

COMP3601 Baseline Project Quickstart

Overview

Every group will receive a baseline FPGA project package consisting of:

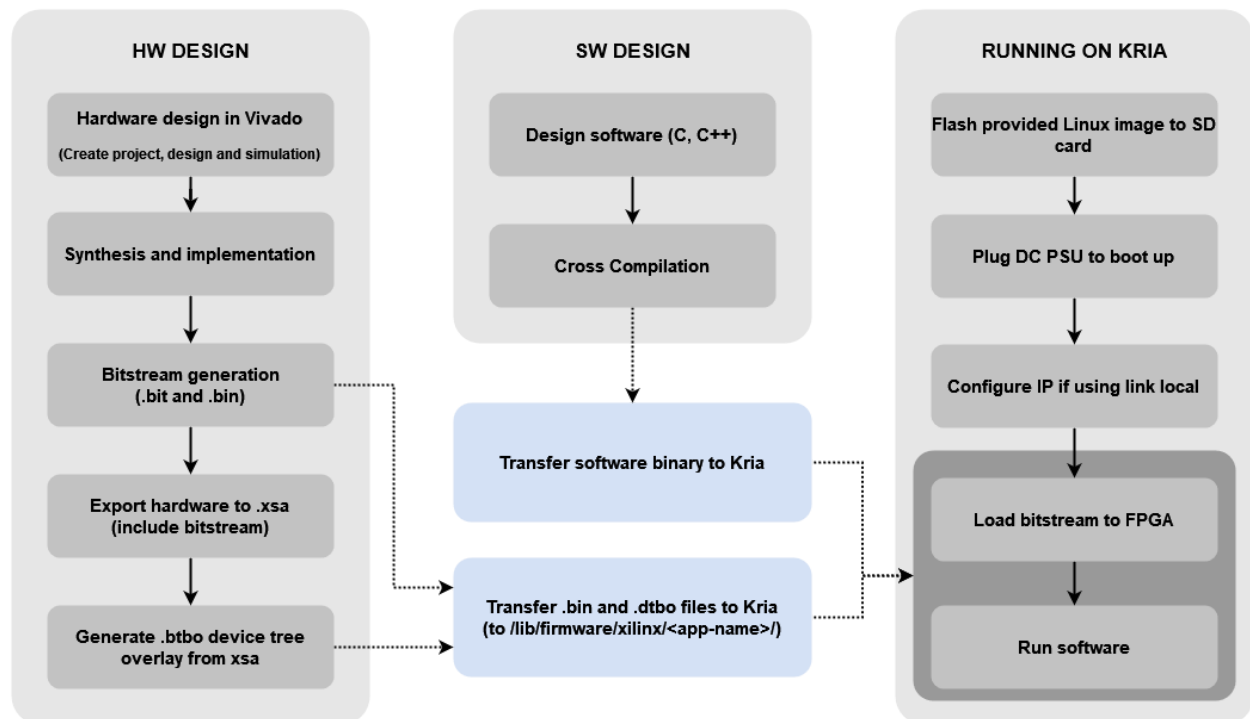
- Pre-flashed Petalinux image with configured software and example bitstream
- Some HDL and C/C++ source code for the example bitstream and software
- Cross-compilation SDK for the Kria board
- (upon request) Dante Virtual Soundcard licence

This guide describes the steps for students to replicate the preloaded example bitstream and software. Students are required to go through this tutorial for familiarisation and to set up their project repo. The diagram below is an overview of the steps of development for the Kria platform. In this guide, we will be using Windows as our OS running on the host machine while using WSL to perform device tree compilation and software compilation. However, the steps are very similar if you are using Ubuntu Linux.

For those wishing to install WSL, here are some guides:

- Official Microsoft Docs on install WSL: [link](#)
- Installing WSL on a non-system drive: [link](#)

I recommend using WSL1 instead of WSL2 for this project. You can temporarily swap between versions by running `wsl --set-version <distro-name> 1` in powershell with admin rights.



Prerequisites

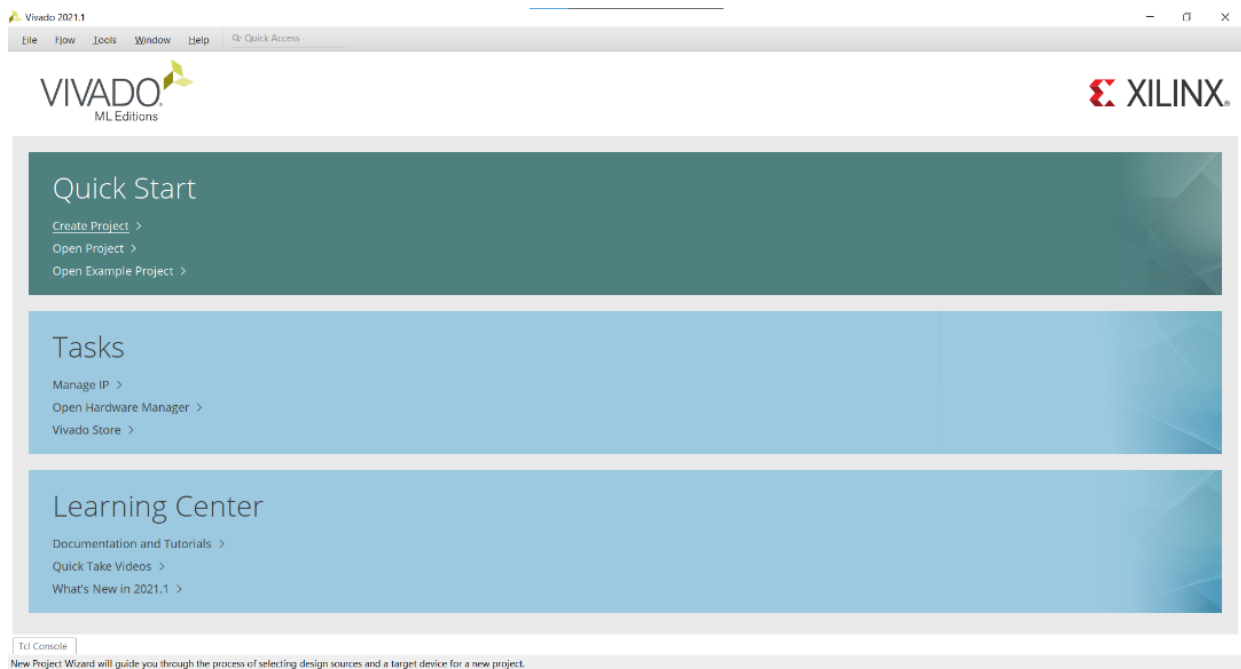
The software you should have installed before starting:

- Vitis Vivado 2021.1 or later (recommended: 2021.1)
 - Xilinx Software Command Line Tool (should come with your Vivado installation)
 - PuTTY or other serial consoles (Ubuntu: use minicom)
 - DTC tool (we'll install this further down, skip for now)
-

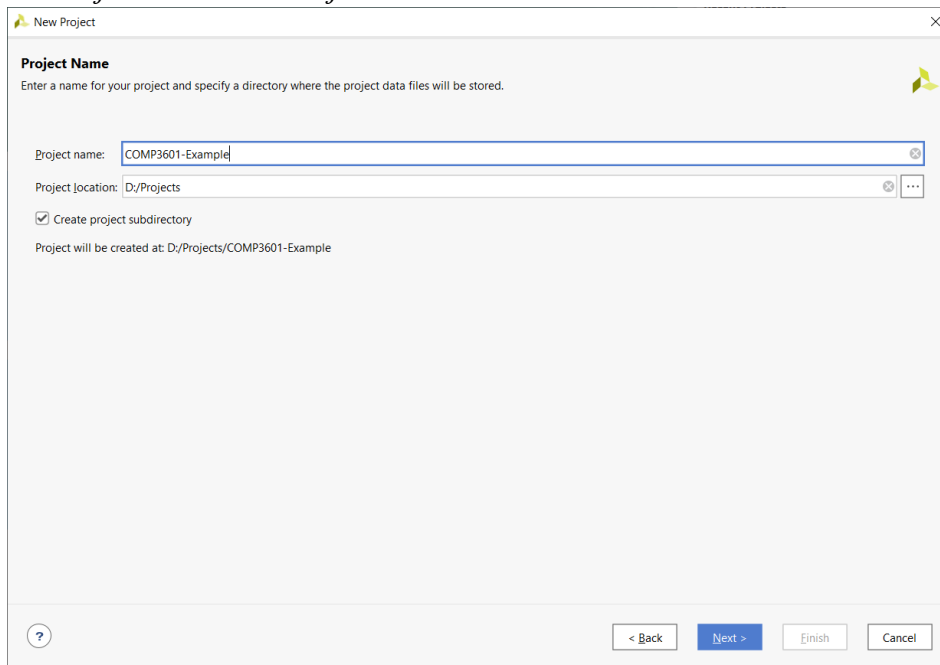
Hardware Project Setup

Creating a new project in Vivado

1. Open Vivado. Click on *Create Project* and click *Next*.

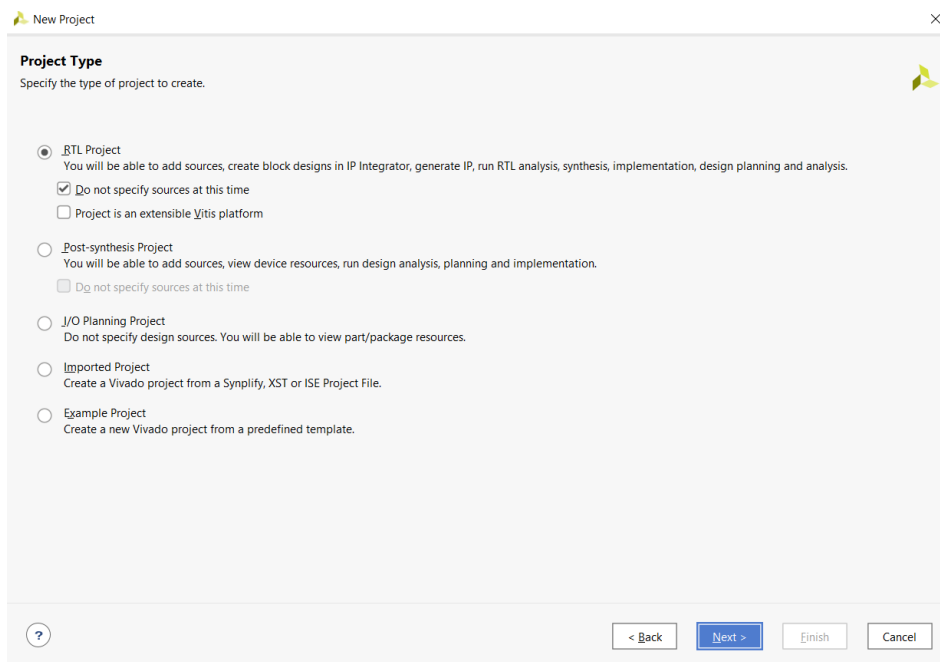


2. Set *Project name* and *Project location* and click *Next*.



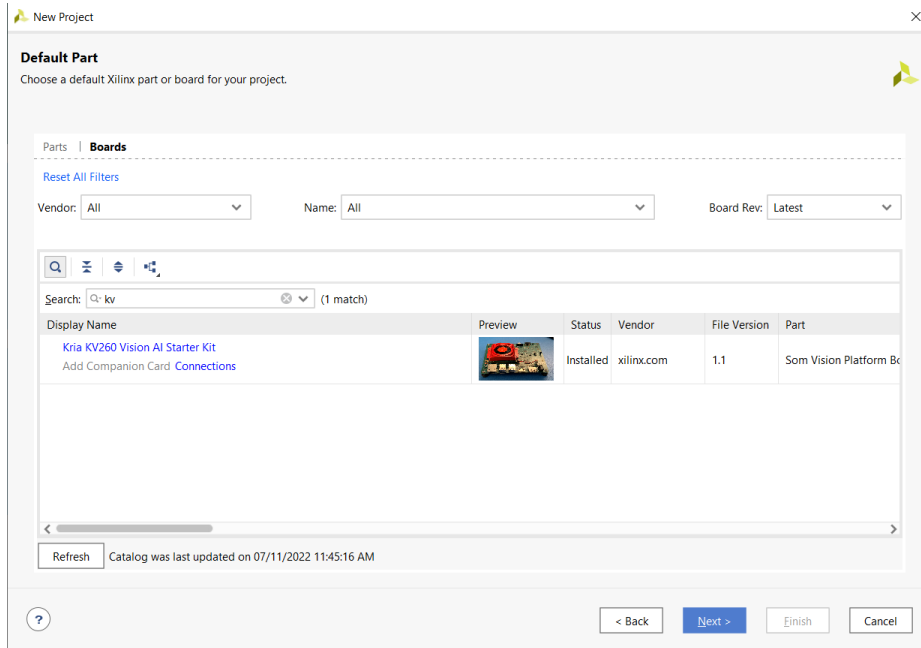
The screenshot shows the 'New Project' dialog box with the title bar 'New Project'. The main heading is 'Project Name'. Below it, a subtitle reads: 'Enter a name for your project and specify a directory where the project data files will be stored.' There are two input fields: 'Project name:' with the text 'COMP3601-Example' and 'Project location:' with the text 'D:/Projects'. Below these fields, there is a checked checkbox labeled 'Create project subdirectory' and a line of text stating 'Project will be created at: D:/Projects/COMP3601-Example'. At the bottom of the dialog, there are four buttons: '< Back', 'Next >' (highlighted in blue), 'Finish', and 'Cancel'. A help icon (?) is located in the bottom left corner.

3. Select *RTL Project*. We will add the sources later so tick *Do not specify sources at this time*. Click *Next*.



The screenshot shows the 'New Project' dialog box with the title bar 'New Project'. The main heading is 'Project Type'. Below it, a subtitle reads: 'Specify the type of project to create.' There are five radio button options: 1. 'RTL Project' (selected): 'You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.' It has a checked checkbox 'Do not specify sources at this time' and an unchecked checkbox 'Project is an extensible Vitis platform'. 2. 'Post-synthesis Project': 'You will be able to add sources, view device resources, run design analysis, planning and implementation.' It has an unchecked checkbox 'Do not specify sources at this time'. 3. 'J/O Planning Project': 'Do not specify design sources. You will be able to view part/package resources.' 4. 'Imported Project': 'Create a Vivado project from a Synplify, XST or ISE Project File.' 5. 'Example Project': 'Create a new Vivado project from a predefined template.' At the bottom of the dialog, there are four buttons: '< Back', 'Next >' (highlighted in blue), 'Finish', and 'Cancel'. A help icon (?) is located in the bottom left corner.

- In the *Default Part* section, click on the **Boards** tab and type "kv" in the search bar. Select **Kria KV260 Vision AI Starter Kit** and click *Next*.

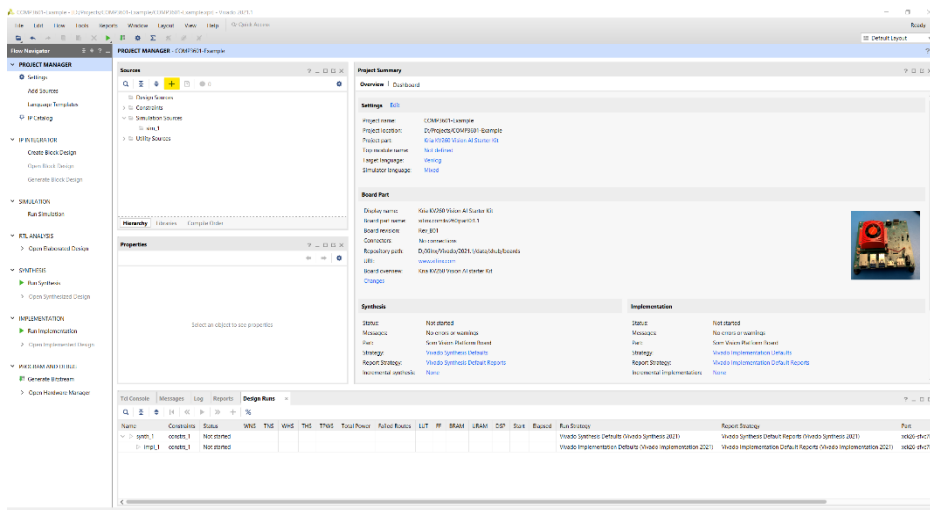


- Click *Finish* to create your project.

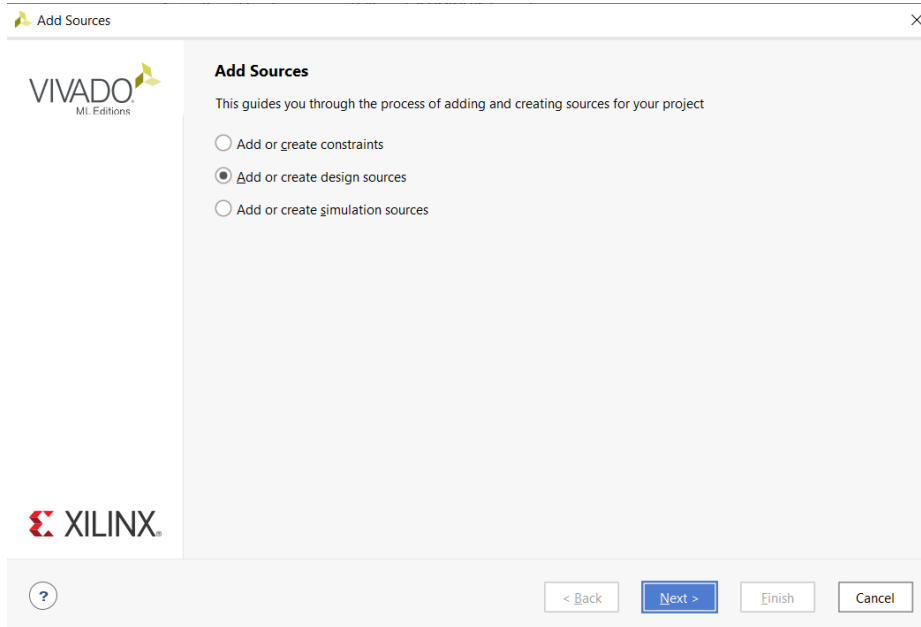
Make sure the *Project part* shows **Kria KV260 Vision AI Starter Kit**.

Adding constraints to the project

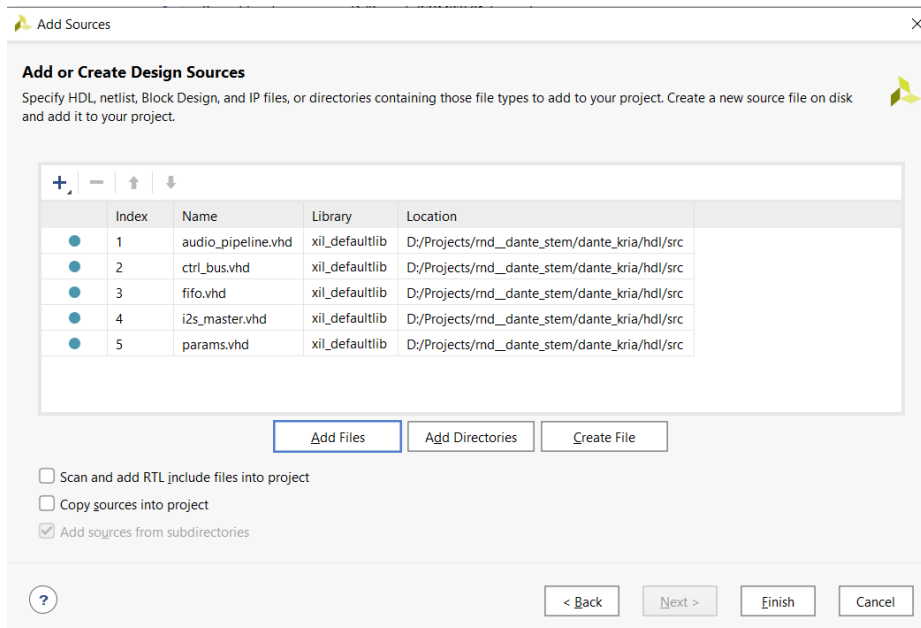
- In the *Project Manager*, click on + in the *Sources* window.



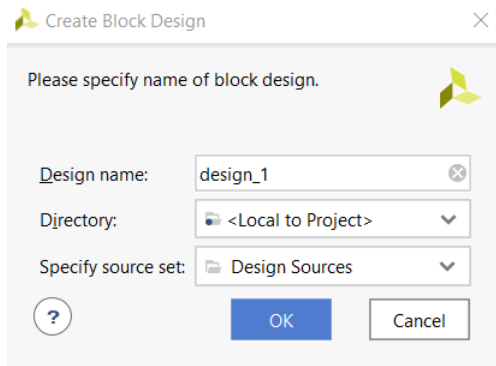
2. Select *Add or create design sources* and click *Next*.



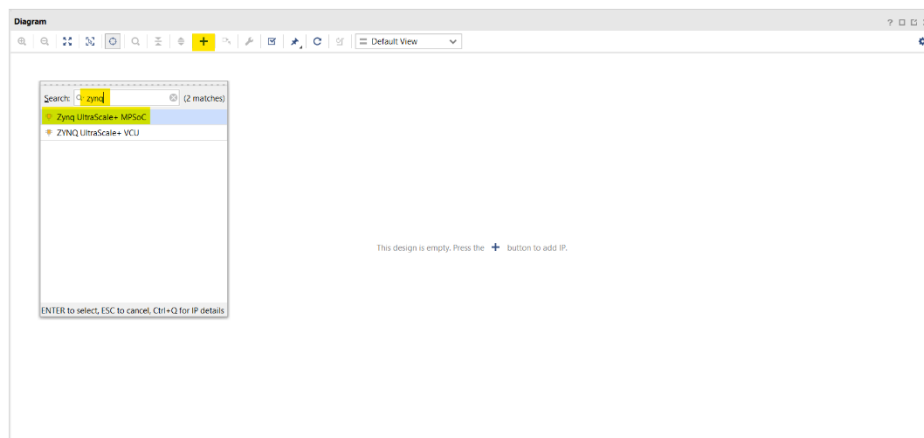
3. Add the HDL design files from Teams. I recommend **not** ticking *Copy sources into project* for version control purposes. Click *Finish* when done.
 1. Note: We provide an i2s_master.vhd file, but it is only a skeleton. You'll need to finish it off!



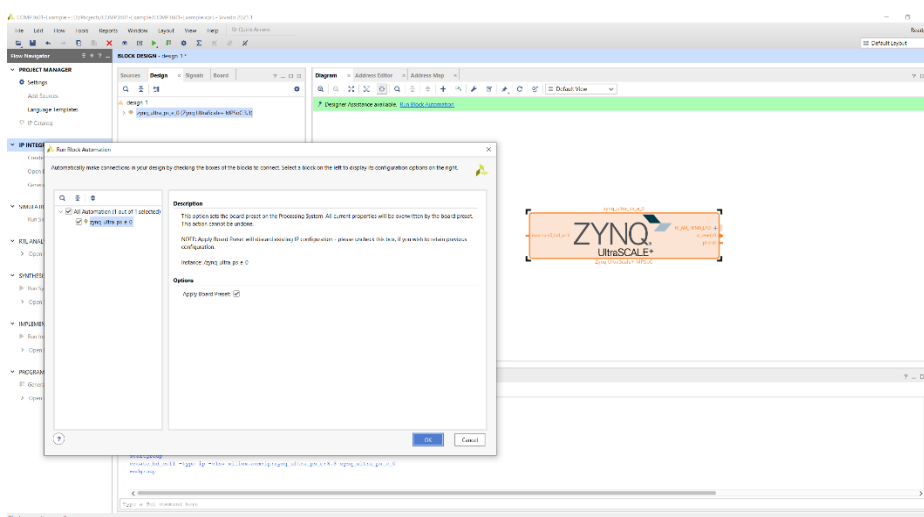
4. Add the constraint file for the Kria board into the project by choosing *Add or create constraints* in *Add sources*. Click *Finish* when done.



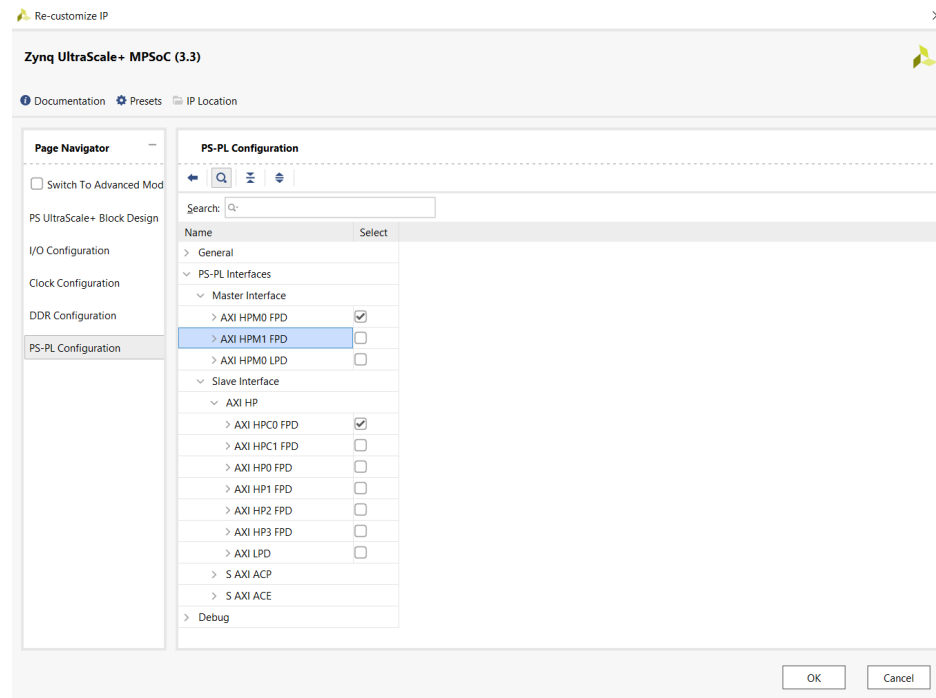
3. An empty *Diagram* should show up. Click on + and search for “zynq”. Double click on *Zynq UltraScale+ MPSoC* to add the IP into your block design.



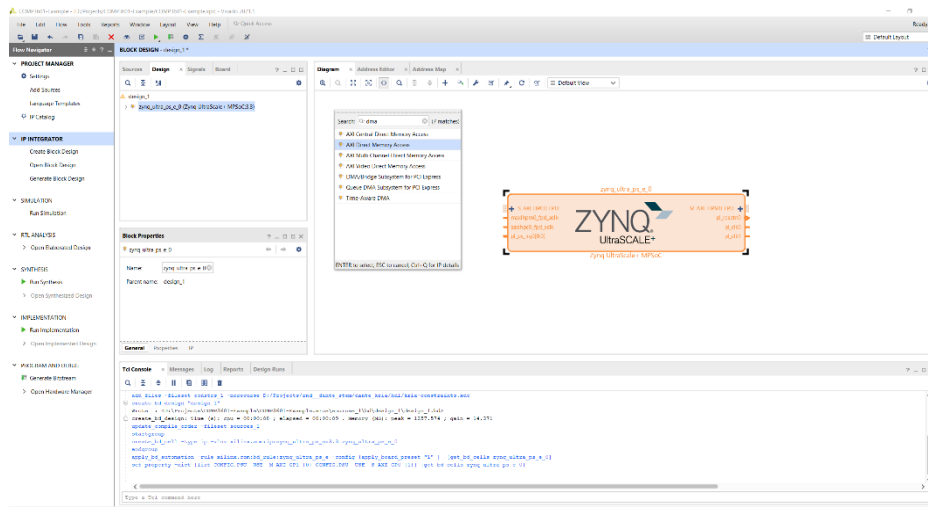
4. Click on *Run Block Automation* and click *OK* which will apply the Kria AI Starter Kit preset to the Zynq device.



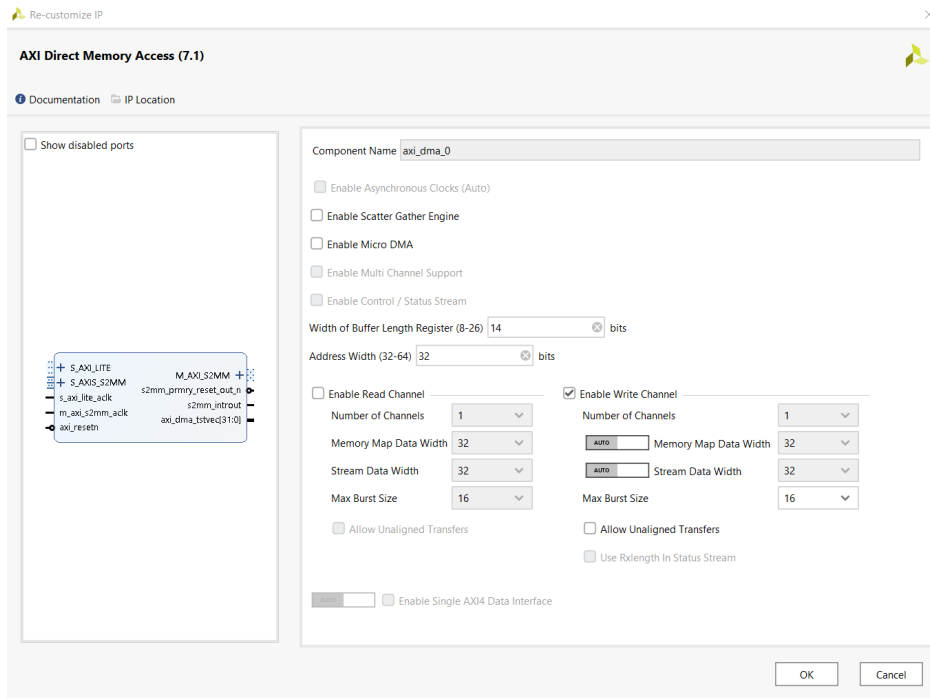
5. Double click on the Zynq MPSoC IP and navigate to *PS-PL Configuration* → *PS-PL Interfaces* and do the following configurations:
 1. *Master Interface* → Untick *AXI HPM1 FPD*
 2. *Slave Interface* → Enable *AXI HPC0 FPD*



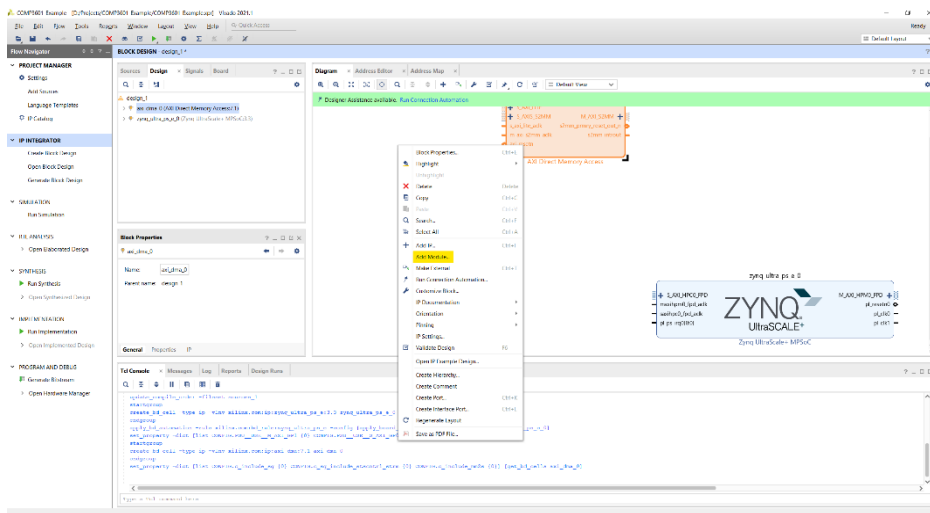
6. Click on +, search for “dma” and double click on *AXI Direct Memory Access* to add the IP into your block design.



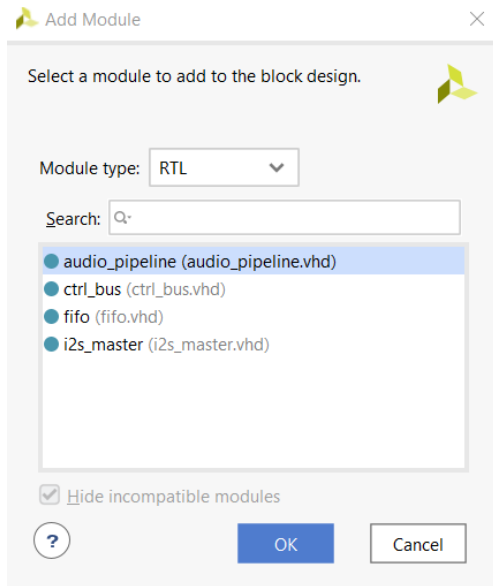
7. Double click on the DMA device and untick *Enable Scatter Gather Engine* and *Enable Read Channel*. Click *OK*.



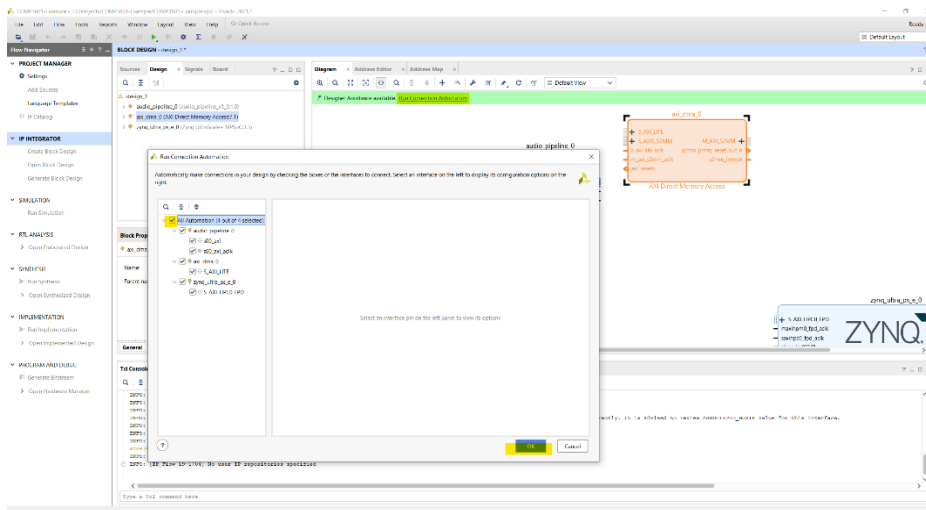
8. Right-click on the diagram and select *Add Module*.



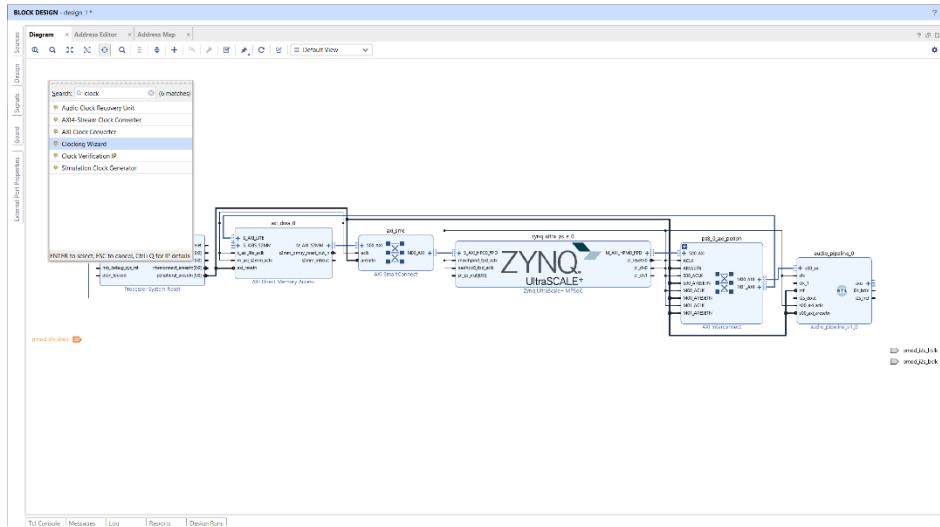
9. Select *audio_pipeline* and click *OK*.



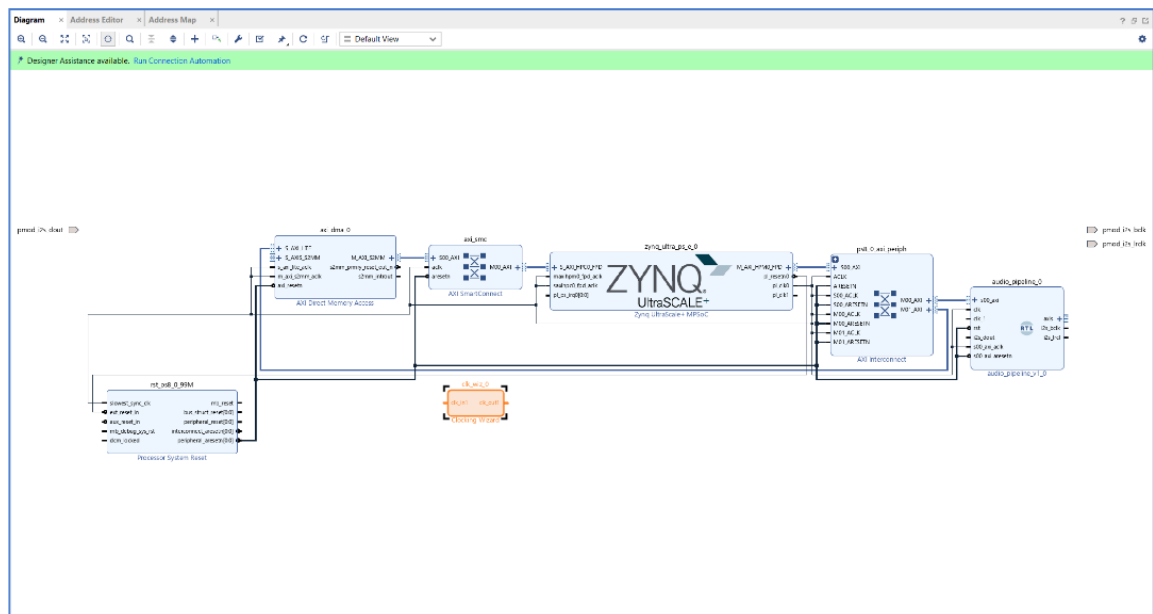
10. Click on *Run Connection Automation* and tick on all boxes to let the IP Integrator connect and add the required interconnect IPs between the HDL module and the Zynq device.



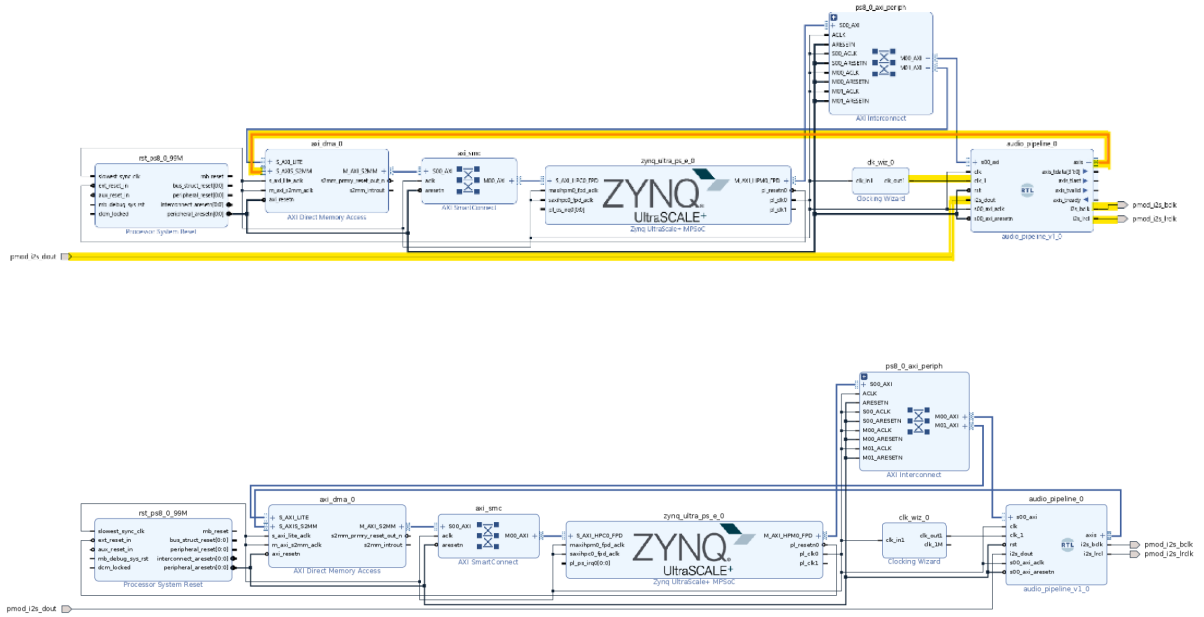
11. Right-click on the diagram and select *Create Port*. Add the following ports:
 1. pmod_i2s_lrcclk; *Output; Other*
 2. pmod_i2s_bclk; *Output; Other*
 3. pmod_i2s_dout; *Input; Other*
12. Click on + to add another IP and search for “clocking wizard”. Double click to add the IP to your block design.



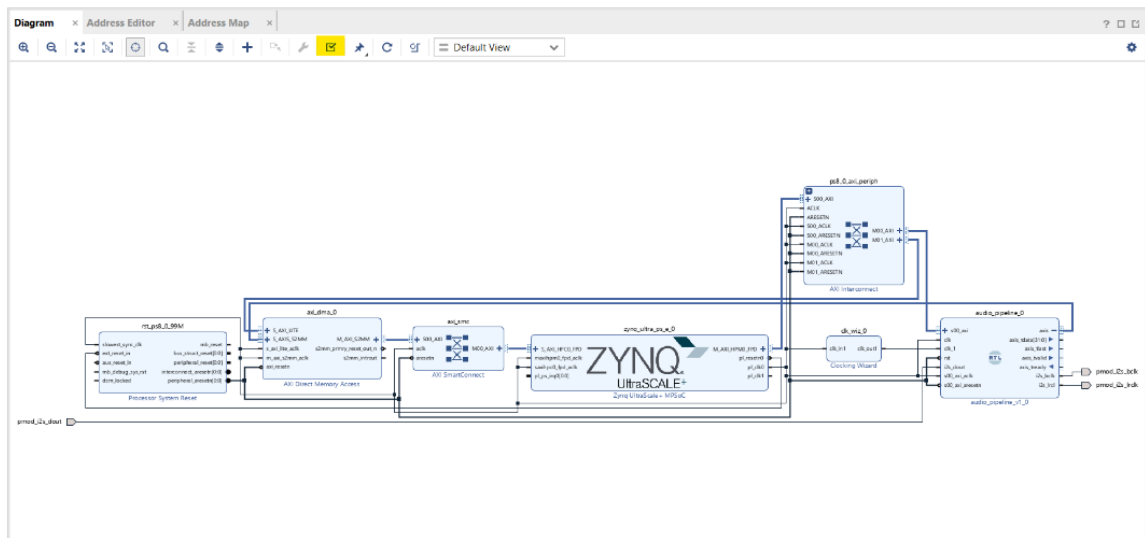
13. Double click on the clocking wizard IP and navigate to the *Output Clocks* tab.
 1. Set the *clk_out1* clock to a requested **49.152MHz** frequency.
 2. Scroll down and under *Enable Optional Inputs / Outputs for MMCM/PLL* untick the *reset* and *locked* ports.
 3. Click *OK* to finish the configuration.
14. At this point your diagram should look like this:



15. Connect the ports on the audio_pipeline_0 IP to the corresponding pins which should look like the following diagram:



16. Validate the block design by clicking on the ticked-box icon.



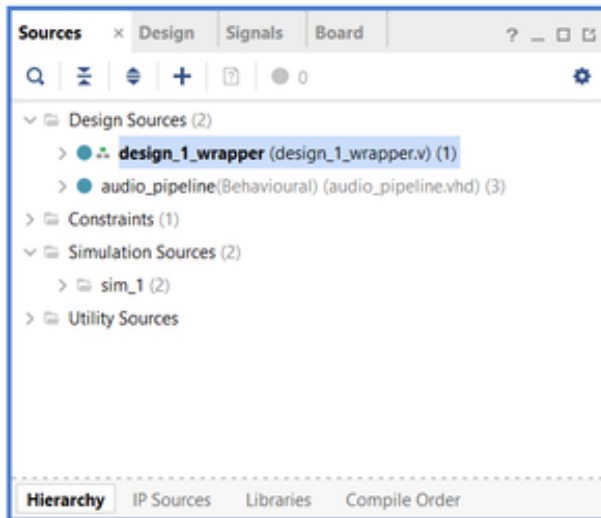
17. Under the *Sources* tab, right-click on the design_1.bd block design and click *Create HDL Wrapper*. Select *Let Vivado manage wrapper and auto-update* and click *OK*.

18. Go to the *Address Editor* tab and make sure that under Network 1, the /audio_pipeline_0/s00_axi has address 0x00_A001_0000 and

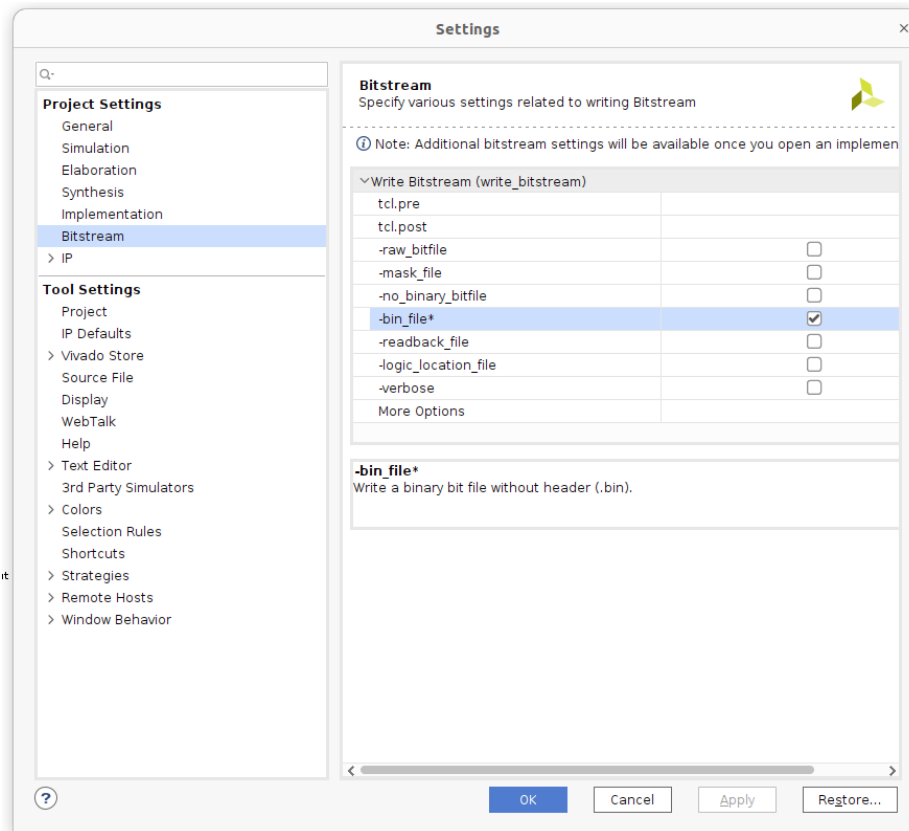
/axi_dma_0/S_AXI_LITE has the address 0x00A000_0000

Diagram	design_1_wrapper.vhd	kria-constraints.xdc	Address Editor	Address Map	audio_pipeline.vhd
<input checked="" type="checkbox"/> Assigned (4) <input checked="" type="checkbox"/> Unassigned (0) <input checked="" type="checkbox"/> Excluded (2) <input type="button" value="Hide All"/>					
Name	Interface	Slave Segment	Master Base Address	Range	Master High Address
Network 0					
/axi_dma_0					
/axi_dma_0/Data_S2MM (32 address bits : 4G)					
/zynq_ultra_ps_e_0/SAXIGP0	S_AXI_HPC0_FPD	HPC0_QSPI	0xC000_0000	512M	0xDFFF_FFFF
/zynq_ultra_ps_e_0/SAXIGP0	S_AXI_HPC0_FPD	HPC0_DDR_LOW	0x0000_0000	2G	0x7FFF_FFFF
Excluded (2)					
/zynq_ultra_ps_e_0/SAXIGP0	S_AXI_HPC0_FPD	HPC0_DDR_HIGH			
/zynq_ultra_ps_e_0/SAXIGP0	S_AXI_HPC0_FPD	HPC0_LPS_OCM	0xFF00_0000	16M	0xFFFF_FFFF
Network 1					
/zynq_ultra_ps_e_0					
/zynq_ultra_ps_e_0/Data (39 address bits : 0x00A0000000 [256M] , 0x0400000000 [4G] , 0x1000000000 [224G])					
/audio_pipeline_0/s00_axi	s00_axi	reg0	0x00_A001_0000	64K	0x00_A001_FFFF
/axi_dma_0/S_AXI_LITE	S_AXI_LITE	Reg	0x00_A000_0000	64K	0x00_A000_FFFF

19. Right-click on the *design_1_wrapper* and click on *Set as Top*. Now the *design_1_wrapper* should be bold and on the top in *Design Sources*.



20. On the *Flow Navigator* tab, click on *Settings* under the *PROJECT MANAGER*. If you want the HDL wrapper you created to be in VHDL set the *Target Language* to VHDL. In *Bitstream*, enable `-bin_file` so that generating bitstream will give you a .bin version of the bitstream which will be needed for the Kria board.



20. Note that you can try include your LED blinking code module as well, and add another clock output from the clocking wizard...

Building the project

1. Remember it won't actually work until you've finished the `i2s_master.vhd` file.
2. In the *Flow Navigator* window, click on *Run Synthesis* and launch the synthesis tasks.
3. Once the synthesis has been completed, click on *Run Implementation* to run implementation tasks.
4. Once the implementation has been completed, click on *Generate Bitstream* to generate the .bit and .bin bitstream files. (The files will be located at `<vivado-project-location>/design_1.runs/impl1/` under the name `design_1_wrapper.bit` and `design_1_wrapper.bin`)

5. Click on *File* → *Export* → *Export Hardware*. Click *Next* and select *Include bitstream*. Set the xsa filename and choose your location to export the file to. Click *Next* and click *Finish* to complete your export.

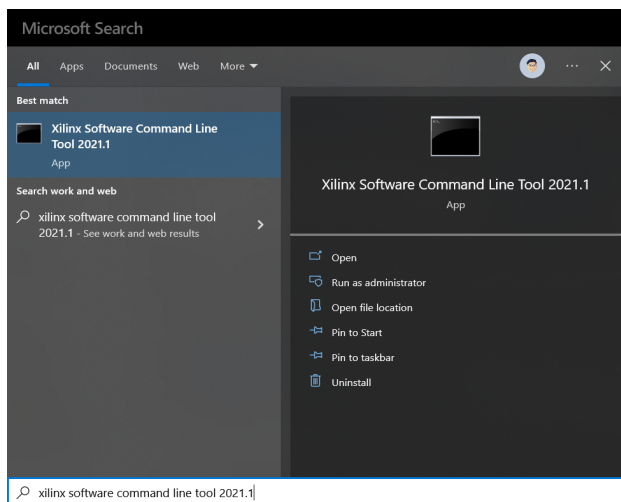
Generating device tree overlay files for the custom hardware module

For the Petalinux image to recognise the interfaces between the Zynq MPSoC and PL hardware, we need device tree overlays to describe the interfaces.

1. Obtain the DTG tool from Xilinx's repo

```
git clone https://github.com/Xilinx/device-tree-xlnx
cd device-tree-xlnx
git checkout xlnx_rel_v2021.1
```
2. Obtain the DTC tool if not obtained:

```
sudo apt install dtc
# or
git clone https://git.kernel.org/pub/scm/utils/dtc/dtc.git
cd dtc
make
export PATH=$PATH:/<path-to-dtc>/dtc
```
3. In Windows, click Start and search for “Xilinx Command Line Tool 2021.1”.



On Ubuntu, you will source the settings.sh file from your Vivado install directory, e.g

```
$ source /tools/Xilinx/Vivado/2021.1/settings64.sh
```

4. Change the directory to the location where the exported XSA file is located. (using PowerShell/Linux commands E.g. `ls`, `pwd`, `cd`)
5. Run the following commands to generate the device tree file for the module which will generate the DTS file (human readable)

```
hsi open_hw_design <design_name.xsa>
hsi set_repo_path <path to device-tree-xlnx repository>
hsi create_sw_design device-tree -os device_tree -proc psu_cortexa53_0
hsi set_property CONFIG.dt_overlay true [hsi::get_os]
hsi generate_target -dir my_dts
hsi close_hw_design [current_hw_design]
```

6. Open a WSL terminal, navigate to the location of the `my_dts` directory generated from the previous step and compile the `.dtsi` file into `.dtbo` using DTC (wsl)
7.

```
cd <path-to>/my_dts
dtc -@ -O dtb -o pl.dtbo pl.dtsi
```

At this point, you have generated the two files needed to run your custom hardware module on the Kria board, including:

- **pl.dtbo** located at `<path-to-vivado-project>/my_dts/pl.dtbo`
- **design_1_wrapper.bin** at `<path-to-vivado-project>/design_1.runs/impl1/design_1_wrapper.bin`

Running custom hardware on Kria

Loading the firmware to Kria

Once the above steps are completed, you should have the `pl.dtbo` and `design_1_wrapper.bin` files ready.

1. `scp` these files to the `/lib/firmware/xilinx/<your-app-name>/` directory on the kria board with the naming convention of `<your-app-name>.dtbo` and `<your-app-name>.bit.bin`.
2. Create a `shell.json` file in the `/lib/firmware/xilinx/<your-app-name>/` directory. Paste the following snippet into the `shell.json` file.

```
{
    "shell_type": "XRT_FLAT",
    "num_slots": "1"
}
```

With the app name of `audio-i2s-dbg`, `/lib/firmware/xilinx/audio-i2s-dbg` should look like this:

```
xilinx-k26-starterkit-2021_1:~$ tree /lib/firmware/xilinx/audio-i2s-dbg
/lib/firmware/xilinx/audio-i2s-dbg
|-- audio-i2s-dbg.bit.bin
|-- audio-i2s-dbg.dtbo
`-- shell.json
```

0 directories, 3 files

Loading the module onto the FPGA

1. Run `xmutil listapps` to list out the available hardware (your list may differ)

```
xilinx-k26-starterkit-2021_1:/home/petalinux# xmutil listapps
```

Type	#slots	Accelerator	Base
		Active_slot	
		audio-i2s-dbg	audio-i2s-dbg
XRT_FLAT	0	-1	
		audio-i2s	audio-i2s
XRT_FLAT	0	-1	
		kv260-dp	kv260-dp
XRT_FLAT	0	0,	

Socket 9 closed by client

2. Unload the default kv260-dp app with `xmutil unloadapp`.

```
xilinx-k26-starterkit-2021_1:/home/petalinux# xmutil unloadapp
DFX-MGRD> daemon removing accel at slot 0

DFX-MGRD> Removing accel kv260-dp from slot 0

Accelerator successfully removed.
Socket 9 closed by client
```

3. Assuming that `audio-i2s-dbg` is the name of your project, you then load the `audio-i2s-dbg` app with `xmutil loadapp audio-i2s-dbg`

```
xilinx-k26-starterkit-2021_1:/home/petalinux# xmutil loadapp
audio-i2s-dbg
DFX-MGRD> daemon loading accel audio-i2s-dbg

DFX-MGRD> Successfully loaded base design.

Accelerator loaded to slot 0
Socket 6 closed by client
```

4. Run the software code. We provide two binaries which you can use for debugging:
 1. The `sample256` binary will transfer 256 samples for each transaction and performs 10 transfers in total.
 2. The `ctrl_bus_test` binary interfaces with the control bus of the module and tries to write `0xdeadbeef` to all the registers. Read-only registers will not be overwritten.

Note that the DFX-MGRD messages will only be printed on serial terminals, ssh sessions will not get these messages.

Programming the PL/PS over USB

1. Unlike with the LED lab, which only used the FPGA, the full project needs both the PL and PS to be configured. However, as long as you are not changing the PS's configuration (i.e. any changes to the Zynq UltraSCALE+ module in your block diagram, including changes to the memory map), then you can program the device over USB **after** you have enabled the app using `xmutil loadapp`. This means that your normal development flow will be:
 - a. Power on the board after connecting it to all cables etc.
 - b. Deactivate the default app using `xmutil unloadapp`
 - c. Activate your app using `xmutil loadapp`
 - d. Program the device using the Hardware Manager from within Vivado (as per the LED lab).
 - e. Repeat step d. as needed

Writing software: Environment Setup

What you'll need:

- sdk.sh: SDK for cross-compilation.

1. Install the SDK by running the self-extracting .sh file. You can download this from Teams.

```
$ ./sdk.sh
PetaLinux SDK installer version 2021.1
=====
Enter target directory for SDK (default: /opt/petalinux/2021.1): <>
You are about to install the SDK to "/data/opt/petalinux". Proceed [Y/n]?
Y
Extracting
SDK.....
.....
.....
.....done
Setting it up...done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to
source the environment setup script e.g.
$ .
/data/opt/petalinux/environment-setup-cortexa72-cortexa53-xilinx-linux
```

2. Once the SDK is installed, on every new terminal that you wish to build software for the Kria board, run the following command to set the cross-compilation environment.

```
.
<path-to-SDK-installation>/environment-setup-cortexa72-cortexa53-xilinx-l
inux
```

3. Check if the environment is set with `echo $CC`. The following output means the cross-compilation environment has been successfully set.

```
aarch64-xilinx-linux-gcc -mcpu=cortex-a72.cortex-a53 -march=armv8-a+crc
-fstack-protector-strong -D_FORTIFY_SOURCE=2 -Wformat -Wformat-security
-Werror=format-security
--sysroot=/data/opt/petalinux/sysroots/cortexa72-cortexa53-xilinx-linux
```

4. In the software example code, run make and transfer the binary to the Kria board (e.g. using a USB drive or scp command) and run with the hardware module loaded.
5. Use the code in app/src on Teams. Read what has been provided to you, understand it, and then your first step should be to try to replicate the functionality of the sample256 example program.
6. For the basic requirements, you must save sampled audio as a .wav file.