TD 0x02	Codesign		Bilel
	Hardware debugging		

TD 02: Hardware debugging

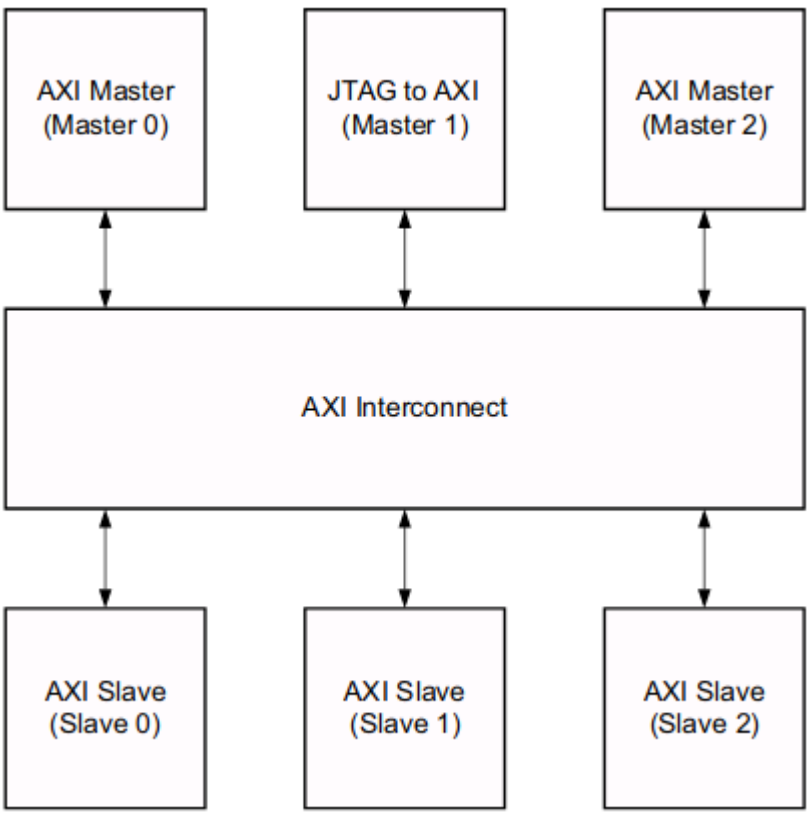
Tools presentation :

JTAG to AXI IP:

The LogiCORE™ JTAG to AXI Master IP core is a customizable core that can generate the AXI transactions and drive the AXI signals internal to the FPGA in the system. The AXI bus interface protocol can be selected using a parameter in the IP customization Vivado®Integrated Design Environment (IDE). The width of the AXI data bus is customizable. This IP can drive AXI4-Lite or AXI4 Memory Mapped Slave through an AXI4 interconnect. Run time interaction with this core requires the use of the Vivado logic analyzer feature.Features:

•Provides AXI4 master interface
•Option to set AXI4 and AXI4-Lite interfaces
•User Selectable AXI data width – 32 and 64
•User Selectable AXI ID width up to four bits
•User Selectable AXI address width – 32 and 64
•Vivado logic analyzer Tcl Console interface to interact with hardware
•Supports AXI4 and AXI4-Lite transactions

The following figure shows an AXI system which uses the JTAG to AXI Master core as an AXI Master. The JTAG to AXI Master core does not have its own address space and responds to all the addresses you initiate. The JTAG to AXI Master core can communicate to all the downstream slaves (S0, S1, and S2 in this case) and can coexist with the other AXI Master in the system.



Maximum Frequencies

The JTAG to AXI Master core is designed to run at design clock frequencies up to 200 MHz, but maximum clock frequency may be limited by other factors in the design such as overall utilization or routing congestion.

TCL commands :

create_hw_axi_txn
Description
Create a hardware AXI transaction object where:
create_hw_axi_txn [transaction name] [jtag_axi_hw_name] [-address <arg>] [-data <arg>] [-type <arg>]

Returns
New hardware AXI transaction object

AXI4-Lite Example
Create a write AXI burst transaction with eight 32-bit data:
create_hw_axi_txn wr_txn_lite [get_hw_axis hw_axi_1] -address 00000000 -data 12345678 -type write

Then run_hw_axi wr_txn_lite

Create a read AXI burst transaction with eight 32-bit data:
create_hw_axi_txn rd_txn_lite [get_hw_axis hw_axi_1] -address 00000000 -type read

ILA (Integrated Logic Analyzer):

The customizable Integrated Logic Analyzer (ILA) IP core is a logic analyzer that can be used to monitor the internal signals of a design. The ILA core includes many advanced features of modern logic analyzers, including boolean trigger equations and edge transition triggers. Because the ILA core is synchronous to the design being monitored, all design clock constraints that are applied to your design are also applied to the components of the ILA core.

Features:

• User-selectable number of probe ports and probe_width
•Multiple probe ports, which can be combined into a single trigger condition
•AXI interface on ILA IP core to debug AXI IP cores in a system

Exercise 01 :

- Use the IP packager tool to interface this module to an AXI-Lite interface .

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity alu is
  Port (
  clk : in std_logic;
  input_reg_1 : in std_logic_vector(31 downto 0);
  input_reg_2 : in std_logic_vector(31 downto 0);
  opcode: in std_logic_vector(1 downto 0);
  output_reg: out std_logic_vector(31 downto 0)
  );
end alu;
architecture Behavioral of alu is
signal output_reg_sig:  std_logic_vector(31 downto 0);
begin
process(clk)
     begin
     if clk='1' and clk'event then
      case(opcode) is
      when "00" => -- Addition
       output_reg_sig <= input_reg_1 + input_reg_2;
      when "01" => -- Subtraction
       output_reg_sig <= input_reg_1 - input_reg_2 ;
      when "10" => -- Multiplication
       output_reg_sig <= input_reg_1(15 downto 0)* input_reg_2(15 downto 0) ;
      when others =>
      output_reg_sig <= output_reg_sig;
      end case;
      end if;
     end process;
output_reg <= output_reg_sig;
end Behavioral;
```

- Create new design bloc.

- Add the ip you created previously to the design.

- add JTAG to AXI Master IP

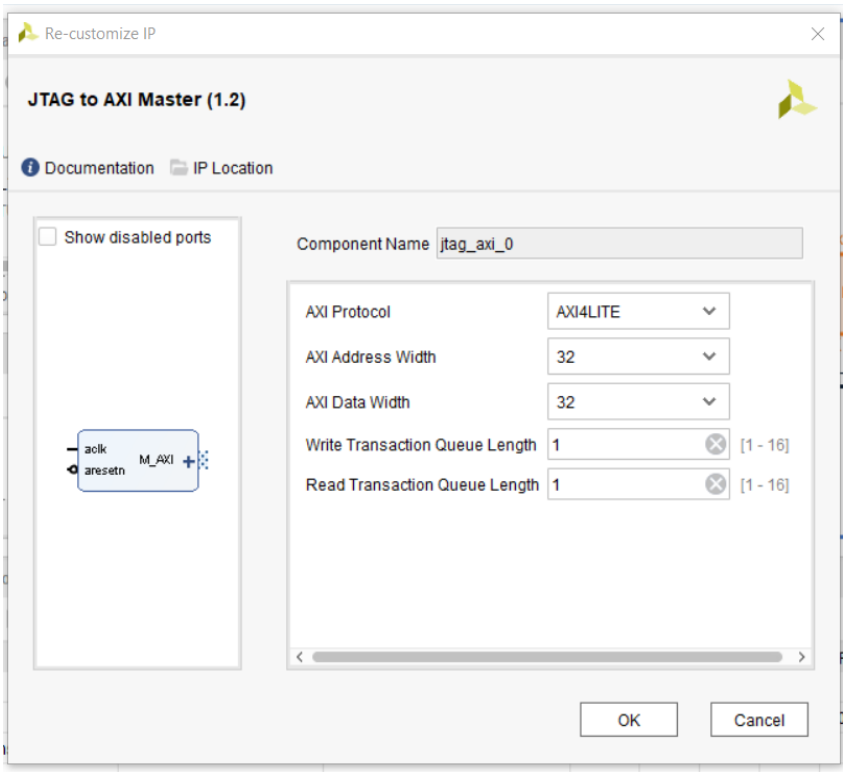
- Add a constant and connect it to the ips reset.

- Make the clock source external and connect it to the IPs clocks

at this point your design should looks like this

- Add the constraints file(comment everything except the clock source).

- Configure the JTAG to AXI Master as the following:



- run synthesis, implementation and bitstream generation.

- Go to open hardware manager, choose auto connect.

- Click program device

- enter tcl commands to test your design (go to TCL commands section in this doccument).

Example :

create_hw_axi_txn wr_txn_lite [get_hw_axis hw_axi_1] -address 0x44A0_0000 -data 00000001 -type write

run_hw_axi wr_txn_lite

create_hw_axi_txn rd_txn_lite [get_hw_axis hw_axi_1] -address 0x44A0_0000 -type read

run_hw_axi rd_txn_lite

Exercise 02 :

- Create new design bloc.

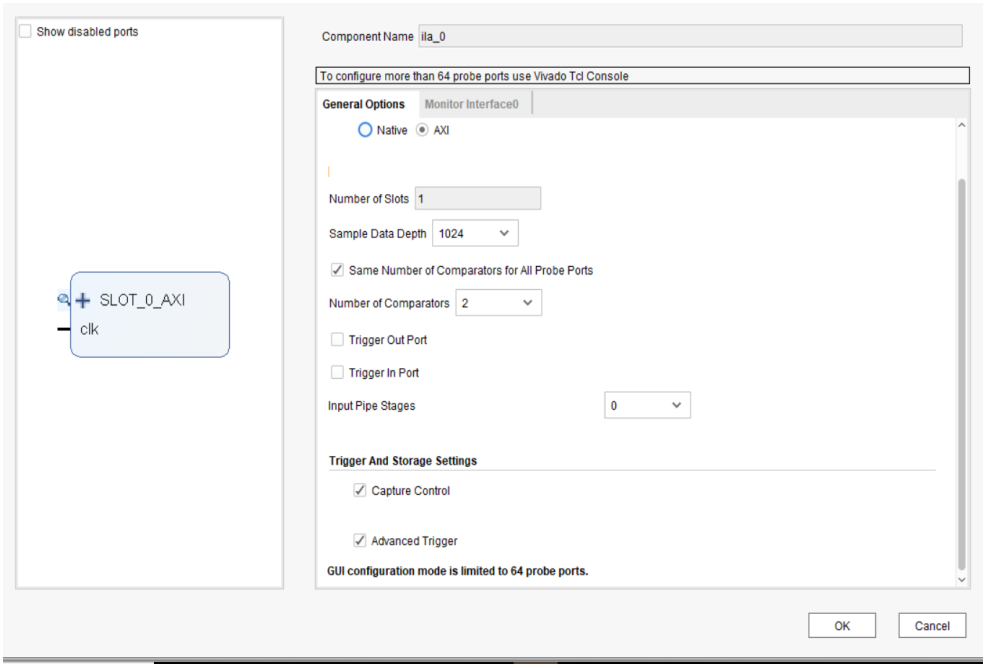
- Add the ip you created to the design.

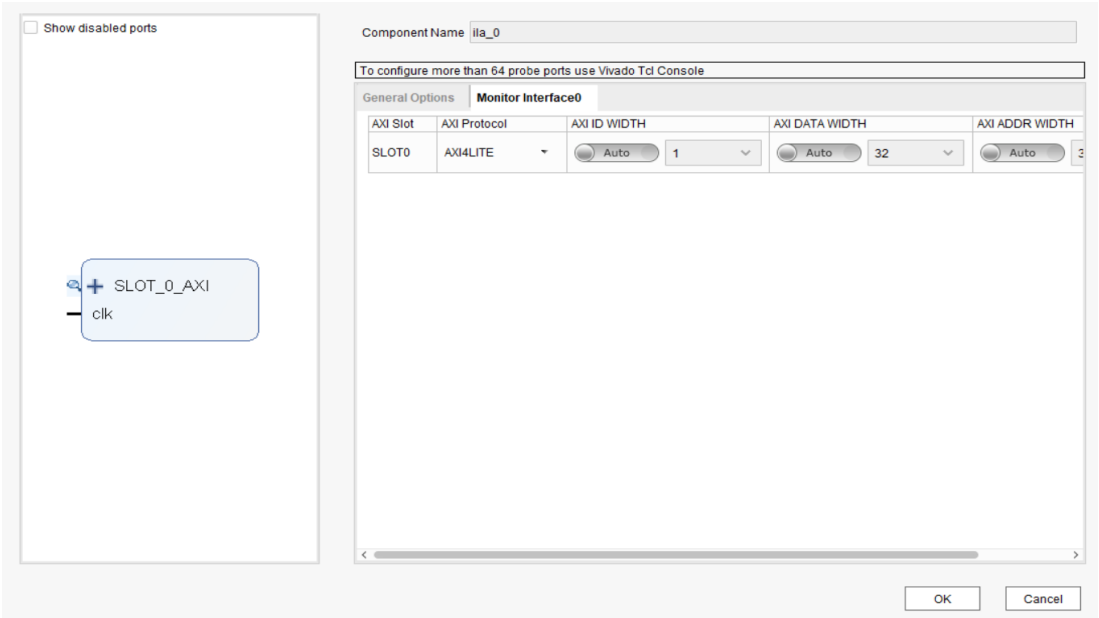
- Add the processing system to your design.

- Run bloc automation and connection automation.

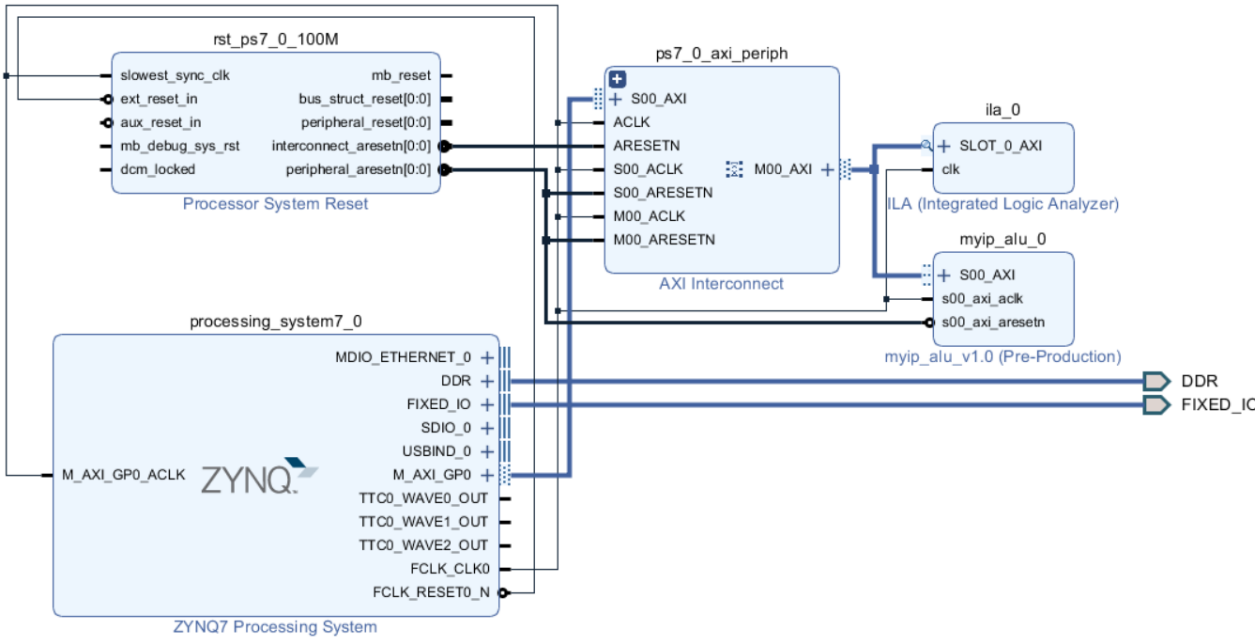
- Add ILA IP to your design.

- use the following configuration for the ip.





- Connect ILA to your IP AXI interface and to the same clock source.



- Generate the hdl wrapper.

- Run synthesis, implement, and generate the bit stream.

- Export the hardware and lunch the sdk.

- use this functions to write and read from your IP

(function to write)
```
Xil_Out32((BaseAddress) + (RegOffset), (u32)(Data));
```

(function to read)
```
Xil_In32((BaseAddress) + (RegOffset));
```

- Check the ILA video tutorial to see how to use ila to capture axi bus data in run time

- write a program to add, subtract, and multiply two numbers with the previously developed IP and check the results with ILA captures.