Nuclei® E603 Instruction Set Architecture Specification

Nucleisys

All rights reserved by Nucleisys, Inc.

Contents

1 Copyright Notice **1**

2 Contact Information **2**

3 Preface **6**
- 3.1 About this specification 6
- 3.2 Abbreviation Table 6
- 3.3 Permission Table 7

4 Revision History **8**

5 Introduction **9**

6 Nuclei RISC-V Instruction Set Overview **10**
- 6.1 RISC-V ISA Extensions supported by Nuclei Core 10
- 6.2 Nuclei Processor Core Classes: N, U, NX and UX 13
- 6.3 Nuclei Hummingbird E603 Core 13

7 Nuclei RISC-V Privileged Architecture **14**
- 7.1 RISC-V Privileged Architecture supported by Nuclei Core 14
- 7.2 Privileged Modes 14
- 7.3 Debug Modes 14
- 7.4 Machine Sub-Mode added by Nuclei 15

8 Exception Handling in Hummingbird E603 **16**
- 8.1 Exception Overview 16
- 8.2 Exception Masking 16
- 8.3 Priority of Exception 16
- 8.4 Entering Exception Handling Mode 16
  - 8.4.1 Update the Machine Sub-Mode 17
- 8.5 Exit the Exception Handling Mode 17
  - 8.5.1 Update the Machine Sub-Mode 18
- 8.6 Exception Service Routine 18

9 NMI Handling in Hummingbird E603 **19**
- 9.1 NMI Overview 19
- 9.2 NMI Masking 19
- 9.3 Entering NMI Handling Mode 19
  - 9.3.1 Execute from the PC Defined by mnvec 20
  - 9.3.2 Update the CSR mcause 20
  - 9.3.3 Update the Privilege Mode 20
  - 9.3.4 Update the Machine Sub-Mode 20
- 9.4 Exit the NMI Handling Mode 21
  - 9.4.1 Update the Machine Sub-Mode 22

9.5 NMI Service Routine . . . 22
9.6 NMI Nesting . . . 22

10 Interrupt Handling in Hummingbird E603 23
10.1 Interrupt Overview . . . 23
10.2 CLIC mode and CLINT mode . . . 23
10.2.1 Setting CLINT or CLIC mode . . . 23
10.2.2 CLINT mode . . . 24
10.2.3 CLIC mode . . . 24
10.3 Interrupt Type . . . 24
10.3.1 External Interrupt . . . 24
10.3.2 Internal Interrupt . . . 25
10.4 Interrupt Masking . . . 25
10.4.1 Global Interrupt Masking . . . 25
10.4.2 Individual Interrupt Masking . . . 25
10.5 Interrupt Levels, Priorities and Arbitration . . . 26
10.6 (CLIC mode) Entering Interrupt Handling Mode . . . 26
10.6.1 Execute from a new PC . . . 27
10.6.2 Update the Privilege Mode . . . 27
10.6.3 Update the Machine Sub-Mode . . . 27
10.6.4 Update the CSR mepc . . . 28
10.6.5 Update the CSRs mcause and mstatus . . . 28
10.7 (CLIC mode) Exit the Interrupt Handling Mode . . . 29
10.7.1 Executing from the Address Defined by mepc . . . 30
10.7.2 Update the CSRs mcause and mstatus . . . 30
10.7.3 Update the Privilege Mode . . . 30
10.7.4 Update the Machine Sub-Mode . . . 30
10.8 (CLIC mode) Interrupt Vector Table . . . 30
10.9 Context Saving and Restoring . . . 31
10.10 Interrupt Response Latency . . . 31
10.11 (CLIC mode) Interrupt Preemption . . . 32
10.12 (CLIC mode) Interrupt Tail-Chaining . . . 33
10.13 (CLIC mode) Vectored and Non-Vectored Processing Mode of Interrupts . . . 33
10.13.1 Non-Vectored Processing Mode . . . 34
10.13.1.1 Feature and Latency of Non-Vectored Processing Mode . . . 34
10.13.1.2 Preemption of Non-Vectored Interrupt . . . 35
10.13.1.3 Non-Vectored Interrupt Tail-Chaining . . . 36
10.13.2 Vectored Processing Mode . . . 37
10.13.2.1 Feature and Latency of Vectored Processing Mode . . . 37
10.13.2.2 Preemption of Vectored Interrupt . . . 38
10.13.2.3 Vectored Interrupt Tail-Chaining . . . 39

11 TIMER Unit Introduction 40
11.1 TIMER Overview . . . 40
11.2 TIMER Registers . . . 40
11.2.1 Time Counter Register mtime . . . 41
11.2.2 Generate the Timer Interrupt through mtime and mtimecmp . . . 41
11.2.3 Control the Timer Counter through mtimectl . . . 41
11.2.4 Generating the Software Interrupt through msip . . . 42
11.2.5 Generating the Soft-Reset Request . . . 42

12 PLIC Unit Introduction 44
12.1 PLIC Overview . . . 44
12.2 PLIC Registers . . . 44
12.2.1 Control S-mode can access plic or not through plic_srw_ctrl . . . 46

13 ECLIC Unit Introduction 47
13.1 ECLIC Overview . . . 47
13.2 ECLIC interrupt target . . . 48
13.3 ECLIC Interrupt Source . . . 48
13.4 ECLIC Interrupt Source ID . . . 49

13.5 ECLIC Registers . . . 49
13.5.1 cliccfg . . . 50
13.5.2 clicinfo . . . 50
13.5.3 mth . . . 51
13.5.4 clicintip[i] . . . 51
13.5.5 clicintie[i] . . . 51
13.5.6 clicintattr[i] . . . 51
13.5.7 clicintctl[i] . . . 52
13.6 ECLIC Interrupt Enable Bit (IE) . . . 52
13.7 ECLIC Interrupt Pending Bit (IP) . . . 52
13.8 ECLIC Interrupt Source Level or Edge-Triggered Attribute . . . 52
13.9 ECLIC Interrupt Level and Priority . . . 53
13.10 ECLIC Interrupt Vectored and Non-Vectored Processing Mode . . . 54
13.11 ECLIC Interrupt Threshold Level . . . 55
13.12 ECLIC Interrupt Arbitration Mechanism . . . 55
13.13 ECLIC Interrupt Taken, Preemption and Tail-Chaining . . . 55

14 ECLIC and PLIC Connection Diagram 56
14.1 Single-core with PLIC/ECLIC configured and PLIC enabled . . . 56
14.2 Single-core with PLIC/ECLIC configured and ECLIC enabled . . . 57

15 PMP Introduction 58
15.1 PMP Overview . . . 58
15.2 PMP Specific Features to Nuclei Core . . . 58
15.2.1 Configurable PMP Entries Number . . . 58
15.2.2 Configurable PMP Grain . . . 58
15.2.3 Support TOR mode in A field of pmpcfg<x> registers . . . 58
15.2.4 Corner cases for Boundary Crossing . . . 59

16 MMU Introduction 60
16.1 MMU Overview . . . 60
16.2 MMU Specific Features to Nuclei Core . . . 60
16.2.1 TLB Entries Number . . . 60

17 Performance Monitor Introduction 61
17.1 Performance Monitor CSRs . . . 61
17.1.1 mhpmcounterx . . . 62
17.1.2 mhpmcounterhx . . . 62
17.1.3 mcountinhibit . . . 62
17.1.4 mhpmeventx . . . 63

18 WFI/WFE Low-Power Mechanism 65
18.1 Enter the Sleep Mode . . . 65
18.2 Wait for Interrupt . . . 66
18.3 Wait for Event . . . 66
18.4 Exit the Sleep Mode . . . 66
18.4.1 Wake Up by NMI . . . 66
18.4.2 Wake Up by Interrupt . . . 66
18.4.3 Wake Up by Event . . . 67
18.4.4 Wake Up by Debug Request . . . 67

19 Hummingbird E603 CSRs Descriptions 68
19.1 CSR Overview . . . 68
19.2 Hummingbird E603 CSRs List . . . 68
19.3 Accessibility of CSRs in the Hummingbird E603 . . . 69
19.4 RISC-V Standard CSRs Supported in the Hummingbird E603 . . . 70
19.4.1 mie . . . 70
19.4.2 mip . . . 70
19.4.3 marchid . . . 70
19.4.4 mimpid . . . 70
19.4.5 mhartid . . . 70
19.4.6 mtvec . . . 71

19.4.7 mcause 71
19.4.8 mcycle and mcycleh 72
19.4.9 minstret and minstreth 72
19.5 Customrized CSRs supported in Hummingbird E603 72
19.5.1 mcountinhibit 72
19.5.2 mnvec 72
19.5.3 msubm 73
19.5.4 mdcause 73
19.5.5 mcache_ctl 74
19.5.6 mmisc_ctl 76
19.5.6.1 Core Bus Error Exception 77
19.5.6.2 Core Bus Error Interrupt 77
19.5.7 msavestatus 78
19.5.8 msaveepc1 and msaveepc2 78
19.5.9 msavecause1 and msavecause2 78
19.5.10 msavedcause1 and msavedcause2 78
19.5.11 mtvt 79
19.5.12 mnxti 79
19.5.13 mintstatus 80
19.5.14 mtvt2 80
19.5.15 jalmnxti 80
19.5.16 pushmsubm 80
19.5.17 pushmcause 81
19.5.18 pushmepc 81
19.5.19 mscratchcsw 81
19.5.20 mscratchcswl 82
19.5.21 sleepvalue 82
19.5.22 txevt 82
19.5.23 wfe 83
19.5.24 ucode 83
19.5.25 mcfg_info 83
19.5.26 micfg_info 85
19.5.27 mdcfg_info 86
19.5.28 mtlbcfg_info 88
19.5.29 mppicfg_info 89
19.5.30 mfiocfg_info 90
19.5.31 mdevb 91
19.5.32 mdevm 92
19.5.33 mnocb 92
19.5.34 mnocm 92
19.5.35 mirgb_info 93
19.5.36 mecc_lock 94
19.5.37 mecc_code 94
19.5.38 mtlb_ctl 95
19.5.39 mfp16mode 95
19.5.40 shartid 96

20 User Extended Introduction **97**

21 Appendix **98**

1 Copyright Notice

Copyright© 2018-2025 Nuclei System Technology. All rights reserved.

Nuclei®, Nucleisys®, NMSIS®, ECLIC®, are trademarks owned by Nuclei System Technology. All other trademarks used herein are the property of their respective owners.

The product described herein is subject to continuous development and improvement; information herein is given by Nuclei in good faith but without warranties.

This document is intended only to assist the reader in the use of the product. Nuclei do not assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation indirect, incidental, special, exemplary, or consequential damages. incorrect use of the product.

2 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.

List of Figures

8.1 The overall process of exception 17
8.2 The CSR mstatus and msubm updating when enter/exit the exception 17
8.3 The overall process of exiting an exception 18

9.1 The overall process of NMI 20
9.2 The CSR mstatus and msubm updating when enter/exit the NMI 21
9.3 The overall process of exiting an NMI 21

10.1 Interrupt Types 24
10.2 Arbitration among Multiple Interrupts 26
10.3 The Overall Process of Interrupt for CLIC mode 27
10.4 The CSR updating when enter/exit the Interrupt 29
10.5 The overall process of exiting an interrupt 29
10.6 Interrupt Vector Table 31
10.7 Interrupt Preemption 32
10.8 Interrupt tail-chaining 33
10.9 Example for non-vectored interrupt 35
10.10 Interrupt preemptions caused by three sequential non-vectored interrupts 36
10.11 Interrupt tail-chaining 37
10.12 Example for vectored interrupt 37
10.13 Example for vectored interrupt supported preemption 38
10.14 Interrupt preemptions caused by three sequential vectored interrupts 39

13.1 The logic structure of the ECLIC unit 47
13.2 ECLIC Connection (when ECLIC is enabled) 48
13.3 clicintctl[i] format example 53
13.4 Example for the decoding of level 54
13.5 Examples of cliccfg settings 54

14.1 Interrupt Connection (for single-core with PLIC/ECLIC configured and PLIC enabled) 56
14.2 Interrupt Connection (for single-core with PLIC/ECLIC configured and ECLIC enabled) 57

List of Tables

3.1 Abbreviation Table 6
3.2 Permission Table 7

11.1 The addresses offset of registers in the TIMER unit 40
11.2 mtimectl bit fields 41
11.3 msip bit fields 42
11.4 msftrst bit fields 42

12.1 The addresses offset of registers in the PLIC unit 44
12.2 plic_srw_ctrl bit fields 46

13.1 ECLIC interrupt sources and assignment 49
13.2 The addresses offset of registers in the ECLIC unit 49
13.3 cliccfg bit fields 50
13.4 clicinfo bit fields 50
13.5 mth bit fields 51
13.6 clicintip[i] bit fields 51
13.7 clicintie[i] bits fields 51
13.8 clicintattr[i] bits fields 51

17.1 Performance Monitor CSRs list 61
17.2 mhpmcounterx 62
17.3 mhpmcounterx 62
17.4 mhpmcounterhx 62
17.5 mcountinhibit 62
17.6 mhpmevent 3-6 63
17.7 mhpmevent7-31 63
17.8 Event Selection Value for Instruction Commit Events 63
17.9 Event Selection Value for Memory Access Events 64

19.1 Customized CSRs in the Hummingbird E603 68
19.2 mtvec register 71
19.3 mcause register 71
19.4 mcountinhibit register 72
19.5 msubm register 73
19.6 mdcause register 73
19.7 mcache_ctl register 74
19.8 mmisc_ctl register 76
19.9 Core Bus Err Type and Exception Coding Mapping 77
19.10 Core Bus Err Type and Interrupt Coding Mapping 77
19.11 msavestatus register 78
19.12 mtvt alignment 79
19.13 mintstatus register 80

19.14 mtvt2 register . . . 80
19.15 sleepvalue register . . . 82
19.16 txevt register . . . 82
19.17 wfe register . . . 83
19.18 ucode register . . . 83
19.19 mcfg_info register . . . 83
19.20 micfg_info register . . . 85
19.21 mdcfg_info register . . . 86
19.22 mtlbcfg_info register . . . 88
19.23 mppicfg_info register . . . 89
19.24 mfiocfg_info register . . . 90
19.25 mdevb register . . . 91
19.26 mdevm register . . . 92
19.27 mnocb register . . . 92
19.28 mnocm register . . . 92
19.29 mirgb_info register . . . 93
19.30 mecc_lock register . . . 94
19.31 mecc_code register . . . 94
19.32 mtlb_ctl register . . . 95
19.33 mfp16mode register . . . 95

3 Preface

3.1 About this specification

This specification describes the Nuclei Hummingbird E603 CPU ISA.

3.2 Abbreviation Table

Table 3.1: Abbreviation Table

Abbreviation	Description
ACLINT	Advanced core local interrupt
BF16	Brain floating point
CC	Cluster cache
CCM	Cache control and maintenance
CIDU	Cluster interrupt distribution unit
CSR	RISC-V Control and status register
DSP	Digital signal processing
ECC	Error checking and correction
ECLIC	Enhanced core local interrupt controller
EEW	VPU element width
FPU	Floating point unit
ICB	Nuclei internal chip bus
NICE	Nuclei instruction co-unit extension
NMI	Non-maskable interrupt
MAC	Multiply-accumulate
MMU	Memory management unit
PLIC	Platform-level interrupt controller
PMA	Physical memory attribute
PMP	Physical memory protection
RVA	RISC-V Profile extension
SIMD	Single instruction multiple data
SMP	Symmetrical multi-processing
SPMP	Supervisor physical memory protection
SMPU	Supervisor mode Memory Protection Unit
TEE	Trusted execution environment
XLCZ	Nuclei code size extension
VPU	Vector processing unit

3.3 Permission Table

Table 3.2: Permission Table

Permission	Description
MRW	Machine mode readable/writeable
MRO	Machine mode read-only
SRW	Supervisor mode readable/writeable
SRO	Supervisor mode read only
S-SRW	Secure statue Supervisor mode readable/writable
S*-SRW	Supervisor mode readable/writable when access condition meet
URW	User mode readable/writeable
URO	User mode read only

4 Revision History

Rev.	Revision Date	Revised Section	Revised Content
1.1.0	2025/8/5	N/A	1.Initial version

5 Introduction

This document describes the ISA (Instruction Set Architecture) implemented in Hummingbird E603, including the instruction set and privileged architecture features.

Basically, Hummingbird E603 is 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.

6 Nuclei RISC-V Instruction Set Overview

6.1 RISC-V ISA Extensions supported by Nuclei Core

Nuclei processor core follows the RISC-V instruction set standard (riscv-spec-20240411.pdf).

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

- Rv32E, v1.9: 32bits architecture, with 16 general purpose registers
- RV32I, v2.1: 32bits architecture, with 32 general purpose registers
- RV64I, v2.1: 64bits architecture, with 32 general purpose registers
- Zicsr, v2.0: Control and Status register Extension.
- Zicntr/Zihpm, v2.0: Counters Extension.
- Zihintntl, v1.0: Non-Temporal Locality Hints Extension.
- Zihintpause, v1.0: Pause Hint Extension.
- M, v2.0: Integer Multiplication and Division instructions.
- Zmmul, v1.0: Integer Multiplication instructions.
- C, v2.0: Compressed Instructions as 16bits Encoding to reduce code size.
- A, v2.1: Atomic Instructions.
- B, v1.0: Bit Manipulation Instructions.
- F, v2.2: Single-Precision Floating-Point Instructions.
- D, v2.2: Double-Precision Floating-Point Instructions.
- Zfh, v1.0: Half-Precision Floating-Point Instructions.
- Zfhmin, v1.0: Mini Half-Precision Floating-Point Instructions.
- Zfa, v1.0: Additional Floating-Point Instructions.
- BF16, v1.0: BF16 Format Floating-Point Instructions.
- P, v0.5.4: Packed-SIMD Instructions.
- V, v1.0: Vector Instructions.
- Zve32x, v1.0: Vector Instructions without FPU and EEW support 8,16,32.
- Zve32f, v1.0: Vector Instructions with Single FPU and EEW support 8,16,32.
- Zve64x, v1.0: Vector Instructions without FPU and EEW support 8,16,32,64
- Zve64f, v1.0: Vector Instructions with Single FPU and EEW support 8,16,32,64
- Zve64d, v1.0: Vector Instructions with Double FPU and EEW support 8,16,32,64

- Zvl128b, v1.0: VLEN is 128.
- Zvl256b, v1.0: VLEN is 256.
- Zvl512b, v1.0: VLEN is 512.
- Zvl1024b, v1.0: VLEN is 1024.
- Zvb, v1.0.0: Vector bit-manipulation Instructions.
- K, v1.0: Scalar & Entropy Source Instructions.
  - Zbkb: Bitmanip instructions for Cryptography.
  - Zbkc: Carry-less multiply instructions.
  - Zbkx: Crossbar permutation instructions.
  - Zknd: NIST Suite: AES Decryption.
  - Zkne: NIST Suite: AES Encryption.
  - Zknh: NIST Suite: Hash Function Instructions.
  - Zksed: ShangMi Suite: SM4 Block Cipher Instructions.
  - Zksh: ShangMi Suite: SM3 Hash Function Instructions.
  - Zkr: Entropy Source Extension.
  - Zkn: NIST Algorithm Suite.
  - Zks: ShangMi Algorithm Suite.
  - Zkt: Data-independent execution latency.
- Zvk, v1.0.0: Vector Scalar Instructions.
  - Zvkt: Vector data-independent execute latency.
- Zc, v1.0: Group of extensions which define subsets of the existing C extension (Zca, Zcd, Zcf) and new extensions which only contain 16-bit encodings.
- Zicond, v1.0: Integer Conditional Operations Extension.
- Zilsd/Zclsd, v1.0: Load/Store Pair for RV32.
- CMO, v1.0: Base Cache Management Operation ISA Extension.
  - Zicbom: Cache-block management instructions.
  - Zicbop: Cache-block prefetch instructions.
  - Zicboz: Cache-block zero instructions.
- Etrace, v1.0: Processor Trace.
- Sstc, v1.0.0: S-mode timer interrupt Extension.
- Svbare, v1.0: The satp mode bare must be supported.
- Sv32, v1.0: Page-based 32bit virtual-memory system.
- Sv39, v1.0: Page-based 39bit virtual-memory system.
- Sv48, v1.0: Page-based 48bit virtual-memory system.
- Svvptc, v1.0: Transitions from invalid to valid PTEs will be visible in bounded time without an explicit memory-management fence.
- Svade, v1.0: Page-fault exception are raised when a page is accessed when A bit is clear, or written when D bit is clear.
- Ssccptr, v1.0: Main memory regions with both the cacheability and coherence PMAs must support hardware page-table reads.
- Sstvecd, v1.0: stvec.MODE must be capable of holding the value 0(Direct) . When stvec.MODE=Direct, stvec.BASE must be capable of holding any valid four-byte-aligned address.

- Sstvala, v1.0: stval must be written with the faulting virtual address for load, store, and instruction page-fault, access-fault, and misaligned exceptions, and for breakpoint exceptions other than those caused by execution of the EBREAK or C.EBREAK instructions. For virtual-instruction and illegal instruction exceptions, stval must be written with the faulting instruction.
- Sscounterrenw: For any hpmcounter that is not read-only zero, the corresponding bit in scounteren must be writeable.
- Svnapot, v1.0: NAPOT Translation Extension.
- Svpbmt, v1.0: Page-Based Memory Types Extension.
- Svinval, v1.0: Fine-Grained Address-Translation Cache Invalidation Extension.
- Smepmp, v1.0: PMP Enhancements for memory access and execution prevention on Machine mode Extension.
- Smwg, v0.4: WorldGuard Extension.
- Sscofpmf, v1.0.0: Count Overflow and Mode-Based Filtering Extension.
- Smpu, v0.9.0: S-mode Memory Protection Extension.
- Smclic,Ssclic,Smclicshv,Smclicconfig, v0.9: Core-Local Interrupt Controller(CLIC) Extension.
- Advanced Core Local interrupt, v1.0: MTIMER, MSWI and SSWI.
- Platform-Level Interrupt Controller(PLIC), v1.0.0_rc5.
- RVA23 Profile, v1.0.
  - Ziccif: Main memory rgions with both the cacheability and coherence PMAs must support instruct fetch, and any instruction fetches of naturally aligned power-of-2 sizes up to min(ILEN,XLEN) are atomic.
  - Ziccrse: Main memory regions with both the cacheability and coherence PMAs must support RsrvEventual.
  - Ziccamoa: Main memory regions with both the cacheability and coherence PMAs must support all atomics in A.
  - Zicclsm: Misaligned loads and stores to main memory regions with both the cacheability and coherence PMAs must be supported.
  - Za64rs: Reservation sets are contiguous, naturally aligned, and a maximum of 64 bytes.
  - Zic64b: Cache blocks must be 64 bytes in size, naturally aligned in the address space.
  - Supm: Pointer masking, with the execution environment providing a means to select PMLEN=0 and PMLEN=7 at minimum.
  - Ssnpm: Pointer masking with senvcfg.PME and henvcfg.PME supporting,at minimum, settings PMLEN=0 and PMLEN=7.
  - Sha: The augmented hypervisor extension
    - H: The hypervisor extension.
    - Ssstateen: Supervisor-mode view of the state-enable extension. The supervisor-mode(sstateen0-3) and hypervisor-mode(hstateen0-3) state-enable registers must be provided.
    - Shcounterenw: For any hpmcounter that is not read-only zero, the corresponding bit in hcounteren must be writeable.
    - Shvstvals: vstval must be written in all cases described above for stval.
    - Shtvala: htval must be written with the faulting guest physical address in all circumstances permitted by the ISA.
    - Shvstvecd: vstvec.MODE must be capable of holding the value 0(Direct). When vstvec.MODE=Direct, vstvec.BASE must be capable of holding any valid four-byte-aligned address.
    - Shvsatpa: All translation modes supported in satp must be supported in vsatp.
    - Shgatpa: For each supported virtual memory scheme SvNN supported in satp, the corresponding hgatp SvNNx4 mode must be supported. The hgatp mode Bare must also be supported.
- RVB23 Profile, v1.0.

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.

6.2 Nuclei Processor Core Classes: N, U, NX and UX

To differentiate the IP products with different positions, Nuclei divides the core products into 4 classes:

- Nuclei N class: support RV32I or RV32E, for the 32bits microcontroller applications.
  - Nuclei N200 series core can be configured to support RV32E or RV32I.
  - Nuclei N300, N600, N900 series core only support RV32I.
- Nuclei U class: support RV32I with MMU, for the 32bits Linux capable applications.
  - The MMU can also be turned off by software, in this mode, U class core can also work as microcontroller, i.e., Nuclei U class core is downward-compatible to Nuclei N class core.
- 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.

6.3 Nuclei Hummingbird E603 Core

The Nuclei Hummingbird E603 processor core features fixed support for the RV64IMAFDC instruction set.

7 Nuclei RISC-V Privileged Architecture

7.1 RISC-V Privileged Architecture supported by Nuclei Core

The Hummingbird E603 processor core follows the RISC-V privileged architecture standard (riscv-privileged-20240411.pdf).

Basically, the Hummingbird E603 processor core is compatible with the RISC-V standard privileged architecture, but there might be some additions and enhancements to the original standard spec, such as the Nuclei-defined custom instruction interface (NICE).

To respect the RISC-V standard, this document may not repeat the contents of the original RISC-V standard, but will highlight the additions and enhancements defined by Nuclei for the E603 core.

7.2 Privileged Modes

Following the RISC-V privileged architecture standard, the Hummingbird E603 processor core supports the 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 the current privilege mode (e.g., machine mode or user mode) via a standard CSR.

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

7.3 Debug Modes

Hummingbird E603 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.

7.4 Machine Sub-Mode added by Nuclei

Besides the above-mentioned standard Privilege Modes, Hummingbird E603 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 *Exception Handling in Hummingbird E603* (page 16) for more details.
- NMI Handling Mode
  - The processor core will be under this sub-mode when the core encountered NMI trap. Please refer to NMI_Handling_in_Nuclei_processor_core for more details.
- Interrupt Handling Mode
  - The processor core will be under this sub-mode when the core encountered interrupt trap. Please refer to *Interrupt Handling in Hummingbird E603* (page 23) 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 *msubm* (page 73) for more details of CSR register msubm.

8

Exception Handling in Hummingbird E603

8.1 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.

8.2 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.

8.3 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.

8.4 Entering Exception Handling Mode

Taking an exception, hardware behaviors of the Hummingbird E603 are shown in *The overall process of exception* (page 17). 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 Hummingbird E603, and will be detailed in the following section.

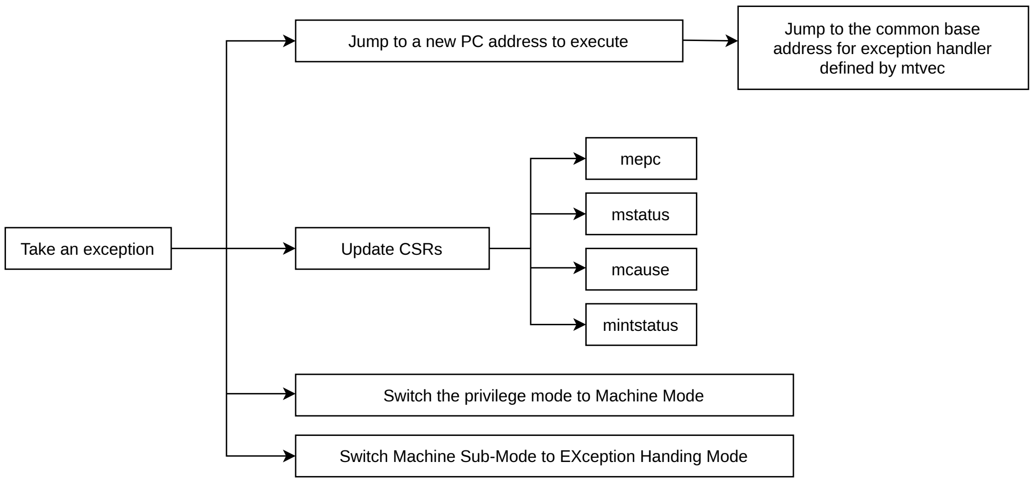

Take an exception
Jump to a new PC address to execute
Jump to the common base address for exception handler defined by mtvec
Update CSRs
mepc
mstatus
mcause
mintstatus
Switch the privilege mode to Machine Mode
Switch Machine Sub-Mode to EXception Handing Mode

Fig. 8.1: The overall process of exception

8.4.1 Update the Machine Sub-Mode

The Machine Sub-Mode of the Hummingbird E603 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 *The CSR mstatus and msubm updating when enter/exit the exception* (page 17), 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 *The CSR mstatus and msubm updating when enter/exit the exception* (page 17). The value of msubm.PTYP will be used to restore the value of msubm.PTYP after exiting the exception handler.

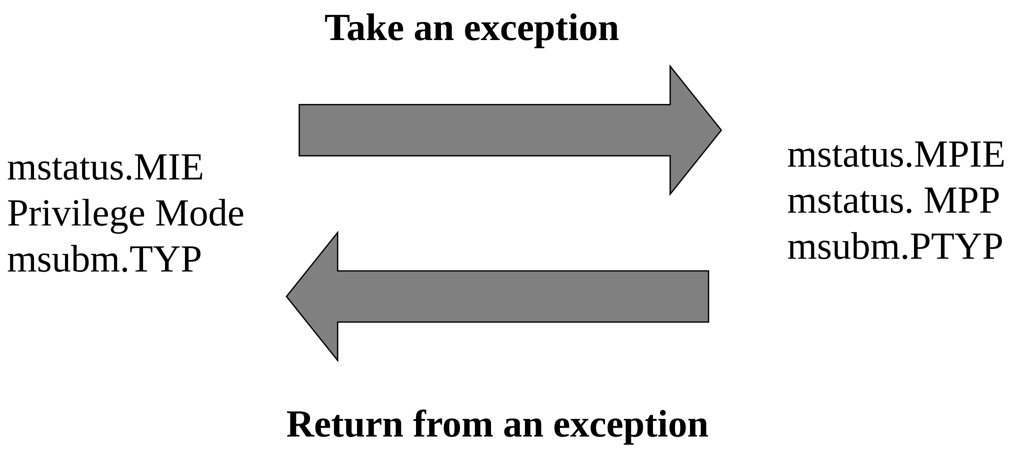


Fig. 8.2: The CSR mstatus and msubm updating when enter/exit the exception

8.5 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 *The overall process of exiting an exception* (page 18). 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 Hummingbird E603, and will be detailed in the following section.

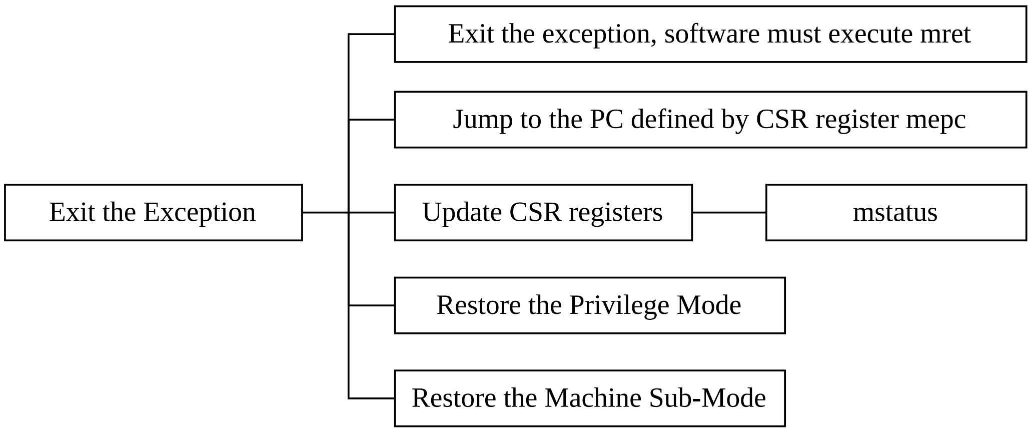

Exit the Exception
Exit the exception, software must execute mret
Jump to the PC defined by CSR register mepc
Update CSR registers
mstatus
Restore the Privilege Mode
Restore the Machine Sub-Mode

Fig. 8.3: The overall process of exiting an exception

8.5.1 Update the Machine Sub-Mode

The value of msubm.TYP indicates the Machine Sub-Mode of the Hummingbird E603 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 *The CSR mstatus and msubm updating when enter/exit the exception* (page 17). Through this mechanism, the Machine Sub-Mode of the core is restored to the same mode before taking the exception.

8.6 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.

9 NMI Handling in Hummingbird E603

9.1 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.

9.2 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.

9.3 Entering NMI Handling Mode

Taking an NMI, hardware behaviors of the Hummingbird E603 are described as shown in fig_The_overall_process_of_NMI. 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 Hummingbird E603, 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 Hummingbird E603, and will be detailed in the following section.
- Update the Privilege Mode and Machine Sub-Mode of the core.
  - This is unique in Hummingbird E603, and will be detailed in the following section.

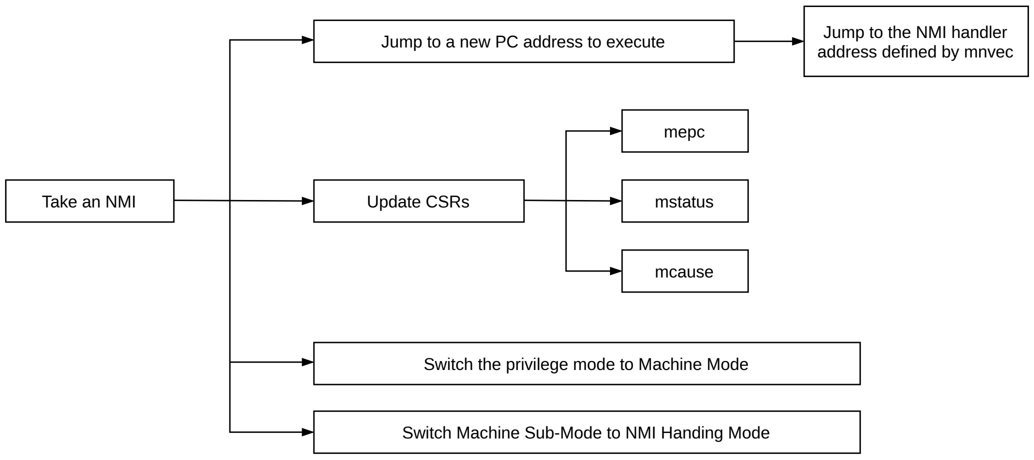

Take an NMI
Jump to a new PC address to execute
Jump to the NMI handler address defined by mnvec
Update CSRs
mepc
mstatus
mcause
Switch the privilege mode to Machine Mode
Switch Machine Sub-Mode to NMI Handing Mode

Fig. 9.1: The overall process of NMI

9.3.1 Execute from the PC Defined by mnvec

The Hummingbird E603 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 Hummingbird E603 for details about this signal.

9.3.2 Update the CSR mcause

The Hummingbird E603 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.

9.3.3 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.

9.3.4 Update the Machine Sub-Mode

The Machine Sub-Mode of the Hummingbird E603 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 fig_The_CSR_mstatus_and_msubm_updating_when_enter_exit_the_NMI, 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 fig_The_CSR_mstatus_and_msubm_updating_when_enter_exit_the_NMI. The value of msubm.PTYP will be used to restore the value of msubm.PTYP after exiting the NMI handler.

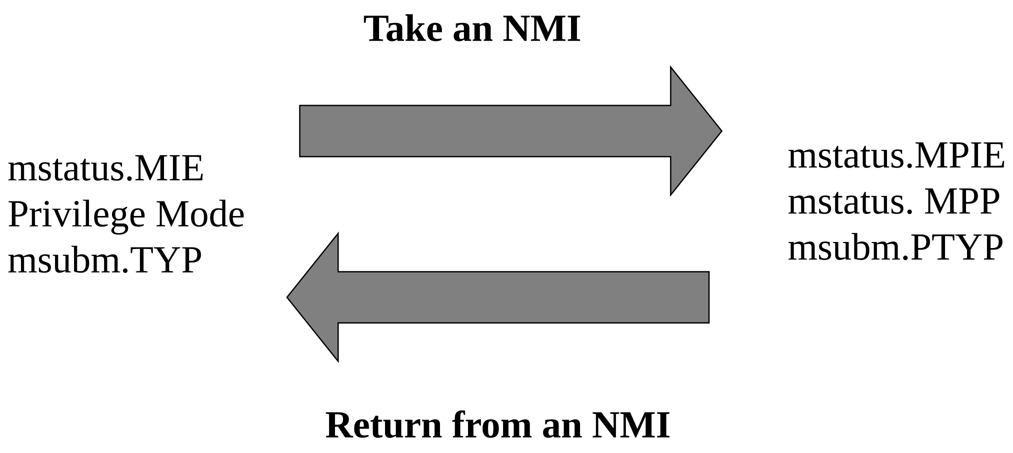


Fig. 9.2: The CSR mstatus and msubm updating when enter/exit the NMI

9.4 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 fig_The_overall_process_of_exiting_an_NMI. 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 Hummingbird E603, and will be detailed in the following section.

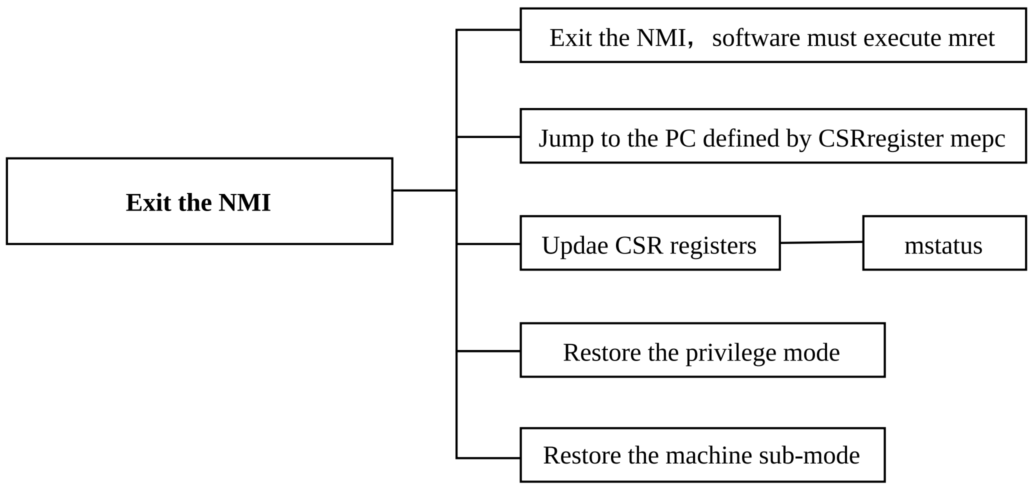

Exit the NMI
Exit the NMI， software must execute mret
Jump to the PC defined by CSRregister mepc
Updae CSR registers
mstatus
Restore the privilege mode
Restore the machine sub-mode

Fig. 9.3: The overall process of exiting an NMI

9.4.1 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 fig_The_CSR_mstatus_and_msubm_updating_when_enter_exit_the_NMI. Through this mechanism, the Machine Sub-Mode of the core is restored to the same mode before taking the NMI.

9.5 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.

9.6 NMI Nesting

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

10 Interrupt Handling in Hummingbird E603

10.1 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 source 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".

10.2 CLIC mode and CLINT mode

10.2.1 Setting CLINT or CLIC mode

The Hummingbird E603 supports the "CLINT 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 *mtvec* (page 71) for more details.

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

10.2.2 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.

10.2.3 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 *ECLIC Unit Introduction* (page 47) for more details.

10.3 Interrupt Type

The types of interrupts supported by the Hummingbird E603 are shown in *Interrupt Types* (page 24).

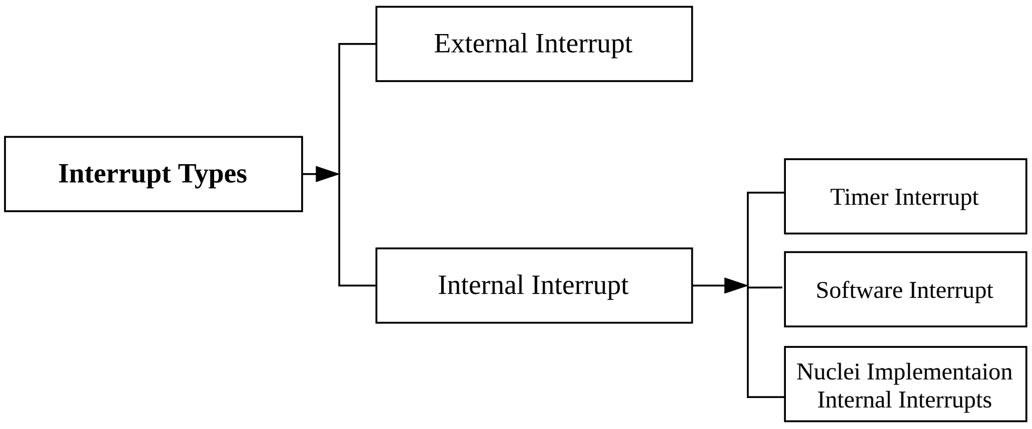


Fig. 10.1: Interrupt Types

These will be detailed in the following sections.

10.3.1 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 Hummingbird E603 supports multiple external interrupt sources.

Note:

- In CLINT mode, all of external interrupts are managed by the PLIC, as depicted in Single-core_with_PLIC_ECLIC_configured_and_PLIC_enabled.
- In CLIC mode, all of external interrupts are managed by the ECLIC, as depicted in *Single-core with PLIC/ECLIC configured and ECLIC enabled* (page 57).

10.3.2 Internal Interrupt

The Hummingbird E603 has several core-internal private interrupts as the followings:

- Software Interrupt
  - The Hummingbird E603 implements a TIMER unit, and an msip register is defined in the TIMER unit, through which software interrupts can be generated. Please see *Generating the Software Interrupt through msip* (page 42) for details.
- Timer Interrupt
  - The Hummingbird E603 implements a TIMER unit, and a counter is defined in the TIMER unit, through which time interrupts can be generated. Please see *Generate the Timer Interrupt through mtime and mtimecmp* (page 41) for details.
- Nuclei Implementation Internal Interrupts
  - Some Hummingbird E603s implement more Internal Interrupts, the later sections of this document show more details.

Note:

- In CLINT mode, the internal interrupts of the Hummingbird E603 are managed by CSR register mie and mip, as depicted in *mie* (page 70) and *mip* (page 70).
- In CLIC mode, the internal interrupts of the Hummingbird E603 are also managed by the ECLIC, as depicted in *Single-core with PLIC/ECLIC configured and ECLIC enabled* (page 57).

10.4 Interrupt Masking

10.4.1 Global Interrupt Masking

Interrupts in machine mode can be masked globally by the control bit of CSR mstatus.MIE in Hummingbird E603. Please refer to RISC-V standard privileged architecture specification for more details.

10.4.2 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. If there are other implementation internal interrupts, the CSR mie related bits will also be implemented. 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 *PLIC Registers* (page 44) 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 *ECLIC Registers* (page 49) for details.

10.5 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 *PLIC Registers* (page 44) 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 *ECLIC Registers* (page 49) for more details.

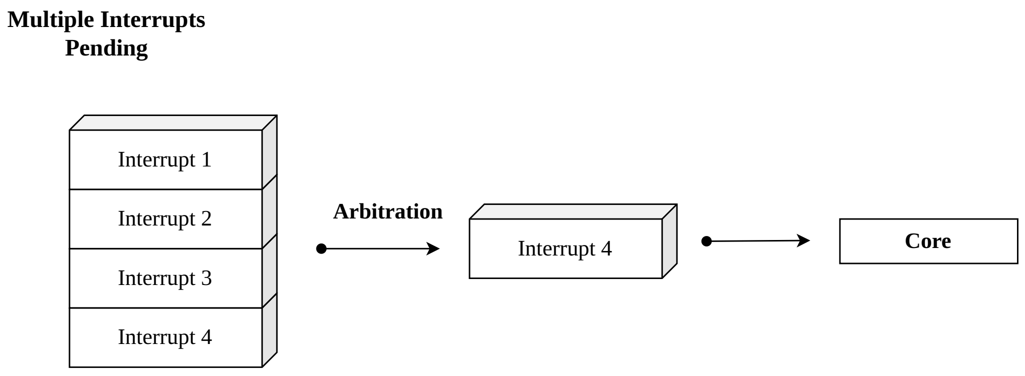


Fig. 10.2: Arbitration among Multiple Interrupts

10.6 (CLIC mode) Entering Interrupt Handling Mode

If it is in CLINT mode, taking an interrupt, hardware behaviors of the Hummingbird E603 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 Hummingbird E603 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 *The Overall Process of Interrupt for CLIC mode* (page 27).

These will be detailed in the following sections.

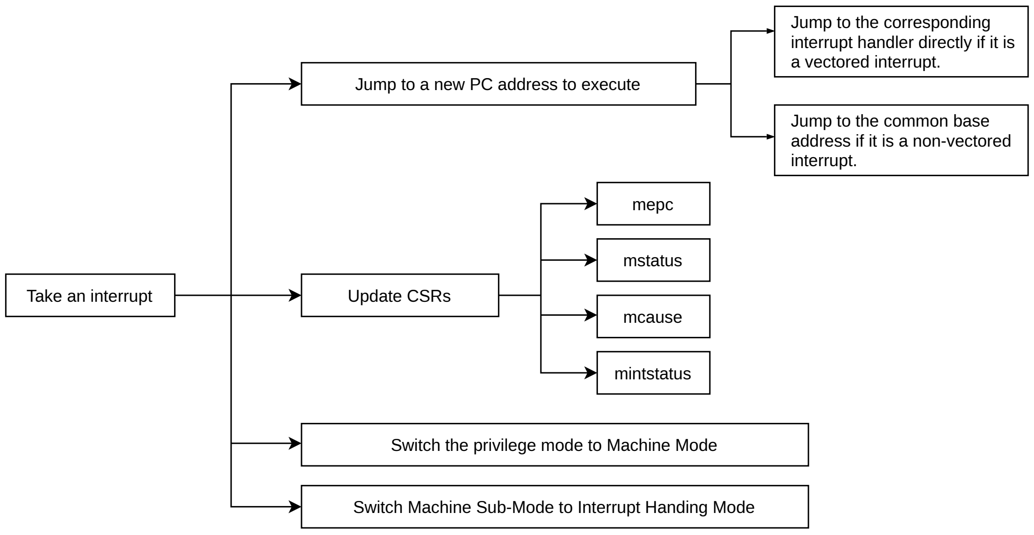

Take an interrupt
Jump to a new PC address to execute
Jump to the corresponding interrupt handler directly if it is a vectored interrupt.
Jump to the common base address if it is a non-vectored interrupt.
Update CSRs
mepc
mstatus
mcause
mintstatus
Switch the privilege mode to Machine Mode
Switch Machine Sub-Mode to Interrupt Handing Mode

Fig. 10.3: The Overall Process of Interrupt for CLIC mode

10.6.1 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 *(CLIC mode) Interrupt Vector Table* (page 30). For details of the vectored processing mode, please refer to *Vectored Processing Mode* (page 37).
- 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 *Non-Vectored Processing Mode* (page 34).

10.6.2 Update the Privilege Mode

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

10.6.3 Update the Machine Sub-Mode

The Machine Sub-Mode of the Hummingbird E603 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 6-4. The value of msubm.PTYP will be used to restore the value of msub.TYP after exiting the interrupt handler.
- The filed msubm.TYP is updated to interrupt handling mode, as described in Figure 6-4, to reflect the current Machine Sub-Mode is “interrupt handling mode”.

10.6.4 Update the CSR mepc

The return address when the Hummingbird E603 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.

10.6.5 Update the CSRs mcause and mstatus

The Hummingbird E603 will update the CSR mcause by the hardware automatically, as described in *The CSR updating when enter/exit the Interrupt* (page 29), explained as follows:

- A mechanism is required to record the ID of the interrupt being taken.
  - When an interrupt is taken by the Hummingbird E603, 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 Hummingbird E603 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 Hummingbird E603 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 Hummingbird E603, the field mcause.MPIL is updated to the value of minstatus.MIL. The value of mcause.MPIL is used to restore the value of mintcause.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 *Vectored Processing Mode* (page 37). 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 Hummingbird E603 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.

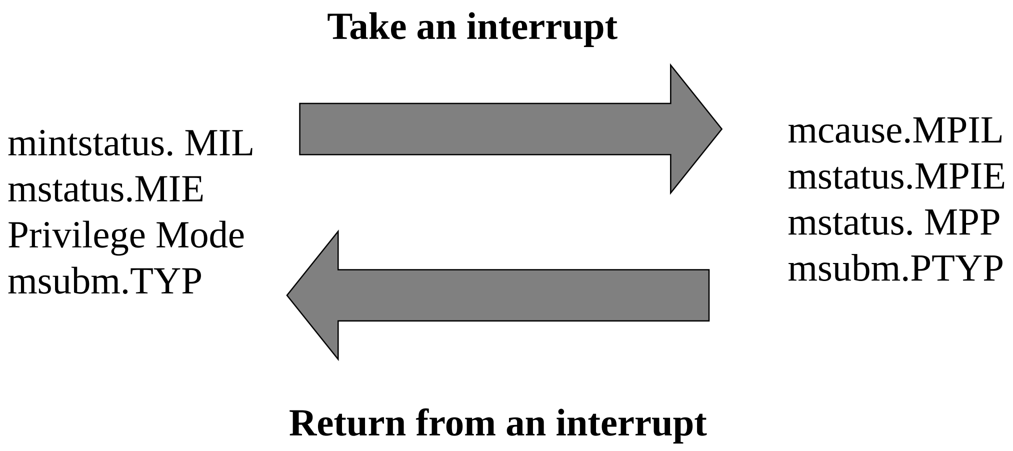


Fig. 10.4: The CSR updating when enter/exit the Interrupt

10.7 (CLIC mode) Exit the Interrupt Handling Mode

If it is in CLINT mode, after handling the interrupt, hardware behaviors of the Hummingbird E603 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 must execute mret to exit the interrupt handler. The hardware behavior of the processor after executing mret instruction is as depicted in *The overall process of exiting an interrupt* (page 29). 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.

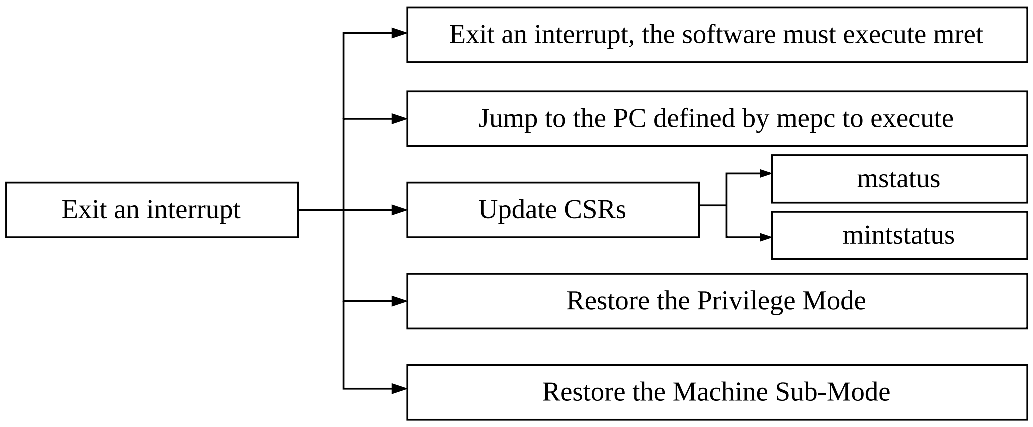

Exit an interrupt
Exit an interrupt, the software must execute mret
Jump to the PC defined by mepc to execute
Update CSRs
mstatus
mintstatus
Restore the Privilege Mode
Restore the Machine Sub-Mode

Fig. 10.5: The overall process of exiting an interrupt

These will be detailed in the following sections.

10.7.1 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.

10.7.2 Update the CSRs mcause and mstatus

The Hummingbird E603 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.

10.7.3 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.

10.7.4 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.

10.8 (CLIC mode) Interrupt Vector Table

If in CLINT mode, Hummingbird E603 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 *Interrupt Vector Table* (page 31), 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 *(CLIC mode) Vectored and Non-Vectored Processing Mode of Interrupts* (page 33) for more details.

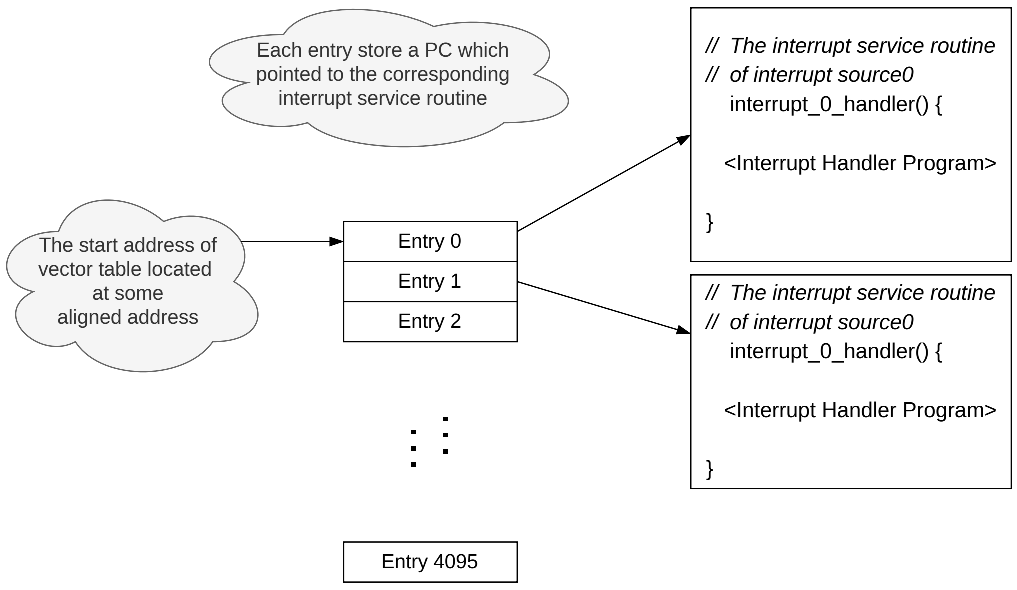

Each entry store a PC which pointed to the corresponding interrupt service routine
The start address of vector table located at some aligned address
Entry 0
Entry 1
Entry 2
Entry 4095
// The interrupt service routine
// of interrupt source0
interrupt_0_handler() {
<Interrupt Handler Program>
}
// The interrupt service routine
// of interrupt source0
interrupt_0_handler() {
<Interrupt Handler Program>
}

Fig. 10.6: Interrupt Vector Table

10.9 Context Saving and Restoring

Hummingbird E603 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 *(CLIC mode) Vectored and Non-Vectored Processing Mode of Interrupts* (page 33) for more details.

10.10 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 *(CLIC mode) Vectored and Non-Vectored Processing Mode of Interrupts* (page 33) for more details.

10.11 (CLIC mode) Interrupt Preemption

If in CLINT mode, Hummingbird E603 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 *Interrupt Preemption* (page 32) 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.

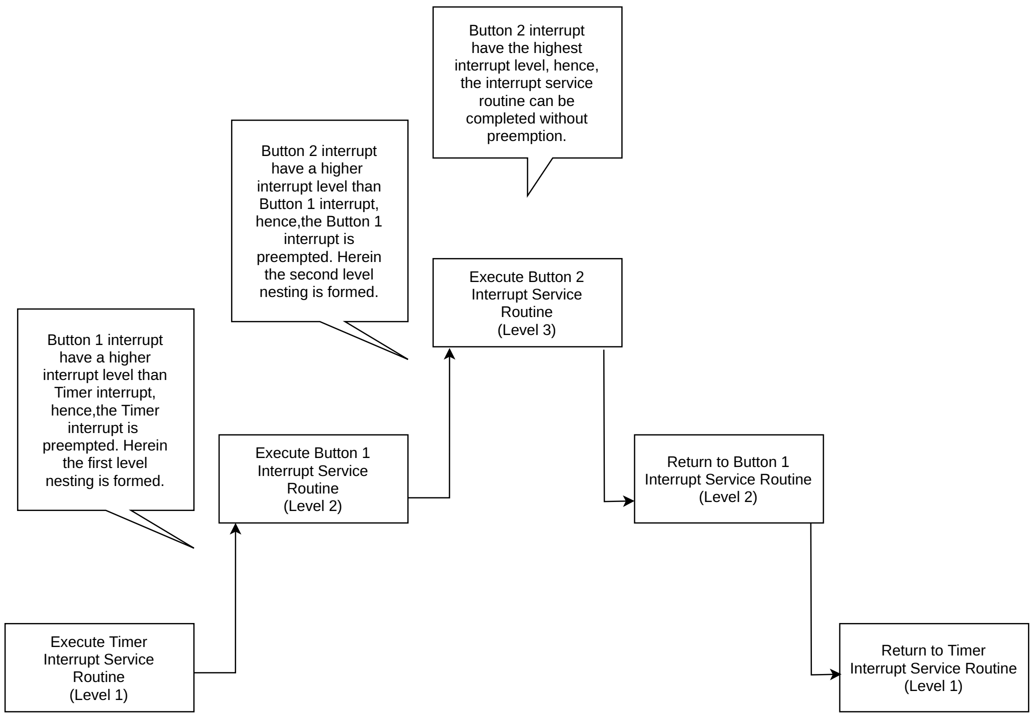

Button 2 interrupt have the highest interrupt level, hence, the interrupt service routine can be completed without preemption.
Button 2 interrupt have a higher interrupt level than Button 1 interrupt, hence,the Button 1 interrupt is preempted. Herein the second level nesting is formed.
Execute Button 2 Interrupt Service Routine (Level 3)
Button 1 interrupt have a higher interrupt level than Timer interrupt, hence,the Timer interrupt is preempted. Herein the first level nesting is formed.
Execute Button 1 Interrupt Service Routine (Level 2)
Return to Button 1 Interrupt Service Routine (Level 2)
Execute Timer Interrupt Service Routine (Level 1)
Return to Timer Interrupt Service Routine (Level 1)

Fig. 10.7: Interrupt Preemption

In the Hummingbird E603, the supported methods for interrupt preemption depending on whether the interrupt is a vectored interrupt or a non-vectored interrupt. Please see *(CLIC mode) Vectored and Non-Vectored Processing Mode of Interrupts* (page 33) for more details.

10.12 (CLIC mode) Interrupt Tail-Chaining

If in CLINT mode, Hummingbird E603 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 *Interrupt tail-chaining* (page 33).

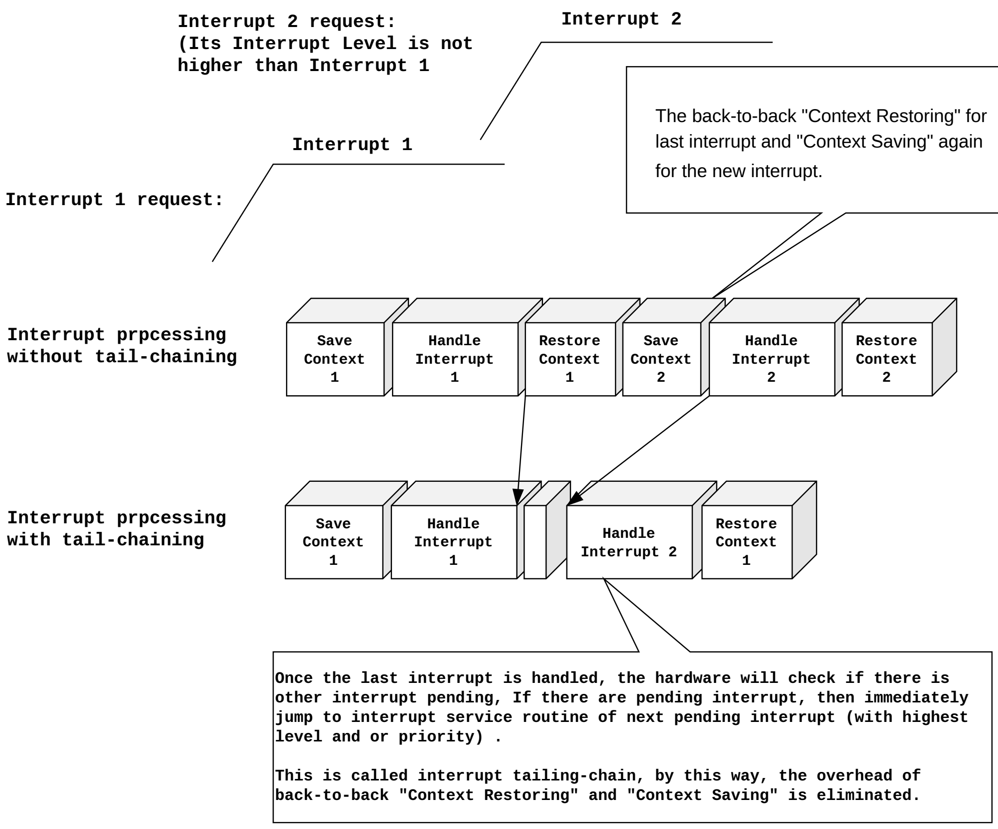

Interrupt 2 request:
(Its Interrupt Level is not
higher than Interrupt 1
Interrupt 2
The back-to-back "Context Restoring" for last interrupt and "Context Saving" again for the new interrupt.
Interrupt 1
Interrupt 1 request:
Interrupt prpcessing without tail-chaining
Save Context 1
Handle Interrupt 1
Restore Context 1
Save Context 2
Handle Interrupt 2
Restore Context 2
Interrupt prpcessing with tail-chaining
Save Context 1
Handle Interrupt 1
Handle Interrupt 2
Restore Context 1
Once the last interrupt is handled, the hardware will check if there is other interrupt pending, If there are pending interrupt, then immediately jump to interrupt service routine of next pending interrupt (with highest level and or priority) .
This is called interrupt tailing-chain, by this way, the overhead of back-to-back "Context Restoring" and "Context Saving" is eliminated.

Fig. 10.8: Interrupt tail-chaining

As for the Hummingbird E603, only non-vectored interrupts (CLIC mode) support the feature of tail-chaining. Please see *Non-Vectored Interrupt Tail-Chaining* (page 36) for more details.

10.13 (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.

10.13.1 Non-Vectored Processing Mode

10.13.1.1 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 6 bits, please refer *mtvec* (page 71) for details). 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 *Example for non-vectored interrupt* (page 35), 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 *Non-Vectored Interrupt Tail-Chaining* (page 36).
- 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.

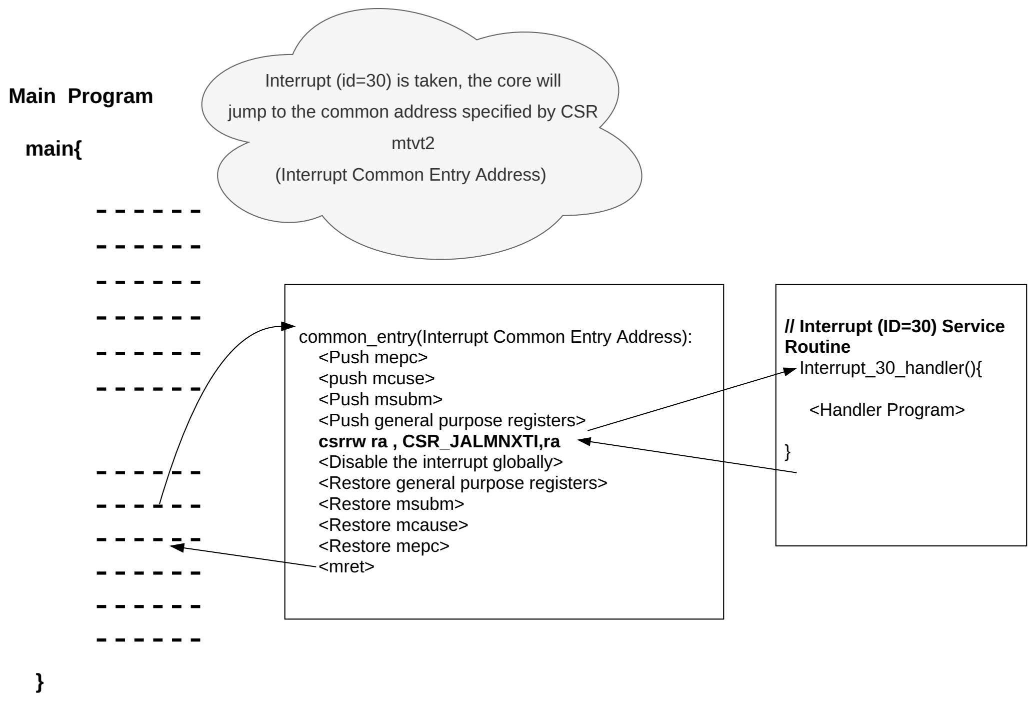

Main Program
main{
Interrupt (id=30) is taken, the core will jump to the common address specified by CSR mtvt2 (Interrupt Common Entry Address)
common_entry(Interrupt Common Entry Address):
<Push mepc>
<push mcuse>
<Push msubm>
<Push general purpose registers>
csrrw ra , CSR_JALMNXTI,ra
<Disable the interrupt globally>
<Restore general purpose registers>
<Restore msubm>
<Restore mcause>
<Restore mepc>
<mret>
// Interrupt (ID=30) Service Routine
Interrupt_30_handler(){
<Handler Program>
}
}

Fig. 10.9: 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.

10.13.1.2 Preemption of Non-Vectored Interrupt

As mentioned above, non-vectored interrupt processing mode can always support interrupt preemption as the example shown in *Interrupt preemptions caused by three sequential non-vectored interrupts* (page 36): 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.

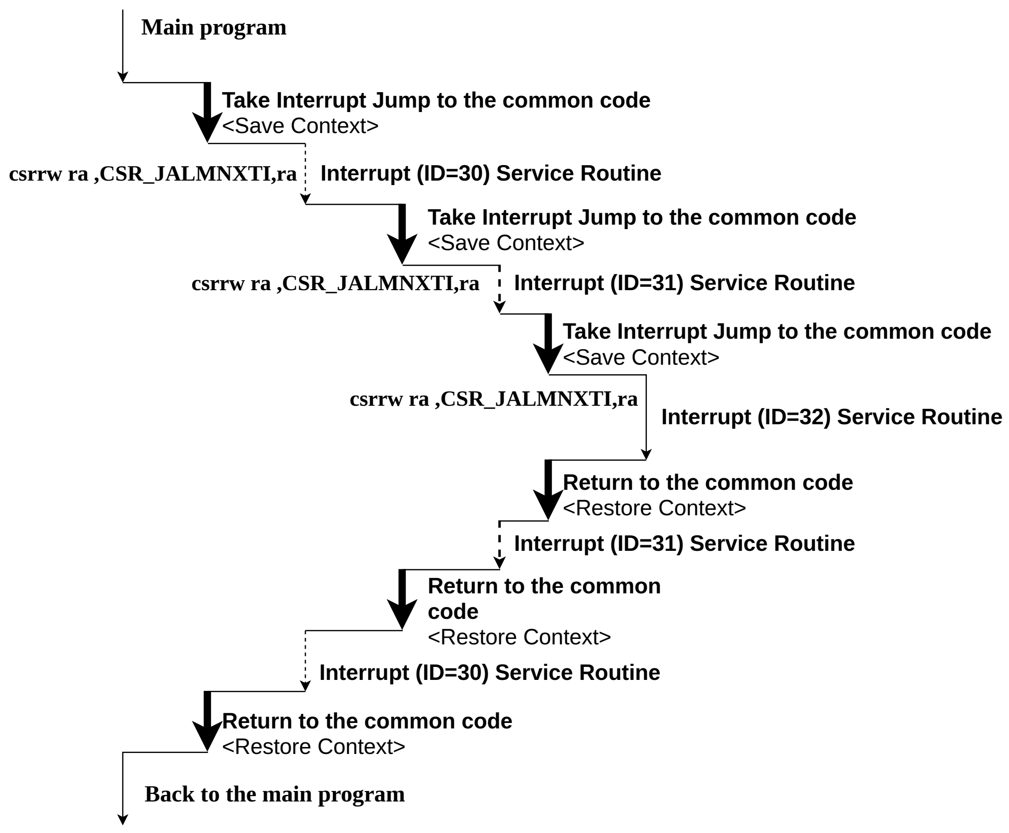

Main program
Take Interrupt Jump to the common code
<Save Context>
csrrw ra ,CSR_JALMNXTI,ra
Interrupt (ID=30) Service Routine
Take Interrupt Jump to the common code
<Save Context>
csrrw ra ,CSR_JALMNXTI,ra
Interrupt (ID=31) Service Routine
Take Interrupt Jump to the common code
<Save Context>
csrrw ra ,CSR_JALMNXTI,ra
Interrupt (ID=32) Service Routine
Return to the common code
<Restore Context>
Interrupt (ID=31) Service Routine
Return to the common code
<Restore Context>
Interrupt (ID=30) Service Routine
Return to the common code
<Restore Context>
Back to the main program

Fig. 10.10: Interrupt preemptions caused by three sequential non-vectored interrupts

10.13.1.3 Non-Vectored Interrupt Tail-Chaining

As mentioned in *(CLIC mode) Interrupt Tail-Chaining* (page 33), 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 *Feature and Latency of Non-Vectored Processing Mode* (page 34), 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 *Interrupt tail-chaining* (page 37): 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.

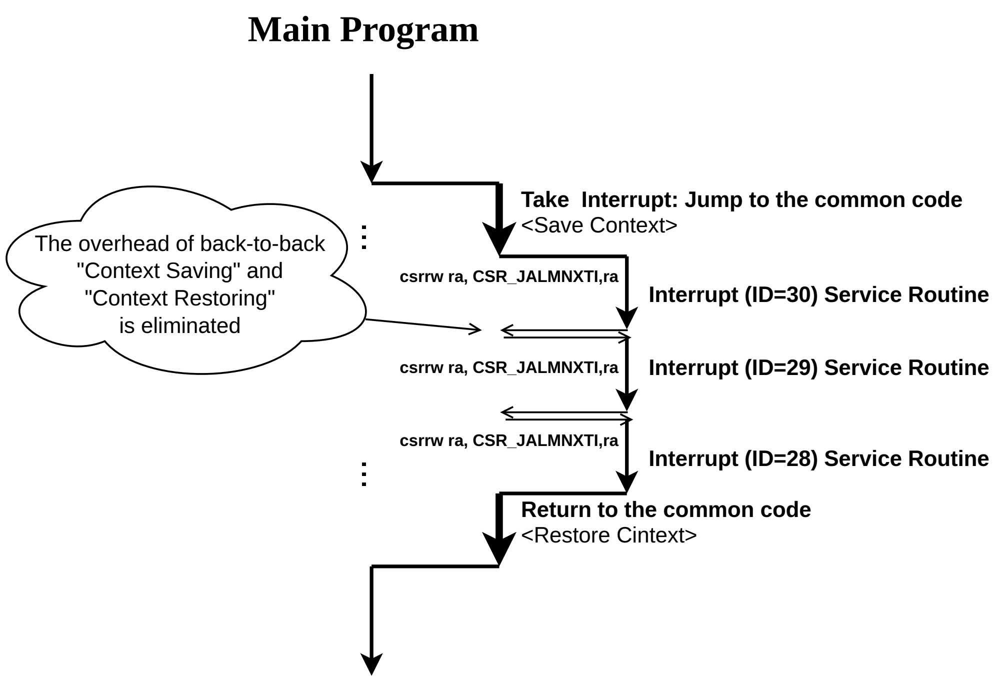
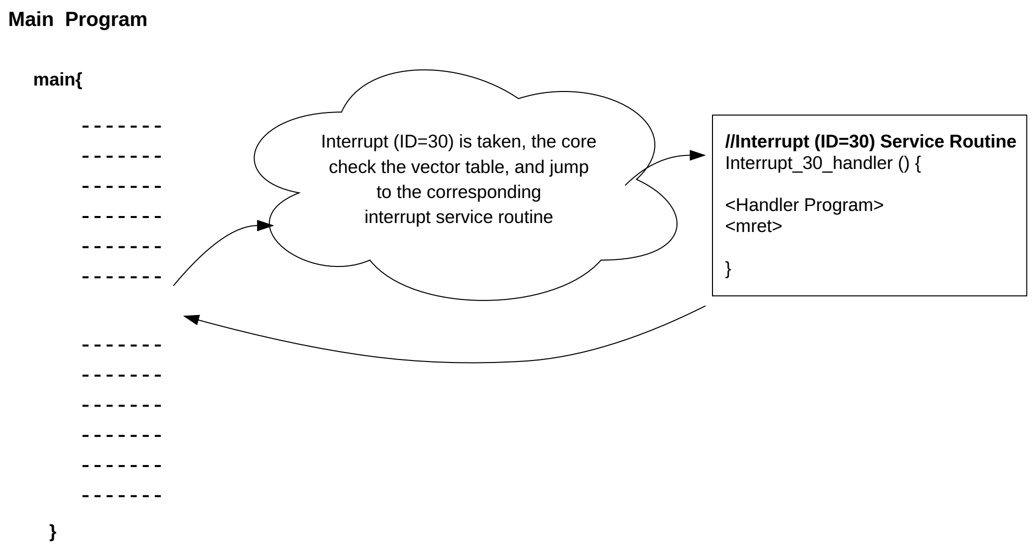

Main Program
Take Interrupt: Jump to the common code
<Save Context>
The overhead of back-to-back "Context Saving" and "Context Restoring" is eliminated
csrrw ra, CSR_JALMNXTI,ra
Interrupt (ID=30) Service Routine
csrrw ra, CSR_JALMNXTI,ra
Interrupt (ID=29) Service Routine
csrrw ra, CSR_JALMNXTI,ra
Interrupt (ID=28) Service Routine
Return to the common code
<Restore Cintext>

Fig. 10.11: Interrupt tail-chaining

10.13.2 Vectored Processing Mode

10.13.2.1 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 *Example for vectored interrupt* (page 37).


Main Program
main{

}
Interrupt (ID=30) is taken, the core check the vector table, and jump to the corresponding interrupt service routine
//Interrupt (ID=30) Service Routine
Interrupt_30_handler () {
<Handler Program>
<mret>
}

Fig. 10.12: Example for vectored interrupt

Vectored Processing Mode has the following features:

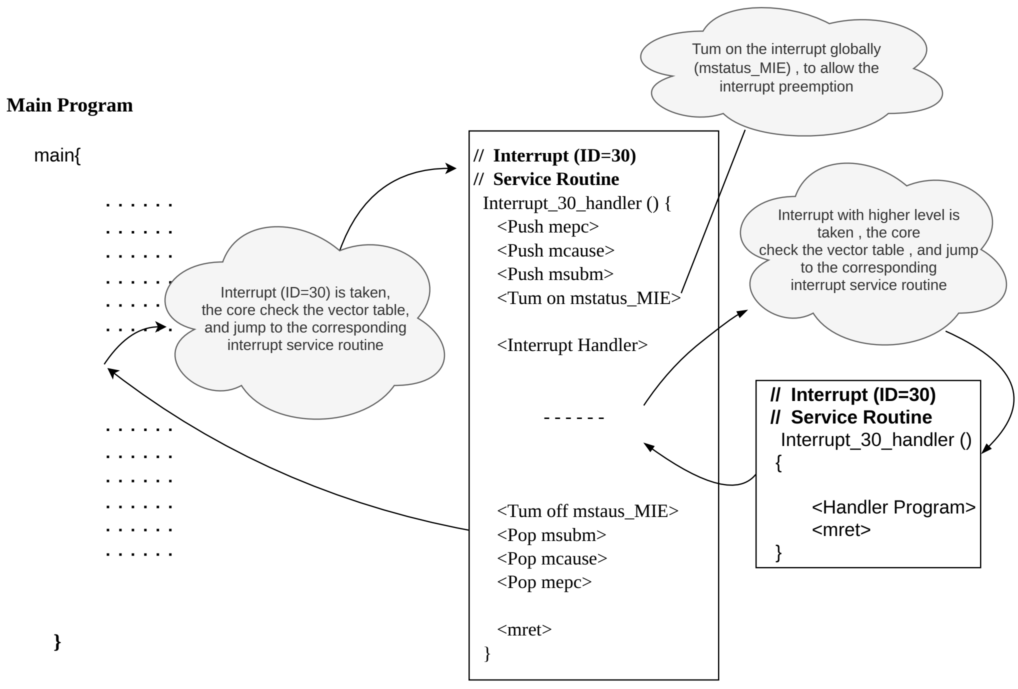
- The core will jump directly to the interrupt service routine without context saving. Therefore, the latency of the vectored interrupt is very short. Ideally, it only takes 6 cycles from the interrupt initiation to the execution of the first instruction of the interrupt service routine (C function), 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.

10.13.2.2 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 *Example for vectored interrupt supported preemption* (page 38):

- 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).


Main Program
main{
......
}
Interrupt (ID=30) is taken, the core check the vector table, and jump to the corresponding interrupt service routine
// Interrupt (ID=30)
// Service Routine
Interrupt_30_handler () {
<Push mepc>
<Push mcause>
<Push msubm>
<Tum on mstatus_MIE>
<Interrupt Handler>

<Tum off mstaus_MIE>
<Pop msubm>
<Pop mcause>
<Pop mepc>
<mret>
}
Tum on the interrupt globally (mstatus_MIE) , to allow the interrupt preemption
Interrupt with higher level is taken , the core check the vector table , and jump to the corresponding interrupt service routine
// Interrupt (ID=30)
// Service Routine
Interrupt_30_handler ()
{
<Handler Program>
<mret>
}

Fig. 10.13: Example for vectored interrupt supported preemption

As described above, with the special processing, the vectored processing mode can support interrupt preemption, as shown in *Interrupt preemptions caused by three sequential vectored interrupts* (page 39): 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.

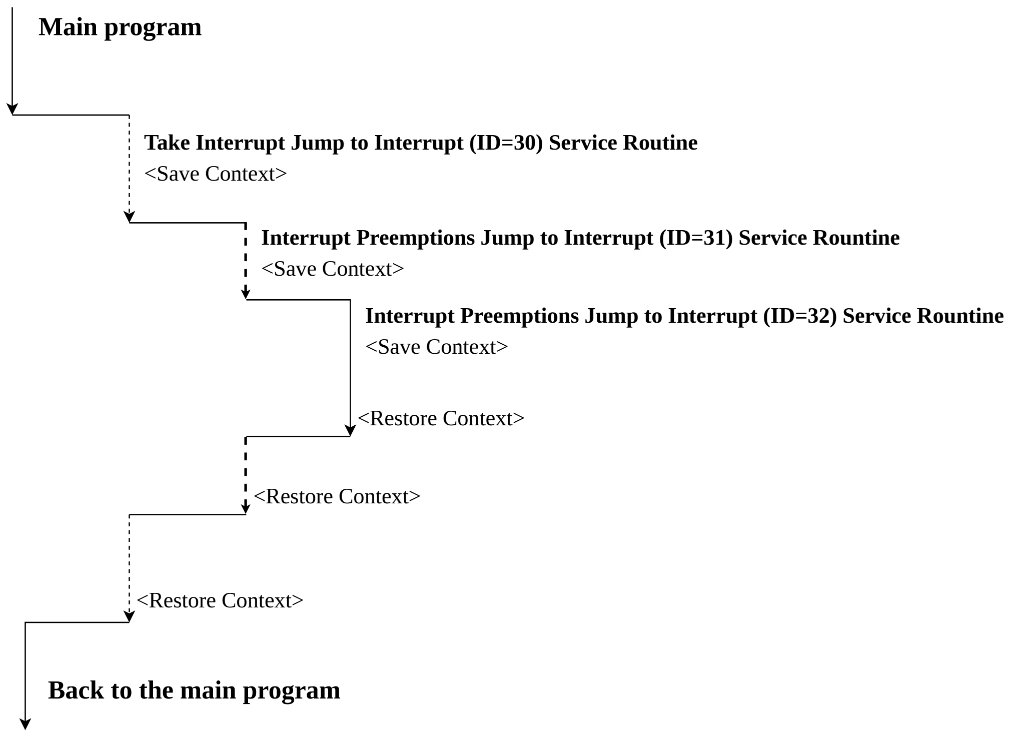

Main program
Take Interrupt Jump to Interrupt (ID=30) Service Routine
<Save Context>
Interrupt Preemptions Jump to Interrupt (ID=31) Service Rountine
<Save Context>
Interrupt Preemptions Jump to Interrupt (ID=32) Service Rountine
<Save Context>
<Restore Context>
<Restore Context>
<Restore Context>
Back to the main program

Fig. 10.14: Interrupt preemptions caused by three sequential vectored interrupts

10.13.2.3 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".

11 TIMER Unit Introduction

11.1 TIMER Overview

The Timer Unit (TIMER for short) is used to generate the Timer Interrupt and Software Interrupt in Hummingbird E603.

11.2 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 Hummingbird E603.
- Registers and the corresponding offset in the TIMER unit are shown in *The addresses offset of registers in the TIMER unit* (page 40)

Table 11.1: The addresses offset of registers in the TIMER unit

Offset	Permission/Width	Register Name	Default Value	Function Description
0x0	MRW/4B	mtime_lo	0x00000000	Reflect the lower 32-bit value of mtime. Shadow copy of MTIME in CLINT mode
0x4	MRW/4B	mtime_hi	0x00000000	Reflect the upper 32-bit value of mtime. Shadow copy of MTIME in CLINT mode
0x8	MRW/4B	mtimecmp_lo	0xFFFFFFFF	Set the lower 32-bit value of mtimecmp. Shadow copy of MTIMECMP for in CLINT mode
0xC	MRW/4B	mtimecmp_hi	0xFFFFFFFF	Set the upper 32-bit value of mtimecmp. Shadow copy of MTIMECMP for in CLINT mode
0xFEC	MRW/4B	mtime_srw_ctrl	0x00000000	Control S-mode can access this timer or not.
0xFF0	MRW/4B	msftrst	0x00000000	Generate soft-reset request.
0xFF8	MRW/4B	mtimectl	0x00000000	Control some features of the time counter.
0xFFC	MRW/4B	msip	0x00000000	Generate the Software Interrupt. Shadow copy of MSIP in CLINT mode

continues on next page

Table 11.1 – continued from previous page

Offset	Permission/Width	Register Name	Default Value	Function Description
0x1000	MRW/4B	MSIP	0x00000000	Software Interrupt in CLINT mode
0x5000	MRW/8B	MTIMECMP	0x00000000	MTIMECMP for Hart-0 in CLINT mode
0xCFF8	MRW/8B	MTIME	0x00000000	MTIME in CLINT mode

Note:

- Registers in the TIMER unit only support aligned read and write access with WORD size.
- The hardware can guarantee that the registers can only be accessed under M-mode.

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

11.2.1 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.

11.2.2 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 *ECLIC Unit Introduction* (page 47) for details of ECLIC.

11.2.3 Control the Timer Counter through mtimectl

The register mtimectl is implemented to control the behaviors of timer counting , as shown in *mtimectl bit fields* (page 41).

Table 11.2: mtimectl bit fields

Field	Bits	Permission	Default Value	Description
Reserved	31:3	N/A	N/A	Reserved, ties to 0

continues on next page

Table 11.2 – continued from previous page

Field	Bits	Permission	Default Value	Description
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 Hummingbird E603 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 interrupt request will be a pulse request. In this case, timer interrupt should be set to edge-trigger mode.

11.2.4 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 *msip bit fields* (page 42), 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 *ECLIC Unit Introduction* (page 47) for details of ECLIC.

Table 11.3: msip bit fields

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

11.2.5 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 *msftrst bit fields* (page 42), 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 Hummingbird E603 for details about the signals sysrstreq, core_reset_n and por_reset_n.

Table 11.4: msftrst bit fields

Field	Bits	Permission	Default Value	Description
MSFTRST	31	RW	0	This bit is used to generate the Soft-Reset Request

continues on next page

Table 11.4 – continued from previous page

Field	Bits	Permission	Default Value	Description
Reserved	30:0	N/A	N/A	Reserved, ties to 0

12 PLIC Unit Introduction

12.1 PLIC Overview

For the Linux capable applications, Nuclei Hummingbird E603 processor core have been equipped with a Platform-Level Interrupt Controller (PLIC), which is part of RISC-V standard privileged architecture specification (riscv-privileged-20240411.pdf).

User can easily get the documents from Nuclei released doc-package or 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 fig_Interrupt_Connection_for_single-core_with_PLIC_ECLIC_configured_and_PLIC_enabled.
- The PLIC need to be enabled by setting the LSB bits of CSR registers mtvec as CLINT mode. Please refer to *Setting CLINT or CLIC mode* (page 23) for the details.
- The PLIC is functionally exclusive to ECLIC. The ECLIC and PLIC connection diagrams are as described in ECLIC_PLIC_and_CIDU_Connection_Diagram.

12.2 PLIC Registers

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

- The base address of the PLIC unit, please refer to the specific databook of the processor core.
- Registers and the corresponding offset in the PLIC unit are shown in *The addresses offset of registers in the PLIC unit* (page 44).

Table 12.1: 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	S*RW	Source 1 priority	0x0
0x00_0008	4B	S*RW	Source 2 priority	0x0
……		……	……	……
0x00_0FFC	4B	S*RW	Source 1023 priority	0x0
……		……	……	……
0x00_1000	4B	S*RW	Start of pending array (bit 0-31)	0x0
……		……	……	……

continues on next page

Table 12.1 – continued from previous page

OFFSET	WIDTH	PERMISSION	DESCRIPTION	DEFAULT VALUE
0x00_107C	4B	S*RW	Last word of pending array (bit 992-1023)	0x0
……		……	……	……
0x00_2000	4B	MRW	Start of Hart 0 M-mode interrupt enables (source 0-31)	0x0
……		……	……	……
0x00_207C	4B	MRW	Last word of Hart 0 M-mode interrupt enables (source 992-1023)	
0x00_2080	4B	SRW	Start of Hart 0 S-mode interrupt enables (source 0-31)	
……		……	……	
0x00_20FC	4B	SRW	Last word of Hart 0 S-mode interrupt enables (source 992-1023)	
0x20_0000	4B	MRW	Hart 0 M-mode Priority threshold	0x0
0x20_0004	4B	MRW	Hart 0 M-mode Claim/Complete	0x0
……			Reserved	
0x20_1000	4B	SRW	Hart 0 S-mode Priority threshold	0x0
0x20_1004	4B	SRW	Hart 0 S-mode Claim/Complete	0x0
……		……	……	
0x3FF_FFFC	4B	MRW	plic_srw_ctrl S-mode Priority/Pending Access 0x0 means S-mode can access all priority/pending registers. 0x1 means S-mode can not access all priority/pending registers.	0x0

Permission:

- MRW: Only M-mode can read/write. S-mode and U-mode write ignore, read return 0.
- S*RW: M-mode can access, S-mode access depends on plic_srw_ctrl register. U-mode write ignore, read return 0.
  - If SRW bit of plic_srw_ctrl is 1, S-mode can read/write.
  - If SRW bit of plic_srw_ctrl is 0, S-mode write ignore, read return 0.
- SRW: M-mode and S-mode can read/write. U-mode write ignore, read return 0.

Note:

- PLIC registers only support aligned access which is the size of word.
- The above "RO" means read-only, and any write to this read-only register will be ignored without generating bus error.
- The hardware can guarantee that the registers can only be accessed under the corresponding privilege mode.

12.2.1 Control S-mode can access plic or not through plic_srw_ctrl

The register plic_srw_ctrl is implemented to control S-mode can access mtime or not , as shown in *plic_srw_ctrl bit fields* (page 46).

Table 12.2: plic_srw_ctrl bit fields

Field	Bits	Attr	Default Value	Description
Reserved	31:1	RO	0	Reserved, ties to 0
SRW	0	RW	0	Control S-mode can read or write plic registers or not. 0: S-Mode can read/write all plic registers. 1: S-Mode can not read/write plic registers

Note: This register only can be accessed by M-mode.

13

ECLIC Unit Introduction

13.1 ECLIC Overview

For the real-time or microcontroller applications, the fast interrupt handling scheme is very important, hence, Hummingbird E603 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 *ECLIC Connection (when ECLIC is enabled)* (page 48).
- The ECLIC need to be enabled by setting the LSB bits of CSR registers mtvec as ECLIC mode.

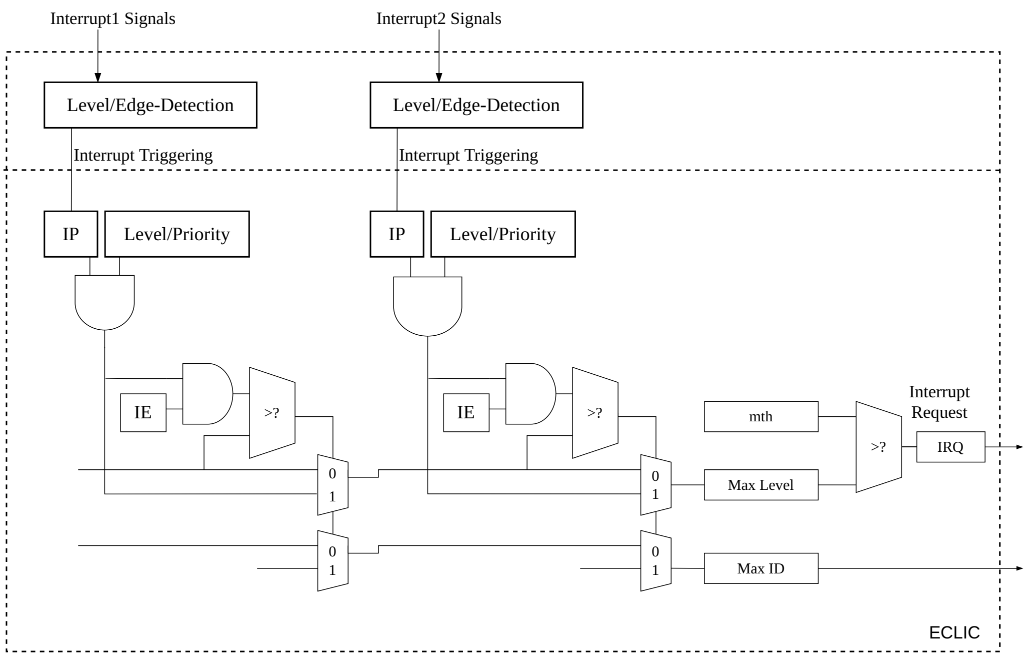


Fig. 13.1: The logic structure of the ECLIC unit

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 *The addresses offset of registers in the ECLIC unit* (page 49), and its structure is shown in *The logic structure of the ECLIC unit* (page 47) 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.

13.2 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 10-2.

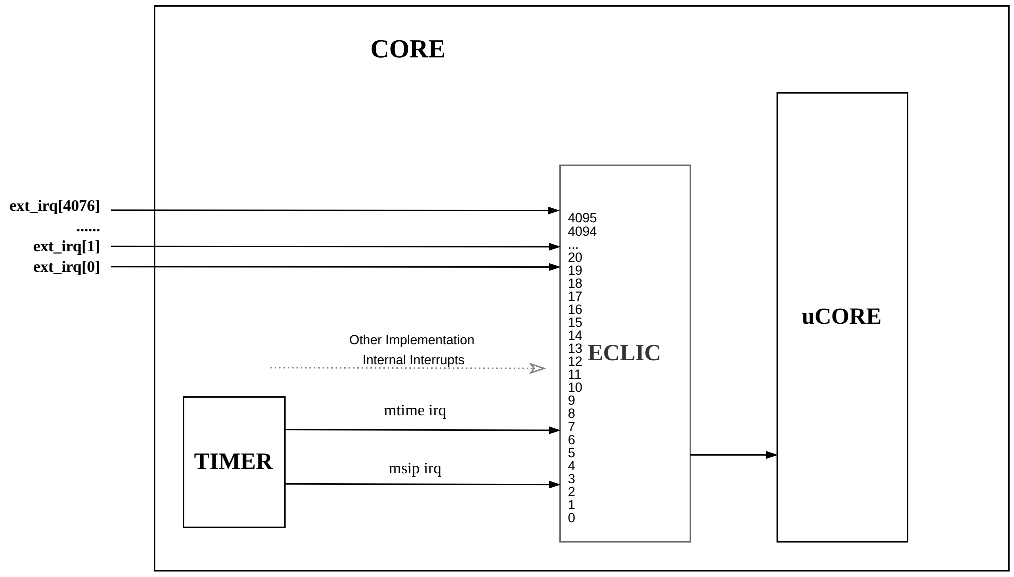


Fig. 13.2: ECLIC Connection (when ECLIC is enabled)

13.3 ECLIC Interrupt Source

As shown in *ECLIC Connection (when ECLIC is enabled)* (page 48), 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.

13.4 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 Hummingbird E603, the interrupt IDs ranged from 0 to 18 are reserved for the core-specified internal interrupts. 3 & 7 are fixed for Software Interrupt and Timer Interrupt in all Hummingbird E603s. Some Nuclei core may have more internal interrupts, this table dose not show them, please check related chapter for more details.
- The interrupt source ID greater than 18 can be used by the user to connect external interrupt sources.

The details are shown in *ECLIC interrupt sources and assignment* (page 49).

Table 13.1: 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.

13.5 ECLIC Registers

The ECLIC is a memory-mapped unit.

- The base address of the ECLIC unit, please refer to the specific datasheet of the Hummingbird E603.
- Registers and the corresponding offset in the ECLIC unit are shown in *The addresses offset of registers in the ECLIC unit* (page 49)

Table 13.2: 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

continues on next page

Table 13.2 – continued from previous page

Offset	Permission	Register	Width
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 ECLIC 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.

13.5.1 cliccfg

This cliccfg register is a global configuration register. The software can set global configurations by write this register. *cliccfg bit fields* (page 50) describes the bit fields of this register.

Table 13.3: 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 *ECLIC Interrupt Level and Priority* (page 53) for more details.
Reserved	0	R	N/A	Reserved, ties to 1.

13.5.2 clicinfo

The clicinfo register is a global info register. The software can query the global parameters by reading this register. *clicinfo bit fields* (page 50) describes the bit fields of this register.

Table 13.4: 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 *ECLIC Interrupt Level and Priority* (page 53) 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.

13.5.3 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. *mth bit fields* (page 51) describes the bit fields of this register.

Table 13.5: mth bit fields

Field	Bits	Permission	Default Value	Description
mth	7:0	RW	N/A	Target threshold level register. Please see *ECLIC Interrupt Threshold Level* (page 55) for more details.

13.5.4 clicintip[i]

The clicintip[i] register is the pending flag register for the interrupt source. *clicintip[i] bit fields* (page 51) describes the bit fields of this register.

Table 13.6: 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 *ECLIC Interrupt Pending Bit (IP)* (page 52) for more details.

13.5.5 clicintie[i]

The clicintie[i] register is the enable bit register for the interrupt source. *clicintie[i] bits fields* (page 51) describes the bit fields of this register.

Table 13.7: clicintie[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 *ECLIC Interrupt Enable Bit (IE)* (page 52) for more details.

13.5.6 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.:ref:*table-clicintattr[i]_bit_fields* describes the bit fields of this register.

Table 13.8: 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 *ECLIC Interrupt Source Level or Edge-Triggered Attribute* (page 52) for more details.
shv	0	RW	0	Used to set whether the interrupt is vectored or non-vectored. Please see *ECLIC Interrupt Vectored and Non-Vectored Processing Mode* (page 54) for more details.

13.5.7 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 *ECLIC Interrupt Level and Priority* (page 53) for more details.

13.6 ECLIC Interrupt Enable Bit (IE)

As shown in *The logic structure of the ECLIC unit* (page 47), 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.

13.7 ECLIC Interrupt Pending Bit (IP)

As shown in *The logic structure of the ECLIC unit* (page 47), 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 *ECLIC Interrupt Source Level or Edge-Triggered Attribute* (page 52).
- 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 *ECLIC Interrupt Source Level or Edge-Triggered Attribute* (page 52) for more details.
- For edge-triggered interrupt source, the IP bit may be cleared by the hardware itself. Please see *ECLIC Interrupt Source Level or Edge-Triggered Attribute* (page 52) for more details.

13.8 ECLIC Interrupt Source Level or Edge-Triggered Attribute

As shown in *The logic structure of the ECLIC unit* (page 47), 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.

13.9 ECLIC Interrupt Level and Priority

As shown in *The logic structure of the ECLIC unit* (page 47), each interrupt sources of the ECLIC is fixed , 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 *clicintctl[i] format example* (page 53).
  - 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 Hummingbird E603.
- 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 *clicintctl[i] format example* (page 53).

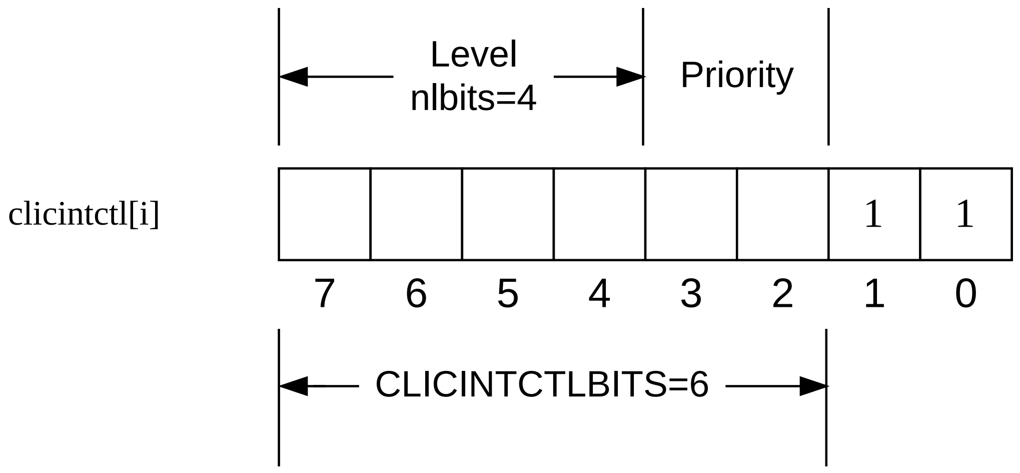
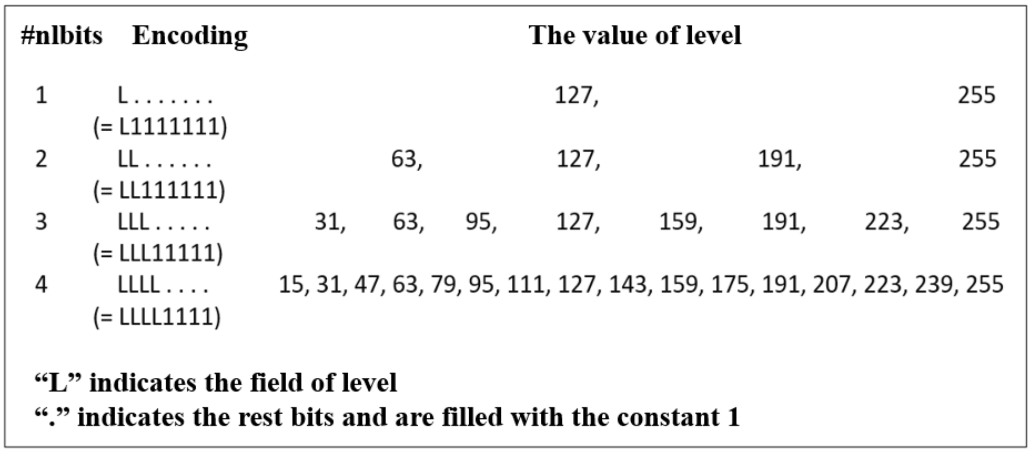


Fig. 13.3: clicintctl[i] format example

- 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 *Example for the decoding of level* (page 54).
    - Note: if cliccfg.nlbits = 0, the value of level will be regarded as a fixed value 255. As shown in *Examples of cliccfg settings* (page 54).
  - 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 *(CLIC mode) Interrupt Preemption* (page 32).

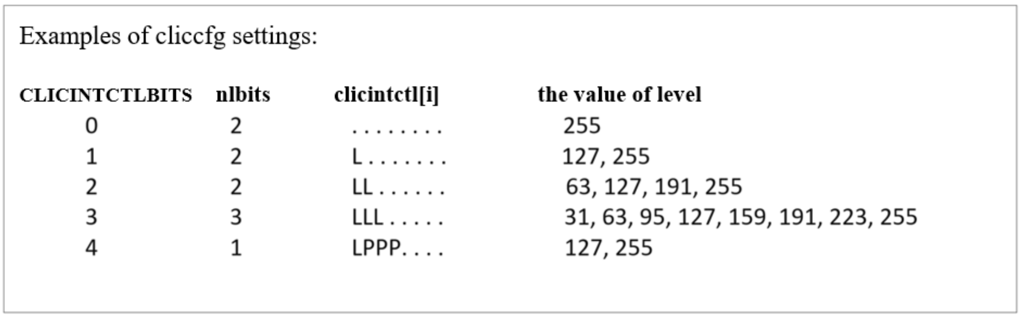
  * If there are multiple pending interrupts (IP is 1), then the ECLIC needs to make an arbitration to determine which interrupt needs to be sent to the core to take. The arbitration needs to take the level of each interrupt source into the consideration. Please see *Interrupt Levels, Priorities and Arbitration* (page 26) 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

Fig. 13.4: 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

Fig. 13.5: 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 *ECLIC Interrupt Arbitration Mechanism* (page 55) for details.

13.10 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 *Vectored Processing Mode* (page 37).
- 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 *Non-Vectored Processing Mode* (page 34).

13.11 ECLIC Interrupt Threshold Level

As shown in *The logic structure of the ECLIC unit* (page 47), 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.

13.12 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.

13.13 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 *(CLIC mode) Interrupt Preemption* (page 32), *(CLIC mode) Interrupt Tail-Chaining* (page 33), and *Non-Vectored Processing Mode* (page 34) for more details.

14 ECLIC and PLIC Connection Diagram

The ECLIC and PLIC key points are as followings:

- 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 *Interrupt Connection (for single-core with PLIC/ECLIC configured and PLIC enabled)* (page 56).
  - If the ECLIC is disabled by software, then the interrupts will be handled by PLIC, and the ECLIC will be bypassed, as depicted in *Interrupt Connection (for single-core with PLIC/ECLIC configured and ECLIC enabled)* (page 57).

Note:

In the flowing diagrams, each core may have other implementation related Internal Interrupts, but we only show the Software Interrupt and Timer Interrupt for short.

14.1 Single-core with PLIC/ECLIC configured and PLIC enabled

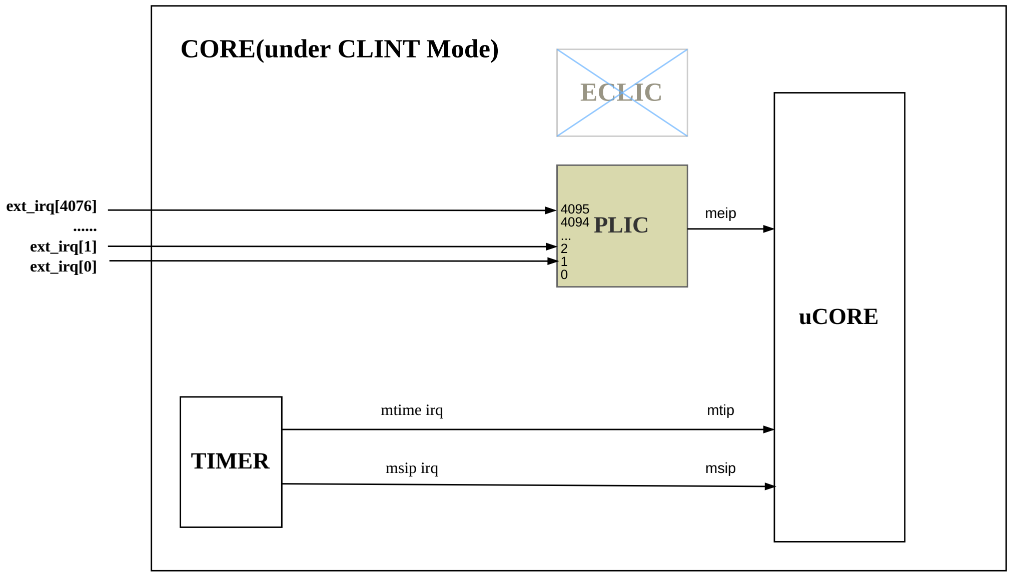


Fig. 14.1: Interrupt Connection (for single-core with PLIC/ECLIC configured and PLIC enabled)

14.2 Single-core with PLIC/ECLIC configured and ECLIC enabled

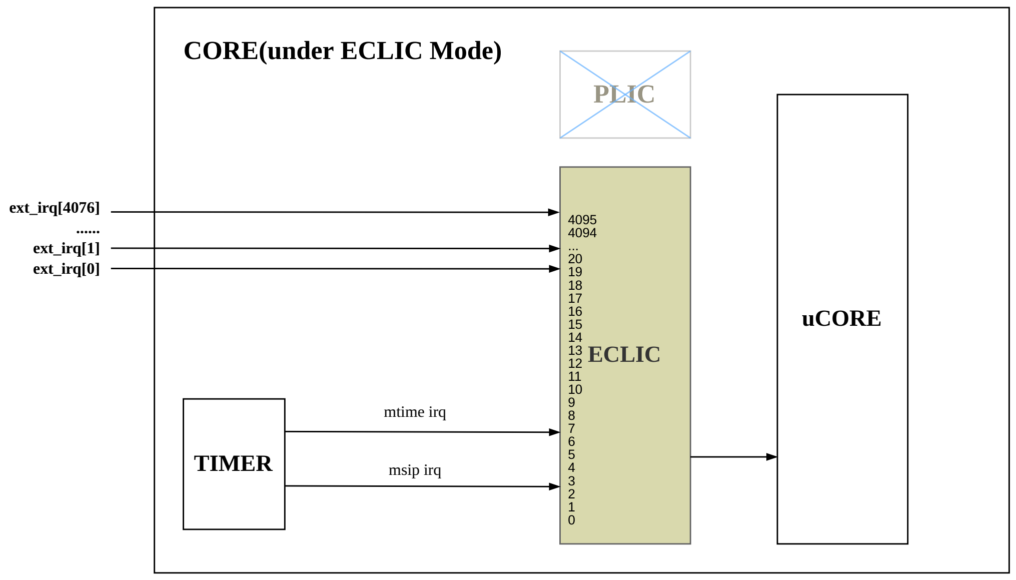


Fig. 14.2: Interrupt Connection (for single-core with PLIC/ECLIC configured and ECLIC enabled)

15 PMP Introduction

15.1 PMP Overview

Hummingbird E603 supports 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.

15.2 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.

15.2.1 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. Hummingbird E603 have the PMP entries number fixed 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.

15.2.2 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. Hummingbird E603 have the PMP grain fixed at the build time, please refer to the specific datasheet of the Hummingbird E603.

15.2.3 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. Hummingbird E603 already supports the TOR mode in A field of pmpcfg<x> registers.

15.2.4 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 Hummingbird E603 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.

16

MMU Introduction

16.1 MMU Overview

Nuclei Hummingbird E603 has MMU (Memory Management Unit) 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.

16.2 MMU Specific Features to Nuclei Core

In order to simplify the hardware implementation, or there are some features are intrinsically hardware implementation relevant, Nuclei Hummingbird E603 processor core has some MMU specific features, which is detailed at next sections.

16.2.1 TLB Entries Number

MMU has two level TLB (Translation Lookaside Buffer) implemented to cache the page tables for fast subsequent accessing: MTLB and I/D-TLB; MTLB is the main/joint TLB for both instruction and data page table, MTLB entry number is fixed; I/D-TLB is dedicate for instruction/data page table, each has 8/16 entries; I/D-TLB will be accessed first, if miss then MTLB 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.

17 Performance Monitor Introduction

Hummingbird E603 supports to configure Performance Monitor which is to count on various events for performance profiling. And our implementation of Performance Monitor exactly follows RISC-V Privilege Spec, users also can refer that for details.

17.1 Performance Monitor CSRs

Table 17.1: Performance Monitor CSRs list

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

continues on next page

Table 17.1 – continued from previous page

Type	CSR ADDR	RW	Name	Description
	0xC03	URO	hpmcounter3	Supervisor/User mode performance monitor counter 3.
	0xC04	URO	hpmcounter4	Supervisor/User mode performance monitor counter 4.
	…	…	…	…
	0xC1F	URO	hpmcounter31	Supervisor/User mode performance monitor counter 31.
	0xC83	URO	hpmcounter3h	Upper 32 bits of hpmcounter 3, RV32 only.
	0xC84	URO	hpmcounter4h	Upper 32 bits of hpmcounter 4, RV32 only.
	…	…	…	…
	0xC9F	URO	hpmcounter31h	Upper 32 bits of hpmcounter 31, RV32 only.

17.1.1 mhpmcounterx

This CSR is used to record specific micro-architecture event number, Note that the width is different for RV32 and RV64.

Table 17.2: mhpmcounterx

Field Name	Bits	RW	Reset Value	Description
Reserved	MXLEN-1:32	RW	0	Tie 0 for RV32, RV64 only.
mhpmcounterx	31:0	RW	0	Machine performance monitor counter x. 3=< x <=6

Table 17.3: mhpmcounterx

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

17.1.2 mhpmcounterhx

This CSR is used to record specific micro-architecture event upper 32 bit number, it is only for RV32.

Table 17.4: mhpmcounterhx

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

17.1.3 mcountinhibit

The counter-inhibit register mcountinhibit is a MXLEN-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 17.5: mcountinhibit

Field Name	Bits	RW	Reset Value	Description
Reserved	MXLEN-1:7	R	0	-
HPM6	6	RW	0	If set, stop increase performance monitor counter 6.

continues on next page

Table 17.5 – continued from previous page

Field Name	Bits	RW	Reset Value	Description
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 17.6: mhpmevent 3-6

Field Name	Bits	RW	Reset Value	Description
Reserved	MXLEN-1:32	R	0	-
mevent_enable	31	RW	1	Enable the corresponding performance monitor counter increment for events in Machine Mode.
Reserved	30	R	0	-
sevent_enable	29	RW	1	Enable the corresponding performance monitor counter increment for events in Supervisor Mode.
uevent_enable	28	RW	1	Enable the corresponding performance monitor counter increment 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 *Event Selection Value for Instruction Commit Events* (page 63) for more details. For memory access events please see *Event Selection Value for Memory Access Events* (page 64) for more details.
event_sel	3:0	RW	0	0: Select the instruction commit events, such as load, store, bjp ect. See *Event Selection Value for Instruction Commit Events* (page 63) for more details. 1: Select the memory access events, such as icache miss, dcache miss ect. See *Event Selection Value for Memory Access Events* (page 64) for more details.

Table 17.7: mhpmevent7-31

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

Table 17.8: Event Selection Value 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

continues on next page

Table 17.8 – continued from previous page

event_idx (mhpeventx[3:0]==0)	Event Name
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 17.9: Event Selection Value 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 WFI/WFE Low-Power Mechanism

The Hummingbird E603 can support sleep mode for lower power consumption.

18.1 Enter the Sleep Mode

The Hummingbird E603 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 Hummingbird E603 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.

18.2 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 *Enter the Sleep Mode* (page 65), the Wait for Interrupt mechanism can be implemented by executing the WFI instruction with setting the value of CSR wfe.WFE to 0.

18.3 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 *Enter the Sleep Mode* (page 65), 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.
```

18.4 Exit the Sleep Mode

The key points of the Hummingbird E603 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.

18.4.1 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.

18.4.2 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.

18.4.3 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 Hummingbird E603 for details about the signal rx_evt.

18.4.4 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.

19 Hummingbird E603 CSRs Descriptions

19.1 CSR Overview

CSR (Control and Status Registers) is part of RISC-V standard privileged architecture. Basically, Hummingbird E603 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.

19.2 Hummingbird E603 CSRs List

In this CSRs List of Hummingbird E603, there are RISC-V standard CSRs and customized CSRs. Following table only describes the customized CSRs.

Table 19.1: Customized CSRs in the Hummingbird E603

Address	R&W	Name	Description
0x307	MRW	mtvt	ECLIC Interrupt Vector Table Base Address
0x320	MRW	mcountinhibit	Customized register used to control the on & off of counters
0x345	MRW	mnxti	Used to enable taking the next interrupt and return the entry address of the next interrupt handler
0x346	MRO	mintstatus	Current Interrupt Level
0x348	MRW	mscratchcsw	Scratch swap register for privileged mode
0x349	MRW	mscratchcswl	Scratch swap register for interrupt mode and normal mode
0x7c0	MRW	milm_crtl	Enable/Disable the ILM address space
0x7c1	MRW	mdlm_crtl	Enable/Disable the DLM address space
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

continues on next page

Table 19.1 – continued from previous page

Address	R&W	Name	Description
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	Customized register to control the TLB
0x7de	MRW	mecc_lock	To lock ECC configure registers, then all the ECC related CSRs cannot be modified, unless by reset
0x7e2	MRW	mfp16mode	Customized register used to set 16 bit float precision mode
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
0x7f3	MRW	mdevb	Device region begin address
0x7f4	MRW	mdevm	Device region address mask
0x7f5	MRW	mnocb	Non-cacheable region begin address
0x7f6	MRW	mnocm	Non-cacheable region address mask
0x7f7	MRO	mirgb_info	IREGION Configuration Information
0x810	URW	wfe	Customized register used to control the WFE mode
0x811	URW	sleepvalue	Customized register used to indicate the WFI sleep mode
0x812	URW	txevt	Customized register used to send an event
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.

19.3 Accessibility of CSRs in the Hummingbird E603

The CSRs Accessibility in the Hummingbird E603:

- 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.

19.4 RISC-V Standard CSRs Supported in the Hummingbird E603

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 Hummingbird E603, but those CSRs who also have some unique differences implemented in Hummingbird E603.

19.4.1 mie

In Hummingbird E603, 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.

19.4.2 mip

In Hummingbird E603, 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.

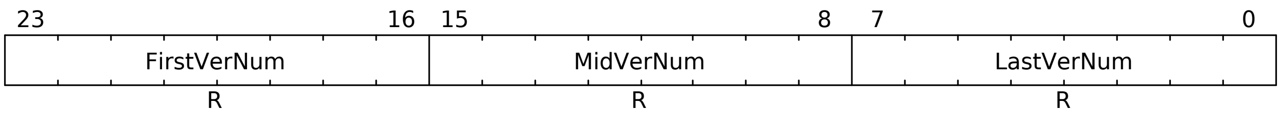
But note: the mip CSR is not effective when the core is in the CLIC mode, and the readout of mip is always 0, the writing is ignored.

19.4.3 marchid

marchid is to indicate the core series . marchid[MXLEN-1:16] is reserved for future use. marchid[15:0] indicates the core series name, Nuclei Hummingbird E603 is 0x603.

19.4.4 mimpid

mimpid is to indicate the core product version (also the RTL version), mimpid[MXLEN-1:24] is reserved for future use , mimpid[23:0] indicates the core product version X.Y.Z, X/Y/Z are all hexadecimal number and each occupies 8 bits.Nuclei Hummingbird E603 is 0x100. The format is as followed:



19.4.5 mhartid

In Hummingbird E603, 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 Hummingbird E603, hart ID is controlled by input signal core_mhartid.

19.4.6 mtvec

The mtvec register holds the entry address of the interrupt and exception handler. Field of mtvec register is shown in following table.

- When mtvec holds the exception entry address:
  - The value of the address field must always be aligned on a 64-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 *Non-Vectored Processing Mode* (page 34) for more information about non-vectored mode interrupt handler entry address.
    - See *Vectored Processing Mode* (page 37) for more information about vectored mode interrupt handler entry address.

Table 19.2: mtvec register

Field	Bits	Description
ADDR	MXLEN-1:6	mtvec address
MODE	5:0	**MODE field determine interrupt mode:** 000011 means CLIC interrupt mode; others means CLINT interrupt mode.

19.4.7 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 Hummingbird E603, there some additional fields added to support CLIC mode interrupt handling.

The mcause register is formatted as shown in following table:

Table 19.3: mcause register

Field	Bits	Description
Reserved	MXLEN-1:32	Reserved 0
INTERRUPT	31	Current trap type: 0: Exception or NMI; 1: Interrupt.
MINHV	30	Indicate processer is reading interrupt vector table
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:

- Filed of MINHV, MPP, MPIE and MPIL are only effect in CLIC mode. When in CLINT mode, these field is masked read as zero, write ignored.
- 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 *mmisc_ctl* (page 76).

19.4.8 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 Hummingbird E603, 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 *mcountinhibit* (page 72) for more information.

19.4.9 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 Hummingbird E603, 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 *mcountinhibit* (page 72) for more information.

19.5 Customrized CSRs supported in Hummingbird E603

This section introduces customized CSRs in the Hummingbird E603.

19.5.1 mcountinhibit

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

Table 19.4: mcountinhibit register

Field	Bits	Description
Reserved	MXLEN-1:3	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.

19.5.2 mnvec

The Hummingbird E603 customized CSR mnvec register holds the NMI entry address.

In order to understand this register, please see Chapter NMI_Handling_in_Nuclei_processor_core 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 *mmisc_ctl* (page 76).

19.5.3 msubm

The Hummingbird E603 customized CSR msubm register holds the current machine sub-mode and the machine sub-mode before the current trap. See Section *Machine Sub-Mode added by Nuclei* (page 15) for details.

Table 19.5: msubm register

Field	Bits	Description
Reserved	MXLEN-1:10	Reserved 0

PTYP	9:8	Machine sub-mode before entering the trap:
		0: Normal Machine Mode;
		1: Interrupt Handling Mode;
		2: Exception Handing Mode;
		3: NMI Handing Mode.

TYP	7:6	Current sub-mode:
		0: Normal Machine Mode;
		1: Interrupt Handling Mode;
		2: Exception Handing Mode;
		3: NMI Handing Mode.

Reserved	5:0	Reserved 0.

19.5.4 mdcause

Since there might be some exceptions share the same mcause.EXCCODE value. To further record the differences, Hummingbird E603 customizes CSR mdcause register to record the detailed information about the exception.

Table 19.6: mdcause register

Field	Bits	Description
Reserved	MXLEN-1:3	Reserved 0

continues on next page

Table 19.6 – continued from previous page

Field	Bits	Description
PTYP	2:0	Further record the detailed information about the exception. When mcause.EXCCODE = 1 (Instruction access fault): 0: Reserved 1: PMP permission violation 2: Bus error 3-7: Reserved When mcause.EXCCODE = 5 (Load access fault): 0: Reserved 1: PMP permission violation bus error 2: Bus error caused by core memory read 3: Bus error caused by NICE write back. 4-7:Reserved 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 bus error 2: Bus error caused by core memory write 3-7: Reserved When mcause.EXCCODE = 12 (Instruction page fault): 0-4: Reserved 5: Instruction page fault caused by MMU 6: Instruction page fault caused by SPMP 7: Reserved When mcause.EXCCODE = 13 (Load page fault): 0-4: Reserved 5: Load page fault caused by MMU 6: Load page fault bus error caused by SPMP 7: Reserved When mcause.EXCCODE = 15 (Store page fault): 0-4: Reserved 5: Store page fault caused by MMU 6: Store page fault bus error caused by SPMP 7: Reserved

19.5.5 mcache_ctl

Hummingbird E603 customizesd CSR mcache_ctl register to control the I-Cache and D-Cache features.

Table 19.7: mcache_ctl register

Field	Bits	Description
Reserved	MXLEN-1:21	Reserved 0.
DC_DRAM_ECC_CHK_EN	20	Controls to check the ECC Code or not when core access to D-Cache data rams. This bit can be turned on for injecting ECC errors to test the ECC handler. 0: Disable to check the ECC codes (Default) 1: Enable to check the ECC codes Note: When D-Cache or ECC is not configured, this field is tied to 0.

continues on next page

Table 19.7 – continued from previous page

Field	Bits	Description
DC_TRAM_ECC_CHK_EN	19	Controls to check the ECC Code or not when core access to D-Cache tag rams. This bit can be turned on for injecting ECC errors to test the ECC handler. 0: Disable to check the ECC codes (Default) 1: Enable to check the ECC codes Note: When D-Cache or ECC is not configured, this field is tied to 0.
DC_ECC_EXCP_EN	18	D-Cache double bit ECC exception enable control: 0: ECC error will not trigger exception (Default) 1: ECC error will trigger exception Note: When D-Cache or ECC is not configured, this field is tied to 0.
DC_ECC_EN	17	D-Cache ECC enable control: 0: Disable ECC 1: Enable ECC (Default) Note: When D-Cache or ECC is not configured, this field is tied to 0.
DC_EN	16	D-Cache enable control: 0: D-Cache Disable (default) 1: D-Cache Enable
Reserved	15:8	Reserved 0.
IC_CANCEL_EN	7	Supported only in 900 series, I-Cache change flow canceling enable control: 0: Disable canceling current accesses in icache e1 stage when change flow happens 1: Enable canceling previous accesses in icache e1 stage when change flow happens Note: When I-Cache is not configured, this field is tied to 0.
IC_PF_EN	6	Supported only in 900 series, I-Cache prefetch enable control: 0: Disable prefetching 1: Enable prefetching Note: When I-Cache is not configured, this field is tied to 0.
IC_DRAM_ECC_CHK_EN	5	Controls to check the ECC Code or not when core access to I-Cache data rams. This bit can be turned on for injecting ECC errors to test the ECC handler. 0: Disable to check the ECC codes (Default) 1: Enable to check the ECC codes Note: When I-Cache or ECC is not configured, this field is tied to 0.
IC_TRAM_ECC_CHK_EN	4	Controls to check the ECC Code or not when core access to I-Cache tag rams. This bit can be turned on for injecting ECC errors to test the ECC handler. 0: Disable to check the ECC codes (Default) 1: Enable to check the ECC codes Note: When I-Cache or ECC is not configured, this field is tied to 0.
IC_ECC_EXCP_EN	3	I-Cache double bit ECC exception enable control: 0: ECC error will not trigger exception (Default) 1: ECC error will trigger exception Note: When I-Cache or ECC is not configured, this field is tied to 0.
IC_ECC_EN	2	I-Cache ECC enable control: 0: Disable ECC 1: Enable ECC (Default) Note: When I-Cache or ECC is not configured, this field is tied to 0.

continues on next page

Table 19.7 – continued from previous page

Field	Bits	Description
IC_SCPD_MOD	1	I-Cache Scratchpad Mode enable control: 0: I-Cache works in the normal mode (default) 1: I-Cache works in 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 control: 0: I-Cache Disable (default) 1: I-Cache Enable

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 works as the Scratchpad.

19.5.6 mmisc_ctl

The Hummingbird E603 customized CSR mmisc_ctl controls various Nuclei micro-architecture implementation related features.

Table 19.8: mmisc_ctl register

Field	Bits	Description
Reserved	MXLEN-1:15	Reserved 0.
DBG_SEC_MODE	14	Indicate the secure mode of load/store or SBA when Core is in Debug Mode: 0: The debug access is not secure. 1: The debug access is secure (Reset Value). Note: currently only 900 v2.7.0 and later version support this.
SEC_MODE	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.
CORE_BUS_ERR	8	Control the bus error caused by core is exception or interrupt: 0: Core Bus Error caused by core is an exception; When Core Bus Error is an excetpion, please check *Core Bus Error Exception* (page 77) for more details. 1: Core Bus Error caused by core is a interrupt. When Core Bus Error is a interrupt, please check *Core Bus Error Interrupt* (page 77) for more details.
Reserved	7	Reserved 0.
UNALGN_ENA_BLE	6	Enable or disable misaligned load/store access, if disabled, accessing misaligned memory locations will trigger an Address Misaligned exception: 0: Disable misaligned load/store access; 1: Enable misaligned load/store access. Note: This field only takes effects for load/store specified in I/F/D Extension, for load/store specified in A Extension, misaligned accesses always trigger an Address Misaligned exception.
Reserved	5:4	Reserved 0.

continues on next page

Table 19.8 – continued from previous page

Field	Bits	Description
BPU_ENABLE	3	Enable or disable the BPU Unit: 0: Disable the BPU Unit; 1: Enable the BPU Unit. Note: BPU is on by default after reset except 900 Series. If BPU is disable by software, then all branches are predicted as jump statically until the BPU is enable again.
Reserved	2:0	Reserved 0.

19.5.6.1 Core Bus Error Exception

Currently Hummingbird E603 implments 7 types Core Bus Error, when it is configured to be exception (by CSR *mmisc_ctl*), the mapping of Core Bus Error type and CSR coding of *mcause* and *mdcause* is defined in below table.

Table 19.9: Core Bus Err Type and Exception Coding Mapping

Core Bus Err Type	*mcause* Exception Coding	*mdcause*
Bus Err casued by core memory read	5 (load access fault)	2
Bus Err casued by core memory write	7 (store access fault)	2
Bus Err casued by core memory read PMP violation	5 (load access fault)	1
Bus Err casued by core memory write PMP violation	7 (store access fault)	1
Bus Err casued by core memory read SPMP violation	13 (load page fault)	6
Bus Err casued by core memory write SPMP violation	15 (store page fault)	6
Bus Err casued by core's NICE write back	5 (load access fault)	3

19.5.6.2 Core Bus Error Interrupt

Currently Hummingbird E603 implments 7 types Core Bus Error, when it is configured to be interrupt (by CSR *mmisc_ctl*), the mapping of Core Bus Error type and CSR coding of *mcause* and *mdcause* is defined in below table. And it belongs to Internal Interrupt, and Interrupt ID is fixed to 18.

Table 19.10: Core Bus Err Type and Interrupt Coding Mapping

Core Bus Err Type	*mcause* Interrupt Coding	*mdcause*
Bus Err casued by core memory read	18	2
Bus Err casued by core memory write	18	2
Bus Err casued by core memory read PMP violation	18	1
Bus Err casued by core memory write PMP violation	18	1
Bus Err casued by core memory read SPMP violation	18	6
Bus Err casued by core memory write SPMP violation	18	6
Bus Err casued by core's NICE write back	18	3

Note:

- As Core Bus Error Interrupt belongs to Internal Interrupt, both CLIC Mode or CLINT Mode (with PLIC), the Interrupt ID is the same (18).
- In CLIC Mode, as Nuclei ECLIC can support 3 types of interrupt trigger and Core Bus Error Interrupt is edge type, so besides to set clicintie[18] = 1'b1 to enable it , user should also set clicintattr[18].trig = 2'b01.
- In CLINT Mode (with PLIC), the Core Bus Error is controlled by CSR *mie* and *mip*. And user has no need to consider the interrupt trigger type.

19.5.7 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 Nesting_of_Interrupt_NMI_and_Exception for more information.

Table 19.11: msavestatus register

Field	Bits	Description
Reserved	MXLEN-1:16	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 first-level NMI/Exception.
MPIE1	0	The interrupt enable bit before taking the first-level NMI/Exception.

19.5.8 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 Nesting_of_Interrupt_NMI_and_Exception for more information.

19.5.9 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 Nesting_of_Interrupt_NMI_and_Exception for more information.

19.5.10 msavedcause1 and 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 Nesting_of_Interrupt_NMI_and_Exception for more information.

19.5.11 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 *(CLIC mode) Interrupt Vector Table* (page 30) for more details.

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 19.12: 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
2049 to 4096	16KB

19.5.12 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.

19.5.13 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 19.13: mintstatus register

Field	Bits	Description
Reserved	MXLEN-1:32	Reserved 0
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.

19.5.14 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 *Non-Vectored Processing Mode* (page 34).

Table 19.14: mtvt2 register

Field	Bits	Description
CMMON-CODE-ENTRY	MXLEN-1:2	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.

19.5.15 jalmnxti

The Hummingbird E603 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 related tail-chaining in Section *Interrupt Handling in Hummingbird E603* (page 23).

19.5.16 pushmsubm

The Hummingbird E603 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.

19.5.17 pushmcause

The Hummingbird E603 customized CSR pushmcause provides a method to store the value of mcause in memory space which base address is SP with CSR instruction csrrwi.

For example:

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

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

19.5.18 pushmepc

The Hummingbird E603 customized CSR pushmepc provides a method to store the value of mepc in memory space which base address is SP with CSR instruction csrrwi.

For example:

```
csrrwi x0, PUSHMEPC, 1
```

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

19.5.19 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 one 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 a 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.

19.5.20 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 interrupt mode (not including exception) and normal mode.

Using the CSR instruction to read the register mscratchcswl, 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 ( (0 == mcause.mpil) != (0 == mintstatus.mil) ) 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 CSR 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.

19.5.21 sleepvalue

The Hummingbird E603 customized CSR sleepvalue controls different sleep modes. See Section *Enter the Sleep Mode* (page 65) for more information.

Table 19.15: sleepvalue register

Field	Bits	Description
Reserved	MXLEN-1:1	Reserved 0.
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.

19.5.22 txevt

The Hummingbird E603 customized CSR txevt controls output events.

Table 19.16: txevt register

Field	Bits	Description
Reserved	MXLEN-1:1	Reserved 0.
TXEVT	0	Event control: 0: No action; 1: The 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. Reset default value is 0.

19.5.23 wfe

The Hummingbird E603 customized CSR wfe controls whether the processor should be awakened by interrupt or event. See Section *Wait for Event* (page 66) for more information.

Table 19.17: wfe register

Field	Bits	Description
Reserved	MXLEN-1:1	Reserved 0.
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.

19.5.24 ucode

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

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

Table 19.18: ucode register

Field	Bits	Description
Reserved	MXLEN-1:1	Reserved 0.
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.

19.5.25 mcfg_info

This CSR is used to show the processor core's configuration information.

Table 19.19: mcfg_info register

Field	Bits	Description
Reserved	MXLEN-1:19	Reserved 0.
VP	18:17	Global configuration for VPU degree of parallelism in Group A: 0: The degree of parallelism is 1; 1: Reserved; 2: The degree of parallelism is 4 ; 3: Reserved. Note: Only take effect when VPU is configured. For RV32 core with VPU, only 0 is valid.
IREGION_EXIST	16	Shows the IREGION exist or not. It is Read Only. 0: IREGION not exist; 1: IREGION exists. Note: 1. IREGION (Internal Region) means the address space of all Private Peripherals in core is continuous and base address is defined by IREGION. If no IREGION, user can seperately configures the address space of each Private Peripheral. Please check Private Peripherals chapter of each series databook for more details. 2. New version of Nuclei Core supports IREGION and it is fixed in these new versions. Please check Configuration Options chapter of related databook for more details.
Reserved	15	Reserved 0.

continues on next page

Table 19.19 – continued from previous page

Field	Bits	Description
DSP_N3	14	Global configuration for DSP N3 Extension Support: 0: No DSP N3 Extension support; 1: Has DSP N3 Extension support. Note: Only take effect when DSP is configured.
DSP_N2	13	Global configuration for DSP N2 Extension Support: 0: No DSP N2 Extension support; 1: Has DSP N2 Extension support. Note: Only take effect when DSP is configured.
DSP_N1	12	Global configuration for DSP N1 Extension Support: 0: No DSP N1 Extension support; 1: Has DSP N1 Extension support. Note: Only take effect when DSP is configured.
SMP	11	Global configuration for SMP support: 0: No SMP support; 1: Has SMP support.
DCACHE	10	Global configuration for D-Cache support: 0: No D-Cache support; 1: Has D-Cache support.
ICACHE	9	Global configuration for I-Cache support: 0: No I-Cache support; 1: Has I-Cache support.
DLM	8	Global configuration for IDLM support: 0: No DLM support; 1: Has DLM support.
ILM	7	Global configuration for ILMsupport: 0: No ILM support; 1: Has ILM support.
NICE	6	Global configuration for NICE support: 0: No NICE support; 1: Has NICE support.
PPI	5	Global configuration for PPI support: 0: No PPI support; 1: Has PPI support.
FIO	4	Global configuration for FIO support: 0: No FIO support; 1: Has FIO support.
PLIC	3	Global configuration for PLIC support: 0: No PLIC support; 1: Has PLIC support.
CLIC	2	Global configuration for CLIC support: 0: No CLIC support; 1: Has CLIC support.
ECC	1	Global configuration for ECC support: 0: No ECC support; 1: Has ECC support.

continues on next page

Table 19.19 – continued from previous page

Field	Bits	Description
TEE	0	Global configuration for TEE support: 0: No TEE support; 1: Has TEE support.

19.5.26 micfg_info

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

Table 19.20: micfg_info register

Field	Bits	Description
Reserved	MXLEN-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 and the size should be power of 2: 0: 0Byte 1: 256 Bytes 2: 512 Bytes 3: 1KB 4: 2KB 5: 4KB 6: 8KB 7: 16KB 8: 32KB 9: 64KB 10: 128KB 11: 256KB 12: 512KB 13: 1MB 14: 2MB 15: 4MB 16: 8MB 17: 16MB 18: 32MB 19: 64MB 20: 128MB 21: 256MB 22: 512MB others: Reserved
Reserved	15:10	Reserved 0.

continues on next page

Table 19.20 – continued from previous page

Field	Bits	Description
IC_LSIZE	9:7	I-Cache line size: 0: No 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

19.5.27 mdcfg_info

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

Table 19.21: mdcfg_info register

Field	Bits	Description
Reserved	MXLEN-1:22	Reserved 0.
DLM_ECC	21	ECC support for the data local memory (DLM): 0: No ECC support; 1: Has ECC support.

continues on next page

Table 19.21 – continued from previous page

Field	Bits	Description
DLM_SIZE	20:16	Indicates the size of DLM and the size should be power of 2: 0: 0Byte 1: 256 Bytes 2: 512 Bytes 3: 1KB 4: 2KB 5: 4KB 6: 8KB 7: 16KB 8: 32KB 9: 64KB 10: 128KB 11: 256KB 12: 512KB 13: 1MB 14: 2MB 15: 4MB 16: 8MB 17: 16MB 18: 32MB 19: 64MB 20: 128MB 21: 256MB 22: 512MB others: Reserved
Reserved	15:10	Reserved 0.
DC_LSIZE	9:7	D-Cache line size: 0: No 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

continues on next page

Table 19.21 – continued from previous page

Field	Bits	Description
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

19.5.28 mtlbcfg_info

This CSR is used to show the TLB configuration information.

Table 19.22: mtlbcfg_info register

Field	Bits	Description
Reserved	MXLEN-1:22	Reserved 0.
DTLB_SIZE	21:19	DTLB size: 0: No DTLB 1: 1 entry 2: 2 entry 3: 4 entry 4: 8 entry 5: 16 entry 6: 32 entry 7: 64 entry
ITLB_SIZE	18:16	ITLB size: 0: No ITLB 1: 1 entry 2: 2 entry 3: 4 entry 4: 8 entry 5: 16 entry 6: 32 entry 7: 64 entry
Reserved	15:11	Reserved 0.
MTLB_ECC	10	Main TLB supports ECC or not: 0: No ECC support; 1: Has ECC support.

continues on next page

Table 19.22 – continued from previous page

Field	Bits	Description
MTLB_LSIZE	9:7	Main TLB line size: 0: No Main TLB 1: 1 entry 2: 2 entry 3: 4 entry 4: 8 entry 5: 16 entry 6: 32 entry 7: 64 entry
MTLB_WAY	6:4	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	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

19.5.29 mppicfg_info

This CSR is used to show the PPI configuration information.

Table 19.23: mppicfg_info register

Field	Bits	Description
PPI_BPA	MXLEN-1:10	The base physical address of PPI. It has to be aligned to multiple of PPI size. For example, to set up PPI 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.

continues on next page

Table 19.23 – continued from previous page

Field	Bits	Description
PPI_SIZE	5:1	Indicates the size of PPI and it should be power of 2: 0: Reserved 1: 1KB 2: 2KB 3: 4KB 4: 8KB 5: 16KB 6: 32KB 7: 64KB 8: 128KB 9: 256KB 10: 512KB 11: 1MB 12: 2MB 13: 4MB 14: 8MB 15: 16MB 16: 32MB 17: 64MB 18: 128MB 19: 256MB 20: 512MB 21: 1GB 22: 2GB others: Reserved
Reserved	0	Reserved 1.

Note: this CSR only exists when PPI is configured.

19.5.30 mfiocfg_info

This CSR is used to show the FIO configuration information.

Table 19.24: mfiocfg_info register

Field	Bits	Description
FIO_BPA	MXLEN-1:10	The base physical address of FIO. It has to be aligned to multiple of FIO size. For example, to set up the FIO 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.

continues on next page

Table 19.24 – continued from previous page

Field	Bits	Description
FIO_SIZE	5:1	Indicates the size of FIO and it should be power of 2: 0: Reserved 1: 1KB 2: 2KB 3: 4KB 4: 8KB 5: 16KB 6: 32KB 7: 64KB 8: 128KB 9: 256KB 10: 512KB 11: 1MB 12: 2MB 13: 4MB 14: 8MB 15: 16MB 16: 32MB 17: 64MB 18: 128MB 19: 256MB 20: 512MB 21: 1GB 22: 2GB others: Reserved
Reserved	0	Reserved 1.

Note: this CSR only exists when FIO is configured.

19.5.31 mdevb

This CSR is used to set device region base address in runtime software. If user already sets the device regions when configures the Core RTL, then the device region descripted by this CSR is a new one. If no device regions is configured in Core RTL, it is the only one when enables.

Table 19.25: mdevb register

Field	Bits	Description
Reserved	MXLEN-1:32	Reserved 0.
DEV_BPA	31:2	The base physical address of device region. The address is 4B aligned.
Reserved	1	Reserved 0.
ENA	0	Software device region enable control. 0: Disable (default); 1: Enable.

19.5.32 mdevm

This CSR is used to set device region address mask, it is used as a set of mdevb.

Table 19.26: mdevm register

Field	Bits	Description
Reserved	MXLEN-1:32	Reserved 0.
DEV_MASK	31:2	The address mask of device region and it is 4B aligned. Reset Value is 0.
Reserved	1:0	Reserved 0.

Note:

- As ENA of *mdevb* is to enable the region, if at this time value of *mdevm* is 0, it means all address space after DEV_BPA is device region. So user should program *mdevm* before *mdevb*.
- The higher bits of *mdevm* should be continuously 1, the left bits should be all 0. Technically the number of 0 means the size of this device region.
- The value of *mdevb* should be integer multiples of the size of this device region. So user should carefully set the *mdevb* and *mdevm*.
- A tip to check a new address is in device region or not: New Address & *mdevm* == *mdevb* & *mdevm*? Y, N.

19.5.33 mnocb

This CSR is used to set non-cacheable region base address in runtime software. If user already sets the non-cacheable regions when configures the Core RTL, then the non-cacheable region descripted by this CSR is a new one. If no non-cacheable regions is configured in Core RTL, it is the only one when enables.

Table 19.27: mnocb register

Field	Bits	Description
Reserved	MXLEN-1:32	Reserved 0.
NOC_BPA	31:2	The base physical address of non-cacheable region. The address is 4B aligned.
Reserved	1	Reserved 0.
ENA	0	Software non-cacheable region enable control. 0: Disable (default); 1: Enable.

19.5.34 mnocm

This CSR is used to set non-cacheable region address mask, it is used as a set of mnocb.

Table 19.28: mnocm register

Field	Bits	Description
Reserved	MXLEN-1:32	Reserved 0.
NOC_MASK	31:2	The address mask of non-cacheable region and it is 4B aligned. Reset value is 0.
Reserved	1:0	Reserved 0.

Note:

- As ENA of *mnocb* is to enable the region, if at this time value of *mnocm* is 0, it means all address space after NOC_BPA is non-cacheable region. So user should program *mnocm* before *mnocb*.

- The higher bits of *mnocm* should be continuously 1, the left bits should be all 0. Technically the number of 0 means the size of this non-cacheable region.
- The value of *mnocb* should be integer multiples of the size of this non-cacheable region. So user should carefully set the *mnocb* and *mnocm*.
- A tip to check a new address is in non-cacheable region or not: New Address & *mnocm* == *mnocb* & *mnocm*? Y, N.

19.5.35 mirgb_info

This CSR is used to show the IREGION configuration.

Table 19.29: mirgb_info register

Field	Bits	Description
Reserved	MXLEN-1:PA_SIZE+1	Reserved 0.
IRG_BASE_ADDR	PA_SIZE:10	IREGION Base Address
Reserved	9:6	Reserved 0.
IREGION_SIZE	5:1	Indicates the size of IREGION and it should be power of 2: 0: Reserved 1: 1KB 2: 2KB 3: 4KB 4: 8KB 5: 16KB 6: 32KB 7: 64KB 8: 128KB 9: 256KB 10: 512KB 11: 1MB 12: 2MB 13: 4MB 14: 8MB 15: 16MB 16: 32MB 17: 64MB 18: 128MB 19: 256MB 20: 512MB 21: 1GB 22: 2GB others: Reserved
Reserved	0	Reserved 1.

Note:

- This CSR always exists in Nuclei new verison Core IP, please check related Databook.
- Previously there is a CSR named msmpcfg_info using this CSR ID, as SMP configuration information is inside the IREGION, so msmpcfg_info is discarded.

19.5.36 mecc_lock

This CSR is used to show ECC lock information.

Table 19.30: mecc_lock register

Field	Bits	Description
Reserved	MXLEN-1:1	Reserved 0.
ECC_LOCK	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

19.5.37 mecc_code

This CSR is used to ECC code injection and ECC error simulation.

Table 19.31: mecc_code register

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

19.5.38 mtlb_ctl

This CSR is used to control Main TLB related features.

Table 19.32: mtlb_ctl register

Field	Bits	Description
Reserved	MXLEN-1:6	Reserved 0
DTLB_CANCEL_EN	5	DTLB change flow canceling enable control: 0: Disable canceling current accesses in DTLB when change flow happens; 1: Enable canceling current accesses in DTLB when change flow happens. Note: When MMU is not configured, this field is tied to 0.
ITLB_CANCEL_EN	4	ITLB change flow canceling enable control: 0: Disable canceling current accesses in ITLB when change flow happens; 1: Enable canceling current accesses in ITLB when change flow happens. Note: When MMU is not configured, this field is tied to 0.
TLB_DRAM_ECC_CHK_EN	3	Controls to check the ECC Code or not when core access to MTLB data rams. This bit can be turned on for injecting ECC errors to test the ECC handler. 0: Disable to check the ECC codes (Default) 1: Enable to check the ECC codes Note: When MMU is not configured, this field is tied to 0.
TLB_TRAM_ECC_CHK_EN	2	Controls to check the ECC Code or not when core access to MTLB tag rams. This bit can be turned on for injecting ECC errors to test the ECC handler. 0: Disable to check the ECC codes (Default) 1: Enable to check the ECC codes Note: When MMU is not configured, this field is tied to 0.
TLB_ECC_EXCP_EN	1	MTLB double bit ECC exception enable control. 0: Disable double bit ECC exception 1: Enable double bit ECC exception Note: When MMU is not configured, this field is tied to 0
TLB_ECC_EN	0	MTLB ECC enable control. 0: Disable ECC (default) 1: Enable ECC Note: When MMU is not configured, this field is tied to 0.

19.5.39 mfp16mode

This CSR is used to set 16 bit floating precision mode.

Table 19.33: mfp16mode register

Field	Bits	Description
Reserved	MXLEN-1:1	Reserved 0.
fp16mode	0	16 bit float precision mode: 0: normal mode(IEEE-2008 half precision), default value. 1: bfloat 16 mode.

19.5.40 shartid

Nuclei UX900 v2.8.0 or later core implments CSR shartid, the format and features of CSR shartid is same as mhartid, it is convenient for S-mode software to distinguish each hart in a SMP cluster.

20 User Extended Introduction

The Nuclei Hummingbird E603 processor core features the Nuclei Instruction Co-processor Extension (NICE) interface, enabling users to add their custom instructions. This capability is a core highlight for the E603, designed as an open academic and teaching platform to foster architectural innovation. The accompanying Nuclei Studio IDE includes a NICE Wizard tool to streamline the custom instruction development workflow. This document provides an introduction; for detailed information, please refer to the <Nuclei_NICE_Extension> document. The E603 open processor core and its comprehensive ecosystem are publicly maintained and available at https://github.com/Nuclei-Software/e603_hbird.

21 Appendix

- **Nuclei RISC-V IP Products**: https://www.nucleisys.com/product.php
- **Nuclei Spec Documentation**: https://nucleisys.com/download.php#spec
- **Nuclei RISCV Tools and Documents**: https://nucleisys.com/download.php
- **Nuclei Prebuilt Toolchain and IDE**: https://nucleisys.com/download.php#tools
- **NMSIS**: https://github.com/Nuclei-Software/NMSIS
- **Nuclei SDK**: https://github.com/Nuclei-Software/nuclei-sdk
- **Nuclei Linux SDK**: https://github.com/Nuclei-Software/nuclei-linux-sdk
- **Nuclei Software Organization in Github**: https://github.com/Nuclei-Software/
- **RISC-V MCU Organization in Github**: https://github.com/riscv-mcu/
- **RISC-V MCU Community Website**: https://www.rvmcu.com/
- **Nuclei riscv-openocd**: https://github.com/riscv-mcu/riscv-openocd
- **Nuclei riscv-gnu-toolchain**: https://github.com/riscv-mcu/riscv-gnu-toolchain