FPGA Remote Lab Setup

Mini project report

Bala Dhinesh M (EE19B011)
Guide: Prof. Nitin Chandrachoodan
Electrical Engineering
IIT Madras

July 3rd, 2022

Github Link | **Video Recording Link**

Introduction:

Field-programmable gate arrays (FPGAs) are hardware circuits that can be programmed to carry out logical operations. They provide a sweet spot between implementing algorithms in software and fabricating application-specific integrated circuits (ASICs). They generally provide far better performance and power-efficiency than software implementations, and they require far less time and expense to implement than ASICs. They can be reprogrammed as needed to upgrade functionality or fix bugs, even after deployment to customers (in the "field").

So can FPGAs replace CPU entirely?

Not really! CPUs are well optimized for processing sequentially and very limited capabilities of parallelism for high-speed processing. FPGAs perform very well in parallel processing and also consumes less power and price. FPGA has its own problems, though far more accessible than ASICs, FPGAs can still be a bit costly and difficult to learn for beginners and students as the programming is relatively more complicated. So the best option is to use the best of both worlds(CPU and FPGA) by combining the two together as a singular entity with features of both. This solution makes them really flexible and efficient as the CPU handles general computing and data processing and FPGA logic blocks are specially used for custom reconfigurable tasks.

This project aims to provide users with access to our FPGAs remotely for testing their designs and abstracting away some of the CPU-FPGA interface configurations by providing an automated script. The user just needs to provide their RTL design(a Verilog file) and the automated script takes care of creating the IP and link with the processor.

Approach:

This project uses Xilinx Zynq FPGA which has an ARM processor and FPGA logic blocks integrated. Advanced eXtensible Interface (AXI) is used for communicating between the FPGA(PL - Programmable Logic) and the ARM processor(PS - Processing System). Processing data in and out efficiently in the FPGA is one of the most crucial parts. There is quite a lot of configuration setup to be done to achieve this. This project abstracts away some of the configuration setups to the user by providing a harness.

PYNQ, an open-source framework by Xilinx is used in this project for interacting between CPU and FPGA. It is a new design concept whereby designers create hardware “overlays” to program the FPGA part of the ZYNQ device, which are then interacted with using Python code running in a Jupyter notebook on the device’s processor.

The main advantages of using the PYNQ framework in this project are:

- The user doesn't require Vivado software installed on their machine.
- PYNQ image provides a Jupyter server to which the client can connect with the board through WiFi or directly through Ethernet cable.
- It interacts with the CPU and FPGA using Python, which is an easy and user-friendly programming language.
- Open-sourced.

The disadvantage of the PYNQ framework:

- PYNQ image works only on Zynq-based Xilinx FPGAs.
- PYNQ images are provided only for limited boards. For other Zynq-based boards, we need to create custom PYNQ images. Creating custom PYNQ images is quite difficult. Zedboard and Zybo are the most commonly used Zynq-based boards that don't have PYNQ images. So we have created custom PYNQ images for these boards.

Steps to run:

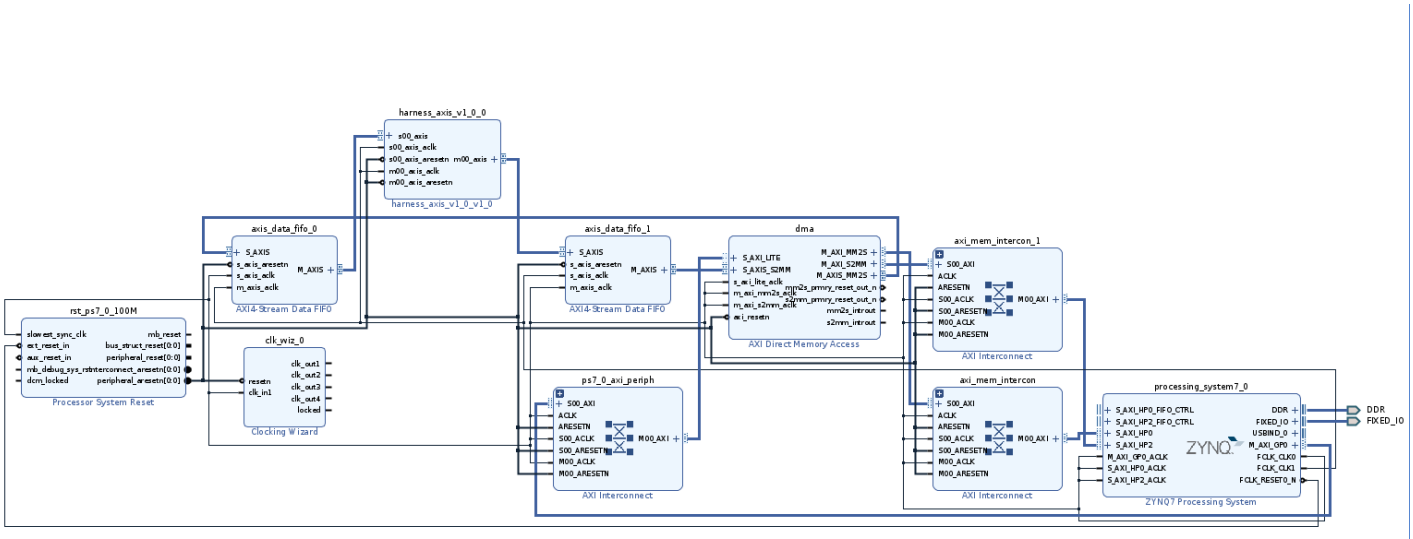
```
python harness.py --interface axis
# Arguments:
-- interface: axis or axi(Required parameter)
-- jobs: launch runs on Vivado for bitstream generation(default: 4)
```

IP Generation:

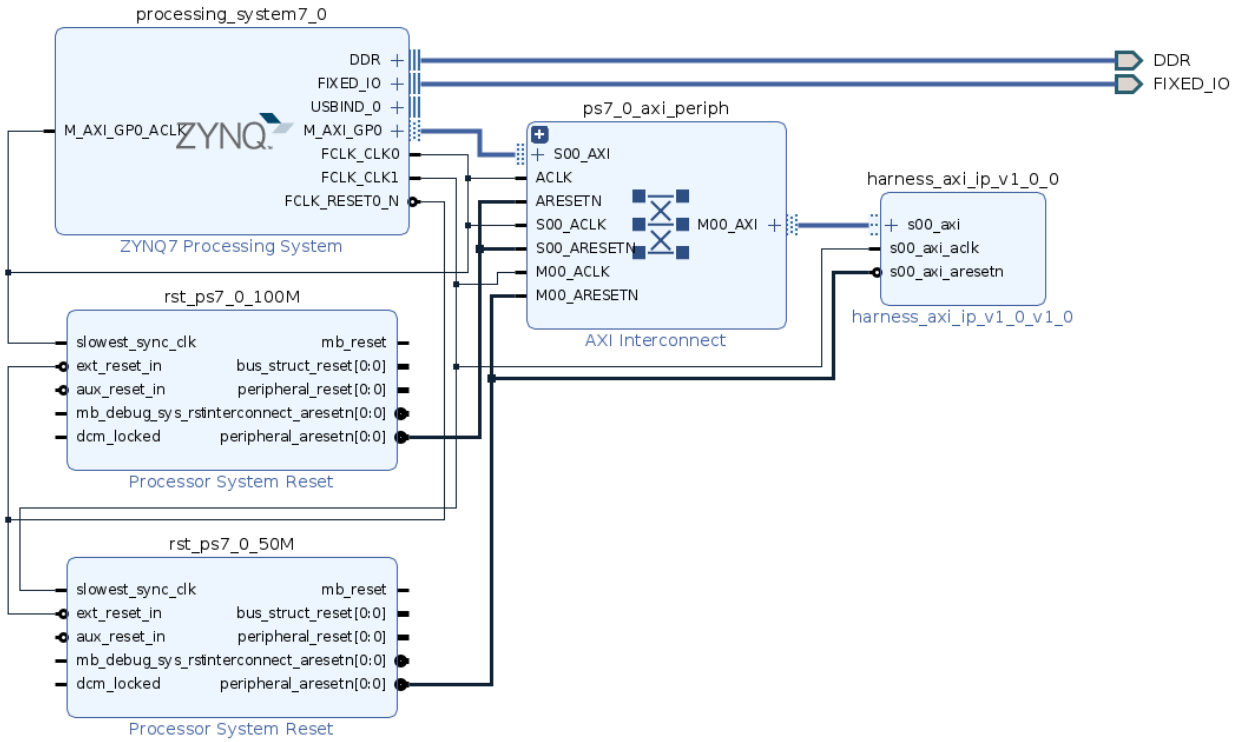
First this will take *harness_axis.v* file for AXI stream and *harness_axis.v* file for AXI lite as input. The filename and module name should be fixed. Also, this file has clk and reset signals as first and second ports. This should also be fixed. The script will generate the IP(AXI stream or AXI Lite from the `--interface` argument) from the user design. For this design to make an AXI Stream/Lite IP, we have created an *ip_create.tcl* script template for each interface inside *src* directory. Since the user design will have a different number of port output and inputs, we need to properly instantiate these port names along with their widths in the IP template. So this *harness.py* automation file will handle that by extracting the input and output ports along with their widths and instantiating the module. The IP project file will be stored inside *out* directory as *harness_axi_ip and harness_axis_ip* for AXI Lite and AXI Stream respectively. Next, the script will generate the block design as below.

Block design generation:

AXI Stream: AXI4-Stream Data FIFO is used before and after the *harness_axis_v1_0_0* IP block to set the clock frequency of user IP higher than other blocks. This helps to run the design at its maximum operable condition.



AXI Lite:



Note:

- We can observe from the diagram that the AXI Lite and Stream harness IPs are connected to a separate clock frequency signal(FCLK_CLK1). We can change the value of the frequency in the Jupyter notebook as Clocks.fclk1_mhz = <ADD FREQ in MHz>. With this, we can find the maximum frequency of operation for the design.

- The automation script for AXI Lite supports up to eight 32-bit registers excluding clk and reset signals. So we can insert up to eight input/output ports in the design.

Bitstream and overlay generation:

Next the script will generate a bitstream file. The block design and bitstream will be generated with the help of *bd_bitstream.tcl* file provided. Once the project is done and the bitstream file is created, the block design project file will be stored inside *out* directory as *harness_axi_proj and harness_axis_proj* for AXI Lite and AXI Stream respectively.

Finally the script will store the overlay - bitstream(.bit), hardware handoff(.hwh) and block design(.tcl) files inside *out/PYNQ_files/overlay* directory.

Cloudflare setup:

Cloudfare is used to share the FPGA Jupyter Notebook lab environment anywhere in the world. Refer to these slides for setting up Cloudflare. For authentication, each user needs to register in the list for accessing the FPGA. Once they are registered every time they log in, they will receive a code through the mail to access the FPGA. We have done this setup and tested it on our Zedboard and PYNQ-Z2 boards under the *pynq.technowiz.org* domain.

Once we get access to the Jupyter environment from *pynq.technowiz.org*, we can write our software interfacing CPU and FPGA on Python by using the overlay created from the script inside *out/PYNQ_files/overlay* directory. Refer to the *example/axis_adder* directory on how to write code. For more details, refer to this documentation.

Example design(Four bit AXI Stream Adder):

Code:

```
module harness_axis (
   input clk,
   input reset,
   input [3:0] a,
   input [3:0] b,
   output [4:0] c
   );

assign c = a + b;

endmodule
```

This 4-bit adder is ran through the automated script. The script will recognize the input and output ports and write the instantiation in a temporary file as below.

```
harness_axis harness_axis_inst(
    .clk(s00_axis_aclk),
    .reset(s00_axis_aresetn),
    .a(axis2pipe_data[4-1:0]),
    .b(axis2pipe_data[8-1:4]),
    .c(pipe2axis_data[5-1:0])
);
```

We can get the overlay files inside out/PYNQ_files directory once the script completes. Jupyter file is included for this example under *example/axis_adder*. On running this on the FPGA Jupyter server we can observe that this adder example shows timing errors when we go above 600MHz(FCLK_CLK1).

PYNQ image file generation

PYNQ custom image for Zedboard and Zybo:
https://drive.google.com/drive/folders/1oRQwTTHbgVj5rta6s7tMRGdDeee47aNy?usp=sharing

Vitis, Vivado and Petalinux version 2020.2 is used to create the images on a Ubuntu 18.04 Virtual Machine. For more details on creating your custom PYNQ image on any Zynq-based boards, refer to this link.

Conclusion:

With the help of this script, we can convert any design to an AXI interfaced IP and interact with the processor and not worry much about the steps behind achieving it. Cloudflare helps to provide FPGA remote access to anywhere in the world with authentication and encryption. PYNQ framework allows easy interaction between PS and PL and also helps in finding the maximum frequency of operation.