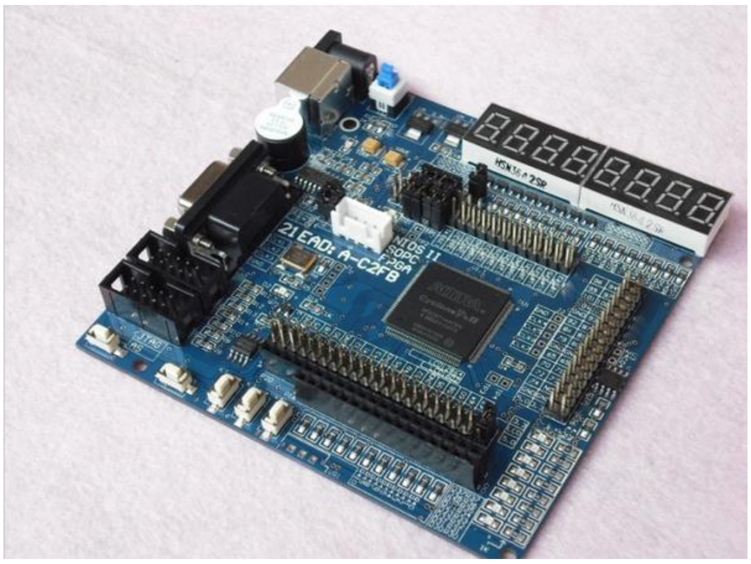
NIOS II – Experiment 1

Scope

The scope of this document is reporting of experiments learning the NIOS II design cycle based upon the Altera®'s SOPC Builder Software when developing for the non-Altera designed board from Master 21EDA.



References on Web

PDF Resources: Designing the Hardware

[1] Altera® Introduction to the Altera SOPC Builder Using VHDL Designs

[2] Altera® SOPC Builder User Guide

[3] NIOS II Hardware Development Tutorial

[4] Translated manual for Master 21EDA board

PDF Resources: Writing and compiling the Software

[5] Altera Monitor Program manual

[6] My First NIOS II Software Tutorial

[7] NIOS Software Build Tools

1 Introduction

The **Headings in red** in this document will mirror the headings in the Altera® tutorial Ref [1] so you can easily map between documents. You **WILL NEED** Ref [1] at least as this document is only providing the gotchas when walking through Ref [1]. Additional **Headings in blue** are internal to this document – used to break things up as you would expect headings to do.

Read the previous paragraph again. You are reading this document along with the Altera® tutorial [1].

Remember also from the blog, we are using Quartus® II version 10.1 – driven by the chip on the board, the 144-pin EP2C5T144C8 Cyclone II.

Legend:

If I have been stumped by something I will use the image to the left to let you know a little investigation was in order.

If an important "Ah Ha!" moment occurred, I will also let you know.

If you're to go to the web I will give the hint.

STOP, we are swapping tutorials

Now don't forget something very important. Quartus ® II is clunky. Recall from the blog the crashing. What you will find is you may need to delete project and start again a couple of times so be prepared both spiritually and emotionally. You will find the Altera® tutorial leaves things out (which we will try to catch). You will also find, as I did, the tool may not even crash, but will not react to menu selections etc. Just take a deep breath and SCREAM, get over it and try again.

Note especially, I'll be muddling through and deleting and restarting in the background – I am also unlikely to find every bug or crash or hung IDE that you will, so do feel free to restart if even this guide doesn't help.

Note also, I will be calling out figures, occasionally, from the Altera® tutorial so I will use Figure x for figures internal to this document and ***Figure y*** when referring to figures in the Altera® tutorial. Similarly, I will use ***Step x***. ***Table x.*** etc. to help remind you to go to the Altera® tutorial.

Ready, set, let's go.

2 Nios II System

The board we are using does not have SRAM, SDRAM – though it does have a flash memory chip. For this tutorial we are not using external ram and are limited by the number of switches to four – see Figure 1 below.

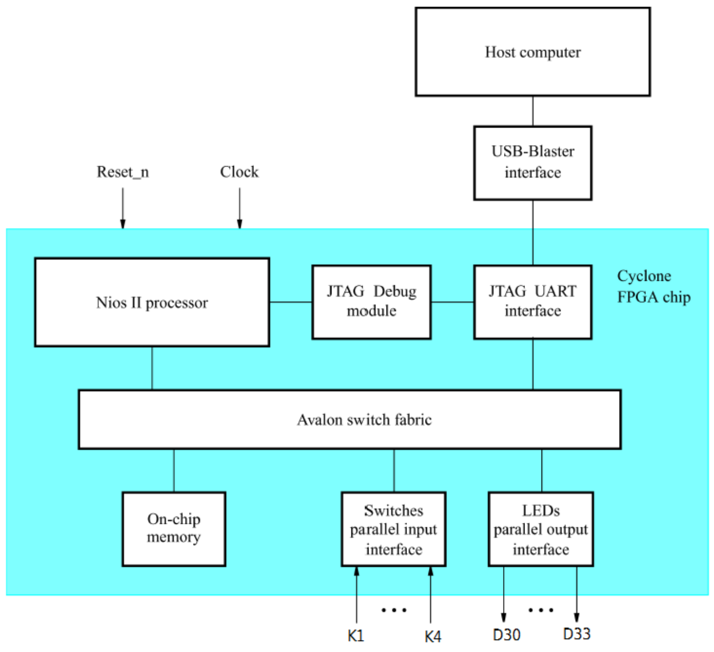


Figure 1. Our simple example Nios II system

The system realizes a trivial task. Four toggle switches on the Master 21EDA board, K1..4 [3], will be used to turn on or off the four red LEDs, D30..33 [3][1].

The focus is on porting the SOPC tutorial to this board and to learn the SOPC design cycle so we won't move too far away from the original.

Basic steps are:

1. Create a project called "lights".
2. Build the "nios_system" (a Nios II soft core) with the SOPC Builder.
3. Code an assembly routine called "nios_system.s".
4. Install and run the assembled code on our Nios II soft core.
5. Code a C routine called "nios_system.c".
6. Install and run the compiled code on our Nios II soft core.
7. Add use of Nios II SBT in Eclipse (here is where we jump out of Ref [1] and into Ref [6]).
8. SCREAM!

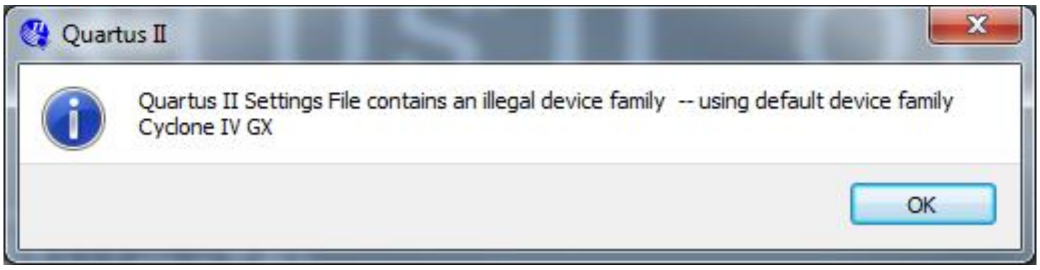
[1] We won't be using all 8 red LED's this tutorial.

3 Altera's SOPC Builder

Just follow the steps in the tutorial.

Have Ref [2] and Ref [3] close by. We won't call out to then here but you are advised to skim them once and then go back to them to sort issues or to consolidate your learning.

***Step 1.* of the Altera® tutorial** Ref [1] may raise the following error window (Figure 2).


Quartus II

Quartus II Settings File contains an illegal device family -- using default device family Cyclone IV GX

Figure 2. Whoops!

Ignore this, just select "OK" and press on. I found, when creating the project, the "Device Select" actually defaulted to Cyclone II, which suits us right!?

Do note:

- The file dialog for the Quartus ® II project wizard, the one you use to create the project folder, is flaky. It may flicker, or exit early leaving a half typed directory name, blah, etc. I would recommend creating a target directory in the Windows ™ file explorer and use the Quartus ® II project wizard browser to move to that folder.
- There may be some problems later when we start with the SOPC tool with the project naming. We'll sort that out. The problem will be somewhere in the tutorial the SOPC filename has not been set and when we go to generate the VHDL files later, the top file will be "unnamed.vhd". We'll elaborate the root-cause once we get to the errant step, I suspect we need to save the SOPC project before generation.

Now, so we don't get into trouble with screaming buzzers and all LEDs on later, we need to do something that is missing from the Altera ® tutorial (there will be a few things along the way).

Save your project and then select the "Device" menu item under "Assignments" menu (Figure 3).

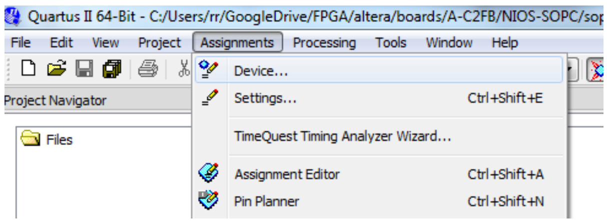
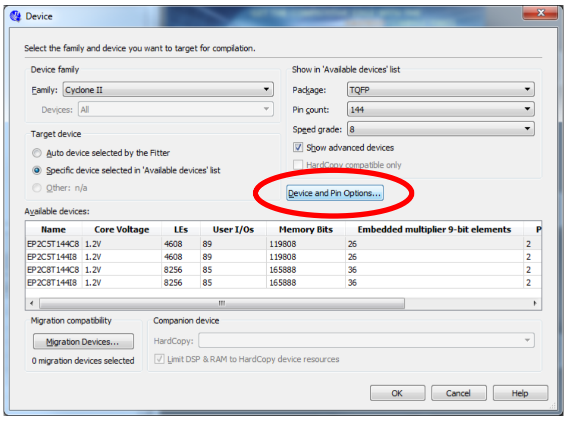

Quartus II 64-Bit - C:/Users/rr/GoogleDrive/FPGA/altera/boards/A-C2FB/NIOS-SOPC/sop
File
Edit
View
Project
Assignments
Processing
Tools
Window
Help
Project Navigator
Files
Device...
Settings... Ctrl+Shift+E
TimeQuest Timing Analyzer Wizard...
Assignment Editor Ctrl+Shift+A
Pin Planner Ctrl+Shift+N

Figure 3. An important missing step.

You will see a button "Device and Pin Options …", select that (Figure 4). This button doesn't exist on the dialog when the project is created so you will need to do this as separate step – right now.


Device

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

Device family
Family: Cyclone II
Devices: All

Show in 'Available devices' list
Package: TQFP
Pin count: 144
Speed grade: 8
[✓] Show advanced devices
[ ] HardCopy compatible only

Target device
( ) Auto device selected by the Fitter
(●) Specific device selected in 'Available devices' list
( ) Other: n/a

Available devices:

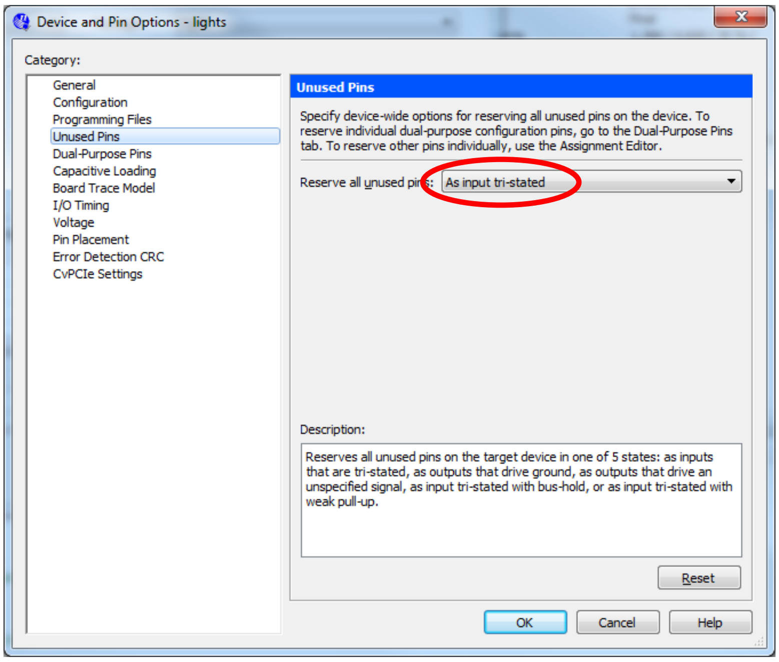
Name	Core Voltage	LEs	User I/Os	Memory Bits	Embedded multiplier 9-bit elements	P
EP2C5T144C8	1.2V	4608	89	119808	26	2
EP2C5T144I8	1.2V	4608	89	119808	26	2
EP2C8T144C8	1.2V	8256	85	165888	36	2
EP2C8T144I8	1.2V	8256	85	165888	36	2

Migration compatibility
0 migration devices selected

Companion device
HardCopy:
[✓] Limit DSP & RAM to HardCopy device resources

Figure 4. We need to do something with our unused pins!

Change unused pins to tri-stated inputs (Figure 5).


Device and Pin Options - lights

Category:
General
Configuration
Programming Files
Unused Pins
Dual-Purpose Pins
Capacitive Loading
Board Trace Model
I/O Timing
Voltage
Pin Placement
Error Detection CRC
CvPCIe Settings

Unused Pins

Specify device-wide options for reserving all unused pins on the device. To reserve individual dual-purpose configuration pins, go to the Dual-Purpose Pins tab. To reserve other pins individually, use the Assignment Editor.

Reserve all unused pins: As input tri-stated

Description:

Reserves all unused pins on the target device in one of 5 states: as inputs that are tri-stated, as outputs that drive ground, as outputs that drive an unspecified signal, as input tri-stated with bus-hold, or as input tri-stated with weak pull-up.

Figure 5. Tell the pins to be "quiet"

If you don't do this, this is why (later) your buzzer will buzz and all your LEDs will light.

Now you can move onto ***Step 2.* of the tutorial**.

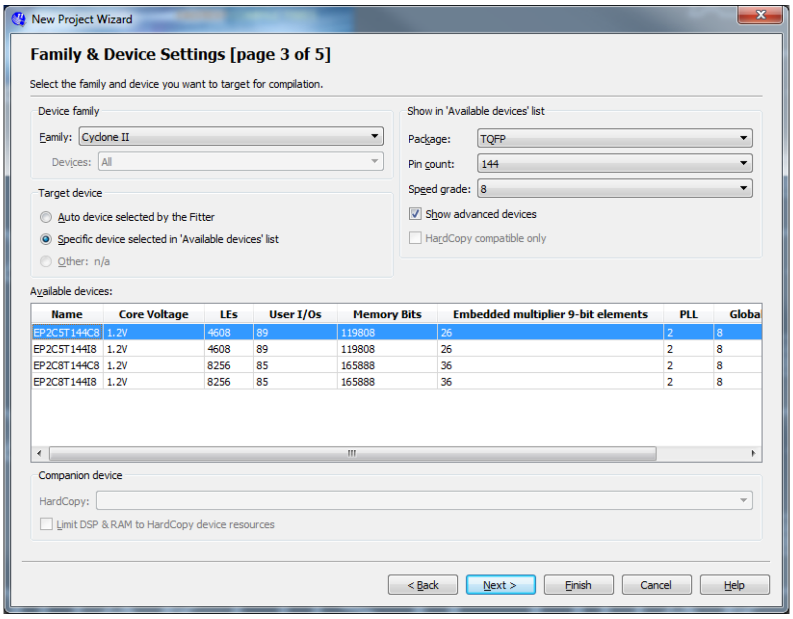
When you get to ***Step 2.*** of the Altera® tutorial, replace the devices in ***Table 1.*** with ours:

Board	Device Name
Product ID: 1758618627	Cyclone II EP2C5T144C8

Table 1. Master 21EDA device name

That is, adjust the "Family & Device Settings" window to look like that below in Figure 6.

Yes, we are setting the chip type in the SOPC Builder, it doesn't appear to pick up the chip type from the overall project, not sure why not, there may be a trick but I never found it.



New Project Wizard

Family & Device Settings [page 3 of 5]

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

Device family
Family: Cyclone II
Devices: All

Target device
( ) Auto device selected by the Fitter
(●) Specific device selected in 'Available devices' list
( ) Other: n/a

Show in 'Available devices' list
Package: TQFP
Pin count: 144
Speed grade: 8
[✓] Show advanced devices
[ ] HardCopy compatible only

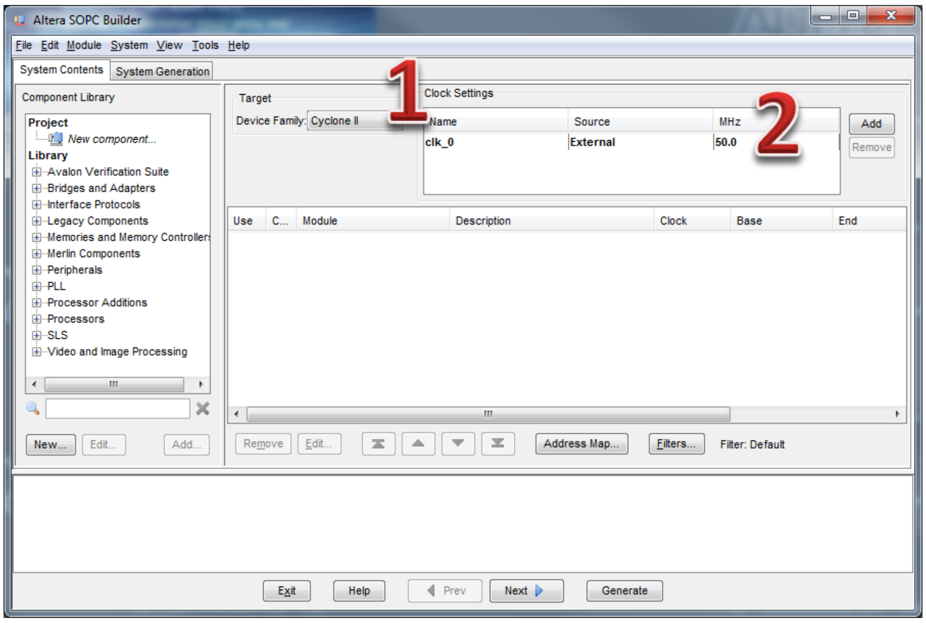
Available devices:

Name	Core Voltage	LEs	User I/Os	Memory Bits	Embedded multiplier 9-bit elements	PLL	Globa
EP2C5T144C8	1.2V	4608	89	119808	26	2	8
EP2C5T144I8	1.2V	4608	89	119808	26	2	8
EP2C8T144C8	1.2V	8256	85	165888	36	2	8
EP2C8T144I8	1.2V	8256	85	165888	36	2	8

Companion device
HardCopy:
[ ] Limit DSP & RAM to HardCopy device resources

Figure 6. Our chip

Okay, so muddle through to ***Step 3.*** and ***Step 4.*** and we should still be good as 1) "Device Family" will have been picked up from project creation and 2) our board has a 50.0 MHz clock (Figure 7).



Target

Device Family: Cyclone II

Clock Settings

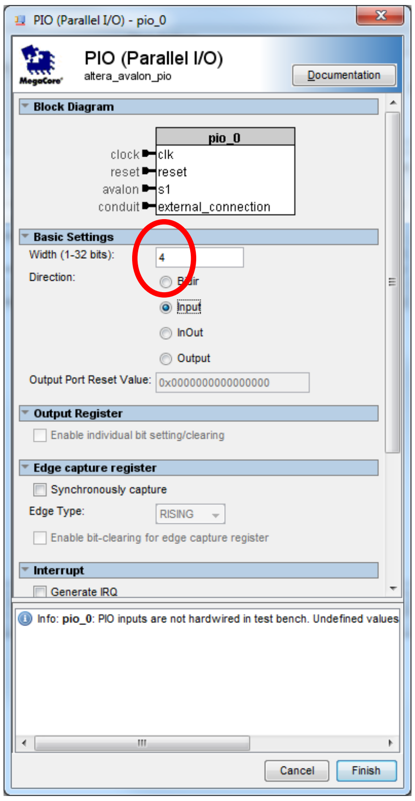
Name	Source	MHz
clk_0	External	50.0

Figure 7. Same same

I recommend you goto "File->Save", in the SOPC builder, and save the file as "nios_system". The reason is, as was mentioned before, when you generate the system later you will have the system called "unnamed.vhd" otherwise. Who knows why they didn't code the software so that the project file save dialog picked up the project name, trust me, just do it.

Now do ***Step 5.*** to ***Step 7.*** and then hold your horses.

In ***Step 7.*** we are setting the inputs from the switches so we will have to reduce the number down to 4 from the 8 recommended in the Altera® tutorial - our board has only four buttons. ***Figure 10***. in the Altera® tutorial is replaced by the Figure 8 (over page).


PIO (Parallel I/O) - pio_0

PIO (Parallel I/O)
altera_avalon_pio

Block Diagram

pio_0: clock → clk, reset → reset, avalon → s1, conduit → external_connection

Basic Settings

Width (1-32 bits): 4

Direction: Bidir / [✓ Input] / InOut / Output

Output Port Reset Value: 0x0000000000000000

Output Register

Enable individual bit setting/clearing

Edge capture register

Synchronously capture

Edge Type: RISING

Enable bit-clearing for edge capture register

Interrupt

Generate IRQ

Info: pio_0: PIO inputs are not hardwired in test bench. Undefined values

Figure 8. Our replacement for Altera's tutorial's Figure 10.

Similarly for ***Step 8.*** we are only having 4 outputs – this is simply as we are pairing Buttons to LEDs.

Okay, bumble through ***Step 9.*** and in ***Step 10.*** where we will take a small liberty. The tutorial suggests renaming the inputs "Switches", I have opted for "Buttons".

We can all agree and LED is an LED is an LED and call the outputs "LEDs".

Step 11. is insidious. It may appear to do naught, however if you look at the diagrams in the tutorial, and note the differences in address displayed between ***Figure 13.*** and ***Figure 14.*** of the tutorial document you'll notes something does happen.

Also, note there is a second "quiet" action to perform that doesn't appear in the tutorial. Namely, run the "Auto-Assign IRQs" (Figure 9 below).

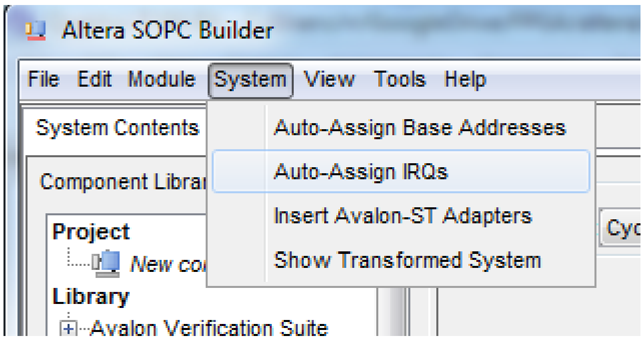
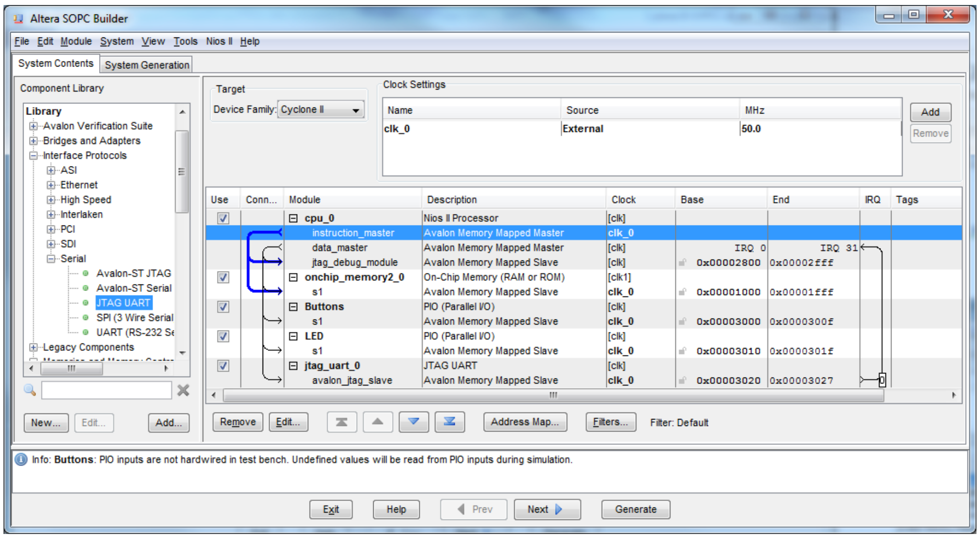

Altera SOPC Builder
File Edit Module System View Tools Help
System Contents
Component Librar
Project
New co
Library
Avalon Verification Suite
Auto-Assign Base Addresses
Auto-Assign IRQs
Insert Avalon-ST Adapters
Show Transformed System
Cyc

Figure 9. Missing setup step.

So, we end up with Figure 10 below.



Target
Device Family: Cyclone II

Clock Settings

Name	Source	MHz
clk_0	External	50.0

Use	Conn...	Module	Description	Clock	Base	End	IRQ	Tags
✓		cpu_0	Nios II Processor	[clk]				
		instruction_master	Avalon Memory Mapped Master	clk_0				
		data_master	Avalon Memory Mapped Master	[clk]	IRQ 0	IRQ 31		
		jtag_debug_module	Avalon Memory Mapped Slave	[clk]	0x00002800	0x00002fff		
✓		onchip_memory2_0	On-Chip Memory (RAM or ROM)	[clk1]				
		s1	Avalon Memory Mapped Slave	clk_0	0x00001000	0x00001fff		
✓		Buttons	PIO (Parallel I/O)	[clk]				
		s1	Avalon Memory Mapped Slave	clk_0	0x00003000	0x0000300f		
✓		LED	PIO (Parallel I/O)	[clk]				
		s1	Avalon Memory Mapped Slave	clk_0	0x00003010	0x0000301f		
✓		jtag_uart_0	JTAG UART	[clk]				
		avalon_jtag_slave	Avalon Memory Mapped Slave	clk_0	0x00003020	0x00003027	0	

Filter: Default

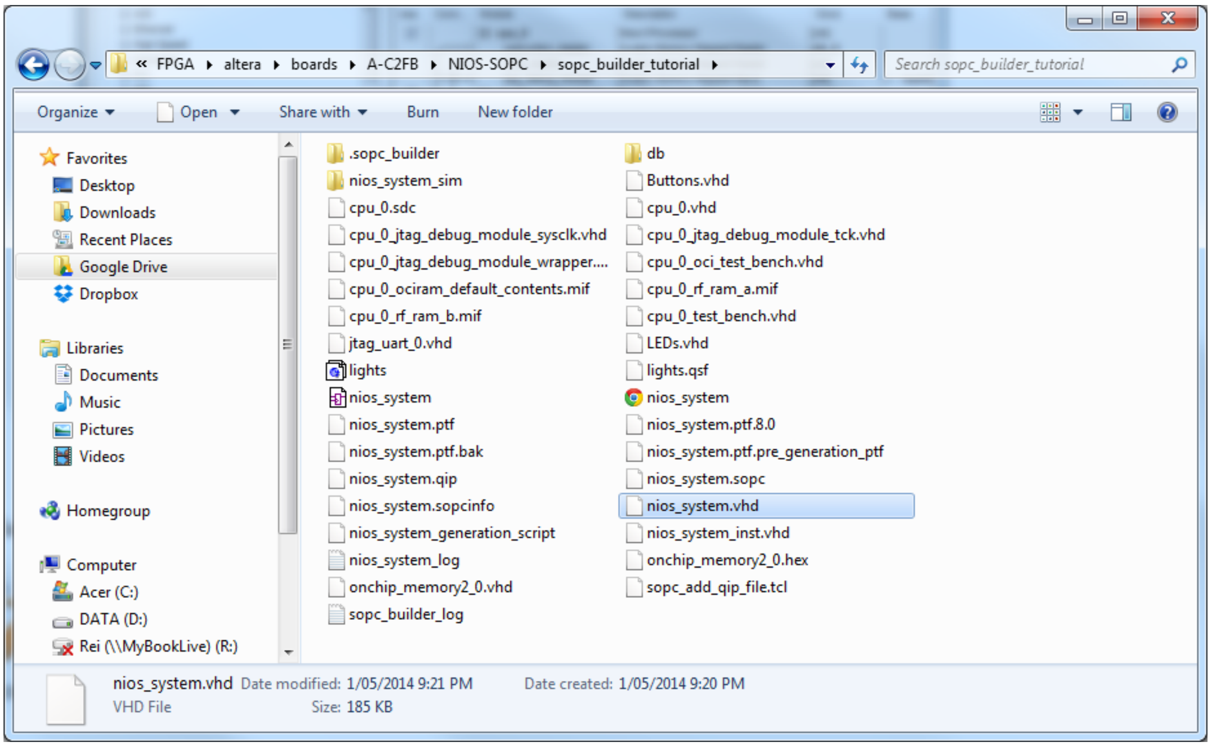
Info: **Buttons**: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.

Figure 10. Almost there

In fact, we can amble through the rest of the steps in the section, simply following the tutorial.

Hit "File->Save" before you generate at ***Step 13.*** If you ignored me earlier, now is the time to call the file "nios_system" as the SOPC Builder should ask you for a file name before generating – however I did generate a system without it asking in my first pass – resulting in the file "unnamed.vhd", did I mention that?

Generation should take a little while. Once it is done, take a break and use File Explorer to open up and look at your project directory (Figure 11 over the page). Note the highlighted file. This is called out in the next section and will be the one called "unknown.vhd" if don't save the SOPC project before generation. I did warn you!



```
.sopc_builder                              db
nios_system_sim                            Buttons.vhd
cpu_0.sdc                                  cpu_0.vhd
cpu_0_jtag_debug_module_sysclk.vhd         cpu_0_jtag_debug_module_tck.vhd
cpu_0_jtag_debug_module_wrapper....        cpu_0_oci_test_bench.vhd
cpu_0_ociram_default_contents.mif          cpu_0_rf_ram_a.mif
cpu_0_rf_ram_b.mif                         cpu_0_test_bench.vhd
jtag_uart_0.vhd                            LEDs.vhd
lights                                     lights.qsf
nios_system                                nios_system
nios_system.ptf                            nios_system.ptf.8.0
nios_system.ptf.bak                        nios_system.ptf.pre_generation_ptf
nios_system.qip                            nios_system.sopc
nios_system.sopcinfo                       nios_system.vhd
nios_system_generation_script              nios_system_inst.vhd
nios_system_log                            onchip_memory2_0.hex
onchip_memory2_0.vhd                       sopc_add_qip_file.tcl
sopc_builder_log

nios_system.vhd  Date modified: 1/05/2014 9:21 PM      Date created: 1/05/2014 9:20 PM
VHD File         Size: 185 KB
```

Figure 11. That's better, a named system file!

4 Integration of the Nios II System into a Quartus II Project

4.1 Instantiation of the Module Generated by the SOPC Builder

Note the misleading statement second paragraph of **page 18** of tutorial, namely:

> "The VHDL entity generated by the SOPC Builder is in the file nios_system.vhd in the directory of the project. Note that the name of the VHDL entity is the same as the system name specified when first using the SOPC Builder."

This will only be true if you hand entered the "nios_system" into the save dialog earlier as suggested. It will be "unnamed.vhd" otherwise. Again, I did warn you!

Now you have a file called "nios_system.vhd", open it and look for "entity nios_system is".

In ***Figure 17.*** of the tutorial the input and output ports are 8 bits wide (7 DOWNTO 0). You'll recall we set ours to 4 bits, so let's check this worked, ours will (should) look like Figure 12 below.

```
entity nios_system is
        port (
              -- 1) global signals:
                 signal clk_0 : IN STD_LOGIC;
                 signal reset_n : IN STD_LOGIC;

              -- the_Buttons
                 signal in_port_to_the_Buttons : IN
STD_LOGIC_VECTOR (3 DOWNTO 0);

              -- the_LEDs
                 signal out_port_from_the_LEDs : OUT
STD_LOGIC_VECTOR (3 DOWNTO 0)
              );
end entity nios_system;
```

Figure 12. 4 not 8 bits of in/out port

Now, buried in the text is that we need to write "lights.vhd" so goto the "Files" menu, and "Add" a new file of type "VHDL" (Figure 13 below).

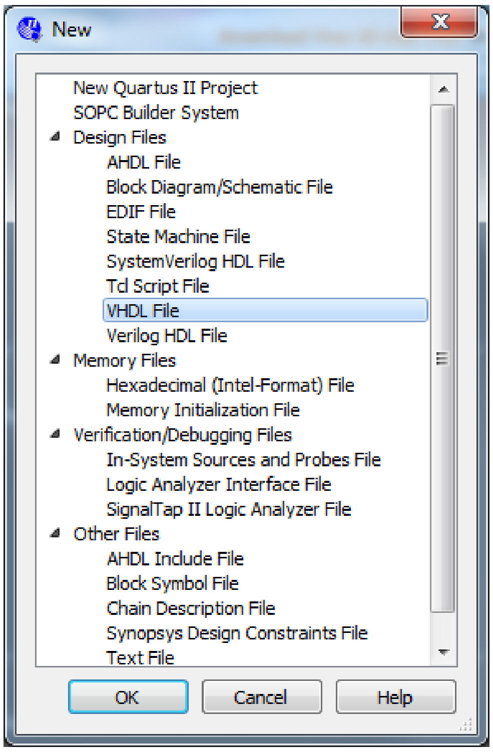
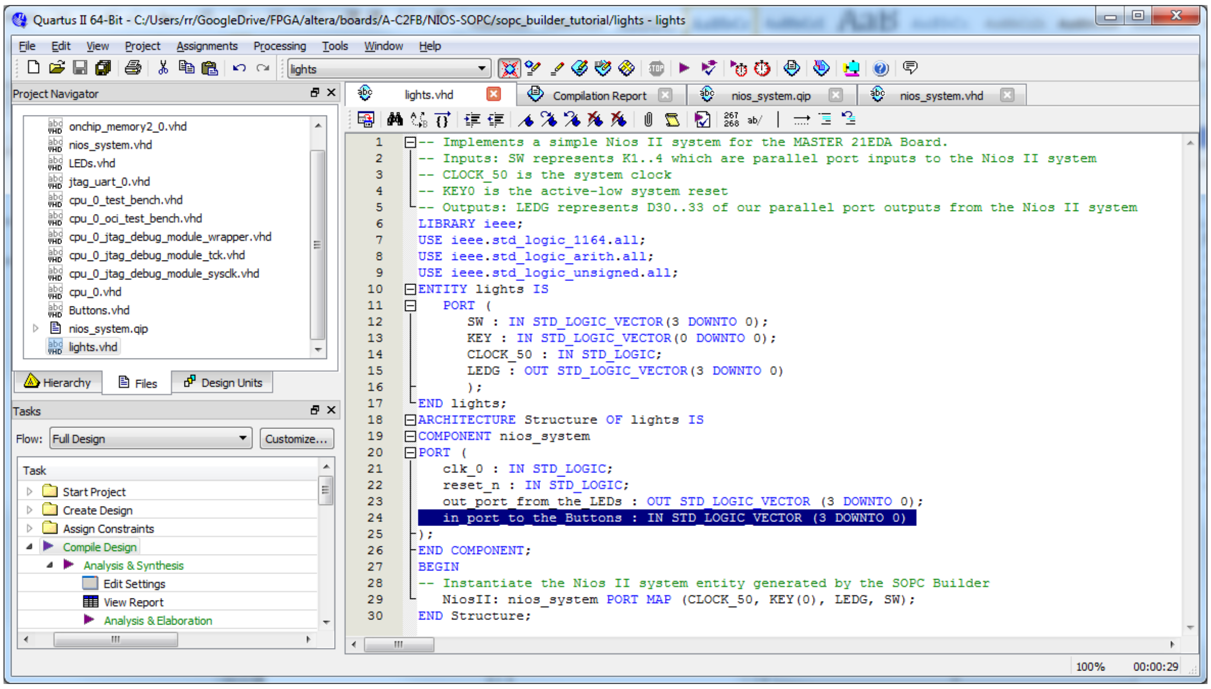


Figure 13. Create "lights.vhd"

Ignore ***Figure 18.*** and edit up your "lights.vhd" to look like our Figure 14 below.

Note here where the code is (3 DOWNTO 0) instead of (7 DOWNTO 0) as in the tutorial.

Also notice where we have changed the input name from "~~Switches~~" to "Buttons" to match what we did in the SPOC builder (highlighted text).



```vhdl
-- Implements a simple Nios II system for the MASTER 21EDA Board.
-- Inputs: SW represents K1..4 which are parallel port inputs to the Nios II system
-- CLOCK_50 is the system clock
-- KEY0 is the active-low system reset
-- Outputs: LEDG represents D30..33 of our parallel port outputs from the Nios II system
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;
ENTITY lights IS
   PORT (
      SW : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      KEY : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
      CLOCK_50 : IN STD_LOGIC;
      LEDG : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
      );
END lights;
ARCHITECTURE Structure OF lights IS
COMPONENT nios_system
PORT (
   clk_0 : IN STD_LOGIC;
   reset_n : IN STD_LOGIC;
   out_port_from_the_LEDs : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
   in_port_to_the_Buttons : IN STD_LOGIC_VECTOR (3 DOWNTO 0)
);
END COMPONENT;
BEGIN
-- Instantiate the Nios II system entity generated by the SOPC Builder
   NiosII: nios_system PORT MAP (CLOCK_50, KEY(0), LEDG, SW);
END Structure;
```

Figure 14. MASTER 21EDA version of "lights.vhd"

To add all of your files from the SOPC generation, go to "Files" tab, right click mouse and add files. It makes it easier to sort files by type "vhd" when adding a more than one file (Figure 15).

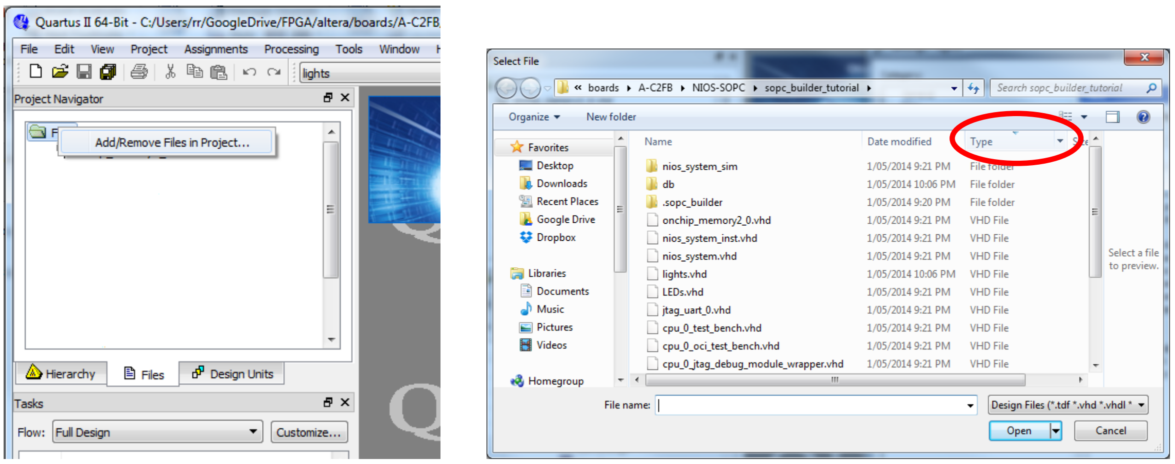


Figure 15. Sort them first

Now, do go back and remove "nios_system_inst.vhd". Sorry, that is the example instantiation file the SPOC Builder creates for us – we don't need it as "lights.vhd" will instantiate the nios_system.

Also, you may have to select "lights.vhd" and make it the top element in your design (Figure 16). This may or may not be necessary, the software may pick it up as the top because it has the same name as the project – but let's make sure.

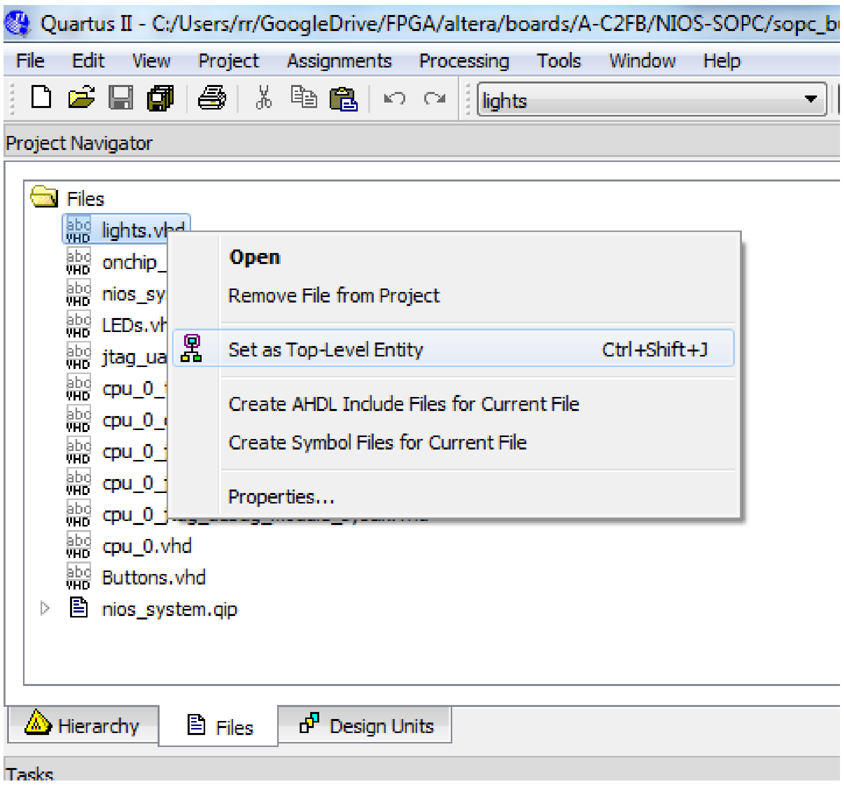

Quartus II - C:/Users/rr/GoogleDrive/FPGA/altera/boards/A-C2FB/NIOS-SOPC/sopc_b
File Edit View Project Assignments Processing Tools Window Help
lights
Project Navigator
Files
lights.vhd
onchip_
nios_sy
LEDs.vh
jtag_ua
cpu_0_
cpu_0_
cpu_0_
cpu_0_
cpu_0_
cpu_0.vhd
Buttons.vhd
nios_system.qip
Open
Remove File from Project
Set as Top-Level Entity Ctrl+Shift+J
Create AHDL Include Files for Current File
Create Symbol Files for Current File
Properties...
Hierarchy
Files
Design Units
Tasks

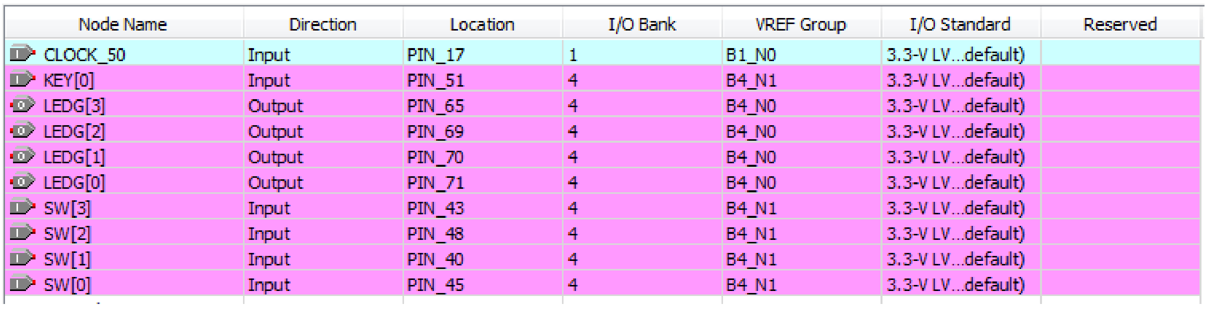
Figure 16. Top Kat!

Note, don't forget to set the pin assignments in the pin assignments dialog, for our board that is the following:

K1	INPUT	PIN_43
K2	INPUT	PIN_48
K3	INPUT	PIN_40
K4	INPUT	PIN_45
CLK_50M	INPUT	PIN_17
D30	OUTPUT	PIN_65
D31	OUTPUT	PIN_69
D32	OUTPUT	PIN_70
D33	OUTPUT	PIN_71

The pins in the table above (bottom of preceding page) are from Ref [4] or from the spreadsheet that comes with your board.

You should end up with the following (Figure 17).

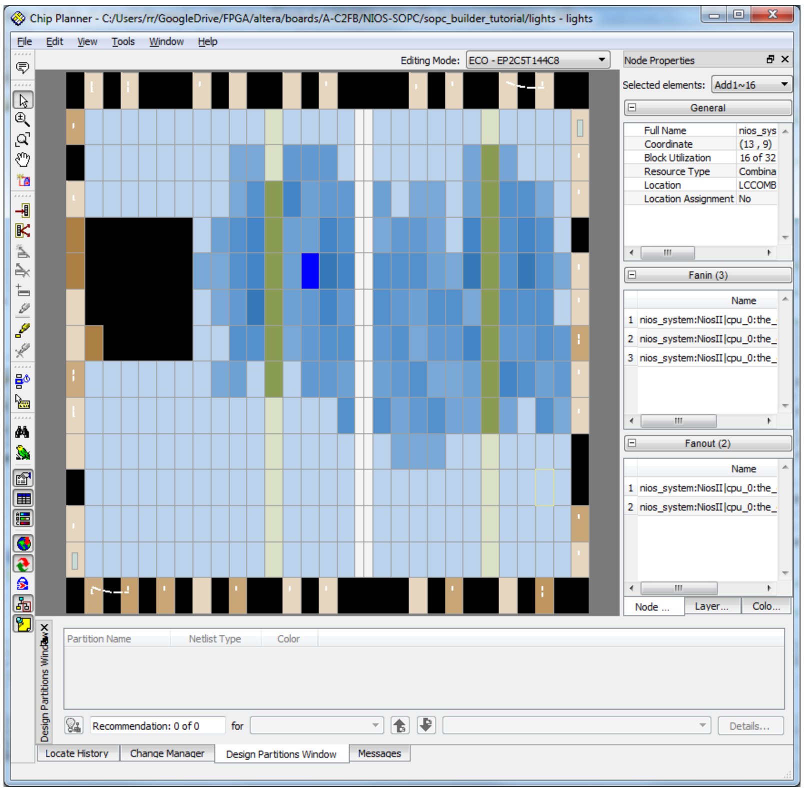


Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
CLOCK_50	Input	PIN_17	1	B1_N0	3.3-V LV...default)	
KEY[0]	Input	PIN_51	4	B4_N1	3.3-V LV...default)	
LEDG[3]	Output	PIN_65	4	B4_N0	3.3-V LV...default)	
LEDG[2]	Output	PIN_69	4	B4_N0	3.3-V LV...default)	
LEDG[1]	Output	PIN_70	4	B4_N0	3.3-V LV...default)	
LEDG[0]	Output	PIN_71	4	B4_N0	3.3-V LV...default)	
SW[3]	Input	PIN_43	4	B4_N1	3.3-V LV...default)	
SW[2]	Input	PIN_48	4	B4_N1	3.3-V LV...default)	
SW[1]	Input	PIN_40	4	B4_N1	3.3-V LV...default)	
SW[0]	Input	PIN_45	4	B4_N1	3.3-V LV...default)	

Figure 17. The actual allocation

The KEY[0] (`Reset_n`) signal can be set to +VCC or VREF so scan through the pins, you are looking for a pin named VREFxxxx (choose any, though PIN_51 is as good a choice as any).

Now the design on the board will take up about a third of the chip area (Figure 18 below). That is interesting to note given we will move onto playing with other soft cores in later experiments.


Chip Planner - C:/Users/rr/GoogleDrive/FPGA/altera/boards/A-C2FB/NIOS-SOPC/sopc_builder_tutorial/lights - lights
File Edit View Tools Window Help
Editing Mode: ECO - EP2C5T144C8
Node Properties
Selected elements: Add1~16
General
Fanin (3)
Fanout (2)
Name
Node ... Layer... Colo...
Partition Name Netlist Type Color
Recommendation: 0 of 0 for
Details...
Design Partitions Window
Locate History Change Manager Design Partitions Window Messages

Full Name	nios_sys
Coordinate	(13 , 9)
Block Utilization	16 of 32
Resource Type	Combina
Location	LCCOMB
Location Assignment	No

Fanin (3):
1 nios_system:NiosII|cpu_0:the_
2 nios_system:NiosII|cpu_0:the_
3 nios_system:NiosII|cpu_0:the_

Fanout (2):
1 nios_system:NiosII|cpu_0:the_
2 nios_system:NiosII|cpu_0:the_

Figure 18. Not so squeezy!

4.2 Programming and Configuration

Bomb board as you normally would. You should then see four LEDs light up (Figure 19 below).

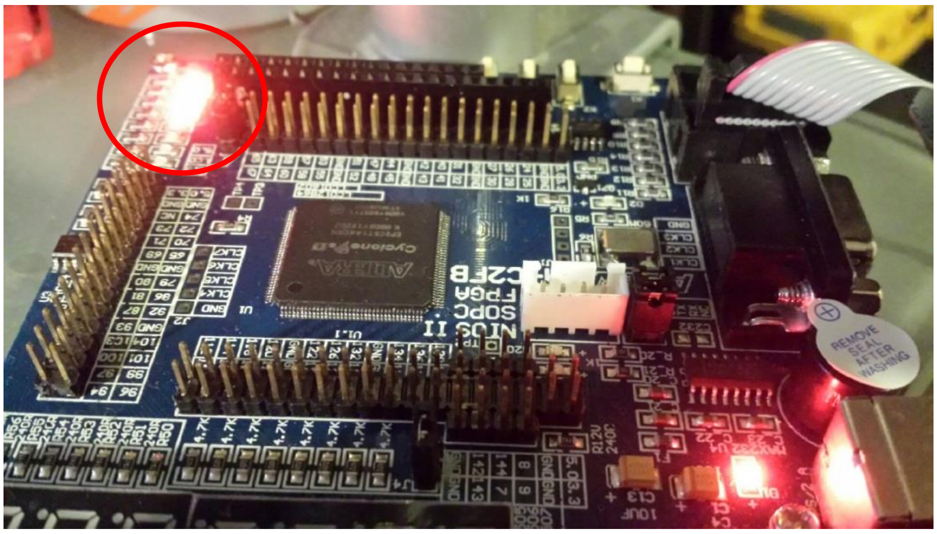


Figure 19. Almost there

Again, if all lights are on and buzzer is buzzing you need to go back and set un-unused pins to "As input tri-stated". Recompile design et cetera.

5 Running the Application Program

5.1 Using a Nios II Assembly Language Program

You will have to get hold of the Altera® Monitor Program from:

http://www.altera.com/education/univ/software/monitor/unv-monitor.html

Ref [5] is handy here.

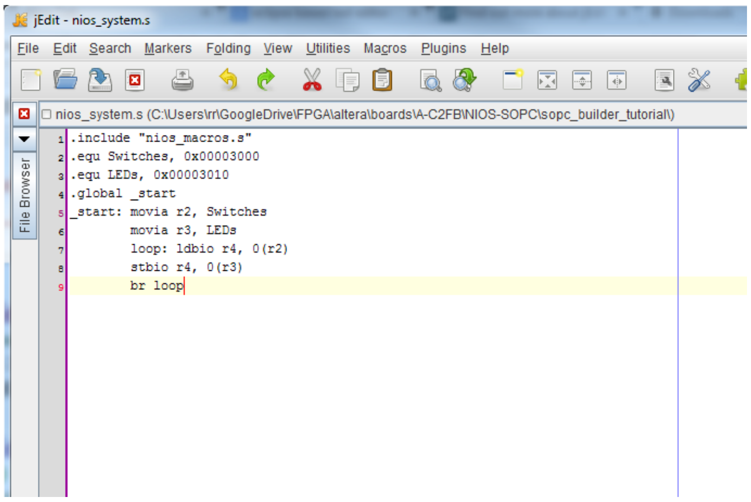
Now don't forget to set the version of Quartus® II to 10.1 on the web page.

If you feel the need you can set the board type to a DE1 BUT the consequences of changing the fields seems to be that the list of companion text material. As long as the version is set to 10.1 we will get a monitor program that matches our other Altera® tools.

Go figure, the installer raises an error but I told it to ignore it and I guess I'll discover what was broken as we go. Grrrrrrrrrr.

It seems that the simulator and another tool is promised in this download, and on the simulator page where the same download file is provide. Don't be surprised, while the downloader promises, only the monitor is really in the download. This may be the cause of the error raised by the installer. Note you will get your Start Menu updated with an empty promise for a "Vector Waveform Editor" and the simulator. The simulator can be downloaded from another link so don't sweat that.

Edit up the assembly file with your favourite programming editor (Figure 20 below) and save as "nios_system.s".

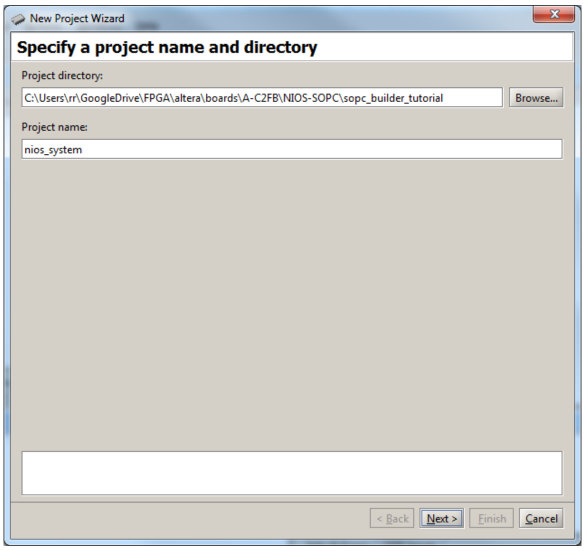

jEdit - nios_system.s
File Edit Search Markers Folding View Utilities Macros Plugins Help
File Browser

nios_system.s (C:\Users\rr\GoogleDrive\FPGA\altera\boards\A-C2FB\NIOS-SOPC\sopc_builder_tutorial\)

```
.include "nios_macros.s"
.equ Switches, 0x00003000
.equ LEDs, 0x00003010
.global _start
_start: movia r2, Switches
        movia r3, LEDs
        loop: ldbio r4, 0(r2)
        stbio r4, 0(r3)
        br loop
```

Figure 20. I am not fussy so I'll use jEdit

Run the monitor and create a new project in our current project directory (Figure 21), I called the project, go figure, "nios_system".

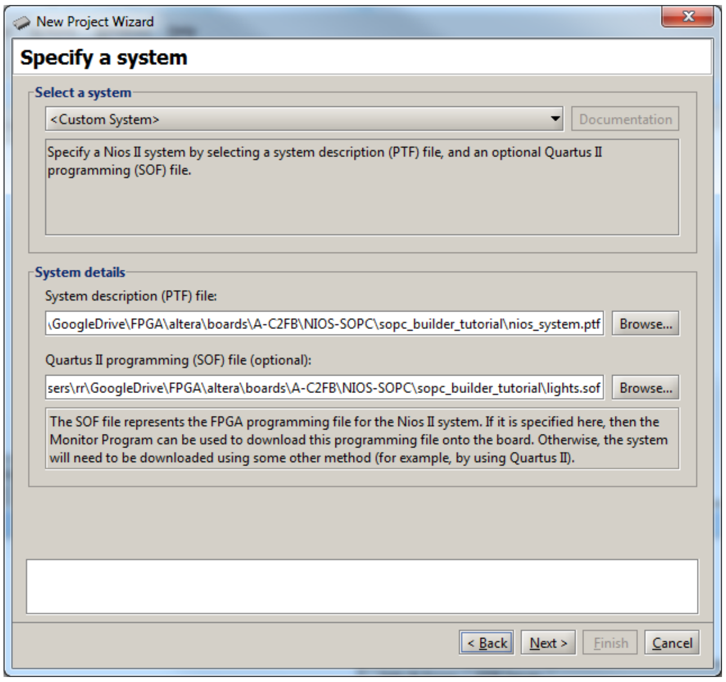


Project directory: C:\Users\rr\GoogleDrive\FPGA\altera\boards\A-C2FB\NIOS-SOPC\sopc_builder_tutorial

Project name: nios_system

Figure 21. Altera Monitor Program "New Project Wizard"

Create a custom project by pointing at the "nios_system.ptf" file create by the SOPC Builder tool that is already in our project directory (Figure 22 below). I pointed to the "lights.sof" file as well … because I could.



Specify a Nios II system by selecting a system description (PTF) file, and an optional Quartus II programming (SOF) file.

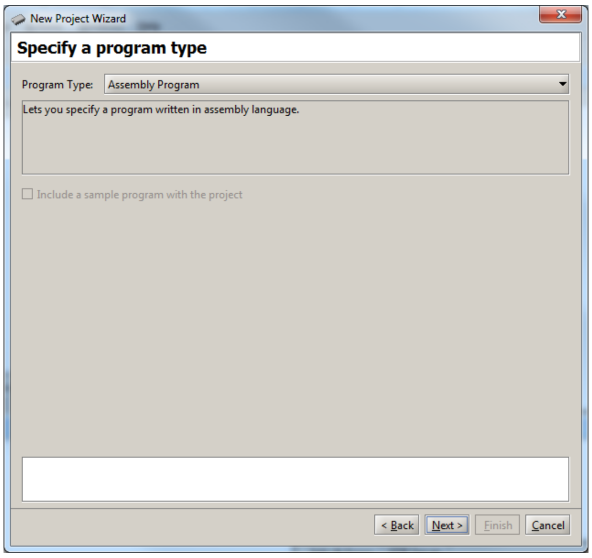
System description (PTF) file: \GoogleDrive\FPGA\altera\boards\A-C2FB\NIOS-SOPC\sopc_builder_tutorial\nios_system.ptf

Quartus II programming (SOF) file (optional): sers\rr\GoogleDrive\FPGA\altera\boards\A-C2FB\NIOS-SOPC\sopc_builder_tutorial\lights.sof

The SOF file represents the FPGA programming file for the Nios II system. If it is specified here, then the Monitor Program can be used to download this programming file onto the board. Otherwise, the system will need to be downloaded using some other method (for example, by using Quartus II).

Figure 22. We're special "Custom" that is.

Specify an "Assembly" program (Figure 23 below) and select "nios_system.s" (which you should have created one page back).

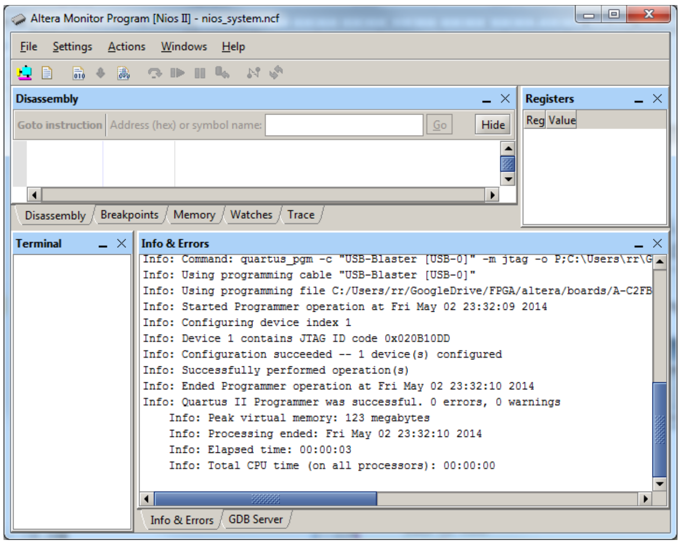


Program Type: Assembly Program

Lets you specify a program written in assembly language.

Figure 23. When was the last time you wrote assembly language?

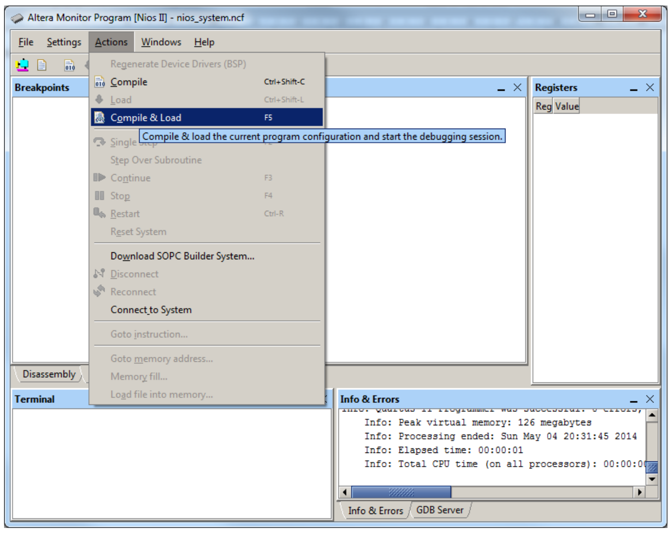
Now following prompts etc., turning on board and downloading the lot and voila! (Figure 24) … well not really, still more to come.


Altera Monitor Program [Nios II] - nios_system.ncf
File Settings Actions Windows Help
Disassembly
Goto instruction Address (hex) or symbol name: Go Hide
Registers
Reg Value
Disassembly Breakpoints Memory Watches Trace
Terminal
Info & Errors
Info & Errors GDB Server

```
Info: Command: quartus_pgm -c "USB-Blaster [USB-0]" -m jtag -o P;C:\Users\rr\G
Info: Using programming cable "USB-Blaster [USB-0]"
Info: Using programming file C:/Users/rr/GoogleDrive/FPGA/altera/boards/A-C2FB
Info: Started Programmer operation at Fri May 02 23:32:09 2014
Info: Configuring device index 1
Info: Device 1 contains JTAG ID code 0x020B10DD
Info: Configuration succeeded -- 1 device(s) configured
Info: Successfully performed operation(s)
Info: Ended Programmer operation at Fri May 02 23:32:10 2014
Info: Quartus II Programmer was successful. 0 errors, 0 warnings
    Info: Peak virtual memory: 123 megabytes
    Info: Processing ended: Fri May 02 23:32:10 2014
    Info: Elapsed time: 00:00:03
    Info: Total CPU time (on all processors): 00:00:00
```

Figure 24. Successful assemble and download of design!

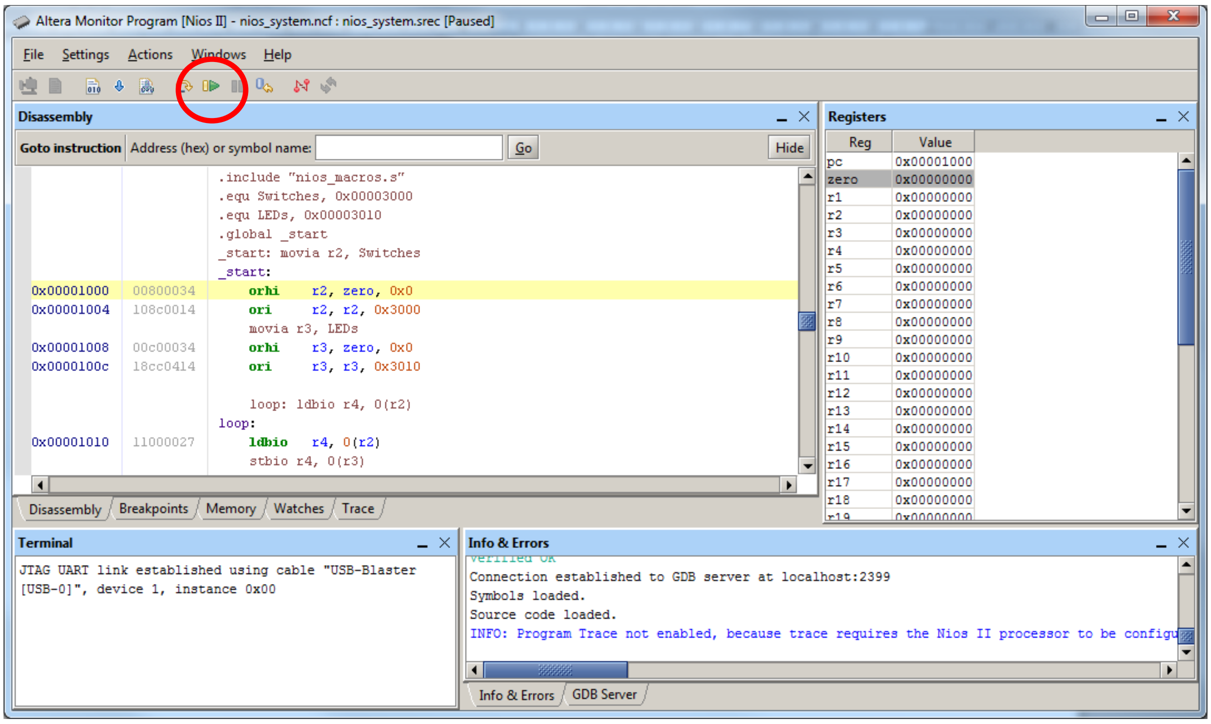
Next thing is to compile and load (Figure 25 below).



```
    Info: Peak virtual memory: 126 megabytes
    Info: Processing ended: Sun May 04 20:31:45 2014
    Info: Elapsed time: 00:00:01
    Info: Total CPU time (on all processors): 00:00:0
```

Figure 25. Load and Compile

Now we are really close (Figure 26). Notice we have a run button! If you have gotten to here then select the green run button and congratulations.


Altera Monitor Program [Nios II] - nios_system.ncf : nios_system.srec [Paused]
File Settings Actions Windows Help
Disassembly
Goto instruction Address (hex) or symbol name: Go Hide
Registers
Disassembly Breakpoints Memory Watches Trace
Terminal
Info & Errors
Info & Errors GDB Server

```
                        .include "nios_macros.s"
                        .equ Switches, 0x00003000
                        .equ LEDs, 0x00003010
                        .global _start
                        _start: movia r2, Switches
                        _start:
0x00001000  00800034        orhi    r2, zero, 0x0
0x00001004  108c0014        ori     r2, r2, 0x3000
                            movia r3, LEDs
0x00001008  00c00034        orhi    r3, zero, 0x0
0x0000100c  18cc0414        ori     r3, r3, 0x3010

                            loop: ldbio r4, 0(r2)
                        loop:
0x00001010  11000027        ldbio   r4, 0(r2)
                            stbio r4, 0(r3)
```

```
Reg   Value
pc    0x00001000
zero  0x00000000
r1    0x00000000
r2    0x00000000
r3    0x00000000
r4    0x00000000
r5    0x00000000
r6    0x00000000
r7    0x00000000
r8    0x00000000
r9    0x00000000
r10   0x00000000
r11   0x00000000
r12   0x00000000
r13   0x00000000
r14   0x00000000
r15   0x00000000
r16   0x00000000
r17   0x00000000
r18   0x00000000
```

```
JTAG UART link established using cable "USB-Blaster
[USB-0]", device 1, instance 0x00
```

```
Verified OK
Connection established to GDB server at localhost:2399
Symbols loaded.
Source code loaded.
INFO: Program Trace not enabled, because trace requires the Nios II processor to be configu
```

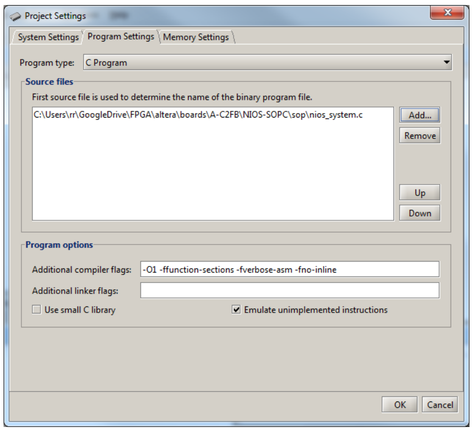
Figure 26. We are there!

What should happen is the LEDs extinguish, but don't fret. You can light them by pressing one through all four buttons.

5.2 Using a C-Language Program

Follow the tutorial steps to use the monitor program to build and install the C program example. Use the same approach as for the Assembly program:

1. Use your favourite editor to write the source file.
2. Select your C source file (Figure 27).



Program type: C Program

Source files

First source file is used to determine the name of the binary program file.

C:\Users\rr\GoogleDrive\FPGA\altera\boards\A-C2FB\NIOS-SOPC\sop\nios_system.c

Program options

Additional compiler flags: -O1 -ffunction-sections -fverbose-asm -fno-inline

Additional linker flags:

☐ Use small C library ☑ Emulate unimplemented instructions

Figure 27. Too easy!

3. Et cetera.

That's it for the Ref [1] tutorial. Now let's stretch ourselves a little.

Using the NIOS II SBT

Here we move away from the original Ref [1] tutorial but still lean on other Altera® tutorial material namely Ref [6].

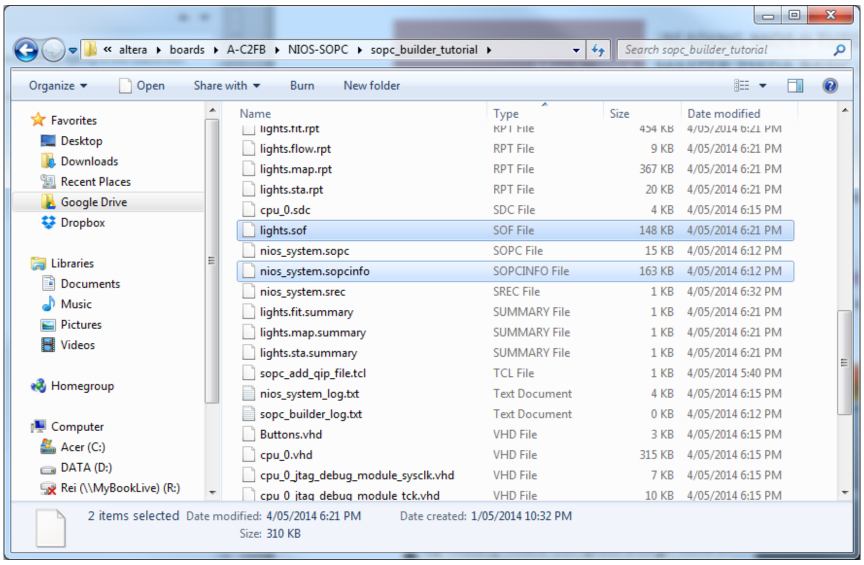
First step here, using Ref [6] My First NIOS II Software Tutorial is to get the "Hello world" application going on our design.

Once we've worked out the "Hello World" example, we can re-work this with the C example to tickle out LEDs from Ref [1].

References in ***red*** are now to Ref [6] - no longer the original Ref [1] – that is, open up Ref [6]. We will work through Ref [6] but with the tweaks below, as needed, for our board. ***Headings and other references in red*** now refer out to those in Ref [6]. You might like to have Ref [7] handing too.

Software and Hardware Requirements

Figure 1-1. of Ref [6], for us, will look like Figure 28 below. We will be picking up "lights.sof" and "nios_system.sopcinfo" in the wizards to follow.



Name	Type	Size	Date modified
lights.fit.rpt	RPT File	454 KB	4/05/2014 6:21 PM
lights.flow.rpt	RPT File	9 KB	4/05/2014 6:21 PM
lights.map.rpt	RPT File	367 KB	4/05/2014 6:21 PM
lights.sta.rpt	RPT File	20 KB	4/05/2014 6:21 PM
cpu_0.sdc	SDC File	4 KB	4/05/2014 6:15 PM
lights.sof	SOF File	148 KB	4/05/2014 6:21 PM
nios_system.sopc	SOPC File	15 KB	4/05/2014 6:12 PM
nios_system.sopcinfo	SOPCINFO File	163 KB	4/05/2014 6:12 PM
nios_system.srec	SREC File	1 KB	4/05/2014 6:32 PM
lights.fit.summary	SUMMARY File	1 KB	4/05/2014 6:21 PM
lights.map.summary	SUMMARY File	1 KB	4/05/2014 6:21 PM
lights.sta.summary	SUMMARY File	1 KB	4/05/2014 6:21 PM
sopc_add_qip_file.tcl	TCL File	1 KB	4/05/2014 5:40 PM
nios_system_log.txt	Text Document	4 KB	4/05/2014 6:15 PM
sopc_builder_log.txt	Text Document	0 KB	4/05/2014 6:12 PM
Buttons.vhd	VHD File	3 KB	4/05/2014 6:15 PM
cpu_0.vhd	VHD File	315 KB	4/05/2014 6:15 PM
cpu_0_jtag_debug_module_sysclk.vhd	VHD File	7 KB	4/05/2014 6:15 PM
cpu_0_jtag_debug_module_tck.vhd	VHD File	10 KB	4/05/2014 6:15 PM

2 items selected Date modified: 4/05/2014 6:21 PM Date created: 1/05/2014 10:32 PM
Size: 310 KB

Figure 28. Our project folder

Download Hardware Design to Target FPGA

Step 1. Step 2. of Ref [6] as is.

Step 3. for us should actually read:

3. On Windows computers, hover mouse over **All Programs** > **Altera** > **Nios II EDS 10.1** > **Nios II 10.1 Software Build Tools for Eclipse** in the Windows Start menu. Right click mouse to bring up menu and go to Properties (Figure 29 below) then select the "Advanced …" button and set the "Advanced Properties" to "Run as administrator" (Figure 30 below). Then choose **All Programs** > **Altera** > **Nios II EDS 10.1** > **Nios II 10.1 Software Build Tools for Eclipse** in the Windows Start menu to run the program.

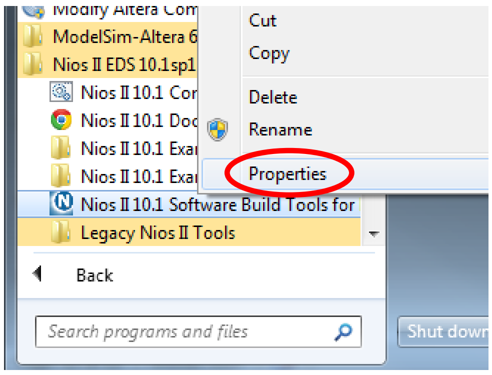
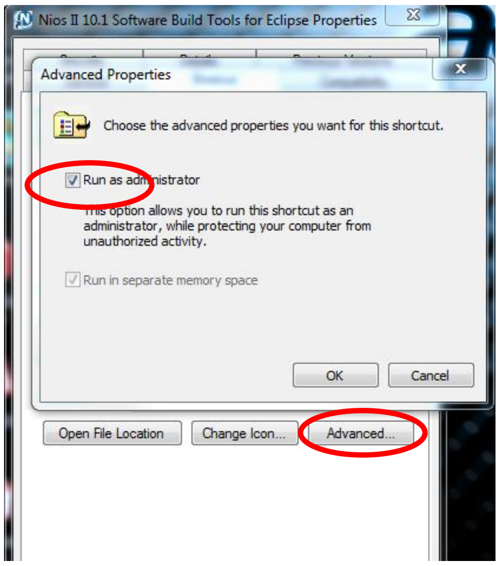


Figure 29. We need to change the properties on the shortcut



Nios II 10.1 Software Build Tools for Eclipse Properties

Advanced Properties

Choose the advanced properties you want for this shortcut.

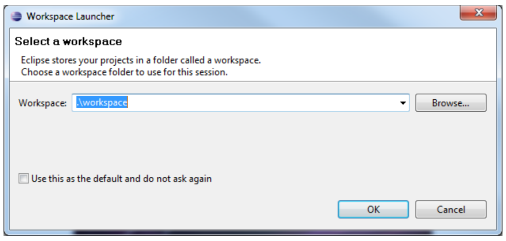
[✓] Run as administrator

This option allows you to run this shortcut as an administrator, while protecting your computer from unauthorized activity.

[✓] Run in separate memory space

Figure 30. Nios II SBT need to have admin rights when running

Step 4. I have so many different Eclipse tools I keep forgetting where the workspaces hide so I always start with the workspace under the Eclipse directory using ".\workspace"[2] (Figure 31 below).


Workspace Launcher

Select a workspace

Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: .\workspace

Figure 31. Find "workspace" under the running Eclipse directory

With eclipse running, ***Step 5.*** takes us to the NIOS II menu shown in Figure 32 below.

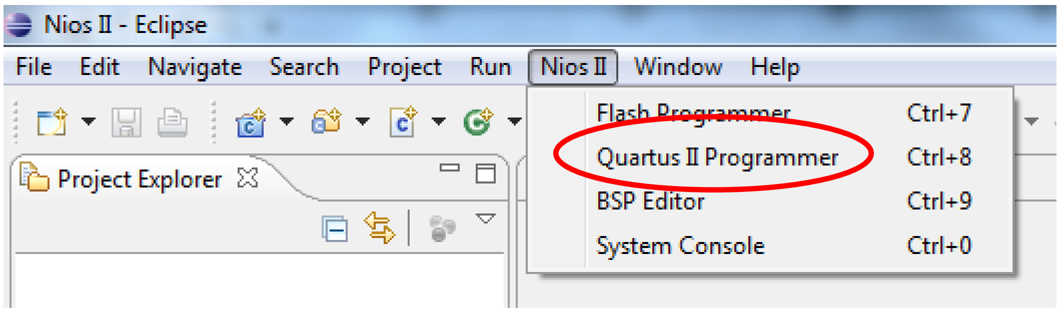


Figure 32. The Altera ® plugin menu

Go figure, without a project you get an empty list (Figure 33) just hit "Cancel" and then "Yes" on the next dialog that opens (Figure 34).

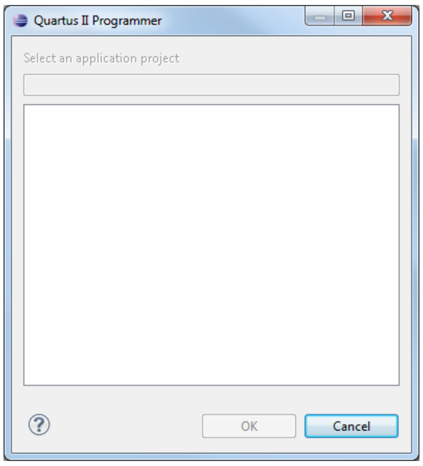
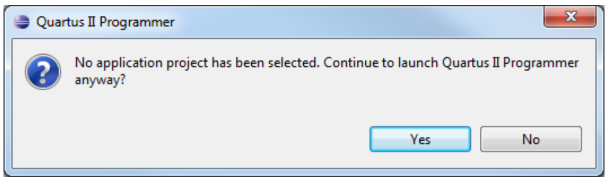


Figure 33. Makes sense, I suppose, no project so nothing in the list

[2] This trick is from the good old DOS command line days as ".\" literally means "in the current directory.


Quartus II Programmer

No application project has been selected. Continue to launch Quartus II Programmer anyway?

Figure 34. Yes, anyway!

You should get something like the following in the Eclipse console view:

```
Executing: C:/altera/10.1sp1/quartus/bin/quartus_pgmw (C:\altera\10.1sp1\nios2eds)
```

Then, after a minute of disk grinding the familiar Quartus II Programmer window of ***Figure 1-3.***

You should be fall through each of ***Step 6.*** to ***Step 15.*** with ease and have the FPGA bombed ready for software.

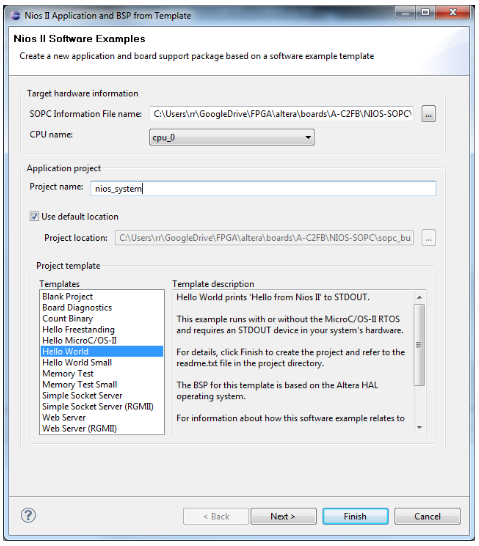
Nios II SBT for Eclipse Build Flow

Create the Hello World Example Project

Step 1. and Step 3. as is.

Step 3. will take a little time to run so don't sweat it.

Step 4. Call your Eclipse Project "nios_system" (Figure 35).


Nios II Application and BSP from Template

Nios II Software Examples

Create a new application and board support package based on a software example template

Target hardware information

SOPC Information File name: C:\Users\rr\GoogleDrive\FPGA\altera\boards\A-C2FB\NIOS-SOPC\

CPU name: cpu_0

Application project

Project name: nios_system

[✓] Use default location

Project location: C:\Users\rr\GoogleDrive\FPGA\altera\boards\A-C2FB\NIOS-SOPC\sopc_bu

Project template

Templates: Blank Project, Board Diagnostics, Count Binary, Hello Freestanding, Hello MicroC/OS-II, [✓ Hello World], Hello World Small, Memory Test, Memory Test Small, Simple Socket Server, Simple Socket Server (RGMII), Web Server, Web Server (RGMII)

Template description

Hello World prints 'Hello from Nios II' to STDOUT.

This example runs with or without the MicroC/OS-II RTOS and requires an STDOUT device in your system's hardware.

For details, click Finish to create the project and refer to the readme.txt file in the project directory.

The BSP for this template is based on the Altera HAL operating system.

For information about how this software example relates to

Figure 35. Our system name is "nios_system"

Step 5. as is.

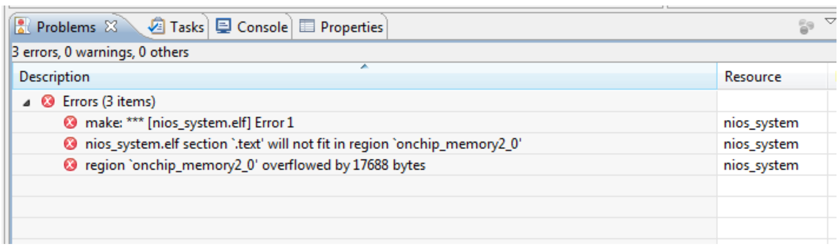
Step 6. will take a little time to run so don't sweat it.

Step 7. of course our project name is "nios_system".

Build and Run the Program

Step 1. And Step 2. as is.

You should get an error, sorry. If we use the "hello_world" example with our design it turns out is too big to fit on into our hardware design (Figure 36).



Problems | Tasks | Console | Properties

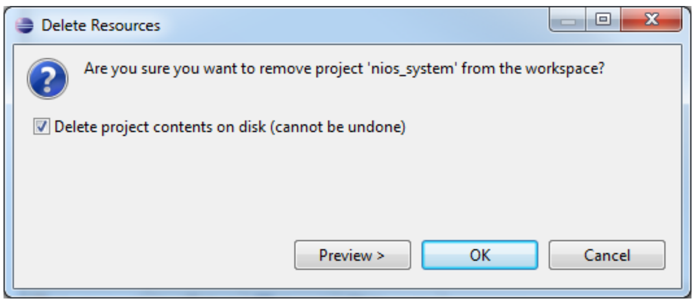
3 errors, 0 warnings, 0 others

Description	Resource
Errors (3 items)	
make: *** [nios_system.elf] Error 1	nios_system
nios_system.elf section `.text' will not fit in region `onchip_memory2_0'	nios_system
region `onchip_memory2_0' overflowed by 17688 bytes	nios_system

Figure 36. Drats!

This is important to note, of course, but I left it in to make a point – that is, the memory allocation in the design needs to be planned to "fit" any software design added. We are NOT worrying about this in our bumbling through our first time here – but keep it in mind for your other projects.

So, delete the "nios_system" and "nios_system_bsp" projects. Make sure you select the delete from disk as well (Figure 37).



Delete Resources

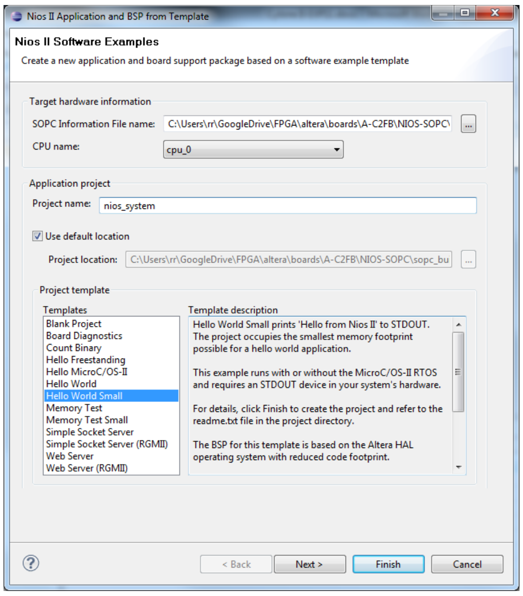
Are you sure you want to remove project 'nios_system' from the workspace?

[✓] Delete project contents on disk (cannot be undone)

Figure 37.

There may be some calamity here with deletion of parts of the projects failing with an exception which has to do with permissions. It won't happen all the time but will occasionally. Deleting "nios_system_bsp" first seems to help. There may be some task still running in the IDE that holds access to the files. If there are problems then exit IDE, delete directories with File Explorer, and restart IDE. If problems persist it is likely best to change workspaces to get to a clean directory.

Go back to "Create the Hello World Example Project" and we'll substitute "Hello World Small" for "Hello World" (Figure 38).



Nios II Application and BSP from Template

Nios II Software Examples

Create a new application and board support package based on a software example template

Target hardware information

SOPC Information File name: C:\Users\rr\GoogleDrive\FPGA\altera\boards\A-C2FB\NIOS-SOPC\

CPU name: cpu_0

Application project

Project name: nios_system

☑ Use default location

Project location: C:\Users\rr\GoogleDrive\FPGA\altera\boards\A-C2FB\NIOS-SOPC\sopc_bu

Project template

Templates:
- Blank Project
- Board Diagnostics
- Count Binary
- Hello Freestanding
- Hello MicroC/OS-II
- Hello World
- Hello World Small
- Memory Test
- Memory Test Small
- Simple Socket Server
- Simple Socket Server (RGMII)
- Web Server
- Web Server (RGMII)

Template description:

Hello World Small prints 'Hello from Nios II' to STDOUT. The project occupies the smallest memory footprint possible for a hello world application.

This example runs with or without the MicroC/OS-II RTOS and requires an STDOUT device in your system's hardware.

For details, click Finish to create the project and refer to the readme.txt file in the project directory.

The BSP for this template is based on the Altera HAL operating system with reduced code footprint.

Figure 38. Smaller version!

Now I know using this "smaller" project worked <u>once</u>. The elf file "nios_sytem.elf" turned up in the Eclipse "Project Explorer" – which is why I suggested the approach. Go figure, trying to repeat the success with the same steps fails on occasion. It seems like either the Eclipse tools has a glitch, or at least junk is kept somehow in the config files for the workspace - so going to a clean workspace is also an option between attempts.

Other problem occurred where on one attempt, while two projects "nios_sytem" and "nios_system_bsp" were created, for some reason the "nios_system" project did not turn up in the build system??? This was mirrored in the menu selection with right mouse click over the "nios_system" project when you went to select "Run as" – you are supposed to see menu items as in Figure 39 when you right click on "nios_system" project but I didn't see that for the broken project.

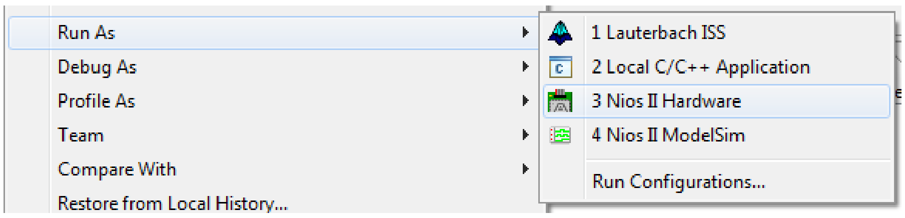


Figure 39. Nios II Hardware run configuration

So, I went to a backup directory and it seems to work. Well, almost. The "**Run As** > **3 Nios II Hardware**" menu item opened a "Run Configuration" window which did not draw completely. In fact, in subsequent tries it turned out I could almost expect to "throw out" the first activation of the "**Run As** > **3 Nios II Hardware**" run configuration window.

If you close the first activation of the dialog and then re-open it, it seems to come good. This is still a pain though.

Safest bet appears to be:

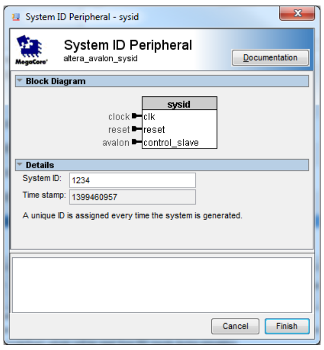
1. Start with clean project directory.
2. Build SPOC cleanly from scratch and bomb board.
3. Jump in with Eclipse and build project in fresh workspace (we'll have to find the old workspaces and delete those to keep things tidy).
4. Create a new templated project from scratch.
5. Run a full clean/build and you get the elf file built.

Next problem was when the "**Run as** > **Run As** > **3 Nios II Hardware**" would open the "Run Configuration" dialog and ask for a "Base Address", a "System ID" and a "Time Stamp"!!??

Turns out there is a special component that has to be installed into the SOPC design (Figure 40 over the page) – that was not in the Ref [1] tutorial design. This was not an error in the original tutorial as it appears not to be required when compiling from the monitor - but is required when using the Nios II SBT.

I broke back into the hardware design and added a "System ID Peripheral", giving the SPOC model the "System ID" of "1234". You can too, or give it any ID you like.

You will not have to remember the ID or timestamp because, if you have this component in your design, the "**Run As** > **3 Nios II Hardware**" will fall straight through to bombing and running board – as you will see in a few moments.

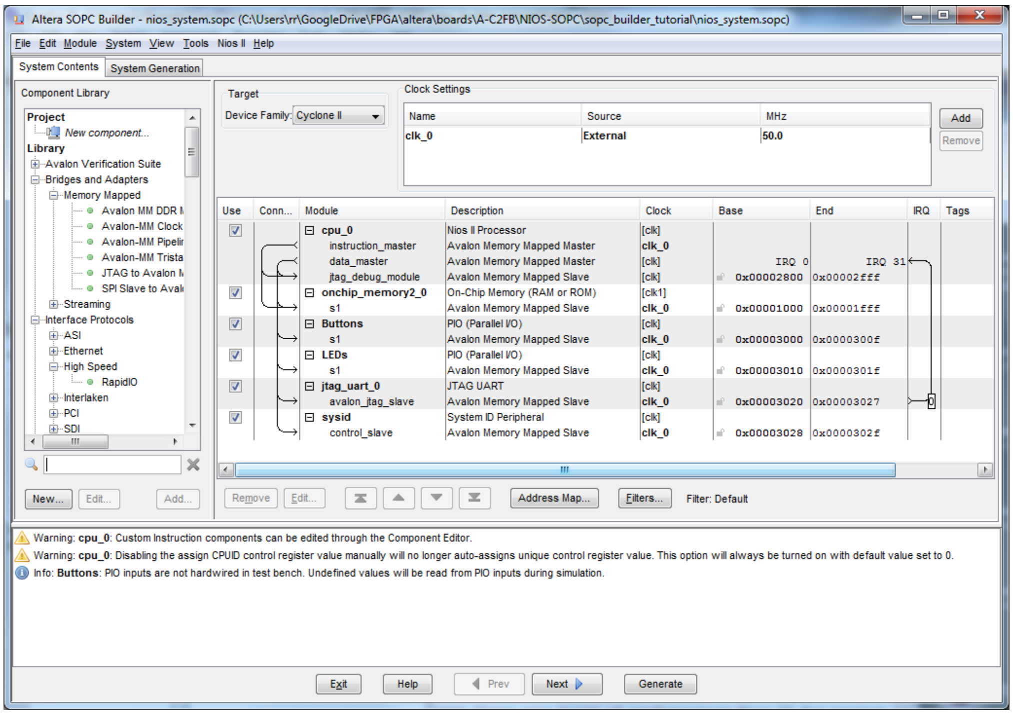


System ID: 1234
Time stamp: 1399460957
A unique ID is assigned every time the system is generated.

Figure 40. When using the Nios II SBT it appears we need one of these "sysid" components

Now, when you add the "sysid" component it will be given the name "sysid_0", you have to rename it "sysid" by convention for things to work downstream – the tool will warn you to do this anyway.

Also, since it has a Base address we want to run "**System** > **Auto-Assign Base Addresses**" and "**System** > **Auto-Assign IRQs**" to set up addresses (Figure 41). The sysid component base address comes to haunt us when using the Nios II SBT otherwise.



Altera SOPC Builder - nios_system.sopc (C:\Users\rr\GoogleDrive\FPGA\altera\boards\A-C2FB\NIOS-SOPC\sopc_builder_tutorial\nios_system.sopc)

Target — Device Family: Cyclone II

Clock Settings

Name	Source	MHz
clk_0	External	50.0

Use	Module	Description	Clock	Base	End	IRQ
✓	cpu_0	Nios II Processor	[clk]			
	instruction_master	Avalon Memory Mapped Master	clk_0			
	data_master	Avalon Memory Mapped Master	[clk]	IRQ 0	IRQ 31	
	jtag_debug_module	Avalon Memory Mapped Slave	[clk]	0x00002800	0x00002fff	
✓	onchip_memory2_0	On-Chip Memory (RAM or ROM)	[clk1]			
	s1	Avalon Memory Mapped Slave	clk_0	0x00001000	0x00001fff	
✓	Buttons	PIO (Parallel I/O)	[clk]			
	s1	Avalon Memory Mapped Slave	clk_0	0x00003000	0x0000300f	
✓	LEDs	PIO (Parallel I/O)	[clk]			
	s1	Avalon Memory Mapped Slave	clk_0	0x00003010	0x0000301f	
✓	jtag_uart_0	JTAG UART	[clk]			
	avalon_jtag_slave	Avalon Memory Mapped Slave	clk_0	0x00003020	0x00003027	0
✓	sysid	System ID Peripheral	[clk]			
	control_slave	Avalon Memory Mapped Slave	clk_0	0x00003028	0x0000302f	

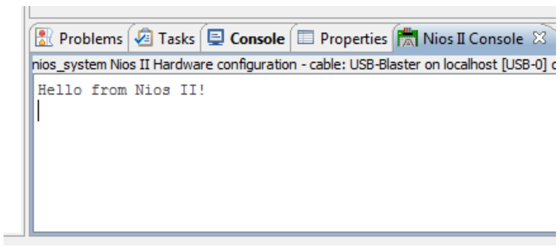
Filter: Default

Warning: **cpu_0**: Custom Instruction components can be edited through the Component Editor.
Warning: **cpu_0**: Disabling the assign CPUID control register value manually will no longer auto-assigns unique control register value. This option will always be turned on with default value set to 0.
Info: **Buttons**: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.

Figure 41. Pesky additional component on top of those from Ref [1] Tutorial

So, yes you will have to go back, change the SOPC design, recompile the VHDL, re-bomb the board, and recompile the software – but this will stick in our heads now. This is also likely your typical design cycle anyway so suck it up Princess!

Now, with the all of that down, take a deep breath and select "**Run as** > **3 Nios II Hardware**" should give you some connection activity on the console and then voila!! (Figure 42) the console clears then the board says hi.



nios_system Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0]

```
Hello from Nios II!
```

Figure 42. How painful was it to get to here!?

Note also all our LEDs are still all lit as there is no code dealing with them yet, so let's add it from our original example in Ref [1].

Here we are going back to the Ref [1] tutorial goal of connection our Buttons to our LEDs.

So, go to the Eclipse IDE, expand the "nios_system" project, and find and change your "hello_world_small.c" source file to read (new bits highlighted):

```
#include "sys/alt_stdio.h"

#define Switches (volatile char *) 0x0003000
#define LEDs (char *) 0x0003010

int main()
{

  alt_putstr("Hello from Nios II!\n");

  /* Event loop never exits. */
  while (1)*LEDs = *Switches;

  return 0;
}
```

Do an clean build and then right click on "nios_system" project then "**Run as** > **Run As** > **3 Nios II Hardware**" and you should still get the "hello world" message BUT your LEDs should also now extinguish BUT your buttons can be pressed to light them as per the original tutorial i.

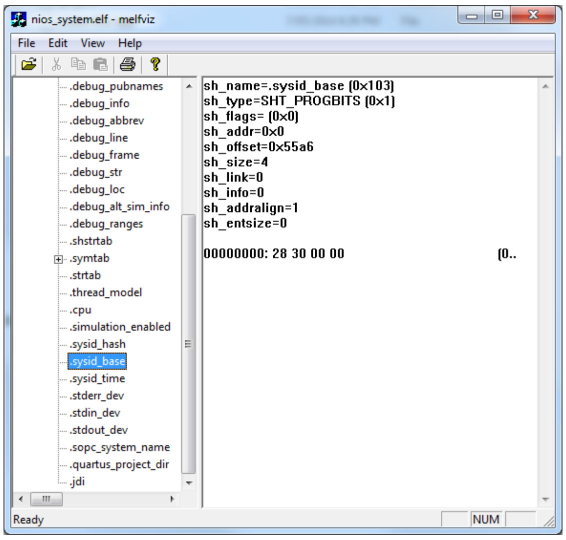
Let's dawdle a while with ELF before we finish

Let's take a moment and open the elf file.

Install ElfViz (http://sourceforge.net/projects/elfviz/)

Have http://www.skyfree.org/linux/references/ELF_Format.pdf handy

ElfViz (or more correctly MelfViz) will allow us to open the ELF file "nios_system.elf" to inspect its contents. If you open up "nios_system.elf" and look for "sysid" you will find things like "sysid_base" which we can relate easily to our base address (0x00003028) in Figure 41 on the preceding page with the value (00000000: 28 30 00 00) below (Figure 43).

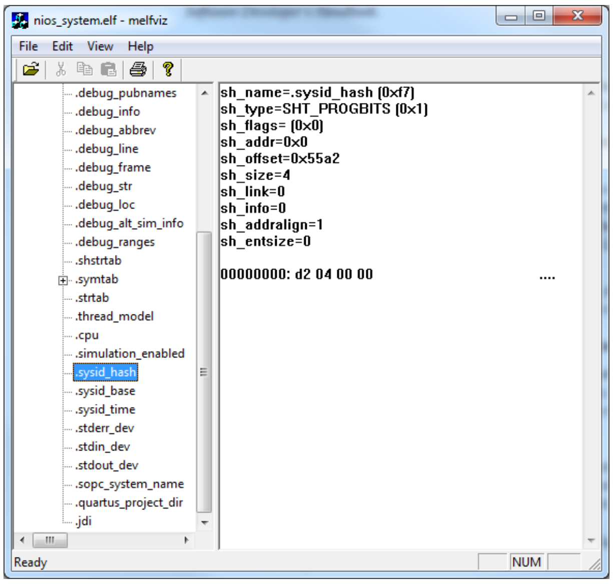


```
sh_name=.sysid_base (0x103)
sh_type=SHT_PROGBITS (0x1)
sh_flags= (0x0)
sh_addr=0x0
sh_offset=0x55a6
sh_size=4
sh_link=0
sh_info=0
sh_addralign=1
sh_entsize=0

00000000: 28 30 00 00                          (0..
```

Figure 43. sysid_base

The "sysid_hash" value (Figure 44 showing 00000000: d2 04 00 00) is mysteriously un-hashed decimal "1234" (from Figure 40).



```
sh_name=.sysid_hash (0xf7)
sh_type=SHT_PROGBITS (0x1)
sh_flags= (0x0)
sh_addr=0x0
sh_offset=0x55a2
sh_size=4
sh_link=0
sh_info=0
sh_addralign=1
sh_entsize=0

00000000: d2 04 00 00                          ....
```

Figure 44. sysid_hash

So, now you have a tool to examine your ELF files if need be. Click on the ".symtab" entry to find all your code symbols.

You may now SCREAM!!