

DRAFT – DO NOT REPRODUCE

Microsoft eMIPS

Release v1.1

Winter 2007

Richard Neil Pittman
Alessandro Forin
Microsoft Research

	Version	Revision
8/21/2007	0.0	Pre-release to Xilinx
10/08/2007	1.0	First Release to TAMU
12/21/2007	1.1	Added eBug Debugger, Extension Peripherals, Software doc. Public release.

Table of Contents

Table of Contents	3
Table of Figures.....	6
1 Introduction.....	8
1.1 Motivations	8
2 Tool Setup	12
2.1 Before You Start.....	12
2.2 Hardware Tools.....	12
2.2.1 Xilinx ISE.....	13
2.2.2 Xilinx ISE Partial Reconfiguration Tools	22
2.2.3 Xilinx PlanAhead.....	27
2.3 Software Tools	34
2.3.1 Compiler.....	34
2.3.2 The BBTools Package	40
2.3.3 Giano	43
2.3.4 The RTOS.....	43
3 Building the eMIPS System.....	45
3.1 Common Tasks	45
3.1.1 Changing the Environment to Non-PR Xilinx ISE.....	45
3.1.2 Changing the Environment to PR Xilinx ISE.....	47
3.1.3 Set up the Project Directory	50
3.1.4 Synthesize the Top Level Module.....	53
3.1.5 Add the Register File Blockram	61
3.1.6 Add the Register File Fifo.	67
3.1.7 Add the Bootloader Blockram	74

3.1.8	Synthesize the Base Design (TISA)	79
3.1.9	Synthesize the Reconfigurable Region (Extension)	92
3.2	Building the Configuration Design Files and the Bit Files	105
3.2.1	Building the PR Version with the PR flow	105
3.2.2	Building the NON-PR Version.....	127
3.3	Verifying the Configuration Bit Files	147
3.4	Generating the System ACE Compact Flash Image	166
3.5	Testing	179
4	Software Procedures	197
4.1	Building and running a standalone program	197
4.2	Building and running a program under the RTOS	199
4.3	Debugging with eBug	202
4.4	Rebuilding the boot loader	202
4.5	Building an SE image that contains an Extension	203
4.6	Using the BBTools to profile a program	206
4.7	Adding Extended Instructions to an image	207
4.8	Generating Extensions using the M2V compiler	208
5	Architecture of the eMIPS System	210
5.1	Overview of Xilinx Virtex 4 and ML401 Board	211
5.2	Overview of Xilinx Partial Reconfiguration	212
5.3	Overview of Xilinx System ACE Configuration Solution	214
5.4	Overview of the RISC CPU Organization	215
5.5	eMIPS System Components	216
5.5.1	Top Level Module	216
5.5.2	Clocking and IO Modules.....	217
5.5.3	Trusted ISA (TISA), Static Design Region.....	218

5.5.4	Memory Subsystem Modules	222
5.5.5	On-Chip Peripherals	223
5.5.6	Extensions, Reconfigurable Design Region	225
6	The eMIPS Extension Interface	226
6.1	Extension Interface Control.....	226
6.2	HDL Coding of Interfaces.....	228
6.3	Interface Protocols	228
6.3.1	Instruction Decode/Arbitration Protocol	228
6.3.2	Register File Interface.....	231
6.3.3	Memory Interface	233
6.3.4	Program Counter/Instruction Fetch.....	235
6.3.5	Pipeline Re-Entry	236
6.4	Design Floor planning	239
7	References	242
	Appendix A: Example eMIPS Extensions.....	247
	mmldiv64.....	247
	loadreturn	248
	timer	249
	uart	249
	eBUG HW 2	250
	eBUG HW 8	250
	Appendix B: Build Scripts	251
	eMIPS_PR_Top.bat.....	251
	eMIPS_PR_TISA.bat	251
	eMIPS_PR_Extension0_mmldiv64.bat.....	251
	eMIPS_PR_Extension0_mmldiv64_merge.bat.....	251

Table of Figures

Figure 1: SE File Format.....	204
Figure 2: Block diagram of the eMIPS architecture.....	210
Figure 3: Xilinx ML401 Evaluation Board with Virtex 4 LX25[20].....	212
Figure 4: Examples of partitioning of a Reconfigurable FPGA design. [5][14]	212
Figure 5: Logical connections of signals crossing a region boundary.[5]	213
Figure 6: LUT Based Bus Macro.[5]	213
Figure 7: System ACE File structure.[12]	215
Figure 8: Design Hierarchy of the Top Level Module.....	217
Figure 9: Design Hierarchy of the Trusted ISA	218
Figure 10: eMIPS Memory Bus	223
Figure 11: Extension Control Register (CP0 register 16)	226
Figure 12: Per Extension Slice of the Extension Control Register.....	226
Figure 13: Instruction Decode/Arbitration Protocol.....	229
Figure 14: Resuming the Pipeline	230
Figure 15: Instruction Decode/Arbitration Protocol for Passive/Parallel Operation	231
Figure 16: Register Read Interface.....	232
Figure 17: Write Register Interface	233
Figure 18: Memory Read Operation	234
Figure 19: Memory Write Operation	235
Figure 20: Extension PC update	236
Figure 21: Reenter TISA Pipeline at Memory Access.....	237
Figure 22: Reenter TISA Pipeline at Writeback	238
Figure 23: eMIPS Floor Plan.....	239

Figure 24: Close up of eMIPS Floor Plan with Bus Macros	240
Figure 25: mmldiv64 shift 128-bit left logical	248

1 Introduction

This document is part of the current release of the eMIPS system, and the primary source of documentation for it. Its target is a researcher or developer that intends to use the system in a practical setting. It describes how to install and use the hardware and software tools that are required for developing and extending the system. It contains information about the architecture and implementation of the system that is required when developing processor Extensions. Other documents are available online and can be found at the eMIPS web page at <http://research.microsoft.com/research/EmbeddedSystems/eMIPS/emips.aspx>.

The document is structured as follows. The rest of this section presents the motivations for the eMIPS research project and its specific contributions to the state of the art in the field. Section 2 describes how to procure and install the required hardware and software tools and packages. Section 3 describes how to build and test a new processor image. Section 4 describes how to create, debug and optimize software applications for the eMIPS processor, to run either standalone or under the provided RTOS. Section 5 describes the modules and interfaces that constitute the implementation of eMIPS, as provided in the release. Section 6 describes in more details the interface between the fixed portion of the processor and the dynamically changeable parts. Section 7 references the research that is most closely related to eMIPS. Appendix A describes the example Extensions that are provided in the release to help developers get started.

1.1 Motivations

Most of the modern microprocessors implement the Reduced Instruction Set Computer architecture, or RISC, which is based on fixed instruction sets. Many different types of RISC microprocessors populate the market based on these different sets of instructions including, MIPS, ARM and PowerPC to name some of the more popular. These microprocessors are realized in the form of application specific integrated circuits, or ASIC, made up of logic fixed at design time that cannot be altered after the chip fabrication process is complete. When designing instruction sets, computer engineers attempt to capture all the instructions necessary to cover the largest space of potential applications, while keeping in mind factors such as size, cost and power. This set of instructions forms the blue print for the instruction set architecture, or ISA, to be implemented on the new microprocessor.

Despite all efforts, the quest for the ‘optimal fixed instruction set architecture’ is an impossible one because the space of applications to which the designers apply general purpose processors evolves constantly. In addition, the trends that govern this evolution shift periodically in response to changes in consumer lifestyles and demands. For example, the need to process more audio and video data has led to extensions for all of the above RISC architectures. Therefore, the selection of instructions for a ‘general purpose’ microprocessor designed to meet the demands of today’s market place may be ill equipped to handle the applications of future markets. We can also argue that the quest for the ‘optimal general purpose’ microprocessor for today’s market does not make sense anymore, especially in the embedded market place.

In the embedded market, system designers work within the strictest constraints of size, cost and power. The systems they design apply to a specialized and significantly reduced space of potential applications. In this context, a ‘general purpose’ microprocessor is

inefficient and under-utilized when the majority of applications never use a large subset of the capabilities it provides. For instance, many embedded systems rarely if ever use floating point operations but most ‘general purpose’ microprocessors include them. For these applications, a popular solution is to use custom microprocessors with reduced instruction sets but with customized instructions added specifically for the intended application space. This requires redesigning the microprocessor architecture and fabricating custom microprocessors for each of the desired application domains; still they suffer from the inflexibility previously discussed. Therefore, manufacturers cannot offset the cost in design and fabrication of the new custom chip if the market for the given application is not large enough. As demonstrated in this discussion, the problem of this inflexible ‘general purpose’ microprocessor architecture becomes a severe hindrance. What is needed in the embedded application space and potentially in all areas of microprocessor hardware design is a new technology that can provide for flexibility and customization, allowing the microprocessors to evolve with their target markets at all stages of their life cycle.

The Field Programmable Gate Array, or FPGA, is a digital semiconductor device often used for prototyping. Developers use FPGAs in prototyping for the ability to configure or program their electrical interconnects to realize an expansive space of applications from glue logic to application coprocessors. As the name indicates, developers are free to apply modifications to the design implemented on the FPGA in the field, after deployment. Developers synthesize the configurations from a hardware description language, like Verilog or VHDL, for the targeted FPGA device. Eventually, the configuration file downloads to the chip through an interface such as JTAG. This flexibility comes with a price. The configurable logic of the FPGA experiences significantly lower performance than the modern ASIC. FPGAs, currently are clocking at frequencies barely higher than 500 MHz, while ASICs are currently clocking at frequencies over 3 GHz. Despite this limitation, FPGA technology has evolved to the point where developers can implement fixed logic microprocessors with performance levels competitive with their ASIC counterparts, especially in the embedded market. These future microprocessors will have the advantage that they can be dynamically updated after deployment to meet new demands. This approach to microprocessor design leads to a new class of microprocessors termed the “dynamically extensible processor”.

Using modern FPGAs it is possible to partition the FPGA into multiple sections that can reconfigure independently of each other. We can implement the dynamically extensible processor using a section containing a standard fixed logic processor core with interconnects to other sections, termed ‘Extensions’ that contain customized instructions and functionality that loads, modifies and enables while the fixed logic continues to function without interruption. In this way, the dynamically extensible processor, using a library of Extensions from which it can draw, adapts to the changing application needs in the field. By using these dynamically extensible processor cores, a “reconfigurable central processing unit” becomes possible. The eMIPS project described in this document argues that such a device is feasible today and proves this thesis by means of an example prototype implementation.

The standard RISC architecture lacks the infrastructure to allow for the kind of flexibility and extensibility possible through the use of FPGAs. This new extensible instruction set computer architecture provides this infrastructure. The FPGA is partitioned into fixed and reconfigurable regions. The fixed logic region constitutes the base functionality of the processor including security sensitive resources such as the system coprocessor and the systems used by the microprocessor to control its configuration. The Extensions to the base

processor make up the reconfigurable region of the FPGA. Alternatively, the fixed logic region can be implemented using ASIC technology and only the reconfigurable region as an FPGA or CPLD, with the added benefits of extra security, speed and reduced area. The eMIPS architecture therefore provides the flexibility and adaptability lacking in the RISC architecture.

Extensions can take the form of new instructions developed to meet the changing computing needs of the market. The Extensions can be thought of similarly to firmware updates in other devices that load when applications requiring those updated instructions are loaded or of added optional features such as floating point operations. In addition, Extensions can implement optimized instructions that can take the place of blocks of code with the same semantics. In the case of the optimized instructions, these instructions are added to the software binaries immediately before the blocks they replace. If the 'Extension' associated with that instruction has been configured to one of the available extension slots in the reconfigurable region of the FPGA, the Extension executes the optimized instruction and the block it replaced is skipped. Otherwise, the data path contained within the fixed region of the FPGA interprets the optimized instruction as a NOP and the original block that is still in the software binary executes normally. The Extension's custom logic implements the block's functionality more efficiently than a sequence of instructions that reuses the same execution units to perform the block's function step by step. For this reason, the Extension executing the same function as the block completes the operation faster. If the block constitutes a large enough percentage of the execution count of the application software the overall performance on this microprocessor is significantly improved. The inclusion of these new instructions requires minimal changes to software binaries, as little as adding the instructions immediately before the blocks they replace.

The eMIPS architecture also addresses the waste associated with including functionality in systems where they are never used. The Extensions of the eMIPS architecture do not load when the microprocessor powers up. The fixed logic system only includes the minimum functionality (system management, reconfiguration support, load, store, arithmetic and logical functions), the Extensions provide any additional functions required by the applications running on the microprocessor. For instance, the floating point co-processor or the media or vector co-processors can be loaded only if and when software applications use them. When applications do not require these functions the Extensions are not loaded, providing for potentially large power savings because the unused Extension slots may have their clocking resources and power disabled to reduce the power consumption. Area savings are also possible because not all Extensions need to be present at all times, as is the case instead for an ASIC implementation. This waste reduction may increase by dynamically loading and unloading not only the co-processors but also the on-chip peripherals that are part of an embedded microcomputer. Rather than including all possible peripherals in the ASIC we can load them on an extensible processor as Extensions, again with power and area reductions.

Through the use of FPGA configuration technology, the eMIPS architecture addresses the inflexibility, performance growth and waste of the modern RISC architecture. We have realized the architecture in a working prototype, exposing the five stages of the base pipeline to the dynamically loaded Extensions. We have implemented the dynamic loading of Extensions leveraging the Partial Reconfiguration tools and processes provided with the manufacturer's synthesis tools. The flexibility and performance benefits of the architecture have been demonstrated by patching the binaries of a number of software applications and measuring the resulting speedups. We used several software systems, ranging from an object

oriented real-time operating system for embedded applications, video games, and the SPEC2000 benchmarks. We find that even very simple Extensions can easily achieve speedups factors of 2x-3x over the original application binaries, using a very small number of Extended Instructions per application.

The eMIPS system makes the following specific contributions to the state of the art. eMIPS is the first realized workstation based entirely on a dynamically extensible processor that is safe for general purpose, multi-user applications. By exposing the individual stages of the data path, eMIPS allows optimizations not previously possible. This includes permitting safe and coherent accesses to memory from within an Extension, optimizing multi-branched blocks, and throwing precise and restartable exceptions from within an Extension.

2 Tool Setup

Development for the eMIPS microprocessor system requires both hardware and software tools due to its configuration capabilities. All the tools mentioned in this section are available for download or purchase from their respective vendors. For hardware extension development, the Xilinx ISE is used to synthesize and build the design for a target Xilinx FPGA. This use of Xilinx FPGA tools and FPGAs is in no way an endorsement of these product lines. The selection has been made based on the availability of the partial reconfiguration feature in Xilinx FPGAs, which is required for the dynamic extension of the eMIPS microprocessor system. Before you attempt to use the eMIPS microprocessor system it is recommended that you familiarize yourself with the Partial Reconfiguration design flow available for Xilinx FPGAs. In addition to the ISE, Xilinx PlanAhead is recommended for floor planning but not required. Creating software to run on the eMIPS is the combination of a GCC cross compiler for the MIPS instruction set, and of a set of additional tools specifically developed for the eMIPS microprocessor and included in this release. Microsoft Giano is used for identifying candidate basic blocks for hardware acceleration, and to estimate the theoretical application performance improvements of potential extensions.

2.1 Before You Start

We will assume in the following that your development machine meets a few basic requirements. You should have plenty of disk space available as some of the tools will require quite a bit of it. If you install all of the tools you might need about 15GB, which is approximately 6 GB per install of the Xilinx ISE tools plus enough space for the other tools and project files. We recommend that you use a state-of-the-art fast processor, because many of the tasks are CPU-intensive. We recommend that you have at least 2 GB of fast RAM and preferably more; you do not want the Xilinx place-and-route tools to start paging to disk.

The instructions assume that you will use the Windows XP operating system on your development machine, preferably with the latest service pack installed. We have successfully used both the Pro and the Server versions of Windows XP. Both the 32 bit and the 64 bit versions worked for us. Previous versions of Windows (e.g. Win2k) might work but are untested. The Xilinx tools are known not to work on Windows Vista, and we therefore discourage you from trying to use it. Non-Windows operating systems might work, but are not supported by the tool binaries provided in this distribution.

The eMIPS system as distributed in this release runs on the Xilinx ML401 development board. You should be able to obtain one from your Xilinx representative or directly from the Xilinx website (<http://www.xilinx.com/onlinestore/index.htm>). We assume that your board is fully tested and operational.

2.2 Hardware Tools

The Xilinx ISE is the integrated development environment for the Xilinx FPGAs and configurable logic products. At the time of this publication, Xilinx had release version 9.2i of the ISE however, this version does not support partial reconfiguration. Support for this feature is expected to return in future updates to version 9.2i. The most recent version that supports partial reconfiguration is version 8.2i. For this reason it is recommend that all eMIPS users select version 8.2i until Xilinx release a new, partial reconfiguration compatible version of the

tools. Configuration files for the base system and some example extensions are provided in the release and can be used as-is, without need for the ISE tools. However, to design, build and use your own extensions with the eMIPS system you will require access to these tools.

A fully floor planned constraint file is provided in the release, but in some cases your design needs might require that you change the initial floor planning. The constraint file can be edited using a text editor to change the area and LOC constraints of the reconfigurable regions, IO and clock resources. Due to the rigid constraints imposed by the Partial Reconfiguration design flow, it is recommended using Xilinx PlanAhead over text editing of the constraint file. Xilinx PlanAhead also provides additional features such as DRC checking of your floor plans and area estimation of the reconfigurable regions.

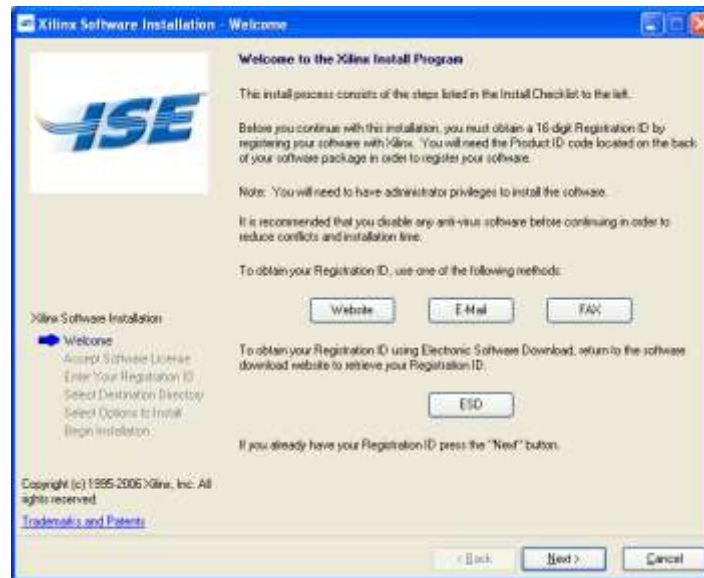
Development of designs using the eMIPS system architecture requires that you have at least one and preferably two installations of the Xilinx ISE installed on your workstation. The first install is a standard install of the Xilinx ISE 8.2i with the latest service packs and updates. This install is used for initial system development and debugging. The second install is the Xilinx ISE 8.2i, upgraded with service pack 1 and with the Partial Reconfiguration Tool overlay installed. This second install is used to build any partial reconfiguration design, both for the base processor and for the extensions. It is recommended that you first create non partial reconfiguration versions of your designs, using the standard install, to explore and debug your design. While the eMIPS project stresses the dynamic configurability feature, it is certainly possible to stop here and to build non partial reconfigurable versions of the eMIPS system, e.g. with fixed extensions included. Do not attempt to build non partial reconfiguration designs with the install with the PR overlay applied. The install with the overlay has been specially setup for partial reconfiguration and non partial reconfiguration design built with it cannot be guaranteed to be correct.

In order for users to build a full eMIPS system including the partial reconfiguration capability, they will require the Xilinx ISE 8.2i with the PR tools. The Xilinx ISE 8.2i with the PR tools is also required to modify the eMIPS system. In order to modify the floor plan of the eMIPS system, you modify the constraint file by hand in the Xilinx ISE text editor; however this can be difficult and prone to error. For modifying the eMIPS floor plan it is recommended that users install the Xilinx PlanAhead tool. For users that are not interested in the partial reconfiguration feature and instead prefer a fully static implementation, the latest version of the Xilinx ISE tools and service pack are the recommend choice. The latest version used in our group is Xilinx ISE 8.2i service pack 3.

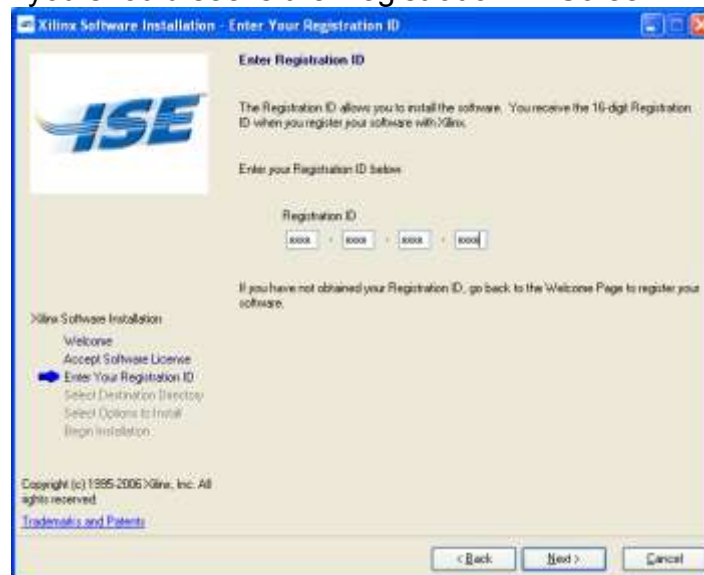
2.2.1 Xilinx ISE

The following instructions are for installing the Xilinx ISE 8.2i from the full version DVD. If you use a different means of installing the tools (evaluation download, DVD, CD, server, etc) your experience may differ. This procedure is for a standard version of the Xilinx ISE tools without modification. This toolset is recommended for development of static designs that do not utilize the partial reconfiguration feature. With this toolset it is possible to implement and build eMIPS configurations that include static Extensions.

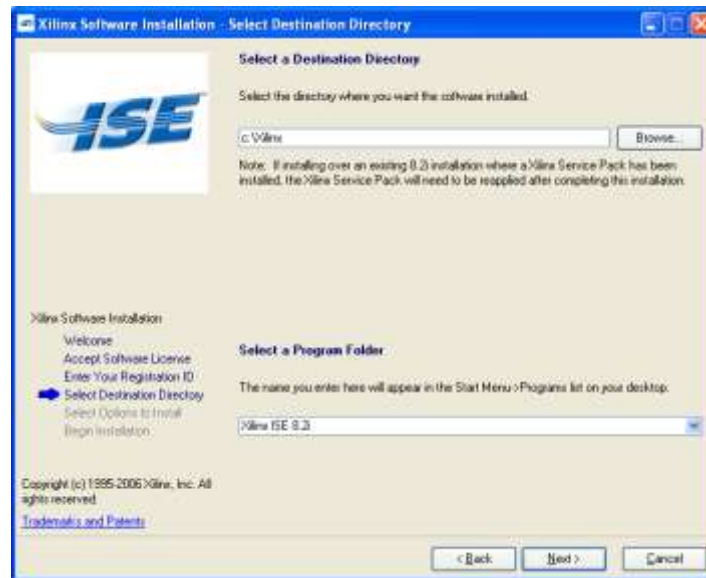
1. Insert the Xilinx ISE 8.2i DVD into the DVD drive tray
2. In a moment the following welcome screen should appear.



3. Click 'Next'.
4. The next screens you should see are the License Agreement screens.
5. Please read carefully the License Agreements. If you feel the terms of the ISE License Agreement are acceptable to you and your organization, check the checkbox labeled "I accept the terms of this software license" and click 'Next' for each.
6. The next screen you should see is the Registration ID Screen.



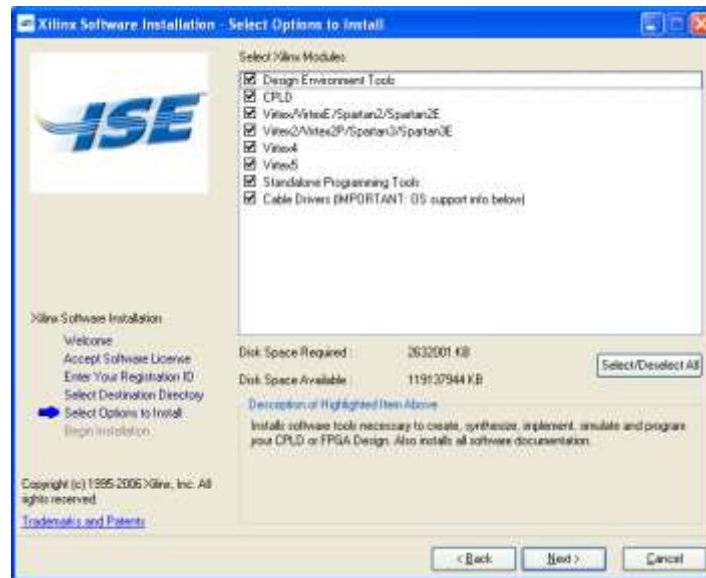
7. Please enter your Registration ID in the fields provided. If you do not have a Registration ID, you will need to obtain one from Xilinx before you can proceed.
8. When you are done entering the ID, please click 'Next'.
9. The next screen you should see is the destination directory screen.



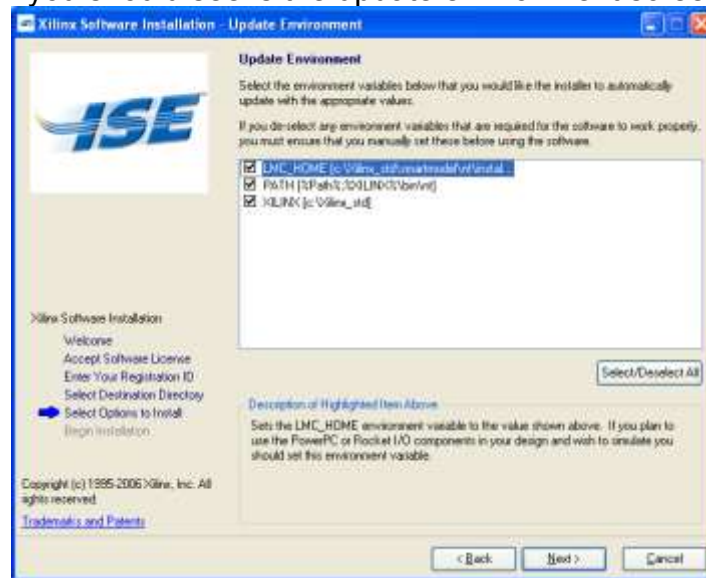
10. Since you will be installing multiple installs of the Xilinx ISE for running both the standard and partial reconfiguration design flows, it is recommended that you use a unique directory name for each install. For this example we will be using “c:\Xilinx_std”. Your selection may vary.



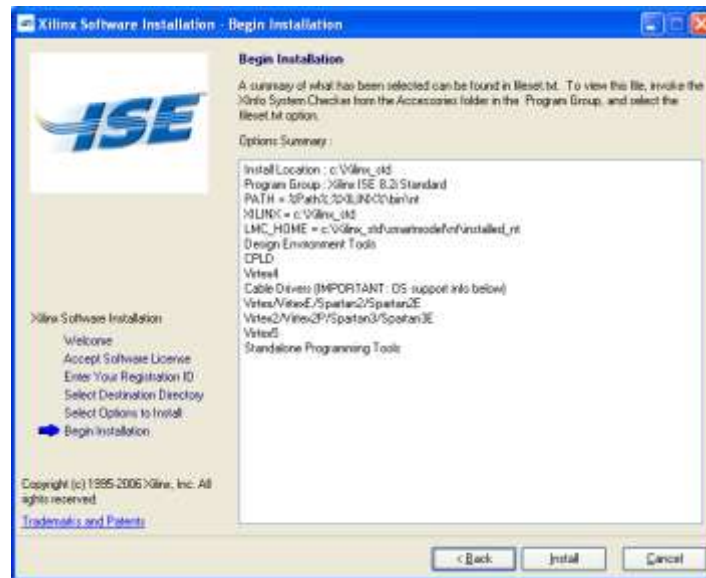
11. After you have selected the destination directory for your Xilinx ISE install, please click 'Next'.
12. The next screen you should see is the installation options screen.



13. It is recommended that you leave all the items in the list checked. After you reviewed the options please click 'Next'.
14. The next screen you should see is the update environment screen.



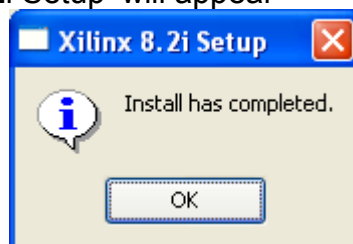
15. This screen will help you setup the environment setting you require for running the Xilinx ISE tools. It is recommended that you leave all of the items checked. After you have reviewed the environment settings please click 'Next'.
16. The next screen you should see is the begin installation screen



17. You should see a listing of all the options you selected in the previous screens. Please review these options to ensure they are correct. After you have reviewed the options listed please click 'Install'. If something is in error click 'Back' to return to the appropriate screen and correct the option.
18. The next screen you should see is the installation progress screen. This may take some time depending on your system, between 30 to 90 minutes.

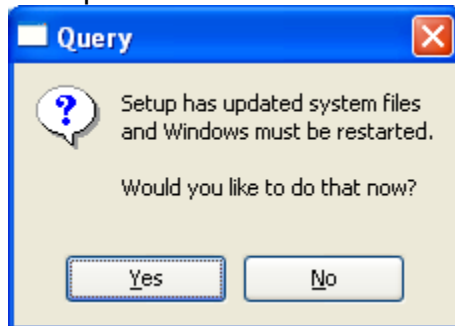


19. Wait for the installation to complete. Advertisements may vary.
20. A window labeled 'set up' will pop up momentarily.
21. A window labeled 'Xilinx 8.2i Setup' will appear



22. Please click 'OK' and the window will close.

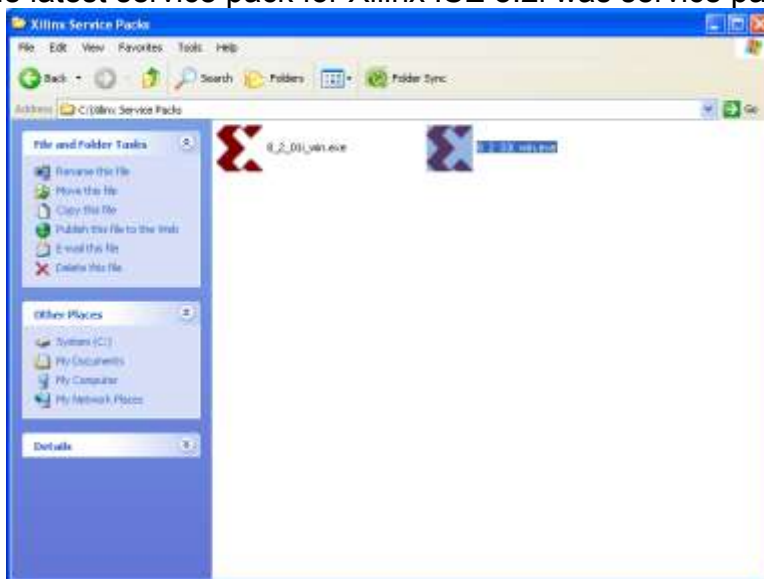
23. If you selected to install the download cable driver, a window labeled 'Query' should appear. Otherwise skip to step 26.



24. If you are ready to update your system please click 'Yes'. Otherwise remember to restart your system before using the Xilinx ISE.

25. Restart your system.

26. Obtain the latest service pack available for Xilinx ISE 8.2i from the Xilinx Download Website (http://www.xilinx.com/xlnx/xil_sw_updates_home.jsp). At the time of this publication the latest service pack for Xilinx ISE 8.2i was service pack 3.

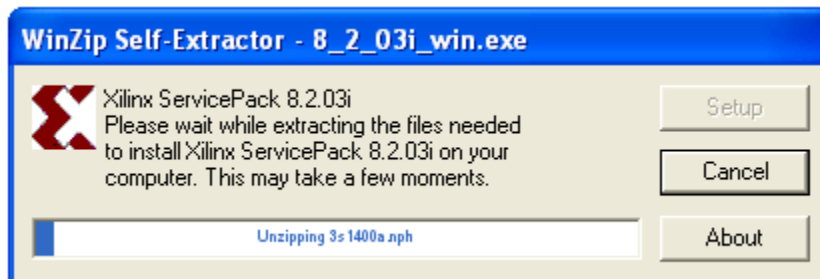


27. Double click on the Service Pack installation file. (8_2_03i_win.exe for Xilinx ISE 8.2i Service Pack 3) Your filenames may vary.

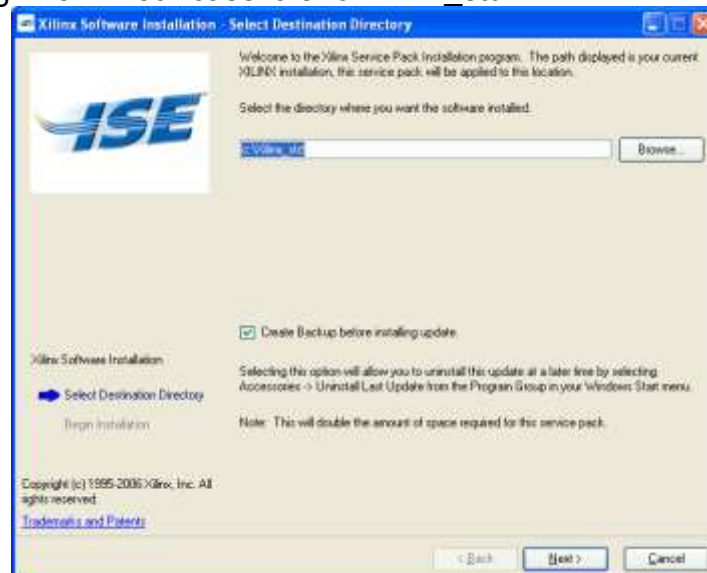
28. If you are running Windows XP, the following window may appear.



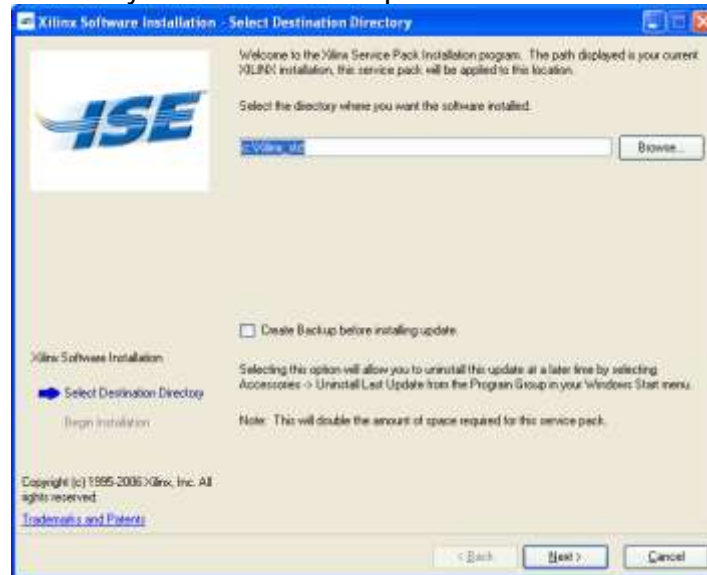
29. It is safe to click 'Run' and continue installing the service pack. Please click 'Run' to continue the service pack installation.
30. The service pack will begin extracting files needed for the update. Depending on your system this may take several minutes.



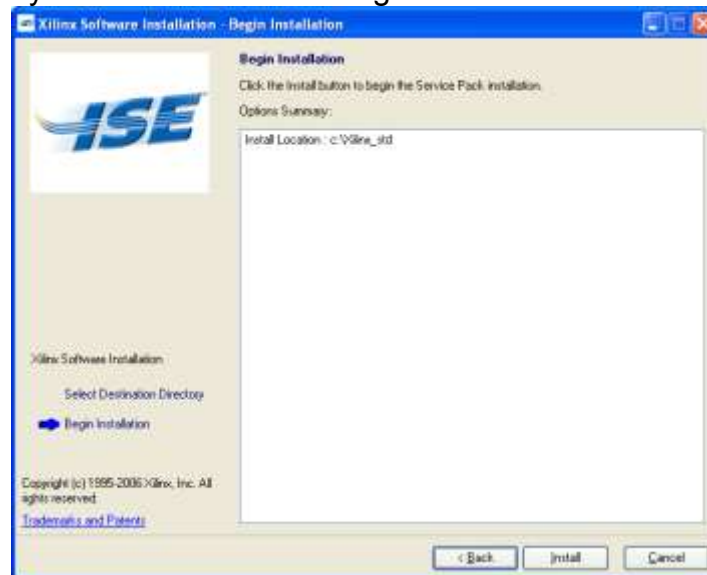
31. The next screen you should see is the destination directory screen. The destination directory listed by default should be the destination directory you previously used in steps 13 through 15. In our case it is "c:\Xilinx_std".



32. Note that the directory listed here must match the directory of the Xilinx ISE 8.2i install you wish to apply the service pack to.
33. It is recommended to uncheck the “Create Backup before installing update” checkbox in order to conserve disk space. If you feel it necessary to revert back to the previous version of the Xilinx ISE tools, you may uninstall it and perform a fresh install.
34. Review the destination directory and the “Create Backup before installing update” checkbox and when they are both correct please click ‘Next’.



35. The next screen you should see is the begin installation screen.



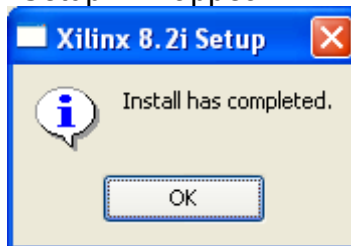
36. You should see a listing of all the options you selected in the previous screens. Please review these options to ensure they are correct. After you have reviewed the options listed please click ‘Install’. If something is in error click ‘Back’ to return to the appropriate screen and correct the option.
37. The next screen you should see is the installation progress screen. This may take some time depending on your system, between 30 to 90 minutes.



38. Wait for the installation to complete. Advertisements may vary.

39. A window labeled 'set up' will pop up momentarily.

40. A window labeled 'Xilinx 8.2i Setup' will appear



41. Please click 'OK' and the window will close.

42. Confirm the update finished correctly; launch the Xilinx ISE install you just updated.

43. Click Help->About.

44. An 'About Project Navigator' window should appear.

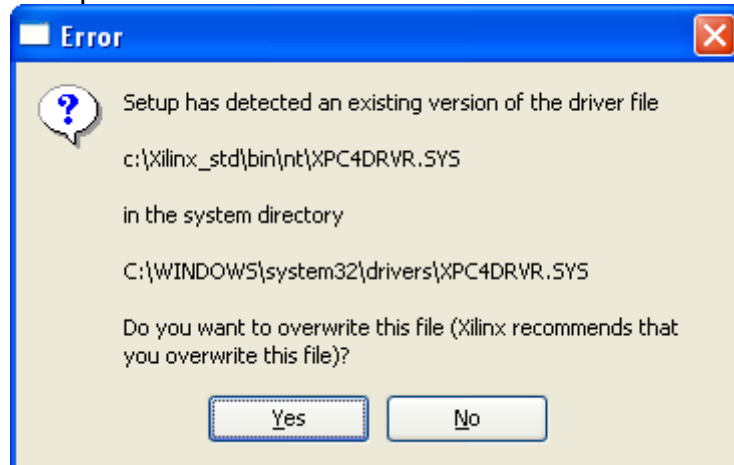


45. Confirm the Application Version matches what you expect.

46. You may now begin using the Xilinx ISE 8.2i with the latest service pack for system development and debugging on Xilinx FPGAs.

2.2.1.1 Potential Errors

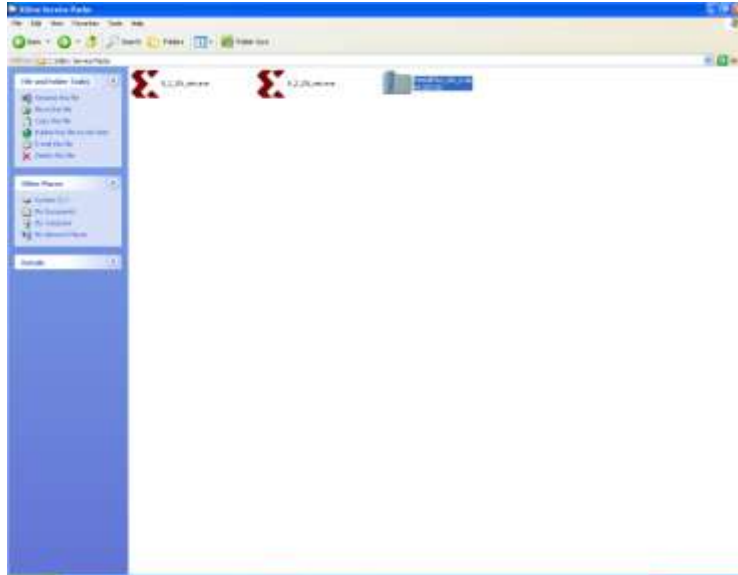
Between steps 24 and 25 if you have a previous install of the Xilinx ISE tools, it is possible for the following window to appear. If your previous install of the Xilinx ISE is older than the Xilinx ISE 8.2i, it is recommended that you overwrite the driver. Click the appropriate button and the installation process should resume.



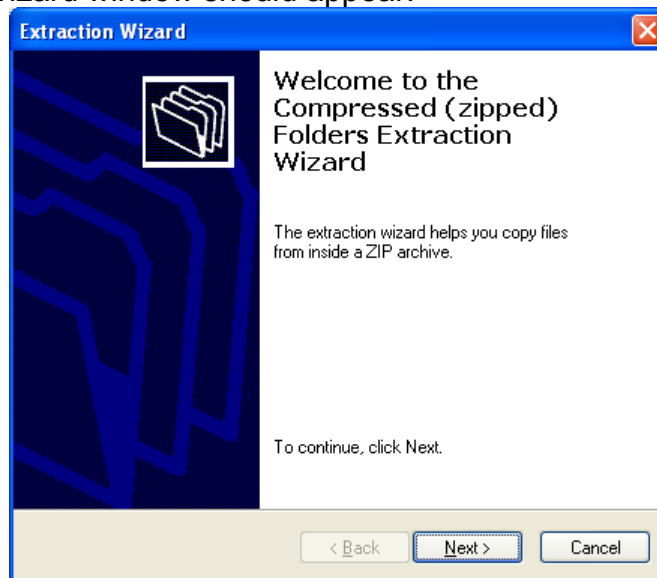
2.2.2 Xilinx ISE Partial Reconfiguration Tools

The following are the instructions for installing the Xilinx ISE for performing the Xilinx Partial Reconfiguration Design Flow. These instructions are supplementary to those provided by Xilinx at the Xilinx Partial Reconfiguration Early Access Lounge (<http://www.xilinx.com/support/prealounge/protected/index.htm>). Please refer to the Xilinx Documentation for any further assistance. This toolset is required to fully utilize all of the features of the eMIPS system. This toolset is the Xilinx ISE updated to Service Pack 1 with additional changes to all for use of the partial reconfiguration feature.

1. Obtain the Xilinx Partial Reconfiguration Tools overlay from the Xilinx Partial Reconfiguration Early Access Lounge.
2. Follow steps 1 through 29 of Xilinx ISE to create a new install of the Xilinx ISE 8.2i in a new destination directory. For this example use "c:\Xilinx_pr". Your destination directory may vary.
3. Obtain Service Pack 1 for Xilinx ISE 8.2i from the Xilinx Partial Reconfiguration Early Access Lounge.
4. Follow steps 31 through 45 of Xilinx ISE to install Service Pack 1 (8_2_01i_win.exe). Your filenames may vary.



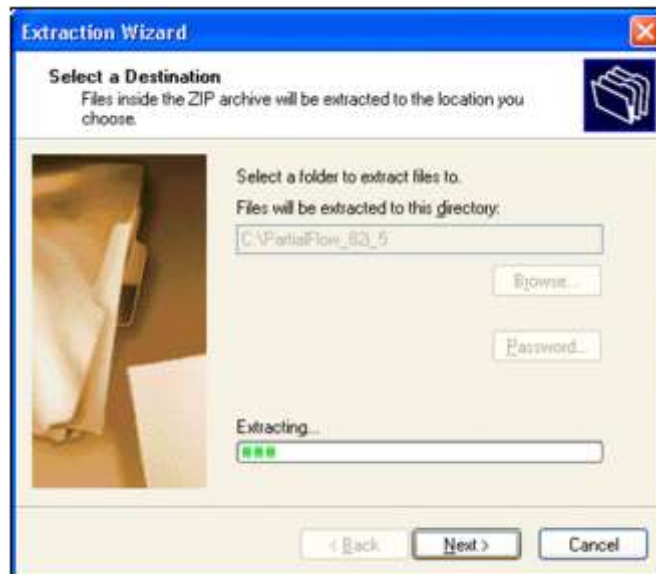
5. If you are running Windows XP, Right-click on the Xilinx Partial Reconfiguration Tools overlay zip file and select 'Extract all'.
6. The Extraction Wizard window should appear.



7. Please click 'Next' to continue.
8. The next screen you should see is the destination selection screen. By default it will have your current directory selected.

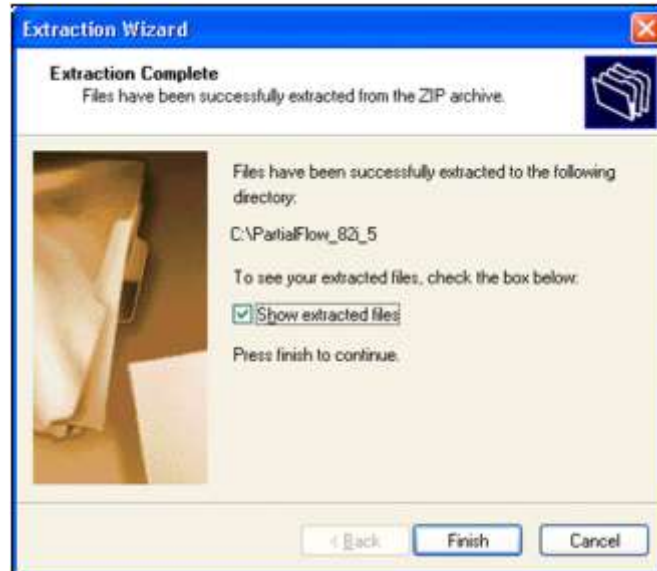


9. Click 'Next' to continue.

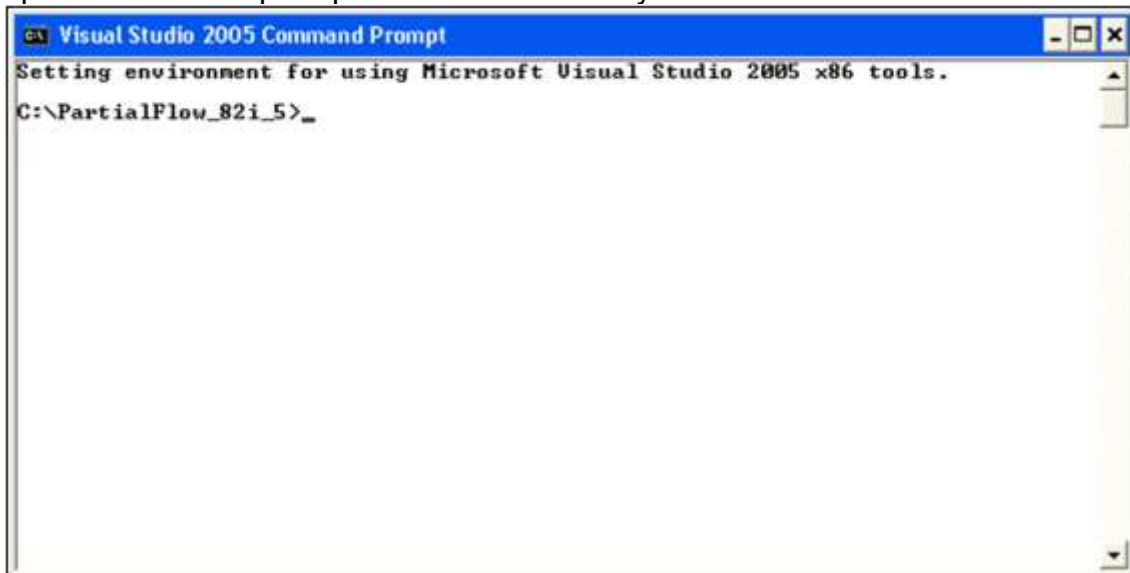


10. You should see the progress bar begin to fill. This may take some time depending on your system, between 10 to 15 minutes.

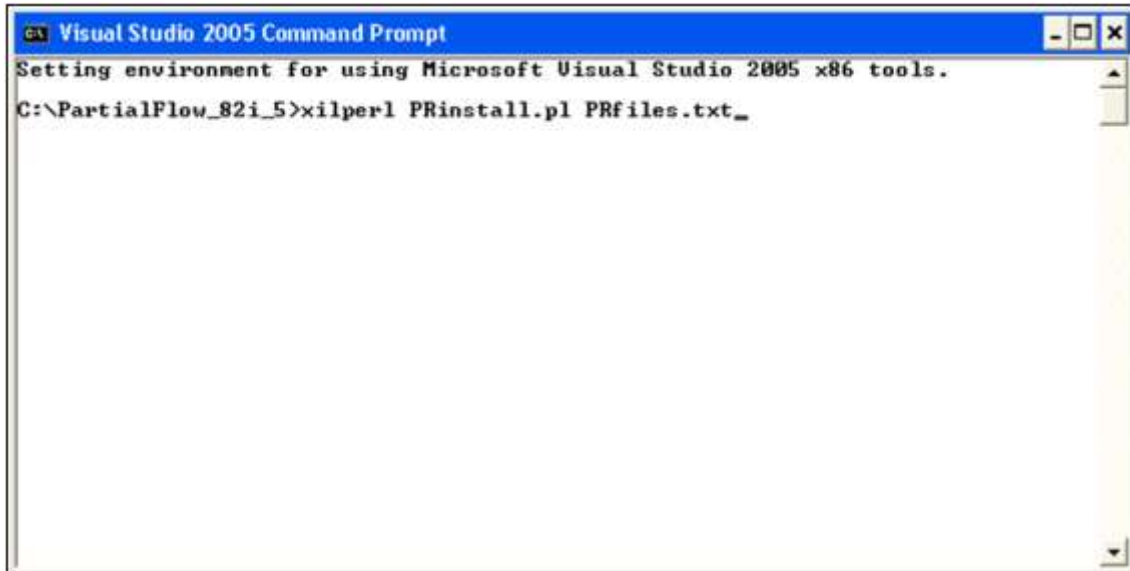
11. The next screen you should see is the extraction complete screen



12. Please click 'Finish' to complete the application of the Xilinx Partial Reconfiguration Tools
13. A folder containing this install of the Xilinx ISE should appear.
14. Open a command prompt to this new directory.



15. Run the install script
 - a. Type: xilperl PRinstall.pl PRfiles.txt



```
Visual Studio 2005 Command Prompt
Setting environment for using Microsoft Visual Studio 2005 x86 tools.
C:\PartialFlow_82i_5>xilperl PRinstall.pl PRfiles.txt_
```

b. Press Enter.

c. Wait for the process to complete. You may disregard the warnings.



```
Visual Studio 2005 Command Prompt
Setting environment for using Microsoft Visual Studio 2005 x86 tools.
C:\PartialFlow_82i_5>xilperl PRinstall.pl PRfiles.txt
Partial Reconfiguration 8.2.01_PR_5 installer (05-07-2007)
  found version = 8.2i
  found version = 8.2.01i
WARNING - Failed to create back up c:\Xilinx_pr/bin/nt/libXdh_DRC.dll_prePR.
          c:\Xilinx_pr/bin/nt/libXdh_DRC.dll does not exist. Installer will use
PR file.
WARNING - Failed to create back up c:\Xilinx_pr/bin/nt/libNgtNm1.dll_prePR.
          c:\Xilinx_pr/bin/nt/libNgtNm1.dll does not exist. Installer will use P
R file.
WARNING - Failed to create back up c:\Xilinx_pr/bin/nt/libPackUtil.dll_prePR.
          c:\Xilinx_pr/bin/nt/libPackUtil.dll does not exist. Installer will use
PR file.
C:\PartialFlow_82i_5>
```

16. Confirm the update finished correctly; launch the Xilinx ISE install you just created.

17. Click Help->About.

18. An 'About Project Navigator' window should appear.

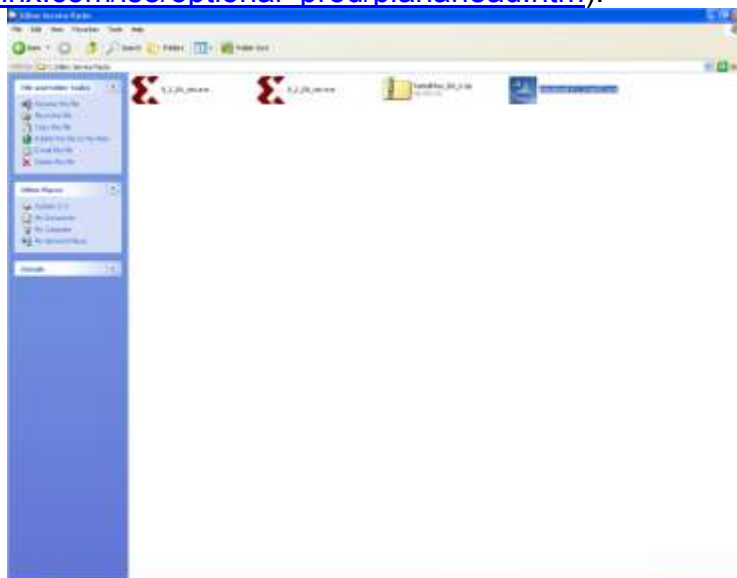


19. Confirm the Application Version matches what you expect.
20. You may now begin using the Xilinx ISE 8.2i with the Partial Reconfiguration Tools for building Partial Reconfiguration enabled designs on Xilinx FPGAs.

2.2.3 Xilinx PlanAhead

The following are the instructions for installing Xilinx PlanAhead, using the Evaluation Version obtained with a download from the appropriate Xilinx PlanAhead website (http://www.xilinx.com/ise/optional_prod/planahead.htm). If you use a different means of installing the tools (DVD, CD, server, etc) your experience may differ. PlanAhead is a tool used for floor planning and for performing design checking of partially reconfigurable designs. If you do not plan to modify the floorplan of the eMIPS system you will not be required to have PlanAhead installed. We provide a fully floorplanned constraint file for use in the build process.

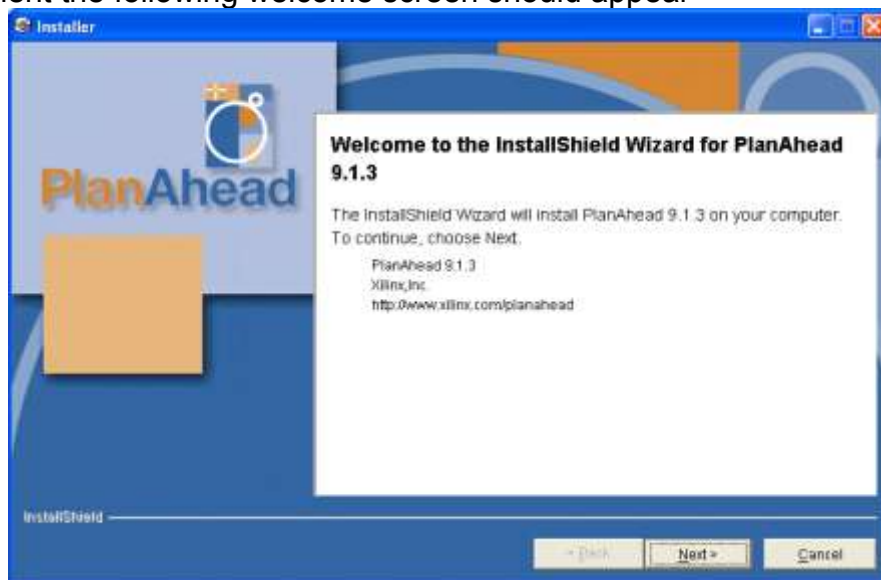
1. Obtain the Xilinx PlanAhead Evaluation Version software from Xilinx website (http://www.xilinx.com/ise/optional_prod/planahead.htm).



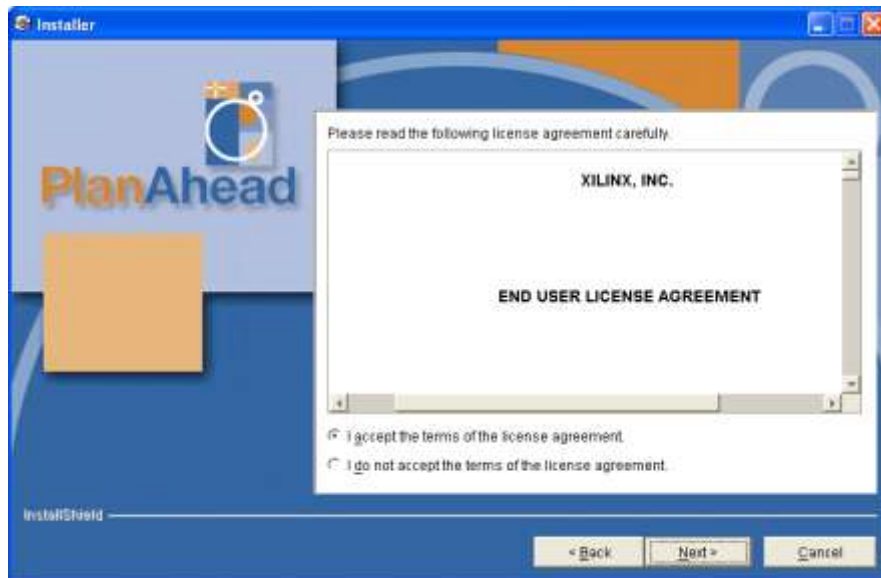
2. Double-click on the Xilinx PlanAhead installation file (planahead-9.1.3-win32.exe).
3. If you are running Windows XP, the following window may appear.



4. It is safe to click 'Run' and continue installing the software. Please click 'Run' to continue the software installation.
5. In a moment the following welcome screen should appear



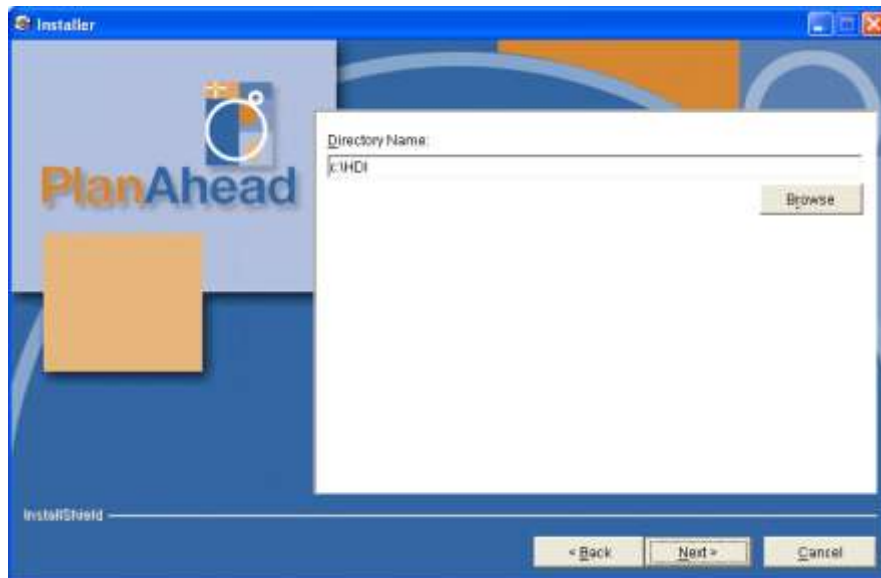
6. Please click 'Next' to continue.
7. The next screen you should see is the End User License Agreement Screen.



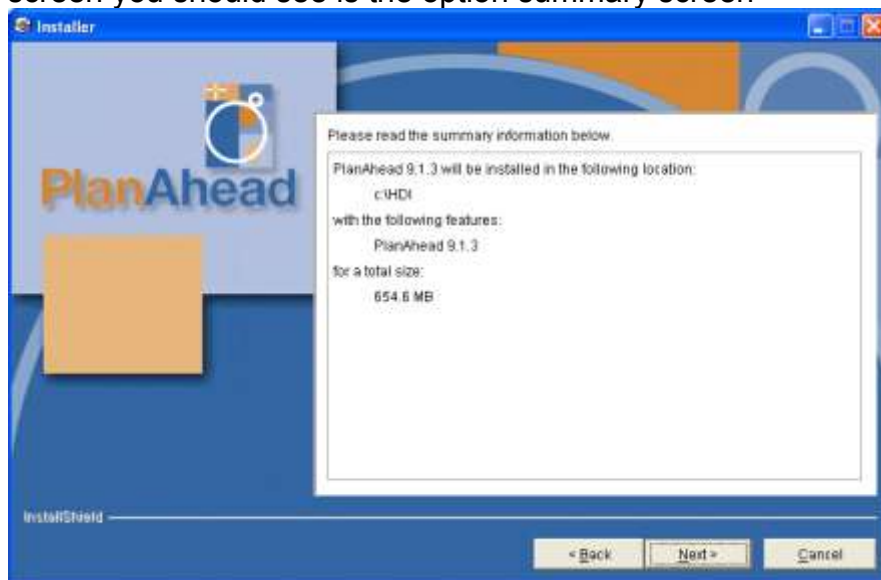
8. Please read the End User License Agreement. If you feel the terms of the End User License Agreement are acceptable to you and your organization, check the checkbox labeled “I accept the terms of this software license” and click ‘Next’.



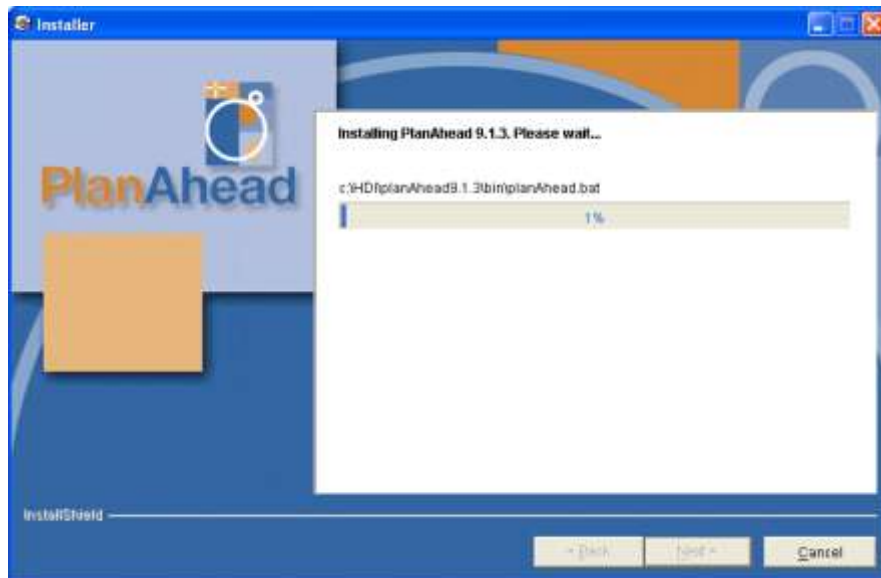
9. The next screen you should see is the destination directory screen.



10. You may use the default directory for this install. After reviewing the destination directory please click 'Next' to continue.
11. The next screen you should see is the option summary screen

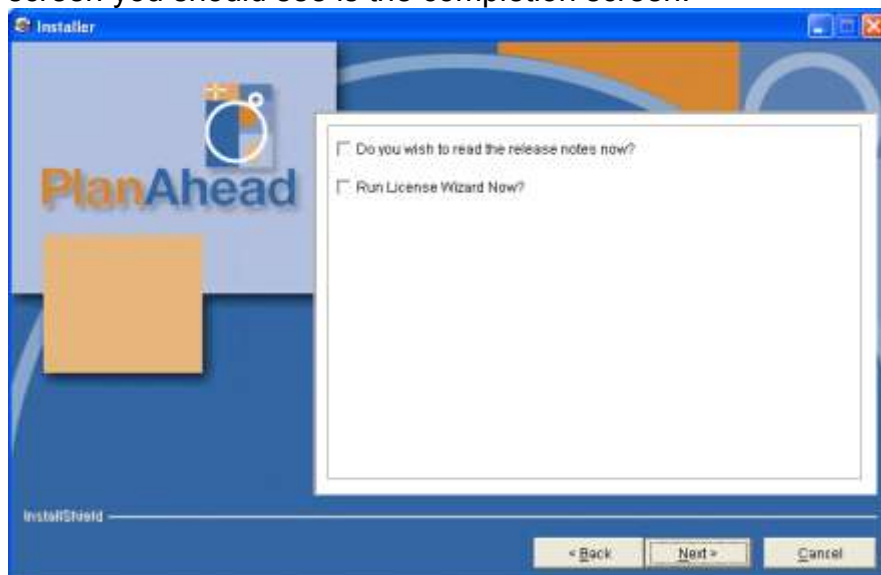


12. Review the options listed and if they are correct, please click 'Next' to continue. Otherwise click 'Back' to return to the previous screens and correct the errors.
13. The next screen you should see is the progress screen. This may take some time depending on your system, between 15 to 30 minutes.



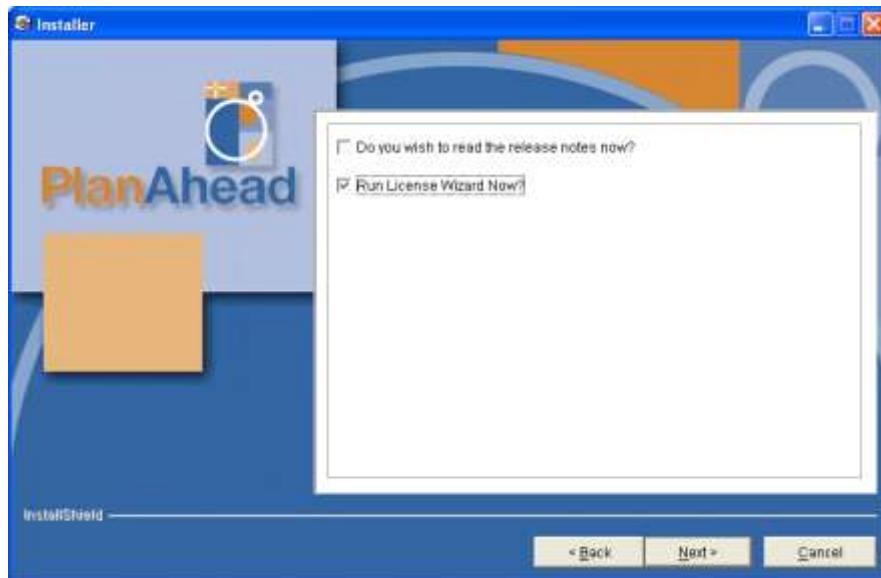
14. Wait for the installation to complete.

15. The next screen you should see is the completion screen.



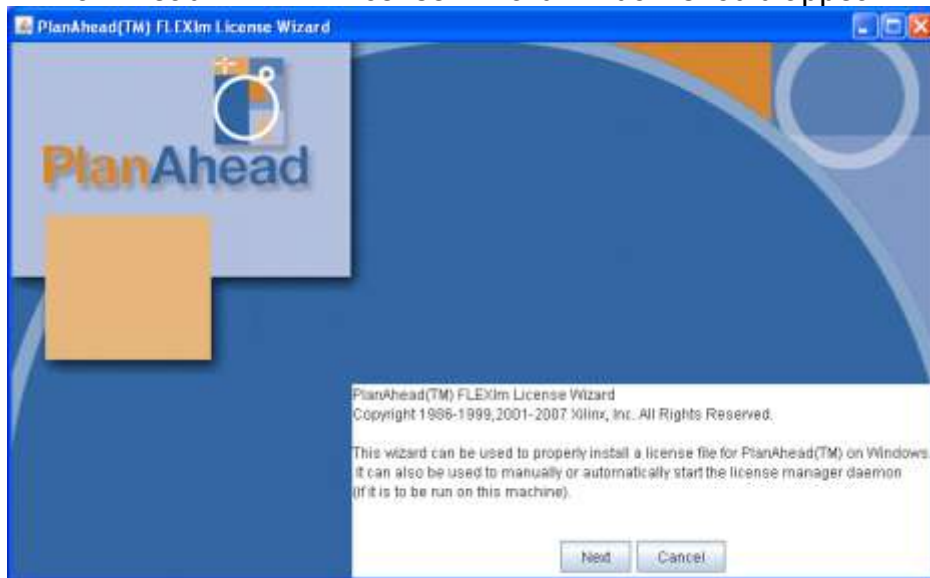
16. If you have a license for Xilinx PlanAhead already you may check the checkbox labeled 'Run License Wizard Now?'.

17. If you have not already obtained a Xilinx PlanAhead license you may obtain one at the Xilinx PlanAhead website (http://www.xilinx.com/ise/optional_prod/planahead.htm).



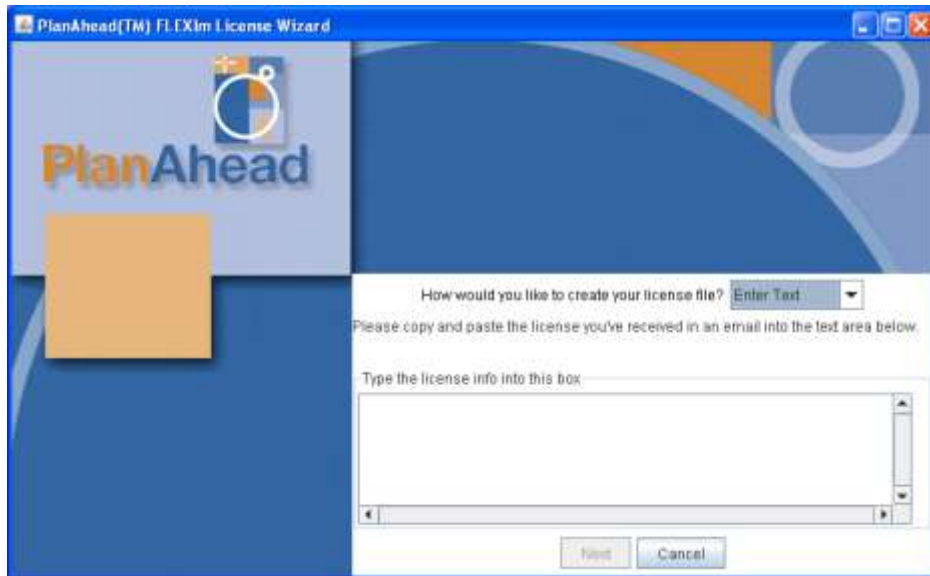
18. After you have reviewed these options please click 'Next' to finish the software installation.

19. The Xilinx PlanAhead FLEXlm License Wizard window should appear.

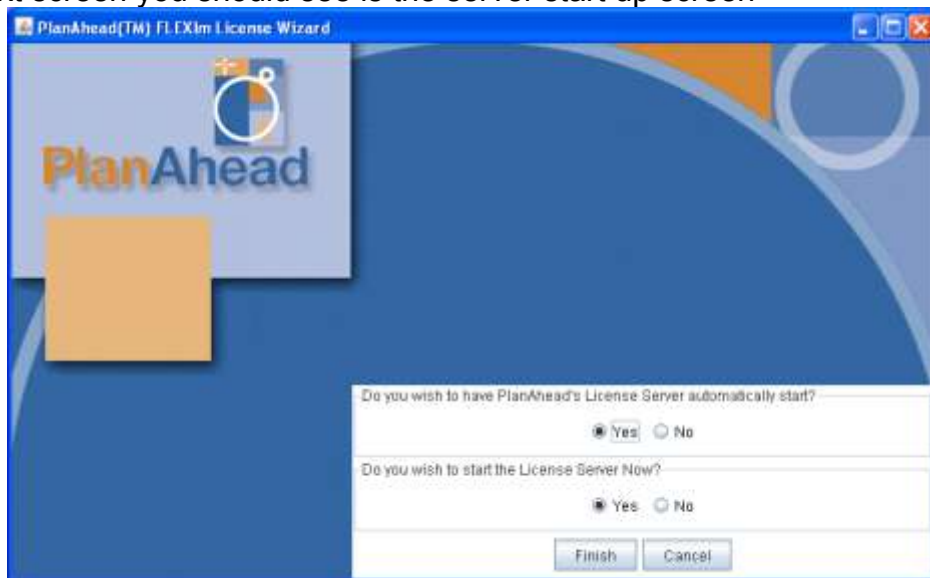


20. Please click 'Next' to continue.

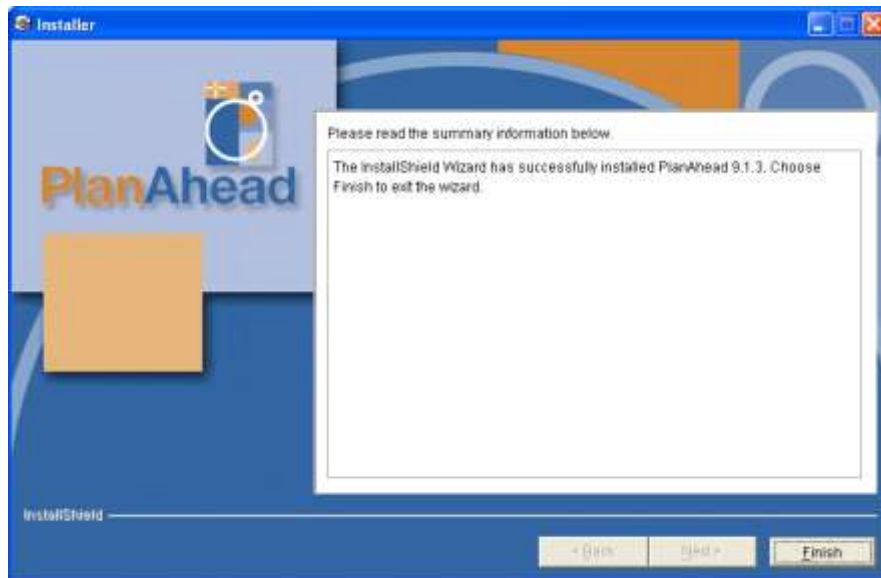
21. The next screen you should see is the create license file screen.



22. Your license should have come in the form of an email message. Please copy and paste your email into the text box provided and click 'Next'.
23. The next screen you should see is the server start up screen



24. Please select 'yes' on both items listed.
25. After reviewing your selections please click 'Finish' to complete the license setup.
26. The Xilinx PlanAhead FLEXIm License Wizard window will close and the next screen you should see is the installation summary screen.



27. Please click 'Finish' to complete the software installation
28. The installation window will close.
29. You may now begin using the Xilinx PlanAhead for floor planning Partial Reconfiguration enabled designs on Xilinx FPGAs.

2.3 Software Tools

The amount of software tools required for developing programs for the eMIPS systems depends on the complexity of the application. For simple programs and tests that exercise the hardware extensions it suffices to use just the compiler toolset. All the eMIPS software test binaries in this distribution were created using the standard GCC v4.2.0 distribution, plus the Binutils package v2.17 and GDB v6.6. The BBTools package, provided in this distribution, is useful for analyzing applications in more details, to patch binaries and to automatically generate an Extension starting from a basic block of MIPS binary code. The Giano full-system simulator is useful for profiling applications, and for developing and debugging Extensions especially when used in concert with Modelsim. If you do not have an ML401 board you can still develop and run your software using the Giano simulation of that board. The Microsoft Invisible Computing RTOS is useful for developing more complex applications, especially those that require real-time interactions with the environment, or access to file systems and networking facilities.

2.3.1 Compiler

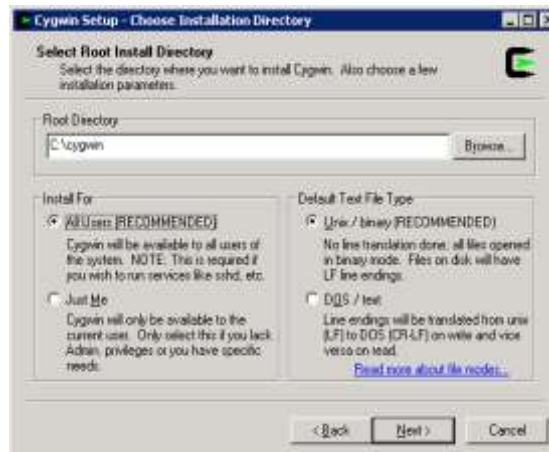
The eMIPS basic data path implements the MIPS-1 instruction set [10]. Any compiler capable of targeting this ISA, for any programming language, is usable with eMIPS. The test programs provided in this distribution are written in C and have been compiled using GCC. In this section we show how to recreate this set of compiler tools. The instructions are for specific versions of the tools. This does not mean they are better or worse than others, just that they worked for us. The following instructions are for installing on Windows XP. There is a pre-built set of GCC tools for eMIPS at http://xoomer.alice.it/giovanni_busonera/eMIPS/eMIPS.htm.

2.3.1.1 Cygwin

The simplest way to compile GNU software is to create a Unix environment. There are many that work under Windows, one that is easy to get and use is the Cygwin one. The site for these tools is <http://www.cygwin.com>. We used version 1.5.24, but it should be ok to just use the most recent one. When we tried to install on a Win64 machine it did not work, but we did not investigate this problem. To get started, download and run the Setup.exe application from <http://www.cygwin.com/setup.exe>. Create a new directory for it, such as c:/cygwin. Download setup.exe in there and start it. This application is also used to get updates if you need them, which is why you want to keep a copy locally. The following screen should popup, hit Next.



The next screen is for the installation options, select the recommended ones and hit Next.



The next few screens are for various installation options such as the location of downloaded programs, proxy settings to connect to the Internet, and the choice of a download site. Select the options that work for you. Eventually you should get to the following package selection screen.



The list is fairly large and your selection can vary. The following list of packages is non-minimal, but it is known to work.

DRAFT – DO NOT REPRODUCE

Base	77k	alternatives: A tool for managing package conflicts
Base, Shells	47k	ash: A Bourne Shell (/bin/sh) workalike
Devel	3k	autoconf: Wrapper scripts for autoconf commands
Devel	203k	autoconf2.1: Stable version of the automatic configure script builder
Devel	791k	autoconf2.5: Development version of the automatic configure script builder
Base	35k	base-files: A set of important system configuration and setup files
Base	1k	base-passwd: A script to set up initial passwords and groups
Base, Shells	516k	bash: The GNU Bourne Again SHell
Math, Utils	64k	bc: The GNU numeric processing language and reverse polish calculator
Devel	4,815k	binutils: The GNU assembler, linker and binary utilities
Devel	294k	bison: A parser generator that is compatible with YACC
Devel	28k	byacc: The Berkeley L&LR parser generator
Utils	407k	bzip2: A high-quality block-sorting file compressor - utilities
Base	2,365k	coreutils: GNU core utilities (includes fileutils, sh-utils and textutils)
Utils	116k	cpio: A backup and archiving utility
Libs	13k	crypt: Encryption/Decryption utility and library
Devel	709k	cvs: Concurrent Version System
Utils	182k	cygutils: A collection of simple utilities
Base	1,363k	cygwin: The UNIX emulation engine
Base, Doc	919k	cygwin-doc: Cygwin-specific documentation, including man pages and User's Guide
Base	28k	editrights: Alter user rights and privileges
Devel, Doc, Interp	111k	expat: XML parser library written in C (development/documentation package)
Base	459k	findutils: Utilities for finding files--find, xargs, locate, updatedb
Devel	88k	flex: A fast lexical analyzer generator
Base, Interpreters	617k	gawk: GNU awk, a pattern scanning and processing language
Devel	1k	gcc: C compiler upgrade helper
Devel	3,618k	gcc-core: C compiler
Devel	2,958k	gcc-g++: C++ compiler
Devel	69k	gcc-mingw-core: Mingw32 support headers and libraries for GCC
Devel	1,894k	gcc-mingw-g++: Mingw32 support headers and libraries for GCC C++
Devel	?	gcc-testsuite: GCC testsuite sources
Devel	3,958k	gdb: The GNU Debugger
Base	151k	grep: Search and print textual input for lines which match a specified pattern
Text	1,991k	groff: GNU troff text-formatting system
Base	62k	gzip: The GNU compression utility
Text	79k	less: A file pager program, similar to more(1)
Libs	27k	libbz2_1: Shared libraries for bzip2 (runtime)
Database	623k	libdb4.2: The Sleepycat Berkeley DB Library v4.1 - runtime
Database	680k	libdb4.3: The Sleepycat Berkeley DB Library v4.3 - runtime
Doc, Interpreters, I	51k	libexpat0: XML parser library written in C (runtime package)
Database, Libs	13k	libgdbm4: GNU dbm database routines (runtime)
Libs	692k	libiconv2: GNU character set conversion library and utilities - runtime (1)
Libs	20k	libintl2: GNU Internationalization runtime library
Libs	16k	libintl3: GNU Internationalization runtime library
Libs	17k	libintl8: GNU Internationalization runtime library
Libs	171k	libncurses8: Libraries for terminal handling (runtime)
Libs	162k	libpcre0: Perl-Compatible Regular Expressions (runtime library)
Libs	12k	libpopt0: Library for parsing cmdline parameters - runtime
Libs	81k	libreadline6: GNU readline and history libraries (runtime)
Base	14k	login: Sign on to a system
Interpreters	207k	m4: GNU implementation of the traditional Unix macro processor.
Devel	348k	make: The GNU version of the 'make' utility
Base, Doc, System	197k	man: Man, apropos and whatis
Mingw	338k	mingw-bzip2: (mingw version) of bzip2, a high-quality block-sorting file compressor
Mingw	27k	mingw-libbz2_1: (mingw version) shared libraries for bzip2 (runtime)
Devel, Libs	267k	mingw-runtime: MinGW Runtime
Devel	11k	mktemp: Allows safe temp file/dir creation from shell scripts
Interpreters, Perl	7,464k	perl: Larry Wall's Practical Extracting and Report Language
Base, System, Utils	90k	rebase: Utilities for rebasing DLLs to load at alternate addresses
Base	35k	run: Start console programs with hidden console
Base	139k	sed: The GNU sed stream editor
Base	696k	tar: A GNU file archiving program
Libs	1,854k	tcltk: TCL/Tk libraries
Base, Libs	20k	termcap: Compatibility package for old termcap-based programs
Base	197k	terminfo: Database for ncurses-style terminal handling
Doc, Text	691k	texinfo: Documentation system for on-line information and printed output
Libs	1,184k	w32api: Win32 API header and library import files
Base	3k	which: Displays where a particular program in your path is located

Once all packages are downloaded and installed you should be able to start the bash shell, which will create a window similar to the following.

```

Your group is currently "mkpasswd". This indicates that
the /etc/passwd (and possibly /etc/group) files should be rebuilt.
See the man pages for mkpasswd and mkgroup then, for example, run
mkpasswd -l [-d] > /etc/passwd
mkgroup -l [-d] > /etc/group
Note that the -d switch is necessary for domain users.

sandrof@msr-build2 ~
$

```

2.3.1.2 GCC

The reference site for GCC is <http://gcc.gnu.org>. You can download a compressed archive file of the sources from the site <ftp://ftp.gnu.org/>, or one of its mirror sites (e.g. chose the one closest to your location). Move to the *gnu/gcc/gcc-4.2.0* sub-directory and retrieve the file *gcc-4.2.0.tar.bz2*. Uncompress this file using the BZIP2 utility, the command is “*bzip2 -d gcc-4.2.0.tar.bz2*”.

```

/cygdrive/c/cygwin
$ ls -l
total 43028
-rwxr-xr-x 1 Administrators mkpasswd 44057527 Dec 10 15:27 gcc-4.2.0.tar.bz2

sandrof@msr-build2 /cygdrive/c/cygwin
$ bzip2 -d gcc-4.2.0.tar.bz2

sandrof@msr-build2 /cygdrive/c/cygwin
$ ls -l
total 294232
-rwxr-xr-x 1 Administrators mkpasswd 301291520 Dec 10 15:27 gcc-4.2.0.tar

sandrof@msr-build2 /cygdrive/c/cygwin
$

```

Use the TAR utility to extract the files, the command is “*tar xf gcc-4.2.0.tar*”. This will take some time, there are many files. Eventually the sources should be in the newly created directory *gcc-4.2.0*.

```

/cygdrive/c/cygwin
total 294232
-rwxr-xr-x 1 Administrators mkpasswd 301291520 Dec 10 15:27 gcc-4.2.0.tar

sandrof@msr-build2 /cygdrive/c/cygwin
$ tar xf gcc-4.2.0.tar

sandrof@msr-build2 /cygdrive/c/cygwin
$ ls -l
total 294232
drwxr-xr-x+ 25 sandrof          mkpasswd          0 May 13  2007 gcc-4.2.0
-rwxr-xr-x  1 Administrators mkpasswd 301291520 Dec 10 15:27 gcc-4.2.0.tar

sandrof@msr-build2 /cygdrive/c/cygwin
$

```

Detailed installation instructions for GCC are at <http://gcc.gnu.org/install/> and we will not repeat them here. You want to target a generic MIPS ELF machine, e.g. configure GCC with arguments like the following:

configure --prefix=/emips --with-divide=breaks --enable-languages=c,c++ --target=mips-elf

The most recent versions of the configure utility are capable of configuring multiple packages at once. You might want to first retrieve Binutils and GDB before you run it. After you have configured all packages, a single *make* command should build everything for you.

2.3.1.3 Binutils

You can download a compressed archive file of the Binutils package from the site <ftp://ftp.gnu.org/>, or one of its mirror sites (e.g. chose the one closest to your location). Move to the *binutils* sub-directory and retrieve the file *binutils-2.17.tar.bz2*. Uncompress and unpack this file as before, using BZIP2 and TAR. Notice that you want to move to the *gcc-4.2.0* directory above before you unpack.

2.3.1.4 GDB

You can download a compressed archive file of the GDB debugger from the site <ftp://ftp.gnu.org/>, or one of its mirror sites (e.g. chose the one closest to your location). Move to the *gdb* sub-directory and retrieve the file *gdb-6.6.tar.bz2*. Uncompress and unpack this file as before, using BZIP2 and TAR. Notice that you want to move to the *gcc-4.2.0* directory above before you unpack.

There is one source change that is necessary before you build GDB. For some reason, the default release assumes that the MIPS processor has hardware support for breakpoints, which is not (entirely) true for eMIPS. To reinstate software breakpoints edit the file *gdb/mips-tdep.c* and add the following statement at line 5154:

```
set_gdbarch_software_single_step (gdbarch, mips_software_single_step);
```

There might be better ways to achieve this effect, but we did not investigate them. If you use eBug with watchpoints enabled you might not need to apply this change.

Once you are done building everything you should have the following binaries:

```
addr2line.exe  
ar.exe  
as.exe  
cc1.exe  
cc1plus.exe  
collect-ld.exe  
collect2.exe  
cpp.exe  
cxxfilt.exe  
g++.exe  
gcc.exe  
gcov-dump.exe  
gcov.exe
```

gdb.exe
ld.exe
nm.exe
objcopy.exe
objdump.exe
ranlib.exe
readelf.exe
size.exe
strings.exe
strip.exe

You can install them in other places, but the script for building the hardware tests assume that you have them in your MMLITE_SDK directory, under the tools sub-directory, as follows:

```

C:\ Command Prompt
E:\gnu>dir \MMLitePublic\tools\emips\bin
Volume in drive E is data
Volume Serial Number is FC6C-420C

Directory of E:\MMLitePublic\tools\emips\bin

09/12/2007  01:19 PM  <DIR>          .
09/12/2007  01:19 PM  <DIR>          ..
06/12/2007  03:58 PM                2,002,828  addr2line.exe
06/12/2007  03:58 PM                1,916,805  ar.exe
06/12/2007  03:58 PM                2,892,319  as.exe
06/12/2007  03:58 PM               14,531,685  cc1.exe
06/12/2007  03:59 PM               16,040,746  cc1plus.exe
06/12/2007  03:59 PM               2,594,757  collect-ld.exe
06/12/2007  03:59 PM                420,968  collect2.exe
06/12/2007  03:59 PM                542,239  cpp.exe
06/12/2007  03:59 PM               1,989,142  cxxfilt.exe
04/06/2007  12:43 PM                120,320  cygexpat-0.dll
06/12/2007  03:59 PM               1,873,811  cygwin1.dll
06/20/2007  01:17 PM                 4,264  emips.x
06/12/2007  03:59 PM                543,247  g++.exe
06/12/2007  03:59 PM                538,585  gcc.exe
06/12/2007  03:59 PM                 89,333  gcov-dump.exe
06/12/2007  03:59 PM                215,800  gcov.exe
07/27/2007  09:13 AM                8,981,520  gdb.exe
06/12/2007  03:59 PM               2,594,757  ld.exe
06/12/2007  03:59 PM               2,055,579  nm.exe
06/12/2007  03:59 PM               2,500,167  objcopy.exe
06/12/2007  03:59 PM               2,730,568  objdump.exe
06/12/2007  03:59 PM               1,918,341  ranlib.exe
06/12/2007  03:59 PM                718,500  readelf.exe
06/12/2007  03:59 PM               1,871,505  size.exe
06/12/2007  03:59 PM               1,869,151  strings.exe
06/12/2007  03:59 PM               2,500,167  strip.exe
06/20/2007  01:34 PM                 416  vssver.scc
                27 File(s)          74,057,520 bytes
                2 Dir(s)      165,045,284,864 bytes free

E:\gnu>_

```

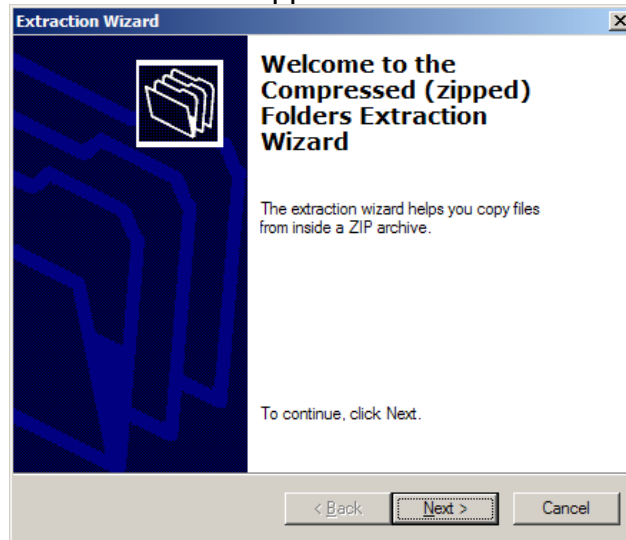
The cyg*.dll files come from the cygwin package, they are needed to run these binaries from within the regular Windows environment (e.g. in a CMD window). The emips.x link script file is included in this release.

2.3.2 The BBTools Package

The Basic Block Tools, or BBTools, are included in this distribution, in the archive file *bbtools.zip*. You should install these binaries in a location where they will be reachable from a CMD or Visual Studio command shell. This can be a directory (that you can write to) that is already on your PATH environment variable, or modify that variable to add your new directory (see section 3.1.1 for how to modify environment variables under Windows). The tools have been compiled with VC8.

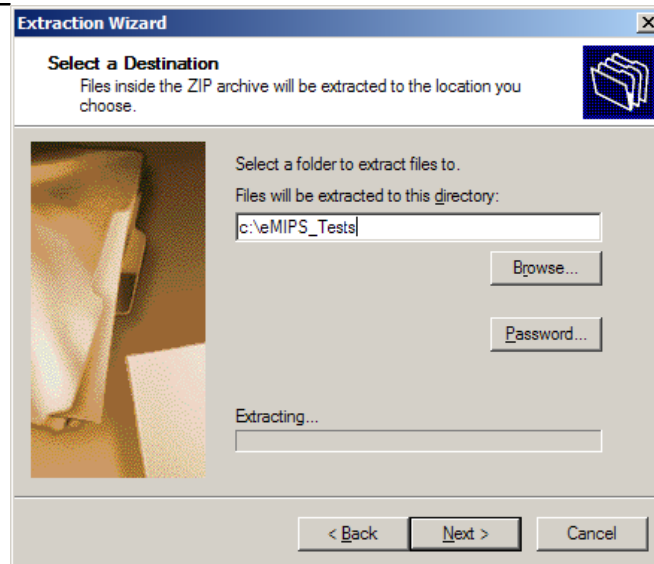
1. Extract the files from the archive. Right-click on the “bbtools.zip” zip file and select ‘Extract all’.

2. The Extraction Wizard window should appear.

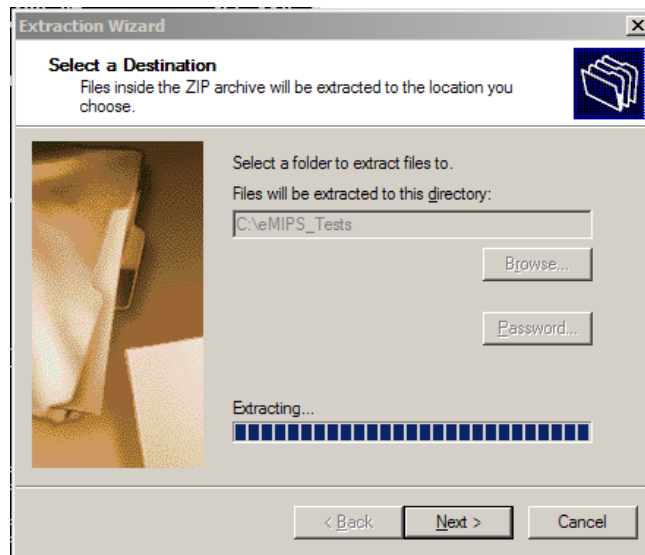


3. Please click ‘Next’ to continue.

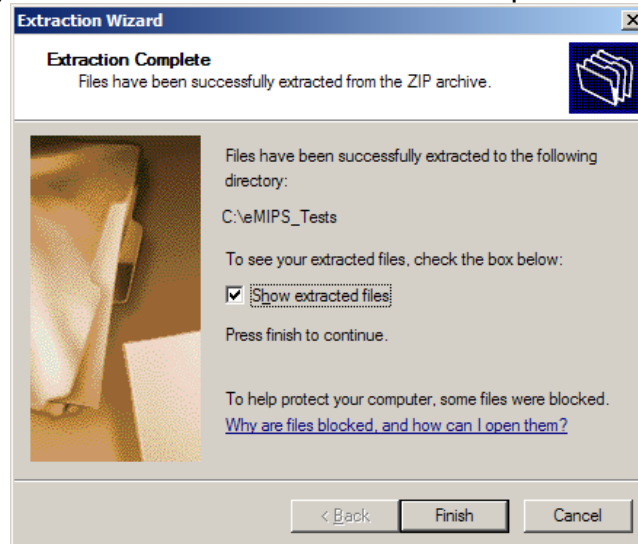
4. The next screen you should see is the destination selection screen. By default it will have your current directory selected. **Change** it to some other location; we will use the folder “c:\eMIPS_Tests” in these instructions.



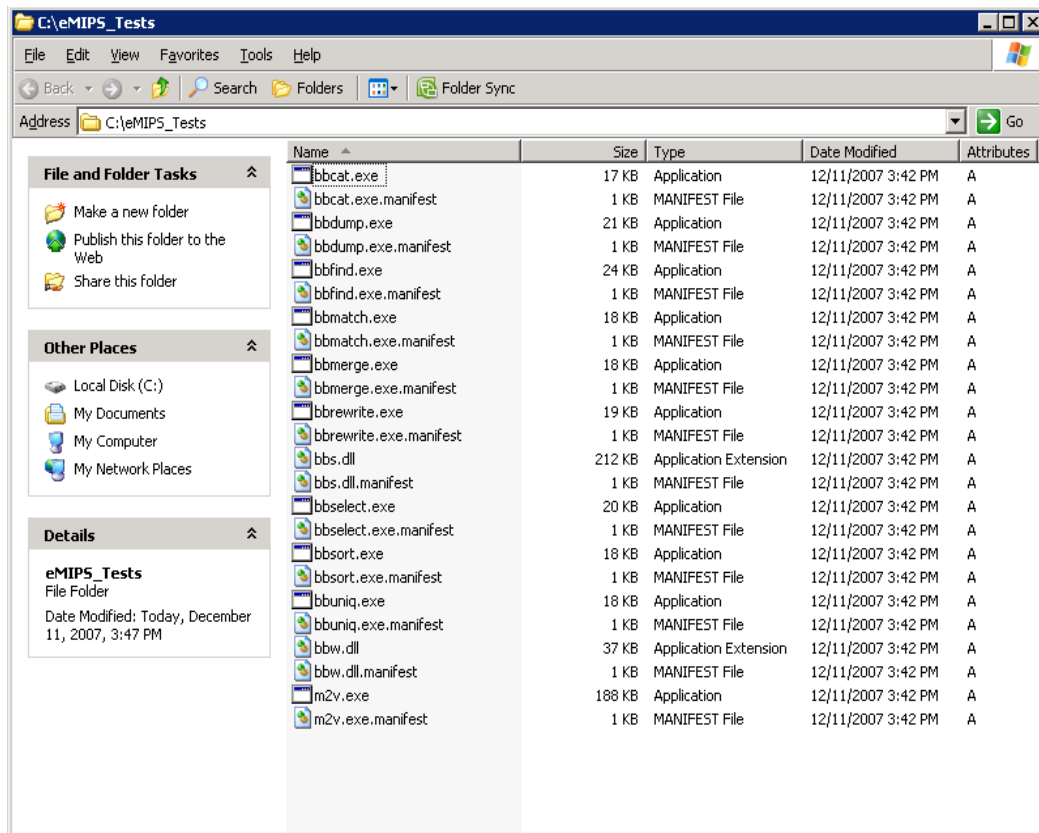
5. Click ‘Next’ to continue.



6. You should see the progress bar begin to fill.
7. The next screen you should see is the extraction complete screen



8. Please click 'Finish' to complete the extraction of the BBTools files.
9. The Folder containing the test programs should appear. It should contain twenty-four files, as follows:



2.3.3 Giano

The latest release of the Giano simulator can be downloaded from the project's web page at <http://research.microsoft.com/research/EmbeddedSystems/Giano/giano.aspx>. The same page contains the installation and test instructions, which we will not repeat here. In the *tests* directory of the distribution there is a platform configuration file for the ML401 board. This is the file you want to use for software simulations of eMIPS.

2.3.4 The RTOS

The latest release of the Microsoft Invisible Computing RTOS can be found at <http://research.microsoft.com/invisible>, along with the installation and build instructions, which we will not repeat here. You want a release with build number 112 or later. When building it, you do not need to build for all processors (e.g. using the *mkall* script). You may build the x86-release, at least for getting the most recent version of servers like SERPLEXD. You must build the *mips_gnu*-release to get the eMIPS images. The command is

```
nmake -nologo TARGETCPU=mips TARGETTYPE=release TOOLS=gnu
```

You may build the debug version, using *TARGETTYPE=debug*. At the end of the build process, you will find the new images in the *build\mips_gnu\release\bin* directory (or *debug*). The image of the base RTOS for eMIPS on the ML401 board is called *MI.bin*. Refer to section 3.5.1.3 for instructions on how to download it, flash it and test it.

To test the newly built RTOS it is advisable to run the program *alltests.exe*. This program in turn runs a number of minimal regression tests. There are a few additional tests that are specific to the ML401 board in the *tests\ml40x* directory.

3 Building the eMIPS System

There are two ways the eMIPS microprocessor system can be built, using the tools installed according to the instructions in the previous sections of this document. First, one can use the standard Xilinx ISE to build a non partial reconfiguration version with a static Extension instantiated. This is recommended for system development and debugging. The second is to use the Xilinx Partial Reconfiguration tools to build a partially reconfigurable version of the design with configurations for multiple Extensions. We will describe both procedures in the following sections.

3.1 Common Tasks

The build procedures include some tasks that are common to all builds. The corresponding procedures are described in this section and later referenced.

3.1.1 Changing the Environment to Non-PR Xilinx ISE

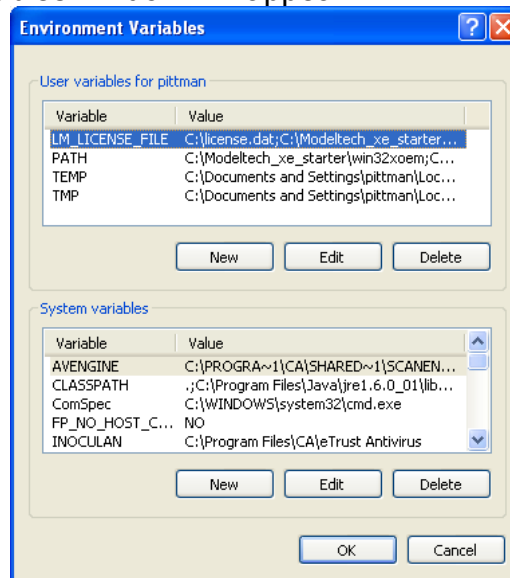
1. Right-click on 'My Computer' and select 'Properties'.
2. The System Properties window will appear.



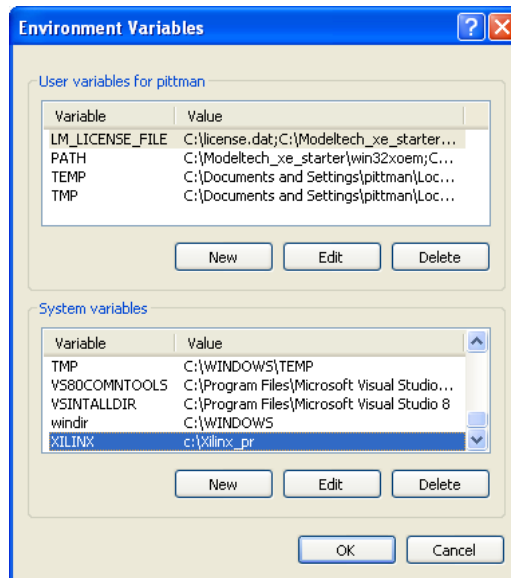
3. Select the 'Advanced' Tab.



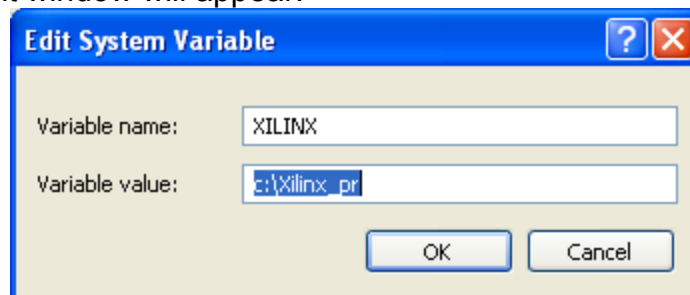
4. Click 'Environment Variables'.
5. The environment variables window will appear.



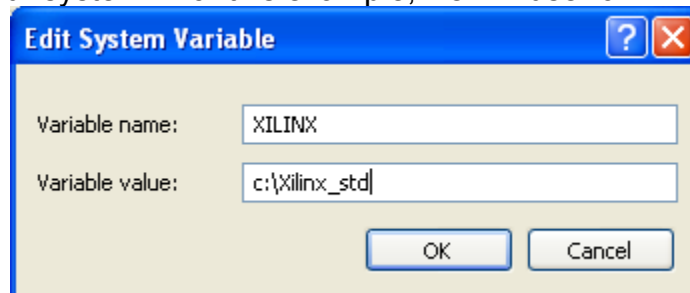
6. Locate the 'XILINX' variable and click 'Edit'.



7. The variable edit window will appear.



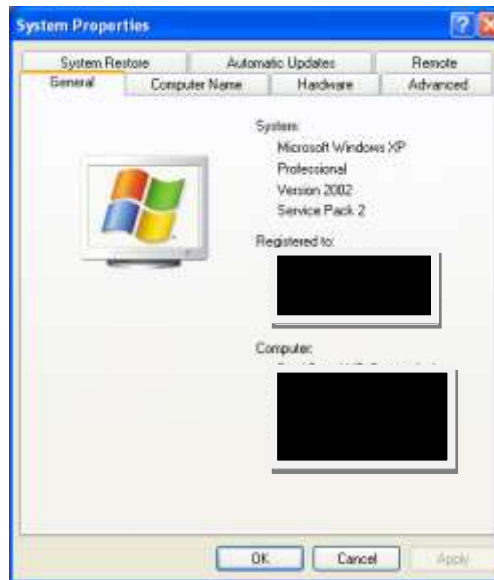
8. Change the value of the variable to the location of the standard non-pr version of the Xilinx ISE on your system. For this example, we will use "c:\Xilinx_std"



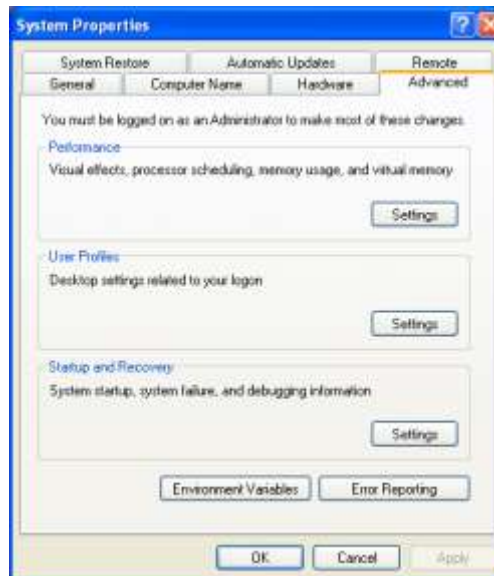
9. After you have reviewed the variable value and the desired directory, please click 'OK'. The edit window will close.
10. Click 'OK' on the environment variable window. The environment variable window will close.
11. Click 'OK' on the system properties window. The system properties window will close.
12. Some systems may require you to restart before continuing for the changes in the variables to take effect. If that is required please do that now.

3.1.2 Changing the Environment to PR Xilinx ISE

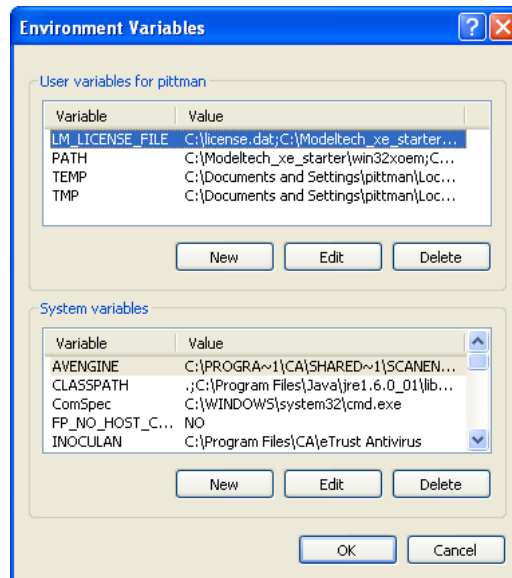
1. Right-click on 'My Computer' and select 'Properties'.
2. The System Properties window will appear.



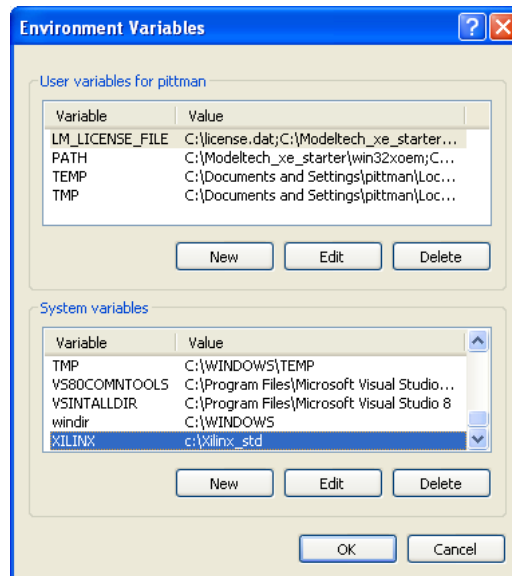
3. Select the 'Advanced' Tab.



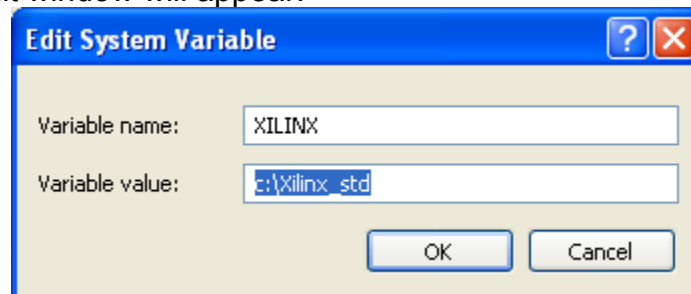
4. Click 'Environment Variables'.
5. The environment variables window will appear.



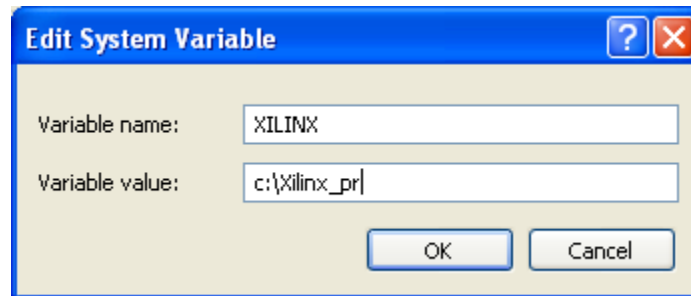
6. Locate the 'XILINX' variable and click 'Edit'.



7. The variable edit window will appear.



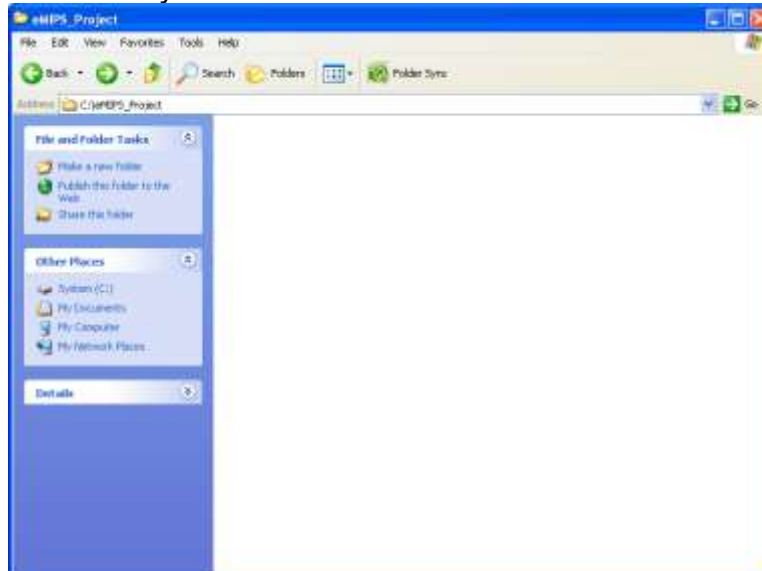
8. Change the value of the variable to the location of the standard pr version of the Xilinx ISE on your system. For this example, we will use "c:\Xilinx_pr"



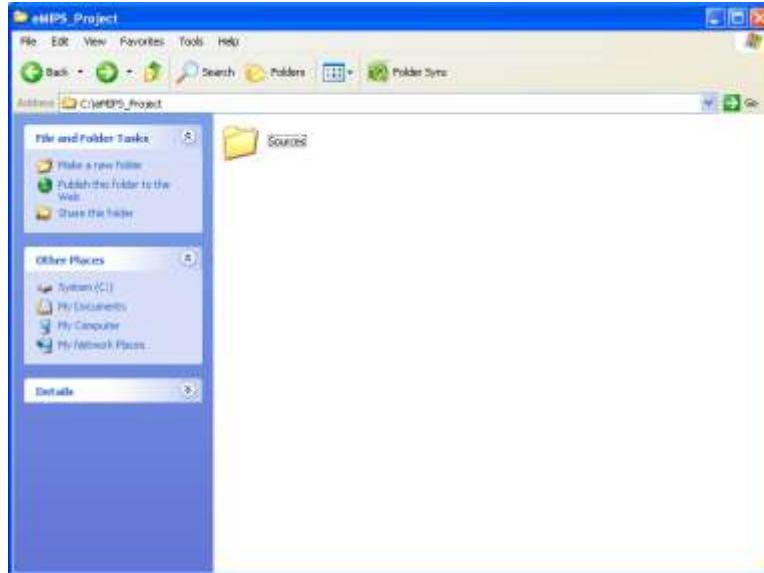
9. After you have reviewed the variable value and the desired directory, click 'OK'. The edit window will close.
10. Click 'OK' on the environment variable window. The environment variable window will close.
11. Click 'OK' on the system properties window. The system properties window will close.
12. Some systems may require you to restart before continuing for the changes in the variables to take effect. If that is required please do that now.

3.1.3 Set up the Project Directory

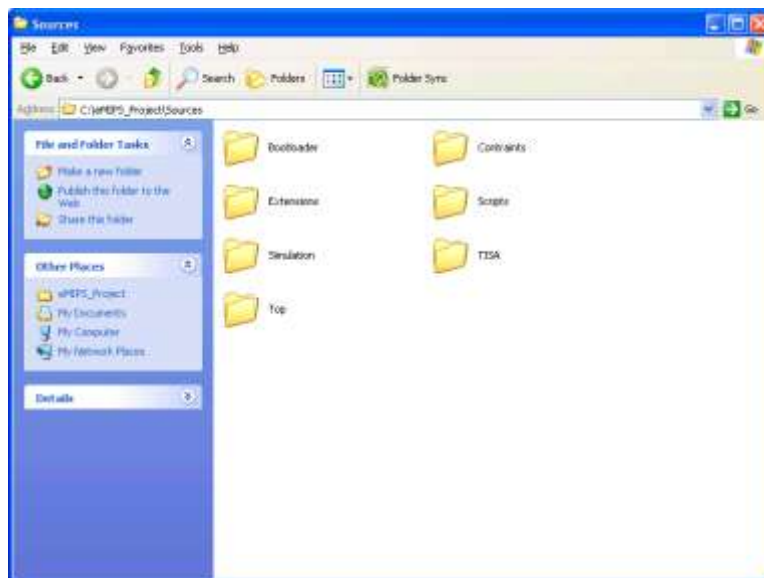
1. Download the file package from the Microsoft eMIPS website at <http://research.microsoft.com/research/EmbeddedSystems/eMIPS/emips.aspx>.
2. Unzip the file package.
3. Create a project directory in a location with 200 MB or more of available space.



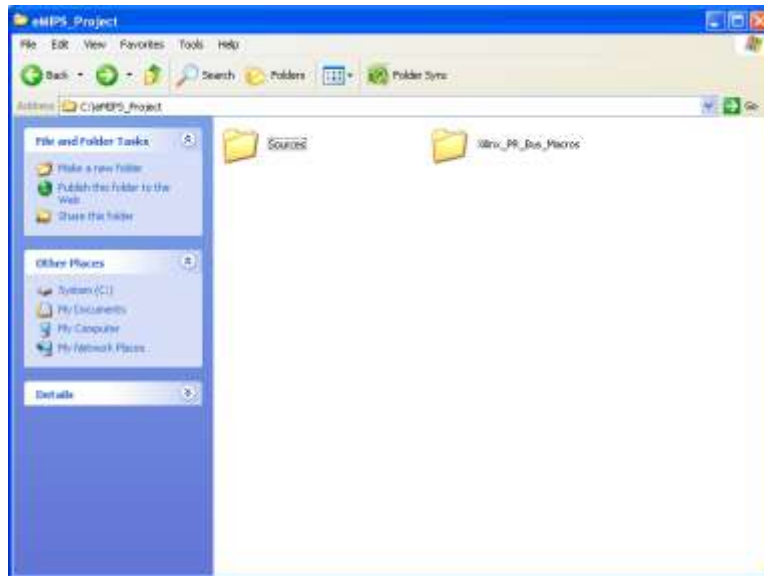
4. Create a 'Sources' folder inside this project directory.



5. Place all the source files from the file package in the 'Sources' folder in the project directory.



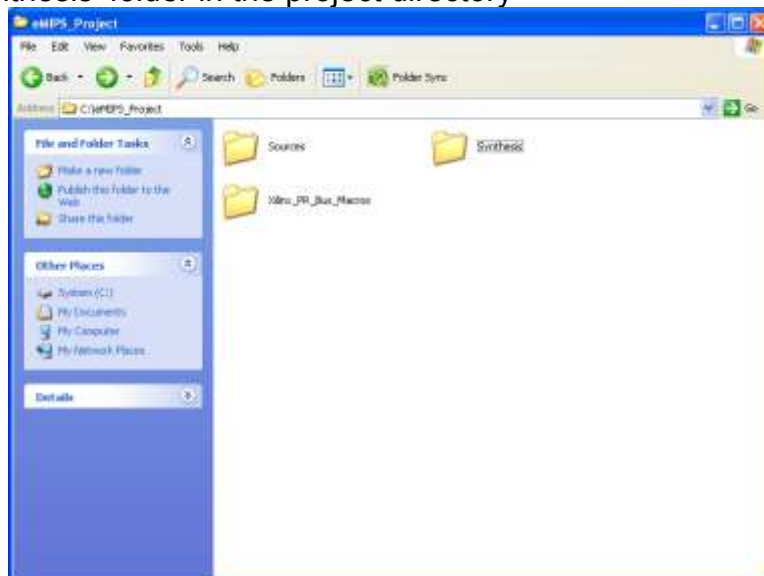
6. Create a folder in the project directory called 'Xilinx_PR_Bus_Macros' for the Bus Macro files.



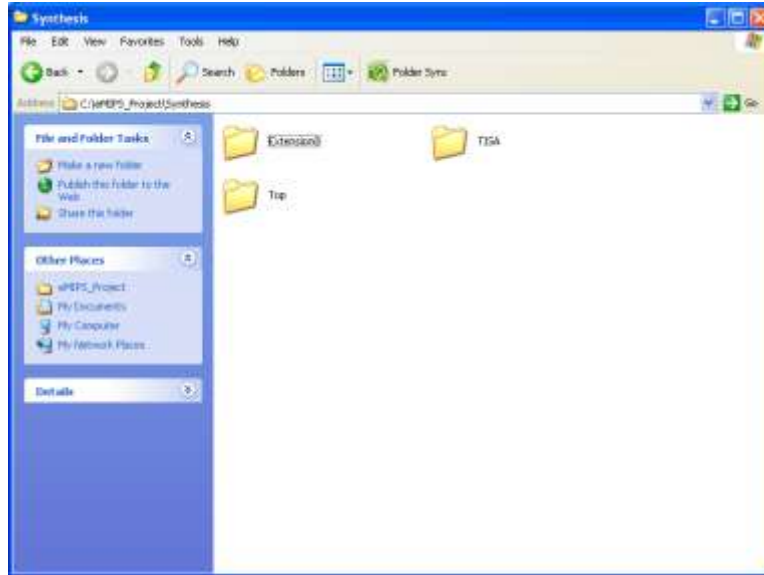
7. Obtain the Bus Macro files from the Xilinx Partial Reconfiguration Early Access Lounge and place them in the 'Xilinx_PR_Bus_Macros' folder inside the project directory.



8. Create a 'Synthesis' folder in the project directory

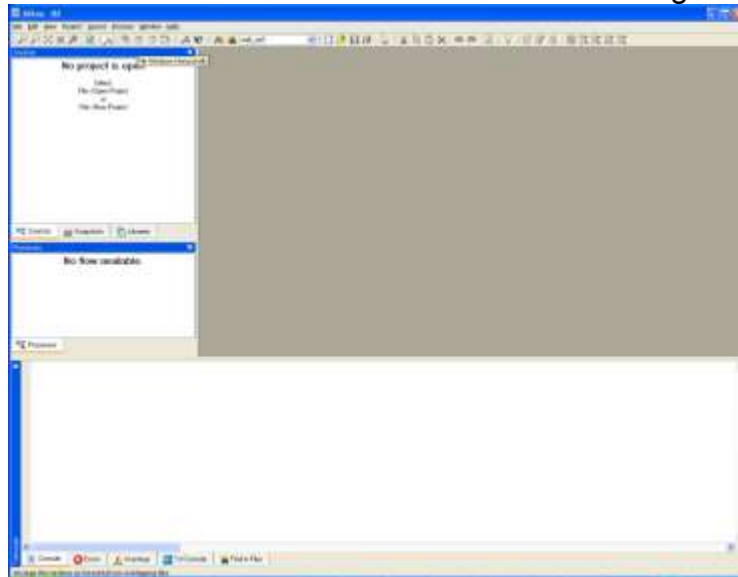


9. Within the 'Synthesis' folder, create three subdirectories: Top, TISA and Extension0.

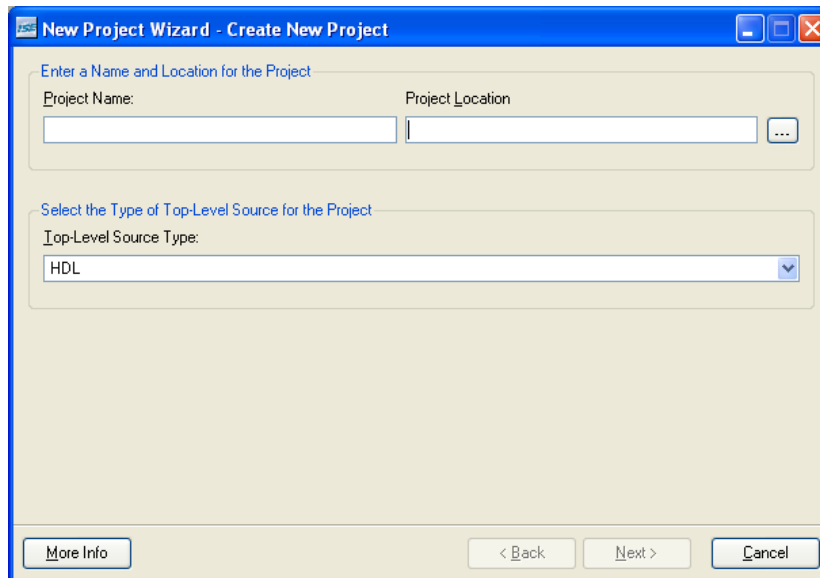


3.1.4 Synthesize the Top Level Module

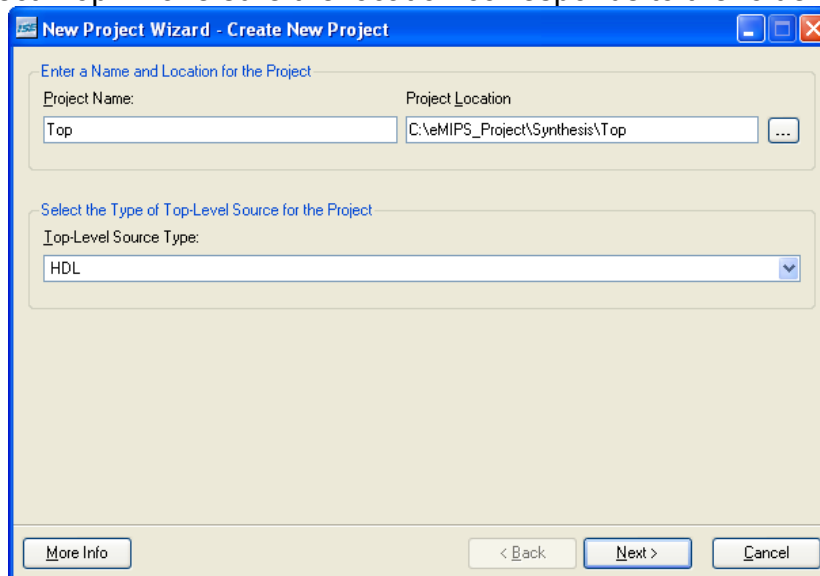
1. Start the install of the Xilinx ISE with the Xilinx Partial Reconfiguration Tools Overlay.



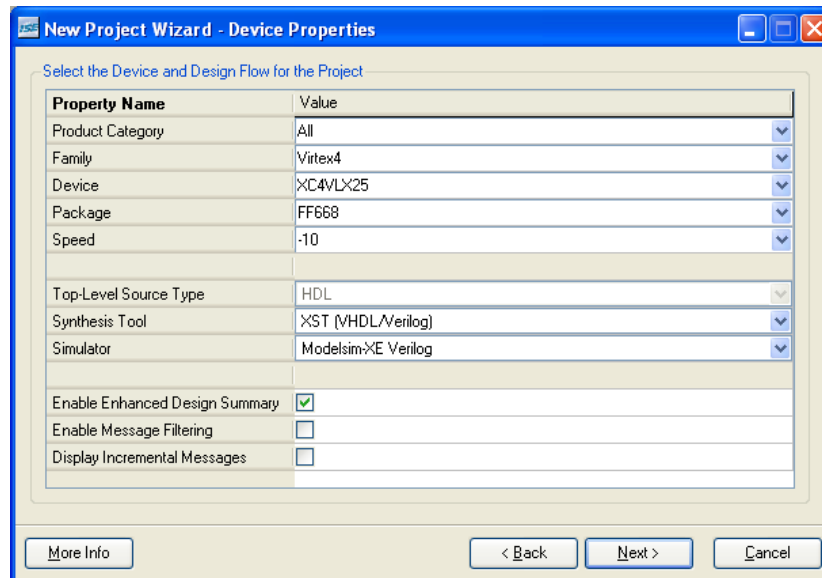
2. Create a new ISE project.
 - a. Click File->New Project in the Menu.
 - b. The 'New Project Wizard' dialog window will appear.



- c. Select the Top Level Synthesis folder as the project location and name the project 'Top'. Make sure the location corresponds to the folder created above.



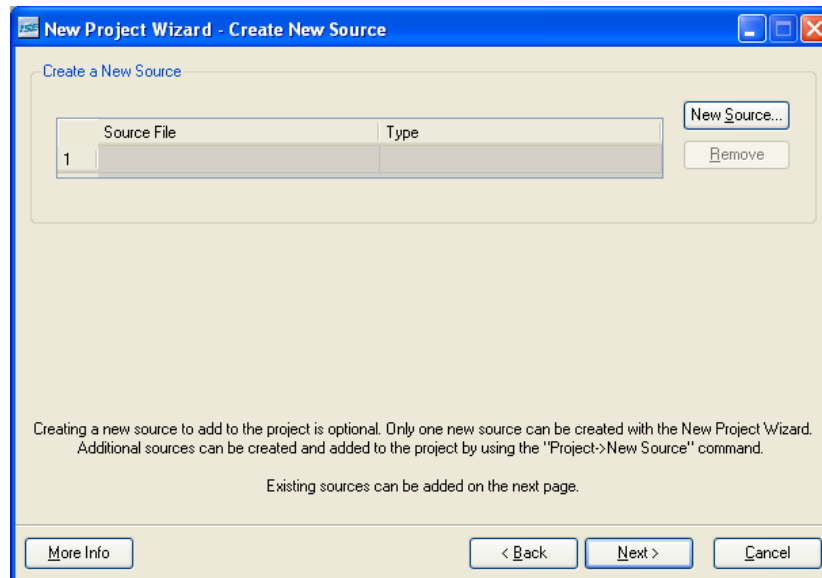
- d. After you have reviewed the entries in the window, click 'Next' continue.
e. The 'Device Properties' screen should appear.



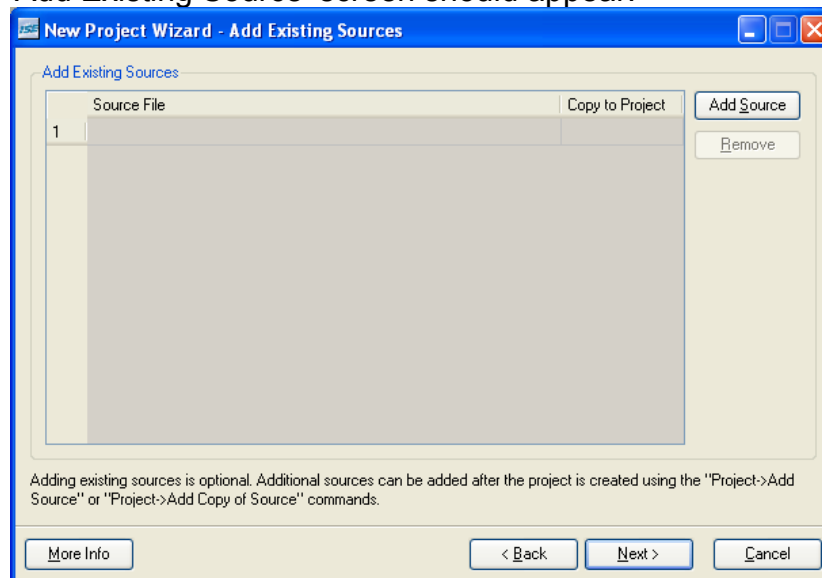
- f. Enter the appropriate settings for your board setup in these fields. For the Xilinx ML401 board, please use the settings in the table below. You may use other synthesis tools other than the XST (Xilinx default) if you chose. However, be aware all of our experiments have used this tool and we cannot vouch for the outcome of another tool.

Property Name	Value
Product Category	All
Family	Virtex4
Device	XC4VLX25
Package	FF668
Speed	-10
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	NA
Enable Enhanced Design Summary	NA
Enable Message Filtering	NA
Display Incremental Messages	NA

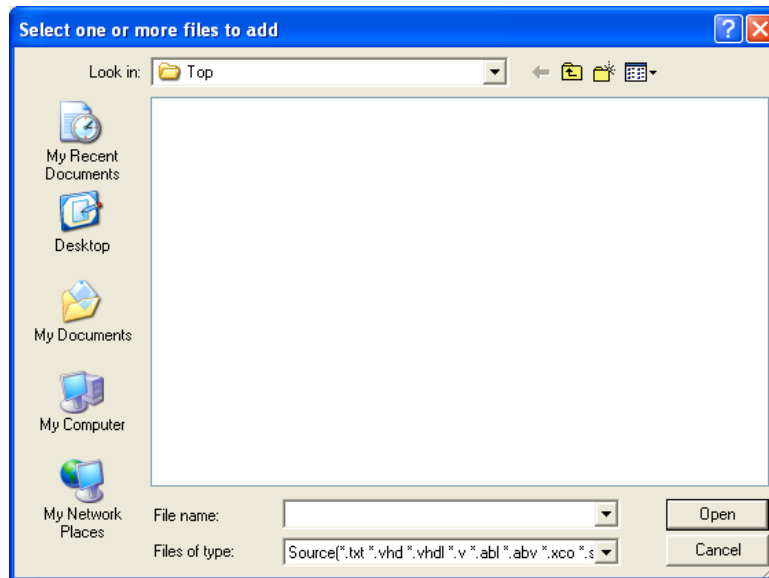
- g. After you have reviewed the entries in the window, click 'Next' to continue.
h. The 'Create New Source' screen should appear.



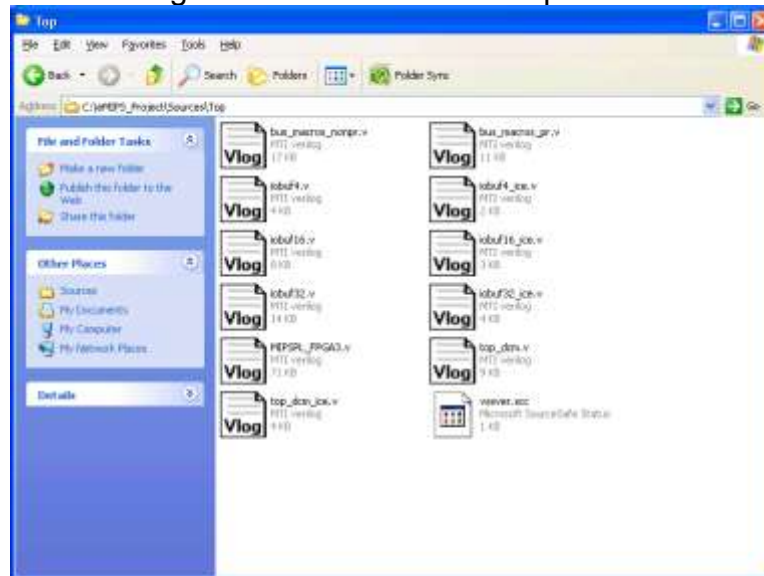
- i. Since you will not be creating any new sources at this time, click 'Next' to continue.
- j. The 'Add Existing Source' screen should appear.



- k. Click the 'Add Source' Button.
- l. The 'Select one or more files to add' dialog should appear.



m. Navigate the dialog window to the eMIPS Top source directory.

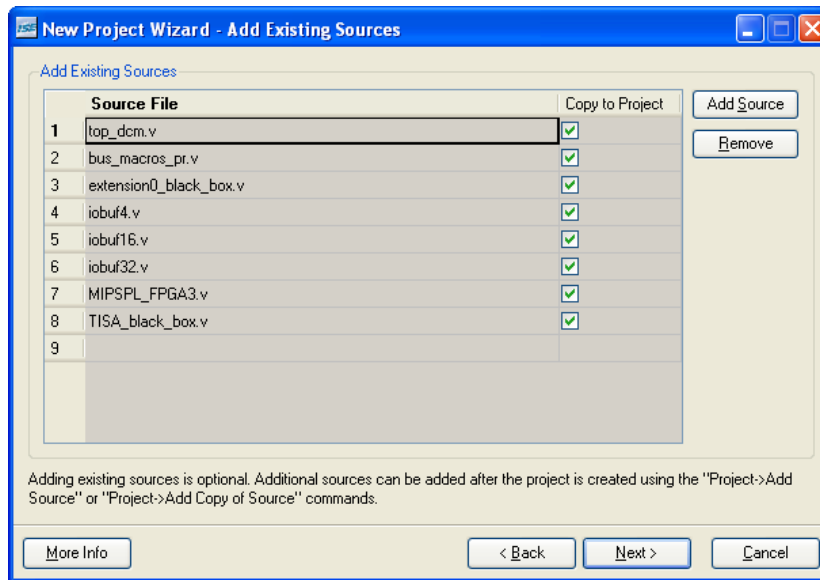


n. Select the files listed in the table below to be added to the project. There are eight files total.

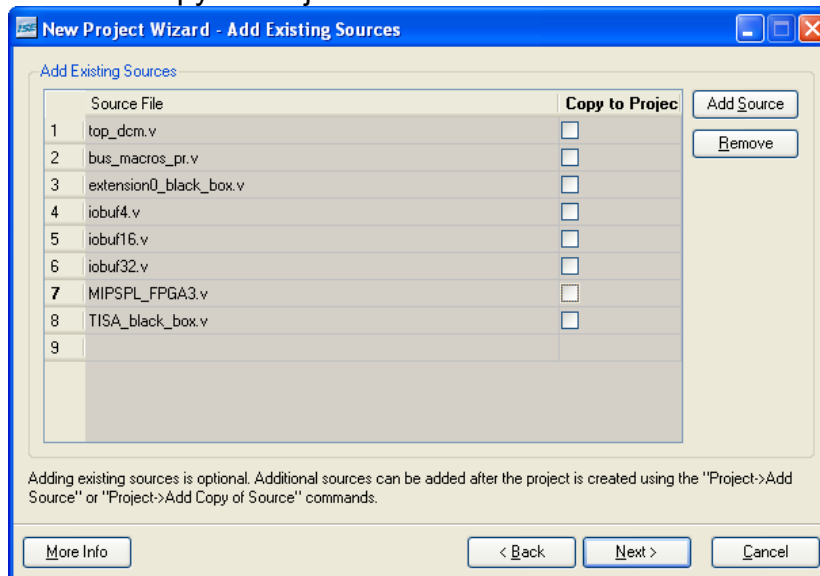
extension0_black_box.v	lobuf4.v	lobuf16.v
lobuf32.v	MIPSPL_FPGA3.v	top_dcm.v
bus_macros_pr.v	TISA_black_box.v	

o. When you are done click 'Open'.

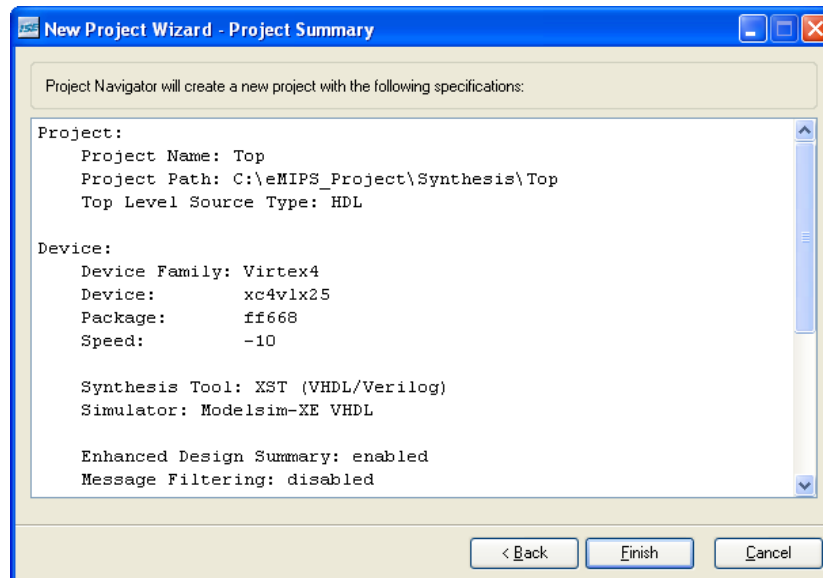
p. The files you selected should be listed in the 'Add Existing Source' screen.



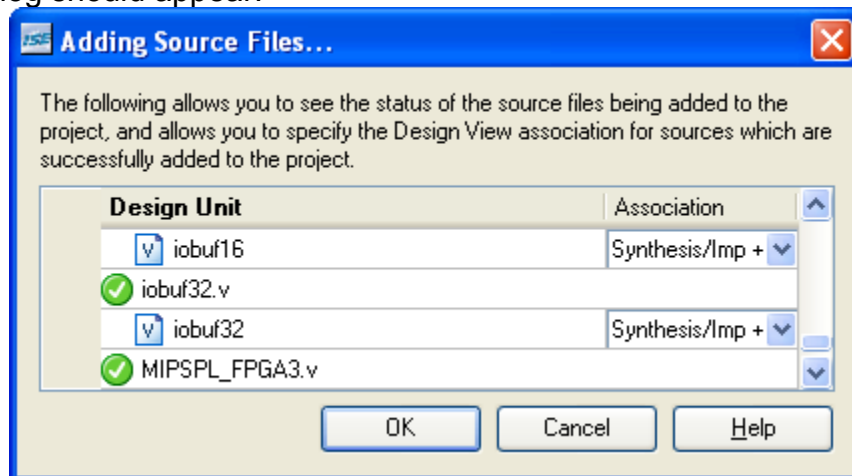
- q. Uncheck the 'Copy to Project' checkbox for each file.



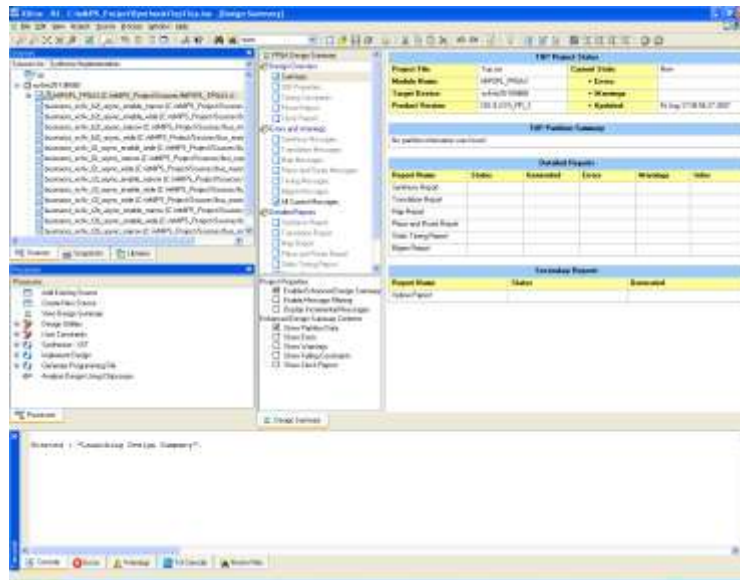
- r. After you have reviewed the entries in the window, click 'Next' continue.
s. The 'Project Summary' screen should appear.



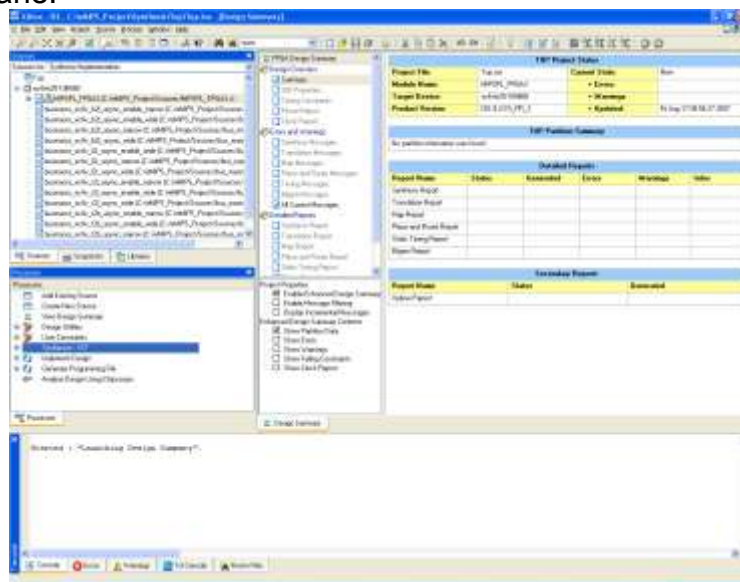
- t. Click finish to complete the project setup.
- u. The 'New Project Wizard' will close and a moment later the 'Adding Source Files' dialog should appear.



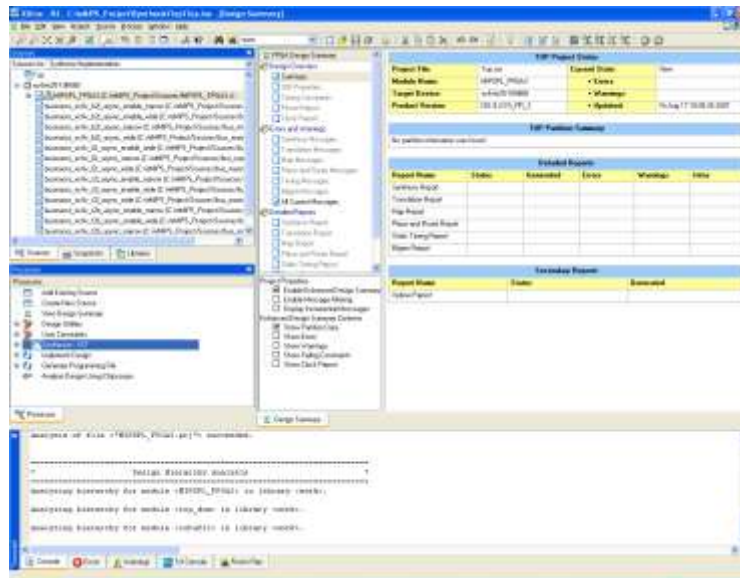
- v. Click 'OK' to continue.
- w. The new project will open in the Xilinx ISE Project Navigator.



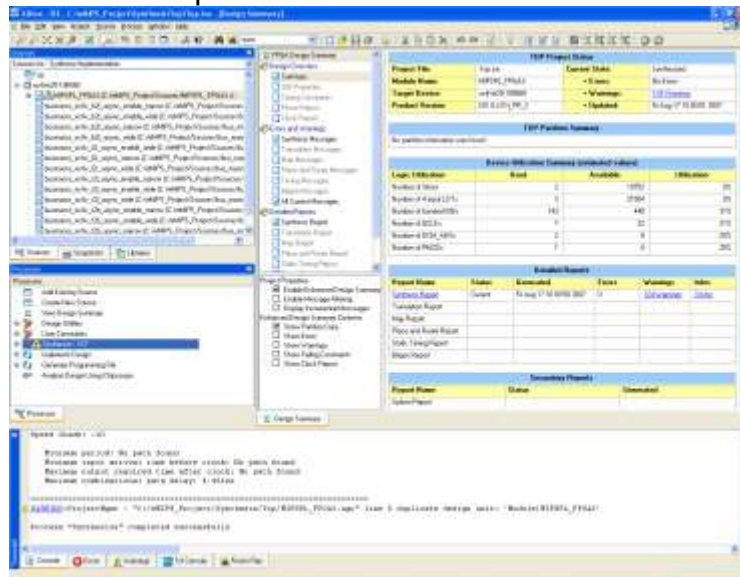
3. The module 'MIPSPL_FPGA3' should be set as the top module in the 'Sources' pane. If not set it as the top module by right-clicking MIPSPL_FPGA3 and selecting 'Set as Top Module'.
4. In the 'Sources' Pane, select 'MIPSPL_FPGA3'. Then select 'Synthesize - XST' in the 'Processes' pane.



5. You may double-click on 'Synthesize - XST' or you may right click and select 'Run' to start the synthesis process.

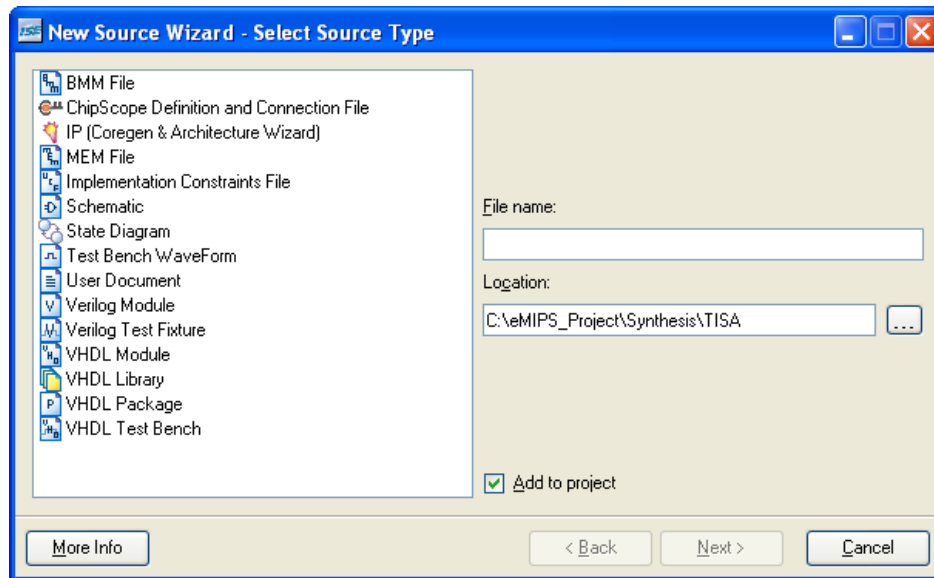


6. You should see a blue spinning icon appear by the 'Synthesize - XST'. As the process is performed some warnings and info messages will be displayed. Most of these are expected.
7. Wait for the process to complete.

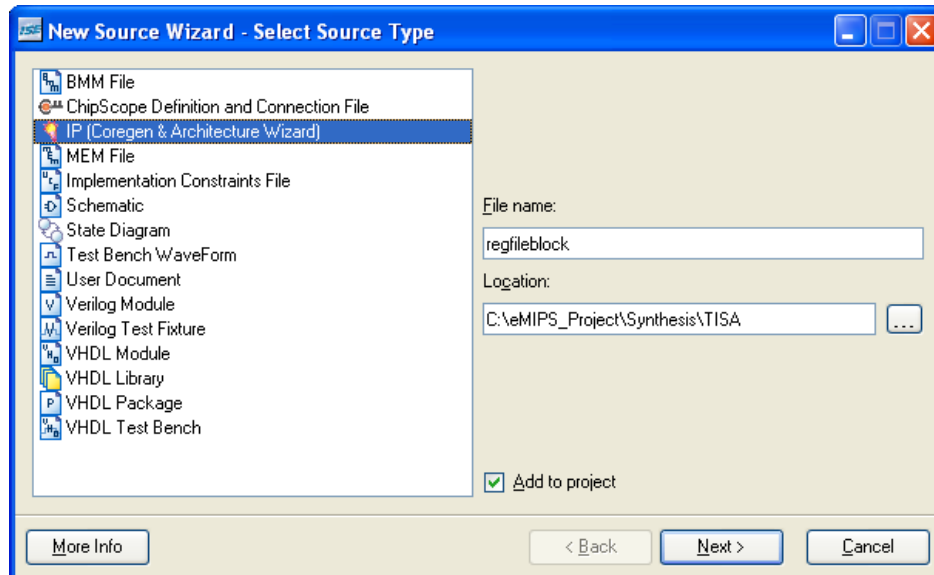


3.1.5 Add the Register File Blockram

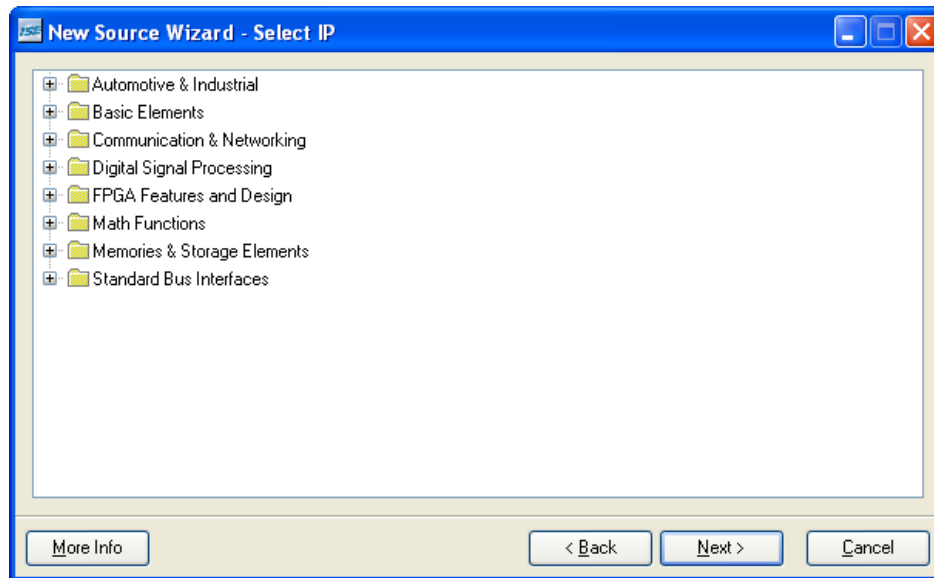
1. Start inside your current Xilinx ISE Project.
2. In the 'Sources' pane, right-click and select 'New Source'.
3. The 'New Source Wizard' dialog window should appear.



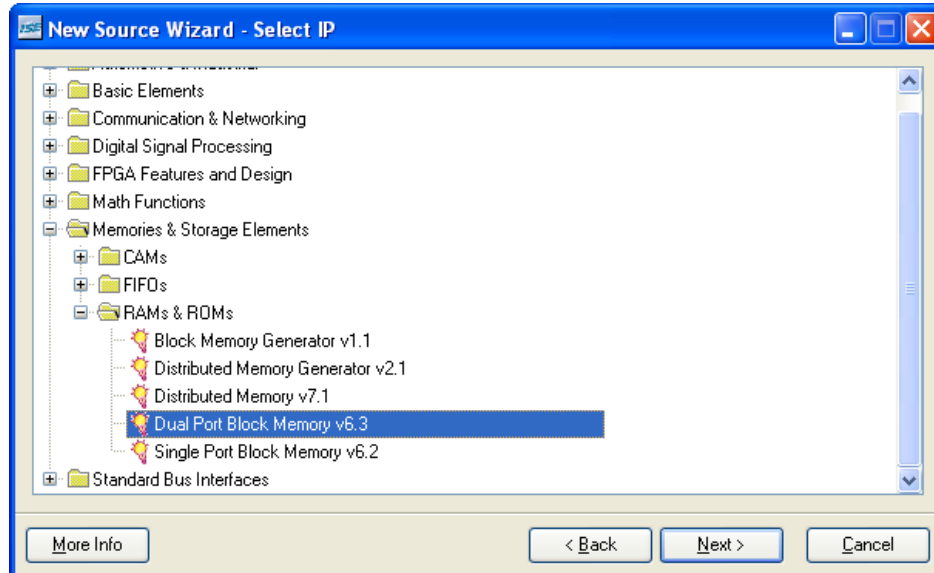
4. Select 'IP (Corgen & Architecture Wizard)' from the pane on the left and enter the file name 'regfileblock'. The Location box should indicate the location of your Xilinx ISE Project. In this example it is in the TISA synthesis folder of the eMIPS Project Directory.



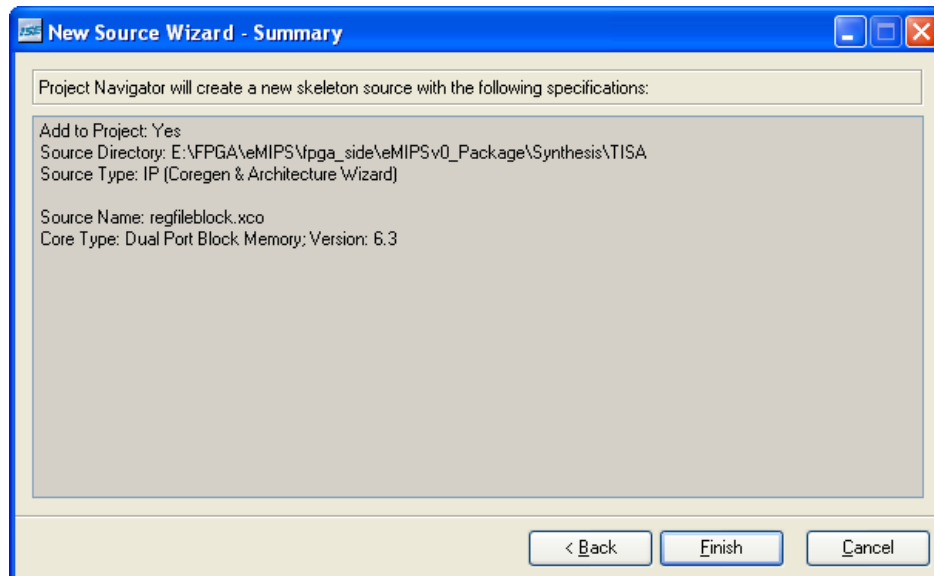
5. After you have reviewed the entries in the window, click 'Next' to continue.
6. The 'Select IP' screen should appear.



7. Select 'Memories & Storage Elements\RAMs & ROMs\Dual Port Block Memory vX.X'.



8. Click 'Next' to continue.
9. The 'Summary' screen should appear. Click 'Finish' to continue.



10. The 'Dual Port Block Memory' configuration dialog should appear. This can take some time.



11. Enter the properties in the table below into the window.

Component Name	Regfileblock
Memory Size	
Width A	32
Width B	32
Depth A	32
Depth B	32
Port A Options	
Configuration	Read Only

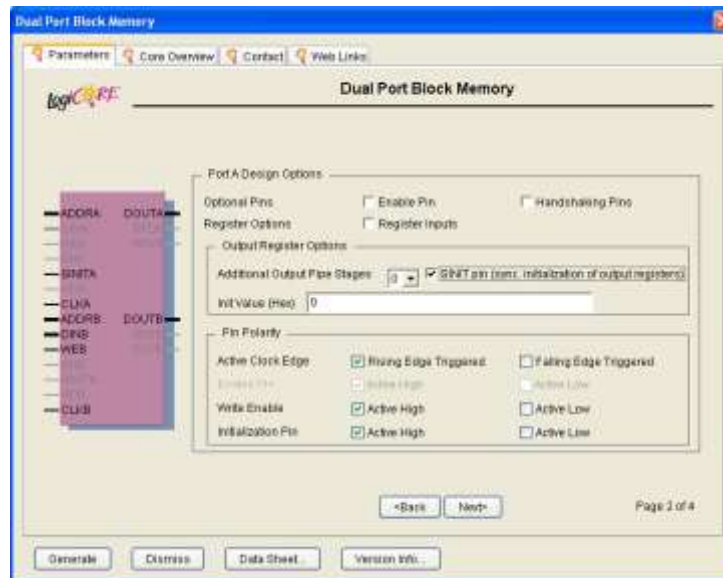
Write Mode	NA
Port B Options	
Configuration	Read And Write
Write Mode	Read Before Write

12. After you have reviewed the entries in the window, click 'Next' to continue.

13. The next screen in the configuration dialog should appear.

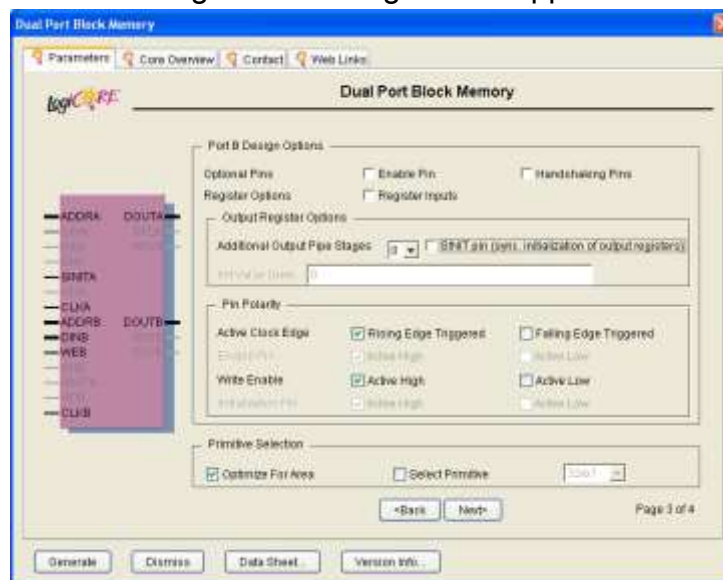
14. Enter the properties in the table below into the window.

Port A Design Options – Output Register Options	
SINIT pin (sync. Initialization of output registers)	Yes



15. After you have reviewed the entries in the window, click 'Next' to continue.

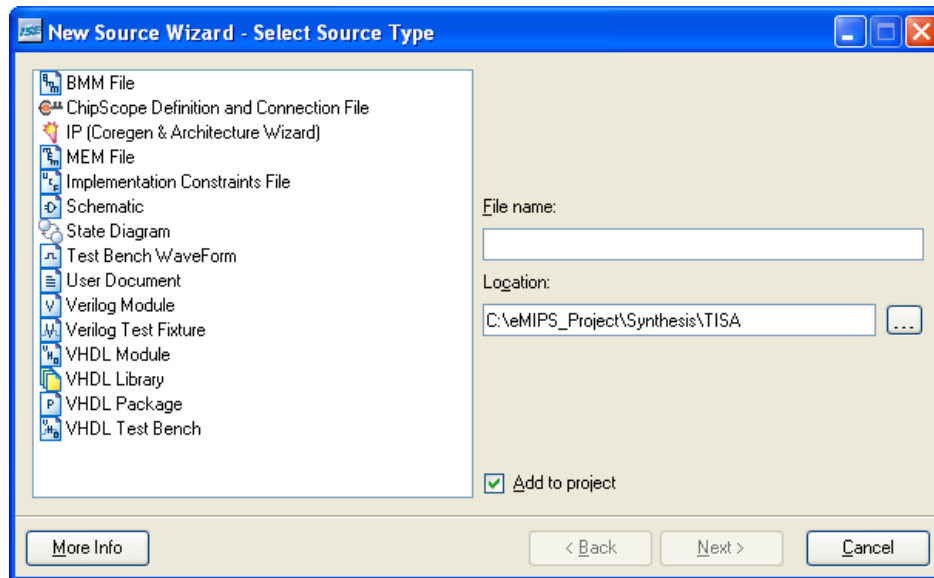
16. The next screen in the configuration dialog should appear.



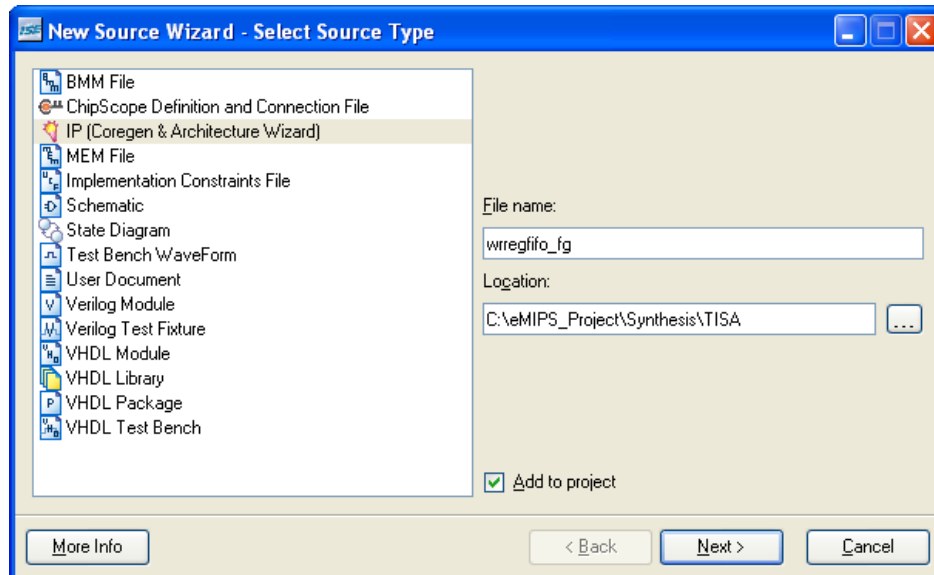
17. Enter the properties in the table below into the window.

Port B Design Options – Output Register Options	
SINIT pin (sync. Initialization of output registers)	Yes

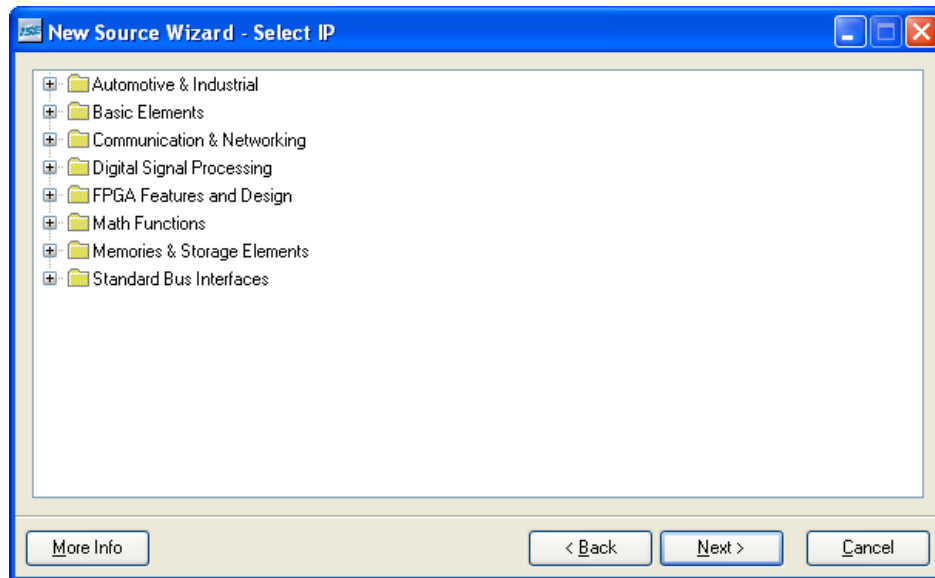




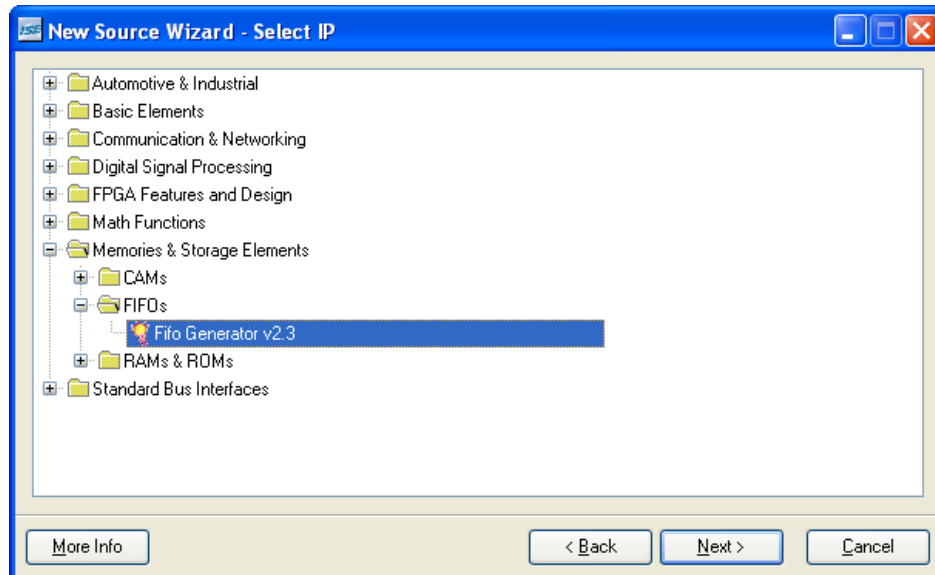
4. Select 'IP (Corgen & Architecture Wizard)' from the pane on the left and enter the file name 'wregfifo_fg'. The Location box should indicate the location of your Xilinx ISE Project. In this example it is in the TISA synthesis folder of the eMIPS Project Directory.



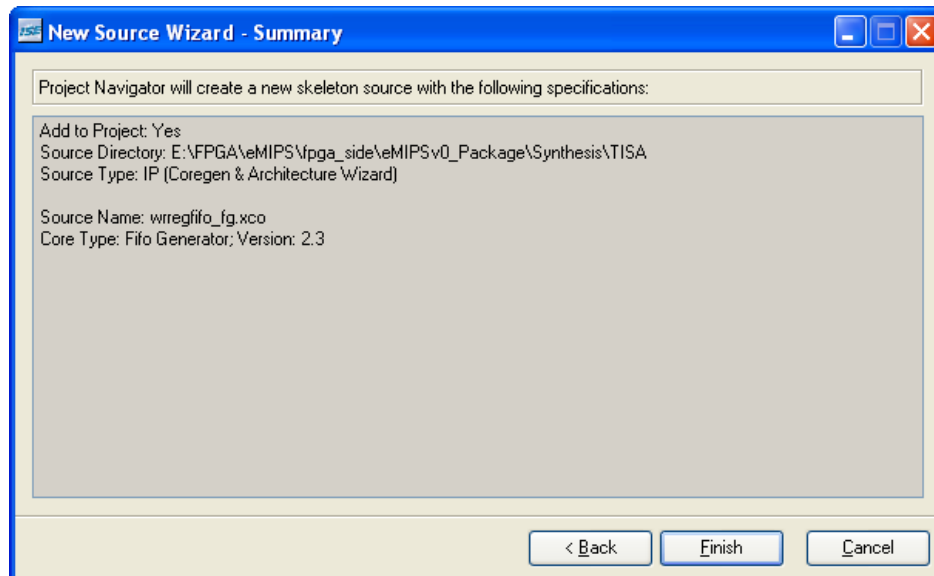
5. After you have reviewed the entries in the window, click 'Next' to continue.
6. The 'Select IP' screen should appear.



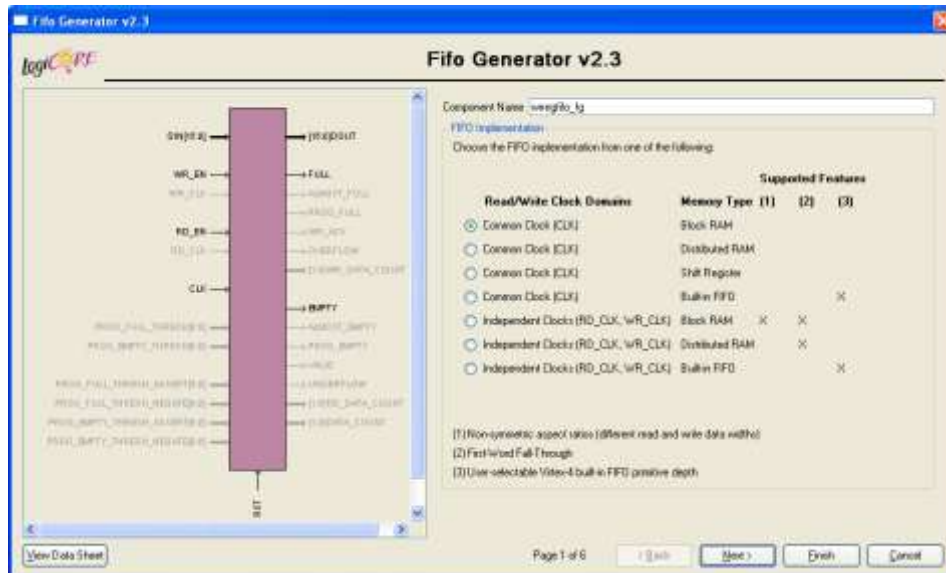
7. Select 'Memories & Storage Elements\FIFOs\Fifo Generator vX.X'.



8. Click 'Next' to continue.
9. The 'Summary' screen should appear. Click 'Finish' to continue.

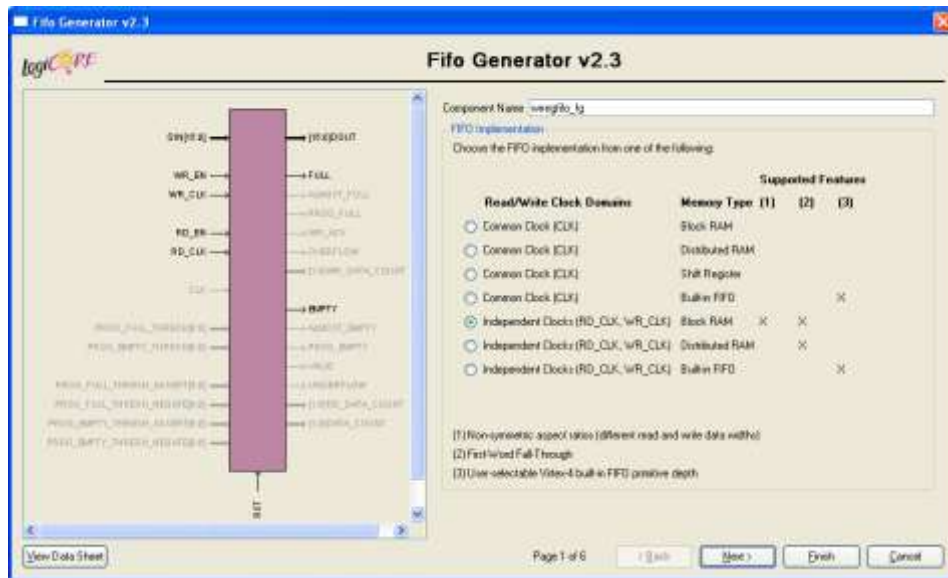


10. The 'Fifo Generator' configuration dialog should appear. This may take some time.



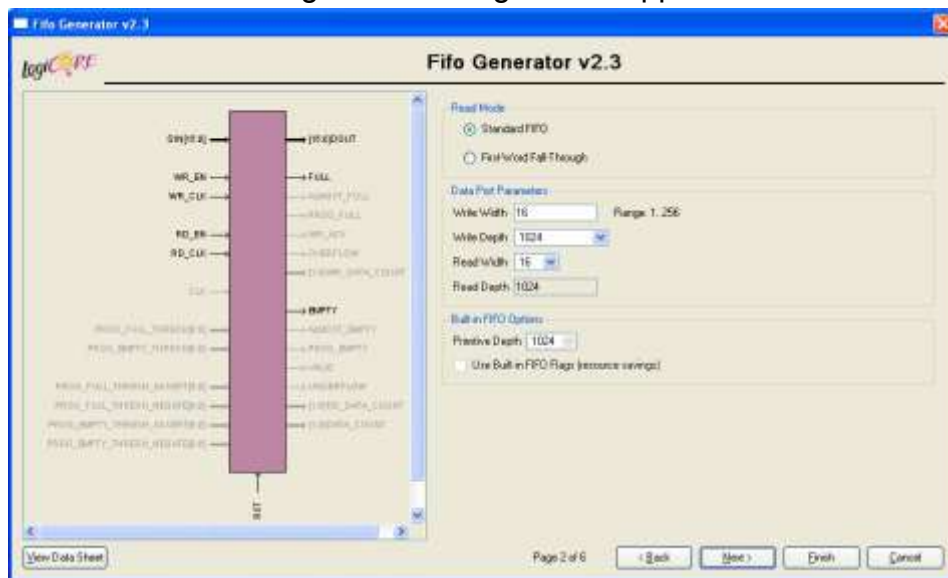
11. Enter the properties in the table below into the window.

Component Name	wrregfifo_fg
FIFO Implementation	
Read/Write Clock Domains	Independent Clocks
Memory Type	Block RAM



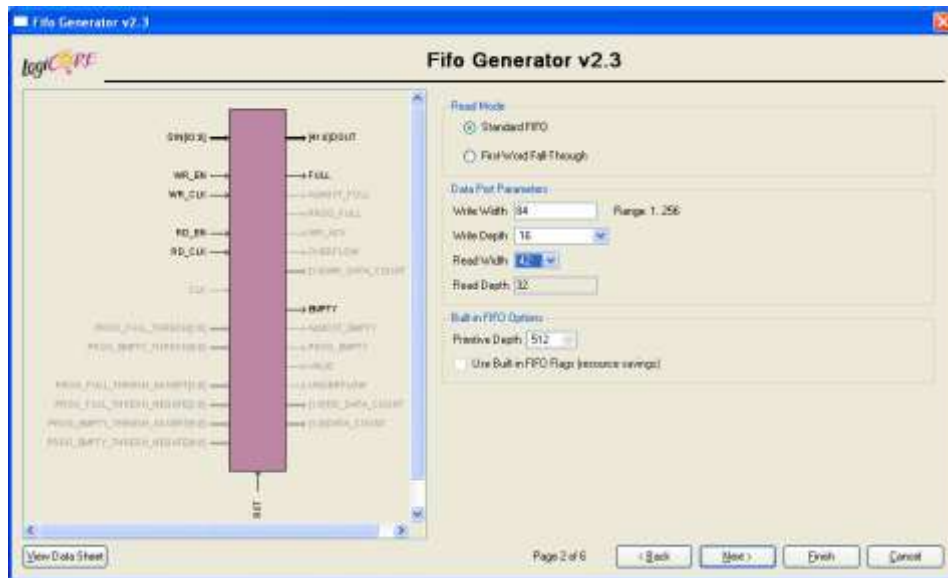
12. After you have reviewed the entries in the window, click 'Next' to continue.

13. The next screen in the configuration dialog should appear.



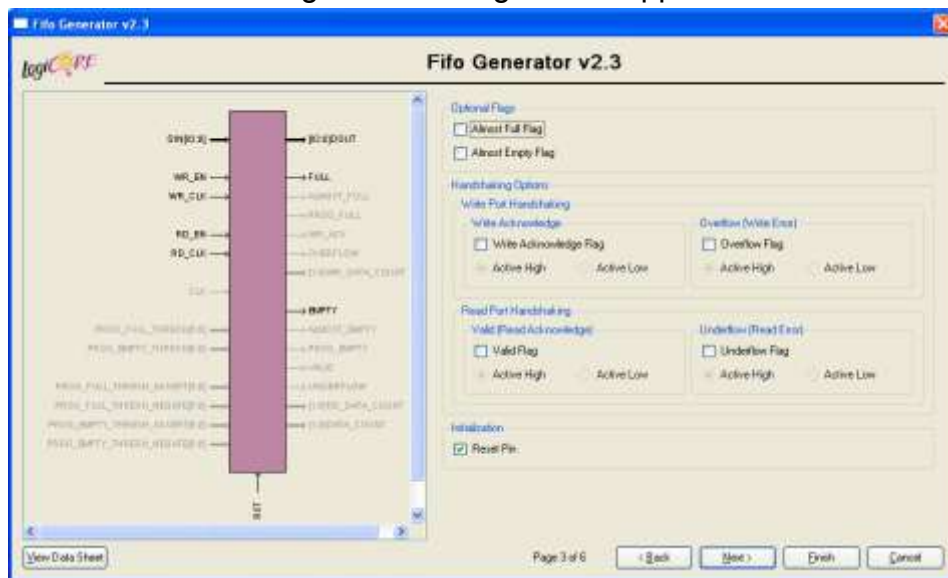
14. Enter the properties in the table below into the window.

Read Mode	Standard FIFO
Data Port Parameters	
Write Width	84
Write Depth	16
Read Width	42
Read Depth	32



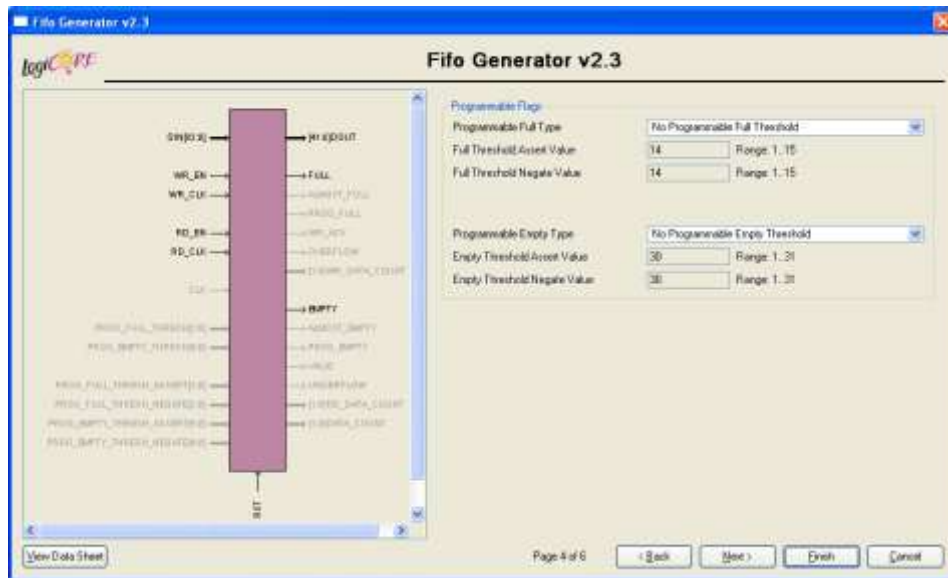
15. After you have reviewed the entries in the window, click 'Next' to continue.

16. The next screen in the configuration dialog should appear.



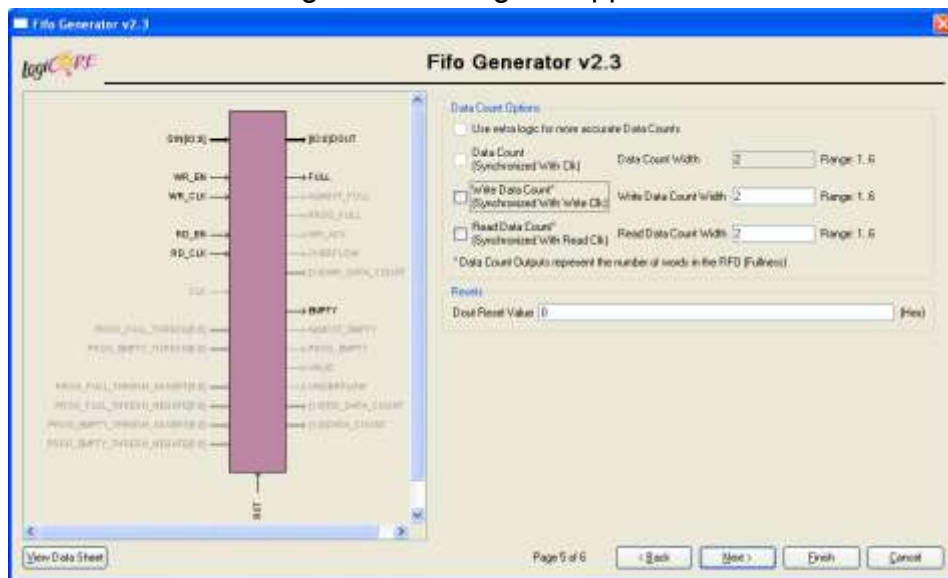
17. Click 'Next' to Continue.

18. The next screen in the configuration dialog will appear.



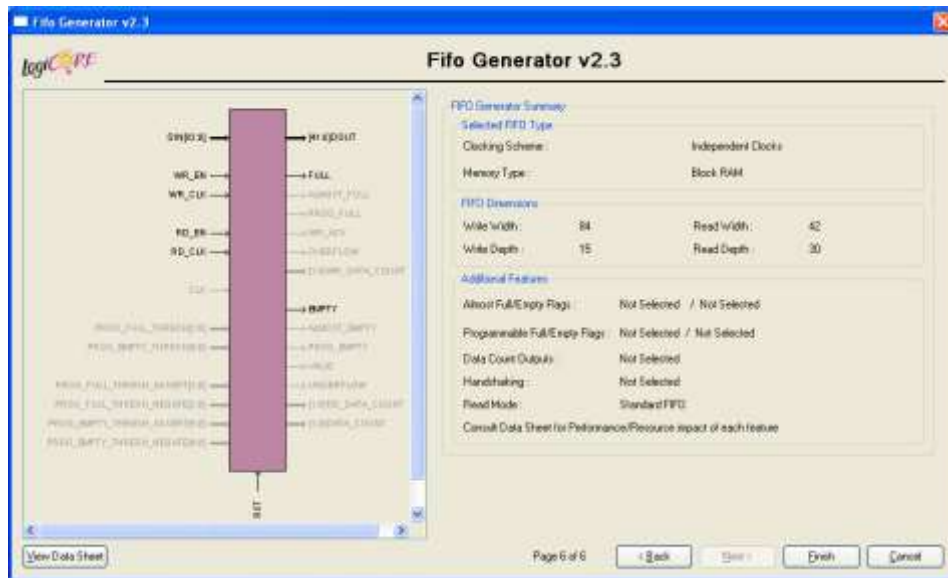
19. Click 'Next' to Continue.

20. The next screen in the configuration dialog will appear.



21. Click 'Next' to Continue.

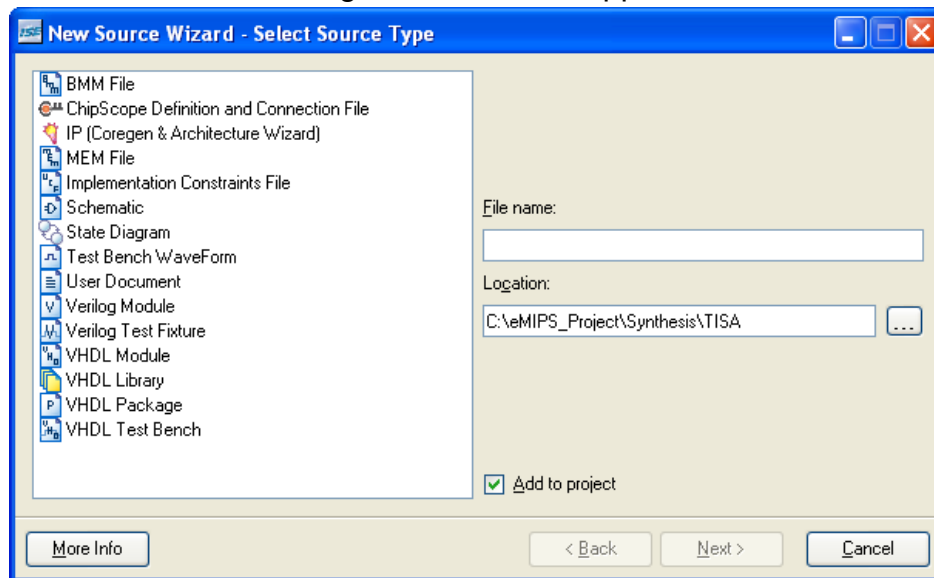
22. The next screen in the configuration dialog should appear.



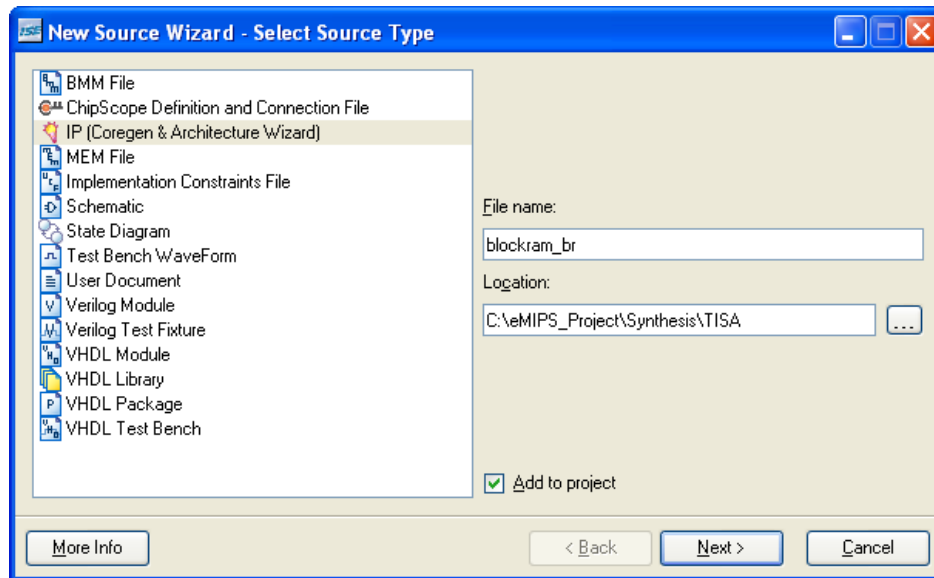
23. Click 'Finish' to complete generating the core.

3.1.7 Add the Bootloader Blockram

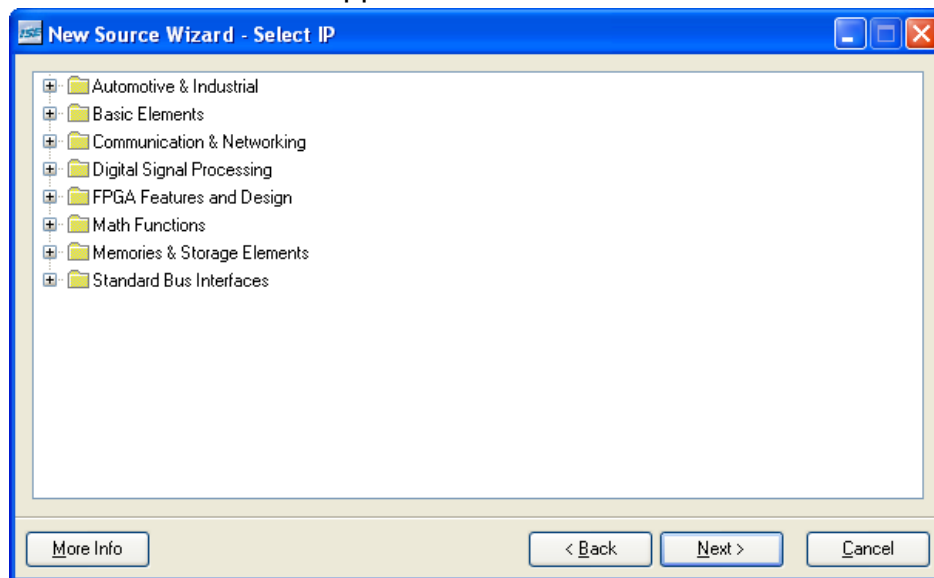
1. Start inside a Xilinx ISE Project.
2. In the 'Sources' pane, right-click and select 'New Source'.
3. The 'New Source Wizard' dialog window should appear.



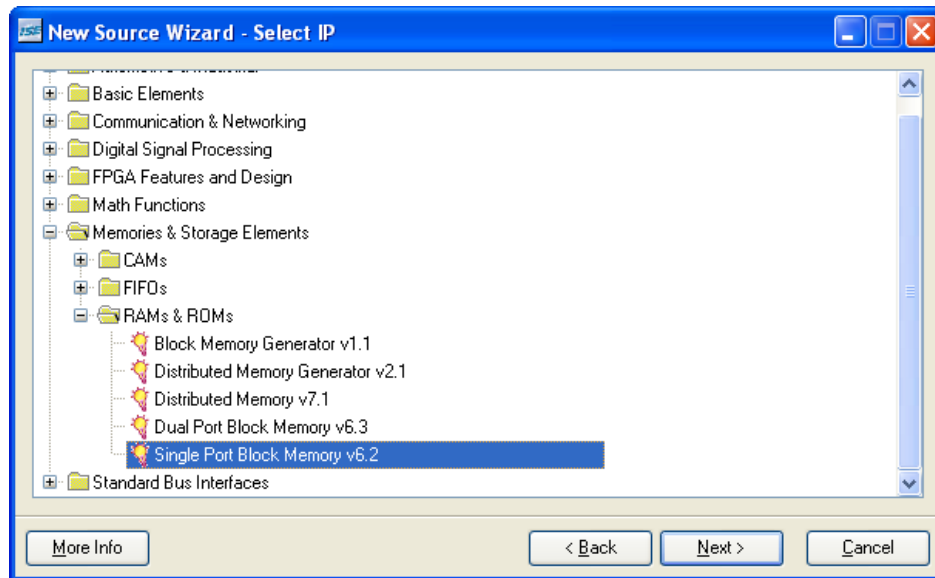
4. Select 'IP (Corgen & Architecture Wizard)' from the pane on the left and enter the file name 'blockram_br'. The Location box should indicate the location of your Xilinx ISE Project. In this example it is in the TISA synthesis folder of the eMIPS Project Directory.



5. After you have reviewed the entries in the window, click 'Next' to continue.
6. The 'Select IP' screen should appear.

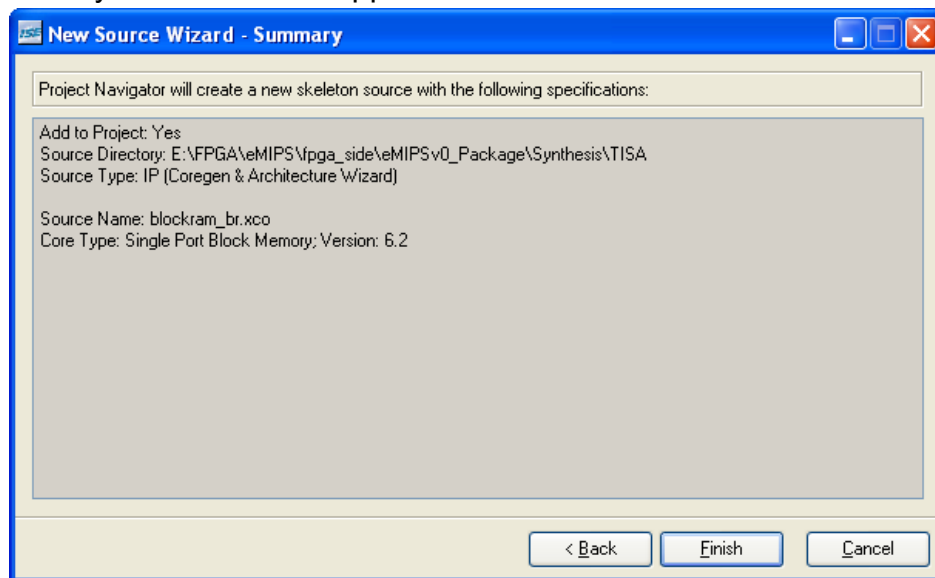


7. Select 'Memories & Storage Elements\RAMs & ROMs\Single Port Block Memory vX.X'.

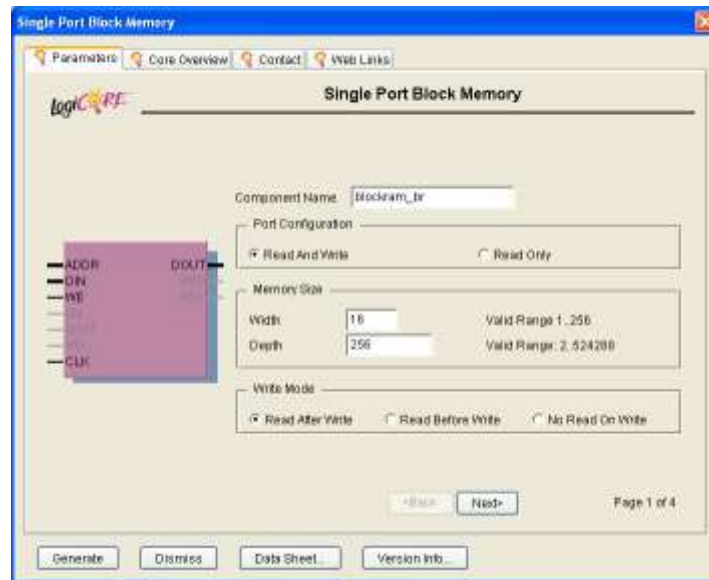


8. Click 'Next' to continue.

9. The 'Summary' screen should appear. Click 'Finish' to continue.

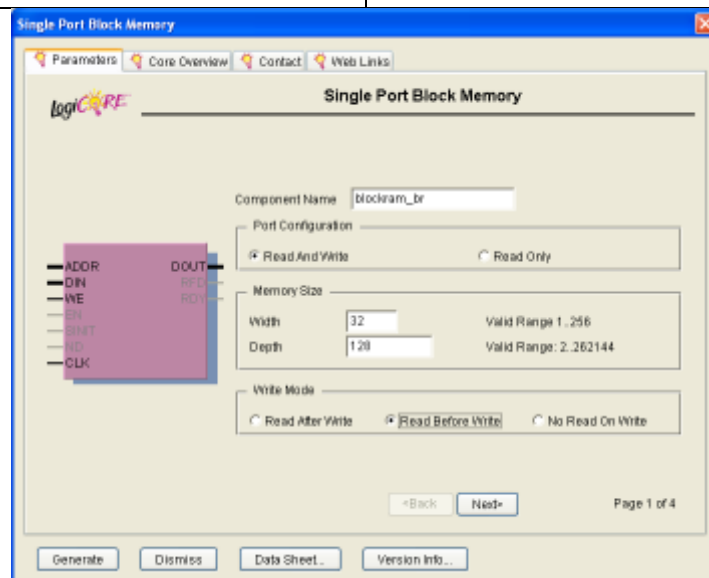


10. The 'Single Port Block Memory' configuration dialog should appear.



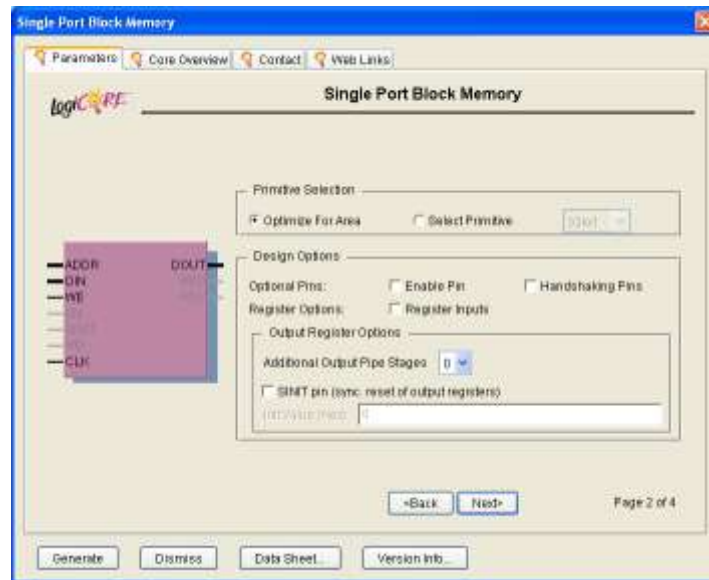
11. Enter the properties in the table below into the window.

Component Name	blockram_br
Memory Size	
Width	32
Depth	128
Write Mode	Read Before Write



12. After you have reviewed the entries in the window, click 'Next' to continue.

13. The next screen in the configuration dialog should appear.



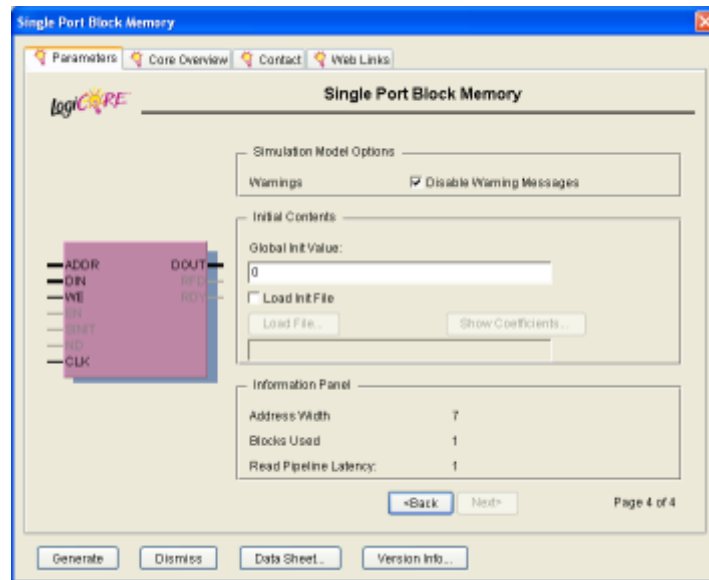
14. Click 'Next' to continue.

15. The next screen in the configuration dialog should appear.



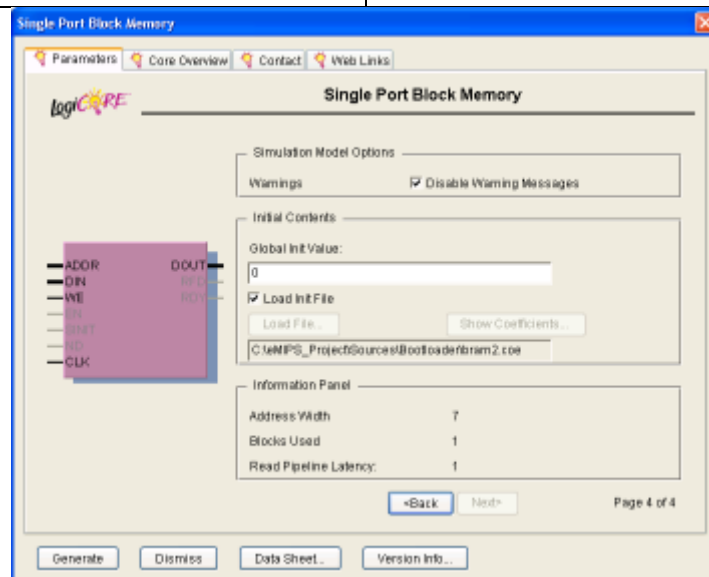
16. Click 'Next' to continue.

17. The next screen in the configuration dialog should appear.



18. Enter the properties in the table below into the window.

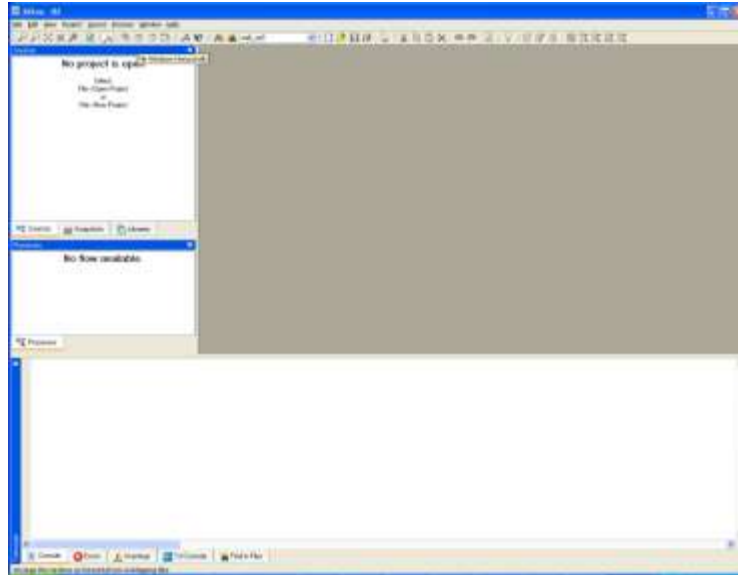
Initial Contents	
Load Init File	Yes
	<location of sources>\bram2.coe



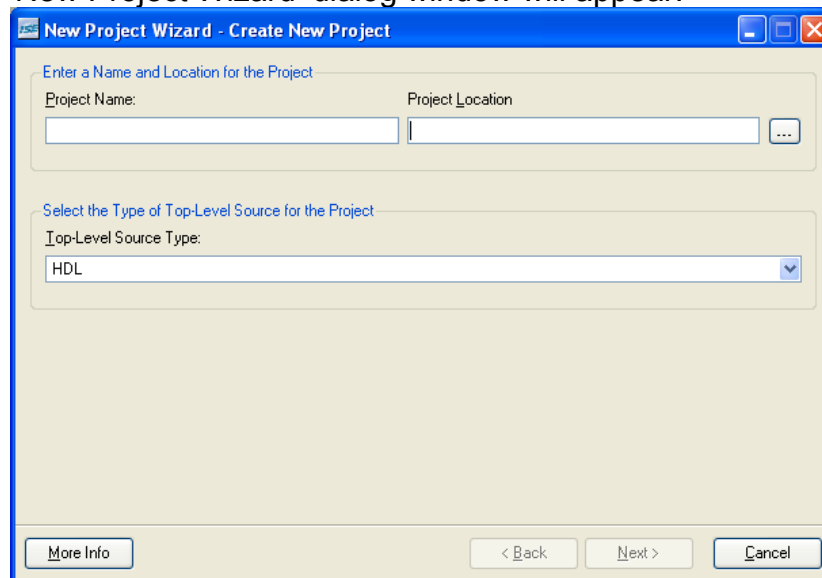
19. After you have reviewed the entries in the window, click 'Generate' to complete generating the core.

3.1.8 Synthesize the Base Design (TISA)

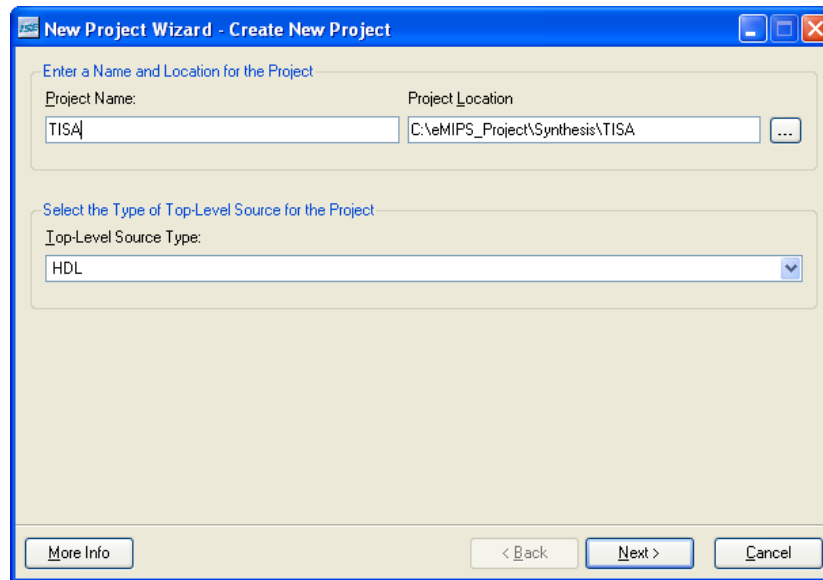
1. Start the install of the Xilinx ISE with the Xilinx Partial Reconfiguration Tools Overlay.



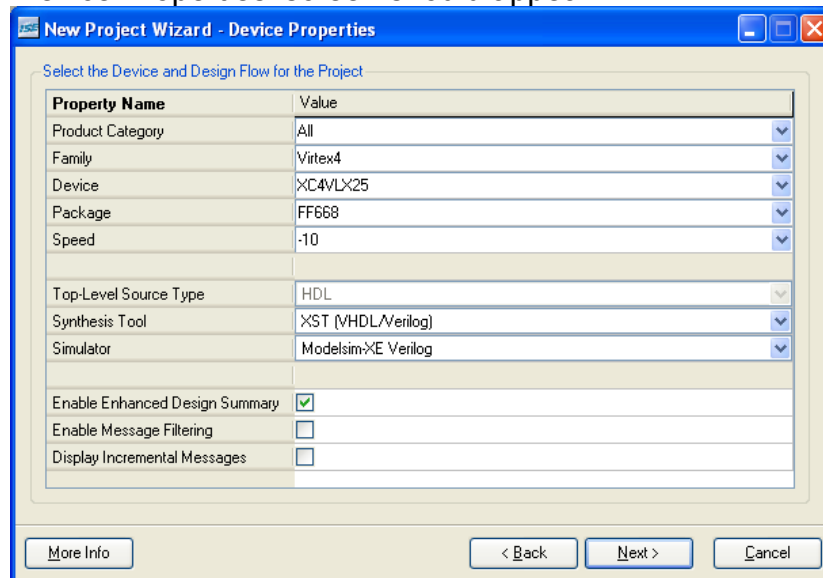
2. Create a new ISE project.
 - a. Click File->New Project in the Menu.
 - b. The 'New Project Wizard' dialog window will appear.



- c. Select the TISA Synthesis folder as the project location and name the project 'TISA'. Make sure the "Project Location" property is as intended.



- d. After you have reviewed the entries in the window, click 'Next' continue.
- e. The 'Device Properties' screen should appear.

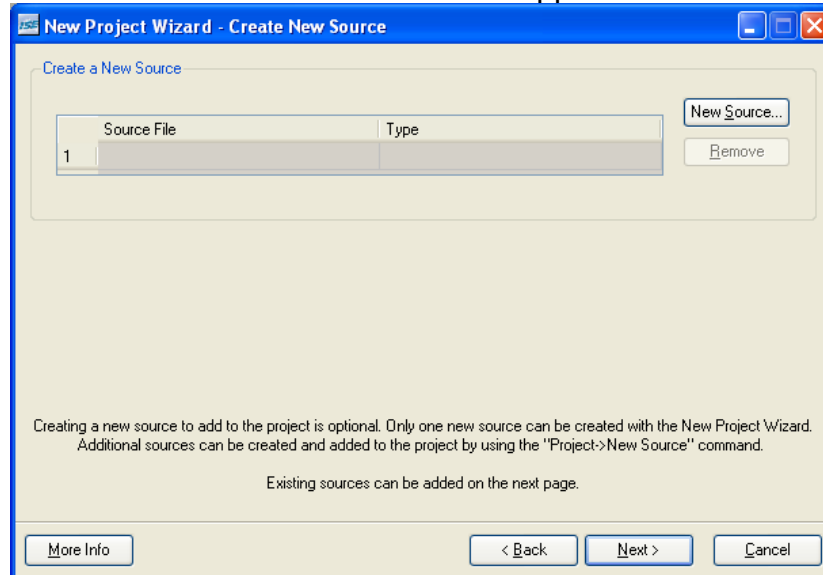


- f. Enter the appropriate settings for your board setup in these fields. For the Xilinx ML401 board, please use the settings in the table below. You may use other synthesis tools other than the XST (Xilinx default) if you chose. However, be aware all of our experiments have used this tool and we cannot vouch for the outcome of another tool.

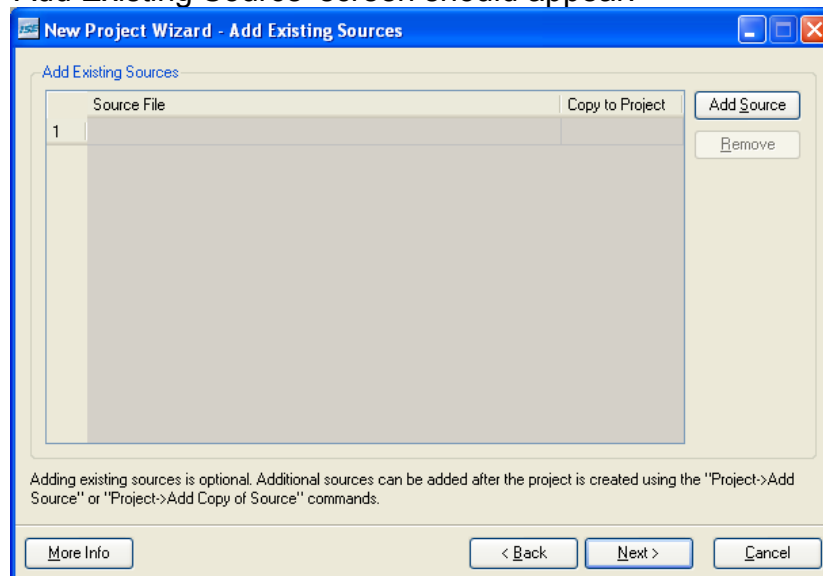
Property Name	Value
Product Category	All
Family	Virtex4
Device	XC4VLX25
Package	FF668
Speed	-10

Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	NA
Enable Enhanced Design Summary	NA
Enable Message Filtering	NA
Display Incremental Messages	NA

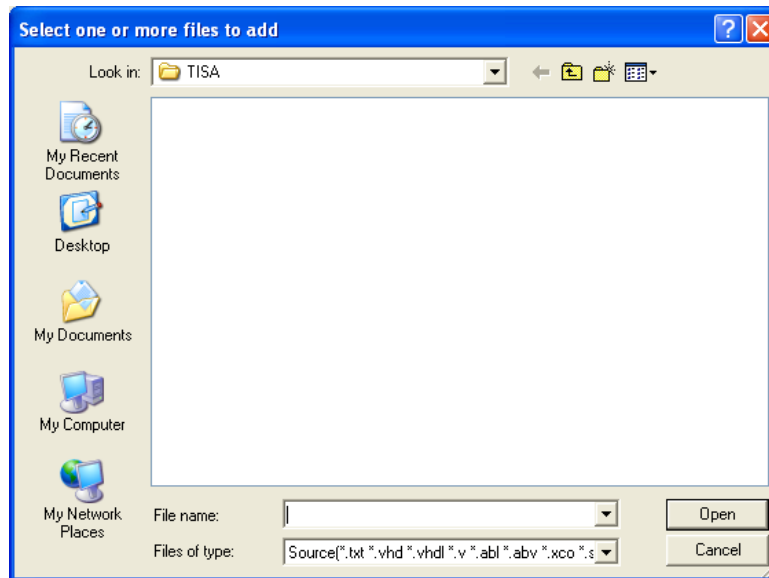
- g. After you have reviewed the entries in the window, click 'Next' continue.
- h. The 'Create New Source' screen should appear.



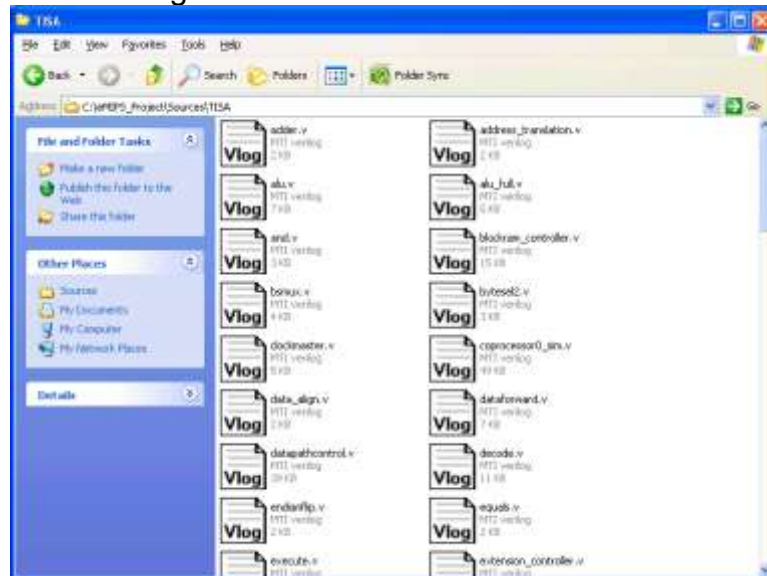
- i. Since you will not be creating any new sources at this time, click 'Next' to continue.
- j. The 'Add Existing Source' screen should appear.



- k. Click the 'Add Source' Button.
- l. The 'Select one or more files to add' dialog should appear.



m. Navigate the dialog window to the eMIPS TISA source directory.

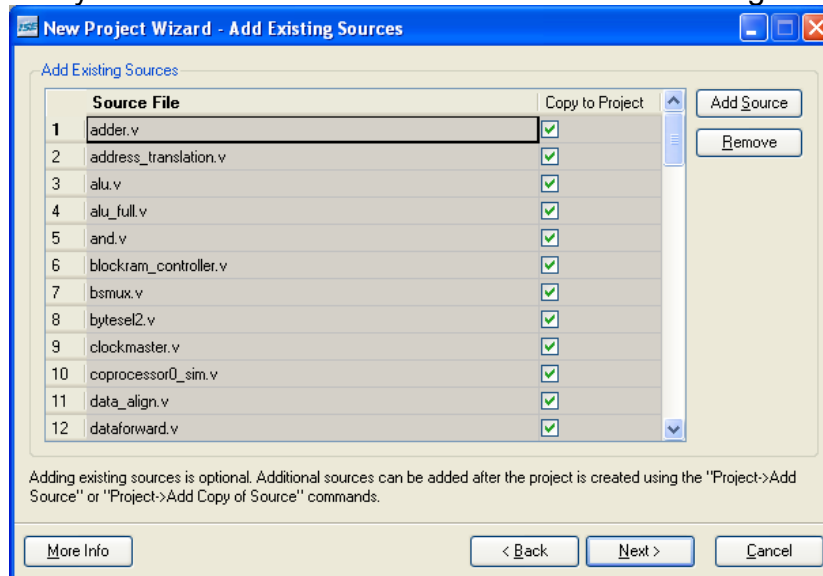


n. Select the files listed in the table below to be added to the project. There are a total of sixty-nine files.

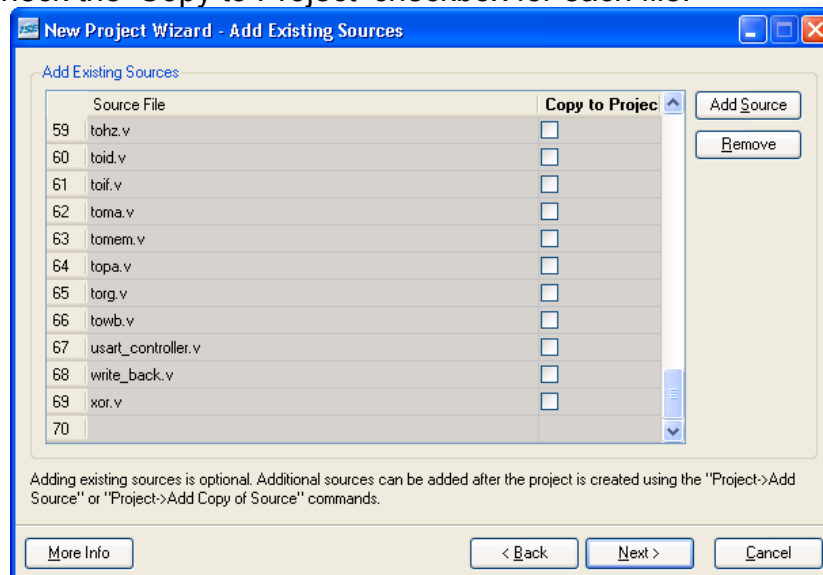
adder.v	address_translation.v	alu.v
alu_full.v	and.v	blockram_controller.v
bsmux.v	bytesel2.v	clockmaster.v
coprocessor0_sim.v	data_align.v	dataforward.v
datapathcontrol.v	decode.v	endianflip.v
equals.v	execute.v	fifo.v
flash_bridge.v	flash_controller.v	flash_interface.v
gpio_controller.v	greaterthan.v	hazard.v
instruction_decode.v	instruction_fetch.v	interrupt_controller.v
lessthan.v	memory_access.v	memory_arbiter.v
memory_bus_front.v	memory_controller.v	memory_interface3.v
multiplier.v	mux.v	or.v

pipeline_arbiter.v	power_management_controller.v	registerblock.v
registerfile4.v	shift2.v	shift.v
signextend16_32.v	sram_bridge.v	sram_controller.v
sram_interface.v	sysace_bridge.v	sysace_controller.v
sysace_interface.v	timer_controller.v	TISA.v
toep0.v	toef.v	toex.v
toext.v	tohz.v	toid.v
toif.v	toma.v	tomem.v
topa.v	torg.v	towb.v
usart_controller.v	write_back.v	xor.v
lock.v	memory_management_unit.v	extension_controller.v

- o. When you are done click 'Open'.
- p. The files you selected should be listed in the 'Add Existing Source' screen.

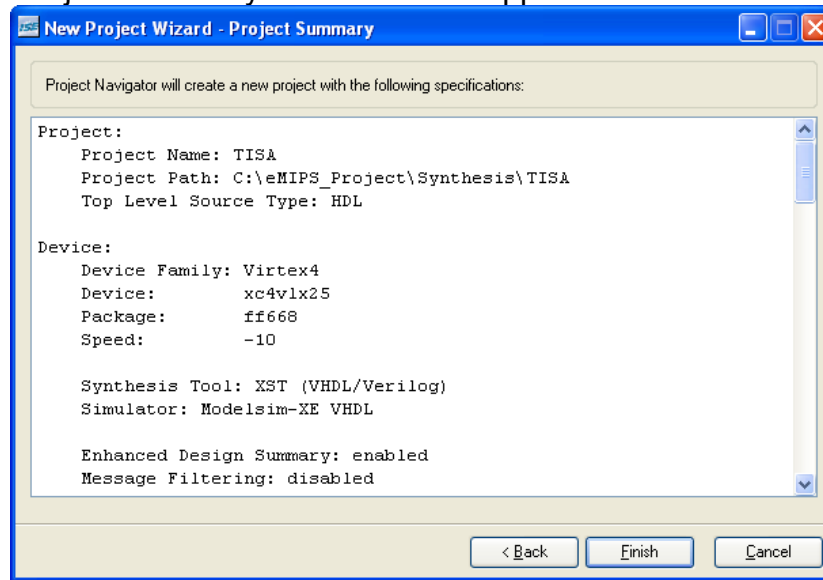


- q. Uncheck the 'Copy to Project' checkbox for each file.

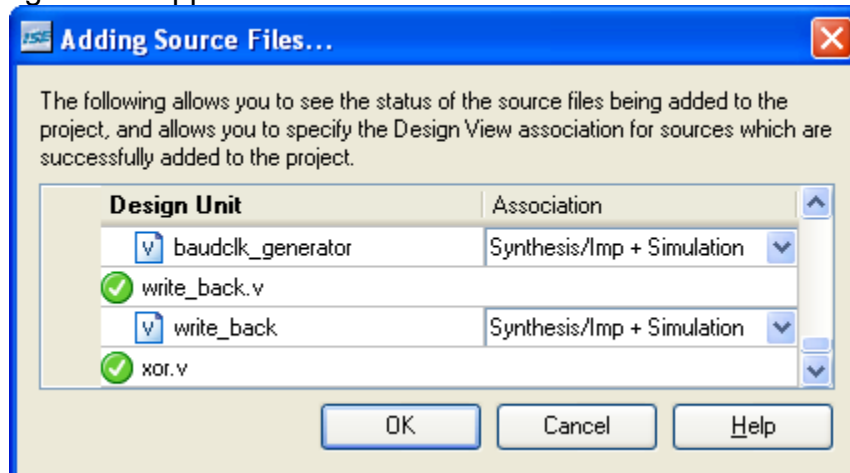


- r. After you have reviewed the entries in the window, click 'Next' continue.

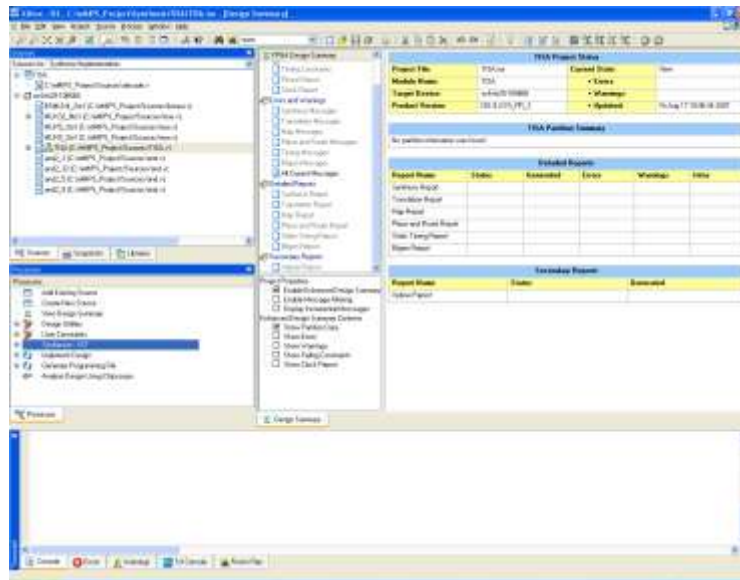
- s. The 'Project Summary' screen should appear.



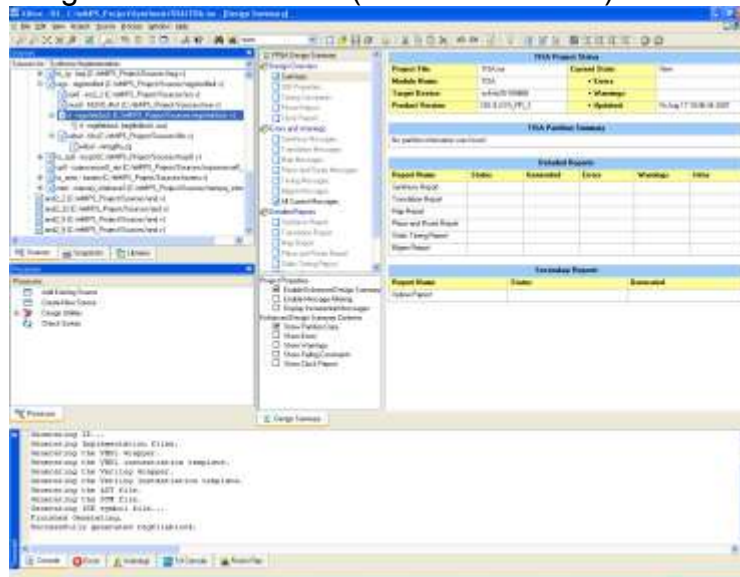
- t. Click finish to complete the project setup.
- u. The 'New Project Wizard' will close and a moment later the 'Adding Source Files' dialog should appear.



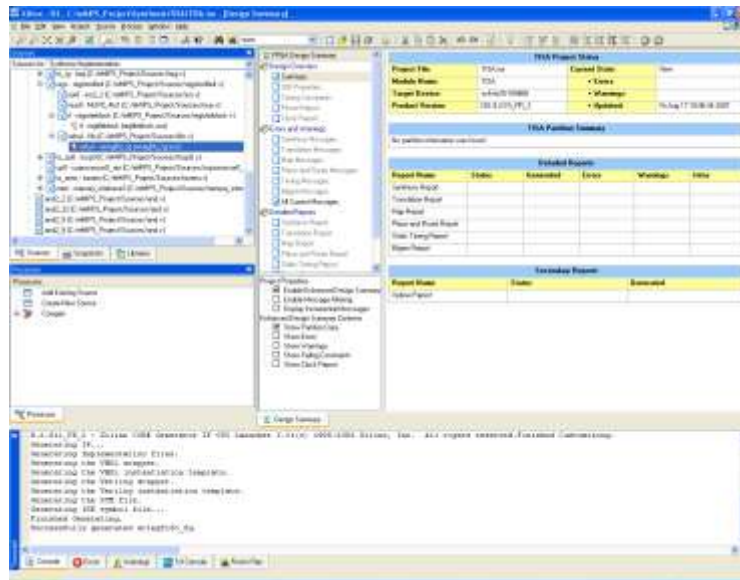
- v. Click 'OK' to continue.
- w. The new project will open in the Xilinx ISE Project Navigator.



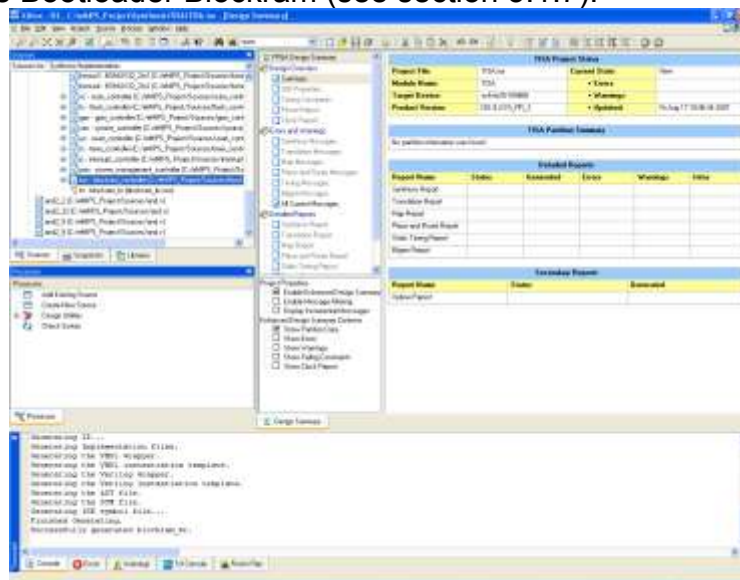
3. The module 'TISA' should be set as the top module in the 'Sources' pane. If not set it as the top module by right-clicking TISA and selecting 'Set as Top Module'.
4. Add the IP Core Modules
 - a. Add the Register File Blockram (see section 3.1.5).



- b. Add the Register File Fifo (see section 3.1.6).

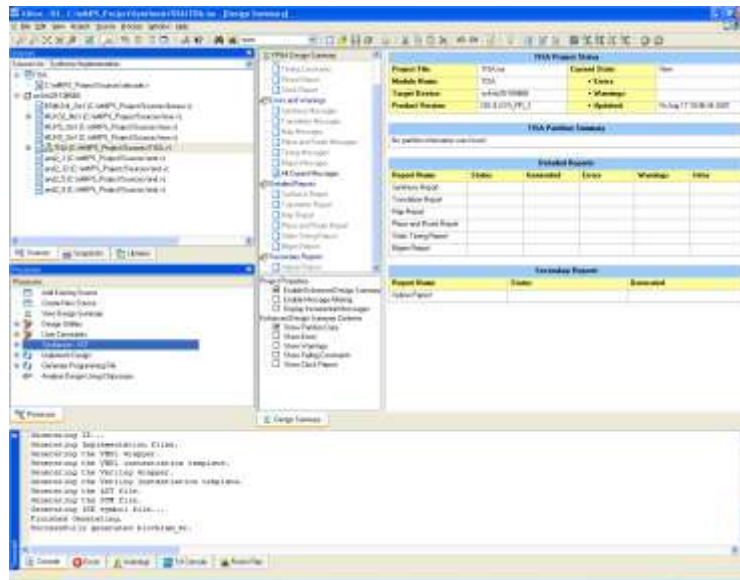


c. Add the Bootloader Blockram (see section 3.1.7).

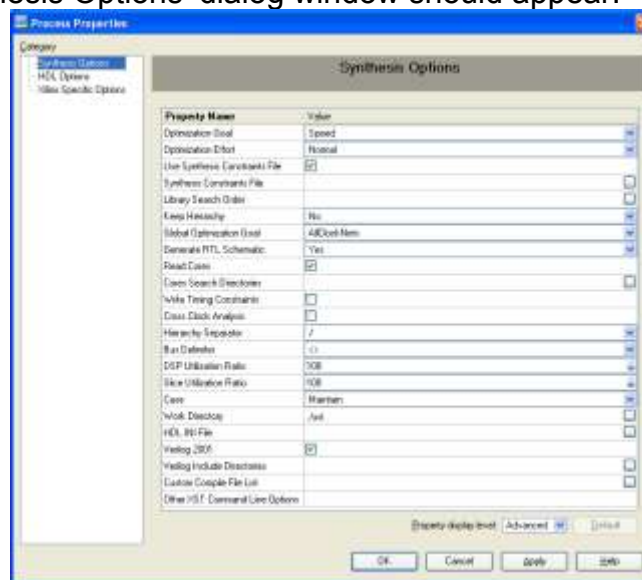


5. Set Synthesis Options

- a. In the 'Sources' pane, select 'TISA'. Then select 'Synthesize - XST' in the 'Processes' pane. Make sure to select the module and not the project icons.

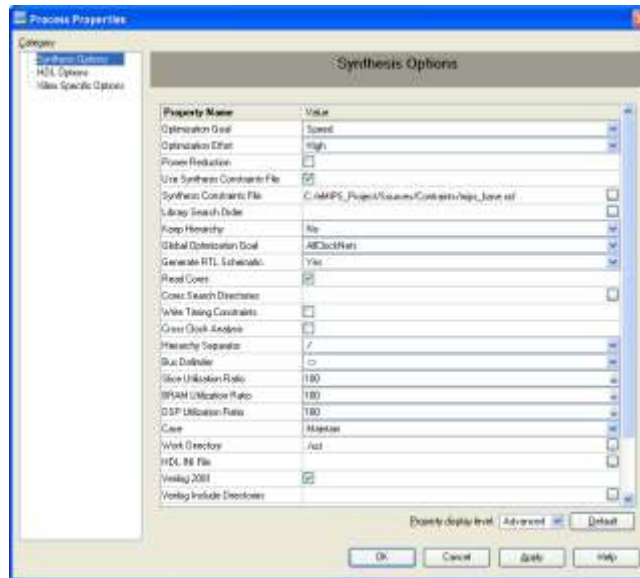


- b. Right-click on 'Synthesize - XST' and select 'Properties'.
- c. The 'Synthesis Options' dialog window should appear.

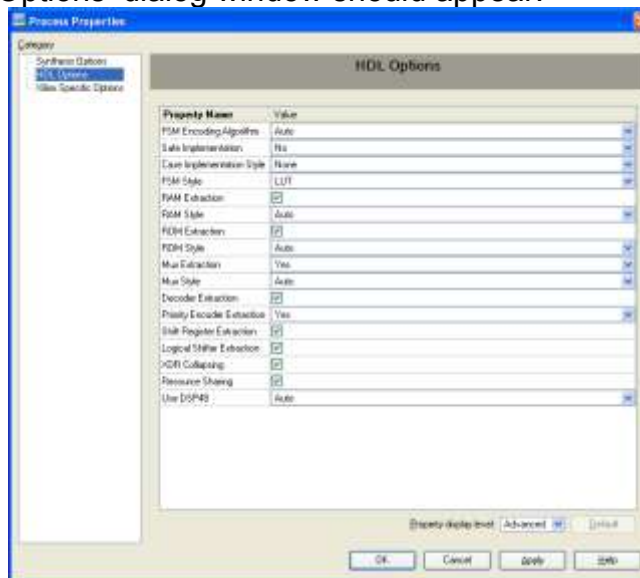


- d. Set the values in the table below to the properties in this window.

Property Name	Value
Optimization Goal	Speed
Optimization Effort	High
Use Synthesis Constraints File	Yes
Synthesis Constraints File	<location of sources>\mips_base.xcf

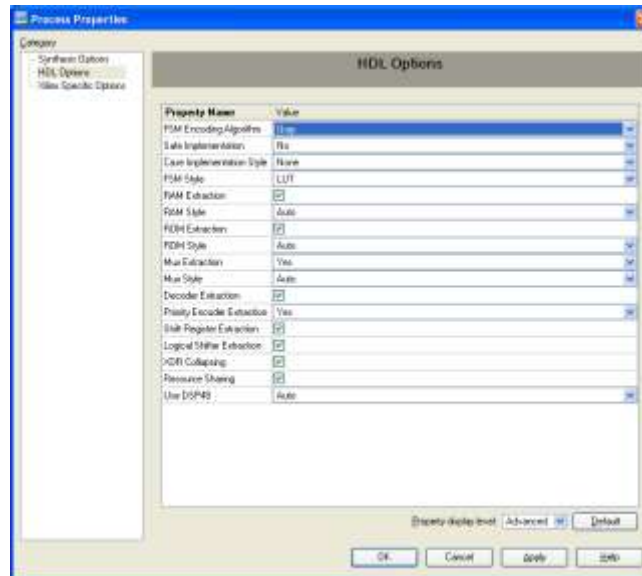


- e. After you have reviewed the entries in the window, click 'HDL Options' in the 'Category' pane.
- f. The 'HDL Options' dialog window should appear.

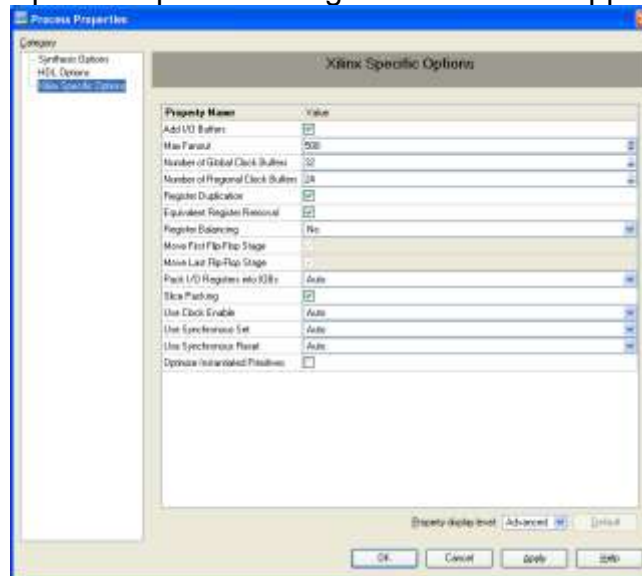


- g. Set the values in the table below to the properties in this window.

Property Name	Value
FSM Encoding Algorithm	Gray

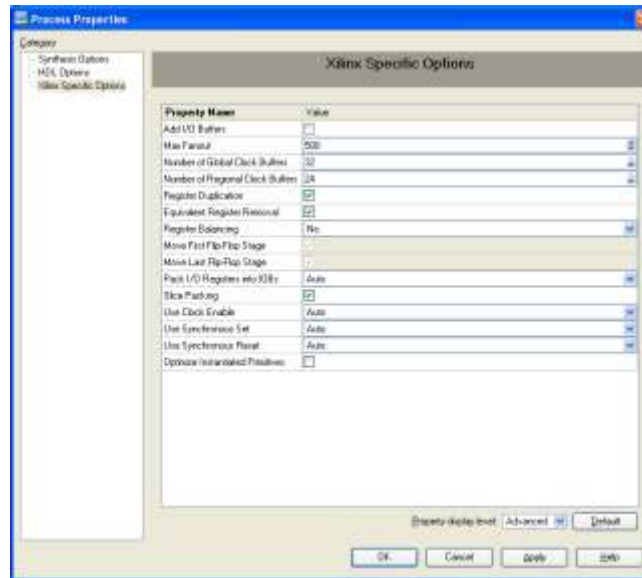


- h. After you have reviewed the entries in the window, click 'Xilinx Specific Options' in the 'Category' pane.
- i. The 'Xilinx Specific Options' dialog window should appear.

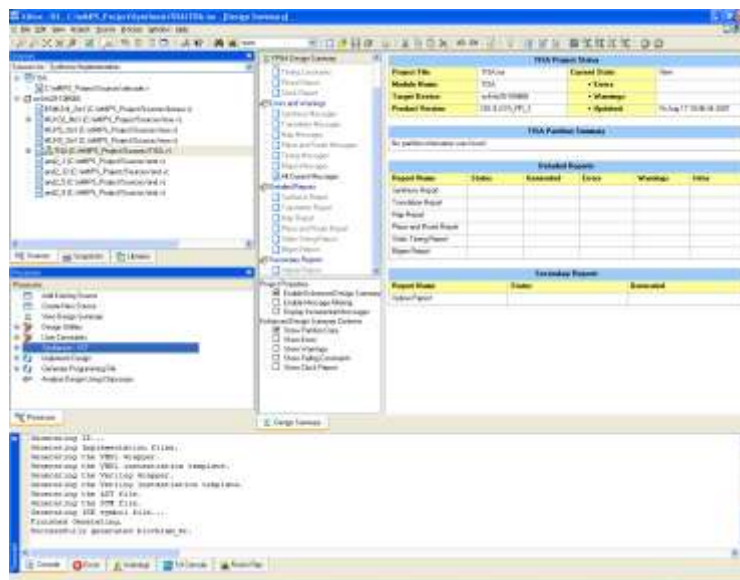


- j. Set the values in the table below to the properties in this window.

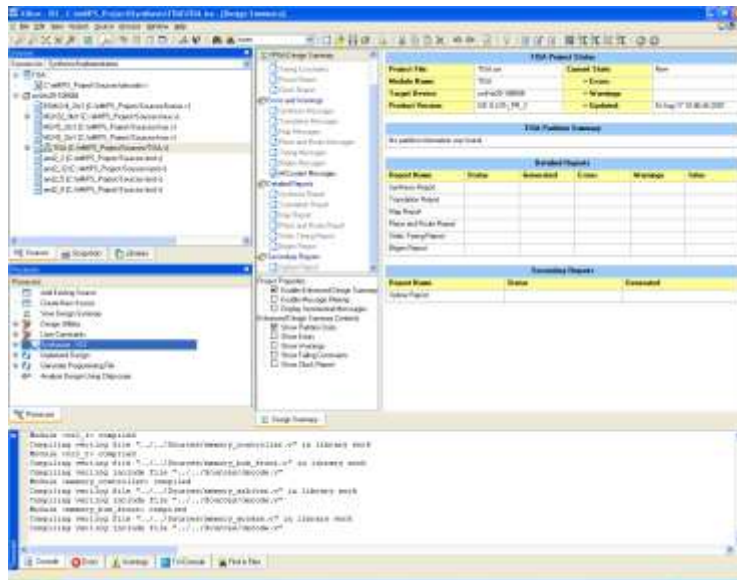
Property Name	Value
Add I/O Buffers	No



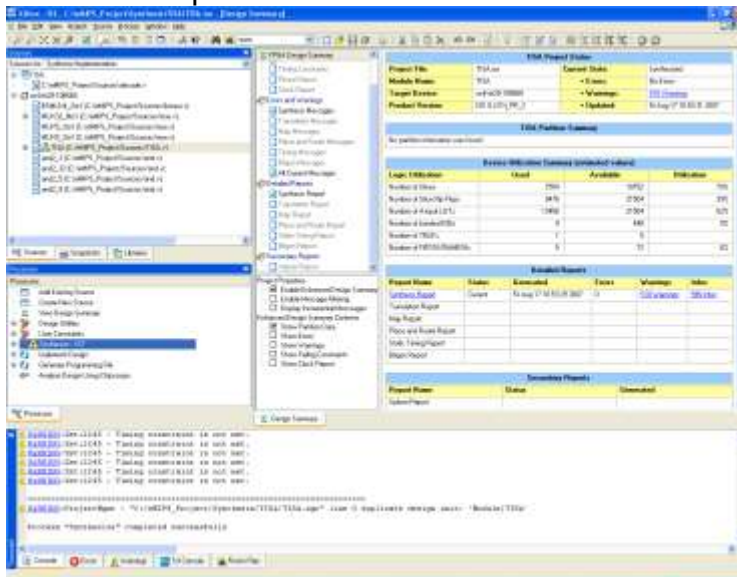
- k. After you have reviewed the entries in the window, click 'OK' to continue.
6. In the 'Sources' pane, select 'TISA'. Then select 'Synthesize - XST' in the 'Processes' pane.



7. You may double-click on 'Synthesize - XST' or you may right click and select 'Run' to start the synthesis process. This may take some time depending on your system, between 30 to 90 minutes.

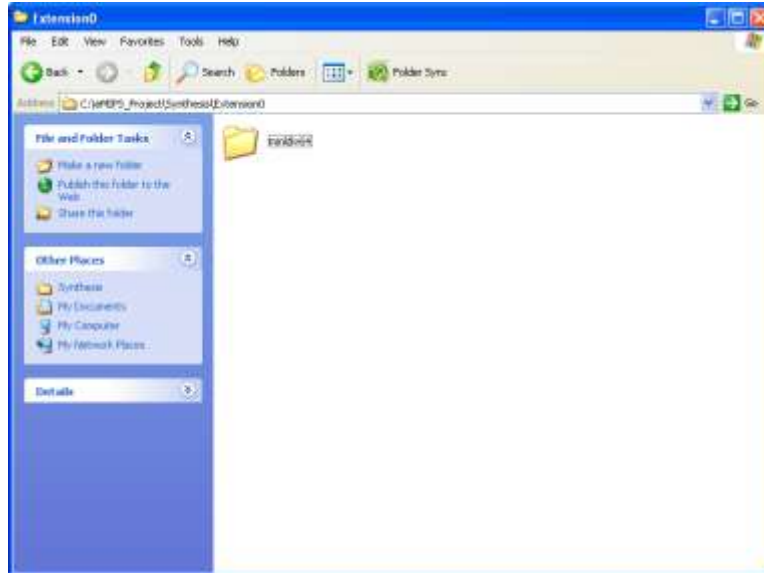


8. You should see a blue spinning icon appear by the 'Synthesize - XST'. As the process is performed some warnings and info messages will be displayed. Most of these are expected.
9. Wait for the process to complete.

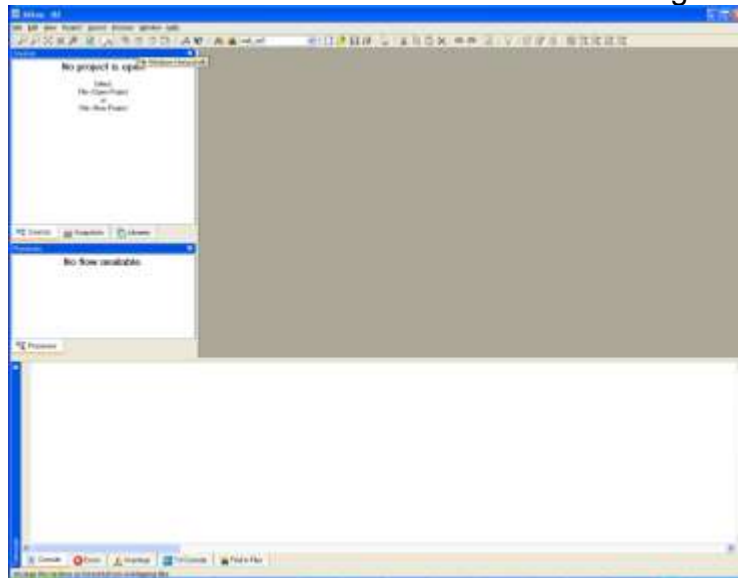


3.1.9 Synthesize the Reconfigurable Region (Extension)

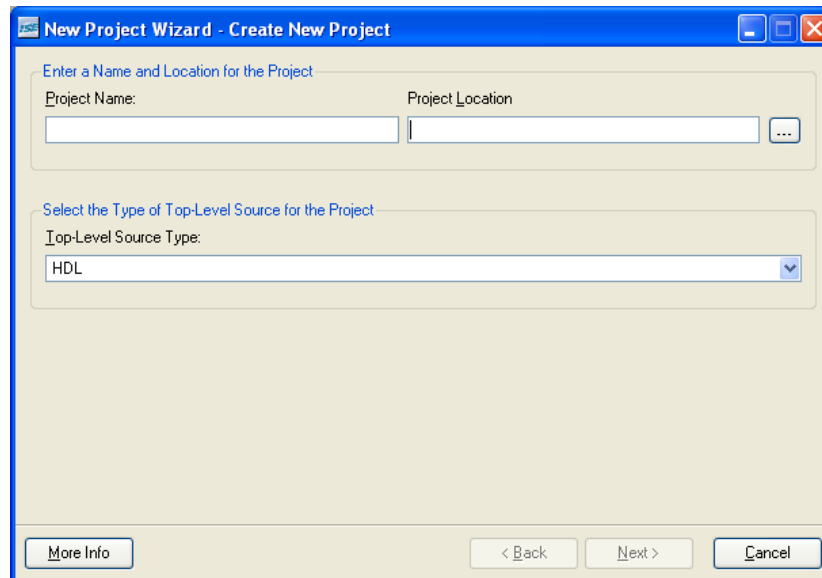
1. In the 'Extension0' synthesis folder, create a folder with the name of your Extension. For the purposes of this example, we will use 'mmldiv64'. Every time you create a new Extension you will repeat this procedure.



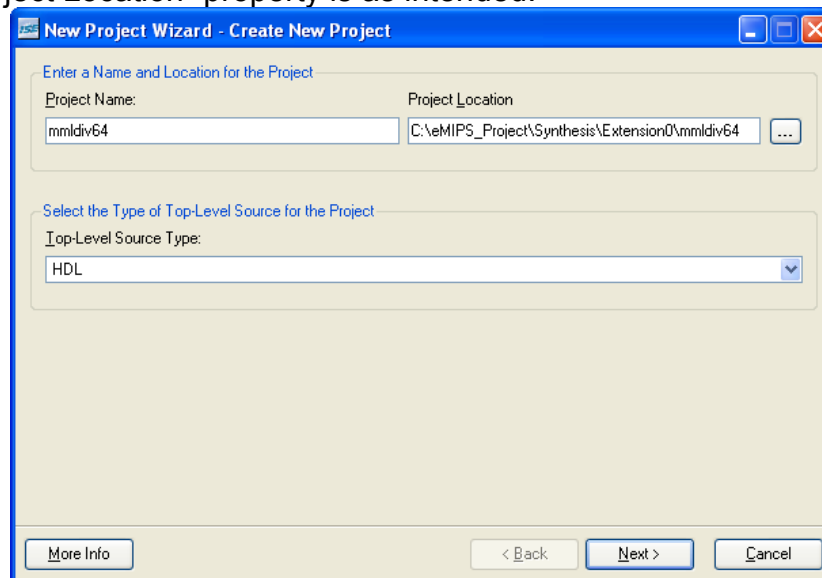
2. Start the install of the Xilinx ISE with the Xilinx Partial Reconfiguration Tools Overlay.



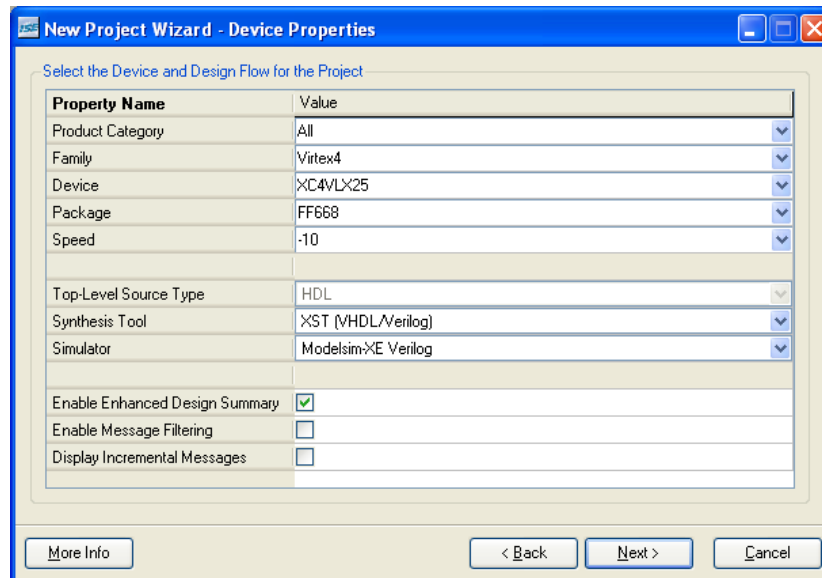
3. Create a new ISE project.
 - a. Click File->New Project in the Menu.
 - b. The 'New Project Wizard' dialog window will appear.



- c. Select the synthesis folder you created for your Extension as the project location and name the project after your Extension. (Ex. mmldiv64). Make sure the “Project Location” property is as intended.



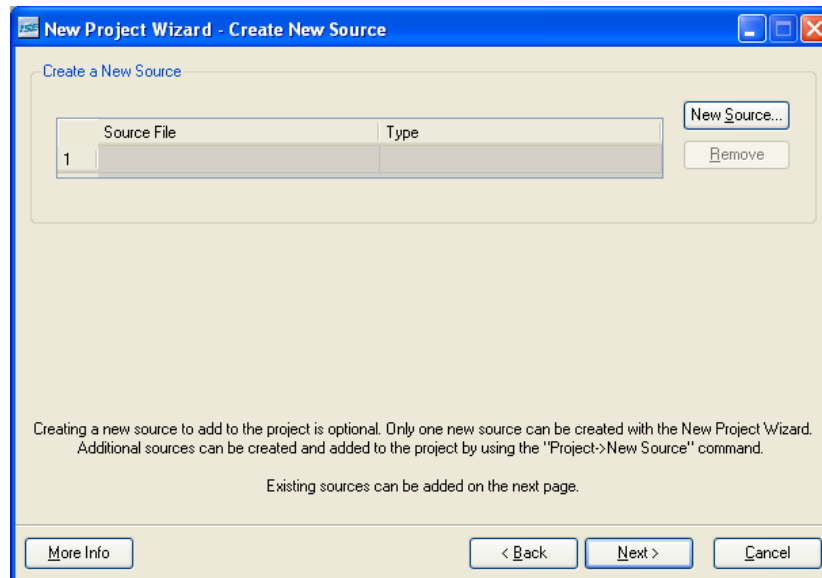
- a. After you have reviewed the entries in the window, click ‘Next’ continue.
b. The ‘Device Properties’ screen should appear.



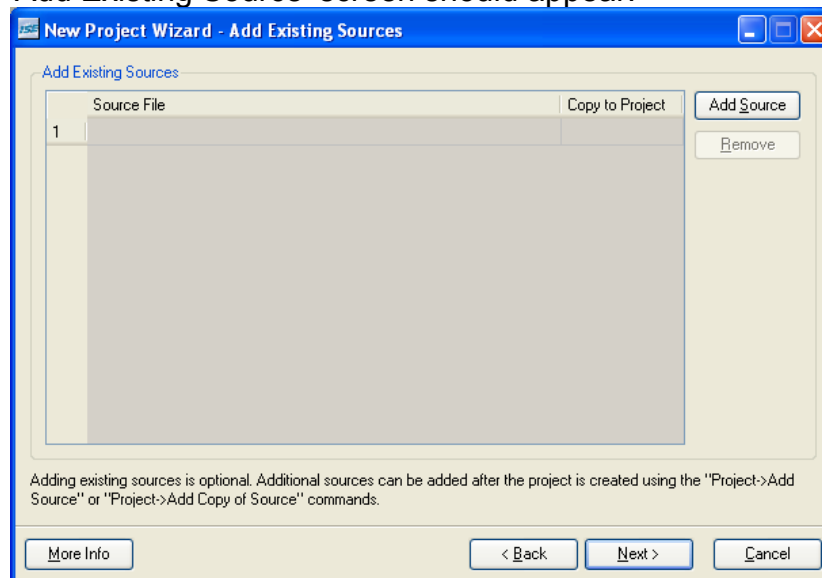
- c. Enter the appropriate settings for your board setup in these fields. For the Xilinx ML401 board, please use the settings in the table below. You may use other synthesis tools other than the XST (Xilinx default) if you chose. However, be aware all of our experiments have used this tool and we cannot vouch for the outcome of another tool.

Property Name	Value
Product Category	All
Family	Virtex4
Device	XC4VLX25
Package	FF668
Speed	-10
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	NA
Enable Enhanced Design Summary	NA
Enable Message Filtering	NA
Display Incremental Messages	NA

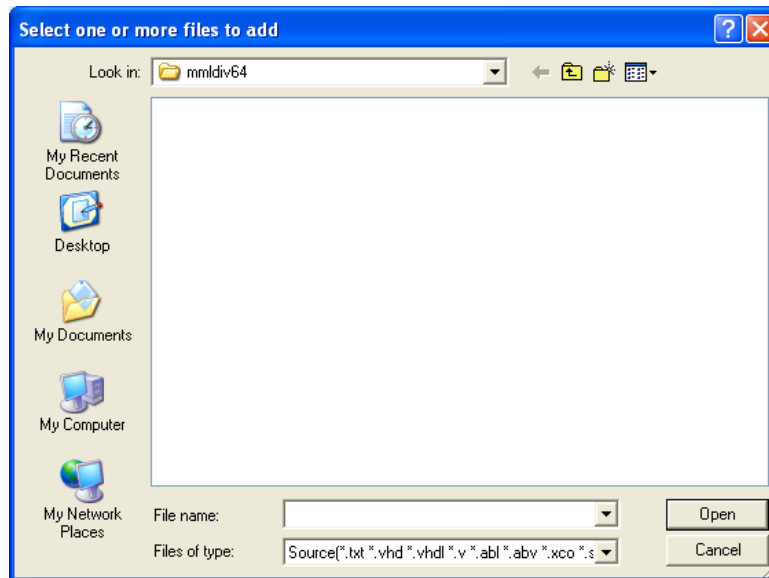
- d. After you have reviewed the entries in the window, click 'Next' continue.
e. The 'Create New Source' screen should appear.



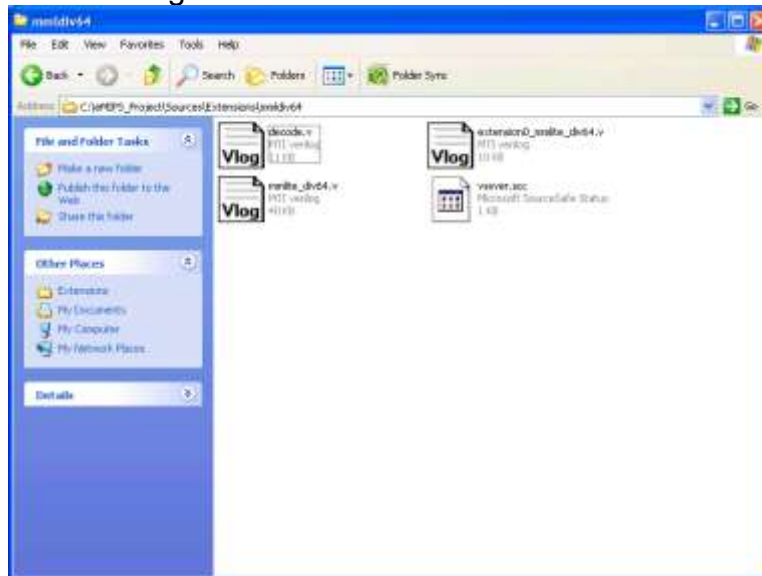
- f. Since you will not be creating any new sources at this time, click 'Next' to continue.
- g. The 'Add Existing Source' screen should appear.



- h. Click the 'Add Source' Button.
- i. The 'Select one or more files to add' dialog should appear.



j. Navigate the dialog window to the eMIPS mmldiv64 source directory.

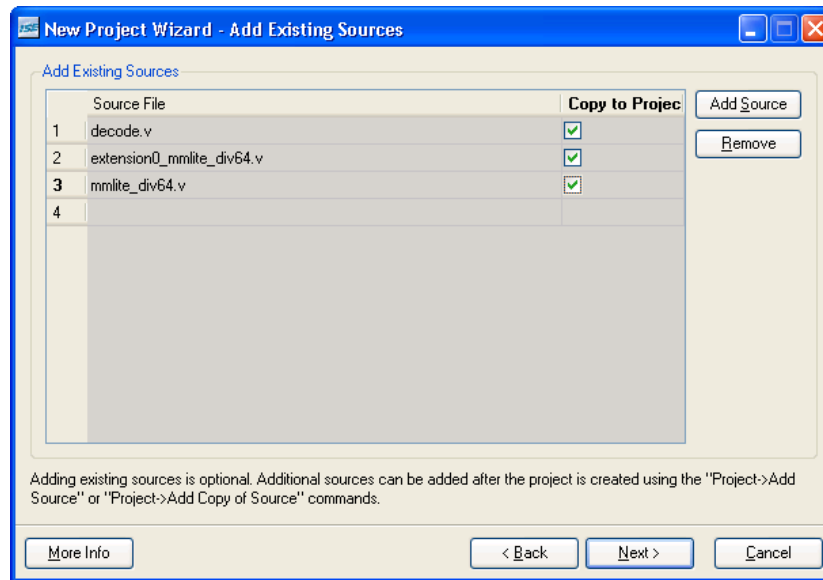


k. Select the files listed in the table below to be added to the project.

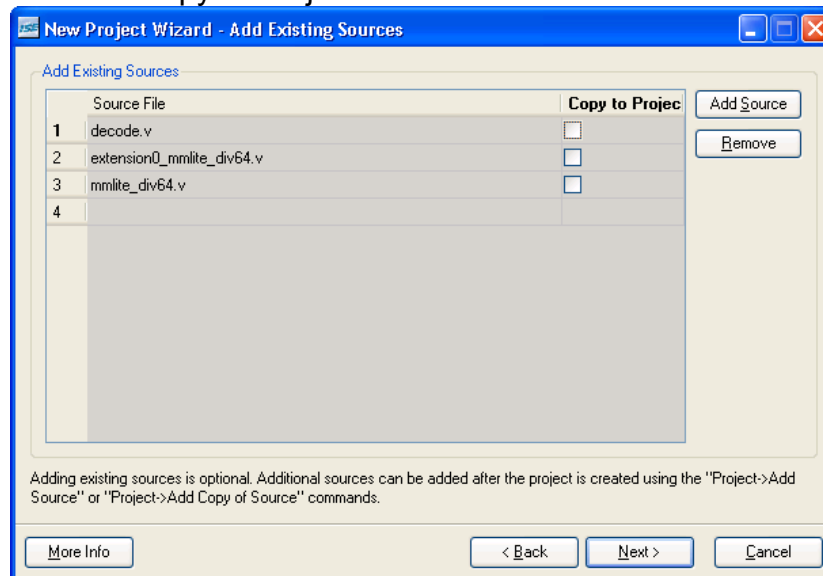
extension0_mmlite_div64.v	mmlite_div64.v	decode.v
---------------------------	----------------	----------

l. When you are done click 'Open'.

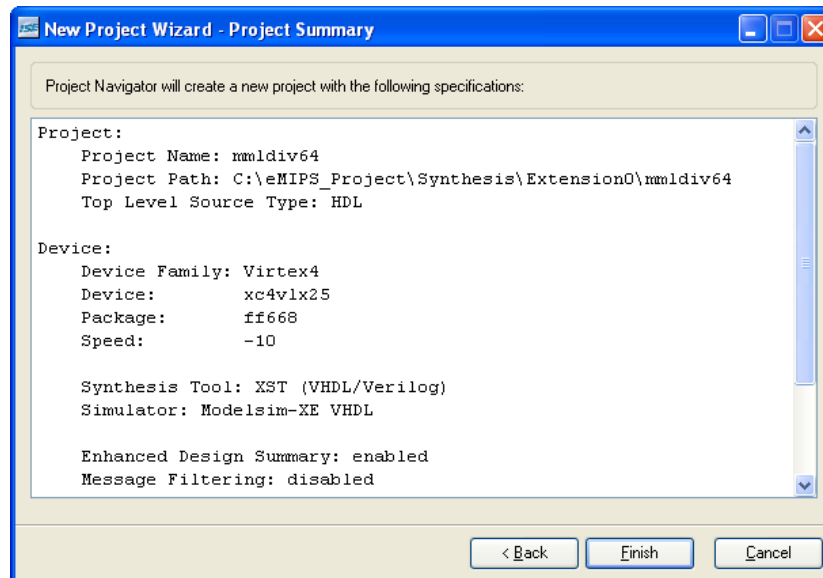
m. The files you selected should be listed in the 'Add Existing Source' screen.



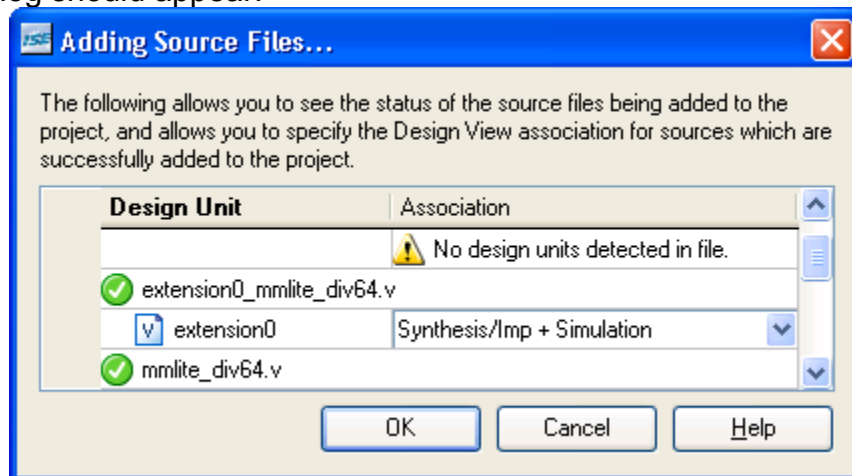
- n. Uncheck the 'Copy to Project' checkbox for each file.



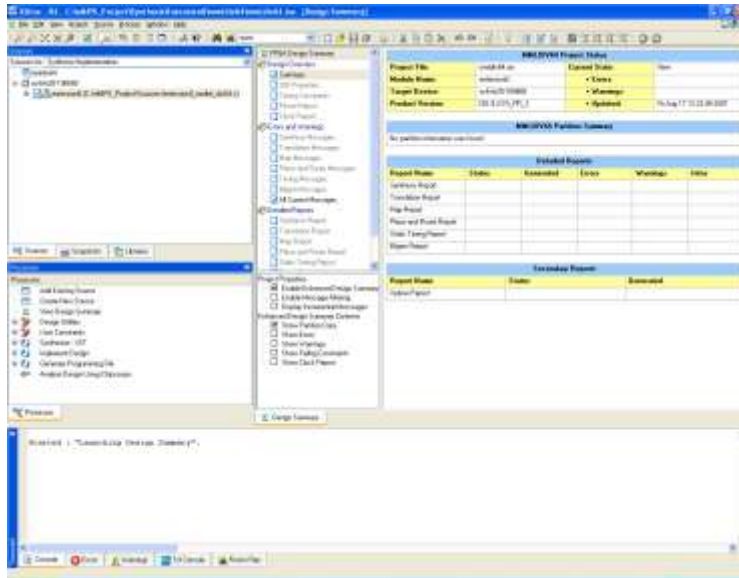
- o. After you have reviewed the entries in the window, click 'Next' continue.
p. The 'Project Summary' screen should appear.



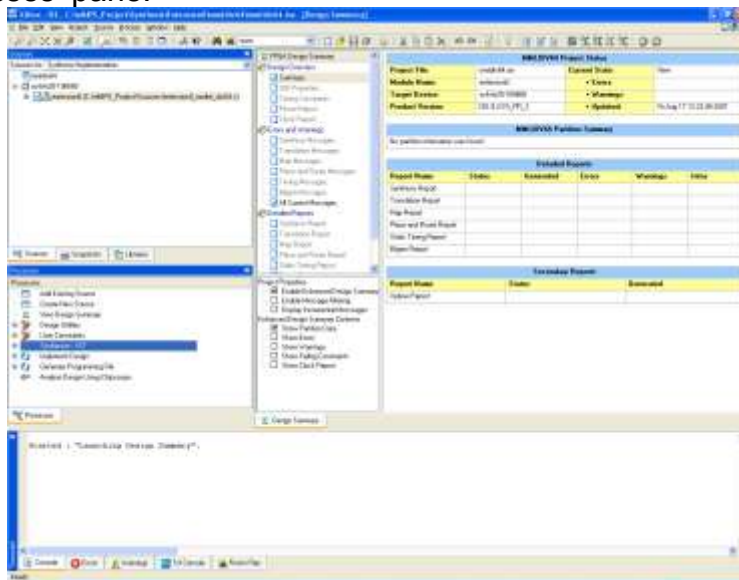
- q. Click finish to complete the project setup.
- r. The 'New Project Wizard' will close and a moment later the 'Adding Source Files' dialog should appear.



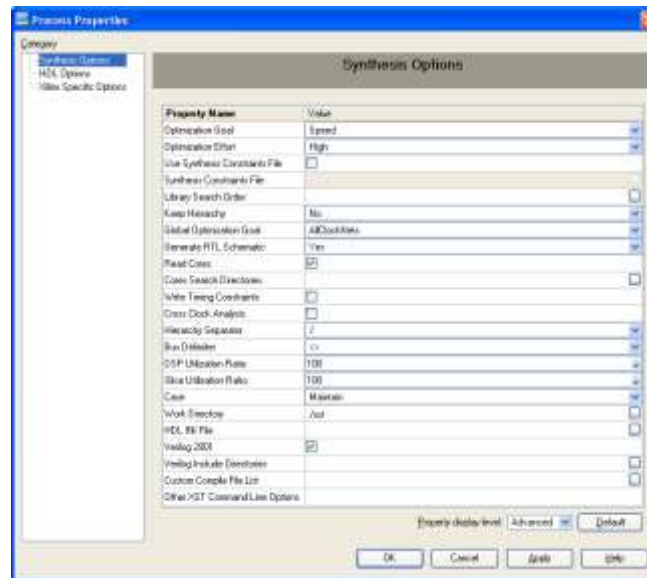
- s. Click 'OK' to continue.
- t. The new project will open in the Xilinx ISE Project Navigator.



4. The module 'extension0' should be set as the top module in the 'Sources' pane. If not set it as the top module by right-clicking 'extension0' and selecting 'Set as Top Module'.
5. If your Extension includes IP Cores, Add the necessary IP Core Modules. In the case of this example we have no IP Cores and no other module is needed.
6. Set Synthesis Options
 - a. In the 'Sources' pane, select 'extension0'. Then select 'Synthesize - XST' in the 'Processes' pane.

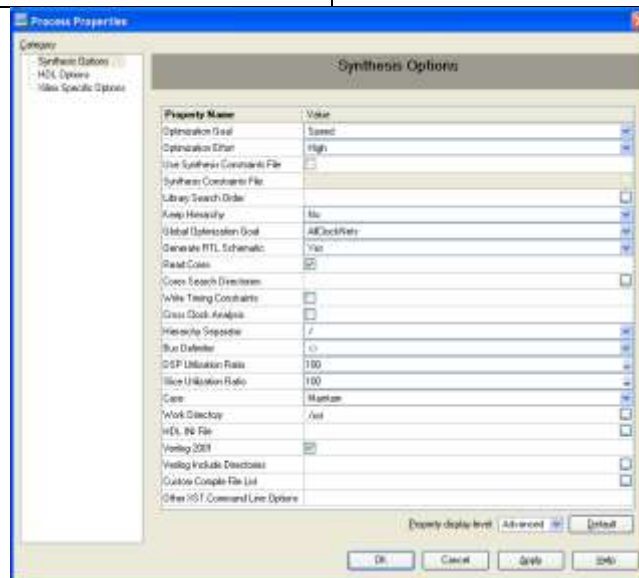


- b. Right-click on 'Synthesize - XST' and select 'Properties'.
 - c. The 'Synthesis Options' dialog window should appear.



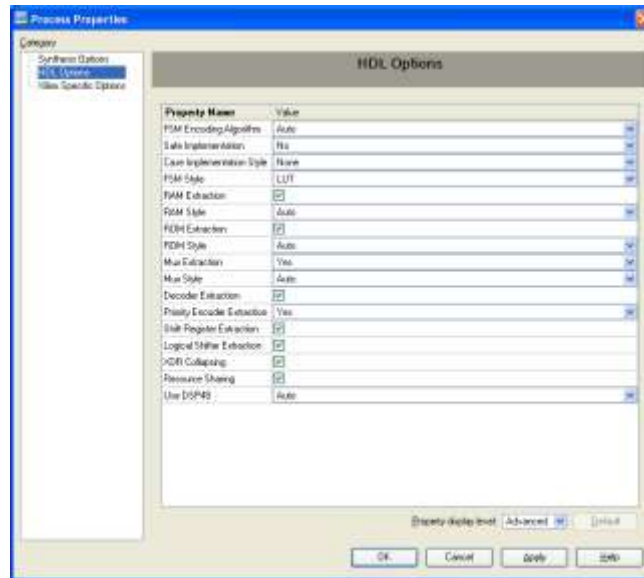
d. Set the values in the table below to the properties in this window.

Property Name	Value
Optimization Goal	Speed
Optimization Effort	High
Use Synthesis Constraints File	NA
Synthesis Constraints File	NA



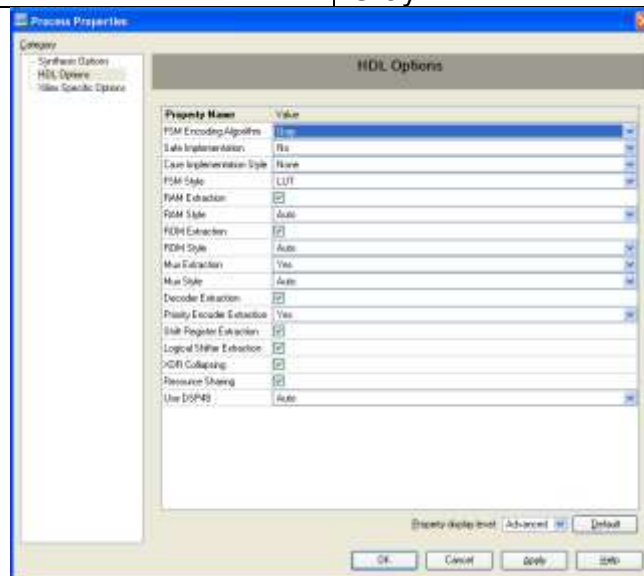
e. After you have reviewed the entries in the window, click 'HDL Options' in the 'Category' pane.

f. The 'HDL Options' dialog window should appear.

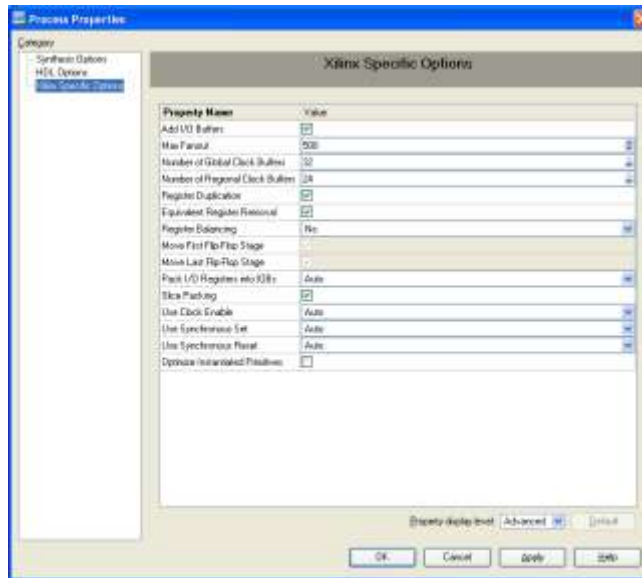


g. Set the values in the table below to the properties in this window.

Property Name	Value
FSM Encoding Algorithm	Gray

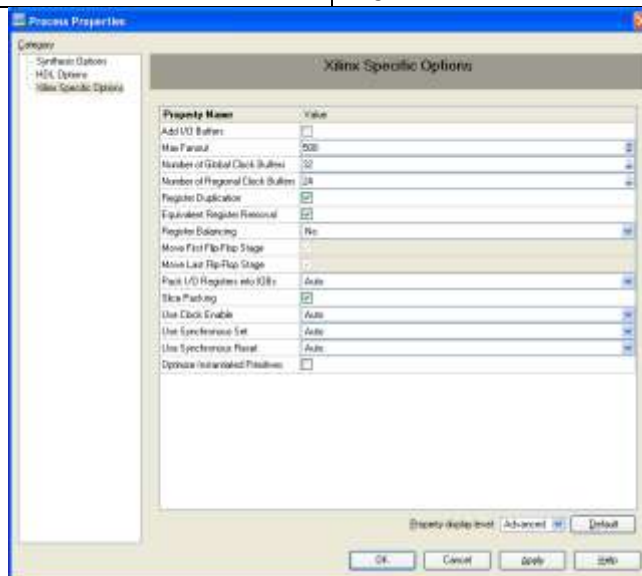


- h. After you have reviewed the entries in the window, click 'Xilinx Specific Options' in the 'Category' pane.
- i. The 'Xilinx Specific Options' dialog window should appear.



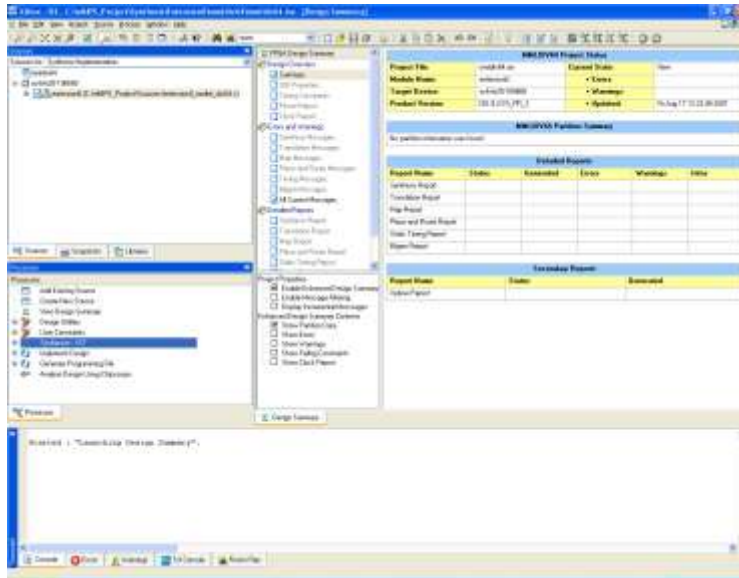
j. Set the values in the table below to the properties in this window.

Property Name	Value
Add I/O Buffers	No

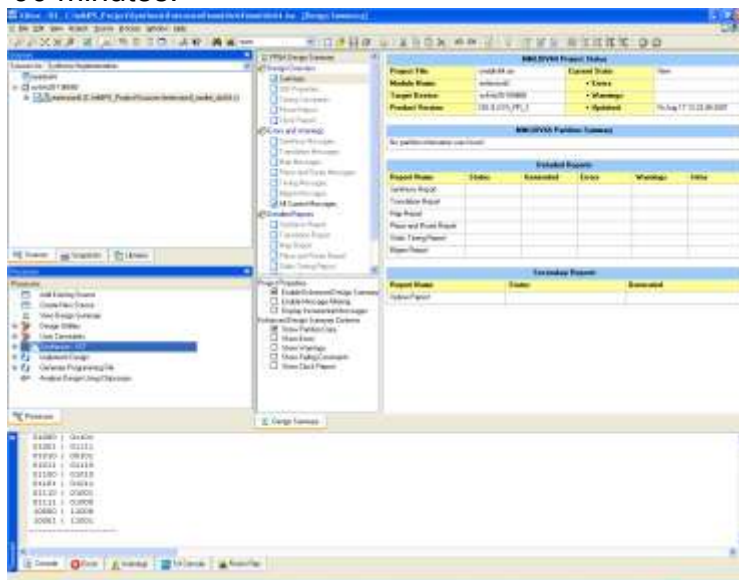


k. After you have reviewed the entries in the window, click 'OK' to continue.

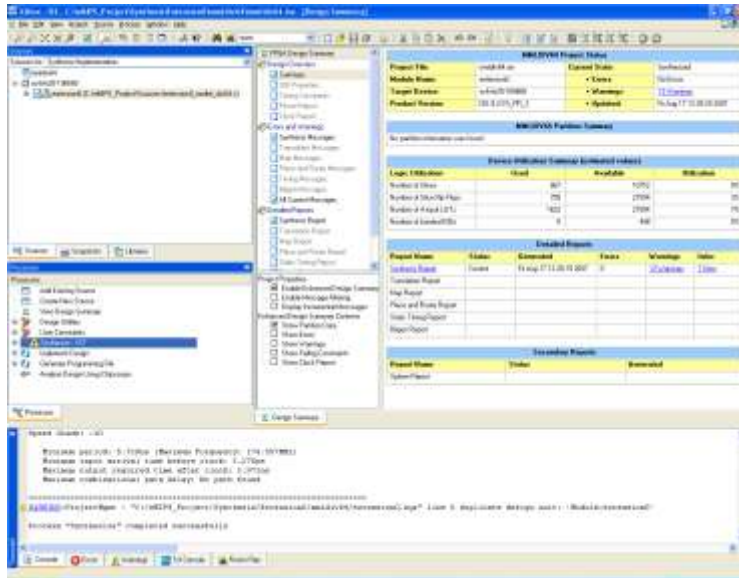
7. In the 'Sources' pane, select 'extension0'. Then select 'Synthesize - XST' in the 'Processes' pane.



8. You may double-click on 'Synthesize - XST' or you may right click and select 'Run' to start the synthesis process. This may take some time depending on your system, between 30 to 90 minutes.



9. You should see a blue spinning icon appear by the 'Synthesizer - XST'. As the process is performed some warnings and info messages will be displayed. Most of these are expected.
10. Wait for the process to complete.

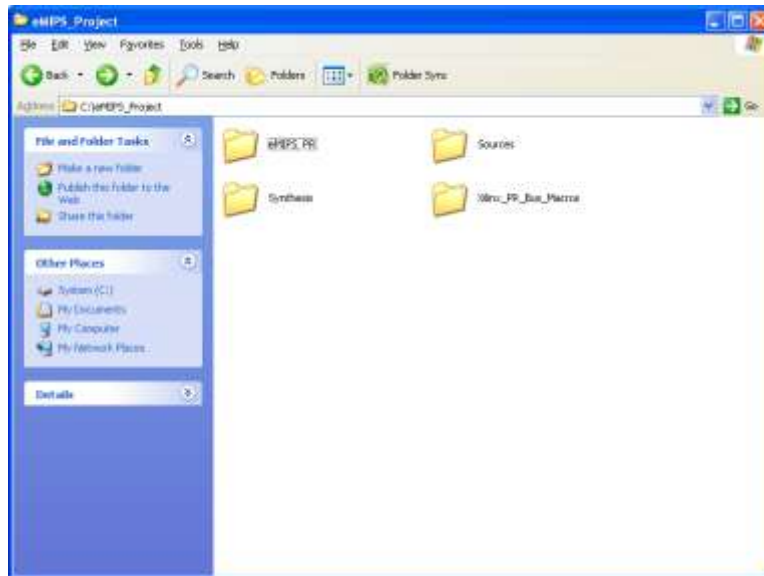


3.2 Building the Configuration Design Files and the Bit Files

3.2.1 Building the PR Version with the PR flow

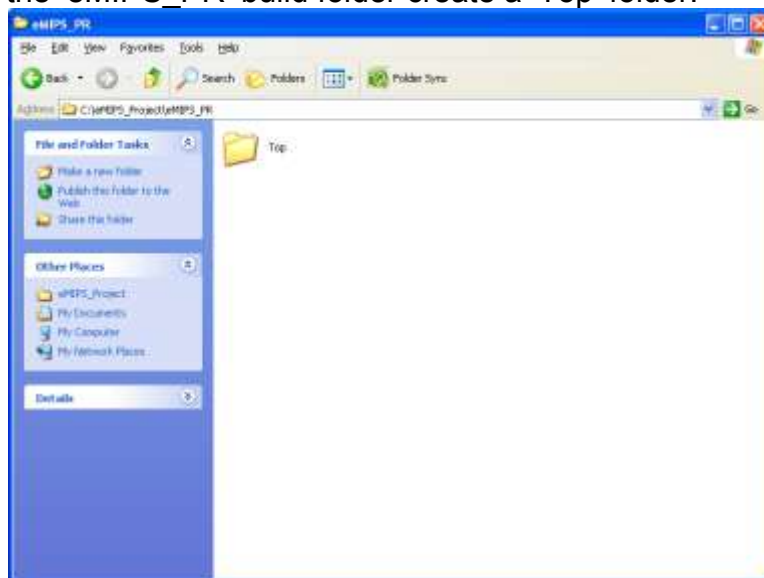
The following instructions are for building the eMIPS microprocessor system using the Xilinx Partial Reconfiguration tools. Any changes in the system sources will require all parts of this process to be carried out for the system to work and the changes to be applied. This is the final deliverable for a partially reconfigurable design. Out of this process will be both the full and partial bit files for initializing the system at start up and for modifying it during run time.

1. If the Project is not yet made, Set up the Project Directory (see 3.1.3)
2. If not already done, Change Environment to PR Xilinx ISE (see 3.1.2)
3. Synthesize the Top Level Module (see 3.1.4)
4. Synthesize the Base Design (TISA) (see 3.1.8)
5. Synthesize the Reconfigurable Region (Extension) (see 3.1.9)
6. Create a build folder, called “eMIPS_PR”, in the project directory.

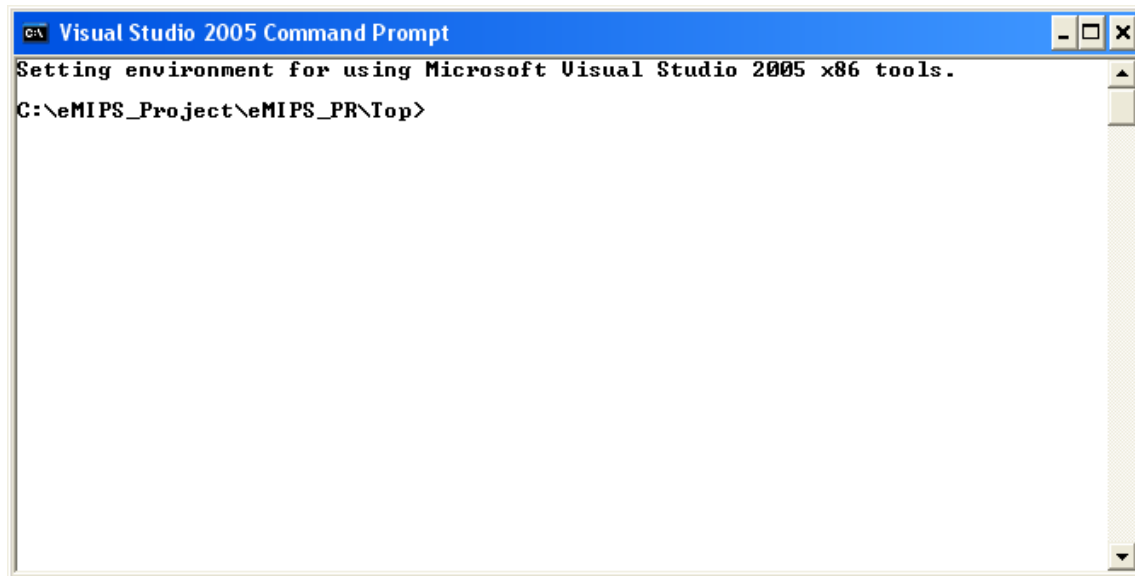


7. Build the Top Level Module

- a. Inside the 'eMIPS_PR' build folder create a 'Top' folder.



- b. Open a command prompt within this folder

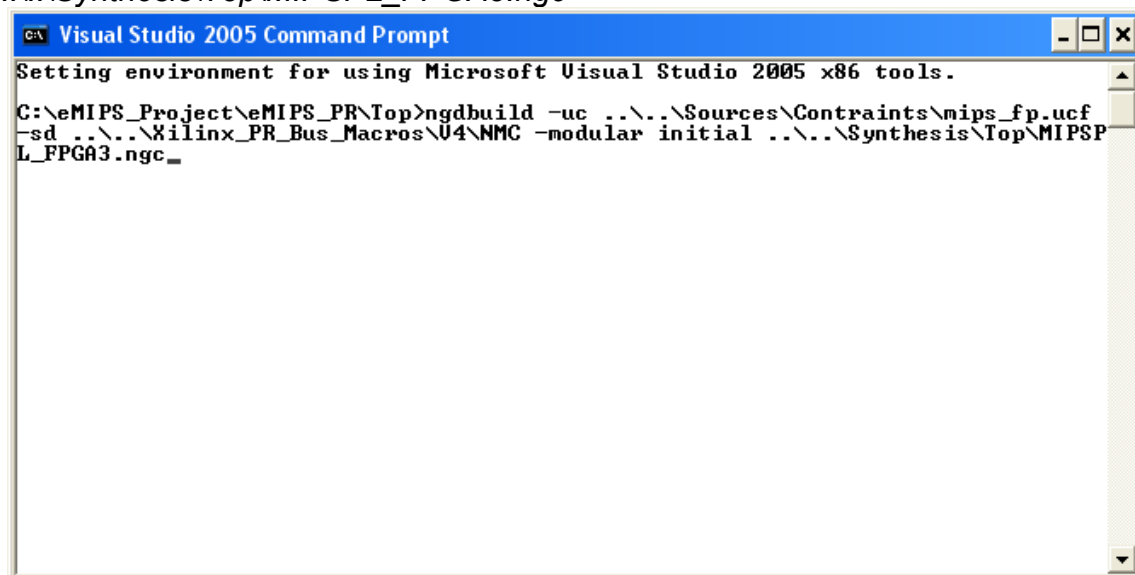


```
C:\ Visual Studio 2005 Command Prompt
Setting environment for using Microsoft Visual Studio 2005 x86 tools.
C:\eMIPS_Project\emIPS_PR\Top>
```

c. Generate the Top Level Netlist.

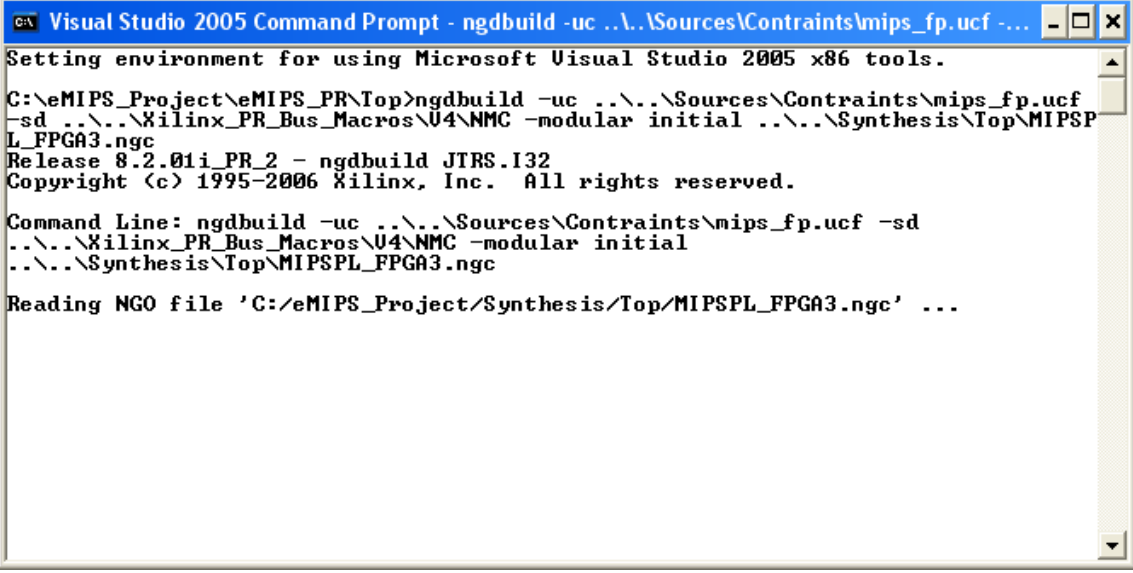
i. Type the command: “ngdbuild -uc <floorplanned ucf file> -sd <location of Bus Macro files> -modular initial <Top Level Netlist>”. In our case:

```
“ngdbuild -uc ..\..\Sources\Constraints\mips_fp.ucf -sd  
..\..\Xilinx_PR_Bus_Macros\V4\NMC -modular initial  
..\..\Synthesis\Top\MIPSPL_FPGA3.ngc”
```



```
C:\ Visual Studio 2005 Command Prompt
Setting environment for using Microsoft Visual Studio 2005 x86 tools.
C:\eMIPS_Project\emIPS_PR\Top>ngdbuild -uc ..\..\Sources\Constraints\mips_fp.ucf  
-sd ..\..\Xilinx_PR_Bus_Macros\V4\NMC -modular initial ..\..\Synthesis\Top\MIPSPL_FPGA3.ngc_
```

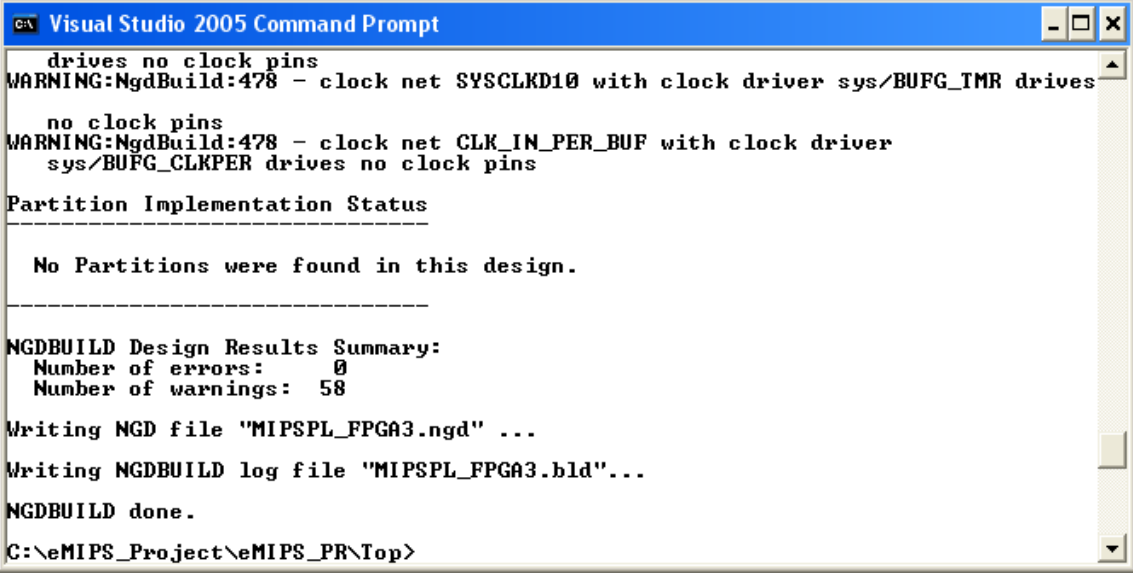
ii. Press Enter. Some warnings and info messages will appear. Most of them are expected.



```

C:\ Visual Studio 2005 Command Prompt - ngdbuild -uc ..\..\Sources\Constraints\mips_fp.ucf -...
Setting environment for using Microsoft Visual Studio 2005 x86 tools.
C:\eMIPS_Project\emIPS_PR\Top>ngdbuild -uc ..\..\Sources\Constraints\mips_fp.ucf
-sd ..\..\Xilinx_PR_Bus_Macros\U4\NMC -modular initial ..\..\Synthesis\Top\MIPSP
L_FPGA3.ngc
Release 8.2.01i_PR_2 - ngdbuild JTRS.I32
Copyright (c) 1995-2006 Xilinx, Inc. All rights reserved.
Command Line: ngdbuild -uc ..\..\Sources\Constraints\mips_fp.ucf -sd
..\..\Xilinx_PR_Bus_Macros\U4\NMC -modular initial
..\..\Synthesis\Top\MIPSPL_FPGA3.ngc
Reading NGO file 'C:/eMIPS_Project/Synthesis/Top/MIPSPL_FPGA3.ngc' ...
  
```

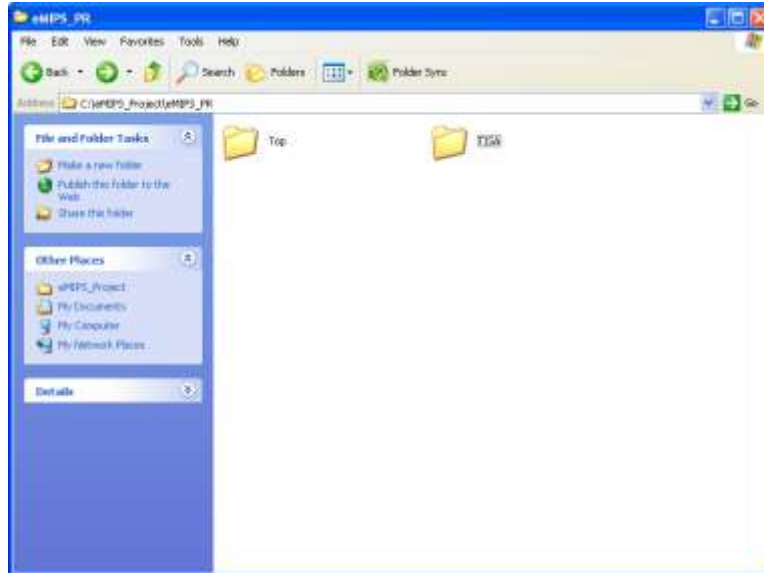
iii. Wait for the process to complete.



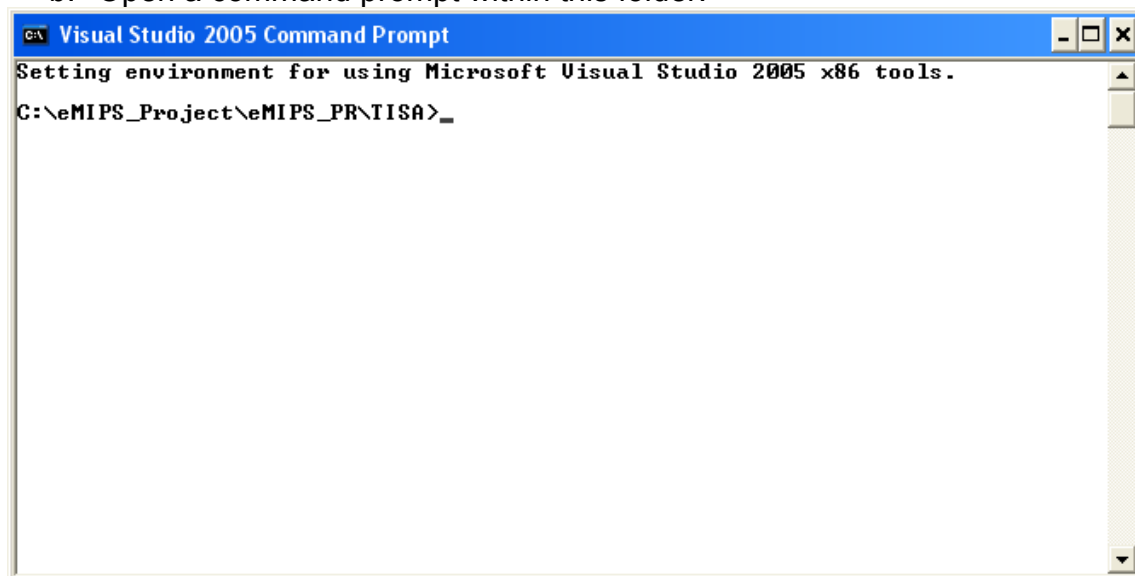
```

C:\ Visual Studio 2005 Command Prompt
drives no clock pins
WARNING:NgdBuild:478 - clock net SYSCLKD10 with clock driver sys/BUFG_TMR drives
no clock pins
WARNING:NgdBuild:478 - clock net CLK_IN_PER_BUF with clock driver
sys/BUFG_CLKPER drives no clock pins
Partition Implementation Status
-----
No Partitions were found in this design.
-----
NGDBUILD Design Results Summary:
Number of errors: 0
Number of warnings: 58
Writing NGD file "MIPSPL_FPGA3.ngd" ...
Writing NGDBUILD log file "MIPSPL_FPGA3.bld"...
NGDBUILD done.
C:\eMIPS_Project\emIPS_PR\Top>
  
```

8. Build the Base Design (TISA)
 - a. Inside the 'eMIPS_PR' build folder create a 'TISA' folder




b. Open a command prompt within this folder.



c. Generate the Base Netlist (TISA)


- i. Make sure the constraint file 'mips_fp.ucf' is not 'READ ONLY' before you begin this step. If it is, it will result in an error.
- ii. Type the command: "ngdbuild -uc <floorplanned ucf file> -sd <location of Bus Macro files> -sd <location of TISA synthesis files> -modular initial <Top Level Netlist>". In our case:

```
"ngdbuild -uc ..\..\Sources\Constraints\mips_fp.ucf -sd ..\..\Xilinx_PR_Bus_Macros\V4\NMC -sd ..\..\Synthesis\TISA -modular initial ..\..\Synthesis\Top\MIPSPL_FPGA3.ngc".
```



```
Visual Studio 2005 Command Prompt
Setting environment for using Microsoft Visual Studio 2005 x86 tools.
C:\eMIPS_Project\emIPS_PR\TISA>ngdbuild -uc ..\..\Sources\Constraints\mips_fp.ucf
-sd ..\..\Xilinx_PR_Bus_Macros\U4\NMC -sd ..\..\Synthesis\TISA -modular initial
..\..\Synthesis\Top\MIPSPL_FPGA3.ngc
```

- iii. Press Enter. Some warnings and info messages will appear. Most of them are expected.

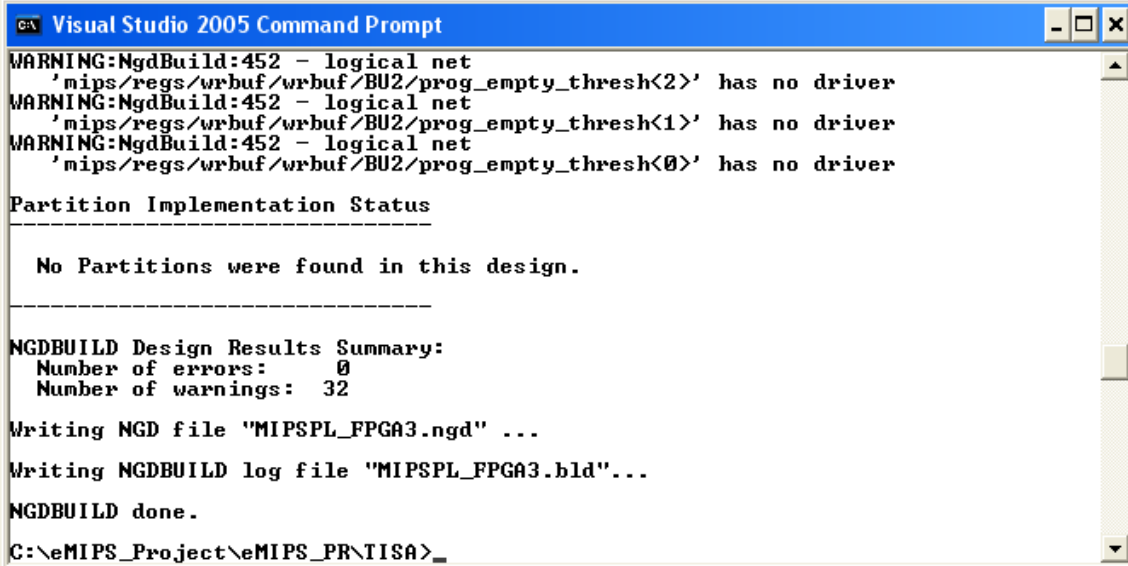


```
Visual Studio 2005 Command Prompt - ngdbuild -uc ..\..\Sources\Constraints\mips_fp.ucf -...
Setting environment for using Microsoft Visual Studio 2005 x86 tools.
C:\eMIPS_Project\emIPS_PR\TISA>ngdbuild -uc ..\..\Sources\Constraints\mips_fp.ucf
-sd ..\..\Xilinx_PR_Bus_Macros\U4\NMC -sd ..\..\Synthesis\TISA -modular initial
..\..\Synthesis\Top\MIPSPL_FPGA3.ngc
Release 8.2.01i_PR_2 - ngdbuild JTRS.I32
Copyright (c) 1995-2006 Xilinx, Inc. All rights reserved.

Command Line: ngdbuild -uc ..\..\Sources\Constraints\mips_fp.ucf -sd
..\..\Xilinx_PR_Bus_Macros\U4\NMC -sd ..\..\Synthesis\TISA -modular initial
..\..\Synthesis\Top\MIPSPL_FPGA3.ngc

Reading NGO file 'C:/eMIPS_Project/Synthesis/Top/MIPSPL_FPGA3.ngc' ...
```

- iv. Wait for the process to complete.



```

C:\ Visual Studio 2005 Command Prompt
WARNING:NgdBuild:452 - logical net
'mips/regs/wrbuf/wrbuf/BU2/prog_empty_thresh<2>' has no driver
WARNING:NgdBuild:452 - logical net
'mips/regs/wrbuf/wrbuf/BU2/prog_empty_thresh<1>' has no driver
WARNING:NgdBuild:452 - logical net
'mips/regs/wrbuf/wrbuf/BU2/prog_empty_thresh<0>' has no driver

Partition Implementation Status
-----

No Partitions were found in this design.

-----

NGDBUILD Design Results Summary:
Number of errors:    0
Number of warnings: 32

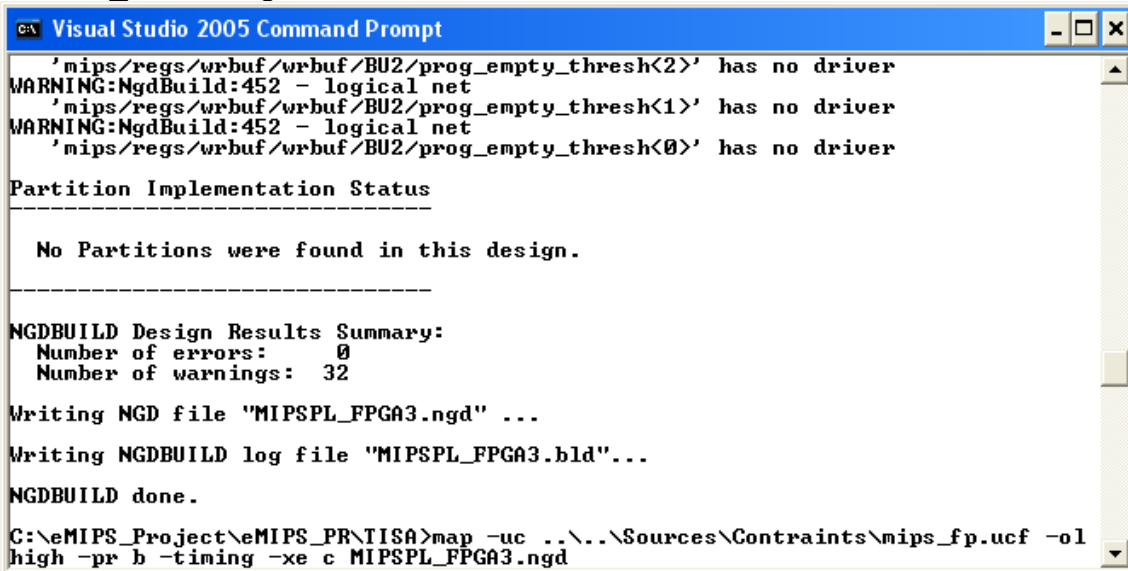
Writing NGD file "MIPSPL_FPGA3.ngd" ...
Writing NGDBUILD log file "MIPSPL_FPGA3.bld"...
NGDBUILD done.
C:\eMIPS_Project\emIPS_PR\TISA>_

```

d. Place/Map Base (TISA) Logic to FPGA hardware.

- i. Type the following: "map -uc ..<floorplanned ucf file> -ol high -pr b -timing -xe c <input file>". In our case:

"map -uc ..\..\Sources\Constraints\mips_fp.ucf -ol high -pr b -timing -xe c MIPSPL_FPGA3.ngd".



```

C:\ Visual Studio 2005 Command Prompt
'mips/regs/wrbuf/wrbuf/BU2/prog_empty_thresh<2>' has no driver
WARNING:NgdBuild:452 - logical net
'mips/regs/wrbuf/wrbuf/BU2/prog_empty_thresh<1>' has no driver
WARNING:NgdBuild:452 - logical net
'mips/regs/wrbuf/wrbuf/BU2/prog_empty_thresh<0>' has no driver

Partition Implementation Status
-----

No Partitions were found in this design.

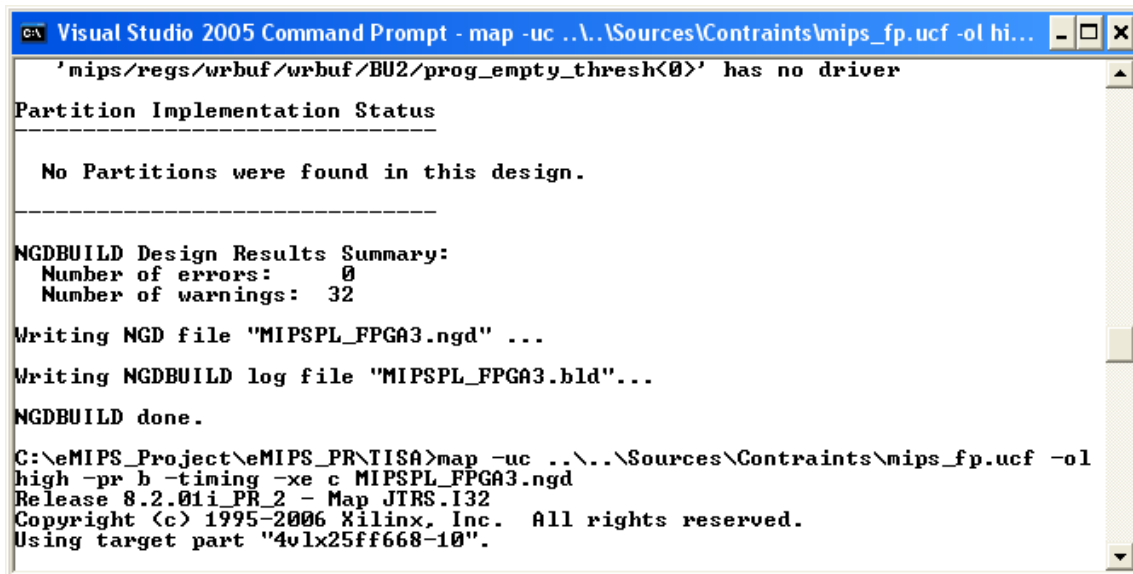
-----

NGDBUILD Design Results Summary:
Number of errors:    0
Number of warnings: 32

Writing NGD file "MIPSPL_FPGA3.ngd" ...
Writing NGDBUILD log file "MIPSPL_FPGA3.bld"...
NGDBUILD done.
C:\eMIPS_Project\emIPS_PR\TISA>map -uc ..\..\Sources\Constraints\mips_fp.ucf -ol
high -pr b -timing -xe c MIPSPL_FPGA3.ngd

```

- ii. Press Enter. Some warnings and info messages will appear. Most of them are expected. This may take some time depending on your system, between 30 to 90 minutes.

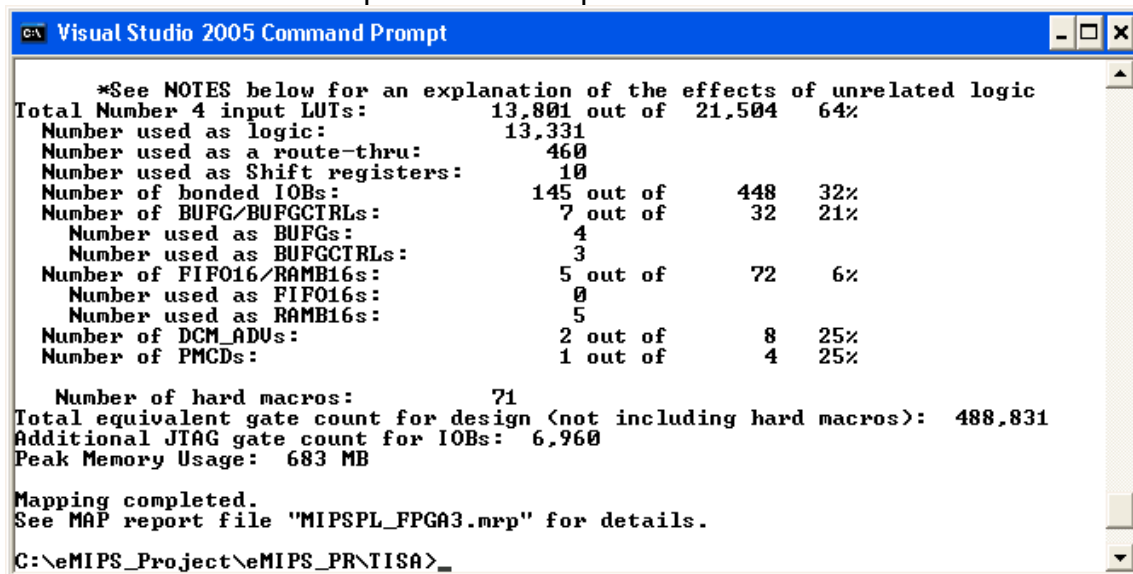


```

C:\ Visual Studio 2005 Command Prompt - map -uc ..\..\Sources\Constraints\mips_fp.ucf -ol hi...
'mips/regs/wrbuf/wrbuf/BU2/prog_empty_thresh<0>' has no driver
Partition Implementation Status
-----
No Partitions were found in this design.
-----
NGDBUILD Design Results Summary:
  Number of errors:    0
  Number of warnings:  32
Writing NGD file "MIPSPL_FPGA3.ngd" ...
Writing NGDBUILD log file "MIPSPL_FPGA3.bld"...
NGDBUILD done.
C:\eMIPS_Project\eMIPS_PR\TISA>map -uc ..\..\Sources\Constraints\mips_fp.ucf -ol
high -pr b -timing -xe c MIPSPL_FPGA3.ngd
Release 8.2.01i_PR_2 - Map JTRS.I32
Copyright (c) 1995-2006 Xilinx, Inc. All rights reserved.
Using target part "4vlx25ff668-10".

```

iii. Wait for the process to complete.



```

C:\ Visual Studio 2005 Command Prompt
*See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:      13,801 out of 21,504 64%
Number used as logic:          13,331
Number used as a route-thru:    460
Number used as Shift registers: 10
Number of bonded IOBs:         145 out of 448 32%
Number of BUFG/BUFGCTRLs:       7 out of 32 21%
Number used as BUFGs:           4
Number used as BUFGCTRLs:       3
Number of FIFO16/RAMB16s:       5 out of 72 6%
Number used as FIFO16s:         0
Number used as RAMB16s:         5
Number of DCM_ADUs:             2 out of 8 25%
Number of PMCDs:               1 out of 4 25%

Number of hard macros:          71
Total equivalent gate count for design (not including hard macros): 488,831
Additional JTAG gate count for IOBs: 6,960
Peak Memory Usage: 683 MB

Mapping completed.
See MAP report file "MIPSPL_FPGA3.mrp" for details.
C:\eMIPS_Project\eMIPS_PR\TISA>_

```

e. Route the Base (TISA) Logic on the FPGA Hardware

- i. Type the command: "par -w -uc <floorplanned ucf file> -ol high -pl high -rl high -xe c <input file> <output file>" In our case:

"par -w -uc ..\..\Sources\Constraints\mips_fp.ucf -ol high -pl high -rl high -xe c MIPSPL_FPGA3.ncd MIPSPL_FPGA3_BASE_ROUTED.ncd"


```

C:\ Visual Studio 2005 Command Prompt
*See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:      13,801 out of 21,504 64%
Number used as logic:          13,331
Number used as a route-thru:   460
Number used as Shift registers: 10
Number of bonded IOBs:        145 out of 448 32%
Number of BUFG/BUFGCTRLs:      7 out of 32 21%
Number used as BUFGs:          4
Number used as BUFGCTRLs:      3
Number of FIFO16/RAMB16s:      5 out of 72 6%
Number used as FIFO16s:        0
Number used as RAMB16s:        5
Number of DCM_ADUs:            2 out of 8 25%
Number of PMCDs:               1 out of 4 25%

Number of hard macros:         71
Total equivalent gate count for design (not including hard macros): 488,831
Additional JTAG gate count for IOBs: 6,960
Peak Memory Usage: 683 MB

Mapping completed.
See MAP report file "MIPSPL_FPGA3.mrp" for details.

C:\eMIPS_Project\emIPS_PR\TISA>par -w -uc ..\..\Sources\Constraints\mips_fp.ucf -
ol high -pl high -rl high -xe c MIPSPL_FPGA3.ncd MIPSPL_FPGA3_BASE_ROUTED.ncd_

```

- ii. Press Enter. Some warnings and info messages will appear. Most of them are expected. This may take some time depending on your system, between 30 to 90 minutes.

```

C:\ Visual Studio 2005 Command Prompt - par -w -uc ..\..\Sources\Constraints\mips_fp.ucf -ol ...
WARNING:Par:69 - Option -pl overrides some effects of -ol.
WARNING:Par:69 - Option -rl overrides some effects of -ol.
INFO:Par:338 -
Extra Effort Level "c"ontinue is not a runtime optimized effort level. It is
intended to be used for designs that are
not meeting timing but where the designer wants the tools to continue iterati
ng on the design until no further design
speed improvements are possible. This can result in very long runtimes since
the tools will continue improving the
design even if the time specs can not be met. If you are looking for the best
possible design speed available from a
long but reasonable runtime use Extra Effort Level "n"ormal. It will stop it
erating on the design when the design
speed improvements have shrunk to the point that the time specs are not expec
ted to be met.

Physical Constraints file: MIPSPL_FPGA3.pcf.
Logical Constraints file: ..\..\Sources\Constraints\mips_fp.ucf.
Loading device for application Rf_Device from file '4vlx25.nph' in environment C
:\Xilinx\8.2splpr.

```

- iii. Wait for the process to complete.

```

C:\ Visual Studio 2005 Command Prompt
are routed. If all signals are
reported as completely routed above then the resultant design is as expected for
this flow. DRC may report these pins
as unrouted.

Total REAL time to PAR completion: 16 mins 4 secs
Total CPU time to PAR completion: 14 mins 44 secs

Peak Memory Usage: 621 MB

Placer: Placement generated during map.
Routing: Completed - No errors found.
Timing: Completed - No errors found.

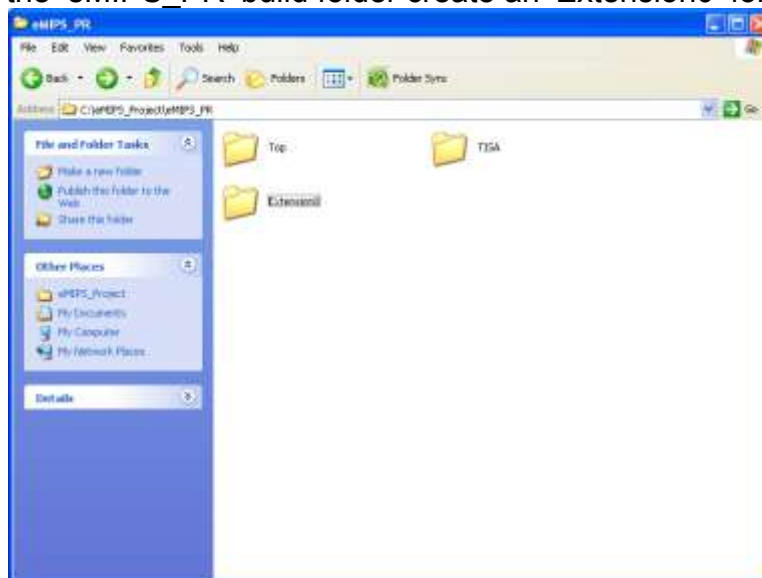
Number of error messages: 0
Number of warning messages: 15
Number of info messages: 3

Writing design to file MIPSPL_FPGA3_BASE_ROUTED.ncd

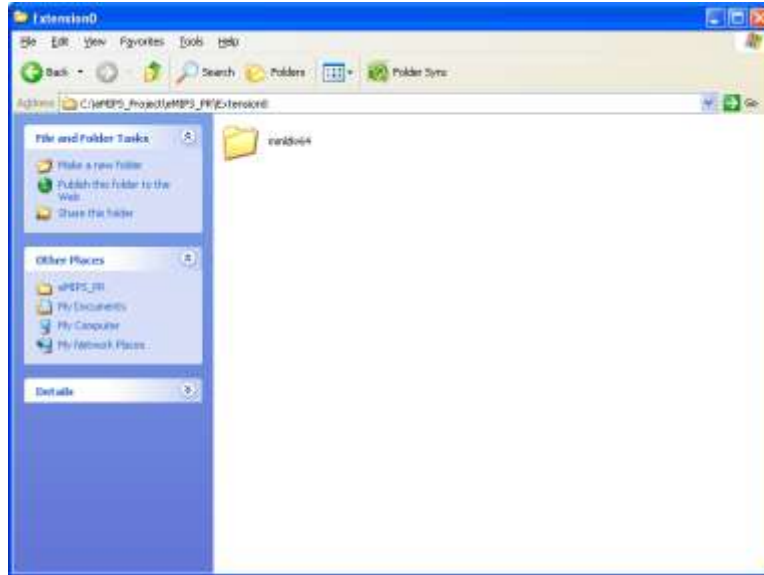
PAR done!
C:\eMIPS_Project\emIPS_PR\TISA>_

```

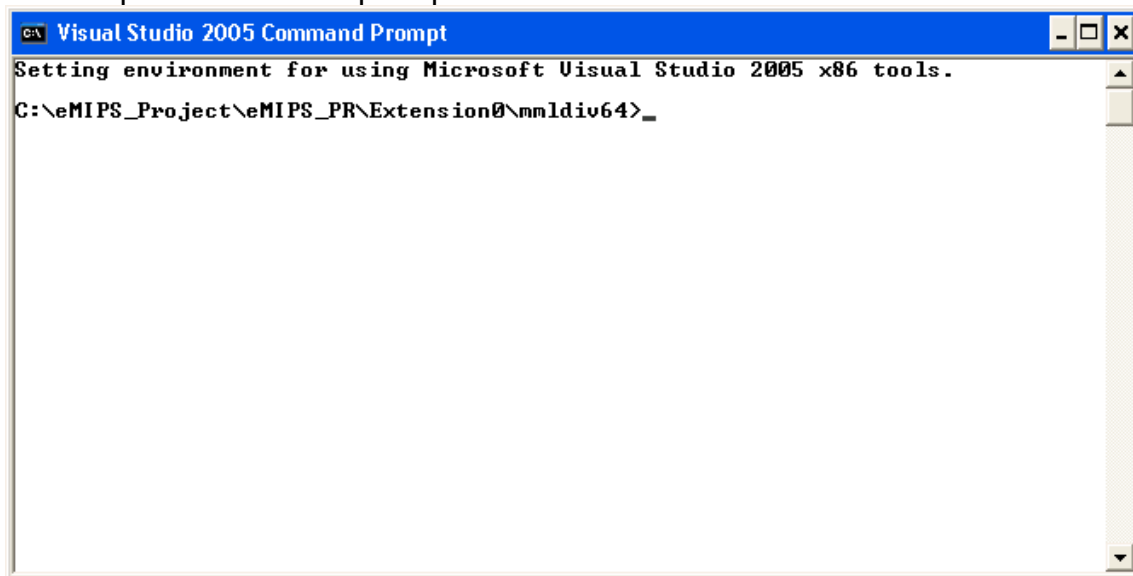
9. Build the Reconfigurable Region (Extension)
 - a. Inside the 'eMIPS_PR' build folder create an 'Extension0' folder



- b. Inside the 'Extension0' folder create a folder with the same name as your Extension. For this example we will use 'mmldiv64'.

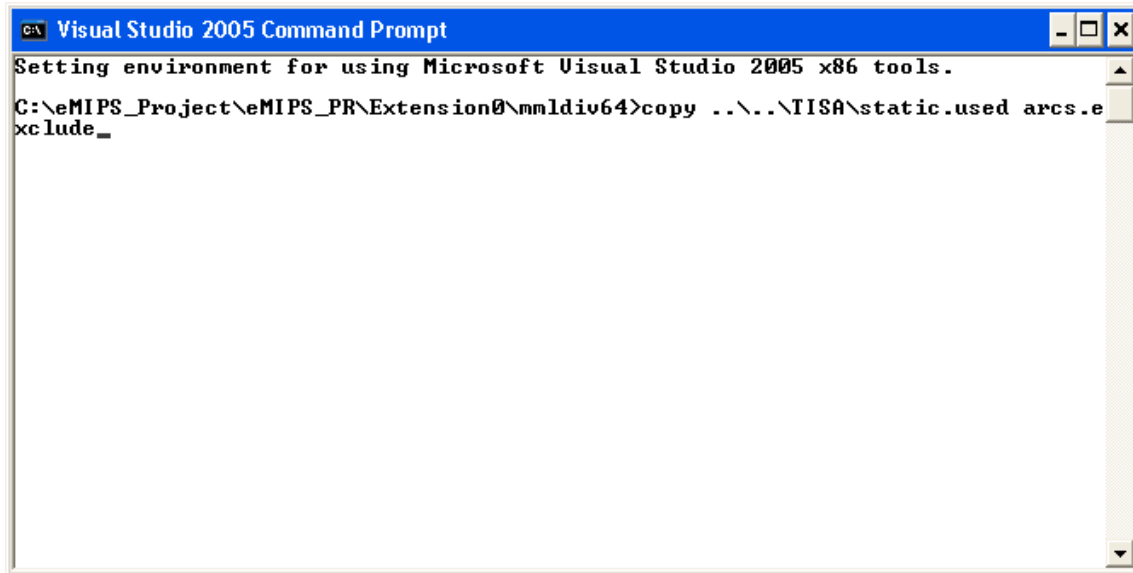


c. Open a command prompt within this folder.



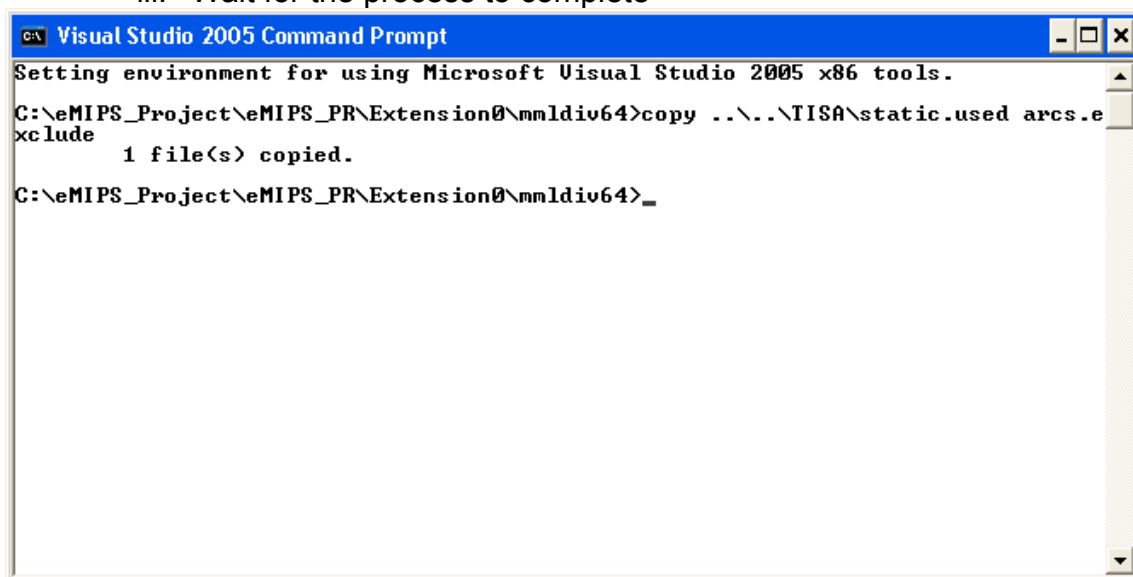
d. Copy the routing database from the Base Design Build.

i. Type the command: *"copy ..\..\TISA\static.used arcs.exclude"*



```
C:\ Visual Studio 2005 Command Prompt
Setting environment for using Microsoft Visual Studio 2005 x86 tools.
C:\eMIPS_Project\eMIPS_PR\Extension0\mmldiv64>copy ..\..\TISA\static.used arcs.e
xclude_
```

- ii. Press Enter.
- iii. Wait for the process to complete

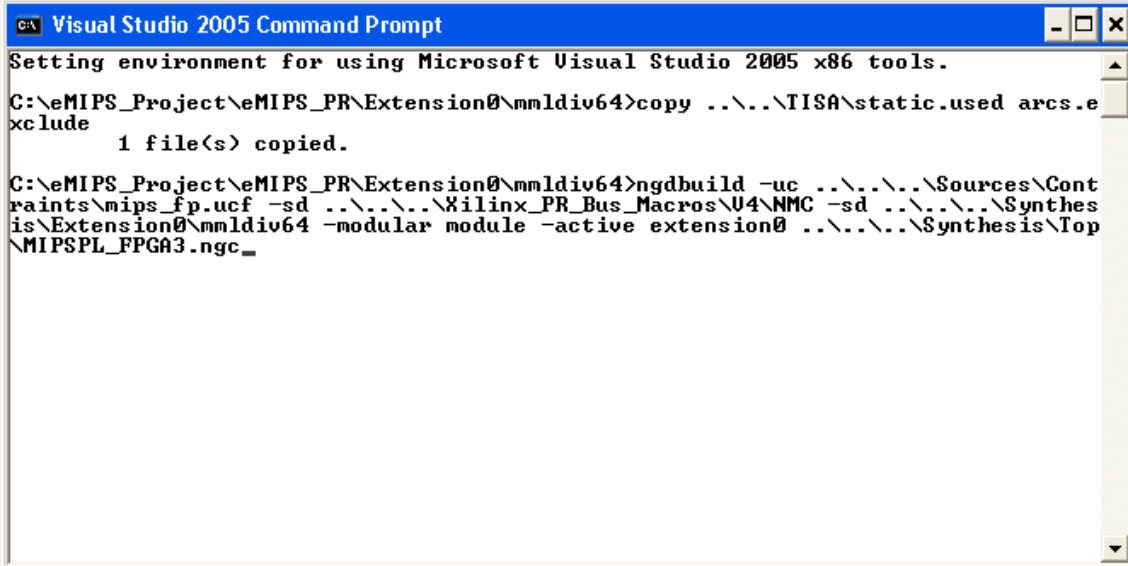


```
C:\ Visual Studio 2005 Command Prompt
Setting environment for using Microsoft Visual Studio 2005 x86 tools.
C:\eMIPS_Project\eMIPS_PR\Extension0\mmldiv64>copy ..\..\TISA\static.used arcs.e
xclude
1 file(s) copied.
C:\eMIPS_Project\eMIPS_PR\Extension0\mmldiv64>_
```

e. Generate the Reconfigurable Region Netlist (Extension)

- i. Type the command: “ngdbuild -uc <floorplanned ucf file> -sd <location of Bus Macro files> -sd <location of Extension synthesis files> -modular module -active extension0 <input file>”. In our case:

*“ngdbuild -uc ..\..\..\Sources\Constraints\mips_fp.ucf -sd ..\..\..\Xilinx_PR_Bus_Macros\V4\NMC
-sd ..\..\..\Synthesis\Extension0\mmldiv64 -modular module -active extension0
..\..\..\Synthesis\Top\MIPSPL_FPGA3.ngc”*



```

C:\ Visual Studio 2005 Command Prompt
Setting environment for using Microsoft Visual Studio 2005 x86 tools.
C:\eMIPS_Project\emIPS_PR\Extension0\mmldiv64>copy ..\..\TISA\static.used arcs.e
xclude
1 file(s) copied.
C:\eMIPS_Project\emIPS_PR\Extension0\mmldiv64>ngdbuild -uc ..\..\..\Sources\Cont
raints\mips_fp.ucf -sd ..\..\..\Xilinx_PR_Bus_Macros\U4\NMC -sd ..\..\..\Synthes
is\Extension0\mmldiv64 -modular module -active extension0 ..\..\..\Synthesis\Top
\MIPSPL_FPGA3.ngc_

```

- ii. Press Enter. Some warnings and info messages will appear. Most of them are expected.

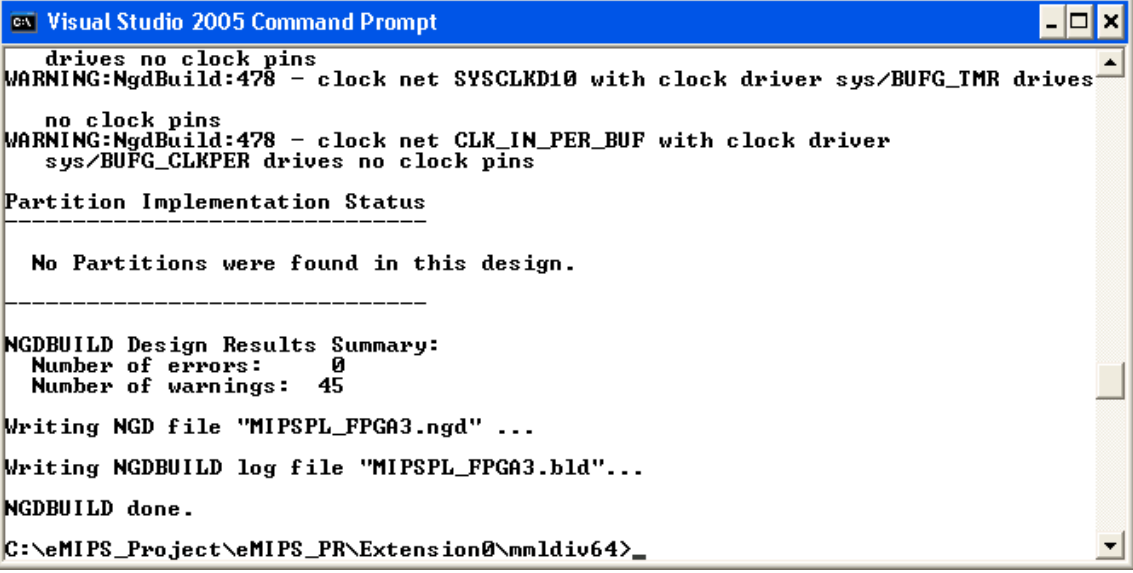


```

C:\ Visual Studio 2005 Command Prompt - ngdbuild -uc ..\..\..\Sources\Constraints\mips_fp.uc...
Setting environment for using Microsoft Visual Studio 2005 x86 tools.
C:\eMIPS_Project\emIPS_PR\Extension0\mmldiv64>copy ..\..\TISA\static.used arcs.e
xclude
1 file(s) copied.
C:\eMIPS_Project\emIPS_PR\Extension0\mmldiv64>ngdbuild -uc ..\..\..\Sources\Cont
raints\mips_fp.ucf -sd ..\..\..\Xilinx_PR_Bus_Macros\U4\NMC -sd ..\..\..\Synthes
is\Extension0\mmldiv64 -modular module -active extension0 ..\..\..\Synthesis\Top
\MIPSPL_FPGA3.ngc
Release 8.2.01i_PR_2 - ngdbuild JTRS.I32
Copyright (c) 1995-2006 Xilinx, Inc. All rights reserved.
Command Line: ngdbuild -uc ..\..\..\Sources\Constraints\mips_fp.ucf -sd
..\..\..\Xilinx_PR_Bus_Macros\U4\NMC -sd ..\..\..\Synthesis\Extension0\mmldiv64
-modular module -active extension0 ..\..\..\Synthesis\Top\MIPSPL_FPGA3.ngc
Reading NGO file 'C:/eMIPS_Project/Synthesis/Top/MIPSPL_FPGA3.ngc' ...
-

```

- iii. Wait for the process to complete.



```

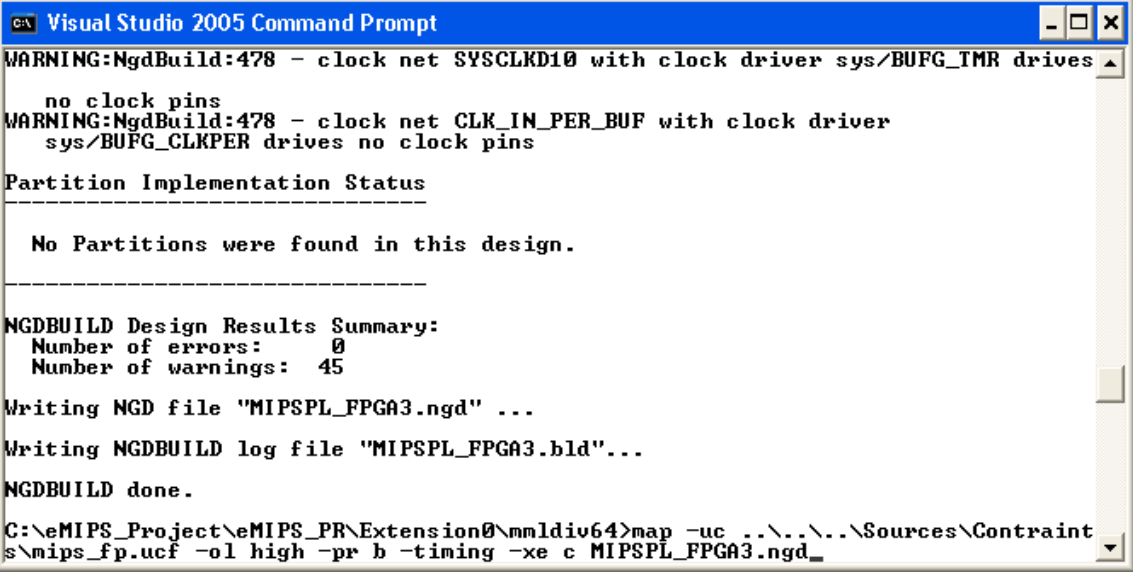
C:\ Visual Studio 2005 Command Prompt
drives no clock pins
WARNING:NgdBuild:478 - clock net SYSCLKD10 with clock driver sys/BUFG_TMR drives
no clock pins
WARNING:NgdBuild:478 - clock net CLK_IN_PER_BUF with clock driver
sys/BUFG_CLKPER drives no clock pins
Partition Implementation Status
-----
No Partitions were found in this design.
-----
NGDBUILD Design Results Summary:
Number of errors: 0
Number of warnings: 45
Writing NGD file "MIPSPL_FPGA3.ngd" ...
Writing NGDBUILD log file "MIPSPL_FPGA3.bld"...
NGDBUILD done.
C:\eMIPS_Project\emIPS_PR\Extension0\mmldiv64>_

```

f. Place/Map the Reconfigurable Region (Extension) Logic to the FPGA hardware.

- i. Type the command: "map -uc <floorplanned ucf file> -ol high -pr b -timing -xe c <input file>". In our case:

"map -uc ..\..\..\Sources\Constraints\mips_fp.ucf -ol high -pr b -timing -xe c MIPSPL_FPGA3.ngd"



```

C:\ Visual Studio 2005 Command Prompt
WARNING:NgdBuild:478 - clock net SYSCLKD10 with clock driver sys/BUFG_TMR drives
no clock pins
WARNING:NgdBuild:478 - clock net CLK_IN_PER_BUF with clock driver
sys/BUFG_CLKPER drives no clock pins
Partition Implementation Status
-----
No Partitions were found in this design.
-----
NGDBUILD Design Results Summary:
Number of errors: 0
Number of warnings: 45
Writing NGD file "MIPSPL_FPGA3.ngd" ...
Writing NGDBUILD log file "MIPSPL_FPGA3.bld"...
NGDBUILD done.
C:\eMIPS_Project\emIPS_PR\Extension0\mmldiv64>map -uc ..\..\..\Sources\Constraint
s\mips_fp.ucf -ol high -pr b -timing -xe c MIPSPL_FPGA3.ngd_

```

- ii. Press Enter. Some warnings and info messages will appear. Most of them are expected. This may take some time depending on your system, between 30 to 90 minutes.

```

C:\ Visual Studio 2005 Command Prompt - map -uc ..\..\..\Sources\Constraints\mips_fp.ucf -ol ...
sys/BUFG_CLKPER drives no clock pins
Partition Implementation Status
-----
No Partitions were found in this design.
-----
NGDBUILD Design Results Summary:
  Number of errors:    0
  Number of warnings:  45
Writing NGD file "MIPSPL_FPGA3.ngd" ...
Writing NGDBUILD log file "MIPSPL_FPGA3.bld"...
NGDBUILD done.
C:\eMIPS_Project\emIPS_PR\Extension0\mmldiv64>map -uc ..\..\..\Sources\Constraint
s\mips_fp.ucf -ol high -pr b -timing -xe c MIPSPL_FPGA3.ngd
Release 8.2.01i_PR_2 - Map JTRS.I32
Copyright (c) 1995-2006 Xilinx, Inc. All rights reserved.
Using target part "4vlx25ff668-10".

```

iii. Wait for the process to complete.

```

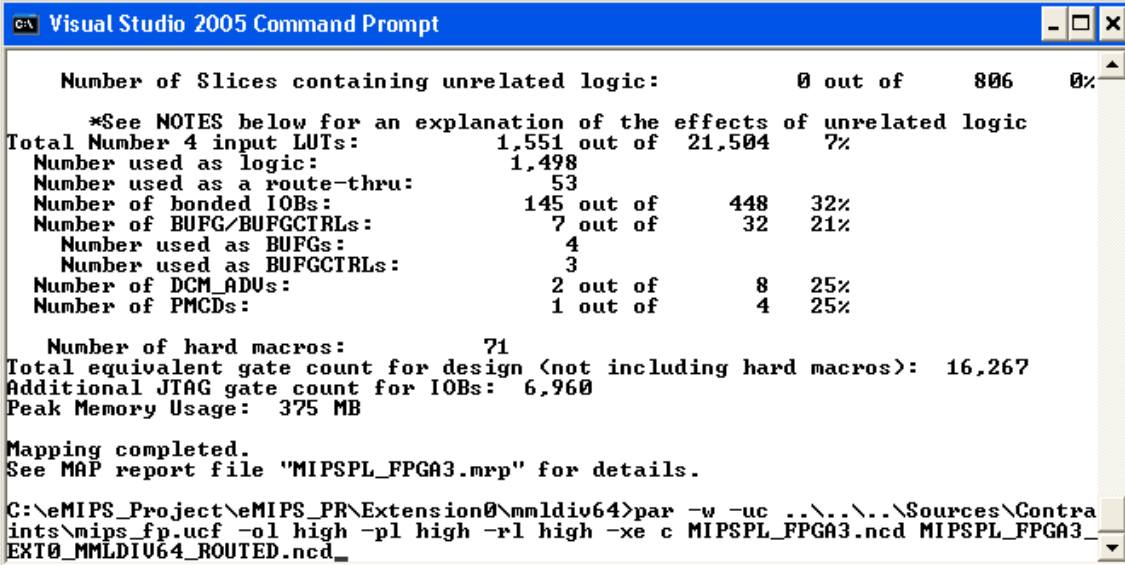
C:\ Visual Studio 2005 Command Prompt
Number of occupied Slices:      806 out of 10,752  7%
Number of Slices containing only related logic:  806 out of 806 100%
Number of Slices containing unrelated logic:      0 out of 806  0%
*See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:      1,551 out of 21,504  7%
Number used as logic:           1,498
Number used as a route-thru:    53
Number of bonded IOBs:         145 out of 448  32%
Number of BUFG/BUFGCTRLs:       7 out of 32  21%
Number used as BUFGs:           4
Number used as BUFGCTRLs:       3
Number of DCM_ADUs:             2 out of 8  25%
Number of PMCDs:               1 out of 4  25%
Number of hard macros:          71
Total equivalent gate count for design (not including hard macros): 16,267
Additional JTAG gate count for IOBs: 6,960
Peak Memory Usage: 375 MB
Mapping completed.
See MAP report file "MIPSPL_FPGA3.mrp" for details.
C:\eMIPS_Project\emIPS_PR\Extension0\mmldiv64>_

```

g. Route the Reconfigurable Region (Extension) Logic on the FPGA hardware.

- i. Type the command: "par -w -uc <floorplanned ucf file> -ol high -pl high -rl high -xe c <input file> <output file>". In our case:

"par -w -uc ..\..\..\Sources\Constraints\mips_fp.ucf -ol high -pl high -rl high -xe c MIPSPL_FPGA3.ncd MIPSPL_FPGA3_EXT0_MMLDIV64_ROUTED.ncd"



```

C:\ Visual Studio 2005 Command Prompt

Number of Slices containing unrelated logic:          0 out of    806    0%
*See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:          1,551 out of  21,504    7%
Number used as logic:          1,498
Number used as a route-thru:          53
Number of bonded IOBs:          145 out of    448    32%
Number of BUFG/BUFGCTRLs:          7 out of    32    21%
Number used as BUFGs:          4
Number used as BUFGCTRLs:          3
Number of DCM_ADUs:          2 out of    8    25%
Number of PMCDs:          1 out of    4    25%

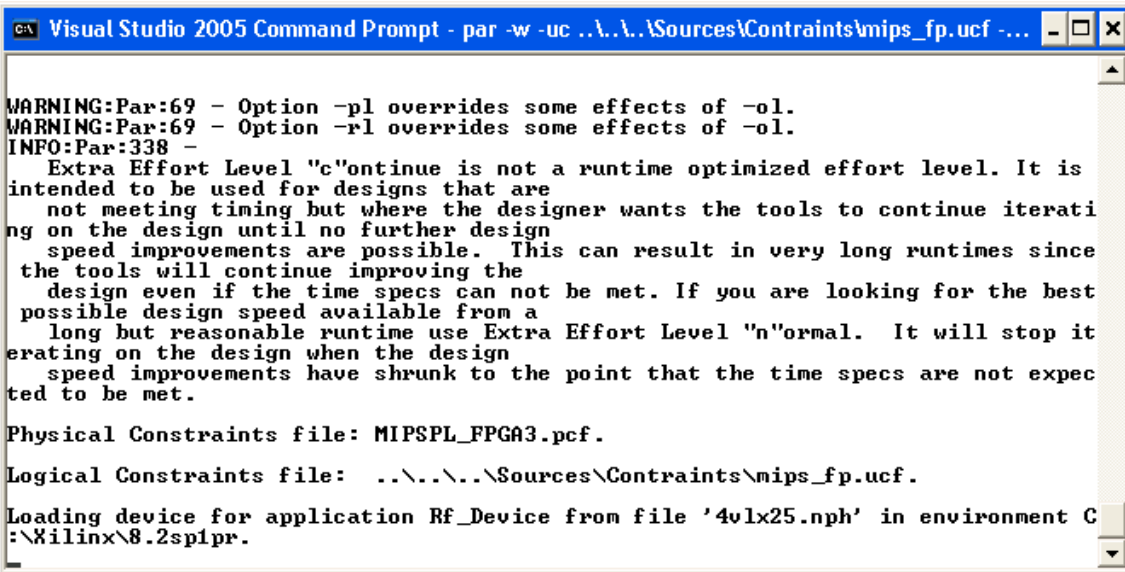
Number of hard macros:          71
Total equivalent gate count for design (not including hard macros):  16,267
Additional JTAG gate count for IOBs:  6,960
Peak Memory Usage:  375 MB

Mapping completed.
See MAP report file "MIPSPL_FPGA3.mrp" for details.

C:\eMIPS_Project\emIPS_PR\Extension0\mmldiv64>par -w -uc ..\..\..\Sources\Contra
ints\mips_fp.ucf -ol high -pl high -rl high -xe c MIPSPL_FPGA3.ncd MIPSPL_FPGA3_
EXT0_MMLDIV64_ROUTED.ncd_

```

- ii. Press Enter. Some warnings and info messages will appear. Most of them are expected. This may take some time depending on your system, between 30 to 90 minutes.



```

C:\ Visual Studio 2005 Command Prompt - par -w -uc ..\..\..\Sources\Constraints\mips_fp.ucf -...

WARNING:Par:69 - Option -pl overrides some effects of -ol.
WARNING:Par:69 - Option -rl overrides some effects of -ol.
INFO:Par:338 -
  Extra Effort Level "c"ontinue is not a runtime optimized effort level. It is
  intended to be used for designs that are
  not meeting timing but where the designer wants the tools to continue iterati
  ng on the design until no further design
  speed improvements are possible. This can result in very long runtimes since
  the tools will continue improving the
  design even if the time specs can not be met. If you are looking for the best
  possible design speed available from a
  long but reasonable runtime use Extra Effort Level "n"ormal. It will stop it
  erating on the design when the design
  speed improvements have shrunk to the point that the time specs are not expec
  ted to be met.

Physical Constraints file: MIPSPL_FPGA3.pcf.
Logical Constraints file: ..\..\..\Sources\Constraints\mips_fp.ucf.
Loading device for application Rf_Device from file '4vlx25.nph' in environment C
:\Xilinx\8.2splpr.

```

- iii. Wait for the process to complete.


```

Visual Studio 2005 Command Prompt
are routed. If all signals are
reported as completely routed above then the resultant design is as expected for
this flow. DRC may report these pins
as unrouted.

Total REAL time to PAR completion: 7 mins 5 secs
Total CPU time to PAR completion: 6 mins 28 secs

Peak Memory Usage: 333 MB

Placer: Placement generated during map.
Routing: Completed - No errors found.
Timing: Completed - No errors found.

Number of error messages: 0
Number of warning messages: 476
Number of info messages: 3

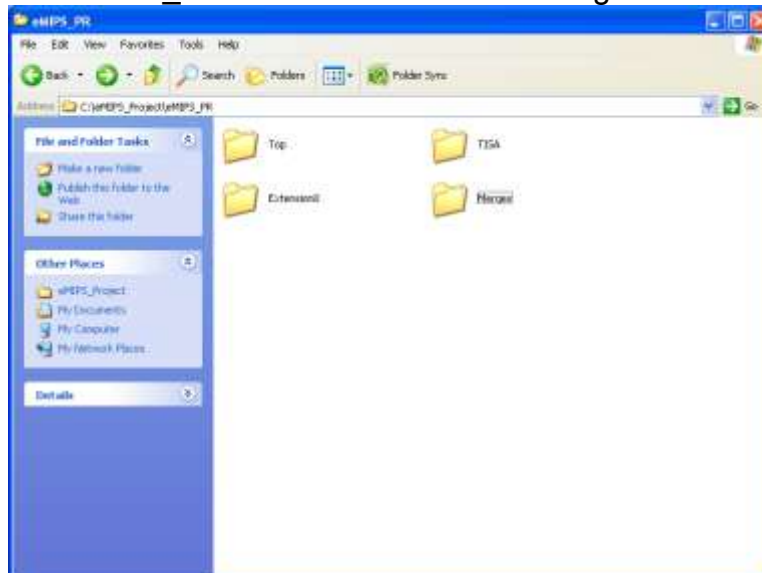
Writing design to file MIPSPL_FPGA3_EXT0_MMLDIV64_ROUTED.ncd

PAR done!

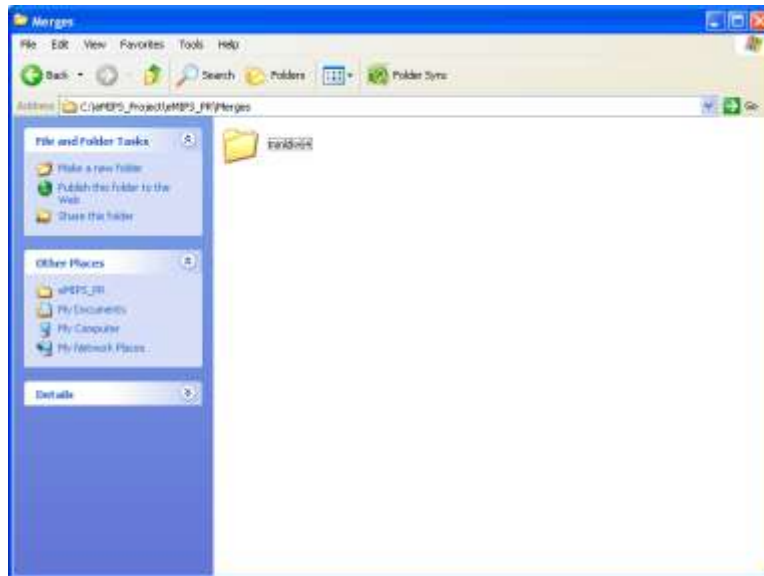
C:\eMIPS_Project\emIPS_PR\Extension0\mmldiv64>

```

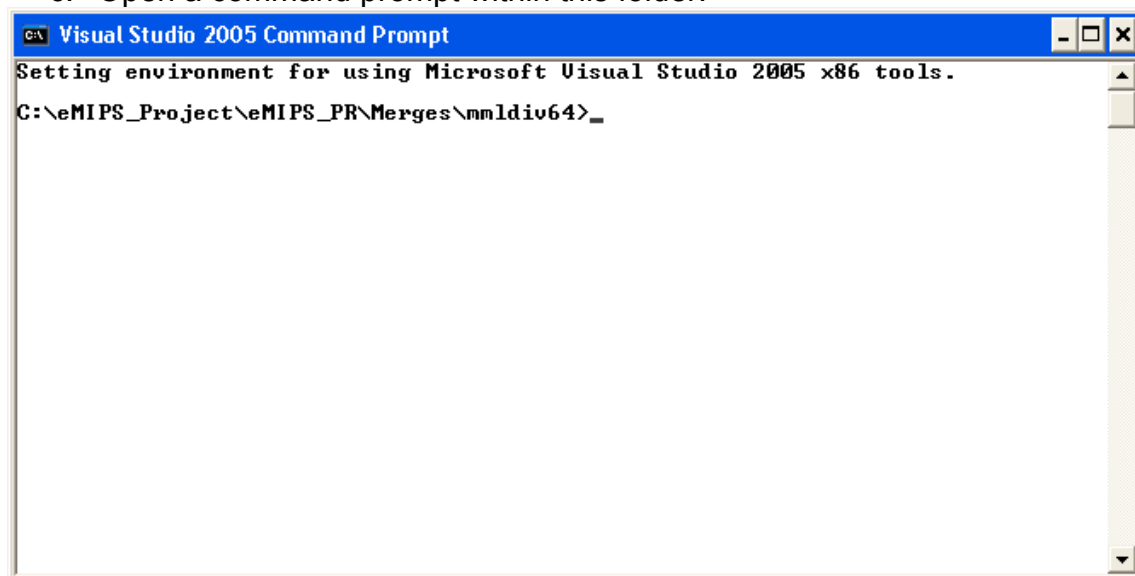
- h. Repeat steps 9.b through 9.g for any additional Extensions you wish to implement for the Extension0 slot.
10. Merge Base and Reconfigurable Region (TISA + Extension)
 - a. Inside the 'eMIPS_PR' build folder create a 'Merges' folder.



- b. Inside the 'Merges' folder create a folder with the same name as you Extension. For this example we will use 'mmldiv64'.



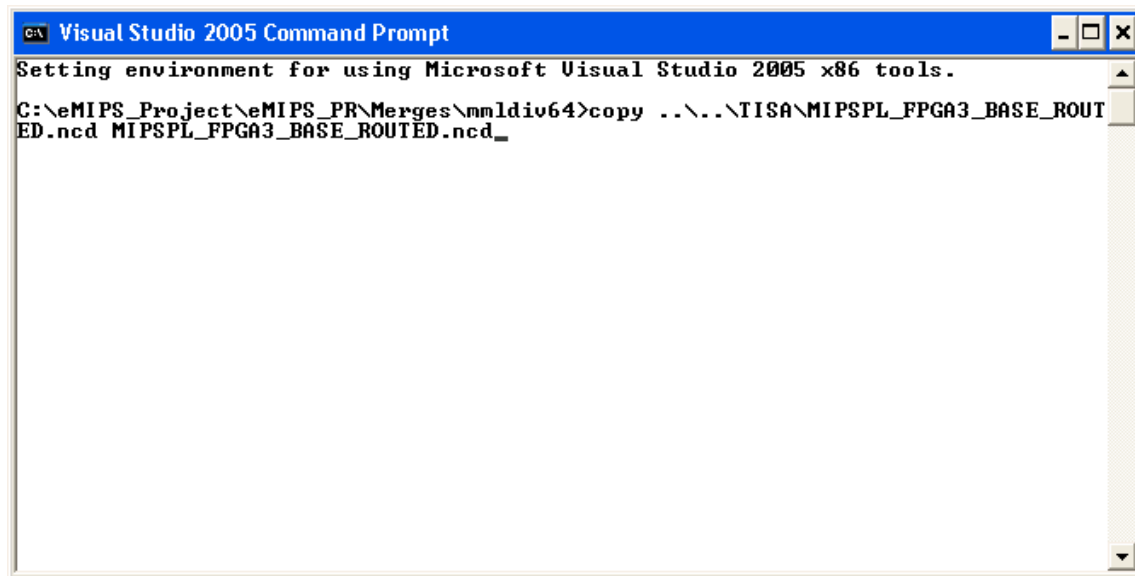
c. Open a command prompt within this folder.



d. Copy the Routed Base Design (TISA) to the Extension merge folder.

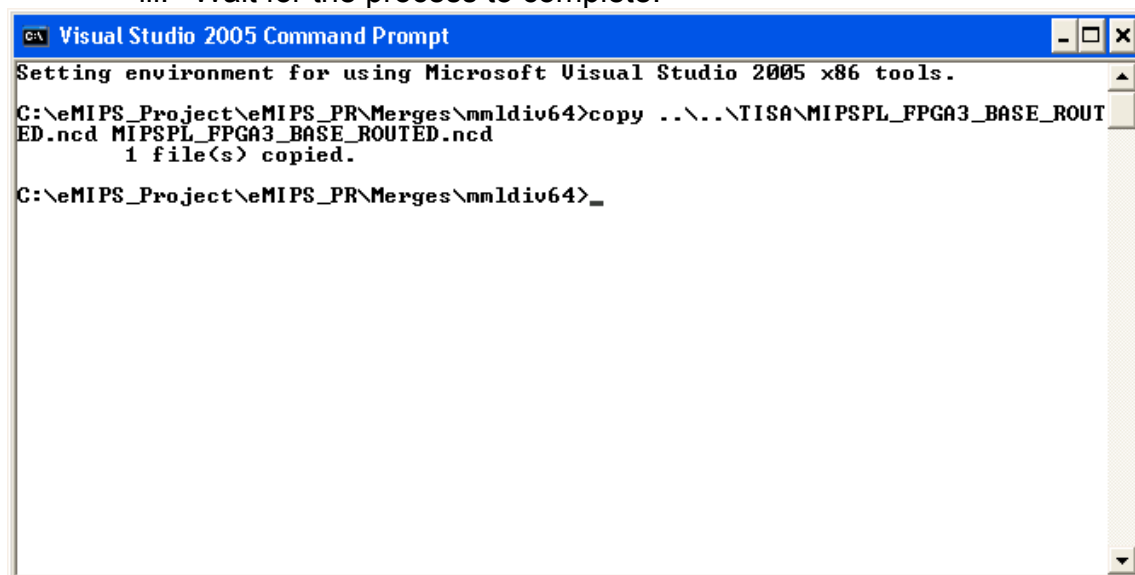
i. Type the command: "copy <Routed Base Design> <Routed Base Design>". In our case:

copy ..\..\TISA\MIPSPL_FPGA3_BASE_ROUTED.ncd MIPSPL_FPGA3_BASE_ROUTED.ncd



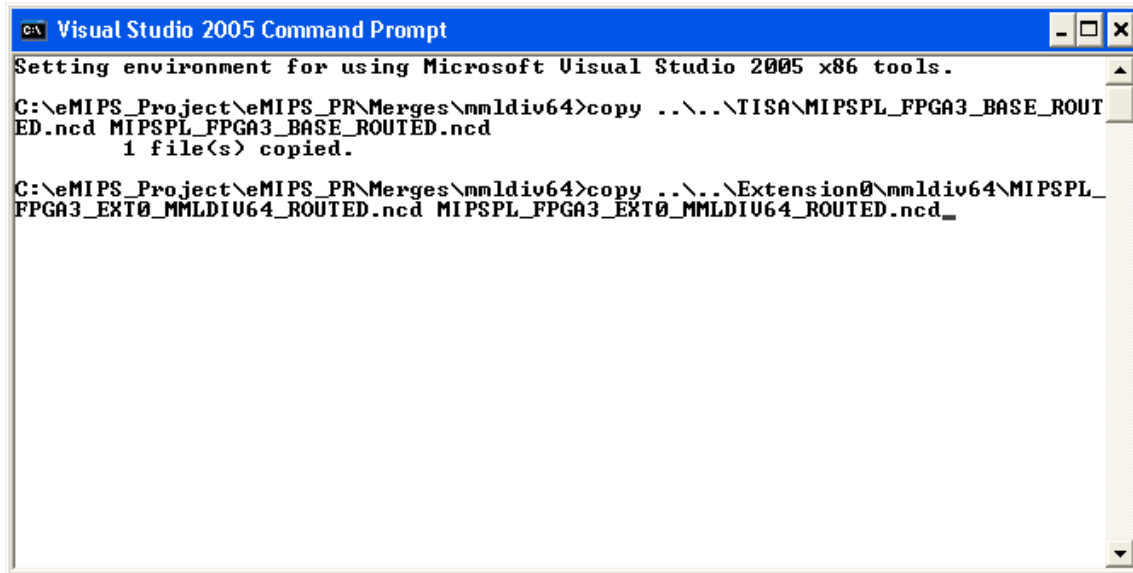
```
Visual Studio 2005 Command Prompt
Setting environment for using Microsoft Visual Studio 2005 x86 tools.
C:\eMIPS_Project\emIPS_PR\Merges\mmldiv64>copy ..\..\TISA\MIPSPL_FPGA3_BASE_ROUT
ED.ncd MIPSPL_FPGA3_BASE_ROUTED.ncd_
```

- ii. Press Enter.
- iii. Wait for the process to complete.



```
Visual Studio 2005 Command Prompt
Setting environment for using Microsoft Visual Studio 2005 x86 tools.
C:\eMIPS_Project\emIPS_PR\Merges\mmldiv64>copy ..\..\TISA\MIPSPL_FPGA3_BASE_ROUT
ED.ncd MIPSPL_FPGA3_BASE_ROUTED.ncd
1 file(s) copied.
C:\eMIPS_Project\emIPS_PR\Merges\mmldiv64>
```

- e. Copy the Routed Reconfigurable Region (Extension) to the Extension merge folder.
 - i. Type the command: “copy <Routed Reconfigurable Region> <Routed Reconfigurable Region>”. In our case:
“copy ..\..\Extension0\mmldiv64\MIPSPL_FPGA3_EXT0_MMLDIV64_ROUTED.ncd MIPSPL_FPGA3_EXT0_MMLDIV64_ROUTED.ncd”

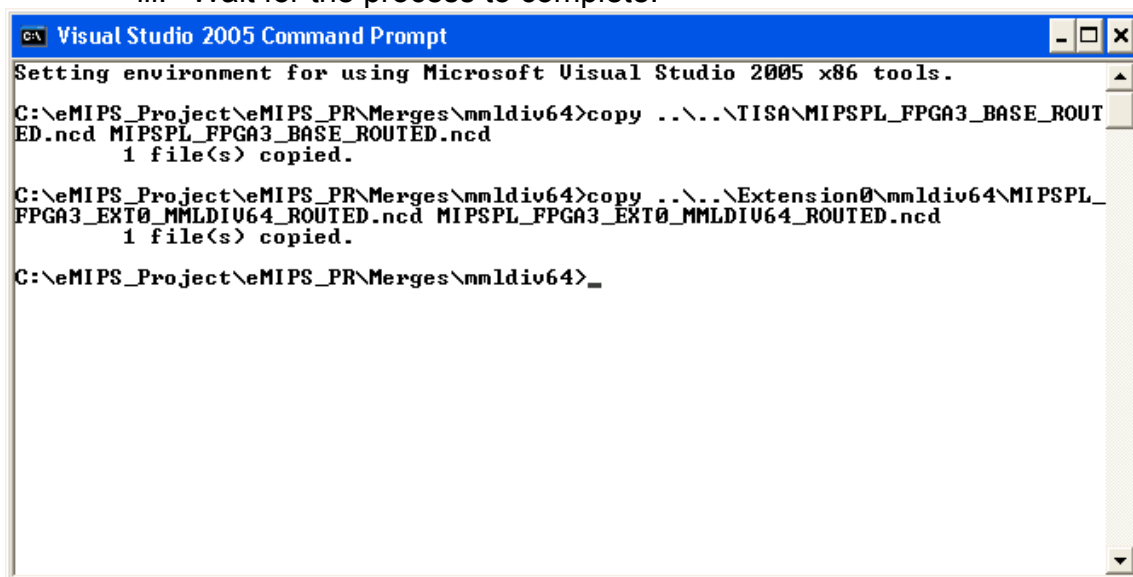


```
C:\ Visual Studio 2005 Command Prompt
Setting environment for using Microsoft Visual Studio 2005 x86 tools.

C:\eMIPS_Project\emIPS_PR\Merges\mmldiv64>copy ..\..\TISA\MIPSPL_FPGA3_BASE_ROUT
ED.ncd MIPSPL_FPGA3_BASE_ROUTED.ncd
1 file(s) copied.

C:\eMIPS_Project\emIPS_PR\Merges\mmldiv64>copy ..\..\Extension0\mmldiv64\MIPSPL_
FPGA3_EXT0_MMLDIU64_ROUTED.ncd MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED.ncd
```

- ii. Press Enter.
- iii. Wait for the process to complete.



```
C:\ Visual Studio 2005 Command Prompt
Setting environment for using Microsoft Visual Studio 2005 x86 tools.

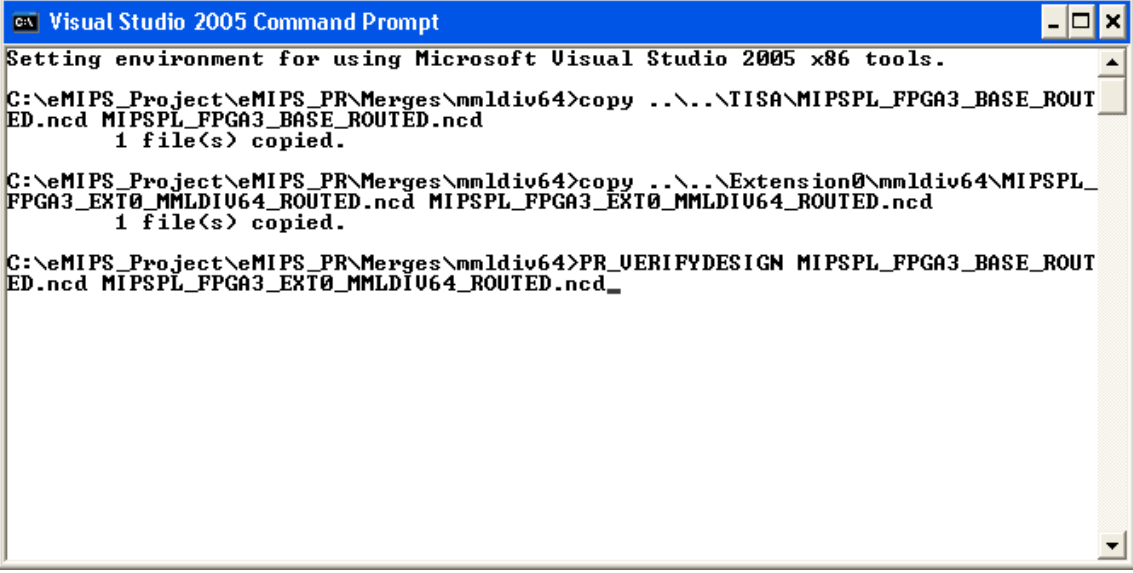
C:\eMIPS_Project\emIPS_PR\Merges\mmldiv64>copy ..\..\TISA\MIPSPL_FPGA3_BASE_ROUT
ED.ncd MIPSPL_FPGA3_BASE_ROUTED.ncd
1 file(s) copied.

C:\eMIPS_Project\emIPS_PR\Merges\mmldiv64>copy ..\..\Extension0\mmldiv64\MIPSPL_
FPGA3_EXT0_MMLDIU64_ROUTED.ncd MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED.ncd
1 file(s) copied.

C:\eMIPS_Project\emIPS_PR\Merges\mmldiv64>
```

- f. Run the Verify Design Script.
 - i. Type the command: "PR_VERIFYDESIGN <Routed Base Design> <Routed Reconfigurable Region>". In our case:

*PR_VERIFYDESIGN MIPSPL_FPGA3_BASE_ROUTED.ncd MIPSPL_FPGA3_EXT0_MMLDI
V64_ROUTED.ncd*



```

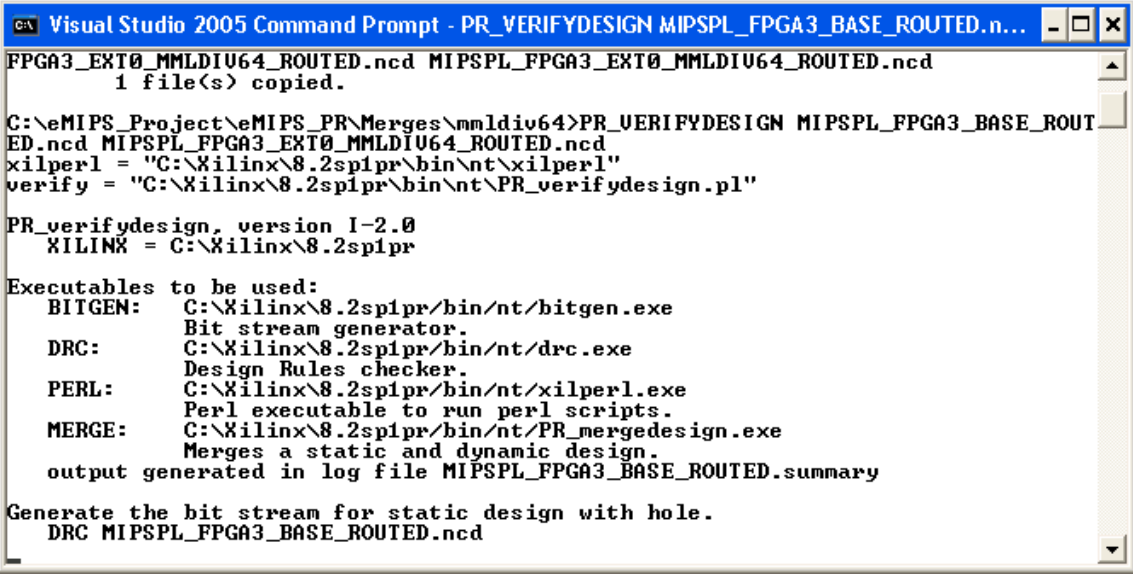
C:\ Visual Studio 2005 Command Prompt
Setting environment for using Microsoft Visual Studio 2005 x86 tools.

C:\eMIPS_Project\emIPS_PR\Merges\mmldiv64>copy ..\..\TISA\MIPSPL_FPGA3_BASE_ROUT
ED.ncd MIPSPL_FPGA3_BASE_ROUTED.ncd
1 file(s) copied.

C:\eMIPS_Project\emIPS_PR\Merges\mmldiv64>copy ..\..\Extension0\mmldiv64\MIPSPL_
FPGA3_EXT0_MMLDIU64_ROUTED.ncd MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED.ncd
1 file(s) copied.

C:\eMIPS_Project\emIPS_PR\Merges\mmldiv64>PR_VERIFYDESIGN MIPSPL_FPGA3_BASE_ROUT
ED.ncd MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED.ncd_
  
```

- ii. Press Enter. This may take some time depending on your system, between 15 to 30 minutes.



```

C:\ Visual Studio 2005 Command Prompt - PR_VERIFYDESIGN MIPSPL_FPGA3_BASE_ROUTED.n...
MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED.ncd MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED.ncd
1 file(s) copied.

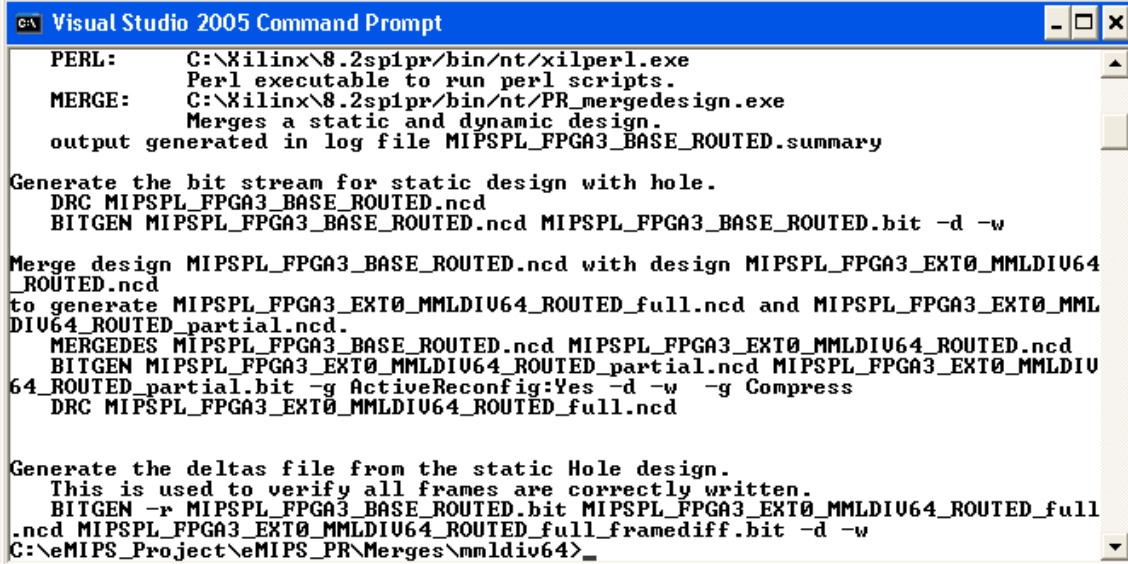
C:\eMIPS_Project\emIPS_PR\Merges\mmldiv64>PR_VERIFYDESIGN MIPSPL_FPGA3_BASE_ROUT
ED.ncd MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED.ncd
xilperl = "C:\Xilinx\8.2sp1pr\bin\nt\xilperl"
verify = "C:\Xilinx\8.2sp1pr\bin\nt\PR_verifydesign.pl"

PR_verifydesign, version 1-2.0
XILINK = C:\Xilinx\8.2sp1pr

Executables to be used:
BITGEN: C:\Xilinx\8.2sp1pr\bin\nt\bitgen.exe
        Bit stream generator.
DRC:    C:\Xilinx\8.2sp1pr\bin\nt\drc.exe
        Design Rules checker.
PERL:   C:\Xilinx\8.2sp1pr\bin\nt\xilperl.exe
        Perl executable to run perl scripts.
MERGE:  C:\Xilinx\8.2sp1pr\bin\nt\PR_mergedesign.exe
        Merges a static and dynamic design.
        output generated in log file MIPSPL_FPGA3_BASE_ROUTED.summary

Generate the bit stream for static design with hole.
DRC MIPSPL_FPGA3_BASE_ROUTED.ncd
  
```

- iii. Wait for the process to complete.



```

C:\ Visual Studio 2005 Command Prompt

PERL:      C:\Xilinx\8.2spi\pr/bin/nt/xilperl.exe
           Perl executable to run perl scripts.
MERGE:      C:\Xilinx\8.2spi\pr/bin/nt/PR_mergedesign.exe
           Merges a static and dynamic design.
           output generated in log file MIPSPL_FPGA3_BASE_ROUTED.summary

Generate the bit stream for static design with hole.
DRC MIPSPL_FPGA3_BASE_ROUTED.ncd
BITGEN MIPSPL_FPGA3_BASE_ROUTED.ncd MIPSPL_FPGA3_BASE_ROUTED.bit -d -w

Merge design MIPSPL_FPGA3_BASE_ROUTED.ncd with design MIPSPL_FPGA3_EXT0_MMLDIU64
_ROUTED.ncd
to generate MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED_full.ncd and MIPSPL_FPGA3_EXT0_MML
DIU64_ROUTED_partial.ncd.
MERGEDES MIPSPL_FPGA3_BASE_ROUTED.ncd MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED.ncd
BITGEN MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED_partial.ncd MIPSPL_FPGA3_EXT0_MMLDIU
64_ROUTED_partial.bit -g ActiveReconfig:Yes -d -w -g Compress
DRC MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED_full.ncd

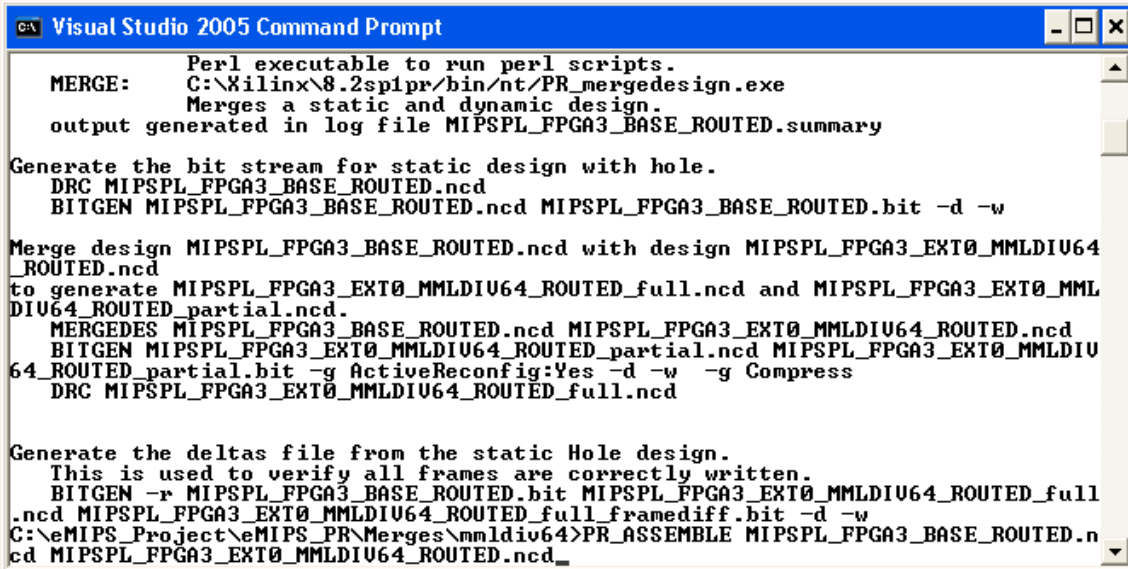
Generate the deltas file from the static Hole design.
This is used to verify all frames are correctly written.
BITGEN -r MIPSPL_FPGA3_BASE_ROUTED.bit MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED_full
.ncd MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED_full_framediff.bit -d -w
C:\eMIPS_Project\emIPS_PR\Merges\mmldiv64>_

```

g. Run the Assemble Script.

- i. Type the command: “PR_ASSEMBLE <Routed Base Design> <Routed Reconfigurable Region>”. In our case:

PR_ASSEMBLE MIPSPL_FPGA3_BASE_ROUTED.ncd MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED.ncd



```

C:\ Visual Studio 2005 Command Prompt

MERGE:      Perl executable to run perl scripts.
           C:\Xilinx\8.2spi\pr/bin/nt/PR_mergedesign.exe
           Merges a static and dynamic design.
           output generated in log file MIPSPL_FPGA3_BASE_ROUTED.summary

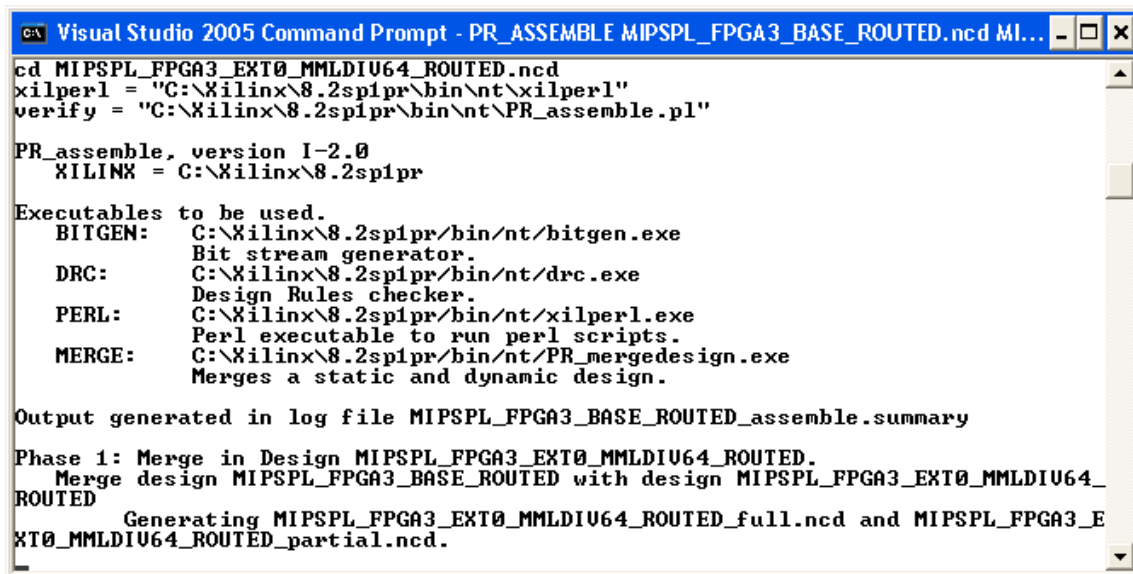
Generate the bit stream for static design with hole.
DRC MIPSPL_FPGA3_BASE_ROUTED.ncd
BITGEN MIPSPL_FPGA3_BASE_ROUTED.ncd MIPSPL_FPGA3_BASE_ROUTED.bit -d -w

Merge design MIPSPL_FPGA3_BASE_ROUTED.ncd with design MIPSPL_FPGA3_EXT0_MMLDIU64
_ROUTED.ncd
to generate MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED_full.ncd and MIPSPL_FPGA3_EXT0_MML
DIU64_ROUTED_partial.ncd.
MERGEDES MIPSPL_FPGA3_BASE_ROUTED.ncd MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED.ncd
BITGEN MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED_partial.ncd MIPSPL_FPGA3_EXT0_MMLDIU
64_ROUTED_partial.bit -g ActiveReconfig:Yes -d -w -g Compress
DRC MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED_full.ncd

Generate the deltas file from the static Hole design.
This is used to verify all frames are correctly written.
BITGEN -r MIPSPL_FPGA3_BASE_ROUTED.bit MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED_full
.ncd MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED_full_framediff.bit -d -w
C:\eMIPS_Project\emIPS_PR\Merges\mmldiv64>PR_ASSEMBLE MIPSPL_FPGA3_BASE_ROUTED.n
cd MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED.ncd_

```

- ii. Press Enter. This may take some time depending on your system, between 15 to 30 minutes.



```

C:\ Visual Studio 2005 Command Prompt - PR_ASSEMBLE MIPSPL_FPGA3_BASE_ROUTED.ncd MI...
cd MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED.ncd
xilperl = "C:\Xilinx\8.2sp1pr\bin\nt\xilperl"
verify = "C:\Xilinx\8.2sp1pr\bin\nt\PR_assemble.pl"

PR_assemble, version 1-2.0
  XILINX = C:\Xilinx\8.2sp1pr

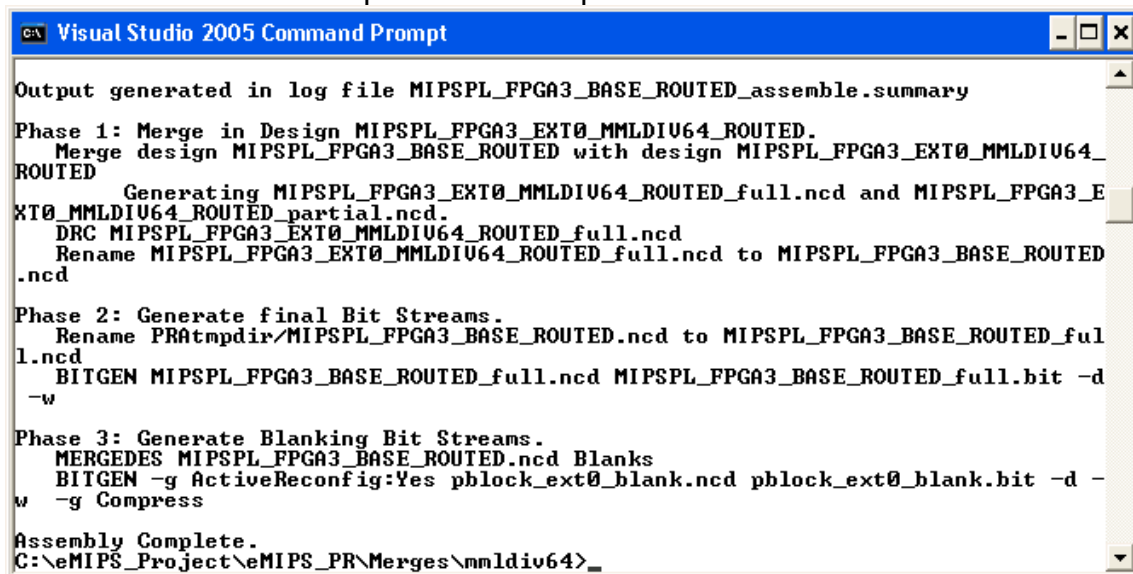
Executables to be used.
  BITGEN: C:\Xilinx\8.2sp1pr\bin\nt\bitgen.exe
           Bit stream generator.
  DRC:    C:\Xilinx\8.2sp1pr\bin\nt\drc.exe
           Design Rules checker.
  PERL:   C:\Xilinx\8.2sp1pr\bin\nt\xilperl.exe
           Perl executable to run perl scripts.
  MERGE:  C:\Xilinx\8.2sp1pr\bin\nt\PR_mergedesign.exe
           Merges a static and dynamic design.

Output generated in log file MIPSPL_FPGA3_BASE_ROUTED_assemble.summary

Phase 1: Merge in Design MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED.
  Merge design MIPSPL_FPGA3_BASE_ROUTED with design MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED
  Generating MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED_full.ncd and MIPSPL_FPGA3_E
XT0_MMLDIU64_ROUTED_partial.ncd.

```

iii. Wait for the process to complete.



```

C:\ Visual Studio 2005 Command Prompt

Output generated in log file MIPSPL_FPGA3_BASE_ROUTED_assemble.summary

Phase 1: Merge in Design MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED.
  Merge design MIPSPL_FPGA3_BASE_ROUTED with design MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED
  Generating MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED_full.ncd and MIPSPL_FPGA3_E
XT0_MMLDIU64_ROUTED_partial.ncd.
  DRC MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED_full.ncd
  Rename MIPSPL_FPGA3_EXT0_MMLDIU64_ROUTED_full.ncd to MIPSPL_FPGA3_BASE_ROUTED
.ncd

Phase 2: Generate final Bit Streams.
  Rename PRAtmpdir\MIPSPL_FPGA3_BASE_ROUTED.ncd to MIPSPL_FPGA3_BASE_ROUTED_ful
l.ncd
  BITGEN MIPSPL_FPGA3_BASE_ROUTED_full.ncd MIPSPL_FPGA3_BASE_ROUTED_full.bit -d
-w

Phase 3: Generate Blanking Bit Streams.
  MERGEDES MIPSPL_FPGA3_BASE_ROUTED.ncd Blanks
  BITGEN -g ActiveReconfig:Yes pblock_ext0_blank.ncd pblock_ext0_blank.bit -d -
w -g Compress

Assembly Complete.
C:\eMIPS_Project\eMIPS_PR\Merges\mmldiv64>

```

h. Repeat steps 10.b through 10.g for any additional Extensions you wish to implement for the Extension0 slot.

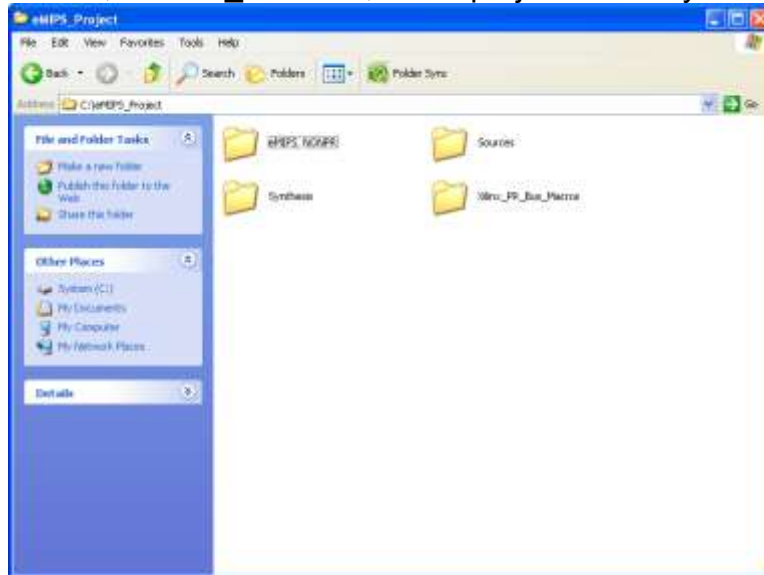
11. Get Configuration Bit Files.

12. Follow the 'Verifying the Configuration Bit Files' procedure (see 3.3) to determine if the Extension has built correctly.

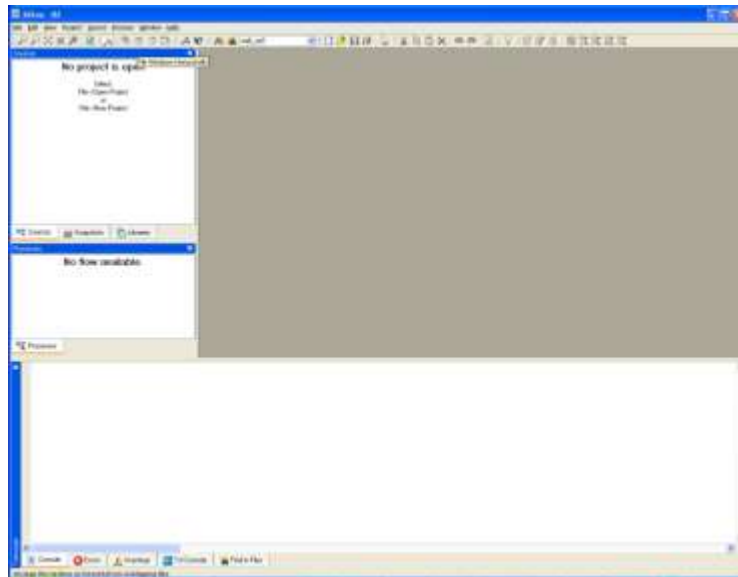
3.2.2 Building the NON-PR Version

The following are the instructions for building the eMIPS microprocessor system without the partial reconfiguration feature, using the Xilinx ISE. By doing this you will not be able to use partial bit files to modify this design during runtime and the Extension instantiated will be static. This build is good for preliminary design and testing without the added complication of the PR flow. It is also recommend if you do not plan to use the PR feature at all and just want a static processor design.

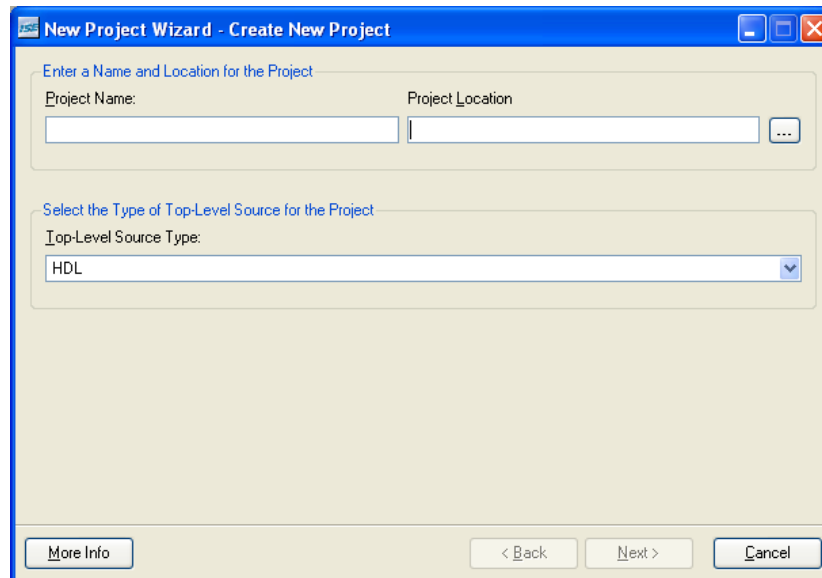
1. If the Project is not yet made, Set up the Project Directory (see 3.1.3)
2. If not already done, Change the Environment to NON-PR Xilinx ISE (see 3.1.1)
3. Create a build folder, 'eMIPS_NONPR', in the project directory.



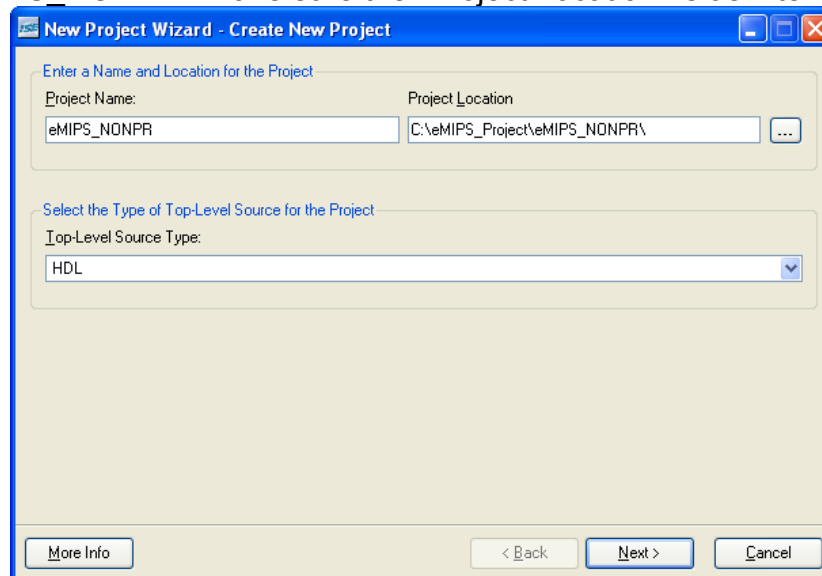
4. Start the NON-PR install of the Xilinx ISE of choice.



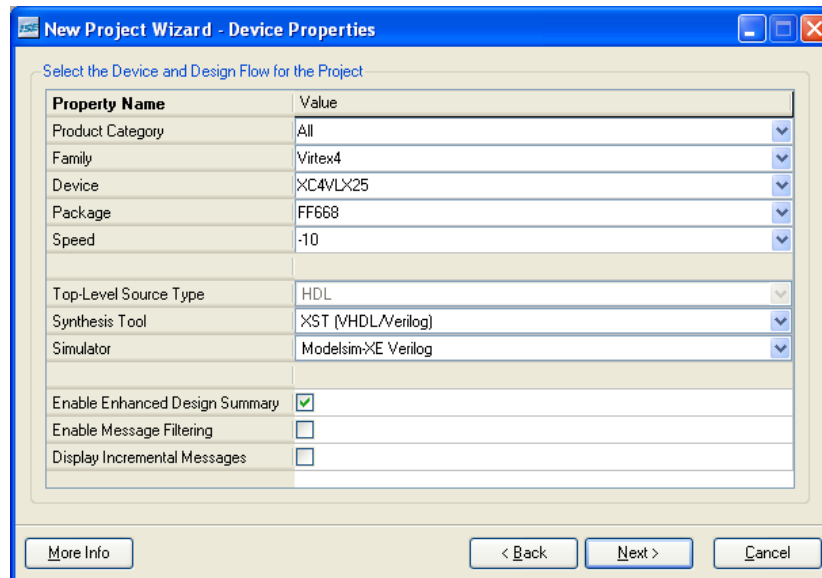
5. Create a new ISE project.
 - a. Click File->New Project in the Menu.
 - b. The 'New Project Wizard' dialog window will appear.



- c. Select the eMIPS_NONPR folder as the project location and name the project 'eMIPS_NONPR'. Make sure the "Project Location" is as intended.



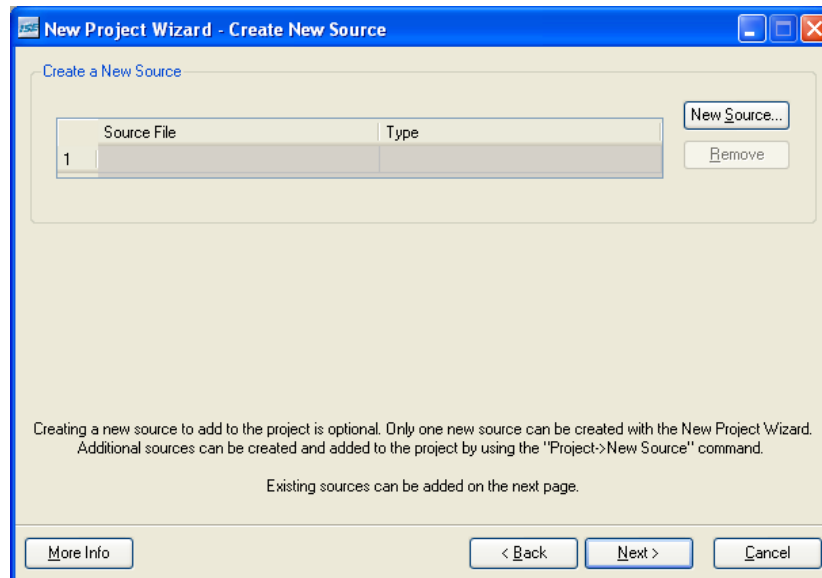
- d. After you have reviewed the entries in the window, click 'Next' to continue.
e. The 'Device Properties' screen should appear.



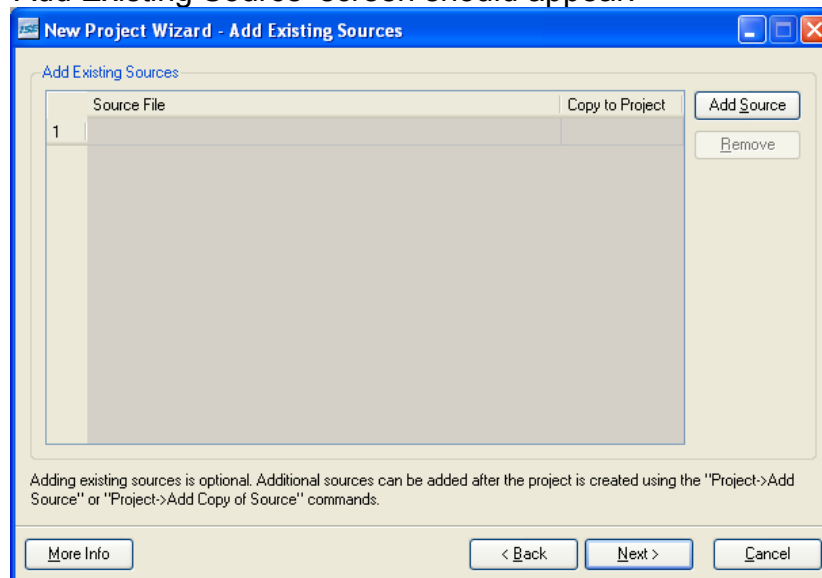
- f. Enter the appropriate settings for your board setup in these fields. For the Xilinx ML401 board, please use the settings in the table below. You may use other synthesis tools other than the XST (Xilinx default) if you chose. However, be aware all of our experiments have used this tool and we cannot vouch for the outcome of another tool.

Property Name	Value
Product Category	All
Family	Virtex4
Device	XC4VLX25
Package	FF668
Speed	-10
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	NA
Enable Enhanced Design Summary	NA
Enable Message Filtering	NA
Display Incremental Messages	NA

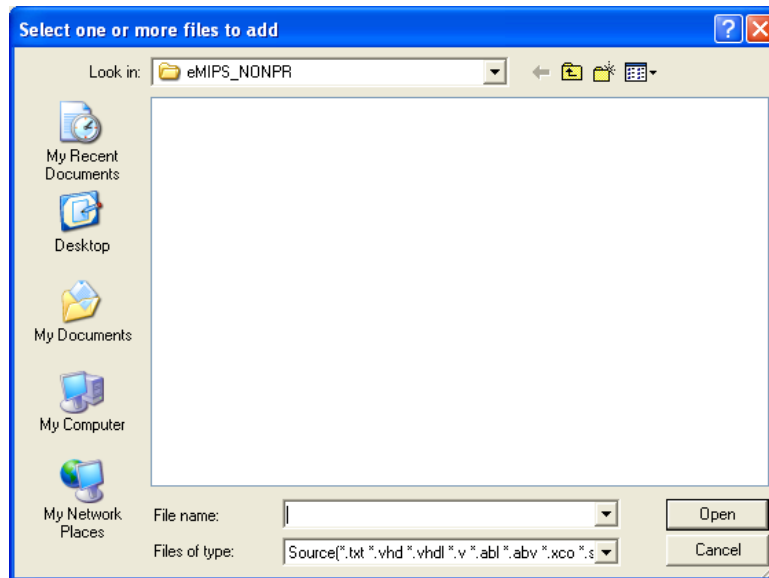
- g. After you have reviewed the entries in the window, click 'Next' continue.
h. The 'Create New Source' screen should appear.



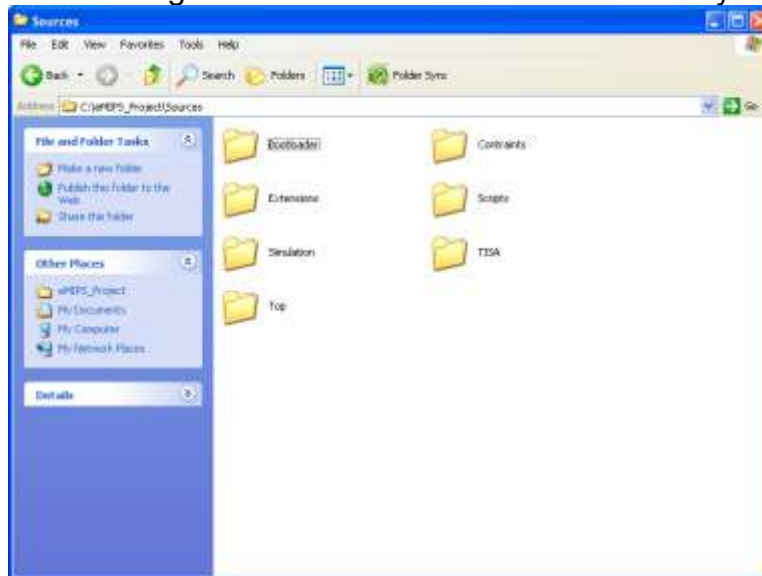
- i. Since you will not be creating any new sources at this time, click 'Next' to continue.
- j. The 'Add Existing Source' screen should appear.



- k. Click the 'Add Source' Button.
- l. The 'Select one or more files to add' dialog should appear.



m. Navigate the dialog window to the eMIPS source directory.

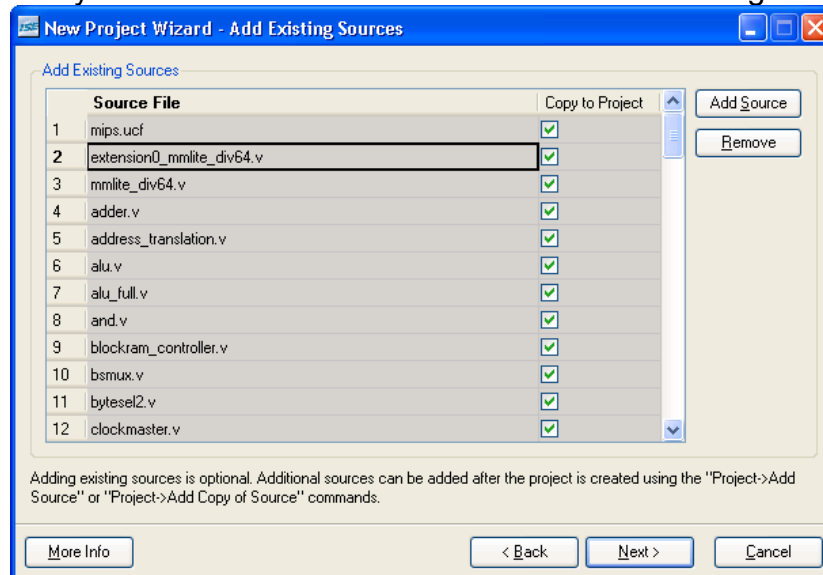


n. Select the files listed in the table below to be added to the project. There are a total of seventy-eight files.

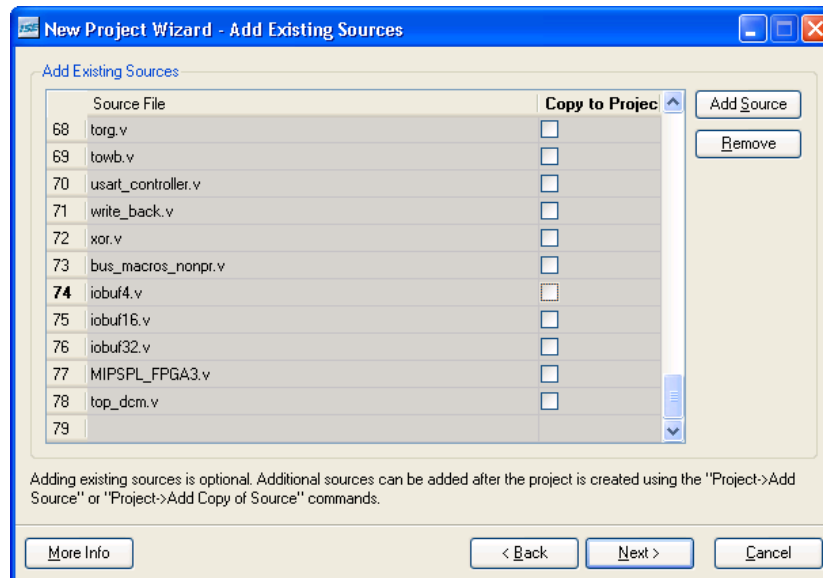
extension0_mmlite_div64.v	lobuf4.v	lobuf16.v
lobuf32.v	MIPSPL_FPGA3.v	top_dcm.v
bus_macros_nonpr.v	TISA.v	mmlite_div64.v
adder.v	address_translation.v	alu.v
alu_full.v	and.v	blockram_controller.v
bsmux.v	bytesel2.v	clockmaster.v
coprocessor0_sim.v	data_align.v	dataforward.v
datapathcontrol.v	decode.v	endianflip.v
equals.v	execute.v	fifo.v
flash_bridge.v	flash_controller.v	flash_interface.v
gpio_controller.v	greaterthan.v	hazard.v
instruction_decode.v	instruction_fetch.v	interrupt_controller.v

lessthan.v	memory_access.v	memory_arbiter.v
memory_bus_front.v	memory_controller.v	memory_interface3.v
multiplier.v	mux.v	or.v
pipeline_arbiter.v	power_management_controller.v	registerblock.v
registerfile4.v	shift2.v	shift.v
signextend16_32.v	sram_bridge.v	sram_controller.v
sram_interface.v	sysace_bridge.v	sysace_controller.v
sysace_interface.v	timer_controller.v	toex.v
tcp0.v	todf.v	toid.v
toext.v	tohz.v	tomem.v
toif.v	toma.v	towb.v
topa.v	torg.v	xor.v
usart_controller.v	write_back.v	mips.ucf
lock.v	memory_management_unit.v	extension_controller.v

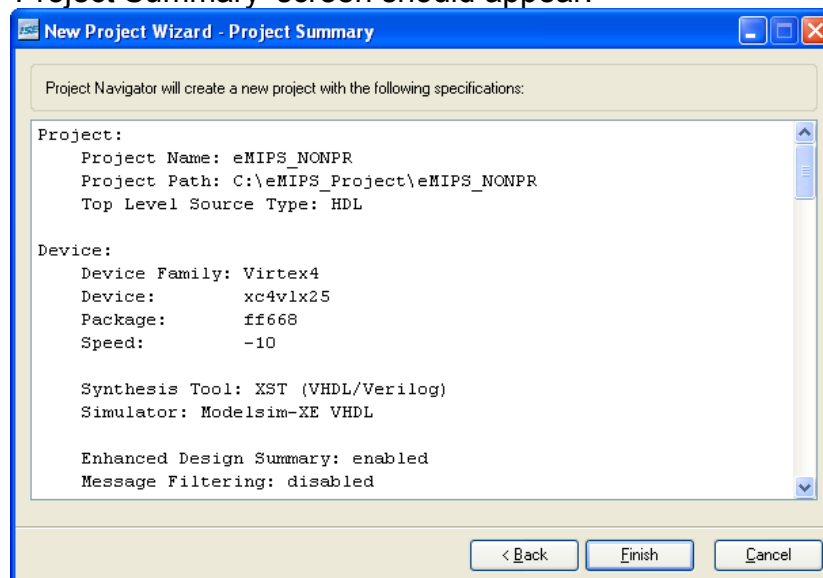
- o. When you are done click 'Open'.
- p. The files you selected should be listed in the 'Add Existing Source' screen.



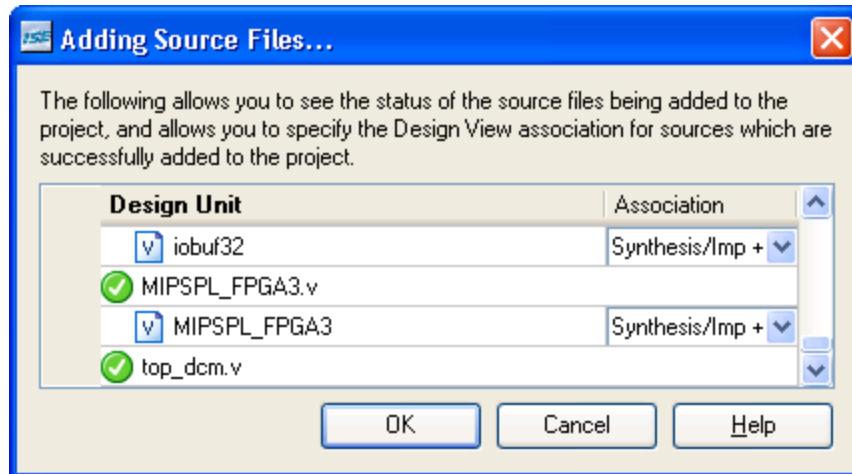
- q. Uncheck the 'Copy to Project' checkbox for each file.



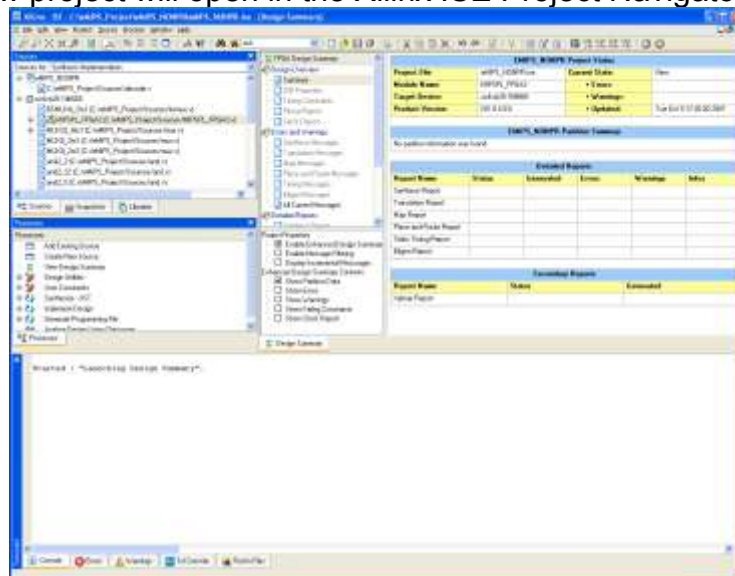
- r. After you have reviewed the entries in the window, click 'Next' continue.
- s. The 'Project Summary' screen should appear.



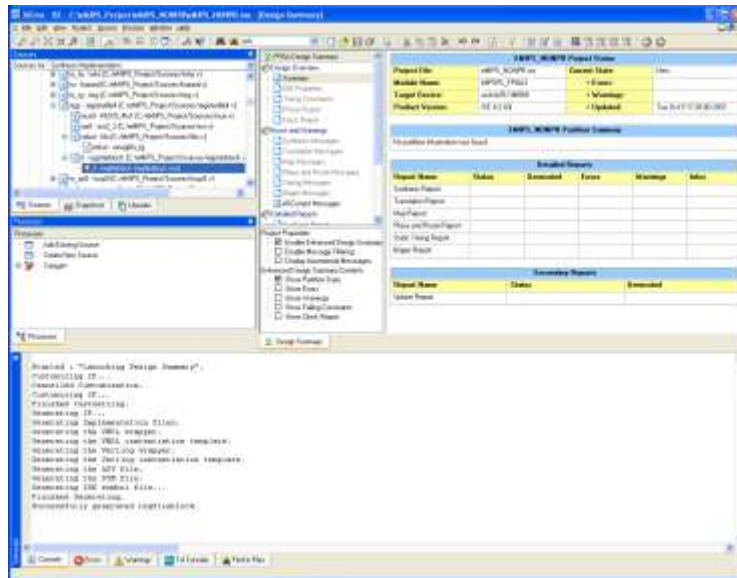
- t. Click finish to complete the project setup.
- u. The 'New Project Wizard' will close and a moment later the 'Adding Source Files' dialog should appear.



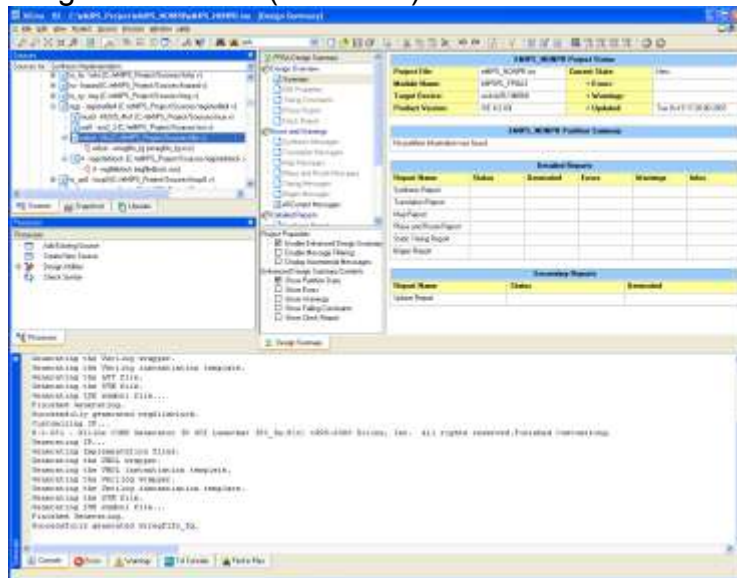
- v. Click 'OK' to continue.
- w. The new project will open in the Xilinx ISE Project Navigator.



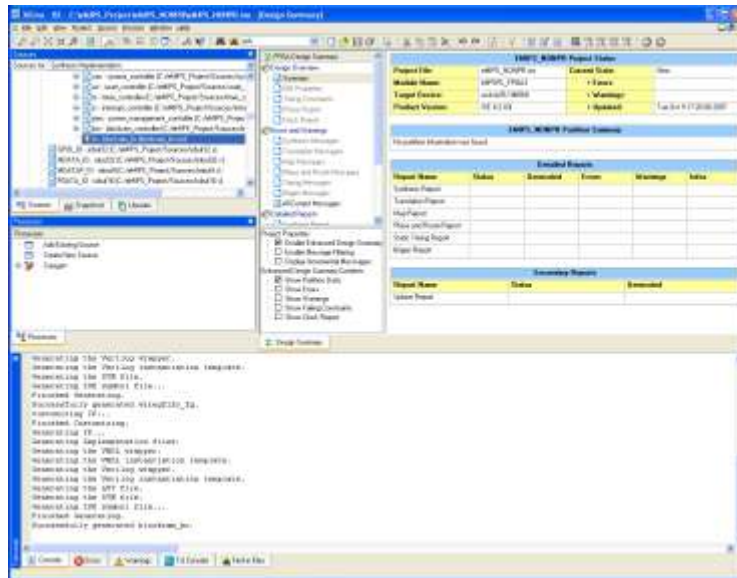
6. The module 'MIPSPL_FPGA3' should be set as the top module in the 'Sources' pane. If not set it as the top module by right-clicking MIPSPL_FPGA3 and selecting 'Set as Top Module'.
7. Add the IP Core Modules
 - a. Add the Register File Blockram (see 3.1.5)



b. Add the Register File Fifo (see 3.1.6).

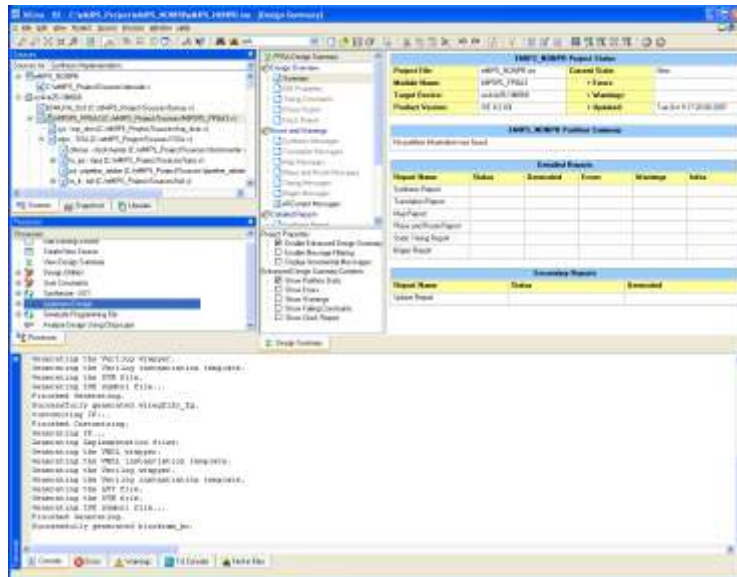


c. Add the Bootloader Blockram (see 3.1.7).

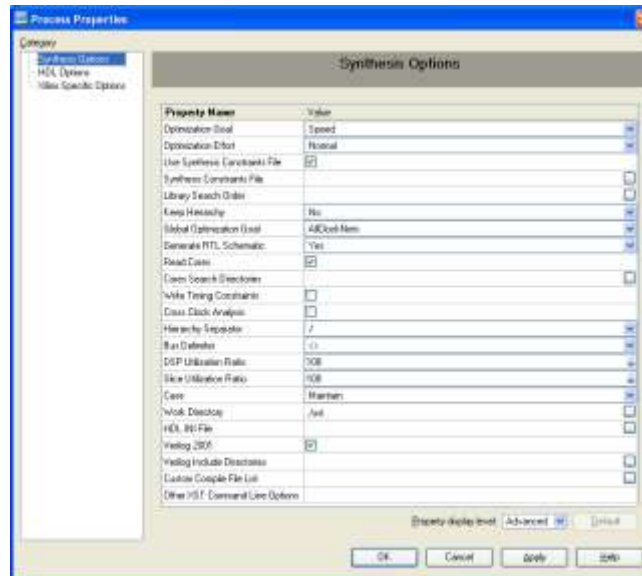


8. Set Synthesis Options

- In the 'Sources' pane, select 'MIPSPL_FPGA3'. Then select 'Synthesize - XST' in the 'Processes' pane. Make sure to select the module and not the project icons.

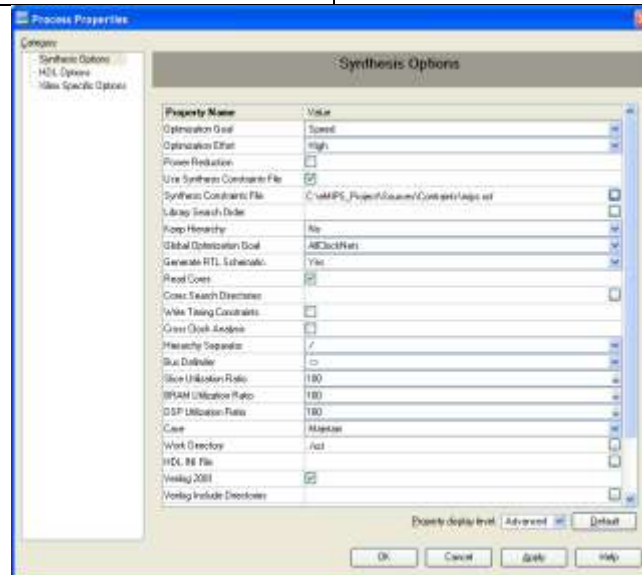


- Right-click on 'Synthesize - XST' and select 'Properties'.
- The 'Synthesis Options' dialog window should appear.



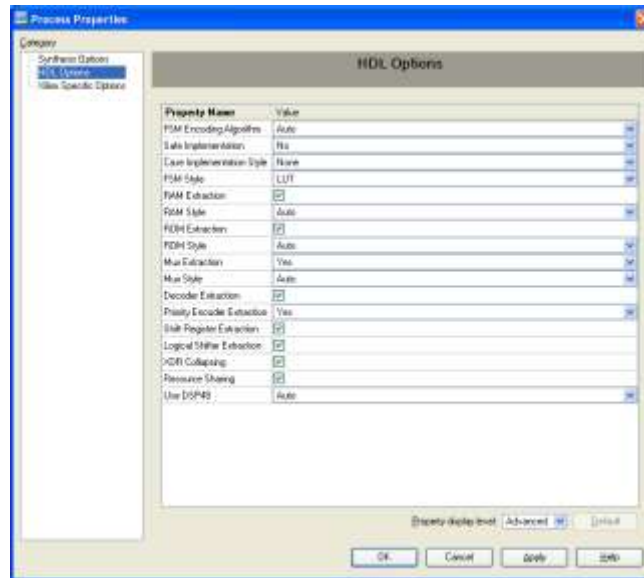
d. Set the values in the table below to the properties in this window.

Property Name	Value
Optimization Goal	Speed
Optimization Effort	High
Use Synthesis Constraints File	Yes
Synthesis Constraints File	<location of sources>\mips.xcf



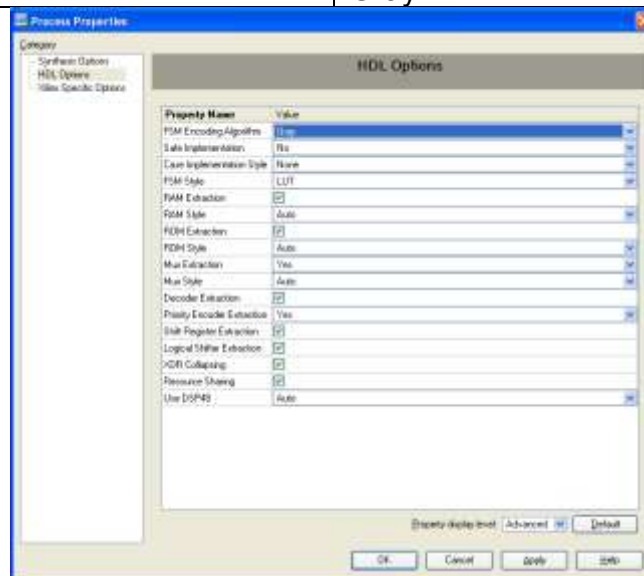
e. After you have reviewed the entries in the window, click 'HDL Options' in the 'Category' pane.

f. The 'HDL Options' dialog window should appear.

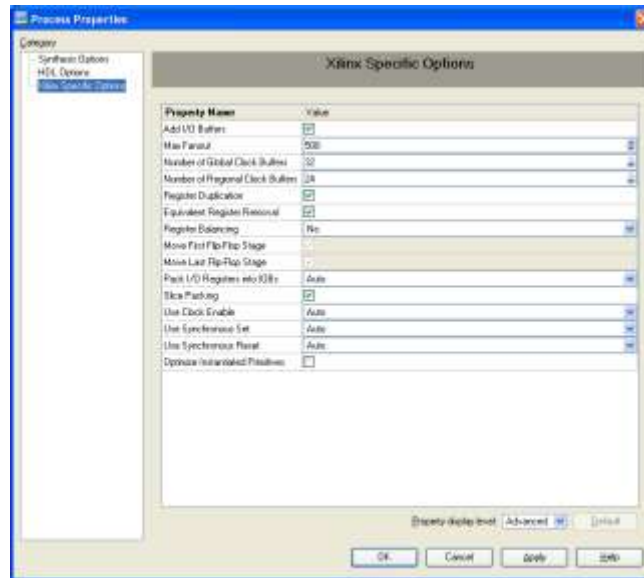


g. Set the values in the table below to the properties in this window.

Property Name	Value
FSM Encoding Algorithm	Gray

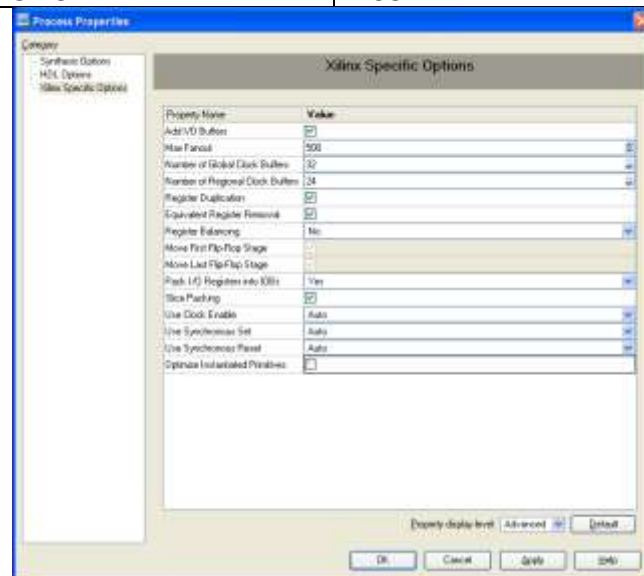


- h. After you have reviewed the entries in the window, click 'Xilinx Specific Options' in the 'Category' pane.
- i. The 'Xilinx Specific Options' dialog window should appear.



j. Set the values in the table below to the properties in this window.

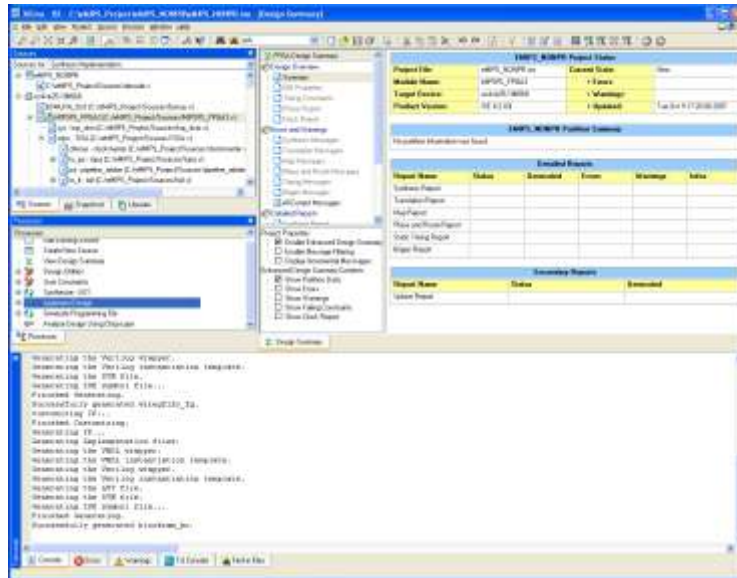
Property Name	Value
Pack I/O Registers into IOBs	Yes



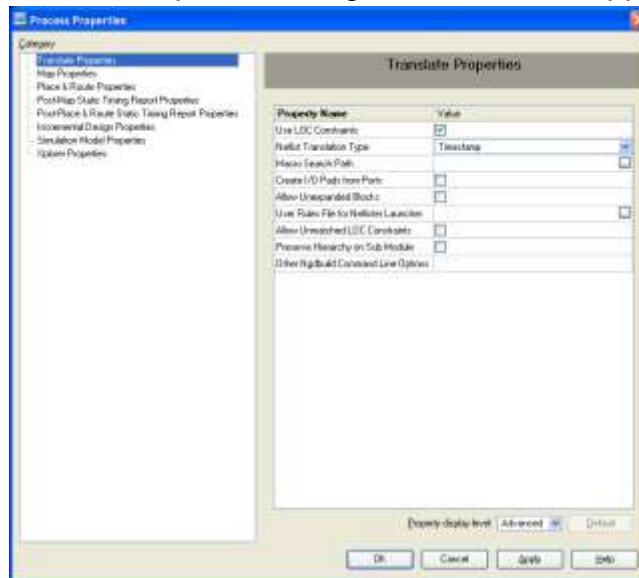
k. After you have reviewed the entries in the window, click 'OK' to continue.

9. Set Implementation Options

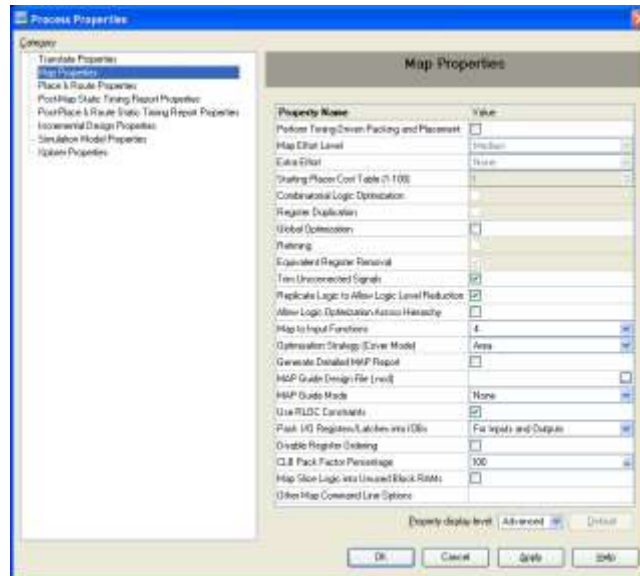
- In the 'Sources' pane, select 'MIPSPL_FPGA3'. Then select 'Implement Design' in the 'Processes' pane. Make sure to select the module and not the project icons.



- b. Right-click on 'Implement Design' and select 'Properties'.
- c. The 'Implementation Options' dialog window should appear.

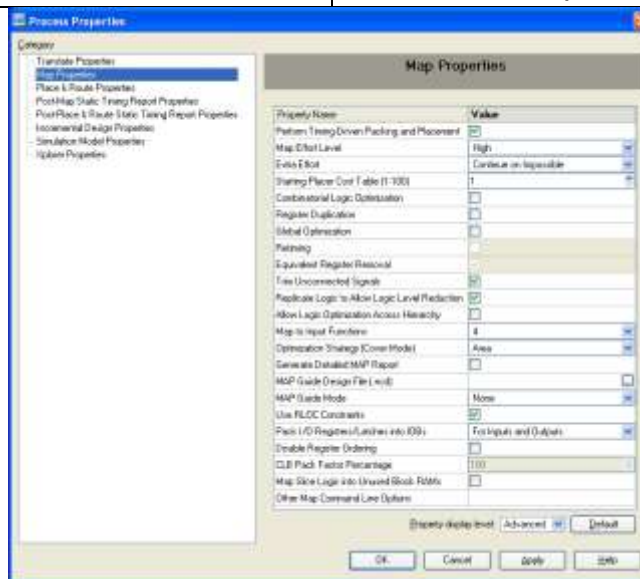


- d. Click 'Map Properties' in the 'Category' pane.
- e. The 'Map Properties' dialog window should appear.



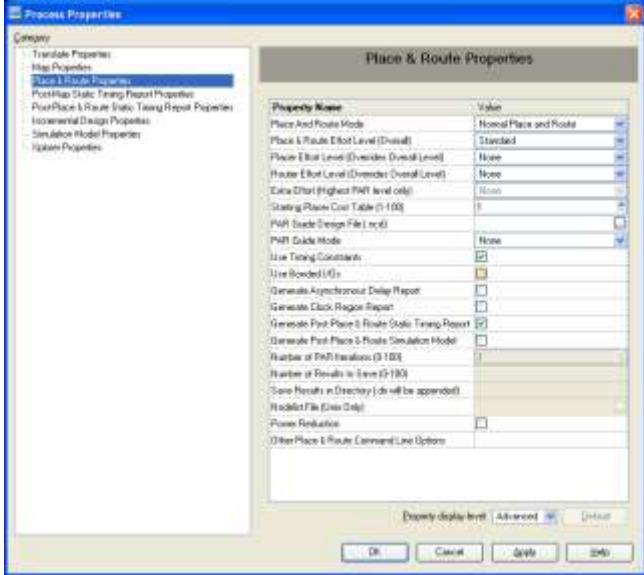
f. Set the values in the table below to the properties in this window.

Property Name	Value
Perform Timing-Driven Packing and Placement	Yes
Map Effort Level	High
Extra Effort	Continue on Impossible



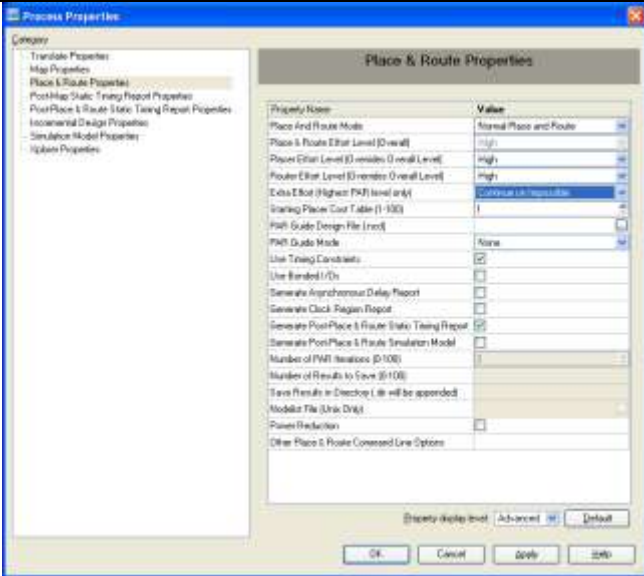
g. Click 'Place & Route Properties' in the 'Category' pane.

h. The 'Place & Route Properties' dialog window should appear.



- i. Set the values in the table below to the properties in this window.

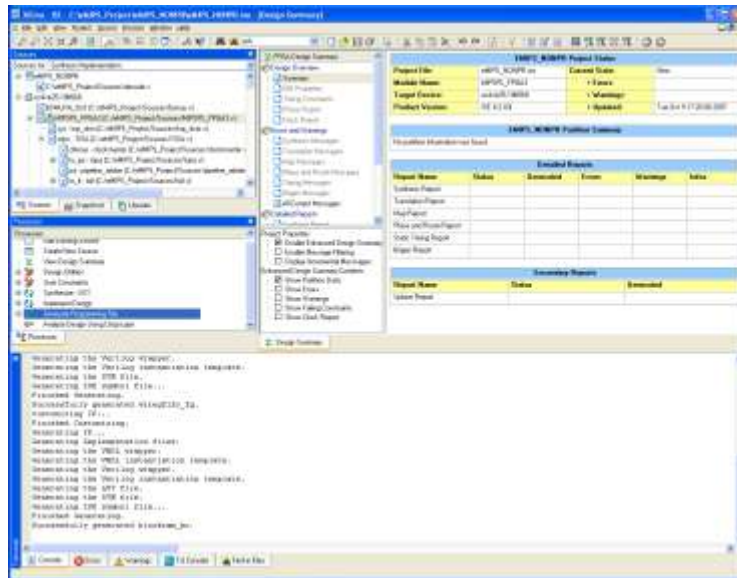
Property Name	Value
Place & Route Effort Level (Overall)	High
Placer Effort Level (Overrides Overall Level)	High
Router Effort Level (Overrides Overall Level)	High
Extra Effort (Highest PAR level only)	Continue on Impossible



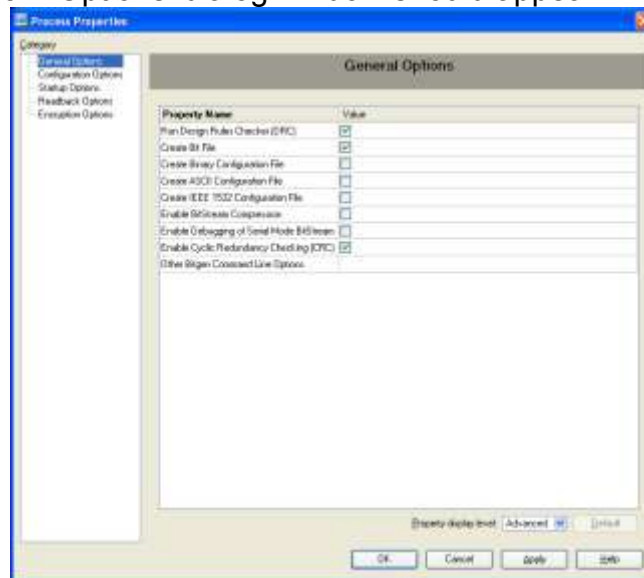
- j. After you have reviewed the entries in the window, click 'OK' to continue.

10. Set Programming Options

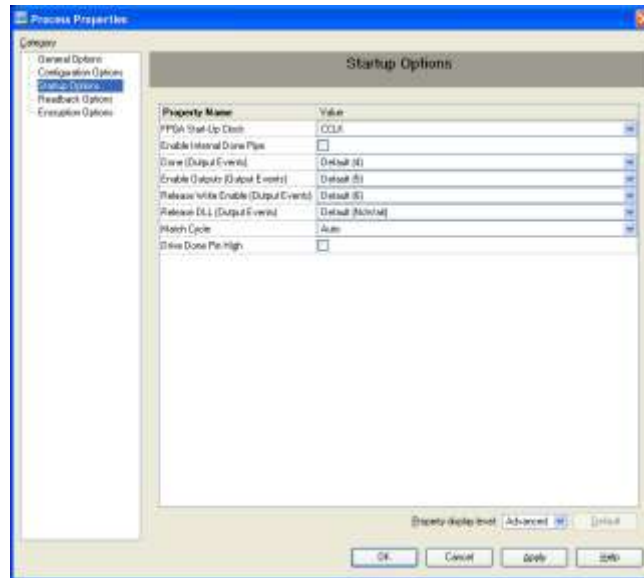
- a. In the 'Sources' pane, select 'MIPSPL_FPGA3'. Then select 'Generate Program File' in the 'Processes' pane. Make sure to select the module and not the project icons.



- b. Right-click on 'Generate Program Files' and select 'Properties'.
- c. The 'Program Options' dialog window should appear.

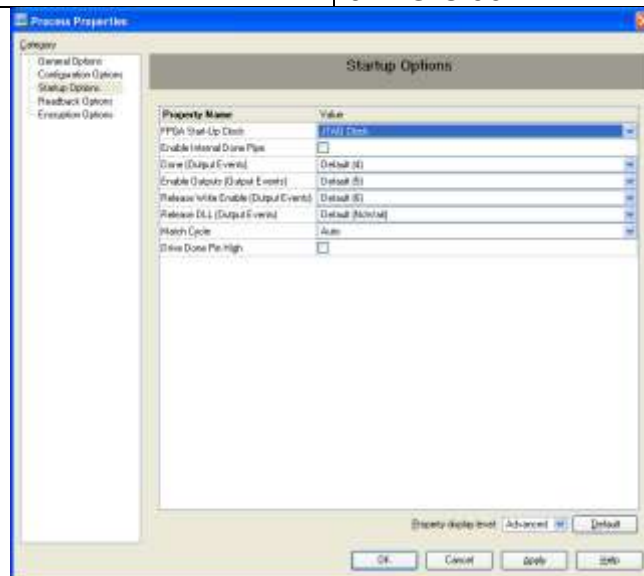


- d. Click 'Startup Options' in the 'Category' pane.
- e. The 'Startup Options' dialog window should appear.



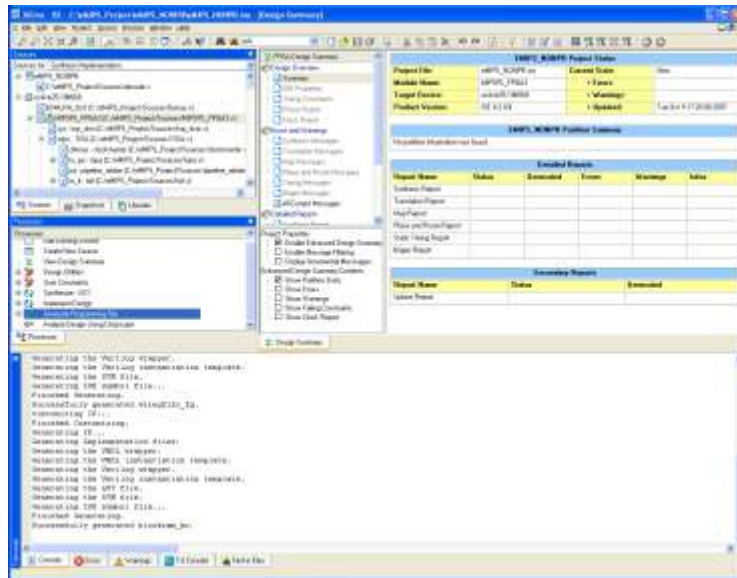
f. Set the values in the table below to the properties in this window.

Property Name	Value
FPGA Start-Up Clock	JTAG Clock

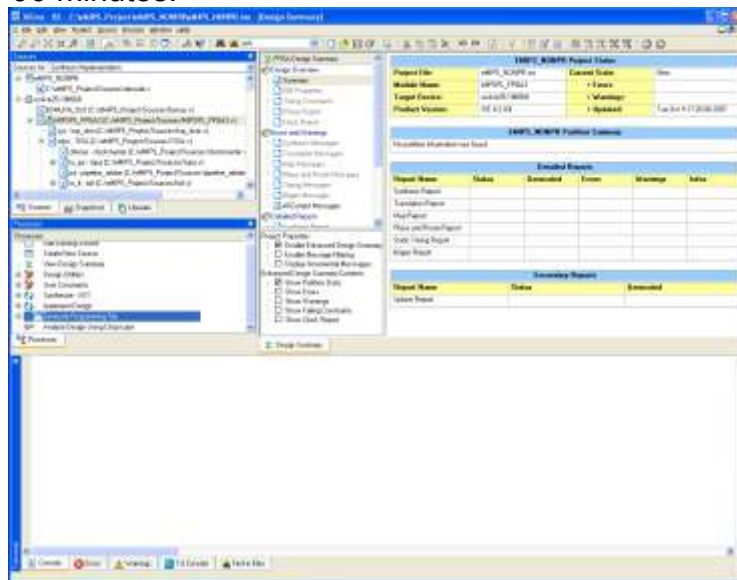


g. After you have reviewed the entries in the window, click 'OK' to continue.

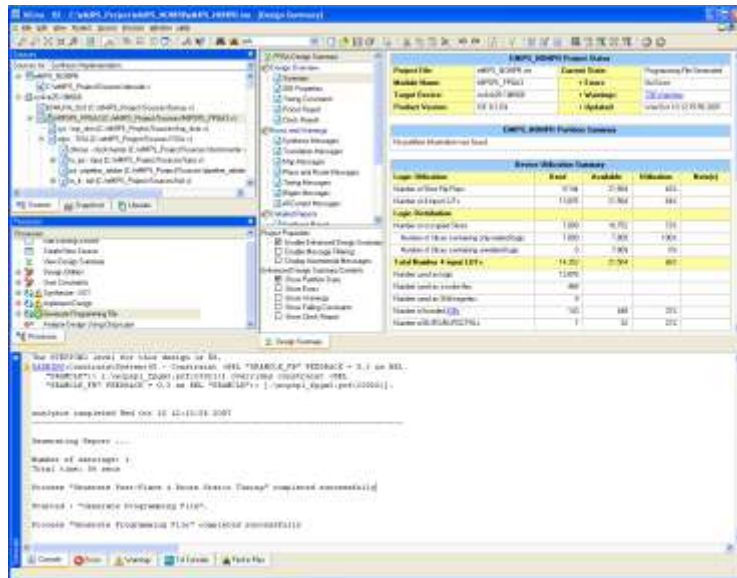
11. In the 'Sources' pane, select 'MIPSPL_FPGA3'. Then select 'Generate Program File' in the 'Processes' pane.



12. You may double-click on 'Generate Program File' or you may right click and select 'Run' to start the build process. This may take some time depending on your system, between 30 to 90 minutes.



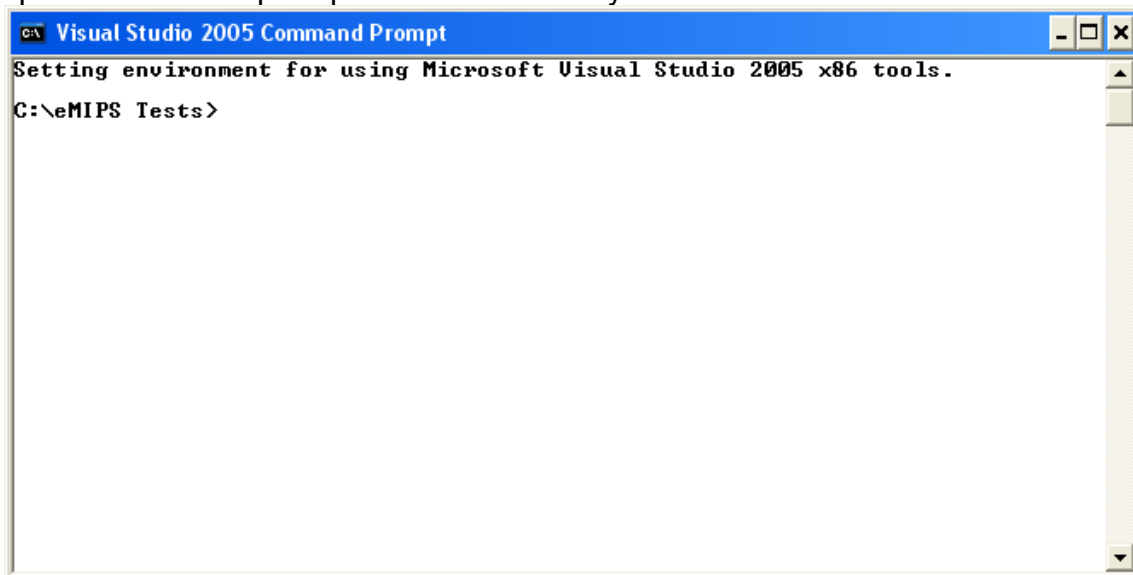
13. You should see a blue spinning icon appear by the 'Generate Program File'. As the process is performed some warnings and info messages will be displayed. Most of these are expected.
14. Wait for the process to complete.



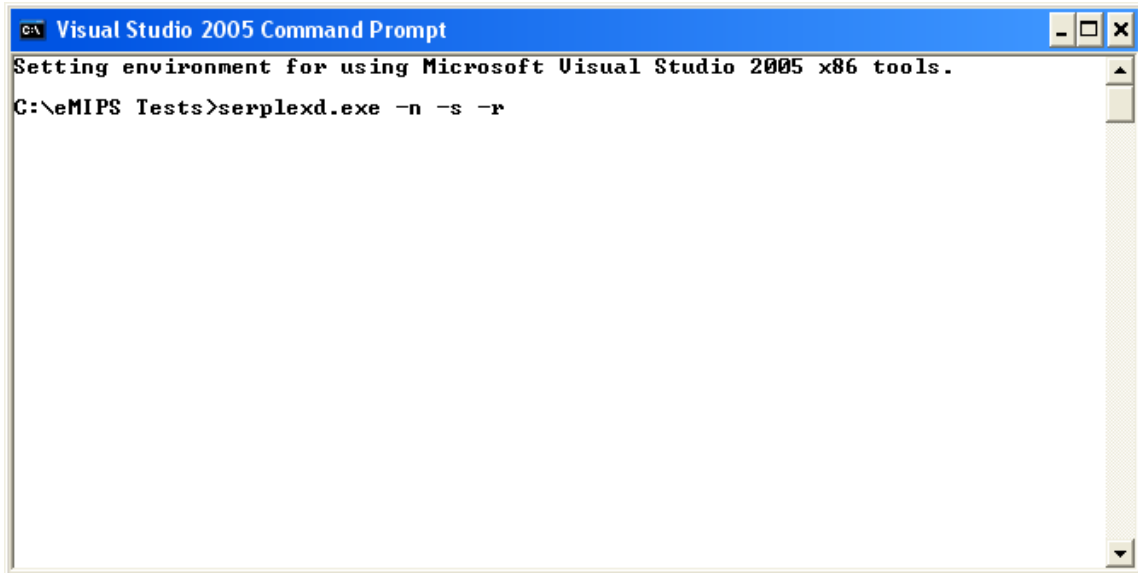
3.3 Verifying the Configuration Bit Files.

The following are the instructions for running a simple test and determine if eMIPS loaded with the mmldiv64 extension is working correctly, and it is performing the configuration functions correctly using the configuration bit files produced.

1. Open a command prompt in the location of your test files.

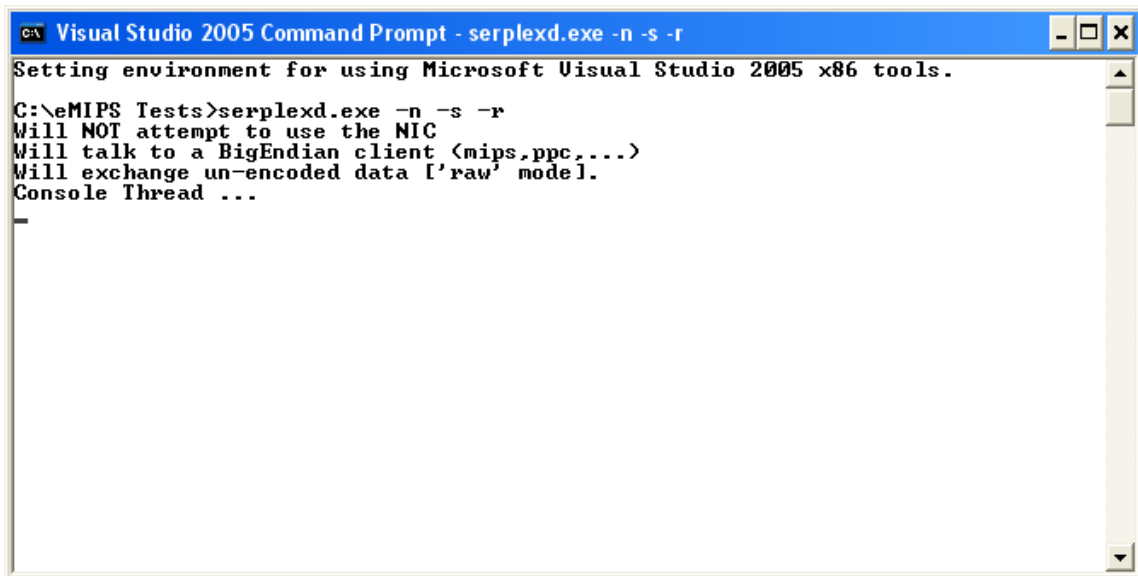


2. Open the serplexd console to the eMIPS board by typing: "serplexd.exe -n -s -r". We are assuming that the serial cable to the ML401 "UART Host" port is working properly and connected to the COM1 port on your PC. If you are using a different port specify it explicitly to serplexd, e.g. "serplexd -n -s -r com3".



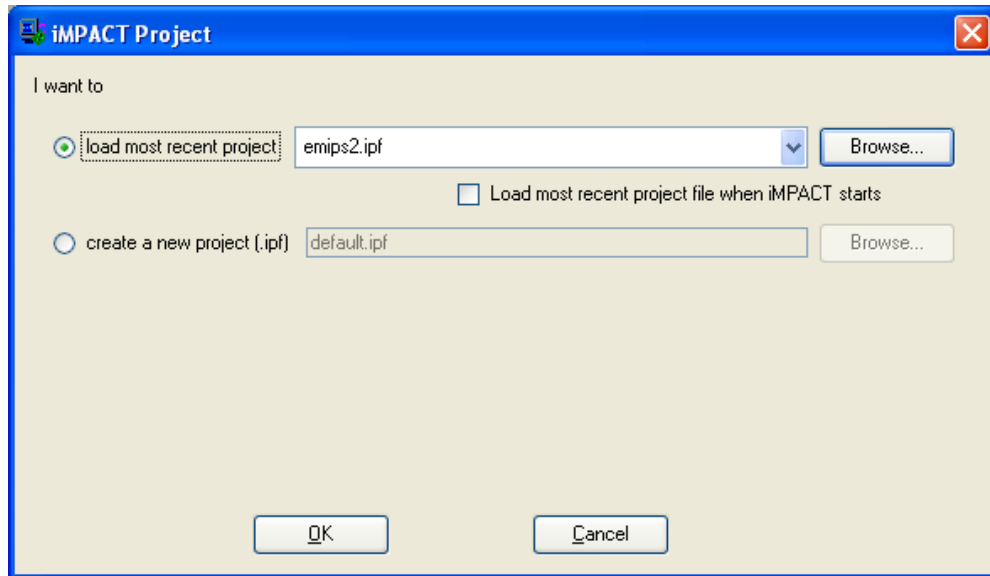
```
Visual Studio 2005 Command Prompt
Setting environment for using Microsoft Visual Studio 2005 x86 tools.
C:\eMIPS Tests>serplexd.exe -n -s -r
```

3. Press Enter.

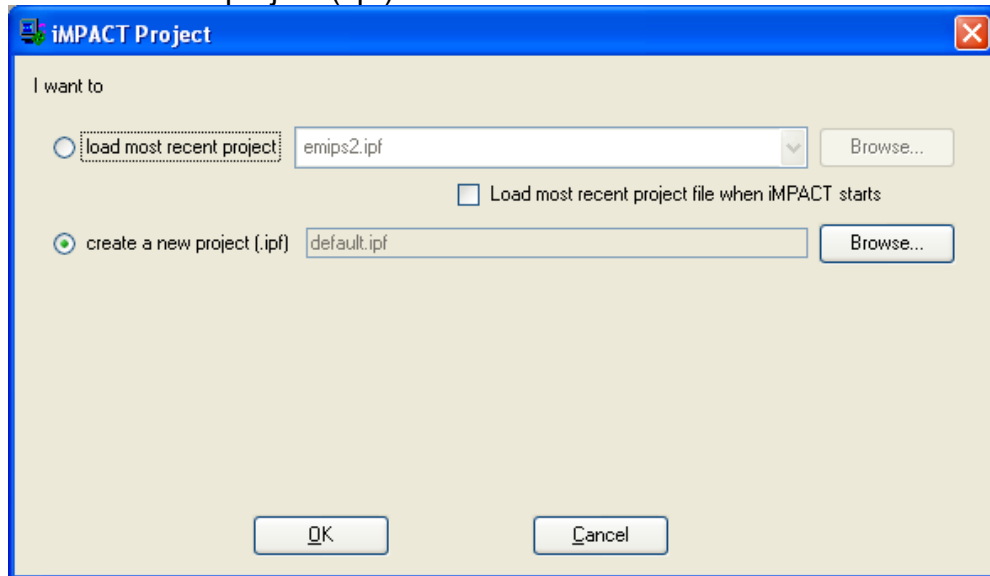


```
Visual Studio 2005 Command Prompt - serplexd.exe -n -s -r
Setting environment for using Microsoft Visual Studio 2005 x86 tools.
C:\eMIPS Tests>serplexd.exe -n -s -r
Will NOT attempt to use the NIC
Will talk to a BigEndian client <mips,ppc,...>
Will exchange un-encoded data ['raw' model].
Console Thread ...
```

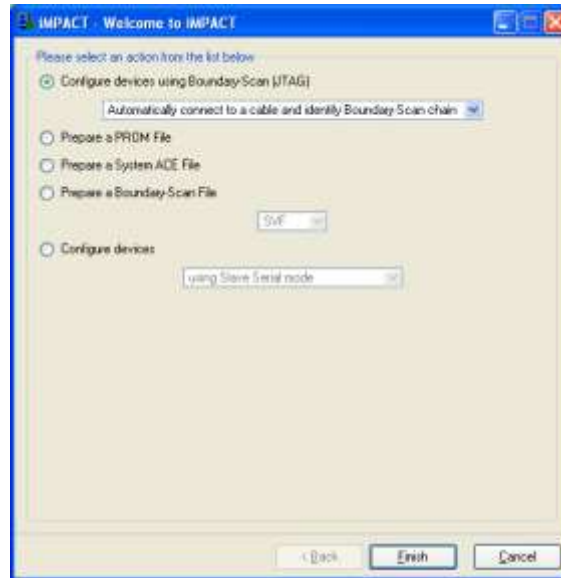
4. Turn on your ML401 board.
5. Flip the switches 1 and 2 on the GPIO DIP switches to the down position
6. Open the Xilinx IMPACT from the version of the Xilinx ISE with the PR Tools. You can verify that you are using the correct version by looking at the version number on the load splash screen.
7. The Xilinx 'IMPACT Project' dialog should appear.



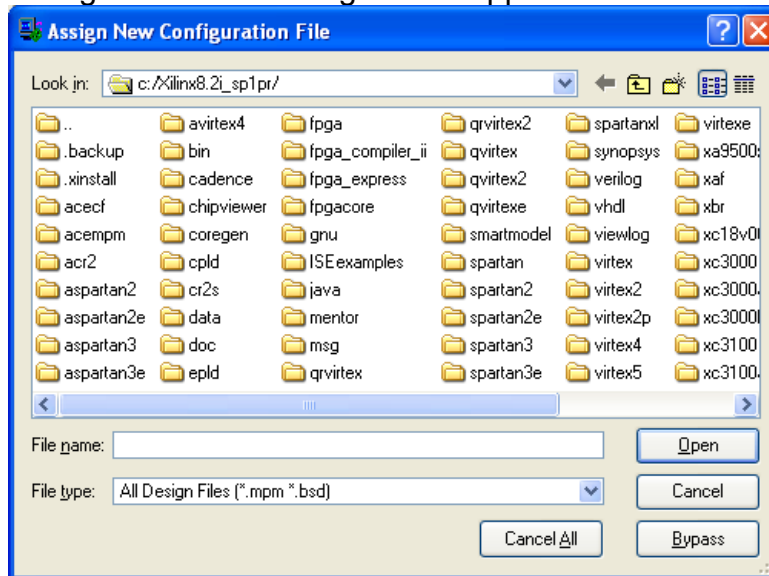
8. Select 'create a new project (.ipf)'



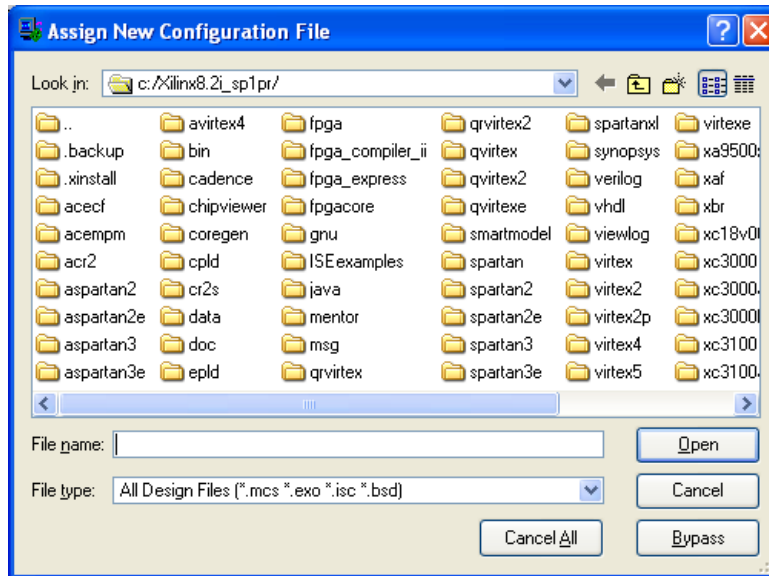
9. Click 'OK' to continue.
10. The 'Welcome to IMPACT' dialog should appear.



11. Click 'Finish' to continue.
12. The 'sysace' object in the JTAG chain window should be highlighted in green, and the 'Assign New Configuration File' dialog should appear.

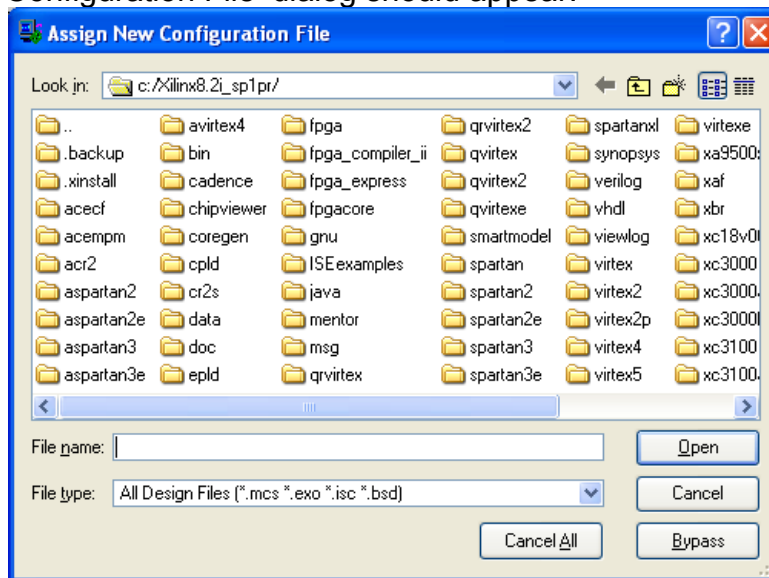


13. Click 'Bypass' to continue.
14. The 'xcf32p' object in the JTAG chain window should be highlighted in green, and the 'Assign New Configuration File' dialog should appear.

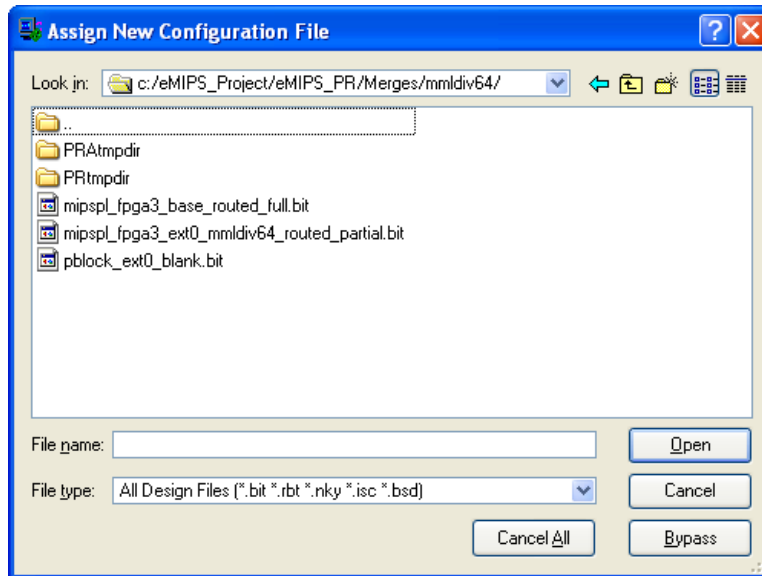


15. Click 'Bypass' to continue.

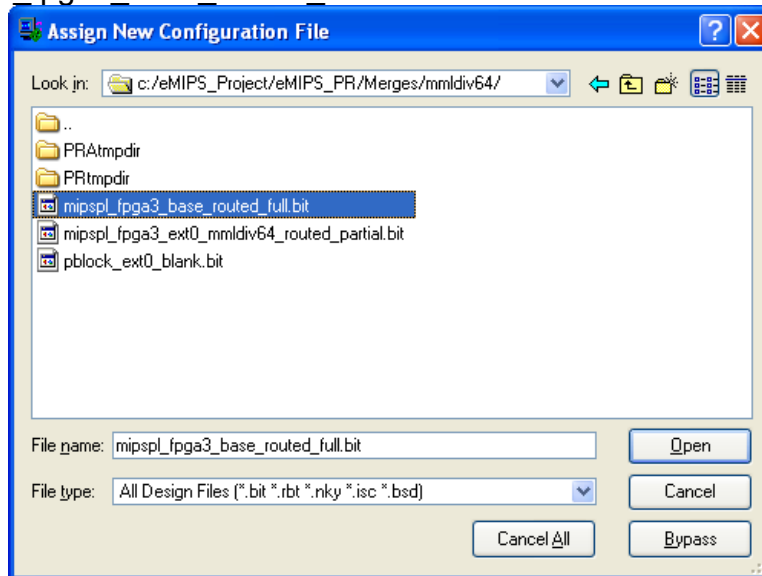
16. The 'xc4vlx25' object in the JTAG chain window should be highlighted in green, and the 'Assign New Configuration File' dialog should appear.



17. Navigate to the location of your bit files.

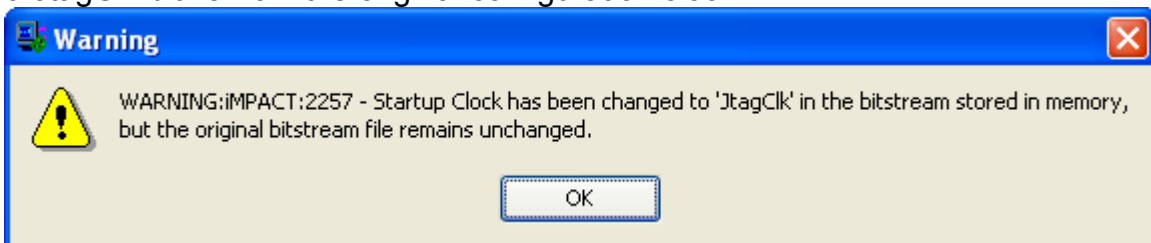


18. Select 'mipspl_fpga3_base_routed_full.bit'.



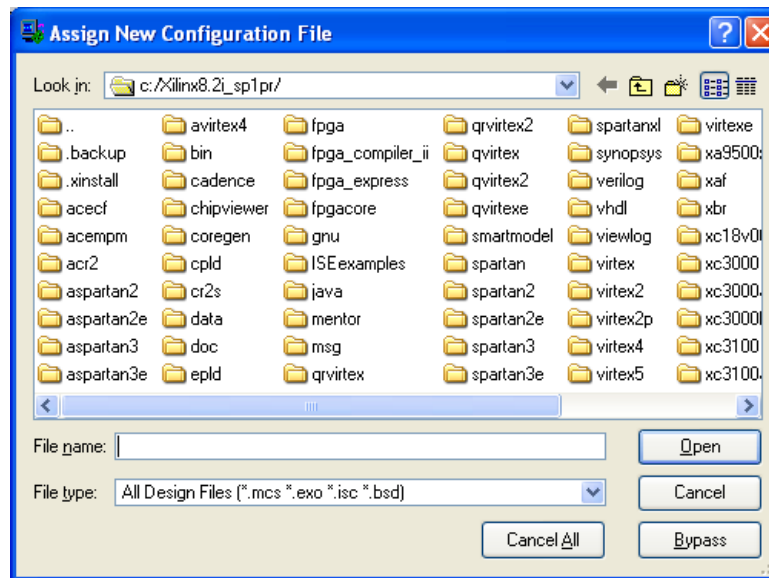
19. Click 'Open' to continue.

20. A 'Warning' dialog will appear notifying you that the IMPACT has converted the bit file to a JtagClk bitfile from the original configuration clock.



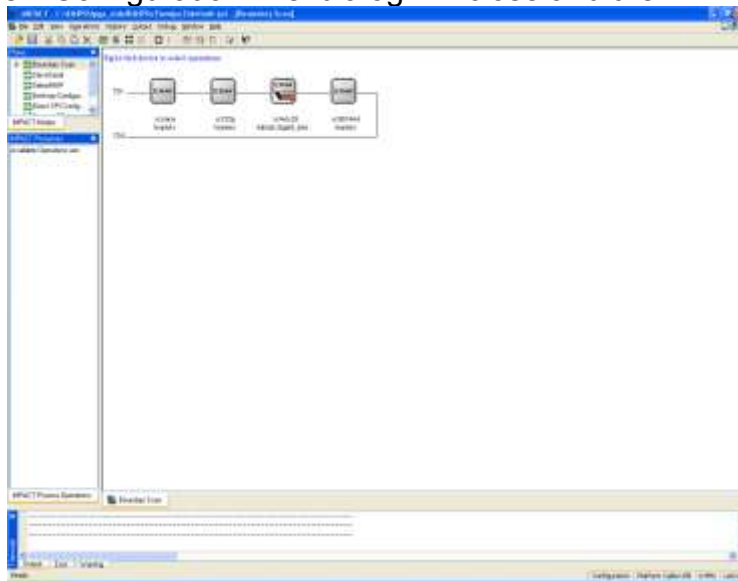
21. Click 'OK' to continue.

22. The 'xc95144xl' object in the JTAG chain window should be highlighted in green, and the 'Assign New Configuration File' dialog should appear.



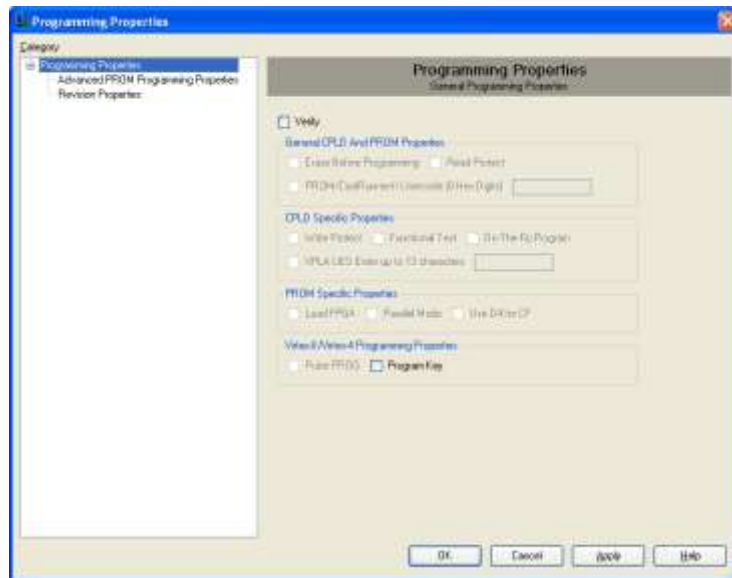
23. Click 'Bypass' to continue.

24. The 'Assign New Configuration File' dialog will close and the IMPACT tool is initialized.



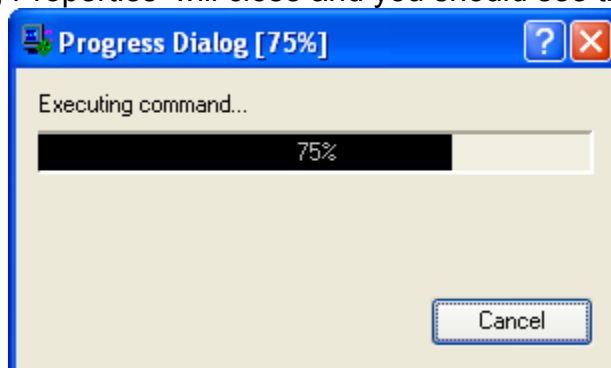
25. Right-click on the 'xc4vlx25' object and select 'Program'

26. The 'Programming Properties' dialog should appear.

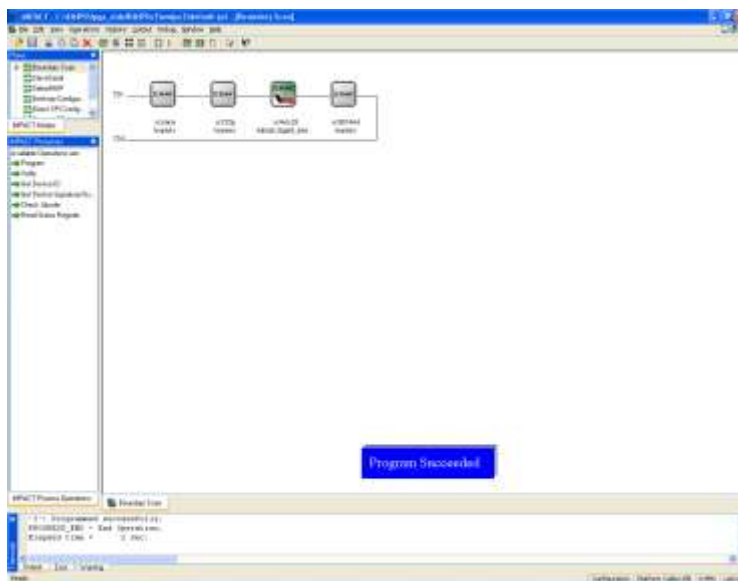


27. Click 'OK' to continue.

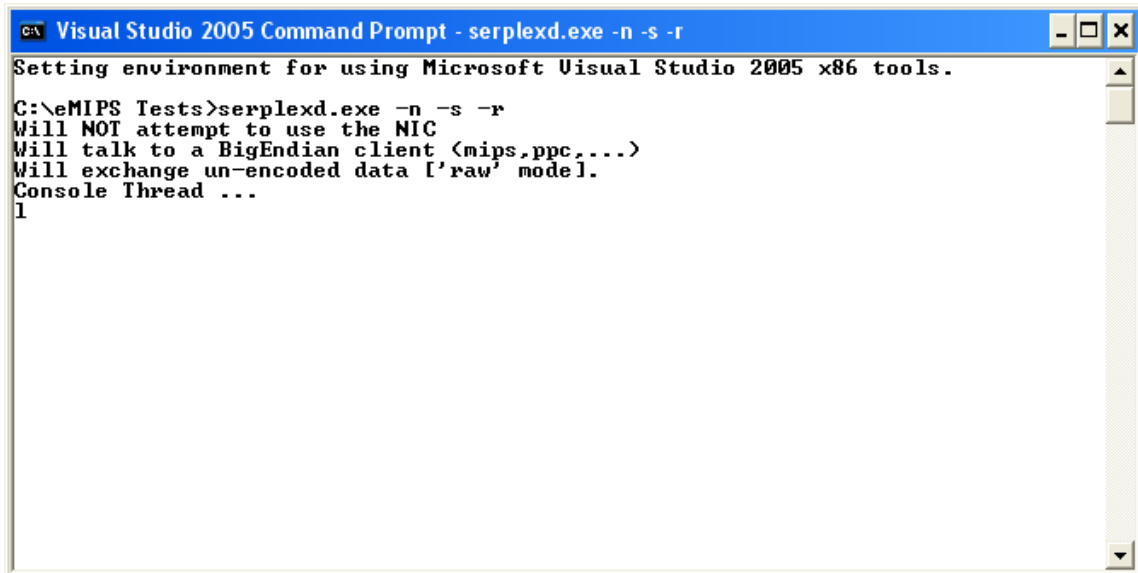
28. The 'Programming Properties' will close and you should see the 'Progress Dialog'.



29. When the configuration is successful you should see the blue 'Program Succeeded' message.



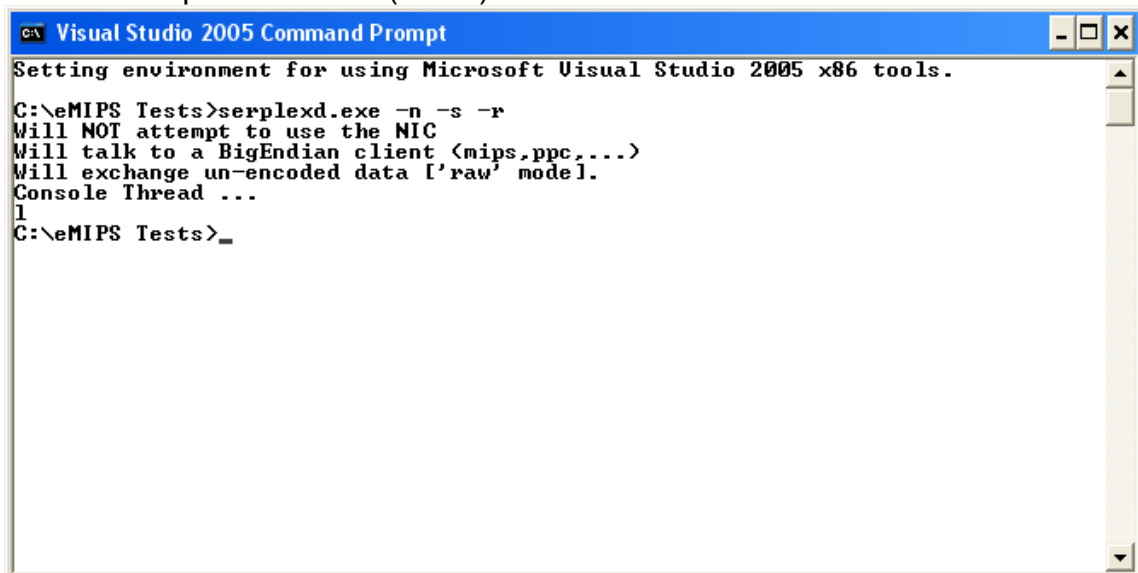
30. Look at the serplexd console and you should see an 'l' on the console. This is the download leader byte, it is printed by the bootloader program to indicate it is ready for a download over the serial line.



```
Visual Studio 2005 Command Prompt - serplexd.exe -n -s -r
Setting environment for using Microsoft Visual Studio 2005 x86 tools.

C:\eMIPS Tests>serplexd.exe -n -s -r
Will NOT attempt to use the NIC
Will talk to a BigEndian client <mips,ppc,...>
Will exchange un-encoded data ['raw' model].
Console Thread ...
l
```

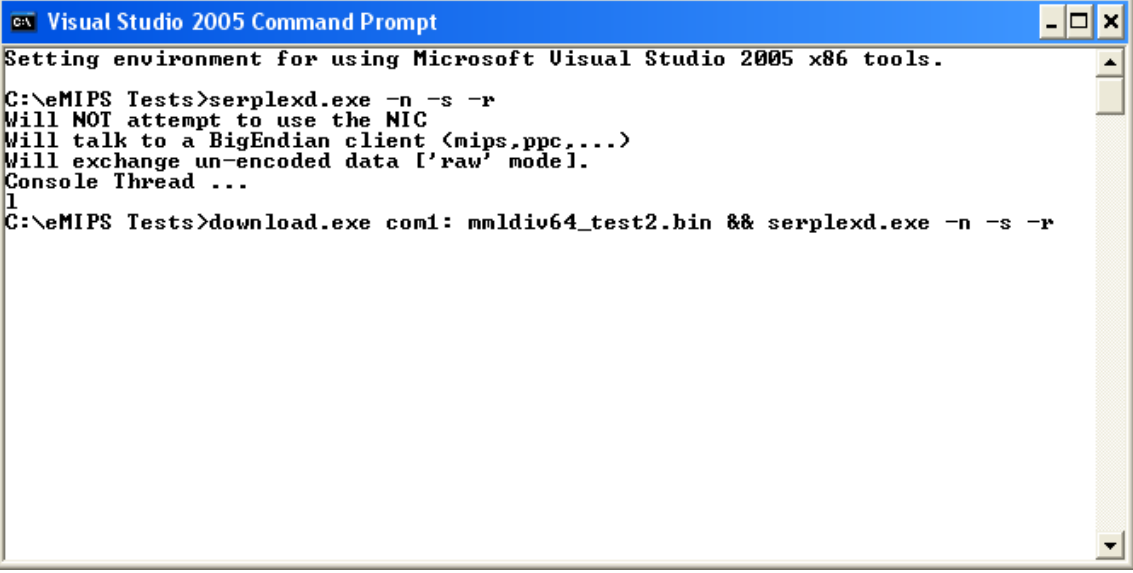
31. Close the serplexd console (hit ^C).



```
Visual Studio 2005 Command Prompt
Setting environment for using Microsoft Visual Studio 2005 x86 tools.

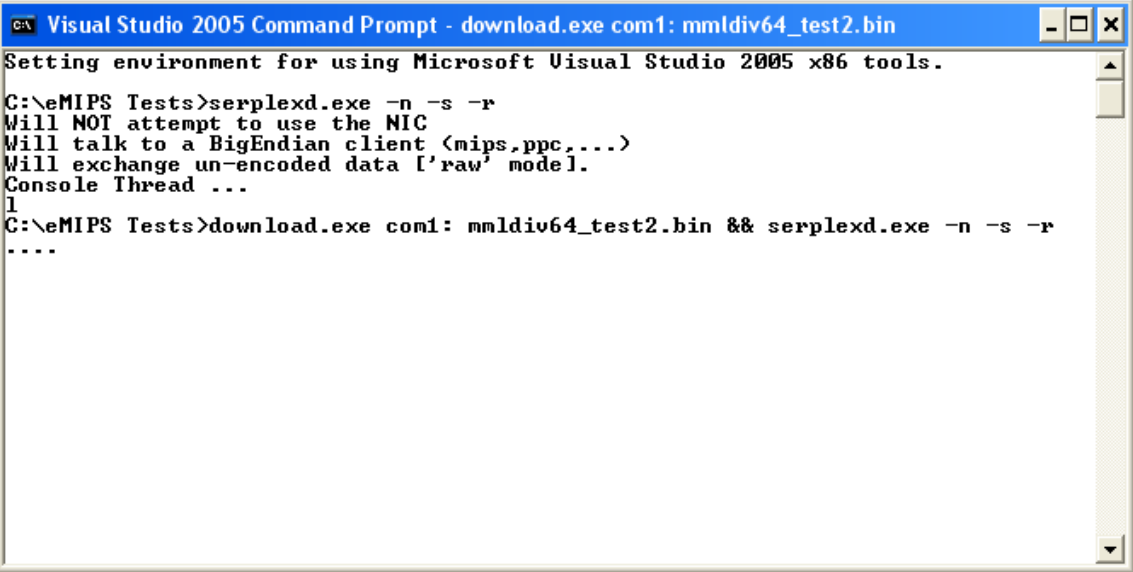
C:\eMIPS Tests>serplexd.exe -n -s -r
Will NOT attempt to use the NIC
Will talk to a BigEndian client <mips,ppc,...>
Will exchange un-encoded data ['raw' model].
Console Thread ...
l
C:\eMIPS Tests>_
```

32. Type the following command: "download.exe com1: mmldiv64_test2.bin && serplexd.exe -n -s -r". Potential errors: One user reported that under his VisualStudio Command Prompt the pipe above did not work, whereas splitting the commands in two command lines did work. Using the regular Windows CMD.EXE also fixed the problem.



```
Visual Studio 2005 Command Prompt
Setting environment for using Microsoft Visual Studio 2005 x86 tools.
C:\eMIPS Tests>serplexd.exe -n -s -r
Will NOT attempt to use the NIC
Will talk to a BigEndian client (mips,ppc,...)
Will exchange un-encoded data ['raw' model.
Console Thread ...
1
C:\eMIPS Tests>download.exe com1: mmldiv64_test2.bin && serplexd.exe -n -s -r
```

33. Press Enter.



```
Visual Studio 2005 Command Prompt - download.exe com1: mmldiv64_test2.bin
Setting environment for using Microsoft Visual Studio 2005 x86 tools.
C:\eMIPS Tests>serplexd.exe -n -s -r
Will NOT attempt to use the NIC
Will talk to a BigEndian client (mips,ppc,...)
Will exchange un-encoded data ['raw' model.
Console Thread ...
1
C:\eMIPS Tests>download.exe com1: mmldiv64_test2.bin && serplexd.exe -n -s -r
....
```

34. The Program will begin to download over the serial line.

35. After the download is complete the serplexd console will launch allowing you to see the output of the program.

```

Visual Studio 2005 Command Prompt - serplexd.exe -n -s -r
Setting environment for using Microsoft Visual Studio 2005 x86 tools.

C:\eMIPS Tests>serplexd.exe -n -s -r
Will NOT attempt to use the NIC
Will talk to a BigEndian client (mips,ppc,...)
Will exchange un-encoded data ['raw' model].
Console Thread ...
1
C:\eMIPS Tests>download.exe com1: mmldiv64_test2.bin && serplexd.exe -n -s -r
.....Download
d complete, 18440 bytes sent
Will NOT attempt to use the NIC
Will talk to a BigEndian client (mips,ppc,...)
Will exchange un-encoded data ['raw' model].
Console Thread ...
HiMom? sp=80001850
GPIO = 00000000

MARK - RUNNING NON EXTENSION TEST
READY...
SET...
GO!!!

DONE...

MARK - RUNNING BASELINE
READY...
SET...
GO!!!

DONE...

MARK - RUNNING EXTENSION TEST
READY...
SET...
GO!!!

DONE...

RESULTS

Errors =          00000000

NON EXTENSION TEST RESULTS
  NOEXT START =    00000000.0002674b
  NOEXT FINISH =   00000000.04b5f3cc
  NOEXT TIME  =    04b38c81

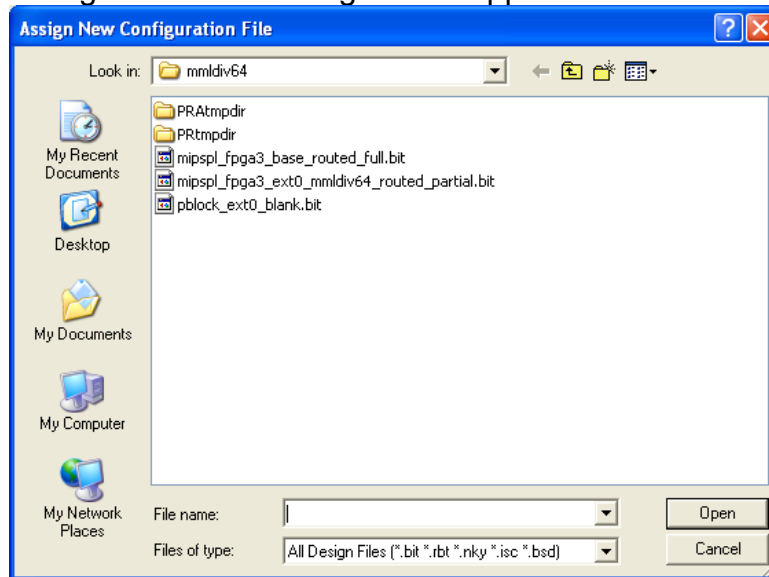
BASELINE RESULTS
  BASE START =     00000000.04b85d32
  BASE FINISH =    00000000.09aff833
  BASE TIME  =     04f79b01

EXTENSION TEST RESULTS
  EXT START =      00000000.09b2a1e9
  EXT FINISH =     00000000.0be01bed
  EXT TIME  =      022d7a04

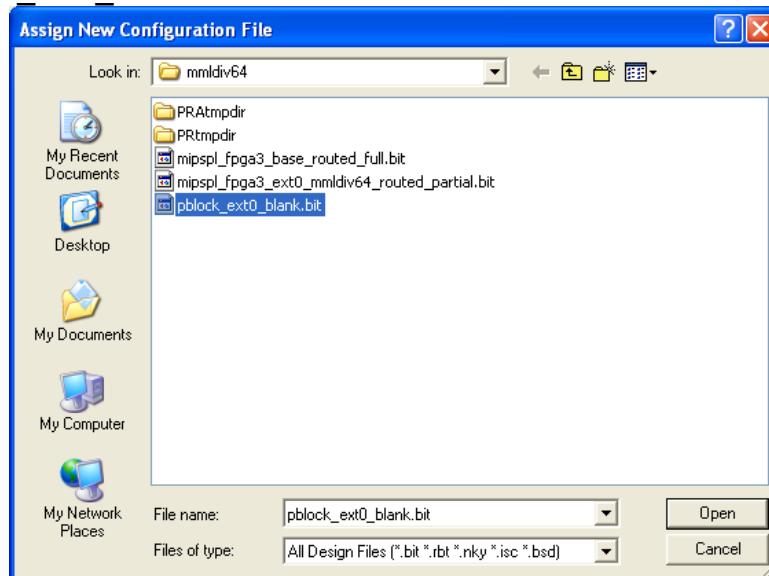
PERFORMANCE over base (base/ext) =          00000002
PERFORMANCE over original (noext/ext) =       00000002
OVERHEAD over original (base/noext) =         00000001
TEST PASSED SUCCESSFULLY 1

```

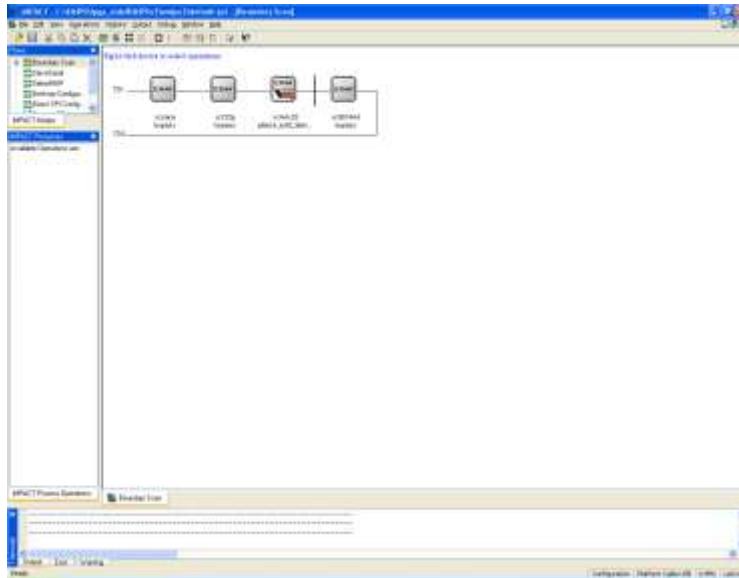
36. This represents that the Extension architecture loaded with the mmldiv64 extension has completed the hardware test successfully. The accelerated code shows a factor of two speed-up.
37. Right-click on the 'xc4vlx25' object and select 'Assign New Configuration File'.
38. The 'xc4vlx25' object in the JTAG chain window should be highlighted in green, and the 'Assign New Configuration File' dialog should appear.



39. Select 'pblock_ext0_blank.bit'.

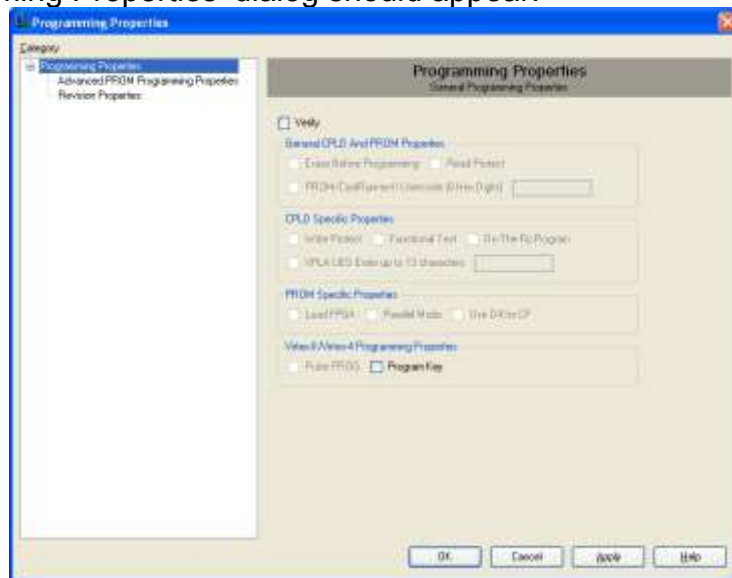


40. Click 'Open' to continue.
41. The 'Assign New Configuration File' dialog will close.



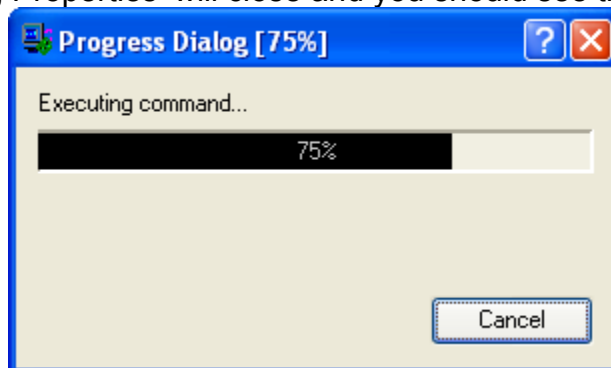
42. Right-click on the 'xc4vlx25' object and select 'Program'

43. The 'Programming Properties' dialog should appear.

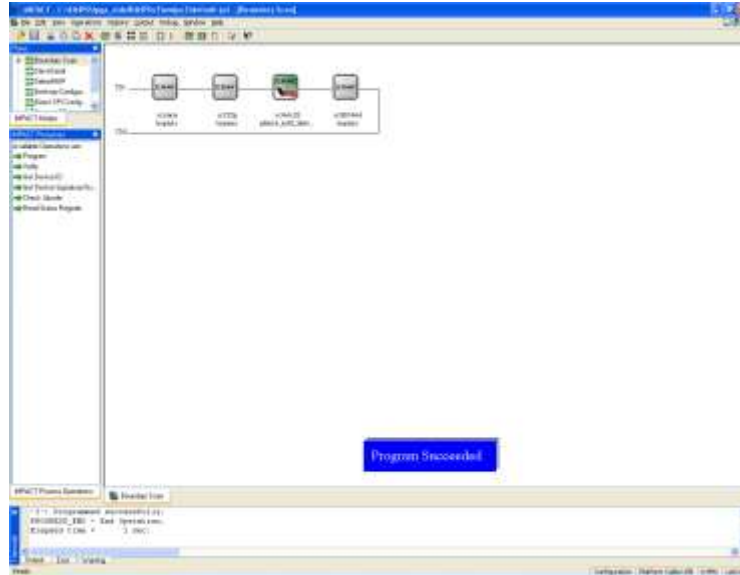


44. Click 'OK' to continue.

45. The 'Programming Properties' will close and you should see the 'Progress Dialog'.



46. When the configuration is successful you should see the blue 'Program Succeeded' message.



47. Look at the serplexd console and you should see a few more 'I's on the console. These may be ignored.
48. Close the serplexd console and run the previous command again (see step 32).

```

Visual Studio 2005 Command Prompt - download.exe com1: mmldiv64_test2.bin
NON EXTENSION TEST RESULTS
  NOEXT START = 00000000.0002674b
  NOEXT FINISH = 00000000.04b5f3cc
  NOEXT TIME = 04b38c81

BASELINE RESULTS
  BASE START = 00000000.04b85d32
  BASE FINISH = 00000000.09aff833
  BASE TIME = 04f79b01

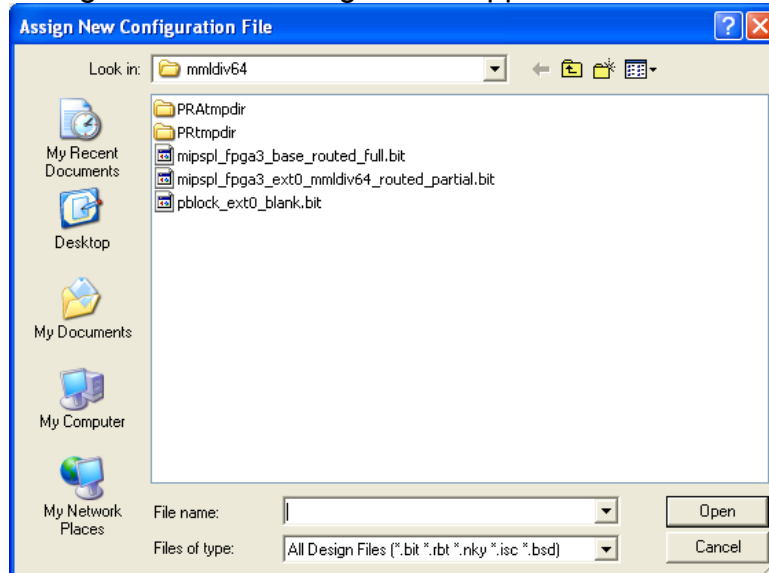
EXTENSION TEST RESULTS
  EXT START = 00000000.09b2a1e9
  EXT FINISH = 00000000.0be01bed
  EXT TIME = 022d7a04

PERFORMANCE over base (base/ext) = 00000002
PERFORMANCE over original (noext/ext) = 00000002
OVERHEAD over original (base/noext) = 00000001
TEST PASSED SUCCESSFULLY !!!
C:\eMIPS Tests>download.exe com1: mmldiv64_test2.bin && serplexd.exe -n -s -r
.....
  
```

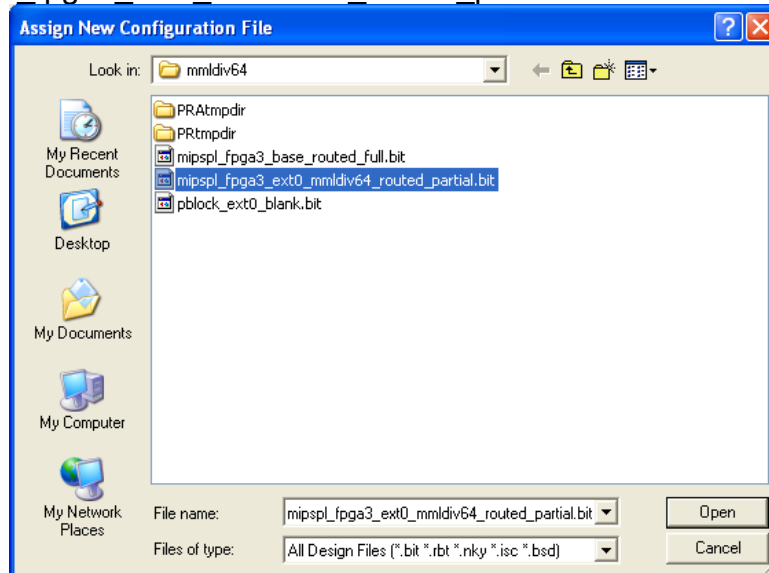
49. After the download is complete the serplexd console will launch allowing you to see the output of the program. You should see a crash.

50. Right-click on the 'xc4vlx25' object and select 'Assign New Configuration File'

51. The 'xc4vlx25' object in the JTAG chain window should be highlighted in green, and the 'Assign New Configuration File' dialog should appear.

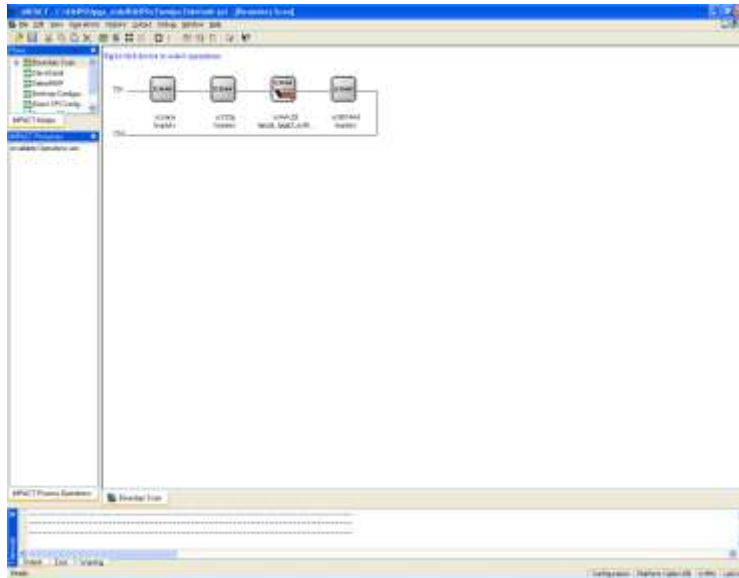


52. Select 'mipspl_fpga3_ext0_mmldiv64_routed_partial.bit'.



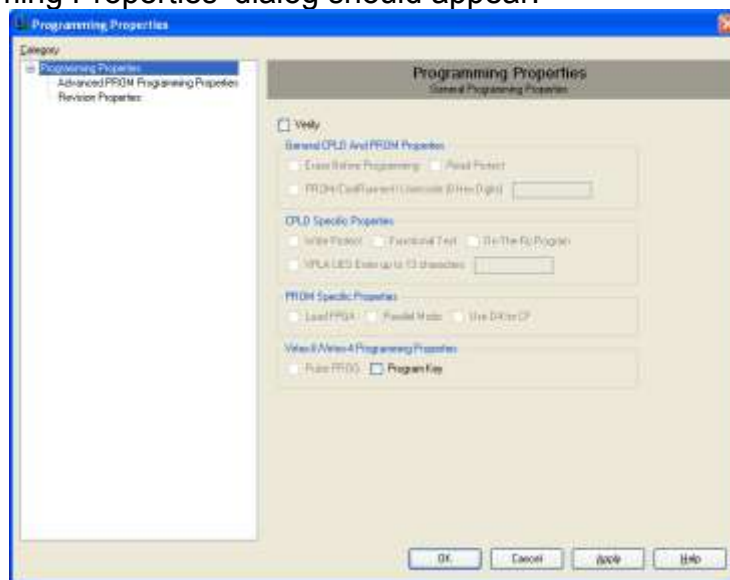
53. Click 'Open' to continue.

54. The 'Assign New Configuration File' dialog will close.



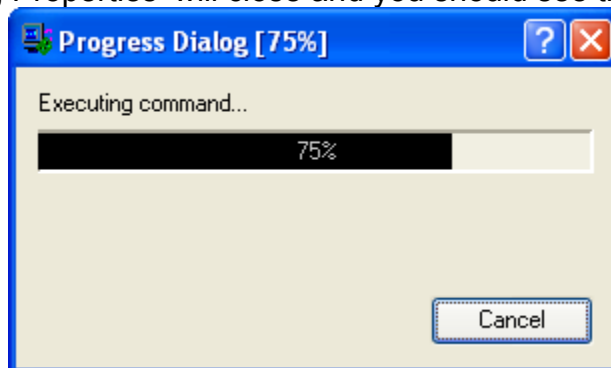
55. Right-click on the 'xc4vlx25' object and select 'Program'

56. The 'Programming Properties' dialog should appear.

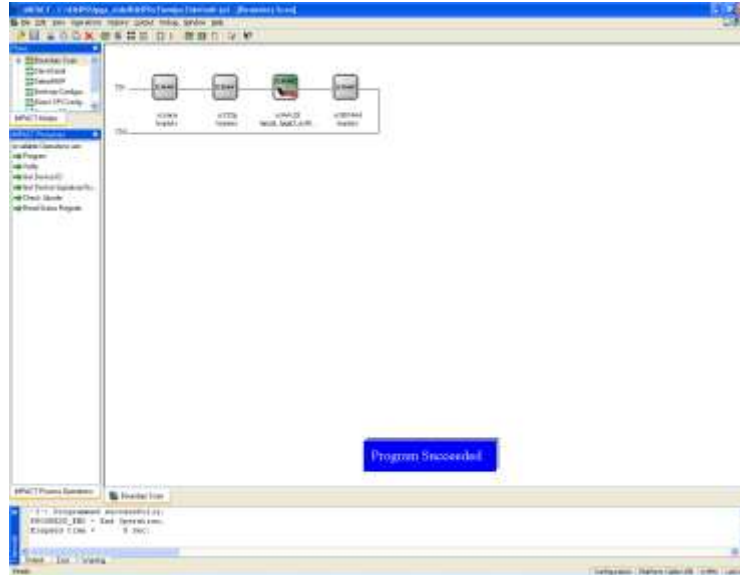


57. Click 'OK' to continue.

58. The 'Programming Properties' will close and you should see the 'Progress Dialog'.



59. When the configuration is successful you should see the blue 'Program Succeeded' message.



60. Look at the serplexd console and you should see a few more 'I's on the console. These may be ignored.

61. Close the serplexd console and run the previous command again (see step 32).



```
Visual Studio 2005 Command Prompt - download.exe com1: mmldiv64_test2.bin
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Had a trap, our Extension is not loaded/enabled.
TEST FAILED at Step=0 lll
C:\eMIPS Tests>download.exe com1: mmldiv64_test2.bin && serplx.exe -n -s -r
.....
```

62. After the download is complete the serplexd console will launch allowing you to see the output of the program. The program should not crash and the correct results should be displayed again.

```

Visual Studio 2005 Command Prompt - serplexd.exe -n -s -r
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Caught trap at pc=8000109c
Had a trap, our Extension is not loaded/enabled.
TEST FAILED at Step=0 lll
C:\eMIPS Tests>download.exe com1: mmldiv64_test2.bin && serplexd.exe -n -s -r
.....Download complete, 18440 bytes sent
Will NOT attempt to use the NIC
Will talk to a BigEndian client (mips,ppc,...)
Will exchange un-encoded data ['raw' model].
Console Thread ...
HiMom! sp=80001850
GPIO = 00000000

MARK - RUNNING NON EXTENSION TEST
READY...
SET...
GO!!!

DONE...

MARK - RUNNING BASELINE
READY...
SET...
GO!!!

DONE...

MARK - RUNNING EXTENSION TEST
READY...
SET...
GO!!!

DONE...

RESULTS

Errors =          00000000

NON EXTENSION TEST RESULTS
  NOEXT START =    00000000.0002674b
  NOEXT FINISH =   00000000.04b5f3cc
  NOEXT TIME =     04b38c81

BASELINE RESULTS
  BASE START =     00000000.04b85cfd
  BASE FINISH =    00000000.09aff7fe
  BASE TIME =      04f79b01

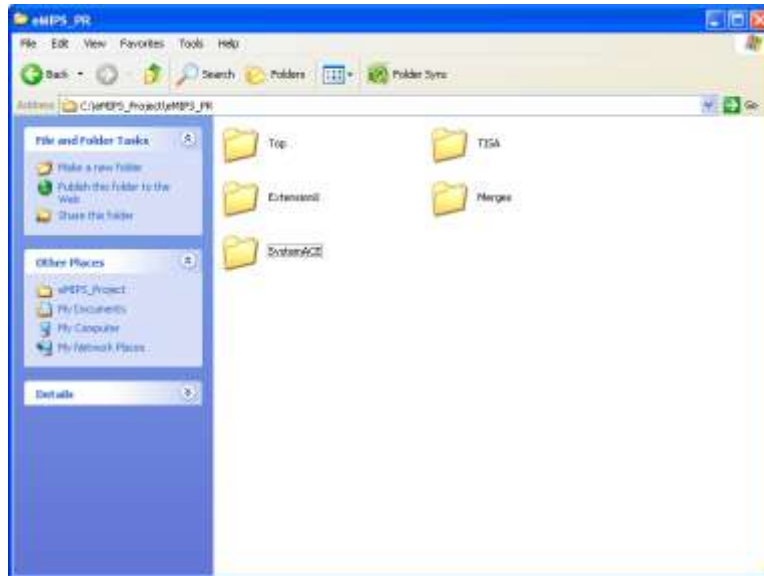
EXTENSION TEST RESULTS
  EXT START =       00000000.09b2a1e9
  EXT FINISH =      00000000.0be01bec
  EXT TIME =        022d7a03

PERFORMANCE over base (base/ext) =          00000002
PERFORMANCE over original (noext/ext) =      00000002
OVERHEAD over original (base/noext) =        00000001
TEST PASSED SUCCESSFULLY 1_

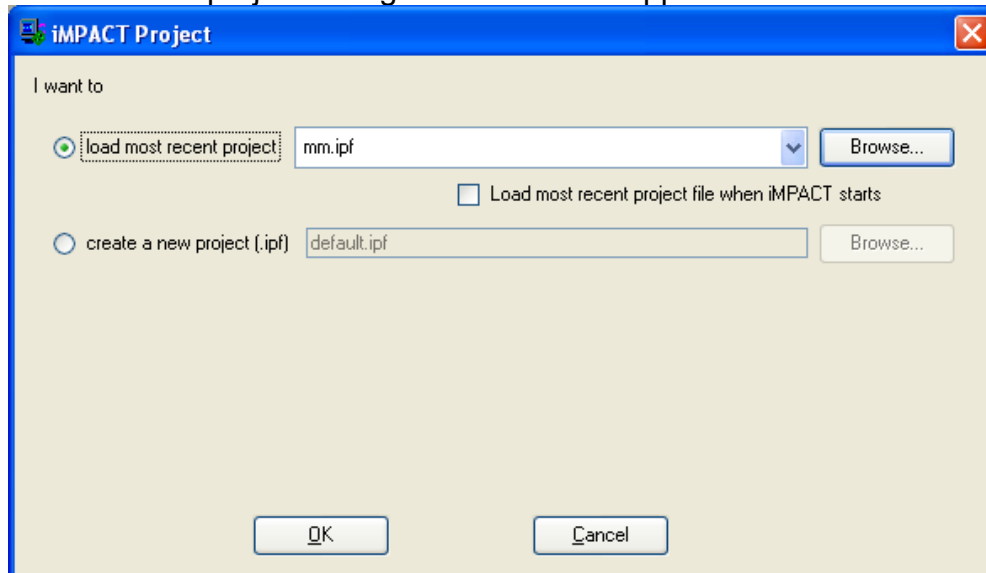
```

3.4 Generating the System ACE Compact Flash Image

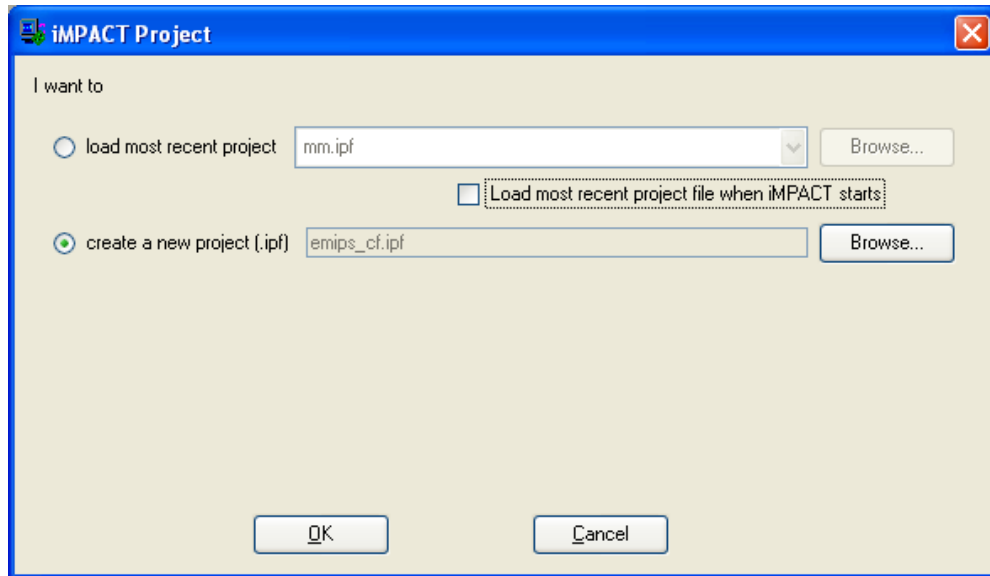
1. In your PR build directory create a new folder called 'SystemACE' for your compact flash image.



2. Start the Xilinx IMPACT from the PR tools install of the Xilinx ISE
3. The Xilinx IMPACT project dialog window should appear.

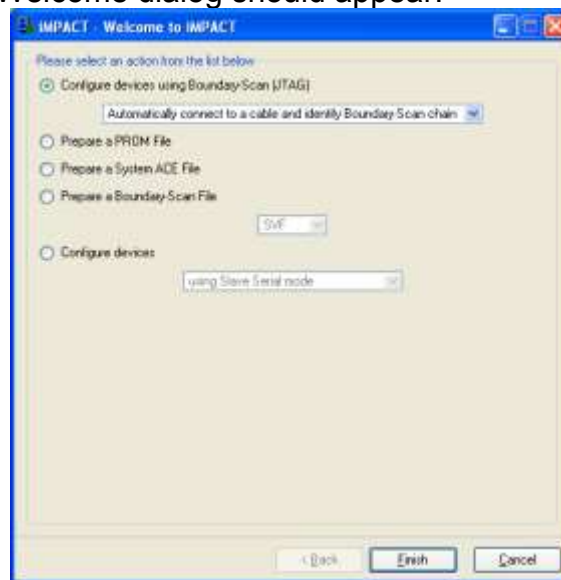


4. Select the 'create a new project (.ipf)' option and save it to the 'SystemACE' folder in your PR build directory. Name your project 'emips_cf'.

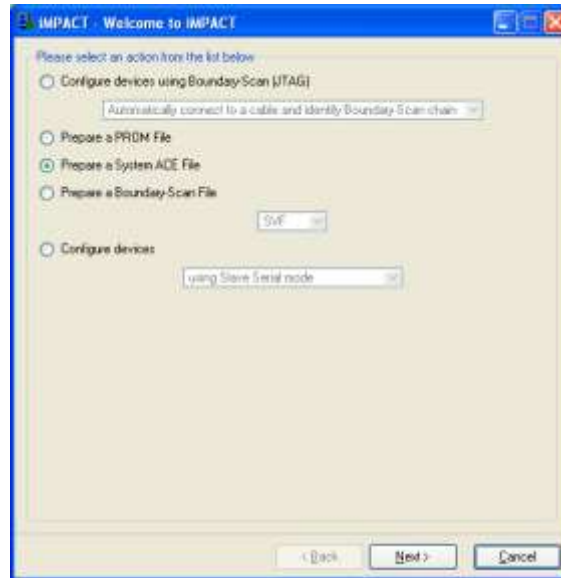


5. After you have reviewed the window click 'OK' to continue.

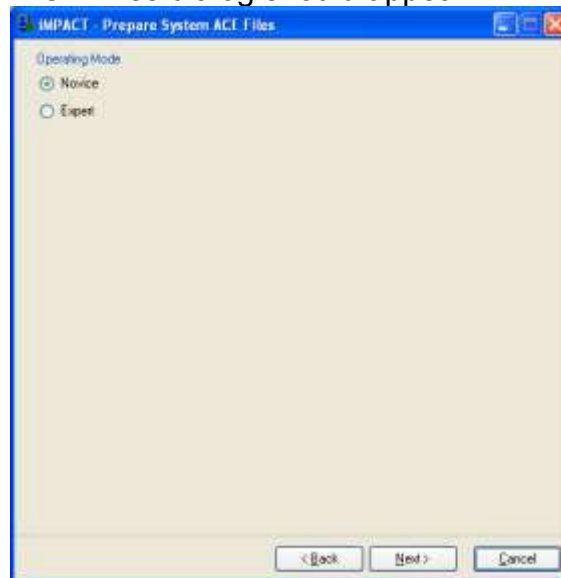
6. The Xilinx iMPACT Welcome dialog should appear.



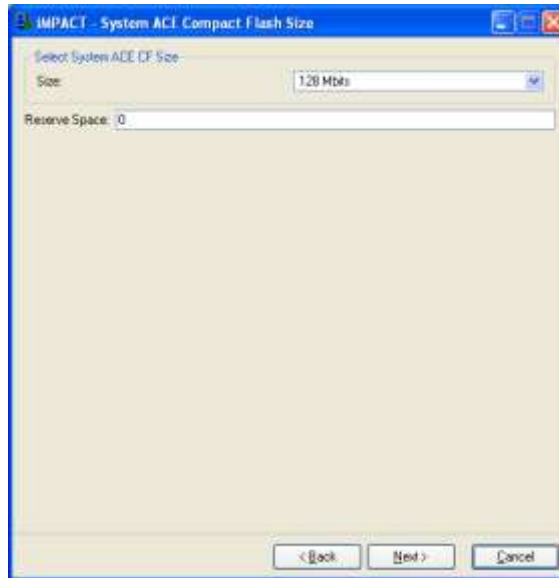
7. Select 'Prepare a System ACE file' from the options available.



8. After reviewing your selection click 'Next' to continue.
9. The Prepare System ACE Files dialog should appear.

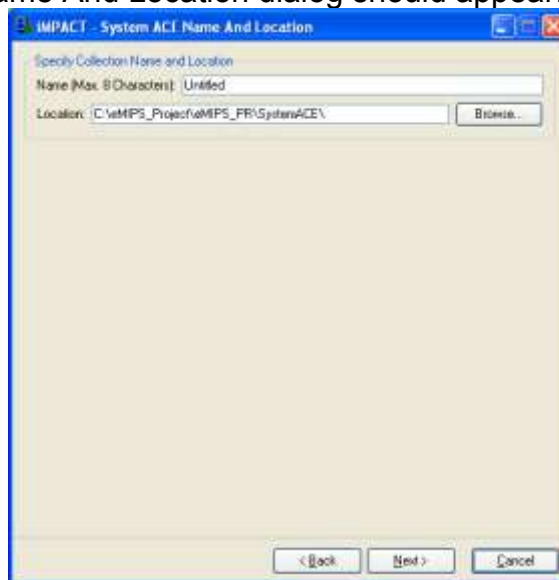


10. Click 'Next' to continue.
11. The System ACE Compact Flash Size dialog should appear.

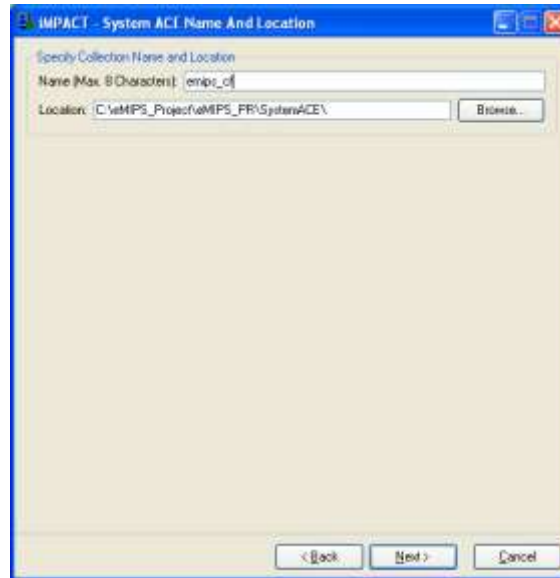


12. Click 'Next' to continue.

13. The System ACE Name And Location dialog should appear.

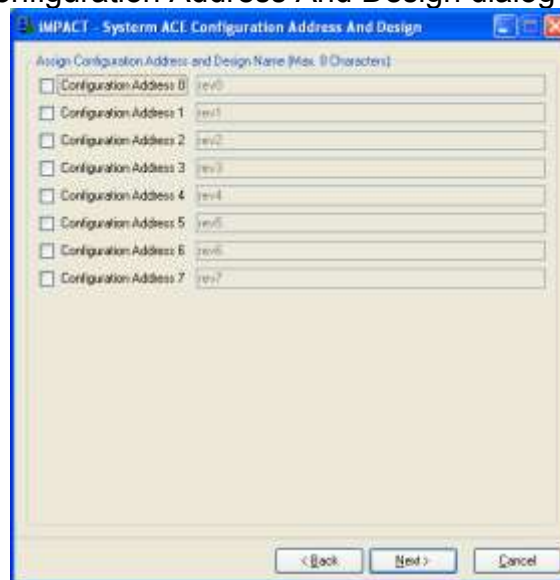


14. Name the image 'emips_cf'.

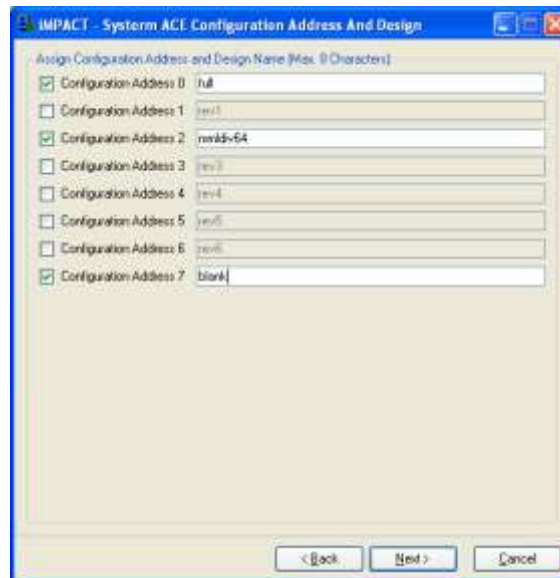


15. After you have reviewed your entry, click 'Next' to continue.

16. The System ACE Configuration Address And Design dialog should appear.

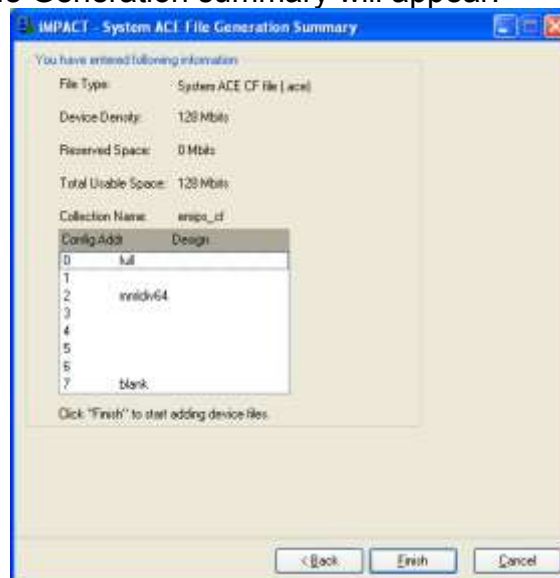


17. Check each configuration address you wish to use and label them.



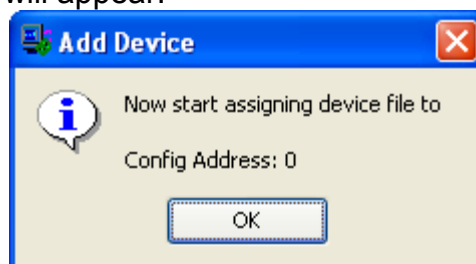
18. After you have reviewed your entries, click 'Next' to continue.

19. The System ACE File Generation summary will appear.



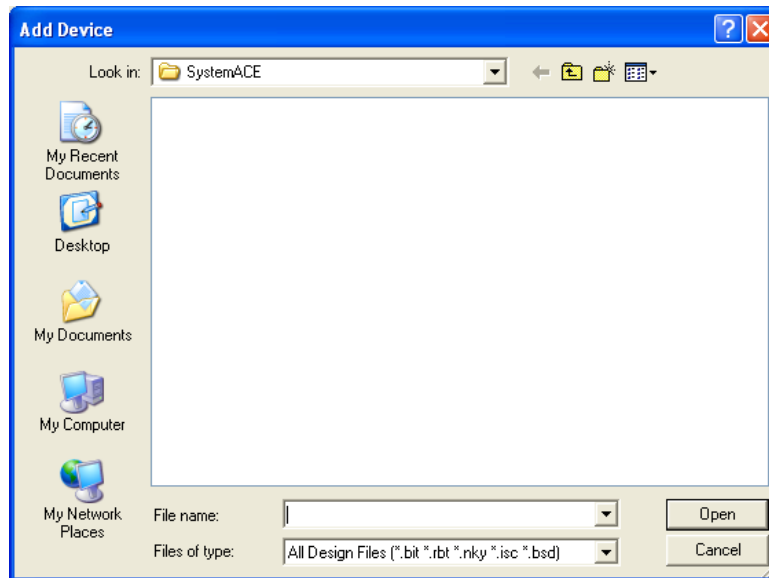
20. Click 'Finish' to continue.

21. The Add Device dialog will appear.

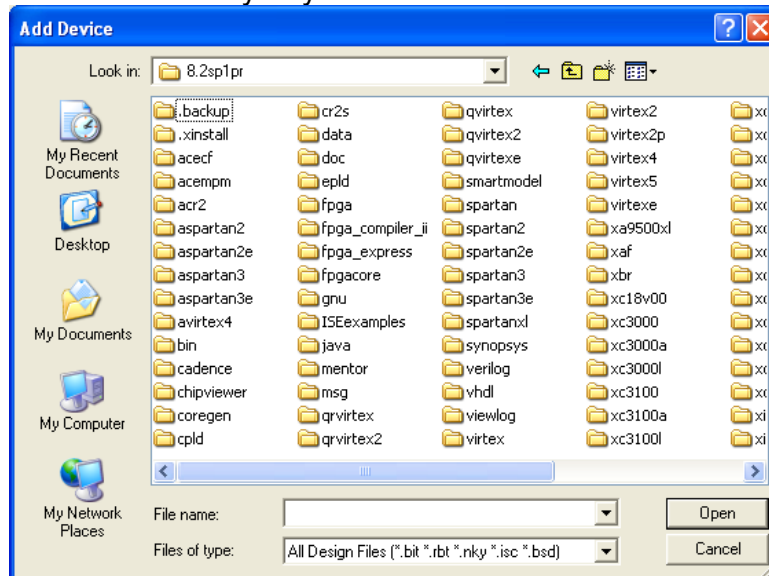


22. Click 'OK' to continue.

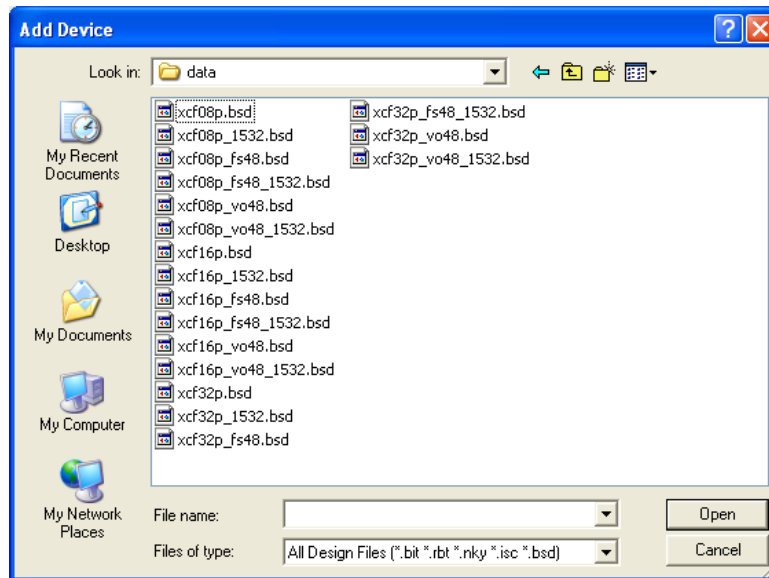
23. The Add Device File dialog will appear.



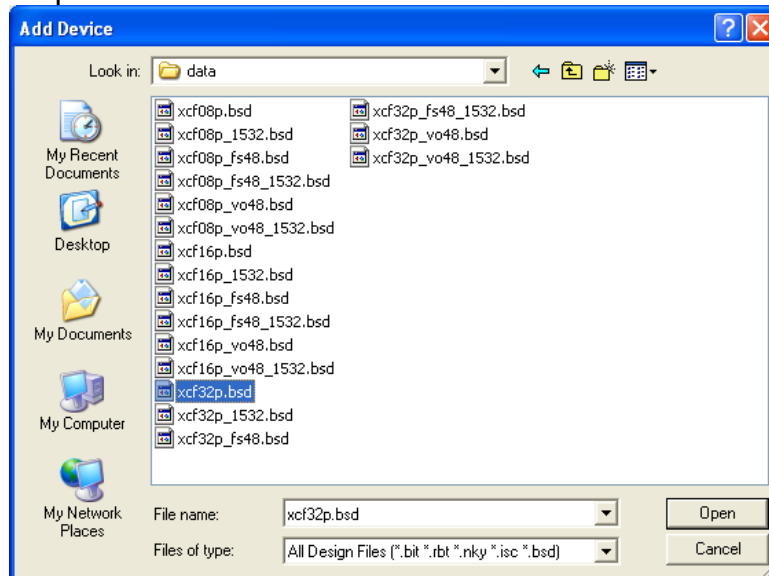
24. Navigate to the install directory of your Xilinx ISE with the PR tools.



25. Select the 'xcfp' directory and Navigate to the data subdirectory.

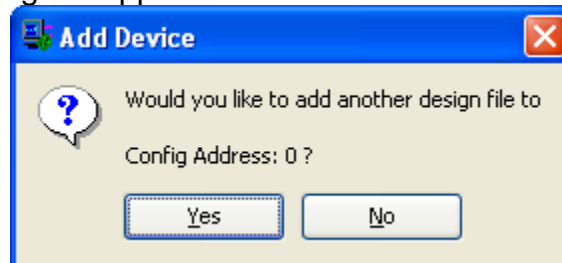


26. Select the 'xcf32p.bsd' file.

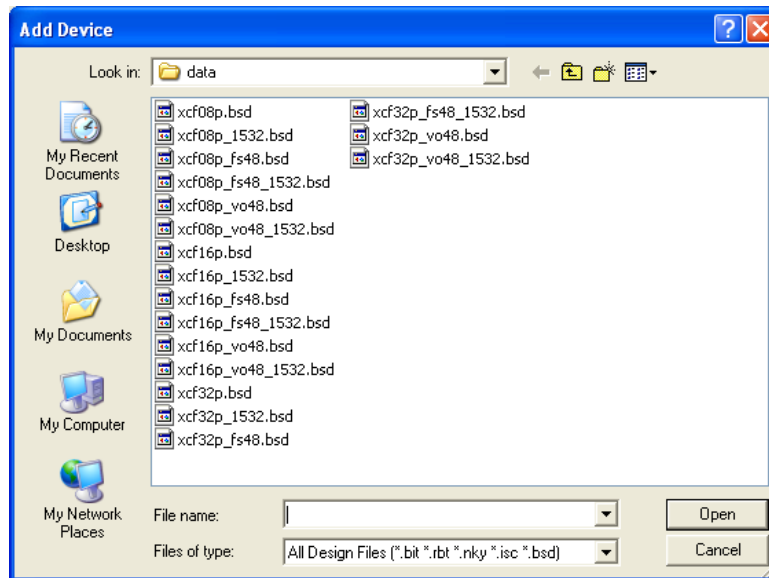


27. Click 'Open' to continue.

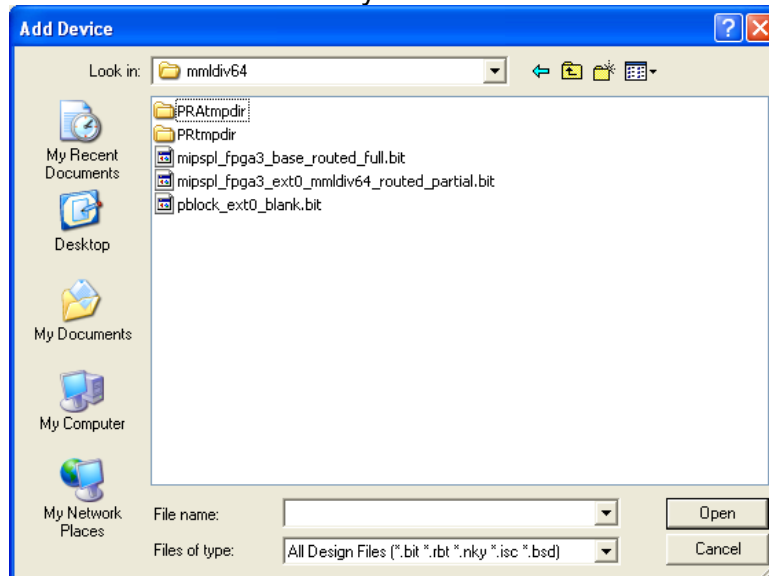
28. The Add Device dialog will appear.



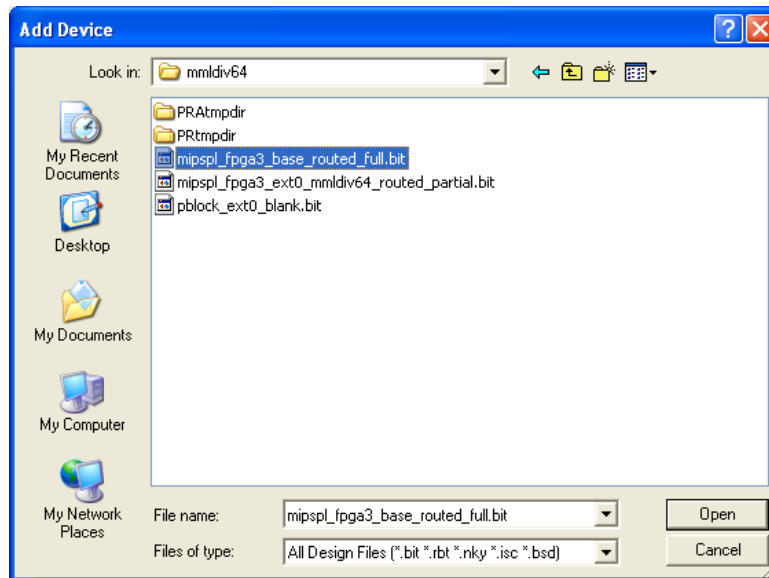
29. Click 'Yes' to continue.



30. Navigate to the location of the bit files you wish to include in this compact flash image.

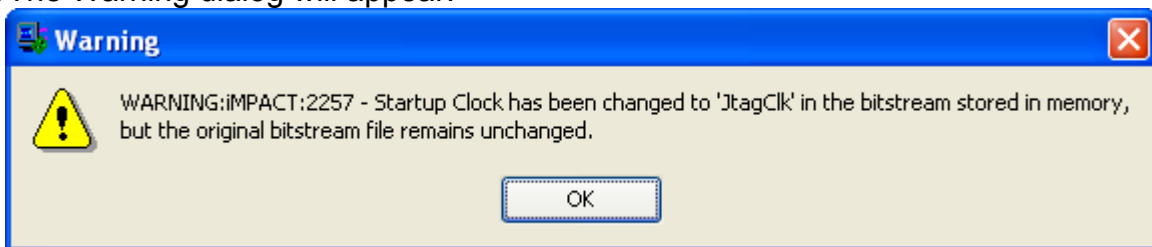


31. Select the desired bit file.



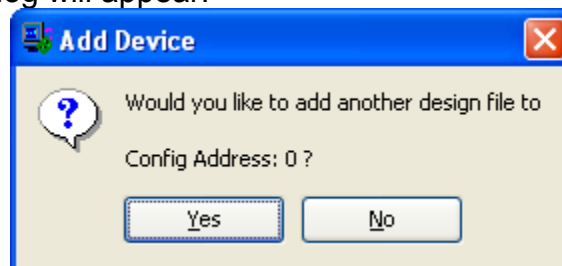
32. Click 'Open' to continue.

33. The Warning dialog will appear.

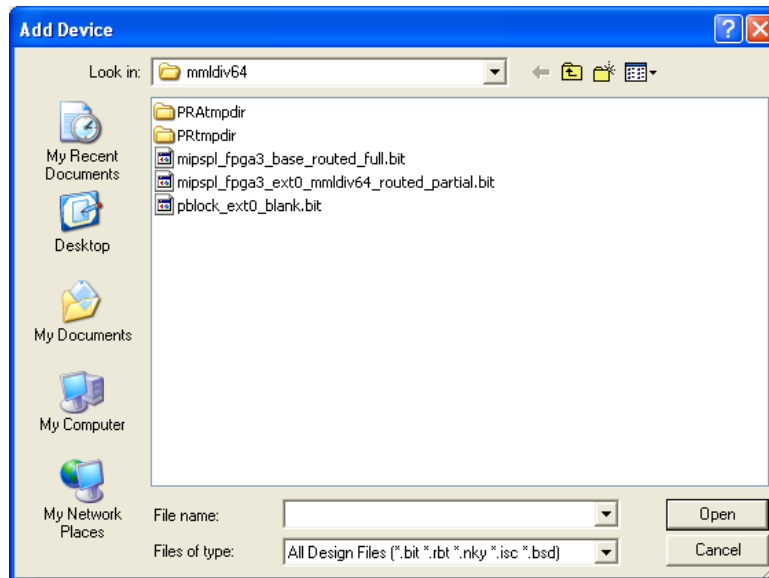


34. Click 'OK' to continue.

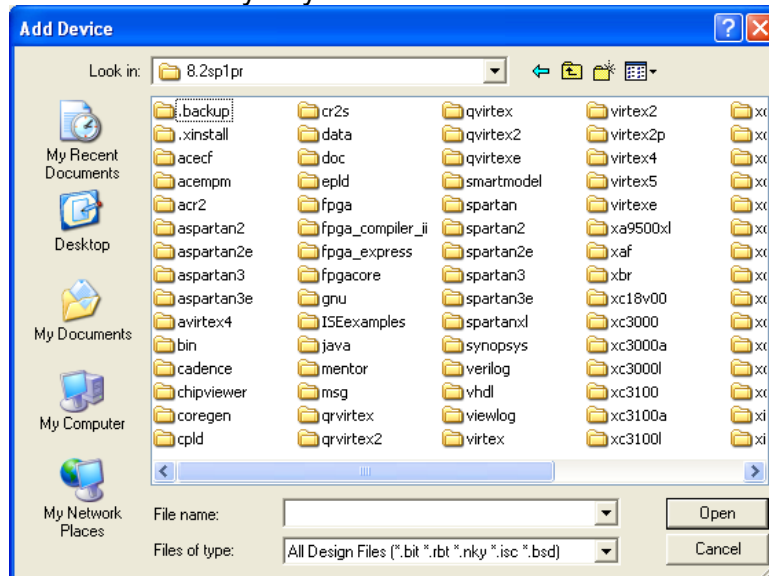
35. The Add Device dialog will appear.



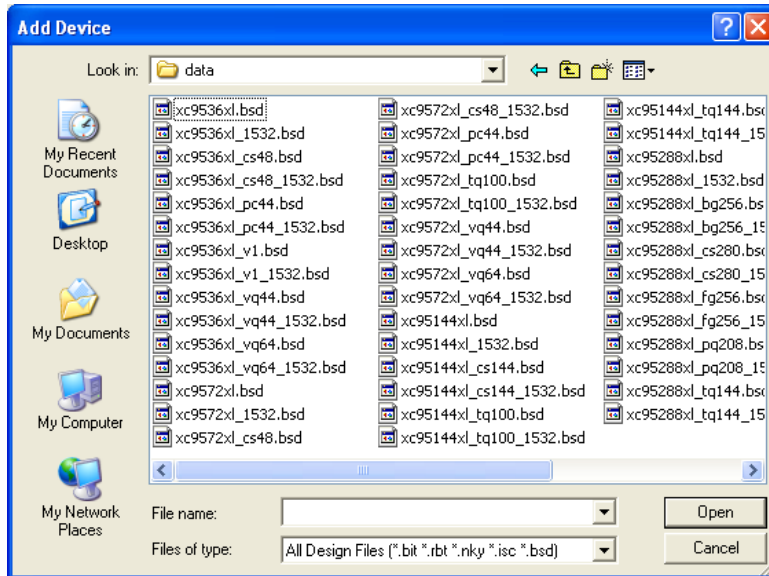
36. Click 'Yes' to continue.



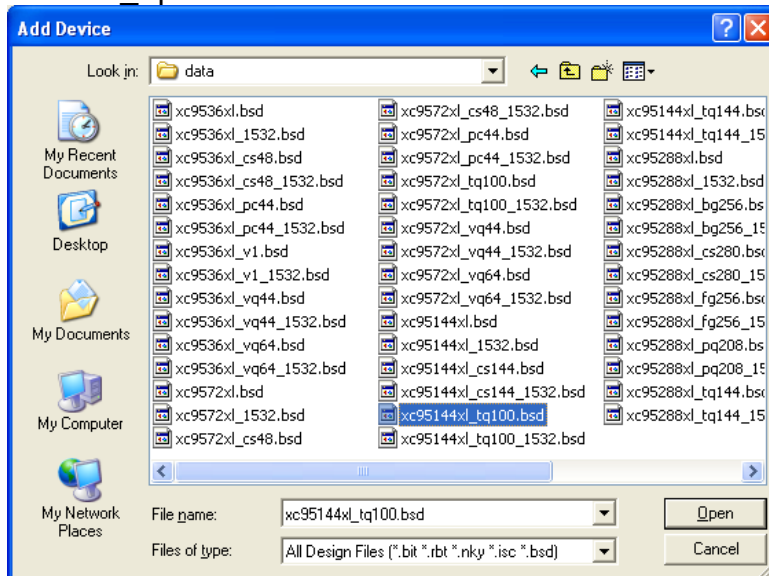
37. Navigate to the install directory of your Xilinx ISE with the PR tools.



38. Select the 'xc9500xl' directory and Navigate to the data subdirectory.

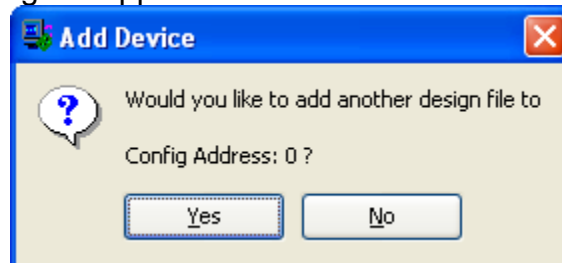


39. Select the 'xc95144xl_tq100.bsd' file.



40. Click 'Open' to continue.

41. The Add Device dialog will appear.

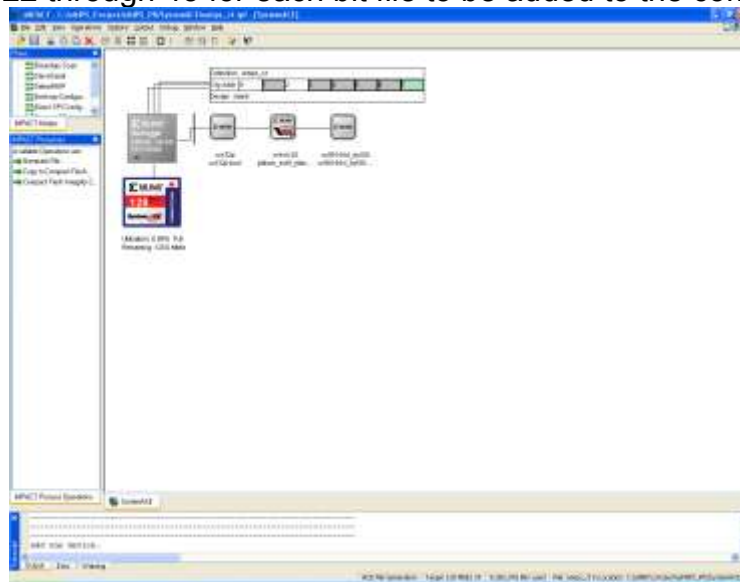


42. Click 'No' to continue.

43. The Config Address dialog should appear.

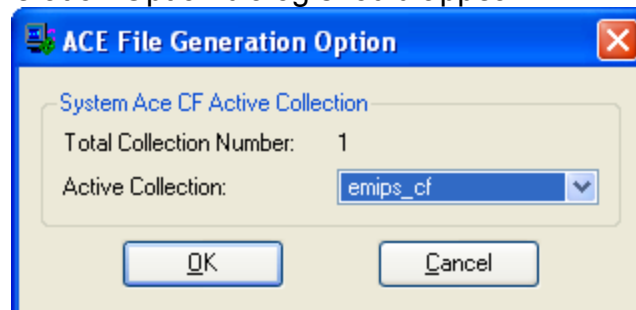


44. Repeat steps 22 through 43 for each bit file to be added to the compact flash image.

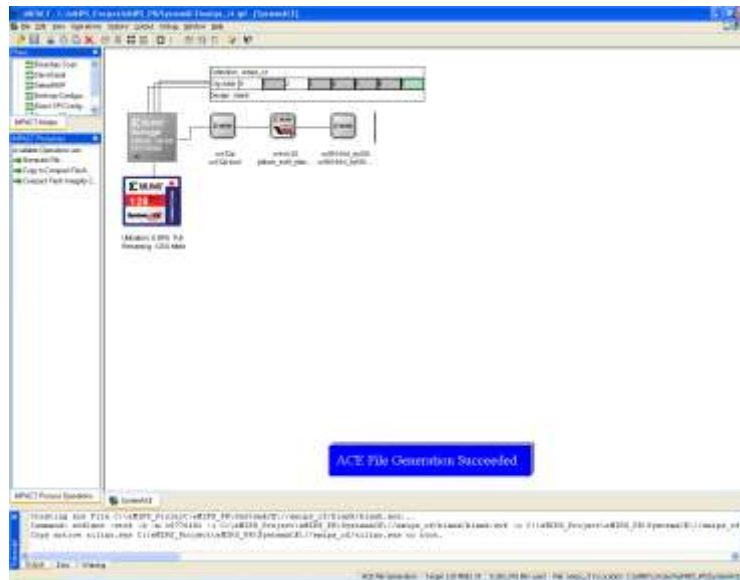


45. After you have reviewed your configurations. Right click on the compact flash icon on the screen and select 'Generate File'

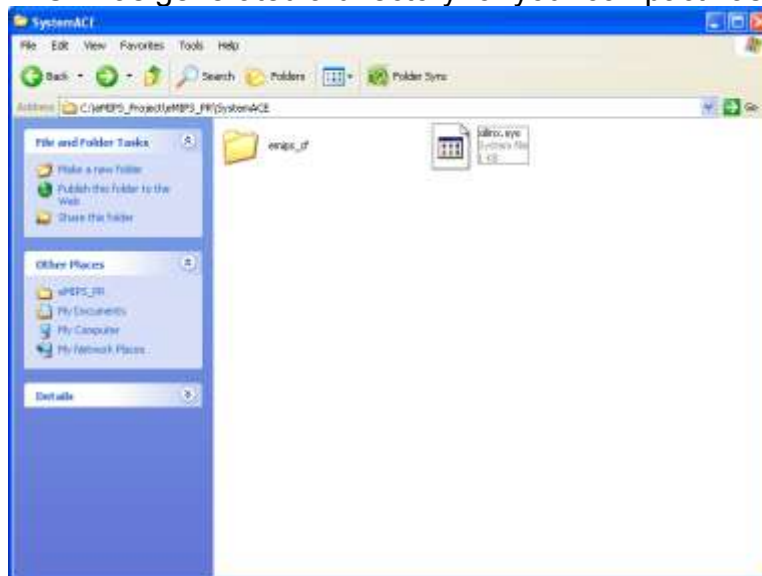
46. The ACE File Generation Option dialog should appear.



47. Click 'OK' to continue.



48. The Xilinx IMPACT has generated a directory for your compact flash.



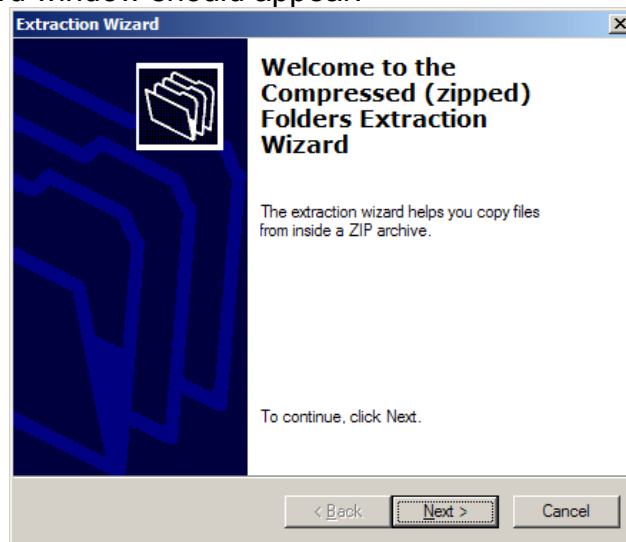
49. Copy the contents of this directory to a compact flash card no larger than 32MB.

3.5 Testing

In addition to the minimal testing described above, this distribution provides a number of other software images to test the build of the eMIPS system. These tests are classified in two classes: (a) the stand-alone tests that can be downloaded and executed in RAM using the boot loader and (b) the tests and programs that run under the Microsoft Invisible Computing operating system, which are installed on the ML401 on-board linear FLASH chip. Both sets of tests are contained in the archive file “eMIPS Tests.zip”. The following are the instructions on how to unpack and execute both sets of tests.

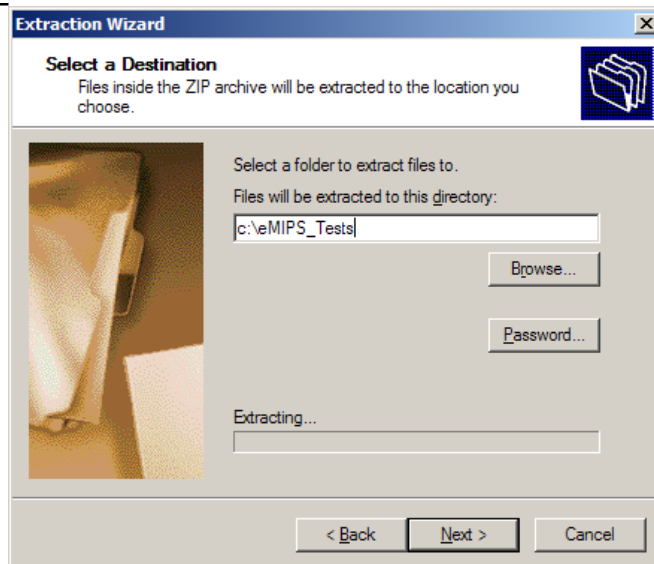
1. Extract the files from the archive. Right-click on the “eMIPS Tests.zip” zip file and select ‘Extract all’.

2. The Extraction Wizard window should appear.

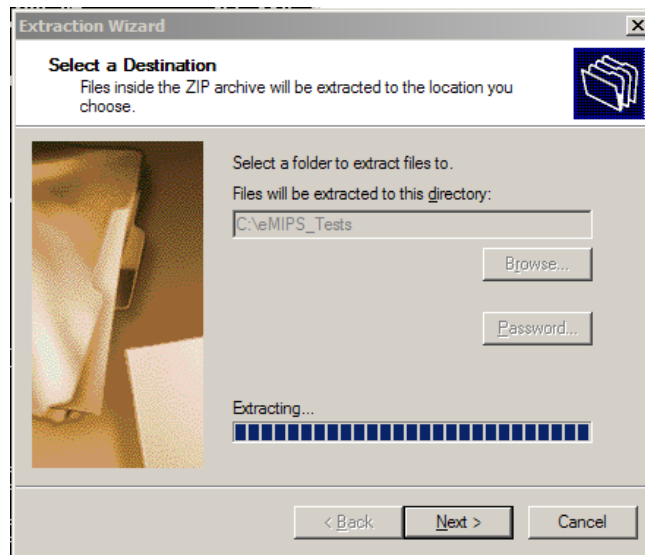


3. Please click 'Next' to continue.

10. The next screen you should see is the destination selection screen. By default it will have your current directory selected. **Change** it to some other location; we will use the folder "c:\eMIPS_Tests" in these instructions.

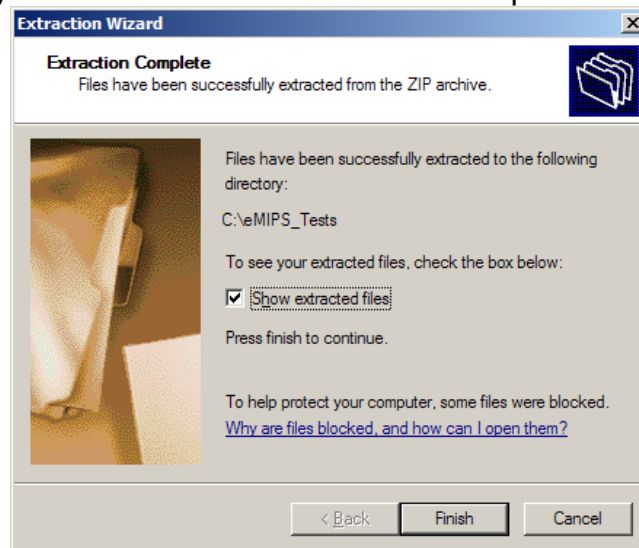


11. Click 'Next' to continue.



12. You should see the progress bar begin to fill. This may take some time depending on your system, between 5 to 10 minutes.

13. The next screen you should see is the extraction complete screen



14. Please click 'Finish' to complete the extraction of the test files.

15. The Folder containing the test programs should appear. You can now run either the stand-alone tests or the OS tests.

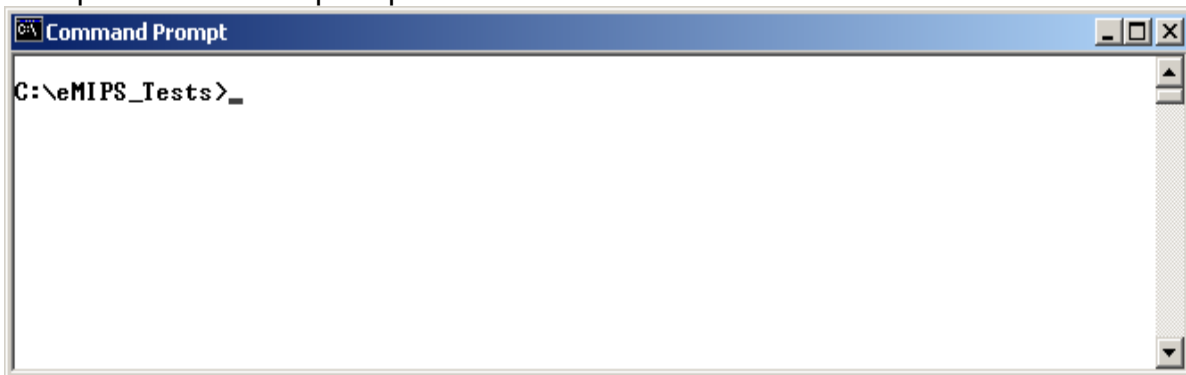
3.5.1.1 Running the stand-alone tests

Each of the stand-alone tests will run and at the end of the execution will display one of two strings: “TEST PASSED SUCCESSFULLY” for a successful completion or “TEST FAILED” in case of failure. Each stand-alone test can be run individually, using the two provided utilities DOWNLOAD.EXE and SERPLEXD.EXE. The tests can also be run in sequence, using the provided utility WIN_TESTIT.EXE. The section “Verifying the Configuration Bit Files” shows how to run an individual stand-alone test, in that case the test `mmldiv64_test2.bin`. The following stand-alone programs are provided and can be run in isolation:

<code>ctimer.bin</code>	<code>ctint.bin</code>	<code>ext_test.bin</code>
<code>jf.bin</code>	<code>mmldiv64_test2.bin</code>	<code>mneg.bin</code>
<code>neg1.bin</code>	<code>pmt.bin</code>	<code>shift.bin</code>
<code>snake.bin</code>	<code>tdiv.bin</code>	<code>tdiv2.bin</code>
<code>tdown.bin</code>	<code>techo.bin</code>	<code>tflash.bin</code>
<code>tgpio.bin</code>	<code>thfs.bin</code>	<code>thfs1.bin</code>
<code>thfs2.bin</code>	<code>tint.bin</code>	<code>tprintf.bin</code>
<code>treset.bin</code>	<code>treturn.bin</code>	<code>tsb.bin</code>
<code>tsysace.bin</code>	<code>tth1.bin</code>	<code>tthreads.bin</code>
<code>tusart.bin</code>	<code>tusart1.bin</code>	

We will first run in isolation the test `PMT.BIN`, which looks at the I/O map of the eMIPS system. We will assume, for illustration purposes, that the serial line link to the ML401 board is now on “com3:” rather than “com1:” as previously assumed, to make clear where this optional argument is used.

1. Open a command prompt in the test folder above



2. Type the command: “download com3: pmt.bin && serplexd -n -r -s com3:”
3. The test is downloaded and executed, the output should be as follows:

```

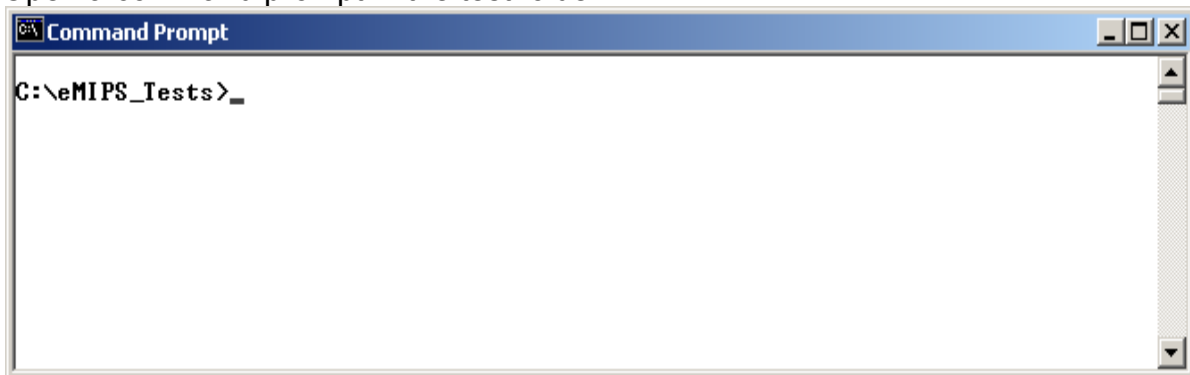
C:\eMIPS_Tests>download.exe com3: pmt.bin && serplexd.exe -n -r -s com3:
.....Download complete, 8000 bytes sent
Will NOT attempt to use the NIC
Will exchange un-encoded data ['raw' model].
Will talk to a BigEndian client (mips,ppc,...)
Console Thread ...
Peripheral Mapping Table, from the top at ffffffff
Pmt[00000000]: 00000001 -- Address = 00000000 Tag = 00000001 PMT
Pmt[00000001]: fffd0002 -- Address = fffd0000 Tag = 00000002 SRAM
SRAM[00000000].AddresssAndTag = 00000002
SRAM[00000000].Control = 00108000
Pmt[00000002]: fffa0005 -- Address = fffa0000 Tag = 00000005 INTERRUPT_CONTROLLE
R
AIC[00000000].Tag = 00000005
AIC[00000000].IrqStatus = 00000000
AIC[00000000].IrqRawStatus = 00000000
AIC[00000000].IrqEnable = 00000000
AIC[00000000].IrqEnableClear = 00000000
AIC[00000000].IrqSoft = 00000000
Pmt[00000003]: fff80007 -- Address = fff80000 Tag = 00000007 TIMER
TC[00000000].Tag = 00000007
TC[00000000].Control = 00000000
TC[00000000].FreeRunning = 0000000000000000
TC[00000000].DownCounter = 0000000000000000
TC[00000000].reserved[0] = ffffffff
TC[00000000].reserved[1] = ffffffff
Pmt[00000004]: fff90006 -- Address = fff90000 Tag = 00000006 USART
USART[00000000].Tag = 00000006
USART[00000000].Control = 02e00050
USART[00000000].IntrEnable = 00000000
USART[00000000].IntrDisable = 00000000
USART[00000000].IntrMask = 00000000
USART[00000000].ChannelStatus = 00000000
USART[00000000].RxData = 00000000
USART[00000000].TxData = ffffffff
USART[00000000].Baud = 00000029
USART[00000000].Timeout = ffffffff
USART[00000000].reserved[0] = ffffffff
USART[00000000].reserved[1] = ffffffff
USART[00000000].reserved[2] = ffffffff
USART[00000000].reserved[3] = ffffffff
USART[00000000].reserved[4] = ffffffff
USART[00000000].reserved[5] = ffffffff
Pmt[00000005]: fff60009 -- Address = fff60000 Tag = 00000009 GPIO
PIO[00000000].Tag = 00000009
PIO[00000000].Enable = 000000ff
PIO[00000000].Disable = ffffffff
PIO[00000000].Direction = 00000000
PIO[00000000].OutDisable = ffffffff
PIO[00000000].PinData = 00000000
PIO[00000000].PinData = 00000000
PIO[00000000].ClearData = ffffffff
PIO[00000000].PinStatus = 00000000
PIO[00000000].IntrStatus = 00000000
PIO[00000000].IntrEnable = 00000000
PIO[00000000].IntrDisable = ffffffff
PIO[00000000].IntrTrigger = 00000000
PIO[00000000].reserved[0] = ffffffff
PIO[00000000].reserved[1] = ffffffff
PIO[00000000].reserved[2] = ffffffff
Pmt[00000006]: fffb0004 -- Address = fffb0000 Tag = 00000004 FLASH
FLASH[00000000].AddresssAndTag = f0000004
FLASH[00000000].Control = 0080801b
Pmt[00000007]: fff5000a -- Address = fff50000 Tag = 0000000a SYSTEM_ACE
ACE[00000000].Tag = 0000000a
ACE[00000000].Control = 4f020100
ACE[00000000].BUSMODEREG = 00000101
ACE[00000000].STATUS = 15151010
ACE[00000000].ERRORREG = 00000000
ACE[00000000].CFGLBAREG = 00000000
ACE[00000000].MPULBAREG = 00000000
ACE[00000000].VERSIONREG = 00000c0c
ACE[00000000].SECCNTCMDREG = 00000101
ACE[00000000].CONTROLREG = 00000808
ACE[00000000].PATSTATREG = 00000000
Pmt[00000008]: ffee0011 -- Address = ffee0000 Tag = 00000011 EXTENSION_CONTROLLE
R
Pmt[00000009]: 0000ffff -- Address = 00000000 Tag = 0000ffff END_OF_TABLE
TEST PASSED SUCCESSFULLY!_

```

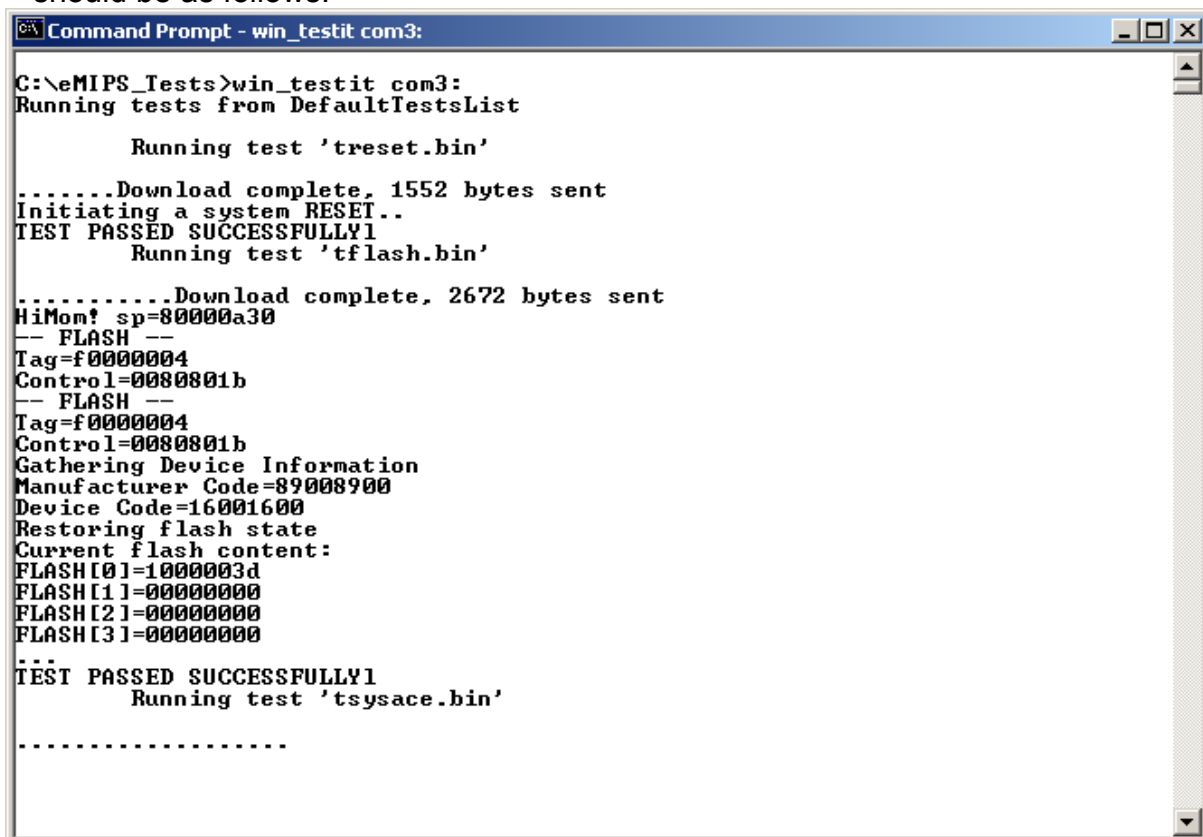
3.5.1.2 Running all the stand-alone tests

The stand-alone tests can be run in sequence using the provided utility WIN_TESTIT.EXE. This program contains a list of tests and their arguments, that are used by the eMIPS developers to verify a new bitfile before running the OS tests.

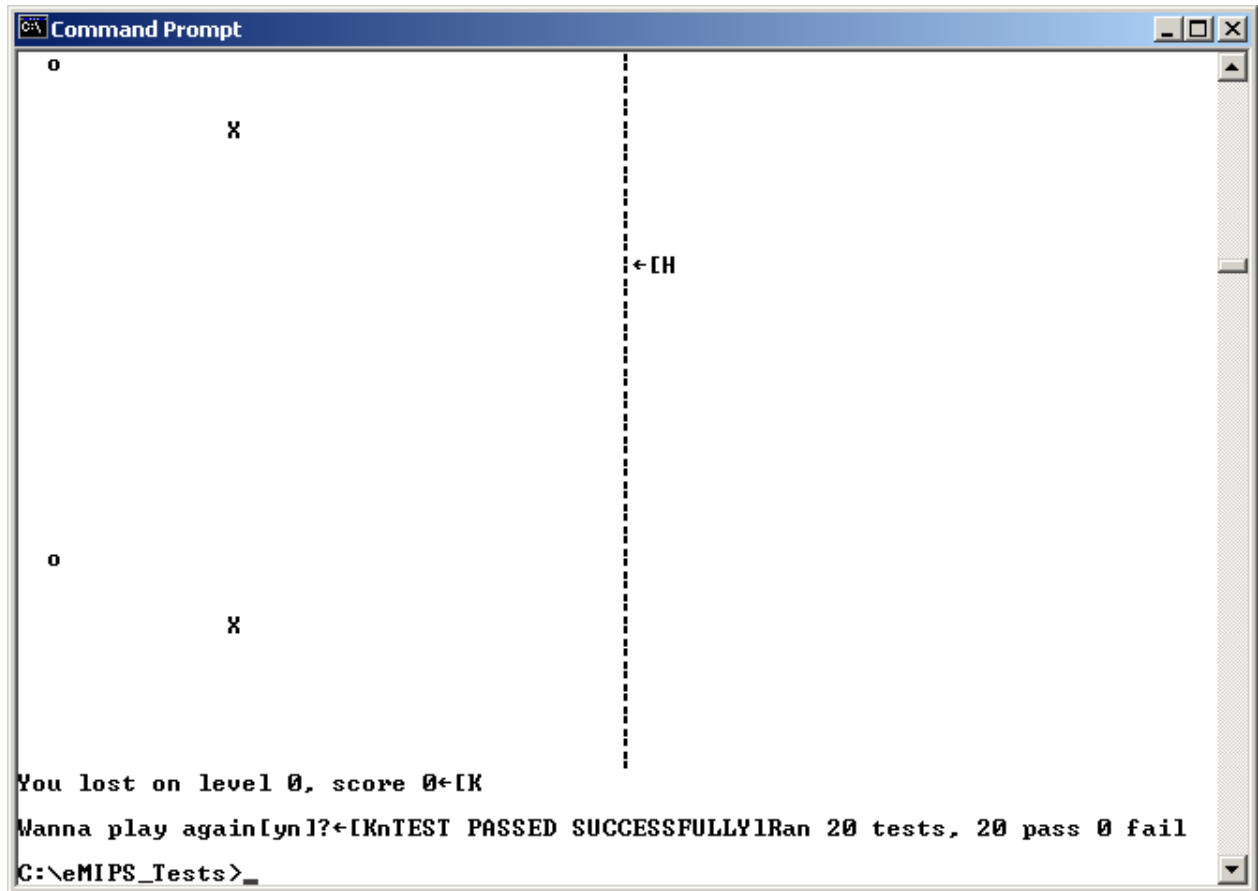
1. Open a command prompt in the test folder



2. Type the command: "win_testit com3:"
3. The program runs and downloads each of the tests in sequence, the initial output should be as follows:



4. The tests will take some time to execute, between 10 and 15 minutes. The final output should be as follows:

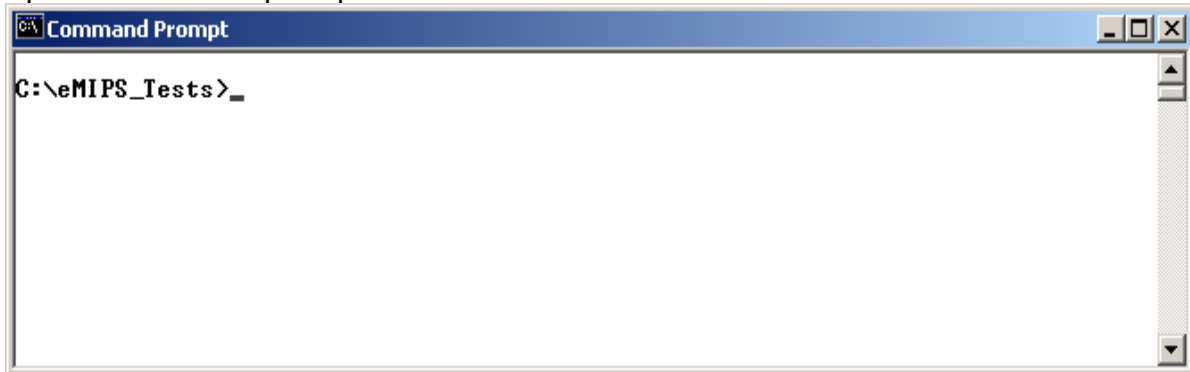


```
Command Prompt
0
X
0
X
You lost on level 0, score 0
Wanna play again[yn]?
TEST PASSED SUCCESSFULLY!
Ran 20 tests, 20 pass 0 fail
C:\eMIPS_Tests>_
```

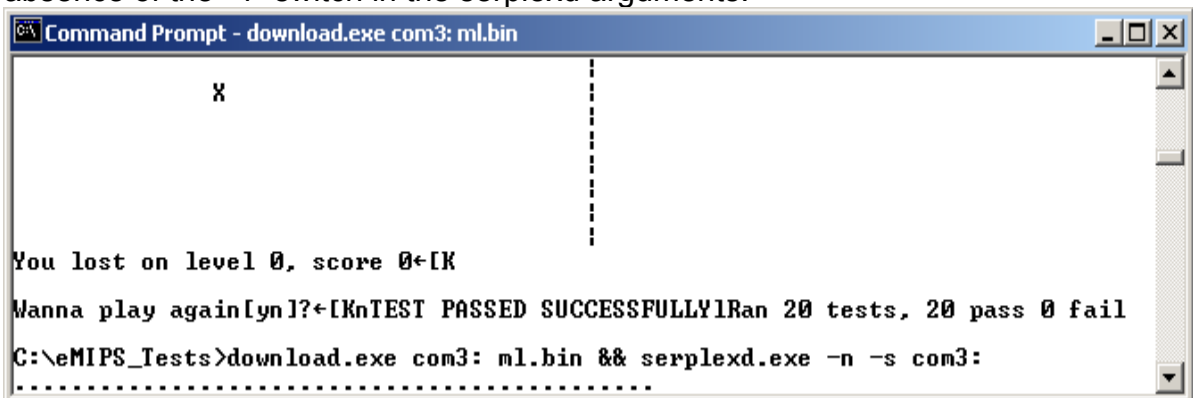
3.5.1.3 Running the OS tests

The OS-level tests require the flashing of the bootable image ML.BIN and of the file system image MLFS.FLP into the ML401 linear FLASH. The following sequence of procedures will boot the OS via the boot loader, execute the provided utility STRATAFLASH.EXE to flash those images, reboot the OS from flash and run a simple test.

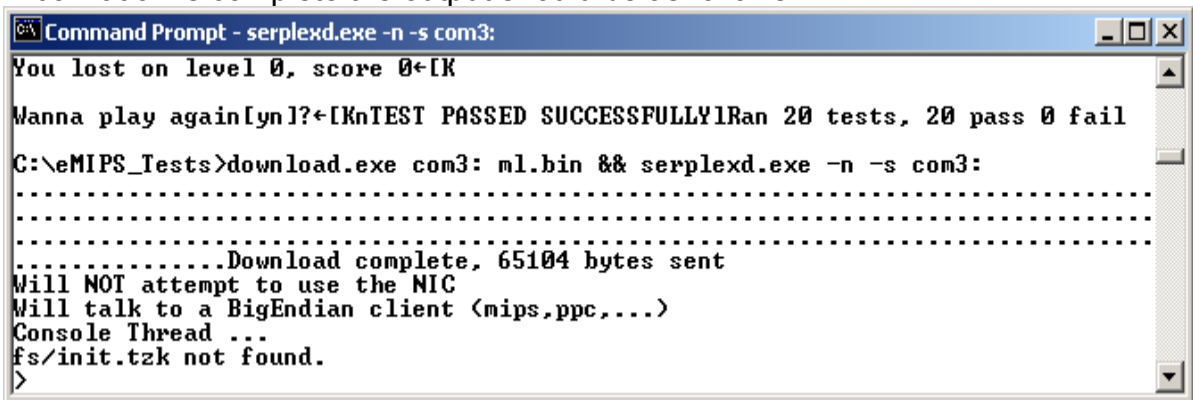
1. Open a command prompt in the test folder



2. Make sure the switches 1 and 2 on the GPIO DIP switches of the ML401 board are still in the down position.
3. Type the command: "download com3: ml.bin && serplexd -n -s com3:". Notice the absence of the "-r" switch in the serplexd arguments.



4. The OS image is downloaded and starts to run, loading the first application program (make sure the file TZK.COB has not been removed from the test folder). Once initialization is complete the output should be as follows:



5. Type a few simple commands to verify the command shell is working properly.

```

C:\eMIPS_Tests>download.exe com3: ml.bin && serplexd.exe -n -s com3:
.....
.....Download complete, 65104 bytes sent
Will NOT attempt to use the NIC
Will talk to a BigEndian client (mips,ppc,...)
Console Thread ...
fs/init.tzk not found.
> ls
vtables
COB
com1
serplex6a
srfs
serplex6d
serplex6c
sysace
stdin
stdout
stderr
fs
> memory
      Total      Free  Reserved   Commit      Used   MaxUsed
983272    958352    983272    983272    24920    30512
>

```

6. At this point we want to flash the OS image to flash. Type the following command to the shell: “strataflash.exe f0000000 fs\ml.bin 0 TWINCHIPS BUFFERED”. Please note that the arguments are case-sensitive. After the program has been loaded over the serial line the output is as follows:

```

fs
> memory
      Total      Free  Reserved   Commit      Used   MaxUsed
983272    958352    983272    983272    24920    30512
> strataflash.exe f0000000 fs\ml.bin 0 TWINCHIPS BUFFERED
Checking FLASH ef000000
ManufacturerId: x89 DeviceId: x16
Flash was recognized, type 28F320
None of the sectors are locked.
Are you sure you want to do this ? _

```

7. Answer “y” to continue. The file is downloaded and flashed, progress is marked by hash marks. When flashing is complete the output is as follows:

```

ManufacturerId: x89 DeviceId: x16
Flash was recognized, type 28F320
None of the sectors are locked.
Are you sure you want to do this ? y
FLASHing file fs\ml.bin at offset 0.

Erasing sector 0 [offset x0] ...done.
##### done.
ByteSum=x3dee10 (4058640).
>

```

8. At this point the base OS image is flashed. Now we want to flash the file system image. Type the following command: “strataflash.exe f0000000 fs\mlfs.flp 80000 TWINCHIPS BUFFERED”. Please note that the arguments are case-sensitive. After the program has been loaded over the serial line the output is as follows:

```

Command Prompt - serplexd.exe -n -s com3:

Erasing sector 0 [offset x0] ...done.
##### done.
ByteSum=x3dee10 (4058640).
> strataflash.exe f0000000 fs\mlfs.flp 80000 TWINCHIPS BUFFERED
Checking FLASH 0f000000
ManufacturerId: x89 DeviceId: x16
Flash was recognized, type 28F320
None of the sectors are locked.
Are you sure you want to do this ?

```

9. Answer “y” to continue. The file is downloaded and flashed; progress is marked by hash marks. This process should take between 5 and 10 minutes. When complete the output is as follows:

```

Command Prompt - serplexd.exe -n -s com3:

> strataflash.exe f0000000 fs\mlfs.flp 80000 TWINCHIPS BUFFERED
Checking FLASH 0f000000
ManufacturerId: x89 DeviceId: x16
Flash was recognized, type 28F320
None of the sectors are locked.
Are you sure you want to do this ? y
FLASHing file fs\mlfs.flp at offset 80000.

Erasing sector 2 [offset x80000] ...done.
#####
Erasing sector 3 [offset xc0000] ...done.
#####
Erasing sector 4 [offset x100000] ...done.
#####
Erasing sector 5 [offset x140000] ...done.
##### done.
ByteSum=x3016ff8 (50425848).
>

```

10. At this point both the OS image and the file system images have been flashed. To test them please flip the switches 1 and 2 on the GPIO DIP switches of the ML401 board to the UP position. Then hit the “CPU Reset” button of the ML401 board. The output is as follows:

```

Command Prompt - serplexd.exe -n -s com3:

Erasing sector 5 [offset x140000] ...done.
##### done.
ByteSum=x3016ff8 (50425848).
> GPIO=3
fs/init.tzk not found.
>

```

11. You can type a few simple commands to test the basic functionality:

```

C:\> Command Prompt - serplexd.exe -n -s com3:
> GPIO=3
fs/init.tzk not found.
> ll.exe \

Namespace \

                vtables*
                COB*
                com1
                serplex6a
                srfs*
                serplex6d
                serplex6c
1025482752      sysace
                stdin
                stdout
                stderr
                fs*

> ll.exe fs

Namespace fs

20416 tzk.cob
8168 sr0.exe
7764 sernet.cob
5228 lsmodule.exe
123404 protocol.cob
7716 ll.exe
33920 sock.exe
16660 datetime.exe
13784 hostfstest.exe
8972 tokenizer.cob
18172 sax.cob
60768 http.cob
21696 soap.cob
212444 convert.cob
4268 tmp_heap.cob
13968 wsaddr.cob
28456 wsdiscov.cob
44032 wsman.cob
42612 cimdb.cob
4108 cimsample.cob
4796 base64.cob
7268 sensor.cob
34412 planner.cob
10424 statistics.cob
3572 sampling.cob
30736 snTP.cob
25192 wsres.cob
10768 dssp.cob
10492 hostfs.cob
45240 ping.exe
16896 calc.cob
11228 tsysace.exe
4416 ext_test.exe
78 net.tzk
20848 strataflash.exe
62 flash_fs.tzk
26888 fatfstest.exe
24016 mmldiv64_ext.exe

```

This concludes the basic OS level tests and the board is operational under the eMIPS system.

The use of the DIP switches in software is as follows:

1. Used by the eMIPS built-in bootloader. If read as “0” (DOWN position) the bootloader downloads an image from the serial line into SRAM, then jumps to it. You should use the DOWNLOAD.EXE utility to communicate with the bootloader. If read as “1” (UP position) the bootloader jumps directly to the linear FLASH.
2. Used by the OS. If read as “0” (DOWN position) it uses the serial line exclusively to communicate with the SERPLEXD.EXE utility and to access all executable images and data remotely, over the serial line. If read as “1” (UP position) it looks first at the linear FLASH for a filesystem image. The OS still enables the serial line file system (“hostfs” protocol) using the name “srfs”.

3.5.1.4 Running the Web Server

This demo has the following pre-requisites:

- A. The OS image and the file system image should have been written to the linear FLASH. See previous section for detailed instructions.
 - B. The VirtualPC NIC driver must have been installed on the Windows host machine. You can either use the VirtualPC product from <http://www.microsoft.com/windows/products/winfamily/virtualpc/default.msp> (recommended) or use just the virtual NIC driver from the Microsoft Invisible Computing distribution at <http://research.microsoft.com/invisible/> Look under Source:src:drivers:net:packet:lib:i386: VMNetSrv.msi
 - C. Your network must be running a DHCP server capable of assigning IP addresses to a previously unknown Ethernet address.
1. Open a command prompt in the test folder



2. Make sure the switches 1 and 2 on the GPIO DIP switches of the ML401 board are in the up position, as they were left for instance by the previous test. Start the SERPLEXD.EXE utility with the following command: “serplexd –s com3:”. Note that we do not use either the “-r” nor the “-n” command arguments. You might optionally hit the “CPU Reset” button to get a clean start. The output is as follows:

```

C:\eMIPS_Tests>serplexd.exe -s com3:
Will talk to a BigEndian client (mips,ppc,...)
Warning: VirtualPC Network driver has unexpected version (Actual: 20004, Expected: 20006)
Console Thread ...

> GPIO=3
fs/init.tzk not found.
>

```

3. A possible warning about an unexpected version of the driver can be ignored. Type the following command: “source fs\net.tzk”. This might take some time, depending on the traffic on your network. Your DHCP server should assign an IP address to the board and the output is similar to this:

```

C:\eMIPS_Tests>
C:\eMIPS_Tests>serplexd.exe -s com3:
Will talk to a BigEndian client (mips,ppc,...)
Warning: VirtualPC Network driver has unexpected version (Actual: 20004, Expected: 20006)
Console Thread ...

> GPIO=3
fs/init.tzk not found.
> source fs\net.tzk
dhcp_sub.c:Could not bind to file hostname.txt
Interface sr0 has IP address 172.31.41.212.
The DHCP information is valid, our lease will expire in 345594 seconds.
The IP address 172.31.41.212 was leased from the DHCP Server 172.31.40.6
Default route through gateway 172.31.40.1 with subnet mask 255.255.248.0
There are 3 DNS server(s)
    239.255.255.253
    157.54.14.162
    157.54.14.178
The domain name on this interface is 'redmond.corp.microsoft.com'
>

```

4. In our case, the board was assigned the IP address 172.31.41.212. Next we start the http server, type the following command: “http.cob”. The output is as follows:

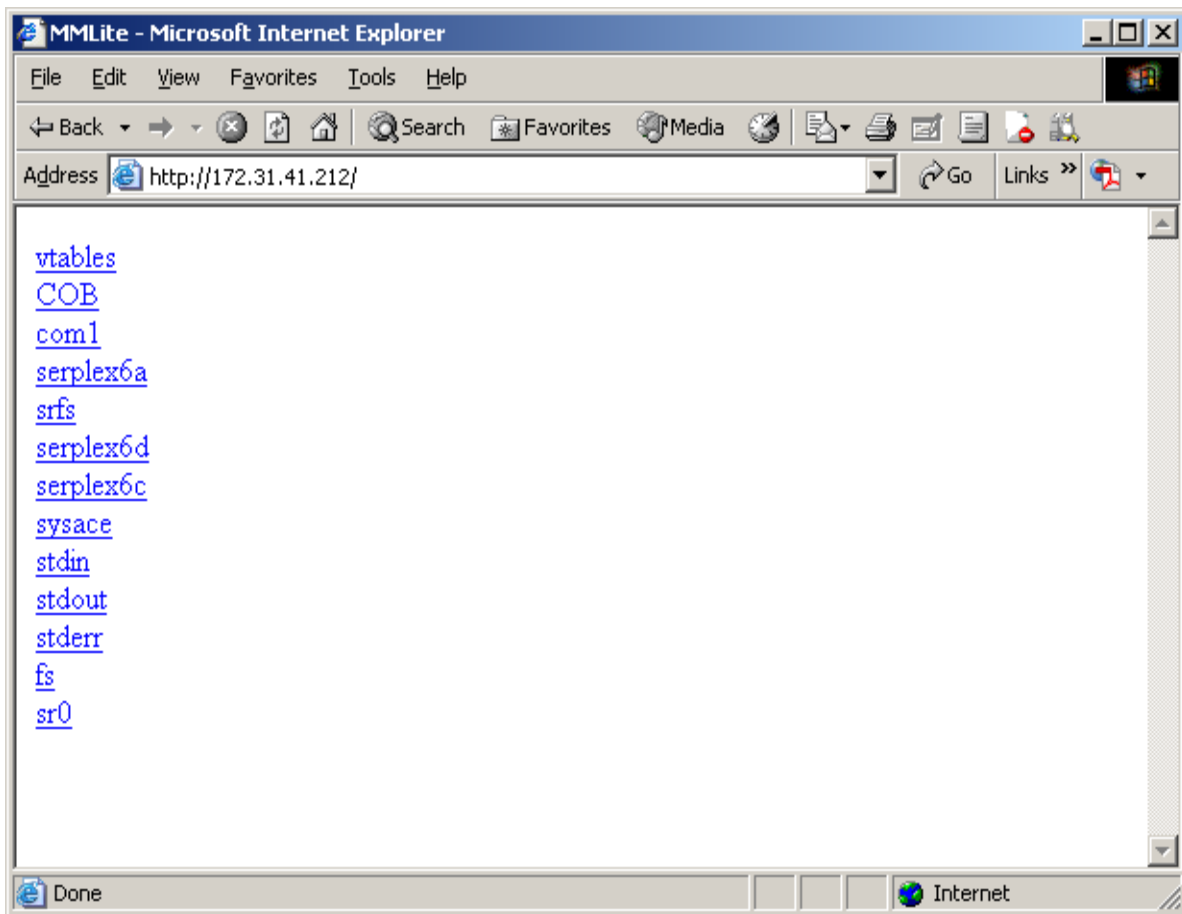
```

C:\eMIPS_Tests>
C:\eMIPS_Tests>serplexd.exe -s com3:
Will talk to a BigEndian client (mips,ppc,...)
Warning: VirtualPC Network driver has unexpected version (Actual: 20004, Expected: 20006)
Console Thread ...

> GPIO=3
fs/init.tzk not found.
> source fs\net.tzk
dhcp_sub.c:Could not bind to file hostname.txt
Interface sr0 has IP address 172.31.41.212.
The DHCP information is valid, our lease will expire in 345594 seconds.
The IP address 172.31.41.212 was leased from the DHCP Server 172.31.40.6
Default route through gateway 172.31.40.1 with subnet mask 255.255.248.0
There are 3 DNS server(s)
    239.255.255.253
    157.54.14.162
    157.54.14.178
The domain name on this interface is 'redmond.corp.microsoft.com'
> http.cob
>

```

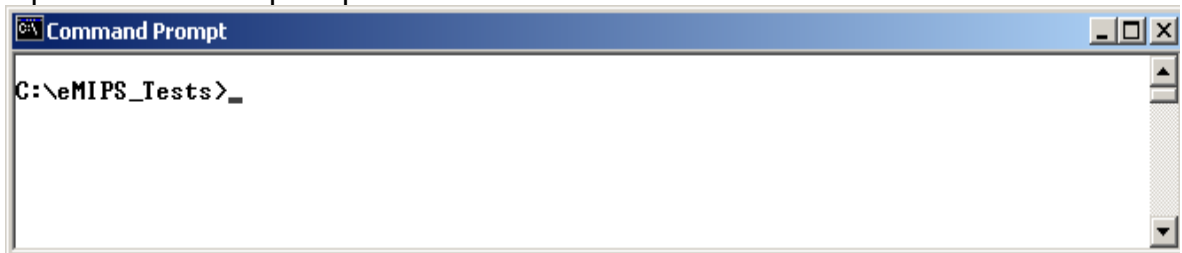
5. The http server is now operational. To test it, open a browser windows and point it to the URL [http://<board IP address>](http://172.31.41.212) which in our case is <http://172.31.41.212>. The output from the browser is as follows:



3.5.1.5 Testing the CompactFlash card

This test assumes that a properly formatted CompactFlash card is inserted in the CF connector on the ML401 board. The test is non-destructive; you can safely use the Xilinx CF card that came with your board.

1. Open a command prompt in the test folder



2. Make sure the switches 1 and 2 on the GPIO DIP switches of the ML401 board are in the up position, as they were left for instance by the previous test. Start the SERPLEXD.EXE utility with the following command: "serplexd -s -n com3:". Note that we do not use the "-r" command argument. You might optionally hit the "CPU Reset" button to get a clean start. The output is as follows:


```

C:\eMIPS_Tests>serplexd.exe -s -n com3:
Will talk to a BigEndian client (mips,ppc,...)
Will NOT attempt to use the NIC
Console Thread ...

> GPIO=3
fs/init.tzk not found.
>

```

3. Type the following command: “start fatfstest.exe fat sysace”. The output is as follows:

```

Will talk to a BigEndian client (mips,ppc,...)
Will NOT attempt to use the NIC
Console Thread ...

> GPIO=3
fs/init.tzk not found.
> start fatfstest.exe fat sysace
>

```

4. Test that the “fat” filesystem is present and look at the CF Card content. Type the following commands in sequence: “ls” and “ll.exe fat”. The output should be something like the following:

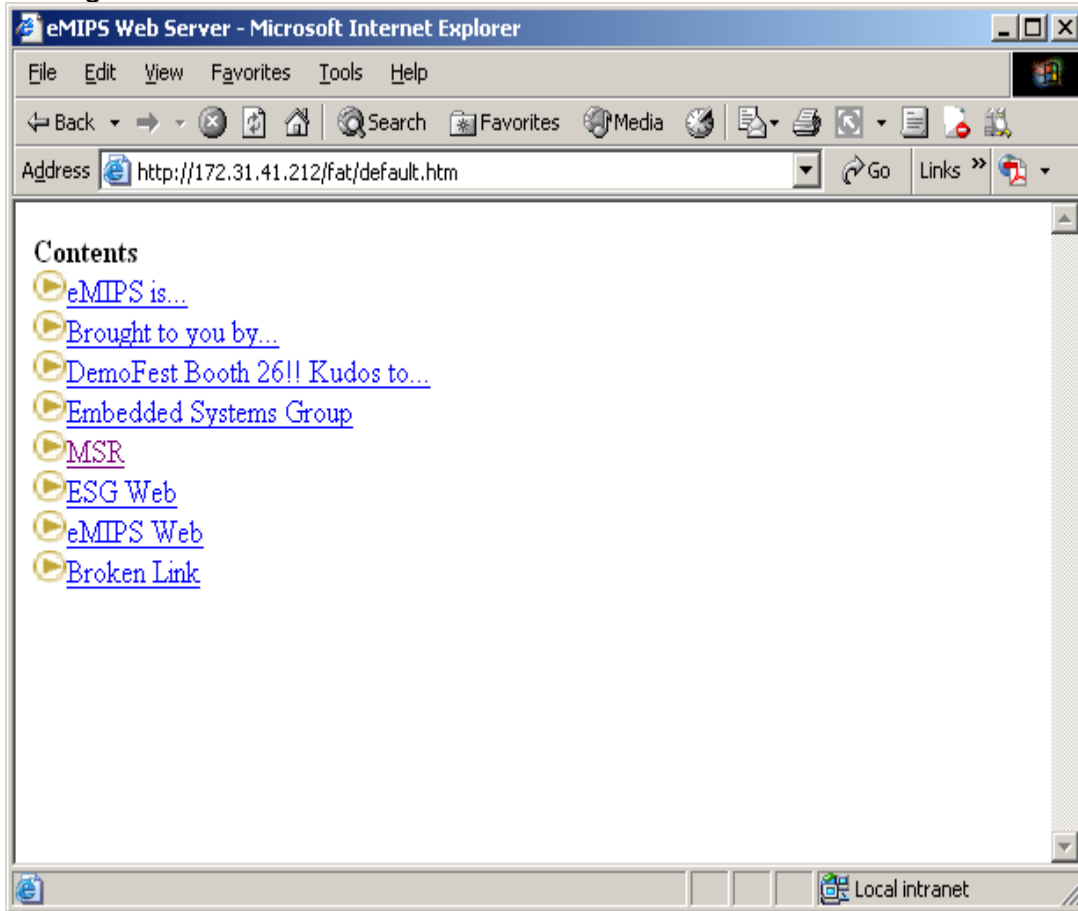
```

> start fatfstest.exe fat sysace
> ls
vtables
GOB
com1
serplex6a
srfs
serplex6d
serplex6c
sysace
stdin
stdout
stderr
fs
fat
fatfstest.exe
> ll.exe fat\
Namespace fat\
          525 404.htm
          903 arrow.gif
          942 contenta.htm
          463 header.htm
        921654 image01.bmp
        921654 image02.bmp
        921654 image03.bmp
        921654 image04.bmp
        921654 image05.bmp
        921654 image06.bmp
        921654 image07.bmp
          1328 index.htm
        13845 large.htm
          521 main.htm
          587 mypage.htm
          403 simple.htm
        397870 sound.wav
          239 xilinx.sys
          m1401/*
          XILINX/*
          mmlite/*
        26819 eMIPS.jpg
        2881  neil.jpg
        2457  sandro.jpg
        2534  juh.jpg
        16846 bharat.jpg
        8102  Brandon.jpg
        7237  liu.jpg
        3209  oscar.jpg
        19156 sabin_mohan.jpg
        51463 Special.jpg
          1494 b26.htm
          1571 default.htm
          564  esg.htm
          444  nas.htm
          7123 phil_face.jpg
          7105 warren.jpg
          5844 karlm.jpg
          9605 hong.jpg
        26840 Nishith.jpg
>

```

You can add files to the CF card; in our case we have added a few files for a simple web site. Repeating the previous procedure we start the http server, and then we can look at the web

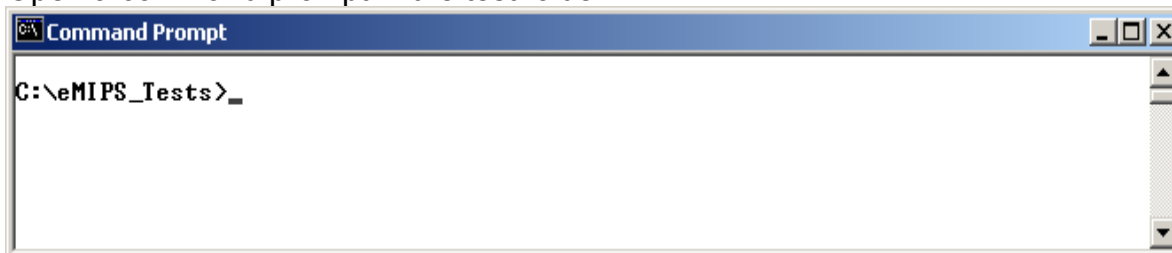
site using the URL “http://172.31.41.212/fat/default.htm”. The output from the browser might be something like this:



3.5.1.6 Testing an Extension using SystemACE

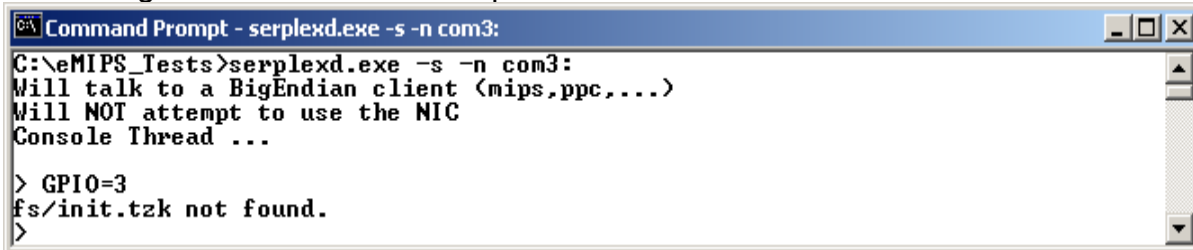
This test assumes that a properly formatted CompactFlash card is inserted in the CF connector on the ML401 board. It further assumes that the partial bit file pblock_ext0_blank.ace has been generated using the procedures described in section 3.4, and located in slot number 4 of the CF card. The test is non-destructive, no file will be modified.

5. Open a command prompt in the test folder



6. Make sure the switches 1 and 2 on the GPIO DIP switches of the ML401 board are in the UP position, as they were left for instance by the previous test. Start the SERPLEXD.EXE utility with the following command: “serplexd -s -n com3:”. Note that

we do not use the “-r” command argument. You might optionally hit the “CPU Reset” button to get a clean start. The output is as follows:



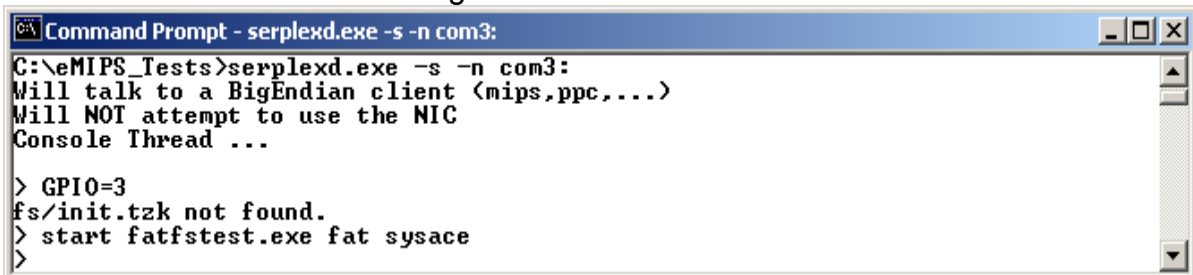
```

Command Prompt - serplexd.exe -s -n com3:
C:\eMIPS_Tests>serplexd.exe -s -n com3:
Will talk to a BigEndian client <mips,ppc,...>
Will NOT attempt to use the NIC
Console Thread ...

> GPIO=3
fs/init.tzk not found.
>

```

7. Type the following command: “start fatfstest.exe fat sysace”. The output is as follows, there should be no error messages:



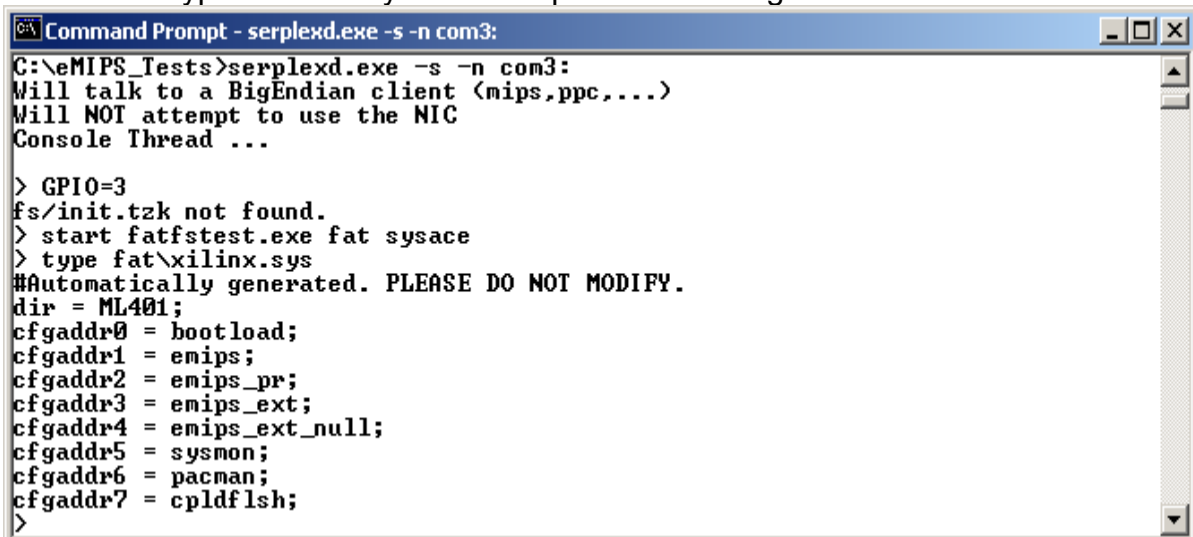
```

Command Prompt - serplexd.exe -s -n com3:
C:\eMIPS_Tests>serplexd.exe -s -n com3:
Will talk to a BigEndian client <mips,ppc,...>
Will NOT attempt to use the NIC
Console Thread ...

> GPIO=3
fs/init.tzk not found.
> start fatfstest.exe fat sysace
>

```

8. Check the content of the Xilinx configuration file on the CF card. Type the following command: “type fat\xilinx.sys”. The output is something like this:



```

Command Prompt - serplexd.exe -s -n com3:
C:\eMIPS_Tests>serplexd.exe -s -n com3:
Will talk to a BigEndian client <mips,ppc,...>
Will NOT attempt to use the NIC
Console Thread ...

> GPIO=3
fs/init.tzk not found.
> start fatfstest.exe fat sysace
> type fat\xilinx.sys
#Automatically generated. PLEASE DO NOT MODIFY.
dir = ML401;
cfgaddr0 = bootload;
cfgaddr1 = emips;
cfgaddr2 = emips_pr;
cfgaddr3 = emips_ext;
cfgaddr4 = emips_ext_null;
cfgaddr5 = sysmon;
cfgaddr6 = pacman;
cfgaddr7 = cpldflsh;
>

```

9. In our case the extension file is in slot number 4 (cfgaddr4), in the directory ML401\emips_ext_null. Type the following command to ask the SystemACE to load the ace file for the test: “tsysace.exe config4”. The output is as follows:

```
Command Prompt - serplexd.exe -s -n com3:
>
C:\eMIPS_Tests>serplexd.exe -s -n com3:
Will talk to a BigEndian client (mips,ppc,...)
Will NOT attempt to use the NIC
Console Thread ...

> GPIO=3
fs/init.tzk not found.
> start fatfstest.exe fat sysace
> type fat\xilinx.sys
#Automatically generated. PLEASE DO NOT MODIFY.
dir = ML401;
cfgaddr0 = bootload;
cfgaddr1 = emips;
cfgaddr2 = emips_pr;
cfgaddr3 = emips_ext;
cfgaddr4 = emips_ext_null;
cfgaddr5 = sysmon;
cfgaddr6 = pacman;
cfgaddr7 = cpldflsh;
> tsysace.exe config4
Asking Sysace@ffff50000 to load configuration bitfile 4 from CPLASH
Test PASSED ok.
>
```

10. Check that the red LED marked “Err” on the board is NOT lit. If it is lit there is an error and the test failed.
11. Otherwise you can proceed to run the software that makes use of your extension, to verify that your design works.

4 Software Procedures

This section describes a number of procedures and tools that can be useful for software development under the eMIPS system.

4.1 Building and running a standalone program

A standalone program is a program that is run immediately after the processor comes out of reset, after the boot loader has initialized some on-chip peripherals. These programs are usually hardware tests, such as those described in section 3.5.1.1, or an operating system such as the Microsoft Invisible Computing RTOS.

To understand the exact conditions of the processor at the time the program receives control it is recommended the user reads and understands the source of the boot loader program. The source is in the file *BramLoader\bram2.s*. Generally speaking, the processor will be in the state of a cold reset and the peripherals will not be initialized. Some exceptions include the GPIO, which the boot loader uses to decide how to boot (download or jump to FLASH), the SRAM and FLASH controllers, and possibly the USART if the program has been downloaded.

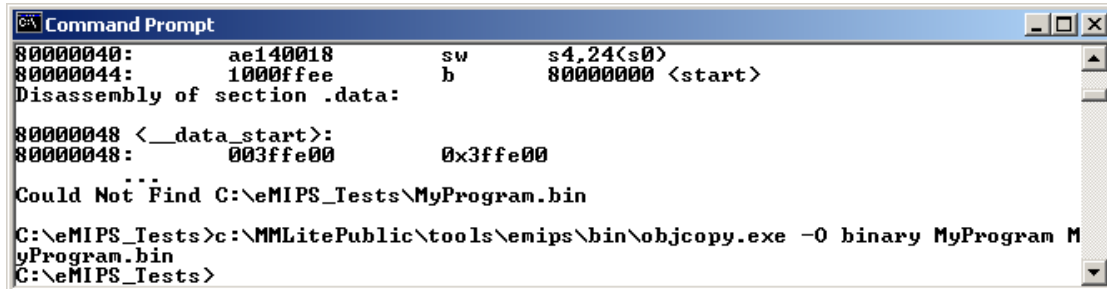
Standalone programs can be developed in assembly when necessary; examples are provided in the *tests* folder. Most examples in the *tests* folder are actually written in C, using the facilities provided in the *print.{h,s}* support file. Other useful header files are included in the RTOS source tree, in the *include\mips* folder. They include the *mips_asm.h* file that defines symbolic register names, macros for exported functions and debugging directives for assembly codes. The file *m140x.h* defines the register interfaces of the built-in peripherals and their preferred locations.

Two simple scripts are provided to help create standalone programs, using the GCC toolset. The script *compile4.cmd* is used to create a downloadable image starting from an assembly source file.

1. Start a CMD or Visual Studio command window.
2. Set the environment variable `MMLITE_SDK` to point to the location of the RTOS sources, augmented by the binaries of the GCC toolset (as described in section 2.3.1.4).
3. Run the script with your program name as argument, e.g. if your assembly source file is *MyProgram.s* invoke the script with “*compile4 MyProgram*”.



The script will compile and list the content of the downloadable image. The results of the script are two files. *MyProgram.bin* is the stripped image that can be downloaded via the boot loader (or written to the linear FLASH). The file *MyProgram* is in the ELF file format and can be given to other tools, such as the GDB debugger.



```

C:\ Command Prompt
80000040:      ae140018      sw      s4,24(s0)
80000044:      1000ffee      b       80000000 <start>
Disassembly of section .data:
80000048 <__data_start>:
80000048:      003ffe00      0x3ffe00

Could Not Find C:\eMIPS_Tests\MyProgram.bin

C:\eMIPS_Tests>c:\MMLitePublic\tools\emips\bin\objcopy.exe -O binary MyProgram M
yProgram.bin
C:\eMIPS_Tests>

```

The script *c_compile4.cmd* is used to create a downloadable image starting from a C source file, and an optional assembly source file.

1. Start a CMD or Visual Studio command window.
2. Set the environment variable `MMLITE_SDK` to point to the location of the RTOS sources, augmented by the binaries of the GCC toolset (as described in section 2.3.1.4).
3. Run the script with your program name as argument, e.g. if your C source file is *MyProgram.c* invoke the script with “*compile4 MyProgram*”.



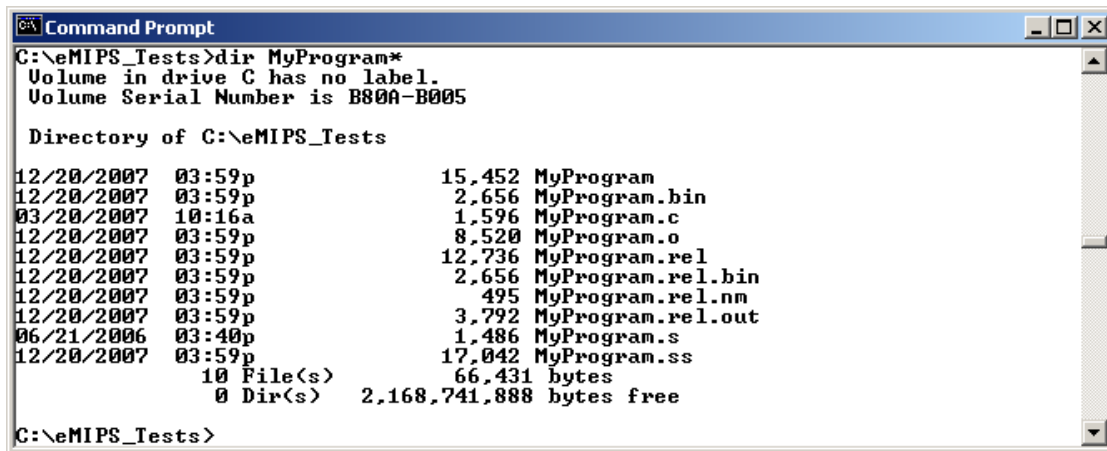
```

C:\ Command Prompt

C:\eMIPS_Tests>set MMLITE_SDK=c:\MMLitePublic
C:\eMIPS_Tests>c_compile4 MyProgram

```

The script will compile and list the content of the downloadable image. The results of the script are two files. *MyProgram.bin* is the stripped image that can be downloaded via the boot loader (or written to the linear FLASH). The file *MyProgram* is in the ELF file format and can be given to other tools, such as the GDB debugger.



```

C:\eMIPS_Tests>dir MyProgram*
Volume in drive C has no label.
Volume Serial Number is B80A-B005

Directory of C:\eMIPS_Tests

12/20/2007  03:59p                15,452  MyProgram
12/20/2007  03:59p                2,656  MyProgram.bin
03/20/2007  10:16a                1,596  MyProgram.c
12/20/2007  03:59p                8,520  MyProgram.o
12/20/2007  03:59p               12,736  MyProgram.rel
12/20/2007  03:59p                2,656  MyProgram.rel.bin
12/20/2007  03:59p                 495  MyProgram.rel.nm
12/20/2007  03:59p                3,792  MyProgram.rel.out
06/21/2006  03:40p                1,486  MyProgram.s
12/20/2007  03:59p               17,042  MyProgram.ss
               10 File(s)              66,431 bytes
               0 Dir(s)  2,168,741,888 bytes free

C:\eMIPS_Tests>

```

Additionally, the script will try to run the *bbfind* tool to create a patched executable image, using the definitions of Extended Instructions in the file *bbtools\patts bbw*. The results are in the *MyProgram.rel.** files. This step is optional and for illustration purposes, if not needed it can be commented out from the script.

4.2 Building and running a program under the RTOS

Programs for eMIPS under the Microsoft Invisible Computing RTOS are no different than other programs for the same RTOS on other platforms. Users familiar with other platforms, such as the ARM EB63 for instance, will notice the only change in the build procedures is in the specification of the *TARGETCPU* variable. This section is for users that are not familiar with the RTOS.

One way to create and build a simple program is to use the RTOS' *tests* folder. This folder contains a number of programs that can be used as examples for various functions. The *makefile* in this directory includes the optional file *private.mk* that can be used to add to the list of target programs built in this directory. Let us assume that your program will be in the file *MyProgram.c*, and that its content is as follows:

```

#include <stdio.h>

int main(int argc, char **argv)
{
    printf("Hi Mom!\n");
}

```

The procedure for building the program is as follows:

1. Start a CMD or Visual Studio command window.

- Set the environment variable `MMLITE_SDK` to point to the location of the RTOS sources, augmented by the binaries of the GCC toolset (as described in section 2.3.1.4).
- Move to the `tests` folder in there.



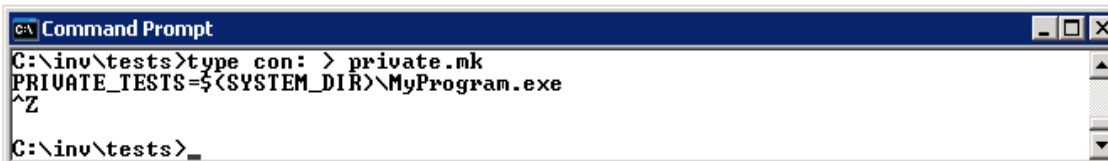
```

C:\>set MMLITE_SDK=c:\inv
C:\>cd c:\inv\tests
C:\inv\tests>_

```

- Create a file called `private.mk`, with the following one-line content:

PRIVATE_TESTS=\$(SYSTEM_DIR)\MyProgram.exe



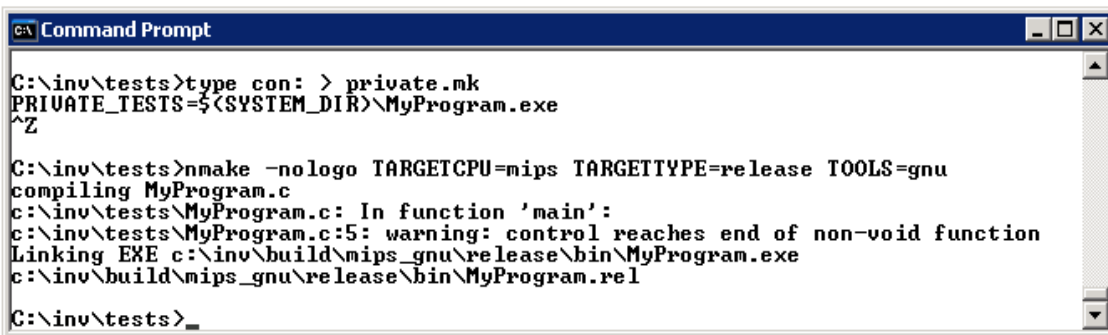
```

C:\inv\tests>type con: > private.mk
PRIVATE_TESTS=$(SYSTEM_DIR)\MyProgram.exe
^Z
C:\inv\tests>_

```

- Build the program with the following command line:

nmake -nologo TARGETCPU=mips TARGETTYPE=release TOOLS=gnu



```

C:\inv\tests>type con: > private.mk
PRIVATE_TESTS=$(SYSTEM_DIR)\MyProgram.exe
^Z

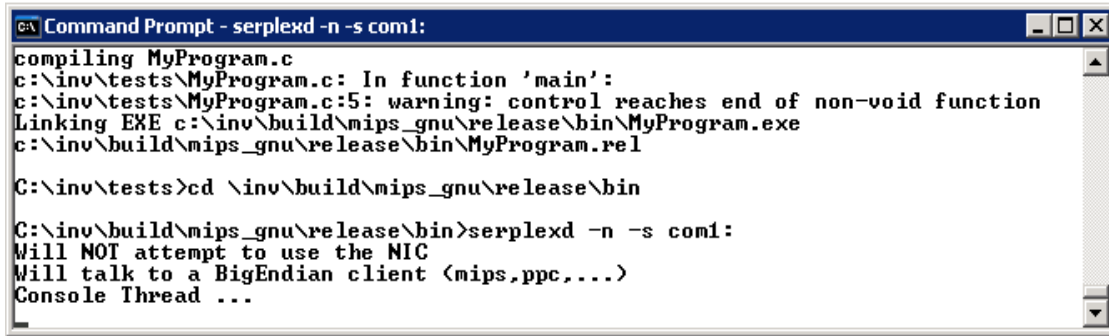
C:\inv\tests>nmake -nologo TARGETCPU=mips TARGETTYPE=release TOOLS=gnu
compiling MyProgram.c
c:\inv\tests\MyProgram.c: In function 'main':
c:\inv\tests\MyProgram.c:5: warning: control reaches end of non-void function
Linking EXE c:\inv\build\mips_gnu\release\bin\MyProgram.exe
c:\inv\build\mips_gnu\release\bin\MyProgram.rel
C:\inv\tests>_

```

The resulting file `MyProgram.exe` is left in the `mips_gnu\release\bin` folder under the *build* tree.

One simple way to execute your program is to install the RTOS in the ML401's linear FLASH, using the procedures described in section 3.5.1.3. Assuming the RTOS is installed and functional on the ML401 board, use the following procedure.

- Move to the build directory and start the SERPLEXD server to communicate with your board:



```

C:\inv\tests>cd \inv\build\mips_gnu\release\bin
C:\inv\build\mips_gnu\release\bin>serplexd -n -s com1:
Will NOT attempt to use the NIC
Will talk to a BigEndian client (mips,ppc,...)
Console Thread ...

```

7. The default file system is in FLASH, and it is visible under the “fs” folder. The remote file system is visible under the “srfs” folder. The RTOS only executes programs from the fs folder, so we need to rename the two. Type the following commands to the RTOS’ shell:

a. *In fs flash*

b. *In srfs fs*

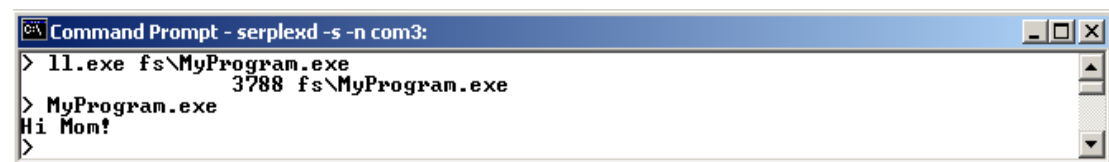


```

> ls
vtables
COB
com1
serplex6a
srfs
serplex6d
serplex6c
sysace
stdin
stdout
stderr
fs
flash
> ln fs flash
> ln srfs fs
> ls
vtables
COB
com1
serplex6a
srfs
serplex6d
serplex6c
sysace
stdin
stdout
stderr
fs
flash
>

```

8. Now you can invoke your program, using the full, case-sensitive file name:



```

> ll.exe fs\MyProgram.exe
3788 fs\MyProgram.exe
> MyProgram.exe
Hi Mom!
>

```

To make changes and re-run your program you might want to use two separate windows, one for building and one for running it. Repeat the steps 1-3 and 5 above to create a new window

and re-compile your program. Move back to the previous window and run your program again. Hint: the RTOS shell has a minimal command history buffer, you can use the Emacs-style ^P and ^N keys to navigate it.

4.3 Debugging with eBug

The technical report on eBug [107] describes how to build and use this tool. Section 7.1.2 of the report shows the procedure for debugging with the ML401 board, using a standalone program. Notice that a standalone program must enable the eBug Extension itself, there is no RTOS to do it. There is a configuration file in this release that corresponds to the one used at step 2 of the afore-mentioned procedure. It already contains eBug compiled-in and you can find it in the *eMIPS_Configurations.zip* archive file; the version with eight watchpoints is called *mipspl_fpga3_ebughw8_routed_full.bit*.

To debug under the RTOS you can use an identical procedure, but instead of downloading your program you will start it from the command shell. Again, your program will have to enable the eBug Extension by itself, because the RTOS is not aware of its presence.

A cleaner procedure is to create an SE image that contains your program and the eBug Extension, as described in section 4.5. In this case the RTOS will be aware of the Extension and it will enable it for you. Notice that eBug needs priority over the TISA, therefore the command line for *ace2se* is

```
ace2se -PhD MyProgram.se MyProgram.exe eBug_hw8.ace
```

The “-PhD” option instructs the program to set the properties field of the hardware image to the hexadecimal value 0xD. This value corresponds to the bits of Figure 12, in the following way:

SE_HW_PRIVILEGED	0x00000001
SE_HW_PERIPHERAL	0x00000002
SE_HW_PRIORITY_MASK	0x0000000c
SE_HW_HIGH_PRIORITY	0x0000000c

4.4 Rebuilding the boot loader

Source and binary for the boot loader are included in this release. Should it become necessary to rebuild the boot loader, use the following procedure.

1. Start a CMD or Visual Studio command window.
2. Set the environment variable MMLITE_SDK to point to the location of the RTOS sources, augmented by the binaries of the GCC toolset (as described in section 2.3.1.4).

3. Move to the Software\BramLoader folder in the release tree and compile the *bin2coe* utility, if you have not done so already:



```

C:\eMIPS_Tests>set MMLITE_SDK=c:\MMLitePublic
C:\eMIPS_Tests>cd Software\BramLoader
C:\eMIPS_Tests\Software\BramLoader>cl bin2coe.c
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 12.00.8168 for 80x86
Copyright (C) Microsoft Corp 1984-1998. All rights reserved.

bin2coe.c
Microsoft (R) Incremental Linker Version 6.00.8447
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

/out:bin2coe.exe
bin2coe.obj
C:\eMIPS_Tests\Software\BramLoader>

```

4. Use the local script compile4.cmd to rebuild the boot loader, type “compile4 bram2”:



```

/out:bin2coe.exe
bin2coe.obj
C:\eMIPS_Tests\Software\BramLoader>compile4 bram2

```

The script creates and disassembles a binary image of the loader, creates and dumps a file with the symbols for it, and in the last step it creates the coefficient file *bram2.coe* to be used to initialize the boot loader’s blockram (see step 18 in section 3.1.7).



```

00c00000 A _ehot
C:\eMIPS_Tests\Software\BramLoader>bin2coe bram2.bin bram2.coe swap
C:\eMIPS_Tests\Software\BramLoader>_

```

4.5 Building an SE image that contains an Extension

Extensions can be loaded into the Extension slot in a number of ways:

1. Using the Impact utility, which is part of the Xilinx ISE
2. Using the System-ACE via the board’s DIP switches
3. Using the RTOS’ test program *tsysace.exe*
4. Using the RTOS tool *ace2se* and the RTOS’ loader

In this section we describe how to use the *ace2se* tool to create Secure Executable, or SE images, to be loaded by the RTOS' system loader. The file format of an SE loadable image is depicted in Figure 1.

Software Image (e.g. an ELF file)	Extension Image (e.g. an ACE file)	Security Signature	SE Header
--------------------------------------	---------------------------------------	--------------------	-----------

Figure 1: SE File Format

The first portion of the SE file is the software executable code, in our case for the eMIPS processor. This is an un-modified ELF executable image, created using the tools and procedures described in sections 2.3.1 and 4.2. Because it is at the beginning of the file, existing tools for ELF images will work on an SE file too. The second portion of the file is the Extension configuration bit file. In our case this is an ACE file, generated following the procedure of section 3.4. The third and fourth portions of the file are created by the *ace2se* tool.

Either of the first two portions could be missing. Even a file with both portions missing is still a valid SE file, albeit a useless one. An SE file that contains only the Software Image will have the added advantage, over a simple ELF file for instance, of a format-independent security signature. Similarly for a file that only contains an Extension Image.

The RTOS' loader for eMIPS recognizes and loads both a simple ELF image and an SE image. If the SE image contains an Extension Image portion, this is loaded into the Extension slot first. The Extension is however not enabled, not until the Software Image has been loaded. If the file contains a Software Image portion, this is loaded in RAM and relocated. At this point a new thread is created, which enables the extension and executes the Software Image.

To create an SE file that contains an Extension you need the Extension's ACE file (say *MyExtension.ace*) and your software application's image file (*MyProgram.exe*).

1. Start a CMD or Visual Studio command window.
2. Set the environment variable MMLITE_SDK to point to the location of the RTOS sources, augmented by the binaries of the GCC toolset (as described in section 2.3.1.4).
3. Move to the folder that contains the files *MyExtension.ace* and *MyProgram.exe*

```

Command Prompt
E:\MMLitePublic\build\mips_gnu\release\bin>dir My*
Volume in drive E is data
Volume Serial Number is FC6C-420C

Directory of E:\MMLitePublic\build\mips_gnu\release\bin

11/19/2007  03:16p             128,423 MyExtension.ace
12/17/2007  03:06p              3,788 MyProgram.exe
12/17/2007  03:06p             38,653 MyProgram.map
12/17/2007  03:06p            272,594 MyProgram.rel
               4 File(s)          443,458 bytes
               0 Dir(s)  165,011,599,360 bytes free

E:\MMLitePublic\build\mips_gnu\release\bin>_

```

4. Type the following command:

`%MMLITE_SDK%\tools\bin\ace2se MyProgram.se MyProgram.exe MyExtension.ace`

```

Command Prompt
               0 Dir(s)  165,011,464,192 bytes free

E:\MMLitePublic\build\mips_gnu\release\bin>%MMLITE_SDK%\tools\bin\ace2se -PhD My
Program.se MyProgram.exe MyExtension.ace

E:\MMLitePublic\build\mips_gnu\release\bin>dir My*
Volume in drive E is data
Volume Serial Number is FC6C-420C

Directory of E:\MMLitePublic\build\mips_gnu\release\bin

11/19/2007  03:16 PM             128,423 MyExtension.ace
12/17/2007  03:06 PM              3,788 MyProgram.exe
12/17/2007  03:06 PM             38,653 MyProgram.map
12/17/2007  03:06 PM            272,594 MyProgram.rel
12/17/2007  04:31 PM            132,332 MyProgram.se
               5 File(s)          575,790 bytes
               0 Dir(s)  165,011,464,192 bytes free

E:\MMLitePublic\build\mips_gnu\release\bin>_

```

The resulting file is the SE image *MyProgram.se*. To omit an image from the SE file you can just give to *ace2se* the name of a non-existent file.

```

Command Prompt
E:\MMLitePublic\build\mips_gnu\release\bin>%MMLITE_SDK%\tools\bin\ace2se -PhD My
Program.se nothing MyExtension.ace
Could not open nothing for reading.
WARNING!! Could not read nothing. Using empty file.

E:\MMLitePublic\build\mips_gnu\release\bin>dir My*
Volume in drive E is data
Volume Serial Number is FC6C-420C

Directory of E:\MMLitePublic\build\mips_gnu\release\bin

11/19/2007  03:16 PM             128,423 MyExtension.ace
12/17/2007  03:06 PM              3,788 MyProgram.exe
12/17/2007  03:06 PM             38,653 MyProgram.map
12/17/2007  03:06 PM            272,594 MyProgram.rel
12/17/2007  04:37 PM            128,524 MyProgram.se
               5 File(s)          571,982 bytes
               0 Dir(s)  165,027,954,688 bytes free

E:\MMLitePublic\build\mips_gnu\release\bin>_

```

4.6 Using the BBTools to profile a program

The BBTools can be used in conjunction with the Giano simulator to obtain accurate and fine-grained profiling information about your program. The general idea is to first optimize the program for best performance, using profiling and any other tools available. Once the program runs at the best performance possible, use this procedure to identify the two-three basic blocks that are executed most frequently. Then create an Extension that executes those blocks in hardware and modify the original image inserting Extended Instructions that invoke the Extension (see section 4.7 for how to do this). Note that the number of times a block is executed is only one of the factors to take into account when selecting the blocks to optimize in hardware. Other factors include but are not limited to: possible payoffs from better memory access patterns, multi-block opportunities (e.g. small function inlining and other global optimizations), and programs that exhibit distinct phases during execution.

This procedure assumes that your program runs under the RTOS. You also need a build of Giano with basic block profiling support enabled (e.g. the conditional `BBS_SUPPORT` in the `mips_cpu.cpp` module). Another way is to just build a debug version of Giano and use that.

When the RTOS executes your program, for instance *MyProgram.exe*, it notifies the debugger of this. The simulator is able to intercept this notification and therefore is aware of your program being executed, and at which address it resides in memory. The simulator then looks for a file called *MyProgram.exe.bbs* in the current directory. If it is not found it will print a message to this effect. If it is found, it loads the information in that file and keeps track of instructions that enter a basic block. When there is a hit, the simulator updates the corresponding `DynamicReplicationCount` counter. When the program terminates, the RTOS sends a second notification. The simulator then writes back the updated counters in the *MyProgram.exe.bbs* file. At this point you can use the other BBTools to dump, sort and inspect the blocks with the highest `DynamicReplicationCount` values. Note that the simulator rewrites the BBS file, you will want to keep a copy of the original for multiple experiments, e.g. one with zeroes in the counters.

1. Start a CMD or Visual Studio command window.
2. Make sure the BBTools binaries are in your PATH environment variable and move to the directory where your *MyProgram.rel* image is located. Notice that you need the image before it is stripped, e.g. the REL file and not the EXE file.
3. Run the BBFIND tool. You might want to also create a file with the symbols from your program, in this case we called it *MyProgram.nm*:

```
bbfind -b MyProgram.exe.bbs -s MyProgram.nm MyProgram.rel
```

4. Start Giano using the *MI401.plx* platform configuration file.

5. Start the SERPLEXD console to talk to the simulation.
6. Run your program. The simulator should notify you that it found the BBS file for your program at load time, and that it rewrote it when the program terminated.
7. Sort the BBS file according to the DynamicReplicationCount metric:

```
bbsort -r -dynamic MyProgram.exe.bbs MyProgram.out.bbs
```

8. Dump the sorted file on a text file, for later inspection with a text editor:

```
bbdump MyProgram.out.bbs MyProgram.nm
```

You will need a disassembly of your program, to reconcile the basic blocks with the C code they came from. You can use the compiler's OBJDUMP utility for this, or pass an additional “-v 1” argument to BBFIND at step 3.

If you plan to use the M2V compiler you will want to isolate the individual basic blocks that you intend to optimize. You can use the BBSELECT tool for this, using the “-hash” option to select the block of your interest:

```
bbselect -hash:X.Y.W.Z MyProgram.out.bbs MyBlock.bbs
```

You can then use the BBMATCH tool to create a skeleton BBW file for your extension:

```
bbmatch -c ignore MyBlock.bbs > MyBlock.bbw
```

The BBW file is also needed if you plan to create Extended Instructions for your Extension, see section 4.7.

4.7 Adding Extended Instructions to an image

If you want to patch your program image with the Extended Instructions that invoke your Extension you will need a BBW file that defines them. The BBFIND tool will use the basic block patterns defined in the file and patch all occurrences of the blocks in the image. One BBW file can contain multiple blocks, and even blocks for multiple processor architectures. Once you have the BBW file (say MyBlocks.bbw), use the BBFIND tool to patch your image:

```
bbfind -m MyBlocks.bbw -b MyProgram.exe.bbs MyProgram.rel MyProgram.exe
```

You might have used the procedure at section 4.6 to create a number of separate BBS files, one per each block of interest. To generate the skeleton BBW file with all your blocks in it, you can first combine the BBS files into a single BBS file using the BBCAT tool, and then use the BBMATCH tool on the combined BBS file:

```
bbcat Block1.bbs Block2.bbs MyBlocks.bbs
```

```
bbcat MyBlocks.bbs Block3.bbs MyBlocks.bbs
```

repeat for all additional blocks

```
bbmatch -c ignore MyBlock.bbs > MyBlocks.bbw
```

Remember that you still need to select an opcode and an encoding for your Extended Instructions, before you attempt to patch images.

4.8 Generating Extensions using the M2V compiler

The technical report on M2V [106] describes the M2V compiler in details. Section 5 of the report shows how to invoke the tool. To be able to create an Extension with the M2V compiler you will need to create first a BBW file that describes it. You can do this manually, with a text editor, once you understand the format of the BBW file.

If your basic block can be found in an image you can use the BBTools instead. The first step is to create a basic block database file (a BBS file) from your image, using the BBFIND tool with the “-b” option. Next use the BBMATCH tool to create a BBW file from the BBS file, using the “-c” option. Then edit the BBW file to isolate the block you want and to select an encoding for your extended instruction. Finally, give the resulting file to M2V.

9. Start a CMD or Visual Studio command window.

10. Make sure the BBTools binaries are in your PATH environment variable and move to the directory where your *MyProgram.rel* image is located. Notice that you need the image before it is stripped, e.g. the REL file and not the EXE file.

11. Run the BBFIND tool:

```
bbfind -b MyProgram.exe.bbs MyProgram.rel
```

12. Run the BBMATCH tool:

```
bbmatch -c ignore MyProgram.exe.bbs > MyExtension.bbw
```

13. Edit the file *MyExtension.bbw* to isolate your basic block and to create an appropriate encoding for the Extended Instruction for it.

14. Invoke the M2V compiler:

```
m2v MyExtension.bbw MyExtension.v
```


DRAFT – DO NOT REPRODUCE

Please be aware that the first version of the M2V compiler included in this release has many and rather severe limitations.

5 Architecture of the eMIPS System

The eMIPS processor is a ‘dynamically extensible microprocessor’ because it is based on a new, extensible architecture. The architecture is extensible because it allows additional logic to interface and interact with the basic data path at all stages of the pipeline. The additional logic, which we term Extensions, can be loaded on-chip dynamically during execution by the processor itself. The architecture therefore possesses the unique ability to extend its own ISA at run-time.

Figure 2 presents a block diagram of the eMIPS processor organization. The pipeline stages, general purpose register file and memory interface match those of a ‘classic’ RISC CPU and are depicted in lighter color in the diagram. These pipeline stages constitute the Trusted ISA or TISA, the core portion of the architecture that is required for initial operation and to provide a base level of trust in the functioning of the processor. These blocks cannot be removed or disabled and must be present at startup of the system. These blocks constitute the fixed partition of the architecture and include all resources that are of a security sensitive nature, such as the system coprocessor. The TISA also includes all the facilities for self-extension, including instructions for loading, unloading, disabling and controlling the unallocated Extension blocks in the microprocessor. At a functional level the pipeline blocks operate similarly to the ‘classic’ CPU, except their interconnections with respect to each other and other blocks differs. Their implementations differ as well and this will be explained later.

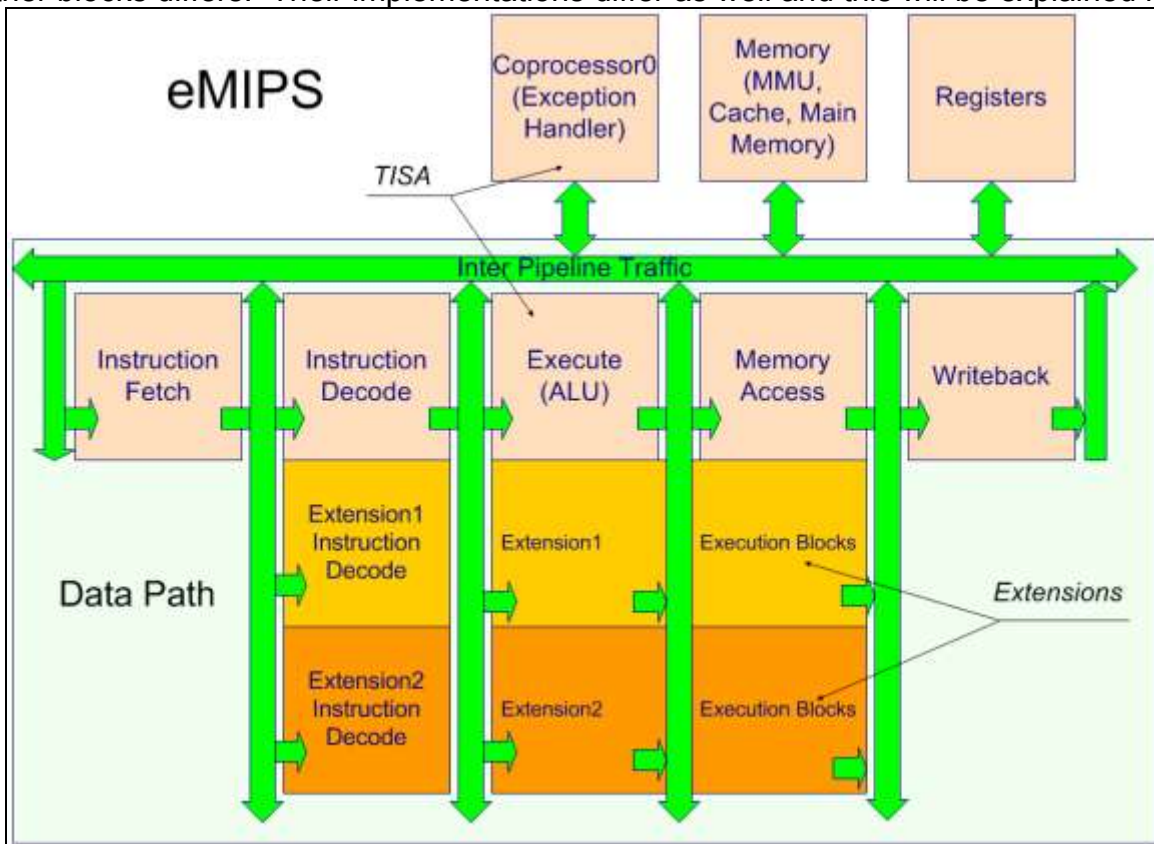


Figure 2: Block diagram of the eMIPS architecture.

Figure 2 shows two sets of blocks labeled “Extensions”. These Extensions distinguish the processor architecture from the established RISC architecture from which it is derived. Through the Extensions the processor overcomes two major shortcomings of the RISC architecture; inflexibility and inability to evolve with changing needs. Using the partial reconfiguration design flow described in Section 5.2 the processor is partitioned into fixed and reconfigurable regions. The TISA is included in the fixed region; the Extensions are included in the reconfigurable regions and are interconnected with the TISA by means of the bus macros described in Section 5.2. By implementing different Extensions for the reconfigurable regions, it becomes possible to adapt the functionality of the processor. The processor may apply these adaptations after deployment, dynamically while the applications continue executing.

Examples of possible Extensions include but are not limited to FPUs, Digital Signal Processors, or DSP, Encryption Coprocessors, Vector Processors and the application specific instructions. Using application execution profiling, engineers identify the Extended Instructions and implement them as hardware modules synthesized for the target device. More than one Extended Instruction might be included in a single Extension. A successful implementation of an Extended Instruction runs in fewer clock cycles than the original instruction sequence it replaces. If the instruction is executed a sufficient number of times, even a single clock cycle reduction in execution could significantly improve performance.

The diagram of Figure 2 depicts two Extension blocks, however the current implementation of the eMIPS system instantiates only one. Depending on space and other limitations imposed by the physical chip, additional Extension slots could be made available in the future.

5.1 Overview of Xilinx Virtex 4 and ML401 Board

The FPGA selected for the development and experimentation of eMIPS is the Xilinx Virtex 4 product line. The Virtex series rates among the most powerful FPGA devices in the market in terms of density, feature set and speeds. These FPGAs clock commonly at frequencies of 100 MHz but the specifications indicate they could operate at much higher frequencies, the specifications claim 500 MHz. These frequencies fall significantly short of modern ASIC frequencies approaching multiple gigahertz but the FPGAs continue to grow in speed with each new generation. The Virtex 4 high-end FPGAs come in three flavors denoted by LX, SX and FX. Each flavor includes a set of special features to allow developers to select an FPGA with the feature set that best fits their application domain. The Virtex 4 LX targets logic design applications. For this reason, the Virtex 4 LX provides the largest number of logical blocks for implementation. Given the floor planning requirements of partial reconfiguration, having more logic area to work with is preferred. Therefore, the implementation of the eMIPS processor targets the Xilinx ML401 Evaluation board with the Virtex LX25.[20] Use of the SX line is also an appealing prospect and, for instance, porting to the corresponding ML402 board should be an easy project.



Figure 3: Xilinx ML401 Evaluation Board with Virtex 4 LX25[20]

5.2 Overview of Xilinx Partial Reconfiguration

Xilinx has supported partial reconfiguration since its Virtex II chip [14] and that feature continues in the more modern Virtex 4 and Spartan III. The smallest reconfigurable unit of the FPGA configuration fabric is called the 'frame'. When partitioning the FPGA into different independently reconfigurable and static regions the boundaries between these regions must coincide with the boundaries of these 'frames'. Multiple frames may be grouped together into a single rectangular region. Regions cannot be smaller than a 'frame'. In the Virtex 4, a column of sixteen slices makes up the 'frame'. In this way, each column of the Virtex 4 contains multiple frames. In the case of the LX25, which has 192 rows of slices, each column contains twelve frames. This architecture provides the Virtex 4 the advantage of allowing for rectangular regions in the form of tiles on the FPGA configuration fabric as opposed to strictly columns as in the previous architectures.[5]

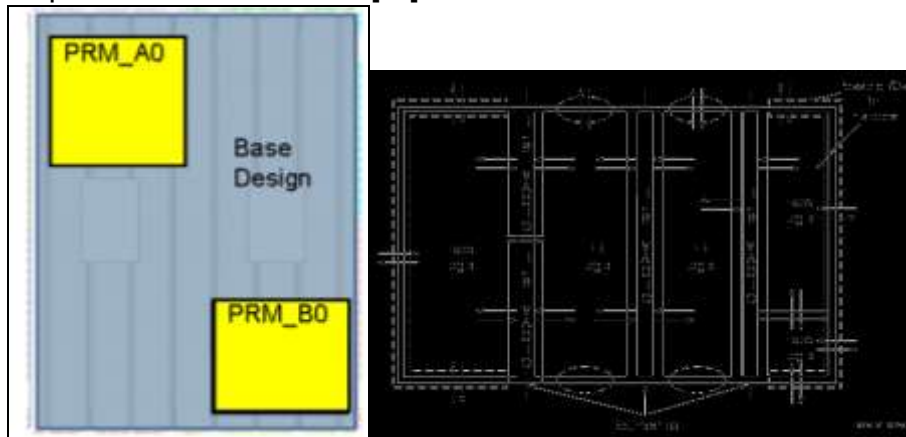


Figure 4: Examples of partitioning of a Reconfigurable FPGA design. [5][14]

Hardware designers must also consider the routing of signals crossing the boundaries of the various regions. Only the region containing the reconfigurable module changes when reconfiguration occurs. The remaining configuration fabric remains unchanged. Therefore any inconsistency from one configuration to the next will result in unpredictable results.

One potential inconsistency can occur when a signal crosses the module boundaries. Consider for instance the case of a signal that crosses the boundary and in one configuration

the signal routes through row four but the same signal is routed in row five in another configuration. When the system undergoes reconfiguration, the signal will not line up on the boundary where the reconfiguration occurred, therefore cutting the signal. To prevent this inconsistency we can restrict the routing of such signals to fixed locations along the region boundaries. This is done by routing the signals through a ‘bus macro’ or a hard pre-routed macro positioned on the module boundary and by forcing the router program to route the signal through a given location in each configuration. For the eMIPS processor, bus-macros are placed between the interfaces of the fixed instruction set logic and the dynamic Extensions.

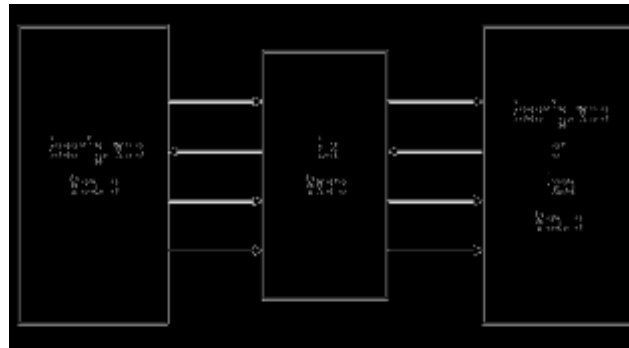


Figure 5: Logical connections of signals crossing a region boundary.[5]

Xilinx provides LUT based ‘bus macros’ for all their products including the partial reconfiguration feature. Xilinx ISE with the Partial Reconfiguration Tools takes the required routing consistency a step further by recording the routing of all fixed logic that passes through reconfigurable regions in a routing database. Xilinx ISE incorporates these routing patterns in the place-and-route phase of compilation, so that the reconfigurable regions will maintain consistency.

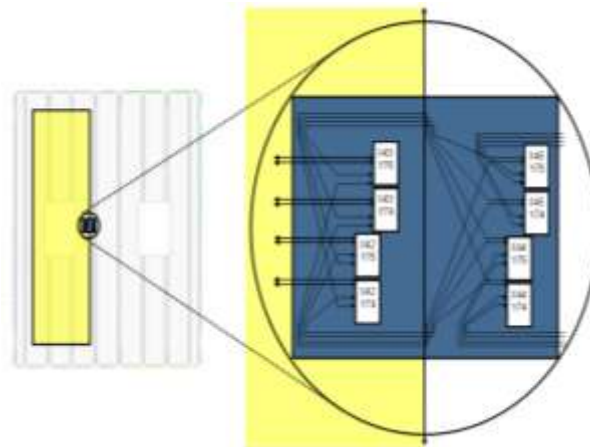


Figure 6: LUT Based Bus Macro.[5]

The partial reconfiguration design flow includes four phases as documented by Xilinx. These phases are Design Entry, Initial Budgeting, Active Module, and Final Assembly. Full details can be found in [5][13][14]. The following is a brief description of each phase:

1. *Design Entry* – This phase involves setting up the project by targeting the desired FPGA device, decide on design partitioning and performing some design planning. Before PlanAhead, this phase also included manually setting up the project directory structure.

PlanAhead now handles this in project setup. In large projects including multiple engineers, this phase is usually carried out by the team lead.

2. *Initial Budgeting* – In this phase the design engineers write the top level module and implementation constraint files. The constraint files include information such as pin assignments, area definitions, assignment of modules to areas and clocking constraints. The top level module defines the ports of the design and instantiates all second level modules and defines their interfaces to each other and to the system ports. This top level should be minimal in its contents. There should be as little logic in this layer as possible and contain only the modules that will be implemented at this layer. Any top level logic that is present goes through place and route and this data is written to the routing database for future use. In most cases, a team lead also carries out this phase.
3. *Active Module* – Design engineers execute this phase of the design flow for each module instantiated in the top level in parallel. The team lead assigns hardware designers to implement the different modules using the interface outlined in the top level written in the previous phase. In the case of reconfigurable modules, hardware designers implement two or more versions of this module. In some cases, designers write module level constraints into the implementation constraint file. The PlanAhead tool synthesizes each module independently of the rest of the design and performs place and route within the region designated for it while taking the contents of the routing database into account.
4. *Final Assembly* – This is the final phase of the design flow. In this phase, the team lead collects the module implementation of each module from the hardware designers and uses PlanAhead to integrate them together. The team lead creates a floor plan of the system for each possible configuration or combination of modules. Using these floor plans PlanAhead completes any additional place and route required and generates configurations files for the desired default configuration and other files for the reconfigurable regions that change dynamically.

For additional information and resources on the Xilinx Partial Reconfiguration Design Flow refer to following:

- <http://www.xilinx.com/support/prealounge/protected/index.htm>
- http://www.xilinx.com/ise/optional_prod/planahead.htm

5.3 Overview of Xilinx System ACE Configuration Solution

The System Advanced Configuration Environment, or System ACE [12], attempts to fill a niche for pre-engineered configuration solutions of multiple FPGA systems. The system applies to the eMIPS processor's need to control and modify its extensible configuration architecture. System ACE works through the interaction of four interfaces: JTAG to host PC, JTAG to FPGA, and Compact Flash & Control from Microprocessor or FPGA. Using the host JTAG interface a configuration file can be downloaded manually to the system and used to configure one or multiple FPGAs. This feature is excellent for debugging, it allows the developer to download test configuration and run code before including the new configuration in the system. When configuring the system from the host JTAG the System ACE reads the bits stream on the host interface and transfers it to the system JTAG chain it controls. After the configuration design completes and the system is ready for deployment, system controlled configuration can be performed via a microcontroller or FPGA control. In the case of a single FPGA system, like the eMIPS processor, the microcontroller interface can be integrated in the FPGA to allow it to control its own configuration. The Compact Flash is a portable, permanent storage device that stores the configuration files and inserts into a reader integrated with the

System ACE. Using the control interface the FPGA or microcontroller can initiate configuration of the system by selecting a configuration file stored in the Compact Flash that the System ACE drives on the system JTAG chain. Alternatively, the microcontroller could store the configuration data in RAM and send it to System ACE itself, using a different set of commands. The System ACE also provides an interface similar to the IDE disk interface commonly found on PCs to allow the microcontroller to read and write to the Compact Flash.

The System ACE controller interface provides a 3-bit configuration selection input to allow the controller to select one of eight potential configurations. Note that the Compact Flash can store more than eight configurations, as illustrated in Figure 7. The configurations are grouped into sets of no more than eight and placed in directories on the Compact Flash. In the root directory there exists a file called 'xilinx.sys'. This file tells System ACE which directory containing configuration files should be considered 'active'. The controller can only use configuration files from the 'active' directory. The 'xilinx.sys' file also assigns to each file the numerical designation zero through seven for the configuration selection. To change which set of configurations is considered active, one must change the assignment in the 'xilinx.sys' file. System software can do this dynamically using the IDE interface to the Compact Flash.[12]

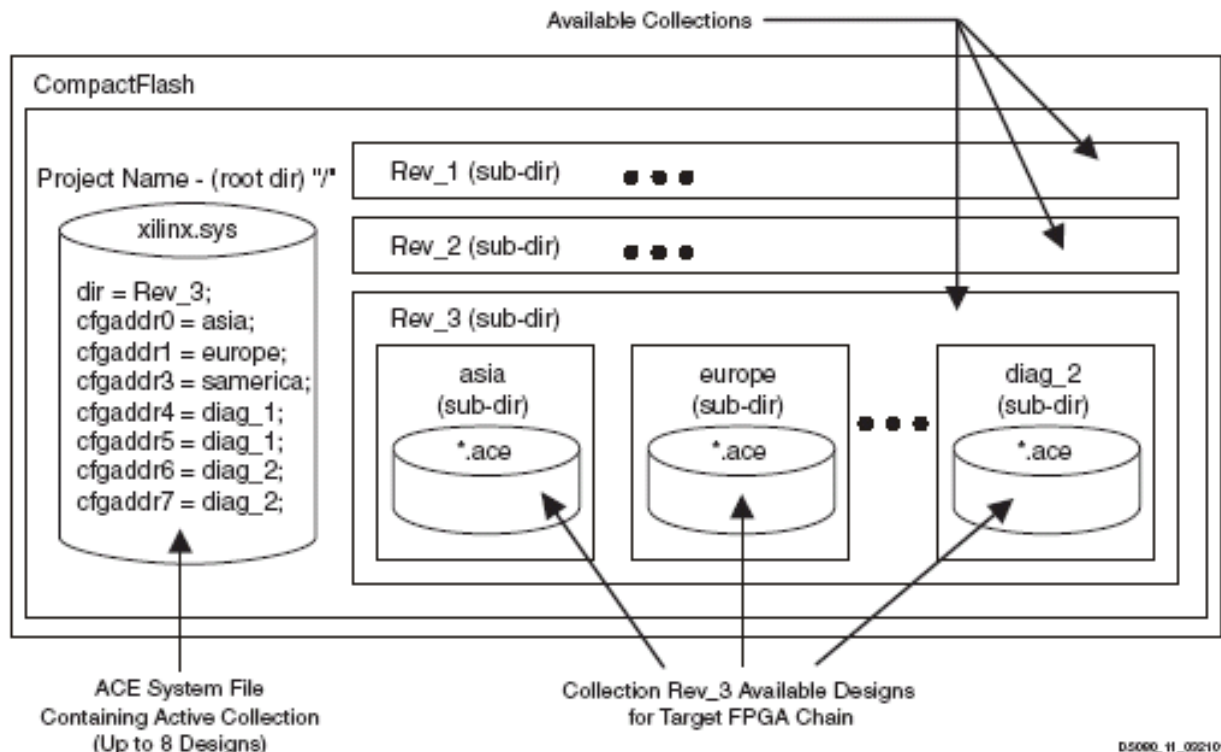


Figure 7: System ACE File structure.[12]

5.4 Overview of the RISC CPU Organization

The eMIPS microprocessor system is derived from the MIPS RISC architecture and thus includes most of the same components one would expect to find in a modern RISC microprocessor. The eMIPS microprocessor is a five stage pipeline including the following pipeline stages: Instruction Fetch (IF), Instruction Decode (ID), Instruction Execute (IE), Memory Access (MA) and Writeback (WB). The functions of these stages are as follows:

- *Instruction Fetch (IF)* – Update the program counter, or PC, and fetch the instruction located in memory at the address stored in the PC.
- *Instruction Decode (ID)* – Using wired logic and LUTs, decode the instruction passed from IF into control signals that control the remainder of the pipeline. Read any data required by the instruction from the general purpose register file. Test branch conditions and calculate the memory location of the next instruction to be executed.
- *Instruction Execute (IE)* – Using an Arithmetic Logic Unit, or ALU, and other special purpose logic perform operations on data based on the control signals passed from ID.
- *Memory Access (MA)* – In case of a load or store instruction, the output of IE is used as the memory location to be read from or written to. Otherwise, the output of IE is passed through.
- *Writeback (WB)* – In the event a register in the general purpose register file is modified by the instruction, the output of MA is written to the desired register.

In addition to this datapath the eMIPS microprocessor's Extension provides a parallel execution path to the one represented by the EX, MA, and WB stages.

To realize greater throughput at a higher frequency, some microprocessor implementations have utilized as many as eight pipeline stages. The deeper the pipeline is the greater the overhead in the event of a branch event, hazards and exceptions on execution. To further offset this overhead, microprocessor designers have developed a variety of features, including branch predictors and speculated execution. These features have highly complex implementations and exist beyond the current scope of the eMIPS project. For this reason, the architecture of the eMIPS microprocessor omits these features and has just the basic five pipeline stages.

5.5 eMIPS System Components

The following describes the different system modules that make up the eMIPS CPU system implementation. Many of these will be recognizable by those familiar with a RISC CPU organization. Other modules are new additions necessary to support eMIPS' unique features.

5.5.1 Top Level Module

MIPSPL_FPGA (eMIPS)

MIPSPL_FPGA is the top level module of the eMIPS microprocessor system design. All other components are sub-modules at some level in the design tree. All input/output pins are defined here as well as the partitioning of the fixed portion of the design, the TISA, from the reconfigurable Extensions.

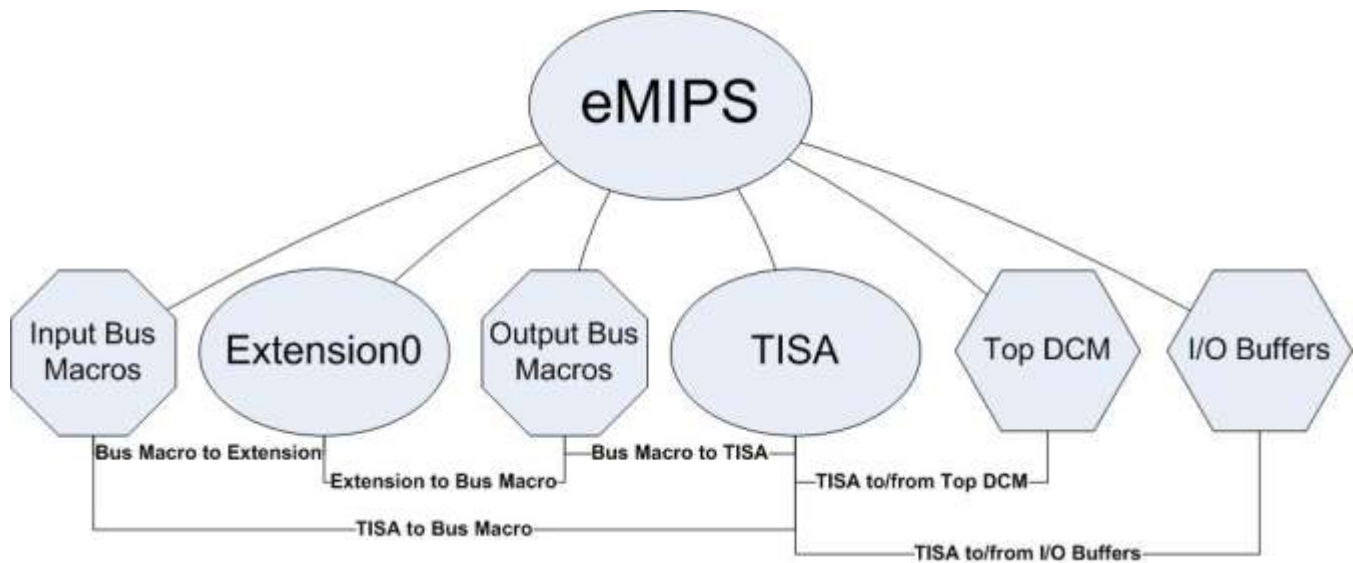


Figure 8: Design Hierarchy of the Top Level Module

5.5.2 Clocking and IO Modules

TOP_DCM (sys)

TOP_DCM is responsible for all the global clocking resources of the eMIPS microprocessor system. This module contains a Digital Clock Manager, or DCM, in series with a PMCD. The DCM takes the source clock from a clock pin of the FPGA as input. The DCM outputs a clock of the same frequency and one equal to that source clock divided by ten. In the case of a 100 MHz source clock, this produces 100 MHz and 10 MHz clocks respectively. These clocks are inputted to the PMCD where their edges are aligned. The PMCD also produces additional clocks based on the source frequency input. The PMCD produces clocks with the source frequency divided by two, four, and eight, all with their edges aligned with the source frequency clock and the ten-divided clock. The output of the PMCD that is of the source frequency is inputted to the DCM as an internal feedback. Using the feedback, the DCM uses phase shifting to minimize the slew of the clock. A second DCM is used in a similar way to minimize the slew of the clock outputted to the on-board SRAM of the ML401 board. The module includes several global clock buffers (BUFG) to route the clocks throughout the FPGA fabric.

IOBUF4 (mdatap_io)

IOBUF4 is a set of four IOBUF primitives wired together to form a four bit bidirectional bus. This is for implementing bidirectional I/Os for the external chip interfaces. One instance is used for the parity bits of the SRAM bus.

IOBUF16 (pdata_io)

IOBUF16 is a set of sixteen IOBUF primitives wired together to form a sixteen bit bidirectional bus. This is for implementing bidirectional I/Os for the external chip interfaces. One instance is used for the data bits of the System Ace bus.

IOBUF32 (gpio_io, mdata_io)

IOBUF32 is a set of thirty-two IOBUF primitives wired together to form a thirty-two bit bidirectional bus. This is for implementing bidirectional I/Os for the external chip interfaces. One instance is used for the data bits of the shared SRAM and FLASH bus. Another one is used for the GPIO.

5.5.3 Trusted ISA (TISA), Static Design Region

TISA (mips)

TISA encapsulates all the modules in the eMIPS microprocessor system that makes up the Trusted ISA. These modules include the pipeline stages and the supporting logic as well as the memory subsystem, including peripheral drivers and busses.

CLOCKMASTER (clkmas)

The CLOCKMASTER is responsible for synchronizing the eMIPS microprocessor pipeline. The CLOCKMASTER generates the synchronizing pipeline clock used for piping instructions through the pipeline. The CLOCKMASTER uses some logic to allow all pipeline stages to complete their work before continuing. The CLOCKMASTER is also responsible for announcing the system reset and generating a soft reset in response to a signal from the system coprocessor zero.

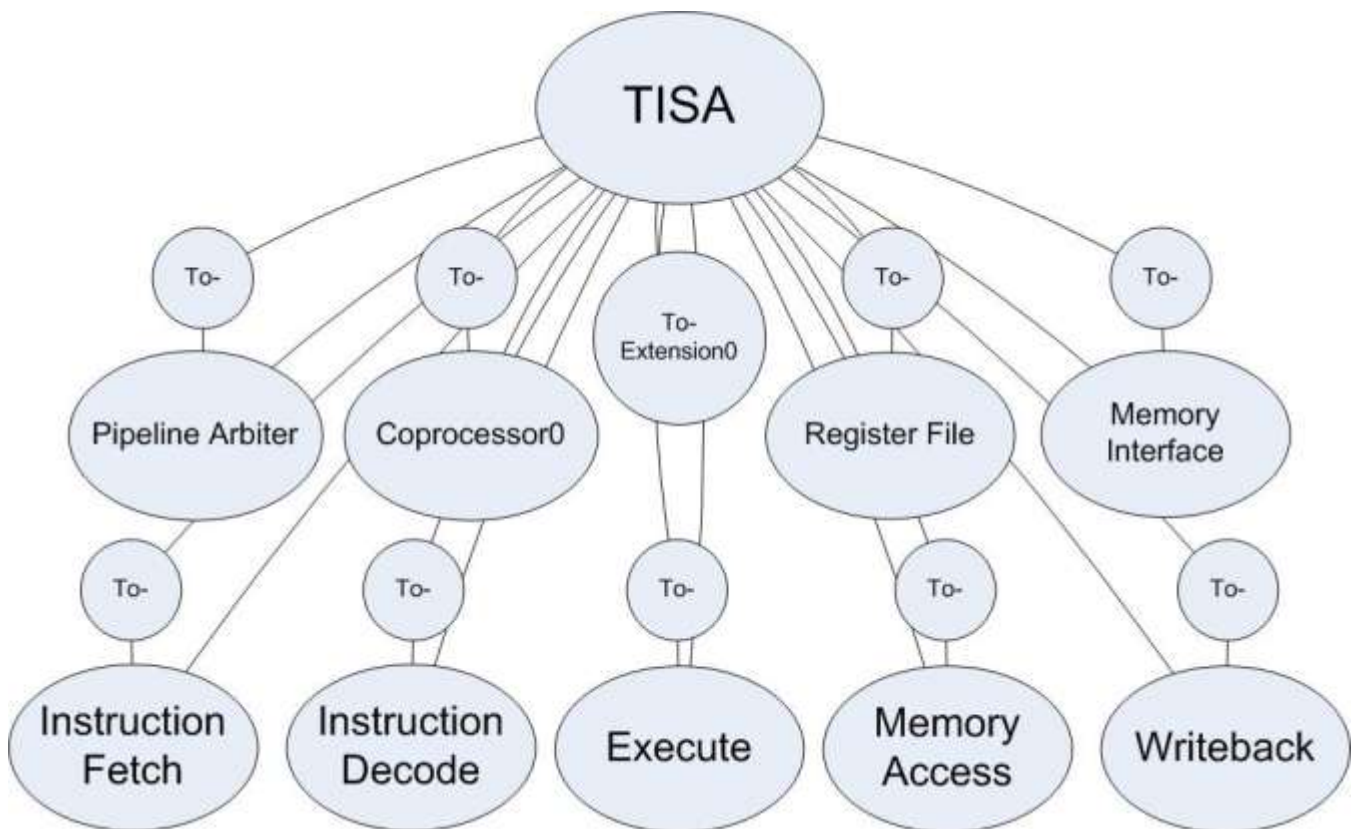


Figure 9: Design Hierarchy of the Trusted ISA

TOPA (to_pa)

The TOPA module gathers all the signals required by the pipeline arbiter and routes them to that region of the design.

PIPELINE_ARBITER (pa)

PIPELINE_ARBITER listens to the instruction decode blocks of the TISA and of the Extensions to determine which, if any, of the available paths are able to decode the instruction previously fetched. The PIPELINE_ARBITER is configured using register sixteen of the system coprocessor zero, the Extension Control register. The Extension Control register is partitioned into four eight-bit sets that each provides configuration information for a single Extension slot. The least significant bit of the set is an active high enable bit allowing software to enable or disable an instantiated Extension. The second bit is the load bit which rises when an Extension slot is filled and remains high until it is unallocated. The third bit is the trap enable bit. When this bit is high and a loaded Extension is disabled the PIPELINE will throw a trap on that Extension's instructions. Otherwise, if the trap bit is low, the Extended instructions associated with that Extension will be treated as NOPs. Following the trap bit is a two bit priority assignment used for arbitrating conflicts between the Extensions and the TISA. In the case that neither the TISA nor any Extension recognizes an instruction the PIPELINE_ARBITER signals a reserved instruction exception or trap. After arbitration is complete the PIPELINE_ARBITER enables the winning data path.

TOIF (to_if)

The TOIF module gathers all signals required by the instruction fetch and routes them to that region of the design. This module also multiplexes signals coming from the extension region.

INSTRUCTION_FETCH (inf)

INSTRUCTION_FETCH is the first stage in the eMIPS microprocessor pipeline. INSTRUCTION_FETCH maintains the current program counter, or PC, and updates it on each positive edge of the pipeline clock. The INSTRUCTION_FETCH dispatches memory read request to the memory interface to fetch instructions from memory. When the instruction returns from the memory interface the INSTRUCTION_FETCH forwards that instruction to the instruction decode phase.

TOID (to_id)

The TOID module gathers all signals required by the instruction decode and routes them to that region of the design. This module also multiplexes signals coming from the extension region.

INSTRUCTION_DECODE

INSTRUCTION_DECODE is the second stage in the eMIPS microprocessor pipeline. INSTRUCTION_DECODE attempts to decode each instruction that is forwarded to it from instruction fetch, in parallel with the loaded Extensions. In the first stage of the instruction decode, the INSTRUCTION_DECODE and the Extension ID evaluate the instruction and if the instruction is recognized by a decoder, that decoder will lower its RI (Recognized Instruction) signal to show it has recognized the instruction. Then the decode blocks wait to be enabled by

the pipeline arbiter. If the INSTRUCTION_DECODE is enabled, it finishes decoding the instruction into control signals for the remaining stages in the pipeline and forwards them to the execution phase. Otherwise, the INSTRUCTION_DECODE forwards the signals that produce a NOP in the pipeline.

TOEX (to_ex)

The TOEX module gathers all signals required by the execute phase and routes them to that region of the design. This module also multiplexes signals coming from the extension region.

EXECUTE (ex)

EXECUTE is the third stage in the eMIPS microprocessor pipeline. EXECUTE is passed control signals and operand data from the instruction decode phase of the pipeline. Using the control signals EXECUTE performs operations on the operand data using its arithmetic logic unit, or ALU, and other logic including a multiplier, divider, shifters and muxes. By the end of this phase, the Execute has produced a result that is either a data result that will be written to the register file, or an effective address for a memory transaction.

TOMA (to_ma)

The TOMA module gathers all signals required by the memory access phase and routes them to that region of the design. This module also multiplexes signals coming from the extension region. This multiplexing is also used when an instruction in the Extension slot intends to reenter the TISA execution path at the memory access phase.

MEMORY_ACCESS (ma)

MEMORY_ACCESS is the fourth stage in the eMIPS microprocessor pipeline. MEMORY_ACCESS is passed control signals and data from the execution phase of the pipeline. In the case of a memory related instruction, MEMORY_ACCESS uses the result from the execution phase as the effective address of a memory transaction. In the case of a write, the MEMORY_ACCESS dispatches a request to write the data passed from the execution phase into the effective address location. For a read, the MEMORY_ACCESS dispatches a read request. When the read request returns, MEMORY_ACCESS then passes the data returned to the writeback phase of the pipeline, to be written to the destination register. For non memory related instructions, the MEMORY_ACCESS passes the signals through to the writeback phase of execution.

TOWB (to_wb)

The TOWB module gathers all signals required by the writeback phase and routes them to that region of the design. This module also multiplexes signals coming from the Extension region. This multiplexing is also used when an instruction in the extension intends to reenter the TISA execution path at the writeback phase.

WRITE_BACK (wb)

WRITE_BACK is the fifth and final stage in the eMIPS microprocessor pipeline. WRITE_BACK is passed control signals and data from the memory access phase of the pipeline. In most cases, WRITE_BACK will be given data and a destination register number

where that data is to be written. The WRITE_BACK will then dispatch a write request to the intended general purpose register file. In other cases, such as a memory write, the WRITE_BACK module does nothing. All instructions are considered to have been committed after they finish the WRITE_BACK.

TOEXT (to_ext)

The TOEXT module gathers all signals required by the Extension interface and routes them to that region of the design. Section 6 describes this interface in details.

TODF (to_df)

The TODF module gathers all signals required by the data forward unit and routes them to that region of the design.

DATAFORWARD (df)

DATAFORWARD is a module found in most RISC pipelines and used to deal with data hazards. DATAFORWARD monitors the source and destination register numbers of the instructions in each of the eMIPS pipeline stages. Using these, the logic DATAFORWARD identifies data dependencies in the pipeline and controls a pair of muxes in the instruction decode and execution phases of the pipeline to forward intermediate data from the memory access and writeback phases as needed.

TOHZ (to_hz)

The TOHZ module gathers all signals required by the hazard unit and routes them to that region of the design.

HAZARD (hz)

HAZARD is a module found in most RISC pipelines and used to deal with control and data hazards. HAZARD monitors the control signals of the pipeline to identify control and data hazards. If the logic in HAZARD identifies a hazard condition, it stalls the pipeline at the instruction decode phase of the pipeline until the hazard condition has been alleviated. Potential hazards include, requiring data from a memory read for a branch test or reading from a multiply unit data register before it is finished with its current operation.

TORG (to_rg)

The TORG module gathers all signals required by the general purpose register file and routes them to that region of the design. This module also multiplexes signals coming from the Extension region.

REGISTERFILE (regs)

REGISTERFILE is a collection of thirty-two general purpose registers that are used to store operand data for the instructions executed on the eMIPS microprocessor. The standard MIPS microprocessor register file has two read ports for reading operand data in parallel and a single write port. Due to the increased data requirements of Extensions, the REGISTERFILE for eMIPS has been implemented with four read ports and two write ports for increased throughput. The current implementation experiences a maximum read latency of 40 ns and a maximum write latency of 70 ns from the Extension.

TOCP0 (to_cp0)

The TOCP0 module gathers all signals required by the system coprocessor and routes them to that region of the design. This module also multiplexes signals coming from the extension region.

COPROCESSOR0 (cp0)

COPROCESSOR0 is the system coprocessor and the exception handler included in the standard MIPS microprocessor architecture. COPROCESSOR0 controls the CPU state of the eMIPS microprocessor and handles incoming exceptions and interrupts. The COPROCESSOR allows for software handling of exceptions and interrupts and recovery through recording of the CPU context in these events for software processing.

TOMEM (to_mem)

The TOMEM module gathers all signals required by the memory interface and routes them to that region of the design. This module also multiplexes signals coming from the Extension region.

LOCK (lk)

The LOCK module detects the insertion of Extension peripherals. LOCK presents a serial interface to a key inside the Extension peripheral, using the PRESENT output signal of the Extension interface. After the Extension peripheral is loaded into the slot and the clock is activated the Extension peripheral begins sending a bit stream called the key value. The LOCK module compares this bit stream to the system key and if it matches, LOCK asserts the PRESENT bit for that Extension slot in the extension controller. When this occurs, the peripheral discovery and configuration process begins in system software. If there is no match the PRESENT bit is not asserted. The key inside of the Extension must continue sending the key bit stream until its LD bit is asserted. The LD is asserted the PRESENT output must be held high until the Extension peripheral is unloaded. When this occurs, LOCK lowers the PRESENT bit in the Extension controller and the peripheral removal process reclaims the system resources from the Extension peripheral.

5.5.4 Memory Subsystem Modules

MEMORY_INTERFACE (mem)

MEMORY_INTERFACE is the top level interface between the memory subsystem and the eMIPS microprocessor pipeline. Like in the standard MIPS, the interface exposes two ports: one for instruction fetching and one for memory accesses. Because the system has actually only a single bus to memory the MEMORY_INTERFACE is broken into three parts.

MEMORY_ARBITER (ma)

MEMORY_ARBITER is the part of the memory interface that latches the incoming memory requests, serializes them, and sends them to the memory bus. The MEMORY_ARBITER gives higher priority to the instruction read memory transactions over any data transaction.

MEMORY_BUS_FRONT (mbf)

MEMORY_BUS_FRONT receives memory requests from the memory arbiter one at a time, and generates the control signals to carry out the memory request on the memory controller.

MEMORY_CONTROLLER (mc)

MEMORY_CONTROLLER combines all the on-chip peripherals and the peripheral interfaces together on a single bus. Each peripheral listens to the address bus for the address ranges assigned to it. If the peripheral recognizes an incoming address, it latches the control, address and data when the start signal is raised. The peripheral acknowledges the request by lowering its *done* signal. When the memory transaction is complete the peripheral raises the *done* signal. In the case of a memory read, the data is valid with *done* is raised.

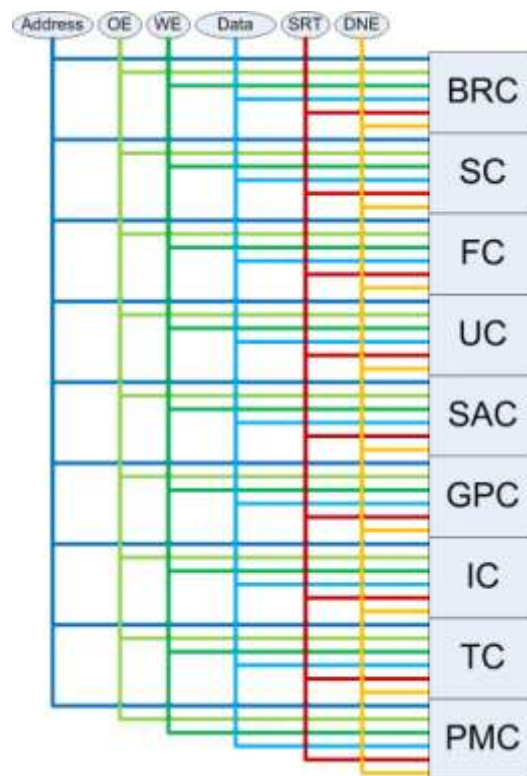
5.5.5 On-Chip Peripherals

Figure 10: eMIPS Memory Bus

BLOCKRAM_CONTROLLER (brc)

BLOCKRAM_CONTROLLER interfaces the TISA to a small on-chip memory called a BlockRAM. This BlockRAM contains the bootloader program, which is responsible for initializing the eMIPS CPU at start up and after reset. The BlockRAM also contains the Peripheral Mapping Table, or PMT, which represents the allocation of the eMIPS memory space amongst the peripherals currently present in the system. The table is dynamically maintained by system software and modified when loadable peripherals are loaded/unloaded.

in the Extension slot. The bootloader builds the PMT through a discovery process each time the eMIPS CPU is initialized.

SRAM_CONTROLLER (sc)

SRAM_CONTROLLER interfaces the TISA to the ZBT SRAM available on the Xilinx ML401 board. The application software running the eMIPS CPU is usually copied here from either Flash or the USART and then executed.

FLASH_CONTROLLER (fc)

FLASH_CONTROLLER interfaces the TISA to the parallel Flash Memories on the Xilinx ML401 Board. It is possible to store in Flash a binary image of an application for the eMIPS system to run at start up. In the case of our experiments this has been a embedded operating system called Microsoft Invisible Computing (see section 2.3).

USART_CONTROLLER (uc)

USART_CONTROLLER interfaces the TISA to a fully programmable, universal serial asynchronous receiver/transmitter. The USART is used to download programs and to communicate with the command console on the host PC.

SYSACE_CONTROLLER (sac)

SYSACE_CONTROLLER interfaces the TISA to the System ACE Configuration Solution chipset available on the Xilinx ML401 Board. The System ACE provides an IDE like disk interface to the Compact Flash card available on the Xilinx ML401. The System ACE acts as a JTAG programmer to allow for dynamic reconfiguration of the eMIPS system.

GPIO_CONTROLLER (gpc)

GPIO_CONTROLLER interfaces the TISA to the switches, buttons, LEDs, and pins of the Xilinx ML401 Board.

INTERRUPT_CONTROLLER (ic)

INTERRUPT_CONTROLLER supports programmable interrupts for the eMIPS system.

TIMER_CONTROLLER (tc)

TIMER_CONTROLLER interfaces two 64-bit counters to the TISA. One of the counters is a 64-bit down counter and the other is a 64-bit free counter.

POWER_MANAGEMENT_CONTROLLER (pmc)

POWER_MANAGEMENT_CONTROLLER is a stub module for adding power management in a later release.

EXTENSION_CONTROLLER (ec)

EXTENSION_CONTROLLER acts as a proxy to talk to the Extension peripheral's Base Address Translation registers, or BATs during the configuration phase after an Extension peripheral is loaded and discovered. Through the EXTENSION_CONTROLLER's interface the peripheral's address space, interrupt and privilege needs are determined by system software. Then using this same interface the peripheral is assigned its address space and changed to a

running state. After the system software has configured the Extension peripheral using this interface, the Extension peripheral is allowed to carry out transactions on the memory bus.

5.5.6 Extensions, Reconfigurable Design Region

EXTENSION0 (ext0)

EXTENSION0 is an instantiation of an eMIPS Extension. This serves as a wrapper module to define the Extension interface. It is also used if a user wants to insert an Extension into the design for a static build. In this case the user Extension is instantiated inside of this wrapper and connected to the appropriate signals. The EXTENSION0 includes interfaces to the pipeline stages, register file, and memory.

6 The eMIPS Extension Interface

The defining feature of the eMIPS system is its Extension interface, and the ability to dynamically reconfigure the contents of the Extension slots during runtime. Users of the eMIPS system may utilize the example extensions provided with this release to accelerate their programs and extend the eMIPS basic functions. This section describes how to fully take advantage of the eMIPS capabilities by letting users implement their own extensions. Four topics are addressed: software control of the Extension slots, HDL coding of the interfaces, interface protocols and design floor planning.

6.1 Extension Interface Control

The eMIPS Extension interface is controlled by the TISA through the system coprocessor. The system coprocessor of the eMIPS system has been extended to include a new register to control the Extension interface. This register is register 16 (sixteen) of the system coprocessor register file. We will refer to it as the Extension Control Register.

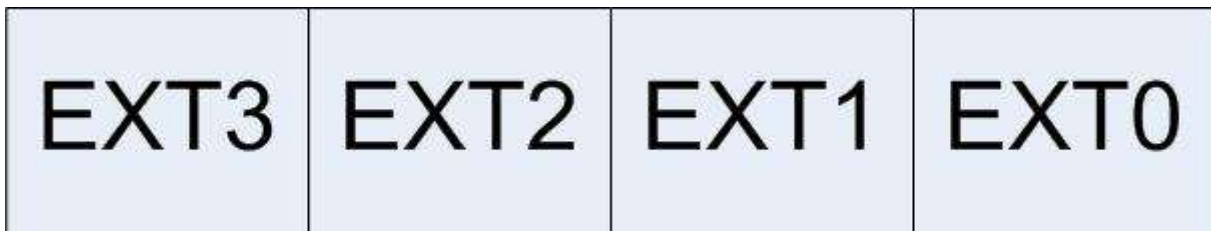


Figure 11: Extension Control Register (CP0 register 16)

The Extension Control Register is a read/write register that is accessible through the standard MIPS coprocessor interface instructions, namely MTC0 and MFC0. The Extension Control Register is divided into four eight-bit slices. Each slice controls the interface to a single Extension slot, as indicated in Figure 11. Since the eMIPS System in this release only has a single Extension slot, only the first eight bits of this register are relevant. Figure 12 provides the layout of the Extension Control Register bits for each slice.



Figure 12: Per Extension Slice of the Extension Control Register

The load bit, or LD, indicates that an image has been loaded in the Extension Slot and is ready for use. When this bit is low, all communication with the Extension slot is switched off.

The Extension slot cannot detect any signals from the TISA and cannot drive any signals either. If the LD is high, the instruction decode bus becomes visible to the Extension and it may attempt to request to execute the instructions that it recognizes. However, these requests will be ignored unless the enable bit, or EN, is high. The transition of the LD bit from high to low also asserts the reset signal to the Extension for a single PCLK cycle, unless the Extension is a peripheral.

When the EN is low, all communication with the Extension slot is switched off, with the exception of the instruction decode bus. This is to allow the Extension to still communicate which instructions belong to it. If those instructions are not otherwise recognized by the TISA the processor may trap with a Reserved Instruction exception. If the trap enable bit, or TR, is low the processor will not trap and treat the instruction as a NOP. If the TR is high, the processor will trap. Raising the EN bit activates all of the Extension's interfaces. This allows the Extension to compete with the TISA for instruction execution.

It is possible that an instruction is recognized by multiple Extensions, or both by an Extension and by the TISA. To deal with these cases, each Extension slot is assigned a priority, defined by the two-bit field PR. In the case of a conflict, the Extension with the highest priority will win the arbitration process. The TISA has a fixed priority of two (2'b10).

The CLK bit is similar in function to the LD bit in how it effects the Extension interfaces, but it has additional uses. Like the LD bit, if the CLK bit is low all communication between the Extension and TISA is disabled. Additionally, when the CLK bit is low the reset signal is asserted, and a few clock cycles later the clock to the Extension is disabled. It is expected that turning off the clock to an Extension will save power and/or allow the system to shut down malfunctioning Extensions before unloading them. When the CLK bit is raised the reset signal is deasserted and the clock to the Extension resumes. If the LD bit is still low the communication to the TISA continues to be latched off except for the PRESENT signal from the Extension, which is used for discovery during the Extension peripheral configuration. The PER bit is a flag used to indicate that an Extension is a peripheral module in addition to, or in place of, an execution path. With the eMIPS system it is possible to realize Extensions that can be interfaced via memory transactions and occupy address space in the memory map, just like the built-in peripheral devices. Examples of these modules are provided in the release and include a timer and a USART (see Appendix A). When the PER bit is high the memory bus is exposed to the Extension to allow it to respond to read and write transactions that match its assigned address space. By setting the PER bit low the connections between the bus and the Extension are latched off.

The PRIV bit is a flag used to indicate the level of trust the operating system places on the Extension image that is occupying this slot. By default the system does not trust the Extension and the PRIV is low. In this state all interactions with the TISA are controlled by the pipeline arbiter and the Extension can do nothing without first requesting permission from the pipeline arbiter. If the PRIV bit is set, this indicates that the system has full trust in the image that is occupying that slot. Currently, this allows the monitoring ports in the Extension interface

to be opened to allow the Extension to monitor memory and register traffic. Other applications of the level of trust are planned for future releases.

6.2 HDL Coding of Interfaces

A good place to start the extension design is the HDL code for the extensions that are provided in this release. The interface ports and their definitions can be found in the ‘extension0_black_box.v’ file provided in the ‘Sources’ package. When looking at the ports in the module definition it is noticeable that many ports have different uses at different times during the extension execution. This was done to reduce the number of signals required to cross the reconfigurable boundary in the design and to reduce the number of bus macros that must be placed to cross the boundary.

The recommended way to implement an Extension is to make a copy of the file ‘extension0_black_box.v’ that contains the Extension0 wrapper and to rename it appropriately. Then an instance of the new extension is instantiated within this wrapper. This approach has the additional benefit of minimizing code changes to the base system, e.g. in case the user extension is used in a static build. An example of this scheme is provided in the release in the following files:

<i>Wrapper file</i>	<i>Implementation file</i>
extension0_mmldiv64.v	mml_div64.v

6.3 Interface Protocols

This section describes the Extension interface protocols that might be required for an Extension to interact with the TISA, the eMIPS standard pipeline. Simple Extensions will not need all of these interfaces. More complex Extensions can utilize several interfaces to execute their functions. These include interfaces to the register file, memory and pipeline stages.

6.3.1 Instruction Decode/Arbitration Protocol

This interface to the pipeline is likely to be included in most every extension a user will ever write. In order for an extension to access the pipeline resources and to interact with the pipeline, it must request and obtain access to these resources. This entails going through an arbitration process with potentially competing Extensions, and possibly the TISA. The most common example is the case of an extension recognizing an instruction and requesting to take over execution of that the instruction. Figure 13 illustrates the interaction between the pipeline arbiter and the Extension during the instruction decode phase. The Extension is requesting to execute the instruction and wins arbitration.

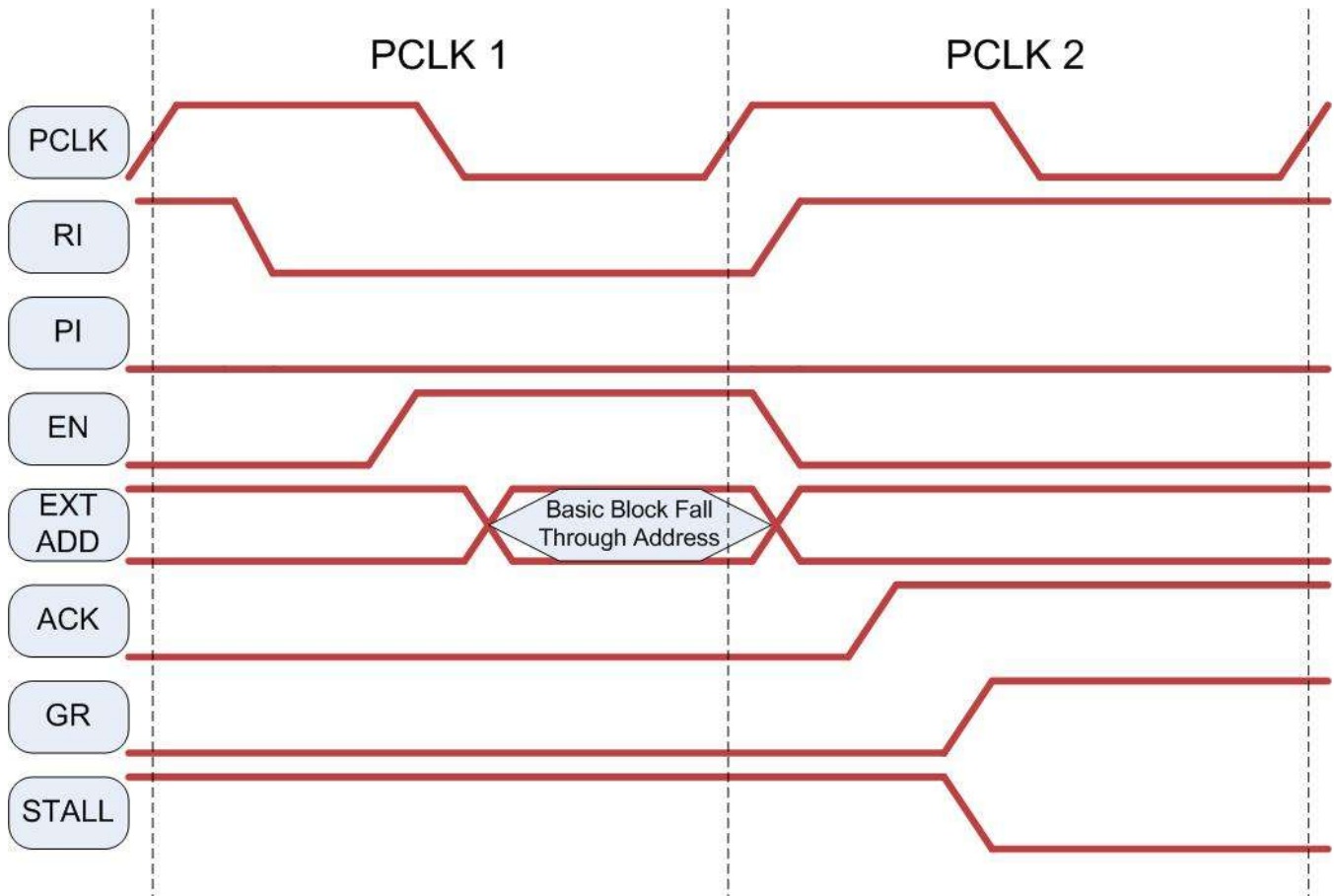


Figure 13: Instruction Decode/Arbitration Protocol

On the rising edge of the pipeline clock, PCLK, the program counter, PC, and the instruction itself will be presented to the instruction decode logic of the TISA. If the Extension's load bit in the system coprocessor is enabled, the Extension will also see the program counter and the current instruction. If the Extension recognizes the instruction it lowers its recognized instruction signal, RI, and keeps it low until the next positive edge of the PCLK. This is when the next instruction in the pipeline will be presented to the Extension and the TISA. The arbitration process begins after the Extension lowers its RI signal. The pipeline arbiter takes the RI signals from the TISA and from each Extension and based on the priority stored in the Extension Control Register, it decides which available path will execute the instruction. If the Extension wins arbitration, the pipeline arbiter will drive high the Extensions enable port, EN, until the next positive edge of PCLK (see Figure 13). During the next PCLK cycle, if the Extension requires multiple clocks or resources such as register or memory values, the Extension must raise its acknowledge signal, ACK, to request them from the pipeline arbiter. After the Extension asserts its ACK signal the pipeline arbiter responds by raising the grant signal, GR, for that Extension. At this point the paths to the pipeline registers and memory are switched to the Extension and the pipeline is stalled. The pipeline arbiter remembers which Extension it enabled during the last cycle and only responds to an ACK signal from that

Extension. The GR signal will remain high as long as the ACK signal is high. The ACK signal should be lowered after the positive edge of the PCLK cycle, before the Extension releases control of the pipeline. The GR will lower after the next positive edge of PCLK and the pipeline will resume as shown in Figure 14.

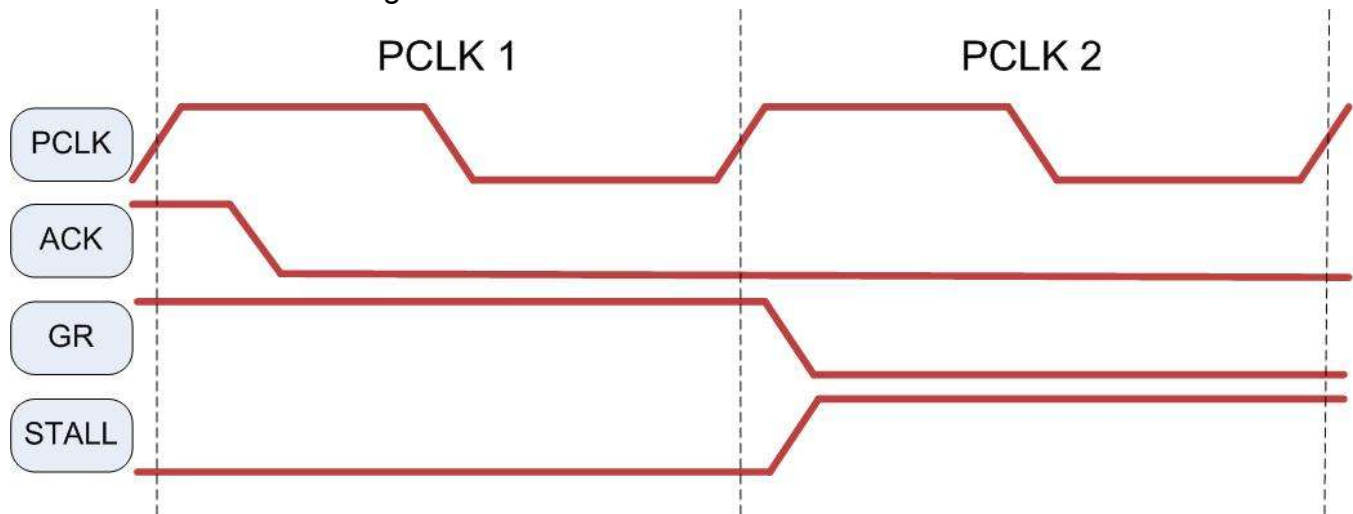


Figure 14: Resuming the Pipeline

When an instruction is assigned to the Extension, a NOP instruction replaces that instruction in the TISA pipeline. In certain cases it may instead be desirable to allow the TISA to execute the instruction normally, while the Extension performs some other operation in parallel (e.g. monitoring, profiling activities). In these cases the Extension must raise its Passive/Parallel Instruction, or PI, signal. The instruction is then piped to the execution phase of the TISA (instead of a NOP) but the Extension can still request pipeline resources and stall the pipeline like in the previous case. This is depicted in Figure 15.

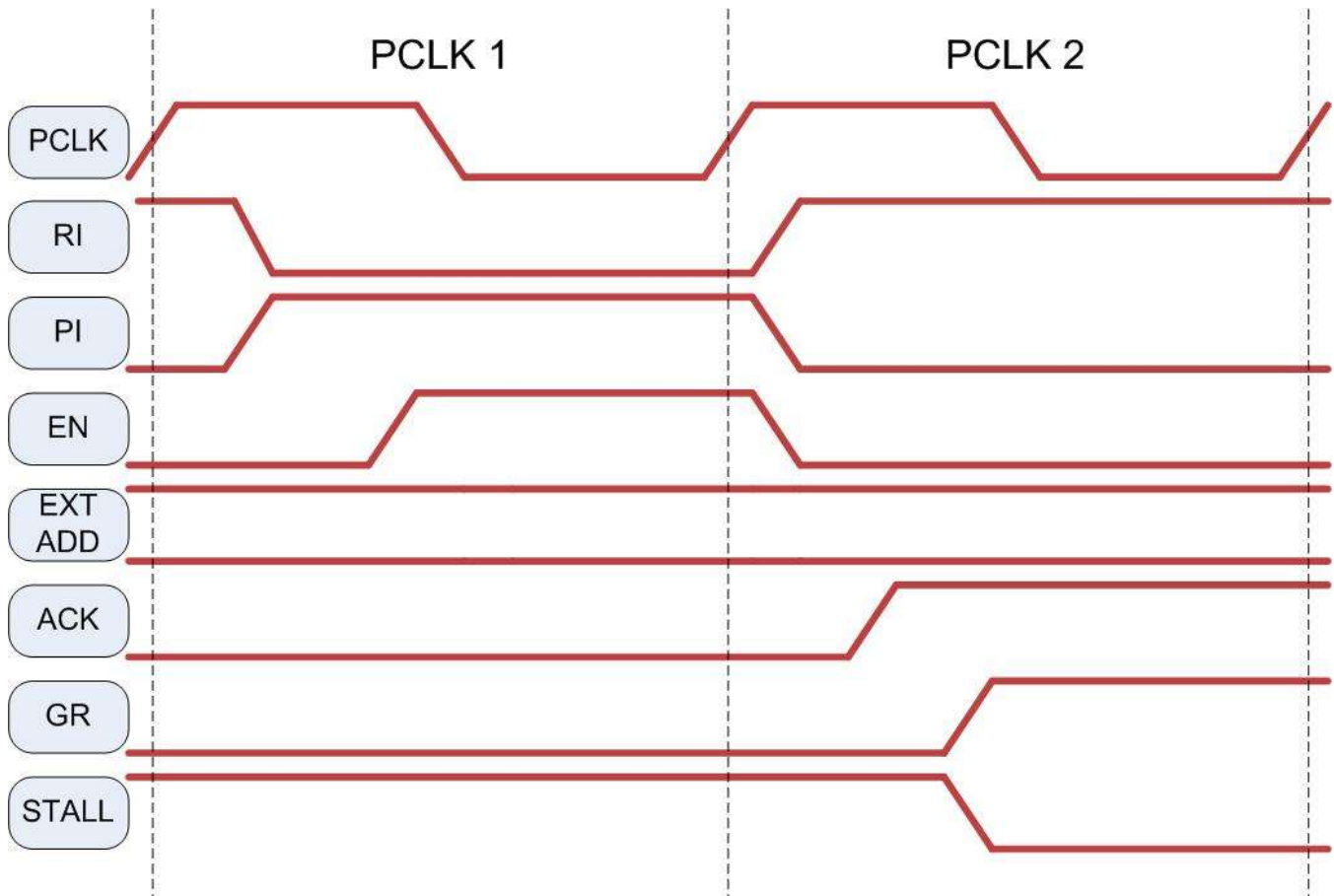


Figure 15: Instruction Decode/Arbitration Protocol for Passive/Parallel Operation

6.3.2 Register File Interface

The register file interface allows the Extension to read from and write to the general purpose register file. The TISA pipeline utilizes two read ports and one write port like the standard MIPS pipeline. It is expected that the Extension logic will usually provide a higher degree of parallelism than the TISA. To reduce the register data latency resulting from multiple register fetches, the Extension register file interface expands the number of ports to four read ports and two write ports. To minimize the number of bus macros and the signals crossing the reconfigurable boundary, two ports in the interface function both as read and write ports. Ports one and two can be used as read ports or write ports, depending on the state of their REGWRITEX signals. Ports three and four are dedicated read ports.

For a register read from the Extension there is currently a four-clock-cycle latency, but this value might be reduced in a future release. Figure 16 shows the protocol for a register read operation. After the Extension's GR signal has been raised by the pipeline arbiter, the Extension may assert a register number on the read port and the data will be available after four clock cycles of the system clock, CLK. The data will remain on the data side of the port for up to four clock cycles, unless the register number is changed on the port.

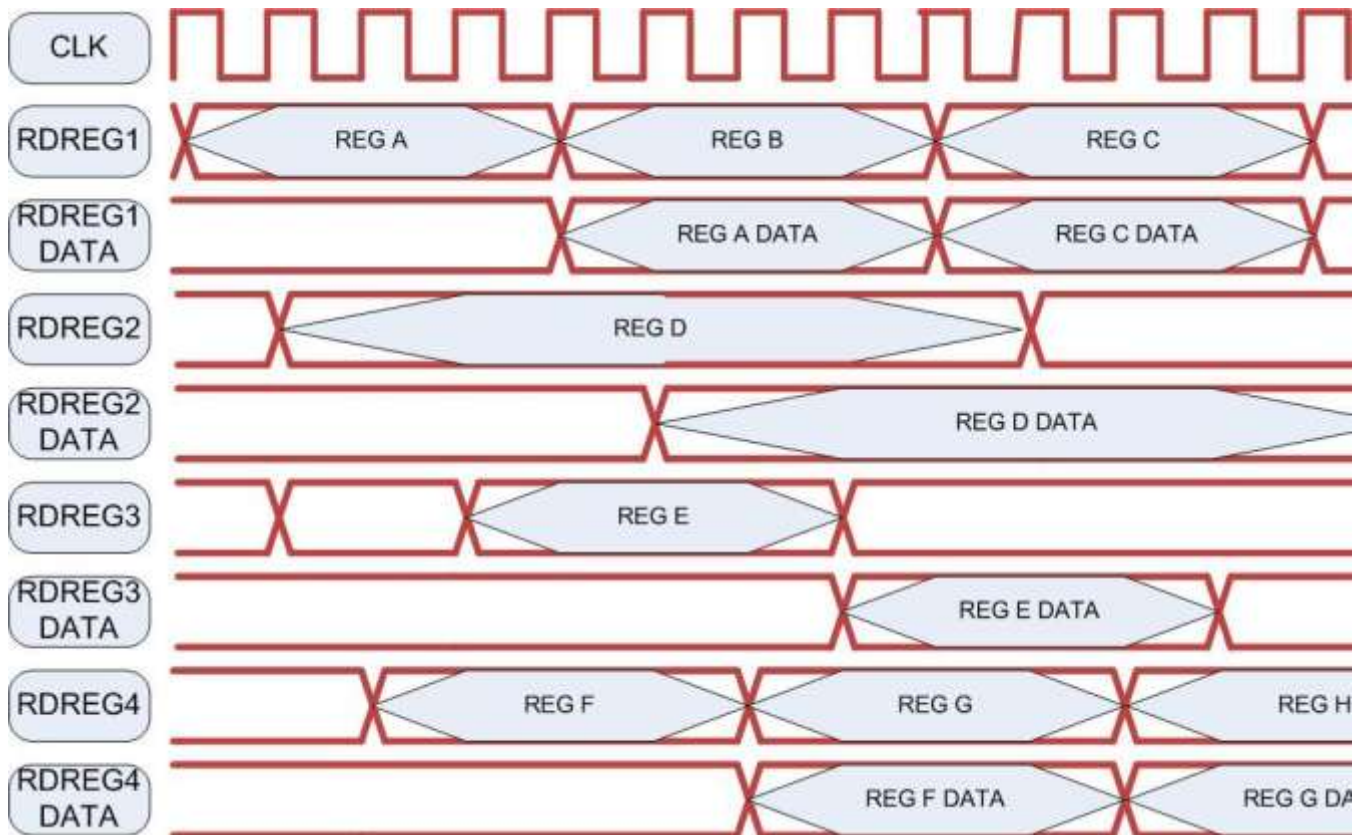


Figure 16: Register Read Interface

Write operations to the register file can be expedited if the Extension uses a write-and-forget approach. The Extension does not actually write to the register file directly, but rather to a FIFO buffer that writes its contents to the register file. It takes a single clock cycle for the Extension to write a register to the FIFO. Then some clock cycles later the FIFO writes that value to the register file. The current latency from the moment the Extension writes to the FIFO to the moment the FIFO writes to the register file is five to seven CLK cycles. The write register protocol is depicted in

Figure 17.

Before the Extension can write to the register file it must wait for the GR signal and the REGRDY signals to be both high. REGRDY high indicates that the FIFO is available and the Extension can write to it. In addition to REGRDY, the Extension must also obey the REGFULL and REGEMPTY signals. These signals indicate the current state of the FIFO, e.g. whether it is full or empty respectively. After all these conditions are met, the Extension can assert the register number and the data on one of the write ports. Then the Extension raises the REGWRITEX signal associated with that port for a single CLK cycle. This completes the write to the FIFO and the REGEMPTY signal will lower (if it is not already low). The Extension may

continue to write to the buffer from both write ports until the REGRDY signal is de-asserted. If this is due to the buffer being full, the REGFULL signal will assert at this time as well.

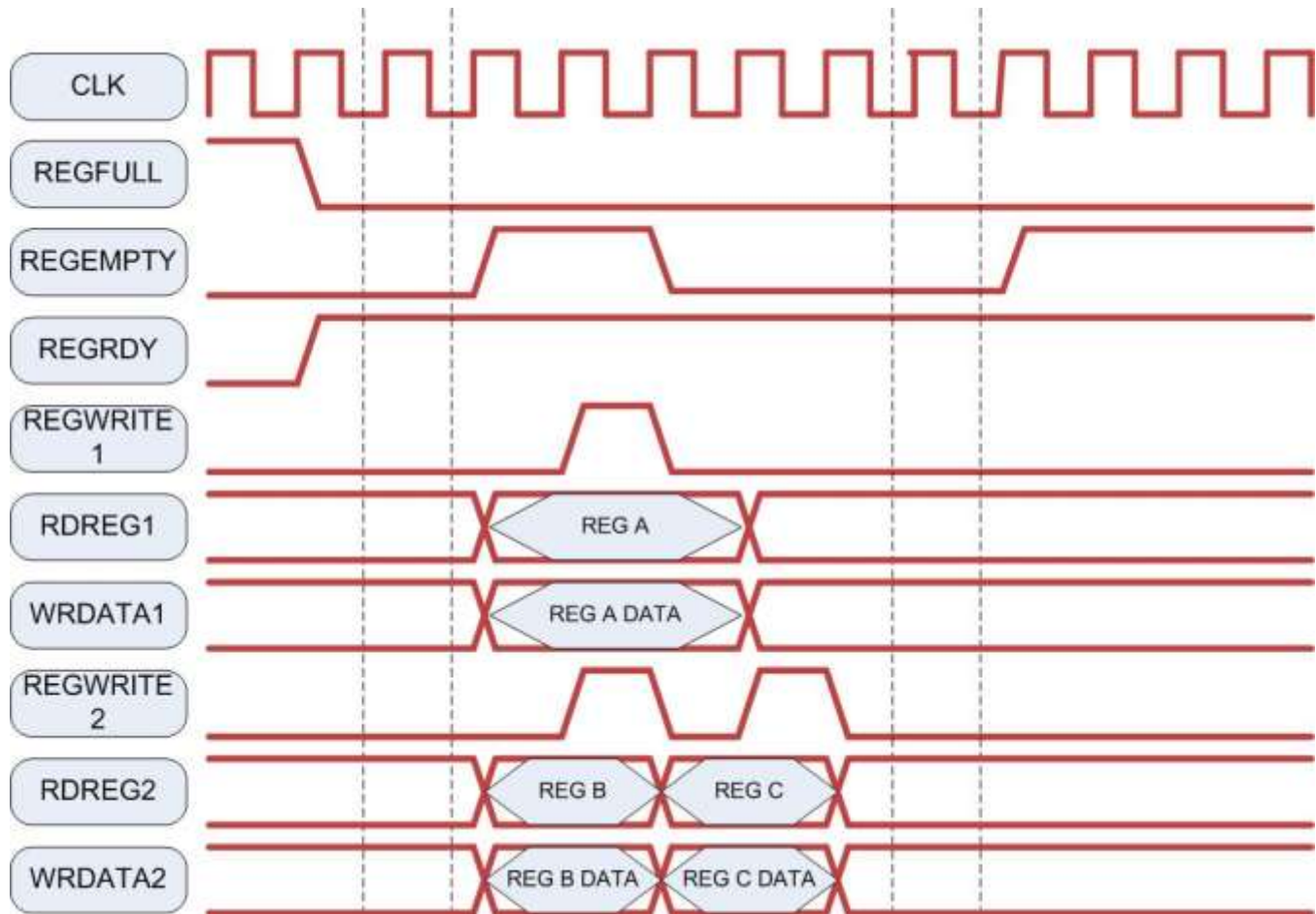


Figure 17: Write Register Interface

6.3.3 Memory Interface

The Extension can access the system memory and perform load and store operations. Currently the pipeline clock, or PCLK, and the memory subsystem are synchronized to each other. This method allows for at most one instruction fetch, data transaction, and instruction commit per PCLK cycle. It is used to simplify the flow of the pipeline for debugging. Future work will decouple these operations to reduce the effects of memory latency on performance. The memory interface is designed to capture memory requests on the falling edge of PCLK. To ensure that the signals are settled and have plenty of time to propagate across the logic of the FPGA, it recommended asserting the signals on the memory interface on the rising edge of the PCLK.

Figure 18 shows the protocol for a memory read. The Extension must assert the memory address and the output enable signal, or MOE, on the rising edge of PCLK. These signals must hold until the falling edge of PCLK. The additional control signals (BLS, HLS, RNL) are

not necessary in case of a read, because the system always performs word loads regardless of the setting of these signals. The Extension will receive an acknowledgement that the memory read request was received by the memory interface when the MDATA_VLD signal falls. The signal MDATA_VLD will remain low until the memory read is complete and the data appears on the MDATA_IN bus. PCLK remains low until all memory requests from the system have been serviced, including instruction fetch, data memory transaction and extension memory transaction. After the falling edge of PCLK, the Extension must lower the MOE signal and may remove the address from the interface. The Extension must wait until the MDATA_VLD signal goes high again before capturing the data on the MDATA_IN bus. Shortly after the MDATA_VLD signal raises so will PCLK. The data will remain valid on the bus until the next falling edge of the PCLK. The Extension may initiate another memory transaction during the next PCLK cycle.

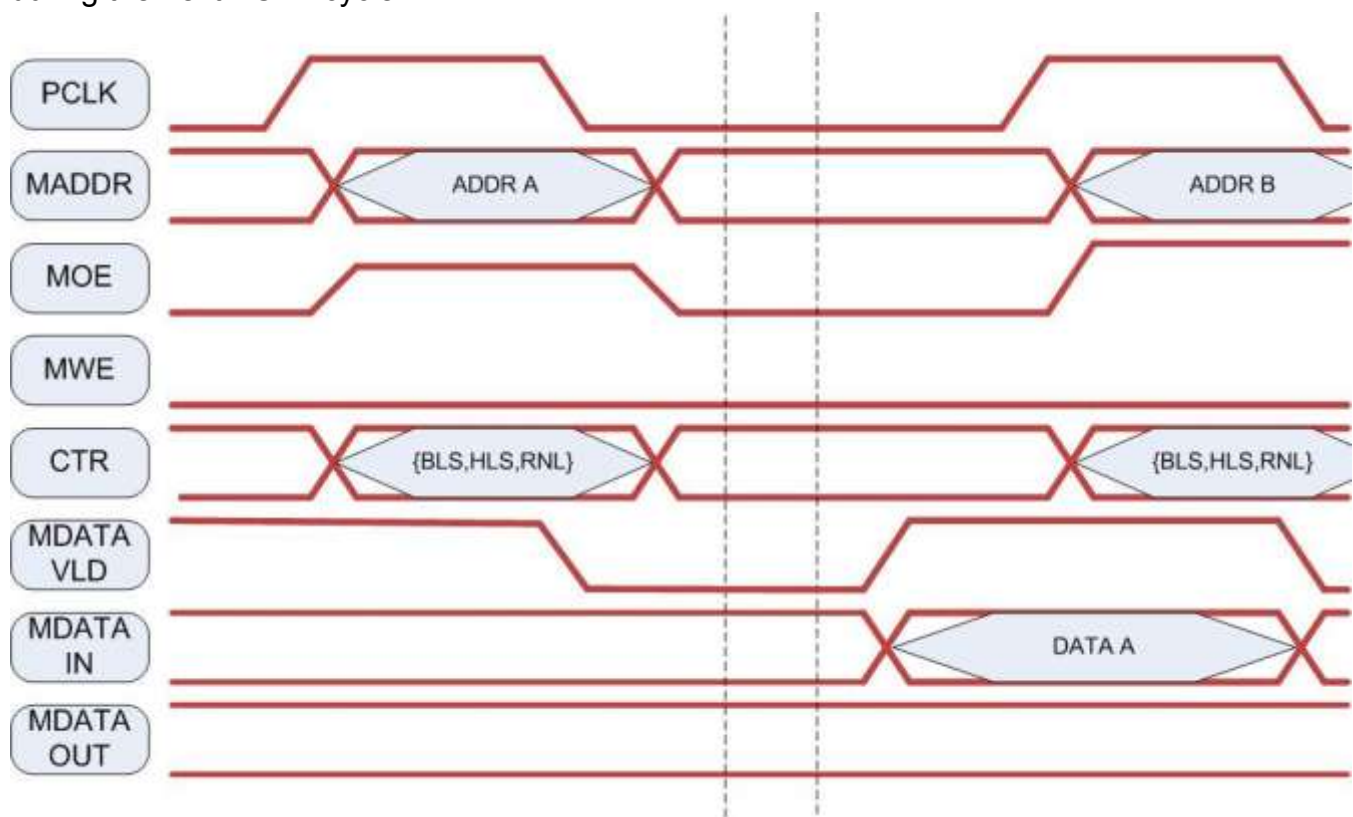


Figure 18: Memory Read Operation

Figure 19 shows the protocol for a memory write transaction. The Extension must assert the memory address, the data and the memory write enable signal, or MWE, on the rising edge of PCLK. The additional control signals BLS, HLS and RNL are used for byte, half word and unaligned stores. For non-word stores, all data must be aligned to the least significant side of the word. These signals must hold until the falling edge of PCLK. The Extension will receive an acknowledgement that the memory write request was received by the memory interface when the MDATA_VLD signal falls. MDATA_VLD will remain low until the memory read is complete and the data appears on the MDATA_IN bus. PCLK remains low until all

memory requests from the system have been serviced, including instruction fetch, data memory transaction and extension memory transaction. For a memory write the Extension does not need the data returned from the transaction and MDATA_VLD may be ignored after it falls. The Extension may initiate another memory transaction during the next PCLK cycle.

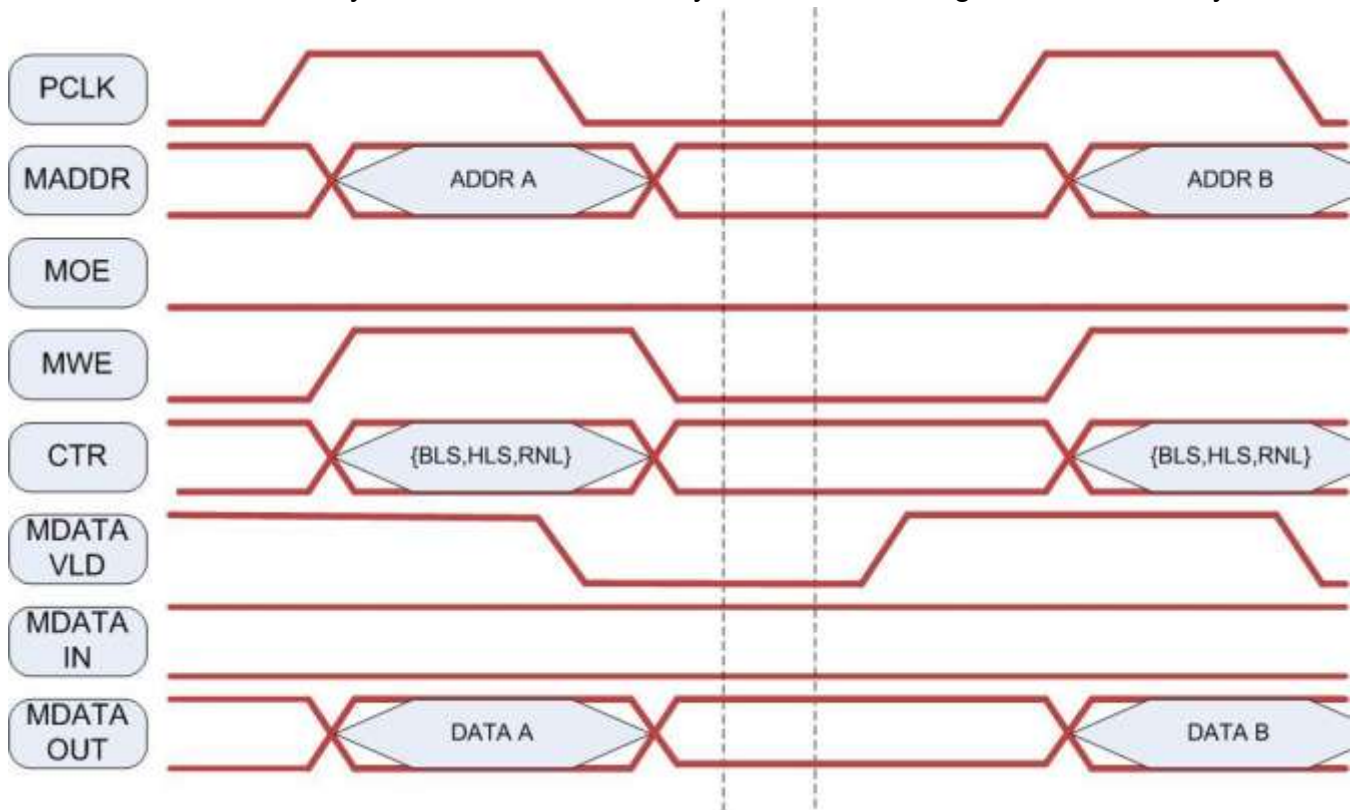


Figure 19: Memory Write Operation

6.3.4 Program Counter/Instruction Fetch

Extensions can modify the program counter, or PC, as a result of executing an instruction. This is needed, for instance, when optimizing basic blocks, because they all include a conditional branch. Other types of Extension may also require the ability to modify the PC.

Figure 20 shows the protocol for updating the PC. The signal PCNEXT indicates that the Extension wants to modify the PC. It should be raised while the GR signal is high and the pipeline is stalled. On the falling edge of PCLK, the Extension raises PCNEXT and asserts the new PC value on the EXTADD bus. The signals must hold until the rising edge of PCLK. The PC is always modified on the rising edge of PCLK. PCNEXT switches the multiplexer that routes the new PC to the PC register and enables the PC write.

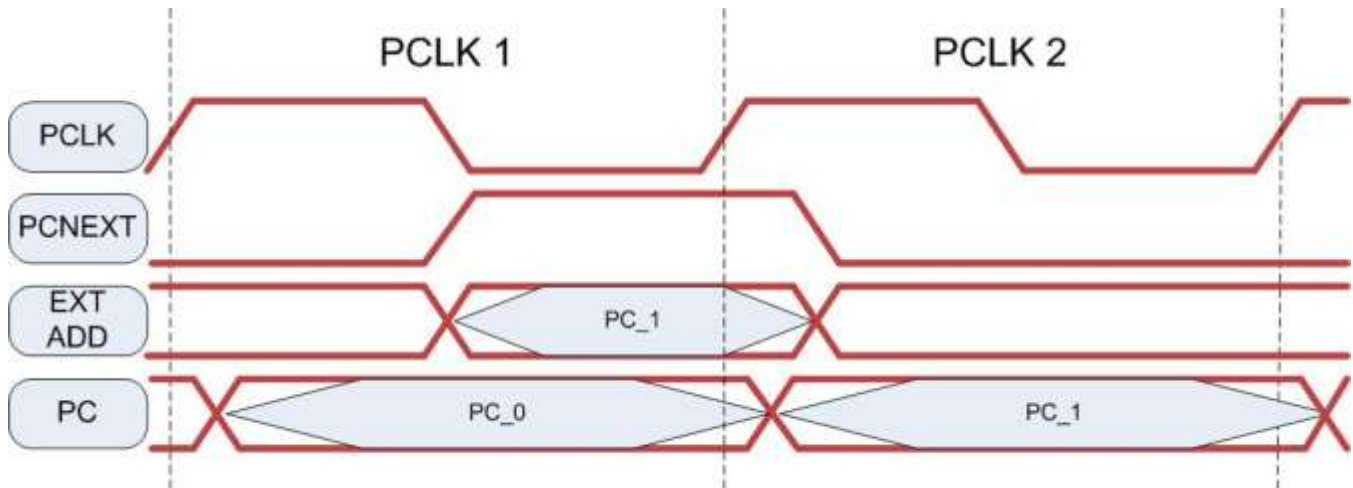


Figure 20: Extension PC update

6.3.5 Pipeline Re-Entry

In the previous sections we have shown how to carry out all the register and memory transactions on the Extension interfaces while the TISA pipeline is stalled. Stalling of the pipeline is required for all Extended instructions that take more than two cycles to complete. Sometimes an Extension only requires a single memory access or register writeback. In these cases it is not necessary to stall the pipeline because the resources in the TISA are sufficient to complete the execution. This optimization could reduce the latency by a whole pipeline clock, or PCLK, cycle. To use this optimization, the Extension interface provides for reentering the pipeline at either the memory access phase, or MA, or the writeback phase, or WB. The ports used to assert the pipeline signals for writing to the pipeline registers are multiplexed with other signals to reduce the number of bus macros and signals crossing the reconfigurable boundary. The pipeline reentry process takes place the on PCLK cycle before the pipeline resumes, when the Extension is releasing the pipeline resources.

Figure 21 shows the protocol for reentering the pipeline at MA. The Extension must lower the ACK signal in the last PCLK cycle before the pipeline resumes and after the positive edge of PCLK (see also Section 6.3.1). By now the data that will be written to the MA pipeline stage should be driven on the Extension interface ports. The table below provides the signal definitions.

Extension Port	Memory Access Pipeline Signal
RDREG2[0]	REGWRITE_EX
RDREG2[1]	MEMTOREG_EX
WRDATA1	ALURESULT_EX
RDREG1	WRREG_EX
RDREG4[2]	RNL_EX
Driven high (1'b1)	BRANCH_EX
EXTADD	PC_EX

Driven low (1'b0)	EXTNOP_EX
WRDATA2	REG2DATA_EX
RDREG4[3]	BLS_EX
RDREG4[1]	MEMREAD_EX
RDREG4[0]	MEMWRITE_EX
RDREG4[4]	HLS_EX
RDREG2[4]	MEXT_EX
RDREG2[3]	LC_EX

On the next negative edge of PCLK, the Extension must raise the Reenter at Memory Access signal, or REMA. While the GR signal is still high for the Extension, REMA will switch the multiplexers that route to the MA pipeline stage from the execute phase ports to the Extension interface ports. The pipeline registers latch on the rising edge of PCLK. After the positive edge of PCLK, the Extension must lower REMA and it may remove the pipeline data from the Extension ports. By this time the Extension GR will have been lowered and the pipeline has resumed.

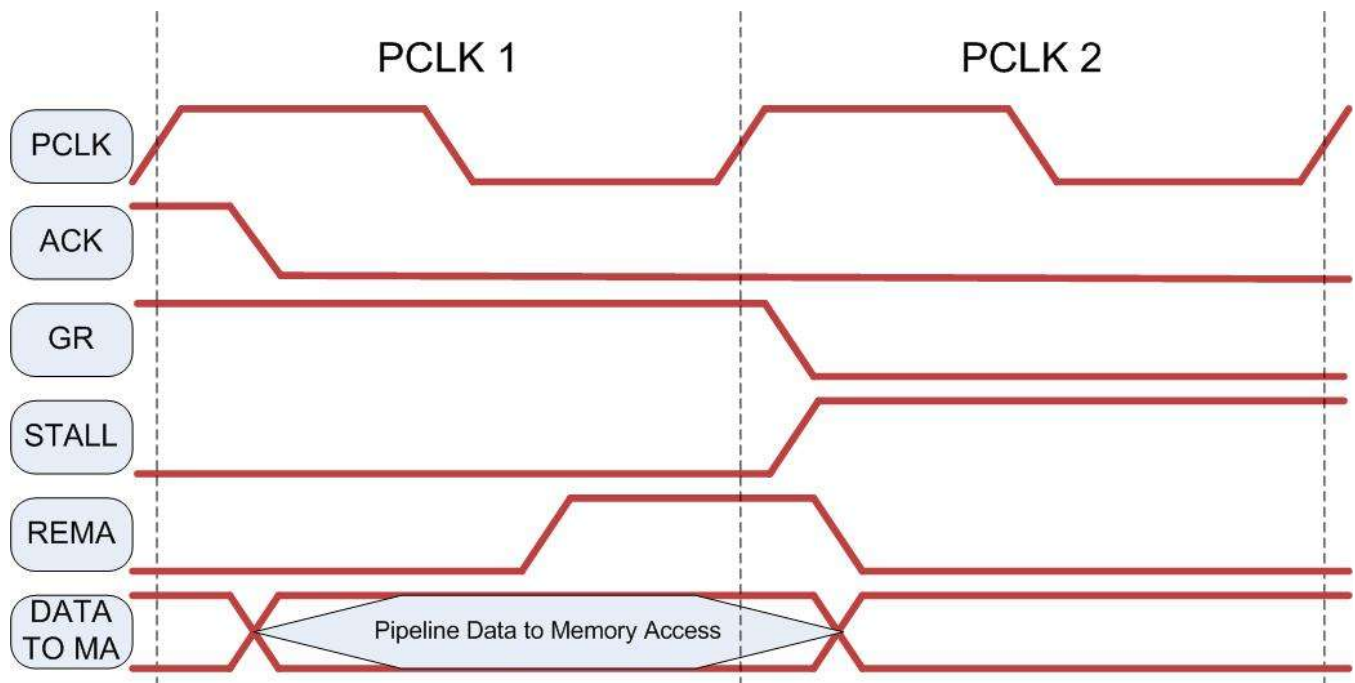


Figure 21: Reenter TISA Pipeline at Memory Access

Figure 22 shows the protocol for reentering the TISA pipeline at WB. The procedure is similar to the MA case. In the last PCLK cycle before the pipeline resumes, the Extension must lower its ACK signal after the positive edge of PCLK (see also Section 6.3.1). By now the data that will be written to the WB pipeline register should be driven on the Extension interface ports. The table below provides the signal definitions.

Extension Port	Writeback Pipeline Signal
RDREG2[0]	REGWRITE_MA
RDREG2[1]	MEMTOREG_MA
WRDATA1	ALURESULT_MA
RDREG1	WRREG_MA
RDREG4[2]	RNL_MA
Driven high (1'b1)	BRANCH_MA
EXTADD	PC_MA
Driven low (1'b0)	EXTNOP_MA
WRDATA2	DMDATAOUT_MA
RDREG4[3]	BHLS_MA
RDREG4[1:0]	DMADD_MA

On the next negative edge of the PCLK, the Extension must raise the Reenter at Writeback signal, or REWB. While the GR signal is still high for the Extension, REWB will switch the multiplexers that route to the WB pipeline stage from the memory access phase ports to the Extension interface ports. The pipeline registers latch on the rising edge of PCLK. After the positive edge of PCLK, the Extension must lower REWB and it may remove the pipeline data from the Extension ports. By this time the Extension GR will have been lowered and the pipeline has resumed.

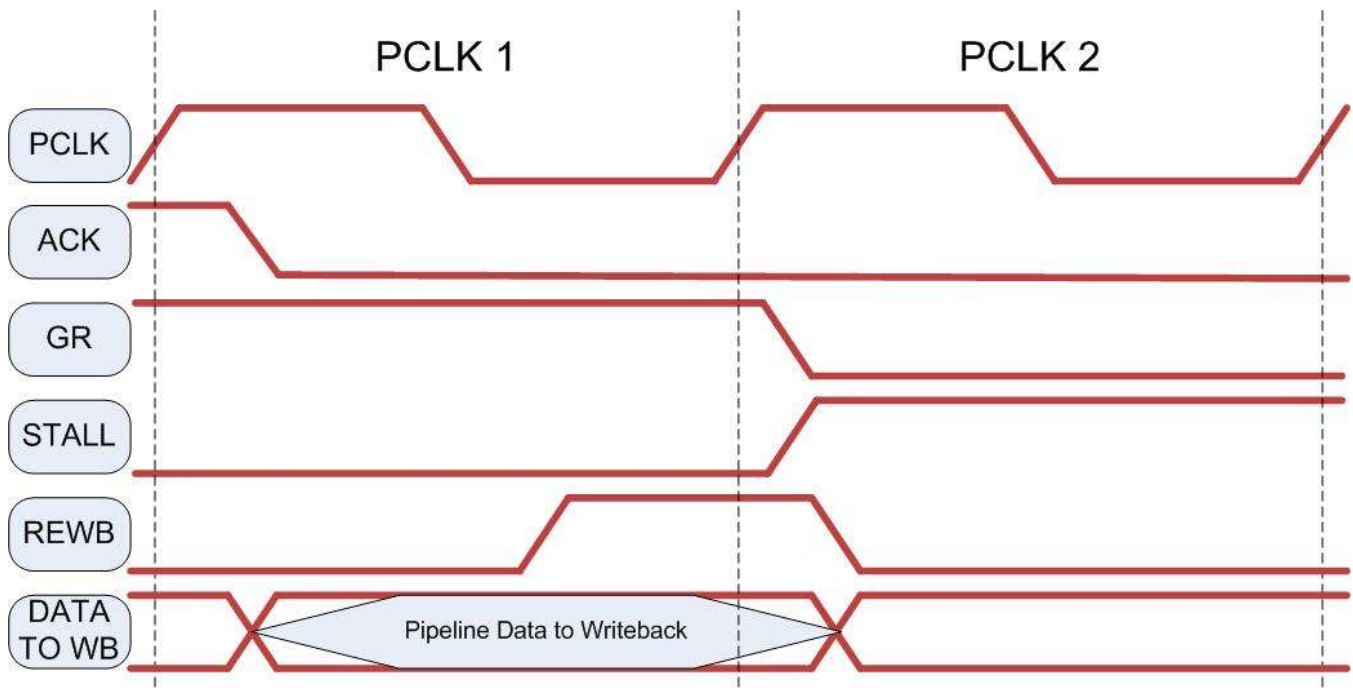


Figure 22: Reenter TISA Pipeline at Writeback

6.4 Design Floor planning

Users that perform advanced experiments with eMIPS may need to modify the Extension interface to suit their application. For instance, any modification that alters the number of ports, the size of ports or the direction of ports will require that the entire eMIPS system be re-floor-planned. A constraint file with floor planned constraints is provided in the release. Figure 23 is a graphical representation of the current eMIPS floor plan generated by Xilinx PlanAhead. The long purple rectangle on the left side of the FPGA is the reconfigurable region containing the Extension.

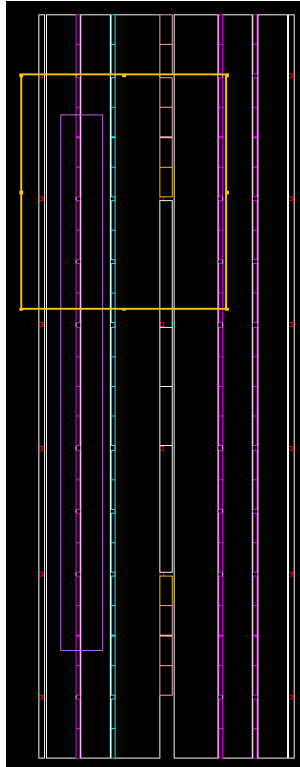


Figure 23: eMIPS Floor Plan

Since the eMIPS system is a dynamically reconfigurable design and the Extension is a reconfigurable region within it, all the signals that cross from the Extension to the rest of the design must pass through a Bus Macro. The only exception to this rule are clock signals. Changing the Extension interface ports may require changing the connectivity of the Bus Macros and possibly adding or removing Bus Macros. All the Bus Macros must be constrained to positions along the region boundary. Figure 24 provides a close up of the Bus Macros in the eMIPS system, represented by the small orange squares.

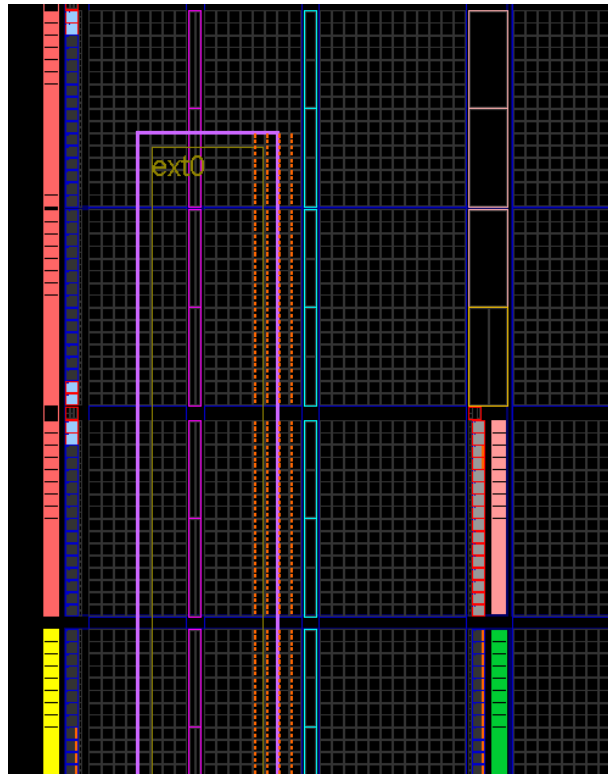


Figure 24: Close up of eMIPS Floor Plan with Bus Macros

The placement of the Bus Macros can be changed graphically using Xilinx PlanAhead or manually using a text editor. For instructions on how to use Xilinx PlanAhead, see the documentation available at the 'Xilinx Partial Reconfiguration Early Access Lounge' at <http://www.xilinx.com/support/prealounge/protected/index.htm>. To assign the Bus Macros locations manually, the procedure is to use a text editor to enter a LOC constraint for each Bus Macro in the provided implementation constraint file (*.ucf). There are some rules for the placement of Bus Macros. The Bus Macro must evenly straddle the edge of the reconfigurable region. Also, the Bus Macro is a hard macro that is two by four slices large. For this reason the x-coordinate must be a number divisible by four and the y-coordinate a number divisible by two. The following is an example of a bus macro LOC constraint: `INST "bmi0bm0" LOC = SLICE_X12Y164;`.

As the Extension designs grow in complexity, we may exhaust the limited chip resources available in the Extension region. Chip area as a whole is a highly limited resource. The baseline eMIPS system currently utilizes about 80% of the area of the Virtex 4 LX25 device available on the ML401 board. The Extension area is about 12% of the total. For this reason, we do not recommend increasing the size of the reconfigurable region, unless all other methods to fit the design into the space provided have been exhausted. To change the reconfigurable region dimensions graphically we recommend using Xilinx PlanAhead. To change the reconfigurable region dimensions manually, a user can change the existing AREA GROUP constraints for the reconfigurable region in the provided floor-planned implementation

constraint file (*.ucf). The AREA GROUP RANGE constraints must include all of the slices, blockrams, fifos and dsp48s within the rectangular region for the Extension.

7 References

- [1] Atmel *ARM Thumb Microcontrollers: AT91M63200*. Atmel Corporation, 1999. Available at http://www.atmel.com/dyn/resources/prod_documents/DOC1028.PDF
- [2] Atmel *AT91EB63 Evaluation Board User Guide*. Atmel Corporation, 2001. Available at http://www.atmel.com/dyn/resources/prod_documents/DOC1359.PDF#search=%22AT91EB63%20Evaluation%20Board%20User%20Guide%22
- [3] Athanas, P., Silverman, H. *Processor Reconfiguration through Instruction-Set Metamorphosis*. Computer Vol. 26, March 1993, pp. 11-18.
- [4] Xilinx *Development System Reference Guide, Chapter 4, Modular Design*. Xilinx Inc., December 2005, pp. 75-112. Available at <http://toolbox.xilinx.com/docsan/xilinx8/books/docs/dev/dev.pdf>
- [5] Xilinx *Development System Reference Guide, Chapter 5, Partial Reconfiguration*. Xilinx Inc., December 2005, pp. 113-140, Available at <http://toolbox.xilinx.com/docsan/xilinx8/books/docs/dev/dev.pdf>
- [6] Xilinx *Chipscope Pro Software and Cores User Guide*. Xilinx Inc., October 2005, Available at http://www.xilinx.com/ise/verification/chipscope_pro_sw_cores_8_li_ug029.pdf
- [7] Clark, N., Blome, J., Chu, M., Mahlke, S., Biles, S., Flautner, K. *An Architecture Framework for Transparent Instruction Set Customization in Embedded Processors*. ISCA 2005, pp. 272-283.
- [8] Xilinx *FPGA Editor Guide*. Xilinx Inc., June 1999. Available at http://www.xilinx.com/support/sw_manuals/2_li/download/fpedit.pdf
- [9] Hennessy, J. L., Patterson, D.A. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann Publishers, San Francisco, CA. 1998.
- [10] Kane, G., Heinrich, J. *MIPS RISC Architecture*. Prentice Hall, Upper Saddle River, NJ. 1992.
- [11] Sutherland, S. *The Verilog PLI Handbook, 2nd ed.* Kluwer Academic Publishers, Norwell, MA. 2002.
- [12] Xilinx *System ACE Compact Flash Solution*. Xilinx Inc., April 2002. Available at <http://www.xilinx.com/bvdocs/publications/ds080.pdf>
- [13] Xilinx *Two Flows for Partial Reconfiguration: Module Based or Difference Based*. Xilinx Inc., November 2003. Available at <http://www.xilinx.com/bvdocs/appnotes/xapp290.pdf>
- [14] Xilinx *Using Partial Reconfiguration to Time Share Device Resources in Virtex II and Virtex II Pro*. Xilinx Inc., May 2005.
- [15] Xilinx *Virtex 4 Configuration Guide*. Xilinx Inc., January 2006. Available at <http://direct.xilinx.com/bvdocs/userguides/ug071.pdf>
- [16] Xilinx *Virtex 4 Datasheet: DC and Switching Characteristics*. Xilinx Inc., February 2006. Available at <http://direct.xilinx.com/bvdocs/publications/ds302.pdf>
- [17] Xilinx *Virtex 4 Family Overview*. Xilinx Inc., June 2005. Available at <http://direct.xilinx.com/bvdocs/publications/ds112.pdf>
- [18] Xilinx *Virtex 4 Packaging and Pinout Specification*. Xilinx Inc., September 2005. Available at <http://direct.xilinx.com/bvdocs/userguides/ug075.pdf>
- [19] Xilinx *Virtex 4 User Guide*. Xilinx Inc., September 2005. Available at <http://direct.xilinx.com/bvdocs/userguides/ug070.pdf>
- [20] Xilinx. *Virtex-4 Development Boards*. Xilinx Inc., 2005. At http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex4/index.htm
- [21] Mentor Graphics *ModelSim* at http://www.mentor.com/products/fpga_pld/simulation/index.cfm

- [22] Microsoft Giano at <http://research.microsoft.com/downloads/> and <http://www.ece.umd.edu/~behnam/giano.html>
- [23] Burger, D., Austin, T. M. *The SimpleScalar Tool Set, Version 2.0*. Technical Report 1342, June 1997, University of Wisconsin-Madison.
- [24] Forin, A., Neekzad, B., Lynch, N., L. *Giano: The Two-Headed Simulator*. Microsoft Research Technical Report MSR-TR-2006-130, September 2006.
- [25] Bossuet, L., Gogniat, G., Burleson, W. *Dynamically Configurable Security for SRAM FPGA Bitstreams*. International Journal of Embedded Systems, 2006.
- [26] Bartzoudis, N., G., et al. *Reconfigurable Computing and Active Networks*. ERSA '03, Las Vegas, NV pp. 27-33.
- [27] Al Faruque, M., A. *Fine Grained Application Profiling for Guiding Application Specific Instrucion Set Processor (ASIPs) Design*. Master Thesis, 2004, Aachen University.
- [28] Clark, N., Zhong, H., Mahlke, S. *Processor Acceleration Through Automated Instruction Set Customization*. Micro '03, 2003.
- [29] Clark, N. et al. *Application-Specific Processing on a General-Purpose Core via Transparent Instruction Set Customization*. Micro '04, 2004.
- [30] Yehia, S. et al. *Exploring the Design Space of LUT-based Transparent Accelerators*. CASES '05, 2005.
- [31] Yehia, S., Teman, O. *From Sequences of Dependent Instructions to Functions: An Approach for Improving Performance without ILP or Speculation*" ISCA '04, 2004.
- [32] Sun, F. et al. *Synthesis of Custom Processors Based On Extensible Platforms*. ICCAD '02, 2002.
- [33] Bracy, A., Prahlad, P., Roth, A. *Dataflow Mini-Graphs: Aplifying Superscalar Capacity and Bandwidth*. MICRO '04, 2004.
- [34] Brisk, P., Kaplan, A., Sarrafzadeh, M. *Area-Efficient Instrucion Set Synthesis for Reconfigurable System-on-Chip Designs*" DAC '04, 2004.
- [35] Dales, M. *Managing a Reconfigurable Processor in a General Purpose Workstation Environment*. DATE '03, 2003.
- [36] Forin, A., Lynch, N., L., Pittman, R. N. *Software Support for Dynamically Extensible Processors*. Microsoft Research Technical Report MSR-TR-2006-147, October 2006.
- [37] Yu, P, Mitra, T. *Characterizing Embedded Applications for Instruction-Set Extensible Processors*. DAC 2004, San Diego CA.
- [38] Fahs, B. et al. *Performance Characterization of a Hardware Framework for Dynamic Optimization*. 34th ISM, December 2001.
- [39] Razdan, R., Smith, M. D. *High-Performance Microarchitectures with Hardware-Programmable Functional Units*. 27th ISM, pagg. 172-180, November 1994.
- [40] Hauser, J. R., Wawrzynek, J. *Garp: A MIPS Processor with a Reconfigurable Coprocessor*. FCCM'97 pagg 12-21, April 1997.
- [41] Lau, D., Pritchard, O., Molson, P. *Automated Generation of Hardware Accelerators with Direct Memory Access from ANSI/ISO Standard C Functions*. FCCM'06, pagg. 45-54, April 2006.
- [42] Hadžić, I., Udani, S., Smith, J. M. *FPGA Viruses*. FPLA'99, pagg291-300, September 1999.
- [43] Wittig, R. D., Chow, P. *OneChip: An FPGA Processor With Reconfigurable Logic*. FCCM'96, pagg. 126-135, 1996.
- [44] Carrillo, J. E., Chow, P. *The Effect of Reconfigurable Units in Superscalar Processors*. FPGA'01, pagg. 141-150, February 2001.
- [45] Lysecky, R., Stitt, G., Vahid, F. *Warp Processors*. DAES Transactions, pagg659-681, July 2006.

- [46] Lysecky, R., Vahid, F. *A Configurable Logic Architecture for Dynamic Hardware/Software Partitioning*. DATE'04, 2004.
 - [47] Estrin, G. *Organization of computer systems: The fixed plus variable structure computer*. Proc. Western Joint Computer Conference, pagg 33-40, New Yowrk 1960.
 - [48] Sawitzki, S., Köhler, S., Spallek, R. *Prototyping Framework for Reconfigurable Processors*. FPL'01, pagg. 6-16, 2001.
 - [49] Goldstein, S. C., et al. *PipeRench: A Reconfigurable Architecture and Compiler*. IEEE Computer, 2000.
 - [50] Schmit, H. Et al. *PipeRench: A Virtualized Programmable Data path in 0.18 Micron Technology*. IEEE CICC'02, 2002.
 - [51] Rowen, C, Maydan, D. *Automated Processor Generation for System-on-Chip*. ESSCIRC'01, 2001.
 - [52] Anderson, E. et al. *Enabling a Uniform Programming Model across the Software/Hardware Boundary*. FCCM'06, pagg. 89-98, April 2006.
 - [53] Altera Corp. *Excalibur Embedded Processor Solutions*, 2005.
<http://www.altera.com/products/devices/excalibur/excindex.html>,
 - [54] Stretch, Inc. <http://www.stretchinc.com> 2006.
 - [55] Tarari, Inc. <http://www.tarari.com> 2002.
 - [56] SRC Computers Inc. <http://www.srccomp.com> 1996.
 - [57] Mitronics, Inc. <http://www.mitronics.com> 2001.
 - [58] Jacob, J., A., Chow, P. *Memory Interfacing and Instruction Specification for Reconfigurable Processors*. FPGA'99, 1999.
 - [59] Davidson, J. *FPGA Implementation of a reconfigurable microprocessor*. CICC'93, May 1993.
 - [60] Hauck, S. et al. *Totem: Domain-Specific Reconfigurabel Logic*. IEEE Transs VLSI.
 - [61] Hauck, S. et al. *The Chimaera Reconfigurable Functional Unit*. IEEE VLSI, 2004.
 - [62] Hauck, S., Agarwal, A. *Software Technologies for Reconfigurable Systems*. NW Univ. Technical Report, 1996.
 - [63] Helander, J., Forin, A. *MMLite: A Highly Componentized System Architecture*. Eight ACM SIGOPS European Workshop, Sintra, Portugal, September 1998.
- Download at <http://research.microsoft.com/invisible/>
- [64] Y. Lai, and P. Wang, *Hierarchical interconnection structures for field programmable gate arrays* IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Volume: 5 Issue: 2, June 1997 Page(s): 186 –196.
 - [65] Aggarwal, A.A.; Lewis, D.M. *Routing architectures for hierarchical field programmable gate arrays* Computer Design: VLSI in Computers and Processors, 1994. ICCD '94. Proceedings., IEEE International Conference on, 10-12 Oct. 1994 Page(s): 475 –478.
 - [66] W. Li, D.K. Banerji, *Routability prediction for hierarchical FPGAs* Ninth Great Lakes Symposium on VLSI, pp. 256 –259 4-6 March 1999
 - [67] V. Betz, *Architecture and CAD for the Speed and Area Optimization of FPGAs* Ph.D. Dissertation, University of Toronto, 1998.
 - [68] Mark Nelson, *Fast String Searching With Suffix Trees*, Dr. Dobb's Journal, August, 1996. At <http://www.dogma.net/markn/articles/suffixt/suffixt.htm>
 - [69] K. Sarrigeorgidis, and J. M. Rabaey, *Massively Parallel Wireless Reconfigurable Processor Architecture and Programming* 10th Reconfigurable Architectures Workshop, Nice, France, April 22, 2003.

- [70] H. Zhang, M. Wan, V. George, and J. Rabaey, *Interconnect Architecture Exploration for Low-Energy Reconfigurable Single-Chip DSPs*. IEEE Computer Society Workshop on VLSI '99 pp. 2-8, April 1999.
- [71] H. Zhang, V. Prabhu, V. George, M. Wan, M. Benes, A. Abnous, and J. M. Rabaey, *A I-V Heterogeneous Optimization by DAG Matching* Proc. of DAC 1987.
- [72] M. Wan, H. Zhang, V. George, M. Benes, A. Abnous, V. Prabhu, and J. M. Rabaey, *Design Methodology of a Low-Energy Reconfigurable Single-Chip DSP System* Journal of VLSI Signal Processing Systems, 28, pp. 47-61, May-June 2001.
- [73] J. Becker, and M. Glesner, *A Parallel Dynamically Reconfigurable Architecture Designed for Flexible Application-Tailored Hardware/Software Systems in Future Mobile Communication* The Journal of Supercomputing, 19(1), pp. 105-127, 2001.
- [74] Atmel Corp. *FPSLIC (AVR with FPGA)*, 2005. At: <http://www.atmel.com/products/FPSLIC/>.
- [75] Berkeley Design Technology, Inc. , 2004. Available at: http://www.bdti.com/articles/info_eet0207fpga.htm#DSPEnhanced%20FPGAs.
- [76] Böhm, W., J. Hammes, B. Draper, M. Chawathe, C. Ross, R. Rinker, and W. Najjar. *Mapping a Single Assignment Programming Language to Reconfigurable Systems*. The Journal of Supercomputing, Vol. 21, pp. 117-130, 2002.
- [77] Chen, W., P. Kosmas, M. Leeser, C. Rappaport. *An FPGA Implementation of the Two-Dimensional Finite-Difference Time-Domain (FDTD) Algorithm*, International Symposium on Field-Programmable Gate Arrays (FPGA), 2004.
- [78] *Critical Blue*, at <http://www.criticalblue.com>, 2005.
- [79] Ernst, R., J. Henkel, T. Benner. *Hardware-Software Cosynthesis for Microcontrollers*. IEEE Design & Test of Computers, pages 64-75, October/December 1993.
- [80] Gokhale, M., J. Stone. *NAPA C: Compiling for hybrid RISC/FPGA architectures*. IEEE Symposium on FPGAs for Custom Computing Machines (FCCM), 1998.
- [81] Gordon-Ross, A., F. Vahid. *Frequent Loop Detection Using Efficient Non-Intrusive On-Chip Hardware*. Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2003.
- [82] Guo, Z., B. Buyukkurt, W. Najjar and K. Vissers. *Optimized Generation of Data-Path from C Codes*. ACM/IEEE Design Automation and Test Europe (DATE), 2005.
- [83] Keane, J., C. Bradley, Clark, C. Ebeling. *A Compiled Accelerator for Biological Cell Signaling Simulations*, International Symposium on Field-Programmable Gate Arrays (FPGA), 2004.
- [84] *Triscend Corp.* <http://www.triscend.com>, 2003.
- [85] Venkataramani, G., W. Najjar, F. Kurdahi, N. Bagherzadeh, W. Bohm. *A Compiler Framework for Mapping Applications to a Coarse-grained Reconfigurable Computer Architecture*. Conference on Compiler, Architecture, and Synthesis for Embedded Systems (CASES), 2001.
- [86] Zaghera, M., B. Larson, S. Turner, and M. Itzkowitz. *Performance Analysis Using the MIPS R10000 Performance Counters*. Supercomputing, Nov. 1996.
- [87] Zhang, X., et al. *System Support for automatic Profiling and Optimization*. Proceedings of the 16th Symposium on Operating Systems Principles, 1997.
- [88] Zilles, C.B. and G.S. Sohi. *A Programmable Co-processor for Profiling*. International Symposium on High-Performance Computer Architectures, 2001.
- [89] Dean, J., et al. *ProfileMe: Hardware Support for Instruction-Level Profiling on Out-of-Order Processors*. MICRO, 1997.
- [90] Graham, S.L., P.B. Kessler and M.K. McKusick. *gprof: a Call Graph Execution Profiler*. SIGPLAN Symp. on Compiler Construction, pp. 120-126, 1982.

- [91] Fu, W., K. Compton. *An Execution Environment for Reconfigurable Computing*. IEEE Symposium on Field-Programmable Custom Computing Machines, 2005.
- [92] Tensilica, Inc. <http://www.tensilica.com>, 2006.
- [93] Cong, J. et al. *Instruction set extension with shadow registers for configurable processors* FPGA'05, pagg 99-106, Monterey CA 2005.
- [94] Cong, J. et al. *Application-specific instruction generation for configurable processor architectures* FPGA'04, Monterey CA 2004.
- [95] Biswas, P., Banerjee, S., Dutt, N., Ienne, P., Pozzi, L. *Performance and Energy Benefits of Instruction Set Extensions in an FPGA Soft Core* VLSI'06, pag. 651-656
- [96] Simat, M., Cotofana, S., van Eijndhoven, J.T.J., Vassiliadis, S., Vissers, K., *An 8x8 IDCT Implementation on an FPGA-Augmented TriMedia* FCCM'01, Pagg. 160-169.
- [97] Simat, M., Cotofana, S., Vassiliadis, S van Eijndhoven, J.T.J., Vissers, K., *MPEG-compliant entropy decoding on FPGA-augmented TriMedia/CPU64* FCCM'02, pagg. 261- 270.
- [98] Guo, Z. et al. *A Quantitative Analysis of the Speedup Factors of FPGAs over Processors* FPGA'04, Monterey CA.
- [99] Borgatti, M., et al. *A Reconfigurable System Featuring Dynamically Extensible Embedded Microprocessor, FPGA, and Customizable I/O* IEEE Journal of Solid-State Circuits, March 2003, Vol. 38, pagg 521-529.
- [100] Cali, L., Lertora, F., Tazzina, C., Besana, M., Borgatti, M. *Platform IC with Embedded Via Programmable Logic for Fast Customization* CICC'04, pagg. 419-422.
- [101] Lertora, F., Borgatti, M. *Handling Different Computational Granularity by a Reconfigurable IS Featuring Embedded FPGAs and a Network-On-Chip* FCCM'05, pagg. 45-54.
- [102] DeHon, A., *DPGA-coupled microprocessors: commodity ICs for the early 21st Century* FCCM'94, pagg. 31-39.
- [103] Hauck, S. *The roles of FPGAs in reprogrammable systems* Proceedings of the IEEE, April 1998, Vol. 86, pagg.615-638.
- [104] Lu, H. and Forin, A. *The Design and Implementation of P2V, An Architecture for Zero-Overhead Online Verification of Software Programs*. Microsoft Research Technical Report MSR-TR-2007-99, August 2007.
- [105] Sukhwani, B., Forin, A., Pittman, R. N. *Extensible On-Chip Peripherals*. Microsoft Research Technical Report MSR-TR-2007-120, September 2007.
- [106] Meier, K., Forin, A. *MIPS-to-Verilog, Hardware Compilation for the eMIPS Processor*. Microsoft Research Technical Report MSR-TR-2007-128, September 2007.
- [107] Busonera, G., Forin, A. *eBug: Debugging Extensions for the eMIPS Dynamically Extensible Processor*. Microsoft Research Technical Report MSR-TR-2007-155, November 2007.

Appendix A: Example eMIPS Extensions

The following is a brief discussion of the example Extensions provided with the release. These examples are meant to be a primer for those seeking to develop their own Extensions. The collection includes working examples of all the Extension protocols and interfaces. The example Extensions are listed below:

1. *mmldiv64* – Hardware Extension for accelerating a software 64-bit division
2. *loadreturn* – Hardware Extension for implementing the return from function code sequence.
3. *timer* – Example of an Extension peripheral realizing an additional timer
4. *usart* – Example of an Extension peripheral realizing an additional USART
5. *eBUG HW 2* – Software Debug Extension with two hardware watchpoints
6. *eBUG HW 8* – Software Debug Extension with eight hardware watchpoints

Developers may refer to these examples for assistance in developing new extensions.

mmldiv64

The *mmldiv64* Extension accelerates the software implementation of the 64-bit division used by the Microsoft Invisible Computing RTOS. A portion of the software routine uses a series of instructions to perform a 128-bit shift-left-by-one operation across four registers. Performing an operation of this kind on 32-bit hardware requires many instructions and is inefficient. The *mmldiv64* Extension accelerates this operation by reading the four registers containing the operand data in parallel and loading them into a 128-bit shift register. After the data is loaded in the Extension the shift takes a single clock cycle. Then the Extension writes the new values of the registers back to the register file, two at a time using the available write ports. This Extension is therefore useful to learn how to access to the register file.

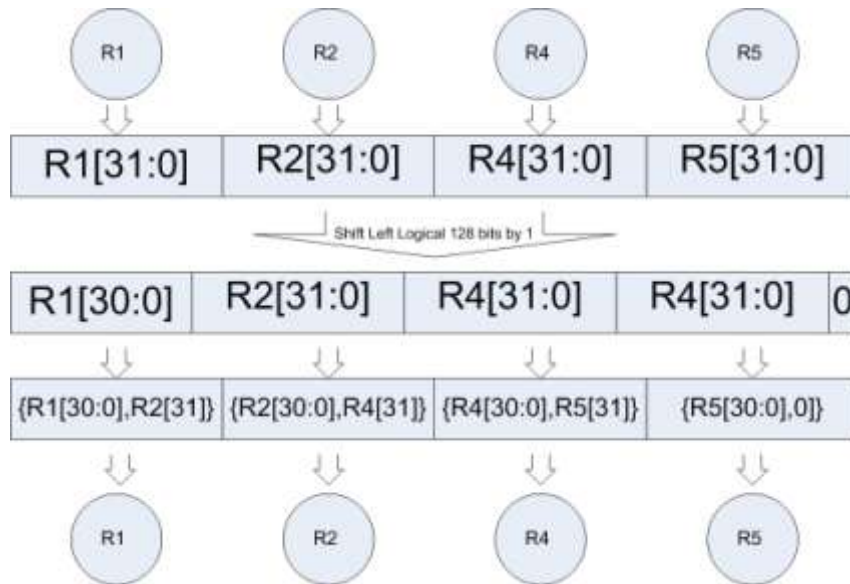


Figure 25: mmldiv64 shift 128-bit left logical

The *mmldiv64* Extension also performs a conditional jump. After the 128-bit shift is complete, the software basic block that the *mmldiv64* Extension replaces performs a conditional branch based on the new values. The Extension performs the branch test after the shift is performed. If the test determines that a jump is required, the Extension calculates the new PC and updates it in the instruction fetch unit using the PC update interface. If a branch is not necessary, the Extension allows the execution to resume at the fall through PC. This Extension is therefore useful to learn how to affect the control flow of the basic data path.

After the branch tests and the registers have been written back, the execution is returned to the TISA data path at the writeback stage.

loadreturn

The *loadreturn* Extension implements the typical return-from-function code sequence. This sequence includes three instructions that load the return address from the stack, update the stack pointer and jump to the return address. The Extension performs these three operations in a single Extended instruction. The *loadreturn* Extension reads the stack pointer register, then uses it to read the return address from memory. Then the Extension updates the stack pointer register using the immediate value. The Extension updates the PC to the value of the return address before returning execution back to the TISA at the memory access stage. This Extension is useful to learn how to perform read transactions on the memory interface.

timer

The *timer* Extension is an example of an Extension Peripheral. This Extension uses the peripheral hot-plug protocol to notify the operating system of its presence, and of its needs for resources, including address space and interrupt. The Extension first sends the key serially using its PRESENT output signal. If the key matches the key stored in the LOCK module, the Extension Controller is notified of the presence of a new peripheral and the processor is interrupted. The operating system takes over at this point and uses the interface of the Extension Controller to configure the Extension peripheral, based on the requested resources stored in the Extension's BATs. After the Extension peripheral is configured it transitions from the *configuration* to the *run* state, and from then on it runs like the other on-chip timer peripheral until it is unloaded.

The *timer* Extension uses the memory and configuration interfaces to the memory bus and two 64-bit counters based on the 100 MHZ system clock. One counter is a 64-bit down counter and the other is a 64-bit free counter. If interrupts are enabled for this peripheral in the interrupt controller module, software can use this peripheral to generate timing interrupts. The software interface of the Extension is the same as the built-in timer peripheral.

This Extension is useful to learn how loadable on-chip peripherals are loaded and unloaded, how they reply to memory transactions, and how they generate interrupts.

usart

The *usart* Extension is an example of an Extension Peripheral. This Extension uses the peripheral hot-plug protocol to notify the operating system of its presence, and of its needs for resources, including address space and interrupt. The Extension first sends the key serially using its PRESENT output signal. If the key matches the key stored in the LOCK module, the Extension Controller is notified of the presence of a new peripheral and the processor is interrupted. The operating system takes over at this point and uses the interface of the Extension Controller to configure the Extension peripheral, based on the requested resources stored in the Extension's BATs. After the Extension peripheral is configured it transitions from the *configuration* to the *run* state, and from then on it runs like the other on-chip USART peripheral until it is unloaded.

The *usart* Extension uses the memory and configuration interfaces to the memory bus and a fixed baud USART. The current eMIPS processor design includes only a single programmable USART. Using a Extension such as this one, software could utilize some unused pins on the ML401 board to provide an additional USART as needed.

This Extension is useful to learn how loadable on-chip peripherals are loaded and unloaded, how they reply to memory transactions, how they generate interrupts, and how they can access I/O pins on the FPGA fabric.

eBUG HW 2

The eBUG HW 2 is a software debugging solution implemented using the eMIPS Extension hardware. The Extension monitors the execution flow, and the register and memory traffic contained within a software process. The Extension takes over execution of the BREAK instruction, stalling the processor and initiating a debug session. The software developer can interface with the processor using the GDB debug console to suspend, continue and monitor memory and registers. Additionally, this version provides two hardware watchpoints for monitoring the values of variables in the software program. eBug is described in [107].

This extension is useful to learn how to monitor and override instructions in the TISA, how to read and write to the register file, how to read and write to memory, how to stall and resume the processor, and how to implement a simple communication protocol.

eBUG HW 8

The eBUG HW 8 is a software debugging solution implemented using the eMIPS Extension hardware. The Extension monitors the execution flow, and the register and memory traffic contained within a software process. The Extension takes over execution of the BREAK instruction, stalling the processor and initiating a debug session. The software developer can interface with the processor using the GDB debug console to suspend, continue and monitor memory and registers. Additionally, this version provides eight hardware watchpoints for monitoring the values of variables in the software program. eBug is described in [107].

This extension is useful to learn how to monitor and override instructions in the TISA, how to read and write to the register file, how to read and write to memory, how to stall and resume the processor, and how to implement a simple communication protocol. This is the preferred solution for debugging software on the eMIPS system.

Appendix B: Build Scripts

The following are the commands for building the eMIPS Processor system using the Xilinx Partial Reconfiguration Flow. These commands may be copied into batch files and modified to ease the build process. The following build scripts are included in the release.

1. eMIPS_PR_Top.bat – Script for building the Top level module
2. eMIPS_PR_TISA.bat – Script for building the TISA (Fixed Region)
3. eMIPS_PR_Extension0_mmldiv64.bat – Script for building the Extension0 (Reconfigurable Region)
4. eMIPS_PR_Extension0_mmldiv64_merge.bat – Script for merging the TISA and Extension0 (Generates full and partial bit files)

eMIPS_PR_Top.bat

```
ngdbuild -uc ..\..\Sources\Constraints\mips_fp.ucf -sd ..\..\Xilinx_PR_Bus_Macros\V4\NMC -modular initial
..\..\Synthesis\Top\MIPSPL_FPGA3.ngc
```

eMIPS_PR_TISA.bat

```
ngdbuild -uc ..\..\Sources\Constraints\mips_fp.ucf -sd ..\..\Xilinx_PR_Bus_Macros\V4\NMC -sd
..\..\Synthesis\TISA -modular initial ..\..\Synthesis\Top\MIPSPL_FPGA3.ngc
map -uc ..\..\Sources\Constraints\mips_fp.ucf -ol high -pr b -timing -xe c MIPSPL_FPGA3.ngd
par -w -uc ..\..\Sources\Constraints\mips_fp.ucf -ol high -pl high -rl high -xe c MIPSPL_FPGA3.ncd
MIPSPL_FPGA3_BASE_ROUTED.ncd
```

eMIPS_PR_Extension0_mmldiv64.bat

```
copy ..\..\TISA\static.used arcs.exclude
ngdbuild -uc ..\..\Sources\Constraints\mips_fp.ucf -sd ..\..\Xilinx_PR_Bus_Macros\V4\NMC -sd
..\..\Synthesis\Extension0_mmldiv64 -modular module -active extension0
..\..\Synthesis\Top\MIPSPL_FPGA3.ngc
map -uc ..\..\Sources\Constraints\mips_fp.ucf -ol high -pr b -timing -xe c MIPSPL_FPGA3.ngd
par -w -uc ..\..\Sources\Constraints\mips_fp.ucf -ol high -pl high -rl high -xe c MIPSPL_FPGA3.ncd
MIPSPL_FPGA3_EXT0_MMLDIV64_ROUTED.ncd
```

eMIPS_PR_Extension0_mmldiv64_merge.bat

```
copy ..\..\TISA\MIPSPL_FPGA3_BASE_ROUTED.ncd MIPSPL_FPGA3_BASE_ROUTED.ncd
copy ..\..\Extension0_mmldiv64\MIPSPL_FPGA3_EXT0_MMLDIV64_ROUTED.ncd
MIPSPL_FPGA3_EXT0_MMLDIV64_ROUTED.ncd
call PR_VERIFYDESIGN MIPSPL_FPGA3_BASE_ROUTED.ncd MIPSPL_FPGA3_EXT0_MMLDIV64_ROUTED.ncd
call PR_ASSEMBLE MIPSPL_FPGA3_BASE_ROUTED.ncd MIPSPL_FPGA3_EXT0_MMLDIV64_ROUTED.ncd
```