Xilinx Integrated Logic Analyzer

ELECTGON
www.electgon.com
contact@electgon.com

30.12.2019

Contents

Logic Analyzers are very powerful and important debugging tools that are crucial for any digital electronic development system. For FPGAs development, digital designers can insert Logic Analyzers also within their design in the FPGA. These FPGA Logic Analyzers are provided by every FPGA vendor so that developers can utilize it in order to ease debugging of their system. In this document we will see Logic Analyzer provided by Xilinx and how we can implement it.

1 How Xilinx Logic Analyzer Works

Like an oscilloscope, signals that need to be observed shall be connected to the probes of the analyzer. A trigger signal shall be connected also. Since we target to implement the logic analyzer in the FPGA, a clock signal shall be provided also. These are main connections that shall be prepared for the logic analyzer.

- Data/Probe: Data (or Probe) ports shall be connected to the data been captured. During initialization of logic analyzer core, we have to choose how many samples we need to capture from the observed signals.
- Clk: logic analyzer instance shall operate in the same clock domain of the captured signals. Therefore, it is feasible to capture only signals that belong to the same clock domain. If signals from different clock domain are needed to be captured, use logic analyzer instance for each domain.
- Trig_in: This port shall be connected to signals that will cause the trigger of the logic analyzer. Once trigger signal is asserted, the logic analyzer will capture signals connected to its Data (Probe) ports.
- Trig_out: The logic analyzer can also generate a trigger signal when a certain condition is fulfilled. The condition can be defined during run time using the connected Computer machine via JTAG. Example of Trig_out usage is we can use it to generate an interrupt signal that can be connected to an included processor (such as microblaze) or even we can use it off-chip by connecting trig_out to an FPGA pin.
- Trigger Condition: Default or native conditions that are available are simple trigger conditions like equal, less than, greater than, rising edge, falling edge, etc. In Vivado flow there is an option for more complex trigger conditions in which you can specify more triggering conditions using counters, branch statements, etc. This option can be enabled by choosing the option “Advanced Trigger”. However, this option consumes a lot of resources when implemented in the FPGA so it is not advised to enable it if you can work with simple trigger conditions. For more details about this option please refer to Appendix B in Vivado guide (UG908).
- Capture Samples: Captured values of the signals are stored in RAM instances on the FPGA. Size of the used block RAM is defined by one option called “Sample Data Depth”. The more data width (or probe width in case of Vivado flow), the more RAM used to

store captured samples. Table 1 shows relation between number of used RAM, Data depth and Data width.

		Data Depth					
		1024	2048	4096	8192	16384	32768
Data Width	32	1	2	4	7.5	15	30
	64	2	4	7.5	14.5	29	58
	128	4	7.5	14.5	29	57.5	115
	256	7.5	14.5	29	57.5	114.5	229
	512	15	29	57.5	114.5	228.5	457
	1024	29.5	57.5	114.5	228.5	456.5	913

Table 1: Number of Block RAM used by Logic Analyzer [2]

In case of Vivado flow, we don't have single Data port but group of Probe ports, each Probe port will have its own buffer. Choosing Data Depth in Vivado flow will assign this depth for each buffer of the probes.

2 Using Chipscope (ISE Flow)

Using Chipscope you can insert hardware modules into your design in order to check a working functionality during a real running of your design in Xilinx FPGA. These inserted hardware modules will act as Logic Analyzer that is connected internally with your FPGA running design. The output of this Logic Analyzer (Chipscope hardware modules) can be seen by running Chipscope Analyzer in your computer machine. In that sense you will have a Logic Analyzer whose probes are inserted inside the FPGA, and monitor is displayed in your computer.

Like a Logic Analyzer, ChipScope hardware module is waiting for signal inside the FPGA to be triggered to start capturing other desired signals. All these triggering and captured signals are defined by the user. We can visualize then how the ChipScope modules are connected with the design as in figure 1.

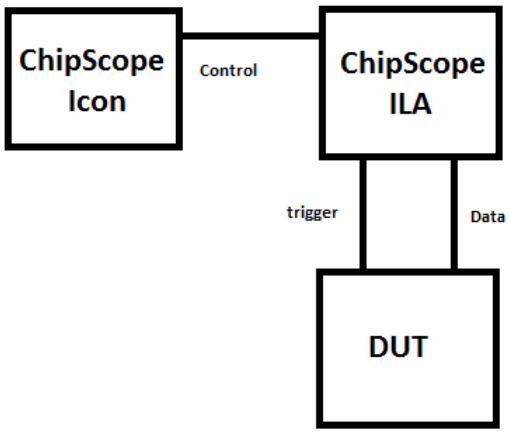

Figure 1: Chipscope Modules

As shown in the figure, ChipScope consists of ICON module and another capture module. ICON stands for Integrated Control. It is simply the interface between JTAG interface and the capture modules. This module can control up to 15 capture modules. Captures cores are modules that capture target signals values when the trigger event takes place. There are several types of capture cores provided by Xilinx. Most common module is the ILA (Integrated Logic Analyzer). This ILA is connected to the ports and signal of the DUT so that it can observe trigger and store captures data. Therefore, this ILA is consuming Block RAM and Slice Logic from the FPGA. To reduce Block RAM usage, you may reduce observed signals or reduce sample data depth.

Other Capture cores provided by Xilinx are:

- ATC2 (Integrated Logic Analyzer with Agilent Trace) core: similar to the ILA core, except data is captured off-chip by the Agilent Trace Port Analyzer.
- IBA/OPB (Integrated Bus Analyzer for CoreConnect On-Chip Peripheral Bus) core: Capture core for debugging CoreConnect OPB.
- IBA/PLB (Integrated Bus Analyzer for CoreConnect Processor Local Bus) core: Similar to the IBA/OPB core, except for the PLB bus.
- VIO (Virtual Input/Output core): Define and generate virtual I/O ports.

Because the ChipScope analyzer logic is implemented in the FPGA, it has some important limitations. The sample memory of the analyzer is limited by the memory resources of the FPGA. In a design that uses much of the FPGA's memory, there may not be much memory left over for the ChipScope cores. Also, ChipScope cannot sample as quickly as an external logic analyzer. Generally, ChipScope sampling rate will be the same as the design's clock frequency. It is therefore not possible to detect glitches with ChipScope [1].

ChipScope modules can be added in RTL design or into netlist files. In order to insert it into RTL files, the modules have to be created first using Core Generator. In the following section, We will see show to add ChipScope cores manually.

- Open Xilinx Core Generator.

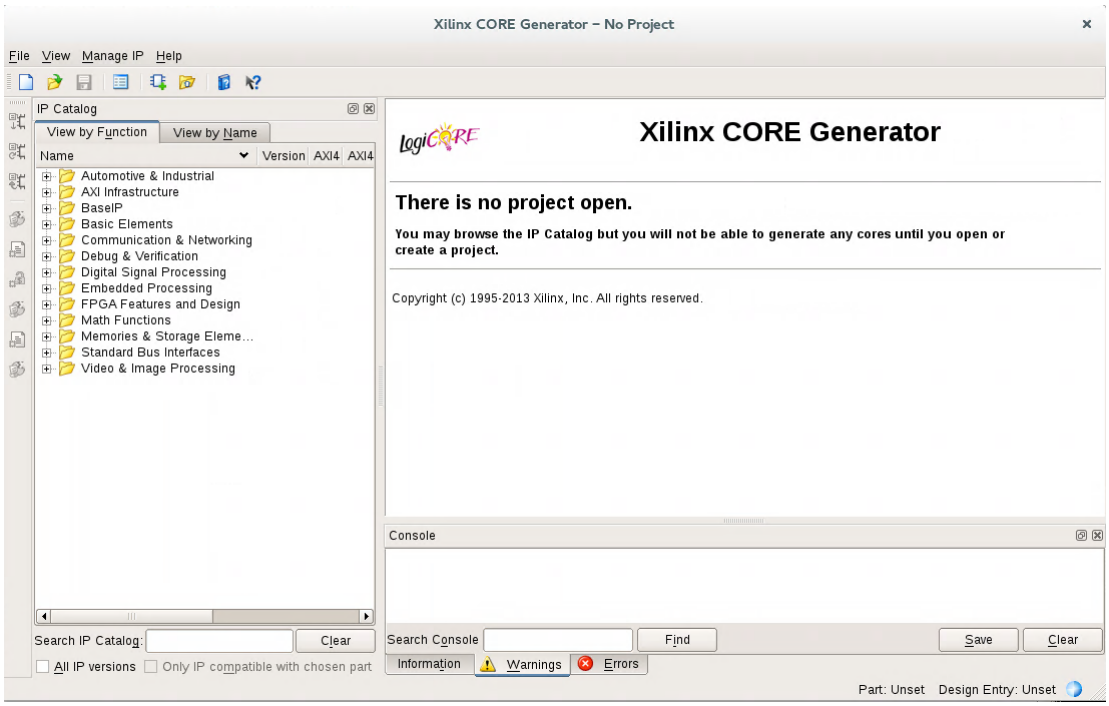

Figure 2: ISE Core Generator

- Create New Project

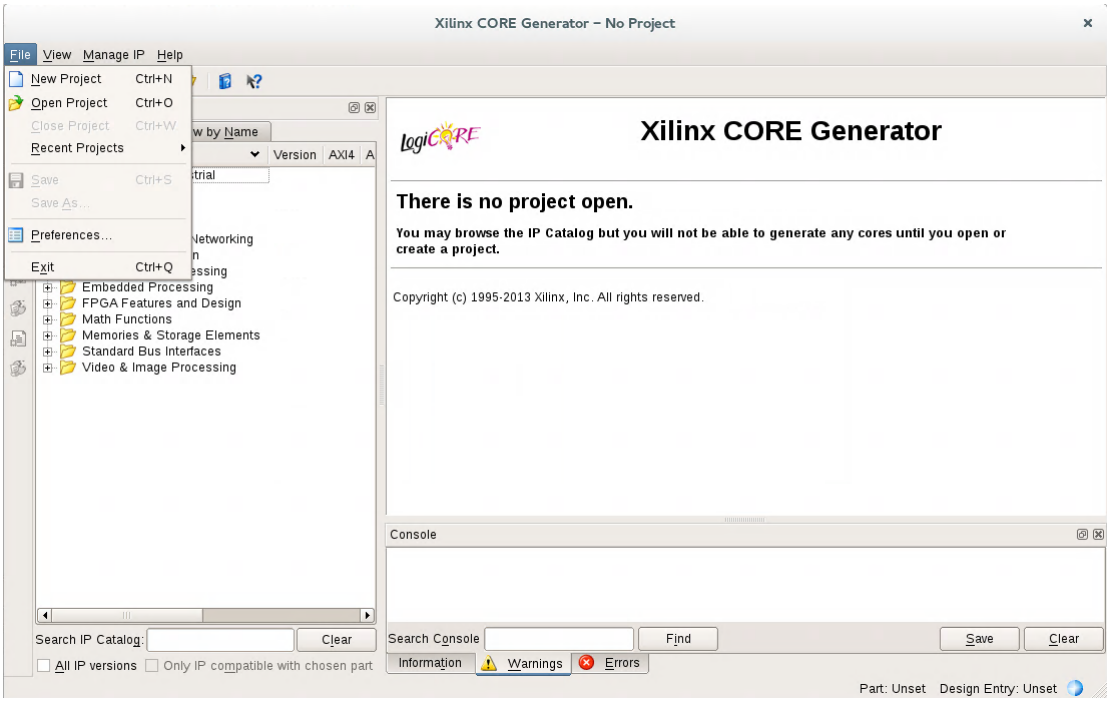

Figure 3: ISE Start Project

- Choose the right options for the project according to your target device family.

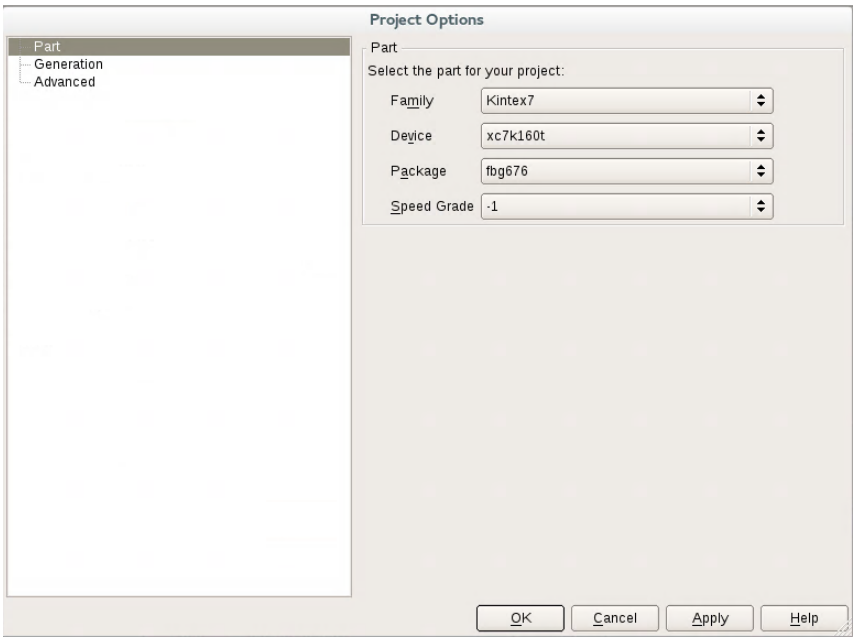

Figure 4: Target Device

- Choose in which language should the design be generated.

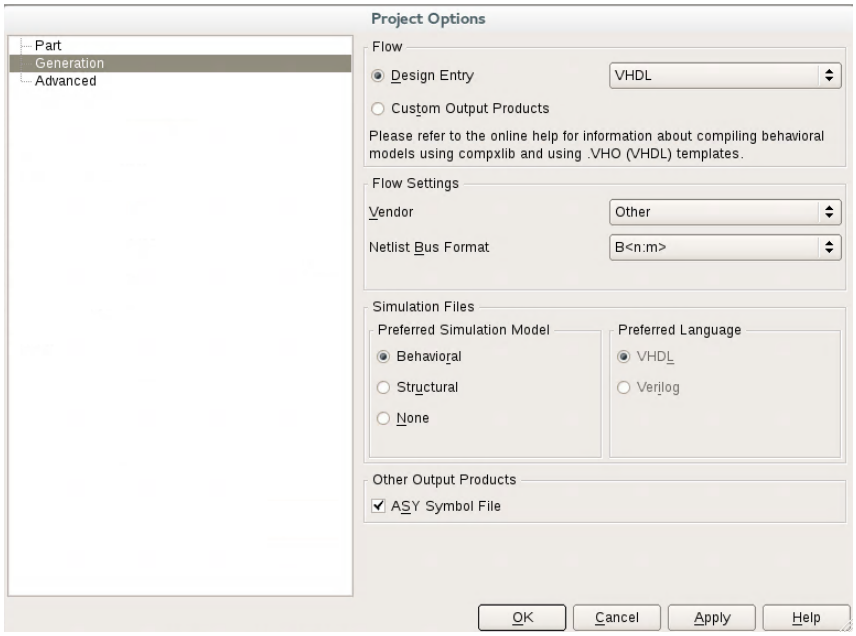

Figure 5: Generation Language

- Find ChipScope core in the available cores list.

Figure 6: ICON and ILA Generation

- Choose the ICON core to be generated. Configure it according to your debugging need.

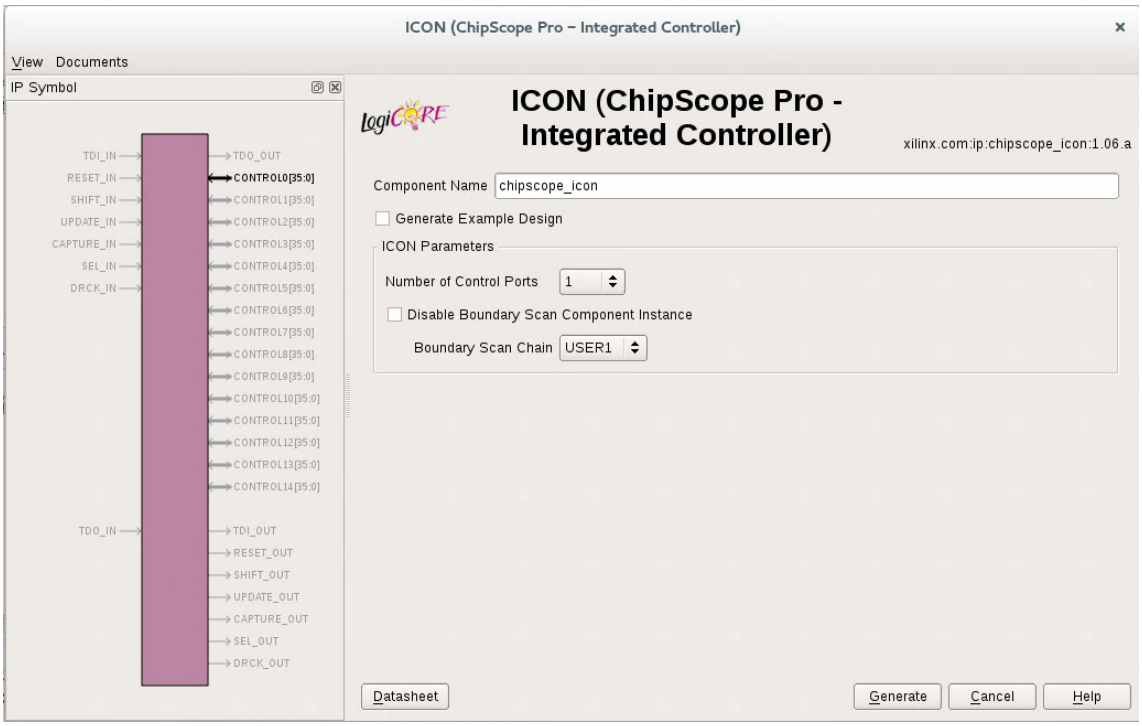

Figure 7: ICON Control Port

- Choose the ILA core to be generated. Configure it according to your debugging need.

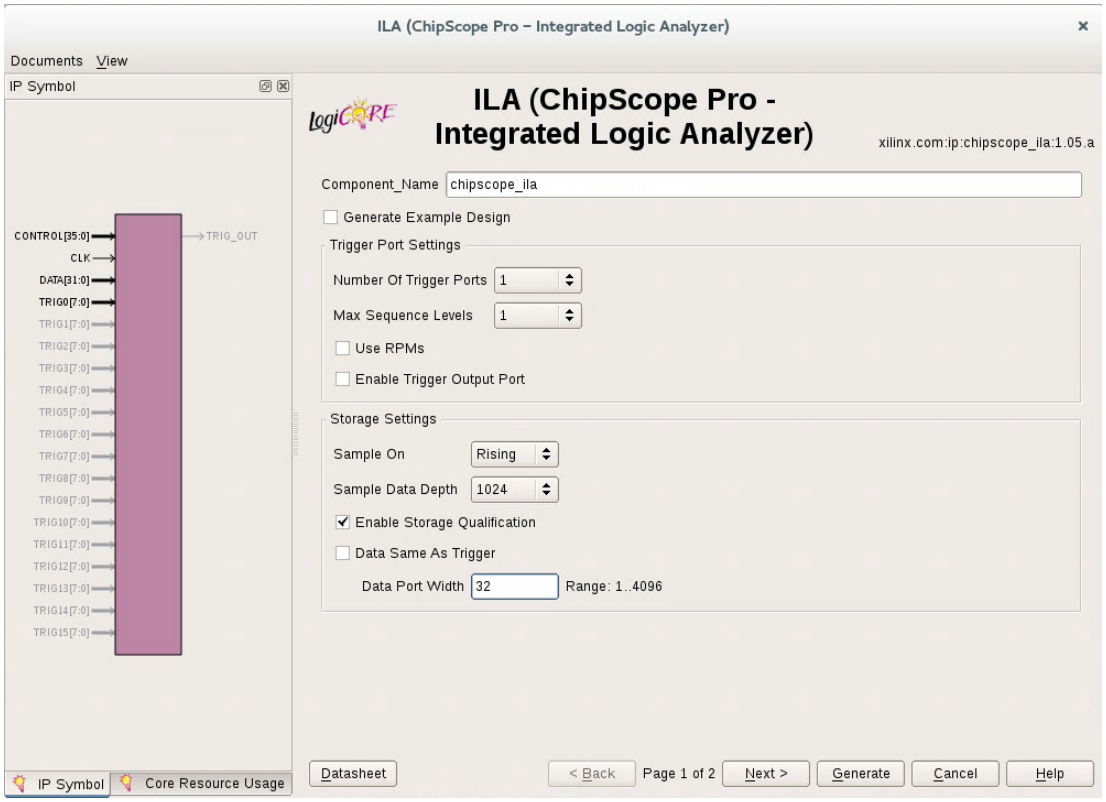

Figure 8: Setting Capturing Ports

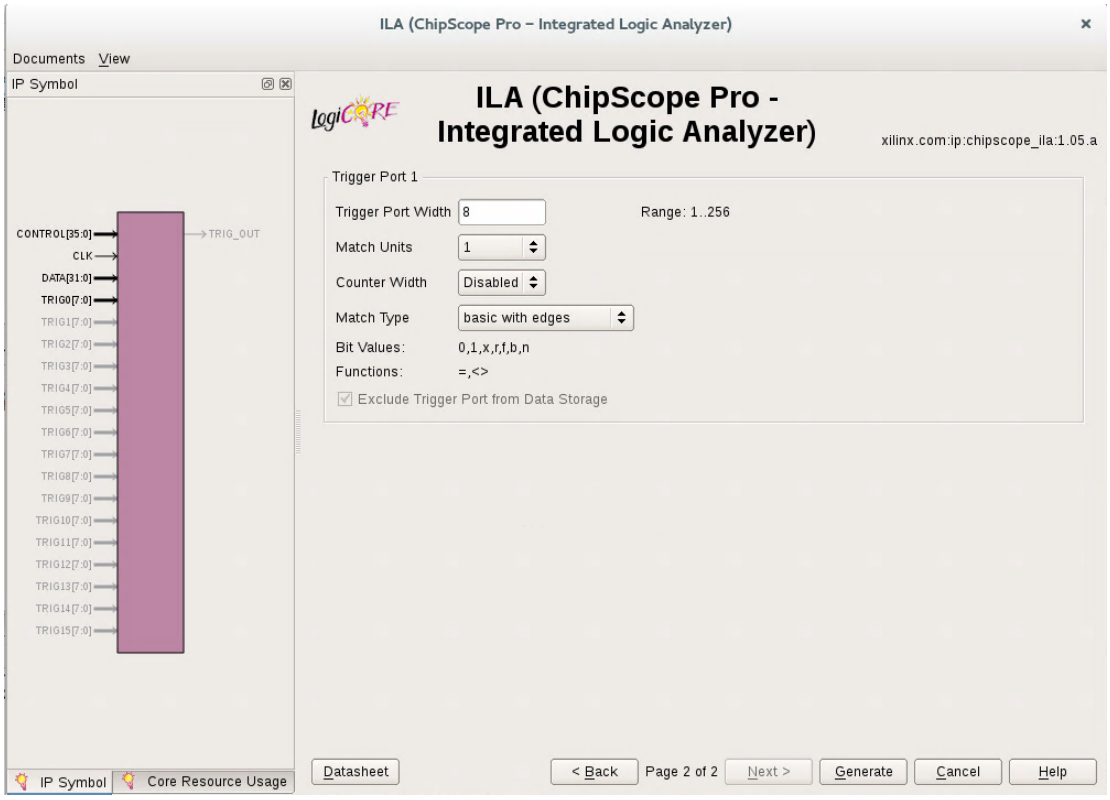

Figure 9: Setting Port Width

- Xilinx Core Generator tool will create a directory for these generated cores within your

created project directory. Browse through this directory to find two important files: chipscope_icon.ngc and chipscope_ila.ngc.

- Copy these two files into your design synthesis directory (directory where your design under test is synthesized). Or simply attach these files to your synthesis tool.

- In your HDL design file of the DUT, insert the ChipScope modules. First insert the component declaration and needed internal signals.

```
component chipscope_icon
  port (
    CONTROL0 : inout std_logic_vector(35 downto 0)
    );
end component;

component chipscope_ila
  port (
    CONTROL  : inout std_logic_vector(35 downto 0);
    CLK      : in    std_logic;
    DATA     : in    std_logic_vector(9 downto 0);
    TRIG0    : in    std_logic_vector(0 to 0);
    TRIG_OUT : out   std_logic
    );
end component;

signal CONTROL_0    : std_logic_vector(35 downto 0);
signal ChipScope_DATA : std_logic_vector(9 downto 0);
signal TRIG0_0        : std_logic_vector(0 to 0);
signal TRIG_OUT_0  : std_logic;
```

- In your DUT architecture, insert the ChipScope instances. Connect your target capture signal to the data bus and connect the triggering signal to the ChipScope trigger

```
TRIG0_0(0) <= start_trig;
ChipScope_DATA (3 downto 0) <= DDR_In2;
ChipScope_DATA (7 downto 4) <= DDR_In1;
ChipScope_DATA (8) <=int_rgmii_tx_ctl;
ChipScope_DATA (9) <= CLK_125_MHZ;

icon_i : chipscope_icon
  port map (
    CONTROL0 => CONTROL_0
    );

ila_i0 : chipscope_ila
  port map (
    CONTROL  => CONTROL_0,
    CLK      => CLK,
    DATA     => ChipScope_DATA( 9 downto 0 ),
    TRIG0    => TRIG0_0,
    TRIG_OUT => TRIG_OUT_0
    );
```

* Note, in this example you can leave TRIG_OUT as open port.

- Synthesize and implement the design into your Xilinx FPGA.
- In your Computer machine, Open ChipScope Analyzer software.

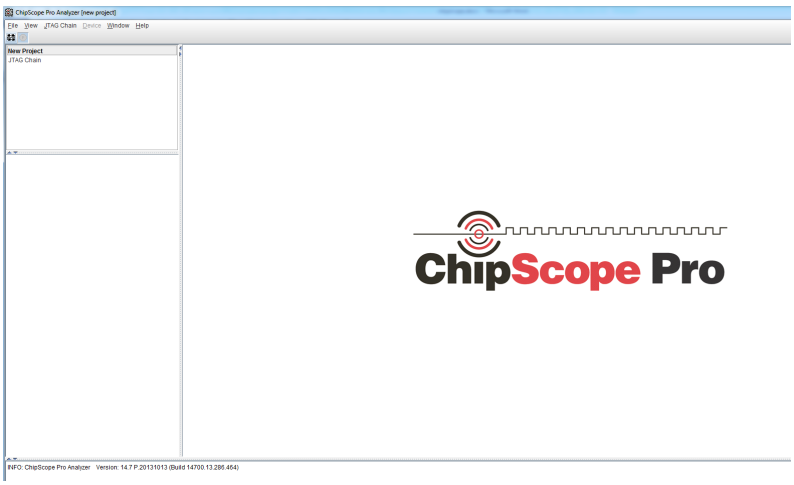

Figure 10: ISE Chipscope

- Click on the 'Open Cable/Search JTAG Chain' button to detect your FPGA.

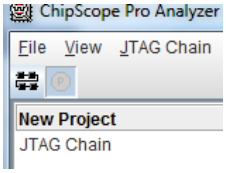

Figure 11: Open JTAG Chain

- Set the trigger value at which you are waiting the capturing event.

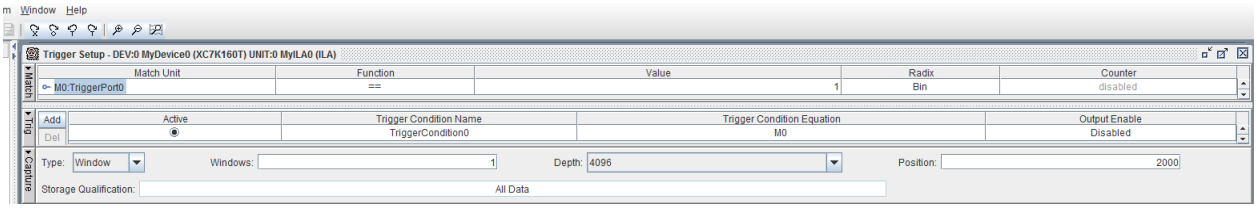

Figure 12: Setting Trigger Condition

- While your design is running in the FPGA, click on the run button to capture the data.

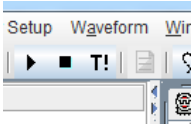

Figure 13: Chipscope Run-Stop

- When the event takes place, You will see captured values in the waveform window. You can adopt view method of the waveform according to your needs.

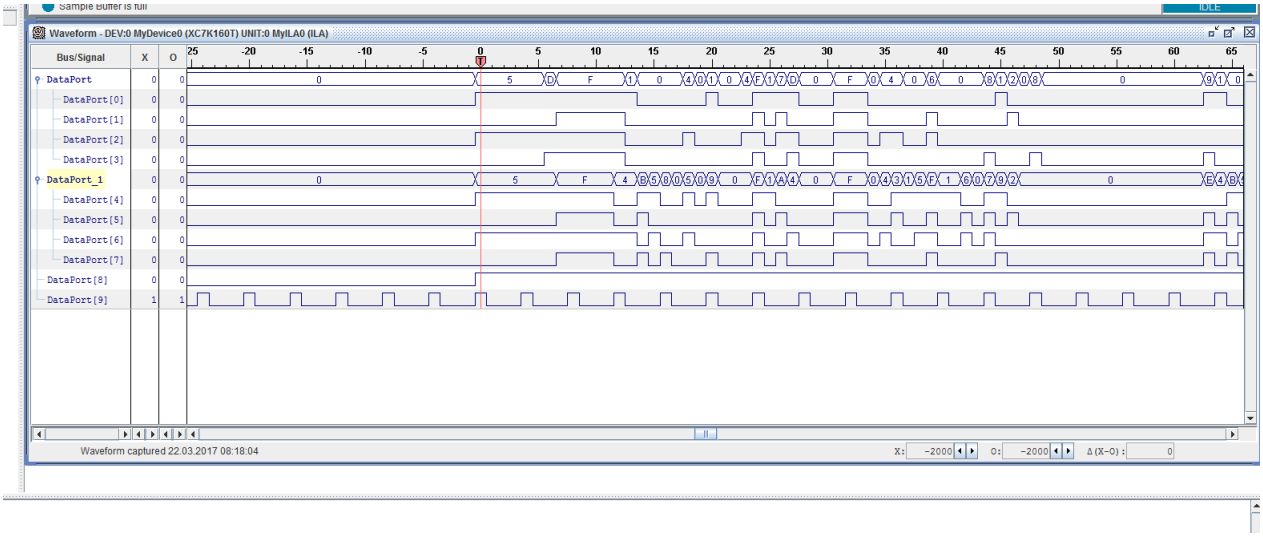

Figure 14: Chipscope Waveform Viewer

3 Using Vivado

Key-Player Framework in Xilinx has been changed in 2010 to be Vivado instead of ISE. A lot of utilities have been added/changed as well. This includes also ChipScope software that was part of ISE package. Vivado has all tools integrated, so it is not using ChipScope but other utility called Vivado Logic Analyzer. Not only but also, the hardware instance of ILA has been changed also. In Xilinx ISE, when we add ILA in the design, we have to add ICON instance as interface between ILA and JTAG interface. In Vivado, when we add ILA we don't have to add anything as Vivado will instantiate automatically a debug core hub to build the link between ILA and JTAG interface.

Therefore, the ILA instance used in Vivado is different than ILA instance used in ISE. ILA instance used in ISE flow has the following ports: clk, data, control, trig_out and trig<n>. Where n can be from 0 to 15. Each trig<n> port can consist of 1 to 256 signals (see Xilinx DS299). While ILA in Vivado has: clk, trig_in, trig_in_ack, trig_out, trig_out_ack, probe<n>. Where n can be from 0 to 1023. (see Xilinx PG172).

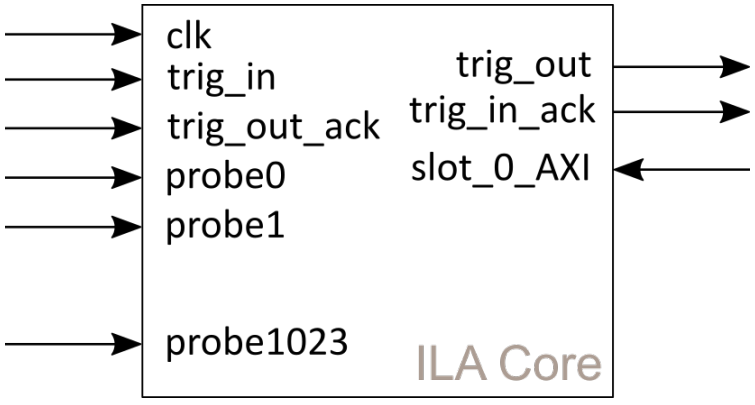

Figure 15: ILA Core Block[3]

Adding ILA in Vivado flow is straight forward, open the IP Catalog in Vivado and search for the ILA IP. Double click on the ILA IP to start creation wizard.

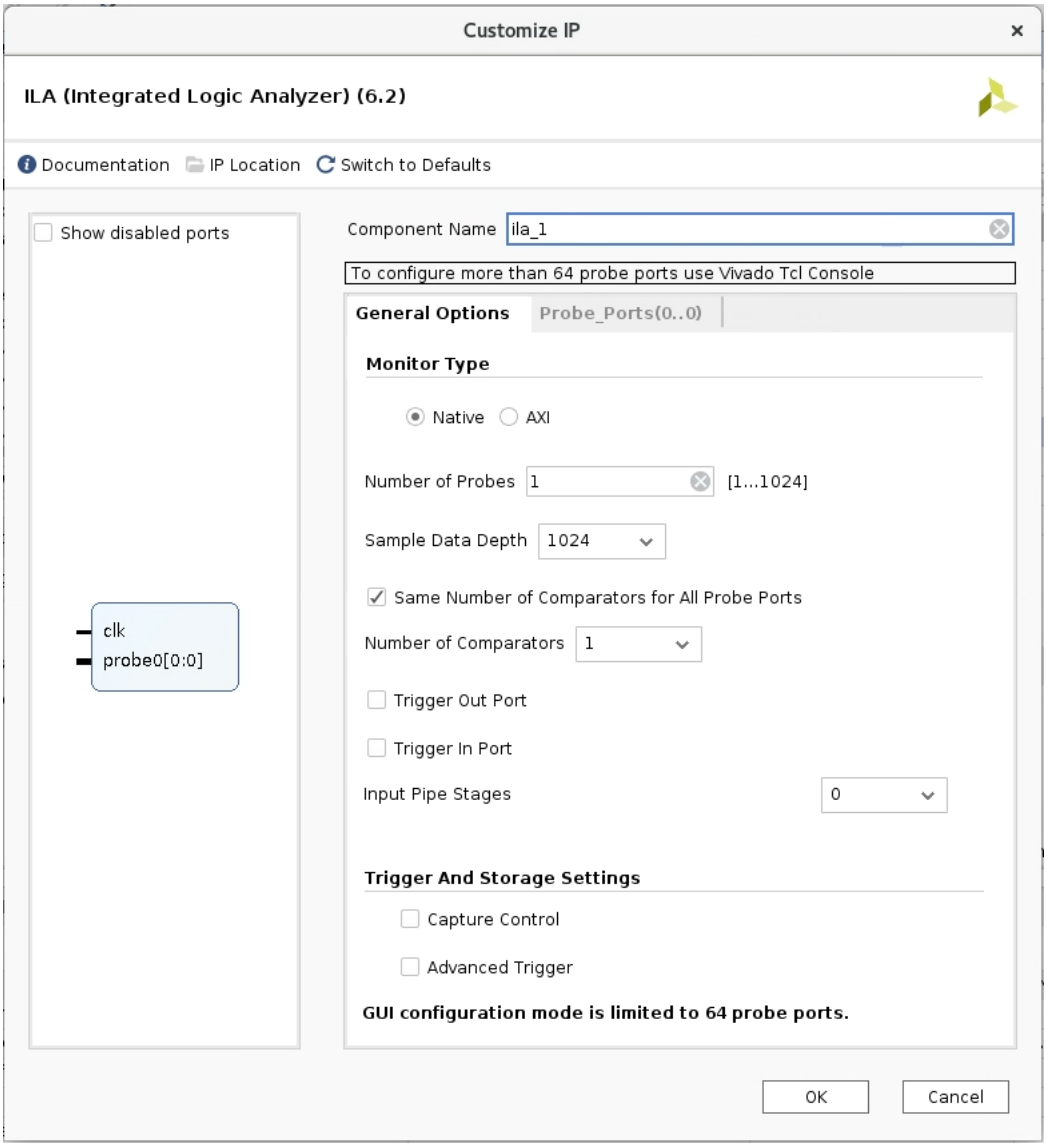

Figure 16: ILA Vivado Wizard

There you can configure the generated ILA. When you click OK, Vivado will need to generate outputs of the created ILA.

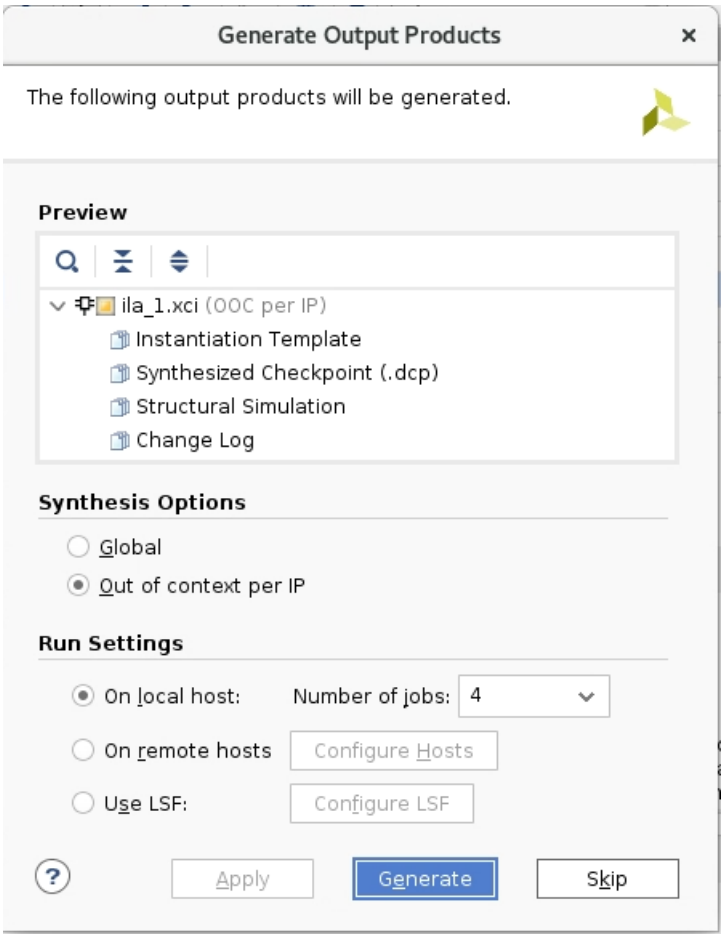

Figure 17: Generate ILA in Vivado

Click generate to see a generated template of the ILA. What is meant by generated template of ILA is a VHDL (or verilog) file that contains instantiation of the ILA component so that user can copy this component instantiation and insert it in his design source code.

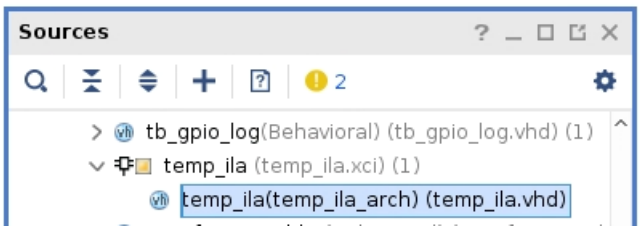

Figure 18: Instantiated ILA

So add the ILA as component in your design file and connect the target signal to be observed with probe ports of the ILA component instantiation.

3.1 Frequent Usage of ILA

So far what we did to add ILA is :

- Click “Generate”: this generates netlist of the Xilinx ILA IP.

- Add VHDL instantiation: this will include ILA as component in your design.

When you get familiar with ILA and with frequent usage of it, it will be time consuming to use Vivado GUI to create ILA instance first then launch your synthesis job. Therefore, it is easier to have a script that can create ILA instance within synthesis job without the need to create it manually. The instantiation of ILA in your VHDL still has to be carried out yourself. The "Generate" of ILA can be carried out using the following TCL script

```
set ila_module_name "ila_name"

create_ip -vlnv xilinx.com:ip:ila:6.2 -module_name $ila_module_name
set ila_instance_name [ get_ips "${ila_module_name}" ]

set_property -dict {
  CONFIG.C_NUM_OF_PROBES {15}
  CONFIG.C_PROBE11_WIDTH {8}
  CONFIG.C_PROBE12_WIDTH {8}
  CONFIG.C_PROBE13_WIDTH {64}
  CONFIG.C_PROBE14_WIDTH {64}
  CONFIG.Component_Name {$ila_module_name}
} $ila_instance_name
```

You then call this script after creating the project and importing your design source files. For understanding or modification of this script, you can observe TCL console window in Vivado when generating an ILA using GUI. You will find then how the GUI procedure is interpreted in the TCL language.

Bibliography

[1] 22.03.2017: http://www-mtl.mit.edu/Courses/6.111/labkit/chipscope.shtml

[2] Designing with Xilinx FPGAs: using Vivado

[3] Integrated Logic Analyzer, Xilinx PG172, 5 oct. 2016