The Programmable Solutions Company®

- **Devices**
  - Stratix II™
  - Stratix
  - Stratix GX
  - Cyclone II™
  - Cyclone
  - MAX® II

- **Intellectual Property (IP)**
  - Signal Processing
  - Communications
  - Embedded Processors
    - Nios®, Nios II

- **Devices (continued)**
  - Mercury™ Devices
  - ACEX® Devices
  - FLEX® Devices
  - MAX® Devices

- **Tools**
  - Quartus® II Software
  - Quartus II Web Edition
  - SOPC Builder
  - DSP Builder
  - Nios II IDE

Copyright © Altera Corporation
What is Nios II?

- Altera’s Second Generation Soft-Core 32 Bit RISC Microprocessor
  - Nios II Plus All Peripherals Written In HDL
  - Can Be Targeted For All Altera FPGAs
  - Synthesis Using Quartus II Integrated Synthesis

FPGA

Nios II

CPU

Debug

Cache

Avalon Switch Fabric

UART

GPIO

Timer

SPI

SDRAM Controller

On-Chip

ROM

On-Chip

RAM
Problem: Reduce Cost, Complexity & Power

Solution: Replace External Devices with Programmable Logic

System On A Programmable Chip (SOPC)

CPU is a Critical Control Function Required for System-Level Integration
FPGA Hardware Design Flow

**Design Specification**
- SOPC Builder
  - Functional Simulation (Modelsim, Quartus II)
  - Verify Logic Model & Data Flow (No Timing Delays)

**Synthesis**
- Translate Design into Device Specific Primitives
- Optimization to Meet Required Area & Performance Constraints
- Spectrum, Synplify, Quartus II

**Place & Route**
- Map Primitives to Specific Locations Inside Target Technology with Reference to Area & Performance Constraints
- Specify Routing Resources to Be Used

**FPGA Hardware Design Flow**

**Timing Analysis**
- Verify Performance Specifications Were Met
- Static Timing Analysis

**Gate Level Simulation**
- Timing Simulation
- Verify Design Will Work in Target Technology

**Test FPGA on PC Board**
- Program & Test Device on Board
- Use SignalTap II for Debugging
Nios II System Architecture

Nios II Block Diagram
Nios II Processor Architecture

- Classic Pipelined RISC Machine
  - 32 General Purpose Registers
  - 3 Instruction Formats
  - 32-Bit Instructions
  - 32-Bit Data Path
  - Flat Register File
  - Separate Instruction and Data Cache (configurable sizes)
  - Branch Prediction
  - 32 Prioritized Interrupts
  - Custom Instructions
  - JTAG-Based Hardware Debug Unit

Nios II Versions

- Nios II Processor Comes In Three ISA Compatible Versions
  - FAST: Optimized for Speed
  - STANDARD: Balanced for Speed and Size
  - ECONOMY: Optimized for Size

- Software
  - Code is Binary Compatible
    - No Changes Required When CPU is Changed
Binary Compatibility / Flexible Performance

<table>
<thead>
<tr>
<th></th>
<th>Nios II /f Fast</th>
<th>Nios II /s Standard</th>
<th>Nios II /e Economy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>6 Stage</td>
<td>5 Stage</td>
<td>None</td>
</tr>
<tr>
<td>H/W Multiplier &amp; Barrel Shifter</td>
<td>1 Cycle</td>
<td>3 Cycle</td>
<td>Emulated In Software</td>
</tr>
<tr>
<td>Branch Prediction</td>
<td>Dynamic</td>
<td>Static</td>
<td>None</td>
</tr>
<tr>
<td>Instruction Cache</td>
<td>Configurable</td>
<td>Configurable</td>
<td>None</td>
</tr>
<tr>
<td>Data Cache</td>
<td>Configurable</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Logic Usage (Logic Elements)</td>
<td>1400 - 1800</td>
<td>1200 – 1400</td>
<td>600 – 700</td>
</tr>
<tr>
<td>Custom Instructions</td>
<td>Up to 256</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Hardware Multiplier Acceleration

- **Nios II Economy version - No Multiply Hardware**
  - Uses GNUPro Math Library to Implement Multiplier

- **Nios II Standard - Full Hardware Multiplier**
  - 32 x 32 $\rightarrow$ 32 in 3 Clock Cycles if DSP block present, else uses software only multiplier

- **Nios II Fast - Full Hardware Multiplier**
  - 32 x 32 $\rightarrow$ 32 in 1 Clock Cycles if DSP block present, else uses software only multiplier

<table>
<thead>
<tr>
<th>Acceleration Hardware</th>
<th>Clock Cycles (32 x 32 $\rightarrow$ 32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>250</td>
</tr>
<tr>
<td>Standard MUL in Stratix</td>
<td>3</td>
</tr>
<tr>
<td>Fast MUL in Stratix</td>
<td>1</td>
</tr>
</tbody>
</table>
Variation with FPGA Device

**Nios II**

- **Fast**
- **Standard**
- **Economy**

**FPGA Device Variations**

<table>
<thead>
<tr>
<th>FPGA Device</th>
<th>Nios II/f</th>
<th>Nios II/s</th>
<th>Nios II/e</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Stratix II</strong></td>
<td>200 DMIPS @ 175MHz 1180 LEs 1 of 8 DSP 4K Icache, 2K Dcache Stratix 2S10-C5</td>
<td>90 DMIPS @ 175MHz 800 LEs 4K Icache, No Dcache Stratix 2S10-C5</td>
<td>28 DMIPS @ 190MHz 400 LEs No Icache, No Dcache Stratix 2S10-C5</td>
</tr>
<tr>
<td><strong>Stratix</strong></td>
<td>150 DMIPS @ 135MHz 1800 LEs 1 of 8 DSP 4K Icache, 2K Dcache Stratix 1S10-C5</td>
<td>67 DMIPS @ 135MHz 1200 LEs 4K Icache, No Dcache Stratix 1S10-C5</td>
<td>22 DMIPS @ 150MHz 550 LEs No Icache, No Dcache Stratix 1S10-C5</td>
</tr>
<tr>
<td><strong>Cyclone</strong></td>
<td>100 DMIPS @ 125MHz 1800 LEs 4K Icache, 1K Dcache Cyclone 1C4-C6</td>
<td>62 DMIPS @ 125MHz 1200 LEs 2K Icache, No Dcache Cyclone 1C4-C6</td>
<td>20 DMIPS @ 140MHz 550 LEs No Icache, No Dcache Cyclone 1C4-C6</td>
</tr>
</tbody>
</table>

* FMax Numbers Based Reference Design Running From On-Chip Memory (Nios II/f \( \geq 1.15 \) DMIPS / MHz)
Nios II CPU Configured in SOPC Builder

- Hardware designer selects which Nios II version to use when creating system
Selecting JTAG Debug Core

- Configuration is chosen when hardware designer selects appropriate Nios II processor core
SOPC Builder – System Generation Page

SOPC Builder Produces a .PTF File

- Text file that records SOPC Builder edits
- Describes Nios II System
- Used by software development tools
Integrate SOPC Builder O/P in Quartus II

- Integrate SOPC Builder block symbol to Quartus II schematic (as shown below) and compile design
- Or, instantiate top module into your HDL design and compile

New Peripherals for Nios II

- **System ID Peripheral**
  - Used to Ensure Hardware/Software Version Synchronization
  - Simple 2 read-only register peripheral containing hardware ID tags.
    - Register 1 contains random number
    - Register 2 contains time and date when system was generated in SOPC Builder
  - Can be checked at runtime to ensure that the software to be downloaded matches the hardware image

- **Memory Interfaces**
  - EPCS Serial Flash Controller
  - On-Chip
    - RAM, ROM
  - Off-Chip
    - SRAM
    - CFI Flash

- **LCD Display**
New Peripherals for Nios II

- **JTAG UART**
  - Single JTAG Connection For:
    - Device Configuration
    - Flash Programming
    - Code Download
    - Debug
    - Target STDIO (printing)

- **Compact Flash Interface**
  - Mass Storage Support
    - True IDE Mode
    - Compact Flash Mode
  - Software Supports
    - Low-Level API
    - MicroC/OS-II File System Support
    - μCLinux File System Support

Project Directories

- **Hardware**
  - HDL Source & Netlist
  - db - Quartus project database

- **Software**
  - Application source code
  - Library files

- **Simulation**
  - Testbench
  - Automatically generated test memory and vectors
Nios II Software Development

SOPC Builder Flow

Hardware Development
- Processor Library
- Peripheral Library
  - HDL Source Files
  - Testbench
- Synthesis & Fitter
  - User Design
  - Other IP Blocks
- Quartus II

SOPC Builder GUI
- Configure Processor
- Select & Configure Peripherals, IP
- Connect Blocks
- Generate

Software Development
- Custom Instructions
- IP Modules
- Nios II IDE
  - C Header files
  - Custom Library
  - Peripheral Drivers
- Compiler, Linker, Debugger
  - User Code
  - Libraries
  - RTOS
- GNU Tools
  - User Code
  - Libraries
  - RTOS

Altera PLD
- On-Chip Debug
  - Software Trace
  - Hard Breakpoints
  - SignalTap® II
- Altera PLD
Nios II IDE (Integrated Development Environment)*

- Leading Edge Software Development Tool
- Target Connections
  - Hardware (JTAG)
  - Instruction Set Simulator
  - ModelSim®-Altera Software
- Advanced Hardware Debug Features
  - Software and Hardware Break Points, Data Triggers, Trace
- Flash Memory Programming Support

* Based on Eclipse Project

Copyright © 2005 Altera Corporation

Opening the Nios II IDE

Launch the Nios II IDE from the SOPC Builder or from the Windows Start menu
**Nios II IDE**

- List of Open Projects
- File Viewer Window (for C code, C++, and assembly*)
- Terminal window

*Note: C++ files must have extension .cpp
In-line assembly code offset by asm();

---

**Nios II IDE C/C++ Projects/Navigator**

- Lists all open projects
- Displays source files associated with project
- List all open and closed projects
- Allows you to drag and drop new files into existing projects

---

Copyright © 2005 Altera Corporation
Creating a C/C++ Application

File > New > Project

Link to a System Library
- Select a pre-existing library
- Or create a new library
This Creates Two Software Projects
- Application and System Library Project

Application Project
- contains application source code

System Library Project
- contains system header file, etc.

Drivers Directory
- contains all device drivers – DO NOT DELETE!

Application and System Library Projects

- **Application Projects** build executables
- **System Library Projects** contain interface to the hardware
  - Nios II device drivers (Hardware Abstraction Layer)
  - Optional RTOS (MicroC/OS-II)
  - Optional software components (Lightweight TCP/IP stack, Read Only Zip File System)
System Library Options

Partition the memory map

Software Compilation

- To compile a software application, highlight your project and select **Build Project** from the Projects menu.
Directory Structure After Compilation

- **Application Project**
  - hello_world
  - Binaries
  - Debug
    - hello_world.elf [altera03.3e]
    - ext_fsh.fsh
    - find_sopc_component_dir.dat
    - generated_app.sh
    - makefile
    - subdir.mk
  - hello_world.c
  - stdlib
    - main0.int
  - application.sif
  - readme.txt

- **System Library Project**
  - hello_world
  - Binaries
  - Debug
    - obj
      - system_description
      - system.h
    - intf
      - hello_world_syslib
      - hello_world.h
    - generated.gdb
    - generated.sh
    - generated.x
    - generated_all.mk
    - generated_app.mk
    - hello_world_syslib.mk
    - hello_world.sif
    - readme.txt
    - system.sif

---

Hardware Abstraction Layer

- A lightweight runtime environment for Nios II software
  - Provides a level of abstraction between application code and low-level hardware
- HAL libraries are generated by Nios II IDE
- A HAL contains:
  - device drivers
  - initialization software
  - file system
  - stdio, stderr
Hardware Abstraction Layer

- Provides generic device models for classes of peripherals common in embedded systems
  - eg. timers, I/O peripherals, etc.
- Gives a consistent POSIX-like API, regardless of underlying hardware
- Make programming as familiar as possible to software engineers who may not be familiar with the specific peripheral architectures
  - ANSI C (through the Newlib library)
  - UNIX style interface (i.e. POSIX like)
  - Altera extensions where standards don’t exist or were inappropriate (watch for the alt_* extension)

Nios II HAL: Runtime Library

The HAL ‘UNIX Style’ Functions are the glue between the C library and the device drivers
HAL File System

- Device names match those set in SOPC builder.
- Can only access nodes, not directories.
- All paths must be absolute (no current directory)

Familiar File/Device Access

- ANSI C:
  ```c
  fp = fopen("/dev/lcd0", "w"); fprintf(fp, "%s", msg);
  ```

- UNIX Style:
  ```c
  fd = open("/dev/lcd0", O_WRONLY); write(fd, msg, strlen(msg));
  ```

- Newlib also supports C++ streams:
  ```cpp
  ofstream ofp("/dev/lcd0", ios::out); ofp << msg;
  ```

- Existing code (outside the Nios world) uses these interfaces. Porting is now much easier.

- Use of existing standards means there’s nothing new to learn.
system.h

- Contains macro definitions for system parameters, including peripheral configuration, for instance:
  - Hardware configuration of the peripheral
  - Base address
  - IRQ priority (if any)
  - Symbolic name for peripheral
- Does not include: static information, function prototypes, or device structures (unlike the old excalibur.h)
- Located in the syslib project directory
- Rarely necessary to include it explicitly in your application code, which improves rebuild time
system.h - example

- Defines system settings and peripheral configurations:
  - Replaces excalibur.h (from Nios)

/*
 * system configuration
 *
*/
#define ALT_IRQ_BASE NULL
#define ALT_CPU_FREQ 50000000
#define ALT_CPU_ARCHITECTURE "altera_nios2"
#define ALT_DEVICE_FAMILY "STRATIX"
#define ALTERA_NIOS_DEV_BOARD_STRATIX_1S10_ES
#define ALT_STDIN "/dev/iag_uart"
#define ALTSTDOUT "/dev/iag_uart"
#define ALT_STDEV "/dev/iag_uart"
#define ALT_CPU_FREQ 50000000
#define ALT_CPP_CONSTRUCTORS
#define ALT_IRQ_BASE NULL
*/
#define BUTTON_PIO_IRQ_TYPE "EDGE"
#define BUTTON_PIO_EDGE_TYPE "ANY"
#define BUTTON_PIO_CAPTURE 1
#define BUTTON_PIO_NAME "/dev/button_pio"
#define BUTTON_PIO_HAS_TRI 0
#define BUTTON_PIO_BASE 0x00920830

HAL References

- Each HAL project references library routines and drivers for the components included in your Nios II system
Reading/Writing Hardware in Nios II

- Instead use I/O macros to access hardware
  - I/O macros bypass the cache for hardware accesses
  - They set bit 31 of address bus high (i.e., control bit)

  - **IORD(BASE, REGNUM)**
    - Reads value at register REGNUM offset from base address BASE

  - **IOWR(BASE, REGNUM, DATA)**
    - Writes DATA to register REGNUM offset from base address BASE

Header Files for Nios II Peripherals

- Each Nios II peripheral has specific read/write macros for each register
  - Example: UART (altera_avalon_uart_regs.h)

```
#define IORD_ALTERA_AVALON_UART_RXDATA(base)        IORD(base, 0)
#define IOWR_ALTERA_AVALON_UART_RXDATA(base, data)  IOWR(base, 0, data)
#define IORD_ALTERA_AVALON_UART_TXDATA(base)        IORD(base, 1)
#define IOWR_ALTERA_AVALON_UART_TXDATA(base, data)  IOWR(base, 1, data)
#define IORD_ALTERA_AVALON_UART_STATUS(base)        IORD(base, 2)
#define IOWR_ALTERA_AVALON_UART_STATUS(base, data)  IOWR(base, 2, data)
```
Interrupts

- **HAL API for ISRs - Functions**
  - `alt_irq_register()`
    - Associates interrupt with your ISR function.
  - `alt_irq_disable_all()`
    - Disables all IRQs
  - `alt_irq_enable_all()`
    - Enables all IRQs
  - `alt_irq_interruptible()`
    - Used in ISR function body. Allows ISR to be interrupted by higher priority IRQs.
  - `alt_irq_non_interruptible()`
    - Used to make ISRs uninterruptible (default behavior).

Nios II OS / RTOS Support

<table>
<thead>
<tr>
<th>Product</th>
<th>Provider</th>
<th>Source Code</th>
<th>Standards</th>
<th>TCP/IP Stack</th>
<th>File System</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Lightweight IP TCP/IP Stack</td>
<td>Open Source</td>
<td>Yes</td>
<td>Sockets API IP, ICMP, UDP, TCP</td>
<td></td>
<td></td>
<td>μC/OS-II Support</td>
</tr>
<tr>
<td>** Nucleus Plus</td>
<td>ATI/Mentor</td>
<td>Yes</td>
<td>OSEK, μTRON</td>
<td>Opt.</td>
<td>Opt.</td>
<td>GUI, SNMP, RMON, SPAN</td>
</tr>
<tr>
<td>μCLinux</td>
<td>Open Source (GPL)</td>
<td>Yes</td>
<td>Incl. Many, inc. FAT and JFFS2</td>
<td></td>
<td></td>
<td>Extensive drivers and middleware, inc USB, IPSec, etc.</td>
</tr>
</tbody>
</table>

* Included in Nios II Development Kits
** Evaluation Version Included in Nios II Development Kits

<continued on next slide>
Nios II MicroC/OS-II

- Single-seat developers license included for free with Nios II kits
- Licensing fee req’d when you productize your system
- Full source code included
- Preemptive operating system
- Small footprint
  - Code Size (min 5KB, max 20KB)
  - Data Space (min 1KB, max 5KB)
- Supports Semaphores, and Mailboxes for task synchronization
Running Code On A Target

- Nios II IDE can be used to download code to target board

```
#include <stdio.h>

int main()
{
    printf("Hello from Nios II!");
    return 0;
}
```
Running Code On A Target

- Download messages, stdout and stdin appear in console window

Nios II IDE Run Options

- Nios II IDE > Run > Run…
Nios II IDE JTAG Debugger

- Requirements
  - Must have JTAG Debug Core enabled in CPU

Nios II IDE Debug Perspective

Basic Debug
- Run Controls
- Stack View
- Active Debug Sessions

Double-click to add breakpoints

Memory View
- Variables
- Registers
- Signals
Nios II IDE Debugger

- Standard debug windows
  - memory
  - registers
  - Variables
  - breakpoints
  - expressions
  - signals
Nios II IDE: Debugger

- Debug each CPU by selecting its program thread
Nios II/f – Fast version

- Pipelined RISC Architecture
- 32-Bit Instruction and Data Paths
- 6 Stage Pipeline
- 32 General Purpose Registers
- 32 External Interrupt Sources
- Configurable Size Instruction Cache
- Dynamic Branch Prediction
- Hardware Multiply
- Barrel Shifter
- Custom Instructions
- Configurable Size Data Cache
- Hardware Breakpoints
- Optional Hardware Divide

- 135MHz
- 1.2 DMIPS/MHz
- <1800 LEs and <900 ALMs

Nios II/s – Standard version

- Pipelined RISC Architecture
- 32-Bit Instruction and Data Paths
- 5 Stage Pipeline
- 32 General Purpose Registers
- 32 External Interrupt Sources
- Configurable Size Instruction Cache
- Branch Prediction
- Hardware Multiply
- Barrel Shifter
- Custom Instructions

- 135MHz
- 0.75 DMIPS/MHz
- <1400 LEs and <700 ALMs
Nios II/e – Economy version

- Pipelined RISC Architecture
- 32-Bit Instruction and Data Paths
- 5 Stage Single Instruction Pipeline
- 32 General Purpose Registers
- 32 External Interrupt Sources
- Custom Instructions

- 150 MHz
- 0.16 DMIPS/MHz
- <700 LEs and <350 ALMs