**Module: hal\_core\_interrupts**

**Overview**

The hal\_core\_interrupts module is responsible for managing hardware and software **interrupts** within the **Hardware Abstraction Layer (HAL)**. It provides a unified interface for handling **interrupt requests (IRQs)**, **exceptions**, and **interrupt service routines (ISRs)** across different processors and architectures.

This module ensures efficient handling of interrupts, enabling **low-latency** responses, **real-time processing**, and **prioritized execution** for critical hardware events.

**Key Responsibilities of hal\_core\_interrupts**

1. **Interrupt Vector Table (IVT) Management**
   * Maintains the interrupt vector table that maps **interrupt sources** to corresponding **interrupt handlers (ISRs)**.
   * Supports **dynamic** ISR registration and modification.
2. **Interrupt Prioritization and Preemption**
   * Assigns **priority levels** to different interrupts to ensure time-sensitive tasks are executed first.
   * Supports **nested interrupt handling**, where higher-priority interrupts can **preempt** lower-priority ones.
3. **Interrupt Routing and Multiplexing**
   * Routes interrupts from different devices to the appropriate **processor cores** in **multi-core** or **heterogeneous** architectures.
   * Handles **shared interrupts** for devices that trigger the same IRQ line.
4. **Exception Handling and Fault Recovery**
   * Detects and handles **CPU exceptions** (e.g., **division by zero**, **access violations**, **illegal instructions**).
   * Provides a mechanism for **graceful recovery** in case of errors.
5. **Software-Generated Interrupts (SGIs) & Inter-Processor Communication (IPC)**
   * Enables software-triggered interrupts for inter-thread or inter-process communication.
   * Provides mechanisms for **Inter-Processor Interrupts (IPIs)** in multi-core systems.
6. **Low-Power Interrupt Processing**
   * Supports **wake-up interrupts** from low-power or sleep states.
   * Ensures efficient **power management** by only waking up the processor for critical events.
7. **Real-Time & Deterministic Interrupt Handling**
   * Provides support for **real-time systems** with deterministic interrupt latency.
   * Enables use in **RTOS-based environments** where timing is crucial.

**Interrupt Handling Workflow**

**1. Interrupt Occurrence**

* A **hardware device**, **timer**, or **software event** triggers an interrupt.
* The CPU **pauses the current execution** and switches to the **interrupt handling routine**.

**2. Interrupt Controller Receives & Routes the Interrupt**

* The **Interrupt Controller (IC)** (e.g., **GIC** for ARM, **APIC** for x86) identifies the source of the interrupt.
* It checks **priority levels** and determines whether the interrupt should be **processed immediately** or deferred.

**3. Interrupt Service Routine (ISR) Execution**

* The system jumps to the **ISR** registered for the interrupt source.
* The ISR executes the required task (e.g., handling a keypress, processing a network packet).
* The ISR **acknowledges** or **clears** the interrupt after execution.

**4. Return from Interrupt (RTI)**

* The system **restores the previous CPU state** and resumes normal execution.
* If there are **pending higher-priority interrupts**, they are executed before returning to normal execution.

**Key Components of hal\_core\_interrupts**

| **Component** | **Description** |
| --- | --- |
| **Interrupt Vector Table (IVT)** | Stores mappings between IRQ numbers and ISR functions. |
| **Global Interrupt Controller (GIC/APIC)** | Routes, prioritizes, and dispatches interrupts to CPU cores. |
| **Nested Interrupt Handling** | Allows interrupts to preempt lower-priority ones. |
| **Soft Interrupts (SGIs)** | Allows software-generated and inter-processor interrupts. |
| **Low-Power Wake-up Interrupts** | Triggers CPU wake-up for critical events. |
| **Exception Handlers** | Manages CPU-level exceptions and faults. |

**Interrupt Handling Code Example**

**Registering an Interrupt Handler**

#include "hal\_core\_interrupts.h"

// Define an ISR for a hardware timer interrupt

void timer\_interrupt\_handler(void) {

// Clear interrupt flag

hal\_clear\_interrupt(TIMER\_IRQ);

// Process timer event

process\_timer\_event();

}

// Register ISR during initialization

void init\_interrupts(void) {

hal\_register\_interrupt(TIMER\_IRQ, timer\_interrupt\_handler, INTERRUPT\_PRIORITY\_HIGH);

}

## ****Interaction with Other HAL Components****

| **HAL Component** | **Role in Interrupt Handling** |
| --- | --- |
| hal\_core\_config | Defines system-wide interrupt settings and priorities. |
| hal\_cpu | Handles CPU-specific interrupt processing. |
| hal\_mem\_manager | Ensures memory-mapped interrupt vectors are properly allocated. |
| hal\_io | Interfaces with hardware devices that generate interrupts. |
| hal\_trustzone | Ensures secure execution of interrupt handlers in trusted environments. |

## ****Future Enhancements****

* **AI-Powered Interrupt Prioritization**
  + Use **machine learning** to dynamically adjust interrupt priorities based on system load.
* **Asynchronous Interrupt Handling**
  + Offload ISR execution to **dedicated cores** or **real-time accelerators**.
* **Virtualized Interrupt Management**
  + Improve support for **virtualized environments**, enabling seamless interrupt handling across VMs.

## ****Summary****

The **hal\_core\_interrupts** module is a critical component for **handling hardware and software interrupts**, enabling **efficient event-driven processing**, **low-latency execution**, and **real-time responsiveness**. It ensures smooth interaction between hardware devices and software processes while optimizing power, security, and system stability.