



CoDeveloper Platform Support Package

**Pico Computing M501 PSP
User Guide – Linux
Version 1.0.1**

Impulse Accelerated Technologies, Inc.

www.ImpulseAccelerated.com

1.0 Table of Contents

1.0	TABLE OF CONTENTS	2
2.0	TABLE OF FIGURES	4
3.0	REVISION HISTORY	6
4.0	OVERVIEW	7
4.1.	Hardware Block Diagram	7
4.1.1.	Pico Computing Block Diagram	7
5.0	BEFORE GETTING STARTED: READ THIS FIRST.....	9
5.1.	Hardware limitations:	9
5.2.	Required Software Tools:.....	9
5.3.	Additional Required Files	10
5.4.	Required Hardware.....	10
6.0	HOST SYSTEM SETUP (LINUX)	11
6.1.	Install Pico M501 driver.....	11
6.2.	Running ISE on Linux	11
6.2.1.	Install Xilinx ISE 13.4	11
6.3.	Running ISE on Windows	11
6.3.1.	Copy Pico Installer source from Host PC to Development PC.....	11
7.0	DEVELOPMENT SYSTEM SETUP (WINDOWS).....	12
7.1.	Install Impulse CoDeveloper v3.70.e.6 or newer	12
7.2.	Running ISE on Windows	12
7.2.1.	Install Xilinx ISE 13.4	12
7.2.2.	Copy the Linux Pico Installer.....	12
7.2.3.	Configure Environment Variables.....	12
8.0	PASSTHROUGH EXAMPLE AND TUTORIAL.....	13
8.1.	Prerequisites.....	14
8.2.	CoDeveloper Project Files	15
8.3.	Opening Project	17
8.4.	Building Desktop Simulation Executable	18
8.5.	Running Desktop Simulation Executable	19
8.6.	Project Setup Before Hardware/Software Generation and Export	21
8.7.	Generating Hardware.....	22
8.8.	Exporting Hardware	24
8.9.	Compiling FPGA in Xilinx ISE 13.4	26
8.9.1.	Building bitfile under Windows	27
8.9.2.	Building bitfile under Linux	33
8.10.	Exporting Software	39
8.11.	Programming the FPGA	41
8.12.	Running Target Executable on the Host System	42
9.0	MEMTEST EXAMPLE AND TUTORIAL	44
9.1.	Prerequisites.....	44
9.2.	Memtest Example Description	45
9.2.1.	Overview of memtest.h	45

9.2.2.	Overview of memtest_hw.c	45
9.2.3.	Overview of memtest_sw.c	47
9.3.	CoDeveloper Project Files	50
9.4.	Opening Project	52
9.5.	Building Desktop Simulation Executable	53
9.6.	Running Desktop Simulation Executable	54
9.7.	Project Setup Before Hardware/Software Generation and Export	55
9.8.	Generating Hardware.....	56
9.9.	Exporting Hardware	58
9.10.	Compiling FPGA in Xilinx ISE 13.4.....	59
9.10.1.	Building bitfile under Windows.....	59
9.10.2.	Building bitfile under Linux.....	63
9.11.	Exporting Software	64
9.12.	Programming the FPGA	66
9.13.	Running Target Executable on the Host System	67

2.0 Table Of Figures

Figure 1 – Firmware Architecture (Pico M50X Series Platform Support Package Users Guide) .	8
Figure 2 - Impulse C Header File with 64 bit co_stream	15
Figure 3 - ImpulseC Hardware File with 64 bit co_stream	16
Figure 4 - Opening a project in CoDeveloper	17
Figure 5 - Build Simulation Desktop in CoDeveloper using pull-down menu.....	18
Figure 6 - Build Simulation Desktop in CoDeveloper using toolbar icon	18
Figure 7 - Output within the CoDeveloper IDE build window	18
Figure 8 - Launch software simulation window using pull-down menu.....	19
Figure 9 - Launch software simulation using toolbar icon	19
Figure 10 - Pop-up window during desktop simulation	20
Figure 11 - Project setup to pick Platform Support Package.....	21
Figure 12 - Generate HDL using pull-down menu	22
Figure 13 - Generate HDL using toolbar icon	22
Figure 14 - Build window output	23
Figure 15 - Export Generated Hardware (HDL) using pull-down menu.....	24
Figure 16 - Export Generated Hardware (HDL) using toolbar icon	24
Figure 17 - Build window output	25
Figure 18 - Compiling FPGA in exported ISE directory structure	26
Figure 19 - Generate ISE project files	27
Figure 20 - Expected gen50x_xise.log report	28
Figure 21 - Initial Xilinx ISE GUI screen	29
Figure 22 - Xilinx ISE regenerate IP core fifo128x512	29
Figure 23 - Xilinx ISE regenerate IP Core coregen_fifo_32x128	29
Figure 24 - Xilinx ISE 13.4 with timing score = 0	30
Figure 25 - Build bitfile in ISE 13.4	31
Figure 26 – Xilinx ISE 13.4 compile log file – timing score	32
Figure 27 - Generate ISE project files	33
Figure 28 - Expected gen50x_xise.log report	33
Figure 29 - Select the ISE project	34
Figure 30 - Initial project status after loading project	35
Figure 31 - Regenerate fifo128x512.....	35
Figure 32 - Regenerate coregen_fifo_32x128	36
Figure 33 - Place and Route and bitfile generation with timing score equal to zero	36
Figure 34 - Ubuntu Xilinx ISE 13.4 Command Line output	37
Figure 35 - Xilinx ISE 13.4 results file with timing score equal to zero	38
Figure 36 - Export Generated Software.....	39
Figure 37 - Build window output	40
Figure 38 - Exported software directory	40
Figure 39 - Files & Directories to be copied to the Host System	42
Figure 40 - Exported SW executed on target platform.....	43
Figure 41 - memtest.h	45
Figure 42 - User defined funtion in memtest_hw.c	46
Figure 43 - Configuration in memtest_hw.c	47
Figure 44 - User defined stimulus in memtest_sw.c	48
Figure 45 - Main program in memtest_sw.c	49
Figure 46 - Impulse C Header File	50
Figure 47 - ImpulseC Hardware File.....	51
Figure 48 - Opening a project in CoDeveloper	52
Figure 49 - Build Simulation Desktop in CoDeveloper using pull-down menu.....	53

Figure 50 - Build Simulation Desktop in CoDeveloper using toolbar icon	53
Figure 51 - Output within the CoDeveloper IDE build window	53
Figure 52 - Launch software simulation window using pull-down menu.....	54
Figure 53 - Launch software simulation using toolbar icon	54
Figure 54 - Pop-up window during desktop simulation	54
Figure 55 - Project setup to pick Platform Support Package.....	55
Figure 56 - Generate HDL using pull-down menu	56
Figure 57 - Generate HDL using toolbar icon	56
Figure 58 - Build window output	57
Figure 59 - Export Generated Hardware (HDL) using pull-down menu.....	58
Figure 60 - Export Generated Hardware (HDL) using toolbar icon	58
Figure 61 - Build window output	58
Figure 62 - Compiling FPGA in Quartus directory structure.....	59
Figure 63 - Generate ISE project files	60
Figure 64 - Expected Gen_Ise_File log report.....	60
Figure 65 - Initial Xilinx ISE 13.4 GUI screen	61
Figure 66 - Xilinx ISE 13.4 regenerate IP core fifo128x512com	61
Figure 67 - Xilinx ISE 13.4 regenerate IP Core fifo128x512	62
Figure 68 - Xilinx ISE 13.4 regenerate IP Core coregen_fifo_32x128.....	62
Figure 69 - Xilinx ISE 13.4 with timing score = 0	62
Figure 70 - Export Generated Software.....	64
Figure 71 - Build window output	64
Figure 72 - Exported software directory	65
Figure 73 - Files & Directories to be copied to the Host System.....	67
Figure 74 - Exported SW executed on target platform.....	68

3.0 Revision History

Date	Version	Description	Author
11/3/2012	1.0	Initial Creation	Shaumil Dave
11/10/2012	1.1	Added Memtest description	Shaumil Dave

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 referred to as the “bitfile”. The bitfile may be built under Windows or Linux and either via the ISE GUI or from command line using a build script.
- Application executable built via Makefile on a Linux platform.
- 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:

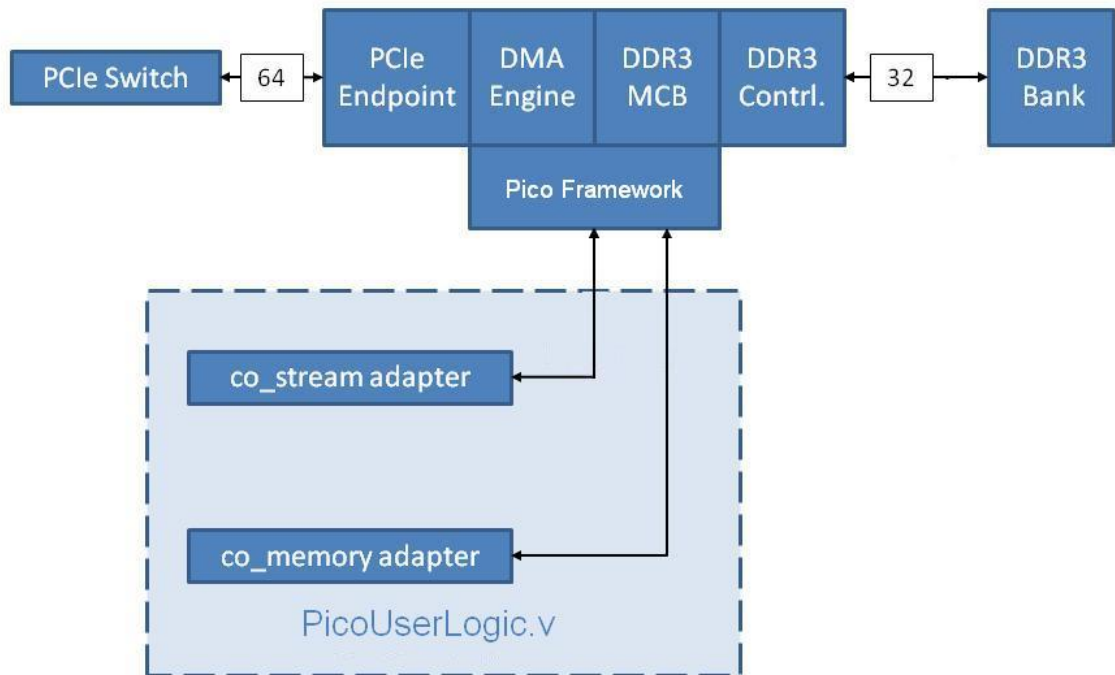


Figure 1 – Firmware Architecture (Pico M50X Series Platform Support Package Users Guide)

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

5.0 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.

5.1. Hardware limitations:

1. co_streams require all data transfers to be a multiple of 128-bits/16bytes
2. co_memory (not used in Passthrough example) use limitations:
 - a. The memory interface requires that memory transfers begin at an address with 256-byte alignment
 - b. The memory interface assumes that the transfer size in bytes is a multiple of 16

5.2. Required Software Tools:

- Impulse CoDeveloper v3.70.e.6 or newer for the Development System.
- 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 13.4 for Windows (exactly, not newer) for the Development System.
- Windows 7 for Impulse CoDeveloper and Xilinx ISE 13.4 (the Development System).
- Ubuntu (Debian) 12.04 LTS for the Host System (With EX500 PCIe card and either M501 or M503 FPGA modules)
- Pico Linux installer package version 5.0.6.1 or newer for the Host System:
 - Go to <http://picocomputing.com/support/software>
 - Select *Linux Installer (M-501, M-503, M-505)*
 - Linux_5.0.6.1_all.deb

5.3. 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
```

CoDeveloper will use the project file, “passthrough.icProj”, to generate HDL as well as a software application to load the FPGA image.

5.4. Required Hardware

The following hardware is required for development using this PSP:

- EX500 x16 full length full height PCIe FPGA Development Board.
- M501 or M503 FPGA module.
- Host System with Ubuntu 12.04 LTS 64-bit OS based development PC for running all tools – Recommended: 100GB disk space available for tools installation and 12GB RAM.
- Host System that has an 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 in the Host System. Consideration for your video card must be given if it also requires a separate 12 volt power connection.
- Development System for installation of CoDeveloper and Xilinx ISE 13.4.
 - Impulse CoDeveloper and M50x PSP: Windows only (32 or 64-bit)
 - Xilinx ISE 13.4: Windows 64-bit or Linux 64-bit

6.0 Host System Setup (Linux)

6.1. Install Pico M501 driver

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

1. After downloading the Pico Linux installer package (Linux_*.deb), follow the “Linux_PicoGettingStarted.pdf” to install Linux software, firmware, and the driver for the M50x.

6.2. Running ISE on Linux

6.2.1. Install Xilinx ISE 13.4

1. Please see vendor supplied documentation for installation.
2. Please see vendor supplied documentation for licensing.

6.3. Running ISE on Windows

6.3.1. Copy Pico Installer source from Host PC to Development PC

If Xilinx ISE will be run on Windows, prepare to copy the directory “/usr/src/picocomputing-5.0.6.1” from the Linux Host PC to the Windows Development system. For example, save the directory to a USB flash drive.

7.0 Development System Setup (Windows)

7.1. Install Impulse CoDeveloper v3.70.e.6 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

1. Add the Pico M5XX PSP to the CoDeveloper installation.
 - a. Copy the supplied "**Architectures**" directory to "**Impulse\CoDeveloper3**".

Architectures

Pico

pico_m501_linux_vhdl.xml

pico_m503_linux_vhdl.xml

2. Copy the supplied "**Examples**" directory to a working directory on the development PC for access to the pre-built example files.
3. Download the latest version 3.x and installation note
 - a. <http://www.impulseaccelerated.com/ReleaseFiles/>
 - b. (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.

7.2. Running ISE on Windows

7.2.1. Install Xilinx ISE 13.4

1. Please see vendor supplied documentation for installation.
2. Please see vendor supplied documentation for licensing.

7.2.2. Copy the Linux Pico Installer

1. If Xilinx ISE is to be run on Windows: From the Host PC, copy "/usr/src/picocomputing-5.0.6.1" to "c:\usr\src\picocomputing-5.0.6.1"

7.2.3. Configure Environment Variables

1. Add environment variable PICOBASE = "c:\usr\src\picocomputing-5.0.6.1"
2. Add environment variable XILINX = "c:\Xilinx\13.4\ISE_DS\ISE"
3. Add environment variable XILINX_BASE = "c:\Xilinx\13.4"

8.0 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 13.4-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

8.1. Prerequisites

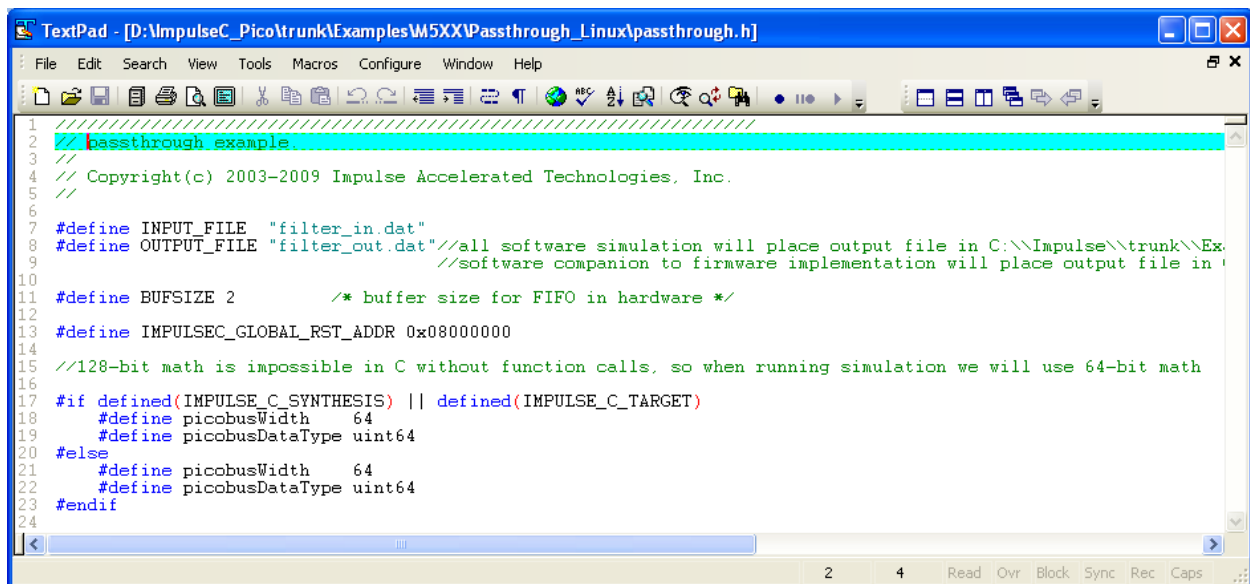
The tutorial in this Platform Support Package assumes that you have read and understand the introductory sections of the CoDeveloper User's Guide, installed with CoDeveloper and accessed from the Help menu. In particular, you should take the time to go through the tutorials provided with CoDeveloper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

8.2. 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.



```
1 ////////////////////////////////////////////////////////////////////
2 // passthrough example
3 //
4 // Copyright(c) 2003-2009 Impulse Accelerated Technologies, Inc.
5 //
6
7 #define INPUT_FILE "filter_in.dat"
8 #define OUTPUT_FILE "filter_out.dat"//all software simulation will place output file in C:\\Impulse\\trunk\\Ex
9 //software companion to firmware implementation will place output file in
10
11 #define BUFSIZE 2 /* buffer size for FIFO in hardware */
12
13 #define IMPULSE_GLOBAL_RST_ADDR 0x08000000
14
15 //128-bit math is impossible in C without function calls, so when running simulation we will use 64-bit math
16
17 #if defined(IMPULSE_C_SYNTHESIS) || defined(IMPULSE_C_TARGET)
18 #define picobusWidth 64
19 #define picobusDataType uint64
20 #else
21 #define picobusWidth 64
22 #define picobusDataType uint64
23 #endif
24
```

Figure 2 - Impulse C Header File with 64 bit co_stream

```

1 ////////////////////////////////////////////////////////////////////
2 // Passthrough filter example.
3 //
4 // Copyright(c) 2003-2009 Impulse Accelerated Technologies, Inc.
5 //
6 // passthrough_hw.c: includes the hardware process and configuration
7 // function.
8 //
9 // See additional comments in passthrough.h.
10 //
11
12 #include <stdio.h>
13 #include "co.h"
14 #include "cosim_log.h"
15 #include "passthrough.h"
16
17 extern void test_producer(co_stream waveform_in);
18 extern void test_consumer(co_stream waveform_out);
19
20 ////////////////////////////////////////////////////////////////////
21 // This is the
22 // passthrough hardware process.
23 // The process accepts a stream representing a
24 // waveform of 32-bit samples and returns the
25 // stream
26 //
27 // Adjust the StageDelay pragma to balance max
28 // frequency with pipeline rate
29 //
30 // This sample has been modified to use the Pico M501's
31 // 128-bit bus.
32
33 void passthrough(co_stream filter_in, co_stream filter_out)
34 {
35     IF_SIM(int samplesread = 0;)
36     IF_SIM(int sampleswritten = 0;)
37
38     picobusDataType nSample;
39
40     IF_SIM( cosim_logwindow log; )
41     IF_SIM ( log = cosim_logwindow_create("passthrough"); )
42
43     do { // Hardware processes run forever
44
45         co_stream_open(filter_in, O_RDONLY, UINT_TYPE((picobusWidth)));
46         co_stream_open(filter_out, O_WRONLY, UINT_TYPE((picobusWidth)));
47
48         while (1) {
49             co_stream_read(filter_in, &nSample, sizeof(picobusDataType));
50             IF_SIM(samplesread++;)
51             co_stream_write(filter_out, &nSample, sizeof(picobusDataType));
52             IF_SIM(sampleswritten++;)
53             IF_SIM(if (sampleswritten == 10) break;)
54         }
55
56         co_stream_close(filter_in);
57         co_stream_close(filter_out);
58
59         IF_SIM(break;) // Only run once for desktop simulation
60     } while(1);
61 }
62
63
64 //

```

For Help, press F1 38 5 Read Ovr Block S

Figure 3 - ImpulseC Hardware File with 64 bit co_stream

8.3. Opening Project

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

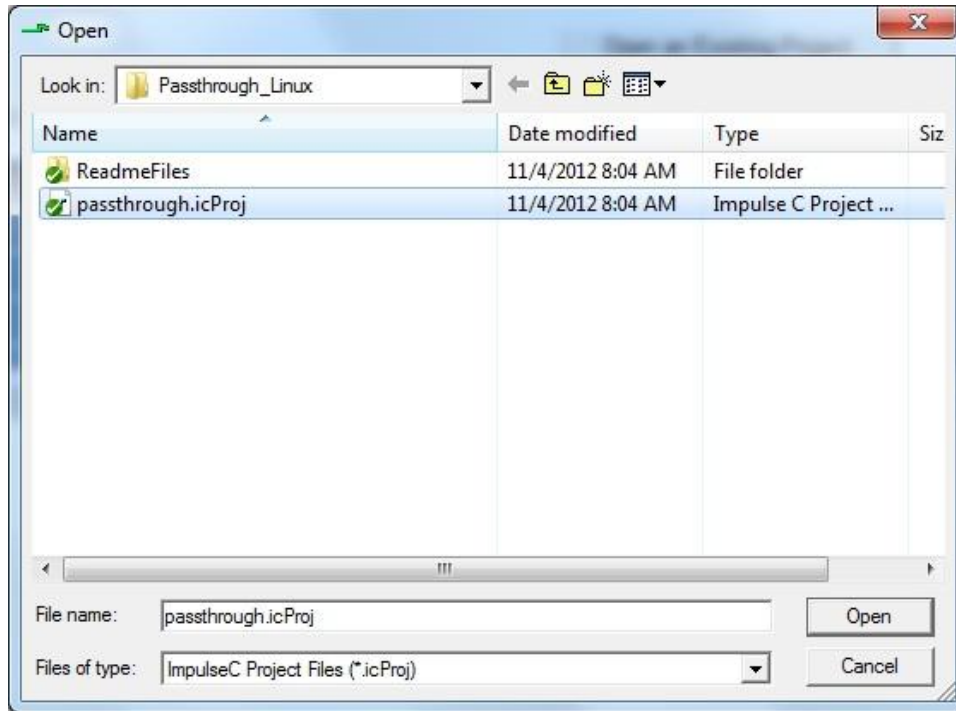


Figure 4 - Opening a project in CoDeveloper

8.4. Building Desktop Simulation Executable

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

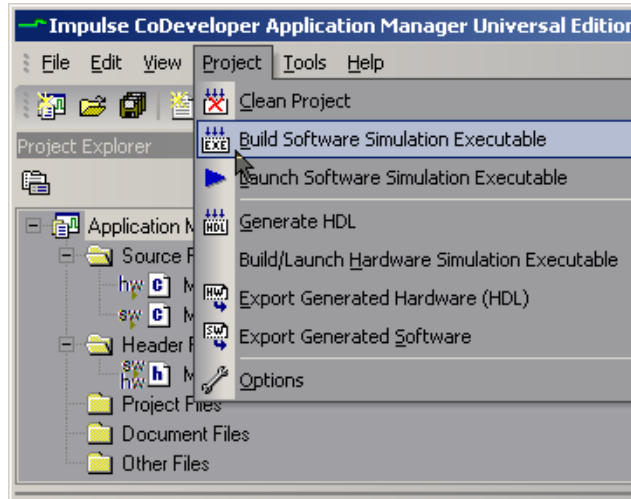


Figure 5 - Build Simulation Desktop in CoDeveloper using pull-down menu

Or via toolbar:

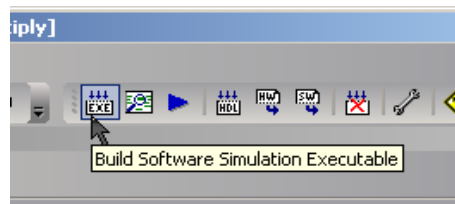


Figure 6 - Build Simulation Desktop in CoDeveloper using toolbar icon

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

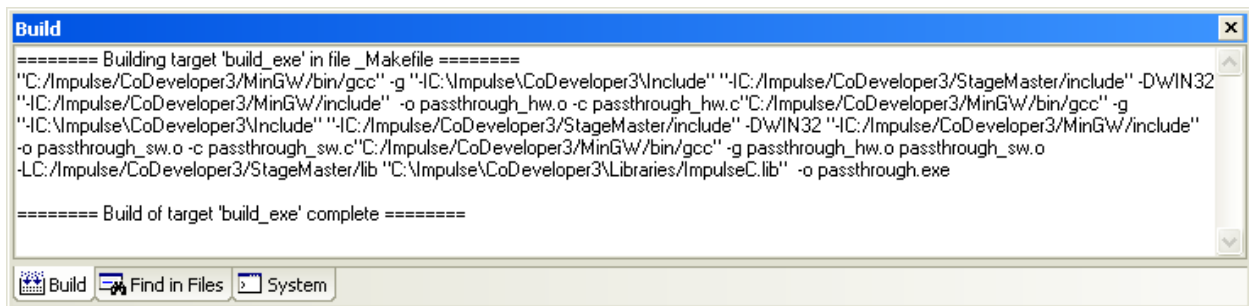


Figure 7 - Output within the CoDeveloper IDE build window

8.5. Running Desktop Simulation Executable

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

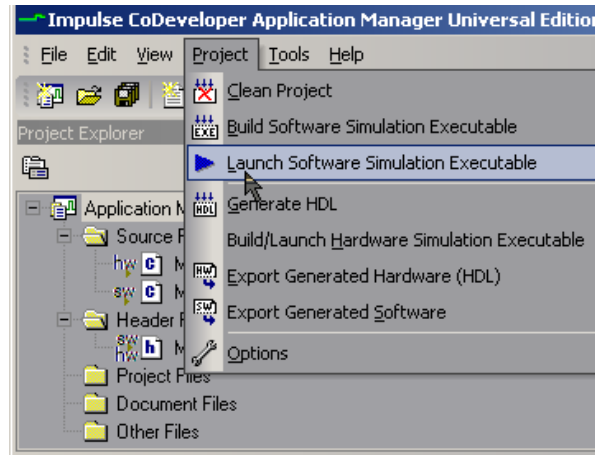


Figure 8 - Launch software simulation window using pull-down menu

Or via toolbar:

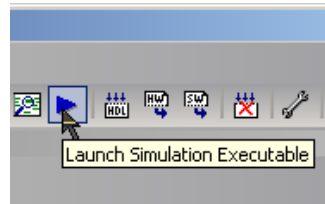
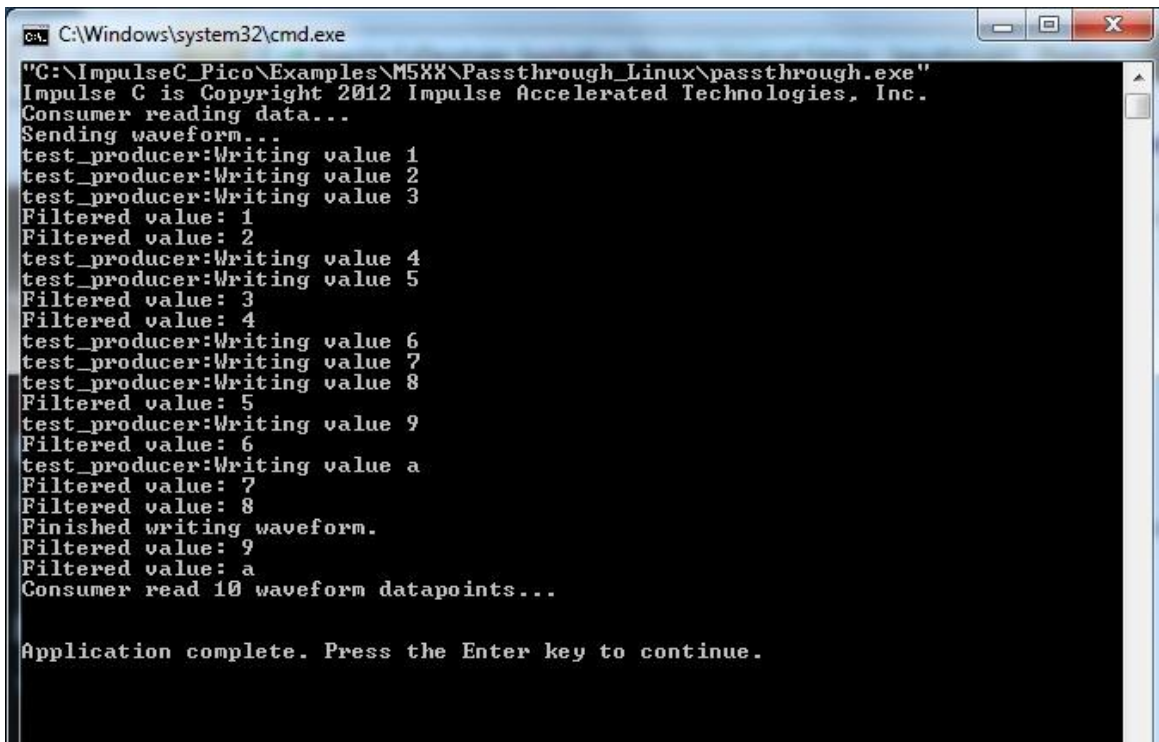


Figure 9 - Launch software simulation using toolbar icon

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



```
C:\Windows\system32\cmd.exe
"C:\ImpulseC_Pico\Examples\M5XX\Passthrough_Linux\passthrough.exe"
Impulse C is Copyright 2012 Impulse Accelerated Technologies, Inc.
Consumer reading data...
Sending waveform...
test_producer:Writing value 1
test_producer:Writing value 2
test_producer:Writing value 3
Filtered value: 1
Filtered value: 2
test_producer:Writing value 4
test_producer:Writing value 5
Filtered value: 3
Filtered value: 4
test_producer:Writing value 6
test_producer:Writing value 7
test_producer:Writing value 8
Filtered value: 5
test_producer:Writing value 9
Filtered value: 6
test_producer:Writing value a
Filtered value: 7
Filtered value: 8
Finished writing waveform.
Filtered value: 9
Filtered value: a
Consumer read 10 waveform datapoints...

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

Figure 10 - Pop-up window during desktop simulation

8.6. 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 Linux (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:

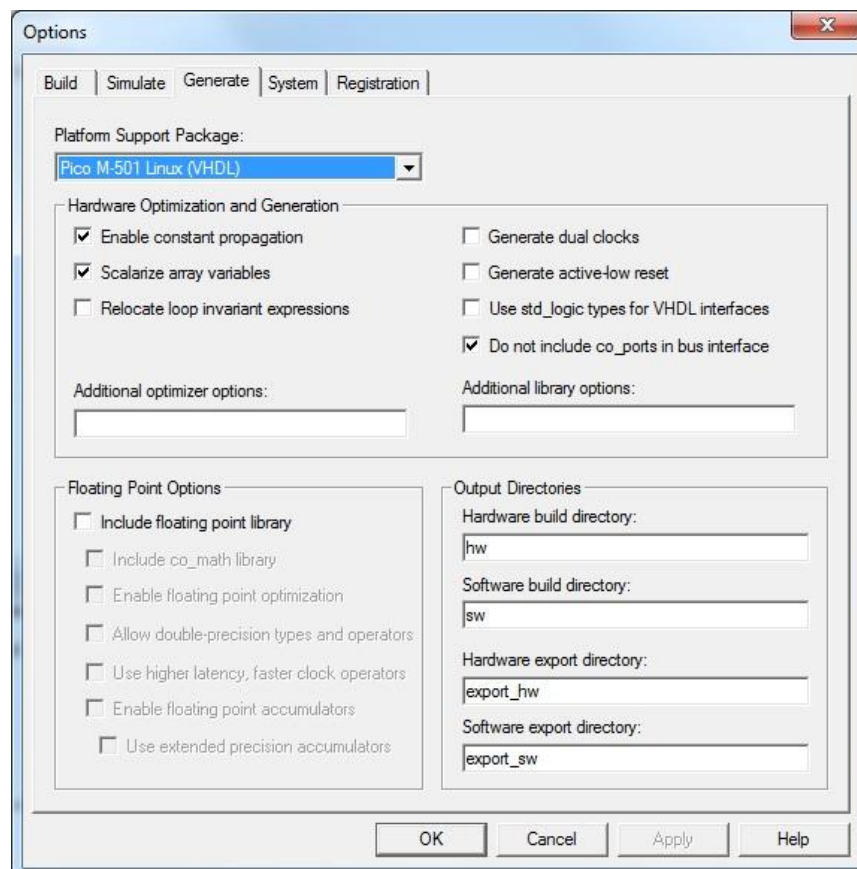


Figure 11 - Project setup to pick Platform Support Package

8.7. Generating Hardware

Generate hardware via “Project” menu:

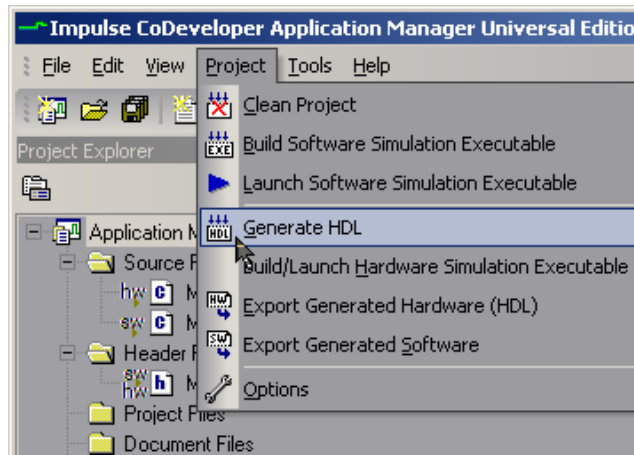


Figure 12 - Generate HDL using pull-down menu

Or via toolbar:

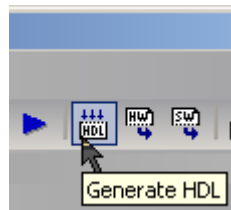


Figure 13 - Generate HDL using toolbar icon

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:

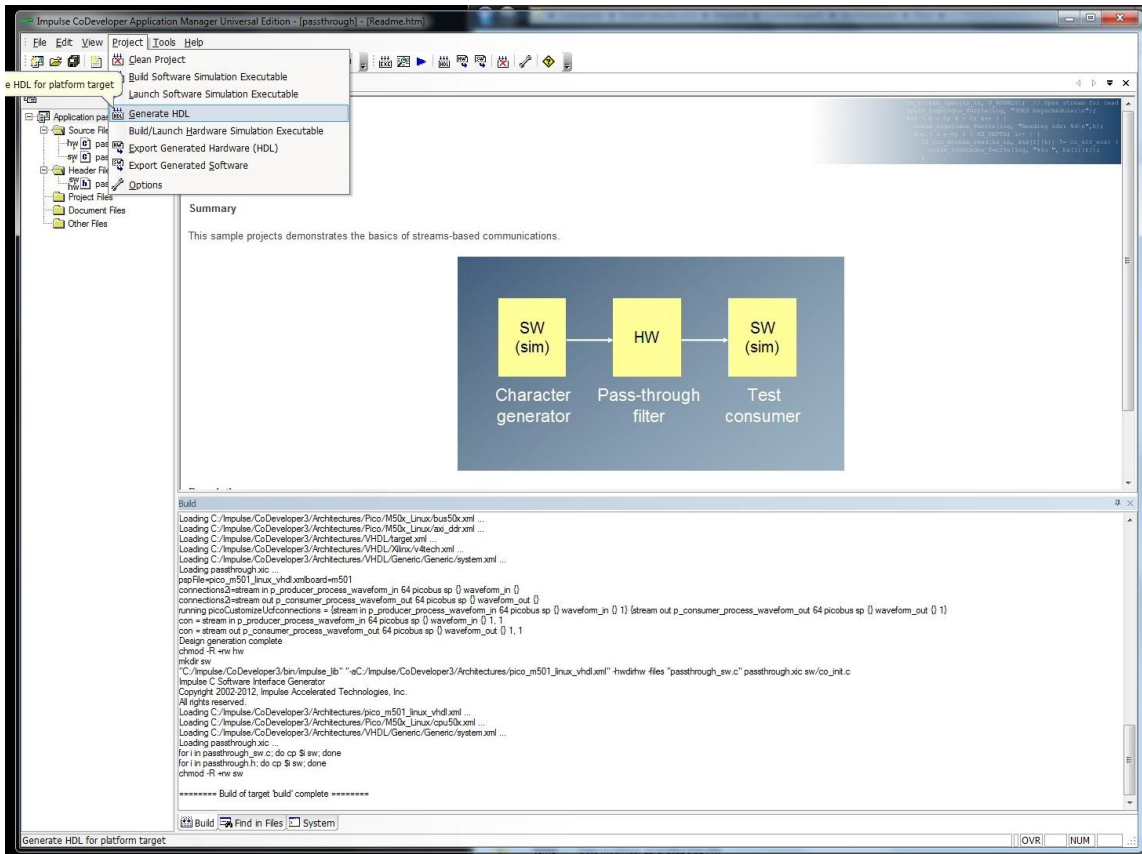


Figure 14 - Build window output

8.8. Exporting Hardware

Export hardware via “Project” menu:

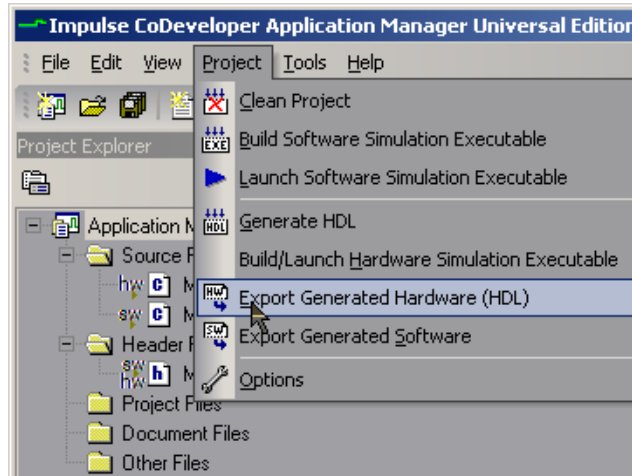


Figure 15 - Export Generated Hardware (HDL) using pull-down menu

Or via toolbar:

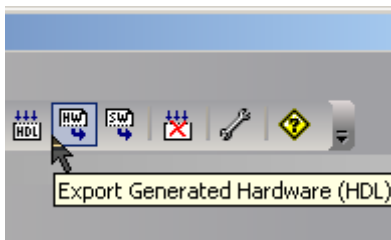


Figure 16 - Export Generated Hardware (HDL) using toolbar icon

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:

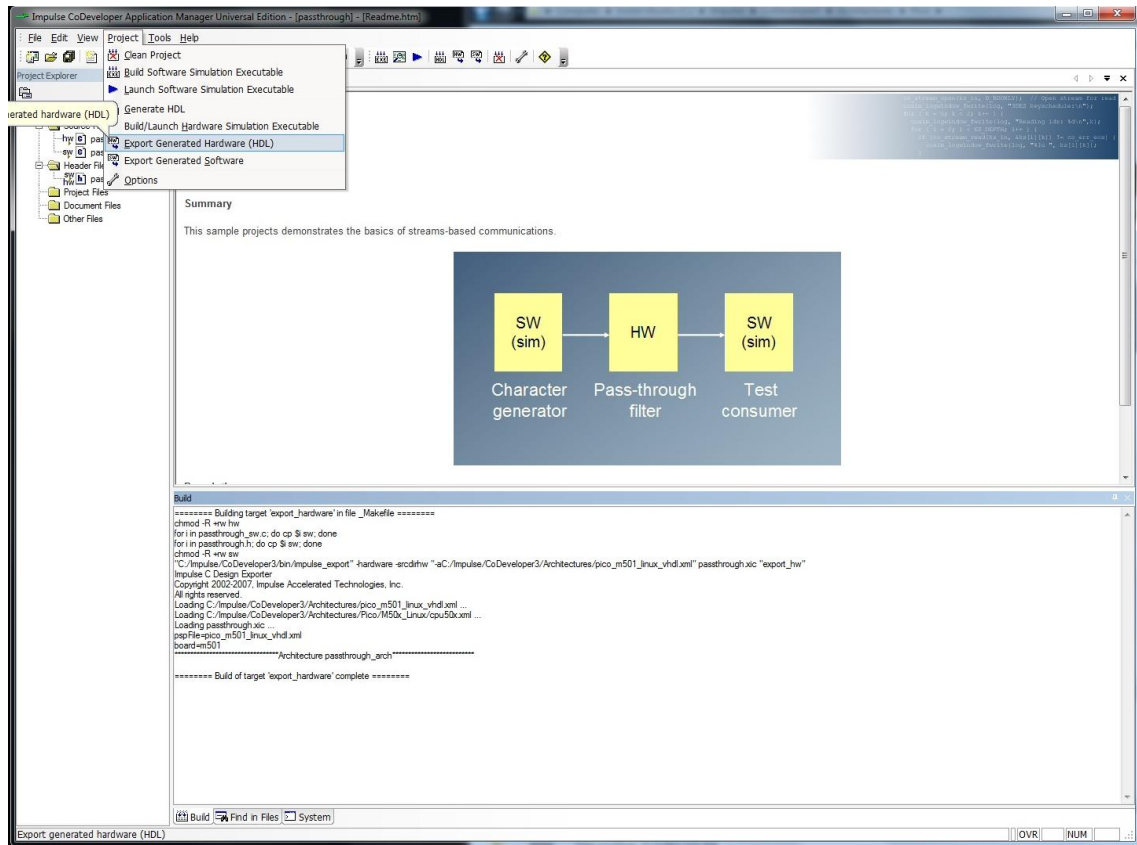


Figure 17 - Build window output

8.9. Compiling FPGA in Xilinx ISE 13.4

After exporting hardware, under the specified hardware export directory will be a directory structure that includes all necessary files for building the FPGA binary.

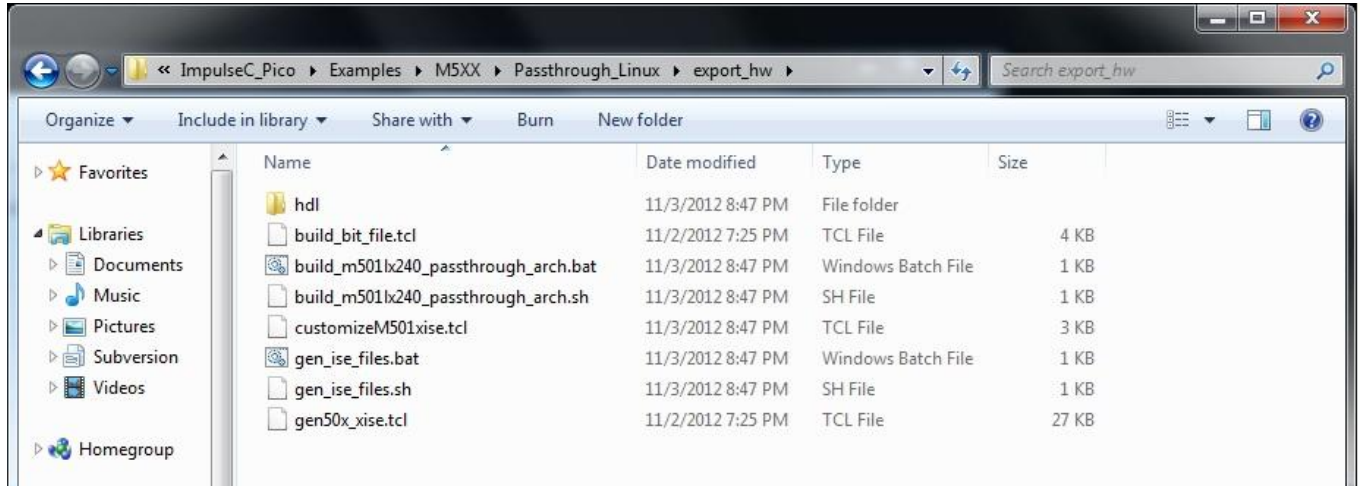


Figure 18 - Compiling FPGA in exported ISE directory structure

At this point either Windows or Linux may be used to produce the bitfile running Xilinx ISE either through the GUI or from command line. Windows and Linux both use the following similar steps. Choose the one that suits your needs.

Windows:

1. Generate the ISE project files by running “gen_ise_files.bat”
2. Build the bit file using Xilinx ISE either by:
 - a. Running the generated “build_m50*.bat” file
 - b. Launching the Xilinx ISE GUI opening the newly built “m50*.xise” file

Linux:

3. Generate the ISE project files by running “gen_ise_files.sh”
4. Build the bit file using Xilinx ISE either by:
 - a. Running the generated “build_m50*.sh” file
 - b. Launching the Xilinx ISE GUI opening the newly built “m50*.xise” file

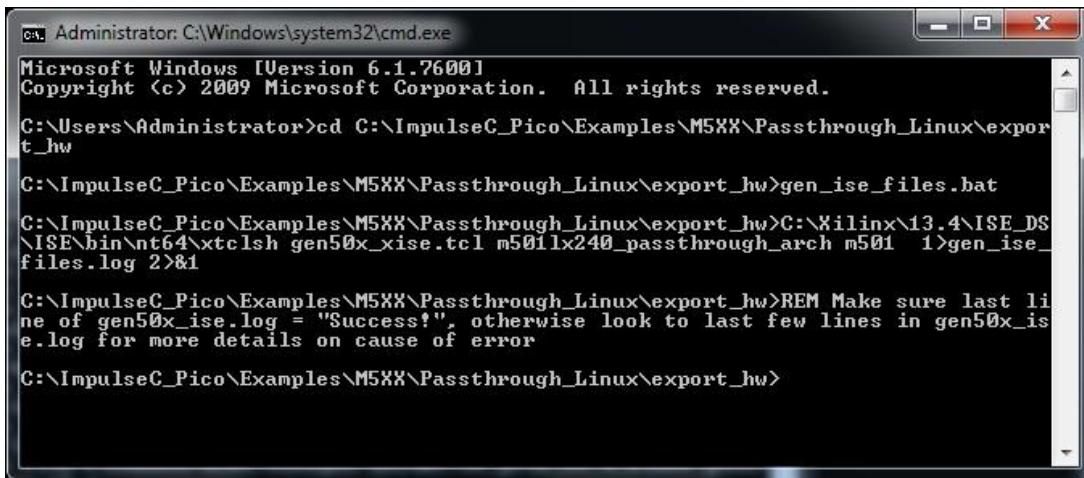
These methods will generate the necessary bitfile to be loaded on the Host System and are outlined in the sections that follow.

8.9.1. Building bitfile under Windows

8.9.1.1. Generate Xilinx ISE Project Files

First generate the ISE project by executing “gen_ise_files.bat”. This process will create an ISE project file as well as copy all the necessary HDL files and CORE Generator components from the Pico installation needed by ISE to generate the FPGA bitfile.

1. Open a command window
2. Change directories to “export_hw”
3. Execute “gen_ise_files.bat” as shown in Figure 19.
4. Verify that the files were created successfully by viewing the end of log file, as shown in Figure 20. “Successful!” should be the last line in the log file (use “type gen50x_xise.log” to display log).



```
ca. Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>cd C:\ImpulseC_Pico\Examples\M5XX\Passthrough_Linux\export_hw
C:\ImpulseC_Pico\Examples\M5XX\Passthrough_Linux\export_hw>gen_ise_files.bat
C:\ImpulseC_Pico\Examples\M5XX\Passthrough_Linux\export_hw>C:\Xilinx\13.4\ISE_DS\ISE\bin\nt64\xtclsh gen50x_xise.tcl m501lx240_passthrough_arch m501 1>gen_ise_files.log 2>&1
C:\ImpulseC_Pico\Examples\M5XX\Passthrough_Linux\export_hw>REM Make sure last line of gen50x_xise.log = "Successful!", otherwise look to last few lines in gen50x_xise.log for more details on cause of error
C:\ImpulseC_Pico\Examples\M5XX\Passthrough_Linux\export_hw>
```

Figure 19 - Generate ISE project files

```

Administrator: C:\Windows\system32\cmd.exe
source/axi_basic_rx_pipeline.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/axi_basic_top.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/axi_basic_tx.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/axi_basic_tx_pipeline.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/axi_basic_tx_thrtl_ctl.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/gtx_drp_chanalign_fix_3752_v6.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/gtx_rx_valid_filter_v6.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/gtx_tx_sync_rate_v6.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/gtx_wrapper_v6.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/pcie_128_if.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/pcie_2_0_v6.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/pcie_brms_v6.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/pcie_bram_top_v6.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/pcie_bram_v6.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/pcie_cfg_128.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/pcie_clocking_v6.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/pcie_gtx_v6.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/pcie_pipe_lane_v6.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/pcie_pipe_misc_v6.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/pcie_pipe_v6.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/pcie_reset_delay_v6.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/pcie_trn_128.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/pcie_upconfig_fix_3451_v6.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/sync_fifo.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/trn_rx_128.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/trn_tx_128.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/v6_pcie_v2_4_8lane_gen2.v
xfile_add: add file=firmware/m501/src/M501_LX240T_DDR3.ucf
xfile_add: add file=firmware/m501/src/M501_LX240T_PCIE.ucf
xfile_add:End
Configuring project properties
closing project
Success!
C:\ImpulseC_Pico\Examples\M5XX\Passthrough_Linux\export_hw>

```

Figure 20 - Expected gen50x_xise.log report

8.9.1.2. Compiling FPGA in Xilinx ISE GUI

Launch the Xilinx ISE GUI and open the ISE project located in the **export_hw** directory. Select Pico_Toplevel in the *Hierarchy* window and double-click “generate programming file” in the *Processes* window.

NOTE: Xilinx ISE will ask the user to regenerate two fifos (a one time process). The following figures illustrate initial project launch, regenerate the two corgen fifos, and final output with timing score equal to zero.

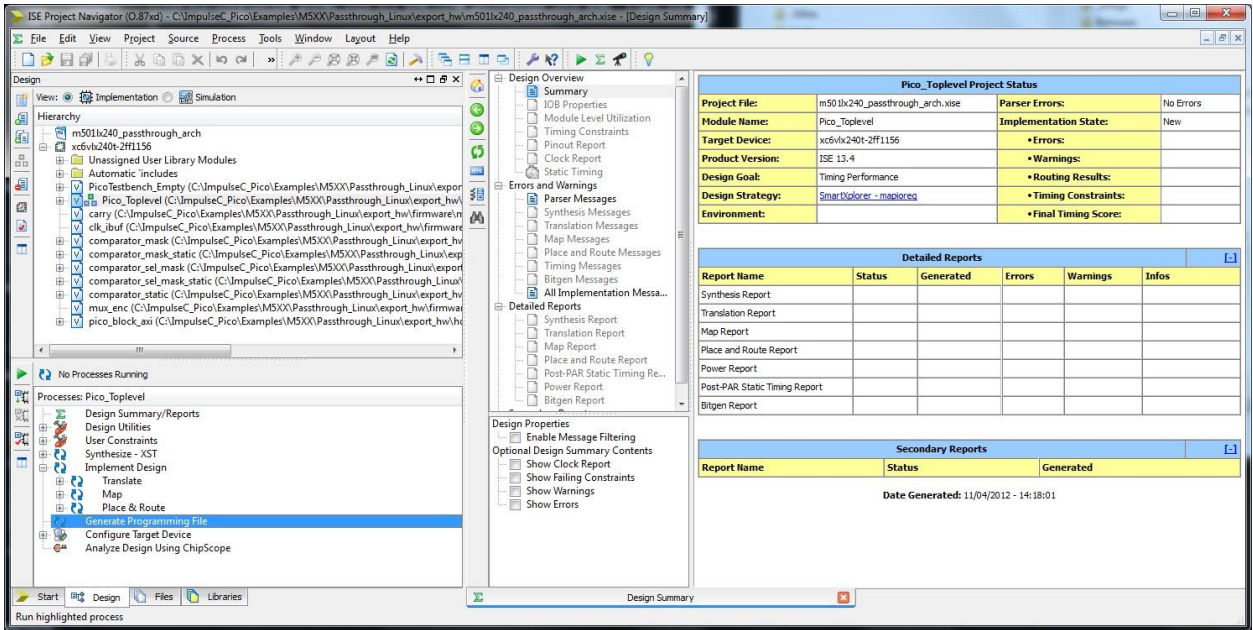


Figure 21 - Initial Xilinx ISE GUI screen

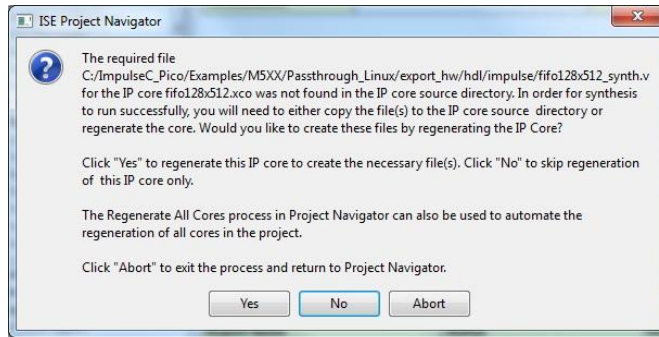


Figure 22 - Xilinx ISE regenerate IP core fifo128x512

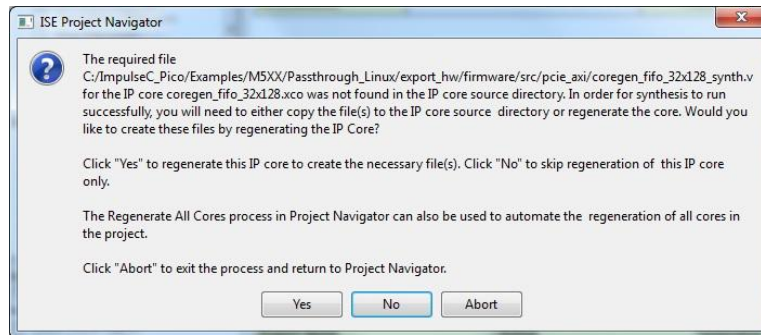


Figure 23 - Xilinx ISE regenerate IP Core coregen_fifo_32x128

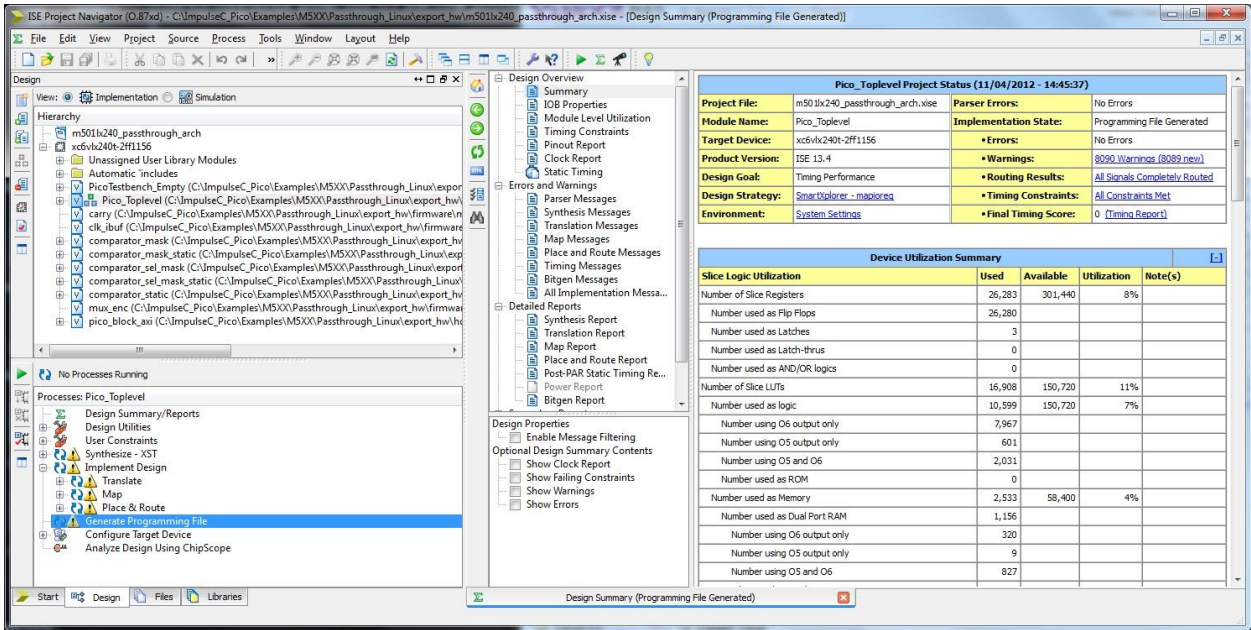


Figure 24 - Xilinx ISE 13.4 with timing score = 0

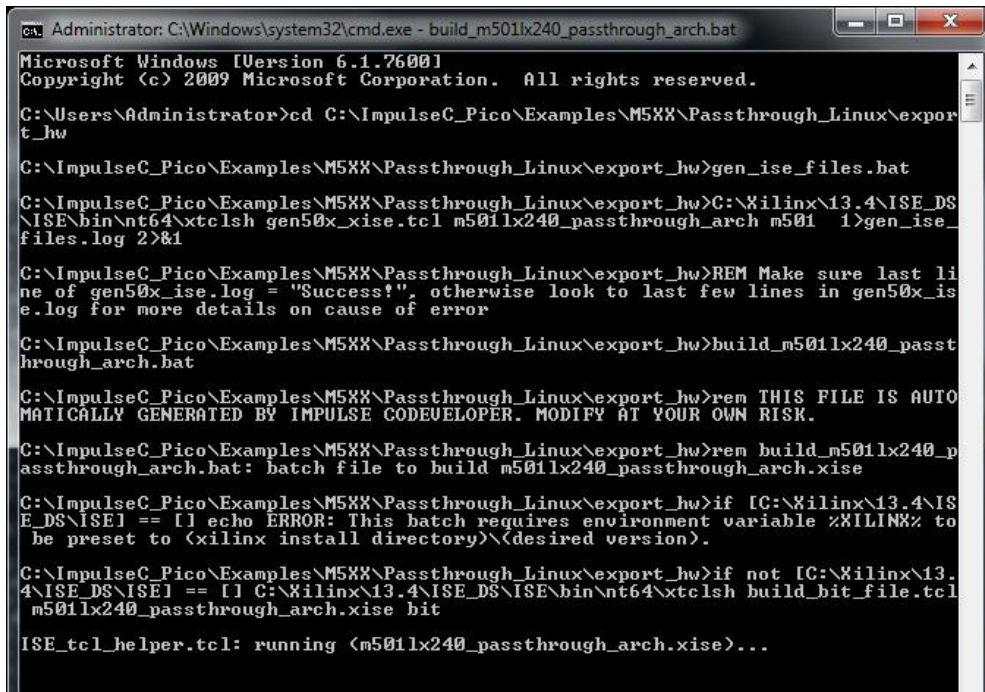
Once ISE 13.4 has completed, a bitfile “pico_toplevel.bit” will be present in the **export_hw** directory.

Note: The bitfile will need to be renamed to “Pico_Toplevel.bit” when copying to Linux to run with the software in the example.

8.9.1.3. Compiling FPGA in Xilinx ISE using Command Line

In the top directory there will be the batch file “build_passthrough_arch.bat” used to automatically run Xilinx ISE to create the necessary .bit file used to program the M501 FPGA.

1. Open a command window
2. Execute “build_m501x240_passthrough_arch.bat” as shown in Figure 25.



```

Administrator: C:\Windows\system32\cmd.exe - build_m501x240_passthrough_arch.bat
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>cd C:\ImpulseC_Pico\Examples\M5XX\Passthrough_Linux\export_hw
C:\ImpulseC_Pico\Examples\M5XX\Passthrough_Linux\export_hw>gen_ise_files.bat
C:\ImpulseC_Pico\Examples\M5XX\Passthrough_Linux\export_hw>C:\Xilinx\13.4\ISE_DS\ISE\bin\nt64\xtclsh gen50x_ise.tcl m501x240_passthrough_arch m501 1>gen_ise_files.log 2>&1
C:\ImpulseC_Pico\Examples\M5XX\Passthrough_Linux\export_hw>REM Make sure last line of gen50x_ise.log = "Success!", otherwise look to last few lines in gen50x_ise.log for more details on cause of error
C:\ImpulseC_Pico\Examples\M5XX\Passthrough_Linux\export_hw>build_m501x240_passthrough_arch.bat
C:\ImpulseC_Pico\Examples\M5XX\Passthrough_Linux\export_hw>rem THIS FILE IS AUTOMATICALLY GENERATED BY IMPULSE CODEVELOPER. MODIFY AT YOUR OWN RISK.
C:\ImpulseC_Pico\Examples\M5XX\Passthrough_Linux\export_hw>rem build_m501x240_passthrough_arch.bat: batch file to build m501x240_passthrough_arch.xise
C:\ImpulseC_Pico\Examples\M5XX\Passthrough_Linux\export_hw>if [C:\Xilinx\13.4\ISE_DS\ISE] == [] echo ERROR: This batch requires environment variable %XILINX% to be preset to <xilinx install directory>\<desired version>.
C:\ImpulseC_Pico\Examples\M5XX\Passthrough_Linux\export_hw>if not [C:\Xilinx\13.4\ISE_DS\ISE] == [] C:\Xilinx\13.4\ISE_DS\ISE\bin\nt64\xtclsh build_bit_file.tcl m501x240_passthrough_arch.xise bit
ISE_tcl_helper.tcl: running (m501x240_passthrough_arch.xise)...
  
```

Figure 25 - Build bitfile in ISE 13.4

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 and a bitfile “pico_toplevel.bit” will be present in the **export_hw** directory.

Note: The bitfile will need to be renamed to “Pico_Toplevel.bit” when copying to Linux to run with the software in the example.

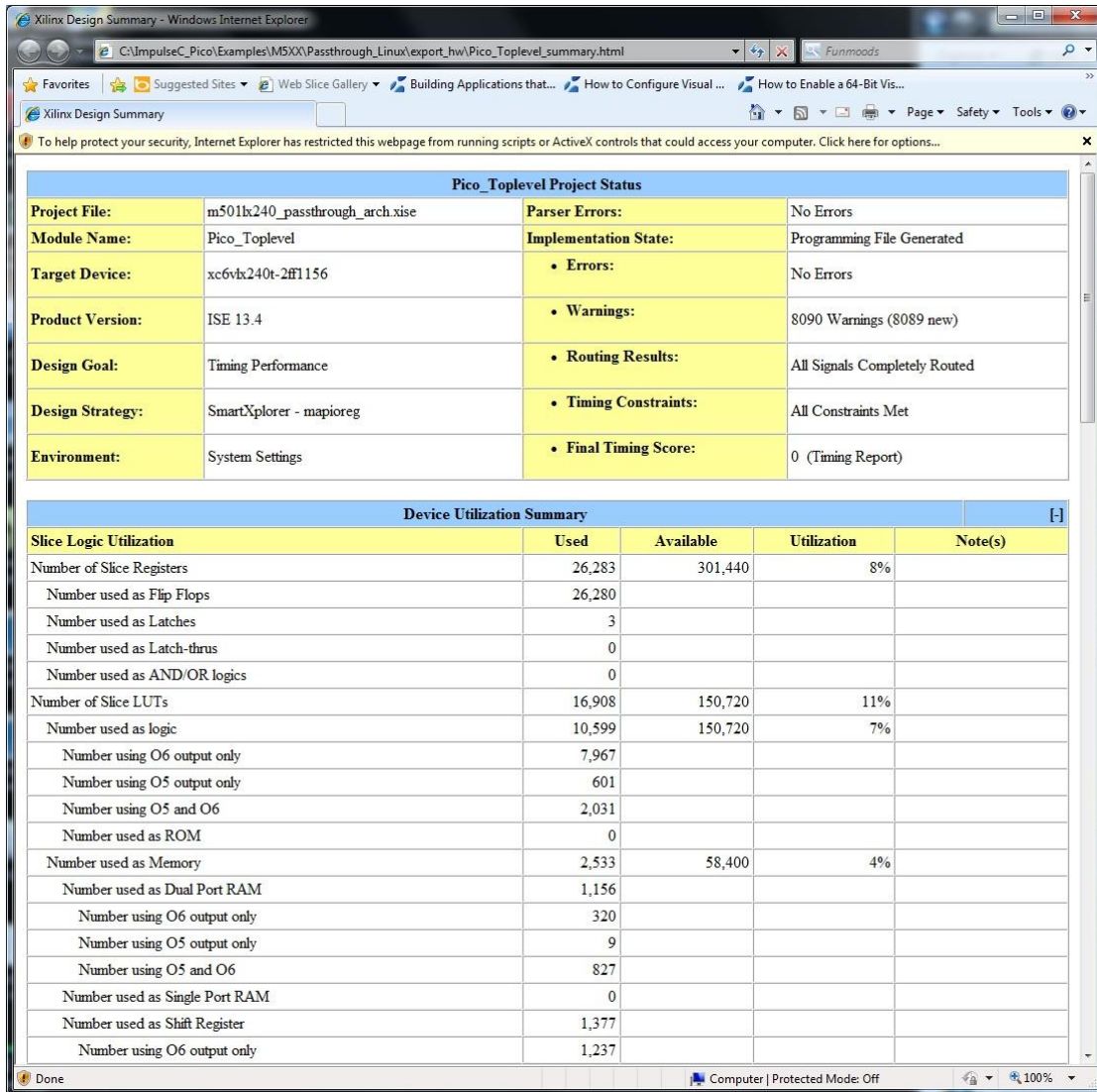


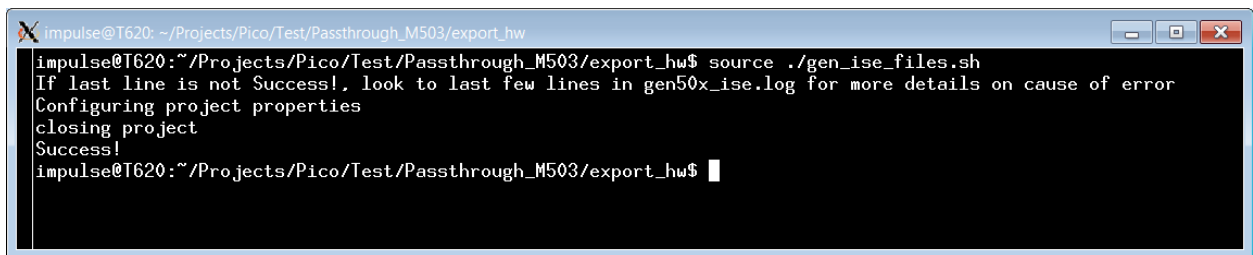
Figure 26 – Xilinx ISE 13.4 compile log file – timing score

8.9.2. Building bitfile under Linux

8.9.2.1. Generate Xilinx ISE Project Files

First generate the ISE project by executing “gen_ise_files.sh”. This process will create an ISE project file as well as copy all the necessary HDL files and CORE Generator components from the Pico installation needed by ISE to generate the FPGA bitfile.

1. Copy the **export_hw** directory to a location on your Host System that has the Linux operating system. (ie Pico/Passthrough/export_hw).
2. Open a command console
3. Change directories to “export_hw”
4. Execute “source ./gen_ise_files.sh” as shown in Figure 27.
5. Verify that the files were created successfully by viewing the end of log file, as shown in Figure 28. “Successful!” should be the last line in the log file (use “cat gen50x_xise.log” to display log).

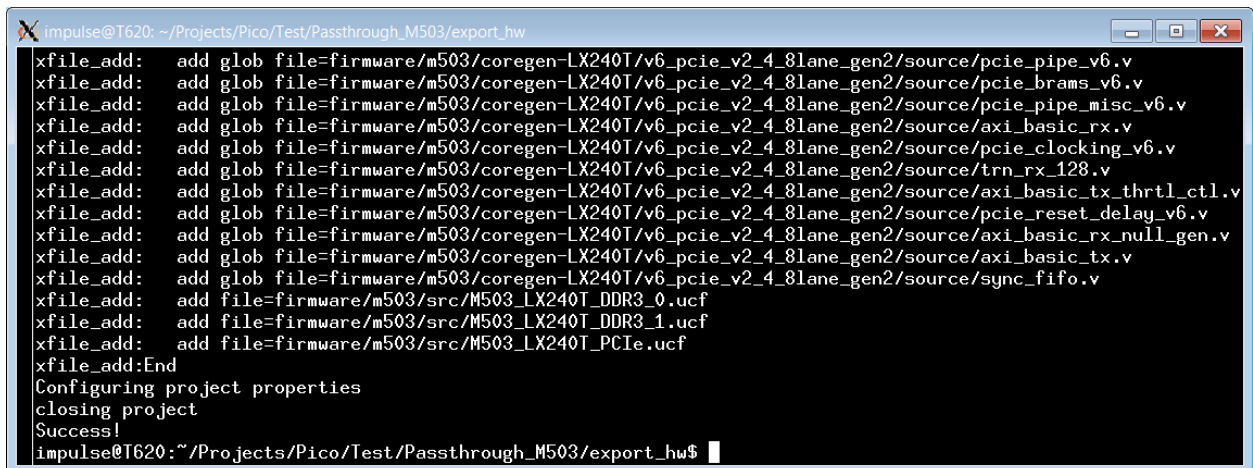


```

impulse@T620: ~/Projects/Pico/Test/Passthrough_M503/export_hw
impulse@T620:~/Projects/Pico/Test/Passthrough_M503/export_hw$ source ./gen_ise_files.sh
If last line is not Successful!, look to last few lines in gen50x_xise.log for more details on cause of error
Configuring project properties
closing project
Success!
impulse@T620:~/Projects/Pico/Test/Passthrough_M503/export_hw$

```

Figure 27 - Generate ISE project files



```

xfile_add: add glob file=firmware/m503/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/source/pcie_pipe_v6.v
xfile_add: add glob file=firmware/m503/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/source/pcie_brams_v6.v
xfile_add: add glob file=firmware/m503/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/source/pcie_pipe_misc_v6.v
xfile_add: add glob file=firmware/m503/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/source/axi_basic_rx.v
xfile_add: add glob file=firmware/m503/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/source/pcie_clocking_v6.v
xfile_add: add glob file=firmware/m503/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/source/trn_rx_128.v
xfile_add: add glob file=firmware/m503/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/source/axi_basic_tx_thrtl_ctl.v
xfile_add: add glob file=firmware/m503/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/source/pcie_reset_delay_v6.v
xfile_add: add glob file=firmware/m503/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/source/axi_basic_rx_null_gen.v
xfile_add: add glob file=firmware/m503/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/source/axi_basic_tx.v
xfile_add: add glob file=firmware/m503/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/source/sync_fifo.v
xfile_add: add file=firmware/m503/src/M503_LX240T_DDR3_0.ucf
xfile_add: add file=firmware/m503/src/M503_LX240T_DDR3_1.ucf
xfile_add: add file=firmware/m503/src/M503_LX240T_PCIE.ucf
xfile_add:End
Configuring project properties
closing project
Success!
impulse@T620:~/Projects/Pico/Test/Passthrough_M503/export_hw$

```

Figure 28 - Expected gen50x_xise.log report

8.9.2.2. Compiling FPGA in Xilinx ISE GUI

1. If you already haven't done so, launch or verify that the license manager is operational and that your license is valid. Contact Xilinx for more information on licensing related to a Linux installation.
2. Open a terminal window and set environment variables by executing the "settings64.sh" script as well as the following variables:
 - a. `source /opt/Xilinx/13.4/ISE_DS/setting64.sh`
 - b. `export PICOBASE=/usr/src/picocomputing-5.0.6.1`
 - c. `export XILINX=/opt/Xilinx/13.4/ISE_DS/ISE`
3. Go to the **export_hw** directory and begin executing the shell scripts to generate the Xilinx ISE project.
 - a. `cd /Pico/Passthrough/export_hw`
 - b. `source gen_ise_files.sh`
4. Launch the Xilinx ISE 13.4 GUI, select the project, regenerate coregent files, and generate the bitfile.
 - a. `/opt/Xilinx/13.4/ISE_DS/ISE/bin/lin64/ise &`

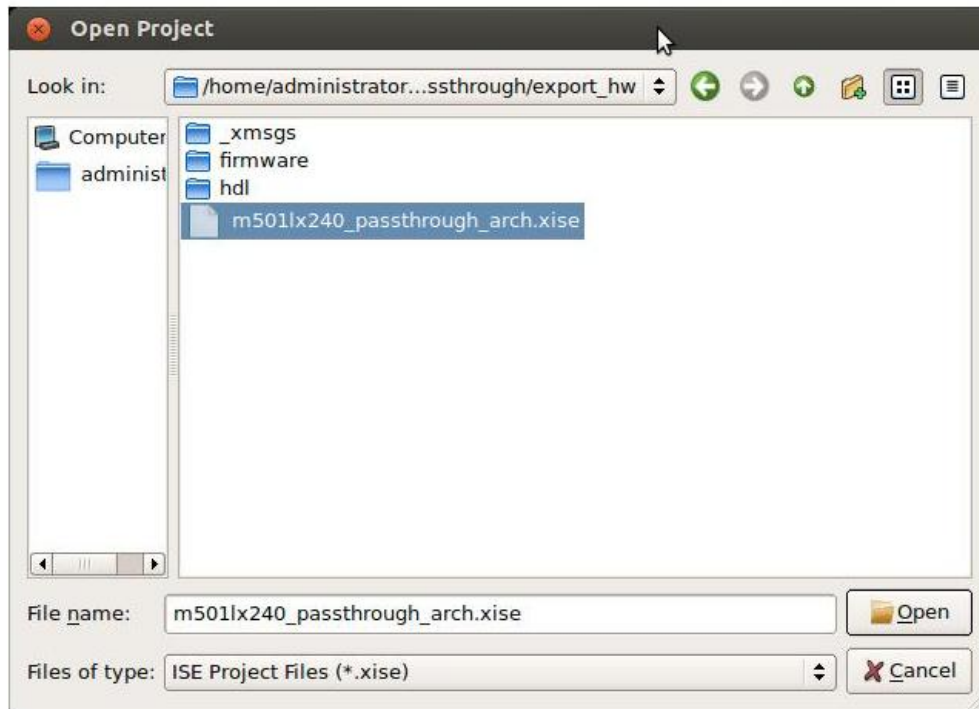


Figure 29 - Select the ISE project

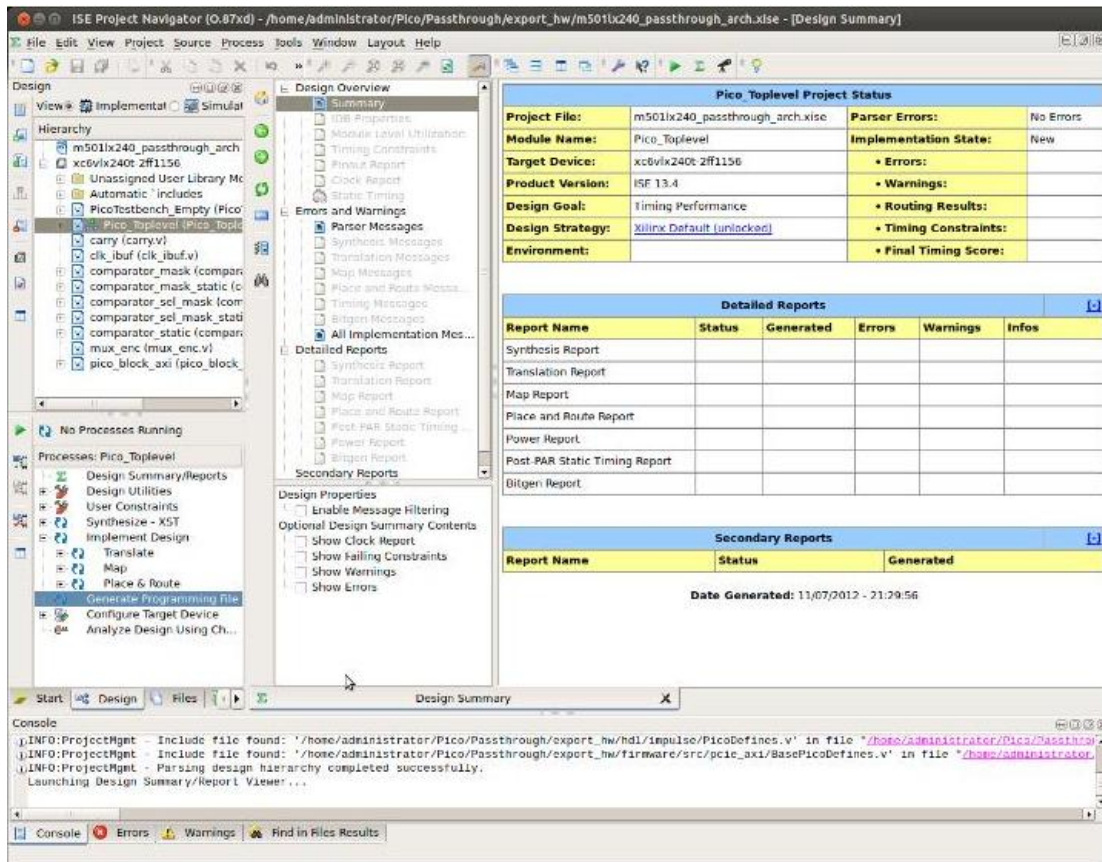


Figure 30 - Initial project status after loading project

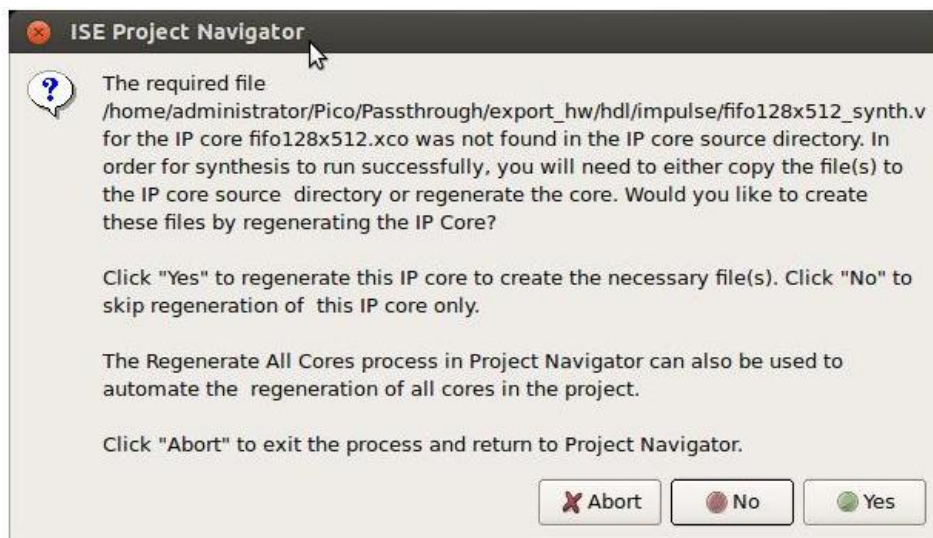


Figure 31 - Regenerate fifo128x512

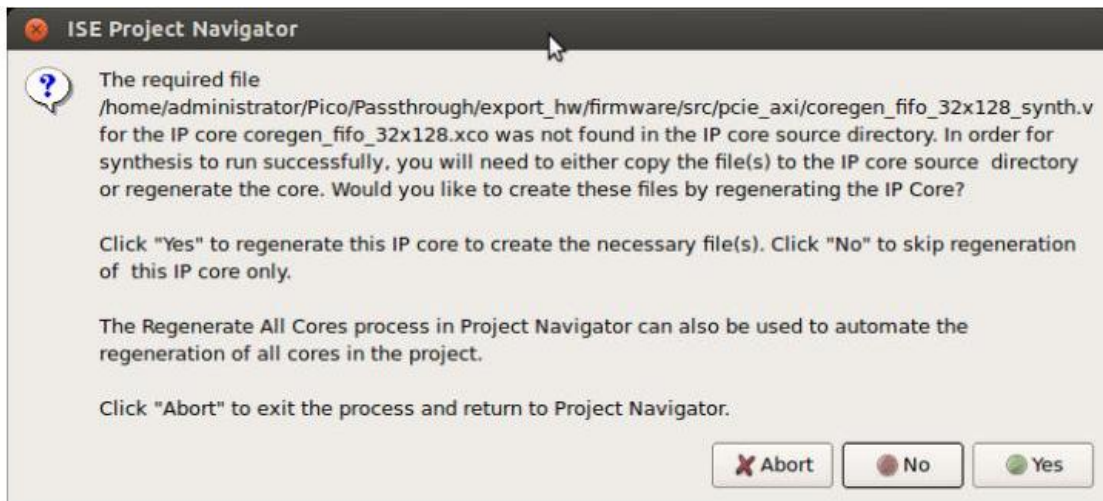


Figure 32 - Regenerate coregen_fifo_32x128

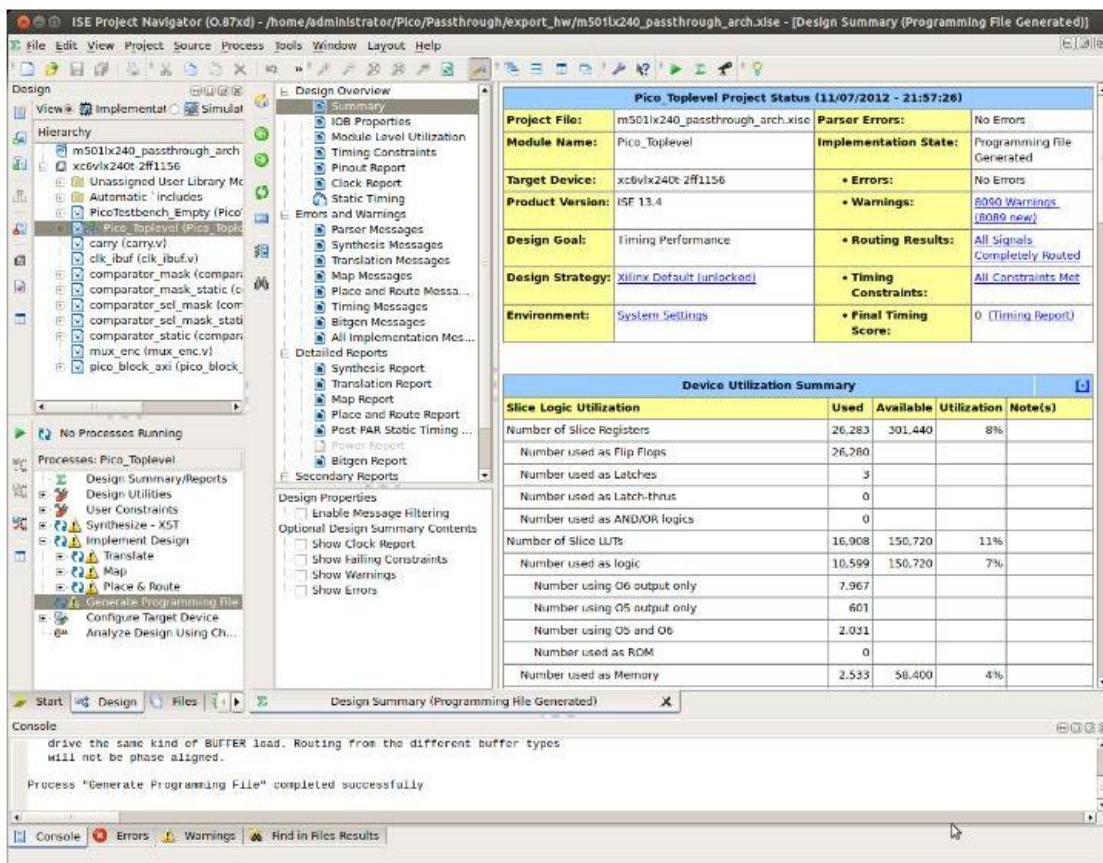


Figure 33 - Place and Route and bitfile generation with timing score equal to zero

8.9.2.3. Compiling FPGA in Xilinx ISE using Command Line

1. Copy the **export_hw** directory to a location on your Host System that has the Ubuntu operating system. (ie Pico/Passthrough/export_hw).

2. If you already haven't done so, launch or verify that the license manager is operational and that your license is valid. Contact Xilinx for more information on licensing related to a Linux installation.
3. Open a terminal window and set environment variables by executing the "settings64.sh" script as well as the following variables:
 - c. `cd /opt/Xilinx/13.4/ISE_DS`
 - d. `sudo ./setting64.sh`
 - e. `export PICOBASE=/usr/src/picocomputing-5.0.6.1`
 - f. `export XILINX=/opt/Xilinx/13.4/ISE_DS/IS`
4. Go to the **export_hw** directory and begin executing the shell scripts to generate then build the FPGA bitfile.
 - g. `cd /Pico/Passthrough/export_hw`
 - h. `source gen_ise_files.sh`
 - i. `source build_m501lx240_passthrough_arch.sh`
5. Verify that the bitfile was generated and that the timing score equals zero.
 - j. `ll *.bit` (should see Pico_Toplevel.bit)
 - k. `grep -i score *.par`

```

administrator@ubuntu: ~/Pico/Passthrough/export_hw
is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
<PicoFramework/app/FrameworkPicoBus/s126i_stream/s0_desc_fifo/U0/xst_fifo_gen
erator/gconvfifo.rf/grf.rf/gntv_or_sync_fifo.men/gdm.dm/Mram_RAM20_RAMD_D1_0>
is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
<PicoFramework/app/FrameworkPicoBus/s126i_stream/s0_desc_fifo/U0/xst_fifo_gen
erator/gconvfifo.rf/grf.rf/gntv_or_sync_fifo.men/gdm.dm/Mram_RAM18_RAMD_D1_0>
is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
<PicoFramework/app/FrameworkPicoBus/s126i_stream/s0_desc_fifo/U0/xst_fifo_gen
erator/gconvfifo.rf/grf.rf/gntv_or_sync_fifo.men/gdm.dm/Mram_RAM16_RAMD_D1_0>
is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
<PicoFramework/app/FrameworkPicoBus/s126i_stream/s0_desc_fifo/U0/xst_fifo_gen
erator/gconvfifo.rf/grf.rf/gntv_or_sync_fifo.men/gdm.dm/Mram_RAM13_RAMD_D1_0>
is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
<PicoFramework/app/FrameworkPicoBus/s126i_stream/s0_desc_fifo/U0/xst_fifo_gen
erator/gconvfifo.rf/grf.rf/gntv_or_sync_fifo.men/gdm.dm/Mram_RAM14_RAMD_D1_0>
is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
<PicoFramework/app/FrameworkPicoBus/s126i_stream/s0_desc_fifo/U0/xst_fifo_gen
erator/gconvfifo.rf/grf.rf/gntv_or_sync_fifo.men/gdm.dm/Mram_RAM15_RAMD_D1_0>
is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:367 - The signal
<PicoFramework/app/FrameworkPicoBus/s126i_stream/s0_desc_fifo/U0/xst_fifo_gen
erator/gconvfifo.rf/grf.rf/gntv_or_sync_fifo.men/gdm.dm/Mram_RAM12_RAMD_D1_0>
is incomplete. The signal does not drive any load pins in the design.
WARNING:PhysDesignRules:2045 - The MMCM_ADV block
<nig_v3_8_DDR3_0/u_infrastructure/u_mmc Adv> has CLKOUT pins that do not
drive the same kind of BUFFER load. Routing from the different buffer types
will not be phase aligned.

Process "Generate Programming File" completed successfully
INFO:ICLTasksC:1850 - process run : Generate Programming File is done.
Run completed.
administrator@ubuntu:~/Pico/Passthrough/export_hw$

```

Figure 34 - Ubuntu Xilinx ISE 13.4 Command Line output

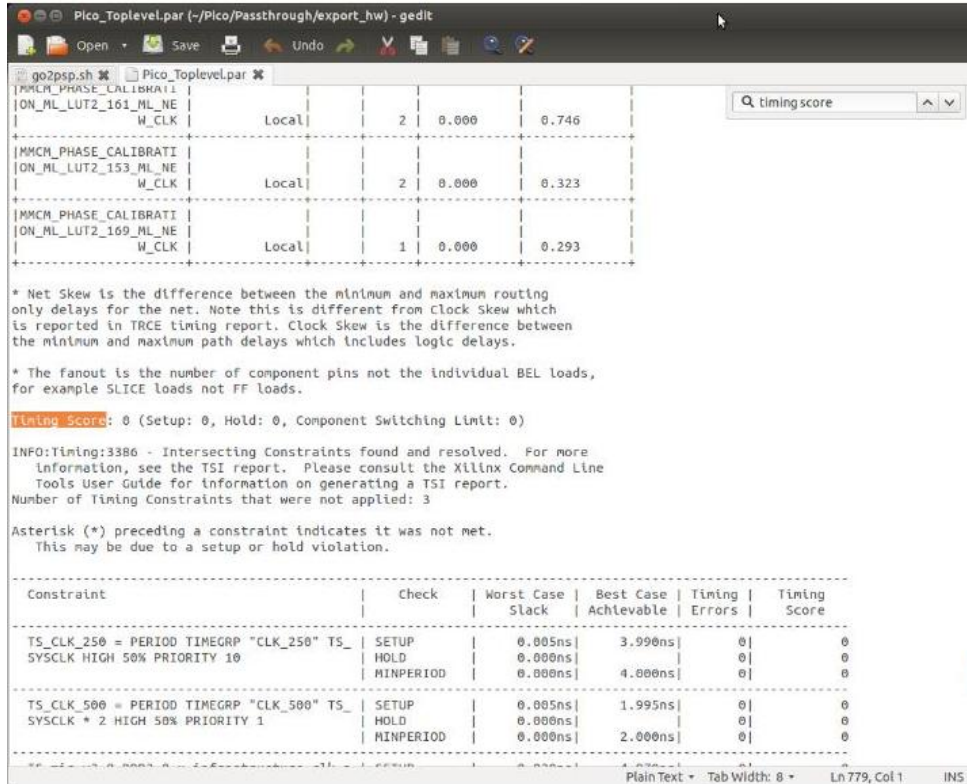


Figure 35 - Xilinx ISE 13.4 results file with timing score equal to zero

8.10. 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.

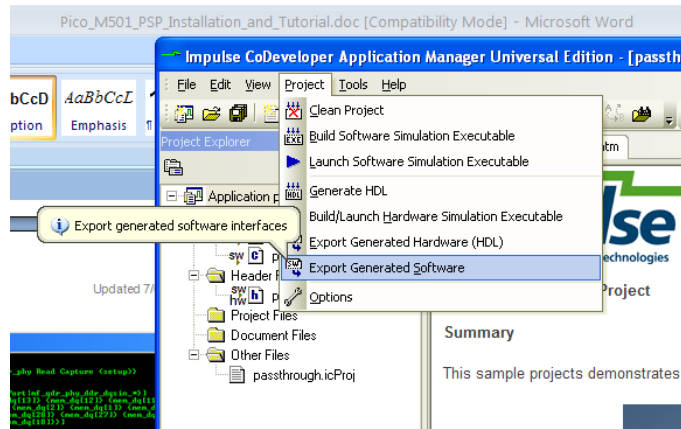


Figure 36 - Export Generated Software

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.

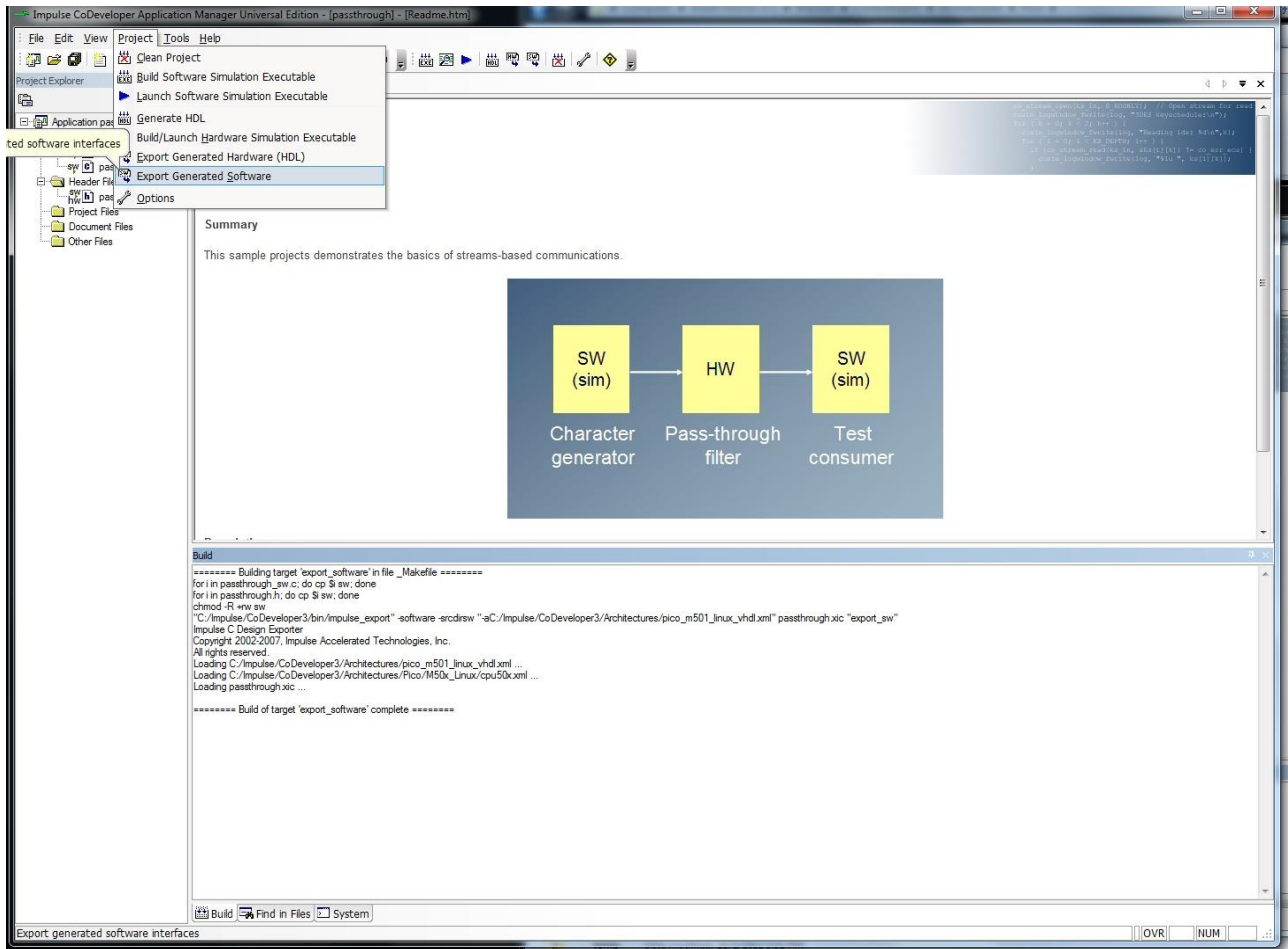


Figure 37 - Build window output

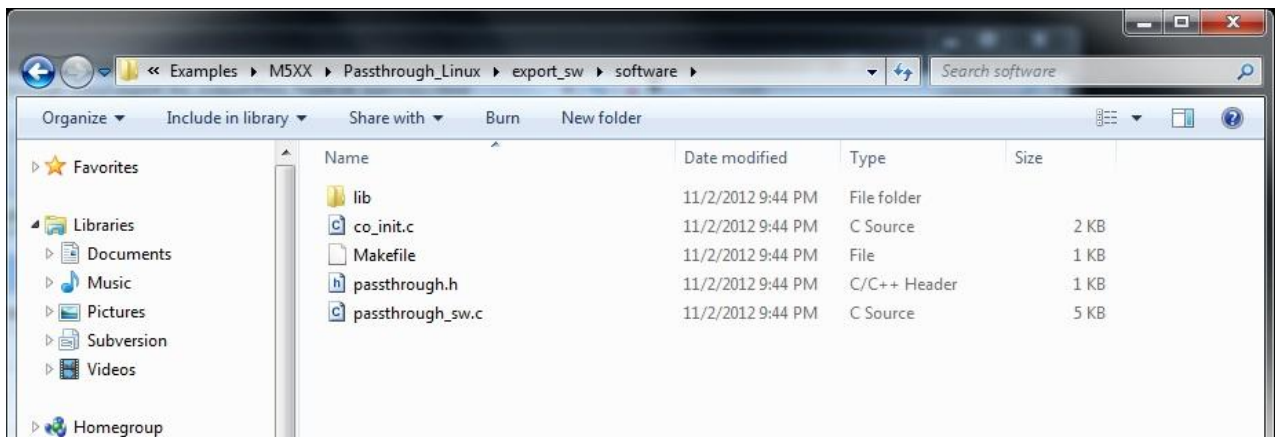


Figure 38 - Exported software directory

8.11. Programming the FPGA

The compiled software application in the “export_sw” directory will program the FPGA on the M501. Please continue to the next section.

8.12. Running Target Executable on the Host System

The Host System will need the following files:

1. Compiled bitfile output from ISE 13.4
2. Input data file
3. Application software source code
4. Makefile to compile the software

Please copy the following files and directory over to the Host System (figure 30). Create a folder (ie Pico) and copy the following to that folder.

1. **export_hw** directory which includes “pico_toplevel.bit” bitfile
2. **export_sw** directory which includes the Makefile and software application source code
3. **filter_in.dat** file which will be used to input stimulus to the FPGA via the software application.
4. Once the files are copied, please copy **filter_in.dat** to the **export_sw** directory. The application software will expect the input data file to be present in that directory.

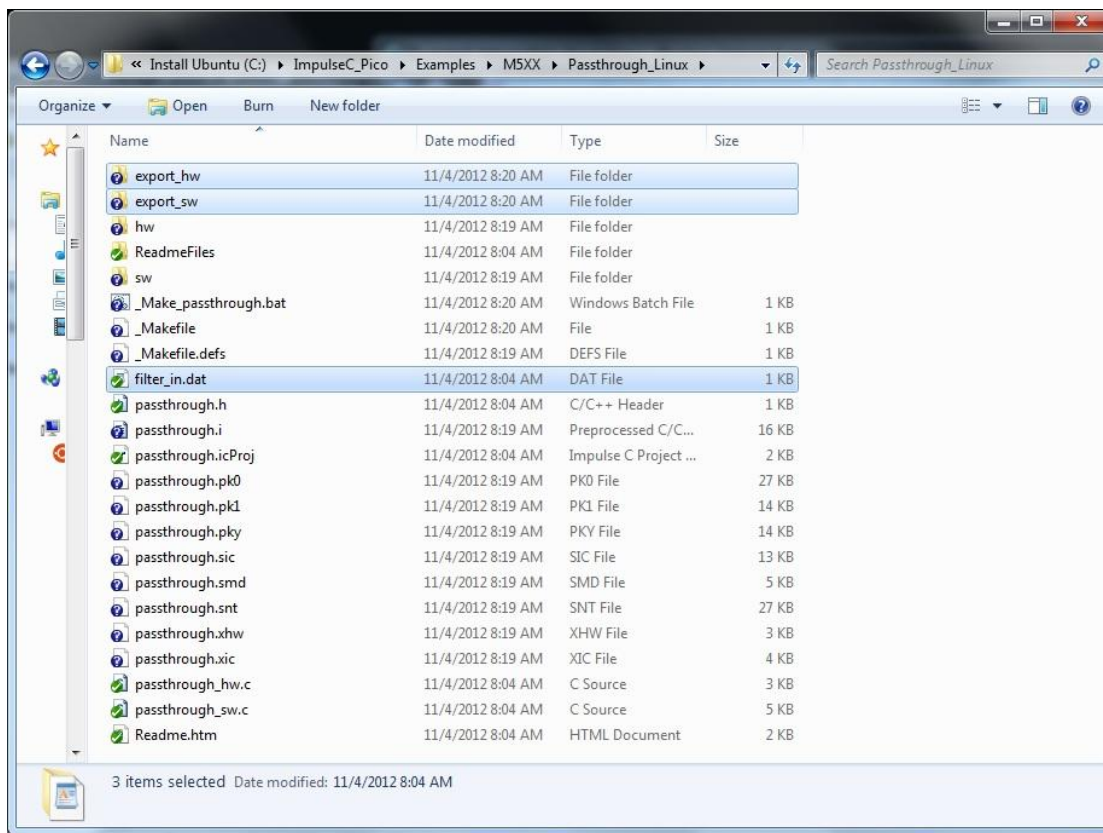
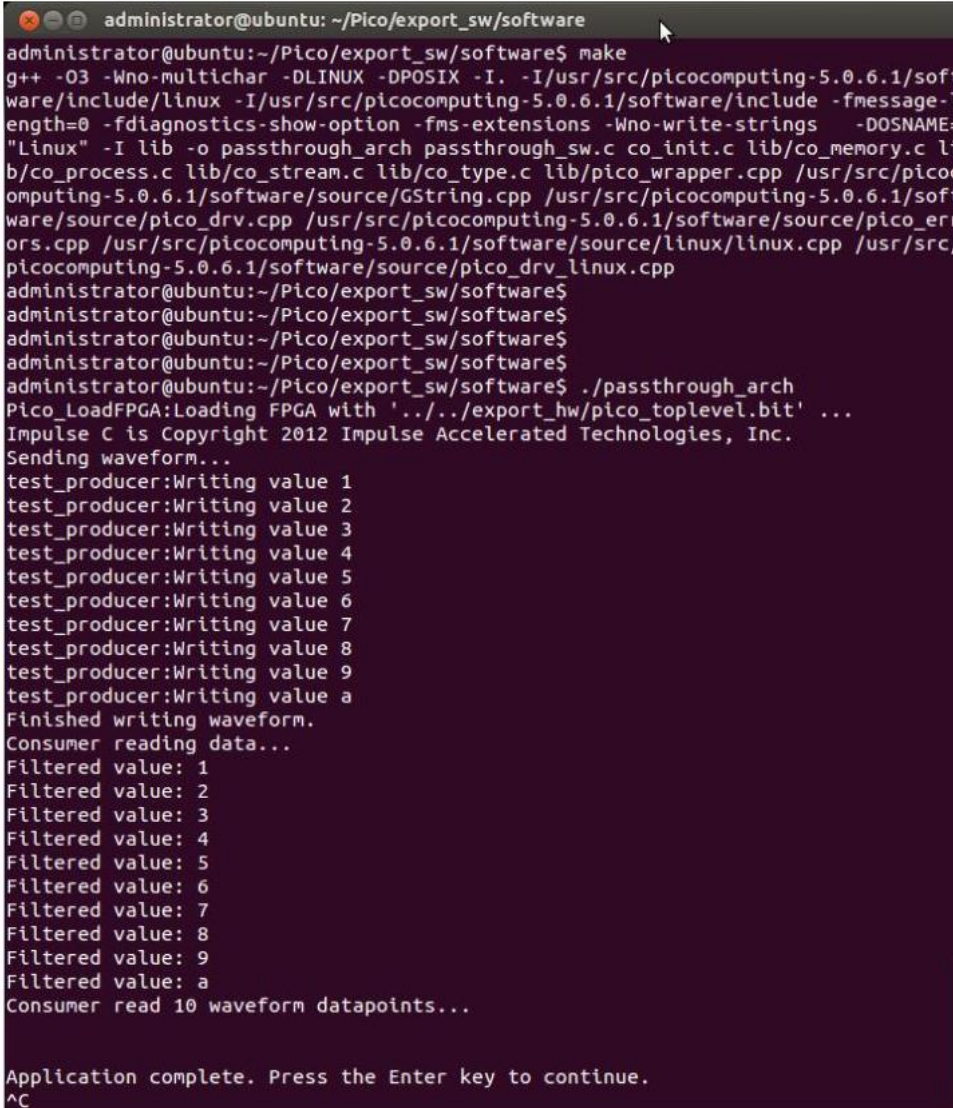


Figure 39 - Files & Directories to be copied to the Host System

The software application will load the FPGA every time it is executed and report the results upon completion. Please note that the process of loading the FPGA may take up to 20 seconds due to the driver re-start sequence after the FPGA bitfile has been transferred to the card.

1. Open a terminal window.
2. Navigate to the location of your copied files and directories:
(ie. `cd Pico/export_sw/software`).
3. Run “make”
4. Execute the generated application: “`./passthrough_arch`”



```

administrator@ubuntu: ~/Pico/export_sw/software
administrator@ubuntu:~/Pico/export_sw/software$ make
g++ -O3 -Wno-multichar -DLINUX -DPOSIX -I. -I/usr/src/picocomputing-5.0.6.1/software/include/linux -I/usr/src/picocomputing-5.0.6.1/software/include -fmessage-length=0 -fdiagnostics-show-option -fms-extensions -Wno-write-strings -DOSNAME="Linux" -I lib -o passthrough_arch passthrough_sw.c co_init.c lib/co_memory.c lib/co_process.c lib/co_stream.c lib/co_type.c lib/pico_wrapper.cpp /usr/src/picocomputing-5.0.6.1/software/source/GString.cpp /usr/src/picocomputing-5.0.6.1/software/source/pico_drv.cpp /usr/src/picocomputing-5.0.6.1/software/source/pico_errors.cpp /usr/src/picocomputing-5.0.6.1/software/source/linux/linux.cpp /usr/src/picocomputing-5.0.6.1/software/source/pico_drv_linux.cpp
administrator@ubuntu:~/Pico/export_sw/software$
administrator@ubuntu:~/Pico/export_sw/software$
administrator@ubuntu:~/Pico/export_sw/software$
administrator@ubuntu:~/Pico/export_sw/software$ ./passthrough_arch
Pico_LoadFPGA:Loading FPGA with '../export_hw/pico_toplevel.bit' ...
Impulse C is Copyright 2012 Impulse Accelerated Technologies, Inc.
Sending waveform...
test_producer:Writing value 1
test_producer:Writing value 2
test_producer:Writing value 3
test_producer:Writing value 4
test_producer:Writing value 5
test_producer:Writing value 6
test_producer:Writing value 7
test_producer:Writing value 8
test_producer:Writing value 9
test_producer:Writing value a
Finished writing waveform.
Consumer reading data...
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
Consumer read 10 waveform datapoints...

Application complete. Press the Enter key to continue.
^C

```

Figure 40 - Exported SW executed on target platform

9.0 Memtest Example and Tutorial

9.1. 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.

9.2. Memtest Example Description

The Memtest example is a CoDeveloper project that includes three source files. A brief description of the files and their functions is outlined in the following sections. This example targets existing external DDR3 memory available on the M501 FPGA module.

1. memtest.h
2. memtest_hw.c
3. memtest_sc.c

The Memtest example allows a stream to write to DDR3 memory. The Impulse User module will read from DDR3, increment the data by one, and write it back out to a stream. The software application generates the data and checks the data returned from the FPGA.

9.2.1. Overview of memtest.h

The figure below is a screen shot of header file. It defines the stream widths (STREAMWIDTH) as 128 bits. It also defines the size of the target memory (MEMORY_SIZE) that will be used by the co_memory API calls.

```

1 //
2 //
3 // Change BUFSIZE to match your desired stream depth size
4 //
5 // Change STREAMWIDTH to match your desired stream width
6 //
7 #define STREAMDEPTH 4          /* buffer size for FIFO in hardware */
8 #define STREAMWIDTH 128
9 #define NUM_ELEMENTS 8192
10 #define MEMORY_SIZE (8192*sizeof(int64))
11

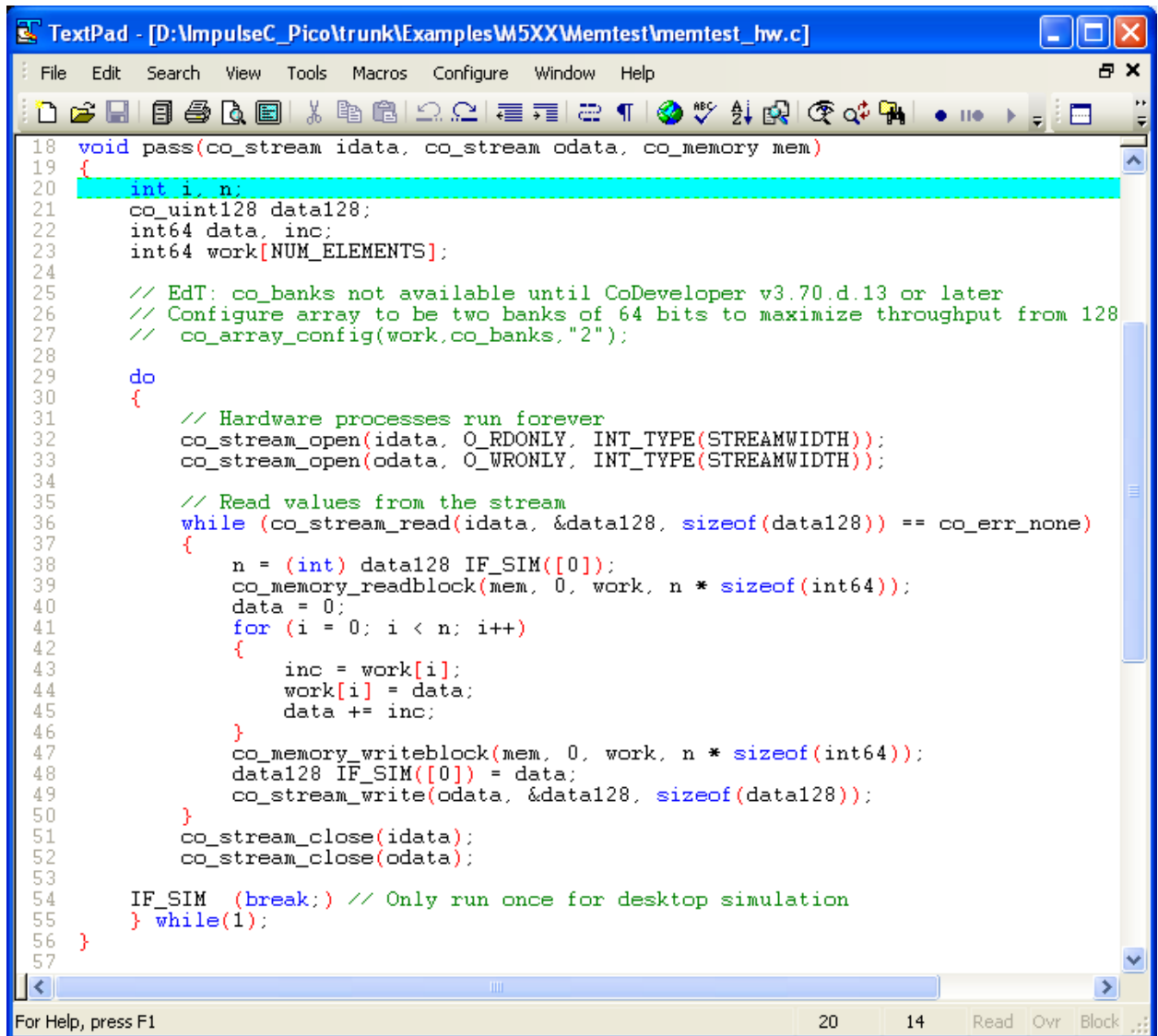
```

Figure 41 - memtest.h

9.2.2. Overview of memtest_hw.c

The memtest_hw.c file is the user defined function that accesses external DDR3 memory. The figure below shows the user defined function, *pass*, that will be implemented on the FPGA. On line 18, the ports are defined as input stream *idata*, output stream *odata*, and memory port *mem*.

In this example, the function *pass* will receive, on stream *idata*, the number of words to read from external DDR3. The “do” loop on line 29 indicates that this hardware process run continuously. Input and output streams, *idata* and *odata*, are open as shown on lines 32 & 33. Line 36 is an Impulse API to read in data from the stream. Line 39 shows the Impulse API to read a block of data from external memory. The “for” loop on line 41 operates on each word as it is received from external memory and is also incremented by one. Once all data had been read and incremented, it is written back to external memory as shown on line 47. The data is also written back out on the stream, *odata*, as shown on line 49. Once all data and processing is complete, the input and output streams are closed.



```

18 void pass(co_stream idata, co_stream odata, co_memory mem)
19 {
20     int i, n;
21     co_uint128 data128;
22     int64 data, inc;
23     int64 work[NUM_ELEMENTS];
24
25     // EdT: co_banks not available until CoDeveloper v3.70.d.13 or later
26     // Configure array to be two banks of 64 bits to maximize throughput from 128
27     // co_array_config(work, co_banks, "2");
28
29     do
30     {
31         // Hardware processes run forever
32         co_stream_open(idata, O_RDONLY, INT_TYPE(STREAMWIDTH));
33         co_stream_open(odata, O_WRONLY, INT_TYPE(STREAMWIDTH));
34
35         // Read values from the stream
36         while (co_stream_read(idata, &data128, sizeof(data128)) == co_err_none)
37         {
38             n = (int) data128 IF_SIM([0]);
39             co_memory_readblock(mem, 0, work, n * sizeof(int64));
40             data = 0;
41             for (i = 0; i < n; i++)
42             {
43                 inc = work[i];
44                 work[i] = data;
45                 data += inc;
46             }
47             co_memory_writeblock(mem, 0, work, n * sizeof(int64));
48             data128 IF_SIM([0]) = data;
49             co_stream_write(odata, &data128, sizeof(data128));
50         }
51         co_stream_close(idata);
52         co_stream_close(odata);
53
54         IF_SIM (break;) // Only run once for desktop simulation
55     } while(1);
56 }
57

```

Figure 42 - User defined funtion in memtest_hw.c

The configuration process defines the processes and ports that can be accessed by the user defined function **pass**. Line 58 is the configuration **config_memtest**. Line 60 thru 62 defined the available ports while line 64 & 65 define processes. Lines 67 shows the Impulse defined **co_memory** with type "mc0" and size **MEMORY_SIZE**. Lines 68 & 69 shows the Impulse defined **co_streams** of widths **STREAMWIDTH** and depth **STREAMDEPTH** (both defined in the header file). The configuration statement also defines the testbench process (line 71) as well as the user process (line 77). Line 86 is necessary when using **co_memory**.

```

57
58 void config_memtest(void *arg)
59 {
60     co_stream ipassdata;
61     co_stream opassdata;
62     co_memory aemem;
63
64     co_process pass_process;
65     co_process testbench_process;
66
67     aemem = co_memory_create("workmem", "mc0", MEMORY_SIZE);
68     ipassdata = co_stream_create("idata", INT_TYPE(STREAMWIDTH), STREAMDEPTH);
69     opassdata = co_stream_create("odata", INT_TYPE(STREAMWIDTH), STREAMDEPTH);
70
71     testbench_process = co_process_create("testbench", (co_function) Testbench,
72     3,
73     ipassdata,
74     opassdata,
75     aemem);
76
77     pass_process = co_process_create("pass", (co_function) pass,
78     3,
79     ipassdata,
80     opassdata,
81     aemem);
82
83     co_process_config(pass_process, co_loc, "PE0");
84 }
85
86 co_architecture co_initialize(int param)
87 {
88     return (co_architecture_create("memtest_arch", "Generic", config_memtest, (voi
89 }
90

```

Figure 43 - Configuration in memtest_hw.c

9.2.3. Overview of memtest_sw.c

The software application provides stimulus and receives return data for post processing of the user defined functions. Line 20 defines a process “Testbench” which also has ports **ipassdata**, **opassdata** and **mem**. The port **opassdata** will send data to the user defined function. In this example, line 31 generates data to be written to external memory. Similar to the hardware process, streams must be open in order to read or write to them. Lines 38 & 39 open streams. Line 42 calls the Impulse API `co_memory` to write a block of data to external memory. Line 46 sends the number of bytes to be read by the hardware process while line 49 waits for data to be return by the hardware process. Line 55 reads data from external memory and that data is used to verify the data received on the input stream as shown in the “for” loop on line 58. Finally, the streams are closed as shown on lines 66 & 67.

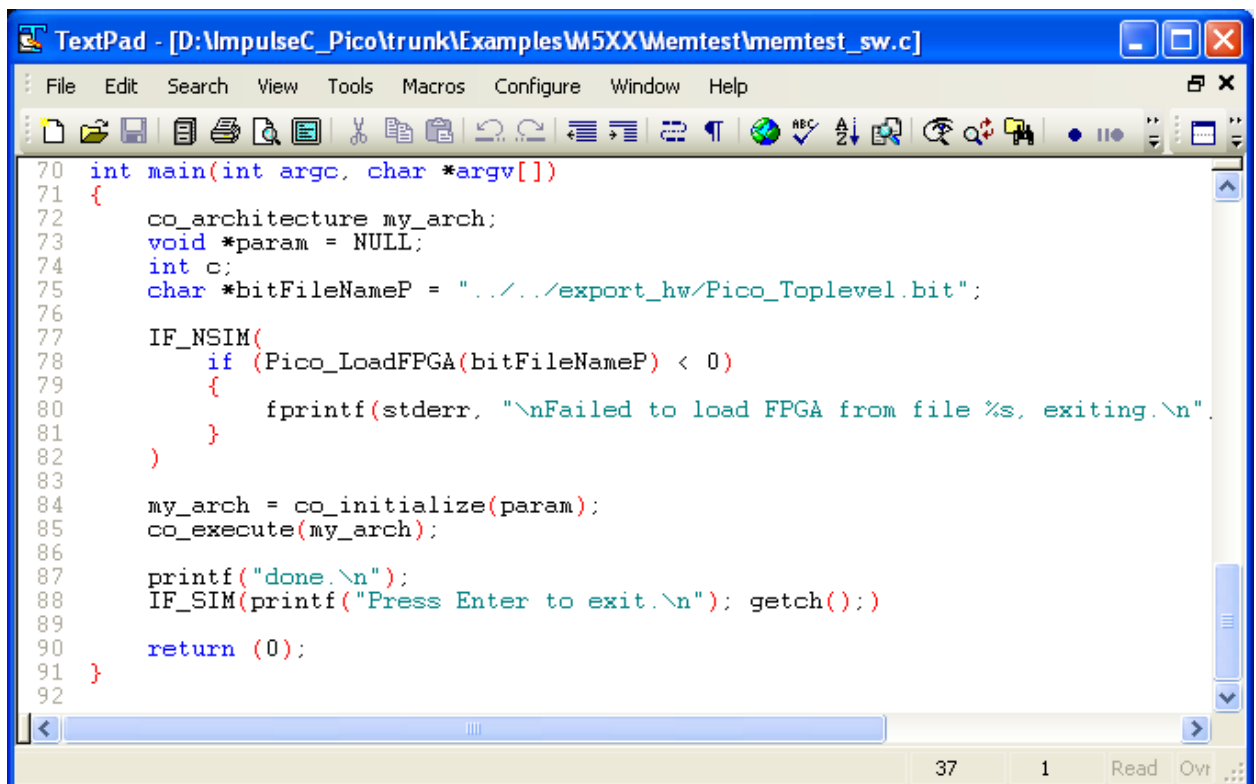

```

19
20 void Testbench(co_stream ipassdata, co_stream opassdata, co_memory mem)
21 {
22     int i;
23     co_uint128 data;
24     int64 test[NUM_ELEMENTS];
25
26     data[0] = 0;
27     data[1] = 0;
28     data[2] = 0;
29     data[3] = 0;
30
31     for (i = 0; i < NUM_ELEMENTS; i++)
32     {
33         test[i] = i;
34         data[0] += i;
35     }
36     printf("sw total: %d\n", (int) data[0]);
37
38     co_stream_open(ipassdata, O_WRONLY, INT_TYPE(STREAMWIDTH));
39     co_stream_open(opassdata, O_RDONLY, INT_TYPE(STREAMWIDTH));
40
41     printf("writing memory\n");
42     co_memory_writeblock(mem, 0, test, sizeof(test));
43
44     printf("writing stream\n");
45     data[0] = NUM_ELEMENTS;
46     co_stream_write(ipassdata, &data, sizeof(data));
47
48     printf("reading stream\n");
49     co_stream_read(opassdata, &data, sizeof(data));
50     printf("hw total: %d\n", (int) data[0]);
51
52     memset(test, 0, sizeof(test));
53
54     printf("reading memory\n");
55     co_memory_readblock(mem, 0, test, sizeof(test));
56
57     data[0] = 0;
58     for (i = 0; i < NUM_ELEMENTS; i++)
59     {
60         if (test[i] != data[0])
61             printf("error test[%d]=%d (expecting %d)\n", i, (int) test[i],
62                   (int) data[0]);
63         data[0] += i;
64     }
65
66     co_stream_close(ipassdata);
67     co_stream_close(opassdata);
68 }
69

```

Figure 44 - User defined stimulus in memtest_sw.c

The main function is shown below. The first step for the software application is to load the FPGA bitfile using the Pico API `Pico_LoadFPGA` as shown on line 78. If the file does not exist or an error is encountered, then an error is reported. Otherwise the testbench is executed.



```
70 int main(int argc, char *argv[])
71 {
72     co_architecture my_arch;
73     void *param = NULL;
74     int c;
75     char *bitFileNameP = ".../export_hw/Pico_Toplevel.bit";
76
77     IF_NSIM(
78         if (Pico_LoadFPGA(bitFileNameP) < 0)
79         {
80             fprintf(stderr, "\nFailed to load FPGA from file %s, exiting.\n");
81         }
82     )
83
84     my_arch = co_initialize(param);
85     co_execute(my_arch);
86
87     printf("done.\n");
88     IF_SIM(printf("Press Enter to exit.\n"); getch();)
89
90     return (0);
91 }
92
```

Figure 45 - Main program in memtest_sw.c

9.3. CoDeveloper Project Files

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

- memtest.icProj – CoDeveloper project file
- memtest_hw.c – Source code for hardware process
- memtest_sw.c – Source code for software processes
- memtest.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 memtest_hw.c file. The default example defines the steam to be the width of 128 bit data bus.

```

1 ////////////////////////////////////////////////////
2 //
3 // Change BUFSIZE to match your desired stream depth size
4 //
5 // Change STREAMWIDTH to match your desired stream width
6 //
7 #define STREAMDEPTH 4 /* buffer size for FIFO in hardware */
8 #define STREAMWIDTH 128
9 #define NUM_ELEMENTS 8192
10 #define MEMORY_SIZE (8192*sizeof(int64))
11 //

```

Figure 46 - Impulse C Header File

```

1 ////////////////////////////////////////////////////////////////////
2 // memtest_hw.c: includes the hardware process and configuration
3 // function.
4 //
5 // See additional comments in memtest.h.
6 //
7
8 #include <stdio.h>
9 #include "co.h"
10 #include "cosim_log.h"
11 #include "memtest.h"
12
13 extern void Testbench(co_stream ipassdata, co_stream opassdata, co_memory mem);
14
15 ////////////////////////////////////////////////////////////////////
16 // This is the hardware process.
17 //
18 void pass(co_stream idata, co_stream odata, co_memory mem)
19 {
20     int i, n;
21     co_uint128 data128;
22     int64 data, inc;
23     int64 work[NUM_ELEMENTS];
24
25     // EdT: co_banks not available until CoDeveloper v3.70.d.13 or later
26     // Configure array to be two banks of 64 bits to maximize throughput from 128 bit memory.
27     // co_array_config(work,co_banks,"2");
28
29     do
30     {
31         // Hardware processes run forever
32         co_stream_open(idata, O_RDONLY, INT_TYPE(STREAMWIDTH));
33         co_stream_open(odata, O_WRONLY, INT_TYPE(STREAMWIDTH));
34
35         // Read values from the stream
36         while (co_stream_read(idata, &data128, sizeof(data128)) == co_err_none)
37         {
38             n = (int) data128 IF_SIM([0]);
39             co_memory_readblock(mem, 0, work, n * sizeof(int64));
40             data = 0;
41             for (i = 0; i < n; i++)
42             {
43                 inc = work[i];
44                 work[i] = data;
45                 data += inc;
46             }
47             co_memory_writeblock(mem, 0, work, n * sizeof(int64));
48             data128 IF_SIM([0]) = data;
49             co_stream_write(odata, &data128, sizeof(data128));
50         }
51         co_stream_close(idata);
52         co_stream_close(odata);

```

C source file length: 2317 lines: 90 Ln:1 Col:1 Sel:0 UNIX ANSI INS

Figure 47 - ImpulseC Hardware File

9.4. Opening Project

Open the CoDeveloper project file 'Memtest.icProj' by selecting and pressing 'Enter' or by double-clicking it:

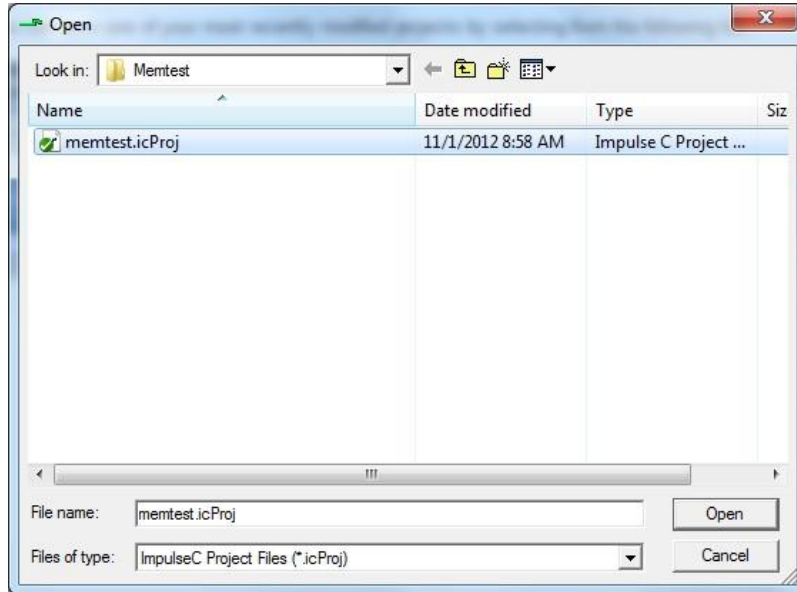


Figure 48 - Opening a project in CoDeveloper

9.5. Building Desktop Simulation Executable

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

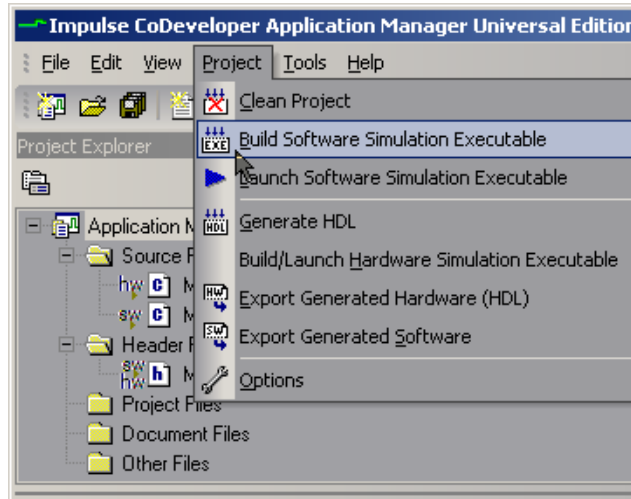


Figure 49 - Build Simulation Desktop in CoDeveloper using pull-down menu

Or via toolbar:

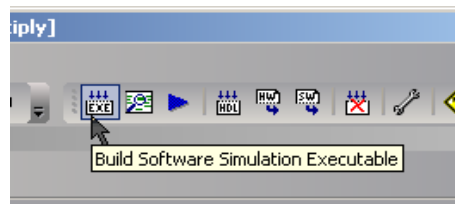


Figure 50 - Build Simulation Desktop in CoDeveloper using toolbar icon

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

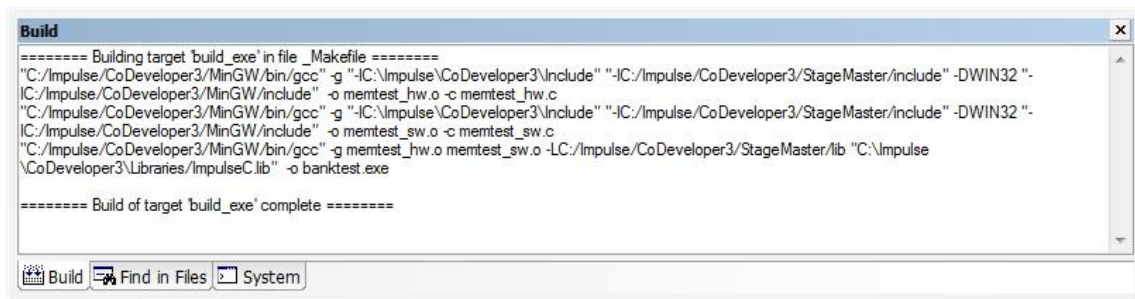


Figure 51 - Output within the CoDeveloper IDE build window

9.6. Running Desktop Simulation Executable

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

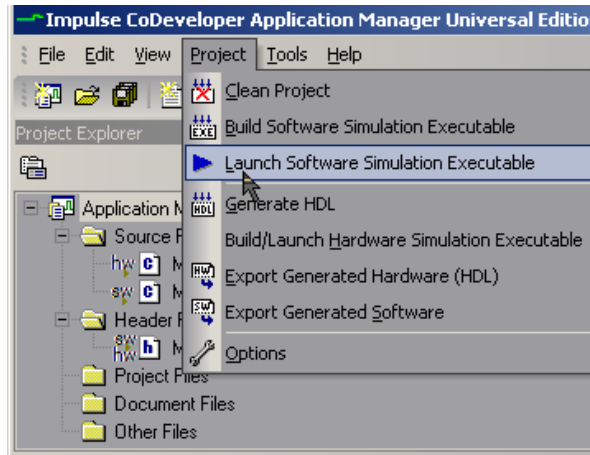


Figure 52 - Launch software simulation window using pull-down menu

Or via toolbar:

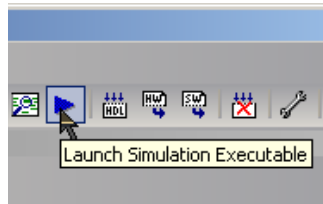


Figure 53 - Launch software simulation using toolbar icon

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

```
C:\Windows\system32\cmd.exe
"C:\ImpulseC_Pico\Examples\M5XX\Mentest\banktest.exe"
sw total: 33550336
writing memory
writing stream
reading stream
hw total: 33550336
reading memory
done.
Press Enter to exit.
-
```

Figure 54 - Pop-up window during desktop simulation

9.7. 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 Linux (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 Memtest example:

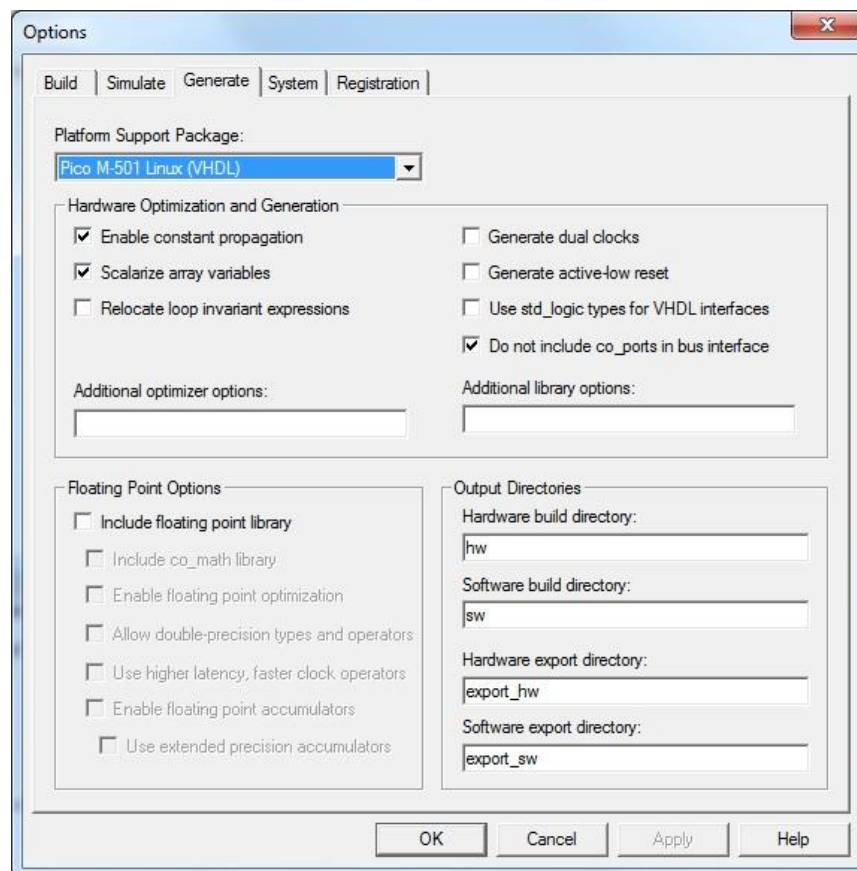


Figure 55 - Project setup to pick Platform Support Package

9.8. Generating Hardware

Generate hardware via “Project” menu:

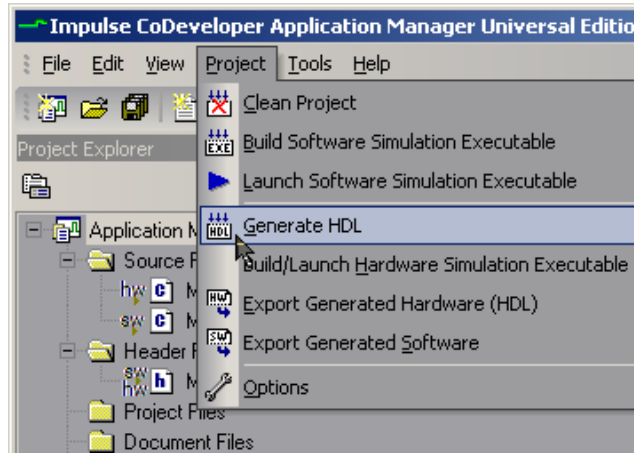


Figure 56 - Generate HDL using pull-down menu

Or via toolbar:

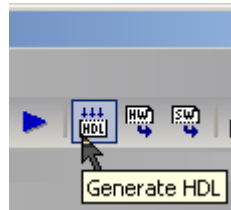


Figure 57 - Generate HDL using toolbar icon

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:


```

Build
Copyright 2002-2009, Impulse Accelerated Technologies, Inc.
All rights reserved.
Generating pe0/pass ...
Component generation complete.
---Software activated---
"C:/Impulse/CoDeveloper3/bin/impulse_arch" "-aC:/Impulse/CoDeveloper3/Architectures/pico_m501_linux_vhdl.xml" -no_port_bus_connect -swdirsw -
files "memtest_comp.vhd memtest_top.vhd" -srcs "memtest_hw.c memtest.h" memtest.xic hw/memtest_top.vhd
Impulse C HDL Design Generator
Copyright 2002-2012, Impulse Accelerated Technologies, Inc.
All rights reserved.
Loading C:/Impulse/CoDeveloper3/Architectures/pico_m501_linux_vhdl.xml ...
Loading C:/Impulse/CoDeveloper3/Architectures/Pico/M50x_Linux/bus50x.xml ...
Loading C:/Impulse/CoDeveloper3/Architectures/Pico/M50x_Linux/axi_ddr.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 memtest.xic ...
pspFile=pico_m501_linux_vhdl.xml
board=m501
connections2=memory mc0 axidrr {} mc0 {}
connections2=stream in p_testbench_ipassdata 128 picobus sp {} idata {}
connections2=stream out p_testbench_opassdata 128 picobus sp {} odata {}
connections2=stream in config 32 picobus sp {} config {}
running picoCustomizeUcf
connections = {memory mc0 axidrr {} mc0 {} 0} {stream in p_testbench_ipassdata 128 picobus sp {} idata {} 1} {stream out p_testbench_opassdata 128
picobus sp {} odata {} 1} {stream in config 32 picobus sp {} config {} 2}
con = memory mc0 axidrr {} mc0 {} 0, 0
con = stream in p_testbench_ipassdata 128 picobus sp {} idata {} 1, 1
con = stream out p_testbench_opassdata 128 picobus sp {} odata {} 1, 1
con = stream in config 32 picobus sp {} config {} 2, 2
Design generation complete
chmod -R +rw hw
mkdir sw
"C:/Impulse/CoDeveloper3/bin/impulse_lib" "-aC:/Impulse/CoDeveloper3/Architectures/pico_m501_linux_vhdl.xml" -hwdirhw -files "memtest_sw.c"
memtest.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_linux_vhdl.xml ...
Loading C:/Impulse/CoDeveloper3/Architectures/Pico/M50x_Linux/cpu50x.xml ...
Loading C:/Impulse/CoDeveloper3/Architectures/VHDL/Generic/Generic/system.xml ...
Loading memtest.xic ...
for i in memtest_sw.c; do cp $i sw; done
for i in memtest.h; do cp $i sw; done
chmod -R +rw sw

==== Build of target 'build' complete =====
Build Find in Files System

```

Figure 58 - Build window output

9.9. Exporting Hardware

Export hardware via “Project” menu:

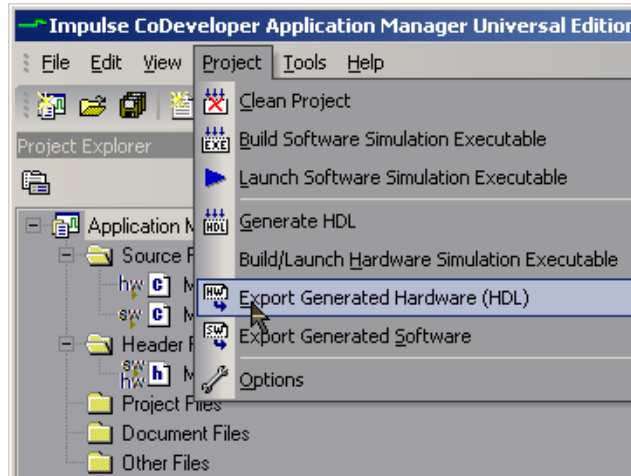


Figure 59 - Export Generated Hardware (HDL) using pull-down menu

Or via toolbar:

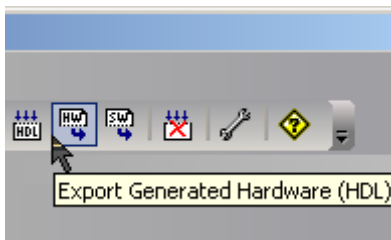


Figure 60 - Export Generated Hardware (HDL) using toolbar icon

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:

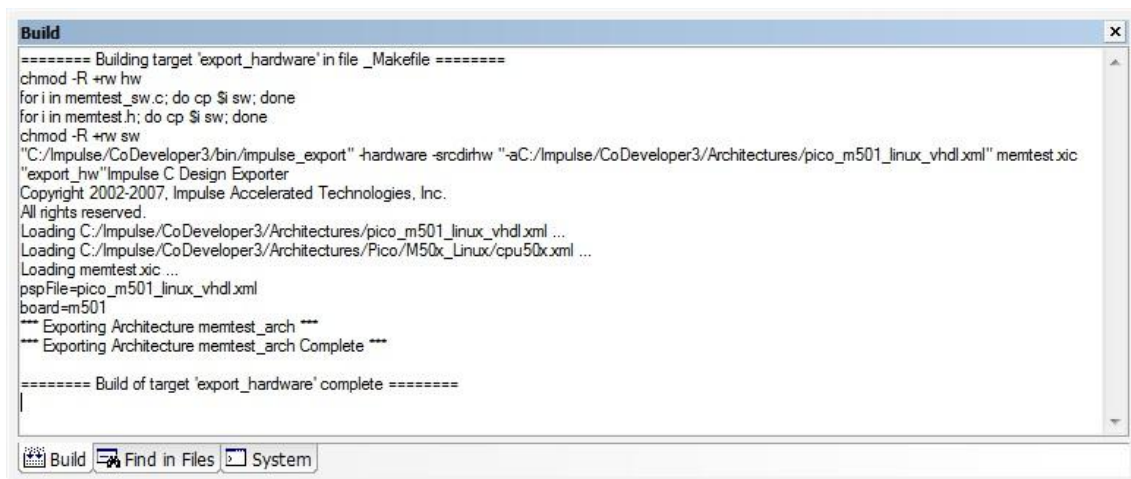


Figure 61 - Build window output

9.10. Compiling FPGA in Xilinx ISE 13.4

After exporting hardware, under the specified hardware export directory will be a directory structure that includes all necessary files for building the FPGA binary.

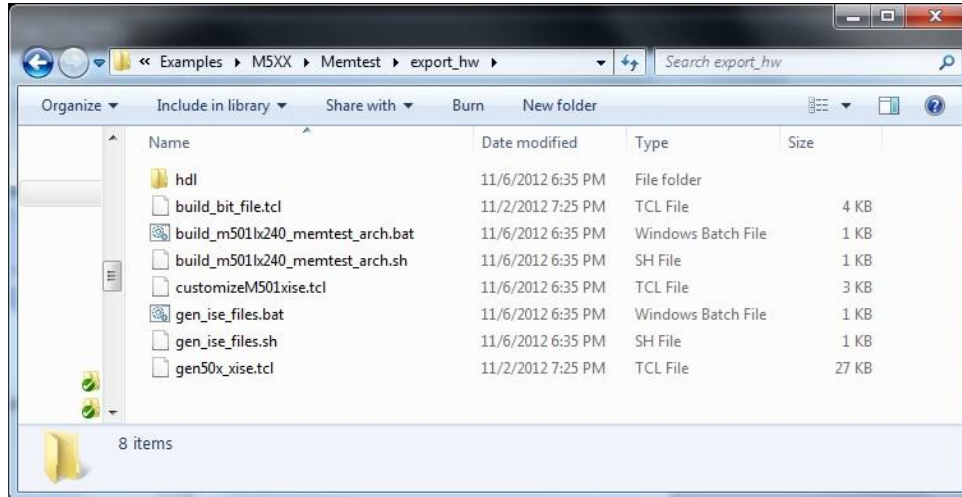


Figure 62 - Compiling FPGA in Quartus directory structure

At this point either Windows or Linux may be used to produce the bitfile running Xilinx ISE either through the GUI or from command line. Windows and Linux both use the following similar steps. Choose the one that suits your needs.

Windows:

1. Generate the ISE project files by running “gen_ise_files.bat”
2. Build the bit file using Xilinx ISE either by:
 - a. Running the generated “build_m50*.bat” file
 - b. Launching the Xilinx ISE GUI opening the newly built “m50*.xise” file

Linux:

1. Generate the ISE project files by running “gen_ise_files.sh”
2. Build the bit file using Xilinx ISE either by:
 - a. Running the generated “build_m50*.sh” file
 - b. Launching the Xilinx ISE GUI opening the newly built “m50*.xise” file

These methods will generate the necessary bitfile to be loaded on the Host System and are outlined in the sections that follow.

9.10.1. Building bitfile under Windows

9.10.1.1. Generate Xilinx ISE Project Files

First generate the ISE 13.4 project by executing “gen_ise_files.bat”. This process will create an ISE project file as well as all the necessary HDL files and corgen components used to generate the bitfile when building in ISE.

1. Open a command window
2. Change directories to “export_hw”
3. Execute “gen_ise_files.bat” as shown in Figure 58.
4. Verify that the files were created successfully by viewing the end of log file, as shown in Figure 59. “Successful!” should be the last line in the log file.

```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>cd C:\ImpulseC_Pico\Examples\M5XX\Memtest\export_hw
C:\ImpulseC_Pico\Examples\M5XX\Memtest\export_hw>
C:\ImpulseC_Pico\Examples\M5XX\Memtest\export_hw>gen_ise_files.bat
C:\ImpulseC_Pico\Examples\M5XX\Memtest\export_hw>REM THIS FILE IS AUTOMATICALLY
GENERATED BY IMPULSE CODEVELOPER. MODIFY AT YOUR OWN RISK.
C:\ImpulseC_Pico\Examples\M5XX\Memtest\export_hw>C:\Xilinx\13.4\ISE_DS\ISE\bin\n
t64\xtclsh gen50x_xise.tcl m501lx240_memtest_arch m501 1>gen_ise_files.log 2>&1
C:\ImpulseC_Pico\Examples\M5XX\Memtest\export_hw>REM Make sure last line of gen5
0x_ise.log = "Successful!", otherwise look to last few lines in gen50x_ise.log for
more details on cause of error
C:\ImpulseC_Pico\Examples\M5XX\Memtest\export_hw>_
```

Figure 63 - Generate ISE project files

```
Administrator: C:\Windows\system32\cmd.exe
source/pcie_gtx_v6.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/pcie_pipe_lane_v6.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/pcie_pipe_misc_v6.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/pcie_pipe_v6.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/pcie_reset_delay_v6.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/pcie_trn_128.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/pcie_upconfig_fix_3451_v6.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/sync_fifo.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/trn_rx_128.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/trn_tx_128.v
xfile_add: add glob file=firmware/m501/coregen-LX240T/v6_pcie_v2_4_8lane_gen2/
source/v6_pcie_v2_4_8lane_gen2.v
xfile_add: add file=firmware/m501/src/M501_LX240T_DDR3.ucf
xfile_add: add file=firmware/m501/src/M501_LX240T_PCIE.ucf
xfile_add:End
Configuring project properties
closing project
Success!
C:\ImpulseC_Pico\Examples\M5XX\Memtest\export_hw>_
```

Figure 64 - Expected Gen_Ise_File log report

There are two method to produce the bitfile. The first is to launch the Xilinx ISE GUI and generate the bitfile. The second method it to execute the build script in the **export_hw** directory. Both method will generate the necessary bitfile to be loaded on the Host System.

9.10.1.2. Compiling FPGA in Xilinx ISE GUI

Launch the Xilinx ISE GUI and open the ISE project located in the **export_hw** directory. Select Pico_Toplevel in the *Hierarchy* window and double-click “generate programming file” in the *Processes* window.

NOTE: Xilinx ISE 13.4 will ask the user to regenerate three fifos (a one time process). The following figures illustrate initial project launch, regenerate the three corgen fifos, and final output with timing score equal to zero.

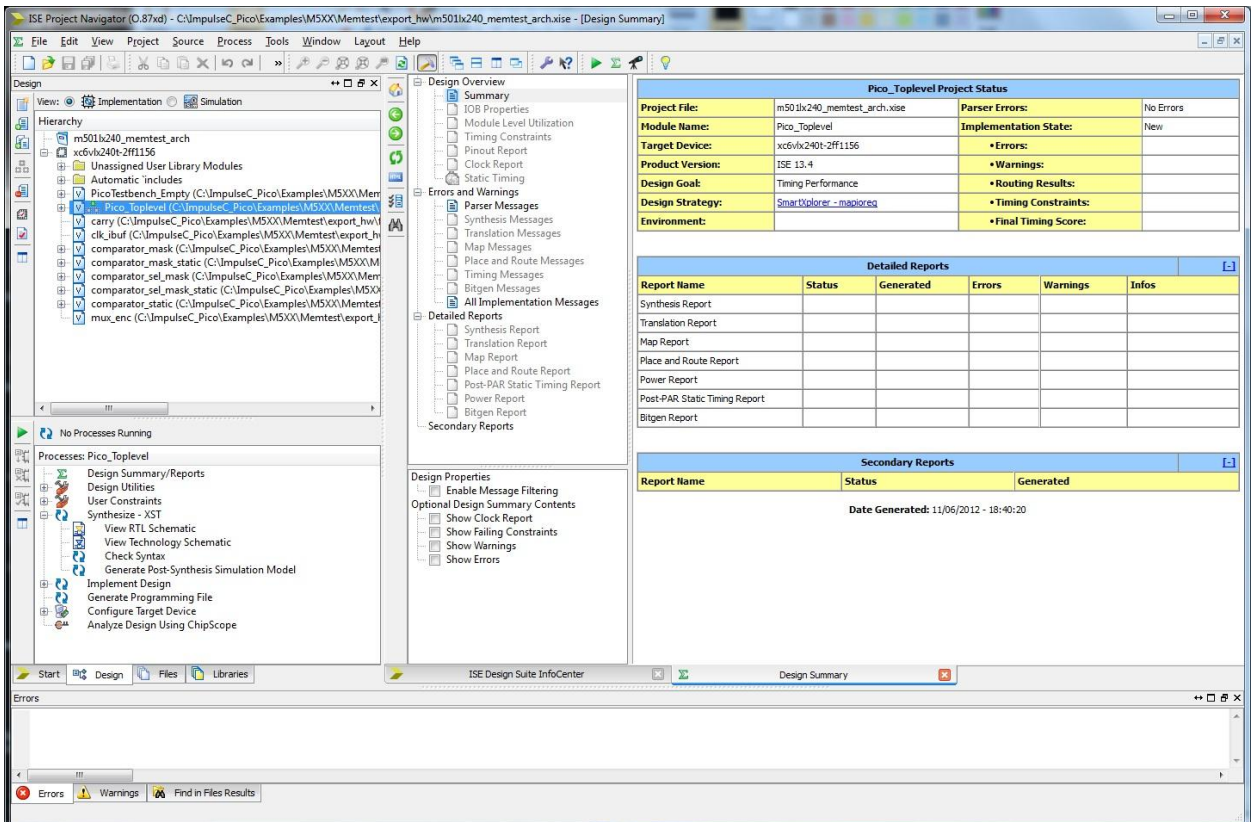


Figure 65 - Initial Xilinx ISE 13.4 GUI screen

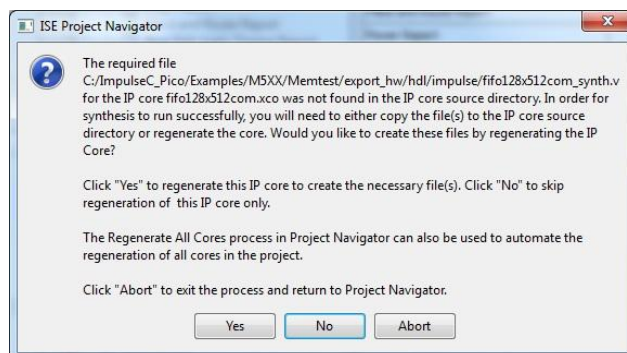


Figure 66 - Xilinx ISE 13.4 regenerate IP core fifo128x512com

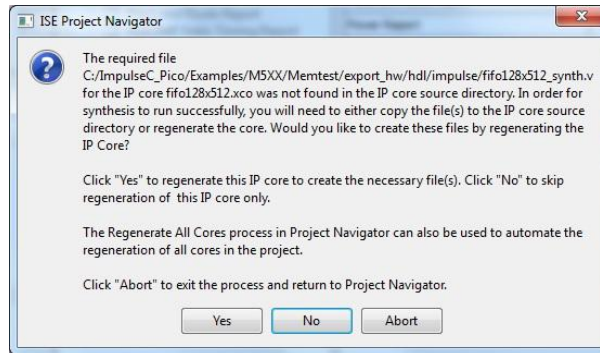


Figure 67 - Xilinx ISE 13.4 regenerate IP Core fifo128x512

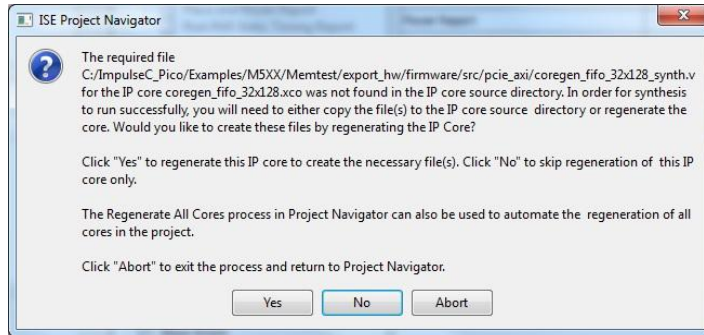


Figure 68 - Xilinx ISE 13.4 regenerate IP Core coregen_fifo_32x128

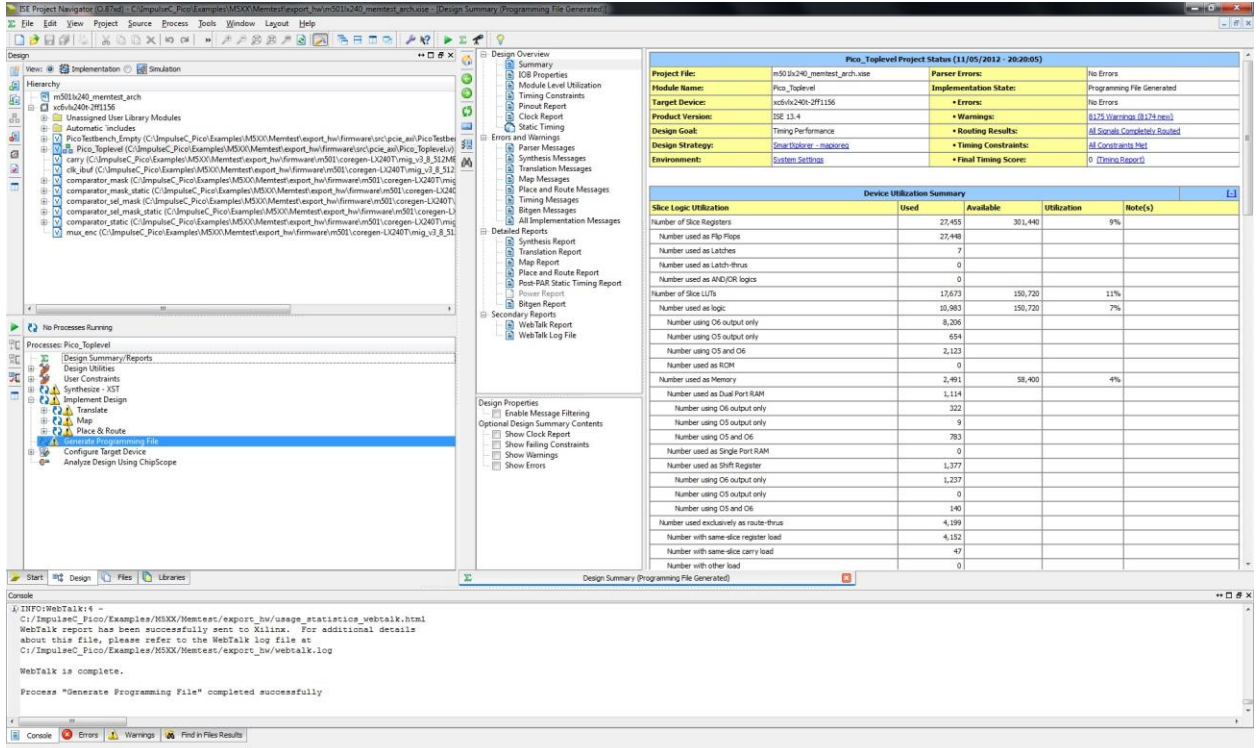


Figure 69 - Xilinx ISE 13.4 with timing score = 0

Once ISE 13.4 has completed, a bitfile “pico_toplevel.bit” will be present in the **export_hw** directory.

9.10.1.3. Compiling FPGA in Xilinx ISE using Command Line

In the top directory there will be the batch file “build_memtest_arch.bat” used to automatically run Xilinx ISE 13.4 to create the necessary .bit file used to program the M501 FPGA.

1. Open a command window
2. Execute “build_m501lx240_memtest_arch.bat”.

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 and a bitfile “pico_toplevel.bit” will be present in the **export_hw** directory.

9.10.2. Building bitfile under Linux

Please follow the steps outlined in the Passthrough example, **Sections 8.9.1 – Building bitfile under Linux**, in order to build the bitfile for the Memtest example under Linux. It is left to the user, as an exercise, to successfully build the bitfile.

9.11. 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.

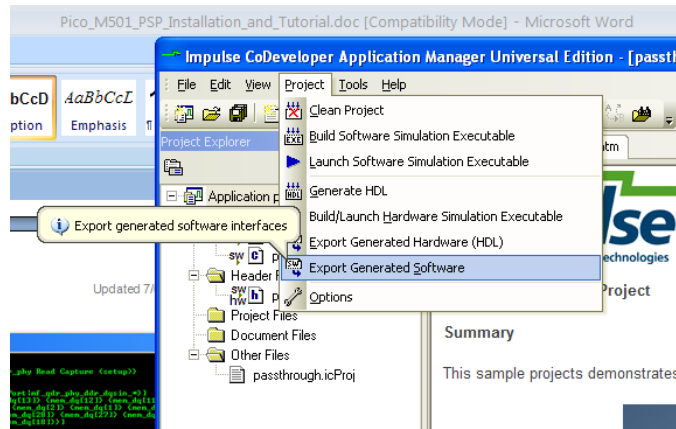


Figure 70 - Export Generated Software

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.

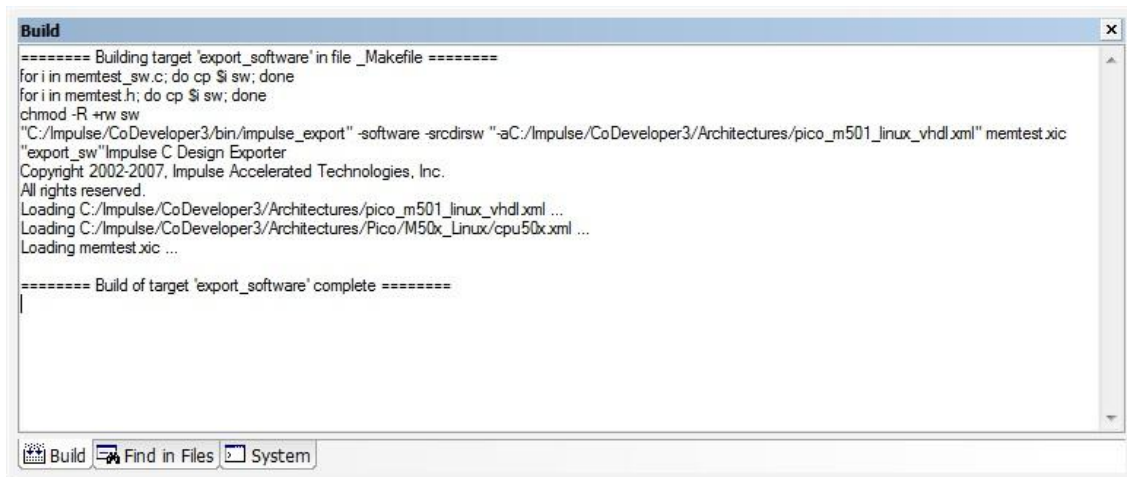


Figure 71 - Build window output

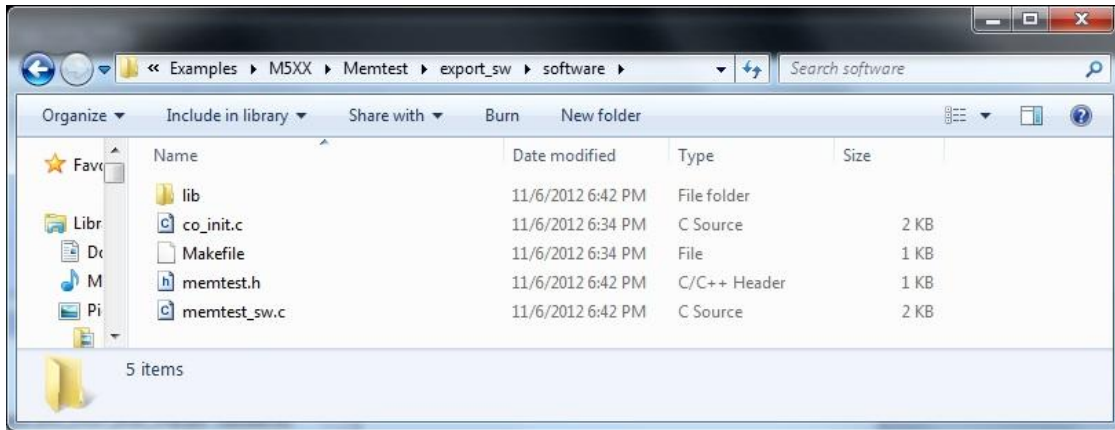


Figure 72 - Exported software directory

9.12. Programming the FPGA

The compiled software application in the “export_sw” directory will program the FPGA on the M501. Please continue to the next section.

9.13. Running Target Executable on the Host System

The Host System will need the following files:

1. Compiled bitfile output from ISE 13.4
2. Input data file
3. Application software source code
4. Makefile to compile the software

Please copy the following files and directory over to the Host System (figure 68). Create a folder (ie Pico/Memtest) and copy the following to that folder.

1. **export_hw** directory which includes “pico_toplevel.bit” bitfile
2. **export_sw** directory which includes the Makefile and software application source code

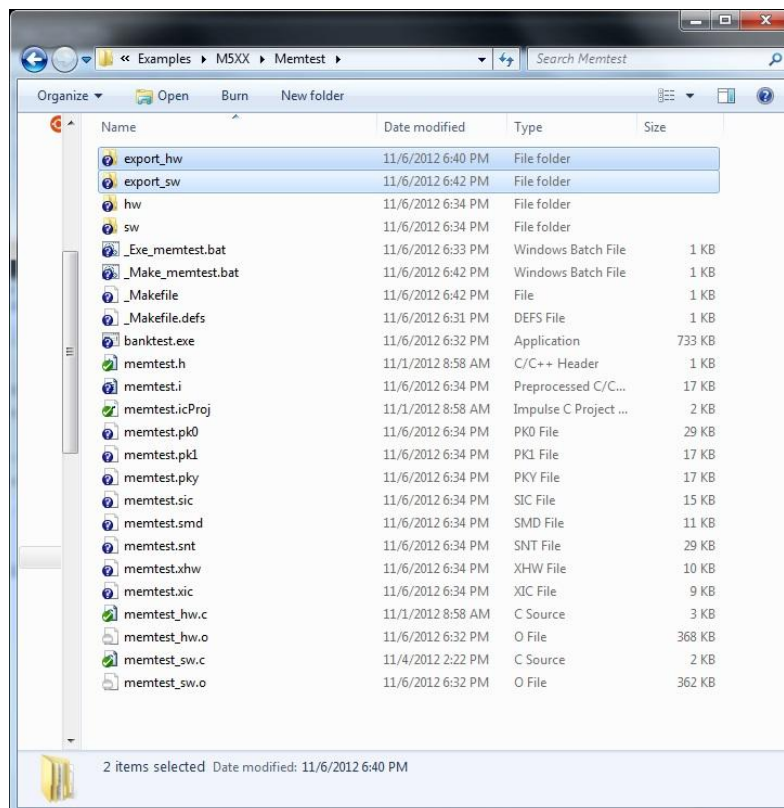
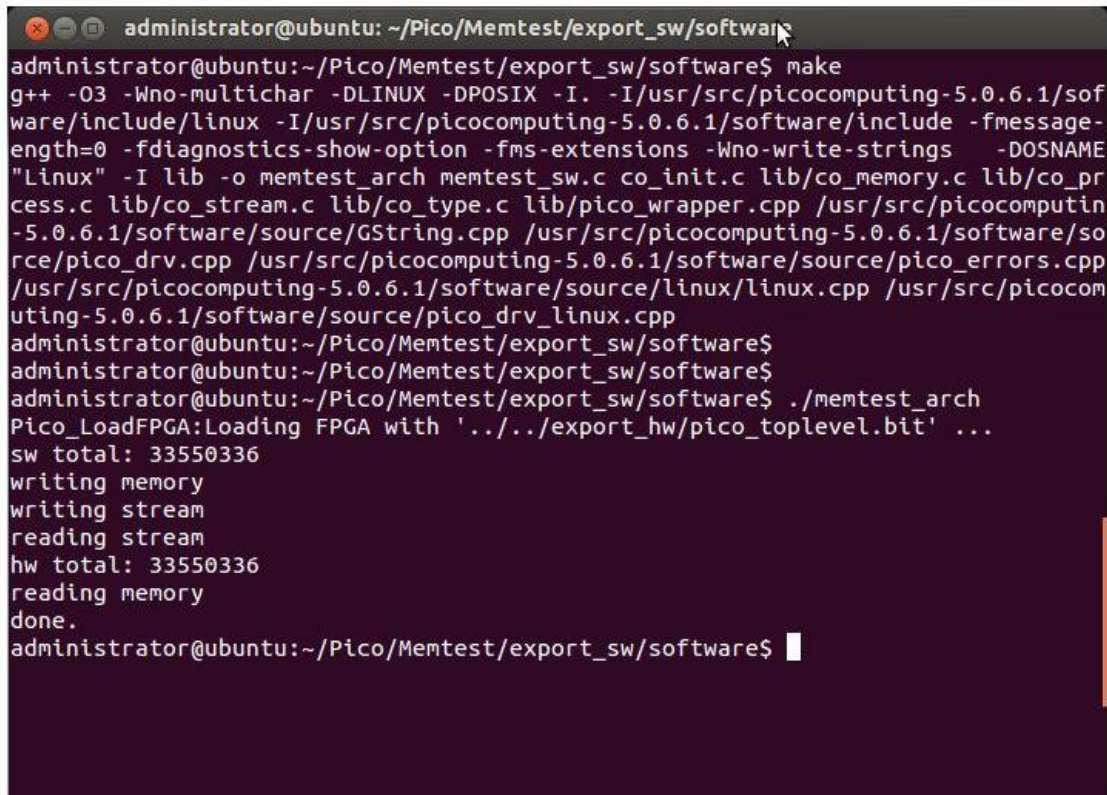


Figure 73 - Files & Directories to be copied to the Host System

The software application will load the FPGA every time it is executed and report the results upon completion. Please note that the process of loading the FPGA may take up

to 20 seconds due to the driver re-start sequence after the FPGA bitfile has been transferred to the card.

1. Open a terminal window.
2. Navigate to the location of your copied files and directories:
(ie. `cd Pico/Memtest/export_sw/software`).
3. Run “make”
4. Execute the generated application: “`./memtest_arch`”



```
administrator@ubuntu: ~/Pico/Memtest/export_sw/software
administrator@ubuntu:~/Pico/Memtest/export_sw/software$ make
g++ -O3 -Wno-multichar -DLINUX -DPOSIX -I. -I/usr/src/picocomputing-5.0.6.1/software/include/linux -I/usr/src/picocomputing-5.0.6.1/software/include -fmessage-length=0 -fdiagnostics-show-option -fms-extensions -Wno-write-strings -DOSNAME "Linux" -I lib -o memtest_arch memtest_sw.c co_init.c lib/co_memory.c lib/co_process.c lib/co_stream.c lib/co_type.c lib/pico_wrapper.cpp /usr/src/picocomputing-5.0.6.1/software/source/GString.cpp /usr/src/picocomputing-5.0.6.1/software/source/pico_drv.cpp /usr/src/picocomputing-5.0.6.1/software/source/pico_errors.cpp /usr/src/picocomputing-5.0.6.1/software/source/linux/linux.cpp /usr/src/picocomputing-5.0.6.1/software/source/pico_drv_linux.cpp
administrator@ubuntu:~/Pico/Memtest/export_sw/software$
administrator@ubuntu:~/Pico/Memtest/export_sw/software$
administrator@ubuntu:~/Pico/Memtest/export_sw/software$ ./memtest_arch
Pico_LoadFPGA:Loading FPGA with '../../export_hw/pico_toplevel.bit' ...
sw total: 33550336
writing memory
writing stream
reading stream
hw total: 33550336
reading memory
done.
administrator@ubuntu:~/Pico/Memtest/export_sw/software$
```

Figure 74 - Exported SW executed on target platform