RISC-V Privileged Architecture

Allen Baum
Esperanto Technologies.
allen.baum@esperantotech.com

8th RISC-V Workshop
Barcelona, Spain
May 7, 2018

Esperanto Technologies

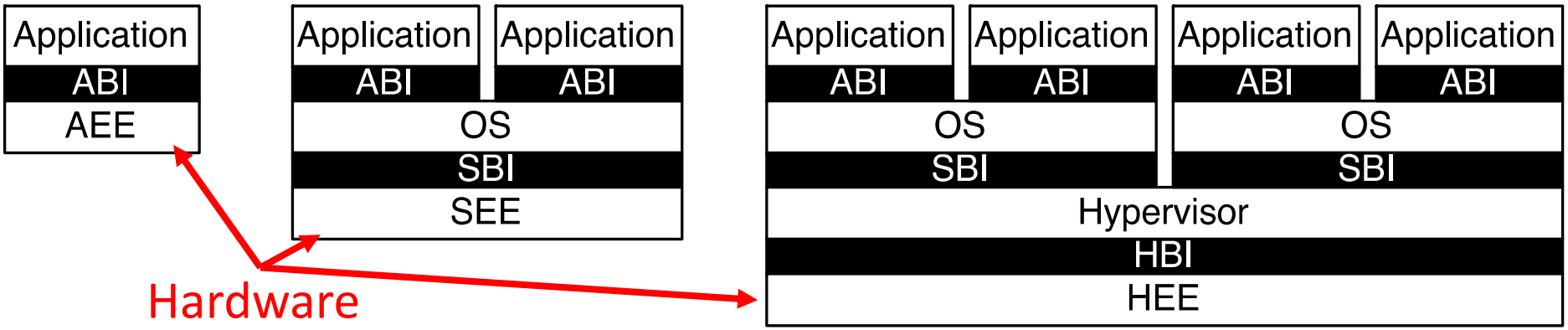
Introduction to RISC-V Privileged Architecture

- **Why a Privileged Architecture?**
- Profiles
- Privileges and Modes
- Privileged Features
  - CSRs
  - Instructions
- Memory Addressing
  - Translation
  - Protection
- Trap Handling
  - Exceptions
  - Interrupts
- Counters
  - Time
  - Performance

Why a Privileged Architecture?

- We need ways of managing shared resources
  - Memory
  - I/O Devices
  - Cores
- We need ways of protecting shared resources
  - Memory: use virtual memory mapping
  - I/O: also virtual memory mapping
  - Access permissions: integrated into mapping (or as separate functionality)
- We need ways of insulating implementation details
  - Trapping unimplemented ops for SW emulation
  - Handling external asynchronous events
    - (IO events, Timer events, SW interrupts from other threads
  - 2 Level address translation for Hypervisor support

RISC-V Privileged Architecture Layers



- Provides clean split between layers of the software stack

Layer	Communicates with	via
Application	Application Execution Environment (AEE)	Application Binary Interface (ABI)
Operating System	Supervisor Execution Environment (SEE)	System Binary Interface (SBI)
Hypervisor	Hypervisor Execution Environment (HEE)	Hypervisor Binary Interface (HBI)

- **ECALL** instruction used for the communication
- All ISA levels designed to support virtualization

- Why a Privileged Architecture?
- **Profiles**
- Privileges and Modes
- Privileged Features
  - CSRs
  - Instructions
- Memory Addressing
  - Translation
  - Protection
- Trap Handling
  - Exceptions
  - Interrupts
- Counters
  - Time
  - Performance

Profiles

Platform Profile Concept: Some basic profiles

Profile	Modes	Trust	Mem Protect	Other
Embedded without Protection	M	all Trusted	None	Low cost:16B each of arch. State/timers/counters
Embedded with Protection	M+ U	Apps untrusted	Phys Mem Protect	Optional N-extension for user int. handling
Unix-like OS capable	M+ S+U	OS Trusted	Vmem + RWX	Vaddr size options: 32,39,48b
Cloud OS capable	M+ [V](S+ U)	Hypervisor Trusted	2-level Vmem + RWX	Unix+ (Supports >1 OS) +new/background CSRs

RiscV has rich set of architectural modes & optional features

A profile is restricted combination of all possible options

See https://github.com/riscv/riscv-platform-specs

- Why a Privileged Architecture?
- Profiles
- **Privileges and Modes**
- Privileged Features
  - CSRs
  - Instructions
- Memory Addressing
  - Translation
  - Protection
- Trap Handling
  - Exceptions
  - Interrupts
- Counters
  - Time
  - Performance

Privileges and Modes

RISC-V Privilege Modes

- User & 2+ privileged modes (hierarchical)
  - User (U-mode), normal **and** virtualized* (lowest privileges)
  - Supervisor (S-mode), normal **and** virtualized*
  - Machine (M-mode) (highest privileges)**
- Supported combinations of modes:
  - M (simple embedded systems)
  - M, U (embedded systems with protection)
  - M, S, U (systems running Unix-like operating systems)
  - M, [V]S, [V]U (systems running multiple Oses)
- Each privileged mode add a few ops, and Control/Status Registers (CSRs) that control operations
  - CSRs accessible only by code running at a specific privilege mode or higher
  - There are (often) multiple CSR copies/views for each mode

**Virtualized modes for hypervisor support- not covered here*

*** A higher (Debug) mode exists: only entered if debug port connected & enabled, has separate state saving CSRs, but otherwise much like M-mode*

- Why a Privileged Architecture?
- Profiles
- Privileges and Modes
- **Privileged Features**
  - CSRs
  - Instructions
- Memory Addressing
  - Translation
  - Protection
- Trap Handling
  - Exceptions
  - Interrupts
- Counters
  - Time
  - Performance

Privileged Features: Instructions and CSRs

Mode Specific Instructions

Privileged Machine (M-mode) & Supervisor (S-mode) add instructions to base U-mode ops

- Priv Insts can only be executed from appropriate mode (**or higher**)

- All modes
  - **ECALL**: generates <*curr_mode*>environment_call exception
  - **EBREAK**: generates breakpoint exception
  - **FENCE[.I]**: synchronizes updates to memory
  - **<*x*>RET**: returns from a trap *from* the specified mode
    - **SRET** provided only if S-mode is implemented
    - **URET** provided only if U-mode traps supported (N-extension)
- S-mode (+M-mode): adds
  - **SFENCE.VMA**: synchs updates to implicitly accessed memory
- M-mode: adds
  - **WFI**: stall the current hart until an interrupt needs service
    - Is a hint only (could be noop,

Mode Specific CSRs

- Control/Status Regs (CSRs) have their own address space
  - Direct address mode only
- Each hart has its own set of 4K CSRs (1K/mode)
- CSRs are accessed by dedicated ops
  - that can implement atomic swap or bit set/clear
- CSRs are mode sensitive
  - Can only be accessed by code in appropriate or higher privileged mode; accesses by lower privilege modes will trap
- Many CSRs optional/ have optional fields/mode dependent
  - Accesses to non-existent CSRs will trap
  - Writes to Read_Only CSRs will trap
    - But writes to read_only fields in read/write CSRs are ignored
  - Accesses to optional CSRs read zeroes, & (if RW) ignore writes
    - Note that optional vs. non-existent can depend on architecture!

CSR address space

CSR Address [11:10]	CSR Address [7:6]	Machine-Mode [9:8]=11	Supervisor-Mode [9:8]=01	User-Mode [9:8]=00	Binary Encoding
00	XX				0b00MM xxxx xxxx
01	0X				0b01MM 0xxx xxxx
01	10				0b01MM 100x xxxx
01	10	Debug only			0b01MM 101x xxxx
01	11	← NonStandard →			0b01MM 11xx xxxx
10	~11				0b10MM ~11xx xxxx
10	11	← nonStandard →			0b10MM 11xx xxxx
11	~11	← RdOnly →			0b11MM ~11xx xxxx
11 (RO)	11	← NonStandard	Rd Only →		0b11MM 11xx xxxx

Addr[9:8]==10 currently reserved for Hypervisor CSs

Read Only | Non_Standard | Debug

CSRs and categories

Category	CSR Name (some replicated/mode)
FP CSRs	Exceptions, Rounding_Mode, Reg_State
Information	Vendor/ Architecture/Implementation/Thread_IDs
Protection/ Translation	Address_Translation_Protection, PhysMemProtection Config[ ]/Addr[ ]
Trap Setup	Status, ISA+Extension, Int/Excep_Delegation, Trap_Vector, Int_Enable, Cntr_Enab
Trap Handling	Exception_PC, Scratch, Int_Pending Trap_Cause/Value
Counter/ Timers	Cycles, Inst_Retired, Time PerfmonCntr [ ]
Counter Setup	Perfmon Event selector[ ]
Debug/ Trace	Control/Status/PC/Scratch Trigger_RegSelect/Data[ ]

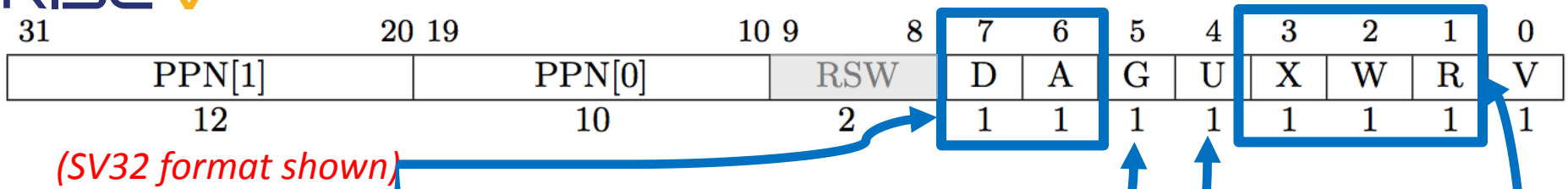
- Why a Privileged Architecture?
- Profiles
- Privileges and Modes
- Privileged Features
  - CSRs
  - Instructions
- **Memory Addressing**
  - **Translation**
  - **Protection**
- Trap Handling
  - Exceptions
  - Interrupts
- Counters
  - Time
  - Performance

Memory Addressing: Translation

Memory Address Translation: Virtual Memory

- S-mode adds virtual memory page mapping
  - Smallest unit of protection/mapping is 4 KB pages
- Supports multiple User mode processes w/ separate address spaces (using Addr_Space_ID field in SATP CSR)
- Page tables have multiple levels that are walked:
  - 2 levels for RV32 (Sv32)
  - 3,4 levels for RV64 (Sv39, Sv48)
  - 5,6 levels RSVD (Sv57, Sv64)
- Page Table walk can stop at any level to create Superpages
  - e.g. for Sv39 2 MB if stopped at 2 levels
  - or 1 GB if stopped at 1 levels
- HW Page Table walk semantics specified in Priv Mode spec
  - But could trap to M-mode for software TLB refill

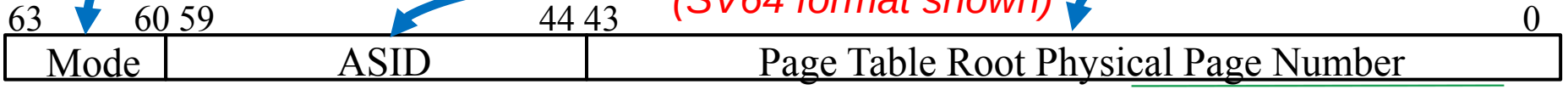
RISC-V Page Table Entries



(SV32 format shown)

- **A**ccessed/**D**irty bits optionally HW managed
  - Updates must be atomic w.r.t. permissions check
  - Complex to implement, so trap if A/D is clear
- **G**lobal bit indicates mapping belongs to all addr spaces (e.g. Unix systems kernel pages
- Page granularity permissions (**U**ser/**R**ead/**W**rite/e**X**ecute)
  - (000 XWR indicates a non-leaf entry)
- Virtual Addr width, Current **ASID**, and **PageTable** root controlled by SATP CSR

(SV64 format shown)



More control: Memory Fences

- S-mode implements **SFENCE.VMA** instruction to synchronize updates to memory data structures
  - All page table levels, or just those corresponding to an addr
  - All address spaces, or just a specific address space (not global)
- Generalization of TLB flush on other architectures
- Guarantees that all prior stores are ordered before all subsequent implicit references in the instruction stream
- Affects only the local hart
  - Synchronization with other harts requires IPIs

- Why a Privileged Architecture?
- Profiles
- Privileges and Modes
- Privileged Features
  - CSRs
  - Instructions
- Memory Addressing
  - Translation
  - **Protection**
- Trap Handling
  - Exceptions
  - Interrupts
- Counters
  - Time
  - Performance

Memory Addressing: Protection

RISC-V Virtual Memory Protections

- Standard **RWX** permissions configurable for every page
  - Supports **X**-only pages
  - **W** & ~**R** combination reserved
- By default, S-mode can't access user pages
  - Helps detect OS/driver bugs
  - Still need ability to read user memory, e.g. on system call
  - Set "Supervisor Access to User Memory" (**SUM**) bit in sStatus to read user memory, then turn it off again
  - S-mode cannot execute from U-mode pages even if **SUM**=1
- Similarly, S-mode can't read e**x**ecute-only pages
  - Set sStatus "Make e**X**ecutable Readable" **MXR** bit to override
  - Useful for illegal-instruction trap handlers
- S-mode can enable/disable VM and choose page-table depth in SATP register

RISC-V Physical Memory Protection Unit

- Optional new feature in v1.10
- Makes S/U-modes have no permissions by default
- Grants **R**/**W**/**X** permissions to up to 16 PMP regions
  - Naturally aligned 2^N-byte regions (N>=2)
  - Or use adjacent PMP regs to form base/bounds region
- PMPs can be *locked*
  - Affects M-mode also
  - Only a reset can unlock
- The fine print:
  - If VM enabled, VM (& page faults) occur before PMP checks
  - Useful for untrusted S-Mode

Physical Memory Attributes

- RISC-V systems have the concept of Physical Memory Attributes: *platform and implementation specific*
  - Maps access to a bus transaction type, or an error
- PMA is dedicated HW that maps specific address ranges to certain access attributes, e.g.
  - Access widths allowed (e.g. 1/2/4/8/16/64B)
  - Alignment restrictions (e.g. can't cross 2^x byte boundary)
  - Idempotency (enabling speculation)
  - Ordering (Strong/Weak per Channel)
  - Cacheability (incl Wt Thru, Wt Combining, etc)
  - Priority (e.g. high/low if conflicting)
  - Atomicity allowed (none, swap, logical, arithmetic)
  - Allowed access modes (M/S/U/debug)
- Some attributes could be configurable

- Why a Privileged Architecture?
- Profiles
- Privileges and Modes
- Privileged Features
  - CSRs
  - Instructions
- Memory Addressing
  - Translation
  - Protection
- **Trap Handling**
  - **Exceptions**
  - **Interrupts**
- Counters
  - Time
  - Performance

Trap Handling: Exceptions and Interrupts

Interrupts vs Exceptions

- Exceptions: Synchronous events
  - Synchronous: caused by a specific instruction execution
- Interrupts: Asynchronous events
  - not caused by an inst: I/O, timer, SW (from another hart)
- Both handled (almost) identically by trapping:
  - **xTVEC** CSR holds handler address
    - Interrupts optionally vector to **xTVEC**+4*xCause
  - **xI/EDELEG** CSRs: select mode to trap into (next slide)
  - **xCause** CSR (x=new mode) saves cause ID
    - MSB: interrupt vs. exception, LSBs: interrupt /exception ID code
  - **xTVAL** CSR saves additional information about cause
    - This could be an illegal address, or illegal opcode *(not for int)*
  - **xEPC** CSR saves return Program Counter
    - could be next instruction (interrupts) or same inst (enabling retry)
  - **xSTATUS** CSR saves curr Mode/**IntEn** bits
    - **xSTATUS[IntEn]** cleared

Where to trap

Mode to trap into

Reason for trap

How to return from trap

Trap setup

Trap handling

Trap Setup
Interrupt/Exception Handler Delegation

- Traps always sent to M-mode, but…
- Can be delegated to lower priv level, reducing overhead
  - Never to a less privileged mode than the one that trapped!
- Bits in delegation CSR send traps to next lower priv level
  - **m[i/e]deleg:** M→S (or M→U if no S-mode & N_extension)
  - **s[i/e]deleg:** S→U (if delegated to S-mode & N_extension)
- Int Delegation occurs only if corresponding enable bits set (**<x>ie** CSR)
  - But enable bit used only for delegated (higher mode)
  - Exceptions are always enabled
- Interrupts that trap set corresponding **<x>ip** CSR bit

Trap Handling
Interrupt/Exception Causes

- **<x>cause** CSR indicates which interrupt/exception occurred
- Corresponding bit is set in **<x>E/IP** CSR

Trap Priority for simultaneous interrupts/exceptions:

Interrupts > Exceptions
M-mode > S-mode > U-mode
Pending[N] > Pending[M] if N>M

Trap code[62:0]	Exception (Cause[MSB]=0)	Interrupt (Cause[MSB]==1)
0	Instruction addr misaligned	User Software Interrupt
1	Instruction access fault	Supervisor Software Interrupt
2	Illegal instruction	Reserved
3	Breakpoint	Machine Software Interrupt
4	Load address misaligned	User Timer Interrupt
5	Load access fault	Supervisor Timer Interrupt
6	Store/AMO addr misaligned	Reserved
7	Store/AMO access fault	Machine Timer Interrupt
8	Environment call	User External Interrupt
9	Reserved	Supervisor External Interrupt
10		Reserved
11		Machine External Interrupt
12	Instruction page fault	Reserved
13	Load page fault	
14	Reserved	
15	Store/AMO page fault	
>=16	Reserved	Reserved

Local

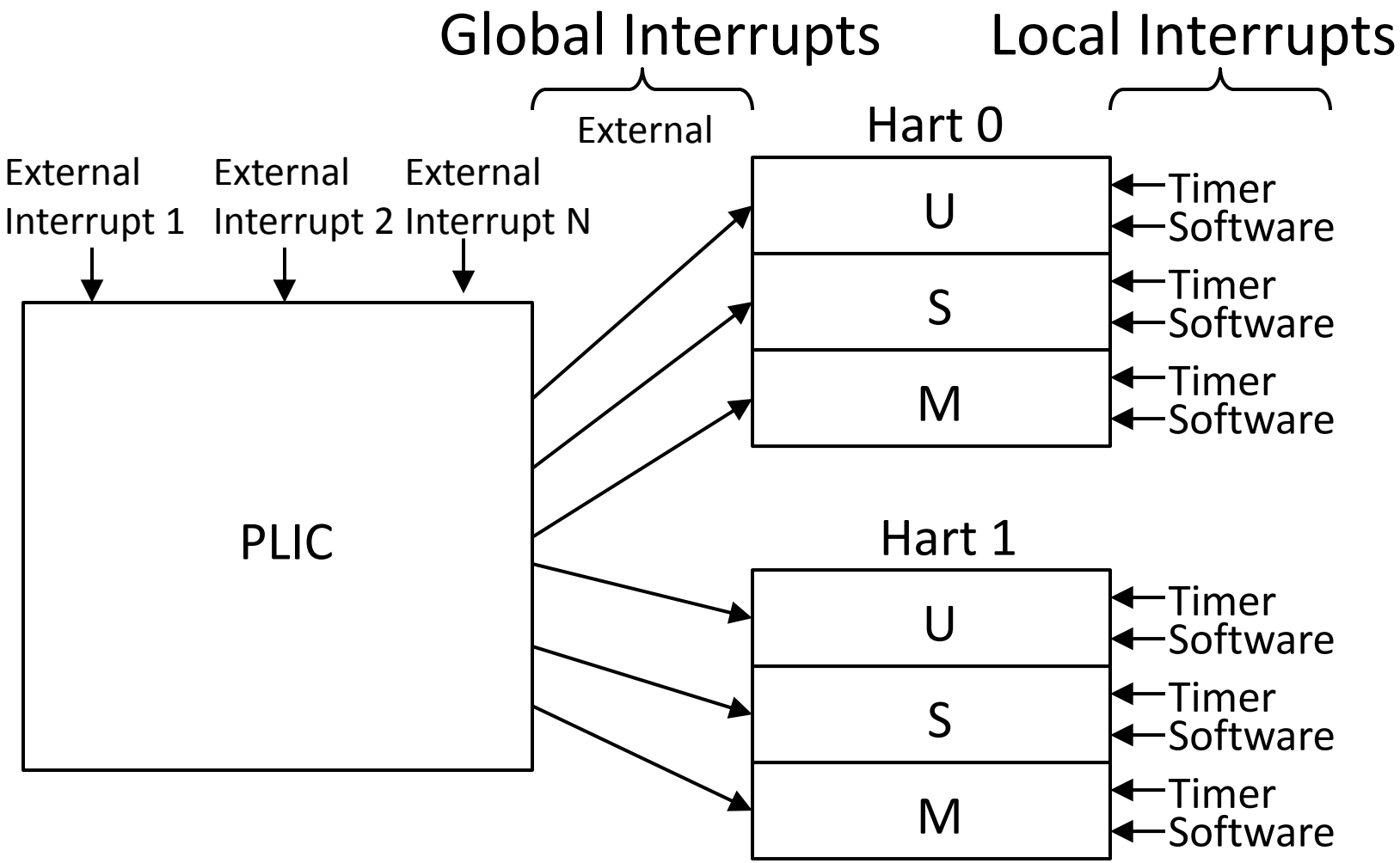
External

Special case: Timer & SW interrupt priorities swapped!

- Why a Privileged Architecture?
- Profiles
- Privileges and Modes
- Privileged Features
  - CSRs
  - Instructions
- Memory Addressing
  - Translation
  - Protection
- Trap Handling
  - Exceptions
  - **Interrupts**
- Counters
  - Time
  - Performance

Interrupts

Platform-Level Interrupt Overview



RISC-V Interrupt Source Categories

- Global (External) Interrupts
  - Routed to harts via Platform-Level Interrupt Controller (PLIC)
  - Actual source determined by read of PLIC MMIO CSR
- Local Interrupts
  - Directly connected to one hart, independent of other harts
  - Cause determined directly from **<*x*>cause** CSR
  - Only two standard local interrupts (software, timer)
- Any interrupt can target **any** M/S/U mode
  - Except for priority during simultaneous interrupts, handling is identical

External Interrupts

- Inputs from a Platform-Level Interrupt Controller (PLIC)
  - PLIC targets hart based on hart interrupt threshold & enable, and interrupt priority
- Interrupts can target multiple harts simultaneously
  - Harts must arbitrate to determine which services it
  - E.g. by racing to read MMIO mapped interrupt source CSR
- PLIC labels each output with a privilege mode
  - Which can be handled differently using delegation
- Interrupts cleared via MMIO mapped LD/ST to PLIC
- Software can inject SEIP and UEIP interrupts to support virtualizing the PLIC by writing CSR directly

Software Interrupts

- Software interrupt are how harts interrupt each other
  - Mechanism for inter-hart interrupts (IPIs)
- Setting the appropriate <x>SIP bit in another hart is performed by a MMIO write
  - But a hart can set its own <x>SIP bit if currmode >= <x>
- App/OS performs inter-hart ints only via ABI/SBI calls
  - Destination virtual hart might be descheduled
  - Interrupts virtualized by M-mode software using MSIP

Timer Interrupts

- Single M-mode 64b real-time HW timer per system
- Single M-mode 64b Time comparator per hart (logically)
- **NOT** CSRs, but MMIO addressed
  - Must count at a fixed rate, regardless of core clock or power
  - **mtime** >= **mtimecmp** causes hart's **MTIP** bit to be set
- M-mode responsible for virtualizing the single HW timer and hart comparator for lower-privilege modes
  - CSR reads by U-mode will trap & be handled by M-mode
- **STIP** and **UTIP** CSR bits are handled by M-mode
  - ABI/SBI calls to set up timer
  - M-mode software writes/clears **STIP,UTIP**

- Why a Privileged Architecture?
- Profiles
- Privileges and Modes
- Privileged Features
  - CSRs
  - Instructions
- Memory Addressing
  - Translation
  - Protection
- Trap Handling
  - Exceptions
  - Interrupts
- **Counters**
  - **Time**
  - **Performance**

Counters: Time and Performance

Timers and Counters

- RISC-V has several architected Timers and Counters implemented (mostly) as CSRs
- All are 64 bits (split into 2 CSRs for RV32 only)
  - Real Time Clock `Time`, described in Timer Interrupt slide
    - U/S-mode read of CSR traps to M-mode, which does MMIO read
  - Instructions_Retired `InstRet` Counter
    - M-mode RW, U-mode RO, used for **RDINSTRET** pseudo-instructions
  - Cycles `Cycle` Counter
    - M-mode RW, U-mode RO, used for **RDCYCLE** pseudo-instructions
  - 0..29 HW Performance Monitors `mhpmcounters`
    - each w/corresponding `HPMEvent` to select what to count

Timer/Counter protections

- Easily accessible timers have issues
  - Lack of reproducibility
  - Side channel security attacks (Meltdown, Spectre…)
- <x>CounterEn CSRs enables access to the counters
  - 1 bit per counter (Time/Cycle/InstRet/HPMCounter[])
  - Accessing <**x**>timer/counter in a mode<**x** will trap if corresponding bit in <x>CounterEn is clear for x<y
  - Any bit may be optionally hardwired to zero

Wrap Up

Privileged Architecture is Stable

- Latest version is v1.11 draft
- keeps compatibility with v1.9.1 for machine-mode-only implementations
- Future releases should be compatible with v1.10 for supervisor ISA, too
- Adds draft Hypervisor support
- Caveat: these are proposals; not yet ratified by Foundation

Implementation Status

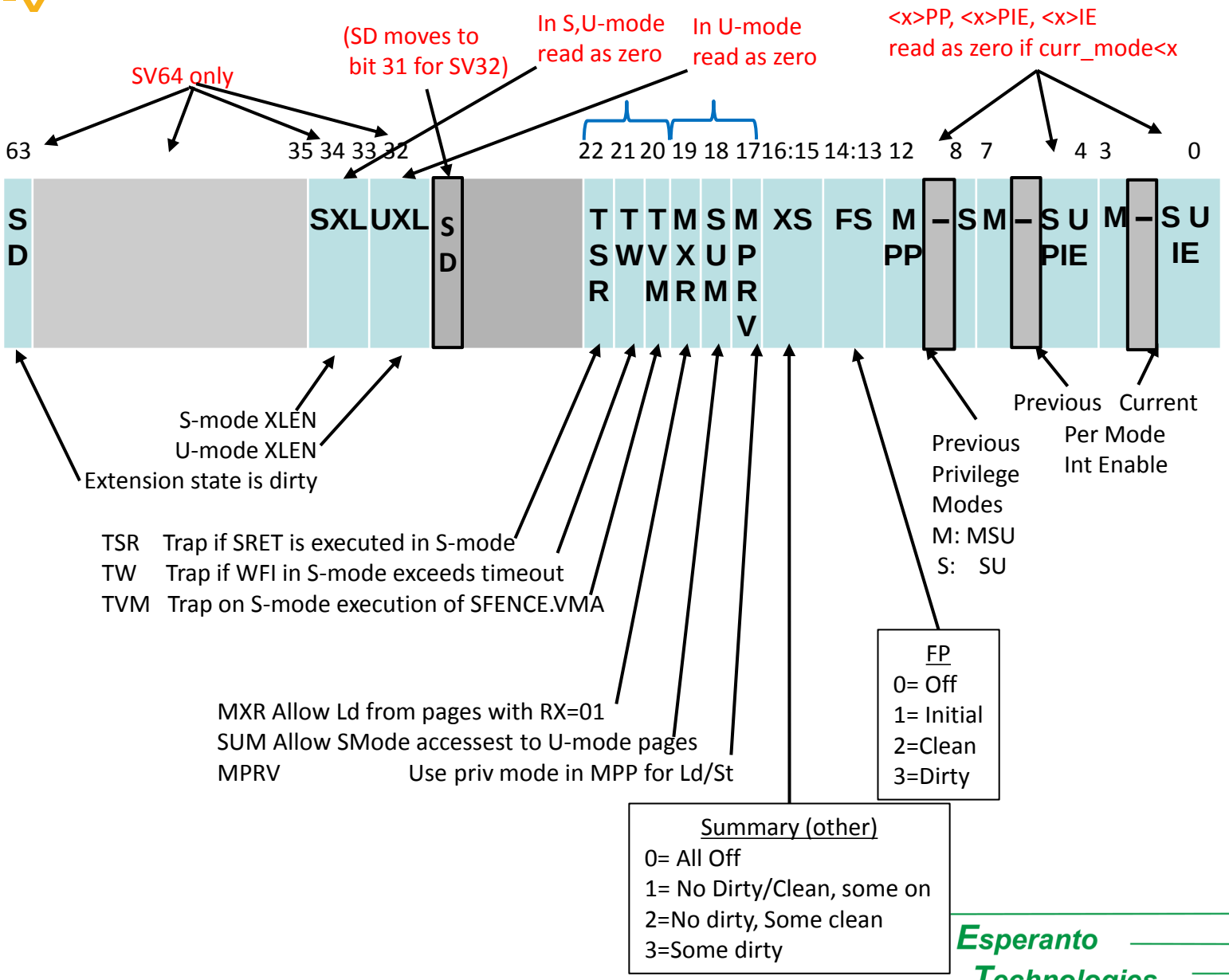
- Spike and UCB Rocket-Chip conform to v1.11
- Linux port is upstreamed and conforms to v1.11
  - works with Spike/Rocket
- QEMU port is upstreamed and conforms to v1.11
- Upstream GCC and binutils ports are compatible

Questions?

Specs available at
https://github.com/riscv/riscv-isa-manual

Backup

<x>Status CSR


SV64 only
(SD moves to bit 31 for SV32)
In S,U-mode read as zero
In U-mode read as zero
<x>PP, <x>PIE, <x>IE read as zero if curr_mode<x
63
35 34 33 32
22 21 20 19 18 17 16:15 14:13 12
8 7
4 3
0
SD
SXL UXL
SD
TSR
TW
TVM
MXR
SUM
MPRV
XS
FS
MPP
– S M – SUPIE
M – SUIE
S-mode XLEN
U-mode XLEN
Extension state is dirty
Previous
Current
Per Mode
Int Enable
Previous Privilege Modes
M: MSU
S: SU
TSR Trap if SRET is executed in S-mode
TW Trap if WFI in S-mode exceeds timeout
TVM Trap on S-mode execution of SFENCE.VMA
MXR Allow Ld from pages with RX=01
SUM Allow SMode accessest to U-mode pages
MPRV Use priv mode in MPP for Ld/St
FP
0= Off
1= Initial
2=Clean
3=Dirty
Summary (other)
0= All Off
1= No Dirty/Clean, some on
2=No dirty, Some clean
3=Some dirty

CSRs, Privileged Modes, & Options

CSR Category	CSR Name	Comments	M-mode	S-mode	Umode
Floating-Point CRs	Accrued Exceptions				X(DF)
	Dynamic Rounding Mode				X(DF)
	Ctl & Status Reg (frm + fflags)				X(DF)
Information	Vendor ID (o)	Encoded JEDEC ID	X		
	Architecture ID (o)	MSB==Commercial	X		
	Implementation ID (o)		X		
	Hardware_thread_ID	Hart 0 must exist			
Trap Setup	Status (S)		X	HBX (H)	X
	ISAs and Extensions		X		
	Exception_Delegation		X (SN)	BX (H,N)	
	Interrupt_ Delegation		X (SN)	BX (H,N)	
	Interrupt Enable (R)		X	BX (H)	X (UN)
	Trap_Vector_Base_Address (z)		X	BX (H)	X (UN)
	Counter Enable (Z)		X	X	
Trap Handling	Scratch Register	For Trap handlers	X	BX (H)	X (UN)
	Exception_Program_Counter		X	BX (H)	X (UN)
	Trap_Cause (P,Z)		X	BX (H)	X (UN)
	Trap_Value (Z)		X	HBX	X (UN)
	Interrupt pending (R)		X	BX (H)	X (UN)
Protection/Translation	Addr Translation & Protection(RZ)			BX (H)	
	Phys Mem Prot Config[3:0] (on)		X		
	Phys Mem Prot Addr[15:0] (on)		X		
Counter/Timers	Cycle_counter	For RDCYCLE inst	X		X
	Time	For RTIME inst			X
	Instr-retired_counter	For RDINSTRET inst	X		X
	Perfmon counters[31:3] (Zn)		X		X
	Upper_32b_of_cycle		X (I)		X (U32)
	Upper_32b_of_time				X (U32)
	Upper_32b_of_instret		X (I)		X (U32)
	Upper_32b_of_perfmon[31:3] (Z)		X (I,n)		X(U32n)
Counter Setup	Perfmon Event selector[31:3] (Z)		X(n)		
Debug/Trace	Debug/Trace Trigger Reg Select		X		
	Debug/Trace trigger data Reg[3:1]		X		
	Debug Control/Status		X		
	Debug PC		X		
	Debug Scratch Reg		X		

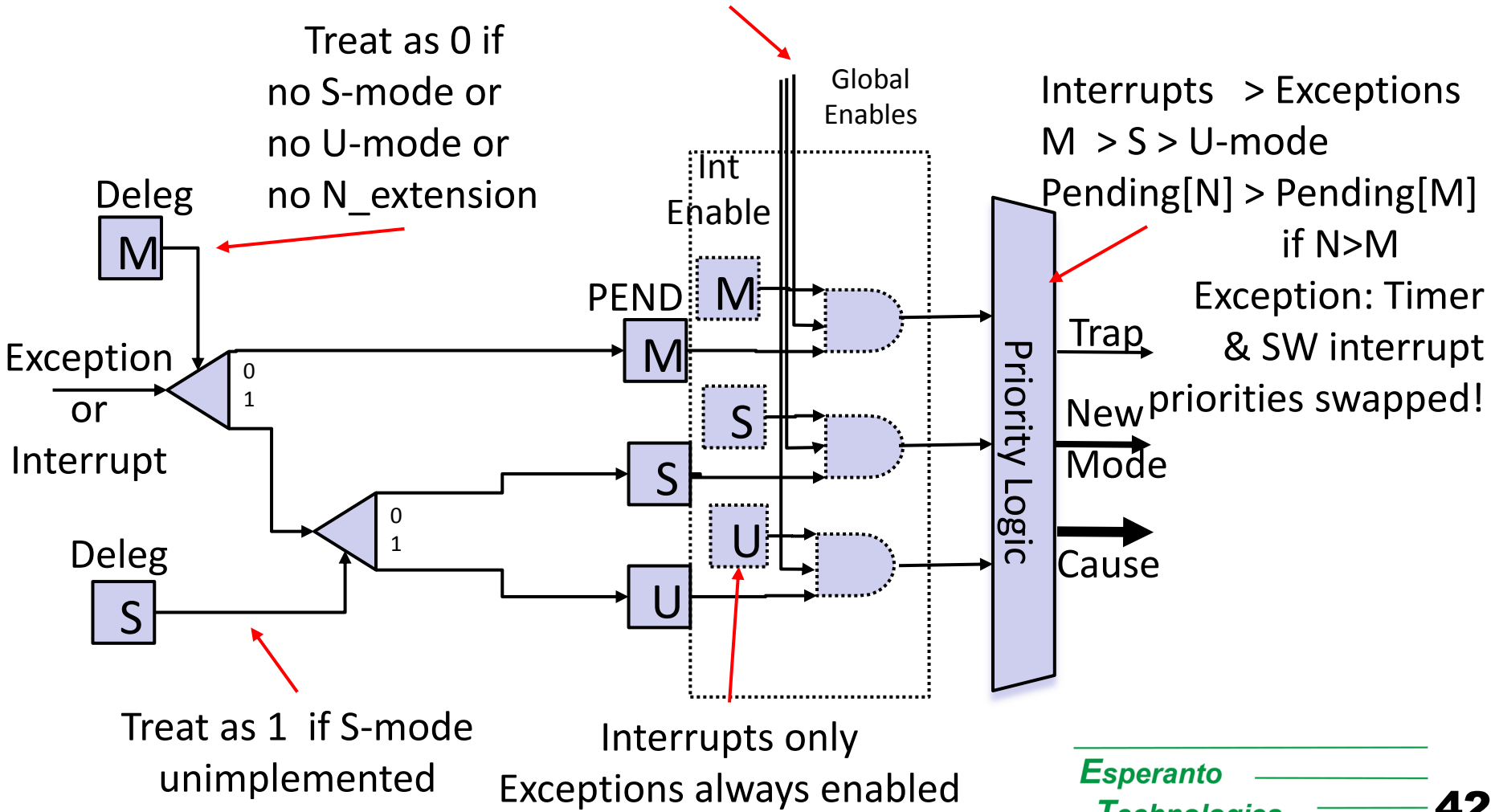
(DF) Optional unless D/F extension implemented
(UN) Exists only in Umode & N-extension implemented
(U32)Exists only in Umode & RV32I architecture
(n) Exact number is implementation dependent
(I) exist only if RV23I architecture
(R) Single register with restricted view
(P) Only bits corresponding to <= curr mode are visible
(H) Swapped with background CSR on VM entry/exit
(o) Optional (Read 0 if unimplemented)
(SN)Exists only if Smode | N-extension implemented
(z) Some bits may be hardwired RdOnly
(Z) May be hardwired RdOnly zero

* = optional (read as zero)
Types of CSR fields:
- WIRI (Reserved) Writes ignored, Reads ignored
- WPRI (Reserved) Writes preserved, Reads ignored
- WLRL Write Legal, Read Legal
- WARL Write Any, Read Legal

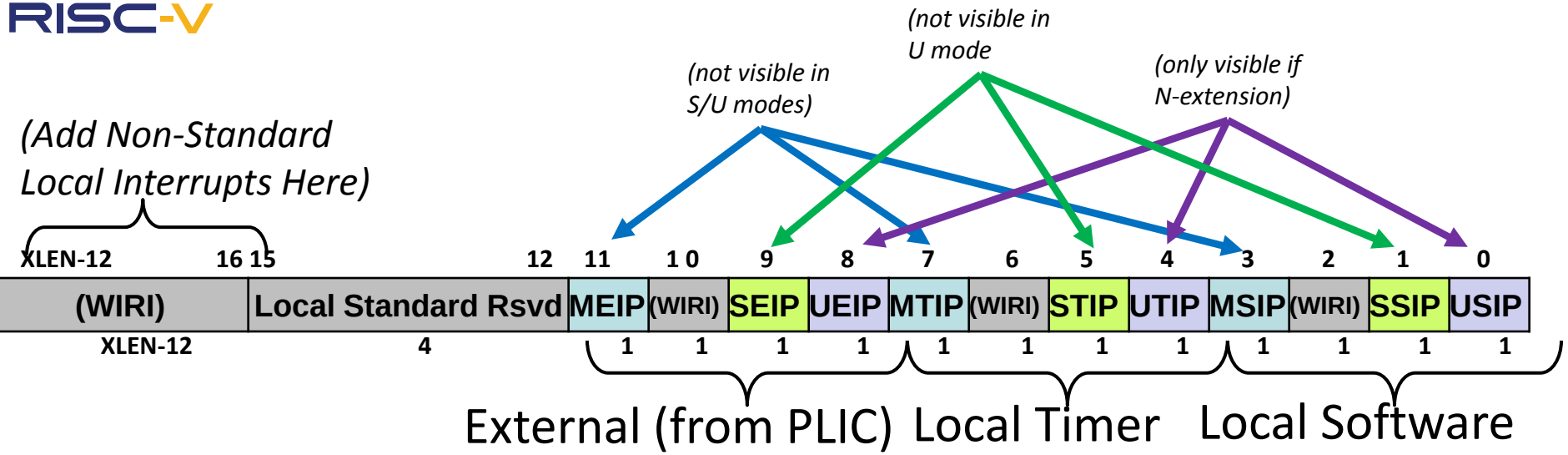
H= new CSR version added for Hypervisor extension
B = background CSR added for Hypervisor extension

Interrupt/Exception Handler Delegation

Global Enable M = currMode<M + currMode==M & Mstatus.MIE
Global Enable S = currMode<S + currMode== S & Mstatus.SIE
Global Enable U = currMode<U + currMode== U & Mstatus.UIE

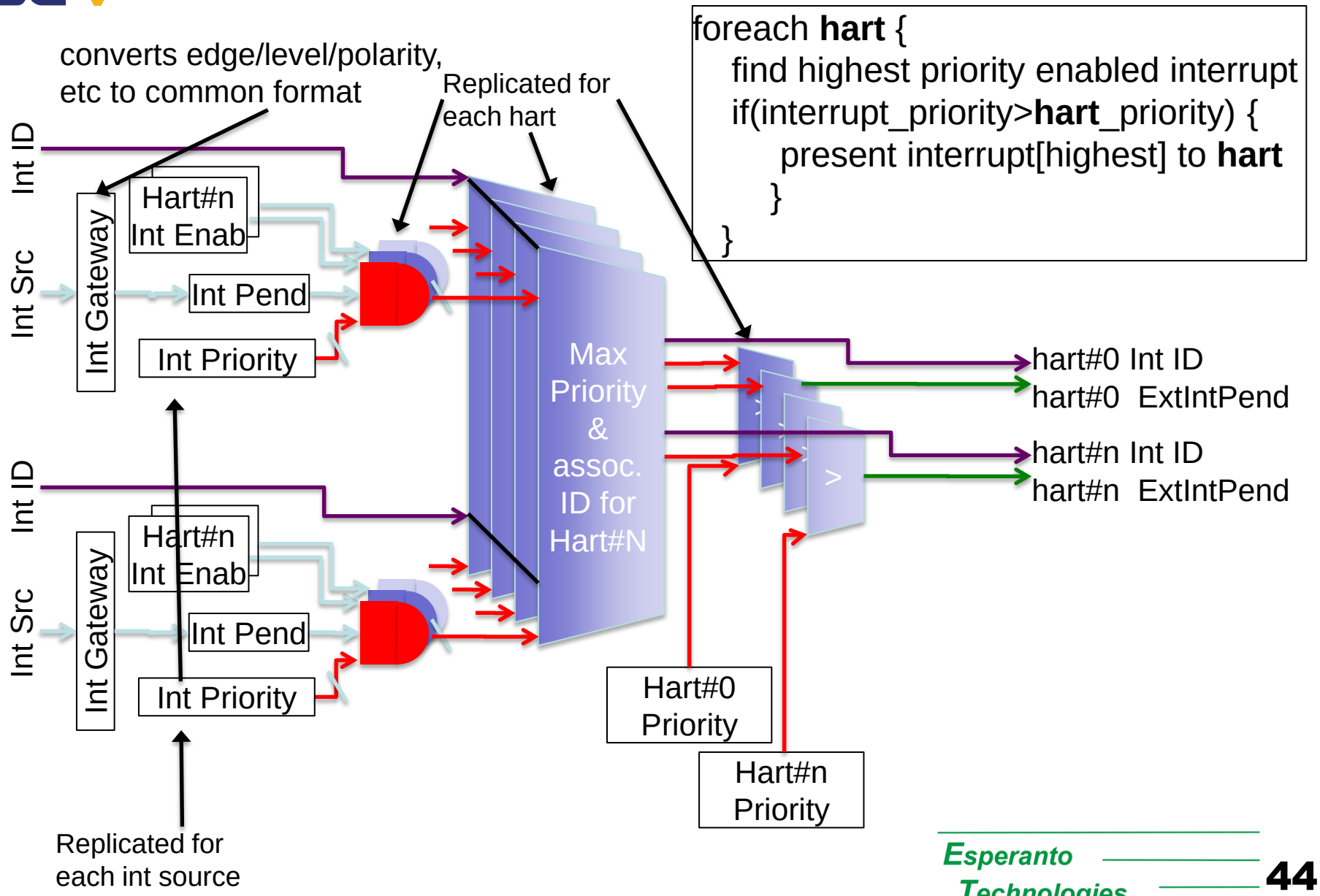


Interrupt Pending/Enable CSRs <x>ip,ie



- **<x>ip** reflects pending status of interrupts for hart
  - Enabled by corresponding bits of **<x>ie** with same per/mode visibility
  - In addition to global interrupt enables in **<x>status** for each privilege mode
- Separate ints for each priv level (M/S/U), directed to M-mode
  - M-mode can delegate to S-mode and U-modes
  - Higher privilege modes override lower privilege modes
- Opt. User interrupt handling ("N") feature when U-mode present
- Interrupts always disabled for privimodes lower than current mode; always enabled for privilege modes higher than current mode

PLIC Block Diagram



Hypervisor mode

- Feedback led us to HW support for Type-2 hypervisors (like KVM)
  - Can also support type-1
- Hypervisors run in S-mode
- Guests run in virtualized (V) S and U modes
  - Major difference is 2-level page table walk
  - Certain operations can be inhibited or trap (individually)
    - Execution of WFI if the wait exceeds some limit can be trapped
    - SRET ,FENCE.VMA, SATP and counters CSR accesses can be trapped
      - But CYCLE and INSTRET will still count
    - Force translation to use supervisor previous priv level
  - Additional bits added to STATUS CSR
    - Previous Virtualization mode
    - Translation fault level
  - Some control bits interpreted differently: SPRV, SPV, SPP
- Vmode changes cause CSR swap/selection w/background versions
  - Use HSTATUS, HEDELEG, HIDELEG, HTVAL
  - Swaps SSTATUS, SIE,SIP, STVEC, SSCRATCH, SCAUSE, SEPC, STVAL, SATP