Embedded Processor Solutions
Agenda

- Excalibur™ Embedded Processor Solutions
- Nios™ Processor Architecture
- Development Tool Flow
- System Reference Design
- Creating the Hardware Design
- Software Development with GNUPro for Nios
- Creating a “Flash-Bootable” Image
Excalibur Integrates Embedded Processors

Excalibur Embedded Processor Solutions Provide the Programmable Flexibility of PLDs with MPU Compute Performance for Fast Time-to-Market in an SOPC Solution.

Complete SOPC Solution
Altera® SOPC Innovation

1999

- High-Density, Feature-Rich SOPC Delivery Vehicle
- Methodology for SOPC Development
- Building Blocks & Design Reuse in SOPC Applications

2000

- Embedded Processor Solutions
- Processor Core & Compiler Licensees
- Wide-Open Business Model

© 2000
Excalibur Benefits

Traditional PLD Benefits

- Faster Time-to-Market
- Programmable Flexibility
- Low Risk

New Benefits

- Cost-Effective Access to Embedded Processor Cores
- Higher Integration
Excalibur Embedded Processor Solutions

Excalibur Solutions Provide Flexibility & Horsepower for Broad Market Coverage

Performance (MIPs)

ARM Core

MIPS Core

Nios™ Core

Soft Core

Hard Cores
ARM & MIPS Hardcore Embedded Processors
ARM®- & MIPS-Based™ Devices Overview

- 32-Bit System-on-a-Programmable-Chip (SOPC) Embedded Processor
  - Combines Embedded Processor “Stripe” with Programmable Logic Device (PLD)
  - 200-MHz Operation

- PLD Provides System Customization
  - Multi-Master, Multi-Slave Implementation

- Embedded Processor Stripe Embeds:
  - Processor, Caches & Memory Management Unit (MMU)
  - Large SRAM
  - Multiple Dual-Port RAM Blocks
  - Segmented High-Speed Bus & PLD Bridges
  - SDR/DDR SDRAM Controller
  - Flash Interface
  - Useful Peripherals
Block Diagram

- Embedded Processor Stripe
  - DPRAM
  - SRAM (Single Port)
  - SDRAM Interface
  - Flash Interface
  - SDRAM Controller
  - External Bus Interface
  - Phase-Locked Loops
  - ARM- or MIPS-Based Processor
  - Dual-Port RAM Interface
  - Master Port
  - Slave Port
  - PLD
  - Bridge
ARM- & MIPS-Based Devices Overview

- Three Members Spanning:
  - 4,160 to 38,400 Logic Elements (LEs)
  - 64 to 256 Kbytes SRAM
  - 8 to 128 Kbytes Dual-Port RAM

- MegaWizard® Plug-Ins
  - Attach Peripherals to Bus
  - Configure System Resources
  - Generate Address Maps

- Flexible Operating Modes
  - Real-Time Operating System (RTOS) Support
  - Different Initialization Modes
  - On-Board Flash Programming
  - Shared I/O
ARM-Based Excalibur Devices

- Based on ARM922 (ARM920 Derivative) & Incorporating ARM9TDMI
- High-Speed Cache (8-Kbyte Instruction & 8-Kbyte Data)
- Single-Cycle Repeat-Rate SRAM & DPRAM
- MMU Facilitates Virtual Memory Support & Implementation of RTOS
- ~200 MHz on Altera 0.18-m Process at TSMC
- Advanced Built-In System Debugging Features
- Fast Context Switch
MIPS-Based Excalibur Devices

- Based on MIPS32™ 4KC™ Core
- High-Speed Cache (16-Kbyte Instruction & 16-Kbyte Data)
- Single-Cycle Repeat-Rate SRAM & DPRAM
- MMU Facilitates Virtual Memory Support & Implementation of RTOS
- ~200 MHz on Altera 1P/6M, 0.18-m Process at TSMC
- Advanced Built-In System Debugging Features (EJTAG)
- Fast Multiply/Divide Unit Allows 32-Bit × 16-Bit MAC Instructions to Be Issued Every Cycle
  - 32-Bit × 32-Bit MAC Instructions Can Be Issued Every 2 Clock Cycles
Device Organization

- Processor & Interfaces
- SRAM & DPRAM
  - EPXA1
  - EPXM1
- SRAM & DPRAM
  - EPXA4
  - EPXM4
- SRAM & DPRAM
  - EPXA10
  - EPXM10

Embedded Processor Stripe
PLD
Structure of EPXA10

- ARM922T Processor
- ETM9 Trace Module
- I/O
- Fixed Logic

Embedded Processor Section

- 2 PLLs
- 128K (4K x 8) Dual-Port RAM Blocks
- 256K (8K x 32) Single Port SRAM Blocks
- Fixed Logic

Programmable Logic Section

- 160 MegaLAB™ Structures (4 Columns x 40 Rows)
- 24 MegaLAB Logic Array Blocks Each Consist of 10 Logic Elements
- Embedded System Block (2,048 Bits of Memory Configured)
  - Additional Logic Resources
  - Content-Addressable Memory (CAM)
  - Dual-Port RAM
  - Read-Only Memory (ROM)
  - First-In First-Out (FIFO) Buffers
Embedded Processor Operation

- Independent of PLD
  - Boots from External Memory
  - Interacts with External Devices
  - Configures & Updates PLD Functionality
  - Programs External Flash Devices
  - Runs Interactive Debugging Sessions
AMBA™ High-Performance Bus (AHB)

- AMBA: Advanced Microcontroller Bus Architecture
- Connects Embedded Processor & PLD Master & Slave Devices
- 100- to 200-MHz Clock Rates
- 32-Bit Wide Pipelined Bus
  - Burst Transfers
  - Non-Tri-State Implementation
- Multi-Master with Distributed Address Decoding
  - Single-Cycle Bus Master Handover
- Split Transactions Extensions
  - Needed to Fully Exploit Bus Bandwidth in Multi-Master Bus
- AHB Increasingly “The” SOC/SOPC Bus Standard
Hardware Development Tools

- **Design Entry**
  - Verilog HDL, VHDL, Altera Hardware Description Language (AHDL), or Schematic Entry

- **Simulation**
  - Quartus™ Software (ModelSim Tool)
  - Other Third-Party Tools
  - Cycle-Accurate Models of Embedded Processor Provided

- **Synthesis**
  - Quartus Software
    - Mentor Graphics LeonardoSpectrum Tool
    - Synopsys FPGA Compiler II
    - Synopsys FPGA Express Tool
  - Other Third-Party Tools
Software Development Tools

- Design Entry
  - Configuration MegaWizard Plug-Ins
    - Hard Core Operation
    - Memory Map
    - Bus Development Kit
  - Peripheral Device Drivers
  - C/C++-Aware Text Editor

- Compilation
  - C/C++ Compiler/Assembler/Linker/Debugger

- ARM Tools Will Be Primary Route for EPXA Parts
  - Basic Version within Quartus Software (No-Cost Addition for Subscribers)
  - Enhanced Features (e.g., Embedded Trace Support) Available with Upgrade to Full Excalibur Version of ARM Development Suite (ADS)
Nios Soft Core
Embedded Processor
Nios Flexibility & Scalability

High-Performance Embedded Processor
- APEX™ EP20K100E
  - 75K Gates Available

Custom DSP
- APEX EP20K200E
  - 150K Gates Available

Multi-Processor Micro-Coded System
- APEX EP20K1000E
  - 500K Gates Available
Excalibur Nios Roadmap

Extensions for Lower Cost & Higher Performance
Excalibur Nios Embedded Processor Core

- Configurable Soft Core Optimized for Altera PLD Architecture
- 32-Bit RISC Architecture
- Up to 50-MIPS Performance
Excalibur Nios Embedded Processor Core

- Most Instructions Execute in 1 Clock
- Large Internal Register File
- 1,100 Logic Cells (16-Bit Data Path)
- 1,700 Logic Cells (32-Bit Data Path)

Volume Price Point
$5 for 50 MIPs
## Benchmark Comparisons

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Processor</th>
<th>Clock (MHz)</th>
<th>MIPS</th>
<th>MIPS/MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altera</td>
<td>Nios (32-Bit)</td>
<td>50</td>
<td>44</td>
<td>0.88</td>
</tr>
<tr>
<td>ARM</td>
<td>7Thumb</td>
<td>59</td>
<td>53</td>
<td>0.90</td>
</tr>
<tr>
<td>Motorola</td>
<td>M-Core</td>
<td>33</td>
<td>31</td>
<td>0.94</td>
</tr>
<tr>
<td>Motorola</td>
<td>68360</td>
<td>33</td>
<td>8.3</td>
<td>0.25</td>
</tr>
<tr>
<td>Altera</td>
<td>Nios (16-Bit)</td>
<td>50</td>
<td>44</td>
<td>0.88</td>
</tr>
<tr>
<td>Hitachi</td>
<td>H8S</td>
<td>25</td>
<td>12</td>
<td>0.50</td>
</tr>
<tr>
<td>Philips</td>
<td>XA (8051)</td>
<td>33</td>
<td>5</td>
<td>0.15</td>
</tr>
<tr>
<td>Motorola</td>
<td>68HC16</td>
<td>25</td>
<td>2.5</td>
<td>0.10</td>
</tr>
</tbody>
</table>
Nios RISC Processor Block Diagram

- Standard RISC Components
- Fully-Synchronous Interface
Windowed Register File

- Common Technique Used by High-Performance CPUs
  - Up to 512 General-Purpose Registers
  - Movable Window Provides Access to 32 Registers
- Fast Subroutine Calls
- Automatically Used by 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:
    l << 9;  /* Shift Left by 9 Bits */

Before

| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

After

| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

- Requires 2 Clocks
  - Shift by 7
  - Shift by 2
Multiply: MSTEP

- Optional Instruction
  - Selected During Nios CPU Configuration
  - Add Logic to the Design
  - One-Bit-Per-Clock Multiplication

- Included in Software Library
  - Adds Entry to Custom SDK “Make” File
  - Multiply Will Use MSTEP Assembly Instruction
Development Tool Flow
Nios Work Flow

Nios Processor \(\rightarrow\) Configure Processor

Peripheral Pool \(\rightarrow\) Select Peripherals

Hardware

- Verilog/VHDL files
- Test Bench

- Quartus™ & LeonardoSpectrum™*
  - User Design
  - Other IP

Software

- C Header files
- Peripheral drivers

- Cygnus/Red Hat GNUPro
  - User Code
  - Libraries
  - RTOS

Download Cable

Generate

Debug Info

SOF

PC Trace

JTAG/Serial

APEX
Nios MegaWizard™ Plug-Ins

- Generate:
  - HDL Files
  - System Test Bench
Hardware Generation

- **3rd Party Synthesis**
  - Exemplar: LeonardoSpectrum
  - Synopsis: FPGA Express
  - Synplicity: Synplify

- **Quartus: Place & Route**
  - Input: Netlist
  - User Actions:
    - Select APEX Device
    - Assign Signals to Pins
  - Output: Hexadecimal Output File
Excalibur Development Kit

- Altera 32-Bit RISC CPU
- Peripherals
  - UART
  - Timer
  - Parallel I/O (PIO)
  - Memory I/F
  - Generic Port I/F
  - On-Chip Bus
- Development Board
  - Reference Design
- Development Tools
  - Quartus Excalibur Edition
  - LeonardoSpectrum
  - FPGA Express
  - Cygnus GNUPro

Excalibur Development Kit Featuring Nios $995
Software Generation

- GNUPro C Development Kit
  - Compiler
  - Assembler
  - Linker
  - Debugger
  - Nios Software Libraries
- Development Utilities
  - Compile
  - Download
  - Boot ROM
  - SREC to Simulation Files
- Example Code (C & Assembly)
Integration Tools

- Insight Debugger
  - Run Control
  - Breakpoints
  - Single Step
  - Memory Examination

The Source window lets you visually debug mixed source and assembly language.

The Registers window reveals the state of all registers and allows for modifications on the fly.

The Breakpoint window displays a tabular view of standard, conditional, and temporary breakpoints.

The Local Variables window allows you to view and modify variables and analyze data structures.
Integration Tools

- SignalTap Plus System Analyzer*
  - Processor Trace
    - Inverse Assembly
  - SignalTap ELA
    - Internal Bus Analysis

* Coming in 2001 (Not Included in Nios Development Kit)
System Reference Design
Nios System Block Diagram

Processor Core

Peripheral Bus Module

- Address Decode
- Interrupt Control
- Wait State Generation
- Data In Multiplexer
- Bus Sizing (Optional)

Port Interface

Peripherals

- UART
- Timer
- External Memory
- Internal Memory
- User-Defined Peripheral
- User-Defined Peripheral
Nios Development Board
Nios Reference Design

Legend

- On-Chip (APEI)
- Off-Chip

Nios-32
- MStep enabled
- 256 registers
- Shift x 7
- Reset (0x0000)
- VetBase (0x40000)

PBM
- Chip Select
- Data In/Out
- Interrupt Control

User Switch PIO
- Momentary Contact x 4
- LED PIO (bidir)
- 7-seg LED

DIP 8
- LED x 2
- 7-seg LED

PIO-32 (bidir, int)

Expansion Connectors

Address and Data
Buses are Registered
at the I/O Pins

ROM (with monitor)
- UART
- Timer

Level Shifter

© 2000
Reference Design

Factory H/W Image: (1C0000 - 1FFFFFF)
User H/W Image: (180000 - 1BFFFFF)
User Code Space: (100000 - 17FFFFF)

Memory Map

- FLASH: (100000 - 1FFFFF) - 1MByte
- SRAM: (40000 - 7FFFF) - 256KBytes

Vector Table - 40000

Interrupts

- PIO: LCD (0480 - 048C)
- PIO: Switches (0470)
- PIO: LED (0460 - 046C)
- Timer: (0440 - 0458)
- PIO: 7-segment LED (0420)
- UART: (0400 - 0414)
- ROM: (0000 - 03FF)

Reset Address - 0000

Data Width

31 . . . . . . . . . . . . . 0

ALTERA®
Nios Hardware Configuration Process

- Single Image Flash
  - APEX Hardware Configuration
  - User Software

- APEX Configured from Flash
  - 7064 Configures APEX from Flash
  - During Configuration, 7064:
    - Loads User APEX Image
    - If Fail, Loads Factory APEX Image
Nios Processor Boot Process

- Boots from “Reset” Address
  - Default Reset Address 0x0000
    - On-Chip ROM
    - 512 16-Bit Instructions
    - Contains GERMS Monitor

- During Boot
  - Monitor Checks User Code Space
    - Runs Code Found at Address 140000
Nios Software Development Utilities

- BASH: Standard UNIX-like Command Line Shell

- Nios Development Utilities
  - nios-build: Compile, Assemble & Link
  - nios-run: Download Executable & Run
  - srec2flash: Create Flash-bootable Code

- Generic UNIX Utilities Such As:
  - Concatenate: cat
  - Make: make
  - Profile: gprof
Hello World

- Sends “Hello world” Message Out Serial Port
- Open Sample C Program (hello.c)
  - Open **Nios Workshop Folder** on Desktop
  - Double-click **hello.c**
- Compile Sample Program
  - Double-click **BASH**
  - Type: **nios-build hello.c**
  - **Shortcut: nb hello.c**
- Download to Development Board
  - Type: **nios-run hello.srec**
  - **Shortcut: nr hello.srec**
Creating an SOPC Hardware Design
Run Quartus

- Default Design: my_first_nios

32-bit Nios System:

- Targeted for Evaluation/Nios Kit development board
- System includes:
  * 32-bit Nios CPU core
  * Serial Port (115200 baud, N81)
  * Internal ROM with "%EEROM" boot monitor program
  * Interfaces to DIOetty 32-bit non-external RAM
  * Interfaces to Digital external Flash memory
  * Parallel I/O ports connect to LEDs, switches, etc.
Adding a Nios Processor

- Double-Click the Schematic
- Choose: MegaWizard Plug-In Manager...
- Create New Custom Megafunction Variation
Configure a Nios Processor

- File Type: Verilog HDL
- Output File: nios

Create “nios_cpu”...
Configure a Nios Processor

- ALU & Data Bus
  - 16-Bit
  - 32-Bit
Configure a Nios Processor

Address Bus Width
Configure a Nios Processor

- General-Purpose Register File
  - 128, 256, or 512
  - Uses Embedded System Blocks

<table>
<thead>
<tr>
<th>Reg</th>
<th>Nios-32</th>
<th>Nios-16</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>256</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>512</td>
<td>16</td>
<td>8</td>
</tr>
</tbody>
</table>
Configure a Nios Processor

- Internal Shifter Speed
- Multiply-Step Unit
Configure a Nios Processor

■ Create the Nios Processor

```
Creating Nios Core "nios_cpu"
----- BEGIN DEBUG INFO -----------------------------
"D:\QUARTUS\libraries\megafungions\nios\bin\perl"
-ID:\QUARTUS\libraries\megafungions\nios\bin\-
-ID:\QUARTUS\libraries\megafungions\nios\bin\lib
D:\QUARTUS\libraries\megafungions\nios\bin\..\jniowizard\ak_nios.pl
name=nios_cpu,project_dir=.,bits=32,num_regs=256,shift_size=7,mstep=yes,wvalid
----- END DEBUG INFO -----------------------------
Pass 1...
D:\QUARTUS\ies\megafungions\nios\bin\vpp\process_sdf.vpp... done
D:\QUARTUS\ies\nios\jniowizard\vpp_source\cpu_interface.vpp... done
Pass 2...
D:\QUARTUS\ies\megafungions\nios\bin\vpp\process_sdf.vpp... done
D:\QUARTUS\ies\nios\jniowizard\vpp_source\cpu_interface.vpp...
```

Use New CPU

- Click **Next**
Nios System Builder

- Displays System Configuration
- Add Peripheral Content
On-Chip ROM
On-Chip ROM

- Uses Embedded System Blocks (ESB)
  - Up to 2,048 16-Bit Values/Instructions
  - Zero-Wait-State Access

- Loaded With Boot Code
  - GERMS Monitor Code Included (Source and Executable)

- Boot Code Development Utility (srec2mif)
  - Converts srec (Compiler Output) to mif (APEX ROM Format)
Adding On-Chip ROM

- Peripheral Name: boot_rom
- Peripheral Type: On-Chip ROM
Configuring On-Chip ROM

- **ROM Size**
  - 128 to 2,048 Instructions (16-Bit)

- **ROM Input File**
  - SREC or MIF
On-Chip ROM Alignment & Address

- Alignment: Half-Word
- Base Address: 0x0
Memory Interface
Memory Interface

- External/Internal Bus
- Data/Address Bus Width
- Chip-Select Signals
- Wait States
- Interrupt Request
Adding a Memory Interface: SRAM

- Peripheral Name: ext_ram
- Peripheral Type: Memory Interface
Configuring a Memory Interface: SRAM

Peripheral Description (Optional)

![Memory Interface Configuration](image-url)
Configuring a Memory Interface: SRAM

- **On-Chip**
  - Dedicated Buses & Control Signals

- **Off-Chip**
  - Shared Buses & Control Signals
Configuring a Memory Interface: SRAM

- Data Bus Width: 32 (Word Alignment)
- Address Bus Width: 16
Configuring a Memory Interface: SRAM

- Registered Chip Select
  - Uses Fast I/O
  - Zero-Wait-State Sustained Access to 50 MHz
Configuring a Memory Interface: SRAM

- Chip Selects
  - Two Identical (Bank Operation)

Duplicate (Bank) Chip-Select Outputs

Wide external memory devices are often constructed from banks of narrower devices (e.g., two 16-bit SRAM chips in parallel to make a 32-bit memory). When interfacing to a bank of narrow devices, it is sometimes necessary to produce multiple, logically-identical chip-select signals.

The system module will produce
- One Single
- Two identical

registered chip-select outputs.
Configuring a Memory Interface: SRAM

- Fixed Wait States
  - Read = 0
  - Write = 0

- No Peripheral Interrupt
Memory Interface: Alignment & Address

- Alignment: Word
- Base Address: 0x40000
Adding a Memory Interface: FLASH

- Peripheral Name: ext_flash
- Peripheral Type: Memory Interface
Configuring a Memory Interface: FLASH

- Peripheral Description (Optional)

You have requested a memory-mapped interface to a device outside the system module. For example:

- Off-chip memory
- Memory-mapped user-logic devices
- 3rd Party peripheral devices

This wizard adds I/O ports (and associated bus logic) to the system module. These ports are customized for attaching your peripheral to the Nios bus.

Peripheral Type or Description (optional)

1MByte Flash

[Image: Megawizard Plug-In Manager - [Page 1 of 7]]
Configuring a Memory Interface: FLASH

- Off-Chip
  - Shared Buses & Control Signals
Configuring a Memory Interface: FLASH

- Data Bus Width: 16 (Half-word Alignment)
- Address Bus Width: 19
Configuring a Memory Interface: FLASH

- Registered Chip Select
  - Uses Fast I/O
  - Zero-Wait-State Sustained Access to 50 MHz
Configuring a Memory Interface: FLASH

- Wait States
  - Read = 8
  - Write = 8
- No Peripheral Interrupt
Memory Interface Alignment: FLASH

- Alignment: Half-Word
Memory Interface Address: FLASH

- Base Address: 0x100000
UART Peripheral

- RS-232 Asynchronous Receiver / Transmitter
  - Start/Stop/Parity/Baud Rate Set at Compile-Time
  - Double-Buffered on TxD & RxD

- External Level-Shifting Buffer Required to Comply with RS-232 Voltage Signaling Specifications
# UART Register Map

| A2 A0 | Register Name | Byte 15 | Byte 14 | Byte 13 | Byte 12 | Byte 11 | Byte 10 | Byte 9 | Byte 8 | Byte 7 | Byte 6 | Byte 5 | Byte 4 | Byte 3 | Byte 2 | Byte 1 | Byte 0 | Read-only Value |
|-------|---------------|---------|---------|---------|---------|---------|---------|-------|-------|-------|-------|-------|-------|-------|-------|-------|----------------|
| 0     | Rx Data       |         |         |         |         |         |         |       |       |       |       |       |       |       |       |       |       |                |
| 1     | Tx Data       |         |         |         |         |         |         |       |       |       |       |       |       |       |       |       |       |                |
| 2     | Status        |         |         | E†      | RRDY    | TRDY    | TMT     | TOE†  | ROE†  | BRK†  | FE†   | PE†   |       |       |       |       |       |                |
| 3     | Control       |         |         | TBRK    | IE      | RRDY    | ITRDY   | ITMT  | iTOE  | iROE  | iBRK  | iFE   | iPE   |       |       |       |       |                |
| 4     | Divisor       |         |         |         |         |         |         |       |       |       |       |       |       |       |       |       |       | Baud Rate Divisor (optional) |

- **Read-only value.**
- **Host-written control value.** Can be read-back at any time.
- **Write-event register.** A write-operation to this address causes an event in the device.

## Register Addresses on Word Boundaries (32-Bit)

- **Example: UART Base Address 0x0400**

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Byte 3</th>
<th>Byte 2</th>
<th>Byte 1</th>
<th>Byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RxData</td>
<td>0x0400</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>XXXXXXXXX</td>
</tr>
<tr>
<td>TxData</td>
<td>0x0404</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>XXXXXXXXX</td>
</tr>
<tr>
<td>Status</td>
<td>0x0408</td>
<td>00000000</td>
<td>00000000</td>
<td>0000000X</td>
<td>XXXXXXXXX</td>
</tr>
<tr>
<td>Control</td>
<td>0x040C</td>
<td>00000000</td>
<td>00000000</td>
<td>000000XX</td>
<td>XXXXXXXXX</td>
</tr>
<tr>
<td>Divisor</td>
<td>0x0410</td>
<td>00000000</td>
<td>00000000</td>
<td>XXXXXXXXX</td>
<td></td>
</tr>
</tbody>
</table>
Adding a UART Peripheral

- Peripheral Name: uart1
- Peripheral Type: UART (Serial Port)
Configuring a UART

- Clock Frequency
- Baud Rate
- Baud Rate Divisor Register
- Parity
- Data Bits
- Stop Bits
UART Alignment, Address & IRQ

- Alignment: Word
- Base Address: 0x400
- IRQ: 16
Timer Peripheral
Timer Peripheral

- 32-Bit Interval Timer
  - Counts Down to 0 from Preset Value
  - 16-Bit Peripheral, Compatible with Nios-16
  - Two 16-Bit Registers

- Polled Operation
  - Periodically Read Coherent “Snapshot” of Counter Value

- Interrupt Operation
  - Generate a Single (Maskable) Interrupt on Time Out

- Runs Off Single Master Clock Input (Clk)
Timer Register Map

Register Addresses on Word Boundaries (32-Bit)
- Example: Timer Base Address 0x0440

| A2..A0 | Register Name | 13 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 0 | 5 | 4 | 3 | 2 | 1 | 0 |
|--------|---------------|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|
| 0      | Status        | 0  | 0  | 0  | 0  | 0  | 0  | 0 | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | XX |
| 1      | Control       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |
| 2      | Period(L)     |    |    |    |    |    |    |    | Run | Stop | Start | Cont |    |    |    |    |    |    |   |
| 3      | Period(H)     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |
| 4      | Snap(L)*      |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |
| 5      | Snap(H)*      |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |

- Read-only value.
- Host-written control value. Can be read-back at any time.
- Write-event register. A write-operation to this address causes an event in the device.

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Byte 3</th>
<th>Byte 2</th>
<th>Byte 1</th>
<th>Byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>0x0440</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>000000XX</td>
</tr>
<tr>
<td>Control</td>
<td>0x0444</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>0000XXXX</td>
</tr>
<tr>
<td>Period(L)</td>
<td>0x0448</td>
<td>00000000</td>
<td>00000000</td>
<td>XXXXXXXX</td>
<td>XXXXXXXX</td>
</tr>
<tr>
<td>Period(H)</td>
<td>0x044C</td>
<td>00000000</td>
<td>00000000</td>
<td>XXXXXXXX</td>
<td>XXXXXXXX</td>
</tr>
<tr>
<td>Snap(L)</td>
<td>0x0450</td>
<td>00000000</td>
<td>00000000</td>
<td>XXXXXXXX</td>
<td>XXXXXXXX</td>
</tr>
<tr>
<td>Snap(H)</td>
<td>0x0454</td>
<td>00000000</td>
<td>00000000</td>
<td>XXXXXXXX</td>
<td>XXXXXXXX</td>
</tr>
</tbody>
</table>

© 2000
Adding a Timer Peripheral

- Peripheral Name: timer1
- Peripheral Type: Interval Timer
Configuring a Timer

- No Parameters Required
Timer Alignment, Address & IRQ

- Alignment: Word
- Base Address: 0x440
- IRQ Priority: 15
Parallel I/O (PIO)
Peripheral
PIO Peripheral

- 1 to 32-Bit Parallel Input/Output
  - 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
  - Bit-maskable
  - IRQ Source
    • Input Level
    • Edge Detection Register
# PIO Register Map

<table>
<thead>
<tr>
<th>(A1,A0)</th>
<th>Register Name</th>
<th>Variable Size 1 .. 32 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Data-in Data-out</td>
<td>Data value currently on PIO inputs (read only).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New value to drive on PIO outputs (write only).</td>
</tr>
<tr>
<td>1</td>
<td>DataDir</td>
<td>Data Direction (optional) Individual direction control for each port bit. 1=Out, 0=In.</td>
</tr>
<tr>
<td>2</td>
<td>Int Mask</td>
<td>Interrupt Mask (Optional) Per-bit irq enable/disable.</td>
</tr>
<tr>
<td>3</td>
<td>Edge Capture?</td>
<td>Edge Capture (Optional) Per-bit synchronous edge detect-and-hold.</td>
</tr>
</tbody>
</table>

- **Read-only register.**
- **Host-written control value.** Can be read-back at any time.
- **Write-event register.** A write operation to this address causes an event in the device.

*A write-operation to the Data-out register changes the value on the PIO’s output pins, if any.*

*A write-operation to the Edge Capture register clears all bits in the register to 0.*

## Register Addresses on Word Boundaries (32-bit)

- **Example: PIO Base Address 0x0480**

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data-In/Out</td>
<td>0x0480</td>
</tr>
<tr>
<td>DataDir</td>
<td>0x0484</td>
</tr>
<tr>
<td>Int Mask</td>
<td>0x0488</td>
</tr>
</tbody>
</table>
Adding a PIO Peripheral: Buttons

- Peripheral Name: button_pio
- Peripheral Type: Parallel I/O
Configuring a PIO: Buttons

- PIO Bits: 12
- Type of Pins: Input Only
Configuring a PIO: Buttons

- Capture Any Edge
- Interrupt on Edge Condition
### PIO Alignment, Address & IRQ: Buttons

- **Alignment:** Word
- **Base Address:** 0x470
- **IRQ Priority:** 19

#### Diagram

![Diagram of Megawizard Plug-In Manager - Nios System Builder](image)

#### Table

<table>
<thead>
<tr>
<th>Use</th>
<th>Name</th>
<th>Type</th>
<th>Alignment</th>
<th>Base Addr</th>
<th>End Addr</th>
<th>IRQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>boot_rom</td>
<td>archchip_rom</td>
<td>harword</td>
<td></td>
<td>0x0</td>
<td>0x0003FF</td>
<td>N/A</td>
</tr>
<tr>
<td>ext_ram</td>
<td>Ext SRAM x 2</td>
<td>word</td>
<td></td>
<td>0x40000</td>
<td>0x07FFFF</td>
<td>N/A</td>
</tr>
<tr>
<td>ext_flash</td>
<td>1MByte Flash</td>
<td>harword</td>
<td></td>
<td>0x100000</td>
<td>0x1FFFFF</td>
<td>N/A</td>
</tr>
<tr>
<td>uart1</td>
<td>uart</td>
<td>word</td>
<td></td>
<td>0x400</td>
<td>0x00041F</td>
<td>16</td>
</tr>
<tr>
<td>timer1</td>
<td>timer</td>
<td>word</td>
<td></td>
<td>0x440</td>
<td>0x00045F</td>
<td>15</td>
</tr>
<tr>
<td>button_pio</td>
<td>pio</td>
<td>word</td>
<td></td>
<td>0x470</td>
<td>0x00047F</td>
<td>19</td>
</tr>
</tbody>
</table>

*Double-click on a peripheral name to edit. Select a cell to view errors / edit contents.*
Adding a PIO Peripheral: LED

- Peripheral Name: led_pio
- Peripheral Type: Parallel I/O
Configuring a PIO: LED

- PIO Bits: 2
- Type of Pins: Tri-state
Configuring a PIO: LED

- No Edge Capture
- No Interrupt
PIO Alignment, Address: LED

- Alignment: Word
- Base Address: 0x460
Adding a PIO Peripheral: 7-Segment LED

- Peripheral Name: seven_seg_pio
- Peripheral Type: Parallel I/O
Configuring a PIO: 7-Segment LED

- PIO Bits: 16
- Type of Pins: Output Only
PIO Alignment, Address: 7-Segment LED

- Alignment: Word
- Base Address: 0x420
Adding a PIO Peripheral: LCD

- Peripheral Name: lcd_pio
- Peripheral Type: Parallel I/O
Configuring a PIO: LCD

- PIO Bits: 11
- Type of Pins: Tri-State
Configuring a PIO: LCD

- No Edge Capture
- No Interrupt
PIO Alignment, Address: LCD

- Alignment: Word
- Base Address: 0x480
System Configuration
System Configuration

- Reset Address: Location of Boot Code
- Exception Vector Table
- Main Program Memory: Software Located Here

![Image showing System Configuration settings in a software interface.](Image)
System Configuration

- Synthesis Tool Selection

![Synthesis Tool Selection](image_url)

Synthesize the Nios System module

The nios module must be synthesized before it can be included in your design. This wizard can launch any of the following synthesis tools and synthesize nios automatically. Please select your preferred tool, which must be properly installed and licensed on your system.

The tool you select will produce the file: nios.edf as output.

- Leonardo Spectrum
- FPGA Express
- None. I will synthesize the design later.
System Configuration

- Generate Design
Other MegaWizard Plug-Ins
Memory-Mapped Peripheral

- Adds Port for Connecting Other IP or External Devices
- Parameters
  - Peripheral Description
  - On-Chip / Off-Chip Port
  - Data Bus Width
  - Address Bus Width
  - Wait States
  - Interrupt Request
On-Chip RAM

- Uses Embedded System Blocks (ESB)
  - Zero Wait State Access
- RAM Width
  - 8, 16, or 32 Bits
- RAM Depth
  - 128 to 4,096 Addresses
Completing the Design
Select Nios Symbol

- Click OK
Place on Pre-Defined Pins

32-bit Nios system

- Targeted for Xilinx® Nios® bit development board

System Includes:
- 32-bit Nios® CPU core
- Serial Port (155,000 baud, NRZ)
- Internal RAM with resident *SERNS* boot monitor program
- Interface to 336-byte off-chip external RAM
- Interface to 32-byte off-chip external Flash
- Parallel I/O ports connected to LEDs, switches, etc.

Note: Development board requires:
- That unused I/O pins be left as hi-z
- Do not use unused I/O pins

What if there's a Pin? Some pins can be used for:
- D/A converter
- Timer/counters
- Watchdog

© 2000
Specify EDA Tools

- Select Synthesis Tool BEFORE Place & Route
  - LeonardoSpectrum
- Project > EDA Tool Settings
Unused Pins

- Assign Unused Pins
  - As Inputs, Tri-Stated
  - Necessary for Dev Board

- Processing >
  Compiler Settings >
  Chips & Devices Tab >
  Device & Pin Options >
  Unused Pins Tab >
  Inputs, tri-stated
Compile Design

- Processing > Start Compilation
Behavioral Testbench
Nios System Testbench

- Nios Testbench Generated Automatically

- Testbench Stimulus Programmable via C
  - Executes Code from Simulation RAM
  - RAM Contents Contain Executable Code
Nios System Testbench

Character Stream:

"GO" (Run Code)
At 40100
Nios System Testbench

```c
printf "Some text" .
```
Nios System Testbench

![Diagram of Nios System Testbench](image)

- **Nios**
- **Timer**
- **UART**
- **PIO**
- **ROM**
- **RAM**
- **External RAM**
  - Byte Lanes 0 - 3
- **C Code** (srec)
  - printf
    - "Some text"
- **srec2dat**

Some text
Using the Testbench

- Character Stream File
- External RAM Lane Files
Character Stream

// Character_Stream.dat
//
// In simulation, the Nios UART will read this file as a source
// of simulated character-input. These characters will "arrive"
// one-at-a-time through the UART.
//
// This is a standard Verilog memory-initialization file.
//
// This example file "types" the string:
//
//     "<LF> G 40100 <CR> <LF>"
//
// Which is a command to the monitor telling it to start executing
// code at address 40100 (somewhere in the external RAM).
//
// For this to work properly, the external RAMs will have had to be
// initialized through their own .dat-files.
//
// Alternatively, this file (right here!) can "type" an entire S-record
// into the UART, then run it. That's a mighty slow way to do things,
// but I've done it before.

00000
OA 47 34 30 31 30 30 OA OD OA OD
00
srec2dat

- Converts Executable File to RAM Data Files
- Example:
  srec2dat hello.srec
- Generates:
  external_ram_lane_0.dat
  external_ram_lane_1.dat
  external_ram_lane_2.dat
  external_ram_lane_3.dat
Software Development with GNUPro for Nios
Monitor Program Runs from On-Chip ROM
Communicates to Host Via Serial Port
Basic Development Facilities:
  – Download Executable Code to Board
  – Examine/Modify Memory
  – Run Programs
GERMS Monitor Commands

■ Usage
  – <Command><Parameter>

■ Example
  – G40100

■ Commands
  – G Go (Run Program)
  – E Erase Flash
  – R Relocate Next Download
  – M Memory Set & Dump
  – S Transfer S-record to Memory
  – : Transfer I-hex Records to Memory

■ No Backspace or Delete Key Support
  – Press <Esc> to Abort Current Command then Retype
Nios-Run: Terminal Mode

- Switch to BASH
- Type:
  
  `nr -t`

![Terminal Mode](image)
Using GERMS

- Examine Interrupt Vector Table
  m40000-40100<cr>

- Write Data to Seven-Segment LED
  m420:DF4C<cr>

- Fill Memory Range with Data
  m50000-50100:aa55<cr>
  <cr>

- Run Program in SRAM
  g40100<cr>
Creating a “Flash-Bootable” Image
Download APEX Design to Flash

- User APEX Image
  - my_first_nios
- Erase User APEX Segment
  - nr -t
e180000
e190000
e1a0000
e1b0000
- Relocate Next Download to User APEX Segment
  - r180000
- Exit Terminal Mode
  - CTRL+C
Download APEX Design to Flash

- Send hexout File
  
  nr my_first_nios.hexout

- Check Factory Design
  
  - View lcd_pio address: m480

- Reconfigure APEX with User Hardware Design
  
  - Press Reset Button (RED)
  - View lcd_pio address: m480
  - Exit Terminal Mode: CTRL+C
Application Software

- Example Program (lcd_demo.c)
  - Sends Characters to lcd_pio Peripheral

- Double-Click lcd_demo.c

- Edit Text:
  - “Your message goes here”

- Save File
Application Software

- Compile Program
  `nb lcd_demo.c`

- Download & Run from SRAM First
  `nr lcd_demo.srec`

- Verify that Your Message Appears

- Stop the Program
  - Press the Clear Button (Green)
  - Exit Terminal Mode: CTRL+C
Application Software

- Boot Program from Flash
  - Copy Executable File (srec) to Flash
  - Code Stored at Address 140000 is Executed Automatically

- Use Flash for Program Storage
  - Flash is Slow
  - Difficult Writing to Flash

- Execute Code From SRAM
  - srec2flash Utility
  - Adds ‘Copy’ Code to srec File
    - Transfers Program to SRAM
    - Executes Program from SRAM
Application Software

- Prepare for Flash
  srec2flash lcd_demo.srec

- Download to Flash
  nr  lcd_demo.flash

- Cycle Power
Recovering When Things Go Wrong

- Software: Return to the “Monitor”
  - Ignore Code at Address 140000
    - Press & Hold SW4
    - Press & Release Clear
Recovering When Things Go Wrong

- Hardware: Load Factory APEX Configuration
  - Ignore User APEX Configuration
    - Short Jumper JP2
    - Press Reset
Nios Development Kit
Excalibur Development Kit

- Altera 32-Bit RISC CPU
- Peripherals
- Development Board
- Development Software
- Download Cable
- One Year of Updates

Excalibur Development Kit Featuring Nios
$995
Nios Release 1.1: December 2000

- ACEX™ 1K & FLEX® 10KE Support
- Peripherals
  - I²C
  - SPI
  - JSRT
  - Watchdog Timer
- Fast Multiply (16 x 16 ➔ 32, 2 clocks)
- Additional Synthesis Tool Support
  - FPGA Express
  - Synplify
- Dynamic Bus Sizing
- Monitor Builder
Coming in 2001

- SDRAM Support *
  - Main Memory
  - Data Storage

- Ethernet Support *
  - Internet Appliance

- On-Chip Debug Core
  - Hardware Breakpoints
  - Bus Monitor

* Sold Separately
Excalibur Benefits

- Eliminates Barriers to System-on-Chip Designs
  - Low Risk
  - Fast Development Cycle
  - Low Initial Cost

- Customizable Microcontroller
  - High Level of Integration
  - Peripheral Content Tailored to Application
  - Hardware / Software Optimization

- Complete Development Kit
  - “Boots” Out of Box
  - Software Development Can Begin Immediately
  - Additional Development Boards Available
Soft Core Solution: Nios

- Altera-Designed, 32-Bit RISC Processor
  - 16-bit Data Path Configuration
  - Provides Highest Efficiency & Performance in APEX While Guaranteeing Lowest Cost to Customers

- Multiprocessor Implementation

- $0 License
  - No Royalties or License Fees

- Nios Price & Performance Migrates with PLD
Hard Core Solutions: ARM & MIPS

- High-Performance Processors
  - Up to 200 MIPS
- Included on Chip (Hard Logic)
  - CPU
  - External Memory Interface
  - Cache
  - On-Chip Bus
  - RAM
  - UART
- No Royalties or License Fees
- Targeted for Specific APEX Devices
System on a Programmable Chip

- Excalibur Solutions
  - High Integration
  - Custom Solution
  - Fast Time-to-Market
  - Low Risk
Thank You
for Your Participation!