Copyright Notice

Contact Information

Should you have any problems with the information contained herein or any suggestions, please contact Nuclei System Technology by email support@nucleisys.com, or visit “Nuclei User Center” website http://user.nucleisys.com for supports or online discussion.

Revision History

Rev.	Revision Date	Revised Section	Revised Content
1.1.0	2020/1/20	N/A	1. First version as the full English
2.0.0	2020/4/28	15	1. Add some cfg_info CSR registers 2. Refer to Nuclei_DSP_QuickStart 2.0.0 version 3. Refer to Nuclei_CCM_Mechanism 1.0 version
2.0.1	2020/5/27	15.5.7	1. Change the mcache_ctl[1]/scrathpad mode default value to be 1 after reset
2.0.2	2020/6/29	15, 16	1. Add ECC related CSRs 2. ADD ECC introduction
2.0.3	2020/8/6	8	1. Update TIMER to be compatiable with CLINT mode for UX class to run Linux
2.0.3	2020/11/24	9	1. Update the PLIC registers memory map
2.1.0	2021/1/20	15.4	1. Add marchid and mimpid to indicate the corresponding RTL hardware version, Core ID, databook and ISA Spec version.
2.2.0	2021/2/4	12.2.3	1. Add PMP TOR supported
2.3.0	2021/5/18	17	1. Add Performance Monitor descriptions
2.4.0	2021/8/2	15.4.5	1. Add support hartid to be non-zero value in non-SMP system

Table of Contents

List of Tables

List of Figures

1. Introduction

This document describes the ISA (Instruction Set Architecture) implemented in Nuclei processor core, including the instruction set and privileged architecture features.

Basically, Nuclei processor core are following and compatible to RISC-V standard architecture, but there might be some additions and enhancements to the original standard spec.

To respect the RISC-V standard, this document may not repeat the contents of original RISC-V standard, but will highlight the additions and enhancements of Nuclei defined.

2. Nuclei RISC-V Instruction Set Overview

2.1. RISC-V ISA Modules supported by Nuclei Core

Nuclei processor core follows the RISC-V instruction set standard (riscv-spec-v2.2.pdf), user can easily get the original copy (riscv-spec-v2.2.pdf) from “Nuclei User Center” website http://user.nucleisys.com or from other public channels.

RISC-V is the configurable modular instruction set. Nuclei processor core support the following instruction set modules:

- Rv32E: 32bits architecture, with 16 general purpose registers
- RV32I: 32bits architecture, with 32 general purpose registers
- RV64I: 64bits architecture, with 32 general purpose registers
- M: Integer Multiplication and Division instructions
- C: Compressed Instructions as 16bits Encoding to reduce code size
- A: Atomic Instructions
- F: Single-Precision Floating-Point Instructions
- D: Double-Precision Floating-Point Instructions
- P: Packed-SIMD Instructions

According to the naming rule from RISC-V standard, the above-mentioned instruction set module can be combined, e.g., RV32IMAC, RV32IMFC, RV32IMAFDC, RV32IMAFDCP, etc. RISC-V standard also use the abbreviation G for the IMAFD combination, hence, RV32IMAFDC or RV64IMAFDC can be also abbreviated as RV32GC or RV64GC.

2.2. Nuclei Processor Core Classes: N, NX and UX

To differentiate the IP products with different positions, Nuclei divide the core products into 3 classes:

- Nuclei N class: support RV32I or RV32E, for the 32bits microcontroller applications.
  - Nuclei N100 series core only support RV32E.
    - Note: N100 is an extremely tiny-area core, which is not following

the architectural spec in this document. N100 has its own dedicated architectural spec. Please visit http://www.nucleisys.com/n100.php for more info.

  - Nuclei N200 series core can be configured to support RV32E or RV32I.
  - Nuclei N300, N600, N900 series core only support RV32I.

- Nuclei NX class: support RV64I without MMU, for the 64bits microcontroller applications.
- Nuclei UX class: support RV64I with MMU, for the 64bits Linux capable applications.
  - The MMU can also be turned off by software, in this mode, UX class core can also work as microcontroller, i.e., Nuclei UX class core is downward-compatible to Nuclei NX class core.

3. Nuclei RISC-V Privileged Architecture

3.1. RISC-V Privileged Architecture supported by Nuclei Core

Nuclei processor core follows the RISC-V privileged architecture standard (riscv-privileged-v1.11.pdf), user can easily get the original copy (riscv-privileged-v1.11.pdf) from “Nuclei User Center” website http://user.nucleisys.com or from other public channels.

Basically, Nuclei processor core are following and compatible to RISC-V standard privileged architecture, but there might be some additions and enhancements to the original standard spec.

To respect the RISC-V standard, this document may not repeat the contents of original RISC-V standard, but will highlight the additions and enhancements of Nuclei defined.

3.2. Privilege Modes

Following the RISC-V privileged architecture standard, Nuclei processor core support following Privilege Modes :

- Machine Mode
- Supervisor Mode
- User Mode

Note: According to the RISC-V standard privileged architecture, there is no way for the software to check current privileged mode (e.g., machine mode or user mode).

Please refer to RISC-V standard privileged architecture for more details.

3.3. Debug Mode

Nuclei processor core also support debug mode to support off-chip debugging.

Nuclei processor core follows the RISC-V debug standard (riscv-debug-spec-v.0.13.2.pdf), user can easily get the original copy (riscv-debug-spec-v.0.13.2.pdf) from “Nuclei User Center” website http://user.nucleisys.com or from other public

channels.

3.4. Machine Sub-Mode added by Nuclei

Besides the above-mentioned standard Privilege Modes, Nuclei processor core further defined 4 types of sub-mode, to differentiate the exact machine mode status, called Machine Sub-Mode:

- Normal Machine Mode
  - The processor core will be under this default sub-mode when out of reset.
  - If the processor core does not encounter exception, NMI, interrupt, debug request, or does not switch mode explicitly, then it will remain in this sub-mode.
- Exception Handling Mode
  - The processor core will be under this sub-mode when the core encountered exception trap. Please refer to Chapter 4. for more details.
- NMI Handling Mode
  - The processor core will be under this sub-mode when the core encountered NMI trap. Please refer to Chapter 5. for more details.
- Interrupt Handling Mode
  - The processor core will be under this sub-mode when the core encountered interrupt trap. Please refer to Chapter 6. for more details.

Nuclei defined a CSR register msubm to reflect processor core's current machine sub-mode (msubm.TYP) and previous machine sub-mode (msubm.PTYP). Please refer to Section 15.5.5. for more details of CSR register msubm.

4. Exception Handling in Nuclei processor core

4.1. Exception Overview

Exception is that the processor core suddenly encounters an abnormal situation when executing the program instruction stream, and aborts execution of the current program, and turns to handle the exception instead. The key points are as follows:

- The “abnormal event” which the core encounters is called an exception. An exception is caused by an internal event in the core or an event during the execution of the program, such as a hardware failure, a program failure, or the execution of a special system call instruction. In short, it is a core-internal issue.
- When the exception is taken, the core will enter the exception handler program.

4.2. Exception Masking

According to the RISC-V architecture, exception cannot be masked, which means if the core encounters an exception, it must stop current execution and turns to handle the exception.

4.3. Priority of Exception

It is possible that the core encounters multiple exceptions at the same time, so exceptions also have priority. The priority of the exception is defined in RISC-V standard privileged architecture, please refer to RISC-V standard privileged architecture for more details.

4.4. Entering Exception Handling Mode

Taking an exception, hardware behaviors of the Nuclei processor core are shown in Figure 4 -1. Note that the following operations are done simultaneously in one cycle:

- Stop the execution of the current program, and start from the PC address

defined by the CSR mtvec. Update the CSR registers: mcause, mepc, mtval, and mstatus. Update the Privilege Mode.

- These behaviors are following RISC-V standard privileged architecture specification. This document will not repeat its content, please refer to RISC-V standard privileged architecture specification for more details.

- Update the Machine Sub-Mode of the core.
  - This is unique in Nuclei processor core, and will be detailed in the following section.

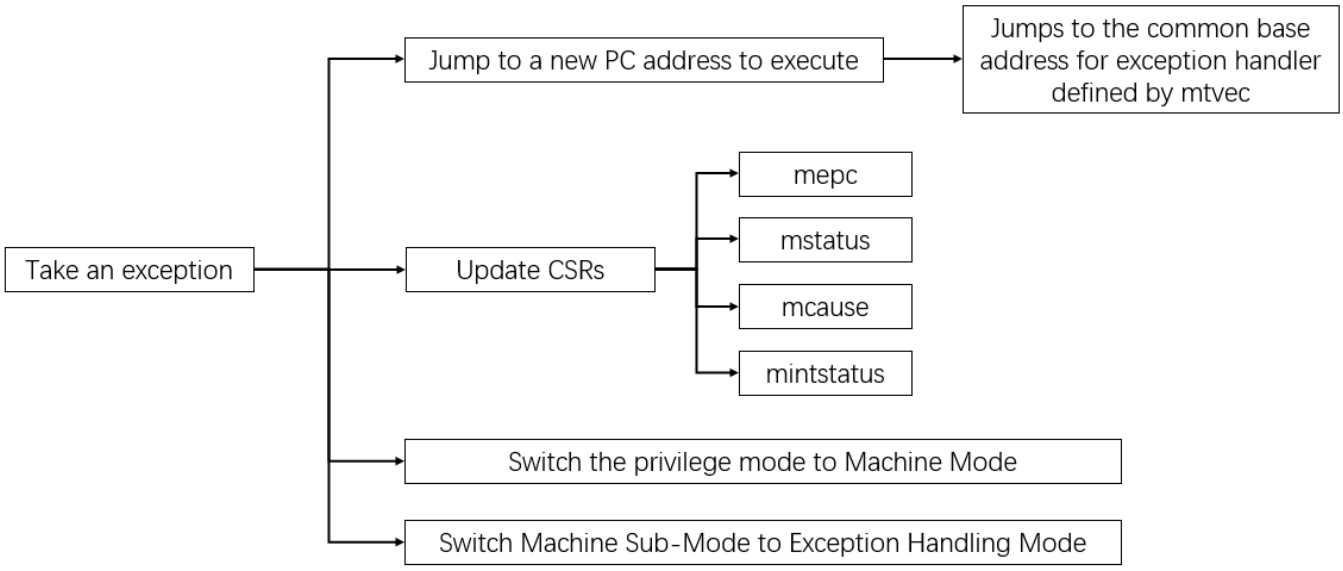


Figure 4-1 The overall process of exception

4.4.1. Update the Machine Sub-Mode

The Machine Sub-Mode of the Nuclei processor core is indicated in the msubm.TYP filed in real time. When the core takes an exception, the Machine Sub-Mode will be updated to exception handling mode, so:

- The filed msubm.TYP is updated to exception handling mode, as shown in Figure 4 -2, to reflect the current Machine Sub-Mode is “Exception Handling Mode”.

- The value of msubm.PTYP will be updated to the value of msub.TYP before taking the exception, as shown in Figure 4 -2. The value of msubm.PTYP will be used to restore the value of msubm.PTYP after exiting the exception handler.

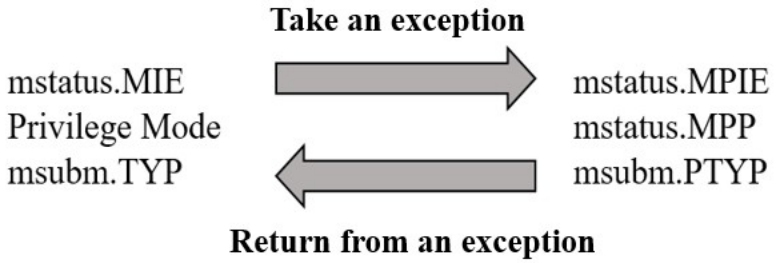


Figure 4-2 The CSR mstatus and msubm updating when enter/exit the exception

4.5. Exit the Exception Handling Mode

After handling the exception, the core needs to exit from the exception handler eventually. Since the exception is handling in Machine Mode, the software has to execute mret to exit the exception handler.

The hardware behavior of the processor after executing mret instruction is as shown in Figure 4 -3. Note that the following hardware behaviors are done simultaneously in one cycle:

- Stop the execution of the current program, and start from the PC address defined by the CSR mepc. Update the CSR mstatus. And update the Privilege Mode.
  - These behaviors are following RISC-V standard privileged architecture specification. This document will not repeat its content, please refer to RISC-V standard privileged architecture specification for more details.
- Update the Machine Sub-Mode of the core.
  - This is unique in Nuclei processor core, and will be detailed in the following section.

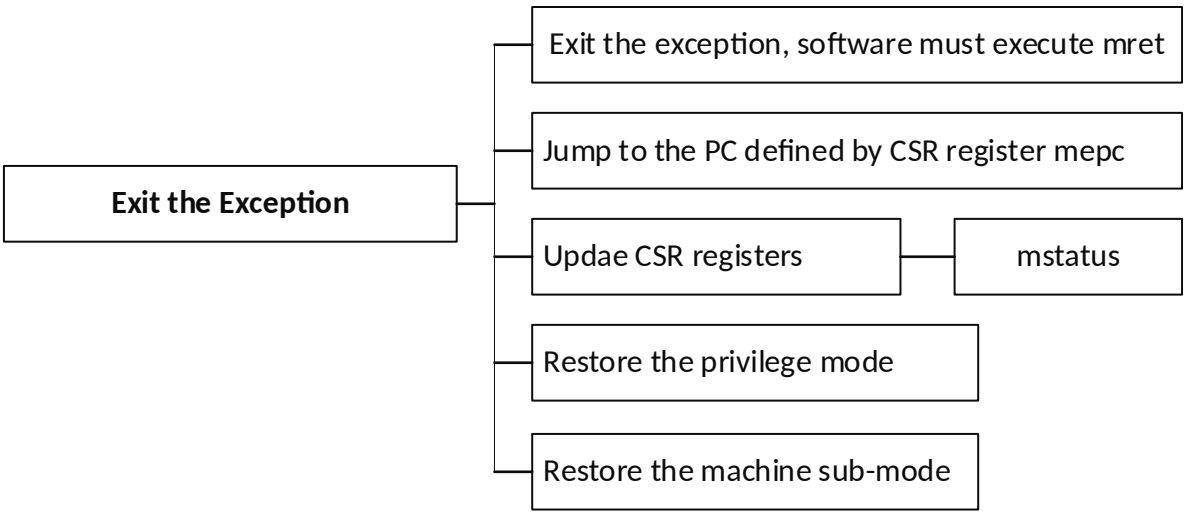


Figure 4-3 The overall process of exiting an exception.

4.5.1. Update the Machine Sub-Mode

The value of msubm.TYP indicates the Machine Sub-Mode of the Nuclei processor core in real time. After executing the mret instruction, the hardware will automatically restore the core's Machine Sub-Mode by the value of msubm.PTYP:

- Taking an exception, the value of msubm.PTYP is updated to the Machine Sub-Mode before taking the exception. After executing the mret instruction, the hardware will automatically restore the Machine Sub-Mode using the value of msubm.PTYP, as shown in Figure 4 -2. Through this mechanism, the Machine Sub-Mode of the core is restored to the same mode before taking the exception.

4.6. Exception Service Routine

When the core takes one exception, it starts to execute the program starting at the address defined by mtvec, and this program is usually an exception service routine. The program can decide to jump further to the specified exception service routine by querying the exception code in the CSR mcause. For example, if the exception code in mcause is 0x2, which indicates that this exception is caused by an illegal instruction, then it can jump to the specific handler for illegal instruction fault.

Note: Since there is no hardware to save and restore the execution context automatically when take or exit an exception, so the software needs to explicitly use the instruction (in assembly language) for context saving and restoring.

4.7. Exception Nesting

Please See Chapter 7. for more details about "Nesting of Interrupt, NMI and Exception".

5. NMI Handling in Nuclei processor core

5.1. NMI Overview

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

5.2. NMI Masking

In the RISC-V architecture, NMI cannot be masked, which means if the core encounters an NMI, it must stop current execution and turns to handle the NMI.

5.3. Entering NMI Handling Mode

Taking an NMI, hardware behaviors of the Nuclei processor core are described as shown in Figure 5 -4. Note that the following operations are done simultaneously in one cycle:

- Update the CSR registers: mepc and mstatus.
  - These behaviors are following RISC-V standard privileged architecture specification. This document will not repeat its content, please refer to RISC-V standard privileged architecture specification for more details.
- Update the CSR registers: mcause.
  - The value of mcause for NMI is unique in Nuclei processor core, and will be detailed in the following section.
- Stop the execution of the current program, and start from the PC address defined by the CSR mnvec.
  - The value of mnvec is unique in Nuclei processor core, and will be detailed in the following section.
- Update the Privilege Mode and Machine Sub-Mode of the core.
  - This is unique in Nuclei processor core, and will be detailed in the following section.

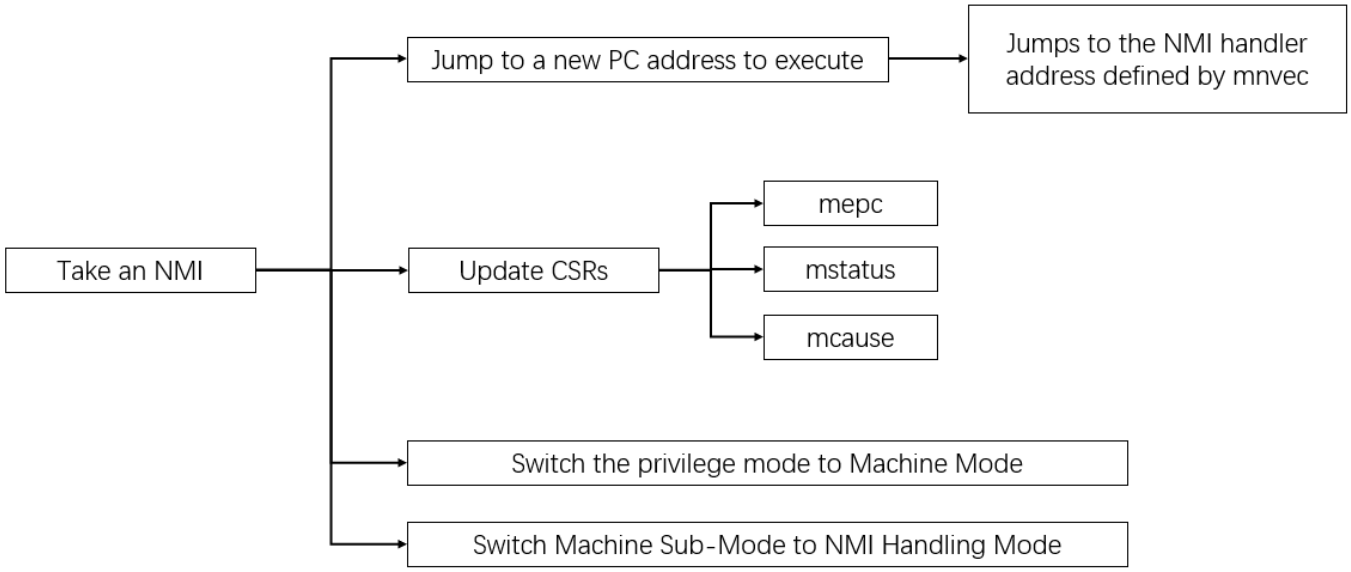


Figure 5-4 The overall process of NMI

5.3.1. Execute from the PC Defined by mnvec

The Nuclei processor core jumps to the PC defined by the CSR mnvec after encountering an NMI. The CSR mnvec has two potential values controlled by CSR register mmisc_ctl:

- When mmisc_ctl[9]=1, the value of mnvec is equal to the value of mtvec, which means NMIs and exceptions share the same trap entry address.
- When mmisc_ctl[9]=0, the value of mnvec equals to the value of reset_vector which is the PC value after a reset. The reset_vector is the core's input signal. Please refer to the specific datasheet of the Nuclei processor core for details about this signal.

5.3.2. Update the CSR mcause

The Nuclei processor core will save the NMI code into the CSR mcause.EXCCODE by the hardware automatically when take a NMI. Interrupts, exceptions and NMIs all have their own specified Trap ID. The Trap ID of NMI has two potential values controlled by CSR register mmisc_ctl:

- When mmisc_ctl[9]=1 , the Trap ID of NMI is 0xfff.
- When mmisc_ctl[9]=0 , the Trap ID of NMI is 0x1.

The software can recognize the Trap reason querying the Trap ID, and build the

corresponding trap handler program for different types of traps.

5.3.3. Update the Privilege Mode

NMI is handed in Machine Mode, so the privilege mode will be switched to Machine Mode when the core takes an NMI.

5.3.4. Update the Machine Sub-Mode

The Machine Sub-Mode of the Nuclei processor core is indicated in the msubm.TYP filed in real time. When the core takes an NMI, the Machine Sub-Mode will be updated to NMI handling mode, so:

- The filed msubm.TYP is updated to NMI handling mode, as described in Figure 5 -5, to reflect the current Machine Sub-Mode is “NMI handling mode”.
- The value of msubm.PTYP will be updated to the value of msub.TYP before taking the NMI, as shown in Figure 5 -5. The value of msubm.PTYP will be used to restore the value of msubm.PTYP after exiting the NMI handler.

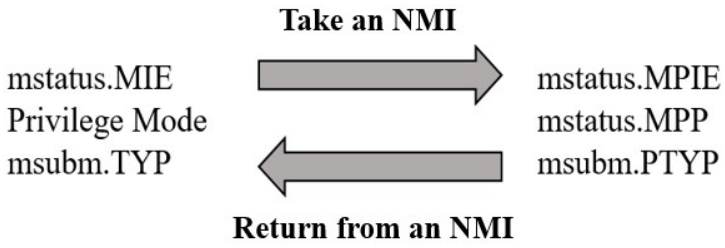


Figure 5-5 The CSR mstatus and msubm updating when enter/exit the NMI

5.4. Exit the NMI Handling Mode

After handling the NMI, the core needs to exit from the NMI handler eventually, and return to execute the main program. Since the NMI is handling in Machine Mode, the software has to execute mret to exit the NMI handler.

The hardware behavior of the processor after executing mret instruction is as shown in Figure 5 -6. Note that the following hardware behaviors are done simultaneously in one cycle:

- Stop the execution of the current program, and start from the PC address defined by the CSR mepc. Update the CSR mstatus.Update the Privilege

Mode.

- These behaviors are following RISC-V standard privileged architecture specification. And the behaviors are exactly same as the behaviors “Exit the Exception Handling Mode”, this document will not repeat its content here, please refer to RISC-V standard privileged architecture specification for more details.

- Update the Machine Sub-Mode.
  - This is unique in Nuclei processor core, and will be detailed in the following section.

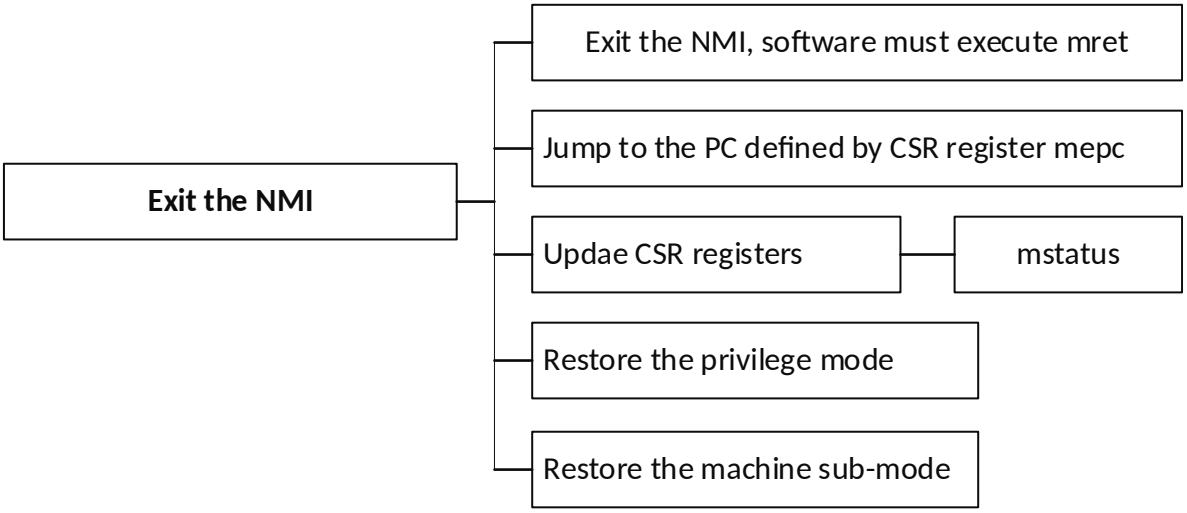


Figure 5-6 The overall process of exiting an NMI.

5.4.1. Update the Machine Sub-Mode

The value of msubm.TYP indicates the Machine Sub-Mode of the Nuclei processor core in real time. After executing the mret instruction, the hardware will automatically restore the core’s Machine Sub-Mode by the value of msubm.PTYP:

- Taking an NMI, the value of msubm.PTYP is updated to the Machine Sub-Mode before taking the NMI. After executing the mret instruction, the hardware will automatically restore the Machine Sub-Mode using the value of msubm.PTYP, as shown in Figure 5 -5. Through this mechanism, the Machine Sub-Mode of the core is restored to the same mode before taking the NMI.

5.5. NMI Service Routine

When the core takes an NMI, it will jump to execute the program at the address

defined by mnvec, which is usually the NMI service routine.

Note: Since there is no hardware to save and restore the execution context automatically when take or exit an NMI, so the software needs to explicitly use the instruction (in assembly language) for context saving and restoring.

5.6. NMI Nesting

Please See Chapter 7. for more details about “Nesting of Interrupt, NMI and Exception”.

6. Interrupt Handling in Nuclei processor core

6.1. Interrupt Overview

Interrupt, that is, the core is suddenly interrupted by other requests during the execution of the current program, and the current program is stopped, and then the core turns to handle other requests. After handling other requests, the core goes back and continues to execute the previous program.

The key points of interrupts are the followings:

- The “other request” interrupts the processor core is called Interrupt Request. The source of this request is called the Interrupt Source. The interrupt s0urce is usually comes from outside the core which is called the External Interrupt Source, but some of the interrupt sources are core-internal, which are called the Internal Interrupt Sources.
- The program used to handle the “other request” is called the Interrupt Service Routine (ISR).
- Interrupt mechanism is a normal mechanism, not an error situation. Once the core receives an interrupt request, it needs to save the context of the current execution status, which is referred as “context saving”. After processing the request, the core needs to restore the previous status, referred to “context restoring”, thereby continuing to execute the previously interrupted program.
- There may be multiple interrupt sources that simultaneously initiate requests to the core, and an arbitration is needed to select one from these sources to determine which interrupt source is prioritized. This scenario is called “interrupt arbitration”, and different interrupts can be assigned with different levels and priorities to facilitate the arbitration, so there is a concept of “interrupt level” and “interrupt priority”.

6.2. CLIC mode and CLINT mode

6.2.1. Setting CLINT or CLIC mode

The Nuclei processor core supports the “CLNT interrupt mode (CLINT mode in short)” and “CLIC interrupt mode (CLIC mode in short)”. Software can set the different mode by writing least significant bits of mtvec, please refer to Section 15.4.6. for more details.

Please refer to following sections for the recommendations of when to set the CLIC mode or CLINT mode.

6.2.2. CLINT mode

CLINT mode is the default mode after reset, it is a simple interrupt handling scheme.

The CLINT mode relying on the PLIC (Platform Level Interrupt Controller) in conjunction with the CSR register mie and mip, which is part of RISC-V standard privileged architecture specification. Please refer to RISC-V standard privileged architecture specification for more details.

The CLINT mode is recommended to be used in Linux capable applications or symmetric multi-processor (SMP) applications, please refer to Chapter 9. for more details.

6.2.3. CLIC mode

CLIC mode is not the default mode after reset, hence need software to explicitly turn it on.

CLIC mode is a relevantly complicate interrupt handling scheme. The CLIC mode relying on the ECLIC (Enhanced Core Local Interrupt Controller), but the CSR register mie and mip are functionally bypassed in this mode.

The CLIC mode is recommended to be used in real-time or microcontroller applications, please refer to Chapter 10. for more details.

6.3. Interrupt Type

The types of interrupts supported by the Nuclei processor core are shown in Figure 6 -7.

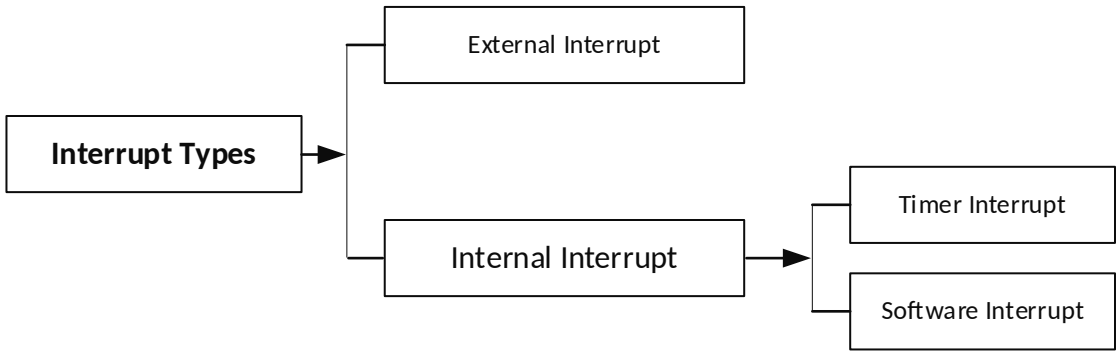


Figure 6-7 Interrupt Types

These will be detailed in the following sections.

6.3.1. External Interrupt

An external interrupt is an interrupt initiated from outside the core. External interrupts allow user to connect to an external interrupt source, such as an interrupt generated by an external device like UART, GPIO and so on.

The Nuclei processor core supports multiple external interrupt sources.

Note:

- In CLINT mode, all of external interrupts are managed by the PLIC, as depicted in Section 11.2..
- In CLIC mode, all of external interrupts are managed by the ECLIC, as depicted in Section 11.3..

6.3.2. Internal Interrupt

The Nuclei processor core has several core-internal private interrupts as the followings:

- Software Interrupt
  - The Nuclei processor core implements a TIMER unit, and an msip register is defined in the TIMER unit, through which software interrupts can be generated. Please see Section 8.2.4. for details.
- Timer Interrupt
  - The Nuclei processor core implements a TIMER unit, and a counter is defined in the TIMER unit, through which time interrupts can be generated. Please see Section 8.2.2. for details.

Note:

- In CLINT mode, the internal interrupts of the Nuclei processor core are managed by CSR register mie and mip, as depicted in Section 15.4.1. and Section 15.4.1..
- In CLIC mode, the internal interrupts of the Nuclei processor core are also managed by the ECLIC, as depicted in Section 11.3..

6.4. Interrupt Masking

6.4.1. Global Interrupt Masking

Interrupts in machine mode can be masked globally by the control bit of CSR mstatus.MIE in Nuclei processor core. Please refer to RISC-V standard privileged architecture specification for more details.

6.4.2. Individual Interrupt Masking

It can also be masked individually for different interrupt sources:

- In CLINT mode:
  - In machine mode, the CSR register mie.MSIE/MTIE can be used to disable software interrupt, timer interrupt individually respectively. The mie.MEIE can be used to disable all the external interrupts managed by PLIC. Please refer to RISC-V standard privileged architecture specification for more details.
  - And PLIC unit also have memory mapped registers to enable/disable each interrupt source managed by PLIC. Please see Section 9.2. for details.
- In CLIC mode, ECLIC have memory mapped register to enable/disable each interrupt source managed by ECLIC. Users can program the corresponding ECLIC register to manage some specified interrupt sources. Please see Section 10.5. for details.

6.5. Interrupt Levels, Priorities and Arbitration

When multiple interrupts are initiated at the same time, the arbitration is required:

- In CLINT mode:
  - The PLIC manages all external interrupts. PLIC assigns its own interrupt priority registers to each external interrupt source. Users can program the PLIC registers to manage the priority of the specified interrupt sources. When multiple interrupts occur simultaneously, the PLIC will select the one that has the highest priority and sent interrupt request to the core (as meip). Please see Section 9.2. for details.
- In CLIC mode:
  - The ECLIC manages all interrupts. ECLIC assigns its own interrupt level

and priority registers to each interrupt source. Users can program the ECLIC registers to manage the level and priority of the specified interrupt sources. When multiple interrupts occur simultaneously, the ECLIC will select the one that has the highest level/priority to be taken. Please see Section 10.5. for more details.

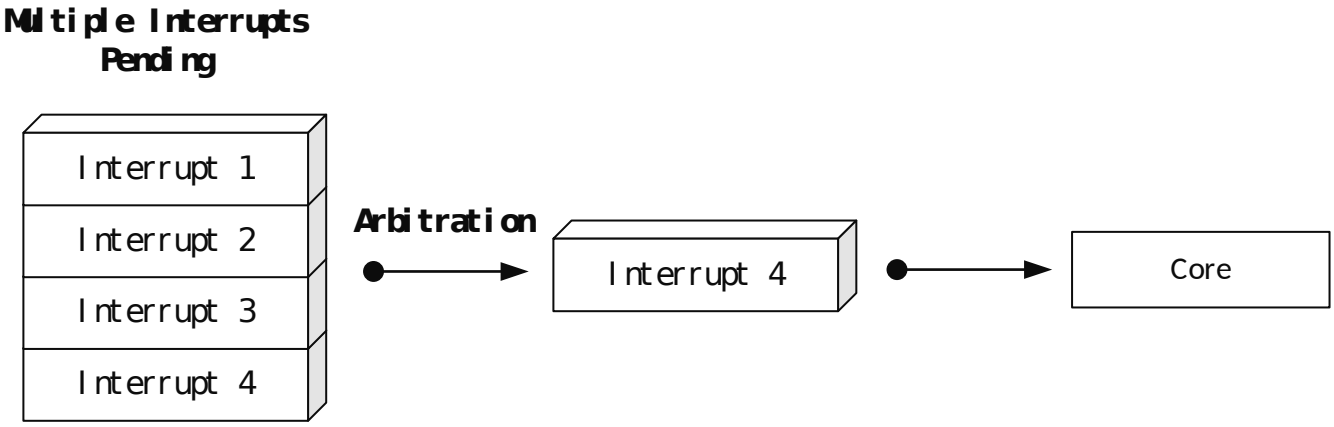


Figure 6-8 Arbitration among Multiple Interrupts

6.6. (CLIC mode) Entering Interrupt Handling Mode

If it is in CLINT mode, taking an interrupt, hardware behaviors of the Nuclei processor core are following RISC-V standard privileged architecture specification. This document will not repeat its content, please refer to RISC-V standard privileged architecture specification for more details.

If it is in CLIC mode, taking an interrupt, hardware behaviors of the Nuclei processor core are described as below. Note that the following operations are done simultaneously in one cycle:

- Stop the execution of the current program, and jump to another PC to execute.
  - Update the following CSR registers: mcause
  - mepc
  - mstatus
  - mintstatus
- Update the Privilege Mode and Machine Sub-Mode of the core.
- The overall process of interrupt is shown in Figure 6 -9.

These will be detailed in the following sections.

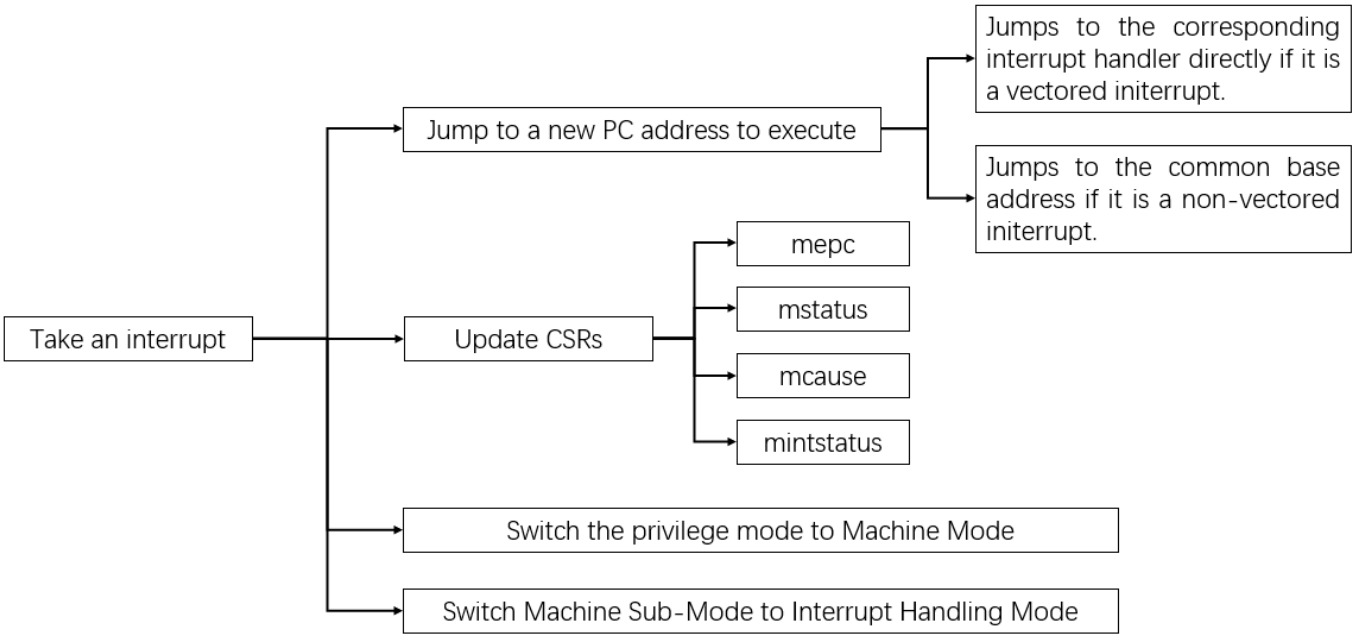


Figure 6-9 The Overall Process of Interrupt for CLIC mode

6.6.1. Execute from a new PC

In CLIC mode, each interrupt source of the ECLIC can be set to vectored or non-vectored interrupt (via the shv filed of the register clicintattr[i]). The key points are as follows:

- If the interrupt is configured as a vectored interrupt, then the core will jump to the corresponding target address of this interrupt in the Vector Table Entry when this interrupt is taken. For details about the Interrupt Vector Table, please refer to Section 6.8.. For details of the vectored processing mode, please refer to Section 6.13.2..
- If the interrupt is configured as a non-vectored interrupt, then the core will jump to a common base address shared by all interrupts. For details of the non-vectored processing mode, please refer to Section 6.13.1..

6.6.2. Update the Privilege Mode

The privilege mode will be switched to Machine Mode when the core takes an Interrupt.

6.6.3. Update the Machine Sub-Mode

The Machine Sub-Mode of the Nuclei processor core is indicated in the msubm.TYP filed in real time. When the core takes an interrupt, the Machine Sub-Mode will be updated to interrupt handling mode, so:

- The value of msubm.PTYP will be updated to the value of msub.TYP before taking the interrupt as shown in Figure 6 -10. The value of msubm.PTYP will be used to restore the value of msubm.PTYP after exiting the interrupt handler.
- The filed msubm.TYP is updated to interrupt handling mode, as described in Figure 6 -10, to reflect the current Machine Sub-Mode is “interrupt handling mode”.

6.6.4. Update the CSR mepc

The return address when the Nuclei processor core exits the interrupt handler is stored in the CSR mepc. When the core takes an interrupt, the hardware will update the CSR mepc automatically, and the value in this CSR will be the return address when exit the interrupt handler. After handling the interrupt, the PC value is restored from this CSR mepc to return to the execution point that was previously stopped.

Note:

- When an interrupt is taken, the CSR mepc is updated to the PC of the instruction that encounters the interrupt. Then after exiting the interrupt, the program will continue to execute from the instruction that encounters the interrupt.
- Although the CSR mepc can be updated automatically encountering an interrupt, it is a both readable and writeable register, so the software can modify it explicitly.

6.6.5. Update the CSRs mcause and mstatus

The Nuclei processor core will update the CSR mcause by the hardware automatically, as described in Figure 6 -10, explained as follows:

- A mechanism is required to record the ID of the interrupt being taken.
  - When an interrupt is taken by the Nuclei processor core, the field mcause.EXCCODE is updated to the ID of the taken interrupt by the

ECLIC, so the software can query the ID of this selected interrupt by reading this register.

- When the current interrupt is taken, a mechanism is required to record the global interrupt enable bit and the Privilege Mode before taking the interrupt.

  - When the Nuclei processor core takes an interrupt, the filed mstatus.MPIE will be updated to the value of mstatus.MIE, and the filed mstatus.MIE will be set to 0, which means interrupts are globally masked, and all interrupts will not be taken.
  - When the Nuclei processor core takes an interrupt, the Privilege Mode of the core will be switched to Machine Mode, and the field mstatus.MPP will be set to the Privilege Mode before taking the interrupt.

- When the current interrupt is taken, possibly it is preempting the interrupt who was previously being processed (whose interrupt level is relatively lower, so it can be preempted), and a mechanism is needed to record the interrupt level of the preempted interrupt.

  - When an interrupt is taken by the Nuclei processor core, the field mcause.MPIL is updated to the value of minstatus.MIL. The value of mcause.MPIL is used to restore the value of mcause.MIL after handling the interrupt.

- If the taken interrupt is a vectored interrupt, the core will jump to the corresponding target address stored in the Vector Table Entry. For a detailed description of the vectored interrupt processing mode, please see Section 5.13.2. In terms of the hardware implementation, the processing of an interrupt needs to be divided into two steps. The first step is to query the target address from the Vector Table, and then jump to the target address in the second step. Then, it is possible that a memory access occurs in the first step, querying the target address from the Vector Table, so a mechanism is required to record such a special memory access error.

  - When the Nuclei processor core takes an interrupt, if the interrupt is a vectored mode interrupt, the value of mcause.minhv will be updated to

1, and then cleared to 0 when the above “two-step” operation is completed. Assuming a memory access error occurs midway, it will raise an Instruction Access Fault exception, and the value of mcause.minhv will be 1 assuming this bit is not cleared.

- Note: the mcause.MPIE and mcause.MPP are mirrored with the field of mstatus.MPIE and mstatus.MPP. Which means normally the value of mstatus.MPIE is always the same as the value of mcause.MPIE and the value of mstatus.MPP is the same as the value of mcasue.MPP.

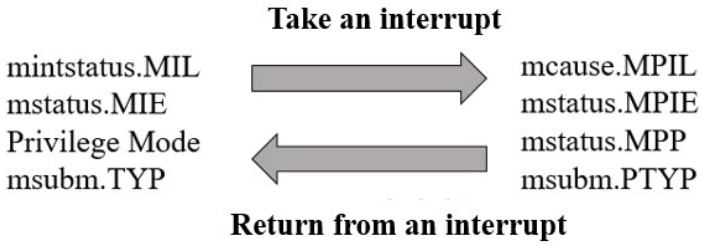


Figure 6-10 The CSR updating when enter/exit the Interrupt

6.7. (CLIC mode) Exit the Interrupt Handling Mode

If it is in CLINT mode, after handling the interrupt, hardware behaviors of the Nuclei processor core are following RISC-V standard privileged architecture specification. This document will not repeat its content, please refer to RISC-V standard privileged architecture specification for more details.

If it is in CLIC mode, after handling the interrupt, the core needs to exit from the interrupt handler eventually, and return to execute the main program. Since the interrupt is handling in Machine Mode, the software has to execute mret to exit the interrupt handler. The hardware behavior of the processor after executing mret instruction is as depicted in Figure 6 -11. Note that the following hardware behaviors are done simultaneously in one cycle:

- Stop the execution of the current program, and start from the PC address defined by the CSR mepc.
- Update the following CSRs:
  - mstatus
  - mcause
  - mintstatus
- Update the Privilege Mode and the Machine Sub-Mode.

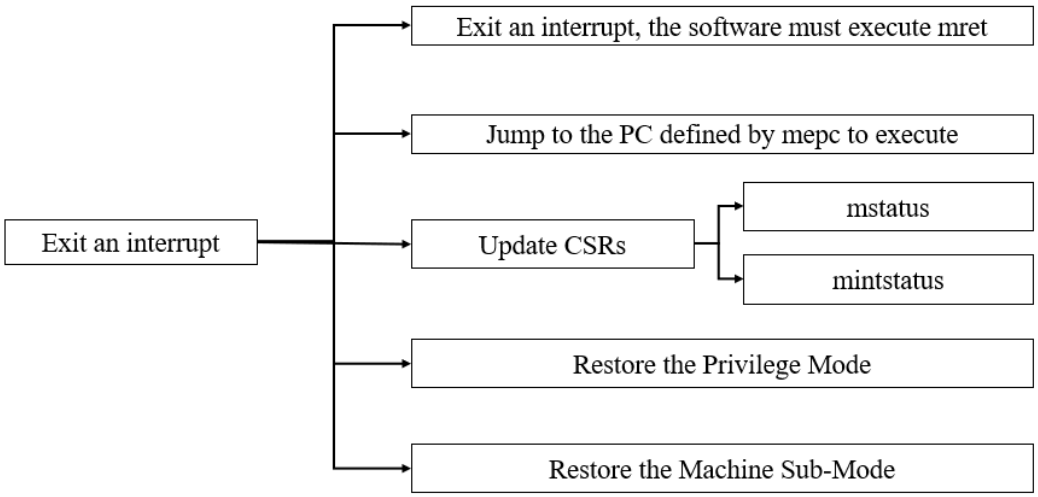


Figure 6-11 The overall process of exiting an interrupt

These will be detailed in the following sections.

6.7.1. Executing from the Address Defined by mepc

When an interrupt is taking, the mepc is updated to the PC value of the instruction encountered the interrupt. Through this mechanism, executing the mret instruction, the core will return to the instruction encountered the interrupt, and continues to execute the program.

6.7.2. Update the CSRs mcause and mstatus

The Nuclei processor core will update the CSR mcause when executes one mret instruction, explained as follows:

- When an interrupt is taken, the value of mcause.MPIL will be updated to the value of mintstatus.MIL before taking the interrupt. The hardware will restore the value of minstatus.MIL using the value of mcause.MPIL when executes the mret instruction to exit the interrupt handler. Through this mechanism, the value of mintstatus.MIL is restored to the previous value before taking the interrupt.
- When an interrupt is taken, the value of mcause.MPIE will be updated to the value of mintstatus.MIE before taking the interrupt. The hardware will restore the value of minstatus.MIE using the value of mcause.MPIE when executes the mret instruction to exit the interrupt handler. Through this mechanism, the value of mintstatus.MIE is restored to the previous value before taking the interrupt.

- When an interrupt is taken, the value of mcause.MPP will be updated to the Privilege Mode before taking the interrupt. The hardware will restore the Privilege Mode using the value of mcause.MPP when executes the mret instruction to exit the interrupt handler. Through this mechanism, the Privilege Mode is restored to the previous value before taking the interrupt.
- Note: the mcause.MPIE and mcause.MPP are mirrored with the field of mstatus.MPIE and mstatus.MPP. Which means normally the value of mstatus.MPIE is always the same as the value of mcause.MPIE and the value of mstatus.MPP is the same as the value of mcasue.MPP.

6.7.3. Update the Privilege Mode

The hardware will update the Privilege Mode using the value of mcause.MPP automatically after the execution of the mret instruction:

- Taking an interrupt, the value of mstatus.MPP was updated to the Privilege Mode of the core before taking the interrupt, and after executing the mret instruction, the value of Privilege Mode is restored by the value of mstatus.MPP. Through this mechanism, the core is guaranteed to return to the Privilege Mode before taking the interrupt.

6.7.4. Update the Machine Sub-Mode

The value of msubm.TYP indicates the Machine Sub-Mode of the Nuclei processor core in real time. After executing the mret instruction, the hardware will automatically restore the core's Machine Sub-Mode by the value of msubm.PTYP:

- Taking an interrupt, the value of msubm.PTYP is updated to the Machine Sub-Mode before taking the interrupt. After executing the mret instruction, the hardware will automatically restore the Machine Sub-Mode using the value of msubm.PTYP. Through this mechanism, the Machine Sub-Mode of the core is restored to the same mode before taking the interrupt.

6.8. (CLIC mode) Interrupt Vector Table

If in CLINT mode, Nuclei processor core does not support the vector mode. Hence, there is no vector table relevant. Herein this section only introduces the CLIC mode interrupt vector table.

If in CLIC mode, as shown in Figure 6 -12, the interrupt vector table is a

contiguous address space in the memory, and each word of this address space is used to store the address of the interrupt service routine corresponding to each interrupt source of the ECLIC.

The base address of the interrupt vector table is defined by the CSR mtvt.

The role of the interrupt vector table is very important. When the core takes an interrupt, no matter a vectored or non-vectored interrupt, the hardware will eventually jump to the corresponding PC of the interrupt service routine by querying the interrupt vector table. Please see Section 6.13. for more details.

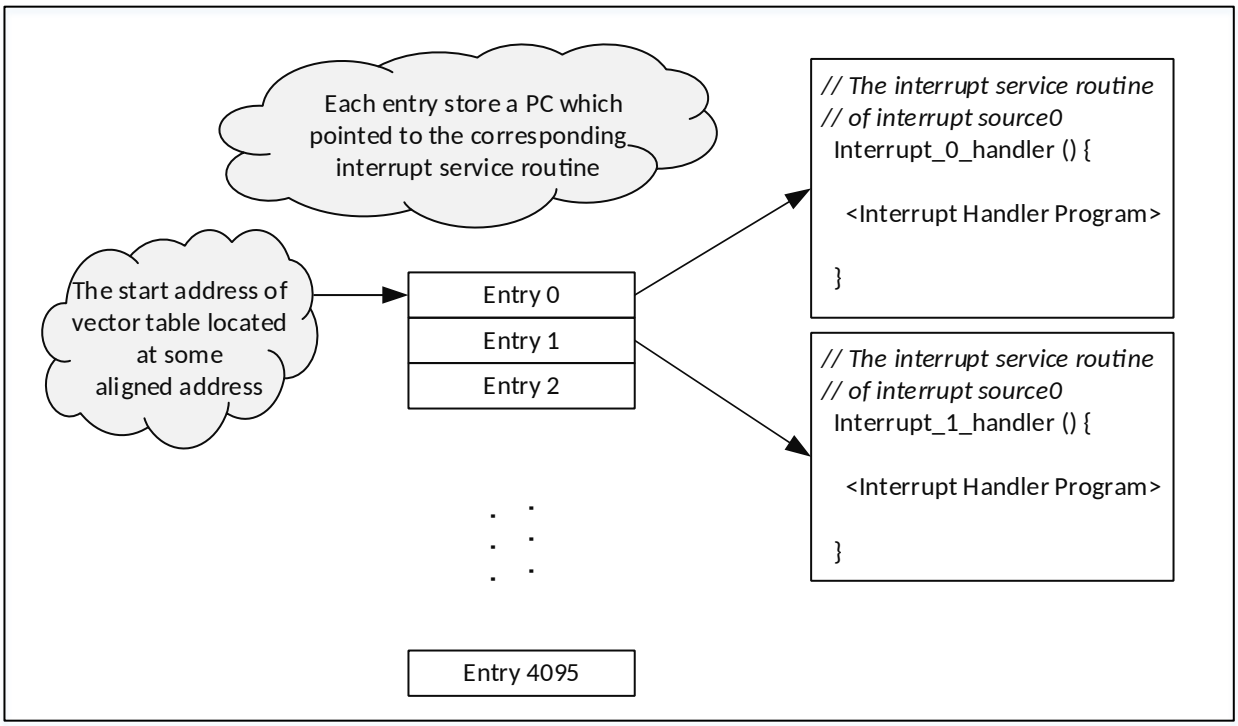


Figure 6-12 Interrupt Vector Table

6.9. Context Saving and Restoring

Nuclei processor core based on the RISC-V architecture do not support the hardware automatic context saving and restoring when take or exit an interrupt. So the software is required to write the instructions (in assembly language) for context saving and restoring.

For CLIC mode, depending on whether the interrupt is a vectored or non-vectored, the context requiring saving and restoring will vary. Please see Section 6.13. for more details.

6.10. Interrupt Response Latency

The concept of interrupt response latency usually refers to the cycle consumed from the time point “external interrupt source asserting” to the time point “the first instruction in the corresponding interrupt service routine of C function is executed”. Therefore, the interrupt latency usually includes the following aspects of the cycle overhead:

- The overhead of jumping to the target PC
- The overhead of context saving
- The overhead of jumping to the Interrupt Service Routine of C function

For CLIC mode, interrupt response latency varies depending on whether the interrupt is a vectored or non-vectored. Please see Section 6.13. for more details.

6.11. (CLIC mode) Interrupt Preemption

If in CLINT mode, Nuclei processor core does not support the interrupt preemption. Herein this section only introduces the CLIC mode interrupt preemption.

If in CLIC mode, while the core is handling an interrupt, there may be another new interrupt request of a higher level, and then the core can stop the current interrupt service routine and start to taken the new one and execute its “Interrupt Service Routine”. Hence, the interrupt preemption is formed (that is, the previous interrupt has not returned yet, and the new interrupt is taken), and there could be multi-level of nesting.

Take the case in Figure 6 -13 as an example:

- Assuming that the core is handling one timer interrupt and suddenly an interrupt is initiated by button 1 and this interrupt has a higher level than the timer interrupt. The core will stop processing the timer interrupt and start to handle the interrupt initiated by button 1.
- Then another interrupt is initiated by button 2, which has a higher level than the interrupt initiated by button 1, so the core will stop processing the interrupt of button 1 and start to handle the interrupt of button 2.
- After that no other higher-level interrupts arrive, the button 2 interrupt will not be preempted, and the core can successfully complete the interrupt service routine of the button 2 interrupt, and then return to process the button 1 interrupt.
- Completing the interrupt service routine of button 1 interrupt, the core will

return to execute the timer interrupt service routine to handle the timer interrupt.

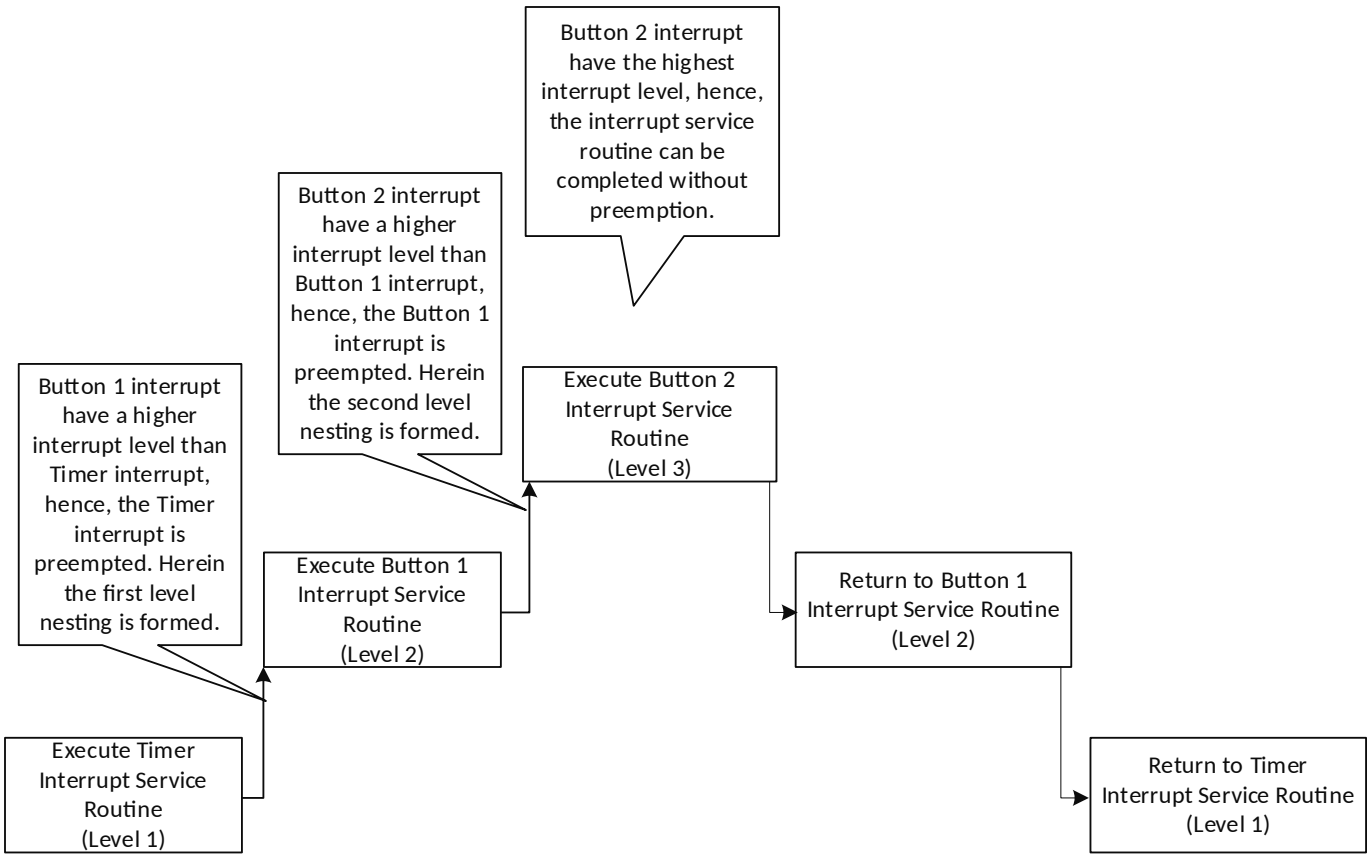


Figure 6-13 Interrupt Preemption

In the Nuclei processor core, the supported methods for interrupt preemption depending on whether the interrupt is a vectored interrupt or a non-vectored interrupt. Please see Section 6.13. for more details.

6.12. (CLIC mode) Interrupt Tail-Chaining

If in CLINT mode, Nuclei processor core does not support the interrupt tail-chaining. Herein this section only introduces the CLIC mode interrupt tail-chaining.

If in CLIC mode, while the core is processing one interrupt, a new interrupt request is initiated, but the level of the new request is not higher than the handling one, so the new interrupt request cannot preempt the handling one.

After handling the current interrupt, theoretically it is necessary to restore the

context, then exit the interrupt service routine, return to the main program, and then take the new interrupt. To take the new interrupt, it is necessary to save the context again. Therefore, there is a back-to-back “context saving” and “context restoring”. The “tail-chaining” can save the cost of this back-to-back “context saving” and “context-restoring”, as shown in the Figure 6 -14.

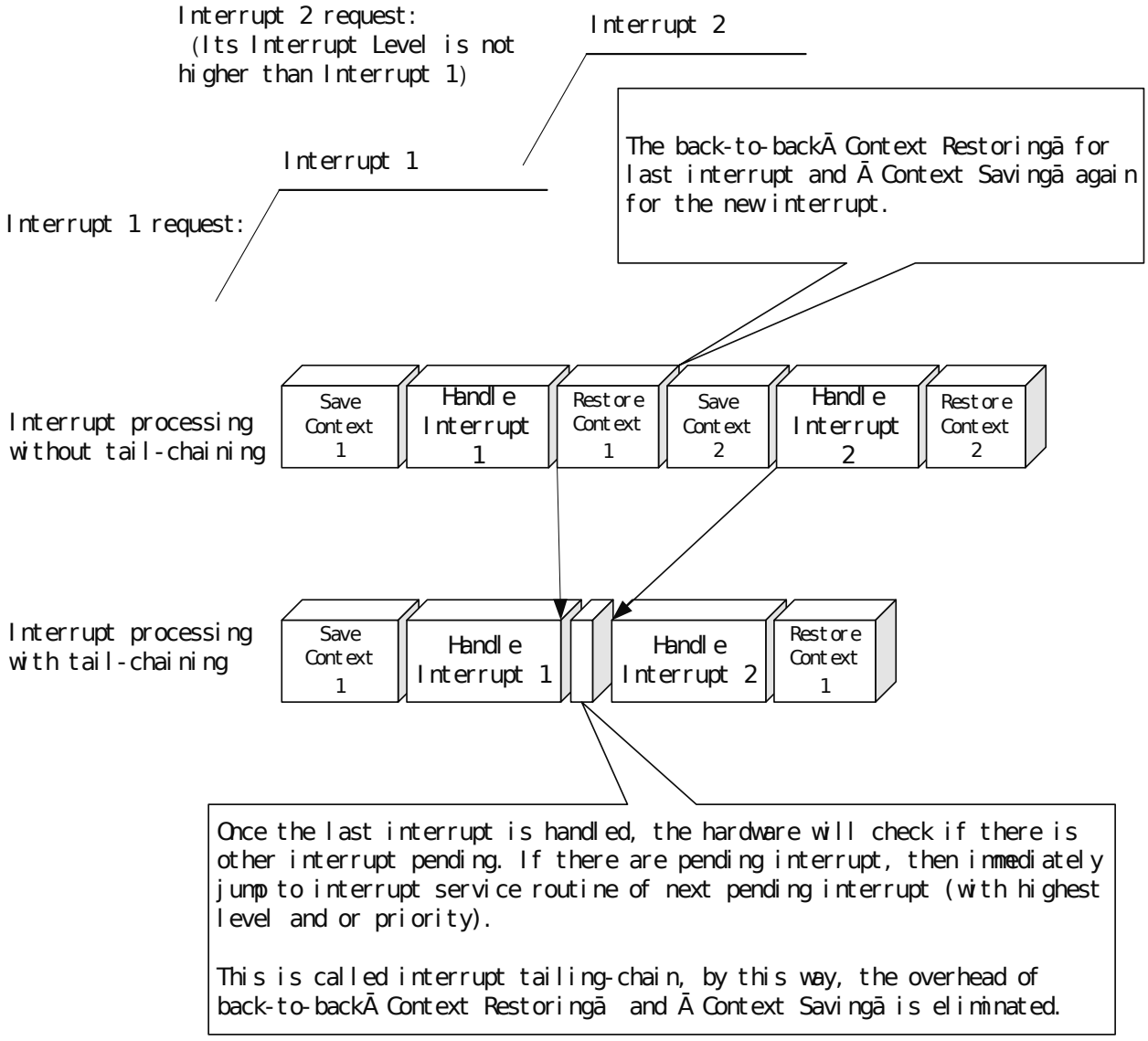


Figure 6-14 Interrupt tail-chaining

As for the Nuclei processor core, only non-vectored interrupts (CLIC mode) support the feature of tail-chaining. Please see Section 6.13.1.3 for more details.

6.13. (CLIC mode) Vectored and Non-Vectored Processing Mode of Interrupts

In CLIC mode, each interrupt source can be configured to vectored or non-vectored processing mode (via the shv field of the ECLIC register clicintattr[i]). There is obvious difference between the vectored and non-vector processing mode, which are described in the following sections.

6.13.1. Non-Vectored Processing Mode

6.13.1.1 Feature and Latency of Non-Vectored Processing Mode

If the interrupt is non-vectored, once it is taken, the core will jump to the common base entry shared by all non-vectored interrupts, and the address of this entry can be set by software:

- If the least significant bit of the CSR mtvt2 is 0 (default value after reset), the common base address shared by all non-vectored interrupts is specified by the CSR mtvec (ignoring the value of the lowest 2 bits). Since the CSR mtvec also indicates the entry address of exceptions, which means exceptions and all non-vector interrupts share the entry address.
- If the least significant bit of the CSR mtvt2 is 1, the common entry address of all non-vectored interrupts is defined by the CSR mtvt2 (ignoring the value of the lowest 2 bits). In order to handle the interrupt as fast as possible, it is recommended to set the least significant bit of the CSR mtvt2 to 1, which means the entry address for all non-vectored interrupts is separated from the entry of exceptions (exception entry is defined by the CSR mtvec).

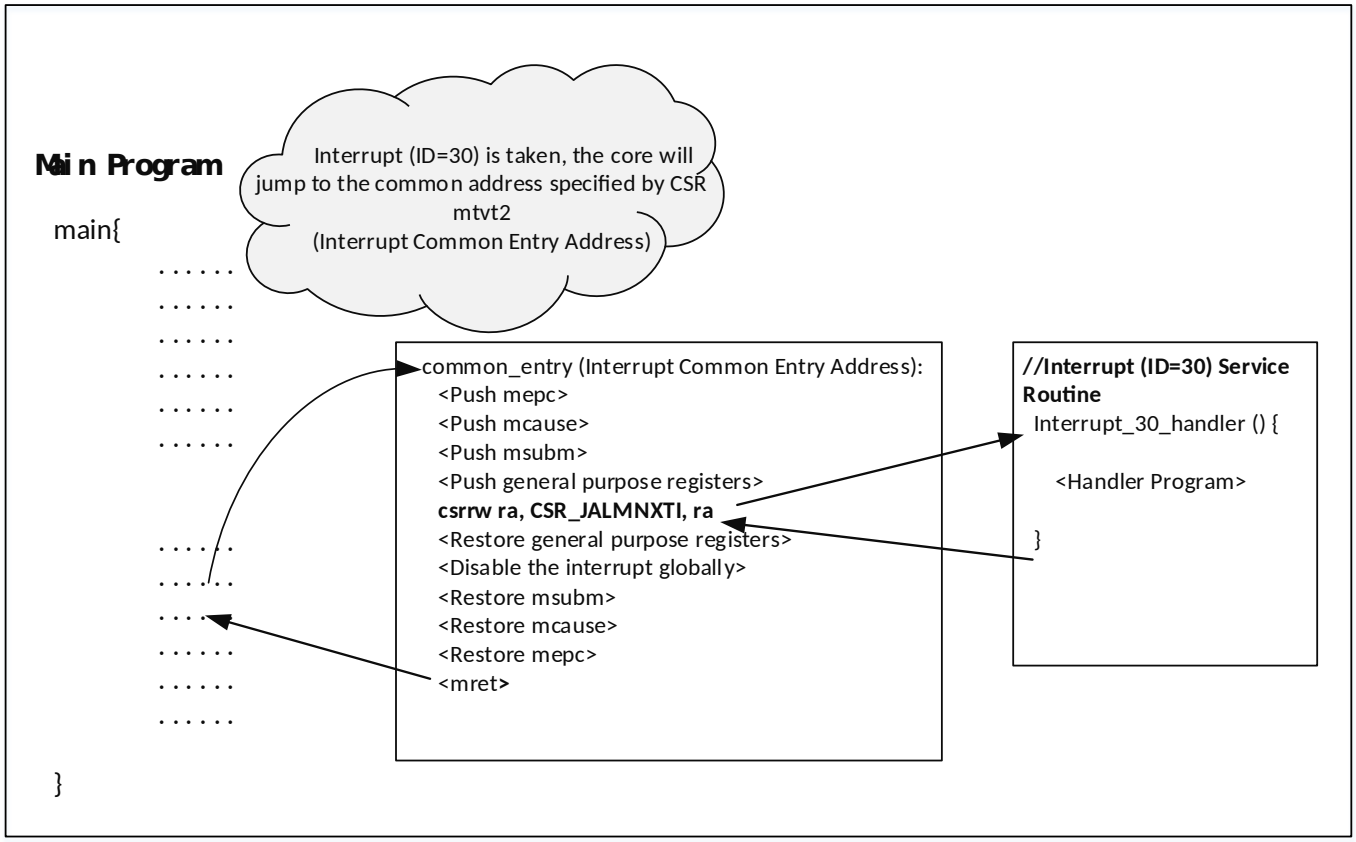
After entering the common base entry of non-vectored interrupts, the core will start to execute a common program, as the example shown in Feature 6 -15, the program is typically as follows:

- Firstly, save the CSR mepc, mcause, msubm into the stack. These CSR registers are saved to ensure that subsequent preempted interruption can be handled correctly, because taken the new preempted interrupt will overwrite the values of mepc, mcause, msubm, so they need to be saved into the stack first.
- Save several general-purpose registers (the execution context) into the stack.
- Then execute a Nuclei self-defined instruction “csrrw ra, CSR_JALMNXTI, ra”. If there is no pending interrupt, then this instruction will be regarded as a Nop. If there is a pending interrupt, the core will take the following

operations:

- Jump to the target address stored in Vector Table Entry and execute the corresponding Interrupt Service Routine.
- The hardware will set the global interrupt enable bit mstatus.MIE while the core jumps to the interrupt service routine. Setting the mstatus.MIE bit, new interrupt will be taken and form an interrupt preemption.
- In addition to jump to the Interrupt Service Routine, the instruction "csrrw ra, CSR_JALMNXTI, ra" also have the effect of a JAL (Jump and Link) instruction. The hardware will update the value of the link register to the PC of this instruction as the return address of the function. Therefore, returning from the interrupt handler, the core will return to the instruction "csrrw ra, CSR_JALMNXTI, ra", and re-judge whether there is still an interrupt pending to implement the operation of the tail-chaining. See more description of tail-chaining from Section 6.13.1.3.

- At the end of the interrupt service routine, the software also needs to add the corresponding context restoring operation. Before restoring the CSR mepc, mcause, msubm, and the global interrupt enable bit mstatus.MIE needs to be cleared again to ensure the atomicity of the recovery operations of mepc, mcause, and msubm.



Feature 6-15 Example for non-vectored interrupt

Since the core needs to execute a common handler before jump to the specified interrupt service routine of the corresponding non-vector interrupt. Therefore, the total cycle overhead from the interrupt initiation to the first instruction in the interrupt service routine (C function) is executed are as below:

- The overhead caused by jumping to the interrupt handler which is about 4 cycles ideally.
- The overhead caused by saving CSRs mepc, mcause, msubm into the stack is about 3 cycles ideally.
- The overhead caused by saving the context. If the architecture is RV32E, then it only takes 8 cycles to save 8 general purpose registers; if it is RV32I architecture, then there are 16 general purpose registers required to be saved.
- The overhead caused by jumping to the Interrupt Service Routine which is about 5 cycles ideally.

6.13.1.2 Preemption of Non-Vectored Interrupt

As mentioned above, non-vectored interrupt processing mode can always support interrupt preemption as the example shown in Figure 6 -16: assuming that the three interrupts 30, 31, 32 come sequentially, and the level of interrupt 32 is greater than the level of interrupt 31 which is greater than the level of interrupt 30. Since then, the subsequent interrupts will preempt interrupts that were previously processed to form interrupt preemptions.

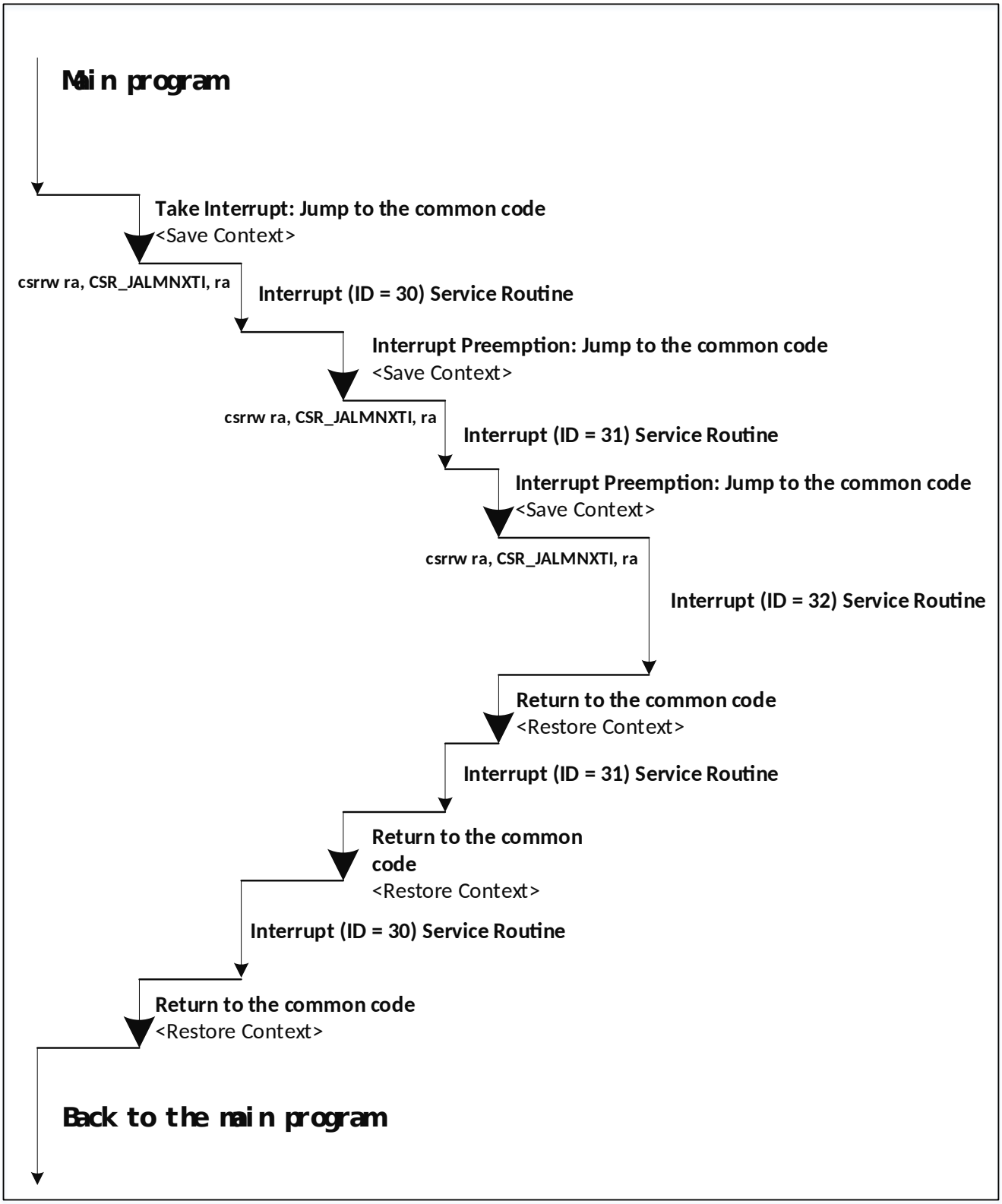


Figure 6-16 Interrupt preemptions caused by three sequential non-vectored interrupts

6.13.1.3 Non-Vectored Interrupt Tail-Chaining

As mentioned in Section 6.12., the tail-chaining can save cycles overhead significantly (reduced a back-to-back context saving and restoring).

For non-vectored interrupts (CLIC mode), as mentioned in Section 6.13.1.1, the instruction “csrrw ra, CSR_JALMNXTI, ra” in the common base handler also achieves the effect of JAL (Jump and Link), which means the hardware will update the value of the Link register to the PC of this instruction as the return address. Therefore, the core will execute the instruction “csrrw ra, CSR_JALMNXTI, ra” again when it return from the interrupt service handler (C function) and re-execute “csrrw ra, CSR_JALMNXTI, ra”, i.e., re-judge if there is a pending interrupt to perform the tail-chaining operation.

As the example shown in Figure 6 -17: assuming the interrupts 30, 29, 28 come successively, and “the level of interrupt 30 ” >= “the level of interrupt 29” >= “the level of interrupt 28”, then the subsequent interrupt will not preempt the interrupt that was taken before, which means no preemption will happen, but all these subsequent interrupt will be marked as “pending”. When the interrupt 30 has been already handled, the core will handle the interrupt 29 directly without the intermediate “context restoring” and “context saving” procedures.

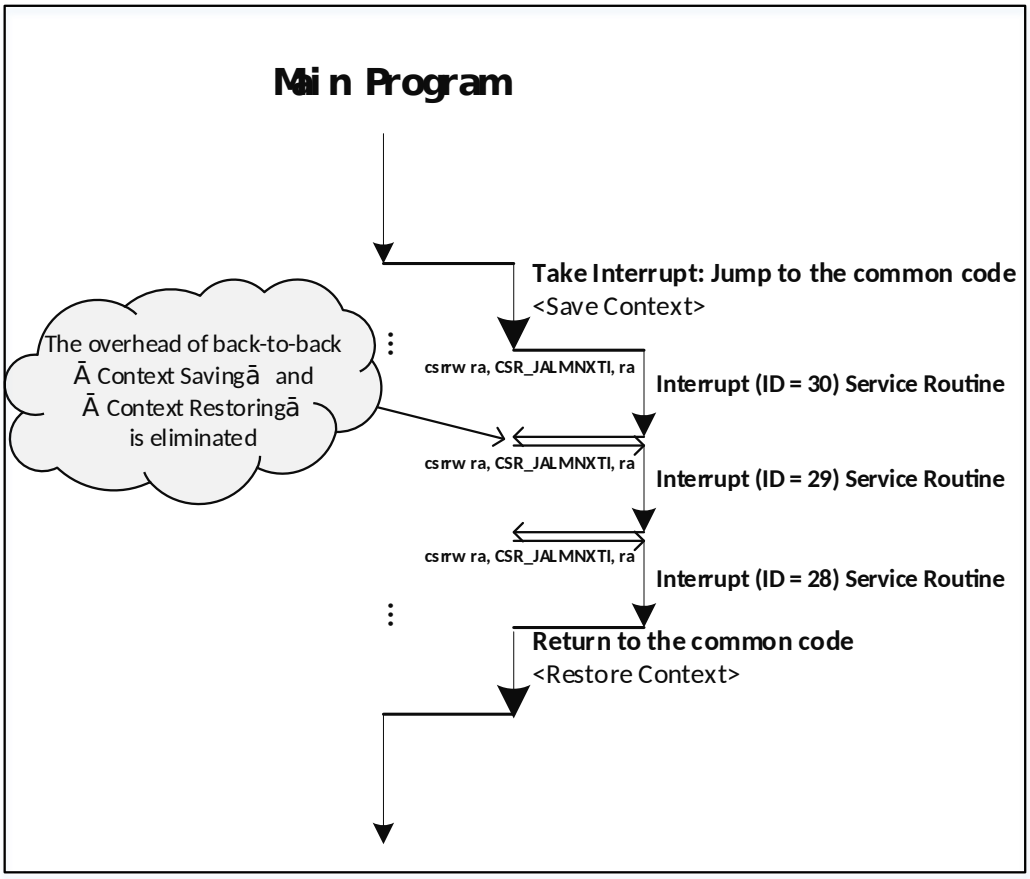
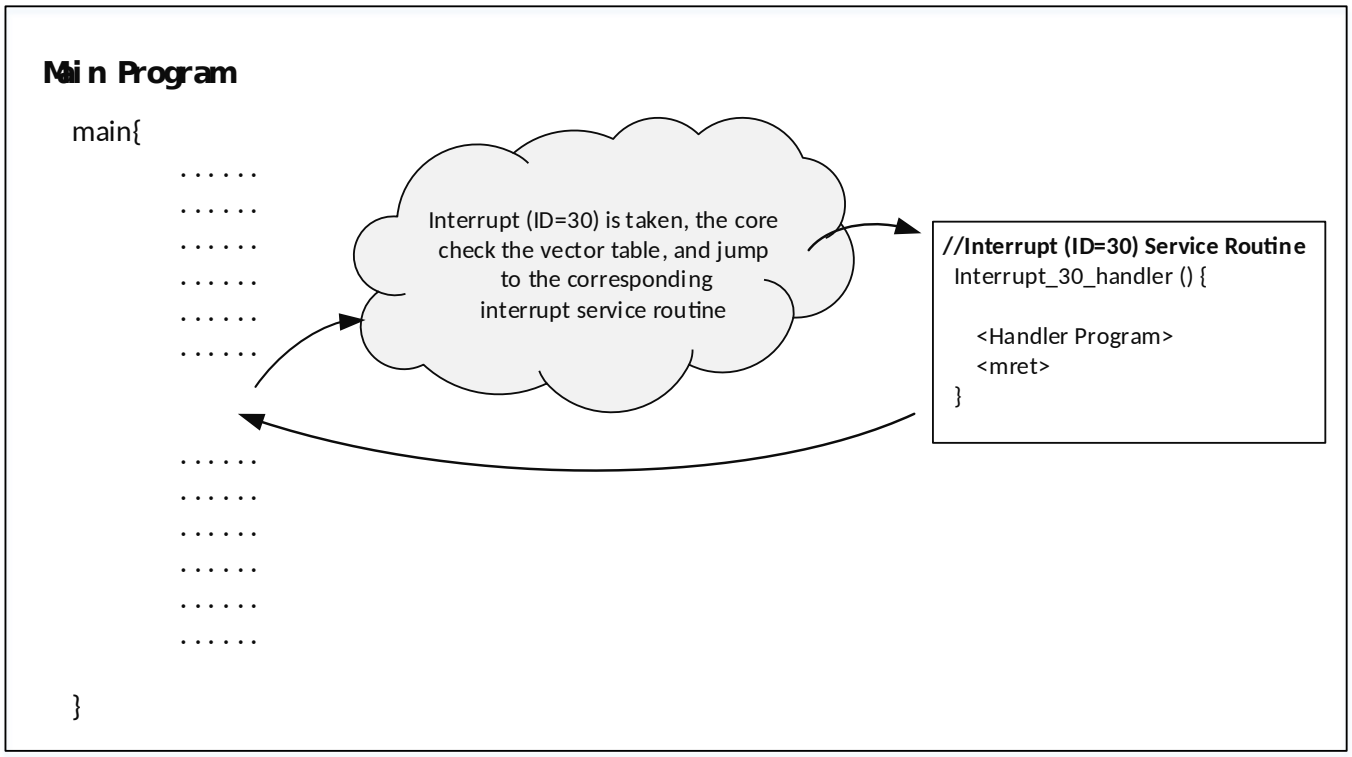


Figure 6-17 Interrupt tail-chaining

6.13.2. Vectored Processing Mode

6.13.2.1 Feature and Latency of Vectored Processing Mode

If the interrupt is vectored, once it is taken, the core will jump to the target address saved in the Vector Table Entry directly, which is the corresponding interrupt service routine (C function) of the interrupt, as shown in Feature 6 -18.



Feature 6-18 Example for vectored interrupt

Vectored Processing Mode has the following features:

- The core will jump directly to the interrupt service routine without context saving. Therefore, the latency of the vectored interrupt is very short. Ideally, it only takes 6 cycles from the interrupt initiation to the execution of the first instruction of the interrupt service routine (C function), because the hardware only need to perform one lookup and jump.
- For an interrupt service routine of a vectored interrupt, the indication" __attribute__ ((interrupt))" is required to indicate compiler this C function

is an interrupt service routine. Why this attribute is needed? Explained as below:

- In the vector processing mode, since the core does not save the context before jumping to the interrupt service routine, theoretically the interrupt handler cannot call any sub-function which means the handler must be a leaf function.
- If the interrupt service routine accidentally calls another sub-function, which means the routine is not a leaf function, it will cause a function error because of context corruption.
- In order to avoid this accidental error, as long as the "__attribute__ ((interrupt))" is used to indicate this function is an interrupt handler, the compiler will automatically detect if this function calls any sub-function. If it calls any sub-function, the compiler will automatically insert a piece of code to save the context. Note: in this case, although the function correctness is guaranteed, the overhead caused by context saving will actually increase the latency of the response of the interrupt (equivalent to the non-vectored interrupt processing) and cause the expansion of the code size. Hence, in practice, it is not recommended to call other sub-functions in the interrupt service routine of a vectored interrupt.

6.13.2.2 Preemption of Vectored Interrupt

In vectored processing mode, the core does not perform any special operation before jumping to the interrupt service routine, and the value of mstatus.MIE is updated to 0 by the hardware, which means the interrupt is global disabled and no new interrupt will be taken once the core is handling the interrupt. Therefore, the vectored processing mode does not support interrupt preemption by default. In order to support vectored interrupt preemption, a special stack-push operation is necessary at the beginning of the interrupt service routine as shown in Figure 6 - 19:

- First save the CSRs mepc, mcause, msubm to the stack. These CSRs are saved to ensure that subsequent interrupt preemption can perform correctly, because the new taken interrupt will overwrite the values of mepc, mcause, and msubm, so they need to be saved to the stack first.

- Re-enable the global interrupt enable bit, that is, set the mstatus.MIE to 1. After the global interrupt enable bit is set, the new interrupt can be taken to allow the mechanism of interrupt preemption.

- At the end of the interrupt service routine, it is necessary to add the operation of context restoring. And before CSRs mepc, mcause, and msubm are restored from the stack, the global interrupt enable bit must be set as 0 to guarantee the atomicity of the restoring operation of CSRs mepc, mcause, and msubm (not interrupted by the new interrupt).

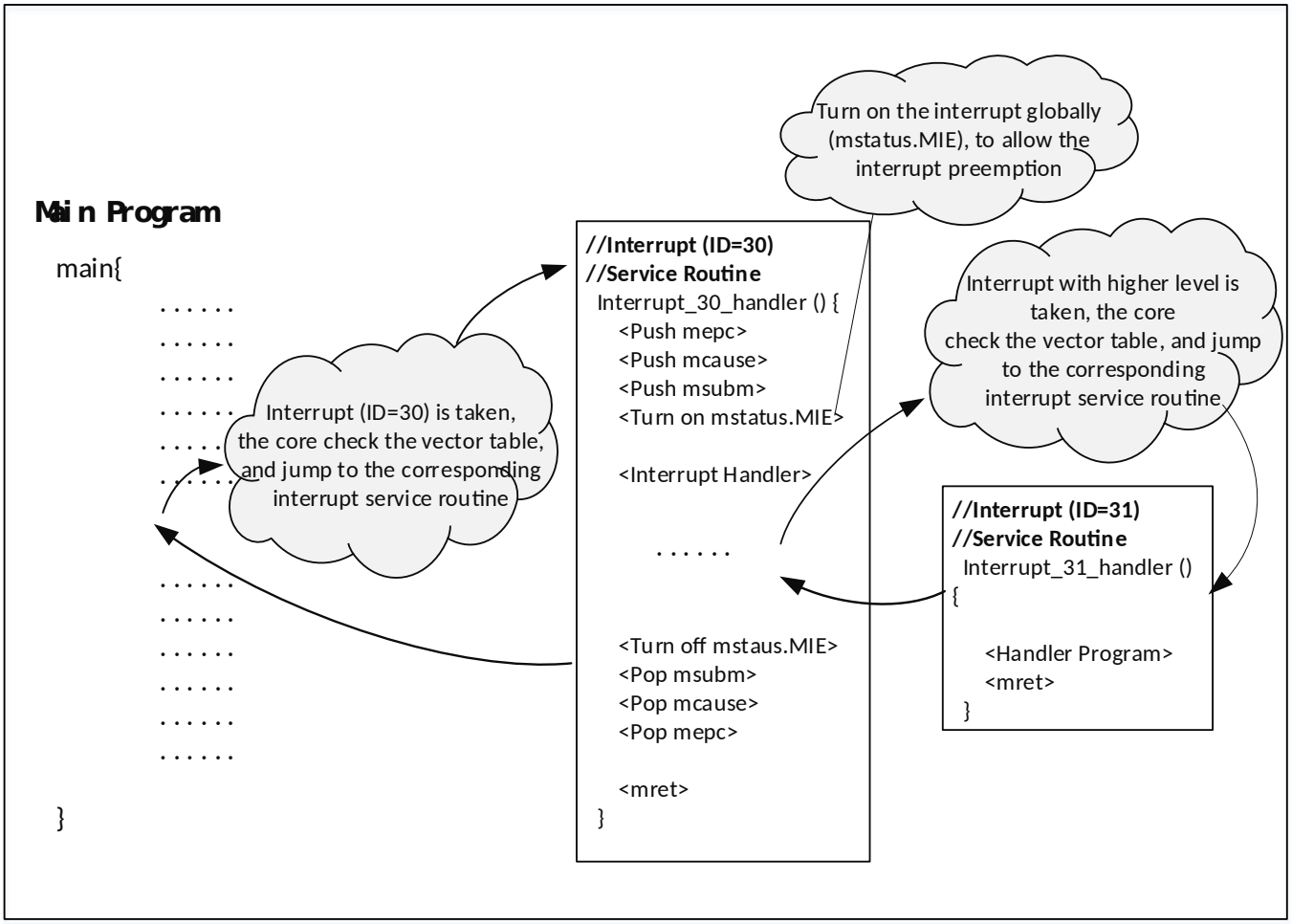


Figure 6-19 Example for vectored interrupt supported preemption

As described above, with the special processing, the vectored processing mode can support interrupt preemption, as shown in Figure 6 -20: assuming that the three interrupts 30, 31, 32 come sequentially, and the level of interrupt 32 is greater than the level of interrupt 31 which is greater than the level of interrupt 30. Since then, the subsequent interrupts will preempt interrupts that were previously processed to form interrupt preemptions.

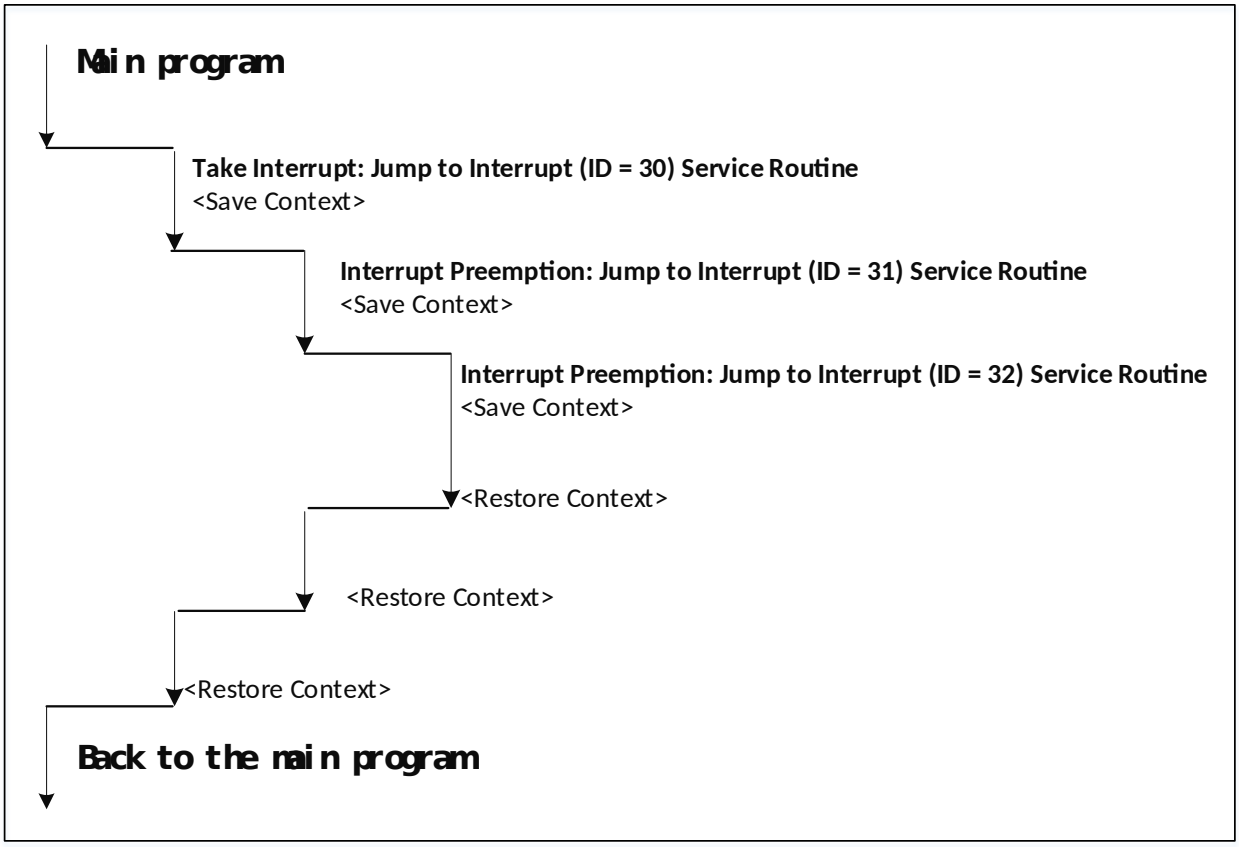


Figure 6-20 Interrupt preemptions caused by three sequential vectored interrupts

6.13.2.3 Vectored Interrupt Tail-Chaining

For the vectored processing mode, the core does not save the context before jumping to the interrupt service routine, so the meaning of “interrupt tail-chaining” is not significant. Therefore, the vectored processing mode does not support the features of “interrupt tail-chaining”.

7. Nesting of Interrupt, NMI and Exception

This section summarized the nesting of Interrupt, NMI and Exception.

- An interrupt can preempt another interrupt, and an exception can preempt another exception, but NMI cannot preempt another NMI, explained as below:

  - If the core is in NMI handling mode, and it encounters another NMI, then this new NMI will be masked. Therefore, NMI cannot preempt another NMI. Please see Section 5.2. for more details.
  - If the core is in exception handling mode, and it encounters another exception, then exception nesting will happen. Please see Section 3.5 for more details.
  - If the core is in interrupt handling mode, and it encounters another interrupt, then interrupt nesting may happen. Please see Section 5.11 for more details.

- And preemption also happens between interrupts, exceptions or NMIs, explained as below:

  - If the core is in interrupt handling mode, and it encounters an exception, then the core will enter exception handling mode.
  - If the core is in NMI handling mode, and it encounters an exception, then the core will enter exception handling mode.
  - If the core is in interrupt handling mode, and it encounters an NMI, then the core will enter NMI handling mode.
  - If the core is in exception handling mode, and it encounters an NMI, then the core will enter NMI handling mode.
  - Note: The global interrupt-enable bit msttaus.MIE is cleared to zero automatically when the core is in exception/NMI handling mode, so the core will not take any interrupt herein (unless the software to turn on mstatus.MIE explicitly in exception/NMI handler).

7.1. Two Levels of NMI/Exception State Save Stacks

As described in last section, the Nuclei processor core can support 3 kinds of NMI/Exception preemption:

- An NMI preempts an exception
- An exception preempts another exception

- An exception preempts another NMI
- Note: Since NMI is masked when the core is in NMI handling mode, one NMI cannot preempt another NMI.

In order to make sure the context (e.g., mepc, mcause, etc.) is not corrupted during nesting, the Nuclei processor core has implemented a configurable hardware feature, that is, self-defined Two Levels of NMI/Exception State Save Stacks which can save up to 3-levels NMI/Exception core execution states.

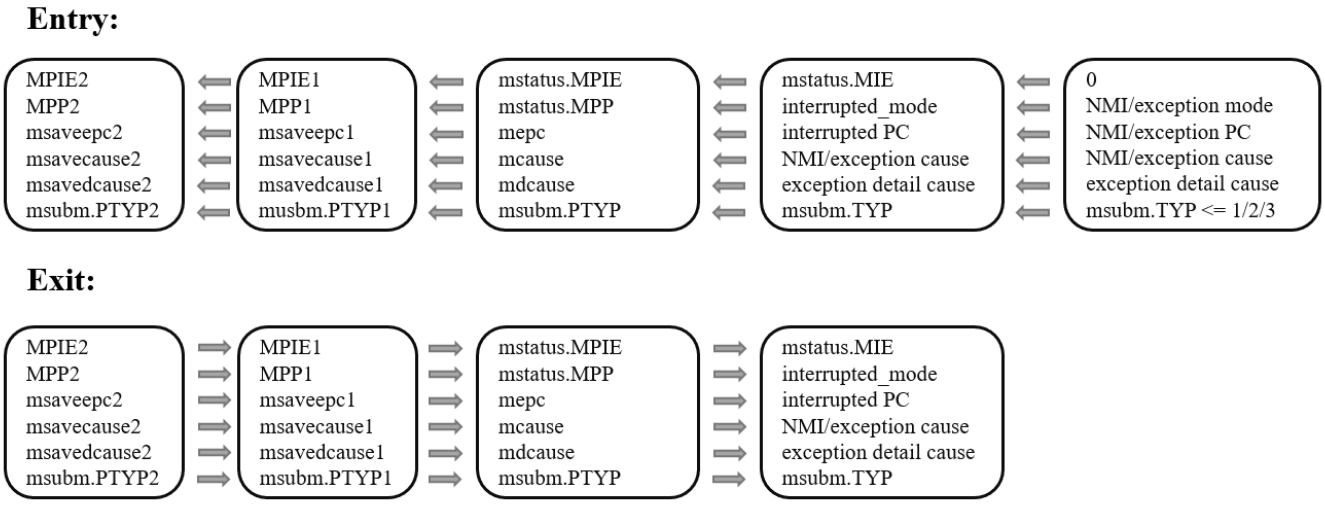


Figure 7-21 Two Levels of NMI/Exception State Save Stacks

As depicted in Figure 7 -21, this implementation can supports up to 2-levels recoverable NMI/Exception nesting, detailed in the next sections.

7.2. Enter NMI/Exception Preemption

When take an NMI or exception, hardware behaviors of the Nuclei processor core are as below:

- Stop executing the current program, and jump to a new PC to execute.
  - If it is an exception trap, then the jump target PC is the address defined in mtvec.
  - If it is an NMI trap, then the jump target PC is the address defined in mnvec.
- NMI/Exception is handling in Machine Mode, so the Privilege Mode will be switched to the Machine Mode when the core takes one NMI/Exception.
- Updates the following relevant CSRs' specified fields in the way as shown in

Figure 7 -21:

- mepc: record the PC encountered the handling NMI/Exception, and can be used to restore the PC after exiting the handling NMI/Exception.
- msaveepc1: the first level NMI/Exception State Save Stack, records value of mepc. This CSR is used to restore the value of mepc when the core returns from the handling NMI/Exception.
- msaveepc2: the second level NMI/Exception State Save Stack, records the value of msaveepc1. This CSR is used to restore the value of msaveepc1 when the core returns from the handling NMI/Exception.
- mstatus: following the RISC-V standard privileged architecture.
  - MPIE: save the value of MIE before taking the handling NMI/Exception.
  - MPP: save the value of Privilege Mode before taking the handling NMI/Exception.
- msavestatus:
  - MPIE1: the first level NMI/Exception State Save Stack, records the value of MPIE. This CSR is used to restore the value of MPIE when the core returns from the handling NMI/Exception.
  - MPIE2: the second level NMI/Exception State Save Stack, records the value of MPIE1. This CSR is used to restore the value of MPIE1 when the core returns from the handling NMI/Exception.
  - MPP1: the first level NMI/Exception State Save Stack, records the value of MPP. This CSR is used to restore the value of MPP when the core returns from the handling NMI/Exception.
  - MPP2: the second level NMI/Exception State Save Stack, records the value of MPP1. This CSR is used to restore the value of MPP1 when the core returns from the handling NMI/Exception.
- mcause: save the cause of the handling NMI/Exception.
- msavecause1: the first level NMI/Exception State Save Stack, records the value of mcause.
- msavecause2: the second level NMI/Exception State Save Stack, records the value of msavecause1.
- msubm:
  - TYP : save the trap type of the current handling NMI/Exception
  - PTYP: records value of TYP.
  - PTYP1: the first level NMI/Exception State Save Stack, records the value of PTYP. This CSR is used to restore the value of PTYP when the core returns from the handling NMI/Exception.
  - PTYP2 : the second level NMI/Exception State Save Stack, records

the value of PTYP1. This CSR is used to restore the value of PTYP1 when the core returns from the handling NMI/Exception.

7.3. Exit NMI/Exception Preemption

After handling the NMI/Exception, the core needs to exit from the NMI/Exception handler eventually, and return to execute the main program or handle the next level preempted NMI/Exception. Before exit the current NMI/Exception handler, the relevant CSRs and core status need to be restored by executing the mret instruction. The hardware behavior of the processor after executing mret instruction are:

- Stop the execution of the current program, and start from the PC address defined by the CSR mepc.
- According to the value of mstatus.MPP to update the Privilege Mode.
- Update the fields of the relevant CSRs in reversed way as shown in Figure 7 -21:
  - mepc
  - msaveepc1
  - msavestatus
  - mcause
  - msavecause1
  - msubm
    - TYP
    - PTYP
    - PTYP

8. TIMER Unit Introduction

8.1. TIMER Overview

The Timer Unit (TIMER for short) is used to generate the Timer Interrupt and Software Interrupt in Nuclei processor core.

8.2. TIMER Registers

The TIMER is a memory-mapped unit:

- For the base address of the TIMER unit, please refer to the specific datasheet of the Nuclei processor core.
- Registers and the corresponding offset in the TIMER unit are shown in Table 8 -1.

Table 8-1 The addresses offset of registers in the TIMER unit

Offset	Permission/Width	Register Name	Default Value	Function Description
0x0	RW/4B	mtime_lo	0x00000000	Reflect the lower 32-bit value of mtime. Shadow copy of MTIME in CLINT mode
0x4	RW/4B	mtime_hi	0x00000000	Reflect the upper 32-bit value of mtime. Shadow copy of MTIME in CLINT mode
0x8	RW/4B	mtimecmp_lo	0xFFFFFFFF	Set the lower 32-bit value of mtimecmp. Shadow copy of MTIMECMP for Hart-0 in CLINT mode
0xC	RW/4B	mtimecmp_hi	0xFFFFFFFF	Set the upper 32-bit value of mtimecmp. Shadow copy of MTIMECMP for Hart-0 in CLINT mode
0xFF0	RW/4B	msftrst	0x00000000	Generate soft-reset request.
0xFF8	RW/4B	mtimectl	0x00000000	Control some features of the time counter.
0xFFC	RW/4B	msip	0x00000000	Generate the Software Interrupt. Shadow copy of MSIP for Hart-0 in CLINT mode
0x1000	RW/4B	MSIP for	0x0000000	Software Interrupt for Hart-0 in

		Hart-0	0	CLINT mode
0x1004	RW/4B	MSIP for Hart-1	0x00000000	Software Interrupt for Hart-1 in CLINT mode
0x1008	RW/4B	MSIP for Hart-2	0x00000000	Software Interrupt for Hart-2 in CLINT mode
0x100c	RW/4B	MSIP for Hart-3	0x00000000	Software Interrupt for Hart-3 in CLINT mode
0x5000	RW/8B	MTIMECMP for Hart-0	0x00000000	MTIMECMP for Hart-0 in CLINT mode
0x5008	RW/8B	MTIMECMP for Hart-1	0x00000000	MTIMECMP for Hart-1 in CLINT mode
0x5010	RW/8B	MTIMECMP for Hart-2	0x00000000	MTIMECMP for Hart-2 in CLINT mode
0x5018	RW/8B	MTIMECMP for Hart-3	0x00000000	MTIMECMP for Hart-3 in CLINT mode
0xCFF8	RW/8B	MTIME	0x00000000	MTIME in CLINT mode
Note: ■ Registers in the TIMER unit only support aligned read and write access with WORD size.				

The functionality of each register is described in the following sections.

8.2.1. Time Counter Register mtime

The key points of TIMER unit are as follows:

- The TIMER implements a 64-bit register mtime, which is composed of {mtime_hi, mtime_lo}. This register reflects the value of the 64-bit timer. The timer is turned on by default, so it will always count after reset.
- The increment frequency of the counter is controlled by the core's input signal mtime_toggle_a or core's always-on clock input core_aon_clk. Please refer to the specific datasheet of the Nuclei processor core for details about this signal.

8.2.2. Generate the Timer Interrupt through mtime and mtimecmp

The TIMER unit can be used to generate the timer interrupt. The key points are as follows:

- The TIMER implements a 64-bit register mtimecmp, which is composed of

{mtimecmp_hi, mtimecmp_lo}. This register is used as the comparison value of the timer. If the value of mtime is greater than the value of mtimecmp, then a timer interrupt is generated.

- If mtimectl.CMPCLREN is set as 1, then the mtime will be automatically cleared to zero when the value of mtime is greater than the value of mtimecmp, and then restart counting from zero.
- If mtimectl.CMPCLREN is set as 0, then the mtime will always increments normally. The software can clear the timer interrupt by overwriting the value of mtimecmp or mtime (so that the value of mtimecmp is greater than the value of mtime).

Note: the timer interrupt is connected to the ECLIC unit as unified interrupt management. Please see Chapter 10. for details of ECLIC.

8.2.3. Control the Timer Counter through mtimectl

The register mtimectl is implemented to control the behaviors of timer counting , as shown in Table 8 -2.

Table 8-2 mtimectl bit fields

Field	Bits	Permission	Default Value	Description
Reserved	7:1	Readable, write ignored	N/A	Reserved, ties to 0
CLKSRC	2	RW	0	Select the source of increment frequency. If this field is 1, then the increment frequency is frequency of core_aon_clk, otherwise the increment frequency is controlled by mtime_toggle_a, please refer to the specific datasheet of the Nuclei processor core for details about this signal.
CMPCLREN	1	RW	0	Control the timer count to clear-to-zero or not. If this field is 1, then the mtime register will be cleared to zero after generated timer interrupt, otherwise it increments normally.
TIMESTOP	0	RW	0	Control the timer count or pause. If this field is 1, then the timer is paused, otherwise it increments normally.

Note : If CMPCLREN is enabled, the timer interupt request will be a pulse request. In this case, timer interrupt should be set to edge-trigger mode.

8.2.4. Generating the Software Interrupt through msip

The TIMER unit can be used to generate the Software Interrupt. The register msip is implemented in the TIMER unit as shown in Table 8 -3, only the least significant bit of msip is an effective bit. This bit is used to generate the software interrupt directly:

- The software generates the software interrupt by writing 1 to the msip register;
- The software clears the software interrupt by writing 0 to the msip register.

Note: the soft interrupt is connected to the ECLIC unit as unified interrupt management. Please see Chapter 10. for details of ECLIC.

Table 8-3 msip bit fields

Field	Bits	Permission	Default Value	Description
Reserved	31:1	Readable, write ignored	N/A	Reserved, ties to 0
MSIP	0	RW	0	This bit is used to generate the software interrupt

8.2.5. Generating the Soft-Reset Request

The TIMER unit can be used to generate the Soft-Reset request. The register msftrst is implemented in the TIMER unit as shown in Table 8 -4, only the least significant bit of msftrst is an effective bit. This bit is used to generate the Soft-Reset request directly:

- The software generates the Soft-Reset request by writing 0x80000a5f to the msftrst register. Requiring to write such a complicate number is to avoid the random mis-operation of software writing.
- The most significant bit of msftrst can only be clear by reset , so if the SoC reset the core in respond to Soft-Reset request, then msftrst register will be reset (and cleared to zero).

Note: The core's output signal sysrstreq (active high) is used to carry out the Soft-Reset request, the SoC should reset the core (assert core_reset_n, not por_reset_n) in respond to the request. Please refer to the specific datasheet of the Nuclei processor core for details about the signals sysrstreq, core_reset_n and por_reset_n.

Table 8-4 msftrst bit fields

Field	Bits	Permission	Default Value	Description
MSFTRST	31	RW	0	This bit is used to generate the Soft-Reset Request
Reserved	30:0	Readable, write ignored	N/A	Reserved, ties to 0

9. PLIC Unit Introduction

9.1. PLIC Overview

For the Linux capable applications or symmetric multi-processor (SMP) applications, Nuclei processor core have been equipped with a Platform-Level Interrupt Controller (PLIC), which is part of RISC-V standard privileged architecture specification (riscv-privileged-v1.11.pdf), user can easily get the original copy (riscv-privileged-v1.11.pdf) from “Nuclei User Center” website http://user.nucleisys.com or from other public channels https://github.com/riscv/riscv-plic-spec/blob/master/riscv-plic.adoc.

Note:

- The PLIC unit arbitrates the interrupt sources to the processor core (as the interrupt target) by a line as shown in Figure 11 -28.
- The PLIC need to be enabled by setting the LSB bits of CSR registers mtvec as CLINT mode. Please refer to Section 6.2.1. for the details.
- The PLIC is functionally exclusive to ECLIC. The ECLIC and PLIC connection diagrams are as described in Chapter 11..

9.2. PLIC Registers

The RISC-V standard privileged architecture specification does not specify the exact register offset for PLIC. The Nuclei processor core implements PLIC as a memory-mapped unit:

- The base address of the PLIC unit, please refer to the specific datasheet of the Nuclei processor core.
- Registers and the corresponding offset in the PLIC unit are shown in Table 9 -5.

Table 9-5 The addresses offset of registers in the PLIC unit

Offset	Width	Permission	Description	Default Value
0x00_0000			Reserved (source 0 does not exist)	

0x00_0004	4B	RW	Source 1 priority	0x0
0x00_0008	4B	RW	Source 2 priority	0x0
……		……	……	……
0x00_0FFC	4B	RW	Source 1023 priority	0x0
……		……	……	……
0x00_1000	4B	R	Start of pending array (bit 0-31)	0x0
……		……	……	……
0x00_107C	4B	R	Last word of pending array (bit 992-1023)	0x0
……		……	……	……
0x00_2000	4B	RW	Start of Hart 0 M-mode interrupt enables (source 0-31)	0x0
……		……	……	……
0x00_207C	4B	RW	Last word of Hart 0 M-mode interrupt enables (source 992-1023)	
0x00_2080	4B	RW	Start of Hart 0 S-mode interrupt enables (source 0-31)	
……		……	……	
0x00_20FC	4B	RW	Last word of Hart 0 S-mode interrupt enables (source 992-1023)	
0x20_0000	4B	RW	Hart 0 M-mode Priority threshold	0x0
0x20_0004	4B	RW	Hart 0 M-mode Claim/Complete	0x0
……			Reserved	
0x20_1000	4B	RW	Hart 0 S-mode Priority threshold	0x0
0x20_1004	4B	RW	Hart 0 S-mode Claim/Complete	0x0

Note:

- PLIC registers only support aligned access which is the size of word.
- The above “R” means read-only, and any write to this read-only register will be ignored without generating bus error.
- The PLIC unit may not be configured to support 1024 interrupt sources. If an interrupt is not present in the hardware, the corresponding registers of memory locations appear hardwired to zero.
- The PLIC unit has M-mode and S-mode dedicated registers, which can be configured to trigger M-mode interrupt or S-mode interrupt, by default, both M-mode and S-mode interrupts are all handled in M-mode, when Mideleg is configured, S-mode interrupts will be delegated to be handled in S-mode, but the M-mode interrupts will still be handled in M-mode regardless of the Mideleg.
- The PLIC unit is memory mapped and needs to be protected by PMP if different access permissions are needed for different sources.

10. ECLIC Unit Introduction

10.1. ECLIC Overview

For the real-time or microcontroller applications, the fast interrupt handling scheme is very important, hence, Nuclei processor core have been equipped with an Enhanced Core Local Interrupt Controller (ECLIC), which is optimized based on the RISC-V standard CLIC, to manage all interrupt sources.

Note:

- The ECLIC unit only serves one core and is private inside the core, as shown in Figure 10 -23.
- The ECLIC need to be enabled by setting the LSB bits of CSR registers mtvec as ECLIC mode. Please refer to Section 6.2.1. for the details.
- The ECLIC is functionally exclusive to PLIC. The ECLIC and PLIC connection diagrams are as described in Chapter 11..

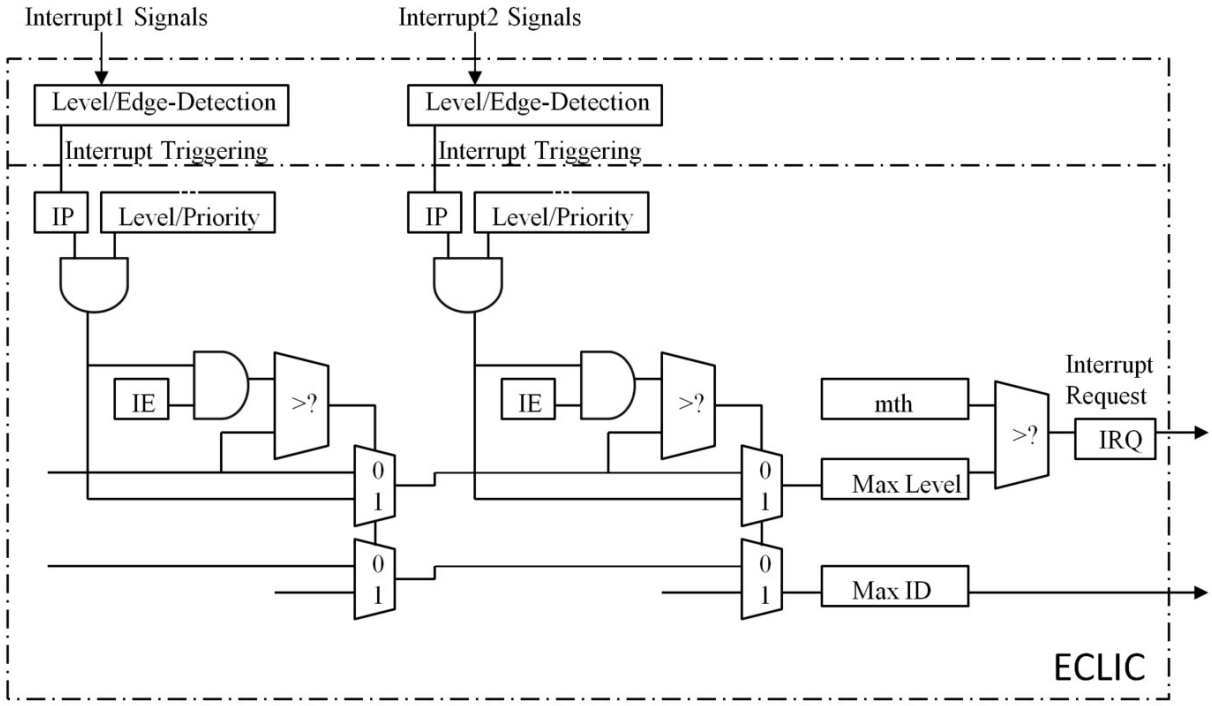


Figure 10-22 The logic structure of the ECLIC unit

The ECLIC unit is used to arbitrate multiple internal and external interrupts, send interrupt request to core, and support the interrupt preemption. The registers of the ECLIC are described in Table 10 -7, and its structure is shown in Figure 10 -22 and the related concepts are as follows:

- ECLIC interrupt target
- ECLIC interrupt source
- ECLIC interrupt source ID
- ECLIC registers
- ECLIC interrupt enable bits
- ECLIC interrupt pending bits
- ECLIC interrupt level or edge triggered attribute
- ECLIC interrupt level and priority
- ECLIC interrupt vectored or non-vectored processing mode
- ECLIC interrupt threshold level
- ECLIC interrupt arbitration mechanism
- ECLIC interrupt response, preemption, tail-chaining mechanism

These will be detailed at next sections.

10.2. ECLIC interrupt target

The ECLIC unit arbitrates the interrupt sources to the processor core (as the interrupt target) by a line as shown in Figure 10 -23.

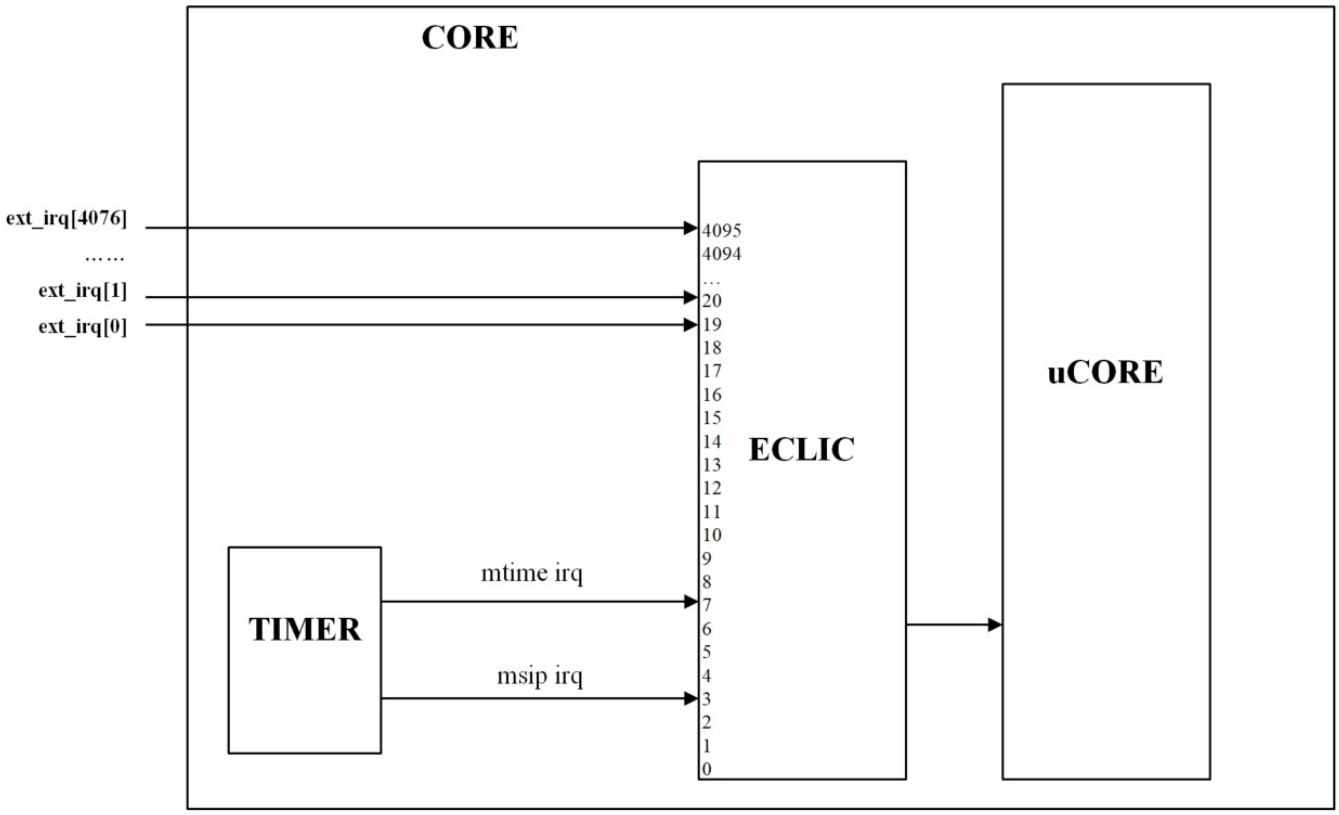


Figure 10-23 ECLIC Connection (when ECLIC is enabled)

10.3. ECLIC Interrupt Source

As shown in Figure 10 -23, the ECLIC unit can support up to 4096 interrupt sources. The ECLIC unit has defined the following concepts for each interrupt source:

- ID
- IE
- IP
- Level or Edge-Triggered
- Level and Priority
- Vector or Non-Vector Mode

These concepts will be detailed at next sections.

10.4. ECLIC Interrupt Source ID

The ECLIC unit has assigned a unique ID to each interrupt source. For example, if a hardware implementation of the ECLIC unit really configured to support 4096 interrupts, then the ID should be 0 to 4095. Note:

- In the Nuclei processor core, the interrupt IDs ranged from 0 to 18 are reserved for the core-specified internal interrupts.
- The interrupt source ID greater than 18 can be used by the user to connect external interrupt sources.

The details are shown in Table 10 -6.

Table 10-6 ECLIC interrupt sources and assignment

ECLIC interrupt ID	Function	Interrupt Source Description
0	*Reserved*	*This source is not used*
1	*Reserved*	*This source is not used*
2	*Reserved*	*This source is not used*
3	Software interrupt	The software interrupt generated by the TIMER
4	*Reserved*	*This source is not used*
5	*Reserved*	*This source is not used*
6	*Reserved*	*This source is not used*
7	Timer interrupt	The timer interrupt generated by the TIMER
8	*Reserved*	*This source is not used*
9	*Reserved*	*This source is not used*
10	*Reserved*	*This source is not used*
11	*Reserved*	*This source is not used*
12	*Reserved*	*This source is not used*
13	*Reserved*	*This source is not used*
14	*Reserved*	*This source is not used*
15	*Reserved*	*This source is not used*
16	*Reserved*	*This source is not used*
17	*Reserved*	*This source is not used*
18	*Reserved*	*This source is not used*
19 ~ 4095	External interrupt	Normal external interrupt defined by users. Note: ■ Although the ECLIC unit can support up to 4096 interrupt sources per programming mode, the actual number of supported interrupt sources is indicated in the field clicinfo.NUM_INTERRUPT.

10.5. ECLIC Registers

The ECLIC is a memory-mapped unit.

- The base address of the ECLIC unit, please refer to the specific datasheet of the Nuclei processor core.
- Registers and the corresponding offset in the ECLIC unit are shown in Table 10 -7.

Table 10-7 The addresses offset of registers in the ECLIC unit

Offset	Permission	Register	Width
0x0000	RW	cliccfg	8-bit
0x0004	R	clicinfo	32-bit
0x000b	RW	mth	8-bit
0x1000+4*i	RW	clicintip[i]	8-bit
0x1001+4*i	RW	clicintie[i]	8-bit
0x1002+4*i	RW	clicintattr[i]	8-bit
0x1003+4*i	RW	clicintctl[i]	8-bit

Note:

- The above “i” indicates the interrupt ID, an interrupt i has its own corresponding clicintip[i], clicintie[i], clicintattr[i], and clicintctl[i] registers.
- ECLIC registers only support aligned access which is the size of byte, half-word or word.
- The above “R” means read-only, and any write to this read-only register will be ignored without generating bus error.
- The ELCIC unit may not be configured to support 4096 interrupt sources. If an index i is not present in the hardware, the corresponding clicintip[i], clicintie[i], clicintctl[i] memory locations appear hardwired to zero.
- The address space of ECLIC registers is the range from 0x0000 to 0xFFFF. The value in an address other than the address listed in the above table is constant 0.

These registers are detailed in the next sections.

10.5.1. cliccfg

This cliccfg register is a global configuration register. The software can set global configurations by write this register. Table 10 -8 describes the bit fields of this register.

Table 10-8 cliccfg bit fields

Field	Bits	Permission	Default Value	Description
Reserved	7:5	R	N/A	Reserved, ties to 0.
nlbits	4:1	RW	0	Used to specified the bit-width of level and priority in the register clicintctl[i]. Please see Section 10.9 for more details.
Reserved	0	R	N/A	Reserved, ties to 1.

10.5.2. clicinfo

The clicinfo register is a global info register. The software can query the global parameters by reading this register. Table 10 -9 describes the bit fields of this register.

Table 10-9 clicinfo bit fields

Field	Bits	Permission	Default Value	Description
Reserved	31:25	R	N/A	Reserved, ties to 0.
CLICINTCTLBITS	24:21	R	N/A	Used to specify the effective bit-width the register clicintctl[i]. Please see Section 10.9 for more details.
VERSION	20:13	R	N/A	Hardware implementation version number.
NUM_INTERRUPT	12:0	R	N/A	Number of interrupt sources supported by the hardware.

10.5.3. mth

The mth register is used the set the target interrupt threshold level. The software can set the target interrupt threshold level by writing this register. Table 10 -10 describes the bit fields of this register.

Table 10-10 mth bit fields

Field	Bits	Permission	Default Value	Descriptipon
mth	7:0	RW	N/A	Target threshold level register. Please see Section 10.11 for more details.

10.5.4. clicintip[i]

The clicintip[i] register is the pending flag register for the interrupt source. Table

10 -11 describes the bit fields of this register.

Table 10-11 clicintip[i] bit fields

Field	Bits	Permission	Default Value	Description
Reserved	7:1	RO	N/A	Reserved, ties to 0
IP	0	RW	0	Interrupt source pending flag. Please see Section 10.7 for more details.

10.5.5. clicintie[i]

The clicintie[i] register is the enable bit register for the interrupt source. Table 10 -12 describes the bit fields of this register.

Table 10-12 clicintip[i] bits fields

Field	Bits	Permission	Default Value	Description
Reserved	7:1	R	N/A	Reserved, ties to 0.
IE	0	RW	0	Interrupt enable bit. Please see Section 10.6 for more details.

10.5.6. clicintattr[i]

The clicintattr[i] register is used to indicate the attribute of the interrupt source. The software can set the attribute of the interrupt source by writing this register. Table 10 -13 describes the bit fields of this register.

Table 10-13 clicintattr[i] bits fields

Field	Bits	Accessibility	Default Value	Description
Reserved	7:6	R	N/A	Reserved, ties to 2'b11
Reserved	5:3	R	N/A	Reserved, ties to 0
trig	2:1	RW	0	Used to set the level or edge triggered attribute of the interrupt source. Please see Section 10.8 for more details.
shv	0	RW	0	Used to set whether the interrupt is vectored or non-vectored. Please see Section 10.10 for more details.

10.5.7. clicintctl[i]

The clicintctl[i] register is the control register of the interrupt source. The software

can set the level and priority by writing this register. The level and priority field are dynamically allocated based on the value of cliccfg.nlbits. Please see Section 10.9. for more details.

10.6. ECLIC Interrupt Enable Bit (IE)

As shown in Figure 10 -22, the ECLIC unit has allocated an interrupt enable bit (IE) for each interrupt source which is the field clicintie[i].IE whose function are the follows:

- The clicintie[i] register of each interrupt source is a both readable and writeable memory-mapped register. Hence the software can program it.
- If the clicintie[i] register is programmed to 0, it means that this interrupt source is masked.
- If the clicintie[i] register is programmed to 1, it means that this interrupt is enabled.

10.7. ECLIC Interrupt Pending Bit (IP)

As shown in Figure 10 -22, the ECLIC unit has allocated an interrupt pending bit (IP) for each interrupt source which is the field clicintip[i].IP whose function are the follows:

- If the IP bit of one interrupt source is 1, it means this interrupt is triggered. The trigger condition of the interrupt source depends on whether this interrupt is level-triggered or edge-triggered as described in Section 10.8..
- The IP bit of the interrupt source is both readable and writeable. The behavior of the software writing IP bits depends on whether the interrupt source is level or edge triggered. Please see Section 10.8. for more details.
- For edge-triggered interrupt source, the IP bit may be cleared by the hardware itself. Please see Section 10.8. for more details.

10.8. ECLIC Interrupt Source Level or Edge-Triggered Attribute

As shown in Figure 10 -22, each ECLIC interrupt source can be set as level triggered or edge triggered by setting the value of clicintattr[i].trig. The key points are the followings:

- When clicintattr[i].trig[0] == 0, this interrupt source is set as a level-triggered interrupt.
  - If the interrupt source is set as level-triggered, the IP bit of the interrupt source will reflect the level of the interrupt source in real time.
  - If the interrupt source is set as level-triggered, the IP bit reflects the level of the interrupt in real time, so software writes to this IP bit is ignored, that is, the software cannot set or clear the IP bit by writing operation. If the software needs to clear the interrupt pending bit, it can only be done by clearing the original source of the interrupt.
- When clicintattr[i].trig[0] == 1 and clicintattr[i].trig[1] == 0, this interrupt source is set as a rising edge-triggered interrupt:
  - If the interrupt source is set as rising edge-triggered, when the ECLIC detects the rising edge of the interrupt source, the interrupt source is triggered in the ECLIC, and the IP bit of the interrupt source is asserted.
  - If the interrupt source is set as rising edge-triggered, the IP bit is writeable for the software, which means the software can set or clear the IP bit by writing operations.
  - Note: for rising edge-triggered interrupt, in order to improve the efficiency of the interrupt processing, when the interrupt is taken and the core jumps to the ISR (Interrupt Service Routine), the hardware of the ECLIC will clear the IP bit automatically, and the software needs not to clear the IP bit in ISR.
- When clicintattr[i].trig[0] == 1 and clicintattr[i].trig[1] == 1, this interrupt source is set as a falling edge-triggered interrupt:
  - If the interrupt source is set as falling edge-triggered, when the ECLIC detects the falling edge of the interrupt source, the interrupt source is triggered in the ECLIC, and the IP bit of the interrupt source is asserted.
  - If the interrupt source is configured as falling edge-triggered, the IP bit

is writeable for the software, which means the software can set or clear the IP bit by writing operations

- Note: for rising edge-triggered interrupt, in order to improve the efficiency of the interrupt processing, when the interrupt is taken and the core jumps to the ISR (Interrupt Service Routine), the hardware of the ECLIC will clear the IP bit automatically, and the software needs not to clear the IP bit in ISR.

10.9. ECLIC Interrupt Level and Priority

As shown in Figure 10 -22, each interrupt sources of the ECLIC can be configured with specified level and priority, and the key points are the followings:

- The register clicintctl[i] of each interrupt source is 8-bit width theoretically, and effective bits actually implemented by the hardware are specified by the CLICINTCTLBITS in the register clicinfo. For example, if the value of the clicinfo.CLICINTCTLBITS field is 6, it means that only the upper 6-bit of the clicintctl[i] register are true valid bits, and the lowest 2 bits are tied to 1, as shown in Figure 10 -24.

  - Note: the field CLICINTCTLBITS is a readable constant value, and the software cannot overwrite it. The theoretically reasonable value range of it is 2 <= CLICINTCTLBITS <= 8. The actual value is determined by the specified hardware implementation. Please refer to the specific datasheet of the Nuclei processor core.

- The effective bits of clicintctl[i] register have two dynamic fields, which are used to specify the level and the priority of the interrupt source. The width of the level filed is defined by field nlbits in cliccfg. For example, if the value of cliccfg.nlbits is 4, it means that the upper 4-bit of the effective bits in clicintctl[i] is the level field while the other lower effective bits is the priority field, as shown in the example in Figure 10 -24.

  - Note: the field cliccfg.nlbits is both readable and writeable, which means the software can change its value.

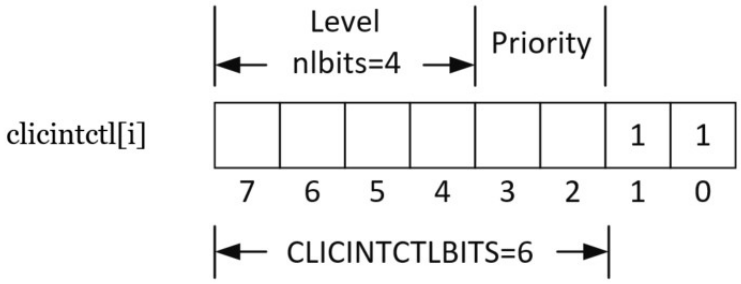


Figure 10-24 clicintctl[i] format example

- The key points of interrupt level are the followings:
  - The value of level is read in a left-aligned manner. Except the effective bits (defined by the value of cliccfg.nlbits), the low ineffective bits are all filled with the constant 1, as shown in the example in Figure 10 -25.
    - Note: if cliccfg.nlbits > clicinfo.CLICINTCTLBITS, it means that the number of bits indicated by nlbits exceeds the effective bits of the clicintctl[i] register, and the excess bits are all filled with the constant 1.
    - Note: if cliccfg.nlbits = 0, the value of level will be regarded as a fixed value 255. As shown in Figure 10 -26.
  - The greater value of level, the higher priority, note:
    - Higher-level interrupts can preempt lower-level interrupts, which is called as interrupt preemption, as detailed in Section 5.11.
    - If there are multiple pending interrupts (IP is 1), then the ECLIC needs to make an arbitration to determine which interrupt needs to be sent to the core to take. The arbitration needs to take the level of each interrupt source into the consideration. Please see Section 5.5 for details.

#nlbits	Encoding	The value of level
1	L..... (= L1111111)	127, 255
2	LL..... (= LL111111)	63, 127, 191, 255
3	LLL..... (= LLL11111)	31, 63, 95, 127, 159, 191, 223, 255
4	LLLL.... (= LLLL1111)	15, 31, 47, 63, 79, 95, 111, 127, 143, 159, 175, 191, 207, 223, 239, 255

"L" indicates the field of level
"." indicates the rest bits and are filled with the constant 1

Figure 10-25 Example for the decoding of level

Examples of cliccfg settings:

CLICINTCTLBITS	nlbits	clicintctl[i]	the value of level
0	2	.....	255
1	2	L.....	127, 255
2	2	LL.....	63, 127, 191, 255
3	3	LLL.....	31, 63, 95, 127, 159, 191, 223, 255
4	1	LPPP....	127, 255

Figure 10-26 Examples of cliccfg settings

- The key points of the interrupt priority are the follows:
  - The value of priority is also read in a left-aligned manner. Except the effective bits (clicinfo.CLICINTCTLBITS - cliccfg.nlbits), the low ineffective bits are all filled with the constant 1.
  - The greater the value of the priority, the higher priority, note:
    - The priority of the interrupt does not participate in the judgment of the interrupt preemption, which means whether the interrupt can be preempted or not has nothing to do with the value of the priority of the interrupt.
    - When multiple interrupts are simultaneously pending, the ECLIC needs to make an arbitration to determine which interrupt is sent to the core to handle. The arbitration needs to refer to the value of level/priority of each interrupt source. Please see Section 10.12. for details.

10.10. ECLIC Interrupt Vectored and Non-Vectored Processing Mode

Each interrupt source of the ECLIC can be set to vectored or non-vectored (via the shv field of the register clicintattr[i]). The key points are the followings:

- If the interrupt is set as vectored (clicintattr[i].shv==1), the core will directly jump to the target address stored in the vector table entry when the interrupt is taken. For a detailed description of the interrupt vectored processing mode, please see Section 6.13.2..

- If the interrupt is set as non-vectored (clicintattr[i].shv==0), the core will

jump to the common base entry shared by all interrupts when the interrupt is taken. For a detailed description of the interrupt non-vectored processing mode, please see Section 6.13.1..

10.11. ECLIC Interrupt Threshold Level

As shown in Figure 10 -22, the ECLIC can set the threshold level (mth) of a specific interrupt threshold level. The key points are as follows: :

- The mth register is an 8-bit register, all bits are readable and writable, and the software can write this register to set the threshold. Note: this threshold indicates a level value.
- Only when the level of the interrupt finally arbitrated by the ECLIC is higher than the value in the mth register, the interrupt can be sent to the processor core.

10.12. ECLIC Interrupt Arbitration Mechanism

The principles for the ECLIC to arbitrate all of its interrupt sources are as follows:

- Only interrupt sources that meet all of the following conditions can participate in the arbitration:
  - The enable bit (clicintie[i]) of the interrupt source must be 1.
  - The pending bit (clicintip[i]) of the interrupt source must be 1.
- The rules for the arbitration among all participated interrupt sources are:
  - First, check the level, the larger the level value of the interrupt source, the higher the arbitration priority.
  - If the level is equal, then check the priority, the interrupt source that has greater value of priority will have higher arbitration priority.
  - If both level and priority are equal, then the ID is taken into the consideration. The interrupt source with the larger interrupt ID has higher arbitration priority.
- If the level value of the interrupt source that wins the arbitration has a

greater value than the level value in mth, then the interrupt request signal to the core will be asserted.

10.13. ECLIC Interrupt Taken, Preemption and Tail-Chaining

After the ECLIC interrupt request is sent to the processor core, the core will respond to it. Through the coordination by the ECLIC and the core, the operation of interrupt preemption and tail-chaining are supported. Please see Section 6.11., Section 6.12., and Section 6.13.1. for more details.

11. ECLIC and PLIC Connection Diagram

The ECLIC and PLIC are independently configurable, so they can be co-existed or not, depends on configurations. The key points are as followings:

- For the single-core real-time or micro-controller applications, it is recommended to just have ECLIC configured, as depicted in Figure 11 -27.
- For the single-core Linux capable applications, it is recommended to configure PLIC. But in this case, the ECLIC can also be configured, hence, PLIC and ECLIC become coexisting.
  - If the ECLIC is enabled by software, then the interrupts will be handled by ECLIC, and the PLIC will be bypassed, as depicted in Figure 11 -28.
    - In this mode, the hardware of UX class core can also worked as microcontroller, i.e., Nuclei UX class core is downward-compatible to Nuclei NX class core.
  - If the ECLIC is disabled by software, then the interrupts will be handled by PLIC, and the ECLIC will be bypassed, as depicted in Figure 11 -29.
- For the symmetric multi-processor (SMP) Linux capable applications, it is recommended to just have PLIC configured only, as depicted in Figure 11 -30.

11.1. Single-core with ECLIC configured only

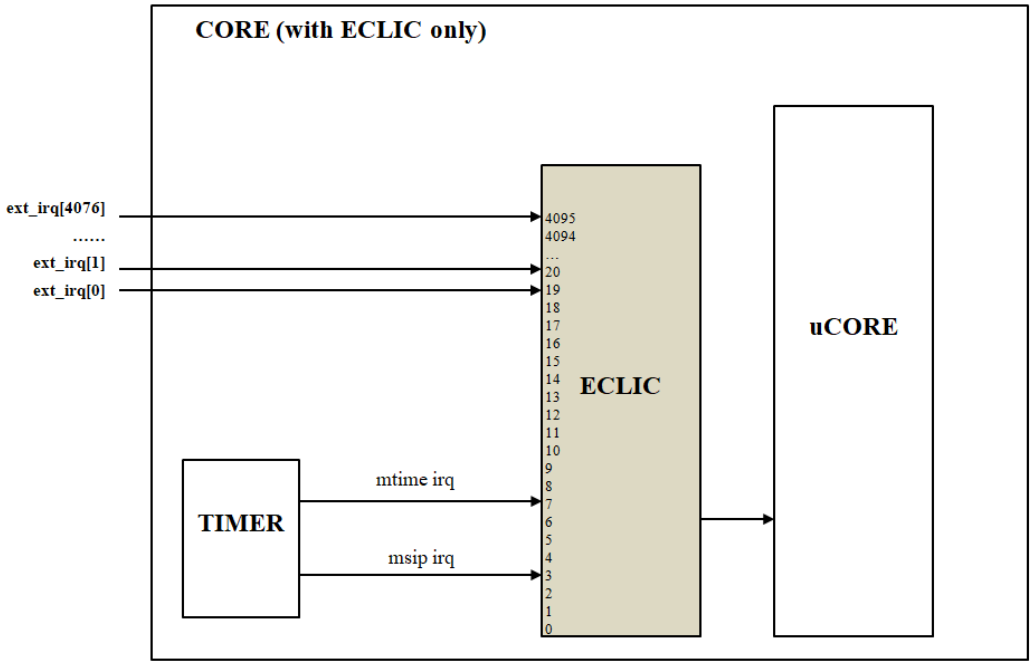


Figure 11-27 Interrupt Connection (for single-core with ECLIC configured only)

11.2. Single-core with PLIC/ECLIC configured and PLIC enabled

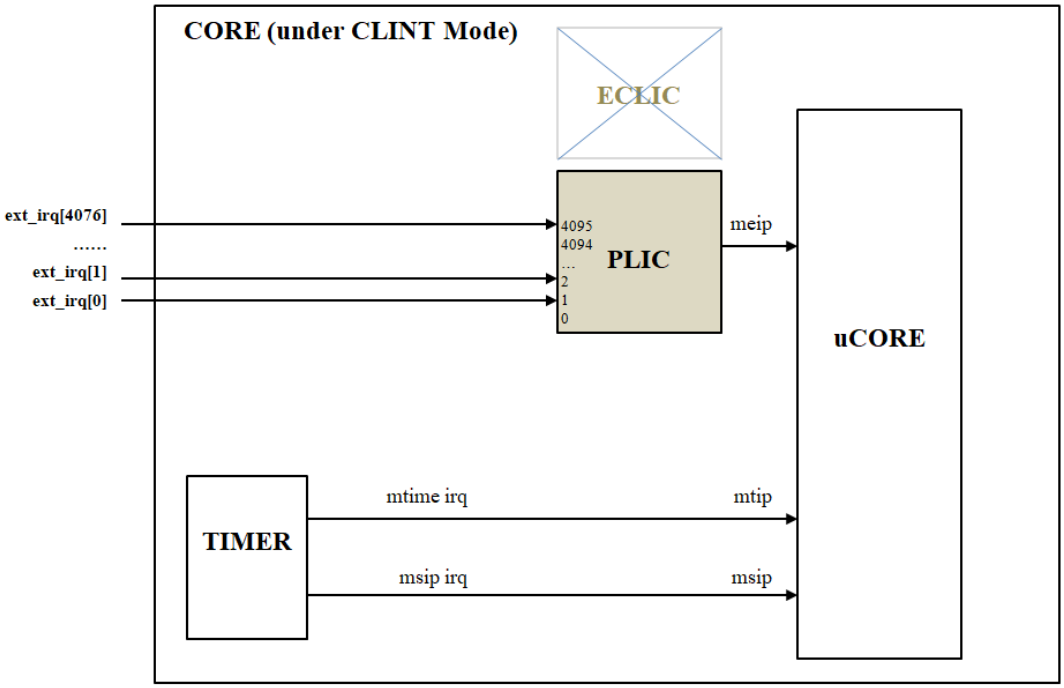


Figure 11-28 Interrupt Connection (for single-core with PLIC/ECLIC configured and PLIC

enabled)

11.3. Single-core with PLIC/ECLIC configured and ECLIC enabled

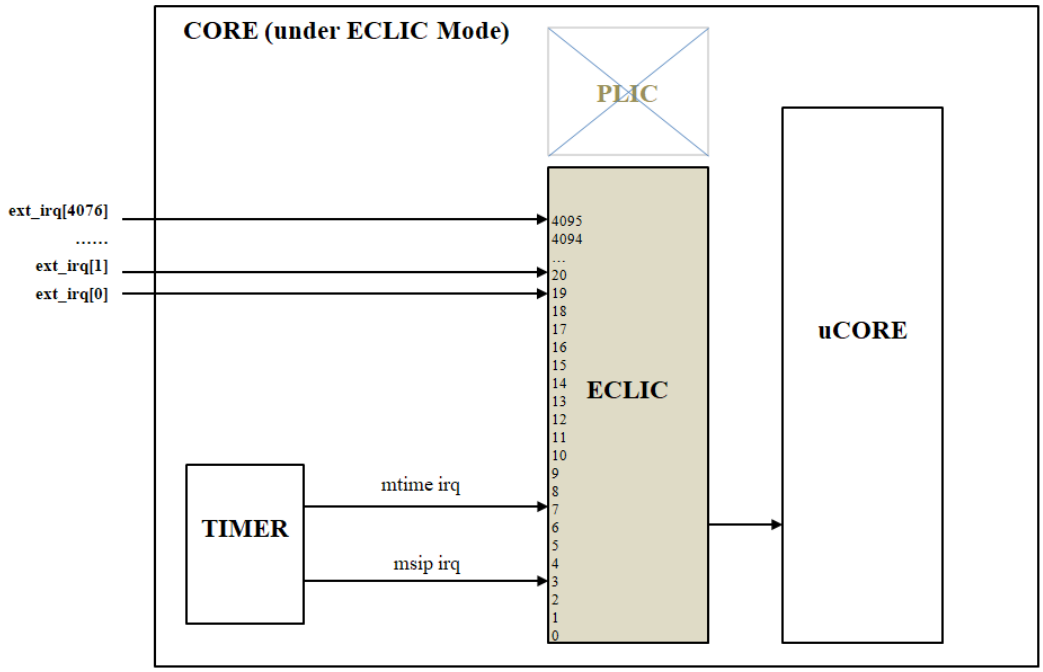


Figure 11-29 Interrupt Connection (for single-core with PLIC/ECLIC configured and ECLIC enabled)

11.4. Multi-core with PLIC configured only

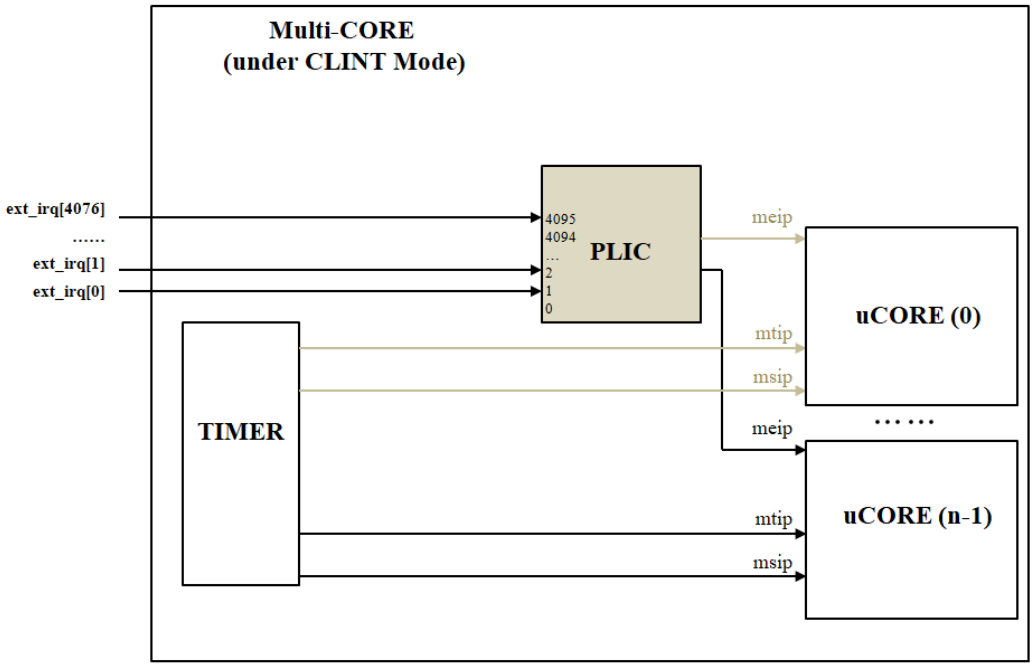


Figure 11-30 Interrupt Connection (for multi-core with PLIC configured only)

12. PMP Introduction

12.1. PMP Overview

Nuclei processor core can optionally support the PMP (Physical Memory Protection) features, which is part of RISC-V standard privileged architecture specification. User can easily get the original copy from “Nuclei User Center” website http://user.nucleisys.com or from other public channels.

12.2. PMP Specific Features to Nuclei Core

In order to simplify the hardware implementation, or there are some features are intrinsically hardware implementation relevant, Nuclei processor core have some PMP specific features, which is detailed at next sections.

12.2.1. Configurable PMP Entries Number

RISC-V standard privileged architecture specified the number of PMP entries up to 16, but in the real hardware the PMP entries is limited. Nuclei processor core have the PMP entries number configurable at the build time, please refer to the specific datasheet of the Nuclei processor core.

For those PMP CSR registers relevant to non-existed PMP entries, they are tied to zeros. For example, if the PMP Entries Number is configurable to 8, then the PMP entry 9 ~ 16 relevant CSR registers are tied to zeros.

12.2.2. Configurable PMP Grain

RISC-V standard privileged architecture allows platform to define its own PMP grain, please refer to the RISC-V privileged specification for more details of this PMP grain definitions. Nuclei processor core have the PMP grain configurable at the build time, please refer to the specific datasheet of the Nuclei processor core.

12.2.3. Support TOR mode in A field of pmpcfg<x> registers

RISC-V standard privileged architecture specified 4 types of modes in A field of pmpcfg<x> registers. Nuclei processor core already supports the TOR mode in A field of pmpcfg<x> registers.

12.2.4. Corner cases for Boundary Crossing

Since the RISC-V standard privileged architecture specified the PMP regions as naturally aligned power-of-2 regions (NAPOT), but in the Nuclei processor core hardware implementations, there might be unaligned access crossing the boundary, which is handled in this way:

- If it is normal load/store instructions:
  - If the processor core is configured to not support the unaligned memory access, then it will trigger address misaligned exception.
  - If the processor core is configured to support the unaligned memory access, then hardware will break the unaligned access into multiple byte or half-word aligned memory accesses (called micro-operations), each micro-opertation will be checked with PMP, if violated PMP permission, it will trigger Load or Store access fault exception, and the exception is imprecise.
- If it is AMO/LR/SC instructions:
  - Since the RISC-V standard privileged architecture specified AMO/LR/SC instructions definitely not support the unaligned memory access, hence it will trigger address misaligned exception.
- If it is the instruction fetching, since the processor core will always break the instruction fetching as the aligned memory access (e.g., 32bits or 64bits aligned), each aligned memory access will be checked with PMP, if violated PMP permission, it will trigger instruction access fault exception, and the exception is precise.

13. MMU Introduction

13.1. MMU Overview

Nuclei processor UX class core can have MMU (Memory Management Unit) configured for Linux capable applications, which implements the SV39 mode defined in RISC-V privileged specification, to support Page-Based 39-bit Virtual-Memory System, which can be used for converting Virtual-Address to Physical-Address and corresponding Permission Checking. MMU is part of RISC-V standard privileged architecture specification. User can easily get the original copy from "Nuclei User Center" website http://user.nucleisys.com or from other public channels.

13.2. MMU Specific Features to Nuclei Core

In order to simplify the hardware implementation, or there are some features are intrinsically hardware implementation relevant, Nuclei processor core have some MMU specific features, which is detailed at next sections.

13.2.1. Configurable TLB Entries Number

MMU has two level TLB (Translation Lookaside Buffer) implemented to cache the page tables for fast subsequent accessing: jTLB and i/d-tlb; jTLB is the joint TLB for both instruction and data page table, jTLB can be configured as 32, 64 or 128 entries; i/d-tlb is dedicate for instruction/data page table, each has 8 entries; i/d-tlb will be accessed first, if miss then jTLB will be accessed.

MMU supports 4KB, 2MB and 1GB page types, which uses Hardware Translation Table Walk mechanism to fetch page tables from memory when TLB miss without software handling.

14. WFI/WFE Low-Power Mechanism

The Nuclei processor core can support sleep mode for lower power consumption.

14.1. Enter the Sleep Mode

The Nuclei processor core can enter sleep mode by executing the WFI instruction. When the core executes the WFI instruction, it will perform following operations:

- Stop executing the current instruction stream immediately.
- Waiting for the core to complete any outstanding transactions, such as fetching instructions, load or store operations, to ensure that all the transactions sent to the bus are completed.
  - Note: if a memory access error exception occurs while waiting for a bus operation to complete, the core will enter the exception handling mode rather than sleep mode.
- When all of the outstanding transactions are completed, the core safely enters an idle state, which is called the sleep mode.
- When entered the sleep mode:
  - The clocks of the main units inside the core will be gated off to save dynamic power consumption;
  - The output signal core_wfi_mode of the core will be asserted to indicate that this core is in the sleep mode after executing the WFI instruction;
  - The output signal core_sleep_value of the core will output the value of the CSR register sleepvalue (Note: this signal is valid only when the core_wfi_mode is asserted; if the signal core_wfi_mode is 0, then the value of core_sleeep_value must be 0). The software can indicate different sleep modes (0 as shallow sleep or 1 as deep sleep) by set the CSR register sleepvalue in advance. Note:
    - The Nuclei processor core behaves exactly the same for different sleep modes. These sleep modes only provide different output value (via output signal core_sleeep_value) for different controlling scheme to the Power Management Unit (PMU) at the SoC system level.
    - Note: When entering to the deep sleep mode, the processor core will no longer be able to be debugged by JTAG interface.

14.2. Wait for Interrupt

The Wait for Interrupt mechanism refers to make the core enter the sleep mode, and the core keeps waiting for an interrupt to wake up.

As described in Section 14.1., the Wait for Interrupt mechanism can be implemented by executing the WFI instruction with setting the value of CSR wfe.WFE to 0.

14.3. Wait for Event

The Wait for Event mechanism refers to make the core enter the sleep mode, and the core keeps waiting for an event to wake up. When the core wakes up by the event, it continues to execute the previously interrupted instruction stream.

As mentioned in Section 14.1., the Wait for Event mechanism can be implemented by executing the WFI instruction combined with the following sequence of instructions:

> First step: set the value of wfe.WFE to 1.
>
> Second step: execute the WFI instruction. The core will stay in the sleep mode until an event or NMI wakes it up.
>
> Third step: restore the value of wfe.WFE to 0.

14.4. Exit the Sleep Mode

The key points of the Nuclei processor core exiting the sleep mode are as follows:

- The output signal core_wfi_mode of the core is cleared to 0.
- The core can be woken up in four ways:
  - NMI
  - Interrupt
  - Event
  - Debug request

These will be described in detail next.

14.4.1. Wake Up by NMI

NMI can always wake up the core. When the core detects a rising edge of the input signal nmi, the core is woken up and jumps to the NMI service routine.

14.4.2. Wake Up by Interrupt

Interrupts can wake up the core as well:

- If the value of wfe.WFE is set to 0, t he core will be waited for the interrupt to wake up. In this case, the behavior of WFI wake up is just following the RISC-V standard architecture. This document will not repeat its content here, please refer to RISC-V standard privileged architecture specification for more details.
- If the value of wfe.WFE is set to 1, the core will be waited for an event to wake up. Please see the detailed description in the next section.

14.4.3. Wake Up by Event

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

- If the value of wfe.WFE is set to 1, then:
  - When the core detects that the input signal rx_evt (called the event signal) is asserted, the core will be woken up and continue to execute the previously interrupted instruction stream. Please refer to the specific datasheet of the Nuclei processor core for details about the signal rx_evt.

14.4.4. Wake Up by Debug Request

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

15. Nuclei processor core CSRs Descriptions

15.1. CSR Overview

CSR (Control and Status Registers) is part of RISC-V standard privileged architecture. Basically, Nuclei processor core are following and compatible to RISC-V standard CSR definitions, but there are some additions and enhancements to the original standard spec.

To respect the RISC-V standard, this document may not repeat the contents of original RISC-V standard, but will highlight the additions and enhancements of Nuclei defined.

15.2. Nuclei processor core CSRs List

Table 15 -14 describes the CSRs in the Nuclei processor core. In this CSRs List, there are RISC-V standard CSRs and customized CSRs in the Nuclei processor core.

Table 15-14 CSR supported in the Nuclei processor core

Type	Address	R & W	Name	Description
RISC-V Standard CSRs	These CSRs are following RISC-V standard privileged architecture specification. This document will not repeat its content here, please refer to RISC-V standard privileged architecture specification for more details.			
Nuclei Customized CSRs	0x307	MRW	mtvt	ECLIC Interrupt Vector Table Base Address
	0x320	MRW	mcountinhibit	Customized register used to control the on & off of counters
	ox345	MRW	mnxti	Used to enable taking the next interrupt and return the entry address of the next interrupt handler.
	0x346	MRO	mintstatus	Current Interrupt Levels
	0x348	MRW	mscratchcsw	Scratch swap register for privileged mode.
	0x349	MRW	mscratchcswl	Scratch swap register for interrupt levels.
	0x7c0	MRW	milm_ctl	Enable/Disable the ILM address space.
	0x7c1	MRW	mdlm_ctl	Enable/Disable the DLM address space.
	0x7c2	MRW	mecc_code	ECC code injection register, can be used to simulate ECC error
	0x7c3	MRO	mnvec	Customized register used to indicate the NMI handler entry address
	0x7c4	MRW	msubm	Customized register storing current trap type and the previous trap type before trapped.

	0x7c9	MRW	mdcause	Customized register storing current trap's detailed cause.
	0x7ca	MRW	mcache_ctl	Customized register to control the Cache features.
	0x7d0	MRW	mmisc_ctl	Customized register controlling the selection of the NMI Handler Entry Address.
	0x7d6	MRW	msavestatus	Customized register storing the value of mstatus.
	0x7d7	MRW	msaveepc1	Customized register storing the value of mepc for the first-level preempted NMI or Exception.
	0x7d8	MRW	msavecause1	Customized register storing the value of mcause for the first-level preempted NMI or Exception.
	0x7d9	MRW	msaveepc2	Customized register storing the value of mepc for the second-level preempted NMI or Exception.
	0x7da	MRW	msavecause2	Customized register storing the value of mcause for the second-level preempted NMI or Exception.
	0x7dd	MRW	mtlb_ctl	TLB control register
	0x7de	MRW	mecc_lock	To lock ECC configure registers, then all the ECC related CSRs cannot be modified, unless by reset
	0x7eb	MRW	pushmsubm	Customized register used to push the value of msubm into the stack memory.
	0x7ec	MRW	mtvt2	Customized register used to indicate the common handler entry address of non-vectored interrupts.
	0x7ed	MRW	jalmnxti	Customized register used to enable the ECLIC interrupt. The read operation of this register will take the next interrupt, return the entry address of next interrupt handler, and jump to the corresponding handler at the same time.
	0x7ee	MRW	pushmcause	Customized register used to push the value of mcause into the stack memory.
	0x7ef	MRW	pushmepc	Customized register used to push the value of mepc into the stack memory.
	0x7f0	MRO	mppicfg_info	PPI configuration information.
	0x7f1	MRO	mfiocfg_info	FIO configuration information.
	0x811	MRW	sleepvalue	Customized register used to indicate the WFI sleep mode.
	0x812	MRW	txevt	Customized register used to send an event.
	0x810	MRW	wfe	Customized register used to control the WFE mode.
	0xfc0	MRO	micfg_info	ILM and I-Cache configuration information.
	0xfc1	MRO	mdcfg_info	DLM and D-Cache configuration information.
	0xfc2	MRO	mcfg_info	Processor configuration information.

Note:

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

- URO -- User Mode Read-Only

15.3. Accessibility of CSRs in the Nuclei processor core

The CSRs accessibility in the Nuclei processor core:

- Read or write to a non-existent CSR will raise an Illegal Instruction Exception.
- Write to RO CSRs will raise an Illegal Instruction Exception.
  - Note : According to the RISC-V standard, the URO registers like cycle、cycleh、time、timeh、instret、instreth are special, the read accessibility of which are determined by the corresponding field in mcounteren.
- Access the higher privilege mode CSR raise an Illegal Instruction Exception. For example, in User Mode accessing to MRO or MRW CSRs will raise an Illegal Instruction Exception.

15.4. RISC-V Standard CSRs Supported in the Nuclei processor core

These CSRs are following RISC-V standard privileged architecture specification. This document will not repeat its content here, please refer to RISC-V standard privileged architecture specification for more details.

This chapter only introduces the RISC-V Standard CSRs supported in the Nuclei processor core, but those CSRs who also have some unique differences implemented in Nuclei processor core.

15.4.1. mie

In Nuclei processor core, the format and features of CSR mie is same as RISC-V standard, please refer to RISC-V standard privileged architecture specification for more details of CSR mie.

But note: the mie CSR is not effective when the core is in the CLIC mode, and the readout of mie is always 0, the writing is ignored.

15.4.2. mip

In Nuclei processor core, the format and features of CSR mip is same as RISC-V standard, please refer to RISC-V standard privileged architecture specification for more details of CSR mip.

But note: the mip CSR is not effective when the core is in the CLIC mode, and the readout of mip is always 0, the writing is ignored.

15.4.3. marchid

marchid is to indicate the microarchitecture ID, marchid[31:16] indicates the corresponding RTL hardware version, marchid[15:0] indicates the Core ID.

15.4.4. mimpid

marchid is to indicate the implementation ID, mimpid[31:16] indicates the corresponding databook version, mimpid[15:0] indicates the corresponding ISA Spec version.

15.4.5. mhartid

In Nuclei processor core, the format and features of CSR mhartid is same as RISC-V standard, please refer to RISC-V standard privileged architecture specification for more details of CSR mhartid.

In the Nuclei processor core, hart ID is controlled by input signal core_mhartid. Note: According to RISC-V architecture, one hart is required to have a known hart ID of 0, other harts ID can be in 1~1023.

15.4.6. mtvec

The mtvec register holds the entry address of the interrupt and exception handler. Field of mtvec register is shown in Table 15 -15.

- When mtvec holds the exception entry address:
  - The value of the address field must always be aligned on a 4-byte boundary.
- When mtvec holds the interrupt entry address:
  - When mtvec.MODE != 6’b000011, the processor uses the CLINT interrupt mode.
  - When mtvec.MODE = 6’b000011, the processor uses the CLIC interrupt

mode.

- See Section 6.13.1. for more information about non-vectored mode interrupt handler entry address.
- See Section 6.13.2. for more information about vectored mode interrupt handler entry address.

Table 15-15 mtvec register

Field	Bit	Description
ADDR	31:6	mtvec address
MODE	5:0	■ MODE field determine interrupt mode : ● 000011: CLIC interrupt mode ● Others: CLINT interrupt mode

15.4.7. mcause

The mcause is written with a code indicating the reason that caused the trap. The format and features of CSR mcause is basically same as RISC-V standard, please refer to RISC-V standard privileged architecture specification for more details. But in CLIC mode for Nuclei processor core, there some additional fields added to support CLIC mode interrupt handling.

The mcause register is formatted as shown in Table 15 -16.

Table 15-16 mcause register

Field	Bit	Description	Note
INTERRUPT	31	Current trap type: ■ 0: Exception or NMI ■ 1: Interrupt	
MINHV	30	Indicate processer is reading interrupt vector table	Note: These fields are only effect in CLIC mode. When in CLINT mode, these field is masked read as zero, write ignored.
MPP	29:28	privilege mode before interrupt	
MPIE	27	Interrupt enable before interrupt	
Reserved	26:24	Reserved 0	
MPIL	23:16	Previous interrupt level	
Reserved	15:12	Reserved 0	
EXCCODE	11:0	Exception/Interrupt Encoding	

Note:

- In CLIC mode, the mstatus.MPIE and mstatus.MPP are the mirror images of

mcause.MPIE and mcause.MPP.

- The mcause.EXCCODE of NMI can be 0x1 or 0xfff , the value is controlled by mmisc_ctl, see more detail in Section 15.5.8..

15.4.8. mcycle and mcycleh

The RISC-V architecture defines a 64-bits width cycle counter which indicates how many cycles the processor has executed. Whenever the processor is working, the counter will increase automatically.

The mcycle register records the lower 32-bits of counter and mcycleh records the higher 32-bits. The format and features of CSR mcycle/mcycleh are basically same as RISC-V standard, please refer to RISC-V standard privileged architecture specification for more details.

But in Nuclei processor core, considering the counter increases the power consumption, there is an extra bit in the customized CSR mcountinhibit that can pause the counter to save power when users don’t need to monitor the performance through the counter. See Section 15.5.1. for more information about mcountinhibit.

15.4.9. minstret and minstreth

The RISC-V architecture defines a 64-bits width counter which records how many instructions have been executed successfully.

The minstret register records the low 32-bits of counter and minstreth records the high 32-bits. The format and features of CSR minstret/minstreth are basically same as RISC-V standard, please refer to RISC-V standard privileged architecture specification for more details.

But in Nuclei processor core, considering the counter has power consumption, there is an extra bit in the customized CSR mcountinhibit that can turn off the counter to save power when users don’t need to learn the performance through the counter. See Section 15.5.1. for more information about mcountinhibit.

15.5. Nuclei processor core Customized CSR

This section introduces customized CSRs in the Nuclei processor core.

15.5.1. mcountinhibit

The mcountinhibit register controls the counting of mcycle/mcycleh and minstret/minstreth registers to save power when users don't need them. See Section 15.4.8. for more information.

Table 15-17 mcountinhibit register

Field	Bit	Description
Reserved	31:3	Reserved 0
IR	2	When IR is 1, minstret/minstreth is stop counting
Reserved	1	Reserved 0
CY	0	When CY is 1, mcycle/mcycleh is is stop counting

15.5.2. milm_ctl

The milm_ctl register controls the ILM (Instruction Local Memory) address space to enable or disable it based on user's application scenarios.

Table 15-18 milm_ctl register

Field	Bit	Description
ILM_BPA	XLEN-1:10 (Read-Only)	The base physical address of ILM. It has to be aligned to multiple of ILM size. For example, to set up an instruction local memory of size 4KB starting at address 12KB (0x3000), we simply program ILM_BPA to 0xC (take the upper 22 bits of 0x3000).
Reserved	9:4	Reserved 0
ILM_RWECC	3	Controls diagnostic accesses of ECC codes of the ILM rams through the mecc_code register. This bit can be turned on for injecting ECC errors to test the ECC handler. 0: Disable diagnostic accesses of ECC codes 1: Enable diagnostic accesses of ECC codes Notice: When ILM is not configured , this field is tied to 0.
ILM_ECC_EXCP_EN	2	ILM double bit ECC exception enable control: 0: Disable double bit ECC exception 1: Enable double bit ECC exception Notice: When ILM is not configured , this field is tied to 0.
ILM_ECC_EN	1	ILM ECC enable control: 0: Disable ECC (default) 1: Enable ECC Notice: When ILM is not configured , this field is tied to 0.
ILM_ENABLE	0 (Writeable	Instruction Local Memory enable bit. 0: ILM is disabled. 1: ILM is enabled. (default)

	in UX class cores)	

Note: ILM can only be disabled in UX class core when MMU and ILM coexist.

15.5.3. mdlm_ctl

The mdlm_ctl register controls the DLM (Data Local Memory) address space to enable or disable it based on user's application scenarios.

Table 15-19 mdlm_ctl register

Field	Bit	Description
DLM_BPA	XLEN-1:10 (Read-Only)	The base physical address of DLM. It has to be aligned to multiple of DLM size. For example, to set up a data local memory of size 4KB starting at address 12KB (0x3000), we simply program DLM_BPA to 0xC (take the upper 22 bits of 0x3000).
Reserved	9:4	Reserved 0
DLM_RWECC	3	Controls diagnostic accesses of ECC codes of the DLM rams through the mecc_code register. This bit can be turned on for injecting ECC errors to test the ECC handler. 0: Disable diagnostic accesses of ECC codes 1: Enable diagnostic accesses of ECC codes Notice: When DLM is not configured , this field is tied to 0.
DLM_ECC_EXCP_EN	2	DLM double bit ECC exception enable control: 0: Disable double bit ECC exception 1: Enable double bit ECC exception Notice: When DLM is not configured , this field is tied to 0.
DLM_ECC_EN	1	DLM ECC enable control: 0: Disable ECC (default) 1: Enable ECC Notice: When DLM is not configured , this field is tied to 0.
DLM_ENABLE	0 (Writeable in UX class cores)	Data Local Memory enable bit. 0: DLM is disabled. 1: DLM is enabled. (default)

Note: DLM can only be disabled in UX class core when MMU and DLM coexist.

15.5.4. mnvec

The Nuclei processor core customized CSR mnvec register holds the NMI entry

address.

In order to understand this register, please see Chapter 4 for more information about NMI.

During a processor running a program, the program will be forced to jump into a new PC address when an NMI is triggered. The PC address is determined by mnvec.

The value of mnvec is controlled by mmisc_ctl, see more information in Section 15.5.8..

15.5.5. msubm

The Nuclei processor core customized CSR msubm register holds the current machine sub-mode and the machine sub-mode before the current trap. See Section 3.4. for more information.

Table 15-20 msubm register

Field	Bit	Description
Reserved	31:10	Reserved 0
PTYP	9:8	Machine sub-mode before entering the trap: ■ 0 : Normal machine mode ■ 1 : Interrupt handling mode ■ 2 : Exception handling mode ■ 3 : NMI handling mode
TYP	7:6	Current machine sub-mode: ■ 0 : Normal machine mode ■ 1 : Interrupt handling mode ■ 2 : Exception handling mode ■ 3 : NMI handling mode
Reserved	5:0	Reserved 0

15.5.6. mdcause

Since there might be some exceptions share the same mcause.EXCCODE value. To further record the differences, Nuclei processor core customized CSR mdcause register to record the detailed information about the exception.

Table 15-21 mdcause register

Field	Bit	Description
Reserved	31 : 2	Reserved 0
mdcause	1:0	Further record the detailed information about the exception.

		When mcause.EXCCODE = 1 (Instruction access fault) ■ 0: Reserved ■ 1: PMP permission violation ■ 2: Bus error ■ 3: Reserved When mcause.EXCCODE = 5 (Load access fault) ■ 0: Reserved ■ 1: PMP permission violation ■ 2: Bus error ■ 3: NICE extended long pipeline instruction return error. Note: although this error ideally is nothing to do with the Load access fault, but they just shared the same mcause.EXCCODE to simplify the hardware implementation When mcause.EXCCODE = 7 (Store/AMO access fault) ■ 0: Reserved ■ 1: PMP permission violation ■ 2: Bus error ■ 3: Reserved

15.5.7. mcache_ctl

Nuclei processor core customized CSR mcache_ctl register to control the I-Cache and D-Cache features.

Table 15-22 mcache_ctl register

Field	Bit	Description
Reserved	31 : 21	Reserved 0
DC_RWDECC	20	Controls diagnostic accesses of ECC codes of the D-Cache data rams through the mecc_code register. This bit can be turned on for injecting ECC errors to test the ECC handler. ■ 0: Disable diagnostic accesses of ECC codes ■ 1: Enable diagnostic accesses of ECC codes Notice: When D-Cache is not configured , this field is tied to 0.
DC_RWTECC	19	Controls diagnostic accesses of ECC codes of the D-Cache tag rams through the mecc_code register. This bit can be turned on for injecting ECC errors to test the ECC handler. ■ 0: Disable diagnostic accesses of ECC codes ■ 1: Enable diagnostic accesses of ECC codes Notice: When D-Cache is not configured , this field is tied to 0.
DC_ECC_EXCP_EN	18	D-Cache double bit ECC exception enable control: ■ 0: Disable double bit ECC exception ■ 1: Enable double bit ECC exception Notice: When D-Cache is not configured , this field is tied to 0.

DC_ECC_EN	17	D-Cache ECC enable control: ■ 0: Disable ECC (default) ■ 1: Enable ECC Notice: When D-Cache is not configured , this field is tied to 0.
DC_EN	16	■ 0: D-Cache Disabled, default; ■ 1: D-Cache Enabled;
Reserved	15:6	Reserved 0
IC_RWDECC	5	Controls diagnostic accesses of ECC codes of the I-Cache Data rams through the mecc_code register. This bit can be turned on for injecting ECC errors to test the ECC handler. ■ 0: Disable diagnostic accesses of ECC codes ■ 1: Enable diagnostic accesses of ECC codes Notice: When I-Cache is not configured , this field is tied to 0.
IC_RWTECC	4	Controls diagnostic accesses of ECC codes of the I-Cache tag rams through the mecc_code register. This bit can be turned on for injecting ECC errors to test the ECC handler. ■ 0: Disable diagnostic accesses of ECC codes ■ 1: Enable diagnostic accesses of ECC codes Notice: When I-Cache is not configured , this field is tied to 0.
IC_ECC_EXCP_EN	3	I-Cache double bit ECC exception enable control: ■ 0: Disable double bit ECC exception ■ 1: Enable double bit ECC exception Notice: When I-Cache is not configured , this field is tied to 0.
IC_ECC_EN	2	I-Cache ECC enable control: ■ 0: Disable ECC (default) ■ 1: Enable ECC Notice: When I-Cache is not configured , this field is tied to 0.
IC_SCPD_MOD	1	Scratchpad Mode Enable : ■ 0: I-Cache as the normal mode ■ 1: I-Cache as the Scratchpad mode. When under this mode, the Data SRAM of I-Cache will be reused and downgraded to memory mapped SRAM which can be accessed by instruction fetch and load/store access, like ILM and DLM, but called as Scratchpad here. Note: this mode only take effect when IC_EN bit is disabled by software. (default)
IC_EN	0	I-Cache Enable: ■ 0: I-Cache disabled (default) ■ 1: I-Cache enabled

Note: only when the I-Cache is disabled (IC_EN bit as 0) and scratchpad mode is enabled, i.e., mcache_ctl[1:0] is 2’b10 (default value after reset), I-Cache really work as the Scratchpad.

15.5.8. mmisc_ctl

The Nuclei processor core customized CSR mmisc_ctl controls the value of mnvec and the mcause value of NMI, unaligned access enable/disable and BPU enable/disable.

Table 15-23 mmisc_ctl register

Field	Bit	Description
Reserved	31:10	Reserved 0
NMI_CAUSE_FFF	9	Control mnvec and mcause.EXCCODE of NMI: ■ 0 : The value of mnvec equals the PC address after reset, mcause.EXCCODE of NMI is 0x1. ■ 1 : The value of mnvec is the same as the value of mtvec , mcause.EXCCODE of NMI is 0xfff
Reserved	8:7	Reserved 0
UNALGN_ENABLE	6	Enable or Disable misaligned load/store access, if disabled, accessing misaligned memory locations will trig an Address Misaligned exception: ■ 0: Disable misaligned load/store access. ■ 1: Enable misaligned load/store access. Notice: This field only takes effects for load/store specified in I/F/D Extension, for load/store specified in A Extension，misaligned accesses always trig an Address Misaligned exception.
Reserved	5:4	Reserved 0
BPU_ENABLE	3	Enable or Disable the BPU Unit: ■ 0: Disable the BPU Unit. ■ 1: Enable the BPU Unit. This bit is hard-wired to 0 if branch prediction unit that cannot be disabled and enabled are not implemented.
Reserved	2:0	Reserved 0

15.5.9. msavestatus

The msavestatus holds the value of mstatus and msubm which guarantee mstatus and msubm will not be flushed by NMI or exception. The msavestatus has two-stage stack, and supports up to 3-levels NMI/Exception state save. See Section 7. for more information.

Table 15-24 msavestatus register

Field	Bit	Description
Reserved	31:16	Reserved 0
PTYP2	15:14	The trap type before taking the second-level NMI/Exception.
Reserved	13:11	Reserved 0
MPP2	10:9	The privilege mode before taking the second-level NMI/Exception.

MPIE2	8	The interrupt enable bit before taking the second-level NMI/Exception.
PTYP1	7:6	The trap type before taking the first-level NMI/Exception.
Reserved	5:3	Reserved 0
MPP1	2:1	The privilege mode before taking the second-level NMI/Exception.
MPIE1	0	The interrupt enable bit before taking the second-level NMI/Exception.

15.5.10. msaveepc1 and msaveepc2

msaveepc1 and msaveepc2 are registers of the first-level NMI/Exception status stack and the second-level NMI/Exception status stack, used to save the PC before the first-level NMI/Exception preemption and the second-level NMI/Exception preemption respectively.

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

Executing the mret instruction, and the value of mcause.INTERRUPT is 0 (Such as NMI or exception), msaveepc1 and msaveepc2 are used to restore the PC through the first and second level NMI/Exception Status Stacks.

- msaveepc2 => msaveepc1 => mepc => PC

See Section 7. for more information.

15.5.11. msavecause1 and msavecause2

msavecause1 and msavecause2 are registers of the first-level NMI/Exception status stack and the second-level NMI/Exception status stack, used to save the mcause before the first-level NMI/Exception preemption and the second-level NMI/Exception preemption respectively.

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

Executing the mret instruction, and the value of mcause.INTERRUPT is 0 (Such as NMI or exception), msavecause1 and msavecause2 are used to restore the mcause through the first and second level NMI/Exception Status Stacks.

- msavecause2 => msavecause1 => mcause

See Section 7. for more information.

15.5.12. msavedcause1and msavedcause2

msavedcause1 and msavedcause2 are registers of the first-level NMI/Exception status stack and the second-level NMI/Exception status stack, used to save the mdcause before the first-level NMI/Exception preemption and the second-level NMI/Exception preemption respectively.

- msavedcause2 <= msavedcause1 <= mdcause <= NMI/exception dcause

Executing the mret instruction, and the value of mcause.INTERRUPT is 0 (Such as NMI or exception), msavedcause1 and msavedcause2 are used to restore the mdcause through the first and second level NMI/Exception Status Stacks.

- msavedcause2 => msavedcause1 => mdcause

See Section 7. for more information.

15.5.13. mtvt

This reigister is from the CLIC draft of RISC-V fast interrupt task group.

The mtvt register holds the base address of interrupt vector table (in CLIC mode), and the base address is aligned at least 64-byte boundary. See Section 6.8. for more information.

In order to improve the performance and reduce the gate count, the alignment of the base address in mtvt is determined by the actual number of interrupts, which is shown in the following table.

Table 15-25 mtvt alignment

Max interrupt number	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

15.5.14. mnxti

This reigister is from the CLIC draft of RISC-V fast interrupt task group.

The mnxti register（Next Interrupt Handler Address and Interrupt-Enable CSR）can be used by the software to service the next interrupt when it is in the same privilege mode, without incurring the full cost of an interrupt pipeline flush and context save/restore.

The mnxti CSR is designed to be accessed using CSRRSI/CSRRCI instructions, where the value read is the next interrupt handler address and the write back updates the interrupt-enable status.

Note:

- If the next interrupt is not executed in the same privilege mode, the processor will take the next interrupt directly in a nested way, and mnxti only work when the next interrupt is in the same privilege mode.
- The mnxti CSR instruction is not the same as normal CSR instructions, the return value is different.
  - The return value of mnxti CSR read instruction is shown below:
    - For the following situations, return 0
      - No valid interrupt.
      - The highest priority interrupt is vectored.
    - When the interrupt non-vectored, return the interrupt entry address
  - The mnxti CSR write operation will update following register:
    - mnxti CSR register is a virtual register, i.e., the write operation will actually apply result on mstatus register, i.e., mstatus is the actual RMW（read-modify-write）operation target register.
    - The mcause.EXCCODE field will be updated to the value of the corresponding ID of the taken interrupt.
    - The mintstatus.MIL will be updated to current interrupt level.

15.5.15. mintstatus

This reigister is from the CLIC draft of RISC-V fast interrupt task group.

The mintstatus register holds the active interrupt level for all the privilege mode.

Table 15-26 minstatus register

Field	Bit	Description
MIL	31:24	The active interrupt level in machine mode
Reserved	23：8	Reserved 0
UIL	7:0	The active interrupt level in user mode

15.5.16. mtvt2

mtvt2 is used to indicate the entry address of the common base handler shared by all ECLIC non-vectored interrupts. See more information about mtvt2 in Section 6.13.1.1.

Table 15-27 mtvt2 register

Field	Bit	Description
CMMON-CODE-ENTRY	31:2	When mtvt2.MTVT2EN=1，this field determines the entry address of interrupt common code in ECLIC non-vector mode.
Reserved	1	Reserved 0
MTVT2EN	0	mtvt2 enable： ■ 0：the entry address of interrupt common code in ECLIC non-vector mode is determined by mtvec ■ 1：the entry address of interrupt common code in ECLIC non-vector mode is determined by mtvt2.COMMON-CODE-ENTRY

15.5.17. jalmnxti

The Nuclei processor core customized CSR jalmnxti to reduce the delay for interrupt and accelerates interrupt tail-chaining.

The jalmnxti included all functionality of mnxti, besides it also include enabling the interrupt, handling the next interrupt, jumping to the next interrupt entry address, and jumping to the interrupt handler. So, the jalmnxti can decrease the instruction numbers to speed up the interrupt handling and tail-chaining.

See more information about mtvt2 in Section 6.13.1.1.

15.5.18. pushmsubm

The Nuclei processor core customized CSR pushmsubm provides a method to store the value of msubm in memory space which base address is SP with CSR instruction csrrwi.

For example:

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

This instruction stores the value of msubm in SP+1*4 address.

15.5.19. pushmcause

The Nuclei processor core customized CSR pushmcause provides a fast method to store the value of mcause in memory space which base address is SP with CSR instruction csrrwi. See more information about mtvt2 in Section 6.13.1.1.

For example:

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

This instruction stores the value of mcause in SP+1*4 address.

15.5.20. pushmepc

The Nuclei processor core customized CSR pushmepc provides a fast method to store the value of mepc in memory space which base address is SP with CSR instruction csrrwi. See more information about mtvt2 in Section 6.13.1.1.

For example:

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

This instruction stores the value of mepc in SP+1*4 address.

15.5.21. mscratchcsw

This reigister is from the CLIC draft of RISC-V fast interrupt task group.

The mscratchcsw register is useful to swap the value between the target register and mscratch when privilege mode change.

Using a CSR read instruction to perform mscratchcsw, when the privilege mode is changed after taking an interrupt, following pseudo instruction operations are performed:

```
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
```

When the processor takes an interrupt in a lower privilege mode, the processor enters a higher privilege mode to handle the interrupt and need to store the status of processor into the stack before taking the interrupt. If the processor continues to use SP in low privilege mode, data in the higher privilege mode will be saved in the memory space which is accessible in the lower privilege mode.

RISC-V defines that when the processor is in a lower privilege mode, then data in SP of the higher privilege mode can be stored in mscratch. And in this way, the value of SP can be recovered from mscratch when the processor goes back to the higher privilege mode.

It will cost a lot of cycles to execute the program above using standard instructions, so RISC-V defines mscratchcsw register. After entering an interrupt, the processor executes an mscratchcsw CSR instruction to swap the value between mscratch and SP to recover the value of SP of the higher privilege mode. At the same time, copy the value of SP of the lower privilege to mscratch. Before the mret instruction to exit interrupt, add an mscratchcsw instruction to swap value between mscratch and SP. It will recover the SP value of the lower privilege mode and store the higher privilege mode SP to mscratch again. In this way, only two instructions are needed to solve the stack pointer (SP) switching problem of different privileged modes, which speeds up interrupt processing.

15.5.22. mscratchcswl

This reigister is from the CLIC draft of RISC-V fast interrupt task group.

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

Using the CSR instruction to read the register mscratchcsw, with unchanged privilege mode, the following register operations are performed when there is a switch between the interrupt handler and the application program:

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

```
}
// Usual use: csrrw sp，mscratchcswl，sp
```

In the same privilege mode, separating the interrupt handler task from the task space of the application task can increase robustness, reduce space usage, and facilitate system debugging. The interrupt handler has a non-zero interrupt level while the application task has a zero interrupt level. According to this feature, the RISC-V architecture defines the mscratchcswl register. Similar to mscratchcsw, adding a register instruction of mscratchcswl to the beginning and the end of the interrupt service routine enables a fast stack pointer switch between the interrupt handler and the regular application, ensuring the separation of the stack space between the interrupt handler and the regular application.

15.5.23. sleepvalue

The Nuclei processor core customized CSR sleepvalue controls different sleep modes. See Section 14.1. for more information.

Table 15-28 sleepvalue register

Field	Bit	Description
SLEEPVALUE	0	Control WFI sleep mode: ■ 0 : shallow sleep mode (After WFI, it recommends SoC to turn core_clk off) ■ 1 : deep sleep mode (After WFI, it recommends SoC to turn core_clk and core_aon_clk both off) Reset default value is 0.
Reserved	31:1	Reserved 0

15.5.24. txevt

The Nuclei processor core customized CSR txevt controls output events.

Table 15-29 txevt register

Field	Bit	Description
TXEVT	0	Event control: ■ If this bit is set as 1 , Nuclei processor core will trigger a single-cycle pulse output signal tx_evt as event signal. ■ This bit will be automatically cleared to 0 in the next cycle when it becomes 1. ■ No action, when this bit is set as 0. Reset default value is 0.

Reserved	31:1	Reserved 0

15.5.25. wfe

The Nuclei processor core customized CSR wfe Control whether the processor should be awakened by interrupt or event. See Section 14.3. for more information.

Table 15-30 wfe register

Field	Bit	Description
WFE	0	Control whether the processor should be awakened by interrupt or event. ■ 0: The processor should be awakened by interrupt and NMI in sleep mode. ■ 1: The processor should be awakened by event and NMI in sleep mode. Reset default value is 0.
Reserved	31:1	Reserved 0

15.5.26. ucode

This register is from the P-extension draft of RISC-V P-extension task group.

This register is only existed when the core have been configured to support the P extension. CSR register ucode is used to record if there is overflow happened in DSP instructions.

Table 15-31 ucode register

Field	Bit	Description
OV	0	Record if there is overflow happened in DSP instructions. If there is overflow then this field OV is set as 1, software can write 0 to this register to clear it. Reset default value is 0.
Reserved	31:1	Reserved 0

15.5.27. mcfg_info

This CSR is used to show the processor configuration information.

Table 15-32 mcfg_info register

Field	Bit	Description

Reserved	31:12	Reserved 0
TLB_ECC	11	Global configuration for TLB ECC support: ■ 0: N0 TLB ECC support. ■ 1: Has TLB ECC support.
DCACHE	10	Global configuration for D-CACHE support: ■ 0: N0 D-CACHE support. ■ 1: Has D-CACHE support.
ICACHE	9	Global configuration for I-CACHE support: ■ 0: N0 I-CACHE support. ■ 1: Has I-CACHE support.
DLM	8	Global configuration for DLM support: ■ 0: N0 DLM support. ■ 1: Has DLM support.
ILM	7	Global configuration for ILM support: ■ 0: N0 ILM support. ■ 1: Has ILM support.
NICE	6	Global configuration for NICE support: ■ 0: N0 NICE support. ■ 1: Has NICE support.
PPI	5	Global configuration for PPI support: ■ 0: N0 PPI support. ■ 1: Has PPI support.
FIO	4	Global configuration for FIO support: ■ 0: N0 FIO support. ■ 1: Has FIO support.
PLIC	3	Global configuration for PLIC support: ■ 0: N0 PLIC support. ■ 1: Has PLIC support.
CLIC	2	Global configuration for CLICsupport: ■ 0: N0 CLIC support. ■ 1: Has CLIC support.
ECC	1	Global configuration for ECC support: ■ 0: N0 ECC support. ■ 1: Has ECC support.
TEE	0	Global configuration for TEE support: ■ 0: N0 TEE support. ■ 1: Has TEE support.

15.5.28. micfg_info

This CSR is used to show the ILM and I-Cache configuration information.

Table 15-33 micfg_info register

Field	Bit	Description
Reserved	XLEN-1:23	Reserved 0
ILM_ECC	22	ECC support for the instruction local memory (ILM): ■ 0: No ECC support. ■ 1: Has ECC support.
ILM_XONLY	21	Indicates if ILM is execute-only. If ILM is execute-only, load/store instructions cannot access the ILM region:

		■ 0: ILM is not execute-only. ■ 1: ILM is execute-only.
ILM_SIZE	20:16	Indicates the size of ILM. The size has to be power of 2: ■ 0: 0 Byte ■ 1: 256 Bytes ■ 2: 512 Bytes ■ 3: 1 KiB ■ 4: 2 KiB ■ 5: 4 KiB ■ 6: 8 KiB ■ 7: 16 KiB ■ 8: 32 KiB ■ 9: 64 KiB ■ 10: 128 KiB ■ 11: 256 KiB ■ 12: 512 KiB ■ 13: 1 MiB ■ 14: 2 MiB ■ 15: 4 MiB ■ 16: 8 MiB ■ 17: 16 MiB ■ 18: 32 MiB ■ 19: 64 MiB ■ 20: 128 MiB ■ 20: 256 MiB ■ 21: 512 MiB ■ Others: Reserved
Reserved	15:10	■ Reserved 0
IC_LSIZE	9:7	I-Cache line size: ■ 0: N0 I-Cache ■ 1: 8 bytes ■ 2: 16 bytes ■ 3: 32 bytes ■ 4: 64 bytes ■ 5: 128 bytes ■ Others: Reserved
IC_WAY	6:4	I-Cache ways:: ■ 0: Direct-mapped ■ 1: 2 way ■ 2: 3 way ■ 3: 4 way ■ 4: 5 way ■ 5: 6 way ■ 6: 7 way ■ 7: 8 way
IC_SET	3:0	I-Cache sets per way: ■ 0: 8 ■ 1: 16 ■ 2: 32 ■ 3: 64 ■ 4: 128 ■ 5: 256 ■ 6: 512 ■ 7: 1024 ■ 8: 2048 ■ 9: 4096 ■ 10: 8192 ■ Others: Reserved

15.5.29. mdcfg_info

This CSR is used to show the DLM and D-Cache configuration information.

Table 15-34 mdcfg_info register

Field	Bit	Description
Reserved	XLEN-1:22	Reserved 0
DLM_ECC	21	ECC support for the data local memory (DLM): ■ 0: No ECC support. ■ 1: Has ECC support.
DLM_SIZE	20:16	Indicates the size of DLM. The size has to be power of 2: ■ 0: 0 Byte ■ 1: 256 Bytes ■ 2: 512 Bytes ■ 3: 1 KiB ■ 4: 2 KiB ■ 5: 4 KiB ■ 6: 8 KiB ■ 7: 16 KiB ■ 8: 32 KiB ■ 9: 64 KiB ■ 10: 128 KiB ■ 11: 256 KiB ■ 12: 512 KiB ■ 13: 1 MiB ■ 14: 2 MiB ■ 15: 4 MiB ■ 16: 8 MiB ■ 17: 16 MiB ■ 18: 32 MiB ■ 19: 64 MiB ■ 20: 128 MiB ■ 20: 256 MiB ■ 21: 512 MiB ■ Others: Reserved
Reserved	15:10	■ Reserved 0
DC_LSIZE	9:7	D-Cache line size: ■ 0: N0 D-Cache ■ 1: 8 bytes ■ 2: 16 bytes ■ 3: 32 bytes ■ 4: 64 bytes ■ 5: 128 bytes ■ Others: Reserved
DC_WAY	6:4	D-Cache ways:: ■ 0: Direct-mapped ■ 1: 2 way ■ 2: 3 way ■ 3: 4 way ■ 4: 5 way ■ 5: 6 way

		■ 6: 7 way ■ 7: 8 way
DC_SET	3:0	D-Cache sets per way: ■ 0: 8 ■ 1: 16 ■ 2: 32 ■ 3: 64 ■ 4: 128 ■ 5: 256 ■ 6: 512 ■ 7: 1024 ■ 8: 2048 ■ 9: 4096 ■ 10: 8192 ■ Others: Reserved

15.5.30. mtlbcfg_info

This CSR is used to show the TLB configuration information.

Table 15-35 mtlbcfg_info register

Field Name	Bits	RW	Reset Value	Description
Reserved	XLEN-1:22	RO	0	-
DTLB_SIZE	21:19	RO	IM	DTLB size: ■ 0: N0 DTLB ■ 1: 1 entry ■ 2: 2 entries ■ 3: 4 entries ■ 4: 8 entries ■ 5: 16 entries ■ 6: 32 entries ■ 7: 64 entries
ITLB_SIZE	18:16	RO	IM	ITLB size: ■ 0: N0 ITLB ■ 1: 1 entry ■ 2: 2 entries ■ 3: 4 entries ■ 4: 8 entries ■ 5: 16 entries ■ 6: 32 entries ■ 7: 64 entries
Reserved	15:11	RO	0	
MTLB _ECC	10	RO	IM	ECC support for the Main TLB: 0 : No ECC support. 1: Has ECC support.
MTLB _LSIZE	9:7	RO	IM	Main TLB line size: ■ 0: N0 Main TLB ■ 1: 1 entry ■ 2: 2 entries ■ 3: 4 entries ■ 4: 8 entries ■ 5: 16 entries ■ 6: 32 entries ■ 7: 64 entries

MTLB _WAY	6:4	RO	IM	Main TLB ways:: ■ 0: Direct-mapped ■ 1: 2 way ■ 2: 3 way ■ 3: 4 way ■ 4: 5 way ■ 5: 6 way ■ 6: 7 way 7: 8 way
MTLB_SET	3:0	RO	IM	Main TLB sets per way: ■ 0: 8 ■ 1: 16 ■ 2: 32 ■ 3: 64 ■ 4: 128 ■ 5: 256 ■ 6: 512 ■ 7: 1024 ■ 8: 2048 ■ 9: 4096 ■ 10: 8192 Others: Reserved

15.5.31. mppicfg_info

This CSR is used to show the PPI configuration information.

Table 15-36 mppicfg_info register

Field	Bit	Description
PPI_BPA	XLEN-1:10	The base physical address of PPI. It has to be aligned to multiple of PPI size. For example, to set up an instruction local memory of size 4KB starting at address 12KB (0x3000), we simply program DLM_BPA to 0xC (take the upper 22 bits of 0x3000).
Reserved	9:6	Reserved 0
PPI_SIZE	5:1	Indicates the size of PPI. The size has to be power of 2: ■ 0: Reserved ■ 1: 1 KiB ■ 2: 2 KiB ■ 3: 4 KiB ■ 4: 8 KiB ■ 5: 16 KiB ■ 6: 32 KiB ■ 7: 64 KiB ■ 8: 128 KiB ■ 9: 256 KiB ■ 10: 512 KiB ■ 11: 1 MiB ■ 12: 2 MiB ■ 13: 4 MiB ■ 14: 8 MiB ■ 15: 16 MiB

		■ 16: 32 MiB ■ 17: 64 MiB ■ 18: 128 MiB ■ 19: 256 MiB ■ 20: 512 MiB ■ 21: 1 GiB ■ 22: 2 GiB Others: Reserved
Reserved	0	Reserved 1

Note: this CSR exists only when PPI is configured.

15.5.32. mfiocfg_info

This CSR is used to show the FIO configuration information.

Table 15-37 mfiocfg_info register

Field	Bit	Description
FIO_BPA	XLEN-1:10	The base physical address of FIO. It has to be aligned to multiple of FIO size. For example, to set up an instruction local memory of size 4KB starting at address 12KB (0x3000), we simply program FIO_BPA to 0xC (take the upper 22 bits of 0x3000).
Reserved	9:6	Reserved 0
FIO_SIZE	5:1	Indicates the size of FIO. The size has to be power of 2: ■ 0: Reserved ■ 1: 1 KiB ■ 2: 2 KiB ■ 3: 4 KiB ■ 4: 8 KiB ■ 5: 16 KiB ■ 6: 32 KiB ■ 7: 64 KiB ■ 8: 128 KiB ■ 9: 256 KiB ■ 10: 512 KiB ■ 11: 1 MiB ■ 12: 2 MiB ■ 13: 4 MiB ■ 14: 8 MiB ■ 15: 16 MiB ■ 16: 32 MiB ■ 17: 64 MiB ■ 18: 128 MiB ■ 19: 256 MiB ■ 20: 512 MiB ■ 21: 1 GiB ■ 22: 2 GiB Others: Reserved
Reserved	0	Reserved 1

Note: this CSR exists only when FIO is configured.

15.5.33. mecc_lock

This CSR is used to show ECC lock info.

Table 15-38 mecc_lock register

Field Name	Bits	RW	Reset Value	Description
Reserved	XLEN-1:1	RO	0	-
ECC_LOCK	0	RW	0	To lock ECC related CSRs (*mecc_lock, mecc_code [RAMID and SRAMID excluded], ECC shield in milm_ctl, mdlm_ctl, mcache_ctl, mtlb_ctl*), then cannot be modified: 0: not locked; 1: locked;

15.5.34. mecc_code

This CSR is used to ECC code injection, can be used for ECC error simulation.

Table 15-39 mecc_code register

Field Name	Bits	RW	Reset Value	Description
Reserved	XLEN-1:29	RO	0	-
SRAMID[4:0]	28:24	RW	0	The ID of RAM that has single bit ECC errors. One of these bits is updated when an single ECC error exception occurs, and can be cleared by software. SRAMID[0]: I-Cache has a single ECC error SRAMID[1]: D-Cache has a single ECC error SRAMID[2]: Main TLB has a single ECC error SRAMID[3]: ILM ECC has a single error SRAMID[4]: DLM ECC has a single error
Reserved	23:21	RW	0	-
RAMID[4:0]	20:16	RW	0	The ID of RAM that has double bit ECC errors. One of these bits is updated when a double bit ECC error exception occurs, and can be cleared by software. RAMID[0]: I-Cache has a double bit ECC error RAMID[1]: D-Cache has a double bit ECC ECC error RAMID[2]: Main TLB has a double bit ECC ECC error RAMID[3]: ILM ECC has a double bit ECC error RAMID[4]: DLM ECC has a double bit ECC error
Reserved	[15:7] or [15:8] or [15:9]	RO	0	-
Code	[6:0] or	RW	o	ECC code for injection:

	[7:0] or [8:0]			Max(TLB Width, Data Bus Width) > 64：The width of Code is 9; Max(TLB Width, Data Bus Width) > 32：The width of Code is 8; Max(TLB Width, Data Bus Width) <= 32：The width of Code is 7;

15.5.35. mtlb_ctl

This CSR is used to show the Main TLB control info.

Table 15-40 mtlb_ctl register

Field Name	Bits	RW	Reset Value	Description
Reserved	XLEN-1:4	RO	0	-
TLB_RWD ECC	3	RW	0	Controls diagnostic accesses of ECC codes of the main TLB data rams through the mecc_code register. This bit can be turned on for injecting ECC errors to test the ECC handler. ■ 0: Disable diagnostic accesses of ECC codes ■ 1: Enable diagnostic accesses of ECC codes Notice: When MMU is not configured，this field is tied to 0.
TLB_RWT ECC	2	RW	0	Controls diagnostic accesses of ECC codes of t the main TLB tag rams through the mecc_code register. This bit can be turned on for injecting ECC errors to test the ECC handler. ■ 0: Disable diagnostic accesses of ECC codes ■ 1: Enable diagnostic accesses of ECC codes Notice: When MMU is not configured，this field is tied to 0.
TLB_ECC_ EXCP_EN	1	RW	0	Main TLB double bit ECC exception enable control: ■ 0: Disable double bit ECC exception ■ 1: Enable double bit ECC exception Notice: When MMU is not configured，this field is tied to 0.
TLB_ECC_ EN	0	RW	0	Main TLB ECC enable control: ■ 0: Disable ECC (default) ■ 1: Enable ECC Notice: When MMU is not configured，this field is tied to 0.

16. ECC Introduction

Nuclei processor core can optionally support the ECC protection on ILM, DLM, I-Cache, D-Cache and TLB, if configured.

16.1. Nuclei ECC mechanism

- SECDED: Single Error Correction, Double Error Detection
- For RV32/64, ECC protection granularity:
  - DLM and D-Cache Data-Ram: 32-bit;
  - ILM and I-Cache Data-Ram: 64-bit;
  - I/D-Cache Tag-Ram and TLB: their Actual Size;
- ECC update policy:
  - Full Write: 32/64-bit data/instruction and corresponding ECC code will be updated simultaneously;
  - Partial Write: Read-Modify-Write sequency will be triggered when 8/16-bit data write operation;
- ECC control policy:
  - ECC and ECC Exception can be enable/disable separately on ILM, DLM, I-Cache, D-Cache and TLB;
  - 1-bit ECC error will be corrected automatically by HW and will not trigger ECC exception when ECC is enabled;
  - 2-bit ECC error will trigger ECC exception only when both ECC is enabled and ECC Exception is also enabled, or 2-bit ECC error exception will not be triggered;
- 2-bit ECC error exception cases:
  - Pipeline: IFU instruction fetch will report precise exception, LSU load/store data will report imprecise exception;
  - I-Cache CCM (Control and Maintenance): for by-ADDR operation, store access fault will be reported, mecc_code.RAMID will be set to 1, mtval/stval will be cleared to 0; for INV_ALL operation, I-Cache will be invalidated directly without checking ECC error;
  - D-Cache CCM (Control and Maintenance): for by-ADDR operation, store

access fault will be reported, mecc_code.RAMID will be set to 1 (*RAMID.tlb will be set if in VA-PA translation stage, or RAMID.dcache will be set if in D-Cache accessing stage*), mtval/stval will be cleared to 0; the by-ALL operation will be abort if any cacheline has 2-bit ECC error, store access fault will be reported and mecc_code.RAMID.dcache will be set to 1, mtval/stval will be cleared to 0;

- D-Cache eviction (cpbk): imprecise exception will be reported, mecc_code.RAMID.dcache will be set to 1;
- Slave Port: 2-bit ECC error will be reported as Bus Error;
- Debug SBA: 2-bit ECC error will be reported as Bus Error;
- SFENCE: for by-ADDR operation, store access fault will be reported, mecc_code.RAMID.tlb will be set to 1 and mtval/stval will be cleared to 0; for by-ALL operation, TLB will be invalided directly without checking ECC error;
- mecc_code.RAMID will set to 1 when 2-bit ECC error occurs on ILM/DLM/I-Cache/D-Cache/TLB; However RAMID is set to 1 does not indicate that ECC exception will be triggered, such as, 2-bit ECC error occurs in Slave Port or Debug SBA, or an instruction fetching has 2-bit ECC error but be flushed later in pipeline;
- 2-bit ECC error will be indicated in mecc_code.RAMID but not in mdcause, and ECC exception will be reported as access fault but not page fault, so the access fault handler needs to check the mecc_code.RAMID for 2-bit ECC error;
- 2-bit ECC error signals will be list as output in the CORE top module for SoC integration;
- ECC exception will be reported if 2-bit ECC error occurs on any way of I-Cache/D-Cache/TLB when in Tag Ram comparing stage, even there is a hit way and this hit way has no 2-bit ECC error;
- Access fault will be reported other than page fault when TLB has 2-bit ECC error, to make this ECC exception handled in M-mode;

- 1-bit ECC error cases:
  - 1-bit ECC error will be corrected automatically by HW without triggering exception;
  - mecc_code.SRAMID will be set to 1 when 1-bit ECC error occurs on ILM/DLM/I-Cache/D-Cache/TLB;
  - 1-bit ECC error signals will be list as output in the CORE top module for

SoC integration;

- ECC error injection:
  - mecc_code.Code can be selected to update the ILM (ILM is accessible by LSU) and DLM by STORE instruction, without using the ECC code generated by HW;
  - mecc_code.Code can be selected to update the I-Cache/D-Cache Tag Ram or Data Ram by Linefill when cache miss, without using the ECC code generated by HW;
  - mecc_code.Code can be selected to update the TLB Tag Ram or Data Ram by Refill when TLB miss, without using the ECC code generated by HW;
- ECC lock:
  - ECC related CSRs cannot be modified after ECC is locked unless reset, for security;

16.2. Nuclei ECC CSRs

ECC CSRs are all Nuclei customized ones, see below:

Table 16-41 Nuclei ECC CSRs

Type	CSR ADDR	R/W	Name	Description
ECC CSRs	0xFC0	MRO	micfg_info	ILM/I-Cache configuration info
	0xFC1	MRO	mdcfg_info	DLM/D-Cache configuration info
	0xFC2	MRO	mcfg_info	Core configuration info
	0xFC3	MRO	mtlbcfg_info	TLB configuration info
	0x7C0	MRW	milm_ctl	ILM control
	0x7C1	MRW	mdlm_ctl	DLM control
	0x7C2	MRW	mecc_code	ECC code injection
	0x7DD	MRW	mtlb_ctl	TLB control
	0x7DE	MRW	mecc_lock	ECC lock
	0x7CA	MRW	mcache_ctl	Cache control

17. Performance Monitor Introduction

900 series core supports to configure Performance Monitor which is to count on various events for performance profiling.

17.1. Performance Monitor CSRs

Table 17-42 Performance Monitor CSRs list

Type	CSR Addr	RW	Name	Descriptions
CSR	0xB03	MRW	mhpmcounter3	Machine performance-monitoring counter 3.
	0xB04	MRW	mhpmcounter4	Machine performance-monitoring counter 4.
	0xB05	MRW	mhpmcounter5	Machine performance-monitoring counter 5.
	0xB06	MRW	mhpmcounter6	Machine performance-monitoring counter 6.
	0xB07	MRW	mhpmcounter7	Machine performance-monitoring counter 7(tie 0 in our implementation).
	…	…	…	…
	0xB1F	MRW	mhpmcounter31	Machine performance-monitoring counter31((tie 0 in our implementation)).
	0xB83	MRW	mhpmcounter3h	Upper 32 bits of mhpmcounter3(tie 0 in our implementation), RV32 only.
	0xB84	MRW	mhpmcounter4h	Upper 32 bits of mhpmcounter4(tie 0 in our implementation), RV32 only.
	…	…	…	…
	0xB9F	MRW	mhpmcounter31h	Upper 32 bits of mhpmcounter31(tie 0 in our implementation), RV32 only.
	0x320	MRW	mcountinhibit	Machine counter-inhibit register.
	0x323	MRW	mhpmevent3	Machine performance-monitoring event selector 3.
	0x324	MRW	mhpmevent4	Machine performance-monitoring event selector 4.
	0x325	MRW	mhpmevent5	Machine performance-monitoring event selector 5.
	0x326	MRW	mhpmevent6	Machine performance-monitoring event selector 6.
	0x327	MRW	mhpmevent7	Machine performance-monitoring event selector 7(tie 0 in our implementation).
	…	…	…	…
	0x33F	MRW	mhpmevent31	Machine performance-monitoring event selector 31(tie 0 in our implementation).
	0xC03	URO	hpmcounter3	Supervisor/User mode

				Performance-monitoring counter 3.
	0xC04	URO	hpmcounter4	Supervisor/User mode Performance-monitoring counter 4.
	…	…	…	…
	0xC1F	URO	hpmcounter31	Supervisor/User mode Performance-monitoring counter 31.
	0xC83	URO	hpmcounter3h	Upper 32 bits of hpmcounter3, RV32 only.
	0xC84	URO	hpmcounter4h	Upper 32 bits of hpmcounter4, RV32 only.
	…	…	…	…
	0xC9F	URO	hpmcounter31h	Upper 32 bits of hpmcounter31, RV32 only.

17.1.1. Mhpmcounterx

Table 17-43 mhpmcounter3-6

Field Name	Bits	RW	Reset Value	Description
Reserved	XLEN-1:32	R	0	Tie 0, RV64 only.
mhpmcounterx	31:0	RW	0	Machine performance-monitoring counter x. 3=< x <=6

Table 17-44 mhpmcounter7-31

Field Name	Bits	RW	Reset Value	Description
Reserved	XLEN-1:32	R	0	Tie 0, RV64 only.
mhpmcounterx	31:0	RW	0	Tie 0. 7=< x <=31

17.1.2. mhpmcounterhx

Table 17-45 mhpmcounterhx

Field Name	Bits	RW	Reset Value	Description
mhpmcounterhx	XLEN-1:0	R	0	Tie 0, RV32 only. 0=< x <=31

=

17.1.3. mcountinhibit

The counter-inhibit register mcountinhibit is a 32-bit WARL register that controls which of the hardware performance-monitoring counters increment. The settings in this register only control whether the counters increment; Their accessibility is not affected by the setting of this register.

When the CY, IR, or HPMn bit in the mcountinhibit register is clear, the cycle, instret, or hpmcountern register increments as usual. When the CY, IR, or HPMn bit is set, the corresponding counter does not increment.

Table 17-46 mcountinhibit

Field Name	Bits	RW	Reset Value	Description
Reserved	XLEN-1:7	R	0	-
HPM6	6	RW	0	If set, stop increase performance monitor counter 6.
HPM5	5	RW	0	If set, stop increase performance monitor counter 5.
HPM4	4	RW	0	If set, stop increase performance monitor counter 4.
HPM3	3	RW	0	If set, stop increase performance monitor counter 3.
IR	2	RW	0	If set, stop increase instret counter.
Reserved	1	R	0	-
CY	0	RW	0	If set, stop increase cycle counter.

17.1.4. mhpmeventx

The event selector CSRs, mhpmevent3–mhpmevent31, are MXLEN-bit WARL registers that control which event causes the corresponding counter to increment.

Table 17-47 mhpmevent3-6

Field Name	Bits	RW	Reset Value	Description
Reserved	XLEN-1:32	R	0	-
mevent_enable	31	RW	1	Enable the corresponding performance monitor couter increament for events in Machine Mode.
Reserved	30	R	0	-
sevent_enable	29	RW	1	Enable the corresponding performance monitor couter increament for events in Supervisor Mode.
uevent_enable	28	RW	1	Enable the corresponding performance monitor couter increament for events in User Mode.
Reserved	27:9	R	0	-
event_idx	8:4	RW	0	Detailed event selector. For instruction commit events please see Table 17 -49 for more details, for memory access events please see Error: Reference source not found for more details.
event_sel	3:0	RW	0	0: Select the instruction commit events, such as load, store, bjp ect. See Table 17 -49 for more details. 1: Select the memory access events, such as icache miss, dcache miss ect. See

				Error: Reference source not found for more details.

Table 17-48 mhpmevent7-31

Field Name	Bits	RW	Reset Value	Description
Reserved	XLEN-1:0	R	0	-

Table 17-49 Monitoring Event Selection Value Assignment for Instruction Commit Events

event_idx (mhpeventx[3:0]==0)	Event Name
1	Cycle count
2	Retired instruction count
3	Integer load instruction (includes LR)
4	Integer store instruction (includes SC)
5	Atomic memory operation (do not include LR and SC)
6	System instruction
7	Integer computational instruction(excluding multiplication/division/remainder)
8	Conditional branch
9	Taken conditional branch
10	JAL instruction
11	JALR instruction
12	Return instruction
13	Control transfer instruction (CBR+JAL+JALR)
14	-
15	Integer multiplication instruction
16	Integer division/remainder instruction
17	Floating-point load instruction
18	Floating-point store instruction
19	Floating-point addition/subtraction
20	Floating-point multiplication
21	Floating-point fused multiply-add (FMADD, FMSUB, FNMSUB, FNMADD)
22	Floating-point division or square-root
23	Other floating-point instruction
24	Conditional branch prediction fail
25	JAL prediction fail
26	JALR prediction fail

Table 17-50 Monitoring Event Selection Value Assignment for Memory Access Events

event_idx (mhpeventx[3:0]==1)	Event Name
1	Icache miss
2	Dcache miss
3	ITLB miss
4	DTLB miss
5	Main TLB miss

18. TEE Introduction

Nuclei processor core can optionally support the TEE (Trust Execution Environment) features. This document will not detail it, please refer to another document <Nuclei_TEE_Architecture> for the details. User can easily get the copy from “Nuclei User Center” website http://user.nucleisys.com.

19. Packed-SIMD DSP Introduction

Nuclei processor core can optionally support the DSP features with P extension (Packed-SIMD) instructions. This document will not detail it, please refer to another document <Nuclei_DSP_QuickStart> for the details. User can easily get the copy from “Nuclei User Center” website http://user.nucleisys.com.

20. User Extended Introduction

Nuclei processor core can optionally support user to add their custom instructions with NICE (Nuclei Instruction Co-unit Extension) interface. This document will not detail it, please refer to another document <Nuclei_NICE_Extension> for the details. User can easily get the copy from “Nuclei User Center” website http://user.nucleisys.com.