




EDK Training at University of Toronto

Processor IP Team
November 2003

The background of the slide features a blue circuit board, likely an FPGA, positioned diagonally. Overlaid on the bottom right is a map of North America, with the United States and Canada visible. The entire slide has a light blue, textured background with faint binary code (0s and 1s) scattered across it.

EDK Training at University of Toronto

Processor IP Team
November 2003

The background of the slide features a blue circuit board, likely an FPGA, positioned diagonally. Overlaid on the bottom right is a map of North America, with the United States and Canada visible. The entire slide has a light blue, textured background with faint binary code (0s and 1s) scattered across it.

EDK Training at University of Toronto

Processor IP Team
November 2003

The background of the slide features a blue-tinted image. In the upper right, there is a close-up of a circuit board, likely a Xilinx EDK board, showing its intricate components and connectors. Below and to the left of the board, there is a faint, stylized map of North America, specifically highlighting the United States and Canada. The overall aesthetic is technical and professional, consistent with a presentation from a processor IP team.

Table of Contents

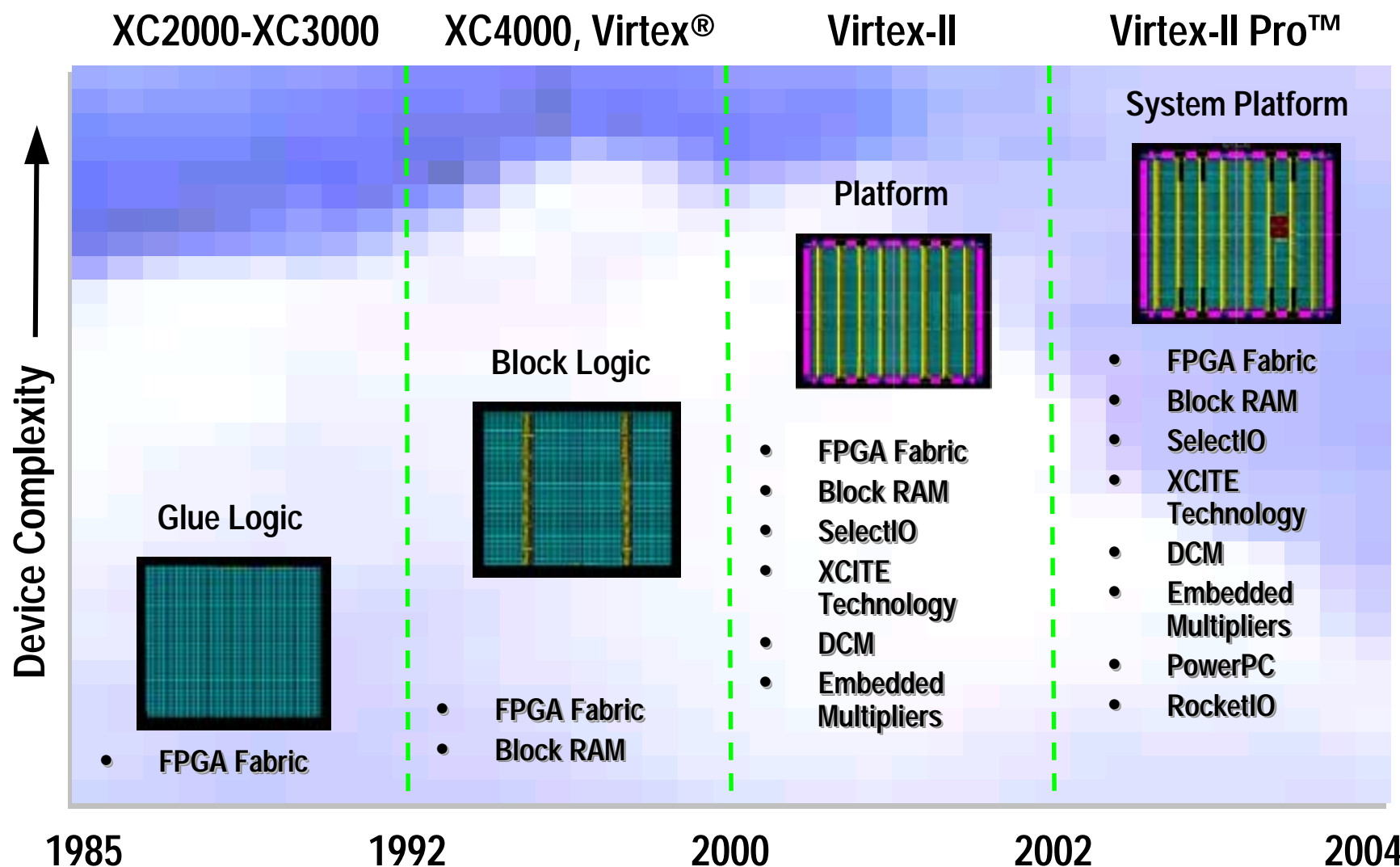
Introduction and Overview	Page 3
PowerPC and MicroBlaze	Page 11
Processor IP	Page 19
Creating a Simple MicroBlaze System with XPS with Base System Builder	Page 49
Creating a Simple MicroBlaze System with XPS without Base System Builder	Page 79
Adding I/O Peripherals to a System	Page 96
Software Development with the EDK and XPS	Page 110
Device Drivers & Software Infrastructure	Page 119



Introduction and Overview

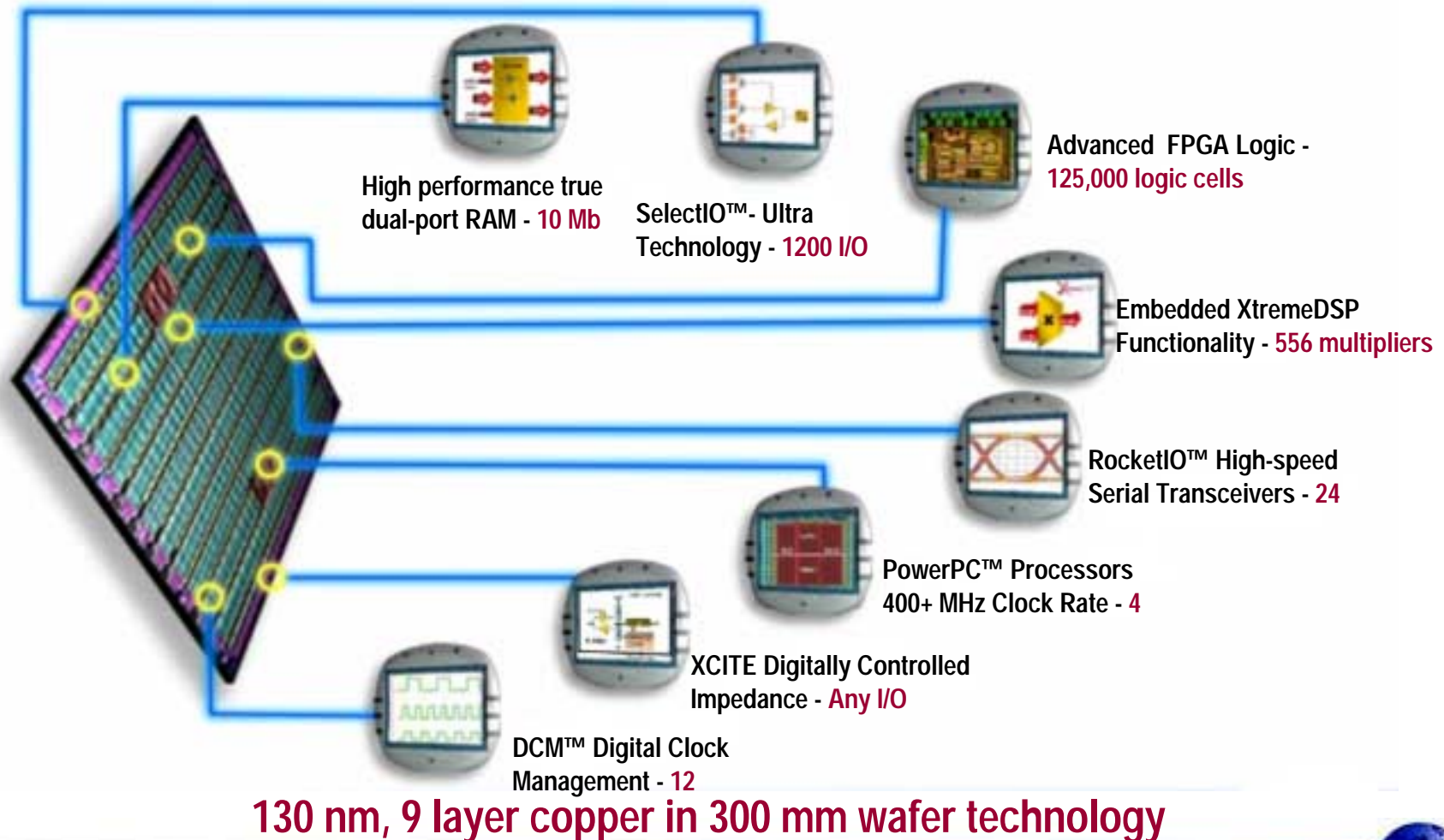


Programmable Logic Evolution

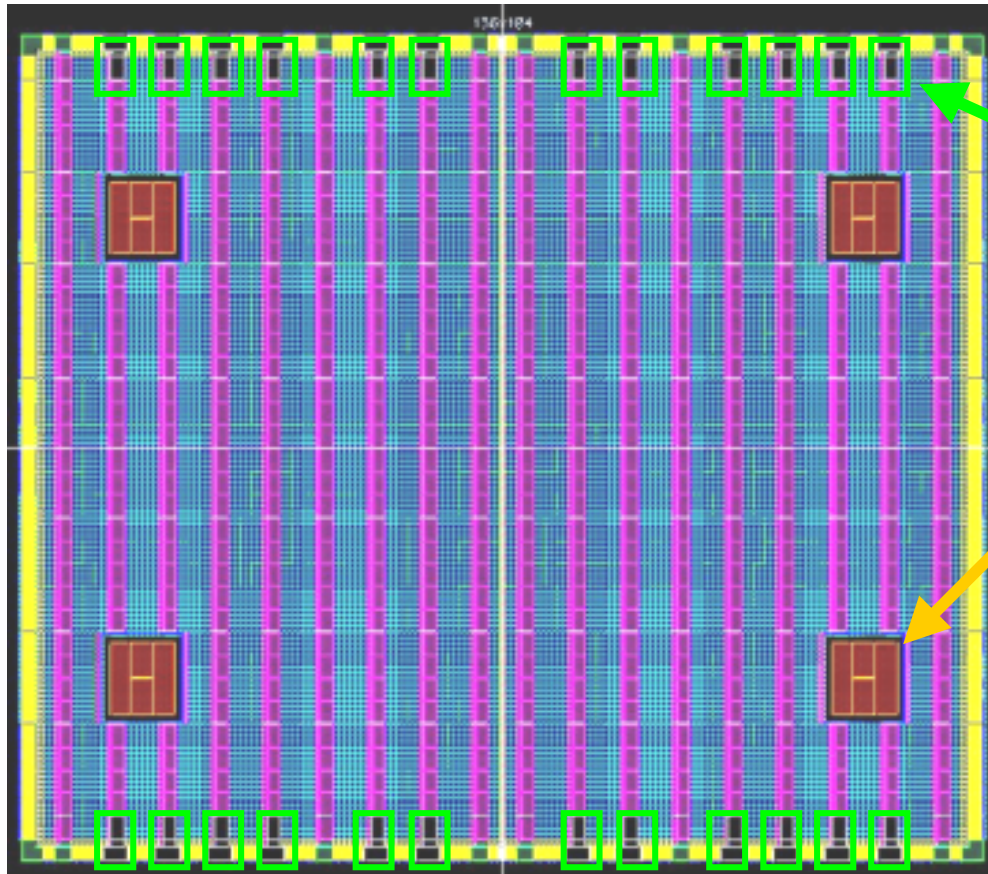


Xilinx Virtex-II Pro FPGA

Setting the Standard in Programmable Logic



Virtex-II Pro Revolution



RocketIO™

- Up to 24 Serial Transceivers
 - 622 Mbps to 3.125 Gbps

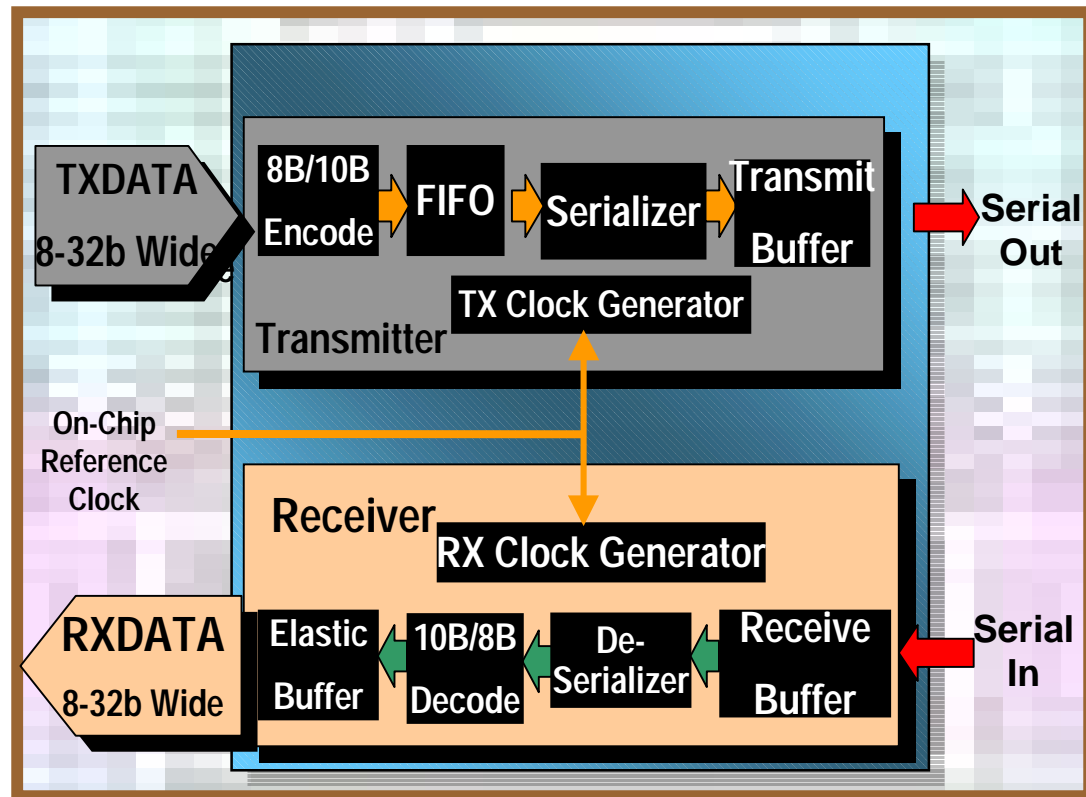
PowerPC™

- Up to 4 PowerPC 405 Processors
 - Industry standard
 - 420 DMIPS at 300 MHz

Built on the Success of Virtex-II Fabric

RocketIO™ SerDes

Leading-Edge High-Speed Serial




- **Multi-Rate**
 - 3.125, 2.5, 2.0, 1.25, 1.0 Gbps
 - 2 - 24 transceivers
- **Multi-Protocol**
 - 1G, 10 G Ethernet (XAUI)
 - PCI Express
 - Serial ATA
 - InfiniBand
 - FibreChannel
 - Serial RapidIO
 - Serial backplanes...
- **Key Features**
 - Embedded 8B/10B Coding
 - 4-Level Pre-Emphasis
 - Programmable Output Swing
 - AC & DC Coupling
 - Channel bonding



Virtex-II Pro Device Family

Covers the Entire Design Spectrum

 EasyPath Logic Cells BRAM (Kbits) DCMs PowerPC™ Processors RocketIO™ Multi-Gigabit Transceivers Xilinx® 3250 Multiplier Blocks		2VP2	2VP4	2VP7	2VP20	2VP30	2VP40	2VP50	2VP70	2VP100	2VP125
						XCE2VP30	XCE2VP40	XCE2VP50	XCE2VP70	XCE2VP100	XCE2VP125
		3,168	6,768	11,088	20,880	30,816	43,632	53,136	74,448	99,216	125,136
		216	504	792	1,584	2,448	3,456	4,176	5,904	7,992	10,008
		4	4	4	8	8	8	8	8	12	12
		0	1	1	2	2	2	2	2	2	4
		4	4	8	8	8	12	16	20	20	24
		12	28	44	88	136	192	232	328	444	556
Package	Maximum User I/O										
FG256	140	140	140								
FG456	248	156	248	248							
FF672	396	204	348	396							
FG676	416				404	416	416				
FF896	556			396	556	556					
FF1152	692				564	644	692	692			
FF1148*	812						804	812			
FF1517	964							852	964		
FF1704	1040								996	1040	1040
FF1696*	1200									1164	1200

* Note Special bond out option, NO MGT with Maximum Select I/O

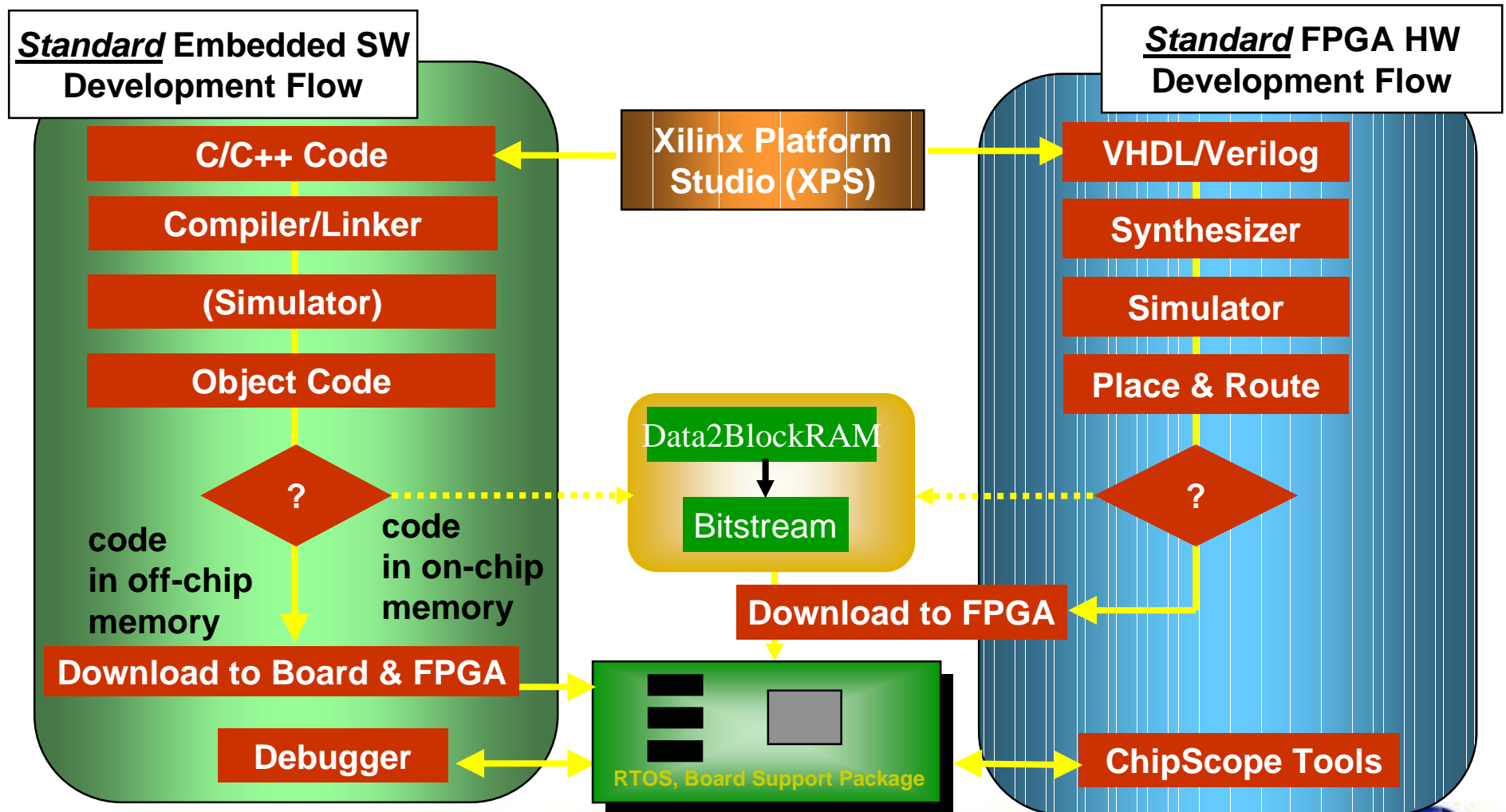
Embedded Development Kit (EDK)

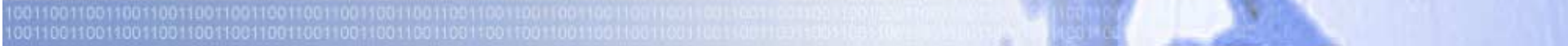
- All encompassing design environment for Virtex-II Pro *PowerPC™* and *MicroBlaze* based embedded systems in Xilinx FPGAs
- Integration of mature FPGA and embedded tools with innovative IP generation and customization tools
- Delivery vehicle for Processor IP



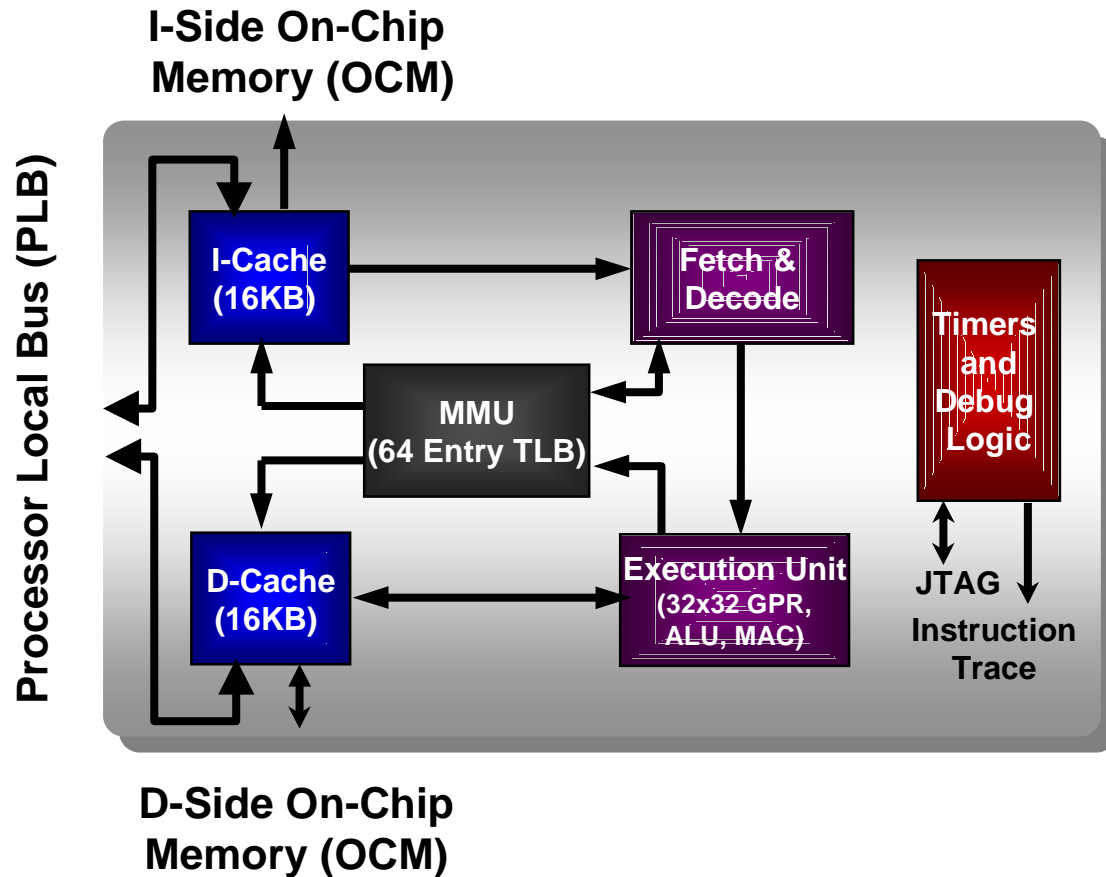
EDK System Design

Comprehensive Tool Chain



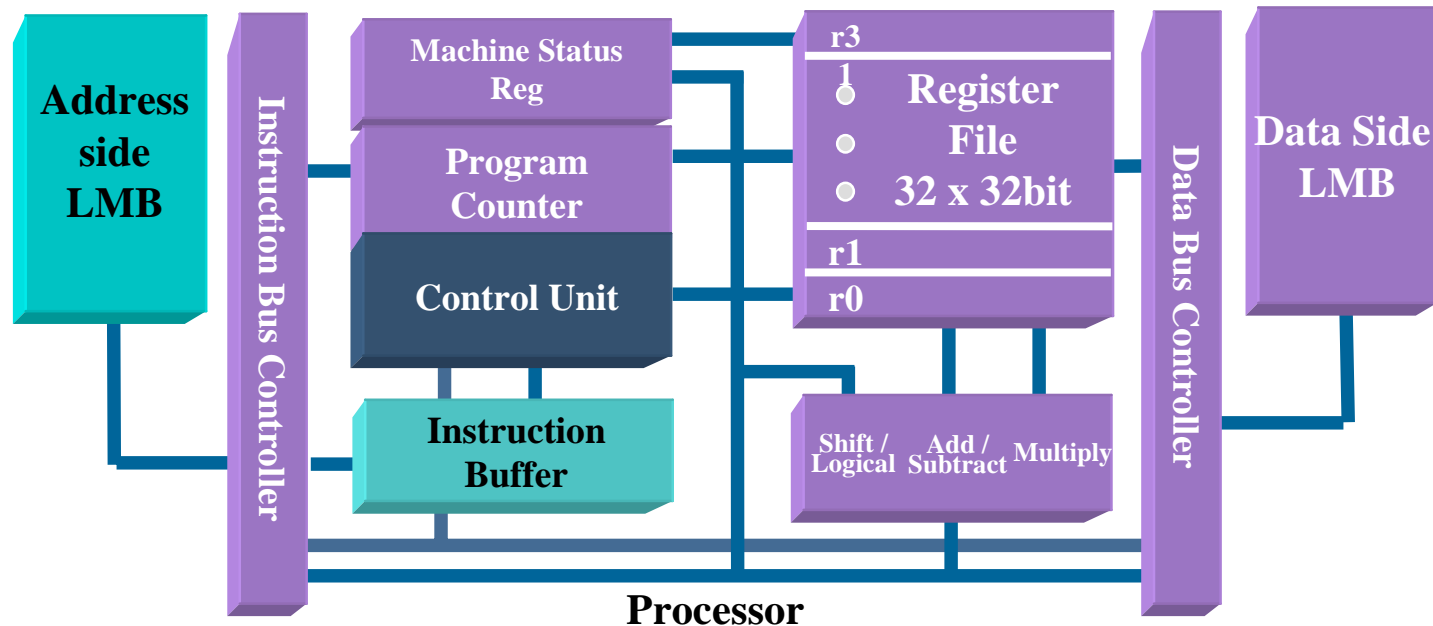


PowerPC 405



- 5-stage data path pipeline
- 16KB D and I Caches
- Embedded Memory Management Unit
- Execution Unit
 - Multiply / divide unit
 - 32 x 32-bit GPR
- Dedicated on-chip memory interfaces
- Timers: PIT, FIT, Watchdog
- Debug and trace support
- Performance:
 - 450 DMIPS at 300 MHz
 - 0.9mW/MHz Typical Power

What is MicroBlaze?



- It's a soft processor, around 900 LUTs
- RISC Architecture
- 32-bit, 32 x 32 general purpose registers
- Supported in Virtex/E/II/IIPro, Spartan-III/III/E

More on MicroBlaze ...

- Harvard Architecture
- Configurable instruction cache, data cache
- Non-intrusive JTAG debug
- Support for 2 buses:
 - LMB (Local Memory Bus) - 1 clock cycle latency, connects to BRAM
 - OPB (On-chip Peripheral Bus) - part of the IBM CoreConnect™ standard, connects to other peripheral “Portable” IP between PPC and MB
- Big endian, same as PowerPC PPC405



MicroBlaze Interrupts and Exceptions

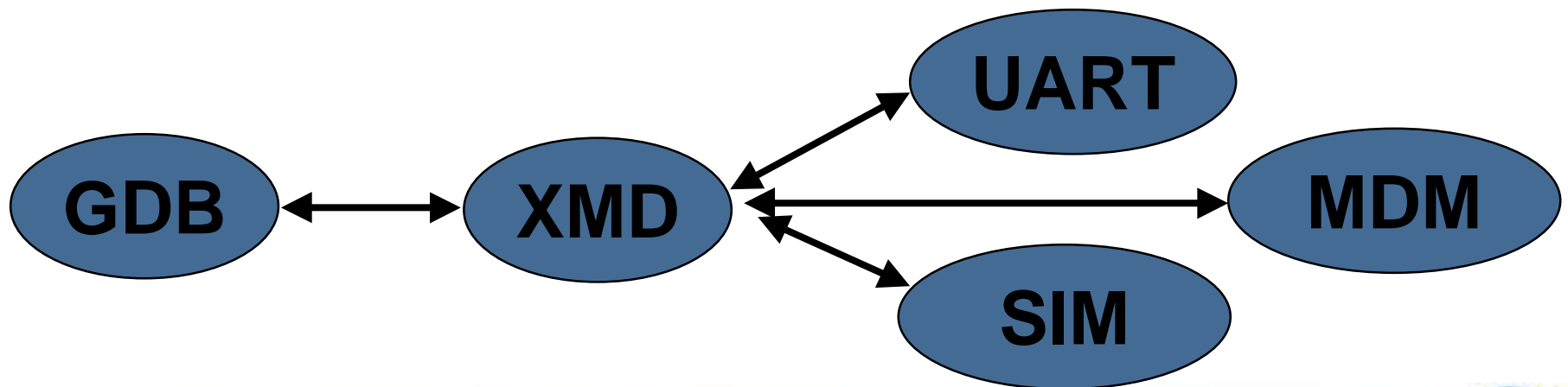
- Interrupt handling
 - 1 Interrupt port
 - 32+ interrupts and masking supported through interrupt controller(s)
- Exception handling
 - No exceptions generated in Virtex-II versions
 - One in Virtex/E and Spartan-II versions for MUL instruction

Software Tools

- GNU tool chain
- GCC - GNU Compiler Collection
- GDB - The GNU debugger
 - Source code debugging
 - Debug either C or assembly code
- XMD - Xilinx Microprocessor Debug utility
 - Separate Process
 - Provides cycle accurate program execution data
 - Supported targets: simulator, hardware board

Software - XMD

- Interfaces GDB to a "target"
- Allows hardware debug without a ROM monitor or reduces debug logic by using xmd-stub (ROM monitor)
- Offers a variety of simulation targets
 - Cycle accurate simulator
 - Real hardware board interface via UART or MDM
- Includes remote debugging capability



MicroBlaze Debug Module

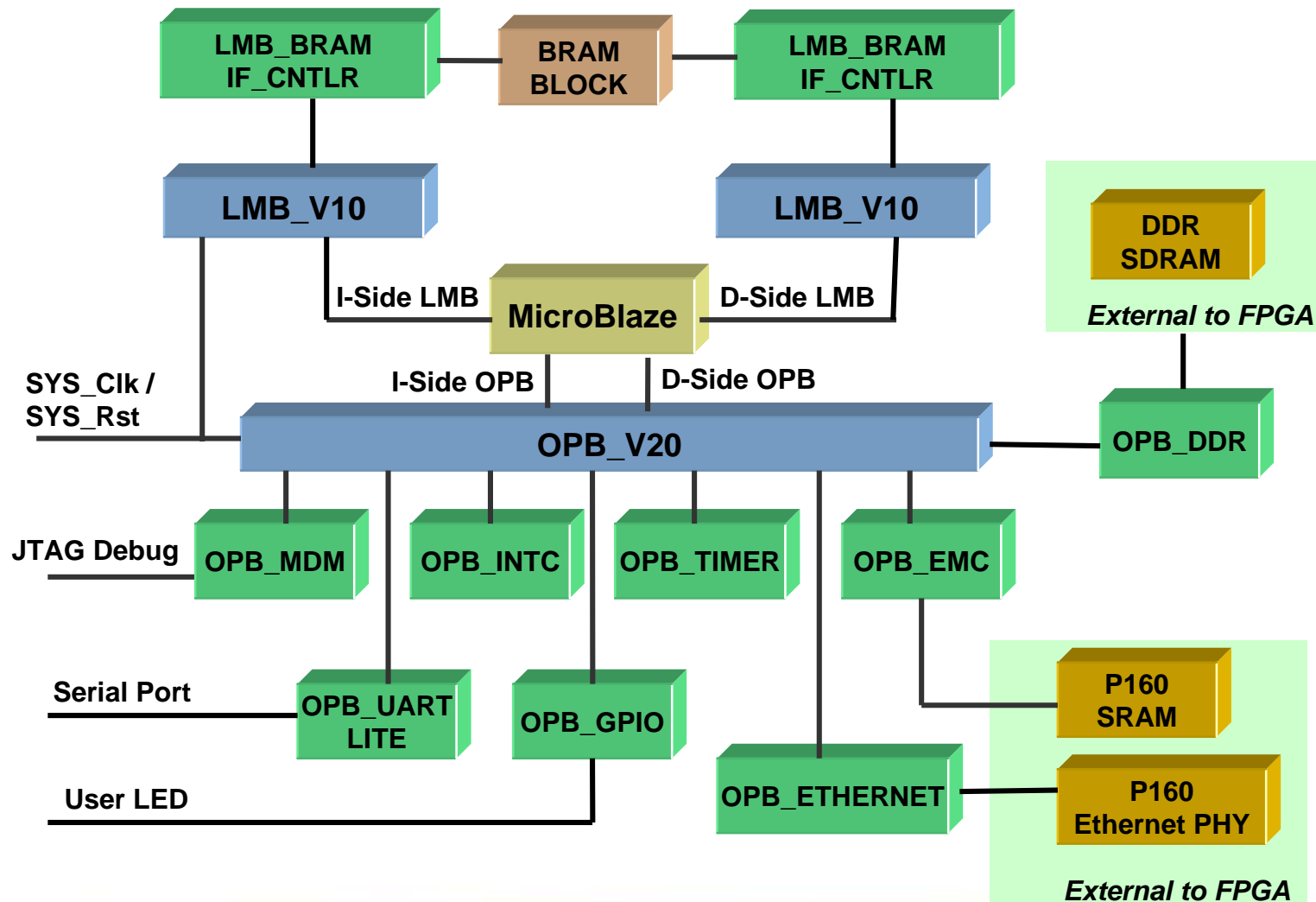
- JTAG debug using BSCAN
- Supports multiple MicroBlazes
- Software non-intrusive debugging
- Read/Write access to internal registers
- Access to all addressable memory
- Hardware single-stepping
- Hardware breakpoints - configurable (max 16)
- Hardware read/write address/data watchpoints
 - configurable (max 8)



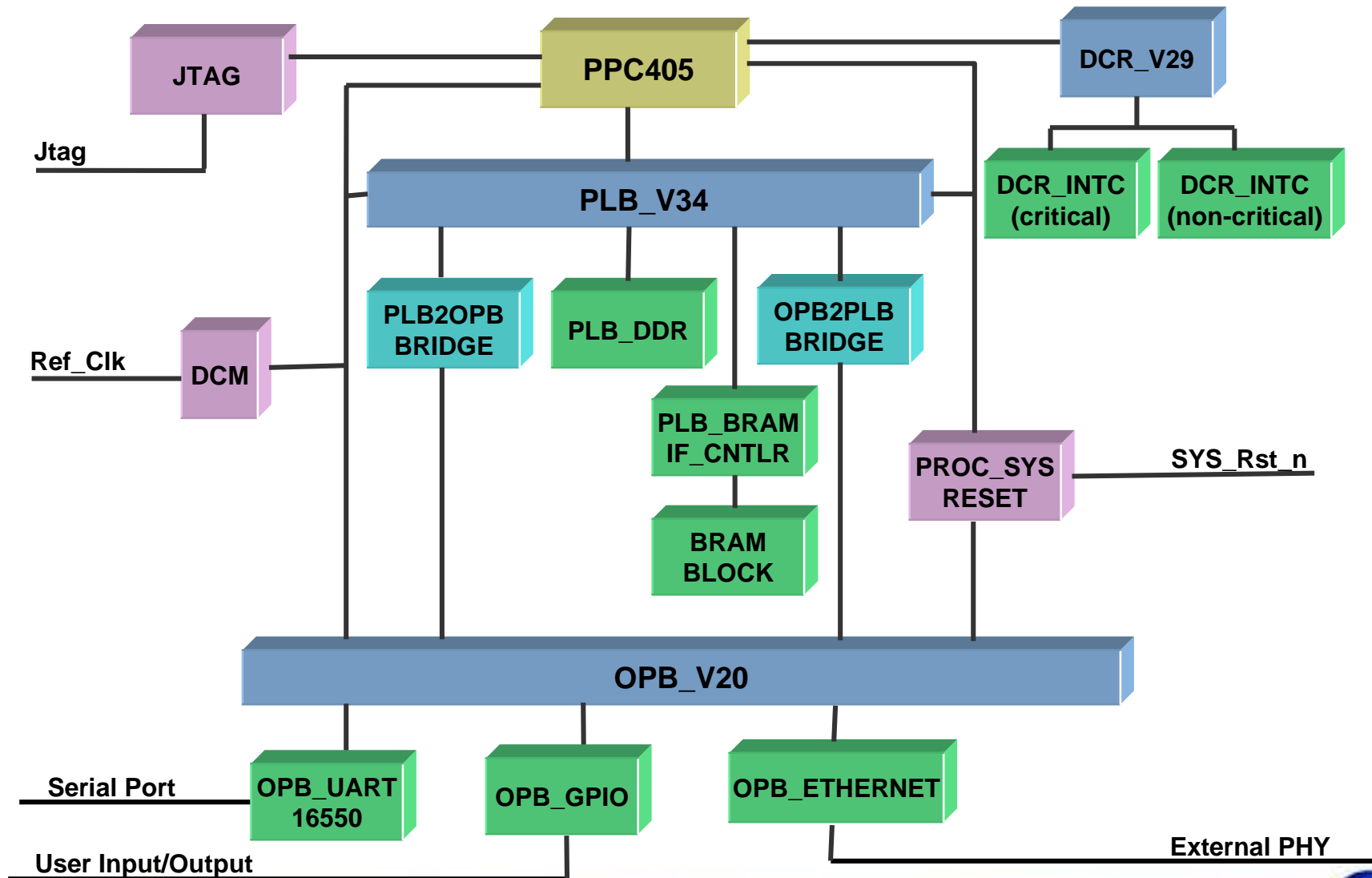
Processor IP

Hardware and Software

Example MicroBlaze System



Example PPC405 System



Processor IP (HW/SW)

Infrastructure (includes Device Drivers)

- MicroBlaze CPU
- LMB2OPB Bridge
- PLB Arbiter & Bus Structure (PLB_V34)
- OPB Arbiter & Bus Structure (OPB_V20)
- DCR Bus Structure (DCR_V29)
- PLB2OPB Bridge
- OPB2PLB Bridge
- OPB2OPB Bridge - Lite
- OPB2DCR Bridge
- System Reset Module
- BSP Generator (SW only)
- ML3 VxWorks BSP (SW only)
- Memory Test Utility (SW only)

OPB IPIF Modules (includes Device Drivers)

- PLB IPIF
 - OPB IPIF-Slave Attachment
 - OPB IPIF-Master Attachment
 - IPIF-Address Decode
 - IPIF-Interrupt Control
 - IPIF-Read Packet FIFOs
 - IPIF-Write Packet FIFOs
 - IPIF-DMA
 - IPIF-Scatter Gather
 - IPIF-FIFOLink
- PLB IPIF
 - PLB IPIF-Slave Attachment
 - PLB IPIF-Master Attachment
 - IPIF-Address Decode
 - IPIF-Interrupt Control
 - IPIF-Read Packet FIFOs
 - IPIF-Write Packet FIFOs
 - IPIF-DMA
 - IPIF-Scatter Gather
 - IPIF-FIFOLink

Processor IP (HW/SW)

Memory Interfaces (includes Device Drivers & Memory Tests)

- PLB EMC (Flash, SRAM, and ZBT)
- PLB BRAM Controller
- PLB DDR Controller
- PLB SDRAM Controller
- OPB EMC (Flash, SRAM, and ZBT)
- OPB BRAM Controller
- OPB DDR Controller
- OPB SDRAM Controller
- OPB SystemACE
- LMB BRAM Controller

Peripherals (includes Device Drivers & RTOS Adapt. Layers)

- OPB Single Channel HDLC Controller
- OPB<->PCI Full Bridge
- OPB 10/100M Ethernet
- OPB 10/100M Ethernet - Lite
- OPB ATM Utopia Level 2 Slave
- OPB ATM Utopia Level 2 Master

Peripherals (continued)

- OPB IIC Master & Slave
- OPB SPI Master & Slave
- OPB UART-16550
- OPB UART-16450
- OPB UART - Lite
- OPB JTAG UART
- OPB Interrupt Controller
- OPB TimeBase/Watch Dog Timer
- OPB Timer/Counter
- OPB GPIO
- PLB 1G Ethernet
- PLB RapidIO
- PLB UART-16550
- PLB UART-16450
- PLB ATM Utopia Level 2 Slave
- PLB ATM Utopia Level 2 Master
- PLB ATM Utopia Level 3 Slave
- PLB ATM Utopia Level 3 Master
- DCR Interrupt Controller

System Infrastructure

- Hardware IP
 - Common PowerPC and MicroBlaze peripherals
 - Peripherals are common across bus types
 - Parameterize for optimal functionality, optimal FPGA usage
 - IP Interface (IPIF) provides common hardware blocks
- Software IP (Device Drivers)
 - Common across processors and operating systems

The Benefits of Parameterization

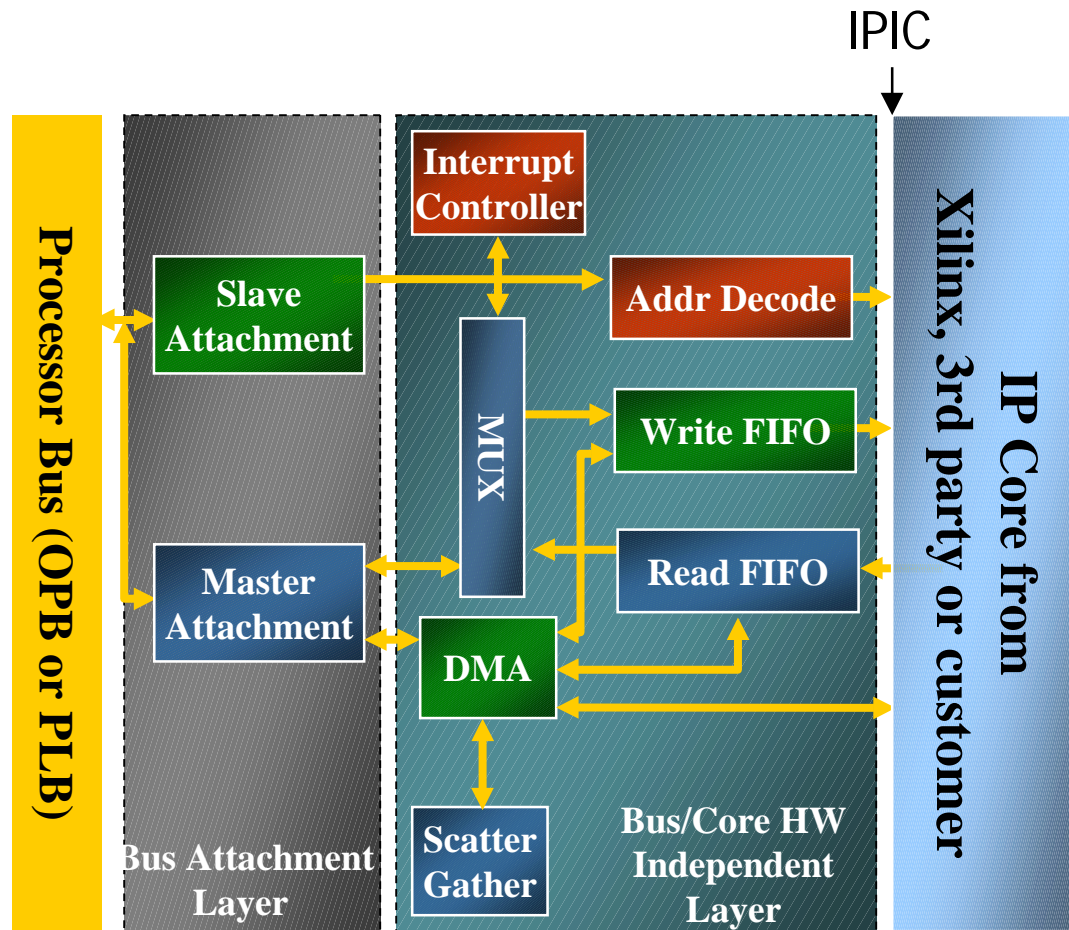
Example: OPB Arbiter

<i>Parameter Values</i>					<i>Resources</i>	<i>F_{MAX}</i>
NUM_MASTERS	PROC_INTERFACE	DYNAM_PRIORITY	PARK	REG_GRANTS	LUTs	MHz
1	N	N	N	N	11	295
2	N	N	N	N	18	223
4	N	N	N	N	34	193
4	Y	N	N	N	59	156
4	N	Y	N	N	54	169
4	N	N	Y	N	83	159
4	N	N	N	Y	34	201
4	Y	Y	Y	Y	146	145
8	Y	Y	Y	Y	388	112

Difference:
>4x in size
>30% in speed

- Significantly increases performance or saves area
- Only include what you need
- This can only be accomplished in a programmable system

Full IP Interface (IPIF)



- Consists of 8 modules
 - Each module is selectable and parameterizable
- Automatically configures a core to the processor bus
 - Xilinx IP Cores
 - 3rd Party IP Cores
 - Customer proprietary cores and external devices
- OPB & PLB supported
 - Bus independent IP Cores and associated Device Drivers
- IPIF will be added to other LogiCOREs

Buses and Arbiters

- PLB
 - Arbitration for up to 16 masters
 - 64-bit and 32-bit masters and slaves
 - IBM PLB compliant
- OPB
 - Includes arbiter with dynamic or fixed priorities and bus parking
 - Parameterized I/O for any number of masters or slaves
 - IBM OPB compliant
- DCR
 - Supports one master and multiple slaves
 - Daisy chain connections for the DCR data bus
 - Required OR function of the DCR slaves' acknowledge signal
- LMB
 - MicroBlaze single-master Local Memory Bus



Bridges

- PLB to OPB
 - Decode up to 4 different address ranges
 - 32-bit or 64-bit PLB slave, 32-bit OPB master
 - Burst and non-burst transfers, cache-line transactions
- OPB to PLB
 - 64-bit PLB master, 32-bit OPB slave
 - Burst and non-burst transfers, cache-line transactions
 - BESR and BEAR
- OPB (slave) to DCR (master)
 - Memory mapped DCR control
- OPB to OPB
 - Allows further OPB partitioning

More System Cores

- Processor System Reset
 - Asynchronous external reset input is synchronized with clock
 - Selectable active high or active low reset
 - DCM Locked input
 - Sequencing of reset signals coming out of reset:
 - First - bus structures come out of reset
 - Second - Peripheral(s) come out of reset 16 clocks later
 - Third - the CPU(s) come out of reset 16 clocks after the peripherals
- JTAG Controller
 - Wrapper for the JTAGPPC primitive.
 - Enables the PowerPC's debug port to be connected to the FPGA JTAG chain
- IPIF User Core Templates
 - Convenient way to add user core to OPB or PLB

Timer / Counter

- Supports 32-bit OPB v2.0 bus interface
- Two programmable interval timers with interrupt, compare, and capture capabilities
- Programmable counter width
- One Pulse Width Modulation (PWM) output

Watchdog Timer / Timebase

- Supports 32-bit bus interfaces
- Watchdog timer (WDT) with selectable timeout period and interrupt
- Two-phase WDT expiration scheme
- Configurable WDT enable: enable-once or enable-disable
- WDT Reset Status (was the last reset caused by the WDT?)
- One 32-bit free-running timebase counter with rollover interrupt



Interrupt Controller

- Number of interrupt inputs is configurable up to the width of the data bus width
- Interrupt controllers can be easily cascaded to provide additional interrupt inputs
- Master Enable Register for disabling the interrupt request output
- Each input is configurable for edge or level sensitivity
 - rising or falling, active high or active low
- Output interrupt request pin is configurable for edge or level generation

UART 16550 / 16450 / Lite

- Register compatible with industry standard 16550/16450
- 5, 6, 7 or 8 bits per character
- Odd, even or no parity detection and generation
- 1, 1.5 or 2 stop bit detection and generation
- Internal baud rate generator and separate RX clock input
- Modem control functions
- Prioritized transmit, receive, line status & modem control interrupts
- Internal loop back diagnostic functionality
- Independent 16 word transmit and receive FIFOs

IIC

- 2-wire (SDA and SCL) serial interface
- Master/Slave protocol
- Multi-master operation with collision detection and arbitration
- Bus busy detection
- Fast Mode 400 KHz or Standard Mode 100 KHz operation
- 7 Bit, 10 Bit, and General Call addressing
- Transmit and Receive FIFOs - 16 bytes deep
- Bus throttling

SPI

- 4-wire serial interface (MOSI, MISO, SCK, and SS)
- Master or slave modes supported
- Multi-master environment supported (requires tri-state drivers and software arbitration for possible conflict)
- Multi-slave environment supported (requires additional decoding and slave select signals)
- Programmable clock phase and polarity
- Optional transmit and receive FIFOs
- Local loopback capability for testing

Ethernet 10/100 MAC

- 32-bit OPB master and slave interfaces
- Media Independent Interface (MII) for connection to external 10/100 Mbps PHY Transceivers
- Full and half duplex modes of operation
- Supports unicast, multicast, broadcast, and promiscuous addressing
- Provides auto or manual source address, pad, and Frame Check Sequence

Ethernet 10/100 MAC (cont)

- Simple DMA and Scatter/Gather DMA architecture for low processor and bus utilization, as well as a simple memory-mapped direct I/O interface
- Independent 2K to 32K transmit and receive FIFOs
- Supports MII management control writes and reads with MII PHYs
- Supports VLAN and Pause frames
- Internal loopback mode

1 Gigabit MAC

- 64-bit PLB master and slave interfaces
- GMII for connection to external PHY Transceivers
- Optional PCS function with Ten Bit Interface (TBI) to external PHY devices
- Option PCS/PMA functions with SerDes interface to external transceiver devices for reduced signal count
- Full duplex only
- Provides auto or manual source address, pad, and Frame Check Sequence

1 Gigabit MAC (cont)

- Simple DMA and Scatter/Gather DMA architecture for low processor and bus utilization, as well as a simple memory-mapped direct I/O interface
- Independent, depth-configurable TX and RX FIFOs
- Supports MII management control writes and reads with MII PHYs
- Jumbo frame and VLAN frame support
- Internal loopback mode

Single Channel HDLC

- Support for a single full duplex HDLC channel
- Selectable 8/16 bit address receive address detection, receive frame address discard, and broadcast address detection
- Selectable 16 bit (CRC-CCITT) or 32 bit (CRC-32) frame check sequence
- Flag sharing between back to back frames
- Data rates up to OPB_Clk frequency

Single Channel HDLC (cont)

- Simple DMA and Scatter/Gather DMA architecture for low processor and bus utilization, as well as a simple memory-mapped direct I/O interface
- Independent, depth-configurable TX and RX FIFOs
- Selectable broadcast address detection and receive frame address discard
- Independent RX and TX data rates

ATM Utopia Level 2

- UTOPIA Level 2 master or slave interface
- UTOPIA interface data path of 8 or 16 bits
- Single channel VPI/VCI service and checking in received cells
- Header error check (HEC) generation and checking
- Parity generation and checking
- Selectively prepend headers to transmit cells, pass entire received cells or payloads only, and transfer 48 byte ATM payloads only



ATM Utopia Level 2 (cont)

- Simple DMA and Scatter/Gather DMA architecture for low processor and bus utilization, as well as a simple memory-mapped direct I/O interface
- Independent, depth-configurable TX and RX FIFOs
- Interface throughput up to 622 Mbps (OC12)
- Internal loopback mode

OPB-PCI Bridge

- 33/66 MHz, 32-bit PCI buses
- Full bridge functionality
 - OPB Master read/write of a remote PCI target (both single and burst)
 - PCI Initiator read/write of a remote OPB slave (both single and multiple)
- Supports up to 3 PCI devices with unique memory PCI memory space
- Supports up to 6 OPB devices with unique memory OPB memory space
- PCI and OPB clocks can be totally independent

System ACE Controller

- Used in conjunction with System ACE CompactFlash Solution to provide a System ACE memory solution.
- System ACE Microprocessor Interface (MPU)
 - Read/Write from or to a CompactFlash device
 - MPU provides a clock for proper synchronization
- ACE Flash (Xilinx-supplied Flash Cards)
 - Densities of 128 MBits and 256 Mbits
 - CompactFlash Type 1 form factor
 - Supports any standard CompactFlash module, or IBM microdrives up to 8 Gbits, all with the same form factor.
- Handles byte, half-word, and word transfers

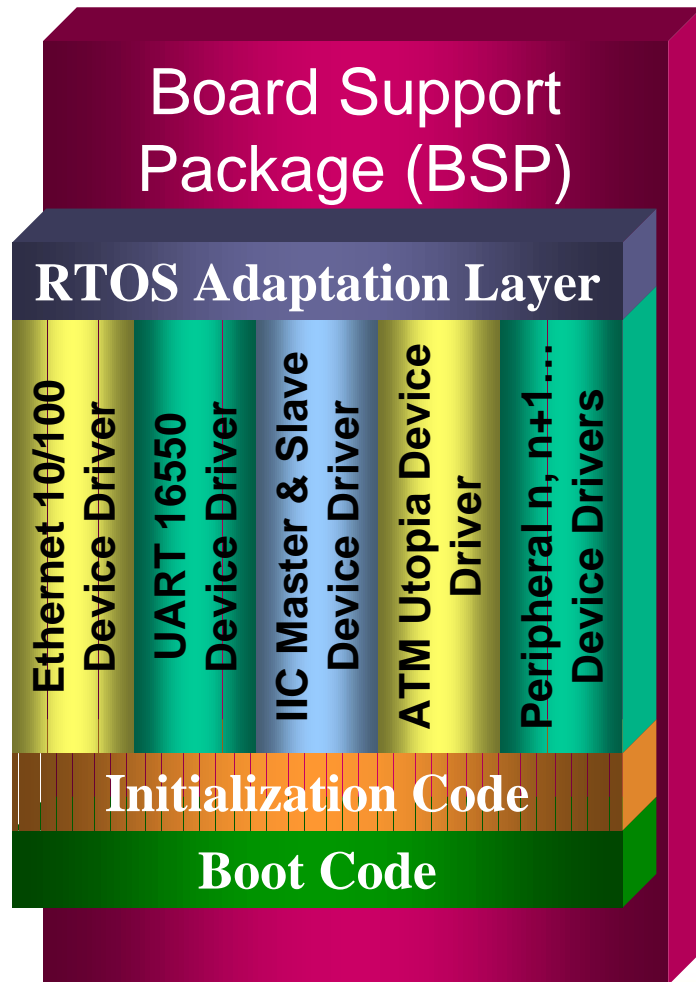
GPIO

- OPB V2.0 bus interface with byte-enable support
- Supports 32-bit bus interface
- Each GPIO bit dynamically programmable as input or output
- Number of GPIO bits configurable up to size of data bus interface
- Can be configured as inputs-only to reduce resource utilization

Memory Controllers

- PLB and OPB interfaces
- External Memory Controller
 - Synchronous Memory (ZBT)
 - Asynchronous Memory (SRAM, Flash)
- Internal Block Memory (BRAM) Controllers
- DDR and SDRAM

Device Drivers and Board Support Packages (BSPs)



- Device drivers are provided for each hardware device
- Device drivers are written in C and are designed to be portable across processors
- Device drivers allow the user to select the desired functionality to minimize the required memory
- BSPs are automatically generated by EDK tools



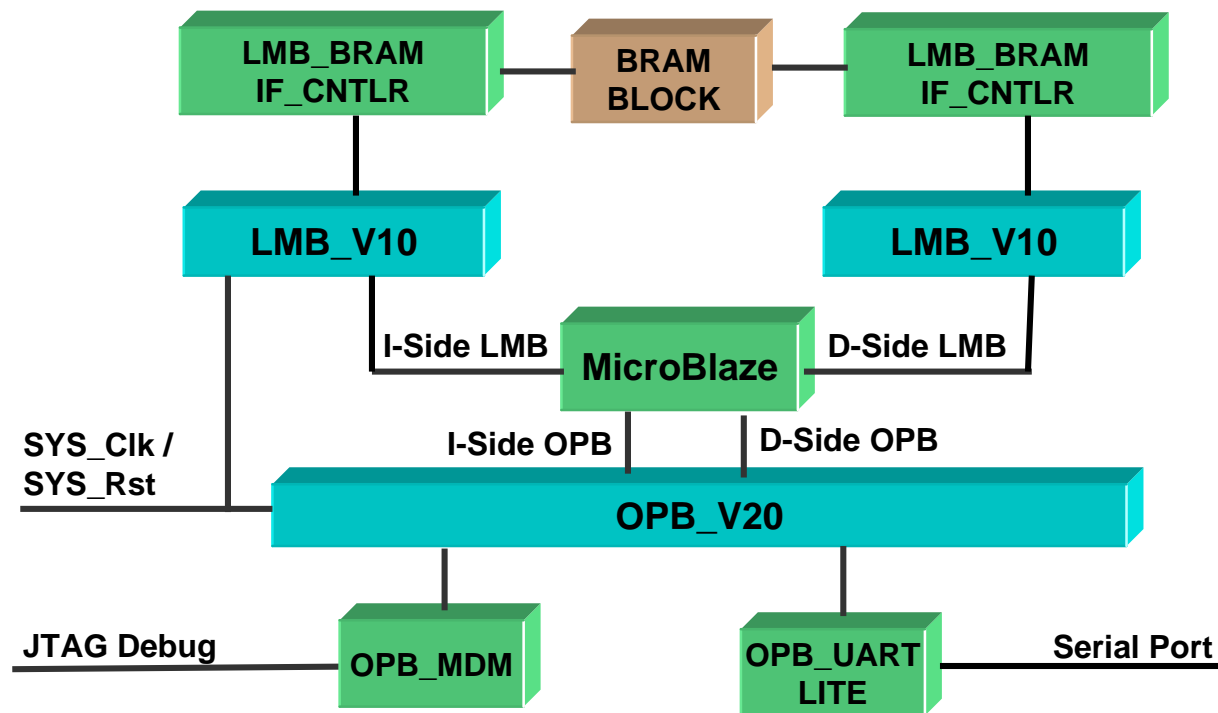
Creating a Simple MicroBlaze System with XPS

with Base System Builder Wizard

Design Flow

- Design Entry with Xilinx Platform Studio using the Base System Builder Wizard
- Generate system netlist with XPS
- Generate hardware bitstream with XPS
- Download and sanity check design with XPS and XMD

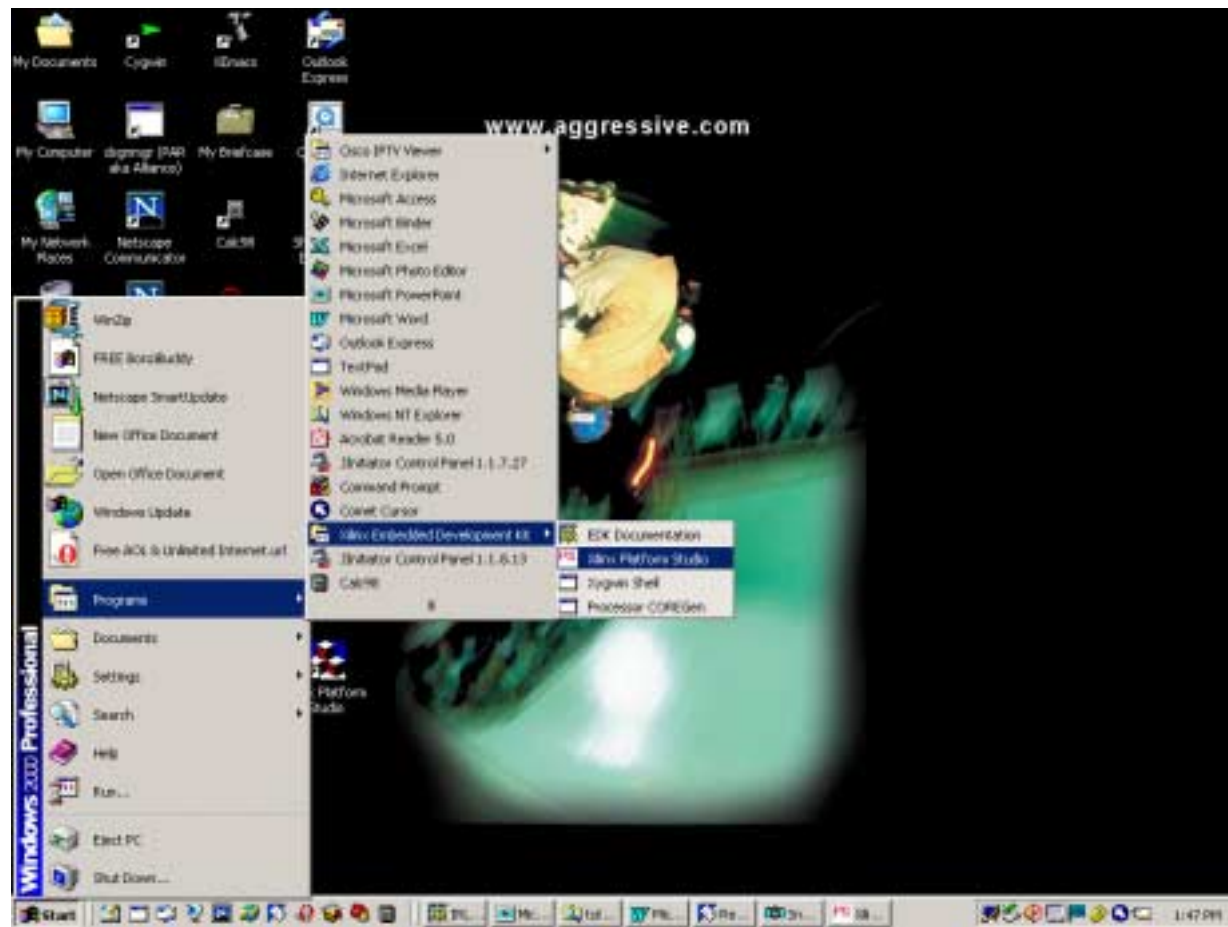
Simple MicroBlaze System Block Diagram



External to FPGA

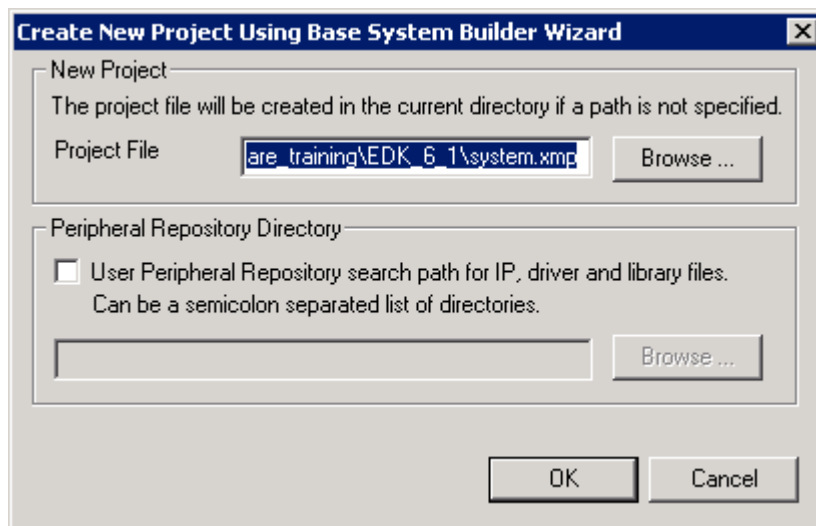


Start Xilinx Platform Studio



Create A New Project

- Select File from the Tools menu
- Select the New Project submenu and the Base System Builder submenu
- Browse to the location where the project is to be located
- Click OK to start creating the project

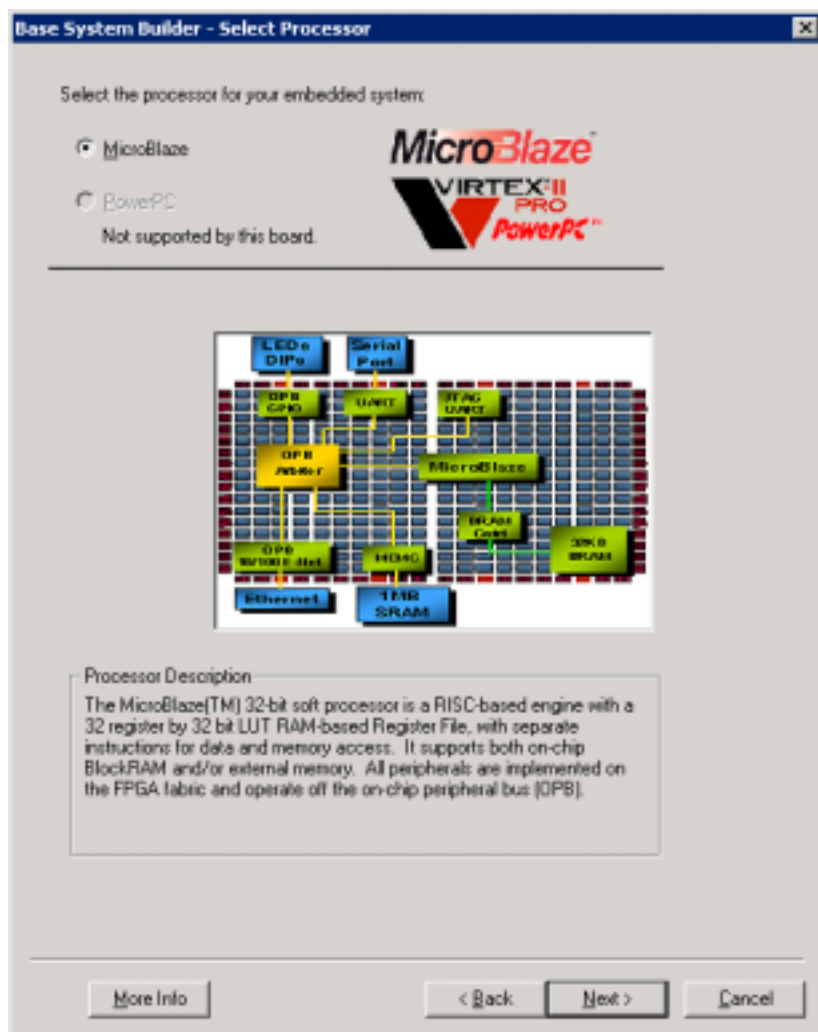


Selecting The Board



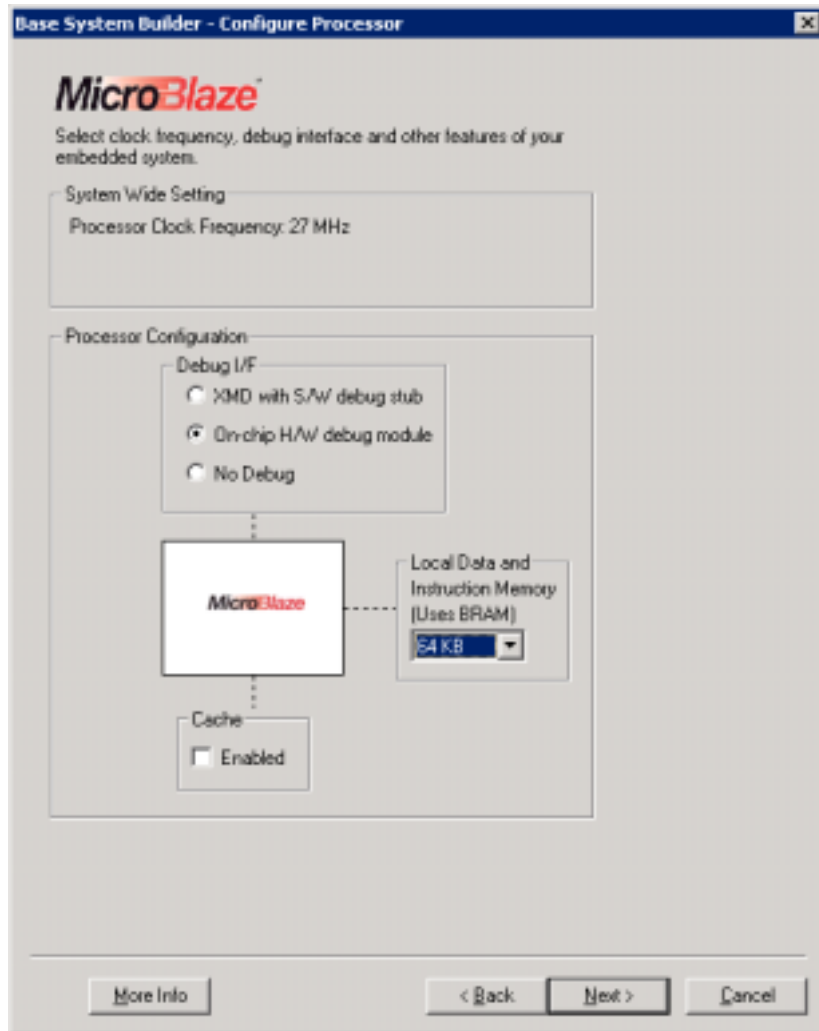
- Select Xilinx as the Board Vendor
- Select Virtex-II Multimedia FF896 Development Board as the Board Name
- Select Board Revision 1
- Click Next to continue to the next step

Select the Processor



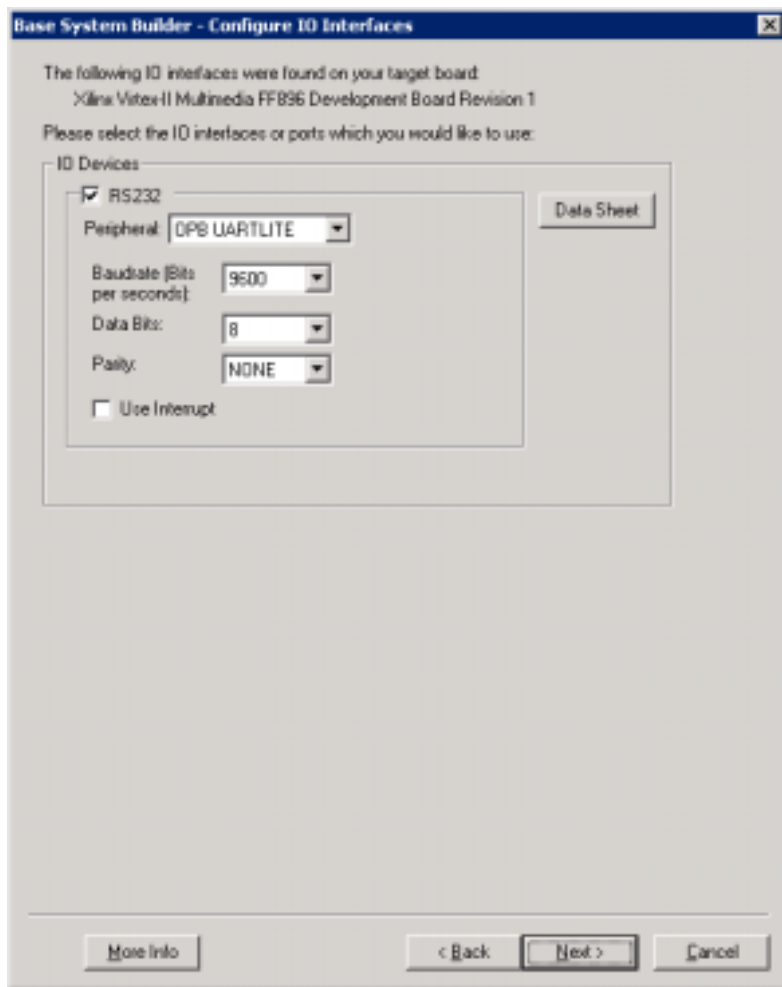
- This board has a Virtex-II FPGA which does not contain a PowerPC processor
- Click Next to continue to the next step

Configuring The Processor



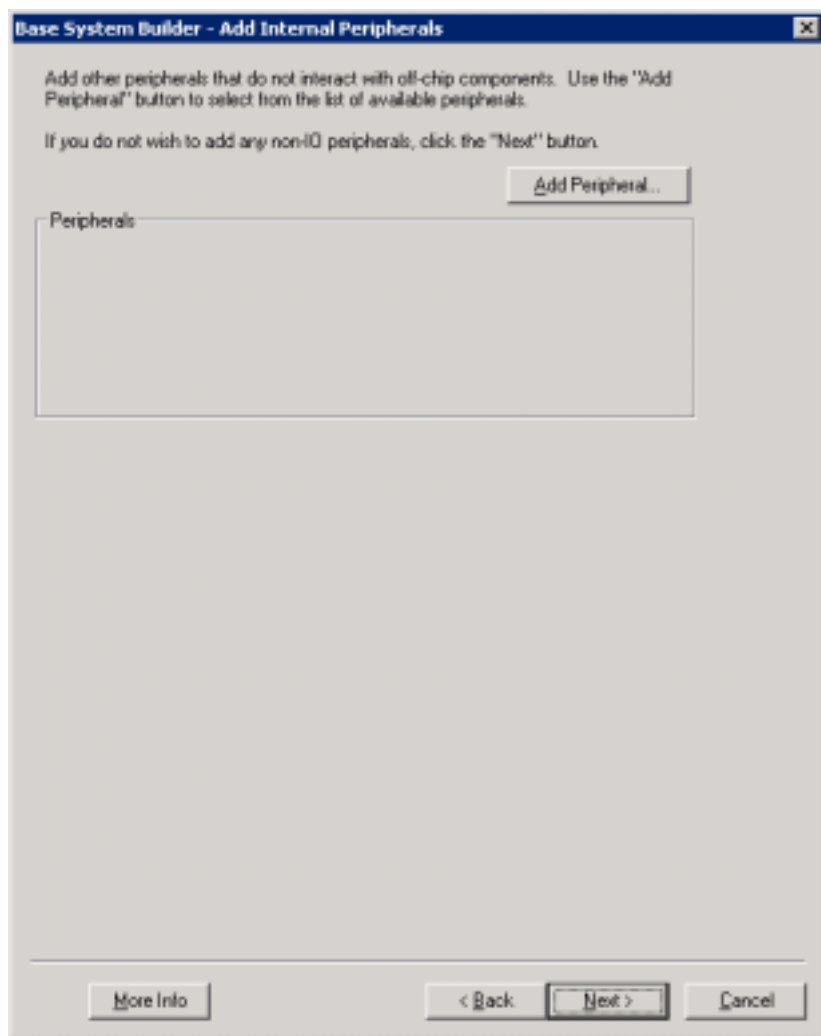
- Select On-chip H/W debug module such that a ROM monitor is not required
- Select 64KB of Local Data and Instruction Memory (BRAM)
- There is no need to select caches when using internal BRAM
- Click Next to continue to the next step

Configuring I/O Interfaces



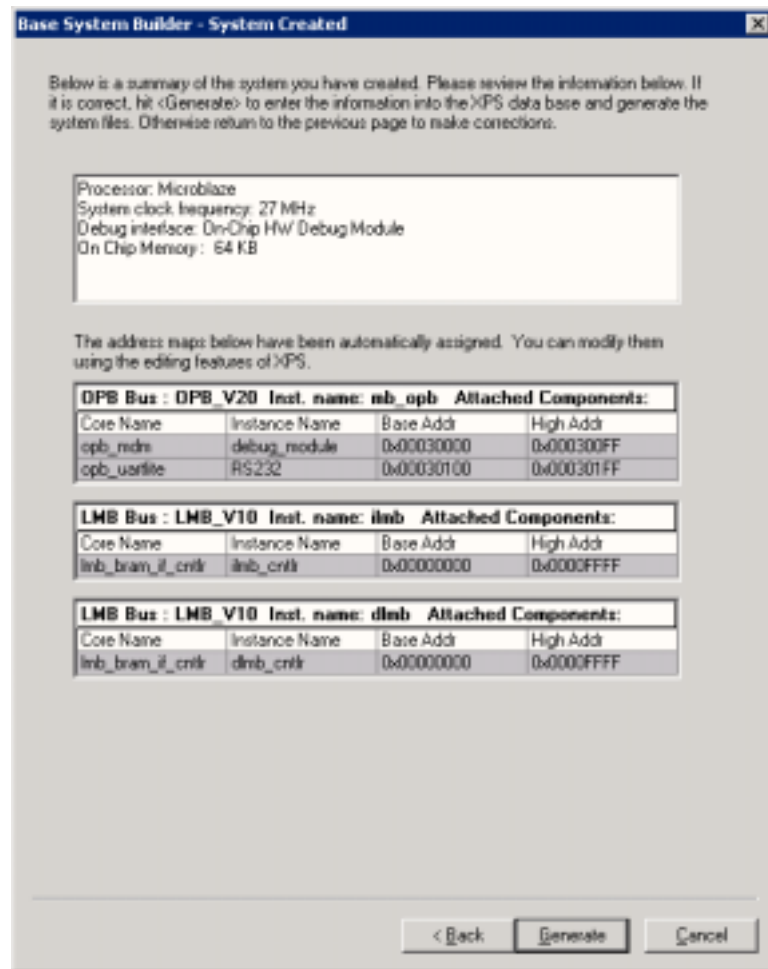
- The board has a serial port and it is default behavior to build the hardware with it
- Since it is used as standard I/O it is not necessary to be interrupt driven
- Click Next to continue to the next step

Adding Internal Peripherals



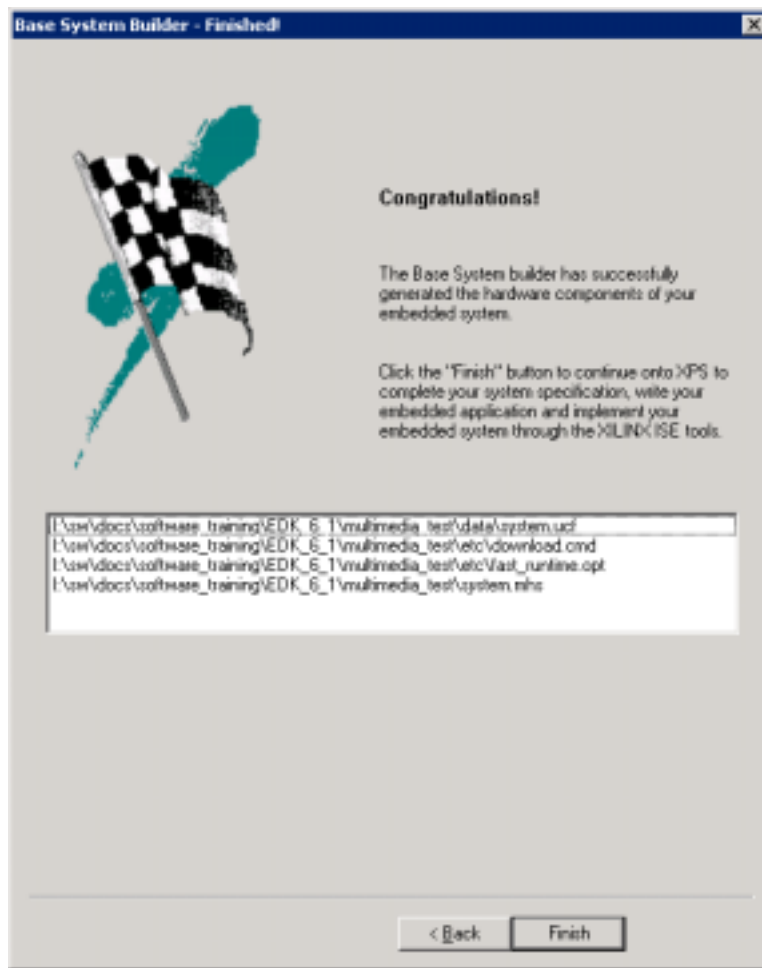
- Other peripherals can be added at this point, such as a timer counter.
- Click Next to continue to the next step

System Summary



- The system has been created and is ready to be generated
- Review the details of the system and backup if necessary to make changes
- Click the Generate button to create the data files for XPS

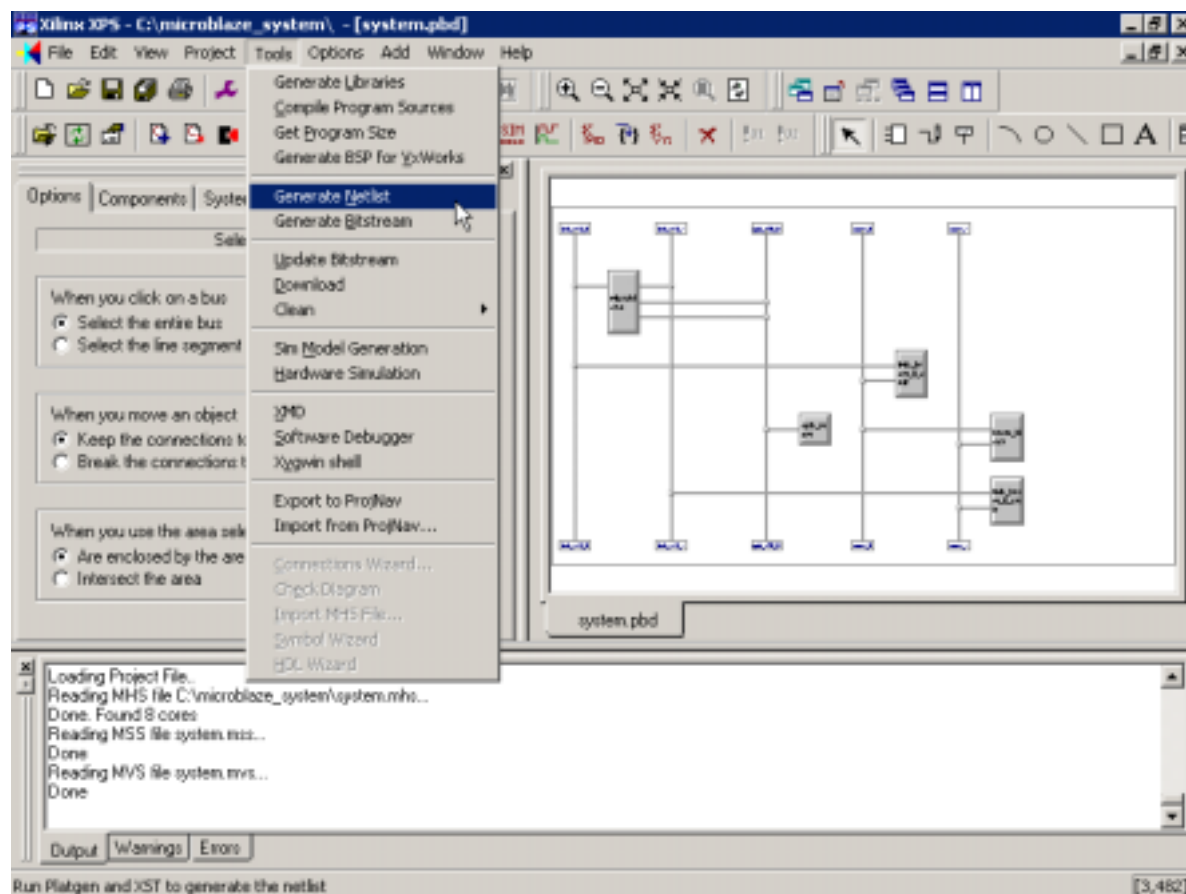
Base System Builder Finished



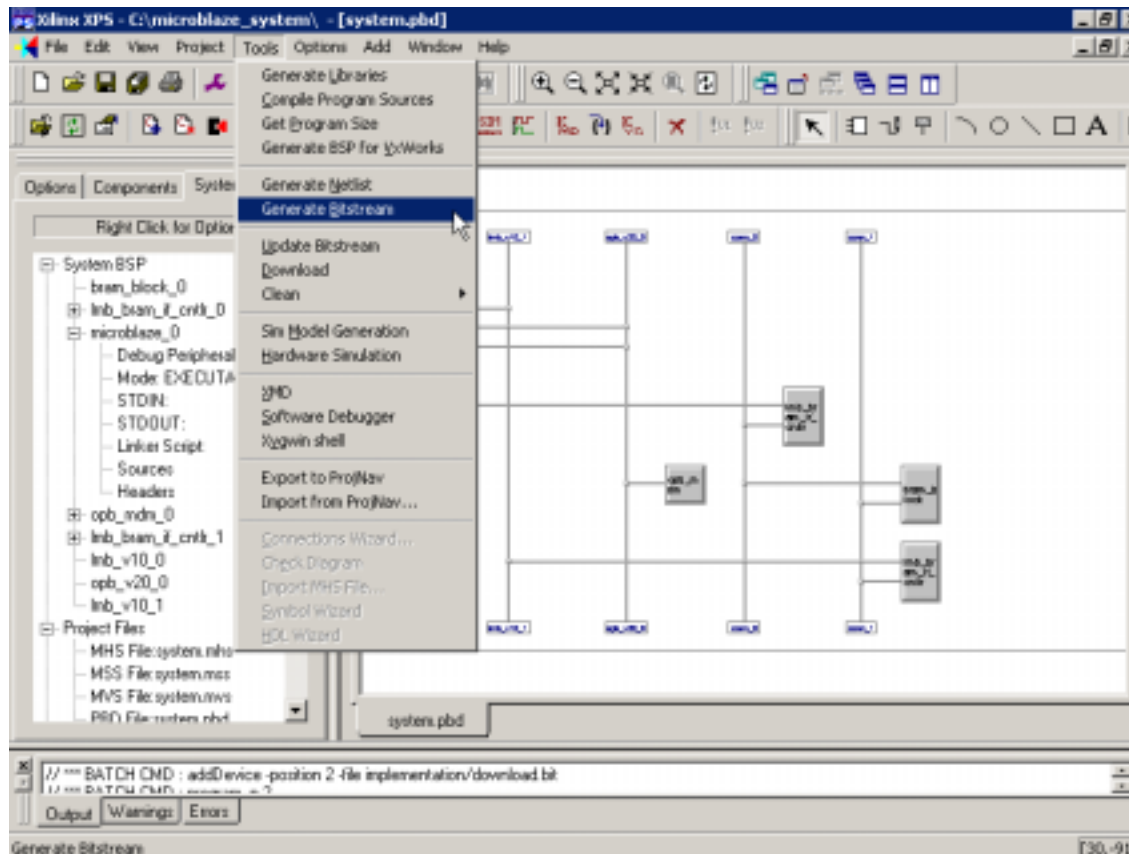
- The Base System Builder Wizard in XPS has completed
- The data files for XPS have been generated such that the system will be contained in an XPS project
- Click the Finish button to exit the wizard & return to XPS

Generating Hardware NetList

- Select the Tools menu
- Select the Generate Netlist submenu
- Wait for the generation to complete

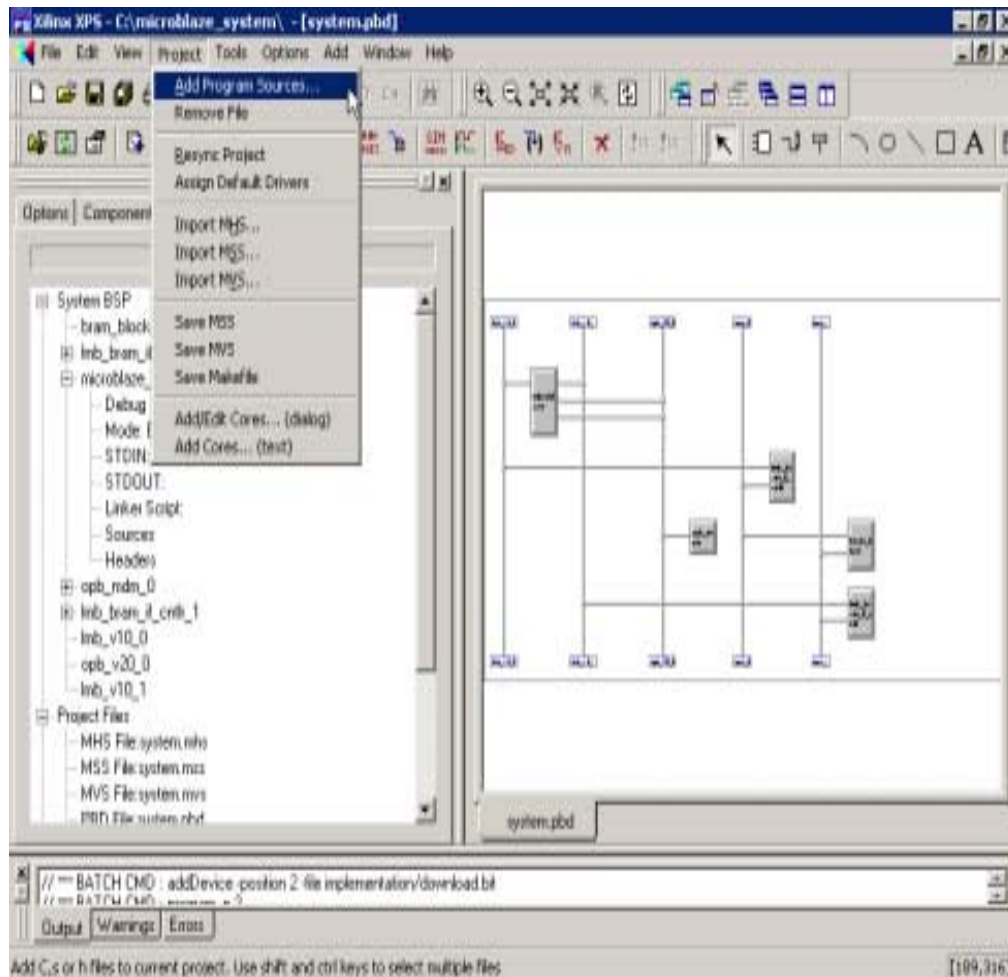


Generating Hardware Bitstream



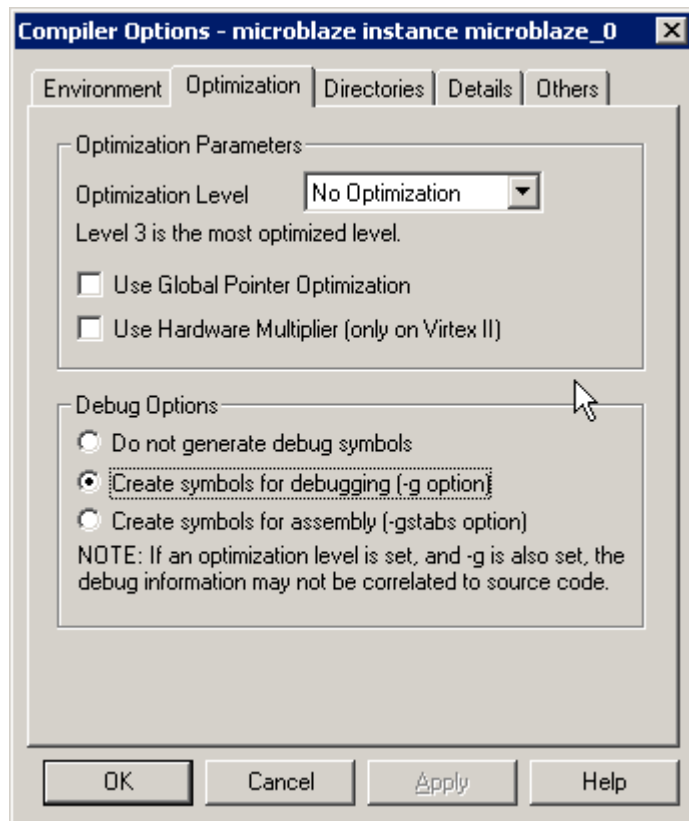
- Select the Tools menu
- Select the Generate Bitstream submenu
- Wait for the bitstream generation to complete

Adding Software Source Files



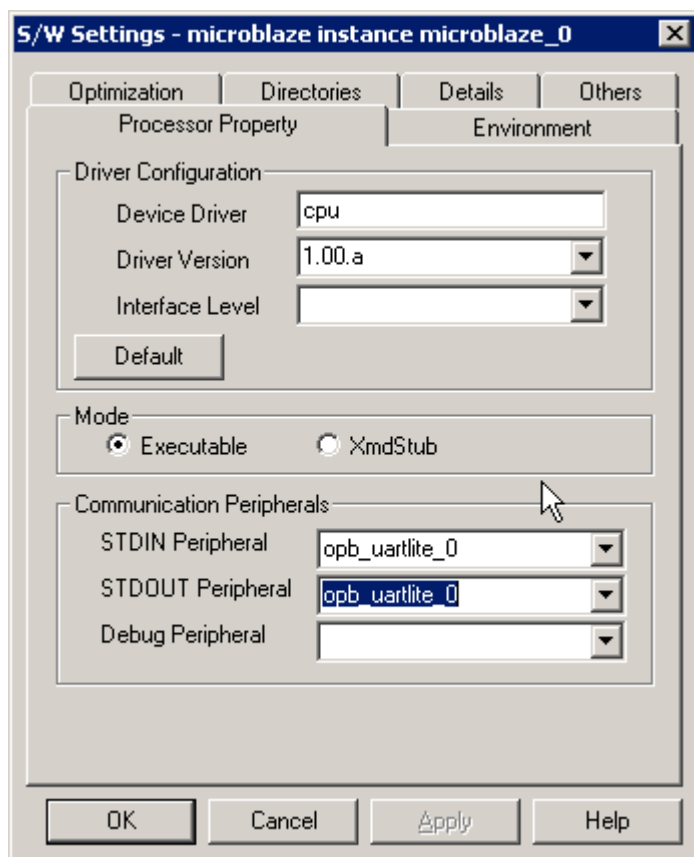
- Select processor in System tab
- Select the Tools menu
- Select the Add Program Sources submenu
- Navigate to the source file and select it.

Setting Compile Options



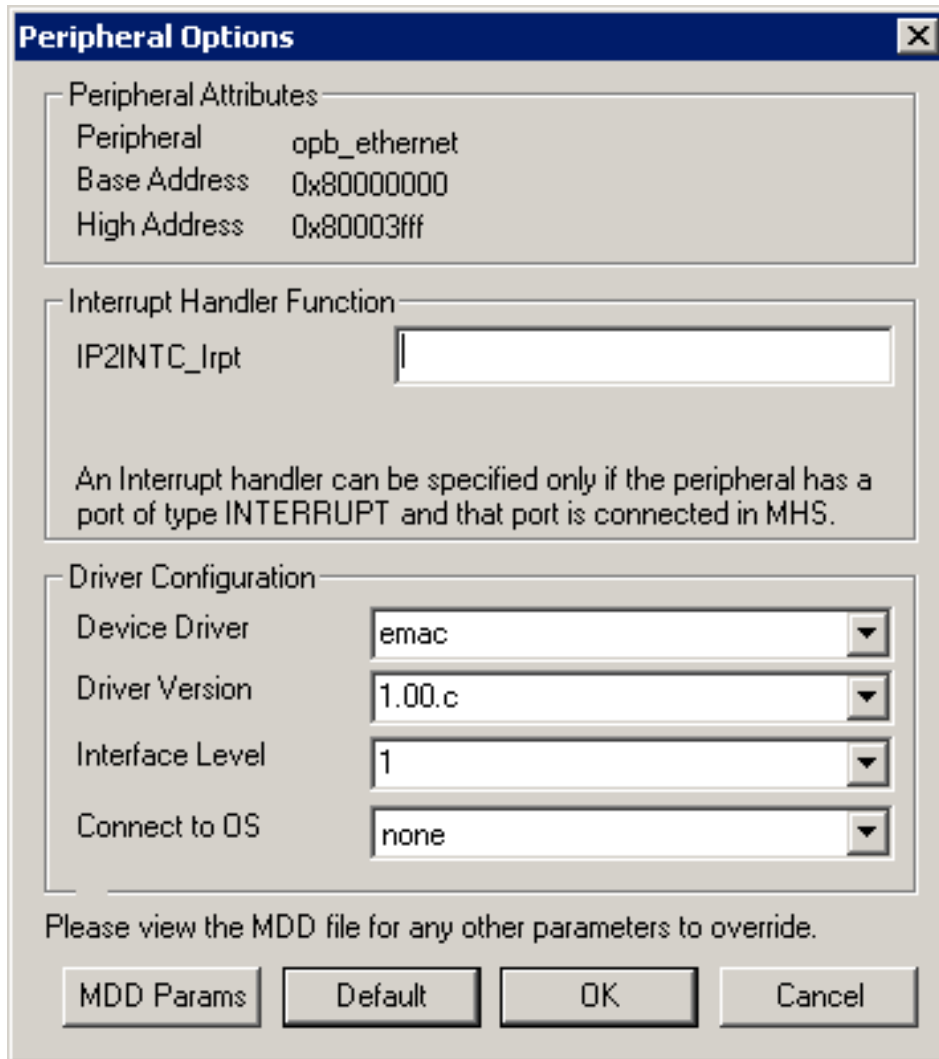
- Select the Options menu
- Select the Compiler Options submenu
- Set the optimization to none
- Set the debug options to create symbols for debugging

Setting Up Standard I/O



- Select the System tab
- Double click on the microblaze_0
- The dialog box illustrated is displayed
- Select the opb_uartlite_0 for the STDIN and STDOUT peripheral and click the OK button

Assigning Drivers To Devices



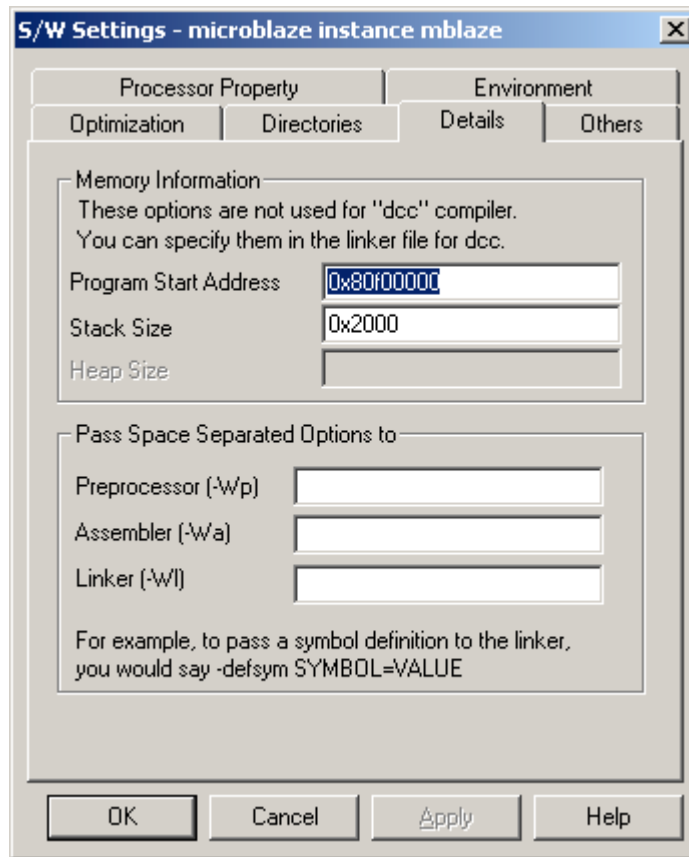
The image shows a 'Peripheral Options' dialog box with the following sections:

- Peripheral Attributes**
 - Peripheral: opb_ethernet
 - Base Address: 0x80000000
 - High Address: 0x80003fff
- Interrupt Handler Function**
 - IP2INTC_Irpt: [Empty text box]
 - Text below: An Interrupt handler can be specified only if the peripheral has a port of type INTERRUPT and that port is connected in MHS.
- Driver Configuration**
 - Device Driver: emac
 - Driver Version: 1.00.c
 - Interface Level: 1
 - Connect to OS: none

At the bottom, there is a note: 'Please view the MDD file for any other parameters to override.' and four buttons: 'MDD Params', 'Default', 'OK', and 'Cancel'.

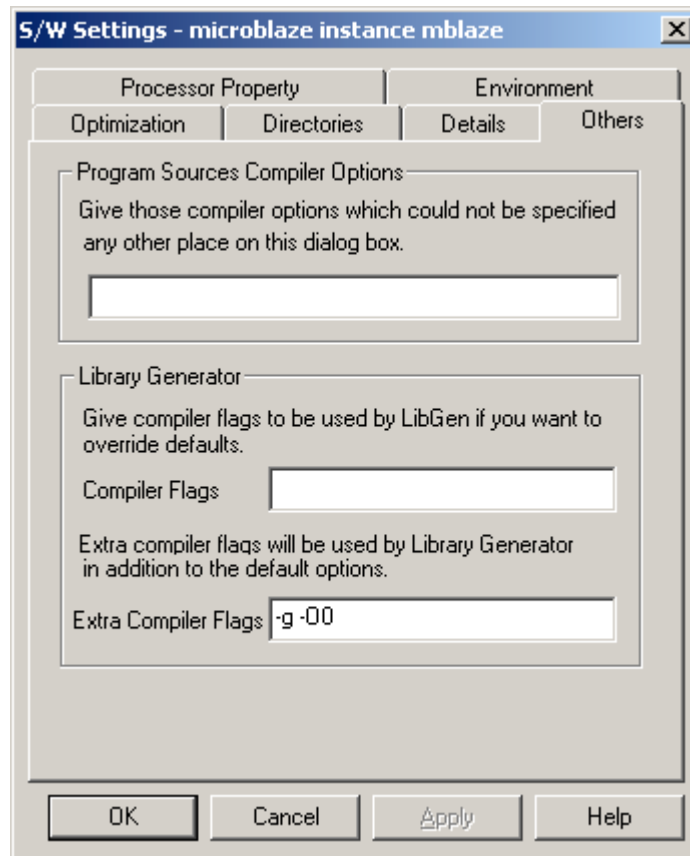
- Assigning a driver to a device causes the driver to be compiled into the library & linked with the application
- Double click on the device in the System tab of the XPS project
- The latest version of the driver is displayed by default
- Choose the appropriate Interface Level of the driver

Setting Memory Options



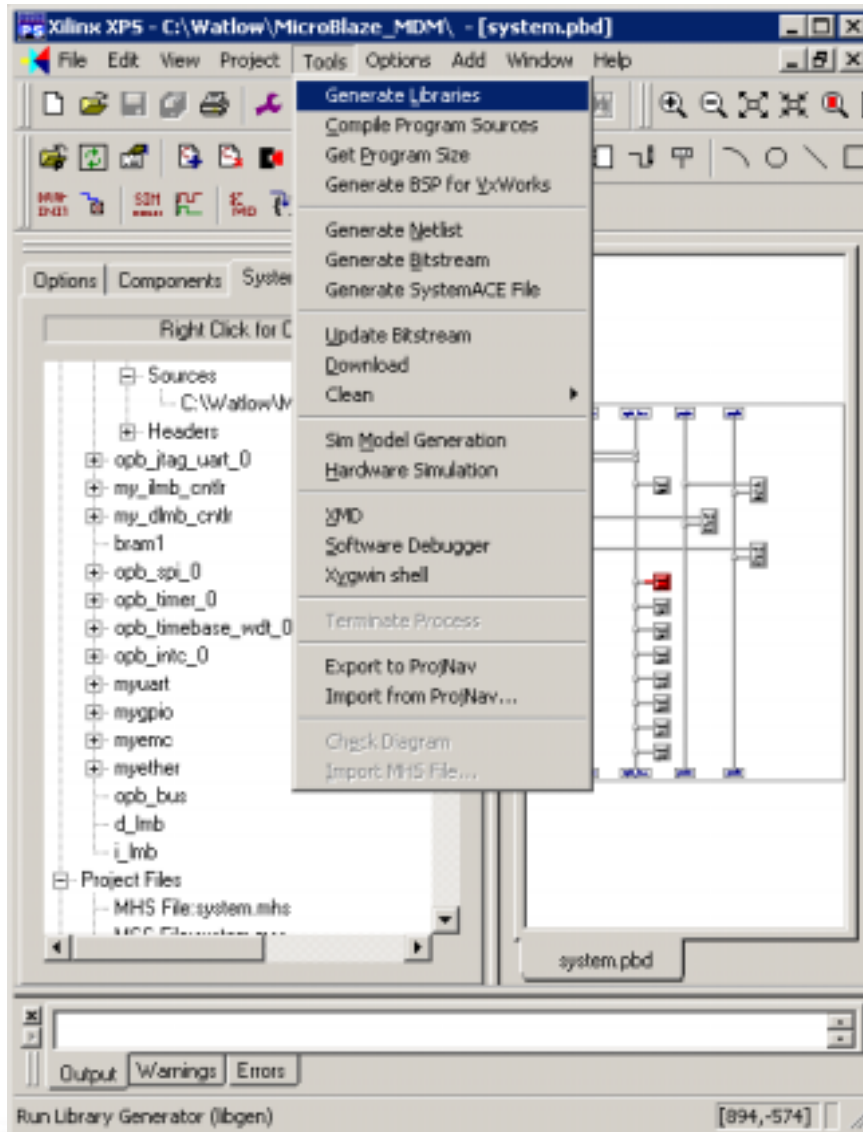
- Double click on the processor instance in the System tab
- Select the Details tab
- Change the program start address or stack size
- Add other options to the preprocessor, assembler, or linker

Setting Library Options



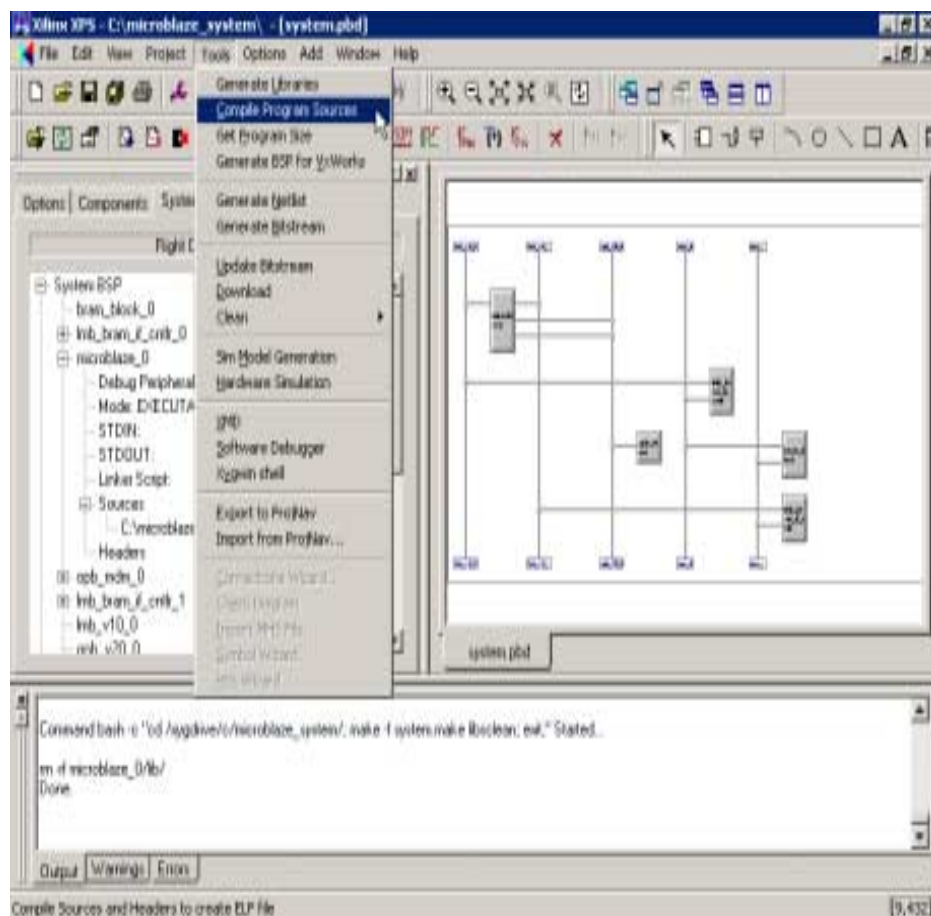
- Double click on the processor instance in the System tab
- Select the Others tab
- Add or change options for library compilation
- Add other options for application compilation

Generating Libraries In XPS



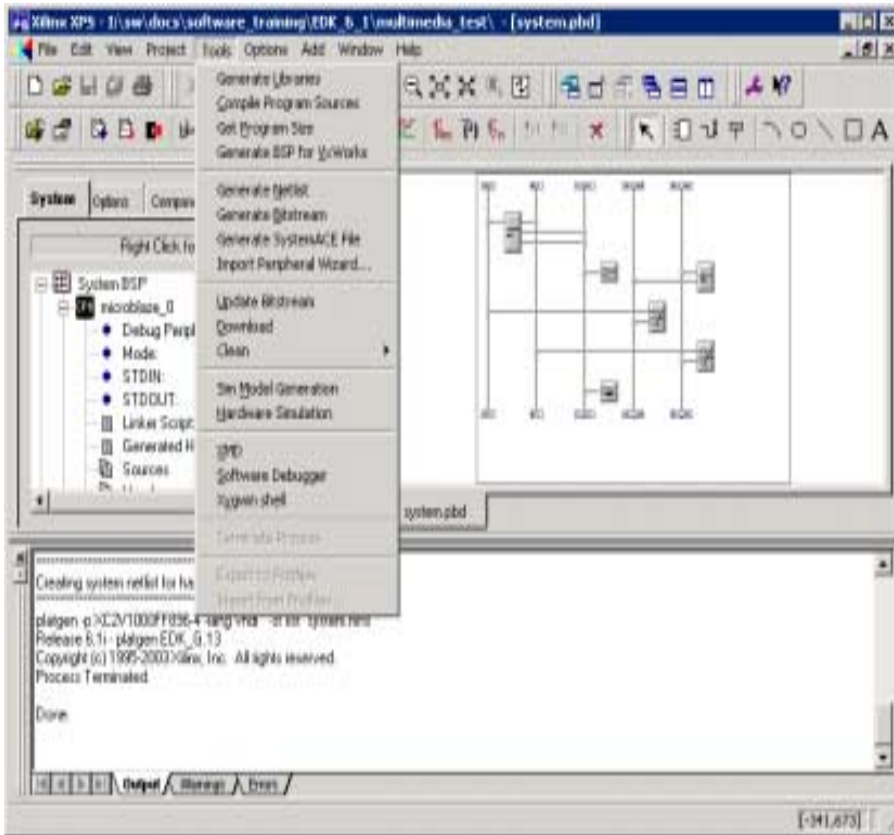
- A Library containing the device drivers and the startup code is built for an application to be linked against
- The Library helps separate Board Support Package development from application development
- Libraries will automatically be built if they don't exist when the application is built

Compiling The Software



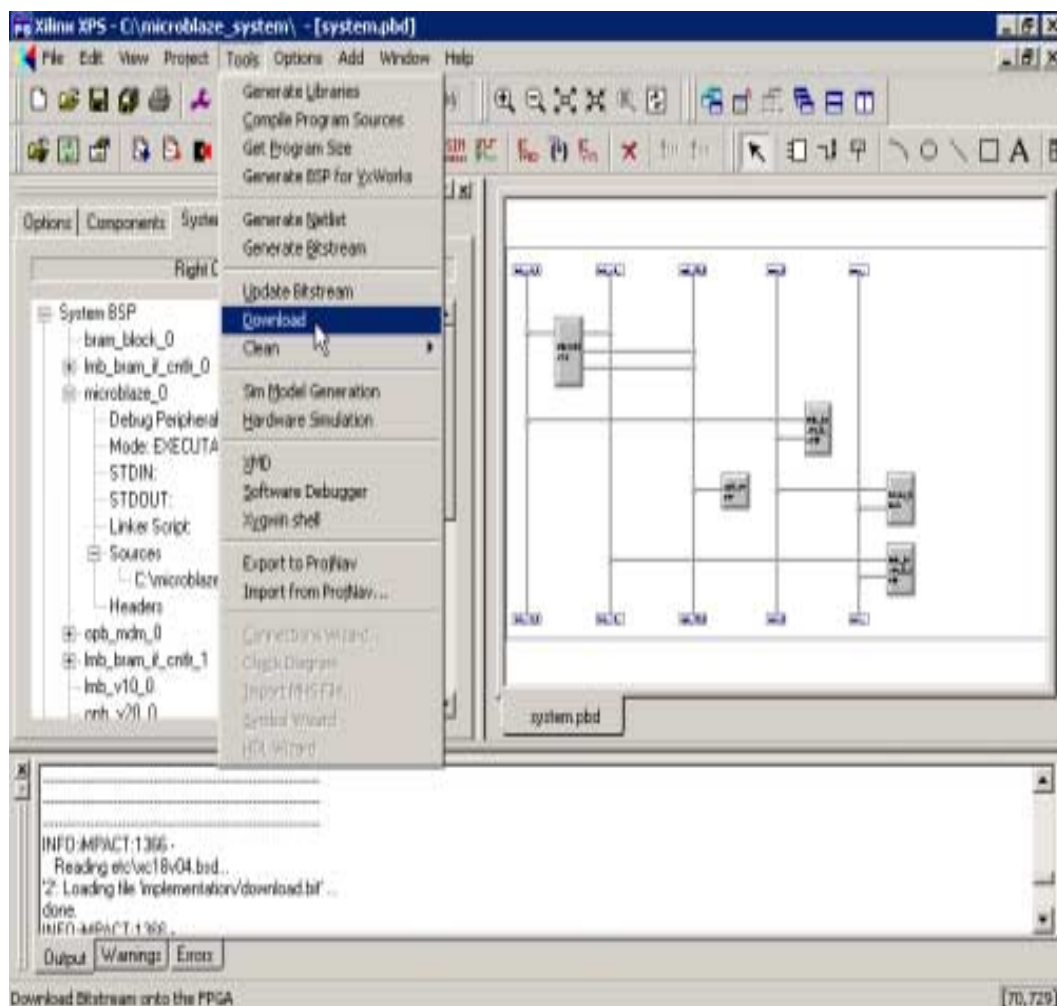
- Select the Tools menu
- Select the Compile Program Sources submenu
- Wait for the compile to complete

Updating the Bitstream



- Select the Tools menu
- Select the Update Bitstream submenu
- The hardware bitstream is updated to contain the contents of the software elf file

Downloading The Hardware



- Make sure that the board power is on and the parallel pod is connected
- Select the Tools menu
- Select the Download submenu
- Wait for the download to complete

Running XMD

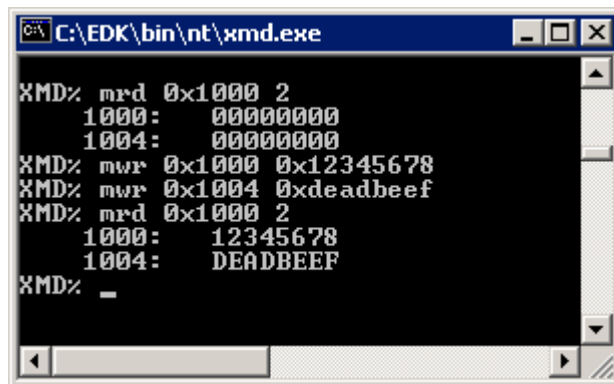
- Select the Tools menu
- Select the XMD submenu
- Type “mbconnect mdm” to connect XMD to the MicroBlaze processor.

```
C:\EDK_6_1\bin\nt\xmd.exe
Xilinx Microprocessor Debug (XMD) Engine
Xilinx EDK 6.1.1 Build EDK_G.13
Copyright (c) 1995-2002 Xilinx, Inc. All rights reserved.
XMD% mbconnect mdm

JTAG chain configuration
-----
Device   ID Code      IR Length  Part Name
1        0a001093      8          System_ACE
2        01028093      6          XC2V1000
Assuming, Device No: 2 contains the MicroBlaze system
Connected to the JTAG MicroBlaze Debug Module (MDM)
No of processors = 1
MicroBlaze Configuration :
Version - 2.00.a
No of PC HW Breakpoints : 8
No of Read Addr/Data Watchpoints : 1
No of Write Addr/Data Watchpoints : 1
Instruction Cache Support : off
Data Cache Support : off

Connected to MicroBlaze "mdm" target. id = 0
Starting GDB server for "mdm" target (id = 0) at TCP port no 1234
XMD% _
```

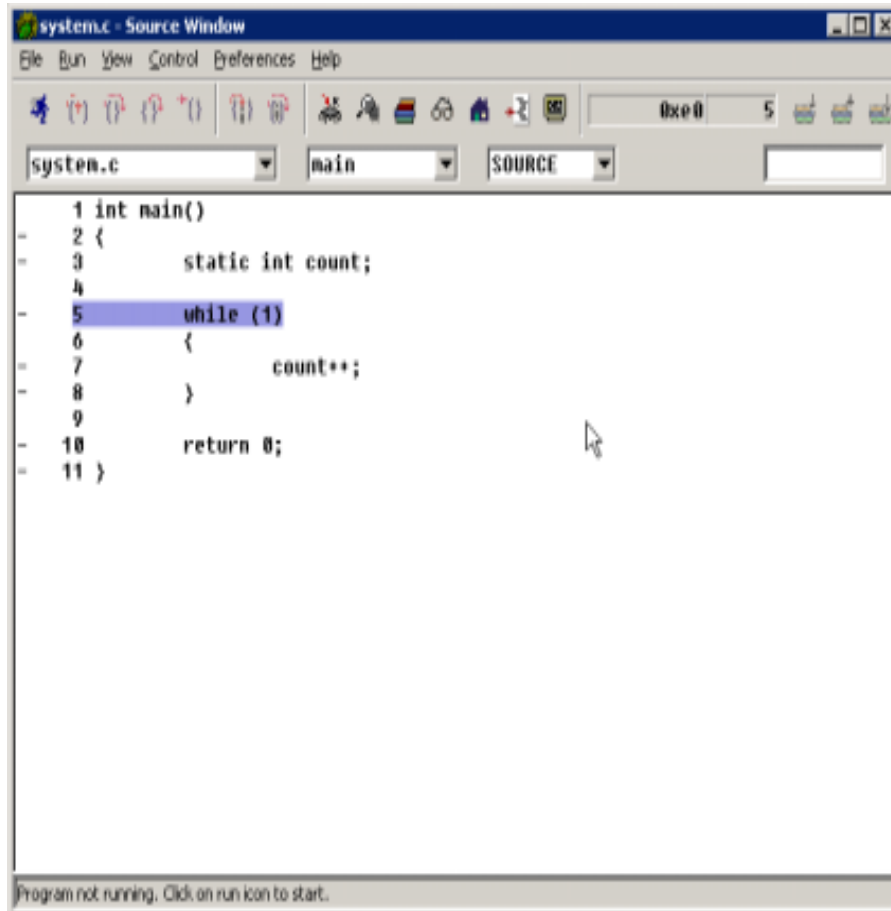
Testing Memory



```
C:\EDK\bin\nt\xmd.exe
XMD% mrd 0x1000 2
1000: 00000000
1004: 00000000
XMD% mwr 0x1000 0x12345678
XMD% mwr 0x1004 0xdeadbeef
XMD% mrd 0x1000 2
1000: 12345678
1004: DEADBEEF
XMD% -
```

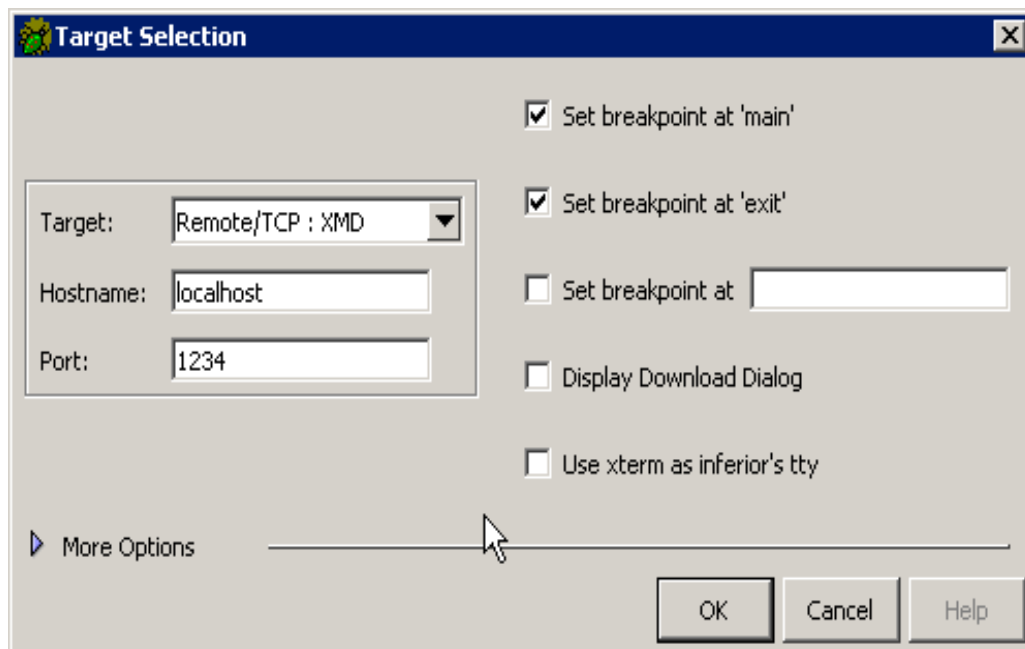
- Type "mrd 0x1000 2" to read 2 memory locations starting at address 0x1000
- Type "mwr 0x1000 0x12345678" to write to memory location 0x1000
- Perform writes to location 0x1004 also
- Type "mrd x1000 2" to read 2 memory locations and verify the values that were written

Starting The GNU Debugger (GDB)



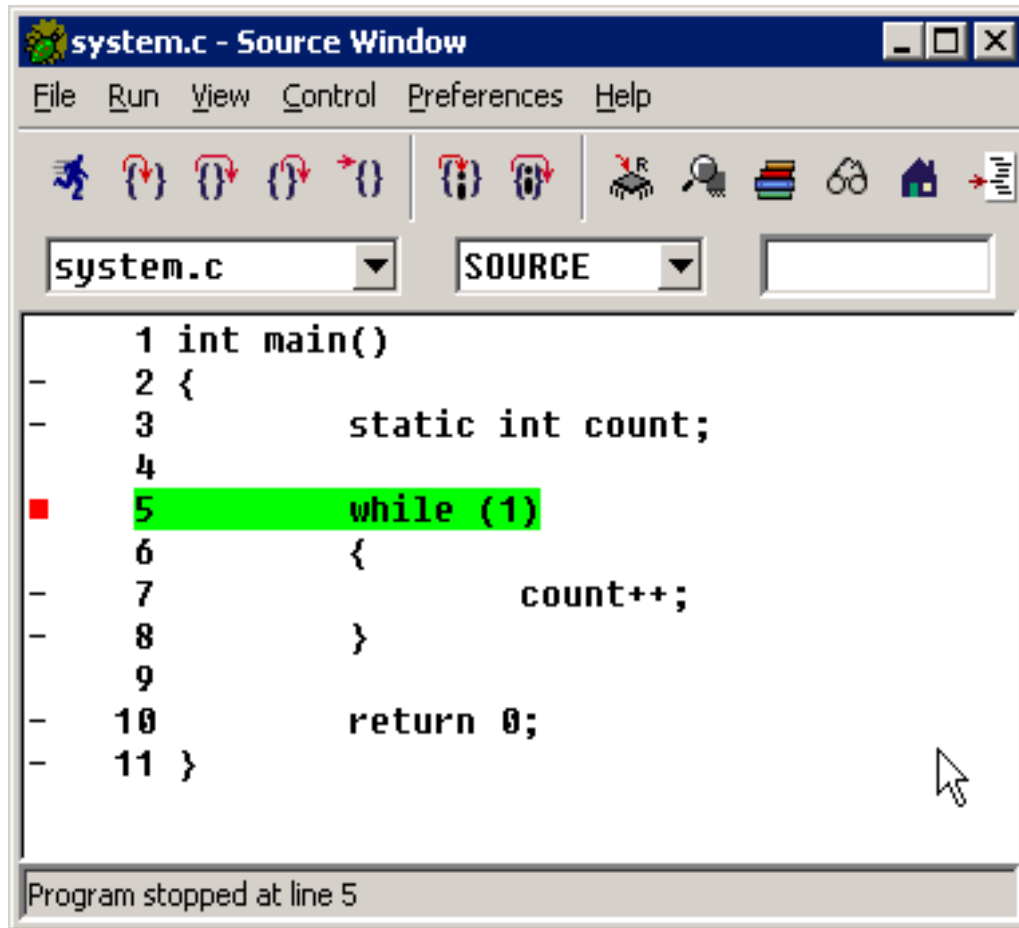
- Select the Tools menu
- Select the Software Debugger submenu

Target Selection & Download



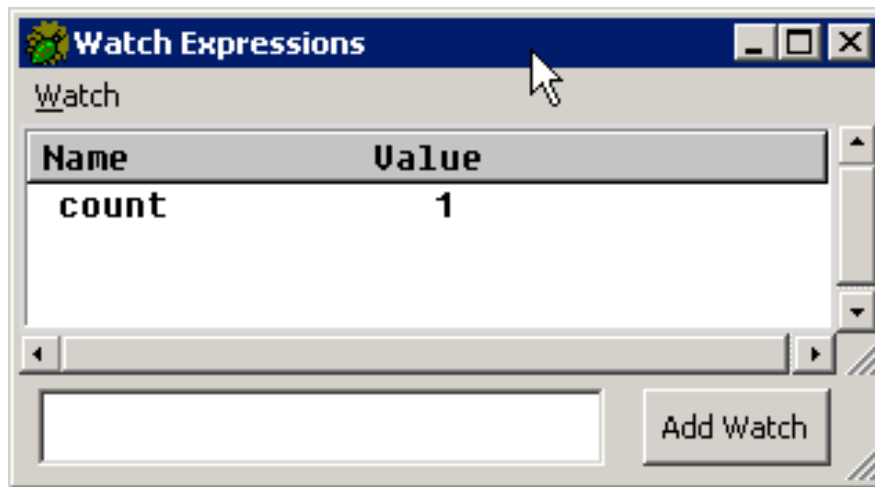
- In GDB, select the Run menu and choose the Run menu item
- Wait for the Target Selection dialog box to be displayed
- Enter the data in the dialog box and click OK

Stepping in GDB



- The debugger is ready, the program counter is at a breakpoint at line 5 of the source file
- Select the Control menu
- Select the Step submenu

Watching A Variable With GDB



- Double click on the variable count such that it is selected
- Right click and select in the submenu to Add count to Watch
- A dialog box is displayed which contains the variable count and it's contents



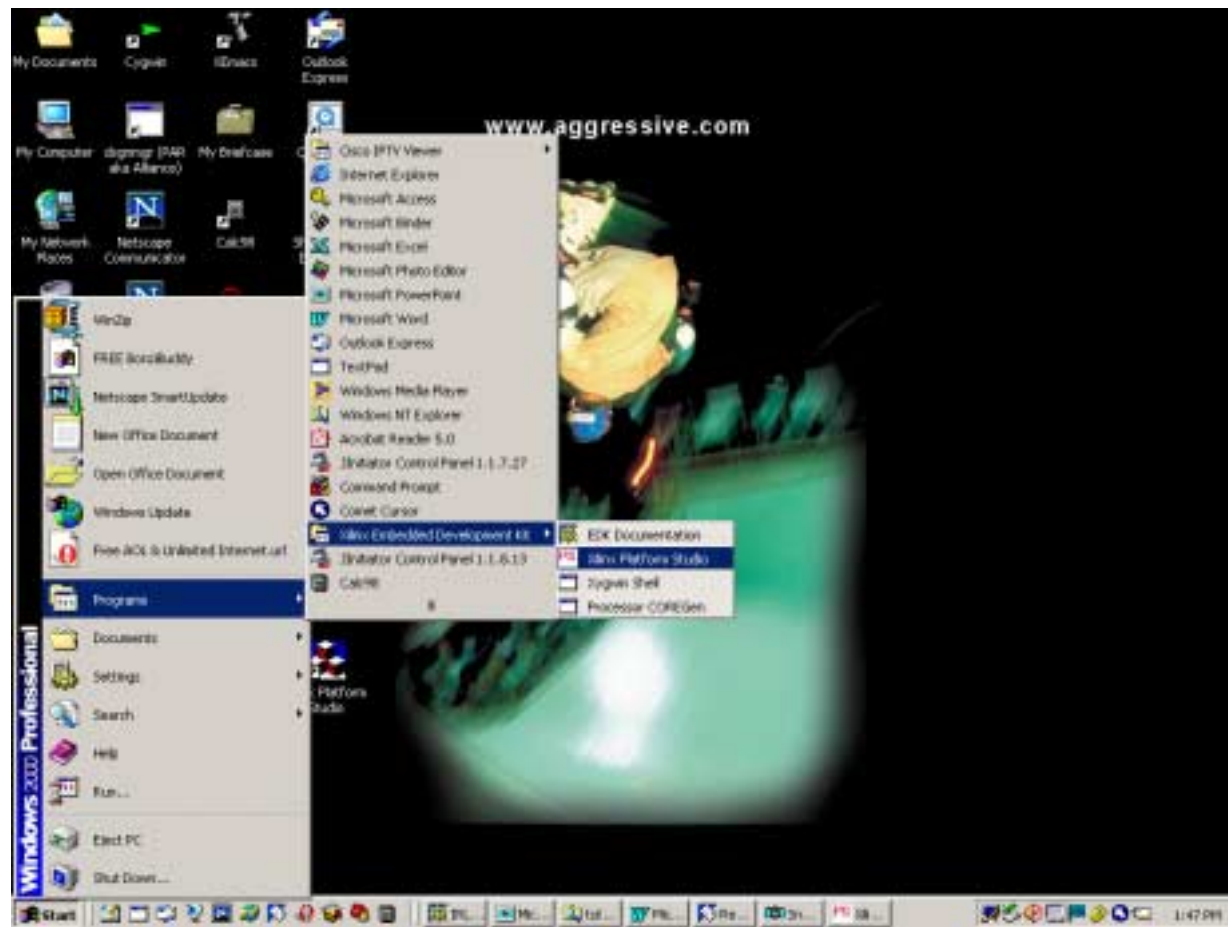
Creating a Simple MicroBlaze System with XPS

without Base System Builder Wizard

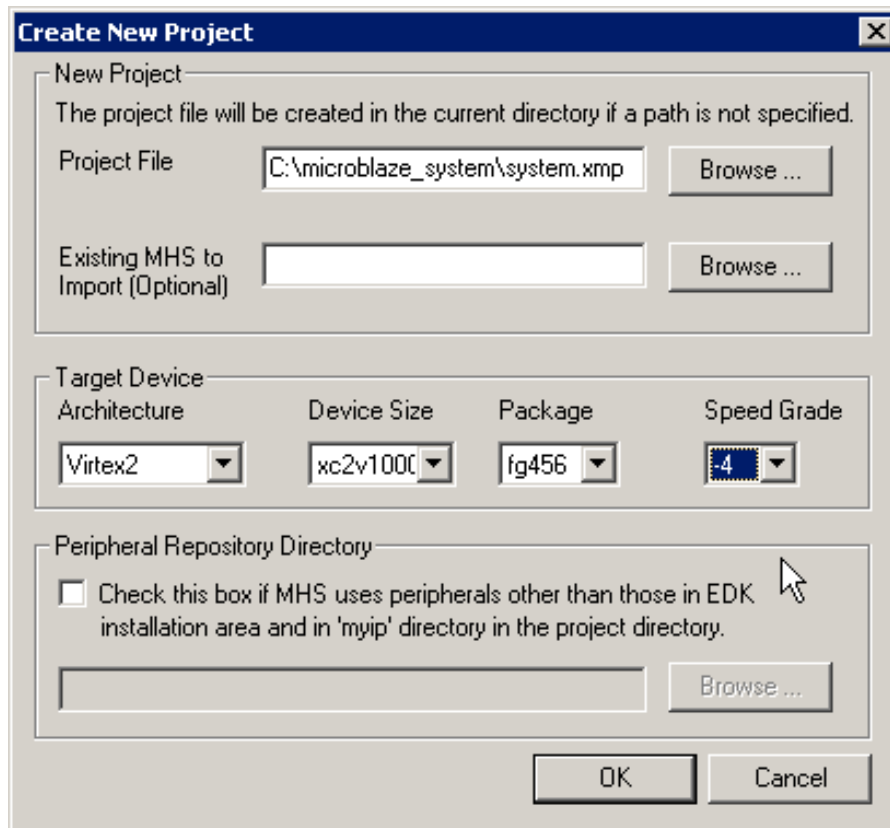
Design Flow

- Design Entry with Xilinx Platform Studio
- Generate system netlist with XPS
- Generate hardware bitstream with XPS
- Download and sanity check design with XPS and XMD

Start Xilinx Platform Studio

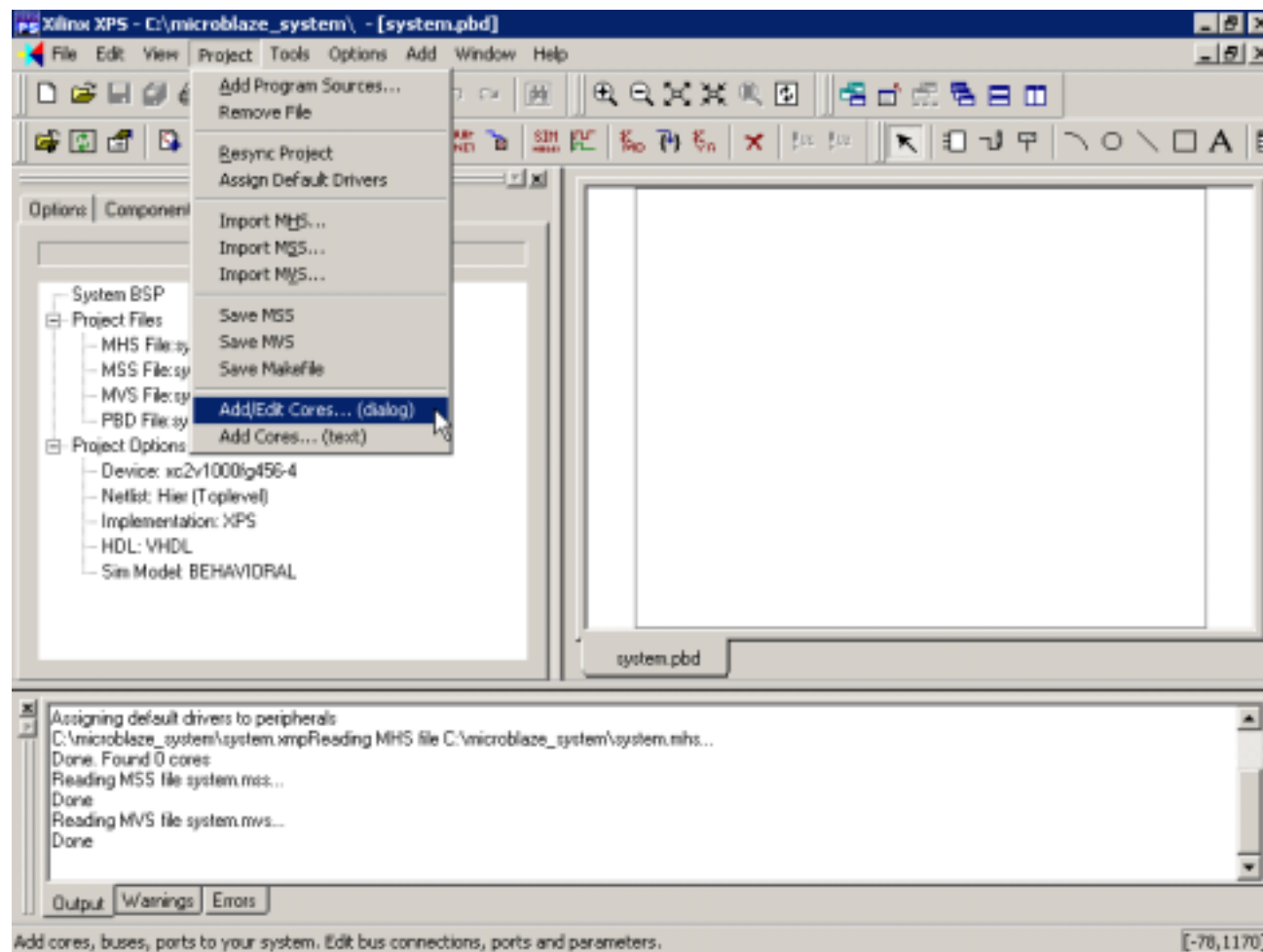


Create A New Project



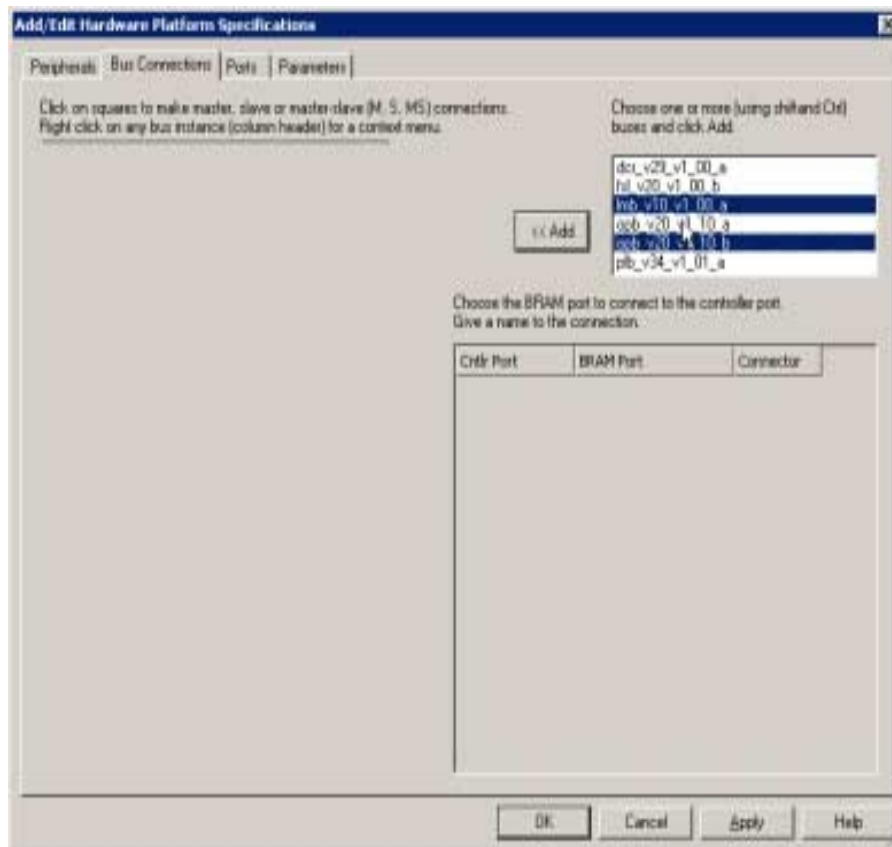
- Select New Project from the Tools menu
- Enter all the project information
- Click OK on this dialog box.
- Click Yes on the next dialog box to start with an empty MHS file

Adding New Cores



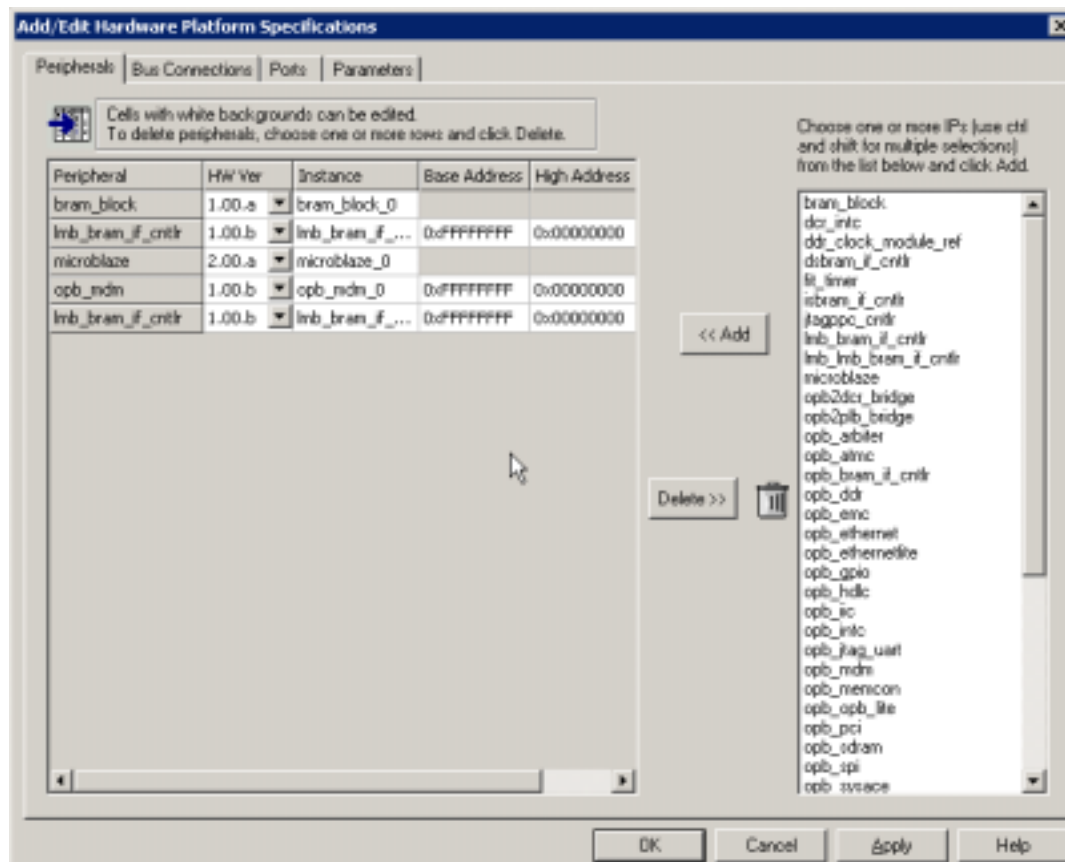
- Select the Project menu
- Select the Add/Edit Cores submenu

Adding Bus Structures



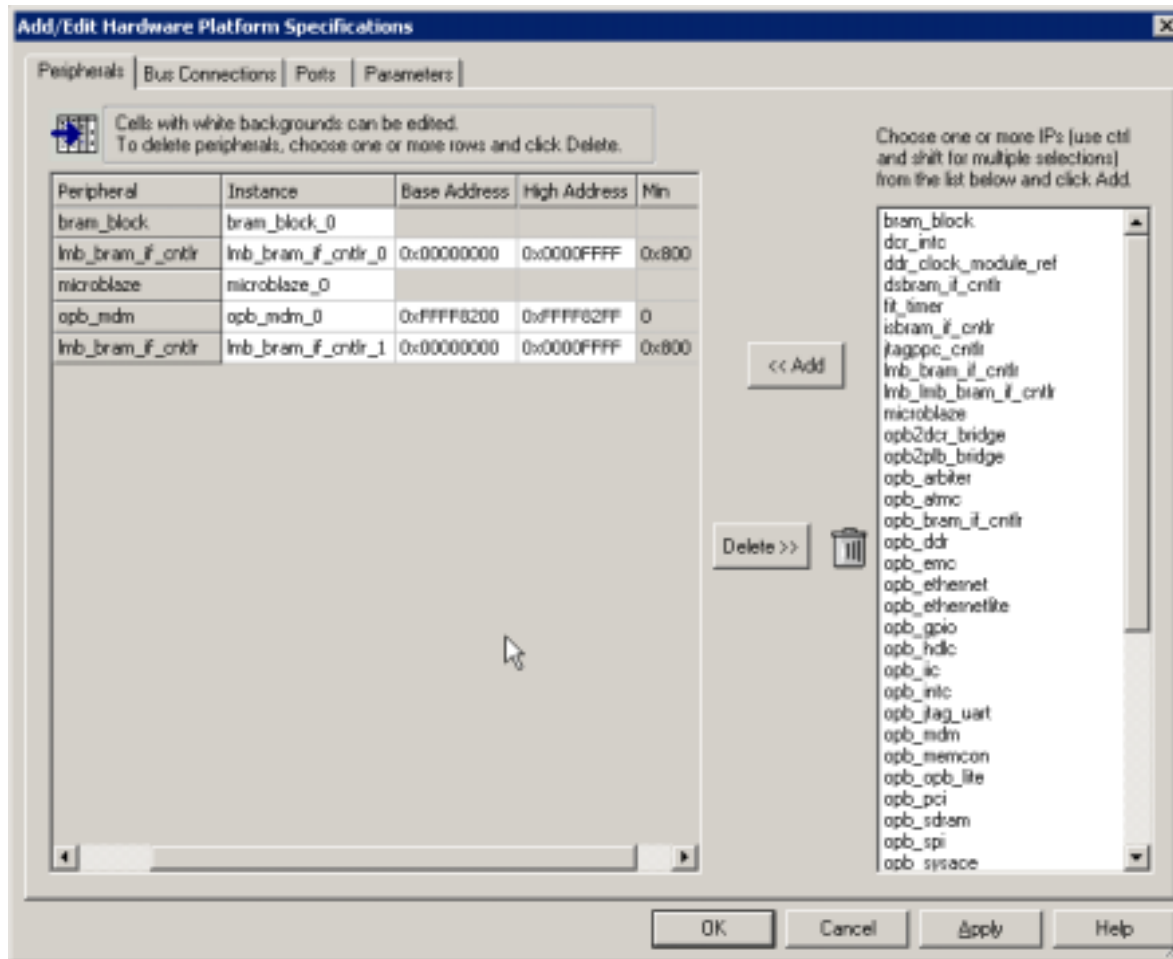
- Select the Bus Connections Tab
- Select the lmb_v10_v1_00_a bus & opb_v20_v1_10_b bus and click the Add button
- Select the lmb_v10_v1_00_a bus and click the Add button

Adding Basic Peripherals



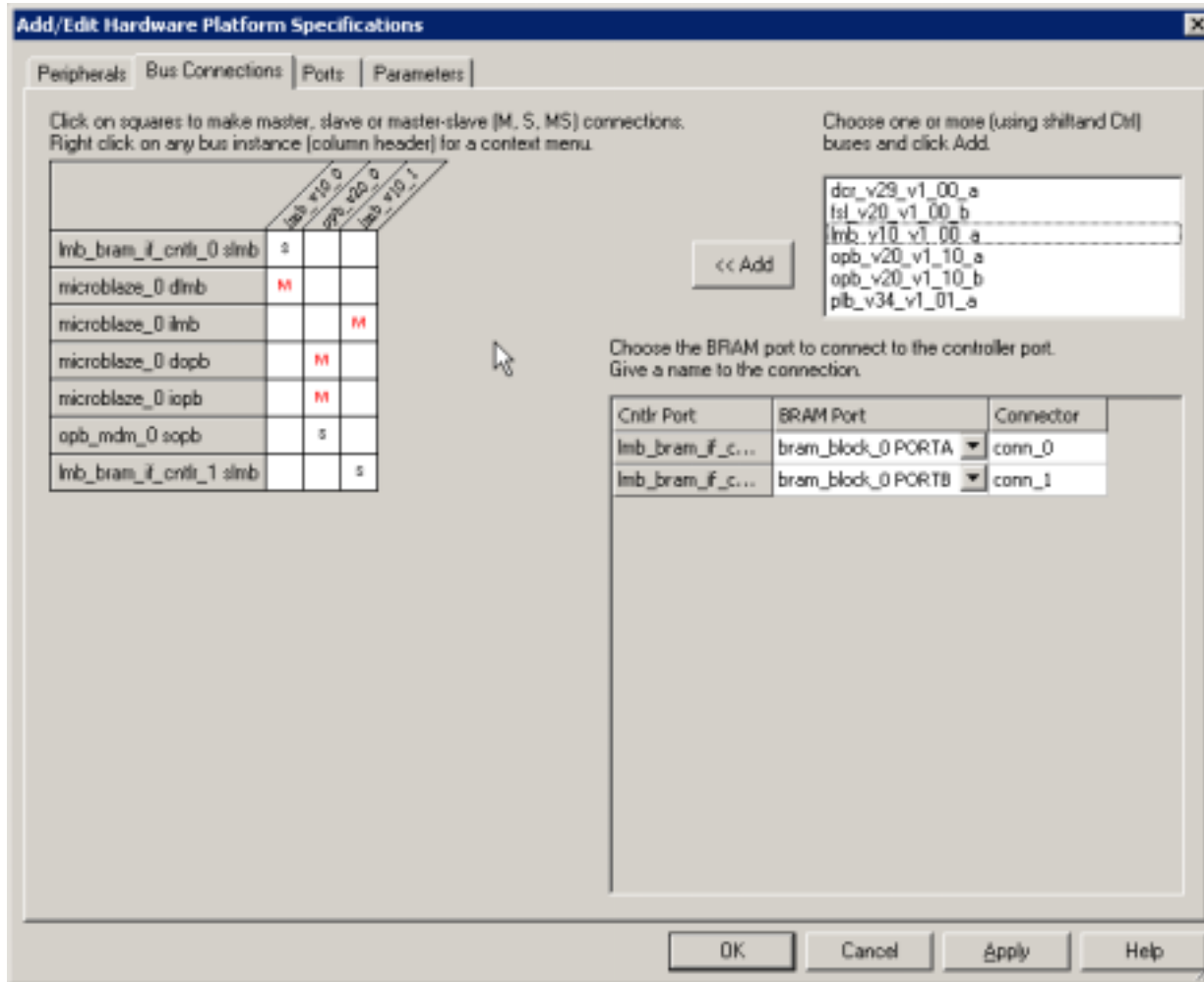
- Select the Peripherals tab
- Select the bram_block, lmb_bram_if_cntlr, microblaze, and opb_mdm and click the Add button.
- Select the lmb_bram_if_cntlr and click the Add button.

Change The Memory Map



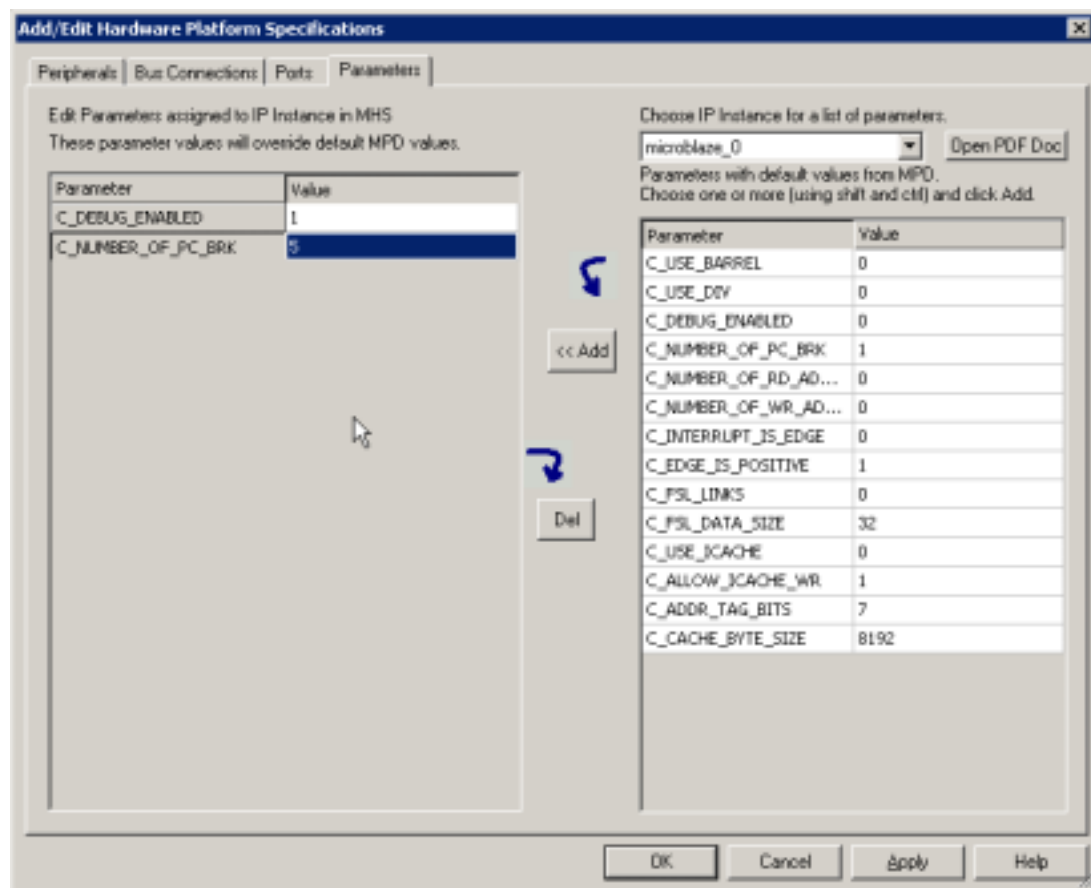
- Edit the Base Address and High Address for the lmb_bram_if_cntlr and opb_mdm peripherals

Setting Bus Masters & Slaves



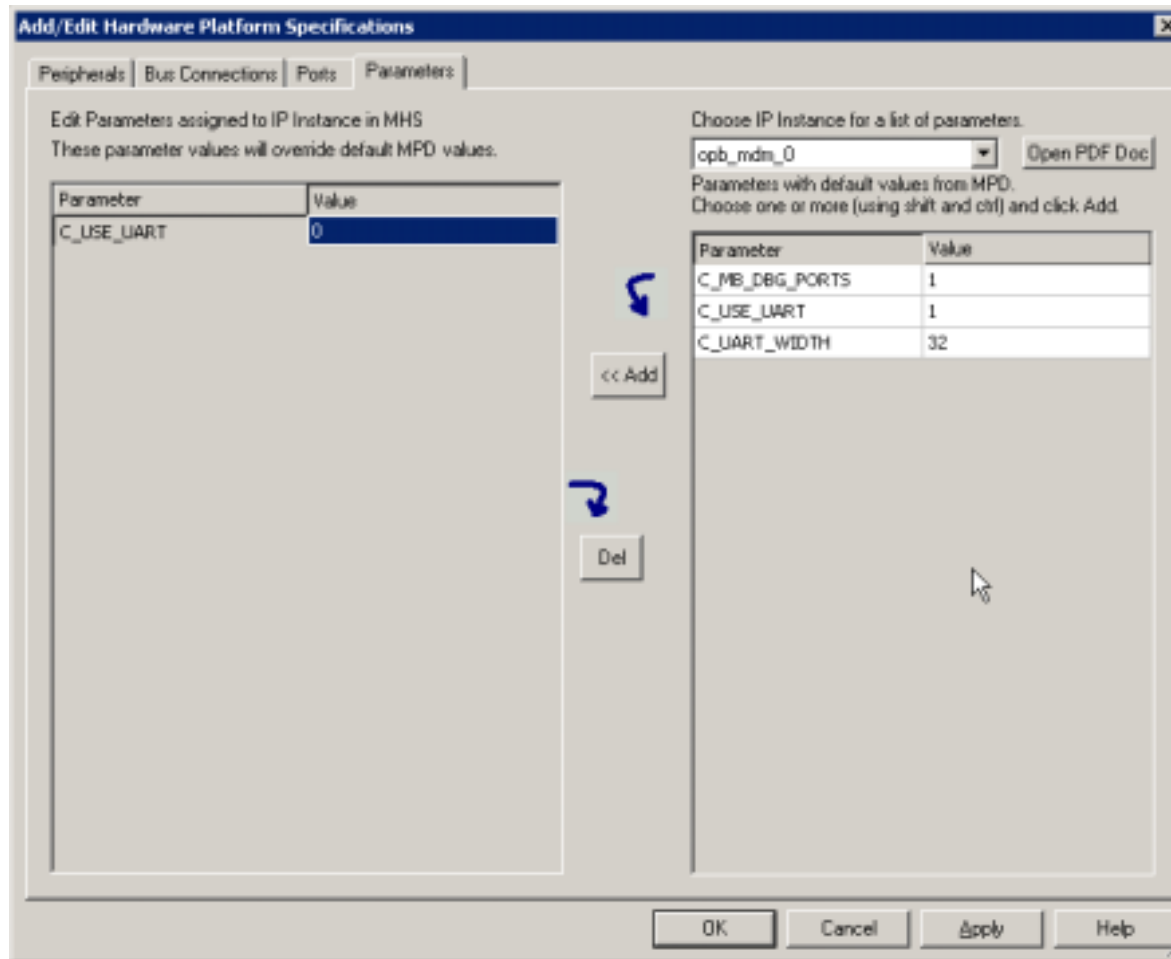
- Select the Bus Connections tab
- Set the masters and slaves on the buses by clicking on the boxes with an 's' and 'M'

Setting MicroBlaze Parameters



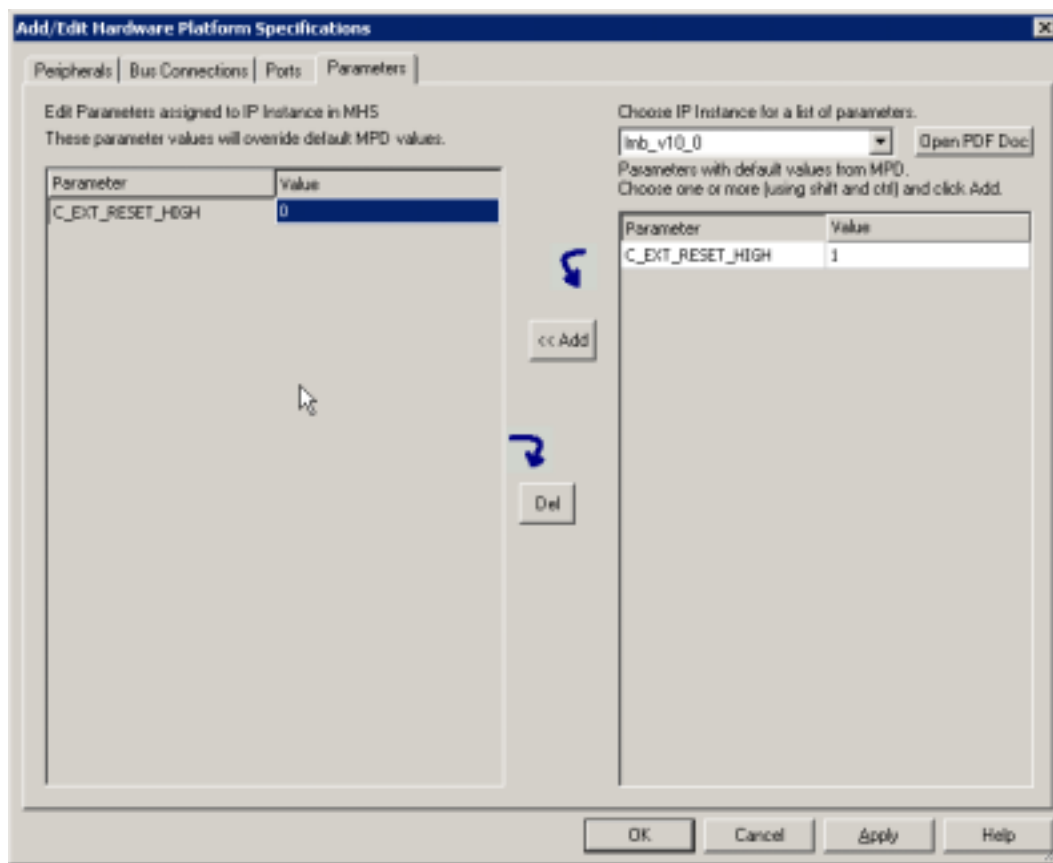
- Select the Parameters tab
- Select microblaze_0 IP instance
- Select the C_DEBUG_ENABLED and C_NUMBER_OF_PC_BRK parameters and click the Add button
- Edit the parameter values

Setting MDM Parameters



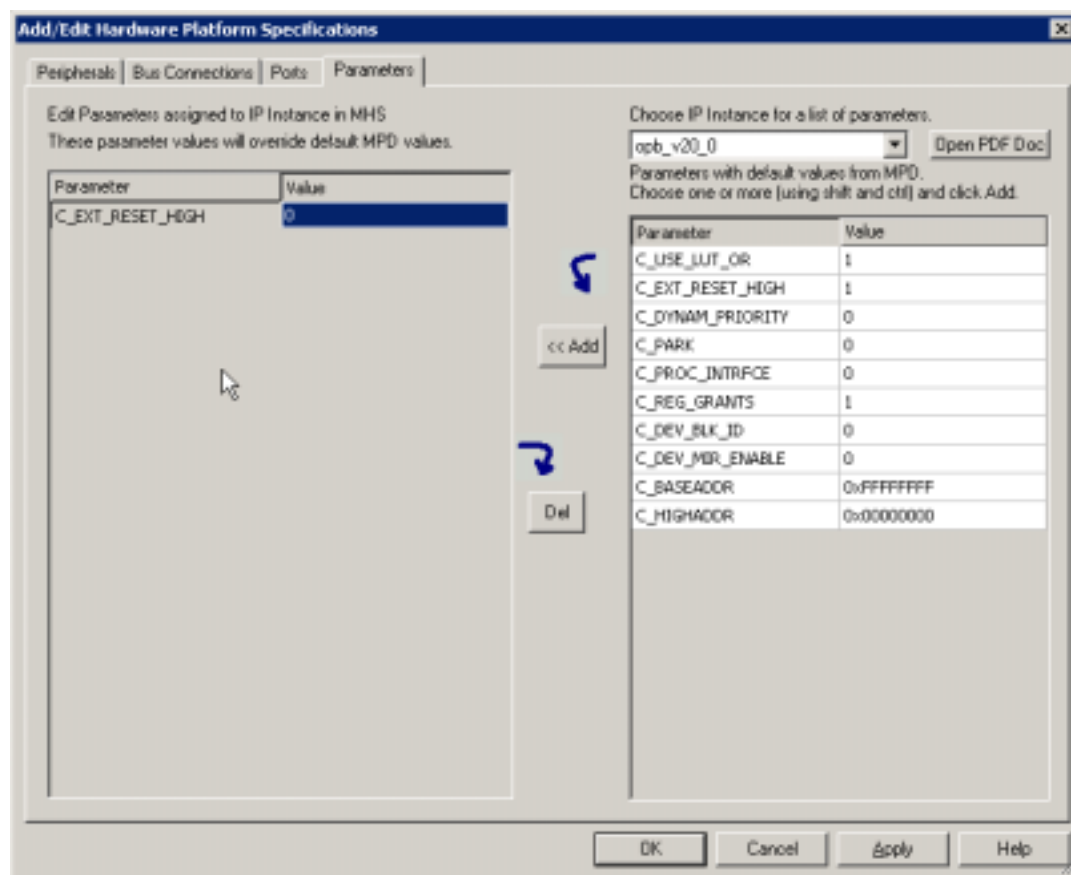
- Select the opb_mdm_0 IP instance
- Select the C_USE_UART parameter and click the Add button
- Edit the parameter value

Setting LMB Parameters



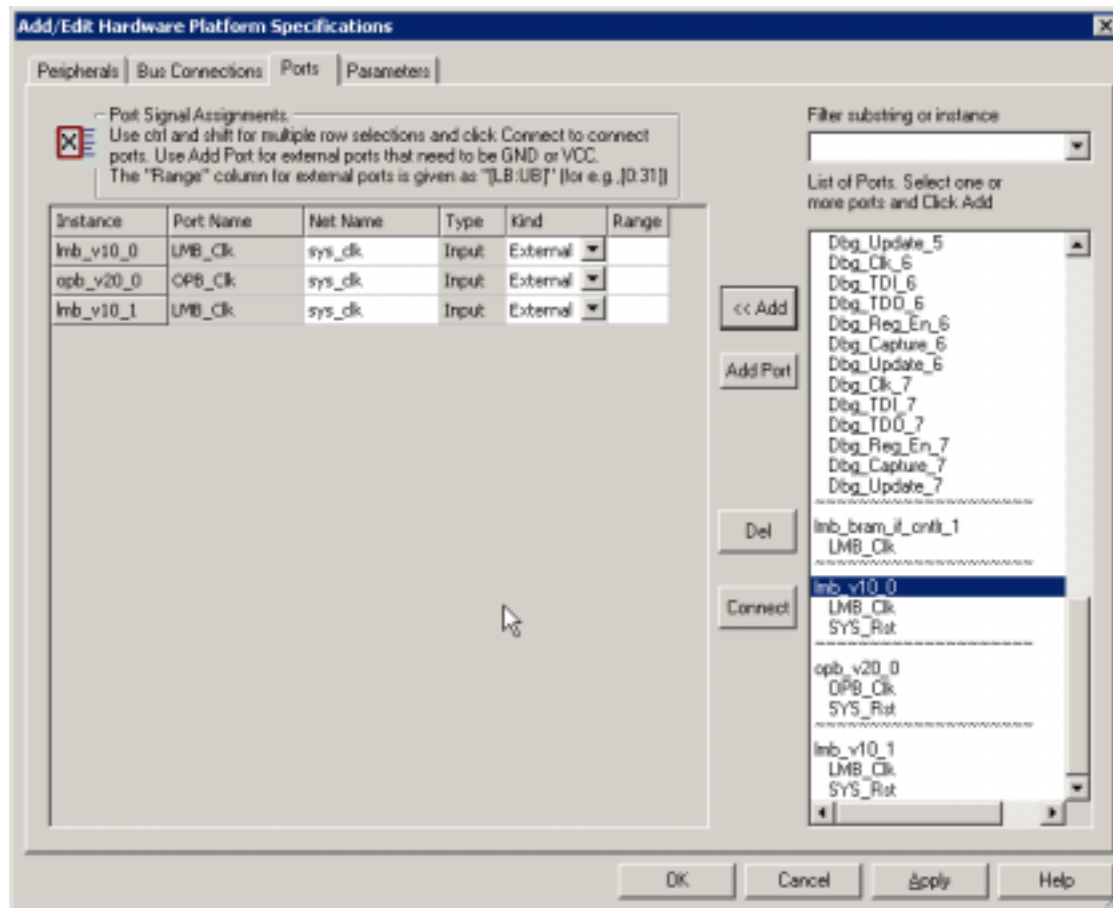
- Select the lmb_v10_0 IP instance
- Select the C_EXT_RESET_HIGH parameter and click the Add button
- Edit the parameter value
- Repeat for lmb_v10_1 IP instance

Setting OPB Parameters



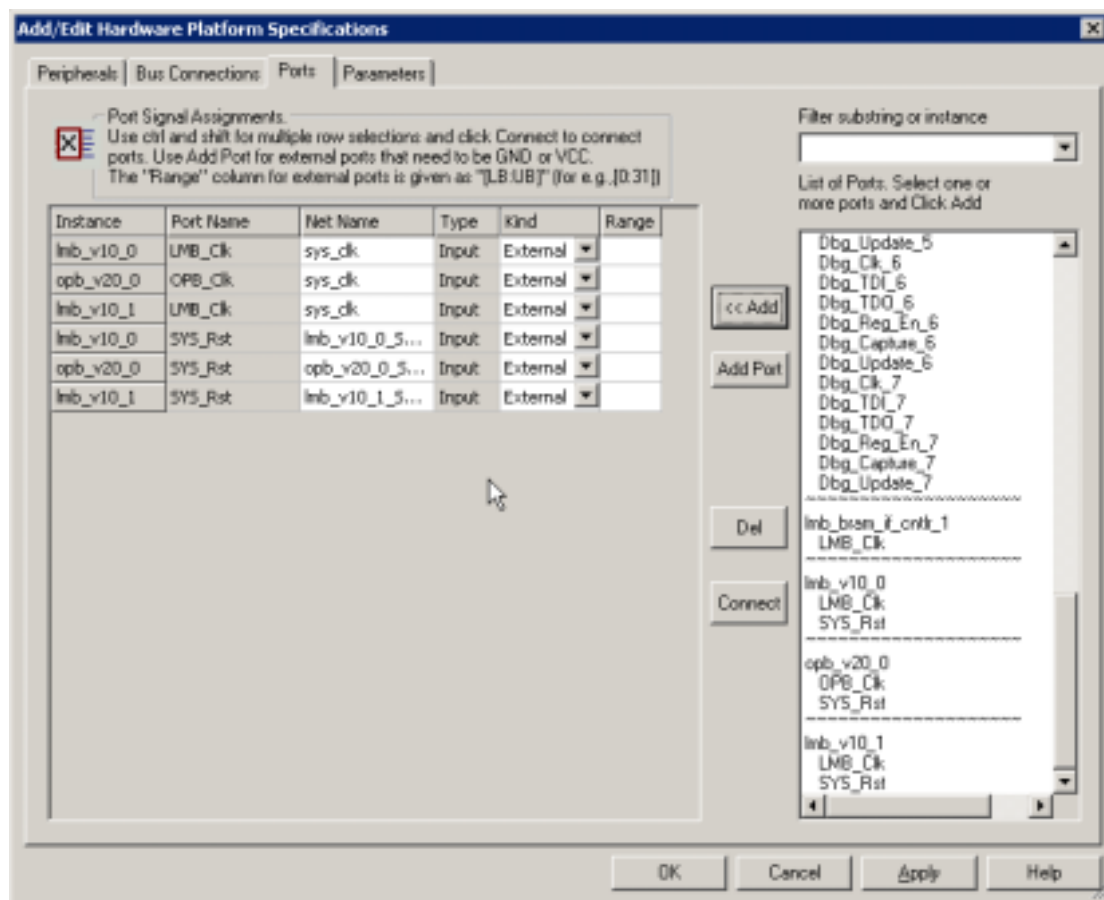
- Select the opb_v20_0 IP instance
- Select the C_EXT_RESET_HIGH parameter and click the Add button
- Edit the parameter value

Connecting The Clock



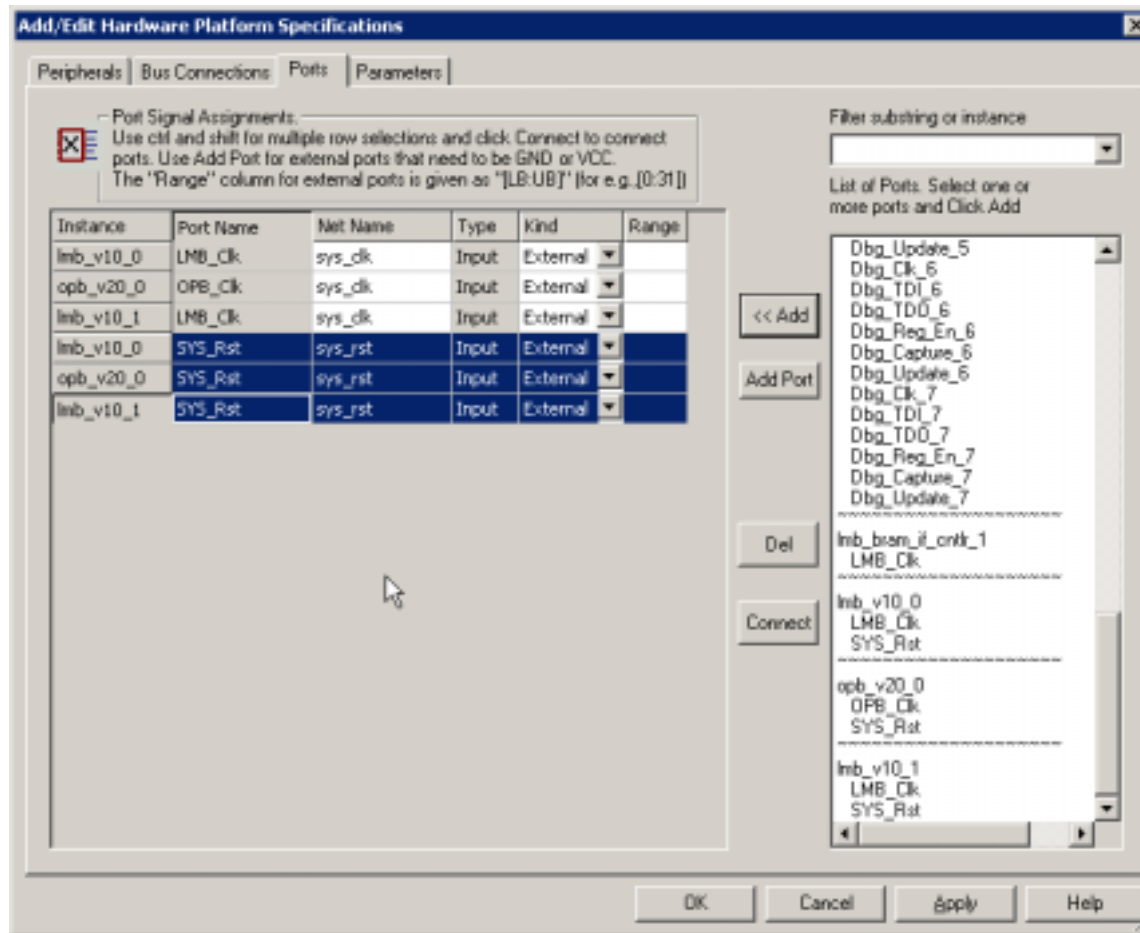
- Select the Ports tab
- Select the LBM_Clk and OPB_Clk ports on the lmb_v10_0, lmb_v10_0 and opb_v20_0 IP instances and click the Add button

Connecting The Reset



- Select the SYS_Rst ports of the lmb_v10_0, lmb_v10_1, and opb_v20_0 IP instances and click the Add button

Connecting The Reset (2)



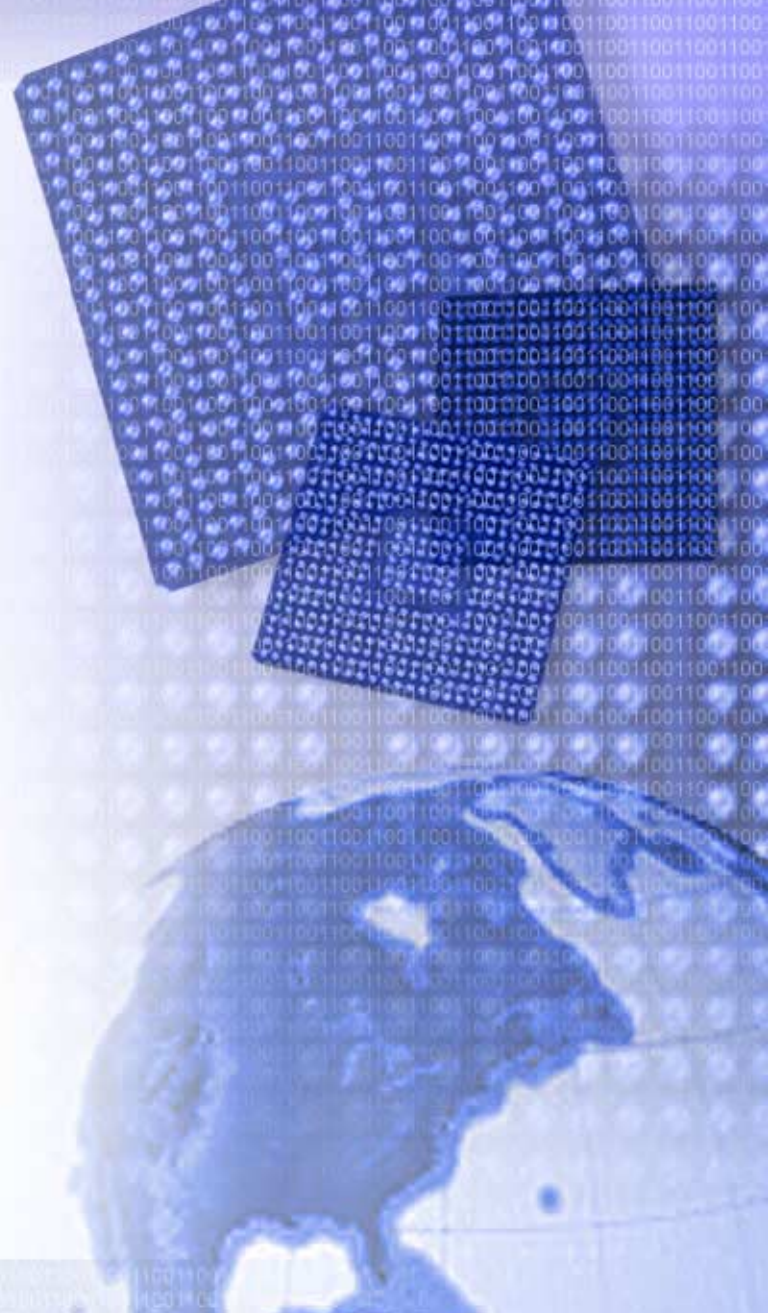
- Select the lmb_v10_0, lmb_v10_1, and opb_v20_0 instances on the left side and click the Connect button
- Enter sys_rst for the net name in the dialog box and click the OK button

Completing the Add/Edit

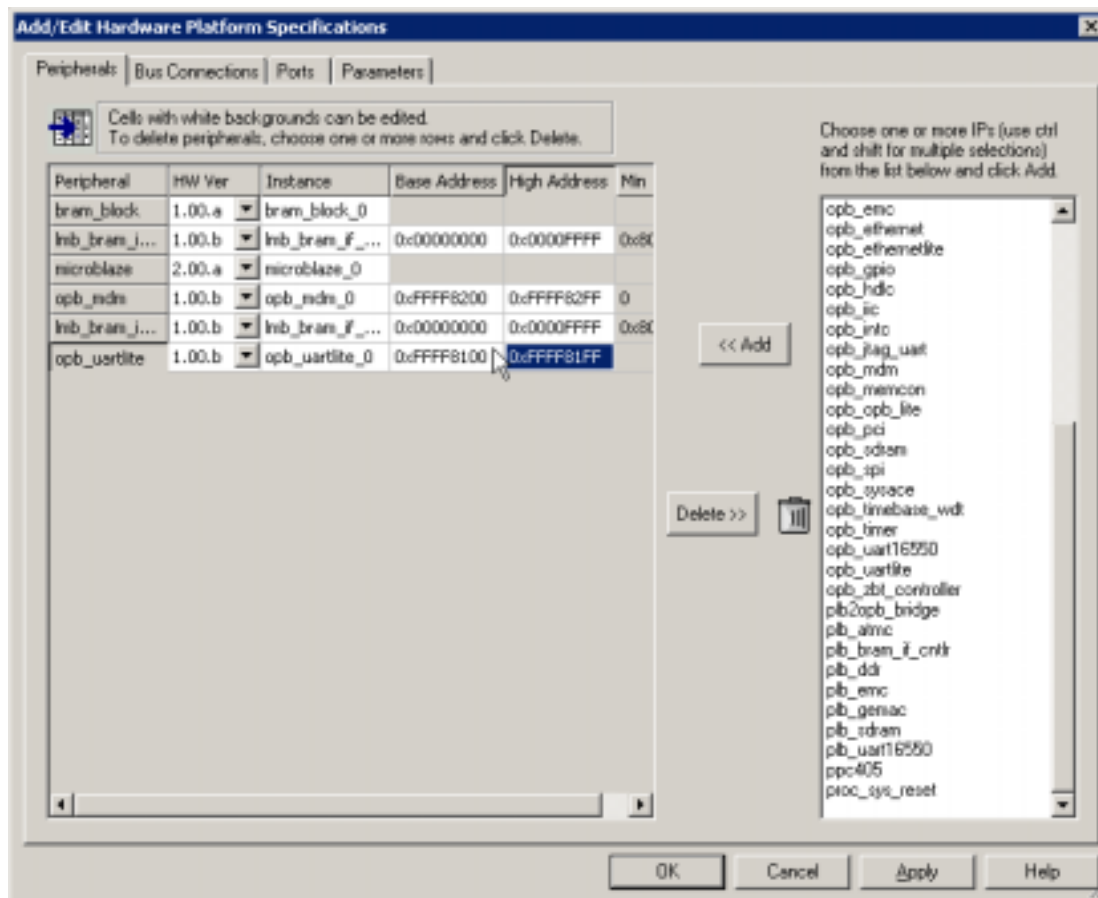
- Click the OK button to set all the items changed in the Add/Edit Cores dialog box



Adding I/O Peripherals to a System

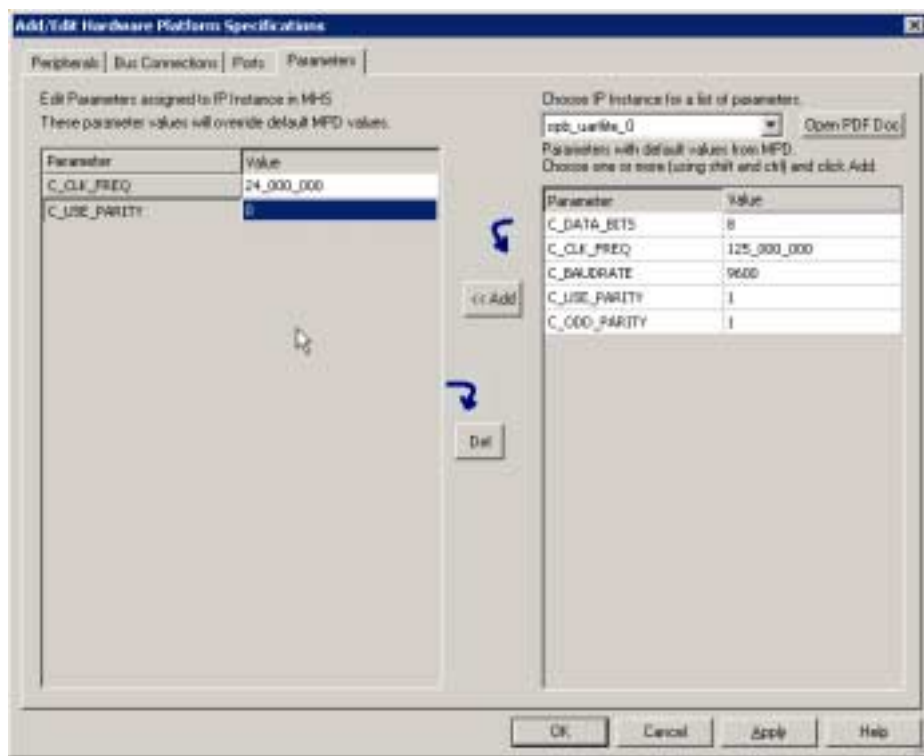


Adding A UartLite



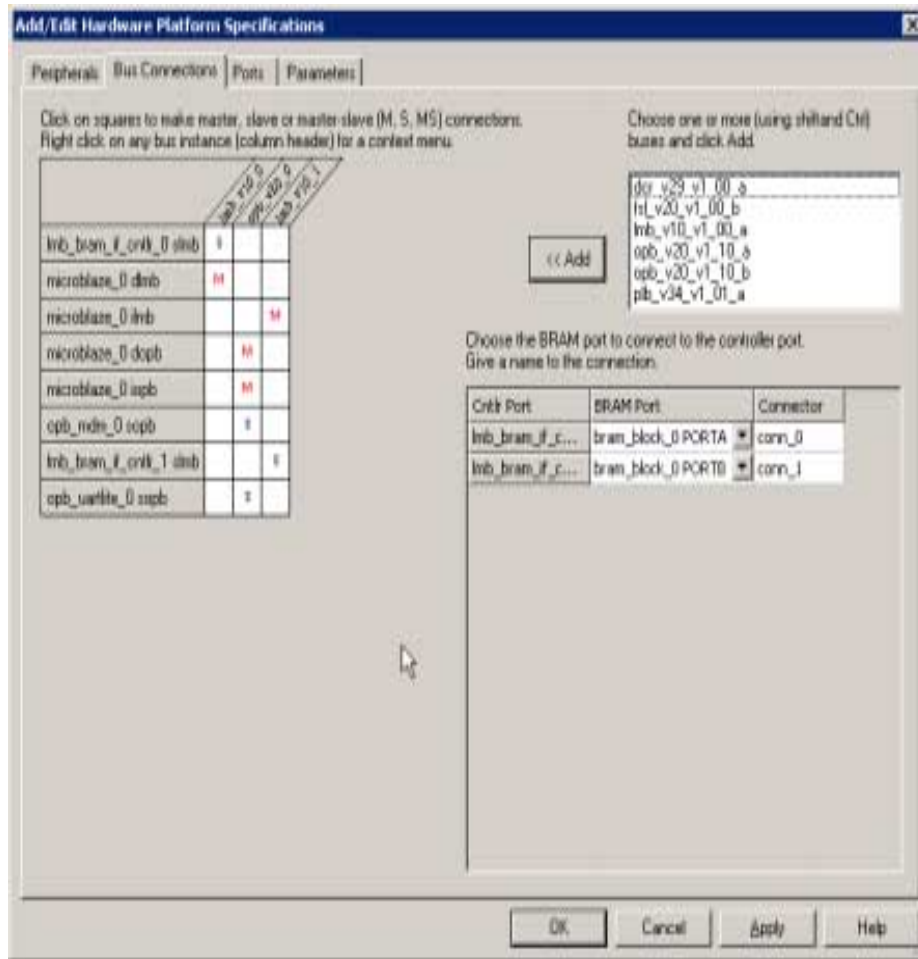
- Select the Project menu
- Select the Add/Edit Cores submenu
- Select the opb_uartlite and click Add
- Edit the Base Address and High Address of the uartlite

Setting UartLite Parameters



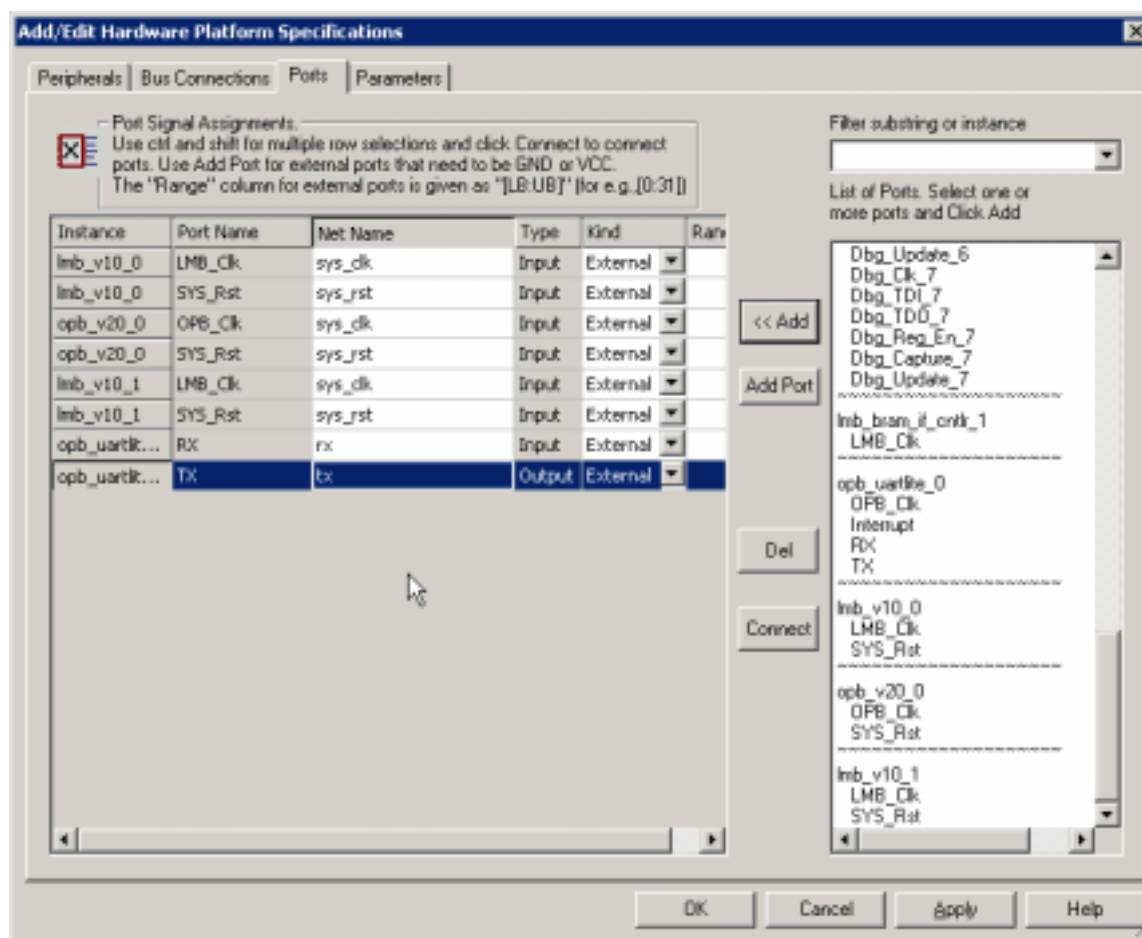
- Select the Parameters tab
- Select the C_CLK_FREQ and C_USE_PARITY parameters and click the Add button
- Edit the parameter values

Put UartLite On the OPB



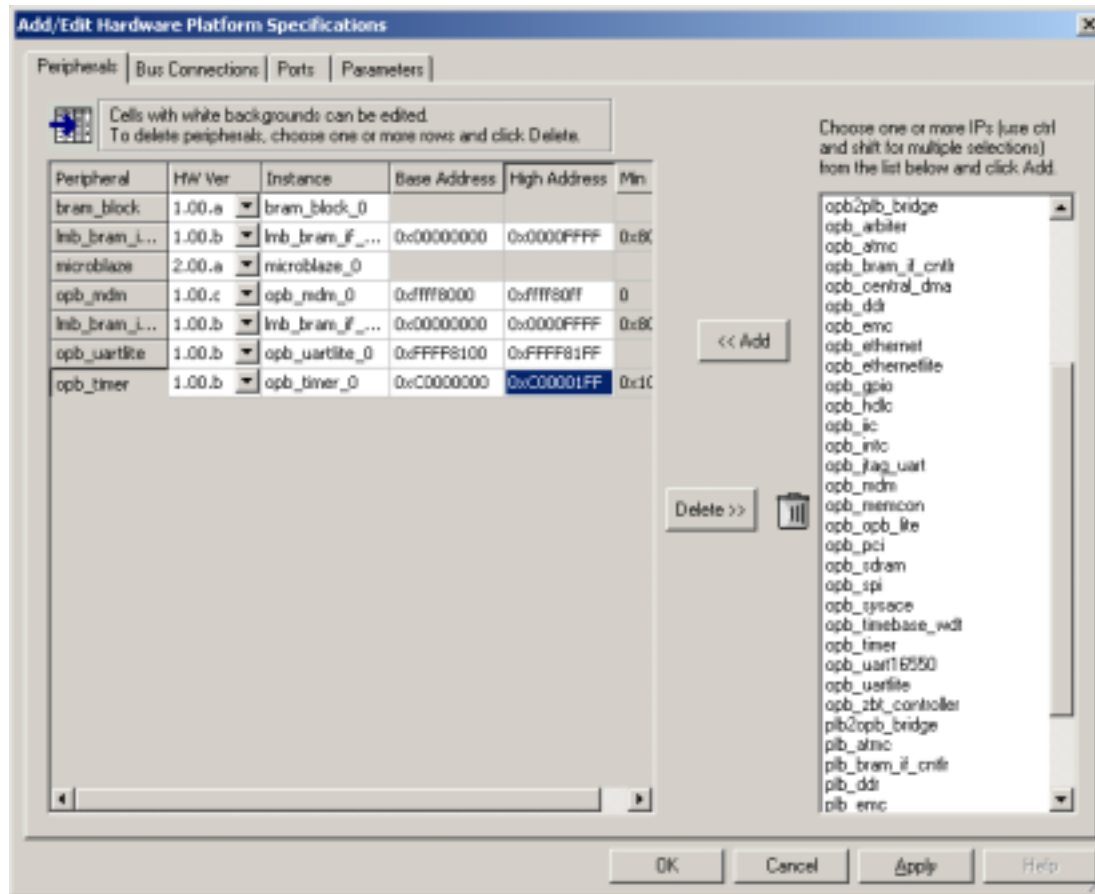
- Select the Bus Connections tab
- Click on the box to mark the UartLite as a slave on the OPB bus

Connecting UartLite I/O



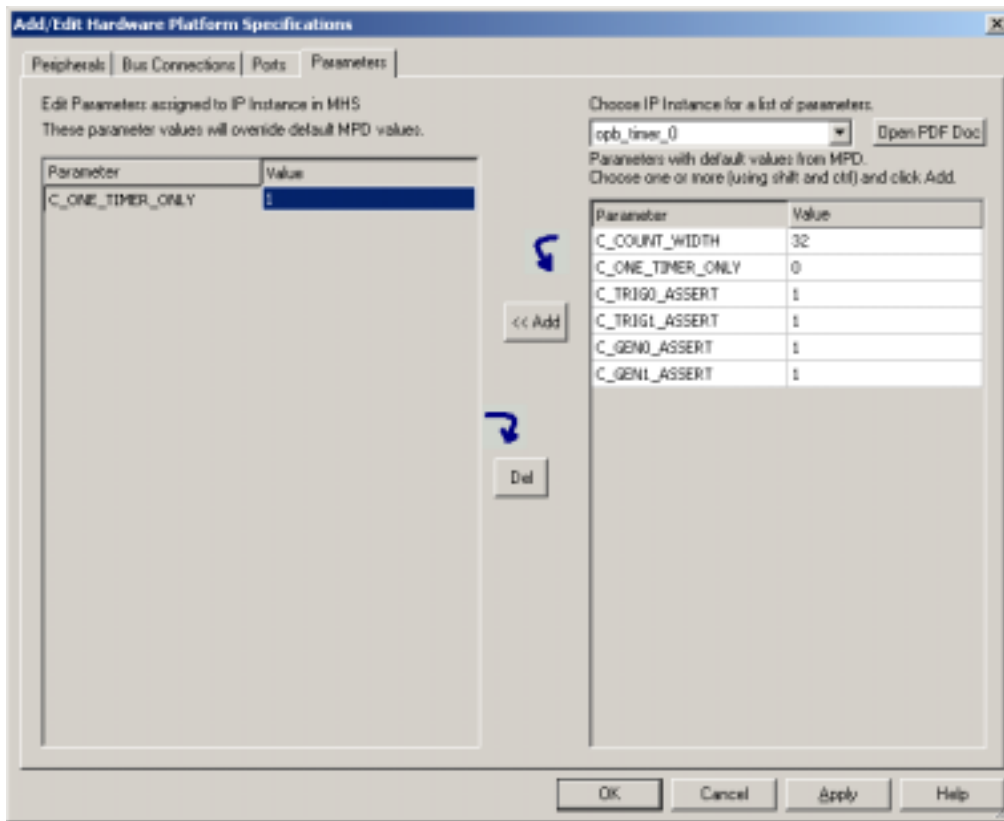
- Select the Ports tab
- Select the RX and TX ports of opb_uartlite_0 and click the Add button
- Edit the net names to be tx and rx for the opb_uartlite_0
- Click OK on the dialog box

Adding A Timer



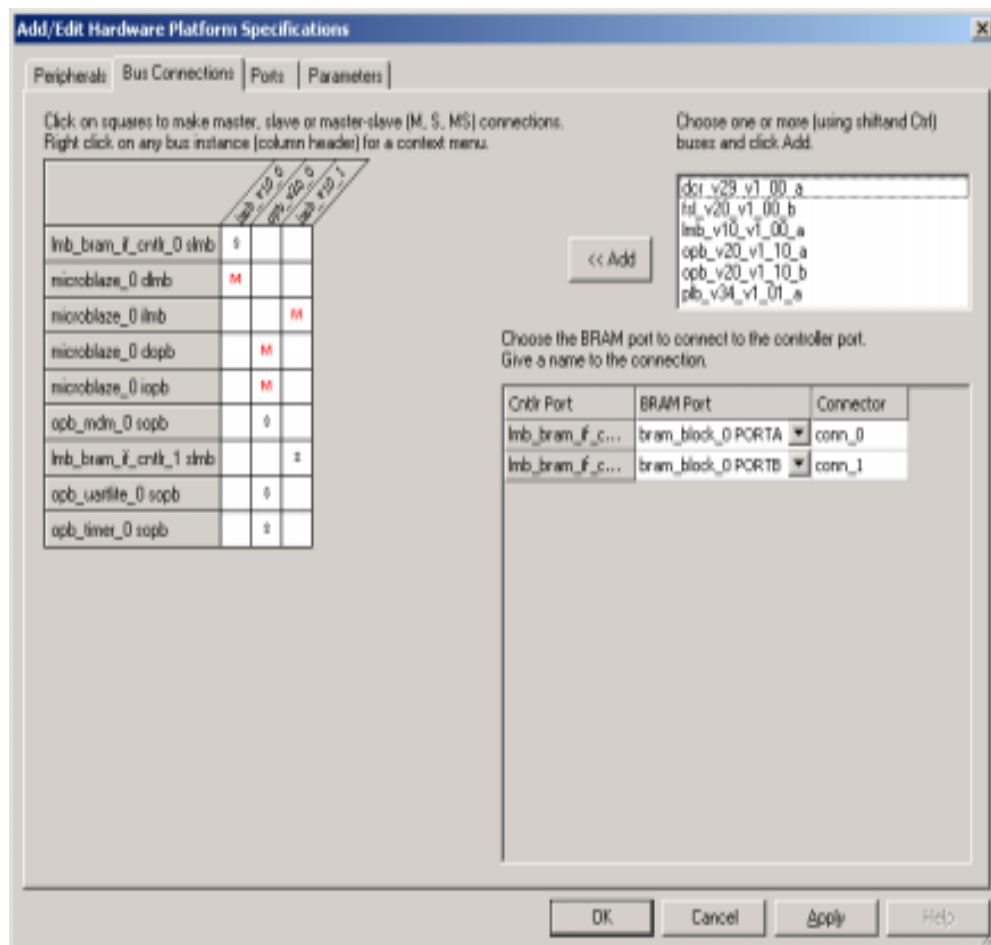
- Select the Project menu
- Select the Add/Edit Cores submenu
- Select the opb_timer and click Add
- Edit the Base Address and High Address of the timer

Setting Timer Parameters



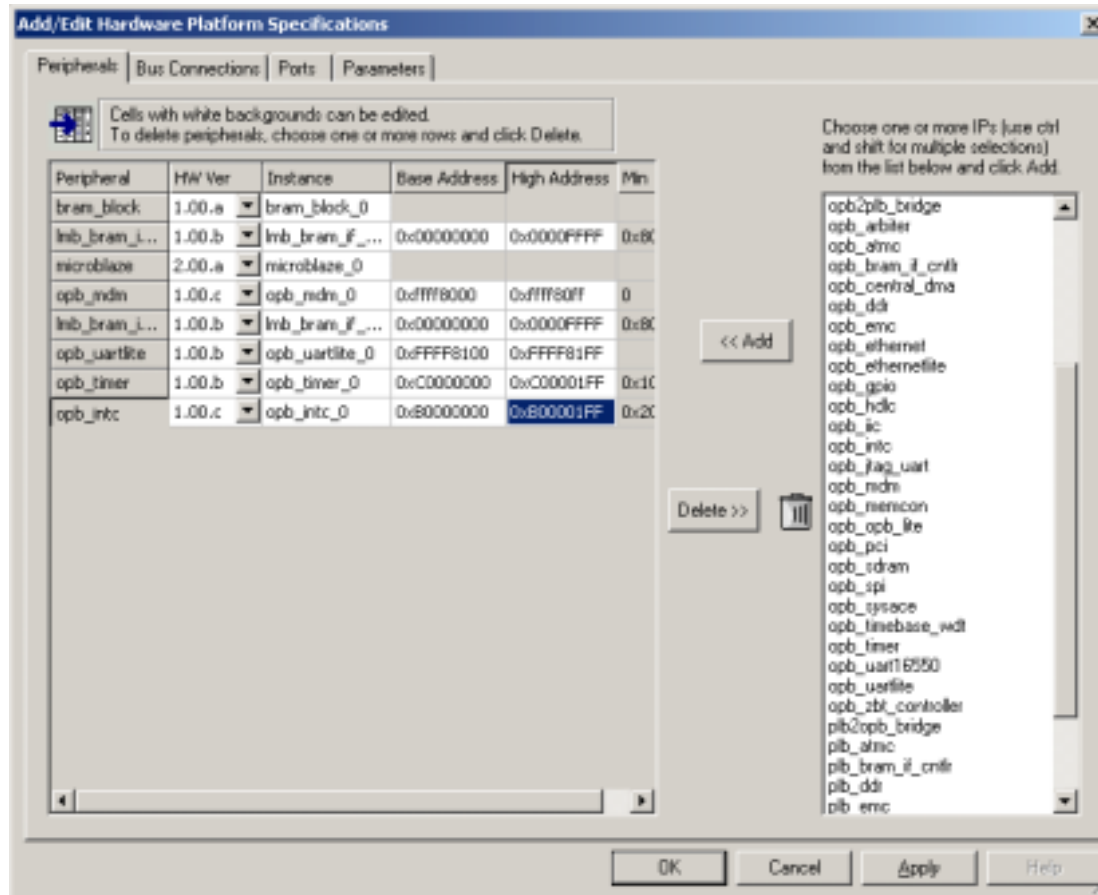
- Select the Parameters tab
- Select the `C_ONE_TIMER_ONLY` parameters and click the Add button
- Edit the parameter value

Put Timer On the OPB



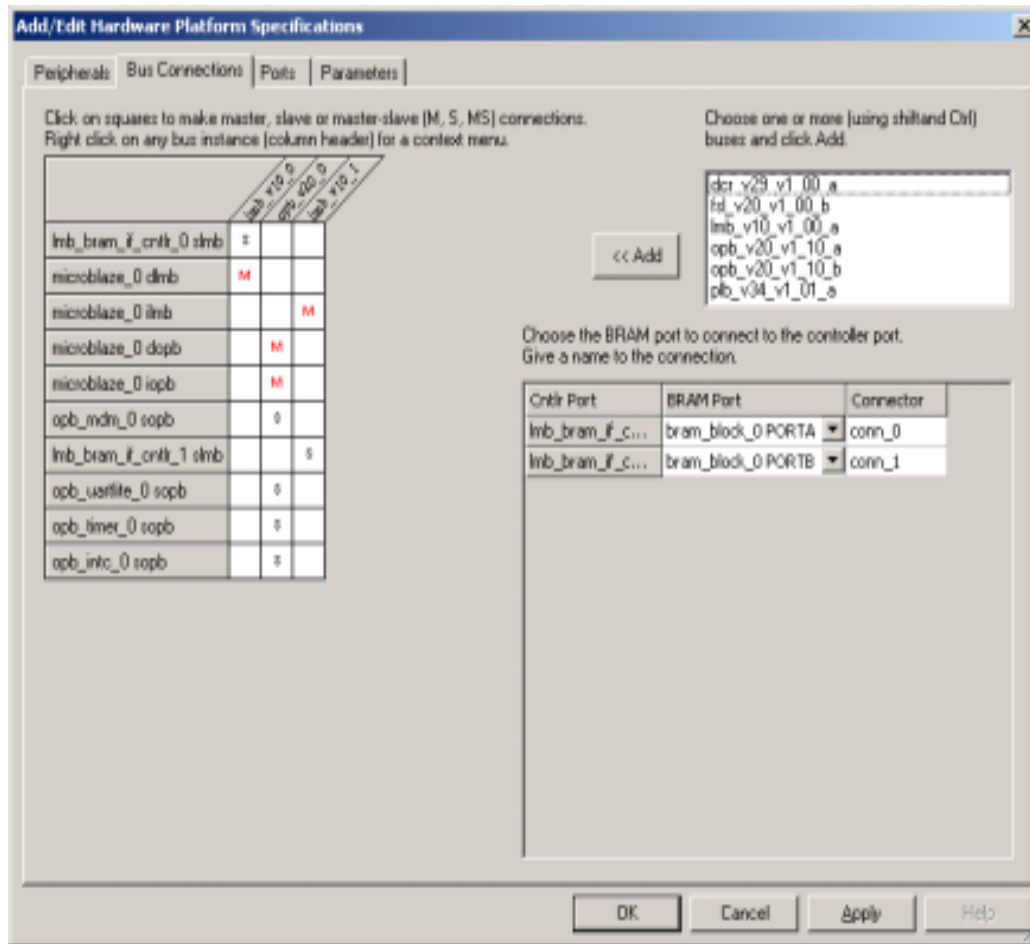
- Select the Bus Connections tab
- Click on the box to mark the Timer as a slave on the OPB bus

Adding an Interrupt Controller



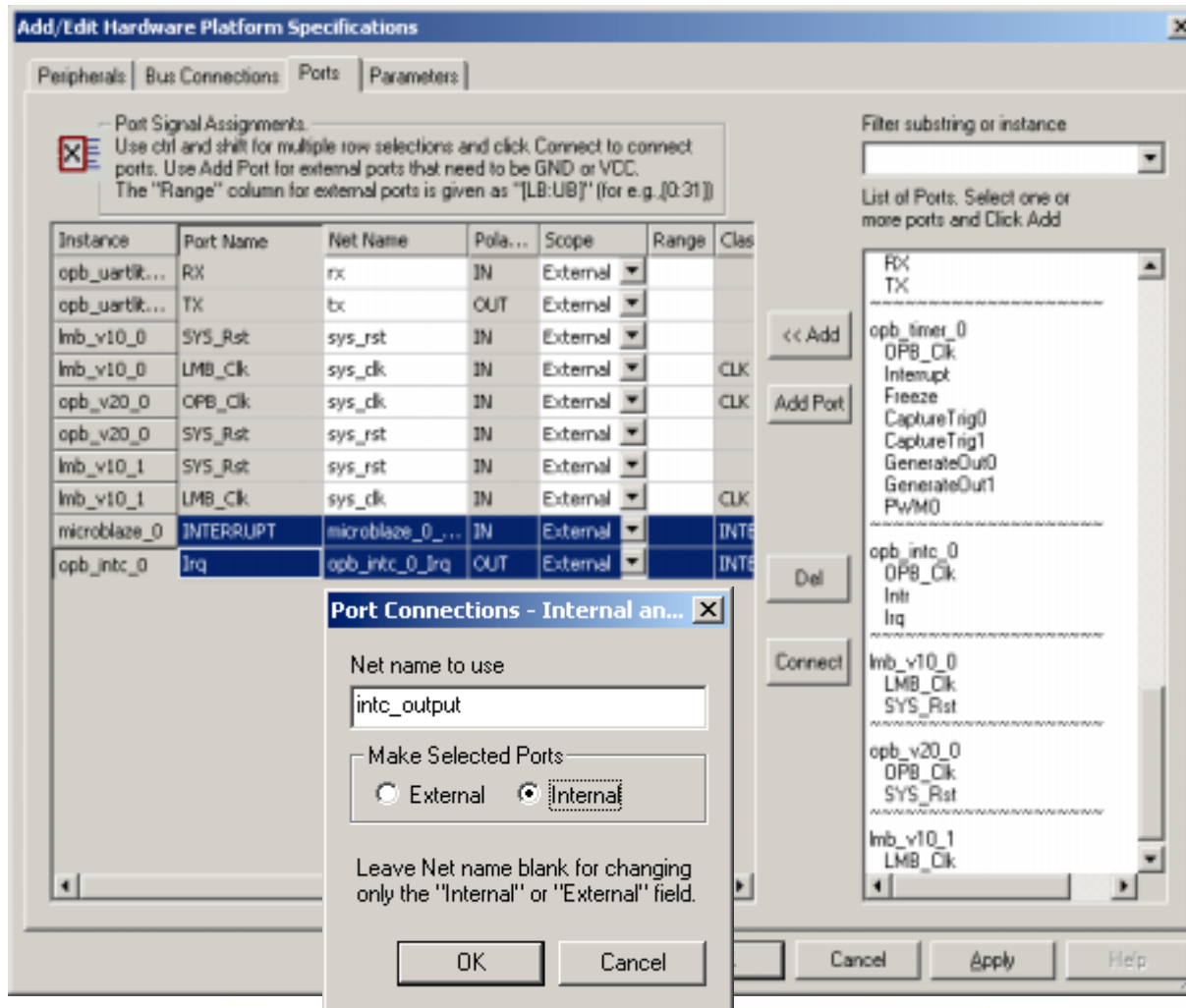
- Select the Project menu
- Select the Add/Edit Cores submenu
- Select the opb_intc and click Add
- Edit the Base Address and High Address of the intc

Put Intc On the OPB



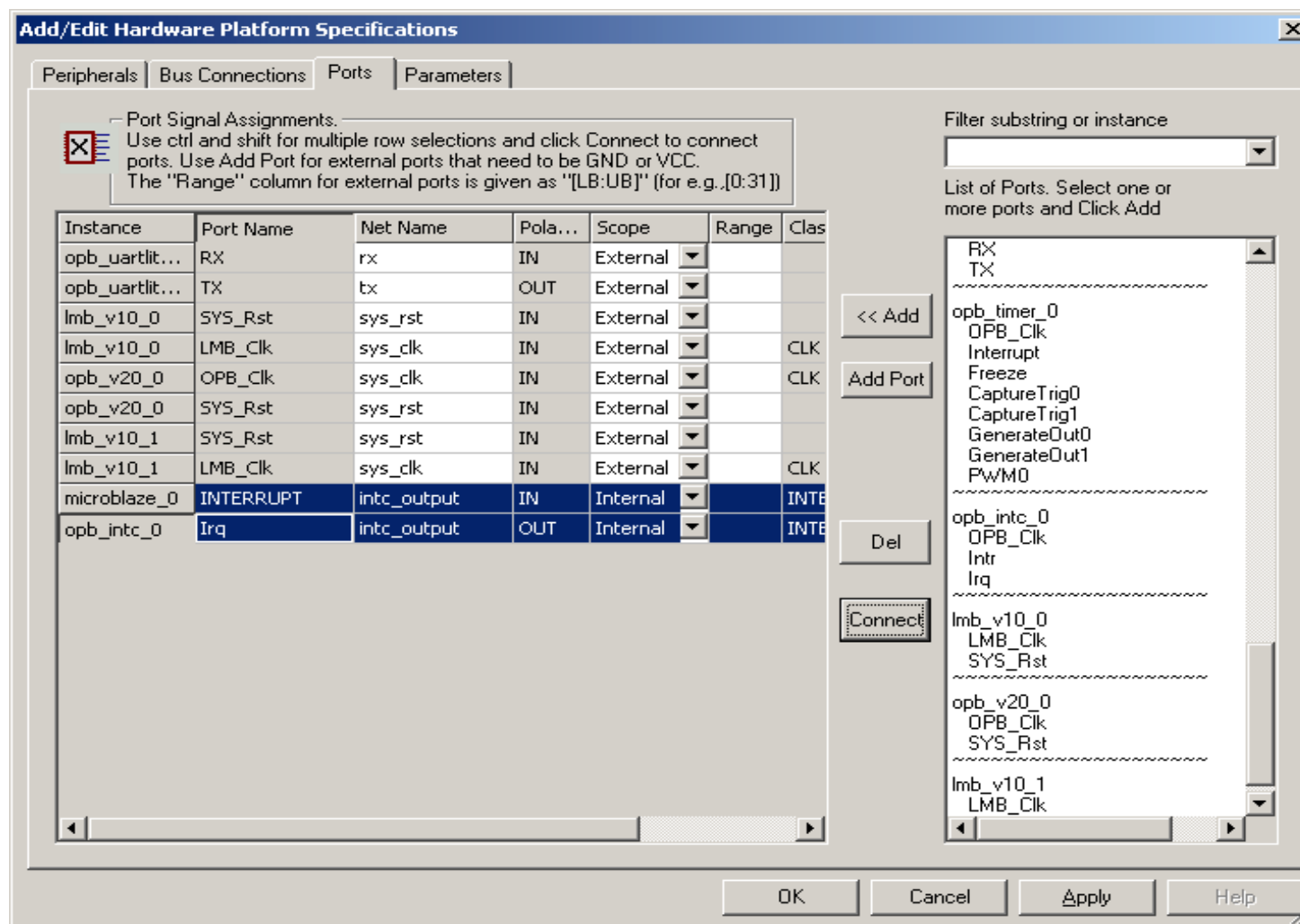
- Select the Bus Connections tab
- Click on the box to mark the Intc as a slave on the OPB bus

Connect Intc to MicroBlaze

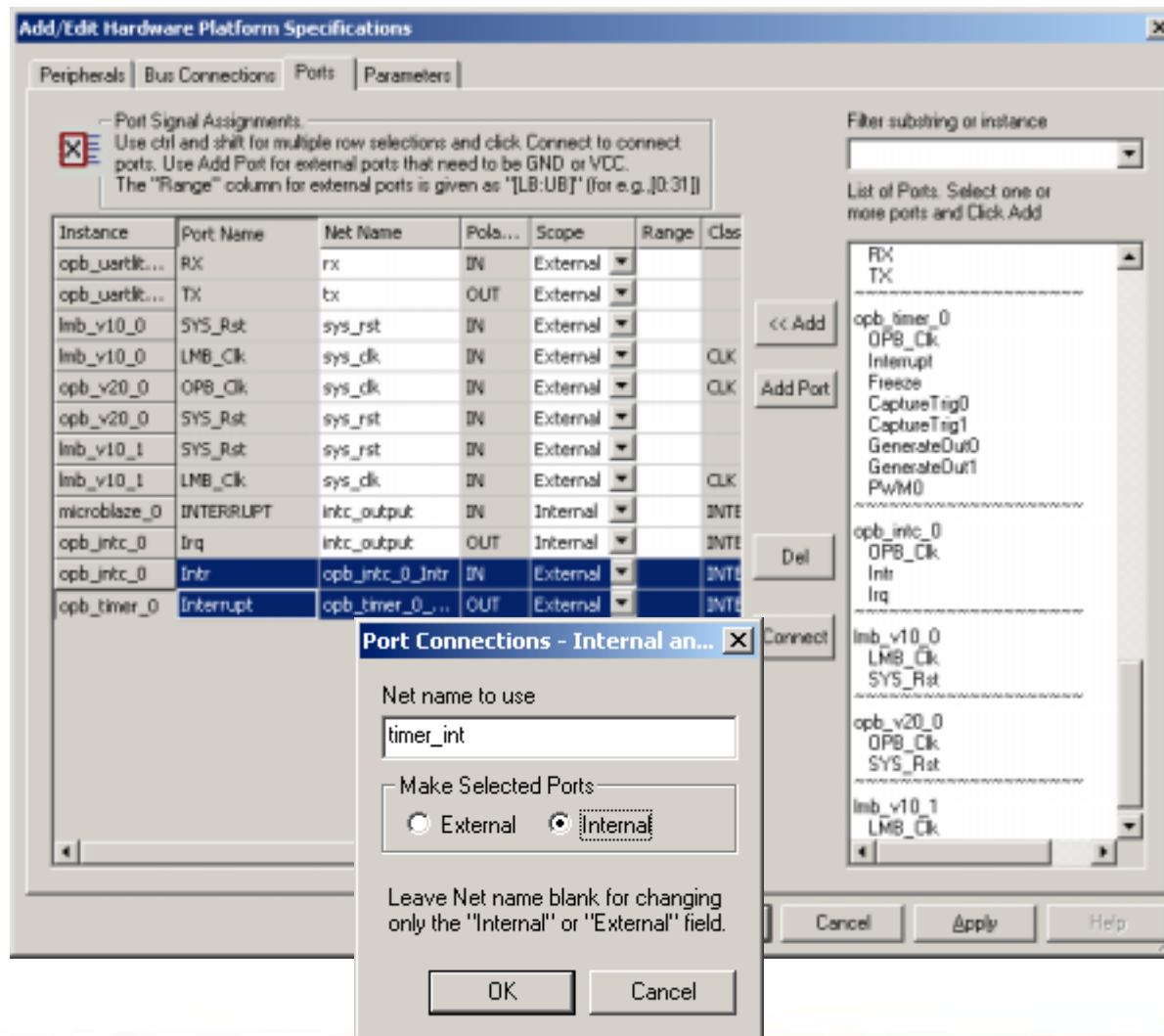


- Select Ports tab
- Add Interrupt input of MicroBlaze and Irq output of Intc
- Highlight both of these and press Connect
- Name the net, select Internal, and press OK

Connect Intc to MicroBlaze

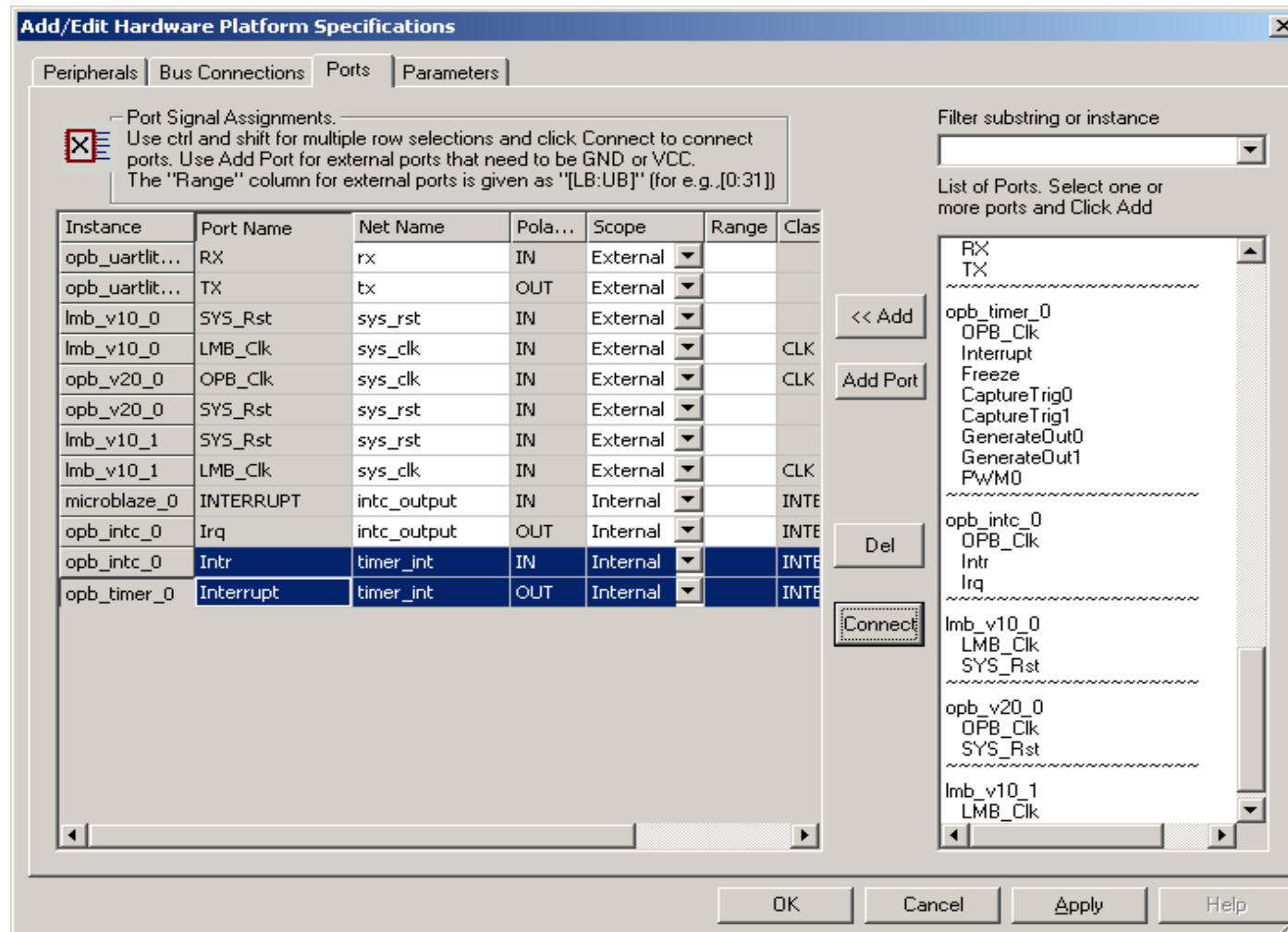


Connect Timer to Intc



- Select Ports tab
- Add Interrupt output of Timer and Intr input of Intc
- Highlight both of these and press Connect
- Name the net, select Internal, and press OK

Connect Timer to Intc



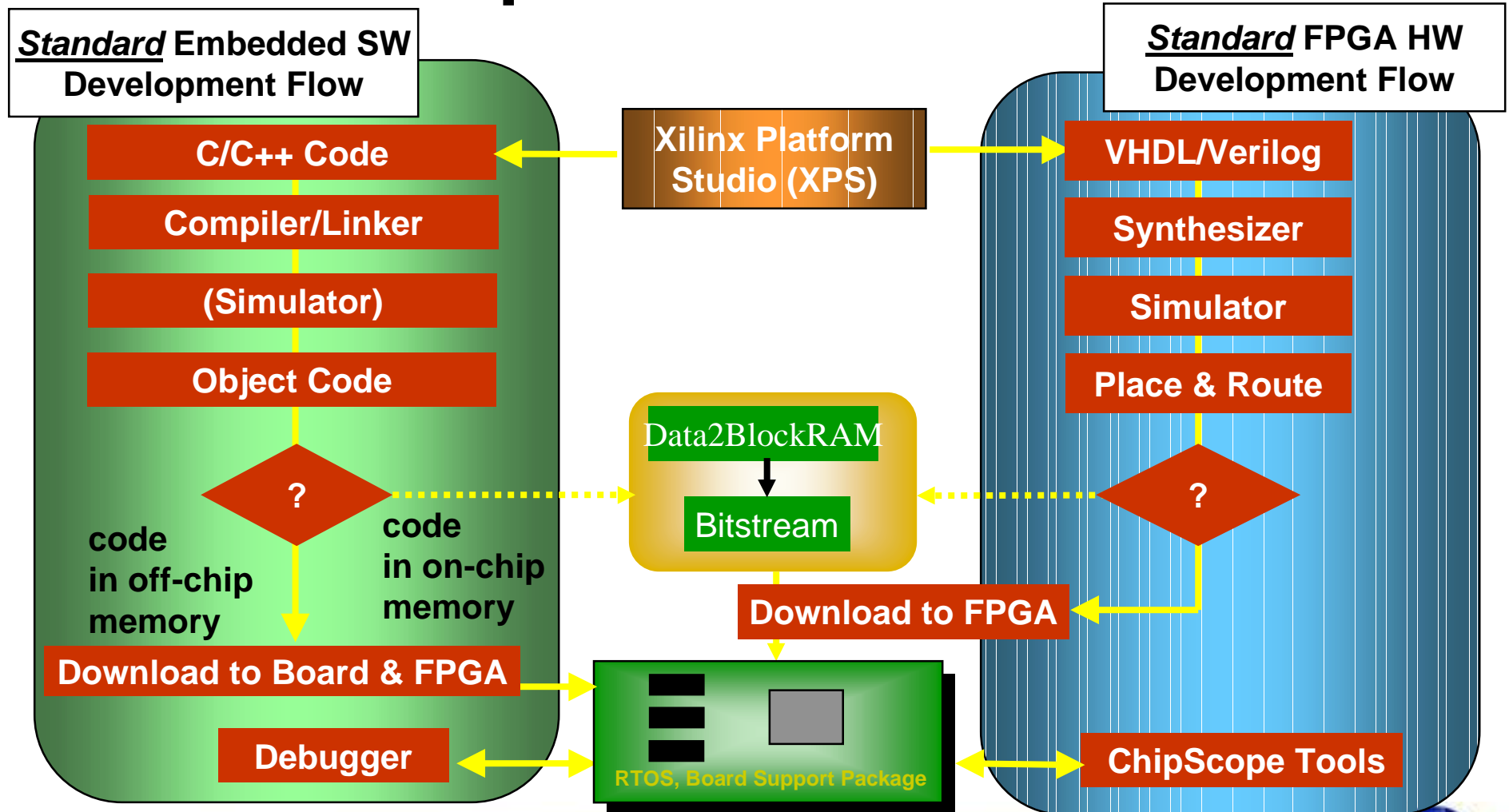


Software Development with the EDK and XPS



EDK System Design

Comprehensive Tool Chain



Building Software in XPS

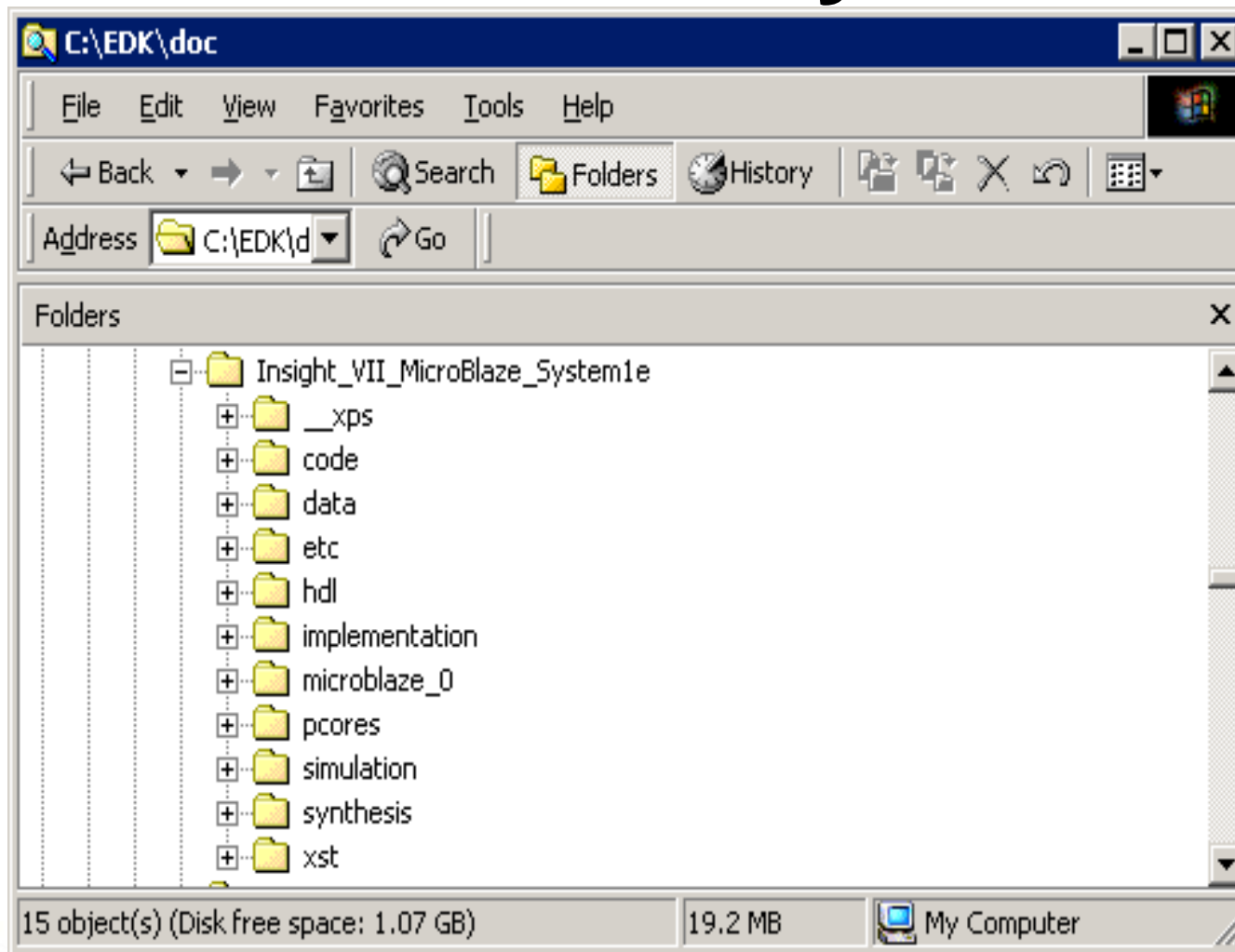
- XPS is an Integrated Development Environment (IDE) similar to other products with the primary difference being it allows the user to build hardware and software.
- The GNU tools (compiler, linker, etc.) including GDB are used by XPS for software development.
- The GNU tools are not native Windows tools such that they execute within a Xygwin (Xilinx Cygwin) environment.



XPS Project Directory Structure

- A project within XPS is a directory that contains multiple subdirectories.
- The *code* subdirectory is created by the user and contains application source code.
- The include files, drivers, and libraries are located in a directory based on the instance name of the microprocessor in the project.

XPS Example Project Directory



Library Generation

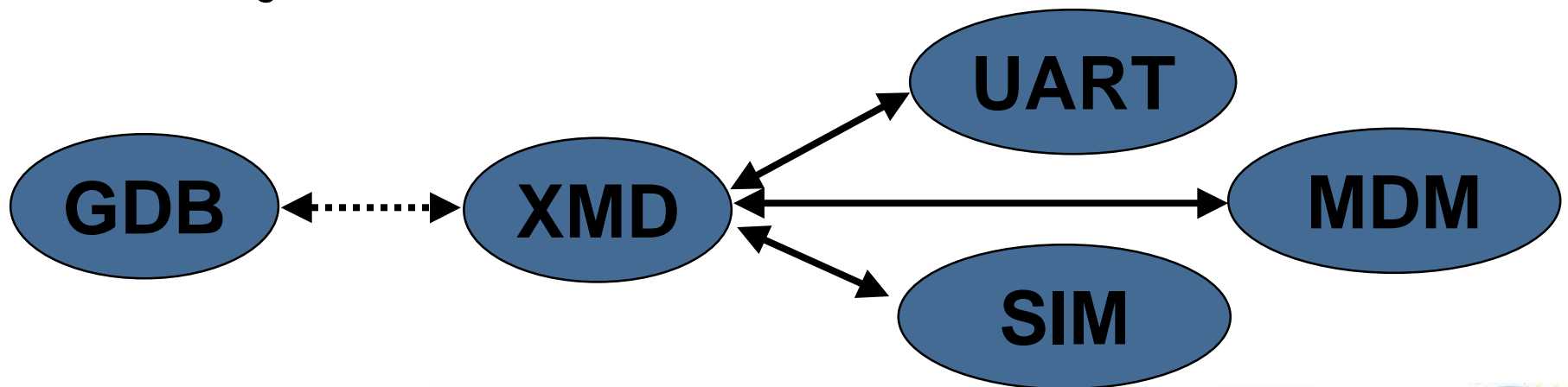
- XPS compiles device drivers and C run-time CRT into a single library that is then linked with a user application program.
- Libgen is the tool executed from within XPS or from a command line, that copies the library source files and device driver source files to the project directory to build the library.

Command Line Builds

- XPS generates a single make file, system.make, that can be used to build the hardware or software from the command line of a Xygwin window.
- This make file could be used to create a build environment for software development groups that build from the command line.

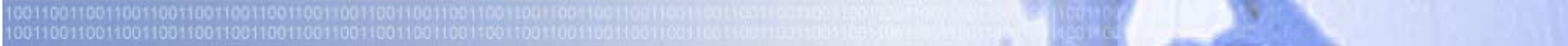
Xilinx Microprocessor Debugger (XMD)

- Interfaces GDB to a "target"
- Supports script interface for built-in commands
- Allows debug with or without a ROM monitor
 - MDM target for true JTAG
 - UART target for ROM monitor (xmd-stub)
 - SIM target for instruction set simulator



Microprocessor Debug Module (MDM)

- JTAG debug using BSCAN
- Software non-intrusive debugging
- Read/Write access to internal registers
- Access to all addressable memory
- Hardware single-stepping
- Hardware breakpoints - configurable (max 16)
- Hardware read/write address/data watchpoints
 - configurable (max 8)



Device Drivers & Software Infrastructure

Device Drivers for FPGA Designs

- Hardware is parameterizable
 - Capabilities and features may change every build
- FPGA space is limited
 - User needs flexible driver architecture
 - Internal memory solution as well as external memory solution
- Processor may change
 - Portability of driver is key

Device Driver Goals

- Portability/Reusability
 - Drivers are to be portable across many different RTOSs, microprocessors, and toolsets
 - Minimize development effort
- Out-of-the-box solution for customers

Driver Design Considerations

Programming Languages

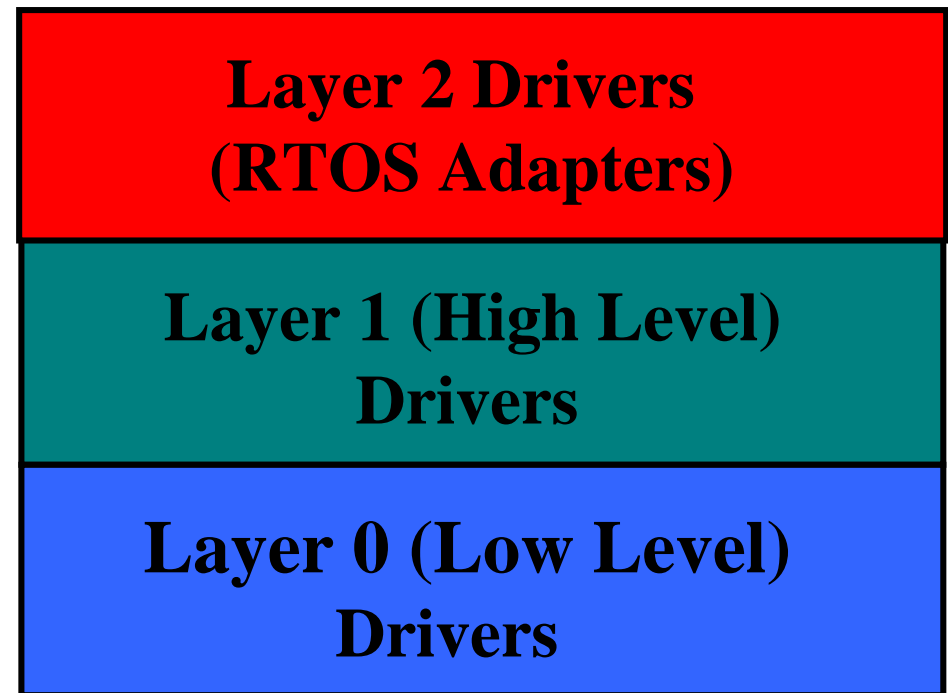
- Assembly Language
 - Minimized to allow maximum portability between microprocessors
 - Only boot code, which executes prior to the C/C++ runtime system, is typically necessary to be assembly language
 - Located in separate source files to help isolate it, as opposed to in-line assembly language in the C source code
- C Programming Language
 - The C programming language is the most utilized language for embedded systems
 - In order to support the largest number of customers, the first implementation utilizes the C programming language

Driver Design Considerations

- Object Oriented Design
 - Emphasize data abstraction, data hiding, and encapsulation in addition to greater potential for code reuse and ease of maintenance
 - Provides an easier transition from non-object oriented languages such as C to more object oriented languages such as C++ and Java
- Delivery Format
 - Delivered to customers in source code format, allowing it to be built and optimized for a wide range of microprocessors using customer selected tools

Device Driver Architecture

- Component based object oriented design implemented in ANSI C
- A device driver supports multiple instances of a device
- Layered device driver architecture to allow user selectable features and size



Device Driver Architecture (continued)

- Source code is provided
- Layer 0 and Layer 1 are OS-independent
- Device drivers in all layers have common characteristics
- Primitive data types for portability (Xuint8, Xuint16, Xuint32, etc.), in xbasic_types.h
- Isolation of I/O accesses for portability (Xlo_In8(), Xlo_Out8(), etc.), in xio.h
- Coding and documentation conventions

Layer 0, Low Level Drivers

- Interface contained in <driver>_l.h file
- Designed for a small system, typically for internal memory of an FPGA.
- Small memory footprint
- No error checking performed
- Supports primary device features only, not comprehensive
- Polled I/O, blocking functions

Layer 1, High Level Drivers

- Interface contained in <driver>.h file
- Designed to allow a developer to utilize all features of a device
- Larger memory footprint
- Robust error checking such as asserting input arguments
- Supports configuration parameters in <driver>_g.c
- Interrupt driven I/O, non-blocking functions
 - Interrupt service routines are provided

Layer 2 Drivers, RTOS Adapters

- Interface contained in <driver>_adapter.h file.
- Converts the Layer 1 device driver interface to an interface that matches the device driver scheme of the RTOS.
- Contains calls specific to the RTOS
- Can use RTOS features such as memory management, threading, inter-task communication, etc.

RTOS Independent Device Drivers

- Driver
 - Responsible for interfacing to the device (peripheral). It encapsulates communication to the device
 - Designed to be portable across processor architectures and operating systems
- Adapter
 - Integrates the driver into an operating system
 - Satisfies the "plug-in" requirements of the operating system
 - Needs to be rewritten for each OS

RTOS Support

- Xilinx supports VxWorks 5.4/5.5 for PowerPC in-house
- 3rd party support includes MontaVista Linux (PPC), ATI Nucleus, uCos, ucLinux
- VxWorks integration:
 - All device drivers can be used directly by the application
 - Some device drivers tightly integrated into VxWorks
 - UARTs to standard and file I/O
 - Ethernet to network stack (Enhanced Network Driver)
 - Interrupt controller
 - System ACE into VxWorks filesystem interface
- Automatic Tornado BSP generation using EDK

Naming Conventions

- A common name is used for all external identifiers of the device driver
 - `<driver_name>_FunctionName();`
 - `<driver_name>_DataType;`
- A common name is used for all source files of the device driver for ease of use
 - `<driver_name>_l.h` low level driver interface definition
 - `<driver_name>.h` high level driver interface definition
 - `<driver_name>.c` primary source file
 - `<driver_name>_g.c` configuration table source file
 - `<driver_name>_intr.c` interrupt processing source file



Multiple Instance Details

- Multiple instances of a single device (such as an Ethernet MAC) typically exist in a system
- A single device driver handles all instances of the device
- A layer 1 device driver uses a data type that is passed as the first argument to each function of the driver. The data type contains information about each device instance such as the base address

Example Layer 0 Device Driver API

- Each function of Layer 0 uses the base address of the device as the first argument
- No state information is kept by the driver and the user must manage multiple instances of the device
- `void XEmac_mSetControlReg(Xuint32 BaseAddress, Xuint32 Mask)`
- `void XEmac_SendFrame(Xuint32 BaseAddress, Xuint8 *FramePtr, int Size)`
- `int XEmac_RecvFrame(Xuint32 BaseAddress, Xuint8 *FramePtr)`



Example Layer 1 Device Driver API

- Each function of Layer 1 uses an instance pointer as the first argument
- XStatus XEmac_Initialize(XEmac *InstancePtr, Xuint16 DeviceId)
- XStatus XEmac_Start(XEmac *InstancePtr)
- XStatus XEmac_Stop(XEmac *InstancePtr)
- void XEmac_Reset(XEmac *InstancePtr)
- XStatus XEmac_SelfTest(XEmac *InstancePtr)

Example Layer 1 Device Driver API For FIFO Interrupt Support

- `XStatus XEmac_FifoSend(XEmac *InstancePtr, Xuint8 *BufPtr, Xuint32 ByteCount);`
- `XStatus XEmac_FifoRecv(XEmac *InstancePtr, Xuint8 *BufPtr, Xuint32 *ByteCountPtr);`
- `void XEmac_SetFifoRecvHandler(XEmac *InstancePtr, void *CallBackRef, XEmac_FifoHandler FuncPtr);`
- `void XEmac_SetFifoSendHandler(XEmac *InstancePtr, void *CallBackRef, XEmac_FifoHandler FuncPtr);`
- `void XEmac_IntrHandlerFifo(void *InstancePtr); /* interrupt handler */`

Device Driver & System Configuration

- `xparameters.h` contains important system parameters used by the drivers & the BSP
- Parameters for each device may include a device ID, a base address, an interrupt identifier, and any device unique parameters
- This file is the best place to start when trying to understand a system
- Libgen automatically generates `xparameters.h`

Example xparameters.h File

- The following example is for a system that has an Ethernet MAC device at address 0x60000000. The name of the hardware instance is opb_ethernet.

```
#define XPAR_XEMAC_NUM_INSTANCES 1
#define XPAR_OPB_ETHERNET_BASEADDR 0x60000000
#define XPAR_OPB_ETHERNET_HIGHADDR 0x60003FFF
#define XPAR_OPB_ETHERNET_DEVICE_ID 0
#define XPAR_OPB_ETHERNET_ERR_COUNT_EXIST 1
#define XPAR_OPB_ETHERNET_DMA_PRESENT 3
#define XPAR_OPB_ETHERNET_MII_EXIST 1
```

Device Driver Configuration Specifics

- Constants describing each device instance are contained in `xparameters.h`
- These constants are also inserted into a configuration table for each device driver contained in the `<driver_name>_g.c`
- The device driver looks up information for the specific instance of the device when it is initialized
- The data type definition for the configuration data is contained in the `<driver_name>.h` file
- Libgen generates the `<driver_name>_g.c` file for each device driver

Device Driver Configuration Example

From
xemac.h
source file

- typedef struct
{
 Xuint16 DeviceId;
 Xuint32 BaseAddress;
 Xboolean HasCounters;
 Xuint8 IpIfDmaConfig;
 Xboolean HasMii;
} XEmac_Config;

From
xemac_g.c
source file

- XEmac_Config XEmac_ConfigTable[] =
{
 {
 XPAR_OPB_ETHERNET_DEVICE_ID,
 XPAR_OPB_ETHERNET_BASEADDR,
 XPAR_OPB_ETHERNET_ERR_COUNT_EXIST,
 XPAR_OPB_ETHERNET_DMA_PRESENT,
 XPAR_OPB_ETHERNET_MII_EXIST
 }



Interrupt Processing

- Layer 1 device drivers provide interrupt driven I/O
- The device driver provides an interrupt handler that must be connected to the interrupt source by the application
- The device driver interrupt handler performs device specific details such as register reads & writes and calls a user specified handler (or callback) to process events and data
- The user application must setup the application callback to be called by the device driver interrupt handler
- The application callback can perform processing in an interrupt context or signal non-interrupt driven processing to perform the processing



Interrupt Processing Example

- When using MicroBlaze, XPS automatically connects the interrupt controller in the system to the exception vector of the processor so that the interrupt controller's interrupt handler gets called when an interrupt occurs
- The following code illustrates setting an application callback for the Ethernet MAC device driver, connecting the Ethernet MAC device driver interrupt handler to the interrupt controller, and enabling the MicroBlaze interrupts

```
XEmac_SetFifoSendHandler(InstancePtr, InstancePtr, SendHandler);  
XEmac_SetFifoRecvHandler(InstancePtr, InstancePtr, RecvHandler);
```

```
XIntc_Connect(&InterruptController,  
             XPAR_OPB_INTC_OPB_ETHERNET_IP2INTC_IRPT_INTR,  
(XInterruptHandler)XEmac_IntrHandlerFifo, InstancePtr);
```

```
microblaze_enable_interrupts();
```

Error Processing

- Device driver functions which detect errors return a data type of XStatus to indicate the detailed error condition
- The error details are contained in xstatus.h
- Device driver functions use asserts to indicate errors during runtime.
- Errors detected during interrupt processing are returned to the application via an asynchronous callback function.

Assert Details

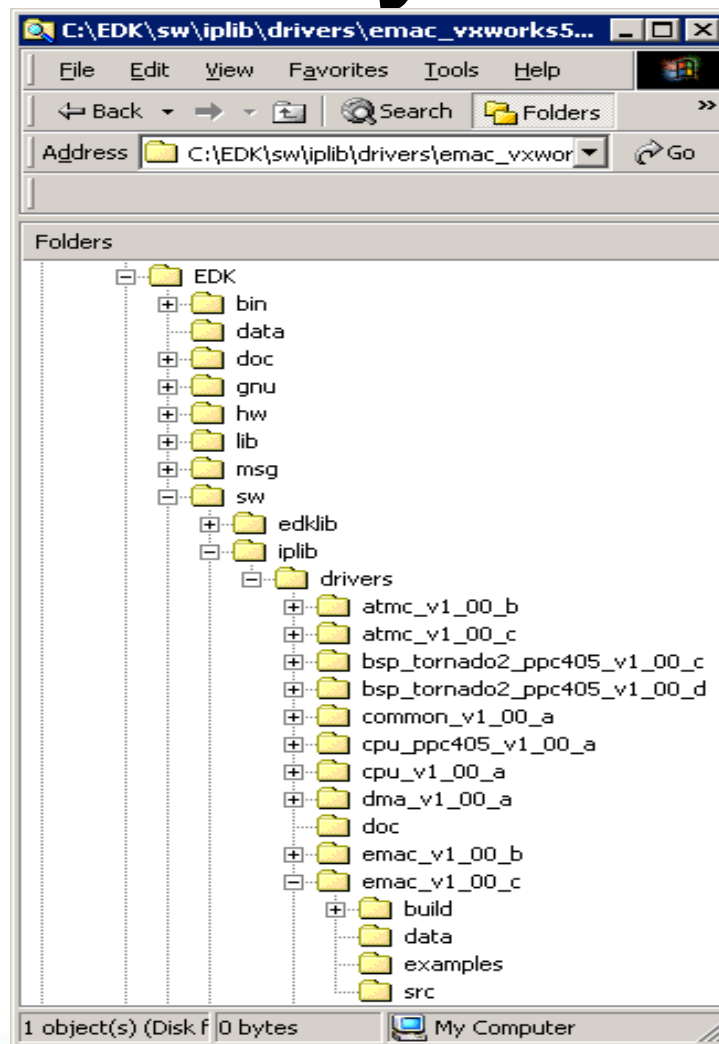
- Device drivers use Assert to validate input arguments
- The default is for asserts to be used by device drivers
- Asserts can be disabled when the libraries are generated by using the -DNDEBUG symbol
- The default behavior of Assert is to loop forever after calling a user defined function if defined
- The user can setup a function to be called when an assert is called.

```
void XAssertSetCallback(XAssertCallback Routine);
```

Device Drivers In The EDK Install Directory

This illustration shows the directories of a device driver (emac) in the EDK install area.

Note that there is an examples directory that contains example source files for using a device driver.



Writing An Application To Use A Device Driver

- Always start with an example provided in the device driver directory of the EDK install to save time
- Choose the example best fits your application, such as polled, interrupts, or DMA, and copy code snippets from the example