

QuickStart Instructions

FPGA-Kit

phyCORE-MPC5200B-I/O

Using ALTERA Quartus® II V7.2SP1 Web Edition tool chain

Note: The PHYTEC Tools CD/DVD for the phyCORE-MPC5200B-I/O with FPGA includes the electronic version of the phyCORE-MPC5200B-I/O English Hardware Manual

Edition: December 2007

A product of a PHYTEC Technology Holding company

In this manual copyrighted products are not explicitly indicated. The absence of the trademark (™) and copyright (©) symbols does not imply that a product is not protected. Additionally, registered patents and trademarks are similarly not expressly indicated in this manual.

The information in this document has been carefully checked and is believed to be entirely reliable. However, PHYTEC Messtechnik GmbH assumes no responsibility for any inaccuracies. PHYTEC Messtechnik GmbH neither gives any guarantee nor accepts any liability whatsoever for consequential damages resulting from the use of this manual or its associated product. PHYTEC Messtechnik GmbH reserves the right to alter the information contained herein without prior notification and accepts no responsibility for any damages which might result.






Additionally, PHYTEC Messtechnik GmbH offers no guarantee nor accepts any liability for damages arising from the improper usage or improper installation of the hardware or software. PHYTEC Messtechnik GmbH further reserves the right to alter the layout and/or design of the hardware without prior notification and accepts no liability for doing so.

© Copyright 2007 PHYTEC Messtechnik GmbH, D-55129 Mainz.

Rights - including those of translation, reprint, broadcast, photomechanical or similar reproduction and storage or processing in computer systems, in whole or in part - are reserved. No reproduction may be made without the explicit written consent from PHYTEC Messtechnik GmbH.

| | EUROPE | NORTH AMERICA |
|-----------------------|--|--|
| Address: | PHYTEC Technologie Holding AG Robert-Koch-Str. 39 55129 Mainz GERMANY | PHYTEC America LLC 203 Parfitt Way SW, Suite G100 Bainbridge Island, WA 98110 USA |
| Ordering Information: | +49 (800) 0749832 order@phytec.de | 1 (800) 278-9913 sales@phytec.com |
| Technical Support: | +49 (6131) 9221-31 support@phytec.de | 1 (800) 278-9913 support@phytec.com |
| Fax: | +49 (6131) 9221-33 | 1 (206) 780-9135 |
| Web Site: | http://www.phytec.de | http://www.phytec.com |

1st Edition: December 2007

| | | | | |
|----------|---|------------|---|---------------|
| 1 | Introduction | 1 |  | |
| 1.1 | Rapid Development Kit Documentation..... | 1 | | |
| 1.2 | Professional Support Packages available..... | 2 | | |
| 1.3 | Overview of this QuickStart Instruction..... | 2 | | 5 min |
| 1.4 | Conventions used in this QuickStart..... | 3 | | |
| 1.5 | System Requirements..... | 4 | | |
| 1.6 | The PHYTEC phyCORE-MPC5200B-I/O | 5 | | |
| 1.7 | ALTERA Quartus® II V7.2SP1 Web Edition tool chain..... | 8 | | |
| 2 | Getting Started..... | 10 |  | |
| 2.1 | Requirements of the Host Platform | 10 | | |
| 2.2 | Configuring the Host Platform | 10 | | |
| 2.2.1 | Installing PHYTEC Tools and ALTERA Quartus® II V7.2SP1 Web Edition tool chain..... | 11 | | 35 min |
| 2.2.2 | Licensing of the ALTERA Quartus® II V7.2SP1 Web Edition | 17 | | |
| 2.3 | Connecting the host to the target's FPGA JTAG interface | 18 | | |
| 2.4 | Copying an Example to the Target | 20 | | |
| 3 | Getting More Involved | 28 |  | |
| 3.1 | Creating and simulation of a new Quartus® II V7.2SP1 Web Edition project..... | 28 | | 70 min |
| 3.1.1 | Creating a new Quartus® II V7.2SP1 Web Edition project | 28 | | |
| 3.1.2 | Simulation of the Counter Project with ALTERA Quartus® II V7.2SP1 Web Edition | 52 | | |
| 3.2 | The Bus System used for addressing the FPGA on the phyCORE-MPC5200B-I/O..... | 63 |  | 30 min |
| 3.2.1 | The Local Plus Bus of the MPC5200B attached to the FPGA | 63 | | |
| 3.2.2 | The Avalon interface | 66 | | |
| 3.2.3 | Signal Conversion from the MPC5200B's Local Plus Bus to the FPGA's Avalon interface..... | 73 | | |
| 3.2.4 | Adding an Avalon slave interface to the counter project | 85 | | |
| 3.2.5 | Compiling and testing the Avalon counter project | 100 | | |
| 3.3 | Use of the SOPC Builder | 113 |  | 20 min |
| 3.3.1 | Creating a new component with the SOPC Builder | 117 | | |
| 3.3.2 | Creating a project with the SOPC Builder..... | 125 | | |
| 4 | FPGA support under Linux..... | 148 | | |
| 4.1 | General | 148 | | |
| 4.2 | Accessing Peripherals..... | 148 | | |

| | | |
|----------|---------------------------------|------------|
| 4.3 | Demo | 149 |
| 5 | Further Information..... | 151 |
| 6 | Summary | 152 |

1 Introduction

**5 min**

In this QuickStart you can find general information on the PHYTEC phyCORE-MPC5200B-I/O and an overview of the ALTERA Quartus® II V7.2SP1 Web Edition tool chain. You can also find instructions on how to run example programs on the FPGA of the phyCORE-MPC5200B-I/O, mounted on the PHYTEC phyCORE Development Board MPC5200B.

Please refer to the phyCORE-MPC5200B-I/O Hardware Manual for specific information on such board-level features as jumper configuration, memory mapping and pin layout.

1.1 Rapid Development Kit Documentation

This "Rapid Development Kit" includes the following electronic documentation on the enclosed "PHYTEC Tools CD/DVD for the phyCORE-MPC5200B-I/O with FPGA":

- PHYTEC phyCORE-MPC5200B-I/O Hardware Manual and Development Board Hardware Manual
- MPC5200B-I/O controller User's Manuals and Data Sheets
- this QuickStart Instruction with general "Rapid Development Kit" description, software installation advice and example programs enabling quick out-of-the box start-up of the phyCORE-MPC5200B-I/O in conjunction with the FPGA and the ALTERA Quartus® II V7.2SP1 Web Edition tool chain.

1.2 Professional Support Packages available

This Kit comes with free installation support. If you do have any questions concerning installation and setup, you are welcome to contact our support department.

For more in-depth questions, we offer a variety of custom tailored packages with different support options (e-mail, phone, direct contact to the developer) and different reaction times.

Please contact our sales team to discuss the appropriate support option if professional support beyond installation and setup is important to you.

1.3 Overview of this QuickStart Instruction

This QuickStart Instruction gives a general "Rapid Development Kit" description, as well as software installation advice and example programs enabling quick out- of-the box start-up of the phyCORE-MPC5200B in conjunction with the FPGA and the ALTERA Quartus® II V7.2SP1 Web Edition tool chain:

- 1) The "*Getting Started*" section describes the configuration of the host platform and the setup to install the tools used in this QuickStart.
- 2) The "*Getting More Involved*" section provides step-by-step instructions on how to modify the example, create and build new projects and copy output files to the phyCORE-MPC5200B's FPGA using the USB- or ByteBlaster interface.

In addition to the dedicated data for this Rapid Development Kit, the PHYTEC Tools CD/DVD for the phyCORE-MPC5200B-I/O with FPGA contains supplemental information on embedded microcontroller design and development.

1.4 Conventions used in this QuickStart

The following is a list of the typographical conventions used in this book:

Italic Used for file and directory names, program and command names, command-line options, menu items, URLs, and other terms that correspond the terms on your desktop.

Bold Used in examples to show commands or other text that should be typed literally by the user.

Pay special attention to notes set apart from the text with the following icons:



At this part you might leave the path of this QuickStart.



This is a warning. It helps you to avoid annoying problems.



You can find useful supplementary information about the topic.



At the beginning of each chapter you can find information of the time to pass the following chapter.



You have successfully passed an important part of this QuickStart.



You can find information to solve problems.

1.5 System Requirements

Use of this "Rapid Development Kit" requires:

- the PHYTEC phyCORE-MPC5200B-I/O,
- the PHYTEC Development Board with the included DB-9 serial cable, AC-to-DC adapter supplying 12 V DC/min. 1.5 A,
- PHYTEC Distribution of IP's for MPC5200B-I/O
- an IBM-compatible host-PC (586 or higher)
- Win 2000 or Win XP OSS (x86)
- recommended free disk space: 4 GB

For more information and example updates, please refer to the following sources:

PHYTEC

<http://www.phytec.de>
support@phytec.de



<http://www.altera.com/>

1.6 The PHYTEC phyCORE-MPC5200B-I/O

The phyCORE-MPC5200B-I/O represents an affordable yet highly functional Single Board Computer (SBC) solution in the size 84 x 57 mm. The standard board is populated with a Freescale PowerPC Microcontroller MPC5200B featuring a 32-bit processor architecture with Double precision FPU, 396 MHz processor speed, Peripheral component interconnect (PCI) controller, ATA controller, BestComm DMA subsystem, 6 programmable serial controllers (PSC) configurable for the following functions:

Fast Ethernet controller (FEC), Universal serial bus controller (USB revision 1.1 host), Two inter-integrated circuit interfaces (I2C), Serial peripheral interface (SPI), Dual CAN 2.0 A/B controller (MSCAN), J1850 byte data link controller (BDLC)

All applicable LocalPlus data/address lines and applicable signals extend to two high-density 400-pin Molex SMT pin header connectors (pin width is 0.635 mm./25mil) lining the circuit board edges. This enables the phyCORE-MPC5200B-I/O to be plugged like a “big chip” into target hardware.

Furthermore the phyCORE-MPC5200B-I/O is equipped with an FPGA, with up to 5000 or 8000 logic elements (LEs). All free IOs of the FPGA which are not assigned to the bus between the FPGA and the controller are available at the 400-pin Molex SMT pin header connector.

The standard memory configurations of the phyCORE-MPC5200B-I/O features 64 MB DDR-SDRAM and 32 MB external Flash. The external Flash supports direct on-board programming without additional programming voltages.

phyCORE-MPC5200B-I/O Technical Highlights

- phyCORE dimensions 84 x 57 mm with two high-density 400-pin Molex SMT pin header connectors
- Processor: Freescale Embedded PowerPC MPC5200B, 396 MHz clock
- FPGA ALTERA EP2C5/8F256 with up to 180 IOs

Internal Features of the MPC5200B:

- e300 core
 - 760 MIPS at 400 MHz (-40 to +85 °C)
 - 32 k instruction cache, 32 k data cache
 - Double precision FPU
 - Instruction and data MMU
 - SDRAM / DDR SDRAM memory Interface
 - up to 132 MHz operation
 - SDRAM and DDR SDRAM support
 - 256 MB addressing range per CS, two CS available
 - Flexible multi-function external bus interface
 - Peripheral component interconnect (PCI) controller
 - ATA controller
 - BestComm DMA subsystem
 - 6 programmable serial controllers (PSC), configurable for the following functions:
 - Fast Ethernet controller (FEC)
 - Supports 100Mbps IEEE 802.3 MII, 10 Mbps IEEE 802.3 MII
 - Universal serial bus controller (USB)
 - USB revision 1.1 host
 - Two inter-integrated circuit interfaces (I2C)
 - Serial peripheral interface (SPI)
 - Dual CAN 2.0 A/B controller (MSCAN)
 - J1850 byte data link controller (BDLC)
 - Test/debug features
 - JTAG (IEEE 1149.1 test access port)
 - Common on-chip processor (COP) debug port
-

The PHYTEC Development Board, in card dimensions (190 x 170 mm/7.5 x 6.7 in.), is fully equipped with all mechanical and electrical components necessary for the speedy and secure insertion of PHYTEC phyCORE-MPC5200B-I/O Single Board Computer.

Development Board Technical Highlights

- reset push button
- one software programmable LED for controller MPC5200B
- one software programmable LED for FPGA
- LEDs for supply voltage monitoring, 3.3V, 3.3V PCI, 5.0V PCI
- power supply for unregulated input voltage of +9-12V and regulated output voltages of +3.3V for the phyCORE-MPC5200B-I/O, as well as +3.3V and +5V for the PCI interface.
- PCI card Interface
- two DB-9 sockets for the RS-232 interface
- two DB-9 plugs for two separate CAN interfaces
- RJ45 socket for 10/100 Mbit Ethernet
- USBA socket for Full Speed USB Host Interface
- IDE Compact Flash Card holder
- AC97 Sound Interface
- Debug Interface socket for MPC5200B (COP-Interface)
- JTAG Interface for FPGA

1.7 ALTERA Quartus® II V7.2SP1 Web Edition tool chain

ALTERA's development tools for the Cyclone II Architecture support every level of developer from the professional applications engineer to the student just learning about logic implementation. The ALTERA Quartus® II V7.2SP1 Web Edition supports all ALTERA FPGA devices including the E2C5/8 devices populated on the phyCORE-MPC5200B-I/O. For a complete list of supported FPGA derivatives go to:

<http://www.altera.com/products/software/products/quartus2web/sof-quarwebmain.html>

Quartus® II Web Edition V7.2SP1, the latest version of ALTERA's popular IDE, combines project management, source code editing, signal analyzing, and FPGA programming in a single, powerful environment. This QuickStart provides an overview of the most commonly used features including:

- Project management, device setup, and tool configuration
- Editor facilities for creating, modifying, and correcting source code
- JTAG/signal analyzing

Once installed, the folder *C:\ALTERA\72SP1\qdesign* is the default destination location for all Quartus® II tools; executables; include, header and example files; as well as online help and documentation, while the Quartus® II executable is located at *C:\ALTERA\72SP1\quartus\bin*. You can start Quartus® II by selecting it from the *Programs* menu using the Windows *Start* button. The ALTERA Quartus® II V7.2SP1 Web Edition icon will also be placed on your desktop for easy startup of the development tools.

The Quartus® II Text Editor is a flexible tool for entering text-based designs in the AHDL, VHDL, and Verilog HDL languages, and the Tcl scripting language that are integrated into the Quartus® II system. You can also use the Text Editor to enter, edit, and view other ASCII files, including those created for or by the Quartus® II software.

Menu-based commands, text entry, and editing features allow you to create Text Design Files (.tdf) with AHDL, VHDL Design Files (.vhd) with VHDL, Verilog Design Files (.v) with Verilog HDL, and Tcl Script Files (.tcl) with Tcl. You can also generate a Tcl Script File from an existing project created in the Quartus® II software and edit it in the Text Editor, or create and edit a Tcl script using the Text Editor. In the Quartus® II software, you can freely combine Text Design Files, VHDL Design Files, and Verilog Design Files with other types of design files in a hierarchical project. You can also use the Text Editor to create or edit Assembly Files (.a, .asm), C/C++ Include Files (.h), C Source Files (.c), and C++ Source Files (.cpp).

You can use any ASCII text editor to create AHDL, VHDL, Verilog HDL designs, and Tcl scripts. With the Quartus® II Text Editor, however, you can take advantage of unique Quartus® II features while you edit, compile, and debug a project. For example, the *Messages* window automatically locates and highlights errors in the Text Editor window. Additionally, you can get context-sensitive help on all AHDL elements, keywords, and statements, as well as on megafunctions and primitives.

The full ALTERA Quartus® II Development Tools can be purchased through the official ALTERA distributors or please contact our sales representatives for more information.

For more information on ALTERA Quartus® II V7.2SP1 Web Edition tools visit their website at:

<http://www.altera.com/products/software/>

2 Getting Started

**35 min**

In this chapter you will carry out the basic configurations to pass the steps in this QuickStart. First you will learn how to set up the host platform. You will install additional software packages and configure your host to connect to the targets FPGA via JTAG Interface. After connecting the host to the target you will copy an application into the targets FPGA. At the end of this chapter you will be able to start a first demo application on the targets FPGA.

2.1 Requirements of the Host Platform

To pass the following steps in this Quick Start, you will need a host pc with either Win 2000 or Win XP (x86) and free available disk space of up to 2GByte.

In the following configuration steps we assume, that the host pc has a free USB or parallel Port available for connecting the programming hardware. The JTAG interface of the FPGA on the target and the host will be connected together with a cable for code download purposes. If your host does not have a free port which corresponds to the equipment of the RDK, you can either free the necessary port, or consider to use a RDK with different programming hardware.

2.2 Configuring the Host Platform

In this section you will learn how to configure the host platform. You will execute the following steps in this passage:

- Installing software packages. These packages are necessary to accomplish the steps in the QuickStart Instruction.
- Setup the JTAG Interface configuration to use the host pc with your target.

To accomplish the steps in the QuickStart Instruction you will have to install a few software packages.



If you don't install all of these packages, the setup may fail or some configuration steps won't work correctly.

2.2.1 Installing PHYTEC Tools and ALTERA Quartus® II V7.2SP1 Web Edition tool chain

When you insert the PHYTEC Tools CD/DVD for the phyCORE-MPC5200B-I/O with FPGA into the CD-ROM drive of your host PC, the PHYTEC Tools CD/DVD should automatically launch a setup program that installs the software required for the Rapid Development Kit. Otherwise the setup program *PCM-032_phyCORE-MPC5200B-IO-1.0.1-Setup.exe* can be manually executed from the root directory of the PHYTEC Tools CD/DVD.

The following window appears:

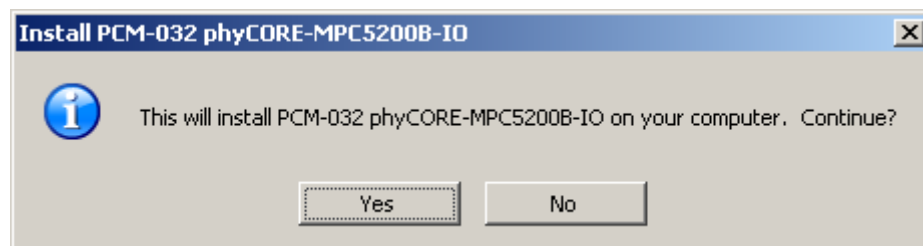


Figure 1: First window after inserting the PHYTEC Tools CD/DVD

- Click *Yes* or press *Enter* to start the Install Wizard for the software package of the Rapid Development Kit.

The *Welcome* window gives some information about the Rapid Development Kit and the version of the software which will be installed.

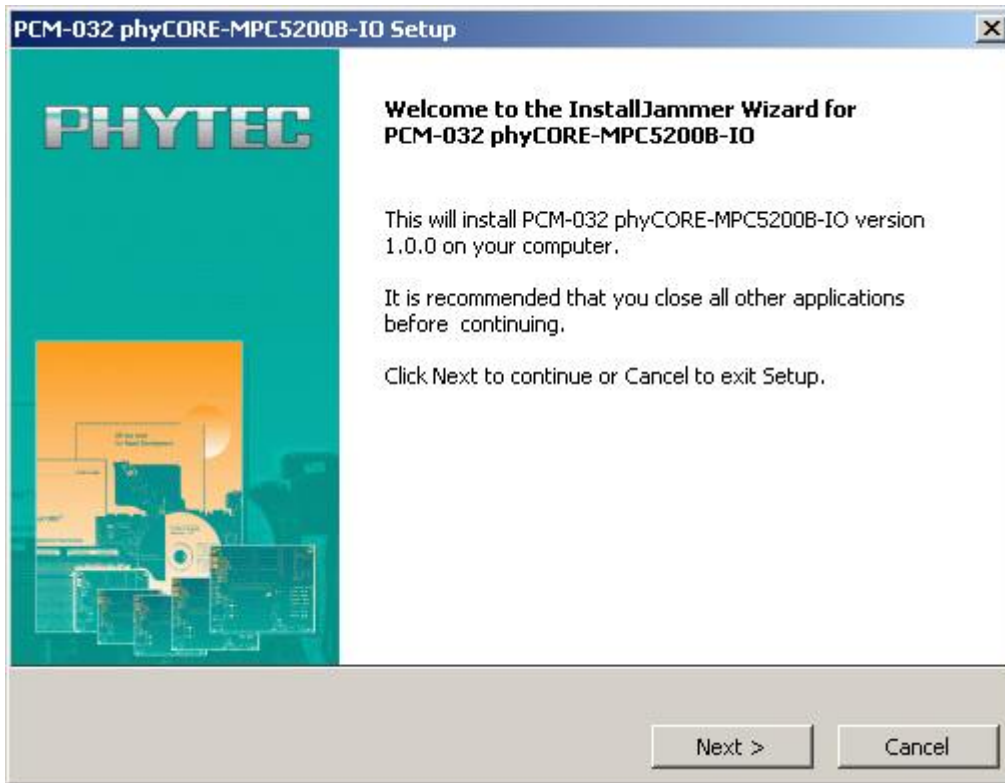


Figure 2: Welcome window of the Install Wizard

- Click *Next* to continue.

In the following window you can select the destination location for installation of Rapid Development Kit software and documentation.



The default destination location for the PHYTEC Tools is `c:\PHYTEC\PCM-032_phyCORE-MPC5200B-I/O`. All path and file statements within this QuickStart Instruction are based on the assumption that you accept the default install paths and drives. If you decide to individually choose different paths you must consider this for all further file and path statements when working with this QuickStart.

We strongly recommend accepting the default destination location.

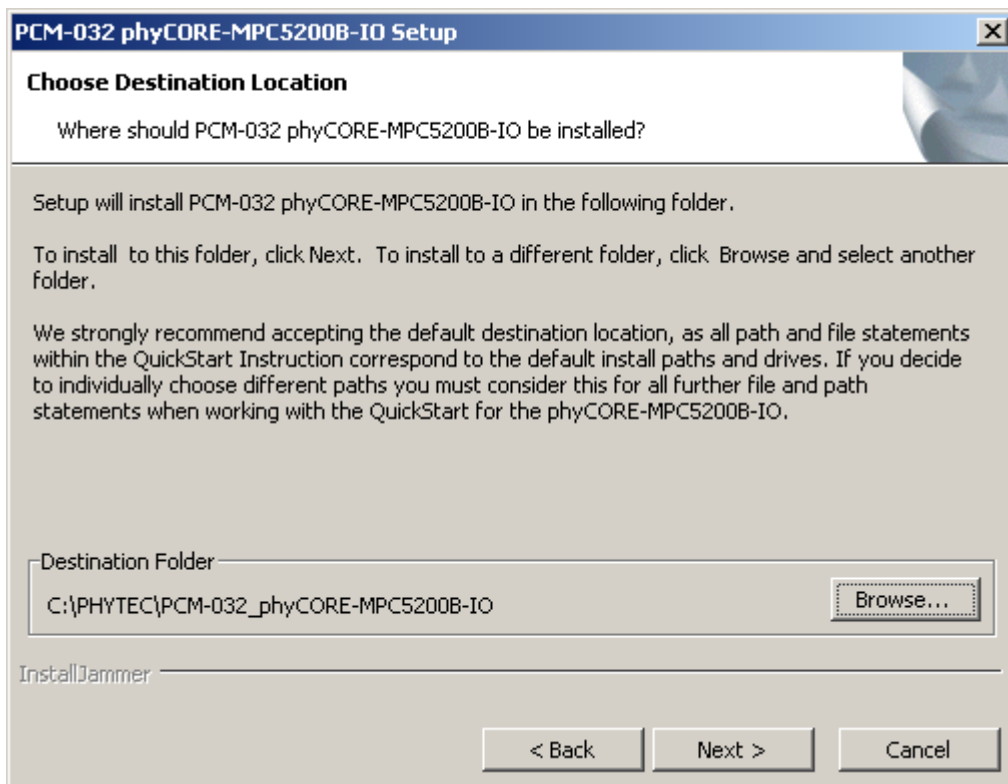


Figure 3: Choosing the default destination location for the PHYTEC Tools

- Click *Next* to continue.
- Click *Next* to skip the Information window and to start copying the PHYTEC Tools.

After the PHYTEC Tools are copied to the host PC you will be asked to proceed with the installation of the ALTERA Quartus® II V7.2SP1 Web Edition tool chain.



As the functionality and the appearance varies for different versions of the ALTERA Quartus® II Web Edition tool chain we strongly recommend to install this version.

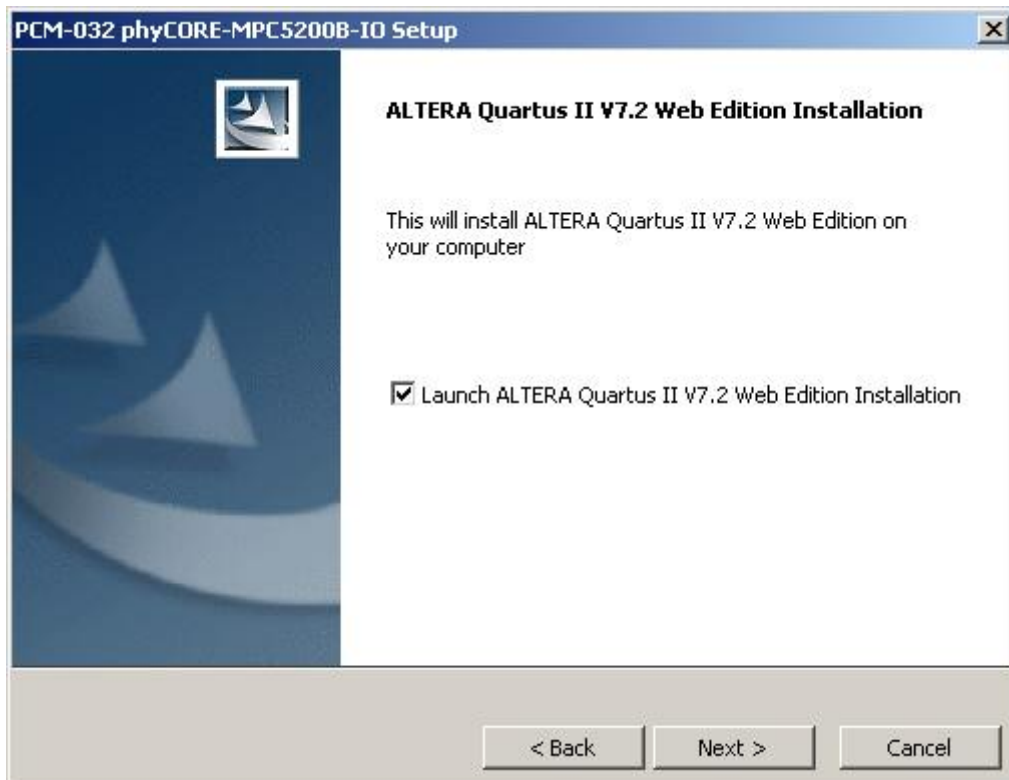


Figure 4: Choose to install the ALTERA Quartus® II Web Edition tool chain

- Click *Next* to start the setup program for the ALTERA Quartus® II Web Edition tool chain.

The *Welcome* window of the Quartus® II InstallShield Wizard will appear.

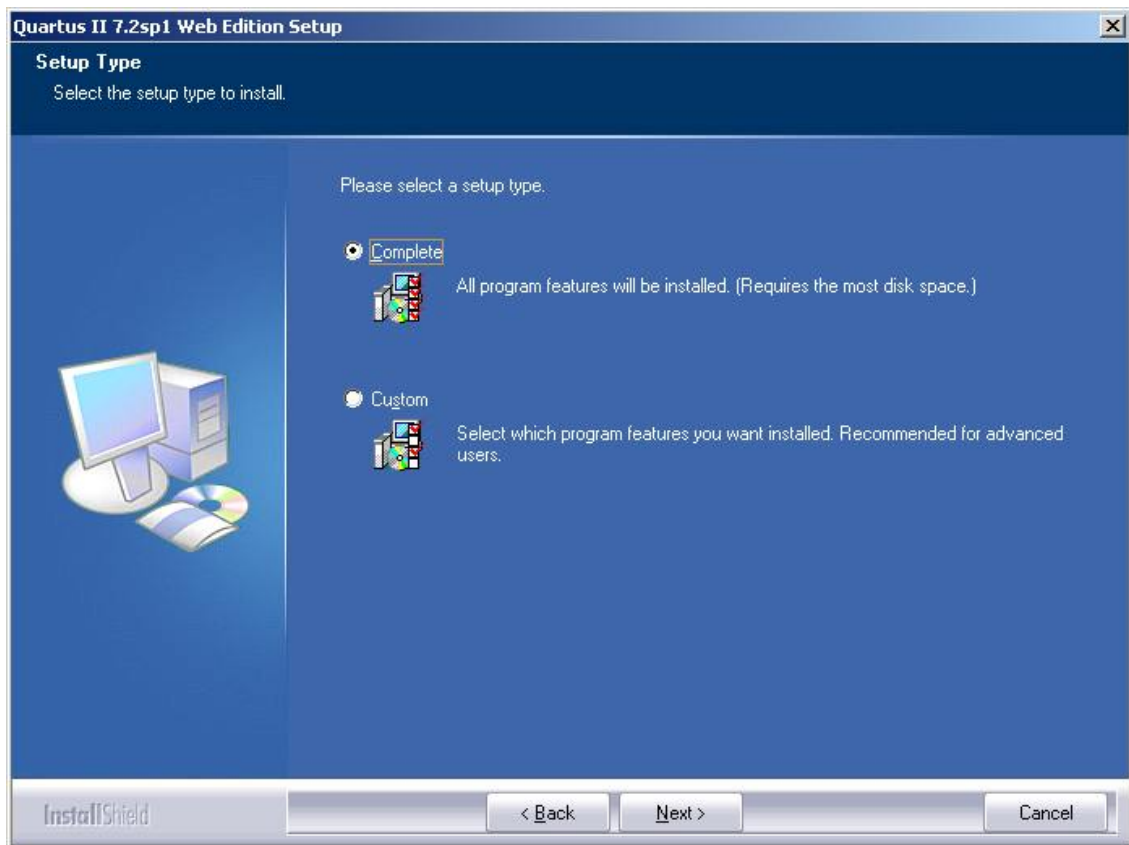


Figure 5: Welcome window of the Quartus® II InstallShield Wizard

- Follow the steps of the Quartus® II InstallShield Wizard to install the ALTERA Quartus® II V7.2SP1 Web Edition tool chain.



The default destination location for the ALTERA tool chain is `c:/ALTERA/72SP1/`. All path and file statements within this QuickStart Instruction are based on the assumption that you accept the default install paths and drives. If you decide to individually choose different paths you must consider this for all further file and path statements when working with this QuickStart.

We strongly recommend accepting the default destination location.

After the successful installation of the ALTERA Quartus® II V7.2SP1 Web Edition tool chain two more software packages from ALTERA must be installed to carry out the examples within this QuickStart Instructions. This is ALTERA's MegaCore IP Library and service pack SP1 for the library.

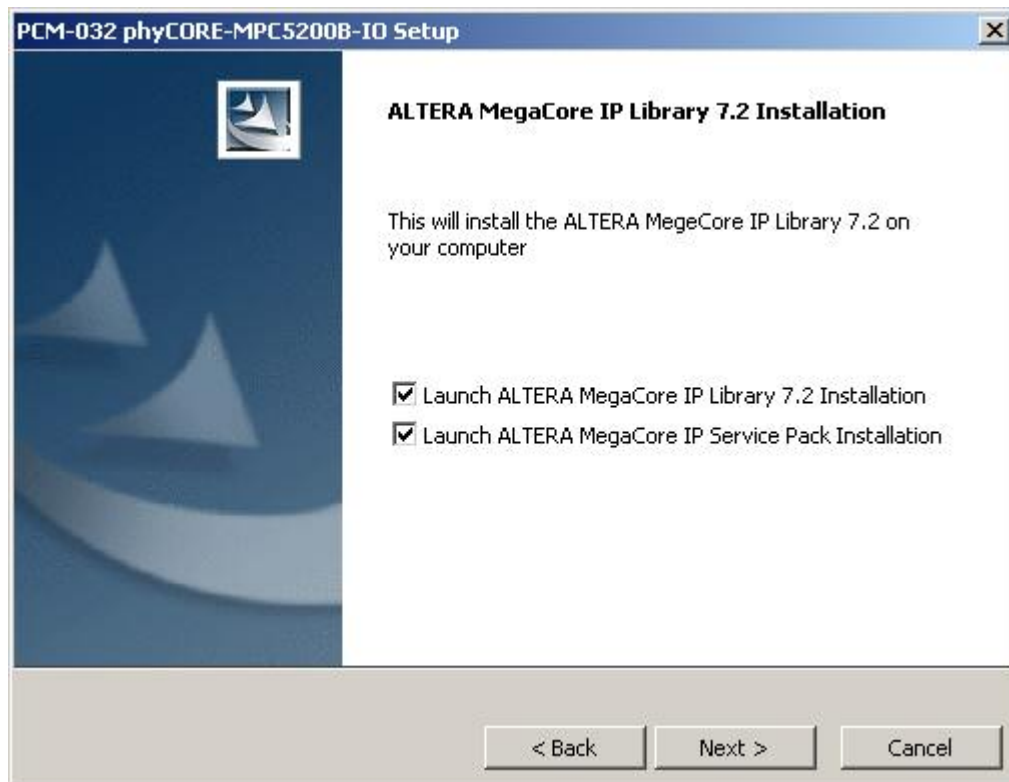


Figure 6: Installation of additional software from ALTERA

- Click *Next* to start the installation of the additional software from ALTERA.
- Follow the steps of the MegaCore IP Library 7.2 setup and the InstallWizard for the Service Pack.

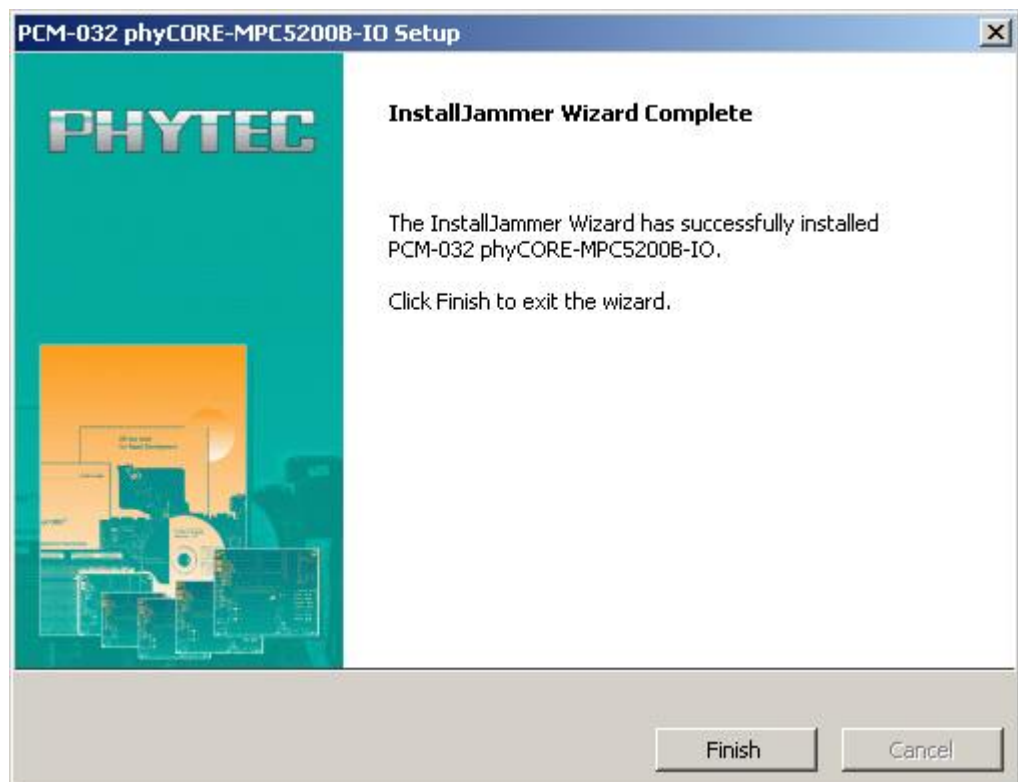


Figure 7: Installation of additional software from ALTERA

- Click *Finish* to end the installation of the PHYTEC Tools CD/DVD.

2.2.2 Licensing of the ALTERA Quartus® II V7.2SP1 Web Edition

For licensing and use of ALTERA Quartus® II V7.2SP1 Web Edition, you must acquire a license key on the ALTERA Homepage. To do this, please follow the link below and carry out the instructions given there.

<http://www.altera.com/support/licensing/lic-index.html>



You have successfully installed the software for the RDK-Kit phyCORE-MPC5200B with FGPA. You can find all programs and source code you will need to develop your own applications for the targets FPGA on your host system. All necessary configurations were done by the setup program.

2.3 Connecting the host to the target's FPGA JTAG interface

In order to connect the host PC to the JTAG interface of the FPGA on the phyCORE-MPC5200B-I/O, please use the cable corresponding to the JTAG programming adapter which is enclosed appropriate to the kit. If a USB-Blaster is enclosed with the kit, please use the enclosed USB cable. If a ByteBlaster is enclosed with the kit, please use the enclosed parallel port cable to connect the programming adapter to the host PC. The JTAG interface of the FPGA on the phyCORE-MPC5200B is extended to connector X8 of the development board. Please make sure that the cable is connected correctly. The 1 mark on the flat ribbon cable of the programming adapter is either identified by a black or white mark. Pin 1 on connector X8 of the motherboard is identified by a chamfer corner in the component printing of the board.

- Connect the appropriate programming adapter with connector X8 on the target's development board and the USB or Parallel cable with the free available port on your host.



Ensure to use the appropriate cable included in this RDK.

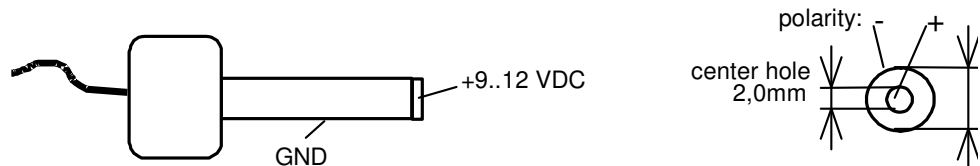
If you have a USB-Blaster, you must install the corresponding USB driver on the host PC. You can find the driver at: `c:\altera\72SP1\quartus\drivers\usb-blaster`. You can find a description concerning installation of the USB-Blaster on the Altera Homepage at:

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

- Connect the AC adapter with the power supply connector X6 (9-12V) on your development board.



The power connector should have 9-12 VDC inside and outside should be ground.



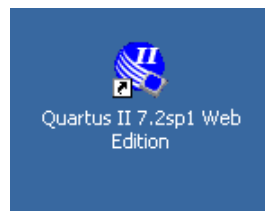
After connecting the board with the power supply the target is ready to use with ALTERA Quartus® II V7.2SP1 Web Edition

2.4 Copying an Example to the Target

In this section you will learn how to copy an example program to the targets FPGA with either USB-Blaster or ByteBlaster as programming adapter. After downloading the program the example will execute automatically on the targets FPGA.

Copying a program to the targets FPGA:

- First double click on the icon *Quartus® II V7.2SP1 Web Edition* on the desktop to launch the program.



The program starts and opens with a blank project navigator, as shown in the following picture.

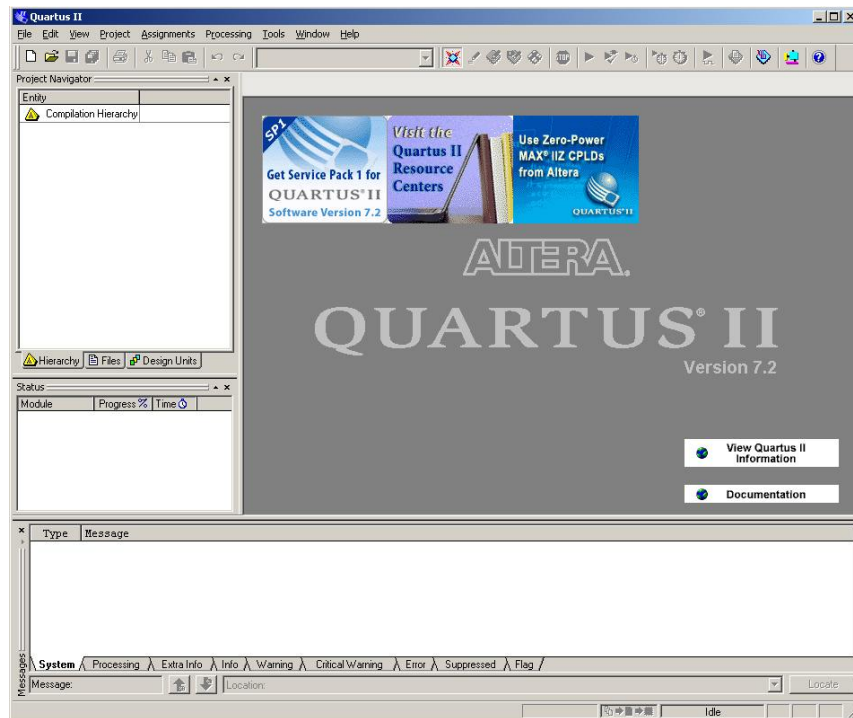


Figure 8: ALTERA Quartus® II V7.2SP1 Web Edition – Project navigator

- Now click *File* on the menu bar and select *Open Project*.

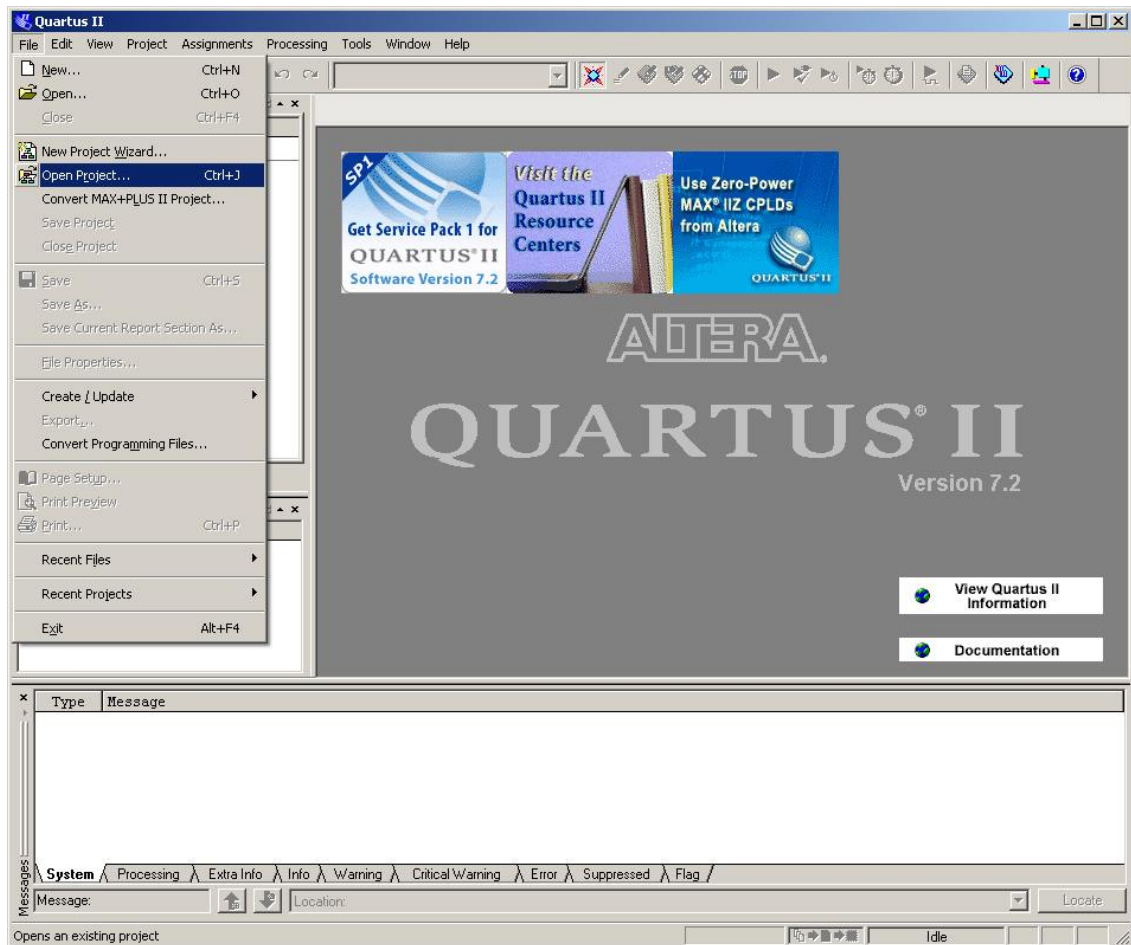


Figure 9: ALTERA Quartus® II – Opening an existing project

- Select the *Blinky* demo project from the path `c:\PHYTEC\PCM-032_phyCORE-MPC5200B-IO\FPGA-Kit\Quickstar\Demos\Blinky`, which was previously installed during the setup.

Now Quartus® II opens the project. After opening, the open project is shown in the project navigator on the left side of the window.

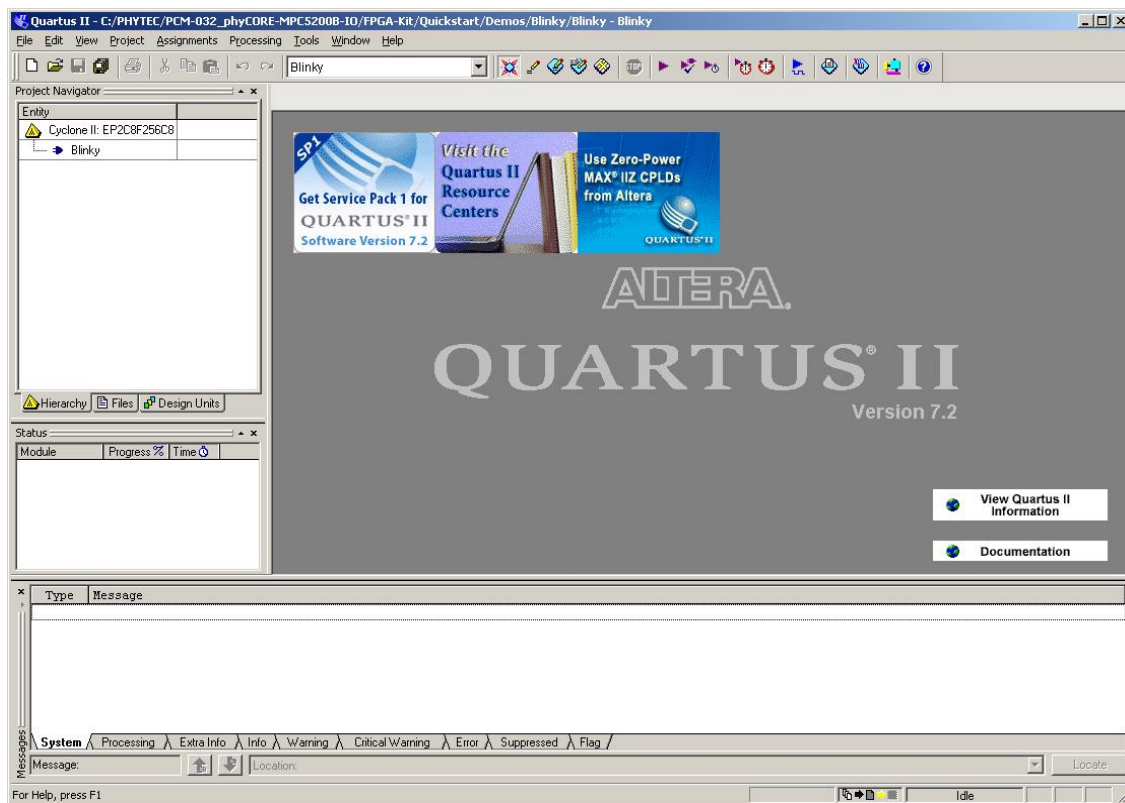


Figure 10: ALTERA Quartus® II – Open demo project

- Now double click on the project name.
- Quartus® II now opens the associated VHDL-File with the project sources.
- Have a look at the source code and try to understand the basic functionality.
- Choose *Programmer* from the *Tools* menu.

Quartus® II now opens the Programmer with the corresponding programming file.

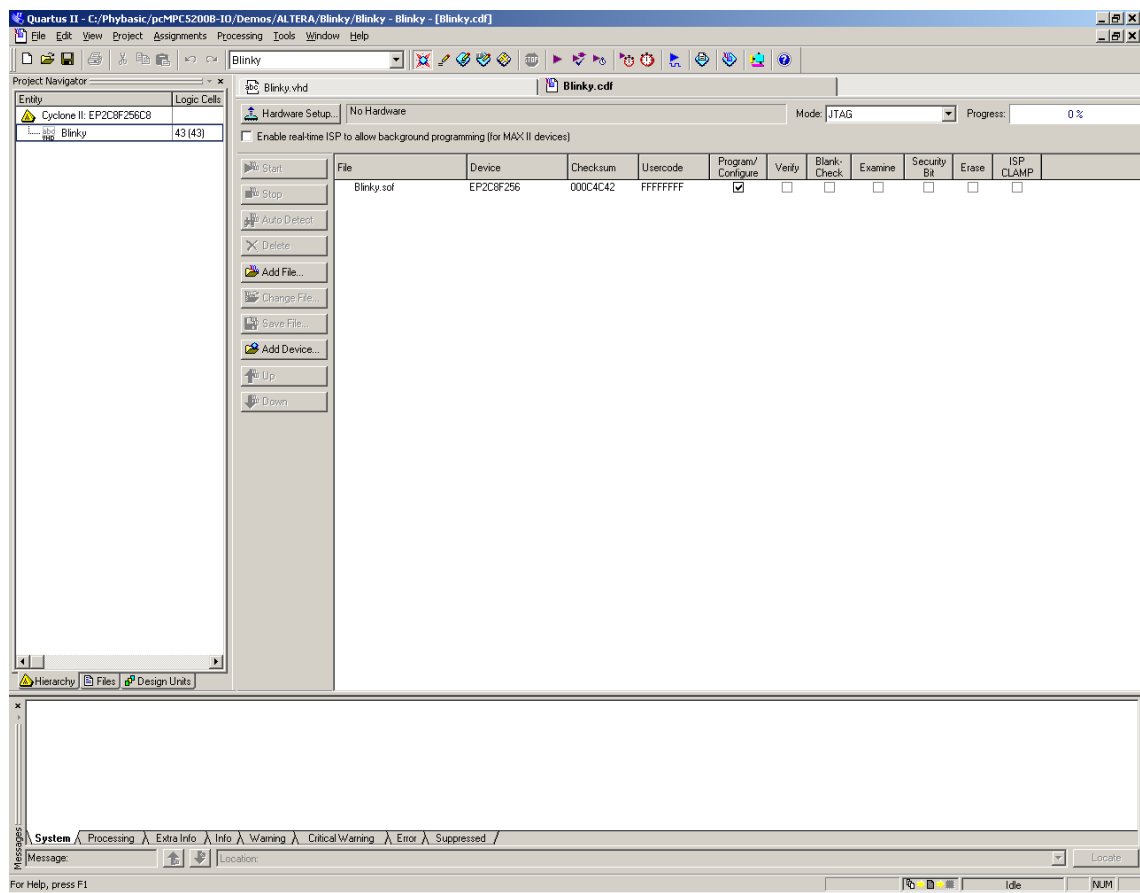


Figure 11: ALTERA Quartus® II – Programmer window with the loaded programming file

- Make sure that the checkmark in the checkbox *Program/Configure* is selected.
- In order to configure the programming device adapter start the hardware setup, by clicking the *Hardware Setup* button.

The window shown next opens and allows to perform the Hardware Setup.

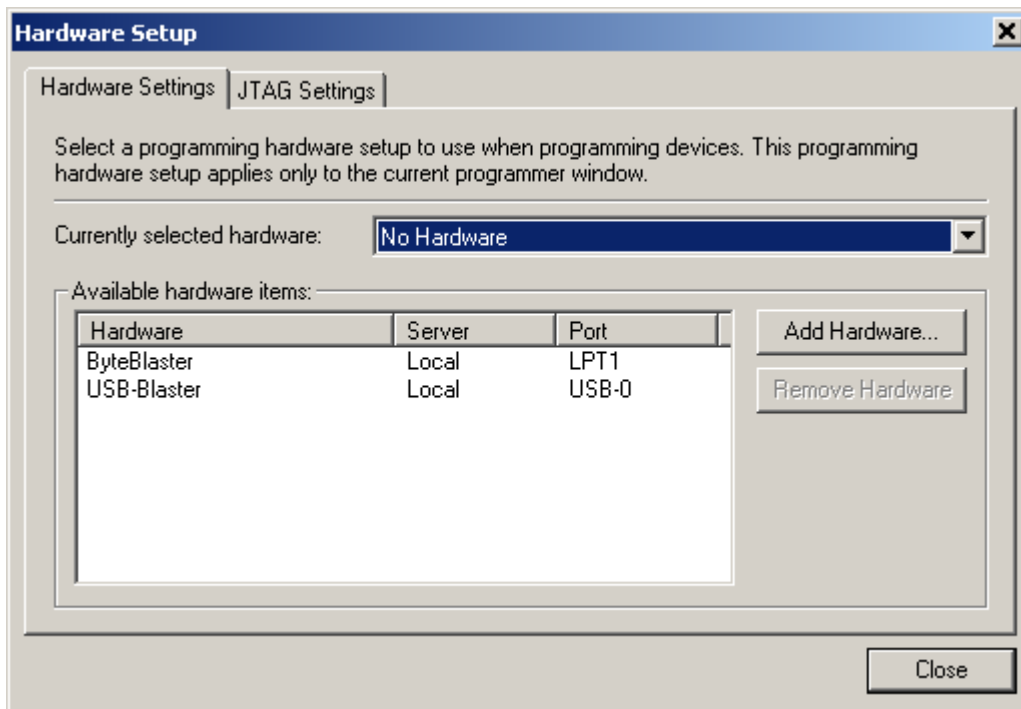


Figure 12: Hardware Setup of the programming interface

- Choose the button *Add Hardware* if no hardware components can be selected in the window *Available hardware items* otherwise skip the next step.
- Now mark the programming hardware which is enclosed in the RDK and add it to the list of available hardware items.



If you have a USB-Blaster, you must install the corresponding USB driver on your desktop PC before executing the hardware setup. Refer to section 2.3.

- In order to complete the setup of the programming adapter choose the correct programming adapter from the drop-down list *Currently Selected Hardware* and finish the setup by clicking the *Close* button.

As shown in the following figure the configuration of the programming adapter should be completed now and you can start with the actual programming process.

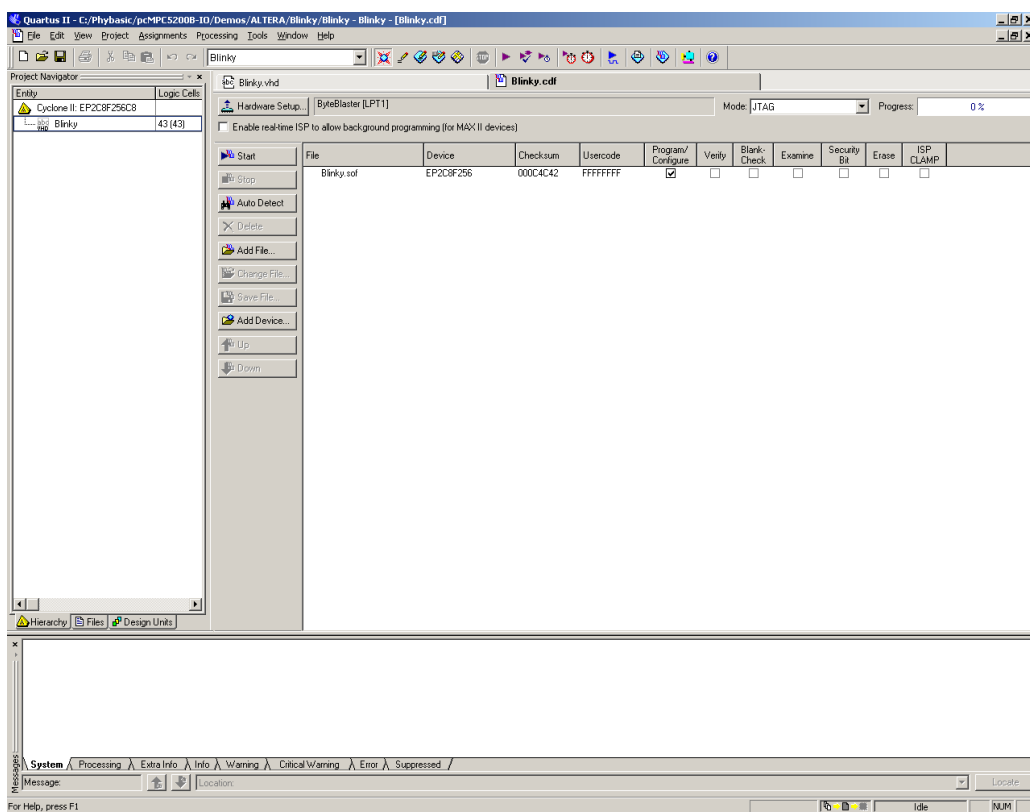


Figure 13 ALTERA Quartus® II – Programmer with completed Hardware Setup

Executing a program on the target:

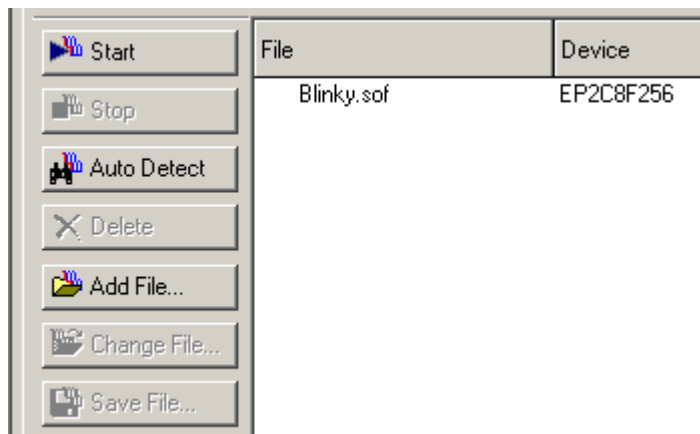


Figure 14: Programmer window – Start button

- Press the *Start* button to start the download process.

The progress of the programming is shown in the upper right corner of the *Programmer* window, whereas additional information on the programming status is displayed in the *System* tab of the message window at the bottom of the screen.

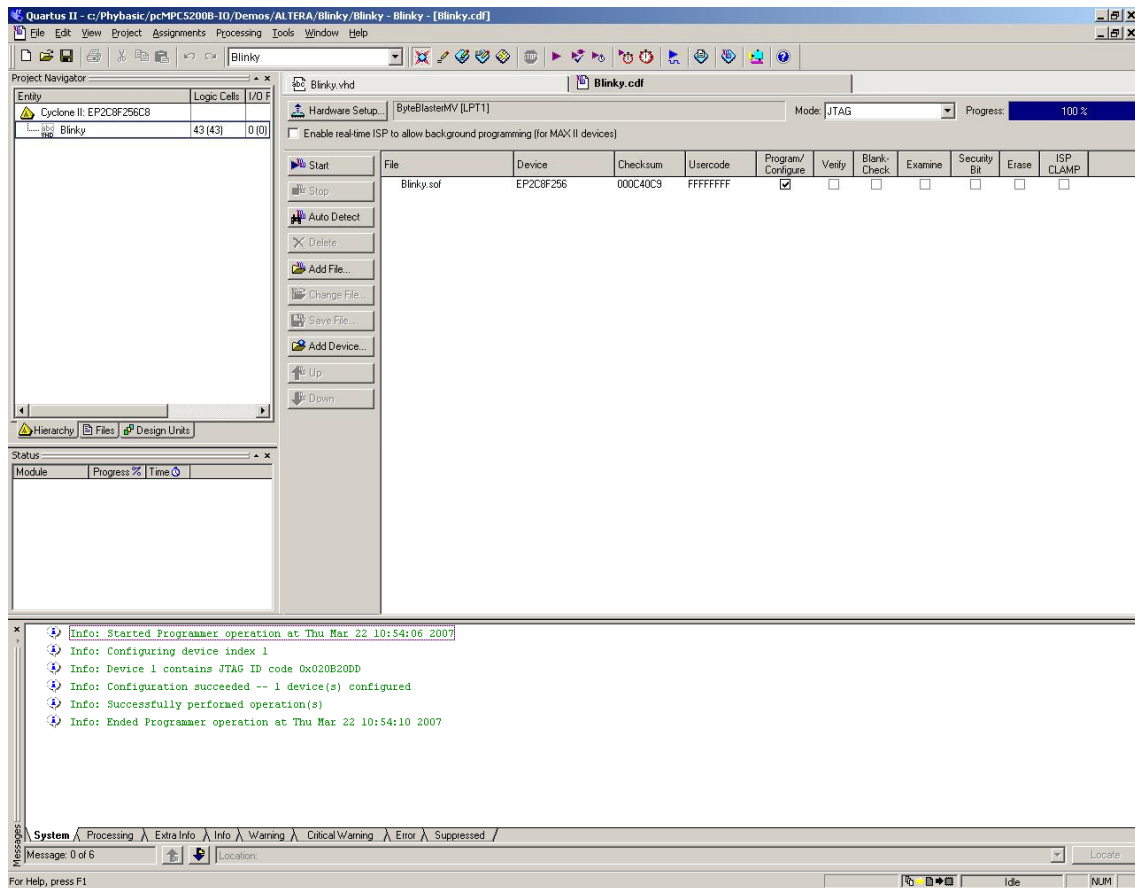


Figure 15: Programmer window after the successful download of the demo program

The program starts automatically and you will see LED D18 of the development board blinking.



Troubleshooting:

If you do not find flashing LED, please check the following:

- Make sure that both jumper J16 and J17 of the development board are closed at position 2 + 3.
- Verify the connection of the programming adapter at the development board as well as at your host PC.
- Check the information about the programming process displayed in the *System* window of Quartus® II tool chain and perform the setup of the programming adapter again if necessary.



You have successfully copied and executed an example on the targets FPGA.



Advanced Information

With the programming tool of the Quartus® II V7.2SP1 web edition software you have even more options:

- Automatic detection of the FPGA-Devices which is attached to JTAG interface.
- Modification of the programming files.
- Adding and deleting of the different FPGA devices which e.g. can be attached to a scan chain.
- Programming of configuration devices which are attached via the active serial interface to the FPGA (e.g. ALTERAEPCS4).

3 Getting More Involved

**70 min**

In this section, you will learn to create, compile and simulate a new project with ALTERA Quartus® II V7.2SP1 Web Edition. Furthermore, you will receive an insight into the connection between FPGA and the MPC5200B on the phyCORE-MPC5200B-I/O on the basis of another project which will give you an understanding of the bus interface. In addition, you will also receive information about the use of the SOPC Builder available in ALTERA Quartus® II in conjunction with hardware specific interfaces of the phyCORE-MPC5200B-I/O.. And finally, the section describes the integration of specially designed logic blocks with the SOPC Builder.

3.1 Creating and simulation of a new Quartus® II V7.2SP1 Web Edition project

3.1.1 Creating a new Quartus® II V7.2SP1 Web Edition project

In this section you will learn how to create and build a new Quartus® II project. First you will create a new project with project specific settings. Then you will write your first vhdl source code for your first own project. After the complete configuration you will compile the new project using the ALTERA Quartus® II V7.2SP1 Web Edition development tool chain and download it to the target's FPGA.

First open ALTERA Quartus® II V7.2SP1 Web Edition



- Double click on the icon *Quartus® II V7.2SP1 Web Edition* on your desktop.

- After starting the Quartus® II software select *New Project Wizard* in the *File* menu to create a new project.

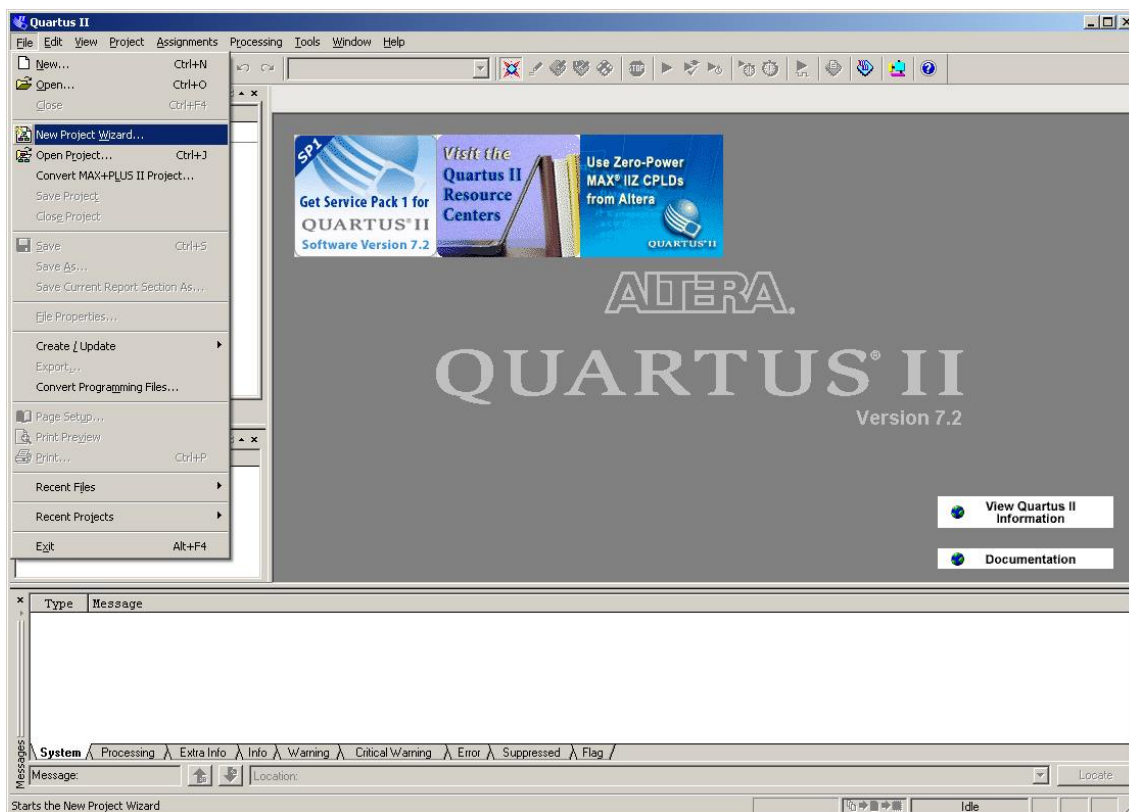


Figure 16: Starting the *New Project Wizards*

- This opens the *Introduction* window of the *New project Wizard*. To continue and to create a new project click on the *Next* button.

Now the directory where all project-related files will be saved for your project must be specified.

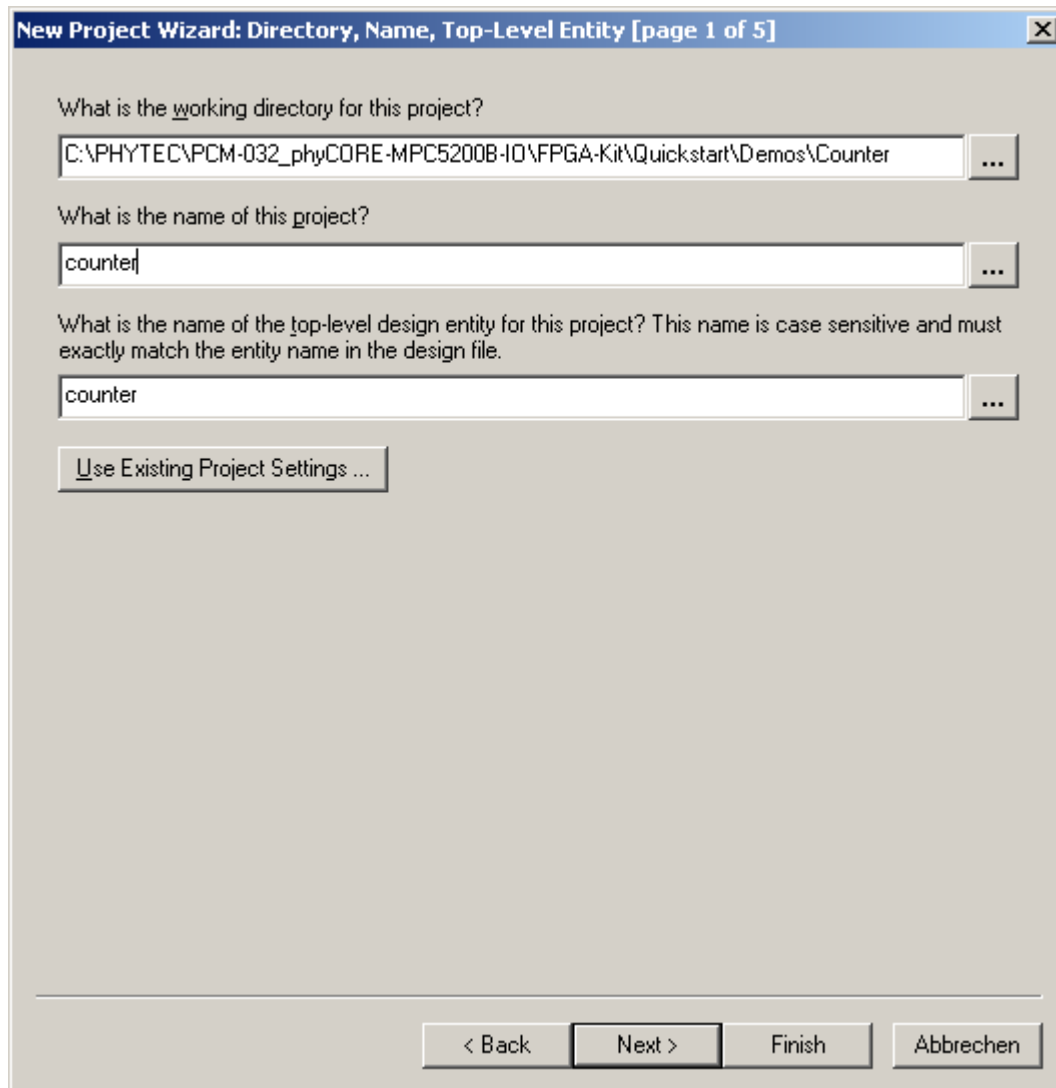


Figure 17: New Project Wizard - Directory, Name and Top-Level-Entity

- As the working directory please enter *C:\PHYTEC\PCM-032_phyCORE-MPC5200B-I/O\FPGA-Kit\Quickstart\Demos\Counter* in the appropriate fields of the *NewProject Wizard* window as shown above.
- For the project's name and the top-level design entity please enter *counter* in the appropriate fields. Click the *Next* button to confirm the entries.



The working directory was previously created during the installation.

If the specified path name or the specified folder does not exist on your drive you will be prompted to confirm their creation. Please confirm the creation of the specified folder with *OK*.

In the next window you can add all available sources in the form of Verilog- or VHDL-files to the newly created project.

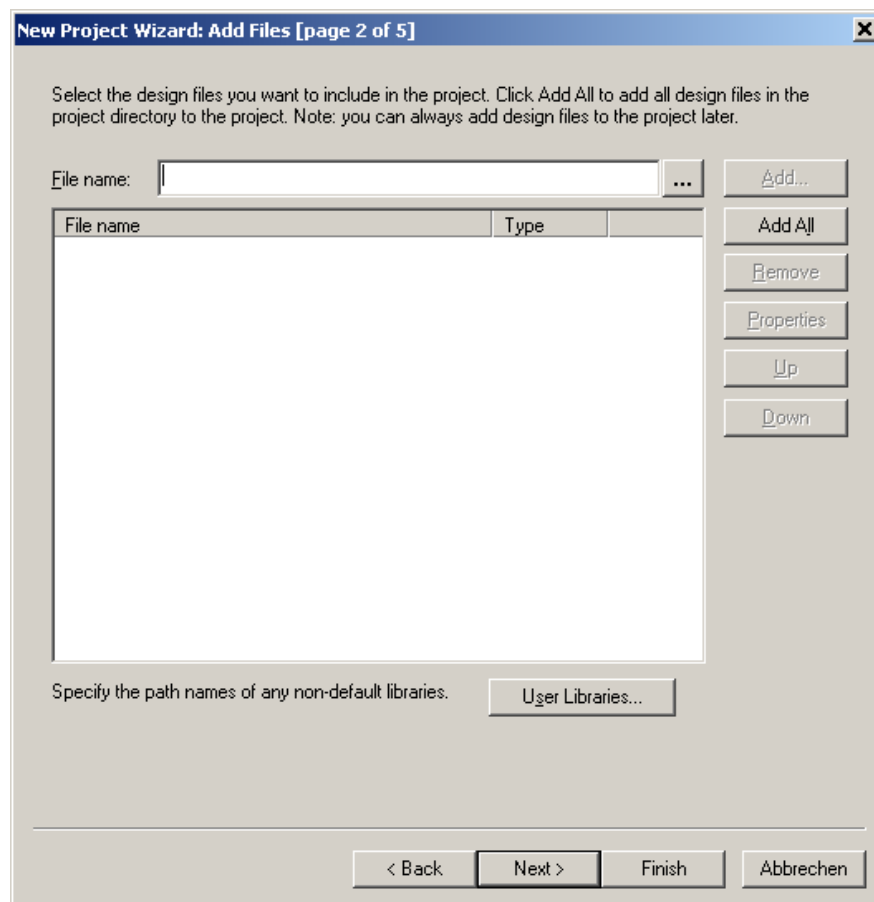


Figure 18: Adding Logic sources in the Project Wizard

As there are no files available yet we can skip this step. Generally adding of design files is possible at any time.

- Click the *Next* button to exit the window without adding design files.

Now the configuration window for selecting the FPGA family and for configuring the specific FPGA device settings opens.

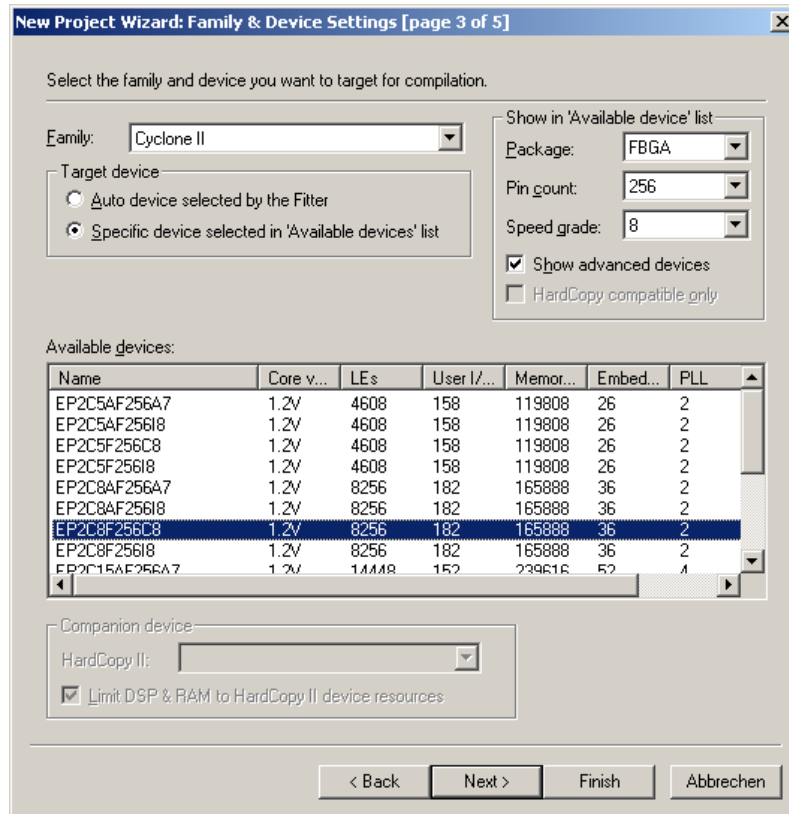


Figure 19: Family and Device configuration

- Select the following settings in the *New Project Wizard: Family & Device Settings [page 3 of 5]* window:
 - Family: Cyclone II
 - Target device: Specific device selected in 'Available devices' list
 - Package: FBGA
 - Pincount: 256
 - Speed Grade: 8
 - In the window 'Available Devices' please choose EP2C8F256C8 or EP2C8F256I8.
- Please confirm the settings by clicking the *Next* button.

- This now opens the window for configuring the EDA tools which can be used in addition to the Quartus® II V7.2SP1 Web Edition. Unfortunately there are no other tools available at the moment. Hence we can pass over this window without changes by clicking the *Next* button.

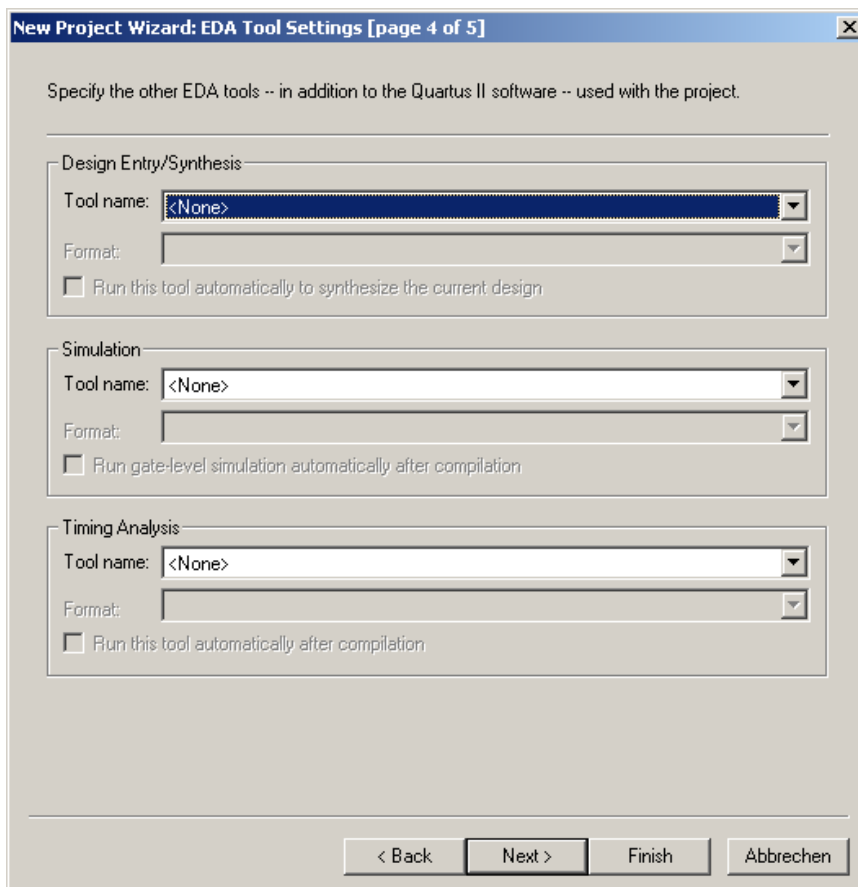


Figure 20: EDA Tool settings

- The last window *Project Wizards : Summary [page 5 of 5]* finally lists all project settings done in the previous steps to allow verification.

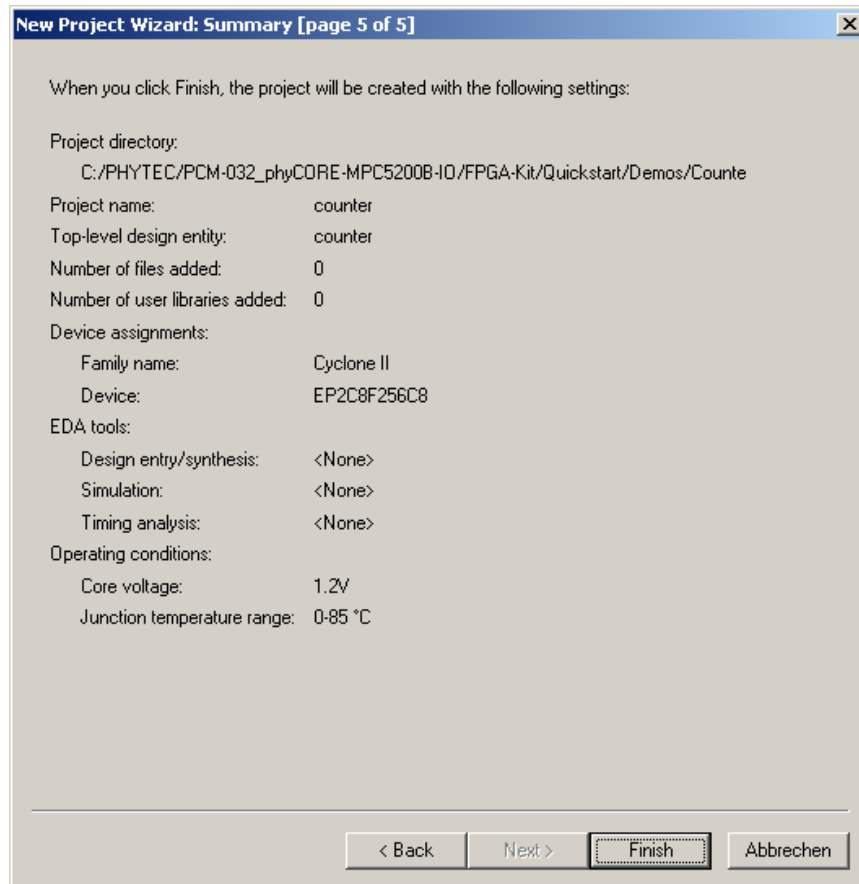


Figure 21: Summary of the project settings

- If all project settings are done properly the summary of the project settings should look the same as in the figure above. Click on the *Finish* button to finalize the project configuration. Corrections in the configuration are possible by using the *Back* button.

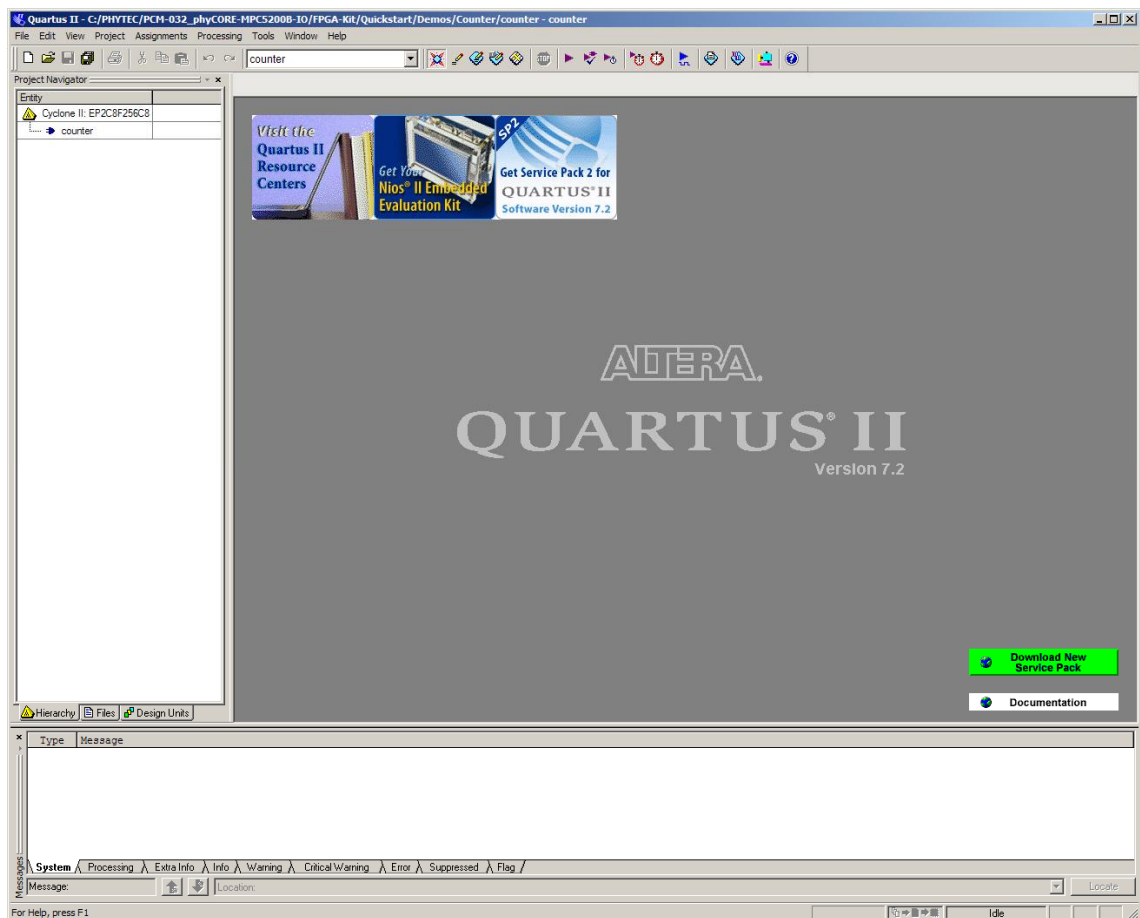


Figure 22: ALTERA Quartus® II with completed project settings

- In order to create a new source file select *New* from the *File* menu. This opens a new window which allows to select the appropriate type of the file we want to add to the project previously created.
- To create a new source file please choose the file type *VHDL File* to add a new VHDL file to the project Counter and confirm with *OK*.

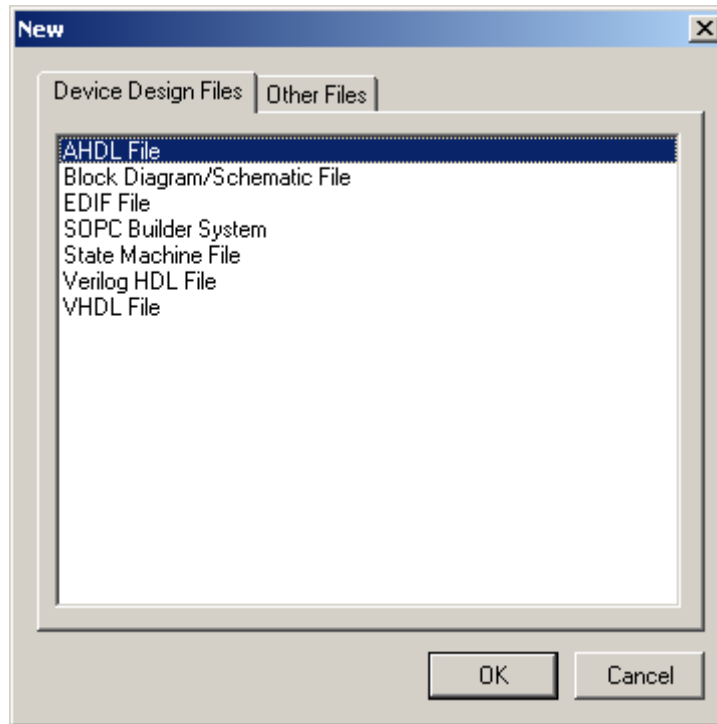


Figure 23: Creating a new design file

ALTERA Quartus® II V7.2SP1 Web Edition opens a new VHDL file. Saving the new file will add this automatically to the project settings of the present project.

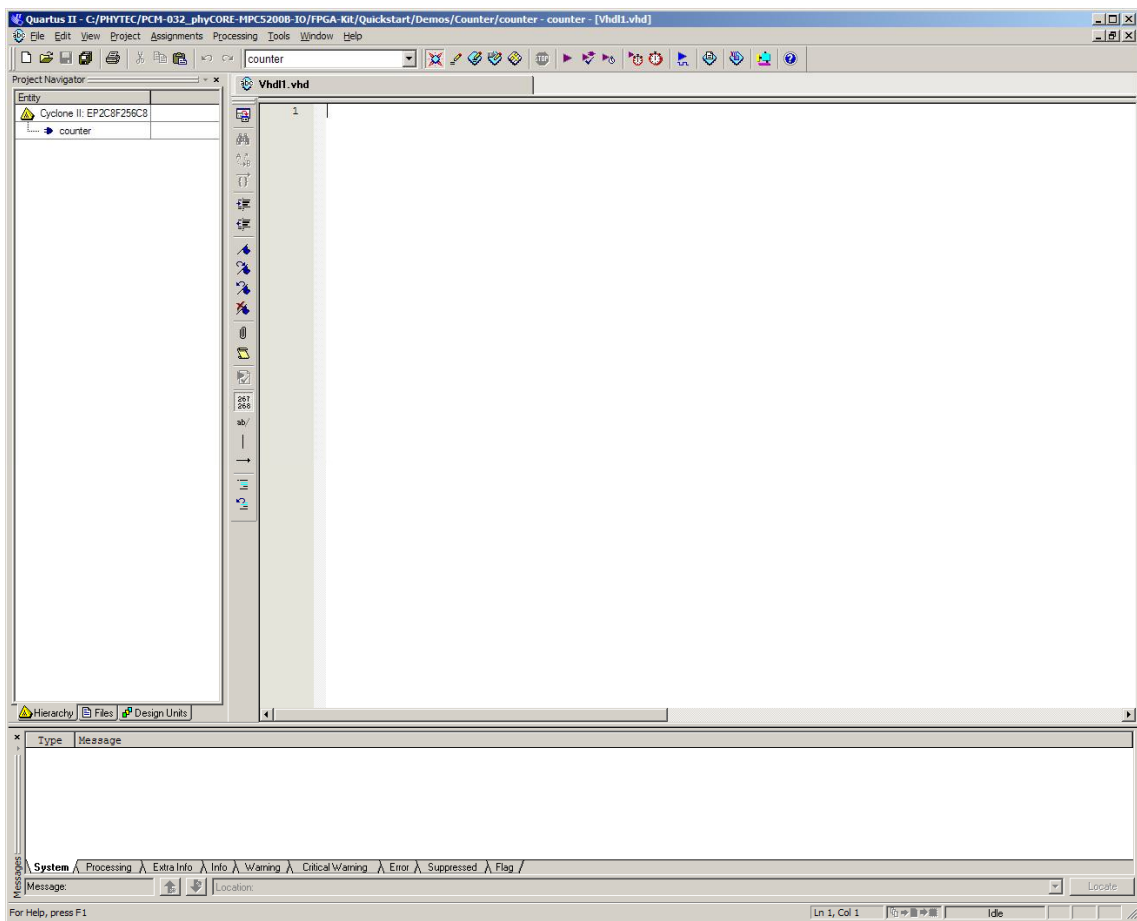


Figure 24: ALTERA Quartus® II with an open design file

- Please enter the following source code for the new VHDL file in the Editor window of ALTERA Quartus® II (the editor window is titled with the name of the file, in this case *Vhdl1.vhd*) :

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY counter IS
  PORT
  (
    clk      : IN   STD_LOGIC;
    clrn     : IN   STD_LOGIC;
    count    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
  );
END counter;

ARCHITECTURE rtl OF counter IS

  SIGNAL count_signal      : UNSIGNED(7 DOWNTO 0);
  SIGNAL pre_count_signal  : INTEGER RANGE 0 TO 1000000;

BEGIN
  PROCESS (clk, clrn)
  BEGIN
    IF clrn = '0' THEN
      count_signal <= (OTHERS => '0');
      pre_count_signal <= 0;
    ELSIF (clk'EVENT AND clk = '1') THEN
      pre_count_signal <= pre_count_signal + 1;
      IF pre_count_signal = 1000000 THEN
        count_signal <= count_signal + "00000001";
        pre_count_signal <= 0;
      END IF;
    END IF;
  END PROCESS;
  count <= STD_LOGIC_VECTOR(count_signal);
END rtl;
```

Figure 25: Source code of the project Counter

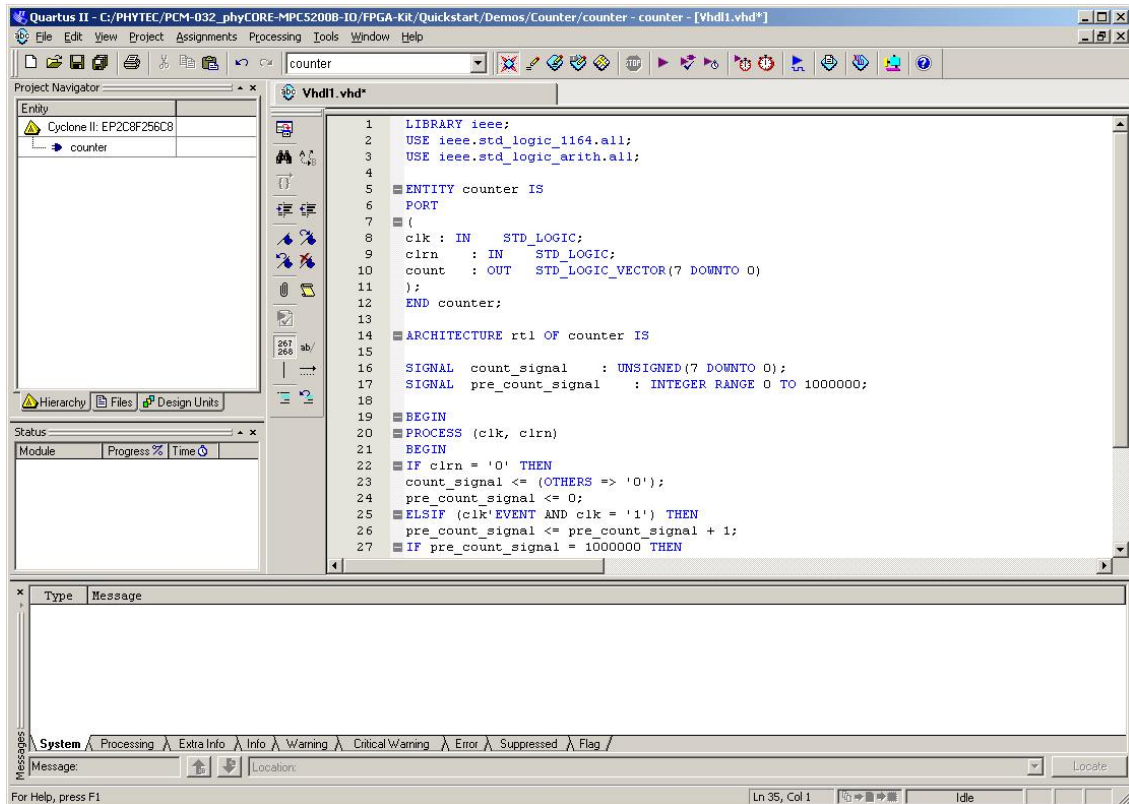



Figure 26: ALTERA Quartus® II with the source code implemented for the project Counter

- After entering the source code, it is necessary that you save the VHDL file by clicking the icon *Save* or choosing *Save as* from the *File* menu. The new VHDL file can be saved e.g. under the name counter.vhd. If the appropriate checkbox was checked during saving the file it is automatically added to the project settings and is now available for the present project.
- Now click on the icon *Settings*  to enter additional project settings or to check newly incorporated project settings.

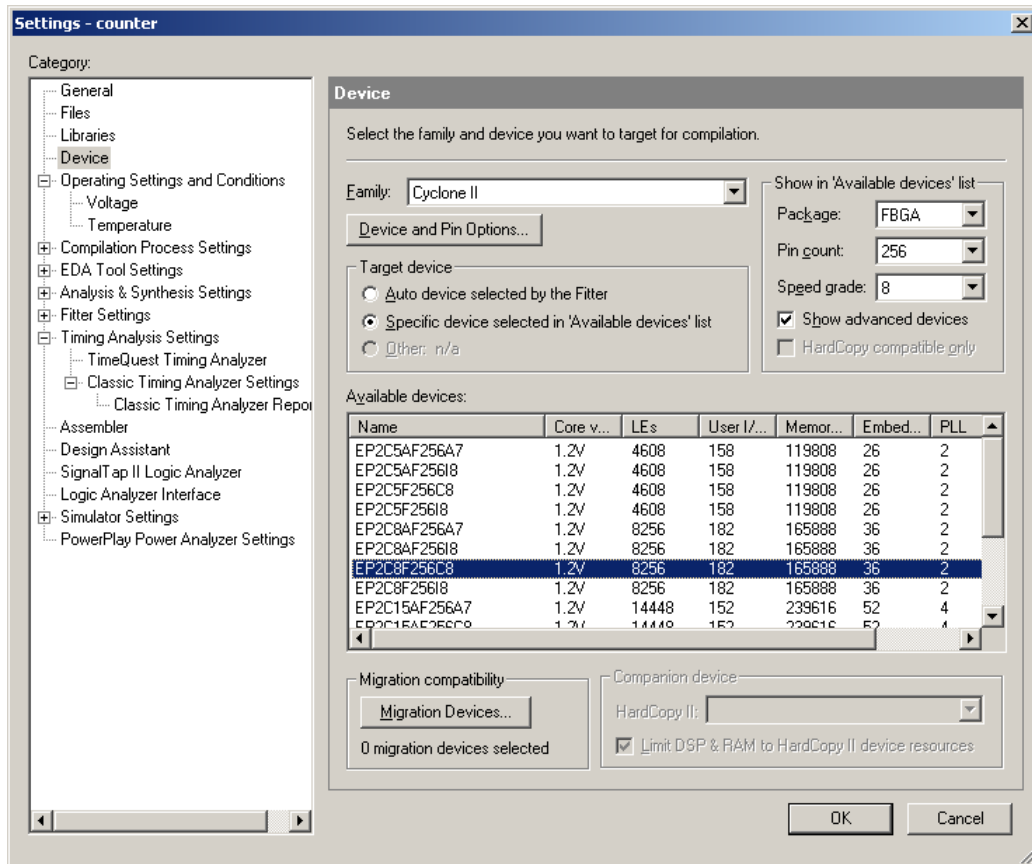


Figure 27: Additional project settings

- In the category *Device* other project settings are to be carried out. Please click on the *Device and Pin Options...* button.
- Select the tab sheet *Unused Pins* and choose *As input tri-stated with weak pull-up resistors* from the drop down list *Reserve all unused pins* to achieve a well-defined behavior of the unused pins. The default option should not be used, because pins of the FPGA, which are connected otherwise (e.g., to the bus system of the MPC5200B) could accidentally float against corresponding quiescent levels.

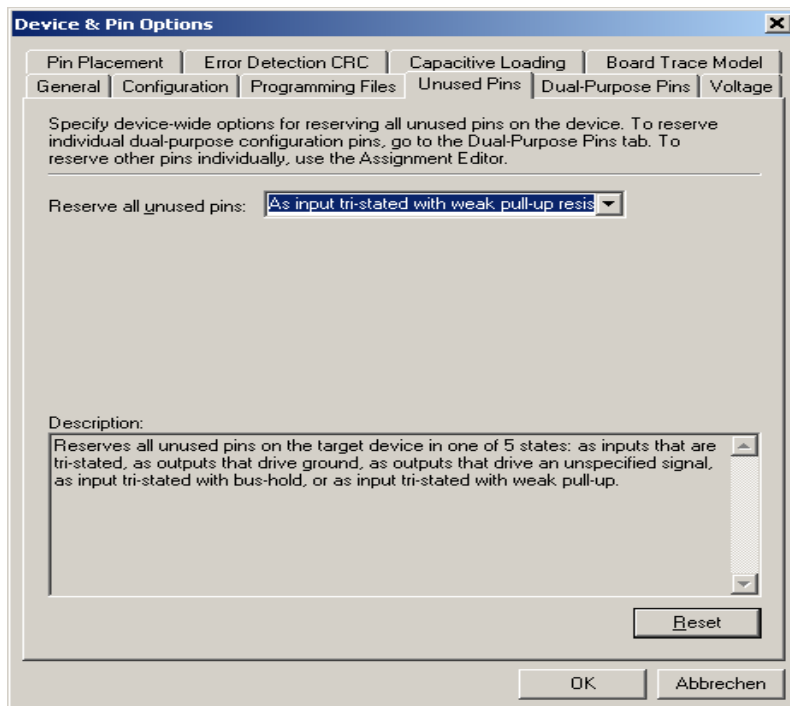


Figure 28: Device and Pin Optionen – Unused Pins

- Click on the *OK* button to confirm all settings made and to return to the Setting window.
- Now expand the category *Timing Analysis Settings*.
- Choose *Classic Timing Analysis Settings* from this category and click on the *Individual Clocks...* button.

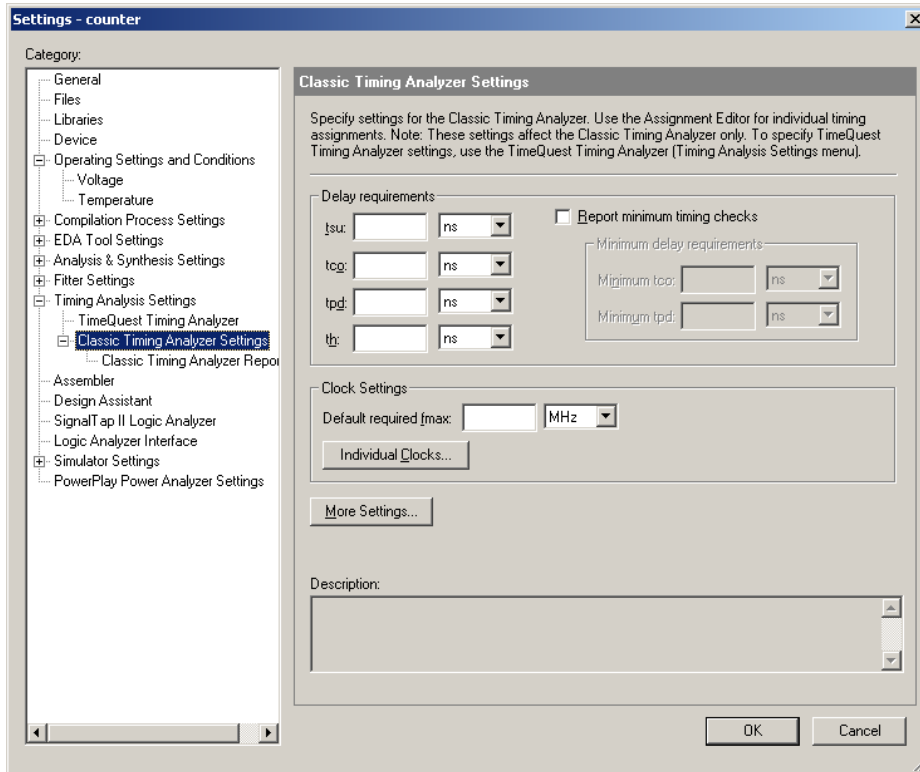


Figure 29: Classic Timing Analysis Settings

- In the next window click on the *New* button to add a new clock signal.

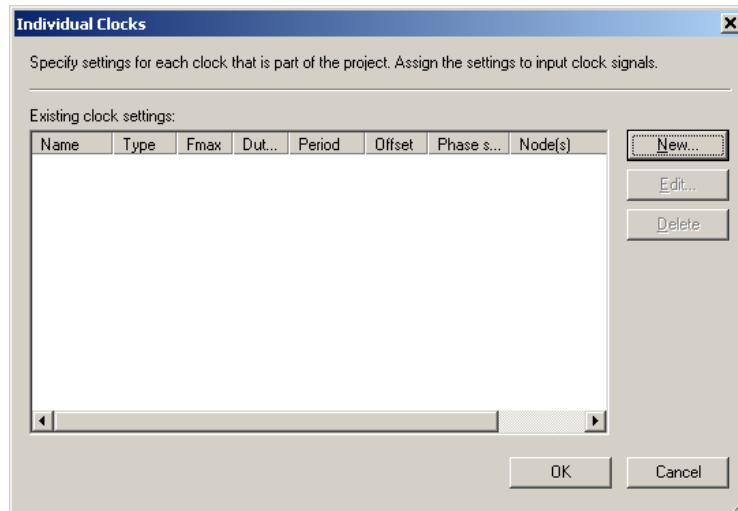


Figure 30: Individual Clocks

- Choose any name for the clock setting and enter the clock name appropriate for the design into the field *Applies to node*.

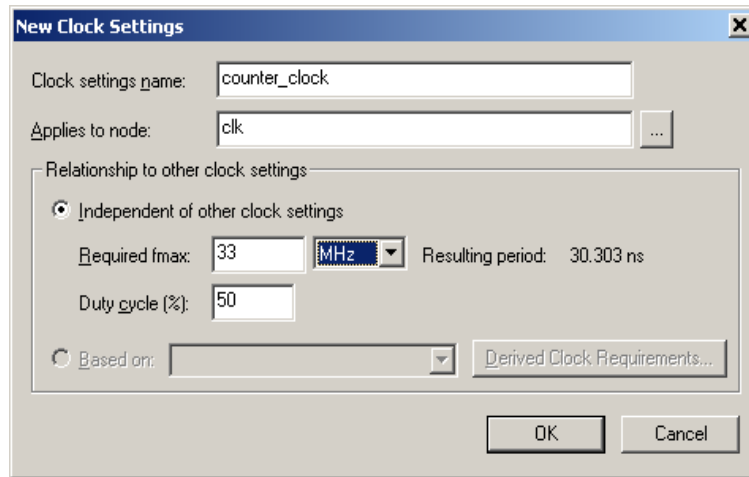



Figure 31: New clock settings

- Enter the frequency of this signal in the field *Required fmax* according to the clock signal from the phyCORE-MPC5200M-I/O (usually 33MHz) and confirm your settings by clicking the *OK* button until you returned to the Main Project Window.

For the further configuration of the project and for the definition of the different IOs of the FPGA the assignment editor has to be started. This editor helps to define the IO-routing and provides information about the attached IOs for the compiler or the fitter of the Quartus® II V7.2SP1 Web Edition.

- Start the Assignment Editor by selecting *Assignment Editor* in the *Assignments* menu or by clicking the *Assignment Editor* icon .

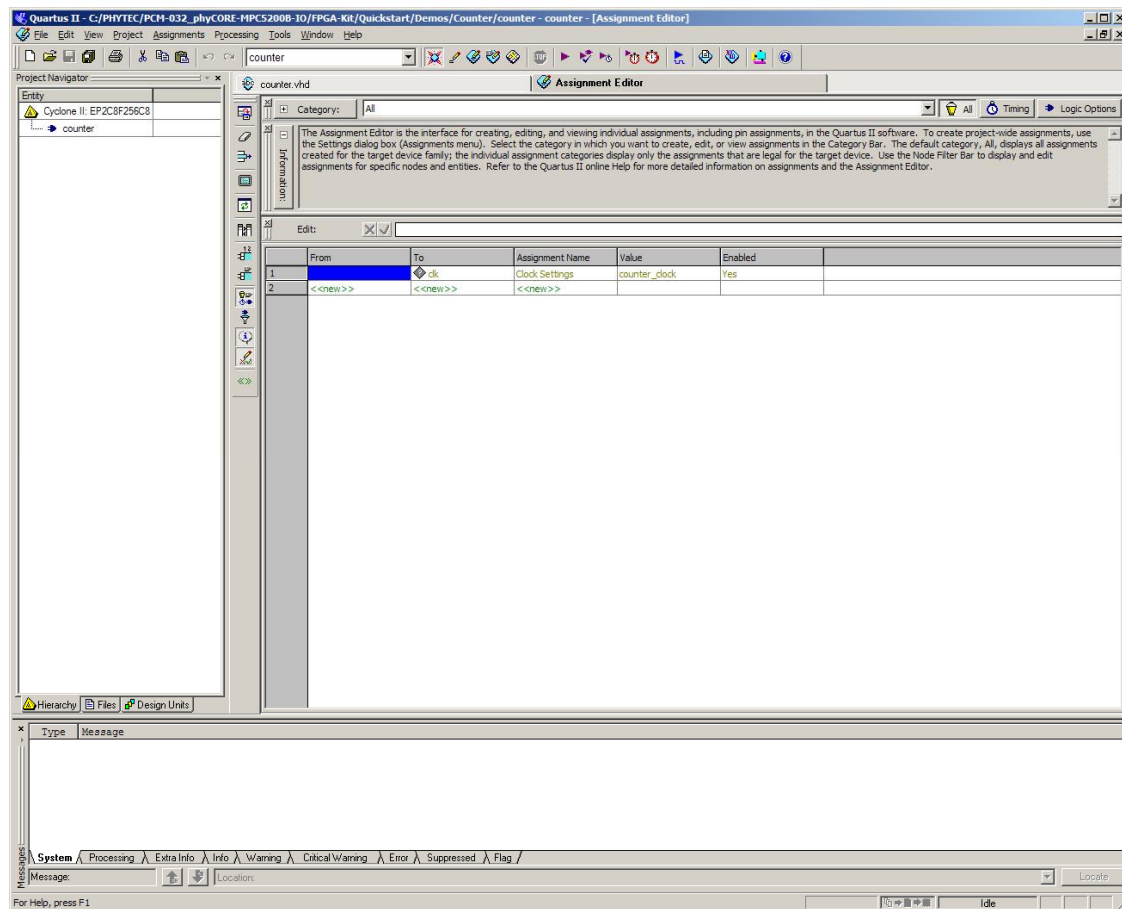


Figure 32: The Assignment Editor of the Quartus® II tool chain

- If necessary click the *Information* button to minimize the information bar and afterwards the *Category* button to expand the category bar in order to accomplish the assignments more easily.
- To define the functions of the FPGA's IOs select *Pin* from the category *Locations* within the category bar.

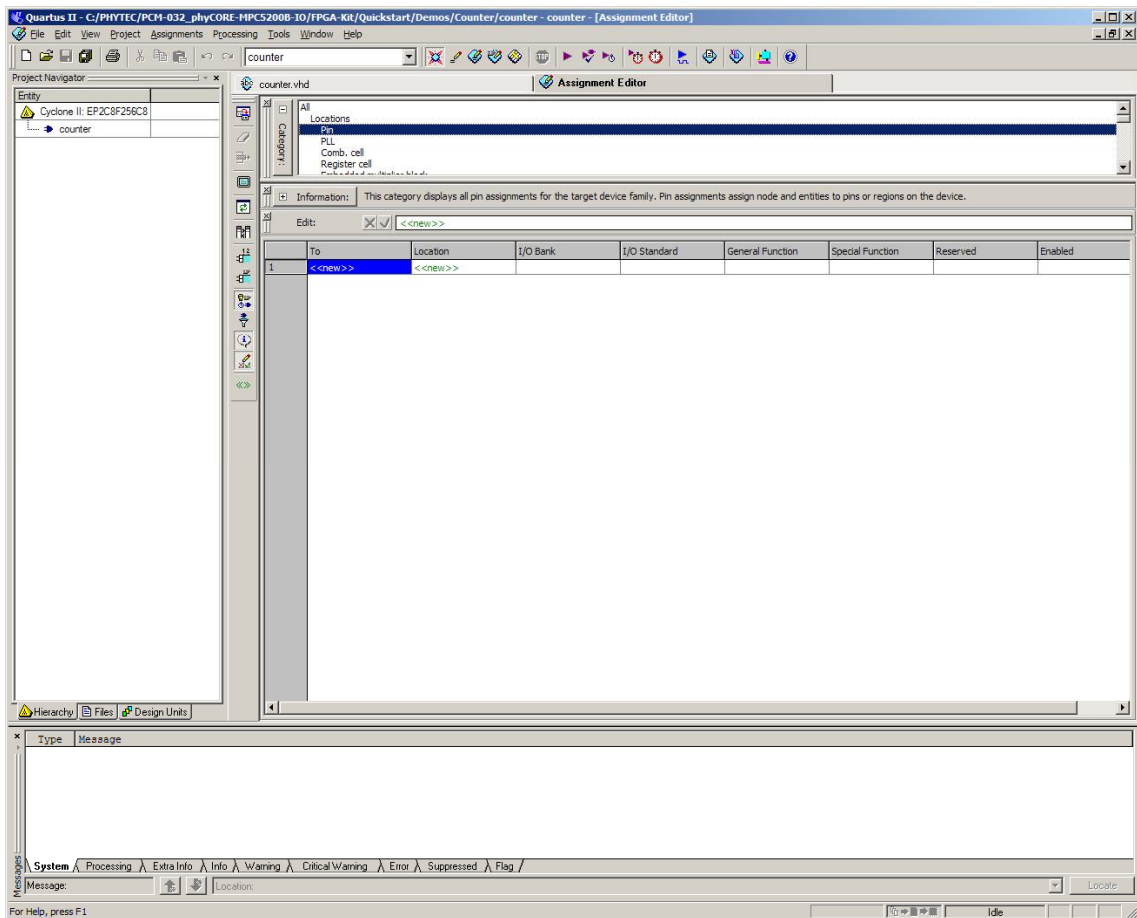


Figure 33: Assignment Editor with expanded category bar

- To start the assignment for a specific pin first double click on the word *new* in the first column (*TO* column) of the spreadsheet.
- Now enter the name of the clock input for this example project (clk) and confirm by pressing *Enter*.
- Next you have to specify the pin used to attach the clock signal to the phyCORE-MPC5200B-I/O. Double click on the cell in the *Location* column next to the name you have just entered. In this example select *PIN_H2* from the drop-down list. You can also start typing the pin number and let the Assignment Editor automatically complete it for you.
- Double click on the cell in the *I/O standard* column and choose 3.3-V LVCMOS from the drop-down list.

- Continue to assign all other I/Os of this example in the same manner. The following table shows a complete list of the pin assignments needed.

| To | Location | I/O Bank | I/O Standard | General Function | Special Function | Reserved | Enabled |
|----------|----------|----------|--------------|------------------|-------------------------|----------|---------|
| clk | PIN_H2 | 1 | 3.3-V LVCMOS | Dedicated Clock | CLK0, LVDSCLK0 p, Input | | Yes |
| clrn | PIN_N9 | 4 | 3.3-V LVCMOS | Column I/O | LVDS66p | | Yes |
| count[0] | PIN_M4 | 1 | 3.3-V LVCMOS | Row I/O | PLL1_OUT n | | Yes |
| count[1] | PIN_N4 | 1 | 3.3-V LVCMOS | Row I/O | LVDS0n | | Yes |
| count[2] | PIN_N3 | 1 | 3.3-V LVCMOS | Row I/O | LVDS0p | | Yes |
| count[3] | PIN_N2 | 1 | 3.3-V LVCMOS | Row I/O | LVDS2n | | Yes |
| count[4] | PIN_N1 | 1 | 3.3-V LVCMOS | Row I/O | LVDS2p | | Yes |
| count[5] | PIN_P3 | 1 | 3.3-V LVCMOS | Row I/O | | | Yes |
| count[6] | PIN_P2 | 1 | 3.3-V LVCMOS | Row I/O | LVDS1n | | Yes |
| count[7] | PIN_P1 | 1 | 3.3-V LVCMOS | Row I/O | LVDS1p | | Yes |

Table 1: Pin Assignments

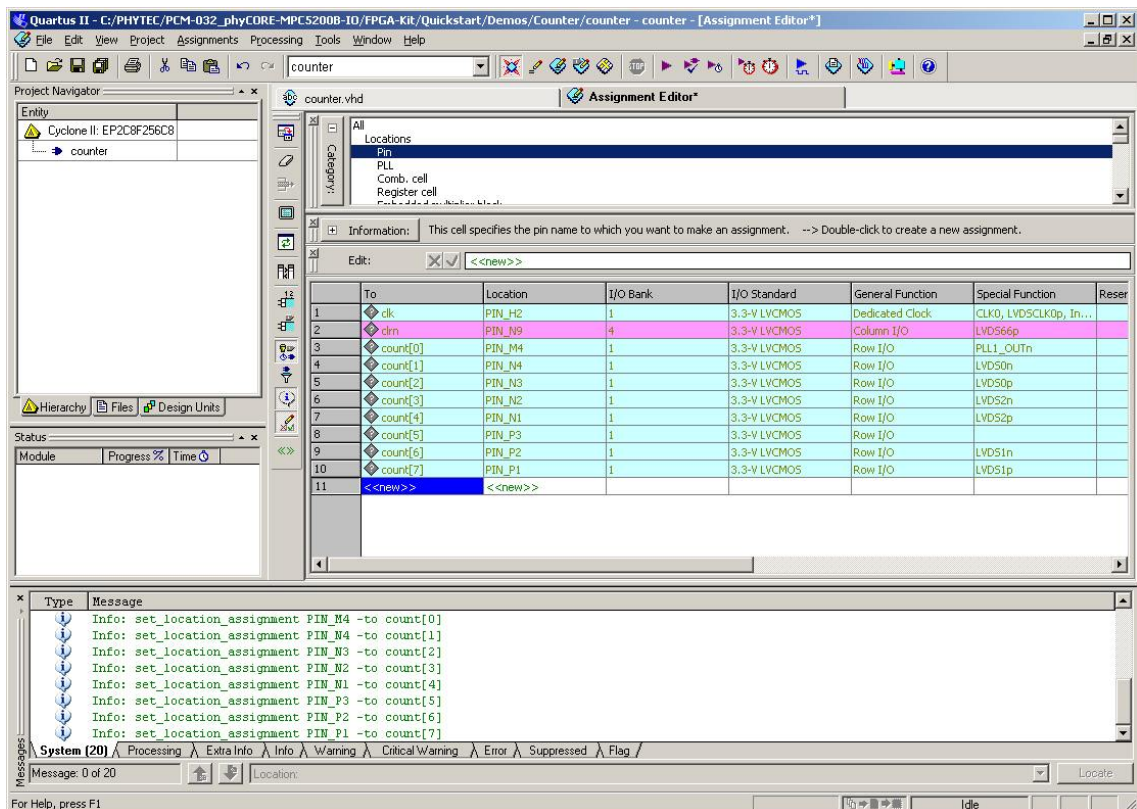


Figure 34: ALTERA Quartus® II after completing the pin assignment



Caution: Configure the signals carefully, otherwise the function of Avalon Slave component can not be guaranteed.

After successfully completing the assignment of the pins the typical IO-Pin load for the outputs must be configured.

- First scroll down in the category bar and select *I/O Features*.
- A new spreadsheet opens in the *Editor* window. As already done in the previous step first double click on the word *new* in the second column (*TO* column) of the new spreadsheet.
- Enter *count[0]* to define the constraint for the first output pin and press *Enter* to confirm.

- Next you have to specify the constraint. Double click on the cell in the *Assignment Name* column, next to the name you have just entered. Select *Output Pin Load* from the drop-down list or type it.
- To enter the correct value click on the cell in the *Value* column. Enter **4** and press *Enter* to confirm.
- Continue to configure the output pin load for all other outputs. The following table shows a complete list of the constraints needed.

| To | Assignment Name | Value | Enabled |
|----------|-----------------|-------|---------|
| count[0] | Output Pin Load | 4 | Yes |
| count[1] | Output Pin Load | 4 | Yes |
| count[2] | Output Pin Load | 4 | Yes |
| count[3] | Output Pin Load | 4 | Yes |
| count[4] | Output Pin Load | 4 | Yes |
| count[5] | Output Pin Load | 4 | Yes |
| count[6] | Output Pin Load | 4 | Yes |
| count[7] | Output Pin Load | 4 | Yes |

Table 2: Constraints for the output pin load

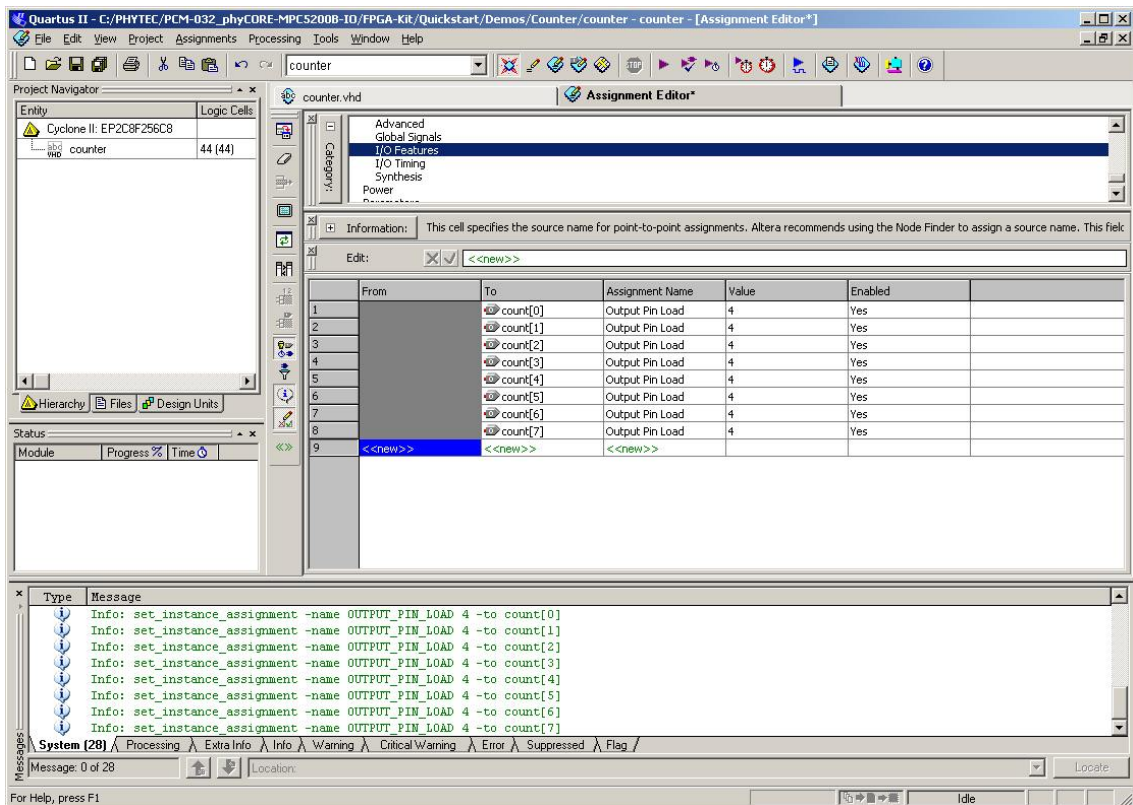



Figure 35: ALTERA Quartus® II after entering the output pin load



Caution: Configure the constraints carefully, otherwise the function of Avalon Slave component can not be guaranteed.

- After completing the pin assignment and entering the constraints, save the project by selecting *Save project* from the *File* menu.

Now the project is ready for compilation.

- Click on the *Compilation* icon  or choose *Start Compilation* from the *Processing* menu.

After successful compilation the following screen should show up.

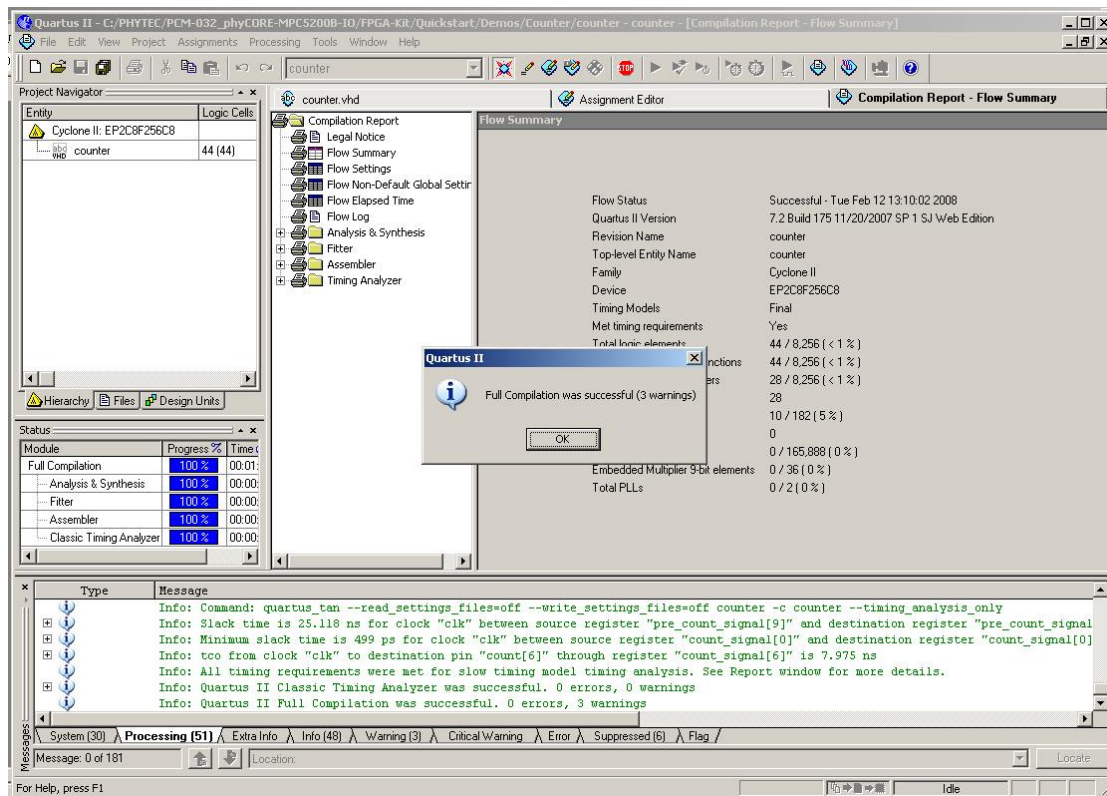


Figure 36: ALTERA Quartus® II after successful compilation of the project



If the process of building the project stops with an error, check your source code and your modified settings especially the assignments done in the assignment editor. Information on errors or warnings will be displayed in the corresponding tab. Simply click on the *Error* or *Warning* tab in the message window at the bottom of the screen.

The 3 warnings you will receive can be viewed in the *Warning* tab of the message window for further evaluation. As they are not critical we can proceed with the download of the new project and the executing on the target.

- Push the *OK* button.

Now you are ready to select *Programmer* in the *Tools* menu in order to download the program to the target.

Ensure that the programming or configuration option is selected by turning on the corresponding checkbox in the programmer.

- Click on the *Start* button to start the download process.

The progress of the programming is shown in the upper right corner of the Programmer window, whereas additional information on the programming status is displayed in the *System* tab of the message window at the bottom of the screen.

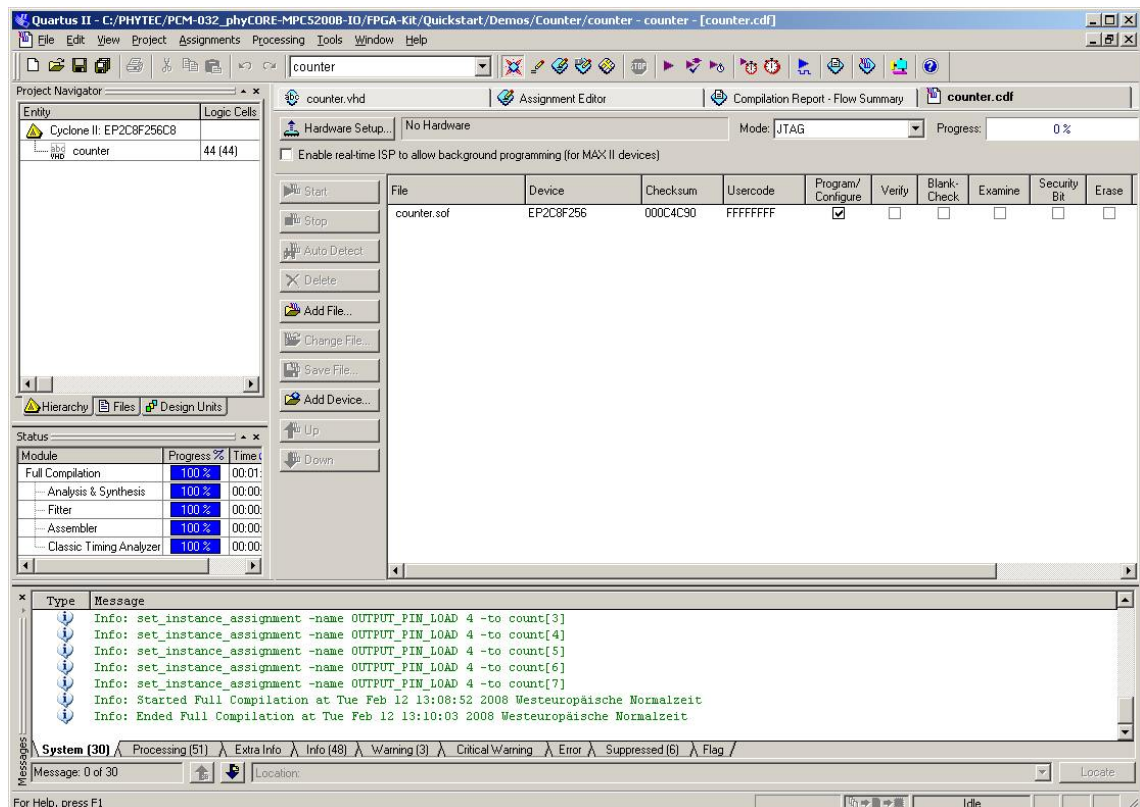


Figure 37: Successful programming of the *Counter* project

After successfully programming the FPGA on the target you will see LED D18 blinking with a lower frequency than in the Blinky project.



You have successfully created and compiled your first Quartus® II V7.2SP1 Web Edition project as well as executed it on the target.

In this section you have learned how to create and compile a new Quartus® II project. Now you are able to implement new logic features in the targets FPGA.

3.1.2 Simulation of the Counter Project with ALTERA Quartus® II V7.2SP1 Web Edition

To simulate the previously created Counter project, a Vector Waveform file is required. Using the menu of the Quartus® II tool chain allows easy creation of this Vector Waveform file.

- Select *Open Project* from the *File* menu to open the previously created Counter project.
- Select *New* from the *File* menu to create a new file.
- Choose the tab sheet *Other Files* in the window that opens now and select Vector Waveform File.

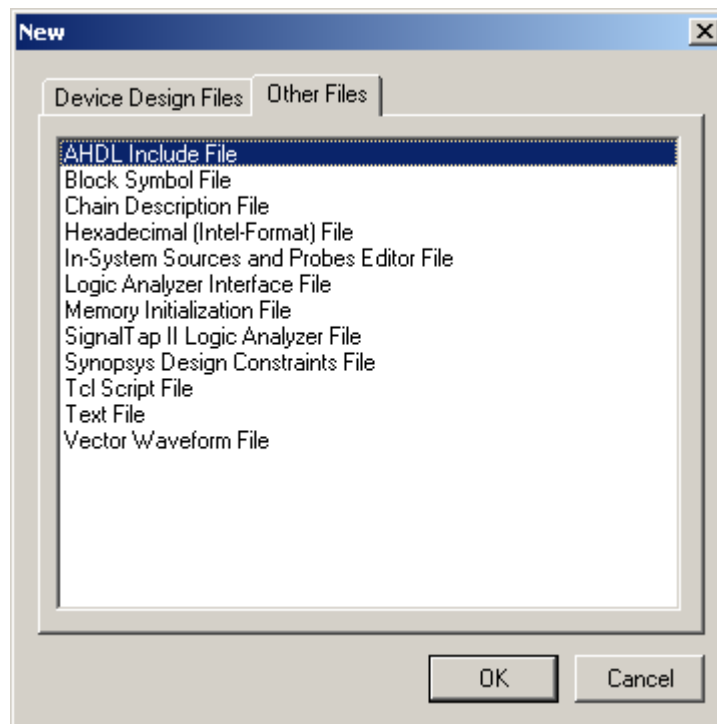


Figure 38: Choosing the right file type

The following figure shows the window that opens after successfully executing the step described above.

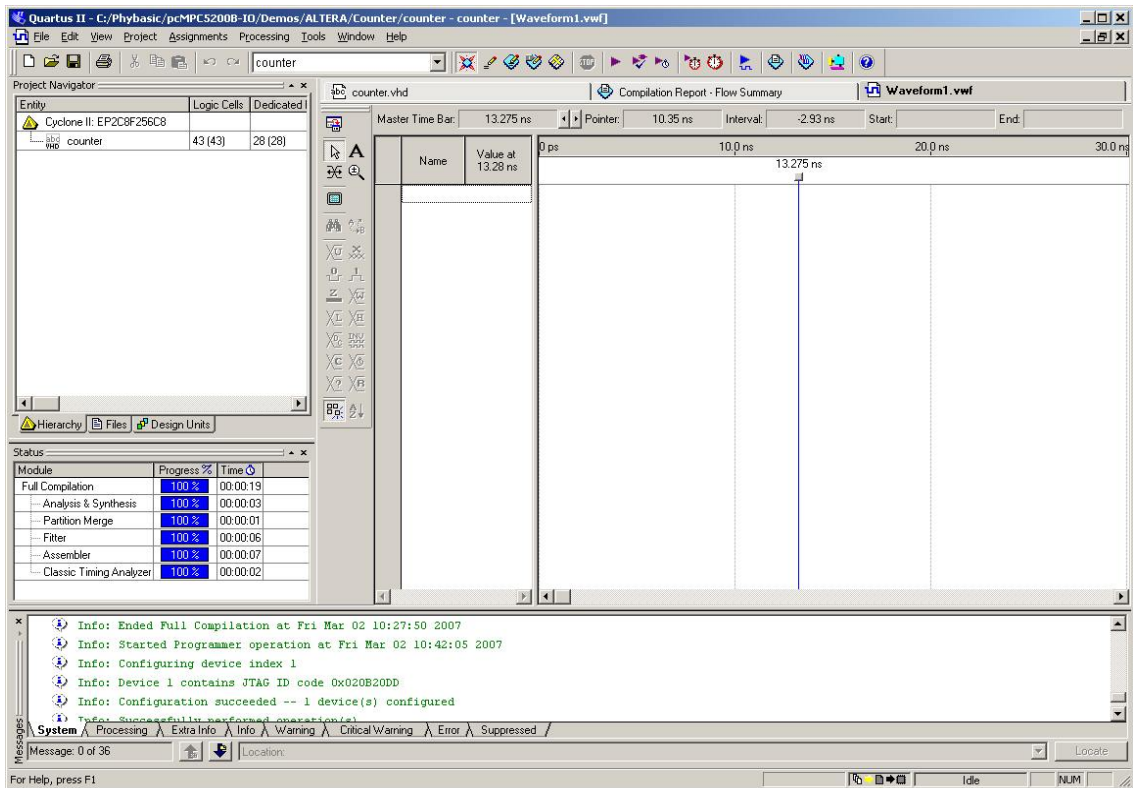


Figure 39: ALTERA Quartus® II with the just created Vector Wave Form File

- Right-click the white area under *Name* in the Waveform Editor window and choose *Insert* and then *Insert Node or Bus* to open the *Insert Node or Bus* dialog box. You can also open the *Insert Node or Bus* dialog box by double-clicking under *Name*.

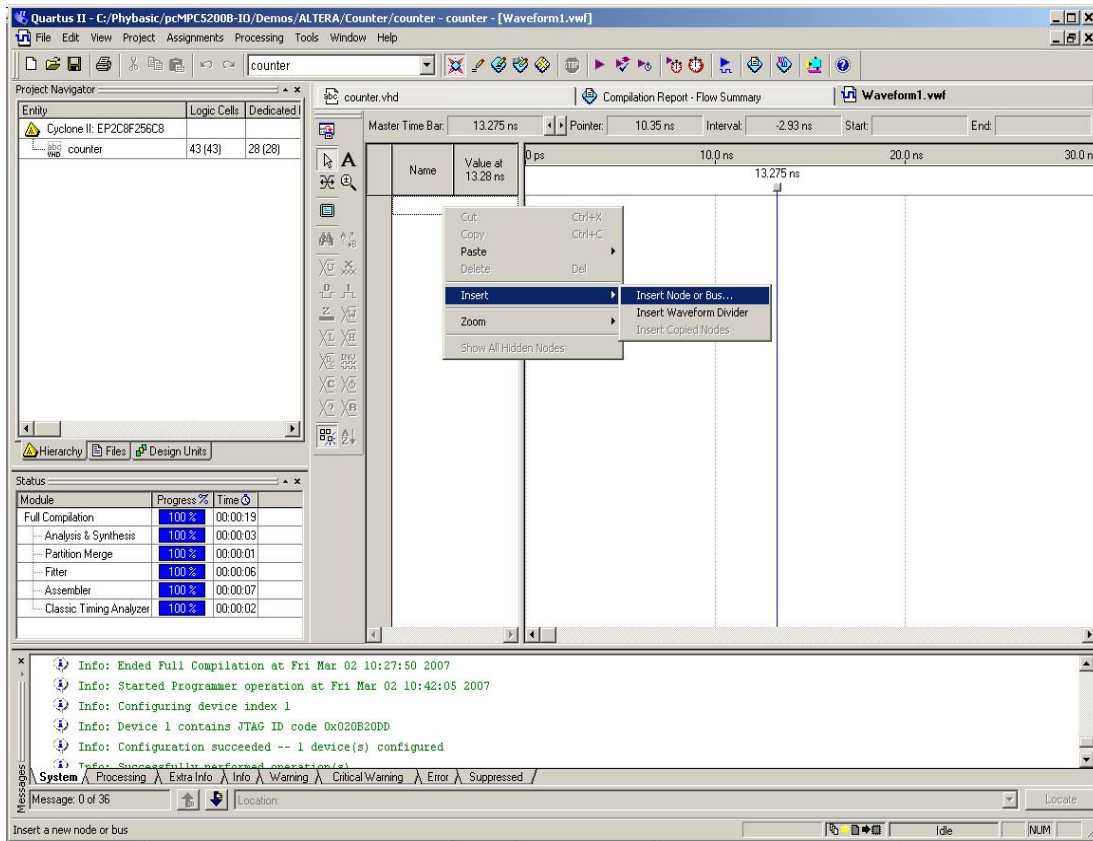


Figure 40: Opening the *Insert Node or Bus* dialog box in order to add new signals to the Vector Wave Form file

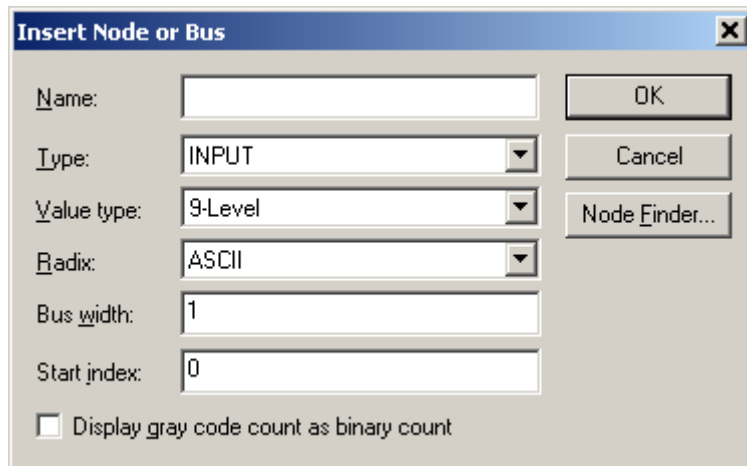


Figure 41: The *Insert Node or Bus* dialog box

- Now click the *Node Finder* button to add different signals to the Vector Wave Form File.

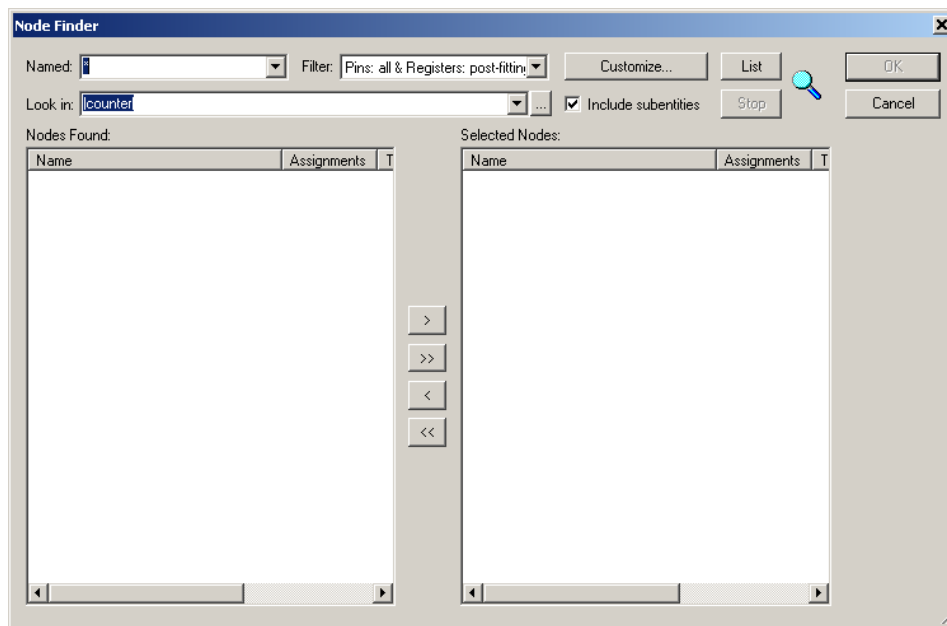


Figure 42: The Node Finder dialog box

- Select *Pins: all & Register: post fitting* from the drop-down list *Filter* and press the *List* button to confirm.

After confirming all IOs and registers available for the simulation will be listed and you can select which signal should be used.

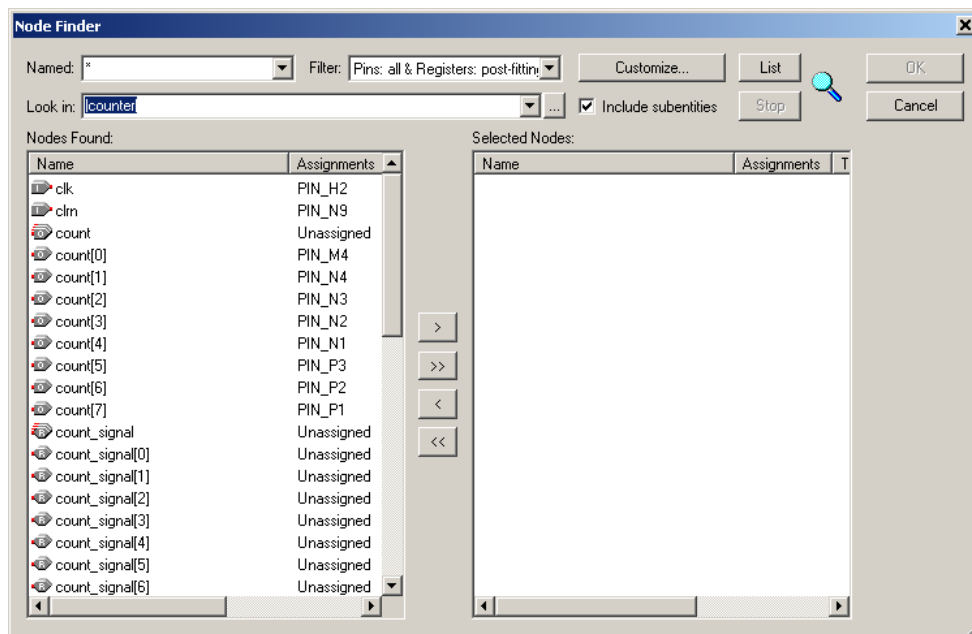



Figure 43: The Node Finder dialog box with the signals available for the simulation

- Mark the signals:
 - clk
 - clrn and
 - pre_count_signal

and click on the  icon to add them to the *Selected Nodes* list.

After successfully adding the different signals the *Node Finder* dialog box should appear as shown in the following figure.

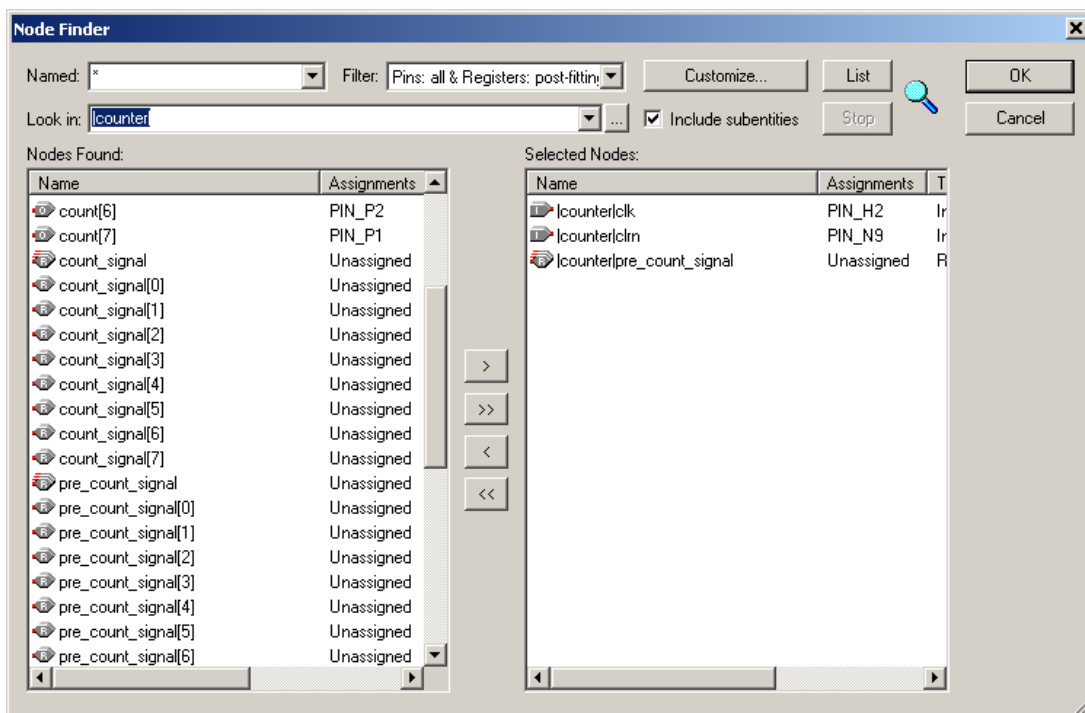


Figure 44: The *Node Finder* dialog box after selecting the signals for the simulation

- Press *OK* to close the *Node Finder* dialog box and to return to the *Insert Node or Bus* dialog box.
- Press *OK* again to return to the Waveform Editor window.

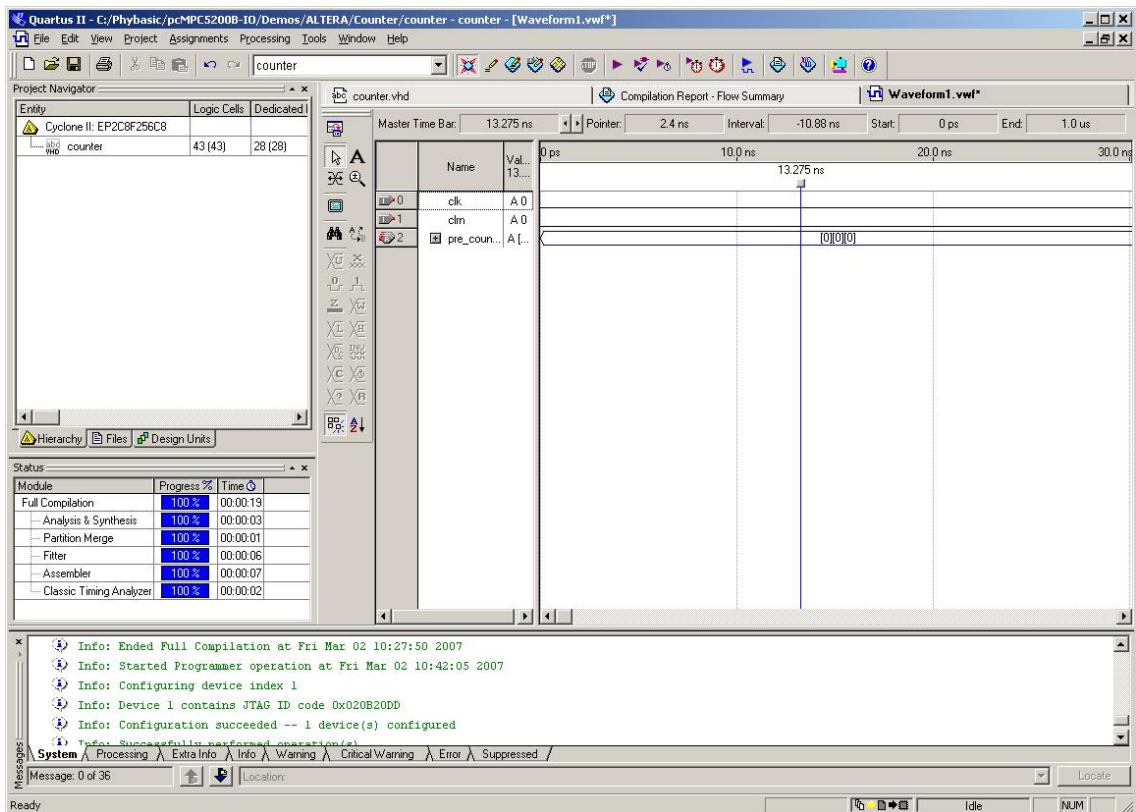



Figure 45: Waveform Editor window with the signals added for the simulation

- To create a waveform for the clock signal mark the clock signal by clicking on its name.
- Click the *Overwrite Clock* button  on the Waveform Editor toolbar to generate a clock signal with the Clock dialog box.
- Enter the following values in the Clock dialog box
 - Start Time: **0 ps**
 - End Time: **1 us**
 - Period: **15 ns**
 - Offset: **0 ns**
 - Duty Cycle: **50 %**
- Press *OK* to confirm.

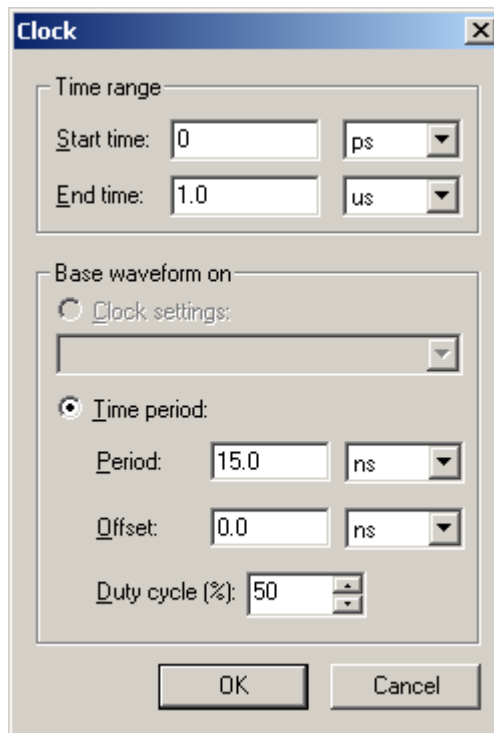




Figure 46: Using the *Clock* dialog box to configure the clock signal

Now you can see, that the waveform of the signal *clk* has changed into a clock signal with a period of 15 ns.



To distinguish the waveforms of the signals easier, it is possible to change the zoom level of the Waveform Editor window. Using *STRG+Shift+Space* zooms in, whereas *STRG+Space* zooms out.

- Now select the signal *clrn* to continue the signal configuration. This signal is low active. Therefore click on the *Forcing High (1)* icon  to change the quiescent level to a high level.
- To define the time where this signal is active (i.e. has a low level) select a portion of the waveform by dragging the cursor over the waveform while you hold down the left mouse button. After releasing the mouse button the box you just created around

the portion you want to change turns blue. Click on the  icon on the Waveform Editor toolbar to change the signal level of the selected portion to a low level.

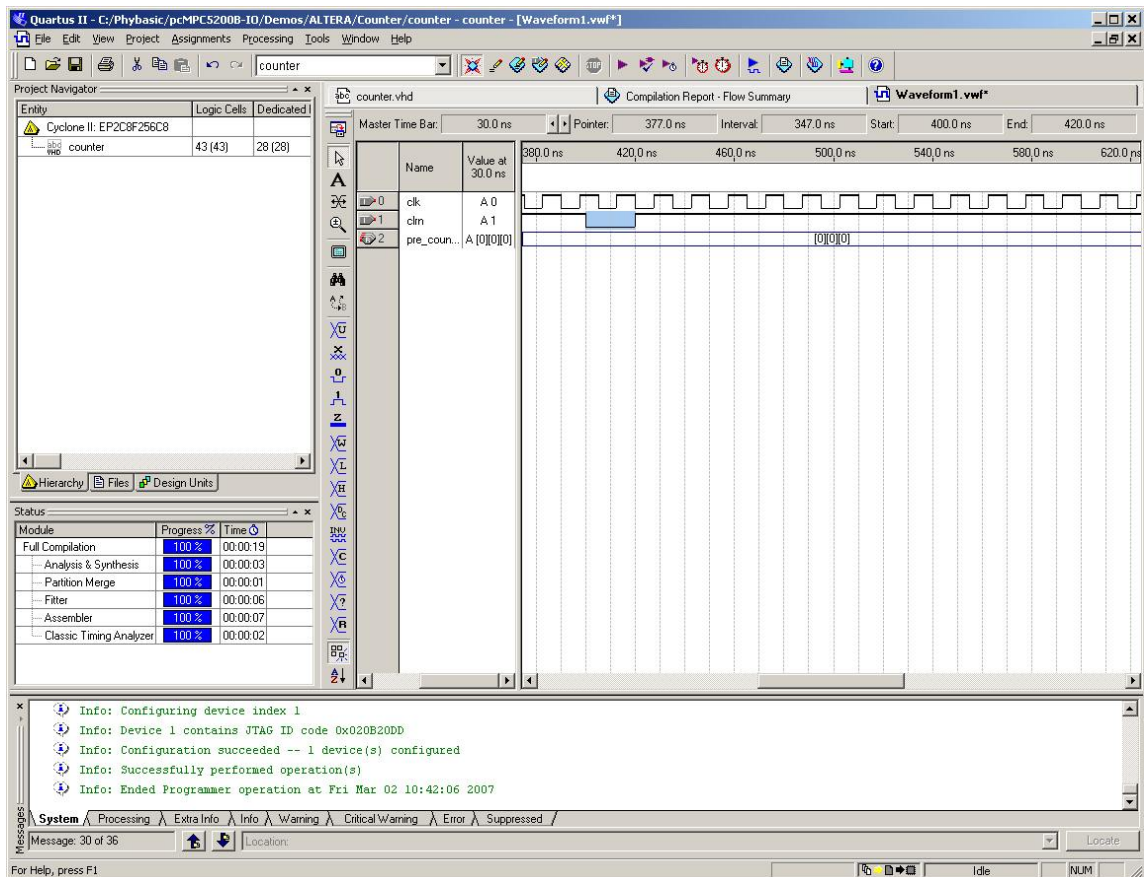



Figure 47: Configuration of the *clrn* signal with the *Waveform Editor*

- To continue click on the  icon or choose *Save As* from the *File* menu. In the *Save As* dialog box enter *counter* and choose *Vector Waveform File (*.vwf)* as file type. Ensure that the checkbox *Add file to current project* is checked.
- Select *Simulator Tool* from the *Processing* menu to proceed with the configuration of the simulator.

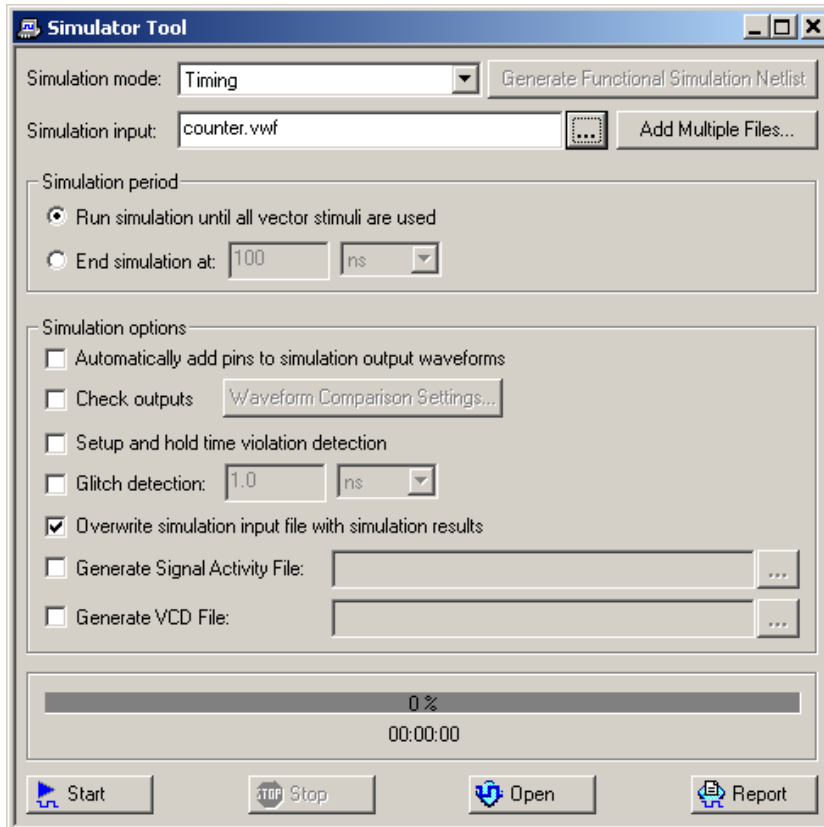


Figure 48: The Simulation Tool of the Quartus® II tool chain

- Verify that the Vector Waveform file counter.vwf is selected in the *Simulation input* box.
- From the *Simulation options* deselect all options except *Overwrite simulation input file with simulation results*.
- Press the *Start* button to start the simulator.

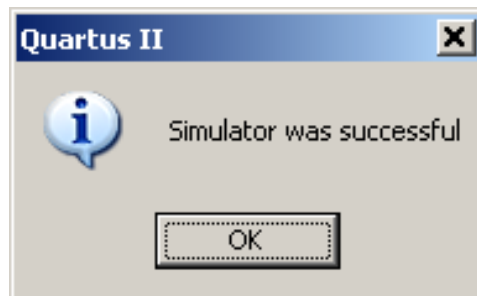


Figure 49: Successful simulation of the project

If the simulation has been carried out successfully Quartus® II V7.2SP1 Web Edition shows the message shown in the previous figure.

- Confirm the successful simulation by pressing the *OK* button.
- Click on the tab of the *Waveform Editor* window to change to this window and to view the waveforms generated by the simulation.
- Press *Yes* to confirm to reload the file. This way the waveforms generated during the simulation will be saved to the Vector Waveform file and can be viewed. Otherwise the results of the simulation will be discarded.
- Click on the plus sign in front of the `pre_count_signal` to expand this signal.

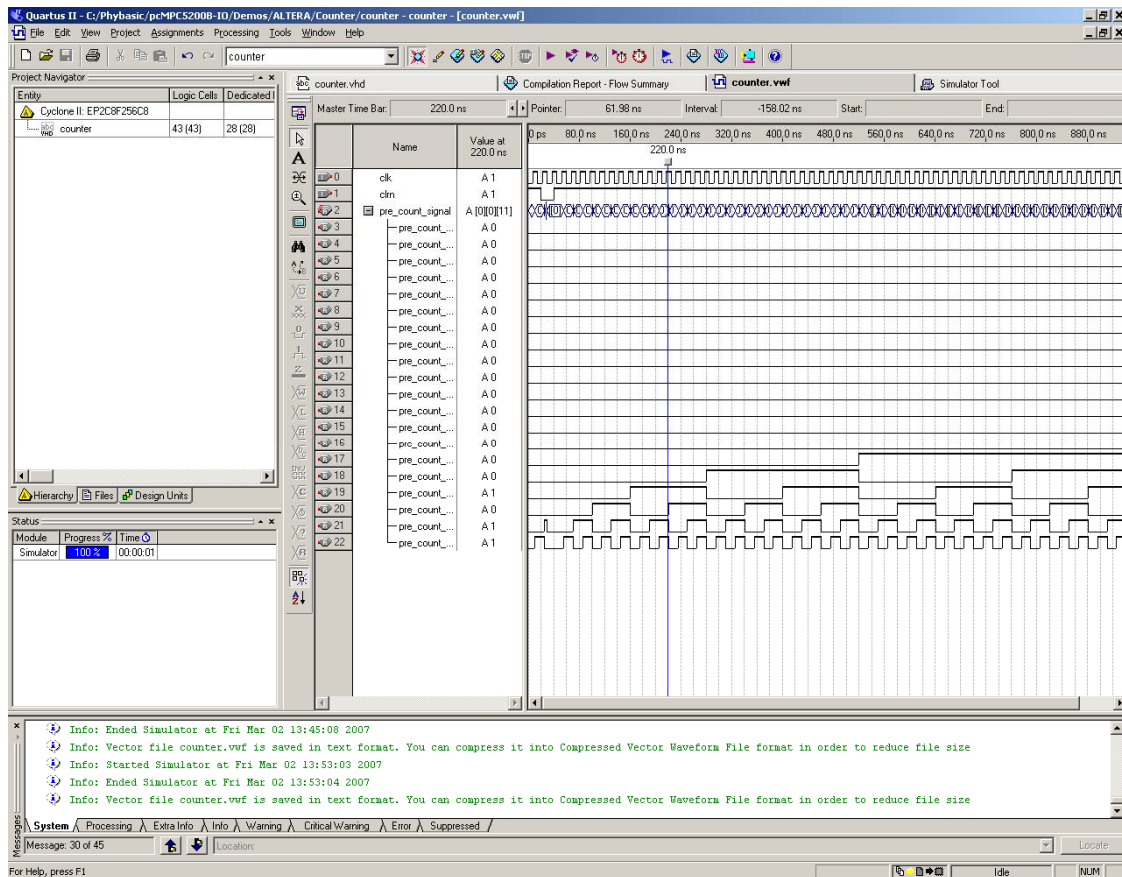


Figure 50: Waveform Editor window with the simulation result of the counter project

The results of your simulation should now correspond to the results shown in the above figure.

You can see that the *pre_count_signal* register is reset to 0 when the *clrn* signal becomes low, and that the pre-counter starts counting again when the *clrn* signal becomes high.



You have successfully created and completed your first Quartus® II V7.2SP1 Web Edition simulation project.

3.2 The Bus System used for addressing the FPGA on the phyCORE-MPC5200B-I/O

3.2.1 The Local Plus Bus of the MPC5200B attached to the FPGA

A Local Plus Bus connects the FPGA of the phyCORE-MPC5200B-I/O to the MPC5200B microprocessor.

The Local Plus Bus interface of the MPC5200B is a multiplexed, asynchronous bus interface. It is neither master nor burst capable.

Because of the bus system used a maximum theoretical data transfer rate of 37 MB/s can be achieved between the FPGA and the microprocessor at a bus clock of 66 MHz.

Numerous control signals are attached between the microprocessor and the FPGA for the data transfer. The following signals are available:

- /LP_CS3
- /LP_CS4
- LP_RDnWR
- /LP_OE
- /LP_ACK
- /LP_ALE
- /LP_TS
- AD0..AD31
- /HReset

The individual bus signals of the MPC5200B have the following functions:

1. The Chip Select signals /LP_CS3 and /LP_CS4

The Chip Select signals are used to select a peripheral component during a bus access on the Local Plus bus. The address area in which the Chip Select signals can be active is configurable from the CPU side.

2. The LP_RDnWR signal

The LP_RDnWR signal serves to control the read and write access to peripheral components of the Local Plus bus. If the signal is high a read access to the peripheral component selected by the Chip Select signal is performed. If the signal is low during the active phase of the Chip Select signal, a write access from the processor to the selected peripheral component is initiated.

3. The /LP_OE signal

Similar to the /LP_RDnWR signal the /LP_OE signal controls the way peripheral components are accessed. If the signal is low, the processor initiates a read access to a peripheral component.

4. The /LP_ACK signal

The /LP_ACK signal serves to terminate the read or write access of the processor. If the signal is pulled low during an access, the processor finishes the read or write access with the next rising edge of the clock signal.

5. The /LP_ALE signal

In a multiplexed bus system the /LP_ALE signal is used to latch the addresses during the address phase of the bus access.

6. The /LP_TS signal

During a bus access the /LP_TS signal indicates the time where the data transfer starts. It becomes active immediately after the address phase to trigger the data phase.

7. The data and address lines AD0..AD31

The data and address lines serve to attach the addresses during the address phase and to transfer the data in the write and read phase.

8. The /HRESET signal

The signal/HReset is a bi-directional reset signal of the MPC5200B microprocessor, which allows to reset the FPGA's internal logic elements to a their initial state. Likewise it is possible to reset the microprocessor with this signal.

The signal sequence during a read or write access on the Local Plus Bus of the MPC5200B is shown in the following figures:

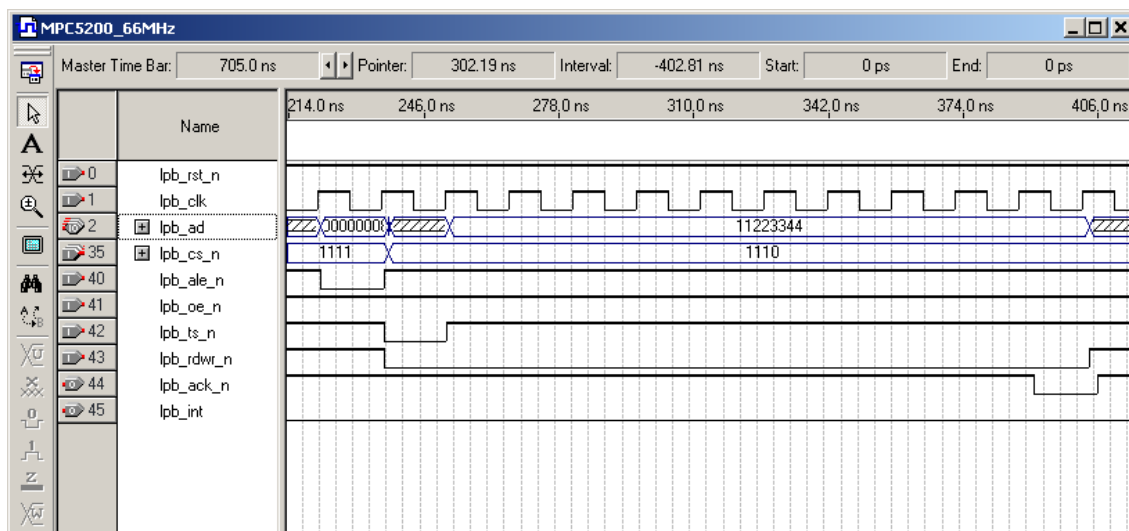


Figure 51: Signal sequence during a read access on the Local Plus Bus of the MPC5200B

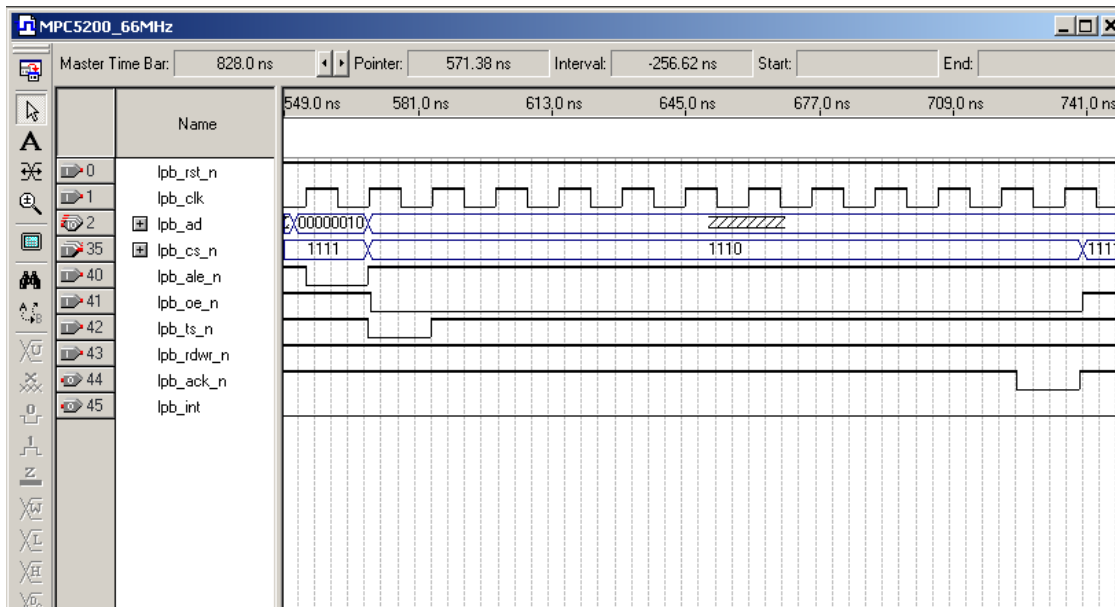


Figure 52: Signal sequence during a write access on the Local Plus Bus of the MPC5200B

Please refer to the corresponding data sheet / user manual for further information on the Local Plus bus.

3.2.2 The Avalon interface

The Avalon interface is a synchronous bus system for FPGA internal data transmission which supports burst and multimaster transfer. This bus system is bound by license to FPGAs from ALTERA. Synchronous Operation allows for a maximum data throughput which can be boosted even more through a flexible frequency selection.

The Avalon interface has the following ports:

3.2.2.1 The Avalon Master Port

| Signal Type | Width | Direction | Required | Description |
|----------------------------|-----------------------------------|-----------|----------|---|
| Fundamental Signals | | | | |
| clk | 1 | In | Yes | Synchronization clock for the Avalon slave interface. All signals are synchronous to clk. |
| waitrequest | 1 | In | Yes | Forces the master port to wait until the Avalon switch fabric is ready to proceed with the transfer. |
| address | 1-32 | Out | Yes | Address lines from the master port to the Avalon switch fabric. The address signal represents a byte address. However, the master port must assert address on word boundaries only. |
| read | 1 | Out | No | Read request signal from master port. Not required if master port never performs read transfers. If used, readdata or data must also be used. |
| readdata | 8,16 ,32, 64, 128 (1) | In | No | Data lines from the Avalon switch fabric for read transfers. Not required if the master port never performs read transfers. If used, read must also be used, and data cannot be used. |
| write | 1 | Out | No | Write request signal from master port. Not required if the master port never performs write transfers. If used, writedata or data must also be used. |
| writedata | 8,16 ,32, 64, 128 (1) | Out | No | Data lines to the Avalon switch fabric for write transfers. Not required if the master port never performs write transfers. If used, write must also be used, and data cannot be used. |
| byteenable | 0,2, 4,8, 16 | Out | No | Byte-enable signals to enable specific byte lane(s) during write transfers to memories of width greater than 8 bits. The master port must assert all byteenable lines during read transfers. |

| Pipeline Signals | | | | |
|----------------------|----------------|---|----|--|
| readdatavalid | 1 | In | No | Used for pipelined read transfers with latency. Indicates that valid data from the Avalon switch fabric is present on the readdata lines. Required if the master is pipelined. |
| flush | 1 | Out | No | Used for pipelined read transfers. The master port asserts flush to clear any pending transfers in the pipeline. |
| Burst Signals | | | | |
| burstcount | 2-32 | Out | No | Used for burst transfers. Indicates the number of transfers in a burst. |
| Flow Control Signals | | | | |
| endofpacket | 1 | In | No | Used for transfers with flow control. Indicates an end-of-packet condition from the Avalon switch fabric. Implementation is peripheral specific. |
| Tristate Signals | | | | |
| data | 8,16,32,64,128 | Bidirectional read and write data for tristate master ports. If used, readdata and writedata cannot be used. | | |
| Other Signals | | | | |
| irq | 1,32 | In | No | Indicates when one or more slave ports have requested an interrupt. If irq is a 32-bit vector, each line corresponds directly to the irq signal on a slave port, with no inherent assumption of priority. If irq is one bit wide, it is the logical OR of all slave irq signals, and the interrupt priority is encoded on irqnumber. |
| irqnumber | 6 | In | No | Indicates the interrupt priority of a slave port asserting its interrupt request. Lower value means higher priority. Used only when the irq signal is one bit wide. |
| reset | 1 | In | No | Global reset signal. Implementation is peripheral specific. |
| resetrequest | 1 | Out | No | Allows the peripheral to reset the entire Avalon system. The result is immediate. |

The following figures show the signal timing of an Avalon master port during read or write access:

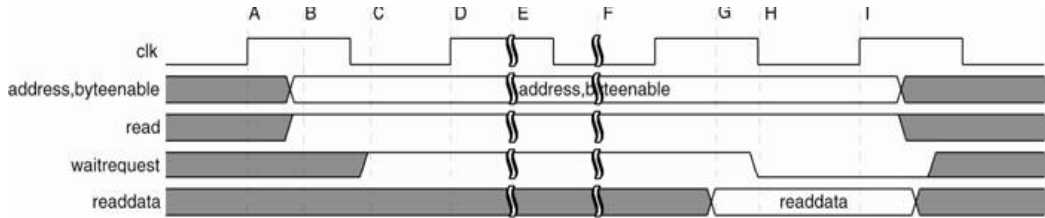


Figure 53: Avalon master port read access with Wait-States

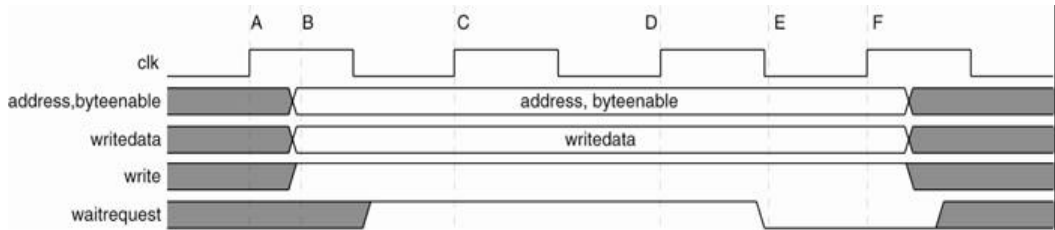


Figure 54: Avalon master port write access with Wait-States

3.2.2.2 The Avalon Slave Port

| Signal Type | Width | Direction | Required | Description |
|----------------------------|---------------------|-----------|----------|--|
| Fundamental Signals | | | | |
| clk | 1 | In | No | Synchronization clock for the Avalon slave interface. All signals are synchronous to clk. Asynchronous slave ports can omit clk. |
| chipselect | 1 | In | No | Chip Select signal to the slave port. The slave port ignores all other Avalon signal inputs unless chip select is asserted. |
| address | 1-32 | In | No | Address lines from the Avalon switch fabric to the slave port. Specifies a word offset into the slave address space. |
| read | 1 | In | No | Read-request signal to the slave port. Not required if the slave port never outputs data. If used, readdata or data must also be used. |
| readdata | 1-128 (1) (2) | Out | No | Data lines to the Avalon switch fabric for read transfers. Not required if the slave port never outputs data. If used, data cannot be used. |
| write | 1 | In | No | Write-request signal to the slave port. Not required if the slave port never receives data from a master. If used, writedata or data must also be used, and writebyteenable cannot be used. |
| writedata | 1-128 (1) (2) | In | No | Data lines from the Avalon switch fabric for write transfers. Not required if the slave port never receives data. If used, write or writebyteenable must also be used, and data cannot be used. |
| byteenable | 0,2, 4,8, 16 | In | No | Byte-enable signals to enable specific byte lane(s) during write transfers to memories of width greater than 8 bits. If used, writedata must also be used, and writebyteenable cannot be used. |
| writebyteenable | 0,2, 4,8, 16 | In | No | Equivalent to the logical AND of the byteenable and write signals. If used, writedata must also be used. write and byteenable cannot be used. |

| | | | | |
|-----------------------------|--------------|----------------|----|--|
| begintransfer | 1 | In | No | Asserted during the first cycle of every transfer. Usage is peripheral-specific. |
| Wait-State Signals | | | | |
| waitrequest | 1 | Out | No | Used to stall the Avalon switch fabric when the slave port is not able to respond immediately. |
| Pipeline Signals | | | | |
| readdatavalid | 1 | Out | No | Used for pipelined read transfers with variable latency. Marks the rising clock edge when the slave asserts valid readdata. |
| Burst Signals | | | | |
| burstcount | 2-32 | In | No | Used for burst transfers. Indicates the number of transfers in a burst. When used, waitrequest must also be used. |
| beginbursttransfer | 1 | In | No | Asserted for the first cycle of a burst to indicate when a burst transfer is starting. Usage is peripheral-specific. |
| Flow Control Signals | | | | |
| readyfordata | 1 | Out | No | Used for transfers with flow control. Indicates that the peripheral is ready for a write transfer. |
| dataavailable | 1 | Out | No | Used for transfers with flow control. Indicates that the peripheral is ready for a read transfer. |
| endofpacket | 1 | Out | No | Used for transfers with flow control. Indicates an end-of-packet condition to the Avalon switch fabric. Implementation is peripheral specific. |
| Tristate Signals | | | | |
| data | 1-128 (1) | Bi-directional | No | Bidirectional read and write data for tristate slave ports. If used, readdata and writedata cannot be used. |
| outputenable | 1 | In | No | Output-enable signal for the data lines. When deasserted, tristate slave port must not drive its data lines. If used, data must also be used. |

| Other Signals | | | | |
|---------------|---|-----|----|--|
| irq | 1 | Out | No | Interrupt request. A slave port asserts irq when it needs to be serviced by a master. |
| reset | 1 | In | No | Peripheral reset signal. When asserted, slave peripheral must enter a deterministic reset state. |
| resetrequest | 1 | Out | No | Allows the peripheral to reset the entire Avalon system. The result is immediate. |

The following figures show the signal timing of an Avalon slave port during read or write access:

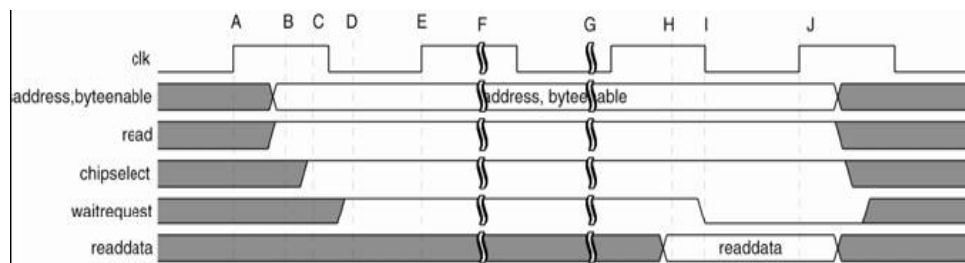


Figure 55: Avalon slave port read access with Wait-States

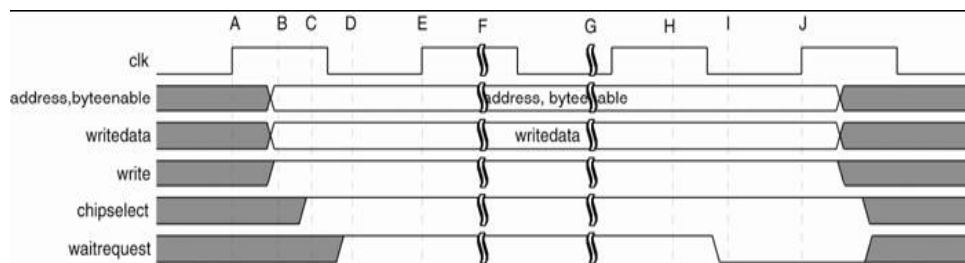


Figure 56: Avalon slave port write access with Wait-States

Please refer to the latest specification for more information on the Avalon interface. To download it go to :

http://www.altera.com/literature/manual/mnl_avalon_spec.pdf

3.2.3 Signal Conversion from the MPC5200B's Local Plus Bus to the FPGA's Avalon interface

A number of FPGA internal processes are necessary to convert the signals from the MPC5200B's Local Plus Bus to the FPGA's Avalon interface and to ensure correct data transmission for reading and writing. This section describes the processes for data conversion and explains some details about the signal processing.



Caution: The Quartus® II V7.2SP1 Web Edition Project introduced in this section is not intended for programming to the FPGA. It merely serves to illustrate the functionality of the data transfer between the MPC5200B's LP bus system and the FPGA's Avalon interface.

If you accepted the default destination location during the installation of the Development Kit CD you can find the source discussed in this section in the following file:

*c:\PHYTEC\PCM-032_phyCORE-MPC5200B-IO\FPGA-Ki\Quickstart
 \Demos\Avalon_Counter\lpb_to_avalon.qpf*

3.2.3.1 The Entity Declaration

The entity declaration specifies the name and the external interface of an VHDL entity. If the VHDL entity is the top level entity in a hierarchical design, the ports of the interface represent the physical ports of the component described.

The following screenshot displays the entity declaration for the signal conversion from the MPC5200B's Local Plus Bus to the FPGA's Avalon interface.

```

5
6 ENTITY lpb_mpc5200b_to_avalon IS
7
8   GENERIC
9   (
10    LPBADDRWIDTH    : integer := 32;
11    LPBDATAWIDTH    : integer := 32;
12    LPBTSIZEWIDTH   : integer := 3;
13    LPBCSWIDTH      : integer := 4;
14    LPBBANKWIDTH    : integer := 2
15  );
16  PORT
17  (
18    -- Avalon Fundametal Signals
19    waitrequest : in std_logic;
20
21    -- Avalon Address/Data Interface
22    address     : out std_logic_vector (31 downto 0);
23
24    read        : out std_logic;
25    readdata    : in std_logic_vector (31 downto 0);
26
27    write       : out std_logic;
28    writedata   : out std_logic_vector (31 downto 0);
29
30    -- Avalon signal (others)
31    byteenable  : out std_logic_vector (3 downto 0);
32
33    -- MPC5200 address/data interface
34    lpb_ad      : in std_logic_vector ((LPBDATAWIDTH-1) downto 0);
35    lpb_ad_o    : out std_logic_vector ((LPBDATAWIDTH-1) downto 0);
36    lpb_ad_en   : out std_logic;
37
38    -- LocalPlus Bus Chip Selects and other signals
39    lpb_cs_n    : in std_logic_vector ((LPBCSWIDTH-1) downto 0);
40    lpb_oe_n    : in std_logic;
41    lpb_ack_n   : out std_logic;
42    lpb_ale_n   : in std_logic;
43    lpb_rdw_n   : in std_logic;
44    lpb_ts_n    : in std_logic;
45
46    -- Local Plus Bus clock signal
47    lpb_clk     : in std_logic;
48
49    -- Processor reset signal
50    lpb_rst_n   : in std_logic;
51
52    -- Interrupt Signal to MPC
53    lpb_int     : out std_logic
54  );
55 END lpb_mpc5200b_to_avalon;
56

```

Figure 57: Entity Declaration

As only part of the signals described in the specification must be used for the basic communication with the Avalon interface, not all signals of the Avalon master port are described in the entity declaration, as you can see in the screenshot. The rest of the signals can optionally be used for the communication whereas the use depends heavily on the application.

3.2.3.2 Signal Declaration

For the processing of process-internal signals some variables which store different logical states must be declared. In a VHDL design this is done after the keyword ARCHITECTUR. The following figure shows the signal declarations which are necessary for the present project.

```

62
63  ARCHITECTURE avalon_master OF lpb_mpc5200b_to_avalon IS
64
65  SIGNAL lpb_adr_q      : STD_LOGIC_VECTOR((LPBADDRWIDTH-1) DOWNTO 0);
66  SIGNAL lpb_data_q    : STD_LOGIC_VECTOR((LPBDATAWIDTH-1) DOWNTO 0);
67  SIGNAL lpb_tsize_q   : STD_LOGIC_VECTOR ((LPBTISIZEWIDTH-1) DOWNTO 0);
68
69  SIGNAL lpb_data_en   : STD_LOGIC;
70  SIGNAL lpb_start    : STD_LOGIC;
71
72  SIGNAL lpb_rd       : STD_LOGIC;
73  SIGNAL lpb_wr       : STD_LOGIC;
74  SIGNAL lpb_ack_i    : STD_LOGIC;
75
76  SIGNAL lpb_start_en : STD_LOGIC;
77
78  SIGNAL readdata_q   : STD_LOGIC_VECTOR (31 DOWNTO 0);
79
80  type state IS (init, act, rst);
81  SIGNAL avalonstate : state;
82
83  BEGIN
84  --activation of FPGA with only one chip select SIGNAL
85  lpb_rd <= (NOT lpb_cs_n(0) AND NOT lpb_oe_n);
86  lpb_wr <= (NOT lpb_cs_n(0) AND NOT lpb_rdw_r_n);
87
88  -- no interrupt function implemented
89  lpb_int <= '0';
90
91  -- external ack SIGNAL gets internal value
92  lpb_ack_n <= NOT lpb_ack_i;
93
94

```

Figure 58: Signal declaration of the *LPB_MPC5200B_TO_AVALON* project

The signals declared have the following functions:

- lpb_adr_q: Register to store the address during a read or write access of the MPC5200B
- lpb_data_q: Register to store the data during a write access of the MPC5200B to the FPGA
- lpb_tsize_q: Register to store the type of bus access from the MPC5200B to the FPGA
- lpb_data_en: Data Enable signal to read the data from the MPC5200B into the FPGA
- lpb_start: Start signal for the FGPA internal State Machine to process the Avalon bus access
- lpb_rd: LP bus access signal during a read cycle
- lpb_wr: LP bus access signal during a write cycle
- lpb_ack_i: internal acknowledge signal to generate the termination of the LP bus access

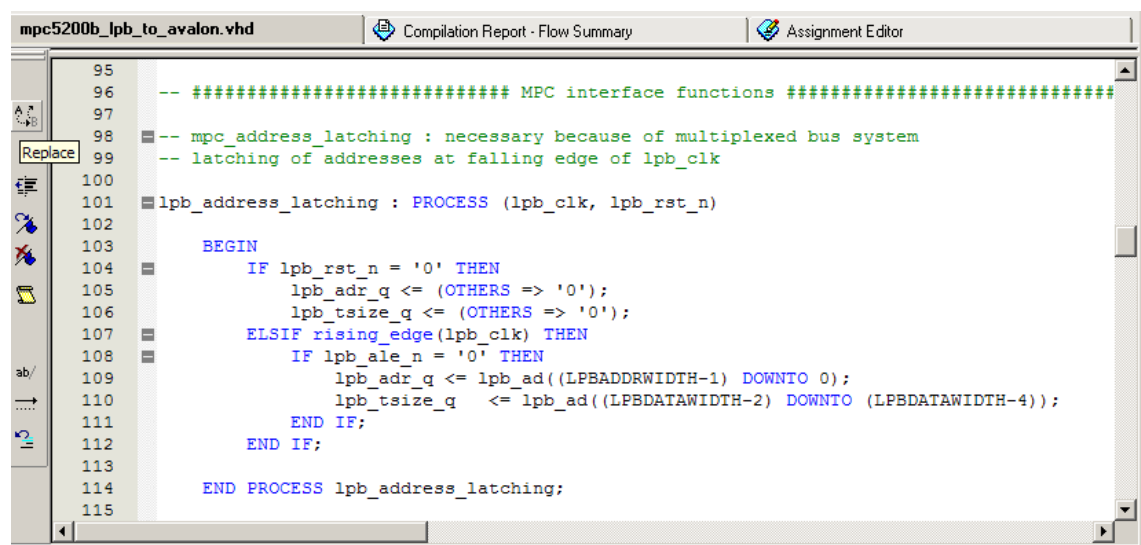
| | |
|---------------|--|
| lpb_start_en: | start enable signal for locking/unlocking the lpb_start signal at a 33MHz bus access |
| readdata_q: | register for latching the Avalon read data |

The signals described have been declared with the suitable data types for the processing. Signals which are solely used within the processes are not described as this would be beyond the scope of this QuickStart Instructions.

3.2.3.3 The address latching process of the LocalPlus bus

As a result of connecting the MPC5200B with the FPGA by use of a multiplexed interface to save I/O pins of the FPGA, a process to read and store the address value is necessary. This process must load the address value into the FPGA during the address phase and provide it for further processing. In the present project this is accomplished by the process *lpb_address_latching*.

The process reads the address value from the data/address bus as well as the access type of the current access (byte, word or double word access) while the address latch enable (ALE) is active and saves this information in appropriate registers for further processing. The following screenshot shows the listing of that process.



```

95
96  -- ##### MPC interface functions #####
97
98  -- mpc_address_latching : necessary because of multiplexed bus system
99  -- latching of addresses at falling edge of lpb_clk
100
101  lpb_address_latching : PROCESS (lpb_clk, lpb_rst_n)
102
103      BEGIN
104          IF lpb_rst_n = '0' THEN
105              lpb_adr_q <= (OTHERS => '0');
106              lpb_tsize_q <= (OTHERS => '0');
107          ELSIF rising_edge(lpb_clk) THEN
108              IF lpb_ale_n = '0' THEN
109                  lpb_adr_q <= lpb_ad((LPBADDRWIDTH-1) DOWNT0 0);
110                  lpb_tsize_q <= lpb_ad((LPBDATAWIDTH-2) DOWNT0 (LPBDATAWIDTH-4));
111              END IF;
112          END IF;
113
114      END PROCESS lpb_address_latching;
115

```

Figure 59: Listing of the process *lpb_address_latching*

3.2.3.4 The process for latching data from the LocalPlus bus

Reading the data transferred from the MPC5200B to the FPGA is accomplished by the process *lpb_write_data_latching*. The process is not only responsible for reading the data from the data/address bus but also for starting the State Machine to access the Avalon bus independently of the access method (read or write) of the LP bus. The following screenshot shows the listing of the *lpb_write_data_latching* process.

```

115
116 -- lpb_write_data_latching
117 -- latching of data of the lpb bus at write cycle
118
119 lpb_write_data_latching : PROCESS (lpb_clk, lpb_rst_n)
120
121 BEGIN
122     IF lpb_rst_n = '0' THEN
123         lpb_data_q <= (OTHERS => '0');
124         lpb_data_en <= '0';
125         lpb_start <= '0';
126     ELSE
127         IF rising_edge (lpb_clk) THEN
128             IF lpb_ts_n = '0' AND lpb_start = '0' THEN
129                 --lpb_start <= '1'; -- for 66MHz we can start here
130                 lpb_start_en <= '1';
131             ELSE
132                 --lpb_start <= '0';
133                 lpb_start_en <= '0';
134             END IF;
135
136             -- needable for 33MHz support, for 66MHz we can start earlier
137             IF lpb_start_en = '1' THEN
138                 lpb_start <= '1';
139             ELSE
140                 lpb_start <= '0';
141             END IF;
142
143             IF lpb_ts_n = '0' AND lpb_rdwr_n = '0' THEN
144                 lpb_data_en <= '1'; -- wait 1 clock cycle for data ready
145             END IF;
146
147             IF lpb_data_en = '1' THEN
148                 lpb_data_q <= lpb_ad;
149                 lpb_data_en <= '0';
150             END IF;
151         END IF;
152     END IF;
153 END PROCESS lpb_write_data_latching;
154

```

Figure 60: Listing of the process *lpb_write_data_latching*

The following part of the listing is important to notice:

```

if lpb_ts_n = '0' and lpb_rdwr_n = '0' then
    lpb_data_en <= '1'; -- wait 1 clock cycle for data ready
end if;

if lpb_data_en = '1' then
    lpb_data_q <= lpb_ad;
    lpb_data_en <= '0';
end if;

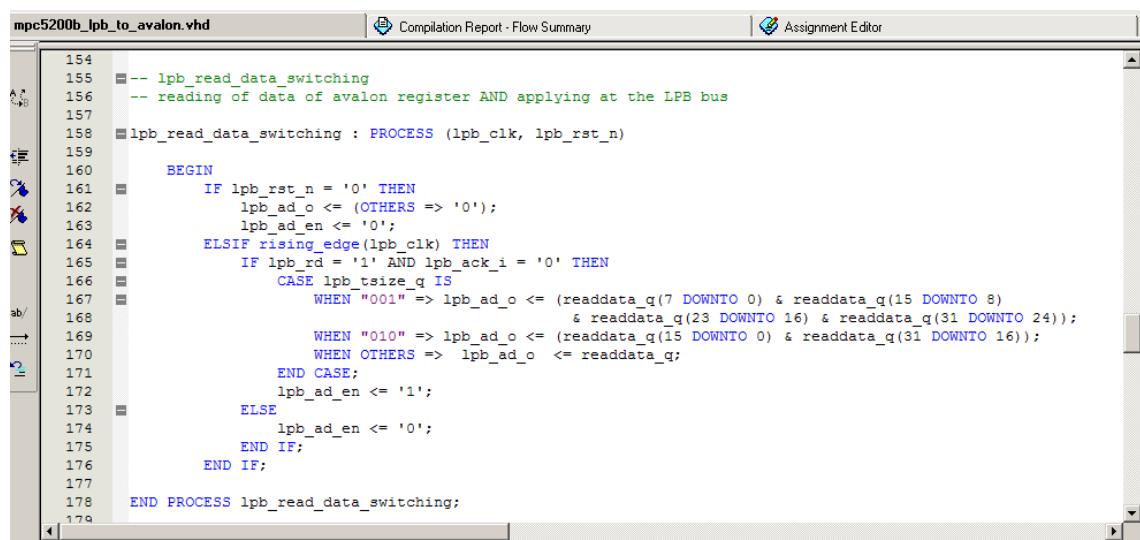
```

As already stated in the listing assigning a "1" to the *lpb_data_en* signal and a parallel query of this signal results in a latency time of one clock cycle before the data of the data bus is transferred to the register *lpb_data_q*. This is necessary because of the MPC5200B's timing when writing the data to the Local Plus Bus. If this wait cycle is not inserted, the data in the register *lpb_data_q* might be inconsistent.

3.2.3.5 The process for writing data to the LocalPlus bus

The process *lpb_read_data_switching* transfers data from the FPGA's Avalon bus to the LP bus of the MPC5200B. The process rearranges the data read from the Avalon bus to adapt it to the Local Plus bus of the processor. At the same time it controls the writing of data to the Local Plus bus with Tristate buffers, which are necessary in a superior instance to connect the two data buses *lpb_ad_o* and *lpb_ad*.

The following screenshot shows the listing of the *lpb_read_data_switching* process.



```

154
155 -- lpb_read_data_switching
156 -- reading of data of avalon register AND applying at the LPB bus
157
158 lpb_read_data_switching : PROCESS (lpb_clk, lpb_rst_n)
159
160 BEGIN
161 IF lpb_rst_n = '0' THEN
162     lpb_ad_o <= (OTHERS => '0');
163     lpb_ad_en <= '0';
164 ELSIF rising_edge(lpb_clk) THEN
165     IF lpb_rd = '1' AND lpb_ack_i = '0' THEN
166         CASE lpb_tsize_q IS
167             WHEN "001" => lpb_ad_o <= (readdata_q(7 DOWNTO 0) & readdata_q(15 DOWNTO 8)
168                                     & readdata_q(23 DOWNTO 16) & readdata_q(31 DOWNTO 24));
169             WHEN "010" => lpb_ad_o <= (readdata_q(15 DOWNTO 0) & readdata_q(31 DOWNTO 16));
170             WHEN OTHERS => lpb_ad_o <= readdata_q;
171         END CASE;
172         lpb_ad_en <= '1';
173     ELSE
174         lpb_ad_en <= '0';
175     END IF;
176 END IF;
177
178 END PROCESS lpb_read_data_switching;
179

```

Figure 61: Listing of the process *lpb_read_data_switching*

The process *lpb_read_data_switching* has the following special implementations:

```
case lpb_tsize_q is
  when "001" => lpb_ad_o <= ( readdata_q(7 downto 0) &
                             readdata_q(15 downto 8) &
                             readdata_q(23 downto 16) &
                             readdata_q(31 downto 24));

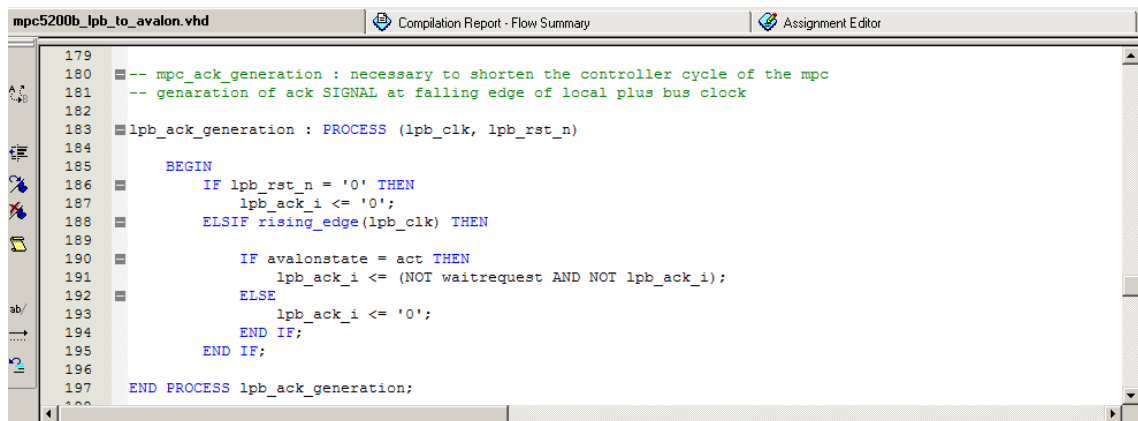
  when "010" => lpb_ad_o <= ( readdata_q(15 downto 0) &
                             readdata_q(31 downto 16));

  when others =>    lpb_ad_o <= readdata_q;
end case;
```

The listing shows a case command which is used to analyze the *lpb_tsize_q* signal. This signal contains the access type of the current read or write access (byte, word or double word access) and provides information on how the data must be written to the LP bus of the MPC5200B for correct data translation. The reason is the internal structure of the MPC5200B which actually performs a Big Endian access to peripheral components, meaning that data is applied to the peripheral bus in reverse order. Since the structure of many IP cores available for FPGAs or ASICs is based on a Little Endian system a reversal of the bus structure is inevitable.

3.2.3.6 The process for generating an acknowledge signal

The MPC5200B provides the *lpb_ack_n* signal, which allows to generate an acknowledge and therefore to shorten the bus access. This is accomplished by the process *lpb_ack_generation*. Activating the *lpb_ack_n* signal ensures the termination of the access to the LP bus even if previously configured Wait States are not expired.



```
179
180 -- mpc_ack_generation : necessary to shorten the controller cycle of the mpc
181 -- generation of ack SIGNAL at falling edge of local plus bus clock
182
183 lpb_ack_generation : PROCESS (lpb_clk, lpb_rst_n)
184
185 BEGIN
186 IF lpb_rst_n = '0' THEN
187   lpb_ack_i <= '0';
188 ELSIF rising_edge(lpb_clk) THEN
189
190   IF avalonstate = act THEN
191     lpb_ack_i <= (NOT waitrequest AND NOT lpb_ack_i);
192   ELSE
193     lpb_ack_i <= '0';
194   END IF;
195 END IF;
196
197 END PROCESS lpb_ack_generation;
```

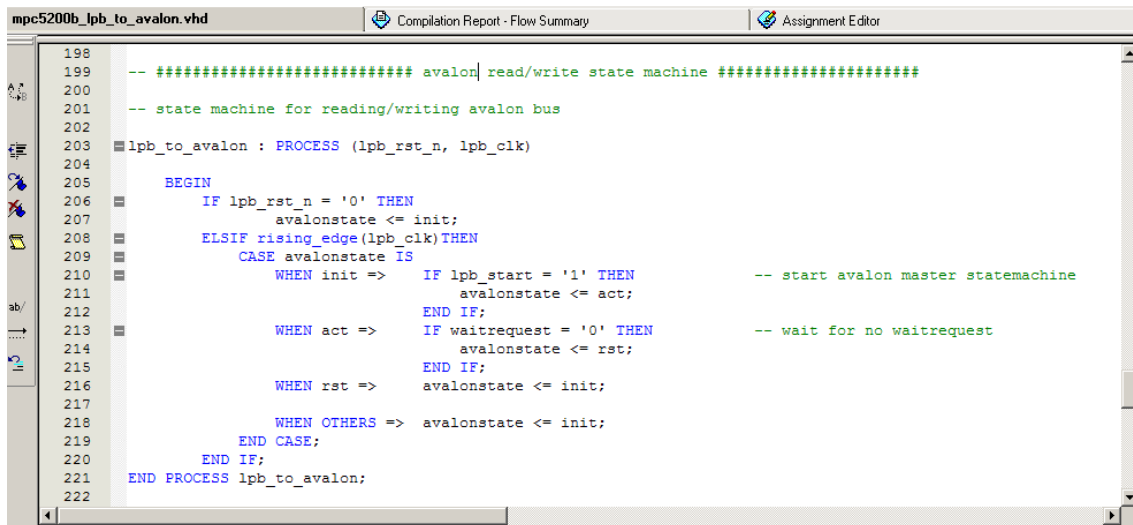
Figure 62: Listing of the process `lpb_ack_generation`

```
if avalonstate = act then
    lpb_ack_i <= (not waitrequest and not lpb_ack_i);
else
    lpb_ack_i <= '0';
end if;
```

As you can see in the above listing, the bus access of the MPC5200B is only terminated when a wait signal is no longer present from the Avalon bus system. The corresponding combinational logic guarantees that the acknowledge signal is only applied to the processor if the data is valid and can be imported.

3.2.3.7 The Avalon State machine

The *lpb_to_avalon* process embodies the state machine for read and write access to the Avalon bus. The state machine is the fundamental process for access to the Avalon bus, and thus accommodates the basic functionality of the complete system. Although the *lpb_to_avalon* process consist of only little code, the complete control of the access to the Avalon bus is implemented therein. The status of the state machine also affects the execution of tasks within other processes.



```
198
199 -- ##### avalon| read/write state machine #####
200
201 -- state machine for reading/writing avalon bus
202
203 lpb_to_avalon : PROCESS (lpb_rst_n, lpb_clk)
204
205 BEGIN
206     IF lpb_rst_n = '0' THEN
207         avalonstate <= init;
208     ELSIF rising_edge(lpb_clk) THEN
209         CASE avalonstate IS
210             WHEN init =>     IF lpb_start = '1' THEN          -- start avalon master statemachine
211                             avalonstate <= act;
212                             END IF;
213             WHEN act =>     IF waitrequest = '0' THEN        -- wait for no waitrequest
214                             avalonstate <= rst;
215                             END IF;
216             WHEN rst =>     avalonstate <= init;
217
218             WHEN OTHERS =>  avalonstate <= init;
219         END CASE;
220     END IF;
221 END PROCESS lpb_to_avalon;
222
```

Figure 63: Listing of the process *lpb_to_avalon* – Main State Machine

The source code of the state machine has been intentionally kept very clear so that it is very easy to comprehend the basic functionality.

The state machine has the following states:

- 1st state: State machine is idle. The state machine starts only if the *lpb_start* signal becomes active (e.g. logical 1).
- 2nd state: State machine is active. The state machine stops only when the *waitrequest* signal of the Avalon bus system is set to a logical 0.
- 3rd state: State machine is returned to original state. The state machine returns to its original state 1.

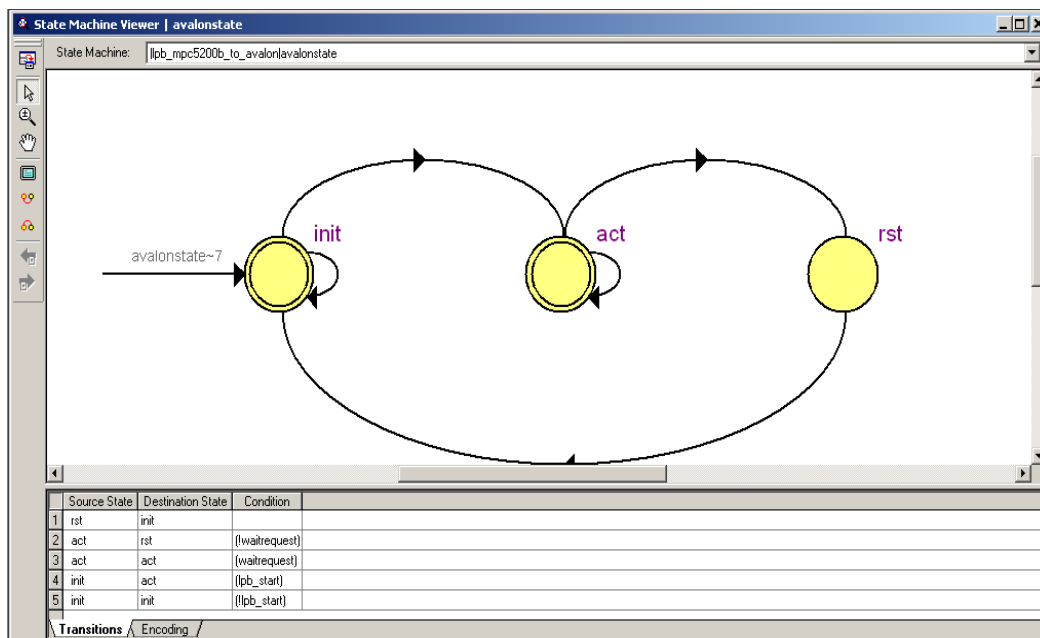


Figure 64: Illustration of the state machine embodied in the *lpb_to_avalon* process

3.2.3.8 The process for controlling the Avalon bus signals

The *avalon_bus* process serves to control all Avalon bus signals which are necessary for data transfer between the LP bus of the MPC5200B and the Avalon bus. It handles both directions, reading and writing. Similar to the *lpb_read_data_switching* process, the *avalon_bus* process evaluates the bus access type of the MPC5200B with a case command when writing and

arranges the bytes as required. In addition, the control signals for the Avalon bus are generated from the control signals of the MPC5200B.

The following screenshot shows the listing of the *avalon_bus* process.

```

223  ┌─ avalon_bus : PROCESS (lpb_rst_n, lpb_clk)
224
225  ┌─ BEGIN
226  ┌─     IF lpb_rst_n = '0' THEN
227
228  ┌─     ELSIF rising_edge(lpb_clk) THEN
229  ┌─         IF avalonstate = init AND lpb_start = '1' THEN
230  ┌─             address <= lpb_adr_q;
231  ┌─             write <= lpb_wr;           -- avalon SIGNAL generation we
232  ┌─             read <= lpb_rd;
233
234  ┌─             CASE lpb_tsize_q IS           -- swap bytes for little endian access
235  ┌─                 WHEN "100" => byteenable <= "1111";
236  ┌─                 writedata <= lpb_data_q;
237  ┌─                 WHEN "010" => CASE lpb_adr_q(1 DOWNTO 0) IS
238  ┌─                     WHEN "00" => byteenable <= "0011";
239  ┌─                     writedata(15 DOWNTO 0) <= lpb_data_q(31 DOWNTO 16);
240  ┌─                     WHEN "10" => byteenable <= "1100";
241  ┌─                     writedata <= lpb_data_q;
242  ┌─                     WHEN OTHERS => byteenable <= "1111";
243  ┌─                     writedata <= lpb_data_q;
244
245  ┌─                     END CASE;
246  ┌─                 WHEN "001" => CASE lpb_adr_q(1 DOWNTO 0) IS
247  ┌─                     WHEN "00" => byteenable <= "0001";
248  ┌─                     writedata(7 DOWNTO 0) <= lpb_data_q(31 DOWNTO 24);
249  ┌─                     WHEN "01" => byteenable <= "0010";
250  ┌─                     writedata(15 DOWNTO 8) <= lpb_data_q(31 DOWNTO 24);
251  ┌─                     WHEN "10" => byteenable <= "0100";
252  ┌─                     writedata(23 DOWNTO 16) <= lpb_data_q(31 DOWNTO 24);
253  ┌─                     WHEN "11" => byteenable <= "1000";
254  ┌─                     writedata <= lpb_data_q;
255
256  ┌─                     END CASE;
257  ┌─                 WHEN OTHERS => byteenable <= "1111";
258  ┌─                 writedata <= lpb_data_q;
259
260  ┌─             END CASE;
261  ┌─         END IF;
262  ┌─     IF avalonstate = act THEN
263  ┌─         readdata_q <= readdata;
264  ┌─         IF waitrequest = '0' THEN
265  ┌─             read <= '0';
266  ┌─             write <= '0';
267  ┌─             address <= (OTHERS => '0');
268  ┌─             writedata <= (OTHERS => '0');
269
270  ┌─         END IF;
271  ┌─     END IF;
272  ┌─ END PROCESS avalon_bus;

```

Figure 65: Listing of the process *avalon_bus*

The following listing again explicitly shows the rearrangement of the bytes which is necessary to adapt the MPC5200B's LP bus to the Avalon bus as described above.


```

case lpb_tsize_q is
    when "100" => byteenable <= "1111";
                  writedata <= lpb_data_q;
    when "010" => case lpb_adr_q(1 downto 0) is
                    when "00" => byteenable <= "0011";
                                writedata(15 downto 0) <= lpb_data_q(31 downto 16);
                    when "10" => byteenable <= "1100";
                                writedata <= lpb_data_q;
                    when others=> byteenable <= "1111";
                                writedata <= lpb_data_q;
                  end case;
    when "001" => case lpb_adr_q(1 downto 0) is
                    when "00" => byteenable <= "0001";
                                writedata(7 downto 0) <= lpb_data_q(31 downto 24);
                    when "01" => byteenable <= "0010";
                                writedata(15 downto 8) <= lpb_data_q(31 downto 24);
                    when "10" => byteenable <= "0100";
                                writedata(23 downto 16) <= lpb_data_q(31 downto 24);
                    when "11" => byteenable <= "1000";
                                writedata <= lpb_data_q;
                  end case;
    when others => byteenable <= "1111";
                  writedata <= lpb_data_q;
end case;

```

3.2.4 Adding an Avalon slave interface to the counter project

In this section an Avalon interface will be added to the *counter* project created in section 3.1.1 to provide an example of the development of an Avalon slave interface. The counter will be expanded with several processes which implement the functionality of the Avalon slave interface. Following creation, it will be possible to start the counter from the MPC5200B microcontroller, to write and read various prefetch registers for loading the counter registers, and to read the counter registers. An output to control an LED is additionally incorporated into the project in order to verify the counter functionality.

To allow easy use of this example, preliminary work has already been carried out in the VHDL files to be added to the project. These files contain the logic required to include the Avalon counter VHDL file in its modified version and to compile the project in an executable logic design.

The VHDL files contain the following functions:

| | |
|-----------------------------|--|
| mpc5200b_lpb_to_avalon.vhd: | Conversion of the LocalPlus bus interface of the MPC5200B to an Avalon master interface |
| lpb_avalon_counter.vhd: | Top level design to connect the two files mpc5200b_lpb_to_avalon.vhd and counter.vhd |
| pll.vhd: | VHDL file provided by ALTERA for creating a PLL to supply a logic design with a clock signal |

The VHDL files are already present in the specified project folder, and need only be added when creating the project.

3.2.4.1 Creating the Avalon-Counter project

In order to add the Avalon interface to the *counter* project it is advisable to create a new project rather than changing the counter project discussed in section 3.1.1.

- First copy the VHDL file *counter.vhd* from the counter project discussed in section 3.1.1 into the working directory of the new project. The new working directory *C:\PHYTEC\PCM-032_phyCORE-MPC5200B-IO\FPGA-Kit\Quickstart\Demos\Avalon_Counter* has been created on your hard disk by the installation process and already contains some files.
- Now use the *New Project Wizard* as described in section 3.1.1 to create a new project.
- As the working directory please enter *C:\PHYTEC\PCM-032_phyCORE-MPC5200B-IO\FPGA-Kit\Quickstart\Demos\Avalon_Counter*.

- For the project's name and the top-level design entity please enter *lpb_avalon_counter* in the appropriate fields of the *NewProject Wizard* window as shown below. Click the *Next* button to confirm the entries.

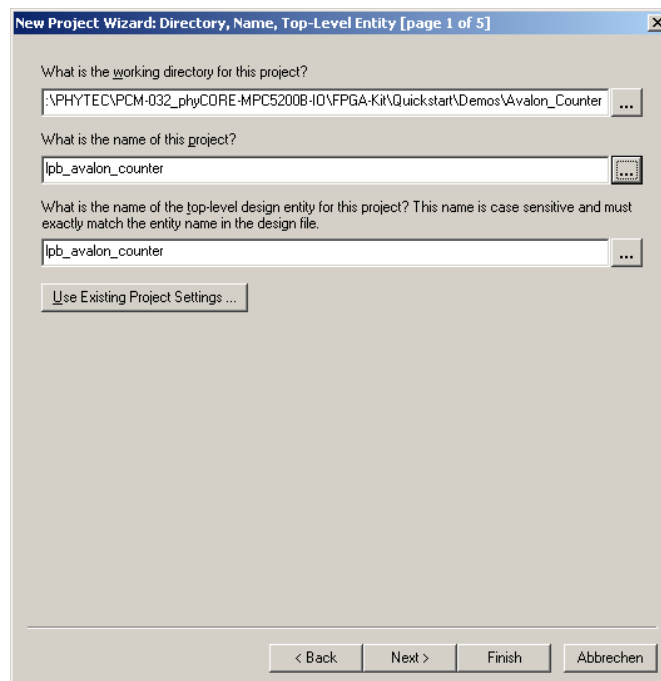


Figure 66: Working directory and project name of the *Avalon-Counter* project

- Add all four VHDL-files which are in the working directory to the new project.

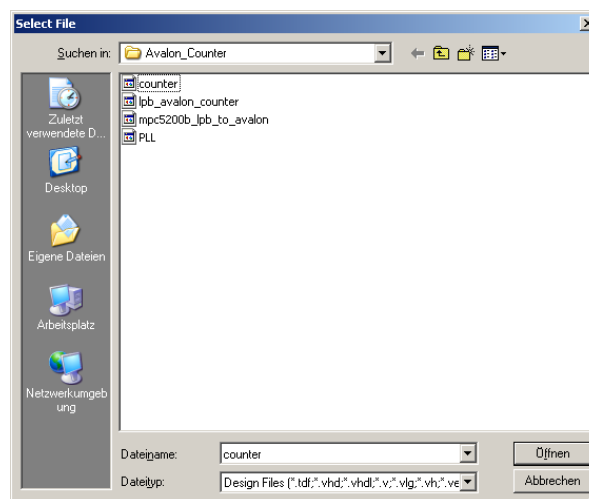


Figure 67: The VHDL-files which belong to the *Avalon-Counter* project

- Continue to configure the project settings as described in section 3.1.1.
- If all project settings are done properly the summary of the project settings should look the same as in the figure below. Click on the *Finish* button to finalize the project configuration.

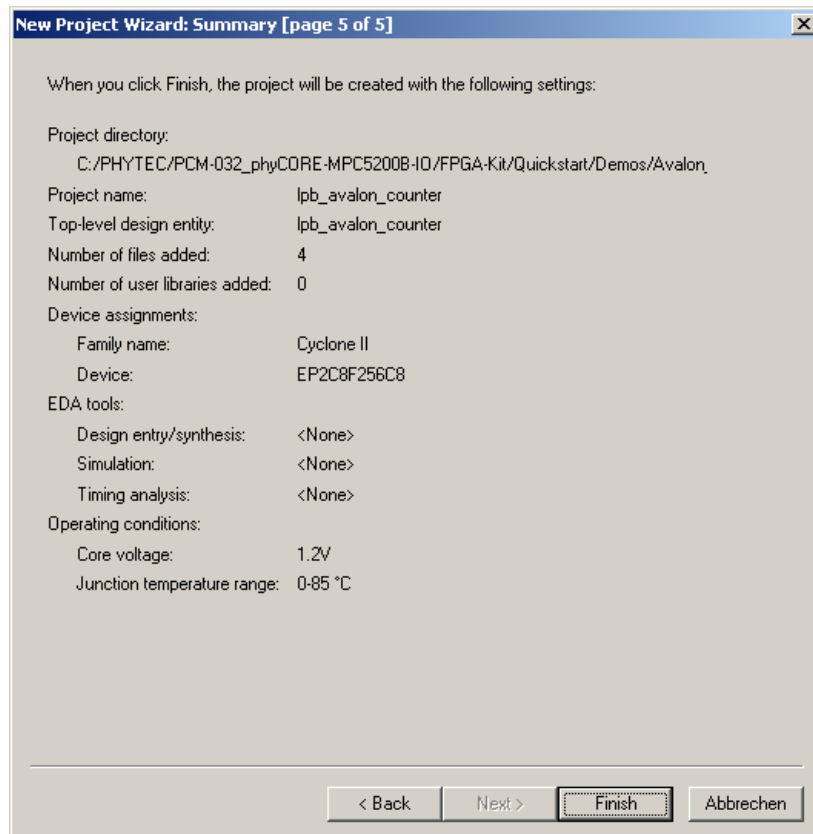



Figure 68: Summary of the project settings

- Now click on the icon *Settings*  to enter additional project settings or to check newly incorporated project settings.
- Carry out the configuration for the unused pins as described in section 3.1.1.

In the following sections the changes which have to be made within the file *counter.vhd* will be explained step-by-step.

- Choose *Open* from the *File* menu to edit the file *counter.vhd*

3.2.4.2 The newly designed entity of the *counter.vhd* file

Because of the new bus structure which must be added to the counter project for reading and writing the various newly added registers, the port structure of the entity must be changed first. In order to be able to implement an Avalon slave interface, the following signals must be inserted into the port structure:

- s_address
- s_chipselect
- s_read
- s_readdata
- s_write
- s_writedata_s_byteenables_waitrequest
- s_irq
- s_out

Please refer to section 3.2.2 for a detailed description of these signals.

- Change the name of the entity counter into *avalon_counter*.
- Now change the port declaration according to the following listing:

```

ENTITY avalon_counter IS
  PORT
  (
    clk:          IN    STD_LOGIC;
    clrn:         IN    STD_LOGIC;

    --Avalon slave interface signals
    s_address:   IN    STD_LOGIC_VECTOR(31 DOWNTO 0);
    s_chipselect: IN    STD_LOGIC;
    s_read:      IN    STD_LOGIC;
    s_readdata : OUT   STD_LOGIC_VECTOR (31 DOWNTO 0);
    s_write:     IN    STD_LOGIC;
    s_writedata: IN    STD_LOGIC_VECTOR (31 DOWNTO 0);
    s_byteenable: IN   STD_LOGIC_VECTOR (3 DOWNTO 0);
    s_waitrequest: OUT  STD_LOGIC;
    s_irq:       OUT  STD_LOGIC;
    s_out:      OUT  STD_LOGIC
  );
END avalon_counter;

```

3.2.4.3 The newly defined signals and constants of the *counter.vhd* file

In order to be able to implement the desired functionality of the Avalon-Counter project, several signals and constants must be newly declared in the *counter.vhd* file. These are necessary in order to generate the required registers and the internal control signals during the synthesis.

The following constants must be declared:

```
CONSTANT PRE_COUNTER_OVERFLOW :UNSIGNED:= x"FFFFFFFF";
CONSTANT COUNTER_OVERFLOW    :UNSIGNED := x"FF";
```

The two constants are used for the overflow detection of the two counter registers *count_signal* and *pre_count_signal*.

To create the desired counter, control and load registers, they must be added as internal signals. This is achieved using the following signal declarations:

```
SIGNAL count_reg0:      STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL count_reg1:      STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL count_ctl:       STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL load_reg0:       STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL load_reg1:       STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL waitrequest_n:   STD_LOGIC;
SIGNAL count_ctl_q:     STD_LOGIC;
SIGNAL count_signal:    UNSIGNED(7 DOWNTO 0);
SIGNAL pre_count_signal: UNSIGNED(31 DOWNTO 0);
```

In order to implement an interrupt output and to visualize the function of the counter with the LED the internal signals *local_irq* and *led_state* must be added. This signals allow reading and storing of the irq- and led-output's actual state:

```
SIGNAL led_state:      STD_LOGIC;
SIGNAL led_state:      STD_LOGIC;
```

For the generation and recognition of the state machine by the Quartus® II tool chain a special type declaration must be implemented. This is carried out using the following declaration:

```
type state is (init, act_wait, act, rst);
signal avalonstate : state;
```

- To proceed with the modifications in the *counter.vhd* file change the name of the entity in the architecture body from *count* into *avalon_counter*
- Now add the constant, signal and type declarations according to the following listing:

```

ARCHITECTURE rtl OF avalon_counter IS

    CONSTANT PRE_COUNTER_OVERROLL: UNSIGNED := x"FFFFFFFF";
    CONSTANT COUNTER_OVERROLL:      UNSIGNED := x"FF";

    -- counter register for reading register count_signal
    SIGNAL count_reg0:    STD_LOGIC_VECTOR(7 DOWNTO 0);
    -- counter register for reading register pre_count_signal
    SIGNAL count_reg1:    STD_LOGIC_VECTOR(31 DOWNTO 0);
    -- counter control register
    SIGNAL count_ctl:     STD_LOGIC_VECTOR(31 DOWNTO 0);
    -- load register for counter register
    SIGNAL load_reg0:     STD_LOGIC_VECTOR(31 DOWNTO 0);
    -- load register for pre counter register
    SIGNAL load_reg1:     STD_LOGIC_VECTOR(31 DOWNTO 0);
    -- wait signal for waitrequest generation
    SIGNAL waitrequest_n: STD_LOGIC;
    -- counter start signal
    SIGNAL count_ctl_q:   STD_LOGIC;
    -- main counter register
    SIGNAL count_signal:  UNSIGNED(7 DOWNTO 0);

    -- pre counter register
    SIGNAL pre_count_signal: UNSIGNED(31 DOWNTO 0);

    SIGNAL local_irq:     STD_LOGIC;

    -- led status signal
    SIGNAL led_state:    STD_LOGIC;

    -- state machine variable
    type state is (init, act_wait, act, rst);
    signal avalonstate: state;

BEGIN

```

However, before the implementation of the processes can be started, it is still necessary to carry out fundamental signal assignments.

First a signal assignment of the *count_signal* or *pre_count_signal* register to the registers *count_reg0* or *count_reg1* with a type conversion into a *STD_LOGIC_VECTOR* is necessary in order to ensure correct reading of the counter registers.

Second, a signal assignment to the *s_waitrequest* signal is used in order to simply generate the *s_waitrequest* signal, required for the Avalon slave interface, as soon as an active chip select signal is present together with the corresponding read or write signal.

And finally the internal variable which stores the state of the interrupt output must be assigned to the actual interrupt signal.

The following additional assignments must be carried out.

```
-- readable counter registers
```

```
count_reg0 <= STD_LOGIC_VECTOR(count_signal);
```

```
count_reg1 <= STD_LOGIC_VECTOR(pre_count_signal);
```

```
-- wait request signal generation
```

```
s_waitrequest <= ((s_chipselect AND ( s_write OR s_read)) AND  
                waitrequest_n);
```

```
-- interrupt signal generation
```

```
s_irq <= local_irq;
```

```
-- led signal generation
```

```
s_out <= led_state;
```

- Add the signal assignments between the *BEGIN* statement and the start of the actual process as shown in the listing below.


```
BEGIN
-- readable counter registers
count_reg0 <= STD_LOGIC_VECTOR(count_signal);
count_reg1 <= STD_LOGIC_VECTOR(pre_count_signal);

-- wait request signal generation
s_waitrequest <= ((s_chipselect AND ( s_write OR s_read)) AND waitrequest_n);

-- interrupt signal generation
s_irq <= local_irq;

-- led signal generation
s_out <= led_state;

count: PROCESS (clk, clrn)
```

3.2.4.4 The *count* process

In order to guarantee a correct function of the Avalon slave the following functions are added to the count process:

1. A new feature added to the count process is the option to load the count registers with a predefined value in order to change the time interval between the interrupt signals. This feature is accomplished with two load registers *load_reg0* and *load_reg1* whose content is loaded into the corresponding counter registers in the event of a register overflow.
2. An additional *count_ctl_q* signal is added to the load registers to control the loading of the counter registers. This signal equals the start signal of the counter delayed by one clock cycle. The *count_ctl_q* signal has also been introduced to allow easy generation of an interrupt signal in case of a counter overflow.
3. The interrupt function is the last new function added to the count process. If the interrupt is enabled (e.g. bit1 of the control register is set) a short interrupt signal is generated when an overflow of the counter occurs. This signal can be used to visualize the functioning of the counter. In this project the LED is attached to this signal and the variation in the flash rate visualizes the counter running with different load values.

- Add the new functions to the count process as shown in the listing below to implement the functionality described above.

```

count: PROCESS (clk, clrn)
BEGIN
  IF clrn = '0' THEN
    count_signal      <= (OTHERS => '0');
    pre_count_signal  <= (OTHERS => '0');
    local_irq         <= '0';
    count_ctl_q       <= '0';
  ELSIF (clk'EVENT AND clk = '1') THEN
    count_ctl_q <= count_ctl(0);

    -- counter function
    IF ((count_ctl(0) = '1') AND (count_ctl_q = '1')) THEN
      pre_count_signal <= pre_count_signal + 1;
      IF pre_count_signal = PRE_COUNTER_OVERROLL THEN
        count_signal <= count_signal + 1;
        pre_count_signal <= UNSIGNED(load_reg1);
      END IF;
      IF (count_signal = COUNTER_OVERROLL) AND
        (pre_count_signal = PRE_COUNTER_OVERROLL) THEN
        count_signal <= UNSIGNED(load_reg0(7 DOWNT0 0));
      END IF;
    END IF;

    -- load counter function - load them at start of counting
    IF ((count_ctl(0) = '1') AND (count_ctl_q = '0')) THEN
      count_signal <= UNSIGNED(load_reg0(7 DOWNT0 0));
      pre_count_signal <= UNSIGNED(load_reg1);
    END IF;

    -- interrupt function
    IF (count_ctl(1) = '1') AND (count_ctl_q = '1') THEN
      IF (count_signal = COUNTER_OVERROLL) AND
        (pre_count_signal = PRE_COUNTER_OVERROLL) THEN
        local_irq <= '1';
      ELSE
        local_irq <= '0';
      END IF;
    END IF;
  END IF;
END PROCESS count;

```

The new features of the expanded process function as follows:

At the beginning a logical 1 must be written to bit0 of the *count_ctl* register to enable the start of the counter. One clock cycle later the *count_ctl_q* signal is automatically set, and the condition to start counting becomes true (i.e. the counter starts). This intermediate step is necessary to allow loading a start value into the two counter registers. The start value defines the number of clock cycles between start and overflow, and allows to vary the period of the counter. With each overflow of a counter register, the start value is also loaded again from the corresponding load register *load_reg0* or *load_reg1*. Changing the load registers before the overflow allows to alter the period. The interrupt function added as an additional function can trigger a short interrupt at an overflow of both counter registers. In this case the *local_irq* signal is set to 1 for one clock cycle and returns to its initial value afterwards. To enable the interrupt function bit 1 of the *count_ctl* register must be set.

3.2.4.5 The *led* process

The *led* process implements an LED output in the *counter.vhd* file in order to visualize counting of the Avalon counter. The LED output is connected to the interrupt signal of the Avalon counter and inverted with each interrupt signal. In order to activate the interrupt of the Avalon counter and thus the LED output, bit 1 of the *count_ctl* register must be set. The following listing shows the implementation of the LED output control in the *counter.vhd* file:

```
led: PROCESS(clk, clrn)
  BEGIN
  IF clrn = '0' THEN
    led_state    <= '1';
  ELSIF (clk'event AND clk = '1') THEN
    IF local_irq = '1' THEN
      led_state <= NOT led_state;
    END IF;
  END IF;
END PROCESS led;
```

--toggle LED on interrupt

3.2.4.6 The avalon_state_machine process

The *avalon_state_machine* process has been newly integrated into the counter VHDL file to implement the access control to the Avalon bus. For better clarity the *avalon_state_machine* process is kept simple and provides only the basic functions to access the Avalon bus. All further functions required are implemented in other processes outside the state machine process.

- Add the *avalon_state_machine* process to the *counter.vhd* file as shown in the listing below to implement the state machine.

```

avalon_state_machine: PROCESS(clk, clrn)
  BEGIN
  IF clrn = '0' THEN
    avalonstate    <= init;
    waitrequest_n  <= '0';
  ELSIF (clk'event AND clk = '1') THEN
    CASE avalonstate IS
      WHEN init => IF s_chipselect = '1' THEN      --start avalon slave state
machine
          avalonstate <= act_wait;      --change avalon state machine
          --state
          END IF;
          waitrequest_n <= '1';
      WHEN act_wait => avalonstate <= act;      --wait one clock cycle for
          --proper read/write data
      WHEN act => avalonstate <= rst;      --one clockcycle wait for
          --applying/reading data
          --on data bus
      WHEN rst => IF s_chipselect = '0' THEN
          avalonstate <= init;      --restoring intial state of state
          --machine
          END IF;
          waitrequest_n <= '0';      --terminating bus access
      WHEN OTHERS => avalonstate <= init;
    END CASE;
  END IF;
END PROCESS avalon_state_machine;

```

As the name of the *avalon_state_machine* process already indicates, the fundamental state machine function for the Avalon bus access is provided in this process and works as follows:

As described in the listing, the state machine is triggered by means of an active *s_chipselect* signal. Then, while setting the *waitrequest_n* signal to a high level the process changes to the *act_wait* status in order to wait for valid read/write data. If valid data is present, the process enters the active state *act* where data is then exchanged between the master and the slave of the Avalon system. As already mentioned, the functions for data exchange are implemented in other processes for better clarity. Following the active state the process switches to the *rst* state. In this state the *waitrequest_n* signal is set to low-level to indicate either the importing of the write data from the processor, or to request importing of the read data into the processor. In case of a reset, the state machine and the *waitrequest_n* signal are reset to their initial value.

3.2.4.7 The *avalon_wr* process

The *avalon_wr* process writes to the registers newly added in the project.

- To allow writing to the registers insert the *avalon_wr* process into the *counter.vhd* file according to the listing below.

```

avalon_wr: PROCESS(clk, clrn)
BEGIN
  IF clrn = '0' THEN
    load_reg0    <= (OTHERS => '0');
    load_reg1    <= (OTHERS => '0');
    count_ctl    <= (OTHERS => '0');
  ELSIF (clk'event AND clk = '1') THEN
    IF (avalonstate = act) AND (s_write = '1') THEN
      CASE s_address IS
        WHEN x"0" =>    count_ctl <= s_writedata;
                      --case s_byteenable is
                      --when "0001" => count_ctl(7 downto 0) <=
--      s_writedata(7 downto 0);
                      --when "0010" => count_ctl(15 downto 8) <=
--      s_writedata(15 downto 8);
                      -- ...
                      --end case;
        WHEN x"1" =>    load_reg0 <= s_writedata;
        WHEN x"2" =>    load_reg1 <= s_writedata;
        WHEN OTHERS =>    -- do nothing
      END CASE;
    END IF;
  END IF;
END PROCESS avalon_wr;

```

The *avalon_wr* process writes the available registers depending on the state of the Avalon state machine. If the state machine is in the active *act* state, the data available on the data bus is stored to the register which has been selected by the address bus *s_address*. The data is only loaded into the corresponding register in the *active* state if the *s_write* signal is at high level. If the address bus *s_address* has a state other than the explicitly evaluated states, no data is stored. If a low active reset is applied to the *clrn* signal, the content of all registers is deleted, and the registers are initialized with 0. Following the implementation described above, 32-bit writing of the registers is carried out during each access. This 32-bit writing could additionally be broken up into 16-bit or 8-bit writing by evaluating the *s_byteenable* signal (see comment in the *avalon_wr* process), but has not been implemented here because such writing to the counter registers would not be useful.

3.2.4.8 The *avalon_rd* process

The *avalon_rd* process reads the registers newly added in the project.

- To allow reading the registers insert the *avalon_rd* process into the *counter.vhd* file according to the listing below.

```

avalon_rd: PROCESS(clk, clrn)
  BEGIN
    IF clrn = '0' THEN
      -- no additional reset condition for this process
    ELSIF (clk'event AND clk = '1') THEN
      IF (s_read = '1') THEN
        CASE s_address IS
          WHEN x"0" => s_readdata <= count_ctl;
          WHEN x"1" => s_readdata <= load_reg0;
          WHEN x"2" => s_readdata <= load_reg1;
          WHEN x"3" => s_readdata(7 DOWNTO 0) <= count_reg0;
                      s_readdata(31 DOWNTO 8) <= (OTHERS => '0');
          WHEN x"4" => s_readdata <= count_reg1;
          WHEN OTHERS => s_readdata <= (OTHERS => '0');
        end case;
      END IF;
    END IF;
  END PROCESS avalon_rd;

```

Depending on the signal *s_read* and the addresses *s_address* the process *avalon_rd* carries out the reading of the registers in the Counter project. A dependency of the reading process on other signals as well as the control

of additional signals is not necessary, as all other signal states are handled from the Avalon state machine.


- To complete editing the *counter.vhd* delete the second last line:

```
count <= STD_LOGIC_VECTOR(count_signal);
```

3.2.5 Compiling and testing the Avalon counter project

3.2.5.1 Compiling

Before we proceed with the final configuration of the project, we can check if all processes of the Avalon counter project have been implemented and adapted as described above.

- Click on the  *Start Analysis & Synthesis* icon to start the analysis and to test the implementation of the new code.
- Please re-examine the implementation if the analysis does not result in the message shown in the following figure.

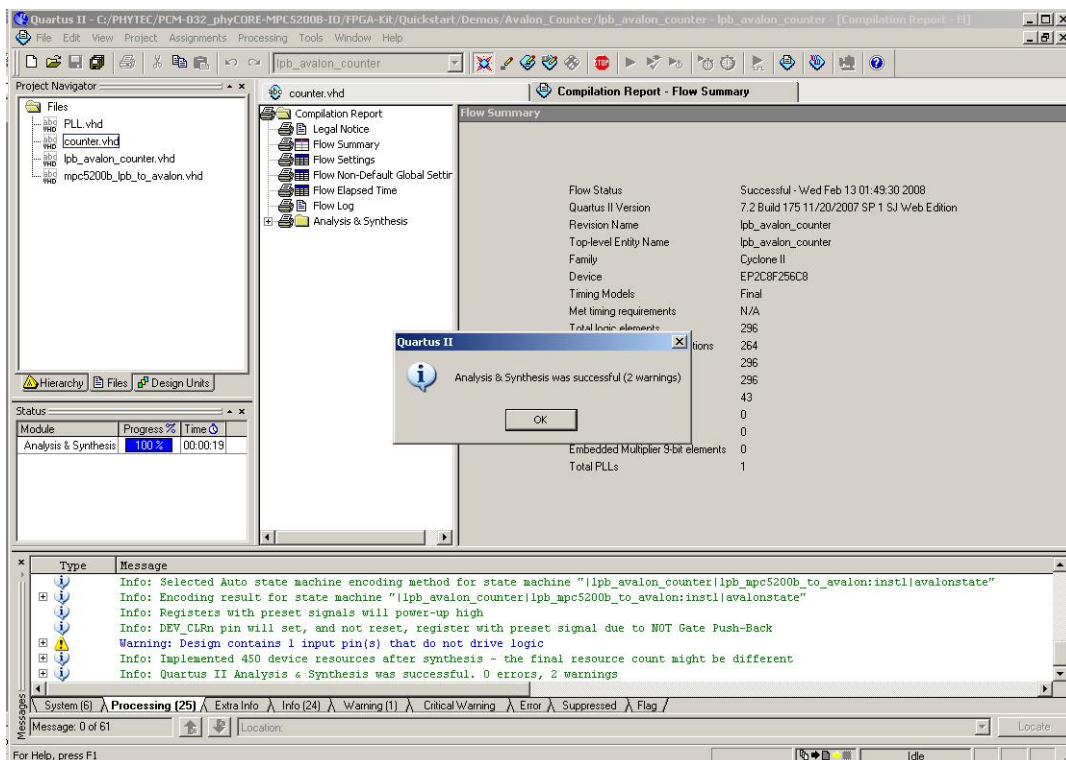


Figure 69: Result of the analysis and synthesis process of the Avalon-Counter project

When carrying out the synthesis, the Quartus® II tool chain generates two warnings resulting from the fact that all signals connected from the MPC5200B to the FPGA have been taken into consideration when instancing the top design file. However, this is not a problem for further

execution of the project. The two warnings are displayed in the window as follows:

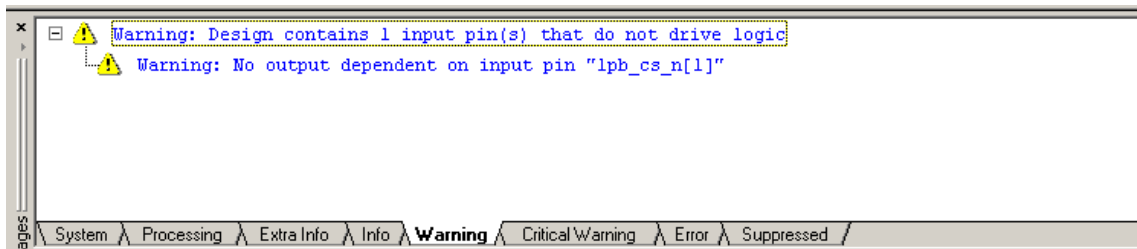


Figure 70: Messages in the *Warning* tab

If all settings of the Counter project were taken over, still the assignments for the bus interface between FPGA and MPC5200B must be carried out.

- Start the Assignment Editor by selecting *Assignment Editor* in the *Assignments* menu.
- Enter the assignments for the bus interface according to the following table (refer to section 3.1.1. for a detailed step-by-step description).
- Continue to configure the output pin load for all signals. A usable value for this constraint is 4 as described in section 3.1.1.

| TO | Location | I/OBank | I/O Standard |
|-------------|----------|---------|--------------|
| lpb_ad[0] | PIN_K11 | 4 | 3.3-V LVCMOS |
| lpb_ad[1] | PIN_K10 | 4 | 3.3-V LVCMOS |
| lpb_ad[2] | PIN_P12 | 4 | 3.3-V LVCMOS |
| lpb_ad[3] | PIN_P13 | 4 | 3.3-V LVCMOS |
| lpb_ad[4] | PIN_T12 | 4 | 3.3-V LVCMOS |
| lpb_ad[5] | PIN_R12 | 4 | 3.3-V LVCMOS |
| lpb_ad[6] | PIN_T13 | 4 | 3.3-V LVCMOS |
| lpb_ad[7] | PIN_R13 | 4 | 3.3-V LVCMOS |
| lpb_ad[8] | PIN_T14 | 4 | 3.3-V LVCMOS |
| lpb_ad[9] | PIN_R14 | 4 | 3.3-V LVCMOS |
| lpb_ad[10] | PIN_M11 | 4 | 3.3-V LVCMOS |
| lpb_ad[11] | PIN_L11 | 4 | 3.3-V LVCMOS |
| lpb_ad[12] | PIN_N11 | 4 | 3.3-V LVCMOS |
| lpb_ad[13] | PIN_P11 | 4 | 3.3-V LVCMOS |
| lpb_ad[14] | PIN_T3 | 4 | 3.3-V LVCMOS |
| lpb_ad[15] | PIN_R3 | 4 | 3.3-V LVCMOS |
| lpb_ad[16] | PIN_P5 | 4 | 3.3-V LVCMOS |
| lpb_ad[17] | PIN_P4 | 4 | 3.3-V LVCMOS |
| lpb_ad[18] | PIN_T4 | 4 | 3.3-V LVCMOS |
| lpb_ad[19] | PIN_R4 | 4 | 3.3-V LVCMOS |
| lpb_ad[20] | PIN_T5 | 4 | 3.3-V LVCMOS |
| lpb_ad[21] | PIN_R5 | 4 | 3.3-V LVCMOS |
| lpb_ad[22] | PIN_T7 | 4 | 3.3-V LVCMOS |
| lpb_ad[23] | PIN_R7 | 4 | 3.3-V LVCMOS |
| lpb_ad[24] | PIN_L7 | 4 | 3.3-V LVCMOS |
| lpb_ad[25] | PIN_L8 | 4 | 3.3-V LVCMOS |
| lpb_ad[26] | PIN_T8 | 4 | 3.3-V LVCMOS |
| lpb_ad[27] | PIN_R8 | 4 | 3.3-V LVCMOS |
| lpb_ad[28] | PIN_T9 | 4 | 3.3-V LVCMOS |
| lpb_ad[29] | PIN_R9 | 4 | 3.3-V LVCMOS |
| lpb_ad[30] | PIN_N8 | 4 | 3.3-V LVCMOS |
| lpb_ad[31] | PIN_T6 | 4 | 3.3-V LVCMOS |
| lpb_cs_n[0] | PIN_N10 | 4 | 3.3-V LVCMOS |
| lpb_cs_n[1] | PIN_T11 | 4 | 3.3-V LVCMOS |
| lpb_oe_n | PIN_L9 | 4 | 3.3-V LVCMOS |
| lpb_ack_n | PIN_R10 | 4 | 3.3-V LVCMOS |
| lpb_ale_n | PIN_R11 | 4 | 3.3-V LVCMOS |
| lpb_rdwr_n | PIN_L10 | 4 | 3.3-V LVCMOS |
| lpb_ts_n | PIN_T10 | 4 | 3.3-V LVCMOS |
| lpb_rst_n | PIN_N9 | 4 | 3.3-V LVCMOS |
| lpb_int | PIN_R6 | 4 | 3.3-V LVCMOS |
| lpb_clk | PIN_H2 | 1 | 3.3-V LVCMOS |
| led_out | PIN_P1 | 1 | 3.3-V LVCMOS |

Table 3: Assignment of the bus interface of the Avalon-Counter project

The following two images show the pin assignments of the MPC5200B - FPGA bus interfaces in the category *Locations / Pin* of the Assignment Editor and *Logic Options / I/O Features*.

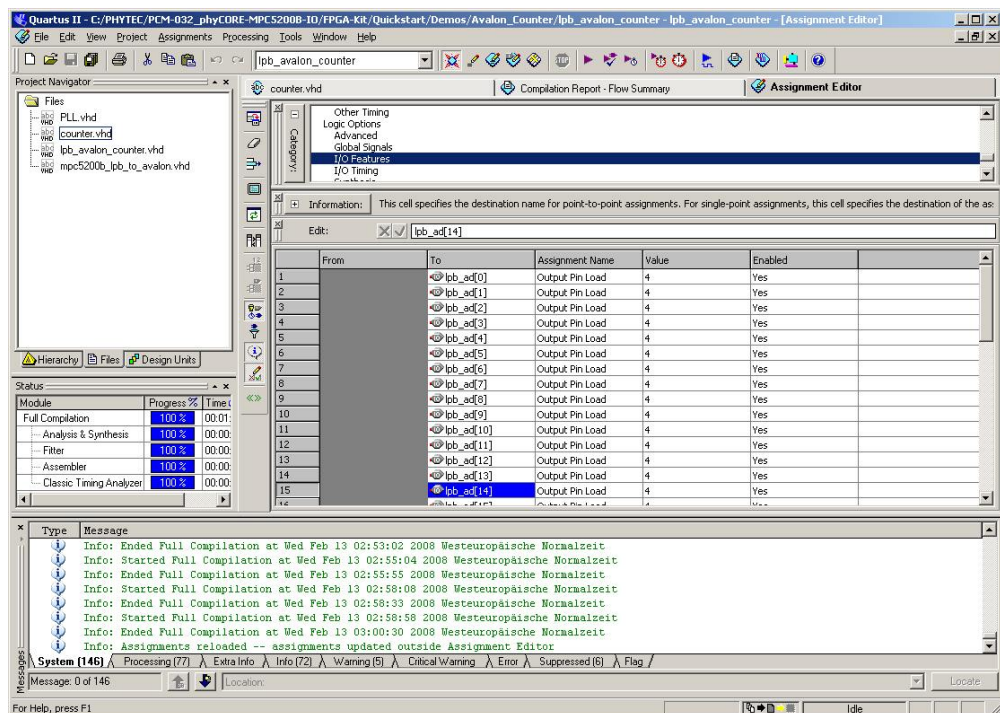
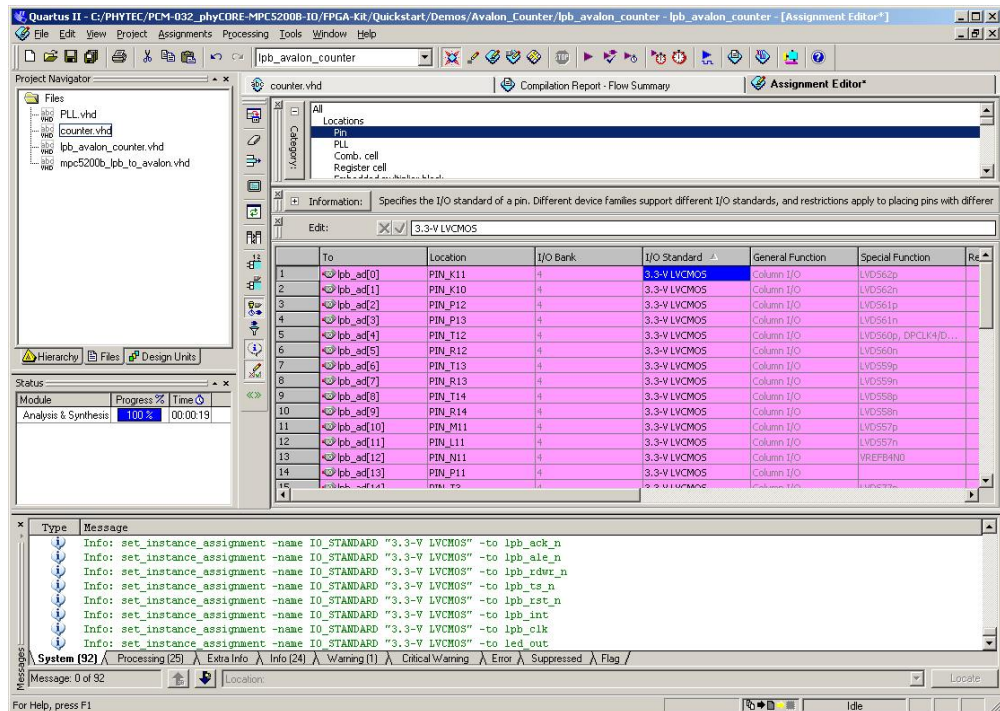



Figure 71: Assignment of the pins of the MPC5200B bus interfaces

Now after successful setting of the Assignments all conditions for a successful compilation of the project are given.

- Click on the  *Start Compilation* icon or choose *Start Compilation* from the *Processing* menu to start the compilation. During compilation the following modules will be executed:

- Analysis and Synthesis
- Fitter
- Assembler
- Classic Timing Analysis

After successful compilation of the project the following screen should be displayed:

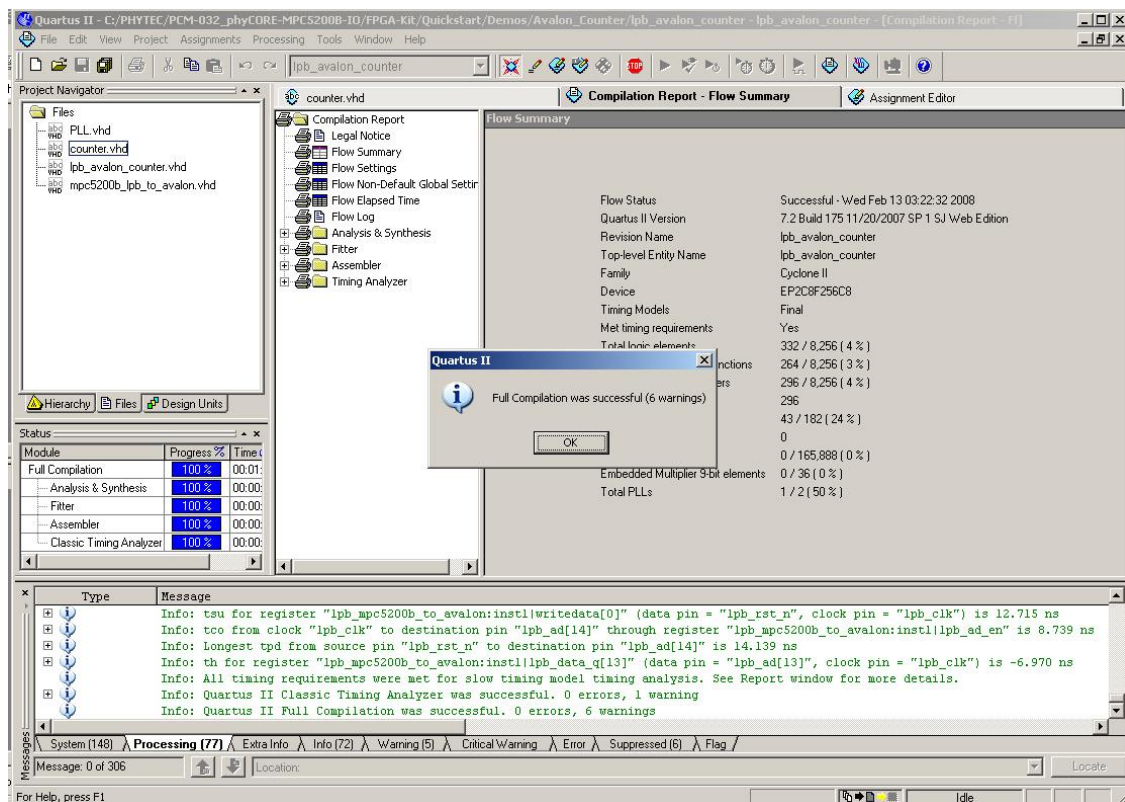


Figure 72 Result of the full compilation process

The 6 warnings you will receive can be viewed in the *Warning* tab of the message window for further evaluation. As they are not critical we can proceed with the download of the new project and the executing on the target.

- Push the *OK* button.

3.2.5.2 Programming the FPGA

Following the successful compilation, the generated programming file can now be prepared for downloading to the FPGA.

- First open the programmer by choosing *Programmer* from the *Tools* menu. When opening the programmer, the associated programming file is automatically loaded by the Quartus® II tool chain and configured for programming.

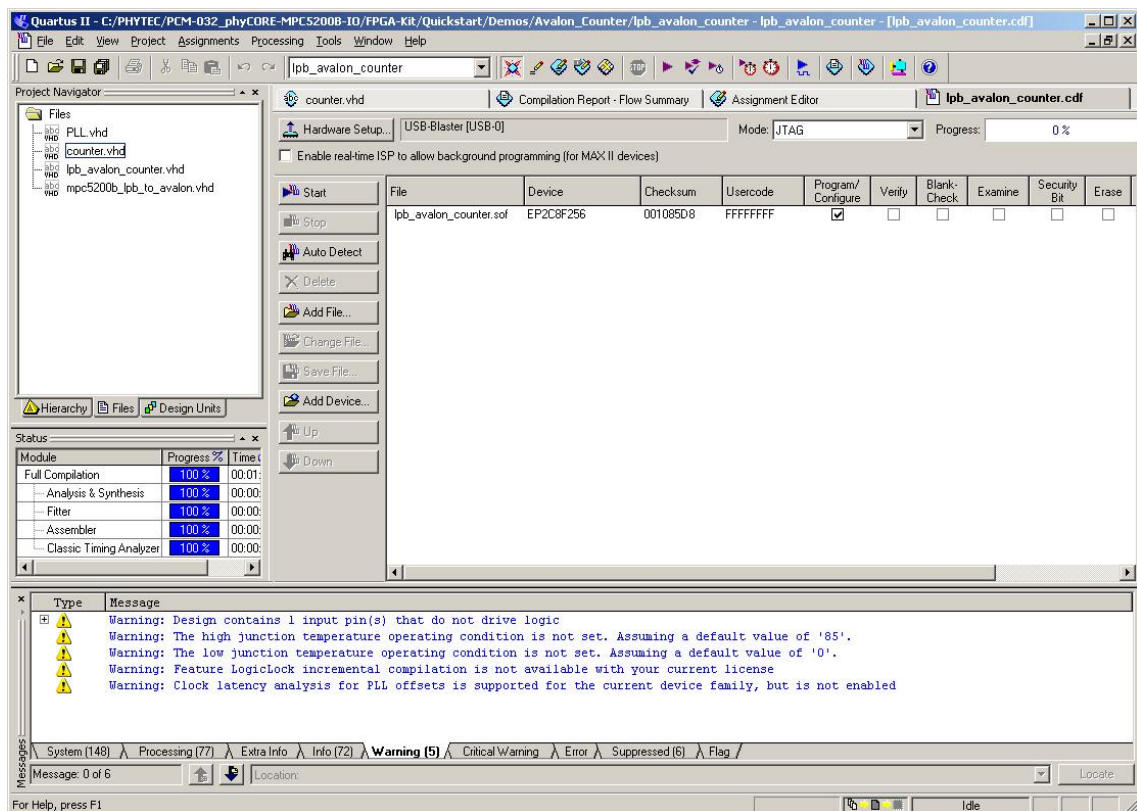



Figure 73 Quartus® II tool chain the open *Programmer* window

Before finally programming the project to the FPGA the programming hardware needs to be connected and the hardware setup must be carried out.

- Connect the host to the target's FPGA JTAG interface as described in section 2.3.
- Now perform the hardware setup to configure the programming device adapter as described in section 2.4
- Press the  *Start* button to initiate the programming process.

The progress of the programming is shown in the upper right corner of the Programmer window, whereas additional information on the programming status is displayed in the *System* tab of the message window at the bottom of the screen.

The following figure shows the Programmer window with the information described above:

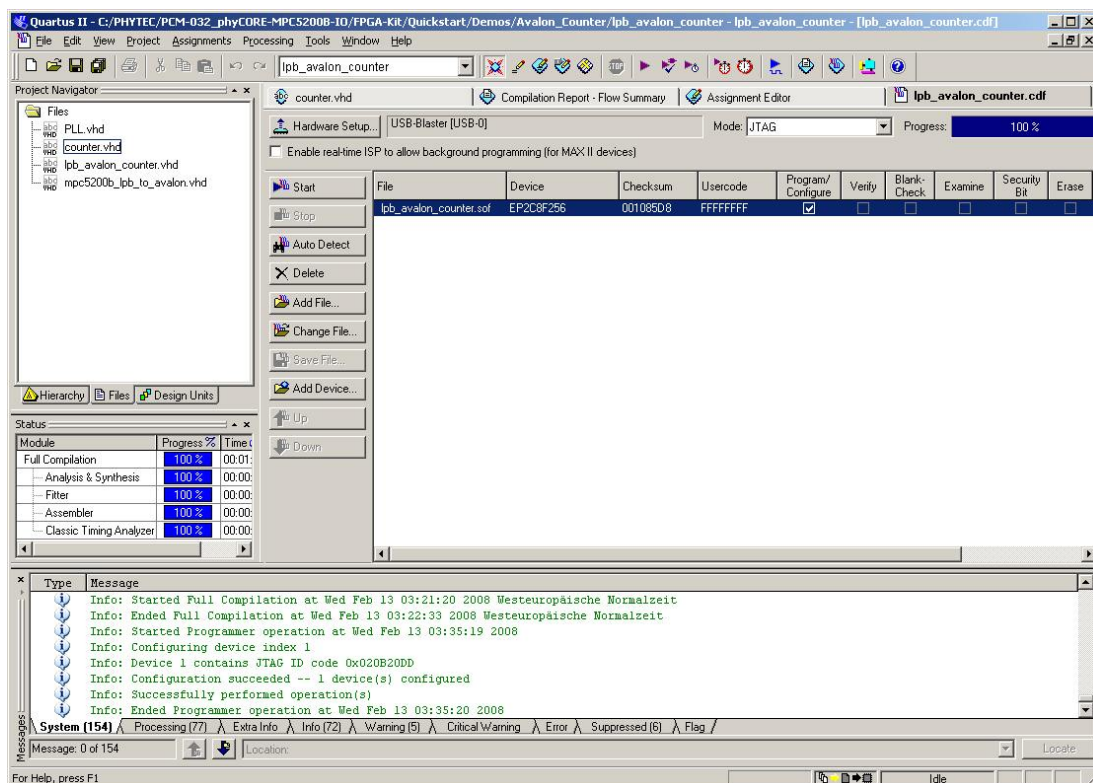


Figure 74: Programmer window with information on the progress and on the programming status

3.2.5.3 Testing of the Avalon_Counter project on the hardware

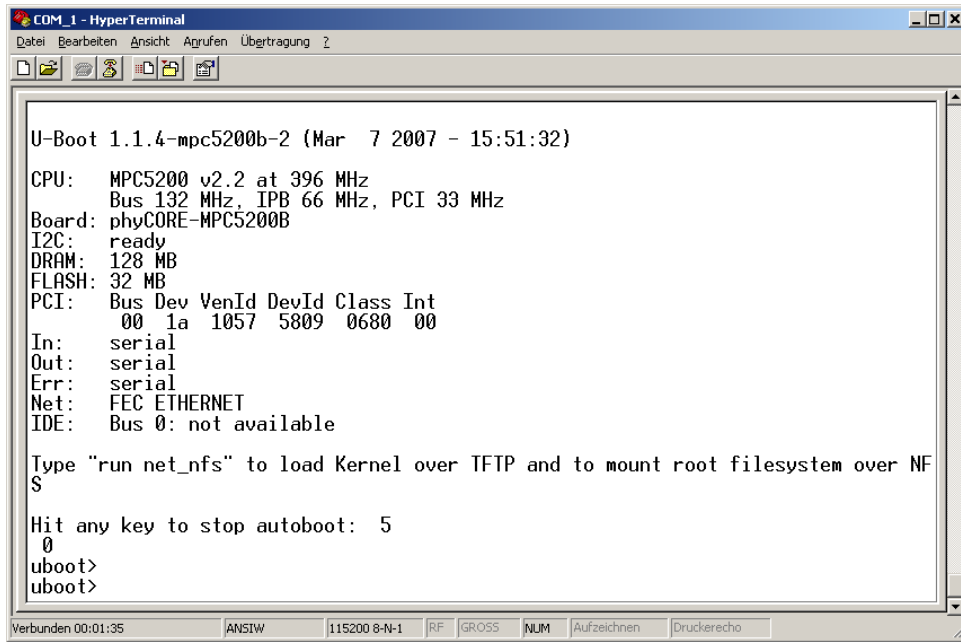
Once programming has been completed the Avalon_Counter project can be tested. In order to test the project the following requirements must be fulfilled:

- A Terminal program with support of transmission rates up to 115.2 Kbps must be installed on the host PC. The host PC must have a serial port.
- You must have the phyCORE-MPC5200B-IO with installed U-Boot loader plugged on to the development board.
- A Serial interface cable must connect COM0 (lower connector of P3) of the development board to a serial port available on the host PC.

If the above listed requirements are fulfilled you can proceed with testing the project.

- Start the terminal program on the host PC. Configure the terminal program with the following parameters to connect to the U-Boot loader on the phyCORE-MPC5200B-IO:
115200 bps, 8, n, 1.
- Since the U-Boot loader has already been executed following power up, reset the board again using the *Reset* button to check the boot process of the U-Boot loader.
- During the boot process, press *Enter* on the keyboard of the host PC to send a CR and LF to the U-Boot loader in order to interrupt any further boot processes. This should result in an U-Boot loader prompt appearing as the last output in the terminal window.

Now you can start testing the project by entering further commands.



```

COM_1 - HyperTerminal
Datei Bearbeiten Ansicht Aufrufen Übertragung ?
U-Boot 1.1.4-mpc5200b-2 (Mar 7 2007 - 15:51:32)
CPU: MPC5200 v2.2 at 396 MHz
      Bus 132 MHz, IPB 66 MHz, PCI 33 MHz
Board: phyCORE-MPC5200B
I2C: ready
DRAM: 128 MB
FLASH: 32 MB
PCI: Bus Dev VenId DevId Class Int
      00 1a 1057 5809 0680 00
In: serial
Out: serial
Err: serial
Net: FEC ETHERNET
IDE: Bus 0: not available

Type "run net_nfs" to load Kernel over TFTP and to mount root filesystem over NFS

Hit any key to stop autoboot: 5
0
uboot>
uboot>

```

Figure 75: Output of the U-Boot loader via the serial interface

As the FPGA is mapped to the memory area of the MPC5200B it is possible to access the FPGA's memory by means of special commands of the U-boot loader. This allows easy testing of the FPGA application. The U-boot loader commands to access the MPC5200B's memory and hence also the FPGA's memory, are “md” (memory dump) and “mw” (memory write).

In the basic configuration of the MPC5200B's U-Boot loader any access to the memory area from $0xE2000000$ to $0xE3FFFFFF$ enables chip select signal /CS3 which selects the FPGA in the Avalon-Counter project. In other words, any read or write access to this memory area allows access to the application programmed in the FPGA.

- Now enter the U-boot loader command **md 0xE2000000** in the terminal window to perform a memory dump starting at address $0xE2000000$ in order to view the registers of the Avalon-Counter project.



The implemented register set recurs completely in a memory dump with an offset of $0x40$, because the higher Avalon addresses of the slave interface are not used for decoding.


```

COM_1 - HyperTerminal
Datei Bearbeiten Ansicht Agrufen Übertragung ?
IDE: Bus 0: not available

Type "run net_nfs" to load Kernel over TFTP and to mount root filesystem over NFS

Hit any key to stop autoboot: 0
uboot> md 0xe2000000
e2000000: 00000000 00000000 00000000 00000000 .....
e2000010: 00000000 00000000 00000000 00000000 .....
e2000020: 00000000 00000000 00000000 00000000 .....
e2000030: 00000000 00000000 00000000 00000000 .....
e2000040: 00000000 00000000 00000000 00000000 .....
e2000050: 00000000 00000000 00000000 00000000 .....
e2000060: 00000000 00000000 00000000 00000000 .....
e2000070: 00000000 00000000 00000000 00000000 .....
e2000080: 00000000 00000000 00000000 00000000 .....
e2000090: 00000000 00000000 00000000 00000000 .....
e20000a0: 00000000 00000000 00000000 00000000 .....
e20000b0: 00000000 00000000 00000000 00000000 .....
e20000c0: 00000000 00000000 00000000 00000000 .....
e20000d0: 00000000 00000000 00000000 00000000 .....
e20000e0: 00000000 00000000 00000000 00000000 .....
e20000f0: 00000000 00000000 00000000 00000000 .....
uboot>
Verbinden 00:01:07 ANSIW 115200 8-N-1 RF GROSS NUM Aufzeichnen Druckerecho

```

Figure 76: FPGA memory dump of the U-boot loader

As can be seen in the memory dump all the registers of the Avalon-Counter project are initialized with logical '0' which corresponds to the assignments made in the *counter.vhd* file.

- Next enter **mw 0xE2000000 0x01** to set bit0 of the counter control register *count_ctl* in order to start the counter.

```

COM_1 - HyperTerminal
Datei Bearbeiten Ansicht Agrufen Übertragung ?
Type "run net_nfs" to load Kernel over TFTP and to mount root filesystem over NFS

Hit any key to stop autoboot: 0
uboot> md 0xe2000000
e2000000: 00000000 00000000 00000000 00000000 .....
e2000010: 00000000 00000000 00000000 00000000 .....
e2000020: 00000000 00000000 00000000 00000000 .....
e2000030: 00000000 00000000 00000000 00000000 .....
e2000040: 00000000 00000000 00000000 00000000 .....
e2000050: 00000000 00000000 00000000 00000000 .....
e2000060: 00000000 00000000 00000000 00000000 .....
e2000070: 00000000 00000000 00000000 00000000 .....
e2000080: 00000000 00000000 00000000 00000000 .....
e2000090: 00000000 00000000 00000000 00000000 .....
e20000a0: 00000000 00000000 00000000 00000000 .....
e20000b0: 00000000 00000000 00000000 00000000 .....
e20000c0: 00000000 00000000 00000000 00000000 .....
e20000d0: 00000000 00000000 00000000 00000000 .....
e20000e0: 00000000 00000000 00000000 00000000 .....
e20000f0: 00000000 00000000 00000000 00000000 .....
uboot> mw 0xe2000000 0x01
uboot>
Verbinden 00:11:53 ANSIW 115200 8-N-1 RF GROSS NUM Aufzeichnen Druckerecho

```

Figure 77: Starting the Avalon counter

- Type **md 0xE2000000** to perform a new memory dump starting at address *0xE2000000* in order to get the actual contents of the registers. Now you can verify the correct start of the counter.

```

COM_1 - HyperTerminal
Datei Bearbeiten Ansicht Anrufen Übertragung ?
e20000b0: 00000000 00000000 00000000 00000000 .....
e20000c0: 00000000 00000000 00000000 00000000 .....
e20000d0: 00000000 00000000 00000000 00000000 .....
e20000e0: 00000000 00000000 00000000 00000000 .....
e20000f0: 00000000 00000000 00000000 00000000 .....
uboot> mw 0xe2000000 0x01
uboot> md 0xe2000000
e2000000: 00000001 00000000 00000000 00000002 .....
e2000010: a4e28dd2 00000000 00000000 00000000 .....
e2000020: 00000000 00000000 00000000 00000000 .....
e2000030: 00000000 00000000 00000000 00000000 .....
e2000040: 00000001 00000000 00000000 00000002 .....
e2000050: a4ee6ed8 00000000 00000000 00000000 ..n.....
e2000060: 00000000 00000000 00000000 00000000 .....
e2000070: 00000000 00000000 00000000 00000000 .....
e2000080: 00000001 00000000 00000000 00000002 .....
e2000090: a4fa4e72 00000000 00000000 00000000 ..Nr.....
e20000a0: 00000000 00000000 00000000 00000000 .....
e20000b0: 00000000 00000000 00000000 00000000 .....
e20000c0: 00000001 00000000 00000000 00000002 .....
e20000d0: a5062ee7 00000000 00000000 00000000 .....
e20000e0: 00000000 00000000 00000000 00000000 .....
e20000f0: 00000000 00000000 00000000 00000000 .....
uboot> _
Verbunden 00:17:36 ANSIV 115200 8-N-1 RF GROSS NUM Aufzeichnen Druckerecho

```

Figure 78: Actual count of the Avalon Counter

As you can see in the above illustration, the value of the *count_ctl* register at address *0xE2000000* is *0x01* which means that the counter is activated. As can be seen further, the two registers at addresses *0xE200000C* and *0xE2000010* have a random value. These are the two counter registers *count_reg0* and *count_reg1* which have been automatically assigned the contents of the actual counter registers *count_signal* and *pre_count_signal*. The two registers at addresses *0xE2000004* and *0xE2000008* are the two load registers *load_reg0* and *load_reg1* which have been implemented for loading the count registers with a predefined value in order to change the time interval between the interrupt signals.

- In order to get a specific period of the counter enter the commands **mw 0xE2000004 0xA0** and **mw 0xE2000008 0xFFFF0000**.

- Type **md 0xE2000000** to perform a new memory dump.

As you can see in the screenshot below the two values are written to the load registers at addresses *0xE2000004* and *0xE2000008*.

```

COM_1 - HyperTerminal
Datei Bearbeiten Ansicht Anrufen Übertragung ?
e20001c0: 00000001 00000000 00000000 00000098 .....
e20001d0: 1202f36f 00000000 00000000 00000000 ...o.....
e20001e0: 00000000 00000000 00000000 00000000 .....
e20001f0: 00000000 00000000 00000000 00000000 .....
uboot> mw 0xe2000004 0xa0
uboot> mw 0xe2000008 0xffff0000
uboot> md 0xe2000000
e2000000: 00000001 000000a0 ffff0000 00000098 .....
e2000010: 4390761a 00000000 00000000 00000000 C.v.....
e2000020: 00000000 00000000 00000000 00000000 .....
e2000030: 00000000 00000000 00000000 00000000 .....
e2000040: 00000001 000000a0 ffff0000 00000098 .....
e2000050: 439c55a7 00000000 00000000 00000000 C.U.....
e2000060: 00000000 00000000 00000000 00000000 .....
e2000070: 00000000 00000000 00000000 00000000 .....
e2000080: 00000001 000000a0 ffff0000 00000098 .....
e2000090: 43a8347b 00000000 00000000 00000000 C.4{.....
e20000a0: 00000000 00000000 00000000 00000000 .....
e20000b0: 00000000 00000000 00000000 00000000 .....
e20000c0: 00000001 000000a0 ffff0000 00000098 .....
e20000d0: 43b41416 00000000 00000000 00000000 C.....
e20000e0: 00000000 00000000 00000000 00000000 .....
e20000f0: 00000000 00000000 00000000 00000000 .....
uboot>
Verbunden 00:48:48 ANSIV 115200 8-N-1 RF GROSS NUM Aufzeichnen Druckerecho

```

Figure 79: Avalon Counter with configured load registers

- Now carry out a memory dump with the **md 0xE2000000** command several times in succession and watch the counting noticeable through the changes in the registers at the addresses *0xe200000c* and *0xe2000010*.

You can recognize that the two counter registers will always have values equal or higher than the contents of the two load registers. This indicates that the two counter registers are always reloaded with the contents of the corresponding load registers following an overflow. You can force the reloading of the counter registers by restarting the counter using the *mw 0xe2000000 0x00* (disable counter) and *mw 0xe2000000 0x01* (restart counter) commands.

```

COM_1 - HyperTerminal
Datei Bearbeiten Ansicht Agrufen Übertragung ?
e20000a0: 00000000 00000000 00000000 00000000 .....
e20000b0: 00000000 00000000 00000000 00000000 .....
e20000c0: 00000001 000000a0 ffff0000 00000098 .....
e20000d0: 43b41416 00000000 00000000 00000000 C.....
e20000e0: 00000000 00000000 00000000 00000000 .....
e20000f0: 00000000 00000000 00000000 00000000 .....
uboot> md 0xe2000000
e2000000: 00000001 000000a0 ffff0000 000000e4 .....
e2000010: ffff0c04 00000000 00000000 00000000 .....
e2000020: 00000000 00000000 00000000 00000000 .....
e2000030: 00000000 00000000 00000000 00000000 .....
e2000040: 00000001 000000a0 ffff0000 000000f0 .....
e2000050: ffffec7a 00000000 00000000 00000000 ...Z.....
e2000060: 00000000 00000000 00000000 00000000 .....
e2000070: 00000000 00000000 00000000 00000000 .....
e2000080: 00000001 000000a0 ffff0000 000000fc .....
e2000090: ffffcc39 00000000 00000000 00000000 ...9.....
e20000a0: 00000000 00000000 00000000 00000000 .....
e20000b0: 00000000 00000000 00000000 00000000 .....
e20000c0: 00000001 000000a0 ffff0000 000000a7 .....
e20000d0: ffffacc0 00000000 00000000 00000000 .....
e20000e0: 00000000 00000000 00000000 00000000 .....
e20000f0: 00000000 00000000 00000000 00000000 .....
uboot>
Verbunden 00:51:01 ANSIW 115200 8-N-1 RF GROSS NUM Aufzeichnen Druckerecho

```

Figure 80: Repeated memory dump of the Avalon counters registers following an overflow

Besides observing the function of the counter within the terminal window we can visualize the counter overflow and the resulting interrupt with the LED.

- In order to activate the interrupt of the Avalon counter set bit 1 of the *count_ctl* register to 1 by entering **mw 0xE2000000 0x03**. Writing 0x03 to the *count_ctl* register not only activates the interrupt, but also restarts the counter. Now the LED on the development board should flash at a frequency of approx. 2.5 Hz.

The flash rate results of the values (0xA0 and 0xFFFF0000) previously written to the two load registers *load_reg0* and *load_reg1*. Varying the two registers' contents will change the flash rate, in doing so values between 0x00 and 0xFF are allowed for *load_reg0* and values between 0x00000000 and 0xFFFFFFFF for *load_reg1*.



In this section you have learned to add an Avalon Slave interface to a logic project in order to make the corresponding design available to an external bus.

3.3 Use of the SOPC Builder

In this section you will learn how to add new Avalon components using the SOPC builder and how to use them in the SOPC builder project.

The SOPC Builder is a powerful system development tool for creating systems including processors, peripherals, and memories. The SOPC Builder enables to define and generate a complete system-on-a-programmable-chip (SOPC) in much less time than using traditional, manual integration methods. The SOPC Builder is included in the Quartus® II software. For detailed information about the SOPC Builder see Quartus® II Handbook, Volume 4.

In order to learn how to use the SOPC builder it is advisable to create a new project rather than changing the counter project discussed in section 3.1.1.

- First copy the VHDL file *counter.vhd* from the counter project discussed in section 3.2.4 into the working directory of the new project. The new working directory `C:\PHYTEC\PCM-032_phyCORE-MPC5200B-IO\FPGA-Ki\Quickstart\Demos\Avalon_Counter_SOPC` has been created on your hard disk by the installation process and already contains some files.
- Now use the *New Project Wizard* as described in section 3.1.1 to create a new project.
- As the working directory please enter `C:\PHYTEC\PCM-032_phyCORE-MPC5200B-IO\FPGA-Ki\Quickstart\Demos\Avalon_Counter_SOPC`.
- For the project's name and the top-level design entity please enter *Avalon_Counter_SOPC* in the appropriate fields of the *NewProject Wizard* window as shown below. Click the *Next* button to confirm the entries.

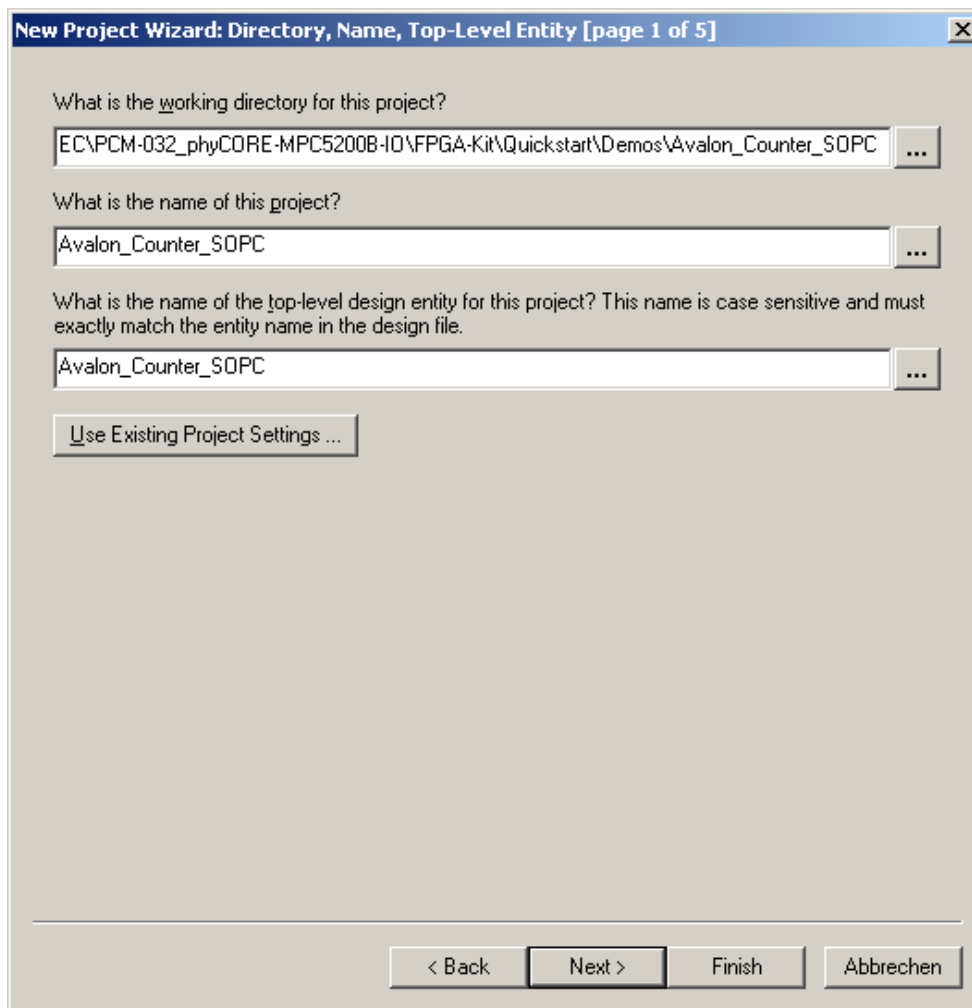


Figure 81: Working directory and project name of the *Avalon_Counter_SOPC* project

- At this point no files will be added to the project. Click the *Next* button to skip the *New Project Wizard: Add Files [page 2 of 5]* window.
- Continue to configure the project settings as described in section 3.1.1.
- If all project settings are done properly the summary of the project settings should look the same as in the next figure. Click on the *Finish* button to finalize the project configuration.

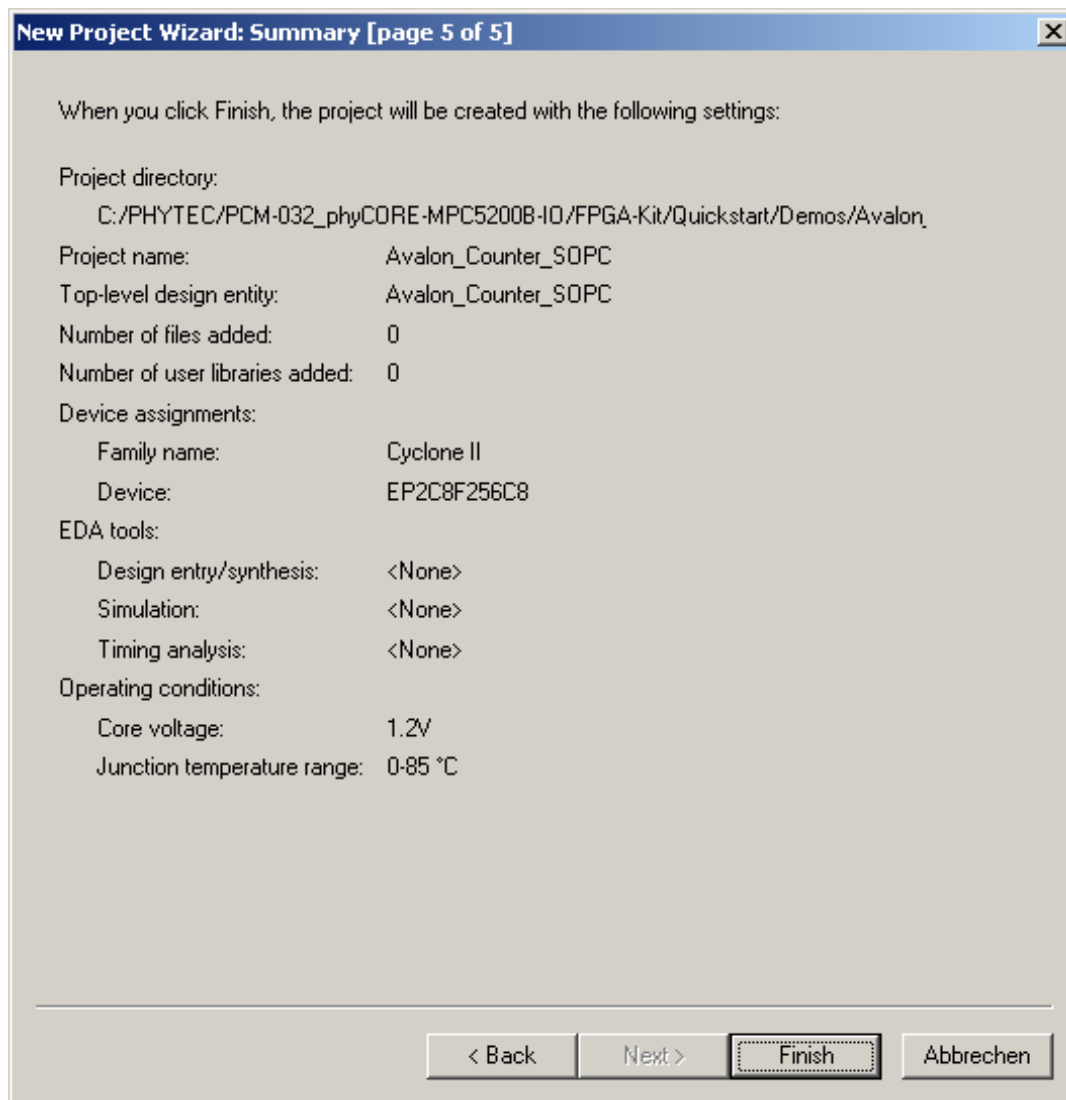



Figure 82: Summary of the project settings

- Start the SOPC Builder by clicking the  *SOPC Builder* icon or choosing *SOPC Builder* from the *Tools* menu.

The dialog window of the SOPC builder appears as shown in the next figure.

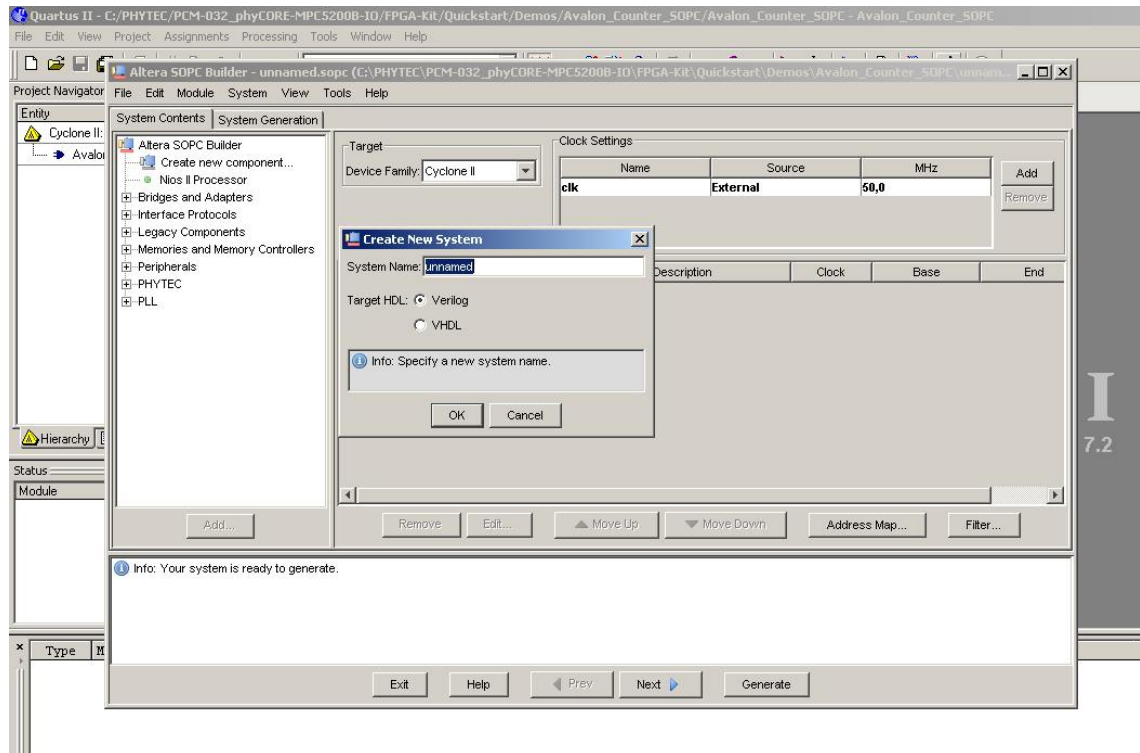


Figure 83: SOPC Builder main window

- In the *Create New System* window which opens automatically after starting the SOPC Builder enter `Avalon_Counter_SOPC` in the field *System Name* and check **VHDL**.

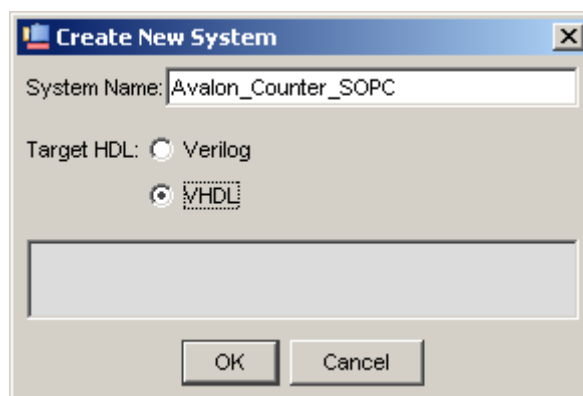


Figure 84: Entering the System Name

- Click OK to return to the main window of the *SOPC Builder*

A system build with the SOPC Builder consists of components which can be created, configured and connected with each other. A list of the components already available can be found in the *System Contents* tab on the left side of the SOPC Builder main window.

- Look for the component group PHYTEC within the *System Contents* tab. Click on the *plus* icon to expand the group. You can see that this group contains the component *lpb_mpc5200b_to_avalon*. The *lpb_mpc5200b_to_avalon* component is preinstalled and embodies the MPC5200B's Local Plus Bus interface which is needed for this example.

3.3.1 Creating a new component with the SOPC Builder

Besides the MPC5200B's Local Plus Bus interface we also need the counter function. The counter will be implemented as a new component.

- Double click on *Create New Component* within the *System Contents* tab of the SOPC Builder to open the Component Editor.

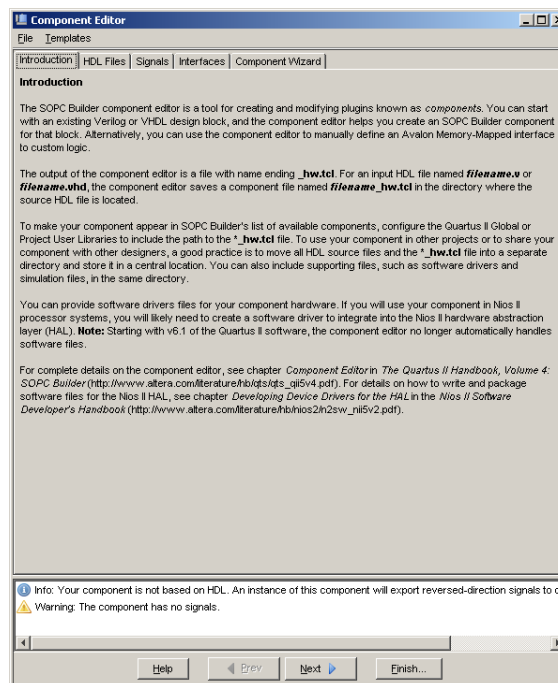


Figure 85: SOPC Component Editor

- Click *Next* to proceed to the *HDL Files* tab sheet which allows adding of HDL files to the new component.
- Press the *Add HDL File...* button to open the *File Open* window. Select the *counter.vhd* file which you copied in a previous step into the working directory *C:\PHYTEC\PCM-032_phyCORE-MPC5200B-IO\FPGA-Kit\Quickstart\Demos\Avalon_Counter_SOPC* and click *Open*.

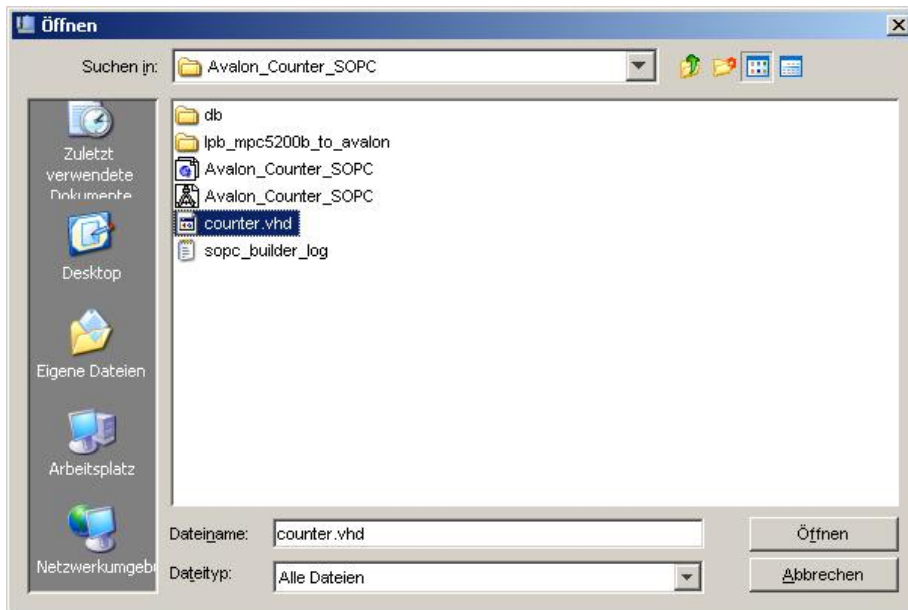


Figure 86: Adding the *counter.vhd* file

After you selected the *counter.vhd* file the Component Editor immediately runs Quartus® II Analysis & Elaboration in the background to analyze the HDL file to determine the ports and parameters of the file, and if the syntax of the file is correct. The following window indicates the analysis.

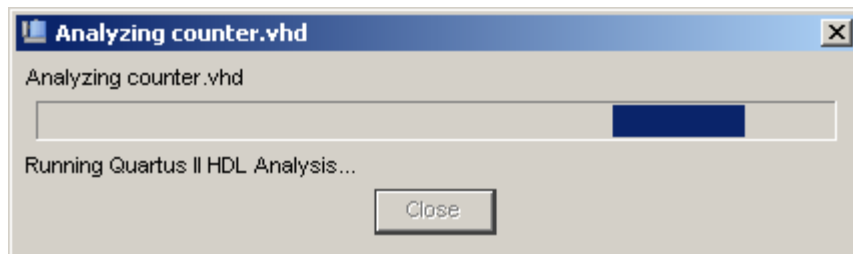


Figure 87: Analysis & Elaboration of the HDL-Files

The result of the analysis will be shown in the following window

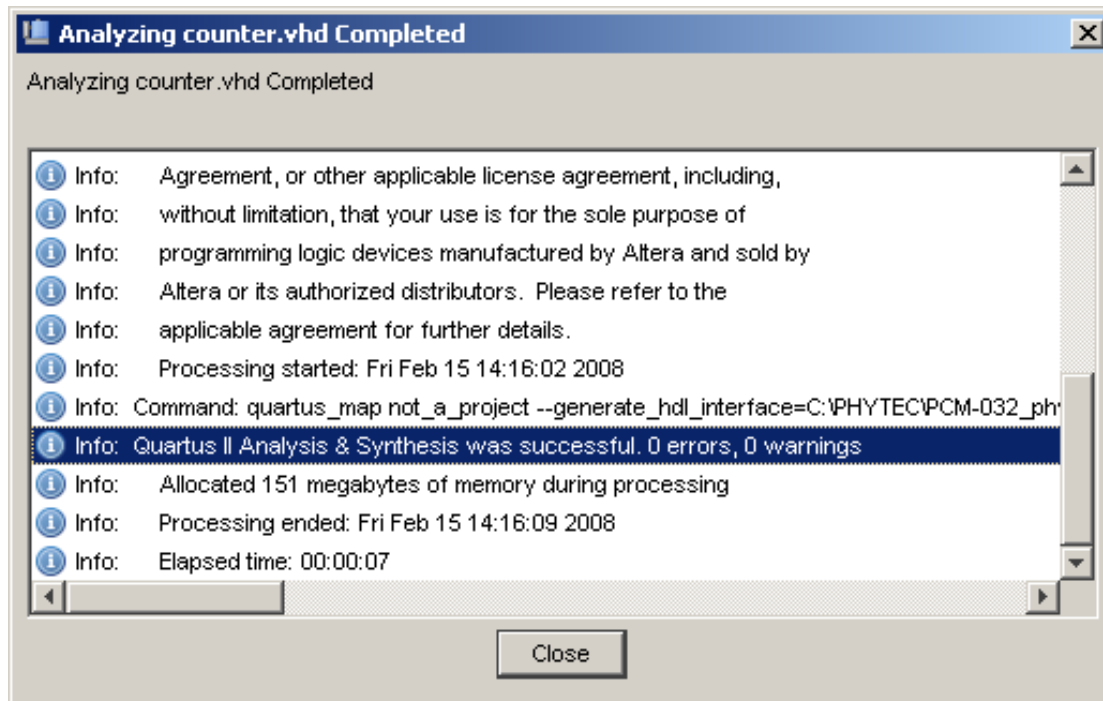


Figure 88: Result of the analysis of the HDL file

- Click *Close* to continue with editing the new component. If you received any errors or warnings verify the syntax of the counter.vhd file.

The *HDL Files* tap of the Component Editor should appear as shown in the next figure.

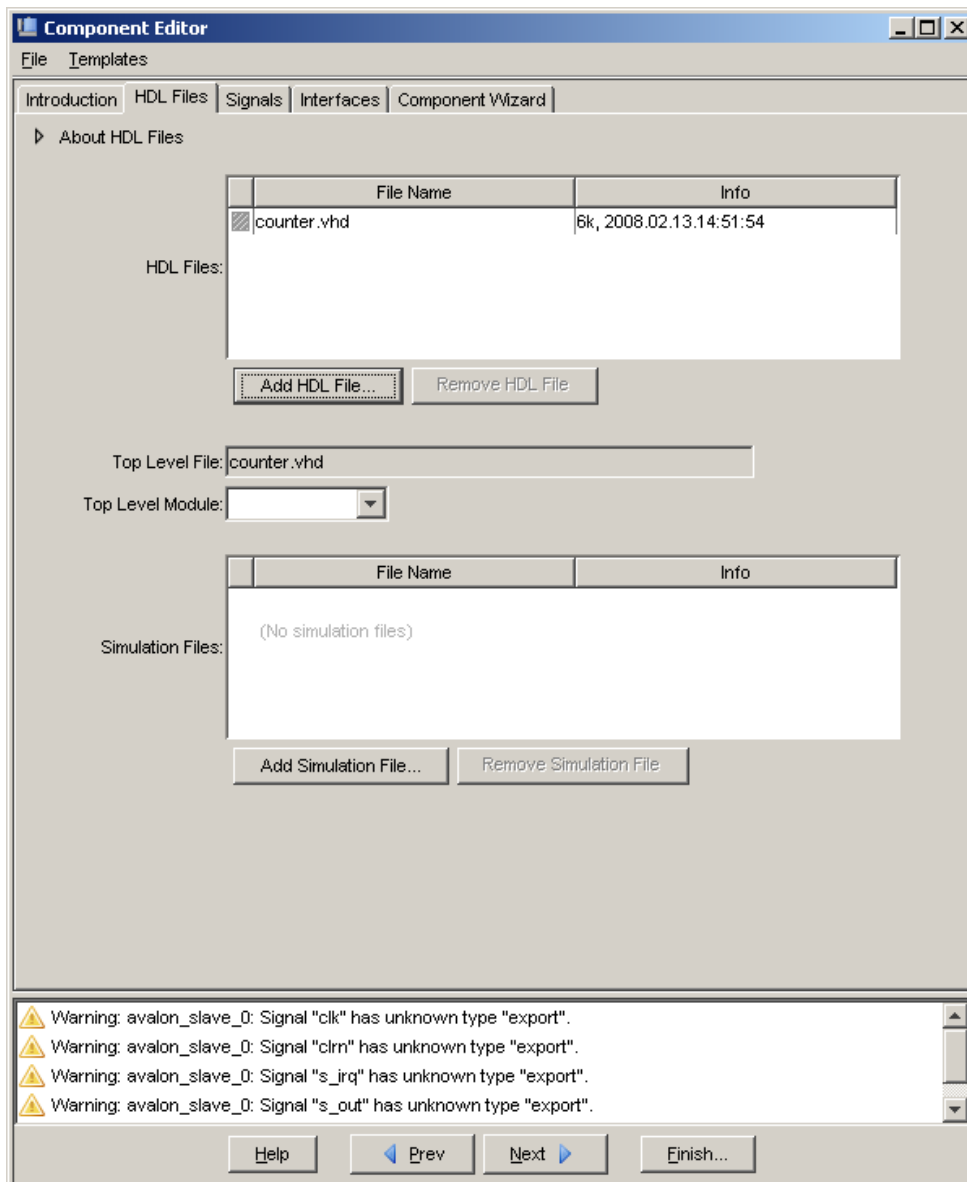


Figure 89: HDL Files tab after adding the counter.vhd file

- Click *Next* to proceed to the *Signals* tab sheet which allows you to assign the signals found in the top-level module of the component to one or more interfaces.
- To configure the signals left click on the cell you want to edit and choose the correct entry from the drop-down list. Configure the signals according to the next figure and the following table.

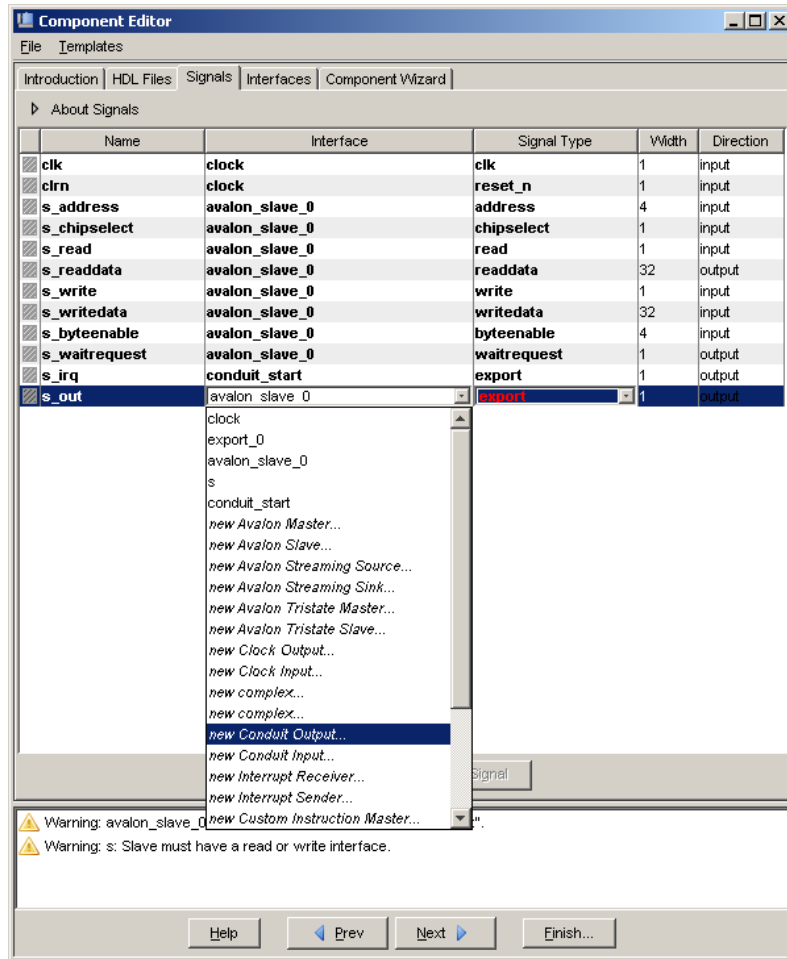


Figure 90: Component Editor – Signals Tab

| Name | Interface | Signal Type | Width | Direction |
|---------------|-----------------|-------------|-------|-----------|
| clk | clock | clk | 1 | input |
| clrn | clock | reset_n | 1 | input |
| s_address | avalon_slave_0 | address | 4 | input |
| s_chipselect | avalon_slave_0 | chipselect | 1 | input |
| s_read | avalon_slave_0 | read | 1 | input |
| s_readdata | avalon_slave_0 | readdata | 32 | output |
| s_write | avalon_slave_0 | write | 1 | input |
| s_writedata | avalon_slave_0 | writedata | 32 | input |
| s_byteenable | avalon_slave_0 | byteenable | 4 | input |
| s_waitrequest | avalon_slave_0 | waitrequest | 1 | output |
| s_irq | conduit start | export | 1 | output |
| s_out | conduit start 1 | export | 1 | output |

Table 4: Signal configuration



Caution: Configure the signals carefully, otherwise the function of Avalon Slave component can not be guaranteed.

- Click *Next* to go to the *Interfaces* tab sheet.

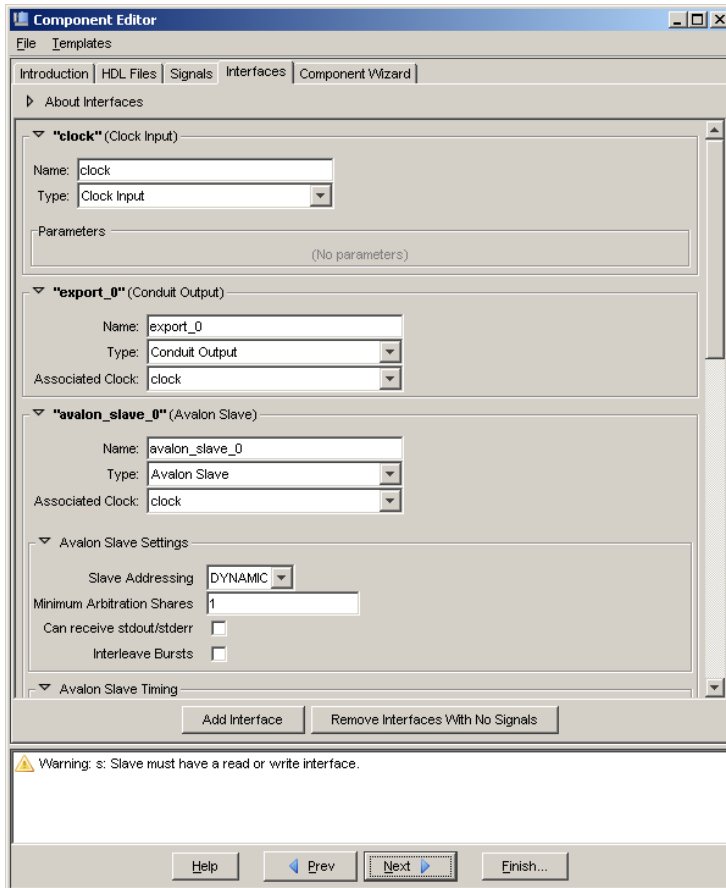


Figure 91: Component Editor –Interfaces tab

- Remove any unused interfaces by clicking the *Remove Interfaces with No Signals* button. This removes the default interfaces which were generated automatically by the component editor when you added the HDL file and which are not necessary, as you created your own interfaces with the automatic type and interface recognition feature.
- Ensure that the *Interfaces* tab looks the same as in the next figure, before you click *Next* to proceed to the *Component Wizard Tab*.

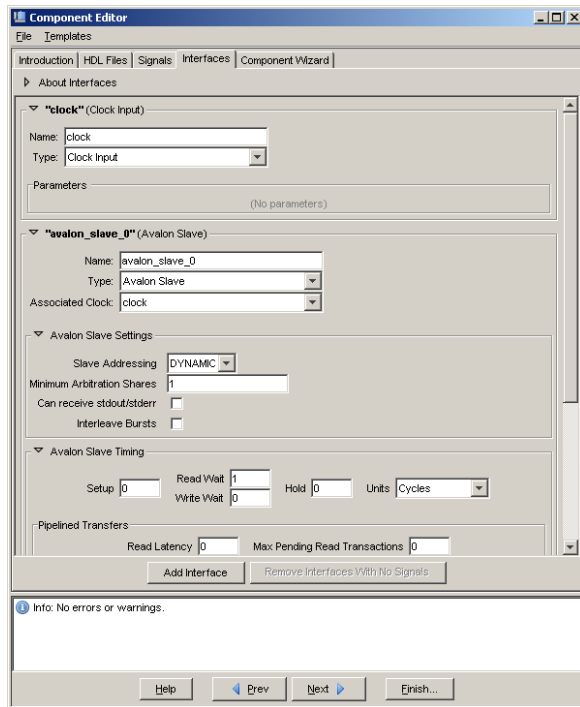


Figure 92: Interfaces tab after removing Interfaces with No Signals

- For the Component Group select PHYTEC from the drop-down list.

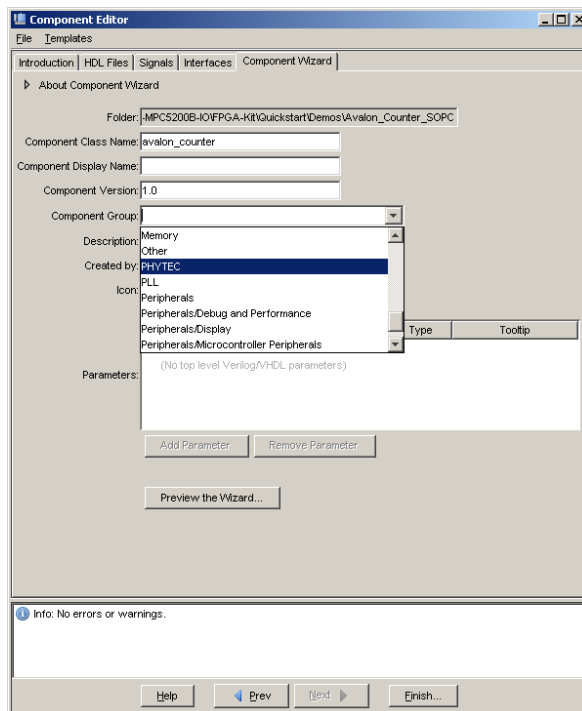



Figure 93: Component Editor – Component Wizard

- Click *Finish* to close the Component Editor and confirm saving the changes by pressing *Yes, Save*.

At this point, the new component is ready to instantiate in an SOPC Builder system.

- Expand the component group PHYTEC in the Component Group list by clicking the  icon. The new component *avalon_counter* should be the first component of the PHYTEC group.

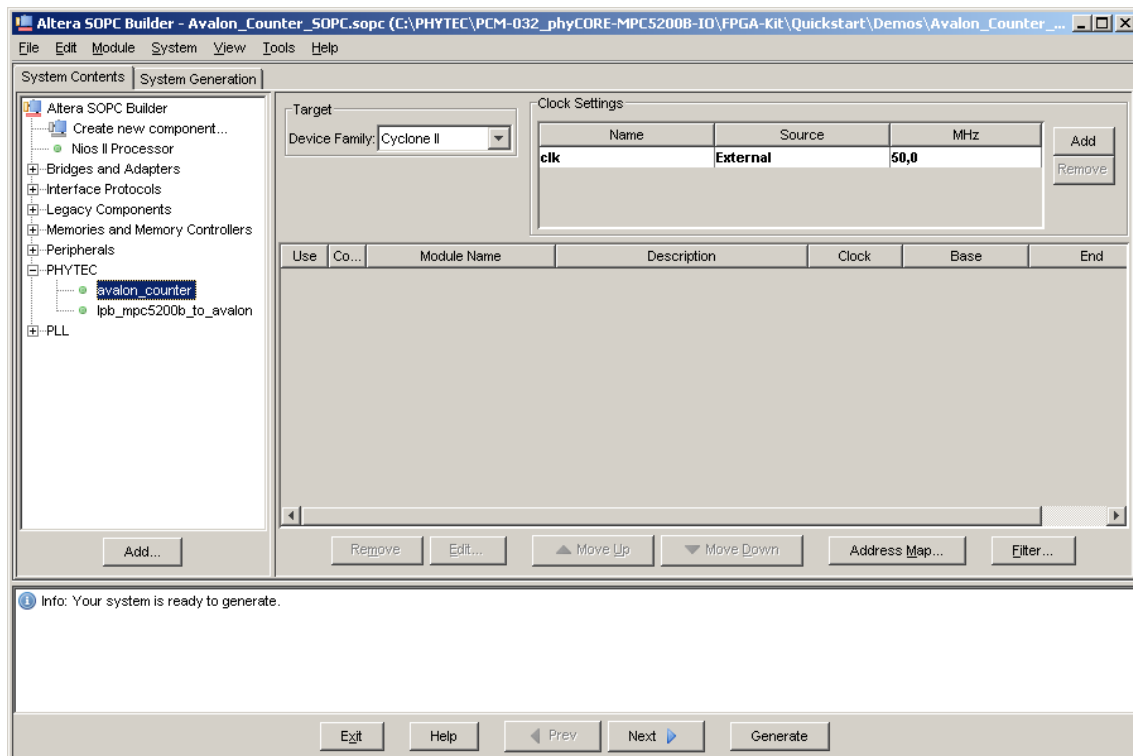


Figure 94: New component *avalon_counter* in the *System Contents* tab

Now we can create the *Avalon_Counter_SOPC* project. All the necessary components for the creation with the SOPC Builder are now available.



In this section, you learned to add a new Avalon Slave component to the SOPC Builder and with this making it available to other projects.

3.3.2 Creating a project with the SOPC Builder

To create a system with the SOPC Builder the components have to be specified, i.e., added to the project, configured and connected together. If necessary clock domains must be created and must be provided to the components.

First we will add the components to the *Avalon_Counter_SOPC* project. The first component will be *lpb_mpc5200b_to_avalon* (the MPC5200B's Local Plus Bus interface).

- Select *lpb_mpc5200b_to_avalon* from the Component Group list and click *Add* to add it to the *Avalon_Counter_SOPC* project or double click on this component.

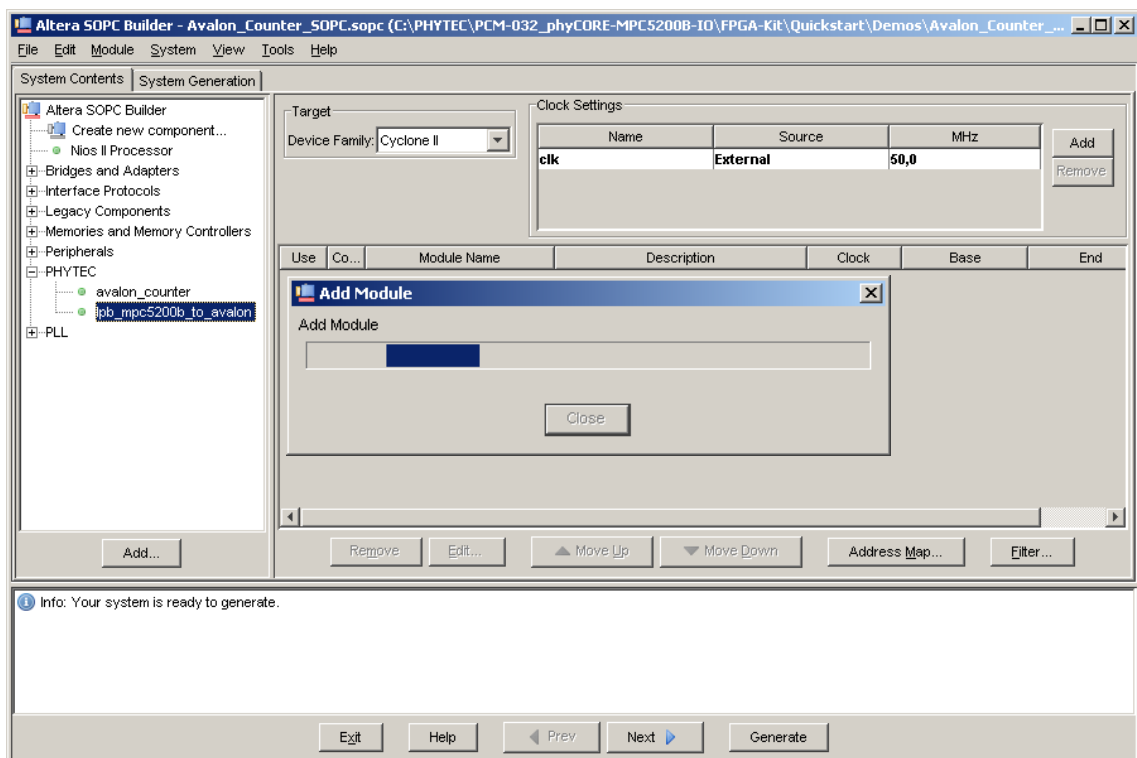


Figure 95: Adding *lpb_mpc5200b_to_avalon* to the *Avalon_Counter_SOPC* project

Before the component will be added to the project a configuration wizard specific for each component will be started. This wizard (so called *component name* MegaWizard) allows to configure the component.

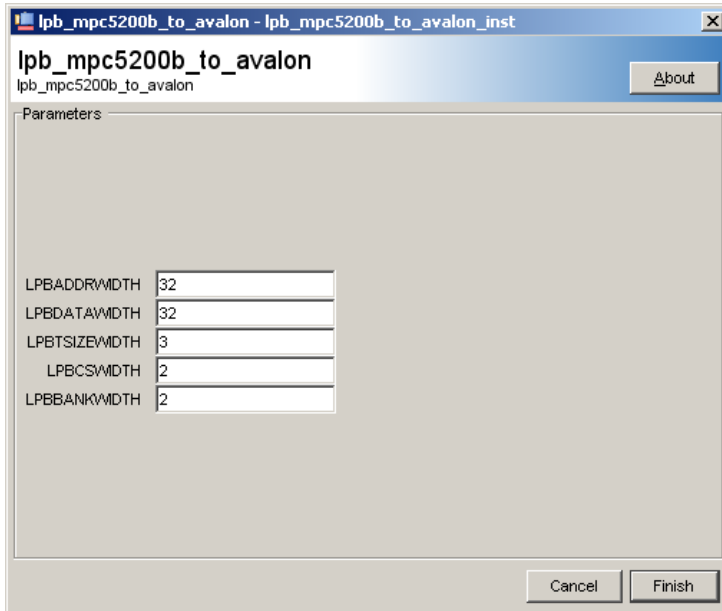


Figure 96: MegaWizard for the *lpb_mpc5200b_to_avalon* component

- Click *Finish* to close the wizard without any changes.

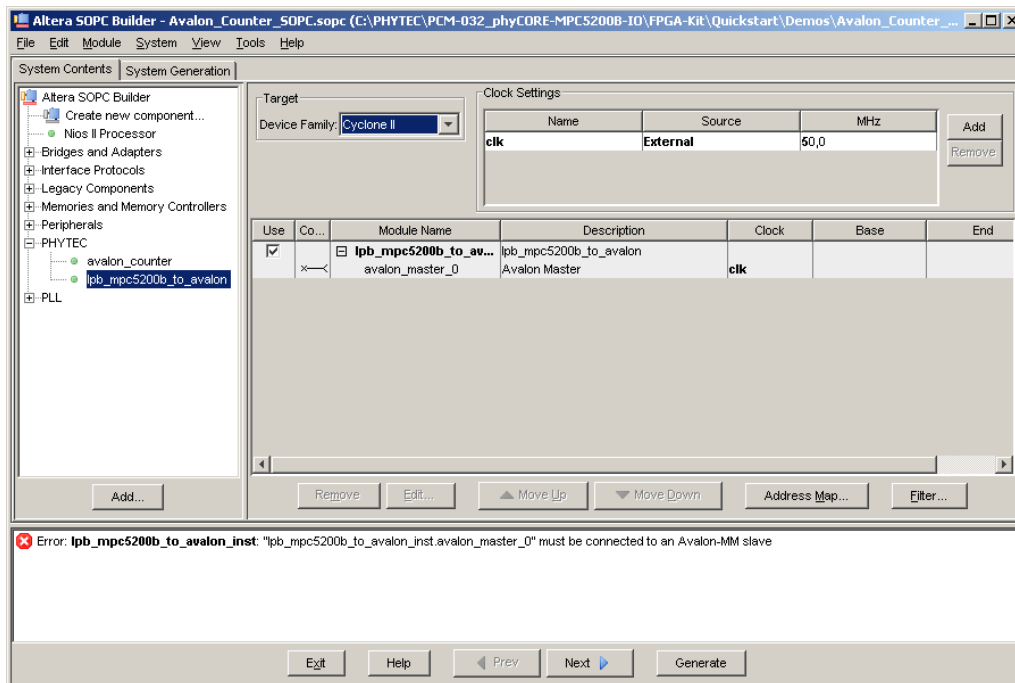


Figure 97: System Contents tab with *lpb_mpc5200b_to_avalon* component

The `lpb_mpc5200b_to_avalon` component is now added to the list of active components.

- Now add the `avalon_counter` component.
- In the `avalon_counter` MegaWizard no further configurations are available for this component. Simply click *Finish* to close the window.

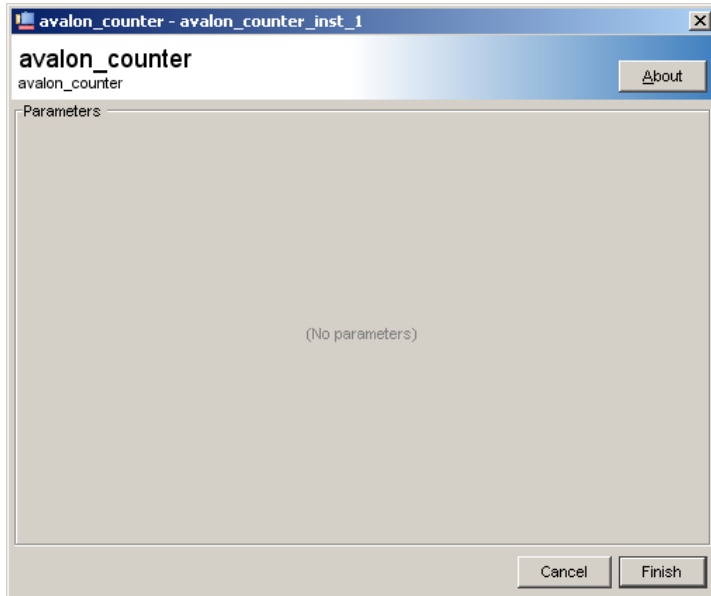


Figure 98: MegaWizard for the `avalon_counter` component

Before we add a clock domain as the last component to the project we have to configure the external clock source.

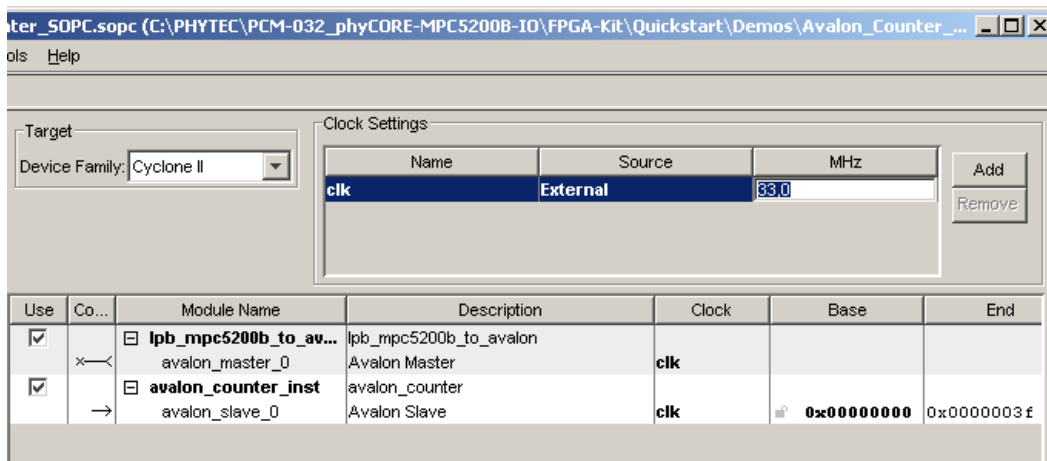


Figure 99: Configuring the external clock source

- Double click on the cell in the MHz column of the Clock Settings table and enter **33** for the clock frequency of the MPC5200B's Local Plus Bus.
- Now expand the PLL group in the Component Group list by clicking the **+** icon.
- Select the PLL and click on the *Add* button or double click on PLL.

Now the *Altera PLL Configuration Wizard* window opens.

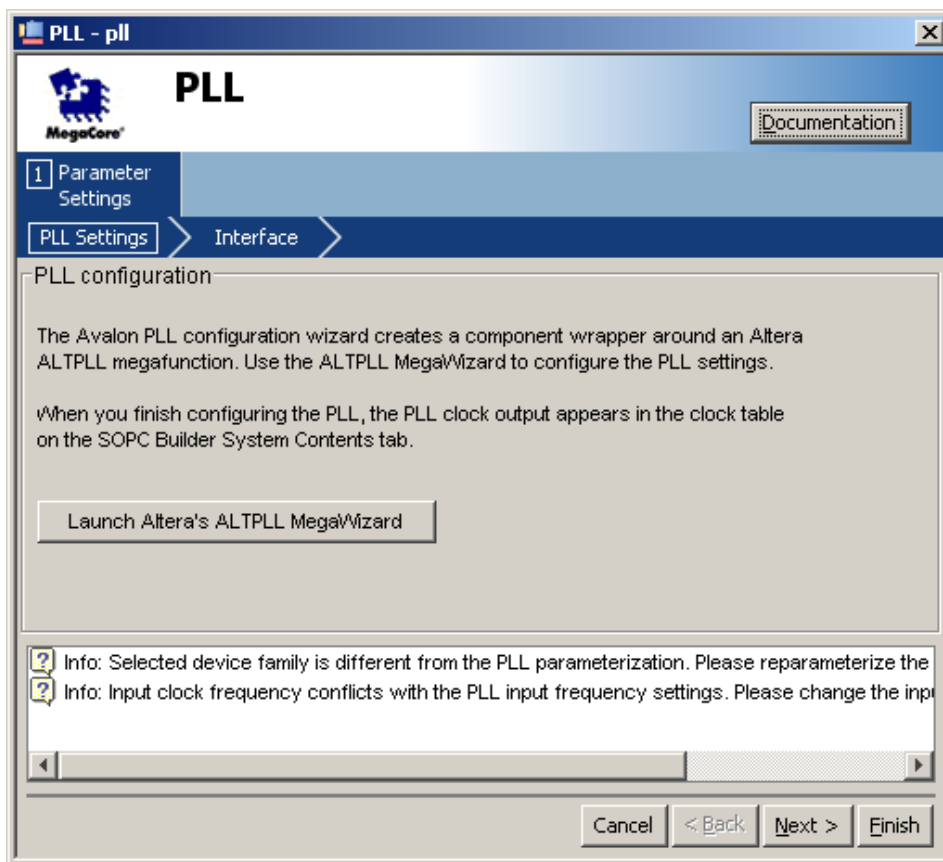


Figure 100: *Altera PLL Configuration Wizard*

- Click on the  *Launch Altera's ALTPLL MegaWizard* button.

The first window of Altera's ALTPLL MegaWizard opens, as shown in the next figure.

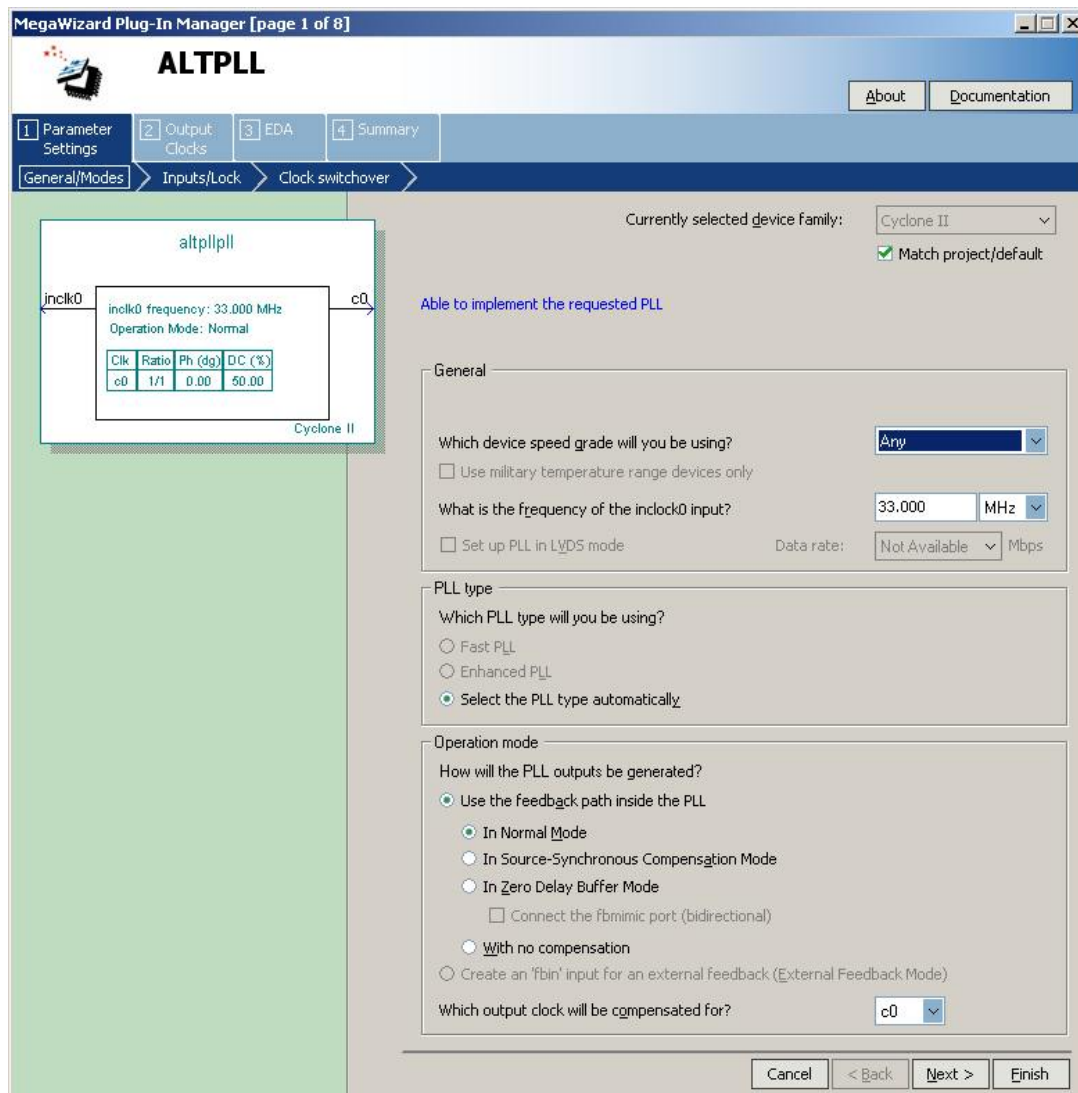


Figure 101: Parameter Settings window of Altera's ALTPLL MegaWizard

- Ensure that the clock frequency is 33 MHz.
- There is no configuration necessary for the PLL. Please click the *Next* button seven times to reach the last window of the configuration wizard.

Use of the *ALTPLL MegaWizard* is mandatory to generate the files needed for the PLL. These files are listed in the last window of the configuration wizard, as shown in the next figure

- Complete the configuration of the PLL by clicking on *Finish*.

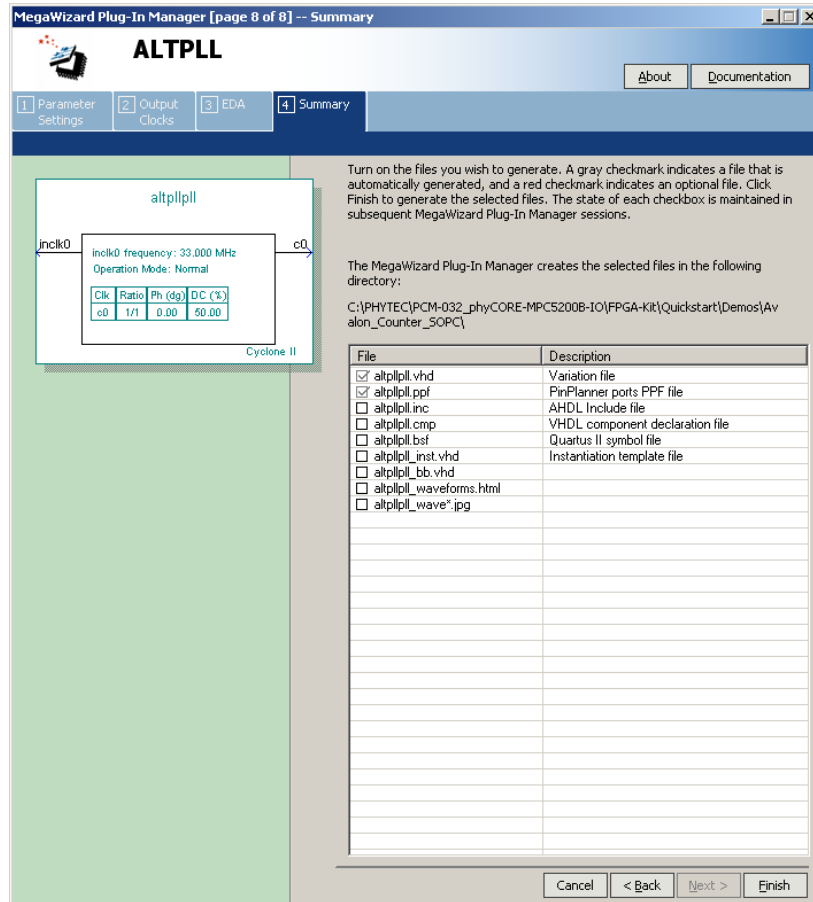


Figure 102: List of files to be generated by the ALTPLL MegaWizard

- Click on *Finish* again to close the PLL Configuration Wizard and to return to the System Contents tab of the SOPC Builder.

In the next figure you can see that the PLL component is added to the list of active components and the new clock domain *pll_c0* appears in the Clock Settings table. This clock will be used as input clock for the avalon components, whereas the previously existing clock signal *clk* represents an external clock signal which will supply the PLL.



The PLL is added to show how to implement multiple clock domains in the SOPC Builder, but is not essential for the functioning of the Avalon_Counter_SOPC project. Supplying the Avalon components only by the external clock signal *clk* is also possible.

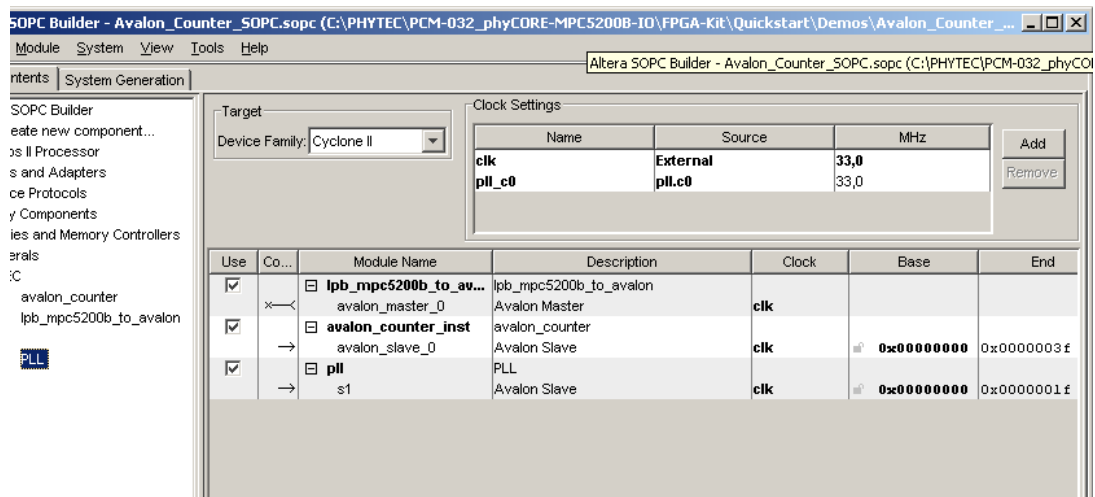


Figure 103: Clock Settings table with the newly generated clock domain *pll_c0*

The last step regarding the clock domains is to specify which clock drives which component.

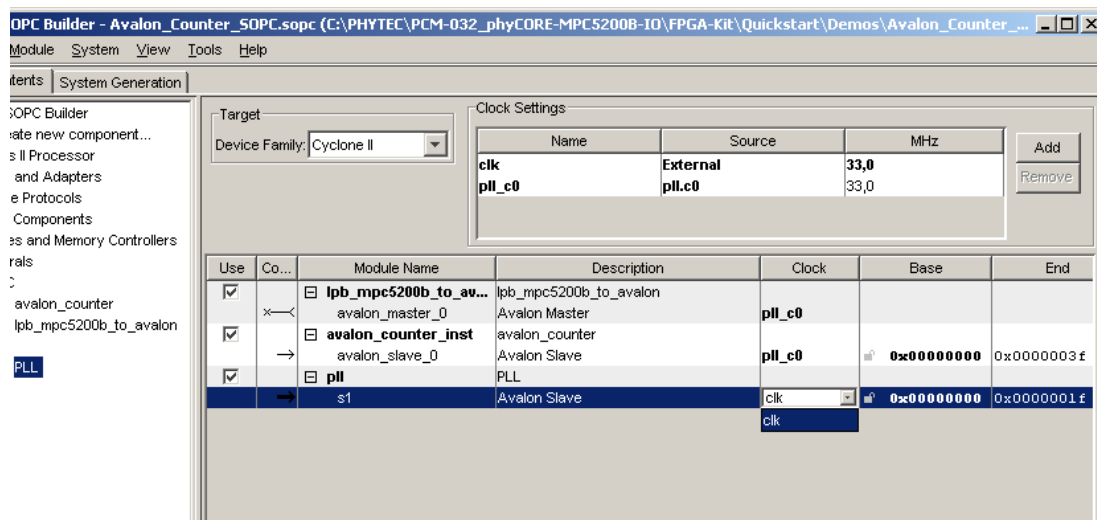


Figure 104: Assigning the clock domains to different components

- Go to the list of active components and click on the clock setting for *lpb_mpc5200b_to_avalon*.
- Select *pll_c0* from the drop-down list.
- Proceed the same way for the other components till the list of active components appears as shown in the above figure.

Now we are ready to connect the three components in the list of active components.

- Move the cursor to the *connections panel* in the System Contents tab.

As shown in the figure below connections between the components will appear.

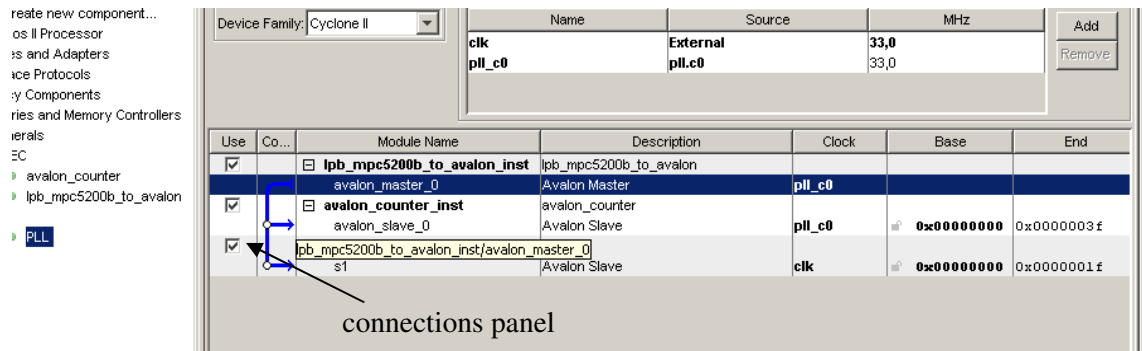


Figure 105: Possible connections shown in the connection panel

As you can recognize the circles at the junctions are not checked yet.

- Left click on the first junction to connect the two components *lpb_mpc5200b_to_avalon_inst* and *avalon_counter_inst*.
- Continue with connecting *avalon_counter_inst* with the *pll*.

The list of available components should now look identical to the next figure.

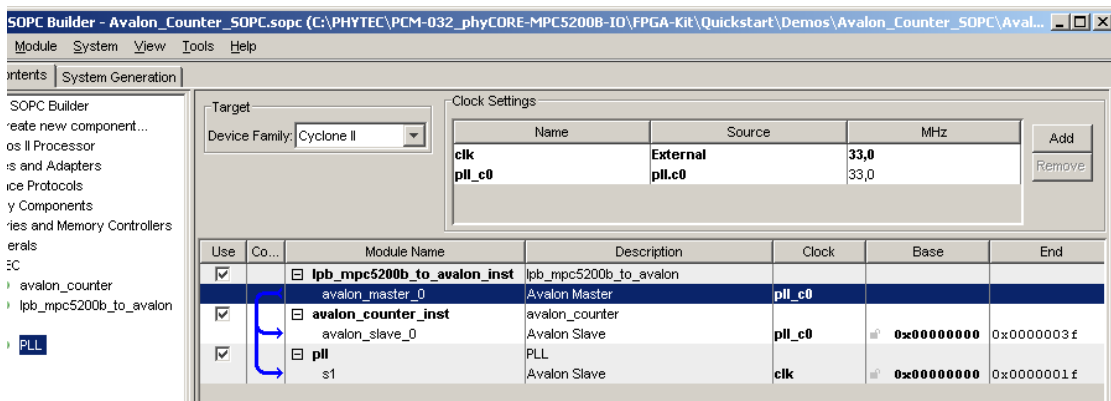


Figure 106: Connection panel with connections between the components

Please note that two of the three errors shown in the message window of the SOPC Builder disappeared after the components were connected.

The final step before we can generate the top-level entity of the Avalon_Counter_SOPC project with the SOPC Builder is to configure the address decoding.

As you can see the base addresses of the *avalon_counter_inst* instance and the pll instance are the same, which means that they overlap. This is also indicated by the remaining error message in the message window. The size of the *avalon_counter_inst* is 0x3F so the pll instance should start at 0x40.

- Double click on the Base settings for the pll and enter **0x00000040**.

The System Contents tab should now contain no further errors.

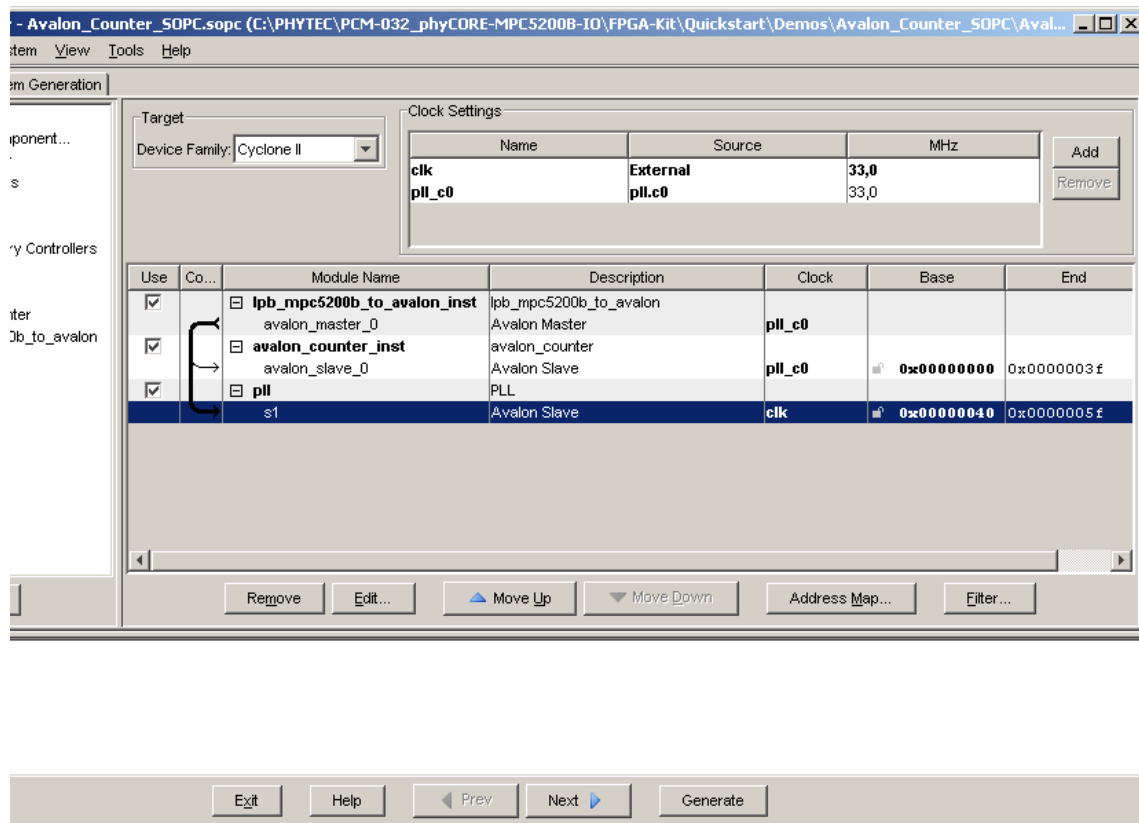


Figure 107: System Contents tab with complete system configuration for the Avalon_Counter_SOPC project

- Open the *System Generation* tab by clicking the *Next* button, or by clicking the appropriate tab.

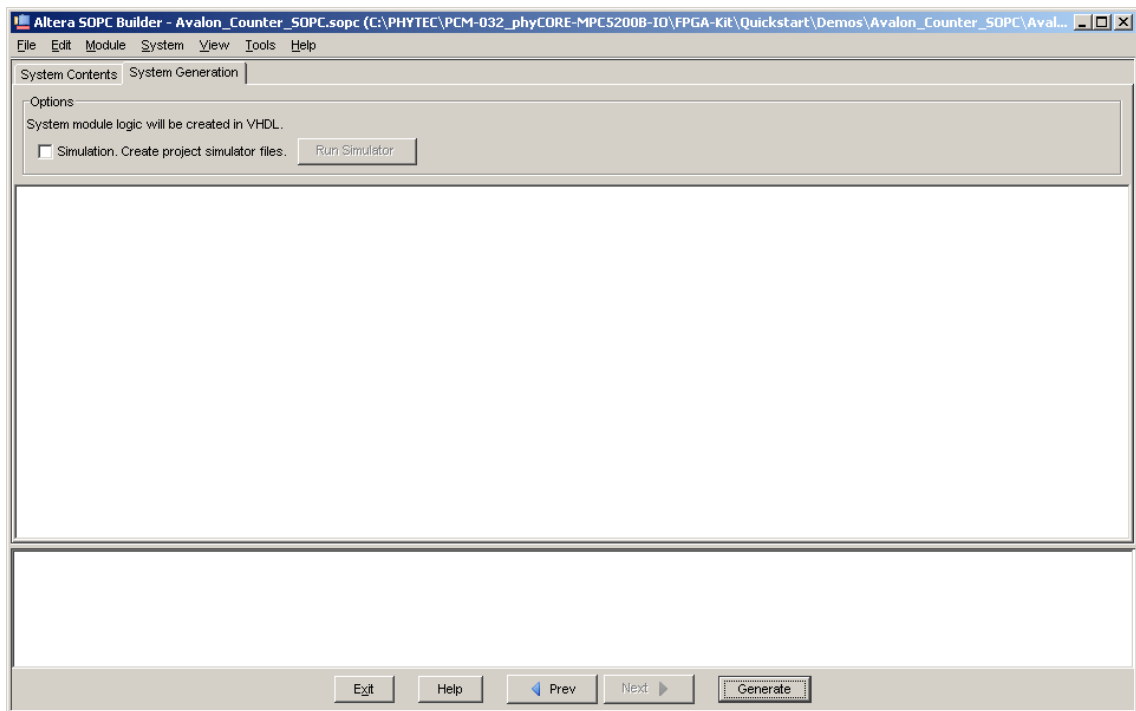


Figure 108: *System Generation* tab

- Click *Generate* to start the generation.
- Confirm saving of the system.

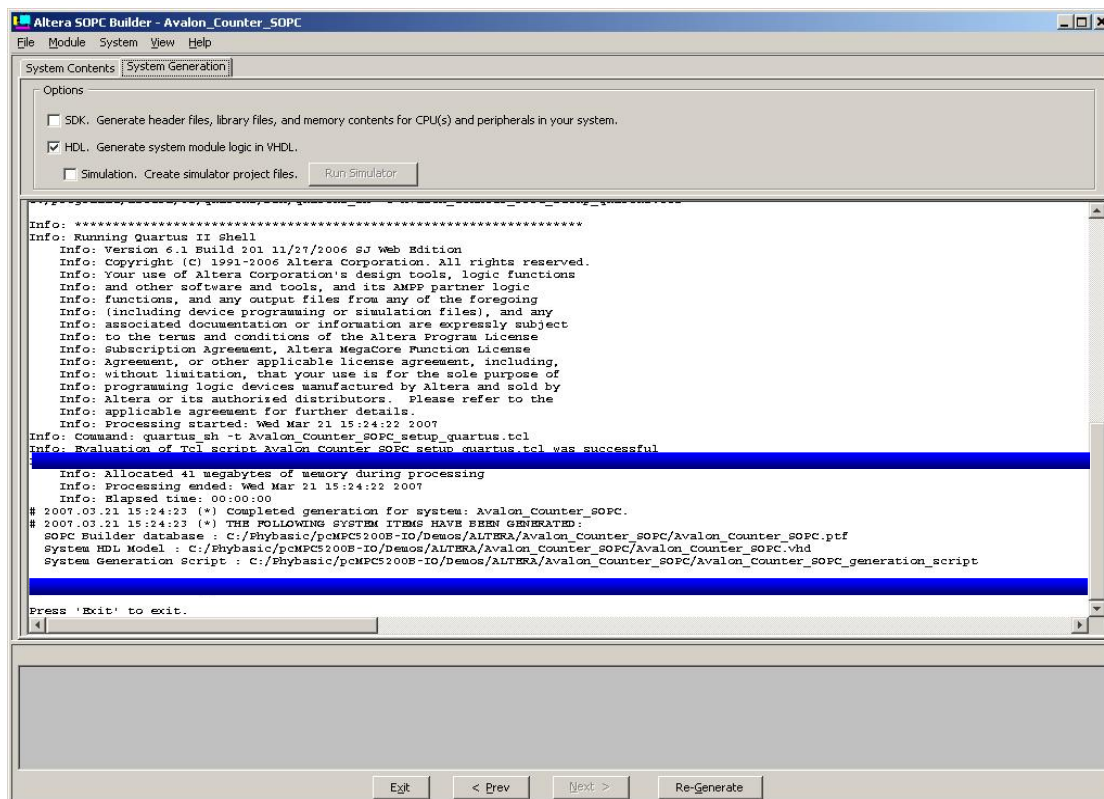




Figure 109: System Generation tab after successful generation

During the generation of the system the SOPC builder will give you lots of information about the single steps of the process. When the system generation is finished the System Generation tab of the SOPC Builder should indicate 0 errors and 0 warnings and should appear as shown in the above figure. All VHDL files which are necessary for the Avalon_Counter_SOPC project are available now.

- Click on *Exit* to close the SOPC Builder and to return to the main window of the Quartus® II tool chain.

After all VHDL files are generated from the SOPC Builder we have to configure the Avalon_Counter_SOPC project the same way as we did with the Avalon_Counter project.

First we will add all necessary files to the project.

- Now click on the icon *Settings*  to enter additional project settings.
- Select *Files* from the Category list to add the files necessary for the project.
- Click the *Browse* button  to open the *Select File* dialog box.
- Enter ***.vhd** in the field *File Name* and press <Enter>

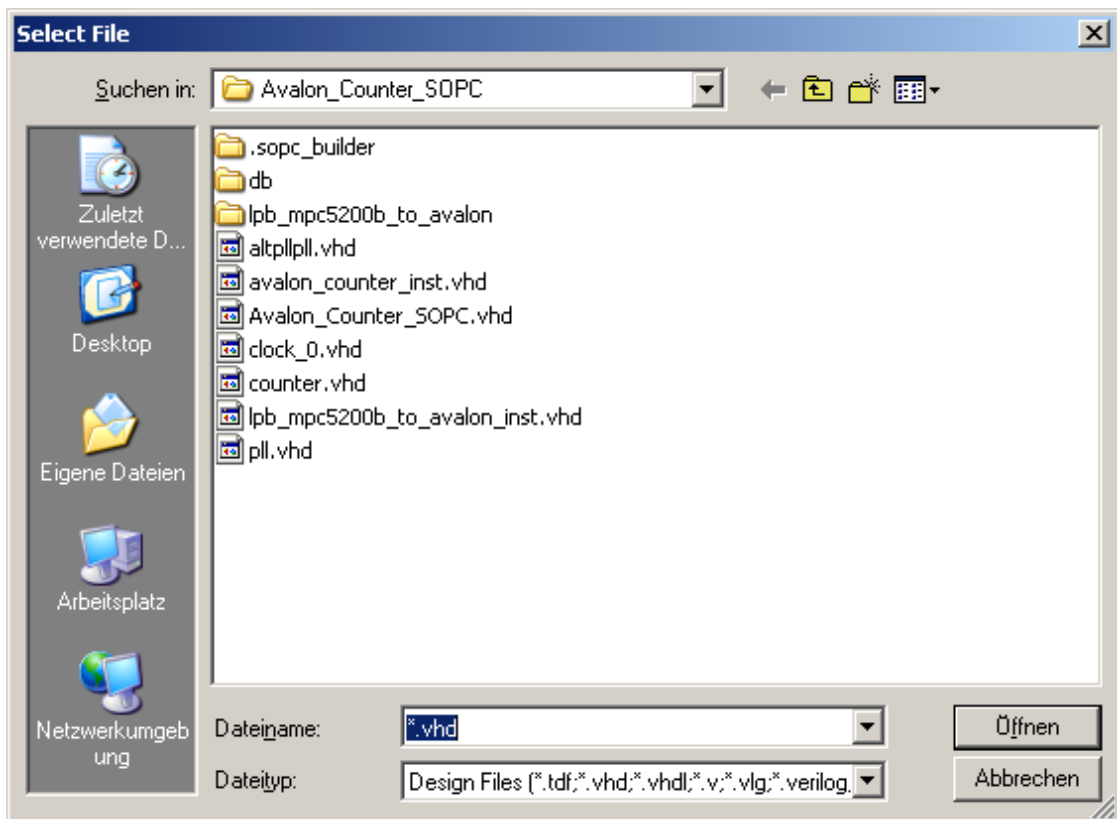



Figure 110: *Select File* dialog box with all available VHDL files

- Now go to the working directory of the project and mark all visible VHDL-files except *counter.vhd* and click *Open* to add them to the project.

The *counter.vhd* file will be added to the project by means of a script file which was generated by the SOPC Builder. However accidental adding the *counter.vhd* file together with the other files wouldn't be prejudicial.

- Again click the *Browse* button  to open the *Select File* dialog box.
- Enter **.qip* in the field *File Name* and press <Enter> or select *Script Files (*.tcl,*.sdc,*.qip)* from the *File Type* drop-down list.
- Select the file *Avalon_Counter_SOPC.qip*
- Click *Open* and then click *Add* to add the selected file to the project.

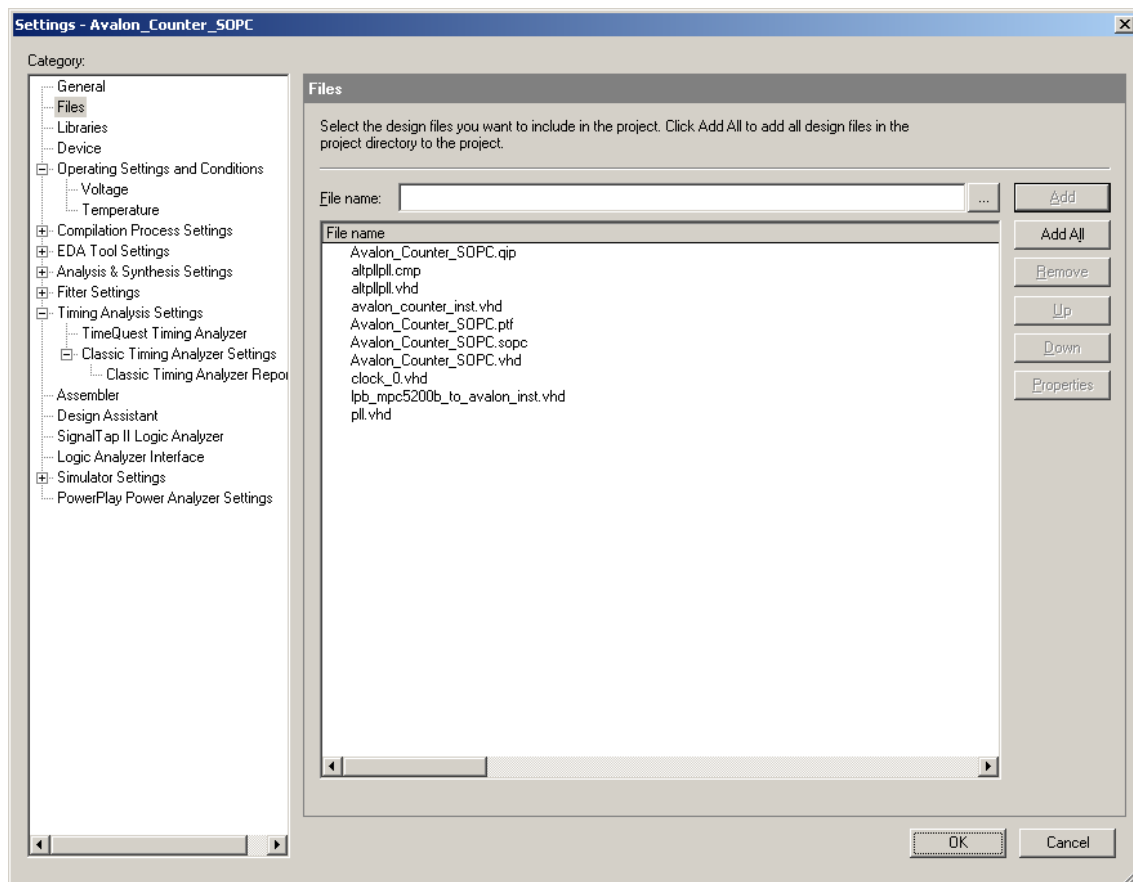



Figure 111: Files list with all necessary files for the *Avalon_Counter_SOPC* project

- Click *OK* to close the *Settings* dialog box.

- Click on the  *Start Analysis & Synthesis* icon to start the analysis and to test the implementation of the new code.
- Please re-examine the implementation if the analysis does not result in the message shown in the following figure.

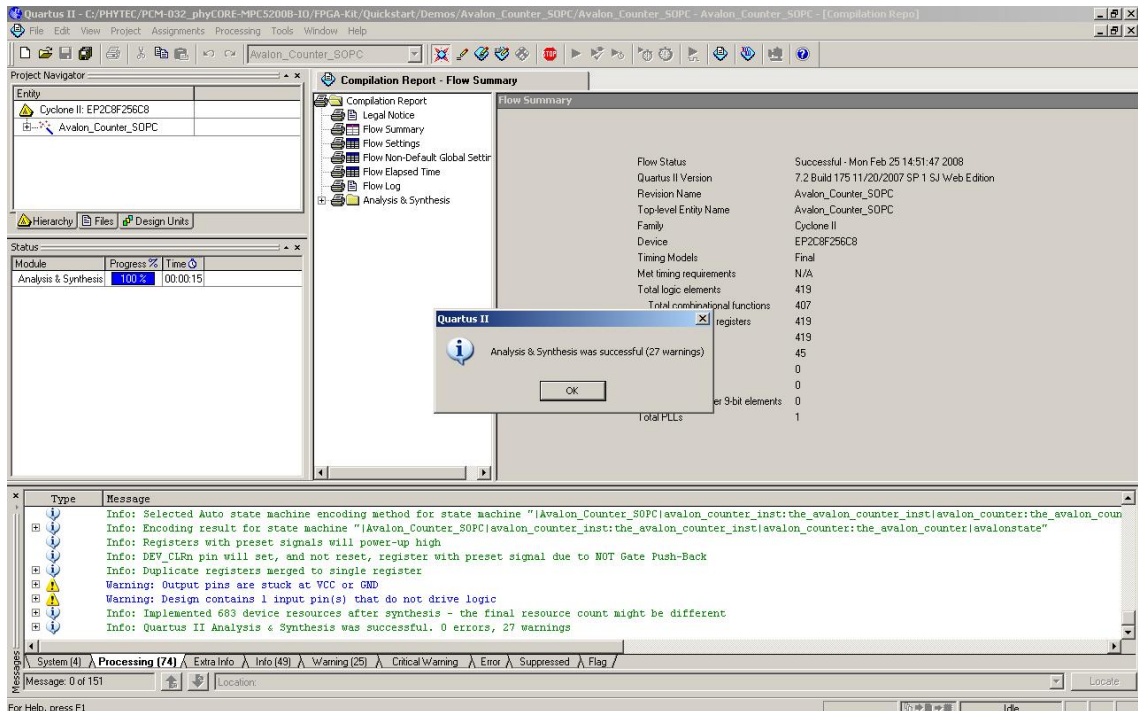


Figure 112: Result of the analysis and synthesis process of the *Avalon_Counter_SOPC* project

When carrying out the synthesis, the Quartus® II tool chain generates 27 warnings. However, this is not a problem for further execution of the project. You can examine the warnings in the *Warning* tab.

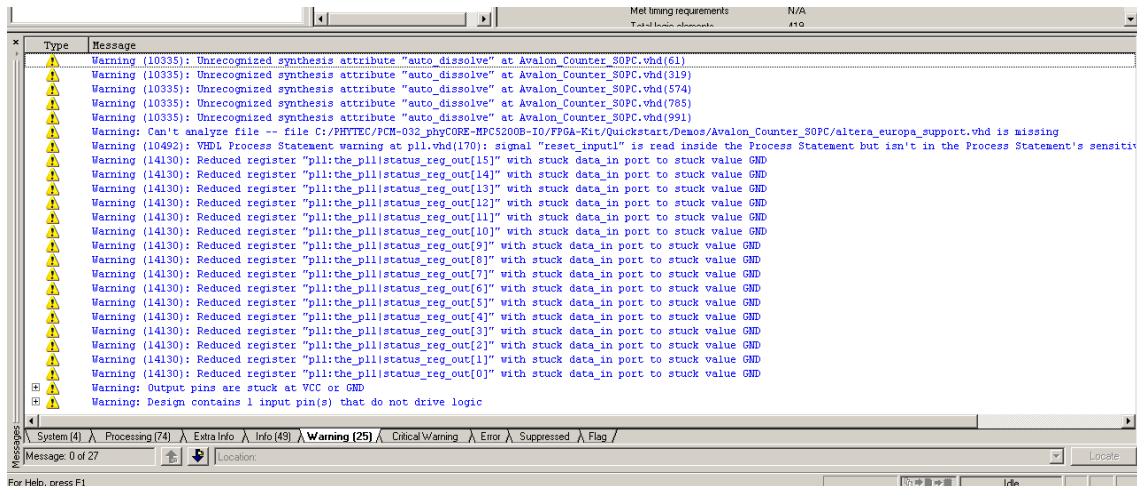



Figure 113: Messages in the Warning tab

In a final step we have to configure the assignments for the Avalon_Counter_SOPC project. As the pin names created by the SOPC Builder are very long and difficult to enter you will learn two alternative ways for entering the signal names without typing.

- Start the Assignment Editor by selecting *Assignment Editor* in the *Assignments* menu or clicking the *Assignment Editor* icon .
- To define the functions of the FPGA's IOs select *Pin* from the category *Locations* within the category bar.


As you already performed an analysis all signal names are known to the Assignment Editor.

- Double click on the word *new* in the first column (*TO* column) of the spreadsheet.
- Now select the signal you want to enter from a drop-down list.
- Next you have to specify the location for the signal you just entered. Double click on the cell in the *Location* column next to the name you have just entered. From the drop-down list select the correct pin according to *Table 5*. You can also start

typing the pin number and let the Assignment Editor automatically complete it for you.

- Continue until all you entered all pins as listed in *Table 5*.
- When you are finished entering all signals for the project move the cursor to the *I/O standard* column. Press the left mouse button and select the whole column.
- Click on the *Edit* bar on top of the spreadsheet and select 3.3-V LVCMOS from the drop-down list.

In the following you will learn a second way to enter the Assignments.

- Click on the *Compilation* icon  or choose *Start Compilation* from the *Processing* menu.
- After successful compilation select *Back-Annotate Assignments ..* from the *Assignments* menu which opens the Back-Annotate Assignments dialog box.

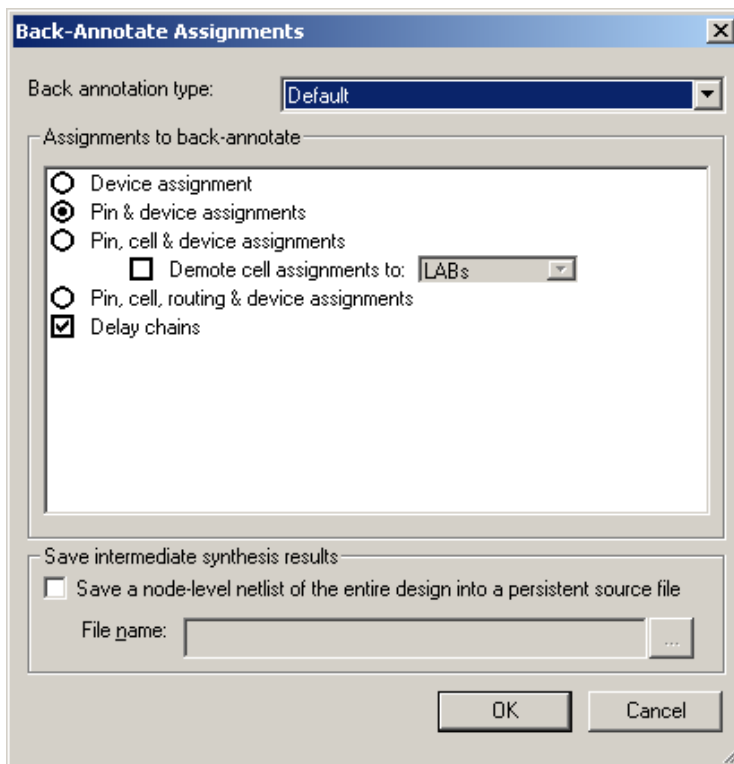



Figure 114: Back-Annotate Assignments dialog box

- Click *OK* without making any changes in the *Back-Annotate Assignments* dialog box, to start the back annotation.
- Return to the Assignment Editor, or Start the Assignment Editor by selecting *Assignment Editor* in the *Assignments* menu or clicking the *Assignment Editor* icon  if not already open.

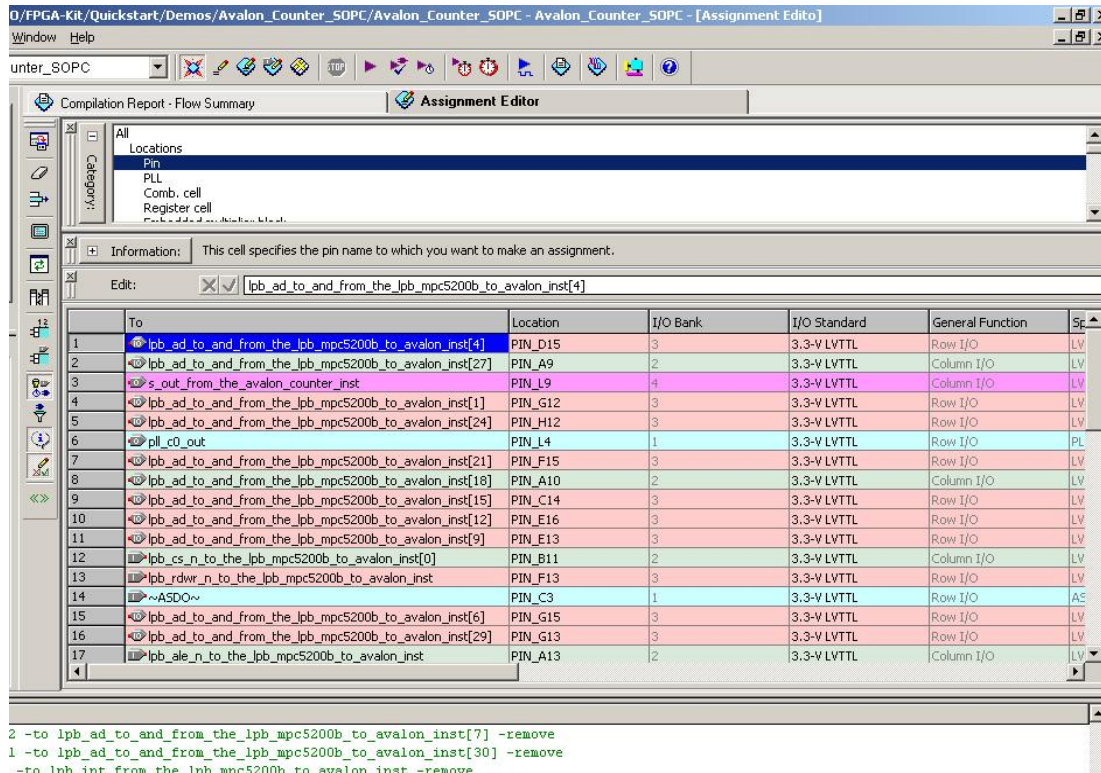


Figure 115: Assignment Editor after back annotation

- It is advisable to sort the signals in alphabetic order to make it easier to compare the spreadsheet with *Table 5*. Click on the title of the first column (*To* column). Now the signals should appear in alphabetic order meaning that *clk* is the first signal of the spreadsheet.
- Next you have to correct the location for the signals. Double click on the cell in the *Location* column next to the name you have just entered. From the drop-down list select the correct pin according to *Table 5*. You can also start

typing the pin number and let the Assignment Editor automatically complete it for you.

- Continue until all you corrected the locations for all pins as listed in *Table 5*.

Use of the Back-Annotate function results in three more signals (starting with a tilde). This signals are of no importance and you don't have to change anything in the corresponding assignments.

- When you are finished entering all signals for the project move the cursor to the *I/O standard* column. Press the left mouse button and select the whole column.
- Click on the *Edit* bar on top of the spreadsheet and select 3.3-V LVCMOS from the drop-down list.
- After you finished all assignments the Assignment Editor should look as shown in the following figures.

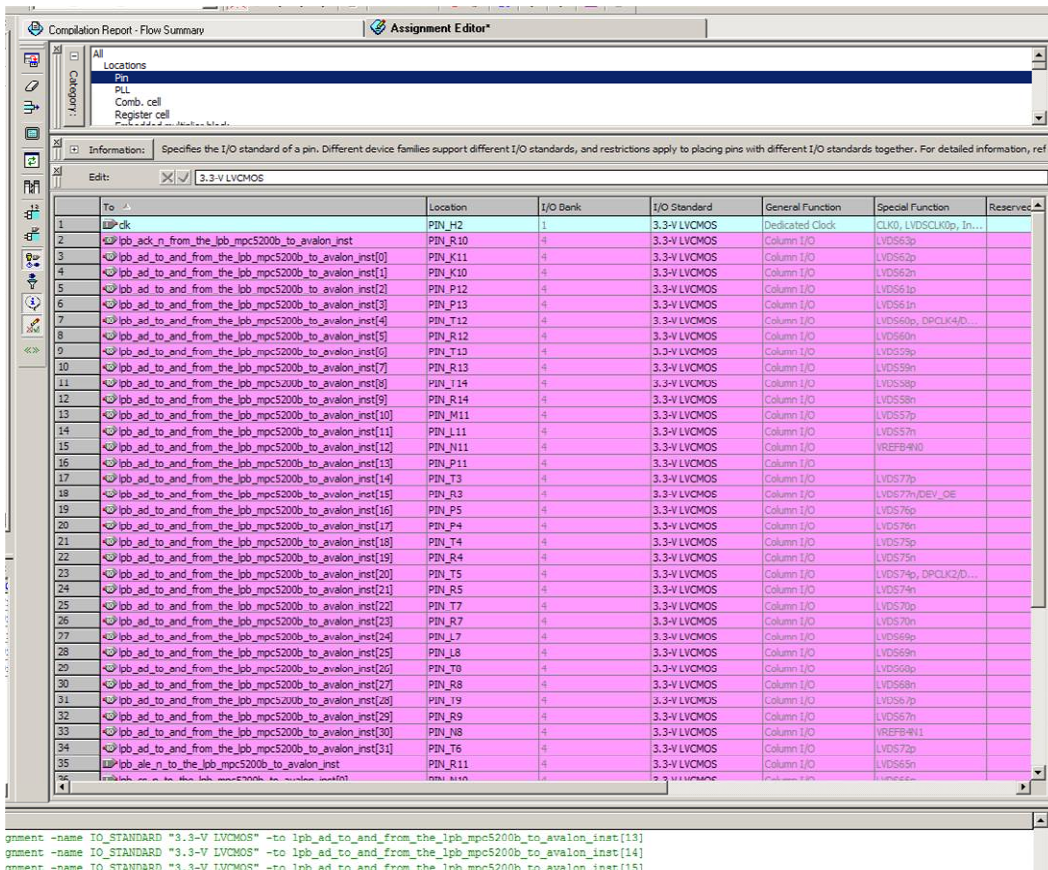


Figure 116: Assignment Editor with completed pin assignments

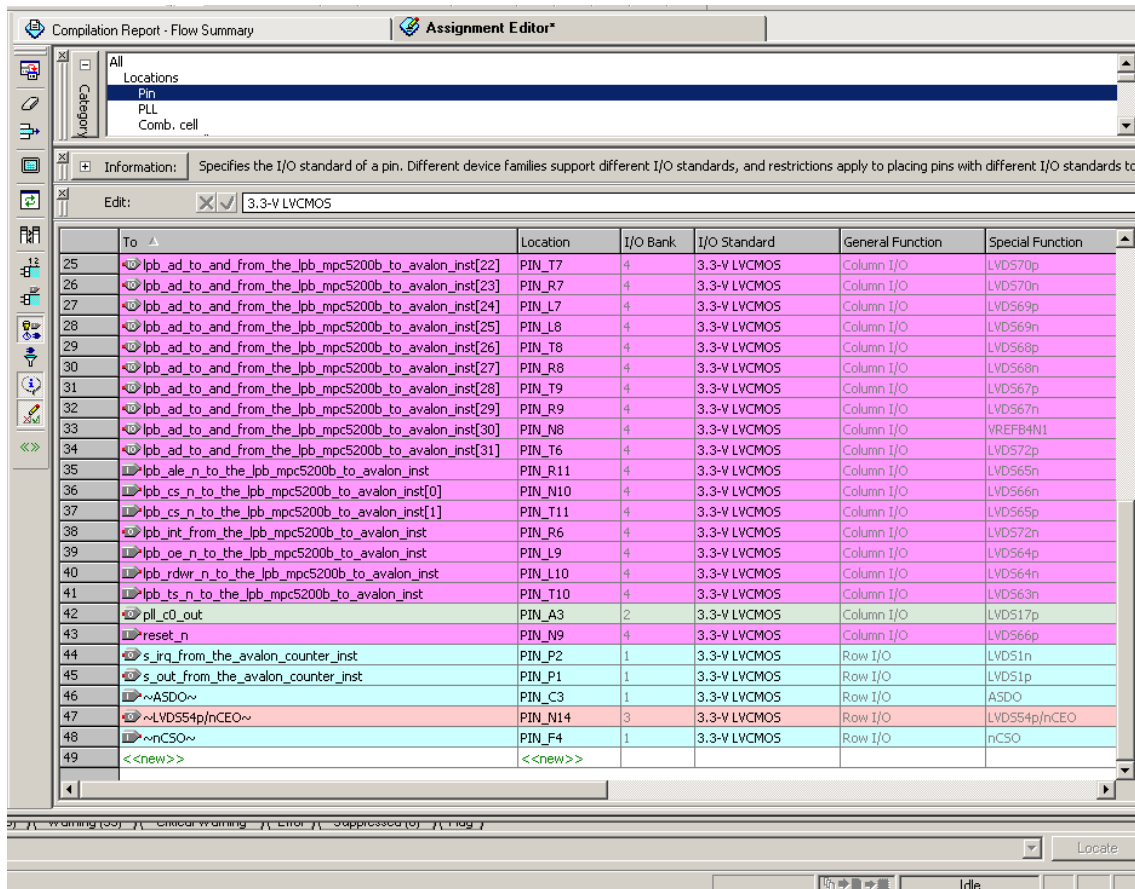


Figure 117: Assignment Editor with completed pin assignments


| TO | Location | I/O Bank | I/O Standard |
|--|----------|----------|--------------|
| clk | PIN_H2 | 1 | 3.3-V LVCMOS |
| lpb_ack_n_from_the_lpb_mpc5200b_to_avalon_inst | PIN_R10 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[0] | PIN_K11 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[1] | PIN_K10 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[2] | PIN_P12 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[3] | PIN_P13 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[4] | PIN_T12 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[5] | PIN_R12 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[6] | PIN_T13 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[7] | PIN_R13 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[8] | PIN_T14 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[9] | PIN_R14 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[10] | PIN_M11 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[11] | PIN_L11 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[12] | PIN_N11 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[13] | PIN_P11 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[14] | PIN_T3 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[15] | PIN_R3 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[16] | PIN_P5 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[17] | PIN_P4 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[18] | PIN_T4 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[19] | PIN_R4 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[20] | PIN_T5 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[21] | PIN_R5 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[22] | PIN_T7 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[23] | PIN_R7 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[24] | PIN_L7 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[25] | PIN_L8 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[26] | PIN_T8 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[27] | PIN_R8 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[28] | PIN_T9 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[29] | PIN_R9 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[30] | PIN_N8 | 4 | 3.3-V LVCMOS |
| lpb_ad_to_and_from_the_lpb_mpc5200b_to_avalon_inst[31] | PIN_T6 | 4 | 3.3-V LVCMOS |
| lpb_ale_n_to_the_lpb_mpc5200b_to_avalon_inst | PIN_R11 | 4 | 3.3-V LVCMOS |
| lpb_cs_n_to_the_lpb_mpc5200b_to_avalon_inst[0] | PIN_N10 | 4 | 3.3-V LVCMOS |
| lpb_cs_n_to_the_lpb_mpc5200b_to_avalon_inst[1] | PIN_T11 | 4 | 3.3-V LVCMOS |
| lpb_int_from_the_lpb_mpc5200b_to_avalon_inst | PIN_R6 | 4 | 3.3-V LVCMOS |
| lpb_oe_n_to_the_lpb_mpc5200b_to_avalon_inst | PIN_L9 | 4 | 3.3-V LVCMOS |
| lpb_rdwr_n_to_the_lpb_mpc5200b_to_avalon_inst | PIN_L10 | 4 | 3.3-V LVCMOS |
| lpb_ts_n_to_the_lpb_mpc5200b_to_avalon_inst | PIN_T10 | 4 | 3.3-V LVCMOS |
| pll_c0_out | PIN_A3 | 2 | 3.3-V LVCMOS |
| reset_n | PIN_N9 | 4 | 3.3-V LVCMOS |
| s_irq_from_the_avalon_counter_inst | PIN_P2 | 1 | 3.3-V LVCMOS |
| s_out_from_the_avalon_counter_inst | PIN_P1 | 1 | 3.3-V LVCMOS |

Table 5: Pin assignments for the Avalon_Counter_SOPC project

As you already learned in section 3.1.1 we finally have to configure the typical IO-Pin load for the outputs.

- Move the cursor to the first cell of the *To* column. Press the left mouse button and select the whole column, i.e. all signal names, except the ones starting with the tilde.
- Copy the column to the clipboard.
- Now scroll down in the category bar of the Assignment Editor and select *I/O Features*.
- A new spreadsheet opens in the *Editor* window. First click on the word *new* in the second column (*TO* column) of the new spreadsheet.
- Paste the signal names into this column.
- Now move the cursor to the *Assignment Name* column. Press the left mouse button and select the whole column.
- Click on the *Edit* bar on top of the spreadsheet and select *Output Pin Load* from the drop-down list.
- Proceed and move the cursor to the *Value* column. Again press the left mouse button and select the whole column.
- Click on the *Edit* bar on top of the spreadsheet and enter **4**.

The Avalon_Counter_SOPC project is now complete and we can start the compilation.

- Click on the *Compilation* icon  or choose *Start Compilation* from the *Processing* menu.

After successful compilation of the project the following screen should be displayed:

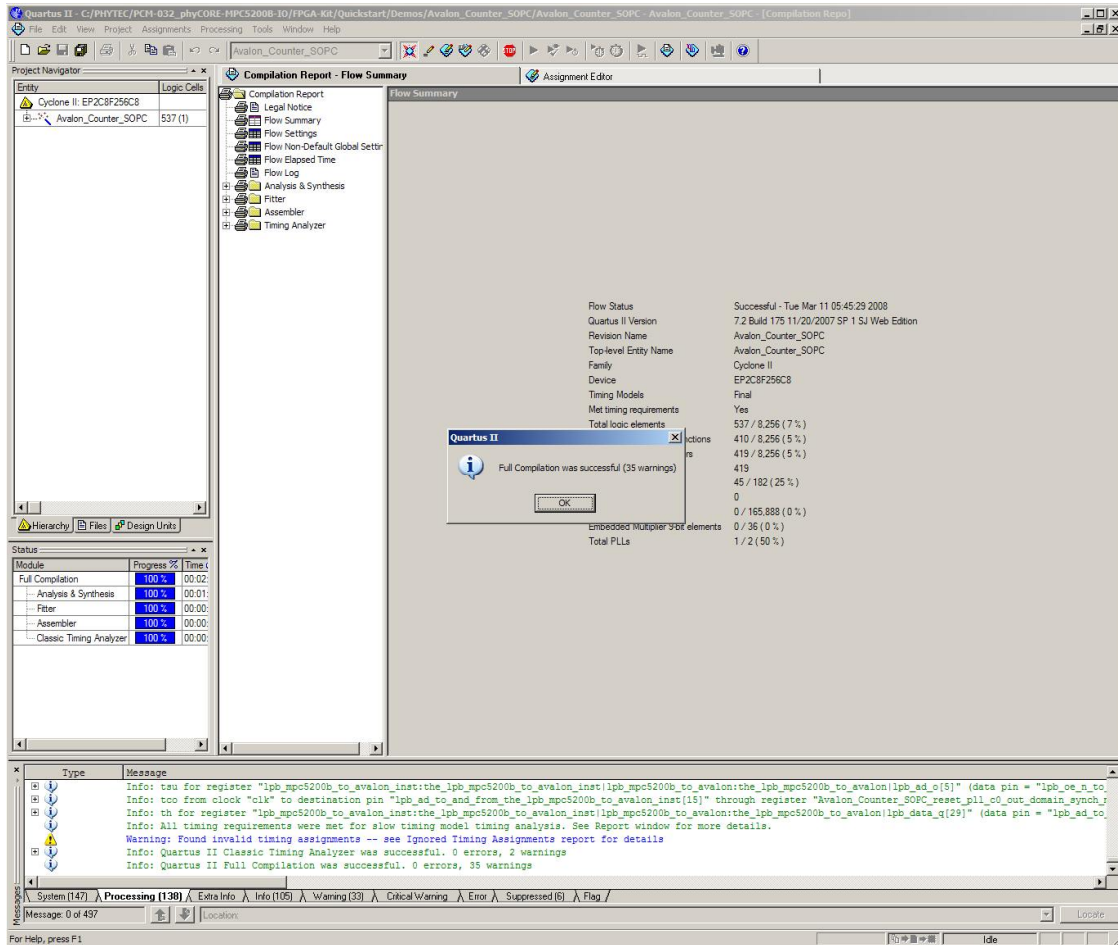


Figure 118 Result of the full compilation process

The 35 warnings you will receive can be viewed in the *Warning* tab of the message window for further evaluation. As they are not critical we can proceed with the download of the new project and the executing on the target.

- Push the *OK* button.

Following the successful compilation, the generated programming file can now be downloaded to the FPGA and tested.

- Open the programmer by choosing *Programmer* from the *Tools* menu.
- Download the program to the FPGA as described in section 3.2.5.2.

Once programming has been completed the *Avalon_Counter_SOPC* project can be tested as described in section 3.2.5.3.



You have successfully created your first own project with the SOPC Builder IDE. You have configured the project to create an executable program for the FPGA on your target platform.

4 FPGA support under Linux

The phyCORE-MPC5200B is shipped with an Altera FPGA of type Cyclone II EP2C8F256C8N. This FPGA is a general purpose device with no special dedication when shipped. Its up to you to blow life in it with your own firmware, for example to do some high speed signal processing. The OSELAS.BSP-phyCORE-MPC5200B provides a mechanism to load the FPGA firmware while the whole system is already running.

4.1 General

The source archive of the OSELAS.BSP-phyCORE-MPC5200B contains the files *fpga.c* and *Makefile*. During the build process of the toolset, a module named *fpga.ko* is created. Here's how to achieve this: We locate the makefile after unpacking and change the KDIR variable in this makefile to the location of our own kernel source directory and then start compiling. We make sure to include the path to the binary of the crosscompiler in our searchpath. Putting the mechanism to work is as easy as loading the module with parameter `firmware=<filename.rbf>`. The firmware must be in RBF (Altera raw binary format) and stored in the place where the firmware agent of our embedded system will expect it. (It depends on the configuration of our udev daemon. Usually it should be `/lib/firmware/`) In practice, the module is loaded like in this example: `~# insmod fpga.ko firmware=my_own_firmware.rbf`. The module assumes specific hardware connections between processor and FPGA which are shown in the following table. The user does not need to take care about these when using the standard hardware. See phyCORE-MPC5200B's datasheet for FPGA's pin assignment.

4.2 Accessing Peripherals

| FPGA Pin | MPC5200B Pin | Pin Loc. | Pin Name | Pin Conf. |
|-------------|---------------|----------|-------------|-----------|
| nCONFIG | UART6 CTS TTL | PSC6 1 | GPIO WKUP 5 | out |
| CONFIG DONE | UART6 RXD TTL | PSC6 0 | GPIO WKUP 4 | in |
| nSTATUS | GPIO7 | | GPIO WKUP 7 | in |
| DCLK | UART6 RTS TTL | PSC6 3 | GPIO IRDA 1 | out |
| DATA0 | UART6 TXD TTL | PSC6 2 | GPIO IRDA 0 | out |

Table 6: FPGA pin connections

4.3 Demo

The BSP is shipped with a simple demo. To download it into the FPGA you should enter:

```
~# insmod /home/fpga.ko firmware=MPC_to_WBSlave.rbf
```

It supports five 32 bit registers at the base address CS1 (chip select) is using.

- Register 0 at CS1 baseaddress + 0x00
- Register 1 at CS1 baseaddress + 0x04
- Register 2 at CS1 baseaddress + 0x08
- Register 3 at CS1 baseaddress + 0x0C
- Register 4 at CS1 baseaddress + 0x10 (mirrored up to the end of CS1 address space).

In the first step to access the FPGA registers we do not need a device driver. Due to PowerPC architecture maps devices in memory we can use a small tool called memedit for testing. The most important part is, where the physical baseaddress of processor's CS1 is located. Usually we can skip this part, the setup is always the same: Physical address starts at 0xe0000000 and ends at 0xe1ff0000. But in case of trouble, it could be helpful to check if the CS1 is on its right location: To gain access to processor's control registers we use the memedit tool on the target: ~# memedit /dev/mem This tool works interactive, so the next step is to map some physical memory space into the memory of the memedit process.

```
-> map 0xf0000000 0x1000
```

0xf0000000 is the base address of the MBAR area, where all processor and its chipset internal control registers are mapped. Next we read back the CS1 configuration register (see processor's manual for further descriptions about location and meaning of these type of registers).

```
-> md 0x0c 0x10 0000000c: 0000e000 0000e1ff 0000fde0 0000fde7  
.....
```

0x0c is an offset to the MBAR (=0xf0000000) and is the start address of CS1. We read back values 0x0000e000 (=start) and 0x0000e1ff (=end). The values have to be shifted by 16 to get the real physical address. In this case it starts at 0xe0000000 and ends at 0xe1ff0000. Unmap the MBAR area first, to do the next step (do not leave memedit):

-> unmap.

After ensuring the physical address area of CS1 we now can map this area to gain access to the FPGA itself. When our mapping of CS1 differs we have to enter other values here. The default should be:

-> map 0xe0000000 0x1ff0000

Now we can read and write into this area and - if the FPGA is ready to work - we can access its registers.

-> md 0x0 0x20

00000000: 00000000 00000000 00000000 00000000

00000010: 00000000 00000000 00000000 00000000

After system reset everything should be zero. We do some writings to check the registers:

-> mm 0x0 0x10

new values:

0x00000000: 00000010

-> mm 0x4 0x20

new values:

0x00000004: 00000020

-> mm 0x8 0x30

new values:

0x00000008: 00000030

-> mm 0xC 0x40

new values:

0x0000000c: 00000040

-> mm 0x10 0x50

new values:

0x00000010: 00000050

And try to read back the written values:

-> md 0x0 0x20

00000000: 00000010 00000020 00000030 000000400...@

00000010: 00000050 00000050 00000050 00000050 ...P...P...P...P

We see the described behavior of FPGA's default firmware. Four single registers at offset 0x0,0x4,0x8 and 0xC, and a mirrored register until the end of FPGA's area starting at offset 0x10.

5 Further Information

For more information about the V7.2SP1 Quartus® II Web Edition software you can download the associated manuals from the website of ALTERA at:

http://www.altera.com/support/software/q_a/sof-q_a.html

Further information with regard to newer versions of the Quartus® II web edition software can be found at:

<http://www.altera.com/download>

6 Summary

This QuickStart Instruction gave a general "Rapid Development Kit" description, as well as software installation advice and example programs enabling quick out-of-the box start-up of the phyCORE[®]-MPC5200B-I/O in conjunction with the Quartus[®] II V7.2SP1 Web Edition software tools.

In the Getting started section you learned to configure your host to provide a basis for working with the FPGA of your target platform. You installed the Rapid Development Kit software and you learned to copy and run an FPGA program on the targets FPGA.

In the Getting More Involved section you got a step-by-step instruction on how to configure and build a new project, modify the example, create and build new projects and copy output files to the phyCORE–MPC5200B-I/Os FPGA using Quartus[®] II V7.2SP1 Web Edition.

The SOPC Builder part of this QuickStart gave you information on setting up and using the SOPC Builder of the Quartus[®] II V7.2SP1 Web Edition. You learned how to setup new Avalon Components, how to configure components which come together with the Quartus[®] II V7.2SP1 Web Edition Software an how to instantiate and generate a complete SOPC Builder design.

Document: phyCORE-MPC5200B-I/O with FPGA
QuickStart Instructions
Document number: L-702e_1, December 2007

How would you improve this manual?

Did you find any mistakes in this manual? page

Submitted by:

Customer number: _____

Name: _____

Company: _____

Address: _____

Return to:

PHYTEC Technologie Holding AG
Robert-Koch-Str. 39
55129 Mainz
Fax: +49 (6131) 9221-26

Published by

PHYTEC

© PHYTEC Messtechnik GmbH 2007

Ordering No. L-702e_1
Printed in Germany