

Contents

Starting Vivado and setting up Ultra96 board design

Segway: GUI layout

Create a system

Build Software Platform

Introduction

This guide is intended to enable just about anyone to modify a ZZSoC system with Xilinx programmable logic and software platform development systems.

These systems are Vivado for programming the Xilinx microcontroller part, zcu3eg-svba484, and Vitus for creating the software platform that runs the ZZSoC/ZedBoard system.

The guide is presented in a step by step fashion. Not a lot of effort is made to explain the works behind the steps, but opportunities to describe the tools are sprinkled throughout.

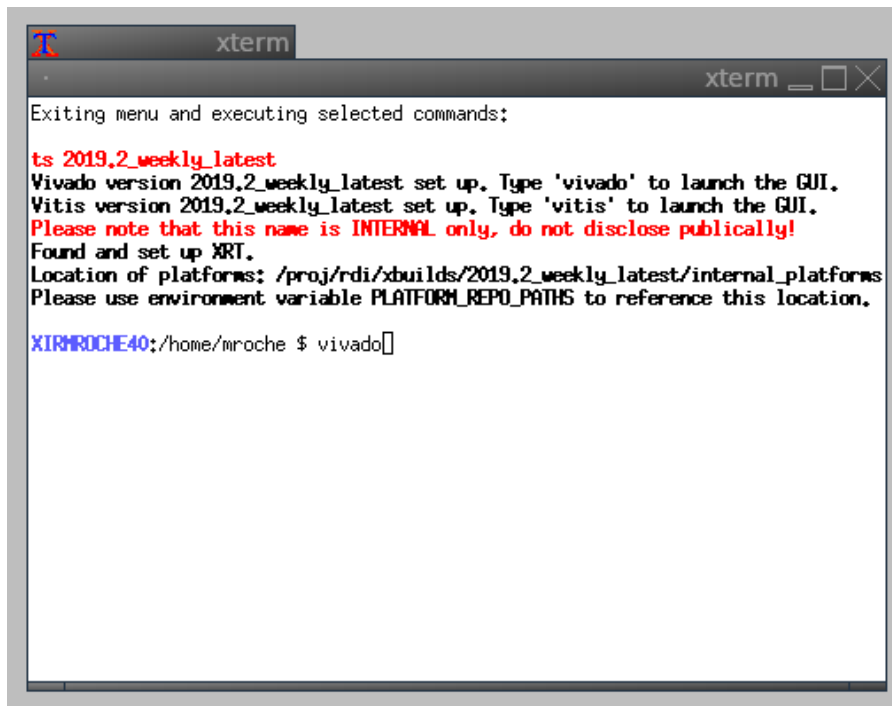
Hopefully by going through these procedures will enable users to learn where to drill down deeper into areas of interest or problems.

The guide dives right into bring up of Vivado, using the Vivado tool flow to create a bitstream programming file, and finishes with the implementation of the software platform that integrates the bitstream file into an operating system that will boot and run the part you've programmed.

It's rare that such a complex development goes without problems. Investigating, debugging, and correcting problems are left as an exercises for the user. Have fun!

Start Vivado tools

Enter Vivado & at the command prompt



```

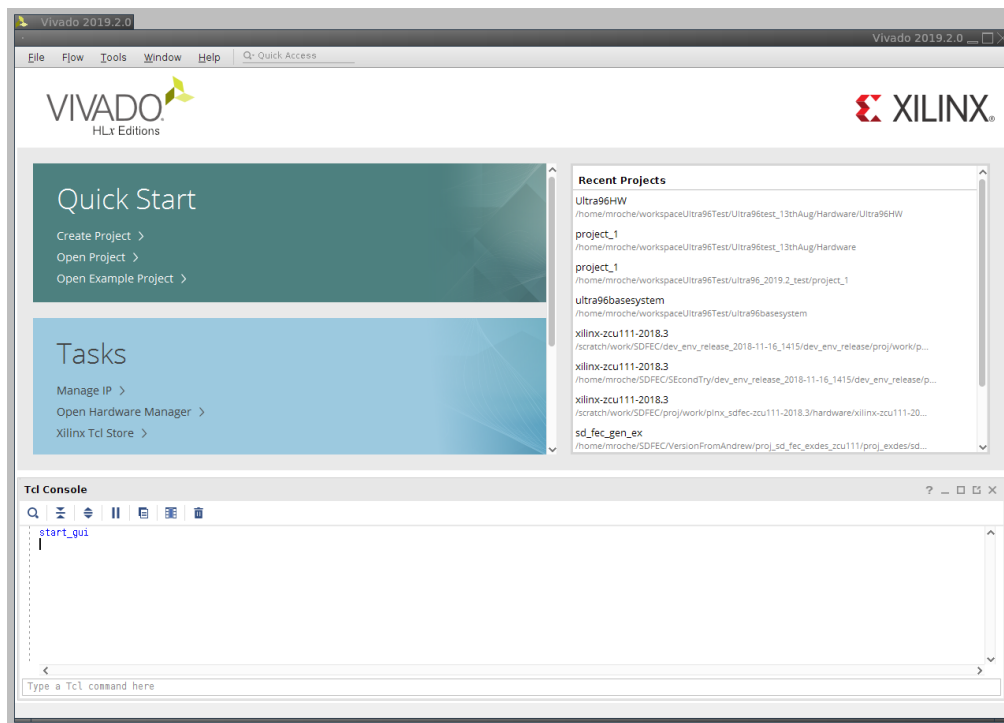
xterm
Exiting menu and executing selected commands:

ts 2019.2_weekly_latest
Vivado version 2019.2_weekly_latest set up. Type 'vivado' to launch the GUI.
Vitis version 2019.2_weekly_latest set up. Type 'vitis' to launch the GUI.
Please note that this name is INTERNAL only, do not disclose publically!
Found and set up XRT.
Location of platforms: /proj/rdi/xbuilds/2019.2_weekly_latest/internal_platforms
Please use environment variable PLATFORM_REPO_PATHS to reference this location.

XIRIROCHE40:/home/mroche $ vivado

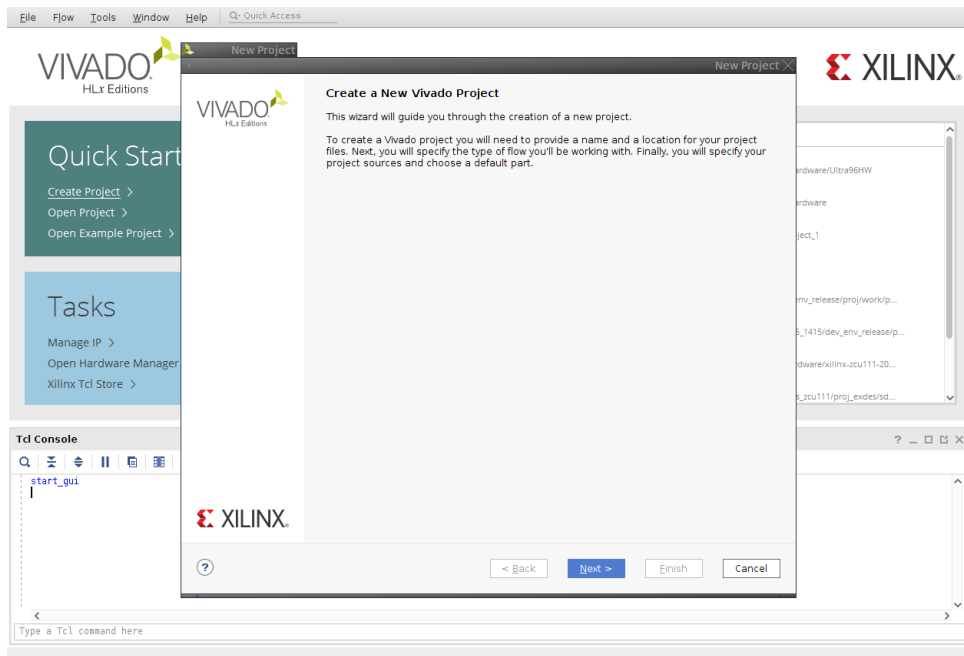
```

Tool Appears



Create the basic project with awareness of the Ultra96 board

Click "Create Project"





Hit Next and Enter Parameters as shown:

Project Name

Enter a name for your project and specify a directory where the project data files will be stored.








Project name:

Project location:  

☒ Create project subdirectory

Project will be created at: /home/mroche/Ultra96System/hardware/Ultra96_Hardware

Click Next

Select RTL Project

Ensure "Do Not Specify sources at this time" is checked

Click Next

- ☒ **RTL Project**
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
 - ☒ **Do not specify sources at this time**

Click on the Boards Tab

enter Ultra96 in Search Tab (You will need to click the "Update Board Repositories" button)

Select Ultra96 Evaluation Platform (NB Note: File Version is 1.2)

Click next

(Note if the board is not present try Update Board Repositories. ~~If that does not work change to a newer version of the vivado tools or browse to the Zedboard site above and install the board files as directed. This last option will not work with the ts tool. you have to instll the tools natively because the board files update has to be able to access the tool directories~~)

Default Part

Choose a default Xilinx part or board for your project.



Parts | **Boards**

[Reset All Filters](#)

Update Board Repositories

Vendor:

All

 Name:

All

 Board Rev:

Latest

Search:

Q ultra96

 (1 match)

Display Name	Preview	Vendor	File Version
Ultra96 Evaluation Platform		em.avnet.com	1.2

<

>

?

< Back

Next >

Finish

Cancel

Click Finish



New Project Summary

- i** A new RTL project named 'Ultra96_Hardware' will be created.
- i** The default part and product family for the new project:
 - Default Board: Ultra96 Evaluation Platform
 - Default Part: xczu3eg-sbva484-1-e
 - Product: Zynq UltraScale+
 - Family: Zynq UltraScale+ MPSoCs
 - Package: sbva484
 - Speed Grade: -1



To create the project, click Finish



< Back

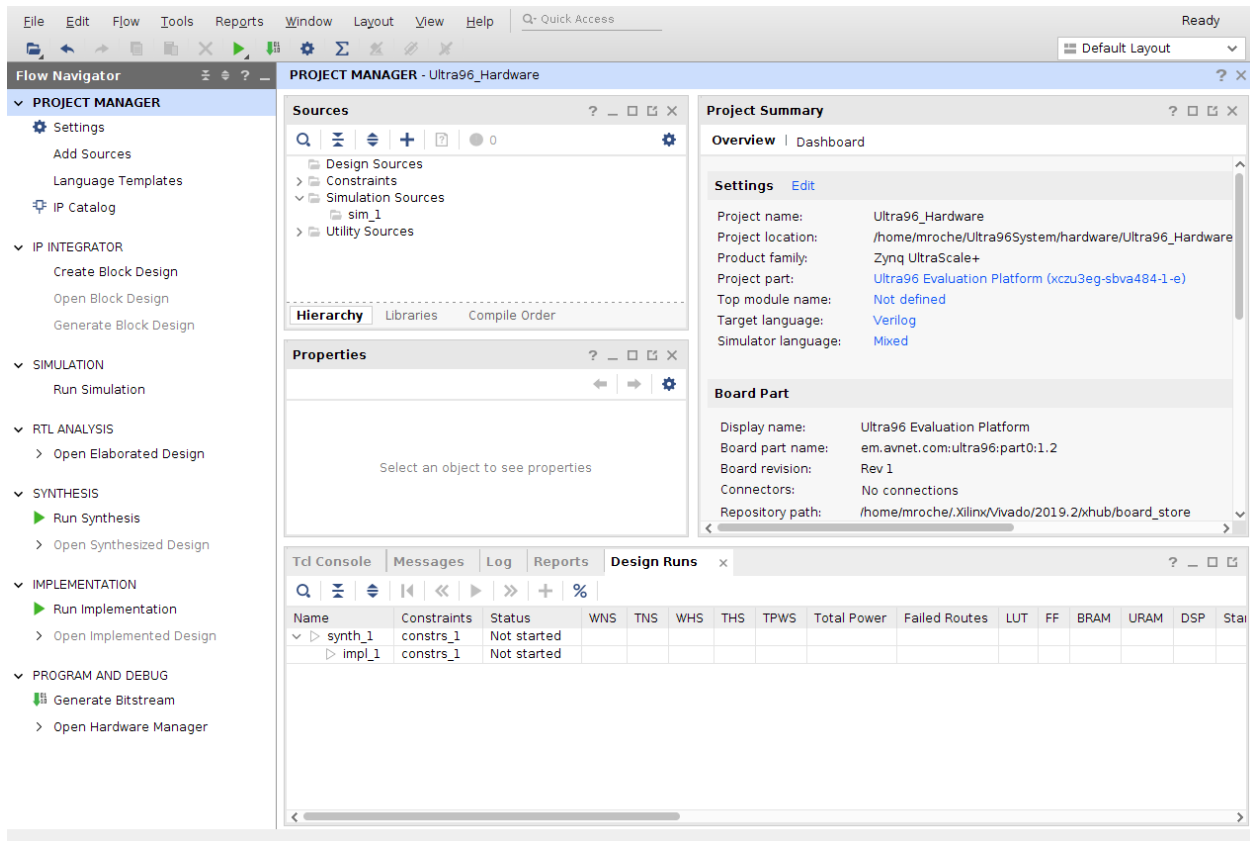
Next >

Finish

Cancel

SEGWAY: Tour of some important bits of the screen

Ok a bit of nattering on: You get the basic Vivado Screen - Note it may look slightly (or completely) different from this screen shot but the basics should be there.



On the left is the tool flow for building a bitstream. The steps are IP Integrator, Simulation, RTL Analysis, Synthesis, Implementation and Program. Please look elsewhere for details of these steps. I will lead you through one path and show some windows as we go. Try to maximize the window sizes as there is a lot of stuff going on.

In the middle you have the sources and properties.

- **Sources:** All Designs are made up of VHDL files and other files. these are text files with a .v extension. The sources tree shows you these files as you create them. Luckily the IPI flow eliminates the need for entering text and a lot of what we will do is automated. However, if you want to add your own custom design you will have to add sources manually yourself. Another tutorial.

There are 4 different source types

1. **Design Sources** → VHDL files which are hierarchal descriptions of your design starting from what is called a top file. We will not do much with this in this tutorial. Just keep an eye open and look at the files as they are created just don't edit them
2. **Constraints:** These files are the constraints you impose on your design to ensure that certain design rules are met. One of these constraints is mapping internal signals to external physical pins. More of this anon
3. **Simulation Sources:** Simulation sources are like test scripts for hardware, kind of like unit test. you can simulate the logic in the host and put various test stimuli etc. we will not be doing this at all. Another tutorial
4. **Utility Sources:** No idea what this is, and we won't be using it.

- **Properties:** Associated with each file and design element (later) there are properties. This window shows these properties. Keep an eye on it as you go through the steps. we don't need to spend too much time

On the bottom of the window are various build related output and control interfaces. Most important for us is the Tcl Console. Everything that is done in Vivado is a tcl script. So, selecting an option or changing a wire or any design translates down to a Tcl Script. This is very useful as then we can automate most of the steps. We will do a couple of things with this, but the detail is for your research.

Create the Basic Bitstream Design

Let's create a system:

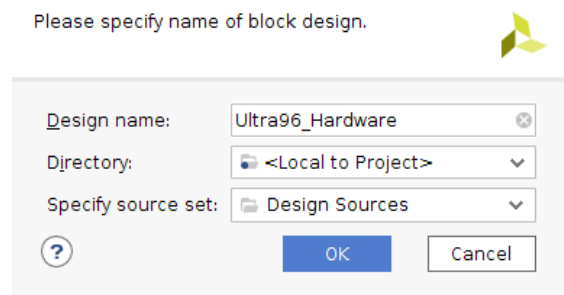
IPI Block Design

Starting with the IP Integrator (IPI) to create your system graphically. Hold on to your hat....

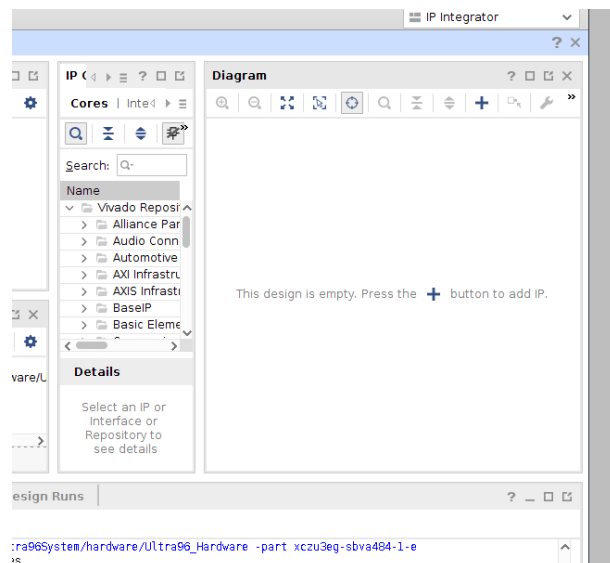
On the left click Create Block Design under the IP INTEGRATOR Step.

Enter a name but don't change anything else.

Click OK

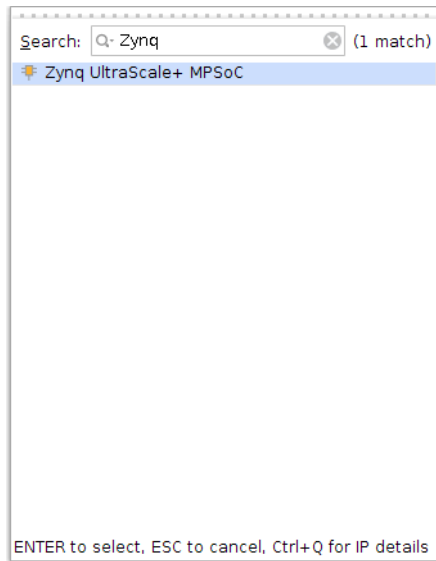


Block Design is created on the right-hand side.



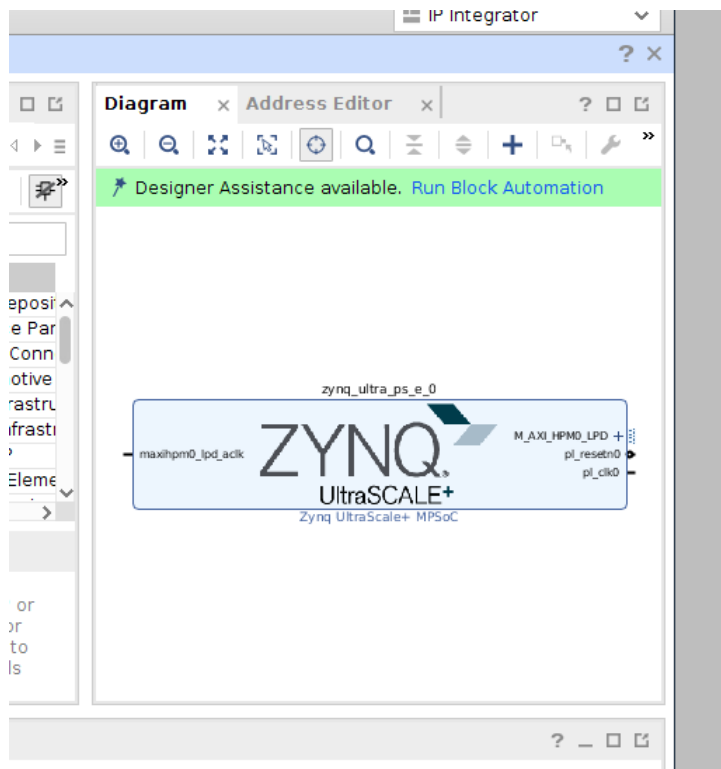
In the Diagram Window click the bit + to add IP (Or the + in the icon bar at the top of the Diagram window)

in the list that appears type Zynq in the Search bar. Double click on the Zynq Ultrascale+ MPSoC



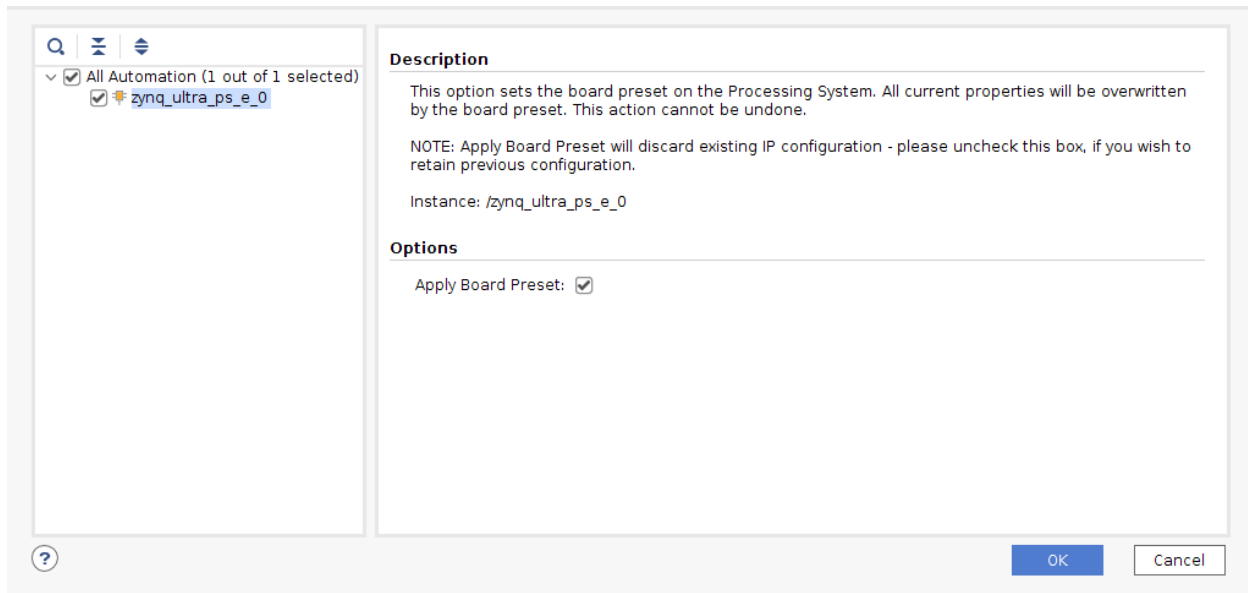
A Zynq block appears on the Design Window.

On the Top of the design window Click on the "Run Block Automation Step"



In the screen that appears check all the boxes on the left and press OK

Automatically make connections in your design by checking the boxes of the blocks to connect. Select a block on the left to display its configuration options on the right.



Automation is Run and the Zynq is now configured for your board. Note the new pins on the Zynq block.

Connecting some clocks

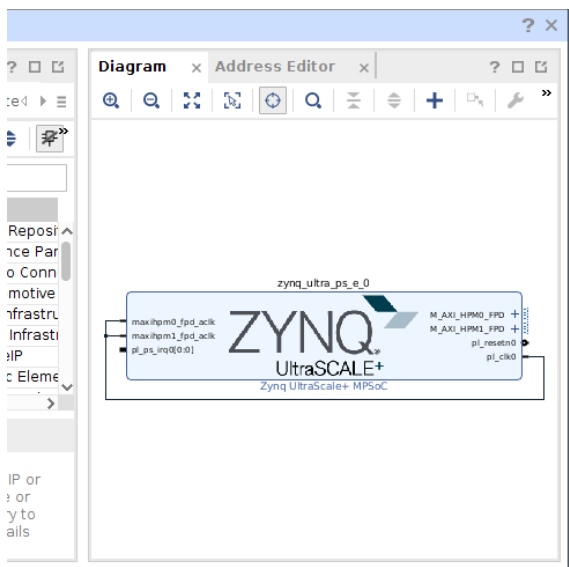
You'd think the Block would just do this step itself but there is a reason for it. For now just do it.

Left Click and hold on the pl_clk0 pin on the right of the block

Drag the line over to the pm0_fpd_aclk and let go mouse button

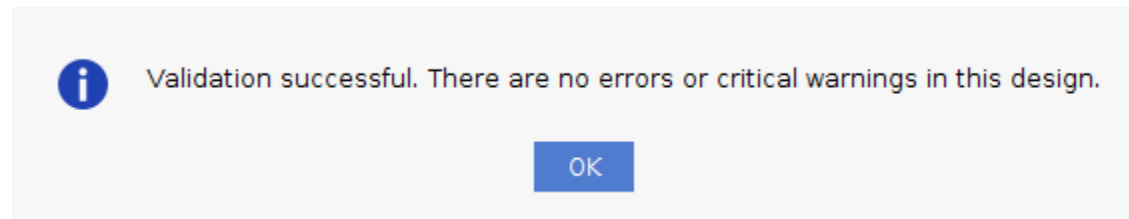
A line is routed to connect the clk sources

Do the same again for the pm1_fpd_aclk



In the Tools menu select "Validate Design"

You get this screen which tells you that so far everything is connected correctly. it is usefull to do this a few times while designing before you go to far as ocasionaly you might break the design. I will assume all is good at this point.



SEGWAY: The self-contained PS Design is complete

Zynq processors contain two distinct domains (among others)

The PS or Processor System: The PS is a microcontroller that runs independently on the chip. The PS microcontroller contains a ton of different peripherals which are connected to the outside world independently of the rest of the chip. These are called MDIO connections and are standard connections. To add a little confusion, you can actually "remap" or select different peripherals for external connection when the device has limited pins. So, you might select 2 spi ports and one serial port or some other combination. This is completely configurable.

The PL or Programable logic: this is custom designs you might make yourself.

At this point it is interesting to know that you have done all the steps needed to create a Processor only design with all the PS peripherals routed to the correct pins on the device on this Ultra96 system. You could click "Generate bitstream" now on the left-hand side and after a lot of time you will get the basic hardware system. However, we are going to take this further and add a GPIO peripheral into the programable logic and connect it as a new peripheral to the PS.

SEGWAY: The Zynq Configuration:

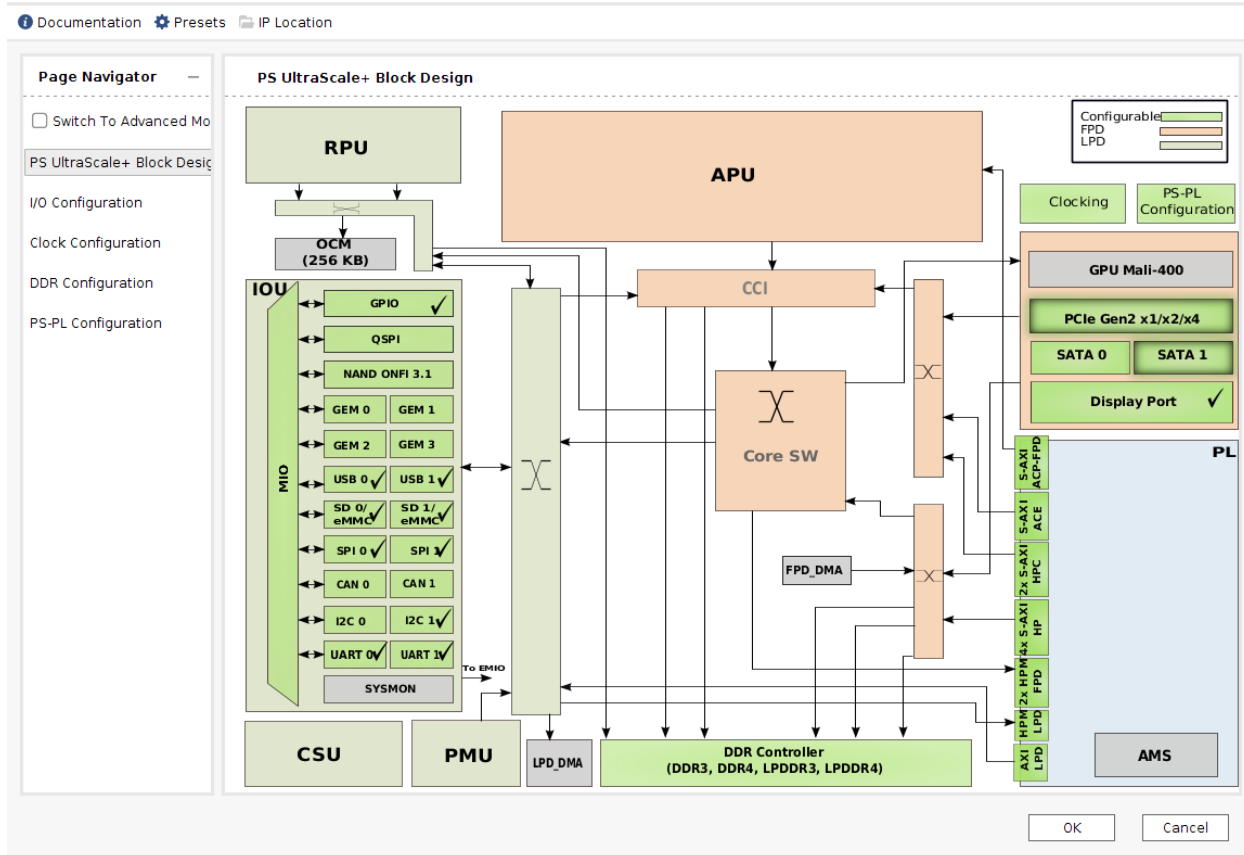
If you double click on the Zynq block in the design

Brings up a window to let you customize the Zynq if you want

Explore around but don't change any of the settings.

The defaults are set up for our board, so you don't have to do anything.

Zynq UltraScale+ MPSoC (3.3)



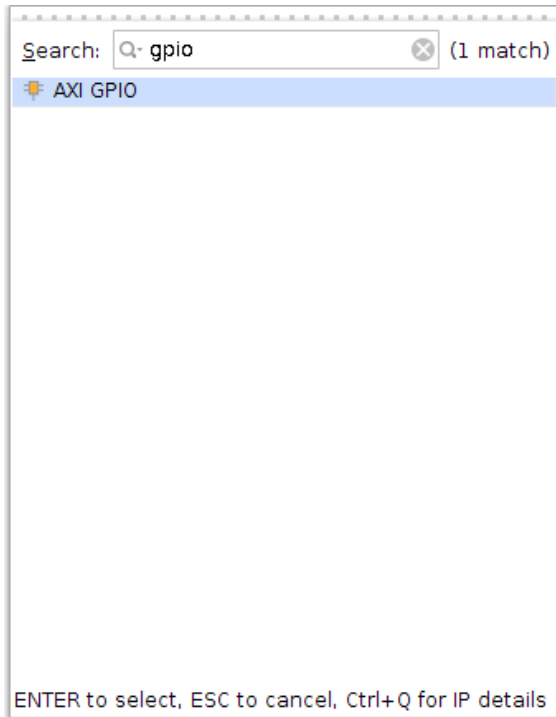
It is interesting to see from the block diagram which of the PS blocks peripherals are connected and which are not. If you select the items on the left, you can see how each part is mapped to MDIO pins and other things. You can change this but that is for another tutorial. Don't change anything. Hit Cancel.

Adding a GPIO Peripheral and configuring

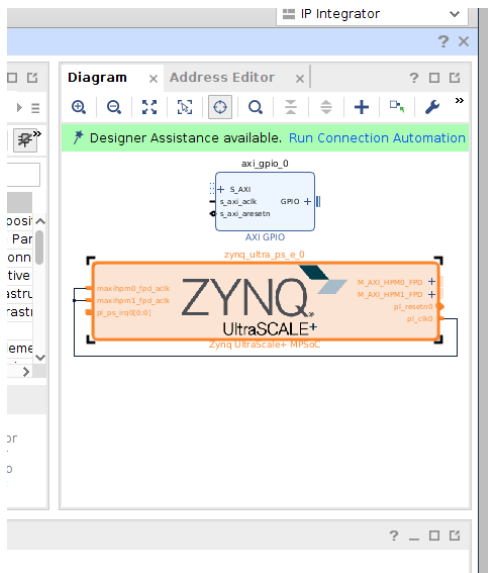
Back to the diagram window and press "+"

Type GPIO in the search box

Double click on the AXI GPIO Block



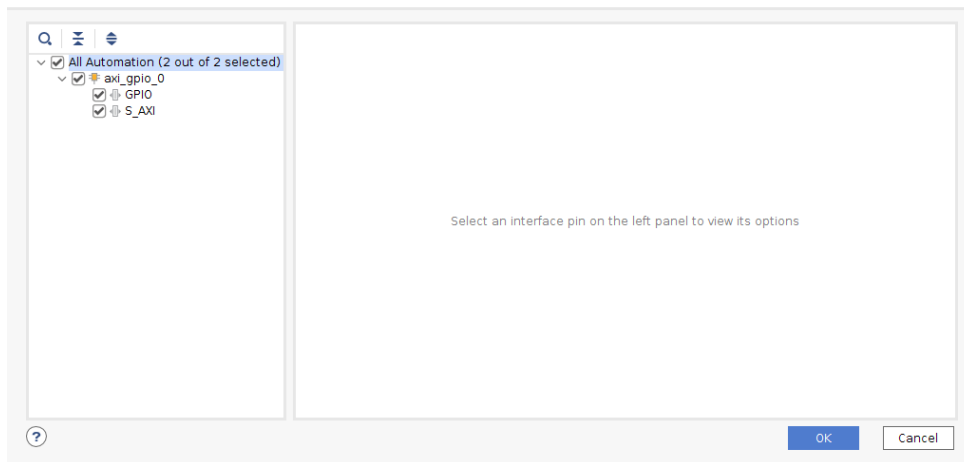
AXI GPIO appears on the Diagram window



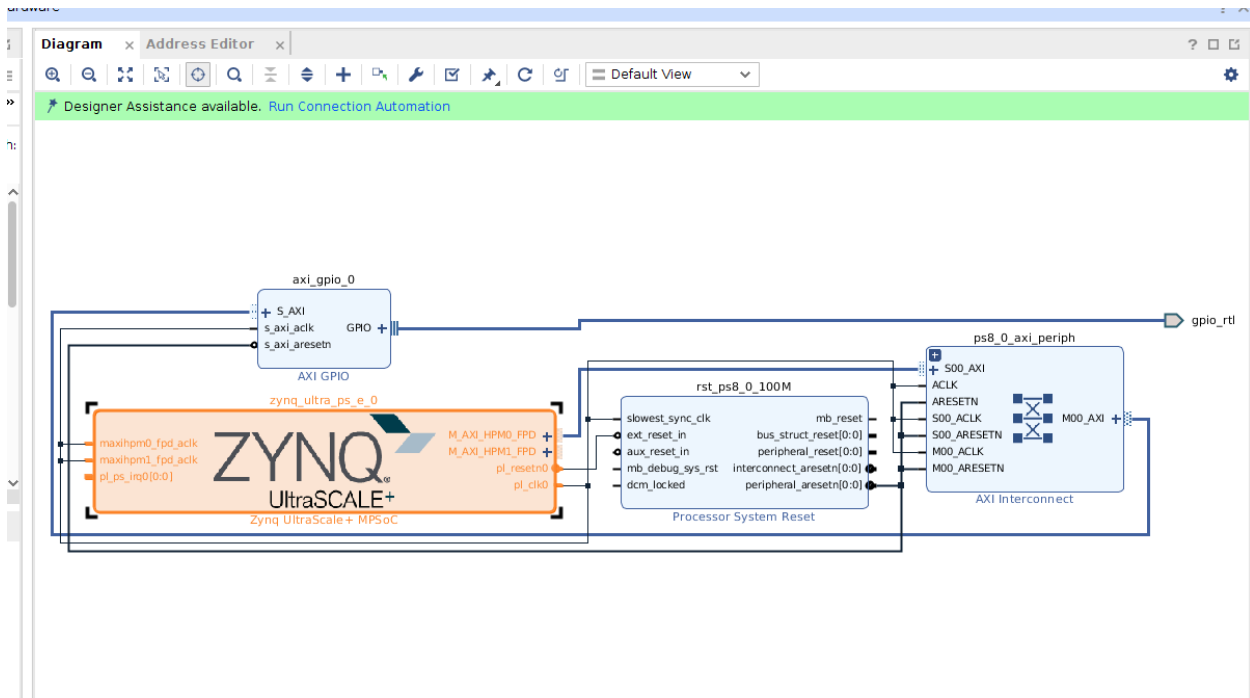
Press Run Connection Automation

Check all boxes on the left-hand side and press OK

Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.



This is to add necessary blocks and connect them together



Press "Run Connection automation" again

Select all the check boxes on the left and press OK

Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.



All Automation (1 out of 1 selected)

zynq_ultra_ps_e_0

M_AXI_HPM1_FPD

Description

Connect Master interface (/zynq_ultra_ps_e_0/M_AXI_HPM1_FPD) to a selected Slave interface. To connect a Master interface to an unconnected Slave interface, please use connection automation starting from the Slave Interface.

Options

Slave interface

/axi_gpio_0/S_AXI

Bridge IP

/ps8_0_axi_periph

Clock source for driving Bridge IP

/zynq_ultra_ps_e_0/pl_clk0 (100 MHz)

Clock source for Master interface

/zynq_ultra_ps_e_0/pl_clk0 (100 MHz)

Clock source for Slave interface

/zynq_ultra_ps_e_0/pl_clk0 (100 MHz)

?

OK

Cancel

This is to connect and configure the added axi blocks to the Processor system on the Ultra96.

The diagram illustrates the configuration of the Ultra96 processor system. It shows the ZYNQ UltraSCALE+ MPSOC block connected to several peripheral blocks. The 'axi_gpio_0' block is connected to the 'S_AXI' port of the 'zynq_ultra_ps_e_0' block. The 'rst_ps8_0_100M' block is connected to the 'mb_reset' and 'peripheral_reset' signals of the 'zynq_ultra_ps_e_0' block. The 'ps8_0_axi_periph' block is connected to the 'S00_ACLK', 'S00_ARESETN', 'M00_ACLK', 'M00_ARESETN', 'S01_ACLK', and 'S01_ARESETN' signals of the 'zynq_ultra_ps_e_0' block. The 'ps8_0_axi_periph' block is also connected to the 'gpio_rtl' block. The diagram shows the internal connections and signal paths between these blocks, including the 'Processor System Reset' and 'AXI Interconnect' blocks.

Select Tools→ Validate Design

Design is Valid.



Validation successful. There are no errors or critical warnings in this design.

OK

SEGWAY: What happened?

the GPIO block is the important part you wanted but what are the other blocks? Vivado added automatically blocks which are necessary to connect any PL Peripheral to the PS. Namely an AXI Bridge IP and a Reset IP.

AXI Bridge: Simply Put AXI is the bus Infrastructure for ARM processors. To connect a peripheral you have to daisy chain it to the processor through this translation. The bridge is a crossbar switch that maps your peripheral into the address space of the processor

Reset Block: Again the Reset infrastructure is critical to the processor. this block connects your peripheral and all subsequent peripherals to the processor.

An IO block called `gpio_rtl` is also added. this is a special place where all your IO's are mapped to named nets. These nets are then connected to Pins.

Connecting the pins - Constraints editing

You have now created the internal Design. The next step is to connect the gpio to actual pins.

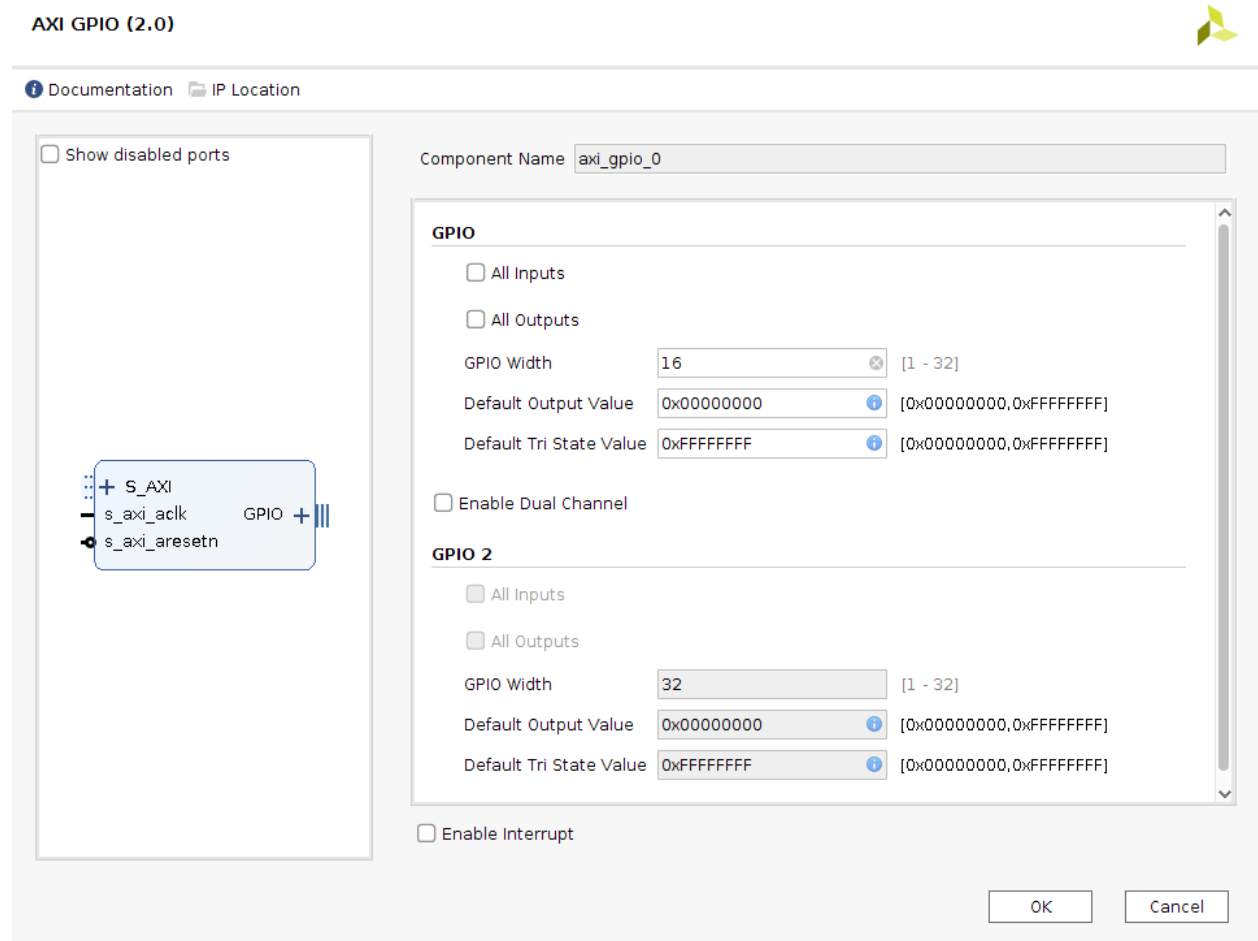
First, we have only 16 gpios on the Ultra96. Change the GPIO peripheral to 16 bits wide

double click on axi_gpio_0

change the GPIO to 16 (from 32 default)

Press OK

Leave all other settings alone for now. However, note you could have a dual channel GPIO and it could be interrupt driven.



Next for convenience we will change the name of the gpio_rtl pins to a more obvious name

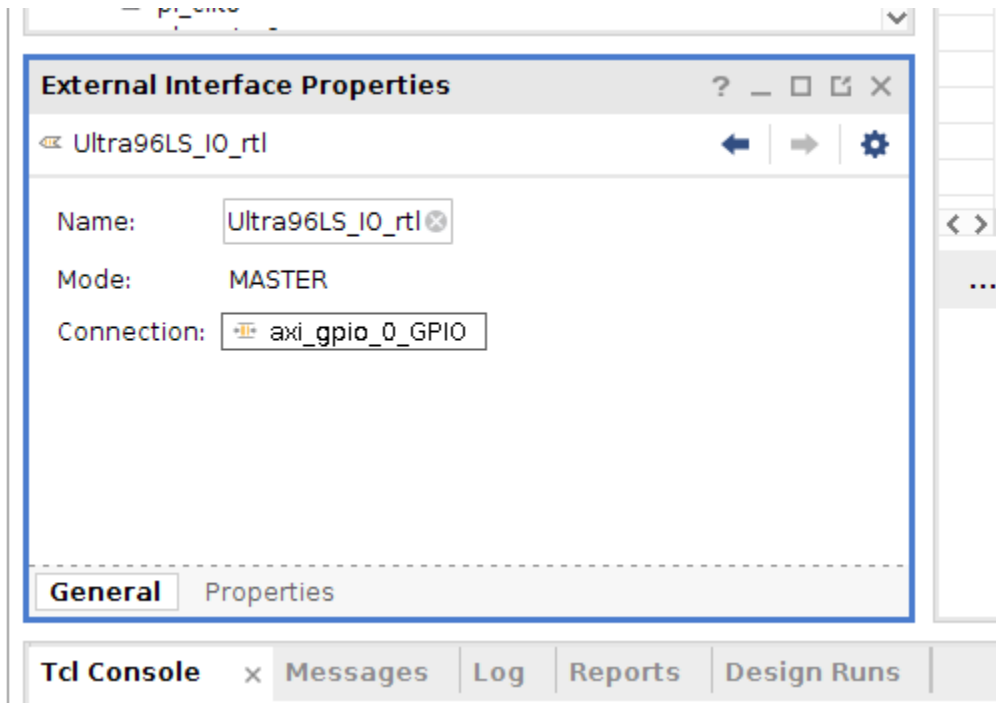
Double click on the gpio_rtl connector



Note that the properties window we discussed before in the middle of the screen changes to External Interface Properties

In the General tab change the Name to Ultra96LS_IO_rtl

<CR>



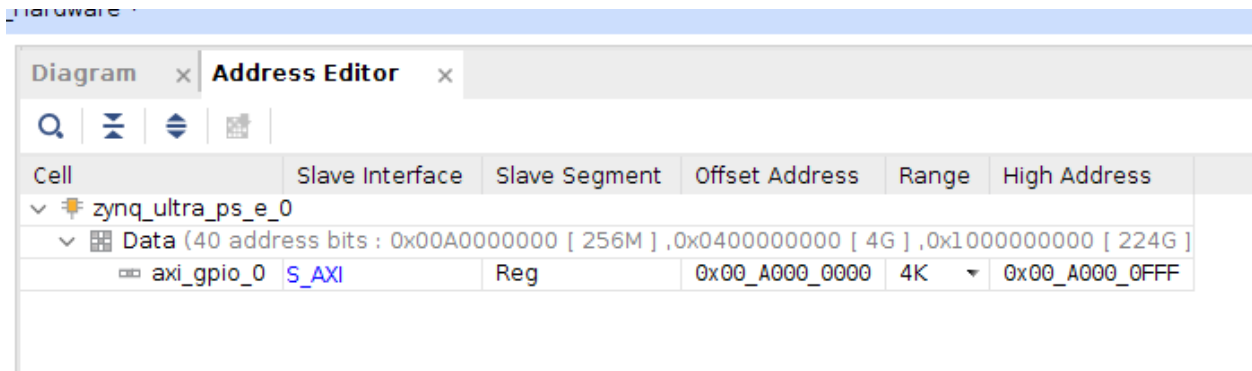
note the name has changed on the Diagram



Click on the Address Editor tab next to the Diagram text in the Diagram Window.

Note the base address for the GPIO is set to a default location by the tools.

Nothing to do here. this is just for information



Next, we must generate what is called the RTL wrapper or top level vhd file for this design

Go to the Sources Window in the middle of the screen.

Select the sources tab

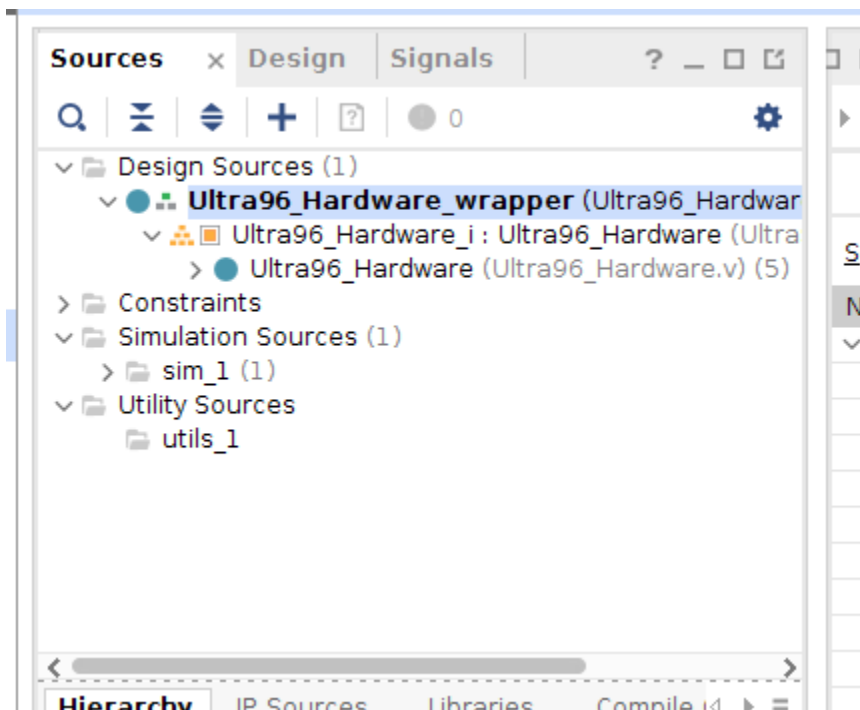
Open the Design Sources drop down and a file is there with extension .bd → this is the block diagram

Right click on the Bd line and select "Create HDL Wrapper" from the menu

Select Let Vivado manage wrapper and auto-update

Select OK

This is yet another automated step where Vivado hides updates to your project files. in the end it will have created a new wrapper file under the Design Sources tab.



OK to do the mapping we now have to generate the bitstream up to the Implementation stage of the build flow.

on the left-hand side of the screen double click on run implementation

you will be asked to generate the other steps first, just press OK and the tool flow will start



There is no netlist available. OK to launch synthesis first? Implementation will automatically start when synthesis completes.

OK

Cancel

On the next Dialog you will be asked how much of your machines resources need to be used to generate the design. Under linux/windows the number of Jobs = the number of cores on your machine. Leave it at the max. press OK.

Launch the selected synthesis or implementation runs.



Launch directory: <Default Launch Directory> ▼

Options

- ☒ Launch runs on local host: Number of jobs: 24 ▼
- ☐ Launch runs on remote hosts
- ☐ Launch runs on Cluster
- ☐ Generate scripts only

☐ Don't show this dialog again

OK

Cancel

Then the design run start dialog will start

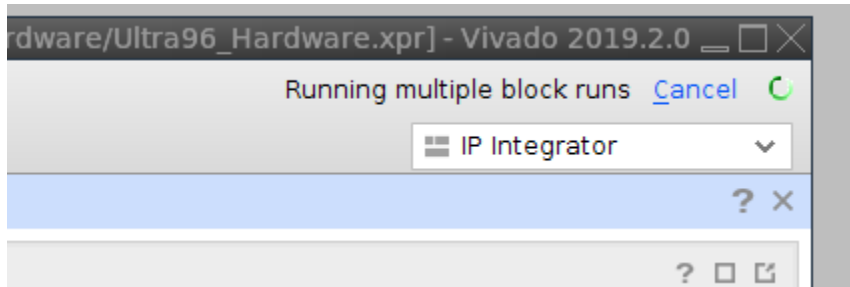
Starting design runs...



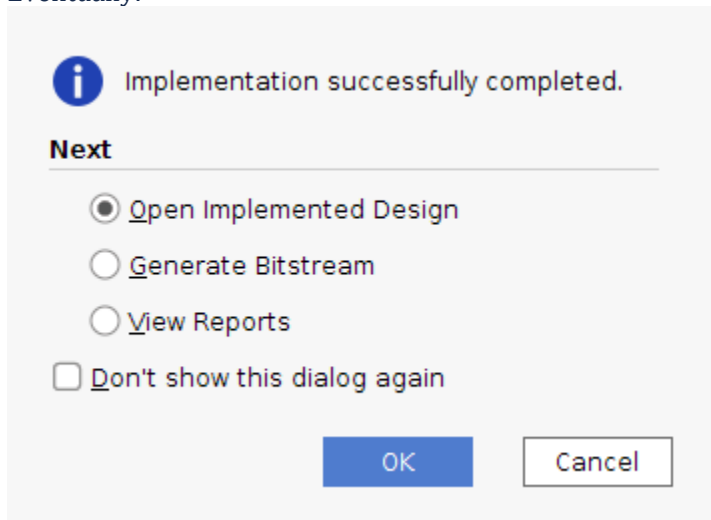
Background

After a while this disappears and Vivado starts synthesis proper. you can see the progress on the top right.

Then either go for a coffee or go home for the night depending on the speed of your machine. This takes a while to complete.



Eventually:



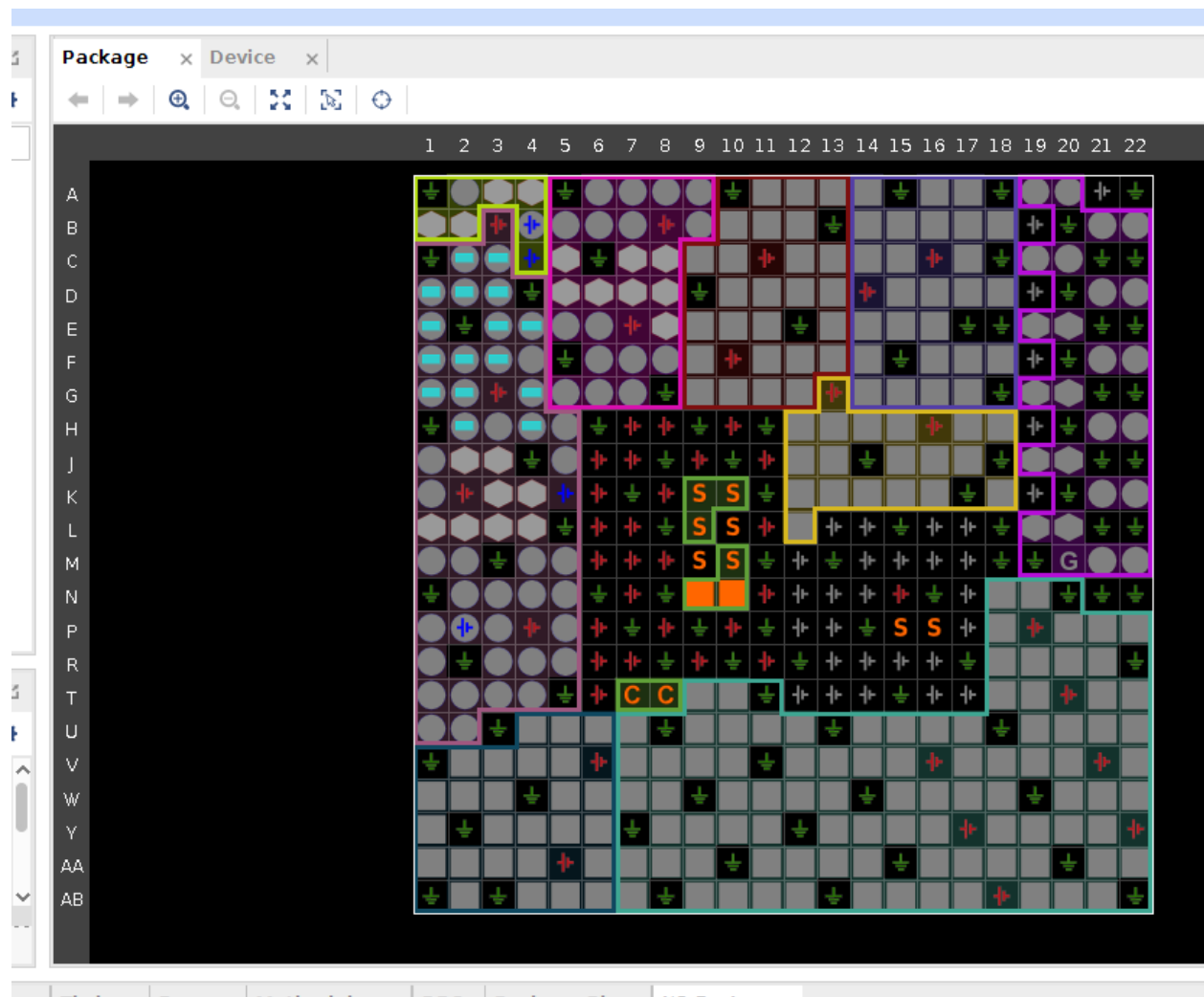
This took 20 minutes on my old dual core i5 Acer laptop with 8G memory. YMMV.

OK now we have a design which is implemented inside the device. all the simulation and analysis place-and-route is done EXCEPT the connection to pins.

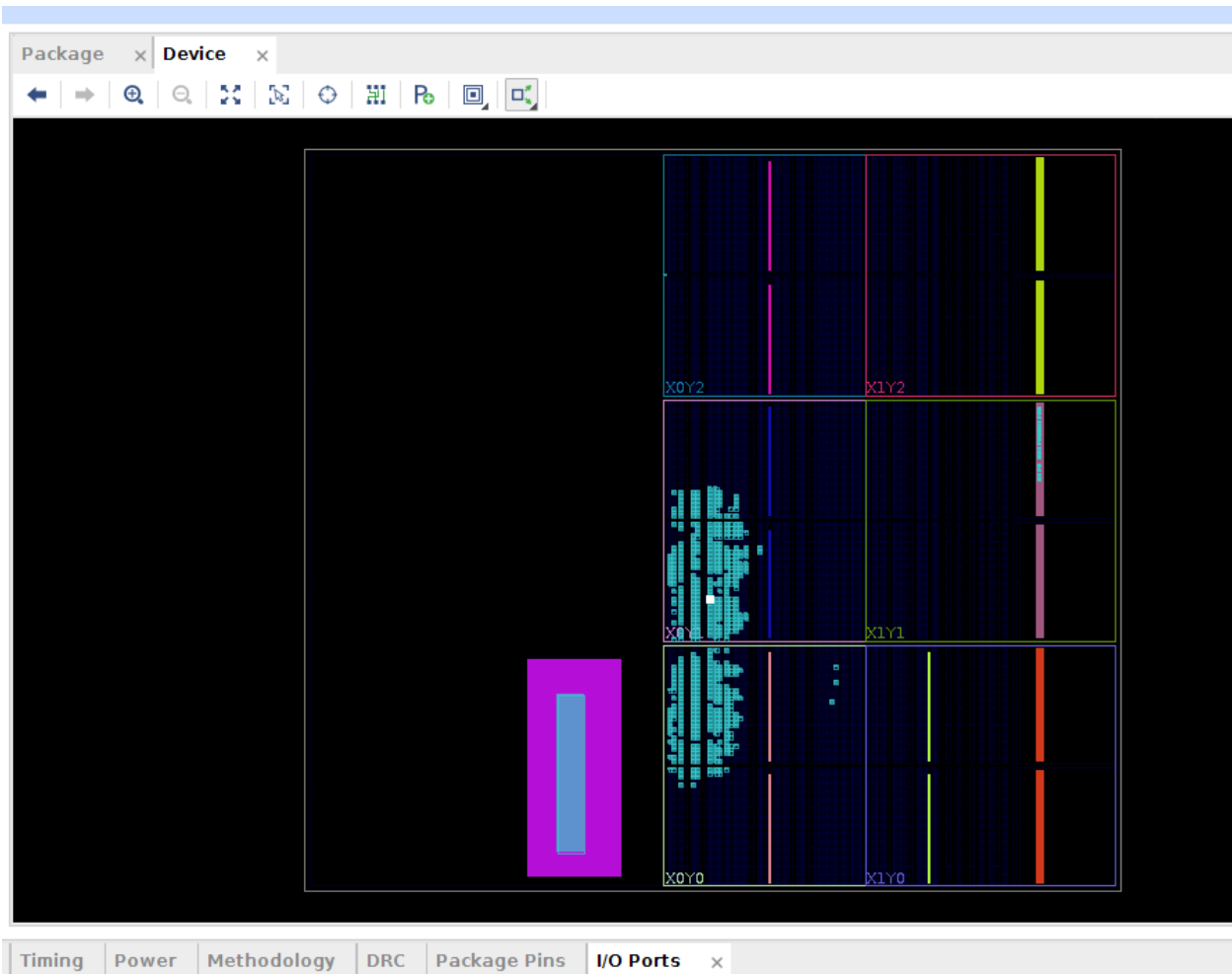
Click OK in the above dialog: new windows will appear*. This is like a new view of the implementation of your design. here you can see things such as:

*Note: If these windows do not appear automatically select Window → Device\Package\I/O Ports

The Package - this shows a view of the pins in the device. in this case it is a Pin Grid array device. pins are specified as x/y coordinates according to the letters and numbers above and to the left of the device:



The Floor plan - this is how vivado used the logic cells in the device. The Device Tab shows a depiction of the Zynq device according to the resources (IO, Logic cells PS, etc.) the Cyan represents used or partially used logic cells. In this case the vertical lines represent the IO banks on the device. You can play with this. Zoom in to the logic cell level.



Of interest to us is the I/O Ports tab on the bottom of the screen. Expand the ports and you will see:

Name	Direction	Board Part Pin	Board Part Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew
▼ All ports (16)												
▼ GPIO_42276 (16)												
▼ Ultra96LS_IO_rtl_tri_io (16)												
↻ Ultra96LS_IO_rtl_tri_io[15]	INOUT				G2	▼	65	default (LVCMOS18)	▼ 1.800		12	▼ SLOV
↻ Ultra96LS_IO_rtl_tri_io[14]	INOUT				H2	▼	65	default (LVCMOS18)	▼ 1.800		12	▼ SLOV
↻ Ultra96LS_IO_rtl_tri_io[13]	INOUT				G4	▼	65	default (LVCMOS18)	▼ 1.800		12	▼ SLOV
↻ Ultra96LS_IO_rtl_tri_io[12]	INOUT				H4	▼	65	default (LVCMOS18)	▼ 1.800		12	▼ SLOV
↻ Ultra96LS_IO_rtl_tri_io[11]	INOUT				F1	▼	65	default (LVCMOS18)	▼ 1.800		12	▼ SLOV
↻ Ultra96LS_IO_rtl_tri_io[10]	INOUT				G1	▼	65	default (LVCMOS18)	▼ 1.800		12	▼ SLOV
↻ Ultra96LS_IO_rtl_tri_io[9]	INOUT				E3	▼	65	default (LVCMOS18)	▼ 1.800		12	▼ SLOV
↻ Ultra96LS_IO_rtl_tri_io[8]	INOUT				F4	▼	65	default (LVCMOS18)	▼ 1.800		12	▼ SLOV

OK Finally Mapping the pins

The Package Pin Column of I/O Ports has the mappings for the pins. For now, the tool has "picked" available outputs from the device and assigned them to whatever IO's were available. These defaults are not according to your board design. So how do I remap them. There are 3 ways:

- 1: Go down through this list one at a time and change the mapping to the correct package pin
- 2: Open the Tcl console and use the `set_ports` command to set it from Tcl
- 3: create a Constraints file - The advantage of this way is that you can create the file before you do the above run through the tool flow saving you a step. I will focus on the Constraints file.

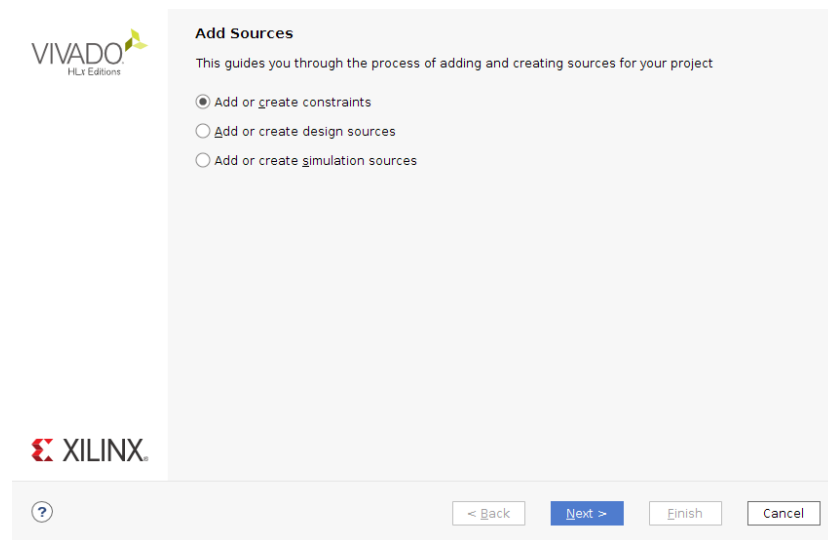
Populating the constraints file

Go to the Sources Tab in the middle of the screen

Right Click on the Constraints header

Click Add Sources


Select Add or Create constraints



Hit Next

Press the + or press Create File

Enter a name for your constraints file for example:

Create a new constraints file and add it to your project 

File type: XDC

File name: Ultra96_Constraints

File location: <Local to Project>

? OK Cancel

Press OK

Press Finish

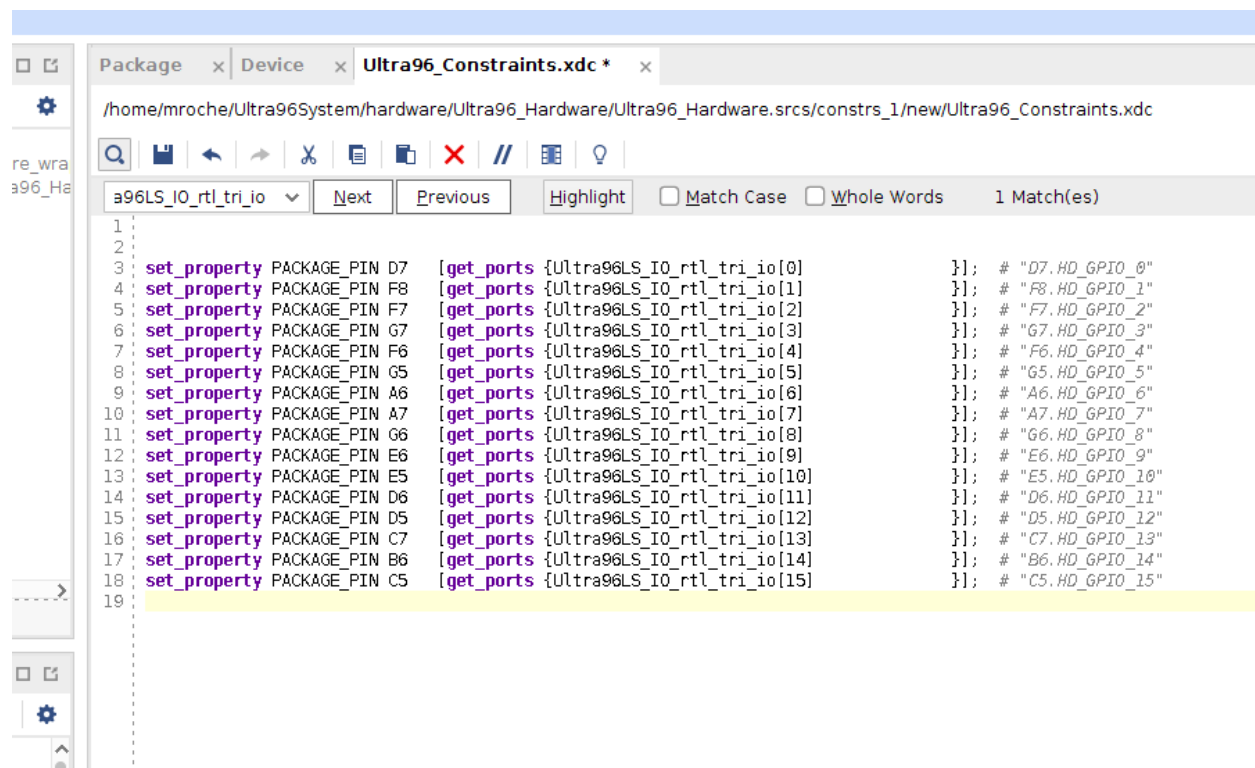
In the sources Window you should see constrs_1

Open this dropdown and there is your constraints file

Open the file by double clicking (see below for an alternate method).

A blank file opens

Enter the constraints for each of the correct IO's:



```
1
2
3 set_property PACKAGE_PIN D7 [get_ports {Ultra96LS_IO_rtl_tri_io[0] }]; # "D7.HD_GPIO_0"
4 set_property PACKAGE_PIN F8 [get_ports {Ultra96LS_IO_rtl_tri_io[1] }]; # "F8.HD_GPIO_1"
5 set_property PACKAGE_PIN F7 [get_ports {Ultra96LS_IO_rtl_tri_io[2] }]; # "F7.HD_GPIO_2"
6 set_property PACKAGE_PIN G7 [get_ports {Ultra96LS_IO_rtl_tri_io[3] }]; # "G7.HD_GPIO_3"
7 set_property PACKAGE_PIN F6 [get_ports {Ultra96LS_IO_rtl_tri_io[4] }]; # "F6.HD_GPIO_4"
8 set_property PACKAGE_PIN G5 [get_ports {Ultra96LS_IO_rtl_tri_io[5] }]; # "G5.HD_GPIO_5"
9 set_property PACKAGE_PIN A6 [get_ports {Ultra96LS_IO_rtl_tri_io[6] }]; # "A6.HD_GPIO_6"
10 set_property PACKAGE_PIN A7 [get_ports {Ultra96LS_IO_rtl_tri_io[7] }]; # "A7.HD_GPIO_7"
11 set_property PACKAGE_PIN G6 [get_ports {Ultra96LS_IO_rtl_tri_io[8] }]; # "G6.HD_GPIO_8"
12 set_property PACKAGE_PIN E6 [get_ports {Ultra96LS_IO_rtl_tri_io[9] }]; # "E6.HD_GPIO_9"
13 set_property PACKAGE_PIN E5 [get_ports {Ultra96LS_IO_rtl_tri_io[10] }]; # "E5.HD_GPIO_10"
14 set_property PACKAGE_PIN D6 [get_ports {Ultra96LS_IO_rtl_tri_io[11] }]; # "D6.HD_GPIO_11"
15 set_property PACKAGE_PIN D5 [get_ports {Ultra96LS_IO_rtl_tri_io[12] }]; # "D5.HD_GPIO_12"
16 set_property PACKAGE_PIN C7 [get_ports {Ultra96LS_IO_rtl_tri_io[13] }]; # "C7.HD_GPIO_13"
17 set_property PACKAGE_PIN B6 [get_ports {Ultra96LS_IO_rtl_tri_io[14] }]; # "B6.HD_GPIO_14"
18 set_property PACKAGE_PIN C5 [get_ports {Ultra96LS_IO_rtl_tri_io[15] }]; # "C5.HD_GPIO_15"
19
```

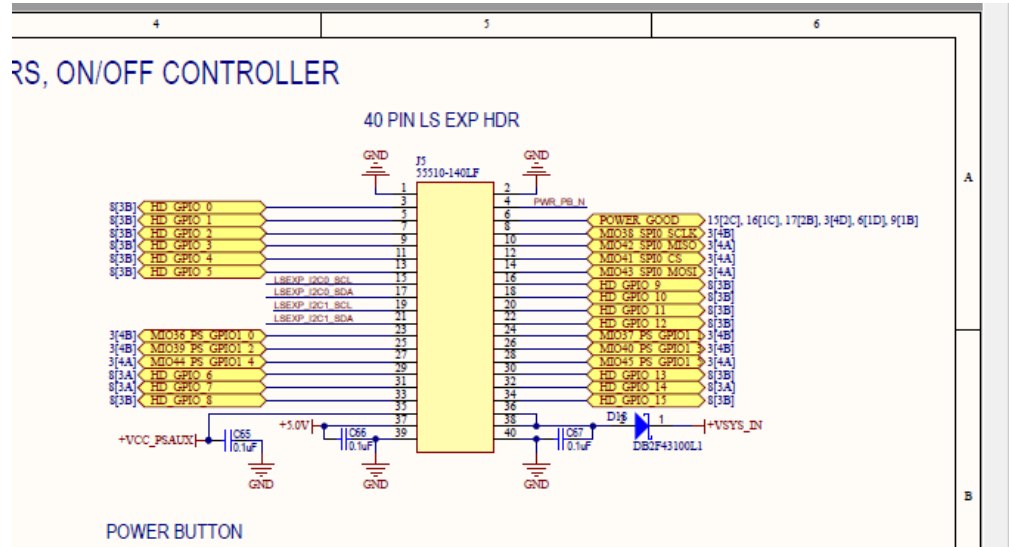
Save the file.

After creating the file open and edit it from your system with your preferred editor. This is a good exercise in locating and writing a constraints file.

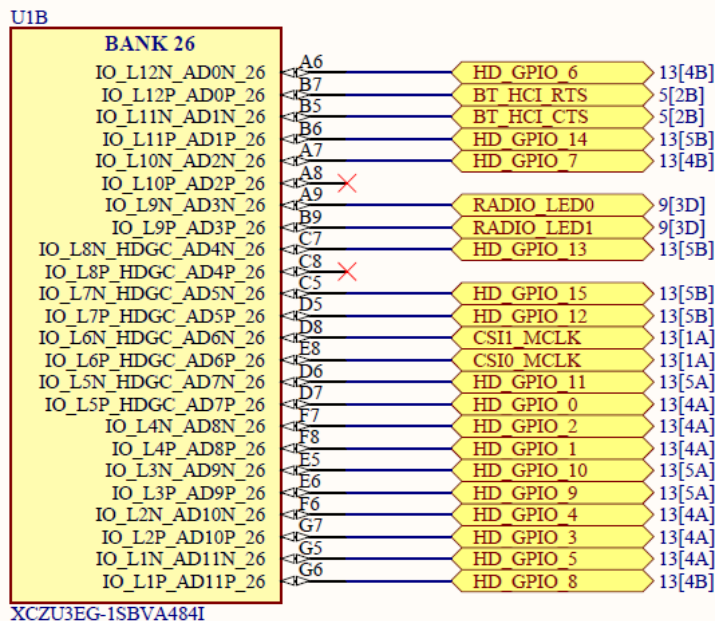
How Do I know these Mappings?

Look at the Schematic on Page 13. The Low Speed connector on top right shows a group of IO's called HD_GPIO_X

Search for these IO's to find which pin is it connected to the Zynq



Here they are on page 8 connected to the Zynq Bank 26 pins



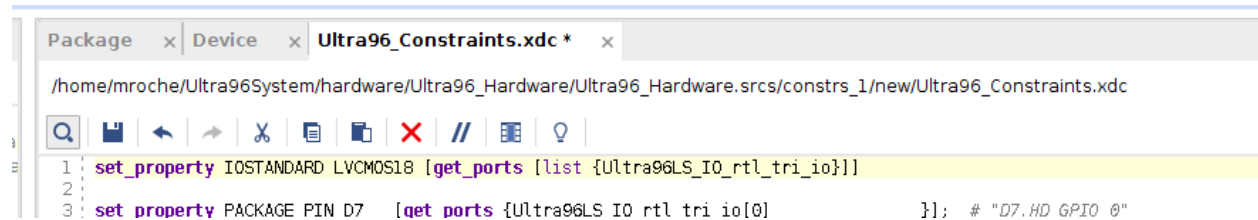
Notice that I have connected them to the pins C5, D5, D6, etc. using the constraints above.

That's it!! Now the pins are properly connected.

There is always a final step

There is one final step to do. When Configuring IP you must specify the I/O Standard you are using. is it 1.8v 1.2v CMOS. High Speed Low Speed etc.

If we look back at the I/O Ports window on the bottom of the screen we can see that Vivado has set the IO Standard to a Default value. A quirk in the tools is that Vivado won't let you create a bitstream unless you set these IO's to actual values. So even if it is the same value you are setting you must go through each one up to 15 and set it to LVCMOS18. The Short cut is to add another Tcl script to the Constraints file to automatically set these values:



```
Package x Device x Ultra96_Constraints.xdc * x
/home/mroche/Ultra96System/hardware/Ultra96_Hardware/Ultra96_Hardware.srcs/constrs_1/new/Ultra96_Constraints.xdc
1 set_property IOSTANDARD LVCMOS18 [get_ports [list {Ultra96LS_IO_rtl_tri_io}]]
2
3 set_property PACKAGE_PIN D7 [get_ports {Ultra96LS_IO_rtl_tri_io[0]}]; # "D7.HD GPIO 0"
```

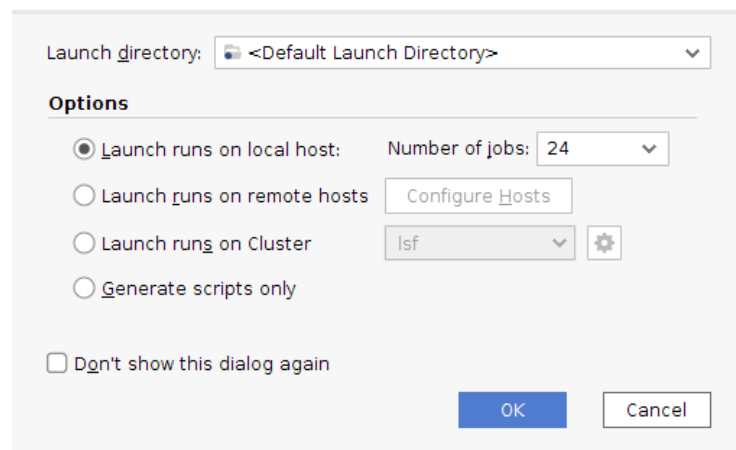
When all of this is done you now must run the tool chain for the new constraints and this time generate the bitstream.

Save the constraints file if you forgot to.

On the Left side of the window select "Generate Bitstream". Click "Yes" to Synthesis is Out-of-date.

Once again allocate all cores on your development machine for the build

Launch the selected synthesis or implementation runs.



Launch directory: <Default Launch Directory>

Options

☒ Launch runs on local host: Number of jobs: 24

☐ Launch runs on remote hosts: Configure Hosts

☐ Launch runs on Cluster: Isf

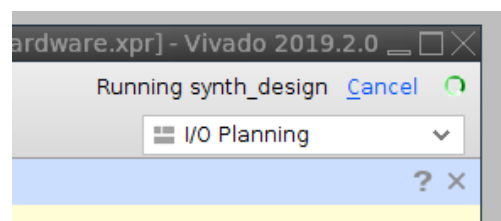
☐ Generate scripts only

☐ Don't show this dialog again

OK Cancel

Press OK

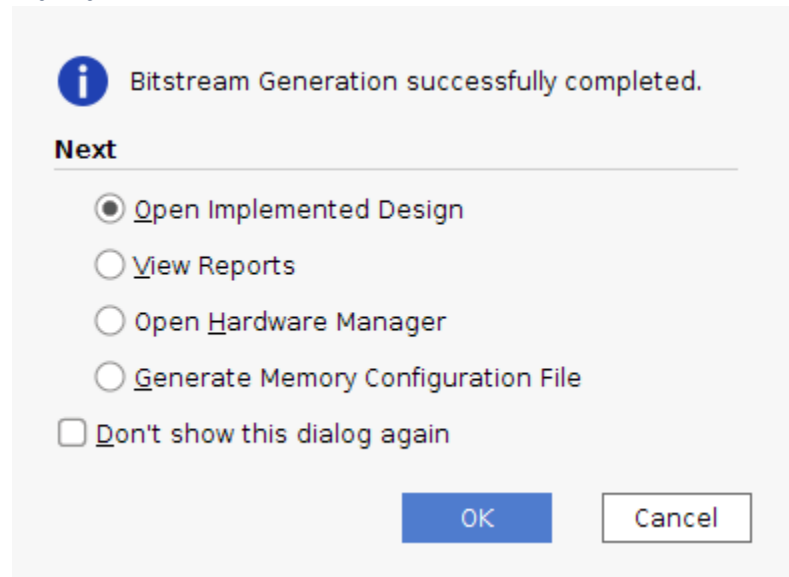
Go for another coffee while the design reruns. It will be quicker this time as it does not have to go all the way through the workflow.



SEGWAY: While we wait:

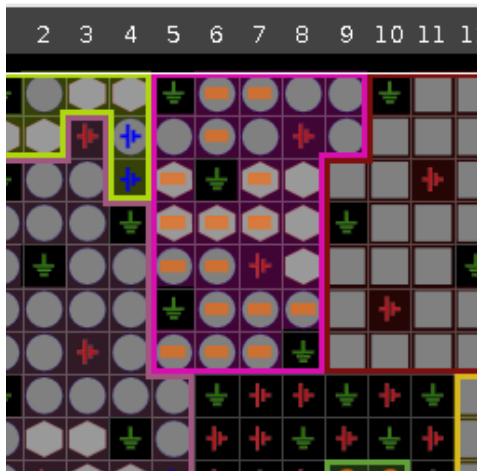
And that's it for the hardware setup!!! Everything is done and all we need now is to export the design to the Software IDE and build the software platform. In the meantime, I would recommend you play around with the different steps on the left-hand side as they become available. They give you more details about your design like device and resource utilization, timing closure, power estimates detailed floor planning IO mappings and other views on the system. This can be where to go next if you want. A lot of it is not need-to-know if you are a software engineer but it is always good to understand the machine for which you are developing.

Huzzah!!



Press OK

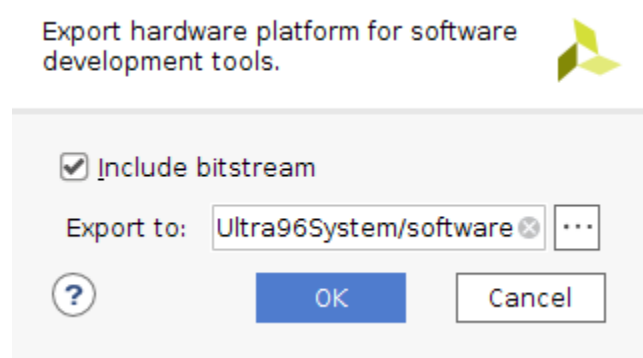
Look at the package window. Hover your mouse over the area with the orange bars at the top of the diagram. You will see your IO is mapped to these pins!!



Export your design.

In Vivado Select Menu File->Export→Export Hardware

Fill in the dialog as follows:



(Make sure you check the include bitstream box)

Files are exported to the famous .hdf file or the new xsa file which we see here. This file contains all details of the hardware design and its interface to the software

```
XIRIROCHE40:/home/mroche/Ultra96System/software $ ls
Ultra96_Hardware_wrapper.xsa
XIRIROCHE40:/home/mroche/Ultra96System/software $
```

We are not interested in Vivado anymore. Let's get some real work done. Software:-)