



ECE4810J SoC Design

Fall 2024

Lab #1 Getting Started with Arty Z7 SoC Development Platform

Due: 11:59pm Oct. 7th, 2024

Logistics:

- This lab is a team exercise.
- Please use the discussion board on Piazza for Q&A.
- All reports and code (if available) MUST be submitted to the assignment of Canvas.
- Internet usage is allowed and encouraged.
- Late penalty (10% per day) will be applied for this lab.

Contents

1 Overview	2
2 Build a basic ZYNQ system on the Arty Z-7 board	2
2.1 Create a new Vivado project	2
2.2 Setup the Zynq Processor System (PS)	3
2.3 Create the HDL wrapper	7
2.4 Generate the bitstream	7
2.5 Export the hardware to Vitis	8
2.6 Create a Software application	9
2.7 Test the design on the hardware	10
3 Build a basic ZYNQ system with peripherals	14
4 Create and use custom IP blocks in Verilog	18
4.1 Create and package the IP block	18
4.2 Add the IP block to the Zynq PS System	21
4.3 Interfacing with the IP Block in software	23
4.4 Follow-up assignment	23
5 (Optional) Create an FIR filter by MATLAB HDL Coder	24
5.1 Direct FIR filter and symmetric FIR filter	24
5.2 System Design with HDL Code Generation from MATLAB and Simulink	25
5.3 Basic HDL Code Generation and FPGA Synthesis from MATLAB	28
5.4 Generate HDL Code from Simulink Model	29

6	Tcl exercises	35
6.1	Average	35
6.2	Factorial	35
6.3	Two-fer	35
7	Post-lab questions	35
8	Deliverables	36
9	Grading policy	36
10	Acknowledgment	37
A	Peer Evaluation Form	38
B	Troubleshooting	39
B.1	Vivado Export Hardware	39
B.2	Launch Xilinx Vitis	39
B.3	Vitis Build	41
B.4	Vitis Program Device	43
B.5	AXI IP Connection	44
B.6	Vivado Startup	44

1 Overview

In this lab, you will go through several exercises to set up and get started with the Arty Z7-20 SoC development board. The goals of this lab are to:

- Get familiar with the board
- Set up the environment correctly
- Be able to build a basic Zynq system on the board
- Learn to create custom IP blocks at RTL level (Verilog, VHDL)
- Use AXI bus to connect an IP block with the Zynq PS

2 Build a basic ZYNQ system on the Arty Z-7 board

Before building, make sure you have done environment setup in Lab#0.

2.1 Create a new Vivado project

Open Xilinx Vivado 2022.1, Quick Start->Create Project. Enter the name of your project and specify a directory where the project data files will be stored. When specifying the type of project to create, choose RTL Project, but do not check "Project is an extensible Vitis platform". When choosing a default Xilinx part or board for your project, choose

"Arty Z7-20". If the board is not listed, go back to the environment setup in Lab #0 and check whether the board files are adequately prepared.

```
start_gui
create_project lab1_1 F:/Documents/GitHub/ECE4810J_FA2022_TA/Lab1/lab1_1
→ -part xc7z020clg400-1
set_property board_part digilentinc.com:arty-z7-20:part0:1.1
→ [current_project]
```

2.2 Setup the Zynq Processor System (PS)

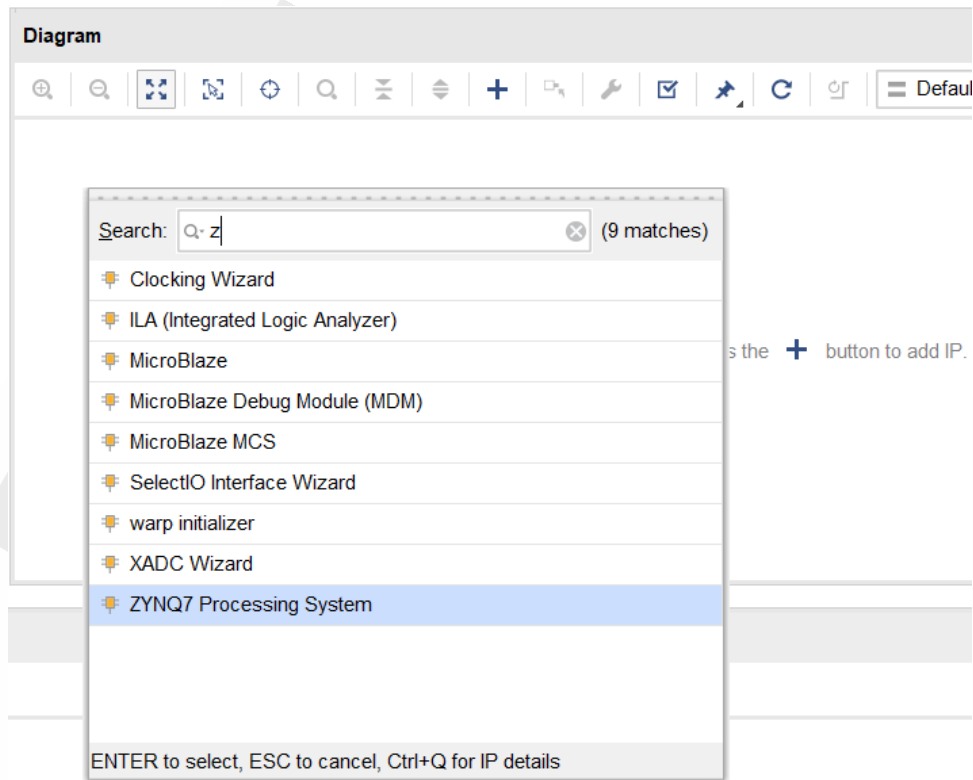
The new Vivado project starts off blank, so to create a functional base design, we need at least to add the Zynq PS (processor system) and make the minimal required connections. Follow these steps to add the PS to the project:

- Flow Navigator->PROJECT MANAGER->IP INTEGRATOR->Create Block Design. Set the Design name as "Zynq_CPU".

```
create_bd_design "Zynq_CPU"
```

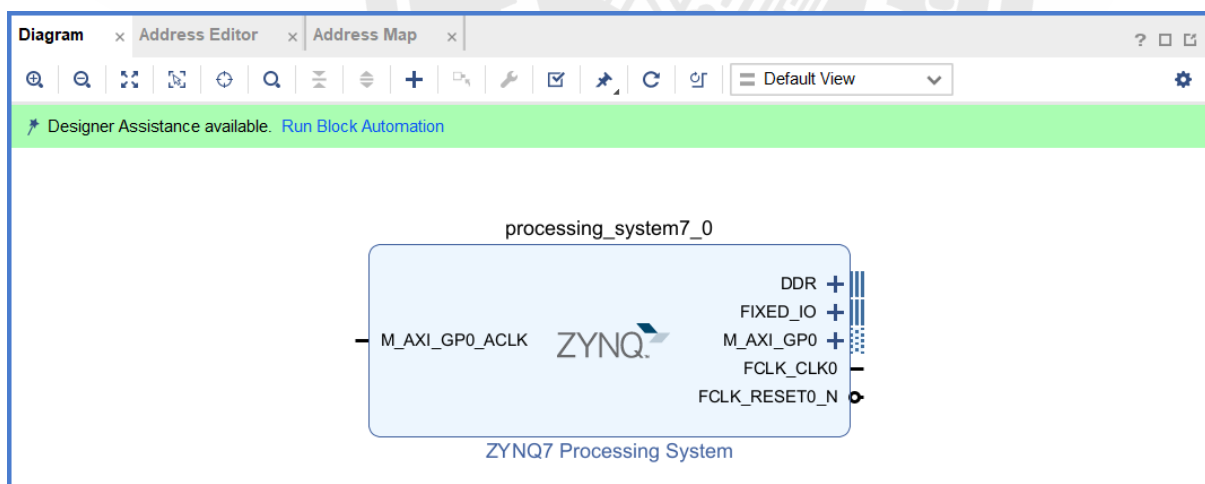
- In Diagram window, it shows "This design is empty. Press the + button to add IP." Press the + (Add IP) button or Ctrl+I; search for "ZYNQ7 Processing System"; double-click it to add IP. If you cannot find "ZYNQ7 Processing System", just delete the project and re-create one.

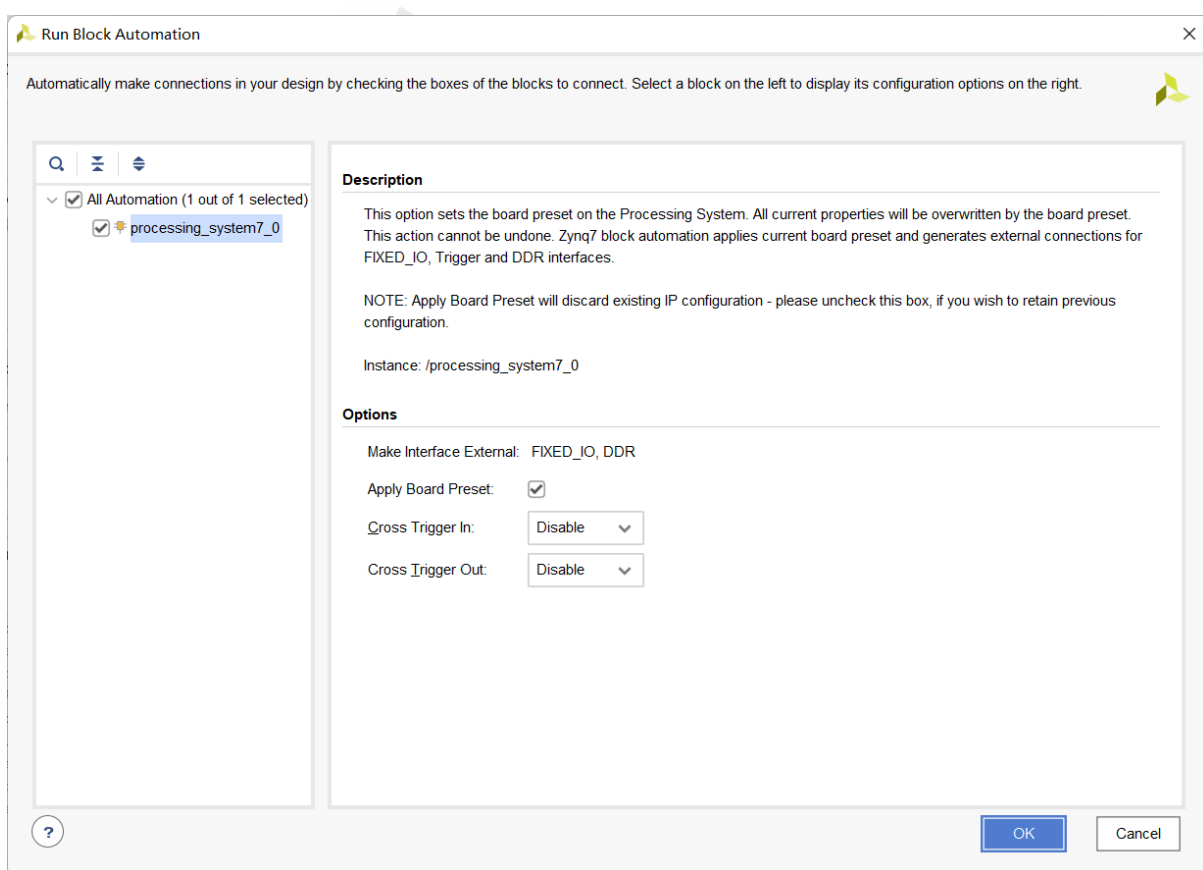
```
update_compile_order -fileset sources_1
startgroup
create_bd_cell -type ip -vlnv xilinx.com:ip:processing_system7:5.5
→ processing_system7_0
endgroup
```



- Click "Run Block Automation":

```
apply_bd_automation -rule xilinx.com:bd_rule:processing_system7
→ -config {make_external "FIXED_IO, DDR" apply_board_preset "1"
→ Master "Disable" Slave "Disable" } [get_bd_cells
→ processing_system7_0]
```

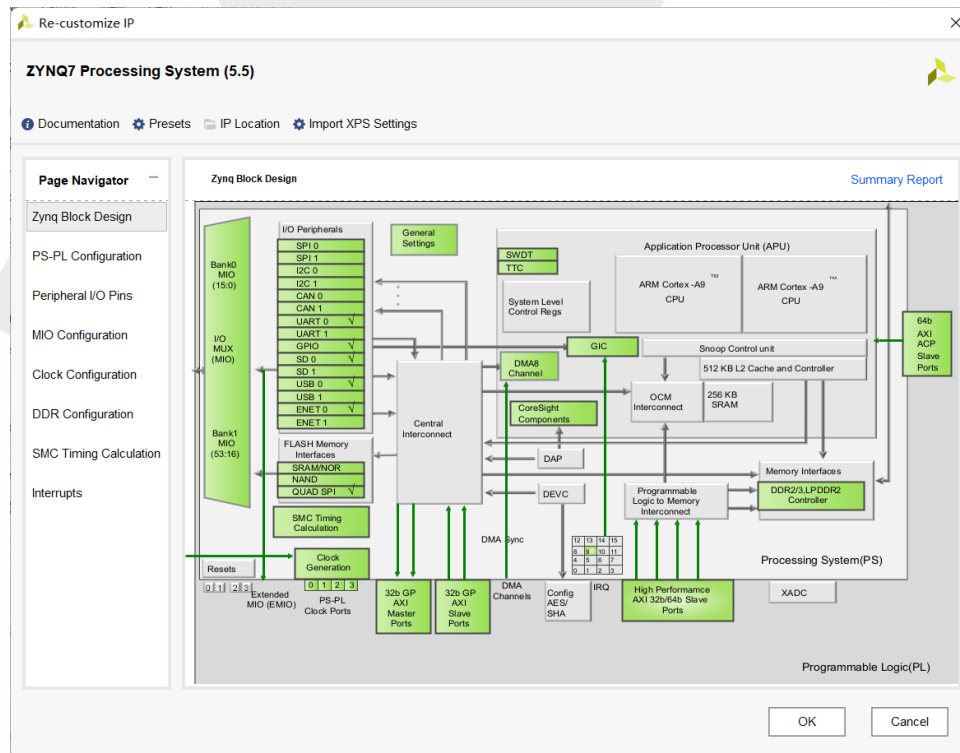




In the pop-up window "Run Block Automation", select `processing_system7_0`. Block Automation makes connections and pin assignments to external hardware such as the DDR and fixed IO using the board definition of the hardware platform.

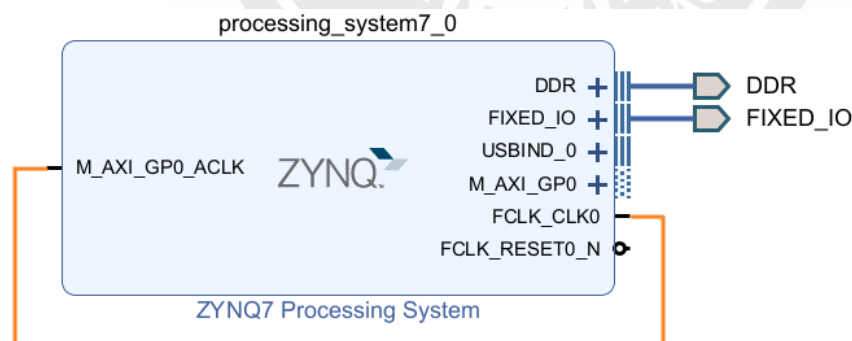
- Now the block diagram has changed, and we can see that the DDR and `FIXED_IO` are connected externally. Double-click the ZYNQ PS block and explore the customization options available in "Re-customize IP" dialog. The Zynq PS can generate multiple clocks that are then provided to the FPGA fabric. These clocks are referred to as FCLKs, and can be found under the *PL Fabric Clocks* dropdown in the *Clock Configuration* tab of the Zynq PS configuration wizard. They can be enabled or disabled with a checkbox, the hardware used to drive the clock can be changed, and the frequency can be modified. All board files for Digilent Zynq boards enable a single Zynq PL clock by default, which is intended to be used with peripherals connected to the Zynq's `M_AXI_GP0` port. Some designs may require additional clocks of specific frequencies to be added to your design. In these cases, enable a second clock and specify the needed frequency. Zynq devices can also use interrupts generated in FPGA fabric to trigger interrupts within the Processing System. Interrupt-related settings can be changed within the configuration wizard's interrupts tab. These interrupts typically use the `IRQ_F2P` port, which can be found under the Fabric Interrupts → `IRQ_F2P` dropdown. To enable this port, both the Fabric Interrupts and `IRQ_F2P` ports must be enabled. While interrupts can be directly connected to the `IRQ_F2P` port (by clicking and dragging from one port to another), some designs may require multiple interrupt

sources. In these cases, add a "Concat" IP to your block design, and manually connect it to the IRQ_F2P port. Additional input ports can be added to a "Concat" block through its configuration wizard (opened by double clicking on the IP). In this section, just leave all configurations as default.



- The only remaining connection to make is the clock that we will use for the AXI buses. Connect the FCLK_CLK0 output to the M_AXI_GP0_ACLK clock input. To do this, click on the FCLK_CLK0 output and drag with the pencil onto the M_AXI_GP0_ACLK input. This will trace a wire between the pins and make the connection.

```
connect_bd_net [get_bd_pins processing_system7_0/FCLK_CLK0]
→ [get_bd_pins processing_system7_0/M_AXI_GP0_ACLK]
```



2.3 Create the HDL wrapper

Now the Zynq Processing System is set up, and all we need to do is to create an HDL wrapper for the design.

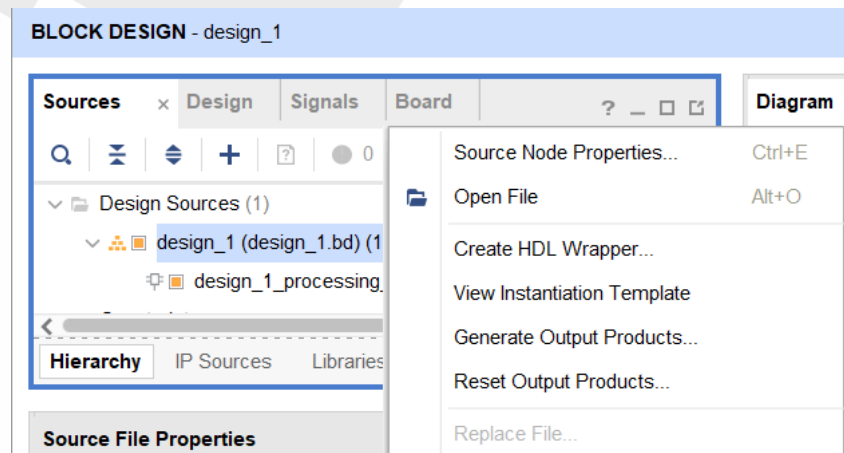
- Save the Block Design, then, under the Sources tab->Design Sources, right-click on Zynq_CPU and select Create HDL wrapper... and then choose "Let Vivado manage wrapper and auto-update".

```
save_bd_design
```

```
make_wrapper -files [get_files F:/Documents/GitHub/ECE4810J_FA2022_]
→ TA/Lab1/lab1_1/lab1_1.srscs/sources_1/bd/Zynq_CPU/Zynq_CPU.bd]
```

```
→ -top
```

```
add_files -norecurse f:/Documents/GitHub/ECE4810J_FA2022_TA/Lab1/la_
→ b1_1/lab1_1.gen/sources_1/bd/Zynq_CPU/hdl/Zynq_CPU_wrapper.v
```



We now have a base design containing the Zynq PS from which we could generate a bitstream and test on the Arty Z7 board. We have not exploited any of the FPGA fabric, but the Zynq PS is already connected to the DDR, Gigabit Ethernet PHY, the USB PHY, the SD card, the UART port and the GPIO, all thanks to the Block Automation feature. So there is already quite a lot we could do with the design at this point, such as running Linux on the PS or running a bare metal application on it.

2.4 Generate the bitstream

- Flow Navigator-> PROGRAM AND DEBUG->Generate Bitstream.

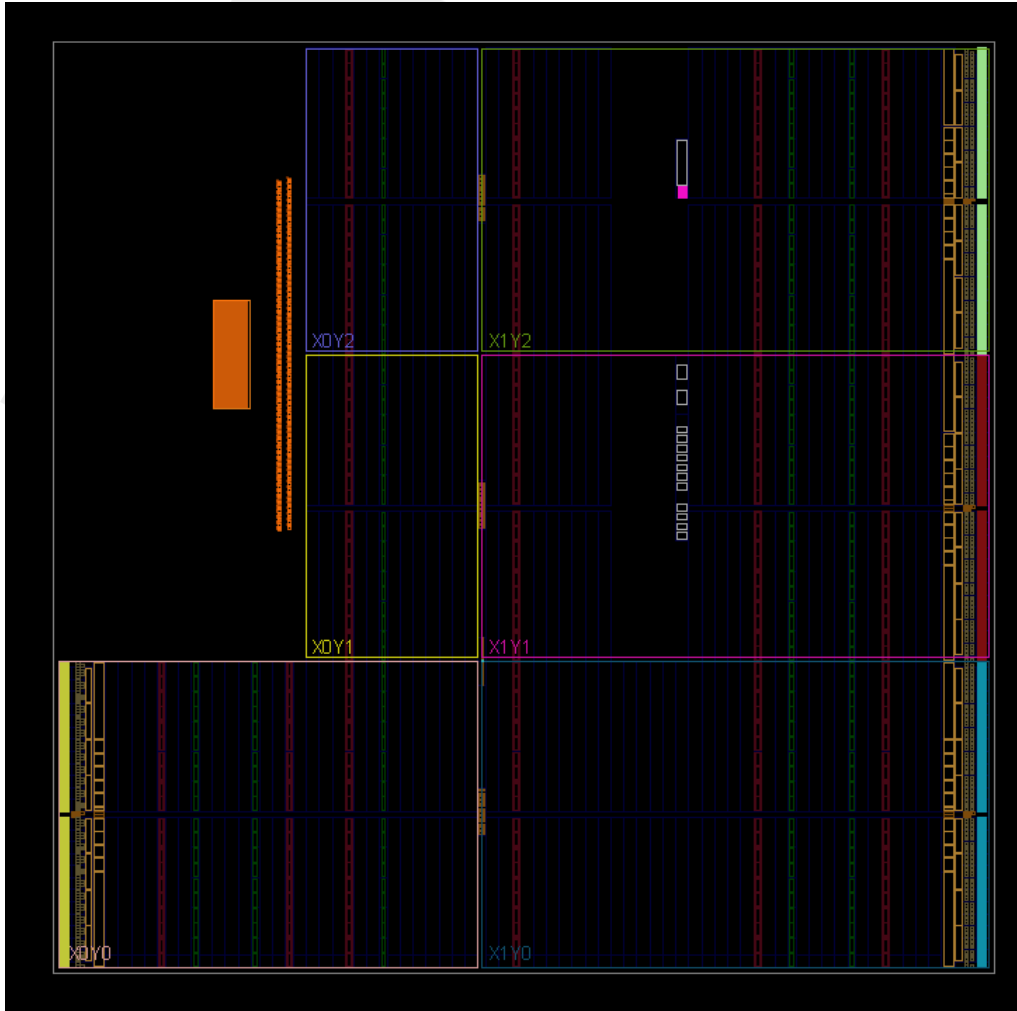
```
launch_runs impl_1 -to_step write_bistream -jobs 6
```

- In the pop-up window "Bitstream Generation Completed", select Open Implemented Design.

```
open_run impl_1
```

- The implemented design will open in Vivado showing you a map of the Zynq device and how the design has been placed. In our case, we haven't used any of the FPGA

fabric (only the PS), so the map is empty for the most part.

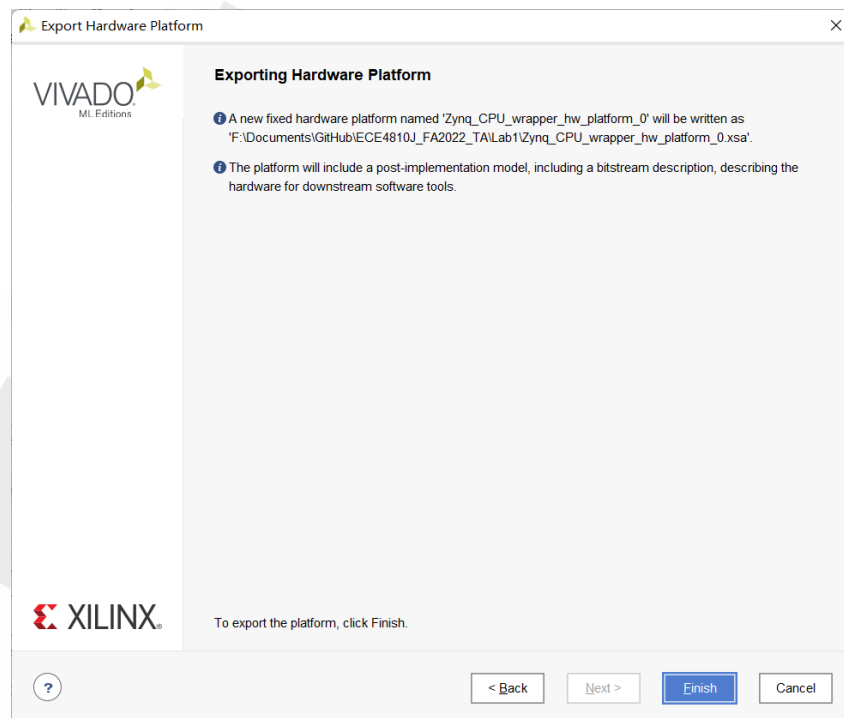


2.5 Export the hardware to Vitis

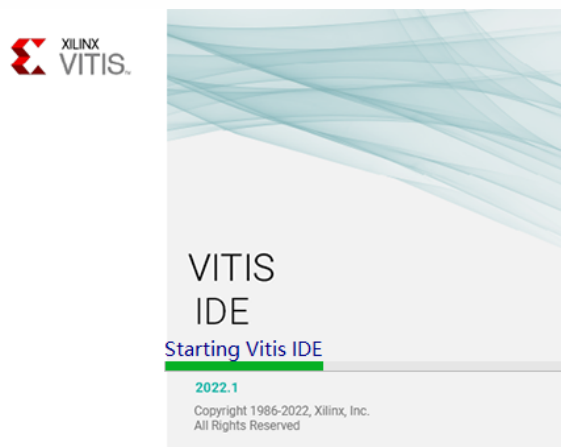
Once the bitstream has been generated, the hardware design is done, and we are ready to develop the code to run on the processor. This part of the design process is done in Xilinx Vitis, so from Vivado we must first export the project to Vitis.

- Vivado->File menu->Export->Export Hardware..., then in pop-up window "Export Hardware Platform", click Next, select "Include bitstream", click Next, input "XSA file name" as Zynq_CPU_wrapper_hw_platform_1, click Next, click Finish. If you use an older version, you may encounter problems; see Appendix B.1.

```
write_hw_platform -fixed -include_bit -force -file F:/Documents/Git_
  ↳ Hub/ECE4810J_FA2022_TA/Lab1/Zynq_CPU_wrapper_hw_platform_1.xsa
```

- Vivado->Tools menu->Launch Vitis IDE. Select workspace and launch.



If Vitis fails to start up or does not respond at start-up, see Appendix [B.2](#). The Vitis workspace will be empty at first.

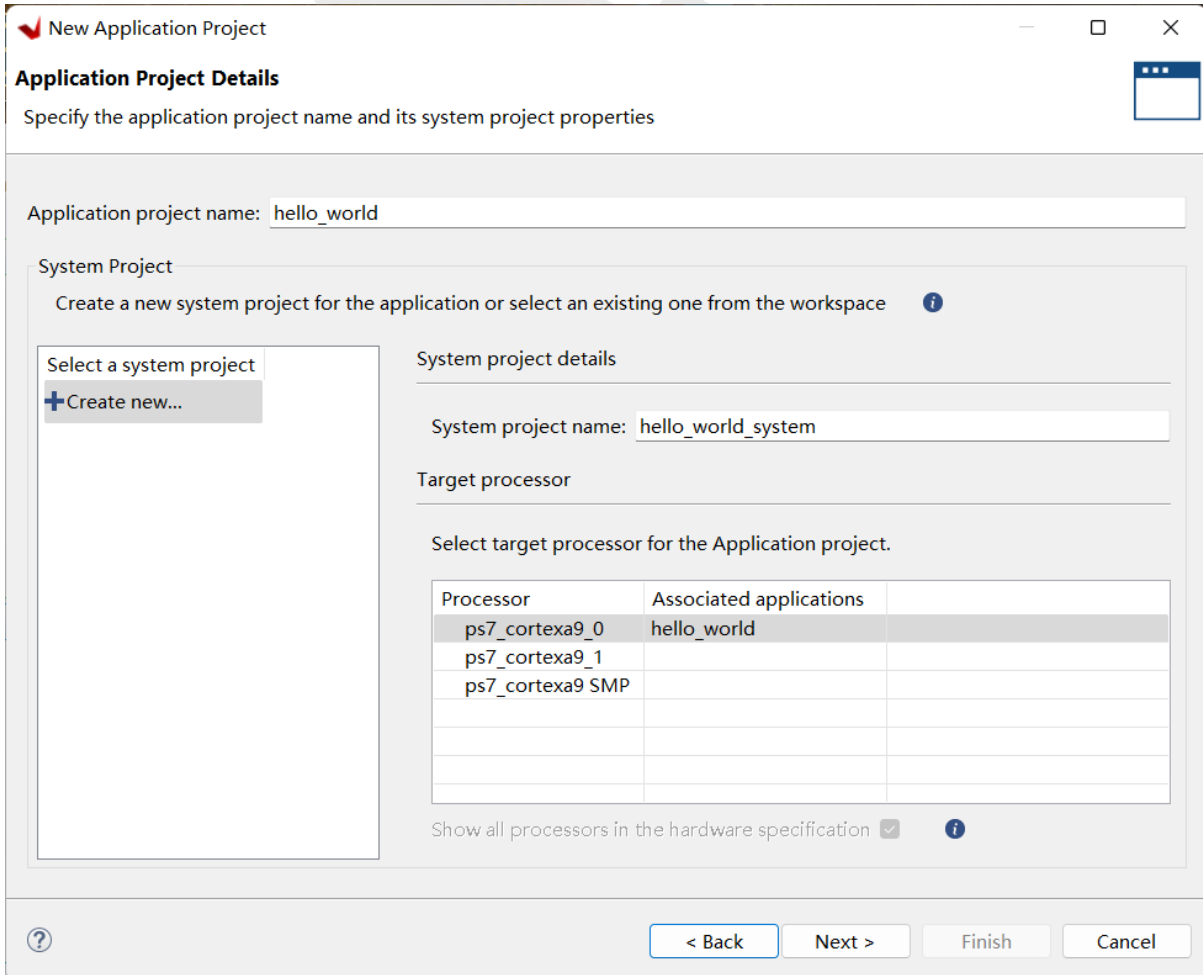
2.6 Create a Software application

To demonstrate creating an application for the Zynq, we'll create a hello world application that will send "Hello, World!" through UART and to your PC.

- In the File menu, select New->Application Project; click Next.
- When choosing a platform for your project, go to the "Create a new platform from hardware (XSA)" tab, in the "Hardware Specification" frame, click "Browse...", se-

lect our previously created `Zynq_CPU_wrapper_hw_platform_1.xsa`. In the "Boot Components" frame, check "Generate boot components". Click Next.

- Specify the application project name as `hello_world`. Click Next.



New Application Project

Application Project Details

Specify the application project name and its system project properties

Application project name: `hello_world`

System Project

Create a new system project for the application or select an existing one from the workspace

Select a system project

+ Create new...

System project details

System project name: `hello_world_system`

Target processor

Select target processor for the Application project.

Processor	Associated applications
ps7_cortexa9_0	hello_world
ps7_cortexa9_1	
ps7_cortexa9 SMP	

Show all processors in the hardware specification

< Back Next > Finish Cancel

- When specifying a domain, keep defaults. Click Next. When selecting a template to create your project, under "Embedded software development templates", select "Hello World" template. Click Finish.

Vitis will generate two new project folders, `hello_world` and `hello_world_system`. The `hello_world_system` is the system project containing the application project `hello_world`.

In Explorer window, click `hello_world_system` to select it. In the Project menu, you must manually click "Build Project" to build; otherwise, you will not be able to run (see Appendix B.3). You can monitor the build in Console. Once the build is finished, it will output "Build Finished (took 6s.673ms)".

2.7 Test the design on the hardware

1. Grab an Arty Z7 board and connect the USB cable to your PC and to the *PROG UART* port.

2. Set the *JP4* jumper to *JTAG* mode to enable chip configuration via the JTAG.
3. Set the *JP5* jumper to *USB* mode to enable power supply via the USB.
4. You should see the LED *LD13* shine red.

Remember to set *JP4* to JTAG; otherwise, an error will occur (see Appendix B.4). Now you need to set up the UART connection between the PC and the board.


- In Windows 10 or Windows 11, right-click on the Windows (Start Menu) button and open Device Manager.
- Under Ports, one of the available COM ports, probably the latest ones, a USB Serial Port, is the Arty Z7 board. Double-click on it or right-click->Properties, go to the "Port Settings" tab, and configure it as follows:

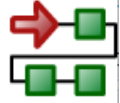
Bits per second	115200
Data bits	8
Parity	None
Stop bits	1
Flow control	None

Table 1: USB Serial Port Properties – Port Settings.

- Note: If you cannot set the Port Settings in Windows due to a lack of administrator permissions, do not worry. Setting the correct settings later in the Vitis Serial Terminal should be sufficient.

Now that your PC is ready to receive the "Hello World" message, we are ready to send our bitstream and software application to the hardware. In the "Xilinx" menu of Xilinx Vitis, click "Program Device".


Program Device
✕

Program Device


Specify the bitstream and the ELF files that reside in BRAM memory

Project:

Connection:

Device:

Bitstream/PDI:

☐ Partial Bitstream

BMM/MMI File:

Software Configuration

Processor	ELF/MEM File to Initialize in Block RAM

☐ Skip Revision Check

Click "Program". If successful, there will be no error messages, and the LED LD12 on the Arty Z7 board will shine yellow. If you see an error:

Program Device failed

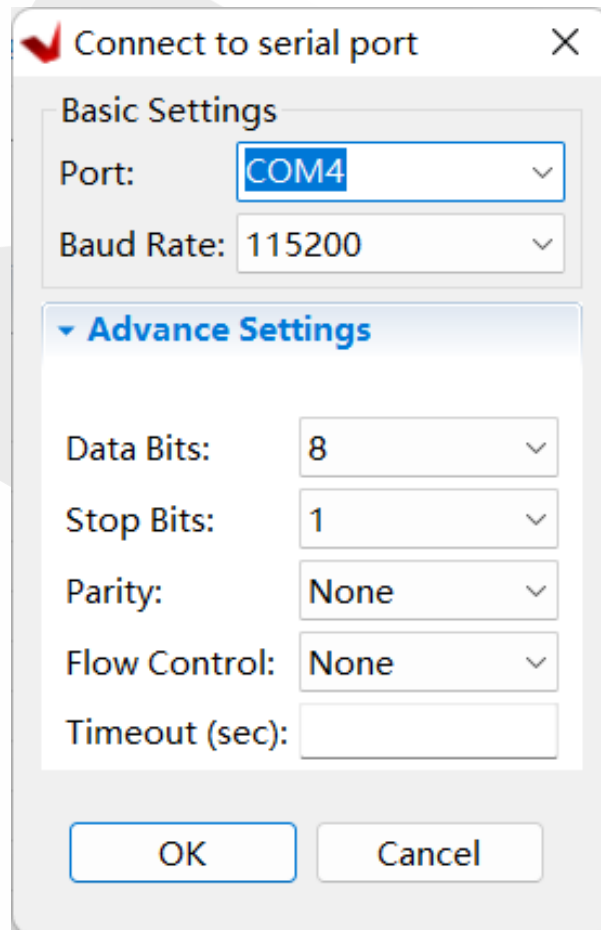
Reason:

Invalid target. User "connect" command to connect to hw_server/TCF agent see Appendix [B.4](#).

Press Ctrl+3 or click the "Access commands and other items" button on the top right. Type and select "Vitis Serial Terminal (Xilinx)" to open it on the bottom right.

Press the green plus button "Connect to serial port" on the top right of the Vitis Serial

Terminal. Select the Port that matches the COM port of Arty Z7 in Device Manager. Keep the Baud Rate of Basic Settings and Advance Settings default.



You should then see the message "Connected to COMx at 115200" in the Vitis Serial Terminal and "Connected to: Serial (COM4, 115200, 0, 8)" in the top bar of the Vitis Serial Terminal. Lastly, select "hello_world" under hello_world_system in Explorer. Below the menubar, press the black triangle right to the green "Run" button. Select Run As->3 Launch Hardware (Single Application Debug (GDB)). This step may automatically do "Program Device". The program will run on the board, and you will see the message in the Vitis Serial Terminal:

```
Hello World
```

```
Successfully ran Hello World application
```

Take ONE screenshot of the whole Vitis IDE window and ONE screenshot of the Vitis Serial Terminal.

3 Build a basic ZYNQ system with peripherals

You have been familiar with Vivado – Vitis workflow without any software or environment issues. Now you are going to take a step further.

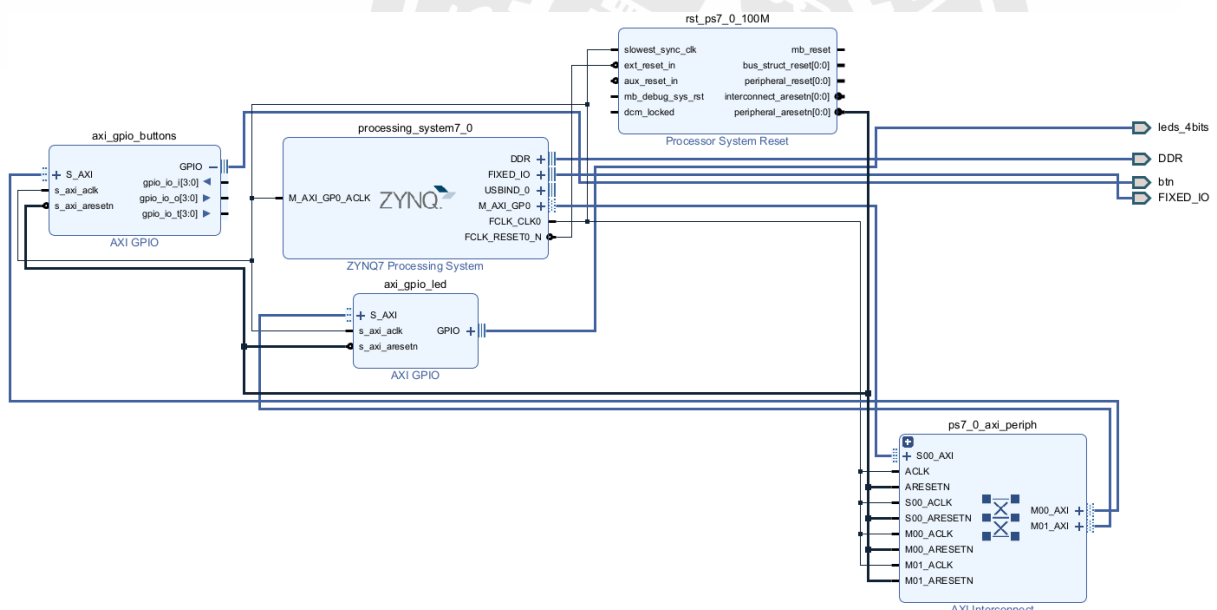
1. Create a new RTL project in Vivado with the name `lab1_2`. Create a block design named "PmodIPs". Pmod is the Digilent defined standard of Peripheral Modules, small I/O interface boards that offer an ideal way to extend the capabilities of programmable logic and microcontroller boards [2]. Add a ZYNQ7 Processing System IP and Run Block Automation. Keep default and check *Apply Board Preset* in the pop-up dialog. In the "BLOCK DESIGN" window->"Board" tab->Arty Z7-20->GPIO->4 LEDs, right click->Connect Board Component... In the pop-up dialog "Connect Board Component", select AXI GPIO->GPIO, and click OK. This will add the IP to your design and connect it to an external port, which will not require any further work to constrain.
2. Next, select the `axi_gpio_0` block. The *Block Properties* pane to the left of the Diagram and below the Board tab will allow you to view some information about the block and modify it in some ways without running through its customization wizard. For now, just change its name to `axi_gpio_led` by typing in the Name field. Pressing enter or clicking out of the text box confirms the change. Using memorable names in your block design makes it easier to remember which IP does what when you are later writing software in Vitis.
3. Next, a second AXI GPIO IP will be manually added to the block diagram and manually constrained with an XDC file. Click the "Add IP" button (+) and search for "AXI GPIO". Double-click on the only result to add the second AXI GPIO block to the design. Once added, rename this IP `axi_gpio_buttons`.
4. Select the `axi_gpio_buttons` IP's GPIO interface by clicking on the text "GPIO", right-click on the highlighted text, and select **Make External**. This option creates a new external interface port that does not rely on the board files. Because the board files are not used here, a Xilinx Design Constraint (XDC) file must be added to the project to tell Vivado which FPGA pins to connect the interface to.
5. Download `Arty-Z7-20-Master.xdc` from [digilent-xdc](#). In Vivado Flow Navigator->PROJECT MANAGER->Add Sources or press Ctrl+A or click "+" button in the "Sources" tab. Choose "Add or create constraints" in "Add Sources" pop-up dialog and add `Arty-Z7-20-Master.xdc`. Master XDC files for Digilent boards contain pin constraints for I/O interfaces the board offers. These constraints are sorted by the interface. Scroll down until you see constraints for the user buttons. These constraints typically are for a bus port named "btn". Uncomment the button constraints by removing the single leading '#' character in each line corresponding to the buttons.
6. Next, the names of the block design's GPIO port for the buttons must be determined so that the buttons can be properly constrained. Return to the Diagram tab, and select the `GPIO_0` external port that is connected to the `axi_gpio_buttons` block. Change the name of the external interface to "btn" in the Properties pane.

7. The AXI GPIO IP automatically uses tri-state buffers for the pins that its interfaces are connected to. The individual I, O, and T buses can be seen when expanding the interface through the plus button (+) next to the interface name (GPIO) on the IP block. As can be seen, the individual ports that make up the interface are named `<interface>_tri_i`, `<interface>_tri_o`, and `<interface>_tri_t`. When constrained to tri-state buffers, the bus that is connected to FPGA ports is named `<interface>_tri_io`.

With this knowledge, return to the XDC file, and change the name of the button bus that is constrained. Specifically, change the text after the `get_ports` call on each line of the button interface to `btn_tri_io[0]` to `btn_tri_io[3]`.

```
set_property -dict { PACKAGE_PIN D19      IOSTANDARD LVCMOS33 }
→ [get_ports { btn_tri_io[0] }]; #IO_L4P_T0_35 Sch=BTN0
set_property -dict { PACKAGE_PIN D20      IOSTANDARD LVCMOS33 }
→ [get_ports { btn_tri_io[1] }]; #IO_L4N_T0_35 Sch=BTN1
set_property -dict { PACKAGE_PIN L20      IOSTANDARD LVCMOS33 }
→ [get_ports { btn_tri_io[2] }]; #IO_L9N_T1_DQS_AD3N_35 Sch=BTN2
set_property -dict { PACKAGE_PIN L19      IOSTANDARD LVCMOS33 }
→ [get_ports { btn_tri_io[3] }]; #IO_L9P_T1_DQS_AD3P_35 Sch=BTN3
```

8. With the constraints for the port finished, the AXI GPIO must be manually configured. In particular, the width of the GPIO interface must match the number of buttons available on the board. Take note of how many buttons are constrained in the XDC. Return to the Diagram tab, and double-click on the `axi_gpio_buttons` block. In "Re-customize IP" pop-up dialog, switch to the "IP Configuration" tab. Enter the number of buttons you constrained into the GPIO interface's GPIO Width field. When finished, click OK to save your changes. Finally, click the Run Connection Automation button in the green Designer Assistance bar. In the pop-up dialog "Run Connection Automation", check "All Automation".



In the unlikely event that Vivado fails to correctly assign addresses to each AXI IP connected to your processor, you may need to set their addresses manually. See Appendix B.5.

9. Before the Vivado project can be built, the block design must be validated. This step runs an automatic check of the block design to see if there are any potential issues with it. Click the Validate Design button in the Diagram pane's toolbar or press F6.
10. Create an HDL wrapper for PmodIPs. Generate bitstream. In the pop-up dialog, specify the Number of jobs in the Options frame as larger as possible to accelerate synthesis and implementation. If specifying 12, the process of building can take less than 5 minutes generally.
11. Export the hardware as Zynq_CPU_wrapper_hw_platform_2.xsa, including bitstream.
12. In Vitis, create a new application project named `pmod_app`. Select `Zynq_CPU_wrapper_hw_platform_2.xsa` as the hardware and the "Empty Application (C)" template under "Embedded software development templates". Right-click Explorer->`pmod_app_system`->`pmod_app`->`src`->New->File. In the pop-up dialog "Create New File", specify the file name as `main.c`.

```
1 start_gui
2 create_project lab1_2 F:/Documents/GitHub/ECE4810J_FA2022_TA/Lab1/lab1_2
  → -part xc7z020clg400-1
3 set_property board_part digilentinc.com:arty-z7-20:part0:1.1
  → [current_project]
4 create_bd_design "PmodIPs"
5 update_compile_order -fileset sources_1
6 startgroup
7 create_bd_cell -type ip -vlnv xilinx.com:ip:processing_system7:5.5
  → processing_system7_0
8 endgroup
9 apply_bd_automation -rule xilinx.com:bd_rule:processing_system7 -config
  → {make_external "FIXED_IO, DDR" apply_board_preset "1" Master
  → "Disable" Slave "Disable"} [get_bd_cells processing_system7_0]
10 startgroup
11 create_bd_cell -type ip -vlnv xilinx.com:ip:axi_gpio:2.0 axi_gpio_0
12 apply_board_connection -board_interface "leds_4bits" -ip_intf
  → "axi_gpio_0/GPIO" -diagram "PmodIPs"
13 endgroup
14 set_property name axi_gpio_led [get_bd_cells axi_gpio_0]
15 startgroup
16 create_bd_cell -type ip -vlnv xilinx.com:ip:axi_gpio:2.0 axi_gpio_0
17 endgroup
18 set_property name axi_gpio_buttons [get_bd_cells axi_gpio_0]
19 startgroup
```

```

20 make_bd_intf_pins_external [get_bd_intf_pins axi_gpio_buttons/GPIO]
21 endgroup
22 add_files -fileset constrs_1 -norecurse
   → F:/Documents/Master/TA/Diligent/diligent-xdc/Arty-Z7-20-Master.xdc
23 set_property name btn [get_bd_intf_ports GPIO_0]
24 startgroup
25 set_property -dict [list CONFIG.C_GPIO_WIDTH {4}] [get_bd_cells
   → axi_gpio_buttons]
26 endgroup
27 startgroup
28 apply_bd_automation -rule xilinx.com:bd_rule:axi4 -config { Clk_master
   → {Auto} Clk_slave {Auto} Clk_xbar {Auto} Master
   → {/processing_system7_0/M_AXI_GP0} Slave {/axi_gpio_buttons/S_AXI}
   → ddr_seg {Auto} intc_ip {New AXI Interconnect} master_apm {0}}
   → [get_bd_intf_pins axi_gpio_buttons/S_AXI]
29 apply_bd_automation -rule xilinx.com:bd_rule:axi4 -config { Clk_master
   → {Auto} Clk_slave {Auto} Clk_xbar {Auto} Master
   → {/processing_system7_0/M_AXI_GP0} Slave {/axi_gpio_led/S_AXI}
   → ddr_seg {Auto} intc_ip {New AXI Interconnect} master_apm {0}}
   → [get_bd_intf_pins axi_gpio_led/S_AXI]
30 endgroup
31 validate_bd_design
32 make_wrapper -files [get_files F:/Documents/GitHub/ECE4810J_FA2022_TA/La
   → b1/lab1_2/lab1_2.srscs/sources_1/bd/PmodIPs/PmodIPs.bd]
   → -top
33 add_files -norecurse f:/Documents/GitHub/ECE4810J_FA2022_TA/Lab1/lab1_2/
   → lab1_2.gen/sources_1/bd/PmodIPs/hdl/PmodIPs_wrapper.v
34 launch_runs impl_1 -to_step write_bitstream -jobs 12 launch_runs impl_1
   → -to_step write_bitstream -jobs 12
35 write_hw_platform -fixed -include_bit -force -file F:/Documents/GitHub/E
   → CE4810J_FA2022_TA/Lab1/lab1_2/Zynq_CPU_wrapper_hw_platform_2.xsa

```

Your task is to write a simple C program based on lab1_starter/lab1-2.c to realize:

1. Initialize the AXI GPIO IPs and drivers.
2. The application constantly loops, checking whether any button is pressed.
3. If any button is pressed, set the **corresponding** LED high. For example, pressing BTN3 should set LD3 high. The buttons should light LEDs independently; for example, to press BTN3->press BTN2 & BTN3->press BTN2->release (the 4 steps) should light LD3->light LD2 & LD3->light LD2->put out.
4. When no buttons are pressed, the LEDs are held low.

Note that in the starter files, `xil_printf` is ended by `\r\n` for Windows. If you use Linux or Mac OS, remember to change them to `\n`. Build and run the project and play with the board. Take four photos of the four steps showing four buttons and four LEDs.

4 Create and use custom IP blocks in Verilog

A system on a chip consisting of both a Hard Processor System and FPGA fabric, such as the Zynq-7000, offers the opportunity of offloading computation to the FPGA. Parallelizable algorithms can thus be accelerated, or more computations can be executed in parallel. In this section, we will create a simple custom AXI IP block that multiplies two numbers and will connect it to the Zynq PS. The multiplier will take as input two 16-bit unsigned numbers and will output the product as one 32-bit unsigned number. A single 32-bit write to the IP block will contain the two 16-bit inputs, separated into the lower and higher 16 bits. A single 32-bit read from the peripheral will contain the result from the multiplication of the two 16-bit inputs. Although this design does not really make much sense as an accelerator, it is a good learning example.

4.1 Create and package the IP block

To make your work easier, you can start with the system you created in section 2.

1. Start by going to menu Tools -> Create and Package New IP...
2. We are interested in a new AXI4 peripheral; therefore select "Create a new AXI4 peripheral" and click Next. To read more about AXI, please go to Canvas>File>Reading Materials>Zynq>ug761_axi_reference_guide.pdf and Xilinx's series knowledge base articles

[AXI Basics 1 - Introduction to AXI](#)

[AXI Basics 2 - Simulating AXI interfaces with the AXI Verification IP \(AXI VIP\)](#)

[AXI Basics 3 - Master AXI4-Lite simulation with the AXI VIP](#)

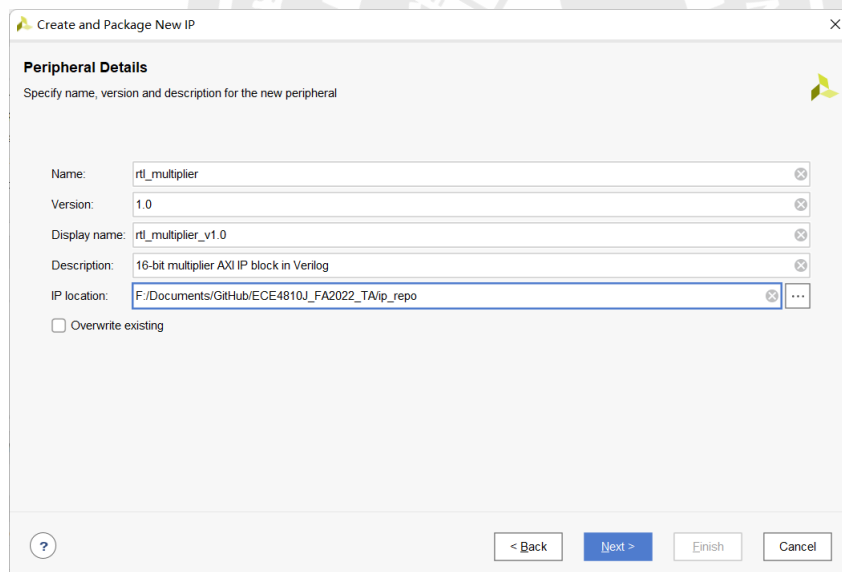
[AXI Basics 4 - Using the AXI VIP as protocol checker for an AXI4 Master interface](#)

[AXI Basics 5 - Create an AXI4-Lite Sniffer IP to use in Xilinx Vivado IP Integrator](#)

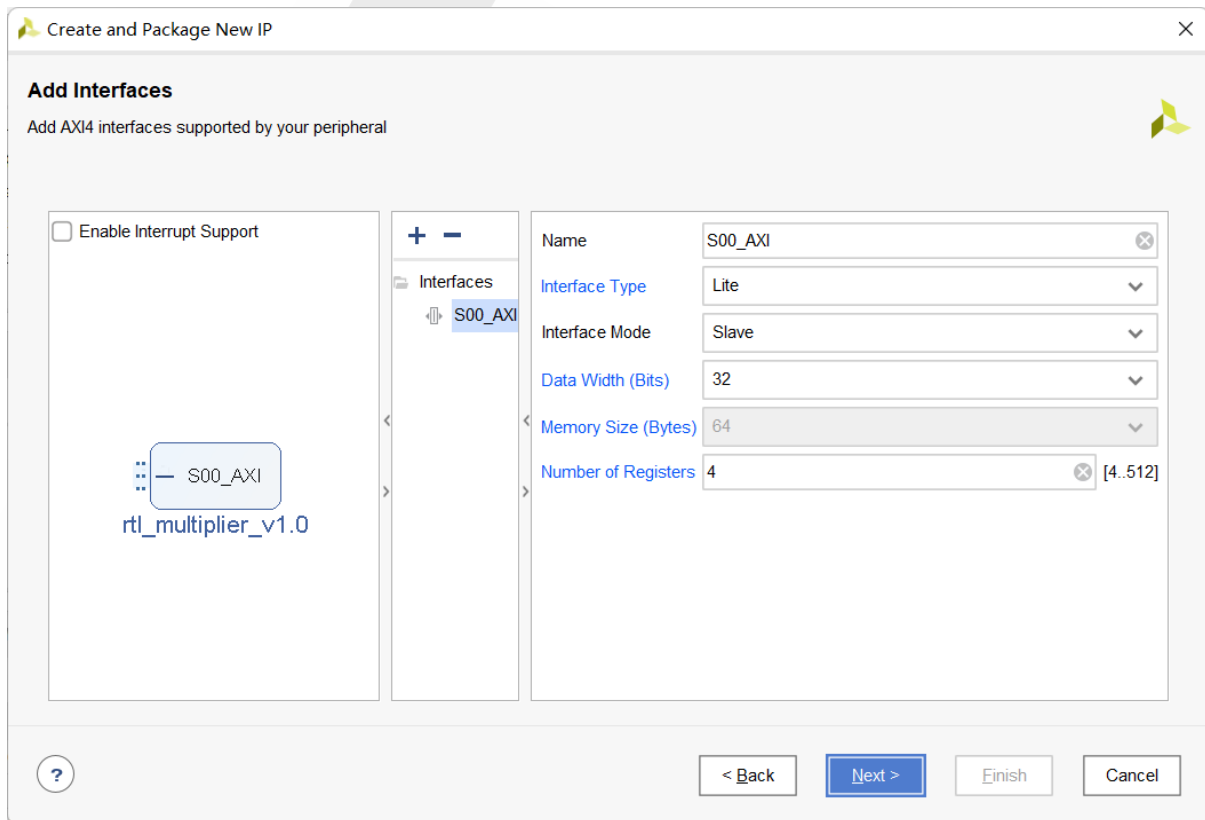
[AXI Basics 6 - Introduction to AXI4-Lite in Vitis HLS](#)

[AXI Basics 7 - Connecting to the PS using AXI4-Lite and Vitis HLS](#)

In Peripheral Details, specify the name as rtl_multiplier and choose an IP location.



3. In Add Interfaces, use the default 32-bit AXI4 Lite Slave interface.



4. On the last page, select **Edit IP** and click Finish. This will open another Vivado window in which we will implement the peripheral.
5. Copy lab1_starter/rtl_multiplier.v to ip_repo/src. Add ip_repo/src/rtl_multiplier.v as the design source to your Vivado project edit_rtl_multiplier_v1_0. Check "Copy sources into IP Directory". At this point the rtl_multiplier.v file is separately part of the Design Sources. Let's connect it to the AXI IP block.
6. Expand the top branch (rtl_multiplier_v1_0.v) and open the file rtl_multiplier_v1_0_S00_AXI_inst : rtl_multiplier_v1_0_S00_AXI (rtl_multiplier_v1_0_S00_AXI.v) by double-clicking on it.
7. Scroll down to the end of the file where the comment "Add user logic here" is and insert the code below before "User logic ends". The code below instantiates the rtl_multiplier module inside the AXI IP block and connects the clock to the AXI clock, the inputs a and b to the 16 MSB and LSB of the first register (slv_reg0), and the output product to a created wire rtl_multiplier_out.

```

401 // wire to hold rtl_multiplier output
402 wire [31:0] rtl_multiplier_out;
403 // instantiate the rtl_multiplier
404 rtl_multiplier rtl_mult_instance_01(
405     .clk(S_AXI_ACLK),

```

```

406         .a(slv_reg0[31:16]),
407         .b(slv_reg0[15:0]),
408         .product(rtl_multiplier_out)
409     );

```

8. Find the code block that implements memory mapped register select and read logic generation. Modify `reg_data_out` assignment from `slv_reg1` to `rtl_multiplier_out` in case that `axi_araddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB]` is `2'h1`:

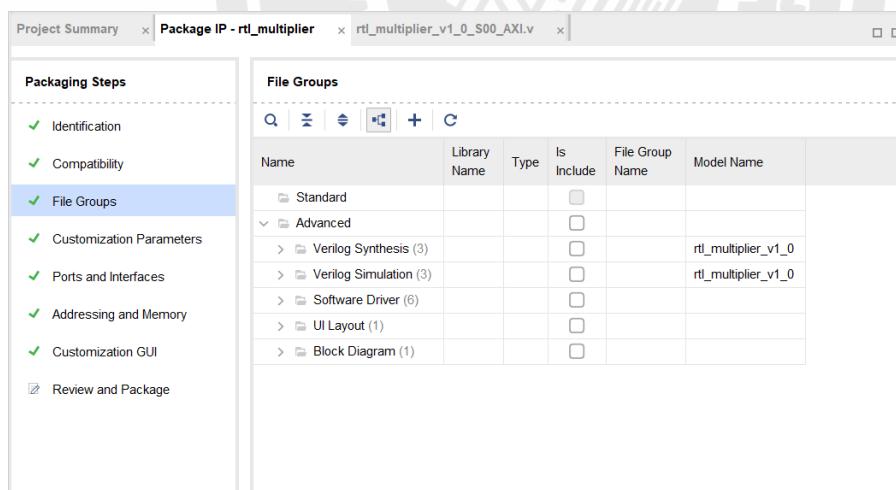
```

365     // Implement memory mapped register select and read logic
366     // Slave register read enable is asserted when valid address is
367     // and the slave is ready to accept the read address.
368     assign slv_reg_rden = axi_arready & S_AXI_ARVALID & ~axi_rvalid;
369     always @(*)
370     begin
371         // Address decoding for reading registers
372         case ( axi_araddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
373             2'h0 : reg_data_out <= slv_reg0;
374             2'h1 : reg_data_out <= rtl_multiplier_out;
375             2'h2 : reg_data_out <= slv_reg2;
376             2'h3 : reg_data_out <= slv_reg3;
377             default : reg_data_out <= 0;
378         endcase
379     end

```

After saving the file, you will notice that the `rtl_multiplier.v` file has been integrated under the AXI file in the hierarchy. Good work so far! Almost there - just a few more rudimentary steps.

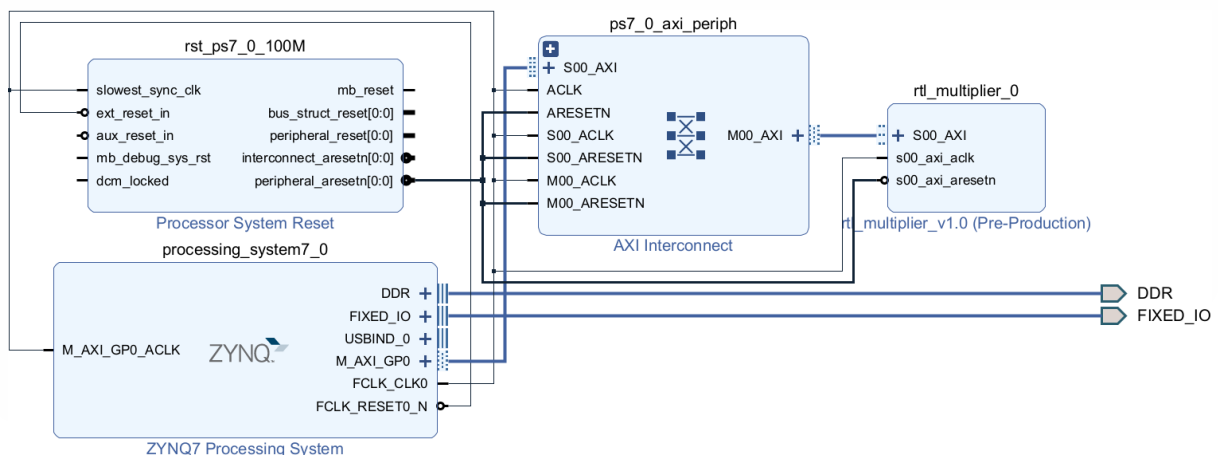
9. In the Package IP - `rtl_multiplier` tab, in "File Groups" of Packaging Steps, click "Merge changes from the File Groups Wizard". This will OK the File Groups step with a flattering green tick badge of great success. :)



- Go to "Review and Package" in Packaging Steps; an info banner will show "IP has been modified". Click "Re-Package IP". The pop-up window "Close Project" shows "Finished packaging 'rtl_multiplier_v1_0' successfully" and you can safely click "Yes" to close the project.

4.2 Add the IP block to the Zynq PS System

- In the original Vivado project containing the Zynq PS system, Flow Navigator->PROJECT MANAGER->IP INTEGRATOR->Open Block Design.
- As in the previous lab, to browse for an IP block, click on the Add IP (+) button and search for our new custom created rtl_multiplier_v1.0. Double-click it to add it to the design.
- Now let Vivado do the "magic" of connecting it to the ZYNQ7 PS by clicking on Run Connection Automation and use the default settings in the new dialog window. The Connection Automation will add a few necessary intermediate IP blocks. It might look scary at first, but fear not! It's just a Processor System Reset and an AXI Interconnect here and there, no biggie. To make it more clear (and hopefully less scary), click on the Regenerate Layout (which looks like a Refresh) button. You should see a "neat" design as shown below.



- This is all! Now, as in the previous tutorial in Section 2, save all files and click on Generate Bitstream. This will run through Synthesis and Implementation and will generate the bitstream file. Compilation takes a while - see the status in the upper-right corner. When completed, select Open Implemented Design. Next, export the hardware design including the bitstream as `Zynq_CPU_wrapper_hw_platform_3.xsa`, and then launch Xilinx Vitis. In the Implementation reports, check the resource utilization and take a screenshot, and analyze if it is reasonable.

```
1 start_gui
2 open_project
  ↪ F:/Documents/GitHub/ECE4810J_FA2022_TA/Lab1/lab1_1/lab1_1.xpr
3 update_compile_order -fileset sources_1
```

```
4 create_peripheral xilinx.com user rtl_multiplier 1.0 -dir
  ↳ F:/Documents/GitHub/ECE4810J_FA2022_TA/ip_repo
5 add_peripheral_interface S00_AXI -interface_mode slave -axi_type lite
  ↳ [ipx::find_open_core xilinx.com:user:rtl_multiplier:1.0]
6 generate_peripheral -driver -bfm_example_design -debug_hw_example_design
  ↳ [ipx::find_open_core xilinx.com:user:rtl_multiplier:1.0]
7 write_peripheral [ipx::find_open_core xilinx.com:user:rtl_multiplier:1.0]
8 set_property ip_repo_paths
  ↳ F:/Documents/GitHub/ECE4810J_FA2022_TA/ip_repo/rtl_multiplier_1.0
  ↳ [current_project]
9 update_ip_catalog -rebuild
10 ipx::edit_ip_in_project -upgrade true -name edit_rtl_multiplier_v1_0
  ↳ -directory F:/Documents/GitHub/ECE4810J_FA2022_TA/ip_repo f:/Documen
  ↳ ts/GitHub/ECE4810J_FA2022_TA/ip_repo/rtl_multiplier_1.0/component.xml
11 update_compile_order -fileset sources_1
12 add_files -norecurse -copy_to f:/Documents/GitHub/ECE4810J_FA2022_TA/ip_
  ↳ repo/rtl_multiplier_1.0/src
  ↳ F:/Documents/GitHub/ECE4810J_FA2022_TA/ip_repo/src/rtl_multiplier.v
13 update_compile_order -fileset sources_1
14 ipx::merge_project_changes files [ipx::current_core]
15 set_property core_revision 2 [ipx::current_core]
16 ipx::update_source_project_archive -component [ipx::current_core]
17 ipx::create_xgui_files [ipx::current_core]
18 ipx::update_checksums [ipx::current_core]
19 ipx::check_integrity [ipx::current_core]
20 ipx::save_core [ipx::current_core]
21 ipx::move_temp_component_back -component [ipx::current_core]
22 close_project -delete
23 update_ip_catalog -rebuild -repo_path
  ↳ f:/Documents/GitHub/ECE4810J_FA2022_TA/ip_repo/rtl_multiplier_1.0
24 open_bd_design {F:/Documents/GitHub/ECE4810J_FA2022_TA/Lab1/lab1_1/lab1_
  ↳ 1.srcs/sources_1/bd/Zynq_CPU/Zynq_CPU.bd}
25 startgroup
26 create_bd_cell -type ip -vlnv xilinx.com:user:rtl_multiplier:1.0
  ↳ rtl_multiplier_0
27 endgroup
28 apply_bd_automation -rule xilinx.com:bd_rule:axi4 -config { Clk_master
  ↳ {/processing_system7_0/FCLK_CLK0 (100 MHz)} Clk_slave {Auto}
  ↳ Clk_xbar {Auto} Master {/processing_system7_0/M_AXI_GPO} Slave
  ↳ {/rtl_multiplier_0/S00_AXI} ddr_seg {Auto} intc_ip {New AXI
  ↳ Interconnect} master_apm {0}} [get_bd_intf_pins
  ↳ rtl_multiplier_0/S00_AXI]
29 regenerate_bd_layout
30 save_bd_design
31 reset_run synth_1
32 launch_runs impl_1 -to_step write_bitstream -jobs 12
```



```
33 open_run impl_1
34 write_hw_platform -fixed -include_bit -force -file F:/Documents/GitHub/EJ
   → CE4810J_FA2022_TA/Lab1/lab1_1/Zynq_CPU_wrapper_hw_platform_3.xsa
```

4.3 Interfacing with the IP Block in software

So far, we have the Zynq PS and the RTL Multiplier as an AXI Lite slave IP block all nicely interconnected, but it is all useless unless we use it in an application - so let's do that.

1. Create a new Vitis Application project based on Zynq_CPU_wrapper_hw_platform_3.xsa named hello_rtl_multiplier from "Hello World" template. Replace helloworld.c by lab1_starter/lab1-3.c in src.
2. Select hello_rtl_multiplier_system and Build Project. The build may probably fail; please see Appendix: [B.4](#).
3. After the finished build, run. Observe the output in Vitis Serial Terminal and take a screenshot.

4.4 Follow-up assignment

Implement an IP block as an AXI Lite Slave that takes in two numbers as arguments and returns the div and mod of them.



5 (Optional) Create an FIR filter by MATLAB HDL Coder

5.1 Direct FIR filter and symmetric FIR filter

Finite Impulse Response filters, usually referred to as FIR filters, are a kind of important filter that is widely used in digital systems. FIR indicates the impulse response will become 0 after a finite duration; thus, they are easy to implement compared to another type of filter, infinite impulse response (IIR) filters. The mathematical definition of FIR filters can be regarded as simply a convolution of a discrete input sequence and a finite length impulse response sequence:

$$y(n) = h(n) * x(n) = \sum_{k=0}^{N-1} h(k)x(n-k) \quad (1)$$

The equation above reveals that the computation of an FIR filter is a difference equation. This difference equation can be implemented by several essential components: adders, multipliers, and delay elements. Therefore, the FIR filter can be implemented using these simple blocks, and the critical problem of FIR filtering design is how to balance resource usage and accuracy. By properly rearranging the structure, the resource usage or delay path can be reduced without degrading the accuracy.

The length of the impulse response, N , is often called the order or the tap of a filter. The order of the FIR filter determines its accuracy. If we increase the order N , the filter will become sharp in its transition band but consumes more hardware usage since it increases the number of computations and delays in the system. On the contrary, if decreasing N , the hardware utilization is reduced, but the accuracy is also degraded. Therefore, the critical question is to choose the appropriate tap or order number of a filter, to balance the trade-off between hardware resource usage and accuracy. By carrying on some research in the Filter Designer of MATLAB's Signal Processing Toolbox, $N = 8$ can basically realize a satisfactory performance in the transition band and stop band.

There are two basic types of FIR filters: direct FIR filter (DFIR) and symmetric FIR (SFIR) filter. The direct form structure takes one input signal per clock cycle and saves them into pipeline registers. The data in these registers will do multiplication with coefficient stored in multiplier, shown in the diagram as $h[i]$. Then the data result after multiplication will add together using an add as the filtered signal output.

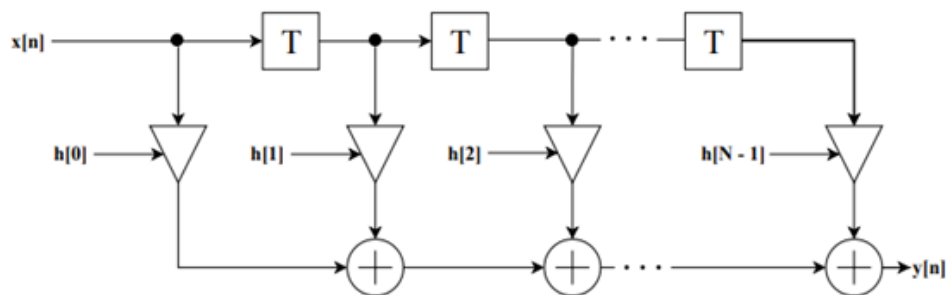


Figure 1: Diagram for the direct form structure.

In the symmetry form structure, one input signal per clock cycle is taken and saved into pipeline registers. The data in these registers will do multiplication with the coefficient stored in the multiplier. Unlike the previous direct form one, the first pipeline register and the last pipeline register use one multiplier together. Therefore, in this structure, we can save 4 multipliers.

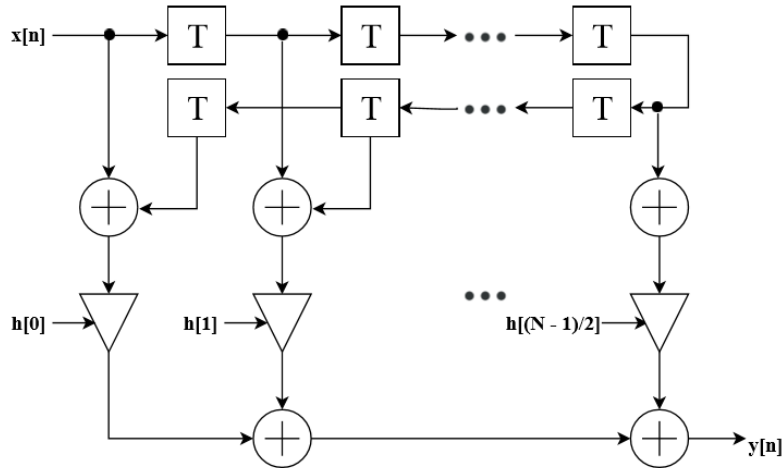


Figure 2: Diagram for the symmetric form structure.

5.2 System Design with HDL Code Generation from MATLAB and Simulink

Now, we are going to get started with MATLAB HDL Coder. Firstly, switch the working directory of MATLAB to the folder where you store the four starter MATLAB scripts. Note that if your Xilinx Vivado needs to run as administrator, you should also run MATLAB as administrator. Run

```
mlhdlc_fir_tb
```

The input/output plot and the power spectral density (PSD) plot are:

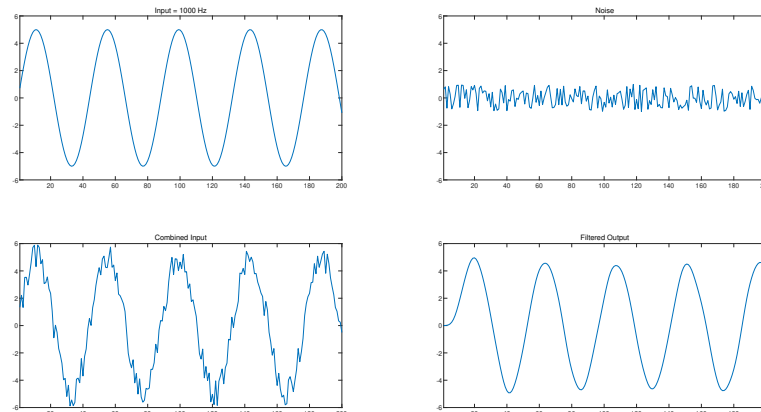


Figure 3: The input/output plot of the FIR filter.

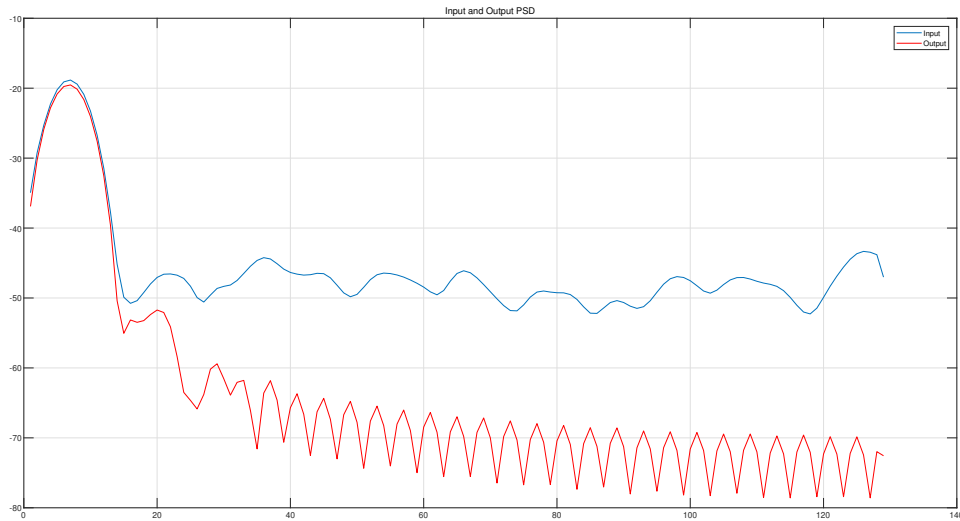


Figure 4: The power spectral density (PSD) plot of the FIR filter.

Run

```
hdlsetuptoolpath('ToolName','Xilinx
↳ Vivado','ToolPath','F:\Xilinx\Vivado\2022.1\bin\vivado.bat')
```

to prepend the Xilinx Vivado path to the system path. Run

```
coder -hdlcoder -new fir_project
```

In the HDL Code Generation tab on the right, add MATLAB function `mlhdlc_fir.m` and add `mlhdlc_fir_tb.m` as MATLAB Test Bench. Click "Autodefine types". In the pop-up dialog "Autodefine Input Types", select `mlhdlc_fir_tb.m` as the Test file, and run. Open the Workflow Advisor, and you can see "Define Input Types" is ticked. The `indatabuf` is defined as a `double(1 × 1)`. Click "Use These Types". Right-click "Fixed-Point Conversion" and run this task, or click DATA COLLECTION->Analyze. The Build Log is

Code generation successful.

The Simulation Output is

```
### Analyzing the design 'mlhdlc_fir'
### Analyzing the test bench(es) 'mlhdlc_fir_tb'
### Begin Floating Point Simulation (Instrumented)
### Floating Point Simulation Completed in 9.8445 sec(s)
### Elapsed Time:          11.2084 sec(s)
```

The Type Validation Output is

```
### Generating Type Proposal Report for 'mlhdlc_fir' mlhdlc_fir_report.html
### Generating Fixed Point MATLAB Code mlhdlc_fir_fixpt using Proposed Types
### Generating Fixed Point MATLAB Design Wrapper mlhdlc_fir_wrapper_fixpt
```

```
### Generating Mex file for ' mlhdlc_fir_wrapper_fixpt '
Code generation successful: View report
```

The Variables tab is:

Variables	Function Replacements	Output							
Variable	Type	Sim Min	Sim Max	Whole Numb...	Proposed Type	Log Data	Max Diff		
<input type="checkbox"/> Input									
indatabuf	double	-5.88	5.89	否	numerictype(1, 14, 10)				
<input type="checkbox"/> Output									
outdatabuf	double	-4.92	4.96	否	numerictype(1, 14, 10)				
<input type="checkbox"/> Persistent									
tap_delay	1 x 16 double	-5.88	5.89	否	numerictype(1, 14, 10)				
<input type="checkbox"/> Local									
coeff	1 x 16 double	0	0.14	否	numerictype(0, 14, 16)				

Then, go to "Select Code Generation Target", and set the target device and synthesis tool. Choose workflow as Generic ASIC/FPGA. If the Synthesis tool is "No synthesis tool available on system path", then click Refresh list as long as you run the `hdlsetuptoolpath` command previously. Then, you should see "No synthesis tool specified". Select Xilinx Vivado. Select Zynq as Chip family, xc7z020 as the device, and clg400 as the package.

In "HDL Code Generation", generate synthesizable HDL code from the fixed-point MATLAB code. In the Target tab, for Target Selection, select Verilog. Check "Check HDL conformance", "Generate HDL", and "Generate EDA scripts". In the Coding Style tab, for Generated Code Comments, check "Preserve MATLAB code comments" and "Generate report". In the Optimizations tab, specify 1 for Input pipelining and Output pipelining, and then select Distribute pipeline registers. In the Advanced tab, check Generate MATLAB Function Block (Simulation license is required) from Simulation Integration. Click Run to generate Verilog code. Look through HDL Code Generation Check Report for 'mlhdlc_fir_fixpt'.

```
### Begin Verilog Code Generation
### Working on mlhdlc_fir_fixpt as mlhdlc_fir_fixpt.v.
### The DUT requires an initial pipeline setup latency. Each output port
  ↳ experiences these additional delays.
### Output port 1: 2 cycles.
### Output port 2: 2 cycles.
### Generating Resource Utilization Report resource_report.html.
  ### Generating HDL Conformance Report
    ↳ mlhdlc_fir_fixpt_hdl_conformance_report.html.
### HDL Conformance check complete with 0 errors, 1 warnings, and 0
  ↳ messages.
### Code generation successful: View report
### Elapsed Time: '      20.4480' sec(s)
```

The summary of the HDL resource utilization report is

Multiplier	16
Adders/Subtractors	31
Registers	19
Total 1 Bit Registers	253
RAMs	0
Multiplexers	0
I/O Bits	32
Shifters	0

Table 2: The summary of HDL resource utilization report.

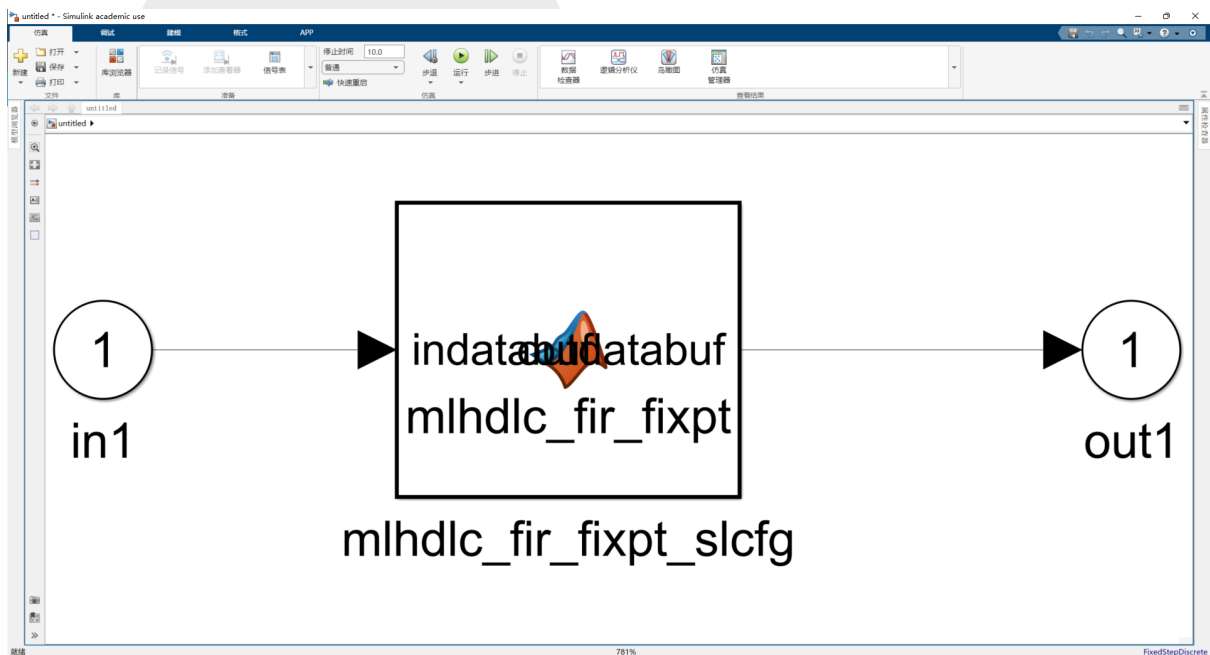


Figure 5: The MATLAB Function Block of FIR filter.

5.3 Basic HDL Code Generation and FPGA Synthesis from MATLAB

Now, your first task is to use `mlhdlc_sfir.m` and `mlhdlc_sfir_tb.m` to finish the full workflow. Run

```
coder -hdlcoder -new sfir_project
```

Finish all steps mentioned before and save the `mlhdlc_sfir_tb_plot`. Then in HDL Verification->Verify with HDL Test Bench, verify the HDL code with test vectors from the test bench using the specified simulation tool. In the Output Settings tab, from HDL Test Bench Generation Settings, check "Generate HDL test bench". If you have Mentor ModelSim installed, from HDL Test Bench Simulation Settings, check Simulate generated HDL test bench and select ModelSim as the Simulation tool. You can check the test bench data files and the compilation report `mlhdlc_sfir_fixpt_vsim_log_compile.txt`, and the simulation report `mlhdlc_sfir_fixpt_vsim_log_sim.txt`.

You can also generate and run a cosimulation test bench in Verify with Cosimulation. In the Output Settings tab, from Cosimulation Test Bench Generation Settings, check "Generate cosimulation test bench". If you have Mentor ModelSim installed, you can check Log outputs for comparison plots, choose Cosimulation for use with Mentor Graphics ModelSim, and select "GUI" for HDL simulator run mode in cosimulation. You may save the plot of `y_out` difference and the plot of `delayed_xout` difference, and take a screenshot of the waveform of all the In and Out signals in ModelSim. You can also check the Simulation generated cosimulation test bench from Cosimulation Test Bench Simulation Settings. The "cosim" directory is under `mlhdlc_sfir/codegen/mlhdlc_sfir`.

Unfortunately, Arty boards are not supported by MATLAB HDL Coder (including MathWorks HDL Coder Team's HDL Coder Support Package for Xilinx FPGA Boards and MathWorks HDLVerifier Team's HDL Verifier Support for Xilinx FPGA Boards) yet, so we have to skip "Verify with FPGA-in-the-Loop" that generates and runs FPGA-in-the-Loop test bench.

You can now directly do Synthesis and Analysis here by Xilinx Vivado. Create a synthesis project for the specified synthesis tool. You can see "Synthesis project creation successful". Then, go to "Run Synthesis" and run to synthesize and map the generated HDL code. You can check the synthesis report `mlhdlc_sfir_fixpt_syn_results.txt` and the parsed resource report file `mlhdlc_sfir_fixpt_utilization_synth.rpt`. Finally, go to "Run Implementation" and run to place and route (PAR) the generated HDL code for the specified hardware. You can check the parsed resource report file `mlhdlc_sfir_fixpt_utilization_placed.rpt`, and the parsed timing report file `timing_post_route.rpt`.

5.4 Generate HDL Code from Simulink Model

In this subsection, you are going to reproduce the result of the article "Design and Comparison of FIR Filter Based on DSP Builder and HDL Coder" in *2020 IEEE 15th International Conference on Solid-State & Integrated Circuit Technology (ICSICT)* [3]. Do not be panic. This article is so simple and low-quality that you may question how it could be selected for the conference.

Before working, you should be familiar with some concepts. The number of **taps** of a filter = the filter length. The **order** of a filter is the number of non-zero taps of a filter. In the article, the number of taps is 16, but the constant term of a general 16-tap FIR filter is 0, so the order of the filter is 15 [5]. The diagram for a common T-tap FIR filter is

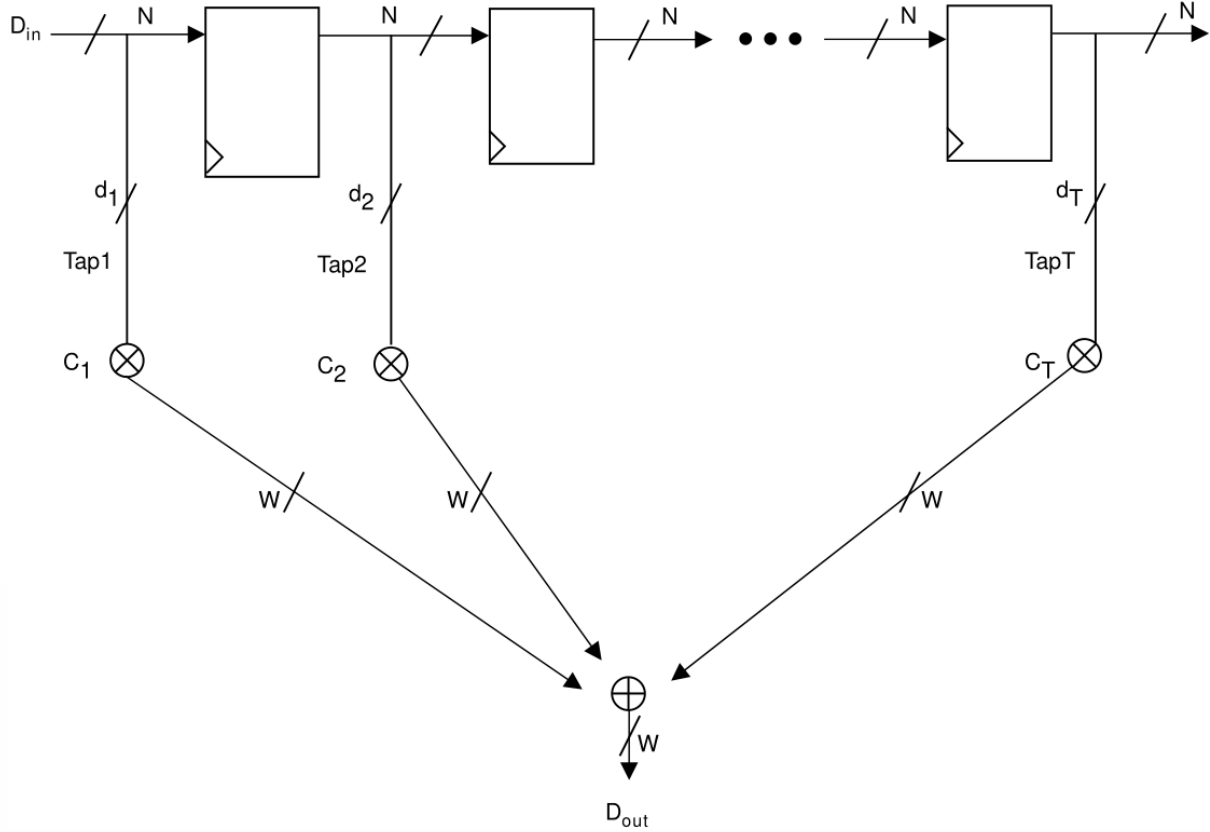


Figure 6: The diagram for a common T-tap FIR filter [1].

Firstly, open the filter designer of MATLAB from the APP tab or run

`filterDesigner`

from the command line. Choose the direct I-type filter structure. Choose low-pass for Response type, FIR - Window for Design Method; specify 15 for Filter Order. In Options, choose Kaiser window with Beta = 0.5. In the Frequency Specifications frame, set Units = MHz, sampling frequency $F_s = 12.5$, cutoff frequency $F_c = 2$ [17].

Kaiser window or Kaiser-Bessel window and its Fourier transform is given by

$$w_0(x) \triangleq \begin{cases} \frac{1}{L} \frac{I_0[\pi\alpha\sqrt{1-(2x/L)^2}]}{I_0[\pi\alpha]}, & |x| \leq L/2 \\ 0, & |x| > L/2 \end{cases} \xleftrightarrow{\mathcal{F}} \frac{\sin\left(\sqrt{(\pi L f)^2 - (\pi\alpha)^2}\right)}{I_0(\pi\alpha) \cdot \sqrt{(\pi L f)^2 - (\pi\alpha)^2}} \quad (2)$$

For digital signal processing, the function can be sampled symmetrically as:

$$w[n] = L \cdot w_0\left(\frac{L}{N}(n - N/2)\right) = \frac{I_0\left[\pi\alpha\sqrt{1 - \left(\frac{2n}{N} - 1\right)^2}\right]}{I_0[\pi\alpha]}, \quad 0 \leq n \leq N \quad (3)$$

The filter configuration is:

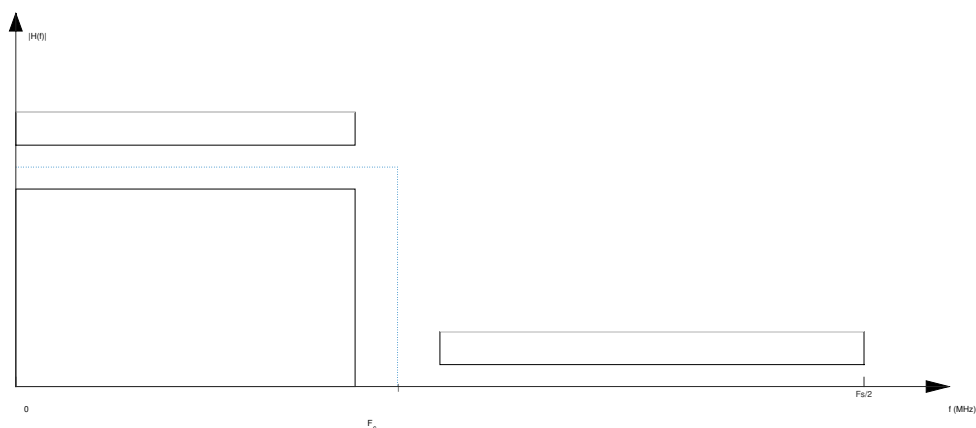


Figure 7: The filter configuration.

The visualization of the Kaiser window is

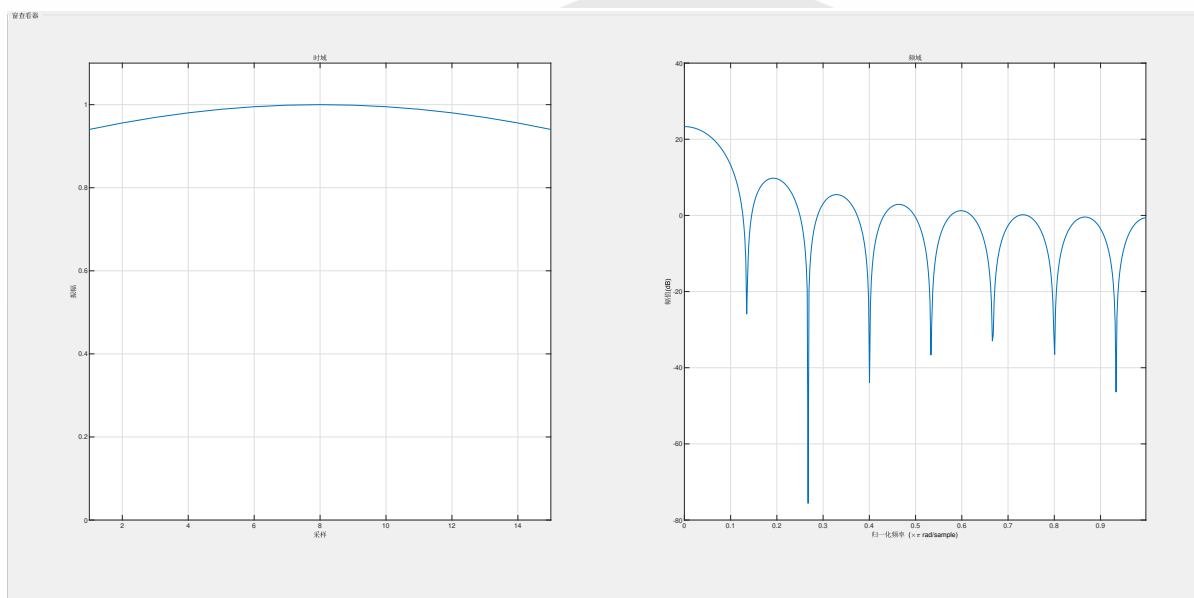


Figure 8: The visualization of the Kaiser window.

The frequency and phase response is

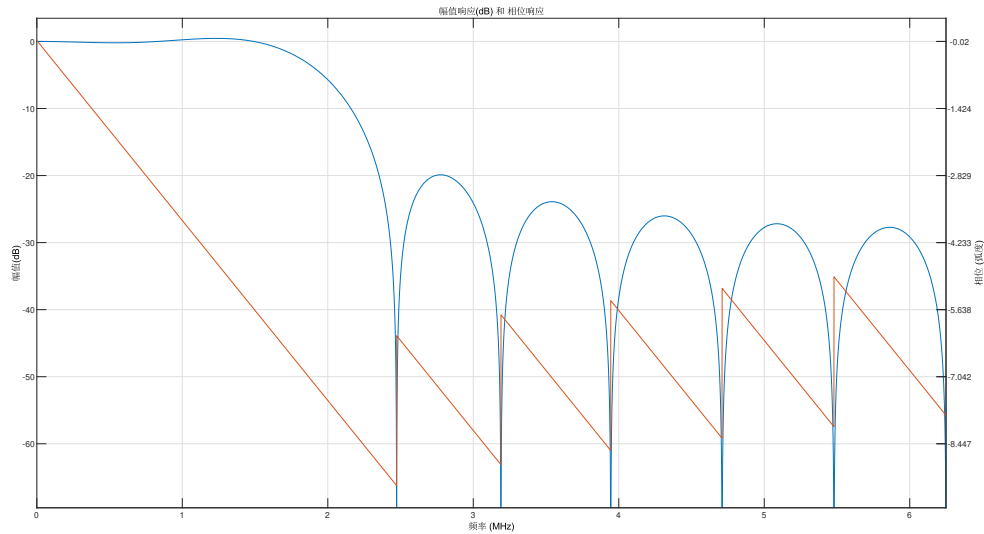


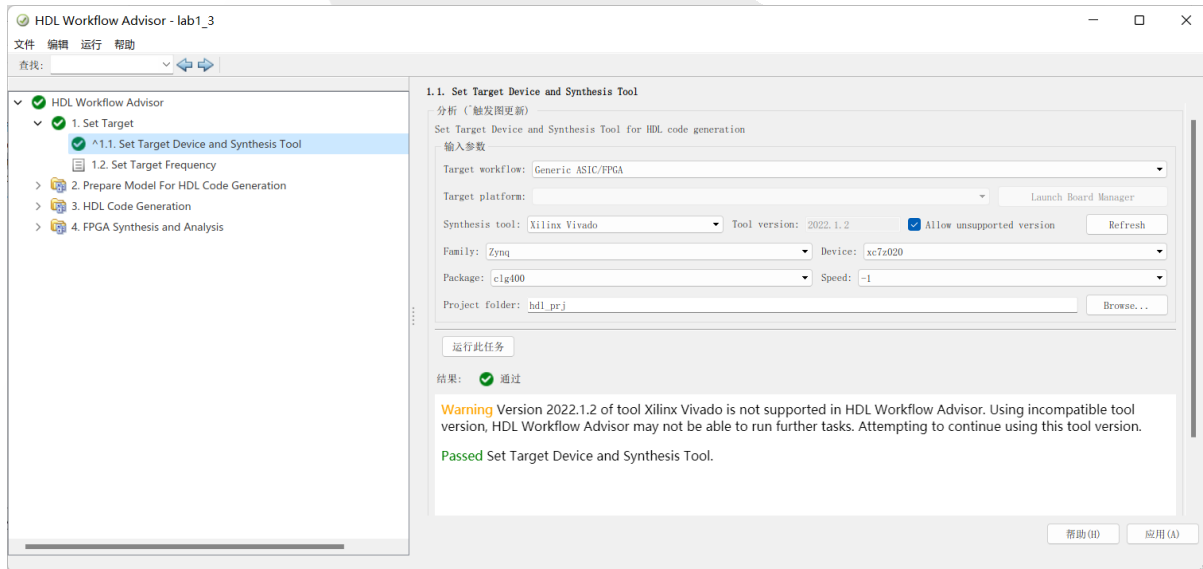
Figure 9: The frequency and phase response.

You can derive the filter coefficients from the filter designer. Note that the filter coefficients are symmetric.

Open Simulink. From HDL Coder, create a Blank DUT SLX. Feel free to refer to [16]. Remember that only add components from the HDL Coder category in the library browser. To inspect the waveform, you can add a scope at the endpoint, but to generate HDL code, you must also add an Out1 from the Sinks category at the endpoint. Since the filter coefficients are symmetric, you can share one coefficient between a pair of delays. For x_{in} , you can either use Sine Waves as the article uses or use the data From Workspace of a trigonometric function as Sources as MATLAB Documentation uses; for h_{in} , use your previously generated filter coefficients, and you can serve them as Constant inputs or as the Gain by multiplication. If using Sine Waves, set Frequency (Hz) and Sample time to proper values. You can establish a model hierarchy that encapsulates your FIR filter into a DUT. Also, carefully set the (output) data types. If you set double, the simulation or code generation will fail because Sine Wave should be computed by Table lookup, and no floating point IP library is to be used, so you have to set by `fixdt`. *Hint: you only need to specify the output data type of source modules explicitly, then Simulink will automatically determine the data type as Inherit: auto and the output data type as Inherit: Inherit via internal rule.* There are various valid data type configurations. Data Type Conversion module is allowed but is not necessary. To make a tidy output waveform, set the proper stop time of the simulation.

After the Simulink model is finished, in the App tab, open HDL Coder, then you can open the Workflow Advisor in the ASSISTANCE frame in the HDL CODE tab. This HDL Workflow Advisor is slightly simpler than what we used before. Set the target device, the synthesis tool, and the target frequency. Unfortunately, Xilinx Vivado Tool version 2022.1.2 is not supported yet, so you need to check "Allow unsupported version". Ignore the warning information:

Warning Version 2022.1.2 of tool Xilinx Vivado is not supported in HDL
 ↳ Workflow Advisor. Using incompatible tool version, HDL Workflow
 ↳ Advisor may not be able to run further tasks. Attempting to continue
 ↳ using this tool version.



In Step 3: HDL Generation->3.1 Set HDL Options, click HDL Code Generation Settings... In the Set Basic Options frame, select Verilog as Language. In Floating Point->Floating Point IP Library, choose None. In Report->Optimization Reports, choose Generate resource utilization report and Generate optimization report. In Step 3.2: Generate RTL Code and Testbench, check all three input arguments. Run. You may need to modify your design if the run fails or the generated Verilog module is empty.

To run the simulation, you can right-click your Scope and press "P" to open the Scope window. Click the green button on the top of the Scope window or the Simulink window to run. In View->Configuration properties(C)... you can set 2 input ports; set automatic Coordinate scaling; in the Record tab, uncheck the first option to prevent limiting the data points to last. Enable legends. You can split the view of two wires into two views in View->Layout(L)... After the simulation is finished, compare the two views and take a screenshot. Your waveform does not need to be completely identical to the article's waveform, but you should write some sentences to explain your result in your report.

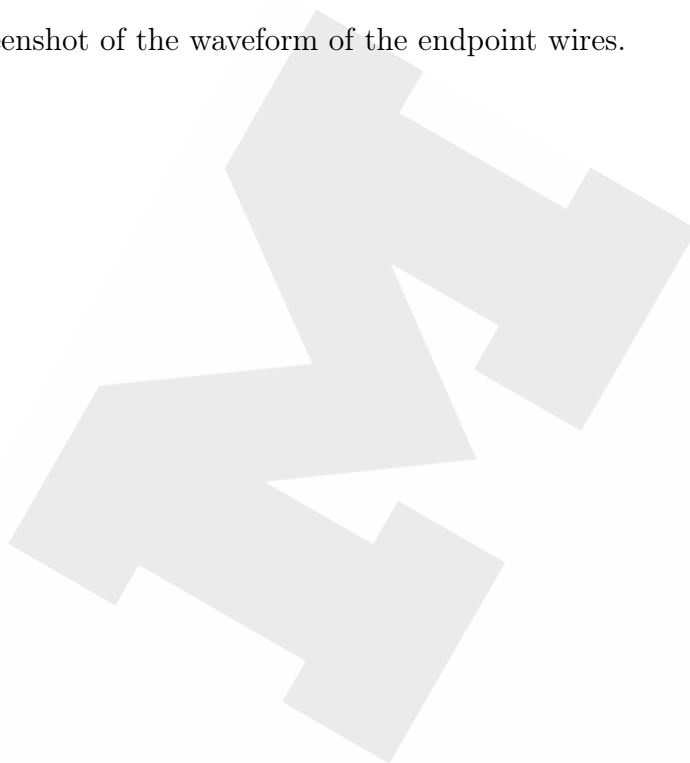
Run Step 4: FPGA Synthesis and Analysis. Ignore the warning:

Warning Downstream Integration: Current version 2022.1.2 of Xilinx Vivado
 ↳ tool on system path may not be compatible with HDL Workflow Advisor.
 ↳ The compatible version: Xilinx Vivado 2020.2. Attempt to continue
 ↳ running Xilinx Vivado ...

Take a screenshot of the Resource summary and the Timing summary of the implementation.

(Optional) If you have ModelSim installed, simulate the output Verilog HDL code. Take

a screenshot of the waveform of the endpoint wires.



6 Tcl exercises

6.1 Average

Write a procedure (proc) called `avg` that takes an array of numbers with any length and returns the average of it.

Example:

```
% puts [avg {70 80 50 60}]  
65
```

6.2 Factorial

Write a procedure (proc) that does the factorial function.

Example:

```
% puts [factorial 3]  
6
```

6.3 Two-fer

Given a file name, return a string with messages. If given name, return "C source: main.c"; otherwise, return "C source:" and the given name.

Requirement:

Not allowed to use the keyword `args`. The body of the procedure should be one line.

Example:

```
% two-fer  
C source: main.c  
% two-fer lab1-1.c  
C source: lab1-1.c
```

7 Post-lab questions

Write short answers to the following questions:

1. What is a bare metal test?
2. How does the Zynq PS communicate with the IP blocks we created in this lab?
3. Summarize the main features of AXI from the reading material

8 Deliverables

Group Deliverables (Compile everything as a single pdf report besides the code):

- Section 2: the screenshot of the Vitis IDE window in the build finished state and the screenshot of Vitis Serial Terminal
- Section 3: the screenshot of the Implemented Design, including the Design Timing Summary of the Xilinx Vivado project, the screenshot of the Vitis IDE window in the build finished state, four photos of the board, and the code
- Section 4: the resource utilization report/screenshot, the screenshot of the Vitis IDE window in the build finished state and the screenshot of Vitis Serial Terminal for both rtl_multiplier and rtl_divider
- Section 5.3 (optional): the mlhdlc_sfir_tb_plot of the SFIR filter, the HDL resource utilization report of the SFIR filter, the MATLAB Function Block of the SFIR filter, the screenshot of the implementation design including the Design Timing Summary of the Vivado project mlhdlc_sfir_fixpt_vivado.xpr; if applicable (not counted for grade), the plot of y_out difference, the plot of delayed_xout difference, mlhdlc_sfir_fixpt_vsim_log_compile.txt, mlhdlc_sfir_fixpt_vsim_log_sim.txt, and the screenshot of the waveform of all the In and Out signals in ModelSim
- Section 5.4 (optional): the Simulink model lab1_3.slx file, your generated Verilog HDL code from Simulink, the screenshot of the Resource summary and the Timing summary of the implementation from Simulink
- Section 6: Tcl code
- Section 7: answer

Remember to add some comments or explanations on all the deliverables of Sections 2 - 5 in your report.

9 Grading policy

Factors	Percentage
Section 2	12%
Section 3	30%
Section 4	40%
Section 5	20% (bonus)
Section 6	9%
Section 7	9%

10 Acknowledgment

We would like to acknowledge previous TA Mr. Yihua Liu (FA 2022) for helping develop this lab and laying solid foundation for this course. Special thanks to the AMD-Xilinx University Program for providing the starters' files and ideas for this lab.

A Peer Evaluation Form

Each team member is required to provide a peer evaluation of the team effort of the lab. The score of the peer evaluation should be integers ranging between 0 to 5, inclusively, with 5 indicating the biggest contribution. A score should be given to each team member including yourself according to the team member's contribution based on your observation. A brief description of the contribution of each team member should also be provided, as shown in the following table.

Part	Your work	Your partner's work	Your score	Your partner's score
Section 2				
Section 3				
Section 4				
Section 5				
Section 6				
Section 7				

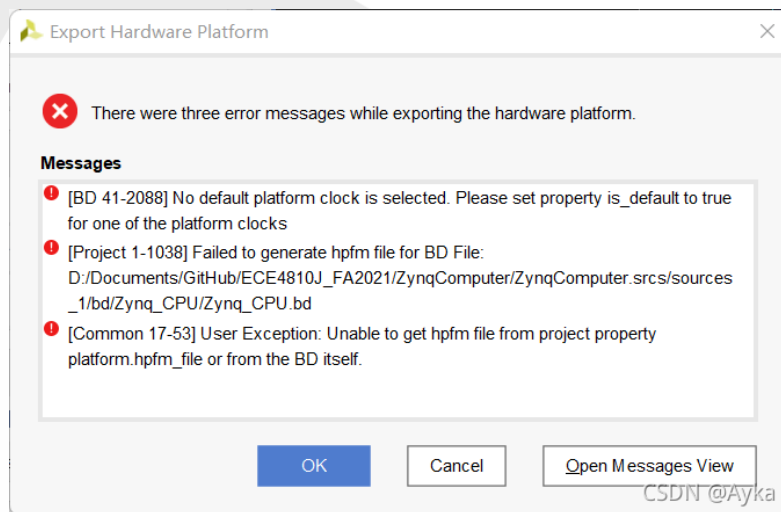
Note that the score you give is not decisive. Your peer evaluation score may be adjusted according to your work described based on certain fair policies so that as long as your partner and you assume a similar workload, both of you can get full scores, but if the workload is very different though you and your partner give the identical scores, the scores will be adjusted. Your final lab grade may be scaled by the individual difference (the individual average divided by the group average). You cannot submit a peer evaluation form with the same contents as your partner's; otherwise, 3 points of deduction will be applied to your total grade for this lab.

B Troubleshooting

B.1 Vivado Export Hardware

If you see error messages like

1. [BD 41-2088] No default platform clock is selected. Please set property is_default to true for one of the platform clocks
2. [Project 1-1038] Failed to generate hpfm file for BD file:
F:/Documents/GitHub/ECE4810J_FA2022_TA/Lab1/lab1_1/lab1_1.srcs/
source_1/bd/Zynq_CPU/Zynq_CPU.bd
3. [Common 17-53] User Exception: Unable to get hpfm file from project property platform.hpfm_file or from the BD itself



Please see the reference article [12].

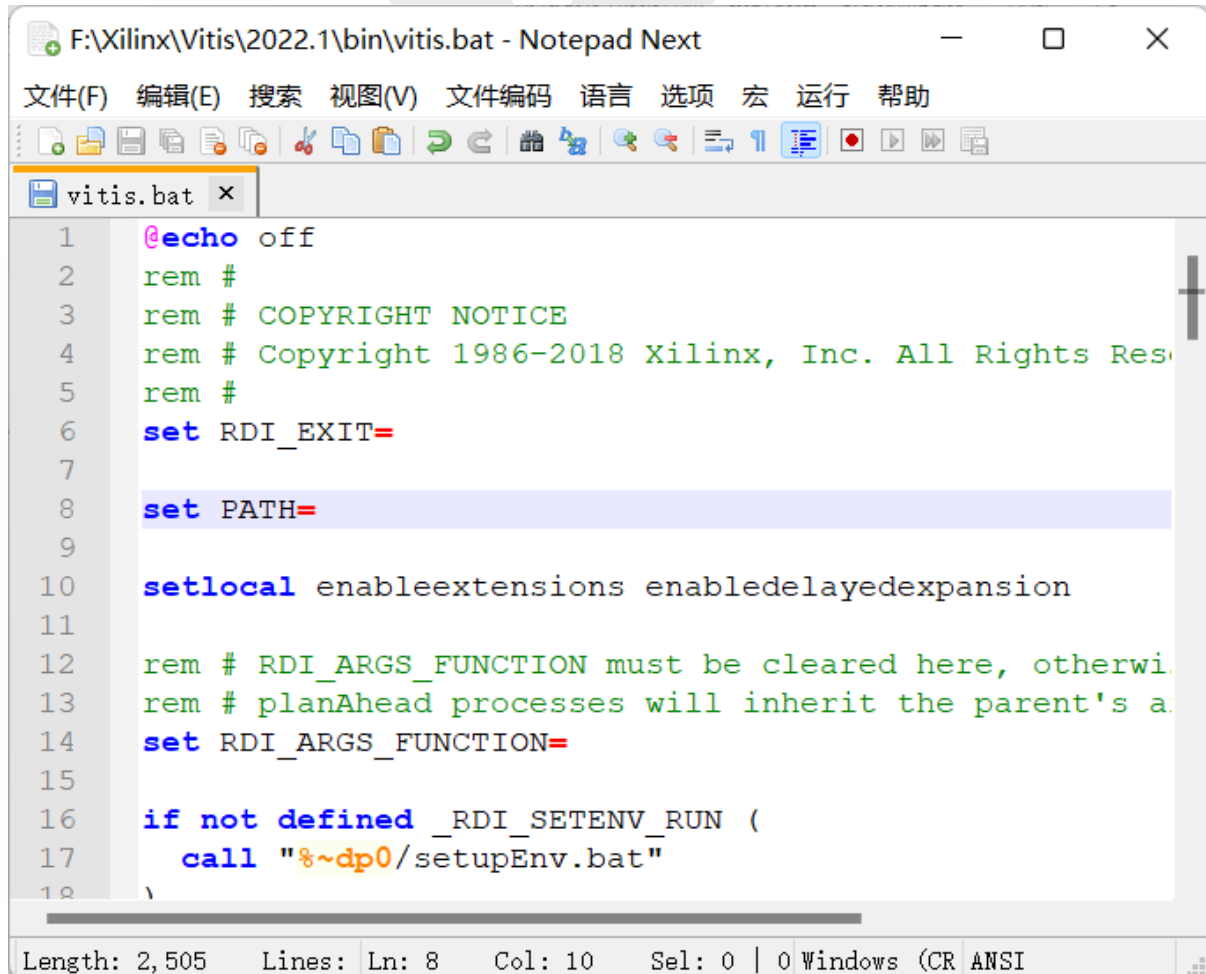
B.2 Launch Xilinx Vitis

If Vitis fails to start up or does not respond at start-up, close Vivado and manually launch Vitis from your desktop or Start Menu [14]. If you are prompted to install Microsoft Visual C++ 2015-2019 Redistributable (x64) - 14.29.30133, you will fail [6].



You can just run Xilinx Vitis as administrator. If the menu of shortcut of Vitis has no "Run as Administrator", your shortcut probably points to F:\Xilinx\Vitis\2022.1\bin\vitis.vbs file under your Vitis installation directory. There are some tricks you can find on Google about running VBS as administrator, but you would better run F:\Xilinx\Vitis\2022.1\bin\vitis.bat as administrator.

If Vitis still does not respond at start-up, use a text editor to edit vitis.bat file [11],



```

F:\Xilinx\Vitis\2022.1\bin\vitis.bat - Notepad Next
文件(F) 编辑(E) 搜索 视图(V) 文件编码 语言 选项 宏 运行 帮助
vitis.bat x
1  @echo off
2  rem #
3  rem # COPYRIGHT NOTICE
4  rem # Copyright 1986-2018 Xilinx, Inc. All Rights Reserved
5  rem #
6  set RDI_EXIT=
7
8  set PATH=
9
10 setlocal enableextensions enabledelayedexpansion
11
12 rem # RDI_ARGS_FUNCTION must be cleared here, otherwise
13 rem # planAhead processes will inherit the parent's a
14 set RDI_ARGS_FUNCTION=
15
16 if not defined _RDI_SETENV_RUN (
17     call "%~dp0/setupEnv.bat"
18 )
  
```

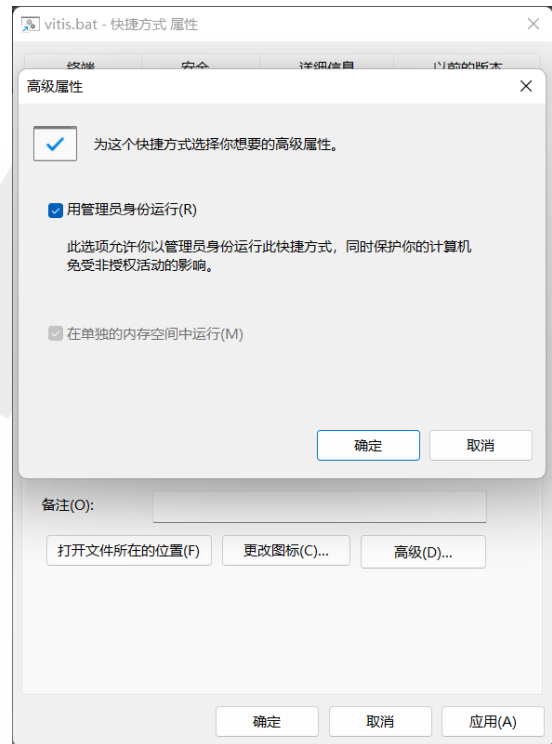
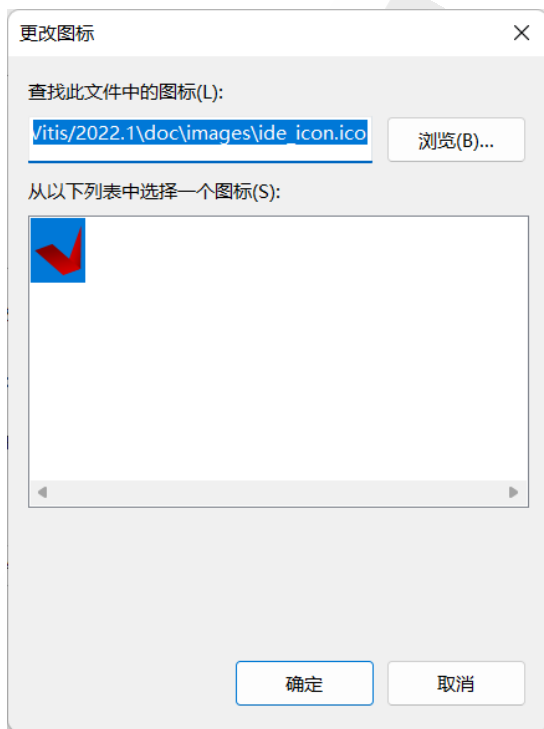
Length: 2,505 Lines: Ln: 8 Col: 10 Sel: 0 | 0 Windows (CR ANSI)

add

```
set PATH=
```

to avoid PATH overflow [4]. After this is done, re-run vitis.bat as administrator, then everything should work.

To make start-up easier, you can create a shortcut of vitis.bat, change the icon as F:\Xilinx\Vitis\2022.1\doc\images\ide_icon.ico; click Advance (D)..., check run as administrator:

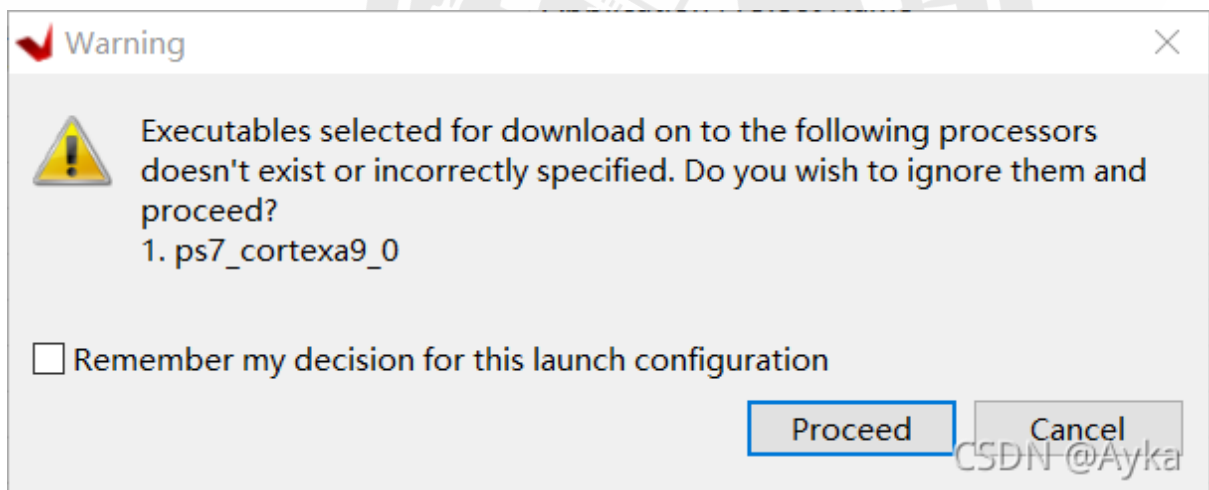


Change the name to "Xilinx Vitis 2022.1" and substitute the original shortcut with it.

B.3 Vitis Build

If you see a Warning window showing:

Executables selected for download on to the following processors doesn't
 → exist or incorrectly specified. Do you wish to ignore them and
 → proceed?
 1. ps7_cortexa9_0



It means that you forgot to build the project before you ran it. Click "Build Project" in

the "Project" menu [10].

If you create the application project from the Empty Application(C) template for Section 4, you will not include platform.h and will get ten errors:

```
1 ../src/lab1-3.c:1:10: fatal error: platform.h: No such file or directory
2     1 | #include "platform.h"
3       |             \~~~~~
4 compilation terminated.
5 make[1]: *** [src/subdir.mk:23: src/lab1-3.o] Error 1
6 make: *** [Makefile:18: all] Error 2
7 make: *** [Makefile:30: zynq_fsbl_bsp/ps7_cortexa9_0/lib/libxil.a] Error
  → 2
8 make: *** [makefile:34: all] Error 2
9 make[1]: *** [Makefile:18: all] Error 2
10 make[1]: *** [Makefile:46:
  → ps7_cortexa9_0/libsrc/rtl_multiplier_v1_0/src/make.libs] Error 2
11 make[1]: *** [src/subdir.mk:23: src/lab1-3.o] Error 1
12 make[2]: *** [Makefile:18: libs] Error 1
13 make[2]: *** [Makefile:46:
  → ps7_cortexa9_0/libsrc/rtl_multiplier_v1_0/src/make.libs] Error 2
14 make[3]: *** [Makefile:18: libs] Error 1
```

If your process of the building failed with:

```
1 "Compiling rtl_multiplier..."
2
3 cc1.exe: fatal error: *.c: Invalid argument
4 compilation terminated.
5 make[2]: *** [Makefile:18: libs] Error 1
6 make[1]: *** [Makefile:46:
  → ps7_cortexa9_0/libsrc/rtl_multiplier_v1_0/src/make.libs] Error 2
7 make: *** [Makefile:18: all] Error 2
8 make: Leaving directory 'F:/Documents/GitHub/ECE4810J_FA2022_TA/Lab1/lab_
  → 1_1/Zynq_CPU_wrapper_hw_platform_3/zynq_fsbl/zynq_fsbl_
9 bsp'
1 "Compiling rtl_multiplier..."
2
3 cc1.exe: fatal error: *.c: Invalid argument
4 compilation terminated.
5 make[3]: *** [Makefile:18: libs] Error 1
6 make[2]: *** [Makefile:46:
  → ps7_cortexa9_0/libsrc/rtl_multiplier_v1_0/src/make.libs] Error 2
7 make[1]: *** [Makefile:18: all] Error 2
8 make[1]: Leaving directory 'F:/Documents/GitHub/ECE4810J_FA2022_TA/Lab1/_
  → lab1_1/Zynq_CPU_wrapper_hw_platform_3/zynq_fsbl/zynq_fs
9 bl_bsp'
```



```

10
11 make: *** [Makefile:30: zynq_fsbl_bsp/ps7_cortexa9_0/lib/libxil.a] Error
   → 2

1 ../src/lab1-3.c:2:10: fatal error: xbasic_types.h: No such file or
   → directory
2     2 | #include "xbasic_types.h"
3       | ~~~~~
4 compilation terminated.
5 make[1]: *** [src/subdir.mk:26: src/lab1-3.o] Error 1
6 make: *** [makefile:34: all] Error 2

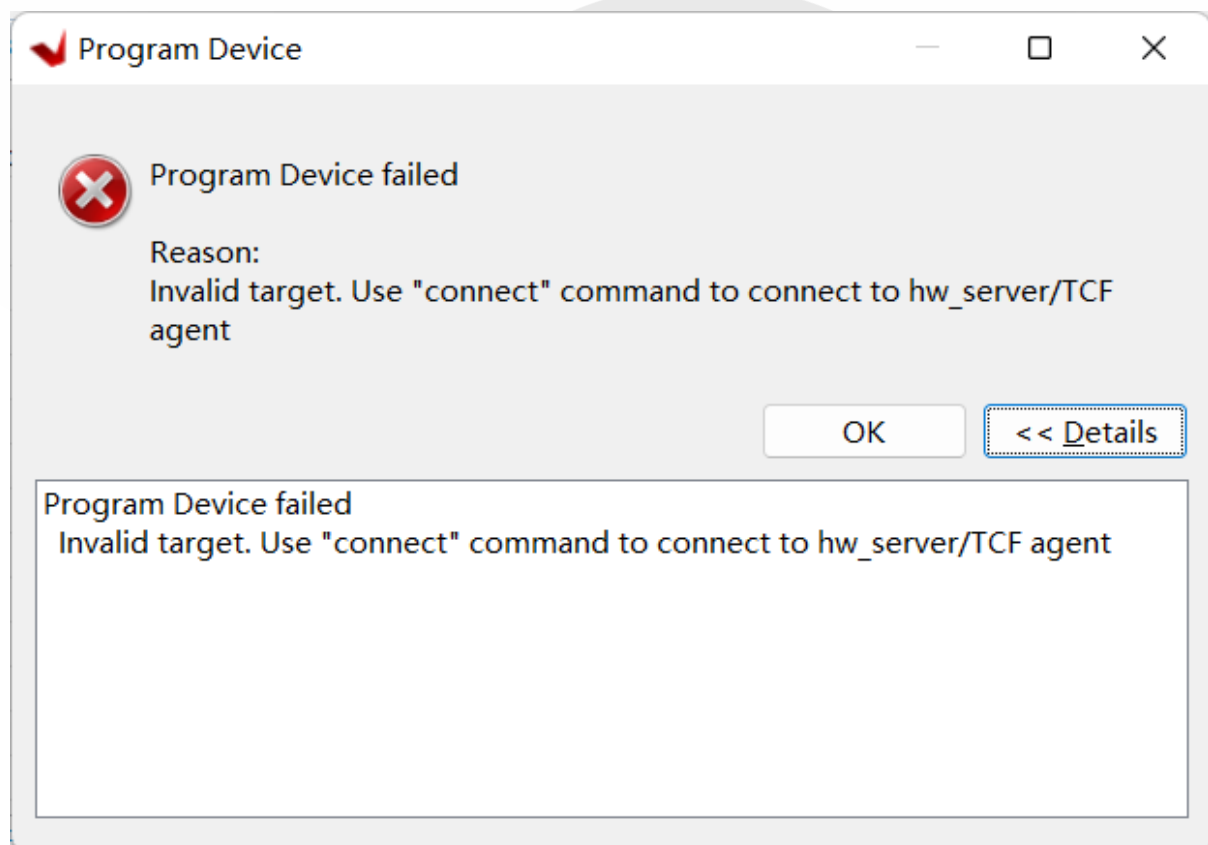
```

you can see that `xbasic_types.h` can be found under
`Zynq_CPU_wrapper_hw_platform_3\zynq_fsbl\zynq_fsbl_bsp\ps7_cortexa9_0\include`.
This is an old bug of Xilinx Vitis. Please read the reference article [\[8\]](#).

B.4 Vitis Program Device

Firstly, you need to check the connection between your computer and the Arty-Z7. If the connection is set up, then open "XSCT Console" in the "Xilinx" menu. Run command [\[15\]](#)

```
xsct% connect
```



Then, it will show:

Cannot create listening port tcp:127.0.0.1:3121: Socket bind error.

You can refer to [7]. In a nutshell, if you are using Windows, just run the command in the Command Line Prompt:

```
net stop winnat
```

This command stops the Windows NAT Driver service. This service is required for the network communication of virtual machines like Windows Subsystem for Linux (WSL), so if you stop it, you will have problems with that. Restarting the computer will reset the configuration.

If you get an "Error Launching Program" window showing

Error while launching program:

Memory write error at 0x100000. MMU section translation fault

Please read the reference article: [9].

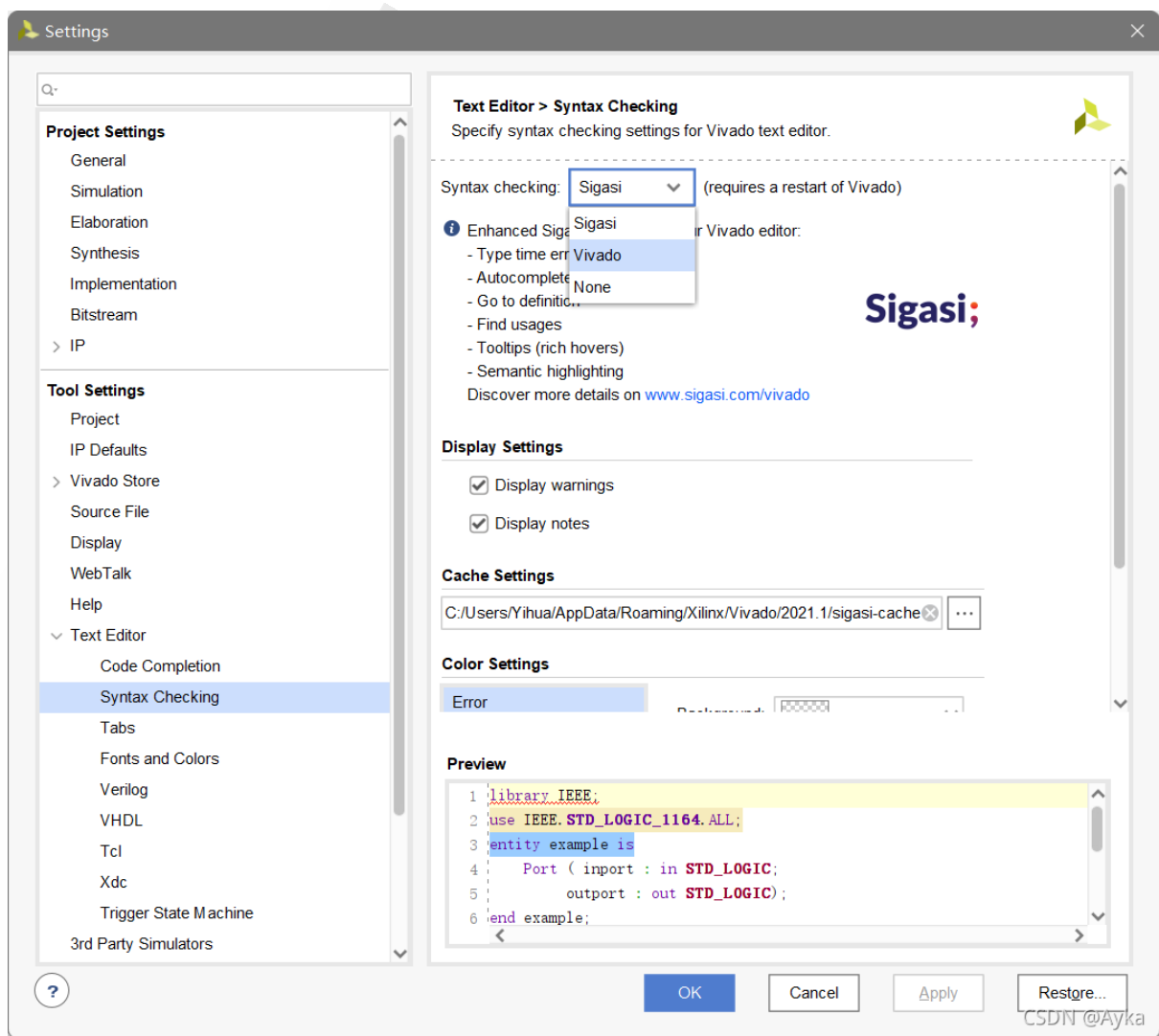
B.5 AXI IP Connection

If Vivado fails to correctly assign addresses to each AXI IP connected to your processor, the Address Editor can be accessed through its tab in the Diagram pane. Addresses can be assigned to unmapped peripherals by typing the desired address into the peripheral's Master Base Address column. It should be noted that addresses must be aligned in the memory space - for instance, an address with a range of 4K (bytes) takes up a range of 0x1000 addresses and must have three trailing zeros. Address ranges for different segments cannot overlap. Assigning a segment to address 0 may result in assertions in some software drivers and should be avoided. After manually assigning addresses, the block design should be re-validated.

<input type="checkbox"/> Assigned (2) <input checked="" type="checkbox"/> Unassigned (0) <input checked="" type="checkbox"/> Excluded (0) <input type="button" value="Hide All"/>						
Name	Interface	Slave Segment	Master Base Address	Range	Master High Address	
Network 0						
/processing_system7_0						
/processing_system7_0/Data (32 address bits : 0x40000000 [1G])						
/axi_gpio_buttons/S_AXI	S_AXI	Reg	0x4120_0000	64	0x4120_FFFF	
/axi_gpio_led/S_AXI	S_AXI	Reg	0x4121_0000	64	0x4121_FFFF	

B.6 Vivado Startup

If Vivado is stuck initializing language server, in Tools -> Settings -> Tool Settings -> Text Editor -> Syntax Checking, change Syntax checking from Sigasi to Vivado [13]:



References

- [1] Actel Corporation. *Designing FIR Filters with Actel FPGAs*. 2022. URL: https://www.microsemi.com/document-portal/doc_download/129923-fir-filters-an.
- [2] Diligent. *Pmod™*. Oct. 28, 2020. URL: <https://diligent.com/reference/pmod/start>.
- [3] Qin Huang and Zilin Wang. "Design and Comparison of FIR Filter Based on DSP Builder and HDL Coder". In: *2020 IEEE 15th International Conference on Solid-State & Integrated Circuit Technology (ICSICT)*. Nov. 2020, pp. 1–3. DOI: [10.1109/ICSICT49897.2020.9278259](https://doi.org/10.1109/ICSICT49897.2020.9278259).
- [4] jditte. *Vitis IDE not responding (frozen) on Windows 10*. Nov. 25, 2019. URL: https://support.xilinx.com/s/question/0D52E00006hpJxsSAE/vitis-ide-not-responding-frozen-on-windows-10?language=en_US.

-
- [5] Mathias Lang. *Filter order vs number of taps vs number of coefficients*. Apr. 16, 2013. URL: <https://dsp.stackexchange.com/questions/8685/filter-order-vs-number-of-taps-vs-number-of-coefficients>.
 - [6] Yihua Liu. *Vivado 2021.2 Tcl Shell no appropriate Visual C++ redistributable error*. June 27, 2022. URL: <https://blog.csdn.net/yihuajack/article/details/125489425>.
 - [7] Yihua Liu. *Xilinx Vitis 2021 Cannot create listening port tcp:127.0.0.1:3121: Socket bind error*. Sept. 13, 2022. URL: <https://blog.csdn.net/yihuajack/article/details/120722348>.
 - [8] Yihua Liu. *Xilinx Vitis arm-xilinx-eabi-gcc.exe: error: *.c: Invalid argument*. Oct. 21, 2021. URL: <https://blog.csdn.net/yihuajack/article/details/120881411>.
 - [9] Yihua Liu. *Xilinx Vitis Error Launching Program: Memory write error MMU section translation fault*. Oct. 21, 2021. URL: <https://blog.csdn.net/yihuajack/article/details/120897127>.
 - [10] Yihua Liu. *Xilinx Vitis executables selected for download on to the following processors doesn't exist*. Sept. 27, 2021. URL: <https://blog.csdn.net/yihuajack/article/details/120515370>.
 - [11] Yihua Liu. *Xilinx Vitis Startup No Response*. Sept. 27, 2021. URL: <https://blog.csdn.net/yihuajack/article/details/120511530>.
 - [12] Yihua Liu. *Xilinx Vivado 2021 Export Platform Hardware Error: No default platform clock is selected*. Oct. 12, 2021. URL: <https://blog.csdn.net/yihuajack/article/details/120714268>.
 - [13] Yihua Liu. *Xilinx Vivado Stuck Initializing Language Server*. Oct. 18, 2021. URL: <https://blog.csdn.net/yihuajack/article/details/120830612>.
 - [14] yangc LPMMLPMM123. *Vitis IDE (Not Responding) for the first time use*. Aug. 24, 2021. URL: https://support.xilinx.com/s/question/0D52E00006hpSSmSAM/vitis-ide-not-responding-for-the-first-time-use?language=en_US.
 - [15] MarkSe. *Vivado Unable to connect to hw_server*. Sept. 4, 2020. URL: https://support.xilinx.com/s/question/0D52E00006hpNYHSA2/vivado-unable-to-connect-to-hwserver?language=en_US.
 - [16] MathWorks. *Generate HDL Code from Simulink Model*. 2022. URL: <https://www.mathworks.com/help/hdlcoder/gs/example-generating-hdl-code-from-a-simulink-model.html>.
 - [17] Manjunatha. P. *UNIT - 7: FIR Filter Design*. J.N.N. College of Engineering, Shimoga, Oct. 25, 2016. URL: <https://jnnc-ece-manjunath.weebly.com/uploads/1/9/2/0/19204775/fir-filter-design.pdf>.