Embedded Systems

The Altera NIOS II Processor

Outline

• NIOS II Processor System Basics
• NIOS II Processor Architecture
• Processor Implementation
• Processor Implementation Tradeoffs
• NIOS II Register File
• NIOS II Arithmetic Logic Unit
• Unimplemented Instructions
• Custom Instructions
• Floating Point Instruction Support
• Memory and I/O Organization
NIOS II Processor System Basics

• A general-purpose RISC processor core
  – 32-bit instruction set, data path, and address space
  – 32 general-purpose registers
  – Optional shadow register sets – useful for context switching on multitasking systems
  – 32 interrupt sources
  – External interrupt controller interface for more interrupt sources
  – Single-instruction $32 \times 32$ multiply and divide producing a 32-bit result
  – Dedicated instructions for computing 64-bit and 128-bit products of multiplication

NIOS II Processor System Basics (continued)

• A general-purpose RISC processor core
  – Floating-point instructions for single-precision floating-point operations
  – Single-instruction barrel shifter
  – Access to a variety of on-chip peripherals, and interfaces to off-chip memories and peripherals
  – Optional memory management unit (MMU) to support operating systems that
    – require MMUs
  – Optional memory protection unit (MPU)
  – Instruction set architecture (ISA) compatible across all NIOS II processor systems
NIOS II Processor System Basics (continued)

• A NIOS II processor system is equivalent to a microcontroller or “computer on a chip” that includes a processor and a combination of peripherals and memory on a single chip.
• A NIOS II processor system consists of:
  – a NIOS II processor core
  – a set of on-chip peripherals
  – on-chip memory, and
  – interfaces to off-chip memory, all implemented on a single Altera device
• Like a microcontroller family, all NIOS II processor systems use a consistent instruction set and programming model
NIOS II Processor Architecture

- The NIOS II architecture defines the following functional units:
  - Register file
  - Arithmetic logic unit (ALU)
  - Interface to custom instruction logic
  - Exception controller
  - Internal or external interrupt controller
  - Instruction bus
  - Data bus
  - Memory management unit (MMU)
  - Memory protection unit (MPU)
  - Instruction and data cache memories
  - Tightly-coupled memory interfaces for instructions and data
  - JTAG debug module
Processor Implementation

- The functional units of the NIOS II architecture form the foundation for the NIOS II instruction set
  - This does not indicate that any unit is implemented in hardware
  - The NIOS II architecture describes an instruction set, not a particular hardware implementation
  - A functional unit can be implemented in hardware, emulated in software, or omitted entirely
- A NIOS II implementation is a set of design choices embodied by a particular NIOS II processor core
- Each implementation achieves specific objectives, such as smaller core size or higher performance
- Allows the NIOS II architecture to adapt to the needs of different target applications

Processor Implementation Tradeoffs

- Implementation variables generally fit one of three trade-off patterns:
  - More or less of a feature
    - Example: To fine-tune performance, you can increase or decrease the amount of instruction cache memory. A larger cache increases execution speed of large programs, while a smaller cache conserves on-chip memory resources.
  - Inclusion or exclusion of a feature
    - Example: To reduce cost, you can choose to omit the JTAG debug module. This decision conserves on-chip logic and memory resources, but it eliminates the ability to use a software debugger to debug applications.
  - Hardware implementation or software emulation of a feature
    - Example: For applications that rarely perform complex arithmetic, you can choose for the division instruction to be emulated in software. Removing the divide hardware conserves on-chip resources but increases the execution time of division operations.
NIOS II Register File

- The NIOS II architecture supports a flat register file, consisting of
  - Thirty two 32-bit general-purpose integer registers, and
  - Up to thirty two 32-bit control registers
- The architecture supports supervisor and user modes that allow system code to protect the control registers from errant applications
- The NIOS II processor can optionally have one or more shadow register sets
  - A shadow register set is a complete set of NIOS II general-purpose registers
  - A typical use of shadow register sets is to accelerate context switching between tasks

Arithmetic Logic Unit

- The NIOS II ALU operates on data stored in general-purpose registers
- ALU operations take one or two inputs from registers, and store a result back in a register
- Supported Operations
  - Addition, subtraction, multiplication and division on signed and unsigned operands
  - Logical operations AND, OR, NOR, and XOR
  - Shift and Rotate
    - The ALU supports shift and rotate operations, and can shift/rotate data by 0 to 31 bit positions per instruction
    - The ALU supports arithmetic shift right and logical shift right/left
    - The ALU supports rotate left/right
Unimplemented Instructions

• Some NIOS II processor core implementations do not provide hardware to support the entire NIOS II instruction set
  – In such a core, instructions without hardware support are known as unimplemented instructions
• The processor generates an exception whenever it issues an unimplemented instruction so an exception handler can call a routine that emulates the operation in software
  – Therefore, unimplemented instructions do not affect the programmer’s view of the processor

Custom Instructions

• The NIOS II architecture supports user-defined custom instructions
• The NIOS II ALU connects directly to custom instruction logic, enabling you to implement in hardware operations that are accessed and used exactly like native instructions
  – This custom instruction feature allows for creation of one or more hardware accelerators that can dramatically improve the execution performance of a given system
  – Common examples:
    • Complex filter operations in DSP systems
    • Image Processing accelerator cores
Floating Point Instructions

- The NIOS II architecture supports single precision floating-point instructions
  - Conforms to the IEEE STD 754-1985
- The basic set of floating-point custom instructions includes single precision floating-point addition, subtraction, and multiplication
- Floating-point division is available as an extension to the basic instruction set
  - Hardware support for floating point division is optionally included because it requires more hardware resources than other operations
- Double precision arithmetic is implemented only in software
  - Be highly aware of this as it can impact performance significantly

Floating Point Software Development Considerations

- The best choice for hardware design depends on a balance among
  - Floating-point usage
  - Hardware resource usage, and
  - Performance
- While the floating-point custom instructions speed up floating-point arithmetic, they substantially add to the size of a hardware design
Floating Point Software Development Considerations (continued)

• You can use `#pragma` directives in your software to compare hardware and software implementations of the floating-point instructions

  `#pragma no_custom_fadds` - Forces software implementation of floating-point add
  
  `#pragma no_custom_fsubs` - Forces software implementation of floating-point subtract
  
  `#pragma no_custom_fmuls` - Forces software implementation of floating-point multiply
  
  `#pragma no_custom_fdivs` - Forces software implementation of floating-point divide

• The scope of the `#pragma` directive is the entire C file

Floating Point Software Development Considerations (continued)

• All the floating-point custom instructions are single-precision operations
• Double-precision operations are implemented in software
• By default, the NIOS II compiler treats floating-point constants as double-precision numbers
• To use the floating-point custom instructions for operations with floating-point constants, append an “f” to the constant
• Example code:
  – `y=x*4.67;` // Double precision math implemented in software
  – `y=x*4.67f;` // Single precision math (probably in hardware)
Memory and I/O Organization

- The flexible nature of the NIOS II memory and I/O organization are the most notable difference between NIOS II processor systems and traditional microcontrollers.
- Because NIOS II processor systems are configurable, the memories and peripherals vary from system to system.
  - As a result, the memory and I/O organization varies from system to system.

---

Memory and I/O Organization (continued)

- A NIOS II core uses one or more of the following to provide memory and I/O access:
  - Instruction master port - An Avalon Memory-Mapped (Avalon-MM) master port that connects to instruction memory via system interconnect fabric.
  - Instruction cache - Fast cache memory internal to the NIOS II core.
  - Data master port - An Avalon-MM master port that connects to data memory and peripherals via system interconnect fabric.
  - Data cache - Fast cache memory internal to the NIOS II core.
  - Tightly-coupled instruction or data memory port - Interface to fast on-chip memory outside the NIOS II core.
- The NIOS II architecture hides the hardware details from the programmer, so programmers can (usually) develop applications without specific knowledge of the hardware implementation.
The NIOS II architecture supports separate instruction and data buses, classifying it as a Harvard architecture.

- Both the instruction and data buses are implemented as Avalon-MM master ports that adhere to the Avalon-MM interface specification.
- The data master port connects to both memory and peripheral components, while the instruction master port connects only to memory components.
Memory and Peripheral Access

- The NIOS II architecture provides memory-mapped I/O access
  - Both data memory and peripherals are mapped into the address space of the data master port
- The NIOS II architecture is little endian
  - Words and halfwords are stored in memory with the more-significant bytes at higher addresses

\[
\begin{array}{cccc}
31 & N+3 & N+2 & N+1 & N
\end{array}
\]

- The NIOS II architecture does not specify anything about the existence of memory and peripherals; the quantity, type, and connection of memory and peripherals are system-dependent
  - Typically, NIOS II processor systems contain a mix of fast on-chip memory and slower off-chip memory
  - Peripherals typically reside on-chip, although interfaces to off-chip peripherals also exist.

Instruction Master Port

- The NIOS II instruction bus is implemented as a 32-bit Avalon-MM master port
  - The instruction master port performs a single function: it fetches instructions to be executed by the processor
  - The instruction master port does not perform any write operations
- The instruction master port is a pipelined Avalon-MM master port
- The instruction master port can issue successive read requests before data has returned from prior requests
- The NIOS II processor can prefetch sequential instructions and perform branch prediction to keep the instruction pipe as active as possible
- The instruction master port always retrieves 32 bits of data
  - The instruction master port relies on dynamic bus-sizing logic contained in the system interconnect fabric
  - By virtue of dynamic bus sizing, every instruction fetch returns a full instruction word, regardless of the width of the target memory
  - Consequently, programs do not need to be aware of the widths of memory in the NIOS II processor system
Data Master Port

- The NIOS II data bus is implemented as a 32-bit Avalon-MM master port. The data master port performs two functions:
  - Read data from memory or a peripheral when the processor executes a load instruction
  - Write data to memory or a peripheral when the processor executes a store instruction
- Byte-enable signals on the master port specify which of the four byte-lane(s) to write during store operations

Shared Memory for Instructions and Data

- Usually the instruction and data master ports share a single memory that contains both instructions and data
- While the processor core has separate instruction and data buses, the overall NIOS II processor system might present a single, shared instruction/data bus to the outside world
- The outside view of the NIOS II processor system depends on the memory and peripherals in the system and the structure of the system interconnect fabric
Cache Memory

- The NIOS II architecture supports cache memories on both the instruction master port (instruction cache) and the data master port (data cache)
- Cache memory resides on-chip as an integral part of the NIOS II processor core
- The cache memories can improve the average memory access time for NIOS II processor systems that use slow off-chip memory such as SDRAM for program and data storage

Configurable Cache Memory Options

- The cache memories are optional. The need for higher memory performance (and by association, the need for cache memory) is application dependent
- Many applications require the smallest possible processor core, and can trade-off performance for size
- A NIOS II processor core might include one, both, or neither of the cache memories
- Furthermore, for cores that provide data and/or instruction cache, the sizes of the cache memories are user-configurable
- The inclusion of cache memory does not affect the functionality of programs, but it does affect the speed at which the processor fetches instructions and reads/writes data
Effective Use of Cache Memory

• The effectiveness of cache memory to improve performance is based on the following premises:
  – Regular memory is located off-chip, and access time is long compared to on-chip memory
• The largest, performance-critical instruction loop is smaller than the instruction cache
• The largest block of performance-critical data is smaller than the data cache
  – There are some software coding techniques we will look at that can address this limitation to some extent (cache architecture aware coding)

Tightly-Coupled Memory

• Tightly-coupled memory provides guaranteed low-latency memory access for performance-critical applications
• Compared to cache memory, tightly-coupled memory provides the following benefits:
  – Performance similar to cache memory
  – Software can guarantee that performance-critical code or data is located in tightly-coupled memory
  – No real-time caching overhead, such as loading, invalidating, or flushing memory
Tightly-Coupled Memory (continued)

- Physically, a tightly-coupled memory port is a separate master port on the NIOS II processor core, similar to the instruction or data master port
  - A NIOS II core can have zero, one, or multiple tightly-coupled memories
  - The NIOS II architecture supports tightly-coupled memories for both instruction and data access
- Each tightly-coupled memory port connects directly to exactly one memory with guaranteed low, fixed latency
- The memory is external to the NIOS II core and is usually located on chip.

Accessing Tightly-Coupled Memory

- Tightly-coupled memories occupy normal address space, the same as other memory devices connected via system interconnect fabric
- The address ranges for tightly-coupled memories (if any) are determined at system generation time
- Software accesses tightly-coupled memory using regular load and store instructions
- From the software’s perspective, there is no difference accessing tightly-coupled memory compared to other memory
Effective Use of Tightly-Coupled Memory

• A system can use tightly-coupled memory to achieve maximum performance for accessing a specific section of code or data
• For example:
  – Interrupt-intensive applications can place exception handler code into a tightly-coupled memory to minimize interrupt latency
  – Compute-intensive digital signal processing (DSP) applications can place data buffers into tightly-coupled memory for the fastest possible data access