

Lab 23: MicroBlaze processor on Basys-3

In this lab exercise, we will implement a MicroBlaze soft processor on the Artix-7 FPGA of the Basys-3 board, and connect it to some GPIOs (Push Buttons, LEDs).

We will then program the processor, to control the GPIOs, using the Vitis Unified Software Platform.

Exercise 1. Create the HDL Design

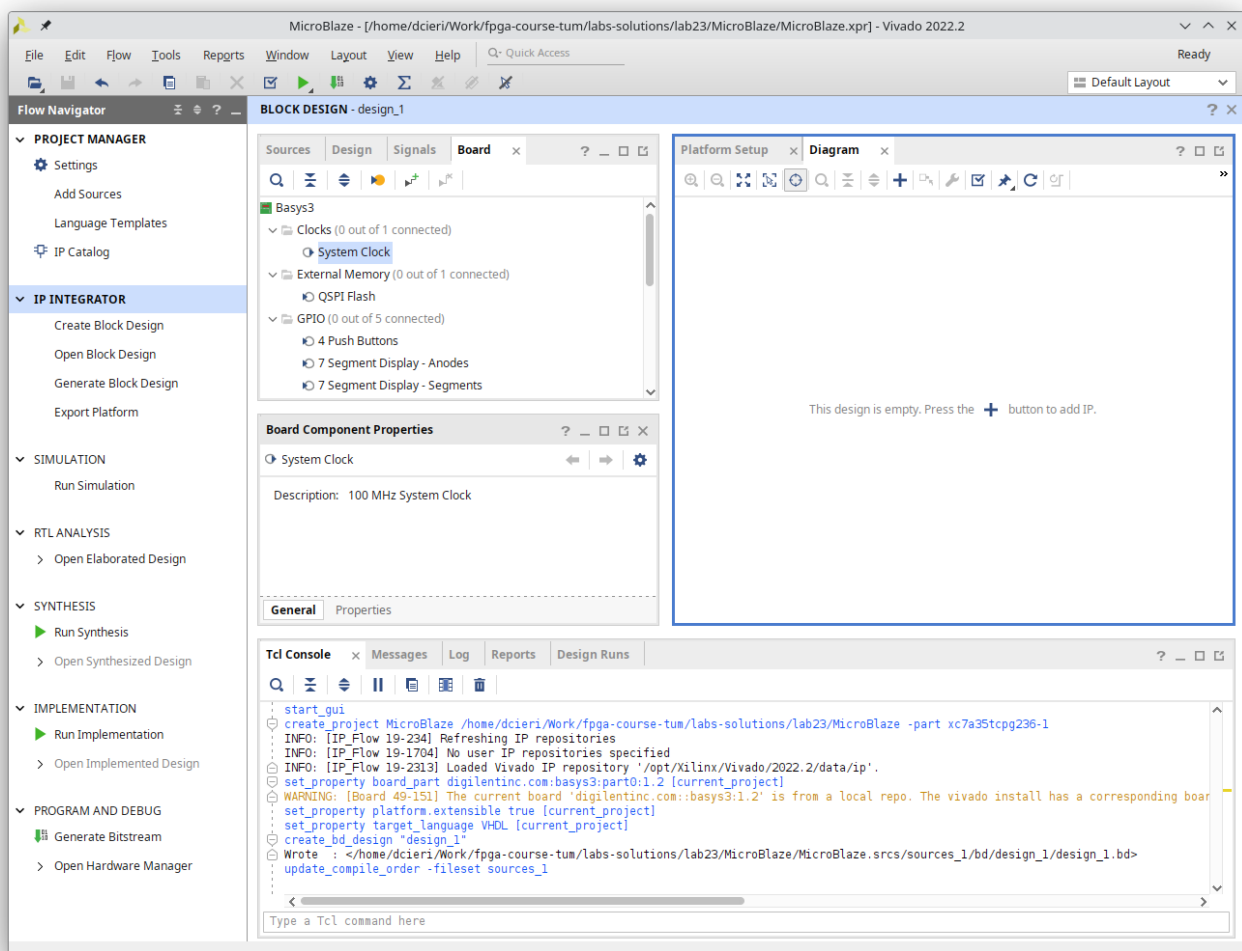
Go to `~/labs/lab23` and create a new project for the Basys-3 board, called *MicroBlaze*. In the *Project Type* section, choose RTL Project, and select:

- In the Add Constraints page, add `src/basys3.xdc`
- Project is an extensible Vitis platform

Once the project is generated, create a new block design, using the command on the left side bar.

Once you are in the Block Design context, click on the *Board* tab on the left window box.

In this window, we can see all the available interfaces for the Basys-3 board. Dragging them into the *Diagram* window, automatically constraints the pins, without the need of constraint file.

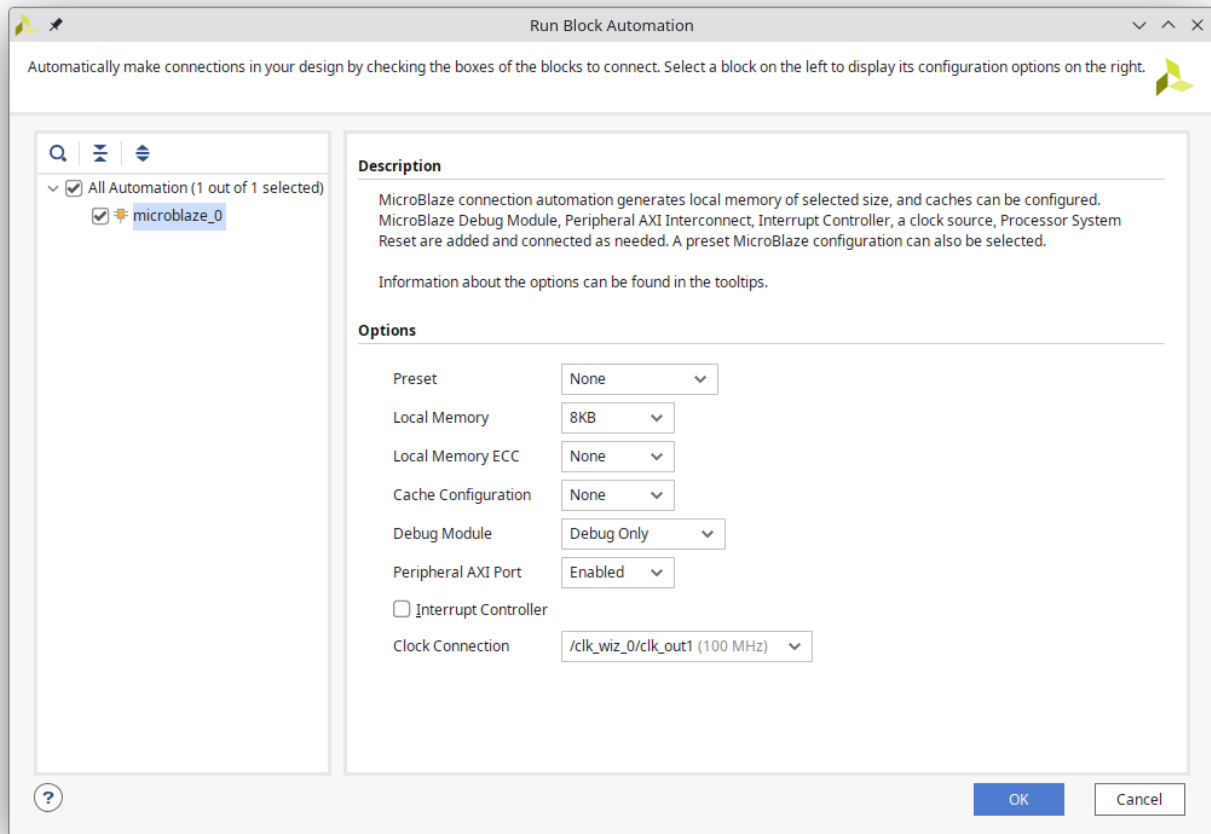


Drag the system clock into the design window. Click then on *Run Connection Automation*, and confirm with OK in the pop-up window.

Double click on the *clk_wiz_0* block, to confirm that the output clock *clk_out1* is 100 MHz.

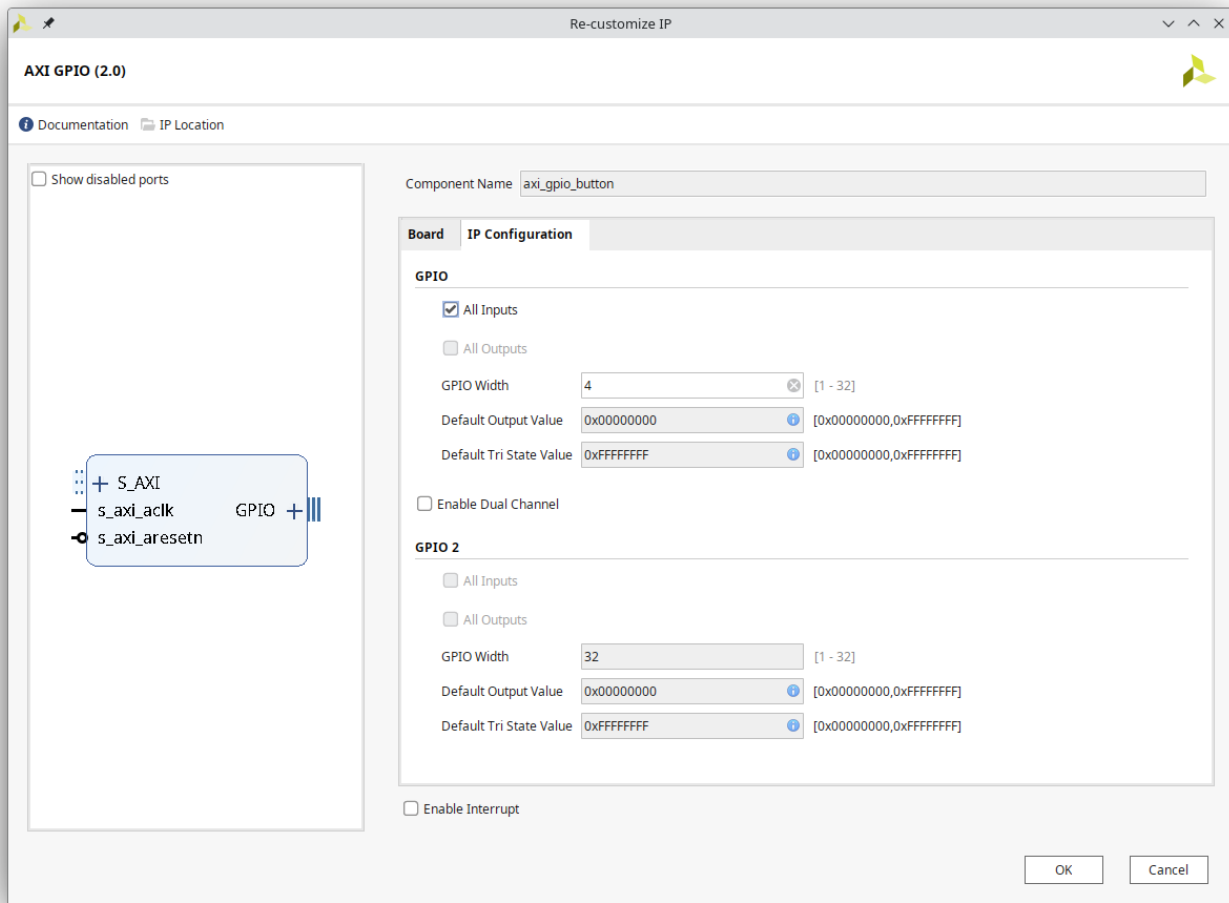
Click now on the plus button and add the MicroBlaze IP. Click then on *Run Block Automation*.

A pop-up window will appear that let us configure the MicroBlade processor.



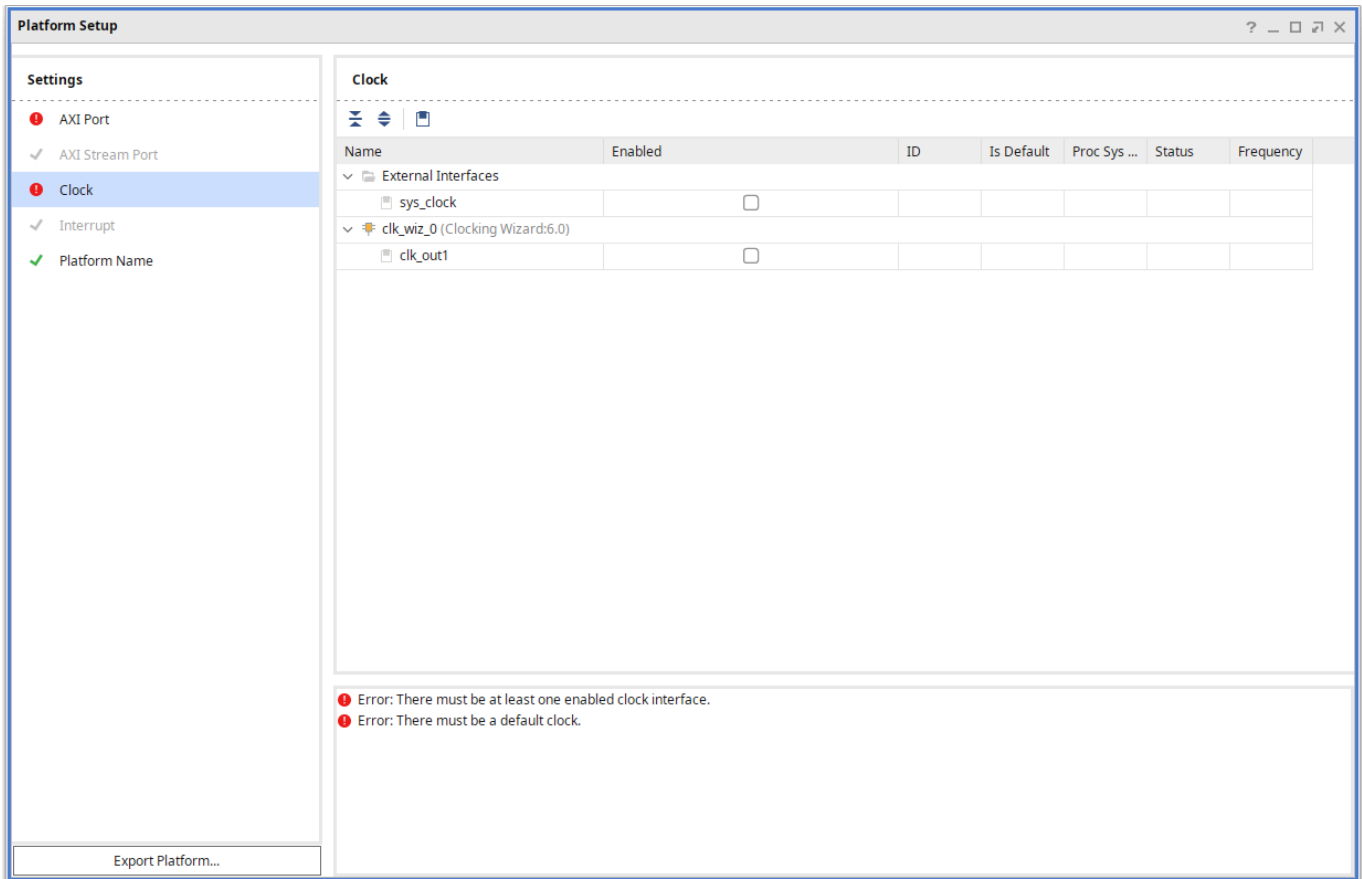
Set the local memory to 32kB and click OK. Now drag to the diagram the 16 LEDs. This will create an *AXI GPIO* block, called *axi_gpio_0*. Click on the new block, and rename it in the properties box *axi_gpio_led*.

Now click on the plus button, and add another *AXI GPIO* block. Double click on it, to open the IP configuration page. Go to the *IP Configuration* tab, select *All Inputs*, set the Width to 4, and disable the dual Channel. Confirm with OK. Then rename the IP, *axi_gpio_btn*.




Back on the Block Diagram, select the GPIO port of the `axi_gpio_button`, right click and select *Make External*. This should connect the port with an external interface (GPIO_0 by default). Click on the interface, and rename it `BTN`.

Click again on *Run Block Automation* and select all the blocks. Now open the *Platform Setup* tab. You should see a couple of Warning point in the left sidebar.

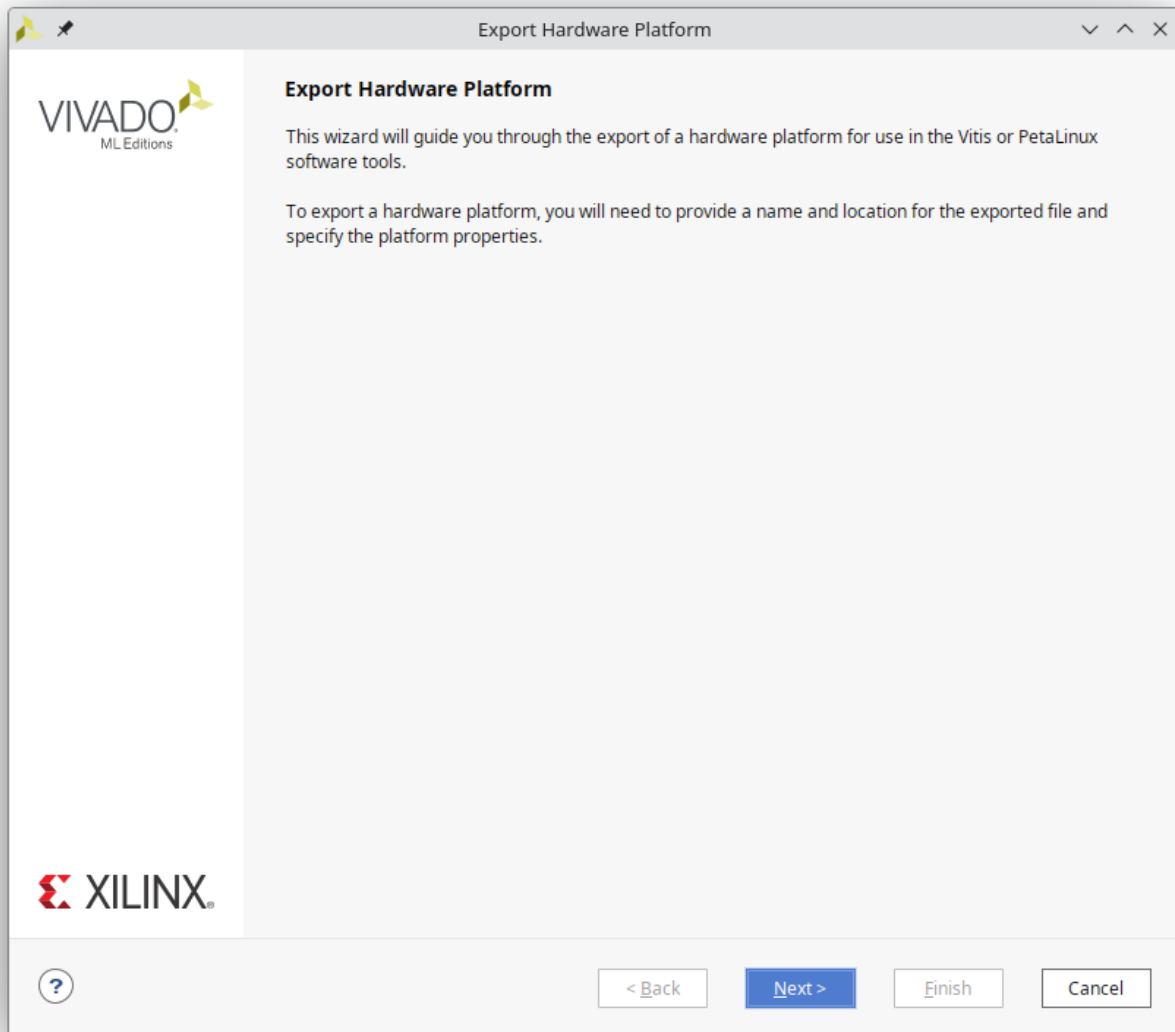


Select Clock, then enable the `clk_out1` signal, and set it as default. Ignore the error on the AXI port, since we don't have any AXI ports that can be made available in our design.

Save the design, and click on the *Validate* icon  to verify.

Open now the *Sources* tab, from the project manager. Right click on the block design file (by default `design_1`), and select *Create HDL Wrapper*. Once the wrapper is generated, click on *Generate Bitstream*.

Once the bitstream is generated, we have to export the hardware platform files. Click on *File > Export > Export Platform*. This tool will export the platform, that could be later used inside Vitis, to program our embedded processor.

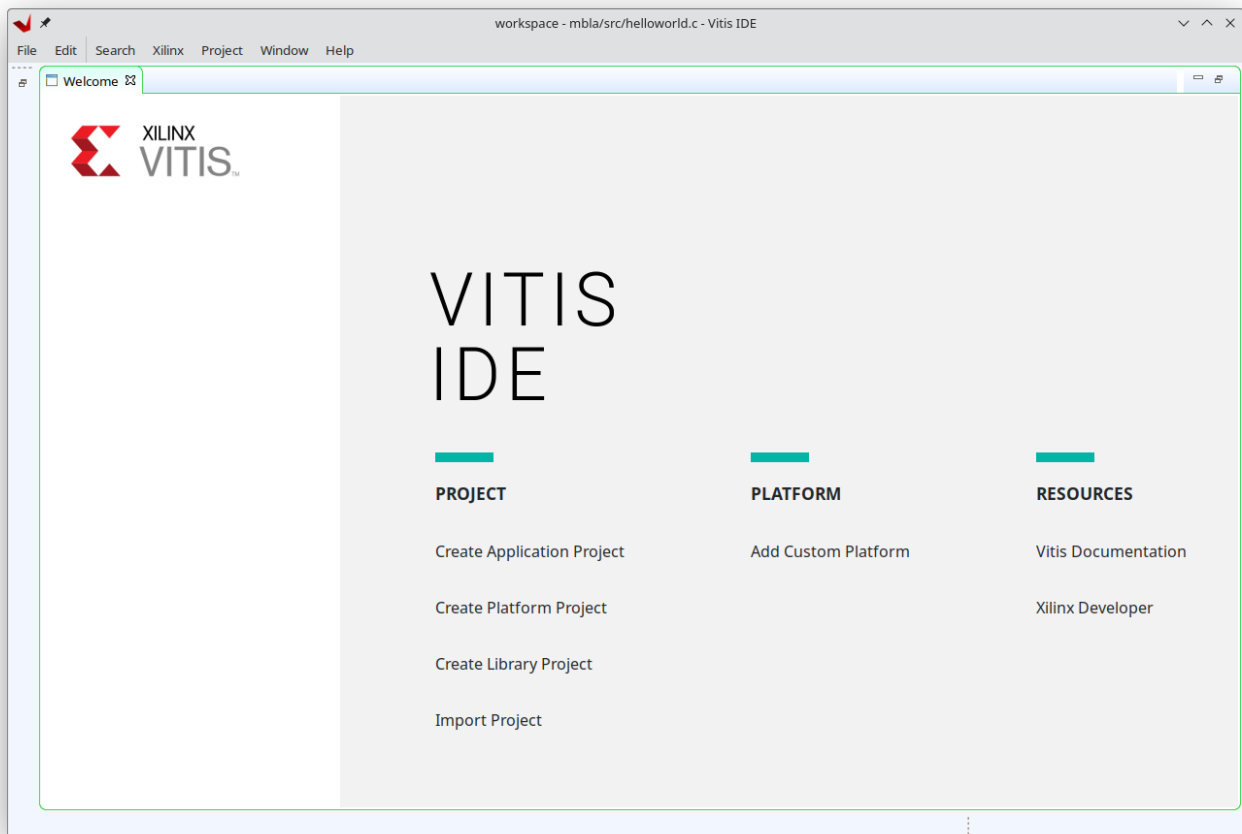


Configure it as follow:

- In the *Platform Type* Window, select Hardware
- In the *Platform State* window, select *Pre-synthesis, and tick the *Include bitstream* box
- Give a reasonable name in the *Platform Properties* window, and keep the rest as default
- Continue and click on Finish

Exercise 2. Program the MicroBlaze processor with Vitis

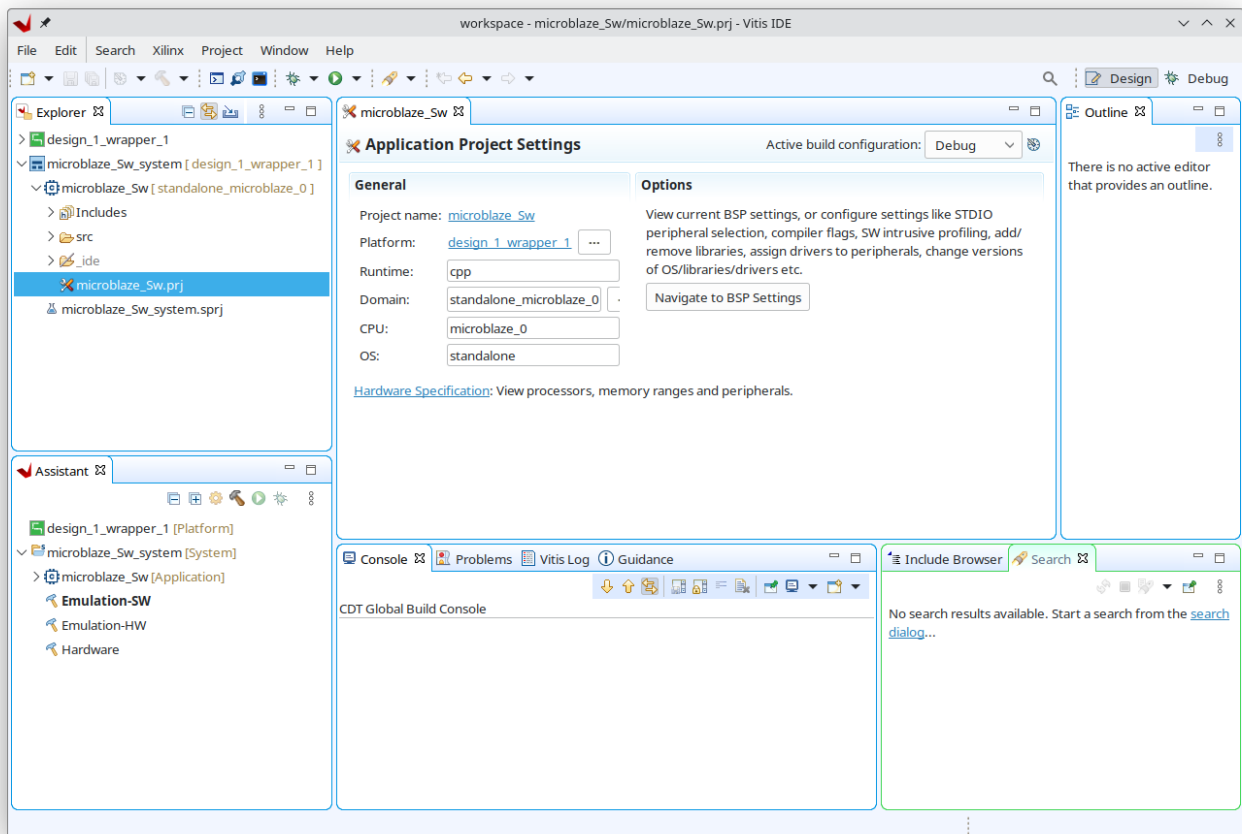
Click now on the toolbar of Vivado **Tools > Launch Vitis**. If this is the first time you launch Vitis, you should be prompted with the welcome page.



Select *Create Application Project*. This will open another wizard window, that guides you in the creation of an application. Configure it with the following instructions:

1. Click Next in the *Create a New Application Project* Page.
2. In the *Platform* page, select the tab *Create a new platform from hardware (XSA)*, click on Browse, and select the xsa file, contained in the Vivado project folder in `~/labs/lab23`.
3. In *Application Project Details*, provide a name for your project, and click Next
4. In *Domain* keep the default and click Next
5. In the Templates, choose *Hello World* and click Finish.

You should now be in the main project window of Vitis.



In the Explorer box on the left, you should see our hardware platform, imported from Vivado, and the new application we just created.

If you expand the `src/` folder of your application, you should see the actual source files. Open the `helloworld.c` file and have a look at its content.

This is a simple C code, that will print "Hello World" on screen.

Now connect the board to your pc, and open a minicom terminal.

```
minicom -D /dev/ttyUSB1 -b 9600
```

Push a button on the board, to verify that the connection works. The default firmware should print something.

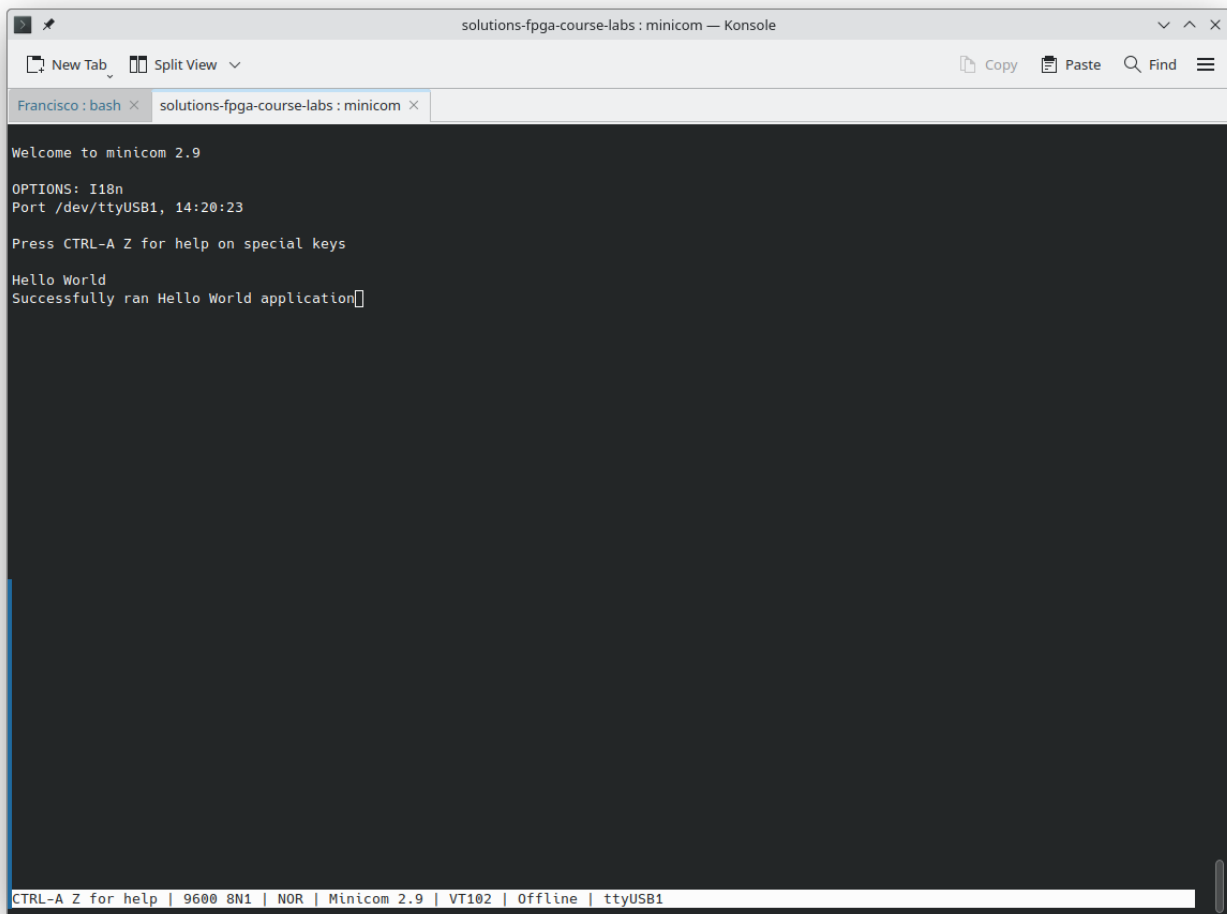
Keep it the terminal open and visible, and go back to Vitis.

In the bottom left box (*Assistant*), expand your application, right click on *Hardware*, and select *Build*.

Once it is done, click with the right button on your application, and select *Run > Launch Hardware*.

If everything went well, you should see the Hello World string appearing in the minicom terminal.

If you get an error, close the pop-up window, power-down and power-up the board, and try again.



```
solutions-fpga-course-labs : minicom — Konsole
New Tab Split View
Francisco: bash x solutions-fpga-course-labs: minicom x
Welcome to minicom 2.9
OPTIONS: I18n
Port /dev/ttyUSB1, 14:20:23
Press CTRL-A Z for help on special keys
Hello World
Successfully ran Hello World application
CTRL-A Z for help | 9600 8N1 | N0R | Minicom 2.9 | VT102 | Offline | ttyUSB1
```

Exercise 3. Drive the GPIO with MicroBlaze

In Vitis, click on *File->New->Application Project* to create another application.

- In the *Platform* page, select the same platform from before
- Give it another name in the *Application Project Details* page
- In the *Templates* page, select Empty Application C

Now select the `src` folder of your new Application in the *Explorer* window, right click and select Import Sources.

Select then the folder `lab23/src`, and import the `gpio.c` file.

Have a look, at this file, to understand what we are doing.

The script initialises the LED / Button GPIOs, using the addresses in the AXI memory, which have been defined in our Vivado Block Diagram and exported to Vitis. (`XGpio_CfgInitialize` function)

With the `XGpio_SetDataDirection` function, we declare if the GPIO is an input (1) or output interface (0).

In the `while` loop, we continuously check for the values of the buttons, and if any of them is pushed, we switch on the first four LEDs.

Build now the project, as we did before, and run it on the hardware, to verify that works.

Modify now the code, to implement the following behaviour:

- If button 0 (BTN UP) is pushed, switch on the first 4 leds
- If button 1 (BTN RIGHT), leds 4-7
- If button 2 (BTN DOWN), leds 8-11
- If button 3 (BTN LEFT), leds 12-15