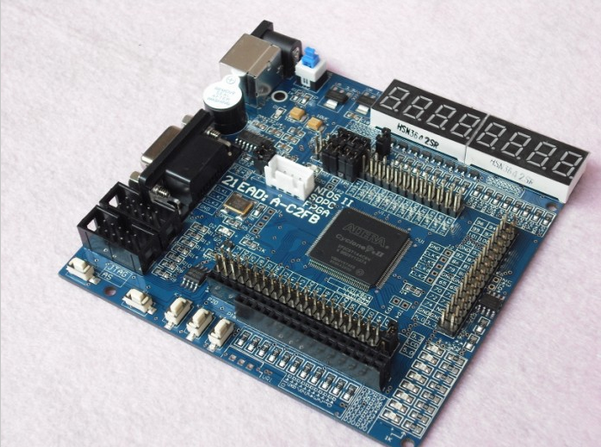
NIOS II – Experiment 1

# 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 Jump out to the interwebby thing

PDF Resources: Designing the Hardware

[1] [Altera® Introduction to the Altera SOPC Builder Using VHDL Designs](ftp://ftp.altera.com/up/pub/Altera_Material/10.1/Tutorials/VHDL/Introduction_to_the_Altera_SOPC_Builder.pdf)

[2] [Altera® SOPC Builder User Guide](http://www.altera.com/literature/ug/ug_sopc_builder.pdf)

[3] [NIOS II Hardware Development Tutorial](http://www.altera.com/literature/tt/tt_nios2_hardware_tutorial.pdf)

[4] [Translated manual for Master 21EDA board](http://organicmonkeymotion.files.wordpress.com/2014/01/altera-fpga-development-board-user-manual.pdf)

PDF Resources: Writing and compiling the Software

[5] [Altera Monitor Program manual](ftp://ftp.altera.com/up/pub/Altera_Material/10.1/Tutorials/Altera_Monitor_Program.pdf)

[6] [My First NIOS II Software Tutorial](http://www.altera.com/literature/tt/tt_my_first_nios_sw.pdf)

[7] [NIOS Software Build Tools](http://www.altera.com/literature/hb/nios2/n2sw_nii52015.pdf)

## 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.

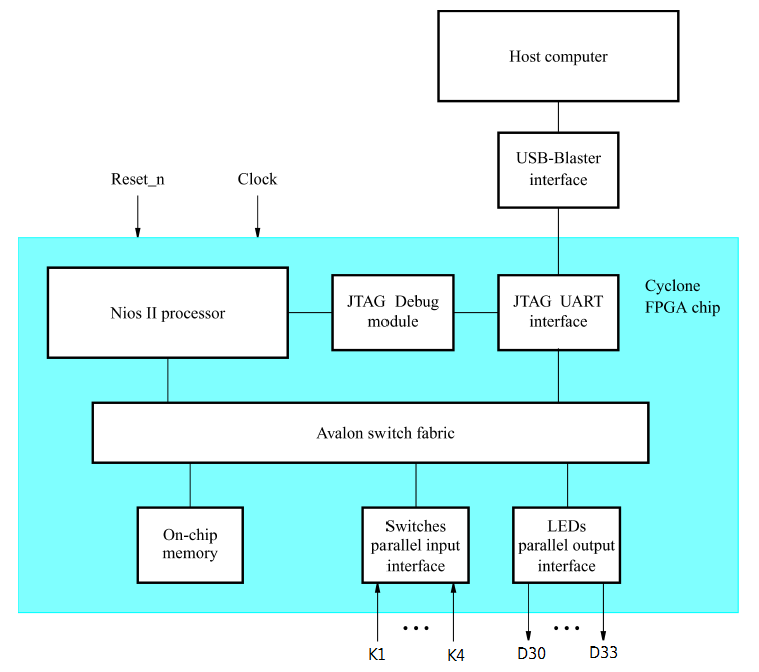


Figure 1. Our simple example Nios II system

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]](#footnote-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!

## 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).

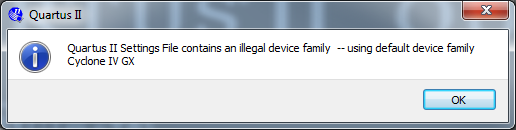


Figure 2. Whoops!

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).

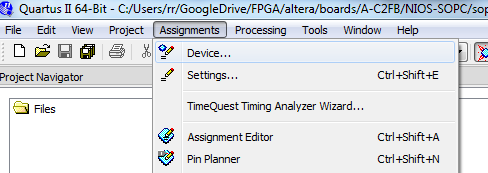


Figure 3. An important missing step.

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.

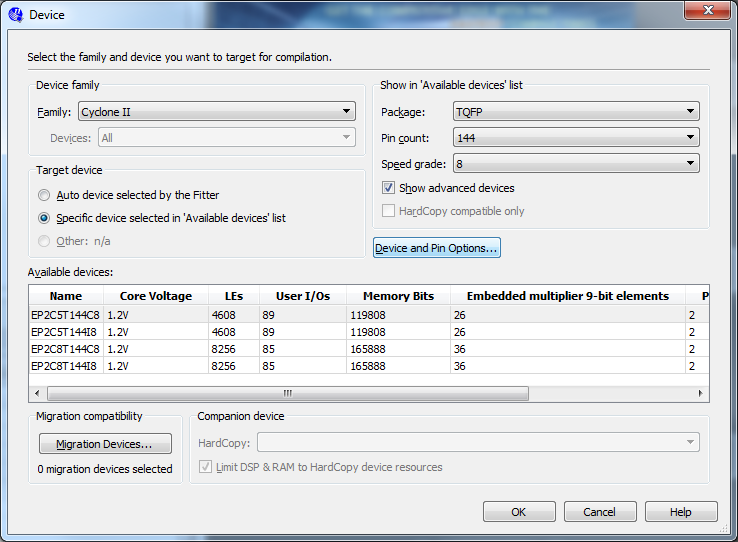


Figure 4. We need to do something with our unused pins!

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

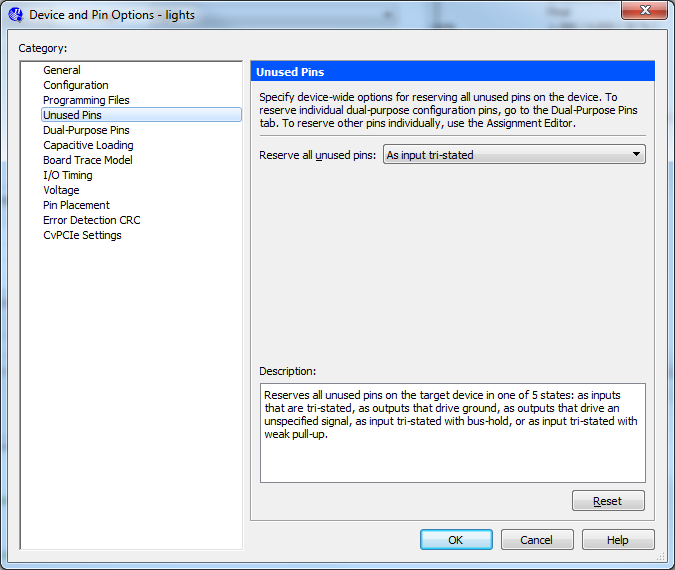


Figure 5. Tell the pins to be “quiet”

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.

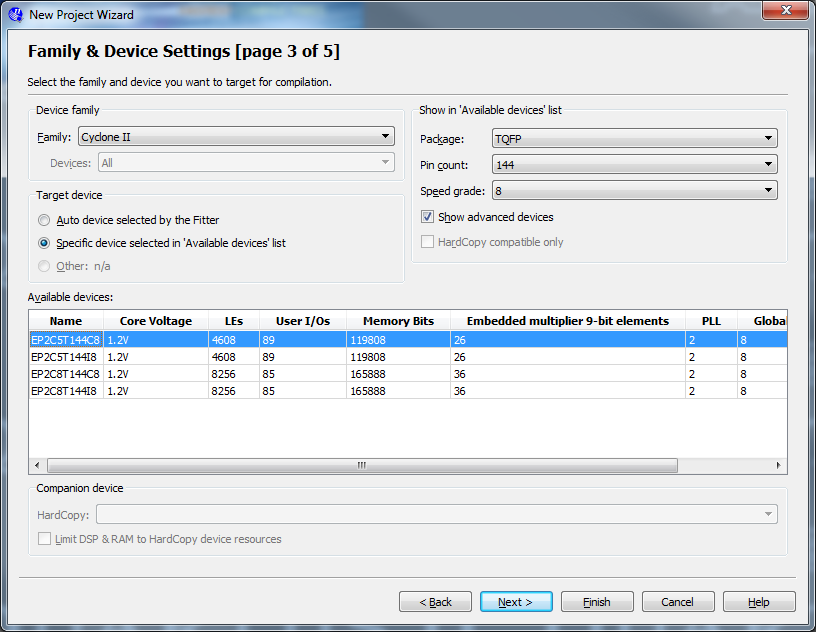
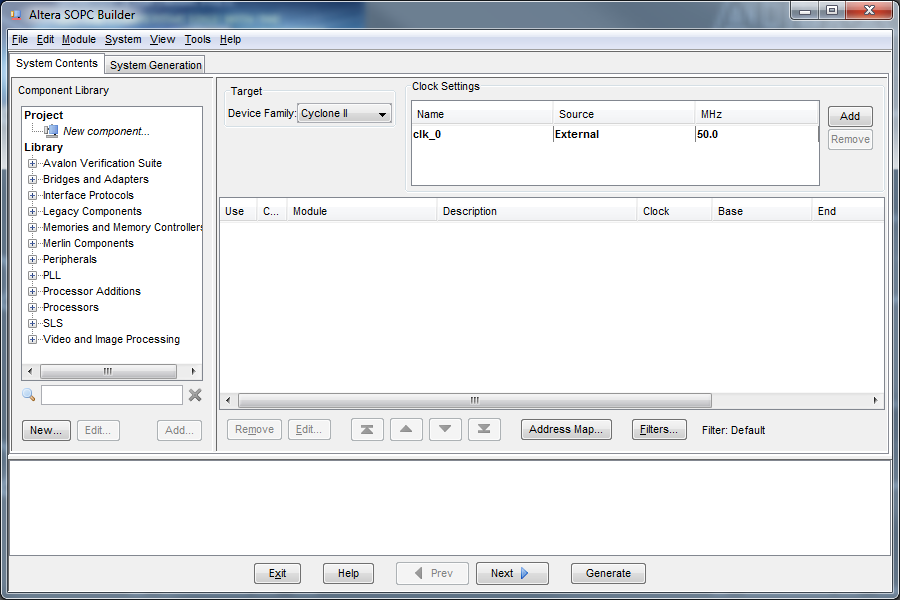


Figure 6. Our chip

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).



**2**

**1**

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).

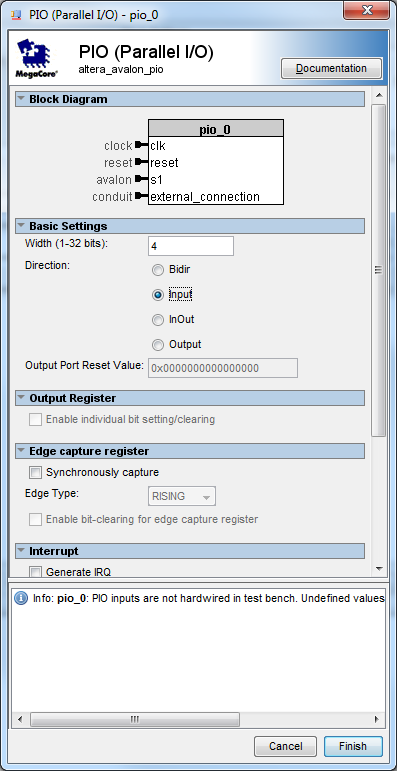


Figure 8. Our replacement for Altera’s tutorial’s Figure 10.

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).

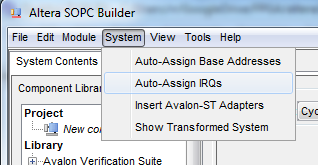


Figure 9. Missing setup step.

So, we end up with Figure 10 below.

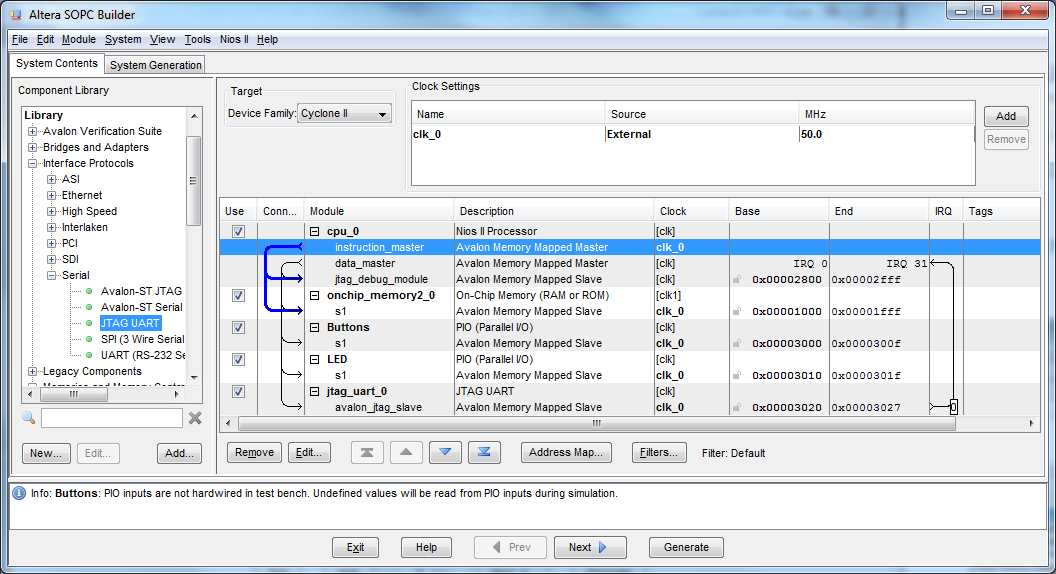


Figure . Almost there

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!

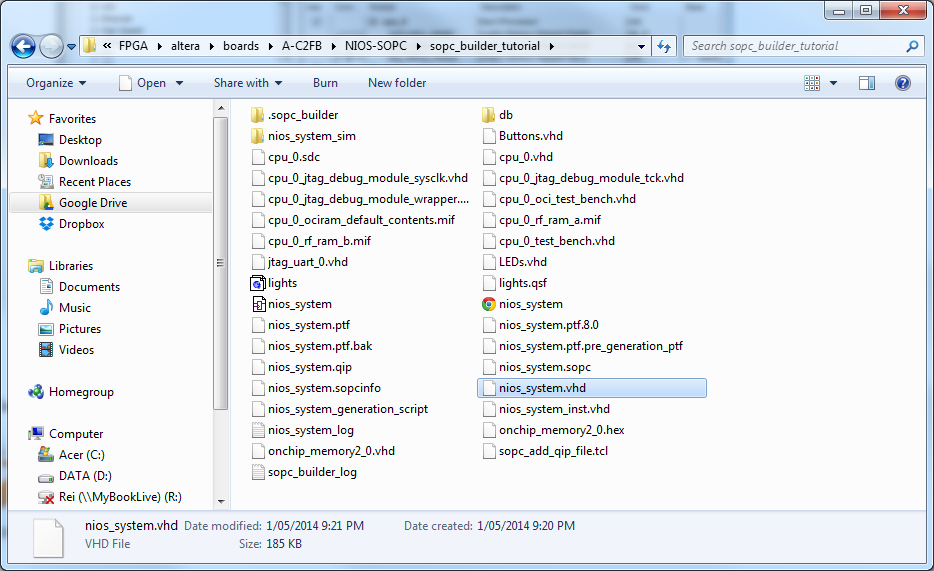


Figure 11. That's better, a named system file!

## 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.

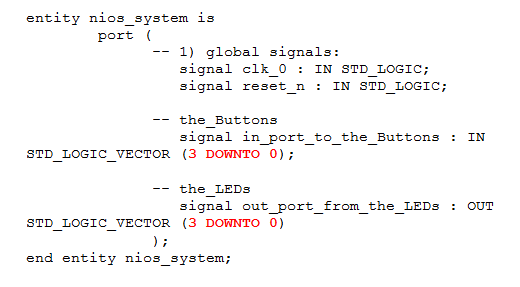


Figure 12. 4 not 8 bits of in/out port

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).

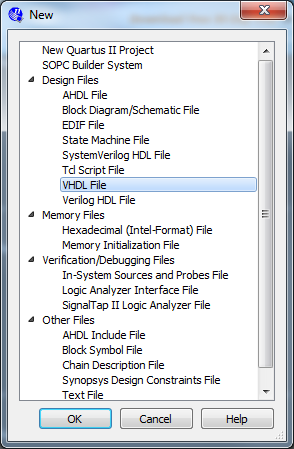


Figure 13. Create "lights.vhd"

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).

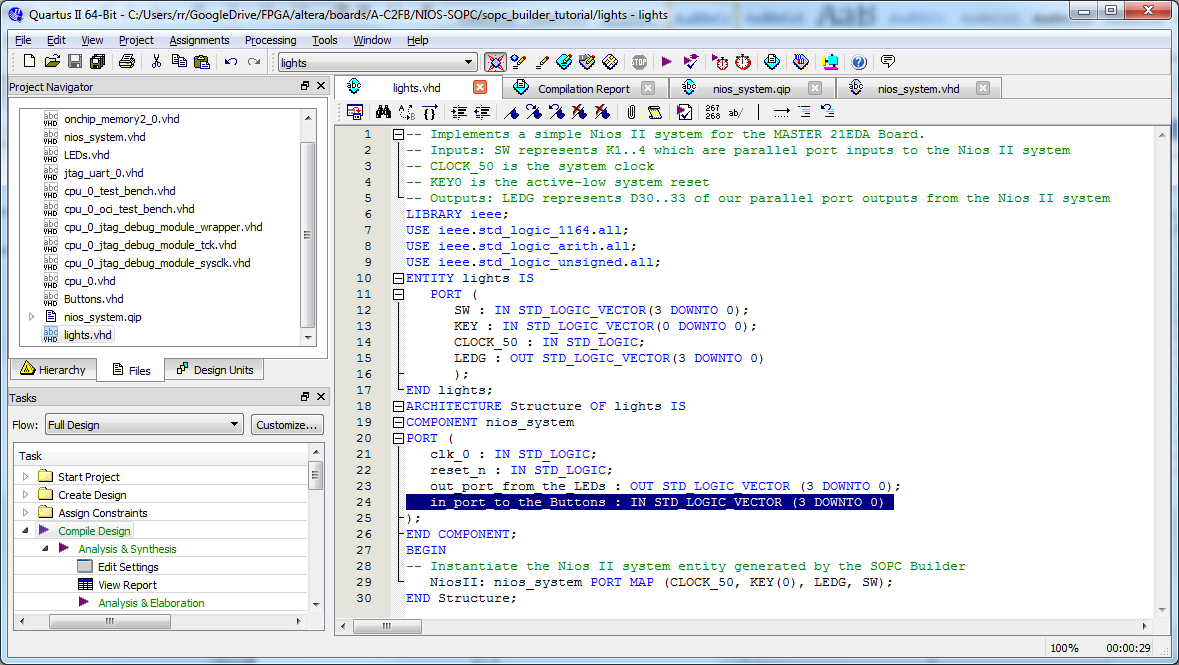


Figure 14. MASTER 21EDA version of "lights.vhd"

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).

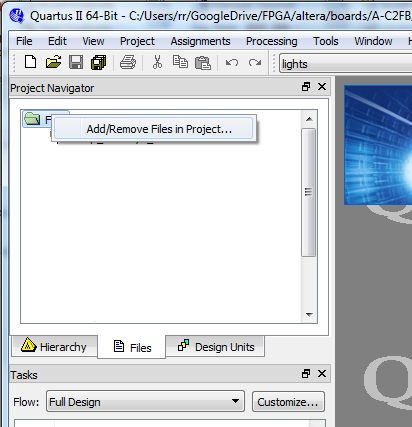
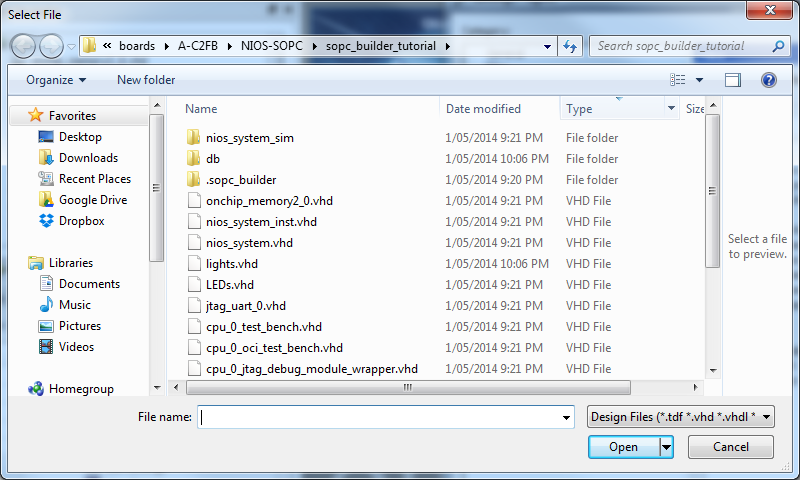
 

Figure 15. Sort them first

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.

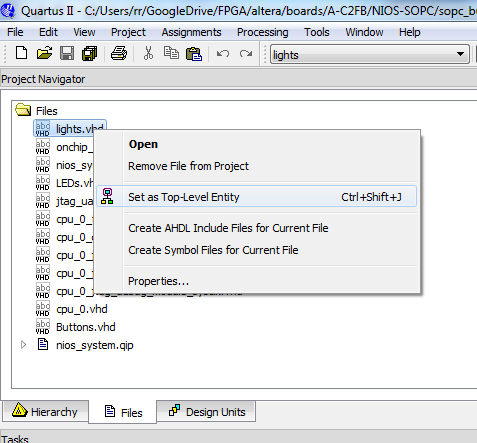


Figure 16. Top Kat!

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).

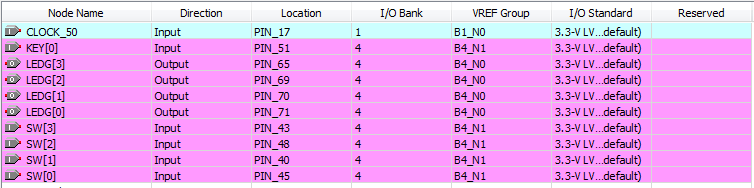


Figure 17. The actual allocation

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.

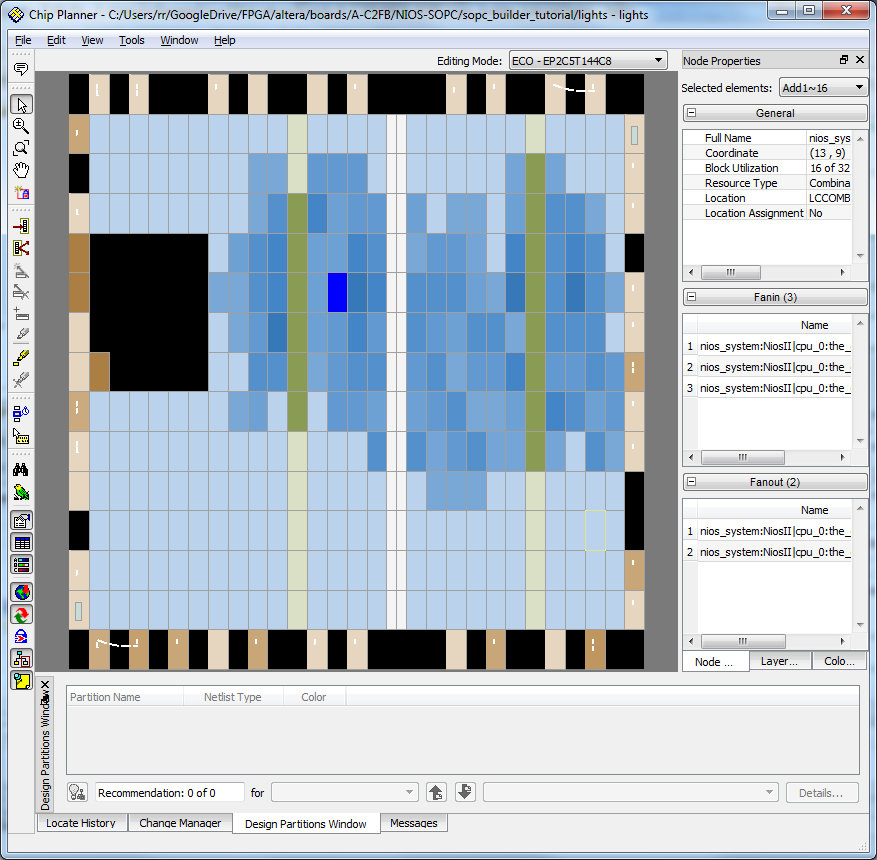


Figure 18. Not so squeezy!

### 4.2 Programming and Configuration

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

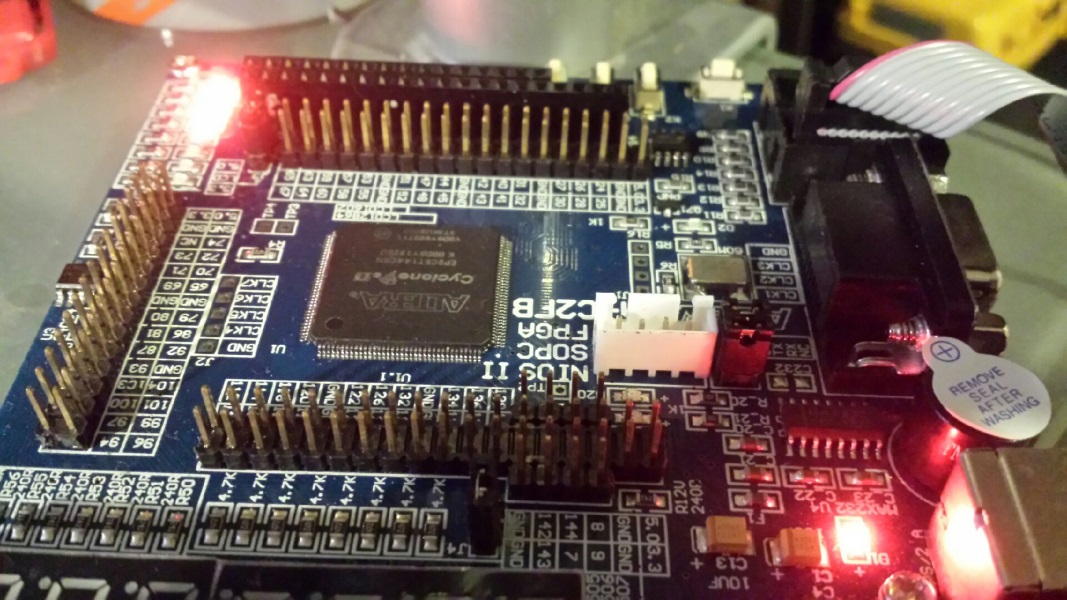


Figure 19. Almost there

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. Grrrrrrrrrrr.

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”.

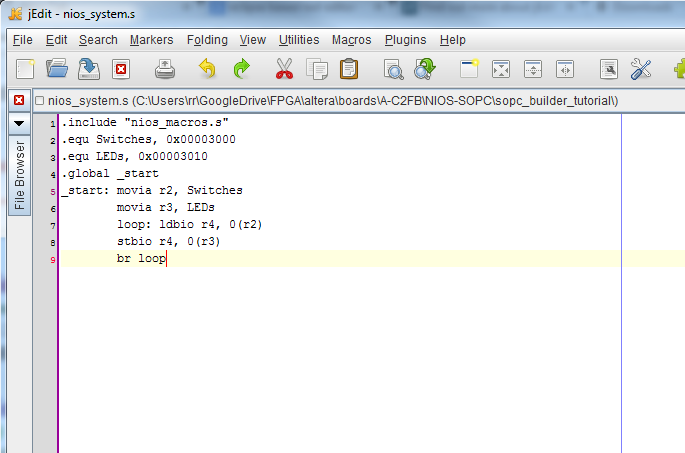


Figure 20. I am not fussy so I'll use jEdit

Run the monitor and create a new project in our current project directory (Figure 21), I called the project, go figure, “nios\_system”.

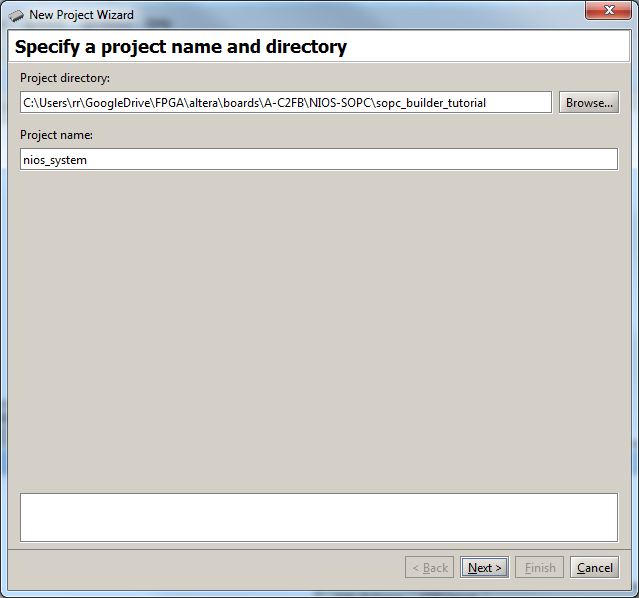


Figure 21. Altera Monitor Program "New Project Wizard"

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.

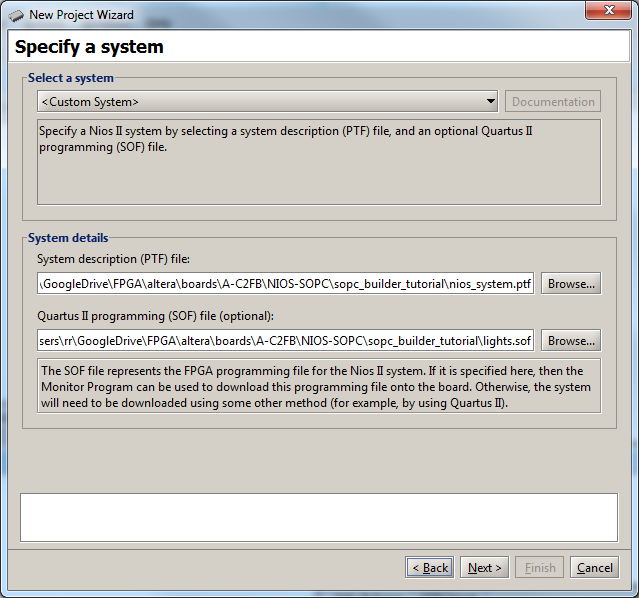


Figure 22. We’re special “Custom” that is.

Specify an “Assembly” program (Figure 23 below) and select “nios\_system.s” (which you should have created one page back).

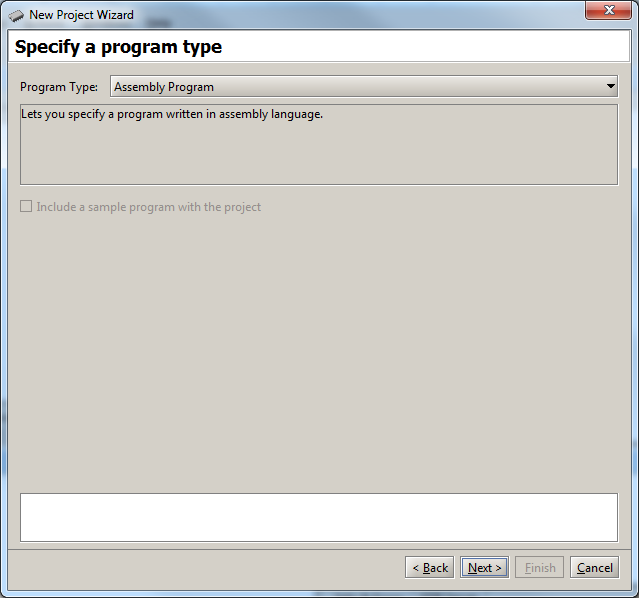


Figure 23. When was the last time you wrote assembly language?

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

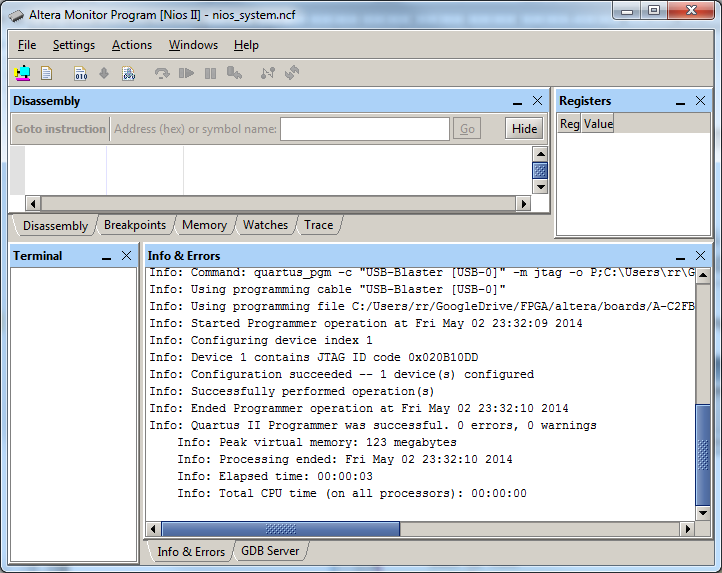


Figure 24. Successful assemble and download of design!

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

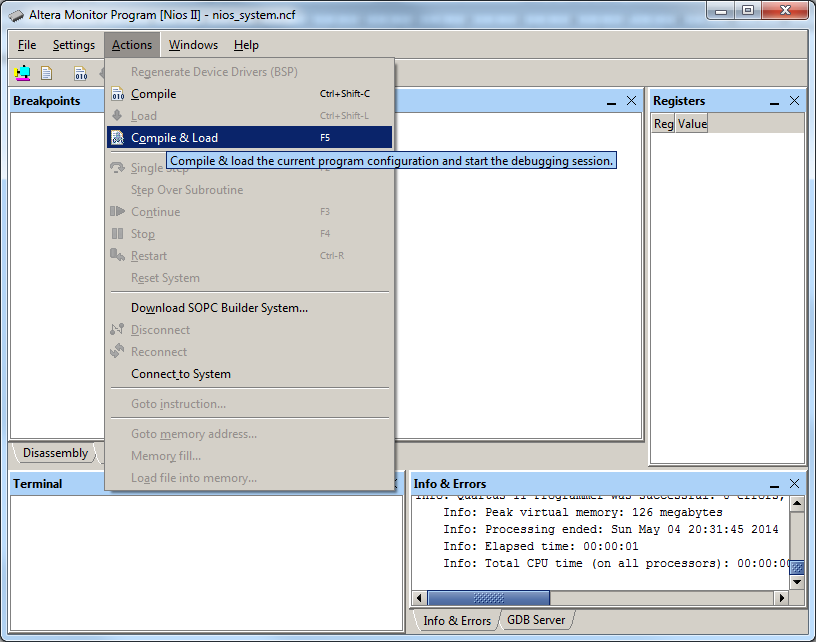


Figure 25. Load and Compile

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.

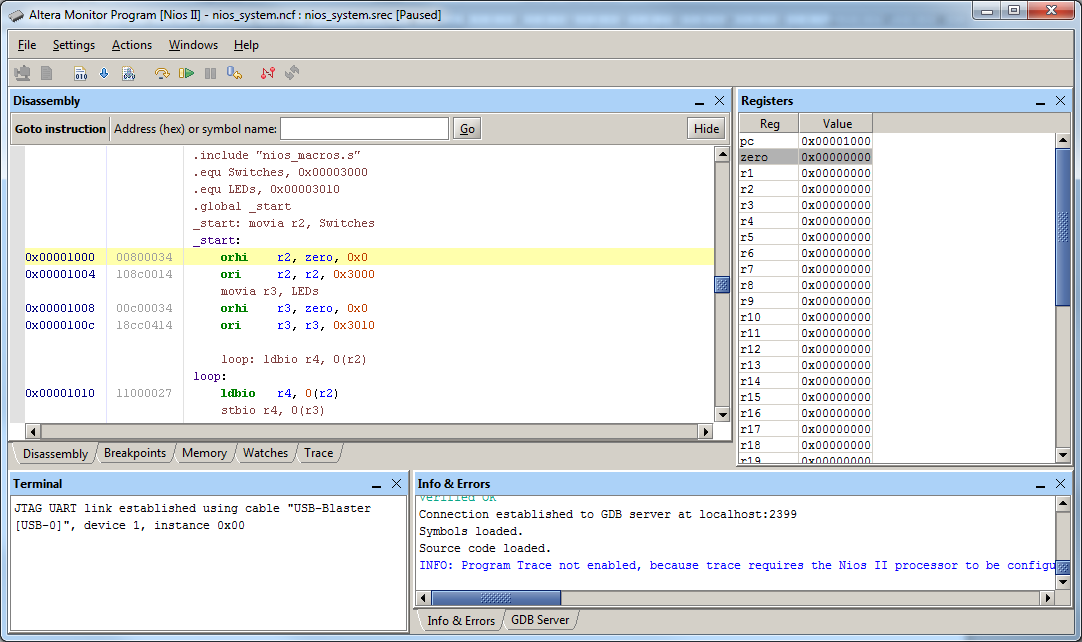


Figure 26. We are there!

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).

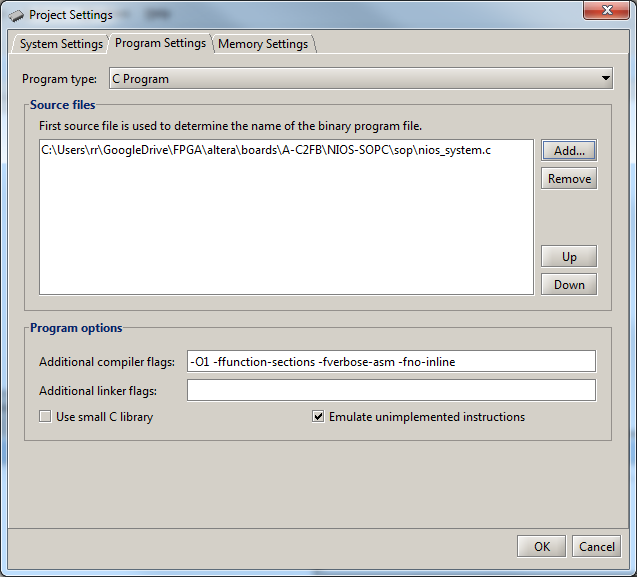


Figure 27. Too easy!

1. 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](http://www.altera.com/literature/tt/tt_my_first_nios_sw.pdf) 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.

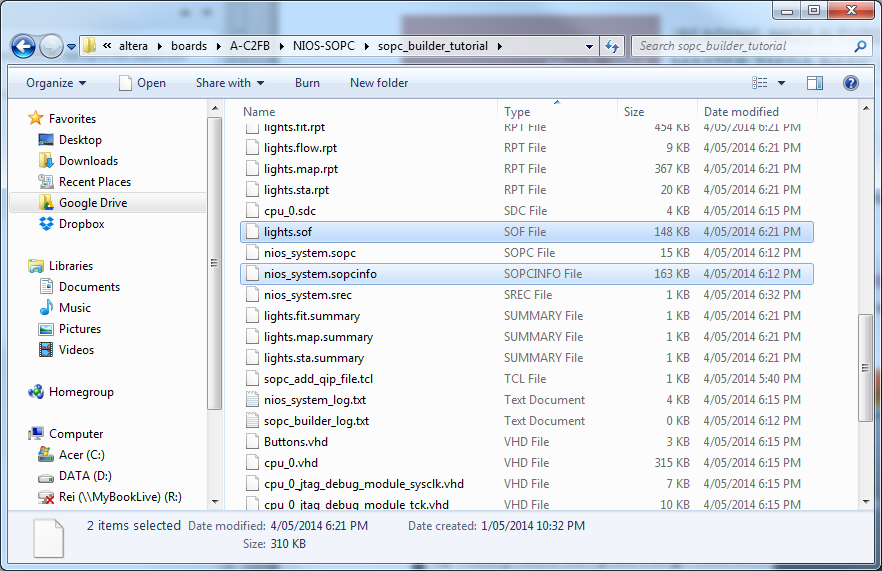


Figure 28. Our project folder

## Download Hardware Design to Target FPGA

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

***Step 3.*** for us should actually read:

1. 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.

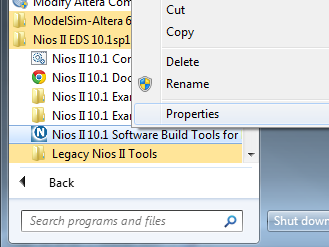


Figure 29. We need to change the properties on the shortcut

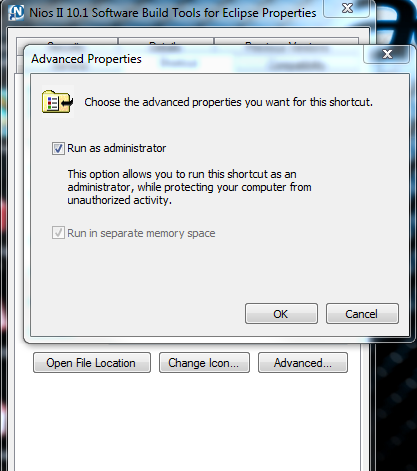


Figure 30. Nios II SBT need to have admin rights when running

***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]](#footnote-2) (Figure 31 below).

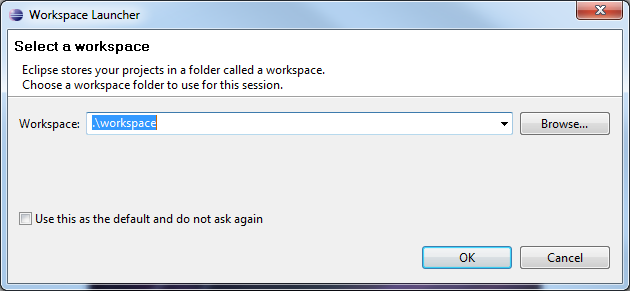


Figure 31. Find “workspace” under the running Eclipse directory

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

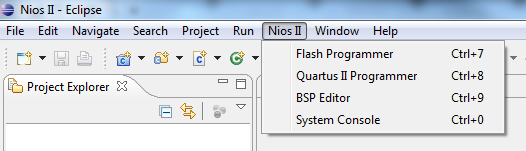


Figure 32. The Altera ® plugin menu

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).

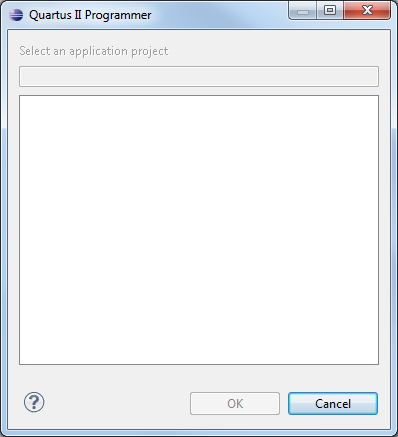


Figure 33. Makes sense, I suppose, no project so nothing in the list

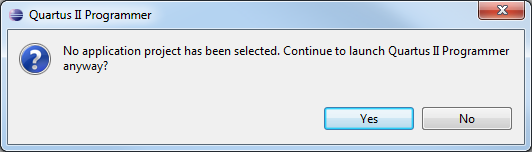


Figure 34. Yes, anyway!

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).

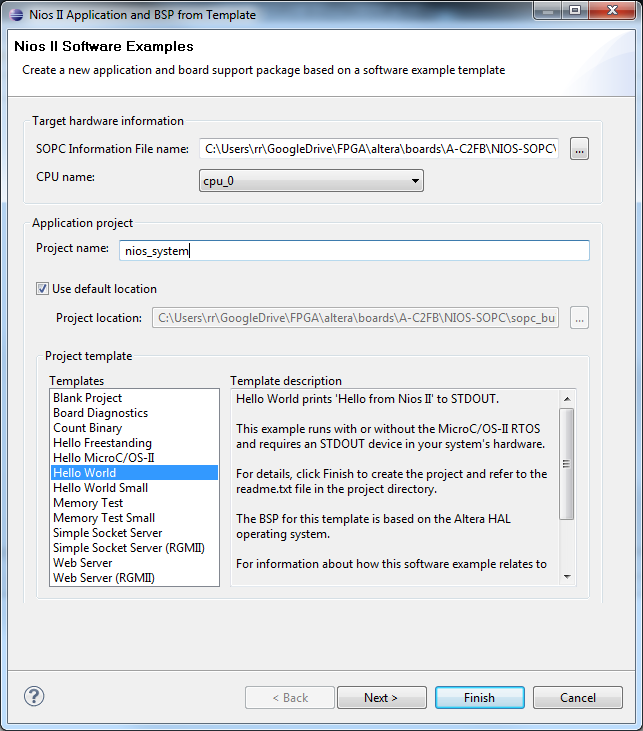


Figure 35. Our system name is "nios\_system"

***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).

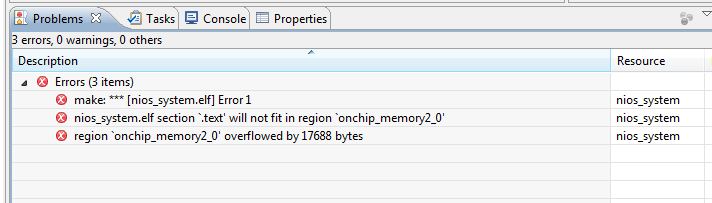


Figure 36. Drats!

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).

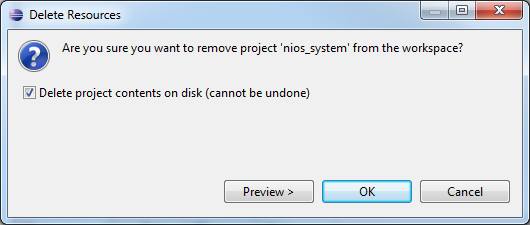


Figure 37.

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).

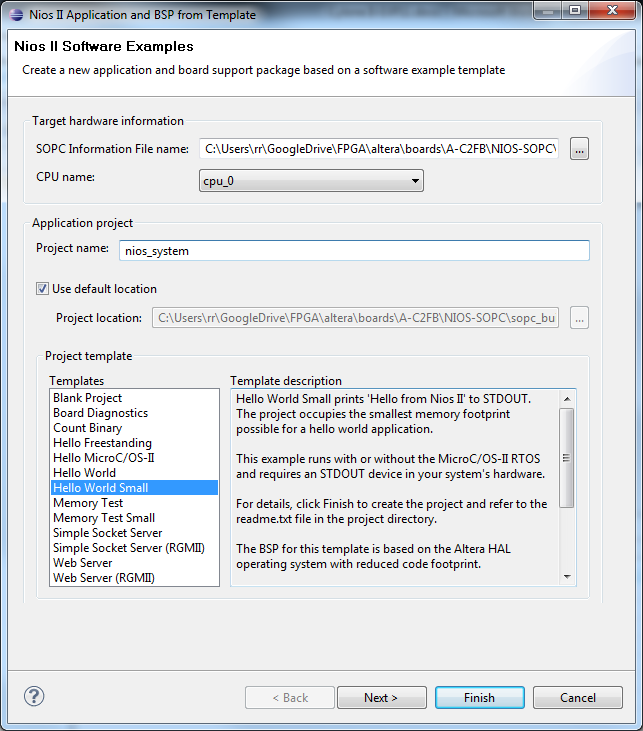


Figure 38. Smaller version!

Now I know using this “smaller” project worked once. 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.

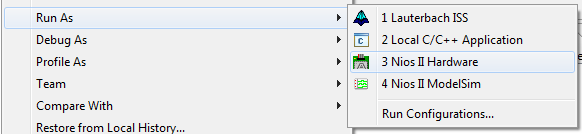


Figure 39. Nios II Hardware run configuration

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.

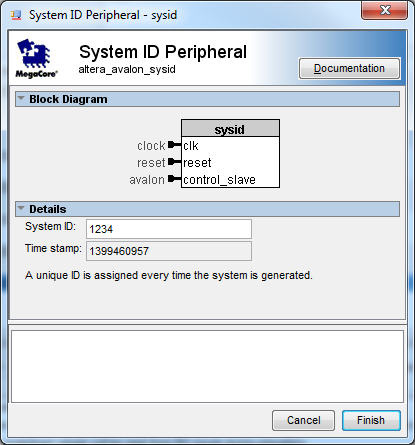


Figure 40. When using the Nios II SBT it appears we need one of these “sysid” components

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.

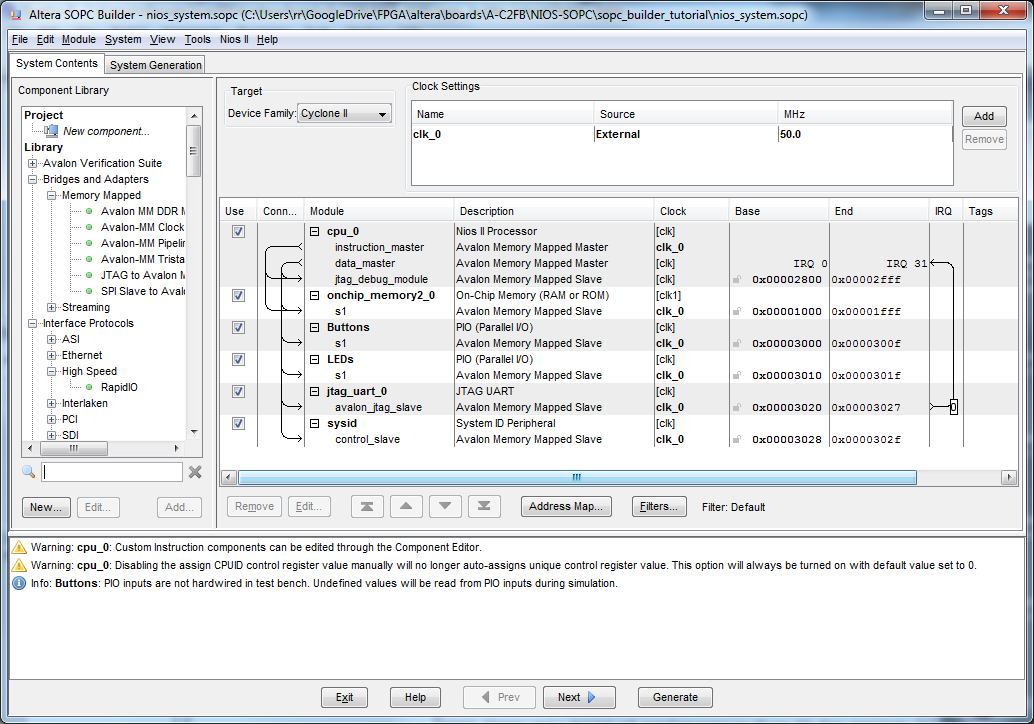


Figure 41. Pesky additional component on top of those from Ref [1] Tutorial

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.

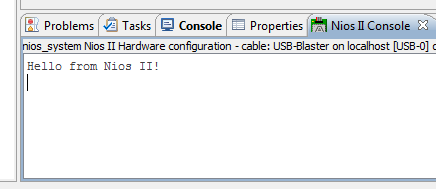
****

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).

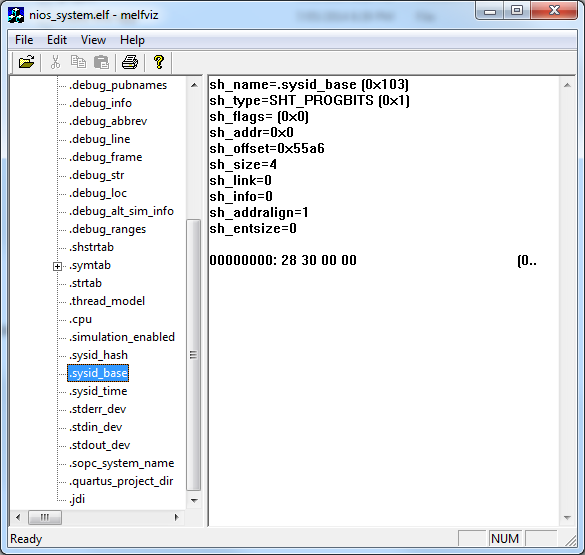


Figure . sysid\_base

The “sysid\_hash” value (Figure 44 showing 00000000: d2 04 00 00) is mysteriously un-hashed decimal “1234” (from Figure 40).

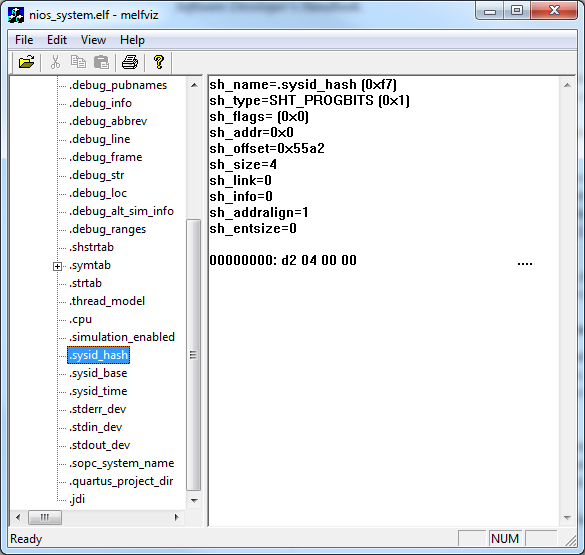
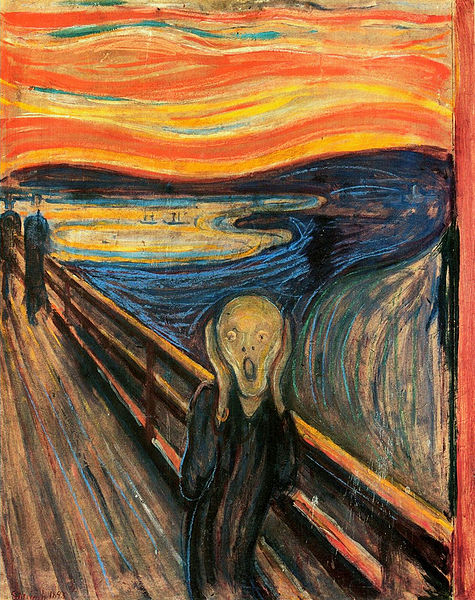


Figure . sysid\_hash

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!!



1. We won’t be using all 8 red LED’s this tutorial. [↑](#footnote-ref-1)
2. This trick is from the good old DOS command line days as “.\” literally means “in the current directory. [↑](#footnote-ref-2)