Excalibur Embedded Processor Solutions: Nios

Nios Soft Core Embedded Processor

FPGA Use in Embedded Applications

- **Custom Microcontroller**
  - Exact-Fit Feature Set
  - Reduce System Cost, Complexity & Power Consumption By Moving Functions From Board to FPGA
  - Avoid Obsolescence

- **Processor Companion Chip**
  - Extend System Features & Performance
  - Add Peripherals & Custom Logic Functions in FPGA

- **Multi-Processor System**
  - Boost System Performance
  - Off-Load Existing Processor
Excalibur Embedded Processor Solutions: Nios

NIOS II Overview

- **Soft IP Core**
  - A soft-core processor is a microprocessor fully described in software, usually in an HDL, which can be synthesized in programmable hardware, such as FPGAs.
- **Reduced Instruction Set Computer (RISC)**
- No pipeline, 5 or 6 stages pipeline configurations
- Full 32-bit instruction set, data path, and address space
- 32 general-purpose registers
- 32 external interrupt sources
- Access to a variety of on-chip peripherals, and interfaces to off-chip memories and peripherals
- Software development environment based on the GNU C/C++ tool chain and Eclipse IDE

NIOS II Scalability

- Powerful multiprocessing systems can be built
Excalibur Embedded Processor Solutions: Nios

Nios Embedded Processor Core

- Configurable Soft Core Processor
- 32-Bit Pipelined RISC Architecture
  - 16-Bit Instructions
  - Most Instructions Take 1 Clock
- Large Internal Register File
- Configurable Data Path
  - 16-bit (1100 LEs)
  - 32-bit (1700 LEs)
- Dynamic Bus Sizing
- 30 to 80 MIPS Performance

Nios RISC Processor Block Diagram

- Standard RISC Components
- Fully-Synchronous Interface
Excalibur Embedded Processor Solutions: Nios

NIOS II Processor Core

Implementation

- The functional units of the Nios II architecture form the foundation for the Nios II instruction set.
- The Nios II architecture describes an instruction set, not a particular hardware implementation.
- Trade-offs:
  - More or less of a feature - amount of instruction cache memory.
  - Inclusion or exclusion of a feature - the JTAG debug module.
  - Hardware implementation or software emulation - divider
Types of Processors

Memory Organization
Cache Performance

<table>
<thead>
<tr>
<th>Memory</th>
<th>I-Cache</th>
<th>D-Cache</th>
<th>Normalised Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDRAM</td>
<td>No</td>
<td>No</td>
<td>40.2%</td>
</tr>
<tr>
<td>SDRAM</td>
<td>No</td>
<td>Yes</td>
<td>55.2%</td>
</tr>
<tr>
<td>SDRAM</td>
<td>Yes</td>
<td>No</td>
<td>64.3%</td>
</tr>
<tr>
<td>SDRAM</td>
<td>Yes</td>
<td>Yes</td>
<td>96.4%</td>
</tr>
<tr>
<td>OnChip</td>
<td>No</td>
<td>No</td>
<td>100.0%</td>
</tr>
<tr>
<td>OnChip</td>
<td>No</td>
<td>Yes</td>
<td>98.0%</td>
</tr>
<tr>
<td>OnChip</td>
<td>Yes</td>
<td>No</td>
<td>110.2%</td>
</tr>
<tr>
<td>OnChip</td>
<td>Yes</td>
<td>Yes</td>
<td>105.6%</td>
</tr>
</tbody>
</table>

Performance relative to on chip RAM with no Cache running dhry.c modified for unbuffered I/O

Tightly Coupled Memory

- Fast data buffers
- Fast sections of code
- Fast interrupt handler
- Critical loop
- Constant access time; guaranteed not to have arbitration delays
- Up to 4 tightly coupled memories

Software Guidelines
- Software accesses tightly-coupled memory addresses just like any other addresses.
- Cache operations have no effect when targeting tightly-coupled
Pipelining

- Static branch prediction is implemented using the branch offset direction;
  - a negative offset is predicted as taken
  - a positive offset is predicted as not-taken

<table>
<thead>
<tr>
<th>Stage Letter</th>
<th>Stage Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>Fetch</td>
</tr>
<tr>
<td>D</td>
<td>Decode</td>
</tr>
<tr>
<td>E</td>
<td>Execute</td>
</tr>
<tr>
<td>M</td>
<td>Memory</td>
</tr>
<tr>
<td>W</td>
<td>Writeback</td>
</tr>
</tbody>
</table>

Table 5-6. Implementation Pipeline Stages for Nios II Core

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Cycles</th>
<th>Penalties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal ALU instructions (e.g., add, comp)</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Combinatorial custom instructions</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Multi-cycle custom instructions</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Branch (correctly predicted taken)</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Branch (correctly predicted not taken)</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Branch (mispredicted)</td>
<td>4</td>
<td>Pipeline flush</td>
</tr>
<tr>
<td>trap, break, ret, bret, flush, wcutl, unimplemented</td>
<td>4</td>
<td>Pipeline flush</td>
</tr>
<tr>
<td>jmp, ret, call, callr</td>
<td>4</td>
<td>Pipeline flush</td>
</tr>
</tbody>
</table>
Windowed Register File

- Common Technique Used by High-Performance CPUs
  - Provides Fast Subroutine Calls

- Up to 512 General-Purpose Registers

- Movable Window With Access to 32 Registers
  - 24 Register Window (Movable)
  - 8 Global Registers (Fixed)

- Automatically Used by C Compiler

Bit Shift Speed

- Provides Multiple Bit Shift in a Single Clock Cycle
  - Increments of up to 3, 5, 7, 15, or 31 Bits Per Clock

- Example:
  - Bit Shift Speed Set to 7:
    i << 9;  /* Shift Left by 9 Bits */
  - Executes in 2 Clocks
Excalibur Embedded Processor Solutions: Nios

**Multiplication Options**

- **Software**
  - Uses GNU Math Library for Multiply Operations

- **MSTEP**
  - Hardware Multiplier - Option 1
  - One-Bit Per Clock Multiply
  - Improvement of ~ 4X over Software Multiplication Routines

- **MUL**
  - Hardware Multiplier - Option 2
  - 16 x 16 → 32 in 2 Clocks

<table>
<thead>
<tr>
<th>Multiplier</th>
<th>Additional Logic Elements Used</th>
<th>Clock Cycles 16x16→32</th>
<th>Clock Cycles 32x32→32</th>
</tr>
</thead>
<tbody>
<tr>
<td>None (Software)</td>
<td>0</td>
<td>80</td>
<td>250</td>
</tr>
<tr>
<td>MSTEP</td>
<td>+200</td>
<td>18</td>
<td>80</td>
</tr>
<tr>
<td>MUL</td>
<td>+400</td>
<td>2</td>
<td>16</td>
</tr>
</tbody>
</table>

**Development Tool Flow**
Excalibur Embedded Processor Solutions: Nios

Nios System Builder Software

- Configure Processor
  - Peripheral Library
    - Process Design (Verilog / VHDL)
    - Simulation Test Bench
  - Select Peripherals
  - Generate
    - Synthesis Place & Route
      - JTAG Serial Ethernet
      - Hardware Configuration File
    - Download & Debug
      - Altera PLD
      - Executable Code
  - Software
    - GNU Compiler
      - User Code
      - S/W Libraries
      - RTOS
    - C Header files
    - Custom Library
    - Boot monitor

Peripheral Devices
Excalibur Embedded Processor Solutions: Nios

**UART Peripheral**

Provides common serial interface with variable baud rate, parity, stop and data bits, and optional flow control signals

---

**Adding a UART Peripheral**

Block Diagram of the UART Core in a Typical System

[Diagram of UART core and connections]

CSE 467
Excalibur Embedded Processor Solutions: Nios

Timer Peripheral

- 32-bit and 64-bit counters.
- Controls to start, stop, and reset the timer.
- Two count modes: count down once and continuous count-down.
- Count-down period register.
- Option to enable or disable the interrupt request (IRQ) when timer reaches zero.
- Optional watchdog timer feature that resets the system if timer ever reaches zero.
- Optional periodic pulse generator feature that outputs a pulse when timer reaches zero.
- Compatible with 32-bit and 16-bit processors.
Excalibur Embedded Processor Solutions: Nios

Parallel I/O (PIO) Peripheral

PIO Peripheral

- 1 to 32-bit Parallel I/O Port
  - Input Only
  - Output Only
  - Bi-directional Port
    - On-chip: Separate Ports for Input & Output
    - Off-chip: Tri-State Control
- Edge Detection on Inputs
- Interrupt Generation
  - Maskable
  - IRQ Source
    - Input Level
    - Edge Detection Register
SPI Port

- Full Duplex, Synchronous Serial Interface
  - Interface to:
    - A/D, D/A
    - Microcontrollers
    - Serial EPROM
- 3-Wire Serial Communications Bus With Slave Select
  - Master Out Slave In - MOSI
  - Master In Slave Out - MISO
  - SPI Clock - SCLK
  - Slave Select - SS_n (Optional)
- Master or Slave Operation
- Supports Up to 16 Slave Devices
- Programmable Word Size (1 to 16 bits)
- Programmable Delay Slot (Enable-to-Active)
Excalibur Embedded Processor Solutions: Nios

**Ethernet Interface**

**Ethernet Port**

- 10/100-Mbps SMSC LAN91C111 single-chip Ethernet controller interface
- Software support provided by the NicheStack TCP/IP Stack - Nios II Edition
- Included with the Nios II development kits
User-Defined Interface

- Interface to Other Peripherals
  - On-Chip & Off-Chip
- Configures Busses and Timing
- Adds Port Signals to Design
Memory Interfaces

- On-chip ROM and RAM
- SDRAM
- SSRAM
- SRAM
- Flash
- Common Flash Interface Controller Core
- Altera serial configuration devices
Excalibur Embedded Processor Solutions: Nios

Common Flash Interface Controller Core

Allows you to easily connect SOPC Builder systems to external flash memory that complies with the Common Flash Interface (CFI) specification.

Common Flash Interface Controller Core

An SOPC Builder System Integrating a CFI Controller
Creating Nios based systems using SOPC and program it using IDE

SOPC builder

Nios II IDE

SOPC Builder Design Flow

Simulation Testbench Generation

System Generation VHDL/Verilog HDL

Software Development Kit Generation

- C Header files
- Custom Library
- Peripheral Drivers
- RTOS
- Middleware
- Software IDE
- Software Debuggers

CSE 467
Hardware Abstraction Layer (HAL)

- Isolates the application software from hardware modifications.
- Applications are device-independent because they abstract information from such systems as:
  - Character mode devices: UART core, JTAG UART core, LCD display controller
  - Flash memory devices
  - Timer devices
  - DMA controller core
  - Ethernet MAC PHY Controller
- HAL application program interface (API) is integrated with the ANSI C standard library.

Layers of HAL API

- HAL library generation:
  1. SOPC Builder generates a hardware system
  2. Nios II IDE generates a custom HAL system library to match the hardware configuration
- Changes in the hardware configuration automatically propagate to the HAL device driver configuration
- NIOS II is programmed in C

![HAL API Layers Diagram]
Programming NIOS II Processor

- Programming UART
  - Standard Input, Standard Output routines in C

```c
#include <stdio.h>
#include <string.h>

int main (void)
{
  char* msg = "hello world";
  FILE* fp;
  fp = fopen("/dev/uart1","w");
  if (fp)
    {
    fprintf(fp,"%s",msg);
    fclose(fp);
    }
  return 0;
}
```

References
