California State University Northridge

ECE 520 - Spring 2022

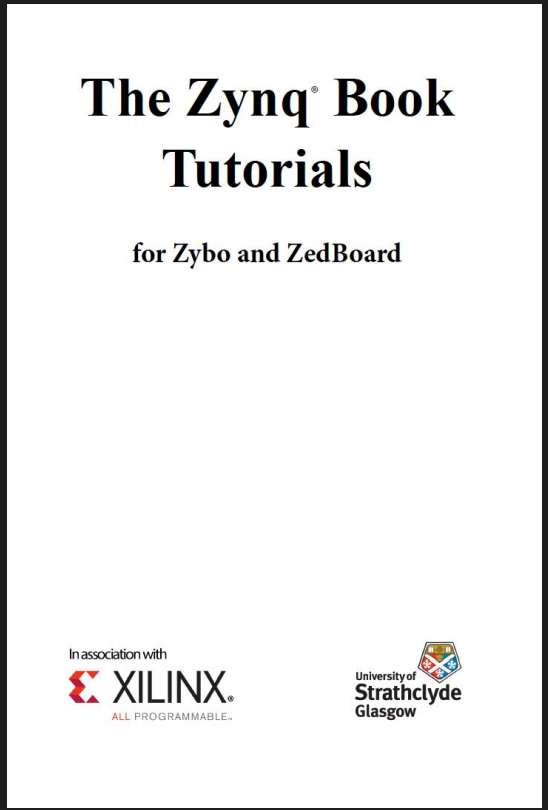
First Design on Zynq

Saba Janamian

Note

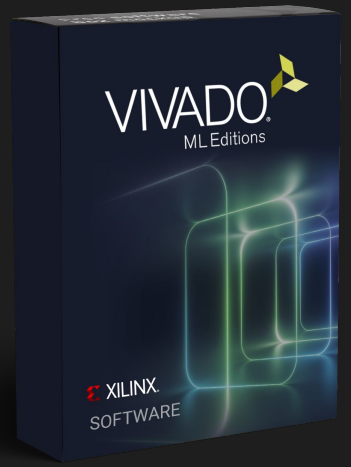
All the content in this slide had been taken from “The Zynq Book Tutorial” by University of Strathclyde Glasgow.

Since the original tutorial had been written for Vivado 2015 and Xilinx SDK, some of the content had been updated to cover change in the Vivado 2022 and Vitis IDE.



Introduction

This tutorial will guide you through the process of creating a first Zynq design using the Vivado Integrated Development Environment (IDE), and introduce the IP Integrator environment for the generation of a simple Zynq processor design to be implemented on the Zybo or ZedBoard.



Introduction

The Vitis development environment will then be used to create a simple software application which will run on the Zynq's ARM Processing System (PS) to control the hardware that is implemented in the Programmable Logic (PL).

Three parts of the lab

The tutorial is split into three exercises, and is organized as follows:

Exercise 1A - This exercise will guide you through the process of launching Vivado IDE and creating a project for the first time. The various stages of the New Project Wizard will be introduced.

Three parts of the lab

The tutorial is split into three exercises.

Part 1

Exercise 1A - This exercise will guide you through the process of launching Vivado IDE and creating a project for the first time. The various stages of the New Project Wizard will be introduced.

Part 2

Exercise 1B - In this exercise, we will use the project that was created in Exercise 1A to build a simple Zynq embedded system with the graphical tool, IP Integrator, and incorporating existing IP from the Vivado IP Catalog.

A number of design aids will be used throughout this exercise, such as the Board Automation feature which automates the customisation of IP modules for a specified device or board; in this case we will be using wither Zybo or ZedBoard Evaluation and Development Kit.

The Designer Assistance feature, which assists with the connections between the Zynq PS and the IP modules in the PL will also be demonstrated.

Once the design is finished, a number of stages will be undertaken to complete the hardware system and generate a bitstream for implementation in the PL.

The completed hardware design will then be exported to the Vitis Software development environment for the development of a simple software application in Exercise 1C.

Part 3

Exercise 1C - In this short third exercise, the Vitis will be introduced, and a short software application will be created to allow the Zynq processor to interact with the IP implemented in the PL.

A connection to the hardware server that allows the Vits to communicate with the Zynq processors will be established.

The software drivers that are automatically created by the Vivado IDE for IP modules will be explored and integrated into the software application, before finally building and executing the software application on the development board.

Note on project address

Throughout all of the practical tutorial exercise we will be using linux address paths in the home directory of the author as the working directory.

Since it is unlikely that the address will be suitable for all users, you can substitute it for a directory of your choice, but you should be aware that you will be required to make alterations to some source files (if any) in order for exercises to complete successfully.

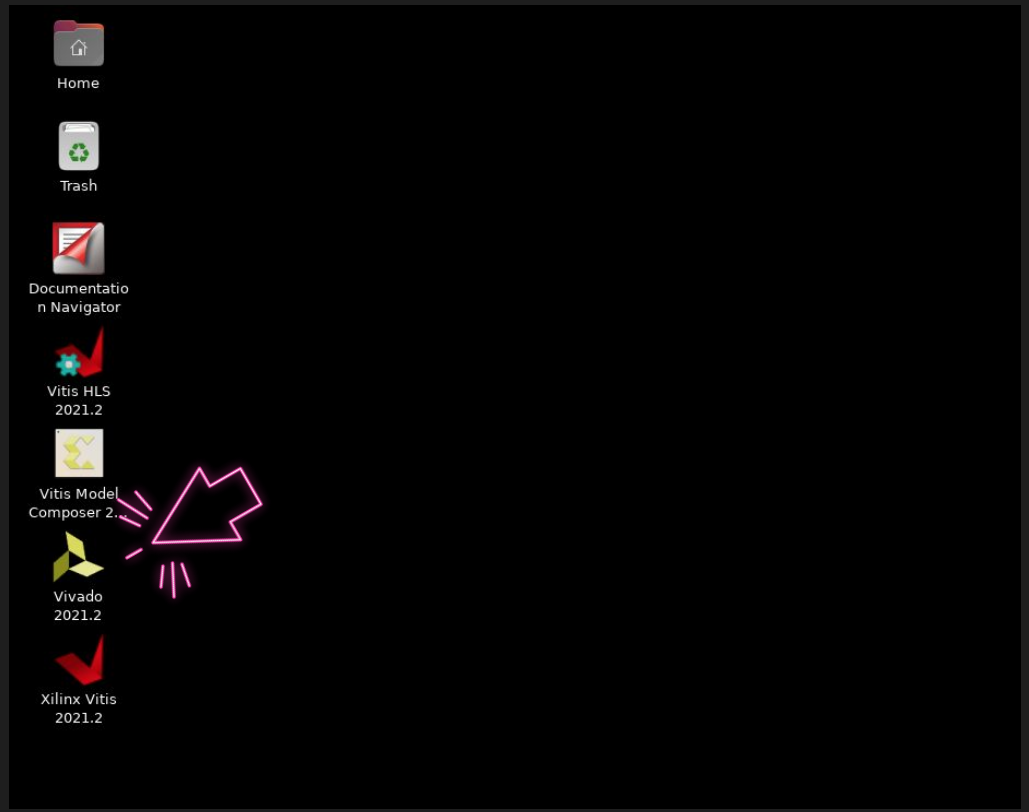
Exercise 1A
Creating a First IP Integrator Design

Part 1 objective

In this exercise we will create a new project in Vivado IDE by moving through the stages of the Vivado IDE New Project Wizard.

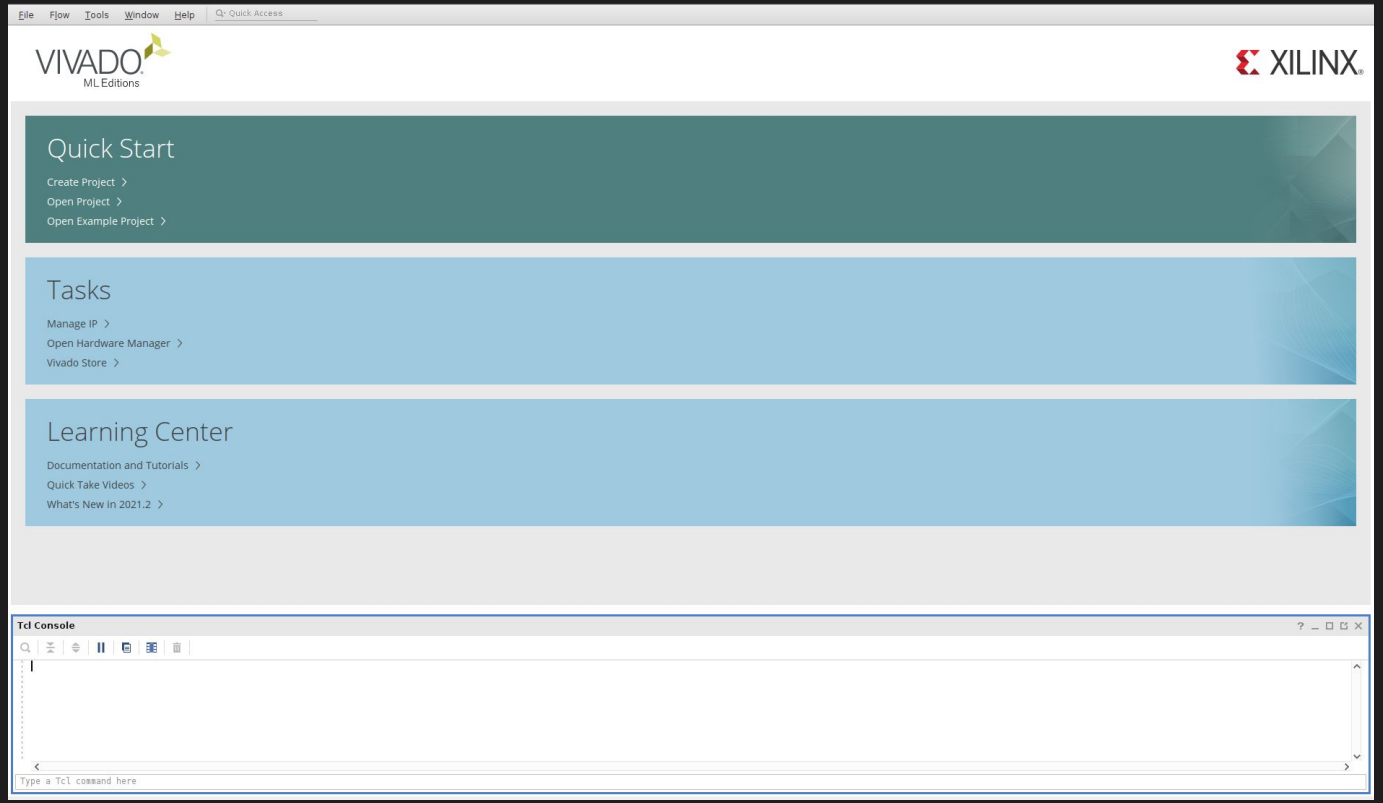
Starting Vivado

Open Vivado via the start menu or desktop shortcut.



The Start Page

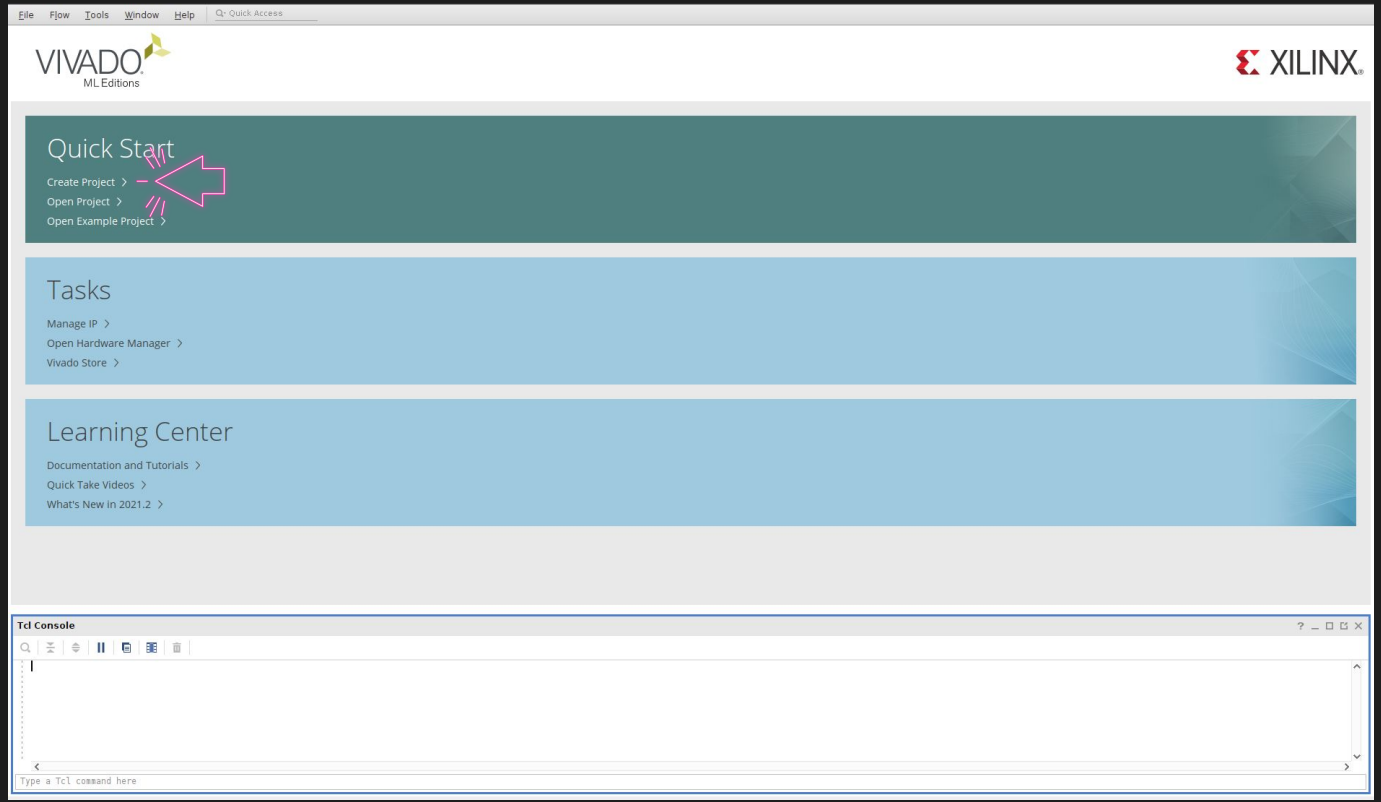
This is the screen Vivado's start-up screen.



The Start Page

Create New Project:

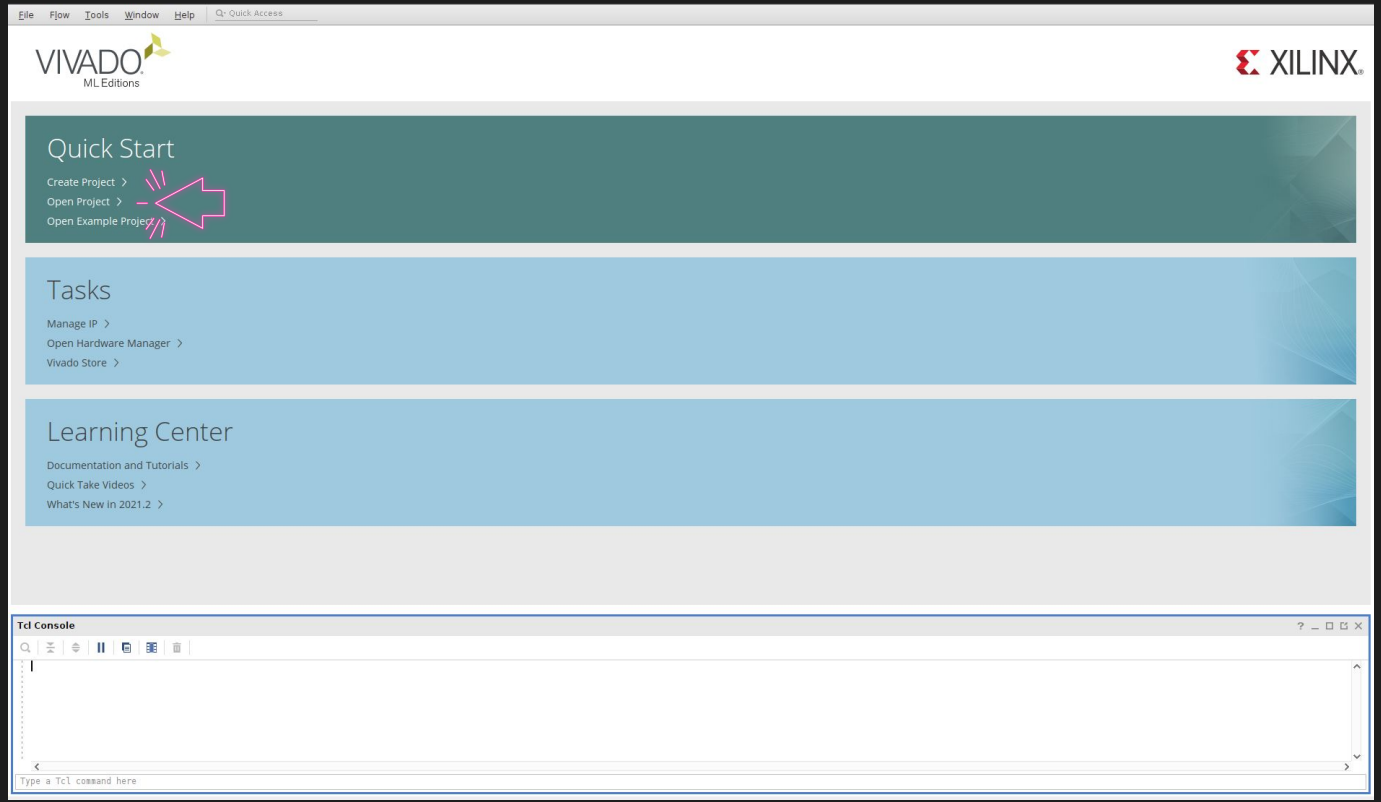
This button will open the New Project wizard. This wizard steps the user through creating a new project.



The Start Page

Open Project:

This button will open a file browser. Navigate to the desired Xilinx Project (.xpr) file and click Open to open the project in Vivado.

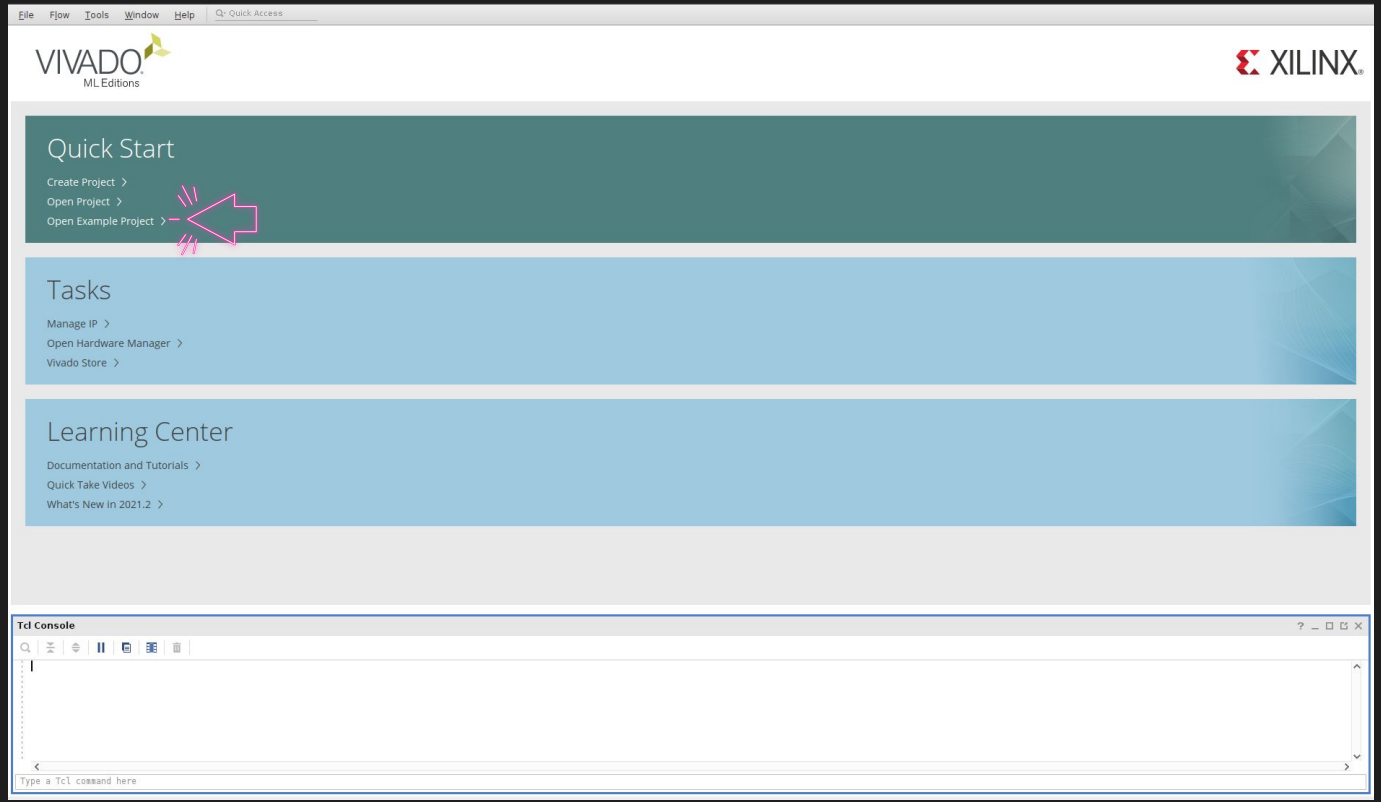


The Start Page

Open Example Project:

This will guide the user through creating a new project based on an example project. These projects will not work on all devices. Many Digilent example projects are instead released on Github, and linked to through the target FPGA System Board's Resource Center, which can be found through the list of

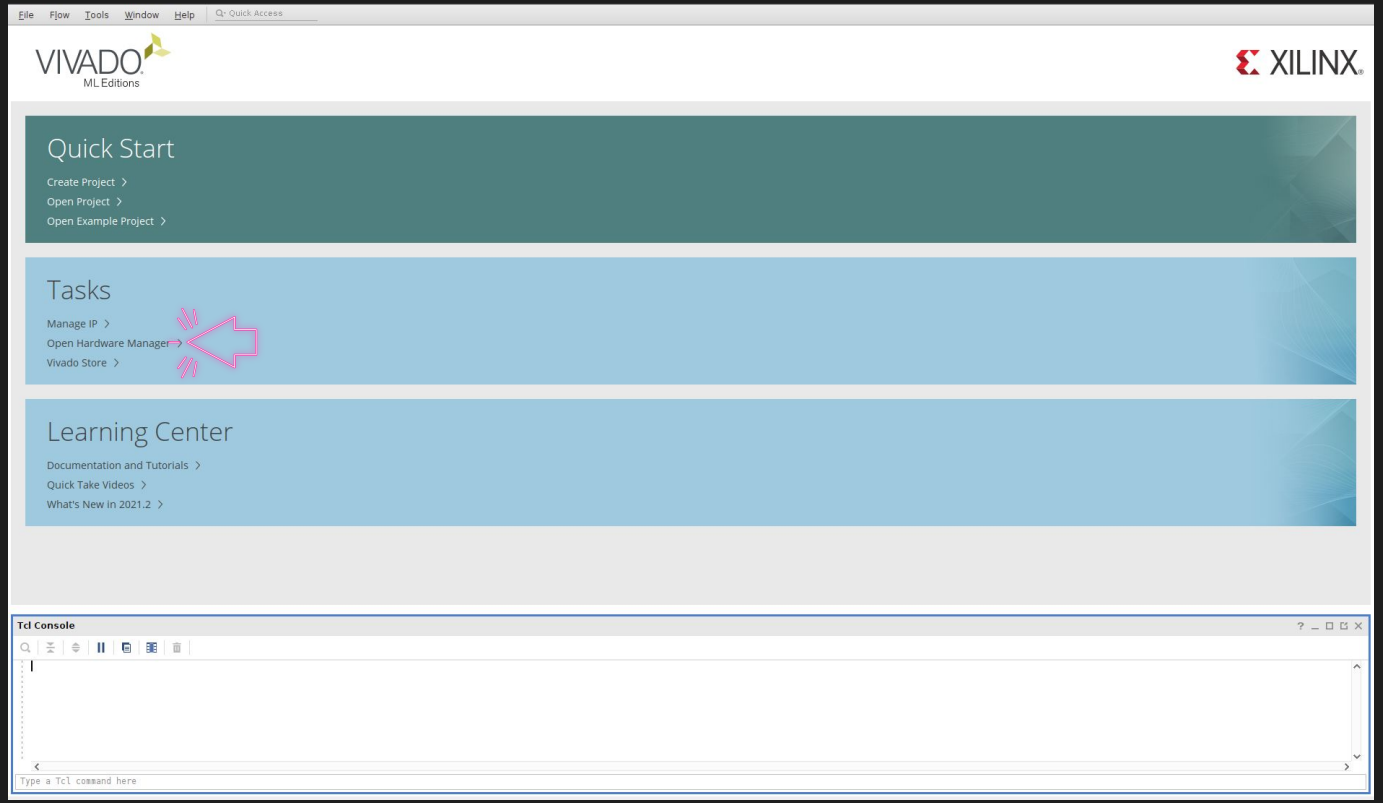
https://digilent.com/shop/boards-and-components/system-boards/fpga-boards/



The Start Page

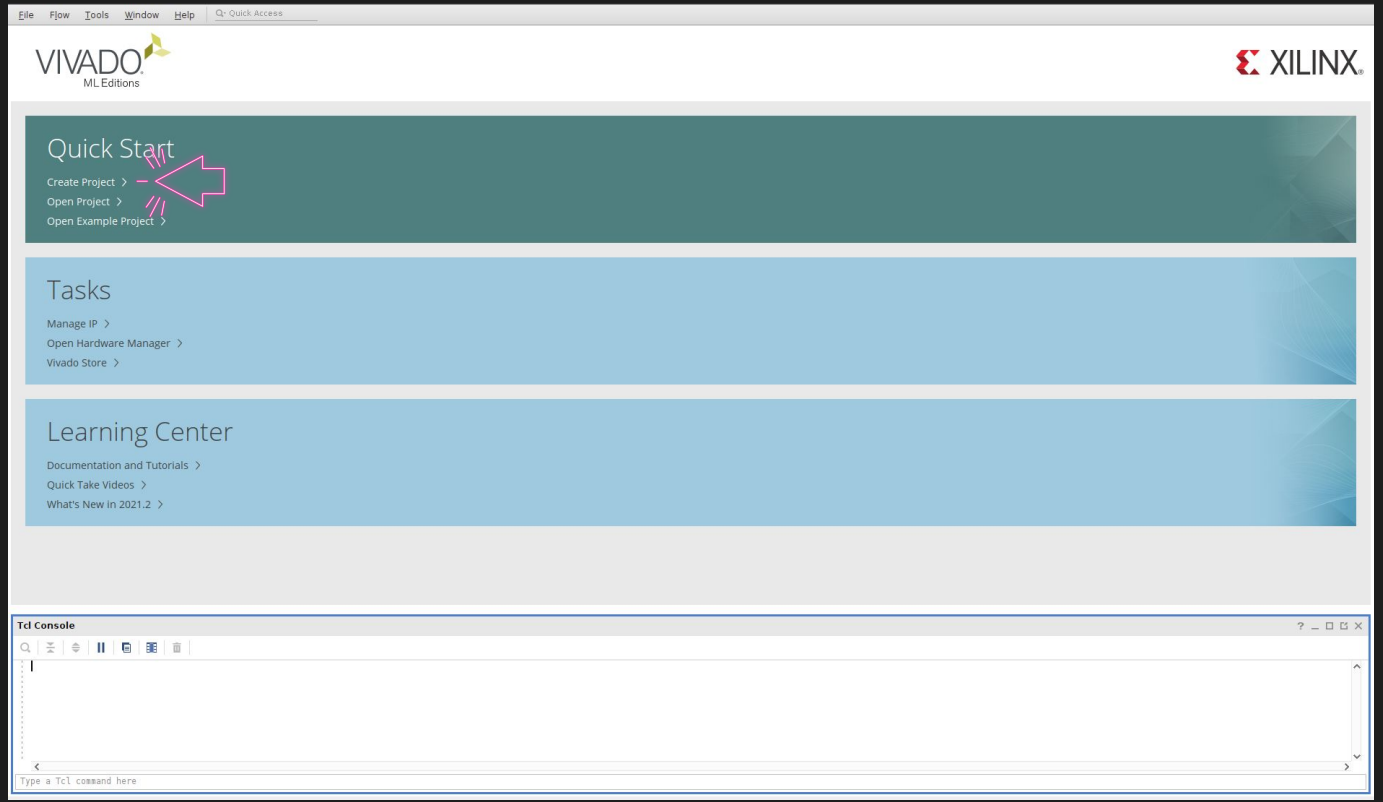
Open Hardware Manager:

This will open the Hardware Manager without an associated project. If connecting to and programming a device is all that the user wants to do, then this is the button to use.



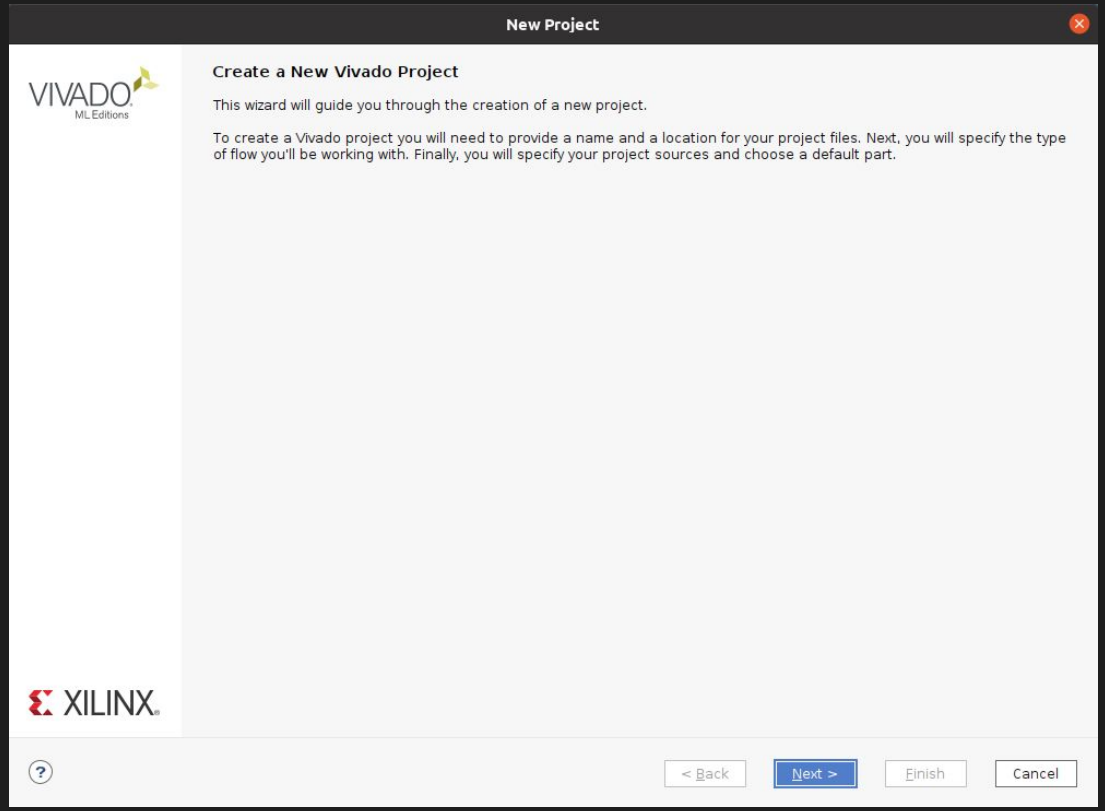
Creating a New Project

From the start page, click the Create New Project button to start the New Project Wizard.



Creating a New Project Wizard

The text in this dialog describes the steps that will be taken to create a project. Click Next to continue.



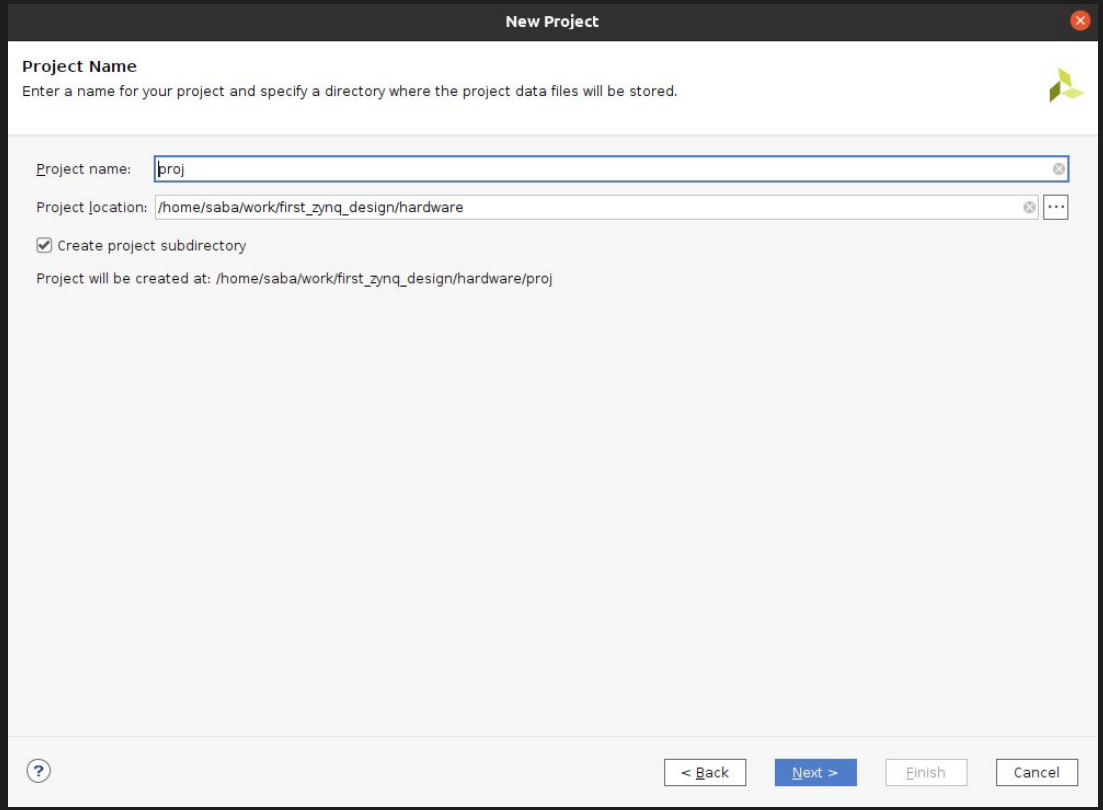
Creating a New Project - Project Name

The first page is used to set the name of the project.

Vivado will use this name when generating its folder structure.

Do NOT use spaces in the project name or location path.

This will cause problems with Vivado.

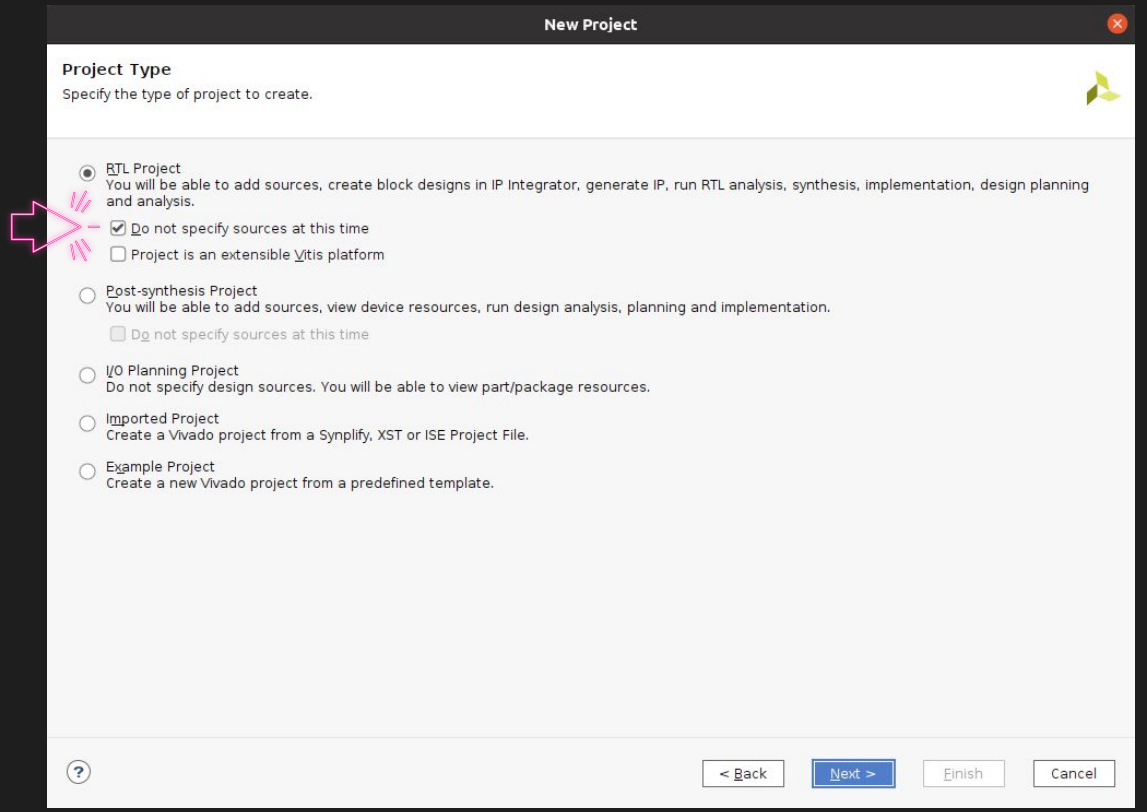


Creating a New Project Wizard - Project Type

Now that the project has a name and a place to save its files we need to select the type of project we will be creating.

Select RTL Project and make sure to check the Do not specify sources at this time box.

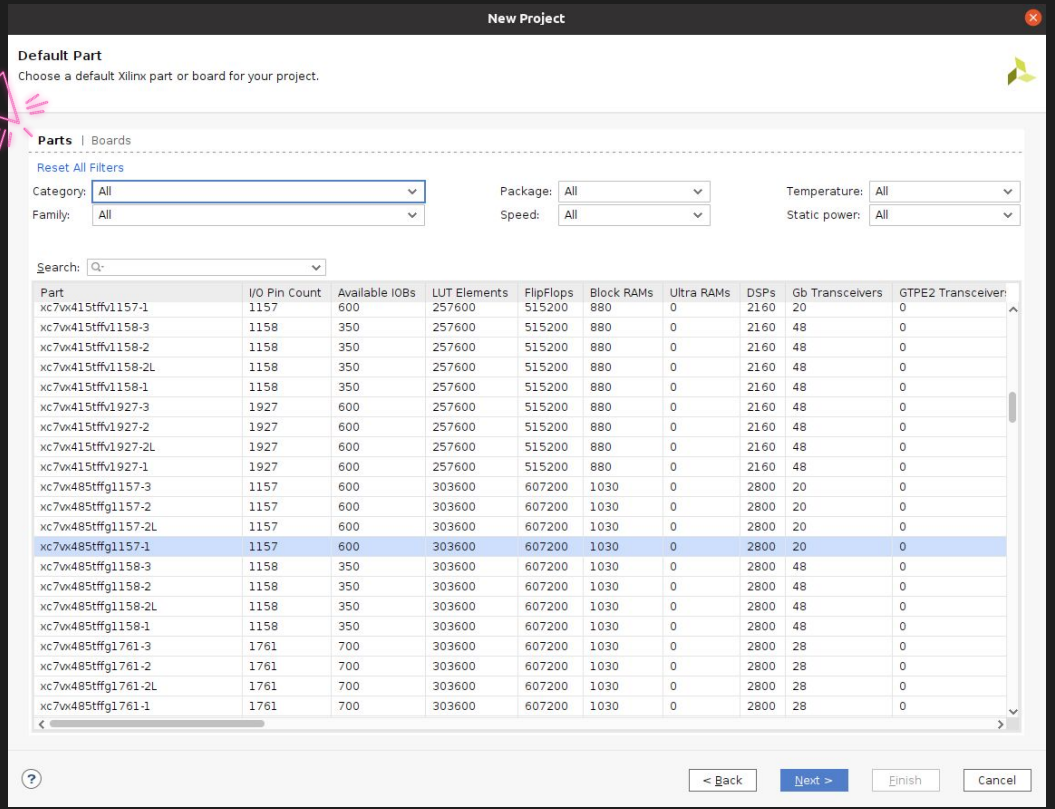
Source files will be added and created after the project has been created.



Creating a New Project Wizard - Default Part

The Default part window allows you to specify the hardware part that you need to create a project for.

If the board that you are working does not have a board support package you need to select the target chip number from the list.

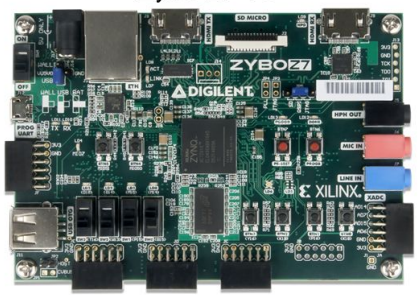


Creating a New Project Wizard - Default Part

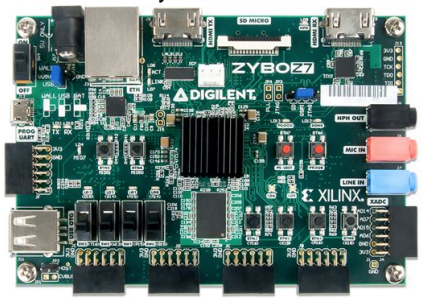
For example Zybo Z7 boards uses the following chips

Zybo Z7-10: XC7Z010-1CLG400C
Zybo Z7-20: XC7Z020-1CLG400C

Zybo Z7-10



Zybo Z7-20



The primary difference between the two variants is the size of the FPGA inside the Zynq AP SoC. The Zynq processors both have the same capabilities, but the -20 has about a 3 times larger internal FPGA than the -10. Also, The Zynq-7010 has slightly fewer FPGA attached pins than the Zynq-7020, which means several features found on the Zybo Z7-20 are not available on the Zybo Z7-10. The differences between the two variants are summarized below:

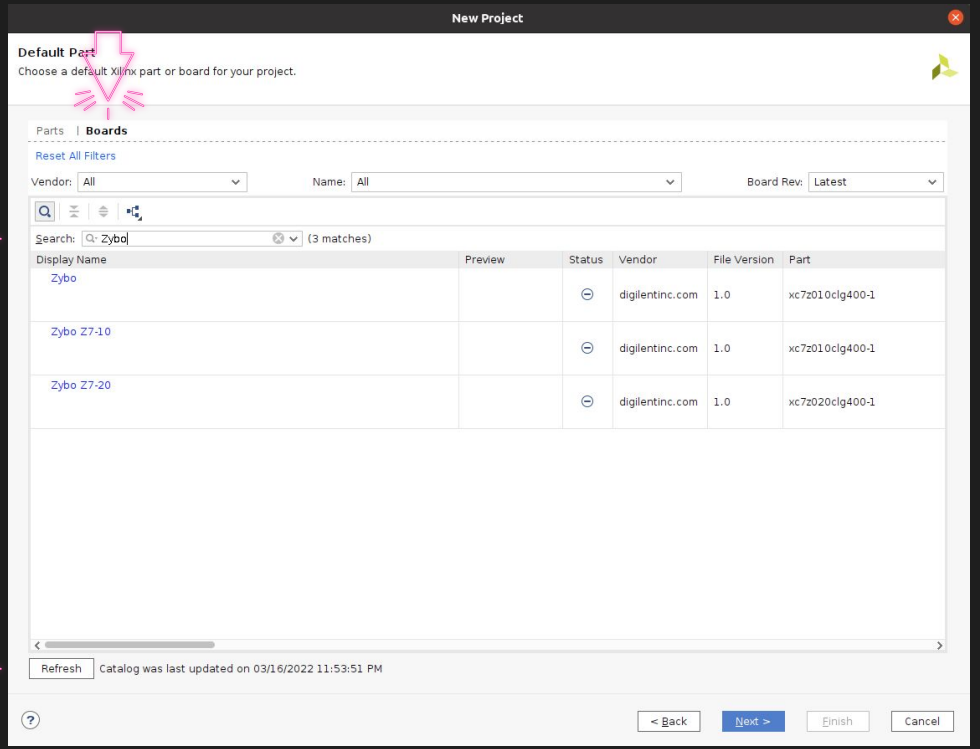
Product Variant	Zybo Z7-10	Zybo Z7-20
Zynq Part	XC7Z010-1CLG400C	XC7Z020-1CLG400C
1 MSPS On-chip ADC	Yes	Yes
Look-up Tables (LUTs)	17,600	53,200
Flip-Flops	35,200	106,400
Block RAM	270 KB	630 KB
Clock Management Tiles	2	4
Total Pmod ports	5	6
Fan connector	No	Yes
Zynq heat sink	No	Yes
HDMI CEC Support	TX port only	TX and RX ports
RGB LED	1	2

Creating a New Project Wizard - Default Part

If the board has a board support package you can select the board from the Boards tab.

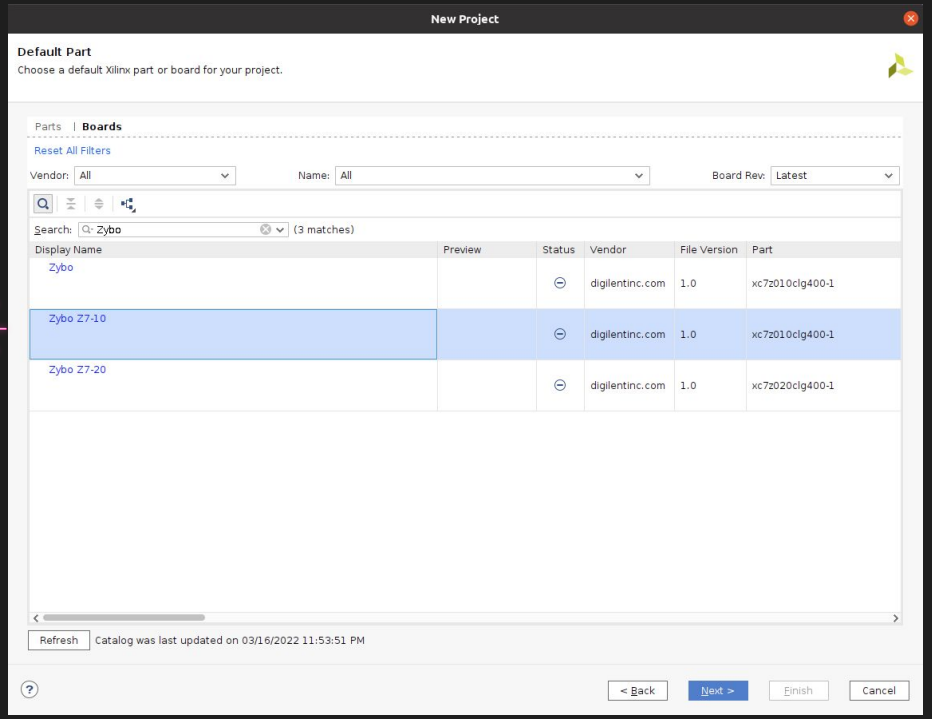
Search for the board name in the search box.

If the board support package is not installed click the Refresh button at the bottom of the menu to install the packages.



Creating a New Project Wizard - Default Part

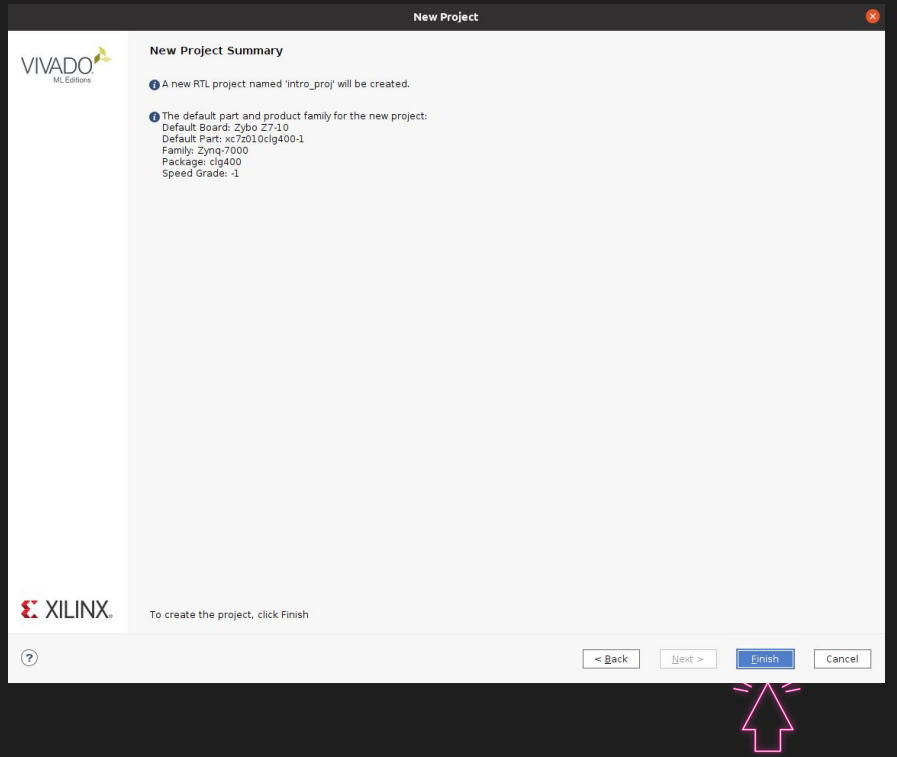
Select the
desired board
and press next.



Creating a New Project Wizard - Project Summary

Last window shows the new project summary.

Press Finish to close the project create wizard.



Using Tcl to create a project

```
create_project proj /home/saba/work/first_zynq_design/hardware/proj -part xc7z010clg400-1
```

Project name

Project address

Project part

```
set_property board_part digilentinc.com:zybo-z7-10:part0:1.1 [current_project]
```

Board support package

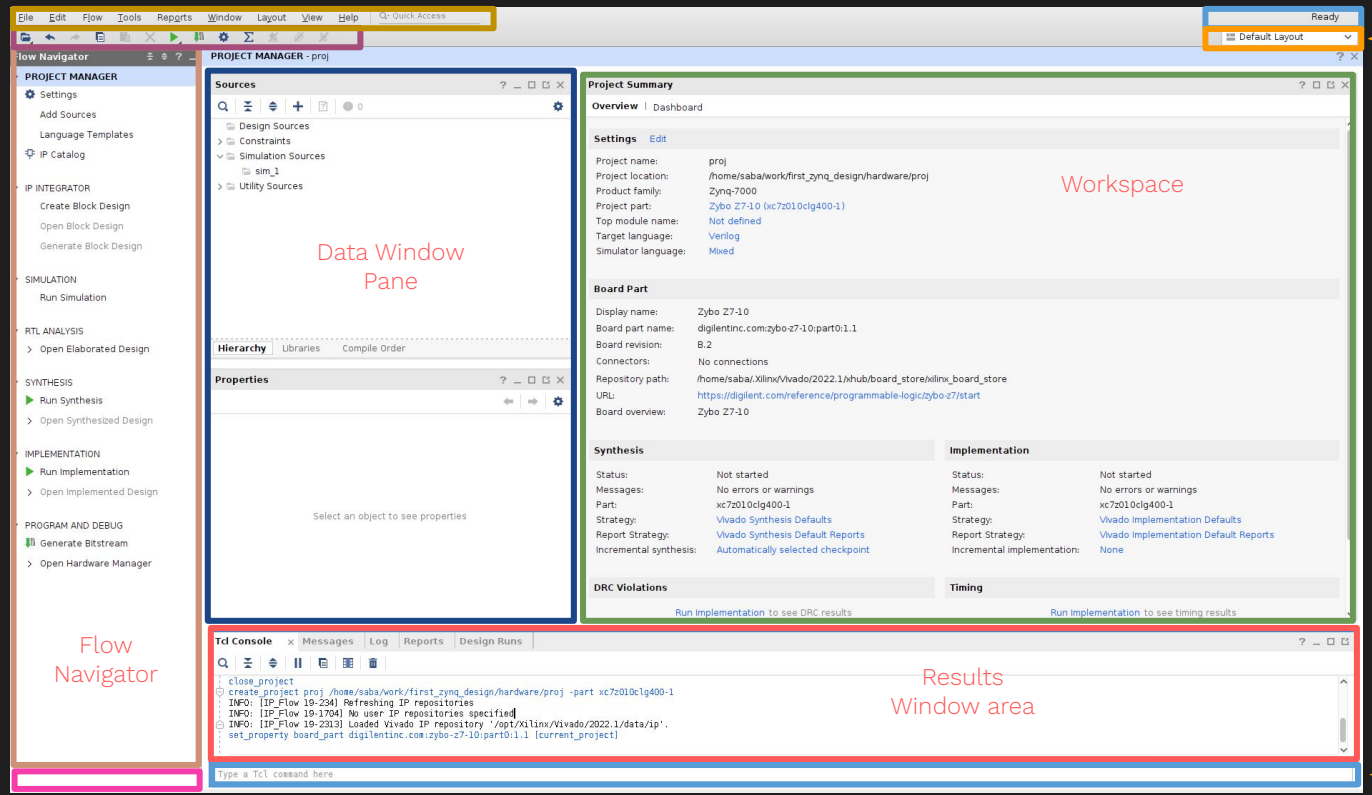
Apply to current project

Getting to know Vivado main layout

Now that we have created our first project in Vivado IDE, we can now move on to creating our first Zynq embedded system design.

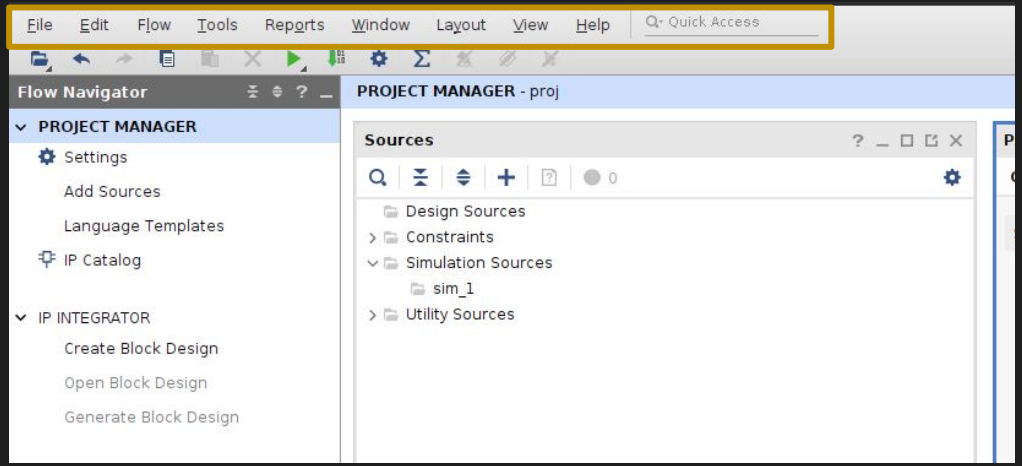
Before doing that, the Vivado IDE tool layout should be introduced. The default Vivado IDE environment layout is shown in Figure (other layouts can be chosen by selecting different perspectives).

Vivado default layout



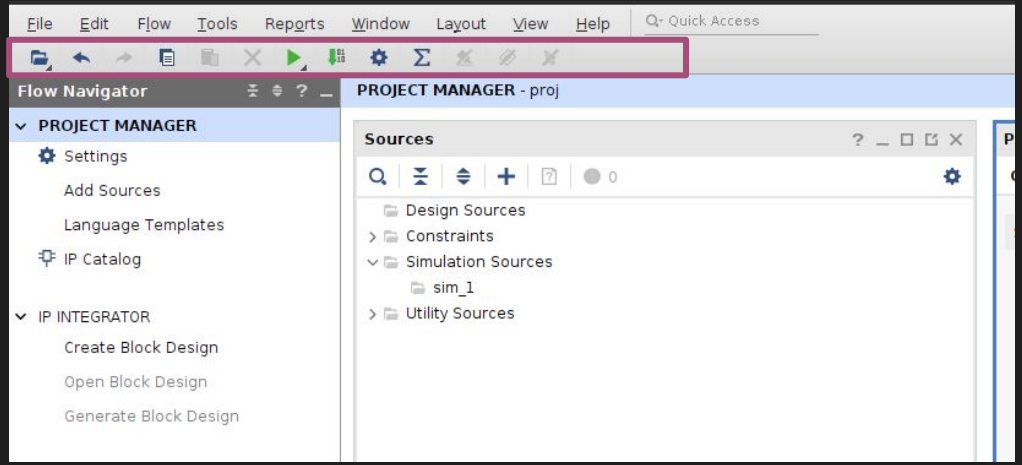
Menu Bar

The main access bar gives access to the Vivado IDE commands.



Main Toolbar

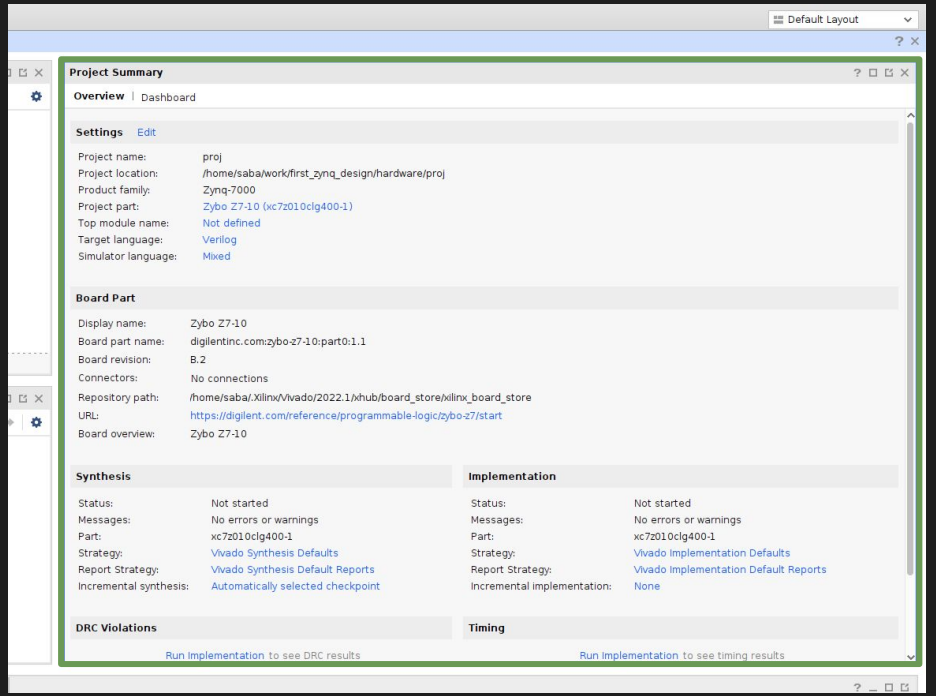
The main toolbar provides easy access to the most commonly used Vivado IDE commands. Tooltips that provide information for each command on the toolbar can be accessed by hovering the mouse pointer over the corresponding button



Workspace

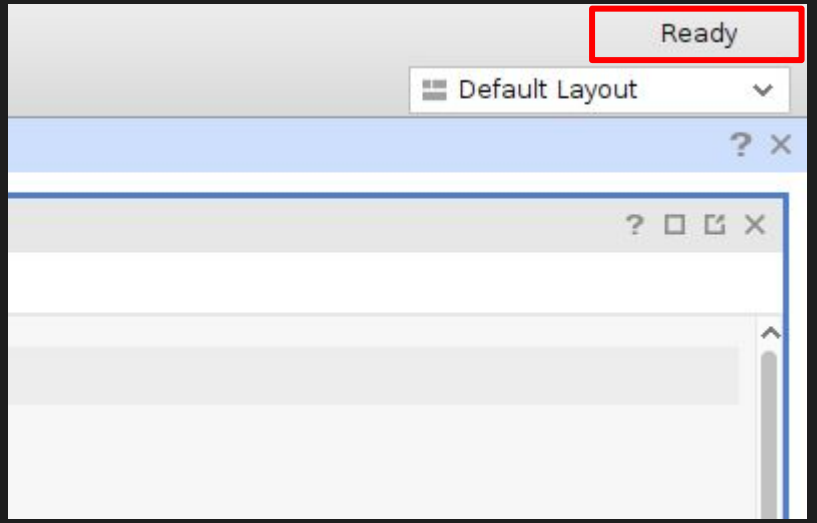
The workspace provides a larger area for panels which require a greater screen space and those with a graphical interface, such as:

- Schematic panel
- Device panel
- Package panel
- Text editor panel



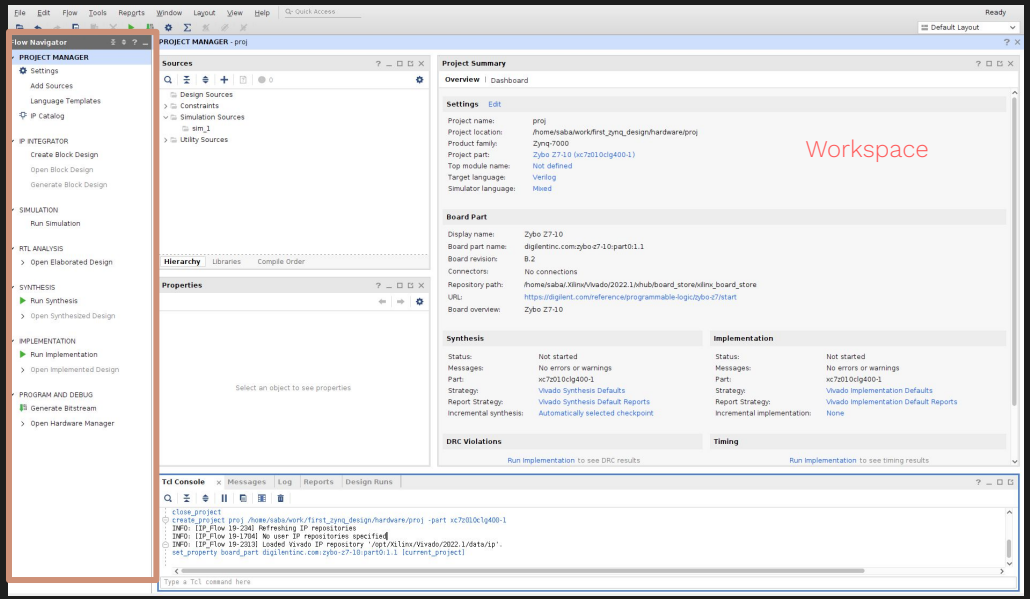
Project Status Bar

The project status bar displays the status of the currently active design.



Flow Navigator

The Flow Navigator provides easy access to the tools and commands that are necessary to guide your design from start to finish.

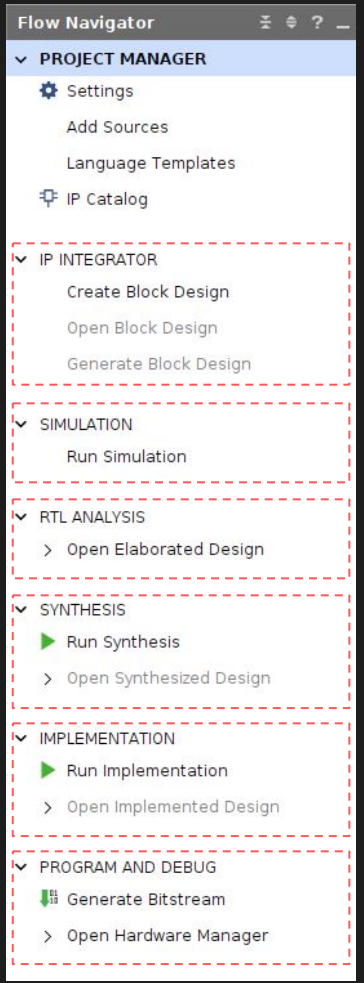


Flow Navigator

The Flow Navigator is the most important pane of the main Vivado window to know. It is how a user navigates between different Vivado tools.

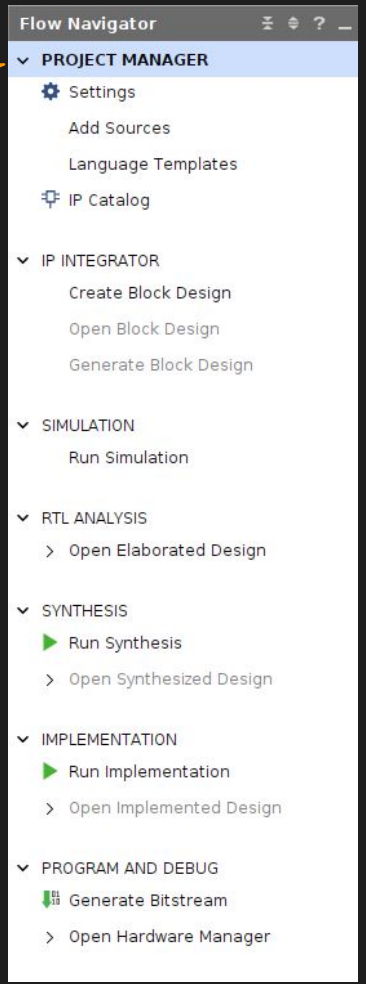
The Navigator is broken into seven sections:

- IP Integrator
- Simulation
- RTL Analysis
- Synthesis
- Implementation
- Program and Debug



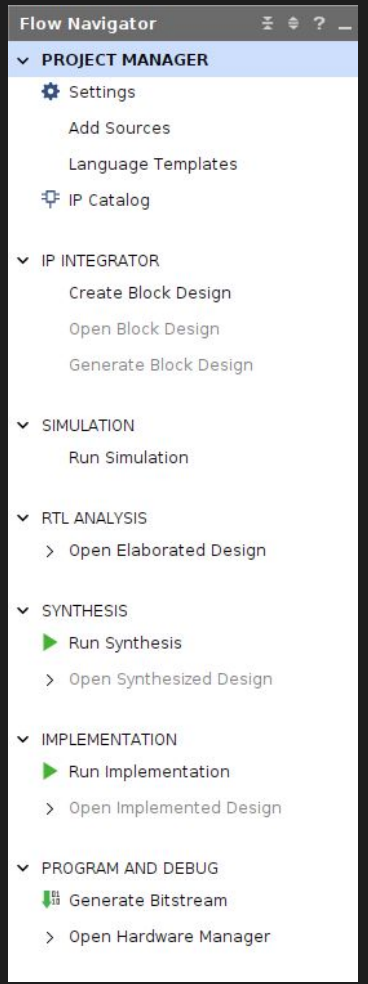
Project Manager Menu

Project Manager: Allows for quick access to project settings, adding sources, language templates, and the IP catalog.



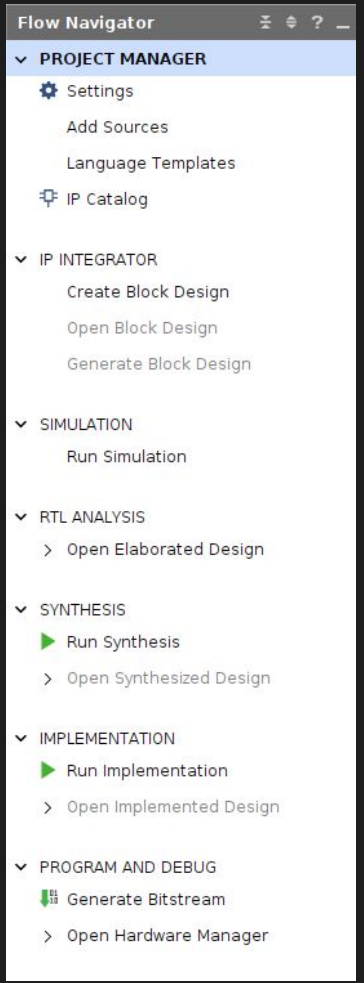
IP Integrator Menu

IP Integrator: Tools for creating complex designs in a graphical user interface, rather than by writing large source files.



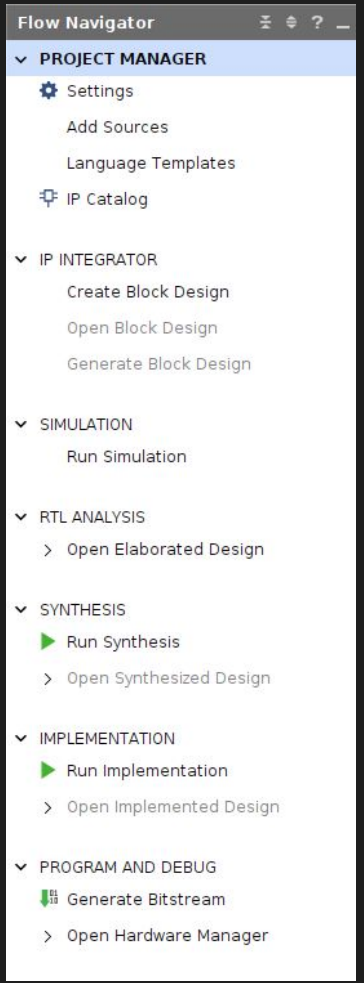
Simulation Menu

Simulation: Allows a developer to verify the output of their design prior to programming the target device.



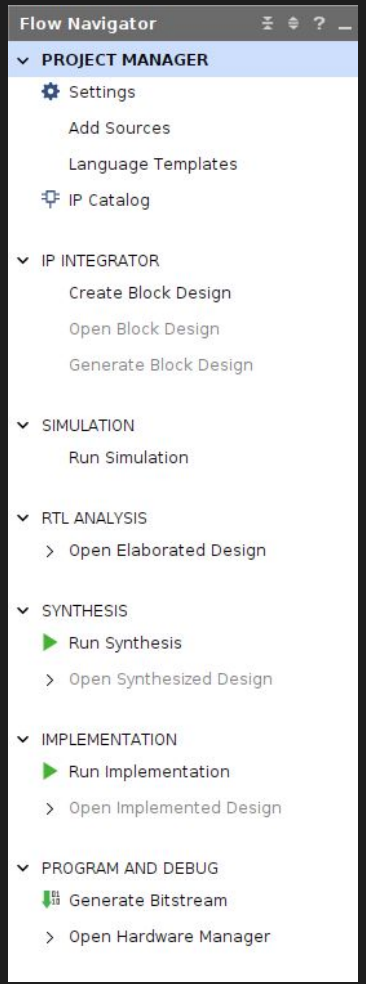
Flow Navigator

RTL Analysis: Lets the developer see how the tools are interpreting their code.

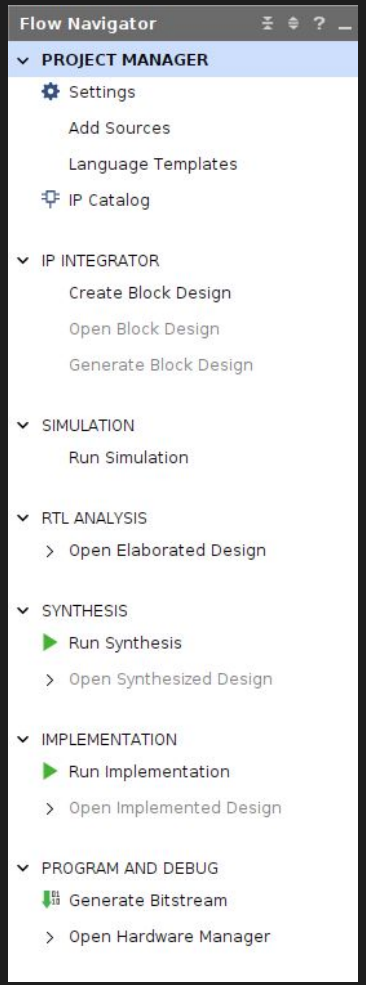


Synthesis Menu

Synthesis: Gives access to Synthesis settings and post-synthesis reports.

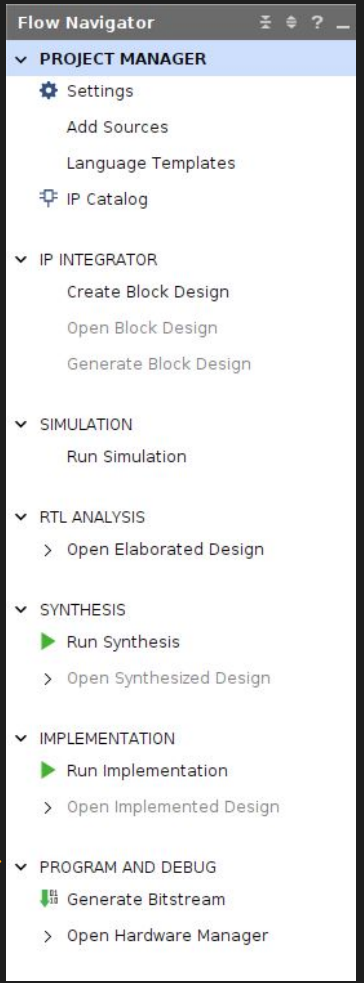


Implementation Menu



Implementation: Gives access to Implementation settings and post-implementation reports.

Program and Debug Menu



Program and Debug: Gives access to settings for bitstream generation and tools to program FPGAs.

Data Window Pane

The Data Windows pane, by default, displays information that relates to design data and sources, including:

Properties window: Shows information about selected logic objects or device resources.

Netlist window: Provides a hierarchical view of the synthesised or elaborated logic design.

Sources window: Shows IP Sources, Hierarchy, Libraries and Compile Order views.

Status bar

The status bar displays a variety of information, including:

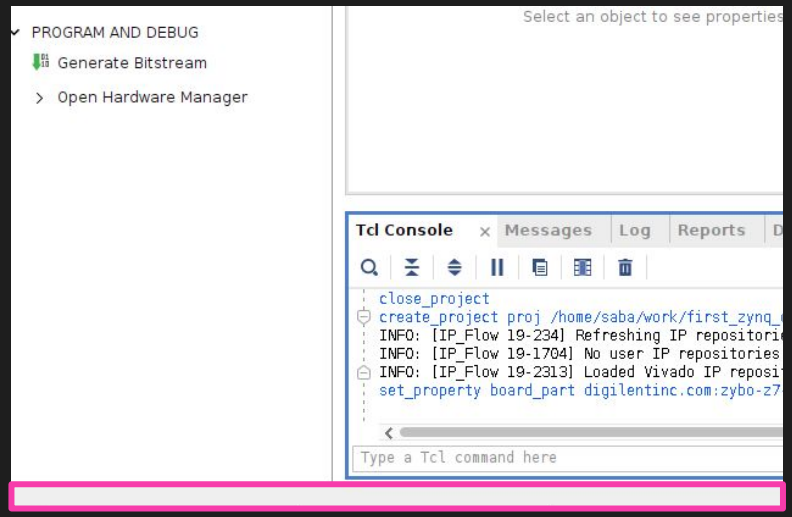
Detailed information regarding menu bar and toolbar commands will be shown in the lower left side of the status bar when the command is accessed.

When hovering over an object in the Schematic window with the mouse pointer, the object details appear in the status bar.

During constraint and placement creation in the Device and Package windows, validity and constraint type will be shown on the left side of the status bar.

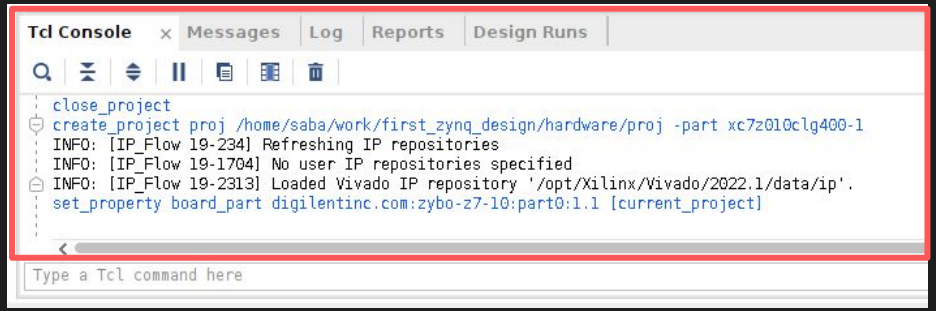
Site coordinates and type will be shown in the right side.

The task progress of a running task will be relocated to the right side of the status bar when the Background button is selected.



Results Window Area

The results window contains a variety of different tabs that show different information about the state of the project.

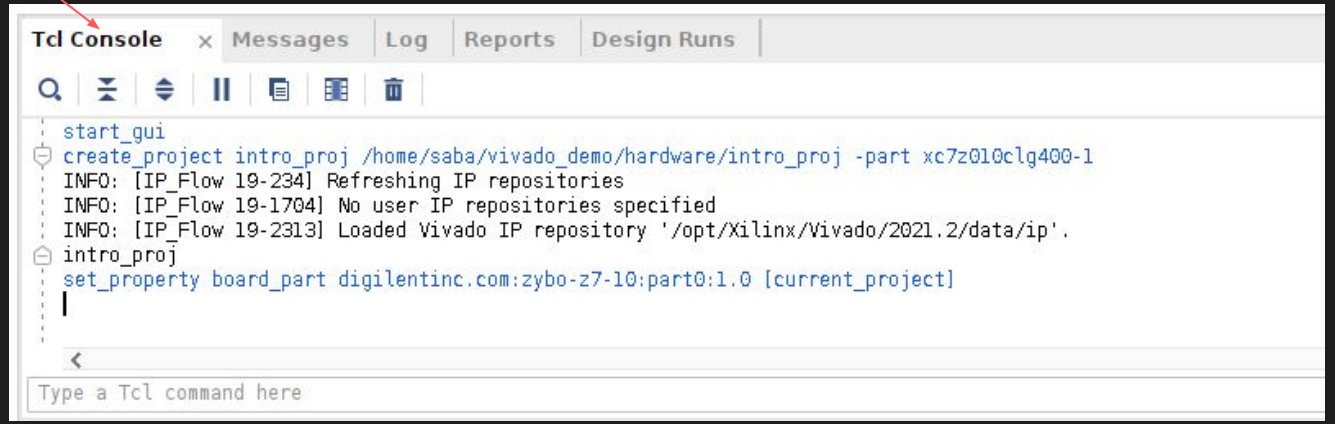


Tcl Console

The Tcl Console is a tool that allows running commands directly without the use of the main graphical user interface.

Note: all the actions done in the GUI correspond to one or more tcl commands.

You can automate your job by reusing the generated Tcl commands!

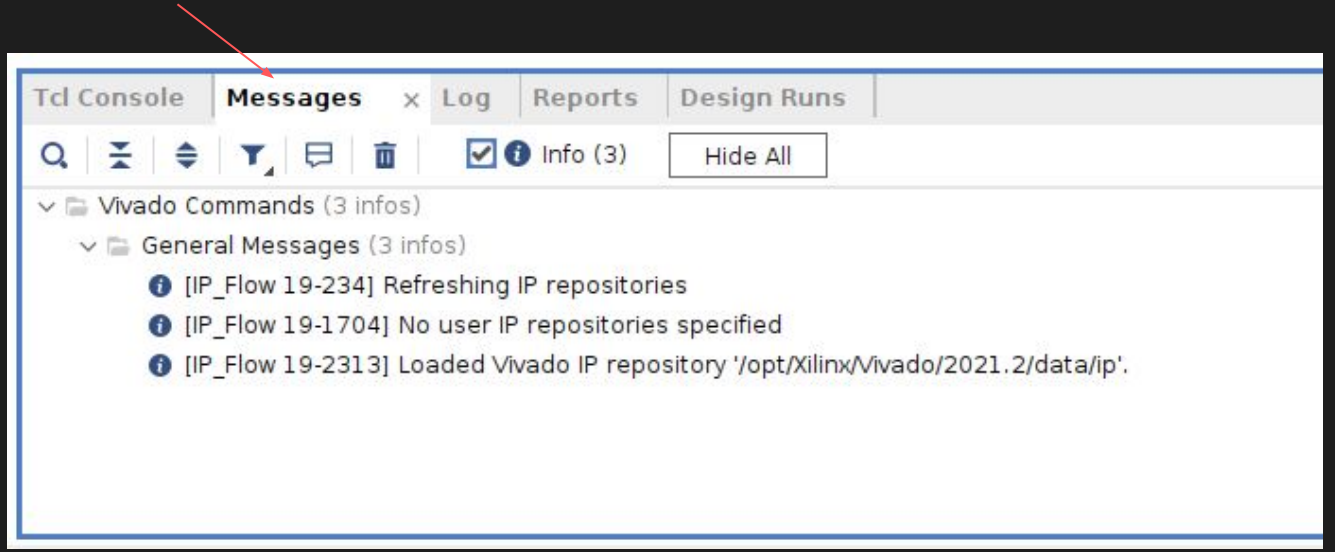


Messages

The Messages tab displays warnings (critical or otherwise) and errors that may occur during the process of building a project.

If anything goes wrong while designing and building a project, check the Messages first.

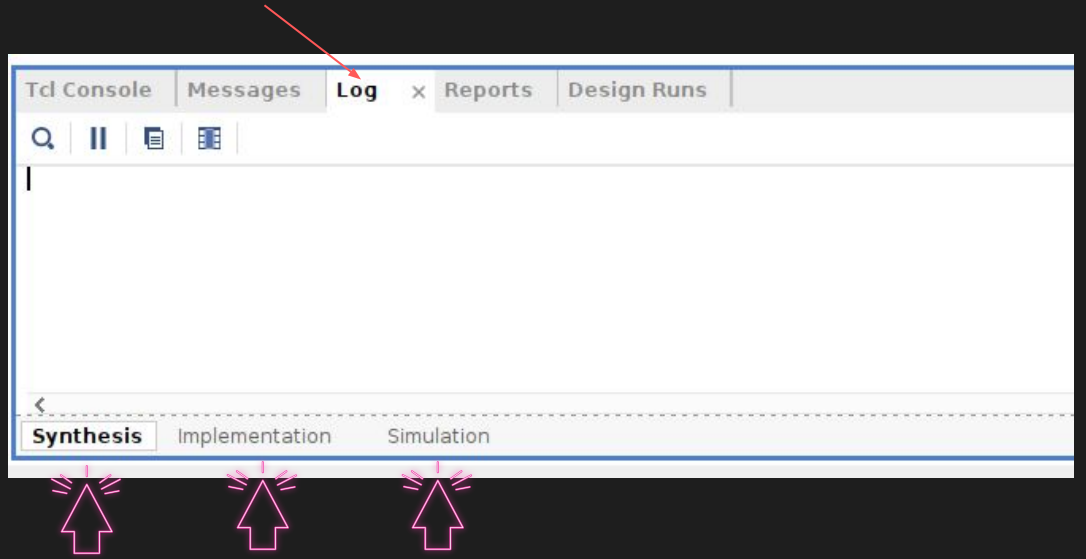
Solutions for many errors and warning can be found by right clicking on the message and selecting “Search for Answer Record”.



Log Tab

The Log displays the output from the latest Synthesis, Implementation, and Simulation runs.

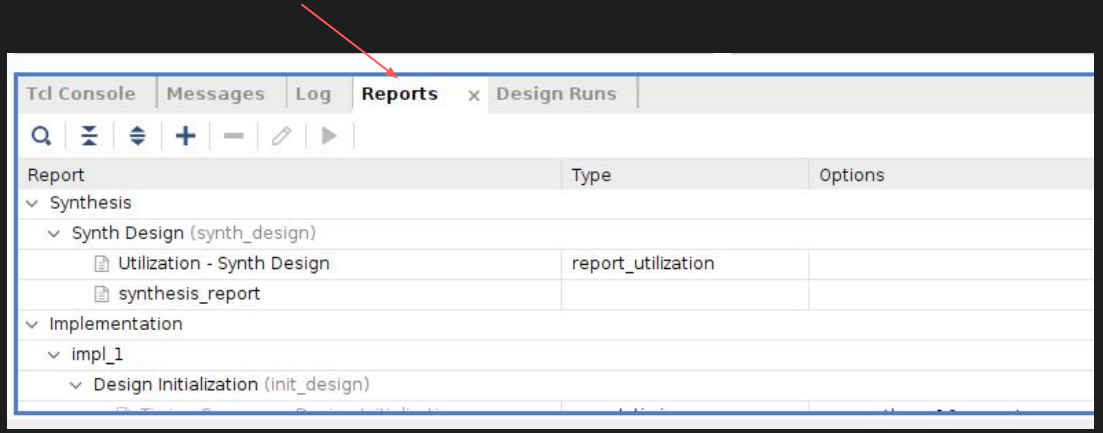
Digging into this is usually not necessary as the reports and messages view store the information in the log in a more readable format.



Reports Tab

The Reports tool is useful for quickly jumping to any one of the many reports that Vivado generates on a design.

These include power, timing, and utilization reports just to name a few.



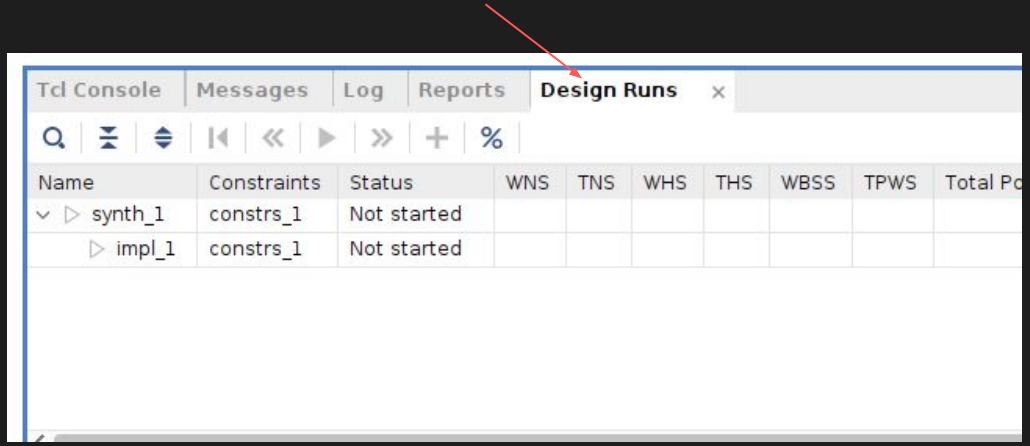
Design Runs Tab

The last tab is the Design Runs.

The Design Runs windows displays run status and information and provides access to run management commands in the popup menu.

You can manage multiple runs from the Design Runs window. When multiple runs exist, the active run is displayed in bold.

The Vivado IDE displays the design information for the active run. The Project Summary, reports, and messages all reflect the results of the active run.

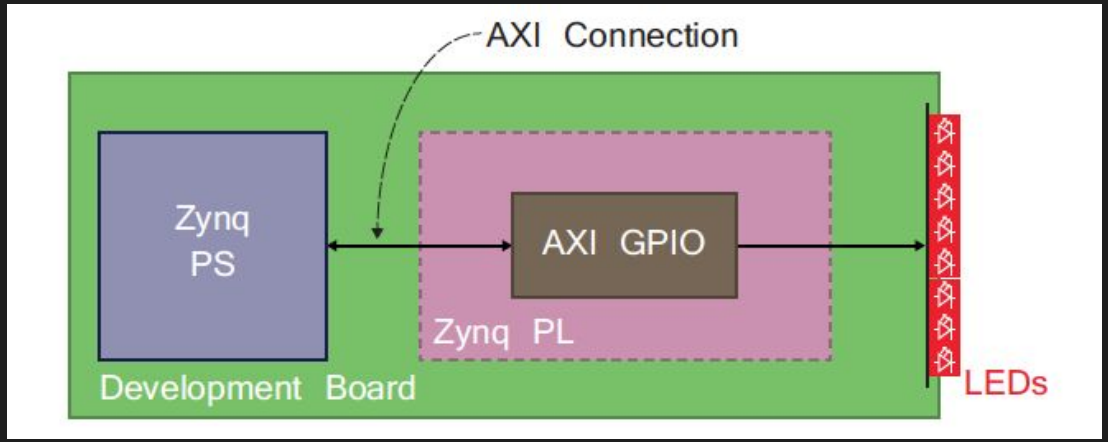


Exercise 1B
Creating a Zynq System in Vivado

Part 2 - Creating Zynq processor

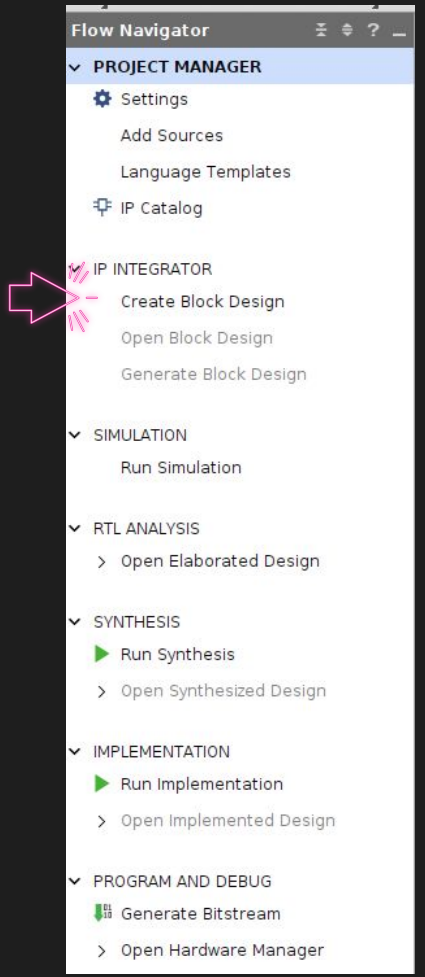
In this exercise we will be create a simple Zynq embedded system which implements a General Purpose Input/Output (GPIO) controller in the PL of the Zynq device on the ZedBoard.

The GPIO controller will connect to the LEDs. It will also be connected to the Zynq processor via an AXI bus connection, allowing the LEDs to be controlled by a software application which we will create in Exercise 1C.



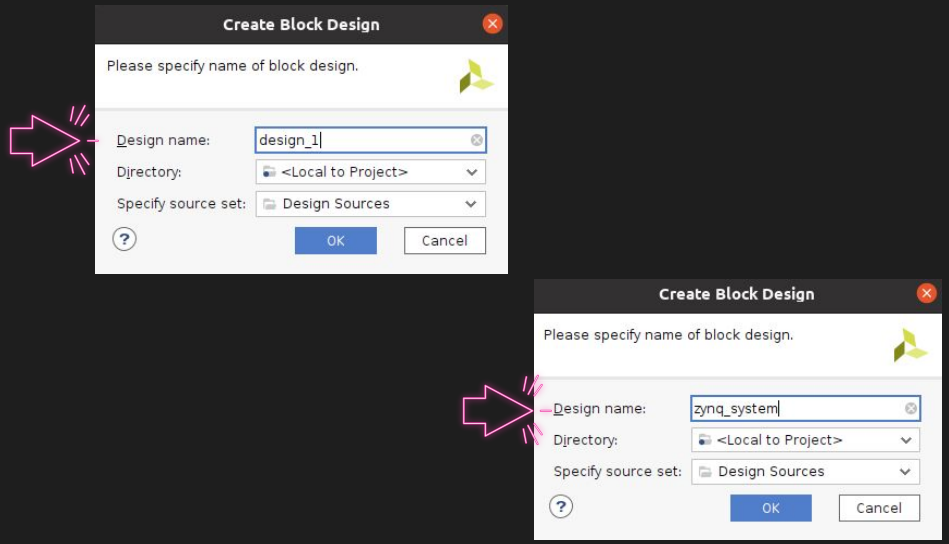
Add a block design

In the Flow Navigator window, select Create Block Design from the IP Integrator section



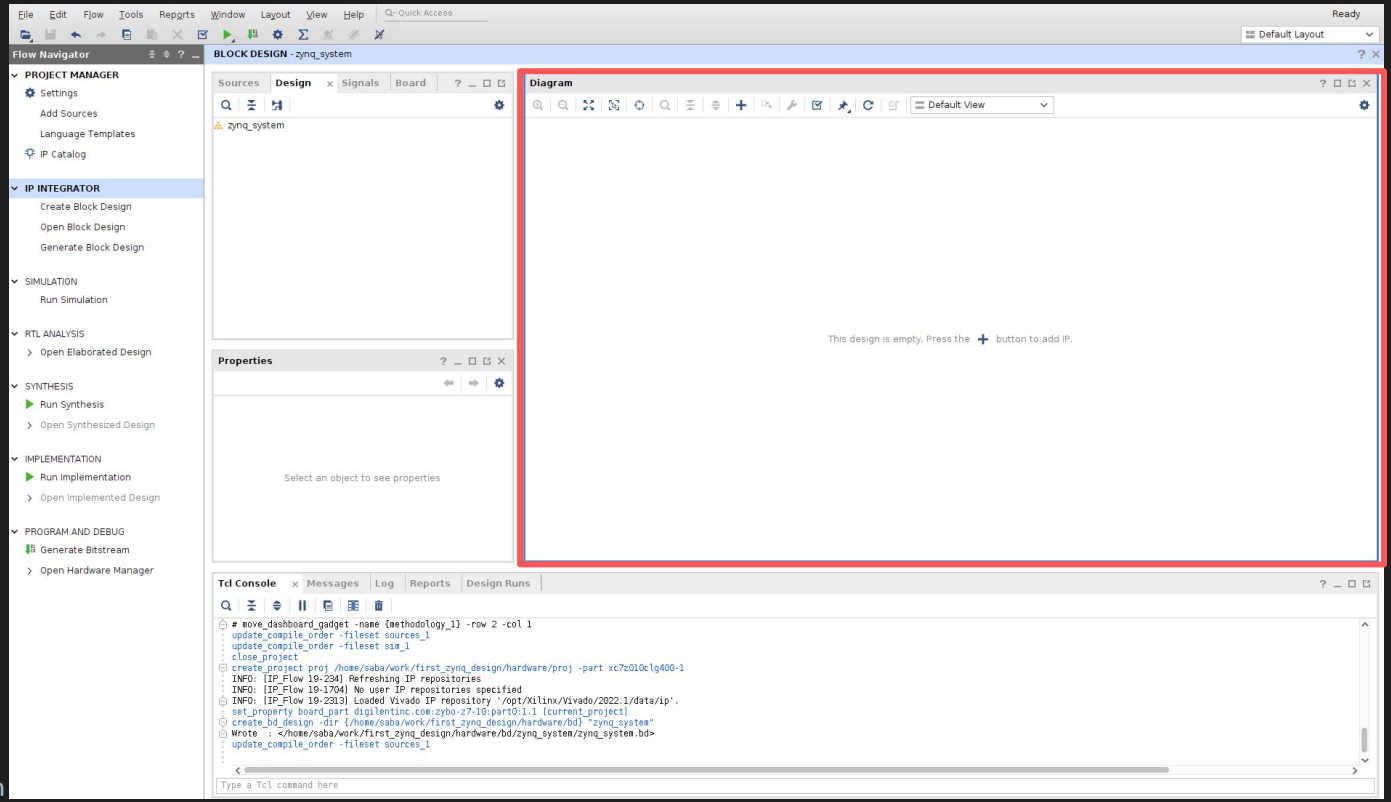
Create Block Design

In the create Block Design popup change the name of the file to “zynq_system”.



Initial Block Design Diagram window

An empty Diagram canvas will show up in the workspace area.

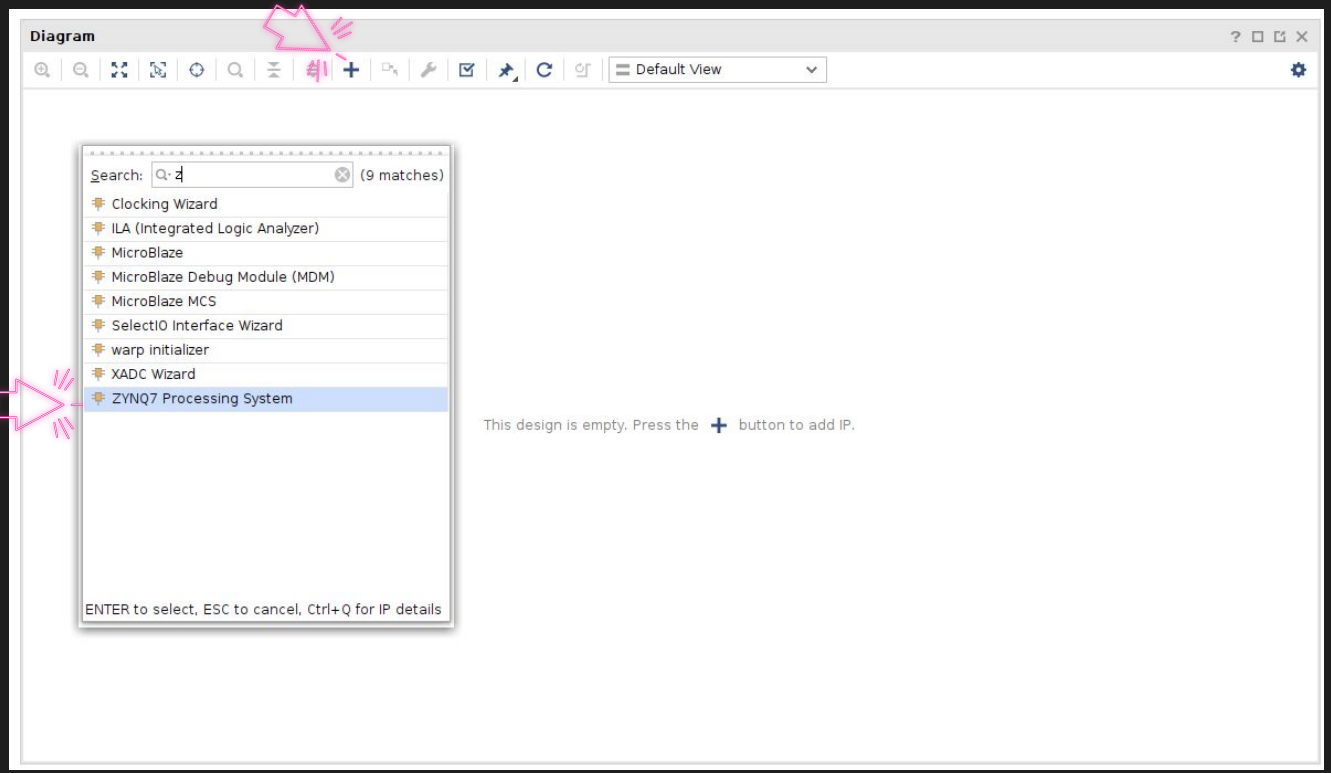


Initial Block Design Diagram window

From the diagram toolbar click on the Plus sign to open the AddIP popup menu.

In the search box type for the zynq to find the Zynq processing system.

Double click on the Zynq7 Processing System to add the IP to the canvas

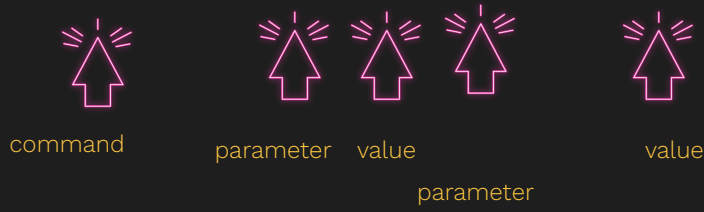


Note about using Tcl

Most of the activities in the Vivado translate to a Tcl command. Vivado always shows the corresponding Tcl command in the Tcl console.

You can use these commands to regenerate the same task.

```
create_bd_cell -type ip -vlnv xilinx.com:ip:processing_system7:5.5 processing_system7_0
```

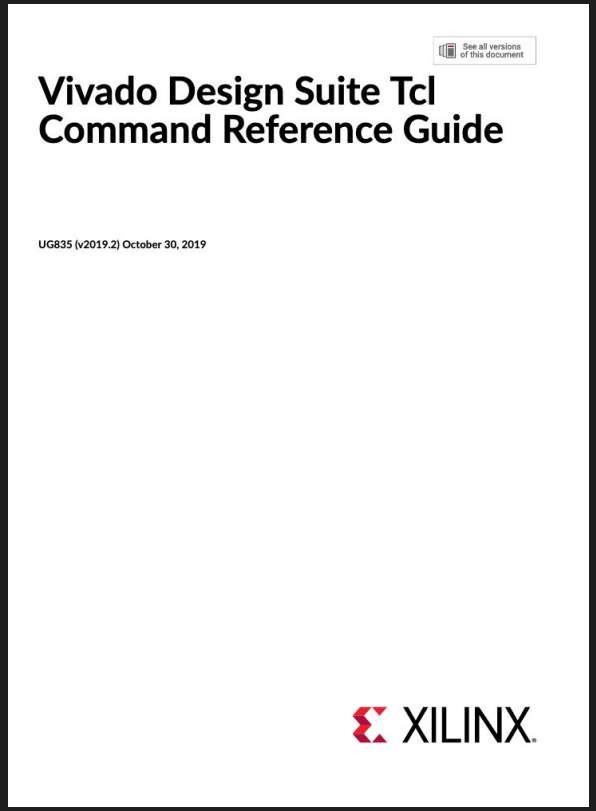


Where to find more information about Tcl commands

Xilinx User Guide UG835 is a reference to the Vivado Design Suite Tcl shell.

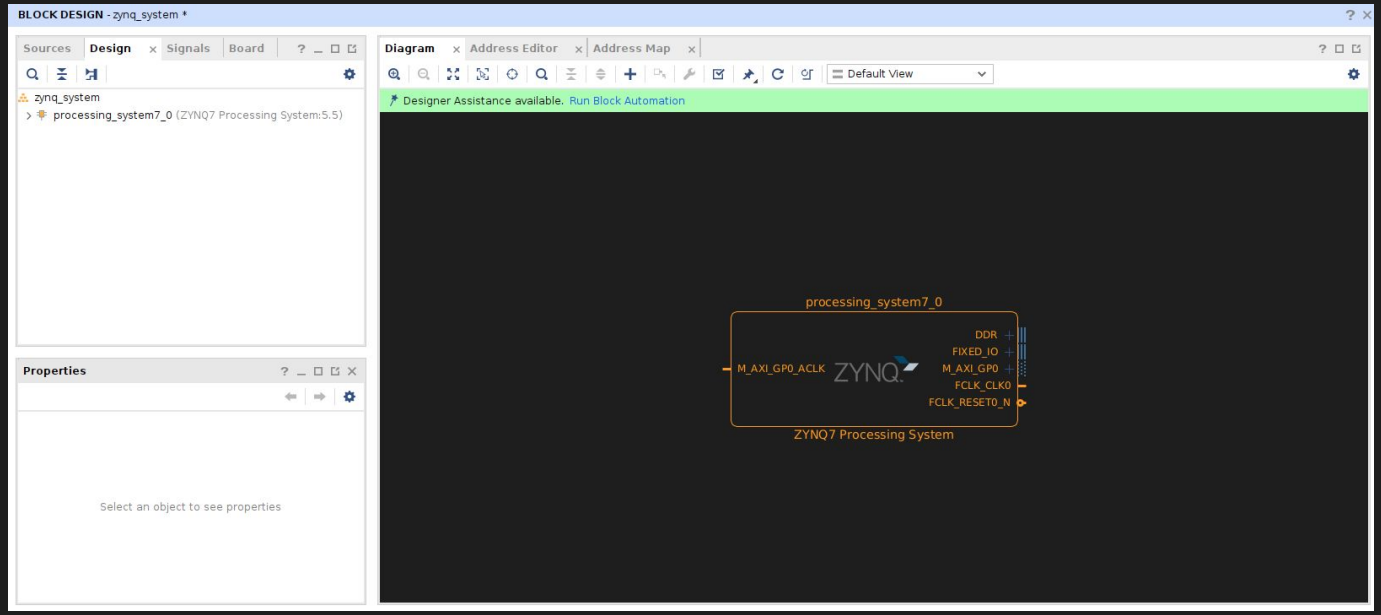
Use the following link to find the reference:

https://docs.xilinx.com/v/u/2019.2-English/ug835-vivado-tcl-commands



Zynq IP

The generated Zynq IP will show up like the following in the Diagram canvas.

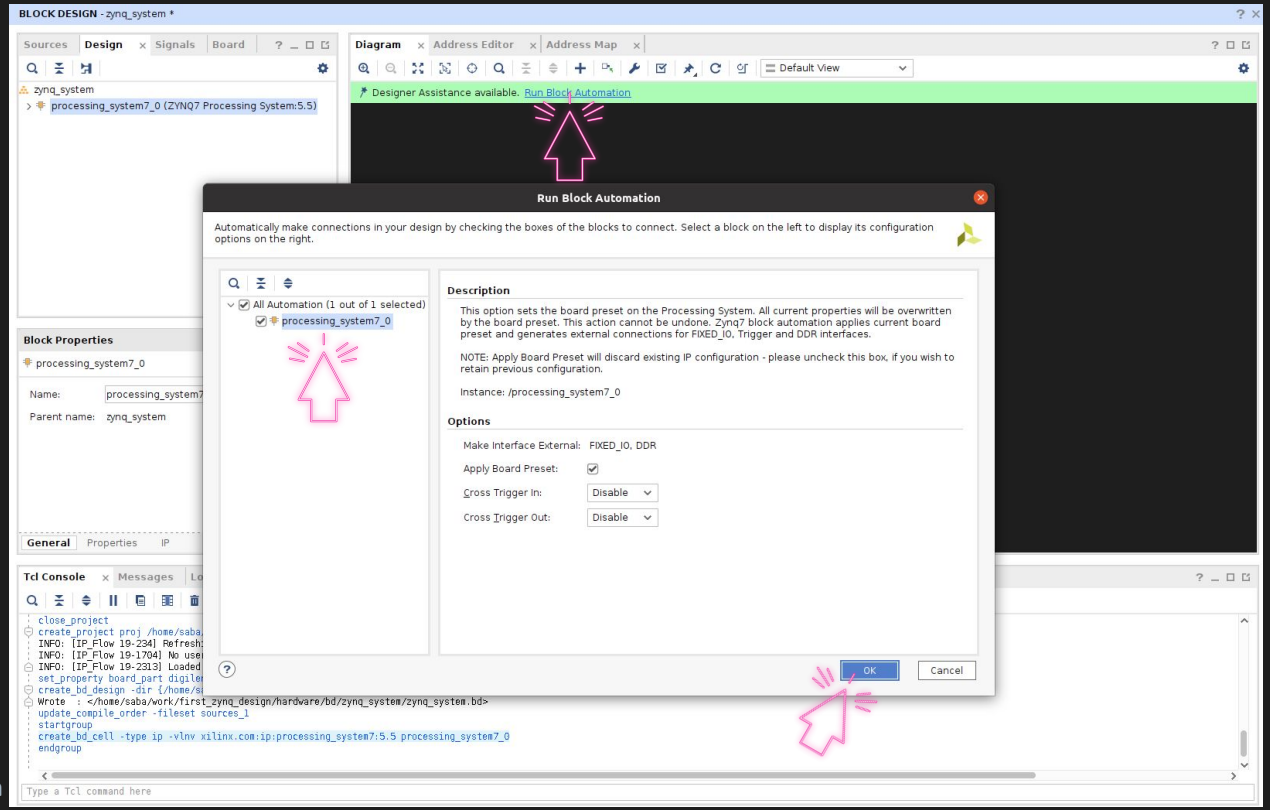


Run Block Automation

The next step is to connect the DDR and FIXED_IO interface ports on the Zynq PS to the top-level interface ports on the design.

Click the Run Block Automation option from the Designer Assistance message at the top of the Diagram window and select /processing_system7_0

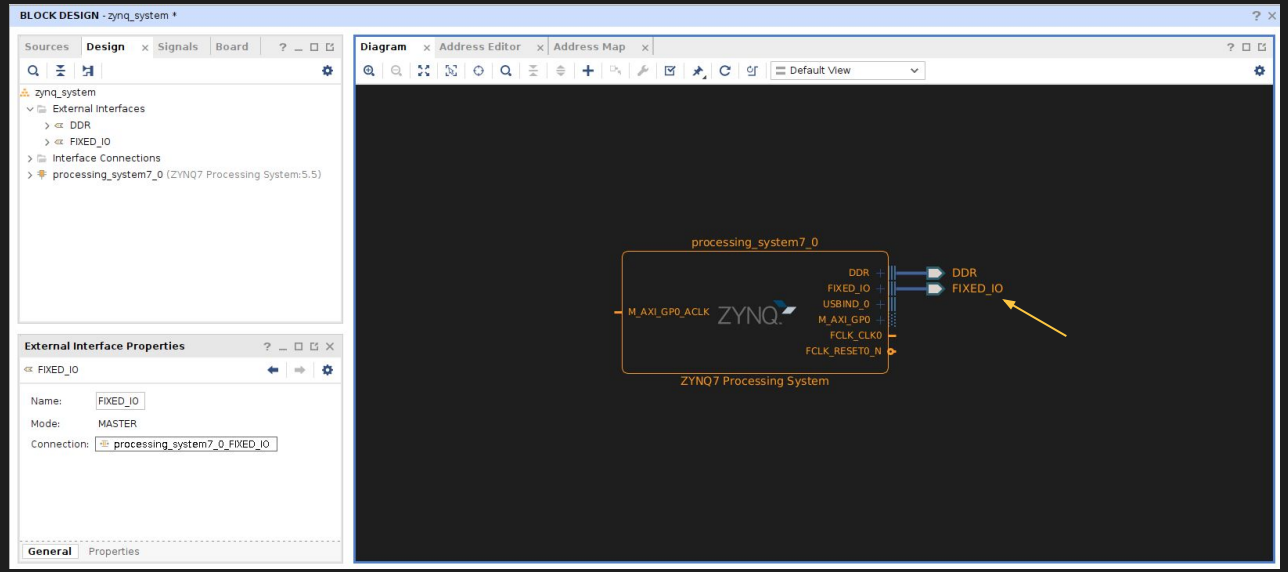
Select OK, to generate the external connections for both the DDR and FIXED_IO interfaces, ensuring that the option to Apply Board Preset is selected.



DDR and FIXED_IO

Vivado will generate two external ports and connect them to the DDR and Fixed_io.

Now that the main Zynq PS has been added to our design and configured, we can now add further blocks which will be placed in the PL to add functionality to the system.



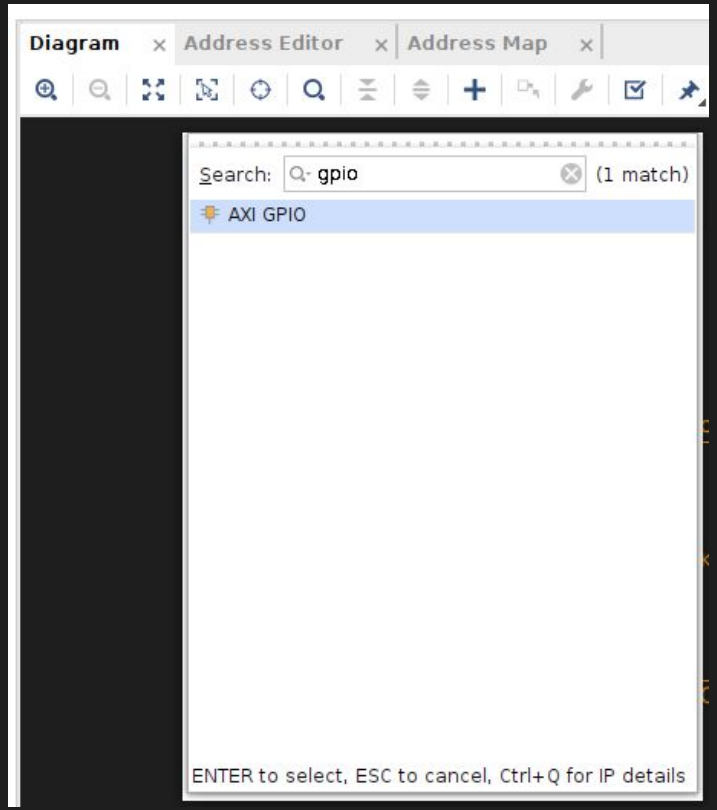
Adding GPIO

GPIO stands for General Purpose Input Output. These pins are reconfigurable pins that can be used to connect to different peripherals.

In this case we will only be adding a single block, AXI GPIO, to allow us to access the LEDs on the Board.

Right-click in an empty area of the Diagram window and select Add IP.

Enter GPIO in the search field and add an instance of the AXI GPIO IP.

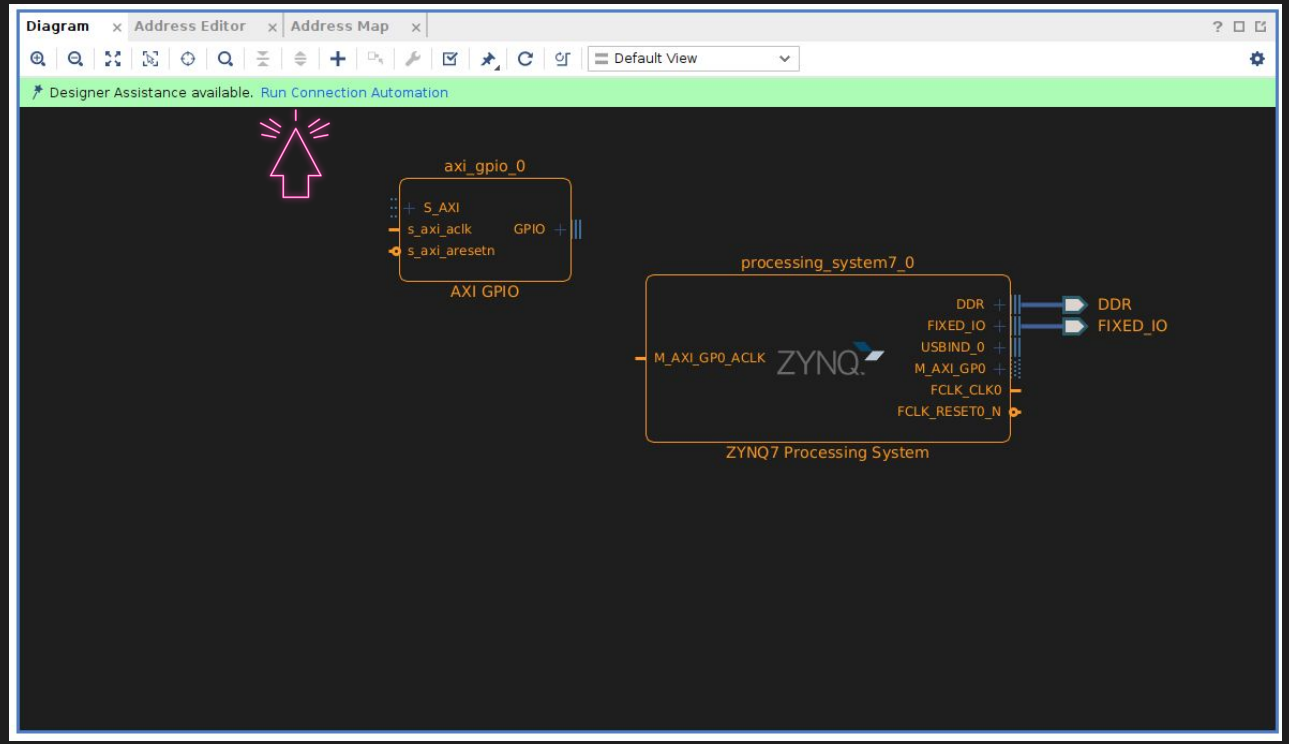


Adding GPIO

The generated GPIO block looks like the following.

Click Run Connection Automation from the Designer Assistance message at the top of the Diagram window.

This will automate the process of connecting the GPIO to an AXI port.



Connect the GPIO to Zynq

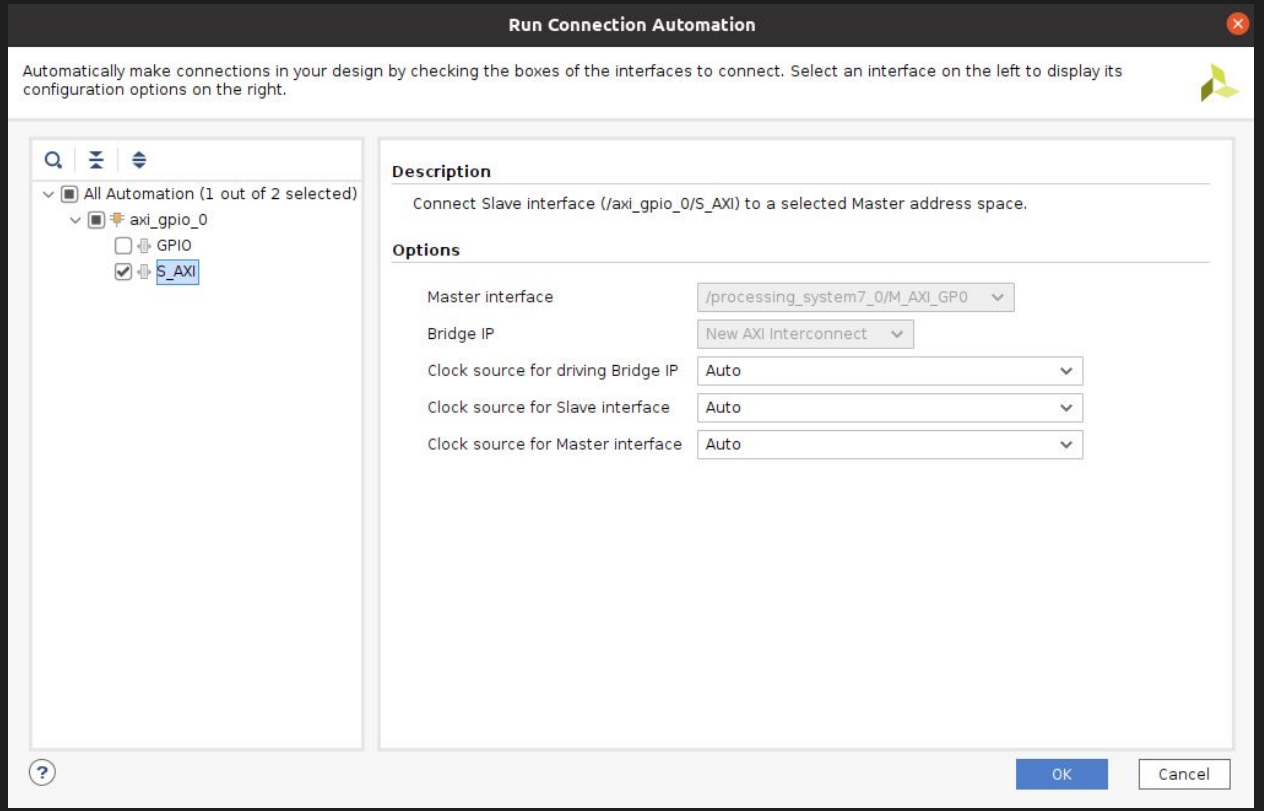
The automator also will automatically instantiate two further IP blocks

Processor System Reset Module: This provides customised resets for an entire processing system, including the peripherals, interconnect and the processor itself.

AXI Interconnect: Provides an AXI interconnect for the system, allowing further IP and peripherals in the PL to communicate with the main processing system.

Leave the option for Clock Connection (for unconnected clks) to Auto, and Click OK.

All connections between the blocks should be made automatically.



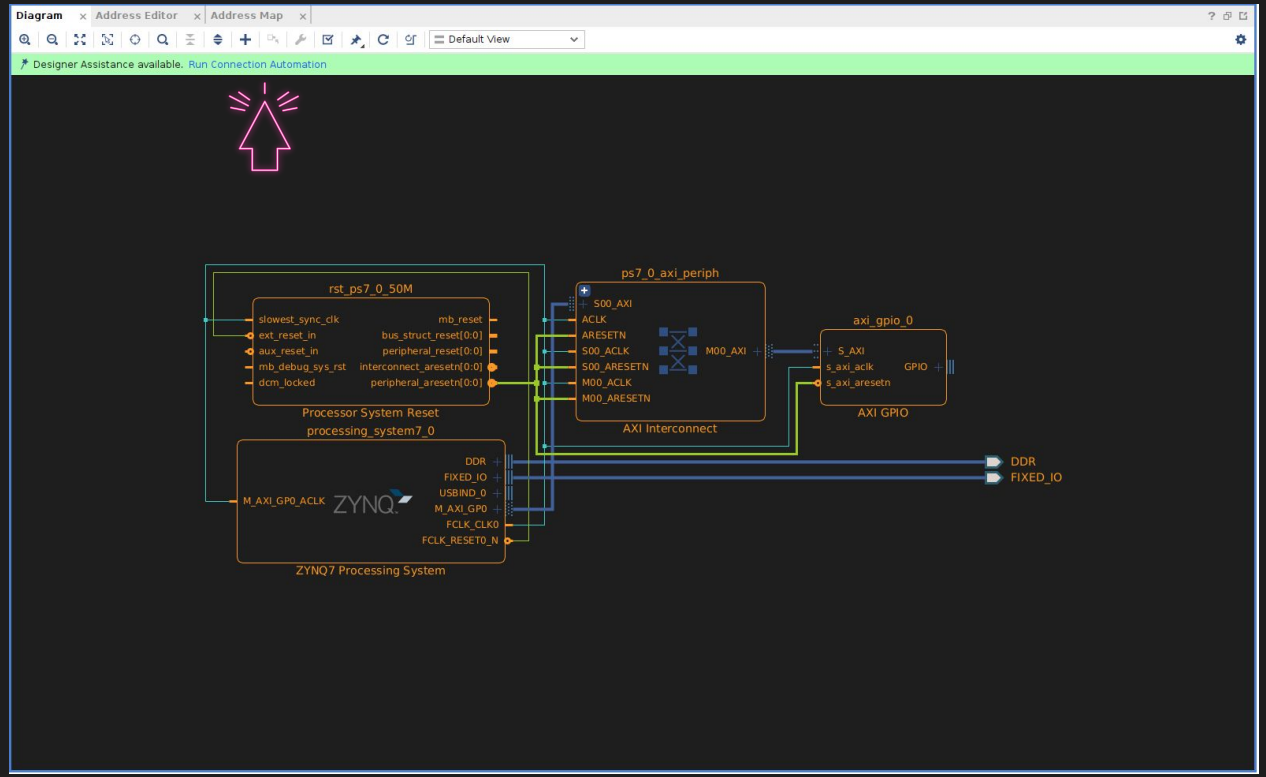
First group of generated blocks

The tools adds a reset IP, and an AXI interconnect to the design.

One final connection is required to connect the AXI GPIO block to the LEDs on the board.

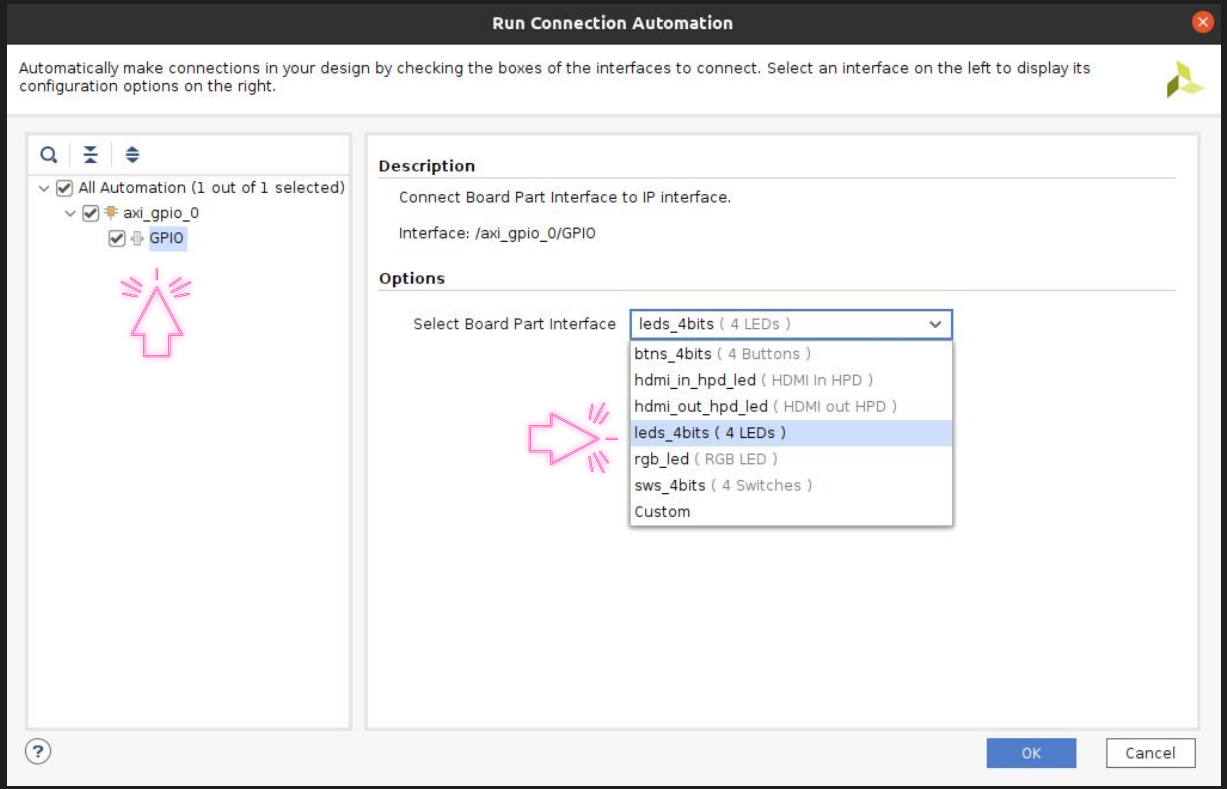
This can also be completed using Designer Assistance.

Click Run Connection Automation from the Designer Assistance message at the top of the Diagram window.



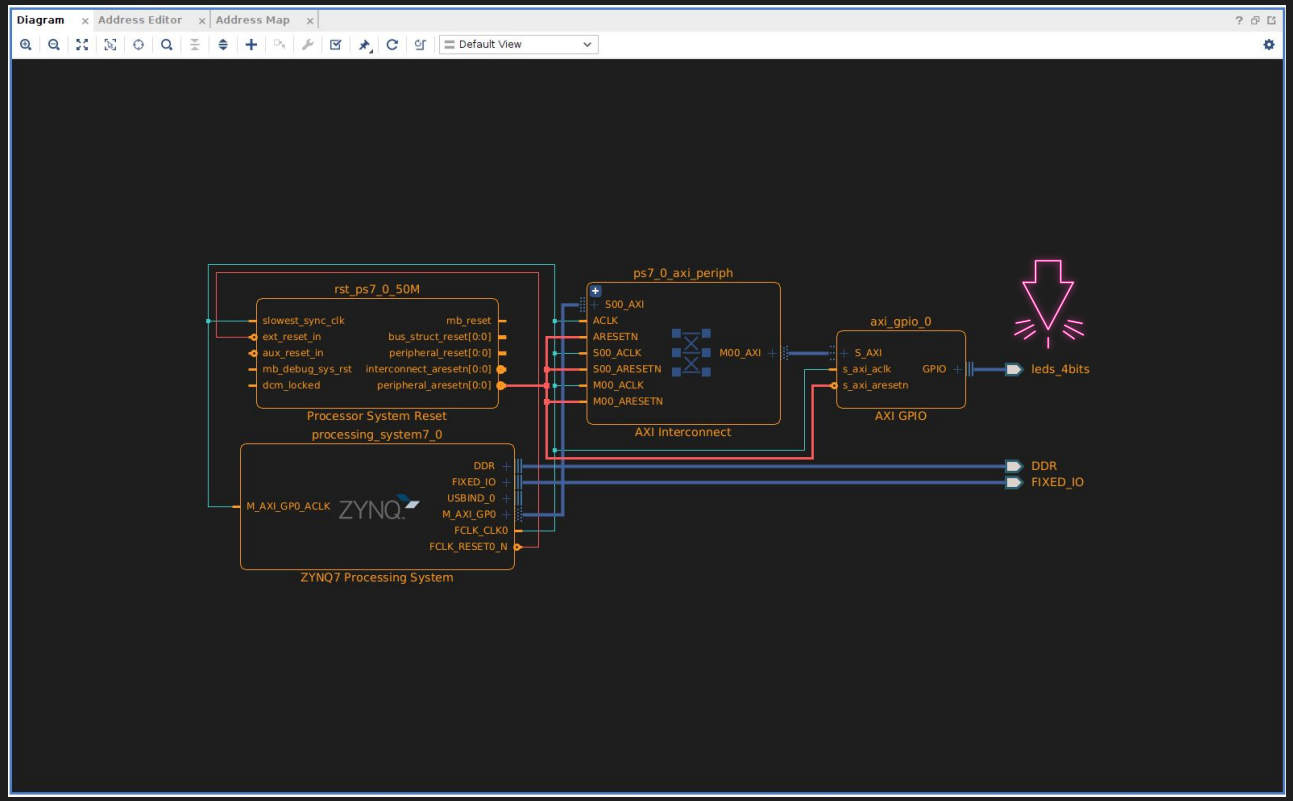
Selecting the type of GPIO interface

Since we wanted to control the LEDs select the leds_4bits from the dropdown menu.



leds_4bits external port

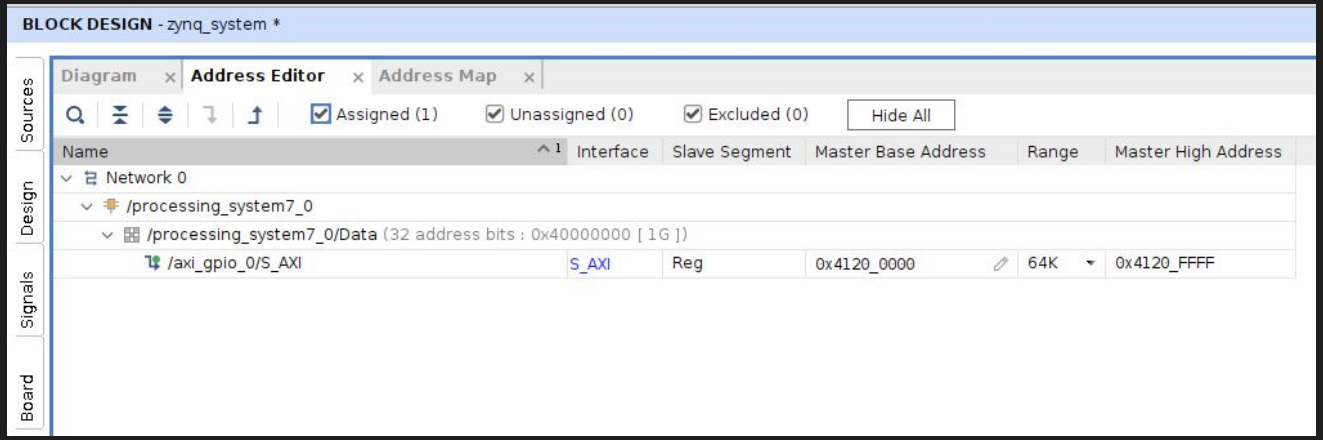
The tool generates an external port for the axi_gpio and name it leds_4bits.



Check the Address Editor

IP Integrator will automatically assign a memory map for all IP that is present in the design.

We will not be changing the memory map in this tutorial, but for future reference we will take a look at the Address Editor.

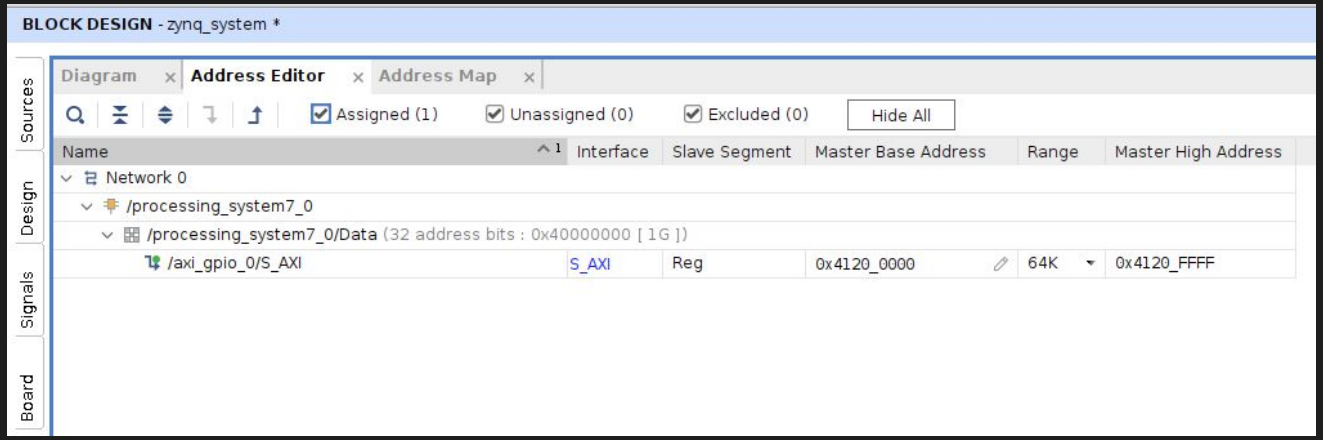


Check the Address Editor

You can see that IP Integrator has already assigned a memory map (the mapping of specific

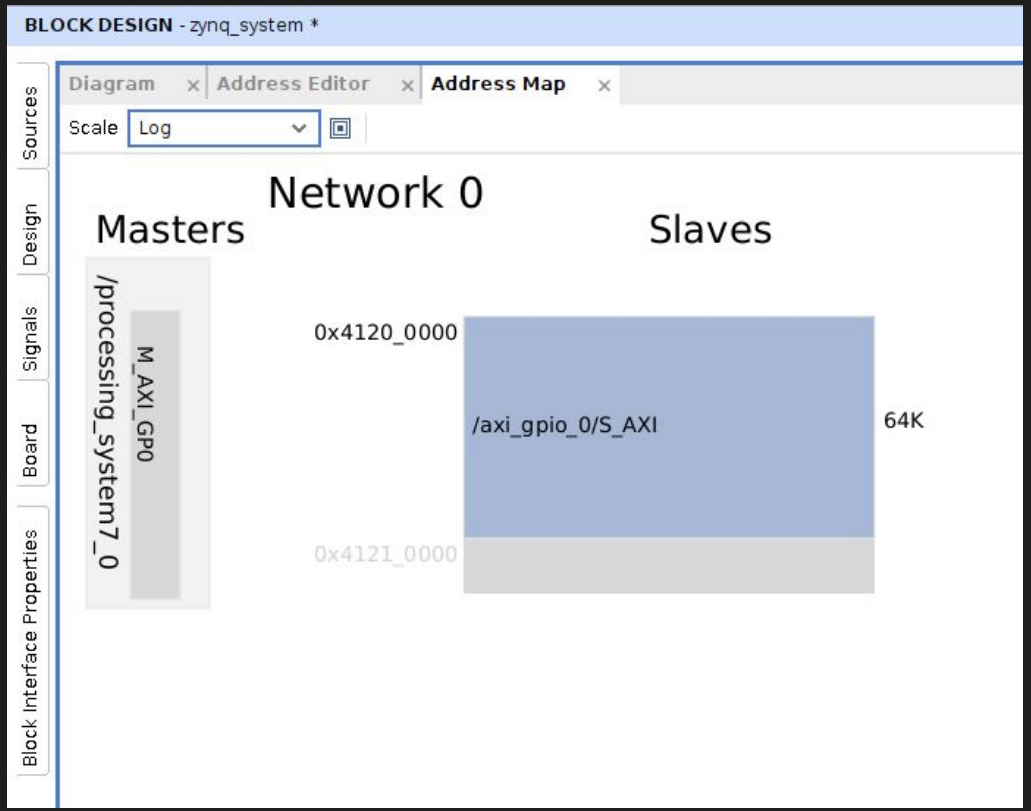
sections of memory to the memory-mapped registers of the IP blocks in the PL) to the to the

AXI GPIO interface, and that it has a range of 64K.



Address Map

You can also take a look at a graphical representation of the memory map under the Address Map tab.

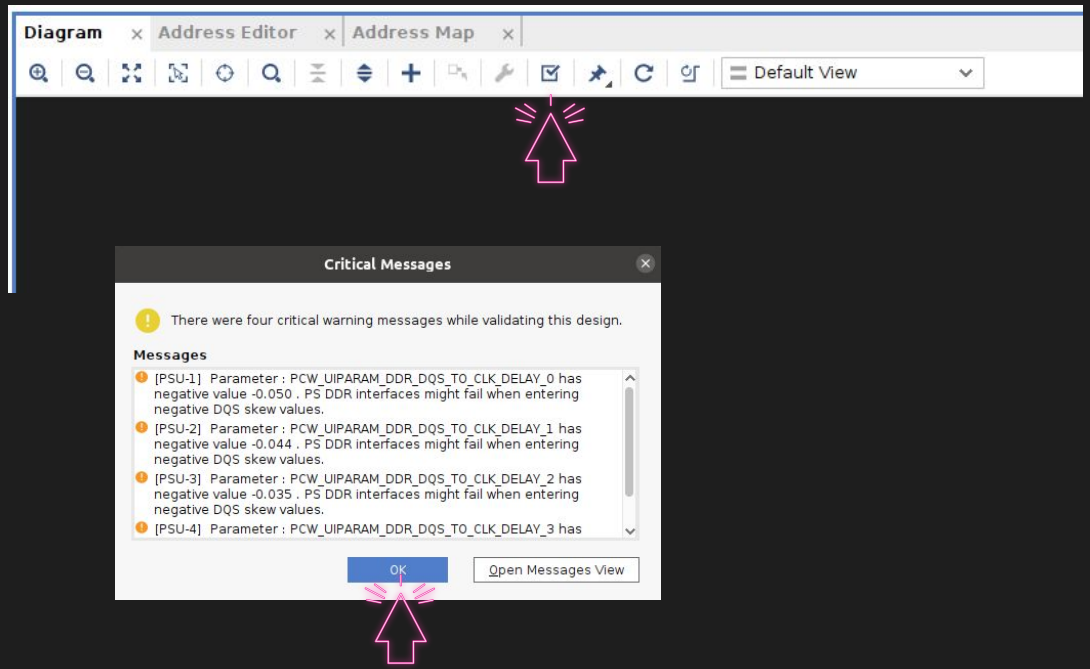


Validate the design

Now that our system is complete, we must first validate the design before generating the HDL design files.

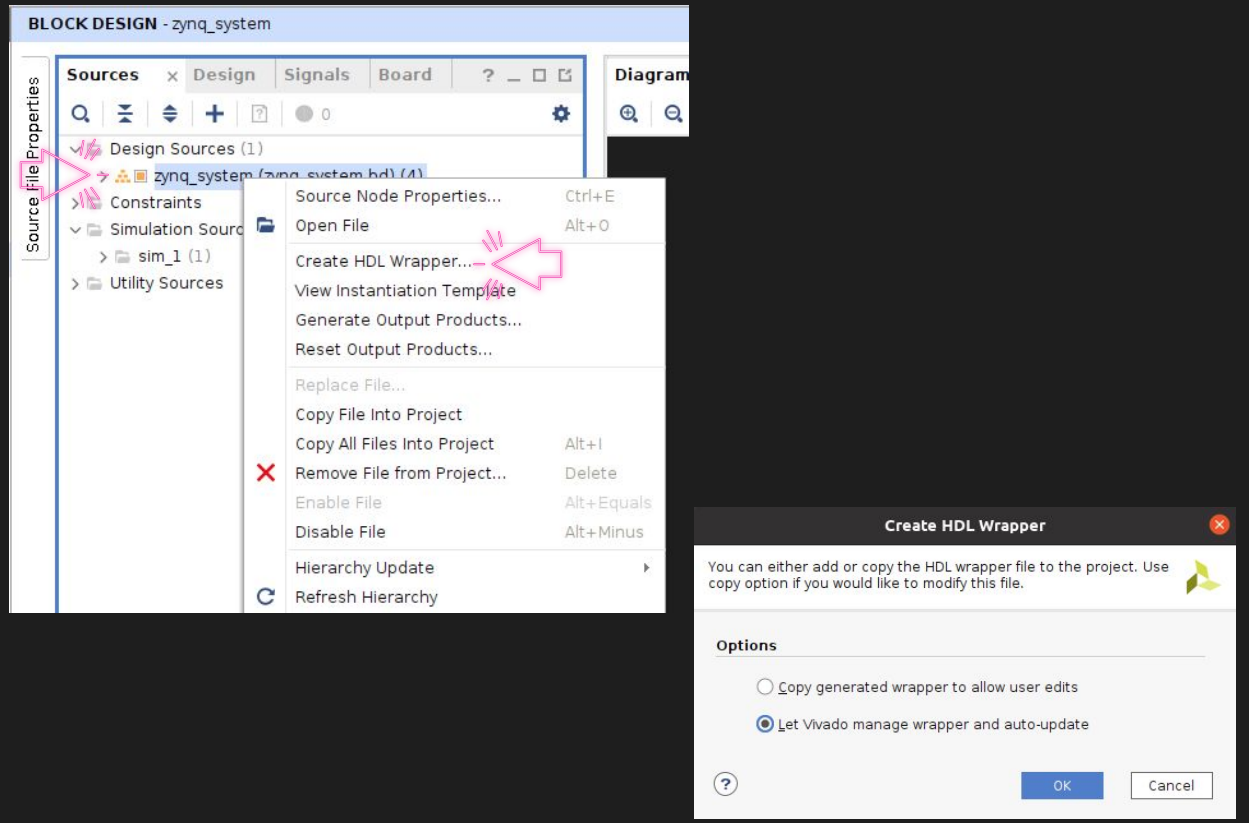
- Save your design by selecting File > Save Block Design from the Menu Bar.
- Validate the design by selecting Tools > Validate Design from the Menu Bar. This will run a Design-Rule-Check (DRC). Alternatively, select the Validate Design button, , from the Main Toolbar, or right-click anywhere in the Diagram canvas and select Validate Design.
- A Validate Design dialogue should appear to confirm that validation of the design was successful. Click OK, to dismiss the message.

Note: For Zybo Z7 boards you will get the following Warning prompt. You safely can ignore the warnings. We will explain what is the issue here.



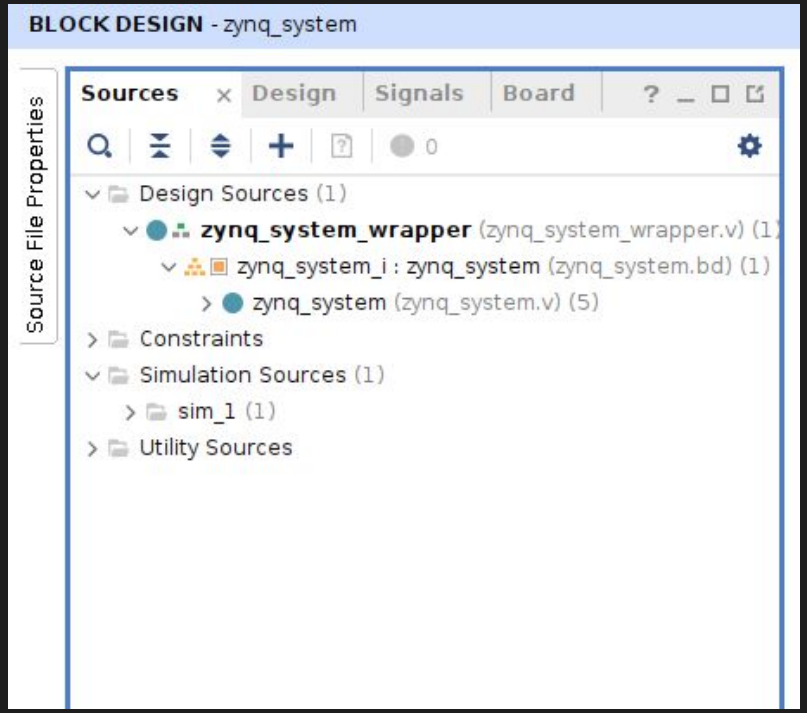
Creating HDL wrapper

With the design successfully validated, we can now move on to generating the HDL design files for the system.



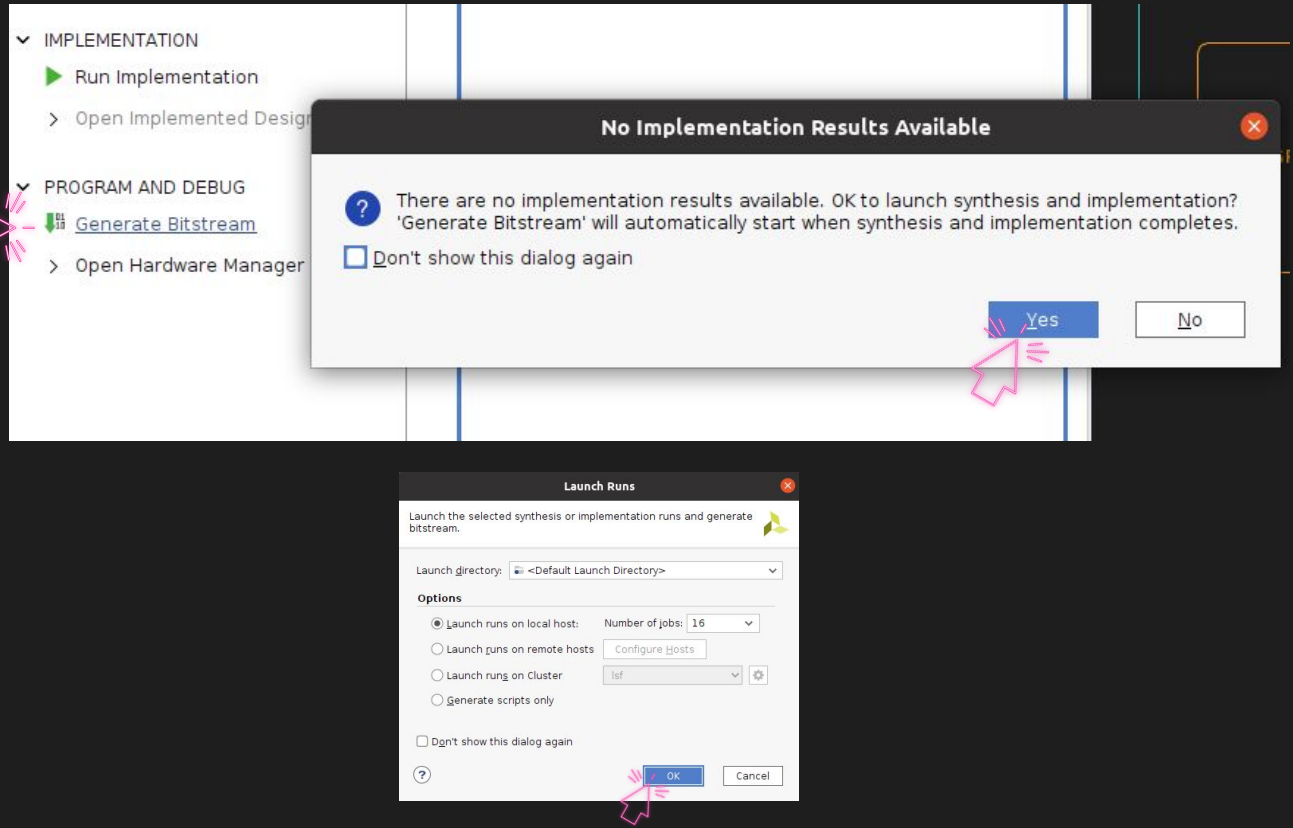
Generated wrapper

The generated wrapper will appear in the source menu.



Generate bitstream

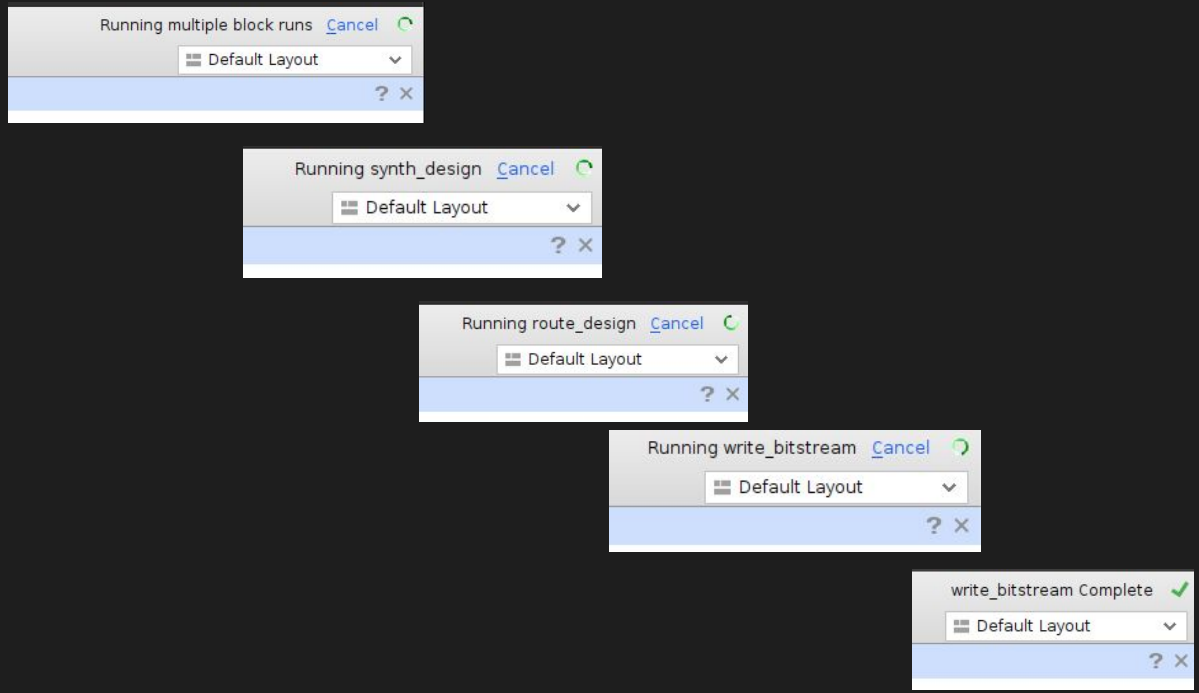
With all HDL design files generated, the next step in Vivado is to implement our design and generate a bitstream file.



Check the program status bar

The program status bar will keep you updated about the status of your build.

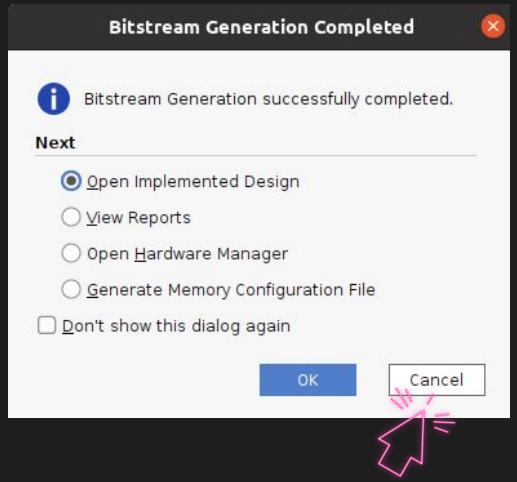
Wait until the write_bitstream completed green check mark appears in the status bar.



Bit stream generation popup

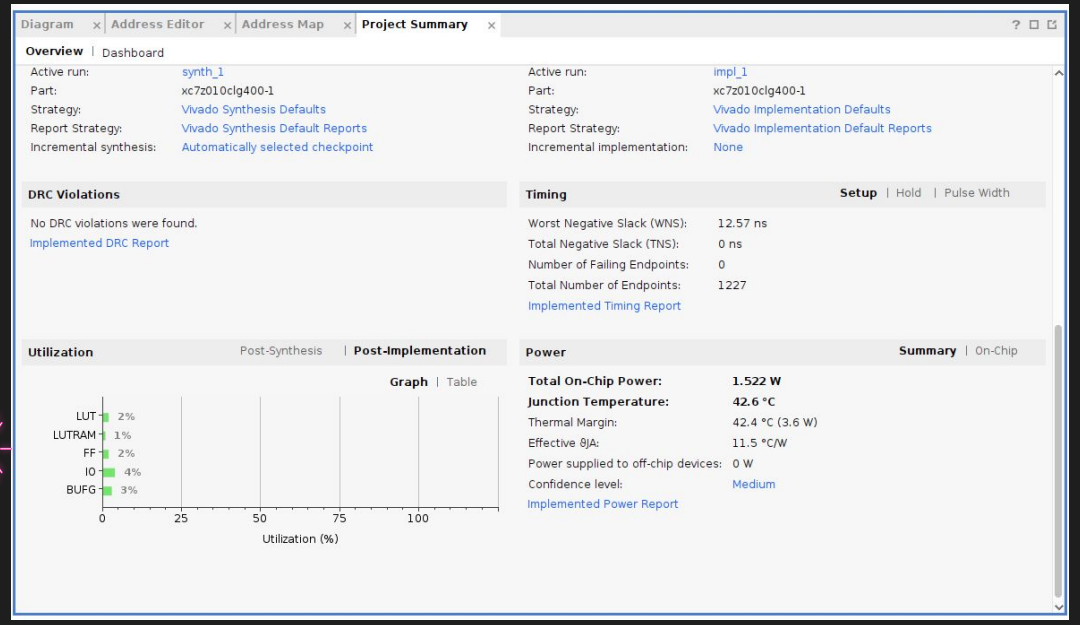
A popup will inform you about the success or failure of your build.

For this project we will not analyze any report so you can press cancel in the prompt.



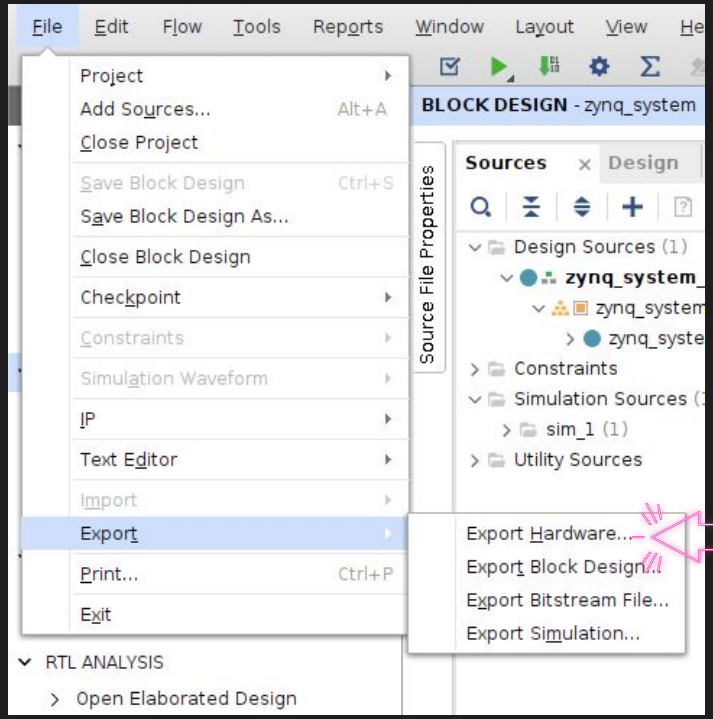
Project summary window after bitstream generation

At this point you will be presented with the Device view, where you can see the PL resources which are utilised by the design.



Export hardware

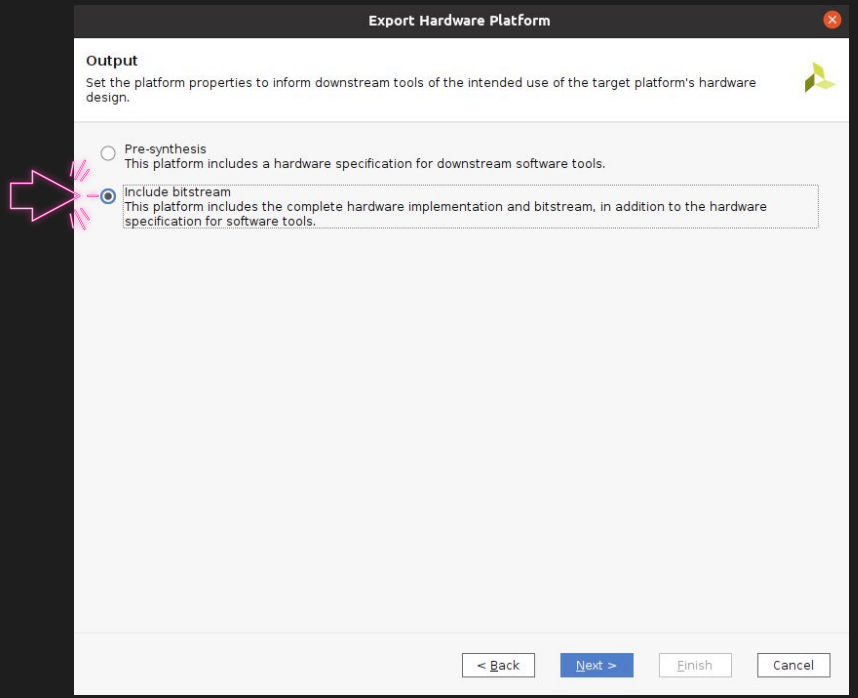
With the bitstream generation complete, the final step in Vivado is to export the design to the Vitis, where we will create the software application that will allow the Zynq PS to control the LEDs on the board.



Include bitstream

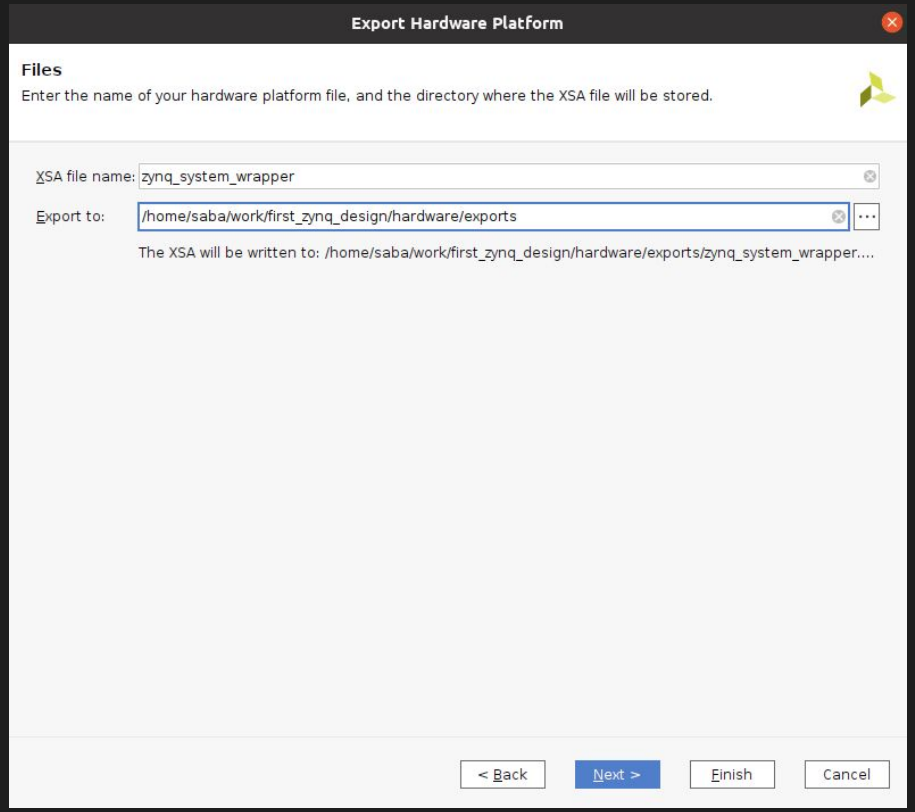
The Export Hardware for Vitis dialogue window will open.

Ensure that the options to Include bitstream and Launch SDK are selected, and Click OK.



Save the XSA file

Save the Xilinx Support Archive (XSA) file in the exports folder outside of the project.



Source Control Vivado using Git

Cal State Northridge - Saba Janamian

Avoid tracking auto generated files

Vivado generates a large number of auto generated files.

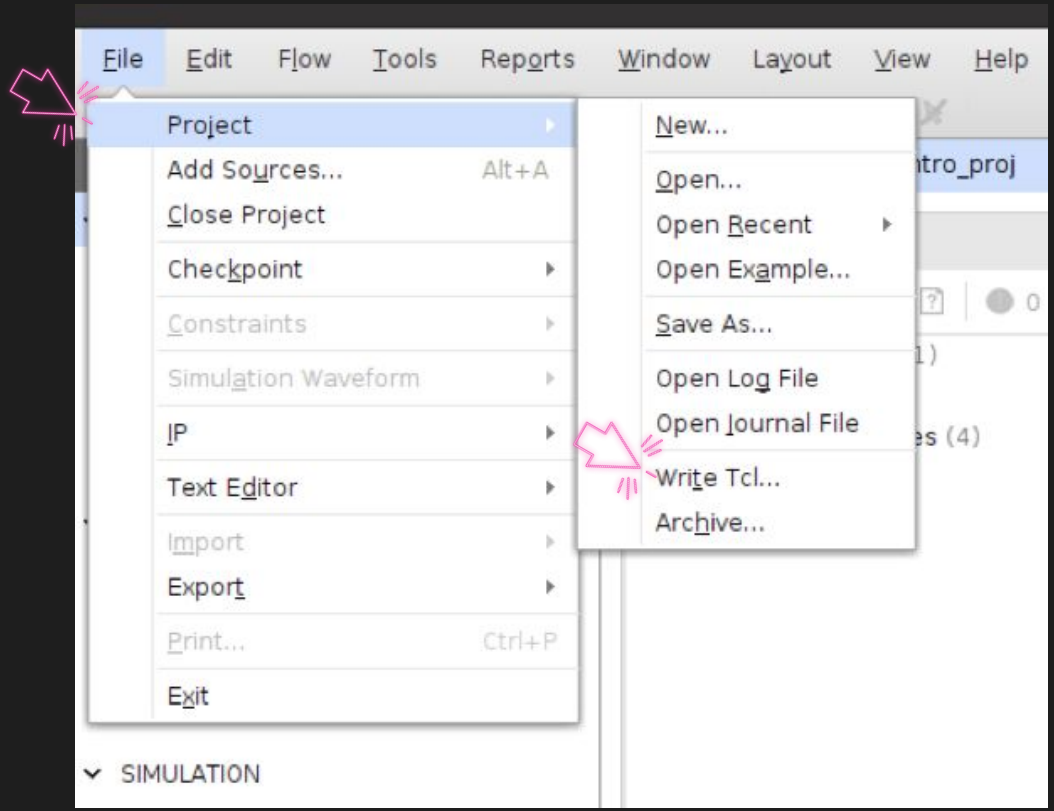
Many of these files are large binary files that can be always regenerated.

We do not want to record these files in a Git repository as it will cause problems in Git

Luckily Vivado allows exporting project into a tcl file which you can “source” to regenerate all the files.

Export project as Tcl

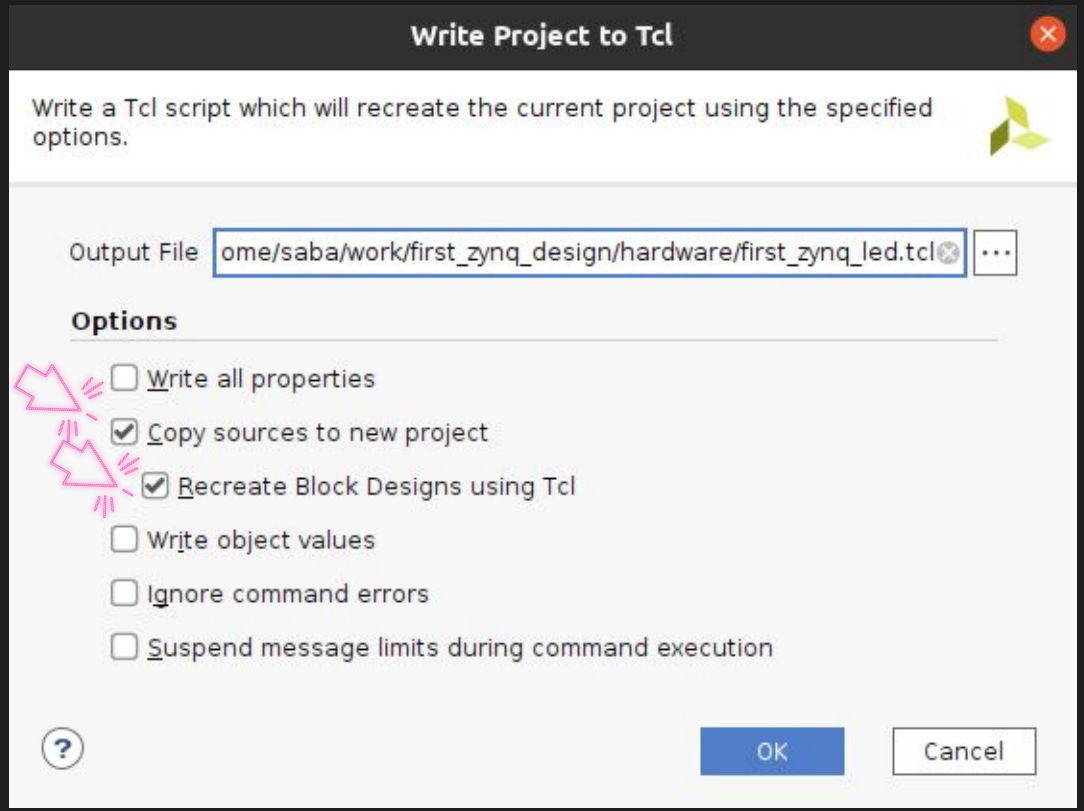
Select Write Tcl from the File > Project menu



Save the file

Save the file one step above the local project folder.

Since we have a block design to manage make sure to check the Create Block Design using Tcl option



Few notes about generated Tcl file

Unfortunately the auto generated Tcl file could have problems that could prevent your from regenerating your project using the generated Tcl script.

You need to always double check the content of the tcl and make sure everything is correct.

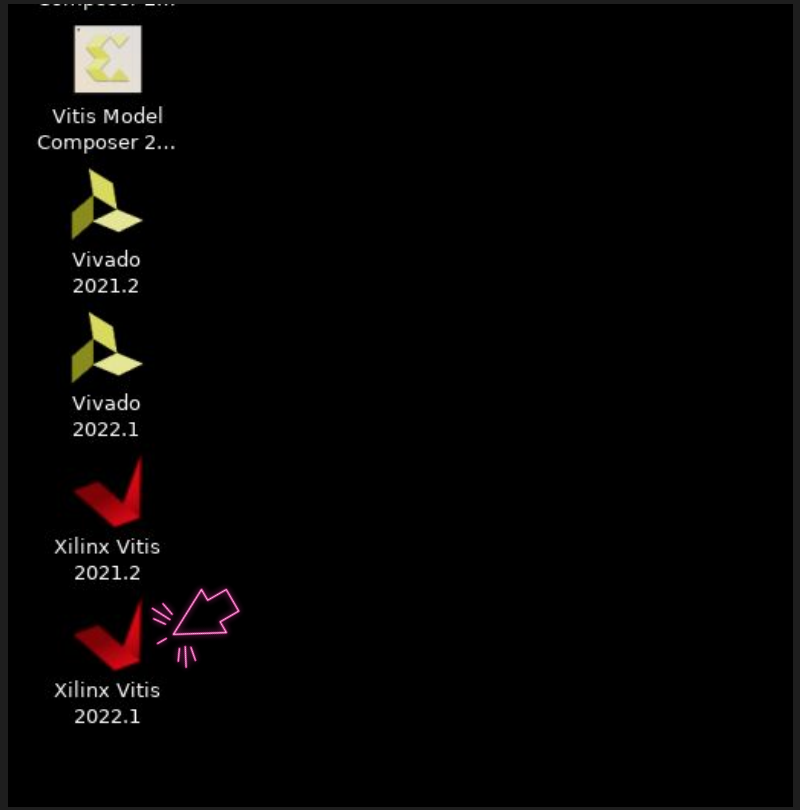
Creating .gitignore file

The .gitignore file is a text file that tells Git which files or folders to ignore in a project.

In the root directory of your project (not Vivado project) create a .gitignore file and place the following lines in it. Note the name of the folder should match your project name.

Exercise 1C
Creating a Software Application in Vitis

Start Vits IDE



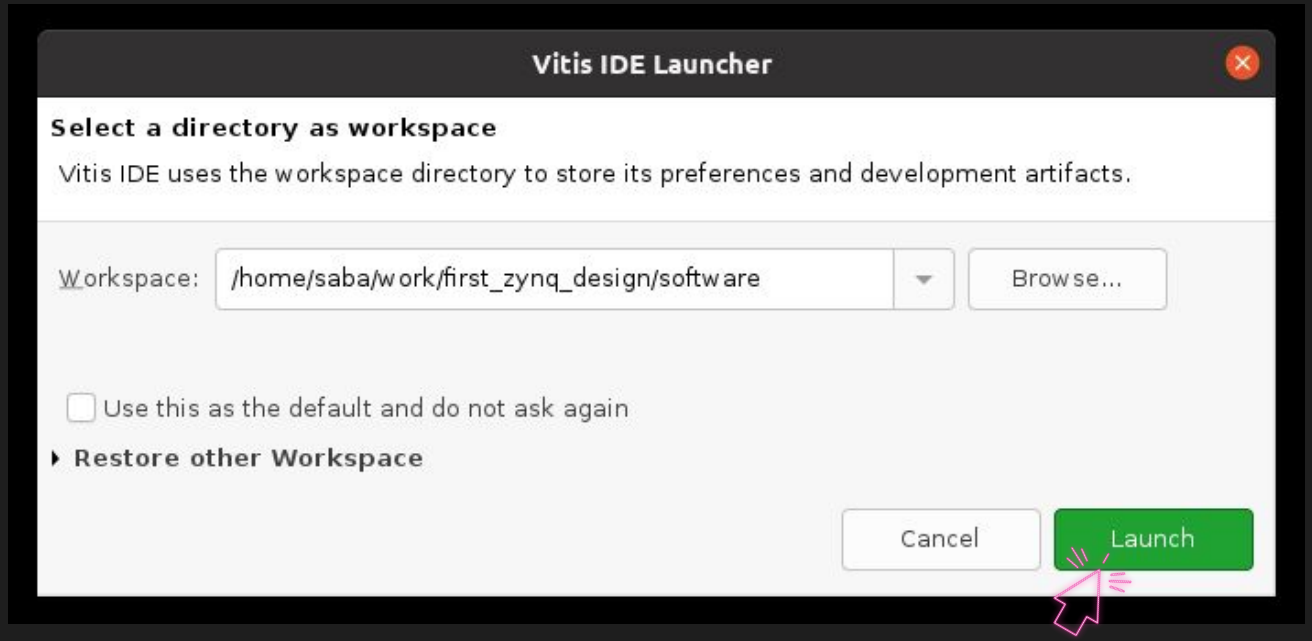
Folder structure

We will manage hardware and software build separately.

Make sure to create a software folder in the base directory of your project and use it as your Vitis workspace.

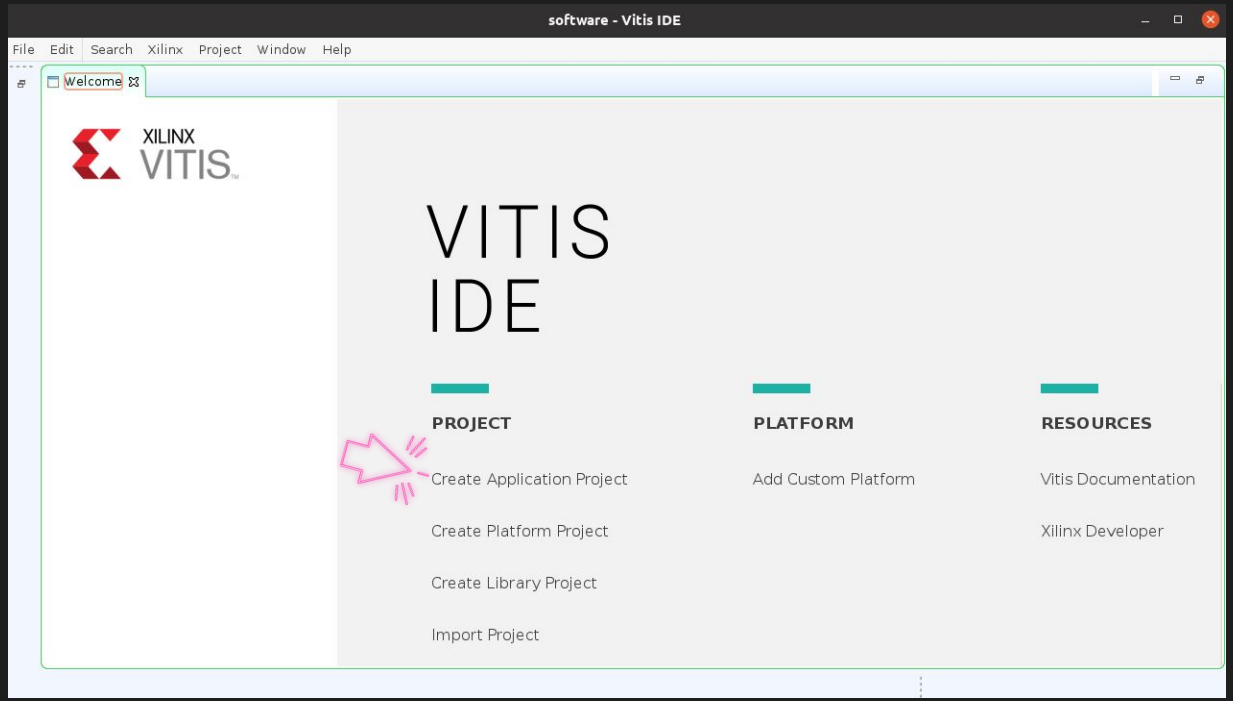
Vitis workspace

Select the software folder for your Vitis workspace



Create Application

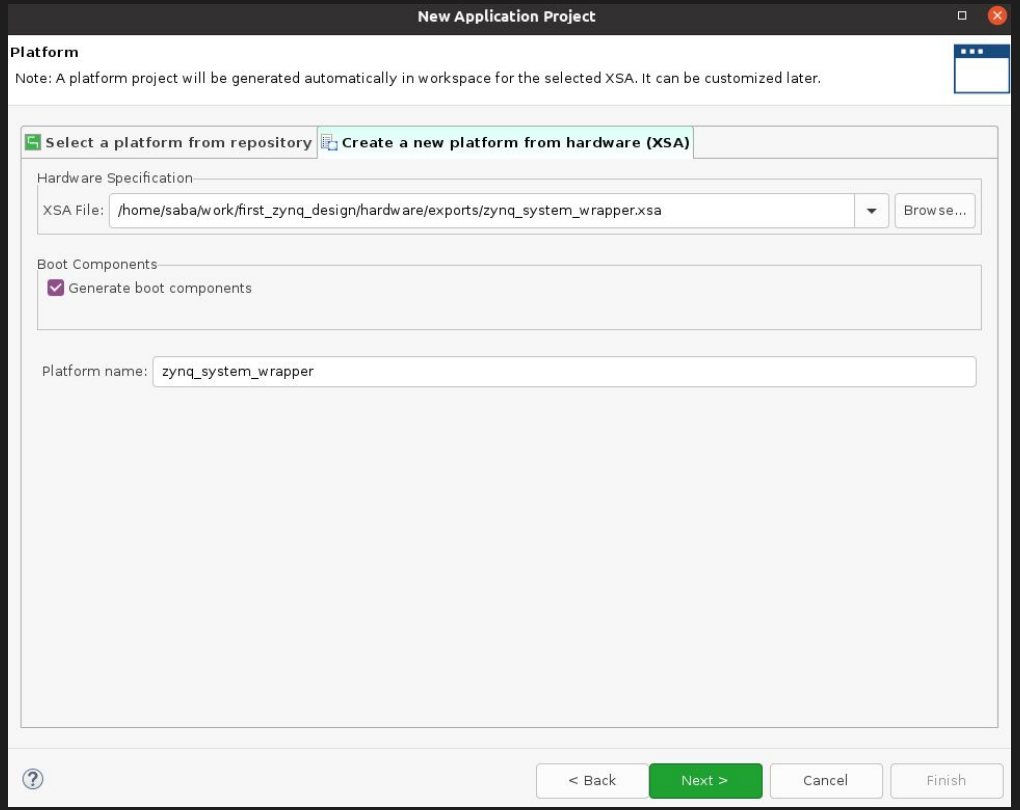
From the welcome menu click on Create Application project.



Create platform from hardware

Select your newly exported XSA file to Vitis.

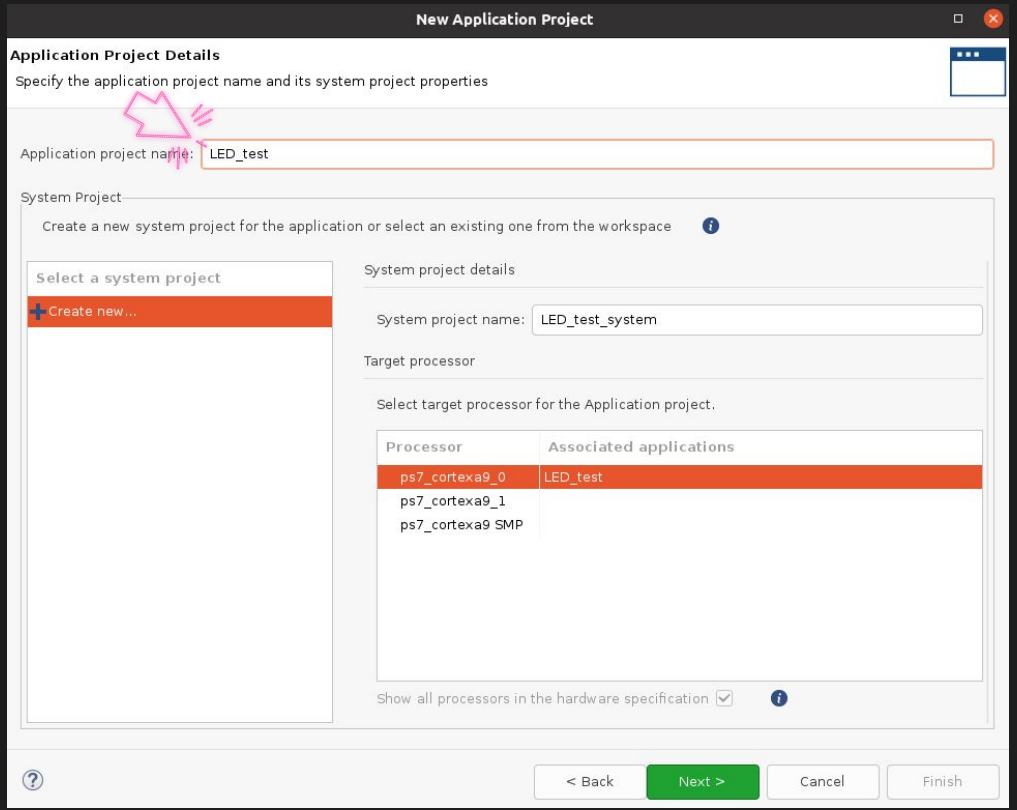
Vitis will use this file to generate all the required drivers to interface the software environment with your FPGA hardware.



Create an application

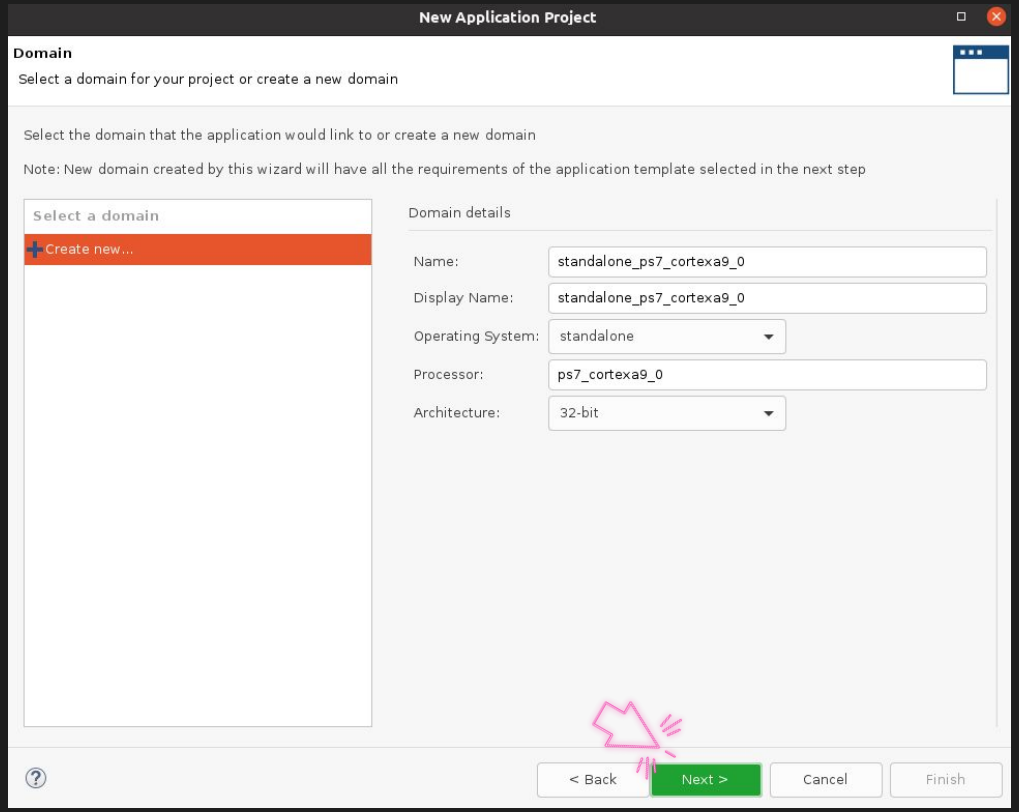
Name your project LED_test.

This will be the main project holding your source codes.



Running as standalone application

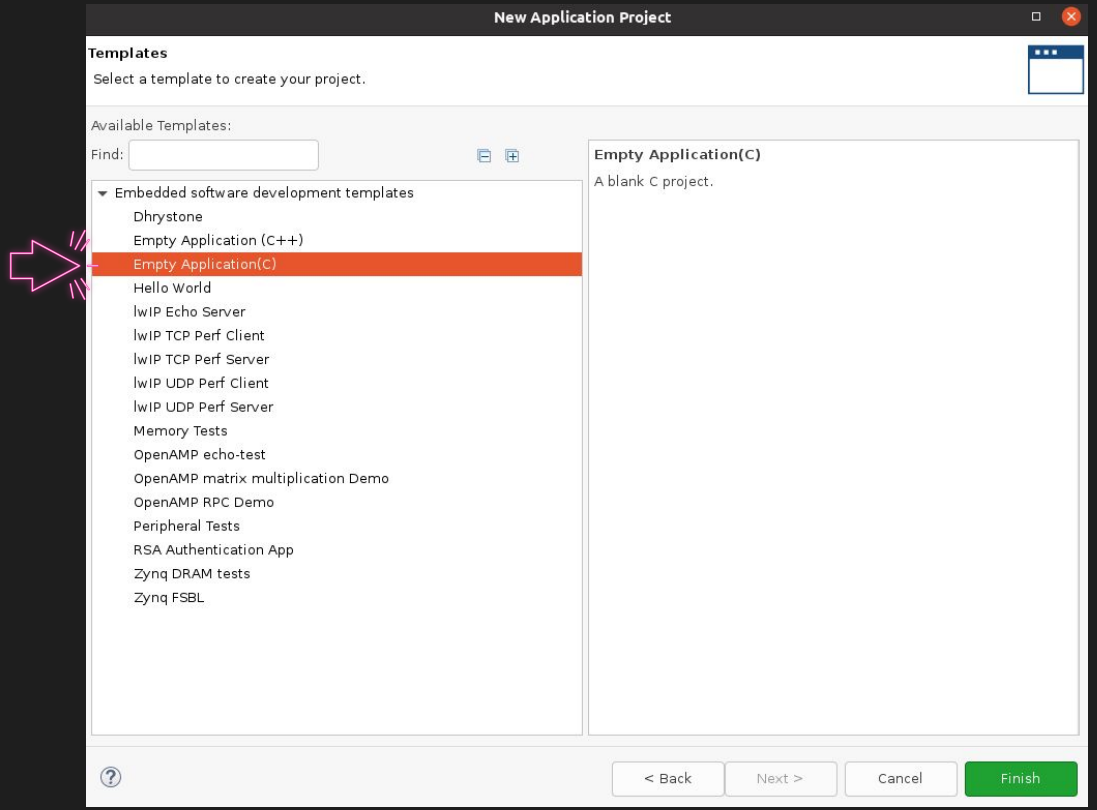
We will not use any OS in this project. Our application will be running on a bare metal processor (ARM cortex A9)



Selecting a template

Vitis provides a list of template projects that can be used to learn about Vitis project structures.

For this project we will use an Empty Application (C) and will add the source code manually.



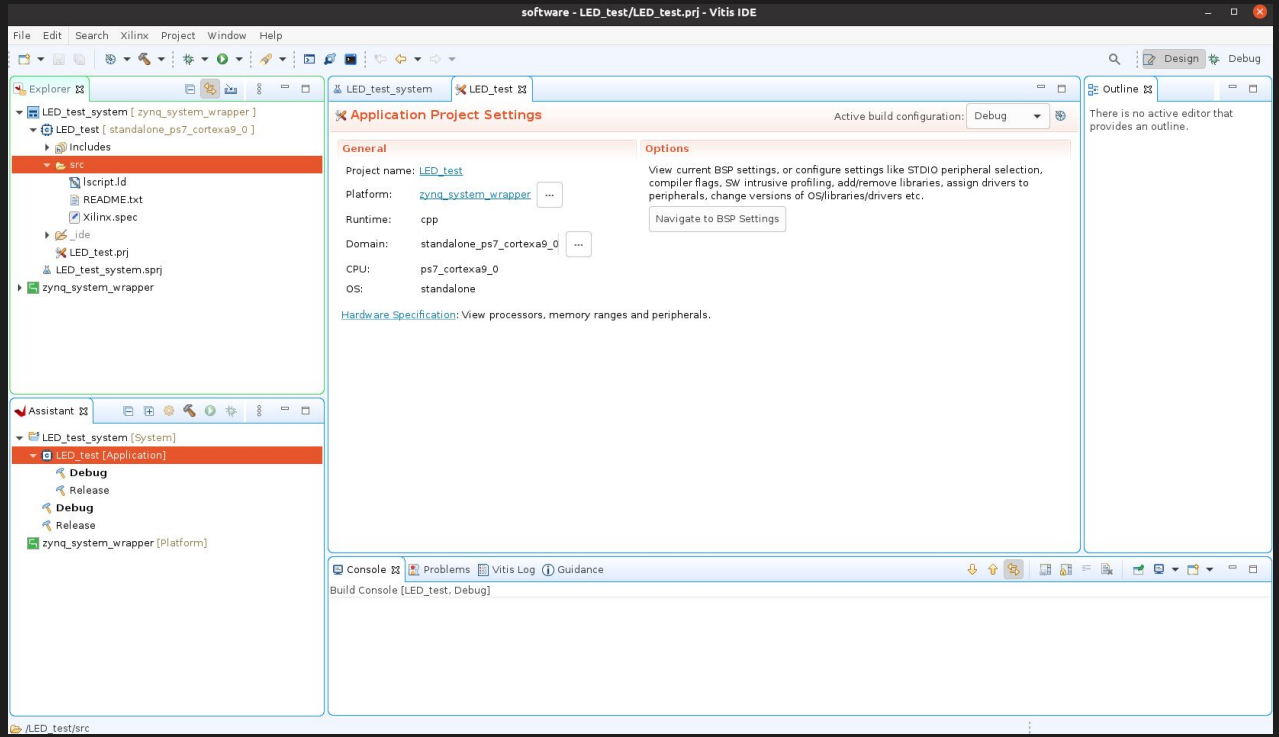
Vitis main layout

The vitis main layout is shown here.

Use the Explorer window to navigate to different files.

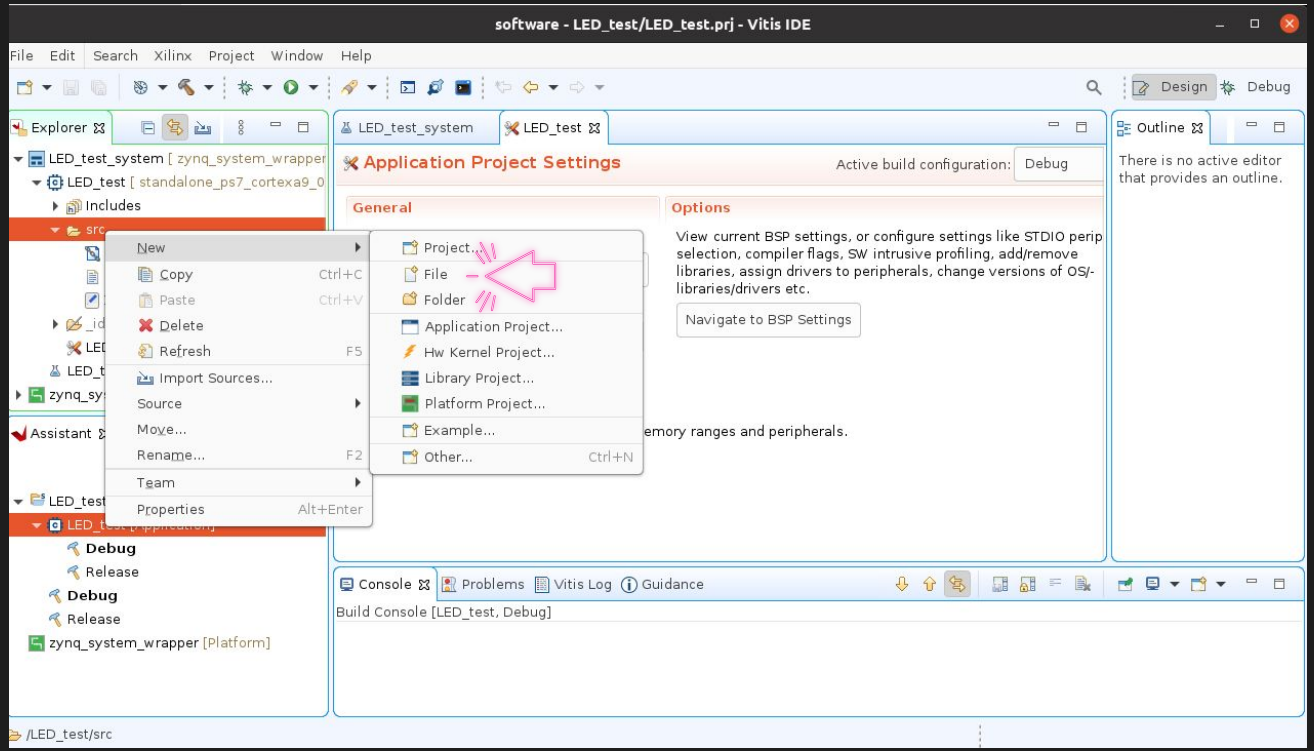
Use the Assistant window to build your projects.

Use the workspace window to view source codes or navigate to different Board Support Package settings.



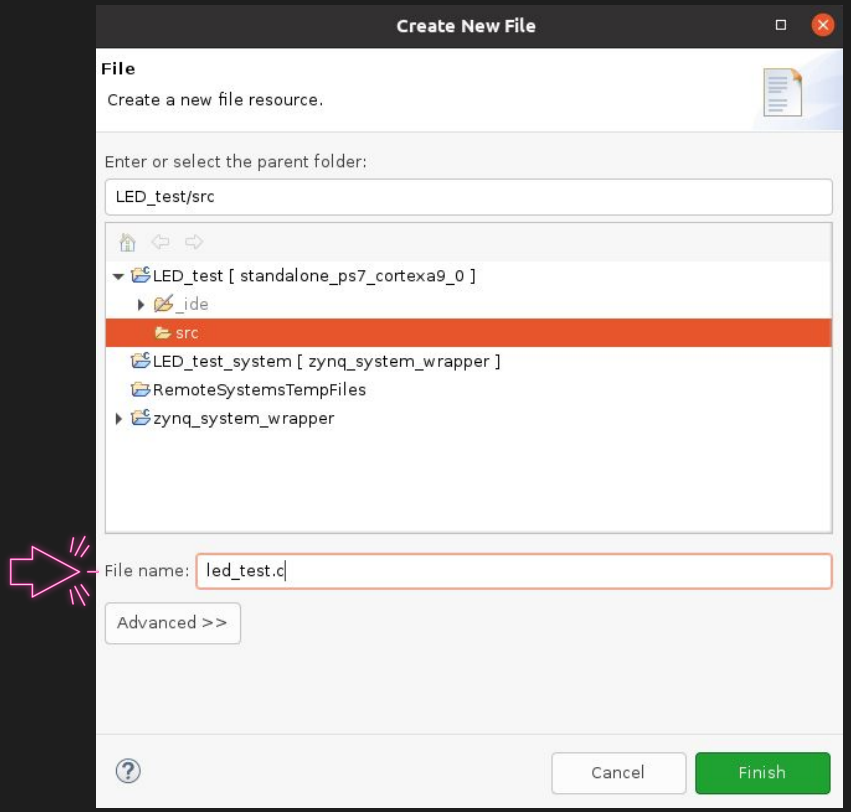
Create a file

To add our source code create a C file in the src folder.



Name the source file

Name the source file
led_test.c



Led_test.c (part 1)

```c
/* Include Files */
#include "xparameters.h"
#include "xgpio.h"
#include "xstatus.h"
#include "xil_printf.h"

/* Definitions */
#define GPIO_DEVICE_ID  XPAR_AXI_GPIO_0_DEVICE_ID /* GPIO device that LEDs are connected to */
#define LED 0x9          /* Initial LED value - X00X */
#define LED_DELAY 10000000       /* Software delay length */
#define LED_CHANNEL 1        /* GPIO port for LEDs */
#define printf xil_printf       /* smaller, optimised printf */

XGpio Gpio;           /* GPIO Device driver instance */
```

GPIO_DEVICE_ID is defined as XPAR_AXI_GPIO_0_DEVICE_ID. The value of XPAR_AXI_GPIO_0_DEVICE_ID can be found by opening the file, xparameters.h, which is automatically generated by Vivado IDE when exporting a hardware design to the Vitis. It contains definitions of all the hardware parameters of the system.

Led_test.c (part 2)

```
int LEDOutputExample(void)
{
   volatile int Delay;
   int Status;
   int led = LED; /* Hold current LED value. Initialise to LED definition */
       /* GPIO driver initialisation */
       Status = XGpio_Initialize(&Gpio, GPIO_DEVICE_ID);
       if (Status != XST_SUCCESS) {
           return XST_FAILURE;
       }
       /*Set the direction for the LEDs to output. */
       XGpio_SetDataDirection(&Gpio, LED_CHANNEL, 0x0);
       /* Loop forever blinking the LED. */
           while (1) {
               /* Write output to the LEDs. */
              XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, led);
               /* Flip LEDs. */
               led = ~led;
               /* Wait a small amount of time so that the LED blinking is visible. */
               for (Delay = 0; Delay < LED_DELAY; Delay++);
           }
       return XST_SUCCESS; /* Should be unreachable */
}
```

This is a function provided by the GPIO device driver in the file xgpio.h. It initialises the XGpio instance, Gpio, with the unique ID of the device specified by GPIO_DEVICE_ID.

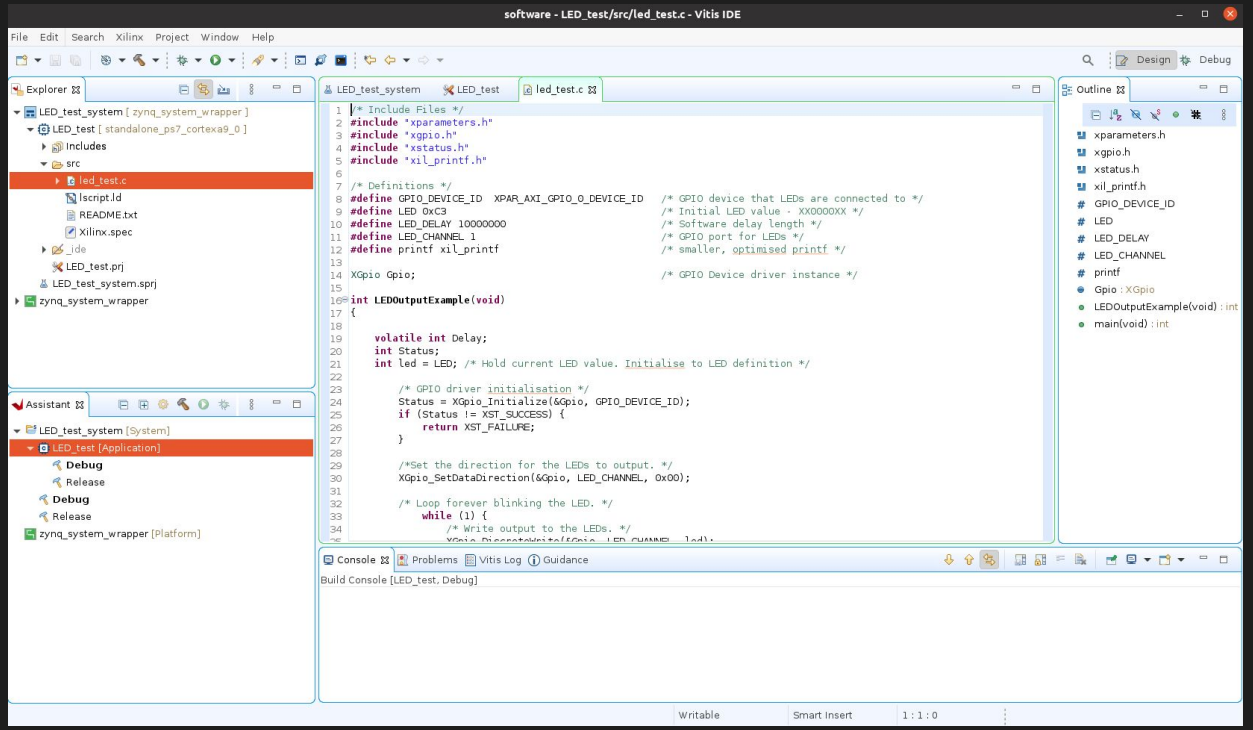
This function is also provided by the GPIO device driver, and sets the direction of the specified GPIO port. As we are specifying the LEDs in this case, it is specifying an output. Bits set to ‘0’ are output, and bits set to ‘1’ are input. As there are 4 LEDs, by setting the LED channel direction to a value of 0x00, or 00000000 in binary, we are setting all LEDs as outputs.

Led_test.c (part 3)

```c
/* Main function. */
int main(void){

   int Status;

   /* Execute the LED output. */
   Status = LEDOutputExample();
   if (Status != XST_SUCCESS) {
       xil_printf("GPIO output to the LEDs failed!\r\n");
   }

   return 0;
}
```

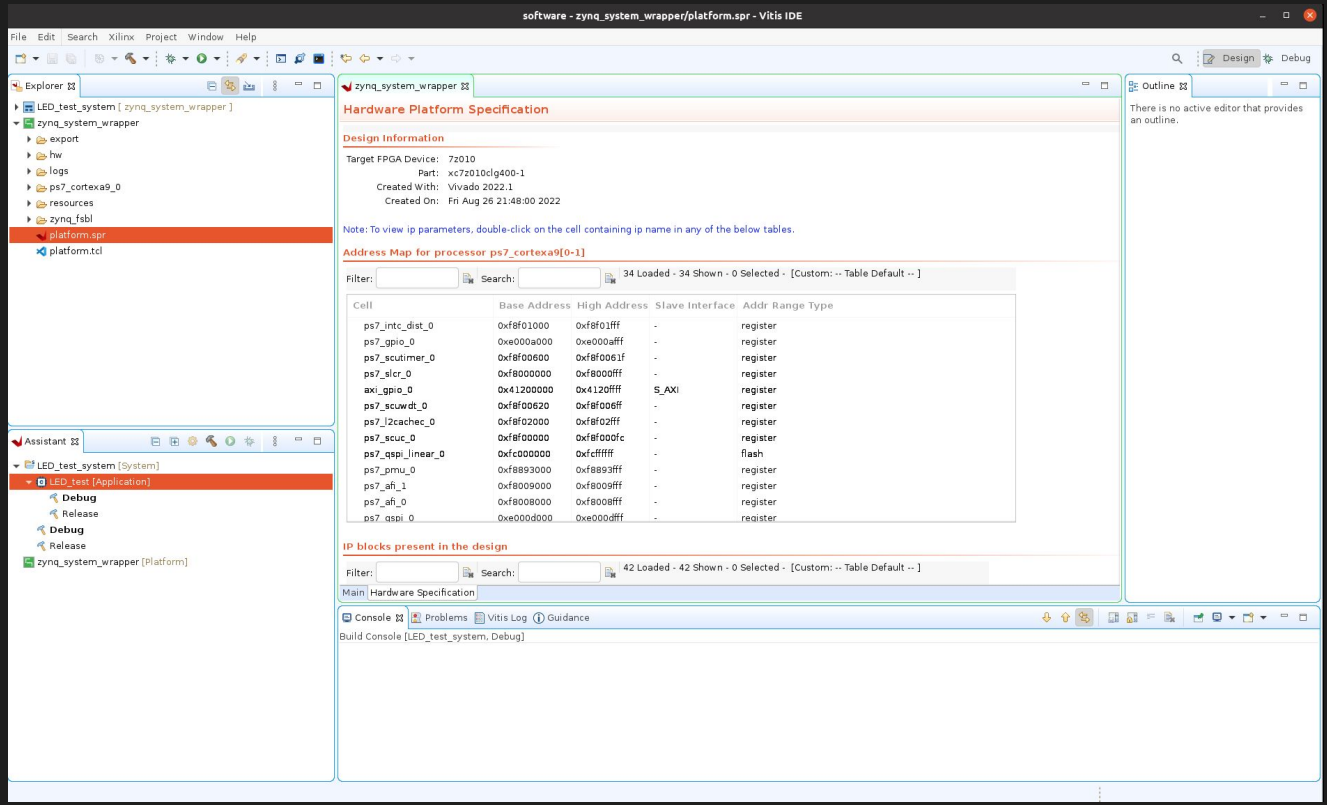
led_test.c

The final file should look like the following.



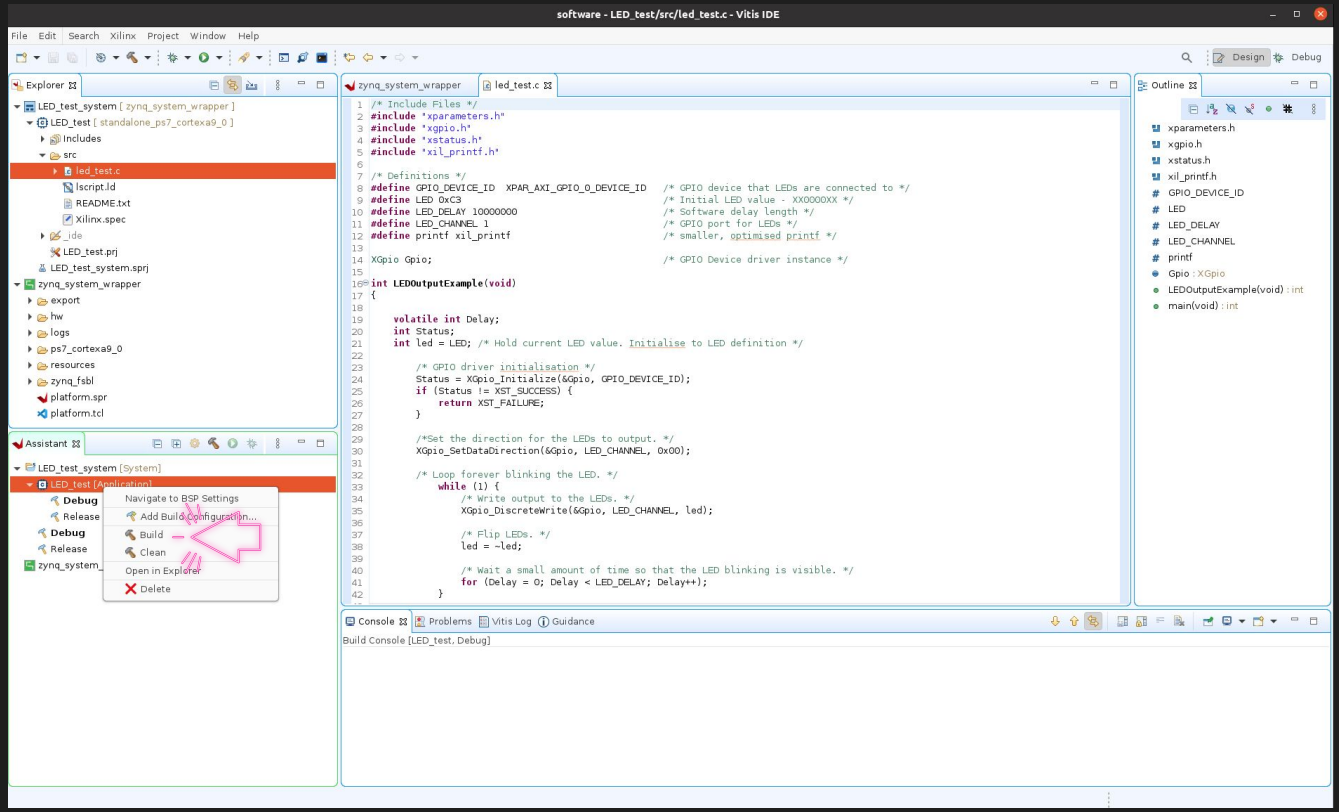
Platform specification

You can look at the peripheral and drivers information in the platform.spr file.



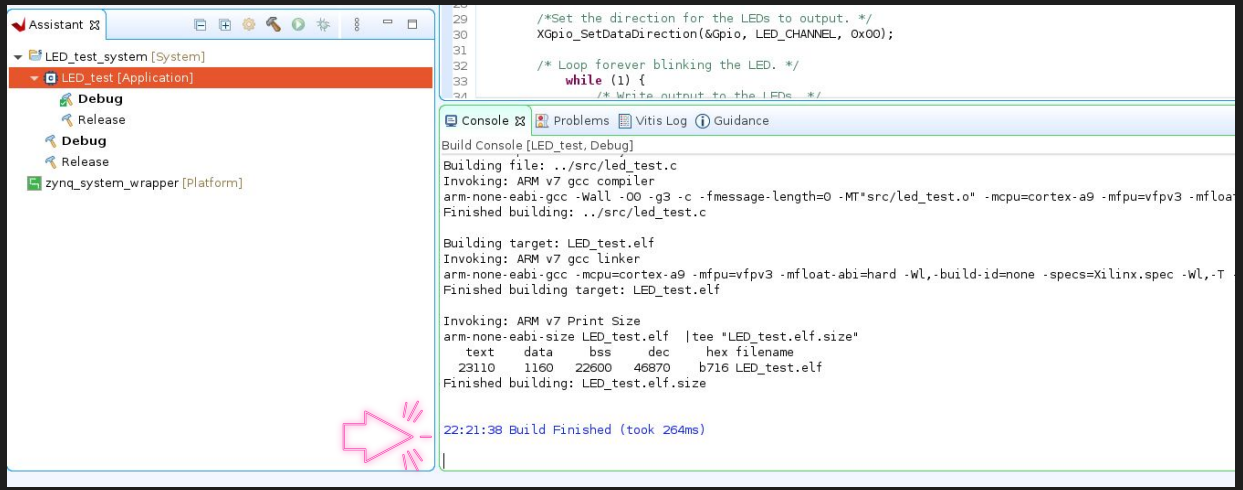
Building the project

Build the project from the Assistant window.



Confirm successful completion

If everything goes as expected you should see a build finished message in the Vitis console window.



Physical connection

Now it's time to
program the board!

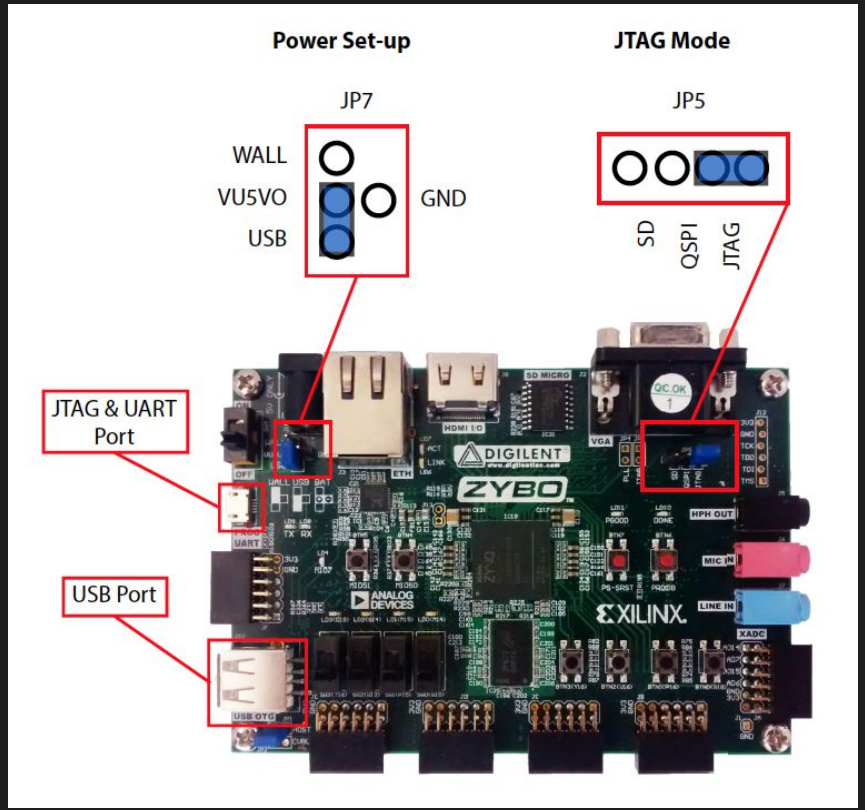
You need a micro USB
B to USB A cable

Jumper config

Connect the cable to port J12 (shown as JTAG & UART Port)

Make sure JP5 jumper configuration is on the most right pin and 2nd most right pin of port.

Make sure the JP7 jumper configuration is as shown in the figure to power up the board from USB.

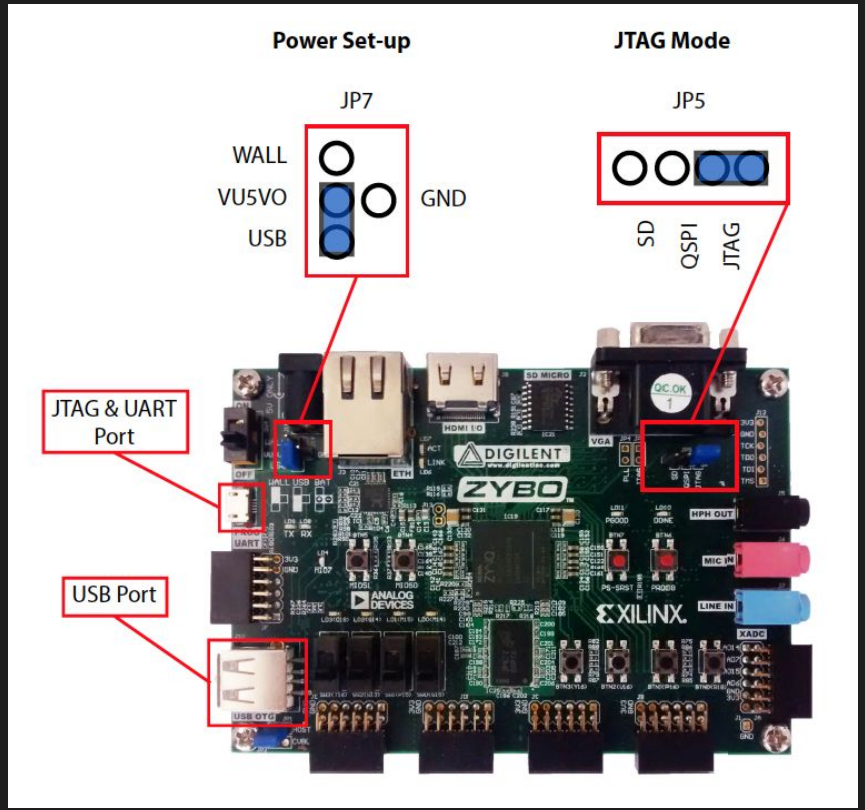


Jumper config

If the JP5 jumper is on pin 0 and 1 the device will be programmed using JTAG.

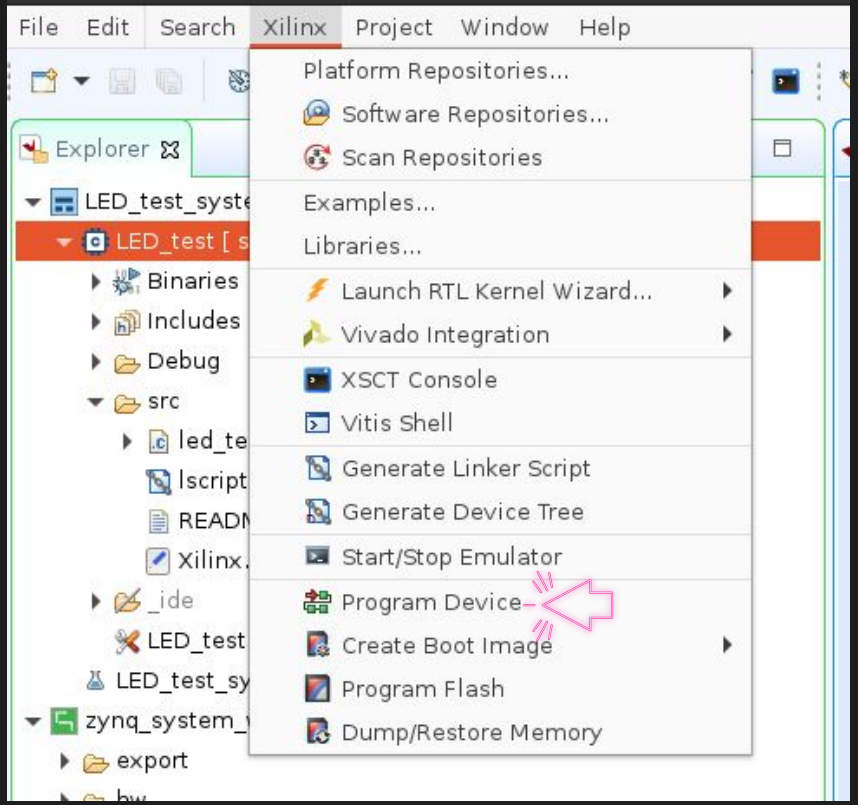
If it is connected to port 1 and 2 the device will boot from QSPI flash on board.

If the jumper is connected to port 2 and 3 the device will boot from the SD card.



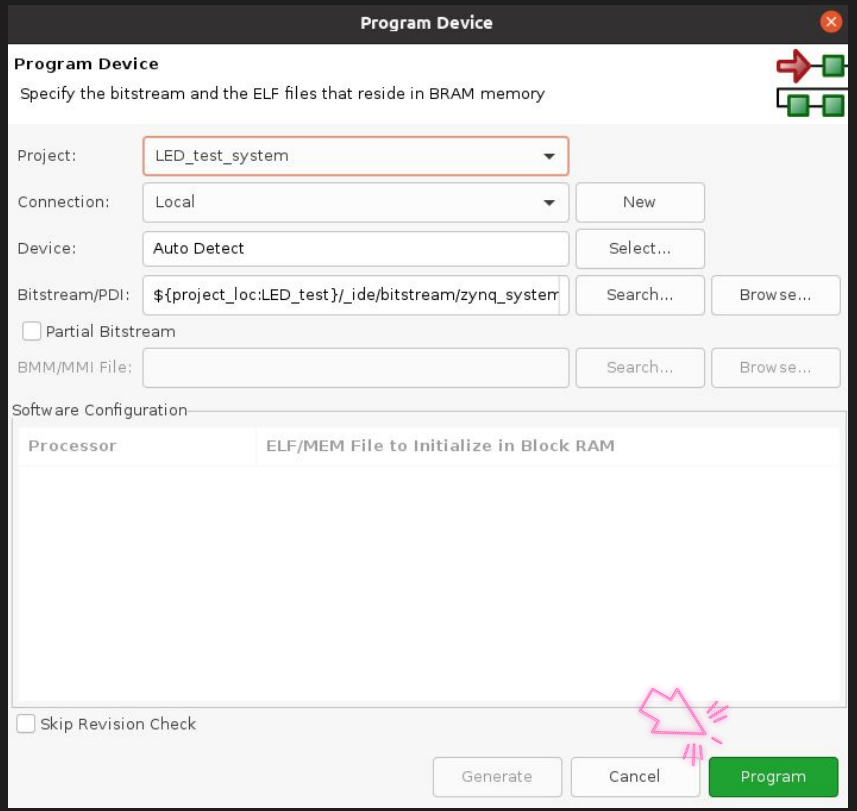
Program Device

From Xilinx menu select Program Device



Program Device

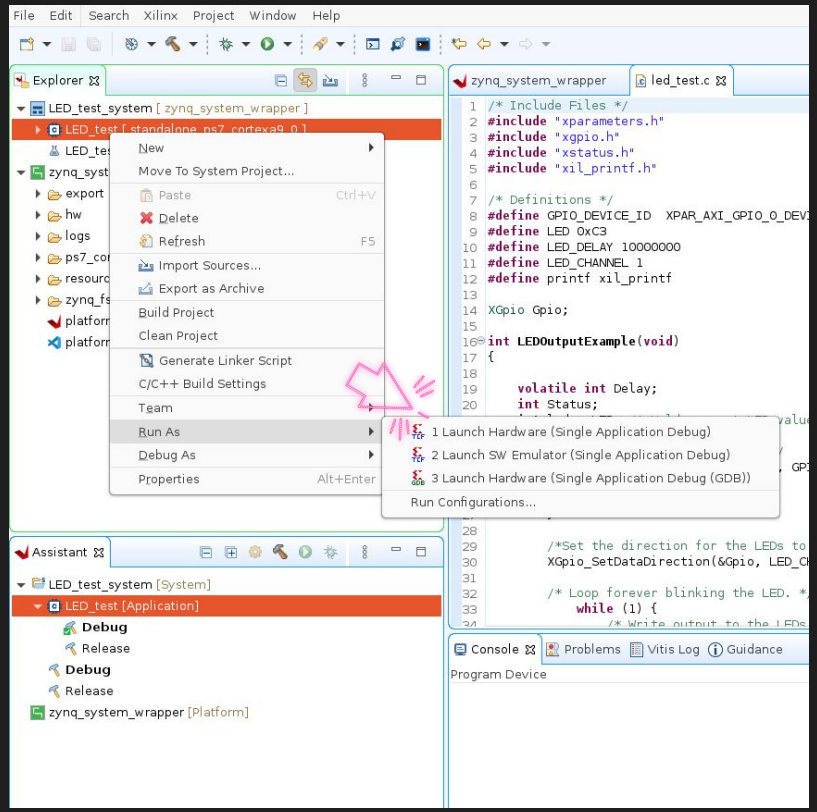
Press the Program button to Download the bitstream and Software binary files to the board.



Run the application

Select the project LED_test in Project Explorer. Right-click and select Run As > Launch on Hardware (GDB).

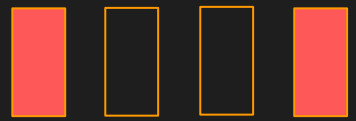
After a few seconds the LEDs on the ZedBoard should begin to flash between the states highlighted



LED Flashing States

You should see the LEDs flash as shown in this figure.

State A:



State B:

Congratulations!

You have successfully created and executed your first software application on the Zynq processing system.

Source Control Vitis using Git

Cal State Northridge - Saba Janamian

Creating .gitignore file

The .gitignore file is a text file that tells Git which files or folders to ignore in a project.

In the root directory of your project (not Vitis project) create a .gitignore file and place the following lines in it. Note the name of the folder should match your project name.

.gitignore

```
hardware/first_zynq_led/*

software/*

!software/LED_test

software/LED_test/*

!software/LED_test/src
```

Check the status of your git repository

List current unstaged file and folders: `git status -u`

```
sa@wo:[/h/s/w/first_zynq_design ]$ git status -u
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        hardware/export/zynq_system_wrapper.xsa
        hardware/first_zynq_led.tcl
        software/LED_test/src/README.txt
        software/LED_test/src/Xilinx.spec
        software/LED_test/src/led_test.c
        software/LED_test/src/lscript.ld
```

As shown here we are just keeping track of important files. The rest of auto-generated files are ignored.

add, commit

Add files to git git : `git add .`

Commit your changes: `git commit -m “added final design”`

Push your changes to the remote repository: git push origin master

Check remote

Check your remote repository address: `git remote -v`

If nothing shows up it means you need to add the url to the remote repo

```
git remote add origin https://github.com/user/repo.git
```

Push your changes

Push your changes to the remote repository:

```
git push origin master
```

THANKS!

Do you have any questions?