SYSTEM ON CHIP – LAB 1

PREREQUISITES

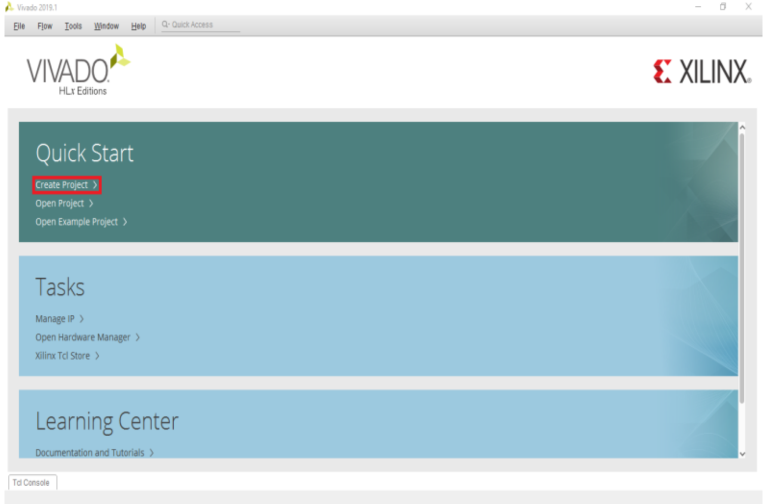
- Knowledge on SoCs (see lessons material), more specifically:
- Basic familiarity with the Vivado environment
- Basic familiarity with the Diligent Zybo Z7 development board
- Basic experience with the VHDL programming language
- Basic experience with the C programming language

OBJECTIVES

- SoC creation
- GPIO use
- UART use

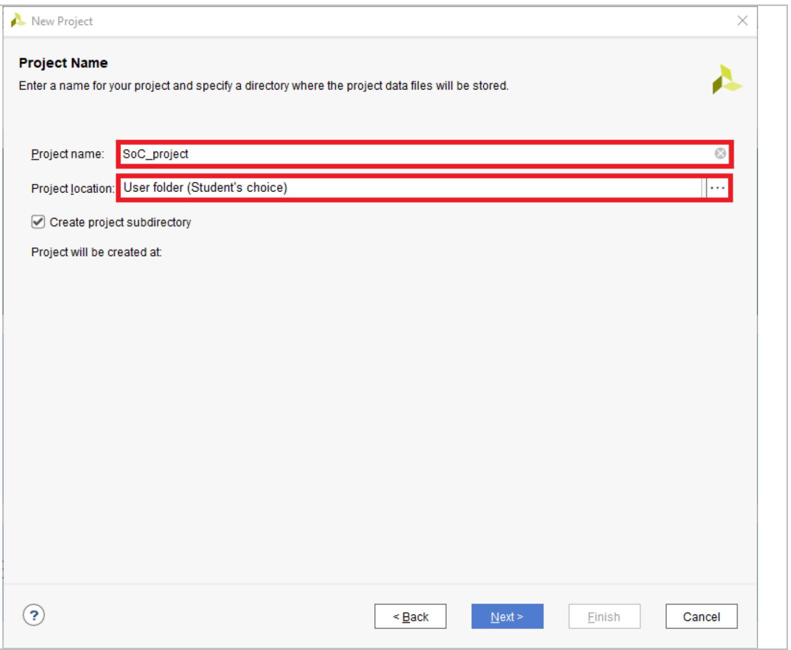
SOC CREATION WITH VIVADO IP INTEGRATOR

The objective of this first section is to setup the Zynq IP along with some peripherals in Vivado. The circuit will then be implemented on a Zynq 7000 SoC on a Digilent Zybo board. This section will use the main features of the IP Integrator design flow of the Vivado environment.

1. Open Vivado and **create a new project.**	

2. The first page of the wizard summarizes the steps involved in creating a project. Click **Next**, then choose a **name** and a **location path** for your project. Vivado will use these inputs when generating its folder structure.

Do NOT use spaces in the project name or location path. This will cause problems with Vivado. Instead use an underscore, a dash…



New Project

Project Name

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

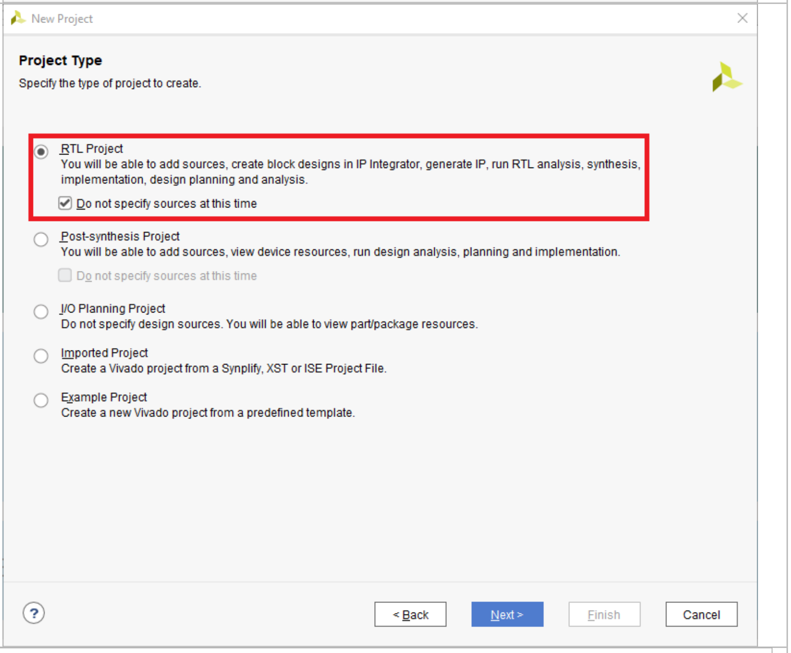
Project name: SoC_project

Project location: User folder (Student's choice)

☑ Create project subdirectory

Project will be created at:

3. At the Select Project Type screen, choose **RTL Project**. Click **Next**.



New Project

Project Type

Specify the type of project to create.

● 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

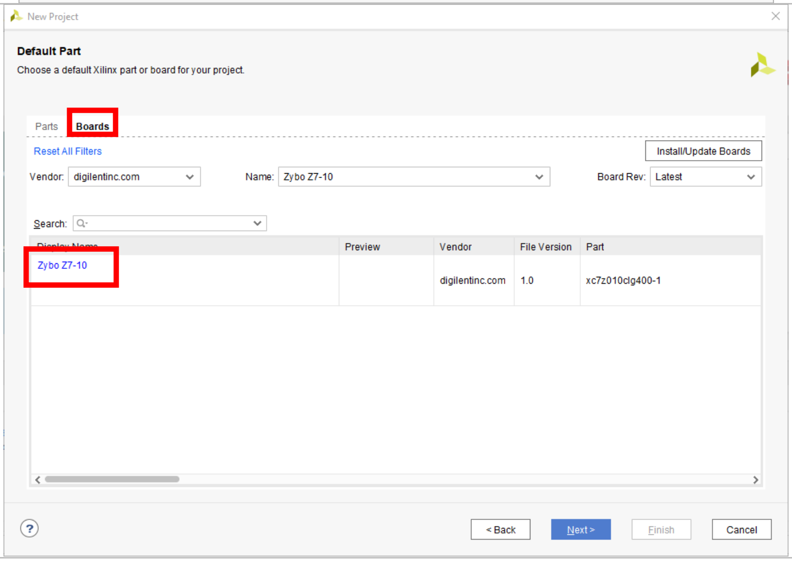
○ Post-synthesis Project
You will be able to add sources, view device resources, run design analysis, planning and implementation.
□ Do not specify sources at this time

○ I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

○ Imported Project
Create a Vivado project from a Synplify, XST or ISE Project File.

○ Example Project
Create a new Vivado project from a predefined template.

4. Choose the "Board" tab and select the Zybo Z7-10 board. All the constraints (buttons, switches, LEDs, and others peripherals) are automatically set.



New Project

Default Part

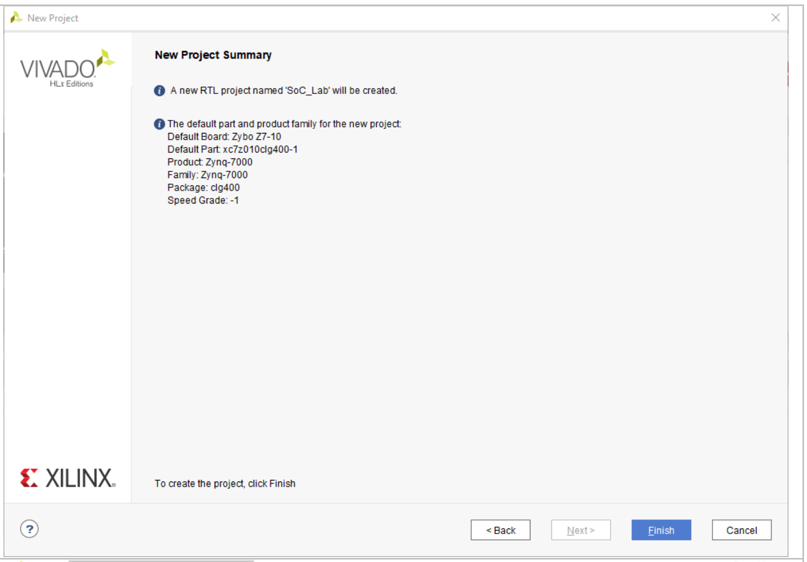
Choose a default Xilinx part or board for your project.

Vendor: digilentinc.com Name: Zybo Z7-10 Board Rev: Latest

Search: Q-

Display Name	Preview	Vendor	File Version	Part
Zybo Z7-10		digilentinc.com	1.0	xc7z010clg400-1

5. This screen summarizes selections chosen in the previous screens. Click **Finish** to finish opening the new project.



New Project

New Project Summary

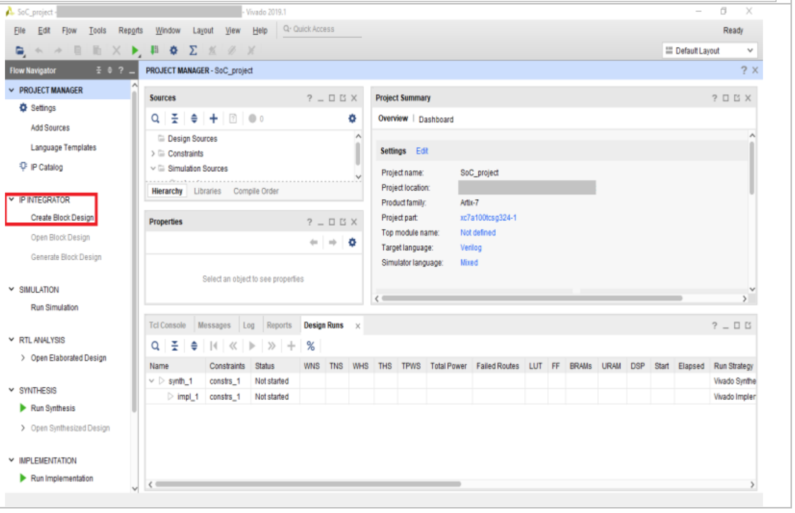
A new RTL project named 'SoC_Lab' will be created.

The default part and product family for the new project:
Default Board: Zybo Z7-10
Default Part: xc7z010clg400-1
Product: Zynq-7000
Family: Zynq-7000
Package: clg400
Speed Grade: -1

To create the project, click Finish

6. This guide will be exclusively using the IP Integrator tool, which can be opened from the **Flow Navigator** on the right side of the window. Expand the **IP Integrator** tab and select **Create Block Design**.

In the dialog box, give the block design a name. The directory location is where the block design will be stored. It is recommended to leave it as **<Local to Project>**. Make sure that **Specify source set** is set to **Design Sources**.



SoC_project - [redacted] - Vivado 2019.1

PROJECT MANAGER - SoC_project

Project Summary

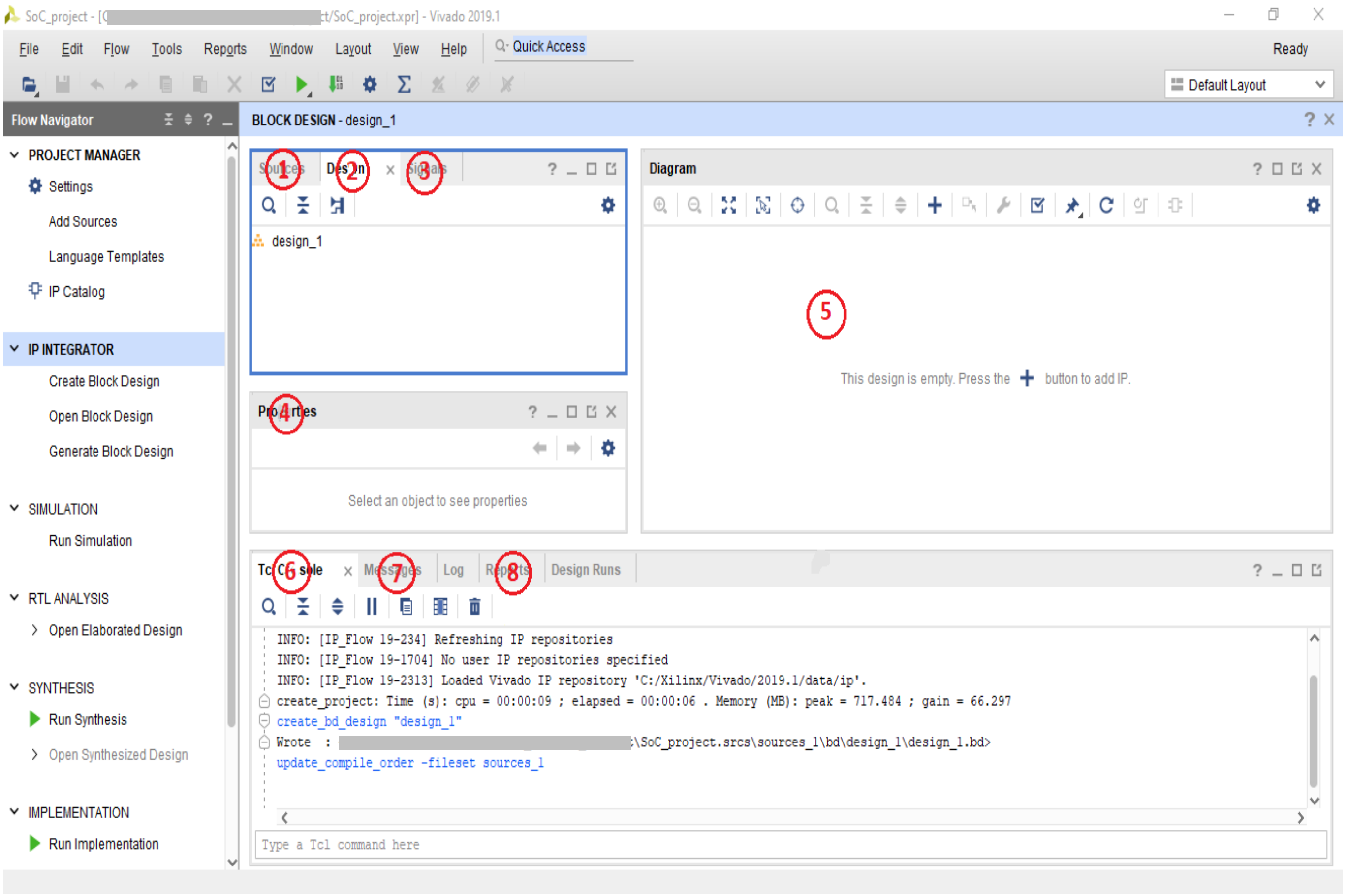
Settings Edit

Project name: SoC_project
Project location: [redacted]
Product family: Artix-7
Project part: xc7a100tcsg324-1
Top module name: Not defined
Target language: Verilog
Simulator language: Mixed

Select an object to see properties

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy
synth_1	constrs_1	Not started															Vivado Synthe
impl_1	constrs_1	Not started															Vivado Implem

IP Integrator Tools Interface



This design is empty. Press the + button to add IP.

Select an object to see properties

```
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository 'C:/Xilinx/Vivado/2019.1/data/ip'.
create_project: Time (s): cpu = 00:00:09 ; elapsed = 00:00:06 . Memory (MB): peak = 717.484 ; gain = 66.297
create_bd_design "design_1"
Wrote  : [redacted]:\SoC_project.srcs\sources_1\bd\design_1\design_1.bd>
update_compile_order -fileset sources_1
```

① The **Sources** tab, contains several sub-tabs. The most useful are:

- The **Hierarchy** sub-tab that shows the set of sources that exist in the project: block designs, sources for all of the IP cores, constraints files (XDC)…
- The **IP Sources** sub-tab that shows the files generated when an IP core has been added to the block design.

② When the **Design** tab is selected, a list of all input and output ports, IP core ports, and connections between IP cores is shown. Selecting an entry in this list will highlight that object in the block design diagram.

③ The **Signals** tab allows the user to view lists of all clock and reset signals in the block design. As before, selecting an entry in either of these lists will highlight that signal in the block design diagram.

④ The **Properties** pane shows the properties of the currently selected object and is typically used to quickly view the clock frequency of a selected clock pin, or to change the name of an IP core or port.

⑤ The **Diagram** pane displays a graphical rendition of the current block design.

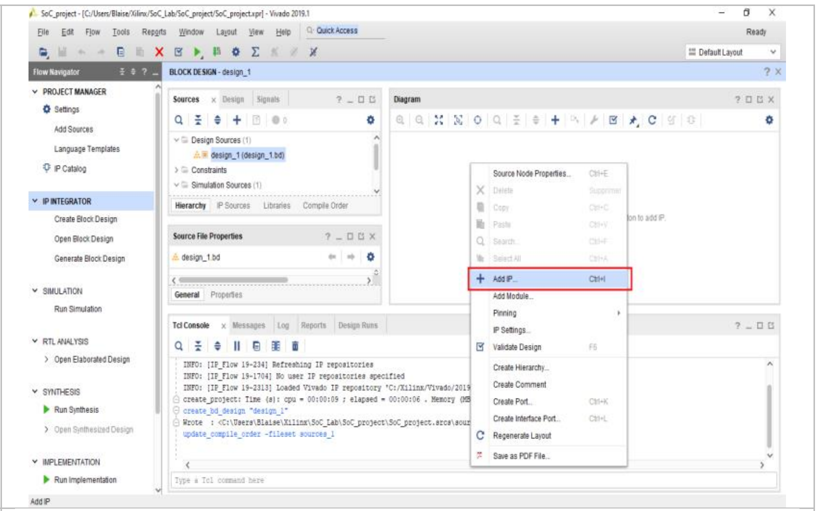
⑥The **TCL Console** displays the scripted commands that Vivado is running whenever a change is made in the graphical interface. Scripts can be created to be run using this tool through the use of the 'source' command.

⑦The **Messages** tab displays Error, Warning, Info, and Status messages created when Vivado takes different actions.

⑧The **Reports** tab contains a list of different reports that Vivado generates as part of the process of generating a bitstream.

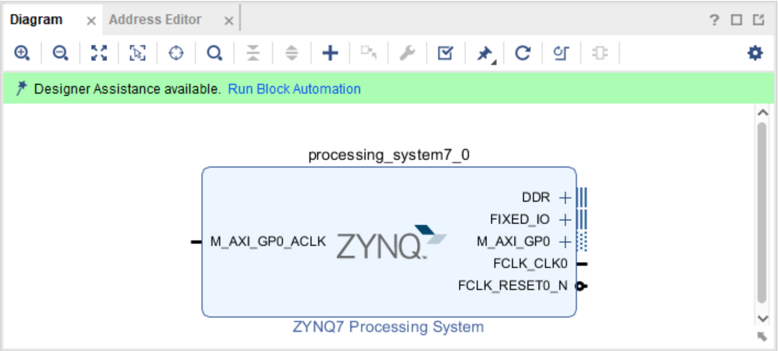
It is now time to build a basic SoC with a **Zynq Microprocessor, a system clock**, an AXI I2S audio codec, a **UART interface**, and a **GPIO interface**.

To add an IP, right click in the Diagram pane, click **Add IP** (or click the '+' symbol in the taskbar), and look for the wanted IP in the dialog box that opens.



```
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository 'C:/Xilinx/Vivado/2019
create_project: Time (s): cpu = 00:00:09 ; elapsed = 00:00:06 . Memory (MB
create_bd_design "design_1"
Wrote  : <C:\Users\Blaise\Xilinx\SoC_Lab\SoC_project\SoC_project.srcs\sour
update_compile_order -fileset sources_1
```

7. Add a **ZYNQ7 Processing System**. **Run Block Automation** with default settings.



8. Add then 2 **AXI GPIO IPs**. Double click the GPIO IPs to set them as follows :
   - On the axi_gpio_0 IP, **map the first GPIO interface to the buttons and the second to the switches.**
   - On the axi_gpio_1 IP, **map the first channel to the leds**.

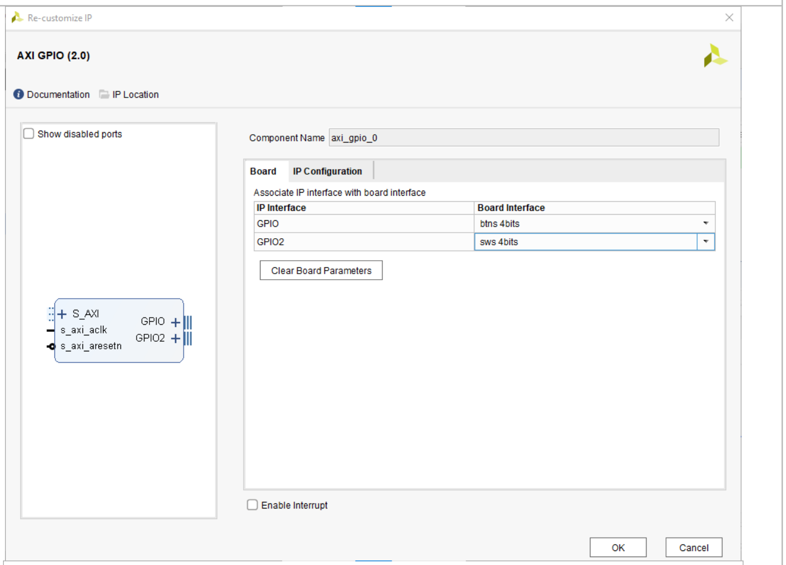
Note: this kind of automatic mapping is possible because the board constraints have previously been defined and loaded (step 4). Otherwise, you can set the GPIO as needed in the "IP configuration" tab and set your inputs/outputs via an xdc constraints file.

Click **Run Connection Automation** in the green bar at the top of the block design diagram pane.

Make sure to check **All automation**. With this box checked, Vivado will automatically connect all the components that interact with the "outside world" with the according port (input or output).

Notice the extra blocks that appears. **What is their purpose?**

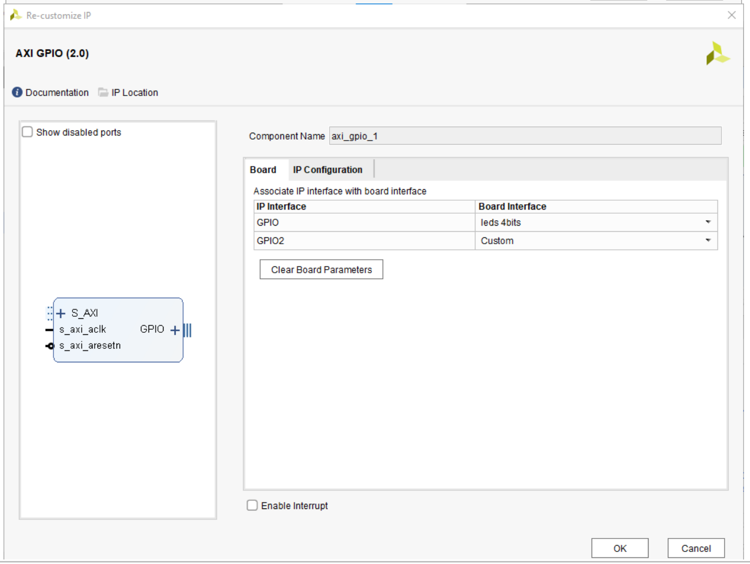
Click on **Regenerate Layout** and **Optimize routing** to optimize the diagram.



Component Name axi_gpio_0

Associate IP interface with board interface

IP Interface	Board Interface
GPIO	btns 4bits
GPIO2	sws 4bits



Component Name axi_gpio_1

Associate IP interface with board interface

IP Interface	Board Interface
GPIO	leds 4bits
GPIO2	Custom

9. With the same method, add the AXI I2S Audio IP, set it with a PL330 DMA type and **Run Connection Automation.** This IP uses the SSM2603 audio codec IC from Analog Devices Inc. Its datasheet can be found on Moodle.

It implements the I2S protocol to record and play audio samples, and the I²C protocol to control its internal registers. Your first assignment is to get familiar with the datasheet and with the 2 protocols. Luckily for you, you will not have to implement any of those, but since you are going to run them, please take time to understand them. Take time to understand the purpose of each internal register as well.

To add, the IP, go to **Settings -> IP -> Repository,** and add the folder where you have previously uncompress the Digilent_libs files available on Moodle.

10. It is now time to set up the Zynq Processing System. Double-click on the IP.

You need to activate:

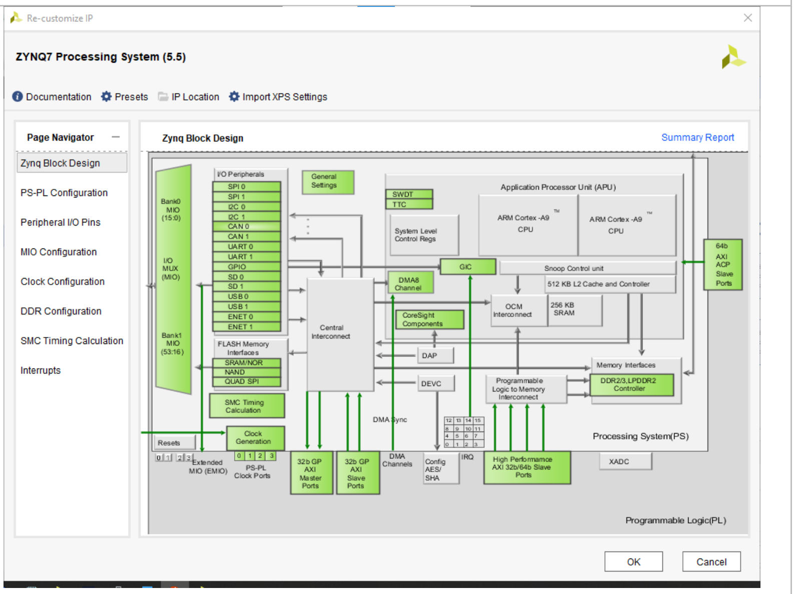
- **The UART 1 interface**
- **The GPIOs interface.**
- **The I2C 0 Interface**
- **2 DMA controllers** (one for the TX of the codec, one for the RX)

You also need to create a secondary PL clock for the SSM2603 master clock. To determine its frequency, see Tab. 30 p. 25 of the datasheet. The specifications are: no secondary Master Clock, 48kHz ADC AND DAC sampling rate, USB=0, SR=0000, BOSR= 0, BLCK=MCLK/4.

You also need to make sure that the DDR controller is set to **MT41K256M16 RE-125.**

Also note the default **UART baud rate.**

Go through the tabs to see all the possible options.

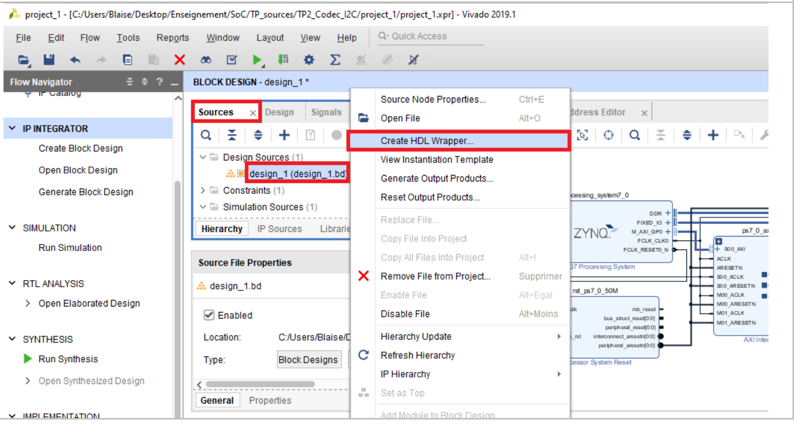


11. The DMA ports, the secondary clocks and the AXI I2S Audio IP ports may not all be connected. Make the needed connections and create the IP ports (IIC_0, SDATA_I, SDATA_O, MCLK, BCLK, MUTEN_O, LRCLK_O)

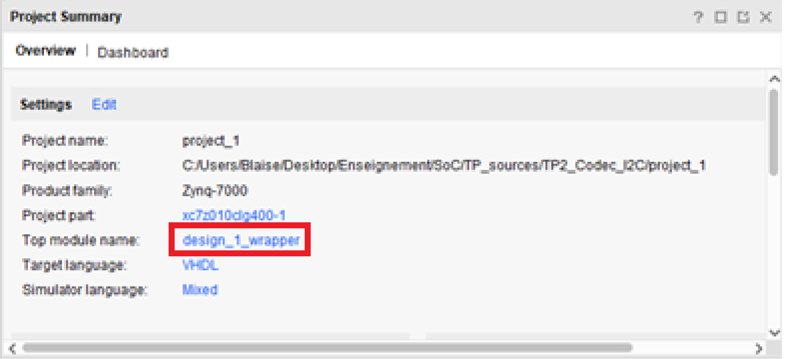
It is now time to **generate the bitstream** of the SoC.

12. First, right click in the diagram pane and check you have no design error by clicking **Validate Design**.

13. The last thing that needs to be done before generating a bitstream is to create a top module file. This file will take the block design and interpret it into a hardware design language so that the synthesis and implementation tools can work properly. Right click on the block design in the **Sources** tab and select **Create HDL Wrapper**. In the confirmation dialog that pops up, make sure that *Let Vivado manage wrapper and auto-update* is selected in the options list.


project_1 - [C:/Users/Blaise/Desktop/Enseignement/SoC/TP_sources/TP2_Codec_I2C/project_1/project_1.xpr] - Vivado 2019.1

You can check in the Project summary that the design wrapper has automatically been set as top module by Vivado.



Project Summary

Overview | Dashboard

Settings Edit

Project name:	project_1
Project location:	C:/Users/Blaise/Desktop/Enseignement/SoC/TP_sources/TP2_Codec_I2C/project_1
Product family:	Zynq-7000
Project part:	xc7z010clg400-1
Top module name:	design_1_wrapper
Target language:	VHDL
Simulator language:	Mixed

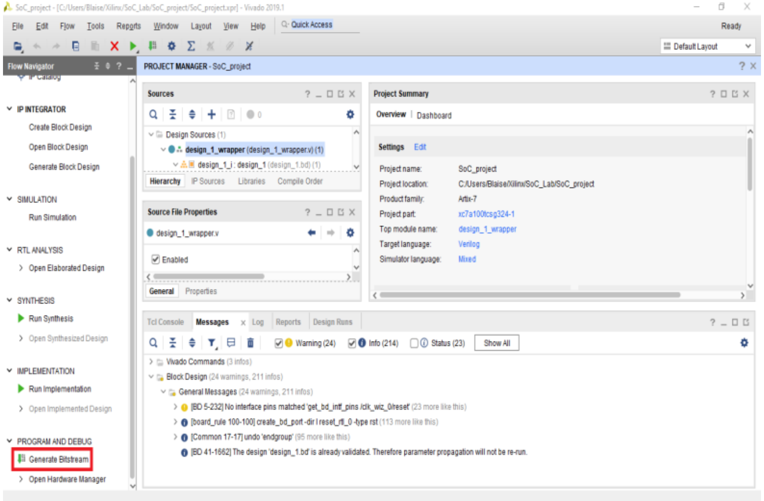
14. Map the AXI I2S Audio IP ports in in a **constraints file** (a template can be found on Github). Be careful, the wrapper tool may slightly change the name of the interfaces: check that the port names in your constraints file match the names in the generated wrapper.

It is not needed to map the buttons, switches and leds in the constraints file since they have already been set.

15. With a validated design and a top module, a bitstream can now be generated: click the **Generate Bitstream** button in the **Flow Navigator**. Leave the options as they are.

You can ignore MOST of the warnings.

Once the bitstream is generated, which may take a little while, Vivado will ask what to do next. Click **Cancel**.



Design Sources (1)

design_1_wrapper (design_1_wrapper.v) (1)

design_1_i : design_1 (design_1.bd) (1)

design_1_wrapper.v

☑ Enabled

Project Summary

Overview | Dashboard

Settings Edit

Project name:	SoC_project
Project location:	C:/Users/Blaise/Xilinx/SoC_Lab/SoC_project
Product family:	Artix-7
Project part:	xc7a100tcsg324-1
Top module name:	design_1_wrapper
Target language:	Verilog
Simulator language:	Mixed

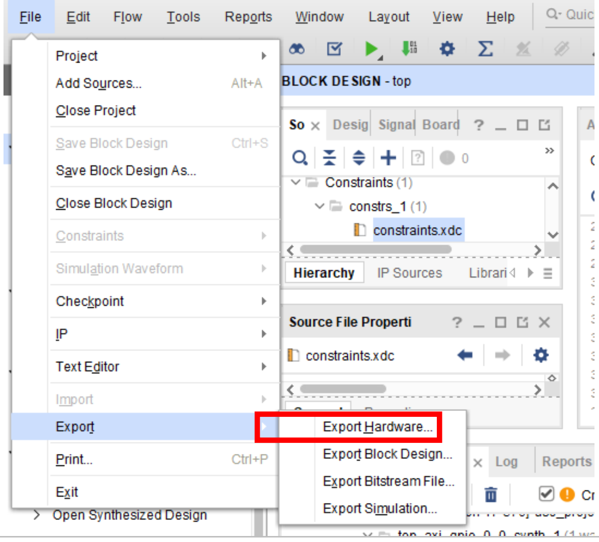
☑ Warning (24) ☑ Info (214) □ Status (23)

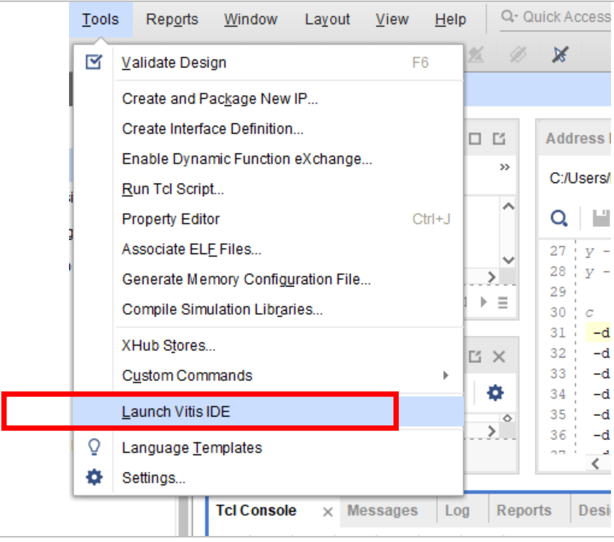
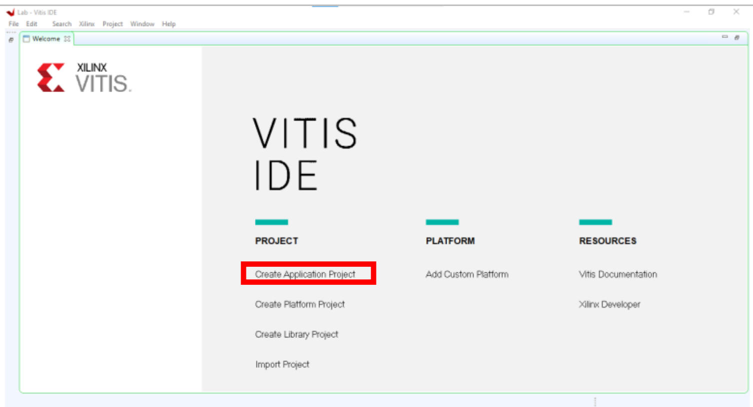
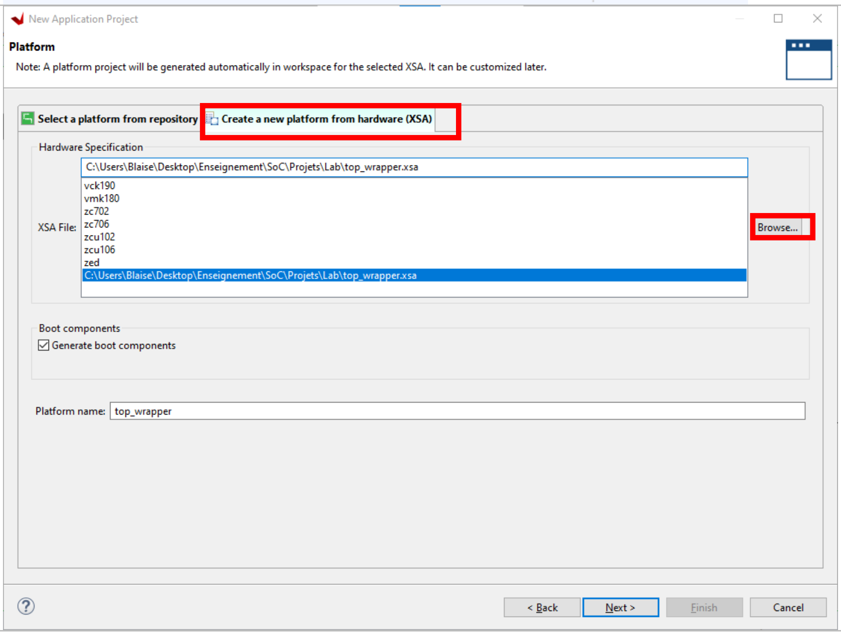
```
Vivado Commands (3 infos)
Block Design (24 warnings, 211 infos)
  General Messages (24 warnings, 211 infos)
    [BD 5-232] No interface pins matched 'get_bd_intf_pins /clk_wiz_0/reset' (23 more like this)
    [board_rule 100-100] create_bd_port -dir I reset_rtl_0 -type rst (113 more like this)
    [Common 17-17] undo 'endgroup' (95 more like this)
    [BD 41-1662] The design 'design_1.bd' is already validated. Therefore parameter propagation will not be re-run.
```

SoC Programming with VITIS

Before Vitis can be launched, it needs to be provided with a hardware handoff file, containing the bitstream and various other information about the peripherals installed in the design.

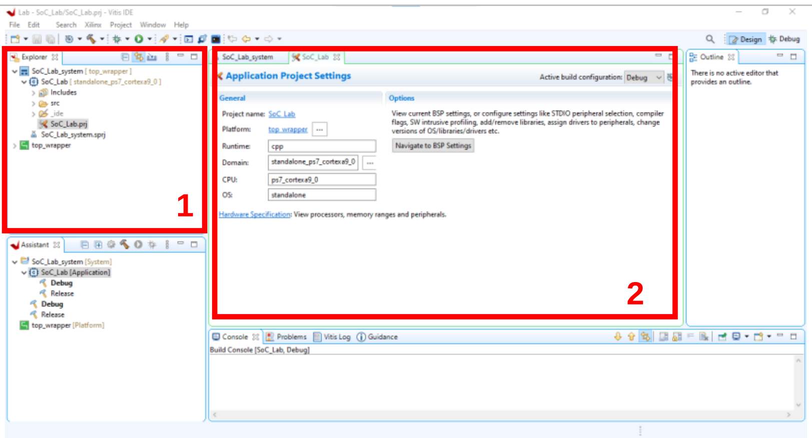
16. In the **File** drop-down, select **Export** then **Export Hardware (1).** Click **Next** then select **Include Bitstream**, so that the FPGA can be programmed from Vitis. Leave the others options as they are.



17. Click Tools, then Lauch IDE Vitis In the dialog that pops up, set the workspace with your project repository. It will avoid unnecessary file spreading.	 Tools Reports Window Layout View Help Quick Access Validate Design F6 Create and Package New IP... Create Interface Definition... Enable Dynamic Function eXchange... Run Tcl Script... Property Editor Ctrl+J Associate ELF Files... Generate Memory Configuration File... Compile Simulation Libraries... XHub Stores... Custom Commands Launch Vitis IDE Language Templates Settings... Tcl Console Messages Log Reports Desi
Note: If the automatic launch of Vitis does not work, you still can open it manually. Afterwards, you will need to set your workspace so that Vitis can find the bitstream you have previously exported.	
18. In the Vitis welcome window, choose **Create Application Project**. The next screen explains the methodology to run an application on a platform. Click **Next**.	 XILINX VITIS VITIS IDE PROJECT Create Application Project Create Platform Project Create Library Project Import Project PLATFORM Add Custom Platform RESOURCES Vitis Documentation Xilinx Developer
19. Choose the **Create a new platform from Hardware (XSA),** browse to your project folder and choose the XSA file you have exported earlier. In the next window, choose a name for your project. Leave the options in the following window as they are. Eventually, create an **Empty Application**.	 New Application Project Platform Note: A platform project will be generated automatically in workspace for the selected XSA. It can be customized later. Select a platform from repository / Create a new platform from hardware (XSA) Hardware Specification XSA File: C:\Users\Blaise\Desktop\Enseignement\SoC\Projets\Lab\top_wrapper.xsa vck190 vmk180 zc702 zc706 zcu102 zcu106 zed C:\Users\Blaise\Desktop\Enseignement\SoC\Projets\Lab\top_wrapper.xsa Browse... Boot components ☑ Generate boot components Platform name: top_wrapper < Back Next > Finish Cancel

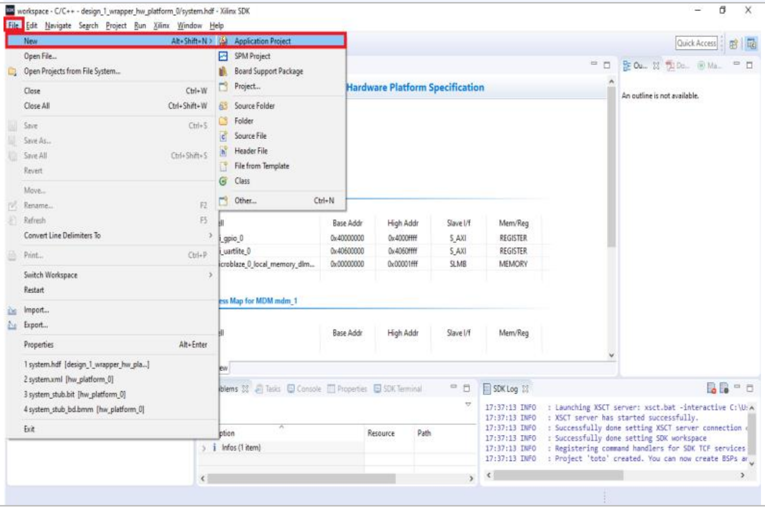
①The **Project Explorer**, found to the left hand side of the window, displays all projects imported into the workspace, each of which can be expanded to view the sources, scripts, and other files that they contain.

②The **File View**, directly in the center of the window, displays currently opened files and information about the project/platform.


Lab - SoC_Lab/SoC_Lab.prj - Vitis IDE
File Edit Search Xilinx Project Window Help
Explorer
SoC_Lab_system [top_wrapper]
SoC_Lab [standalone_ps7_cortexa9_0]
Includes
src
_ide
SoC_Lab.prj
SoC_Lab_system.sprj
top_wrapper
1
Assistant
SoC_Lab_system [System]
SoC_Lab [Application]
Debug
Release
Debug
Release
top_wrapper [Platform]
SoC_Lab_system
SoC_Lab
Application Project Settings
Active build configuration: Debug
General
Project name: SoC_Lab
Platform: top_wrapper
Runtime: cpp
Domain: standalone_ps7_cortexa9_0
CPU: ps7_cortexa9_0
OS: standalone
Options
View current BSP settings, or configure settings like STDIO peripheral selection, compiler flags, SW intrusive profiling, add/remove libraries, assign drivers to peripherals, change versions of OS/libraries/drivers etc.
Navigate to BSP Settings
Hardware Specification: View processors, memory ranges and peripherals.
2
Outline
There is no active editor that provides an outline.
Design
Debug
Console
Problems
Vitis Log
Guidance
Build Console [SoC_Lab, Debug]

Choosing this type of application pre-configures the platform (platform.c, platform.h, platform_config.h).

Explore the **xparameters.h** file: it links the registers from the "software world" to the physical ressources of the "hardware world".

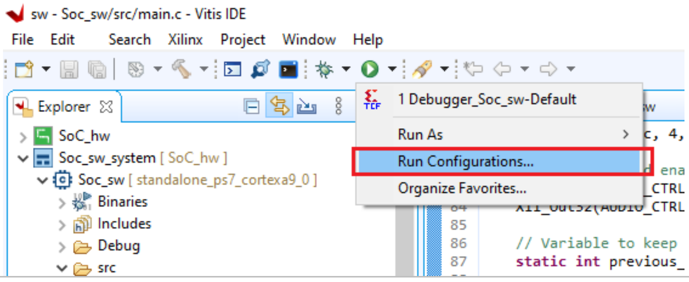


```
17:37:13 INFO  : Launching XSCT server: xsct.bat -interactive C:\U
17:37:13 INFO  : XSCT server has started successfully.
17:37:13 INFO  : Successfully done setting XSCT server connection
17:37:13 INFO  : Successfully done setting SDK workspace
17:37:13 INFO  : Registering command handlers for SDK TCF services
17:37:13 INFO  : Project 'toto' created. You can now create BSPs a
```

20. Import the c-files available on Moodle, look for the function protoypes in the header files and fill the main.c to :
    a. make the codec record and play sound and music. The codec can be muted through sw0.
    b. If the switch 1, 2, 3 or 4 is toggled, the LED 1, 2, 3 or 4 accordingly toggles.
    c. If button 1, 2, 3 or 4 is pressed, a message is sent through UART, detailing which button is activated.

The next step is to program the bit file onto the FPGA and run the C source on the processor.

21. Click on the "Play" button and select **Run configurations…**

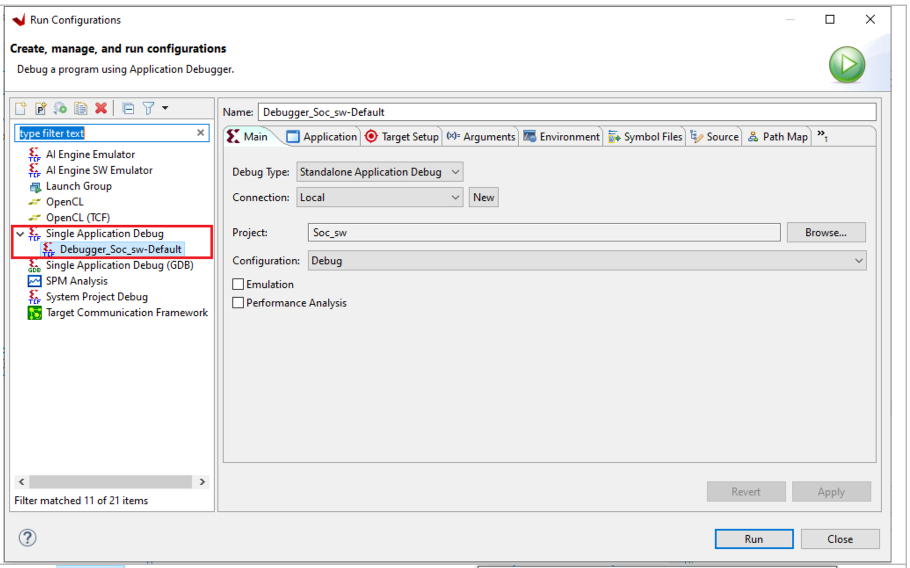

sw - Soc_sw/src/main.c - Vitis IDE
File Edit Search Xilinx Project Window Help
Explorer
SoC_hw
Soc_sw_system [SoC_hw]
Soc_sw [standalone_ps7_cortexa9_0]
Binaries
Includes
Debug
src
1 Debugger_Soc_sw-Default
Run As
Run Configurations...
Organize Favorites...
85
86
87

```
// Variable to keep
static int previous_
```

22. In the pop-up window, create a new **Single Application Debug** run and leave the options as they are.

Click on **Run** to run your application.

From now on, you don't need to go into **Run configuration** to run your code. Just click on the Play button.



Run Configurations

Create, manage, and run configurations

Debug a program using Application Debugger.

Name: Debugger_Soc_sw-Default

Debug Type: Standalone Application Debug

Connection: Local

Project: Soc_sw

Configuration: Debug

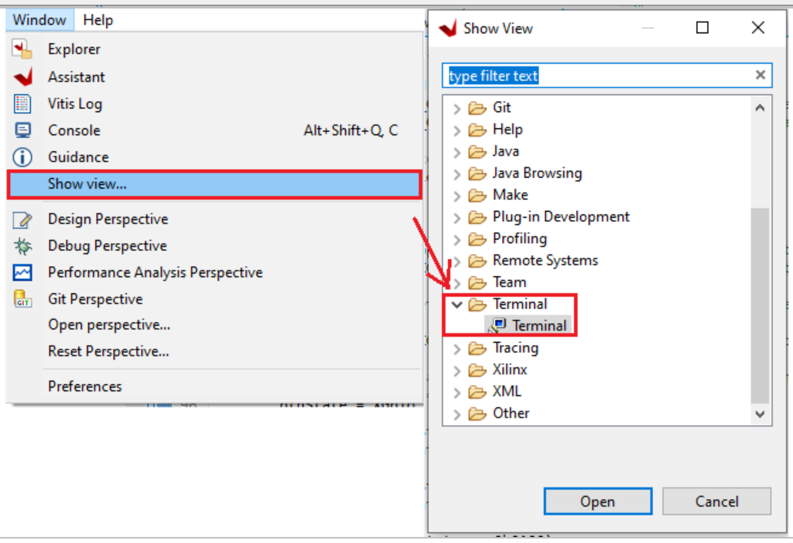
☐ Emulation

☐ Performance Analysis

Filter matched 11 of 21 items

To receive (or send) Messages over UART : UART messages are sent by the 'xil_printf' statements in the project's C source code. To communicate over UART, it is recommended to use a serial console application like **Putty** or the internal Vitis terminal, that can be found in **Window->Show View -> Terminal -> Terminal** . The appropriate port to connect to can be determined by reviewing the Device Manager in Windows. Serial Port settings are determined by the configuration of the Zynq.

Typically, these settings will be **8 Data Bits**, **No Parity Bit**, **1 Stop Bit with a Bad rate determined before**.



The project will now be running on the board. Toggle the switches to see the LEDs flip on and off. Use the terminal to view the pressed button(s).

WHAT'S NEXT…

Now that you have a basic experience with Vivado, Vitis and the interaction between the two environments, the project really begins!

You are asked to develop basic audio effects that can be controlled with the btn/sw of the board, or with external peripherals that are freely available. Be creative !

A guide to develop custom IP linking VHDL code to AXI (and therefore to Vitis) is available on Moodle if needed

You will present your work on January 12th. You are also asked to provide a report similar to a user manual (no code please, expect if extremely fancy!) before January 31st.

Have fun !