of the computing resources are multiplexed by all hardware threads, so the area efficiency is high. In such a hardware hyper-threading processor, there are multiple hardware threads (Hart) in one core.

The Hart number value in the Bumblebee kernel is controlled by the input signal core_mhartid. Note: The RISC-V architecture stipulates that if there is at least one Hart number in the single Hart or multi-Hart system, it must be 0.

7.4.7. mstatus

The mstatus register is the status register in Machine Mode. Each control bit field in the mstatus registerTable 7-2 Table 7-2 Shown.

Table 7-2 Control bits of the mstatus register

area	Bit	Reset value	description
Reserved	2:0	N/A	Unused field is constant 0
MIE	3	0	See section 7.4.8 Festival to learn more about it
Reserved	6:4	N/A	Unused field is constant 0
MPIE	7	0	See section 7.4.9~~7.4.9~~ Festival to learn more about it
Reserved	10:8	N/A	Unused field is constant 0
MPP	12:11	0	See section 7.4.9~~7.4.9~~ Festival to learn more about it
FS	14:13	0	See section 7.4.10~~7.4.10~~ Festival to learn more about it
XS	16:15	0	See section 7.4.11~~7.4.11~~ Festival to learn more

7.4.8. MIE domain of mstatus

The MIE field in the mstatus register indicates global interrupt enable:

When the value of the mie

field is 1, it indicates that

the global switch of the

interrupt is open, and the

interrupt can be responded

normally. When the value of

the mie field is 0, it

indicates that the interrupt

is globally closed, the

interrupt is masked, and the

response cannot be responded.

Note: When the Bumblebee core enters an exception, interrupt, or NMI processing mode, the value of the MIE is updated to 0 (meaning that the interrupt is masked after entering an exception, interrupt, or NMI processing mode).

7.4.9.MPIE and MPP domains for mstatus

The MPIE and MPP fields in the mstatus register are used to automatically save entry exceptions, before NMI and interrupts.

Automatic recovery when mstatus.MIE and privileged mode (Privilege Mode).

Update the hardware behavior of the mstatus registers MPIE and MPP fields when the Bumblebee kernel enters an exception, see 3.4.5

Learn more about it.

When the Bumblebee kernel exits the

exception (execute the mret instruction in exception handling mode), the mstatus register MPIE is updated.

For hardware behavior of the mpp domain see Section 3.5.2 for details.

Update the hardware behavior of the mstatus registers MPIE and MPP fields when the Bumblebee kernel enters the NMI, see 4.3.4

Learn more about it.

When the Bumblebee kernel exits the NMI (execute the mret instruction in exception handling mode), the mstatus register MPIE is updated.

For hardware behavior of the mpp domain see Section 4.4.2 for details.

Update the hardware behavior of the mstatus register MPIE and MPP fields when the Bumblebee kernel enters the interrupt, see 5.6.5

Learn more about it.

When the Bumblebee kernel exits the interrupt (execute the mret instruction in exception handling mode), the mstatus register MPIE is updated.

For hardware behavior of the mpp domain see Section 5.7.2 for details.

Note: The values of the

mstatus.MPIE and mstatus.MPP fields

are mirrored with the values of the

mcause.MPIE and mcause.MPP fields,

ie, under normal circumstances, the

value of the mstatus.MPIE field and

the value of the mcause.MPIE field

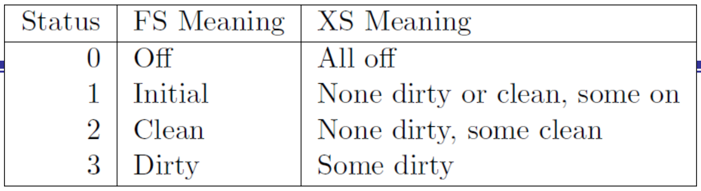
are always Is exactly the same,
The value of the mstatus.MPP field is always exactly the same as the value of the mcause.MPP field.

7.4.10. FS domain of mstatus

The FS field in the mstatus register is used to maintain or reflect the state of the floating point unit.

Note: This field will only exist if a floating point instruction (a subset of ″f″ or ″d″ instructions) is configured.

The fs field consists of two bits, and its encoding is shown in the following figure.



Status	FS Meaning	XS Meaning
0	Off	All off
1	Initial	None dirty or clean, some on
2	Clean	None dirty, some clean
3	Dirty	Some dirty

Figure 7-2 Status code represented by the fs field

The update criteria for the fs domain are as follows:

- The default value of FS after power-on is 0, which means that the state of the floating-point unit is Off.Therefore, in order to use the floating point unit normally, the software needs to use the CSR write instruction to rewrite the value of FS to a non-zero value to turn on the function of the floating point unit (FPU).
- If the value of FS is 1 or 2, the value of FS is automatically switched to 3 after any floating-point instructions are executed, indicating that the state of the floating-point unit is Dirty (the state has changed).
- If the processor does not want to use a floating-point unit (such as powering down the floating-point unit to save power), you can use the CSR write instruction to set the FS field of the mstatus register to 0 to turn off the function of the floating-point unit.Any operation that accesses the floating-point CSR register or any behavior that performs a floating-point instruction will cause an Illegal Instruction exception after the function of the floating-point unit is turned off.

In addition to the above functions, the value of the fs field is also used for the guidance information of the operating system when performing context switching. For interested users, please refer to the original risc-v ″privileged architecture document version 1.10″.

7.4.11. XS domain of mstatus

The XS field in the mstatus register is similar to the FS field, but it is used to maintain or reflect user-defined extended instruction unit status.

The XS field is defined as a read-only field in the standard RISC-V ″privileged architecture document version 1.10″, which is used to reflect the state sum of all custom extended instruction units. Note, however, that in the hardware implementation of the Bumblebee kernel, the XS domain is designed to be a writable readable domain, which acts like a FS domain. Software can override the value of the XS field to turn the coprocessor extension instruction unit on or off. purpose.

Similar to the fs domain, in addition to the above functions, xs is used for guidance information when the operating system performs context switching. For interested users, please refer to the risc-v ″privileged architecture document version 1.10″ original text.

7.4.12. SD domain of mstatus

The SD field in the mstatus register is a read-only field that reflects the Dirty state of the XS or FS domain. Its logical relational expression is: SD = ((FS==11) OR (XS==11)).

The reason for setting this read-only SD domain is to facilitate the software to quickly query whether the XS domain or the FS domain is in a

Dirty state, so that it can quickly determine whether a floating-point unit or an extended instruction unit needs context for context switching. save.

Interested users can refer to the original risc-v "privileged architecture document version 1.10".

7.4.13. mtvec

The mtvec register is used to configure the entry address for interrupts and exception handlers.

- The main points when mtvec configures the interrupted exception handler entry address are as follows:
  - The exception handler uses a 4byte aligned mtvec address (replace mtvec's lower 2bit with 0) as the entry address.
- The main points when mtvec configures the entry address of the interrupt program are as follows:
  - When mtvec.MODE != 6'b000011, the processor uses the "default interrupt mode".
  - This mode is recommended when the processor uses "ECLIC Interrupt Mode" when mtvec.MODE = 6'b000011.
    - The entry address and key points when the interrupt is in non-vector processing mode 5.13.2 As stated in the section.
    - The entry address and points when the interrupt is in vector processing mode are as described in Section 5.13.1.

Each address bit field of the mtvec registerTable 7-3 Table 7-3 Shown.

Table 7-3 mtvec Register Control Bits

area	Bit	description
ADDR	31:6	Mtvec address
MODE	5: 0	■ The mode field is the interrupt handling mode control field: • 000011: eclic interrupt mode (recommended mode) • Others: Default Interrupt Mode

7.4.14. mtvt

The mtvt register is used to hold the base address of the ECLIC interrupt vector table, which is at least 64byte aligned.

In order to improve the performance and reduce the number of hardware gates, the hardware determines the alignment of mtvt according to the number of interrupts actually implemented, such asTable 7-4 Table 7-4 Shown.

Table 7-4 mtvt alignment

Maximum number of interruption s	Mtvt alignment
0 to 16	64-byte
17 to 32	128-byte
33 to 64	256-byte
65 to 128	512-byte
129 to 256	1KB
257 to 512	2KB
513 to 1024	4KB
1025 to 2048	8KB
2045 to 4096	16KB

7.4.15. mscratch

The mscratch register is used by programs in Machine Mode to temporarily save certain data. The mscratch register provides a save and restore mechanism, such as temporarily storing the application's user stack pointer (SP) register in the mscratch register after entering the interrupt or exception handling mode, and then mscratch before exiting the exception handler. The value readout in the register is restored to the User Stack Pointer (SP) register.

7.4.16. mepc

The mepc register is used to hold the PC value that the processor is executing before entering the exception, as the return address of the exception. To understand

this register, see Chapter 3 for

a systematic understanding of

exceptions.

note:

- When the processor enters an exception, the mepc register is updated simultaneously to reflect the PC value of the instruction that is currently experiencing the exception.
- It is worth noting that although the mepc register is automatically updated by hardware when an exception occurs, the mepc register itself is a readable and writable register (in Machine Mode), so the software can also write this register directly to modify its value. .

Each address bit field of the mepc registerTable 7-5 Table 7-5 Shown.

Table 7-5 Control bits of the mepc register

area	Bit	desc ript ion
EPC	31: 1	Saves the pc value of the instruction that the processor is executing before the exception occurs.
Reserved	0	Unused field is constant 0

7.4.17. mcause

The mcause register is used to save the cause of the error before entering the NMI, exception, and interrupt, in order to diagnose and debug the cause of the trap.

Each address field of the mcause register isTable 7-6 Table 7-6 Shown.

Table 7-6 Control bits of the mcause register

area	Bit	desc ript ion

INTERRUPT	31	Indicates the current Trap type: ■ 0: abnormal or nmi ■ 1: interrupt
MINHV	30	Indicates that the processor is reading the interrupt vector table
MPP	29:28	Enter the privileged mode before the interrupt, the same as mstatus.mpp
MPIE	27	Interrupt before entering interrupt is enabled, same as mstatus.mpie
Reserved	26:24	Unused field is constant 0
MPIL	23:16	Previous interrupt level
Reserved	15:12	Unused field is constant 0
EXCCODE	11:0	Exception/interrupt coding

note:

- The MPIE and MPP fields of the mstatus register are mirrored to the MPIE and MPP fields of mcause.
- NMI's mcause.EXCCODE may be 0x1 or 0xfff, and the actual value is controlled by mmisc_ctl. For details, please refer to 7.5.4 Section.

7.4.18. mtval （mbadaddr）

The mtval register (aka mbadaddr, which is recognized by some versions of the toolchain) is used to store the encoded value of the error instruction before entering the exception or the address value of the memory access, in order to diagnose and debug the cause of the exception.

To understand this register, see Chapter 3 for a systematic understanding of exceptions.

When the Bumblebee kernel enters an

exception, the mtval register is updated simultaneously to reflect the current exception.

7.4.19. mip

The control bit of the mip register in ECLIC interrupt mode does not work, and the read mip returns all 0s.

7.4.20. mnxti

Mnxti (Next Interrupt Handler Address and Interrupt-Enable CSR) can be accessed by software to handle the next interrupt in the same Privilege Mode without causing flushing pipelines and context save recovery.

The mnxti register is accessed by the CSRRSI/CSRRCI instruction. The read return value is the handler address of the next interrupt, and the mnxti writeback operation updates the interrupt enable state.

note:

1 For interrupts of different Privilege Modes, the hardware is handled as interrupt nesting, so mnxti will only process the next interrupt in the same Privilege Mode.

2 The mnxti register is not the same as the regular CSR instruction, and its return value is the RMW of the regular register.

The value of the (read-modify-write) operation is different:

- There are two cases where the return value of a mnxti CSR read operation is as follows:

  ◆ The return value is 0 when the following occurs.

    - No interrupts that can respond

- The highest priority interrupt is vector interrupt

- When the interrupt is a non-vector interrupt, the interrupt handler entry address of this interrupt is returned.

- The mnxti CSR write updates the following registers and register fields:
  - Mstatus is the destination register for the current RMW (read-modify-write) operation
  - The mcause.EXCCODE field and the interrupt id that will be updated to the current response interrupt, respectively
  - The mintstatus.MIL field is updated to the interrupt level (Level) of the current response interrupt.

7.4.21. mintstatus

The mintstatus register holds the interrupt level for valid interrupts in each Privilege Mode.

Table 7-7 Control bits of the minstatus register

area	Bit	description
MIL	31:24	Effective interrupt level for Machine Mode
Reserved	23: 8	Unused field is constant 0
UIL	7:0	User Mode effective interrupt level

7.4.22. mscratchcsw

The mscratchcsw register is used

to exchange the destination register with the value of mscratch to speed up interrupt handling when switching between multiple privileged modes.

Use the CSR instruction with a read to access mscratchcsw. When the privileged mode is inconsistent before and after the interrupt occurs, there are register operations as shown in the following directive:

```
csrrw rd, mscratchcsw, rs1

// Pseudocode operation.
if  (mcause.mpp!=M-mode) then {
    t = rs1; rd = mscratch; mscratch = t;
} else {
    rd = rs1; // mscratch unchanged.
}
```

```
        // Usual use: csrrw sp, mscratchcsw, sp
```

The processor interrupts in the Privilege Mode, the processor enters the high privileged mode to handle the interrupt, and when processing the interrupt, the stack is needed to save the state of the processor before entering the interrupt. At this time, if you continue to use the stack pointer (SP) in low privileged mode, the data of the stack in the high privileged mode will be stored in the interval that the low privileged mode can access, resulting in the security of high privileged mode data leakage to the low privileged mode. Vulnerabilities.To avoid this vulnerability, the RISC-V architecture specifies that when the processor is in low privileged mode, the stack pointer (SP) of the high privileged mode needs to be saved to the mscratch register, so that after entering the high privileged mode, the processor can use the mscratch register. The value of the stack pointer (SP) to restore the high privileged mode.

Using the normal instructions to execute the above program requires more cycles. The mscratchcsw register is defined for this RISC-V architecture. The mscratchcsw register instruction is executed immediately after the interrupt is entered, and the values of mscratch and SP are exchanged to restore the stack of the high privileged mode. Pointer (SP) while backing up the low privileged mode stack pointer (SP) to the mscratch register.Before executing the mret instruction to exit the interrupt, a mscratchcsw instruction is also added to exchange the value of the mscratch register and the stack pointer (SP), and the stack pointer (SP) of the high privileged mode is backed up to mscratch again, and the stack pointer of the low privileged mode is restored. (SP). In this way, only two instructions are needed to solve the stack pointer (SP) switching problem of different privileged modes, which speeds up interrupt processing.

Note: To avoid virtualization vulnerabilities, the software cannot directly read the processor’s current privilege mode (Privilege Mode). If the software attempts to access the mscratchcsw register swap operation in a given privileged mode in a lower privileged mode, causing the processor to enter a trap, mscratchcsw does not cause a virtualization vulnerability.

7.4.23. mscratchcswl

The mscratchcswl register is used to exchange the destination register with the value of mscratch to speed up interrupt handling when switching between multiple interrupt levels.

Use the CSR instruction with read to access mscratchcsw. When the privileged mode is unchanged, when there is a switch between the interrupt program and the application, there are register operations as shown in the following directive:

```
csrrw rd, mscratchcswl, rs1

  // Pseudocode operation.
  if  ( (mcause.mpil==0) !=  (mintstatus.mil == 0) ) then
      { t = rs1; rd = mscratch; mscratch = t;
  } else {
      rd = rs1; // mscratch unchanged.
  }
  // Usual use: csrrw sp, mscratchcswl, sp
```

In single privileged mode, separating the interrupt handler task from the stack space of the application task can increase robustness, reduce space usage, and facilitate system debugging. The interrupt handler task has a non-zero interrupt level, while the application task has a zero interrupt level, according to which the mscratchcswl register is defined by the RISC-V architecture. Similar to mscratchcsw, adding a mscratchcswl to the interrupt program entry and exit respectively enables a fast stack pointer switch between the interrupt handler and the application, ensuring separation of the stack space between the interrupt handler and the application.

7.4.24. Mcycle and mcycleh

The risc-v architecture defines a 64-bit wide clock cycle counter that reflects how many clock cycles the processor has executed. This counter continuously increments as long as the processor is in the execution state.

The mcycle register reflects the lower 32 bits of the counter, and the mcycleh register reflects the 32-bit high value of the counter.

The mcycle and mcycleh registers can be used to measure processor performance and have readable and writable properties, so software can override the values in the mcycle and mcycleh registers with CSR instructions.

In the implementation of the Bumblebee

kernel, it is customizable because it takes into account that this counter count consumes some dynamic power.

An additional control field is added to the CSR register mcountinhibit. The software can configure this control field to stop counting the counters corresponding to mcycle and mcycleh, so that the counter can be stopped to save power when performance is not required. See **7.5.1** Learn more about mcountinhibit register information.

Note: This counter does not count

if in debug mode, and the counter will only count in normal function mode.

7.4.25. Minstret and minstreth

The risc-v architecture defines a 64-bit wide instruction completion counter that reflects how many instructions the processor successfully executed. This counter increments as long as the processor completes an instruction every successful execution.

The minstret register reflects the

lower 32 bits of the counter, and the minstreth register reflects the 32-bit high value of the counter.

The minstret and minstreth registers can be used to measure processor performance and have readable and writable properties, so software can override the values in the minstret and minstreth registers with CSR instructions.

Since this counter count consumes some dynamic power, in the implementation of the Bumblebee kernel, an additional control field is added to the custom CSR register mcountinhibit, and the software can configure this control field to minstret and

The counter corresponding to minstreth stops counting, so that the counter is stopped to save power when performance is not required. See **7.5.1** Learn more about mcountinhibit register information.

Note: This counter does not count if in debug mode, and the counter will only count in normal function mode.

7.4.26. Cycle and cycleh

Cycle and cycleh are read-only copies of mcycle and mcycleh, respectively. Is this register readable by User Mode?

The CY bit field of the CSR register mcounteren is controlled, see section **7.4.29** Learn more about it.

7.4.27. Instret and intreth

Instret and intreth are read-only copies of minstret and minstreth, respectively.Whether this register is readable in User Mode is controlled by the IR bit field of the CSR register mcounteren, see section **7.4.29** Learn more about it.

7.4.28. Time and timeh

Time and timeh are read-only copies of mtime and mtimeh, respectively.Is this register readable by User Mode?

The TM bit field of the CSR register mcounteren is controlled, see section **7.4.29** Learn more about it.

7.4.29. mcounteren

This register will only exist if it is configured to support User Mode.Each control bit field in the mcounteren registertable

7-8 Table 7-8 Shown.

Table 7-8 mCounteren Register Control Bits

area	Bit	description
CY	0	This bit controls whether the cycle and cycleh registers can be accessed in User Mode: ■ If this bit is 1, the cycle and cycleh can be accessed normally in User Mode. ■ If this is 0, accessing cycle and cycleh in User Mode will trigger illegal instruction. Exception . Thi s bit resets the default value to 0.
TM	1	This bit controls whether the time and timeh registers can be accessed in User Mode: ■ If this bit is 1, the time and timeh can be accessed normally in User Mode. ■ If this is 0, accessing time and timeh in User Mode will trigger illegal instruction. Exception. This bit resets the default value to 0.
IR	2	This bit controls whether the instret and intreth registers can be accessed in User Mode: ■ If this bit is 1, the instret and instreth can be accessed normally in User Mode. ■ If this is 0, accessing instret and instreth in User Mode will trigger illegal instruction. Exception. This bit resets the default value to 0.
Reserv ed	3~31	Other unused fields are constants 0

7.5. Bumblebee kernel custom CSR

This section describes the CFF registers customized by the Bumblebee kernel.

7.5.1. mcountinhibit

The mcountinhibit register is used to

control the count of mcycle and minstret, each control bit field such asTable 7-9 Table 7-9 Shown.

Table 7-9 Control bits of mcountinhibit register

area	Bit	description
Reserved	31:3	Unused field is constant 0
IR	2	The count of minstret is turned off when IR is 1.
Reserved	1	Unused field is constant 0
CY	0	Count of mcycle is turned off when CY is 1

7.5.2. mnvec

The mnvec register is used to configure the entry address of the NMI

In order to understand this register, please refer to Chapter 4 for a systematic understanding of nmi.

During the execution of the processor program, once the NMI encounters, the current program flow is terminated, the processor is forced to jump to a new PC address, and the PC address jumped into the NMI after the Bumblebee kernel enters the NMI is specified by the mnvec register. .

Note: The value of mnvec is controlled

by mmisc_ctl. For more details, please refer to **7.5.4** Section.

7.5.3. msubm

The Bumblebee kernel custom msubm register is used to hold the Trap type before and after the Trap.

Each control bit field in the msubm register is asTable 7-10 Table 7-10 Shown.

Table 7-10 msubm Register Control Bits

area	Bit	description
Reserved	31:10	Unused field is constant 0
PTYP	9:8	Save the Trap type before entering Trap: ■ 0: Non-Trap status ■ 1: interrupt ■ 2: abnormal ■ 3: NMI
TYP	7:6	Indicates the current Trap type of the Core: ■ 0: Non-Trap status ■ 1: interrupt ■ 2: abnormal ■ 3: NMI
Reserved	5:0	Unused field is constant 0

7.5.4. mmisc_ctl

The Bumblebee kernel custom mmisc_ctl register is used to control the mcause values of mnvec and NMI.

Each control bit field in the mmisc_ctl registerTable 7-11 Table 7-11 Shown.

设置

设置

Table 7-11 Control bits of the mmisc_ctl register

area	Bit	desc ript ion
Reserved	31:10	Unused field is constant 0
NMI_CAUSE_ FFF	9	Control mcause.EXCCODE for mnvec and NMI: ■ 0: The value of mnvec is equal to the PC after processor reset, NMI mcause.EXCCODE is 0x1 ■ 1: The value of mnvec is the same as mtvec, and the mcause.EXCCODE of NMI is 0xfff
Reserved	8:0	Unused field is constant 0

7.5.5. msavestatus

Msavestatus is used to store the values of mstatus and msubm to ensure that the state of each domain of mstatus and msubm is not flushed by NMI or exceptions. Msavestatus has a two-stage stack that supports up to 3 levels of exception/NMI state saving. For more two-level NMI/Exception Status Stack, see

Section 4.6.

Each control bit of the msavestatus register is asTable 7-12 Table 7-12 Shown.

设置

设置

Table 7-12 msavestatus Register Control Bits

area	Bit	description
Reserved	31:16	Unused field is constant 0
PTYP2	15:14	Second-level nested NMI/Trap type before exception occurs
Reserved	13:11	Unused field is constant 0
MPP2	10:9	Second-level nested NMI/Privilege mode before exception occurs
MPIE2	8	The second level of nested nmi / interrupt enable state before the exception occurs
PTYP1	7:6	The first level of nested NMI / Trap type before the exception occurs
Reserved	5:3	Unused field is constant 0
MPP1	2:1	The first level of nested NMI / Privilege mode before the exception occurs
MPIE1	0	The first level of nested nmi / interrupt enable state before the exception occurs

7.5.6. Msaveepc1 and msaveepc2

Msaveepc1 and msaveepc2 are used as the primary NMI/exception state stack and the secondary

NMI/exception state stack, respectively, to store the PC before the first-level nested NMI/abnormality, and the PC before the second-level nested NMI/abnormality. .

- msaveepc2 <= msaveepc1 <= mepc <= interrupted PC <= NMI/exception PC

When executing the mret instruction, while mcause.INTERRUPT is 0 (eg NMI, or exception), msaveepc1 and Msaveepc2 recovers the processor's PC through the primary and secondary NMI/Exception Status Stacks, respectively.

- msaveepc2 => msaveepc1 => mepc => PC

7.5.7. Msavecause1 and msavecause2

Msavecause1 and msavecause2 are used as a level 1 NMI/Exception State Stack and a Level 2 NMI/Exception State Stack, respectively, to store the first level of nested NMI/mcause before the exception occurs, and the second level of nested NMI/mcause before the exception occurs. .

- msavecause2 <= msavecause1 <= mcause <= NMI/exception cause

When executing the mret instruction while mcause. INTERRUPT is 0 (eg NMI, or exception), msavecause1

And msavecause2 restores the mcause state through the primary and secondary NMI/Exception State Stacks, respectively.

- msavecause2 => msavecause1 => mcause

7.5.8. pushmsubm

The Bumblebee kernel defines CSR instructions implemented by the csrrwi operation of the pushmsubm

register, storing
The value of msubm

to the stack

pointer is used as

the memory space of

the base address.

This CSR

instruction is

described by taking

the following

instruction as an

example:

```
csrrwi x0, PUSHMSUBM, 1
```

The operation of this instruction is to store the value of the msubm register to the address of SP (stack pointer) + 1 * 4.

7.5.9. mtvt2

Mtvt2 is used to specify the interrupt common-code entry address for ECLIC non-vector mode.

Each control bit field in the mtvt2 register is asTable 7-13 Table 7-13 Shown.

Table 7-13 mtvt2 register control bits

area	Bit	description
CMMON-COD E-ENTRY	31:2	This field determines the ECLIC non-vector mode interrupt when mtvt2.MTVT2EN=1 Common-code entry address.
Reserved	1	Unused field is constant 0
MTVT2EN	0	Mtvt2 enable bit: ■ 0: ECLIC non-vector mode interrupt common-code entry address by mtvec Decide ■ 1: ECLIC non-vector mode interrupt common-code entry address by mtvt2.COMMON-CODE-ENTRY decision

7.5.10. jalmnxti

The Bumblebee kernel defines the jalmnxti register to reduce interrupt latency and speed up interrupt tail biting.

Jalmnxti In addition to the mnxti enable interrupt enable, processing the next interrupt, return to the next interrupt entry address and other functions, there is a jump to the interrupt handler function, so the number of interrupt processing instructions can be shortened, reducing interrupts Delay, speed up the purpose of interrupting the tail bite. For more details on jalmnxti please see **5.13.1.3** Section.

7.5.11. pushmcause

The Bumblebee kernel defines a CSR instruction implemented by the csrrwi operation of the pushmcause register, storing

The value of mcause

to the stack

pointer is used as

the memory space of

the base address.

This CSR

instruction is

described by taking

the following

instruction as an

example:

```
csrrwi x0, PUSHMCAUSE, 1
```

The operation of this instruction is to store the value of the mcause register to the address of SP (stack pointer) + 1 * 4.

7.5.12. pushmepc

The Bumblebee kernel defines a CSR instruction implemented by the csrrwi operation of the pushmepc register, storing the mepc

The value of the stack pointer to the memory space of the base address

Introduce this csr instruction with the following instruction as an example:

```
csrrwi x0, PUSHMPEC, 1
```

The operation of this instruction is to store the value of the mepc register to the address of SP (stack pointer) + 1 * 4.

7.5.13. sleepvalue

The Bumblebee kernel customizes a CSR register sleepvalue to control different sleep modes, see section **8.1** For more details. Each control bit field in the sleepvalue registerTable 7-14 Table 7-14 Shown.

Table 7-14 sleepvalue register control bits

area	Bit	desc ript ion
SLEEPVALUE	0	Control wfi sleep mode ■ 0: Shallow sleep mode (the main working clock of the processor core after executing wfi Core_clk is closed) ■ 1: Deep sleep mode (the main working clock of the processor core after executing wfi Core_clk and the processor core's normally open clock

		core_aon_clk are both turned off) This bit resets the default value to 0.
Reserved	31:1	Unused field is constant 0

7.5.14. txevt

The Bumblebee kernel has a custom CSR register, txevt, for sending events to the outside world.

Each control bit field in the txevt registerTable 7-15 Table 7-15 Shown.

Table 7-15 txevt Register Control Bits

area	Bit	description
TXEVT	0	Control to send Event: ■ If you write 1 to this bit, it will trigger the output signal tx_evt of the Bumblebee kernel. A single-cycle pulse signal is generated as an external Event signal. ■ This bit is a self-clearing bit, that is, after a 1 is written to this bit, it is self-cleared to 0 in the next cycle. ■ Writing 0 to this bit has no reaction or operation. This bit resets the default value to 0.
Reserved	31:1	Unused field is constant 0

7.5.15. wfe

The Bumblebee kernel has a custom CSR register wfe that controls whether the WFI instruction wakes up using an interrupt or an Event. See section **8.2.3** For more details.

Each control bit field in the wfe registerTable 7-16 Table 7-16 Shown.

Table 7-16 Control bits of the wfe register

area	Bit	description
WFE	0	The wake-up condition that controls the WFI instruction is whether to use an interrupt or an Event. ■ 0: The processor core can be interrupted and nmi wake up when it enters sleep mode. ■ 1: When the processor core enters sleep mode, it can be woken up by Event and NMI. This bit resets to a default of 0.
Reserved	31:1	Unused field is constant 0

8. Introduction to the Bumblebee core low power mechanism

The Bumblebee core can support sleep mode for lower static power consumption.

8.1. Go to sleep

The Bumblebee kernel can go to sleep through the WFI instruction. When the processor executes the WFI instruction it will:

- Immediately stop executing the current instruction stream;
- Waiting for the processor core to complete any outstanding outstanding operations (Outstanding Transactions), such as fetching instructions and data read and write operations, to ensure that the operations sent to the bus are completed;
  - Note: If a memory access error exception occurs while waiting for an operation on the bus to complete, it will enter the exception handling mode without sleeping.
- When all of the Outstanding Transactions are completed, the processor safely enters an idle state, which can be referred to as a "sleep" state.
- When entering sleep mode:

- The clocks of the main units inside the Bumblebee core will be gated off to save static power consumption;
- The output signal core_wfi_mode of the Bumblebee kernel is pulled high, indicating that this processor core is executing WFI

The sleep state after the instruction;

- The Bumblebee kernel's output signal core_sleep_value will output the value of the CSR register sleepvalue (note: this signal is only valid when the core_wfi_mode signal is high; the core_sleep_value must be 0 when the core_wfi_mode signal is low).Software can configure the CSR register by prior

Sleepvalue to indicate a different sleep mode (0 or 1).note:

- ◆ The Bumblebee kernel behaves exactly the same for different sleep modes.This sleep mode is only for the corresponding control of the PMU (Power Management Unit) at the SoC system level.

8.2. Exit hibernation

The main points of the Bumblebee kernel processor exiting sleep mode are as follows:

- The output signal core_wfi_mode of the Bumblebee kernel is pulled low accordingly.
- The Bumblebee kernel processor can be woken up in four ways:

  - NMI
  - Interrupt
  - Event
  - Debug request

This will be described in detail below

8.2.1. Nmi wake up

NMI can always wake up the processor core. When the processor core detects a rising edge of the input signal nmi, the processor core wakes up and proceeds to the NMI service routine to begin execution.

8.2.2. Interrupt wakeup

Interrupts can also wake up the processor core:

- If the mstatus.MIE field is configured to 1 (indicating that the global interrupt is turned on):
  - When eclic (by arbitrating an externally requested interrupt) sends an interrupt to the processor core, the processor core wakes up and proceeds to the interrupt service routine to begin execution.
- If the mstatus.MIE field is configured to 0 (indicating that the global interrupt is closed):
  - If the CSR register wfe.WFE field is configured to 0, then:
    - When eclic (by arbitrating an externally requested interrupt) sends an interrupt to the processor core, the processor core wakes up and continues to execute the previously stopped instruction stream (instead of entering the interrupt service routine).

- If the CSR register wfe.WFE field is configured to 1, then wait for Event wakeup, see the description in the next section.

8.2.3. Event wake up

Event can wake up the processor core when the following conditions are met:

- If the mstatus.MIE field is configured to 0 (indicating that the global interrupt is turned off) and the CSR register wfe.WFE field is configured to 1, then:
    - When the processor core detects that the input signal rx_evt (called the Event signal) is high, the processor core wakes up and continues executing the previously stopped instruction stream (instead of entering the interrupt service routine).

8.2.4. Debug wake up

The Debug request always wakes up the processor core. If the debugger is Debugged, it will wake up the processor core and enter debug mode.

8.3. Wait for Interrupt mechanism

The Wait for Interrupt mechanism refers to putting the processor core into sleep mode, then waiting for the interrupt to wake up the processor core, and waking up to the corresponding interrupt handler.

Such as the first **8.1** Festival and section 8.2~~8.2~~ As mentioned in the section, the Wait for Interrupt mechanism can directly pass the WFI instruction (cooperate 设置

The mstatus.MIE field is configured to 1) complete.

8.4. Wait for Event mechanism

The Wait for Event mechanism

refers to putting the processor core

into sleep mode, then waiting for Event to wake up the processor core, and waking up to continue the previously stopped program (instead of entering the interrupt handler).

Such as the first **8.1** Festival and section **8.2~~8.2~~** As mentioned in the section, the Wait for Event mechanism can directly pass the WFI instruction with the following instructions. 设置

Sequence completion:

Step 1: Configure the mstatus.MIE field
to 0 to turn off global interrupts Step
2: Configure the wfe.WFE domain to be 1
Step 3: Call the WFI instruction.After this instruction is called, the processor enters sleep mode and will continue to execute downwards when Event or NMI wakes up.Step 4: Restore the wfe.WFE field to 0

Step 5: Restore the mstatus.MIE field to the previous value