Dünner Kirchweg 77
32257 Bünde
Germany
www.trenz-electronic.de

Spartan-II Development System

2001-November-6

Tutorial: First Steps with WebPACK ISE

Introduction

This tutorial was created to give the interested engineer a quick hands on experience on the Xilinx' *WebPACK ISE* software suite.

The tutorial provides only a very brief overview of FPGA design methodology without going into too much detail. After completing this tutorial and the first own design steps, it is highly recommended to proceed with further readings, e.g. the various application notes and online manuals.

While concentrating on the basic steps and methods, it is also the goal of this tutorial to cover the complete development cycle. Beginning with design entry, through simulation, synthesis and implementation up to the download to physical hardware all covered in a single, consistent project.

To achieve both of these goals, we decided to design a very simple hexadecimal counter which is implemented on our *TE-XC2S* Spartan-II Development System. All steps required to complete this design are covered by this tutorial. There are no hidden steps or imported and half-completed files. While this limits the possible size of the project (without exceeding a certain time frame) we feel that there is still no better way of learning something than from own experience.

Happy developing!

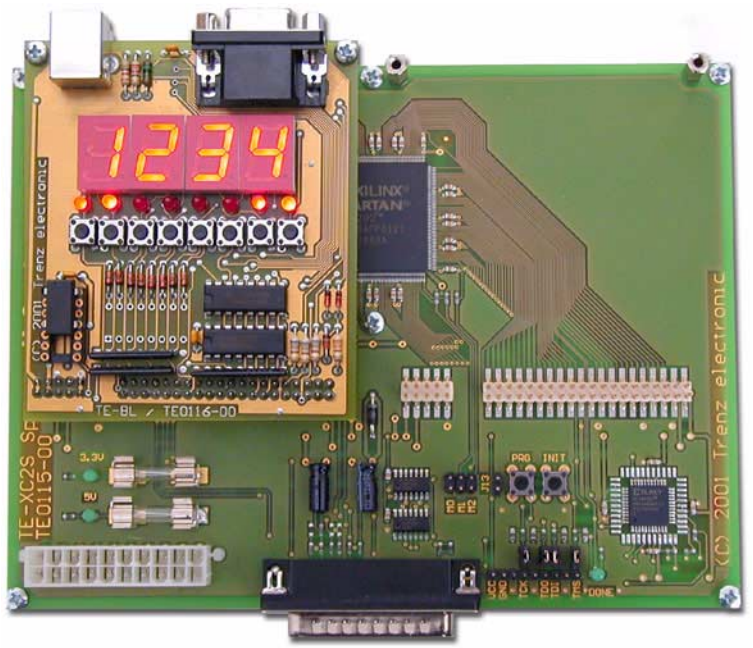


Figure 1: Spartan-II Development System

Download and Installation

In case you did not receive the Xilinx *WebPACK ISE* software on CD-ROM, you may download the latest revision from *www.xilinx.com*. From the home page follow the links to

```
Buy Online
WebPACK
```

If you do not have a Xilinx account yet, you will need to register with Xilinx and create and account. To do so, follow the link to

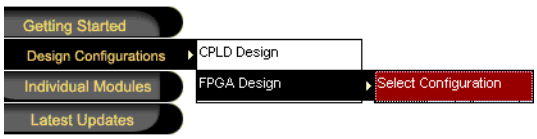
```
Register for WebPACK ISE
```

In case you already registered with Xilinx, follow the link to

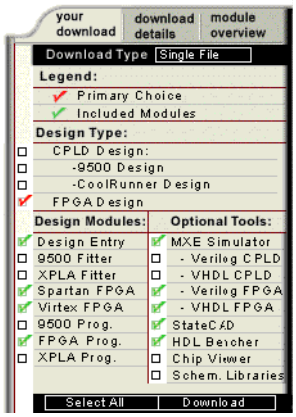
```
Single File Download
```

This will lead you to the following screen:

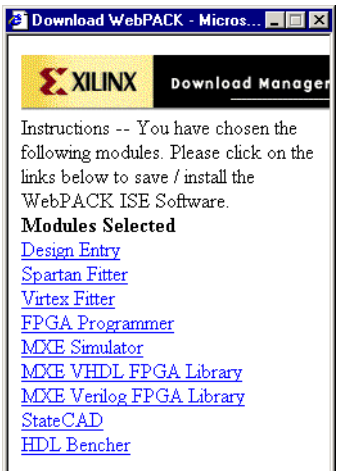
Click Design Configurations, FPGA Design, Select Configuration...



... and the following box appears:



Clicking the Download button, pops up the Download Manager window. The software is divided into several modules, which may be downloaded and installed separately.



The following tutorial assumes, that you download and install the following modules:

- Design Entry
- Spartan Fitter
- FPGA Programmer
- MXE Simulator
- MXE VHDL FPGA Library
- StateCAD

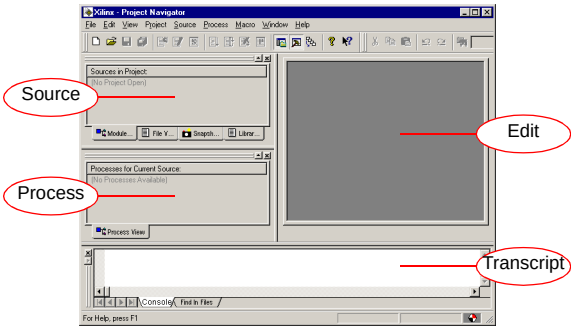
In Version 3.3WP8.x these files sum up to approximately 92MB of transfer volume and about 350MB of disk space after installing.

Optionally, you may download the following optional files, which sum up to another 40MB:

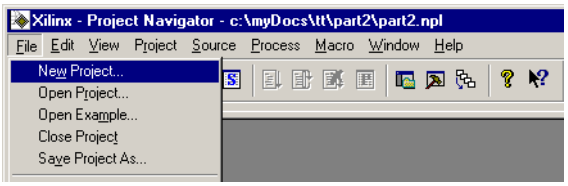
- Virtex Fitter
- MXE Verilog FPGA Library
- HDL Bencher

Creating a design

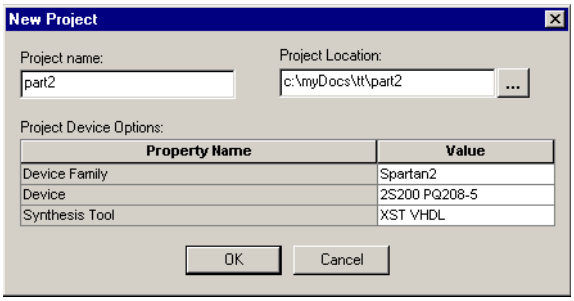
After launching the *WebPACK Project Navigator* for the first time, it comes up with an empty screen, divided into several panes. The *Source* window lists all sources being included in a project. The *Process* window lists the possible actions for a selected source. The *Edit* window is used to display and modify HDL sources. The *Transcript* window shows a log of the most recent shell operations.



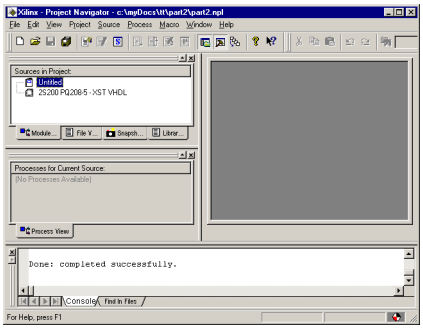
Before going further, we need to create a new project.



A dialog pops up to specify the project's name and location. Furthermore the target family and device as well as the synthesis tool need to be chosen. To target the *TE-XC2S* board, we need to specify *Spartan2* as the device family and *2S200PQ208-5* as the device. Our synthesis tool is *XST VHDL*.



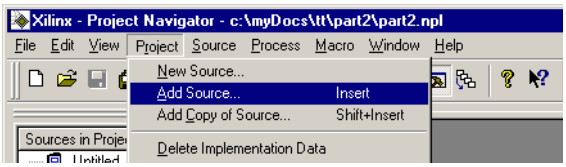
After closing the dialog, an empty project will be created in the project location specified above:



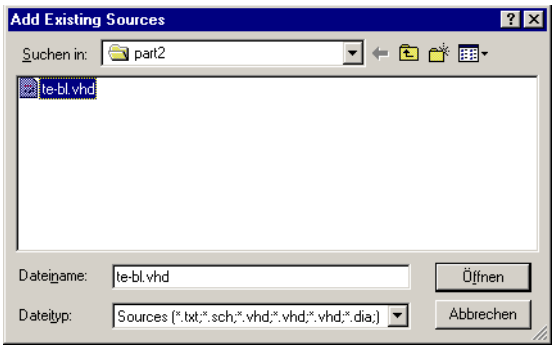
Design entry

Adding existing Source Files

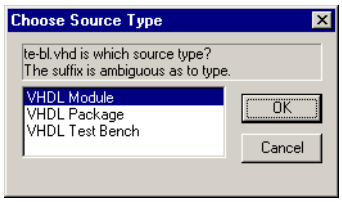
We start our project by adding an existing VHDL module to it. Copy the file from the CD-ROM into your project location, before proceeding. You may use the Windows Explorer to do so.



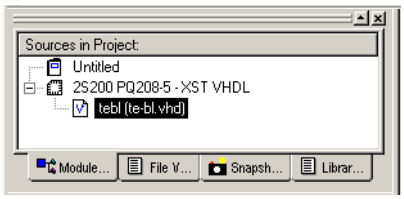
The file *TE-BL.vhd* comes with your *TE-XC2S* Development System and contains commonly used code to interface with the *Buttons & Lights* Expansion Board.



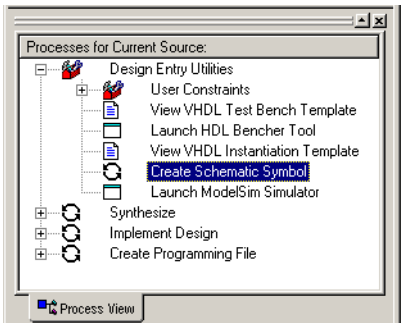
A *.vhd* suffix is ambiguous, as VHDL files may contain packages, modules, or test benches. Our file contains a VHDL entity or *module*.



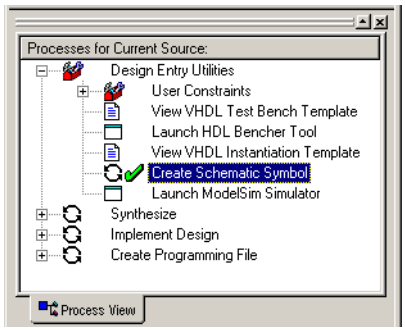
The file is added to the source window. The name of the VHDL entity, *tebl*, is displayed, followed by the name of the source file, *te-bl.vhd*, in parentheses.



In the process window, we double click *Create Schematic Symbol*, as we want to use the entity tebl in a schematic diagram later on.

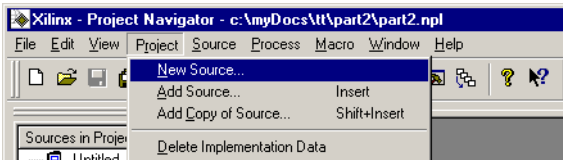


Successful generation of the schematic symbol is indicated by a green check mark.

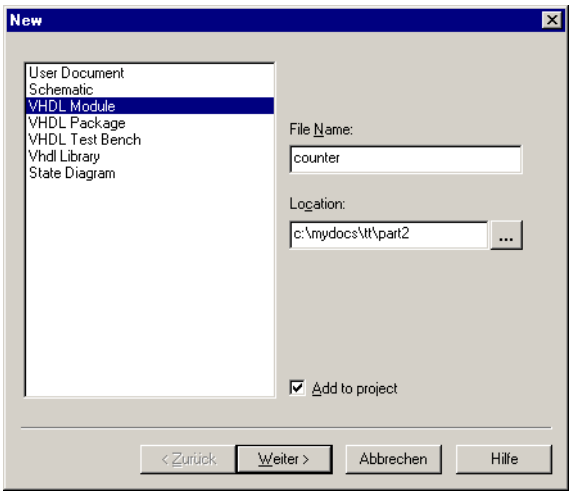


Using the HDL Editor

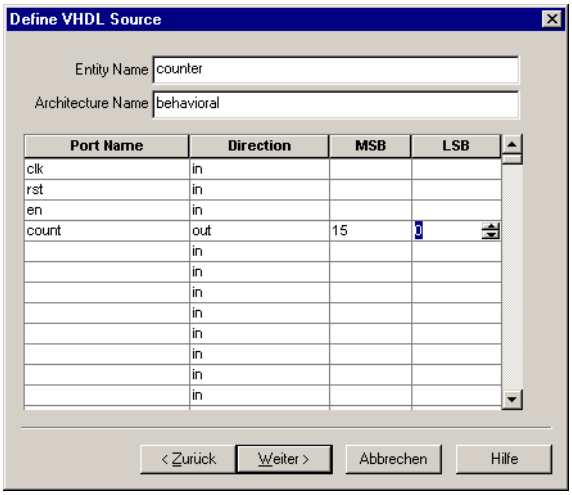
Next, we create a new VHDL file and add it to the project.



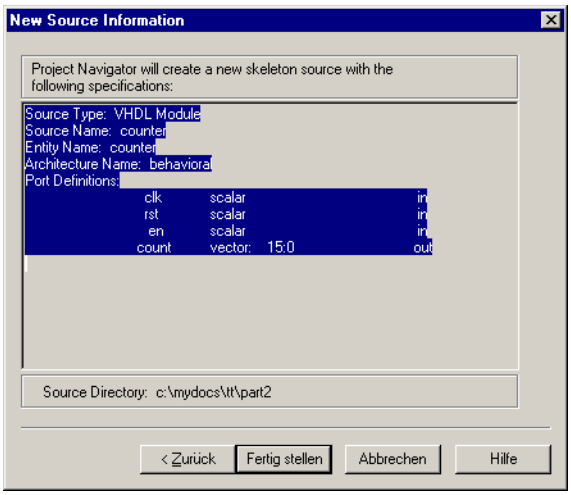
A dialog box appears, to specify the type of source to create, the file name and location. As we want to create another VHDL entity, we chose *VHDL Module*.



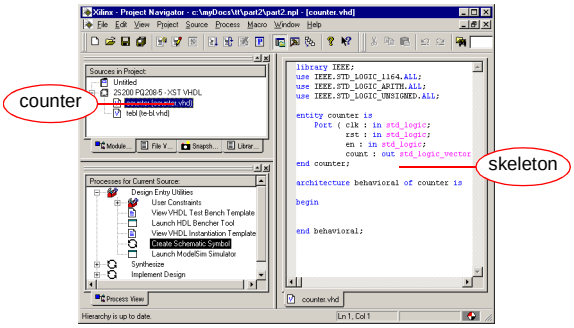
Next, an assistant pops up, querying information required, to create a skeleton for our VHDL entity declaration and architecture body. We need to add port name, and direction for every port. For buses, we need to specify the MSB and LSB as well.



Before creating the skeleton source, a summary window is displayed.



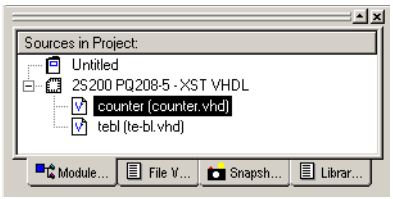
And finally, the source file is created and added to the project.



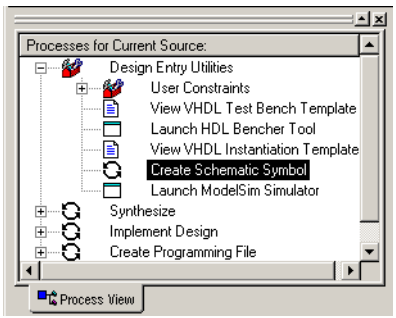
We complete the architecture body by adding the following lines to it:

```
architecture behavioral of counter is
    signal counter:   STD_LOGIC_VECTOR(count'range);
    signal div:       STD_LOGIC_VECTOR(16 downto 0);
    signal divl, en2: STD_LOGIC;
begin

    process(clk, rst)
    begin
        if rst= '1' then
            div <= (others=> '0');
            divl<= '0';
            en2 <= '0';
        elsif rising_edge(clk) then
            div <= div + 1;
            divl<= div(div'left);
            en2 <= div(div'left) and not(divl);
        end if;
    end process;

    process(clk, rst)
    begin
        if rst= '1' then
            counter<= x"0000";
        elsif rising_edge(clk) then
            if en= '1' and en2= '1' then
                counter<= counter + 1;
            end if;
        end if;
    end process;

    count<= counter;

end behavioral;
```

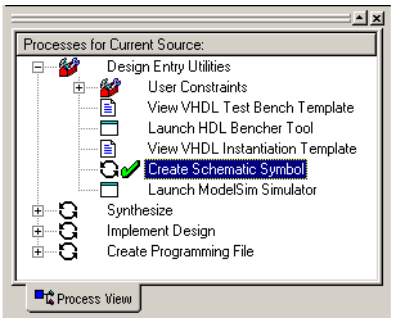
And like before, create a schematic symbol. We need to select our newly created source first...



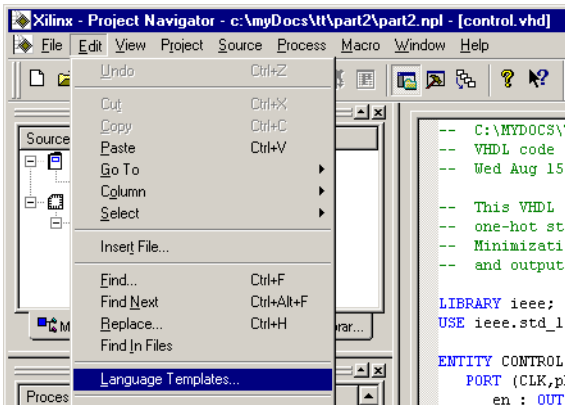
... double click *Create Schematic Symbol*...



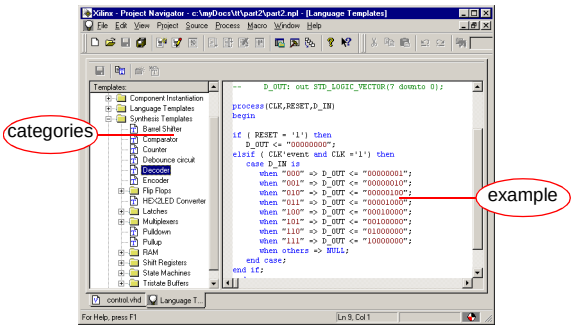
... and a green check mark indicates successful generation of the symbol.



In case we have questions about VHDL, or need some inspiration, there are language templates.



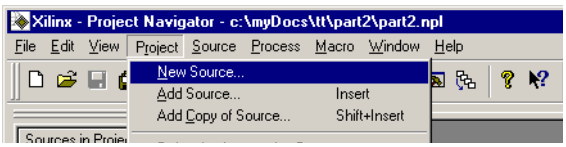
The left pane sorts the templates by category, while the right pane displays the actual sample code.



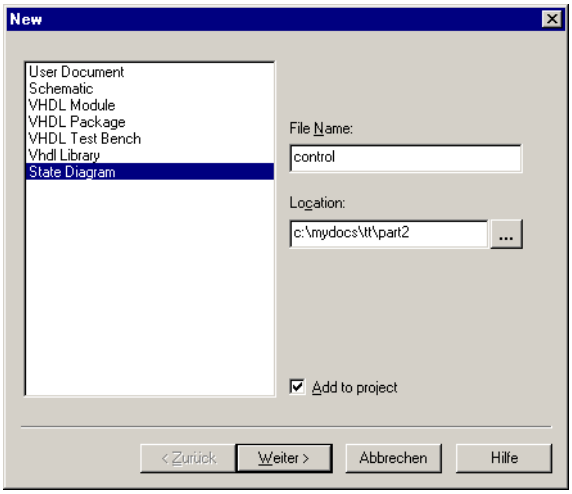
Check out the language templates and get familiar with the editor! It behaves much like any other programming editor, so you shouldn't have a hard time doing so.

Using the FSM Editor

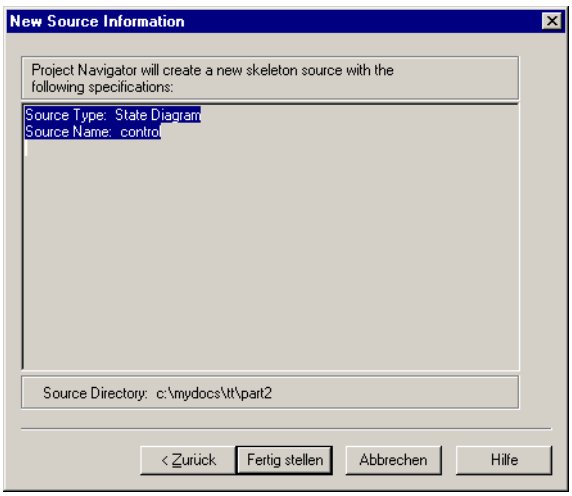
Now we create a finite-state machine using the FSM Editor.



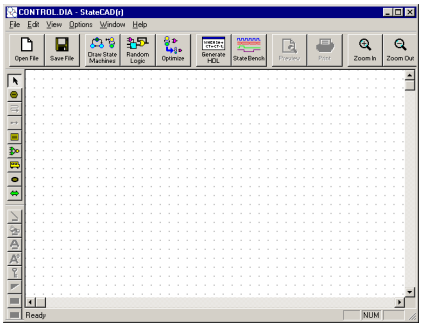
A dialog pops up, to query further information about the new source file. We chose *State Diagram* here.



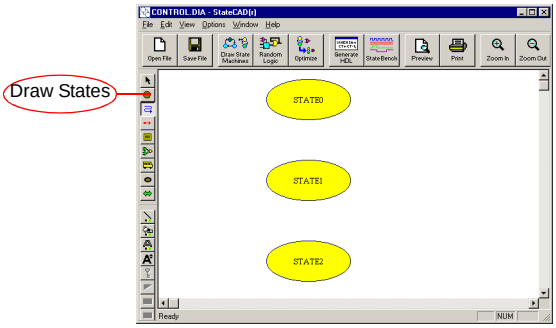
Before creating the new diagram, a summary window is displayed.



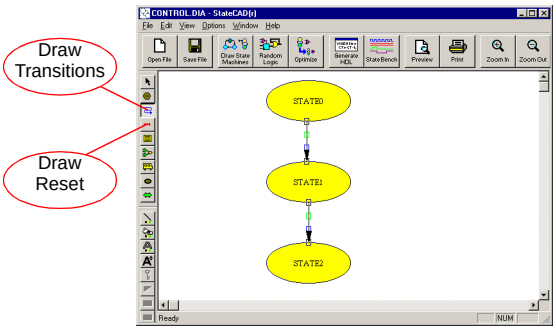
Next, the FSM Editor is launched, displaying an empty sheet.



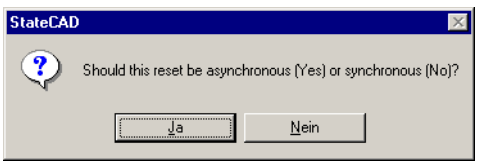
We use the *Draw States* tool, to add a few states to our FSM.



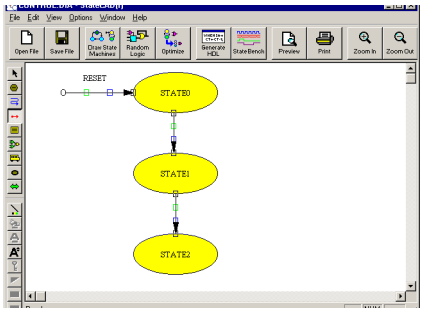
Next, we use the *Draw Transitions* tool, to connect our states.



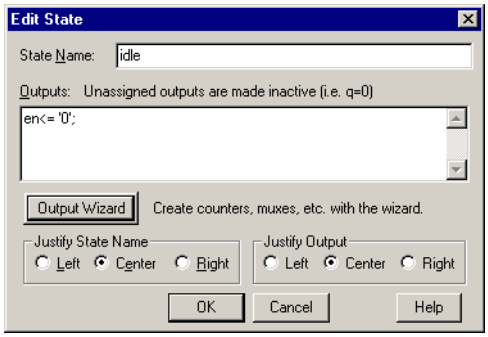
We chose the *Draw Reset* tool, to add a reset to *State0*, and will be prompted for the type of reset. We select *asynchronous* here.



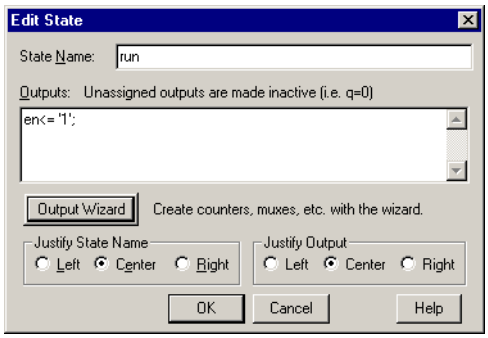
Now our machine looks like this:



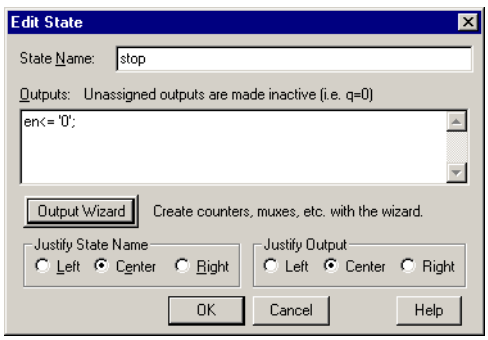
In the following, we edit the state's properties. A double click on *State0* pops up the following dialog. We alter the state name and add an output.



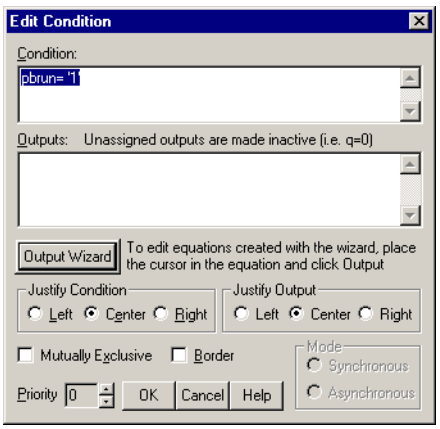
We do the same for *State1*...



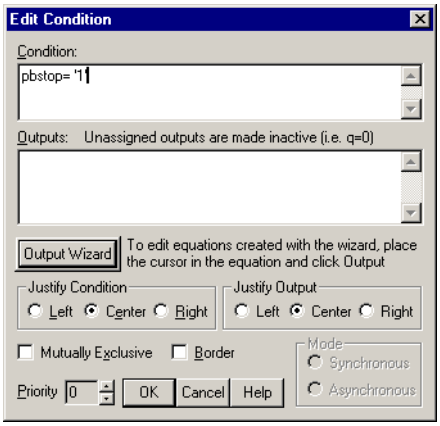
... and *State2*.



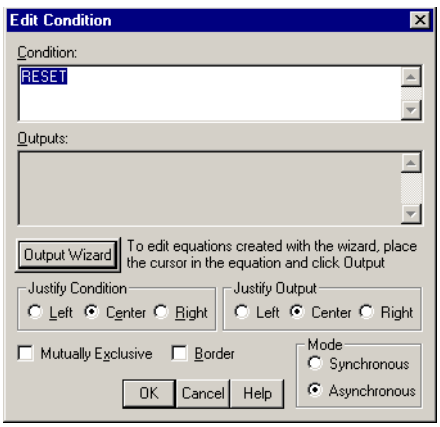
Next, we need to set up the transitions. Double clicking on the transition from *idle* to *run* pops up the following dialog. We add a condition to this transition.



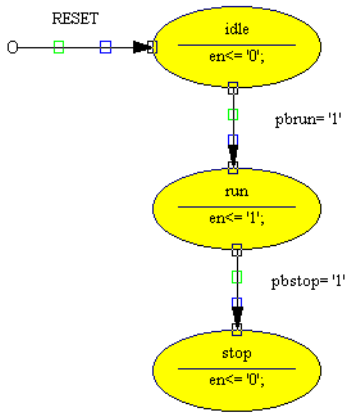
For the transition connecting *run* with *stop* we need to add a similar condition:



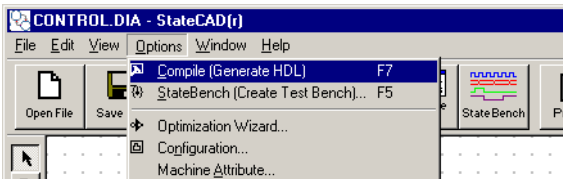
Our reset vector is already properly set up, but to be sure, we may open the according dialog as well:



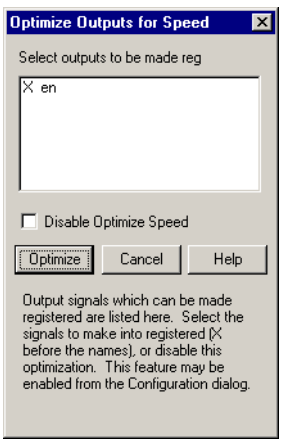
Now we finished editing our FSM, the final result should look like this:



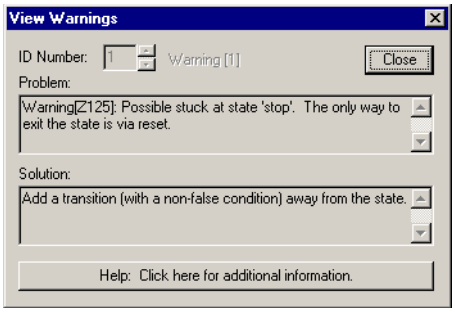
Next, we compile our graphical representation of the state machine into synthesizable VHDL.



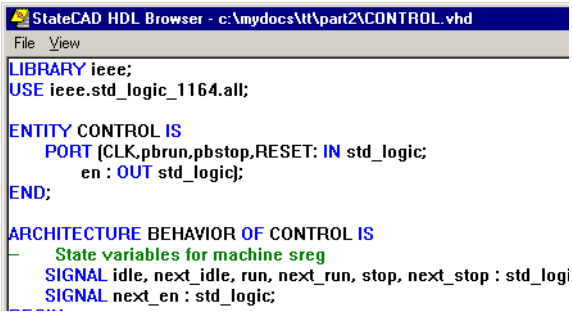
There are several options, how our statemachine can be translated to VHDL. The most important option is, whether the outputs are registered or not. We chose not to register our outputs.



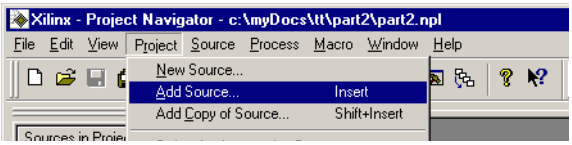
The parser detected, that there is no transition leaving the stop state, except for a reset condition. For our machine this is intentional, so we may safely ignore this warning.



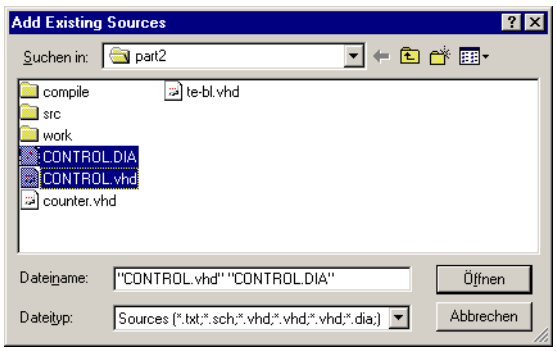
Finally, the generated source code is displayed for review. You should have a look at the code and check, if this reflects, what you had in mind.



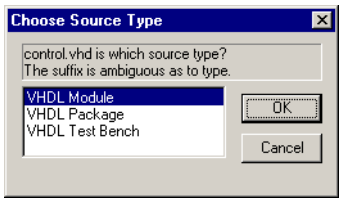
We exit the FSM Editor now and find ourselves back in the Project Navigator. Unfortunately our statemachine has not been added to project, so we need to do this manually.



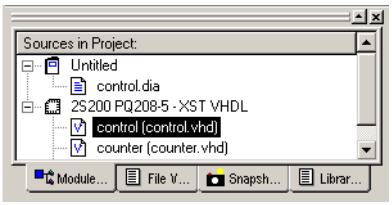
For convenience, we add both, the VHDL file, and its graphical representation to our project. This allows us to open the diagram later on by simply double clicking the *.dia* file. Unlike the *.vhd* file, the *.dia* file is *not* required for synthesis.



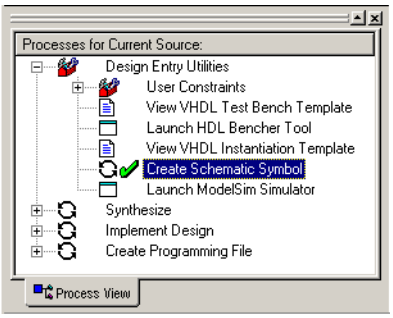
Again, we are asked for the type of our VHDL source.



To create a schematic symbol for our state machine, we select the source file.



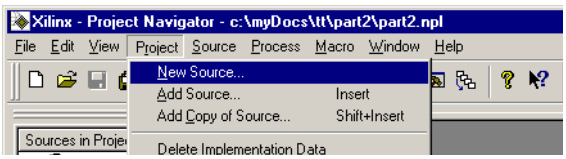
Double clicking *Create Schematic Symbol* creates the symbol, which is indicated by the green check mark.



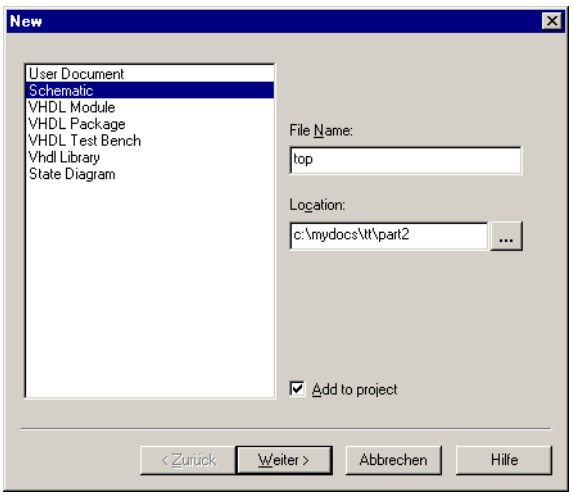
Creating state machines is a non-trivial task. It is highly recommended to get familiar with the various options of the state machine editor, their effect on the generated VHDL code and their effect on synthesis and implementation.

Using the Schematic Editor

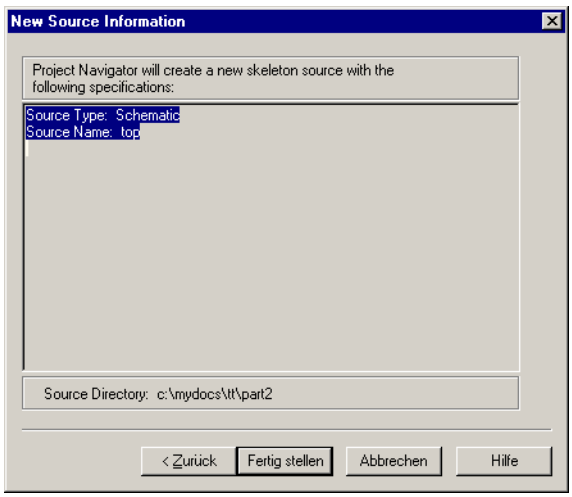
The top layer of our design is created with the schematic editor, instantiating the symbols being defined during previous design entry steps. First, we need to create a new source.



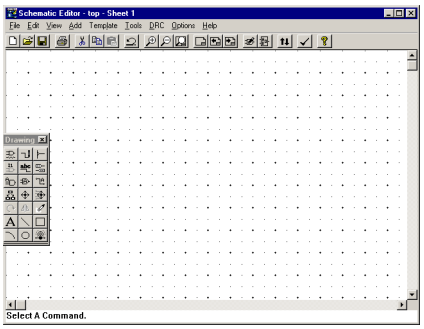
We specify *Schematic* as the source type and enter file name and location.



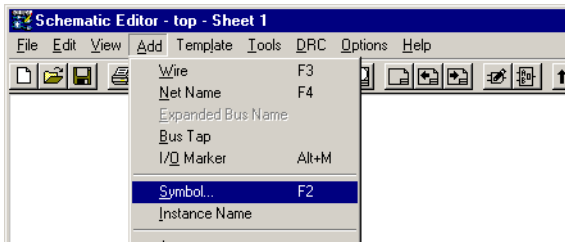
A summary is displayed, before actually creating the new source.



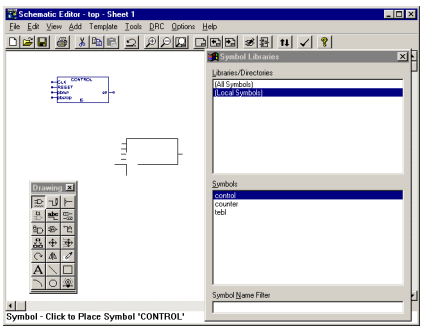
Then, the schematic editor is launched, displaying an empty sheet.



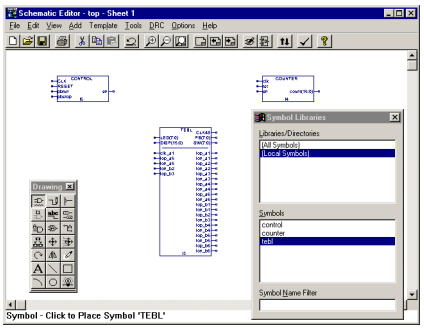
First, we add our previously created schematic symbols to the sheet.



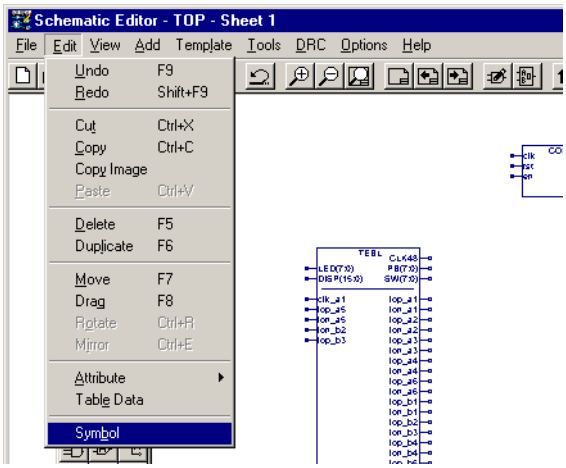
The library window pops up. We add the symbols *control*, *counter* and *tebl* to the sheet.



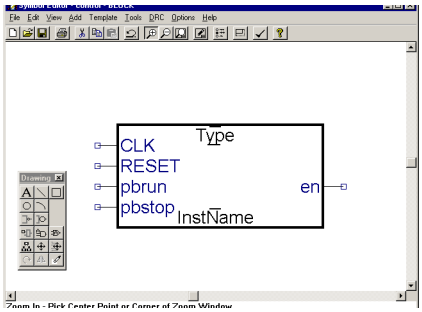
After adding the symbols, our sheet looks like this.



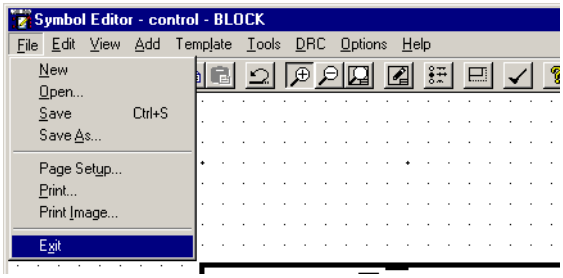
Probably, we need to tidy up our symbols a bit.



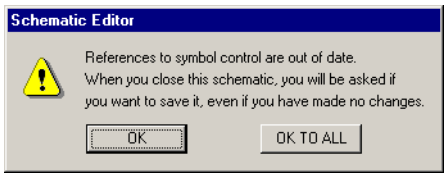
Clicking on symbol will open the symbol editor window. We adjust *control* to look like this.



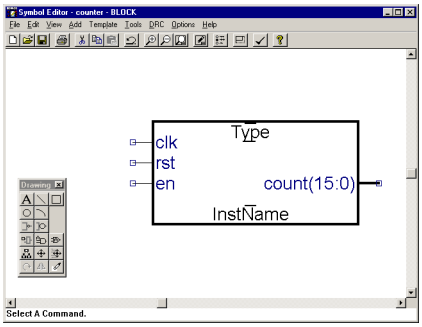
And exit the symbol editor.



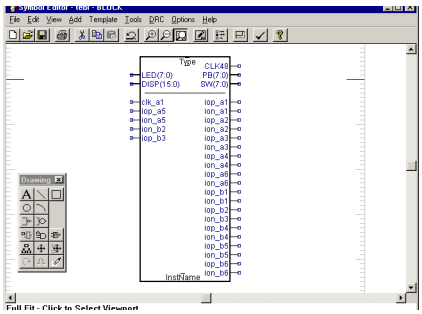
The schematic editor notifies us before updating the symbols. We click *OK TO ALL*.



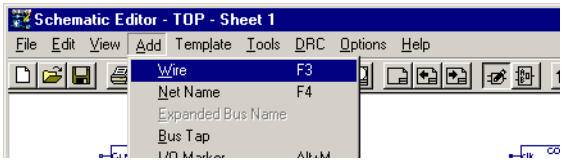
Next, we adjust *counter* to look like this...

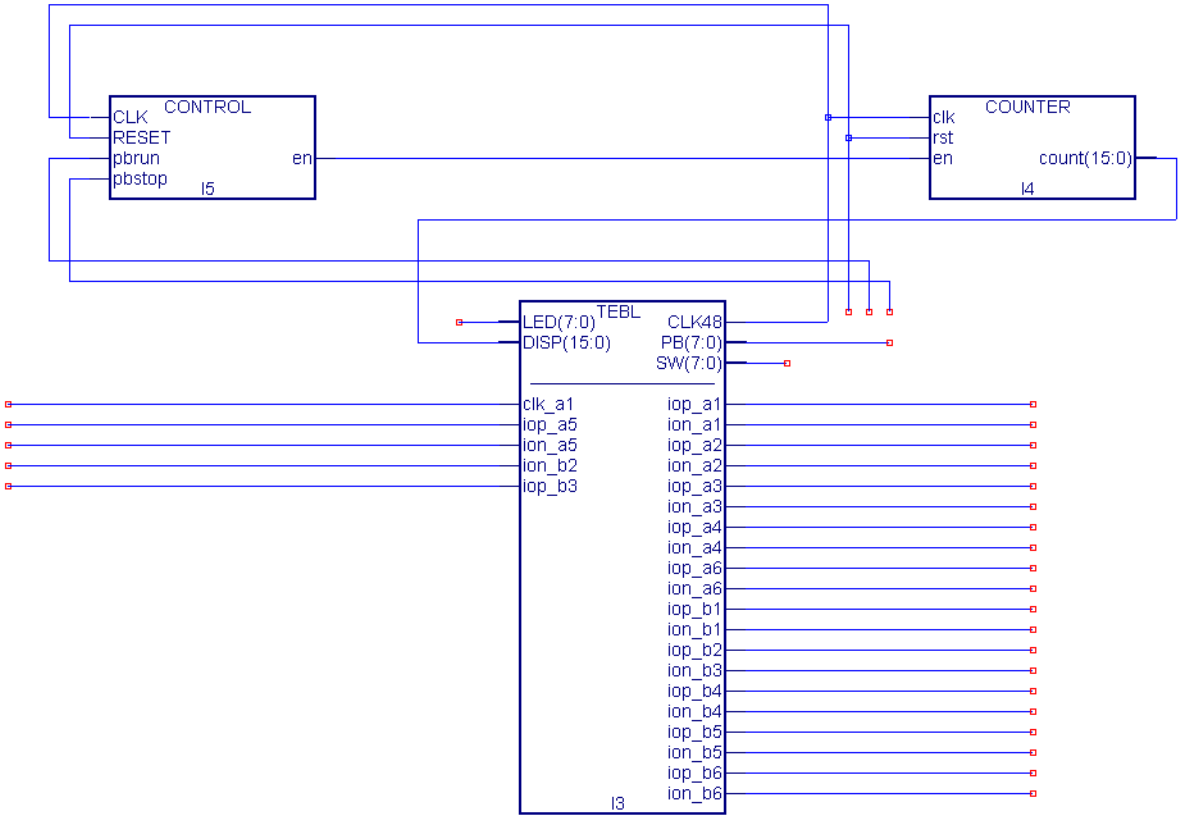


... and *tebl* to look like this.



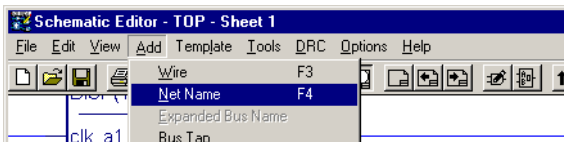
Now we interconnect the symbol instances with a few wires.



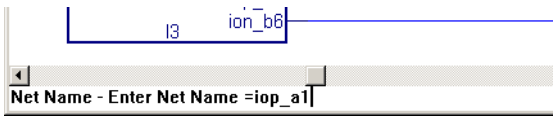


The result should look like above.

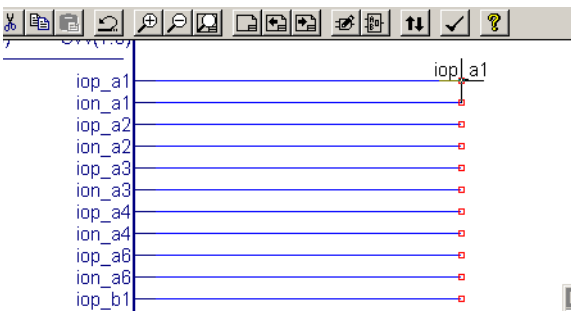
Next, we assign names to some signals.



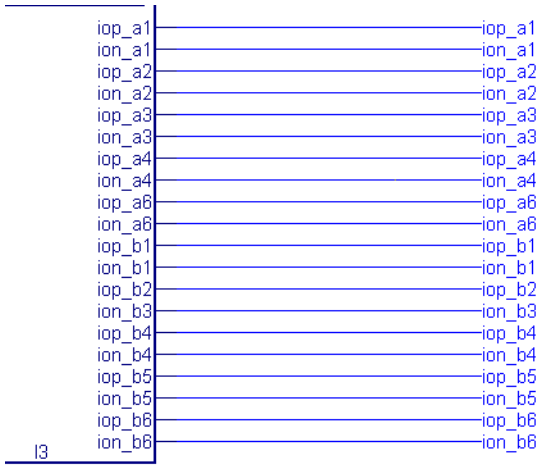
To do so, we edit the name in the status line...



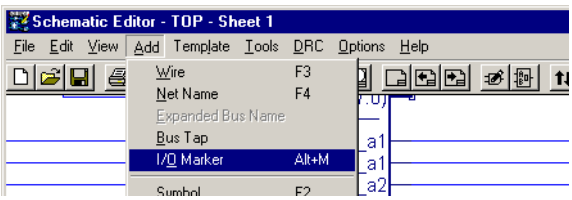
... and after hitting return, we can assign the name to a wire, by clicking on its red marker.



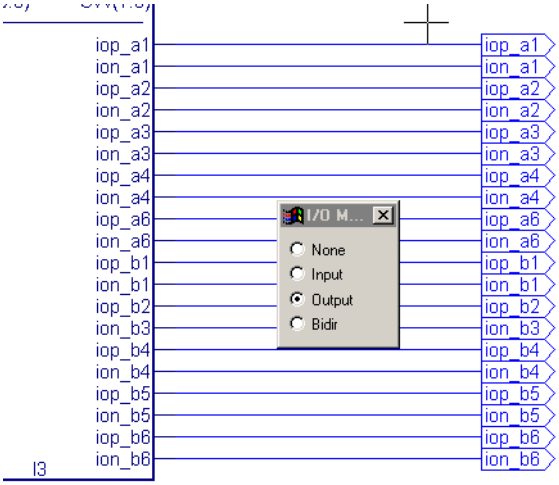
We proceed to do so with some other lines...



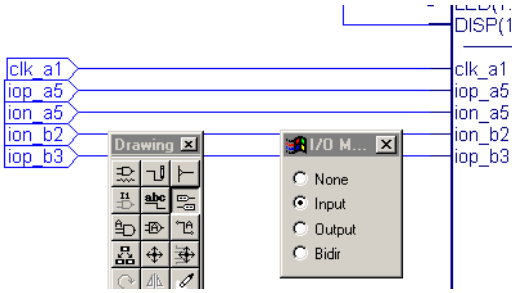
... and add I/O Markers to them.



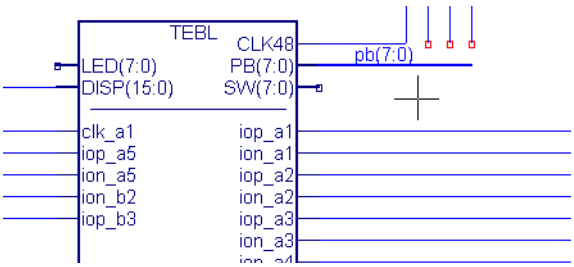
The I/O Marker selector pops up. We select output, and click all of our previously named wires.



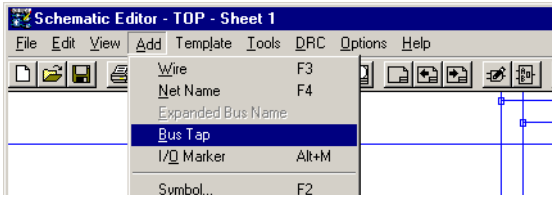
The input pads are created accordingly.



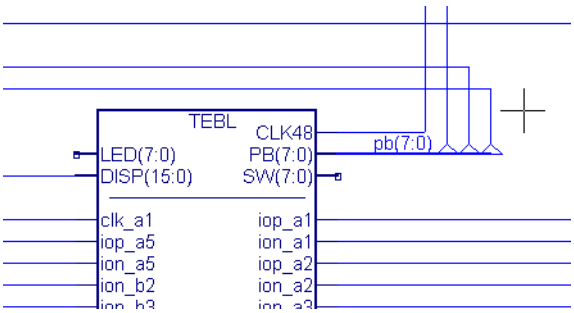
Next, we assign a name to a bus. The thickness of the wire is increased to reflect this.



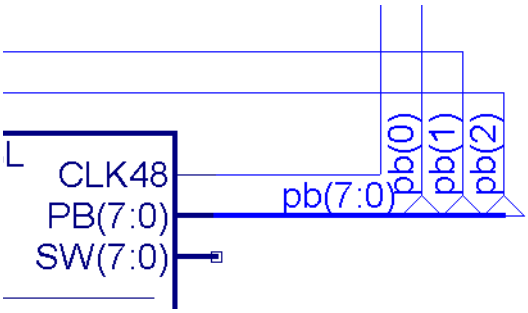
We need to split the bus and create some taps...



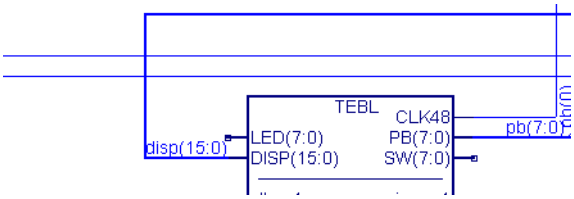
... like this:

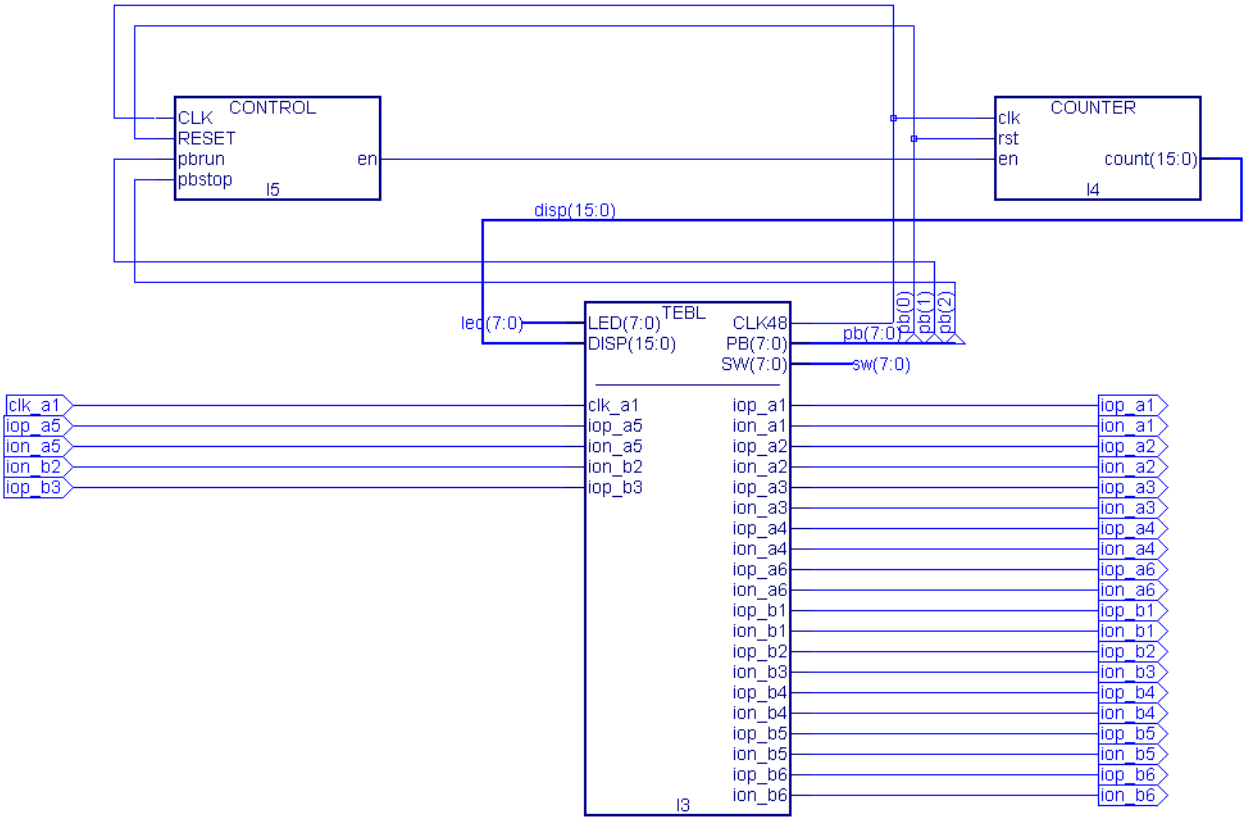


To assign a bus tap to a specific bus signal, we need to assign net names again.



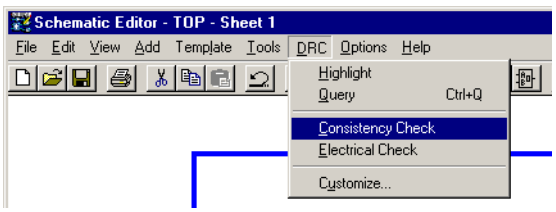
Finally, we create another bus...



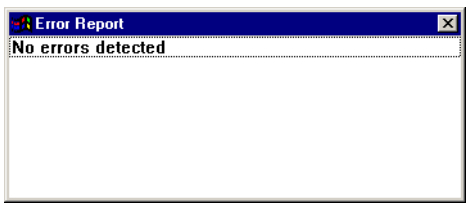


... and yield the result above.

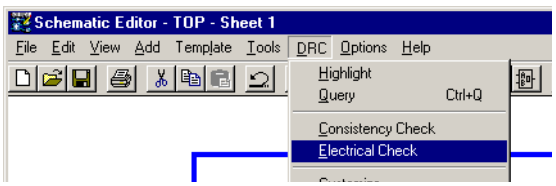
Now, we perform a consistency check...



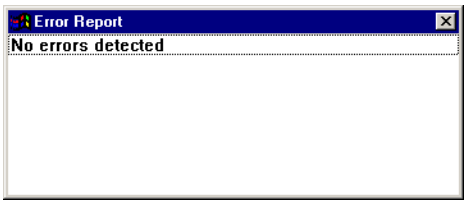
... which goes just fine...



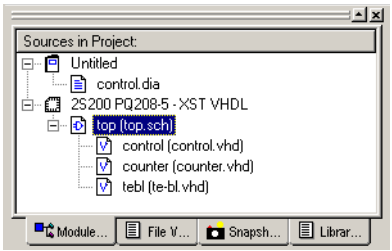
and an electrical check...



... which is also just fine.



Now we have completed design entry and may exit the schematic editor. The Project Navigator visualizes the hierarchy of our project.

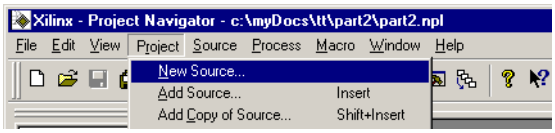


Similar to the state machine editor, the schematic editor is only a graphical frontend to automatic VHDL code generation. This may be important in some rare cases, e.g. when signal naming conflicts with VHDL conventions.

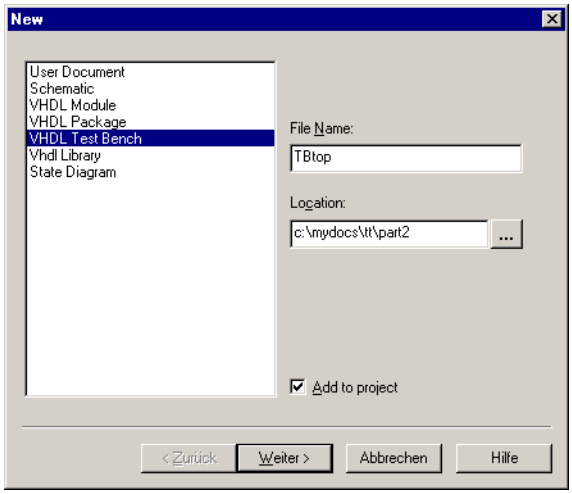
Behavioral simulation

Creating a Testbench

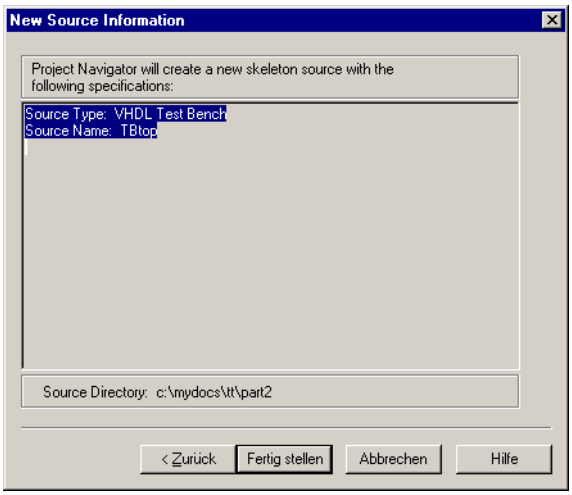
Before simulating our design, we need to create a Testbench, which is applying stimuli to our circuit. Create a new source...



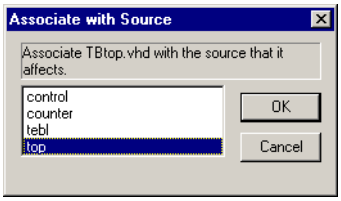
... with the wizard.



Review the information...



... and assign the newly created file to the design's top level.

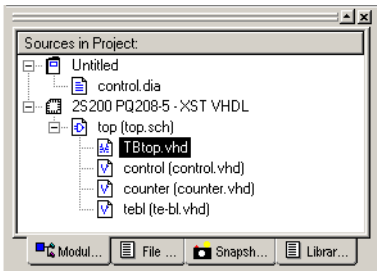


Then type in your testbench code. The following code is the simplest form of a testbench, as it is only applying stimuli to the design. More sophisticated testbenches also perform checks on the design's responses.

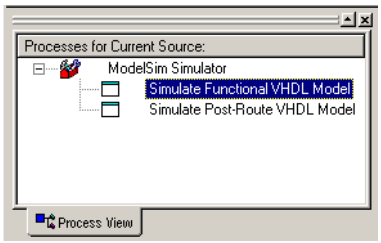
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

entity testbench is
end testbench;

architecture bhv of testbench is
    component top port (
            clk_a1: In  std_logic;
            iop_a1: Out std_logic; ion_a1: Out std_logic;
            iop_a2: Out std_logic; ion_a2: Out std_logic;
            iop_a3: Out std_logic; ion_a3: Out std_logic;
            iop_a4: Out std_logic; ion_a4: Out std_logic;
            iop_a5: In  std_logic; ion_a5: In  std_logic;
            iop_a6: Out std_logic; ion_a6: Out std_logic;
            iop_b1: Out std_logic; ion_b1: Out std_logic;
            iop_b2: Out std_logic; ion_b2: In  std_logic;
            iop_b3: In  std_logic; ion_b3: Out std_logic;
            iop_b4: Out std_logic; ion_b4: Out std_logic;
            iop_b5: Out std_logic; ion_b5: Out std_logic;
            iop_b6: Out std_logic; ion_b6: Out std_logic);
    end component;

    signal   seg_g, seg_f, seg_a, c_pb: STD_LOGIC;
    signal   clk:                       STD_LOGIC:= '0';
    signal   sim:                       STD_LOGIC:= '1';
    signal   sw2, sw3, sw4:             STD_LOGIC;
    constant zeroC:                     STD_LOGIC:= '0';
    signal   zero:                      STD_LOGIC;
begin
    zero<= zeroC;

    sim<= '1', '0' after 15 ms;
    clk<= sim and not(clk) after (1 us/48)/2;

    sw2<= '1', '0' after 1.5 ms;
    sw3<= '0', '1' after 2 ms,  '0' after 3.5 ms;
    sw4<= '0', '1' after 13 ms, '0' after 14.5 ms;

    c_pb<= (seg_g and sw2) or (seg_f and sw3) or (seg_a and sw4);

    UUT: top port map(
        clk_a1=> clk, ion_a3=> seg_a, iop_a4=> seg_f,
        ion_a4=> seg_g, iop_a5=> c_pb,
        ion_a5=> zero, ion_b2=> zero, iop_b3=> zero);

end bhv;
```

Using the Simulator

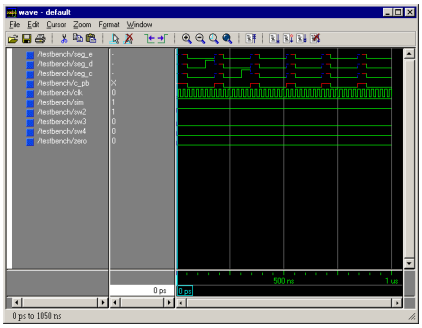
Select *TBtop*...



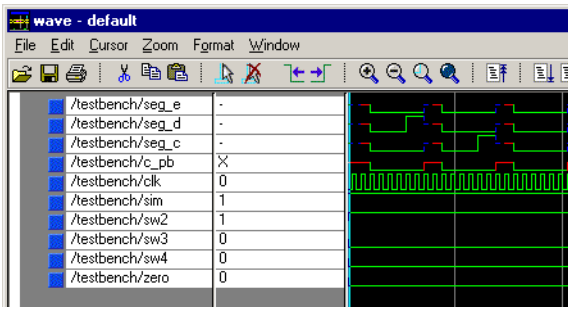
... and double-click *Simulate Functional VHDL Model*...



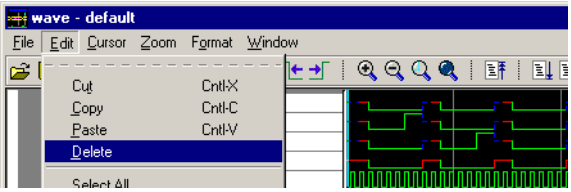
... to launch the *ModelSim* simulator.



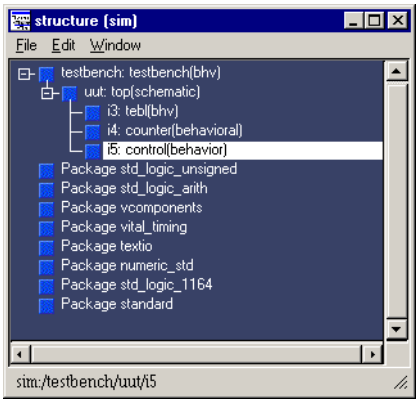
Select all signals...



... and delete them.

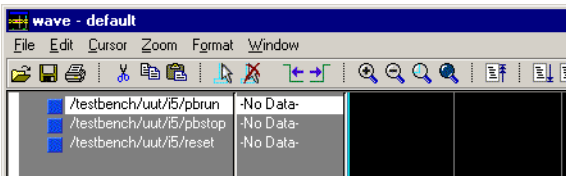


Select the hierarchy level...

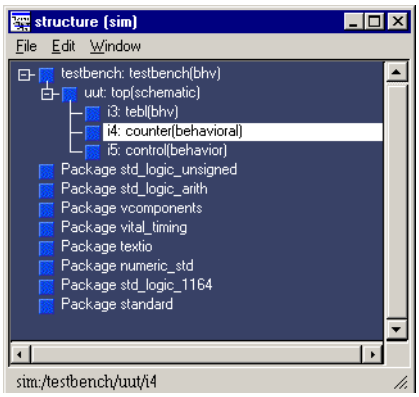


... and the signals to watch...

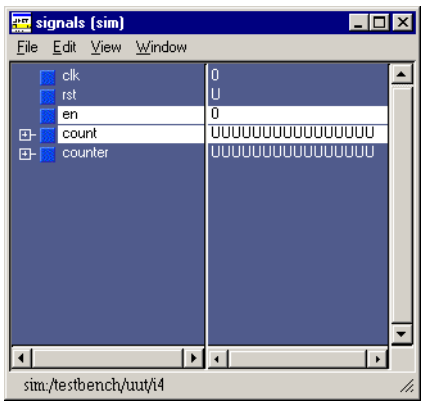
... and drag them into the wave window.



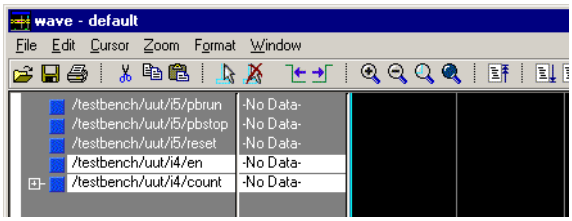
Repeating these steps...



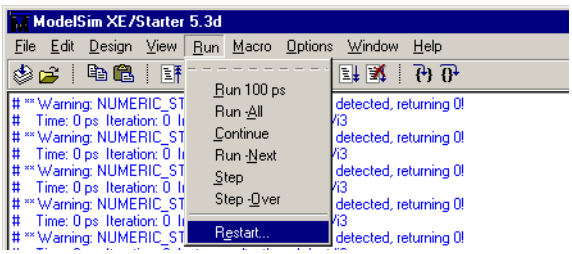
... for entity *counter* as well...



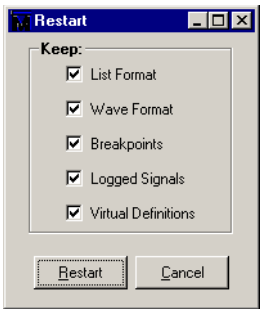
... completes our wave window. Click the save button to store your setup for later sessions.



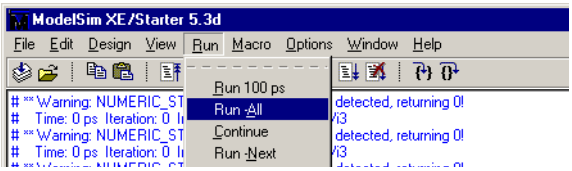
Now restart the simulator...



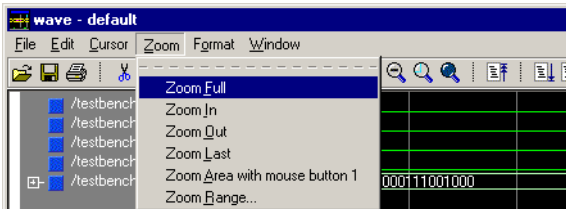
... click *restart*...



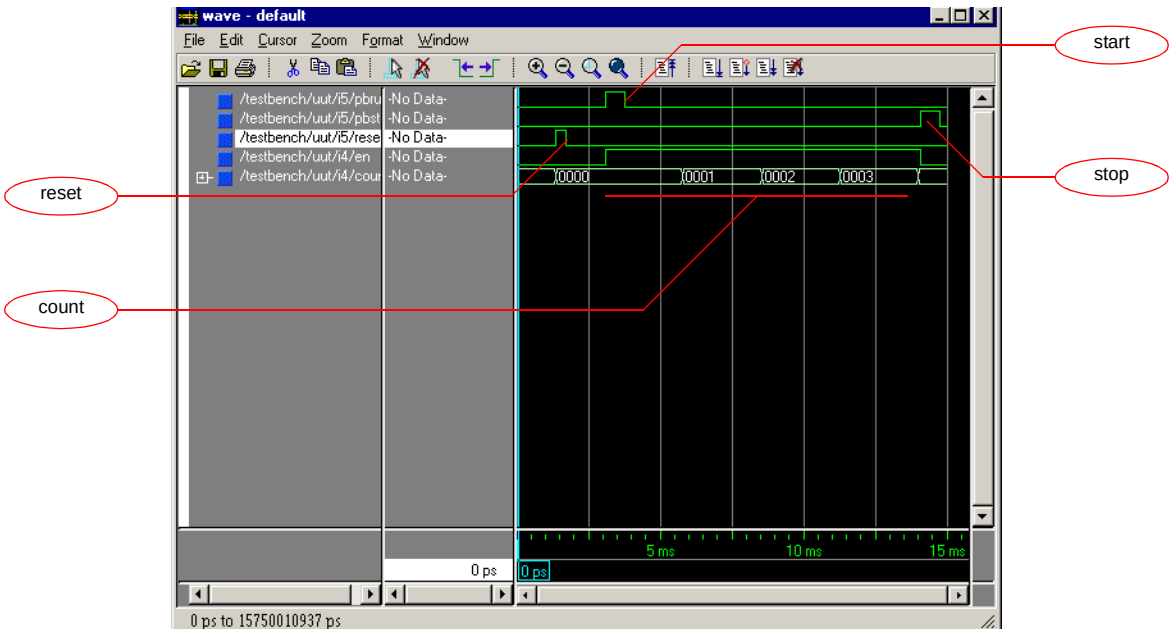
... and run the complete simulation. For our design, the total simulation range is 15ms, which results in a run time of approximately 5 minutes on a *Pentium-III* with 700MHz.



Zoom to full...



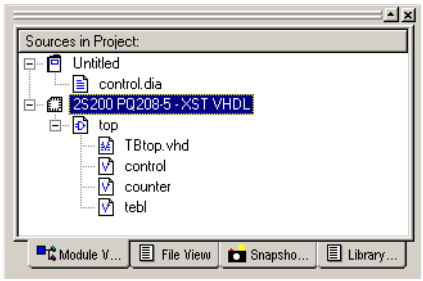
... to see our counter design working. You are invited to play around with the simulator. Browse the design hierarchy, watch signals, set breakpoints in the VHDL code and much more!



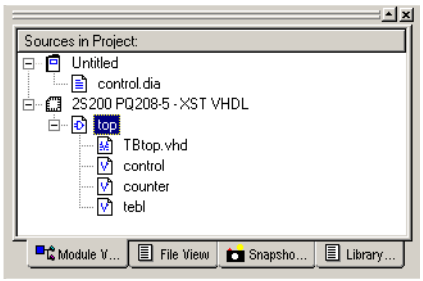
Compiling the design

Synthesis

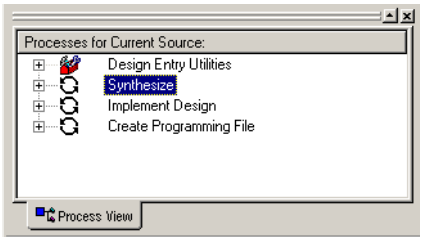
Before synthesizing our design, we double-check that the correct target family is selected. When targeting the *TE-XC2S* board, the correct device is *2S200 PQ208-5*.



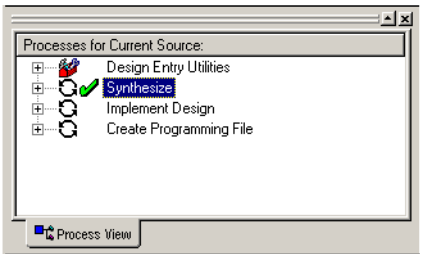
Next select the design's top level...



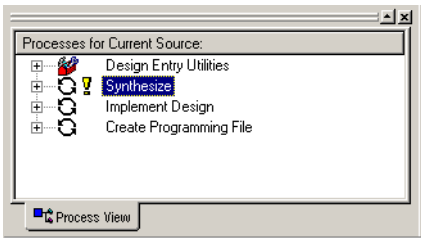
... and double click on the *Synthesize* process.



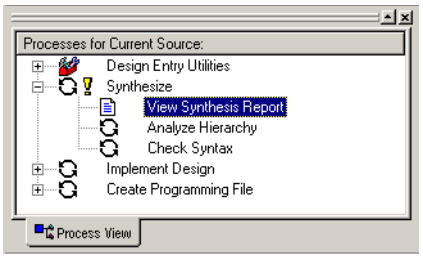
A green check mark indicates success...



... while a yellow exclamation mark indicates warnings.



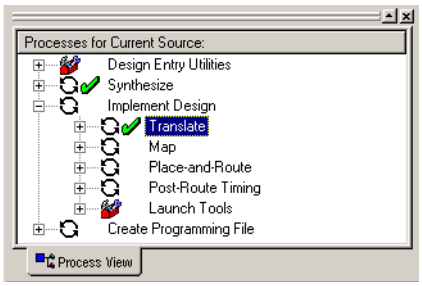
You should review the synthesis report to be sure.



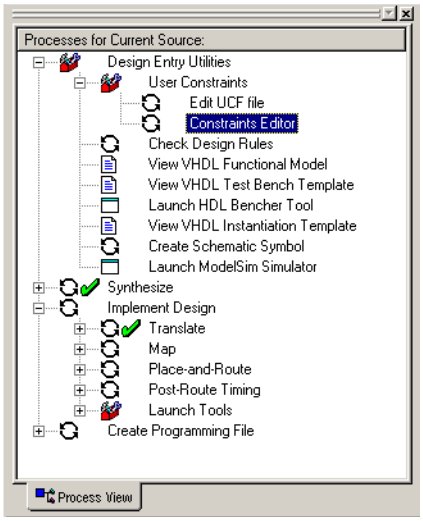
A number of options controls the way your behavioral description is translated into a structural netlist during synthesis. You may safely leave these options to their default values, unless you are working on a high-performance design.

Constraints

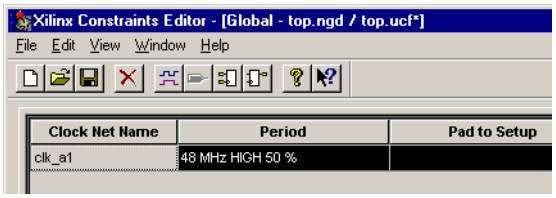
Before editing the constraints, you need to translate your design. To do so, double click on *Translate*.



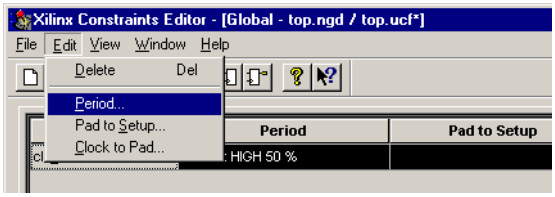
Now launch the *Constraints Editor*.



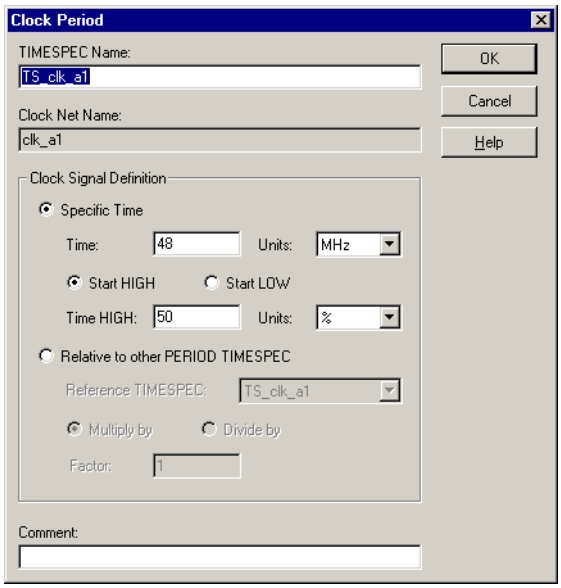
Select the *clk_a1* line...



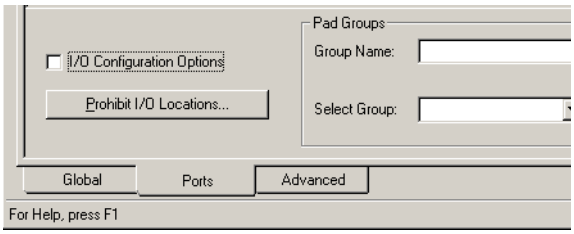
... and edit the clock period...



... to be 48MHz:



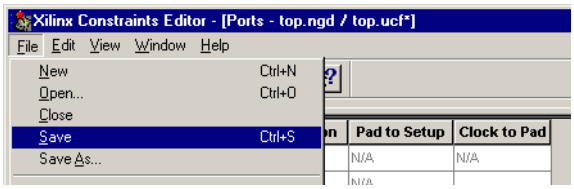
Go to the *Ports* tab...



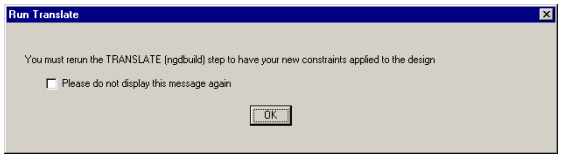
...and enter the following pin locations according to your *TE-XC2S* hardware:

Port Name	Port Direction	Location	Pad to Setup	Clock to Pad
clk_a1	INPUT	p185	N/A	N/A
ion_a1	OUTPUT	p192	N/A	
ion_a2	OUTPUT	p194	N/A	
ion_a3	OUTPUT	p199	N/A	
ion_a4	OUTPUT	p201	N/A	
ion_a6	OUTPUT	p205	N/A	
ion_b1	OUTPUT	p5	N/A	
ion_b3	OUTPUT	p9	N/A	
ion_b4	OUTPUT	p14	N/A	
ion_b5	OUTPUT	p16	N/A	
ion_b6	OUTPUT	p18	N/A	
iop_a1	OUTPUT	p191	N/A	
iop_a2	OUTPUT	p193	N/A	
iop_a3	OUTPUT	p195	N/A	
iop_a4	OUTPUT	p200	N/A	
iop_a5	INPUT	p202		N/A
iop_a6	OUTPUT	p204	N/A	
iop_b1	OUTPUT	p4	N/A	
iop_b2	OUTPUT	p6	N/A	
iop_b4	OUTPUT	p10	N/A	
iop_b5	OUTPUT	p15	N/A	
iop_b6	OUTPUT	p17	N/A	

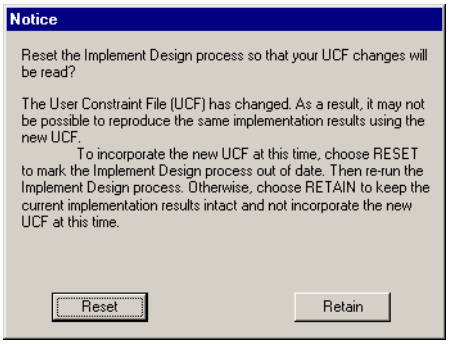
Now save the result...



... click *OK*...



... end exit the *Constraints Editor*. Click *Reset* to incorporate your new constraints.

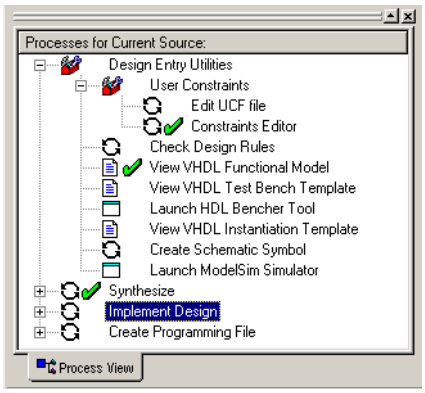


Most likely, you will need to set up your constraints only once. Later implementation steps will keep the settings you made here.

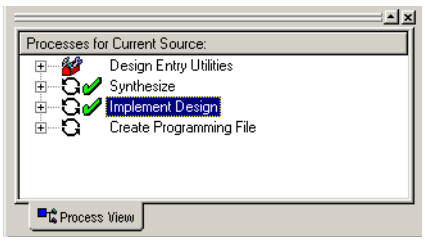
Be warned that wrong or incomplete timing constraints may lead to unexpected behavior, while wrong or missing pin lockings may lead to hardware damage!

Implementation

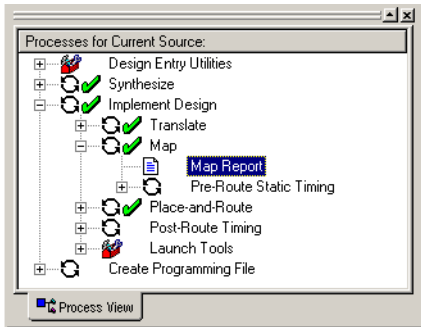
Now you are ready to run implementation. Double click *Implement Design* to do so.



A green check mark indicates successful operation.



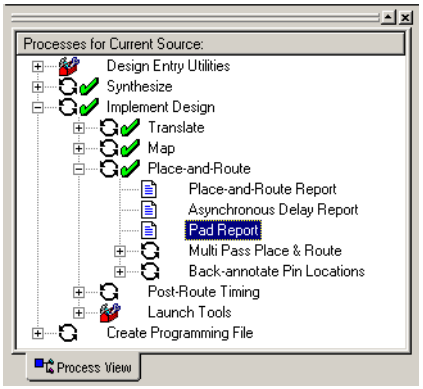
Check the *Map Report*...



... to learn about your design's resource usage.

```
Design Summary
--------------
   Number of errors:      0
   Number of warnings:    1
   Number of Slices:                    88 out of  2,352
   Number of Slices containing
      unrelated logic:                   0 out of     88
   Number of Slice Flip Flops:          73 out of  4,704
   Number of 4 input LUTs:              90 out of  4,704
   Number of bonded IOBs:               21 out of    140
   Number of GCLKs:                      1 out of      4
   Number of GCLKIOBs:                   1 out of      4
Total equivalent gate count for design:  1,376
Additional JTAG gate count for IOBs:  1,056
```

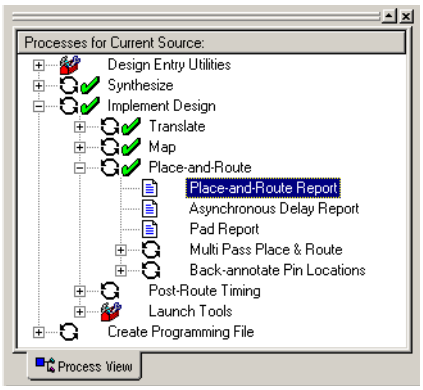
Review the *Pad Report*...



... to learn about your pin locations.

```
Pinout by Pin Name:
+------------------------+-----------+--------+
|                   Pin  | Direction | Pin    |
|                        |           | Number |
+------------------------+-----------+--------+
| clk_a1                 | INPUT     | P185   |
| ion_a1                 | OUTPUT    | P192   |
| ion_a2                 | OUTPUT    | P194   |
| ion_a3                 | OUTPUT    | P199   |
| ion_a4                 | OUTPUT    | P201   |
| ion_a6 (VREF)          | OUTPUT    | P205   |
| ion_b1                 | OUTPUT    | P5     |
| ion_b3 (VREF)          | OUTPUT    | P9     |
| ion_b4                 | OUTPUT    | P14    |
| ion_b5                 | OUTPUT    | P16    |
| ion_b6                 | OUTPUT    | P18    |
| iop_a1                 | OUTPUT    | P191   |
| iop_a2                 | OUTPUT    | P193   |
| iop_a3                 | OUTPUT    | P195   |
| iop_a4 (VREF)          | OUTPUT    | P200   |
| iop_a5                 | INPUT     | P202   |
| iop_a6                 | OUTPUT    | P204   |
| iop_b1 (VREF)          | OUTPUT    | P4     |
| iop_b2 (VREF)          | OUTPUT    | P6     |
| iop_b4                 | OUTPUT    | P10    |
| iop_b5                 | OUTPUT    | P15    |
| iop_b6                 | OUTPUT    | P17    |
+------------------------+-----------+--------+
```

Check your *Place-and-Route Report*...



... to learn about the speed of your design

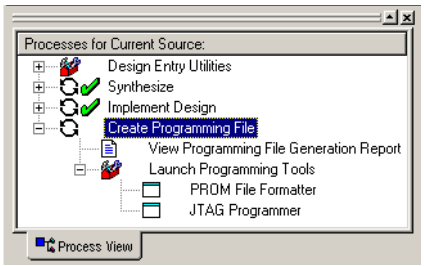
```
-------------------------------------------------------------------
 Constraint                    | Requested  | Actual
                               |            |
-------------------------------------------------------------------
 TS_clk_a1 = PERIOD TIMEGRP    | 20.833ns   | 10.532ns
 33 nS   HIGH 50.000 %         |            |
-------------------------------------------------------------------
```

Working with Hardware

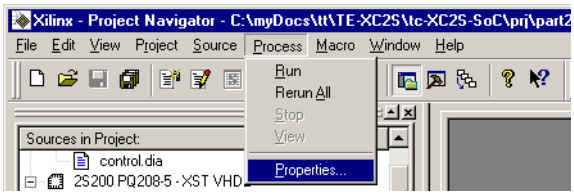
In the following paragraphs, we download our design to the *TE-XC2S* board. Whenever hardware is involved, double check the constraints, especially the pin lockings, to avoid damaging your hardware!

Configuring from PROM

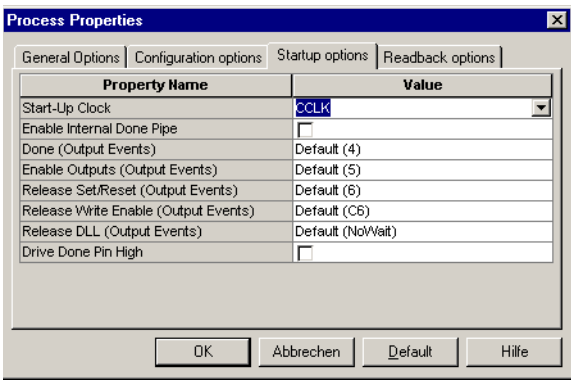
Before creating a programming file, check the options. Select *Create Programming File*...



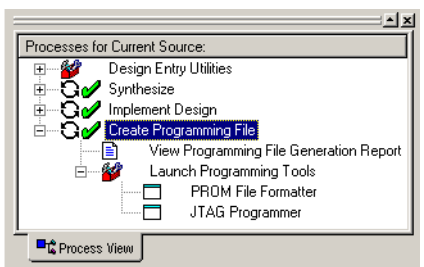
... and open the *Properties*.



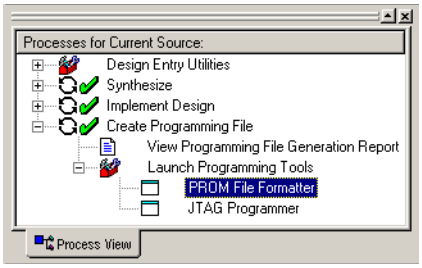
When creating a PROM, make sure the *Start-Up Clock* is set to *CCLK*.



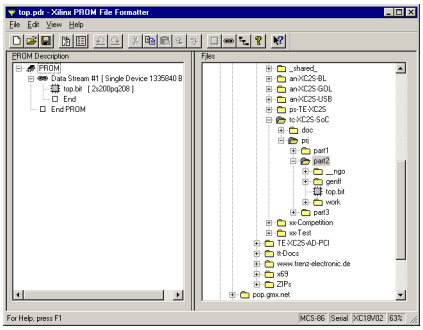
Now double click *Create Programming File* to create the bitstream.



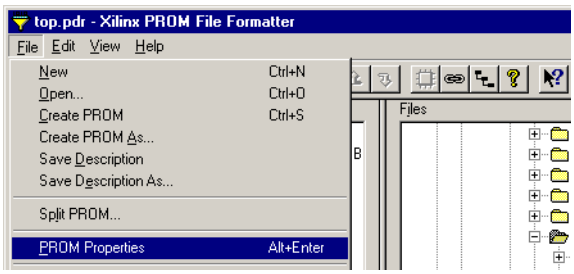
Next, double click *PROM File Formatter*...



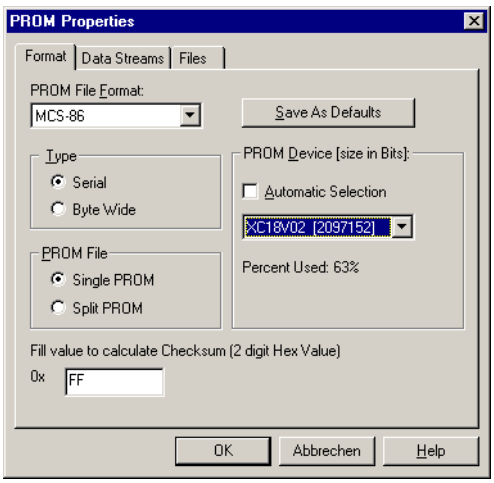
... to open this tool:



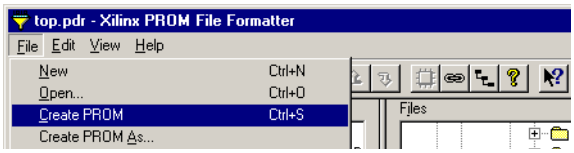
Check the *PROM Properties*...



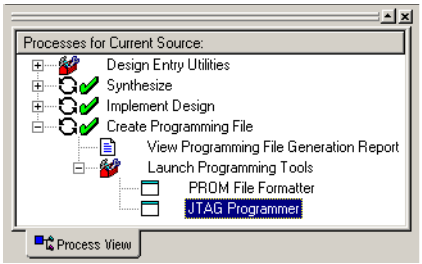
... and make sure the correct device is selected. Your *TE-XC2S* board uses an *XC18V02* Flash PROM.



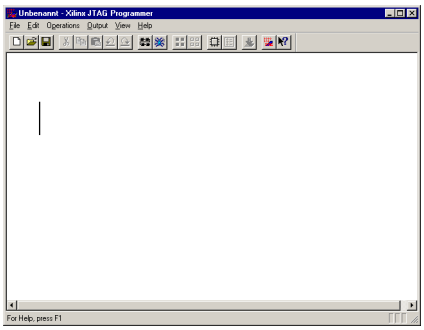
Now create the programming file,...



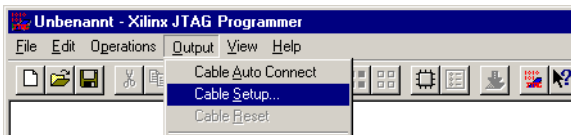
... exit the *PROM File Formatter* and launch the *JTAG Programmer*.



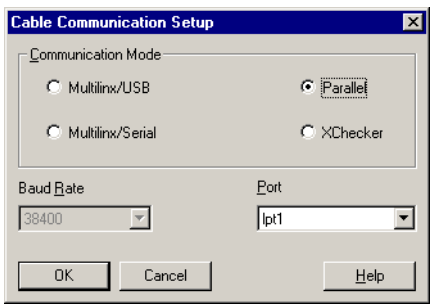
The following screen appears:



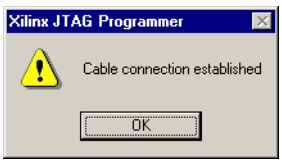
Open the *Cable Setup*...



... and setup for *Parallel Cable III*.

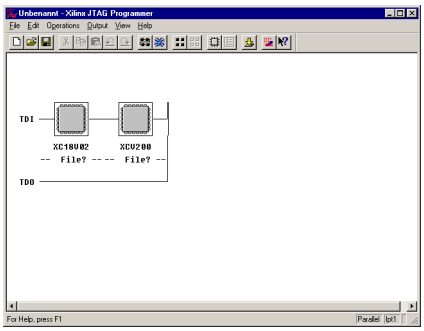


Successful cable detection is indicated like this:

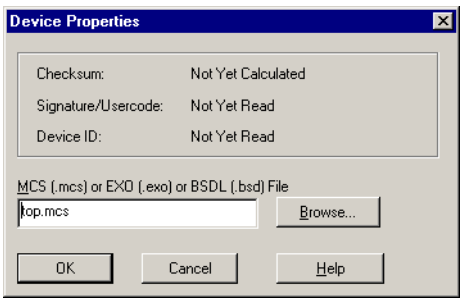


Now scan your JTAG chain...

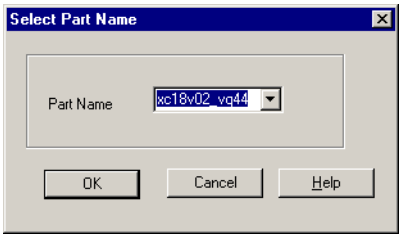
... for the devices on your *TE-XC2S* board.



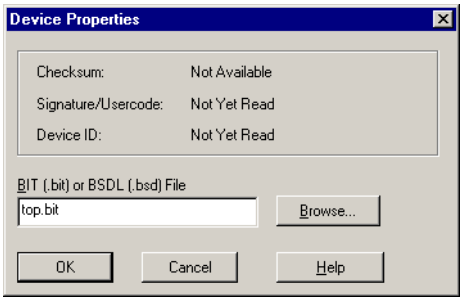
Double click the *XC18V02* PROM and browse for the *MCS* file created by the *PROM File Formatter* a few steps ago.



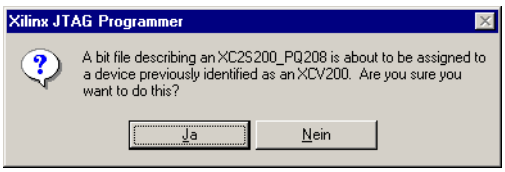
Chose the *XC18V02-VQ44* part.



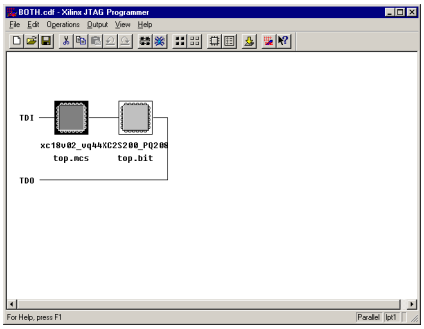
Double click the *XCV200* FPGA and browse for the *BIT* file created during Implementation.



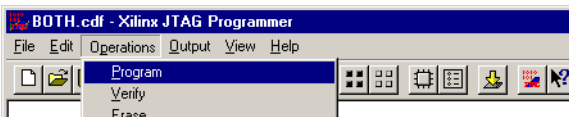
The *Spartan-II* family is a close derivative of the *Virtex* family, for this reason the following warning may be safely ignored.



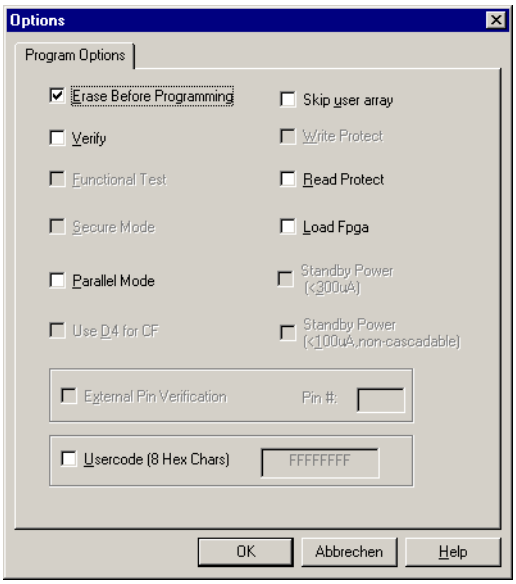
Select the Flash PROM again...



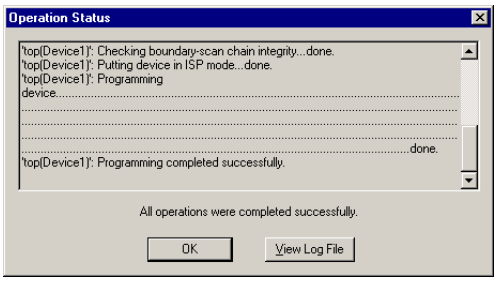
... and chose to program the device.



Setup the programming options...



... and program the device.

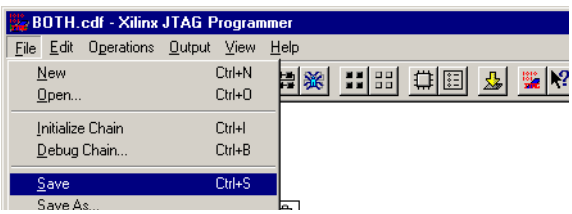


To configure your *TE-XC2S* board from the Flash PROM, setup the jumpers as follows...

Pin	Setting
M0	open
M1	open
M2	open

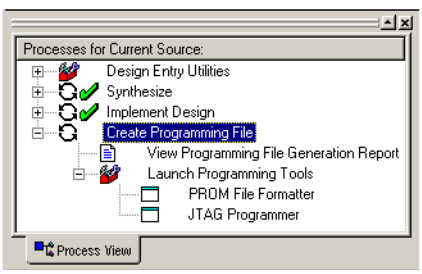
... and hit the *PRG* button to configure the FPGA. When the *DONE* LED lights up, your board is configured successfully.

Don't forget to save your setup.

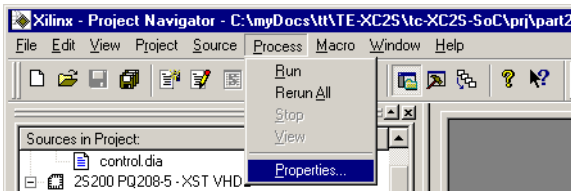


Configuration via JTAG

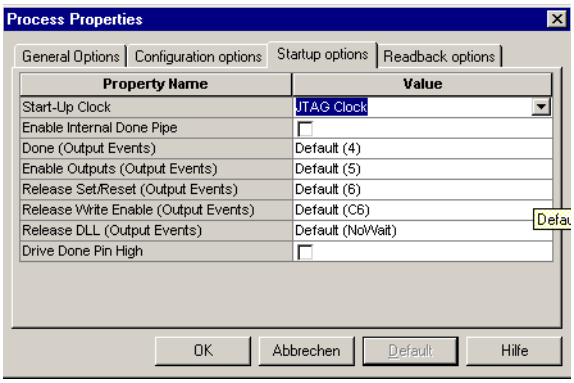
Instead of programming the Flash PROM and configuring the FPGA from the PROM, the FPGA may be configured directly via JTAG. To do so, you will need to create a programming file with modified options. Select *Create Programming File*...



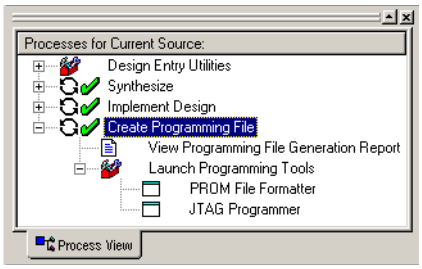
... and open the *Properties*.



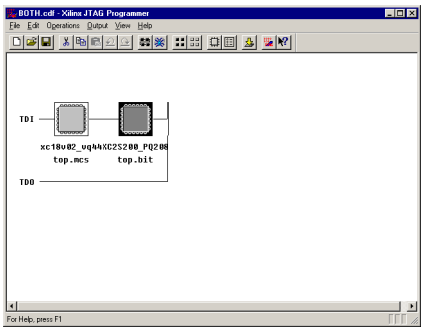
When configuring via JTAG, make sure the *Start-Up Clock* is set to *JTAG*.



Now double click *Create Programming File* to create the bitstream.



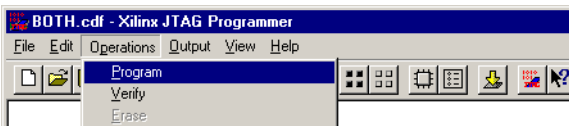
Launch the JTAG Programmer, open the previously created project and select the FPGA.



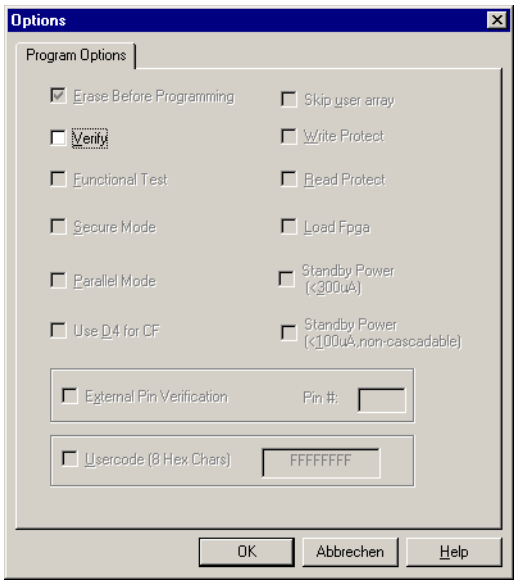
Make sure your *TE-XC2S* board is set up as follows...

Pin	Setting
M0	closed
M1	open
M2	open

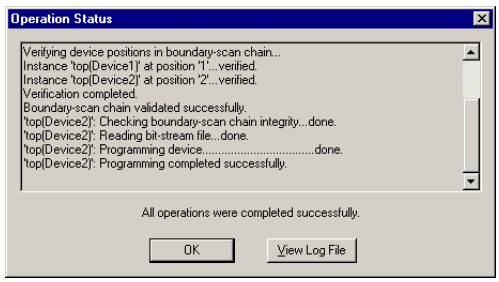
Chose to program the device...



... setup the programming options...



... and program the device.



Like before, the *DONE* LED lights up, once the FPGA configured successfully. Unlike configuring from PROM, the configuration is lost, when pressing the *PRG* button.

Wrap Up

Congratulations! Now that you completed the tutorial, you gained a basic understanding of FPGA development and the tools involved. Now it's up to you, to expand your knowledge using the various internet resources and further readings.

The *References* section lists those documents, which describe the soft- and hardware used in this tutorial in full detail. While these documents provide complete and accurate information, most of them are not too instructive.

An introduction to FPGA technology is given here:

- *Spartan-II Development System*
  Tutorial: Introduction to FPGA Technology
  Trenz Electronic, November 5, 2001

The following application notes provide complete sample projects and helpful hints:

- *Spartan-II Development System*
  Application Note: Buttons & Lights
  Trenz Electronic, September 20, 2001
- *Spartan-II Development System*
  Application Note: Game Of Life
  Trenz Electronic, September 20, 2001

To improve your knowledge on VHDL, we recommend the following resources:

- *The VHDL Cookbook*
  Peter J. Ashenden
  ftp://ftp.cs.adelaide.edu.au/pub/VHDL-Cookbook/
- *The Designer's Guide to VHDL, 2nd Edition*
  Peter J. Ashenden
  ISBN: 1558606742
  Morgan Kaufmann, January, 2001
- *Schaltungsdesign mit VHDL*
  Lehmann, Wunder, Selz
  ISBN 3-7723-6163-3
  Franzis'-Verlag, 1994
  http://www-itiv.etec.uni-karlsruhe.de/FORSCHUNG/VEROEFFENTLICHUNGEN/lws94/lws94.html
- VHDL FAQ
  http://www.vhdl.org/vi/comp.lang.vhdl/

To find a helping hand to answer your questions, you should join the following usenet discussion groups:

- comp.arch.fpga
- comp.lang.vhdl

References

- *Foundation Series ISE 3.1i*
  User Guide
  Xilinx, Inc., October 17, 2000
- *Xilinx Synthesis Technology (XST)*
  User Guide
  Xilinx, Inc., October 17, 2000
- *Development System Reference Guide*
  Xilinx, Inc., October 11, 2000
- *Constraints Editor Guide*
  Xilinx, Inc., October 11, 2000
- *PROM File Formatter Guide*
  Xilinx, Inc., October 11, 2000
- *JTAG Programmer Guide*
  Xilinx, Inc., October 11, 2000
- *Spartan-II Development System*
  Product Specification
  Trenz Electronic, September 12, 2001
- *Spartan-II 2.5V FPGA Family*
  Product Specification
  Xilinx, Inc., October 31, 2000
- *XC18V00 Series of In-System*
  Programmable Configuration PROMs
  Product Specification
  Xilinx, Inc., April 4, 2000
- *Spartan-II FPGA Family Configuration and Readback*
  XAPP176
  Xilinx, Inc., December 4, 1999
- *Configuration and Readback of Virtex FPGAs Using (JTAG) Boundary-Scan*
  XAPP139
  Xilinx, Inc., February 18, 2000

Copyright

Revision History

Version	Date	Who	Description
1.0	2001sep25	FB	Created, WP3.3
1.1	2001nov06	FB	Minor additions

Table 1: Revision History