AMD

MicroBlaze Fast Interrupt

Göran Bilski

Interrupt Latency

- In 2011, we realized that MicroBlaze interrupt latency could be improved
- We introduced “Fast Interrupt” (aka vectored interrupts) that is backward compatible and can be mixed with normal interrupts

MicroBlaze steps with normal interrupts (1)

- Interrupt Controller (INTC)
  - Interrupt is sampled and sets the pending interrupt register if enabled
  - Drives the INT signal to MicroBlaze if there is at least one pending interrupt
- MicroBlaze
  - Samples the signal and attach it to an instruction in OF stage (if allowed)
  - If the instruction reaches the WB (retire) stage, jump to 0x10 (C_BASE_ADDR + 0x10)
  - At this address, another jump is done to __interrupt_handler
  - At __interrupt_handler
    - 14 registers are saved (r3-r12, r17,r18, msr) on the stack
  - Jump to MB_InterruptVectorTable.Handler(MB_InterruptVectorTable.CallBackRef)
    - This is the interrupt handler for Xilinx Interrupt Controller
      XIntc_DeviceInterruptHandler

MicroBlaze steps with normal interrupts (2)

- XIntc_DeviceInterruptHandler
  - 6 registers are saved (r16,r19,r22-r25) on the stack
  - Use XIntc_ConfigTable to find out which interrupt controller to use (can be multiple)
  - Read Interrupt status and enables registers from the Interrupt Controller
  - Use this to get all interrupts that is both active and enabled
  - Loop through all existing interrupts (not only the active)
    - Acknowledge the interrupt at the INTC if edge triggered interrupt
    - Call the registered interrupt handler for that interrupt

MicroBlaze steps with normal interrupts (3)

- Registered Interrupt Handler
  - Saves locally used registers (if any) to the stack
  - Handle the cause of the interrupt
  - Restore register from the stack
  - Return to the XIntc_DeviceInterruptHandler (RTSD)

- XIntc_DeviceInterruptHandler
  - Acknowledge the interrupt at the INTC if level triggered interrupt
  - Restore registers from the stack
  - Return to the __interrupt_handler (RTSD)

- __interrupt_handler
  - Restore registers from the stack
  - Return to the interrupted instruction and enable interrupts (RTID)

MicroBlaze steps with fast interrupts

- Interrupt Controller (INTC)
  - Interrupt is sampled and sets the pending interrupt register if enabled
  - Drives the INT signal to MicroBlaze if there is at least one pending interrupt
  - Also drives the interrupt handler address for the highest priority interrupt
- MicroBlaze
  - Samples the signal and attaches it to an instruction in OF stage (if allowed)
  - If the instruction reaches the WB (retire) stage, jump to the interrupt handler address provided by the INTC
- Registered Interrupt Handler
  - Saves registers (if any) to the stack
  - Handle the cause of the interrupt
  - Restore registers from the stack
  - Return back to the Interrupted instruction and enable interrupts (RTID)

Interrupt Latency Improvements

- MicroBlaze system with caches and one simple timer interrupt
- Normal interrupts had an interrupt latency of 100 to 500 clock cycles (cache hit dependent)
- Fast interrupt ~10 clock cycles
- Backward compatible by setting normal interrupts address to 0x10 in the interrupt controller

MicroBlaze Fast Interrupt Interface

- Interrupt controller provides address for specific interrupt
  - 32b input Interrupt_Address
- MicroBlaze sends back acknowledge information
  - 2b output Interrrupt_Ack
    - 00 - Idle
    - 01 - Jumps to the interrupt handler at Interrupt_Address
    - 10 – Executes RTID instruction (return from interrupt)
    - 11 – Interrupts enabled again (MSR[IE] “0”-> “1”)

MicroBlaze Fast Interrupt (SW)

- Fast Interrupt handler is a normal C function with the attribute “fast_interrupt”
  - void interrupt_handler_name() __attribute__((fast_interrupt));
- Each interrupt is specified in the interrupt controller to be level or edge
  - This information is used by the interrupt controller to automatically clear the interrupt pending register (software don’t need to do this).
  - User code doesn’t need to access interrupt controller at all.
- For interrupts without a fast interrupt handler, the interrupt controller provides a default address for the interrupt.
  - Will also block automatic clear of interrupt pending register

RISC-V Adoption

- MicroBlaze always has a separate interrupt handler address
  - Located at C_BASE_ADDRESS + 0x10
  - RISC-V jumps default to the address in mtvec
- MicroBlaze has a unique return from interrupt instruction (RTID)
  - Used for Ack signalling back to INTC
  - Currently using first MRET instruction instead for RISC-V
- Compiler attribute is called “interrupt” for RISC-V
- In general RISC-V is not well defined for exceptions in exception handlers (or exceptions in interrupt handler)
  - MicroBlaze has separation between interrupts and exceptions
    - Can handle exceptions in the interrupt handler
    - Exceptions in Exceptions handler can cause a Halt (Needed for FT)