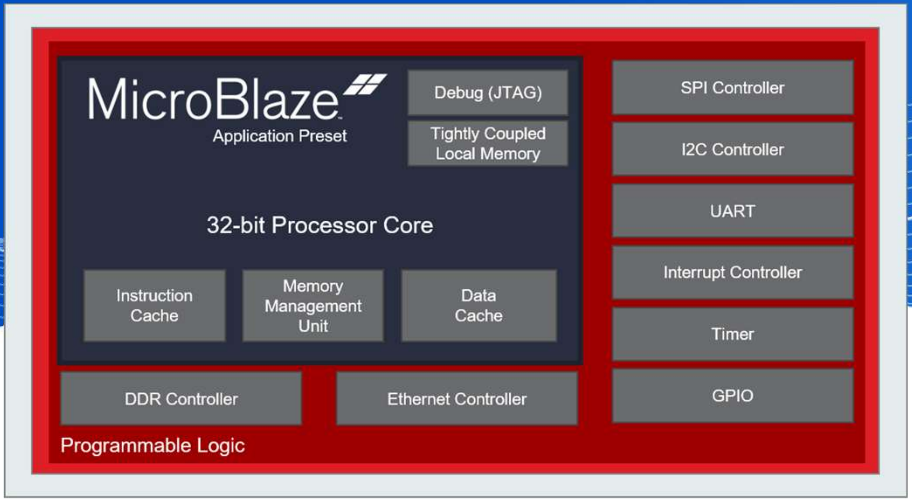
Lecture 7
Microprocessor Interrupts

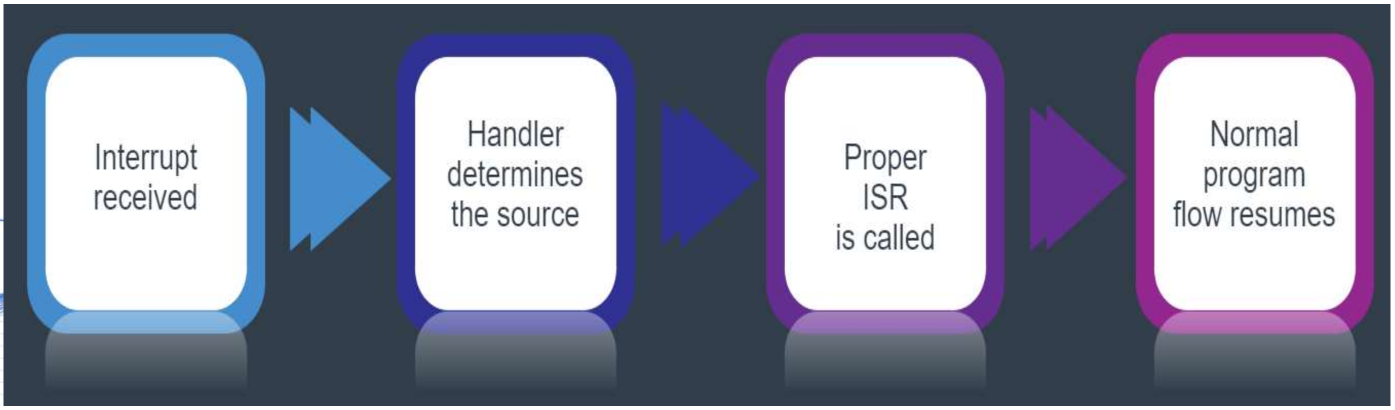


Electrical Engineering Department
Pontificia Universidad Católica de Chile
peclab.ing.uc.cl

Interrupts

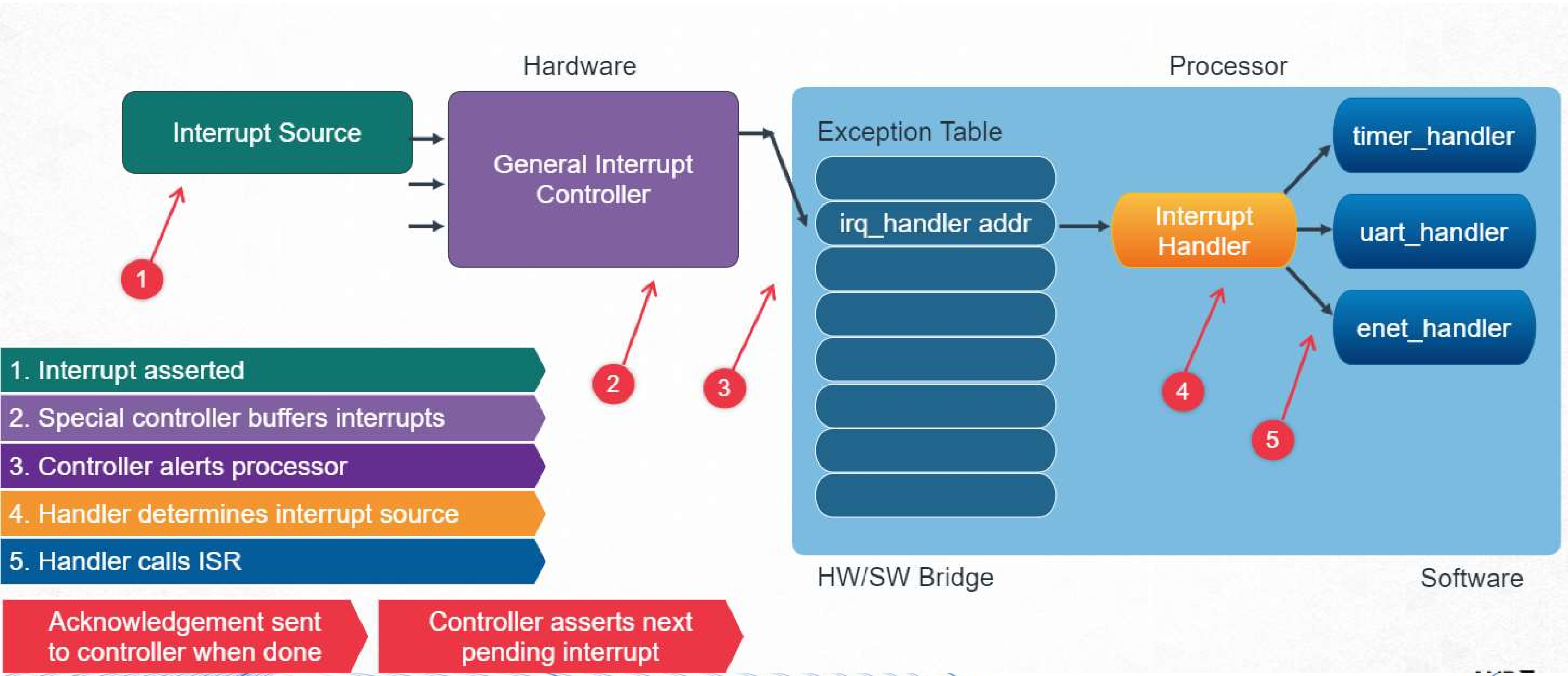
Basic Concepts

1. Interrupts **break the normal program** execution flow.
2. The hardware wait until the actual machine instruction to finish, then **jump to a special piece of code, named interrupt service routine or interrupt handler,** that deals with the interrupt.
3. The interrupt handler **determines the source** of the interrupt.
4. Based on the source, the **interrupt handler calls a more specific routine** that perform the necessary task.
5. Once this small specific routine is completed (handled the interrupt), the processor **come back from where it was before the interrupt.**



Interrupts

Basic Concepts



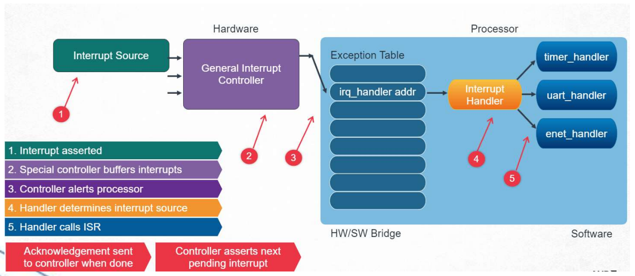
1. Interrupt asserted
2. Special controller buffers interrupts
3. Controller alerts processor
4. Handler determines interrupt source
5. Handler calls ISR

Acknowledgement sent to controller when done

Controller asserts next pending interrupt

Interrupts
Basic Concepts

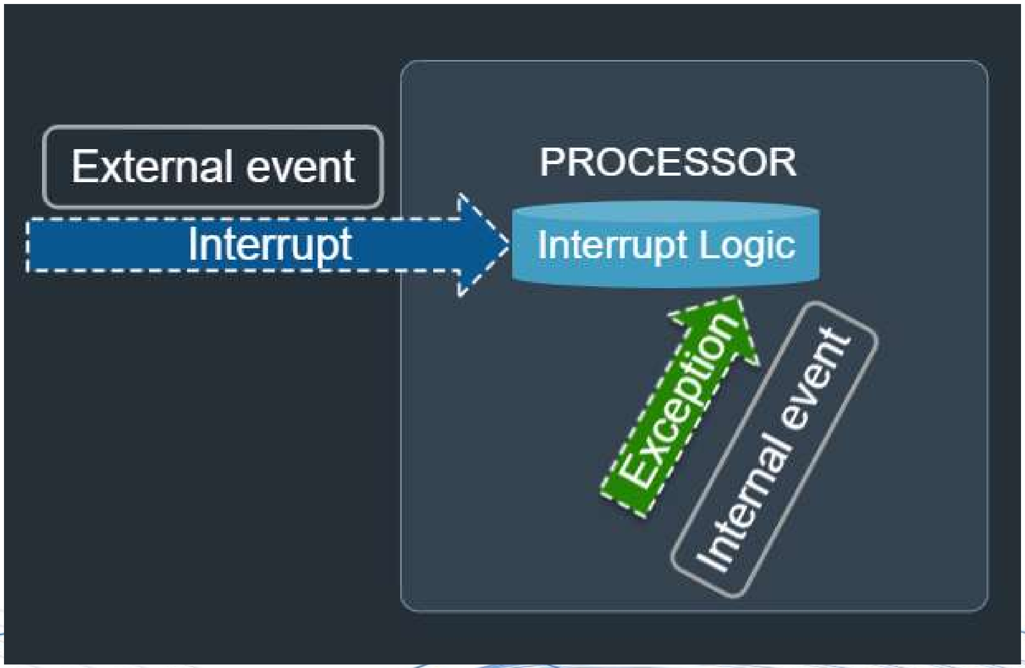
1. A device asserts an interrupt
   1. In simple uP this goes directly to Interrupt handler. (few interrupt sources)
   2. In more complex Systems goes to interrupt controller. (many interrupt sources)
2. A **General Interrupt Controller** (GIC) buffer all interrupts, then select one interrupt and send it to the processor to the **first-level handler.**
3. **First-Level Handler** determines the source of the interrupt (through a simple query to the interrupt controller module)
4. Based on the source, the interrupt handler calls an interrupt service routine. (ISR)
5. When the ISR a signal is sent back to the GIC to indicate that **interrupt has been attended.**
6. The GIC can send the next pending interrupt.


Hardware
Processor
Interrupt Source
General Interrupt Controller
Exception Table
irq_handler addr
Interrupt Handler
timer_handler
uart_handler
enet_handler
1
2
3
4
5
1. Interrupt asserted
2. Special controller buffers interrupts
3. Controller alerts processor
4. Handler determines interrupt source
5. Handler calls ISR
Acknowledgement sent to controller when done
Controller asserts next pending interrupt
HW/SW Bridge
Software

Interrupts

Interrupts and Exceptions

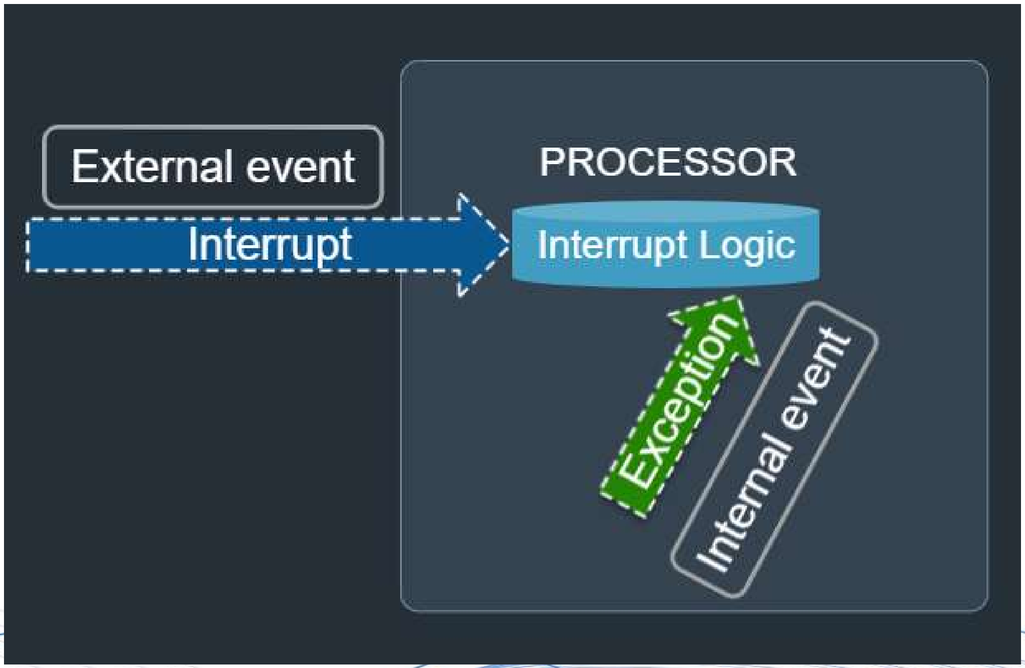
Interrupt



1. Interrupt is used only when the event is originated outside the **processor inmidiate domain.**
2. Processor immediate domain in a Zynq would be the APU. (could come from peripherals inside the PS or from PL)
3. The **interrupts are considered asynchronous**. We can no predict them, and can be asynchronous with the PS clock.
4. Processor architecture determines the number of interrupt pins:
   1. Some have two or more (several)
   2. Some have one pin. (special form for handling several interrupt sources)

Interrupts and Exceptions

Exception



1. Exception cause same behaviour as interupts, but exceptions:
    1. Are generated within the processor.
    2. Are considered synchronous with uP clock. Can be predicted and repeated.
    3. Runs an Exceptions Handler. Almost same thing as Interrupt handler but solve different issues.
    4. Usually are improper conditions on the code as divide by zero.

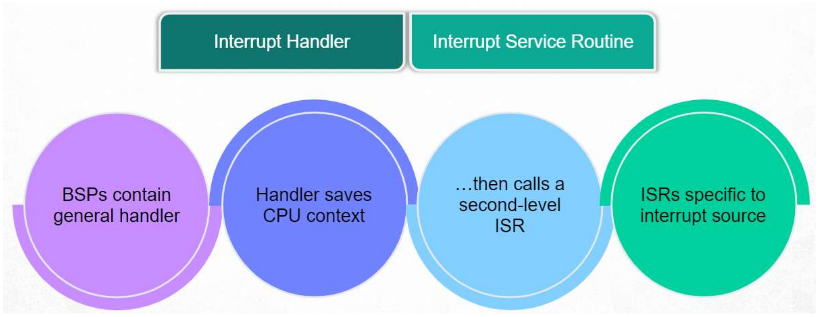
Interrupts

Interrupt Handler or Interrupt Service Routine?

Often these two terms are used indistinctly. It does not matter much the names, but we need to understand that there are two levels of interrupt (and exception) servicing.

1.- A first level of handler saves the processor context when an interrupt is asserted. Saves the stack pointer, set of registers,etc. Then based on the interrupt source, looks in a **preset Interrupt Vector Table (IVT)**, for a **pointer to a second-level** interrupt specialized service routine.

2.- The second level Interrupt Service Routine, is one of many specialized routines. It only runs a specific piece of code.



Interrupt Handler
Interrupt Service Routine
BSPs contain general handler
Handler saves CPU context
…then calls a second-level ISR
ISRs specific to interrupt source

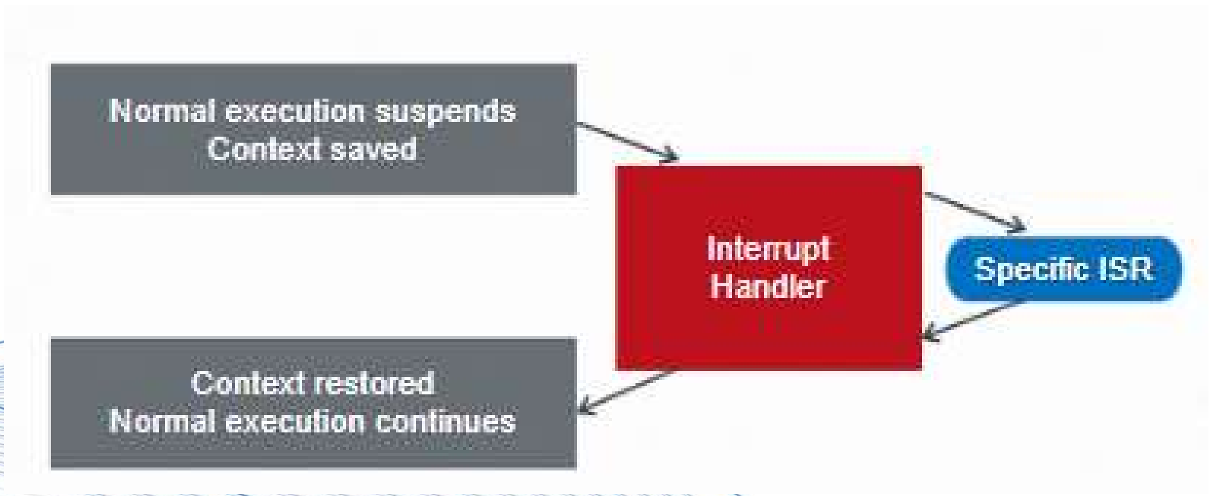
BSP: software containing hardware-specific boot firmware and device drivers and other routines

Interrupts

Interrupt Handler or Interrupt Service Routine?

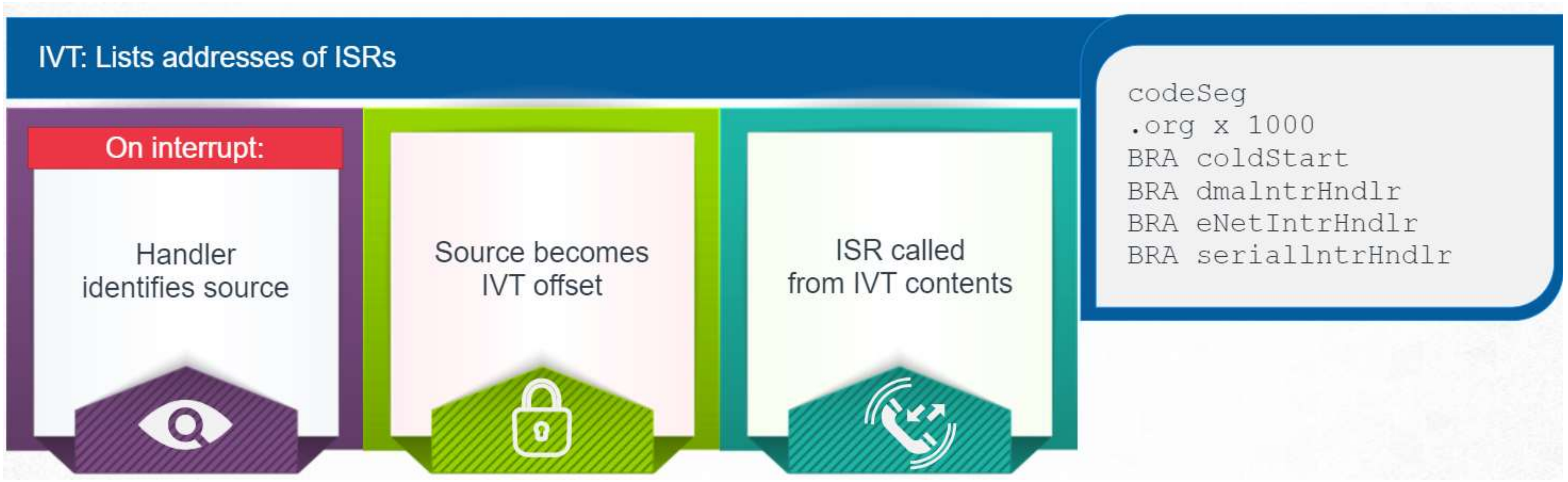
Often these two terms are used indistinctly. It does not matter much the names, but we need to understand that there are two levels of interrupt (and exception) servicing.

3.- The IVT has ISR pointers to interrupt handler of common peripherals (e.g. UART). The user must fill the table with a specific address in case of written its own specialized routine. This is known as **“Registering the Interrupt Handler”.**



Interrupts

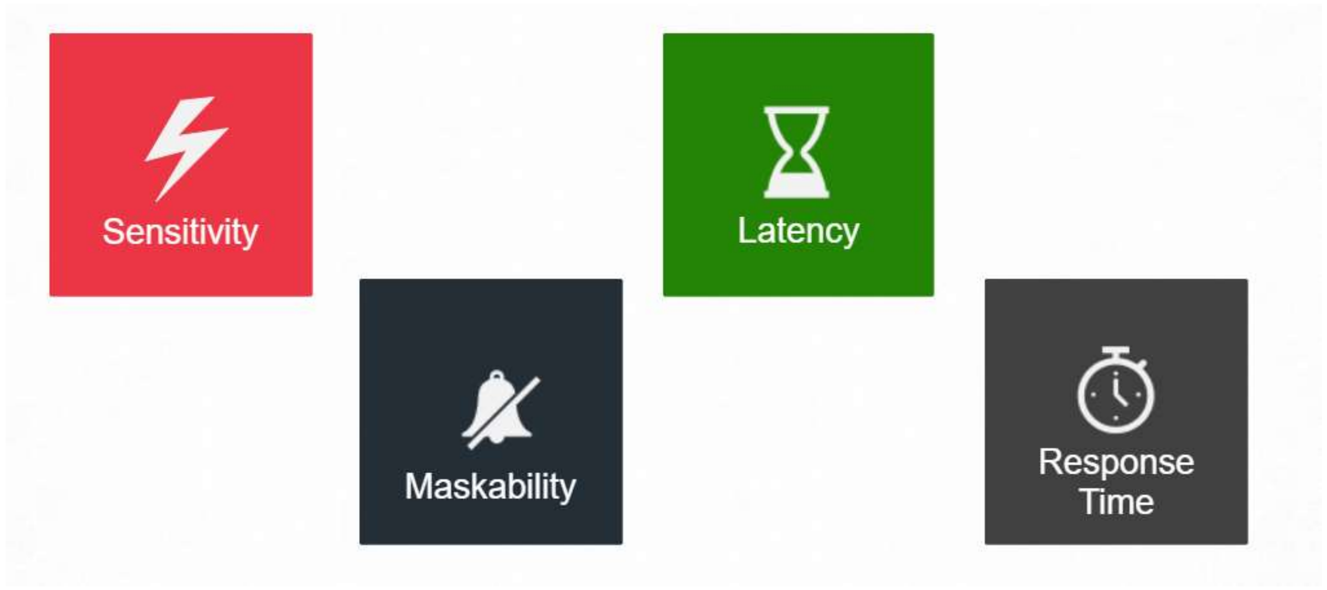
Interrupt Vector Table


IVT: Lists addresses of ISRs
On interrupt:
Handler identifies source
Source becomes IVT offset
ISR called from IVT contents

```
codeSeg
.org x 1000
BRA coldStart
BRA dmaIntrHndlr
BRA eNetIntrHndlr
BRA serialIntrHndlr
```

The IVT contains a list of addresses ointing to specific interrupt handlers.
After identifying the source of interrupt. It is converted to one position of the IVT and point to a specific ISR.

Interrupts
Interrupt Vocabulary



Interrupts

Interrupt Vocabulary

Sensitivity: Define the state of the signal that causes the processor to begin the interrupt handling process, there are four types:

Low-level: Triggered after the signal remains “n” in low state.

High-level: Triggered after the signal remains “n” in high state.

Rising Edge: When rising Edge is detected.

Falling Edge: When falling Edge is detected.

Interrupts

Interrupt Vocabulary

Latency

There are two main definitions:

Definition 1: Time it takes the processor to recognize that an interrupt has been asserted.

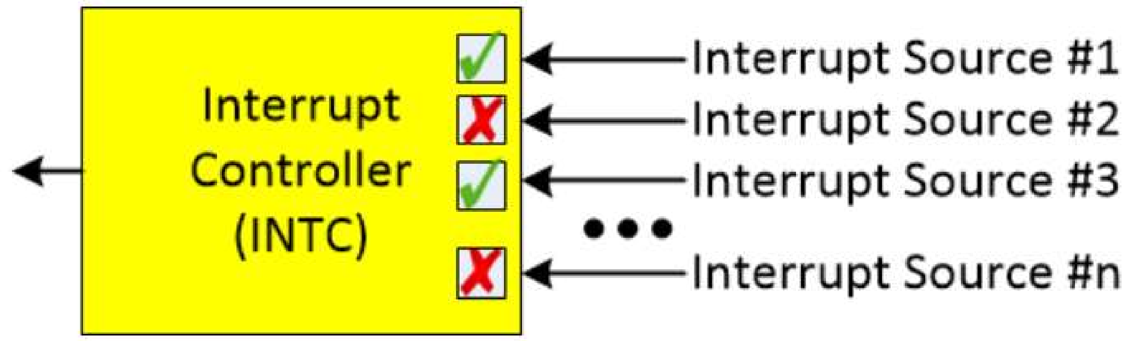
Definition 2: Time from the interrupt occurs until that the specific handler begins.

Interrupts

Interrupt Vocabulary

Maskability

- Some times interrupts needs to be ignored during certain time.
- Maskability is the process of disabling an interrupt during a time frame.
- The maskability is applied to the Interrupt Controller (INTC or GIC)



Interrupts

Interrupt Vocabulary

Response Time

- Time it takes the processor to change from its current task and being running the ISR.
- Most processor today are RISC, therefore, processor take 1 or 2 cycles (one machine instruction) until it starts attending the ISR.

Typical Interrupt Controller
Basic Concepts

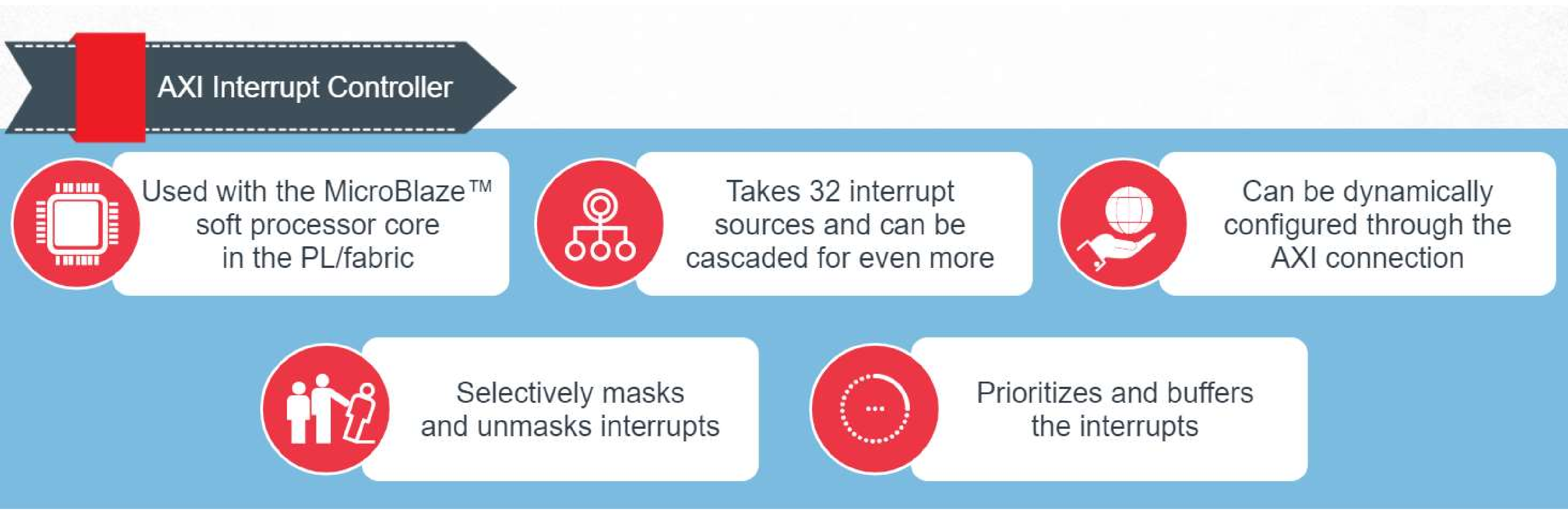
- Most modern processors have only one or two interrupt input.
  - How do we handle several interrupt source?

We do it using an Interrupt controller.

An interrupt controller is a piece of hardware that acts as repository for interrupts.

- A simple Interrupt Controller is the AXI Interrupt Controller used typically in Microblaze

Typical Interrupt Controller
Basic Concepts



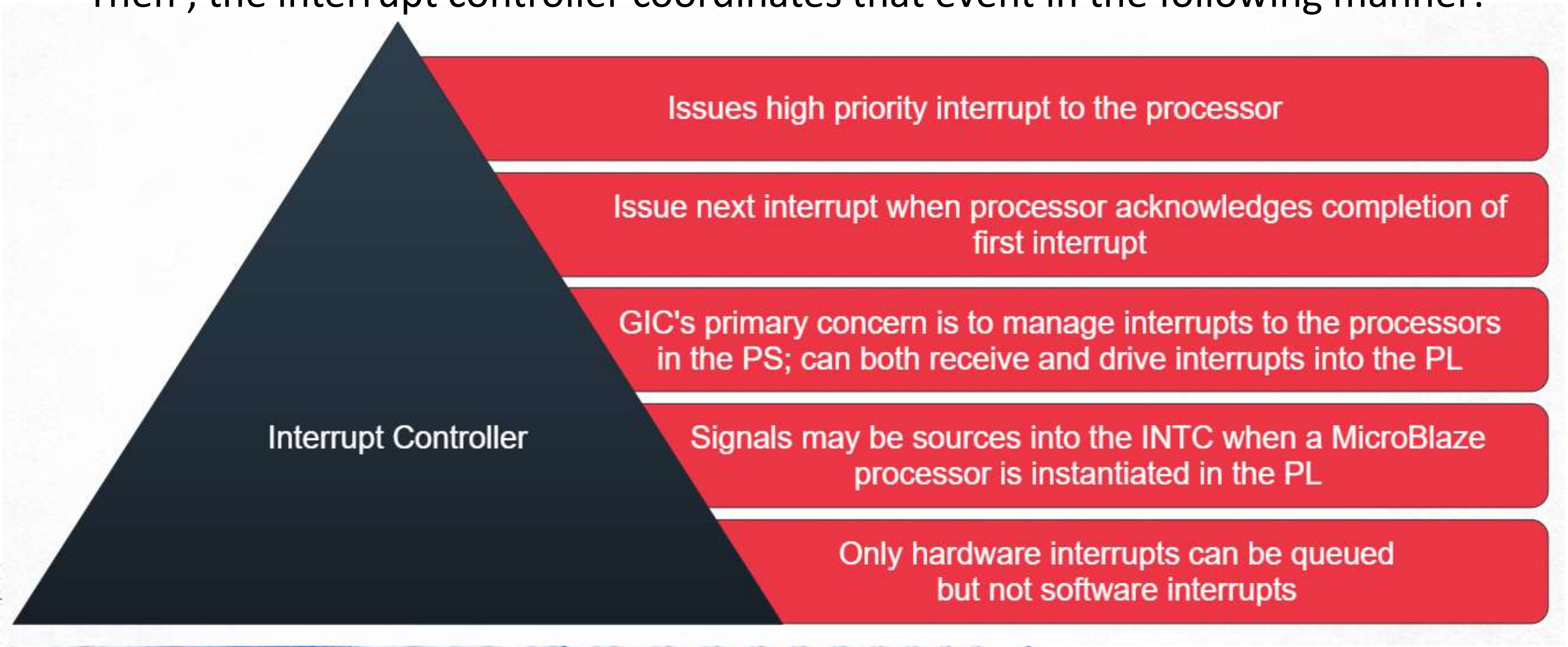
AXI Interrupt Controller

- Used with the MicroBlaze™ soft processor core in the PL/fabric
- Takes 32 interrupt sources and can be cascaded for even more
- Can be dynamically configured through the AXI connection
- Selectively masks and unmasks interrupts
- Prioritizes and buffers the interrupts

Typical Interrupt Controller

What happend if two interrupts occurs simultaneously? Or If one interrupt is being handled and another interrupt arrives?

Then , the interrupt controller coordinates that event in the following manner:


Interrupt Controller
Issues high priority interrupt to the processor
Issue next interrupt when processor acknowledges completion of first interrupt
GIC's primary concern is to manage interrupts to the processors in the PS; can both receive and drive interrupts into the PL
Signals may be sources into the INTC when a MicroBlaze processor is instantiated in the PL
Only hardware interrupts can be queued but not software interrupts

1.- The Zynq Family uses a GIC within the PS as a dedicated silicon hardware.
2.- Exceptions are nor bufferable

Typical Interrupt Controller

Basic Concepts

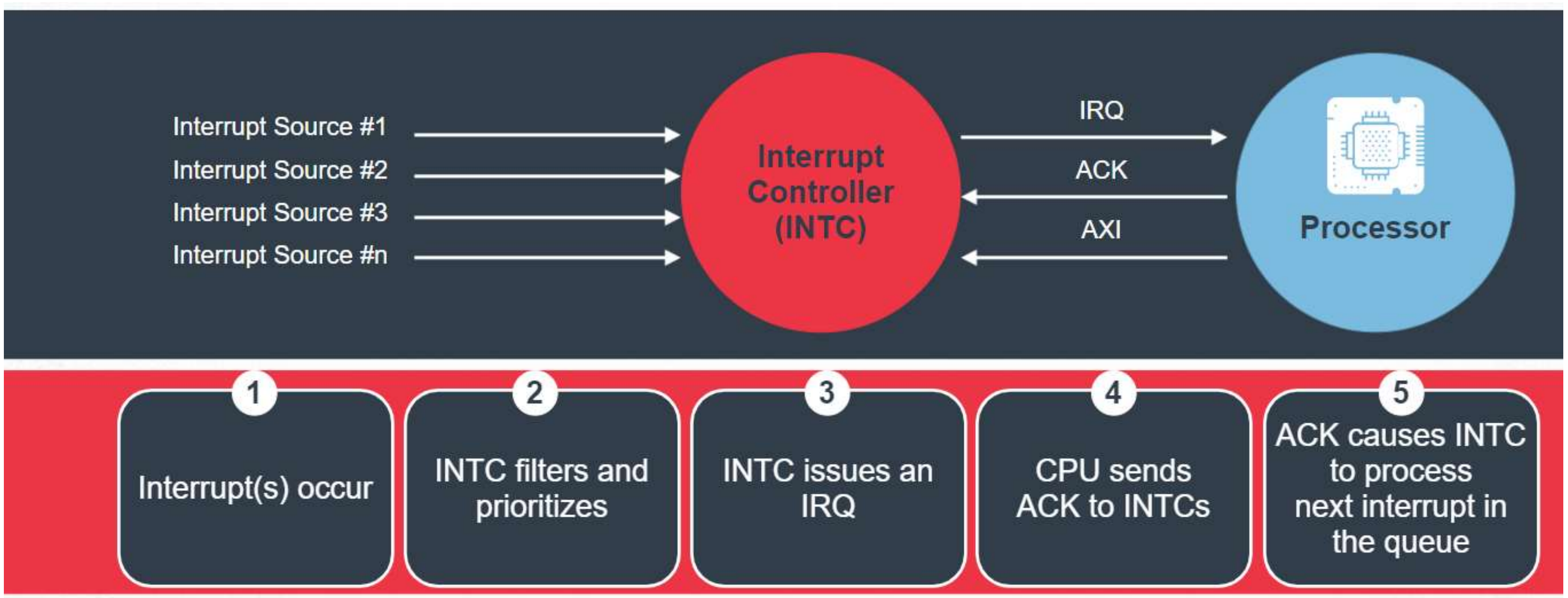
- Interrupt controllers provide a bus interface over which the processor can :
    - Configure interrupts maskability
    - Configure interrupts priority
    - Identify the source of the interrupt
- This bus is accessed by the processor within base-level handler code so it can determine which ISR should be calle.

- Some examples of interrupt controllers are:
    - AXI-INTC IP Core.
    - ARM Generic Interrupt Controller . (used in Zynq family)
    - Both of them use AXI slave interface as primary configuration and status checking interface.

- The processor occurs as follow: (next slide)

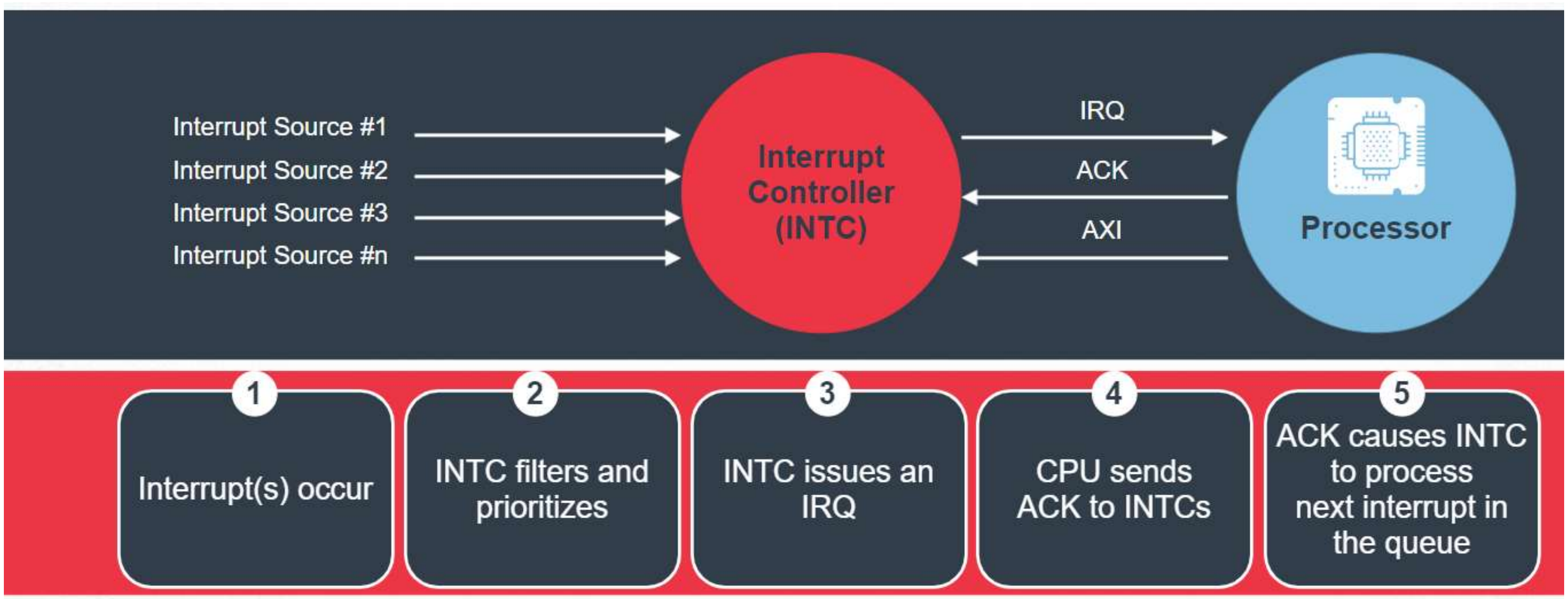
Typical Interrupt Controller

Process



IRQ: Interrupt **R**e**Q**uest

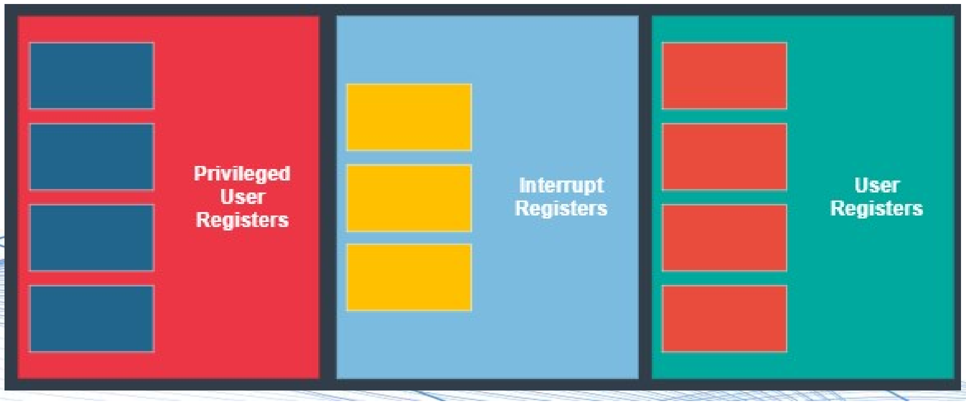
Typical Interrupt Controller
Process



IRQ: Interrupt **ReQ**uest

Interrupt-Related Registers
Registers

- Once an interrrupt has been serviced, the processor must recover its previus state.
  - Some procesors push each register that must be preserved onto the stack, including program counter.
    - This method is simple and used in most basic architectures, but it is slow.
  - More complex processors just switch Banks of registers toa set of dedicated registers for managing interrupts.
    - This method is very fast, completed in one clock, but require more hardware.
    - Switching back to previous bank of registers is equally fast.
    - Great level of protection and osolation



Interrupt-Related Registers

Registers

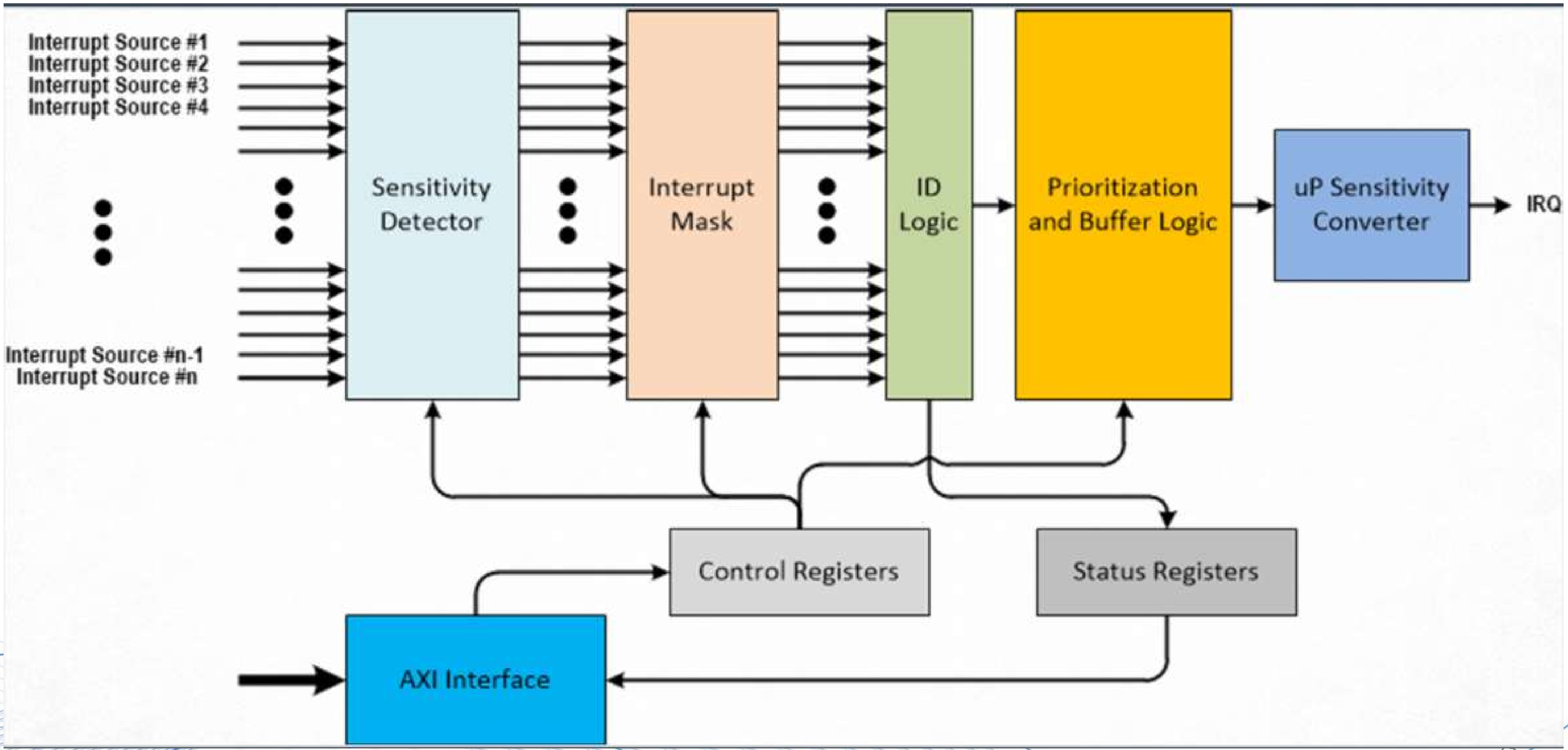
- **Interrupt-Related Registers:**
  - Store the source of interrupt.
  - This saves processor time, by not having to ask this information over the bus interface.

- **Banked Registers:**
  - This set of registers used as general purpose only during interrupt handling. This does not affect the values of the registers outside the interrupt handling.
  - ISR is free to use these banked registers. Avoid to save previous value of the register, use it during the interrupt and them put the previous value back into the register.

- **Vector Base Address Register**
  - Store the location of the vector table used for the exception and interrupt handling.
  - Does not have direct impact on interrupt execution time but provide more flexibility to the system architecture.

The interrupt registers vary from processor to processor. These are just some examples of the more important.

Interrupt Controller
Basic Building Blocks



Interrupt Controller

Basic Building Blocks

AXI Interface

- First, during the initialization phse, processor write the control and configuration registers of the Interrupt Controlle.
- The configuration set priorities, sensitivity, and other parameters.
- Some configurations are set during hardware creating and then can be overwritten by software.

.

Interrupt Controller

Basic Building Blocks

Sensitivity Detector

- This module coverts the configured sensitivity of the interrupt (high-level, low-level, rising edge, falling edge) to an internal consistent behavior. (input by input case).

Interrupt Mask:

- Allow the programmer to enable o disable interrupts.
- Using a mask, interrupts can still be registered, without triggering IRQ to processor.
- Usually, software temporarily disable interrupts when is performing a very important task.

Interrupt Controller

Basic Building Blocks

ID Logic

- This part of the interrupt controller assigns a unique ID to its input.
- Typically referred as IRQ numbers.
- This enables the processor to query the interrupt controller by the source of the interrupt.

Priorization and Buffer Logic:

- Responsible for deciding an interrupt should be immediately trigger an IRQ or be queued until the processor is ready to accept another interrupt.
- If two interrupts occurs simultaneously, this block decide which trigger IRQ.
- When processor indicates that the current interrupt has finished, this block issue the next interrupt.

Interrupt Controller

Basic Building Blocks

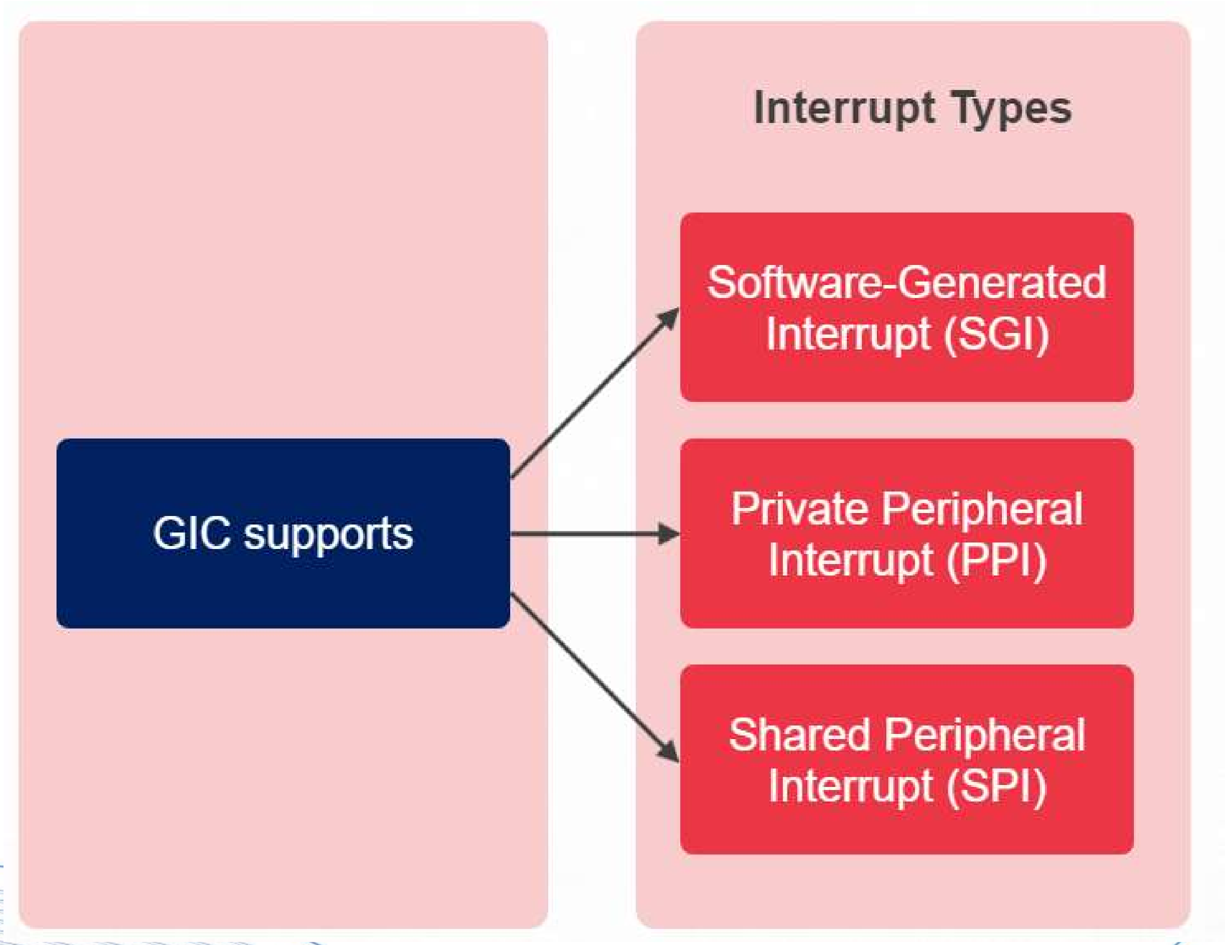
uP Sensitivity Converter

- Converts the internal representation of the interrupt to the proper sensitivity that the processor requires to send the IRQ.

- After the processor finishes the interrupt, the acknowledge ca come through a dedicated line o through AXI connection.

Interrupts
Interrupt Types

- More specific about ARM GIC. It can handle three categories if interrupts.


Interrupt Types
GIC supports
Software-Generated Interrupt (SGI)
Private Peripheral Interrupt (PPI)
Shared Peripheral Interrupt (SPI)

GIC Interrupts
Software Generated Interrupt (SGI)

1. Triggered under software control. By writting a specific register of the GIC.
2. Used for inter-processor communication (IPC)
3. It is a mechanism to interrupt one core with another
4. The GIC supports 16GICs, IRQ1-IRQ16 are udentified by the GIC architecture.
5. SGICs acts as Edge sensitivity interrupts.

GIC Interrupts

Private Peripheral Interrupt (PPI)

1. Interrupts generated by hardware components that are private to a specific processor core.
2. Typical examples are private timer or private watchdog.

GIC Interrupts

Share Peripheral Interrupt (SPI)

1. Interrupts generated by shared hardware components.
2. These interrupts represents the majority of interrupts.
3. The number of SPI depends on each hardware , but always are greater than 32.
4. SPISs can be distributed to a single core or multiple cores.
5. Hardware architecture ensures that only one core will service the interrupt.

peclab

POWER & ENERGY CONVERSION LABORATORY

Electrical Engineering Department

Pontificia Universidad Católica de Chile

peclab.ing.uc.cl