# COMPSYS403
# Lab1 – Using SOPC Builder to Create a NIOS II System

## Building a NIOS II System with SOPC Builder on DE2

**Wei-Tsun Sun**

**Version 2.1, 25th March 2008**

## Table of Contents

**Create a digital system which utilized the DE2 development board**

## 0. Prepare the software tools and environment settings

Extra hardware design provided by the manufacturer of DE2 boards are not installed, therefore you need to included them through few extra steps.

1. Obtain the **de2.zip** provided by us
2. Extract them to c:\altera, you should be able to see a directory called de2 is created. The structure of the c:\altera\de2 should look like Figure 1-1.

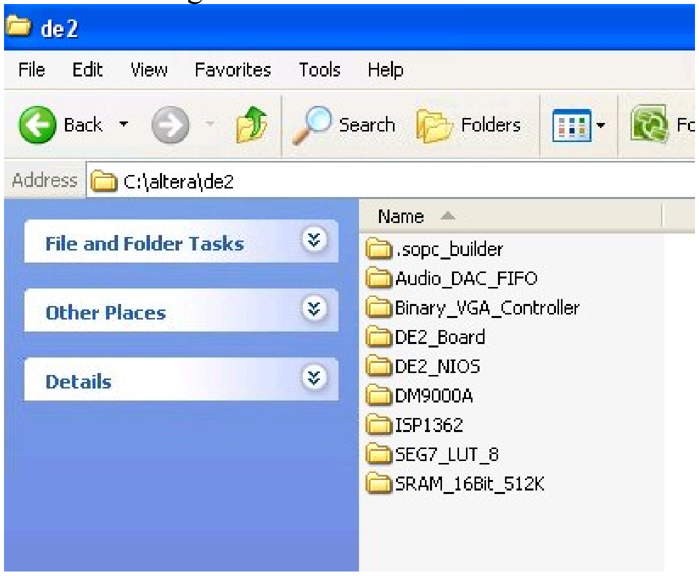
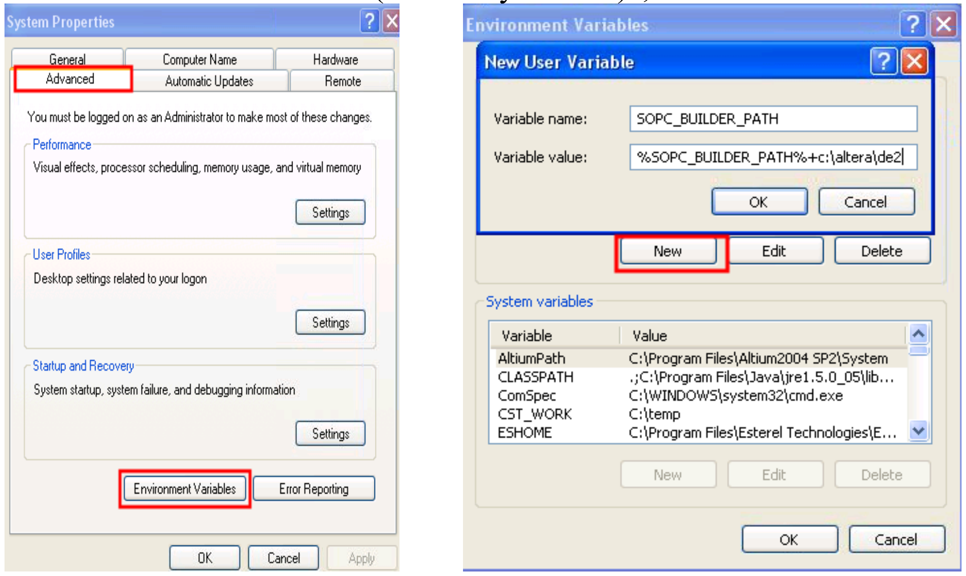


Figure 1-1: The structure of c:\altera\de2

3. Go to the desktop, right click on "My Computer", select "Properties", then select the "Advanced" tab, click the "Environment Variables" buttons at the lower left conner, as shown in Figure 1-2. Under the "User variables for xxxxxxx (which is your UPI)", click New to add a new variable.

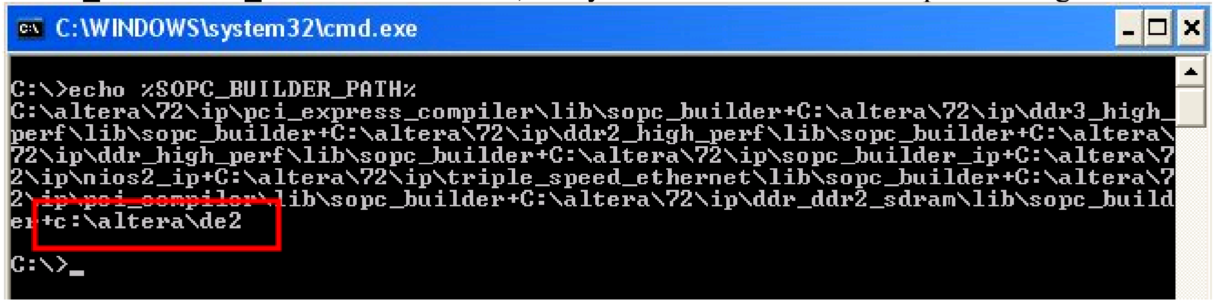

System Properties
General Computer Name Hardware
Advanced Automatic Updates Remote
You must be logged on as an Administrator to make most of these changes.
Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings
User Profiles
Desktop settings related to your logon
Settings
Startup and Recovery
System startup, system failure, and debugging information
Settings
Environment Variables Error Reporting
OK Cancel Apply
Environment Variables
New User Variable
OK Cancel
New Edit Delete
System variables
New Edit Delete
OK Cancel

Variable name: SOPC_BUILDER_PATH
Variable value: %SOPC_BUILDER_PATH%+c:\altera\de2

| Variable | Value |
|---|---|
| AltiumPath | C:\Program Files\Altium2004 SP2\System |
| CLASSPATH | .;C:\Program Files\Java\jre1.5.0_05\lib... |
| ComSpec | C:\WINDOWS\system32\cmd.exe |
| CST_WORK | C:\temp |
| ESHOME | C:\Program Files\Esterel Technologies\E... |

Figure 1-2 Adding new environment variable

4. Make the new variable name as **SOPC_BUILDER_PATH**, and the variable value as **%SOPC_BUILDER_PATH%+c:\altera\de2**, as shown in Figure 1-2. Then click OK. Click another OK from the Environment Variables window, and OK from the System Properties to apply the change.
5. You can check the change by going to the command prompt (Start->Run->cmd->OK). Type **echo %SOPC_BUILDER_PATH%** then enter, and you will be able to see output like Figure 1-3



```
C:\>echo %SOPC_BUILDER_PATH%
C:\altera\72\ip\pci_express_compiler\lib\sopc_builder+C:\altera\72\ip\ddr3_high_
perf\lib\sopc_builder+C:\altera\72\ip\ddr2_high_perf\lib\sopc_builder+C:\altera\
72\ip\ddr_high_perf\lib\sopc_builder+C:\altera\72\ip\sopc_builder_ip+C:\altera\7
2\ip\nios2_ip+C:\altera\72\ip\triple_speed_ethernet\lib\sopc_builder+C:\altera\7
2\ip\pci_compiler\lib\sopc_builder+C:\altera\72\ip\ddr_ddr2_sdram\lib\sopc_build
er+c:\altera\de2

C:\>_
```

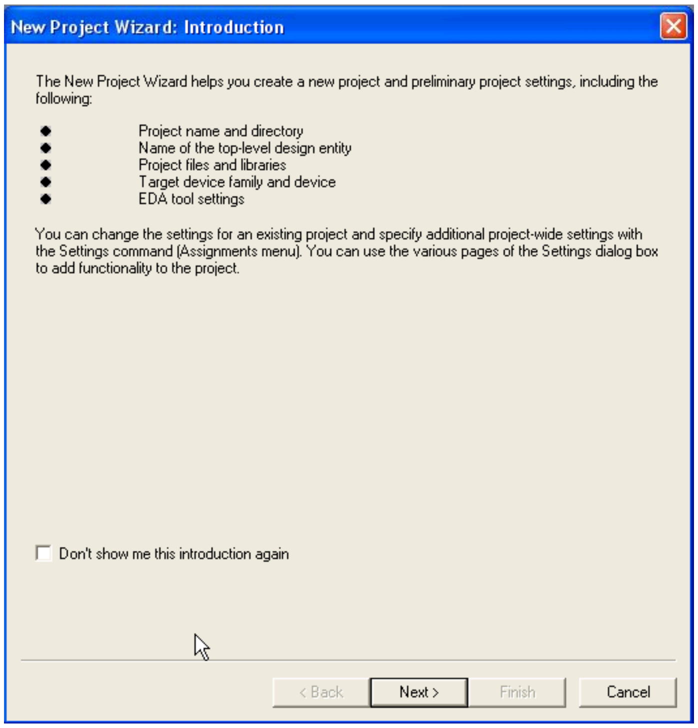
Figure 1-3: The new added environment variable

6. Now you are ready to proceed.

## 1. Hardware design of the DE2 Board

### 1.1 Create a Quartus II project and start a new design

Start Altera Quartus II (Start->Altera->Quartus II 7.2->Quartus II 7.2 (32-Bit)), select "New Project Wizard" from the file menu. Click "Next" in the introduction page as shown in Figure 1-1.



The New Project Wizard helps you create a new project and preliminary project settings, including the following:

- Project name and directory
- Name of the top-level design entity
- Project files and libraries
- Target device family and device
- EDA tool settings

You can change the settings for an existing project and specify additional project-wide settings with the Settings command (Assignments menu). You can use the various pages of the Settings dialog box to add functionality to the project.

Don't show me this introduction again

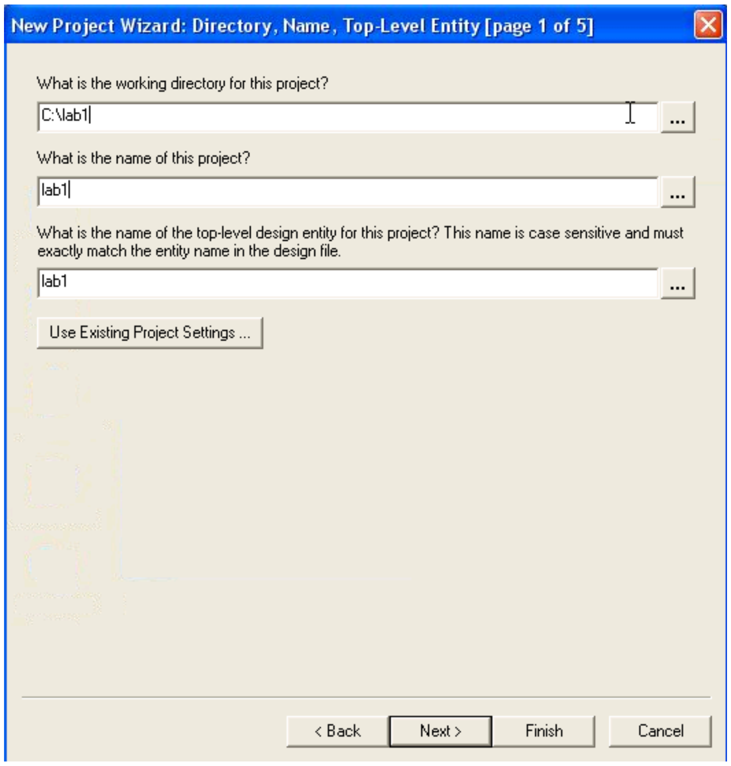
Figure 1-1: The introduction page

In the following step, as shown in Figure 1-2, which let you to decide the name of the directory where the design files will be located, and the name of the top-level entity design. We decided our files will be in the "**C:\lab1**", and the name of the project will be **lab1**. Note that the default name of the top-level design will be the same as the name of project. You are free to change any of these names, we just leave it as default here. Click "Next" after entering the corresponding fields. If the project directory does not exist, the wizard will ask you to create the particular directory, choose "Yes".

In the next step, the wizard will ask you to add file(s) to the project in Figure 1-3, click "Next" here whereas the necessary files will be added in the later sections of the tutorial.

Select "**Cyclone II**" as the device family in the drop-down box, and "**EP2C35F672C6**" as the target device in the "Family and Device Settings page" shown in the Figure 1-4. Click "Next" to next wizard page.

Since we are going to use Quartus II mainly for the course, click "Next" to let Quartus II be the synthesis, simulation, and analysis tools as default, this is shown in Figure 1-5. The final page of the wizard will show the project information briefly as in the Figure 1-6. You may click "Finish" to proceed.


New Project Wizard: Directory, Name, Top-Level Entity [page 1 of 5]

What is the working directory for this project?

C:\lab1

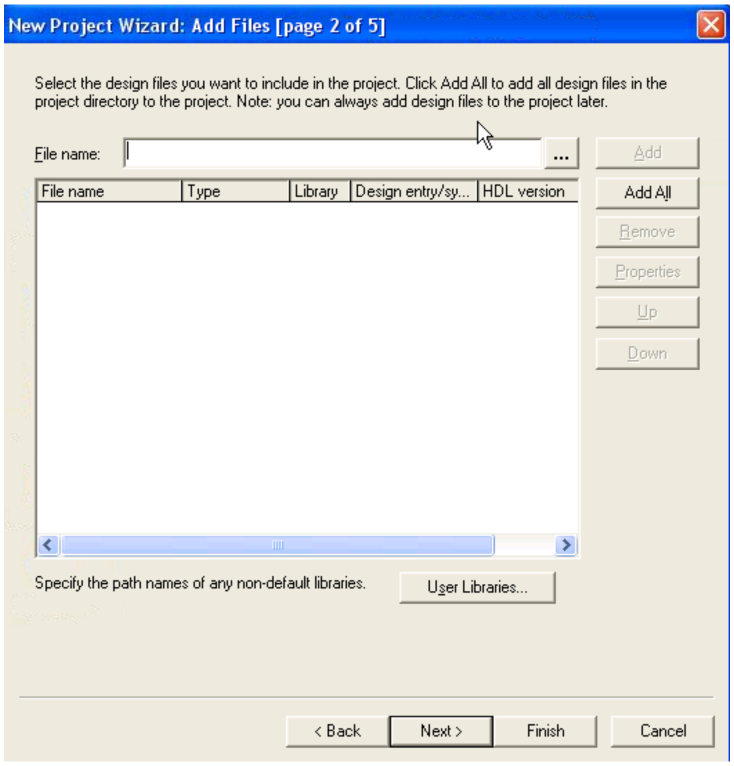
What is the name of this project?

lab1

What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

lab1

Figure 1-2: Project name and directory and the name of the top-level design


New Project Wizard: Add Files [page 2 of 5]

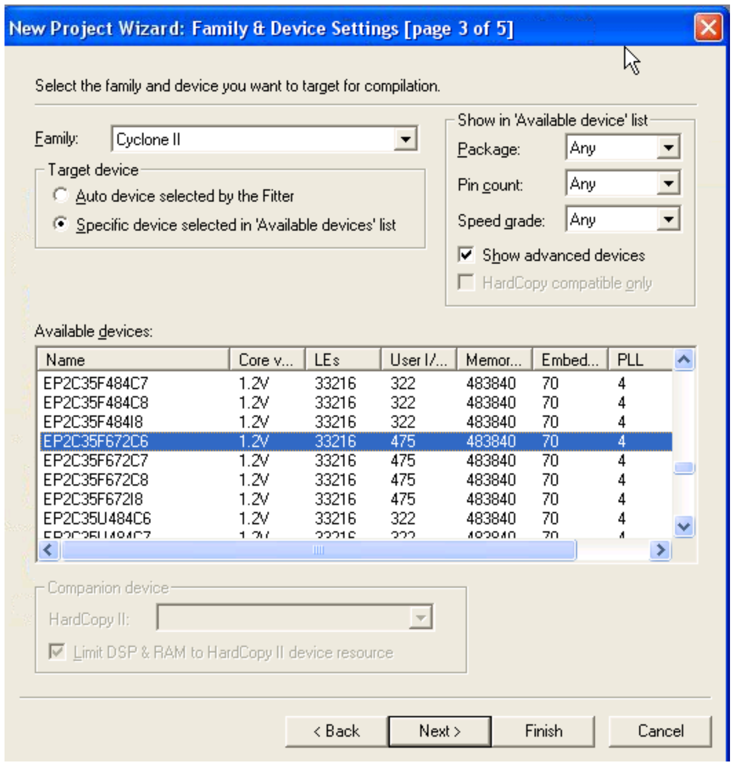
Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project. Note: you can always add design files to the project later.

File name:

| File name | Type | Library | Design entry/sy... | HDL version |
|---|---|---|---|---|

Specify the path names of any non-default libraries.

Figure 1-3: Adding files to the project



New Project Wizard: Family & Device Settings [page 3 of 5]

Select the family and device you want to target for compilation.

Family: Cyclone II

Target device
- [ ] Auto device selected by the Fitter
- [✓ Specific device selected in 'Available devices' list]

Show in 'Available device' list
- Package: Any
- Pin count: Any
- Speed grade: Any
- [✓] Show advanced devices
- [ ] HardCopy compatible only

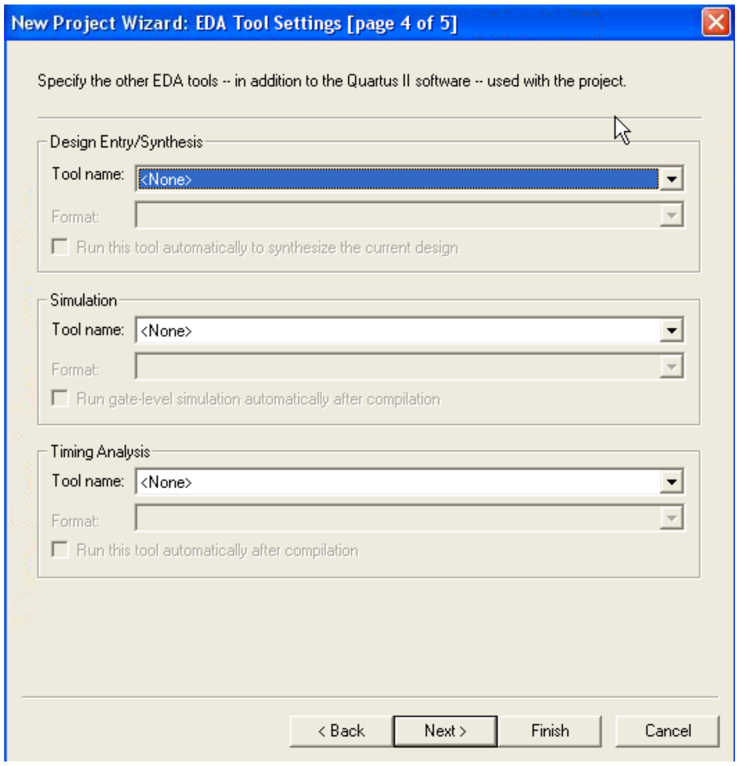
Available devices:

| Name | Core v... | LEs | User I/... | Memor... | Embed... | PLL |
|---|---|---|---|---|---|---|
| EP2C35F484C7 | 1.2V | 33216 | 322 | 483840 | 70 | 4 |
| EP2C35F484C8 | 1.2V | 33216 | 322 | 483840 | 70 | 4 |
| EP2C35F484I8 | 1.2V | 33216 | 322 | 483840 | 70 | 4 |
| EP2C35F672C6 | 1.2V | 33216 | 475 | 483840 | 70 | 4 |
| EP2C35F672C7 | 1.2V | 33216 | 475 | 483840 | 70 | 4 |
| EP2C35F672C8 | 1.2V | 33216 | 475 | 483840 | 70 | 4 |
| EP2C35F672I8 | 1.2V | 33216 | 475 | 483840 | 70 | 4 |
| EP2C35U484C6 | 1.2V | 33216 | 322 | 483840 | 70 | 4 |
| EP2C35U484C7 | 1.2V | 33216 | 322 | 483840 | 70 | 4 |

Companion device
- HardCopy II:
- [✓] Limit DSP & RAM to HardCopy II device resource

< Back | Next > | Finish | Cancel

Figure 1-4: Family and Device Settings dialog



New Project Wizard: EDA Tool Settings [page 4 of 5]

Specify the other EDA tools -- in addition to the Quartus II software -- used with the project.

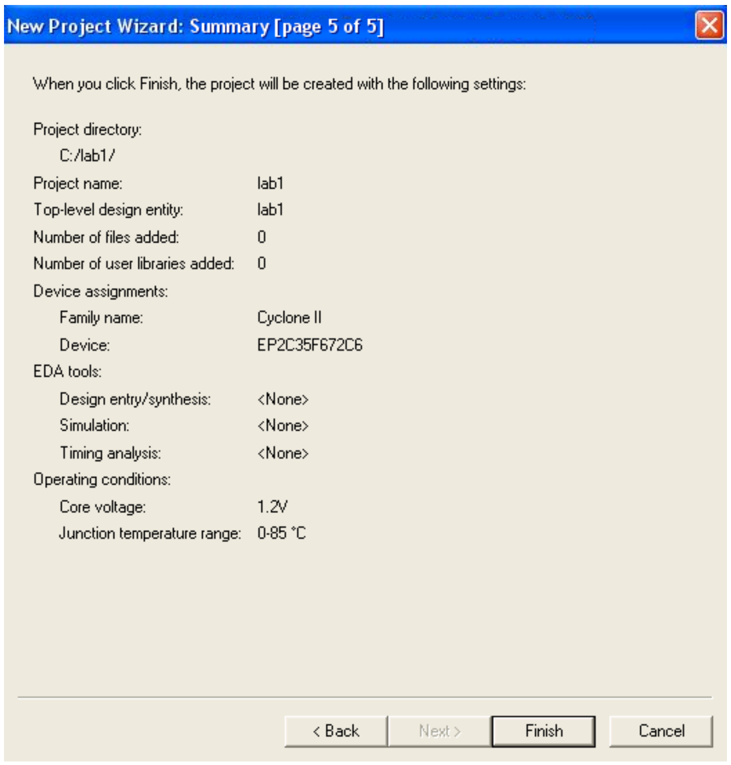
Design Entry/Synthesis
- Tool name: <None>
- Format:
- [ ] Run this tool automatically to synthesize the current design

Simulation
- Tool name: <None>
- Format:
- [ ] Run gate-level simulation automatically after compilation

Timing Analysis
- Tool name: <None>
- Format:
- [ ] Run this tool automatically after compilation

< Back | Next > | Finish | Cancel

Figure 1-5: EDA tools setting


New Project Wizard: Summary [page 5 of 5]

When you click Finish, the project will be created with the following settings:

| | |
|---|---|
| Project directory: | |
| C:/lab1/ | |
| Project name: | lab1 |
| Top-level design entity: | lab1 |
| Number of files added: | 0 |
| Number of user libraries added: | 0 |
| Device assignments: | |
| Family name: | Cyclone II |
| Device: | EP2C35F672C6 |
| EDA tools: | |
| Design entry/synthesis: | <None> |
| Simulation: | <None> |
| Timing analysis: | <None> |
| Operating conditions: | |
| Core voltage: | 1.2V |
| Junction temperature range: | 0-85 °C |

Figure 1-6: Project summary

The next step is to initiate NIOS II processor, which interconnects peripherals through dedicated interfaces. First create a schematic as our top level design by selecting "New" from the "File" menu. Choose "**Block Diagram/Schematic File**" and click "OK" as shown in Figure 1-7.

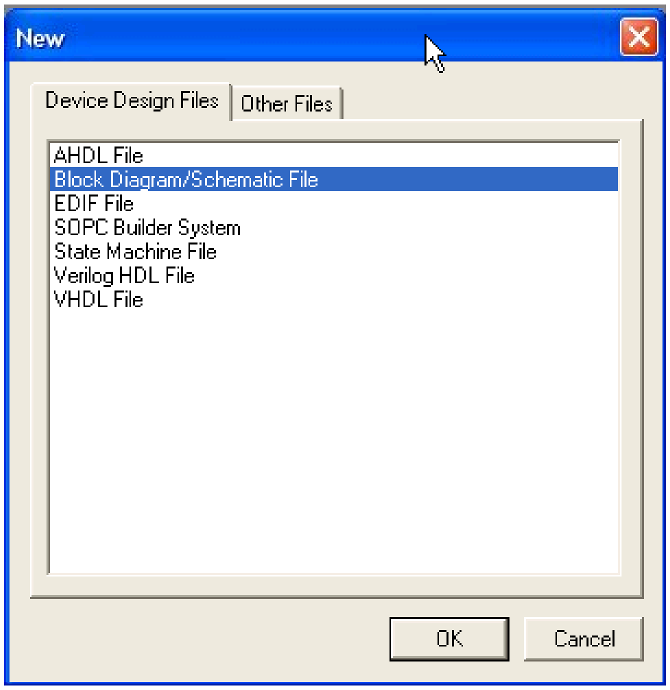


Figure 1-7: Create a new schematic file

The schematic design space will show up in the Quartus II main window, as illustrated in Figure 1-8. Users can carry out design procedures with available symbols of hardware components, and obtain integration of each components through wiring. Save the schematic (as **lab1.bdf** in this case) through "File->Save as" and tick "Add file to current project" in the file saving dialog as presented in Figure 1-9.

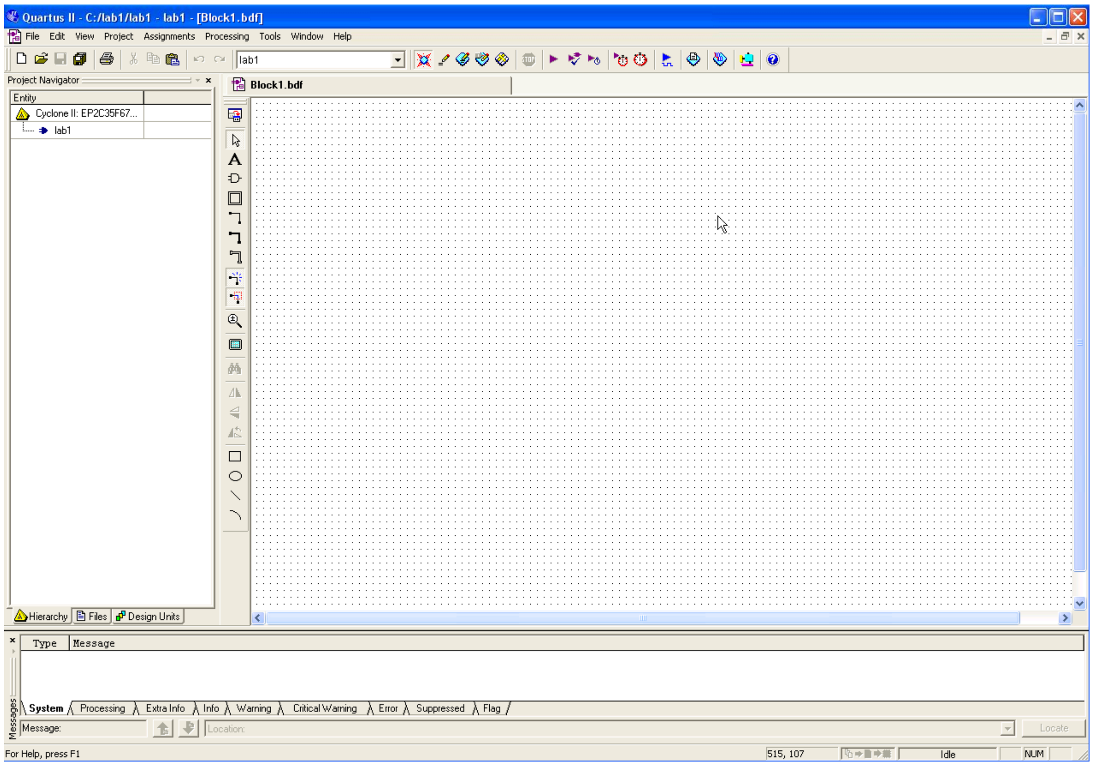
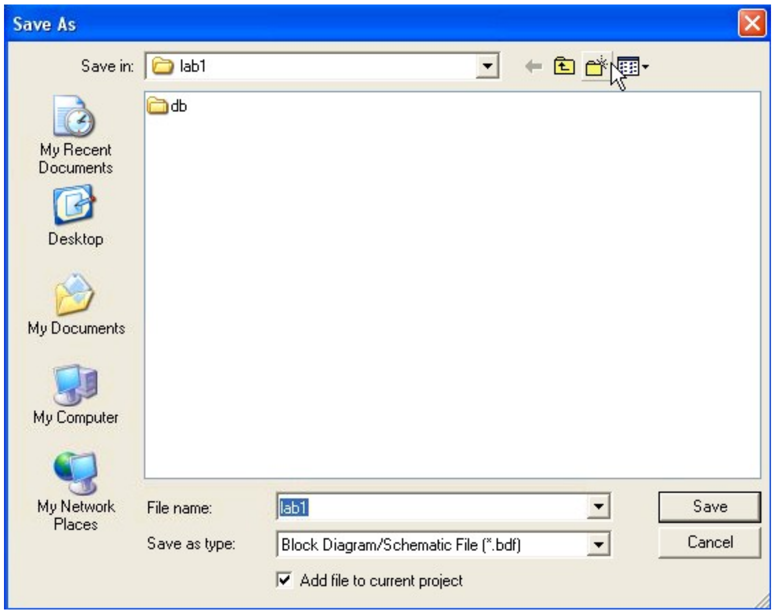

Quartus II - C:/lab1/lab1 - lab1 - [Block1.bdf]
File Edit View Project Assignments Processing Tools Window Help
lab1
Project Navigator
Entity
Cyclone II: EP2C35F67...
lab1
Hierarchy Files Design Units
Block1.bdf
Type Message
System Processing Extra Info Info Warning Critical Warning Error Suppressed Flag
Message:
Location:
Locate
For Help, press F1
515, 107
Idle
NUM

Figure 1-8: Quartus schematic design window



File name: lab1

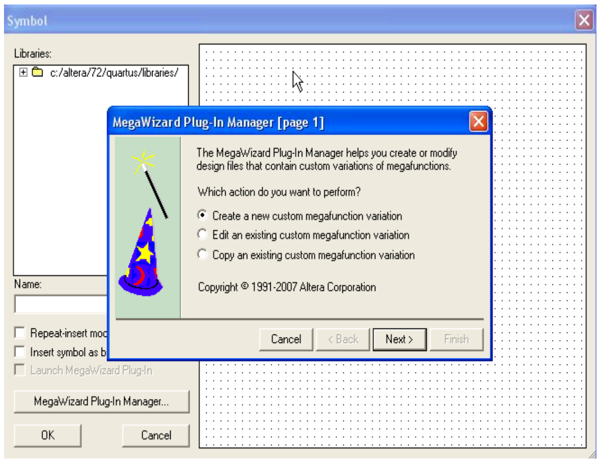
Save as type: Block Diagram/Schematic File (*.bdf)

☑ Add file to current project

Figure 1-9: Save the schematic and add it to the current project

## 1.2 SOPC Builder and Nios II system

We start the SOPC design through following steps, with these steps, a Nios II processor will be instantiated, and so will the peripherals and corresponding interfaces. **Double-click** on the schematic, a list of available component will be shown, click "**MegaWizard Plug-In Manager**"at the right lower conner. Choosing "**Create a new custom megafunction variation**" followed by clicking "Next". Choose "**Altera SOPC Builder**" under the Installed Plug-Ins, choose "**VHDL**" as the output type "**nios2_system**" after the project directory as the output filename and clicking "Next" , details are shown in the Figure 1-10 to Figure 1-12. Make sure you choose "Cyclone II" for the device family at the top right hand conner.



Symbol
Libraries:
c:/altera/72/quartus/libraries/
Name:
Repeat-insert mod
Insert symbol as b
Launch MegaWizard Plug-In
MegaWizard Plug-In Manager...
OK
Cancel

MegaWizard Plug-In Manager [page 1]

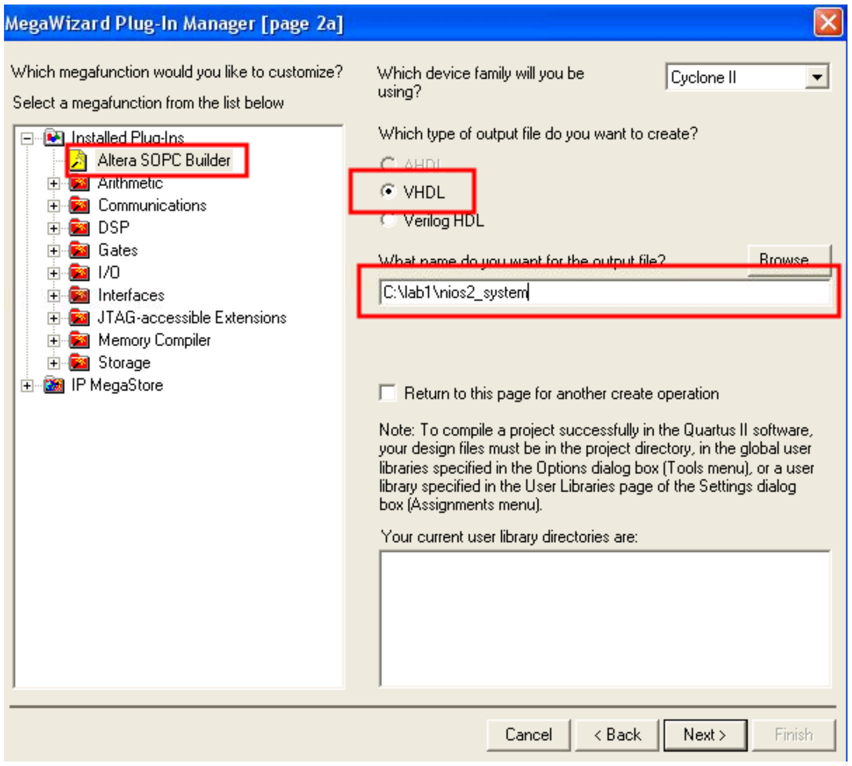
The MegaWizard Plug-In Manager helps you create or modify design files that contain custom variations of megafunctions.

Which action do you want to perform?

[✓ Create a new custom megafunction variation]
Edit an existing custom megafunction variation
Copy an existing custom megafunction variation

Copyright © 1991-2007 Altera Corporation

Figure 1-10: Create a new megafunction variation



MegaWizard Plug-In Manager [page 2a]

Which megafunction would you like to customize?
Select a megafunction from the list below

Which device family will you be using? Cyclone II

Which type of output file do you want to create?

AHDL
[✓ VHDL]
Verilog HDL

What name do you want for the output file? Browse...

C:\lab1\nios2_system

Return to this page for another create operation

Note: To compile a project successfully in the Quartus II software, your design files must be in the project directory, in the global user libraries specified in the Options dialog box (Tools menu), or a user library specified in the User Libraries page of the Settings dialog box (Assignments menu).

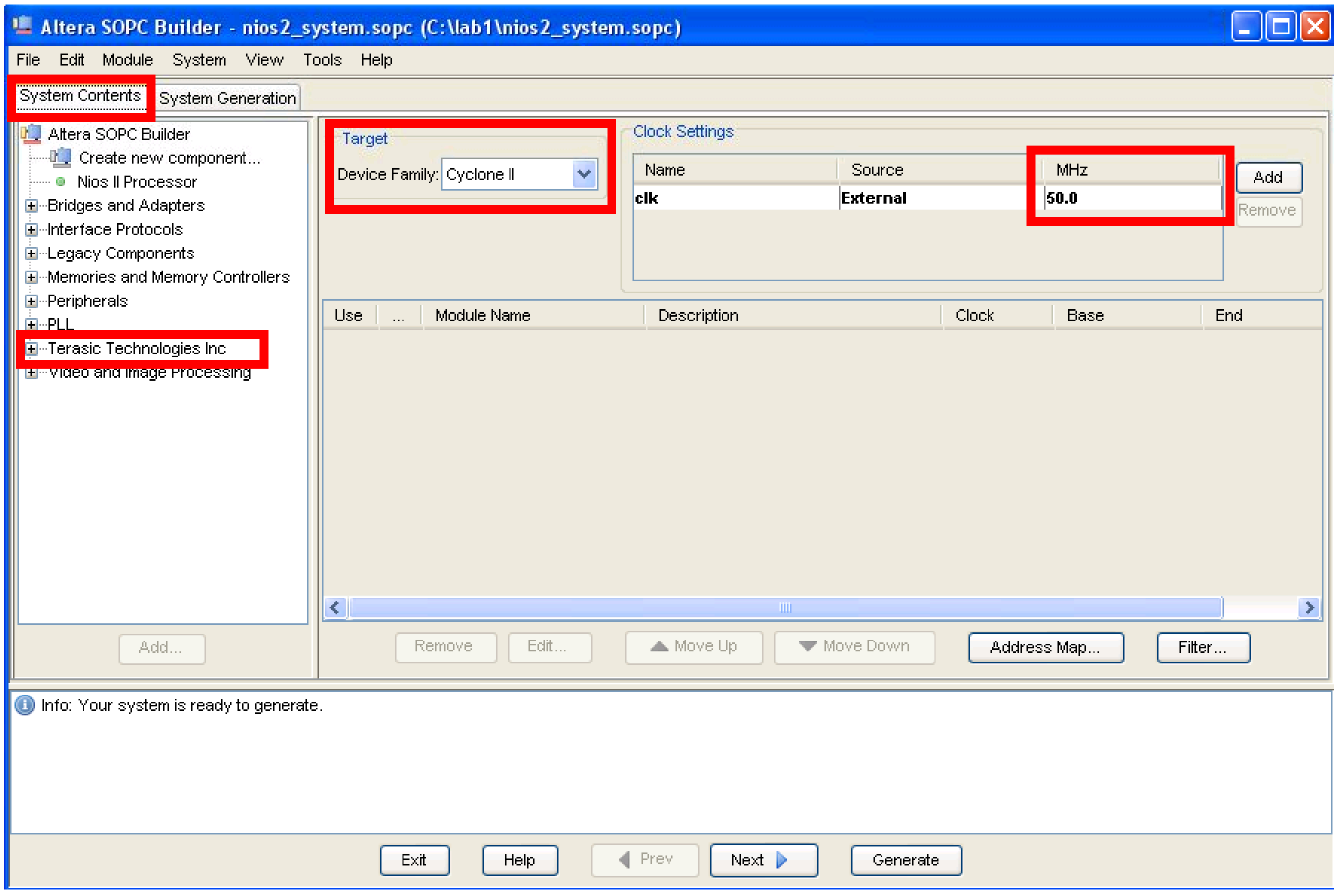
Your current user library directories are:

Figure 1-11: Fill required field for SOPC builder

Altera SOPC Builder 7.1sp1
Copyright ©1999-2007 Altera Corporation. All Rights Reserved.
Initializing Module Library...
Click for startup time tips...

Figure 1-12: Starting up the Altera SOPC Builder

Once the SOPC Builder starts up, use “Cyclone II” for the “Target” section as shown in Figure 1-13. The default clock value is **50MHz**, which is provided by the oscillator on DE2 board (**Y1** from the schematic, located at left hand side of the FPGA chip). You should be able to see a section under the System Contents named **Terasic Technologies Inc** if there was no problem in section 0.



Target
Device Family: Cyclone II

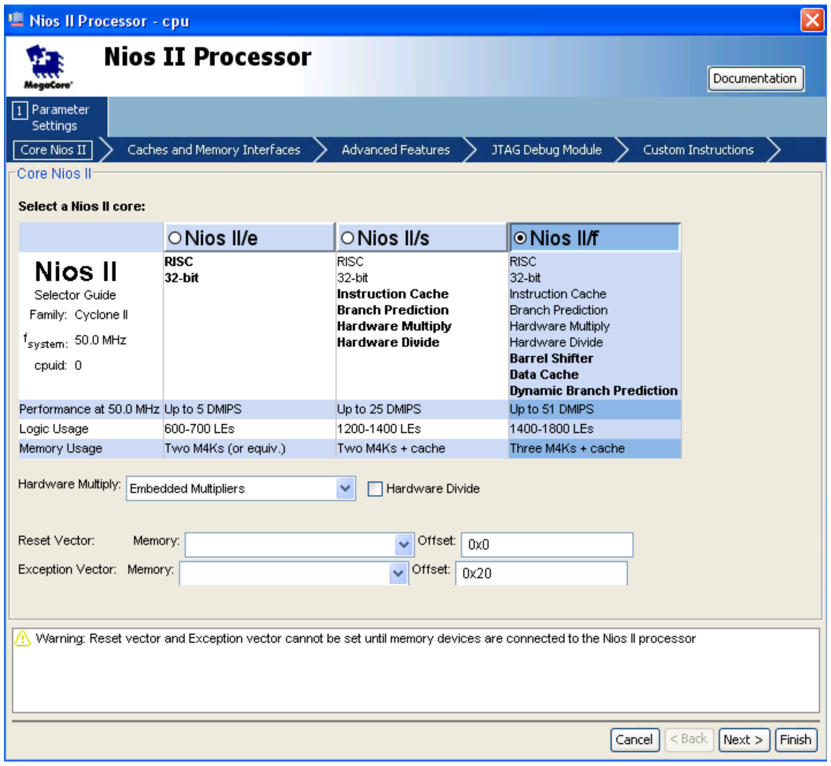
| Name | Source | MHz |
|---|---|---|
| clk | External | 50.0 |

Info: Your system is ready to generate.

Figure 1-13: Settings of the DE2 board in the SOPC Builder

### 1.2.1 Add the Nios II processor

Nios II is the centralized unit of our target system, it can be added by selecting "**Nios II Processor**" under the "**Altera SOPC Builder**" and clicking "**Add**" to include the processor. Select **Nios II/f** in this case, which is the most powerful (and consumes the most logic elements among the three Nios configurations) as shown in Figure 1-14. We are going to use the default settings of this processor, click "**Finish**" straight ahead. Note that we will come back later to set the memory allocation for processor to use afterward.



Select a Nios II core:

| Nios II<br>Selector Guide<br>Family: Cyclone II<br>$f_{system}$: 50.0 MHz<br>cpuid: 0 | ○ Nios II/e | ○ Nios II/s | [✓ Nios II/f] |
|---|---|---|---|
| | RISC<br>32-bit | RISC<br>32-bit<br>Instruction Cache<br>Branch Prediction<br>Hardware Multiply<br>Hardware Divide | RISC<br>32-bit<br>Instruction Cache<br>Branch Prediction<br>Hardware Multiply<br>Hardware Divide<br>Barrel Shifter<br>Data Cache<br>Dynamic Branch Prediction |
| Performance at 50.0 MHz | Up to 5 DMIPS | Up to 25 DMIPS | Up to 51 DMIPS |
| Logic Usage | 600-700 LEs | 1200-1400 LEs | 1400-1800 LEs |
| Memory Usage | Two M4Ks (or equiv.) | Two M4Ks + cache | Three M4Ks + cache |

Hardware Multiply: Embedded Multipliers ☐ Hardware Divide

Reset Vector: Memory: Offset: 0x0

Exception Vector: Memory: Offset: 0x20

⚠ Warning: Reset vector and Exception vector cannot be set until memory devices are connected to the Nios II processor

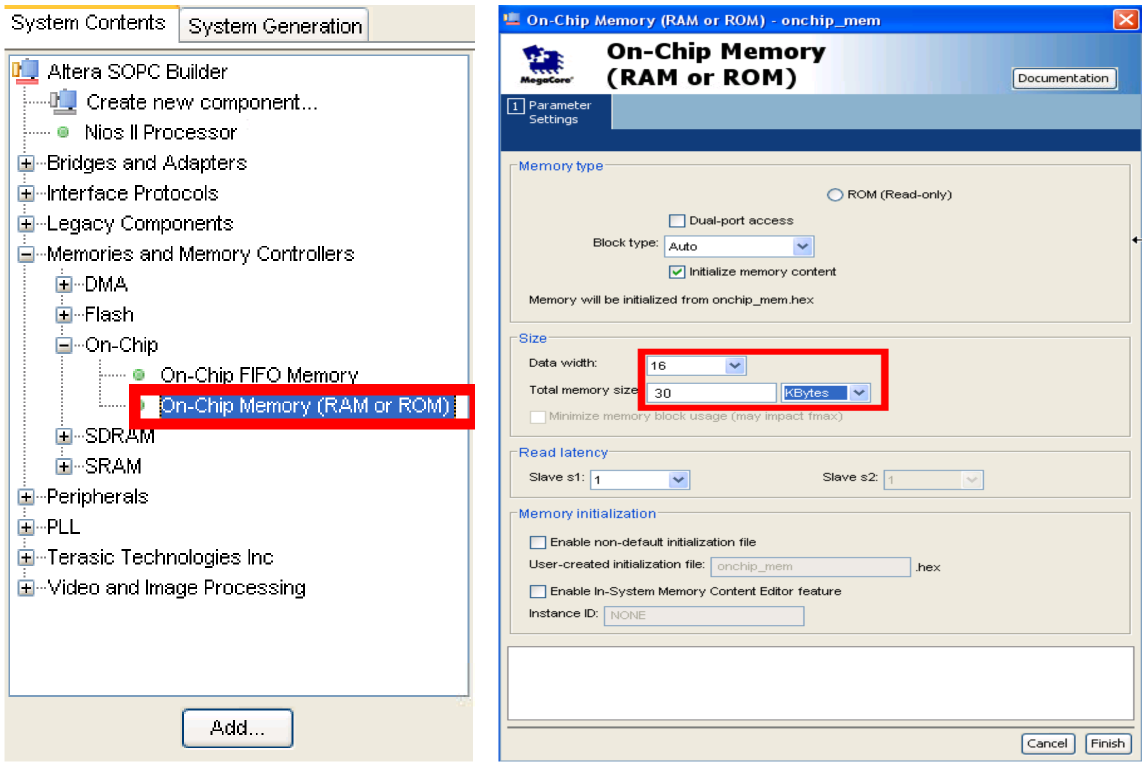
Figure 1-14: Nios II configurations

### 1.2.2 Add On-Chip RAM

We can use memory bits **within the FPGA board** to create **on-chip memory** to be used by the processor. Go under "**Altera SOPC Builder->Memories and Memory Controllers->On-chip**" and click "Add" on "**On-Chip Memory (RAM or ROM)**" as shown in Figure 1-15. Allocate **30KB** of RAM memory space according to Figure 1-15. Note that you might want to decrease the size of on chip RAM in your future designs, since memory bits are precious.You might find some errors regarding to memory settings, we will resolve that in the later stage.

### 1.2.3 Add the SDRAM module

Memories are used by Nios II to store programs and data. In this design, the program will be loaded to the **8MB SDRAM** on the DE2 board. To use the SDRAM, add the "**SDRAM Controller**" under the "**Altera SOPC Builder->Memories and Memory Controllers->SDRAM** ". Modify the presets to "**Custom**" and change the "Data width" to "**16 bits**". The rest of the settings are illustrated in Figure 1-16. Click "Finish" to add the SDRAM module.


System Contents | System Generation

- Altera SOPC Builder
  - Create new component...
  - Nios II Processor
  - Bridges and Adapters
  - Interface Protocols
  - Legacy Components
  - Memories and Memory Controllers
    - DMA
    - Flash
    - On-Chip
      - On-Chip FIFO Memory
      - On-Chip Memory (RAM or ROM)
    - SDRAM
    - SRAM
  - Peripherals
  - PLL
  - Terasic Technologies Inc
  - Video and Image Processing

Add...

On-Chip Memory (RAM or ROM) - onchip_mem

On-Chip Memory (RAM or ROM)

Documentation

1 Parameter Settings

Memory type

ROM (Read-only)

Dual-port access

Block type: Auto

[✓] Initialize memory content

Memory will be initialized from onchip_mem.hex

Size

Data width: 16

Total memory size: 30 KBytes

Minimize memory block usage (may impact fmax)

Read latency

Slave s1: 1 Slave s2: 1

Memory initialization

Enable non-default initialization file

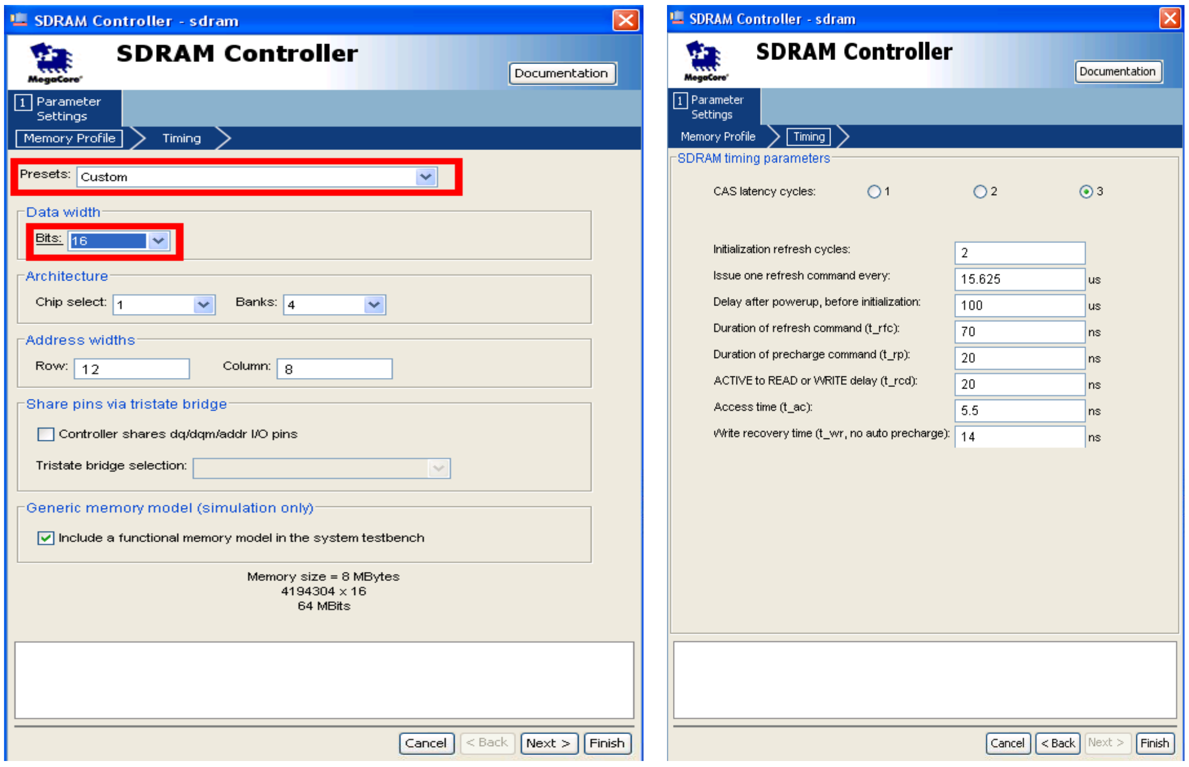
User-created initialization file: onchip_mem .hex

Enable In-System Memory Content Editor feature

Instance ID: NONE

Cancel Finish

Figure 1-15 Adding On-Chip RAM to the nios2 system


SDRAM Controller - sdram

SDRAM Controller

Documentation

1 Parameter Settings

Memory Profile > Timing

Presets: Custom

Data width

Bits: 16

Architecture

Chip select: 1 Banks: 4

Address widths

Row: 12 Column: 8

Share pins via tristate bridge

Controller shares dq/dqm/addr I/O pins

Tristate bridge selection:

Generic memory model (simulation only)

[✓] Include a functional memory model in the system testbench

Memory size = 8 MBytes
4194304 x 16
64 MBits

Cancel < Back Next > Finish

SDRAM Controller - sdram

SDRAM Controller

Documentation

1 Parameter Settings

Memory Profile > Timing

SDRAM timing parameters

CAS latency cycles: 1 2 [✓ 3]

Initialization refresh cycles: 2

Issue one refresh command every: 15.625 us

Delay after powerup, before initialization: 100 us

Duration of refresh command (t_rfc): 70 ns

Duration of precharge command (t_rp): 20 ns

ACTIVE to READ or WRITE delay (t_rcd): 20 ns

Access time (t_ac): 5.5 ns

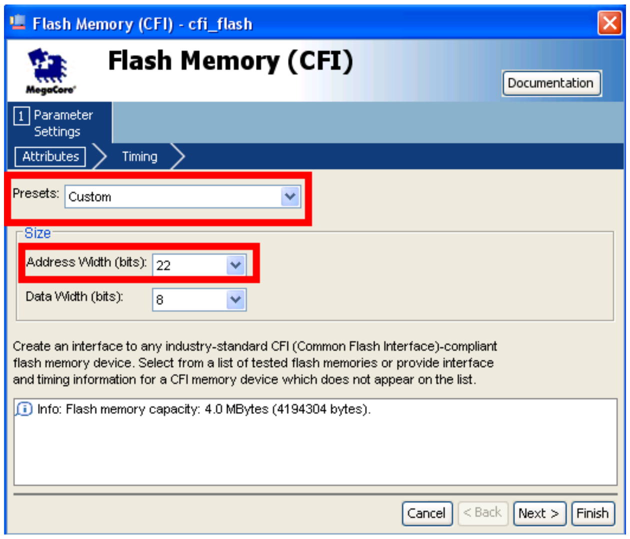
Write recovery time (t_wr, no auto precharge): 14 ns

Cancel < Back Next > Finish

Figure 1-16: The settings of The SDRAM controller

### 1.2.4 Add the Flash memory module and the tri-state Avalon bridge

A 4MB Flash memory on the DE2 board will be used to stored the configuration of the hardware design (which will be loaded into the FGPA) and the software (which will be loaded to the SDRAM and executed by the Nios II). It can be added by selecting "**Altera SOPC Builder->Memories and Memory Controllers->Flash->Flash Memory (CFI)**". Figure 1-17 (a) and (b) shows the settings of the the flash memory. Change the presets to "Custom" and the Address Width to "22" to obtain a 4 MB sized flash memory capacity. In the timing tab, set the **Setup time to 40 ns, Wait time to 160 ns, and the Hold time to 40 ns.** Note that in order to use the flash memory, the Avalon tristate bridge will be required. Add the tristate bridge through "**Altera SOPC Builder->Bridges and Adapters->Memory Mapped->Avalon-MM Tristate Bridge**" The detail settings of the tristate bridge are in Figure 1-18. Also note that you have to connect the flash memory with the tristate master of the tristate bridge, you can do that by **moving the mouse near the s1 of the flash, then click the circle to make it become as a solid one** as shown in Figure 1-19.



Presets: Custom

Size

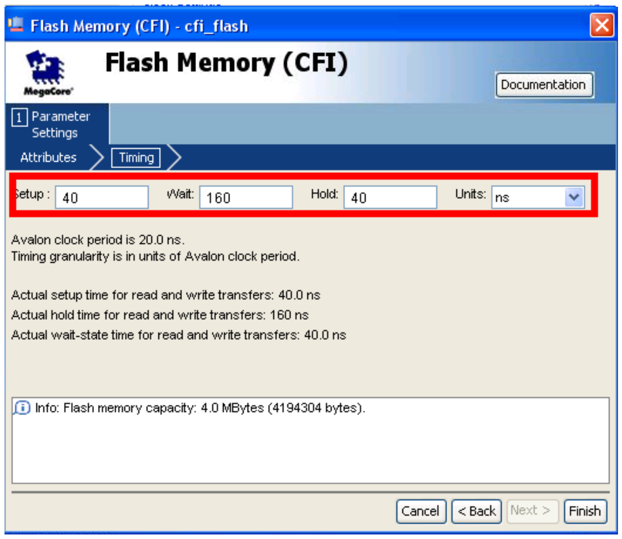
Address Width (bits): 22

Data Width (bits): 8

Create an interface to any industry-standard CFI (Common Flash Interface)-compliant flash memory device. Select from a list of tested flash memories or provide interface and timing information for a CFI memory device which does not appear on the list.

Info: Flash memory capacity: 4.0 MBytes (4194304 bytes).

Figure 1-17 (a): Attributes of the flash memory



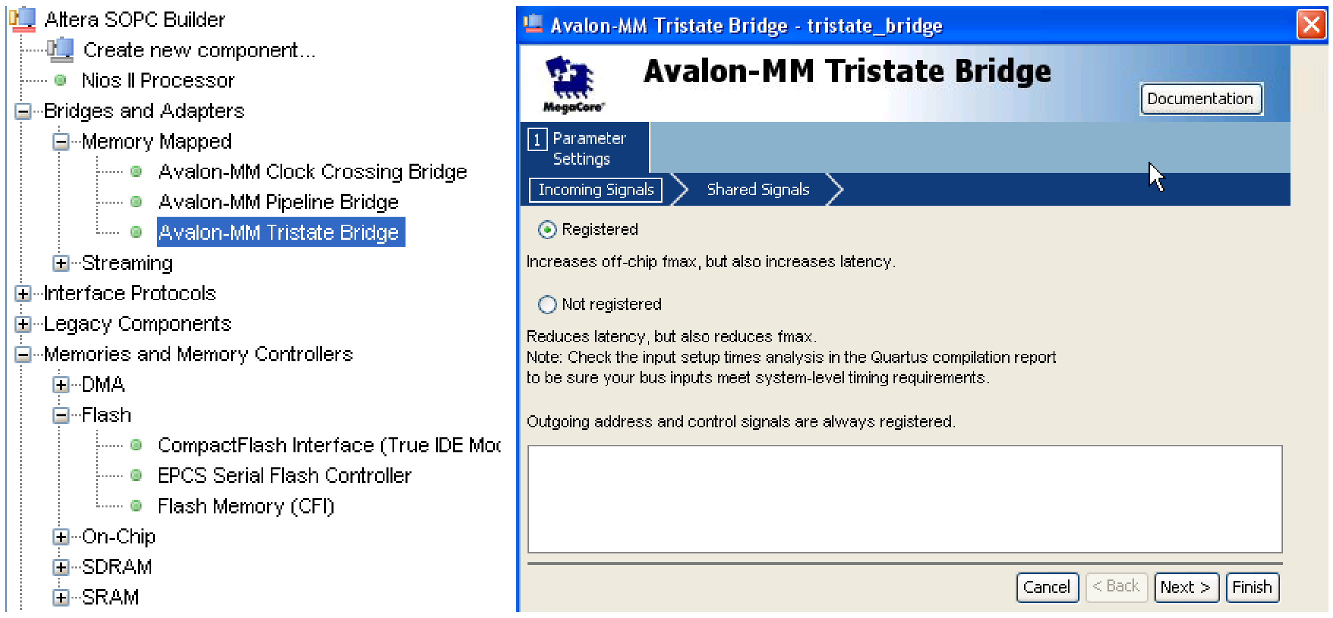
Setup : 40 Wait: 160 Hold: 40 Units: ns

Avalon clock period is 20.0 ns.
Timing granularity is in units of Avalon clock period.

Actual setup time for read and write transfers: 40.0 ns
Actual hold time for read and write transfers: 160 ns
Actual wait-state time for read and write transfers: 40.0 ns

Info: Flash memory capacity: 4.0 MBytes (4194304 bytes).

Figure 1-17 (b): Timing of the flash memory



- Altera SOPC Builder
  - Create new component...
  - Nios II Processor
  - Bridges and Adapters
    - Memory Mapped
      - Avalon-MM Clock Crossing Bridge
      - Avalon-MM Pipeline Bridge
      - Avalon-MM Tristate Bridge
    - Streaming
  - Interface Protocols
  - Legacy Components
  - Memories and Memory Controllers
    - DMA
    - Flash
      - CompactFlash Interface (True IDE Moc
      - EPCS Serial Flash Controller
      - Flash Memory (CFI)
    - On-Chip
    - SDRAM
    - SRAM

Avalon-MM Tristate Bridge - tristate_bridge

**Avalon-MM Tristate Bridge**

[✓ Registered]

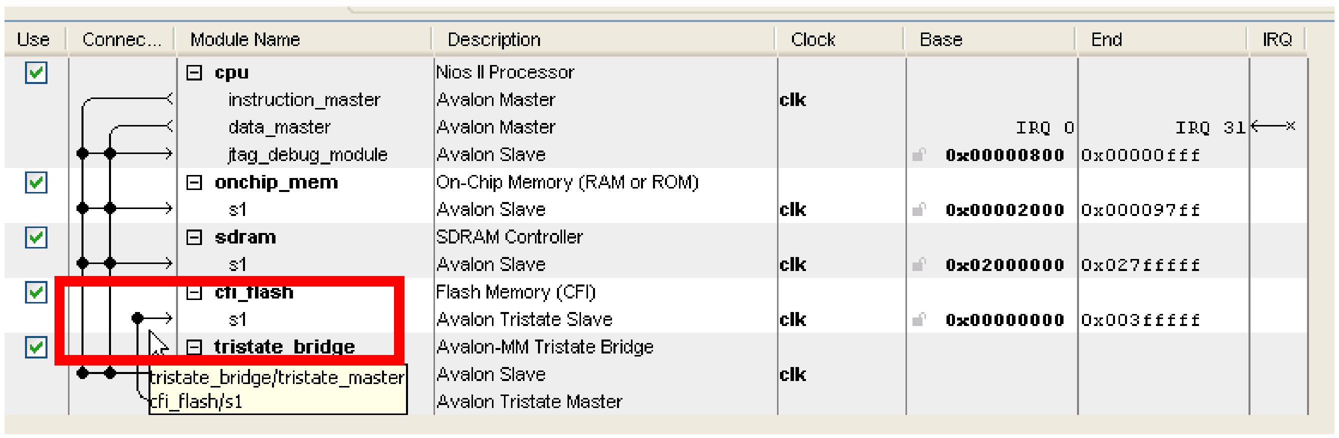
Increases off-chip fmax, but also increases latency.

Not registered

Reduces latency, but also reduces fmax.
Note: Check the input setup times analysis in the Quartus compilation report to be sure your bus inputs meet system-level timing requirements.

Outgoing address and control signals are always registered.

Figure 1-18: Avalon-MM Tristate Bridge



| Use | Connec... | Module Name | Description | Clock | Base | End | IRQ |
|---|---|---|---|---|---|---|---|
| ✓ | | cpu | Nios II Processor | | | | |
| | | instruction_master | Avalon Master | clk | | | |
| | | data_master | Avalon Master | | IRQ 0 | IRQ 31 | |
| | | jtag_debug_module | Avalon Slave | | 0x00000800 | 0x00000fff | |
| ✓ | | onchip_mem | On-Chip Memory (RAM or ROM) | | | | |
| | | s1 | Avalon Slave | clk | 0x00002000 | 0x000097ff | |
| ✓ | | sdram | SDRAM Controller | | | | |
| | | s1 | Avalon Slave | clk | 0x02000000 | 0x027fffff | |
| ✓ | | cfi_flash | Flash Memory (CFI) | | | | |
| | | s1 | Avalon Tristate Slave | clk | 0x00000000 | 0x003fffff | |
| ✓ | | tristate_bridge | Avalon-MM Tristate Bridge | | | | |
| | | tristate_bridge/tristate_master | Avalon Slave | clk | | | |
| | | cfi_flash/s1 | Avalon Tristate Master | | | | |

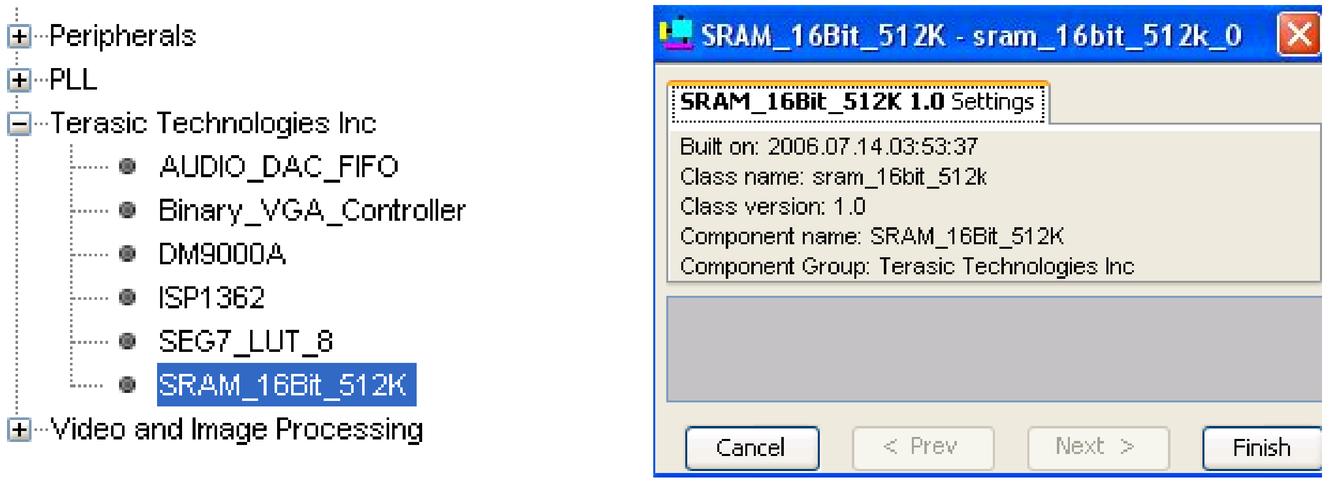
Figure 1-19: Connecting the flash with the tristate bridge

### 1.2.5 Add the SRAM module

We can also utilize the 512KB SRAM on the DE2 board through adding "**SRAM_16Bit_512K**" under the "**Terasic Technologies Inc**" as shown in Figure 1-20. **Rename** (**Note that upper case and lower case are critical in SOPC builder**) the component to **sram_0** (this step is optional) by right click on the "sdram_16bit_512k" in the "Module Name" column, select "Rename", and then typing in "sram_0" as desired.

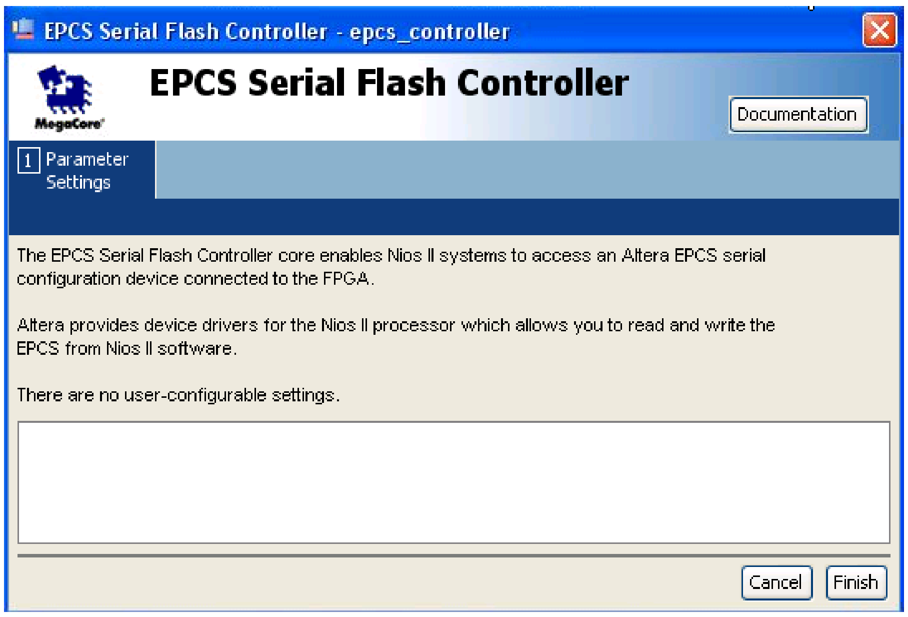
### 1.2.6 Add the EPCS Serial Flash controller

The EPCS serial flash controller on the DE2 board can be included to the system through "**Altera SOPC Builder->Memories and Memory Controllers->Flash->EPCS Serial Flash Controller**" as shown in Figure 1-21. Rename the component to "**epcs_controller**"



Built on: 2006.07.14.03:53:37
Class name: sram_16bit_512k
Class version: 1.0
Component name: SRAM_16Bit_512K
Component Group: Terasic Technologies Inc

Figure 1-20: Include SRAM component to the system


EPCS Serial Flash Controller - epcs_controller
EPCS Serial Flash Controller
MegaCore
Documentation
1 Parameter Settings
Cancel
Finish

The EPCS Serial Flash Controller core enables Nios II systems to access an Altera EPCS serial configuration device connected to the FPGA.

Altera provides device drivers for the Nios II processor which allows you to read and write the EPCS from Nios II software.

There are no user-configurable settings.

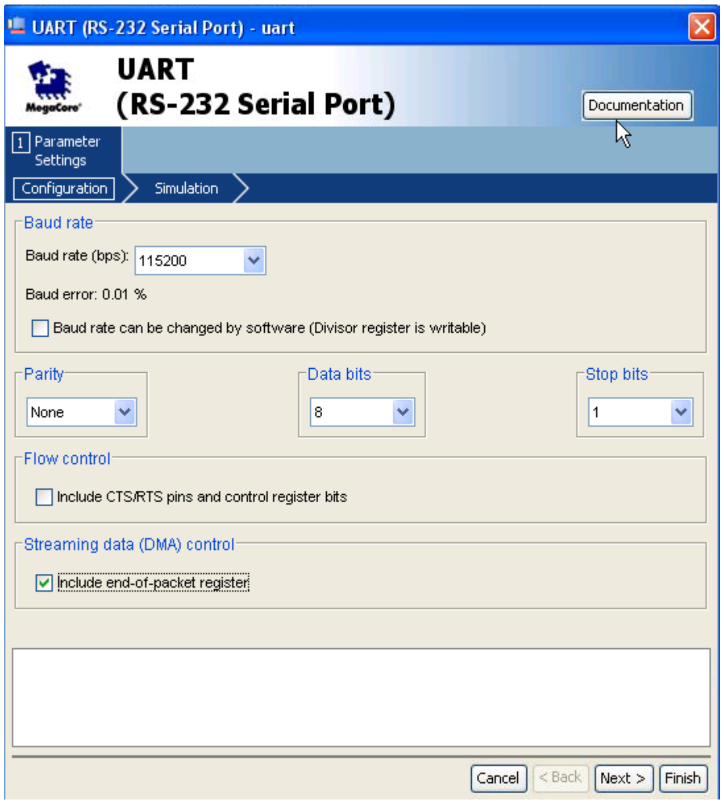
Figure 1-21: Add EPCS serial flash controller

### 1.2.7 Add the UART port support

To use the UART port on the DE2 development board, simply select "**UART (RS-232 serial port)**" under "**Altera SOPC Builder->Interface Protocols->Serial**", set **Baud rate** to **115200**, **Parity** to **None**, **Data bits** to **8**, **Stop bits** to **1**, and tick the "**Include end-of-packet register**" detailed in Figure 1-22. **Flow control is unavailable** since the CTS/RTS are not connected on the DE2 board. Click "Finish" to add the component.

### 1.2.8 Add the JTAG UART support

It is good to add the JTAG UART support to the system, which is very useful in the debugging process of software development on Nios II where the input/output of the Nios II terminal are via this components. "**JTAG UART**" is located under the "**Altera SOPC Builder->Interface Protocols->Serial** " section, and details of the settings are in Figure 1-23.



Baud rate

Baud rate (bps): 115200

Baud error: 0.01 %

☐ Baud rate can be changed by software (Divisor register is writable)

Parity: None

Data bits: 8

Stop bits: 1

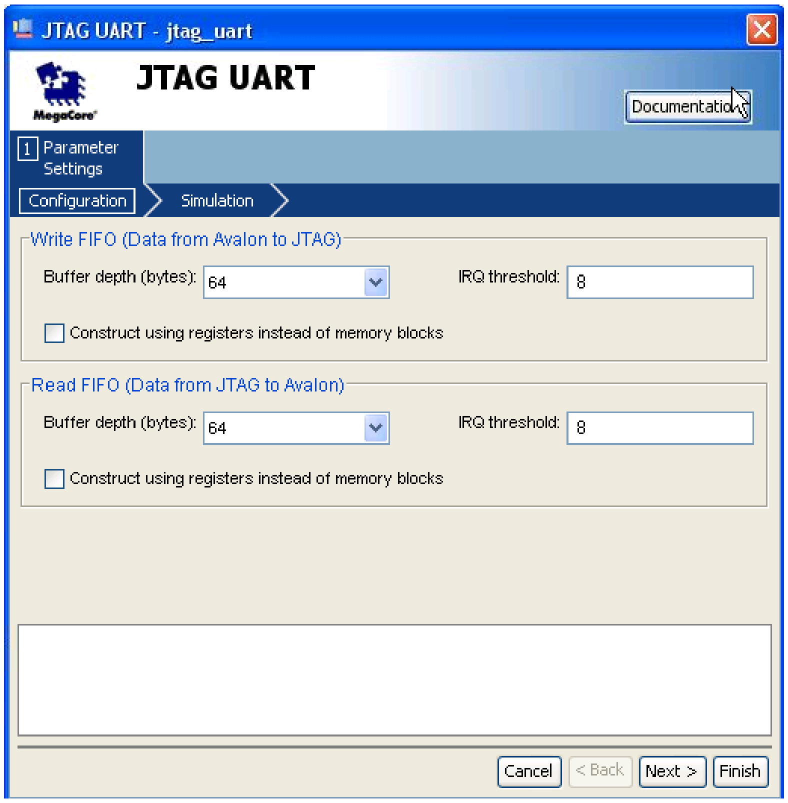
Flow control

☐ Include CTS/RTS pins and control register bits

Streaming data (DMA) control

☑ Include end-of-packet register

Figure 1-22: Settings of the UART port



Write FIFO (Data from Avalon to JTAG)

Buffer depth (bytes): 64 IRQ threshold: 8

☐ Construct using registers instead of memory blocks

Read FIFO (Data from JTAG to Avalon)

Buffer depth (bytes): 64 IRQ threshold: 8

☐ Construct using registers instead of memory blocks

Figure 1-23: Configuration of the JTAG UART

### 1.2.9 Add the interval timers

Timers are useful for some software, which can be added from "**Altera SOPC Builder->Peripherals->Micro controller Peripherals**" by selecting the "**Interval timer**". Two timers of **periods of 1 ms** are added in this case which  are shown in Figure 1-24.

### 1.2.10 Add the 3 push button PIOs

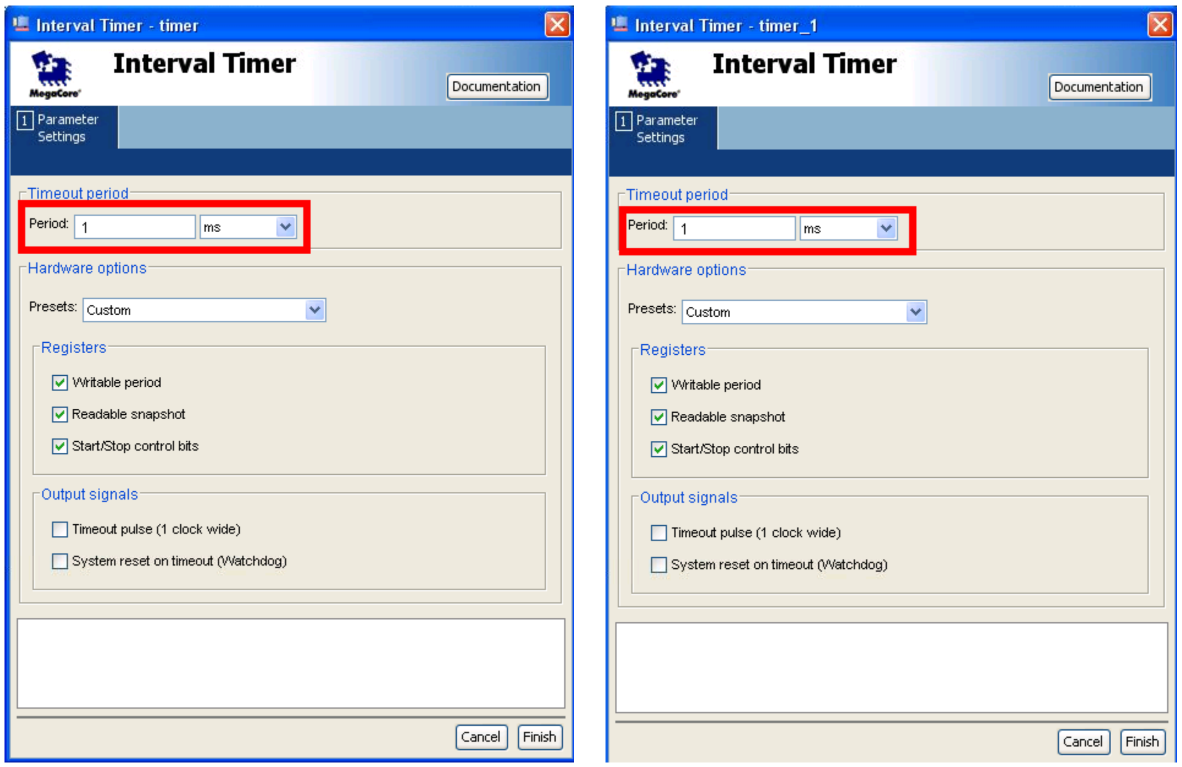
4 push buttons on the DE2 development board are accessed through parallel input ports. Each bit present a single button on the DE2. When a button is pushed, an interrupt will be generated on the falling edge. Selected "**PIO (Parallel I/O)**" from "**Altera SOPC Builder->Peripherals->Micro controller Peripherals**" The settings of the push button ports are as follows, **Width is 3 bits, Input port only, Synchronously capture at Falling edge, Generate IRQ at Edge**, which are also detailed in Figure 1-25. Rename the added PIO to **buttons**.

### 1.2.11 Add a PIO module to access 18 switches

Similar to push buttons, 18 switches which are left to the push buttons are facilitated as parallel input ports. Configurations of switches are illustrated in Figure 1-26. Once add the component, rename it to **switches**. One of the difference between switches and buttons is that, **interrupts** are enabled in the buttons but **not for switches**.

### 1.2.12 Add the LED modules as parallel output

The 18 red and 9 green LEDs are parts of parallel inputs and outputs of the system, which can be included by adding "**PIO (Parallel I/O)**" under the "**Altera SOPC Builder->Peripherals->Micro controller Peripherals** " section. The red and green LEDs are **output port only** as detailed in Figure 1-27. Rename the PIO modules of the red and green LEDs to **led_red** and **led_green** respectively.



Timeout period
Period: 1 ms

Hardware options
Presets: Custom

Registers
☑ Writable period
☑ Readable snapshot
☑ Start/Stop control bits

Output signals
☐ Timeout pulse (1 clock wide)
☐ System reset on timeout (Watchdog)

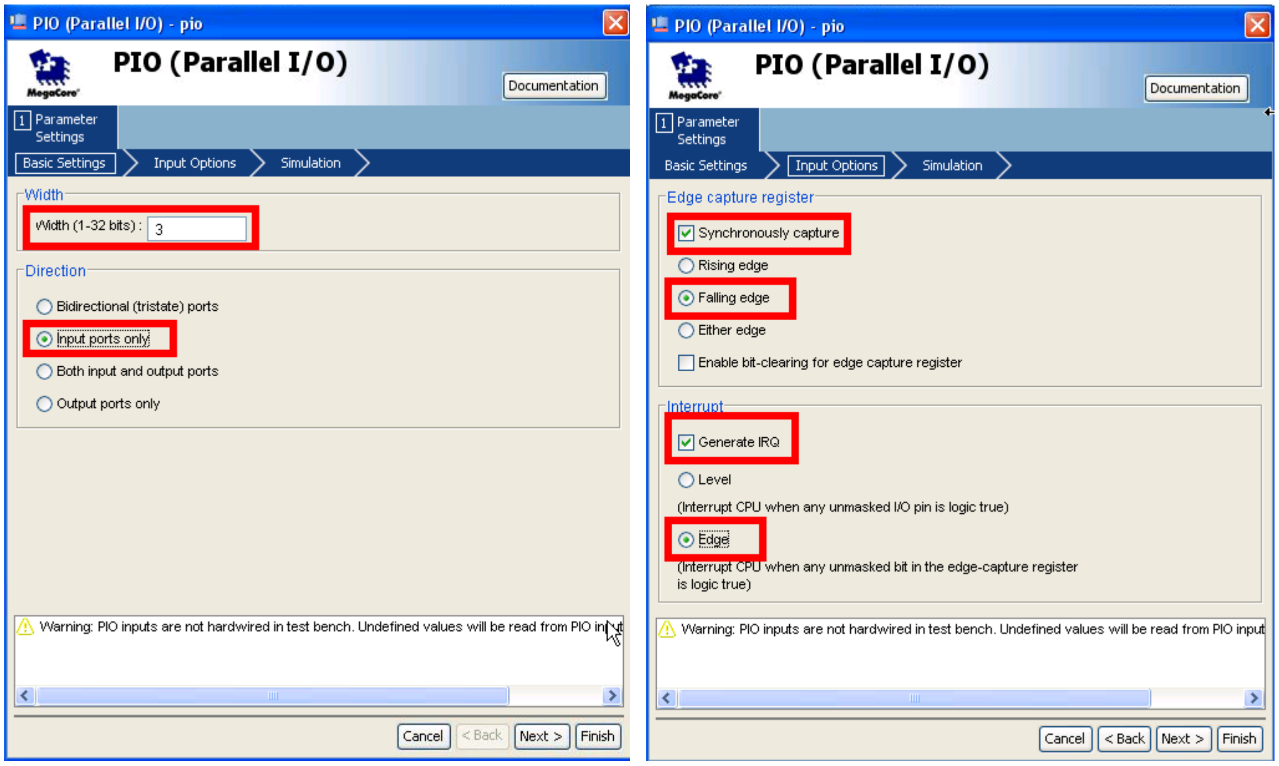
Timeout period
Period: 1 ms

Hardware options
Presets: Custom

Registers
☑ Writable period
☑ Readable snapshot
☑ Start/Stop control bits

Output signals
☐ Timeout pulse (1 clock wide)
☐ System reset on timeout (Watchdog)

Figure 1-24: Configuration of two interval timers



Width

Width (1-32 bits) : 3

Direction

- ○ Bidirectional (tristate) ports
- [✓ Input ports only]
- ○ Both input and output ports
- ○ Output ports only

Warning: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO input

Edge capture register

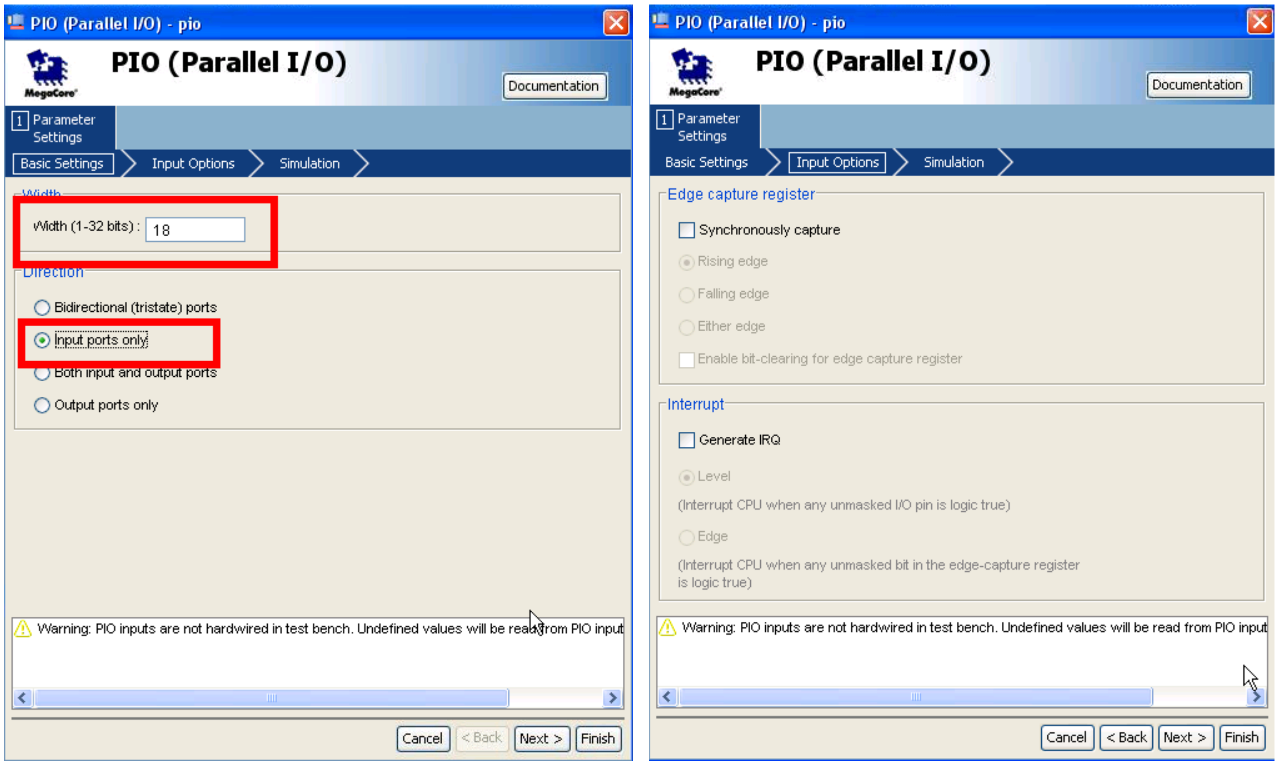
- [✓] Synchronously capture
- ○ Rising edge
- [✓ Falling edge]
- ○ Either edge
- [ ] Enable bit-clearing for edge capture register

Interrupt

- [✓] Generate IRQ
- ○ Level
  (Interrupt CPU when any unmasked I/O pin is logic true)
- [✓ Edge]
  (Interrupt CPU when any unmasked bit in the edge-capture register is logic true)

Warning: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO input

Figure 1-25: Configuration of the 3 push buttons



Width

Width (1-32 bits) : 18

Direction

- ○ Bidirectional (tristate) ports
- [✓ Input ports only]
- ○ Both input and output ports
- ○ Output ports only

Warning: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO input

Edge capture register

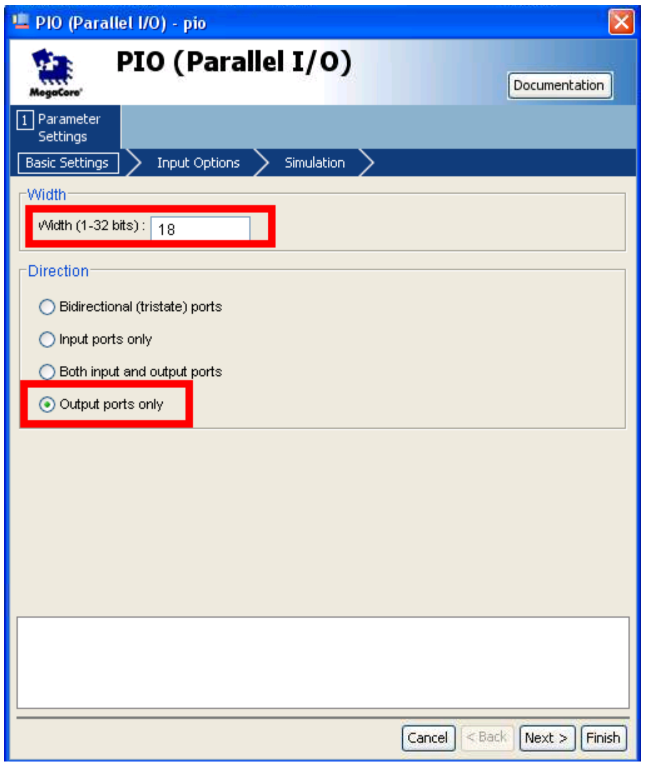
- [ ] Synchronously capture
- ○ Rising edge
- ○ Falling edge
- ○ Either edge
- [ ] Enable bit-clearing for edge capture register

Interrupt

- [ ] Generate IRQ
- ○ Level
  (Interrupt CPU when any unmasked I/O pin is logic true)
- ○ Edge
  (Interrupt CPU when any unmasked bit in the edge-capture register is logic true)

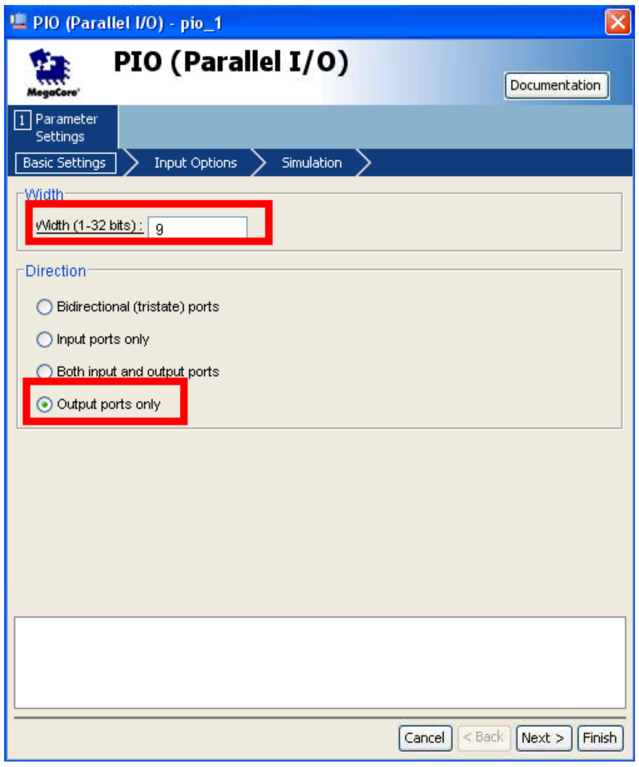
Warning: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO input

Figure 1-26: Configurations of 18 switches on DE2


PIO (Parallel I/O) - pio

Width (1-32 bits) : 18

Direction: ◯ Bidirectional (tristate) ports ◯ Input ports only ◯ Both input and output ports [✓ Output ports only]


PIO (Parallel I/O) - pio_1

Width (1-32 bits) : 9

Direction: ◯ Bidirectional (tristate) ports ◯ Input ports only ◯ Both input and output ports [✓ Output ports only]

Figure 1-27: Settings of the LEDs

### 1.2.13 Add the SD card support

The SD card interface is accessed with 3 groups of parallel I/Os, the clock, the command, and the data. These clock of SD card interface is a single bit input (renamed to **SD_CLK**), whose setting is illustrated in Figure 1-28. The command sent to the SD card is through a bi-directional tristate bit (renamed to **SD_CMD**) with settings detailed in Figure 1-29. The last group of signals to the SD card is the data signal (renamed to **SD_DAT**), which is shown in Figure 1-30.

### 1.2.14 Add the Character LCD display

To work with the 2 lines character LCD display on DE2, add "**Altera SOPC Builder->Peripherals->Display->Character LCD**" where a 16x2, Optrex 16207 will be added to the system.

### 1.2.15 Add the seven segments displays

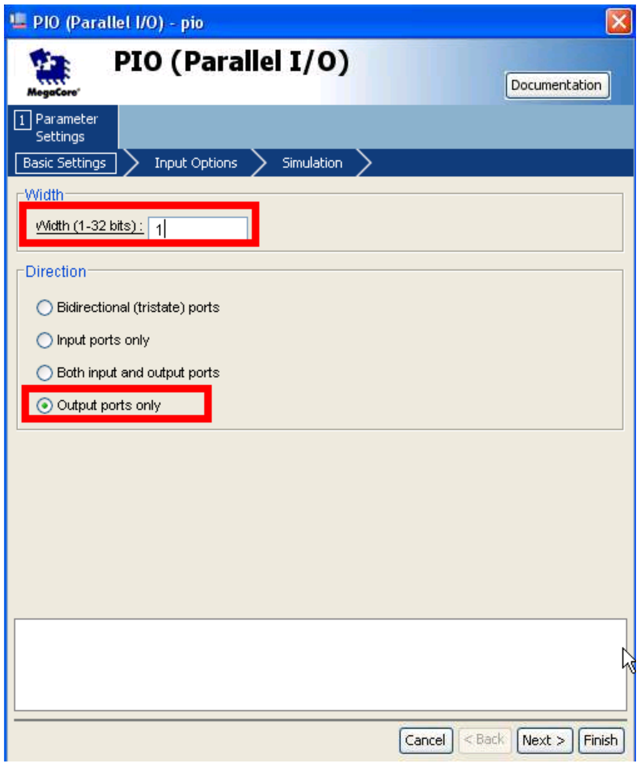
8 seven segments displays can be added by choosing "**Altera SOPC Builder->Terasic Technologies Inc->SEG7_LUT_8**" as shown in Figure 1-31.

### 1.2.16 Add the Ethernet support

Ethernet support can be included by double clicking "**Altera SOPC Builder->Terasic Technologies Inc->DM9000A**" as shown in Figure 1-32.

### 1.2.17 Add the USB device support

USB port can be enabled by selecting "**Altera SOPC Builder->Terasic Technologies Inc->ISP1362**" as shown in Figure 1-33.


PIO (Parallel I/O) - pio

PIO (Parallel I/O)

Parameter Settings: Basic Settings > Input Options > Simulation

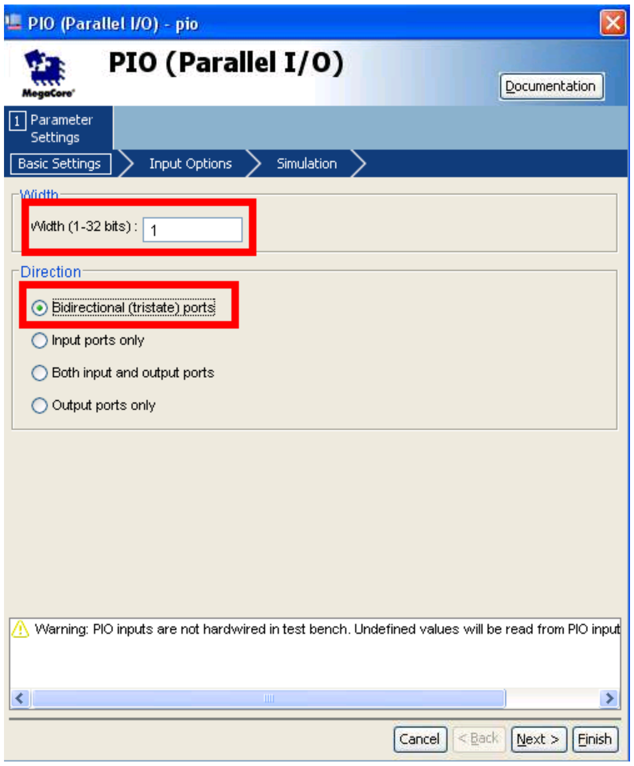
Width

Width (1-32 bits) : 1

Direction

- ( ) Bidirectional (tristate) ports
- ( ) Input ports only
- ( ) Both input and output ports
- (●) Output ports only

Figure 1-28: The clock of the SD card interfaces


PIO (Parallel I/O) - pio

PIO (Parallel I/O)

Parameter Settings: Basic Settings > Input Options > Simulation

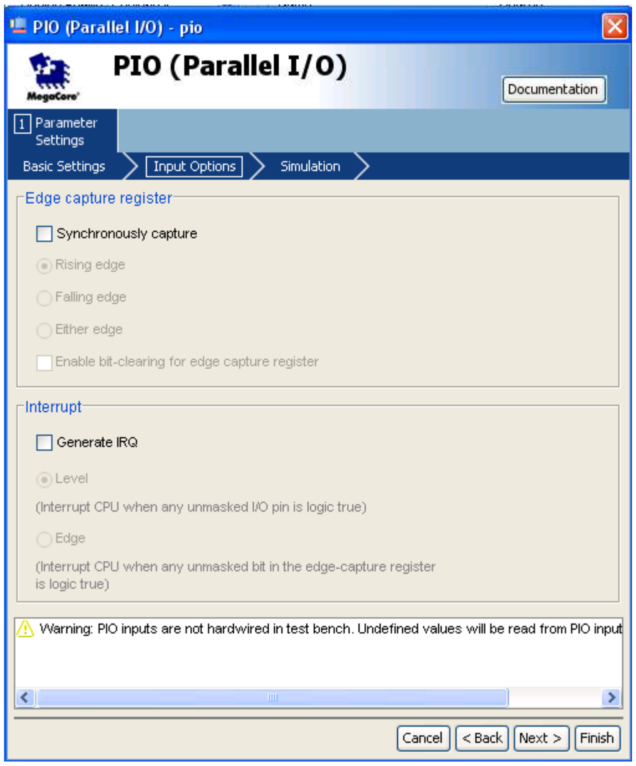
Width

Width (1-32 bits) : 1

Direction

- (●) Bidirectional (tristate) ports
- ( ) Input ports only
- ( ) Both input and output ports
- ( ) Output ports only

Warning: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO input


PIO (Parallel I/O) - pio

PIO (Parallel I/O)

Parameter Settings: Basic Settings > Input Options > Simulation

Edge capture register

- [ ] Synchronously capture
- (●) Rising edge
- ( ) Falling edge
- ( ) Either edge
- [ ] Enable bit-clearing for edge capture register

Interrupt

- [ ] Generate IRQ
- (●) Level

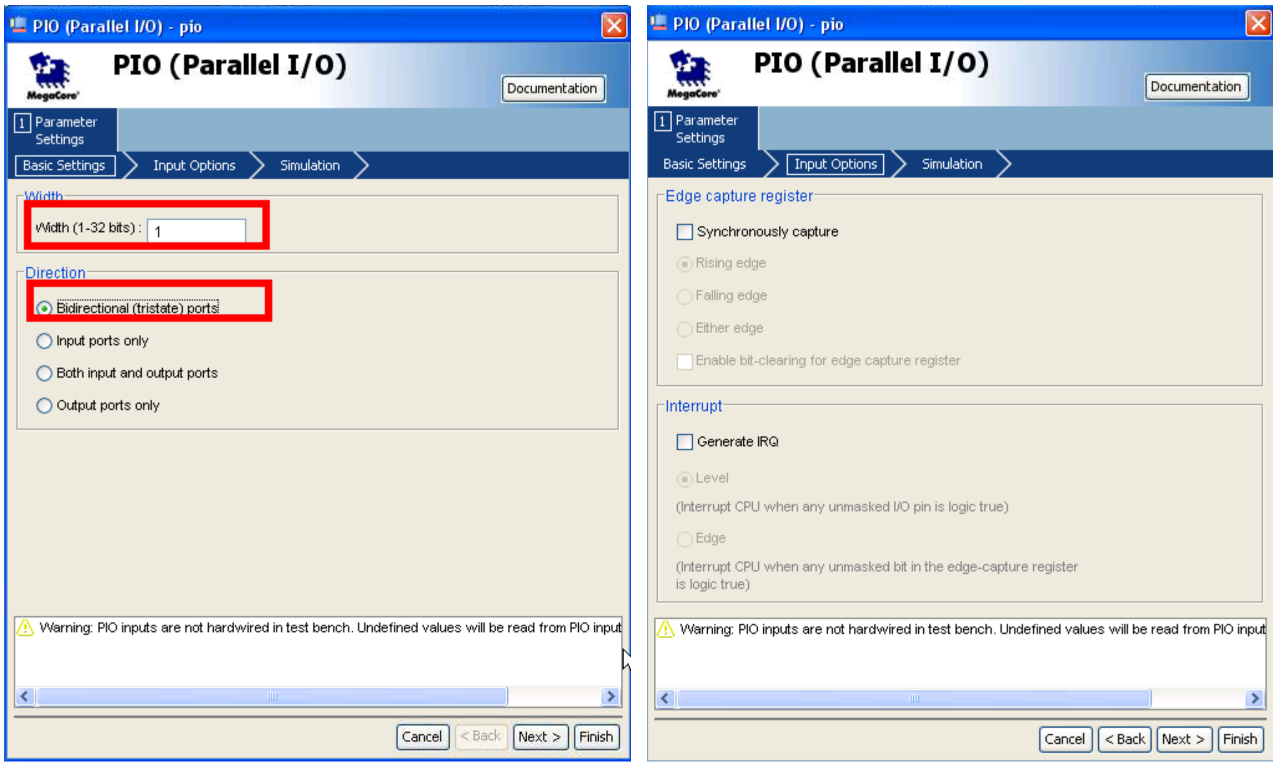
(Interrupt CPU when any unmasked I/O pin is logic true)

- ( ) Edge

(Interrupt CPU when any unmasked bit in the edge-capture register is logic true)

Warning: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO input

Figure 1-29: Settings of the SD card command I/O



Width
Width (1-32 bits) : 1

Direction
[✓ Bidirectional (tristate) ports]
Input ports only
Both input and output ports
Output ports only

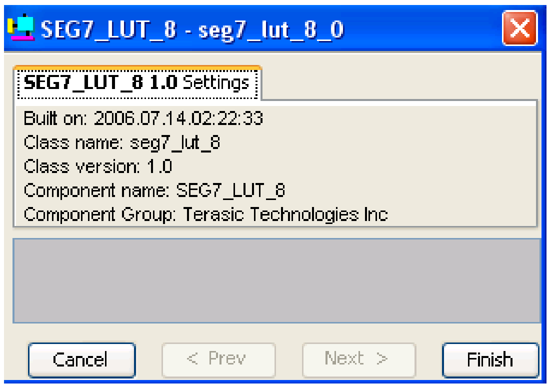
Warning: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO input

Edge capture register
Synchronously capture
Rising edge
Falling edge
Either edge
Enable bit-clearing for edge capture register

Interrupt
Generate IRQ
Level
(Interrupt CPU when any unmasked I/O pin is logic true)
Edge
(Interrupt CPU when any unmasked bit in the edge-capture register is logic true)

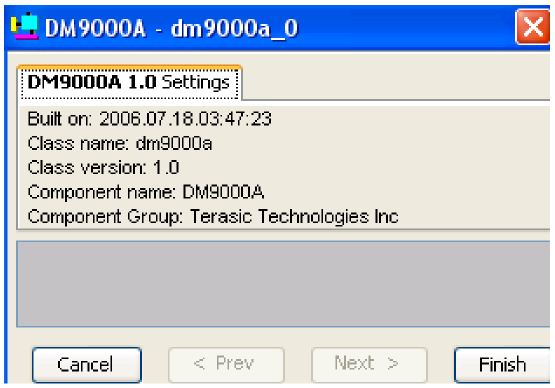
Warning: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO input

Figure 1-30: Settings of the SD card data I/O



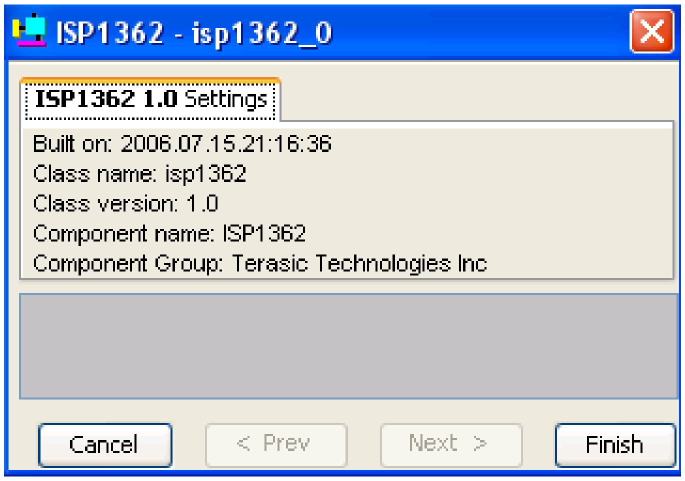
Built on: 2006.07.14.02:22:33
Class name: seg7_lut_8
Class version: 1.0
Component name: SEG7_LUT_8
Component Group: Terasic Technologies Inc

Figure 1-31: The seven segments display



Built on: 2006.07.18.03:47:23
Class name: dm9000a
Class version: 1.0
Component name: DM9000A
Component Group: Terasic Technologies Inc

Figure 1-32: DM9000A the Ethernet controller



Built on: 2006.07.15.21:16:36
Class name: isp1362
Class version: 1.0
Component name: ISP1362
Component Group: Terasic Technologies Inc

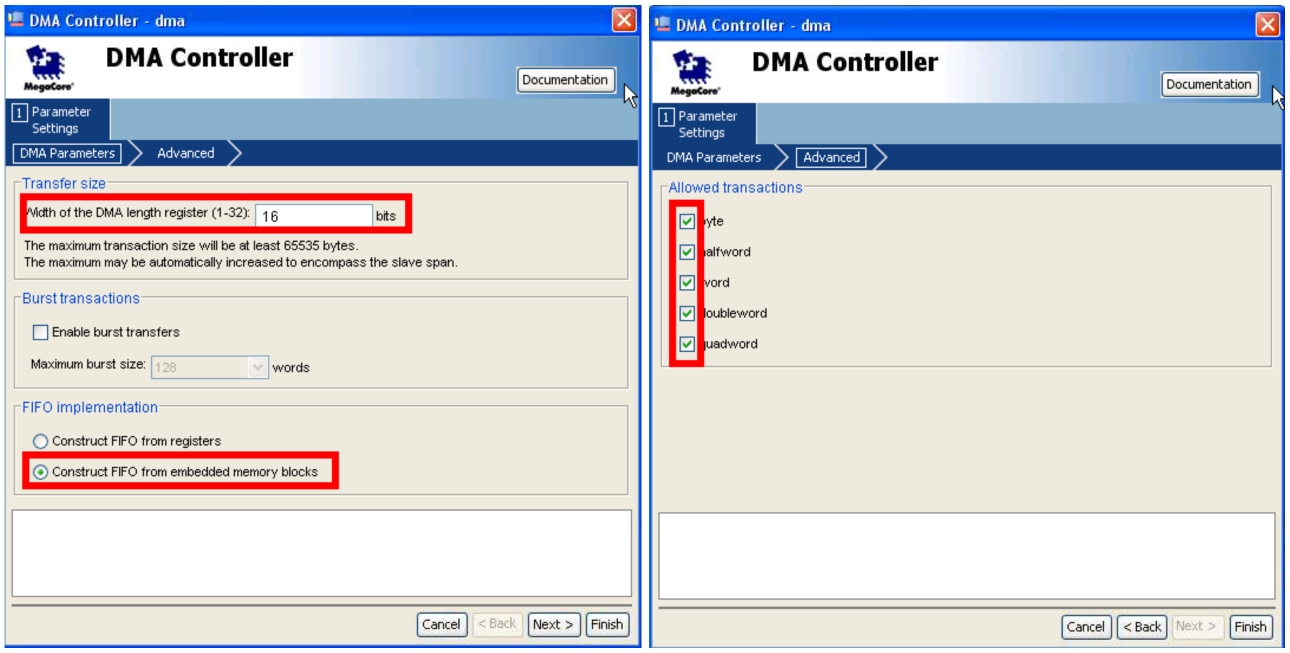
Figure 1-33: ISP1362 USB controller

### 1.2.18 Add the DMA controller to enable DMA operations

The DMA controller enables data to be transferred between device to device, memory to device, or device to memory directly without being preformed through the processor (where data is read into register, and write to the destination from register) to achieve better performance and parallelism between operations. Add the "**DMA Controller**" from "**Altera SOPC Builder->Memories and Memory Controllers->DMA**". From previously added components, say on-chip RAM and SDRAM, they both have 16 bits data width, and for UART, the register is 8 bits wide, therefore a **16 bits of DMA transferring width is sufficient** for our needs. Refer to Figure 1-34 for detailed configuration of the DMA controller. To establish the path (between source/reader and destination/writer) of DMA operations, interconnections must be made. For example, we might want to have direct transfers between on-chip RAM and the SDRAM, we can assign the interconnections in these steps:

1. Make sure **connections** are visible in the added component list of SOPC builder, if it doesn't, change it from "**View->Show Connections Column**" menu.
2. Move the mouse over the connections column, as Figure 1-35, the lines means the available paths of data transfers, a **solid dot** means the transfer path is made, a **hallow dot** means the path is not used.
3. Move the mouse over one of the dot, say the most right dot next to s1 of the on-chip me memory, as shown in Figure 1-35, says "**dma/write_master on-chip_mem/s1**" which means that by enabling this dot, the DMA controller can transfer data (as **writing**) to the on-chip memory through the memory port s1. Same rule applies to the dot at the left, which is used for reading out of the memory.
4. As the requirement here in this example, make 4 dots solid (2 for on-chip memory, for reading and writing, 2 for the SDRAM for reading and writing as well) by clicking the dots, as shown in Figure 1-36.

Next, we need to enable the path for reading out from the UART (RS-232) port, by click on the "**dma/read_master uart/s1**" to make the dot solid.



Transfer size

Width of the DMA length register (1-32): 16 bits

The maximum transaction size will be at least 65535 bytes.
The maximum may be automatically increased to encompass the slave span.

Burst transactions

☐ Enable burst transfers

Maximum burst size: 128 words

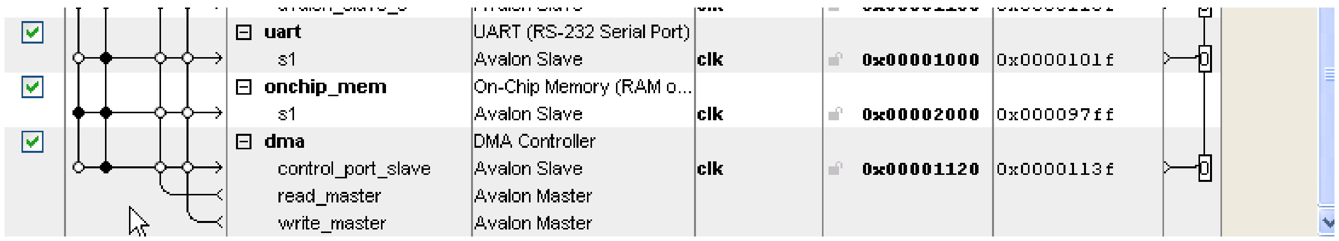
FIFO implementation

( ) Construct FIFO from registers
(●) Construct FIFO from embedded memory blocks

Allowed transactions

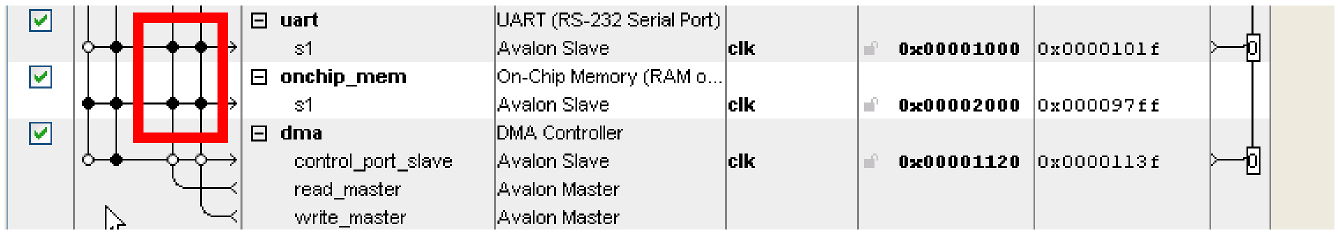
☑ yte
☑ alfword
☑ vord
☑ loubleword
☑ uadword

Figure 1-34: Attributes of the flash memory



| | | | | | |
|---|---|---|---|---|---|
| ☑ | ⊟ uart | UART (RS-232 Serial Port) | | | |
| | s1 | Avalon Slave | clk | 0x00001000 | 0x0000101f |
| ☑ | ⊟ onchip_mem | On-Chip Memory (RAM o... | | | |
| | s1 | Avalon Slave | clk | 0x00002000 | 0x000097ff |
| ☑ | ⊟ dma | DMA Controller | | | |
| | control_port_slave | Avalon Slave | clk | 0x00001120 | 0x0000113f |
| | read_master | Avalon Master | | | |
| | write_master | Avalon Master | | | |

Figure 1-35: Connections column and transfer path

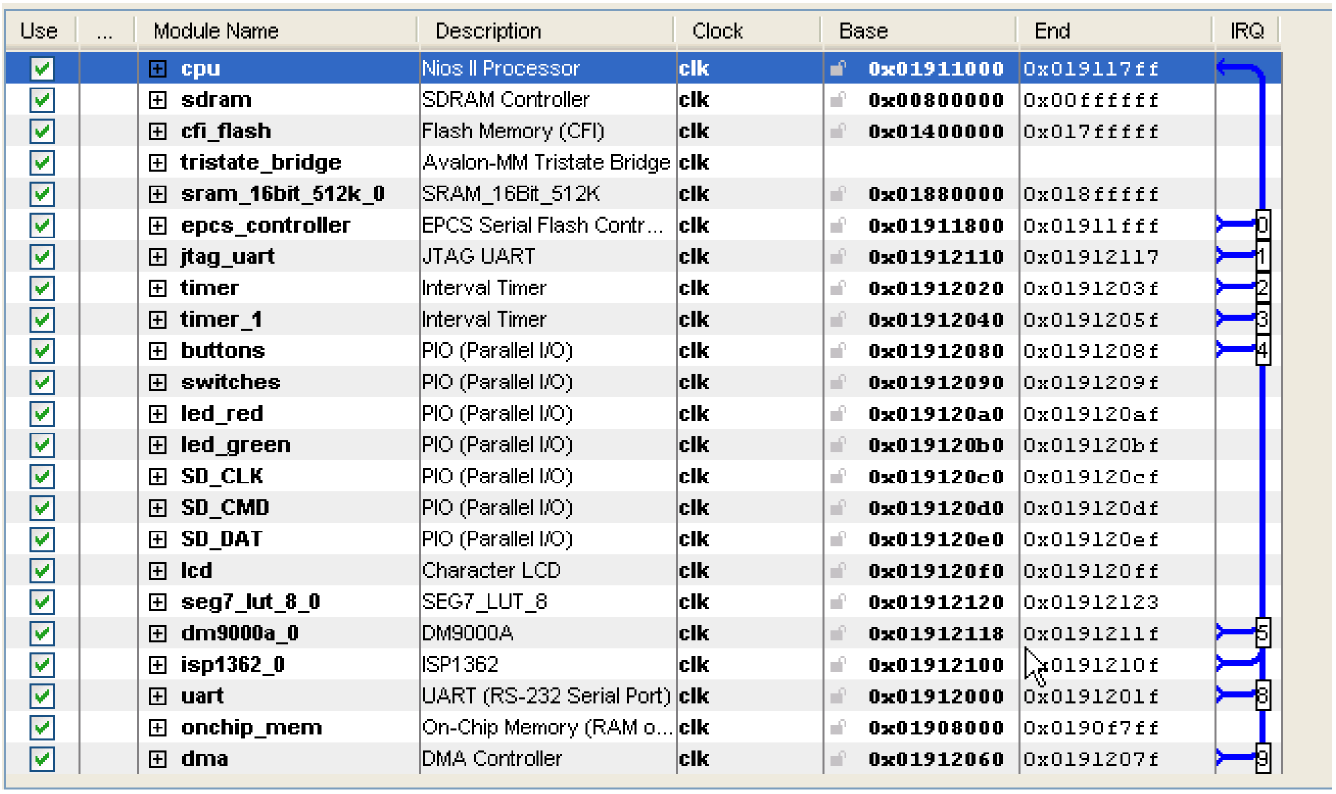


| | | | | | |
|---|---|---|---|---|---|
| ☑ | ⊟ uart | UART (RS-232 Serial Port) | | | |
| | s1 | Avalon Slave | clk | 0x00001000 | 0x0000101f |
| ☑ | ⊟ onchip_mem | On-Chip Memory (RAM o... | | | |
| | s1 | Avalon Slave | clk | 0x00002000 | 0x000097ff |
| ☑ | ⊟ dma | DMA Controller | | | |
| | control_port_slave | Avalon Slave | clk | 0x00001120 | 0x0000113f |
| | read_master | Avalon Master | | | |
| | write_master | Avalon Master | | | |

Figure 1-36: DMA transfers are enabled between memories and UART

## 1.2.19 Resolve conflicts of base addresses

After adding up the required components, we have to ensure the addresses and IRQ number of each components do not conflict each other. It is achieved through menu "**System->Auto-Assign Base Addresses**", you should find out that a lot of error messages are suppressed. Now we can sort out the IRQ conflicts in the similar fashion through "**System->Auto-Assign IRQs**". You can assigned IRQ manually depending on the priorities/importance of reactivity of a device. In this case, set the dma IRQ to 9 (if it hasn't been 9). Now all the conflicts should be resolved as shown in Figure 1-37 (from menu View->Collapse All, the address might not be the same).



| Use | ... | Module Name | Description | Clock | Base | End | IRQ |
|---|---|---|---|---|---|---|---|
| ✓ | | ⊞ cpu | Nios II Processor | clk | 0x01911000 | 0x019117ff | |
| ✓ | | ⊞ sdram | SDRAM Controller | clk | 0x00800000 | 0x00ffffff | |
| ✓ | | ⊞ cfi_flash | Flash Memory (CFI) | clk | 0x01400000 | 0x017fffff | |
| ✓ | | ⊞ tristate_bridge | Avalon-MM Tristate Bridge | clk | | | |
| ✓ | | ⊞ sram_16bit_512k_0 | SRAM_16Bit_512K | clk | 0x01880000 | 0x018fffff | |
| ✓ | | ⊞ epcs_controller | EPCS Serial Flash Contr... | clk | 0x01911800 | 0x01911fff | 0 |
| ✓ | | ⊞ jtag_uart | JTAG UART | clk | 0x01912110 | 0x01912117 | 1 |
| ✓ | | ⊞ timer | Interval Timer | clk | 0x01912020 | 0x0191203f | 2 |
| ✓ | | ⊞ timer_1 | Interval Timer | clk | 0x01912040 | 0x0191205f | 3 |
| ✓ | | ⊞ buttons | PIO (Parallel I/O) | clk | 0x01912080 | 0x0191208f | 4 |
| ✓ | | ⊞ switches | PIO (Parallel I/O) | clk | 0x01912090 | 0x0191209f | |
| ✓ | | ⊞ led_red | PIO (Parallel I/O) | clk | 0x019120a0 | 0x019120af | |
| ✓ | | ⊞ led_green | PIO (Parallel I/O) | clk | 0x019120b0 | 0x019120bf | |
| ✓ | | ⊞ SD_CLK | PIO (Parallel I/O) | clk | 0x019120c0 | 0x019120cf | |
| ✓ | | ⊞ SD_CMD | PIO (Parallel I/O) | clk | 0x019120d0 | 0x019120df | |
| ✓ | | ⊞ SD_DAT | PIO (Parallel I/O) | clk | 0x019120e0 | 0x019120ef | |
| ✓ | | ⊞ lcd | Character LCD | clk | 0x019120f0 | 0x019120ff | |
| ✓ | | ⊞ seg7_lut_8_0 | SEG7_LUT_8 | clk | 0x01912120 | 0x01912123 | |
| ✓ | | ⊞ dm9000a_0 | DM9000A | clk | 0x01912118 | 0x0191211f | 5 |
| ✓ | | ⊞ isp1362_0 | ISP1362 | clk | 0x01912100 | 0x0191210f | |
| ✓ | | ⊞ uart | UART (RS-232 Serial Port) | clk | 0x01912000 | 0x0191201f | 8 |
| ✓ | | ⊞ onchip_mem | On-Chip Memory (RAM o... | clk | 0x01908000 | 0x0190f7ff | |
| ✓ | | ⊞ dma | DMA Controller | clk | 0x01912060 | 0x0191207f | 9 |

Figure 1-37: The components and their IRQ and addresses in current nios2 system

### 1.2.20 Checking the module names

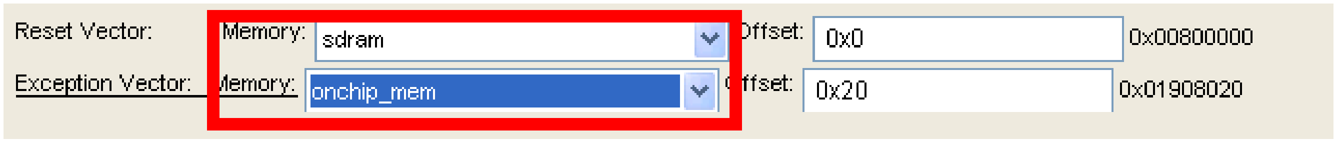
Even though module names for everyone are not necessary to be the same, for saving some troubles and simplifying this lab, module names are asked to be fixed (presenting in the names – descriptions) as follows (order is not important, but names have to be the same):

1. cpu – Nios II Processor
2. sdram – SDRAM controllers
3. cfi_flash – Flash Memory CFI
4. tristate_bridge – Avalon-MM Tristate Bridge
5. sram_16bit_512k_0 – SRAM_16Bit_512K
6. epcs_controller – EPCS Serial Flash Controller
7. jtag_uart – JTAG UART
8. timer – Interval Timer
9. timer_1 – Interval Timer
10. buttons – PIO
11. switches – PIO
12. led_red – PIO
13. led_green – PIO
14. SD_CLK – PIO
15. SD_CMD – PIO
16. SD_DAT – PIO
17. lcd – Character LCD
18. seg7_lut_8_0 – SEG7_LUT_8
19. dm9000a_0 – DM9000A

20. isp1362_0 – ISP1362
21. uart – UART (RS-232 Serial Port)
22. onchip_mem – On-Chip Memory (RAM or ROM)
23. dma – DMA Controller

### 1.2.21 More about Nios II processor settings and generation of the system

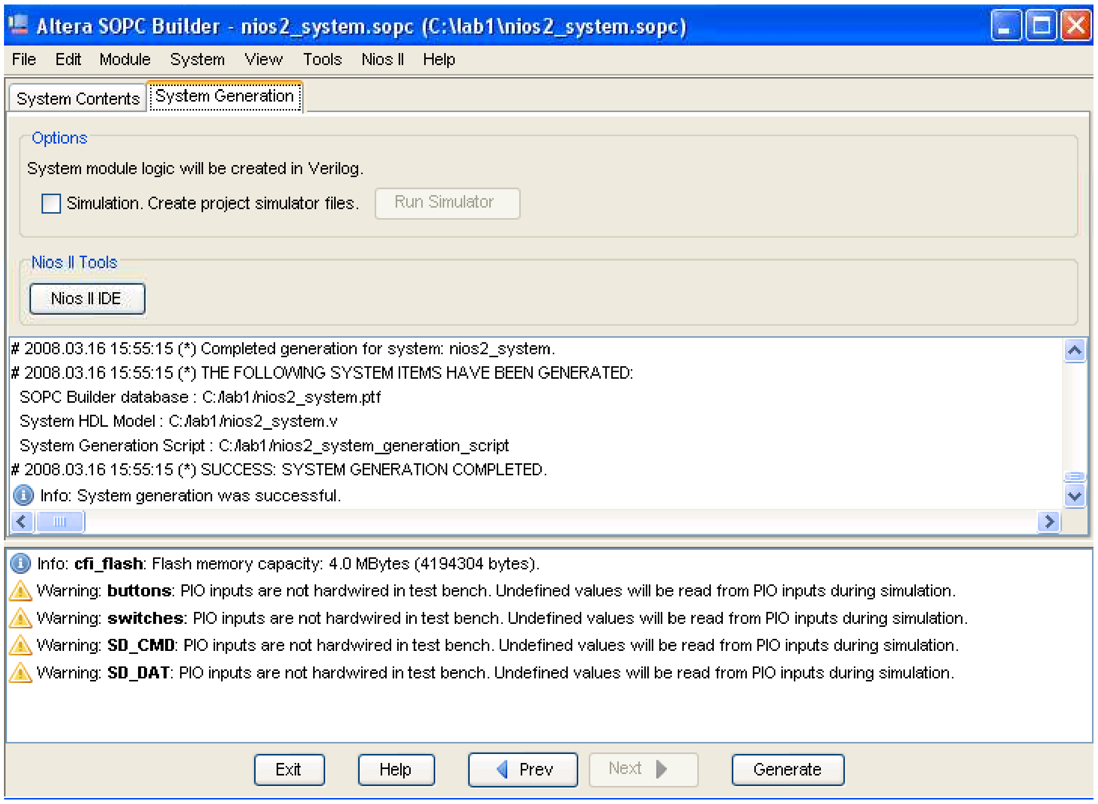
Since currently two memories (on-chip RAM and SDRAM) are available, we will need to decide the usages of them. For instance, the place to store RESET address, or interrupt vector table. Double click on "cpu" from the added-component-list at the right hand side. It will show the details of the NIOS II processor as in Figure 1-14. Reset address is used when reset happens, the program will start from the reset address once the **program is downloaded** or **the processor is rest**. We set the **Reset vector at offset 0x0 of the sdram**, which is going to store the program as well, and set **Exception vector to 0x20 of the onchip_mem**, as illustrated in Figure 1-38.



Reset Vector: Memory: sdram Offset: 0x0 0x00800000
Exception Vector: Memory: onchip_mem Offset: 0x20 0x01908020

Figure 1-38: The setting of the vectors

Move the the "System Generation" tab, un-tick the "Simulation Create simulator project files", and click "Generate" button to start generating the configured system. Click "Save" if it says the system has been modified. Click "Exit" after the system is generated successfully as shown in Figure 1-39.



Altera SOPC Builder - nios2_system.sopc (C:\lab1\nios2_system.sopc)
File Edit Module System View Tools Nios II Help
System Contents
System Generation
Options
Run Simulator
Nios II Tools
Nios II IDE
Exit
Help
Prev
Next
Generate

System module logic will be created in Verilog.

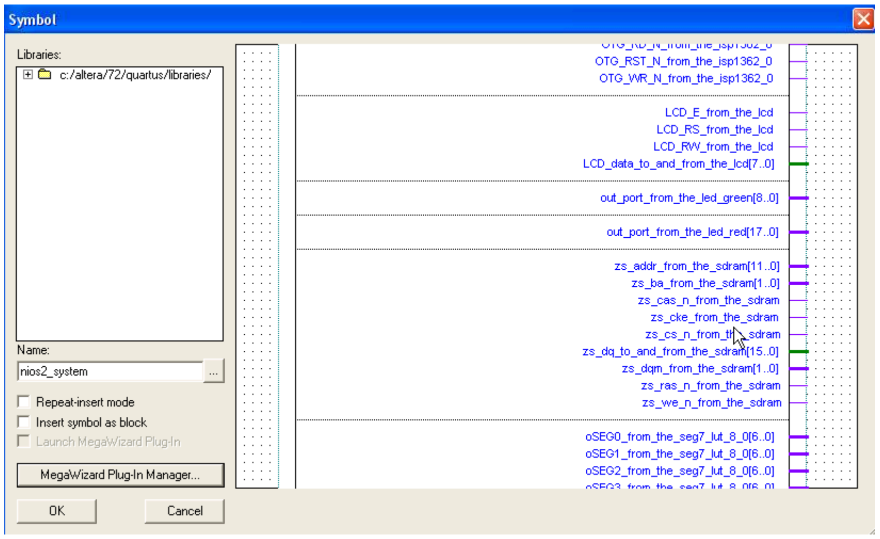
☐ Simulation. Create project simulator files.

```
# 2008.03.16 15:55:15 (*) Completed generation for system: nios2_system.
# 2008.03.16 15:55:15 (*) THE FOLLOWING SYSTEM ITEMS HAVE BEEN GENERATED:
  SOPC Builder database : C:/lab1/nios2_system.ptf
  System HDL Model : C:/lab1/nios2_system.v
  System Generation Script : C:/lab1/nios2_system_generation_script
# 2008.03.16 15:55:15 (*) SUCCESS: SYSTEM GENERATION COMPLETED.
Info: System generation was successful.
```

```
Info: cfi_flash: Flash memory capacity: 4.0 MBytes (4194304 bytes).
Warning: buttons: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
Warning: switches: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
Warning: SD_CMD: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
Warning: SD_DAT: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
```

Figure 1-39: Generating the Nios II system

After that, the Nios II system will appear on the available symbol window as shown in Figure 1-40 as mentioned previously. Click “OK” and click on the schematic to add it.


Symbol
Libraries:
c:/altera/72/quartus/libraries/
OTG_RST_N_from_the_isp1362_0
OTG_WR_N_from_the_isp1362_0
LCD_E_from_the_lcd
LCD_RS_from_the_lcd
LCD_RW_from_the_lcd
LCD_data_to_and_from_the_lcd[7..0]
out_port_from_the_led_green[8..0]
out_port_from_the_led_red[17..0]
zs_addr_from_the_sdram[11..0]
zs_ba_from_the_sdram[1..0]
zs_cas_n_from_the_sdram
zs_cke_from_the_sdram
zs_cs_n_from_the_sdram
zs_dq_to_and_from_the_sdram[15..0]
zs_dqm_from_the_sdram[1..0]
zs_ras_n_from_the_sdram
zs_we_n_from_the_sdram
oSEG0_from_the_seg7_lut_8_0[6..0]
oSEG1_from_the_seg7_lut_8_0[6..0]
oSEG2_from_the_seg7_lut_8_0[6..0]
Repeat-insert mode
Insert symbol as block
Launch MegaWizard Plug-In
MegaWizard Plug-In Manager...
OK
Cancel

Name: nios2_system

Figure 1-40: Nios II system as an available component

From the schematic, you can tell the characteristics of any particular input output port. The blue lines indicate input ports, while **thicker** ones are **buses** and thinner ones are bit lines. The purple lines on the right are output ports, where thicker and thinner lines mean the same thing. The green lines indicate bi-directional ports, which is used for both inputs and outputs.

## 1.3 Integrate Nios II system with the other hardware components on DE2

In order to make the SDRAM chip works correctly, according to [1], an PLL (phase-locked loop) circuit has to be added to delay the SDRAM clock by 3 ns to the NIOS2_system's clock. It is carried out as follows:

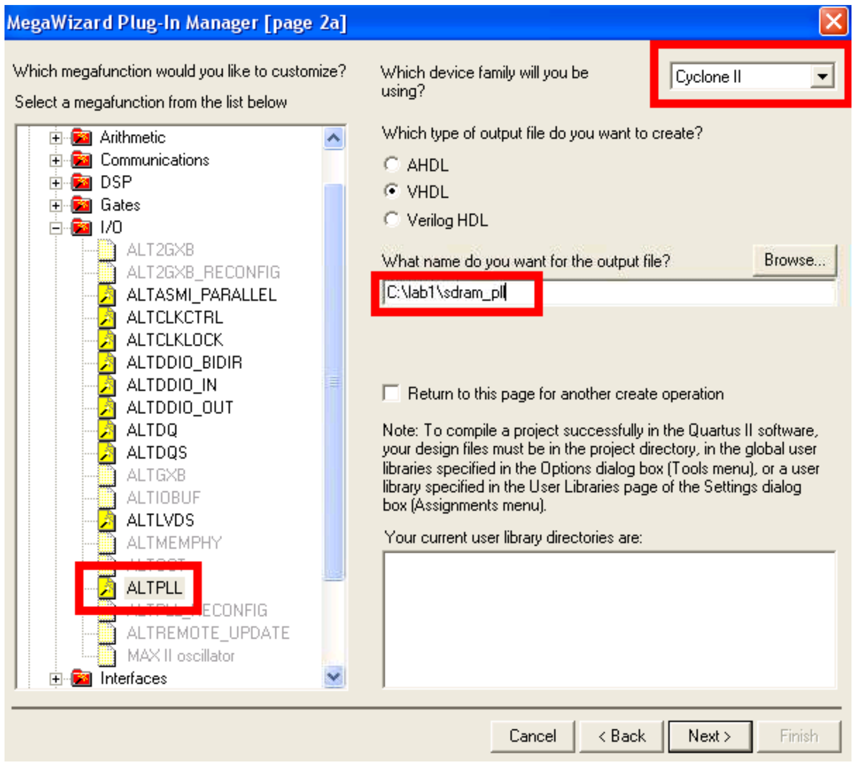
1. Double click on any space of the schematic design
2. Click “**MegaWizard Plug-In Manager**” as shown in 1-10
3. Choose “Create a new custom megafunction variation” and click “Next”
4. Select “Installed Plug-Ins->I/O->ALTPLL” and name it as “sdram_pll” as shown in Figure 1-41, and click “Next”
5. Follow the configuration ins Figure 1-42, set the speed grade to “**Any**”, inclock0 input to “**50MHz**” described previously, operation mode to “**In Normal Mode**”, and compensated clock to “**c0**” and click “next”
6. Un-tick all boxes as in Figure 1-43, and click “next”, followed by another “next”.
7. We are going to use c0 as the output clock to the SDRAM chip's input clock, follow the configuration in Figure 1-44, tick the “Enter output clock frequency” and set it to “50,0 Mhz”, set the “Clock phase shift” to “-3ns” and click “finish”
8. Paste it above the nios2_system as shown in Figure 1-46.

Some of the peripherals require constantly high signal values to work properly, therefore we can create a VHDL entity which outputting `1` all the time. This can be done in the following steps:

1. Create a VHDL entity by "**File->New->VHDL File**"
2. Write the following code for this entity, namely **always_one**:

```
entity always_one is
        port ( output : out bit );
end entity always_one;
architecture behavior of always_one is
begin
        output <= '1';
end architecture behavior;
```

3. Then save it with "**File->Save as**", with the name "**always_one.vhd**", remember to tick "**Add file to current project**" in the "Save as " window.
4. Create a symbol file by "**File->Create/Update->Create Symbol Files for Current File**"
5. Double click on the space of the schematic design (lab1.bdf)
6. Choose **always_one.bdf** as shown in Figure 1-45. Place it in the schematic design detailed in Figure 1-46.


MegaWizard Plug-In Manager [page 2a]

Which megafunction would you like to customize?
Select a megafunction from the list below

Which device family will you be using? Cyclone II

Which type of output file do you want to create?
( ) AHDL
(•) VHDL
( ) Verilog HDL

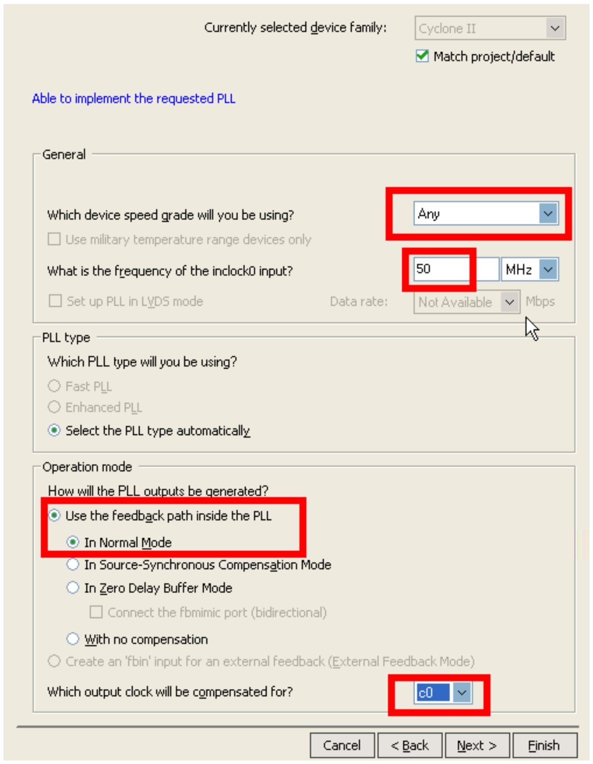
What name do you want for the output file? C:\lab1\sdram_pll

[ ] Return to this page for another create operation

Note: To compile a project successfully in the Quartus II software, your design files must be in the project directory, in the global user libraries specified in the Options dialog box (Tools menu), or a user library specified in the User Libraries page of the Settings dialog box (Assignments menu).

Your current user library directories are:

Figure 1-41: Creating a PLL


Currently selected device family: Cyclone II

☑ Match project/default

Able to implement the requested PLL

General

Which device speed grade will you be using? Any

☐ Use military temperature range devices only

What is the frequency of the inclock0 input? 50 MHz

☐ Set up PLL in LVDS mode Data rate: Not Available Mbps

PLL type

Which PLL type will you be using?

○ Fast PLL

○ Enhanced PLL

● Select the PLL type automatically

Operation mode

How will the PLL outputs be generated?

● Use the feedback path inside the PLL

● In Normal Mode

○ In Source-Synchronous Compensation Mode

○ In Zero Delay Buffer Mode

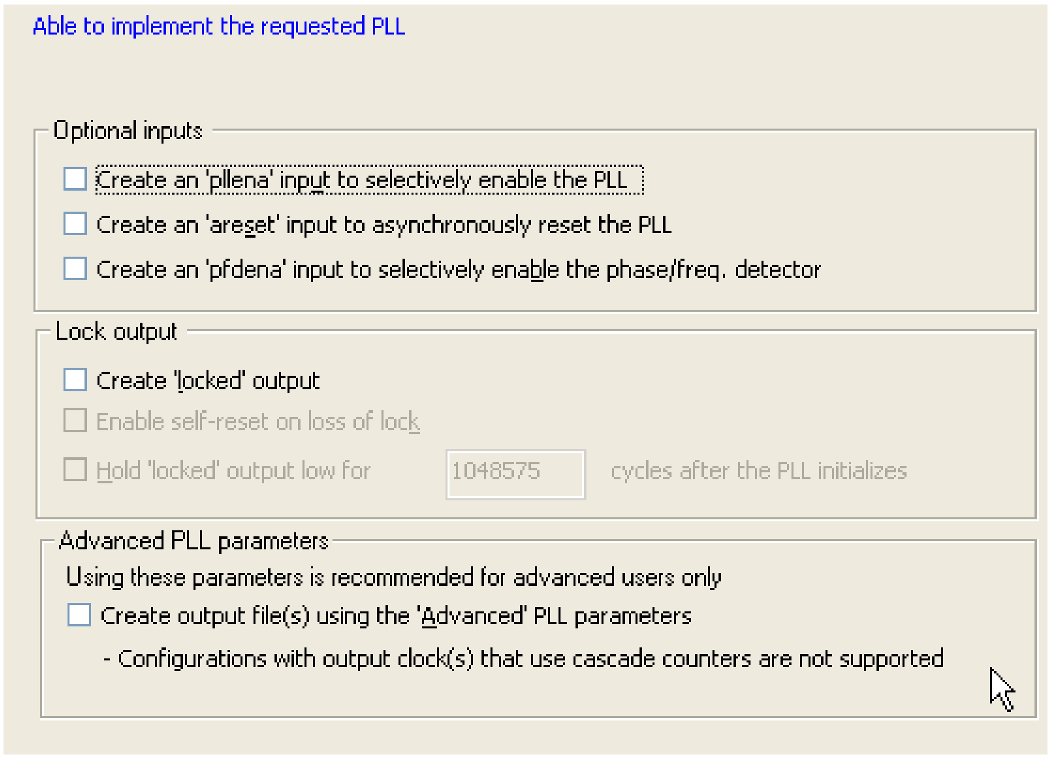
☐ Connect the fbmimic port (bidirectional)

○ With no compensation

○ Create an 'fbin' input for an external feedback (External Feedback Mode)

Which output clock will be compensated for? c0

Figure 1-42: Configuring the sdram_pll


Able to implement the requested PLL

Optional inputs

☐ Create an 'pllena' input to selectively enable the PLL

☐ Create an 'areset' input to asynchronously reset the PLL

☐ Create an 'pfdena' input to selectively enable the phase/freq. detector

Lock output

☐ Create 'locked' output

☐ Enable self-reset on loss of lock

☐ Hold 'locked' output low for 1048575 cycles after the PLL initializes

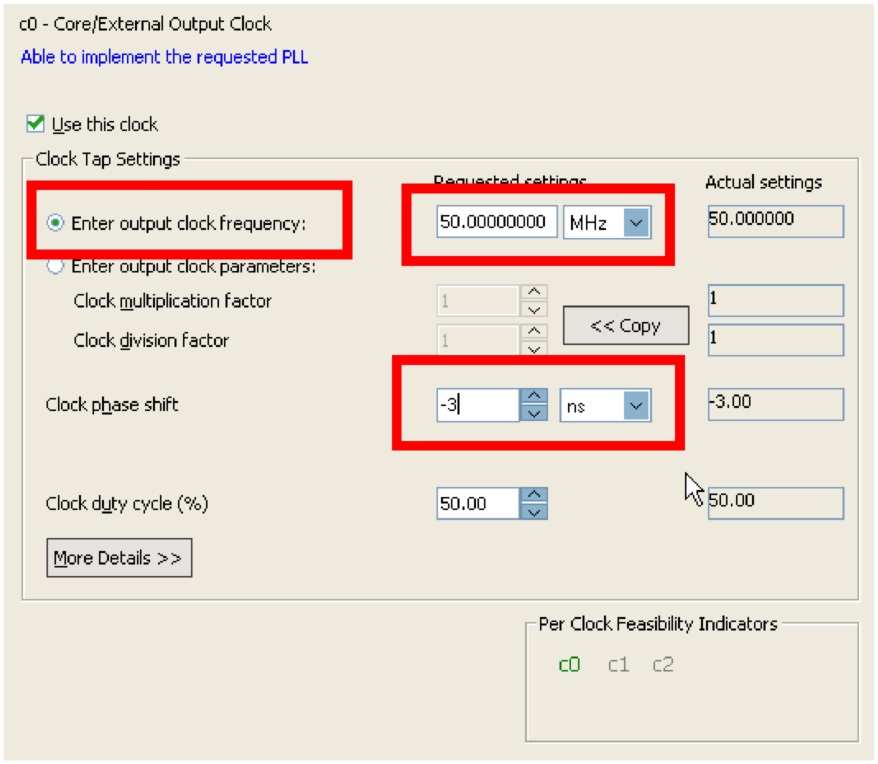
Advanced PLL parameters

Using these parameters is recommended for advanced users only

☐ Create output file(s) using the 'Advanced' PLL parameters

- Configurations with output clock(s) that use cascade counters are not supported

Figure 1-43: Configuring the sdram_pll continued


c0 - Core/External Output Clock

Able to implement the requested PLL

☑ Use this clock

Clock Tap Settings

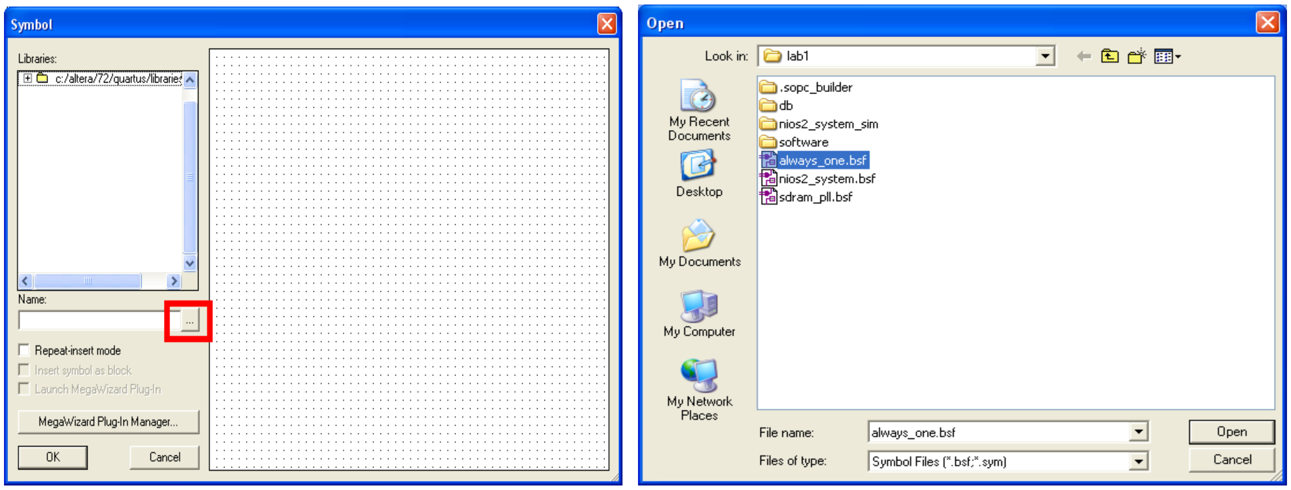
| | Requested settings | | Actual settings |
|---|---|---|---|
| ● Enter output clock frequency: | 50.00000000 | MHz | 50.000000 |
| ○ Enter output clock parameters: | | | |
| Clock multiplication factor | 1 | << Copy | 1 |
| Clock division factor | 1 | | 1 |
| Clock phase shift | -3 | ns | -3.00 |
| Clock duty cycle (%) | 50.00 | | 50.00 |

More Details >>

Per Clock Feasibility Indicators

c0 c1 c2

Figure 1-44: Configuring the sdram_pll continued


Symbol

Libraries:

c:/altera/72/quartus/libraries

Name:

☐ Repeat-insert mode
☐ Insert symbol as block
☐ Launch MegaWizard Plug-In

MegaWizard Plug-In Manager...

OK Cancel

Open

Look in: lab1

My Recent Documents
Desktop
My Documents
My Computer
My Network Places

.sopc_builder
db
nios2_system_sim
software
always_one.bsf
nios2_system.bsf
sdram_pll.bsf

File name: always_one.bsf

Files of type: Symbol Files (*.bsf;*.sym)

Open Cancel

Figure 1-45: Adding “always_one” to the schematic design

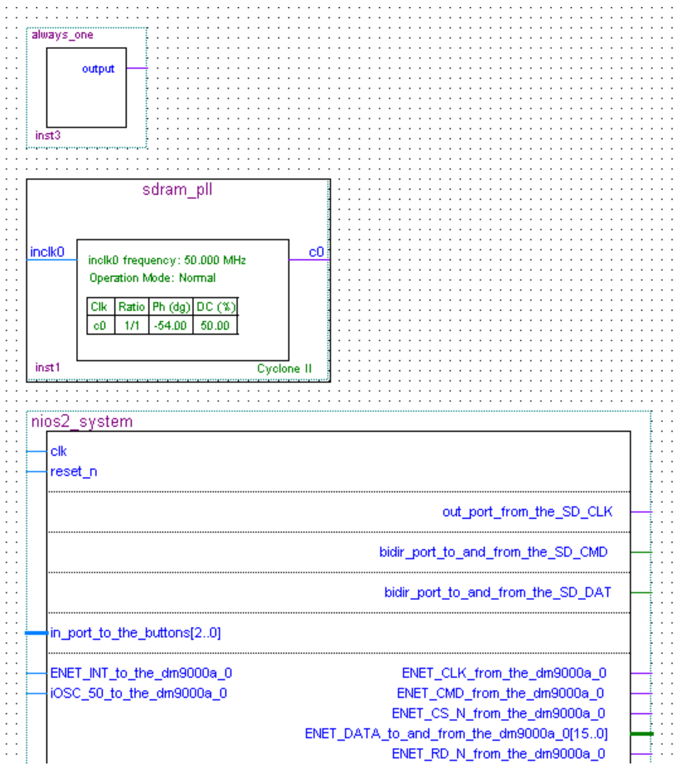

always_one
output
inst3
sdram_pll
inclk0
c0
inclk0 frequency: 50.000 MHz
Operation Mode: Normal
Clk | Ratio | Ph (dg) | DC (%)
c0 | 1/1 | -54.00 | 50.00
inst1
Cyclone II
nios2_system
clk
reset_n
out_port_from_the_SD_CLK
bidir_port_to_and_from_the_SD_CMD
bidir_port_to_and_from_the_SD_DAT
in_port_to_the_buttons[2..0]
ENET_INT_to_the_dm9000a_0
iOSC_50_to_the_dm9000a_0
ENET_CLK_from_the_dm9000a_0
ENET_CMD_from_the_dm9000a_0
ENET_CS_N_from_the_dm9000a_0
ENET_DATA_to_and_from_the_dm9000a_0[15..0]
ENET_RD_N_from_the_dm9000a_0

Figure 1-46: Overview of the schematic design

Now we have to create inputs and outputs on the top level design and interconnections between the sdram_pll and nios2_system. **Double click any space on schematic again**, you can see 3 kinds of pins – **bidir** (bi-directional), **input**, and **output** under the **primitives**, as shown in Figure 1-47. Add 1 bidir, 5 input, 12 outputs ports and rename them by right clicking on the signal and choosing properties and filling up the pin name followed by clicking "OK". The names (**Note that the upper case and lower case are different, brackets are meaningful, two dots are between numbers in the bracket**) of the ports are listed (**PIN/BUS primitive names** – port on nios2_system to connect).

Create the following bidirectional ports on the right:

1. **SD_CMD** - bidir_port_to_and_from_the_SD_CMD
2. **SD_DAT** - bidir_port_to_and_from_the_SD_DAT
3. **ENET_DATA[15..0]** – ENET_DATA_to_and_from_the_dm9000a_0[15..0]
4. **OTG_DATA[15..0]** – OTG_DATA_to_and_from_the_isp1362_0[15..0]
5. **LCD_DATA[7..0] –** LCD_data_to_and_from_the_lcd[7..0]
6. **DRAM_DQ[15..0]** – zs_dq_to_and_from_the_sdram[15..0]
7. **SRAM_DQ [15..0]** – SRAM_DQ_to_and_from_the_sram_16bit_512k_0[15..0]
8. **FL_DQ[7..0]** – data_to_and_from_the_cfi_flash[7..0]

For input ports on the left, the names (from the top to bottom) are:

1. **FL_RST_N** – output of the always one
2. **LCD_ON** – output of the always one
3. **LCD_BLON** – output of the always one
4. **SD_DAT3** – output of the always one

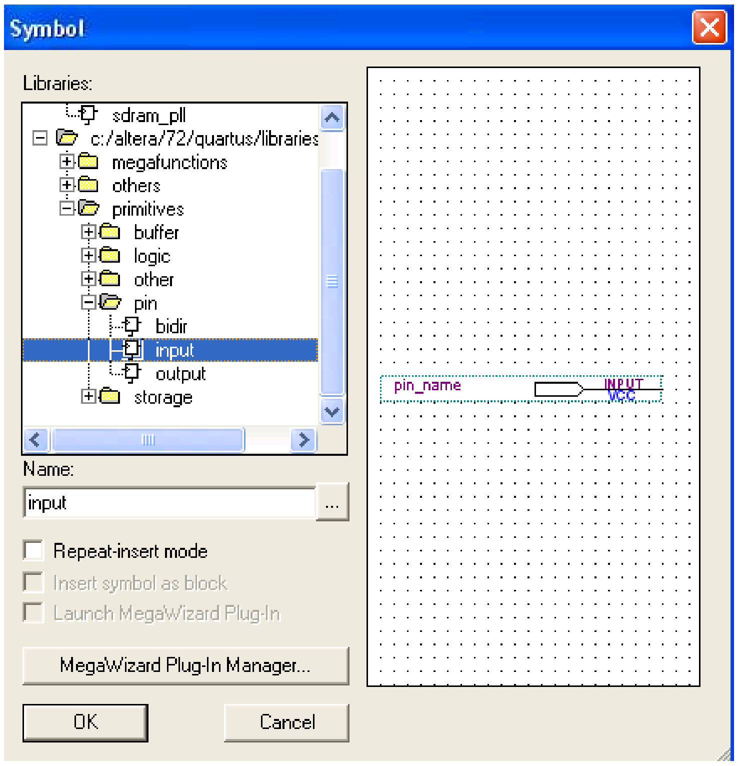
5. **CLOCK_50** – clk, .iOSC_50_to_the_dm9000a_0, inclk0 of sdram_pll
6. **reset_button** – reset_n
7. **KEY[2..0]** – in_port_to_the_buttons[2..0]
8. **ENET_INT-** ENET_INT_to_the_dm9000a_0
9. **OTG_INT0** – OTG_INT0_to_the_isp1362_0
10. **OTG_INT1** – OTG_INT0_to_the_isp1362_0
11. **SW[17..0]** – in_port_to_the_switches[17..0]
12. **UART_RXD** – rxd_to_the_uart

For output ports on the right:

1. **DRAM_CLK** – c0 of the sdram_pll
2. **SD_CLK** – out_port_from_the_SD_CLK
3. **ENET_CLK** – ENET_CLK_from_the_dm9000a_0
4. **ENET_CMD** – ENET_CMD_from_the_dm9000a_0
5. **ENET_CS_N** – ENET_CS_N_from_the_dm9000a
6. **ENET_RD_N** – ENET_RD_N_from_the_dm9000a_0
7. **ENET_RST_N** – ENET_RST_N_from_the_dm9000a_0
8. **ENET_WR_N** – ENET_WR_N_from_the_dm9000a_0
9. **OTG_ADDR[1..0]** – OTG_ADDR_from_the_isp1362_0[1..0]
10. **OTG_CS_N** – OTG_CS_N_from_the_isp1362_0
11. **OTG_RD_N** – OTG_RD_N_from_the_isp1362_0
12. **OTG_RST_N** – OTG_RST_N_from_the_isp1362_0
13. **OTG_WR_N** – OTG_WR_N_from_the_isp1362_0
14. **LCD_EN** – LCD_E_from_the_lcd
15. **LCD_RS** – LCD_RS_from_the_lcd
16. **LCD_RW** – LCD_RW_from_the_lcd
17. **LEDG[8..0]** – out_port_from_the_led_green[8..0]
18. **LEDR[17..0]** – out_port_from_the_led_red[17..0]
19. **DRAM_ADDR[11..0]** – zs_addr_from_the_sdram[11..0]
20. **DRAM_BA_1,DRAM_BA_0** – zs_ba_from_the_sdram
21. **DRAM_CAS_N** – zs_cas_n_from_the_sdram
22. **DRAM_CKE** – zs_cke_from_the_sdram
23. **DRAM_CS_N** – zs_cs_n_from_the_sdram
24. **DRAM_UDQM,DRAM_LDQM** – zs_dqm_from_the_sdram
25. **DRAM_RAS_N** – zs_ras_n_from_the_sdram
26. **DRAM_WE_N** – zs_we_n_from_the_sdram
27. **HEX0[6..0]** – oSEG0_from_the_seg_lut_8_0[6..0]
28. **HEX1[6..0]** – oSEG1_from_the_seg_lut_8_0[6..0]
29. **HEX2[6..0]** – oSEG2_from_the_seg_lut_8_0[6..0]
30. **HEX3[6..0]** – oSEG3_from_the_seg_lut_8_0[6..0]
31. **HEX4[6..0]** – oSEG4_from_the_seg_lut_8_0[6..0]
32. **HEX5[6..0]** – oSEG5_from_the_seg_lut_8_0[6..0]
33. **HEX6[6..0]** – oSEG6_from_the_seg_lut_8_0[6..0]
34. **HEX7[6..0]** – oSEG7_from_the_seg_lut_8_0[6..0]
35. **SRAM_ADDR[17..0]** – SRAM_ADDR_from_the_sram_16bit_512k_0[17..0]
36. **SRAM_CE_N** – SRAM_CE_N_from_the_sram_16bit_512k_0
37. **SRAM_LB_N** – SRAM_LB_N_from_the_sram_16bit_512k_0

38. **SRAM_OE_N** – SRAM_OE_N_from_the_sram_16bit_512k_0
39. **SRAM_UB_N** – SRAM_UB_N_from_the_sram_16bit_512k_0
40. **SRAM_WE_N** – SRAM_WE_N_from_the_sram_16bit_512k_0
41. **FL_ADDR[21..0]** – address_to_the_cfi_flash[21..0]
42. **FL_OE_N** – read_n_to_the_cfi_flash
43. **FL_CE_N** – select_n_to_the_cfi_flash
44. **FL_WE_N** – write_n_to_the_cfi_flash
45. **UART_TXD** – txd_from_the_uart

Connecting pins with the ports as detailed in Figure 1-48.



Name: input

Figure 1-47: Input pin from primitives

Note that there are some PIN names in the form of "Name1, Name2" which means emerging the Name1 and Name2 pins into a single bus, which is going to be assigned in a input/output bus. The next step is to assign the pins of the top level design to the pins of the FPGA chip. From Quartus menu, choose "**Assignments->Assignment Editor**". You can add an pin assignment by double clicking a field in the "**To**" column, enter in the **pin name of the top level design**, then choosing "**location**" in the same row of the "**Assignment Name**" column, and type in the value which is the **pin name on the FPGA** in the same row of the "**Value**" column, and finally set "**yes**" to "Enable" **column**. Figure 1-49 illustrates assigning CLOCK50 to the N2 on FPGA, which connects to the oscillator Y1 on the DE2.

To save your time, we provided a assignment file (lab1-pins.qsf) available to be imported from Quartus menu "**Assignments->Import assignments...**" and select the input file followed by clicking "OK" as shown in Figure 1-50. Remember to clear all existing assignments before importing.

Now the design is ready to be compiled, select “Start Compilation” from the “Processing” menu of Quartus II, or just simply click the purple arrow button to start the compilation process. After a successful compilation, a report will be generated as shown in Figure 1-51.

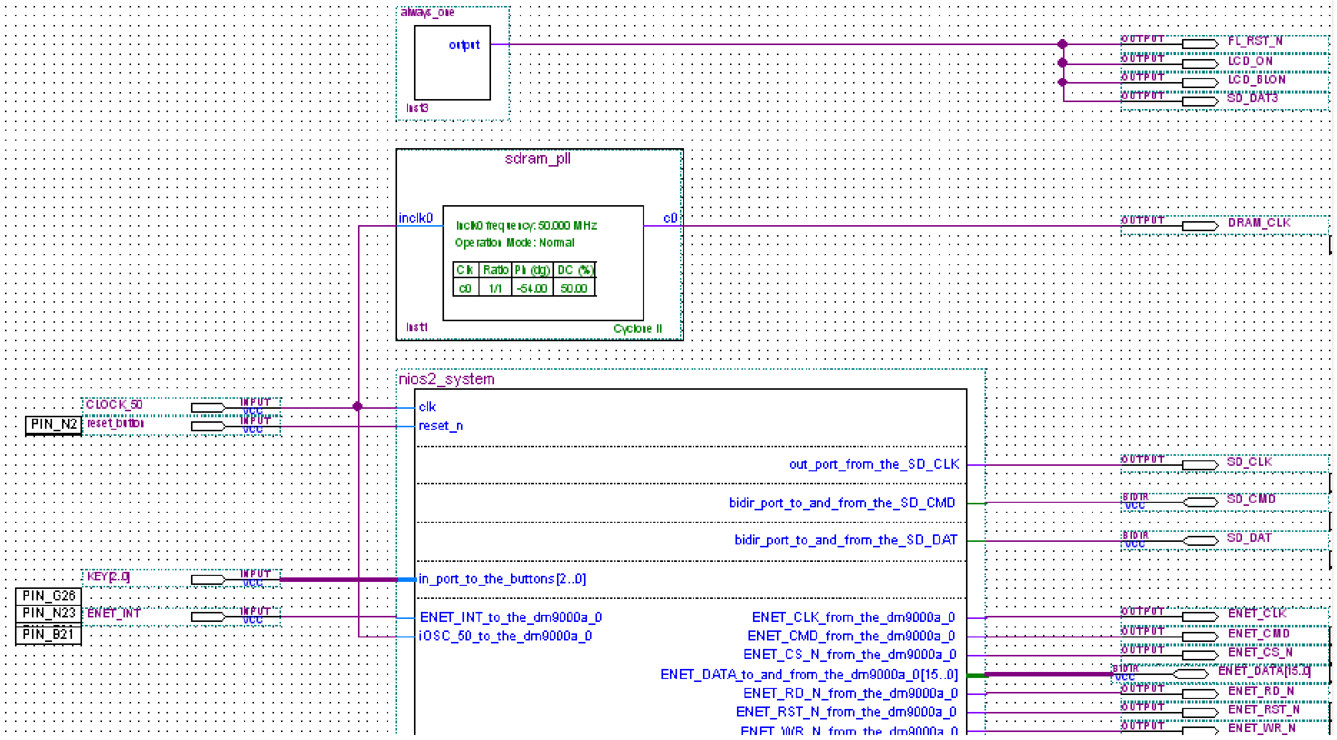
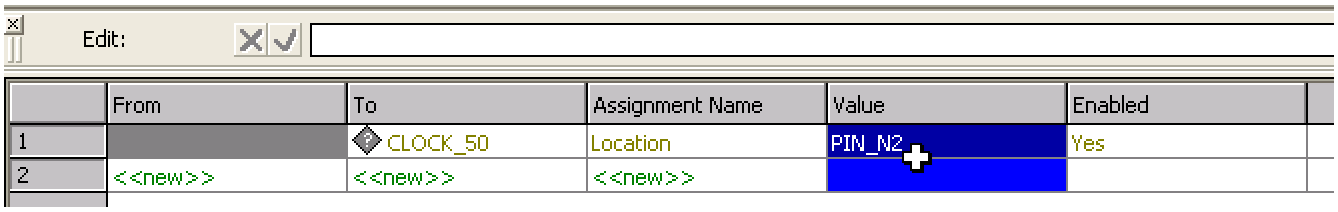
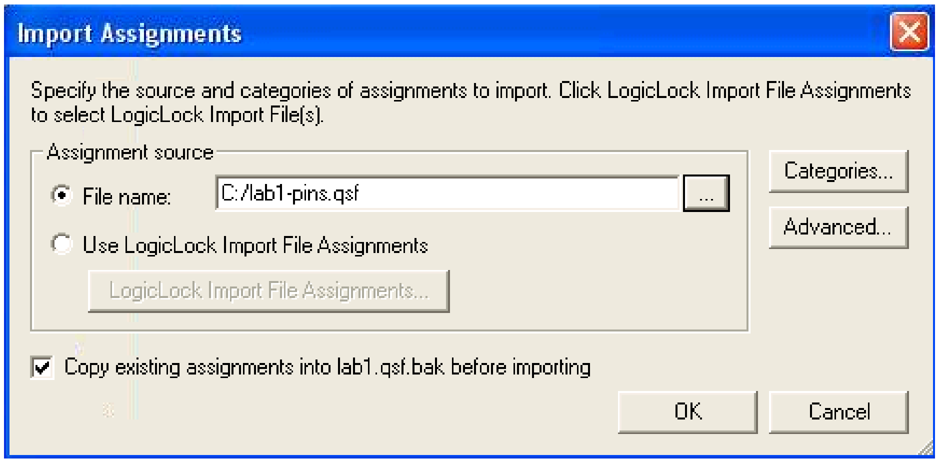


Figure 1-48: Segment of interconnections of the system


Edit:

| | From | To | Assignment Name | Value | Enabled |
|---|---|---|---|---|---|
| 1 | | CLOCK_50 | Location | PIN_N2 | Yes |
| 2 | <<new>> | <<new>> | <<new>> | | |

Figure 1-49: Assigning CLOCK50 to N2


Import Assignments

Specify the source and categories of assignments to import. Click LogicLock Import File Assignments to select LogicLock Import File(s).

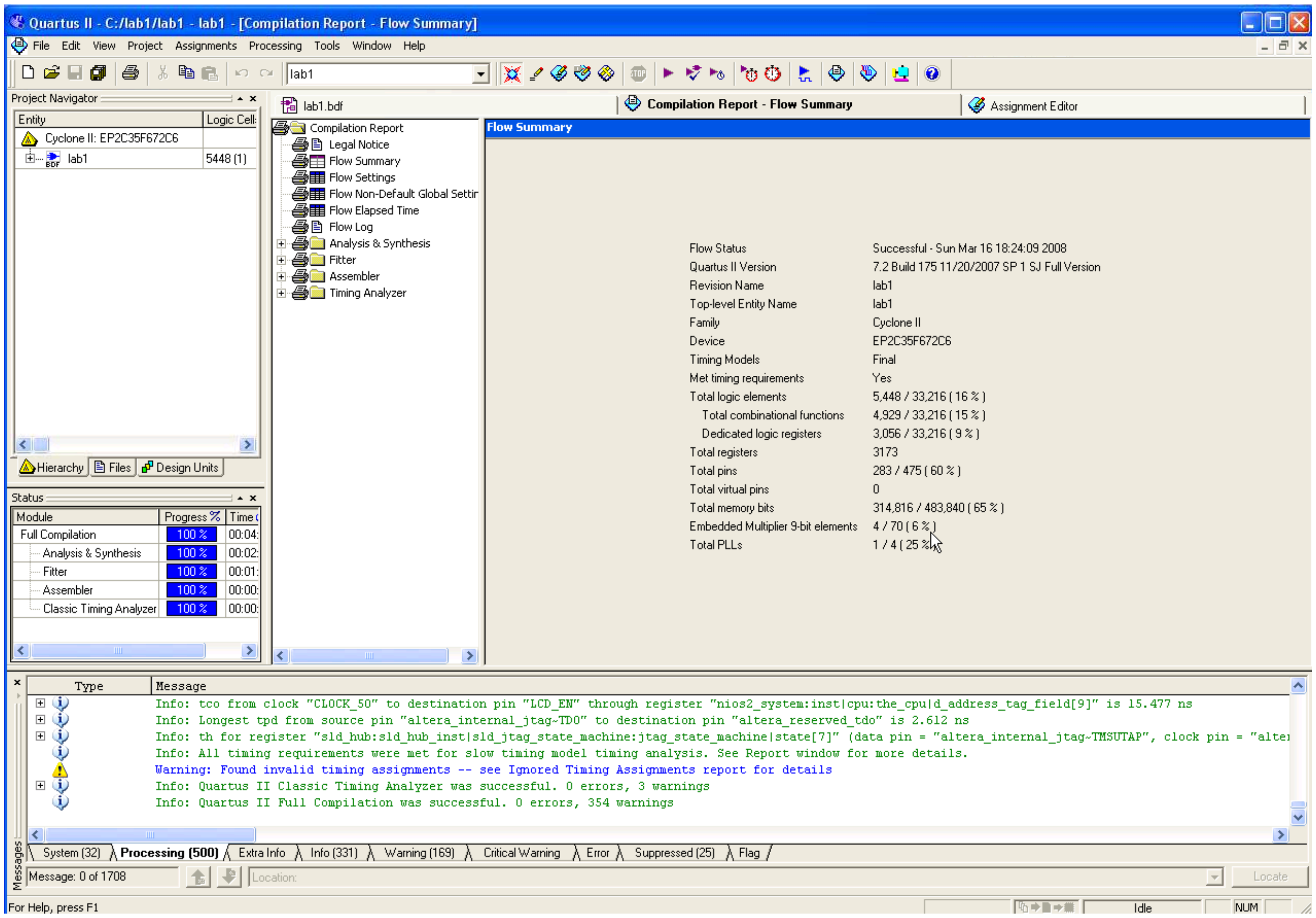
Assignment source

(●) File name: C:/lab1-pins.qsf

( ) Use LogicLock Import File Assignments

[✓] Copy existing assignments into lab1.qsf.bak before importing

Figure 1-50: Importing a existing assignment file



| Entity | Logic Cell |
|---|---|
| Cyclone II: EP2C35F672C6 | |
| lab1 | 5448 (1) |

| Module | Progress % | Time |
|---|---|---|
| Full Compilation | 100 % | 00:04: |
| Analysis & Synthesis | 100 % | 00:02: |
| Fitter | 100 % | 00:01: |
| Assembler | 100 % | 00:00: |
| Classic Timing Analyzer | 100 % | 00:00: |

Flow Summary

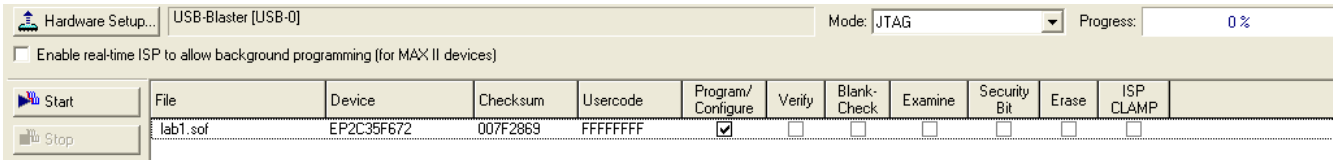
| | |
|---|---|
| Flow Status | Successful - Sun Mar 16 18:24:09 2008 |
| Quartus II Version | 7.2 Build 175 11/20/2007 SP 1 SJ Full Version |
| Revision Name | lab1 |
| Top-level Entity Name | lab1 |
| Family | Cyclone II |
| Device | EP2C35F672C6 |
| Timing Models | Final |
| Met timing requirements | Yes |
| Total logic elements | 5,448 / 33,216 ( 16 % ) |
| Total combinational functions | 4,929 / 33,216 ( 15 % ) |
| Dedicated logic registers | 3,056 / 33,216 ( 9 % ) |
| Total registers | 3173 |
| Total pins | 283 / 475 ( 60 % ) |
| Total virtual pins | 0 |
| Total memory bits | 314,816 / 483,840 ( 65 % ) |
| Embedded Multiplier 9-bit elements | 4 / 70 ( 6 % ) |
| Total PLLs | 1 / 4 ( 25 % ) |

```
Type    Message
Info: tco from clock "CLOCK_50" to destination pin "LCD_EN" through register "nios2_system:inst|cpu:the_cpu|d_address_tag_field[9]" is 15.477 ns
Info: Longest tpd from source pin "altera_internal_jtag~TDO" to destination pin "altera_reserved_tdo" is 2.612 ns
Info: th for register "sld_hub:sld_hub_inst|sld_jtag_state_machine:jtag_state_machine|state[7]" (data pin = "altera_internal_jtag~TMSUTAP", clock pin = "alter
Info: All timing requirements were met for slow timing model timing analysis. See Report window for more details.
Warning: Found invalid timing assignments -- see Ignored Timing Assignments report for details
Info: Quartus II Classic Timing Analyzer was successful. 0 errors, 3 warnings
Info: Quartus II Full Compilation was successful. 0 errors, 354 warnings
```

Figure 1-51: Successful compilation of the hardware design

## 1.4 Download the lab1 digital design to the FPGA chip

Connect the DE2 board with power, and Blaster port to the workstation via USB cable, and select "**Programmer**" from the "**Tools**" menu to open up the programmer as shown in Figure 1-52. Click the "**Hardware Setup**" button to make sure the current selected hardware is **USB Blaster** as shown in Figure 1-53. If not, add one up through "**Add Hardware**" button beside. By clicking the "Add File" or "Change File" button at the left to include "lab1.sof" and tick the box under the "Program/Configure". Click "**Start**" below the "**Hardware Setup**" to download the hardware design to the DE2 board. You will find the progress bar to be 100% when the process is completed. Now you are ready to start the software implementation of the Nios II system.

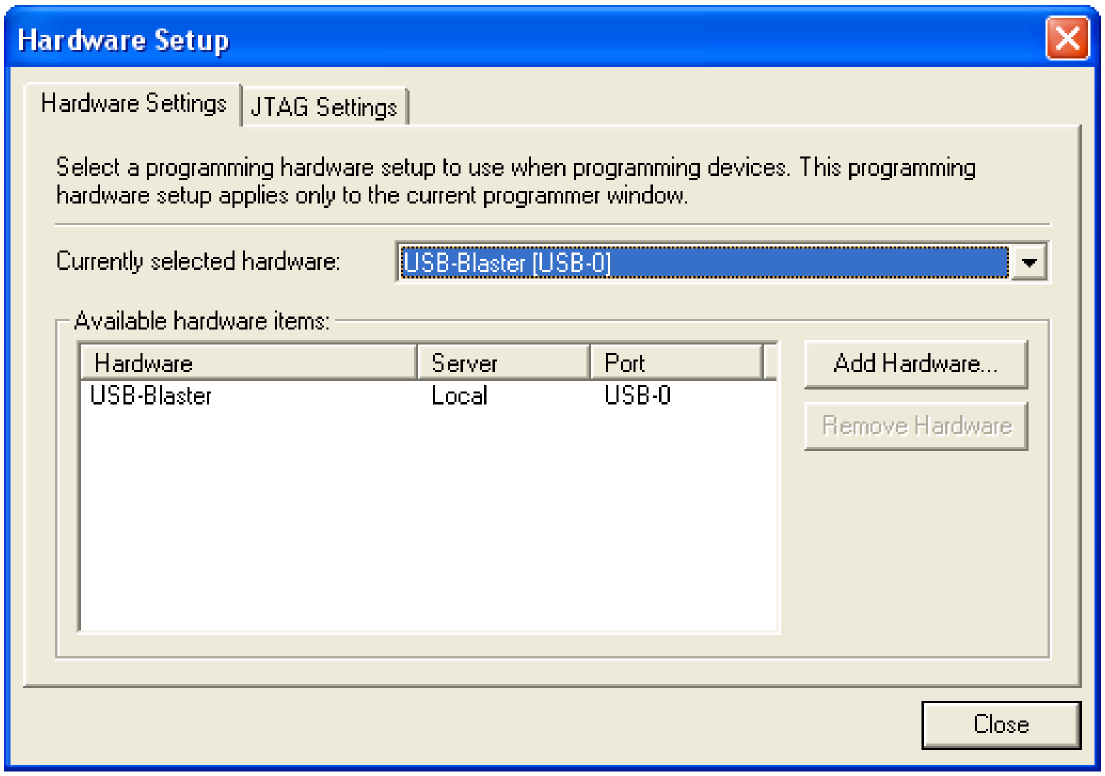


USB-Blaster [USB-0]   Mode: JTAG   Progress: 0 %

Enable real-time ISP to allow background programming (for MAX II devices)

| File | Device | Checksum | Usercode | Program/ Configure | Verify | Blank-Check | Examine | Security Bit | Erase | ISP CLAMP |
|---|---|---|---|---|---|---|---|---|---|---|
| lab1.sof | EP2C35F672 | 007F2869 | FFFFFFFF | ☑ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ |

Figure 1-52: Programmer of Quartus II



Select a programming hardware setup to use when programming devices. This programming hardware setup applies only to the current programmer window.

Currently selected hardware: USB-Blaster [USB-0]

Available hardware items:

| Hardware | Server | Port |
|---|---|---|
| USB-Blaster | Local | USB-0 |

Figure 1-53: Hardware settings of the download cable