NIOS Interrupts

Last updated 7/21/23

NIOS Interrupts

- HAL Framework – Interrupts
  - 2 Options for dealing with interrupts
    - Internal Interrupt Controller – IIC
    - External Interrupt Controller - EIC
  - The IIC has two versions
    - Legacy API
    - Enhanced API
  - Most of the peripherals we are likely to use support the enhanced API – but some of the older peripherals may require the legacy API

NIOS Interrupts

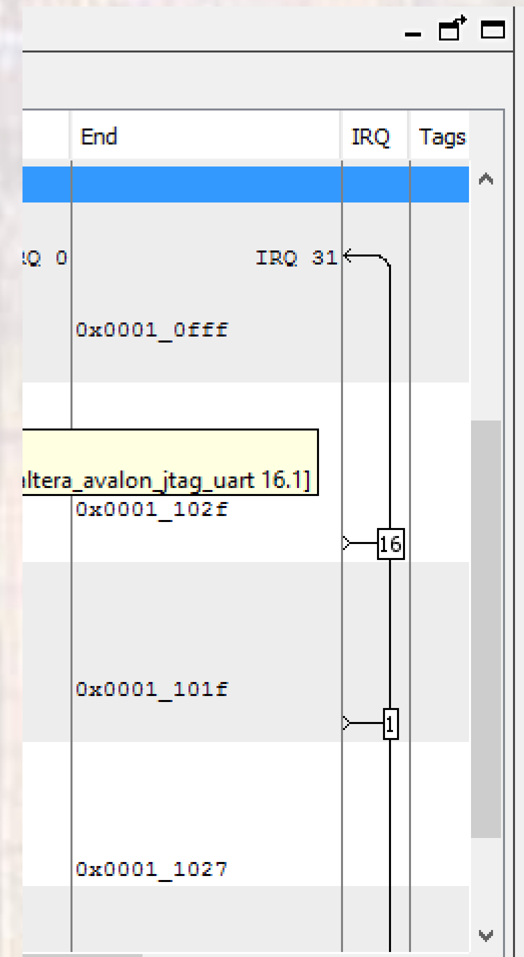
- HAL Framework – Interrupts

  - On an interrupt or exception, the processor
    - Saves the current status
    - Disables HW interrupts
    - Saves the next execution address (Program Counter)
    - Transfers control to the exception handler

  - What about all the registers?
    - NIOS supports shadow registers
    - Removes the need to save the registers to the stack

NIOS Interrupts

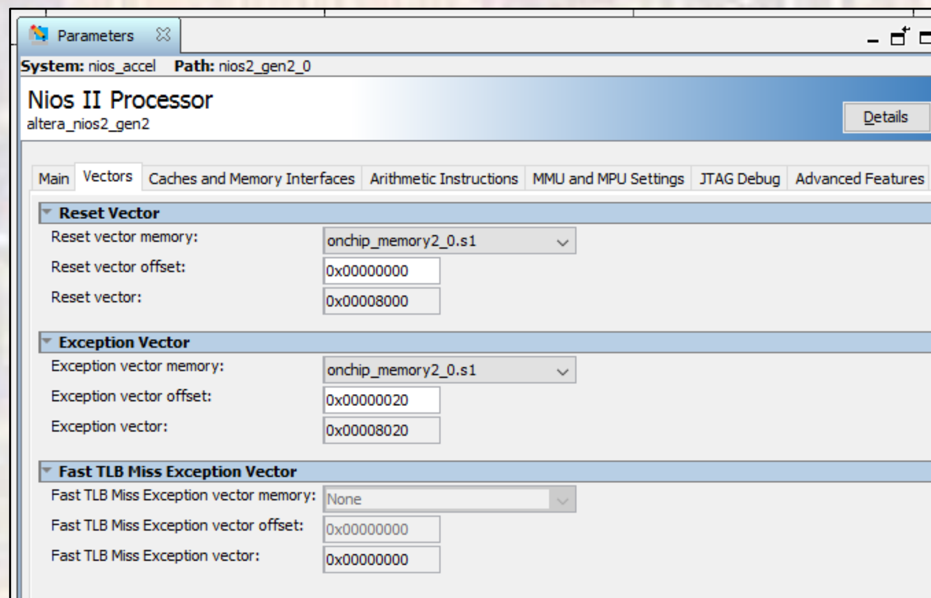
- HAL Framework – Interrupts
  - Support for 32 interrupt signals
  - Interrupt # indicates inverse priority**
  - We set these in QSYS



** Managed by the HAL SW

NIOS Interrupts

- HAL Framework – Interrupts
  - The HAL Internal Interrupt framework uses a single exception controller
    - We set the address for the “exception handler” when we instantiate the NIOS II processor



 © tj

NIOS Interrupts

- HAL Framework – Interrupts
  - The single Exception handler
    - Handles exceptions and interrupts
  - Exception handler keeps a table of ISR priorities
  - To be included in the table – we must “register” the IRQ of each module

NIOS Interrupts

- HAL Framework – Interrupts

  - Enhanced Exception Handler Functions
    - ALT_ENHANCED_INTERRUPT_API_PRESENT in system.h

```
alt_ic_isr_register()
alt_ic_irq_disable()
alt_ic_irq_enable()
alt_irq_disable_all()
alt_irq_enable_all()
alt_ic_irq_enabled()
```


© tj

NIOS Interrupts

- HAL Framework – Interrupts

  - Legacy Exception Handler Functions
    - ALT_LEGACY_INTERRUPT_API_PRESENT in system.h

alt_irq_register()
alt_irq_disable()
alt_irq_enable()
alt_irq_disable_all()
alt_irq_enable_all()

alt_irq_interruptible()
alt_irq_non_interruptible()
alt_ic_irq_enabled()

© tj

NIOS Interrupts

- HAL Framework – Interrupts

  - ISR register function - prototype

```
int alt_ic_isr_register(
    alt_u32   ic_id,             // interrupt controller ID
    alt_u32   irq,               // interrupt ID
    alt_isr_func   isr,          // isr name
    void  *  isr_context,        // pointer to any passed context
    void  *  flags               // reserved – 0
);
```


© tj

NIOS Interrupts

- HAL Framework – Interrupts
  - Register prototype

interrupt controller ID – not used with IIC - 0

interrupt ID – from system.h

isr name – your choice

pointer to any passed context – e.g PIO input value

this can be any type → needs a "void" pointer

```
#define SW_PIO_HAS_IN 1
#define SW_PIO_HAS_OUT 0
#define SW_PIO_HAS_TRI 0
#define SW_PIO_IRQ 10
#define SW_PIO_IRQ_INTERRUPT_CONTROLLER_ID 0
#define SW_PIO_IRQ_TYPE "EDGE"
#define SW_PIO_NAME "/dev/sw_pio"
#define SW_PIO_RESET_VALUE 0
#define SW_PIO_SPAN 16
#define SW_PIO_TYPE "altera_avalon_pio"
```


© tj

NIOS Interrupts

- HAL Framework – Interrupts

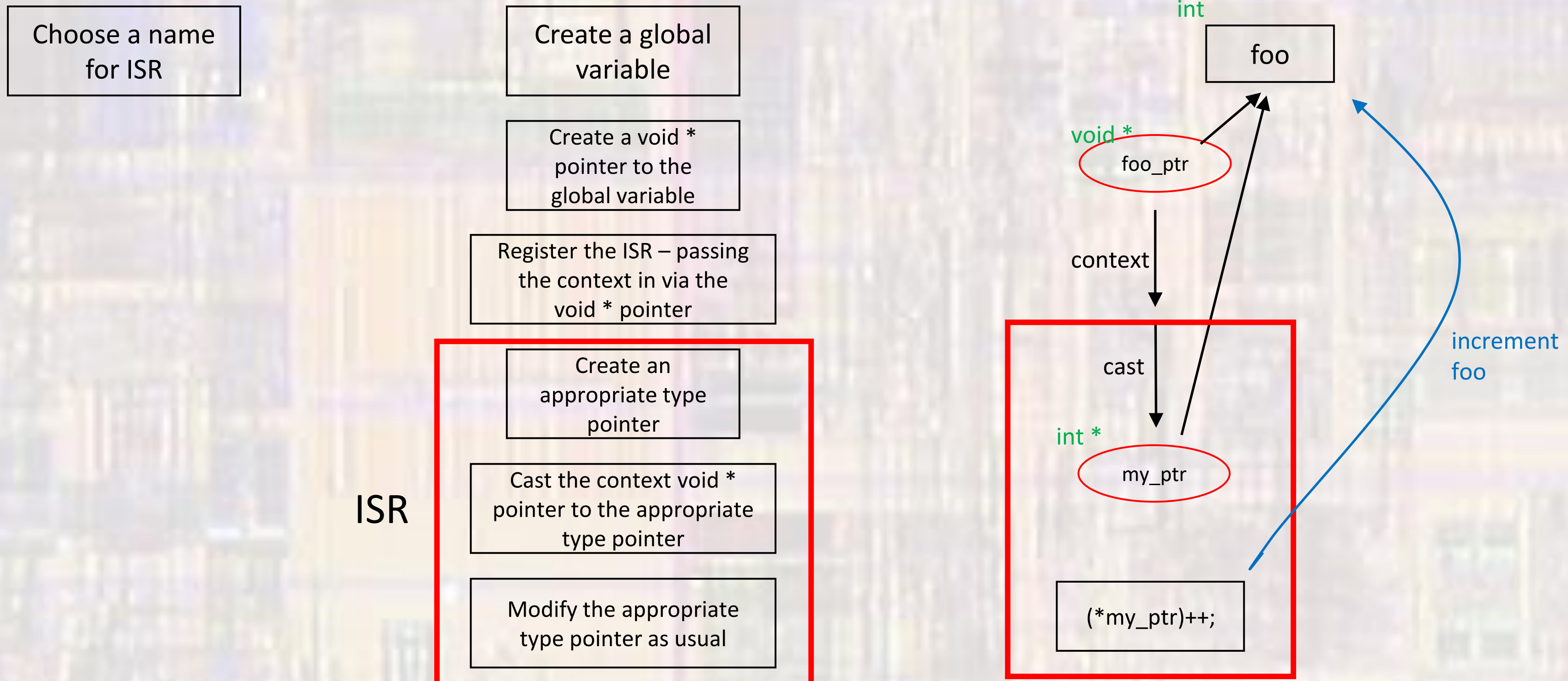
  - ISR prototype

    void isr(void * context)

void pointers do not have a type
void pointers can point to any type
void pointers cannot be dereferenced
void pointers can be cast to any type
pointer arithmetic is not allowed with void pointers

NIOS Interrupts

- HAL Framework – process



© tj

NIOS Interrupts

- HAL Framework – workaround
  - It appears that global variables are available in the ISR
  - You still must create the context and register the ISR