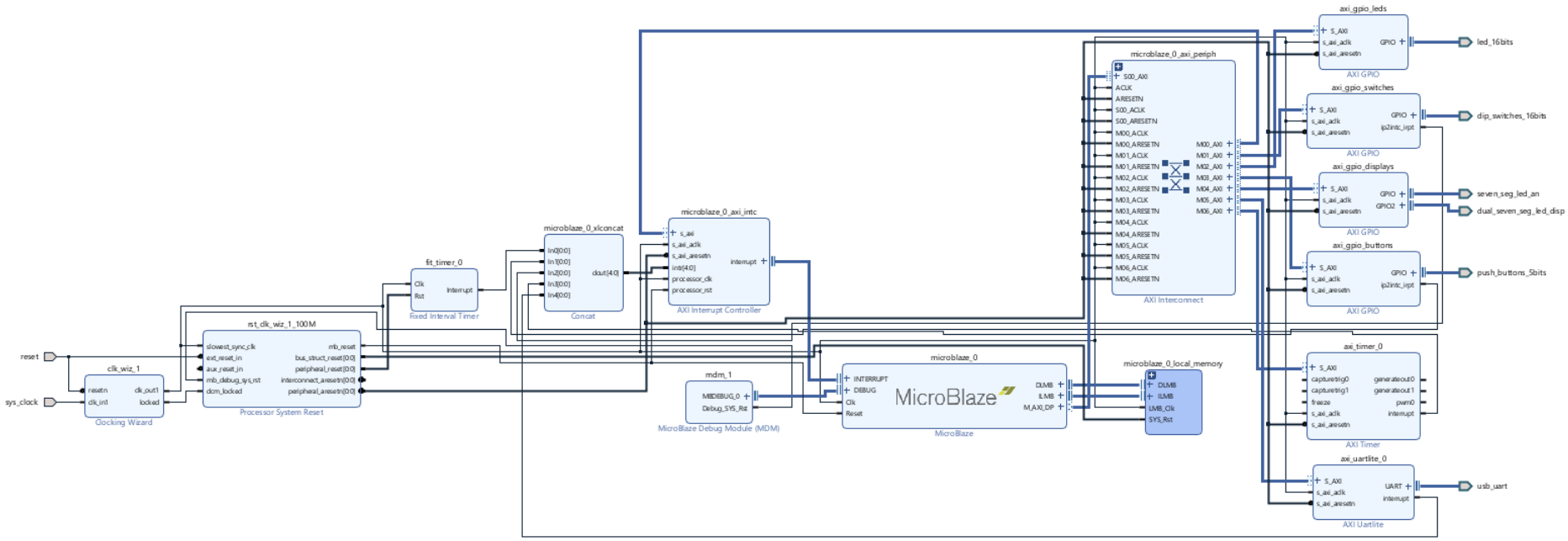
Interrupts

LECTURE 8

PEDRO SANTOS (SLIDES BY IOULIIA SKLIAROVA)

Block Design (BD)

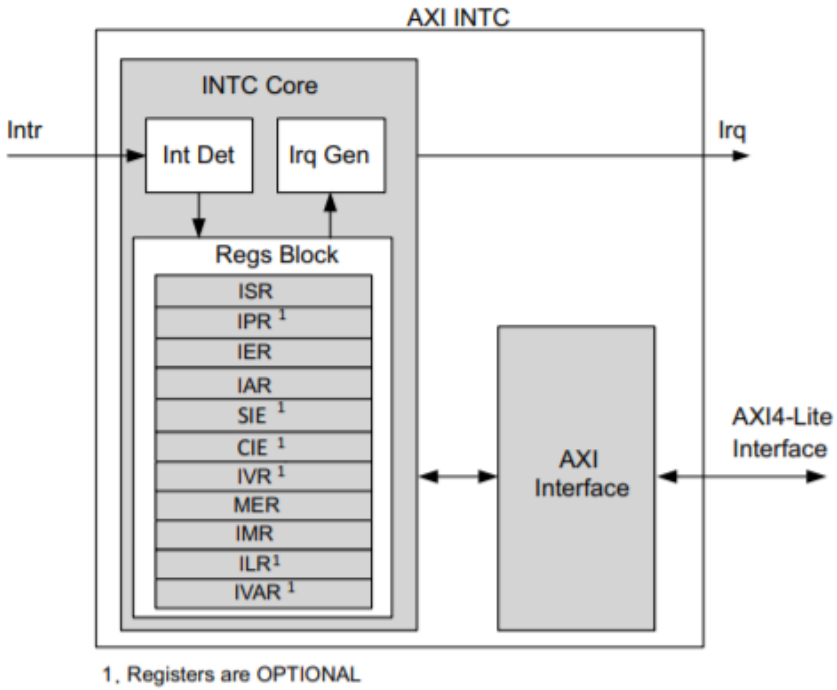


AXI Interrupt Controller

- AXI Interrupt Controller (INTC) core receives multiple interrupt inputs from peripheral devices and merges them into an interrupt output to the system processor.
- The registers used for storing interrupt vector addresses, checking, enabling and acknowledging interrupts are accessed through the AXI4-Lite interface.
- Supports up to 32 interrupts.
- Priority between interrupt requests is determined by vector position. The least significant bit (LSB, in this case bit 0) has the highest priority.
- Interrupt Enable Register for selectively enabling individual interrupt inputs.
- Master Enable Register for enabling interrupts request output.
- Each input is configurable for edge or level sensitivity.

AXI Interrupt Controller

- Registers Block:
  - This block contains control and status registers.
  - They are accessed through the AXI4-lite slave interface.
- Interrupt Detection:
  - This block detects the interrupts input.
  - It can be configured for either level or edge detection for each interrupt input.
- Interrupt Generation:
  - Generates the final output interrupt from the interrupt controller core.
  - Checks for enable conditions in control registers (MER and IER) for interrupt generation.
  - Resets the interrupt after acknowledge.
  - …



INTC Registers

- Interrupt Status Register (ISR)
  - indicates the presence or absence of an active interrupt signal.
- Interrupt Enable Register (IER)
  - Writing a 1 to a bit in this register enables, or unmasks, the corresponding ISR bit, allowing it to affect the irq output.
  - When an interrupt is disabled, the interrupt event occurs but is not passed to the processor.
- Master Enable Register (MER)
  - enables the Irq output signal or mask all interrupt inputs.
- Interrupt Acknowledge Register (IAR)
  - write-only register that clears the interrupt request associated with selected interrupts.
  - In fast interrupt mode, bits in the IAR are automatically cleared by using the information from the processor_ack port

Using Interrupts in Software

1. Connect a **callback handler** that will be called by the ISR for **each of the possible interrupt sources**.

```
360     XIntc_RegisterHandler(intcBaseAddress, HARDWARE_TIMER_INT_ID,
361                           (XInterruptHandler)TimerIntCallbackHandler, (void *)0);
```

```
372     XIntc_RegisterHandler(intcBaseAddress, BUTTONS_INT_ID,
373                           (XInterruptHandler)ButtonsIntCallbackHandler, (void *)0);
```

2. **Enable interrupt requests** for all the involved interrupt sources.

```
375     // Enable interrupt requests at the buttons GPIO
376     XGpio_WriteReg(XPAR_AXI_GPIO_BUTTONS_BASEADDR, XGPIO_IER_OFFSET, XGPIO_IR_CH1_MASK);
377     XGpio_WriteReg(XPAR_AXI_GPIO_BUTTONS_BASEADDR, XGPIO_GIE_OFFSET, XGPIO_GIE_GINTR_ENABLE_MASK);
```

3. **Each bit in the IER corresponding to an interrupt must be set to 1**. This allows the AXI INTC core to begin accepting interrupt input signals. INT0 has the highest priority, and it corresponds to the least significant bit (LSB) in the IER.

```
379     // Enable interrupts for all the peripheral devices that cause interrupts,
380     XIntc_EnableIntr(intcBaseAddress, HARDWARE_TIMER_INT_MASK | BUTTONS_INT_MASK);
```

4. The MER must be programmed based on the intended use of the AXI INTC core. There are two bits in the MER: the Hardware Interrupt Enable (HIE) and the Master IRQ Enable (ME). **The ME bit must be set to enable the interrupt request output.**

```
375     // Enable interrupt requests at the buttons GPIO
376     XGpio_WriteReg(XPAR_AXI_GPIO_BUTTONS_BASEADDR, XGPIO_IER_OFFSET, XGPIO_IR_CH1_MASK);
377     XGpio_WriteReg(XPAR_AXI_GPIO_BUTTONS_BASEADDR, XGPIO_GIE_OFFSET, XGPIO_GIE_GINTR_ENABLE_MASK);
```

5. **Connect the handler for the interrupt controller to the interrupt source for the processor.**

```
343 // This function will be called back by the INTC ISR whenever a button is pressed or released
344 void ButtonsIntCallbackHandler(void* callbackParam) //parameter is not used
```

Final Remarks

At the end of this lecture you should be able to:

- write C programs that use hardware interrupts