Nuclei ™ N200 Series

Processor Core Instruction Architecture Manual

Copyright Notice

contact us

If you have any questions, please email support@nucleisys.comContact Xinlai Technology.

revise history

version number	Revision date	Revised chapter	Revised content
1.0	5/12/2019	N / A	Initial version

Page 4

table of Contents

Page 5

Page 6

Page 7

Page 8

Page 9

Page 10

List of forms

Page 11

Picture list

1. N200 Series Kernel Instruction Set and CSR Introduction

1.1. Introduction to RISC-V instruction set

The standard RISC-V instruction set document version followed by the N200 series kernel is: "Instruction Set Document Version 2.2" (riscv-spec-v2.2.pdf). Users can follow the RISC-V Foundation's website to register and download the full text for free (Https://riscv.org/specifications/).

In addition to the original English version of RISC-V "Instruction Set Document Version 2.2", users can also refer to the Chinese book Appendix A and Appendix C ~ G of "Designing Your CPU — —RISC-V Processor", which uses plain Chinese to RISC-The V instruction set standard is systematically explained.

1.2. N200 series kernel supports instruction set

The RISC-V instruction set is based on a modular design and can be flexibly combined according to configuration. Nuclei N200 series kernel supports Is the following modular instruction set:

- RV32 architecture: 32-bit address space, 32-bit general-purpose register width.
- I: Support 32 general-purpose integer registers.
- M: Supports integer multiplication and division instructions
- C: Supports compression instructions with a length of 16 bits to increase code density.
- A: Supports atomic operation instructions.
- F: Support single precision floating point instruction.
- D: Supports double-precision floating-point instructions.

According to the RISC-V architecture naming rules, the combination of the above instruction subsets can be expressed as RV32IMAC, RV32IMFC, RV32IMAFDC, etc. The RISC-V architecture defines IMAFD as a General Purpose, which is represented by the letter G,

Therefore, RV32IMAFDC can also be expressed as RV32GC.

Note: The N200 is a processor core series. Each processor core (such as

N201, N205, etc.) The combination of supported instruction subsets may be slightly different (eg N201 only supports RV32IC), see

Nuclei_N200 Series Concise Data Sheet for details.

Page 13

1.3. CSR register

Control and Status Registers (CSR) are defined in the RISC-V architecture for

Configure or record the running status of some processor cores. The CSR register is a register inside the processor core, using its proprietary 12-bit

Address encoding space. Refer to section 7 chapter .

Page 14

2. Introduction to N200 Series Kernel Privileged Architecture

2.1. General Introduction

The standard RISC-V privileged architecture document version followed by the N200 series kernel is: "Privileged Architecture Document Version 1.10 privileged-v1.10.pdf). Users can follow the RISC-V Foundation's website to register and download the full original for free Article (https://riscv.org/specifications/).

In addition to the original RISC-V "Privileged Architecture Document Version 1.10", users can also refer to the Chinese book "Hands Teach you to design the CPU-RISC-V processor chapter "Appendix A, Appendix C ~ G part, which uses plain Chinese language The RISC-V privileged architecture standard provides a systematic explanation.

2.2. Privilege Modes

The N200 series kernel supports two Privilege Modes:

- Machine Mode is a required mode. The Privilege Mode code is 0x3.
- User Mode is a configurable mode. The Privilege Mode code is 0x0.
  - If User Mode is configured, it also supports PMP (Physical Memory Protection) unit to physical Address ranges are isolated and protected.

Note: N200 is a series of processor cores. Each processor core (such as N201, N205, etc.) supports The Privilege Modes combination may be slightly different, please refer to the Nuclei_N200 Series Concise Data Sheet for details situation.

2.2.1. Machine Mode

The key points of the N200 series core regarding Machine Mode are as follows:

- After the processor core is reset, it is in Machine Mode by default.
- In Machine Mode, the program can access all CSR registers.

- In Machine Mode, the program can access all physical address areas (except for areas that are not accessible by PMP) In addition, see Chapter 8 for details on PMP).

2.2.2. User Mode

The key points of the N200 series kernel regarding User Mode are as follows:

- In User Mode User Mode can only access the defined CSR register, see Section , 7 .3 section for details.
- In the User Mode can only access permissions set PMP physical address area, see page 8 chapter for the PMP Details.

2.2.3. Machine Sub-Mode

The Machine Mode of the N200 series kernel may be in four different states, which is called the machine sub-mode (Machine Sub-Mode):

- Normal machine mode (The encoding of this Machine Sub-Mode is 0x0):
  - After the processor core is reset, it is in this sub-mode. If no exception occurs after the processor resets, NMI, interrupt, always run under this mode.
- Exception Handling Mode (The encoding of this Machine Sub-Mode is 0x2):
  - The processor core is in this state after responding to an exception.
  - For details on the exception mechanism, see Chapter 3 .
- NMI processing mode (The encoding of this Machine Sub-Mode is 0x3):
  - The processor core is in this state after responding to the NMI.
  - For details on the NMI mechanism, see Chapter 4 .
- Interrupt processing mode (The code of the Machine Sub-Mode is 0x1):
  - The processor core is in this state after responding to an interrupt.
  - For details on the interrupt mechanism, see Chapter 5 .

The Machine Sub-Mode that the processor core is currently in is reflected in the TYP domain of the CSR register msubm, so

Software can read this CSR register to see which Machine Sub-Mode is currently in. Details about the msubm register For details, 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. View of Mode

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 the In any software-visible register (the processor core maintains a hardware register that is not visible to the software), so the software The software program cannot read the current Machine Mode or User by reading any register. Mode.
- The four machine sub-modes of the N200 series kernel are reflected in the CSR register msubm. TYP domain, so software can read the Machine Sub-Mode currently in by reading this CSR register.

2.2.5. Switching from Machine Mode to User Mode

The mret instruction can be executed directly in Machine Mode. Switching from Machine Mode to User Mode only works Occurs by executing the mret instruction. Since Machine Mode can be in four different states as described in Section 2.2.3 , They are introduced as follows:

- If it is in normal machine mode, execute the hardware behavior of mret instruction and execute mret instruction in exception processing mode The same behavior, see 3 .5 section for the details.

- Therefore, if you want to switch from Machine Mode to User Mode in normal machine mode, you need to
  The software first needs to modify the value of the MPP field of mstatus, and then execute the mret instruction to achieve the effect of mode s

  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 is the MPP bit field corresponding to mstatus. Please refer to
                                    // see Section 7.4, Section 7 for details mstatus the bit field.
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                          // Execute the mret instruction, it will switch the mode to User Mode, and start execution from the previous label 1
                                    // program (label 1 is the position of the next instruction of mret)
```


Page 17

```
1:                              // position of label 1
```

- If the hardware behavior of executing the mret instruction is in the exception handling mode, please refer to Section 3.5 for details.
  - In general, the mret instruction is used to exit from the 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), then you need
    The software first modifies the value of the MPP field of mstatus, and then executes the mret instruction to achieve the effect of mode switchi
- If the interrupt handling mode, the hardware instruction execution mret behavior, see page 5 .7 section for the details.
  - Generally speaking, the mret instruction is used to exit from the 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), then you need
    The software first modifies the value of the MPP field of mstatus, and then executes the mret instruction to achieve the effect of mode switchi
- If you are in the NMI processing mode, instruction execution mret hardware behavior, see Section 4. Section 4 for the details.
  - Generally speaking, the mret instruction is used to exit from NMI processing mode to the mode before entering NMI.
  - If you explicitly want to exit from Machine Mode to User Mode (or normal machine mode), then you need
    The software first modifies the value of the MPP field of mstatus, and then executes the mret instruction to achieve the effect of mode switchi

note:

- If the mret instruction is directly executed in User Mode, an illegal instruction (Illegal Instruction) exception will be generated.
- In User Mode, the processor can only access the physical address area set by the PMP, so switch to User
  Before Mode, you need to configure PMP related registers to set the physical address area accessible by User Mode.
  See Chapter 8 for details.

2.2.6. Switching from User Mode to Machine Mode

The N200 series kernel can be switched from User Mode to Machine Mode only through exceptions, response interrupts, or NMI.
The way it happens:

- Enter exception processing mode in response to an exception. See Section 3.4 for details.

Page 18

- Note: The software can forcibly enter the ecall exception handling function by calling the ecall instruction.

- Enter interrupt processing mode in response to an interrupt. See Section 5.6 for details.
- Respond to NMI and enter NMI processing mode. See Section 4. Section 3 understand its details.

2.2.7. Interrupts, exceptions, and NMI nesting

Interrupts and exceptions can nest themselves, NMI cannot nest itself:

- In NMI processing mode, if NMI occurs again, the new NMI will be blocked, so NMI cannot Self-nesting, see Section 4.6 for details.
- In the exception handling model, if an exception occurs again, which is a nested abnormal situation, see on 3. . 7 section for details situation.
- In interrupt processing mode, if an interrupt occurs again, this is an interrupt nesting situation, please refer to Section 5. 11 for details situation.

Interrupts, exceptions, and NMIs may also be nested within each other, as follows:

- If an exception occurs in the interrupt processing mode, the system enters the exception processing mode.
- If an exception occurs in the NMI processing mode, the system enters the exception processing mode.
- If NMI occurs in interrupt processing mode, it will enter NMI processing mode.
- If NMI occurs in the exception processing mode, the NMI processing mode is entered.
- Note: In NMI and exception mode, the global interrupt bit is automatically turned off by hardware, so it will not respond anymore. Off.

In order to ensure that the exception and NMI can be restored to the previous state (Recoverable) after nesting between each other, The N200 series kernel implements a "Two Levels of NMI / Exception State Save Stacks) "technology, see Section 4. Section 6 for more details.

2.3. Physical Memory Protection (PMP)

Since the N200 series processor core is a low-power core for the microcontroller field, it does not support the virtual address management unit

(Memory Management Unit), so all address access operations use physical addresses. In order to Memory physical address range and different Privilege Modes for privilege isolation and protection. The N200 series processor core supports Physical storage protection (PMP) unit. For more information on PMP units, see Chapter 8 .

Page 20

3. Introduction of N200 Series Kernel Exception Mechanism

3.1. Exception Overview

Exception mechanism, that is, the processor core suddenly encountered an exception during the sequential execution of the program instruction flow To suspend the execution of the current program and handle the exception instead, the main points are as follows:

- "Exceptions" encountered by the processor are called exceptions. An exception is caused by an event or program inside the processor Caused by an event during execution, such as its own hardware failure, program failure, or execution of a special system service instruction Caused, in short, an internal cause.
- After an exception occurs, the processor enters the exception service handler.

3.2. Exception Masking

The RISC-V architecture stipulates that exceptions cannot be masked, that is, once an exception occurs, the processor must stop The previous operation switches to the exception handling mode.

3.3. Priority of exceptions

The processor core may have multiple exceptions happening at the same time, so exceptions have priority. The priority of the exception is as follows 3- 1, the lower the number the higher the number of abnormal exception priority.

3.4. Entering Exception Handling Mode

When entering an exception, the hardware behavior of the N200 series kernel can be briefly described as follows. Note that the following hardware beh Simultaneously completed during the period:

- Stop execution of the current program flow and start execution from the PC address defined by the CSR register mtvec.
- Update related CSR registers, which are the following registers:
  - mcause (Machine Cause Register)
  - mepc (Machine Exception Program Counter)

Page 21

  - mtval (Machine Trap Value Register)
  - mstatus (Machine Status Register)
- Update the Privilege Mode and Machine Sub-Mode of the processor core.

The overall process of abnormal response is shown in Figure 3-1 .

Figure 3-1 Overall process of abnormal response

They are detailed below.

3.4.1. Execution starts from the PC address defined by mtvec

The PC address jumped into by the N200 series kernel when it encounters an exception is specified by the CSR register mtvec.

The mtvec register is a readable and writable CSR register, so software can programmatically change its value. mtvec hosting The detailed format of the device is shown in Table 7-3 .

3.4.2. Update the CSR Register mcause

When the N200 series kernel enters an exception, the CSR register mcause is updated at the same time (automatically by hardware) to reflect the curren The type of exception, the software can read the specific cause of the exception by reading this register.

detailed format of the register mcause Table . 7 -6, in which the lower 5 bits of an abnormal number field for indicating various The exception types are shown in Table 3-1 .

Table 3-1 Exception Code in mcause Register

Exception number (**Exception Code**)	Exception and interrupt types	Synchronous / asynchronous	description
0	Instruction address misalignment (Instruction address misaligned)	Synchronize	Instruction PC address is misaligned. Note: This exception type is configured with "C" extension. This cannot happen in processors that develop a subset of instructions.
1	Instruction access error access fault)	Synchronize	Instruction fetch error.
2	Illegal instruction instruction)	Synchronize	Illegal instruction.
3	Breakpoint	Synchronize	The RISC-V architecture defines the EBREAK instruction, When the processor executes the instruction, an exception occurs Often enter the exception service routine. The instruction is often used Used by the debugger (Debugger), such as Breakpoint
4	Read memory address misalignment (Load address misaligned)	Synchronize	The load instruction fetches addresses that are not aligned. Note: N200 series kernel does not support non-address Aligned data memory read and write operations, so when This exception is generated when the access addresses are not aligned.
5	Read memory access error (Load access fault)	Inaccurate asynchronous load instruction fetch error.	
6	Write memory and AMO address are not correct Qi (Store / AMO address misaligned)	Synchronize	Store or AMO instruction fetch address is not correct Qi. Note: The N200 series kernel does not support ground Address memory read and write operations, because This difference occurs when the access addresses are not aligned often.
7	Write memory and AMO access error (Store / AMO access fault)	Inaccurate asynchronous store or AMO instruction fetch error.	
8	User mode environment call (Environment call from U-mode)	Synchronize	Execute the ecall instruction in User Mode. The RISC-V architecture defines ecall instructions, everywhere When the processor executes this instruction, an abnormal progress will occur. Enter the exception service program. This instruction is often provided by software Use to forcibly enter abnormal mode.
11	Machine mode environment call (Environment call from M-mode)	Synchronize	Execute the ecall instruction in Machine Mode. The RISC-V architecture defines ecall instructions, everywhere When the processor executes this instruction, an abnormal progress will occur. Enter the exception service program. This instruction is often provided by software Use to forcibly enter abnormal mode.

3.4.3. Update the CSR register mepc

The return address when the N200 series kernel exits from the exception is the CSR register mepc (Machine Exception Program Counter) Save. Upon entering the exception, the hardware will automatically update the value of the mepc register, which will be used as the exit from the ex The return address, after the abnormal end, can use the PC value it saved to return to the program point where the execution was stopped abnormally.

note:

- When an exception occurs, the value of the exception return address mepc is updated to the instruction PC where the exception currently occurs.

- Although the mepc register is automatically updated by hardware when an exception occurs, the mepc register itself is also a readable A writable register, so software can also write this register directly to modify its value.

3.4.4. Update the CSR register mtval

When the N200 series 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 code that caused the current exception:

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

3.4.5. Update the CSR register mstatus

detailed format of the register mstatus Table 7 shown -2, N200 series kernel exception entry is automatically updated hardware 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 shown in Section and Section 9 The role of the mstatus.MPIE field is to be able to use the value of mstatus.MPIE to recover from an abnormal error after the abnormal end. Mstatus.MIE value before birth.
- 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 responded).

- The value of the mstatus.MPP field is updated to the Privilege Mode before the exception occurred, as shown in Section and Section 9.2 . field is acting mstatus.MPP after abnormal end, it can be used MSTATUS .MPP values recovered abnormality Previous Privilege Mode.

3.4.6. Update Privilege Mode

Exceptions need to be handled in Machine Mode. When entering exceptions, the processor core's Privilege Mode is updated to machine mode.

3.4.7. Update Machine Sub-Mode

The Machine Sub-Mode of the N200 series kernel is reflected in the msubm.TYP domain of the CSR register in real time. Entering different Often, the Machine Sub-Mode of the processor core is updated to the exception handling mode, so:

- The value of the msubm.PTYP field in the CSR register is updated to Machine Sub-Mode before the exception occurred (The value of the msubm.TYP field), as shown in Section and Section 9.2 . The role of the msubm.PTYP domain is to terminate abnormally. After that, the value of msubm.PTYP can be used to recover the Machine Sub-Mode value before the exception occurred.
- The value of the msubm.TYP field in the CSR register is updated to "Exception Handling Mode" as shown in Section and Section 9.2 . It reflects that the current mode is already "Exception Handling Mode".

Figure 3-2 CSR register changes upon entry / exit exception

3.5. Exiting Exception Handling Mode

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

Because the exception handling is in Machine Mode, the software must use the mret instruction when exiting the exception. processor
The hardware behavior after executing the mret instruction is as follows. Note that the following hardware behaviors are completed simultaneously in one clo

- Stop execution of the current program flow and start execution from the PC address defined by the CSR register mepc.
- Update the CSR register mstatus (Machine Status Register) as shown in Section and Section 9.2, and update the processor
  Kernel Privilege Mode and Machine Sub-Mode.

The overall process of exiting the exception is shown in Figure 3-3 .

Figure 3-3 Overall process of exit exception

They are detailed below.

3.5.1. Start execution from the PC address defined by mepc

When entering the exception, the mepc register is updated at the same time to reflect the instruction PC value that encountered the exception at that time
Means that after the mret instruction is executed, the processor returns to the PC address of the instruction that encountered the exception at that time, so that
The previously aborted program flow.

Page 26

Note: It may be necessary to update the value of mepc with software before exiting the exception. For example, if an exception is caused by ecall or ebr
Generated because the value of mepc is updated to the ePC or ebreak instruction of the own PC. Therefore, when the exception returns, if you use
Using the PC value saved by mepc as the return address, it will jump back to the ecall or ebreak instruction again, resulting in an endless loop
Executing an ecall or ebreak instruction causes a re-entry exception). The correct way is that the software changes the mepc pointer in the exception handler
In the next instruction, since ecall / ebreak are both 4-byte instructions, rewrite setting mepc = mepc + 4.

3.5.2. Update the CSR register mstatus

detailed format of the register mstatus Table 7- shown in FIG. After executing the mret instruction, the hardware will automatically update the CSR reg
Domains of the router mstatus:

- The value of the mstatus.MIE field is restored to the current value of mstatus.MPIE.
- The current mstatus.MPIE field is updated to 1.
- The updated value of the mstatus.MPP domain is 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 the exception, the value of mstatus.MPIE was updated to the value of mstatus.MIE before the exception, such as the
As shown in 9.2 . After the mret instruction is executed, the value of the mstatus.MIE field is restored to the value of mstatus.MPIE. Through this machine
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 that
(The previous mstatus.MIE value was 1, which means that interrupts were re-enabled globally).

3.5.3. Update Privilege Mode

When entering the exception, the value of mstatus.MPP was updated to the Privilege Mode before the exception occurred.
After the mret instruction, the processor's Privilege Mode is restored to the value of mstatus.MPP , as shown in Section and Section 9.2 . by
This mechanism ensures that the processor returns to the privilege mode of the processor before the exception occurred.

3.5.4. Update Machine Sub-Mode

Page 27

The Machine Sub-Mode of the N200 series kernel is reflected in the msubm.TYP domain of the CSR register in real time. In execution
After the mret instruction, the hardware will automatically restore the Machine Sub-Mode of the processor to the value of the msubm.PTYP domain:

- When entering the exception, the value of the msubm.PTYP field was updated to the Machine Sub-Mode before the exception occurred
  value. After using the mret instruction to exit the exception, the hardware restores the value of the processor Machine Sub-Mode to
  The value of the msubm.PTYP domain is shown in Figure 4-2. By this mechanism, it means that the processor's
  Machine Sub-Mode is restored to Machine Sub-Mode before the exception occurred.

3.6. Exception Service Program

When the processor enters an exception, it starts to execute a new program from the PC address defined by the mtvec register, which is usually
Exception service program, and the program can also determine further by querying the exception code in mcause
Jump to a more specific exception service routine. For example, when the program queries the value of mcause as 0x2, it is learned that the exception is illeg
Caused by the error (Illegal Instruction), so you can further jump to the illegal instruction error exception service subroutine
go with.

Note: Since there is no hardware operation to automatically save and restore context in the entry exception and exit exception mechanisms, soft
The file explicitly uses instructions (written in assembly language) for context saving and restoration. See the Nuclei_N200 series
The Core SDK Instructions explains it with a complete example of exception service code.

3.7. Exception nesting

The N200 series kernel supports two levels of NMI / Exception State Save (Two Levels of NMI / Exception State Save
Stacks), more details See also 4 .6 festival.

4. N200 series kernel NMI mechanism introduction

4.1. NMI Overview

NMI (Non-Maskable Interrupt) is a special input signal of the processor core, which is often used to indicate the system
Level of emergency errors (such as external hardware failures, etc.). After encountering the NMI, the processor core should abort execution immediately
The previous program turned to handle the NMI error.

4.2. NMI Shield

The NMI in the N200 series kernel cannot be masked, which means that once the NMI occurs, the processor will definitely stop
The current operation instead processes NMI.

4.3. Entering NMI Processing Mode

When entering the NMI processing mode, the hardware behavior of the N200 series kernel can be briefly described as follows. Note that the following h
Simultaneous completion in two clock cycles:

- Stop execution of the current program flow and start execution from the PC address defined by the CSR register mnvec.
- Update related CSR registers, which are the following registers:
  - mepc (Machine Exception Program Counter)
  - mstatus (Machine Status Register)
  - mcause (Machine Cause Register)
- Update the Privilege Mode and Machine Sub-Mode of the processor core.

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

Page 29

Figure 4-1 NMI response overall process

They are detailed below.

4.3.1. Execution starts from the PC address defined by mnvec

The PC address that the N200 series kernel jumps into after encountering the NMI is specified by the CSR register mnvec. mnvec register value There are two situations:

- When mmisc_ctl [9] = 1, the value of the mnvec register is equal to mtvec, that is, the NMI has the same trap as the exception Entry address.
- When mmisc_ctl [9] = 0, the value of the mnvec register is equal to reset_vector, and reset_vector is PC value after reset.

4.3.2. Update the CSR register mepc

The return address when the N200 series kernel exits the NMI is the CSR register mepc (Machine Exception Program Counter) Save. When entering the NMI, the hardware will automatically update the value of the mepc register, which will be used as the exit NMI After the NMI is over, the PC value saved by it can be used to return to the previously stopped program point.

Page 30

note:

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

4.3.3. Update the CSR Register mcause

detailed format of the register mcause Table 7- 6. When the N200 series 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.
NMI's 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, the cause of the Trap can be identified, and the software can Design specific handlers to handle traps.

4.3.4. Update the CSR register mstatus

detailed format of the register mstatus Table 7- 2, N200 series kernel entering NMI, hardware automatically updates 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 NMI occurs, as shown in Figure 4-2 . The role of the mstatus.MPIE field is to restore the NMI using the value of mstatus.MPIE after the NMI is over. The mstatus.MIE value before it happened.
- The value of the mstatus.MIE field is updated to 0 (meaning that the interrupt is globally closed after entering the NMI service routine, so all Some interrupts will be masked and not responded).
- The value of the mstatus.MPP field is updated to Privilege Mode before the NMI occurred. Save the action of the mstatus.MPP domain After the NMI ends, you can use the value of mstatus.MPP to restore the Privilege before the NMI occurred. Mode.

4.3.5. Update Privilege Mode

NMI processing is done in Machine Mode, so when entering NMI, the special characteristics of the processor core Privilege Mode switches to machine mode.

Page 31

4.3.6. Update Machine Sub-Mode

The Machine Sub-Mode of the N200 series kernel is reflected in the msubm.TYP domain of the CSR register in real time. Entering During NMI, the Machine Sub-Mode of the processor core is updated to the NMI processing mode, so:

- The value of the msubm.PTYP field in the CSR register is updated to Machine Sub-Mode before NMI (The value of the msubm.TYP field), as shown in Figure 4-2 . The role of the msubm.PTYP domain is after NMI ends,

The value of msubm.PTYP can be used to restore the Machine Sub-Mode value before the NMI occurred.

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

The current mode is already "NMI processing mode".

Figure 4-2 CSR register changes when entering / exiting NMI

4.4. Exiting NMI Processing Mode

After the program finishes NMI processing, it needs to exit from the NMI service program and return to the main program.

Because NMI processing is in Machine Mode, software must use the mret instruction when exiting NMI. Place

After the processor executes the mret instruction, the hardware behavior is as follows. Note that the following hardware behaviors are completed simultaneou

- Stop execution of the current program flow and start execution from 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 shown in Figure 4-3 .

Page 32

Figure 4-3 Overall process of exiting NMI

They are detailed below.

4.4.1. Start execution from the PC address defined by mepc

When entering the NMI, the mepc register is updated at the same time to reflect the PC value of the next instruction that encountered the NMI. through

After this mechanism, it means that after the mret instruction is executed, the processor returns to the PC address of the next instruction that encountered NM

Instead, execution continues with the previously aborted program flow.

4.4.2. Update the CSR register mstatus

detailed format of the register mstatus Table 7- 2, after executing mret instruction, the hardware will automatically update register CSR Domain mstatus:

- The value of the mstatus.MIE field is restored to the current value of mstatus.MPIE.
- The value of the mstatus.MPIE field is updated to 1.
- The updated value of the mstatus.MPP domain is 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 NMI, the value of mstatus.MPIE was updated to the value of mstatus.MIE before NMI occurred. While mret After the instruction is executed, restore the value of mstatus.MIE to the value of mstatus.MPIE , as shown in Figure 4-2. 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 NMI occurred (assuming the pre The mstatus.MIE value is 1, which means that interrupts are re-enabled 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. After the mret instruction, the processor's Privilege Mode is restored to the value of mstatus.MPP , as shown in Figure 4-2 . through this Mechanism to ensure 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 N200 series kernel is reflected in the msubm.TYP domain of the CSR register in real time. In execution After the mret instruction, the hardware will automatically restore the Machine Sub-Mode of the processor to the value of the msubm.PTYP domain:

- When entering NMI, the value of msubm.PTYP field was updated to Machine Sub-Mode before NMI value. After using the mret instruction to exit NMI, the hardware restores the value of the processor Machine Sub-Mode to The value of the msubm.PTYP domain is shown in Figure 4-2 . By this mechanism, it means that after exiting the NMI, the processor Machine Sub-Mode is restored to Machine Sub-Mode before the NMI occurred.

4.5. NMI Service Program

When the processor enters the NMI, it starts to execute a new program from the PC address defined by the mnvec register. This program usually Service program for NMI.

Note: Since there is no hardware operation to automatically save and restore context in the NMI entry and exit NMI mechanisms, it is required The software explicitly uses instructions (written in assembly language) to save and restore context. See the Nuclei_N200 Series The Kernel SDK Instructions explains it with a complete NMI service program code example.

Page 34

4.6. NMI / Exception Nesting

N200 series customized kernel FIG 4-4 two shown NMI / abnormal state stack (Two Levels of NMI / Exception State Save Stacks), to save up to three levels of NMI / Exception processor state, can achieve level two recovery NMI / Exception Nesting.

Note: Since the NMI response is blocked on the hardware when the processor is in the NMI state, the NMI cannot be implemented. Now self-nesting. N200 series NMI / exception nesting only supports the following 3 kinds of nesting:

- NMI nested exception
- Exception Nesting Exception
- Exceptionally nested NMI

Figure 4-4 Schematic diagram of the two-level NMI / abnormal state stack mechanism of the N200 series kernel

4.6.1. Entering NMI / Exception Nesting

When responding to NMI and exceptions, the hardware behavior of the N200 series core is shown in Figure 4-4 , which can be briefly described as follo

- Stop execution of the current program flow and start execution from a new PC address.
  - If the response is abnormal, execution starts from the PC address stored in mtvec.

Page 35

  - If the response is NMI, the execution starts from the PC address stored in mnvec.
- Update the relevant CSR registers, which are the following registers and their related fields:
  - mepc: Record the current NMI / PC before the exception occurred. NMI / Exception can be restored from mepc when exiting the NMI / abnorma Often occurs before the PC.
  - msaveepc1: first level NMI / exception status stack record first level nested NMI / exception

Often nested NMI / exception) before the PC, that is, the current mepc value before the NMI / exception, exit
When NMI / abnormal, mepc value can be restored from msaveepc1.

- msaveepc2: second level NMI / exception status stack record second level nested NMI / exception (nested by first level NMI / Exception Nested NMI / Exception) PC before the occurrence, that is, current NMI / Exception
  The value of msaveepc1, the value of msaveepc1 can be restored from msaveepc2 when exiting NMI / exception.
- mstatus:
  - MPIE: Record the MIE before the current NMI / exception.
  - MPP: Record the Privilege Mode before the current NMI / exception.
- msavestatus:
  - MPIE1: The first level NMI / exception status stack records the first level nested NMI / MIE before the exception, That is, the current MPIE before the NMI / abnormality, MPIE1 can be restored from MPIE1 when exiting the NMI / abnormal Value.
  - MPIE2: Second-level NMI / Exception status stack record Second-level nested NMI / MIE before exception, That is, MPIE1 before the current NMI / exception, and can be recovered from MPIE2 when exiting the NMI / exception The value of MPIE1.
  - MPP1: First level NMI / Exception status stack record First level nested NMI / Privilege before exception Mode, that is, MPP before the current NMI / abnormal, can be restored from MPP1 when exiting NMI / abnormal The value of MPP.
  - MPP2: Second-level NMI / Exception status stack record Second-level nested NMI / Exception Privilege Mode, that is, MPP1 before the current NMI / exception, you can switch from NMI / exception when exiting MPP2 restores the value of MPP1.

Page 36

- mcause: Record the cause of the current NMI / abnormality.
- 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: Record the current NMI / abnormal Trap type.
  - PTYP: Record the type of trap that the processor was in before the current NMI / exception
  - PTYP1: First level NMI / Exception status stack record First level Nested NMI / Machine before exception Sub Mode, that is, PTYP before the current NMI / exception, can exit from PTYP1 when exiting NMI / exception Restore the value of PTYP.
  - PTYP2: Second-level NMI / Exception status stack record Second-level nested NMI / Exception Machine Sub Mode, which is PTYP1 before the current NMI / exception, can be exited from NMI / exception Restore the value of PTYP1 from PTYP2.

- NMI / Exception processing is done in Machine Mode, so when entering NMI / Exception,
  The processor's privileged mode (Privilege Mode) switches to machine mode.

4.6.2. Exiting NMI / Exception Nesting

After the program finishes NMI / exception processing, it needs to exit from the NMI / exception service program and return to the superior NMI / exce Or the main program, the processor state needs to be restored from the relevant register before exiting. This is done through the mret instruction. The process OK mret hardware behavior after the instruction shown in FIG . 4 -4, can be summarized as follows.

- Stop execution of the current program flow and start execution from the PC address defined by the CSR register mepc.
- Update the relevant CSR registers, which are the following registers and their related fields:
  - mepc (Machine Exception Program Counter): restore to the first level stored in msaveepc1
    Nested PC before NMI / Exception.
  - msaveepc1: the first-level NMI / abnormal state stack, mret from the second-level NMI / abnormal state stack

msaveepc2 restores the value of the msaveepc1 register, that is, restores the second-level nesting stored in msaveepc2
PC before NMI / Exception.

- mstatus (Machine Status Register)
  - MPIE: Revert to the first level of nested NMI / MIE stored before MPIE1.
  - MPP: Restore to the first level of nested NMI stored in MPP1 / Privilege before the exception
    Mode.
- msavestatus:
  - MPIE1: First level NMI / Exception status stack, mret occurs from second level NMI / Exception status stack
    MPIE2 restores the value of the register field msavestatus.MPIE1, that is, restores the value stored in MPIE2.
    Second level nested NMI / MIE before exception.
  - MPP1: First level NMI / Exception status stack, mret occurs from second level NMI / Exception status stack
    MPP2 restores the value of the register field msavestatus.MPP1, that is, restores the second value stored in MPP2
    Level Nested NMI / Privilege Mode before the exception occurred.
- mcause (Machine Cause Register): revert to the first level of nesting stored in msavecause1
  NMI / Exception Cause.
- msavecause1: the first-level NMI / abnormal state stack, mret from the second-level NMI / abnormal state stack
  The stack msavecause2 restores the value of the register msavecause1, that is, the value stored in msavecause2
  Cause of second level nested NMI / exception.
- msubm (Machine Sub-Mode Register)
  - TYP: Revert to the Trap class of the current NMI / exception handler stored in msubm.PTYP
    type.
  - PTYP: Restored to the first-level nested NMI / pre-exception processor stored in msubm.PTYP1
    Trap type.
  - PTYP1: First level NMI / Exception status stack, mret occurs from second level NMI / Exception status stack
    PTYP2 restores the value of the register field msubm.PTYP1, that is, the value stored in msubm.PTYP2

Trap type of the second level nested NMI / exception handler.

Page 38

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

Page 39

5. N200 Series Kernel Interrupt Mechanism

5.1. Interrupt Overview

Interrupt mechanism, that is, the processor core is suddenly interrupted by other requests during the sequential execution of the program instruction flov

And suspend execution of the current program, instead of processing other things, wait for it to finish other things, and then return to the previous program The point of interruption continues execution of the previous program instruction stream.

Some basic points of interruption are as follows:

- The "other request" that interrupts the execution of the processor is called an "Interrupt Request", and the "other request" The source is called Interrupt Source. The interrupt source usually comes from outside the kernel (called external Interrupt source), or from the kernel (becomes an internal interrupt source).
- The "other thing" that the processor turns to handle is called the Interrupt Service Routine, ISR).
- Interrupt handling is a normal mechanism, not an error situation. After the processor receives the interrupt request, it needs to save The site of the current program is referred to as "save site" for short. After the interrupt service routine is processed, the processor needs to recov The previous site, thus continuing to execute the previously interrupted procedure, referred to as "recovery site".
- There may be situations where multiple interrupt sources initiate requests to the processor at the same time, and these interrupt sources need to be arl Select which interrupt source is handled first. This situation is called "interrupt arbitration", and different interrupts can be allocated at the same t Levels and priorities facilitate arbitration, so the concept of "interrupt level" and "interrupt priority" exists for interrupts.

5.2. Interrupt Controller ECLIC

The first 7.4 the .14 section by different software configuration, N200 series kernel supports "default interrupt mode" and "ECLIC interrupt mode", "ECLIC interrupt mode" is recommended. This article only introduces "ECLIC interrupt mode".

The N200 series core implements an "Enhanced Core Local Interrupt Controller (Enhanced Core Local Interrupt Controller, ECLIC) "can be used for the management of multiple interrupt sources. All types in the N200 series kernel (except debug interrupts Outside) the interruption by ECLIC unified management, details about ECLIC, see the 6. 2 Day. About N200 Series Kernel supports all types of interrupts, refer to section 5. Section 3 .

Page 40

5.3. Interrupt types

The interrupt types supported by the N200 series core are shown in Figure 5-1 .

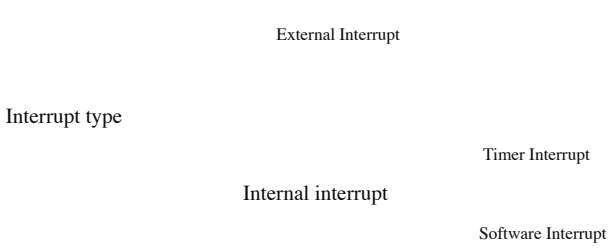


Figure 5-1 Interrupt type diagram

They are detailed below.

5.3.1. External interrupt

External interrupts are interrupts that come from outside the processor core. External interrupt allows users to connect external interrupt sources, such a Prepare for interrupts generated by UART, GPIO, etc.

Note: The N200 series kernel supports multiple external interrupt sources, and all external interrupts are uniformly managed by ECLIC.

5.3.2. Internal interrupt

The N200 series kernel has several kernel-specific internal interrupts, which are:

- Software Interrupt
- Timer Interrupt

Note: The internal interrupts of the N200 series cores are also managed by ECLIC.

Page 41

5.3.2.1 Software interrupt

The main points of software interrupt are as follows:

- The N200 series kernel implements a TIMER unit. A msip register is defined in the TIMER unit.
  Software interrupts can occur after this, see Section 6. 1.6 for details.
- Note: Software interrupts are also managed uniformly by ECLIC.

5.3.2.2 Timer interrupt

The main points of timer interrupt are as follows:

- The N200 series kernel implements a TIMER unit. 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 uniformly by ECLIC.

5.4. Interrupt masking

5.4.1. Interrupt global masking

The interrupt of the N200 series kernel 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. Individual interrupt sources masked

For different interrupt sources, ECLIC allocates its own interrupt enable register for each interrupt source.
Configure the ECLIC register to manage the masking of individual interrupt sources, see Section 6.2. 6 for details.

5.5. Interrupt Level, Priority, and Arbitration

When multiple interrupts occur simultaneously, arbitration is required. For N200 series core processors, ECLIC unified management All interrupts. ECLIC assigns its own interrupt level and priority register to each interrupt source.

ECLIC register to manage the level and priority of each interrupt source. When multiple interrupts occur at the same time, ECLIC will arbitrate the level

Page 42

And the highest priority interrupt, as shown in Figure 5-2 . See or details.

Figure 5-2 Interrupt arbitration

5.6. Enter Interrupt Processing Mode

When responding to an interrupt, the hardware behavior of the N200 series core can be briefly described as follows. Note that the following hardware b Simultaneously completed during the period:

- Stop execution of the current program flow and start execution from a new PC address.
- Entering an interrupt not only causes the processor to jump to the above PC address and start execution, but also causes the hardware to update other CSR registers , as shown in Figure 5-4, are the following registers:
  - mepc (Machine Exception Program Counter)
  - mstatus (Machine Status Register)
  - mcause (Machine Cause Register)
  - mintstatus (Machine Interrupt Status Register)
- In addition, entering the interrupt will also update the Privilege Mode and Machine Sub-Mode of the processor core.
- The overall process is shown in Figure 5-3 .

Page 43

Figure 5-3 Overall process of responding to an interrupt

They are detailed below.

5.6.1. Execution from the new PC address

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

- If configured as vector processing mode (clicintattr [i] .shv = 1), the interrupt will be processed by the processor
  The processor directly jumps into the target address stored in the interrupted Vector Table Entry n
  For a detailed description of the scale, see section 5.8 . For a detailed description of the vector p e section 5.13.2 Section.
- If configured for non-vector processing mode (clicintattr [i] .shv = 0), the interrupt is responded by the processor core
  After that, the processor jumps directly to the entry address shared by all interrupts. Detailed introduction to interrupt non-vector processing mod
  See Section 5.13.1 .

5.6.2. Update Privilege Mode

Upon entering the interrupt, the Privilege Mode of the processor core is updated to Machine Mode.

5.6.3. Update Machine Sub-Mode

The Machine Sub-Mode of the N200 series kernel is reflected in the msubm.TYP domain of the CSR register in real time. In the process of entering When interrupted, the Machine Sub-Mode of the processor core is updated to the interrupt processing mode, so:

- The value of the msubm.PTYP field in the CSR register is updated to the Machine Sub-Mode (msubm.TYP field) before the interrupt occurs. Value), as shown in Figure 5-4 . The purpose of the msubm.PTYP domain is to be able to use msubm.PTYP after the interruption ends. Is restored to the Machine Sub-Mode value before the interrupt occurred.
- The value of the msubm.TYP field in the CSR register is updated to the "Interrupt Processing Mode", as shown in Figure 5-4. The current mode is already "Interrupt Processing Mode".

5.6.4. Update CSR register mepc

The return address when the N200 series kernel exits the interrupt is specified by the CSR register mepc. When entering the interrupt, the hardware will Update the value of the mepc register automatically. This register will be used as the return address to exit the interrupt. After the interrupt is completed, it ca The saved PC value returns to the program point where execution was stopped previously.

note:

- When an interrupt occurs, the interrupt return address mepc is pointed to an instruction. This instruction could not be completed because of the interr carried out. Then after exiting the interrupt, the program will return to the previous program point, from the unexecuted stored in mepc. The instruction starts to execute again.
- Although the mepc register is automatically updated by hardware when an interrupt occurs, the mepc register itself is also a readable A writable register, so software can also write this register directly to modify its value.

5.6.5. Update the CSR registers mcause and mstatus

detailed format of the register mcause Table 7- 6. When N200 series core enters interrupt, CSR register mcause is updated at the same time (automatically by hardware), as shown in Figure 5-4 . The details are as follows:

- After the current interrupt is serviced, a mechanism is required to record the ID number of the current interrupt source.
  - When the N200 series kernel enters the interrupt, the mcause.EXCCODE field of the CSR register is updated to reflect the current The ID number of the responding ECLIC interrupt source, so the software can query the specific source of the interrupt by reading this registe ID.
- The current interrupt is responded, which may interrupt the interrupt that was being processed previously (interrupt level is relatively low, so Interrupted), a mechanism is needed to record the interrupt levels (Interrupt Levels) interrupted by the interrupt.
  - When the N200 series kernel enters the interrupt, the CSR register mcause.MPIL field is updated to reflect the interrupted Interrupt level (value of the mintstatus.MIL field). The purpose of the mcause.MPIL field is to Enough to use the value of mcause.MPIL to restore the value of mintstatus.MIL before the interrupt occurred.
- After the current interrupt is serviced, a mechanism is required to record the global enable status and characteristics Right model.
  - When the N200 series kernel enters an interrupt, the value of the CSR register mstatus.MPIE field is updated to indicate that the interrupt occurr The global enable status of the previous interrupt (value of the mstatus.MIE field). The value of the mstatus.MIE field is updated to 0 (meaning that the interrupt is globally closed after entering the interrupt service routine, all interrupts will be masked and not responded).
  - When the N200 series kernel enters an interrupt, the processor's current Privilege Mode is switched to the machine Mode, and the value of the CSR register mstatus.MPP field is updated before the interrupt occurred Privilege Mode.
- If the currently responding interrupt is in vector processing mode, the processor will directly jump into the interrupt vector input after responding to The destination address of the vector table entry. For a detailed description of interrupt vector processing modes, see See Section 5.13.2 . In terms of hardware implementation, the processor needs to go in two steps. The first step is from the interrupt vector table. Take out the stored target address, and then jump to the target address in the second step. Then, in the first step, Take out the storage target address from the scale "Memory access errors may occur during this memory access operation Errors, a mechanism is needed to record this particular memory access error.
  - When the N200 series 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 above two-step operation is completed successfully The value of the mcause.minhv field is cleared to 0. Assuming a memory access error occurs midway, the final processor will An instruction access fault occurred and the value of the mcause.minhv field is 1 (no

Cleared).

- Note: The values of the mstatus.MPIE and mstatus.MPP fields are the same as the values of the mcause.MPIE and mcause.MPP fields
  Is a mirror relationship, that is, under normal circumstances, the value of the mstatus.MPIE domain and the value of the mcause.MPIE domain ar

Page 46

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

Figure 5-4 CSR register changes when entering / exiting interrupt

5.7. Exiting Interrupt Processing Mode

After the program finishes interrupt processing, it needs to exit from the interrupt service routine and return to the main program. Due to interrupt proce
It is in Machine Mode, so when exiting the interrupt, the software must use the mret instruction. After the processor executes the mret instruction
The hardware behavior is as follows. Note that the following hardware behaviors are completed simultaneously in one clock cycle:

- Stop execution of the current program flow and start execution from the PC address defined by the CSR register mepc.
- Executing the mret instruction not only causes the processor to jump to the above PC address to start execution, but also causes the hardware to upda
  The other CSR registers, as shown in Figure 5-4, are the following registers:
  - mstatus (Machine Status Register)
  - mcause (Machine Cause Register)
  - mintstatus (Machine Interrupt Status Register)
- In addition, entering the interrupt will also update the Privilege Mode and Machine Sub-Mode of the processor core.

Page 47

When exiting the interrupt, the software must use the mret instruction

Start execution from the PC address defined by the CSR register mepc

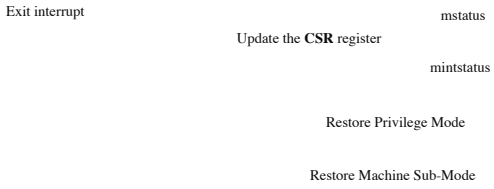


Figure 5-5 Overall process of exit and interrupt

They are detailed below.

5.7.1. Starting 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. Software mus
The mret instruction exits the interrupt. After executing the mret instruction, the processor will restart execution from the pc address defined by mepc. throug
Mechanism, meaning that after the mret instruction is executed, the processor returns to the PC address at the time when the interrupt was encountered, so tha
The aborted program flow.

5.7.2. Update CSR registers mcause and mstatus

The detailed format of the mcause register is shown in Table 7-6 . After the mret instruction is executed, the hardware will automatically update the CSI
Some domains of mcause:

- When entering an interrupt, the value of mcause.MPIL was updated to the value of mintstatus.MIL before the interrupt occurred. So that
  After exiting the interrupt with the mret instruction, the hardware restores the value of mintstatus.MIL to the value of mcause.MPIL. by
  This mechanism means that after exiting the interrupt, the processor's mintstatus.MIL value is restored to the value before the interrupt occurred.
  value.
- When entering an interrupt, the value of mcause.MPIE was updated to the value of mstatus.MIE before the interrupt occurred. While using
  After the mret instruction exits the interrupt, after the mret instruction is executed by the hardware, the value of mstatus.MIE is restored to

  The value of mcause.MPIE is shown in Figure 5-4. By this mechanism, it means that the processor's
  The mstatus.MIE value is restored to what it was before the interrupt occurred.
- When entering the interrupt, the value of mcause.MPP was updated to the privileged mode before the interrupt occurred (Privilege
  Mode). After using the mret instruction to exit the interrupt, the hardware restores the processor's privileged mode (Privilege Mode)
  Is mcause.MPP , as shown in Figure 5-4. By this mechanism, it means that the processor's
  Privilege Mode is restored to the mode it was in before the interrupt occurred.
- Note: The values of the mstatus.MPIE and mstatus.MPP fields are the same as the mcause.MPIE and mcause.MPP fields
  The value of is the mirror relationship, that is, under normal circumstances, the value of the mstatus.MPIE domain and the value of the mcause.MI
  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 domain:

- When entering the interrupt, the value of mcause.MPP was updated to the privileged mode before the interrupt occurred (Privilege

Mode). After using the mret instruction to exit the interrupt, the hardware restores the processor's privileged mode (Privilege Mode) Is the value of mcause.MPP. By this mechanism, it means that the processor's privileged mode after exiting the interrupt (Privilege Mode) is restored to the mode before the interrupt occurred.

5.7.4. Update Machine Sub-Mode

The Machine Sub-Mode of the N200 series kernel is reflected in the msubm.TYP domain of the CSR register in real time. In execution After the mret instruction, the hardware will automatically restore the Machine Sub-Mode of the processor to the value of the msubm.PTYP domain:

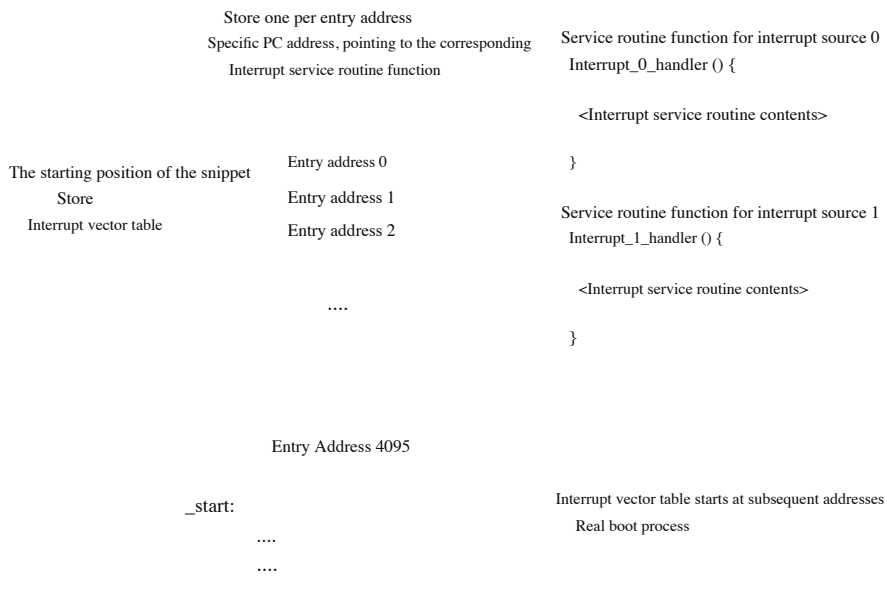
- When entering the interrupt, the value of the msubm.PTYP field was updated to the Machine Sub-Mode before the interrupt occurred value. After using the mret instruction to exit the interrupt, the hardware restores the value of the processor Machine Sub-Mode to The value of the msubm.PTYP domain is shown in Figure 5-4. By this mechanism, it means that the processor's Machine Sub-Mode is restored to Machine Sub-Mode before the interrupt occurred.

5.8. Interrupt vector table

FIG 5- in FIG. 6, the interrupt vector table in the memory means is opened inside a contiguous address space, the address space Each word is used to store the interrupt service routine (Interrupt Service Routine, ISR) PC address of the function.

The starting address of the interrupt vector table is specified by the CSR register mtvt. Generally, the mtvt register can be set to the entire code segment The starting position.

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 In vector processing mode, the hardware will eventually jump to its corresponding interrupt service routine by querying the PC address stored in the interrupt Order functions, see Section 5.13 for more details.



....

Figure 5-6 Interrupt vector table

Page 50

5.9. Context saving and restoring in and out of interrupts

RISC-V-based processors do not have hardware to automatically save and restore context when entering and exiting interrupt processing mode Memory) operations, therefore, the software needs to explicitly use (assembly language) instructions for context saving and restoration. root Depending on whether the interrupt is in vector processing mode or non-vector processing mode, the content involved in saving and restoring the context wil See Section 5.13 for more details.

5.10. Interrupt Response Delay

The concept of interrupt response delay usually refers to the transition from "external interrupt source pulled up" to "the processor actually starts execut The number of instruction cycles consumed by the first instruction in the corresponding Interrupt Service Routine (ISR).
Therefore, the interrupt response delay usually includes the following cycle costs:

- Overhead for processor core to respond to interrupts
- Cycle overhead spent by the processor core to save context
- The overhead of the processor core jumping to the Interrupt Service Routine (ISR).

Depending on whether the interrupt is in vector processing mode or non-vector processing mode, the interrupt response delay varies, see section 5. 13 Section for more details.

5.11. Interrupt nesting

While the processor core is processing an interrupt, a new interrupt request may arrive at a higher level. The processor may
In order to stop the current interrupt service routine and start to respond to the new interrupt and execute its "interrupt service routine", this is formed Interrupt nesting (that is, the previous interrupt has not yet been completed, and it starts to respond to new interrupts), and the nesting level can have many le

Take the example in Figure 5- 7 as an example:

- Suppose the processor is processing a timer interrupt, and suddenly another button 1 interrupt arrives (the level is higher than the timer interrupt High), the processor will suspend processing timer interrupts and begin processing key 1 interrupts.
- But suddenly another key 2 interrupt comes (the level is higher than the key 1 interrupt), then the processor will pause. The interrupt of key 1 is processed, and the interrupt of key 2 is processed.

Page 51

- After no other higher-level interrupt comes, the interrupt of key 2 will not be interrupted any more, and the processor can process smoothly.

After finishing the interruption of button 2, then return to the processing routine of interruption of button 1, and finish where the interruption of but
Management.

- After finishing the interrupt processing of key 1, the processor will return to the timer interrupt processing routine to complete the timing.

  Processor interrupt processing.

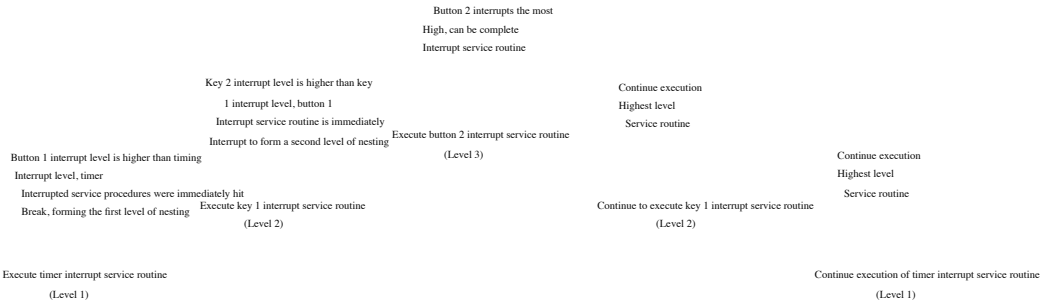


Figure 5-7 Interrupt nesting diagram

Note: Assuming that the new interrupt request has a lower priority (or the same) as the interrupt level being processed, the processor should
In response to this new interrupt request, the processor must complete the current interrupt service routine before considering responding to the new interrupt
The level for a new interrupt request is not higher than the interrupt level currently being processed). In determining the interrupt level, see Section 6.2 .9 Sec
to know more information.

In the N200 series kernel, depending on whether the interrupt is in vector processing mode or non-vector processing mode, the nested support side of th
The guilds differ, see Section 5.13 for more details.

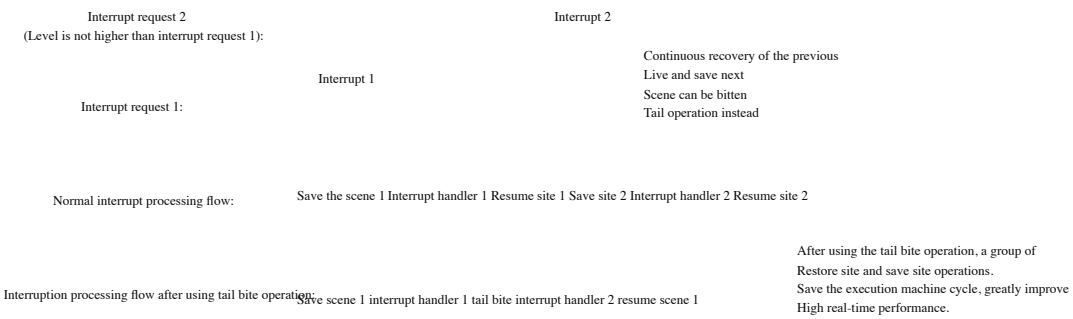
5.12. Interrupting tail biting

While the processor core is processing an interrupt, a new interrupt request may come, but the "new interrupt level" is low
Is at or equal to the "interrupt level currently being processed", so a new interrupt cannot interrupt an interrupt that is currently being processed (hence not
Will form a nest).

When the processor completes the current interrupt, it theoretically needs to restore the context, then exit the interrupt back to the main application, and

Page 52

Re-respond to the new interrupt, and responding to the new interrupt requires saving the context again. Therefore, there is a "back to back"
Below "and" save context "operations, if you omit this back-to-back" restore context "and" save context ",
Called "interrupting tail bite", as shown in Figure 5- 8, it is obvious that interrupting tail bite can speed up the back-to-back processing of multiple interrupts
degree.



Immediately after the execution of the previous interrupt handler function, determine whether there is still waiting (Pending) interruption.
If there is a pending interrupt, immediately respond to the interrupt and execute the interrupt The corresponding interrupt handler.
The tail-biting operation saves the processing time of a restoration and preservation site.

Figure 5-8 Interrupted tail biting

In the N200 series kernel, only the non-vector processing mode supports interrupt tail bite, see Section 5.13.1.3 for more Details.

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

The section 6.2 of the section in .10, ECLIC Each interrupt source can be provided to a vector or non-vector processing (via the configuration register Shlict field of register clicintattr [i], 1 is vector mode, 0 is non-vector mode), vector processing mode and non-vector processing mode 2 There are major differences, as described below.

5.13.1. Non-vector processing mode

Page 53

5.13.1.1 Features and delays of non-vector processing modes

If configured as non-vector processing mode, the processor will jump to all The entry address shared by the non-vector interrupt. This entry address can be set by software:

- If the least significant bit of the CSR register mtvt2 is configured to 0 (the default value after power-on reset), all
  The entry address is specified by the value of the CSR register mtvec (ignoring the value of the least significant 2 bits). Due to the value of the mtv
  The exception's entry address is also specified, which means that in this case, the exception shares the entry with all non-vector interrupt address.
- If the least significant bit of the CSR register mtvt2 is set to 1, the entry address shared by all non-vector interrupts is sent by the CSR
  The value of register mtvt2 (ignoring the value of the lowest 2 digits) is specified. In order for interrupts to be responded to and processed as fast a
  It is recommended to set the least significant bit of the CSR register mtvt2 to 1, that is, an independent entry point is designated by mtvt2
  The address is dedicated to all non-vector interrupts and is completely separate from the exception entry address (specified by the value of mtvec).

After entering all non-vectored interrupt entry address shared processor will start the implementation of a common software code, as Figure For the example shown in Figure 5-9, the content of this software code is usually as follows:

- First save the CSR registers mepc, mcause and msubm onto the stack. These CSR registers are saved for
  Ensure that the subsequent interrupt nesting functions correctly, because the new interrupt response will rewrite mepc, mcause, msubm values, so they need to be saved on the stack first.
- Save several general 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), this instruction is equivalent to a Nop instruction without any operation; if there is an interrupt waiting
  (Pending), the processor will:
  - Jump directly into the target address stored in the vector table entry of the interrupt, that is, the source of the interrupt Interrupt Service Routine (ISR).

- While jumping into the interrupt service routine, the hardware will also enable the global enable of the interrupt, that is, set mstatus The MIE field of the register is 1. After the global interrupt is enabled, new interrupts can be serviced to achieve interrupts. Nested effect.

- While jumping into the interrupt service routine, the "csrrw ra, CSR_JALMNXTI, ra" instruction will also reach The effect of JAL (Jump and Link), the hardware also updates the value of the Link register at the same time as the PC itself

Page 54

The return address as a function call. Therefore, the "csrrw ra, CSR_JALMNXTI, ra "instruction is re-executed, and it is re-judged whether there are interrupts waiting (Pending) to achieve the effect of interrupting tail biting.

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

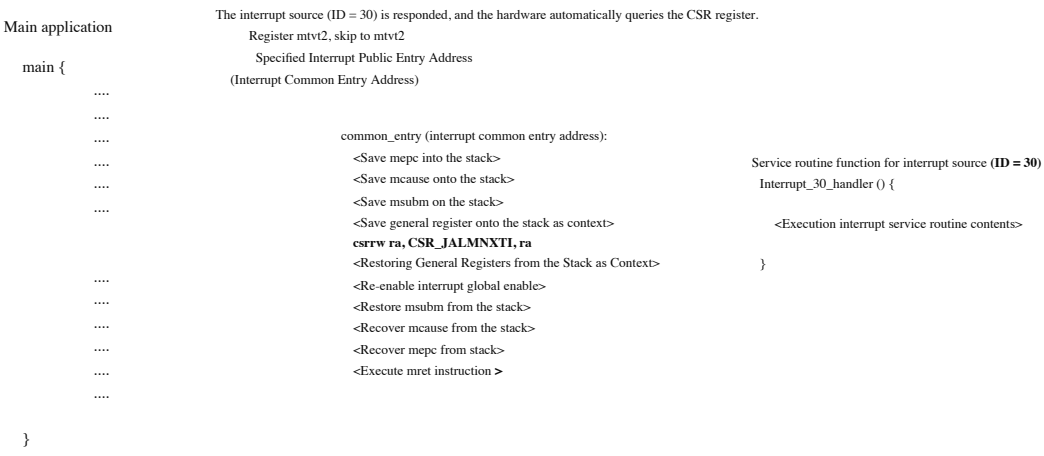


Figure 5-9 Example of non-vector processing mode for interrupts (Nesting is always supported)

Because in non-vector processing mode, the processor needs to execute a common piece of software code before jumping to the interrupt service routin The saving of the context, therefore, from the interrupt source pulled up to the processor to execute the first instruction in the interrupt service routine, it need Clock cycle overhead in the following areas:

- Overhead of the processor core to jump after responding to the interrupt. Ideally about 4 clock cycles.
- The processor core saves the overhead of the CSR registers mepc, mcause, and msubm onto the stack.
- The cycle overhead spent by the processor core to save the context. If the architecture of RV32E, you need to save 8 general

Page 55

Registers, if it is RV32I's architecture, 16 general registers need to be saved.

- The overhead of the processor core jumping to the Interrupt Service Routine (ISR). ideal

  It takes about 5 clock cycles in this case.

5.13.1.2 Interrupt nesting in non-vector processing mode

As mentioned above, non-vector processing modes can always support interrupt nesting, as shown in the example in Figure 5-10 : Assuming interrupts Three interrupt sources, source 30, 31, and 32, have arrived, and the level of interrupt source 32> Level of interrupt source 31> Medium Interrupt source 30 level ", then subsequent interrupts will interrupt interrupts that were being processed to form interrupt nesting.

Page 56

Main application

Respond to the interrupt: enter the interrupt common entry address
<Save context>

Enter service routine with interrupt **ID = 30**
<A part of the execution service program>

Nesting Occurs: Interrupt Public Entry Address
<Save context>

Enter service routine with interrupt **ID = 31**
<A part of the execution service program>

Nesting Occurs: Enter Interrupt Public Entry Address
<Save context>

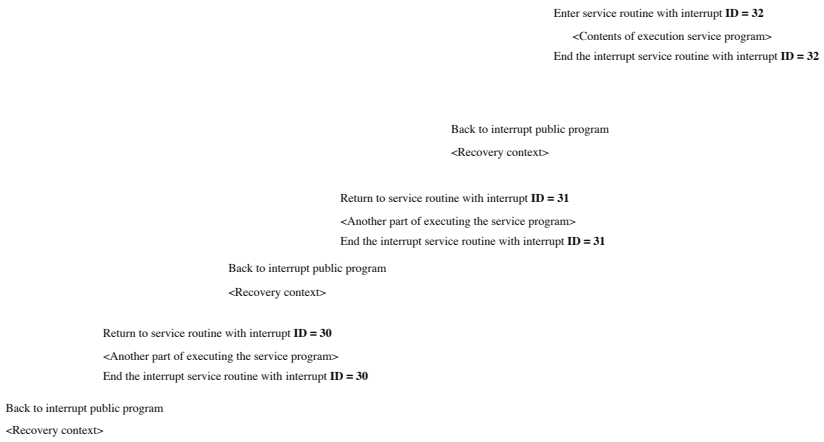


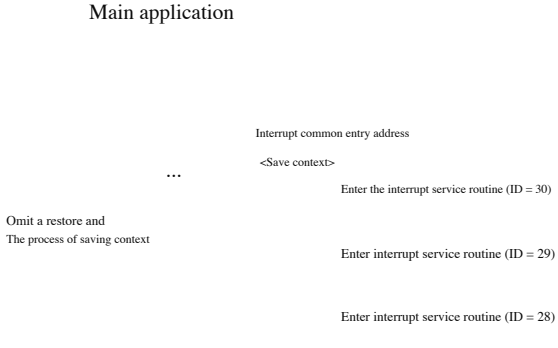
Figure 5-10 Three successive (non-vector processing) interrupts form a nest

5.13.1.3 Interrupt tail-biting in non-vector processing mode

For non-vector processing mode interrupts, the processor must
Saving and restoring text, so “interrupting tail biting” can save significant time (saving a back-to-back save context and
Restore context).

As mentioned above, in the common code segment shared by all non-vector interrupts, while jumping into the interrupt service routine, "csrrw "ra, CSR_JALMNXTI, ra" instruction will also achieve the effect of JAL (Jump and Link), and the hardware will update the Link at the same time. The value of the register is the PC itself of this instruction as the return address of the function call. Therefore, after returning from the interrupt service routi Return to the "csrrw ra, CSR_JALMNXTI, ra" instruction and re-execute, and re-judge whether there are interrupts waiting (Pending) to achieve the effect of interrupting tail biting.

An example is shown in Figure 5-11 : Suppose three interrupt sources, interrupt source 30, 29, and 28 arrive one after the other, and "interrupt source 3(Level "> =" level of interrupt source 29 "> =" level of interrupt source 28 ", then subsequent interrupts will not interrupt the previous Interrupts being processed (no interrupt nesting will occur), but will be placed in a Pending state. When interrupt source 30 finishes After processing, the interrupt processing of interrupt source 29 will be started directly, eliminating the intermediate "restore context" and "save context" proc



<Recovery context>

Figure 5-11 Interrupted tail biting

5.13.2. Vector processing mode

5.13.2.1 Features and delays of vector processing modes

If configured as vector processing mode, after the interrupt is responded by the processor core, the processor will directly jump into the interrupt. Vector table entry (Vector Table Entry) storage destination address, the interrupt service routine (Interrupt Service Routine, ISR), as shown in the example in Figure 5-12.

```
Main application

  main {
            ....
            ....
            ....
            ....
            ....
            ....

            ....
            ....
            ....
            ....
            ....
            ....

  }

Interrupt source (ID = 30) is responded, hardware automatically
    Query interrupt vector table, jump directly
        Corresponding interrupt service routine

Service routine function for interrupt source (ID = 30)
  Interrupt_30_handler () {

      <Execution interrupt service routine contents>
      <Execute mret instruction>
  }
```

Figure 5-12 Example of vector processing mode for interrupts

The vector processing mode has the following characteristics:

- In vector processing mode, the processor jumps directly to the interrupt service routine and does not save the context.
  The interrupt response delay is very short, from the interrupt source pulled up to the processor's execution of the first instruction in the interrupt s
  Basically, only the time overhead of the table lookup and jump by the hardware is needed, ideally about 6 clock cycles.
- For interrupt service routine functions in vector processing mode, be sure to use the special __attribute__ ((Interrupt)) to modify the interrupt service routine function.

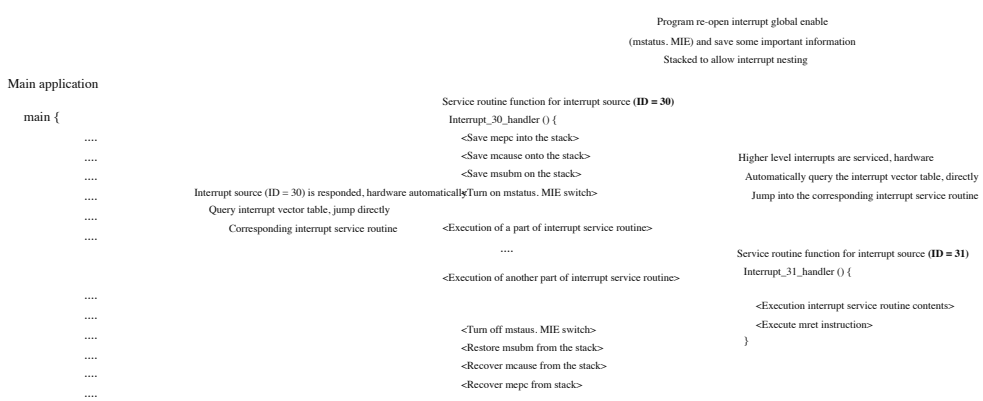
- In vector processing mode, the processor does not save the context before jumping into the interrupt service routine. Therefore, in theory, the ISR function itself cannot make sub-function calls (that is, it must be Leaf Function).

  - If the interrupt service routine function accidentally calls other sub-functions (not Leaf Functions), Processing will cause functional errors. In order to circumvent this accidental situation caused by accident, as long as a special __attribute__ ((interrupt)) to modify the interrupt service routine function, then the compiler will automatically Judgment, when the compiler finds that the function calls other sub-functions, it will automatically insert a piece of code for Save the context. Note: Although the correctness of the function is guaranteed in this case, the The overhead, which will actually increase the interrupt response delay (equivalent to non-vector mode) and cause code Expansion of Code Size. Therefore, in practice, it is not recommended if you use the vector processing mode Call other sub-functions in the interrupt service routine function in vector processing mode.

- In vector processing mode, the processor does not perform any special processing before jumping into the interrupt service routine. And because the processor core responds to the interrupt, the MIE field in the mstatus register will be automatically updated by the hardware to 0 (meaning that the interrupt is globally closed, so it cannot respond to new interrupts). Therefore, the vector processing mode is not supported b To interrupt nesting, in order to achieve the vector processing mode and interrupt nesting effect, as shown in Figure 5-13 To add a special push operation at the beginning of the interrupt service routine:

  - First save the CSR registers mepc, mcause and msubm onto the stack. Save these several CSR registers is In order to ensure that the subsequent interrupt nesting can function correctly, the new interrupt response will rewrite mepc, mcause, msubm, so they need to be saved on the stack first.
  - Re-enable the global enable of the interrupt, that is, set the MIE field of the mstatus register to 1. Turn on interrupt globally After being enabled, new interrupts can be responded to 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 send it in CSR Before the registers mepc, mcause, and msubm are out of the stack, the global interrupt needs to be turned off again to ensure that The atomicity of mepc, mcause, msubm recovery operations (not interrupted by new interrupts).

Page 60



```
}                                   } <Execute mret instruction>
```

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

5.13.2.2 Interrupt nesting in vector processing mode

As mentioned above, interrupts in the vector processing mode can also support interrupt nesting after special processing, as shown in Figure 5-14 . Example: Assume that the three interrupt sources 30, 31, and 32 have come one after the other, and the level of interrupt source 32> Level ">" Level of Interrupt Source 30 ", then subsequent interrupts will interrupt interrupts that were previously processed to form an interrupt nest.

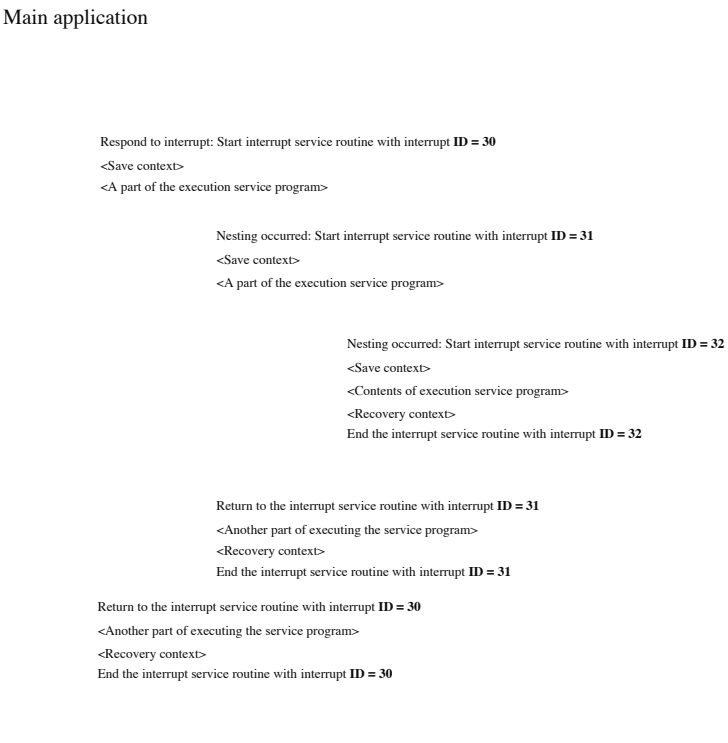


Figure 5-14 Three successive (vector processing mode) interrupts form a nest

5.13.2.3 Interrupted tail biting in vector processing mode

For vector processing mode interrupts, the processor does not perform contextual operations before jumping into the interrupt service routine. Save, so "interrupt tail bite" is not significant. Therefore, the interruption of vector processing mode is not interrupted ability.

Page 62

6. N200 Series Core TIMER and ECLIC Introduction

6.1. TIMER Introduction

6.1.1. Introduction to TIMER

Timer Unit (TIMER), which is mainly used to generate timer interrupts in the N20 er Interrupt) and Software Interrupt. See Section 5.3.2.1 and Section 5.3.2.2. Break and software interrupt details.

6.1.2. TIMER register

TIMER is a unit of memory address mapping:

- For the base address of the TIMER unit in the N200 series kernel, please refer to the Concise Data Manual of Nuclei_N200 Series Core Booklet.
- The register and address offsets in the TIMER unit are shown in Table 6-1 .

Table 6-1 Memory-Mapped Addresses of the TIMER Register

Offset address	read and write attributes in the module	Register name	Reset default	Function description
0x0	Read and write	mtime_lo	0x00000000	reflects the lower 32-bit value of the timer mtime, see section 6.1 section 4.3 to understand its details.
0x4	Read and write	mtime_hi	0x00000000	reflects the high 32-bit value of the timer mtime, see section 6.1 section 4.3 to understand its details.
0x8	Readable and writable	mtimecmp_lo	0xFFFFFFFF	Comparison value of the configuration timer mtimecmp lower 32 bits, See Section 6.1 .5 section for a detailed description.
0xC	Read and write	mtimecmp_hi	0xFFFFFFFF	Configure the comparison value of the timer mtimecmp high 32 bits, See Section 6.1 .5 section for a detailed description.
0xFF8	Read and write	mstop	0x00000000	Controls the pause of the timer, see Section 6.1.4 for details Detailed introduction.
0xFFC	Read and write	msip	0x00000000	Generate software interrupt, see Section 6.1.6 for detailed introduction Shao.

note:

- The TIMER register only supports aligned read and write accesses with an operation size (word).
- The register range of TIMER is 0x00 ~ 0xFF. The values in the addresses other than the registers listed in the table above are constant 0.

Page 63

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

6.1.3. Timing with the mtime register

TIMER can be used for real-time timing, the main points are as follows:

- A 64-bit mtime register is implemented in TIMER, which is composed of {mtime_hi, mtime_lo}.
  The register reflects the value of the 64-bit timer. The timer counts up and down according to the low-speed input beat signal
  The calculator is turned on by default, so it keeps counting.
- In the N200 series core, the self-incrementing frequency of this counter is controlled by the input signal mtime_toggle_a of the processor
  Please refer to the document "Nuclei_N200 Series Core Concise Data Sheet" for details of this input signal.

6.1.4. Pause the timer through the mstop register

Since the timer of TIMER will keep incrementing by default after power on, in order to turn off this timer in some special cases Device count, an mstop register is implemented in TIMER. As shown in Table 6- , MSTOP Only the lowest bit register in FIG 2 as having The effective bit is directly used as the pause control signal of the timer. Therefore, software can set the mstop register to 1 Pause the timer.

Table 6-2 Bit field of register mstop

domain name	Bit	Attributes	Reset value	description
Reserved	7: 1	Read-only, write ignore	N / A	Unused field with constant value 0
TIMESTOP	0	Read and write	0	Controls the timer to run or pause. in case The value of this field is 1, the timer is paused. Count, otherwise count up normally.

6.1.5. Generate timer interrupts through mtime and mtimecmp registers

TIMER can be used to generate timer interrupts, the main points are as follows:

- A 64-bit mtimecmp register is implemented in TIMER by {mtimecmp_hi, mtimecmp_lo}
  Concatenation, this register is used as the comparison value of the timer, assuming that the value of the timer mtime is greater than or equal to The value of mtimecmp generates a timer interrupt. Software can rewrite the value of mtimecmp or mtime
  (Make mtimecmp greater than the value of mtime) to clear the timer interrupt.

Note: The timer interrupt is connected to the ECLIC unit for unified management. For details about ECLIC, see Section 6.2 , Section.

6.1.6. Software Interrupt Generation via msip Register

TIMER can be used to generate software interrupts. A msip register is implemented in TIMER, as shown in Table 6-3 . Only the least significant bit of the msip register is the valid bit, which is directly used as a software interrupt. Therefore:

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

Note: The software interrupt unit is connected to ECLIC unified management, details about ECLIC, see section 6 .2 festival.

Table 6-3 Bit fields of the msip register

domain name	Bit	Attributes	Reset value	description
Reserved	7: 1	Read-only, write ignore	N / A	Unused field with constant value 0
MSIP	0	Read and write	0	This field is used to generate a software interrupt

6.2. Introduction to ECLIC

The N200 series core supports the "Improved Kernel Interrupt Controller" optimized on the basis of the RISC-V standard CLIC (Enhanced Core Local Interrupt Controller, ECLIC) "is used to manage all interrupt sources.

note:

- ECLIC only serves one processor core and is private to that processor core.
- ECLIC's software programming model is also backward compatible with standard CLIC.

Page 65

6.2.1. Introduction to ECLIC

Figure 6-1 ECLIC logical structure

ECLIC is used to arbitrate n external interrupt sources, send requests, and support interrupt nesting. ECLIC Register As shown in Table 6-5 , the logica ı in Figure 6-1. The related concepts are as follows:

- ECLIC interrupt target

- ECLIC interrupt source
- Number of ECLIC interrupt source
- ECLIC Register
- Enable bit of ECLIC interrupt source
- Wait flag for ECLIC interrupt source
- Level or edge properties of the ECLIC interrupt source
- ECLIC interrupt source levels and priorities

- Vector or non-vector processing of ECLIC interrupt sources
- Threshold level of ECLIC interrupt target
- ECLIC interrupt arbitration mechanism
- ECLIC interrupt response, nesting, tail biting mechanism

They 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). The relationship structure is shown in Figure 6-2 . Show.

Figure 6-2 ECLIC relationship structure

Page 67

6.2.3. ECLIC Interrupt Sources

As shown in Figure 6-2 , ECLIC can theoretically support up to 4096 interrupt sources (Interrupt Source). ECLIC defines the following characteristics and parameters for each interrupt source:

- Number (ID)
- Enable bit (IE)
- Wait flag (IP)
- level or edge properties (Level or Edge-Triggered)
- level and priority (Level and Priority)
- Vector or Non-Vector Mode

They are introduced separately below.

6.2.4. ECLIC interrupt source number (ID)

ECLIC assigns a unique number (ID) to each interrupt source. For example, suppose the hardware implementation of an ECLIC is true. 4096 IDs are being supported, the ID should be 0 to 4095. note:

- In the N200 series kernel, interrupts with interrupt ID numbers 0 to 18 are reserved as special kernel internals. Off.
- The interrupt source ID assigned by ordinary external interrupt starts from 19, and users can use it to connect external interrupt sources.

Details are shown in Table 6-4 .

Table 6-4 ECLIC interrupt source numbers and assignments

ECLIC interrupt number	Features	Interrupt source introduction
0	Reserve	*The N200* series kernel does not use this interrupt.
1	Reserve	*The N200* series kernel does not use this interrupt.
2	Reserve	*The N200* series kernel does not use this interrupt.
3	Software interrupt	Software interrupt generated by the TIMER unit of the N200 series core.
4	Reserve	*The N200* series kernel does not use this interrupt.
5	Reserve	*The N200* series kernel does not use this interrupt.
6	Reserve	*The N200* series kernel does not use this interrupt.
7	Timer interrupt	In the timer generated by the TIMER unit of the N200 series core

Page 68

		Off.
8	Reserve	*The N200* series kernel does not use this interrupt.
9	Reserve	*The N200* series kernel does not use this interrupt.
10	Reserve	*The N200* series kernel does not use this interrupt.
11	Reserve	*The N200* series kernel does not use this interrupt.
12	Reserve	*The N200* series kernel does not use this interrupt.
13	Reserve	*The N200* series kernel does not use this interrupt.
14	Reserve	*The N200* series kernel does not use this interrupt.
15	Reserve	*The N200* series kernel does not use this interrupt.
16	Reserve	*The N200* series kernel does not use this interrupt.
17	Reserve	*The N200* series kernel does not use this interrupt.
18	Reserve	*The N200* series kernel does not use this interrupt.
	External Interrupt	Normal external interrupts are provided for user connection. note: - Although ECLIC supports up to 4096 from the programming model

19 ~ 4095	Interrupt sources, but the number of interrupt sources supported by the actual hardware Information register clicinfo.NUM_INTERRUPT.

6.2.5. ECLIC Registers

ECLIC is a unit of memory address mapping:

- The base address of the ECLIC unit in the N200 series kernel is described in the Concise Data Manual of Nuclei_N200 Series Cores. Booklet.
- The register and address offsets in the ECLIC unit are shown in Table 6-5 .

Table 6-5 Intra-Unit Address Offsets of ECLIC Registers

	Attributes	name	width
0x0000	Read and write	cliccfg	8-bit
0x0004	Read-only, write-ignore	clicinfo	32-bit
0x000b	Read and write	mth	8-bit
0x1000 + 4 * i	Read and write	clicintip [i]	8-bit
0x1001 + 4 * i	Read and write	clicintie [i]	8-bit
0x1002 + 4 * i	Read and write	clicintattr [i]	8-bit
0x1003 + 4 * i	Read and write	clicintctl [i]	8-bit

note:

- The above i indicates the ID number of the interrupt, and the register with the [i] suffix indicates There is a separate register for each interrupt source.
- ECLIC registers support operation size (Byte), half-word, or word-aligned read-write access.
- Writes to the above “read-only” registers will be ignored, but will not

Page 69

- The actual ECLIC may not be configured with 4096 interrupt sources, so there is no The value of the corresponding register of the interrupt source is constant 0.
- Register range in ECLIC unit is 0x0000 ~ 0xFFFF, except for the above The values in the addresses other than the registers listed in the table are constant 0.

Each register is described in detail below.

6.2.5.1 Register cliccfg

The cliccfg register is a global configuration register. Software can configure several global parameters by rewriting this register. For information on the specific bit fields, see Table 6-6 .

Table 6-6 Bit field of register cliccfg

domain name	Bit	Attributes	Reset value	description
Reserved	7: 5	Read-only slightly	N / A	Unused field with constant value 0
nlbits	4: 1	Read and write	0	Used to specify the bits of the Level field in the clicintctl [i] register Number, see Section 6.2.9 for a detailed description.
Reserved	0	Read-only slightly	N / A	Unused field, value is constant 1

6.2.5.2 Register clicinfo

The clicinfo register is a global information register. Software can read several global parameters by reading this register. For details about the bit fields, see Table 6-7 .

Table 6-7 Bit field of register clicinfo

domain name	Bit	Attributes	Reset value	description
Reserved	31:25	Read-only slightly	N / A	Unused field with constant value 0
CLICINTCTLBITS	24:21	Read-only slightly	N / A	Used to specify the clicintctl [i] register Number of significant bits, see section 6.2.9 Section for details.
VERSION	20:13	Read-only slightly	N / A	The version number of the hardware implementation
NUM_INTERRUPT 12: 0		Read-only slightly	N / A	Number of interrupt sources supported by the hardware

Page 70

6.2.5.1 Register mth

The mth register is the threshold level register of the interrupt target. Software can configure the threshold of the interrupt target by rewriting this regist See Table 6-8 for information about the specific bit fields .

Table 6-8 Bit field of register mth

domain name	Bit	Attributes	Reset value	description
mth	7: 0	Read and write	N / A	Threshold level register for interrupt target, See Section 6.2.11 for detailed introduction Shao.

6.2.5.2 Register clicintip [i]

The clicintip [i] register is the wait flag register for the interrupt source. For details about the bit field, see Table 6-9 .

Table 6-9 Bit fields of the register clicintip [i]

domain name	Bit	Attributes	Reset value	description
Reserved	7: 1	Read-only, write ignore	N / A	Unused field with constant value 0
IP	0	Read and write	0	Interrupt source wait flag, see section 6.2 .7 section for a detailed description.

6.2.5.3 Register clicintie [i]

The clicintie [i] register is the enable register of the interrupt source. Refer to Table 6-10 for the specific bit fields .

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 with constant value 0
IE	0	Read and write	0	Enable bit for interrupt source, see section 6.2 Jie .6 understand its details.

6.2.5.4 Register clicintattr [i]

The clicintattr [i] register is an attribute register of the interrupt source. Software can configure several attributes of the interrupt source by rewriting See Table 6-11 for information about specific bit fields .

Page 71

Table 6-11 Bit field of register clicintattr [i]

domain name	Bit	Attributes	Reset value	description
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 with constant value 0
trig	2: 1	Read and write	0	Specify the level or edge attribute of this interrupt source Properties, see Section 6 2.2.8 for details Introduction.
shv	0	Read and write	0	Specifies that this interrupt source uses vector processing mode Still non-vector processing mode, 1 is vector Mode, 0 is non-vector mode, see section See Section 6.2. 10 for details.

6.2.5.5 Register clicintctl [i]

The clicintctl [i] register is the control register of the interrupt source. Software can configure the level of the interrupt source by rewriting this register. (Level) and Priority, the Level and Priority fields are dynamically allocated based on the value of cliccfg.nlbits, see The first 6.2 .9 section for a detailed description.

6.2.6. Enable Bit (IE) for ECLIC Interrupt Source

FIG 6- 2, ECLIC Each interrupt source is assigned an interrupt enable bit (IE) is reflected in the register In clicintie [i] .IE, its functions are as follows:

- The clicintie [i] register of each interrupt source is a readable and writable register of memory address To 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 1, it means that this interrupt source is turned on.

6.2.7. Wait flag (IP) for ECLIC interrupt source

FIG 6- 2, ECLIC Each interrupt source is assigned to one interrupt pending flag (IP), is reflected in the register In clicintip [i] .IP, its functions are as follows:

- If the IP bit of an interrupt source is high, it means that the interrupt source is triggered. The trigger condition of the interrupt source depends on whe For the properties of level trigger or edge trigger, please refer to Section 6.2.8 for details.
- The IP bit of the interrupt source is software readable and writable. The behavior of software writing the IP bit depends on whether it is level-trigger

Page 72

Attributes, see Section 6.2. 8 for details.

- For edge-triggered interrupt sources, the IP may also have hardware self-clearing behavior, please refer to Section 6.2.8 for details. Introduction.

6.2.8. Level or Edge-Triggered of ECLIC Interrupt Source

FIG 6- illustrated, ECLIC each interrupt source 2 can be set level-triggered or edge-triggered attribute (through register clicintattr [i] 's trig field), the main points are as follows:

- When clicintattr [i] .trig [0] == 0, set the interrupt property to a level-triggered interrupt:
  - If the interrupt source is configured as a level trigger, the IP bit of the interrupt source will reflect the level value of the interrupt source in real ti
  - 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 rea
    Therefore, the software's write operation to the interrupt IP bit will be ignored, that is, the software cannot be set or cleared by the write opera
    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 property to a rising edge
  Triggered interrupt:
  - If the interrupt source is configured to trigger on a rising edge, when ECLIC detects the rising edge of the interrupt source, the interrupt
    The source is triggered in ECLIC and the IP bit of the interrupt source is set high.
  - If the interrupt source is configured to trigger on a rising edge, software writes to the interrupt IP bit will take effect, that is, software
    The value of the IP bit can be set or cleared by a write operation.
  - Note: For interrupts triggered by rising edges, in order to improve the efficiency of interrupt processing, when the interrupt is sounded
    When the processor core jumps into the Interrupt Service Routines (ISR),
    ECLIC's hardware automatically clears the interrupted IP bit, eliminating the need for software to internally interrupt the IP of the interrupt
    Bit is cleared.
- When clicintattr [i] .trig [0] == 1 and clicintattr [i] .trig [1] == 1, set the interrupt property to falling edge
  Triggered interrupt:
  - If the interrupt source is configured to trigger on a falling edge, when ECLIC detects the falling edge of the interrupt source, the interrupt
    The source is triggered in ECLIC and the IP bit of the interrupt source is set high.
  - If the interrupt source is configured to trigger on a falling edge, software write to the interrupt IP bit will take effect, that is, software
    The value of the IP bit can be set or cleared by a write operation.
  - Note: For interrupts triggered by falling edges, in order to improve the efficiency of interrupt processing,

Page 73

When the processor core jumps into the Interrupt Service Routines (ISR),
ECLIC's hardware automatically clears the interrupted IP bit, eliminating the need for software to internally interrupt the IP of the interrupt
Bit is cleared.

6.2.9. Level and Priority of ECLIC Interrupt Sources

FIG 6- 2, ECLIC Each interrupt source can be provided a specific level and priority (via register clicintctl [i]), the main points are as follows:

- The clicintctl [i] register of each interrupt source is theoretically 8 bits wide, where the effective number of bits
  The CLICINTCTLBITS field of the clicinfo register. For example, suppose clicinfo.CLICINTCTLBITS
  The value of the field is 6, which means that only the upper 6 bits of the clicintctl [i] register are truly significant bits, and the value of the lowest
  The number 1, as shown in the example in Figure 6-3 .
  - Note: The value of the CLICINTCTLBITS field is a read-only fixed constant, which cannot be rewritten by software.
    The theoretical reasonable range is 2 <= CLICINTCTLBITS <= 8, the actual actual value is determined by the processor core
    Hardware implementation decisions.
- In the valid bit of the clicintctl [i] register, there are two dynamic fields that are used to specify the level of the interrupt source
  (Level) and Priority. The width of the Level field is specified by the nlbits field of the cliccfg register. example
  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 Level
  The other low-order significant bits are the Priority field, as shown in the example in Figure 6-3 .
  - Note: The value of the cliccfg.nlbits field is a readable and writable field, which can be overwritten by software.

Figure 6-3 Format example of the register clicintctl [i]

- The main points related to the level of the interrupt source are as follows:
  - The numeric value of Level is interpreted in a left-aligned manner, and the effective bit width (specified by cliccfg.nlbits) is beyond

Page 74

The low-order bits are all filled with a complement constant of 1, as shown in the example in Figure 6-4 .

- Note: If cliccfg.nlbits> clicinfo.CLICINTCTLBITS, it means that nlbits indicates
  If the number of bits exceeds the valid bits of the clicintctl [i] register, all the excess bits are complemented by a constant of 1.
  filling.
- Note: If cliccfg.nlbits = 0, the numeric value of Level is considered to be a fixed 255. As shown
  Examples are shown in 6-5 .

- The higher the numerical value of Level, the higher the level. Note:
  - High-level interrupts can interrupt low-level interrupt processing to form interrupt nesting, see section 5. 11
    A detailed introduction to the section.
  - Multiple interrupts wait at the same time (IP bit is high), ECLIC needs arbitration to decide which interrupt is sent to the kernel
    For processing, the Level digital value of each interrupt source needs to be referenced during arbitration. See 5 .5 detailed section
    Detailed introduction.

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

Page 75

Figure 6-5 Some examples of cliccfg settings

- The main points related to the priority of the interrupt source are as follows:
  - The numeric value of Priority is also interpreted in a left-aligned manner, and the effective bit width is (Clicinfo.CLICINTCTLBITS -cliccfg.nlbits) All the low-order bits are filled with the complement constant 1. Charge.
  - The higher the value of Priority, the higher the priority. Note:
    - Interrupt priority (Priority) does not participate in the judgment of interrupt nesting, that is, whether interrupts can be nested and interrupt (Priority) value has nothing to do with the value of the interrupt level (Level) turn off.
    - When multiple interrupts are pending simultaneously, ECLIC needs arbitration to decide which interrupt is sent to the kernel for processin It is necessary to refer to the Priority digital value of each interrupt source when arbitrating. See section 6.2.12 for details Introduction.

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

Each ECLIC interrupt source can be set to vector or non-vector processing (through the configuration register clicintattr [i] shv Domain), the main points are as follows:

- If configured as vector processing mode (clicintattr [i] .shv = 1), the interrupt will be processed by the processor The processor directly jumps into the target address stored in the interrupted Vector Table Entry. About interruption Details of the amount of processing modes, see Section 5. The 13 is Day.
- If configured for non-vector processing mode (clicintattr [i] .shv = 0), the interrupt is responded by the processor core After that, the processor jumps directly to the entry address shared by all interrupts. Detailed introduction to interrupt non-vector processing mod See Section 5.13 .

6.2.11. Threshold levels for ECLIC interrupt targets

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

- The mth register is a complete 8-bit register. All bits are readable and writable. Software can configure the destination by writing to this register. Standard threshold. Note: This threshold is a numerical value of a level.
- The “Level digital value” of the interrupt finally arbitrated by ECLIC is only higher than the “Value in the mth register”,

Page 76

This interrupt can then be sent to the processor core.

6.2.12. Arbitration mechanism for ECLIC interrupts

FIG 6- principle shown, ECLIC choose arbitrate all its interrupt source 2 as follows:

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

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

- The rules for arbitration from all participating interruption sources are:
  - First, determine the level. The higher the Level value is, the higher the arbitration priority is.
  - If the Levels are equal, the priority is determined next. The interrupt source with a higher Priority value will be The higher the arbitration priority.
  - If Level and Priority are both equal, judge the interrupt ID again. The interrupt source with a larger interrupt ID has the The higher the arbitration priority.
- If the value of the Level of the last arbitrated interrupt source is higher than the threshold level (mth) of the interrupt target, the final Interrupt request, pull up the interrupt request signal to the processor core.

6.2.13. ECLIC interrupt response, nesting, tail biting mechanism

After the ECLIC interrupt request is sent to the processor core, the processo sp rough ECLIC and the kernel protocol At the same time, it can support interrupt nesting, fast tail biting and so on. See S ec ction 5.12 for detailed introductions Shao.

Page 77

7. N200 Series Core CSR Register Introduction

7.1. N200 Series Core CSR Register Overview

RISC-V's architecture defines some control and status registers (Control and Status Registers, CSR). To configure or record some running status. The CSR register is a register inside the processor core, using its proprietary 12-bit address Coding space.

7.2. CSR Register List for N200 Series Cores

The list of CSR registers supported by Nuclei N200 series is shown in Table 7-1 , which includes the CSR register of RISC-V standard (RV32IMAFDC architecture supports Machine Mode and User Mode) and N200 series processor cores Meaning extended CSR register.

Table 7-1 List of CSR registers supported by the N200 series kernel

Types of	**CSR** address	Read and write attributes	ame	Full name
RISC-V standard	0xF11	MRO	mvendorid	Commercial supplier number register Vendor ID Register)
	0xF12	MRO	marchid	1. architecture number register

CSR (Machine Mode)				ID Register)
	0xF13	MRO	mimpid	1. hardware implementation number register Implementation ID Register)
	0xF14	MRO	mhartid	Hart ID Register
	0x300	MRW	mstatus	Exception handling status register
	0x301	MRO	misa	Instruction Set Architecture Register (Machine ISA Register)
	0x304	MRW	mie	1. local interrupt mask control register Interrupt Enable Register)
	0x305	MRW	mtvec	Exception entry base address register
	0x307	MRW	mtvt	ECLIC Interrupt Vector Table Base Address
	0x340	MRW	mscratch	1. temporary register Register)
	0x341	MRW	mepc	1. exception PC register Program Counter)
	0x342	MRW	mcause	1. exception cause register Register)
	0x343	MRW	mtval	1. outlier register Register)

Page 78

	0x344	MRW	mip	1. interrupt wait register Pending Register)
	Ox345	MRW	mnxti	Standard registers are used to enable interrupts and process the next Interrupt and return to the next interrupted handler entry address
	0x346	MRO	mintstatus	Standard register is used to save the current interrupt level
	0x348	MRW	mscratchcsw	Standard register for swapping when privileged mode changes mscratch and destination register value
	0x349	MRW	mscratchcswl	Standard register is used to transfer when interrupt level changes Swap mscratch with destination register value
	0xB00	MRW	mcycle	Lower 32 bits of the cycle counter of Cycle counter)
	0xB80	MRW	mcycleh	Upper 32 bits of the cycle counter 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	Upper 32 bits of the complete instruction counter (Upper 32 bits of Instructions-retired counter)
	N / A	MRW	mtime	1. timer register register)
	N / A	MRW	mtimecmp	1. timer compare register timer compare register)
	N / A	MRW	msip	Machine mode software interrupt wait register (Machine-mode Software Interrupt Pending Register)
	N / A	MRW	stop	Standard register to stop timer
	0x3A0 + x	MRW	pmpcfg <x>	PMP entry authority configuration register
	0x3B0 + x	MRW	pmpaddr <x>	Address configuration register for PMP entry x
	0x001	URW	fflags	1. floating-point accumulated exception
RISC-V standard				Exceptions).
CSR (User				Note: This register is only available if floating-point instructions are configured.
Mode)				("F" or "D" instruction subset)
				in.
	0x002	URW	frm	1. floating-point dynamic rounding mode Dynamic Rounding Mode).
				Note: This register is only available if floating-point instructions are configured.
				("F" or "D" instruction subset)
				in.
	0x003	URW	fcsr	1. floating-point control and status register Control and Status Register).
				Note: This register is only available if floating-point instructions are configured.
				("F" or "D" instruction subset)
				in.
	0xC00	URO	cycle	read-only copy of mcycle register
				Note: Is this register available in User Mode?
				Read the CY ratio from the CSR register mcounteren
				Special domain control, see Section 7.34 for details
				Details.
	0xC01	URO	time	read-only copy of mtime register
				Note: Is this register available in User Mode?
				Read the TM ratio from the CSR register mcounteren
				Special domain control, see Section 7.34 for details

Page 79

				Details.
	0xC02	URO	instret	read-only copy of the minstret register Note: Is this register available in User Mode? Read the IR ratio by the CSR register mcounteren Special domain control, see Section 7.34 for details Details.
	0xC80	URO	cycleh	Read-only copy of mcycleh register Note: Is this register available in User Mode? Read the CY ratio from the CSR register mcounteren Special domain control, see Section 7.34 for details Details.
	0xC81	URO	timeh	read-only copy of mtimeh register Note: Is this register available in User Mode? Read the TM ratio from the CSR register mcounteren Special domain control, see Section 7.34 for details Details.
	0xC82	URO	instreth	read-only copy of the minstreth register Note: Is this register available in User Mode? Read the IR ratio by the CSR register mcounteren Special domain control, see Section 7.34 for details Details.
N200 Custom **CSR**	0x320	MRW	mcountinhibit	Custom registers are used to control the on and off of the counter shut down
	0x7c3	MRO	mnvec	NMI Processing Entry Base Address Register
	0x7c4	MRW	msubm	Custom registers are used to save the current core Trap types and Trap classes before entering Trap type.
	0x7d0	MRW	mmisc_ctl	Custom Registers Used to Control NMI Handlers Entry address
	0x7d6	MRW	msavestatus	Custom register for mstatus value
	0x7d7	MRW	msaveepc1	Custom register for first level nested NMI Or abnormal mepc
	0x7d8	MRW	msavecause1	Custom register for first level nested NMI Or unusual mcause
	0x7d9	MRW	msaveepc2	Custom register to hold second level nested NMI Or abnormal mepc
	0x7da	MRW	msavecause2	Custom register to hold second level nested NMI Or unusual mcause
	0x7eb	MRW	pushmsubm	Custom register is used to store the value of msubm Stack address space
	0x7ec	MRW	mtvt2	Custom register is used to set non-vector interrupt processing Interrupt entry address of the mode
	0x7ed	MRW	jalmnxti	The custom register is used to enable the ECLIC interrupt. Register read operation can handle the next interrupt at the same time Returns the entry address of the next interrupt handler, And jump to this address.
	0x7ee	MRW	pushmcause	Custom register is used to store the value of mcause Stack address space
	0x7ef	MRW	pushmepc	Custom registers are used to store the value of mepc on the heap Stack address space
	0x811	MRW	sleepvalue	WFI Sleep Mode Register
	0x812	MRW	txevt	Send Event Register

Page 80

	0x810	MRW	wfe	Wait for Event control register

note:

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

7.3. Access to CSR registers of N200 series cores

Nuclei N200 series kernels have the following access rights to CSR registers:

- Whether in Machine Mode or User Mode:
  - If a read or write operation is performed on a non-existing CSR register address range, Illegal Instruction will be generated Exception.
  - Note: The fflags, frm, and fcsr registers are only configured with floating-point instructions ("F" or "D" instruction subset) Will exist.
- In Machine Mode:
  - Reading and writing to the CSR register of the MRW or URW attribute is all normal.
  - Reading the CSR register of the MRO or URO attribute is all normal.
  - Writing to the CSR register of the MRO or URO attribute will generate Illegal Instruction Exception.
- In User Mode:
  - Read and write operations to the CSR register of the URW attribute are all normal.
  - Reading the CSR register of the URO attribute is all normal.
    - Note: For cycle, cycleh, time, timeh, instret, instreth properties of URO attributes The readability of the device is also controlled by the relevant bit field of mcounteren, see 7. 4.34 for details. situation.

  - Writing to the CSR register of the URO attribute will generate Illegal Instruction Exception.
  - If read and write operations are performed to the CSR register of the MRW or MRO attribute, Illegal will be generated Instruction Exception.

7.4. RISC-V Standard CSR Supported by N200 Series Core

This section introduces the custom CSR register of the N200 series processor core (RV32IMC architecture supports Machine Mode and User Mode).

7.4.1. Misa

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

The most significant two bits of the misa register indicate the number of architecture bits supported by the current processor:

- If the value of the most significant two bits is 1, it indicates that it is a 32-bit architecture (RV32).

- If the value of the most significant two bits is 2, it indicates that the current architecture is 64-bit (RV64).
- If the value of the most significant two bits is 3, it means the current 128-bit architecture (RV128).

The lower 26 bits of the misa register are used to indicate the different modular instruction subsets in the RISC-V ISA supported by the current processo
The modular instruction subset represented by each bit is shown in Figure 7-1 . The other unused bit fields of this register are constant 0.

Page 82

Figure 7-1 Modular instruction subset 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 documentation, allowing some processors
Is designed to dynamically configure certain features. However, in the implementation of the N200 series kernel, the misa register is a read-only register.
Constantly reflects the modular subset of ISA supported by different model processor cores. For example, if the N205 core supports RV32IMC, it reflects
In this register, the value of the most significant two bits is 1, and the value of the corresponding field of I / M / C in the lower 26 bits is high.

7.4.2. Mie

The control bit of the mie register has no effect in ECLIC interrupt mode. Reading mie returns all 0s.

Page 83

7.4.3. Mvendorid

This register is read-only and is used to reflect the Vendor ID of the processor core.

If the value of this register is 0, it means that this register is not implemented.

7.4.4. Marchid

This register is a read-only register that reflects the hardware-implemented microarchitecture number of the processor core.
ID).

If the value of this register is 0, it means that this register is not implemented.

7.4.5. Mimpid

This register is read-only and is used to reflect the hardware implementation number (Implementation ID) of the processor core.

If the value of this register is 0, it means that this register is not implemented.

7.4.6. Mhartid

This register is read-only and is used to reflect the current Hart ID.

Hart (referred to as "Hardware Thread") means a hardware thread, multiple hardware may be implemented in a single processor core
Thread, such as hardware hyper-threading technology, each thread has its own independent register file and other context information
Source, but most of the computing resources are reused by all hardware threads, so the area efficiency is very high. Hyperthreading in such hardware
In the processor, there are multiple hardware threads (Hart) in a core.

The Hart number value in the N200 processor core is controlled by the input signal core_mhartid. Note: The RISC-V architecture requires that
If in a single Hart or multiple Hart system, at least one Hart number must be 0.

7.4.7. Mstatus

Page 84

The mstatus register is a status register in Machine Mode. Control bits in the mstatus register
The fields are shown in Table 7-2 .

Table 7-2 Control bits of the mstatus register

area	Bit	Reset value	description
Reserved	2: 0	N / A	Unused fields are constant 0
MIE	3	0	See Section 7.4.8 for details
Reserved	6: 4	N / A	Unused fields are constant 0
MPIE	7	0	See Section 7.4.9 for details
Reserved	10: 8	N / A	Unused fields are constant 0
MPP	12:11	0	See Section 7.4.9 for details
FS	14:13	0	See Section 7.4.10 for details
XS	16:15	0	See Section 7.4.11 for details
MPRV	17	0	See Section 7.4.12 for details
Reserved	30:18	N / A	Unused fields are constant 0
SD	31	0	See Section 7.4.13 for details

7.4.8. MIE Domain for mstatus

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

When the value of the MIE field is 1, the global switch of the interrupt is on, and the interrupt can be responded normally;

When the value of the MIE field is 0, it means that the interrupt is globally disabled. The interrupt is masked and cannot be responded to.

Note: When the N200 series processor core enters the exception, interrupt, or NMI processing mode, the value of MIE will be updated to 0 (meaning that the interrupt is masked after entering 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, NMI and before interrupt mstatus.MIE, automatic recovery in Privilege Mode.

N200 series processor cores update the hardware behavior of the MPIE and MPP domains in the mstatus register when an exception occurs, see

3.4. Section 5 for the details.

Update the mstatus register when the N200 series processor core exits abnormally (execute the mret instruction in exception handling mode) For the hardware behavior of the MPIE and MPP domains, see Section 3.5.2 for details.

When the N200 series processor core enters the NMI, it updates the hardware behavior of the mstatus register MPIE and MPP domains. See 4.3 .4 section for the details.

Update the mstatus register when the N200 series processor core exits NMI (execute mret instruction in exception handling mode) For the hardware behavior of the MPIE and MPP domains, see Section 4.4.2 for details.

The hardware behavior of updating the mstatus register MPIE and MPP domains when the N200 series processor core enters an interrupt, see See Section 5.6. 5 for details.

Update the mstatus register when the N200 series processor core exits the interrupt (execute mret instruction in exception handling mode) For the hardware behavior of the MPIE and MPP domains, see Section 5.7.2 for details.

Note: The values of the mstatus.MPIE and mstatus.MPP fields are the same as the values of the mcause.MPIE and mcause.MPP fields Is a mirror relationship, that is, under normal circumstances, the value of the mstatus.MPIE domain and the value of the mcause.MPIE domain are always exa 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 status of the floating-point unit.

Note: This field only exists if a floating-point instruction ("F" or "D" instruction subset) is configured.

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

Figure 7-2 Status code represented by the FS domain

The update guidelines for FS domains are as follows:

- The default value after power-on of the FS is 0, which means that the status of the floating-point unit is Off. So in order to be able to use floating poin
  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
  can.
- If the value of FS is 1 or 2, after any floating-point instruction is executed, the value of FS will automatically switch to 3, indicating
  The status of the floating-point unit is Dirty (the state has changed).
- If the processor does not want to use the floating-point arithmetic unit (for example, power off the floating-point unit to save power), you can use CSF
  The instruction sets the FS field of the mstatus register to 0 to disable the function of the floating-point unit. When the work of a floating-point unit
  After it can be turned off, any operation that accesses the floating-point CSR register or any behavior that executes floating-point instructions will r
  Illegal Instruction is abnormal.

In addition to the above functions, the value of the FS field is also used as guidance information for the operating system during context switching. 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 instructions. Order unit status.

The XS domain is defined as a read-only domain in the standard RISC-V "Privileged Architecture Document Version 1.10", which is used to reflect all The total state of the extended instruction unit. But please note: In the hardware implementation of the N200 series processor core, the XS domain is designed Writable and readable domain, its role is completely similar to FS domain. Software can change the value of XS domain to enable or disable coprocessor. The purpose of the extended instruction unit.

Similar to the FS domain, in addition to the above functions, XS is also used for operating system guidance information when performing context switc For interested users, please refer to the original RISC-V "Privileged Architecture Document Version 1.10".

7.4.12. MPRV domain for mstatus

The MPRV field in the mstatus register is used to control the reading and writing of data in the memory in Machine Mode (Load and Store) operation is regarded as an operation in User Mode for PMP protection.

Page 87

If the value of the MPRV field is 1, and the value of the MPP field indicates User Mode, the PMP reads and writes the data in the memory (Load and Store) operations are considered to occur in User Mode for protection, and are performed according to the R / W attribute configured by the entry. Line permission confirmation, if the permission is violated, an exception is triggered. See Section 8.4 for details.

7.4.13. SD domain of mstatus

The SD domain in the mstatus register is a read-only domain, which reflects that the XS domain or FS domain is in a dirty state. Its logical relationship expression is: SD = ((FS == 11) OR (XS == 11)).

The reason for setting this read-only SD domain is to facilitate software to quickly query whether the XS domain or FS domain is dirty. Status, so you can quickly determine whether the context of the floating-point unit or the extended instruction unit needs to be guaranteed when the context is Save. For interested users, please refer to the original RISC-V "Privileged Architecture Document Version 1.10".

7.4.14. Mtvec

The mtvec register is used to configure the entry addresses for interrupt and exception handlers.

- The key points when mtvec configures the interrupt handler entry address are as follows:
  - The exception handler uses the 4-byte aligned mtvec address (replace the lower 2 bits of mtvec with 0) as the input
    - Address.
- When mtvec configures the entry address of the interrupt program:
  - When mtvec.MODE! = 6'b000011, the processor uses the "default interrupt mode".
  - When mtvec.MODE = 6'b000011, the processor uses "ECLIC interrupt mode". This mode is recommended formula.
    - The entry address and key points when the interrupt is in non-vector processing mode are described in Section 5. 13.2 .
    - The entry address and key points when the interrupt is in vector processing mode are described in Section 5. 13.1 .

The address bit fields of the mtvec register are shown in Table 7-3 .

Page 88

Table 7-3 Control bits of mtvec register

area	Bit	description
ADDR	31: 6	mtvec address

MODE	5: 0	■ The MODE field is the interrupt processing mode control field: • 000011: ECLIC interrupt mode (recommended mode) • Others: default interrupt mode

7.4.15. Mtvt

The mtvt register is used to store the base address of the ECLIC interrupt vector table. This base address 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 As shown in Table 7-4 .

Table 7-4 mtvt alignment

Maximum number of interrupts	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

7.4.16. Mscratch

The mscratch register is used by the program in Machine Mode to temporarily save some data. The mscratch register can provide Provide a save and restore mechanism, for example, after entering the interrupt or exception handling mode, (SP) register is temporarily stored in the mscratch register, then the mscratch register is saved before exiting the exception handler The value read out is restored to the user stack pointer (SP) register.

7.4.17. Mepc

The mepc register is used to store the PC value of the instruction that the processor was executing before entering the exception, as the return address of

In order to understand this register, please refer to Chapter 3 for a systematic understanding of exceptions.

note:

■ When the processor enters the exception, the mepc register is updated to reflect the PC value of the instruction currently encountering the exception.

■ It is worth noting that although the mepc register is automatically updated by hardware when an exception occurs, the mepc register It is also a readable and writable register (in Machine Mode), so software can also directly write the register To modify its value.

Each address bit field of the mepc register is shown in Table 7-5 .

Table 7-5 Control bits of the mepc register

area	Bit	description

EPC	31: 1	Stores the PC value of the instruction that the processor was executing before the exception occurred
Reserved	0	Unused fields are constant 0

7.4.18. Mcause

The mcause register is used to save the cause of the error before entering the NMI, exception and interrupt, so as to facilitate the trap cause Diagnosis and commissioning.

Table 7-6 shows the address bit fields of the mcause register .

Table 7-6 Control bits of mcause register

area	Bit	description
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 privilege mode before interrupt, same as mstatus.mpp

MPIE	27	Interrupt enable before entering interrupt, same as mstatus.mpie
Reserved	26:24	Unused fields are constant 0
MPIL	23:16	Previous interrupt level
Reserved	15:12	Unused fields are constant 0
EXCCODE	11: 0	Exception / interrupt code

note:

- The MPIE and MPP fields of the mstatus register are mirrored with the MPIE and MPP fields of mcause.
- mcause.EXCCODE of NMI may be 0x1 or 0xfff. The actual value is controlled by mmisc_ctl.

  Test 7.5. Section 4.

7.4.19. Mtval (mbadaddr)

mtval register (aka mbadaddr, some versions of the toolchain only recognize this name), used to save before entering the exception The encoding value of the error instruction or the address value of the memory access in order to diagnose and debug the cause of the exception.

In order to understand this register, please refer to Chapter 3 for a systematic understanding of exceptions.

When the N200 series processor core enters an exception, the mtval register is updated at the same time to reflect the information that the exception is c

7.4.20. Mip

The control bit of the mip register has no effect in ECLIC interrupt mode. Reading mip returns all 0s.

7.4.21. Mnxti

mnxti (Next Interrupt Handler Address and Interrupt-Enable CSR) can be used by software access

Handle the next interruption in the same Privilege Mode without causing flushing pipelines and context save restoration.

The mnxti register can be accessed through the CSRRSI / CSRRCI instruction, and the return value of the read operation is the handler of the next inter Address, and the mnxti writeback operation updates the status of the interrupt enable.

Page 91

note:

1. For interrupts in different Privilege Modes, the hardware will handle them in an interrupt nested manner, so mnxti will only handle The next interrupt in the same Privilege Mode.
2. The mnxti register is not the same as a regular CSR instruction, and its return value is the same as the RMW (read-modify-write) operation with different values:
   - The return value of mnxti's CSR read operation has the following two cases:
     - When the following conditions occur, the return value is 0.
       - No interrupts that can be responded to
       - The current highest priority interrupt is a vector interrupt
     - When the interrupt is a non-vector interrupt, the interrupt handler entry address of this interrupt is returned.
   - The CSR write operation of mnxti will update the following registers and register fields:
     - mstatus is the destination register for the current RMW (read-modify-write) operation
     - mcause.EXCCODE field and the interrupt id that will be updated to the current response interrupt
     - The mintstatus.MIL field is updated to the interrupt level (Level) currently responding to the interrupt.

7.4.22. Mintstatus

The mintstatus register holds the interrupt level of each active interrupt in 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 fields are constant 0
UIL	7: 0	User Mode active interrupt level

Page 92

7.4.23. Mscratchcsw

The mscratchcsw register is used to speed up the exchange of the destination register and the mscratch value when switching between multiple privileg

Interrupt processing.

When using the CSR instruction with read operation to access mscratchcsw, when the privilege mode is not consistent before and after the interrupt, Register operation shown by the instruction:

```
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
```

Interrupt occurs when the processor is in low privilege mode (Privilege Mode), the processor enters high privilege mode to handle interrupts. When processing interrupts, you need to use the stack to save the processor state before entering the interrupt. If you continue to use the heap in low privilege Stack pointer (SP), the data of the stack in high privilege mode will be stored in the accessible area of low privilege mode, resulting in high The data of the right mode is leaked to the security hole of the low privilege mode. To avoid this security vulnerability, the RISC-V architecture provides In the low-privileged mode, the stack pointer (SP) of the high-privileged mode needs to be saved to the mscratch register. After privileged mode, the processor can use the value of the mscratch register to restore the stack pointer (SP) of the highly privileged mode.

Using regular instructions to execute the above program requires a lot of cycles. For this reason, the RISC-V architecture is defined mscratchcsw register, execute mscratchcsw register instruction immediately after entering interrupt, exchange mscratch and SP Value, used to restore the stack pointer (SP) of high privilege mode, while backing up the stack pointer (SP) of low privilege mode to mscratch register. Before executing the mret instruction to exit the interrupt, also add an mscratchcsw instruction to exchange the mscratch register And the stack pointer (SP) value, back up the stack pointer (SP) in high privilege mode to mscratch again, while restoring low privilege Stack pointer (SP) for the mode. In this way, only two instructions are needed to solve the stack pointer (SP) switch of different privilege modes Problem, speeding up interrupt processing.

Note: In order to avoid the vulnerability of virtualization, the software cannot directly read the processor's current privileged mode (Privilege Mode). If software attempts to access mscratchcsw in a given privileged mode in a lower privileged mode, do a register swap The operation causes the processor to enter Trap, so mscratchcsw does not cause a virtualization vulnerability.

7.4.24.mscratchcswl

The mscratchcswl register is used to switch the value of the destination register and mscratch to add when switching between multiple interrupt levels. Quick interrupt processing.

Use the CSR instruction with read operation to access mscratchcsw. When the privilege mode is unchanged, interrupt programs and applications occur. When switching, there are register operations shown in the following pseudo instructions:

```
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
```

Separating the stack space of interrupt handler tasks from application tasks in single privilege mode can increase robustness,
Reduces space usage and helps system debugging. Interrupt handler tasks have a non-zero interrupt level and application tasks have zero
Interrupt level. Based on this feature, the RISC-V architecture defines the mscratchcswl register. Similar to mscratchcsw, in
Adding a mscratchcswl to the interrupt program entry and exit can achieve a fast heap between the interrupt handler and the application
The stack pointer switch ensures that the stack space of the interrupt handler and application is separated.

7.4.25. Mcycle and mcycleh

The RISC-V architecture defines a 64-bit wide clock cycle counter to reflect how many clock cycles the processor has executed
period. This counter keeps incrementing as long as the processor is executing.

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

The mcycle and mcycleh registers can be used to measure processor performance and have readable and writable attributes, so software can
The CSR instruction rewrites the values in the mcycle and mcycleh registers.

Considering that this counter consumes some dynamic power consumption, in the implementation of N200 series processors,
An additional control field is added to the mSRinhibit of the CSR register. Software can configure this control field to set mcycle and

The counter corresponding to mcycleh stops counting, thereby stopping the counter to save power when performance measurement is not needed. please
See 7.5. Section 1 mcountinhibit register for more information.

Note: If this counter does not count in debug mode, the counter will only advance in normal function mode.
Row count.

7.4.26. Minstret and minstreth

The RISC-V architecture defines a 64-bit wide instruction completion counter to reflect how many instructions the processor has successfully executed
make. This counter is incremented whenever the processor successfully completes an instruction.

The minstret register reflects the low 32-bit value of the counter, and the minstreth register reflects the high 32-bit value of the counter
value.

The minstret and minstreth registers can be used to measure processor performance and have readable and writable attributes, so software
The values in the minstret and minstreth registers can be overwritten by the CSR instruction.

Considering that this counter consumes some dynamic power consumption, in the implementation of the N200 series processor core, in
An additional control domain is added to the custom CSR register mcountinhibit. Software can configure this control domain to
The counters for minstret and minstreth stop counting, thus stopping the counter to save time when performance is not needed to be measured
The role of electricity. See 7.5 .1 Section mcountinhibit register for more information.

Note: If this counter does not count in debug mode, the counter will only advance in normal function mode.
Row count.

7.4.27. Mtime, mtimecmp, msip, and stop

The RISC-V architecture defines a 64-bit timer according to the system's low-speed Real Time Clock frequency.
Time it out. The value of this timer is reflected in the mtime register in real time. The RISC-V architecture also defines a 64-bit

mtimecmp register, which is used as the comparison value of the timer, assuming the value of the timer mtime is greater than or equal to
The value of mtimecmp generates a timer interrupt. Note: The RISC-V architecture does not set the mtime and mtimecmp registers
It is defined as a CSR register, but a system register defined as a Memory Address Mapped.
The RISC-V architecture of the bank's memory-mapped address is not specified, but is left to the kernel designer to implement.

Page 95

The RISC-V architecture defines a software interrupt that can be triggered by software writing a 1 to msip register. About software interrupts
For information, see Section 6.1.6 . Note: The RISC-V architecture does not define the msip register as a CSR register, but defines it as a memory
Memory address mapping system register, the specific memory mapping address RISC-V architecture is not specified, but is left to the kernel design
The planners implement it by themselves.

In the implementation of the N200 series processor core, mtime / mtimecmp / msip / stop are implemented by the TMR unit.
For the main TMR implementation points of the N200 and the memory address range allocated by mtime / mtimecmp / msip / stop, see
Nuclei_N200 Series Concise Data Sheet.

Note: Because the address of mtime / mtimecmp / msip / stop is the memory address space, its access rights are determined by
PMP is set and protected.

Considering that timer counting consumes some dynamic power consumption, in the implementation of N200 series processors,
An additional control field is added to the stop register. Software can configure this control field to stop the timer corresponding to mtime.
So stop the timer to save power when not needed.

Note: If this counter does not count in debug mode, the counter will only advance in normal function mode.
Row count.

7.4.28.fcsr

This register only exists if a floating-point instruction ("F" or "D" instruction subset) is configured.

The RISC-V architecture stipulates that if single-precision floating-point instructions or double-precision floating-point instructions are supported, a flo
Status register. This register contains the floating-point exception flag bit field (Accrued Exceptions) and floating-point rounding mode
(Rounding Mode) field.

Table 7-8 shows the control bit fields in the fcsr register .

Table 7-8 Control bits of the fcsr register

area		Bit	description
Reserved		31: 8	Unused fields are constant 0
Rounding Mode (frm)		7: 5	Floating-point rounding mode
Accrued Exceptions	NV	4	Illegal operation (Invalid Operation) exception flag
	DZ	3	Divide by Zero exception flag
	OF	2	Overflow exception flag

Page 96

(**Fflags**)	UF	1	Underflow exception flag
	NX	0	Inexact exception flag

7.4.29. Fflags

This register only exists if a floating-point instruction ("F" or "D" instruction subset) is configured.

The fflags register is the field of the floating-point exception flags (Accrued Exceptions) in the floating-point control status register (fcsr). Alias. The reason why a fflags register is defined separately is to facilitate the use of CSR instructions to directly read and write floating-point exception flags Chi bit domain.

As shown in Table 7-8 , the fcsr register contains floating-point exception flag fields (fflags). The exceptions represented by different exception flag bits Types of. If the floating-point arithmetic unit has a corresponding exception in the operation, the corresponding exception flag bit in the fcsr register will be s Set to high and will always accumulate. Software can clear a certain exception flag individually by writing 0.

Note: In many processor architectures, a floating-point operation results in an exception and will trigger an exception trap (Trap) to enter the exception. Normal mode. However, the floating-point instructions of the RISC-V architecture do not jump into the exception mode when a result exception is generated The description only sets the exception flag bit in the fcsr register.

7.4.30. Frm

This register only exists if a floating-point instruction ("F" or "D" instruction subset) is configured.

The frm register is an alias for the floating-point rounding mode field in the floating-point control status register. The reason The unique definition of a frm register is to facilitate the use of CSR instructions to directly read and write floating-point rounding mode separately.

According to the IEEE-754 standard, floating-point operations require a rounding mode, and RISC-V architecture floating-point The rounding mode of the operation can be specified in two ways.

- Static rounding mode: There are 3 bits in the encoding of floating-point instructions as the rounding mode field. The floating-point instruction list and For the encoding, see "Instruction Set Document Version 2.2" (riscv-spec-v2.2.pdf). RISC-V architecture supports such as

- Table 7-9 shows five legal rounding modes.

  - If the rounding mode code is 101 or 110, it is an illegal mode.

  - If the rounding mode is coded as 111, it means that the dynamic rounding mode is used.
- dynamic rounding mode: If a dynamic rounding mode is rounding mode field fcsr register. As shown in Table 7-8 As shown, the fcsr register contains the rounding mode field. The encoding of different rounding modes is the same

- As shown in Table 7-9 , only legal rounding modes are supported. If the rounding mode field in the fcsr register is specified as illegal Rounding mode, subsequent floating-point instructions will generate illegal instruction exceptions.

Table 7-9 Rounding mode bits

Rounding mode encoding	Rounding mode (**a Rounding Mode**)	description
000	RNE	Round to nearest, round to nearest, ties to even)
001	RTZ	Round towards zero
010	RDN	Round down
011	RUP	Round up
100	RMM	Round to the nearest

	nearest, tiest to max magnitude)
101	Illegal value
110	Illegal value
111	Dynamic rounding mode

7.4.31. Cycle and cycleh

cycle and cycleh are read-only copies of mcycle and mcycleh, respectively. Whether this register is readable in User Mode Controlled by the CY bit field of the CSR register mcounteren, see Section 7.4. 34 for details.

7.4.32. Instret and instreth

instret and instreth are read-only copies of minstret and minstreth, respectively. Whether this register is in User Mode IR readable bit field mcounteren controlled by the CSR register, see section 7.4. Section 34 learn more about them.

7.4.33. Time and timeh

time and timeh are read-only copies of mtime and mtimeh, respectively. Whether this register is readable in User Mode is determined by

The TM bit field of the CSR register mcounteren is controlled, see Section 7.4. 34 for details.

7.4.34. Mcounteren

This register is only present in configurations that support User Mode. The control bit fields in the mcounteren register are shown in the table 7-10 .

Table 7-10 Control bits of mcounteren register

area	Bit	description
CY	0	This bit controls whether the cycle and cycle registers can be accessed in User Mode: ■ If this bit is 1, 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. This bit resets to the default value of 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 to the default value of 0
IR	2	This bit controls whether the instret and instreth registers can be accessed in User Mode: ■ If this bit is 1, normal access to instret and instreth in User Mode. ■ If this is 0, accessing instret and instreth in User Mode will trigger illegal instruction exception. This bit resets to the default value of 0
Reserved	3 ~ 31	Other unused fields are constant 0

7.4.35. Pmpcfg <x> register

The pmpcfg <x> register is used to specify the permission configuration register for PMP entries. Each pmpcfg <x> register can be managed. Four PMP table entries, such as pmpcfg0 contains four different domains (pmp0cfg ~ pmp4cfg), and manage entries 0

Go to entry 4. For more information about pmpcfg <x> 8 registers see .2.2 section.

7.4.36. Pmpaddr <x> register

The pmpaddr <x> register is used to specify the address configuration register of each PMP entry. About the pmpaddr <x> register

8. For more information see 2.3 Day.

7.5. N200 Series Core Custom CSR

This section describes the CSR registers defined by the N200 series processor cores.

7.5.1. Mcountinhibit

The mcountinhibit register is used to control the count of mcycle and minstret. Each control bit field is shown in Table 7-11 .

Table 7-11 Control bits of mcountinhibit register

area	Bit	description
Reserved	31: 3	Unused fields are constant 0
IR	2	Minstret count is off when IR is 1
Reserved	1	Unused fields are 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's program, once an NMI occurs, the current program flow is terminated and the processor is forcibly jumped To a new PC address, the PC address jumped into the N200 series processor core after entering the NMI is indicated by the mnvec register set.

Note: The value mnvec by mmisc_ctl controlled more details refer to 7 .5.4 section.

7.5.3. Msubm

The N200 series kernel custom msubm register is used to save the trap type before and after entering the trap.

Table 7-12 shows the control bit fields in the msubm register .

Table 7-12 Control bits of msubm register

area	Bit	description
Reserved	31:10	Unused fields are constant 0
PTYP	9: 8	Save the Trap type before entering Trap: ■ 0: Non-Trap state ■ 1: Interrupt ■ 2: Abnormal ■ 3: NMI
TYP	7: 6	Indicate Core's current trap type: ■ 0: Non-Trap state ■ 1: Interrupt ■ 2: Abnormal ■ 3: NMI
Reserved	5: 0	Unused fields are constant 0

7.5.4. Mmisc_ctl

N200 series kernel custom mmisc_ctl register is used to control mcause value of mnvec and NMI.

Each control bit field in the mmisc_ctl register is shown in Table 7-13 .

Table 7-13 mmisc_ctl register control bits

area	Bit	description
Reserved	31:10	Unused fields are constant 0
NMI_CAUSE_FFF	9	Mcause.EXCCODE that controls mnvec and NMI: ■ 0: The value of mnvec is equal to the PC after the processor reset, the NMI mcause.EXCCODE is 0x1 ■ 1: The value of mnvec is the same as mtvec, mcause.EXCCODE of NMI 0xfff
Reserved	8: 0	Unused fields are constant 0

7.5.5. Msavestatus

msavestatus is used to store the values of mstatus and msubm to ensure the status of each domain of mstatus and msubm
Will not be washed away by NMI or anomalies. msavestatus has a two-level stack that supports up to three levels of exception / NMI state saving.
For more two-level NMI / Exception Status Stack, see Section 4.6 .

The control bits of the msavestatus register are shown in Table 7-14 .

Table 7-14 msavestatus register control bits

area	Bit	description
Reserved	31:16	Unused fields are constant 0
PTYP2	15:14	Trap type before second level nested NMI / exception
Reserved	13:11	Unused fields are constant 0
MPP2	10: 9	Second level nested NMI / Privilege mode before exception
MPIE2	8	Interrupt enable status before second level nested NMI / exception
	7: 6	First level of nested NMI / trap type before exception

PTYP1	5: 3	Unused fields are constant 0
Reserved	2: 1	First level nested NMI / Privilege mode before exception
MPP1	0	Interrupt enable status before the first level of nested NMI / exception
MPIE1		

7.5.6. Msaveepc1 and msaveepc2

msaveepc1 and msaveepc2 are used as the first-level NMI / abnormal state stack and second-level NMI / exception state stack, respectively. To store the PC before the first-level nested NMI / exception and the PC before the second-level nested NMI / exception.

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

When the mret instruction is executed and mcause.INTERRUPT is 0 (such as NMI, or exception), msaveepc1 and msaveepc2 recovers the processor's PC through the one-level and two-level NMI / abnormal state stacks, respectively.

- msaveepc2 => msaveepc1 => mepc => PC

7.5.7. Msavecause1 and msavecause2

msavecause1 and msavecause2 serve as the primary NMI / abnormal state stack and secondary NMI / abnormal state heap, respectively The stack is used to store the mcause before the first level of nested NMI / exception, and the mcause before the second level of nested NMI / exception

mcause.

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

When the mret instruction is executed and mcause.INTERRUPT is 0 (for example, NMI, or exception), msavecause1 And msavecause2 restore mcause state through the one-level and two-level NMI / exception status stacks, respectively.

- msavecause2 => msavecause1 => mcause

7.5.8. Pushmsubm

The N200 series processor defines the CSR instruction implemented by the pushmsubm register csrrwi operation. The value of msubm is the memory space with the stack pointer as the base address.

Take the following instruction as an example to introduce this CSR instruction:

```
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.

Table 7-15 shows the control bit fields in the mtvt2 register .

Table 7-15 Control bits of the mtvt2 register

area	Bit	description
CMMON-CODE-ENTRY	31: 2	When mtvt2.MTVT2EN = 1, this field determines the ECLIC non-vector mode interrupt common-code entry address.
Reserved	1	Unused fields are constant 0
MTVT2EN	0	mtvt2 enable bit: ■ 0: ECLIC non-vector mode interrupt common-code entry address is mtvec Decide ■ 1: ECLIC non-vector mode interrupt common-code entry address is mtvt2.COMMON-CODE-ENTRY decided

Page 103

7.5.10. Jalmnxti

N200 series processors define the jalmnxti register to reduce interrupt latency and speed up interrupt bite.

jalmnxti In addition to enabling the enable interrupt of mnxti, processing the next interrupt, returning the entry address of the next interrupt, etc. In addition to the functions, there is a function to jump to the interrupt handler, so the number of instructions for interrupt processing can be shortened to redu Delay, accelerate the purpose of interrupting tail biting. For more details, see about jalmnxti 5.13 .1.3 section.

7.5.11. Pushmcause

The N200 series processor defines the CSR instruction implemented by the pushmcause register csrrwi operation.
The value of mcause into the memory space of the stack pointer as the base address.

Take the following instruction as an example to introduce this CSR instruction:

```
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 N200 series processor defines the CSR instruction implemented by the pushmepc register csrrwi operation.
The value of mepc goes to the memory space with the stack pointer as the base address.

Take the following instruction as an example to introduce this CSR instruction:

```
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

N200 series customized sleep mode for a CSR register sleepvalue various controls, see page 9 .1 Section

Page 104

Learn more. The control bit fields in the sleepvalue register are shown in Table 7-16 .

Table 7-16 Control bits of the sleepvalue register

area	Bit	description
SLEEPVALUE	0	Controlling WFI's sleep mode ■ 0: Light sleep mode (after WFI, the main working clock of the processor core core_clk is turned off) ■ 1: Deep sleep mode (after WFI, the main working clock of the processor core core_clk and the normally open clock core_aon_clk of the processor core are turned off) This bit resets to the default value of 0
Reserved	31: 1	Unused fields are constant 0

7.5.14.txevt

The N200 series processor core defines a CSR register txevt, which is used to send events externally.

Table 7-17 shows the control bit fields in the txevt register .

Table 7-17 txevt register control bits

area	Bit	description
TXEVT	0	Control Send Event: ■ Writing a 1 to this bit will trigger an output signal from the N200 series processor core Tx_evt generates a single-cycle pulse signal as an external Event signal number. ■ This bit is a self-clear bit, that is, the next week after writing 1 to this bit It is self-clearing to 0. ■ Writing 0 to this bit has no response or operation. This bit resets to the default value of 0
Reserved	31: 1	Unused fields are constant 0

7.5.15. Wfe

The N200 series processor core has a customized CSR register, wfe, which is used to control the wake-up condition of WFI instructions. Interrupts still use Events. See Section 9.2.3 for more details.

Table 7-18 shows the control bit fields in the wfe register .

Table 7-18 Control bits of wfe register

area	Bit	description
WFE	0	Controls whether WFI instruction wake-up conditions use interrupts or events. ■ 0: When the processor core enters sleep mode, it can be woken up by interrupts and NMI. ■ 1: When the processor core enters sleep mode, it can be called by Event and NMI wake. This bit resets to the default value of 0.
Reserved	31: 1	Unused fields are constant 0

8. N200 Series Core Physical Memory Protection Mechanism

8.1. Introduction to PMP

Since the N200 series processor core is a low-power core for the microcontroller field, it does not support the virtual address management unit (MMU), so all address access operations use physical addresses. For different physical address ranges and different Privilege Mode for privilege isolation and protection, N200 series processor core (configurable) supports PMP (Physical Memory Protection) unit.

Note: N200 is a series of processor cores, specifically whether each processor core (such as N201, N205, etc.) Support for PMP may be slightly different, please refer to Nuclei_N200 Series Concise Data Sheet for details.

The relevant points of PMP are as follows:

- PMP supports several entries (Entries), each entry sets the permissions of a certain address range (whether read, write, carried out). The permission setting of each entry of the PMP needs to be configured through the CSR register. The details of the relevant CSR reg Solution See 8 .2 festival.
  - Note: The number of entries supported by the PMP of the N200 series kernel can be configured, please refer to the Nuclei_N200 series List concise data sheets for configuration details.
- PMP can support the minimum 4-byte (32bits) address range for permission setting. Related CSR Registers The Detailed See 8 .2 festival.

8.2. CSR register of PMP

8.2.1. CSR Register List for PMP

The list of CSR registers supported by PMP of Nuclei N200 series core is shown in Table 8-1 .

Table 8-1 CSR register list of PMP

Page 107

Types of	**CSR** address	Read and write attributes	name	Full name
PMP related **CSR** register	0x3A0	MRW	pmpcfg0	PMP entry authority configuration register
	0x3B0	MRW	pmpaddr0	Address configuration register for PMP entry 0
	0x3B1	MRW	pmpaddr1	Address configuration register for PMP entry 1
	0x3B2	MRW	pmpaddr2	Address configuration register for PMP entry 2
	0x3B3	MRW	pmpaddr3	Address configuration register for PMP entry 3
	...	MRW	...	...
	(0x3B0 + n)	MRW	pmpaddrn	Address configuration register for PMP entry n

note:

- In the N200 series processor core, if User Mode is not configured, the corresponding PMP feature and supporting CSR will not be available.

  Memory.

- MRW means it can be read and written only in Machine Mode.

8.2.2. Pmpcfg <x> Register

The pmpcfg <x> register is used to specify the permission configuration register for PMP entries. Each pmpcfg <x> register can be managed. Four PMP table entries, such as pmpcfg0 contains four different domains (pmp0cfg ~ pmp4cfg), and manage entries 0 Go to entry 4. The other analogy is shown in Figure 8-1 .

Figure 8-1 Format of the CSR register pmpcfg0

Table 8-2 shows the control bit fields in the permission configuration register pmp <x> cfg of each entry .

Page 108

Table 8-2 Control domains in pmp <x> cfg

area	Bit	description
R	0	This bit controls the data read permissions of this entry: ■ If this bit is 1, it indicates that the address range set by this entry has data read permission. ■ If this is 0, it means that the address range set by this entry does not have data read permission. This bit resets to the default value of 0.
W	1	This bit controls the data writability of this entry: ■ If this bit is 1, it means that the address range set by this entry has the right to write data. ■ If this is 0, it means that the address range set by this entry does not have permission to write data. This bit resets to the default value of 0.
X	2	This bit controls the code executable permissions of the entry: ■ If this bit is 1, it means that the address range set by this entry has the code executable permission. ■ If this is 0, it means that the address range set by this entry does not have the code execution authority. This bit resets to the default value of 0.
A	4: 3	This bit controls the address range mode of this entry. See Table 8-3 for details. This bit resets to the default value of 0.
Reserved	6: 5	Unused fields are constant 0
L	7	This bit controls whether the PMP entry is locked: ■ If this bit is 1, the entry is locked ■ If this is 0, the entry is not locked See Section 8.3 for details on entry locks. This bit resets to the default value of 0.

The "A" bit field (bits 4 ~ 3) in the permission configuration register pmp <x> cfg of each entry is used to control the location of the entry For the address range mode, the meanings of the codes and modes are shown in Table 8-3 .

Table 8-3 "A" bit field in register pmp <x> cfg

coding	Pattern name	Full English name	Chinese explanation
0	OFF	Null region (disabled)	Empty mode, that is, this entry is not configured
1	Note: The N200 series processor core does not support this encoding mode. If you write 1 to this configuration domain, the result obtained on the hardware is still 0.		
2	NA4	Naturally aligned four-byte region (address alignment), see Table 8-5 for details	situation.
3	NAPOT	Naturally aligned power-of-two region (> = 8 bytes)	Power-of-two area (area size is at least 8 words Section and address alignment), see Table 8-5 for details.

8.2.3. Pmpaddr <x> Register

The pmpaddr <x> register is used to specify the address configuration register for each entry in the PMP. For each pmpaddr <x> register The format is shown in Table 8-4 .

Table 8-4 Control domains in pmpaddr <x>

area	Bit	description
ADDR	29: 0	Configure the address and address mode. See Table 8-5 for details.

Reserved	31:30	Unused fields are constant 0

The size of the address range set by each entry in the PMP needs to be set by the corresponding address configuration register pmpaddr <x>. The "ADDR" bit field and the "A" bit field of the privilege configuration register pmp <x> cfg are jointly set, as shown in Table 8-5 .

Table 8-5 Address range sizes set by PMP entries

Entry address register " **ADDR** " by **pmpaddr <x>** Bit field setting value	Entry authority register " **A** " for **pmp <x> cfg** Bit field setting mode	The size of the address range set by the entry
aa… .aaaa	NA4	4-byte area (address aligned), the base address of this area is "Aa ... aaaa00"
aa… .aaa0	NAPOT	"2 to the third power (8)" byte area (address aligned), this area The base address of the domain is "aa ... aaa000"
aa… .aa01	NAPOT	"2 to the 4th power (16)" byte area (address aligned), the The base address of the area is "aa ... aa0000"
aa… .a011	NAPOT	"2 to the 5th power (32)" byte area (address aligned), the The base address of the area is "aa ... a00000"
……	……	……
01… .1111	NAPOT	"2 to the power of 32 (4G)" byte area (address aligned). by The address space of the N200 series processor core is 32 bits, which means The entire 32-bit address space is covered.

Note: The letter a in the above table means any value.

8.3. Locking of PMP entries

Page 110

The "L" bit field (bit 7) of the pmp <x> cfg register is used to control whether the PMP table entry <x> is locked. The points are as follows:

- If the "L" bit field (bit 7) of the pmp <x> cfg register is written as 1, the entry <x> is locked.
- When the entry <x> is locked, the authority configuration register pmp <x> cfg and the address configuration register corresponding to the entry <x> pmpaddr <x> can no longer be rewritten.
- When the entry <x> is locked, the corresponding authority configuration register pmp <x> cfg cannot be rewritten again, so Software cannot unlock by clearing the "L" bit field. The only way to unlock PMP entries is within the entire processor Core reset.
- When the entry <x> is not locked, the permissions and address range protection set by the entry are only applied to the operation in User Mode. Row protection; when the entry <x> is locked, the permissions and address range protection set by the entry also apply to Machine Operation in Mode.

8.4. PMP protection trigger exception

Any memory access operation requires PMP permission detection. For the principles of PMP permission detection, see Section 8.7 .

If the access to the memory violates the authority set by PMP, the processor will trigger an exception, and the type of exception triggered by PMP According to the situation:

- If the exception is caused by instruction fetch operation, it is "Instruction access fault".
- If it is a load instruction (data memory read operation), the exception is "Load access fault".

- If the exception is caused by Store instruction (data memory write operation) or AMO (atomic operation instruction), "Store / AMO access fault".

For more information on "Exceptions", see Chapter 3 .

8.5. Priority of PMP entry match

Since the N200 series processor core supports multiple PMP entries, when an operation matches multiple entries at the same time, then

Page 111

The lower numbered entries have higher priority.

8.6. Cross-boundary case of PMP entry match

The address mode that can be set in the "A" bit field in the pmp <x> cfg of the N200 series processor PMP is a power of two Power (or 4 byte) aligned address range, so non-aligned memory reads and writes may span a PMP The set address range boundary is handled as follows:

- For normal data storage read and write operations (Load, Store instructions):
  - If the N200 series processor core is configured as hardware that does not support address memory read and write operations, Then this "address misalignment" exception will be generated when the access address is not aligned, so there is no chance to generate a PMP Exception.
  - If the N200 series processor core is configured to support address misaligned data memory read and write operations, then When the access addresses are not aligned, the hardware will split the original operation into byte-by-byte access operations. For example, A "half-word" non-aligned operation is split into two "byte" operations. that Well, for each "byte" operation, a PMP check is performed separately. If the PMP setting is violated, Permission, an exception is triggered. Note: This type of exception is an inexact exception.
- If it is AMO data memory read and write operation (AMO instruction):
  - The RISC-V architecture stipulates that the AMO instruction does not support data memory read and write operations with non-aligned addresse This "Address Misalignment" exception is generated when addresses are misaligned, so there is no chance to generate an exception triggered
- If it is an instruction fetch, a 32-bit wide instruction may be located on an unaligned boundary. Since N200 series The column processor core always prefetches instructions in 32-bit alignment, that is, a non-aligned 32-bit instruction prefetch The fetch will be split into two 32-bit aligned read operations, so these two independent read operations (32-bit aligned) will Perform the PMP permission detection separately. As long as any part violates the permissions set by the PMP, an exception will be triggered.

8.7. Principles of PMP permission detection

The principles and main points of PMP permission detection are as follows:

- If the current operation is in Machine Mode:
  - If the current operation does not match any PMP entry, the operation will not trigger an exception.
  - If the current operation matches a PMP entry:
    - First determine whether the value of the mstatus.MPRV field is 1:
      - If the value of the mstatus.MPRV field is 1, and the value of the mstatus.MPP field indicates User Mode, then PMP reads and writes data in the memory using the permission detection rules under User Mode (Load and Store) operations do permission detection, and perform permissions based on the R / W attribute of the entry configuratio Confirm that if permissions are violated, an exception is triggered.
    - Otherwise, continue to determine whether the entry is locked:
      - If the entry is not locked, it means that permission detection is not applicable to Machine Mode. Check permissions for the current operation.
      - If the entry is locked, it means that permission detection is also applicable to Machine Mode. The X / R / W attribute of the item configuration is used to confirm the permission. If the permission is violated, an exception is trig
- If the current operation is in User Mode:
  - If the current operation does not match any PMP entries, the operation must trigger an exception. This means in In User Mode, if the PMP is not configured with appropriate entries, any access operation cannot be successful.
  - If the current operation matches a PMP entry, you need to verify permissions based on the X / R / W attributes configured on the entry. If the permission is violated, an exception is triggered.

9. Introduction of N200 Series Core Low Power Consumption Mechanism

The N200 series core can support sleep mode for lower static power consumption.

9.1. Go to sleep

The N200 series core can enter the sleep state through the WFI instruction. When the processor executes the WFI instruction, it will:

- Immediately stop execution of the current instruction flow;
- Wait for the processor core to complete any outstanding outstanding operations (such as fetching instructions)
  Order 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 the operation on the bus to complete, it will enter
    To exception handling mode without sleeping.
- When all Outstanding Transactions are completed, the processor will safely enter an empty
  Idle state, this idle state can be called "sleep" state.
- When entering sleep mode:
  - The clocks of the main units in the N200 series core will be gated off to save static power consumption;
  - The output signal core_wfi_mode of the N200 series core will be pulled high, indicating that this processor core is executing WFI
    Sleep state after the instruction;
  - The output signal core_sleep_value of the N200 series kernel will output the value of sleepvalue in the CSR register
    (Note: This signal is only effective when the core_wfi_mode signal is high; the core_wfi_mode signal
    Core_sleep_value must be 0 when low). Software can configure CSR register in advance
    sleepvalue to indicate different sleep modes (0 or 1). note:
    - The behavior of the N200 series cores is exactly the same for different sleep modes. This sleep mode is only
      Provides different control for PMU (Power Management Unit) at SoC system level.

9.2. Exit hibernation

The main points of the N200 series core processor exiting the sleep mode are as follows:

- The output signal core_wfi_mode of the N200 series core will be pulled down accordingly.
- The N200 series core processor can be woken up in the following four ways:
  - NMI
  - interrupt
  - Event
  - Debug request

This is described in detail below.

9.2.1. Wake from NMI

NMI can always wake up the processor core. When the processor core detects the rising edge of the input signal nmi, the processor core is called Wake up, enter the NMI service program and start execution.

9.2.2. Interrupt wakeup

Interrupts can also wake up the processor core:

- If the mstatus.MIE domain is configured to 1 (meaning that global interrupts are turned on), then:
  - When ECLIC (by arbitrating an externally requested interrupt) sends an interrupt to the processor core,
    The core wakes up and enters the interrupt service routine to begin execution.
- If the mstatus.MIE domain is configured to 0 (meaning that global interrupts are turned off), then:
  - If the CSR register wfe.WFE field is configured as 0, then:
    - When ECLIC (by arbitrating an externally requested interrupt) sends an interrupt to the processor core, the processor

The kernel wakes up and continues to execute the previously stopped instruction stream (instead of entering the interrupt service routine)

  - If the cfe register wfe.WFE field is set to 1, then wait for the event to wake up, see the description in the next section.

9.2.3. Event wakeup

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

- If the mstatus.MIE field is set to 0 (meaning that global interrupts are disabled) and the CSR register wfe.WFE field is set
  Configured to 1, then:
  - When the processor core detects that the input signal rx_evt (called the Event signal) is high,
    The core wakes up and resumes execution of the previously stopped instruction flow (instead of entering the interrupt service routine).

9.2.4. Debug wakeup

Debug requests can always wake up the processor core. If the debugger is connected, it will also wake up the processor core.
And enter debug mode.

9.3. Wait for Interrupt mechanism

The Wait for Interrupt mechanism refers to putting the processor core into sleep mode and then waiting for the interrupt to wake up the processor core.
After waking up, enter th ıding interrupt processing function.

As described in Sec nd 9.2, the Wait for Interrupt mechanism can be directly
The mstatus.MIE domai red as 1) done.

9.4. Wait for Event mechanism

The Wait for Event mechanism refers to putting the processor core into sleep mode, and then waiting for the Event to wake up the processor core.
Wake up and continue the previously stopped program (instead of entering the interrupt handler).

Page 116

As described in Sec nd 9.2, the Wait for Event mechanism can directly pass the WFI instruction in cooperation with the following instruction Column completion:

Step 1: Configure the mstatus.MIE field to 0 to turn off global interrupts

Step 2: Configure the wfe.WFE domain to 1

Step 3: Call the WFI instruction. After calling this instruction, the processor will enter the sleep mode, and will continue to execute after the event or NMI wakes it up.

Step 4: Restore the wfe.WFE domain to 0

Step 5: Restore the mstatus.MIE domain to its previous value