CoDeveloper Platform Support Package

Pico Computing M501 PSP User Guide Version 1.0.2

Impulse Accelerated Technologies, Inc.

www.ImpulseAccelerated.com

1.0 Table of Contents

2.0 Table Of Figures

3.0 Revision History

Date	Version	Description	Author
7/3/2012	1.0	Initial Creation	Shaumil Dave
7/9/2012	1.0.1	Added notes on setting PICOBASE and XILINX_BASE environment variables	Ed Trexel
7/10/2012	1.0.2	Added note on floating point patch	Ed Trexel

4.0 Overview

This user guide covers the CoDeveloper Platform Support Package (PSP) for the Pico Computing M501 module (referred hereon simply as "M501"). Highlights for this PSP include:

- Automatic creation of a complete ready-to-build Xilinx ISE project upon exporting hardware for creating FPGA binary '.bit' file built via GUI.
- Application executable built via Microsoft Visual Studio.
- Loading of the FPGA on the M501 via host CPU over PCIe .

4.1. Hardware Block Diagram

4.1.1. Pico Computing Block Diagram

Below is a diagram of the M50X platform:

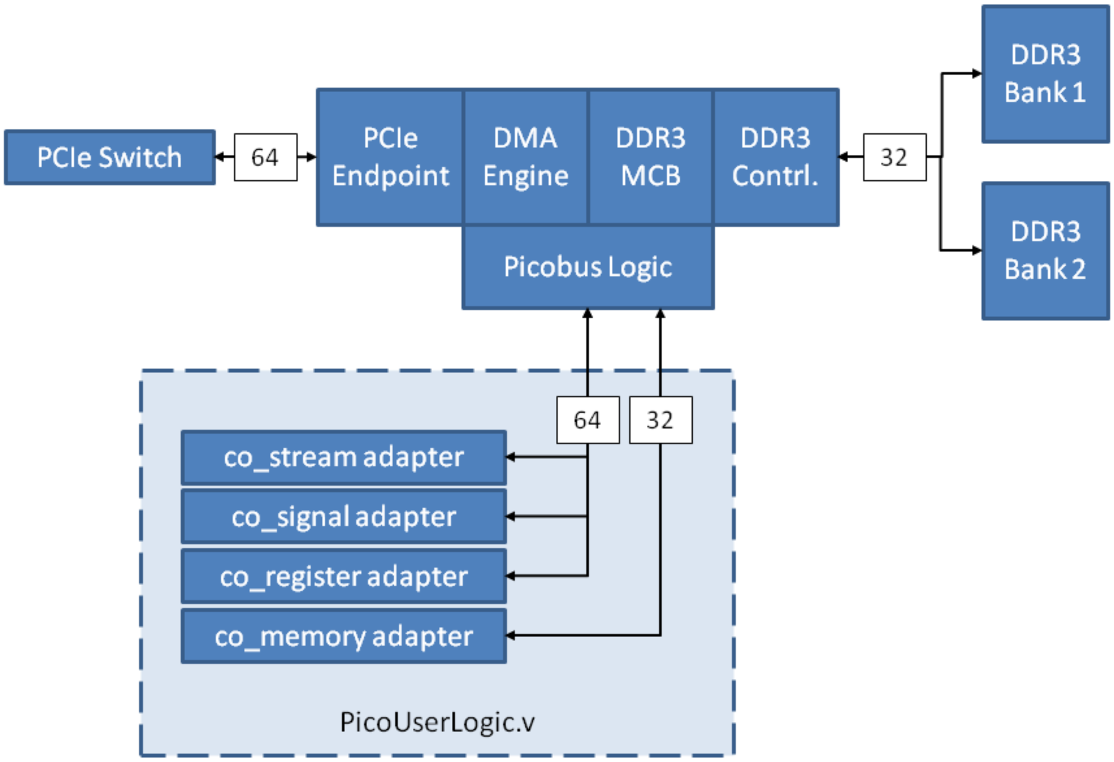


Figure 1 – Firmware Architecture (From Pico M50X Series Platform Supprt Package Users Guide)

The only elements of the architecture the user has control of are those contained inside the Verilog module “PicoUserLogic.v”. All of the logic generated by Impulse CoDeveloper and any external HDL modules must be instantiated inside this top level user module

4.2. Software Directory Structure

Below is an outline of the software directory structure present in the PSP. The software used on the development system is exported from CoDeveloper. The directory names are user defined and are defined in this instance as “export_sw”. The CoDeveloper tool will place the software application files in this directory to be compiled and executed with a compiler, like Visual Studio. Directories are indicated by **bold letters**.

- **Examples**
  - **M5XX**
    - **Passthrough**
      - filter_in.dat
      - passthrough.icProj
      - passthrough_hw.c
      - passthrough_sw.c
      - passthrough.h
        - **export_sw**
          - co.h
          - co_if_sim.h
          - co_init.c
          - co_math.h
          - co+process.c
          - co_stream.c
          - co_types.c
          - cosim_log.h
          - passthrough.h
          - passthrough_arch.vcxproj
          - passthrough_sw.c
          - Pico_channel_wrapper.cpp
          - pico_channel_wrapper.h

4.3. Before Getting Started: Read This First

Before getting started, please ensure that you have obtained and installed all the necessary software tools, additional files, and hardware as described below.

4.3.1. Required Software Tools:

- Impulse CoDeveloper v3.70.d.11 or newer
- NOTE: The use of floating point for Virtex-6 and newer devices requires the use of Xilinx's v5.0 CORE Generator cores which is supported via a patch to CoDeveloper made available via the 'XilinxFPv5BetaPatch' link under the supplied Pico PSP link.
- Xilinx ISE 12.4 (exactly, not newer)
- Windows 7 operating system
- Visual Studio Express C++ (64 bit configuration)
- Pico Installer 6.2.1.3 or newer

4.3.2. Additional Required Files

The following Examples files are not included with the installation of the software tools and are required for development using this PSP. They include the CoDeveloper project file (*.icProj), associated design files (*_hw.c, *_sw.c, *.h), and data files (*.dat) used as stimulus. A link to download the M5xx PSP and Examples files should have already been provided, if not please email support@impulsec.com to request one.

```
Examples
        M5XX
                Passthrough
                        filter_in.dat
                        passthrough.icProj
                        passthrough_hw.c
                        passthrough_sw.c
                        passthrough.h
```

4.3.3. Required Hardware

The following hardware is required for development using this PSP:

- M501 x16 full length full height PCIe FPGA Development Board.
- Windows 7 64-bit OS based development PC for running all tools – Recommended: 1TB disk space available for tools installation and 12GB RAM.
- Intel Motherboard with x58 chipset and available x16 (physical) Gen 2 PCIe express slot. Consideration for your PCIe video card must be given if it is a x16 video card.

The motherboard must accommodate (not a shared resource) independent x16 Gen 2 PCIe slots.

- Available 12 volt PCIe power supply for the M501 FPGA development board. Consideration for your video card must be given if it also requires a separate 12 volt power connection.

4.3.4. Software Installation

The following steps cover all tool and any additional files that need to be installed:

Development PC:

1) Xilinx ISE 12.4 – Please see vendor supplied documentation for licensing and installation.

   NOTE: The PSP requires that the environment variable ‘XILINX_BASE’ be set to the top directory of where ISE is installed, typically something like “C:\Xilinx\12.4”

2) Impulse CoDeveloper – Download the latest version 3.x and installation notes from: http://www.impulseaccelerated.com/ReleaseFiles/

   NOTE: The use of floating point for Virtex-6 and newer devices requires the use of Xilinx’s v5.0 CORE Generator cores which is supported via a patch to CoDeveloper made available via the ‘XilinxFPv5BetaPatch’ link under the supplied Pico PSP link.

   a. (optional)Installation of floating point support is via unzipping the .zip file (password: impulsefpv5beta) into ‘**Impulse**’ after each CoDeveloper installation. Please see enclosed README file for specific notes on the patch.

3) Add the Pico M5XX PSP to the CoDeveloper installation .

   a. Copy the supplied “**Architectures**” directory to “**Impulse****CoDeveloper3**\”.

4) Copy the supplied “**Examples**” directory to a working directory on the development PC for access to the pre-built example files.

5) Run the Pico Installer

   NOTE: The PSP requires that the environment variable ‘PICOBASE’ be set to the top directory of where the Pico software is installed, typically something like “C:\Pico\6.2.1.3”

4.4. Target System Setup

4.4.1. Install Pico M501 driver

After the Pico M501 and EX-xxx carrier are installed, the user must install the drivers.

4.4.1.1. Follow the “Windows_Getting_Started_Gude.pdf” to install the driver. The current installer is “PicoInstaller_6.2.1.3.exe”.

4.4.2. Install Visual Studio Express C++ in 64 bit configuration

Please see the “Installation_Microsoft_VSExpress2010and64bitSDK.pdf” for installation instructions and how to configure an exported VS C++ project.

4.4.2.1. Install Visual Studio Express C++ 2010

4.4.2.2. Install Window 7 SDK

5.0 Passthrough Example and Tutorial

“Passthrough” is provided as an example that may be used for quickly creating user applications and for the purpose of a tutorial showing the steps involved to go from an Impulse C application in CoDeveloper all the way through to a Xilinx ISE 12.4-compiled FPGA binary and target application executable. The base files required for recreating the example using this tutorial are provided within the Impulse supplied examples which needs to be copied to a working directory on the development PC in order to run the tutorial.

> NOTE: Ensure there are no spaces (‘ ‘) in the directory path chosen to avoid potential path issues with any of the tools.

The Passthrough example’s hardware process is “Passthrough()”, located in the source file “Passthrough_hw.c”. It performs the following operations:

1) Read value from co_stream “input_stream”
2) Write value to co_stream to “output_stream”

   NOTE: The hardware code runs continuously

5.1. Prerequisites

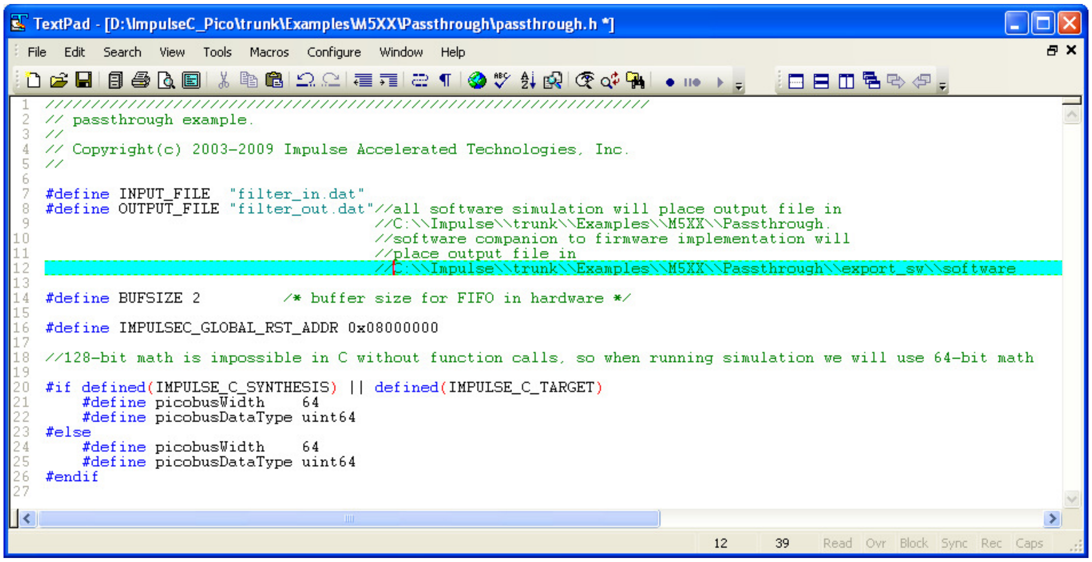
The tutorial in this Platform Support Package assumes that you have read and understand the introductory sections of the CoDeveloper User's Guide, installed with CoDeveloper and accessed from the Help menu. In particular, you should take the time to go through the tutorials provided with CoDeveloper so you have a good understanding of the front-end design flow including both desktop software simulation and hardware compilation.

5.2. CoDeveloper Project Files

The Passthrough example CoDeveloper project is made up of the following files:

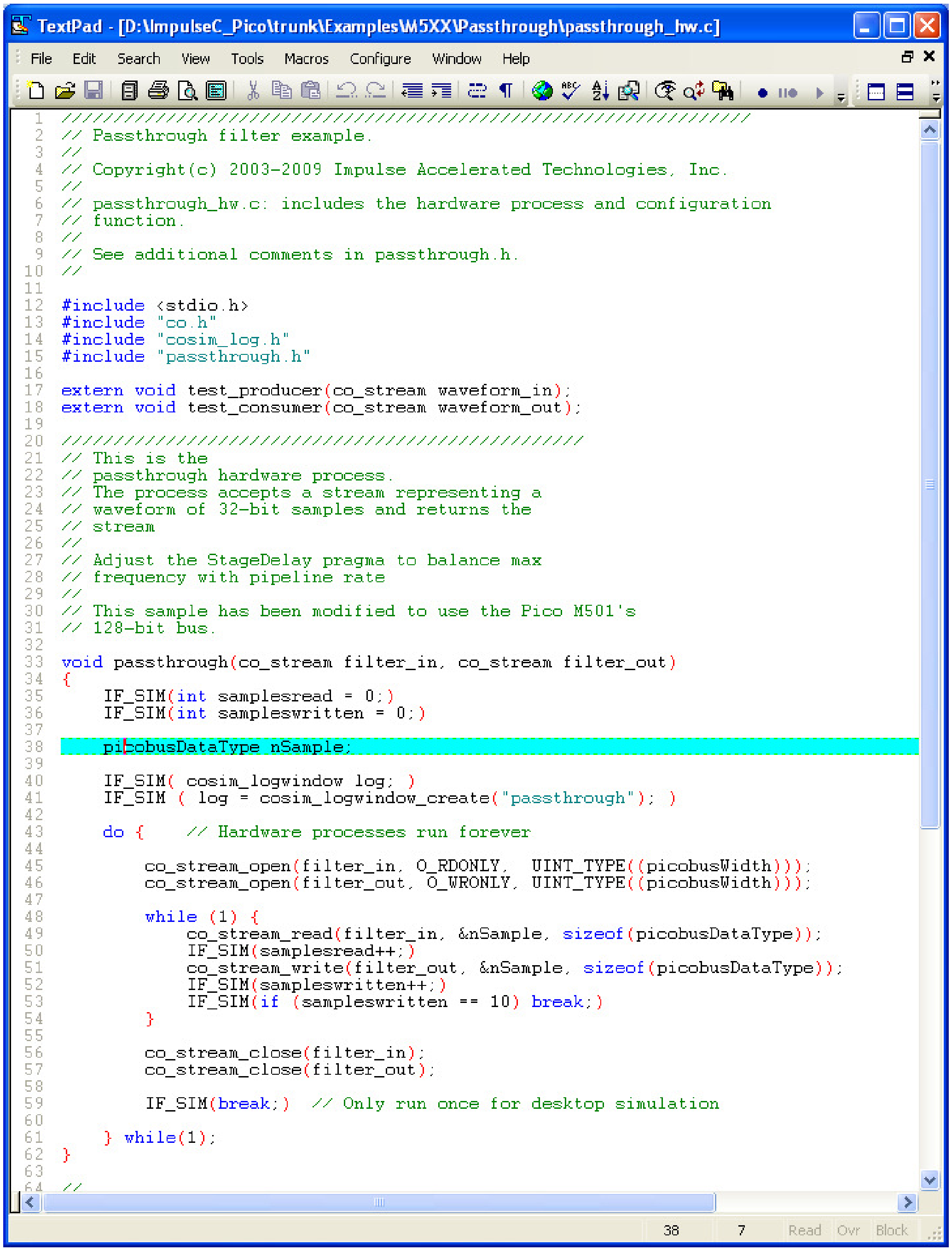
- Passthrough.icProj – CoDeveloper project file
- Passthrough_hw.c – Source code for hardware process
- Passthrough_sw.c – Source code for software processes
- Passthrough.h – Header file that defines the width of the stream

When you define the width of the steam, you must make the changes in the header file as well as the in the passthrough_hw.c file. The default example defines the steam to be the maximum width of 64 bit data bus.


TextPad - [D:\ImpulseC_Pico\trunk\Examples\M5XX\Passthrough\passthrough.h *]
File
Edit
Search
View
Tools
Macros
Configure
Window
Help
12
39
Read
Ovr
Block
Sync
Rec
Caps

```
//////////////////////////////////////////////////////////////////
// passthrough example.
//
// Copyright(c) 2003-2009 Impulse Accelerated Technologies, Inc.
//

#define INPUT_FILE  "filter_in.dat"
#define OUTPUT_FILE "filter_out.dat"//all software simulation will place output file in
                                    //C:\\Impulse\\trunk\\Examples\\M5XX\\Passthrough.
                                    //software companion to firmware implementation will
                                    //place output file in
                                    //C:\\Impulse\\trunk\\Examples\\M5XX\\Passthrough\\export_sw\\software

#define BUFSIZE 2         /* buffer size for FIFO in hardware */

#define IMPULSEC_GLOBAL_RST_ADDR 0x08000000

//128-bit math is impossible in C without function calls, so when running simulation we will use 64-bit math

#if defined(IMPULSE_C_SYNTHESIS) || defined(IMPULSE_C_TARGET)
    #define picobusWidth    64
    #define picobusDataType uint64
#else
    #define picobusWidth    64
    #define picobusDataType uint64
#endif

```

Figure 2 - Impulse C Header File with 64 bit co_stream

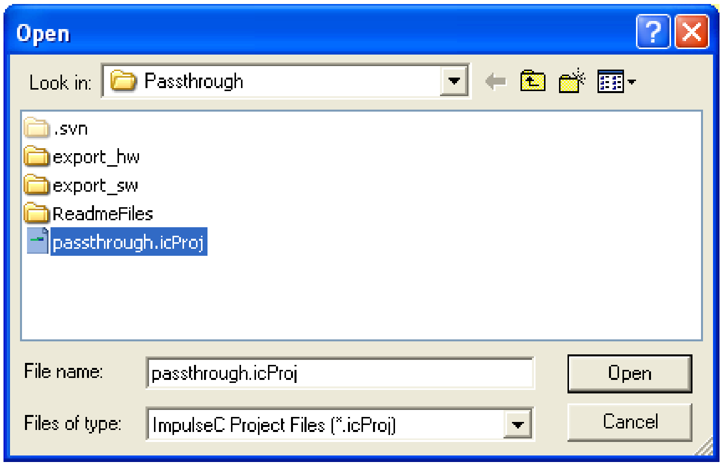

TextPad - [D:\ImpulseC_Pico\trunk\Examples\M5XX\Passthrough\passthrough_hw.c]
File
Edit
Search
View
Tools
Macros
Configure
Window
Help
38
7
Read
Ovr
Block

```
//////////////////////////////////////////////////////////////////
// Passthrough filter example.
//
// Copyright(c) 2003-2009 Impulse Accelerated Technologies, Inc.
//
// passthrough_hw.c: includes the hardware process and configuration
// function.
//
// See additional comments in passthrough.h.
//

#include <stdio.h>
#include "co.h"
#include "cosim_log.h"
#include "passthrough.h"

extern void test_producer(co_stream waveform_in);
extern void test_consumer(co_stream waveform_out);

//////////////////////////////////////////////////
// This is the
// passthrough hardware process.
// The process accepts a stream representing a
// waveform of 32-bit samples and returns the
// stream
//
// Adjust the StageDelay pragma to balance max
// frequency with pipeline rate
//
// This sample has been modified to use the Pico M501's
// 128-bit bus.

void passthrough(co_stream filter_in, co_stream filter_out)
{
    IF_SIM(int samplesread = 0;)
    IF_SIM(int sampleswritten = 0;)

    picobusDataType nSample;

    IF_SIM( cosim_logwindow log; )
    IF_SIM ( log = cosim_logwindow_create("passthrough"); )

    do {    // Hardware processes run forever

        co_stream_open(filter_in, O_RDONLY,  UINT_TYPE((picobusWidth)));
        co_stream_open(filter_out, O_WRONLY, UINT_TYPE((picobusWidth)));

        while (1) {
            co_stream_read(filter_in, &nSample, sizeof(picobusDataType));
            IF_SIM(samplesread++;)
            co_stream_write(filter_out, &nSample, sizeof(picobusDataType));
            IF_SIM(sampleswritten++;)
            IF_SIM(if (sampleswritten == 10) break;)
        }

        co_stream_close(filter_in);
        co_stream_close(filter_out);

        IF_SIM(break;)  // Only run once for desktop simulation

    } while(1);
}

//
```

Figure 3 - ImpulseC Hardware File with 64 bit co_stream

5.3. Opening Project

Open the CoDeveloper project file ‘Passthrough.icProj’ by selecting and pressing ‘Enter’ or by double-clicking it:



File name: passthrough.icProj

Files of type: ImpulseC Project Files (*.icProj)

Figure 4 - Opening a project in CoDeveloper

5.4. Building Desktop Simulation Executable

Build the desktop software simulation executable via the “Project” menu:

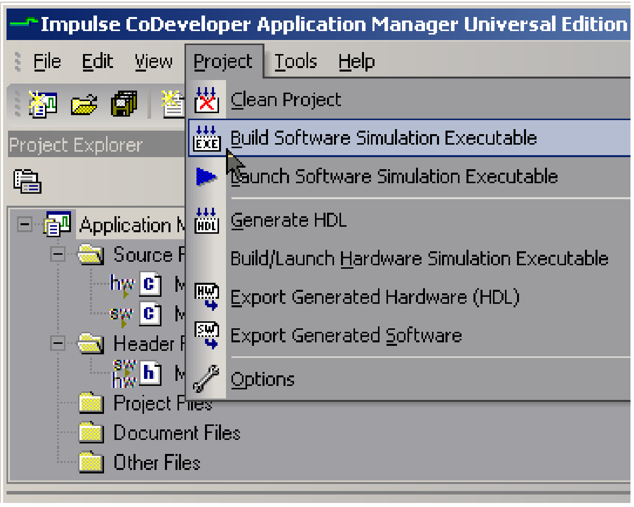

Impulse CoDeveloper Application Manager Universal Edition
File
Edit
View
Project
Tools
Help
Project Explorer
Application M
Source F
hw
sw
Header F
sw
hw
Project Files
Document Files
Other Files
Clean Project
Build Software Simulation Executable
Launch Software Simulation Executable
Generate HDL
Build/Launch Hardware Simulation Executable
Export Generated Hardware (HDL)
Export Generated Software
Options

Figure 5 - Build Simulation Desktop in CoDeveloper using pull-down menu

Or via toolbar:

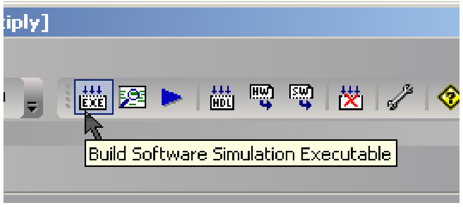
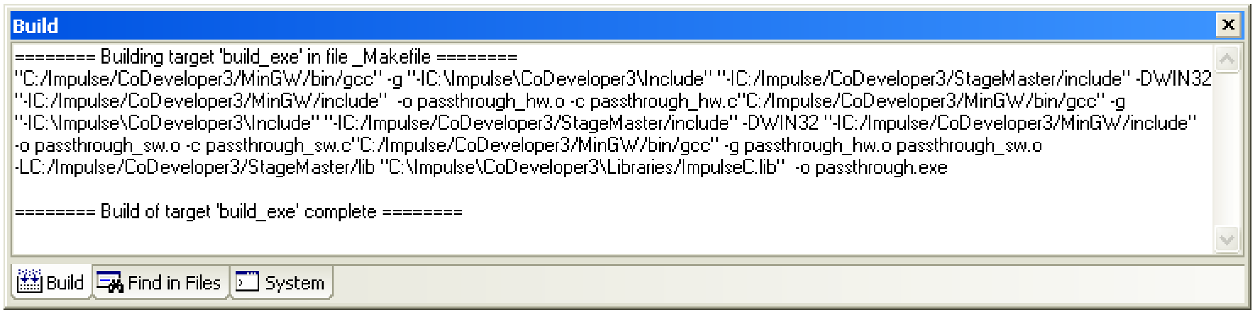


Figure 6 - Build Simulation Desktop in CoDeveloper using toolbar icon

Note the compiler output in the CoDeveloper IDE “Build” window:



```
======== Building target 'build_exe' in file _Makefile ========
"C:/Impulse/CoDeveloper3/MinGW/bin/gcc" -g "-IC:\Impulse\CoDeveloper3\Include" "-IC:/Impulse/CoDeveloper3/StageMaster/include" -DWIN32
"-IC:/Impulse/CoDeveloper3/MinGW/include"  -o passthrough_hw.o -c passthrough_hw.c"C:/Impulse/CoDeveloper3/MinGW/bin/gcc" -g
"-IC:\Impulse\CoDeveloper3\Include" "-IC:/Impulse/CoDeveloper3/StageMaster/include" -DWIN32 "-IC:/Impulse/CoDeveloper3/MinGW/include"
-o passthrough_sw.o -c passthrough_sw.c"C:/Impulse/CoDeveloper3/MinGW/bin/gcc" -g passthrough_hw.o passthrough_sw.o
-LC:/Impulse/CoDeveloper3/StageMaster/lib "C:\Impulse\CoDeveloper3\Libraries/ImpulseC.lib"  -o passthrough.exe

======== Build of target 'build_exe' complete ========
```

Figure 7 - Output within the CoDeveloper IDE build window

5.5. Running Desktop Simulation Executable

Launch the desktop software simulation executable via "Project" menu:

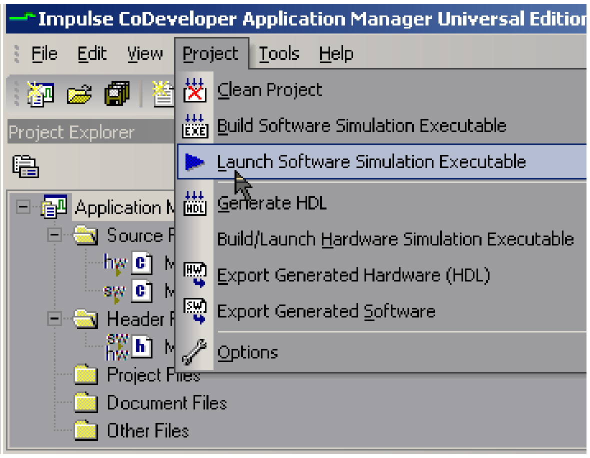

Impulse CoDeveloper Application Manager Universal Edition
File
Edit
View
Project
Tools
Help
Project Explorer
Clean Project
Build Software Simulation Executable
Launch Software Simulation Executable
Generate HDL
Build/Launch Hardware Simulation Executable
Export Generated Hardware (HDL)
Export Generated Software
Options
Application M
Source F
Header F
Project Files
Document Files
Other Files

Figure 8 - Launch software simulation window using pull-down menu

Or via toolbar:

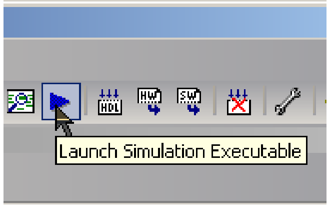
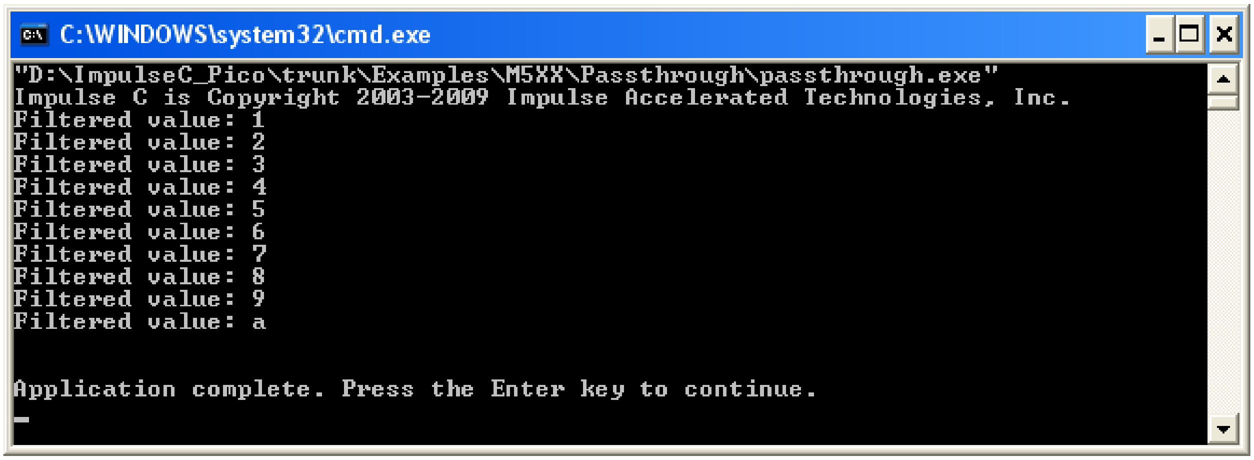


Figure 9 - Launch software simulation using toolbar icon

A command window will pop up in which the desktop simulation executable runs. Press "Enter" to exit:



```
"D:\ImpulseC_Pico\trunk\Examples\M5XX\Passthrough\passthrough.exe"
Impulse C is Copyright 2003-2009 Impulse Accelerated Technologies, Inc.
Filtered value: 1
Filtered value: 2
Filtered value: 3
Filtered value: 4
Filtered value: 5
Filtered value: 6
Filtered value: 7
Filtered value: 8
Filtered value: 9
Filtered value: a


Application complete. Press the Enter key to continue.
_
```

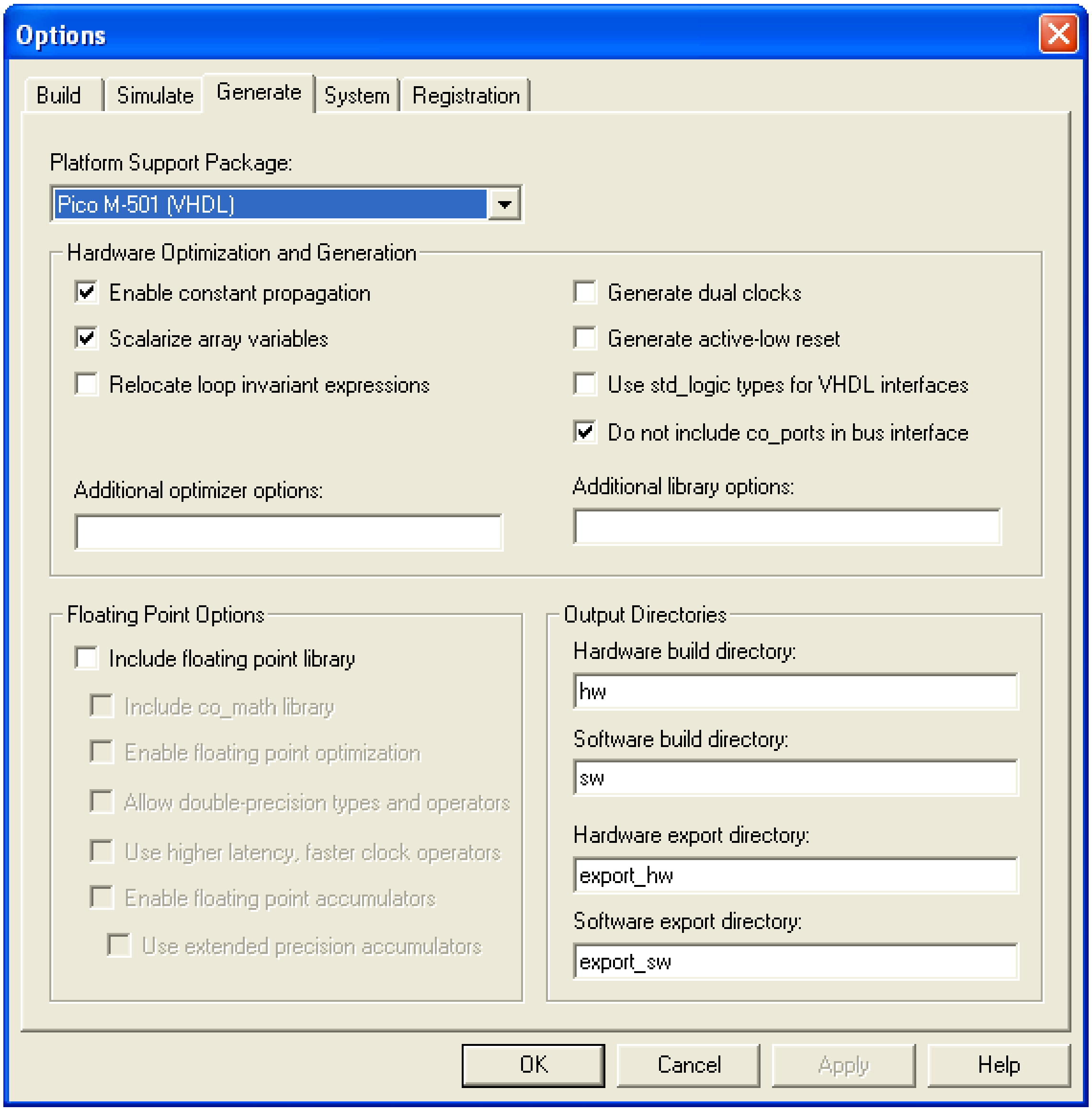
Figure 10 - Pop-up window during desktop simulation

5.6. Project Setup Before Hardware/Software Generation and Export

Settings within the CoDeveloper IDE necessary for generating and exporting both hardware and software using this PSP are summarized below:

- Platform Support Package: “Pico M-501 (VHDL)”
- Hardware export directory: <user hardware export directory>
- Software export directory: <user software export directory>
- Unsupported settings include:
    - Generate dual clocks (must be unchecked)
    - Active-low reset (must be unchecked)
    - Include floating point library (must be unchecked)

An example of these settings as it appears in the Passthrough example:



Platform Support Package: Pico M-501 (VHDL)

Hardware Optimization and Generation

☑ Enable constant propagation

☑ Scalarize array variables

□ Relocate loop invariant expressions

□ Generate dual clocks

□ Generate active-low reset

□ Use std_logic types for VHDL interfaces

☑ Do not include co_ports in bus interface

Additional optimizer options:

Additional library options:

Floating Point Options

□ Include floating point library

□ Include co_math library

□ Enable floating point optimization

□ Allow double-precision types and operators

□ Use higher latency, faster clock operators

□ Enable floating point accumulators

□ Use extended precision accumulators

Output Directories

Hardware build directory: hw

Software build directory: sw

Hardware export directory: export_hw

Software export directory: export_sw

Figure 11 - Project setup to pick Platform Support Package

5.7. Generating Hardware

Generate hardware via “Project” menu:

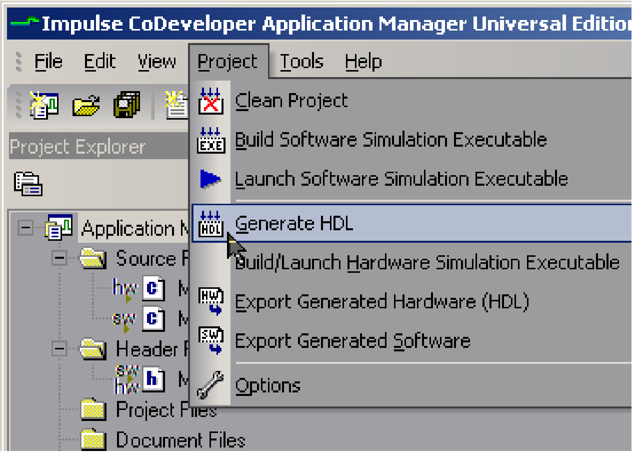

Impulse CoDeveloper Application Manager Universal Edition
File
Edit
View
Project
Tools
Help
Project Explorer
Application M
Source F
hw
sw
Header F
Project Files
Document Files
Clean Project
Build Software Simulation Executable
Launch Software Simulation Executable
Generate HDL
Build/Launch Hardware Simulation Executable
Export Generated Hardware (HDL)
Export Generated Software
Options

Figure 12 - Generate HDL using pull-down menu

Or via toolbar:

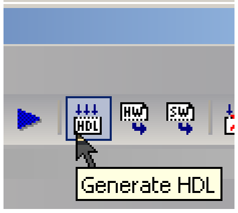
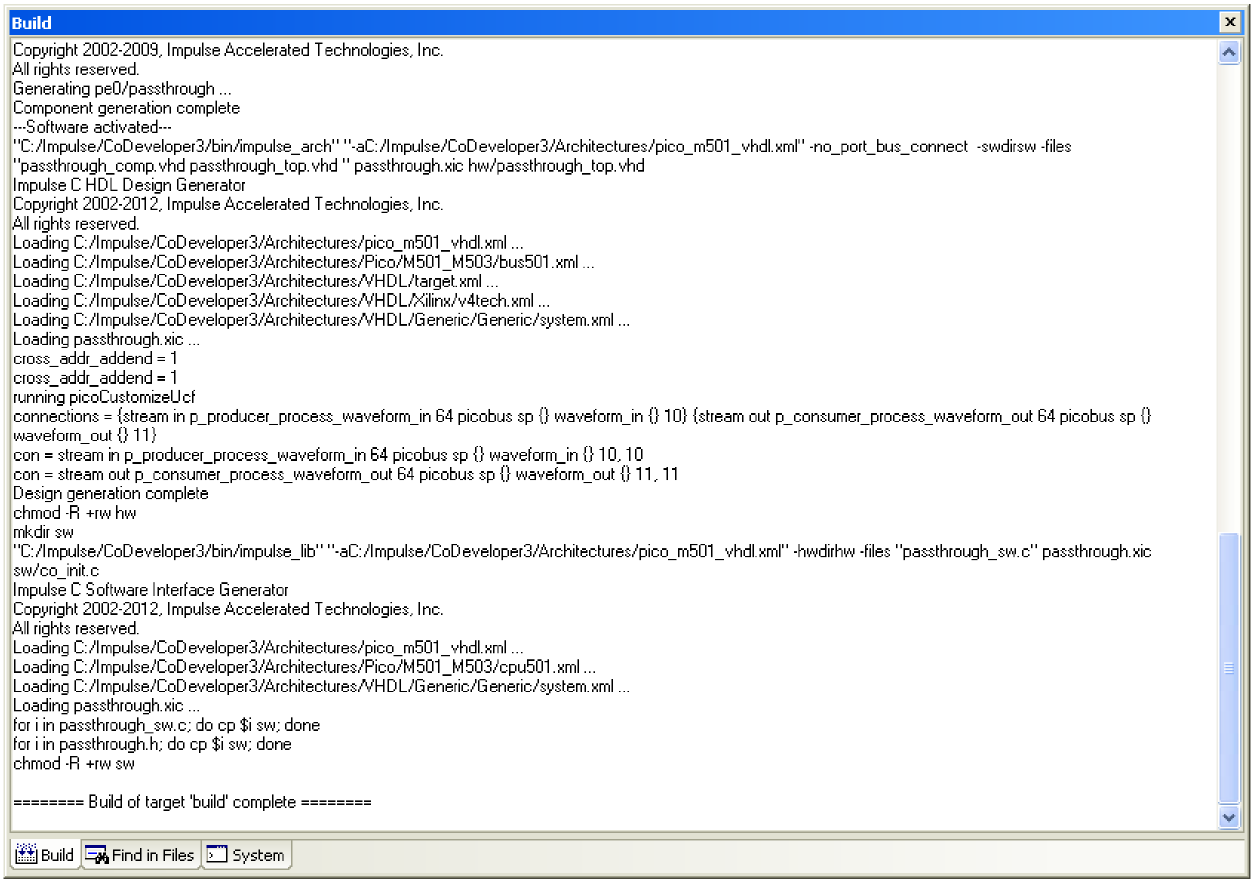


Figure 13 - Generate HDL using toolbar icon

Final results will appear in the directory specified during project setup in “Hardware build directory”. Note the final output in the CoDeveloper IDE’s “Build” window:



```
Copyright 2002-2009, Impulse Accelerated Technologies, Inc.
All rights reserved.
Generating pe0/passthrough ...
Component generation complete
---Software activated---
"C:/Impulse/CoDeveloper3/bin/impulse_arch" "-aC:/Impulse/CoDeveloper3/Architectures/pico_m501_vhdl.xml" -no_port_bus_connect  -swdirsw -files
"passthrough_comp.vhd passthrough_top.vhd " passthrough.xic hw/passthrough_top.vhd
Impulse C HDL Design Generator
Copyright 2002-2012, Impulse Accelerated Technologies, Inc.
All rights reserved.
Loading C:/Impulse/CoDeveloper3/Architectures/pico_m501_vhdl.xml ...
Loading C:/Impulse/CoDeveloper3/Architectures/Pico/M501_M503/bus501.xml ...
Loading C:/Impulse/CoDeveloper3/Architectures/VHDL/target.xml ...
Loading C:/Impulse/CoDeveloper3/Architectures/VHDL/Xilinx/v4tech.xml ...
Loading C:/Impulse/CoDeveloper3/Architectures/VHDL/Generic/Generic/system.xml ...
Loading passthrough.xic ...
cross_addr_addend = 1
cross_addr_addend = 1
running picoCustomizeUcf
connections = {stream in p_producer_process_waveform_in 64 picobus sp {} waveform_in {} 10} {stream out p_consumer_process_waveform_out 64 picobus sp {}
waveform_out {} 11}
con = stream in p_producer_process_waveform_in 64 picobus sp {} waveform_in {} 10, 10
con = stream out p_consumer_process_waveform_out 64 picobus sp {} waveform_out {} 11, 11
Design generation complete
chmod -R +rw hw
mkdir sw
"C:/Impulse/CoDeveloper3/bin/impulse_lib" "-aC:/Impulse/CoDeveloper3/Architectures/pico_m501_vhdl.xml" -hwdirhw -files "passthrough_sw.c" passthrough.xic
sw/co_init.c
Impulse C Software Interface Generator
Copyright 2002-2012, Impulse Accelerated Technologies, Inc.
All rights reserved.
Loading C:/Impulse/CoDeveloper3/Architectures/pico_m501_vhdl.xml ...
Loading C:/Impulse/CoDeveloper3/Architectures/Pico/M501_M503/cpu501.xml ...
Loading C:/Impulse/CoDeveloper3/Architectures/VHDL/Generic/Generic/system.xml ...
Loading passthrough.xic ...
for i in passthrough_sw.c; do cp $i sw; done
for i in passthrough.h; do cp $i sw; done
chmod -R +rw sw

======== Build of target 'build' complete ========
```

Figure 14 - Build window output

5.8. Exporting Hardware

Export hardware via “Project” menu:

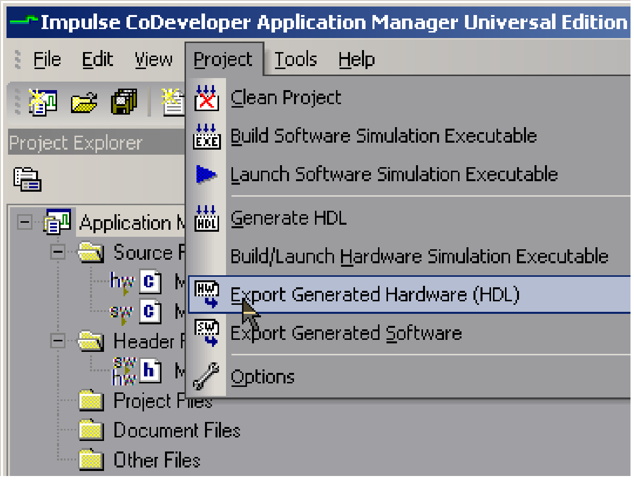

Impulse CoDeveloper Application Manager Universal Edition
File Edit View Project Tools Help
Clean Project
Build Software Simulation Executable
Launch Software Simulation Executable
Generate HDL
Build/Launch Hardware Simulation Executable
Export Generated Hardware (HDL)
Export Generated Software
Options
Project Explorer
Application M
Source F
hw
sw
Header F
sw
hw
Project Files
Document Files
Other Files

Figure 15 - Export Generated Hardware (HDL) using pull-down menu

Or via toolbar:

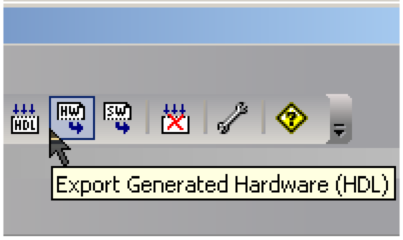
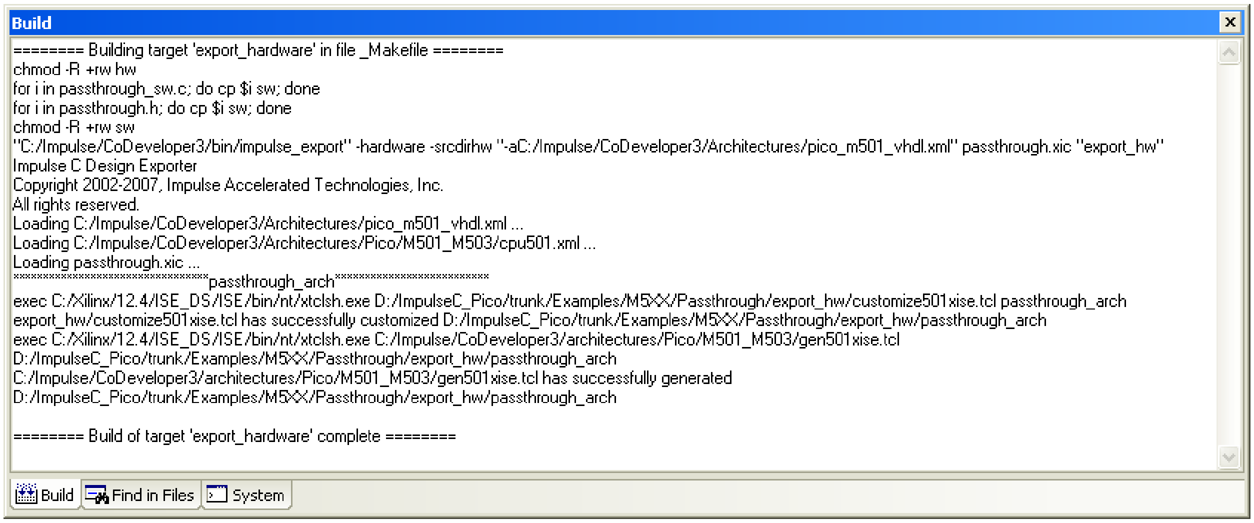


Figure 16 - Export Generated Hardware (HDL) using toolbar icon

Final results will appear in the directory specified during project setup in “Hardware export directory”. Note the final output in the CoDeveloper IDE’s “Build” window:

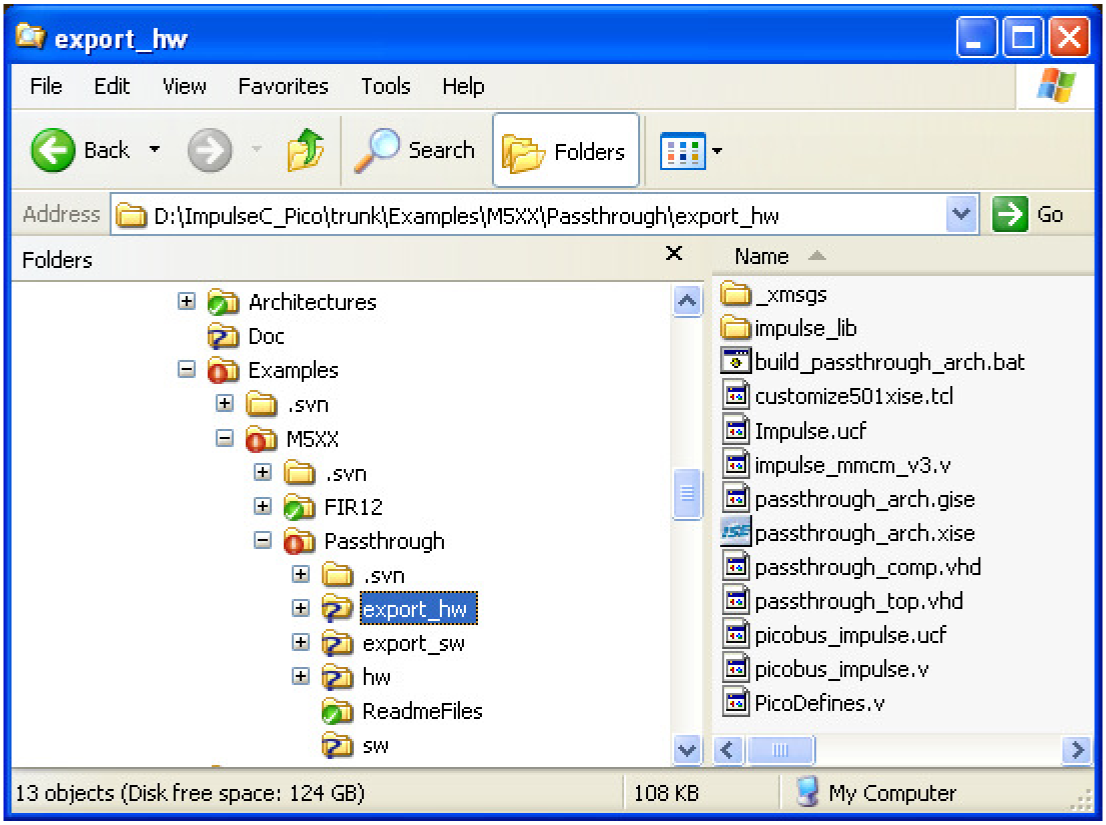


```
======== Building target 'export_hardware' in file _Makefile ========
chmod -R +rw hw
for i in passthrough_sw.c; do cp $i sw; done
for i in passthrough.h; do cp $i sw; done
chmod -R +rw sw
"C:/Impulse/CoDeveloper3/bin/impulse_export" -hardware -srcdirhw "-aC:/Impulse/CoDeveloper3/Architectures/pico_m501_vhdl.xml" passthrough.xic "export_hw"
Impulse C Design Exporter
Copyright 2002-2007, Impulse Accelerated Technologies, Inc.
All rights reserved.
Loading C:/Impulse/CoDeveloper3/Architectures/pico_m501_vhdl.xml ...
Loading C:/Impulse/CoDeveloper3/Architectures/Pico/M501_M503/cpu501.xml ...
Loading passthrough.xic ...
****************************************passthrough_arch*****************************
exec C:/Xilinx/12.4/ISE_DS/ISE/bin/nt/xtclsh.exe D:/ImpulseC_Pico/trunk/Examples/M5XX/Passthrough/export_hw/customize501xise.tcl passthrough_arch
export_hw/customize501xise.tcl has successfully customized D:/ImpulseC_Pico/trunk/Examples/M5XX/Passthrough/export_hw/passthrough_arch
exec C:/Xilinx/12.4/ISE_DS/ISE/bin/nt/xtclsh.exe C:/Impulse/CoDeveloper3/architectures/Pico/M501_M503/gen501xise.tcl
D:/ImpulseC_Pico/trunk/Examples/M5XX/Passthrough/export_hw/passthrough_arch
C:/Impulse/CoDeveloper3/architectures/Pico/M501_M503/gen501xise.tcl has successfully generated
D:/ImpulseC_Pico/trunk/Examples/M5XX/Passthrough/export_hw/passthrough_arch

======== Build of target 'export_hardware' complete ========
```

Figure 17 - Build window output

5.9. Compiling FPGA in Xilinx ISE 12.4

After exporting hardware, under the specified hardware export directory will be a directory structure that includes all necessary files for building the FPGA binary. In the top directory there will be the batch file "build_passthrough_arch.bat" used to automatically run Xilinx ISE 12.4 to create the necessary .bit file used to program the M501 FPGA (select then "Enter" or double-click).



Address: D:\ImpulseC_Pico\trunk\Examples\M5XX\Passthrough\export_hw

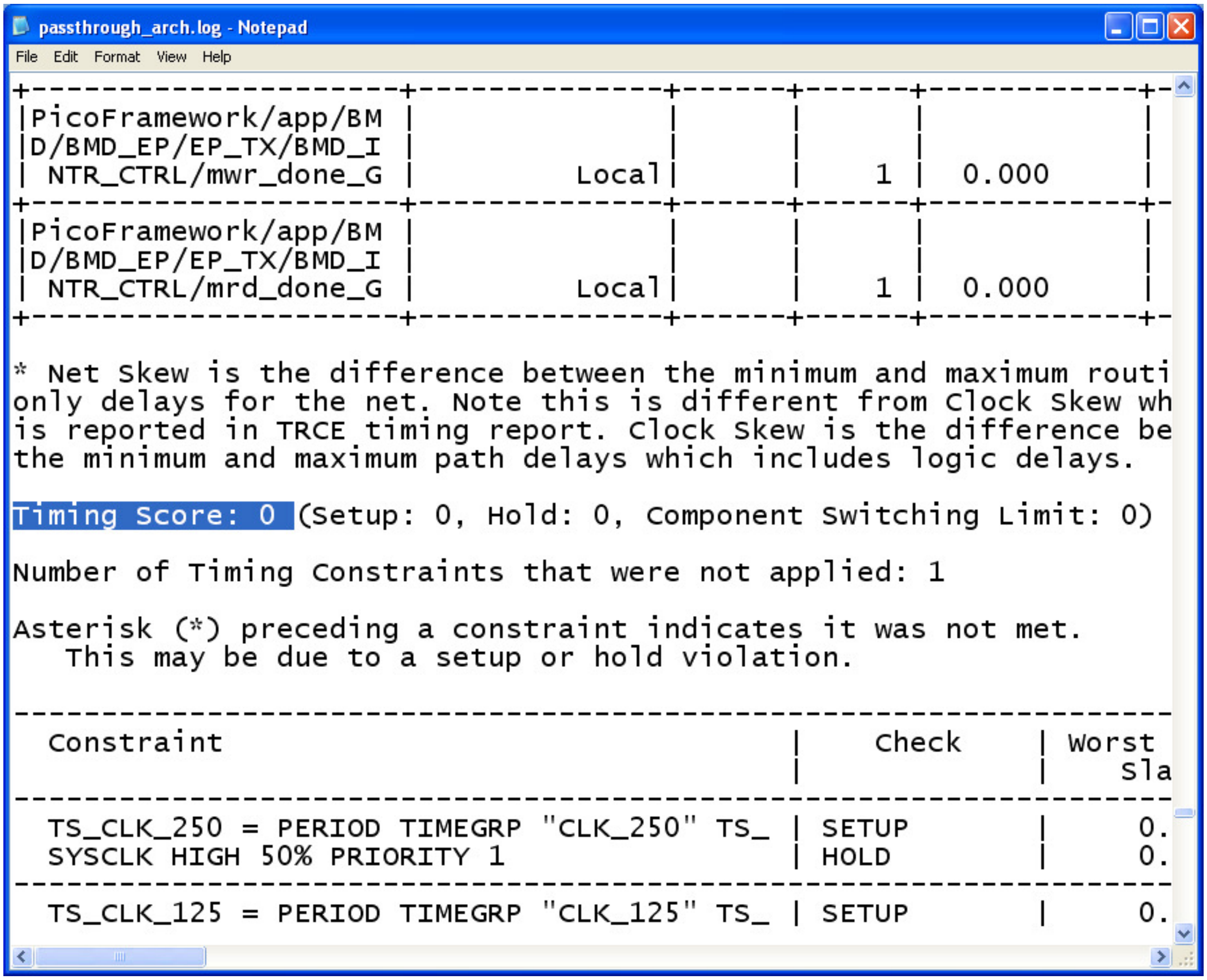
```
⊞ Architectures
  Doc
⊟ Examples
   ⊞ .svn
   ⊟ M5XX
      ⊞ .svn
      ⊞ FIR12
      ⊟ Passthrough
         ⊞ .svn
         ⊞ export_hw
         ⊞ export_sw
         ⊞ hw
           ReadmeFiles
           sw
```

_xmsgs
impulse_lib
build_passthrough_arch.bat
customize501xise.tcl
Impulse.ucf
impulse_mmcm_v3.v
passthrough_arch.gise
passthrough_arch.xise
passthrough_comp.vhd
passthrough_top.vhd
picobus_impulse.ucf
picobus_impulse.v
PicoDefines.v

13 objects (Disk free space: 124 GB) 108 KB My Computer

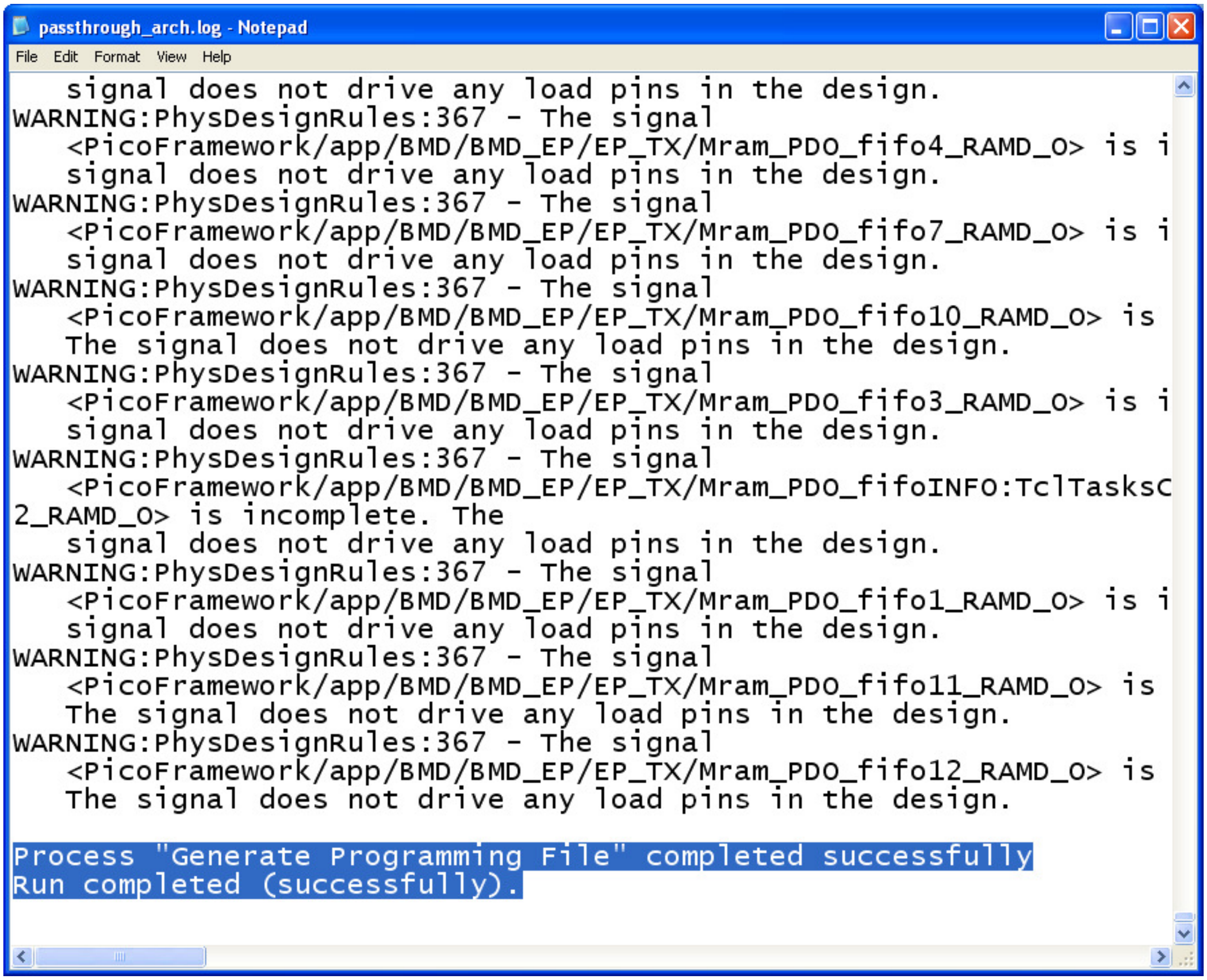
Figure 18 - Compiling FPGA in Quartus directory structure

A command window will appear showing the FPGA build process (primarily made up of many, many info and warning messages). Compile time will vary by machine depending upon project size. When completed successfully, something similar to the following will appear:



```
+---------------------+--------------+------+------+------------+-
|PicoFramework/app/BM |              |      |      |            |
|D/BMD_EP/EP_TX/BMD_I |              |      |      |            |
| NTR_CTRL/mwr_done_G |         Local|      |    1 |  0.000     |
+---------------------+--------------+------+------+------------+-
|PicoFramework/app/BM |              |      |      |            |
|D/BMD_EP/EP_TX/BMD_I |              |      |      |            |
| NTR_CTRL/mrd_done_G |         Local|      |    1 |  0.000     |
+---------------------+--------------+------+------+------------+-

* Net Skew is the difference between the minimum and maximum routi
only delays for the net. Note this is different from Clock Skew wh
is reported in TRCE timing report. Clock Skew is the difference be
the minimum and maximum path delays which includes logic delays.

Timing Score: 0 (Setup: 0, Hold: 0, Component Switching Limit: 0)

Number of Timing Constraints that were not applied: 1

Asterisk (*) preceding a constraint indicates it was not met.
   This may be due to a setup or hold violation.

------------------------------------------------------------------
  Constraint                                | Check        | Worst
                                            |              |   Sla
------------------------------------------------------------------
  TS_CLK_250 = PERIOD TIMEGRP "CLK_250" TS_ | SETUP        |    0.
  SYSCLK HIGH 50% PRIORITY 1                | HOLD         |    0.
------------------------------------------------------------------
  TS_CLK_125 = PERIOD TIMEGRP "CLK_125" TS_ | SETUP        |    0.
```

Figure 19 – Xilinx ISE 12.4 compile log file – timing score



```
   signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <PicoFramework/app/BMD/BMD_EP/EP_TX/Mram_PDO_fifo4_RAMD_O> is i
   signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <PicoFramework/app/BMD/BMD_EP/EP_TX/Mram_PDO_fifo7_RAMD_O> is i
   signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <PicoFramework/app/BMD/BMD_EP/EP_TX/Mram_PDO_fifo10_RAMD_O> is
   The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <PicoFramework/app/BMD/BMD_EP/EP_TX/Mram_PDO_fifo3_RAMD_O> is i
   signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <PicoFramework/app/BMD/BMD_EP/EP_TX/Mram_PDO_fifoINFO:TclTasksC
2_RAMD_O> is incomplete. The
   signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <PicoFramework/app/BMD/BMD_EP/EP_TX/Mram_PDO_fifo1_RAMD_O> is i
   signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <PicoFramework/app/BMD/BMD_EP/EP_TX/Mram_PDO_fifo11_RAMD_O> is
   The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
   <PicoFramework/app/BMD/BMD_EP/EP_TX/Mram_PDO_fifo12_RAMD_O> is
   The signal does not drive any load pins in the design.

Process "Generate Programming File" completed successfully
Run completed (successfully).
```

Figure 20 - Xilinx ISE 12.4 compile log file - complete

5.10. Exporting Software

The software application to be run on the host computer (with the M501 installed with it's drivers) can be exported in CoDeveloper.

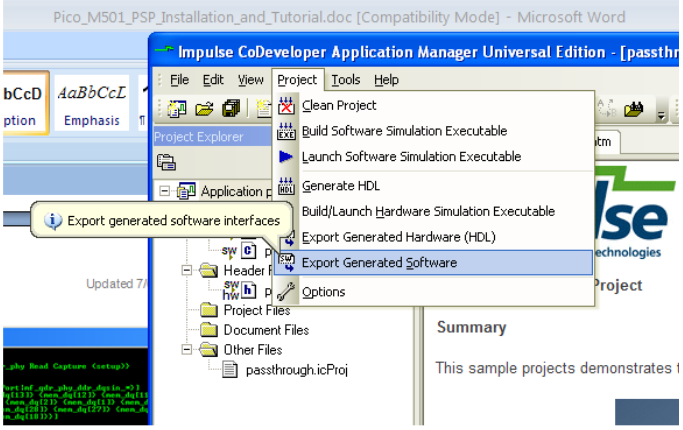
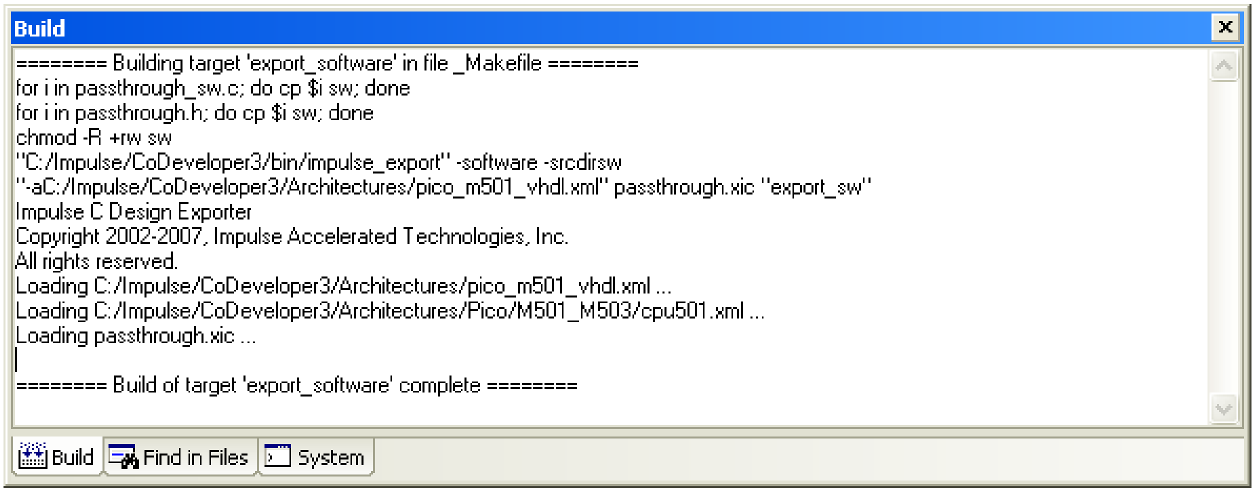

Pico_M501_PSP_Installation_and_Tutorial.doc [Compatibility Mode] - Microsoft Word
Impulse CoDeveloper Application Manager Universal Edition - [passthr
File Edit View Project Tools Help
Project Explorer
Application p
Export generated software interfaces
Header F
Project Files
Document Files
Other Files
passthrough.icProj
Clean Project
Build Software Simulation Executable
Launch Software Simulation Executable
Generate HDL
Build/Launch Hardware Simulation Executable
Export Generated Hardware (HDL)
Export Generated Software
Options
Project
Summary
This sample projects demonstrates t

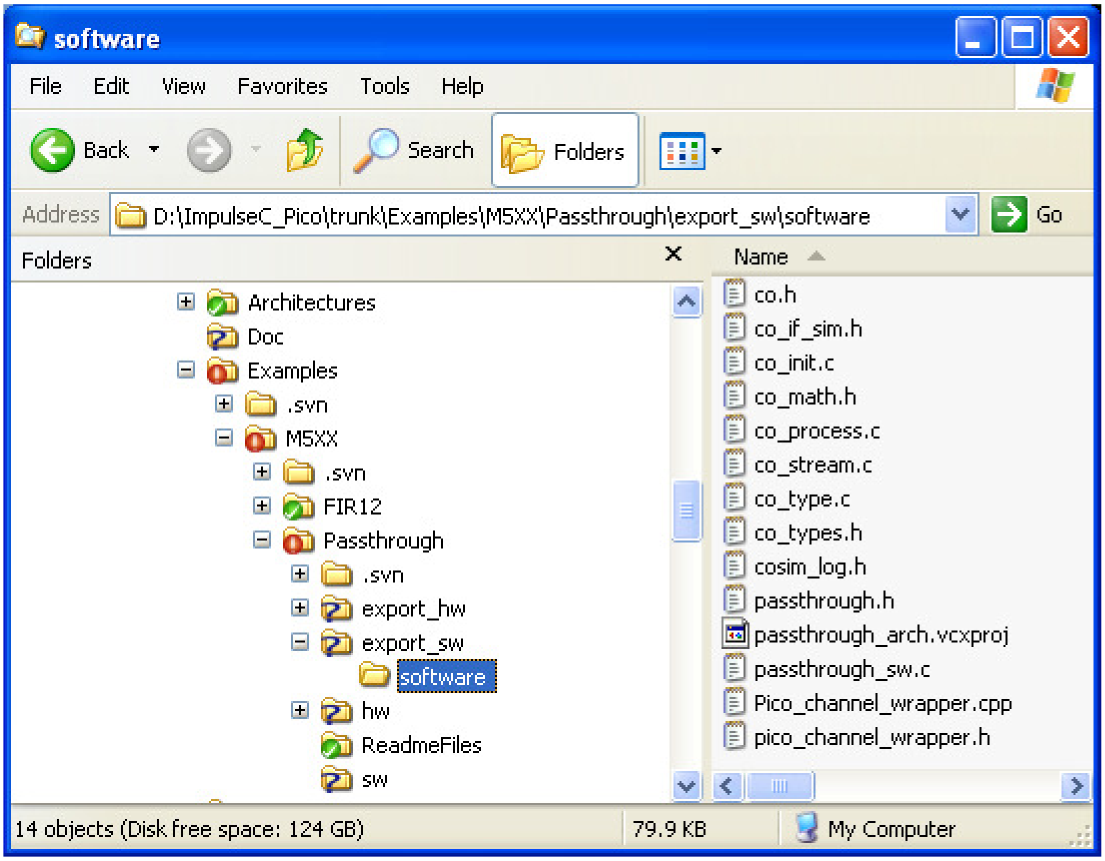
Figure 21 - Export Generated Software

Once completed without error in the Build window, it should be noted that the software code will be written to a newly created directory. The user can modify the target directory name. In this example, export_sw contains the exported software files.



```
======== Building target 'export_software' in file _Makefile ========
for i in passthrough_sw.c; do cp $i sw; done
for i in passthrough.h; do cp $i sw; done
chmod -R +rw sw
"C:/Impulse/CoDeveloper3/bin/impulse_export" -software -srcdirsw
"-aC:/Impulse/CoDeveloper3/Architectures/pico_m501_vhdl.xml" passthrough.xic "export_sw"
Impulse C Design Exporter
Copyright 2002-2007, Impulse Accelerated Technologies, Inc.
All rights reserved.
Loading C:/Impulse/CoDeveloper3/Architectures/pico_m501_vhdl.xml ...
Loading C:/Impulse/CoDeveloper3/Architectures/Pico/M501_M503/cpu501.xml ...
Loading passthrough.xic ...
|
======== Build of target 'export_software' complete ========
```

Figure 22 - Build window output



Address: D:\ImpulseC_Pico\trunk\Examples\M5XX\Passthrough\export_sw\software

```
co.h
co_if_sim.h
co_init.c
co_math.h
co_process.c
co_stream.c
co_type.c
co_types.h
cosim_log.h
passthrough.h
passthrough_arch.vcxproj
passthrough_sw.c
Pico_channel_wrapper.cpp
pico_channel_wrapper.h
```

14 objects (Disk free space: 124 GB)

Figure 23 - Exported software directory

5.11. Programming the FPGA

The compiled software application will program the FPGA on the M501.

5.12. Running Target Executable

The target application must be build using Visual Studio Express C++ 2010. Launch the GUI and load the project, passthrough_arch.vcxproj. Build the software executable. A new directory will be created with the software executable.

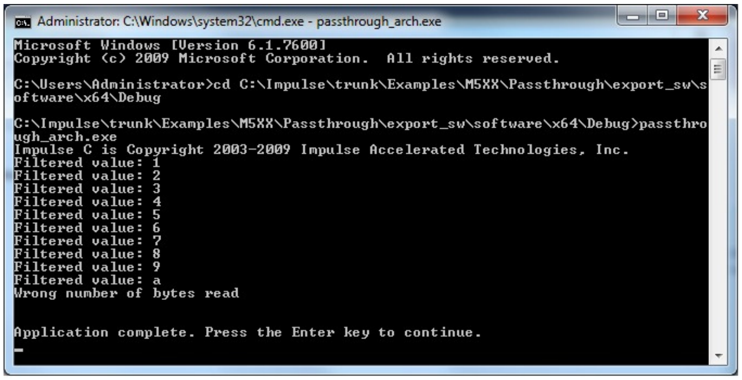
Passthrough**export_sw****software****x64****Debug**\passthrough_arch.exe

NOTE: The user MUST have administrator privileges in order to disable PCIe card after FPGA is programmed.

Open ("run as administrator") a command prompt. Navigate to the software directory and run the executable.

The results should mimic the software simulation exercised in CoDeveloper. It will load the FPGA and reset the PCIe card so there will be a slight delay until the card comes up and the application runs.

NOTE: The input file (filter_in.dat) needs to be copied to the executable directory.


Administrator: C:\Windows\system32\cmd.exe - passthrough_arch.exe

```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Administrator>cd C:\Impulse\trunk\Examples\M5XX\Passthrough\export_sw\s
oftware\x64\Debug

C:\Impulse\trunk\Examples\M5XX\Passthrough\export_sw\software\x64\Debug>passthro
ugh_arch.exe
Impulse C is Copyright 2003-2009 Impulse Accelerated Technologies, Inc.
Filtered value: 1
Filtered value: 2
Filtered value: 3
Filtered value: 4
Filtered value: 5
Filtered value: 6
Filtered value: 7
Filtered value: 8
Filtered value: 9
Filtered value: a
Wrong number of bytes read


Application complete. Press the Enter key to continue.
_
```

Figure 24 - Exported SW executed on target platform