Designing with the Nios II Processor and SOPC Builder

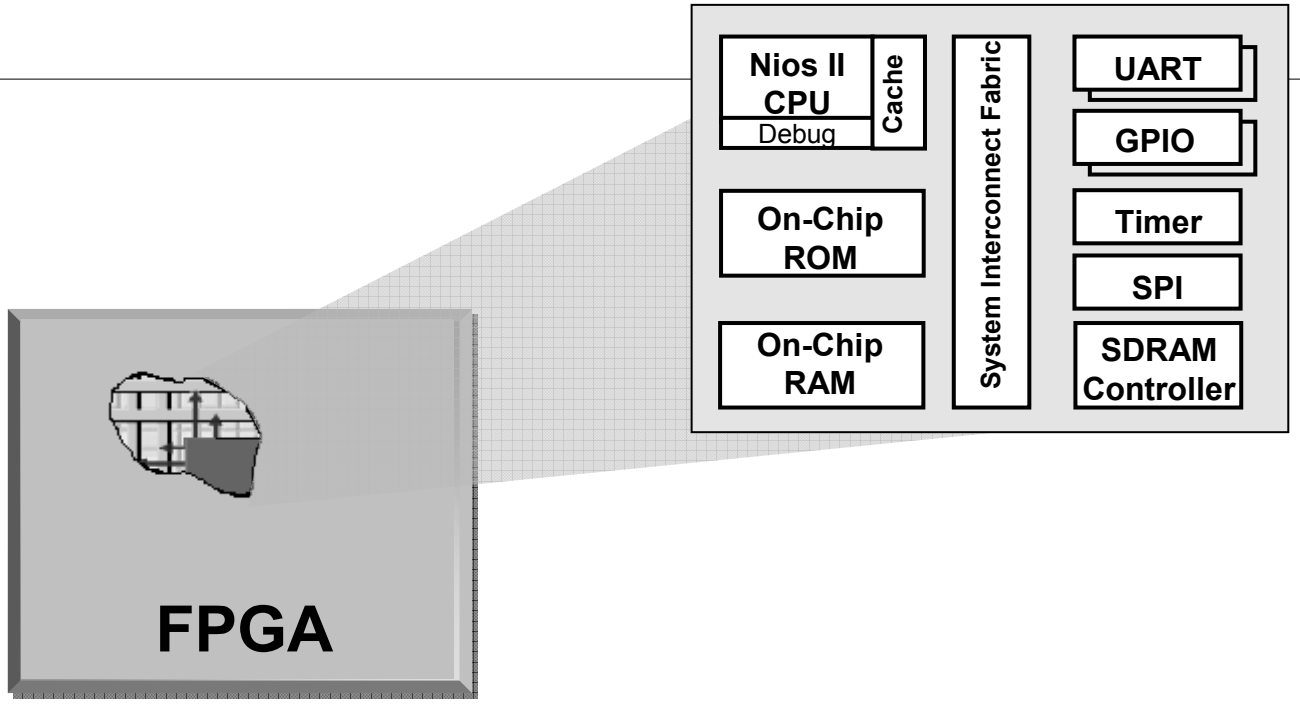
Objectives

- Students will be able to:
  - Describe the Nios® II softcore processor
  - Use the SOPC Builder tool to create complex systems
  - Create and debug software for the Nios II processor
  - Perform an RTL Simulation in the ModelSim® simulator
  - Build custom peripherals
  - Tie in custom peripherals to the system interconnect fabric (SIF) and utilize its multi-mastering capabilities
  - Append a custom instruction to the Nios II instruction set
  - Program the development board
  - Program Flash memory

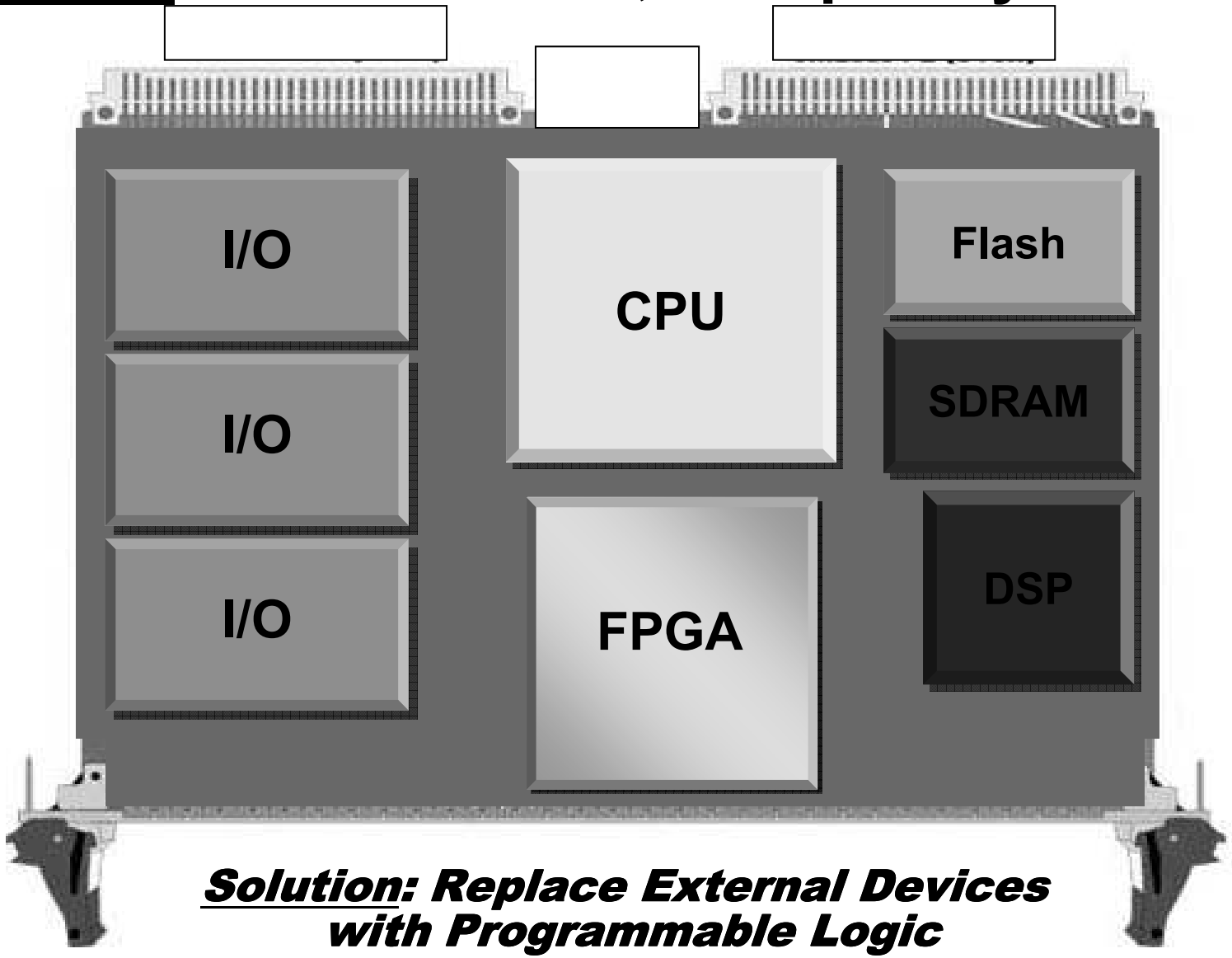
Nios II Processor - Hardware Development

What is the Nios II Processor?

- Second Generation Soft-Core 32 Bit RISC Microprocessor
  - Nios II Processor + all peripherals written in HDL
  - Can be targeted for all Altera FPGAs
  - Synthesis using Quartus® II integrated synthesis engine

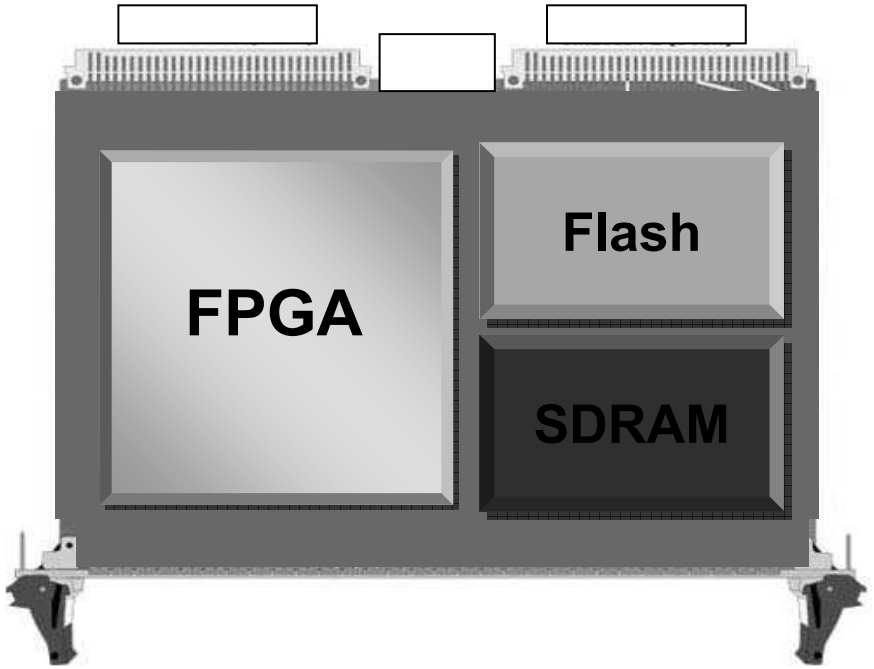


Problem: Reduce Cost, Complexity & Power



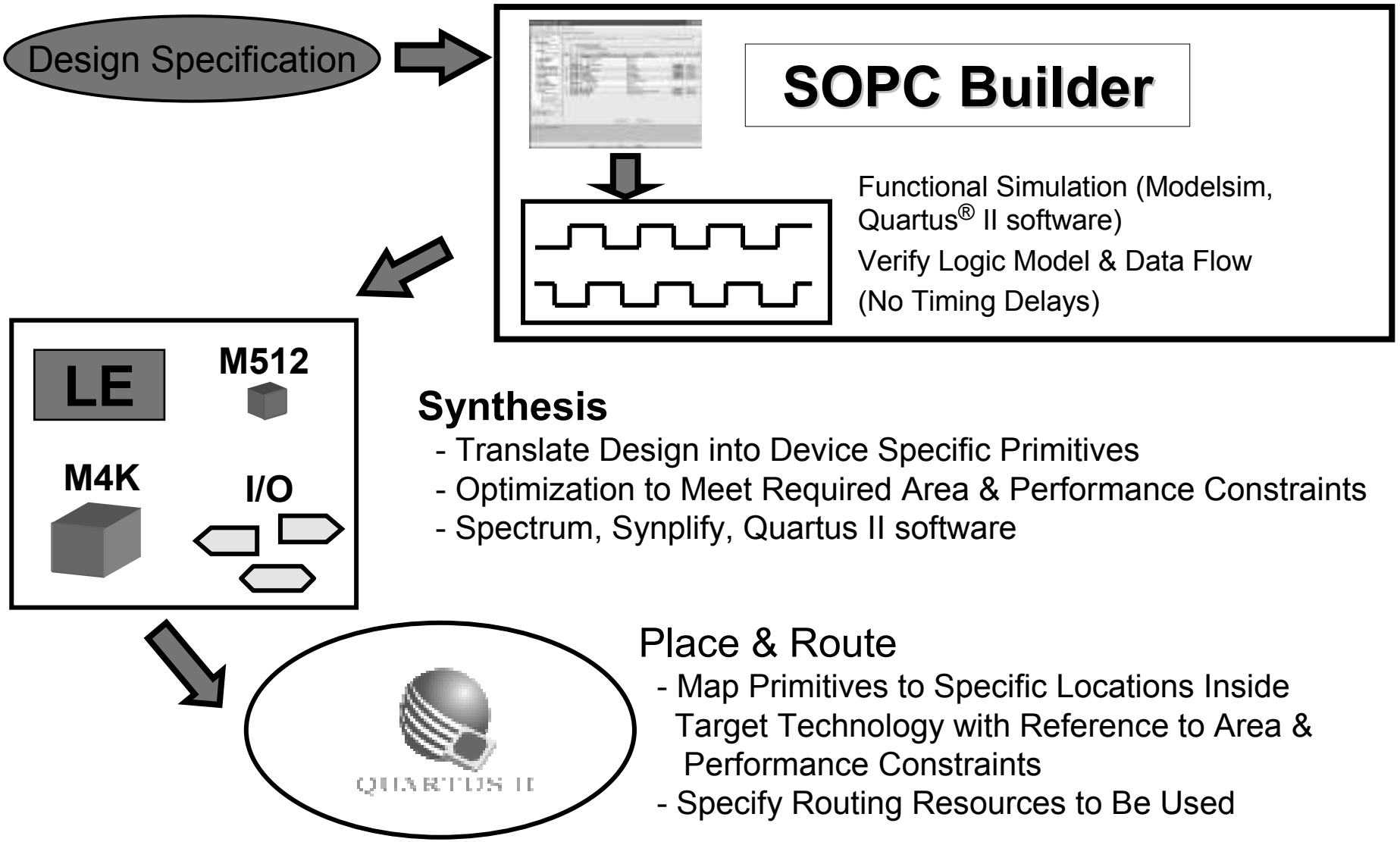
Solution: Replace External Devices with Programmable Logic

System On A Programmable Chip (SOPC)



CPU is a Critical Control Function Required for System-Level Integration

FPGA Hardware Design Flow



Synthesis

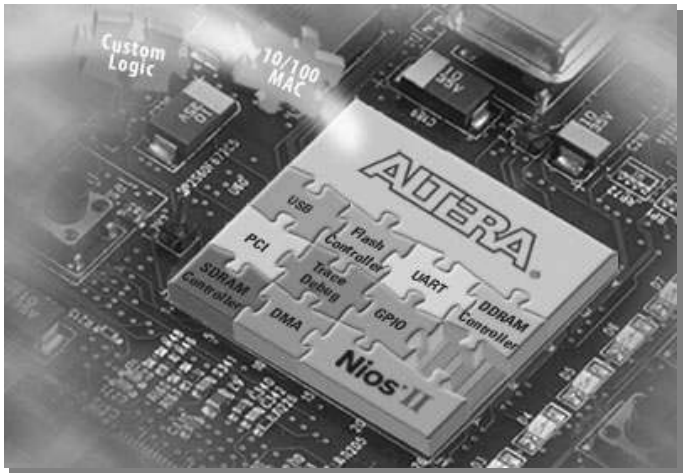
- Translate Design into Device Specific Primitives
- Optimization to Meet Required Area & Performance Constraints
- Spectrum, Synplify, Quartus II software

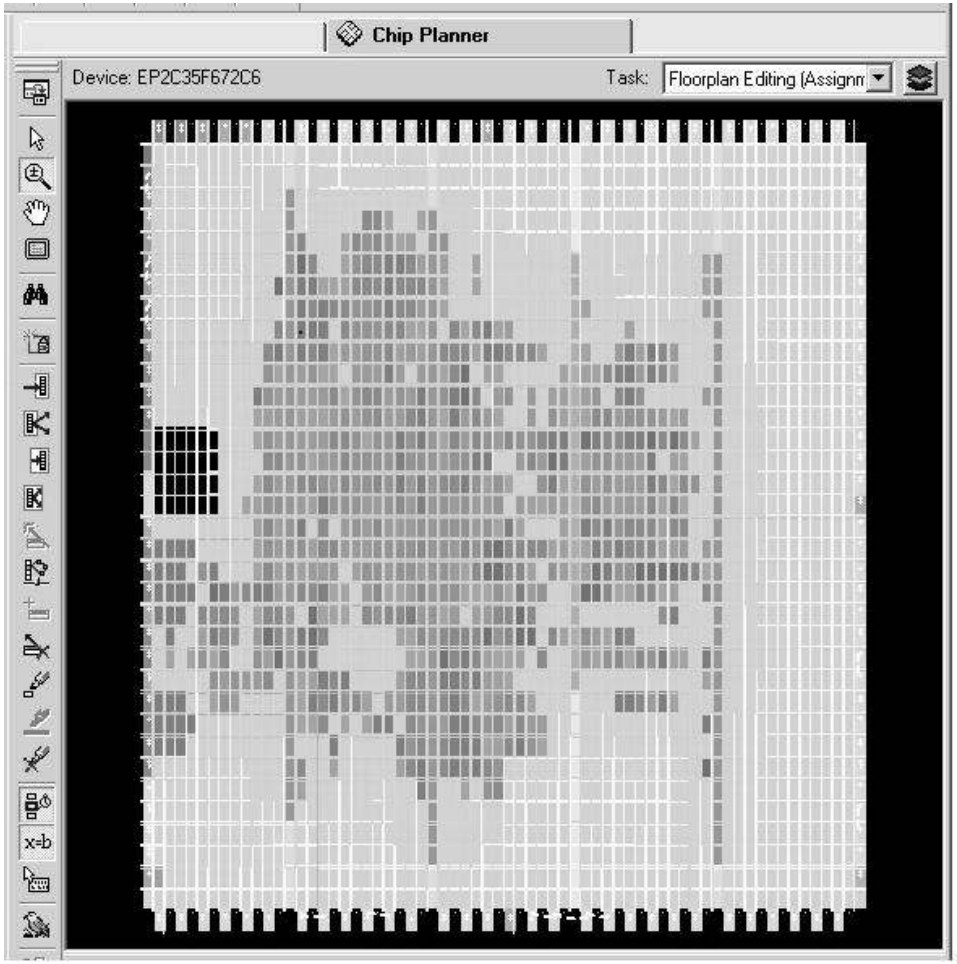
Place & Route

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

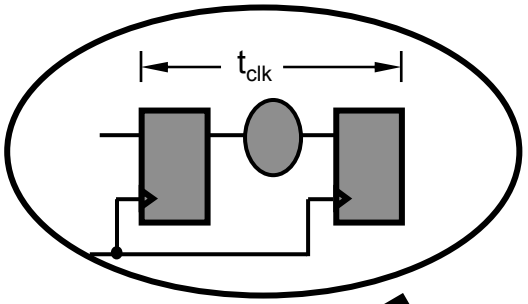
Compiled SOPC System Inside FPGA

- Use Quartus II software Integrated Synthesis and Place and Route engines to implement system in FPGA logic

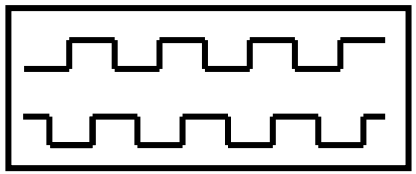




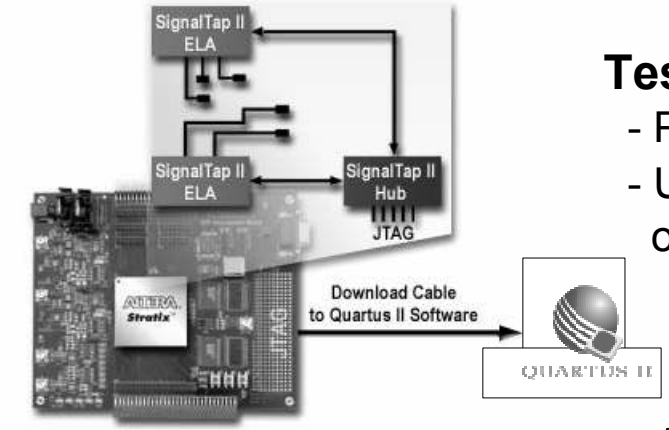
FPGA Hardware Design Flow (cont.)



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

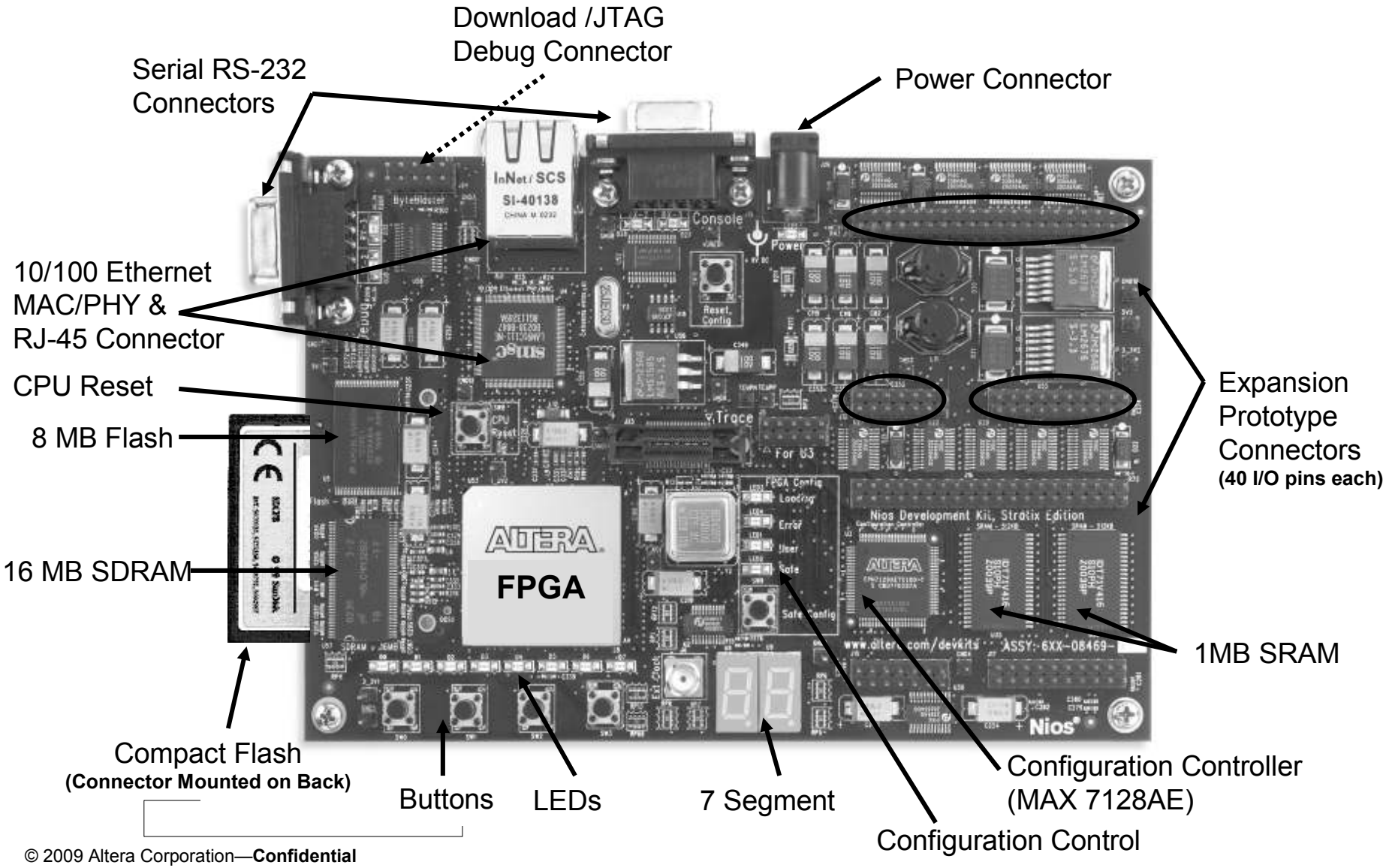


Gate Level Simulation
- Timing Simulation
- Verify Design Will Work in Target Technology



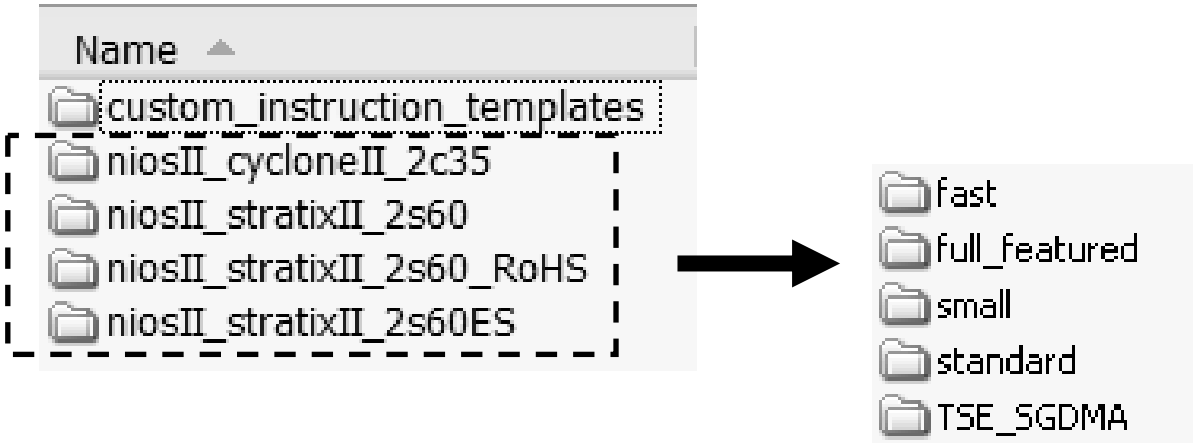
Test FPGA on PC Board
- Program & Test Device on Board
- Use **SignalTap® II** logic analyzer or **Signal Probe** for Debugging
  - Discussed in depth in Advanced Quartus II software class

Development Kits



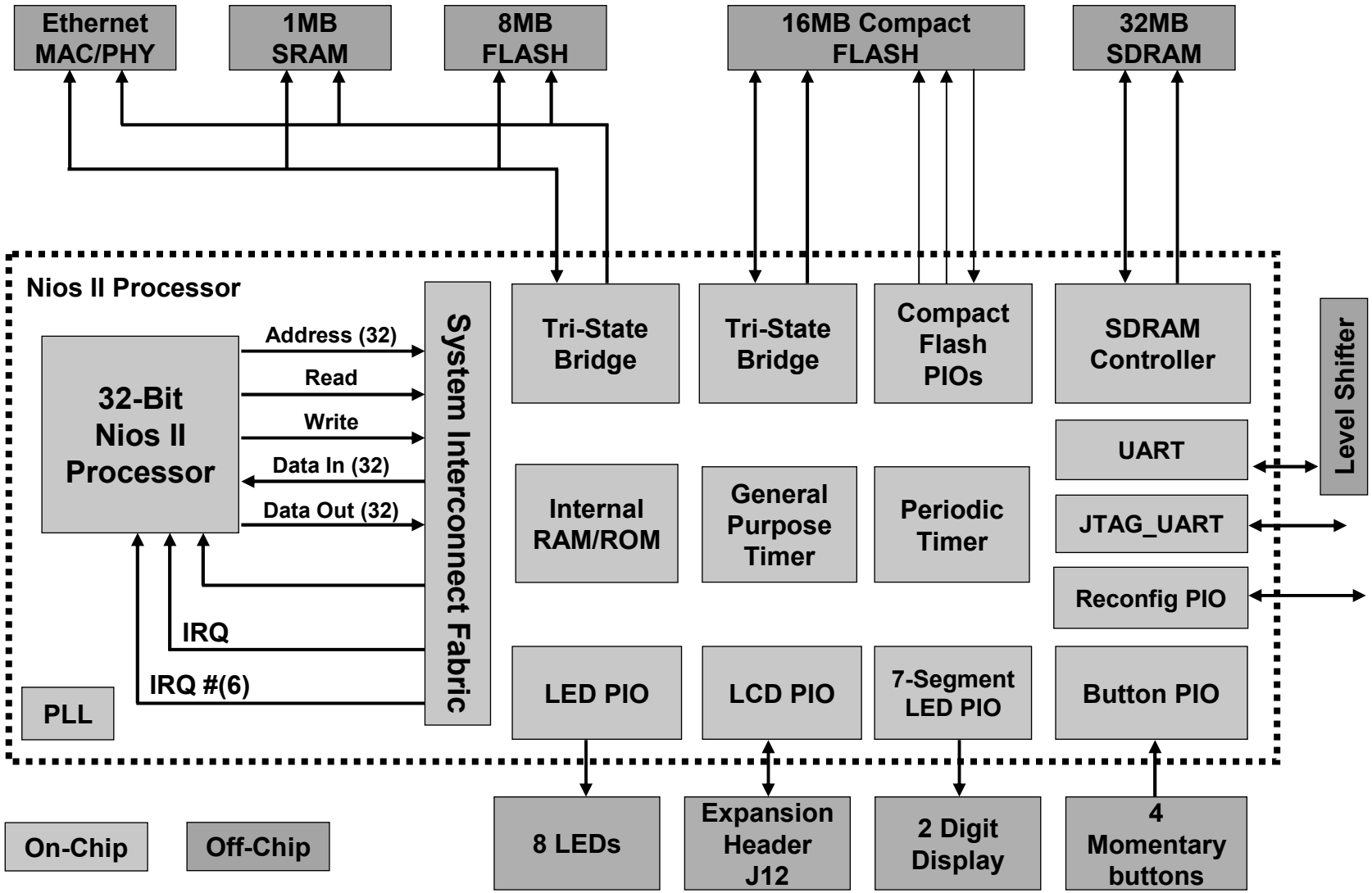
Reference Designs For Dev Kits

- Several reference designs are available
  - See **C:\altera\<ver>\nios2eds\examples\verilog**
  - and **C:\altera\<ver>\nios2eds\examples\vhdl**

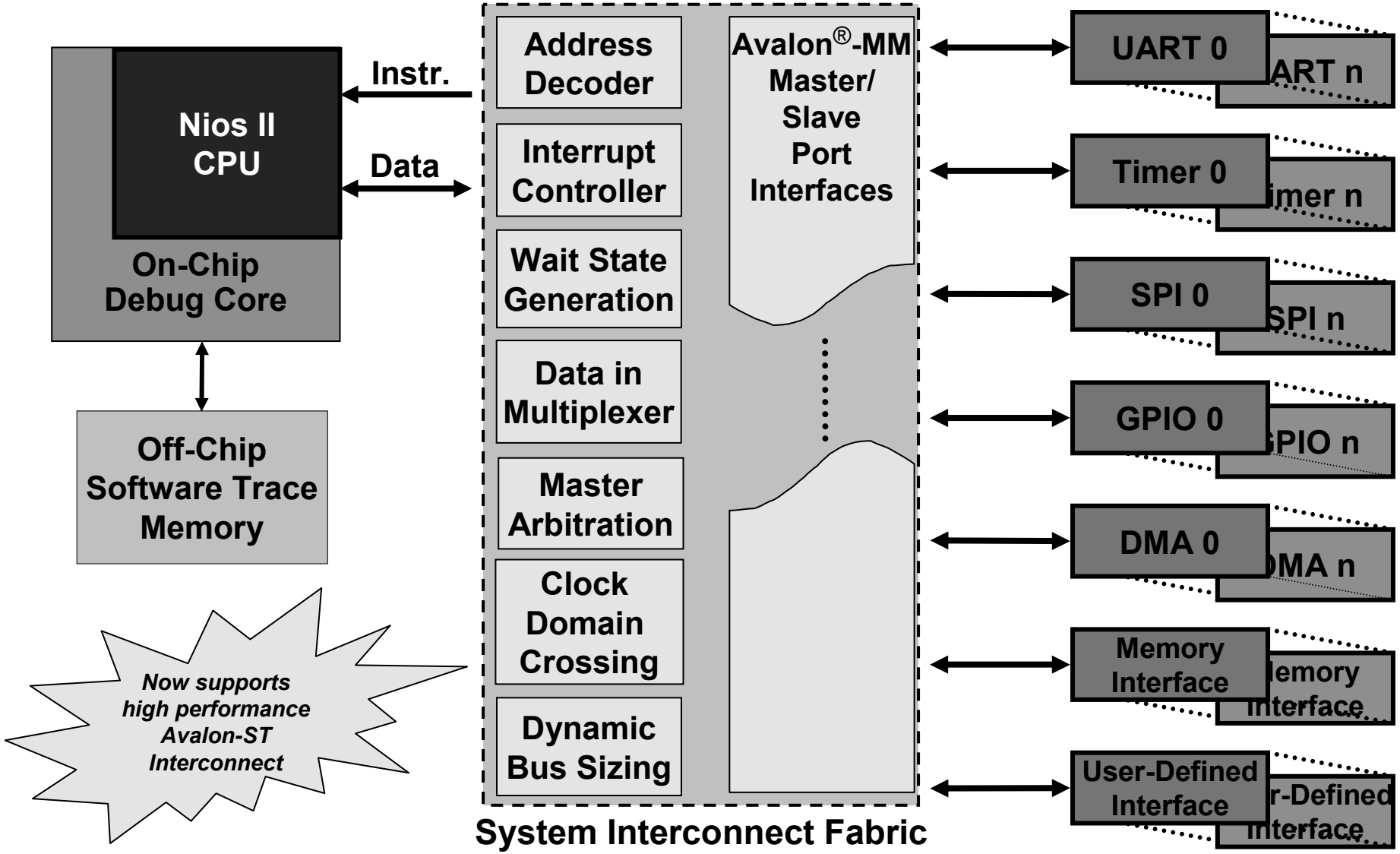


- Can be used *as-is* in final hardware platform or customized for system-specific needs

Standard Reference Design Block Diagram



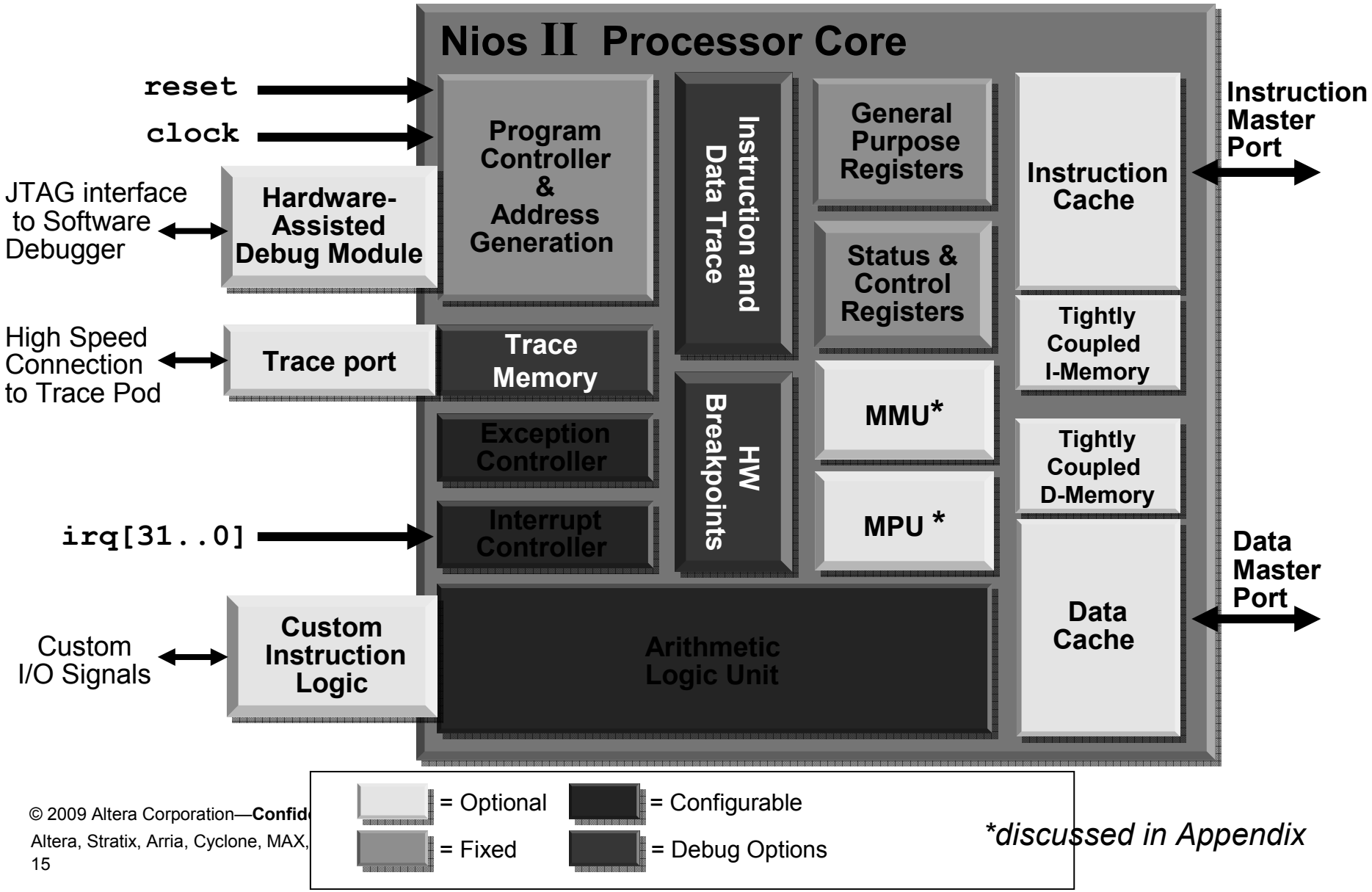
Typical System Architecture



Nios II Processor Architecture

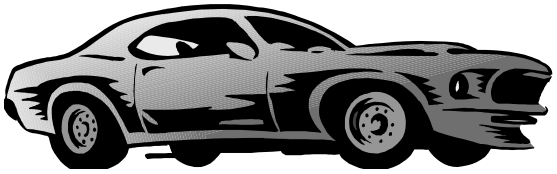
- **Classic Pipelined RISC Machine**
  - 32 General Purpose Registers
  - 3 Instruction Formats
  - 32-Bit Instructions
  - 32-Bit Data Path
  - Flat Register File
  - Separate Instruction and Data Cache (configurable sizes)
  - Tightly-Coupled Memory Options
  - Branch Prediction
  - 32 Prioritized Interrupts
  - On-Chip Hardware (Multiply, Shift, Rotate)
  - Memory Management Unit (MMU)
  - Memory Protection Unit (MPU)
  - Custom Instructions
  - JTAG-Based Hardware Debug Unit

Nios II Processor Block Diagram



Nios II Processor 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

Nios II Processor Comparison Chart

	Nios II /f Fast	Nios II /s Standard	Nios II /e Economy
Pipeline	6 Stage	5 Stage	None
H/W Multiplier & Barrel Shifter	1 Cycle	3 Cycle	Emulated In Software
Branch Prediction	Dynamic	Static	None
Instruction Cache	Configurable	Configurable	None
Data Cache	Configurable	None	None
Logic Requirements (Typical LEs)	1800 w/o MMU 3200 w/ MMU	1200	600
Custom Instructions	Up to 256		

Hardware Multiplier Acceleration

- Nios II Processor, Economy version - No Multiply Hardware
  - Uses GNUPro Math Library to Implement Multiplier
- Nios II Processor, Standard - Full Hardware Multiplier
  - 32 x 32 → 32 in 3 Clock Cycles if DSP block present, else uses software only multiplier
- Nios II Processor, Fast - Full Hardware Multiplier
  - 32 x 32 → 32 in 1 Clock Cycles if DSP block present, else uses software only multiplier

Acceleration Hardware	Clock Cycles (32 x 32 → 32)
None	250
Standard MUL in Stratix® FPGA	3
Fast MUL in Stratix FPGA	1

Hardware Multiplier Support

- Stratix Device Family DSP Blocks
- Cyclone® II and Cyclone III Device Family Multiplier Blocks
  - Multiplication using 18 x 18 Multiplier Block
- Optional LE Implementation
  - Enables HW multiplier support for Cyclone Device Family
  - Can also be used in lieu of DSP Blocks
  - Mul, Shift, Rotate (~ 11 Clocks Per Mul)
  - **Eliminates need for DSP blocks for Nios II MUL (multiplication)**

See **Nios II Processor Handbook**

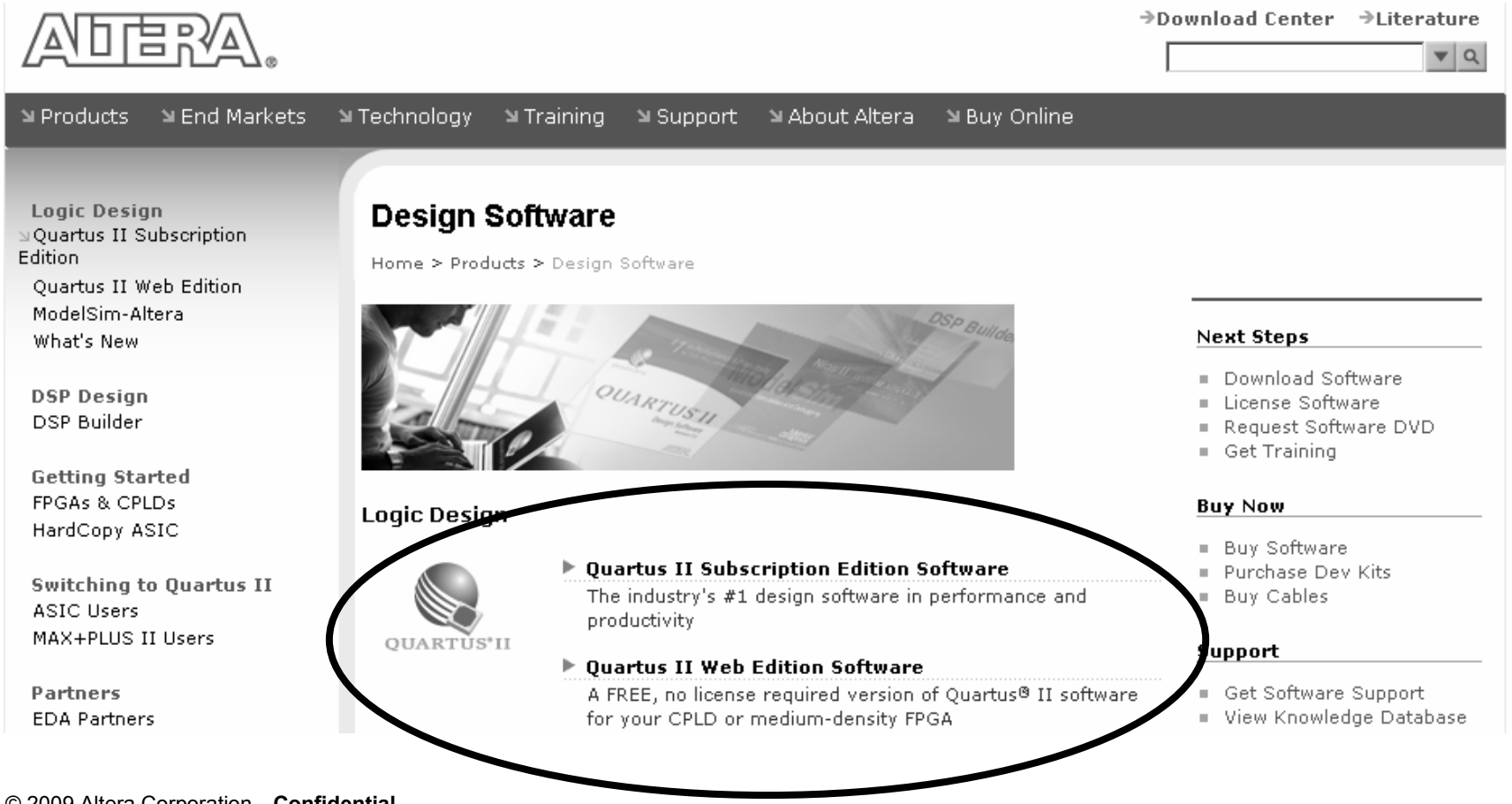
Chapter 5: Core Implementation Details

Licensing

- Nios II Processor Delivered As Encrypted Megacore
  - Licensed Via Feature Line In Existing Quartus II Software License File
  - Consistent With General Altera Megacore Delivery Mechanism
  - Enables Detection Of Nios II Processor IP in Customer Designs (Talkback)
- No Nios II Processor Feature Line (OpenCore Plus Mode)
  - System Runs If Tethered To Host PC
  - System Times Out If Disconnected from PC After ~ 1 hr
- Nios II Processor Feature Line (Active Subscriber)
  - Subscription and New Dev Kit Customers Obtain Licenses From www.altera.com
  - Nios II CPU RTL Remains Encrypted
  - No extra cost when migrating to HardCopy devices
- Nios II Processor Source License
  - Available Upon Request On Case-By-Case Basis
  - Required when migrating to non-Altera ASIC

Installation

- Web Download (or install DVD in Kit)
  - Note: Limited Quartus II Software Web Edition available for free

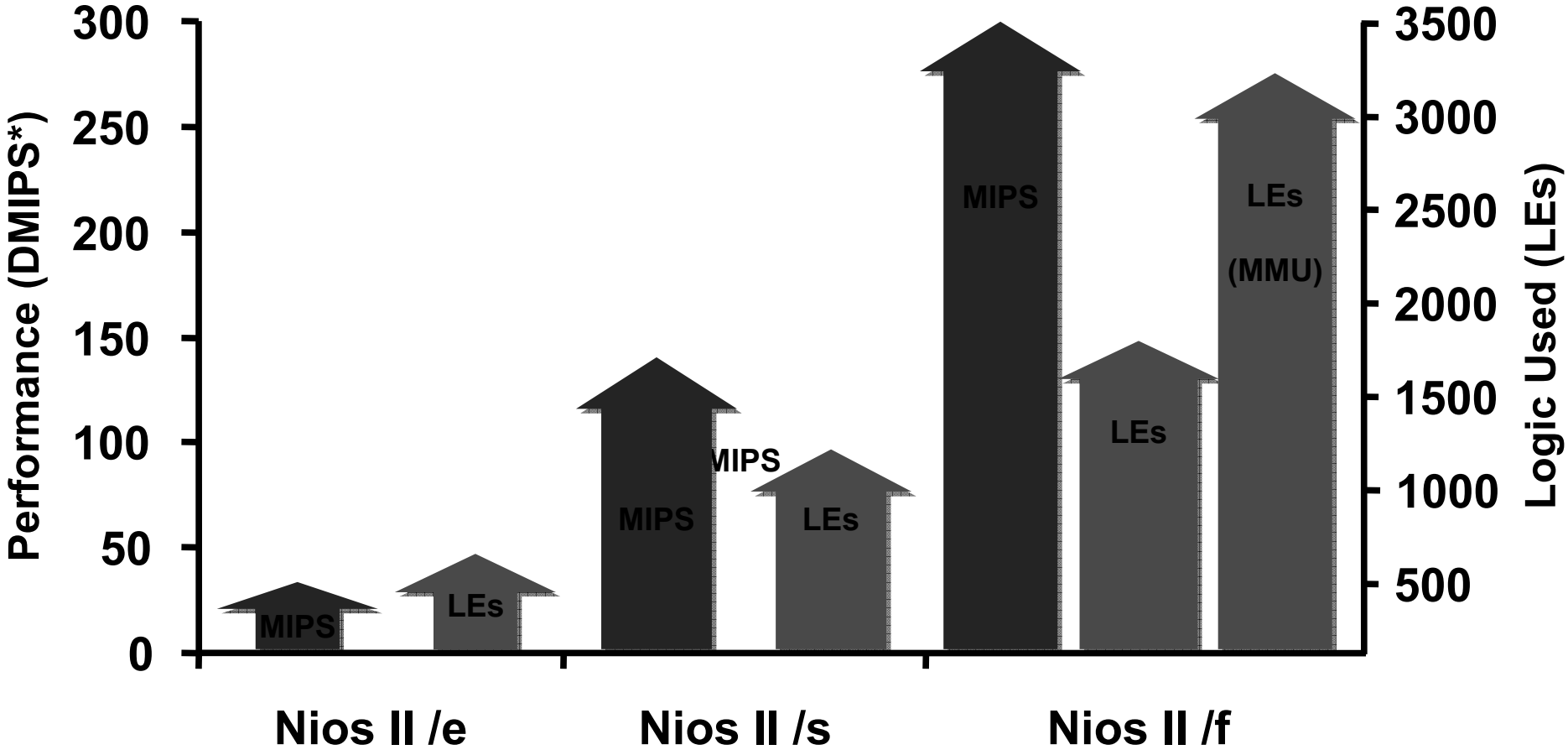


Reqmnts. for Nios II Processor Designs

- Quartus II software version 8.1
  - Required for Nios II Processor version 8.1
- No spaces in Quartus II project pathname
- No spaces in installation path
  - Follow defaults → Install in **altera** directory
- Nios II Processor license

  or
- Programming cable tethered to PC to run OpenCore Plus version of the Nios II processor

Performance Range in FPGA of Nios II Processor



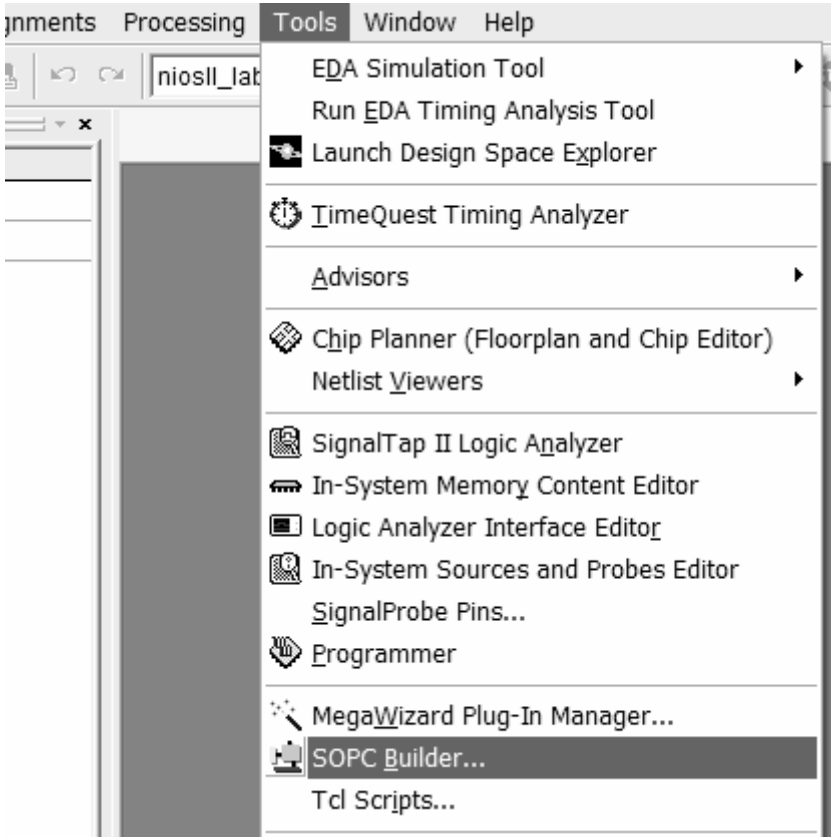
* Dhrystone 2.1 Benchmark

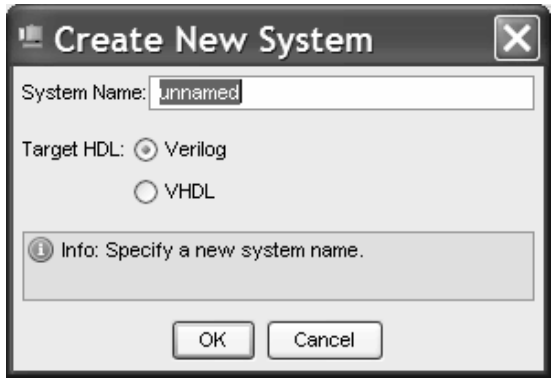
Nios II Processor Performance (DMIPS)

Device Family	Nios II /f	Nios II /s	Nios II /e
Std Cell ASIC (90 nm est.)	>500		
Stratix III / Stratix IV FPGAs (prelim)	340	140/150	48
Stratix II FPGAs	250	110	45
Stratix FPGAs	170	80	27
Hardcopy II devices	230	130	50
Hardcopy devices	165	85	27
Cyclone III FPGAs	195	90	30
Cyclone II FPGAs	145	55	18
Cyclone FPGAs	130	52	17

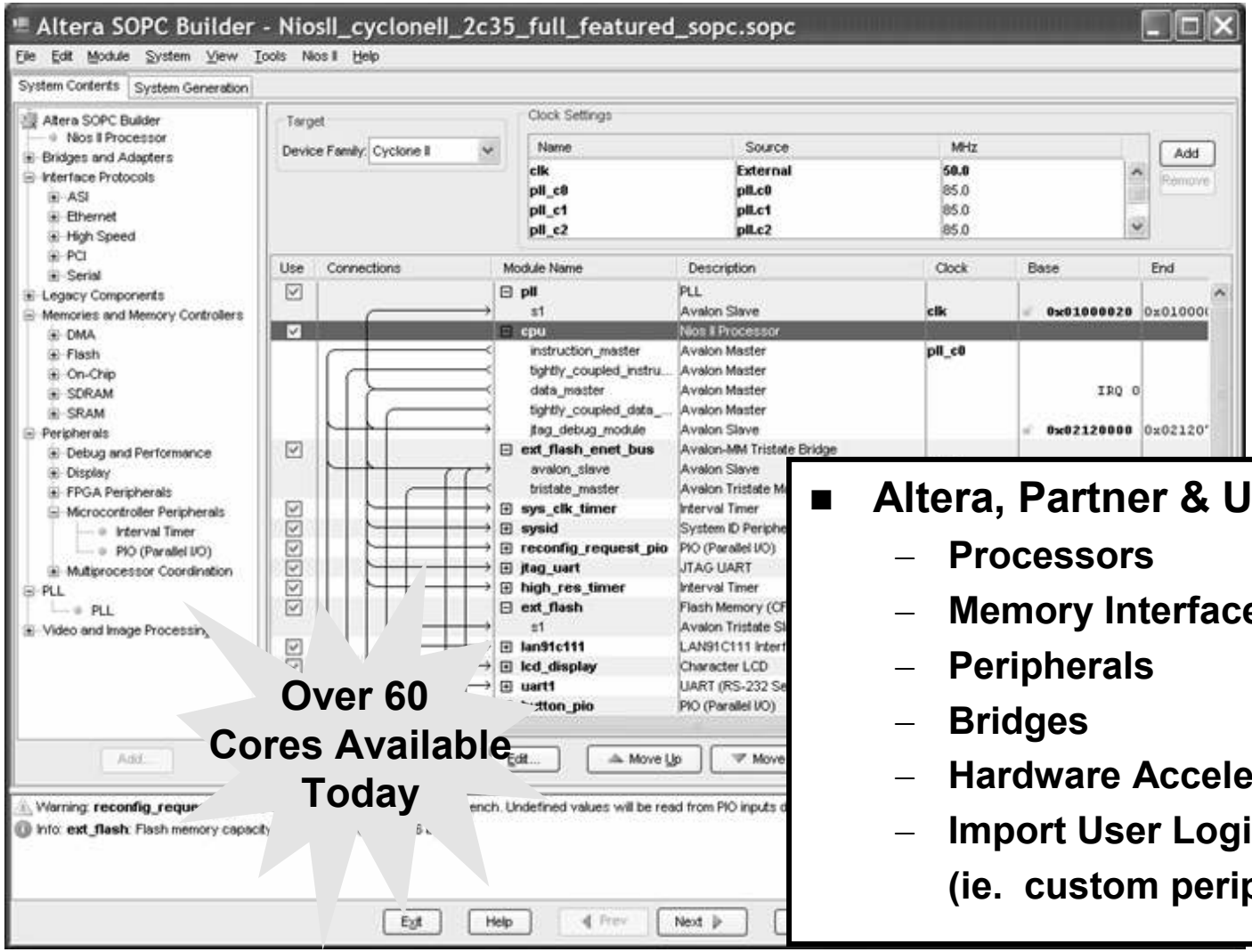
Open SOPC Builder from Quartus II Software

- See Quartus II software **Tools** Menu
- Choose implementation language



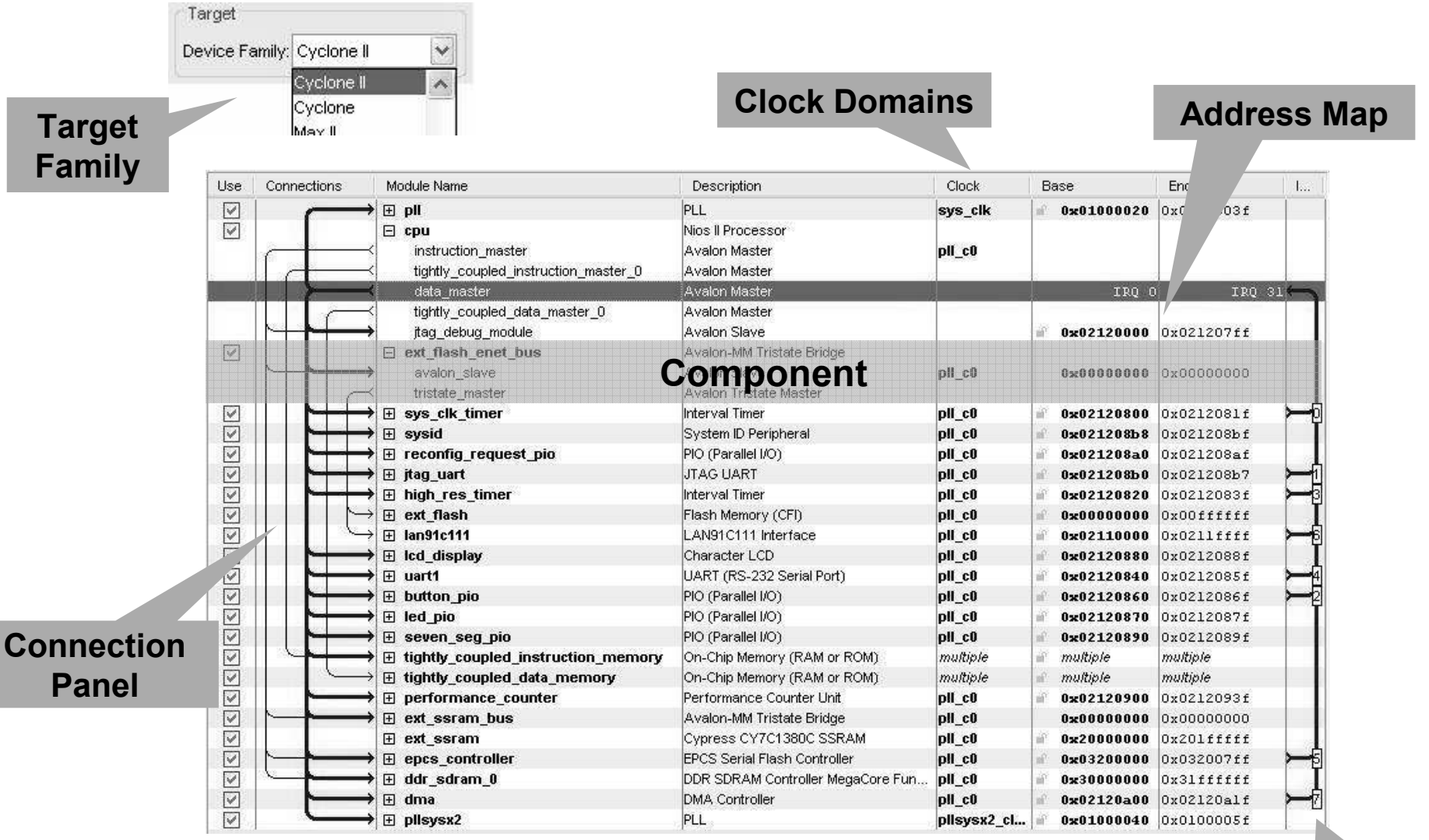


SOPC Builder - System Contents Page

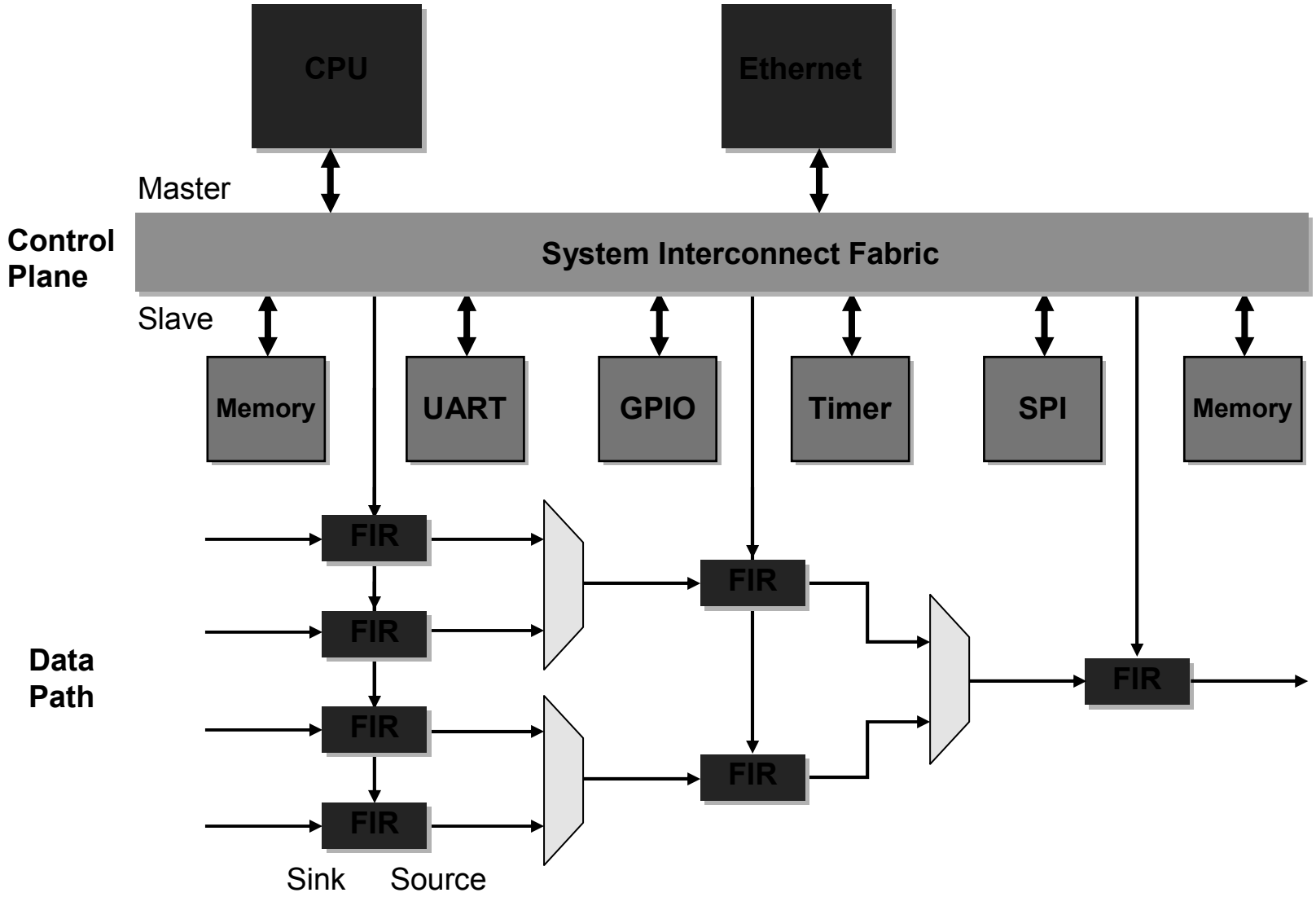


- **Altera, Partner & User Cores**
  - **Processors**
  - **Memory Interfaces**
  - **Peripherals**
  - **Bridges**
  - **Hardware Accelerators**
  - **Import User Logic (ie. custom peripherals)**

System Contents Page Features

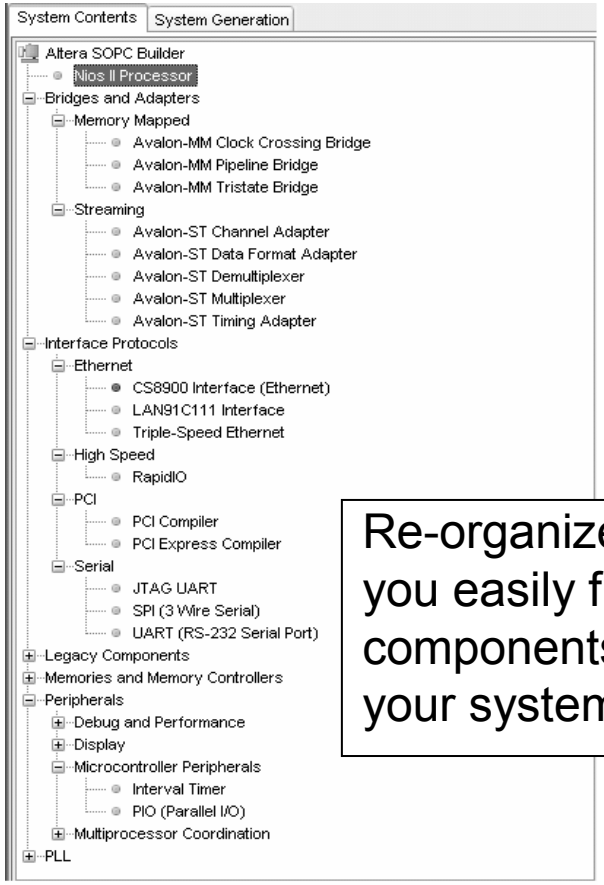


Build Up “Control Plane” & “Data Path”



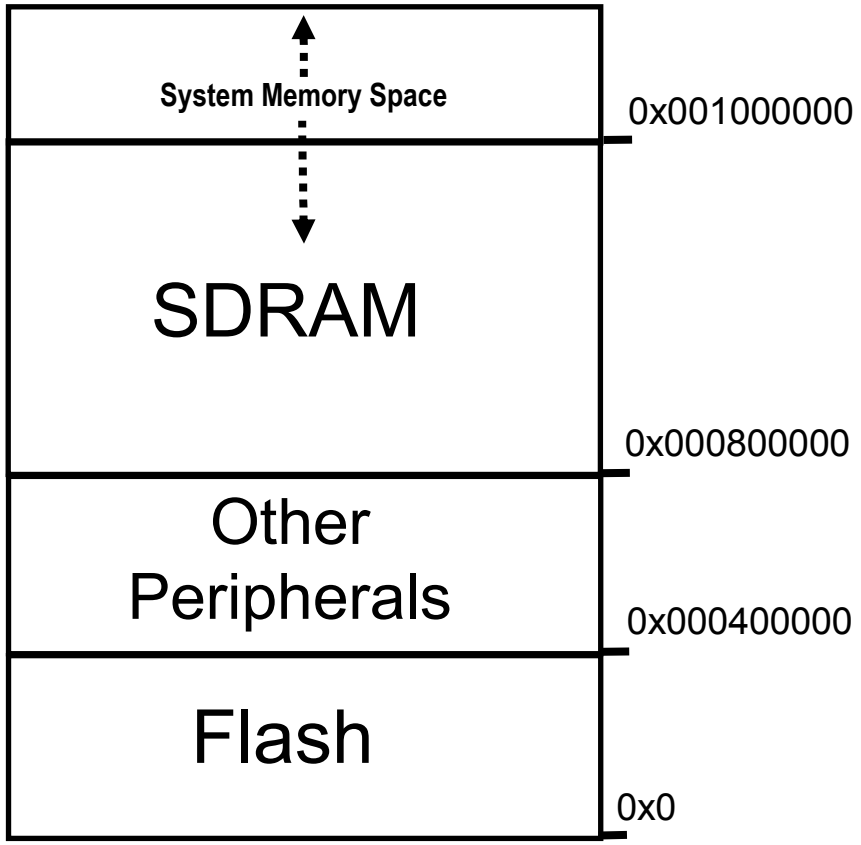
Insert Peripherals Including Nios II Processor

- Double-click on peripheral **or** press **Add…**
  - Build up memory map of your embedded system

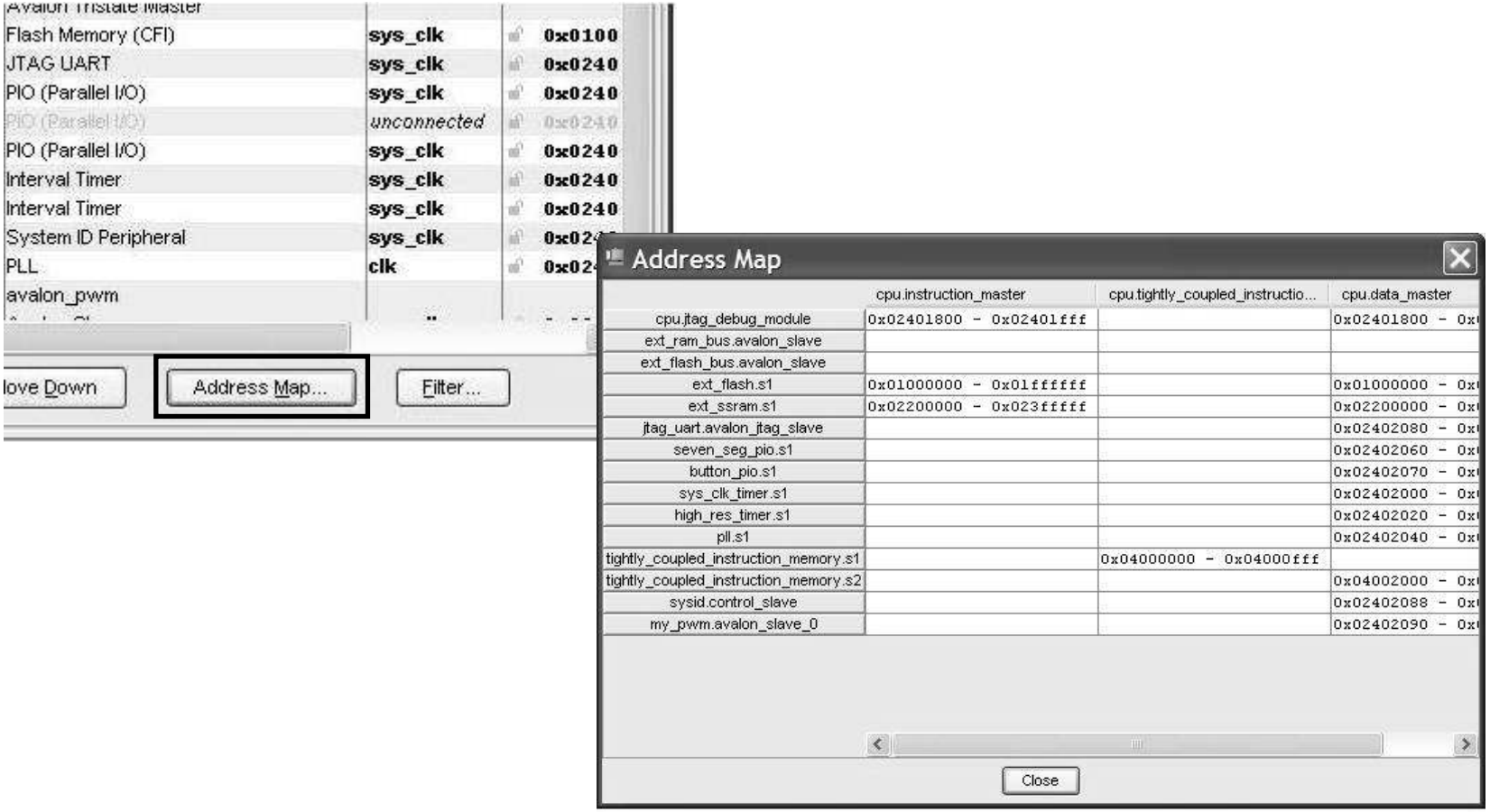


Re-organized to help you easily find components to add to your system

Eg.

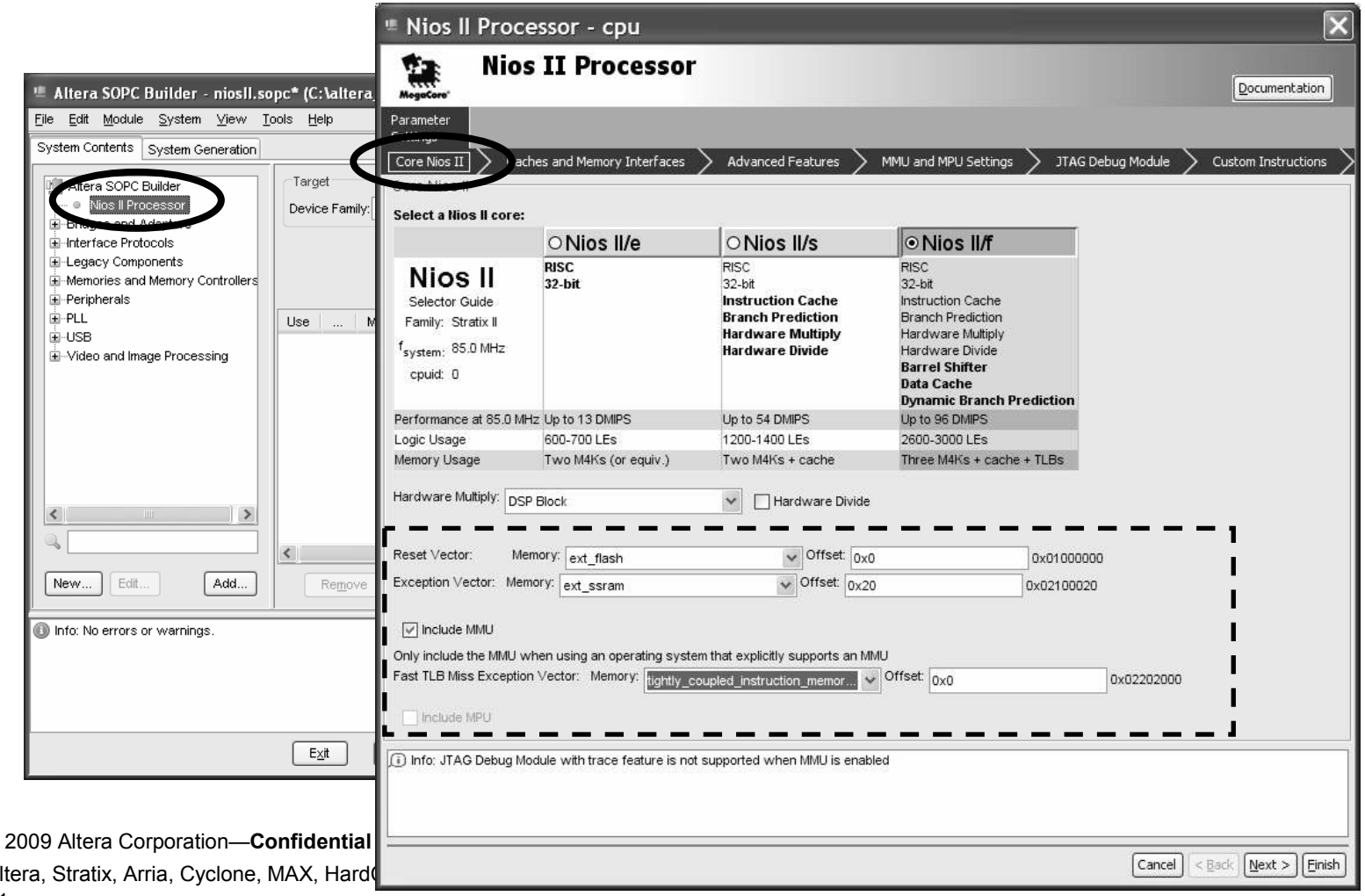


View Map of Memory Addresses



Add and Configure Nios II CPU

- Hardware designer selects Nios II processor version
  - Economy, Standard, or Fast



Memories must be defined in order to select **reset** or **exception** address

Reset and **Exception** Addresses

- Set *after* memory component added to system
- Reset should be in non-volatile memory
  - In systems not connected to Quartus II Programmer
- Exceptions processed at “exception location”
  - Exception handler code provided by HAL system library
  - Supported Exception Types
    - Software Exceptions
      - Software Traps (currently, not implemented)
      - Unimplemented instructions
        - Maintains compatibility between Nios II processor cores
    - Hardware Interrupts
      - 32 Level-sensitive interrupts are supported.
    - More exceptions will be supported as features are added

Select Cache and TCM Settings

- Adjust Size of Instruction and Data Cache Memory
  - Can now completely disable data cache on fast core
  - And also disable instruction cache as long as TCM used
- Enable Instruction / Data **T**ightly **C**oupled **M**emory masters
- Control Data Cache Line
  - Up to 32 byte cache line width possible for better burst support

Tightly Coupled Masters

- Gives Nios II Processor Fast Access to On-Chip Memories
- CPU Adds Extra “Local” Master Interfaces

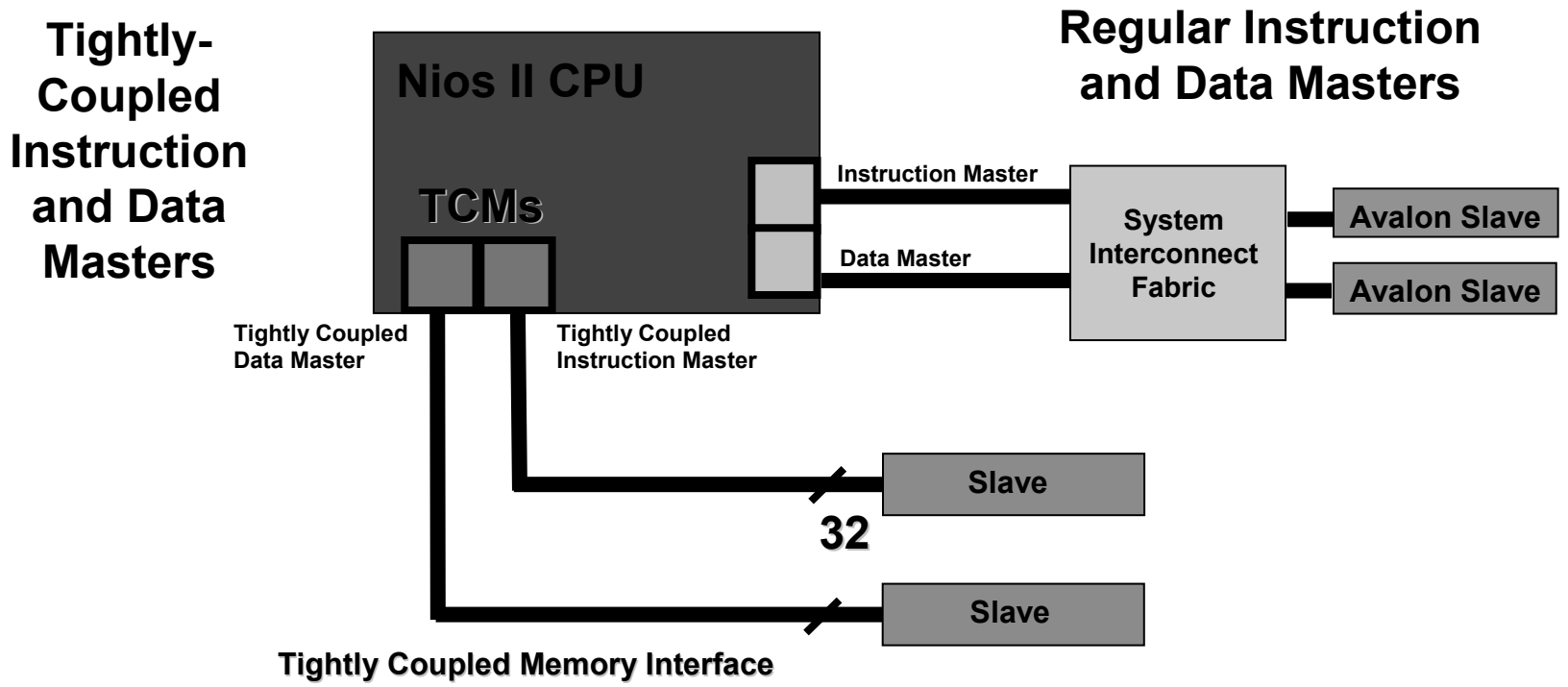
	Instruction TCM	**Data TCM**
Nios II/f	Up to 4	Up to 4
Nios II/s	Up to 4	None
Nios II/e	None	None

- Only Fast “Qualified” Slaves Can Attach
  - ie. On-Chip Memory
    - Read Latency of 1, Write Latency of 0
    - Dual-port Memories Required for Rapid Data Flow

Performance Enhancement!

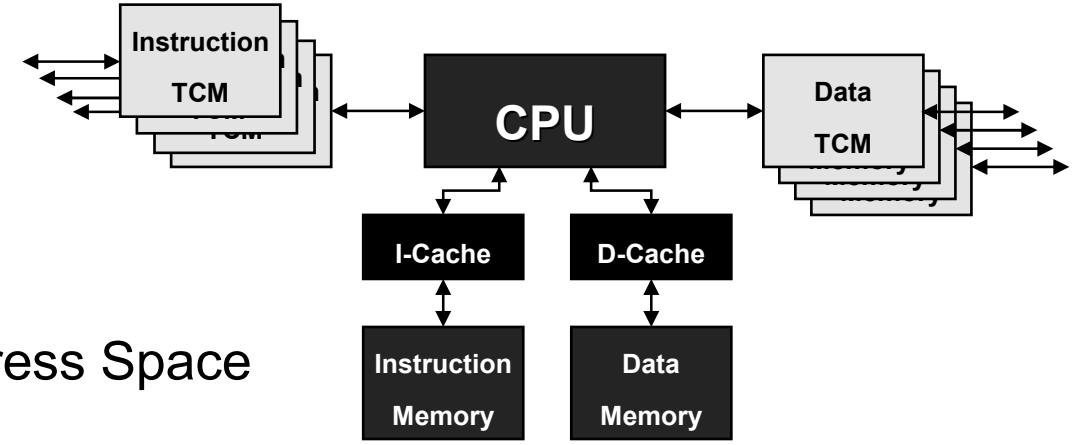
Tightly Coupled Masters

- Connected to tightly-coupled slaves through
  - "**Tightly Coupled Memory Interfaces**"
    - To on-chip true dual port memories, allowing **"Normal"** data master to connect to second port, allowing reading and writing of data



Accessed in Parallel to Cache

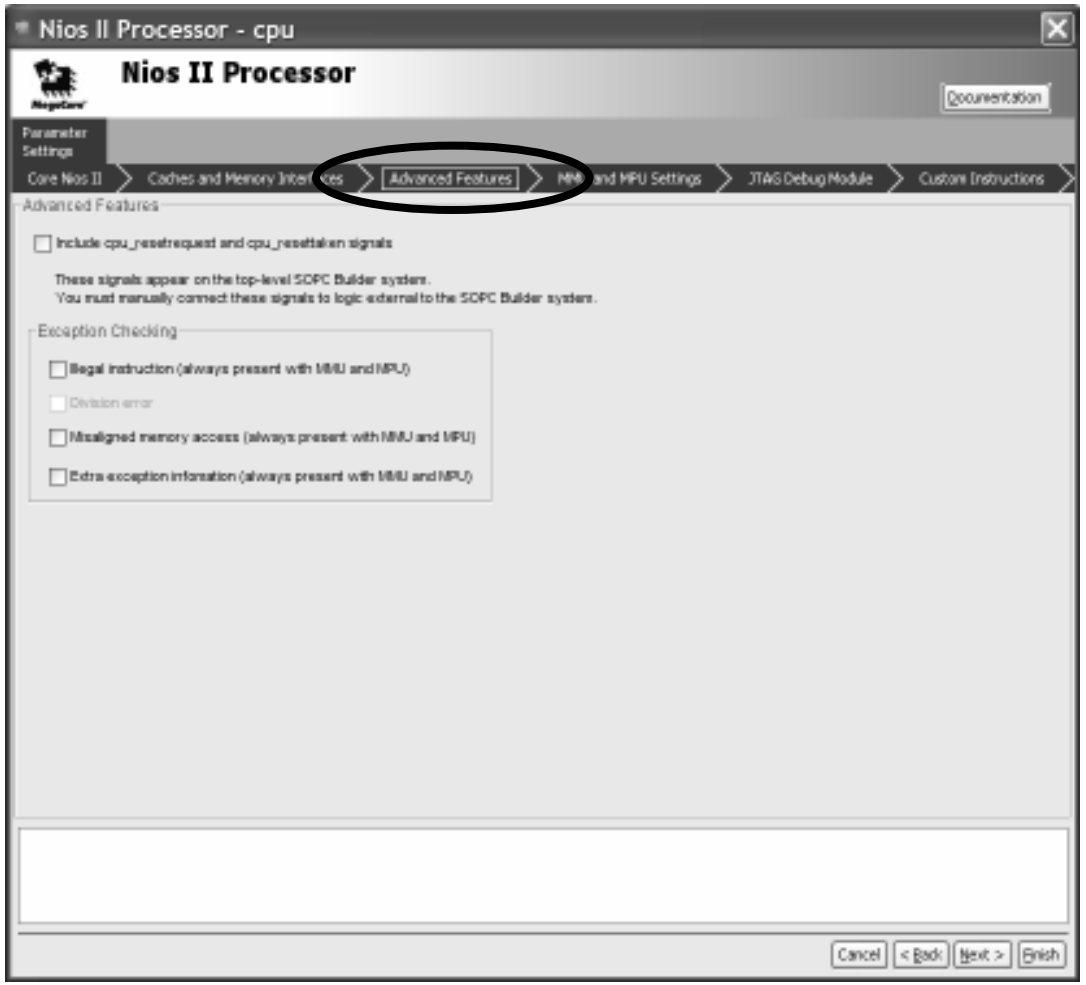
- Tightly-coupled memories are accessed in parallel with cache (ie. instruction or data)
  - Act like cache with a 100% hit rate
  - Great for ISRs and other time critical functions
- Address decoders in CPU determine if address resides in TCM or normal system address range



- Assign TCM's to High Address Space
  - To minimize addressing logic
  - Maximize performance

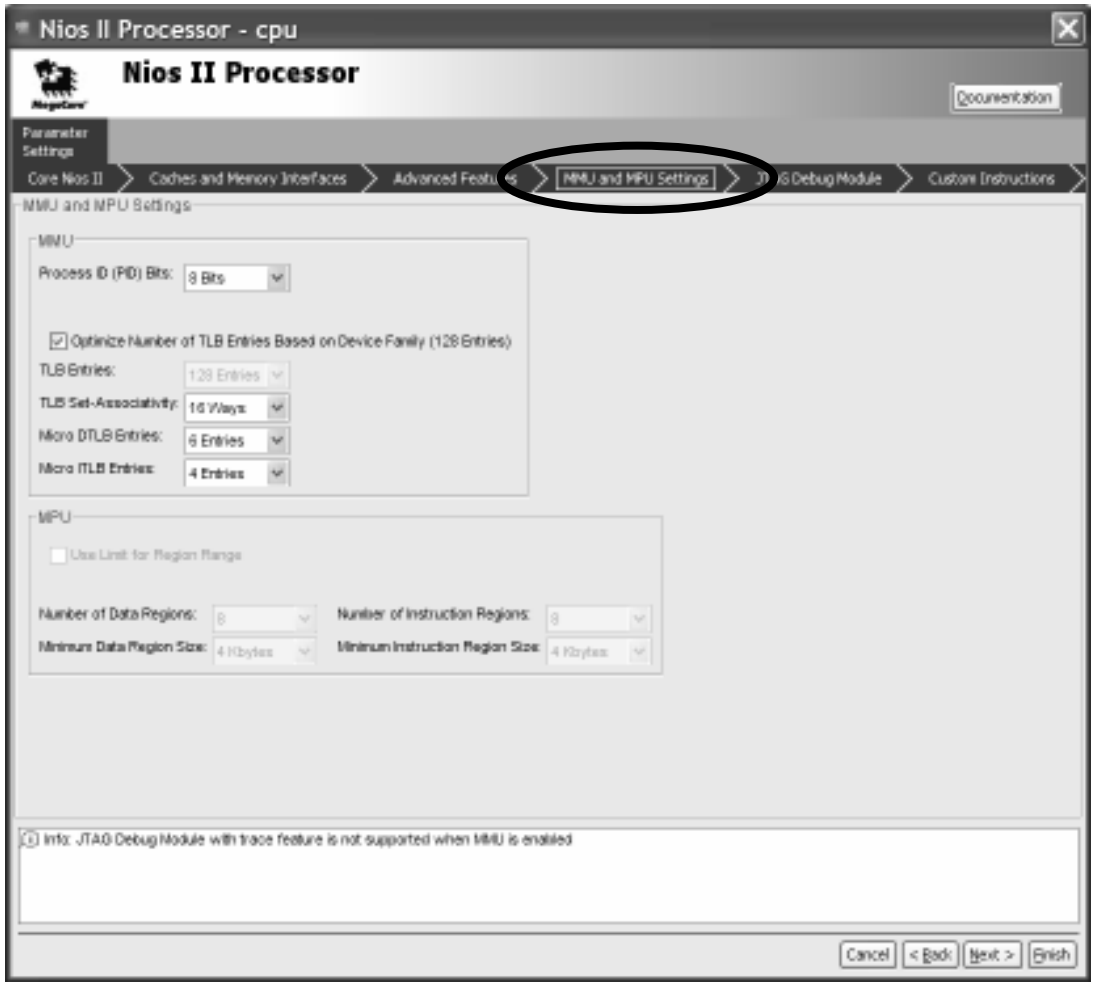
Advanced Features Tab

- Export **cpu_resetrequest** and **cpu_resettaken** signals



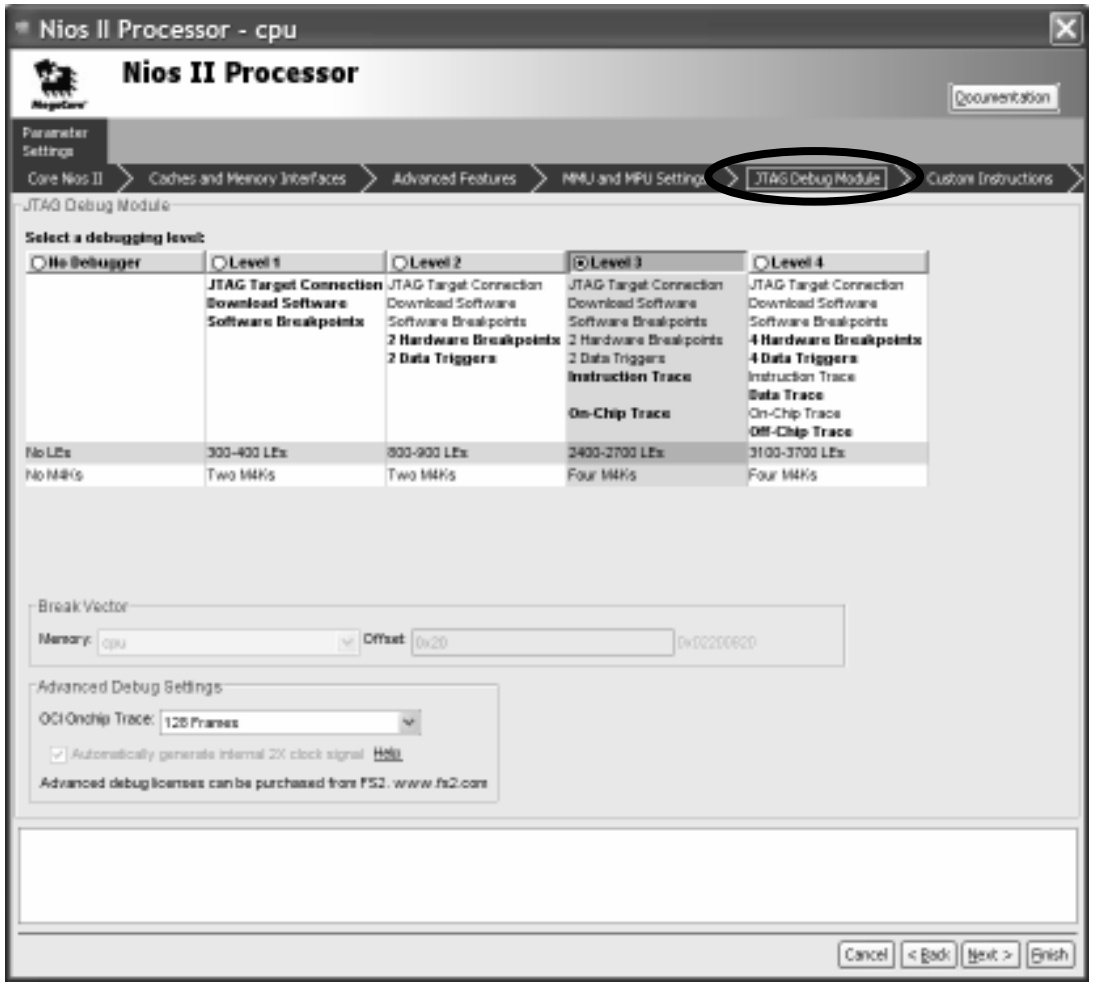
MMU and MPU Settings

- Configure features as required by application

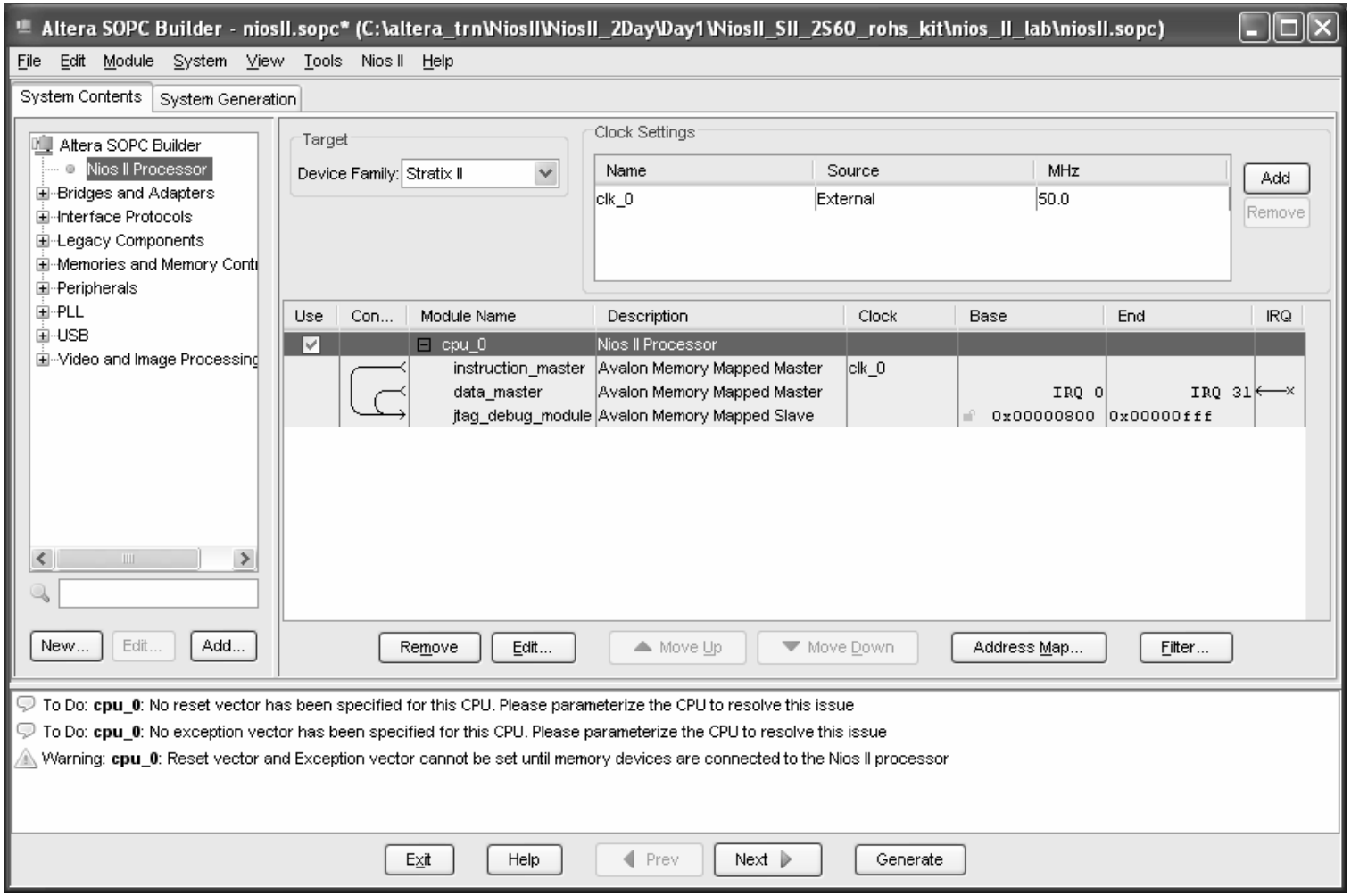


Choose JTAG Debug Core

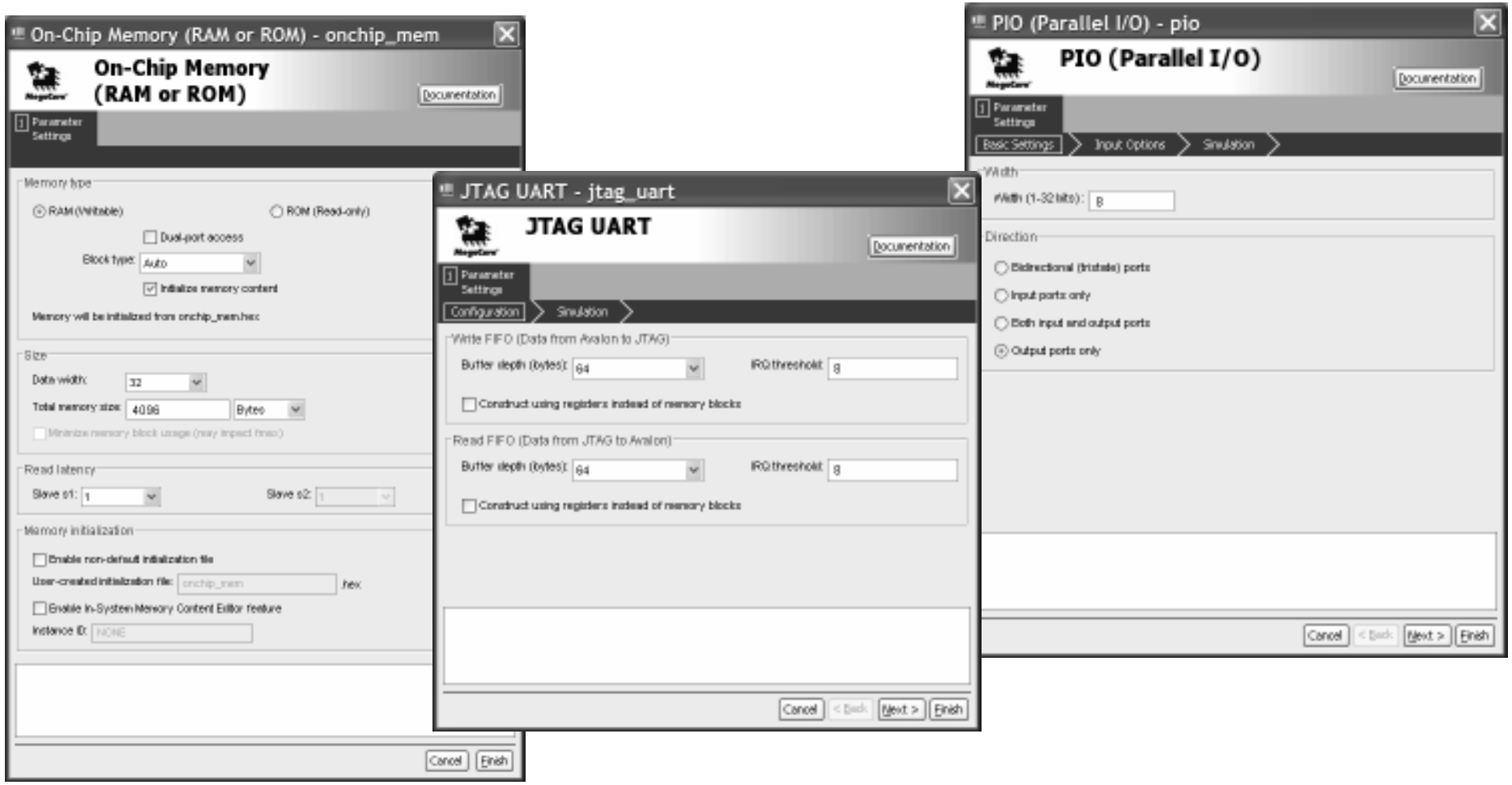
- Select appropriate JTAG Debug level when configuring core



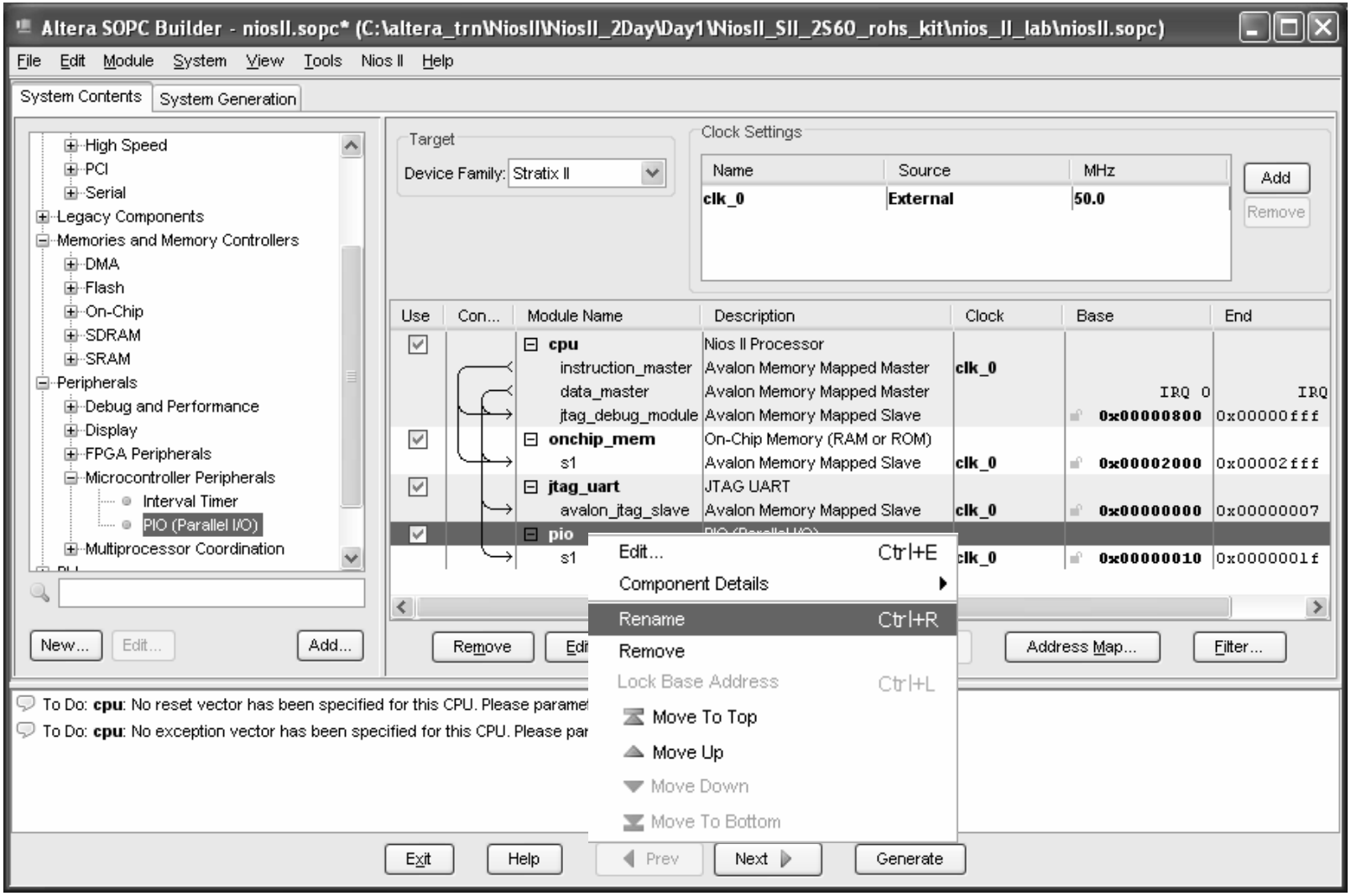
CPU Added to System



Other Example Component GUIs

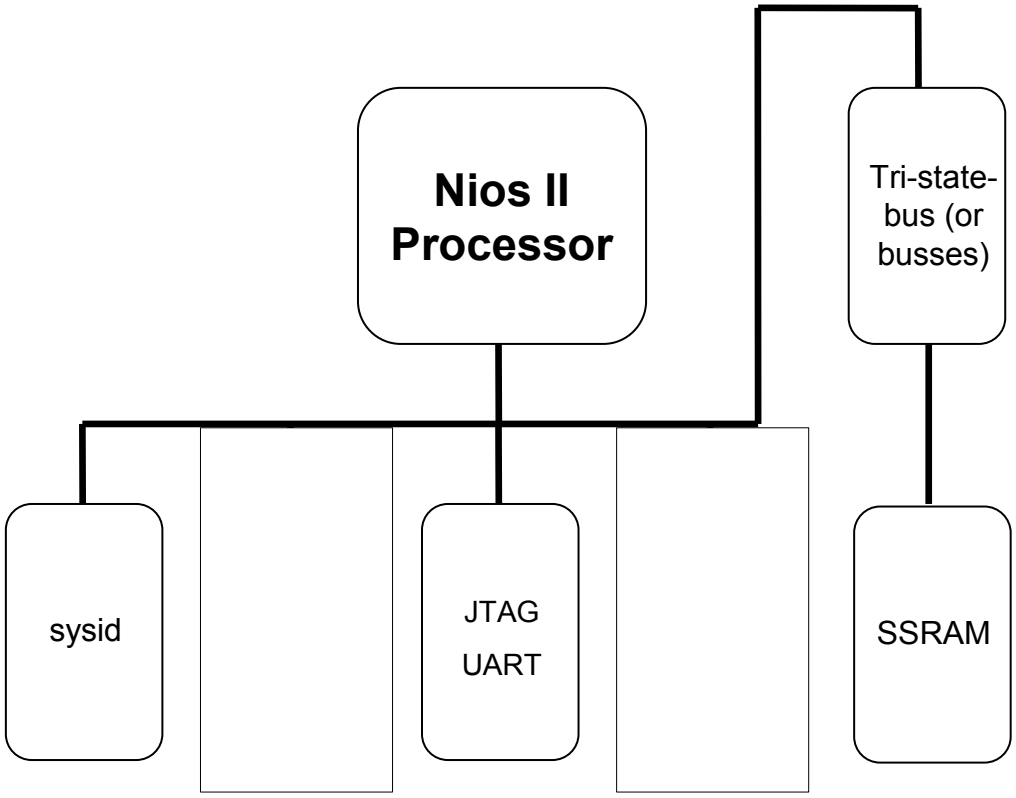


Other Components Added



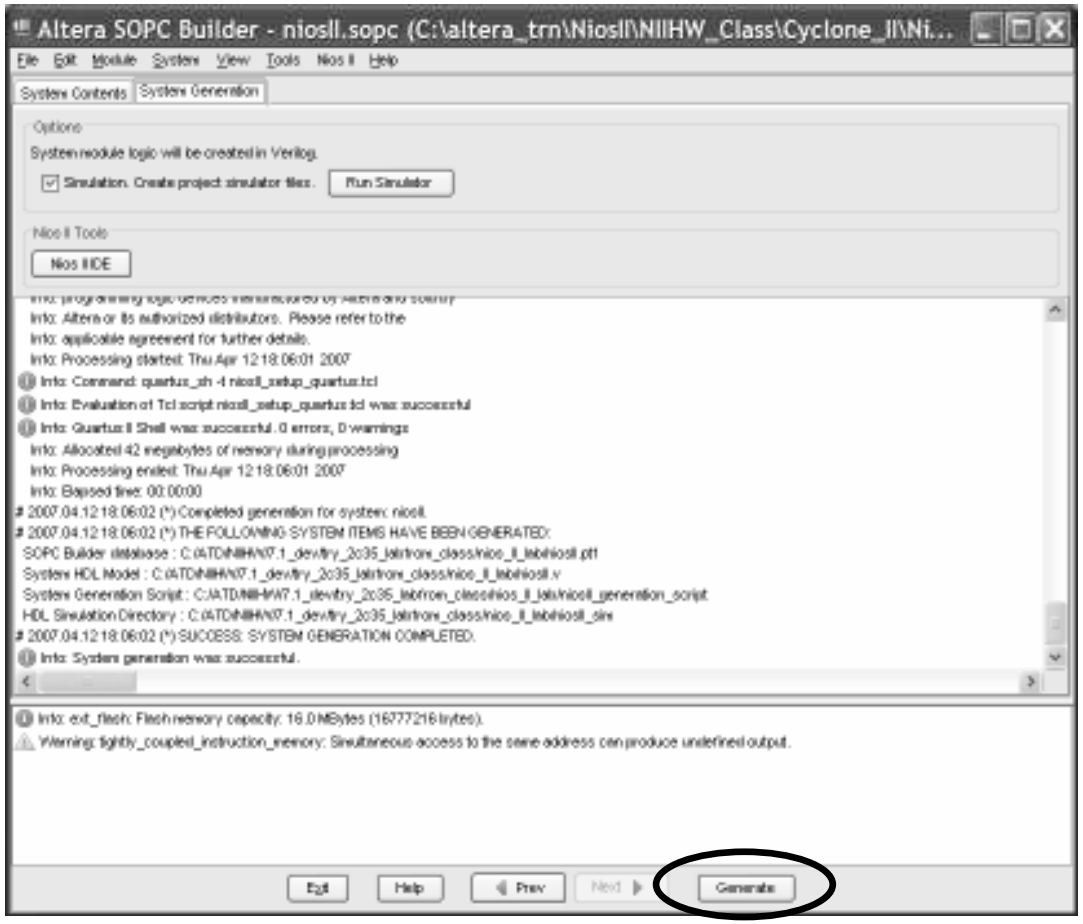
System Block Diagram

- System Resembles the following



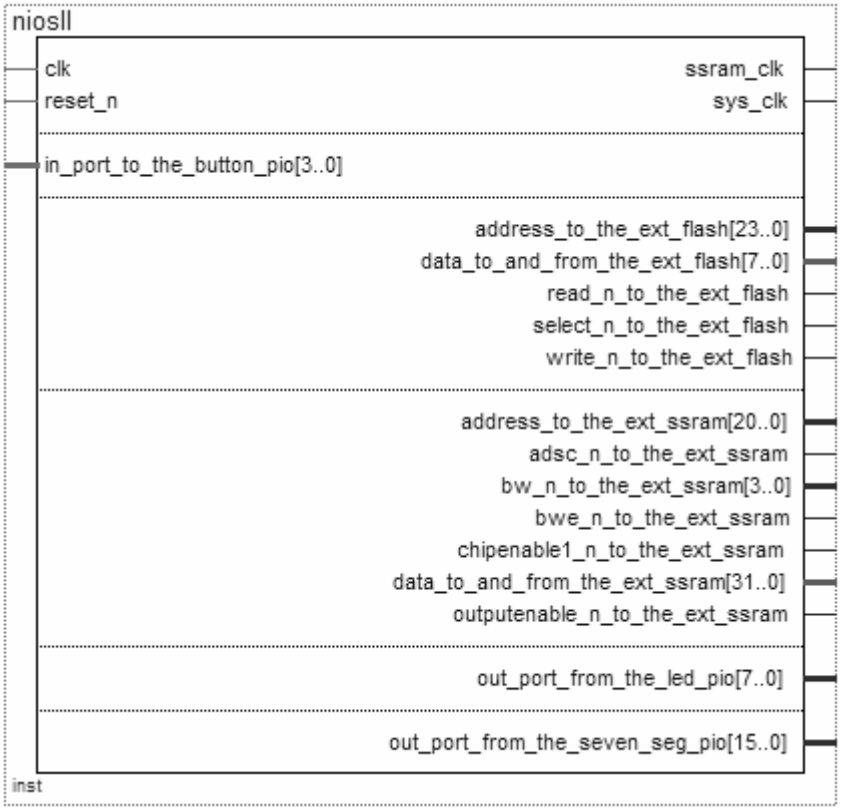
SOPC Builder – System Generation Page

- Generate **ModelSim Project**
  - *Note: Can also launch **Nios II IDE** from here or start **Simulator***



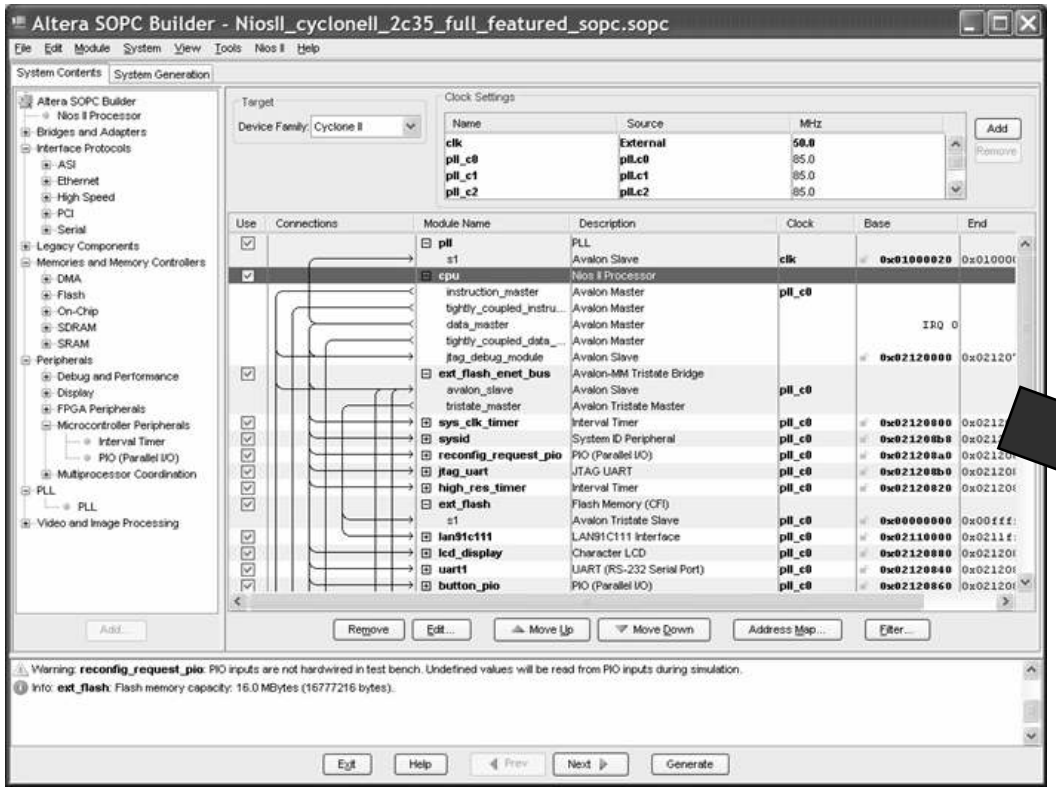
System Output Files

- **Verilog** or **VHDL** system description files
- **Block Diagram** File
  for top level system



Other SOPC Builder Output Files

- .SOPC File
  - Text file that records SOPC Builder edits and describes Nios II System
- .PTF file
  - Needed by Nios II IDE to describe system contents

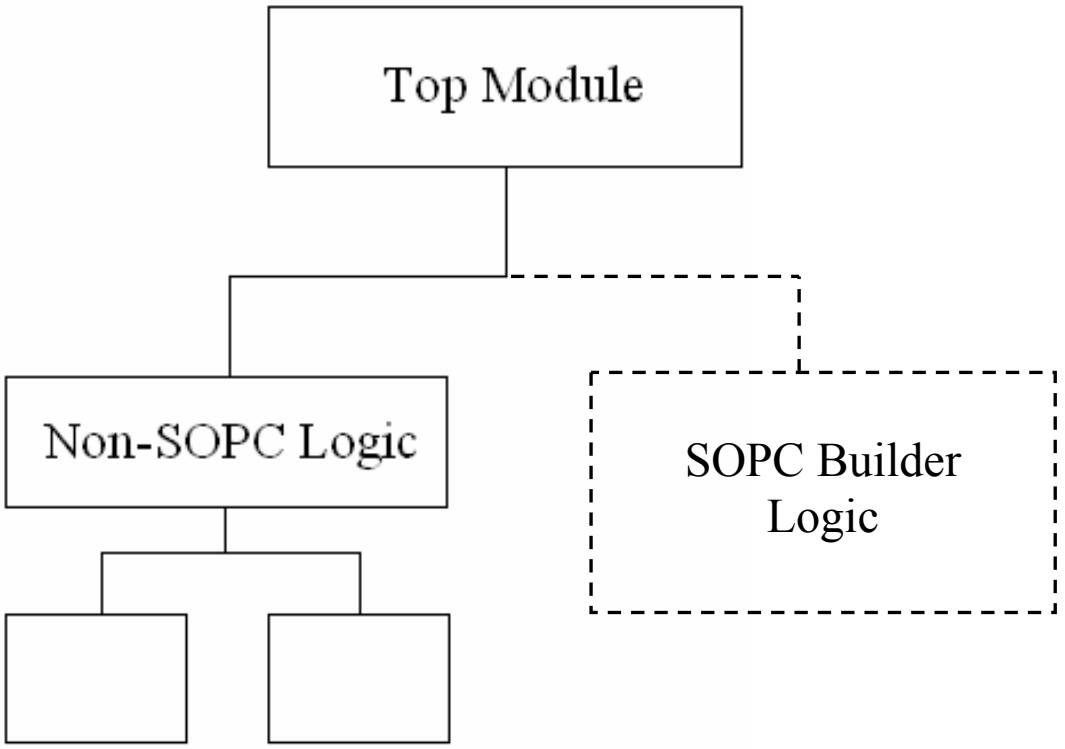


.SOPCInfo File

- Needed by Command Flow (ie. the “BSP Tools”) to describe system contents
  - Use this instead of the .ptf file
- Contains the following information:
  - Project Name & SOPC Builder tool version
  - HDL Language
  - Component Names & version in search path
  - File Locations on disk
  - Module Names & versions
  - Interface information, including signal names, types, properties
  - Parameter names & values
  - Information about each connection, such as
    - What components & interfaces are being connected
    - Base address (MM), IRQ Number (IRQs), etc.
    - Memory map as seen by each master

Integrate SOPC Builder Hardware Sub-System

- Into top level design in Quartus II using either HDL code or schematic entry tool



Verilog Instantiation

- *Locate in "system" HDL file* →

```
// Adapted from low-cost reference design:

module  top_level (
                        // inputs:
                        in_port_to_the_button_pio,
                        reset_n,
                        sys_clk,

                        // outputs:
                        clk_to_sdram,
                        clk_to_sdram_n,
                        ddr_a,
                                    ⋮
                        ddr_ras_n,
                        ddr_we_n,
                        out_port_from_the_led_pio,
                        out_port_from_the_seven_seg_pio,
                        pll_c0_out,
                        pll_c1_out
);


// Port Declarations …
                                    ⋮

// Wire Declarations …
                                    ⋮
```

```
SOPC_system SOPC_system_inst
  (
    .clk_to_sdram_from_the_ddr_sdram_0 (single_bit_clk_to_sdram),
    .clk_to_sdram_n_from_the_ddr_sdram_0 (single_bit_clk_to_sdram_n),
    .ddr_a_from_the_ddr_sdram_0 (ddr_a),
    .ddr_ba_from_the_ddr_sdram_0 (ddr_ba),
    .ddr_cas_n_from_the_ddr_sdram_0 (ddr_cas_n),
    .ddr_cke_from_the_ddr_sdram_0 (single_bit_ddr_cke),
    .ddr_cs_n_from_the_ddr_sdram_0 (single_bit_ddr_cs_n),
    .ddr_dm_from_the_ddr_sdram_0 (ddr_dm),
    .ddr_dq_to_and_from_the_ddr_sdram_0 (ddr_dq),
    .ddr_dqs_to_and_from_the_ddr_sdram_0 (ddr_dqs),
    .ddr_ras_n_from_the_ddr_sdram_0 (ddr_ras_n),
    .ddr_we_n_from_the_ddr_sdram_0 (ddr_we_n),
    .in_port_to_the_button_pio (in_port_to_the_button_pio),
    .out_port_from_the_led_pio (out_port_from_the_led_pio),
    .out_port_from_the_seven_seg_pio (out_port_from_the_seven_seg_pio),
    .pll_c0_out (pll_c0_out),
    .pll_c1_out (pll_c1_out),
    .pll_c2_out (pll_c2_out),
    .reset_n (reset_n),
    .sys_clk (sys_clk),
    .write_clk_to_the_ddr_sdram_0 (write_clk_to_the_ddr_sdram_0)
  );

// Local assignments:
                                    ⋮

endmodule
```

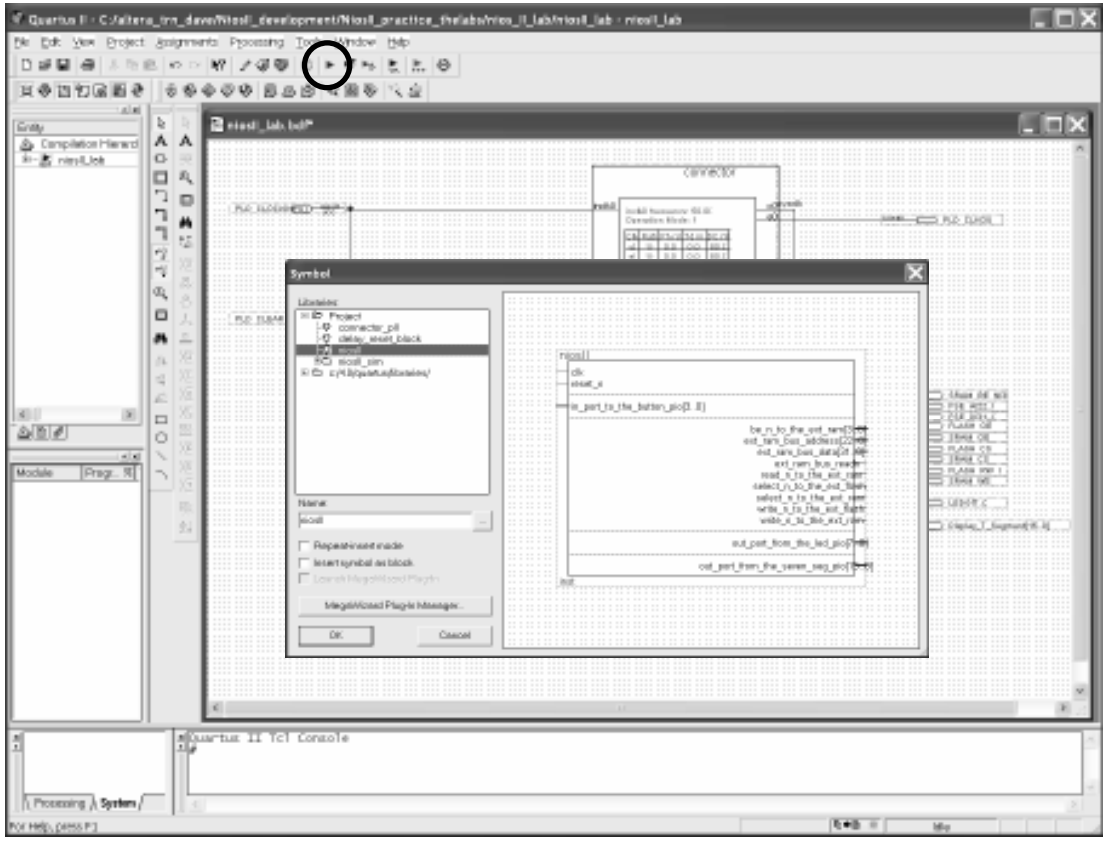
VHDL Instantiation

```
// Adapted from low-cost reference design:     (locate in system HDL file)
library altera;
use altera.altera_europa_support_lib.all;
library ieee;

use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


entity top_level is
  port (
-- inputs:
  signal in_port_to_the_button_pio : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
  signal reset_n : IN STD_LOGIC;
  signal sys_clk : IN STD_LOGIC;
-- outputs:
  signal clk_to_sdram : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
  signal clk_to_sdram_n : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
  signal ddr_a : OUT STD_LOGIC_VECTOR (12 DOWNTO 0);
                                        ⋮
end entity NiosII_cycloneII_2c35_low_cost;


architecture europa of top_level is

component SOPC_system is
    PORT (
    signal ddr_dm_from_the_ddr_sdram_0 : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
    signal clk_to_sdram_from_the_ddr_sdram_0 : OUT STD_LOGIC;
    signal ddr_ba_from_the_ddr_sdram_0 : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
                                        ⋮
    signal reset_n : IN STD_LOGIC;
    signal sys_clk : IN STD_LOGIC;
                                        ⋮
end component SOPC_system;

// Signal Declarations …
                                        ⋮
```

```
                                        ⋮
begin

SOPC_system_instance  SOPC_system
    port map(
    clk_to_sdram_from_the_ddr_sdram_0 => single_bit_clk_to_sdram,
    clk_to_sdram_n_from_the_ddr_sdram_0 => single_bit_clk_to_sdram_n,
    ddr_a_from_the_ddr_sdram_0 => internal_ddr_a,
    ddr_ba_from_the_ddr_sdram_0 => internal_ddr_ba,
    ddr_cas_n_from_the_ddr_sdram_0 => internal_ddr_cas_n,
    ddr_cke_from_the_ddr_sdram_0 => single_bit_ddr_cke,
    ddr_cs_n_from_the_ddr_sdram_0 => single_bit_ddr_cs_n,
    ddr_dm_from_the_ddr_sdram_0 => internal_ddr_dm,
    ddr_dq_to_and_from_the_ddr_sdram_0 => ddr_dq,
    ddr_dqs_to_and_from_the_ddr_sdram_0 => ddr_dqs,
    ddr_ras_n_from_the_ddr_sdram_0 => internal_ddr_ras_n,
    ddr_we_n_from_the_ddr_sdram_0 => internal_ddr_we_n,
    in_port_to_the_button_pio => in_port_to_the_button_pio,
    out_port_from_the_led_pio => internal_out_port_from_the_led_pio,
    out_port_from_the_seven_seg_pio => internal_out_port_from_the_seven_seg_pio,
    pll_c0_out => internal_pll_c0_out,
    pll_c1_out => internal_pll_c1_out,
    pll_c2_out => pll_c2_out,
    reset_n => reset_n,
    sys_clk => sys_clk,
    write_clk_to_the_ddr_sdram_0    );

-- Local assignments:
                                        ⋮
end europa;
```

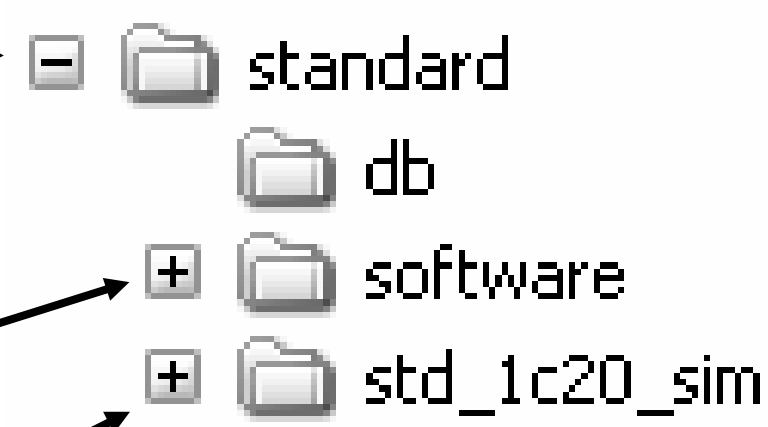
Instantiation in Block Diagram File

- Drop in component as shown below
- Then **compile** design  Quartus II software



Quartus II Software - Project Directories

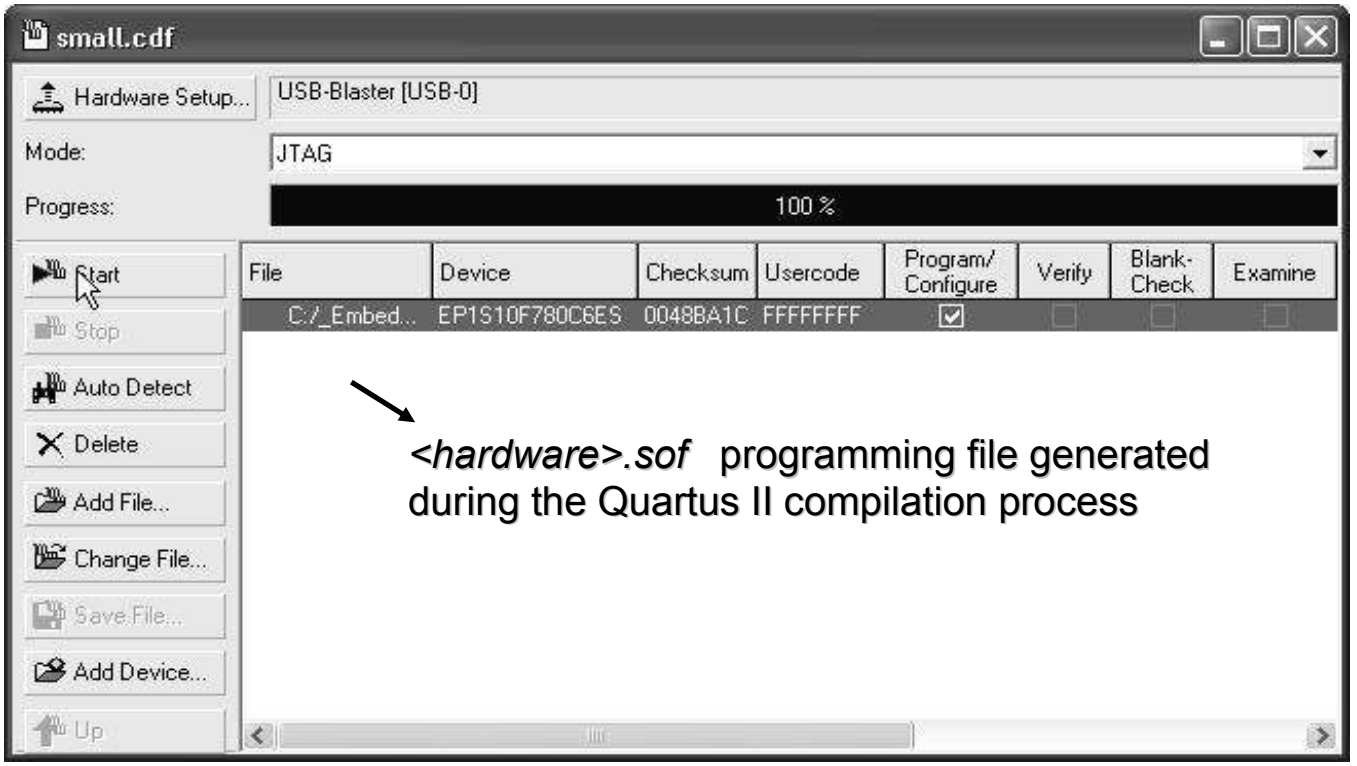
- **Hardware**
  - HDL Source & Netlist
  - db - Quartus project database
- **Software**
  - Application source code
  - Library files
- **Simulation**
  - ModelSim project
  - Automatically generated test memory and vectors



All other Quartus II and SOPC Builder output files …

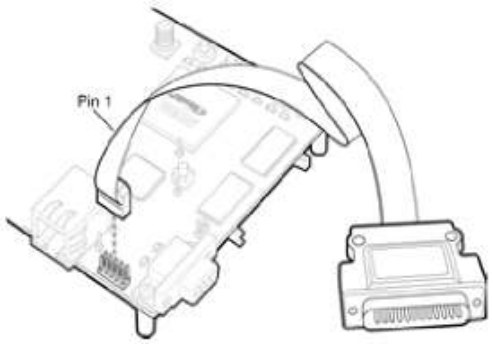
Using Quartus II Programmer

- Launch from Quartus II software after compiling design
  - To program FPGA with .sof file (ie. FPGA programming bitstream)



Some Noteworthy Peripherals

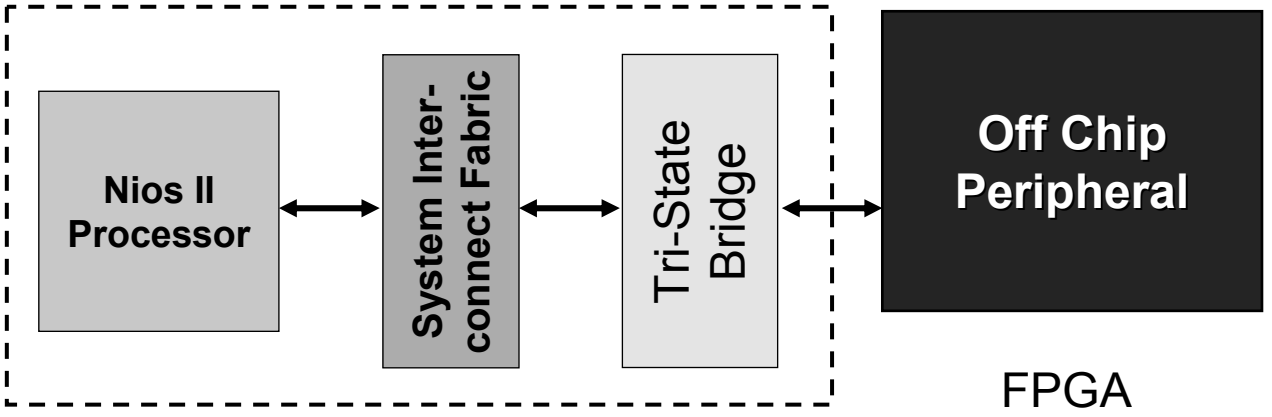
- **JTAG UART**
  - Single JTAG Connection:
    - Device Configuration
    - Flash Programming
    - Code Download
    - Debug
    - Target STDIO (printing)



- **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

Example Peripherals (cont.)

- Avalon-MM Tristate Bridge
  - Available as an SOPC Builder component
  - Connect to off-chip tri-state peripherals
    - Also lets you share pins



- Defined by the presence of a bi-direction data port
- *Note: Off-chip peripherals do not **have** to be tri-state*

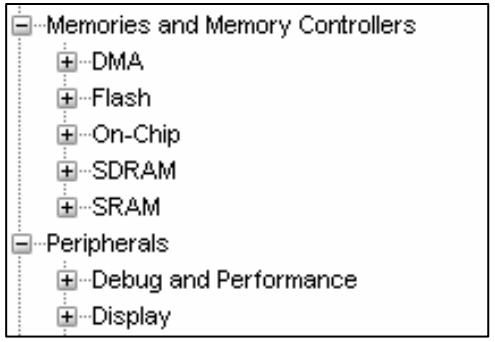
Example Peripherals (cont.)

- **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

Peripheral Now Provided with the Nios II IDE and Supported through the Nios Forum

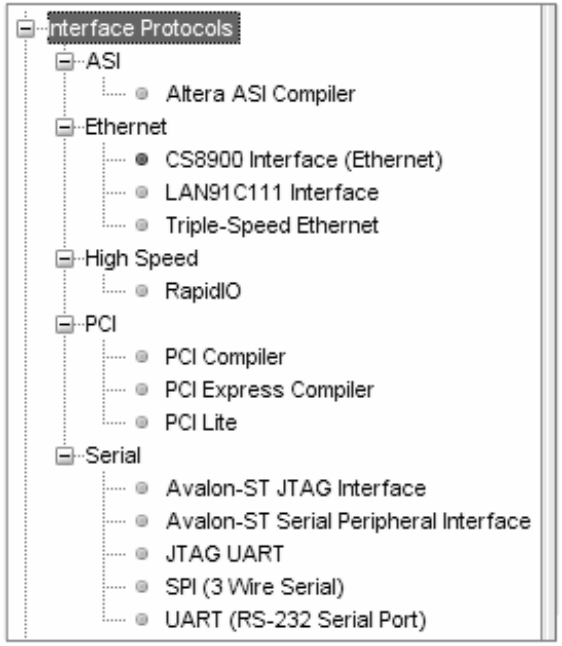
www.niosforum.com

- **LCD Display**
- **Memory Interfaces**
  - EPCS Serial Flash Controller
  - On-Chip
    - RAM, ROM
  - Off-Chip
    - SRAM
    - SDRAM
    - CFI Flash
    - SSRAM Controller
      - Cypress CY7C1380C Sync SRAM controller



Example Peripherals (cont.)

- **Support for DDR/DDR2 in SOPC Builder GUI**
  - With burst adapter
    - Sequential master to interleaved slave enhancement
  - Separate READ/Write duplex slaves
    - Automatically matches address of read/write slaves
    - Arbitration logic connects read/write masters to both slaves
- **Now, High Performance DDR/DDR2**
- **Support for PCI and Bursting DMA in SOPC Builder GUI**
  - Higher bandwidth transfers through PCI
- **Serial RapidIO, PCIe(x1,x4)**
- **Triple-Speed Ethernet MegaCore**



High Performance Memory Interfaces

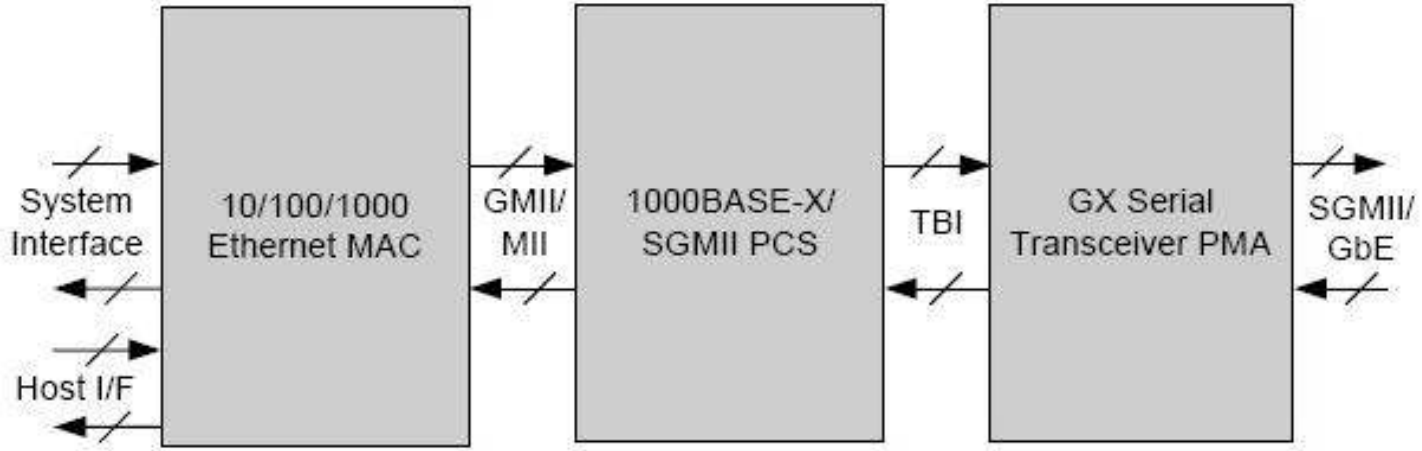
- DDR and DDR2 high-performance controller MegaCore functions
  - Full rate local interface: reduces latency and logic resources
  - ECC support: 1-bit correction and 2-bit detection

Note:

*Avoid placing **reset** or **exception** address at 0x0 due to self-check performed by High Performance memory controller on first 32 bytes of memory at boot-up time → use offset of 0x20 instead*

Triple-Speed Ethernet Megacore

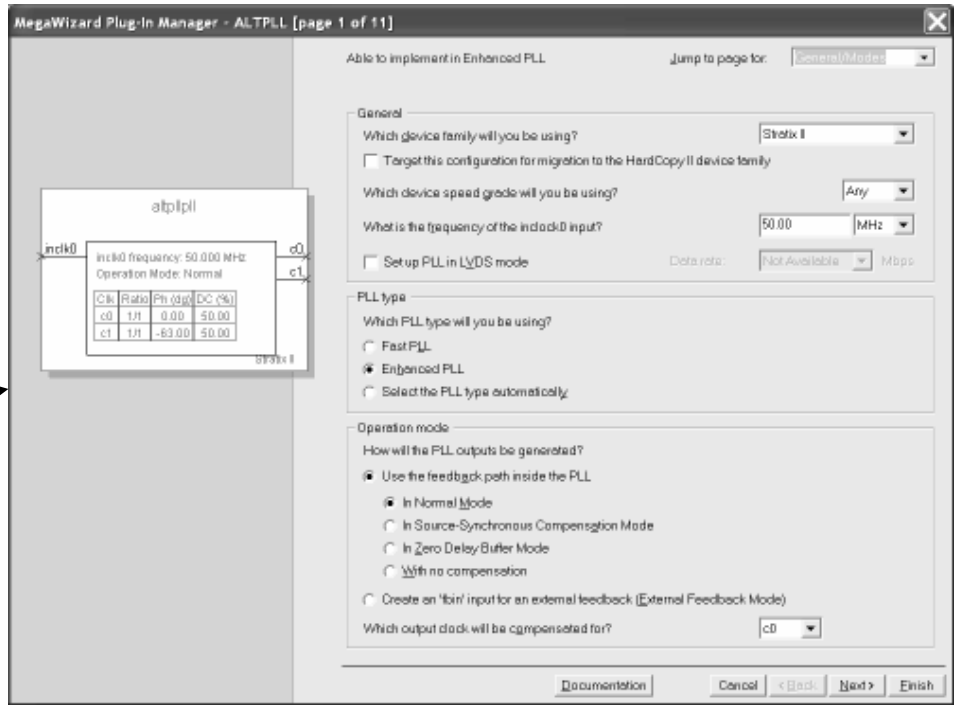
- Combines 10/100/1000 Ethernet media access controller (MAC), 1000BASE-X physical coding sub-layer (PCS), and 1000Base-X / SGMII physical attachment layer (PMA) functionality
- Available through Quartus II MegaWizard
  - ModelSim Testbench generated for you



Example Peripherals (cont.)

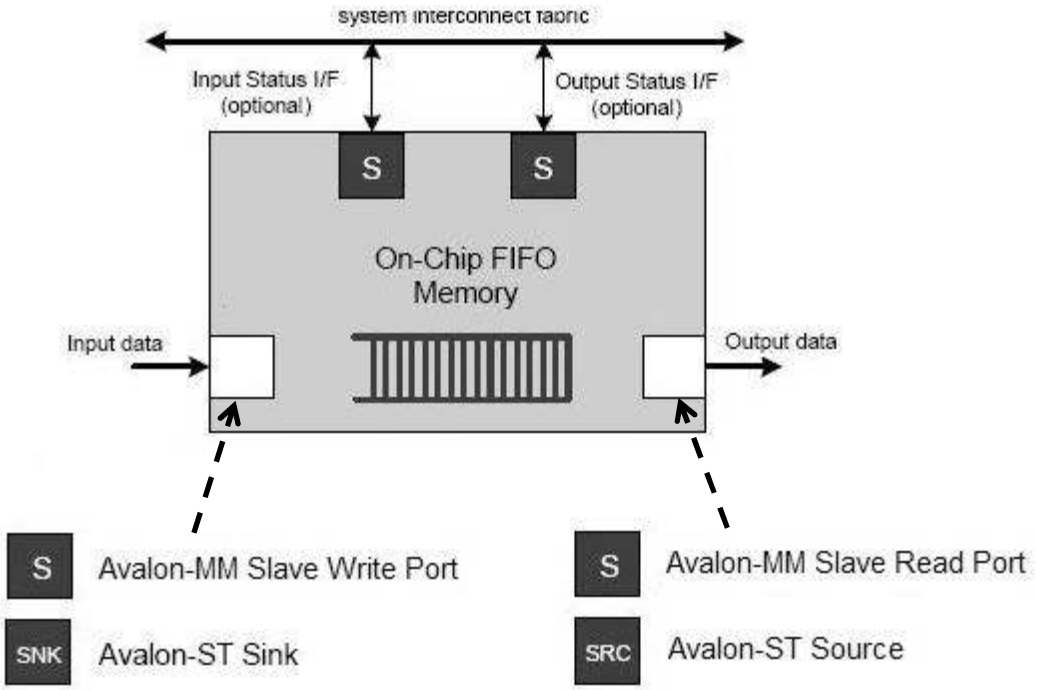
- **PLL Component**
  - Helps minimize amount of logic in top level Quartus II software project
  - Open **MegaWizard** inside component

Name	Source	MHz
clk	External	50.0
sys_clk	pll.c0	85.0
ssram_clk	pll.c1	85.0



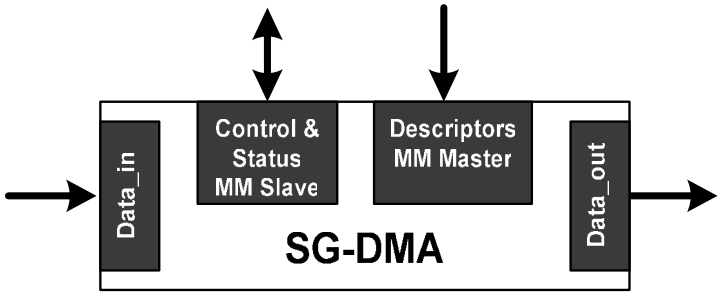
Example Peripherals (cont.)

- Configurable **On-Chip FIFO**
  - Used to buffer data and provide flow control in SOPC system
  - Single or double clocking
  - Connects to Avalon-MM or Avalon-ST components



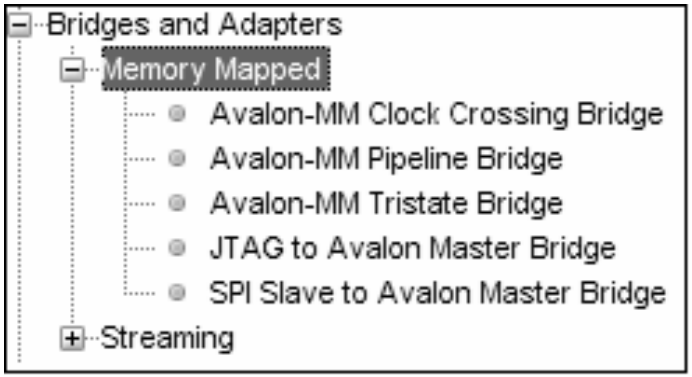
Example Peripherals (cont.)

- **Scatter Gather DMA (SG-DMA)**
  - Transfers and merges **non-contiguous memory to a continuous address space** or vice versa
    - Significant performance improvement over DMA peripheral
      - Reads a series of descriptors specifying data to be transferred and proceeds **w/o** further CPU intervention
      - Descriptor table can reside in on-chip or off-chip memory
      - Internal FIFO used to pipeline descriptors and hide read latency
      - Processor provides system control
  - Three different configurations
    - Memory to memory *(Avalon-MM to MM)*
    - Memory to stream *(Avalon-MM to ST)*
    - Stream to memory *(Avalon-ST to MM)*



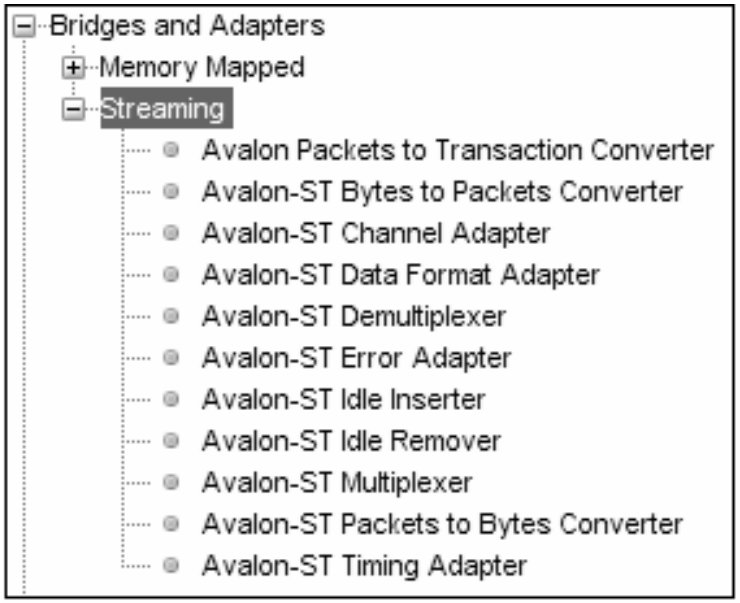
Example Peripherals (cont.)

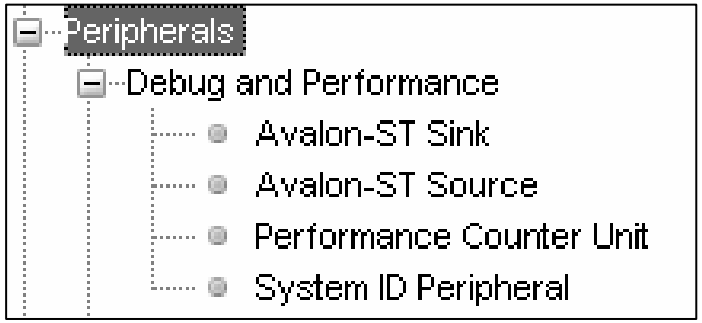
- **Avalon-MM Bridges**
  - **Clock Crossing**
    - For Buffered high-throughput clock domain crossing from Avalon-MM master to slave
  - **Pipelined**
    - Pinpoint pipelining of data path segments
      - Helps manage larger designs
  - **Tristate**
    - Defined by the presence of a bi-directional data port
  - **JTAG / Avalon Master Bridge**
  - **SPI / Avalon Master Bridge**



Avalon-ST Components

- For manipulating Avalon Streaming (Avalon-ST) data path signals *(discussed later)*





Some Example Avalon-ST Components

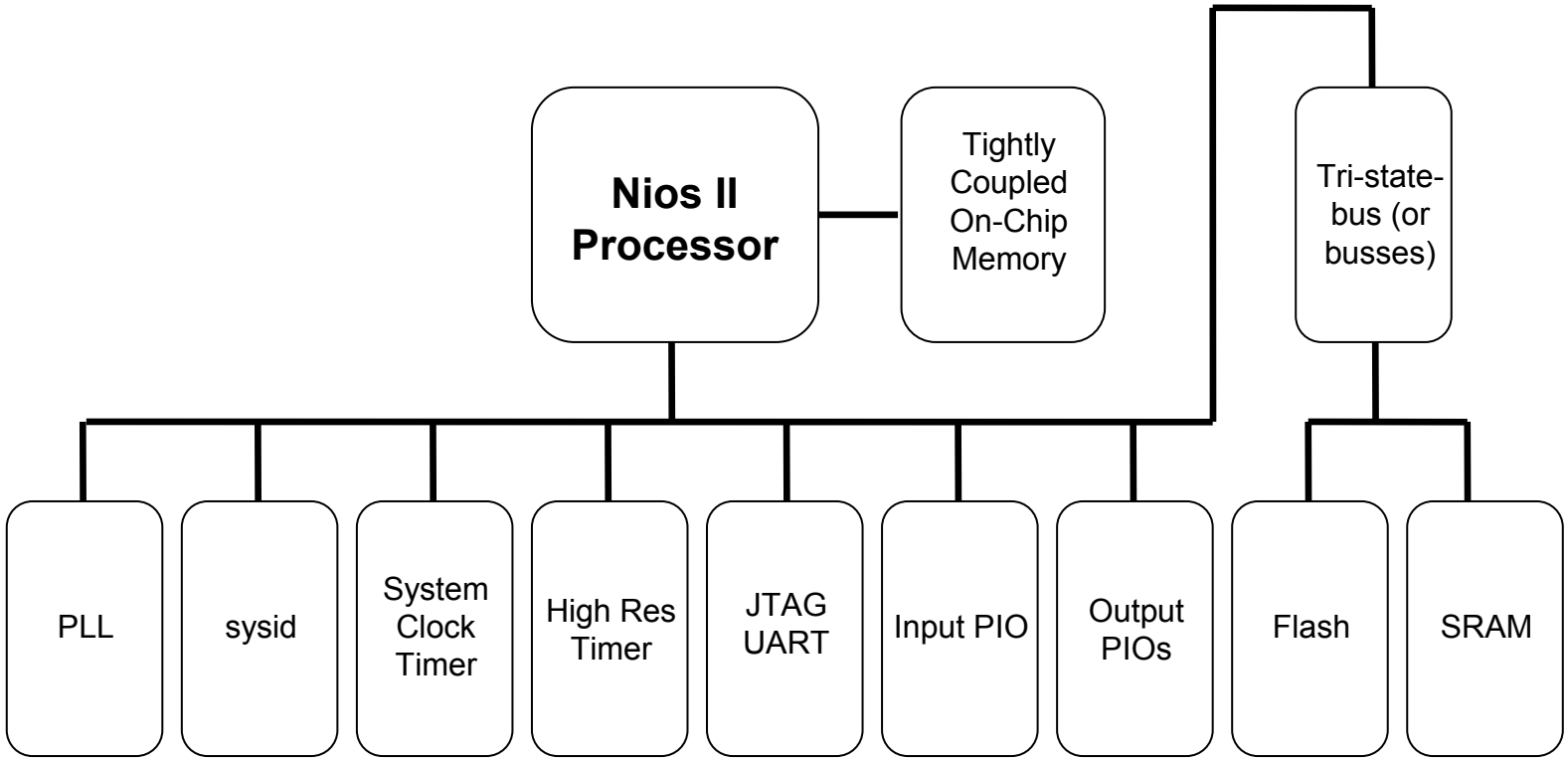
- Avalon-ST FIFO
- Avalon-ST JTAG Interface
- Avalon-ST SPI Interface
- Avalon-ST Bytes to Packets Converter
- Avalon-ST Bytes to Transactions Converter
- Avalon-ST PLI Interface

Lab 1
Build a Nios II Processor Design

45 min

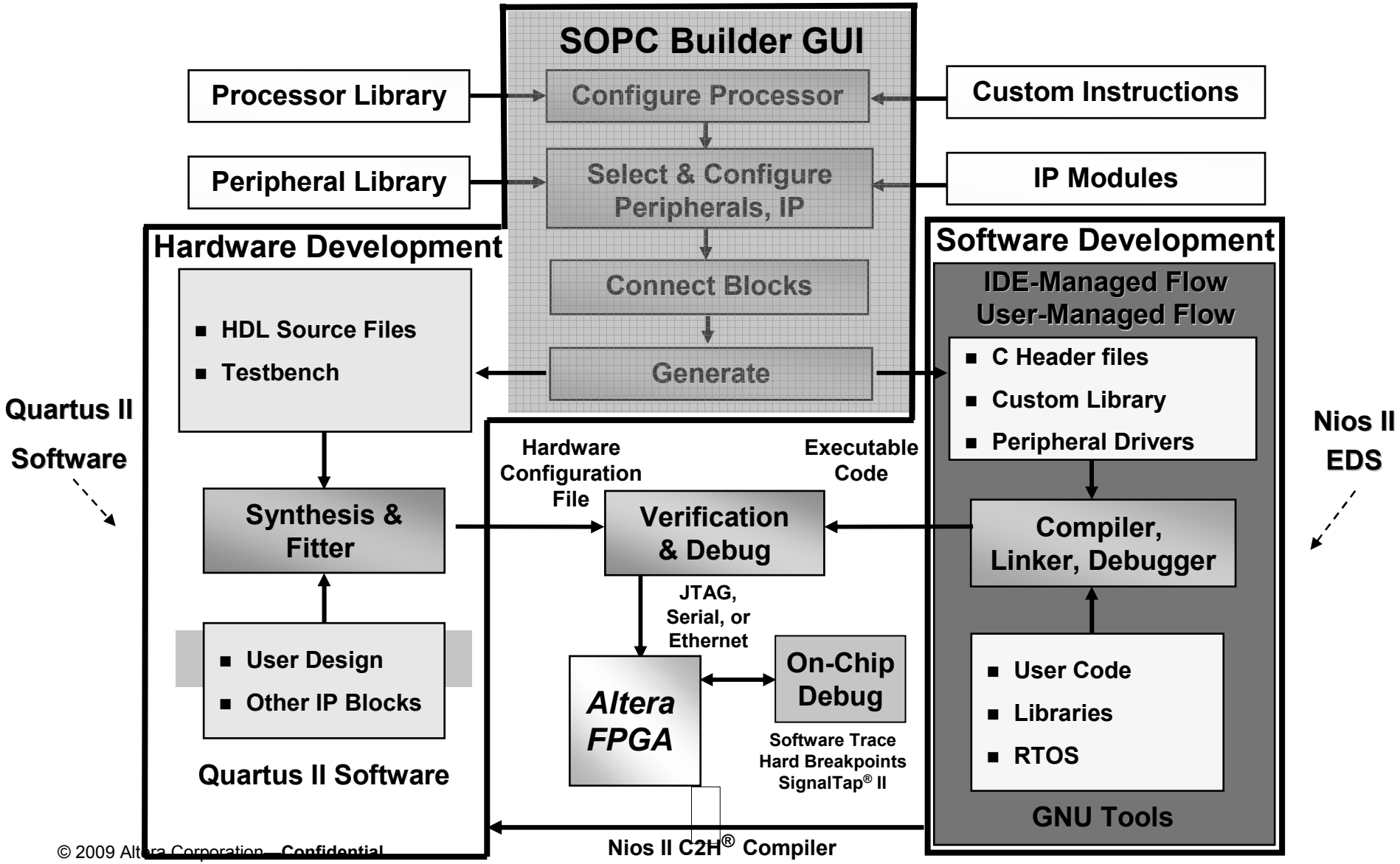
Outline of Lab 1 System

- You will build a system that resembles the following



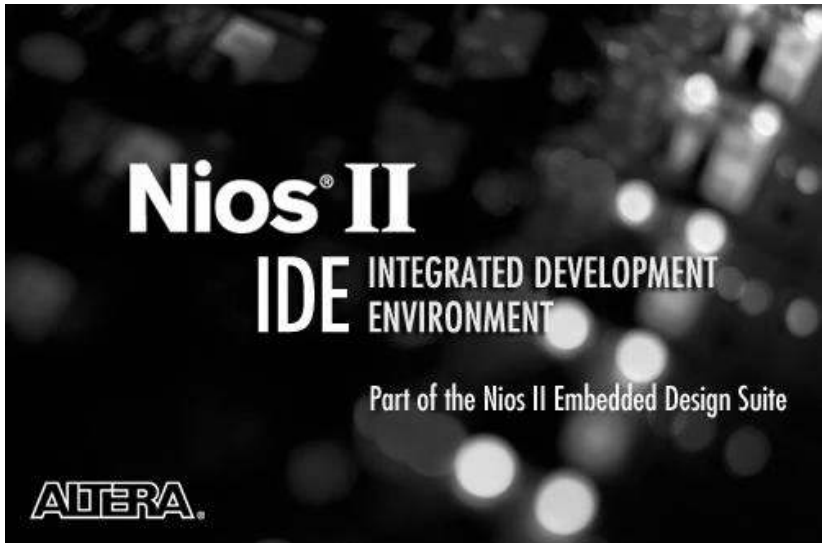
Nios II Processor - Software Development and System Testing

Nios II Processor System Design Flow



Nios II IDE (Integrated Development Environment)*

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



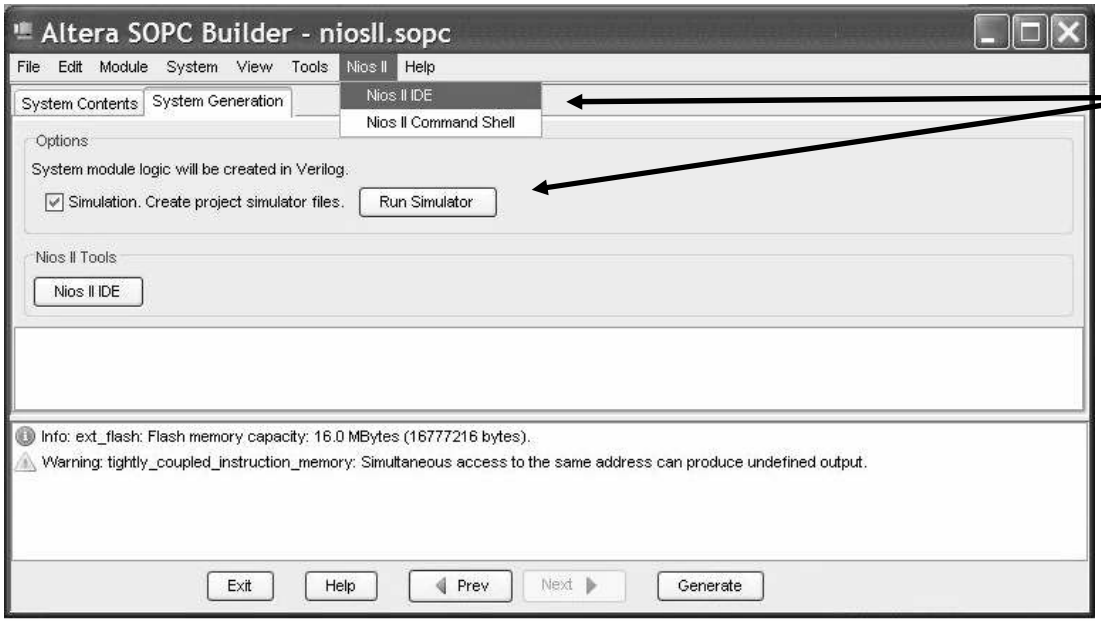
*** Based on Eclipse 3.2/CDT 3.1**

Command Line Tools

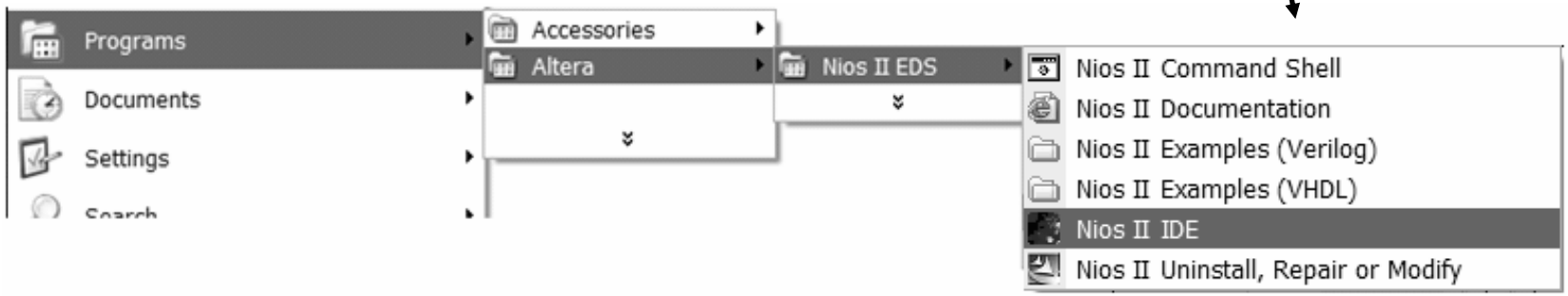
- New Tcl-based *user-managed* flow available for complete application and BSP development
  - Deep command-based control over software flow
  - Scripting capability for regression testing
  - Discussed in Chapters 1 and 2 of the “Nios II Software Development Handbook”

- ***BSP*** = *“Board Support Package”*

Opening the Tools

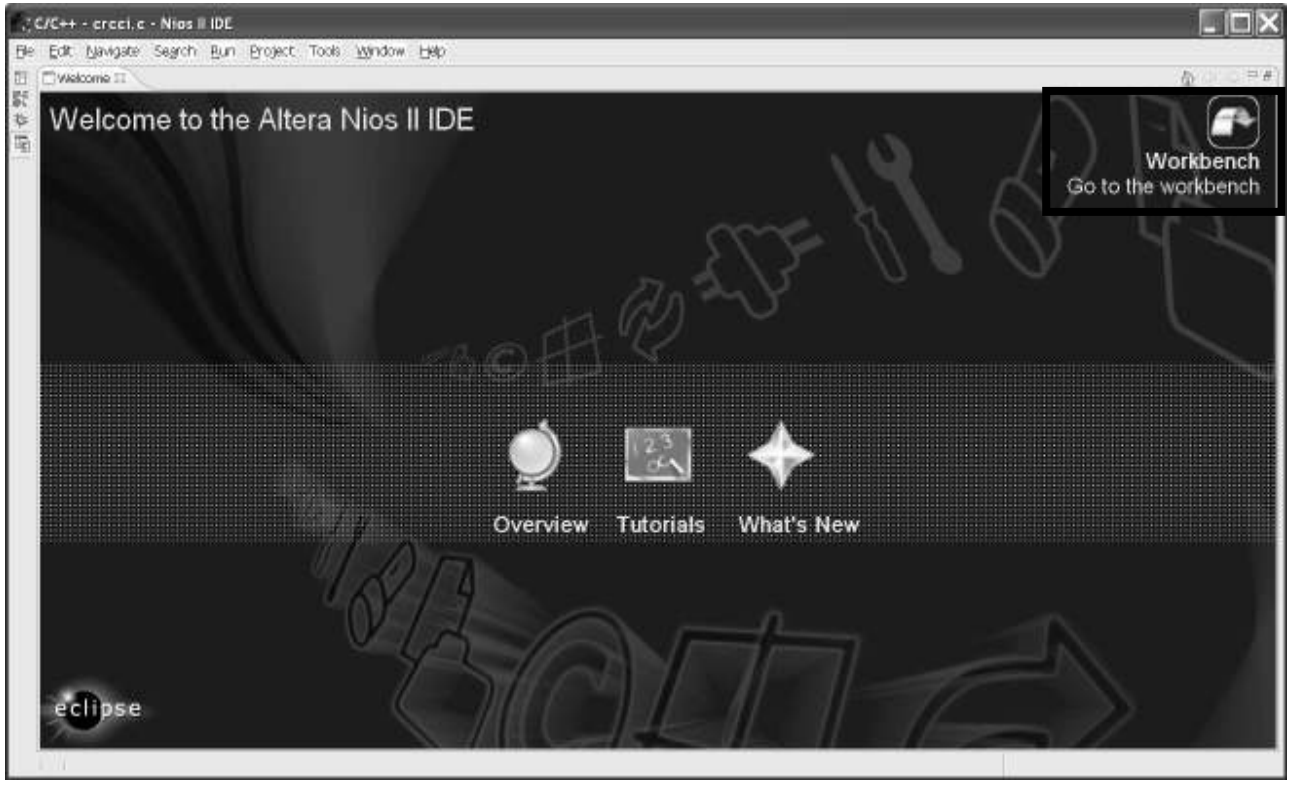


Launch the Nios II IDE from the **SOPC Builder** or from the **Windows Start menu**

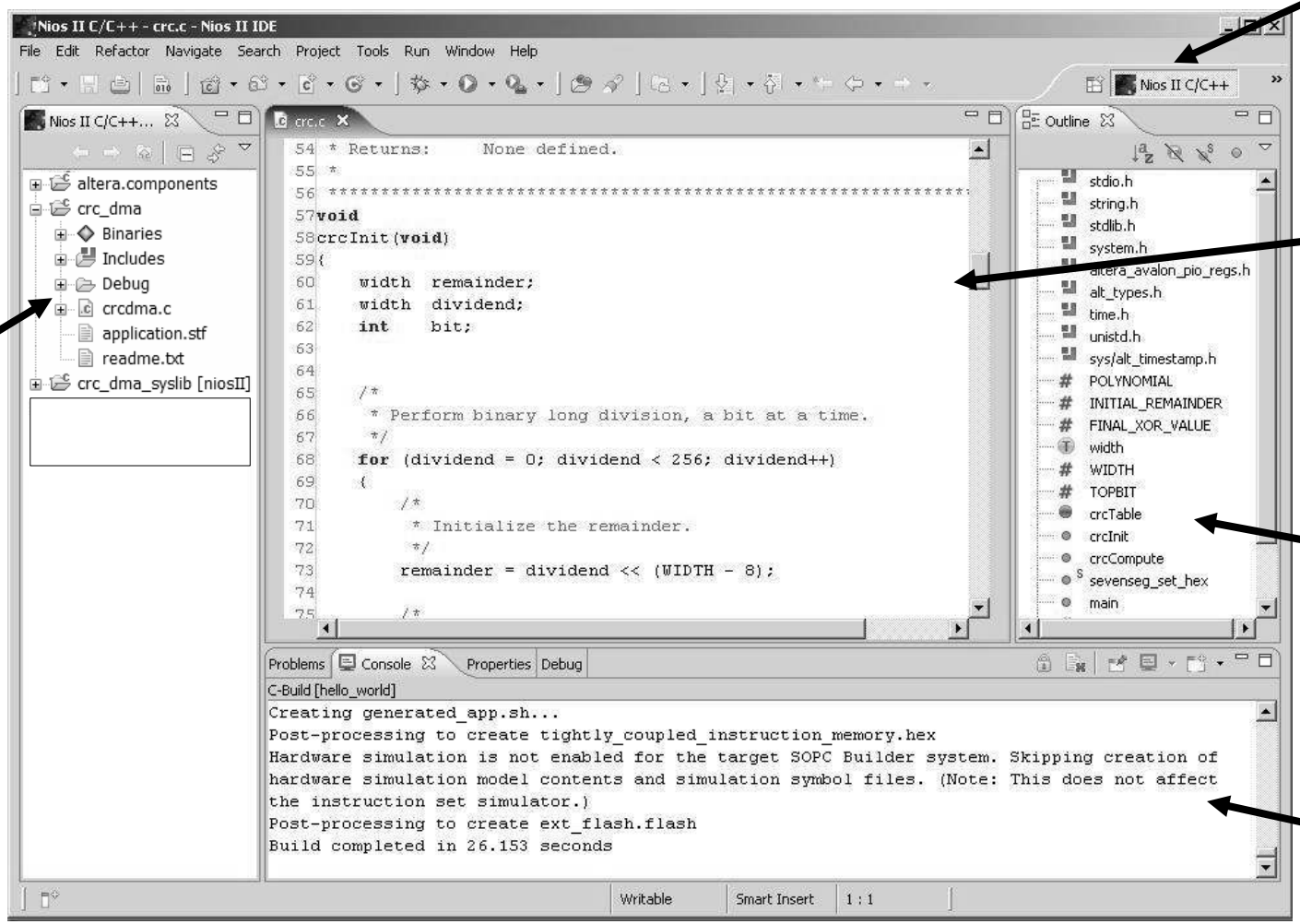


Nios II IDE Welcome Page

- Get A Tool Overview
- Access Tutorials
- Check Out New Features
- Open IDE Workbench



Nios II IDE Workbench



List of Open Projects

Select Perspective

Defaults to **Nios II C/C++**

File Viewer Window

(for C code, C++, and assembly*)

Outline View

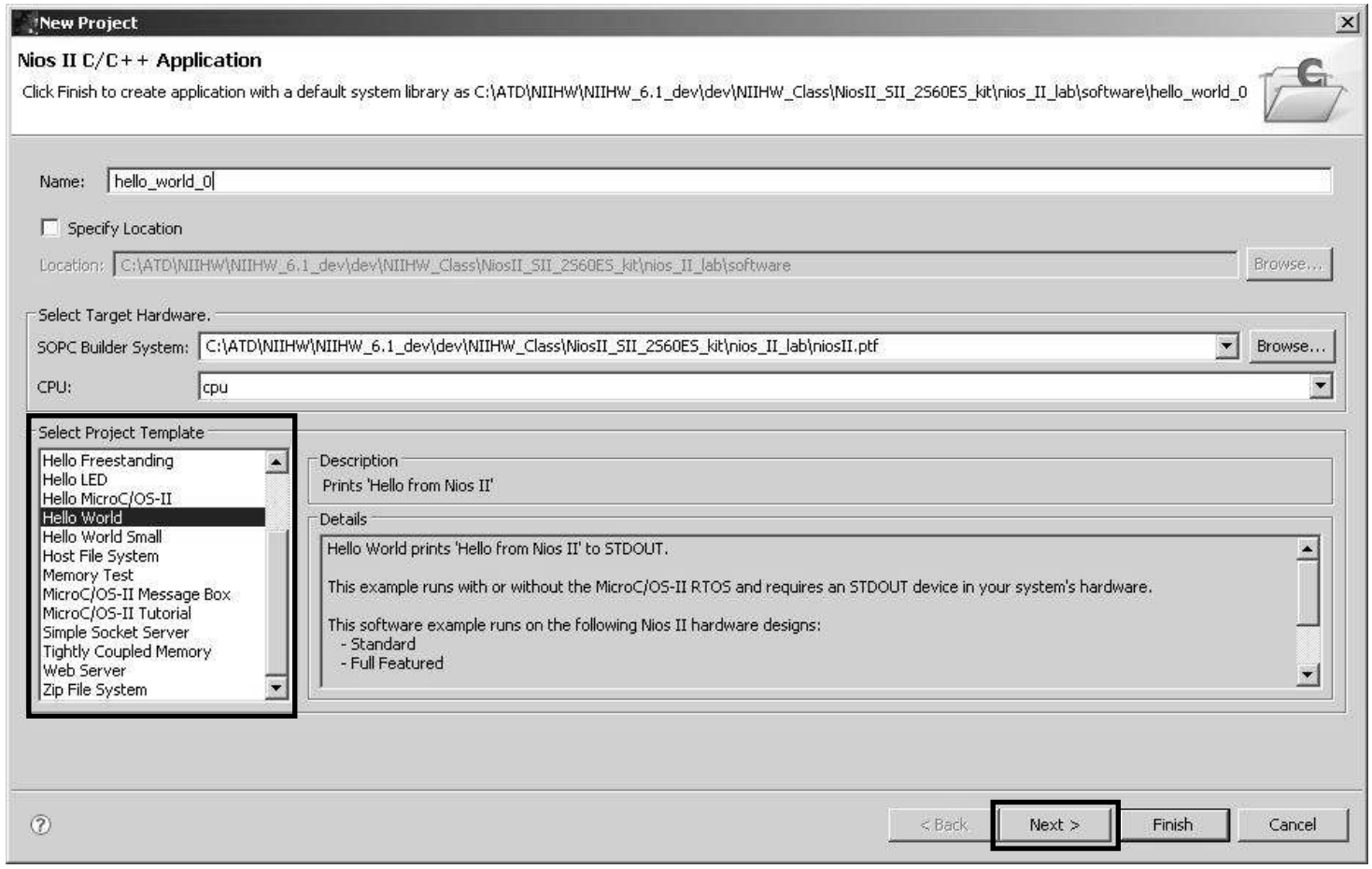
(view and/or open funcs, enums, classes, unions, structs, typedefs, etc.)

Terminal window

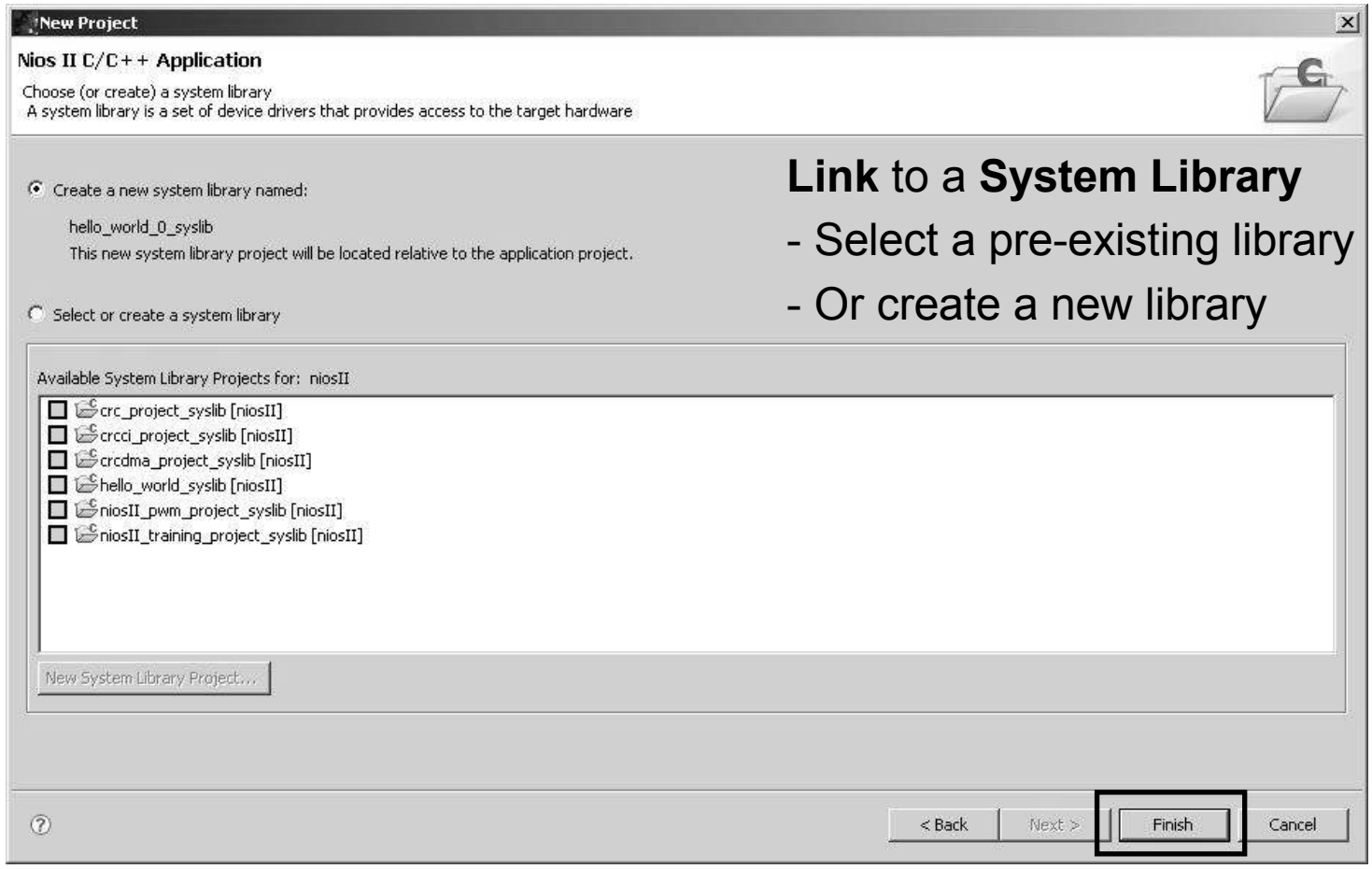
• ***Note:*** *C++ files must have extension* ***.cpp***
In-line assembly code offset by ***asm();***

Creating a C/C++ Application

File > New > Nios II C/C++ Application



Creating a C/C++ Application



Link to a **System Library**

- Select a pre-existing library
- Or create a new library

This Creates Two SW Project Folders
- Application *and* System Library

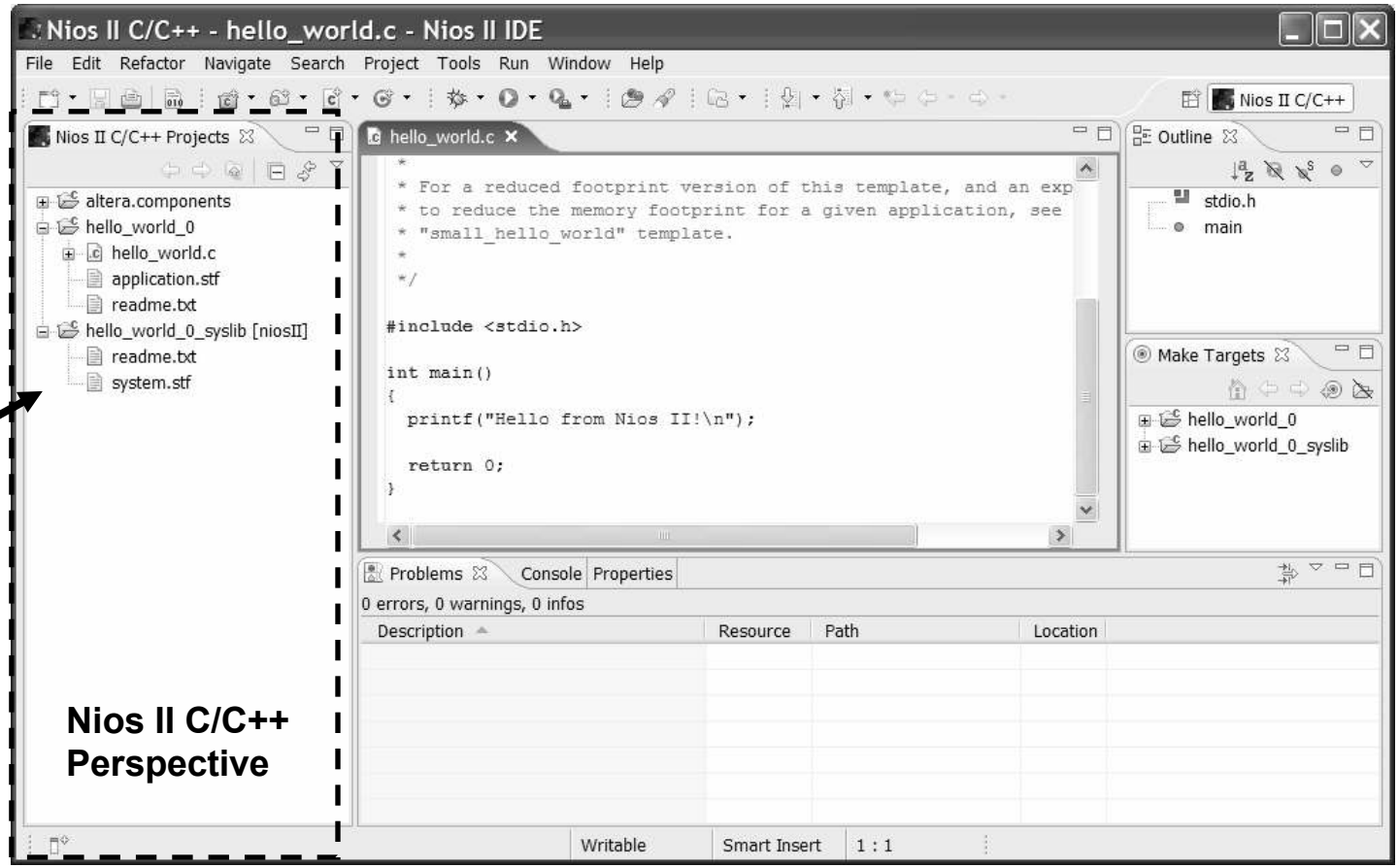
Application Project

- contains application source code

System Library Project

- contains system header file, etc.

- links to device driver source code

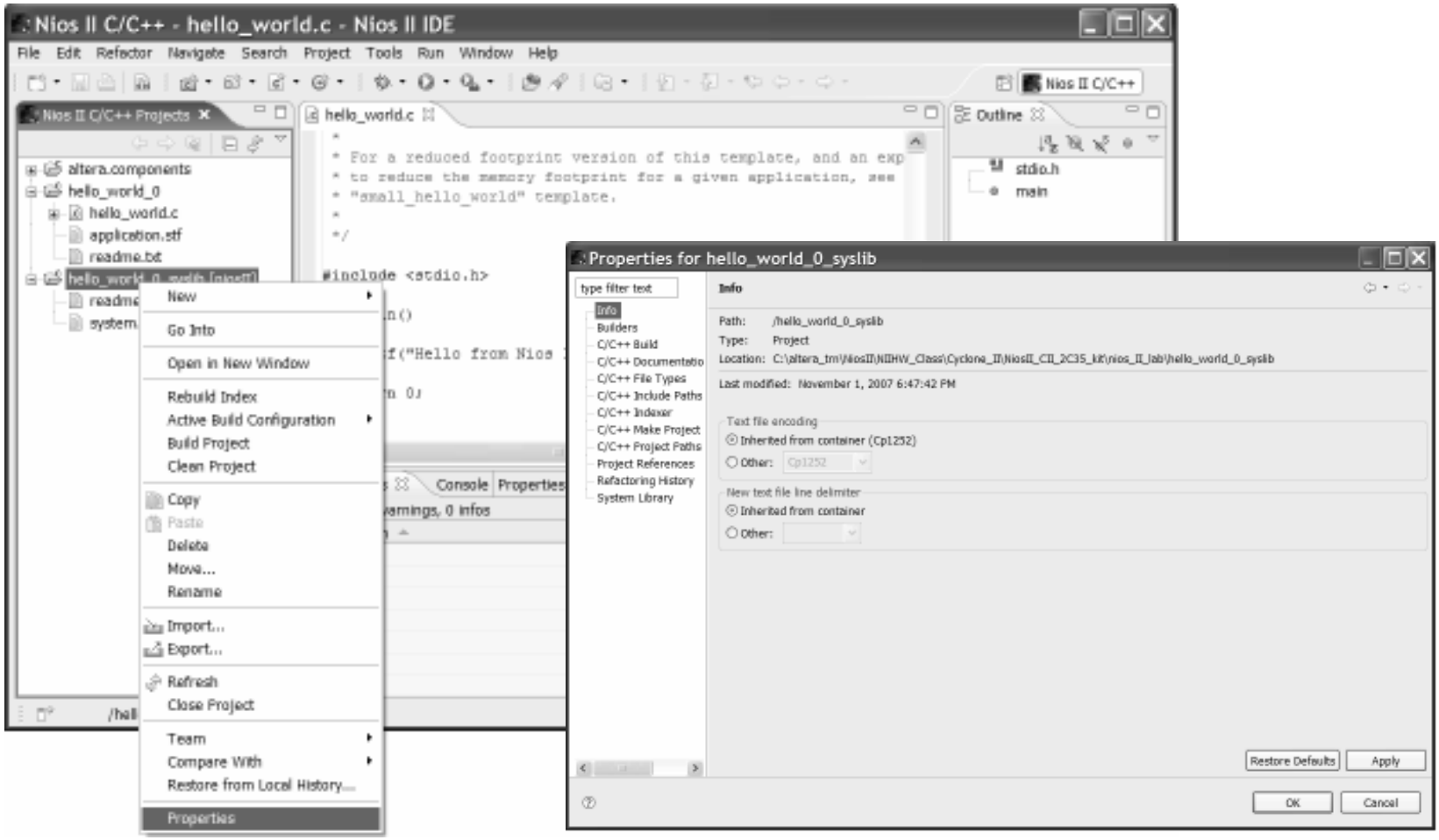


Application and **System Library Projects**

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

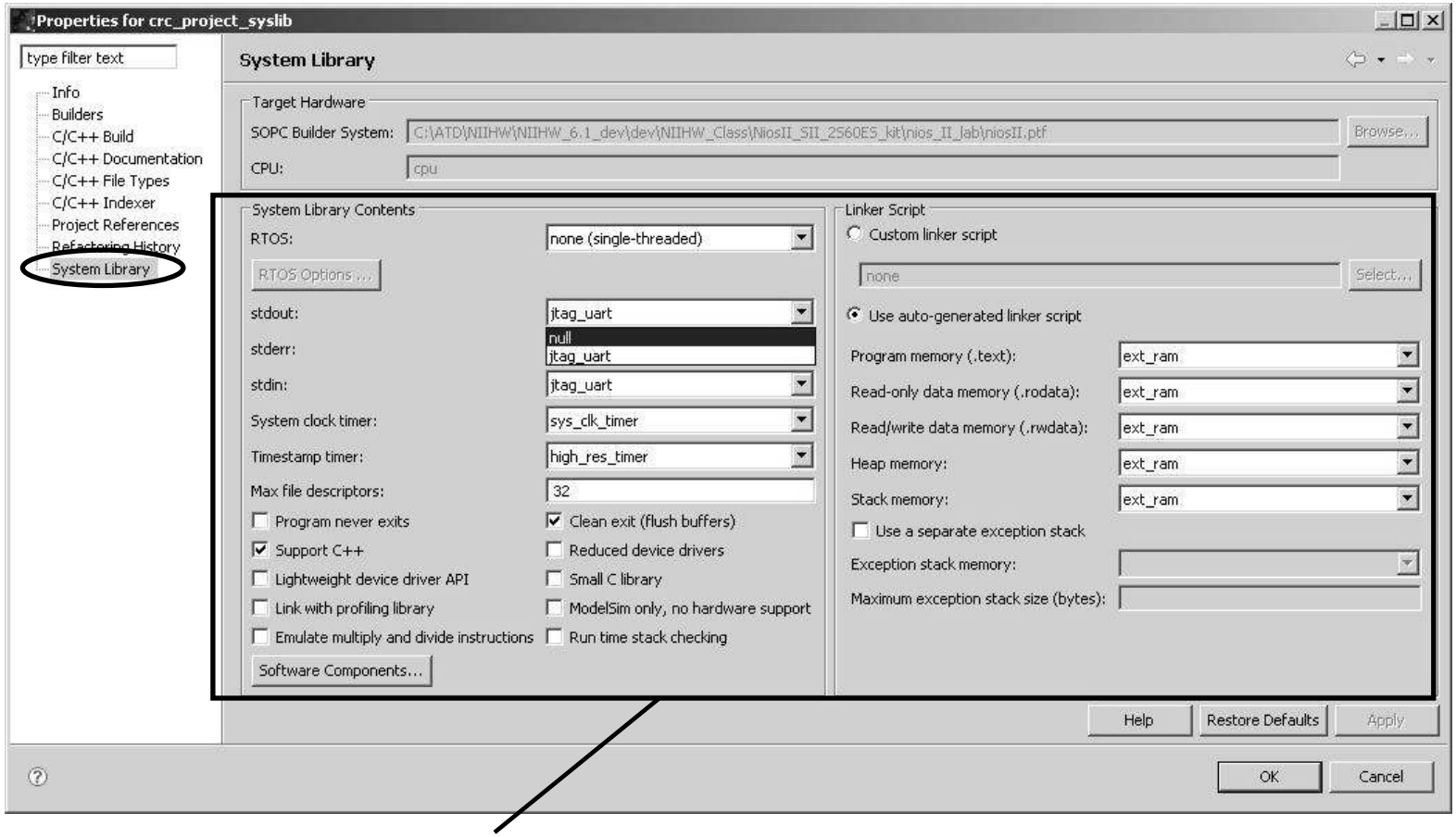
Setting Project Properties

- **Application** and **System Library** both have project **Properties** pages



System Library Project Properties

(and specifying stdio devices)

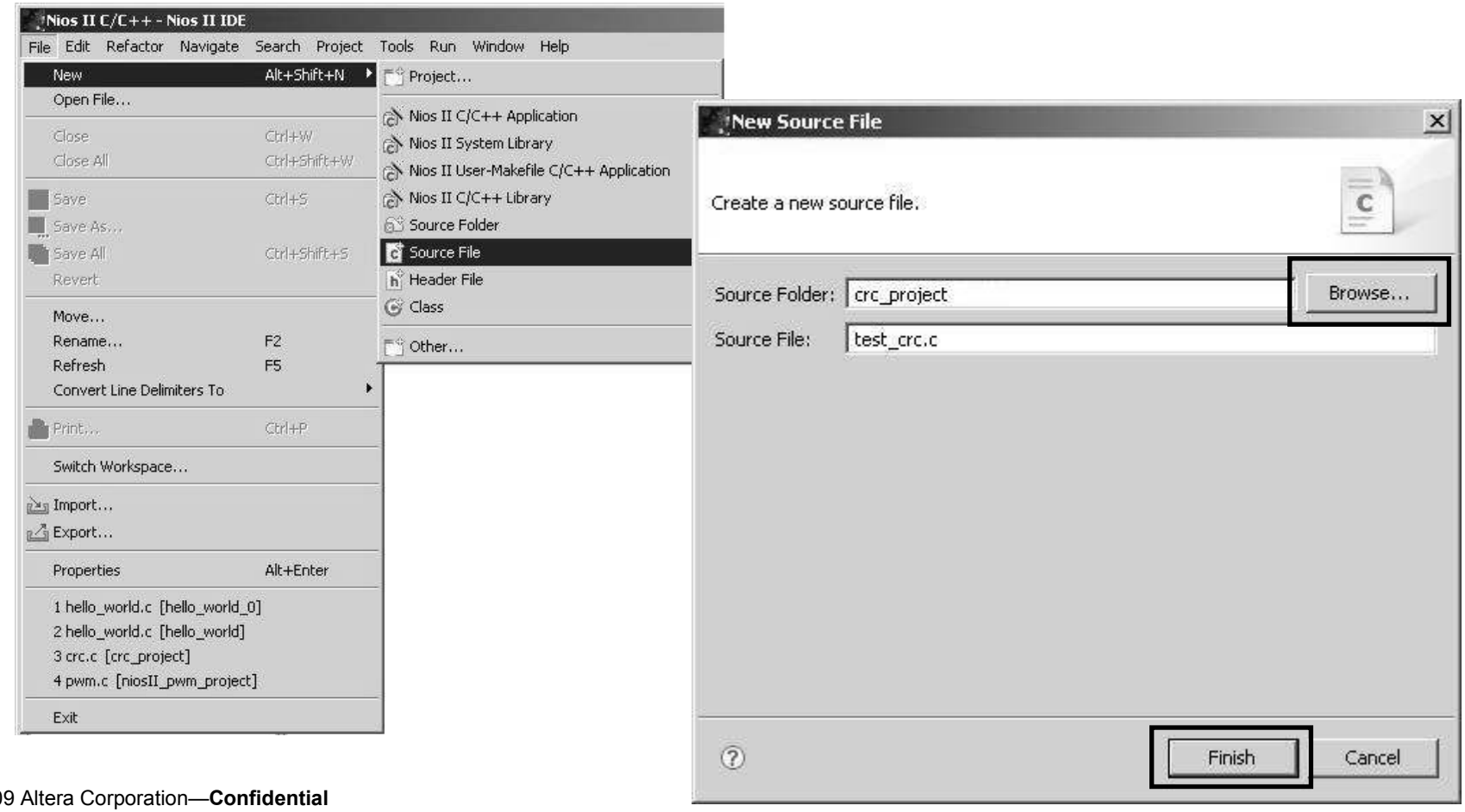


Set properties for the System Library Project

(Including, partitioning the memory map)

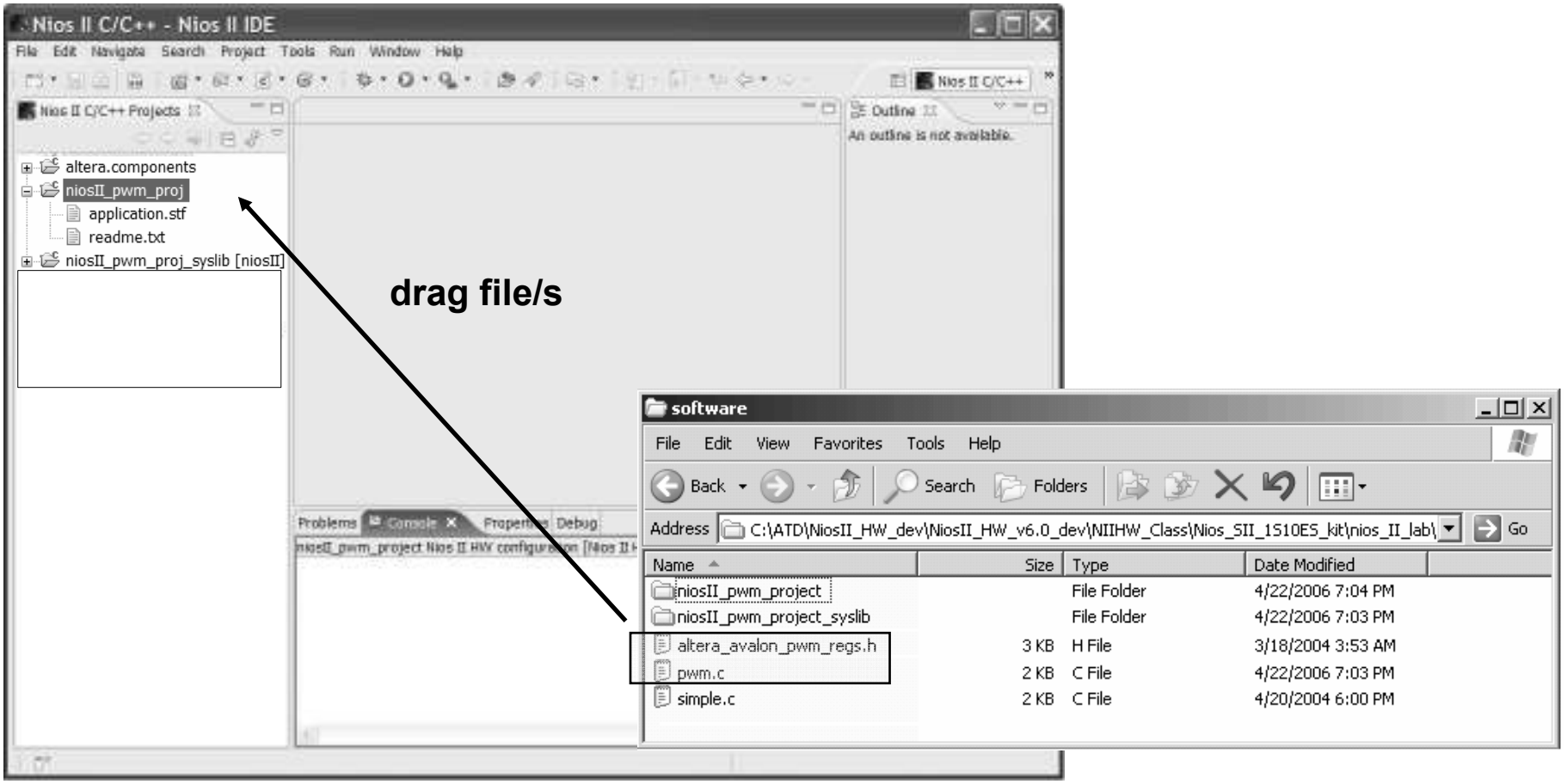
Creating a New Source File

- From within the Nios II IDE
  - Specify **Application Project** folder and **<file_name>.c**



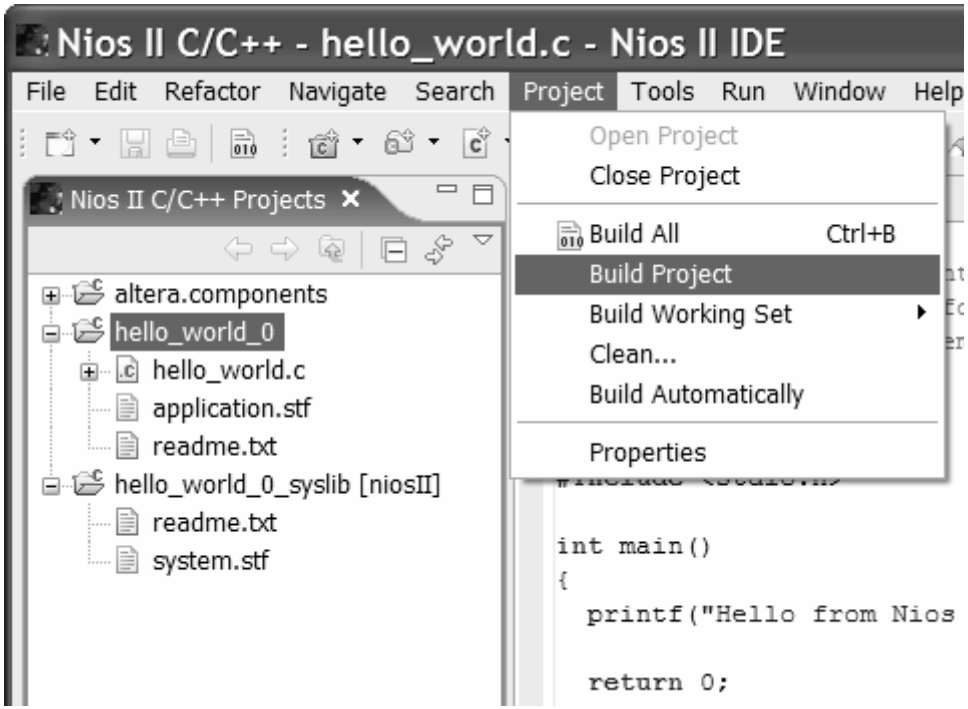
Adding Source Files to a Project

- **Import** files or just **drag** them into Application Project
  - Right-Click and **Refresh** to update project if necessary



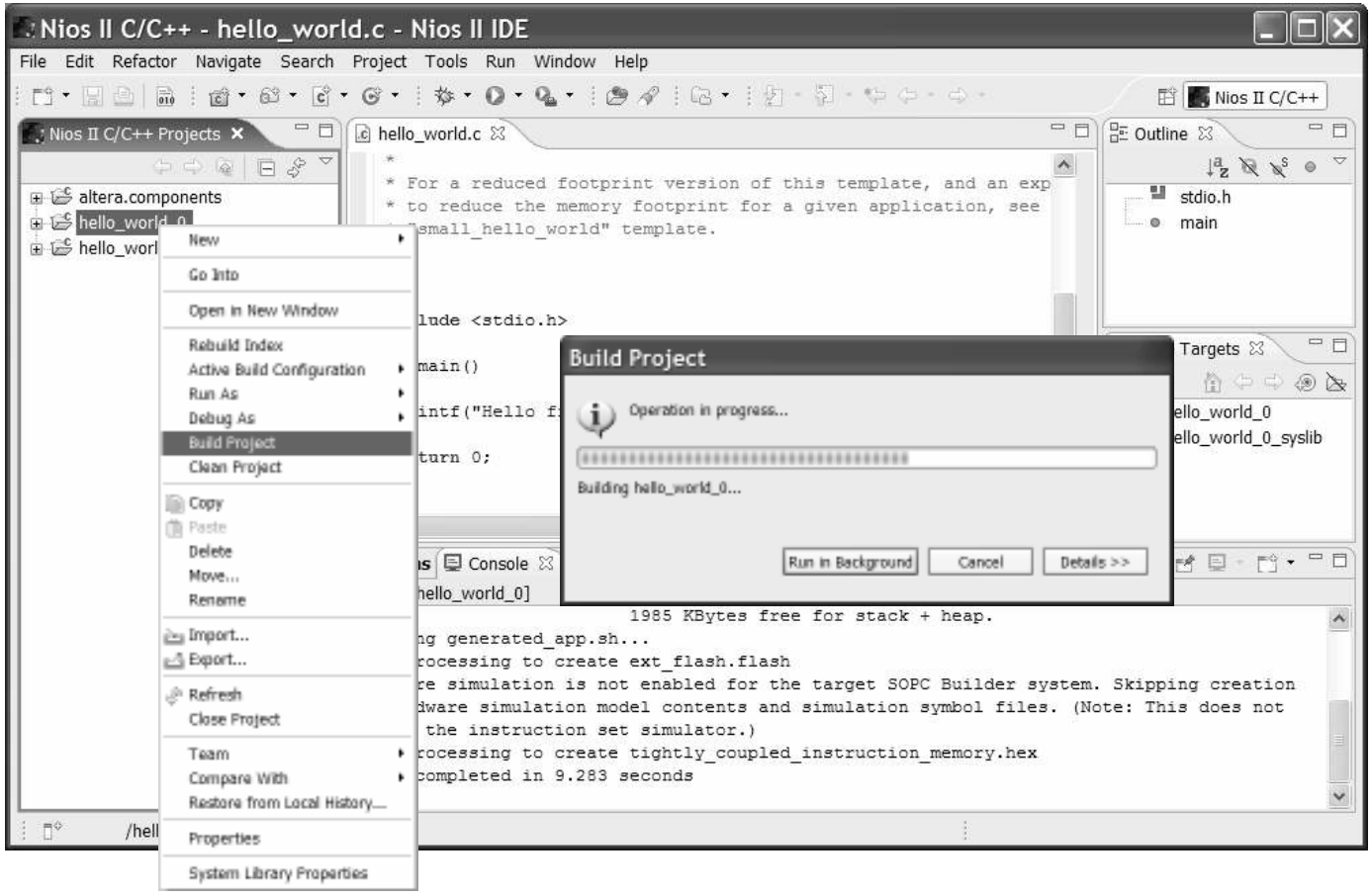
Software Compilation

- To compile a software application, highlight project, right-click, and select **Build Project,** or go to **Projects** menu
  - Compiles syslib project first on initial build
  - Evaluates makefile for compiling application code



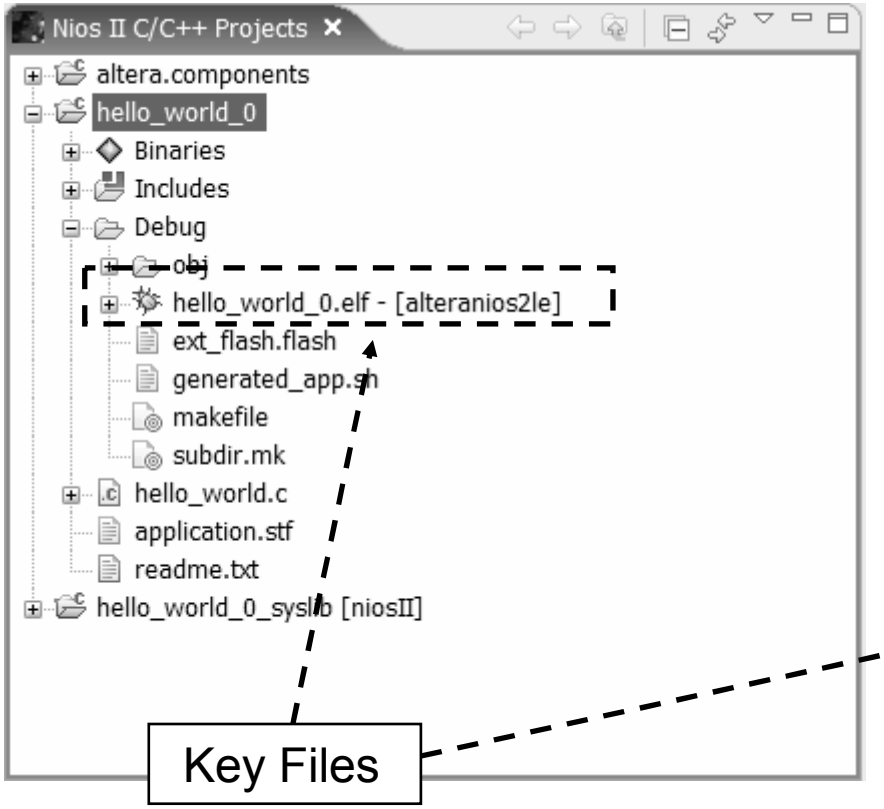
Can Build In Background

- Option for Improved Productivity
  - Carry on other activities in foreground in the Nios II IDE

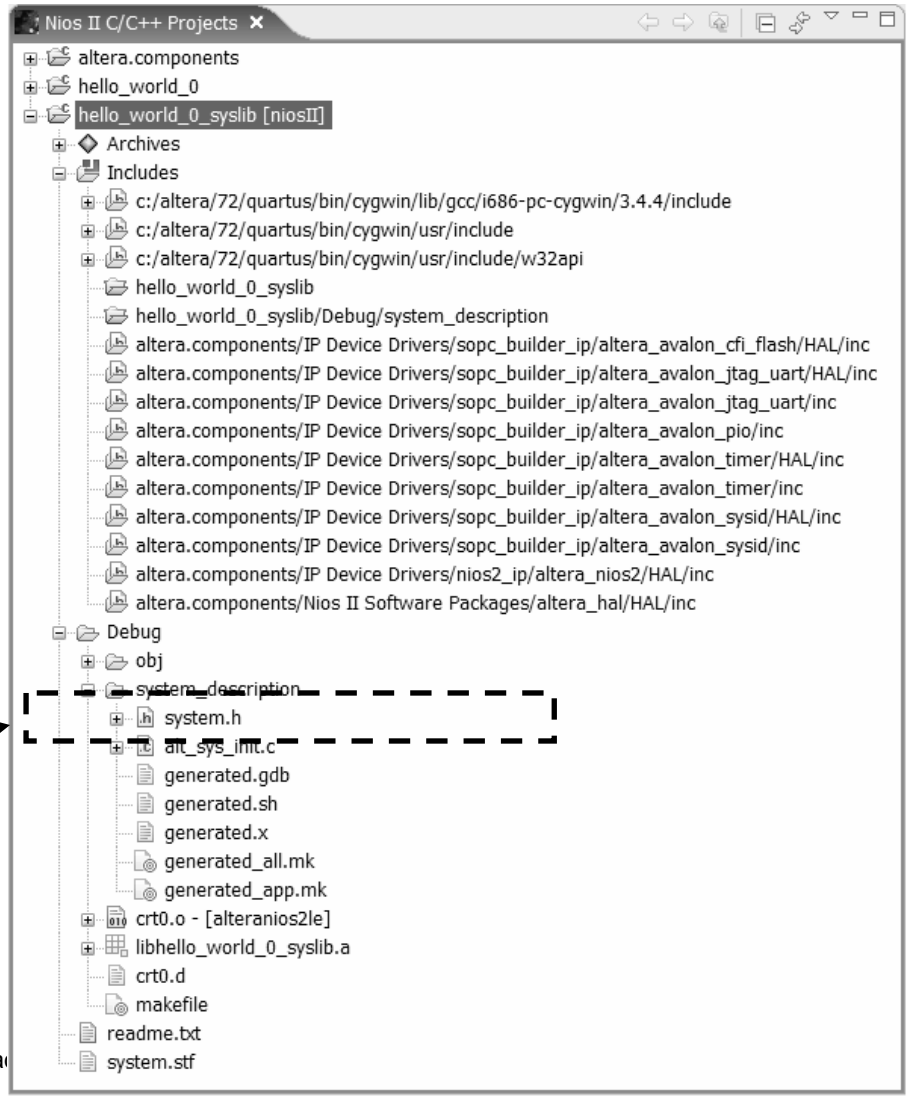


Directory Structure After Build

- **Application Project**

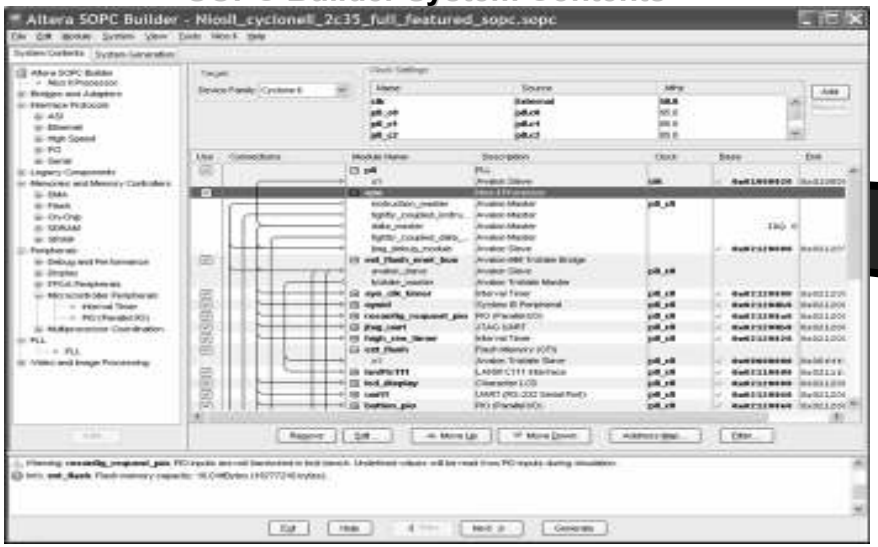


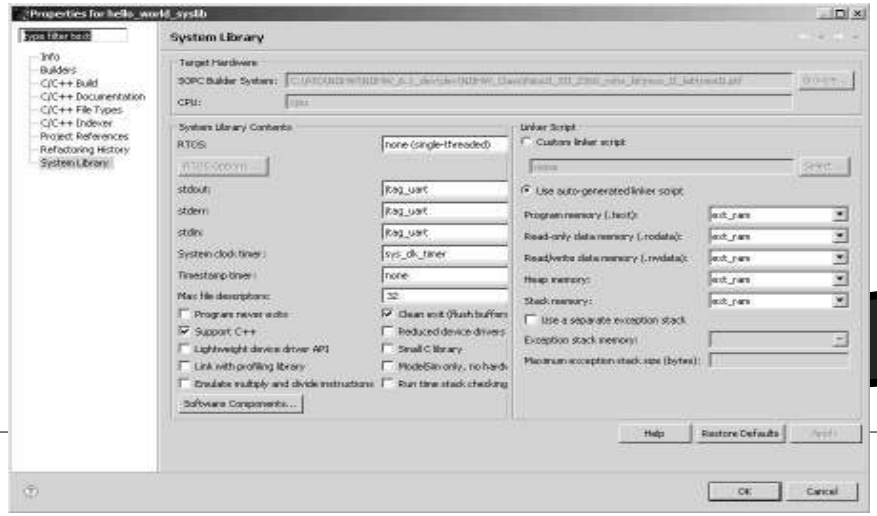
- **System Library Project**



System Header File

SOPC Builder System Contents





System Library Settings

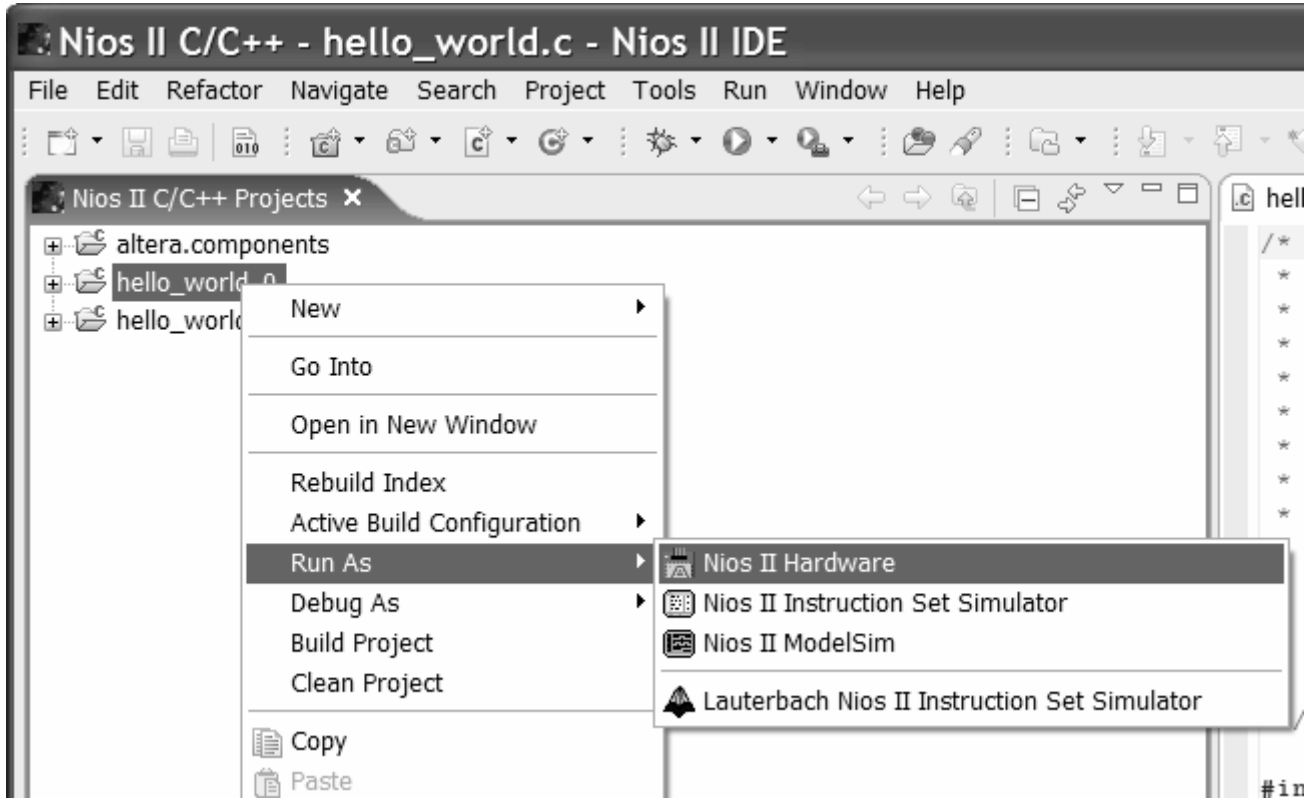
system.h

Contains all symbolic C-language definitions for the peripherals in your hardware system, plus more…

Software Run & Debug

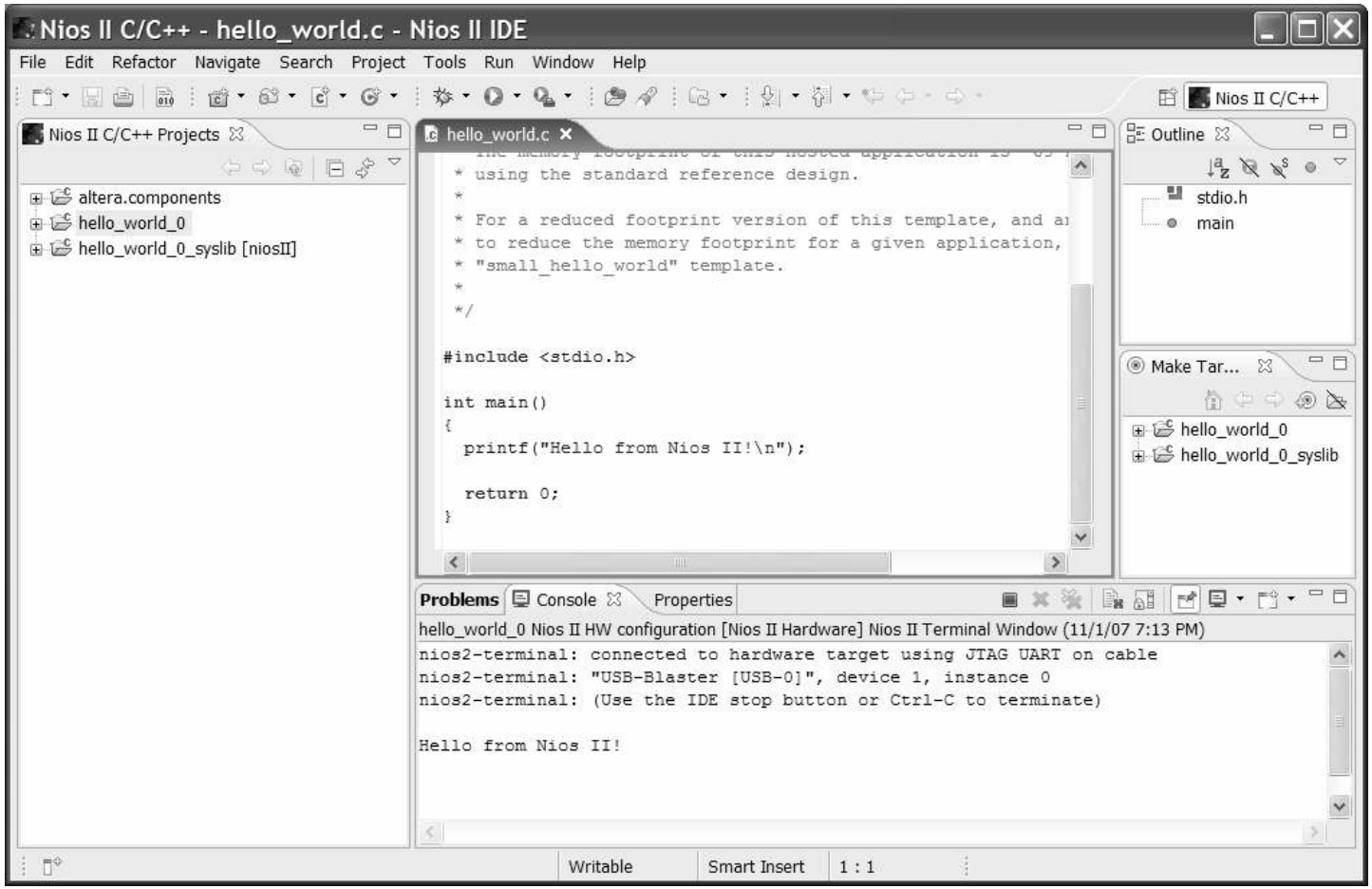
Running Code On A Target

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



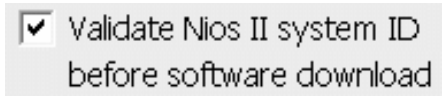
Running Code On A Target

- Download messages and stdio appear in console window

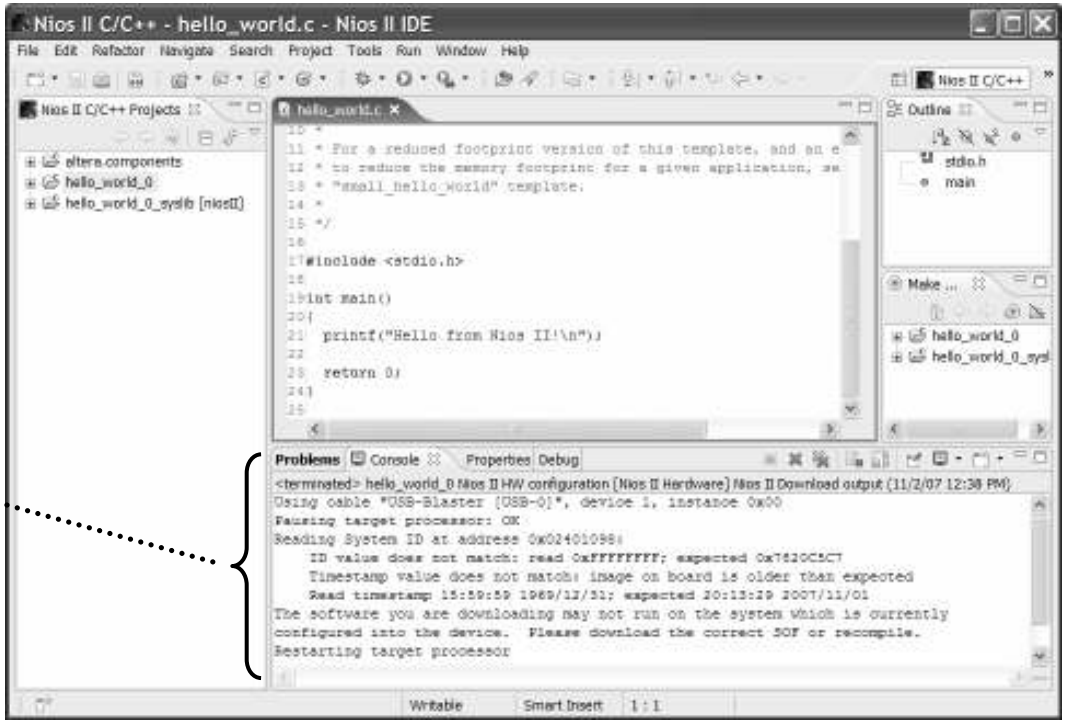


System ID Peripheral Checked at Run Time

- Nios II IDE computes expected System ID peripheral values from PTF file (ie. checks PTF vs. SOF)
  - If computed ID values do not match System ID variables stored on the target board then an error is flagged
  - Generally, to fix this the hardware must be recompiled
  - To disable this option see **Run > Run** main page

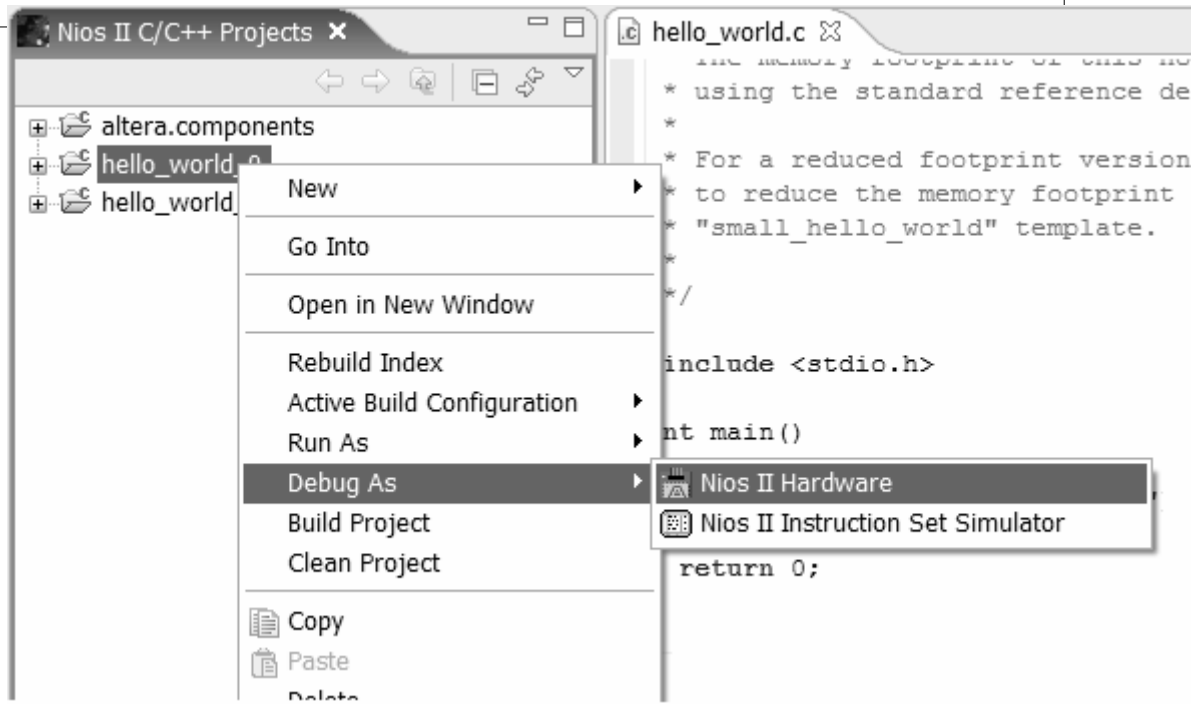


System ID peripheral error messages



Nios II IDE JTAG Debugger

- **Requirements**
  - Must have JTAG Debug Core enabled in CPU



Nios II IDE Debug Perspective

Re-start **Debugger**

Re-**Run** Program

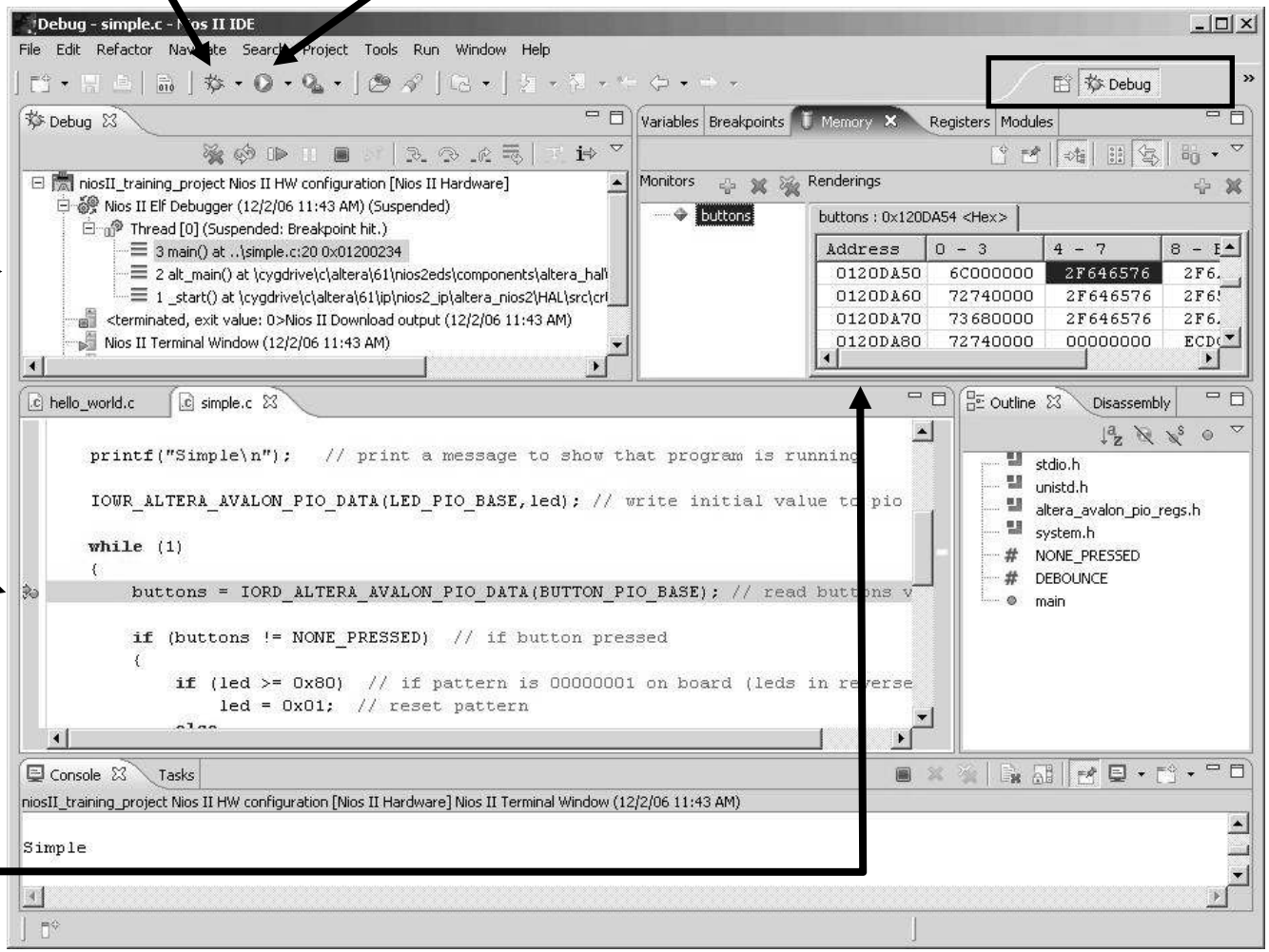
Basic Debug

- Run Controls
- Stack View
- Active Debug Sessions

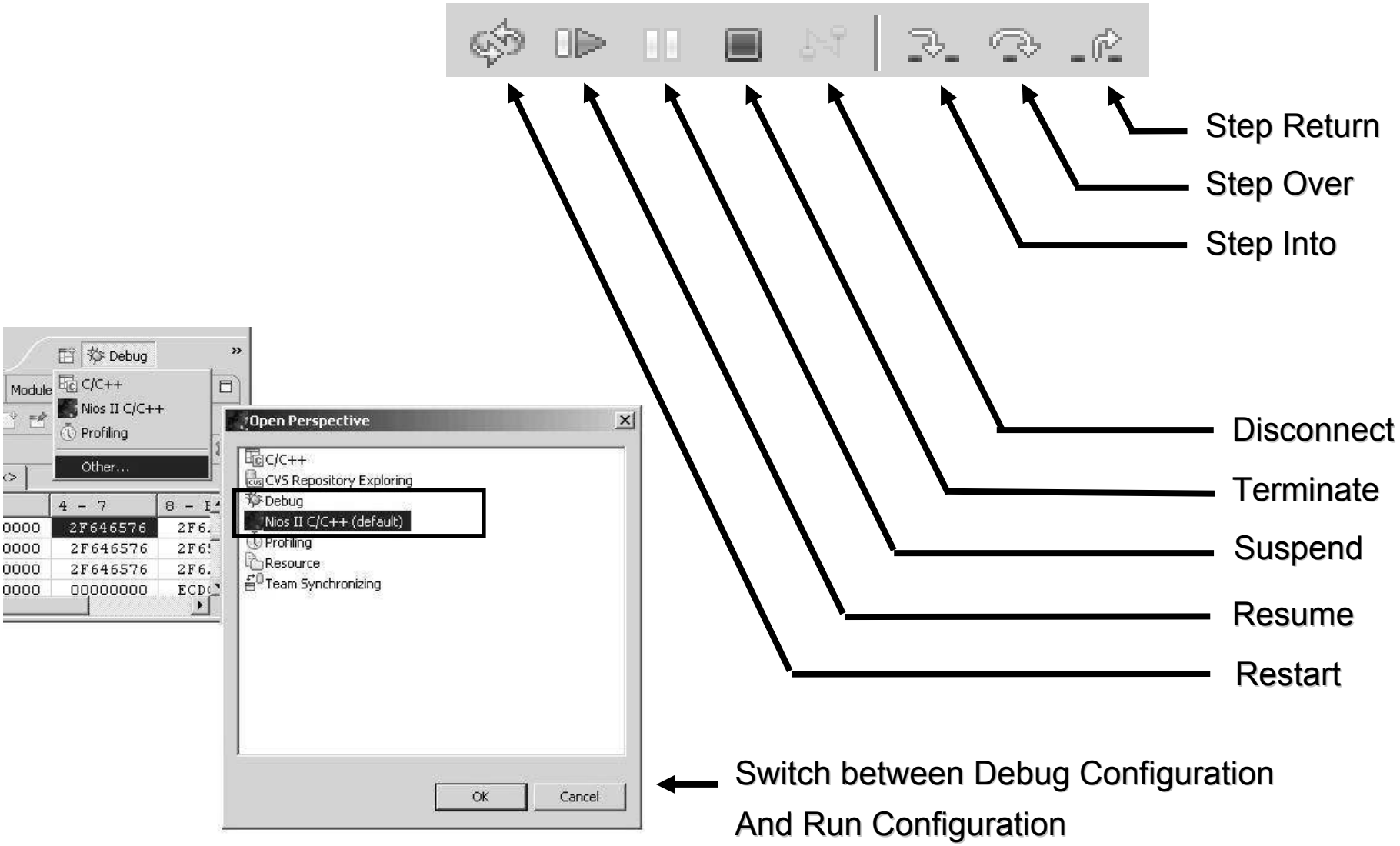
Double-click to add breakpoints

Memory View

- Variables
- Registers
- Signals



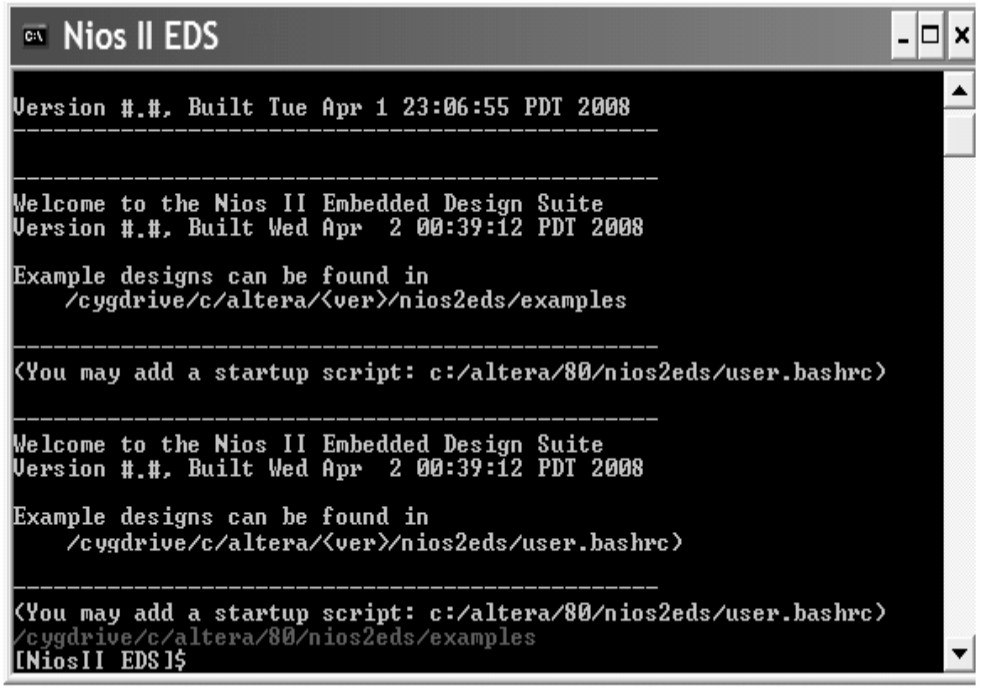
Nios II IDE Debugger Controls



Nios II Command Shell

- Used to support shell-driven flow
  - as well as other general commands
- Can launch terminal to interface to JTAG UART
- Compile and Run code
- Create scripts to control build process
- Provides UNIX-like interface

- Open from **Start Menu** or **SOPC Builder**



Also Available for Nios II Processor

- RTOS Support
  - Various operating systems available
    - MicroC/OS-II developer's license included with kit!
      - License required to ship products
- Middleware Support
  - Protocol stacks, file systems, graphics libraries, etc.
- Various Built-In Software Components
  - ROZIPFS, TCP/IP Stack, Host-Based File System
    - Interniche TCP/IP stack included with kit (small licensing fee)
- Different Debugger and Compiler tool options

- See www.altera.com > Embedded Processors > Embedded Software Partners
- *See also* ***www.niosforum.com***
- *And* ***www.nioswiki.jot.com***

Altera / Third Party Tool Choices

- Debuggers

IDE/Debuggers

Company	Product	Supported Debug Cable					
		Altera		FS2		Lauterbach	
		ByteBlaster™ II	USB-Blaster™ *(1)*	ISA-NIOSII	ISA-NIOSII/T	Power Debug	Power Trace
Altera	Nios II IDE *(1)*	✓	✓	✓	✓	-	-
First Silicon Solutions (FS2)	Nios II IDE Enhancements	✓	✓	✓	✓	-	-
Lauterbach	TRACE32-PowerView	-	-	-	-	✓	✓
Mentor Graphics®		✓	✓	✓	-	-	-

- Compilers

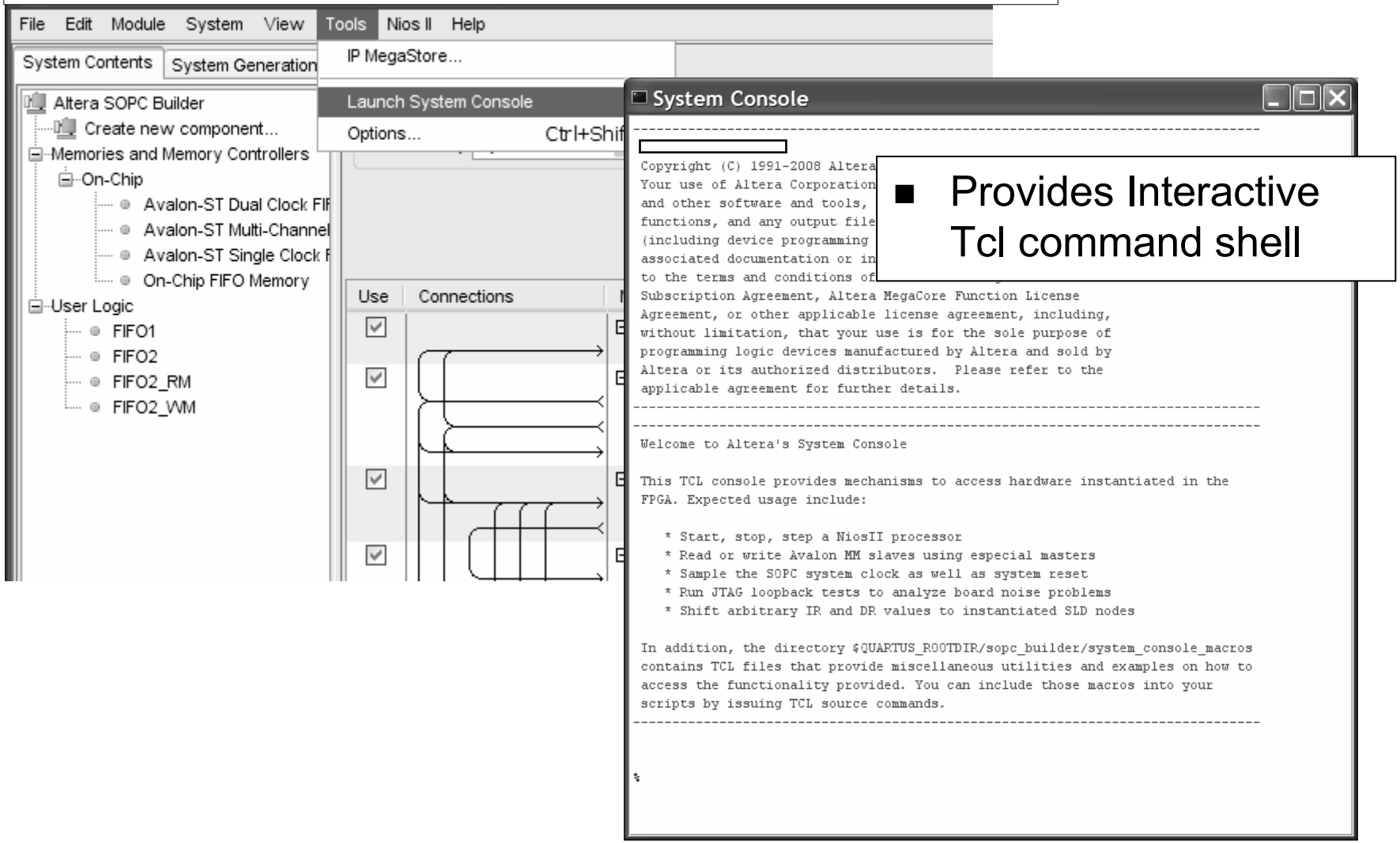
Compilers

Company	Product	Description
Altera	GCC compiler *(1)*	Standard GNU compiler for the Nios II processor.
Altium	TASKING VX-toolset	Optimizing C compiler, assembler, linker, and locator. Supports Eclipse-based Nios II IDE with plug-in.

http://www.altera.com/products/ip/processors/nios2/tools/ide/ni2-ide.html

Board Bring-Up Tools

SOPC Builder “System Console”



- Provides Interactive Tcl command shell

Command Line Interface

- Available from Nios II Command Shell
  - Provides alternate interface to system-console utilities

To Launch, type:

- `system-console --cli`

Recommendation:

*Run **scripts** from here, not System Console because you can CTRL-C them easily if they should hang*

Provides Host Target Communications

- Access to FPGA design at runtime
  - Via Nios II Processor, JTAG Bridge, or PLI (with Modelsim)
  - Supports an Avalon-MM or ST Master from a host PC
  - Board bring-up utilities for debug, diagnostics, and configuration
  - Based on Host-Target communications architecture
    - Physical transport mechanism independent\
- Via System Service Layer
  - ie. Services to high level test programs and applications
  - Current Services:
    - Board Bring-up: Clock sampling, reset control, SLD Node control
    - Avalon-ST Byte stream & packet stream
    - Avalon-MM Transaction Master
    - Nios Processor Control

Expected Use Cases

- Control system with a single Nios II processor core
- Control system with many Nios II processor cores
- Control system with no Nios II processor
  - Utilizing JTAG / Avalon Master Bridge peripheral
- Debug custom logic in isolation
- Troubleshoot clock and reset
- Troubleshoot pin-out issues
- Read and Write JTAG UART

Example Commands for JTAG-Avalon-MM Master

"turnon_LEDs.tcl"

```
# Define and initialize variables
set led_val 4
set led_pio 0x4000000

# Define a variable to service path: "master"
set jtag_master [lindex [get_service_paths master] 0]

# Open master service path
open_service master $jtag_master

# Utilize master to write to (poke) led peripheral
master_write_8 $jtag_master $led_pio $led_val

# Close the master service path
close_service master $jtag_master
```

- **system-console –script=turnon_LEDS.tcl**

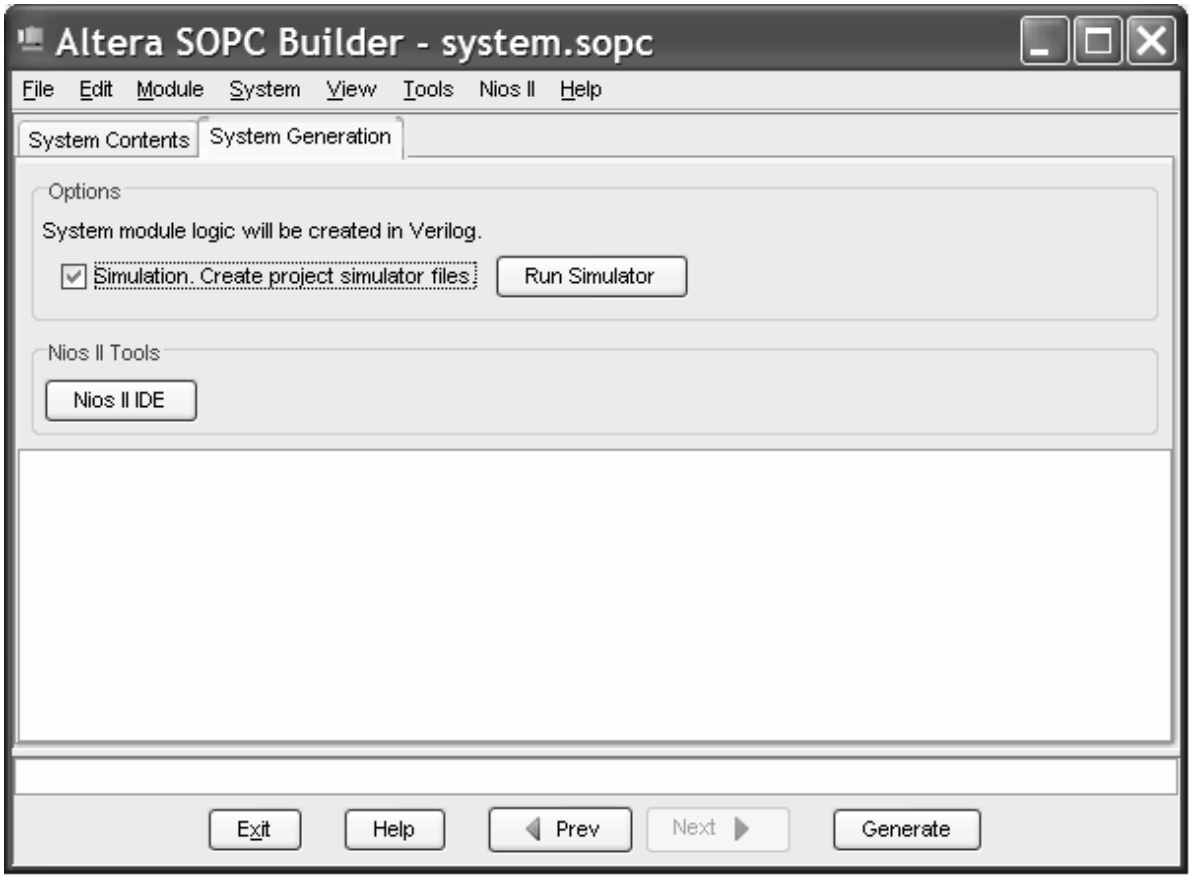
Lab 2
Build a Nios II Software Project

45 min

RTL Simulation

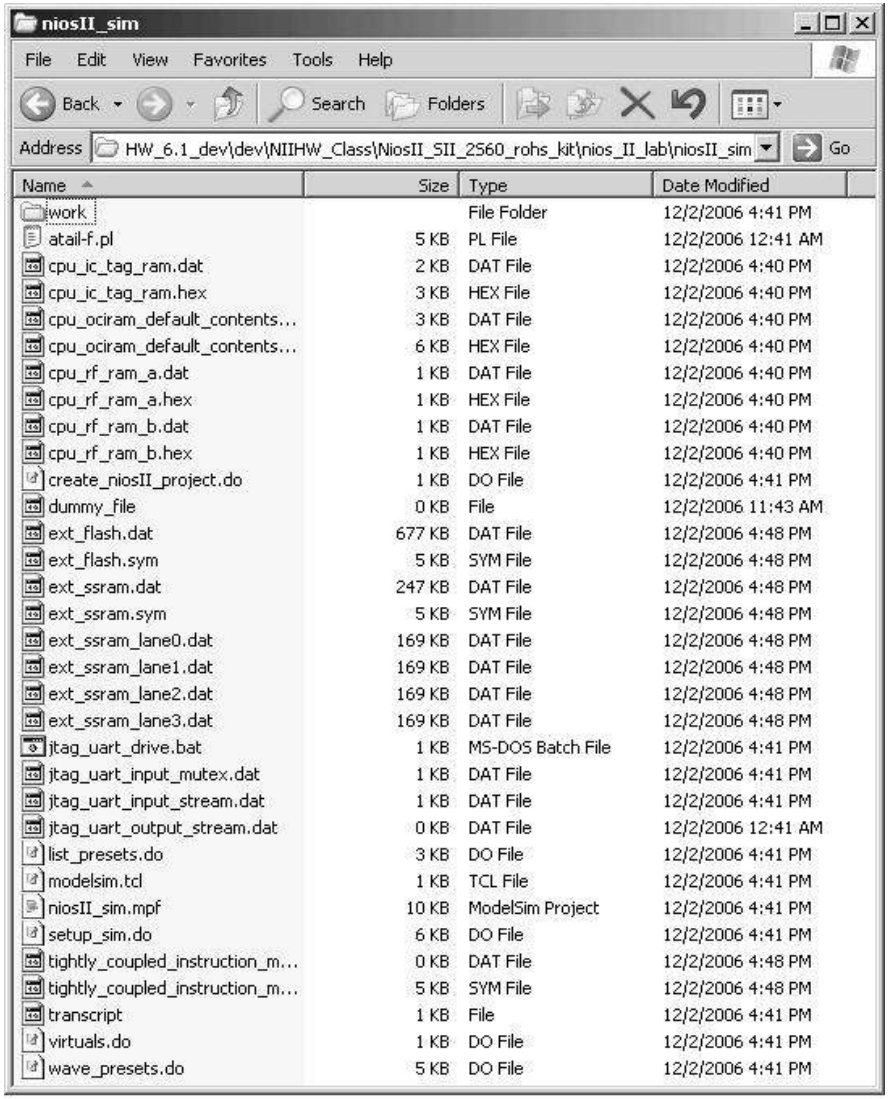
Enable ModelSim Simulation

- Check Simulation box in SOPC Builder before generating system



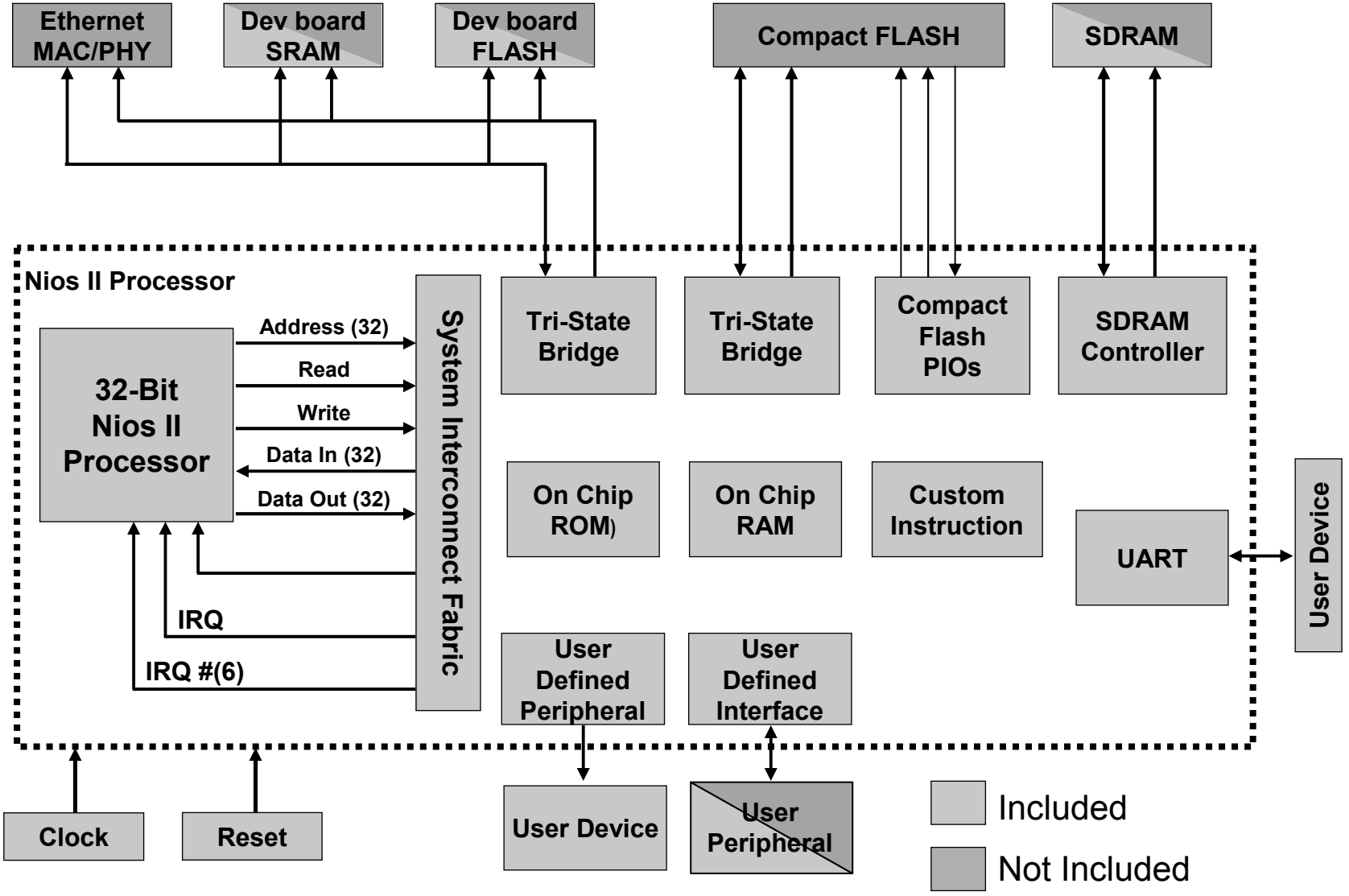
Files Generated in "<*system*>_sim" Directory

- Initialization files for simulation
- ModelSim Tcl Scripts
  - Eg. "setup_sim.do"
- ModelSim project (.mpf file)



Name	Size	Type	Date Modified
work		File Folder	12/2/2006 4:41 PM
atail-f.pl	5 KB	PL File	12/2/2006 12:41 AM
cpu_ic_tag_ram.dat	2 KB	DAT File	12/2/2006 4:40 PM
cpu_ic_tag_ram.hex	3 KB	HEX File	12/2/2006 4:40 PM
cpu_ociram_default_contents...	3 KB	DAT File	12/2/2006 4:40 PM
cpu_ociram_default_contents...	6 KB	HEX File	12/2/2006 4:40 PM
cpu_rf_ram_a.dat	1 KB	DAT File	12/2/2006 4:40 PM
cpu_rf_ram_a.hex	1 KB	HEX File	12/2/2006 4:40 PM
cpu_rf_ram_b.dat	1 KB	DAT File	12/2/2006 4:40 PM
cpu_rf_ram_b.hex	1 KB	HEX File	12/2/2006 4:40 PM
create_niosII_project.do	1 KB	DO File	12/2/2006 4:41 PM
dummy_file	0 KB	File	12/2/2006 11:43 AM
ext_flash.dat	677 KB	DAT File	12/2/2006 4:48 PM
ext_flash.sym	5 KB	SYM File	12/2/2006 4:48 PM
ext_ssram.dat	247 KB	DAT File	12/2/2006 4:48 PM
ext_ssram.sym	5 KB	SYM File	12/2/2006 4:48 PM
ext_ssram_lane0.dat	169 KB	DAT File	12/2/2006 4:48 PM
ext_ssram_lane1.dat	169 KB	DAT File	12/2/2006 4:48 PM
ext_ssram_lane2.dat	169 KB	DAT File	12/2/2006 4:48 PM
ext_ssram_lane3.dat	169 KB	DAT File	12/2/2006 4:48 PM
jtag_uart_drive.bat	1 KB	MS-DOS Batch File	12/2/2006 4:41 PM
jtag_uart_input_mutex.dat	1 KB	DAT File	12/2/2006 4:41 PM
jtag_uart_input_stream.dat	1 KB	DAT File	12/2/2006 4:41 PM
jtag_uart_output_stream.dat	0 KB	DAT File	12/2/2006 12:41 AM
list_presets.do	3 KB	DO File	12/2/2006 4:41 PM
modelsim.tcl	1 KB	TCL File	12/2/2006 4:41 PM
niosII_sim.mpf	10 KB	ModelSim Project	12/2/2006 4:41 PM
setup_sim.do	6 KB	DO File	12/2/2006 4:41 PM
tightly_coupled_instruction_m...	0 KB	DAT File	12/2/2006 4:48 PM
tightly_coupled_instruction_m...	5 KB	SYM File	12/2/2006 4:48 PM
transcript	1 KB	File	12/2/2006 4:41 PM
virtuals.do	1 KB	DO File	12/2/2006 4:41 PM
wave_presets.do	5 KB	DO File	12/2/2006 4:41 PM

Simulation Testbench



Simulation Testbench

```
3784
3785 // <ALTERA_NOTE> CODE INSERTED BETWEEN HERE
3786 //include additional files here
3787 // AND HERE WILL BE PRESERVED </ALTERA_NOTE>
3788
3789
3790
3791 module test_bench ;
3792
3793
3794   wire    [ 11: 0] in_port_to_the_button_pio;
3795   reg              clk;
3796   wire    [  1: 0] bidir_port_to_and_from_the_led_pio;
3797   wire    [  3: 0] be_n;
3798   wire             write_n_to_the_ext_flash;
3799   wire    [ 15: 0] address;
3800   wire             select0_n_to_the_ext_ram;
3801   wire             uart1_s1_readyfordata_from_sa;
3802   wire    [ 19: 0] off_chip_bus_address;
3803   wire             write_n;
3804   wire    [  3: 0] off_chip_bus_byteenablen;
3805   reg              reset_n;
3806   wire    [ 31: 0] off_chip_bus_data;
3807   wire             select1_n_to_the_ext_ram;
3808   wire    [ 15: 0] out_port_from_the_seven_seg_pio;
3809   wire             select0_n;
3810   wire             off_chip_bus_readn;
3811   wire             read_n;
3812   wire             txd_from_the_uart1;
3813   wire             rxd_to_the_uart1;
3814   wire             select_n_to_the_ext_flash;
3815   wire             uart1_s1_dataavailable_from_sa;
3816   wire    [ 31: 0] data;
3817   wire             select1_n;
3818   wire             write_n_to_the_ext_ram;
3819
3820
3821 // <ALTERA_NOTE> CODE INSERTED BETWEEN HERE
3822 //  add your signals and additional architecture here
3823 // AND HERE WILL BE PRESERVED </ALTERA_NOTE>
3824
```

- SOPC Builder creates testbench embedded inside top level file

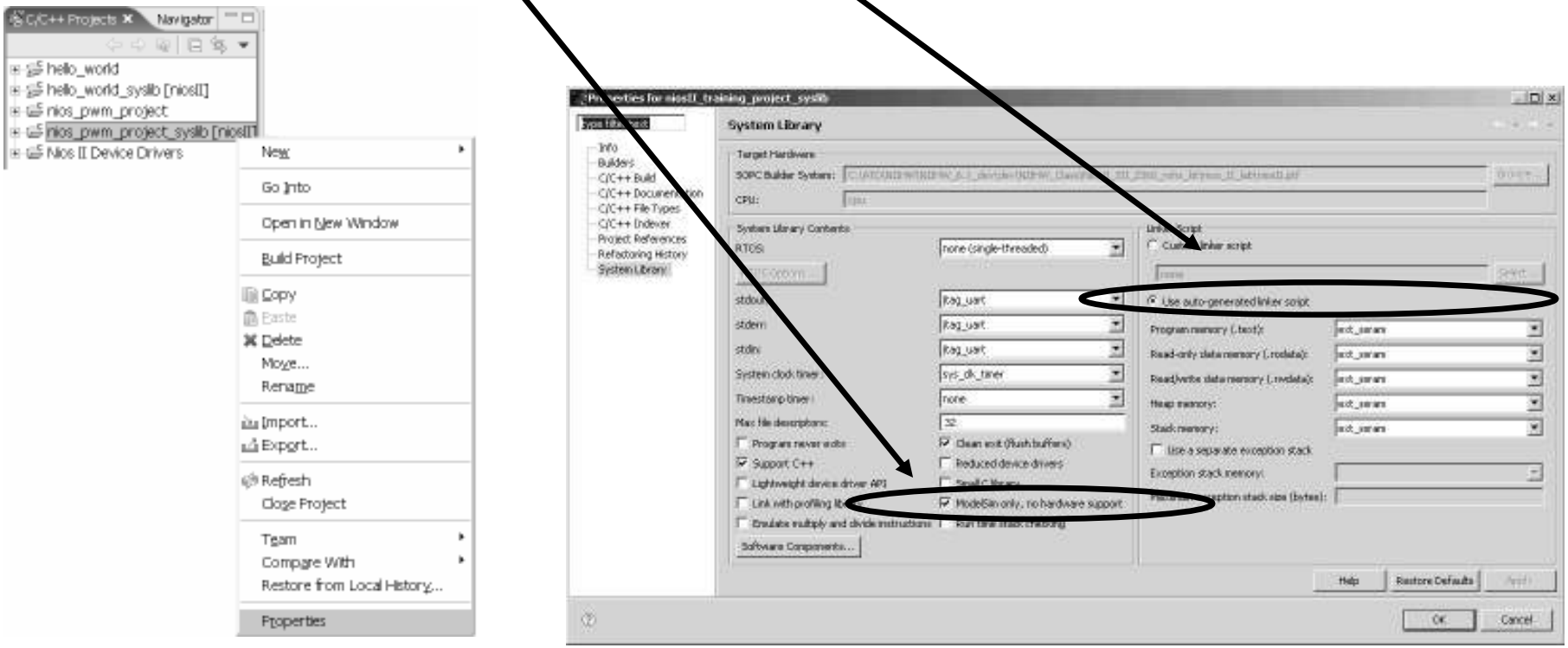
  (eg. niosII.v)

- Several sections are reserved within this file to add user files and code

- These sections are preserved if the SOPC builder is used to re-generate the Nios II processor system

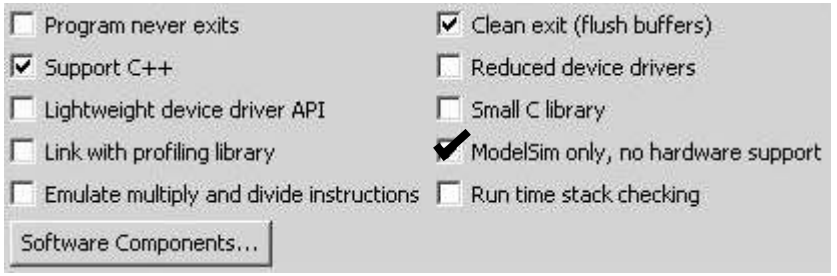
Running an RTL Simulation

- Modify Nios II IDE System Library For Simulation:
  - Specify Program memory
  - Set up as simulation only



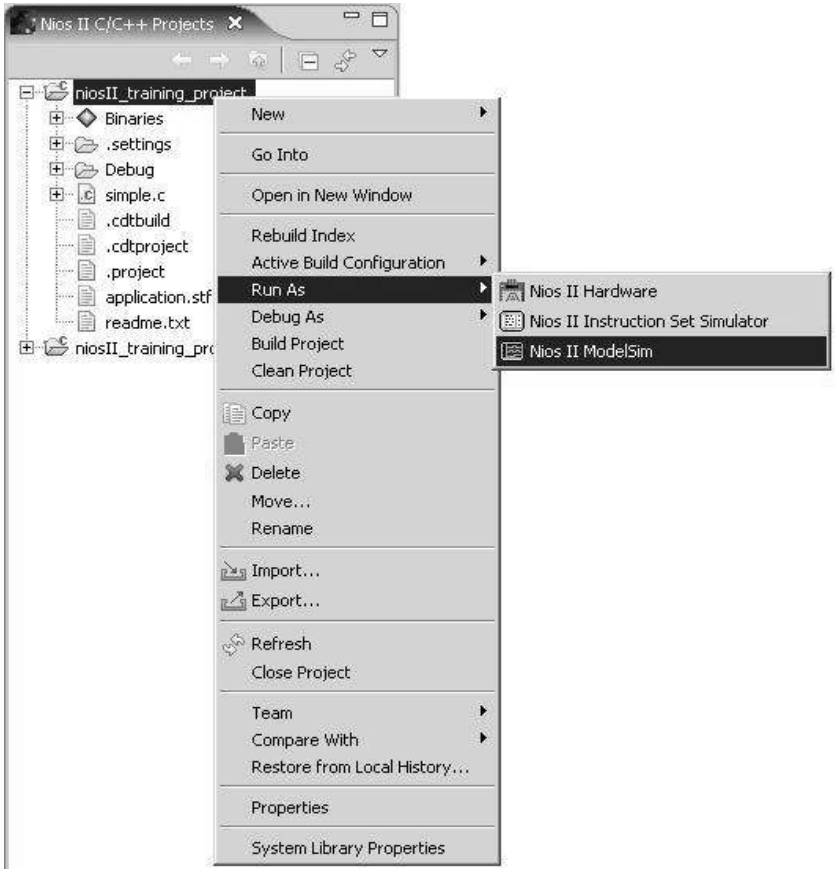
Running an RTL Simulation

- Checking the “ModelSim only, no hardware support” button:
  - Leaves caches un-initialized
  - Does not initialize .bss section of executable file
- As a result simulation speeds are increased
- You can still simulate with this button unchecked but simulation time will be much longer



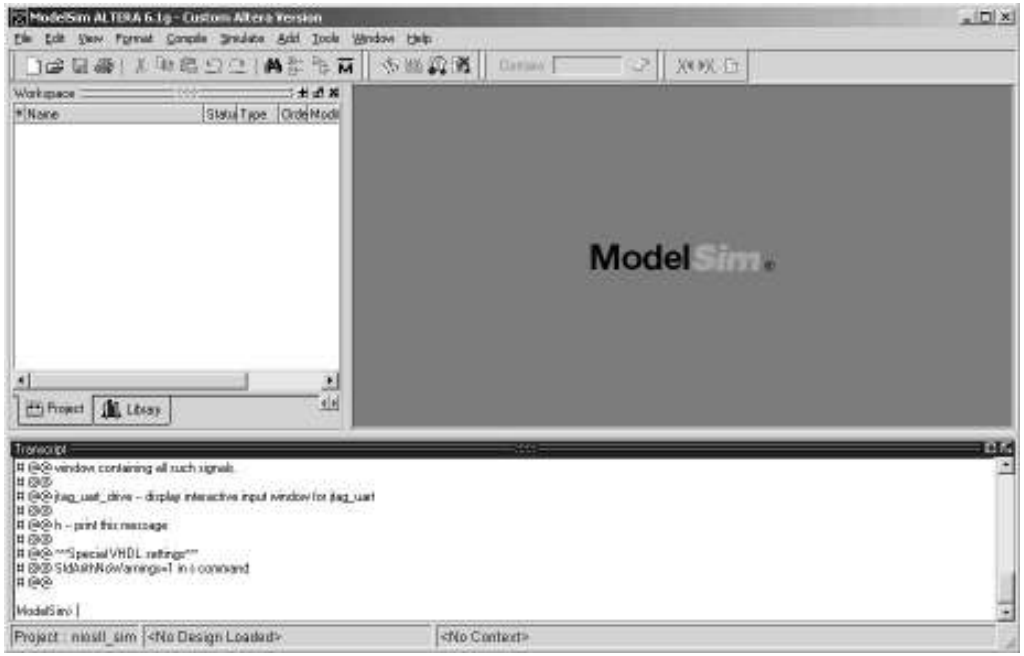
Running an RTL Simulation

- Launch ModelSim from Nios II IDE:
  - Highlight Software Project in **Nios II C/C++ Projects** panel
  - Right click
  - Run As > Nios II ModelSim



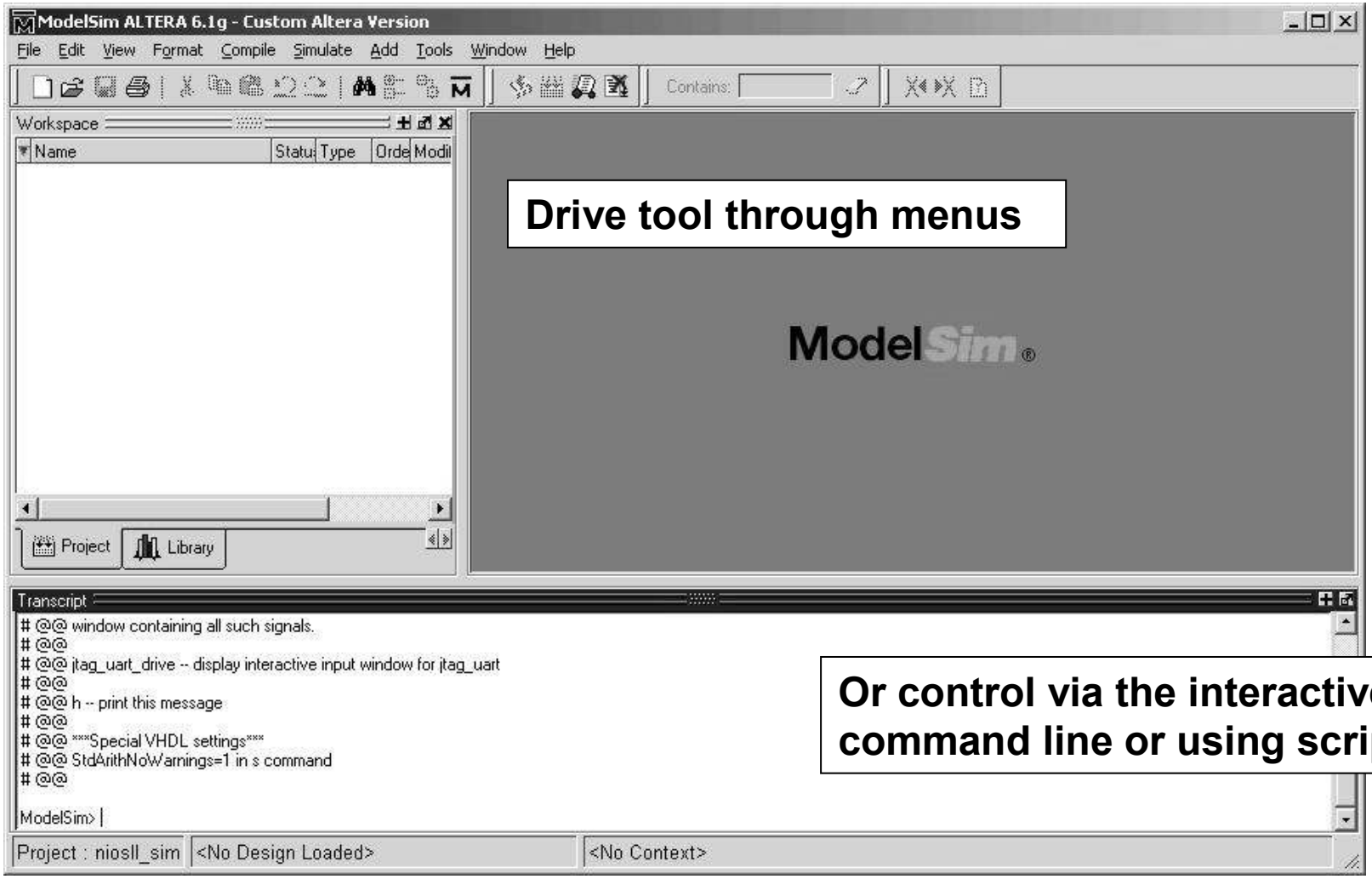
“Run As > Nios II ModelSim”

- Opens ModelSim
  1. Opens **.mpf** project file inside ModelSim
  2. Sources Altera-provided **“setup_sim.do”** script



Readies you for simulation!

Model*Sim* Main GUI



“setup_sim.do” Simulation Script

- Automatically creates aliases for other simulation scripts:
  - **s**↵ Compiles HDL source code and loads design
  - **w**↵ Opens Wave window with “useful” signals
  - **l**↵ Opens List window with “useful” signals
  - **h**↵ Displays help message describing scripts

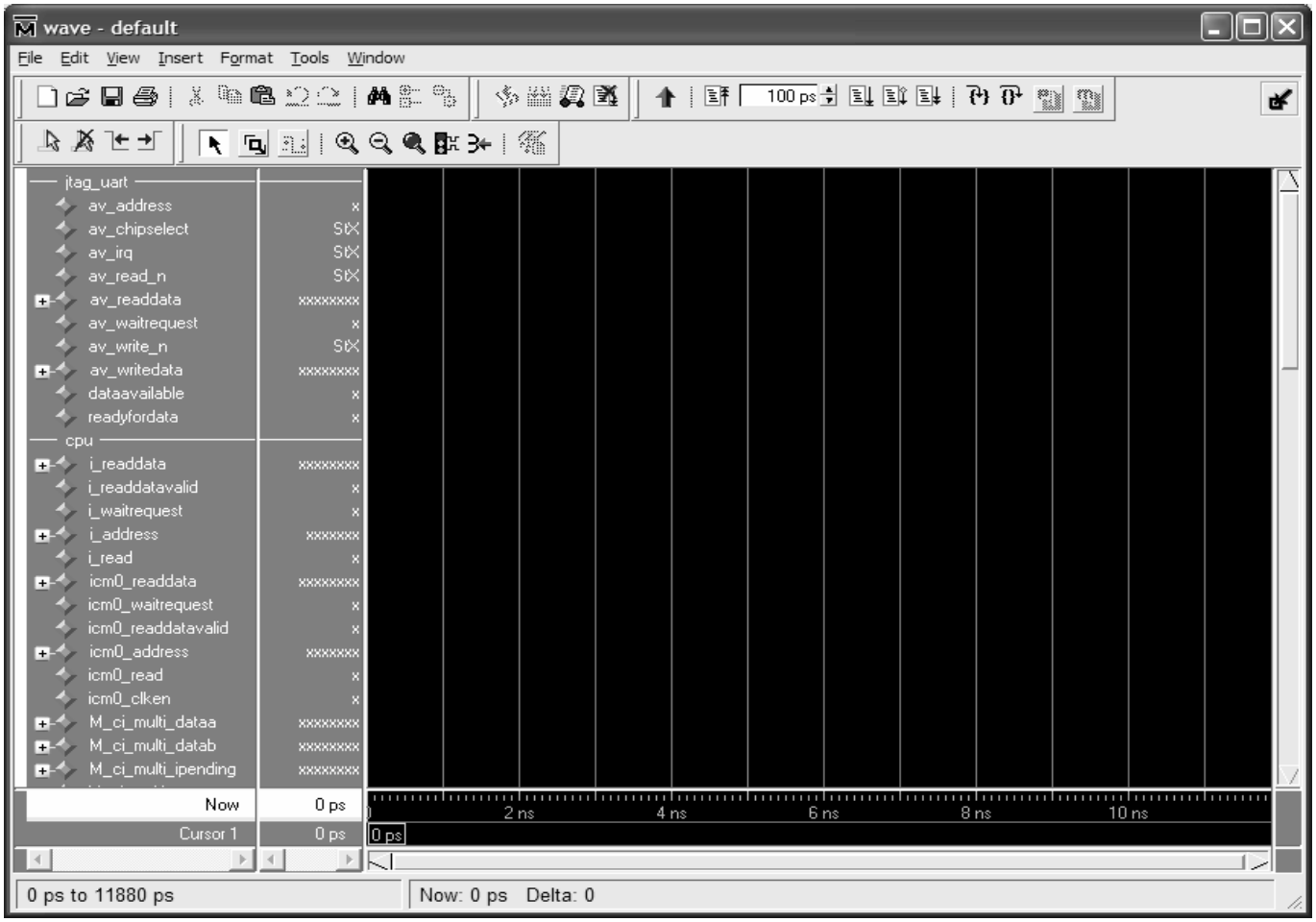
Time Saver!

- You can also create your own scripts
  - Run them on the command line by typing **do** or **source** followed by the name of the script:

Eg. ModelSim> do my_script.do

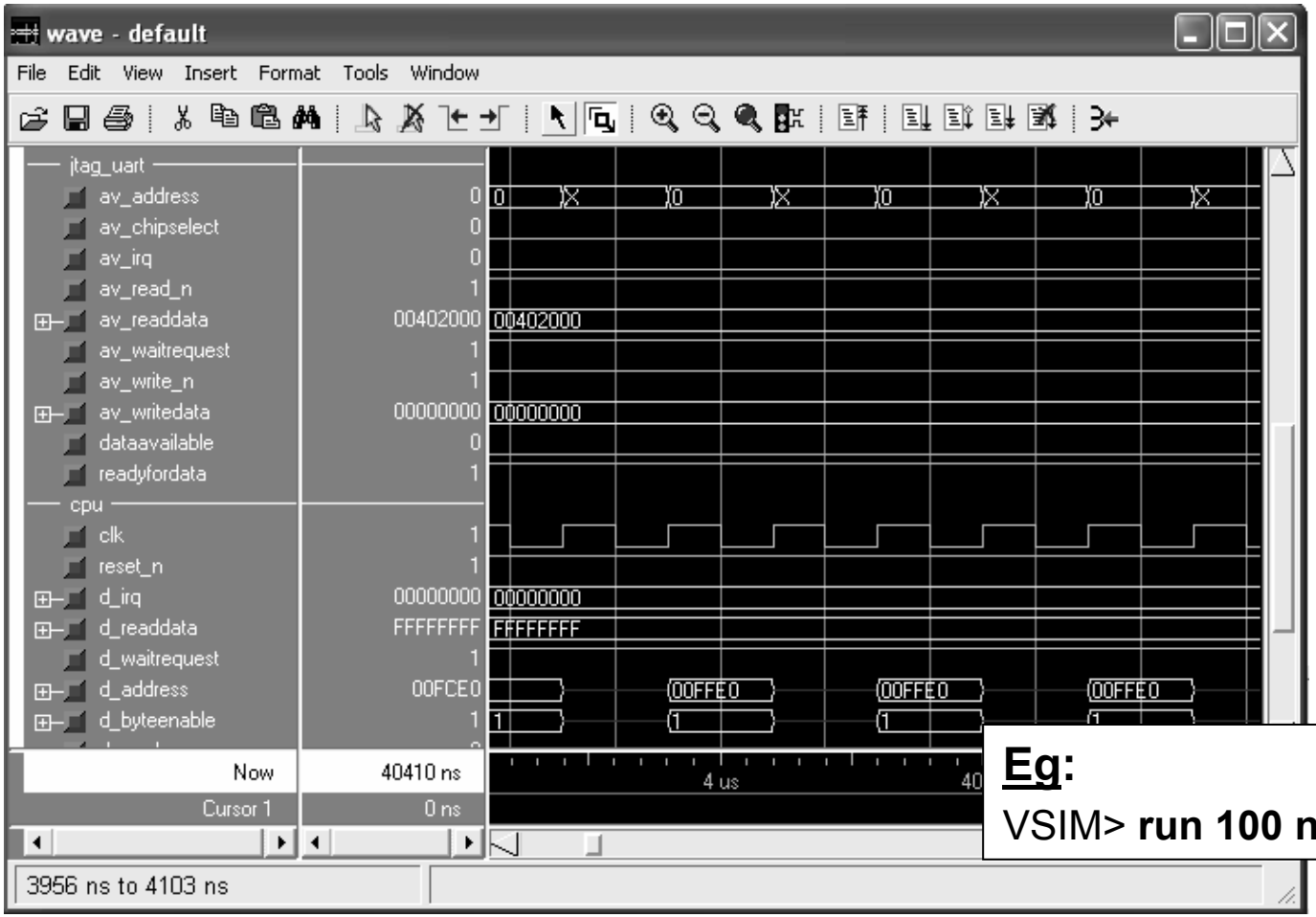
Open Wave Window –Type “w”

- Adds UART, CPU, other relevant signals by default



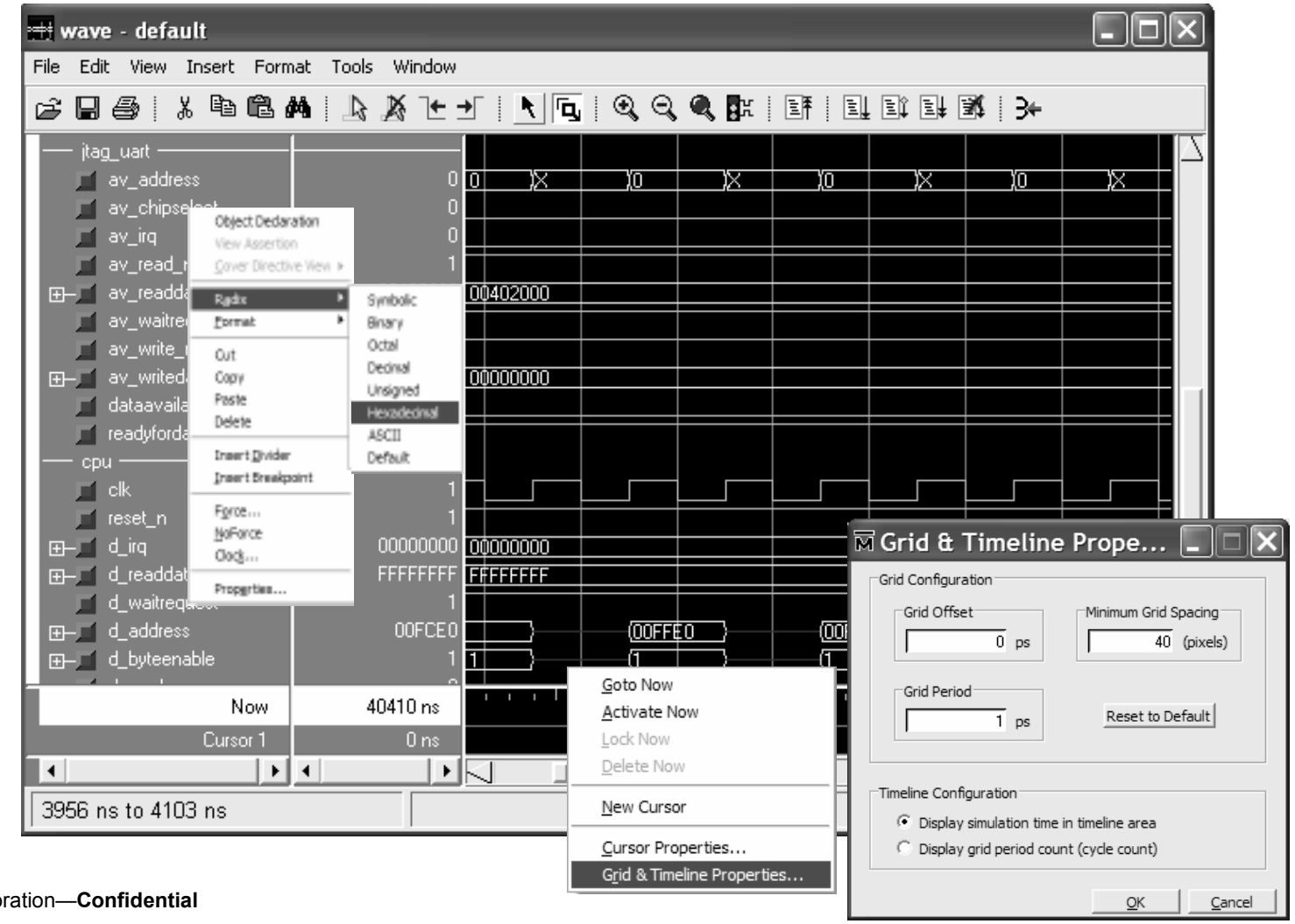
Advance the Simulation

- VSIM> **`run <time> <unit>`**

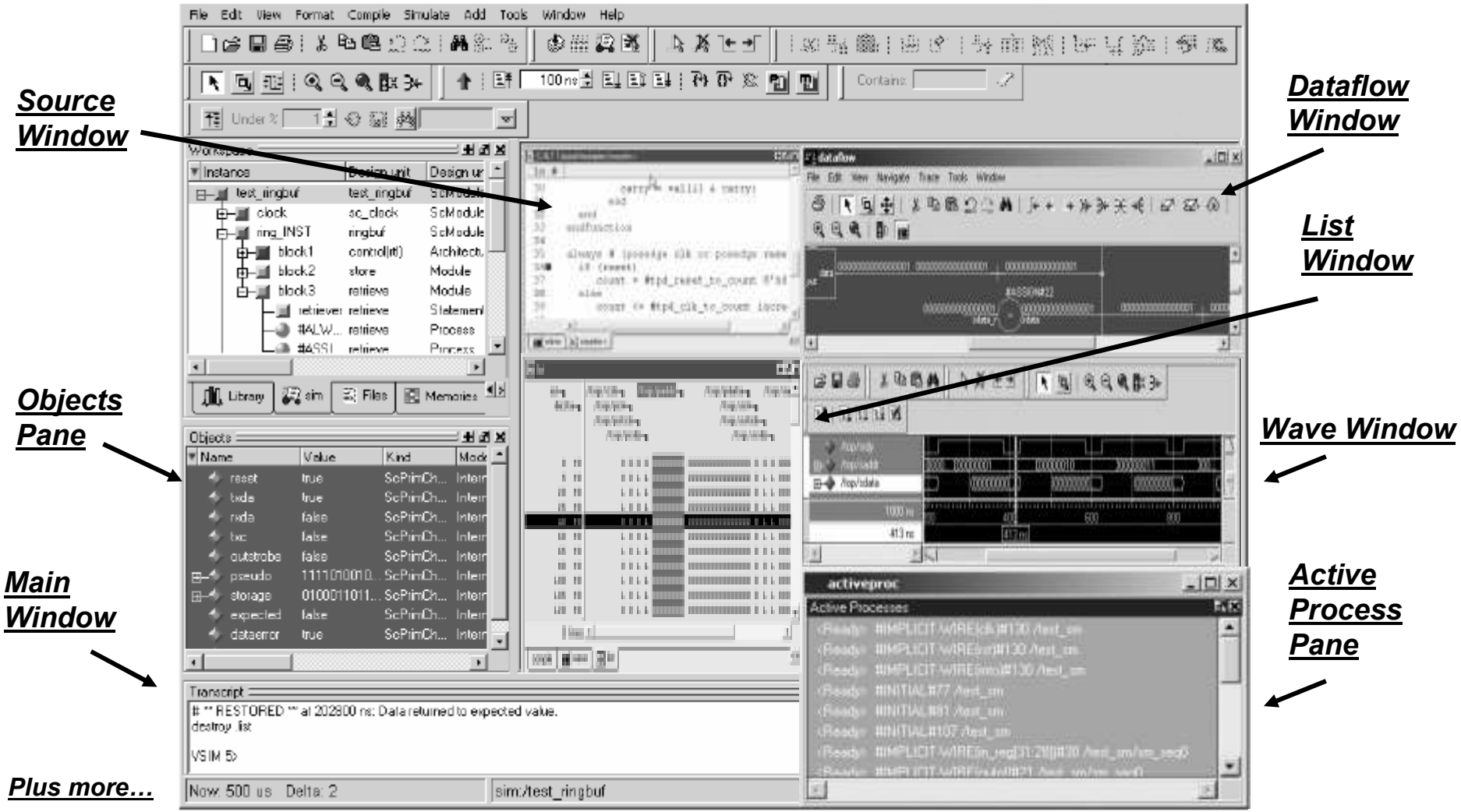


Control Display Properties

- Data format and time-axis units, for instance



Other Model*Sim* Debug Windows



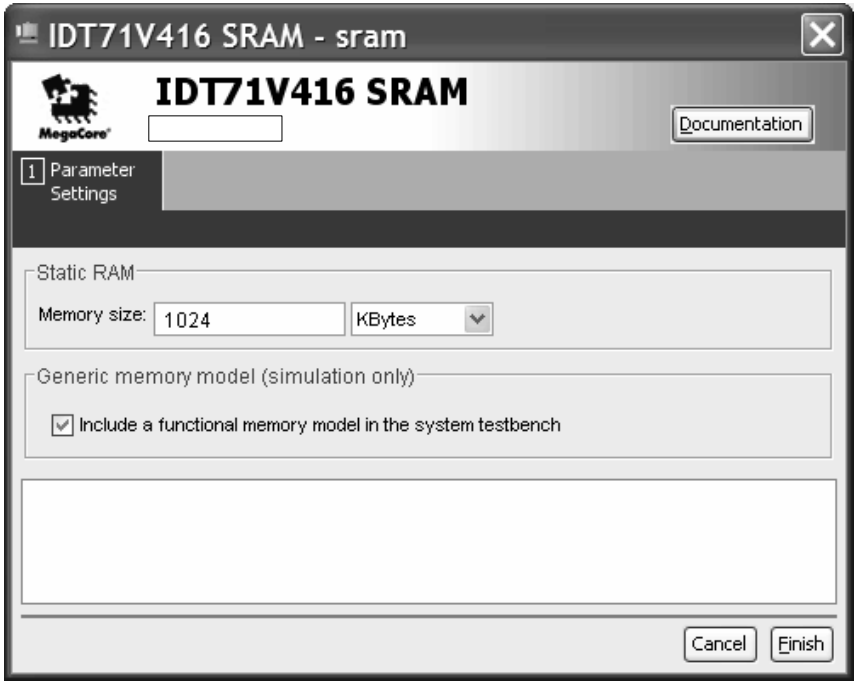
Simulation Models Set in SOPC Builder

- Applies to the following memories
  - On Chip Memory (ROM or RAM)
  - SRAM
  - Flash Memory and now SDRAM

Include SDRAM Model for Simulation

Memory Device Simulation Models

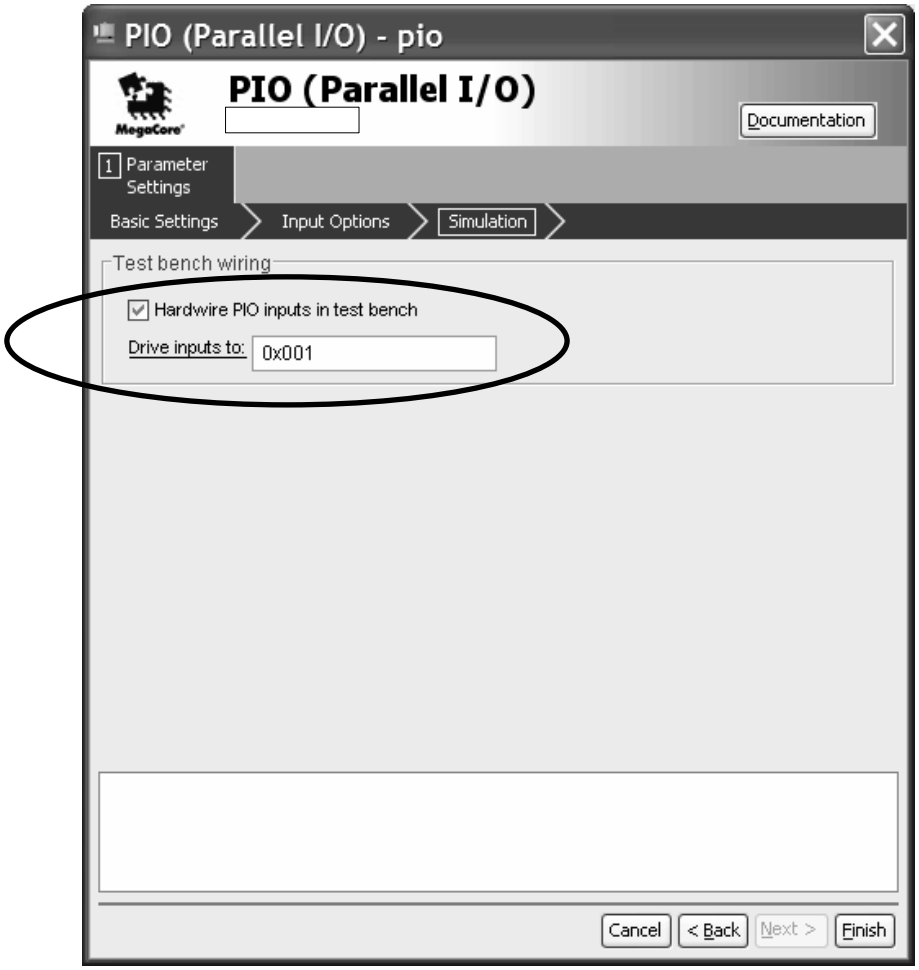
- You cannot *initialize* memories in SOPC Builder
  - Memory init files are created by the Nios II IDE



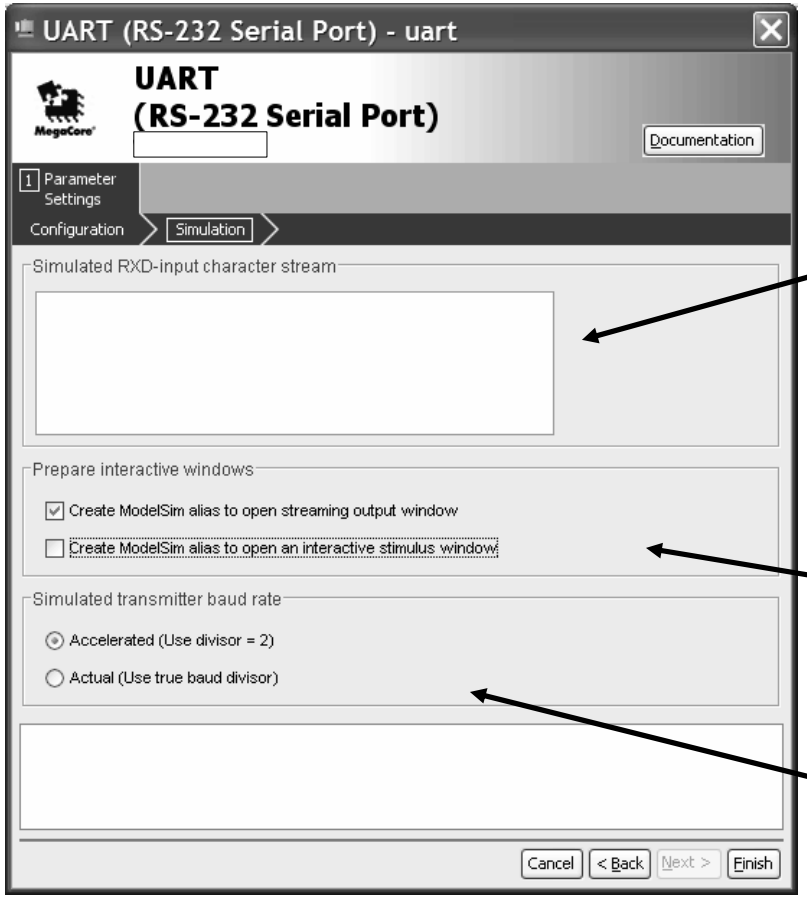
- **Eg. ext_ram** will be initialized for simulation with the **ext_ram.dat** file
  - You must compile software in Nios II IDE to populate
- Onchip memories are initialized with **<component_name>.hex**
  - On-chip memory initialization files can be created by an editor or by Nios II IDE

Initializing PIO Peripherals

- Eg.



UART Simulation



- Enter text to be transmitted to UART during simulation
- Creates and saves txt file containing UART tx stream
- Creates window to input text at simulation run time

Note: *ModelSim Options are mutually exclusive*

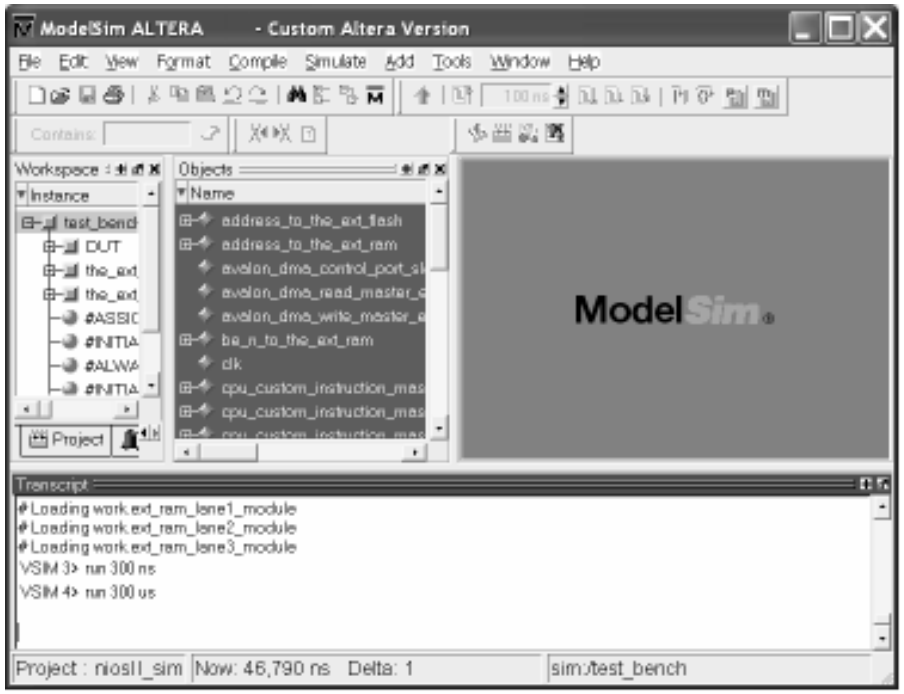
- Accelerates simulation

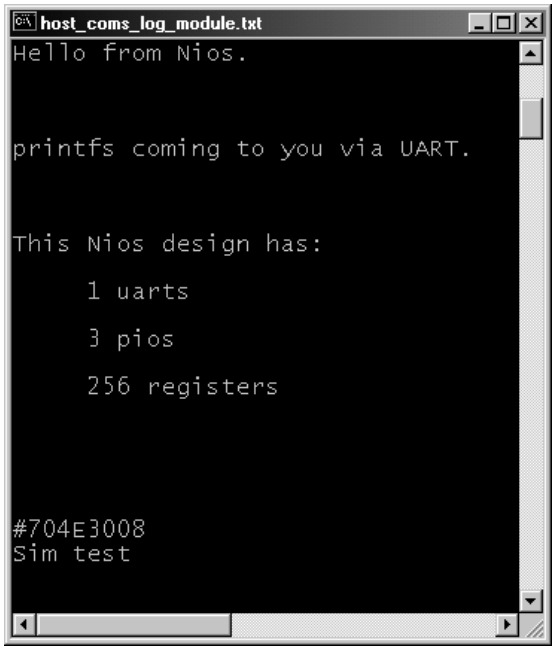
UART Simulation

- Input predefined or interactive
- Output displayed / saved independently for each UART

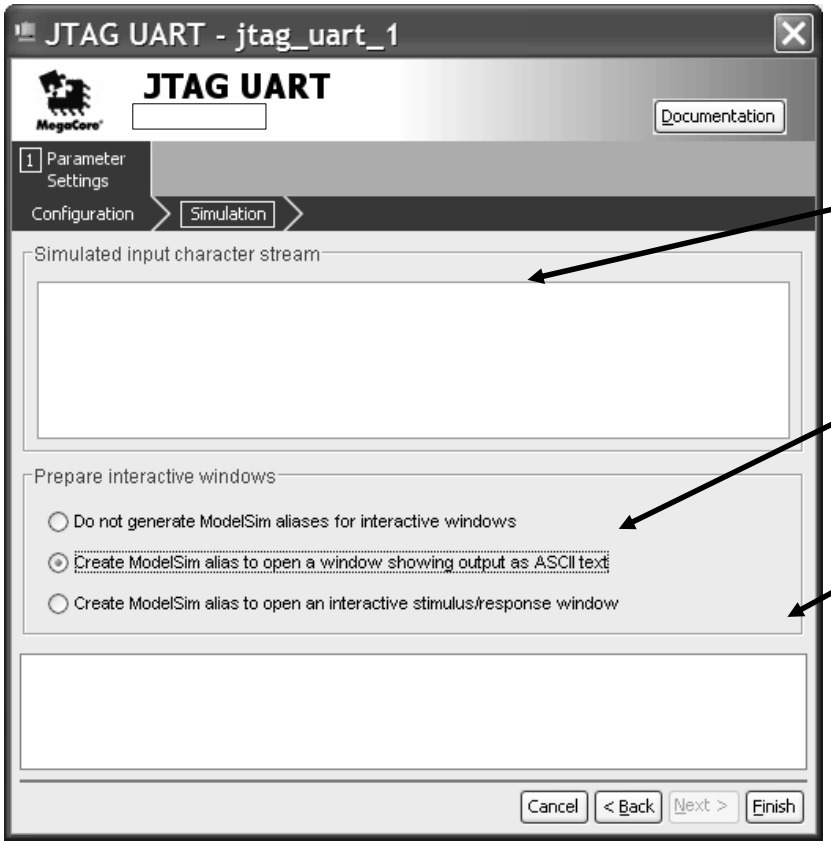
Type:

- **uart_name_log → *for log window***
- **uart_name_drive → *for interactive stimulus window***





JTAG_UART Simulation

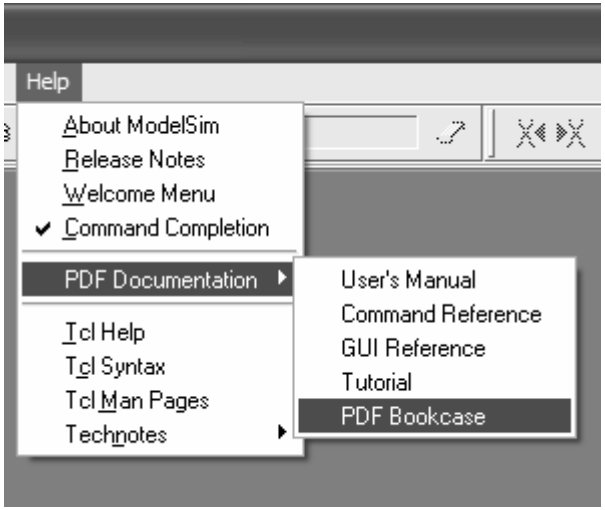


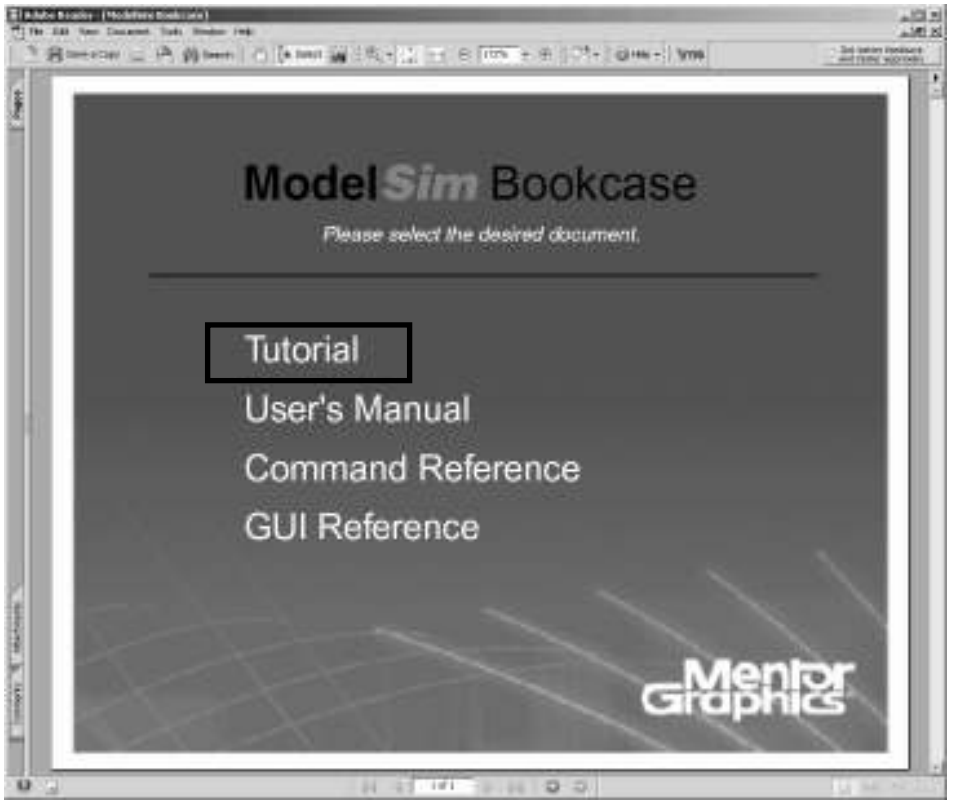
- Text is transmitted to JTAG_UART peripheral during simulation
- Creates and saves txt file containing UART tx stream
- Creates window to input text at simulation run time

Note: ModelSim simulation options are mutually exclusive

For More Information on ModelSim

- See On-Line training at Altera
- Go through ModelSim tutorial in installed HELP documentation folder
  - **C:\Modeltech_ae\docs**



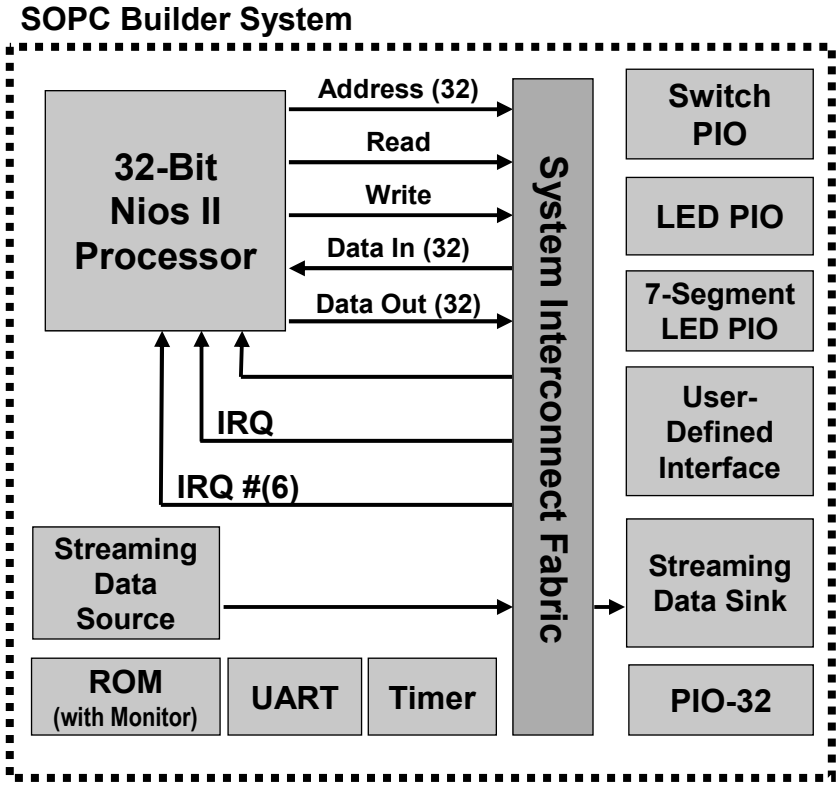


System Interconnect Fabric

Basics

System Interconnect Fabric

- Interconnect specification used within SOPC Builder
- Principal design goals
  - Low resource utilization for bus logic
  - Simplicity
  - Synchronous operation
- Transfer Types
  - Slave Transfers
  - Master Transfers
  - Latency-Aware Transfers
  - Burst Transfers
  - Transfers w/t Flow Control
  - Streaming Transfers

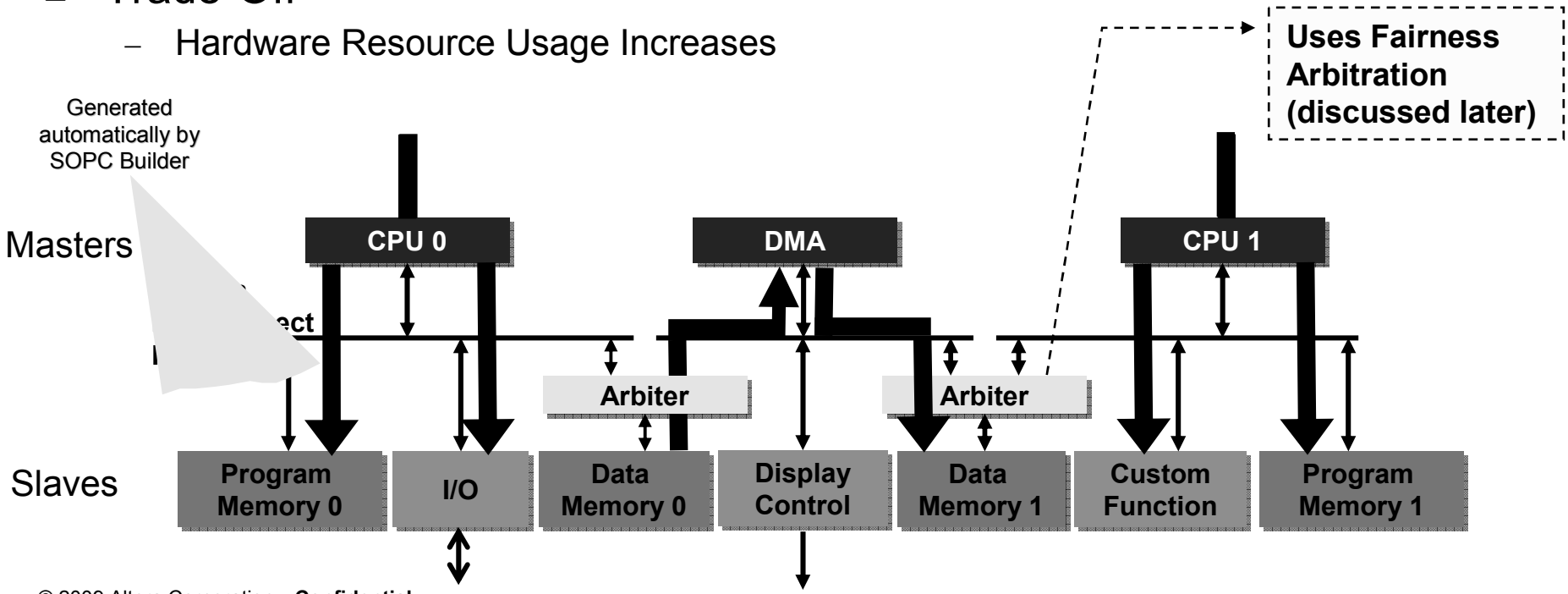


System Interconnect Fabric (SIF)

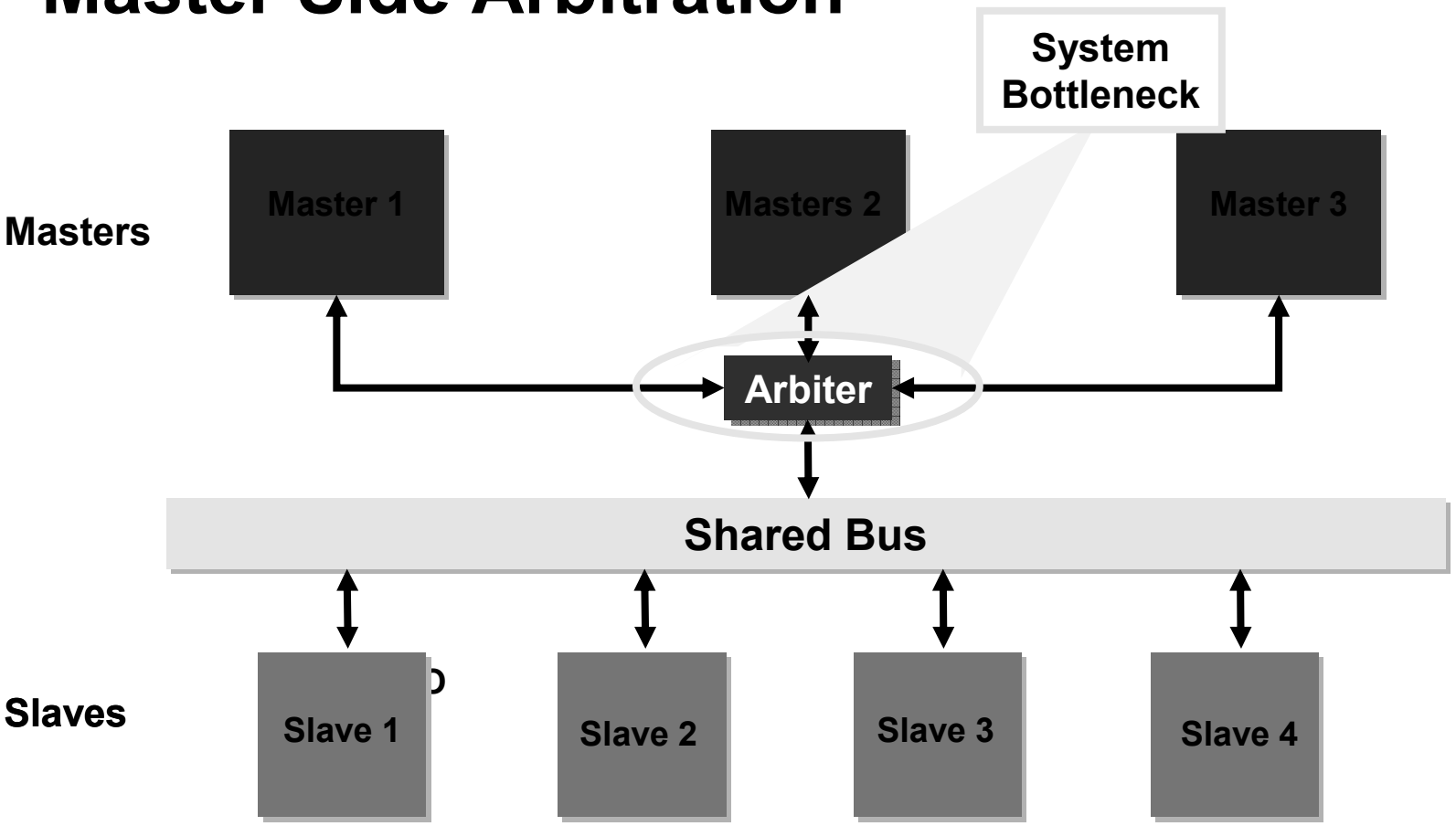
- **Two Standards:**
  - **Avalon Memory Mapped Interface *(Avalon-MM)***
    - Formerly called “Avalon Switch Fabric”
    - Defines interconnect strategy for peripherals
      - *peripheral* ←→ *interconnect* ←→ *peripheral*
    - Peripherals only need to implement the specific signal types needed to support desired transfers
    - Supports **simultaneous multi-mastering**
  - **Avalon Streaming Interface *(Avalon-ST)***
    - Defines standard, flexible, and modular protocol for **unidirectional**, synchronous transfer of data from ***source*** to a ***sink***
      - Multiplexed data streams, packets, and DSP data
      - Point-to-point connections
    - Connect components through SOPC Builder or standard HDL
    - For high throughput, low latency datapath implementation

SIF Allows Simultaneous Multi-Mastering

- System Interconnect Composed of FPGA Routing Resources
  - Point-to-point connections inside chip
  - Slave-side arbitration
    - Multiple Bus Transactions Can Operate Simultaneously
    - Provided they don't access same slave during bus cycle
  - I/O Devices Can be Grouped Based on Bandwidth Requirement
- Trade-Off
  - Hardware Resource Usage Increases



Versus a “Traditional” Architecture: Master Side Arbitration



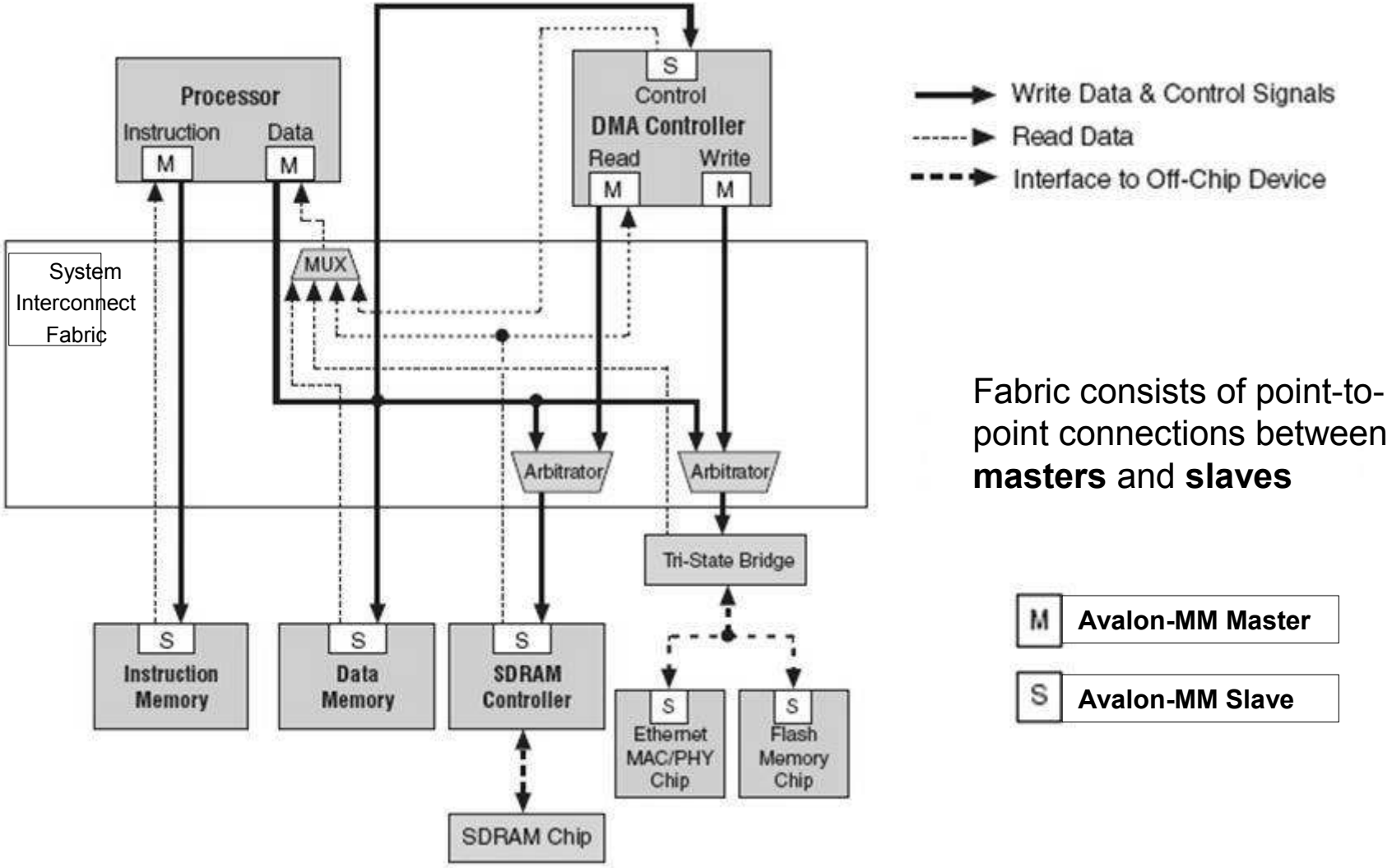
…where Masters wait in line for shared bus

Interconnections

- Automatically generated by SOPC Builder
- Custom generated for peripherals in system
  - Contingencies are on per-peripheral basis
  - System not burdened by unnecessary bus complexity

- SOPC Builder takes care of
  - Arbitration
  - Address Decoding
  - Data Path Multiplexing
  - Bus Sizing
  - Wait-State Generation
  - Interrupts

Avalon-MM Interface Block Diagram



Fabric consists of point-to-point connections between **masters** and **slaves**

Avalon-MM Master Ports

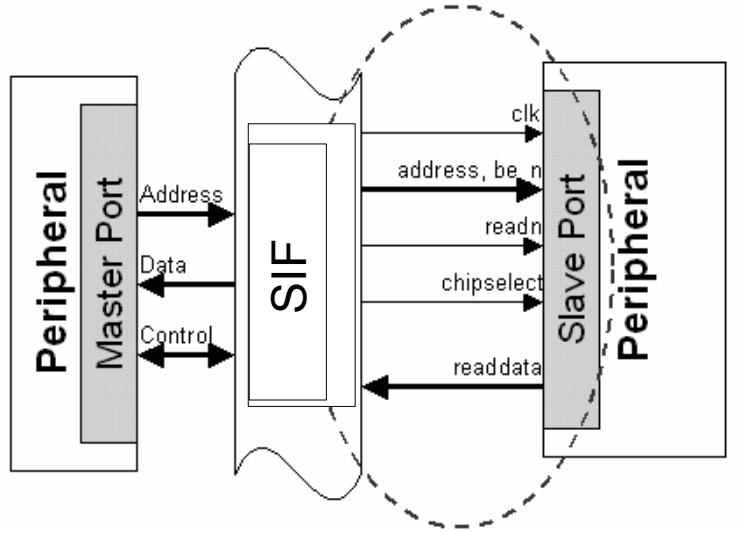
- Initiate Transfers with System Interconnect Fabric
- Transfer Types
  - Fundamental Read
  - Fundamental Write
- Avalon-MM Masters must honor **waitrequest** signal
- Transfer Properties
  - Latency awareness
  - Burst
  - Flow Control

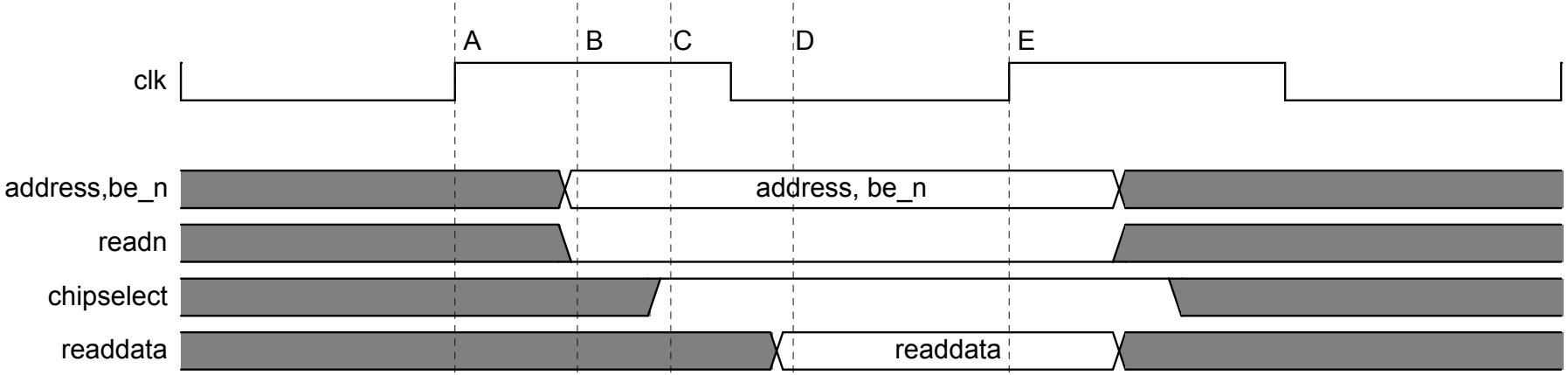
Avalon-MM Slave Ports

- Respond to Transfer Requests from System Interconnect Fabric
- Transfer Types
  - Fundamental Read
  - Fundamental Write
- Transfer Properties
  - Wait States
  - Latency
  - Bursting
  - Flow Control

Avalon-MM Slave Read Transfer

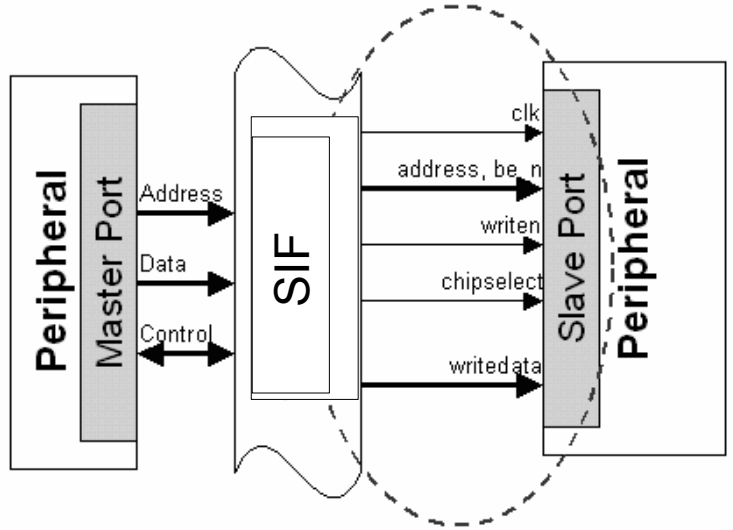
- 0 Setup Cycles
- 0 Wait Cycles

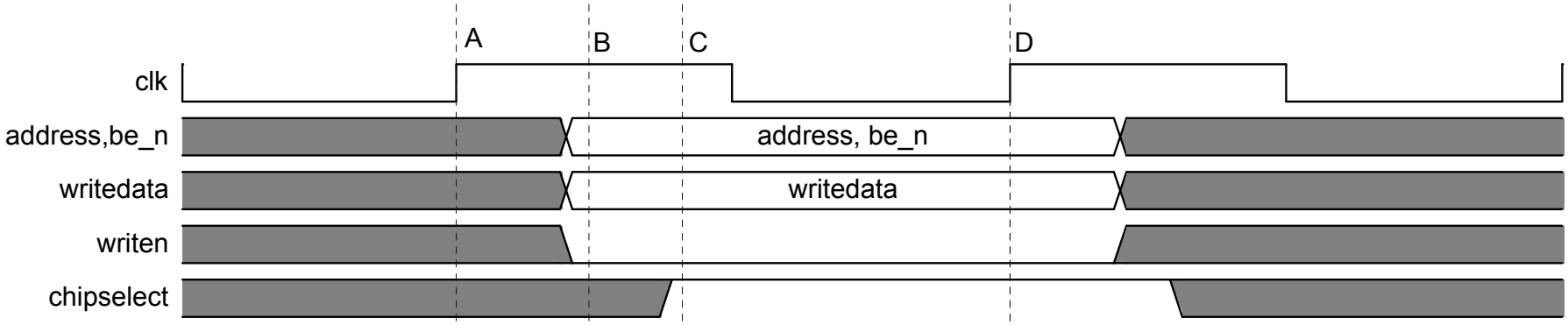




Avalon-MM Slave Write Transfer

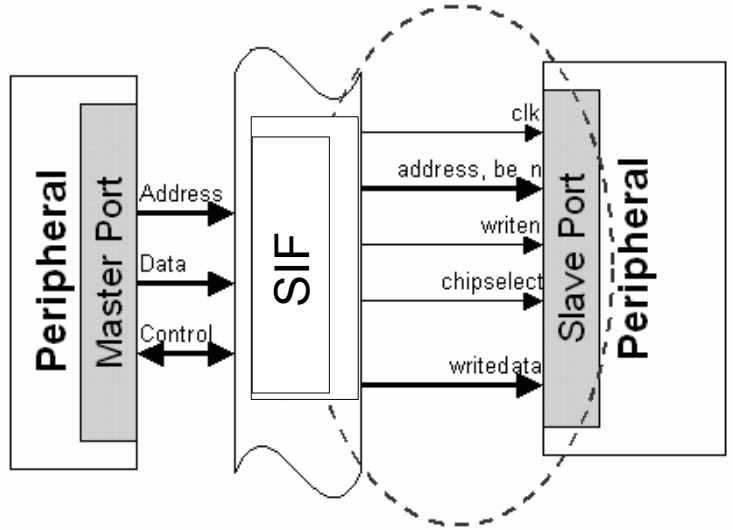
- 0 Setup Cycles
- 0 Wait Cycles
- 0 Hold Cycles

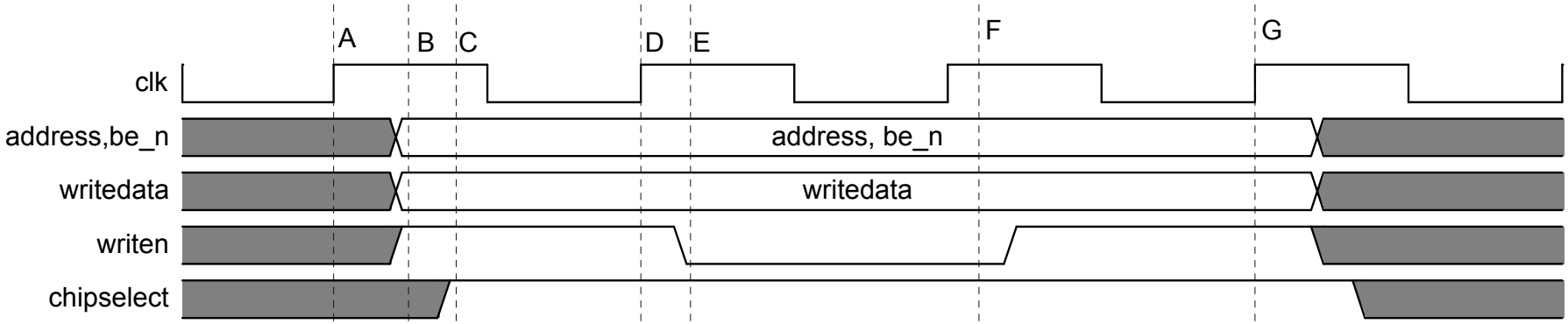




Avalon-MM Slave Write Transfer

- 1 Setup Cycle
- 0 Wait Cycles
- 1 Hold Cycle





User-Defined Custom Peripherals

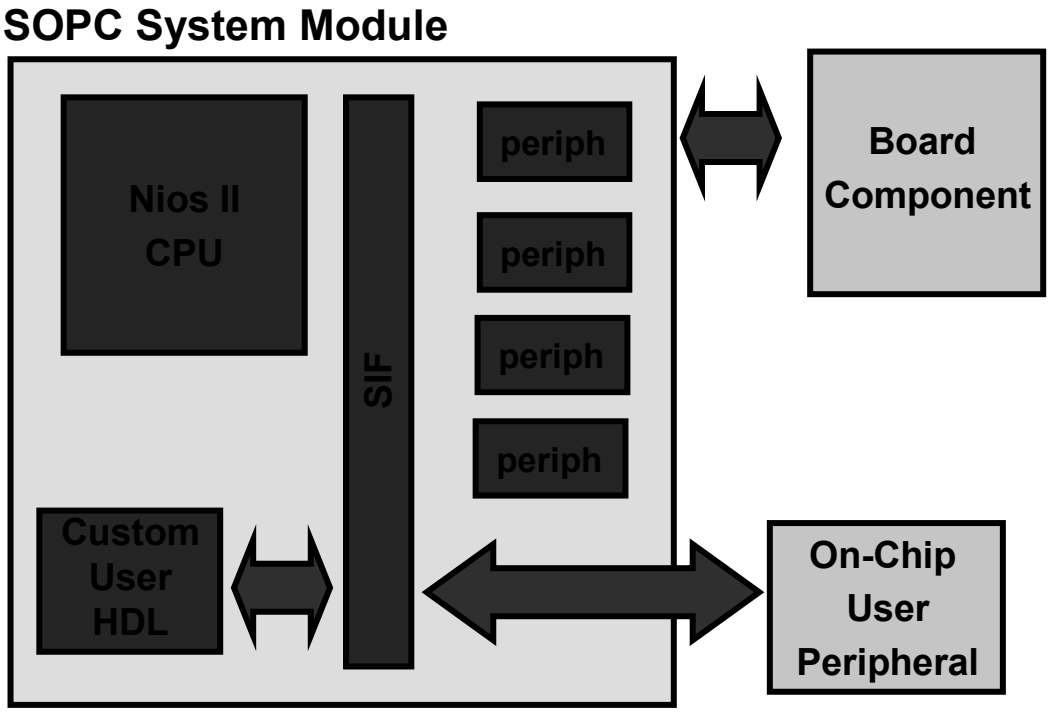
Custom Peripherals

- Add a peripheral not included with SOPC Builder
  - To perform some kind of proprietary function or perhaps a standard function that is not yet included as part of the kit
  - To expand or accelerate system capabilities

- You are now going learn how to connect your own design directly to the system through the System Interconnect Fabric
  - *Note:* As many peripherals contain registers, you could *also* have chosen to use a PIO rather than connect directly to the bus

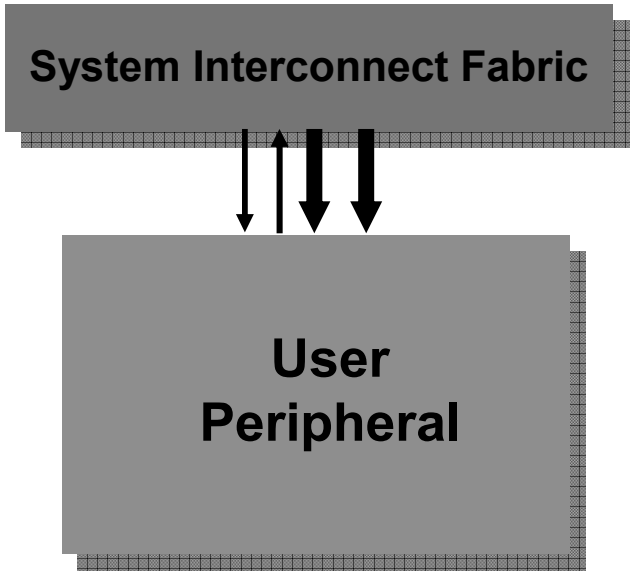
Custom Peripherals

- Map into Nios II processor memory space
- Can be on-chip or off-chip
  - HDL code or an external component on your board
    - HDL code can map inside SOPC Builder system or out



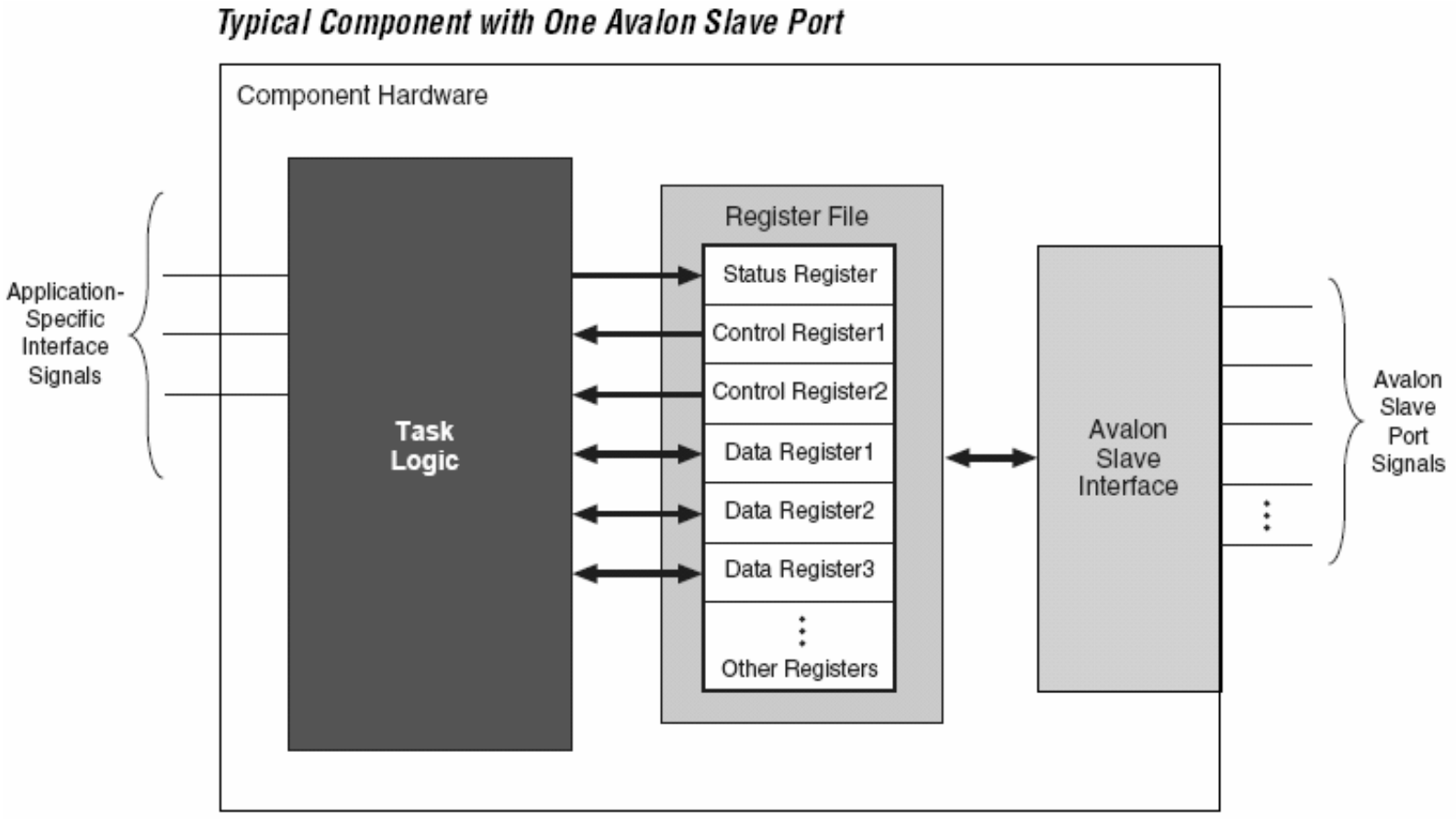
Creating Custom Peripherals

- No need to worry about creating the bus interface to System Interconnect Fabric inside your peripheral
  - Implement only the signals you need
  - Avalon Memory Mapped Interface will adapt to connect to the peripheral's ports
  - Timing handled automatically
  - Fabric created for you
  - Arbiters generated as needed

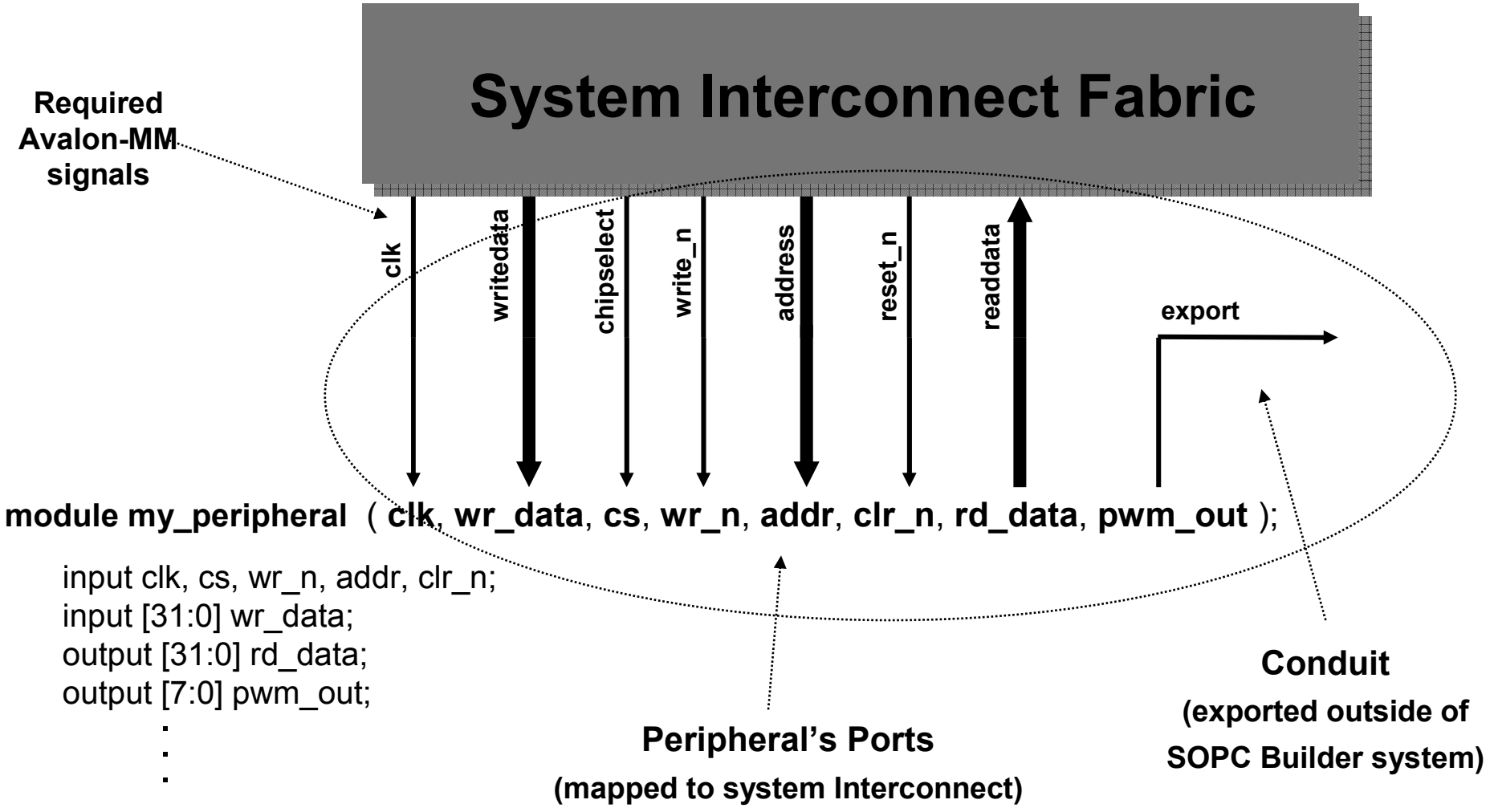


Concentrate Effort on Peripheral Functionality!

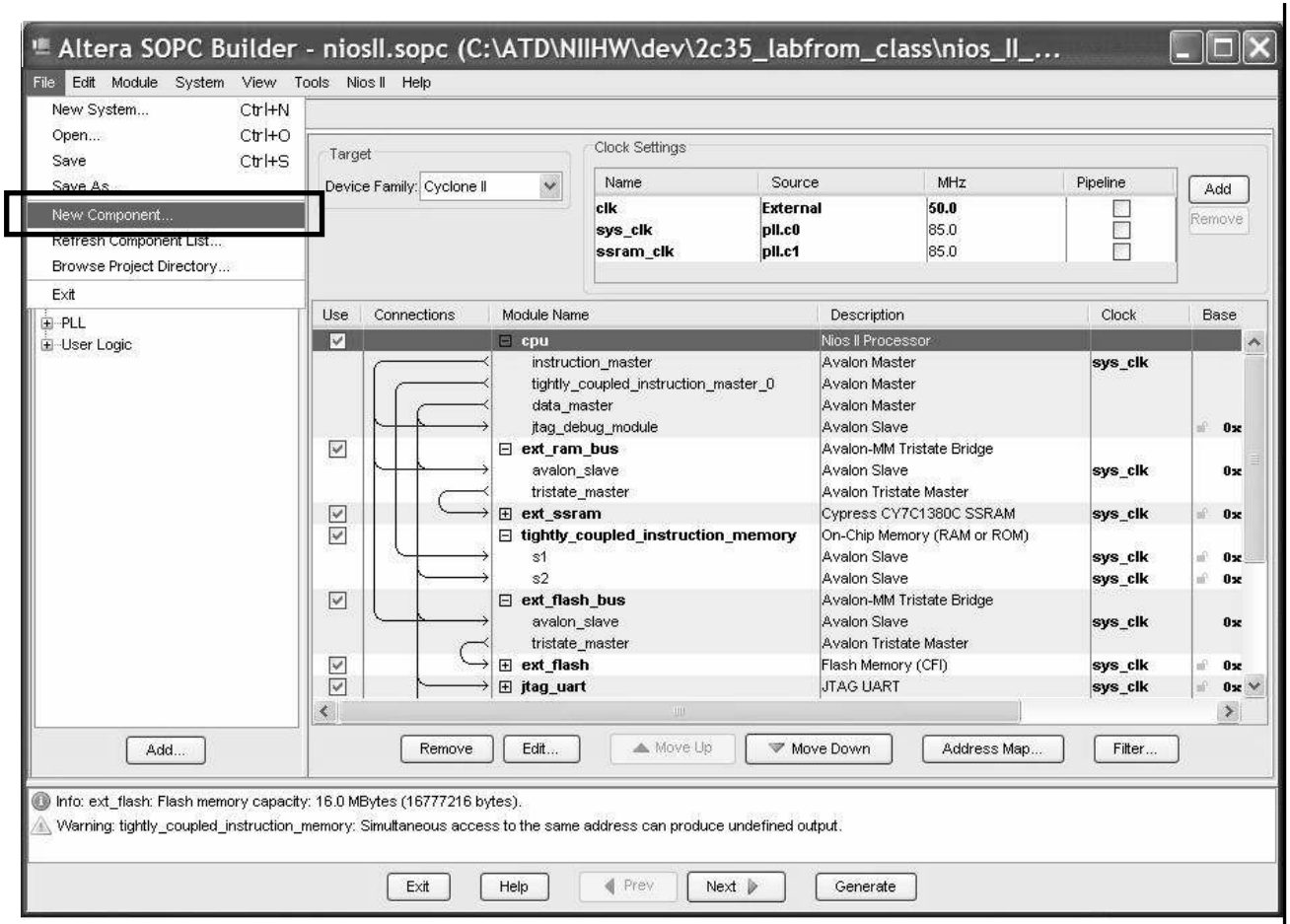
Example Registered Avalon-MM Slave



Map Ports to Interconnect Signal Types



Component Editor



Three Uses:

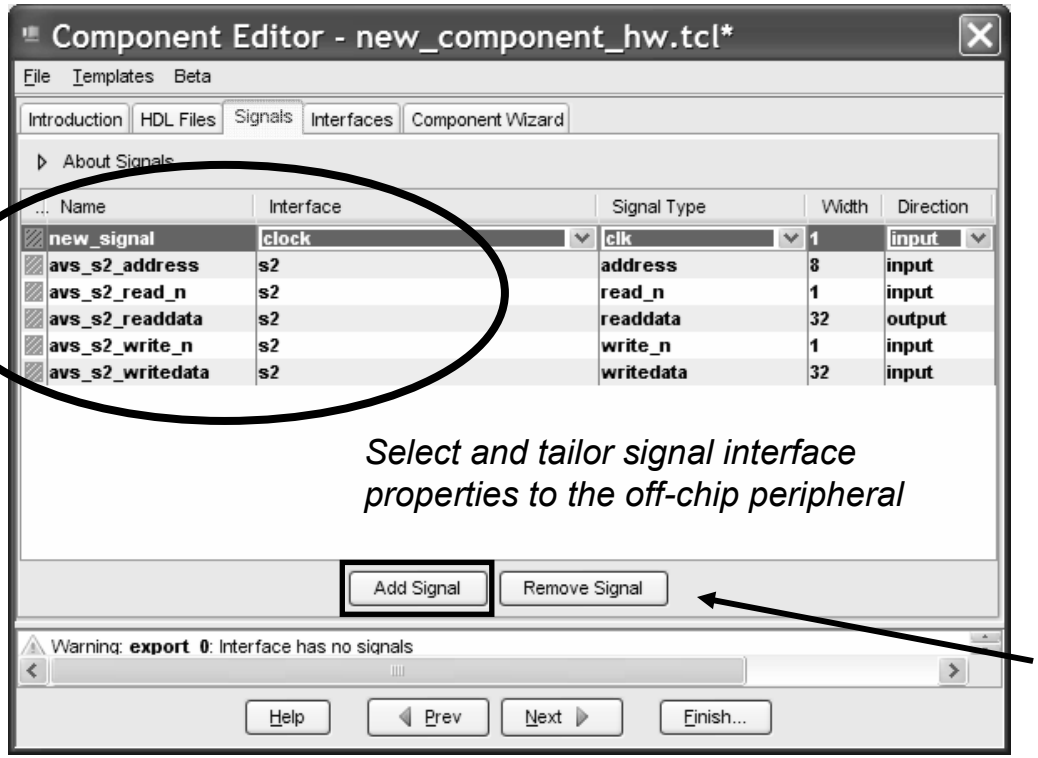
1. Create top level ports to connect S.I.F. to a peripheral living **outside** SOPC System *(on or off-chip)*

2. Create HDL template to import into SOPC Builder

3. Create direct *on-chip* connection between system bus and user HDL Code **inside** SOPC Builder system

1. Map Top Level Ports to S.I.F.

- To connect to peripherals outside SOPC Builder system (on or off-chip)
  - Eg. Create signal interface based on data sheet requirements



Select and tailor signal interface properties to the off-chip peripheral

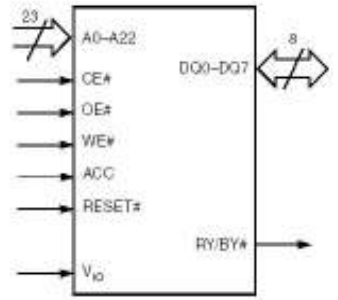
AMD29LV065AD CFI Flash Chip

AMD

PIN DESCRIPTION

A0–A22	=	23 Addresses inputs
DQ0–DQ7	=	8 Data inputs/outputs
CE#	=	Chip Enable input
OE#	=	Output Enable input
WE#	=	Write Enable input
ACC	=	Acceleration Input
RESET#	=	Hardware Reset Pin input
RY/BY#	=	Ready/Busy output
V_{CC}	=	3.0 volt-only single power supply (see Product Selector Guide for speed options and voltage supply tolerances)
V_{IO}	=	Output Buffer power
V_{SS}	=	Device Ground
NC	=	Pin Not Connected Internally

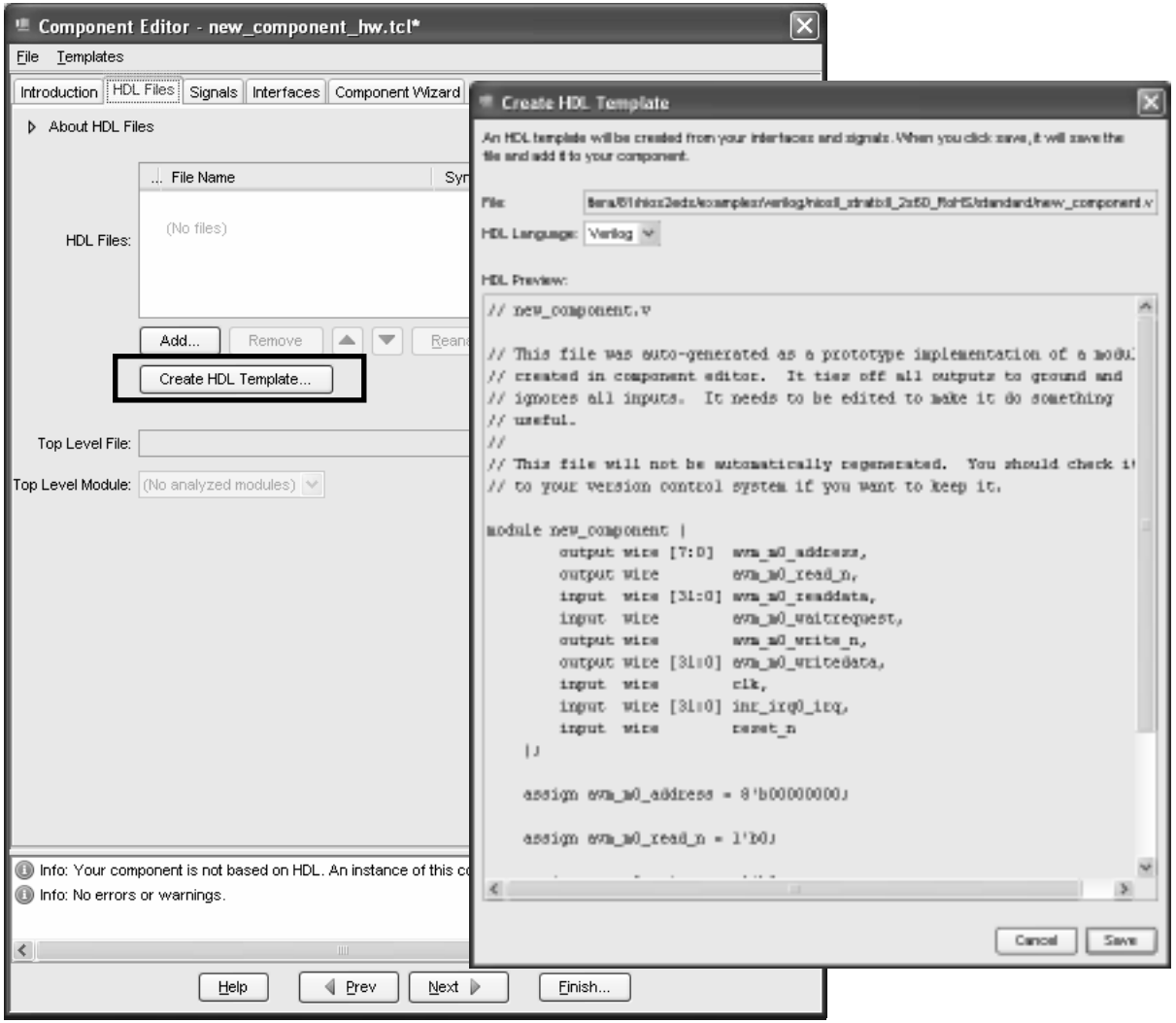
LOGIC SYMBOL



- Can utilize existing HDL templates to speed up process

Can also manually remove ports

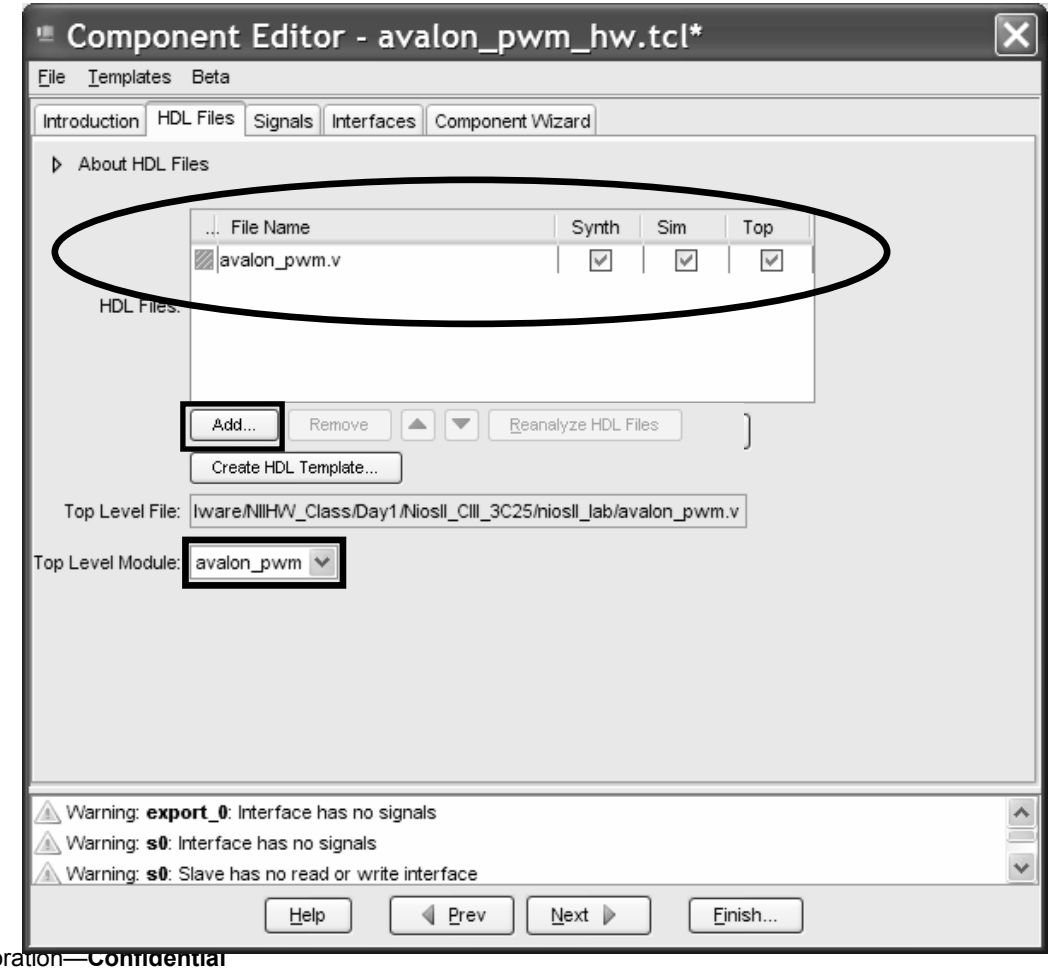
2. Component Authoring from Templates



- Go from Signals Tab back to HDL tab
- Select “Create HDL Template”

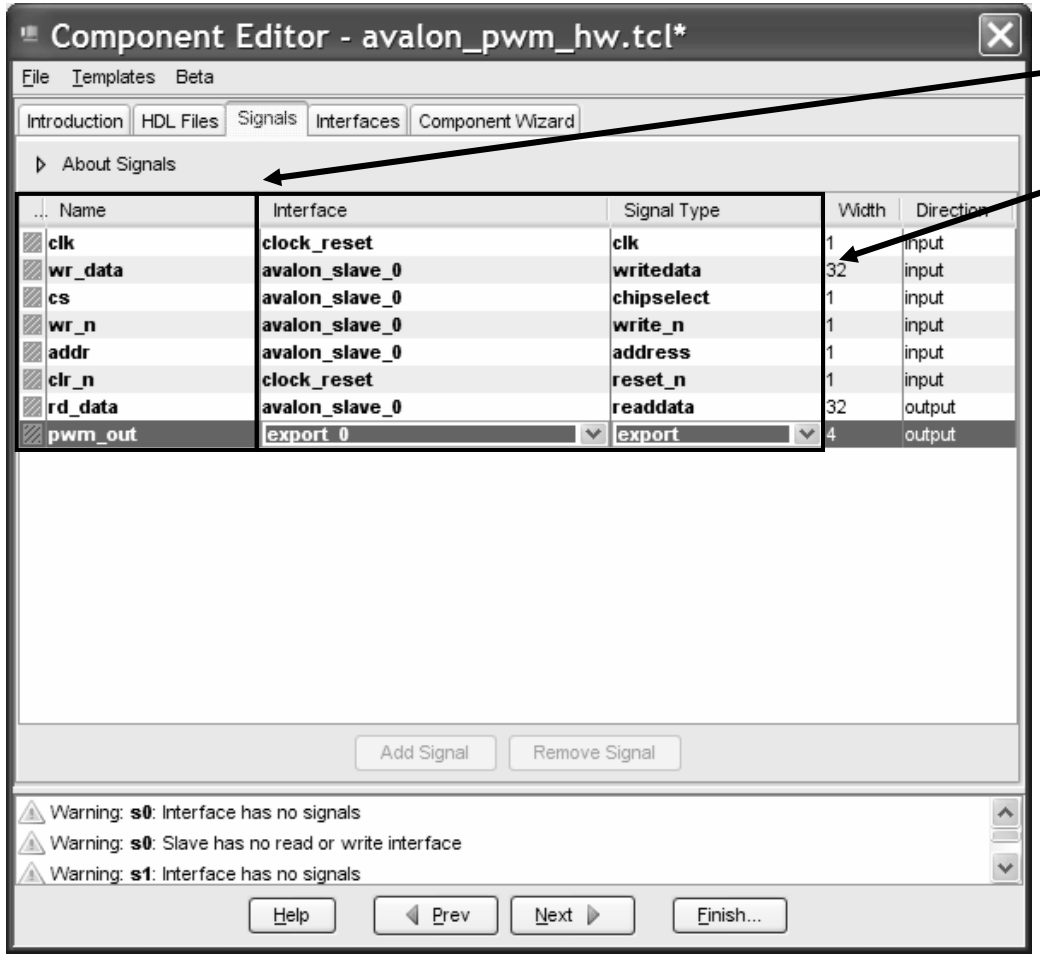
3. Create Interface to HDL File

- **Add HDL File/s** to SOPC Builder system
  - Peripheral will be targeted for synthesis inside FPGA



Uses Quartus II software to analyze design

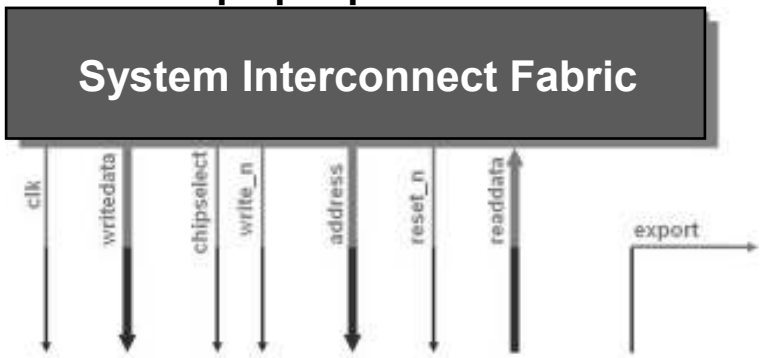
Map Component Signals to Interconnect



Tool automatically populates port **name** column (from HDL file)

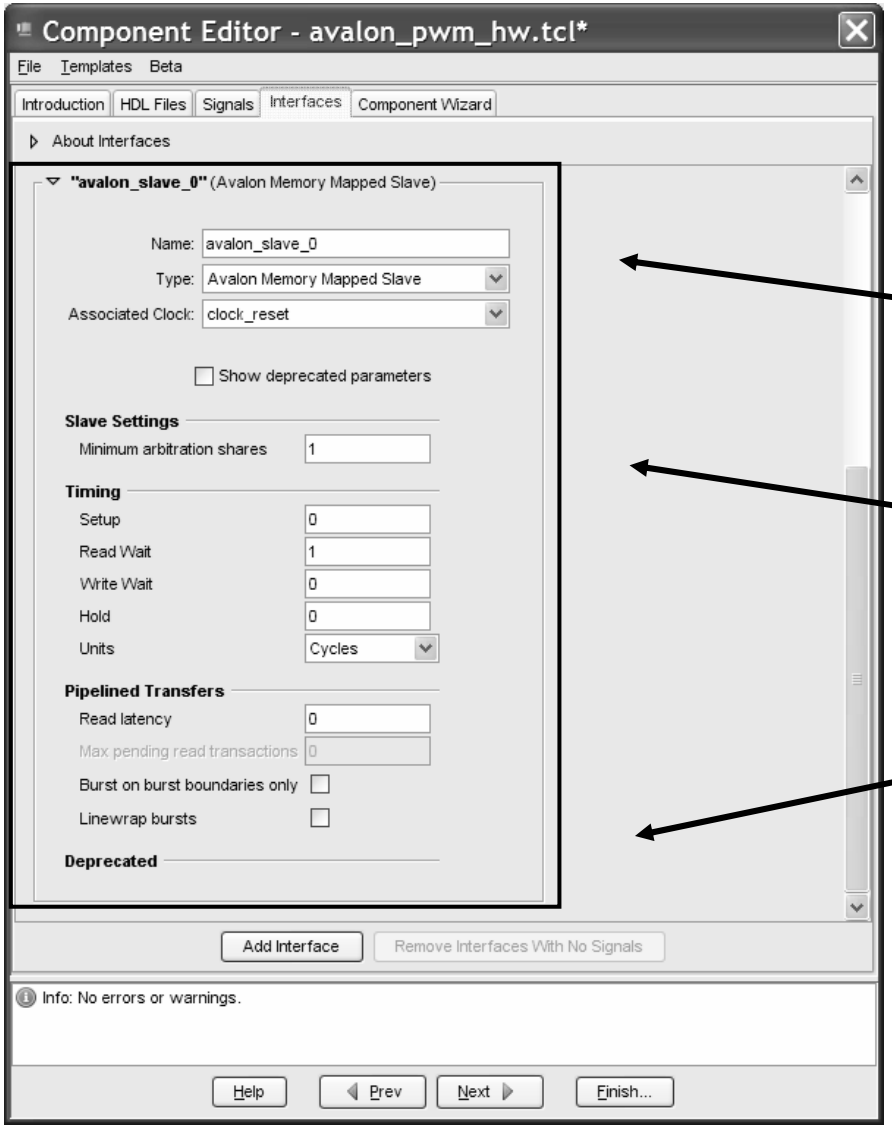
You choose **interface** style and map peripheral's ports to SIF **signal types** (see drop down lists)

This maps peripheral to SIF



Clocks and resets must have interface type "clock" or "clock_reset"

Define Interface Parameters



Exported Signals:

Must be of type **"conduit"** and have associated clock

Clock Inputs

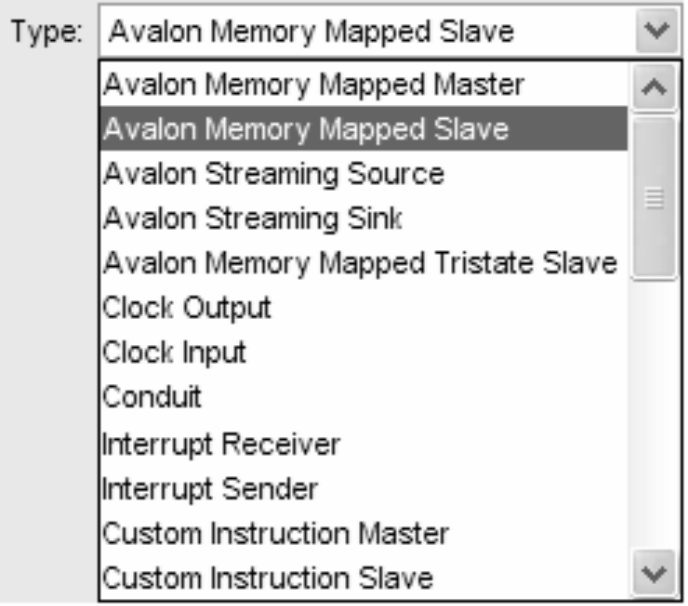
Must be defined as an interface type and feed all clocked interfaces

Avalon Slaves

Must also have associated clock

Can adjust timing parameters

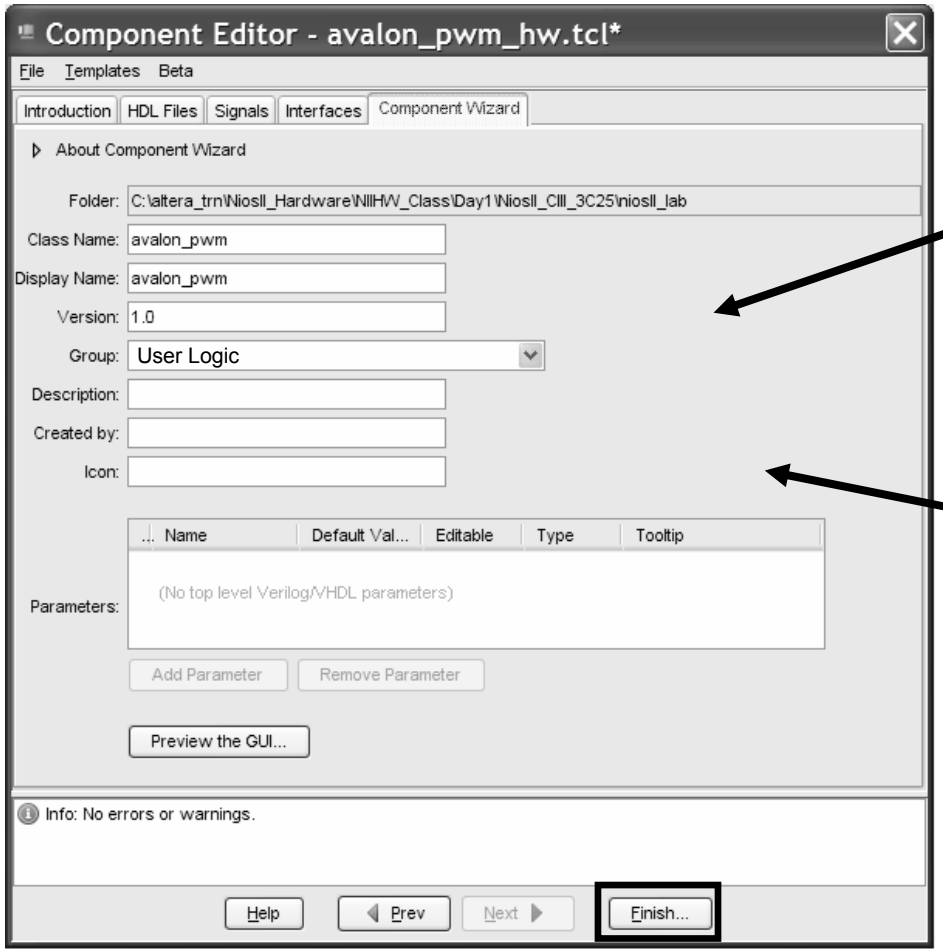
Component Interfaces



- Avalon-MM masters and slaves
- Streaming sources and sinks
  - Define streaming ports for component
- Tristate slaves
  - To connect to off-chip tri-state buses
- Clocks - input and output
  - Clock and reset source for Avalon-MM and Avalon-ST Interfaces
- Conduits – input and output
  - Used for exporting signals
- Interrupts – sender and receiver
  - Used for interrupt support for component
- Custom Instruction Interfaces

Create Component Wizard

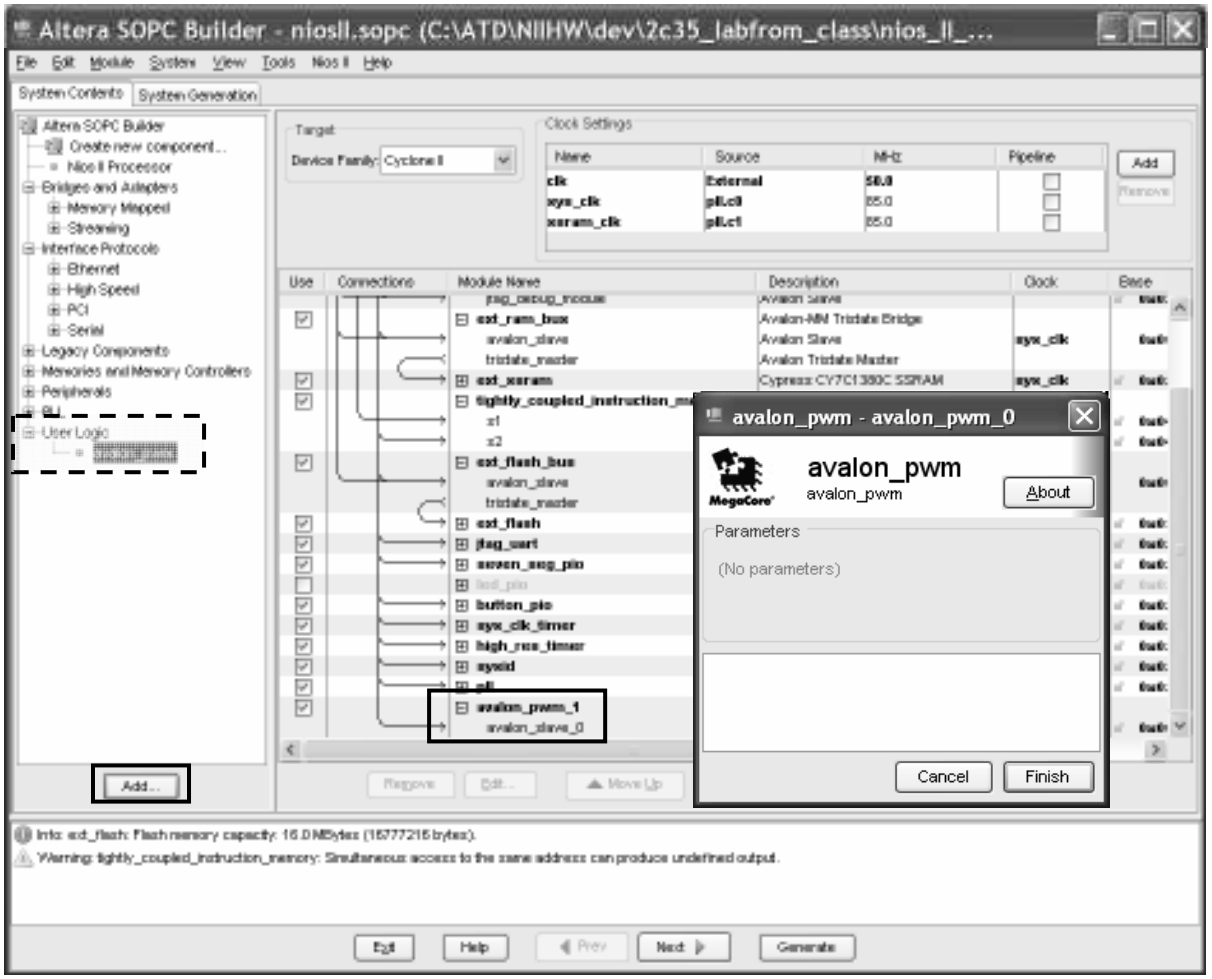
- Publish and create a wizard for your component



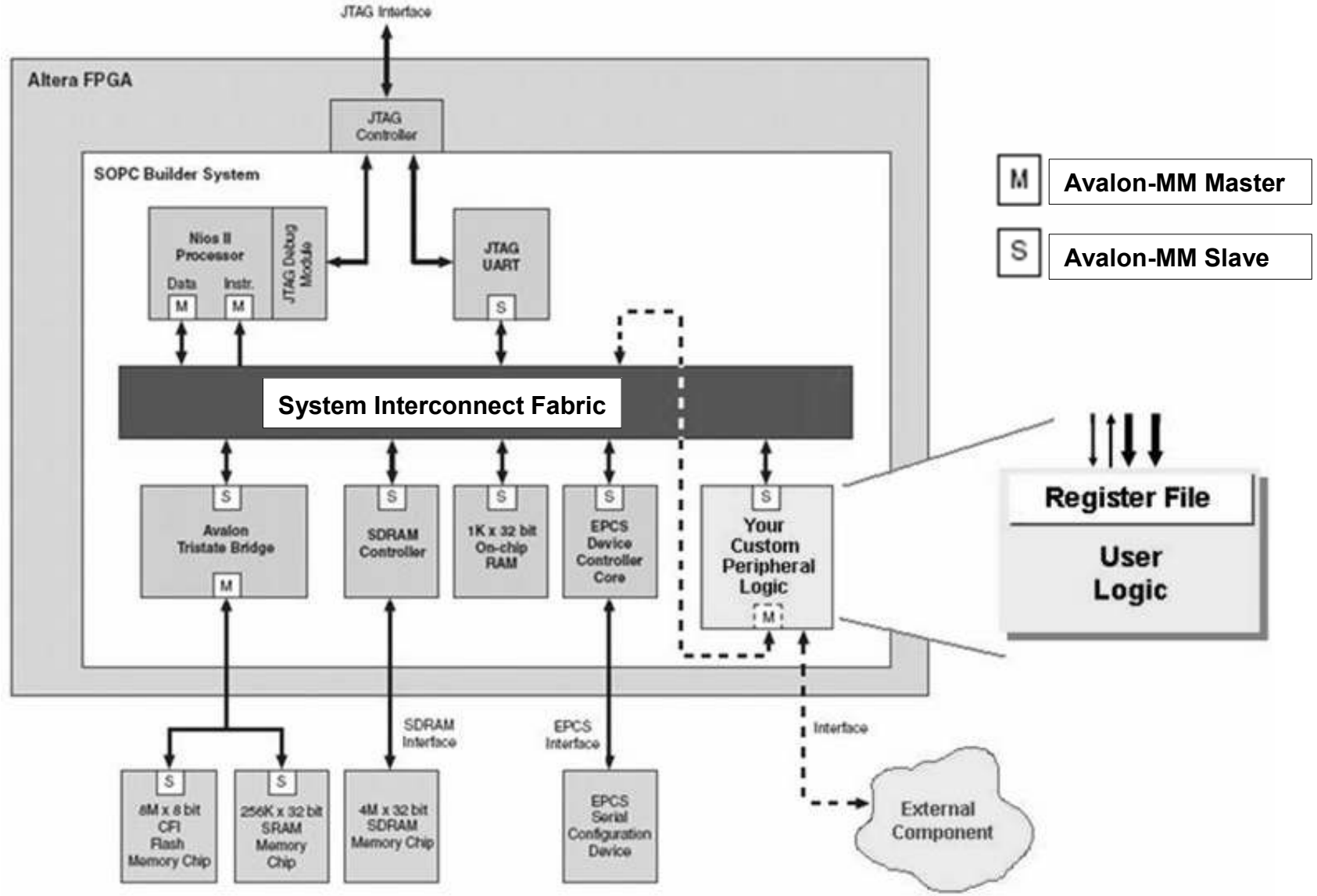
- Define component name
- Select SOPC Builder Component Group folder for component to live
  - Eg. **User Logic**
- Can even accommodate parameterizable HDL components
  - Based on the **GENERICS** or **PARAMETERS** in the VHDL or Verilog code

Add Component to SOPC System

- Locate peripheral *(eg. in **User Logic** folder)*



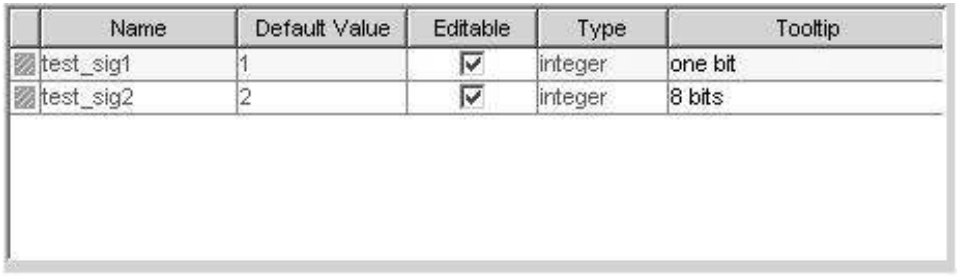
Custom Peripheral Integration Into SIF



Note: Parameterizable Component (Verilog)

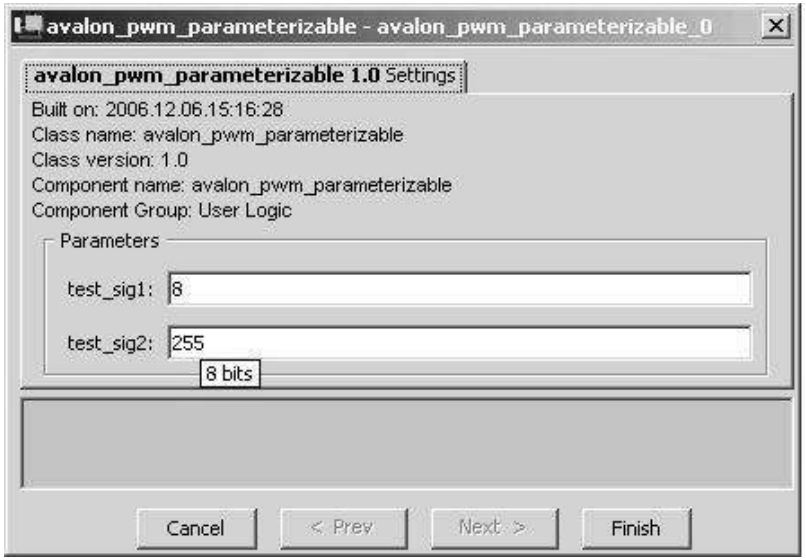
```
module my_peripheral
(
        clk, wr_data, cs, wr_n, addr, clr_n, rd_data, pwm_out, test_out1, test_out2
);
        parameter test_sig1 = 1;   // Initialized (default)
        parameter test_sig2 = 2;   // values of parameters

        output [31:0] rd_data;
        output [7:0] pwm_out;
        output          test_out1;      // Single-bit test output
        output [7:0] test_out2;        // 8-bit test output

        input clk;
        input [31:0] wr_data;
        input cs;
        input wr_n;
                :
                :
```

Component Wizard (Parameters Pane):



	Name	Default Value	Editable	Type	Tooltip
	test_sig1	1	☑	integer	one bit
	test_sig2	2	☑	integer	8 bits

Pass Parameters to Peripheral via Wizard:



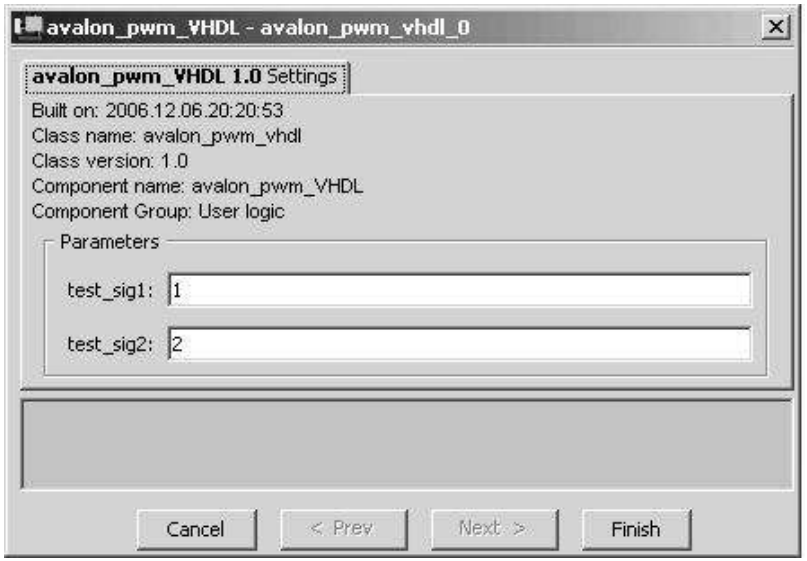
Note: Parameterizable Component (VHDL)

```
entity my_periph is
GENERIC (     test_sig1 : integer := 1 ;
              test_sig2 : integer := 2   );
port (
      test_out1 : out std_logic_vector (0 downto 0) ;
      -- single bit test output
      test_out2 : out std_logic_vector (7 downto 0) ;
     -- 8-bit test output
      clk : in std_logic;
      wr_data : in std_logic_vector (31 downto 0);
              :
              :
);
end my_periph;
              :
              :
```

Component Wizard (Parameters Pane):

	Name	Default Value	Editable	Type	Tooltip
	test_sig1	1	☑	integer	one bit
	test_sig2	2	☑	integer	8 bits

Pass Parameters to Peripheral via Wizard:



Component Editor Output File

- Tcl format file describing component
  - ***<top_level_module>_hw.tcl***
  - Located in same directory as HDL files
- Tcl file can be edited in future *(as can HDL code)* when component must be changed
- Delete Tcl file to remove Custom Component from pick-list

Building Components with Tcl

- Scripting approach now possible
  - Provides a programming interface option
  - Update multiple components w/o accessing each through GUI
- Create a component description file

  <top_level> **_hw.tcl**
  - Store in same directory as HDL code
  - *See Chapter 6, "Hardware Developer's Handbook" and Appendix*

PWM example: ("avalon_pwm_hw.tcl")

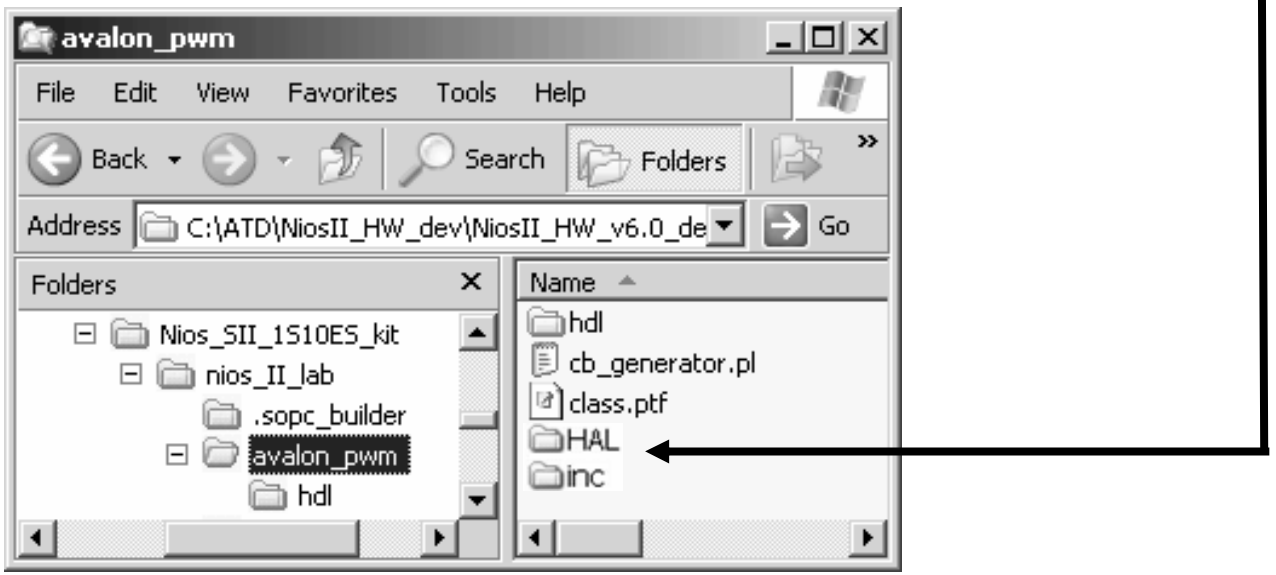
```
add_file                avalon_pwm.vhd {SYNTHESIS SIMULATION}
set_module_property     NAME           avalon_pwm
set_module_property     VERSION        1.0
set_module_property     GROUP          "User Logic"
set_module_property     DISPLAY_NAME   avalon_pwm
                              ⋮
```

More Example Tcl Syntax

```
# | connection point avalon_slave_0
add_interface           avalon_slave_0  avalon end
set_interface_property  avalon_slave_0  holdTime 0
set_interface_property  avalon_slave_0  linewrapBursts false
set_interface_property  avalon_slave_0  minimumUninterruptedRunLength 1
set_interface_property  avalon_slave_0  bridgesToMaster ""
set_interface_property  avalon_slave_0  isMemoryDevice false
set_interface_property  avalon_slave_0  burstOnBurstBoundariesOnly false
set_interface_property  avalon_slave_0  addressSpan 4
set_interface_property  avalon_slave_0  timingUnits Cycles
set_interface_property  avalon_slave_0  setupTime 0
set_interface_property  avalon_slave_0  writeWaitTime 0
set_interface_property  avalon_slave_0  readWaitStates 0
set_interface_property  avalon_slave_0  maximumPendingReadTransactions 0
set_interface_property  avalon_slave_0  readWaitTime 0
set_interface_property  avalon_slave_0  readLatency 0
set_interface_property  avalon_slave_0  ASSOCIATED_CLOCK clock_reset
add_interface_port      avalon_slave_0  wr_data writedata Input 32
add_interface_port      avalon_slave_0  cs chipselect Input 1
add_interface_port      avalon_slave_0  wr_n write_n Input 1
add_interface_port      avalon_slave_0  addr address Input 1
add_interface_port      avalon_slave_0  rd_data readdata Output 32
                                        .
                                        .
                                        .
```

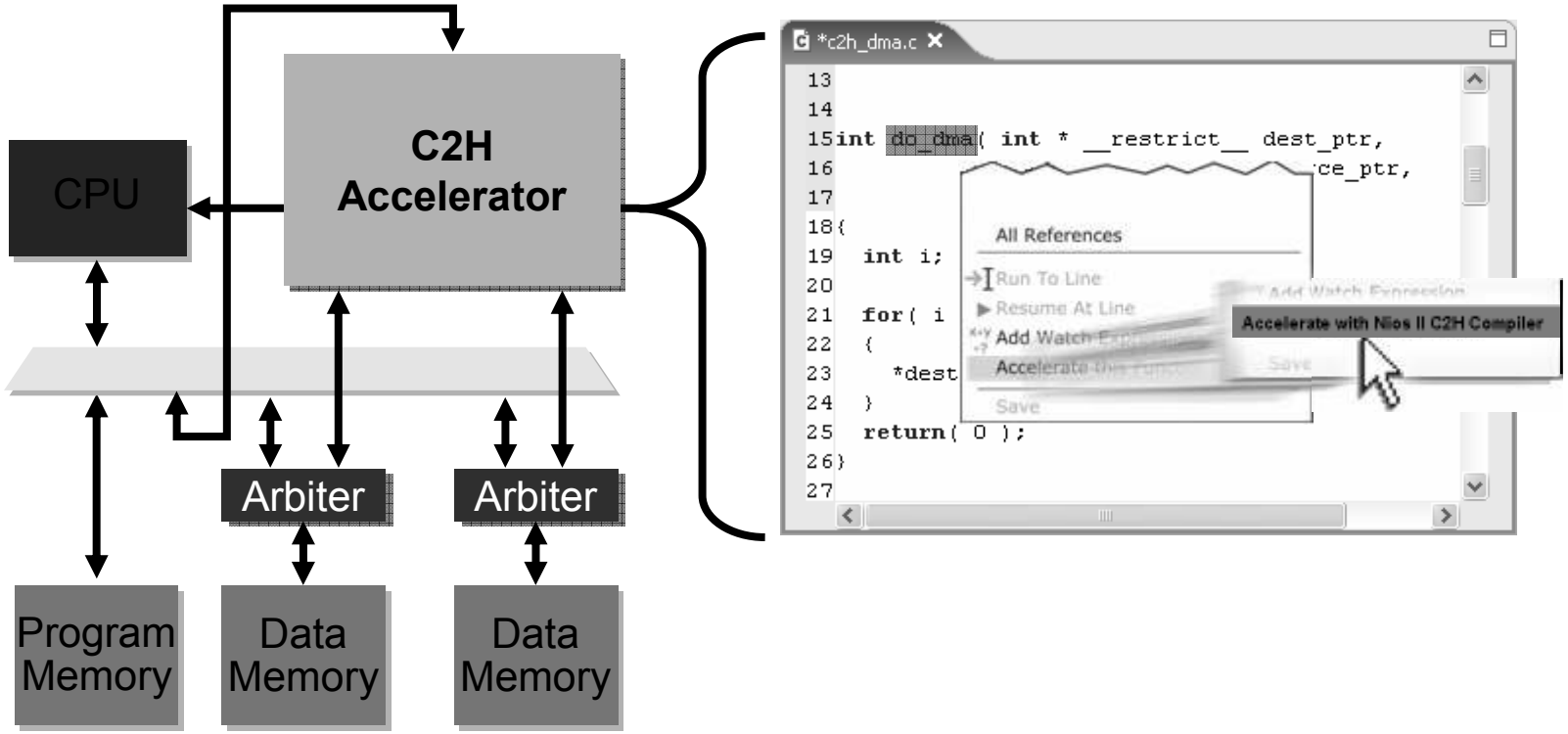
“Legacy” Components

- Can still be read by SOPC Builder
  - SOPC Builder will **not** continue to save components in this format
- Contents:
  - Copies of HDL code
  - Generator Perl script
  - Component “class.ptf” file
  - (Optionally) the device driver folder hierarchy



Foot Note: Nios II C2H Complier

- Generates ***Hardware Accelerated*** Custom Peripheral from ANSI C code, integrating it automatically into SOPC Builder system



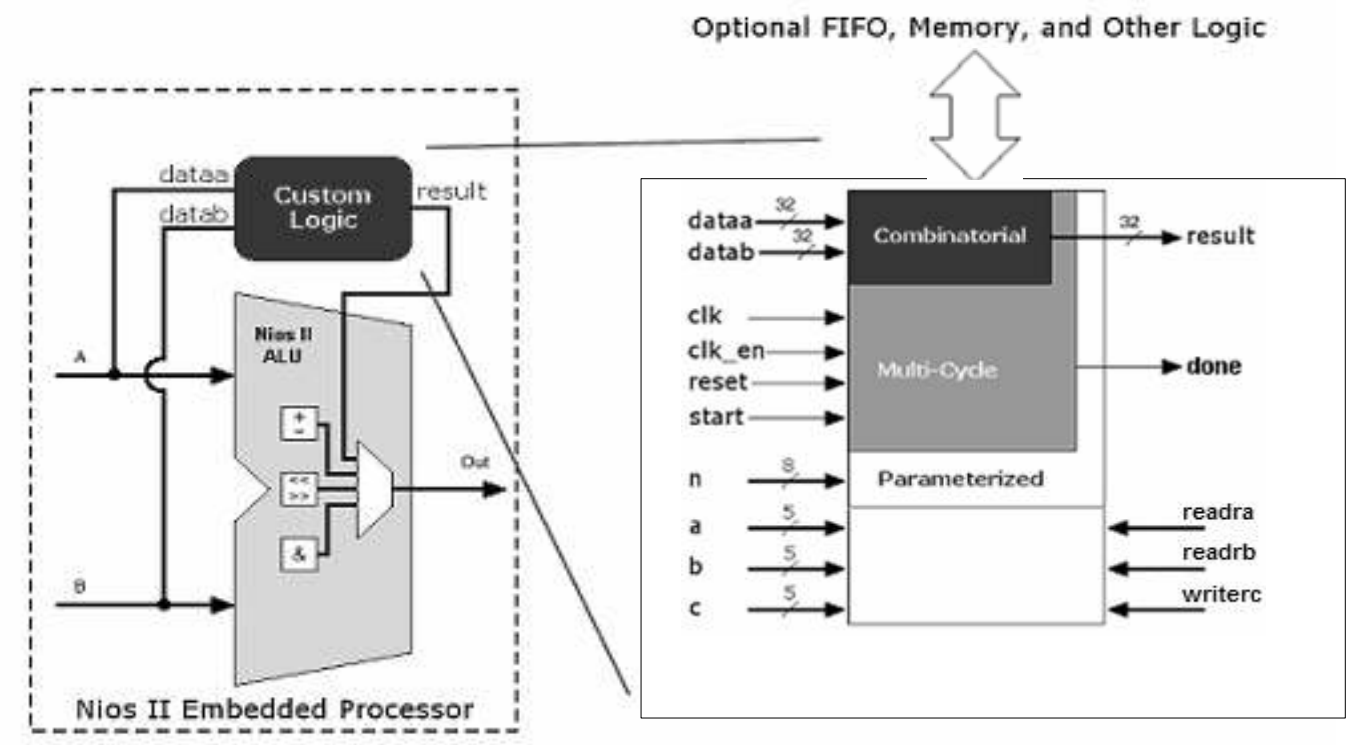
Custom Instructions

Custom Instructions

- Add custom functionality to the Nios II processor design
  - To take full advantage of the flexibility of FPGA
- Dramatically Boost Processing Performance
  - With no Increase in f_{MAX} required
- Application Examples
  - Data Stream Processing (eg. Network Applications)
  - Application Specific Processing (eg. MP3 Audio Decode)
  - Software Inner Loop Optimization

Custom Instructions

- Augment Nios II Processor Instruction Set
  - Mux User Logic Into ALU Path of Processor Pipeline



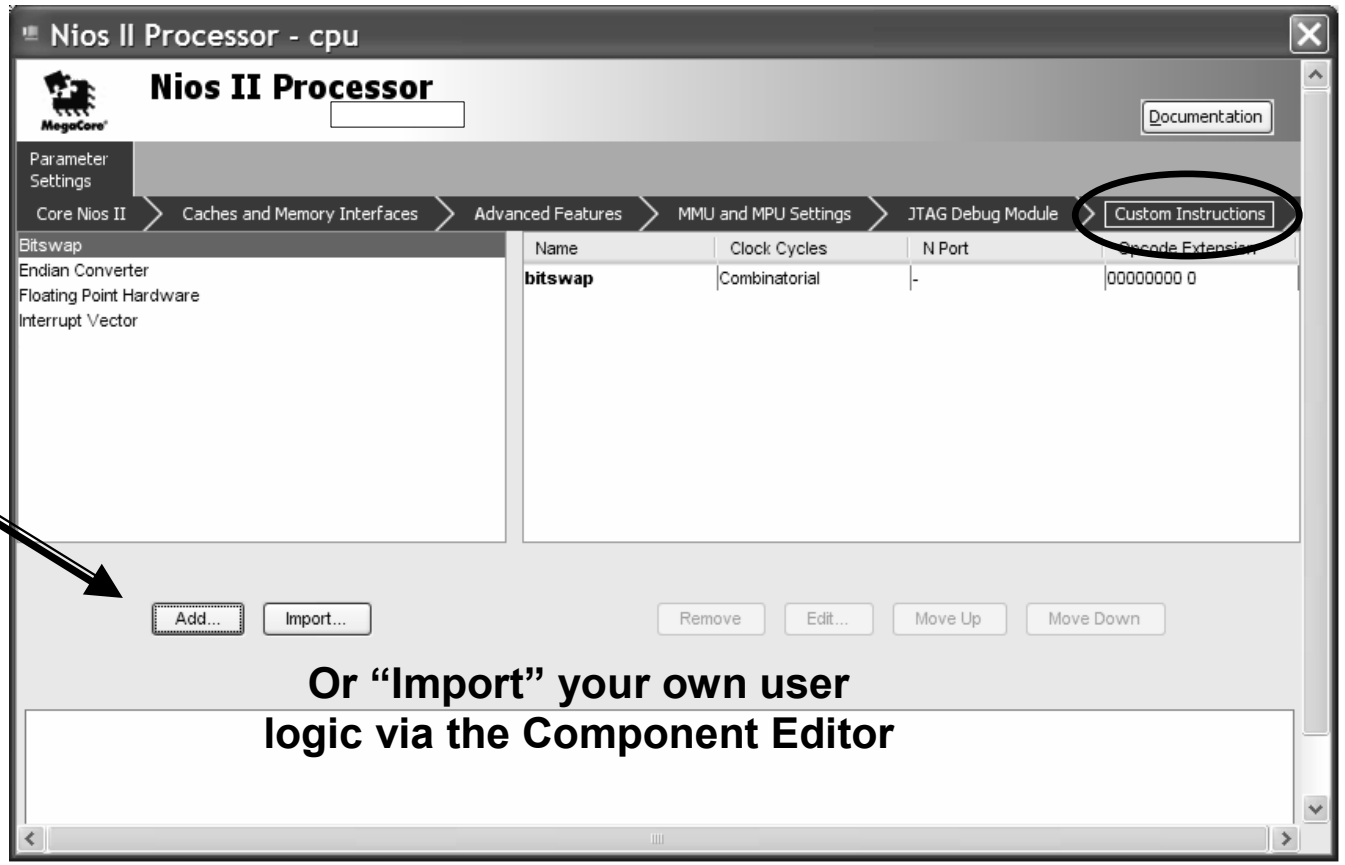
Custom Instructions

- Integrated Into Nios II Processor Development Tools
  - SOPC Builder design tool handles op-code assignment
  - Generates C and assembly-language macros
  - Up to 256 different custom instructions possible
  - Multi-cycle instructions can have variable duration
  - Parameterization of custom instructions has changed

Custom Instructions Tab

- Enabled from the **Custom Instructions** tab in the Nios II CPU Wizard in SOPC Builder

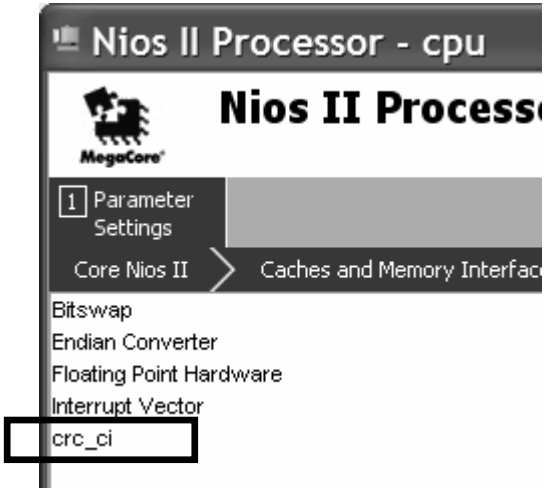
"Add" a custom instruction from built-in library



To Import Custom Instruction

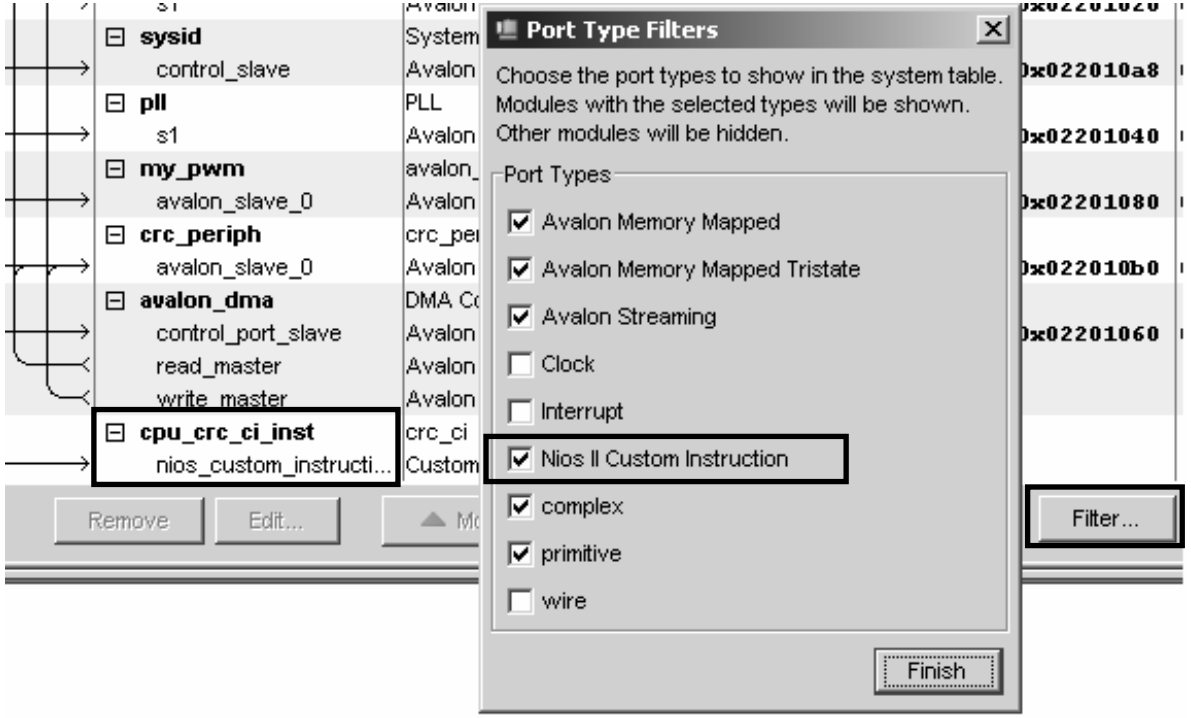
- Use Component Editor
  - Map signals to **nios_custom_instruction** interface
  - Then find new instruction in CPU Custom Instruction Tab
    - May have to close Nios II CPU wizard and re-open again to see new instruction
    - Note: Custom Instruction module can be of following formats:
      - VHDL
      - Verilog HDL
      - EDIF

Name	Interface	Signal Type	Width	Direction
reset	nios_custom_instruction_slave	reset	1	input
clk	nios_custom_instruction_slave	clk	1	input
start	nios_custom_instruction_slave	start	1	input
clk_en	nios_custom_instruction_slave	clk_en	1	input
dataa	nios_custom_instruction_slave	dataa	32	input
datab	nios_custom_instruction_slave	datab	32	input
result	nios_custom_instruction_slave	result	32	output



Note: To Remove Custom Instructions

- Manually delete from project
  - "<custom_instruction>_hw.tcl"
- Remove Custom Instruction from instantiated components list on System Contents page (must "un-filter" to view)



C Language Software Interface

- NIOS II IDE generates macros automatically during build process
- Macros defined in **system.h** file

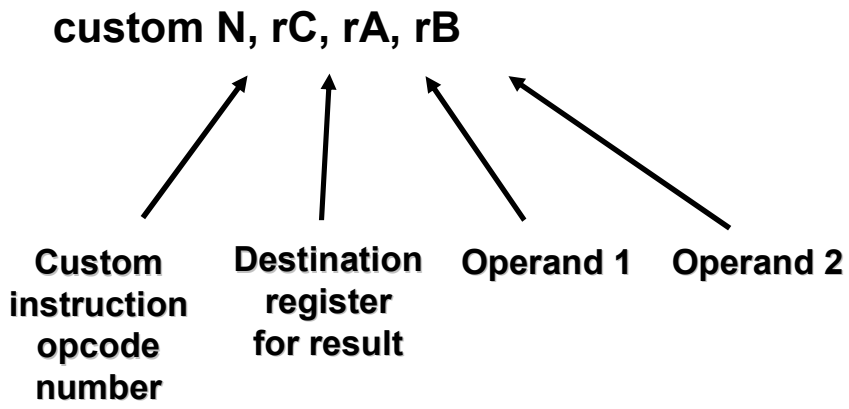
  #define ALT_CI_<your instruction_name>(instruction arguments)

- Example of user C-code that references Bitswap custom instruction:

```
#include "system.h"
int main (void)
{
    int a = 0x12345678;
    int a_swap = 0;

    a_swap = ALT_CI_BSWAP(a);
    return 0;
}
```

Assembly Language Interface

- Assembler syntax for the custom instruction:



- Two Examples:

```
custom 0, r6, r7, r8
custom 3, c1, r2, c4
```

r = Nios II processor register

c = Custom instruction internal register

Why Custom Instruction?

- Reduce Complex Sequence of Instructions to Few or One Instruction
- **Example:** Floating Point Multiply (performed in 6 clock cycles)

> **Roughly 30x performance improvement**
>
> ***Significantly Faster!***

- **Typical Flow**
  - Profile Code
  - Identify Critical Inner Loop
  - Create Custom Instruction Logic
    - Replace One or All Instructions in Inner Loop
  - Import Custom Instruction Logic into Design
  - Call Custom Instruction from C or Assembly

Floating Point Custom Instructions

- Implement single precision floating-point arithmetic operations
  - Use custom instructions to accelerate floating-point operations in your application
- Available on every Nios II processor core
  - Includes single precision floating-point addition, subtraction, multiplication, and division
  - Floating-point division is available as an extension to the basic instruction set

Faster Floating Point Divide

- Uses new Quartus fp_div megafunction
- Up to 2x Fmax increase depending on design and target device
- Increased memory utilization
  - 160,000 memory bits
- Increased latency
  - 32 (26 in version 6.0)

Can You Use Integer Arithmetic Instead?

- While floating-point custom instructions are faster than software-implemented floating-point operations, they are slower than hardware-based integer (ie. *fixed point*) arithmetic
- A common integer technique is to represent numerical values with an implicit scaling factor
  - As a simple example, if you are calculating **milliamps**, you might represent your values internally as **micro-amps** to eliminate decimals

Floating Point CI Macros

- Map to regular arithmetic symbols unless specific pragmas are included in C function
  - The following will force compiler to use software implementation of floating-point operations even if CI FP hardware exists in your system

Addition	#pragma no_custom_fadds
Subtraction	#pragma no_custom_fsubs
Multiplication	#pragma no_custom_fmuls
Division	#pragma no_custom_fdivs

Interrupt Vector CI

- Fully supported by HAL and MicroC/OS-II
- Partial support for Thread/X today and full support down the road (they have a patch)

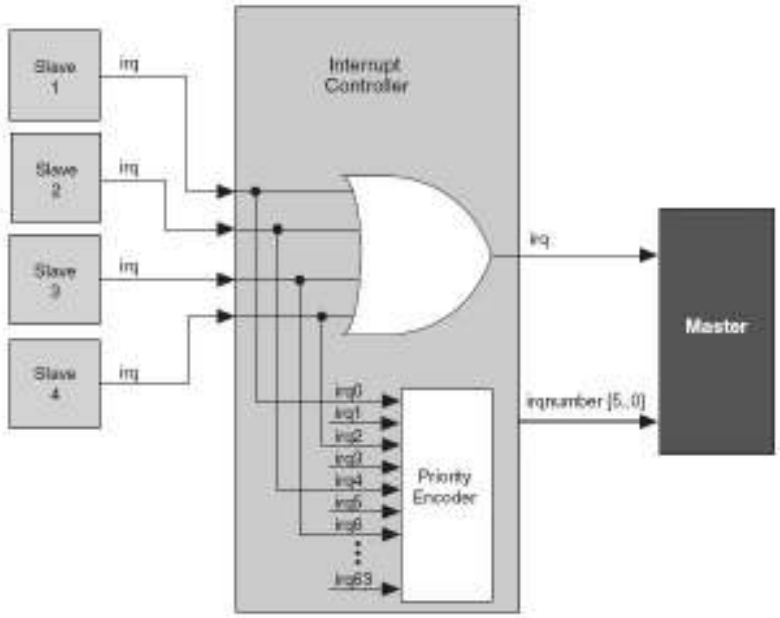
Interrupt Vector – Benefit

- Internal study using TCMs

SW Opt. Level	Without custom instr.	With custom instr.	Latency reduction
-O0	149 cycles	81 cycles	46%
-O3	79 cycles	57 cycles	28%

Interrupt Vector – Cost

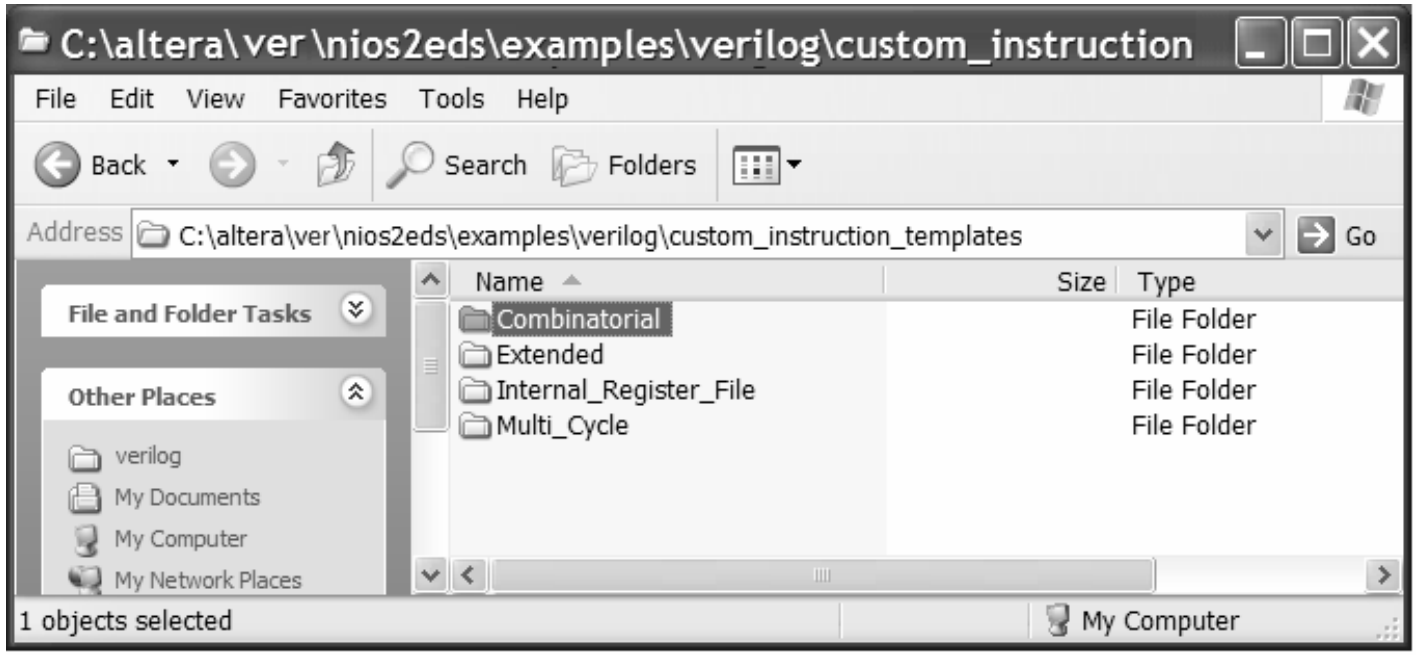
- Priority encoder implemented with muxes
- Depth of muxes proportional to number of interrupts connected to the Nios II processor
  - Cost is just a few LEs
  - May hurt Fmax if there are many interrupts connected



Verilog and VHDL Templates Available

C:\altera\<ver>\nios2*eds*\examples\verilog\custom_instruction_template

C:\ altera\<ver>\nios2*eds*\examples\VHDL\custom_instruction_template



Eg. Combinational Custom Instructions

// EXAMPLE: Verilog Custom Instruction Template File for Combinatorial Logic

```
module custom_instruction(
      dataa,  // Operand A (always required)
      datab,  // Operand B (optional)
      result   // result (always required)
);

// INPUTS
input       [31:0]      dataa;
input       [31:0]      datab;

// OUTPUTS
output      [31:0]      result;

// Custom instruction logic (note:  no external
// interfaces are allowed in combinatorial logic)
                        .
                        .
                        .
endmodule
```

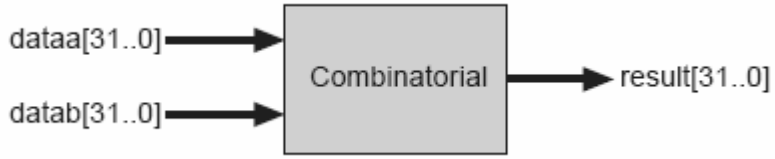
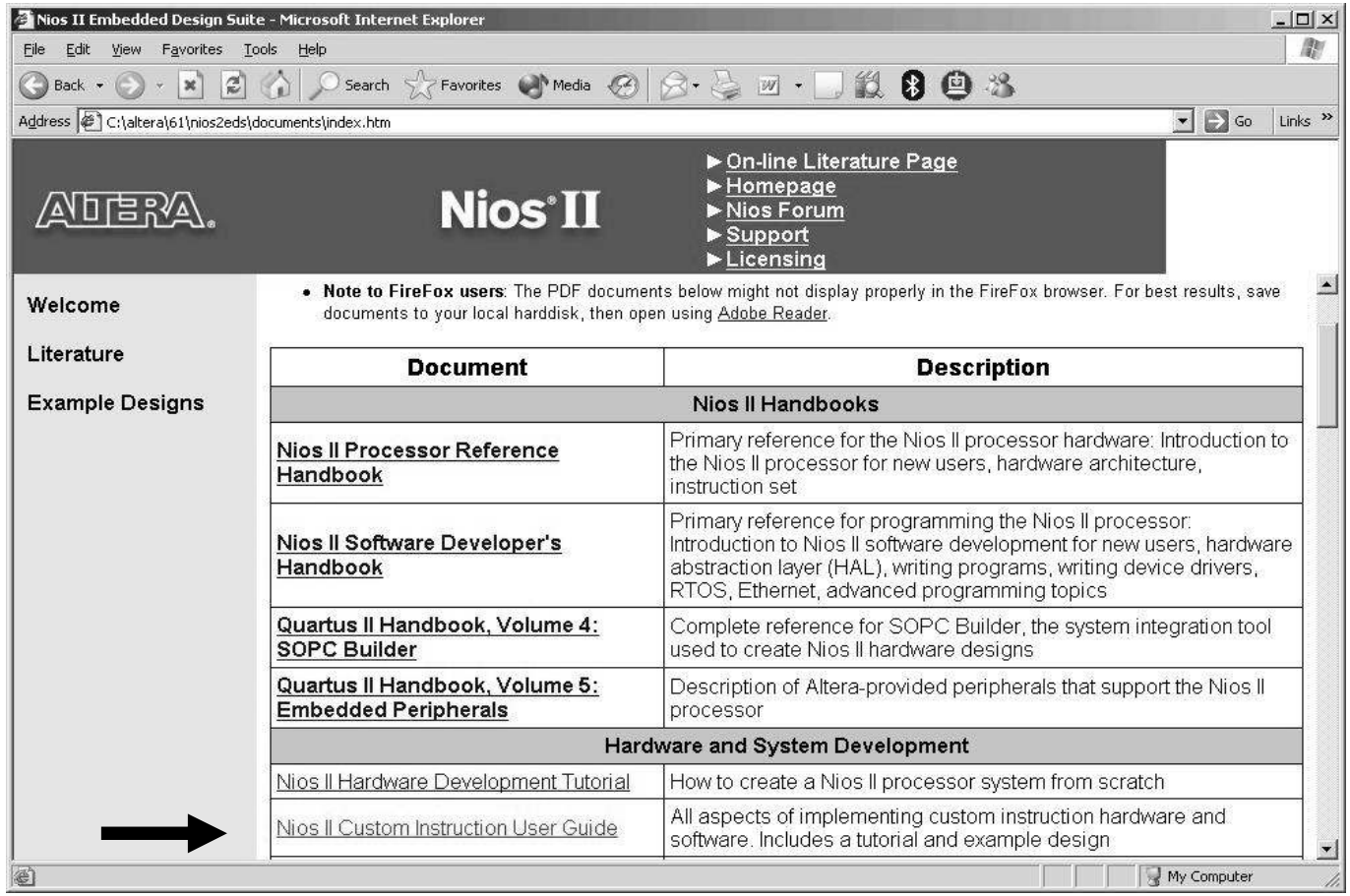


Table 1–2. Combinatorial Custom Instruction Signals

Signal Name	Direction	Required	Purpose
dataa[31..0]	Input	No	Input Operand to custom instruction
datab[31..0]	Input	No	Input Operand to custom instruction
result[31..0]	Output	Yes	Result from custom instruction

Nios II Custom Instruction User Guide

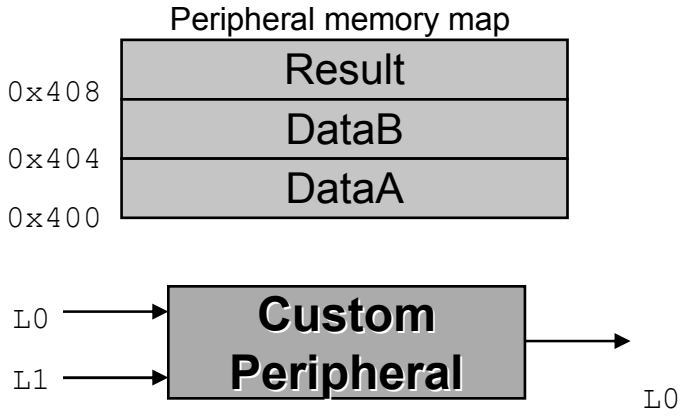
http://www.altera.com/literature/ug/ug_nios2_custom_instruction.pdf



Custom Instruction vs. Peripheral

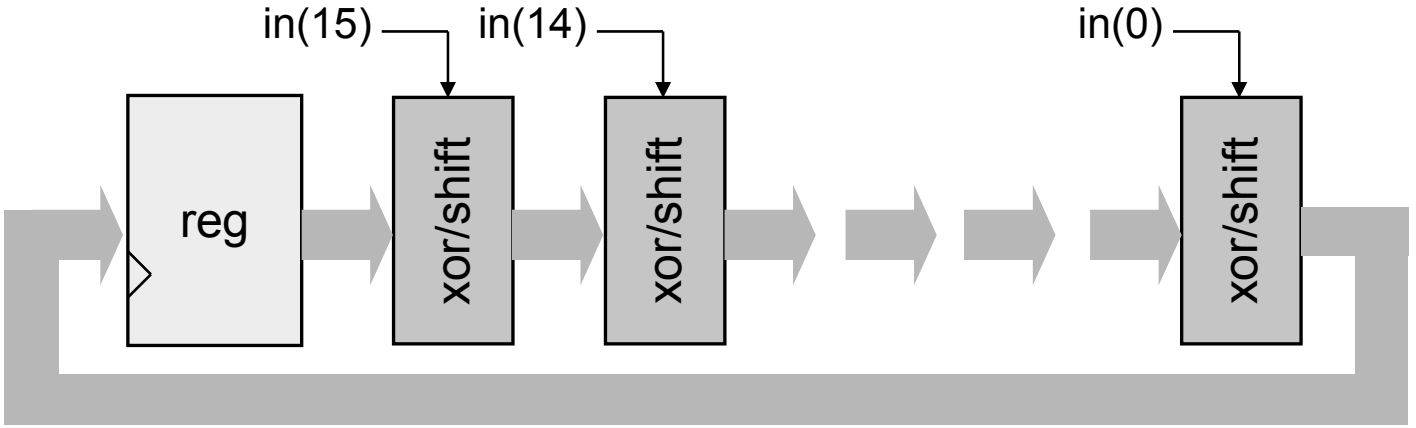
- Custom Instruction can execute in a single cycle
  - No overhead for call to custom Hardware

- Access to same hardware as peripheral takes multiple cycles
  - Write DataA, then write DataB, and finally read Result



Example: Accelerating CRC

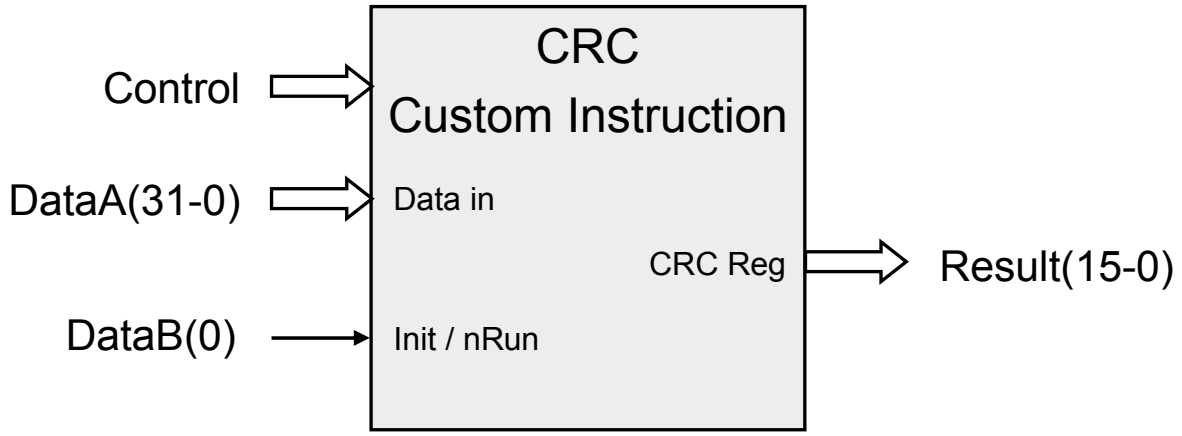
- Implementing the shift and XOR for each bit takes many clock cycles ~50
- Software algorithms tend to use look up tables to pre-compute each byte
- Parallel Hardware is fastest



CRC Custom Instruction

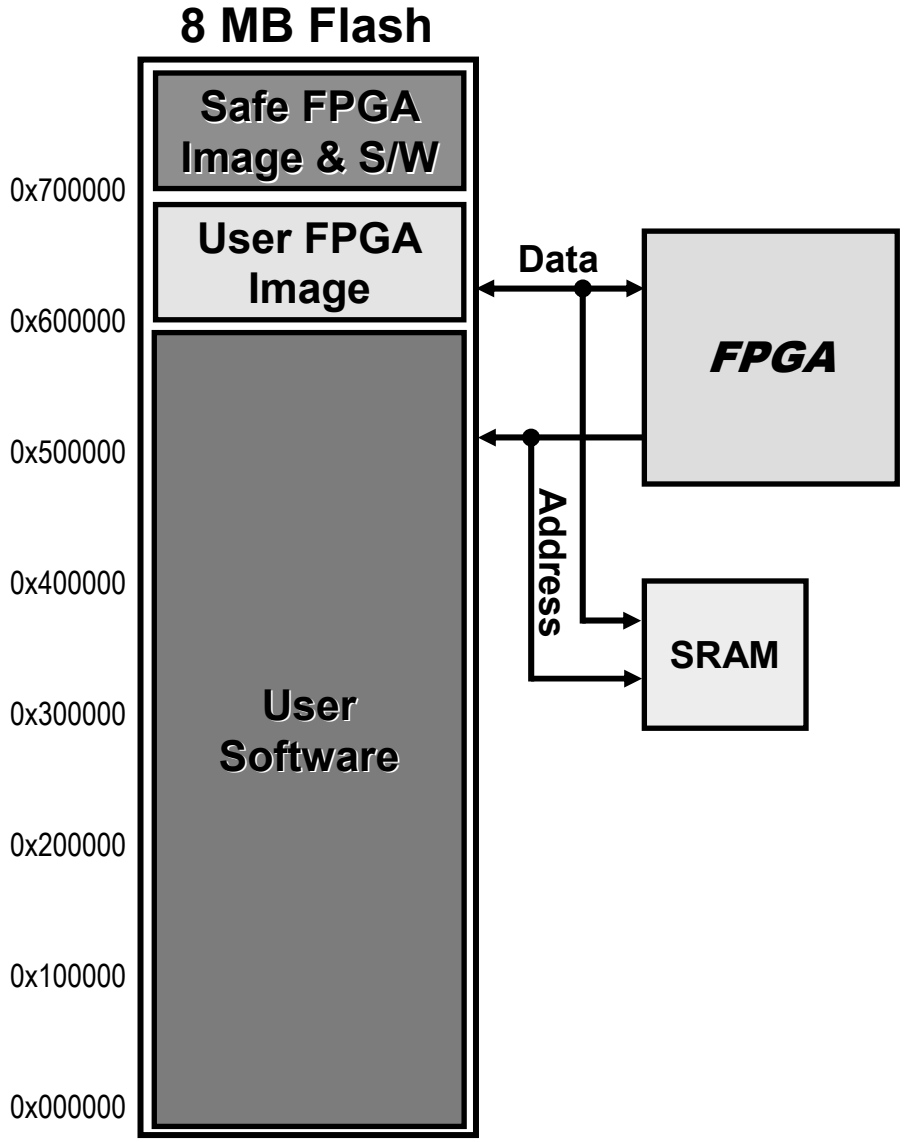
- CRC16-CCITT needs to be preset to 0xFFFF at the start of each computation
- Can use the Data B input to select between run and load
  - Use of prefix would waste a clock cycle

```
// reset crc
ALT_CI_CRC(0xFFFF,1);

// run crc
ALT_CI_CRC(word,0);
```



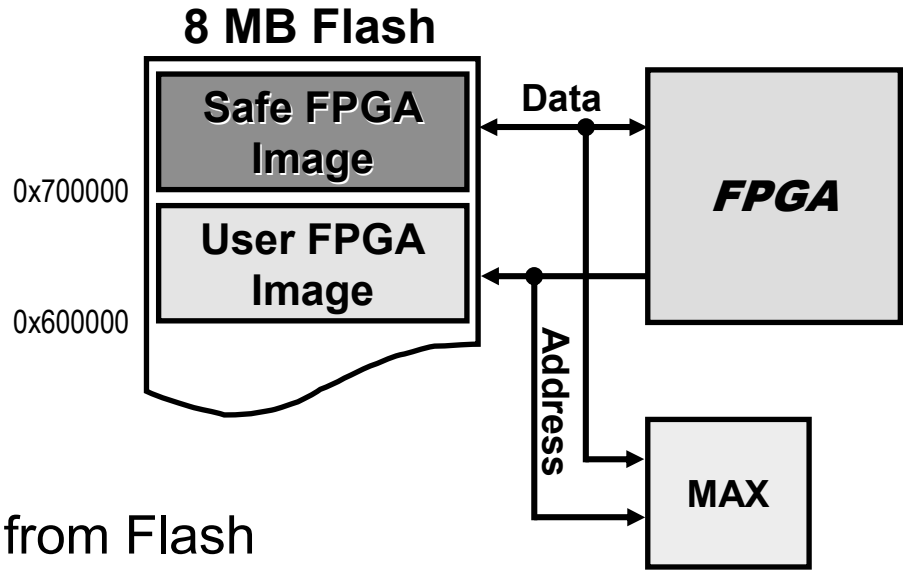
Working with the Development Board

Flash Memory Configuration

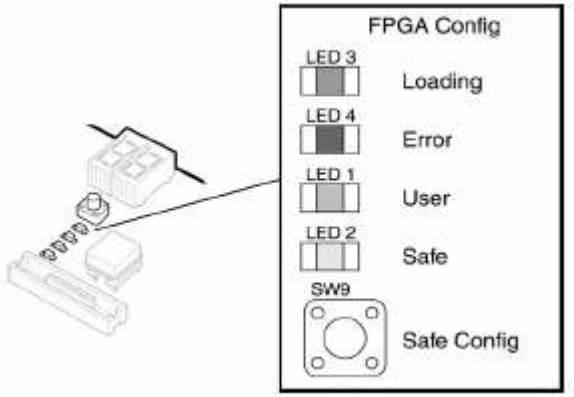


Hardware Configuration Process

- Flash Configuration
  - Two FPGA images
    - Safe Image
    - User Image

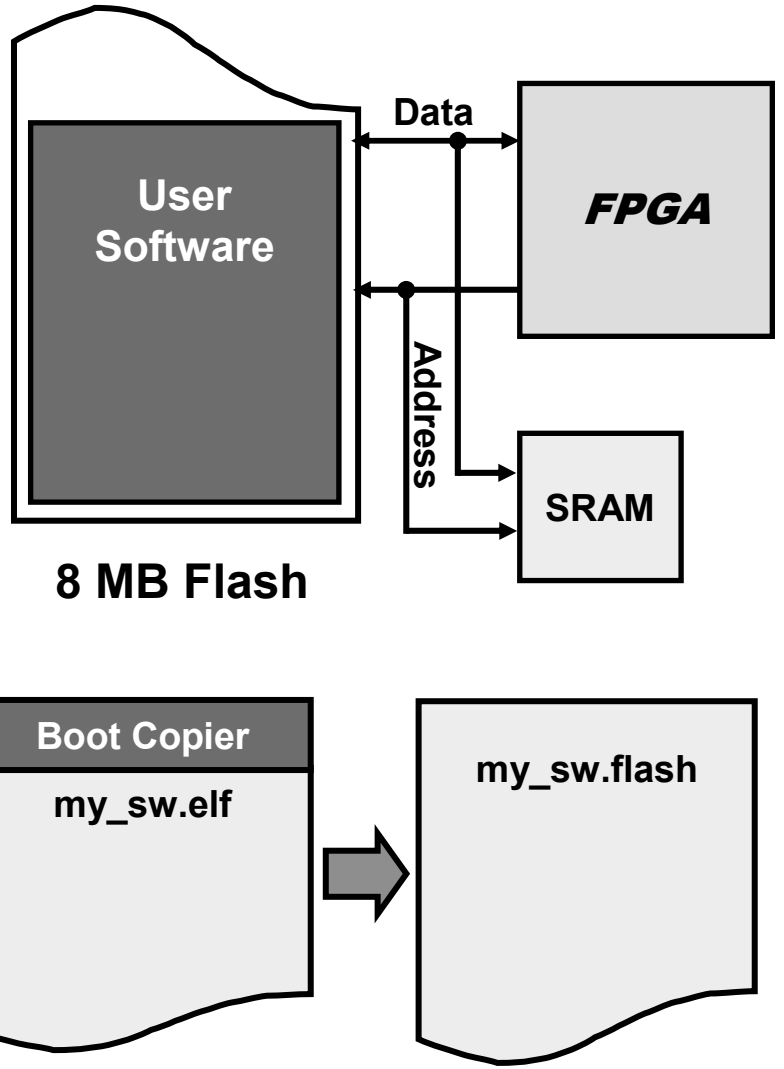


- MAX® EPM7128 Configures FPGA from Flash
  - Upon power up or press of ***Reset Config***
    - MAX Device Loads User Image into FPGA
    - If This Fails MAX Device Loads Safe Image
      - Failure includes no user image present
  - Upon press of ***Safe Config***
    - MAX Device Loads Safe Image into FPGA



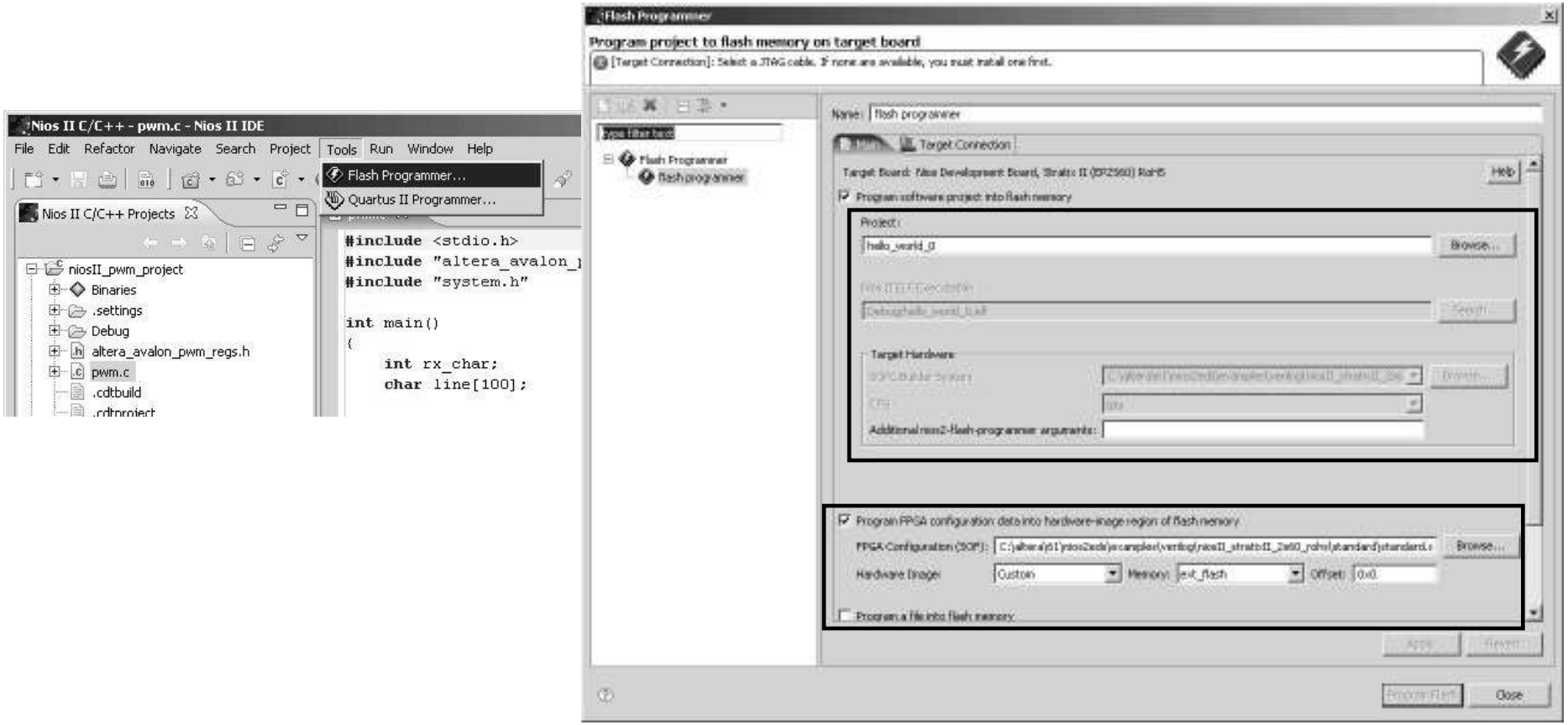
Boot Copier

- **Use Flash for Program Storage**
  - Running from Flash is slow
- **Nios II IDE Flash Programmer Automatically Prepends Boot Copier to Program Code**
  - if Reset Address is in Flash and Program Memory is in RAM



Nios II Flash Programmer

- Can program Flash from Nios II IDE or command line
  - Can be used without an IDE Project
  - GUI supports command line options



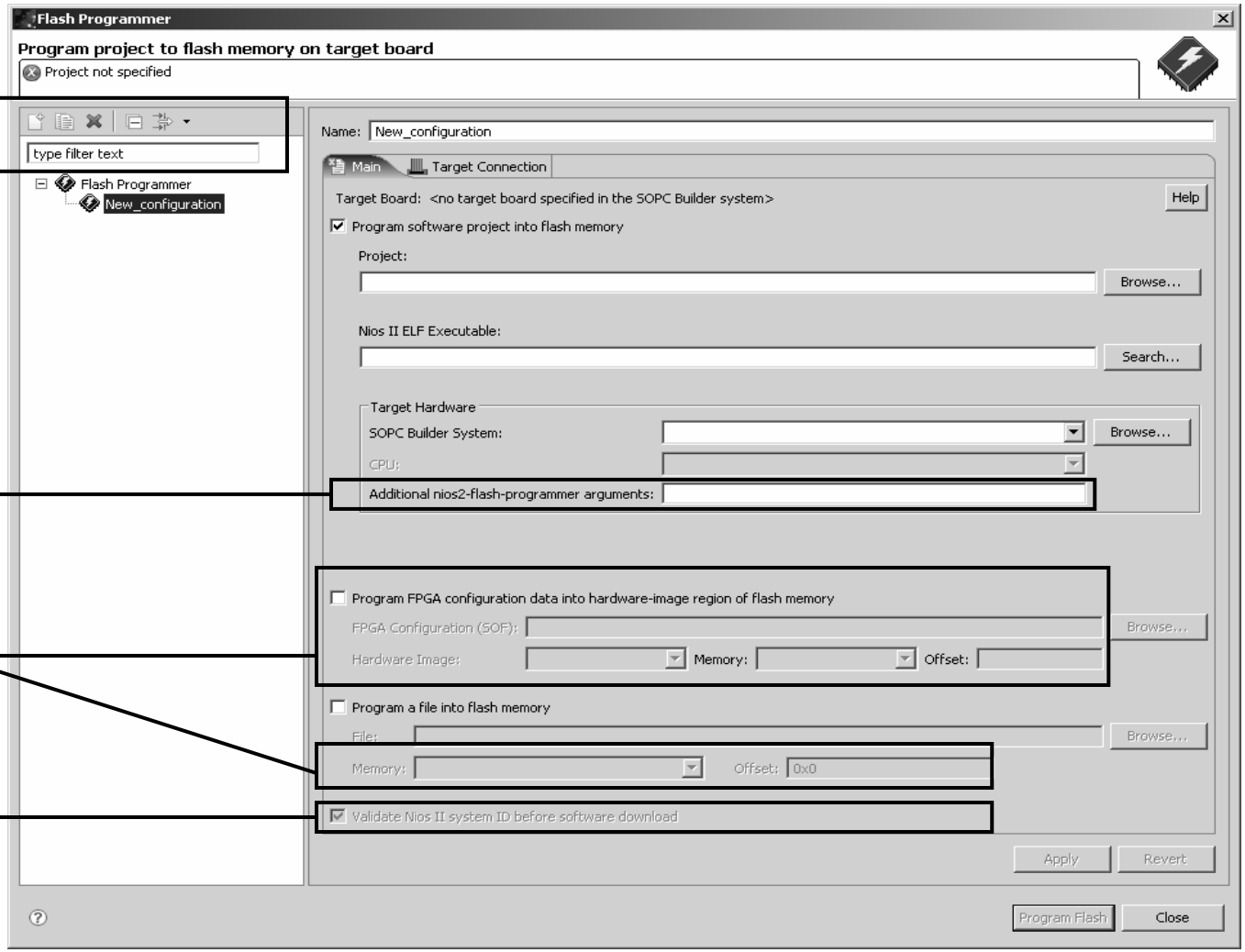
Extra Features

New Eclipse Buttons and filter text

Additional args e.g. --instance=1

Image/Memory/ Offset Selection (allows customization)

Validate SYSID



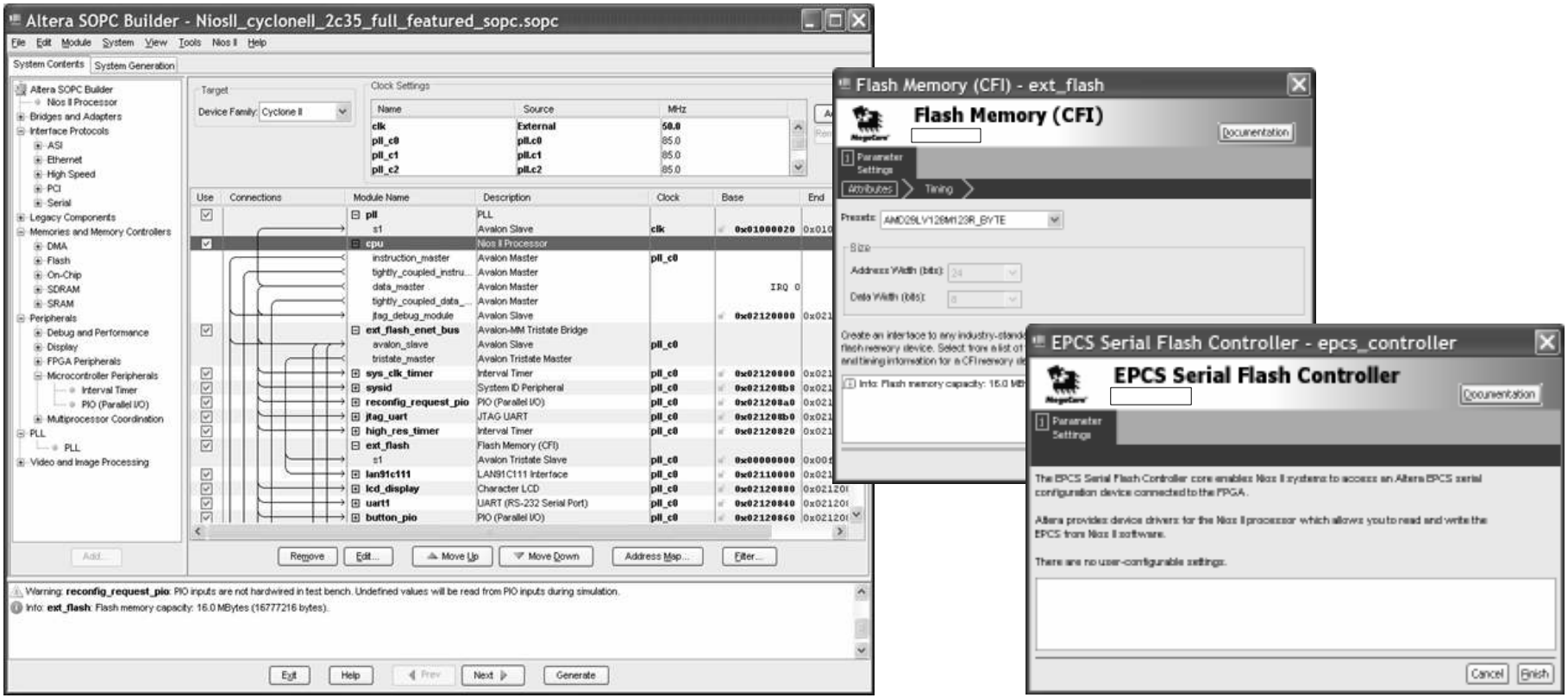
Nios II Flash Programmer

- Command Shell Utilities
  - `elf2flash`
  - `sof2flash`
  - `bin2flash`
  - `nios2-flash-programmer`

*(see **"Nios II Flash Programmer User Guide"**)*

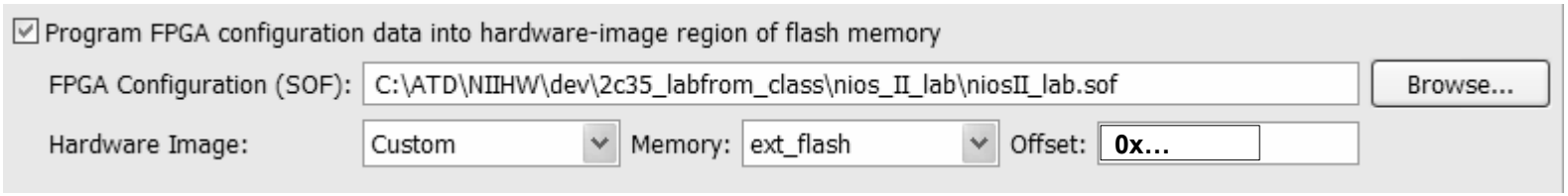
Instantiating Flash in Target System

- Need CFI (Common Flash Interface) Flash Memory
  - No more **Reference Designator** for individual flash components
- EPCS Serial Flash Controller req'd if booting from EPCS device



What if You Have a Custom Board?

- Just ensure that your design has CFI flash peripheral and real CFI compliant flash chip on the board
  - Can customize offset of your flash in SOPC Builder design



- Target design also requires Nios II processor with at least Level 1 JTAG Debug core
  - Flash programming step utilizes this core
- And a tri-state bridge peripheral to access the off-chip bus
  - See new *Nios II Flash Programmer User Guide* for details

What If Factory Safe Flash Image Overwritten?

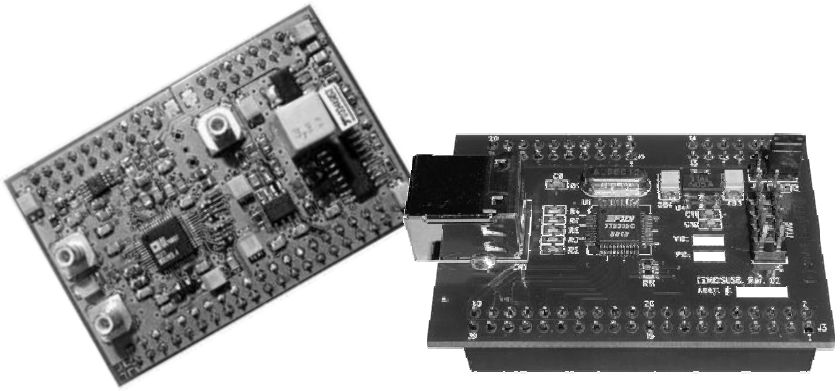
- Open Nios II Command Shell
  - **Start > Programs > Altera > Nios II EDS** <*version*> **> Nios II Command Shell**
- Change to factory-recovery directory for your development kit
  - cd examples/factory_recovery/niosII_cyclone_1c20
- Run flash-restoration script
  - ./restore_my_flash
- Follow the script's instructions

Diverse Portfolio of Altera Development Kits

- eg. Stratix II, Cyclone II FPGAs
  - Altera ROHS kits now available
    (Reduction of Hazardous Materials)
  - Microtronix

- Daughter Cards
  - Microtronix: VGA / PS2
  - SLS: USB 2.0
  - El Camino GmbH: RF A/D D/A
  - EasyFPGA: USB 2.0

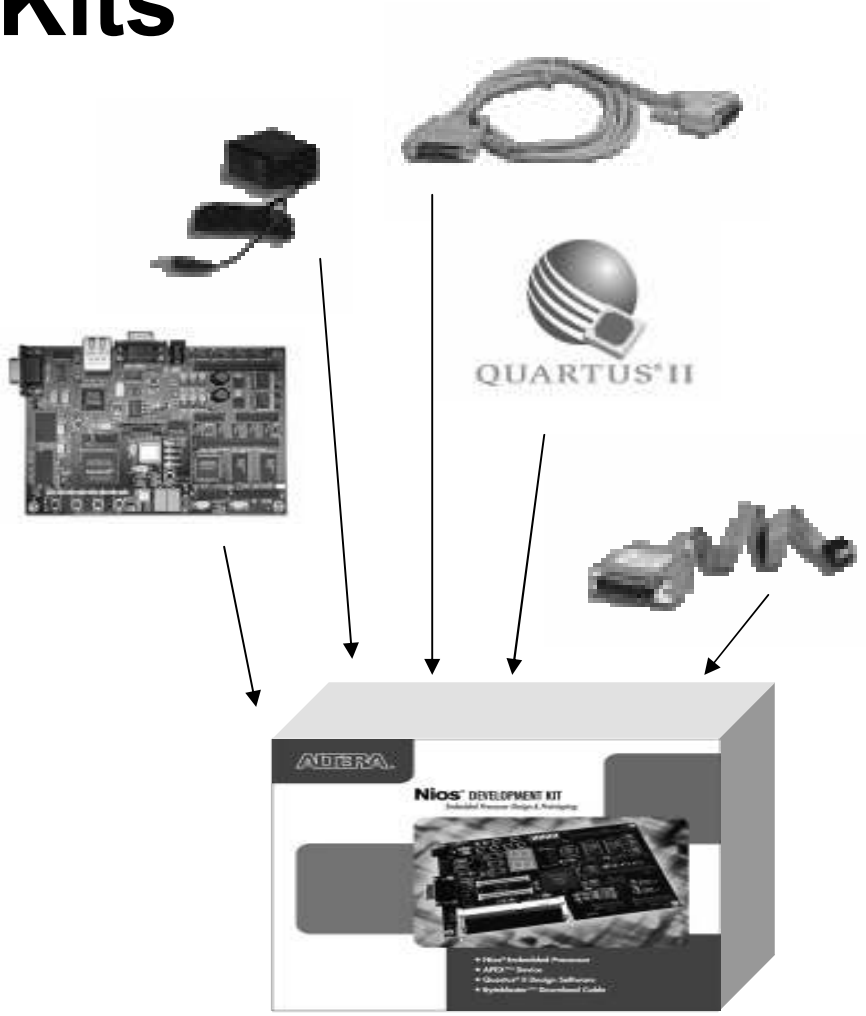
The List Keeps Growing



Altera Development Kits

- Focus On Quality & Completeness
  - All boards are fully tested and verified before shipment
  - Accompanied by accurate, technical documentation
- Provide a Complete Design Environment
  - Board w/featured Altera device
  - Quartus II software (DKE version)
  - “Kit” CD with reference designs and utilities
  - Cables and accessories as necessary
  - OOBE (out of Box experience)

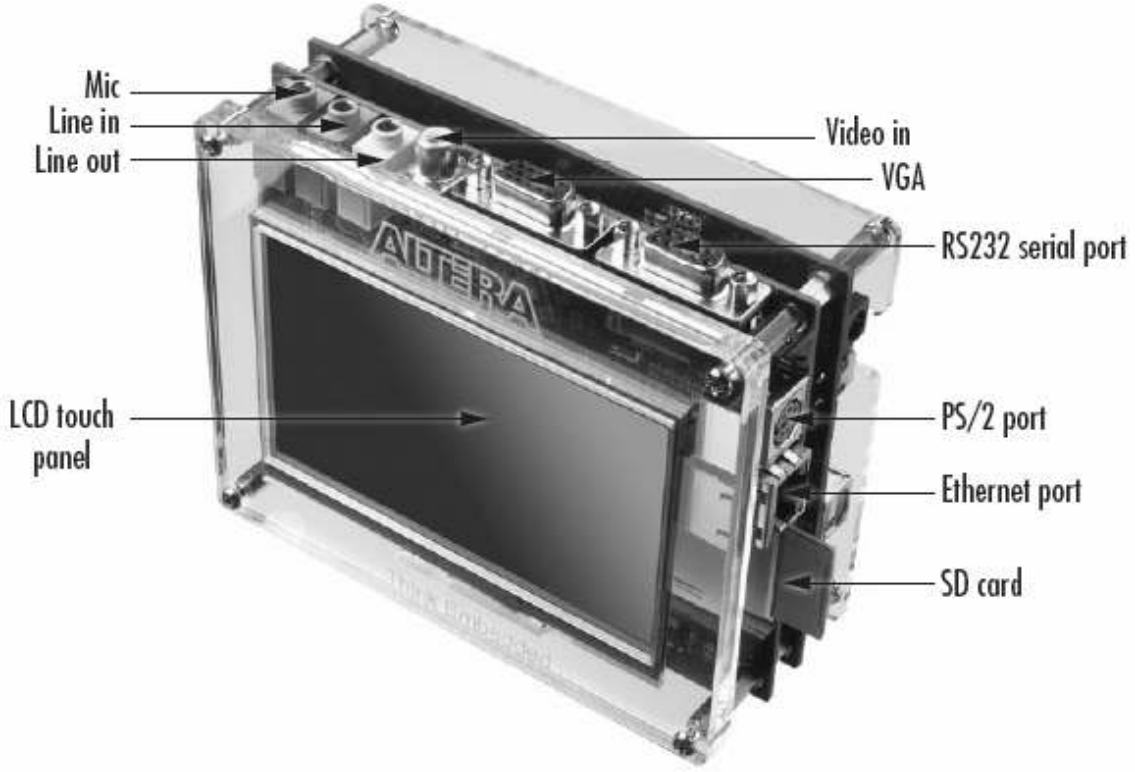
Just Add Electricity!



It’s all in the box!

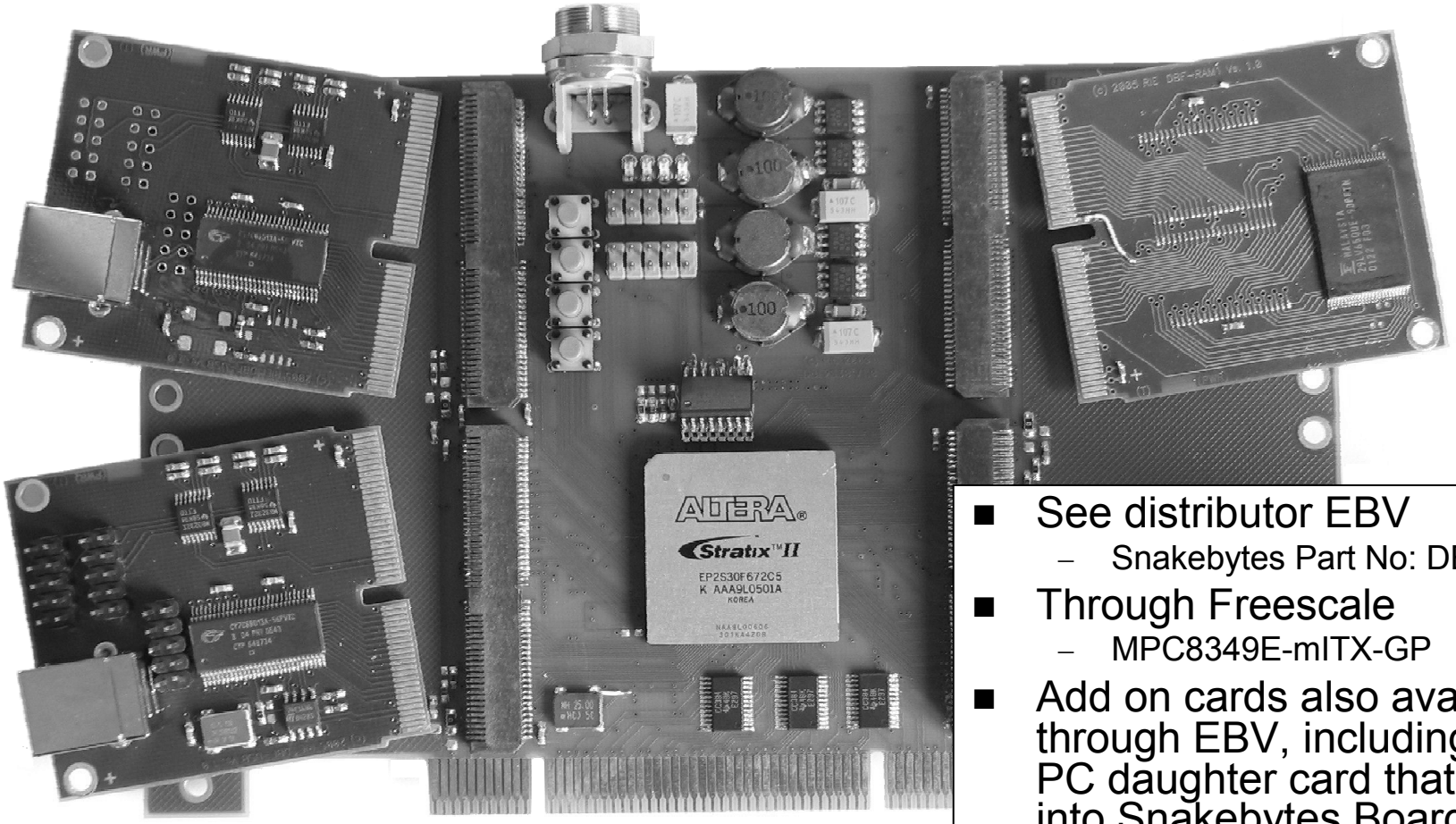
Nios II Embedded Evaluation Kit (NEEK)

- Multi-board evaluation platform now available
  - "Cyclone III FPGA Starter Board"
  - Embedded LCD/VGA HSMC Daughter Card



SnakeBytes: DBF2S30 Board

- Demonstrates Nios II / Power QUICC II Co-Processing



- See distributor EBV
  - Snakebytes Part No: DBF2S30
- Through Freescale
  - MPC8349E-mITX-GP
- Add on cards also available through EBV, including Power PC daughter card that plugs into Snakebytes Board

For Complete List of Dev Kits

- **Refer to www.altera.com:**

  > Products > Dev Kits / Cables

Product Name	Device	Price
Arria GX Development Kit	Arria GX 1AGX60DF780	$ 995
Audio Video Development Kit, Stratix II GX Edition	Stratix II GX 2SGX90	$4,995
Cyclone II Starter Development Kit	Cyclone II 2C20	$ 150
Cyclone III FPGA Development Kit	Cyclone III 3C120N	$1,495
Cyclone III Starter FPGA Kit	Cyclone III 3C25	$ 199
DSP Development Kit, Cyclone II Edition	Cyclone II 2C70	$ 995
DSP Development Kit, Cyclone III Edition	Cyclone III EP3C120N	$1,895
DSP Development Kit, Stratix II Edition	Stratix II 2S60	$1,995
DSP Development Kit, Stratix II Professional Edition	Stratix II 2S180	$5,995
DSP Development Kit, Stratix III Edition	Stratix III 3SL150	$2,895
MAX II Development Kit	MAX II 1270	$ 150
Nios II Development Kit, Cyclone II Edition (2C35)	Cyclone II 2C35	$ 995
Nios II Development Kit, Stratix II Edition	Stratix II 2S60	$ 995
Nios II Embedded Evaluation Kit, Cyclone III Edition	Cyclone III 3C25	$ 449
PCI Development Kit, Cyclone II Edition	Cyclone II 2C35	$ 995
PCI Express Development Kit, Stratix II GX Edition	Stratix II GX 2SGX90	$2,995
Stratix III FPGA Development Kit	Stratix III 3SL150	$2,495
Transceiver Signal Integrity Development Kit, Stratix II GX Edition	Stratix II GX 2SGX90	$1,295
Video Development Kit, Cyclone II Edition	Cyclone II 2C70	$1,095