

IAPP SCHOOL - VHDL DESIGN: CHIPSCOPE AND DEBUGGING

DANIEL MAGALOTTI



SUMMARY

➤ Introduction of debugging problem

- Testing and debugging
- Why debugging is important?

➤ Description of ChipScope™ Pro software

- Minimal impact to FPGA design
- Optimized cores consume minimal FPGA resources

➤ How to add ChipScope Pro software into design

➤ Describe the ChipScope Pro cores and how to allow you to focus on solving problems

- Integrated Logic Analyzer (ILA) for viewing results
- IBERT for high speed serial link

TEST & DEBUG

➤ Do you know the difference between **testing** and **debugging** problems?

TEST & DEBUG

- Do you know the difference between **testing** and **debugging** problems?
- **In testing**, the goal is to determine as quickly as possible whether the chip is working correctly, with high, but not absolute, certainty.

TEST & DEBUG

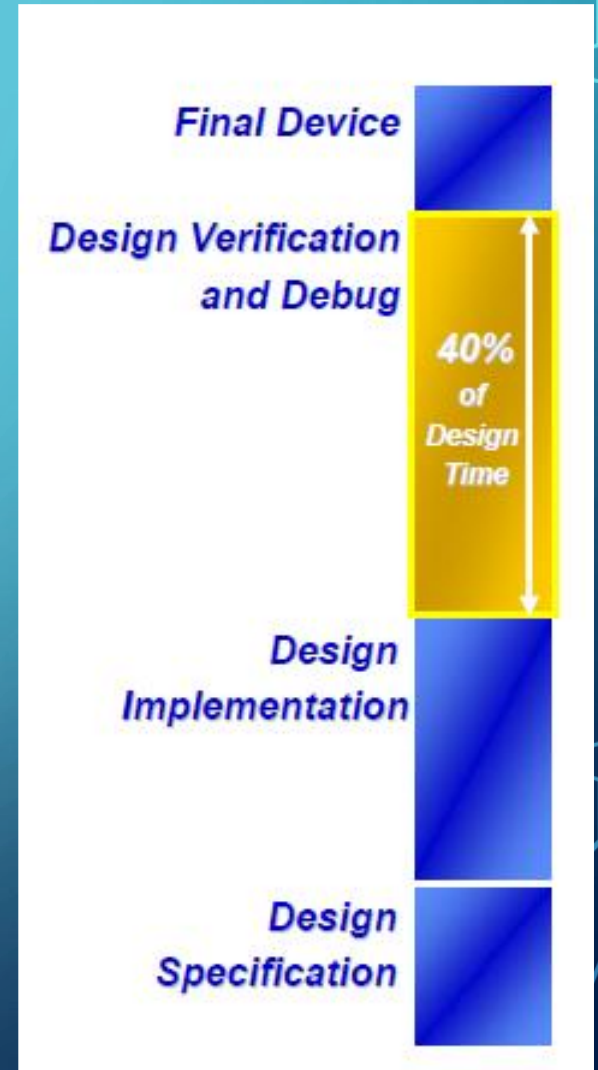
- Do you know the difference between **testing** and **debugging** problems?
- **In testing**, the goal is to determine as quickly as possible whether the chip is working correctly, with high, but not absolute, certainty.
- **In debugging**, the goal is not simply to determine that the chip is not working, but to find out why it is not working. It is not automatic, but requires the participation of the chip-design team. And it occurs at discrete points in the design cycle.

WHY DEBUGGING IS IMPORTANT?

- FPGA designs are becoming more complex
 - Designs are becoming faster
 - Design times are becoming shorter
- Debugging and verification are more challenging
 - Debugging and verification consume a significant portion of FPGA design time
 - An FPGA design survey conducted by Xilinx indicates that FPGA debugging and verification accounts for nearly 40% of the FPGA design time
 - Debugging and verification need to be easier and integrated into the FPGA design flow

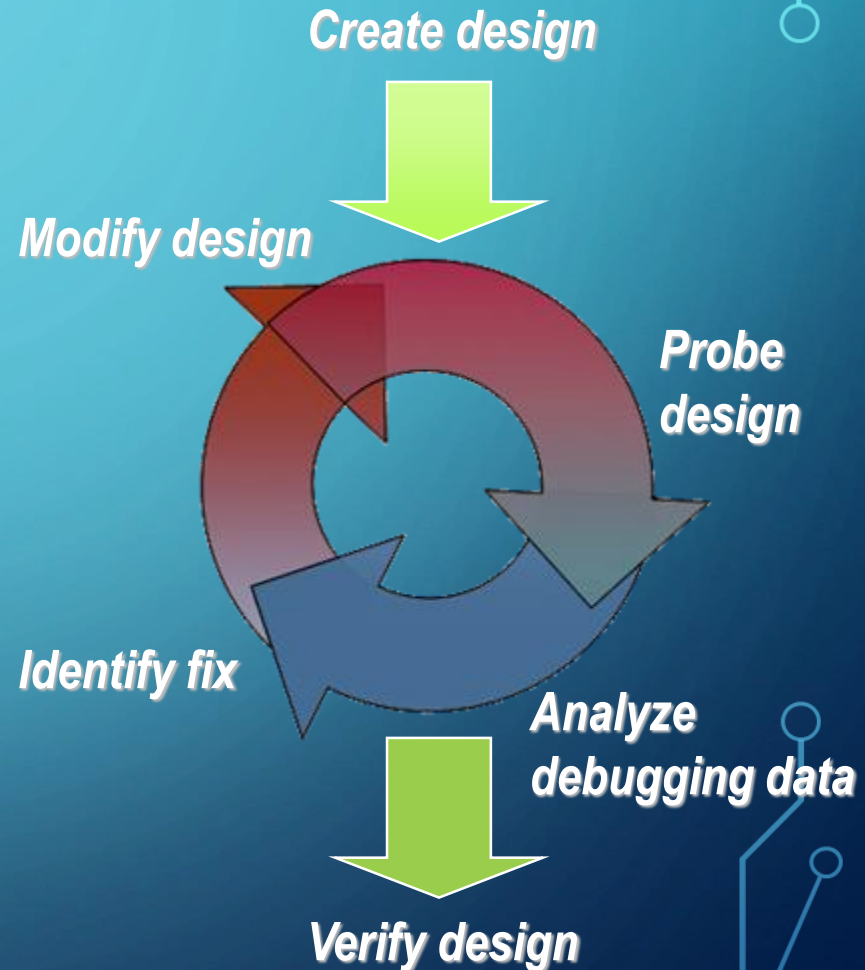
DEBUG AND VERIFICATION IS CRITICAL

- Debug and verification can account for over 40% of an FPGA design time
- Serial nature of debug and verification can make it difficult to optimize
- Inefficient strategy may result in product launch delay
 - Loss in market share
 - Loss of first-to-market advantages



LOGIC OF DEBUGGING

- Debugging is problem solving
 - Break a problem into basic parts
 - Remove or reduce variables and variation
 - Predict and verify
- Debugging is an iterative process
- Verification is a component of debugging
 - Confirming no problems remain



TRADITIONAL DEBUG CHALLENGES

- In order to understand typical debugging problem I show you a “RANDOM” board
- Who guess? I give you three options

LAMB

AMB

AM

TRADITIONAL DEBUG CHALLENGES

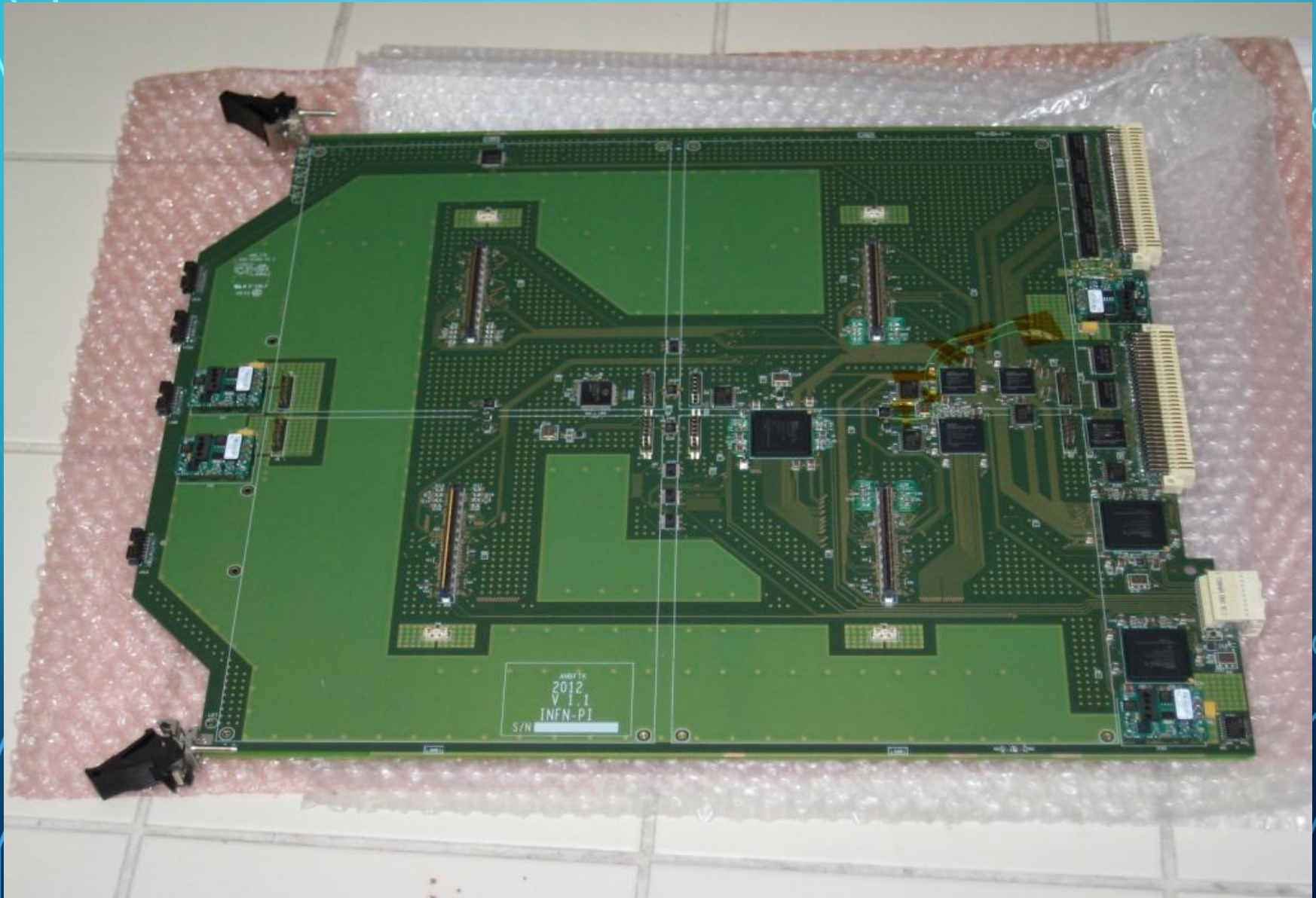
- In order to understand typical debugging problem I show you a “RANDOM” board
- Who guess? I give you three options

LAMB

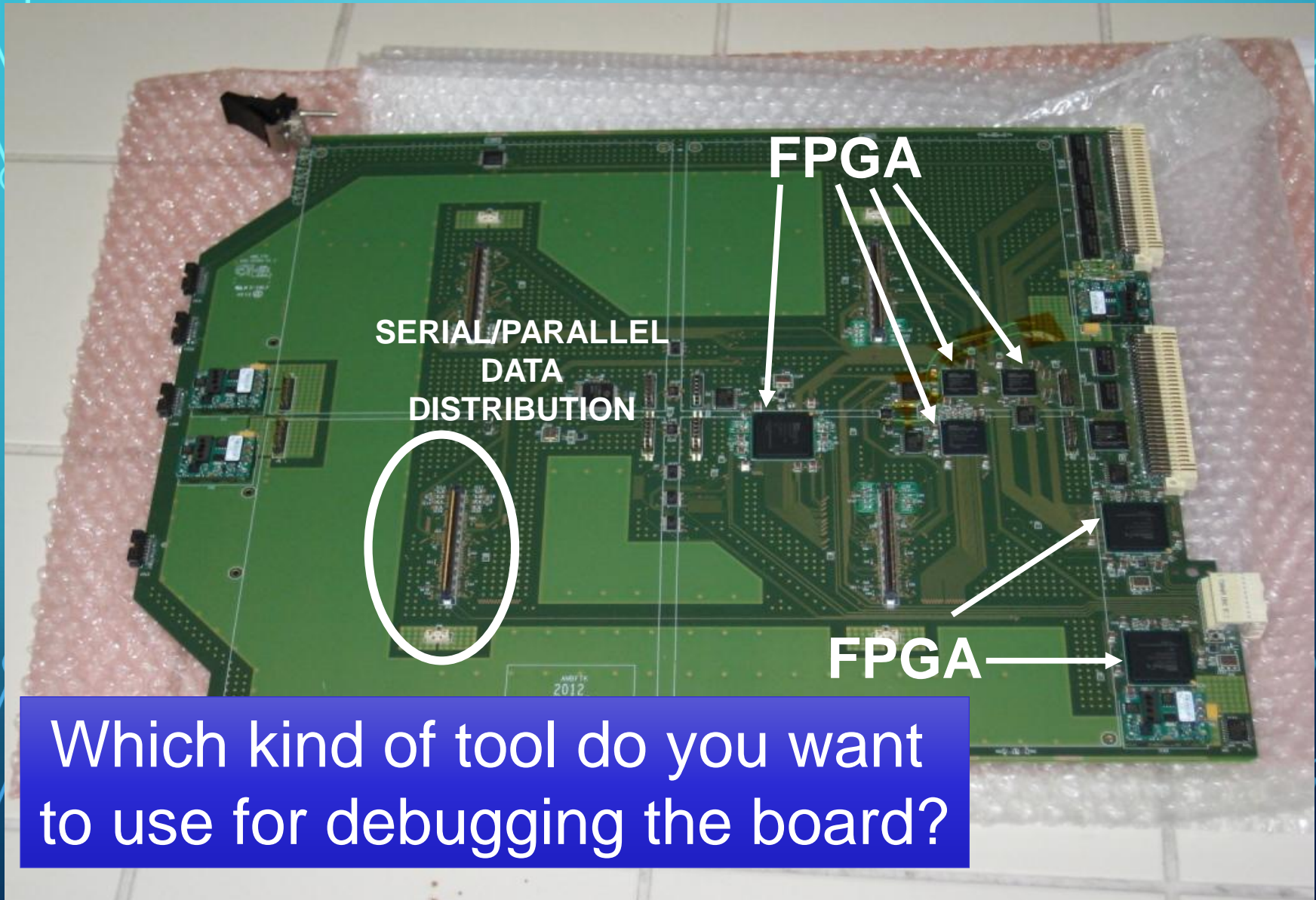
AMB

AM

TRADITIONAL DEBUG CHALLENGES

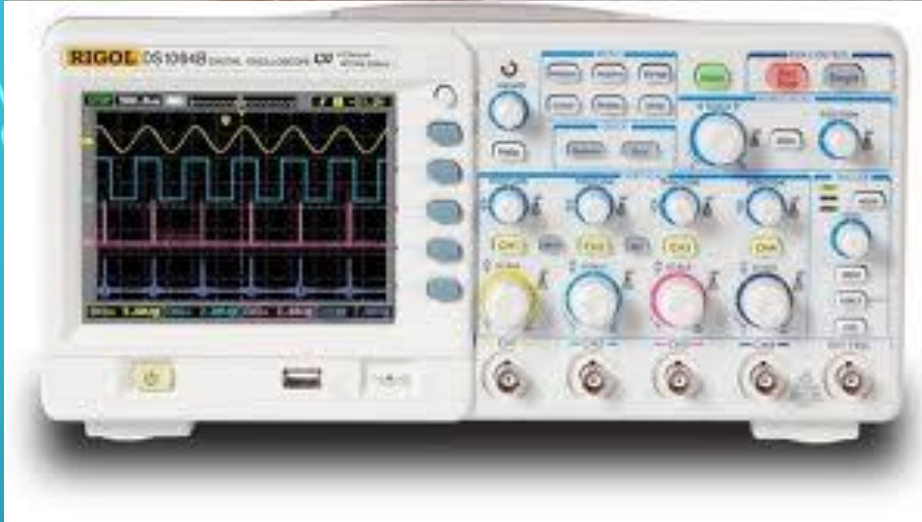


TRADITIONAL DEBUG CHALLENGES

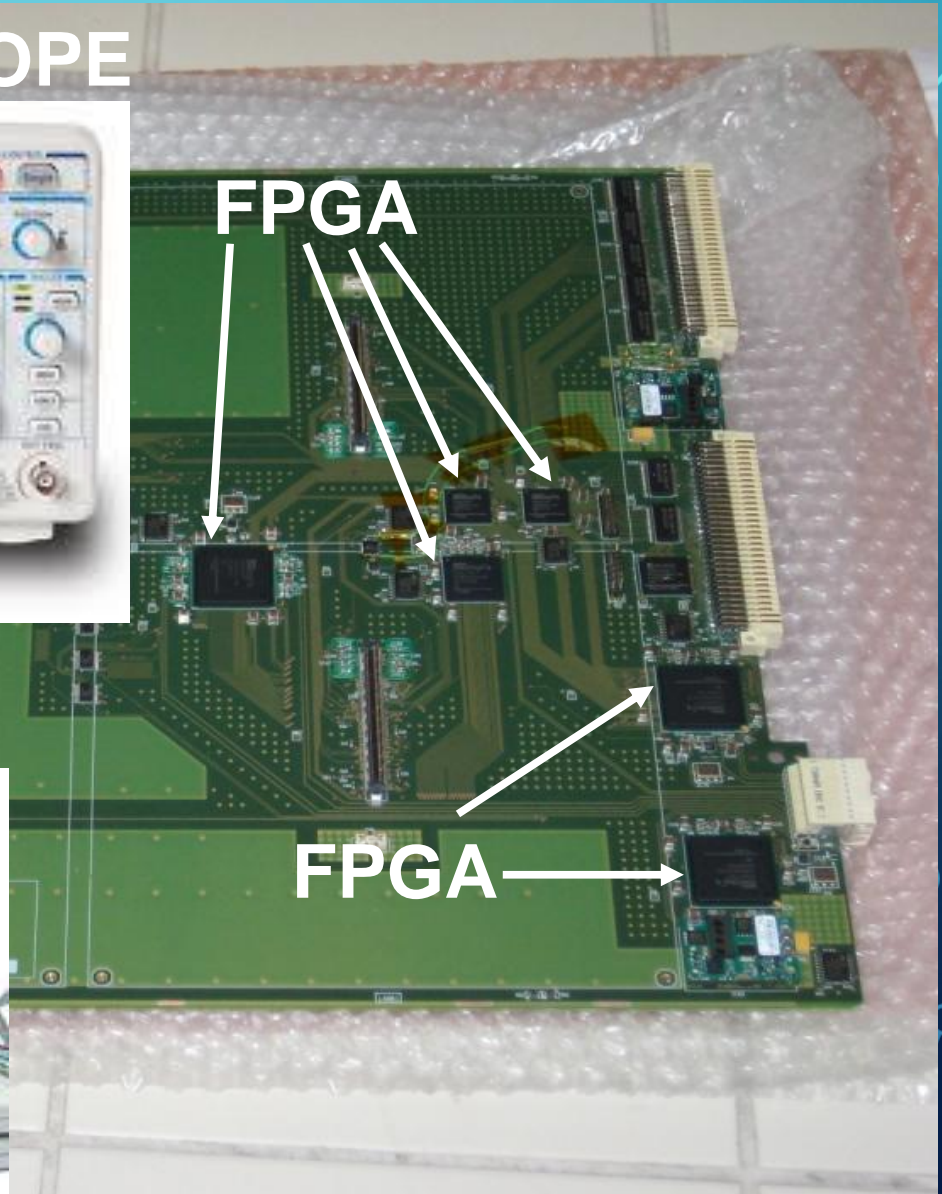


TRADITIONAL DEBUG CHALLENGES

DIGITAL OSCILLOSCOPE



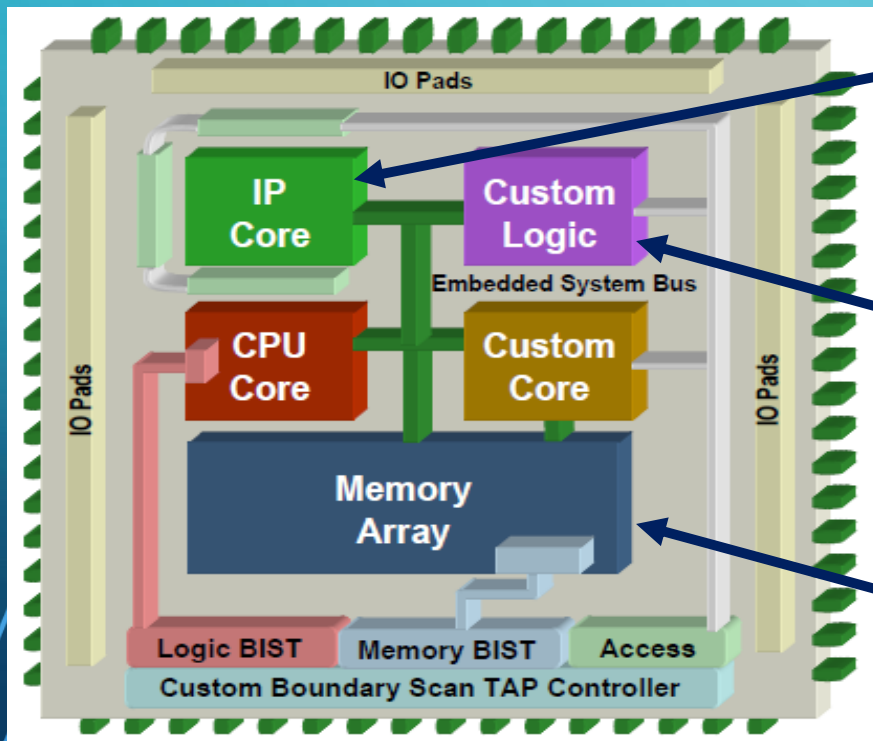
LOGIC ANALYZER



TRADITIONAL DEBUG CHALLENGES

➤ We focalize to debug a single FPGA

Internal Block of FPGA



FIFO

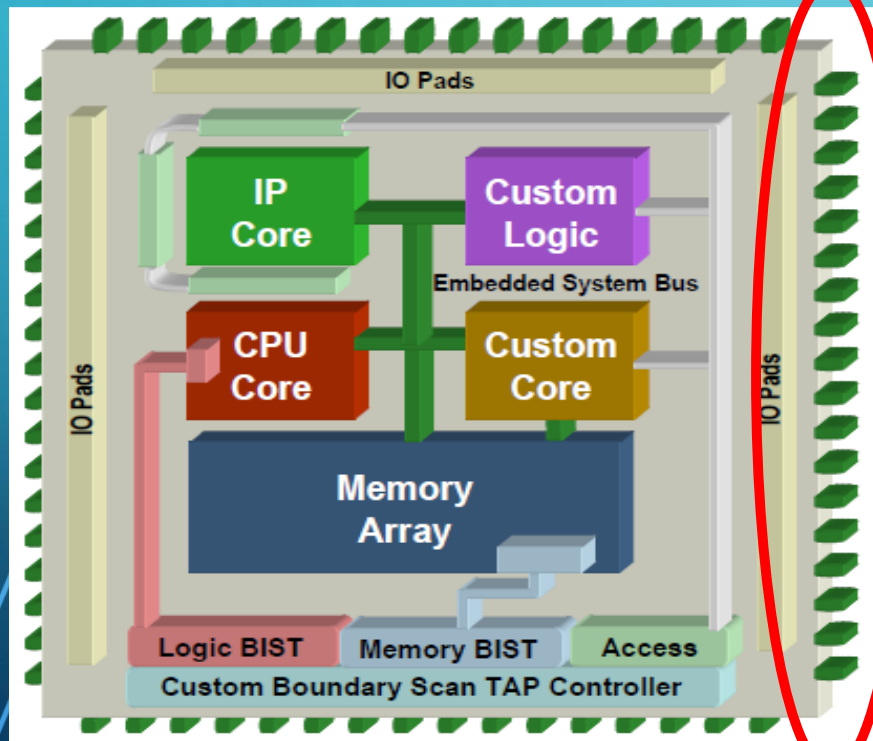
COUNTER or
MULTIPLEXER or
FLIP FLOP D

RAM

TRADITIONAL DEBUG CHALLENGES

➤ We focalize to debug a single FPGA

Internal Block of FPGA



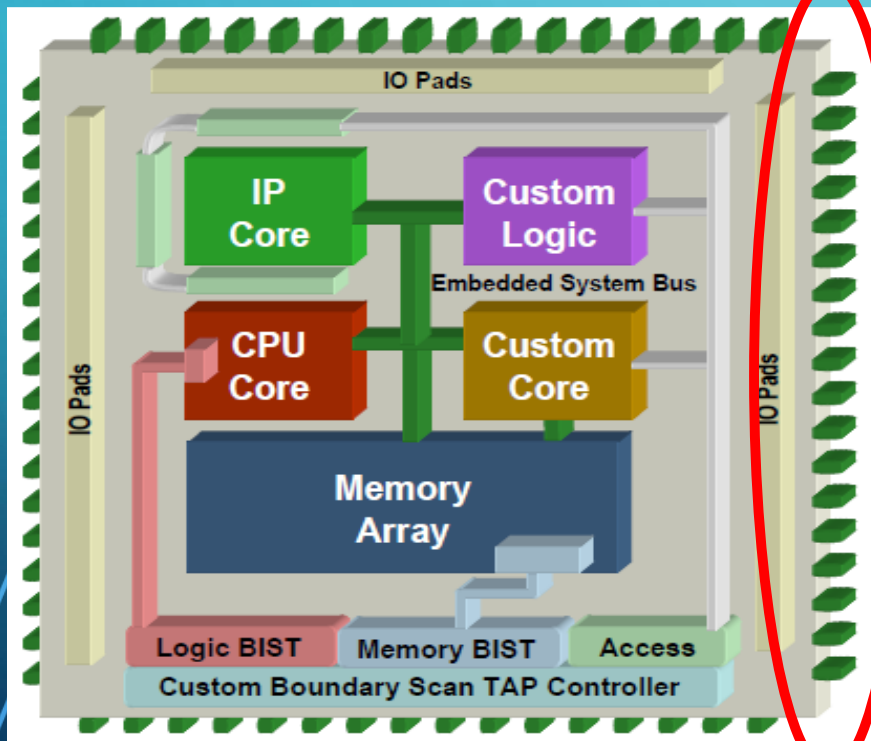
The point that you can use to access to you FPGA is the **IO pin**



TRADITIONAL DEBUG CHALLENGES

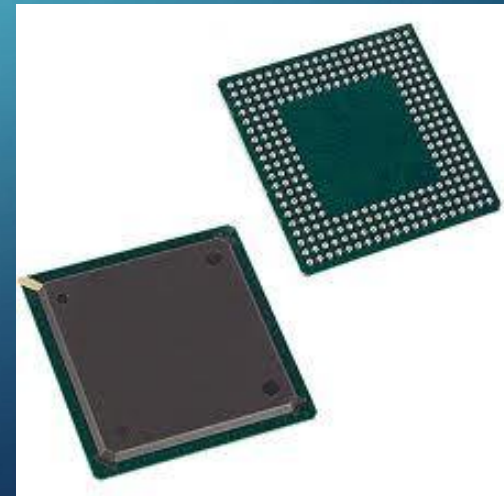
➤ We focalize to debug a single FPGA

Internal Block of FPGA



The point that you can use to access to you FPGA is the **IO pin**

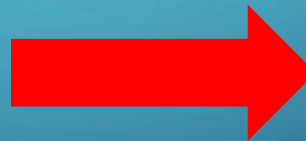
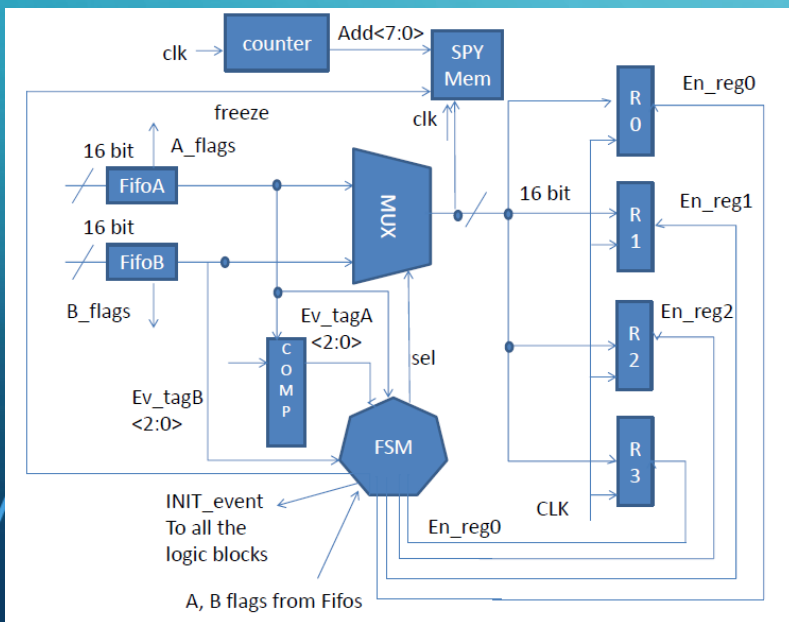
A typical package for the FPGA is a **BGA**



STEP OF DEBUGGING

➤ Which the step in which we can divide the design of the FPGA

➤ 1° STEP: definition of the specification and architecture design by VHDL code



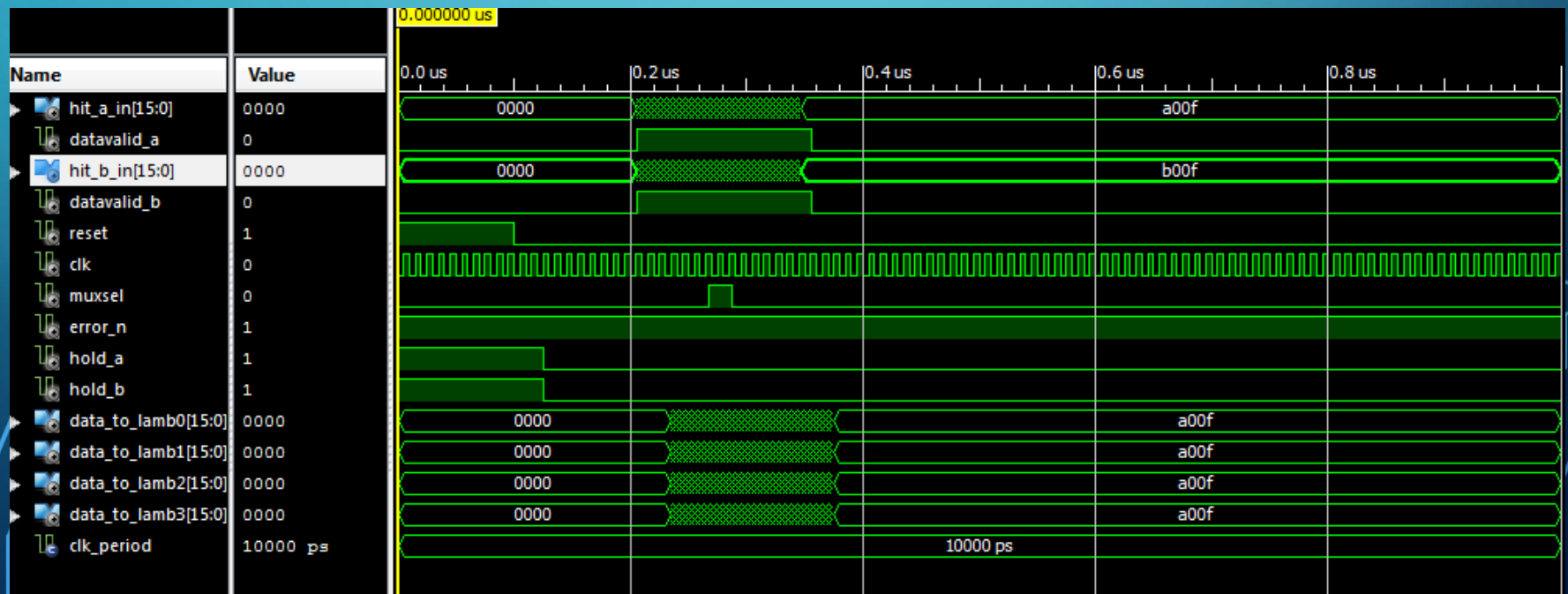
```
40 --library UNISIM;
41 --use UNISIM.VComponents.all;
42
43 entity babyFTK_TOP_v2 is
44     Port ( HIT_A_IN : in  STD_LOGIC_VECTOR (15 downto 0);
45           DataValid_A : in  STD_LOGIC;
46
47           HIT_B_IN : in  STD_LOGIC_VECTOR (15 downto 0);
48           DataValid_B : in  STD_LOGIC;
49
50           RESET : in  STD_LOGIC;
51           CLK : in  STD_LOGIC;
52           MUXSEL : in  STD_LOGIC;
53
54           ERROR_n : out  STD_LOGIC;
55           HOLD_A : out  STD_LOGIC;
56           HOLD_B : out  STD_LOGIC;
57           DATA_TO_LAMB0 : out  STD_LOGIC_VECTOR (15 downto 0);
58           DATA_TO_LAMB1 : out  STD_LOGIC_VECTOR (15 downto 0);
59           DATA_TO_LAMB2 : out  STD_LOGIC_VECTOR (15 downto 0);
60           DATA_TO_LAMB3 : out  STD_LOGIC_VECTOR (15 downto 0)
61     );
62 end babyFTK_TOP_v2;
63
64 architecture Behavioral of babyFTK_TOP_v2 is
65
66     ----- components declaration -----
67
68     component COMP_EQUAL is
69         Port ( INPUT1 : in  STD_LOGIC_VECTOR (7 downto 0);
70               INPUT2 : in  STD_LOGIC_VECTOR (7 downto 0);
71               EQUAL : out  STD_LOGIC
72         );
73     end component;
```

Design Summary

STEP OF DEBUGGING

➤ Which the step in which we can divide the design of the FPGA

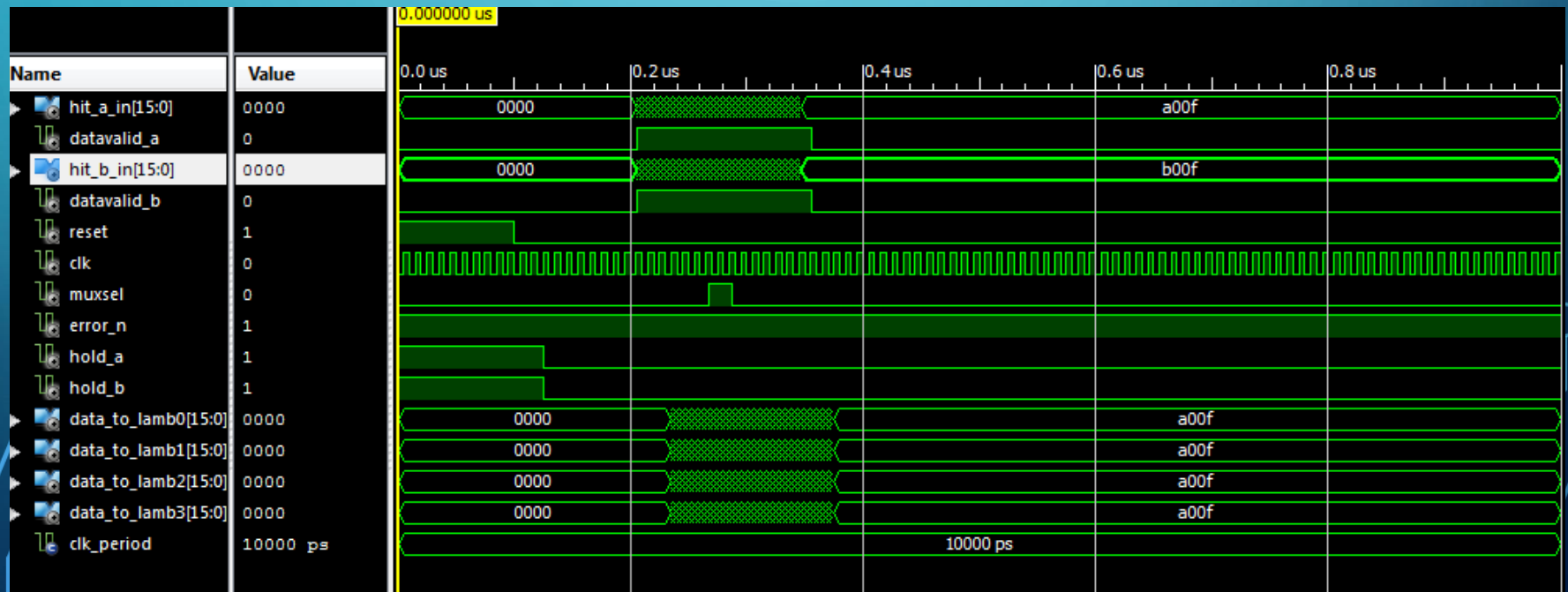
➤ 2° STEP: synthetize the code and perform a behavioral simulation of your code



STEP OF DEBUGGING

➤ Which the step in which we can divide the design of the FPGA

➤ 3° STEP: implementation of your code and timing simulation of your code

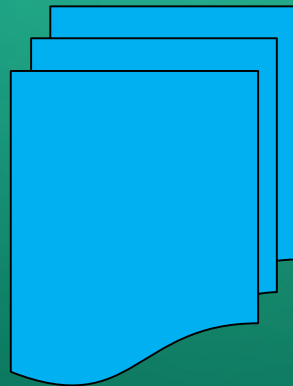


STEP OF DEBUGGING

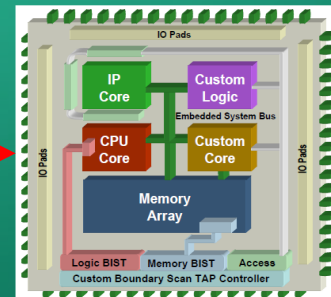
- Which the step in which we can divide the design of the FPGA
- 4° STEP: modify you testbench file in order to consider all the possible input stimulus to your design

TESTBENCH Module

File 1,2,3...



UUT1



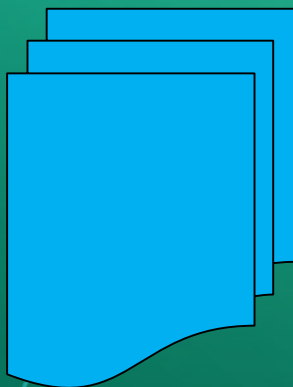
TRADITIONAL DEBUG CHALLENGES

➤ Which the step in which we can divide the design of the FPGA

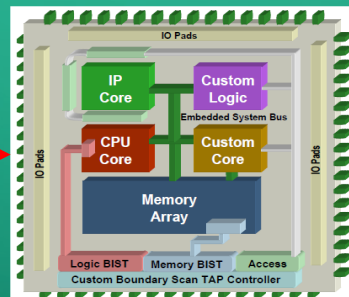
➤ 5° STEP: insert more than a FPGA into testbench to simulate the protocol between them

TESTBENCH Module

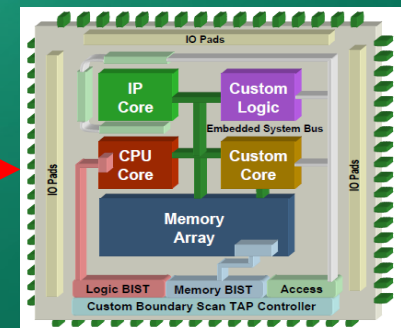
File 1,2,3...



UUT1



UUT2



STEP OF DEBUGGING

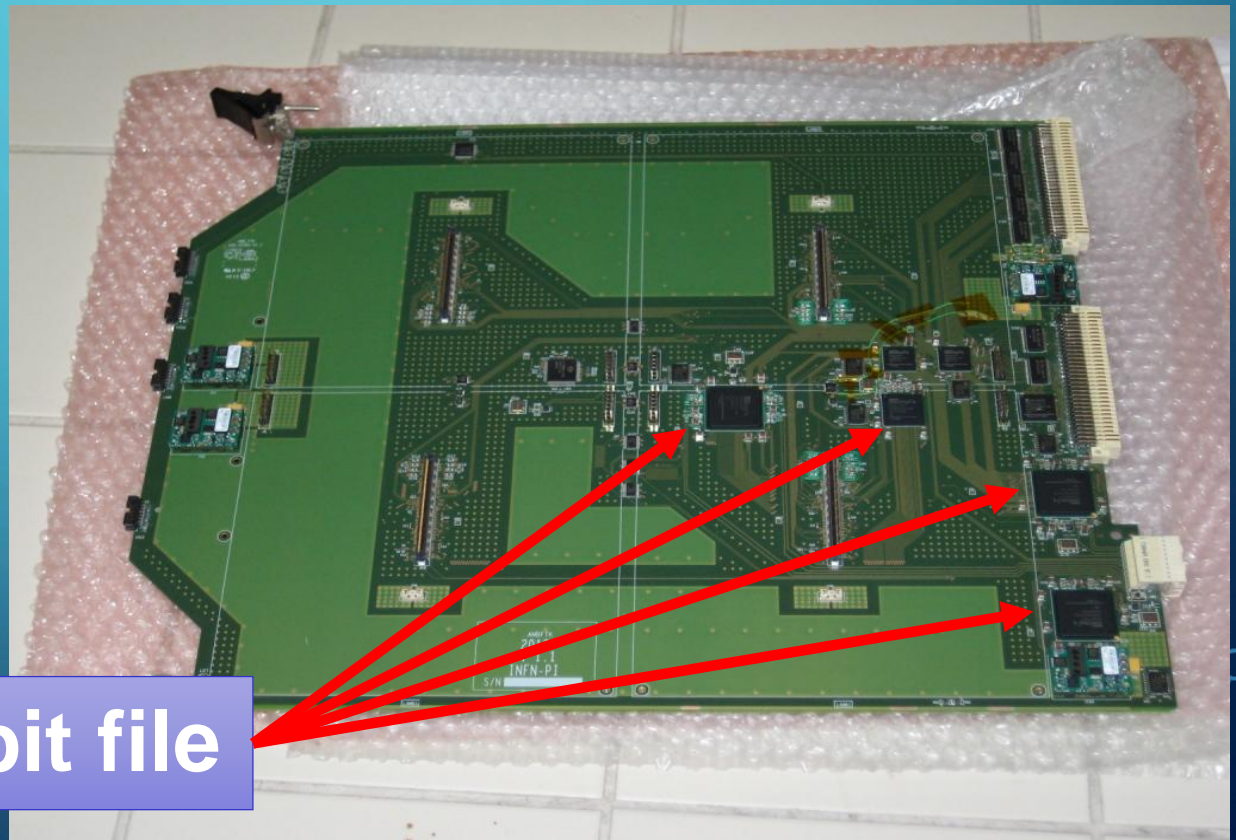
- Which the step in which we can divide the design of the FPGA
 - 1° STEP: definition of the specification and architecture design by VHDL code
 - 2° STEP: synthesize the code and perform a behavioral simulation of your code
 - 3° STEP: implementation of your code and timing simulation of your code
 - 4° STEP: modify your testbench file in order to consider all the possible input stimulus to your design
 - 5° STEP: insert more than a FPGA into testbench to simulate the protocol between them

STEP OF DEBUGGING

➤ Which the step in which we can divide the design of the FPGA

➤ 6° STEP: Programming the FPGA

```
40 --library UNISIM;  
41 --use UNISIM.VComponents.all;  
42  
43 entity babyFTK_TOP_v2 is  
44   Port ( HIT_A_IN : in  STD_LOGIC_VECTOR (15 downto 0);  
45         DataValid_A : in  STD_LOGIC;  
46  
47         HIT_B_IN : in  STD_LOGIC_VECTOR (15 downto 0);  
48         DataValid_B : in  STD_LOGIC;  
49  
50         RESET : in  STD_LOGIC;  
51         CLK : in  STD_LOGIC;  
52         MUXSEL : in  STD_LOGIC;  
53  
54         ERROR_n : out  STD_LOGIC;  
55         HOLD_A : out  STD_LOGIC;  
56         HOLD_B : out  STD_LOGIC;  
57         DATA_TO_LAMB0 : out  STD_LOGIC_VECTOR (15 downto 0);  
58         DATA_TO_LAMB1 : out  STD_LOGIC_VECTOR (15 downto 0);  
59         DATA_TO_LAMB2 : out  STD_LOGIC_VECTOR (15 downto 0);  
60         DATA_TO_LAMB3 : out  STD_LOGIC_VECTOR (15 downto 0);  
61   );  
62 end babyFTK_TOP_v2;  
63  
64 architecture Behavioral of babyFTK_TOP_v2 is  
65  
66   ----- components declaration -----  
67  
68   component COMP_EQUAL is  
69     Port ( INPUT1 : in  STD_LOGIC_VECTOR (7 downto 0);  
70           INPUT2 : in  STD_LOGIC_VECTOR (7 downto 0);  
71           EQUAL : out  STD_LOGIC  
72   );  
73 end component;
```



.bit file

STEP OF DEBUGGING

➤ Which the step in which we can divide the design of the FPGA

➤ 6° STEP: Programming the FPGA

New parameter into the real design that you cannot considering into simulation

- Error in the board design
- Temperature variation
- ...

```
40 --library UNISIM;
41 --use UNISIM.VComponents.all;
42
43 entity babyFTK_TOP_v2 is
44   Port ( HIT_A_IN : in  STD_LOGIC_VECTOR (15 downto 0);
45         DataValid_A : in  STD_LOGIC;
46
47         HIT_B_IN : in  STD_LOGIC_VECTOR (15 downto 0);
48         DataValid_B : in  STD_LOGIC;
49
50         RESET : in  STD_LOGIC;
51         CLK : in  STD_LOGIC;
52         MUXSEL : in  STD_LOGIC;
53
54         ERROR_n : out  STD_LOGIC;
55         HOLD_A : out  STD_LOGIC;
56         HOLD_B : out  STD_LOGIC;
57         DATA_TO_LAMBDA : out  STD_LOGIC_VECTOR (15 downto 0);
58         DATA_TO_LAMBDA1 : out  STD_LOGIC_VECTOR (15 downto 0);
59         DATA_TO_LAMBDA2 : out  STD_LOGIC_VECTOR (15 downto 0);
60         DATA_TO_LAMBDA3 : out  STD_LOGIC_VECTOR (15 downto 0);
61   );
62 end entity babyFTK_TOP_v2;
63
64 architecture Behavioral of babyFTK_TOP_v2 is
65
66   ----- components declaration -----
67
68   component COMP_EQUAL is
69     Port ( INFUT1 : in  STD_LOGIC_VECTOR (7 downto 0);
70           INFUT2 : in  STD_LOGIC_VECTOR (7 downto 0);
71           EQUAL : out  STD_LOGIC
72   );
73 end component COMP_EQUAL;
```

.bit file



TRADITIONAL LOGIC ANALYSIS METHOD

➤ Requires Extensive Dedicated I/O for Debug

- Driving signals to **external I/O** introduces additional problems

➤ Inflexible solution

- Difficult or **impossible to add additional debug pins** if needed

➤ **Limited** visibility to on-chip activity

Dedicated pins connected to logic analyzer

Virtex-II Pro

XC2VP20
FF1152

Probe
points

Pins

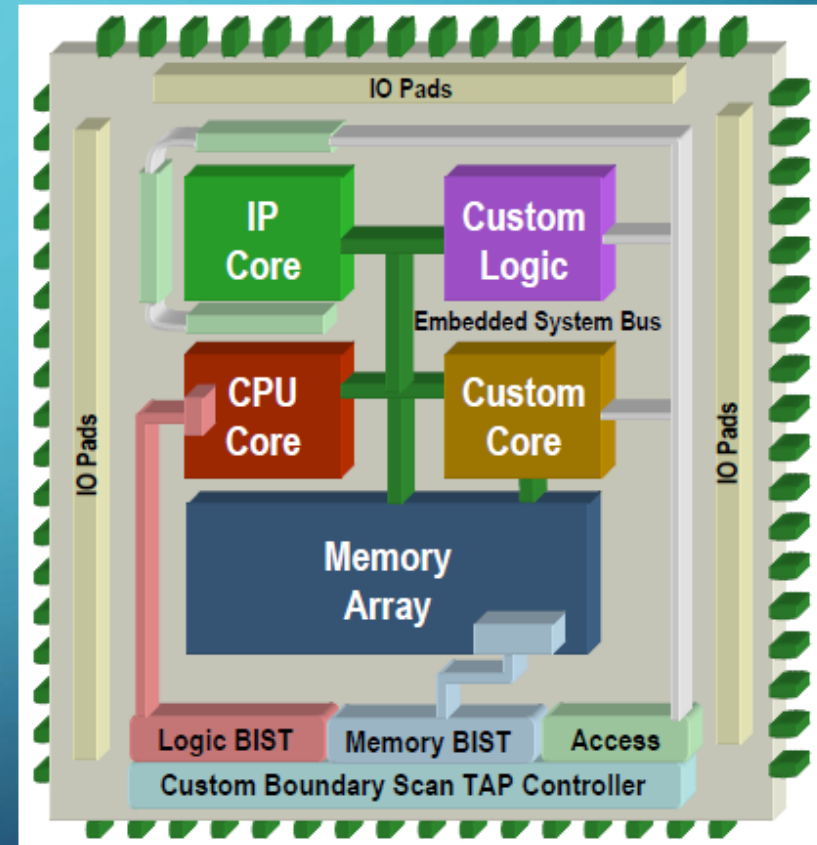
External Logic
Analyzer



WHAT DO WE WANT

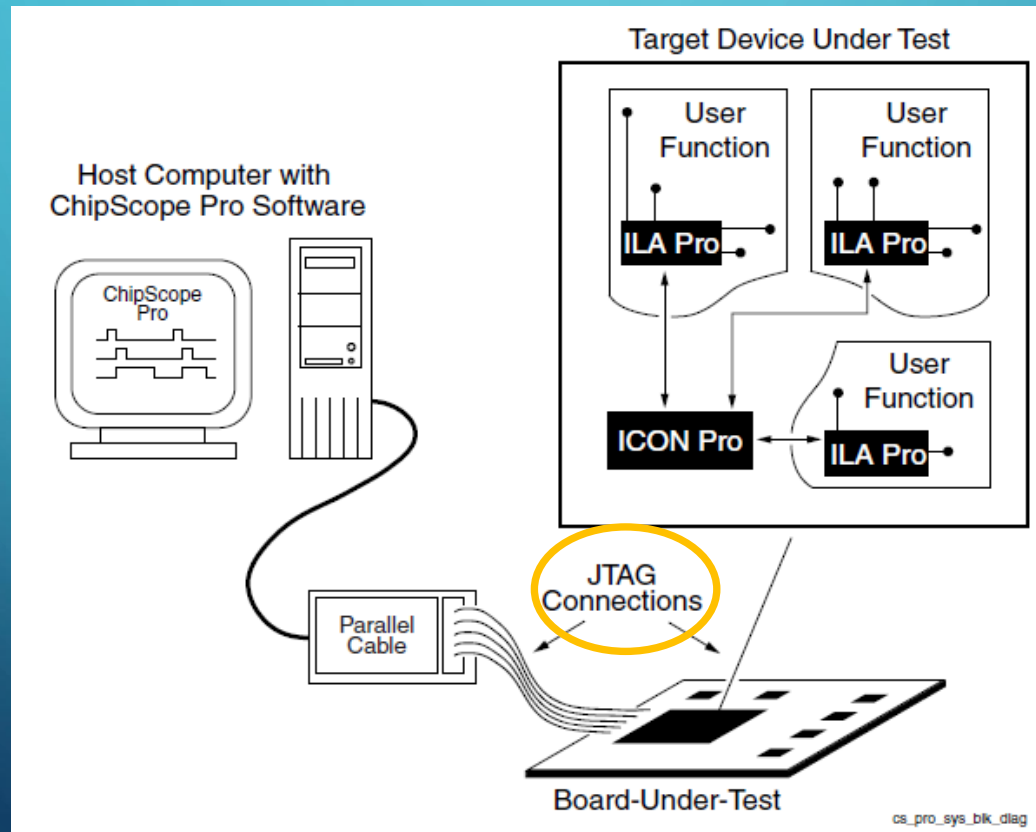
- FPGA Enables **Full Internal Visibility**
 - Complete on-chip access
- Access Processor System Busses
 - Integrated Bus Analyzer
- **Flexible On-Chip Debug**
 - Small, efficient cores access any node or signal and can be removed at any time
- Enable Complete **System Verification**
 - Debug systems in real-time

Internal Block of FPGA



CHIPSCOPE PRO ON-CHIP DEBUG

- No I/O pins required for debug
 - Access via the **JTAG Port**
- On-Chip access to **every signal and node** in the FPGA design
- Add and remove cores at any time in the design process



CHIPSCOPE PRO ON-CHIP DEBUG

- Use the ChipScope Pro software for
 - **Verification and debug**
 - Injecting short signal sequences
 - Capturing data for **post-bench analysis**
- Do not use the ChipScope Pro software for
 - **A replacement for a simulation tool**
 - Accessing the System Monitor
 - **Testing high-speed I/O**
 - Remote diagnostics/monitoring



OPTIMIZED DEBUGGING CORES

➤ There are 5 CORE integrated into ChipScope

Virtual Input Output Core (VIO)

- Virtual Inputs and Outputs
- Stimulate logic with pulse trains

Integrated Bus Analysis Core (IBA)

- PLB and OPB specific Bus analysis cores
- Protocol detection
- Debug and verify control, address, and data buses

Integrated Bit Error Ratio (IBERT)

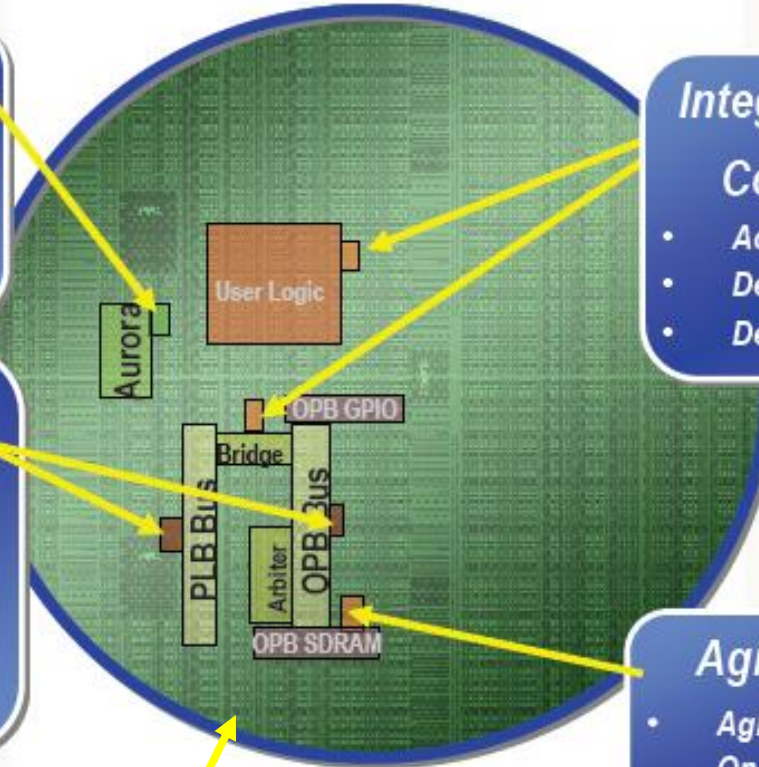
- monitor the functionality of transceivers (GTP, GTH, GTX)

Integrated Logic Analysis Core (ILA)

- Access internal nodes and signals
- Debug and verify signal behavior
- Define detailed trigger conditions

Agilent Trace Core 2 (ATC2)

- Agilent created core enabling On-chip debug of Xilinx FPGAs using Agilent FPGA Dynamic Probing




CORE RESOURCES

➤ ChipScope™ Pro software cores utilize FPGA resources

- For what?
 - **Block RAM:** trigger and data storage
 - **Slice logic:** trigger comparisons

➤ You must leave room for the ChipScope Pro software cores in the FPGA

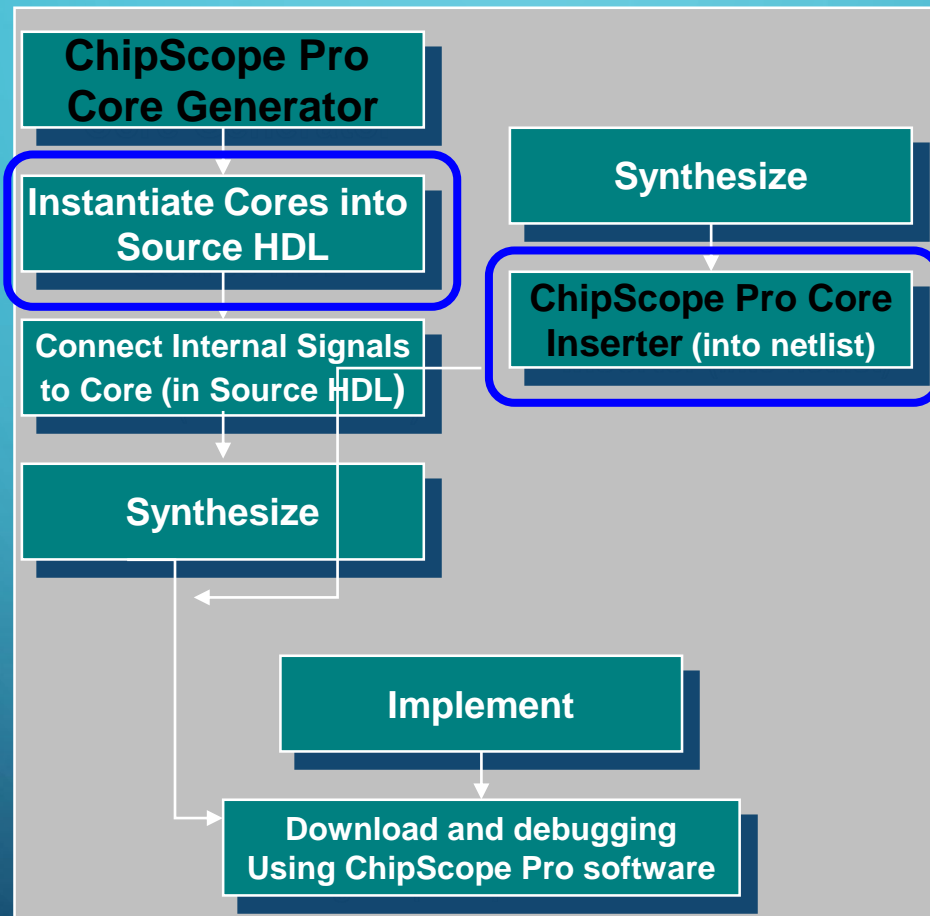
- This may require using a larger part in the same package as you will use in production
- The CORE Generator and Core Inserter tools can estimate block RAM usage, but the design may still end up with too many block RAMs
- If MAP issues an error, reduce the number of observed signals or the sample data depth to reduce block RAM usage

A decorative graphic on the left side of the slide consisting of white lines and circles on a blue gradient background, resembling a circuit board or a stylized tree structure.

ADD CHIPSCOPE IN THE DESIGN

USING CHIPSCOPE PRO SOFTWARE

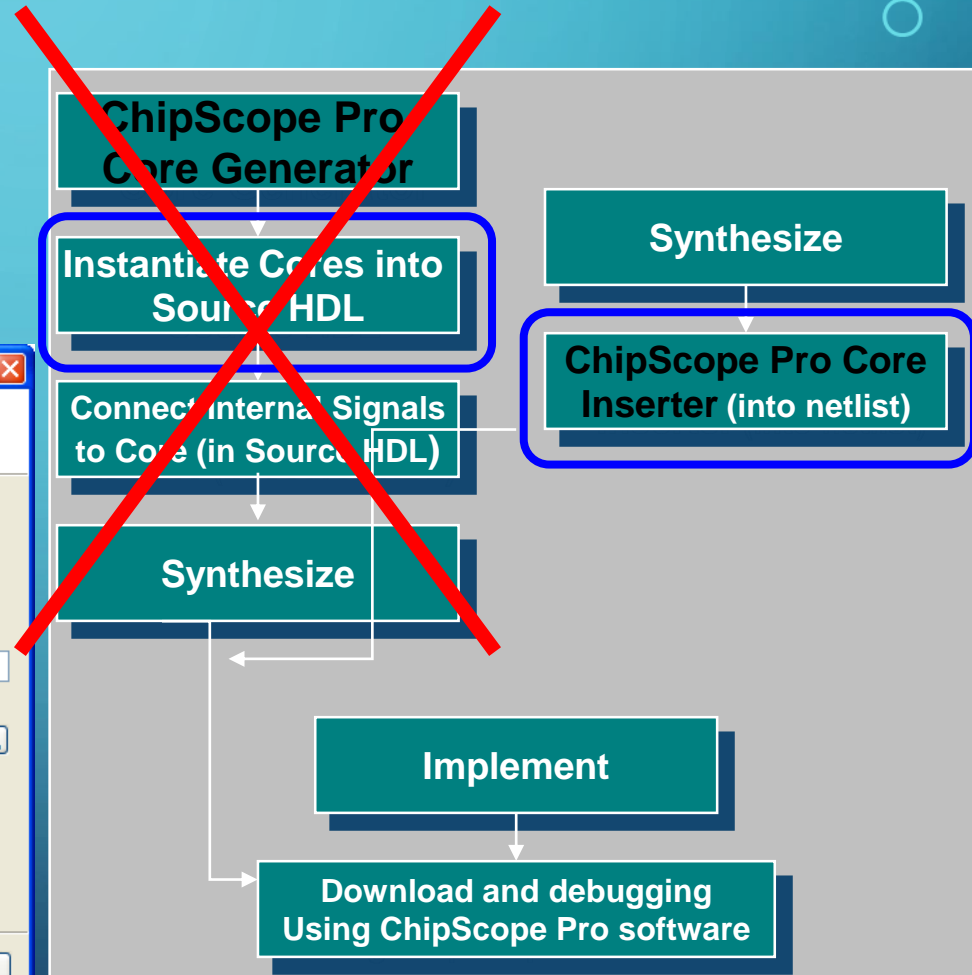
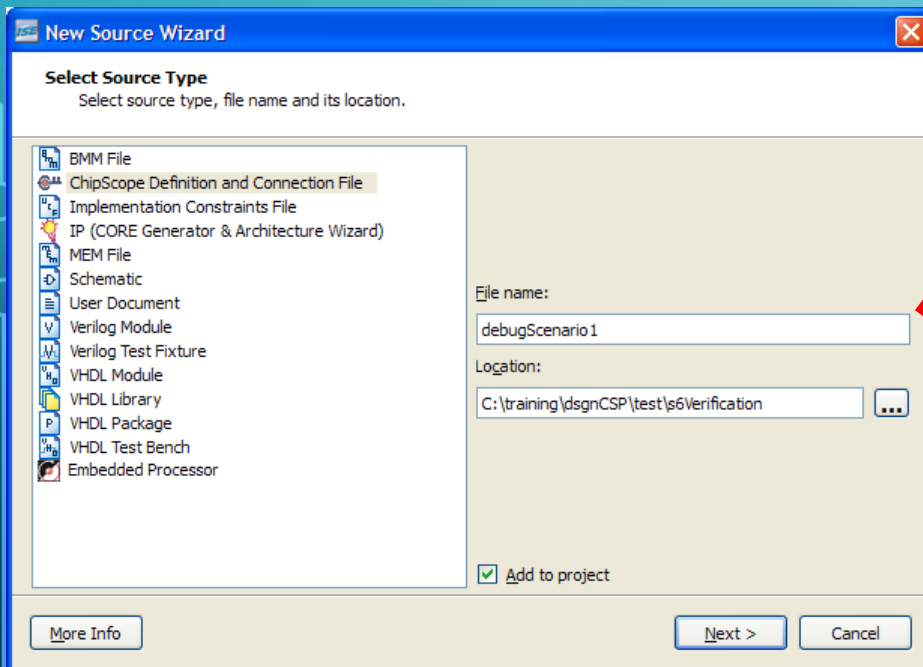
- Two way to insert the ChipScope: into **HDL source** or with **Core Insert**



USING CHIPSCOPE PRO SOFTWARE

- Two way to insert the ChipScope: into **HDL source** or with **Core Insert**
- **Core Insert**

- Click Project -> New Source
- Select ChipScope Definition and Connection File (CDC)

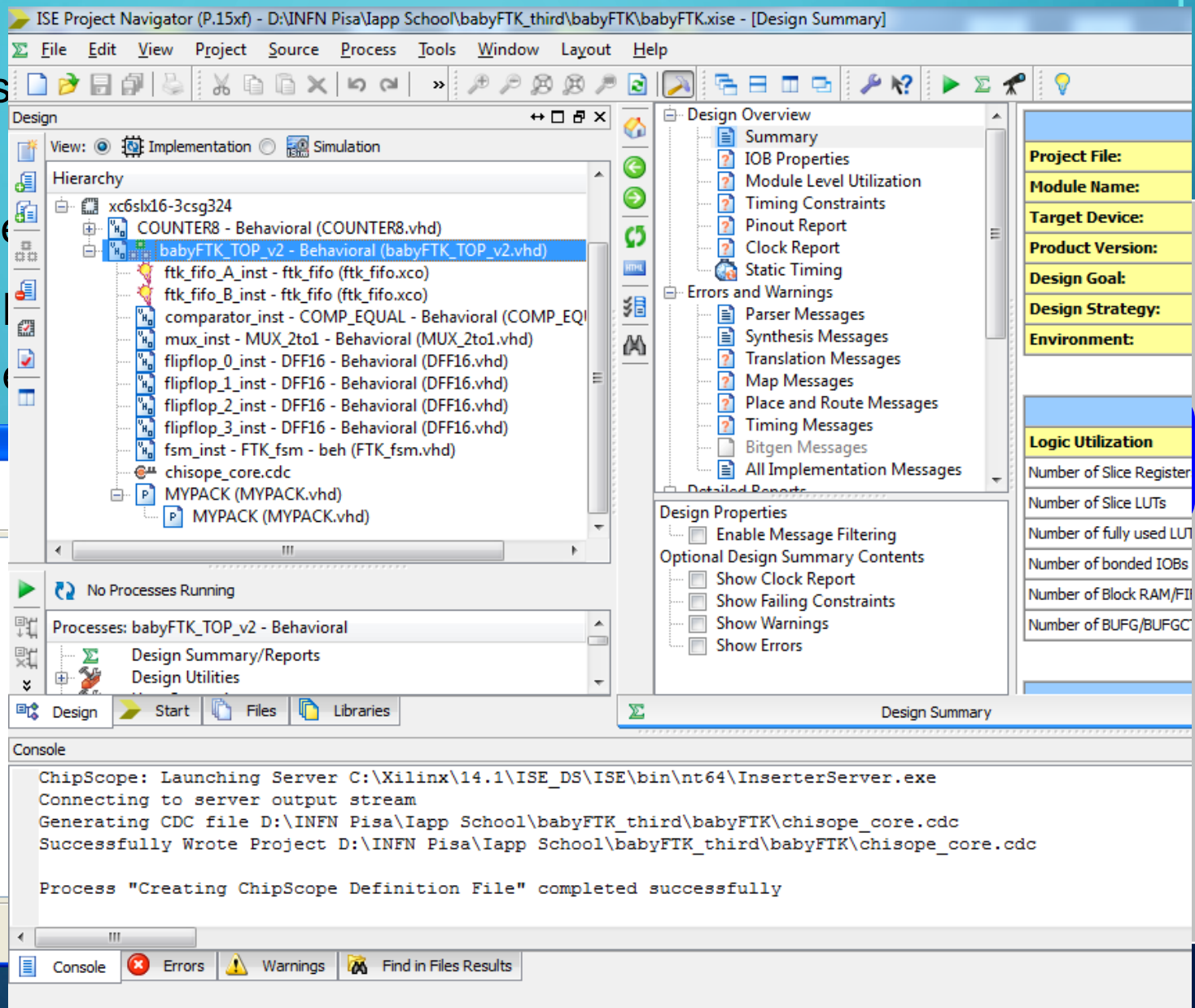
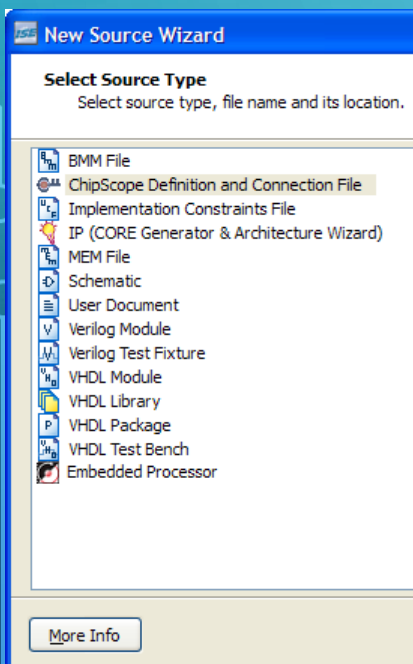


USING CHIPSCOPE PRO SOFTWARE

➤ Two way to insert

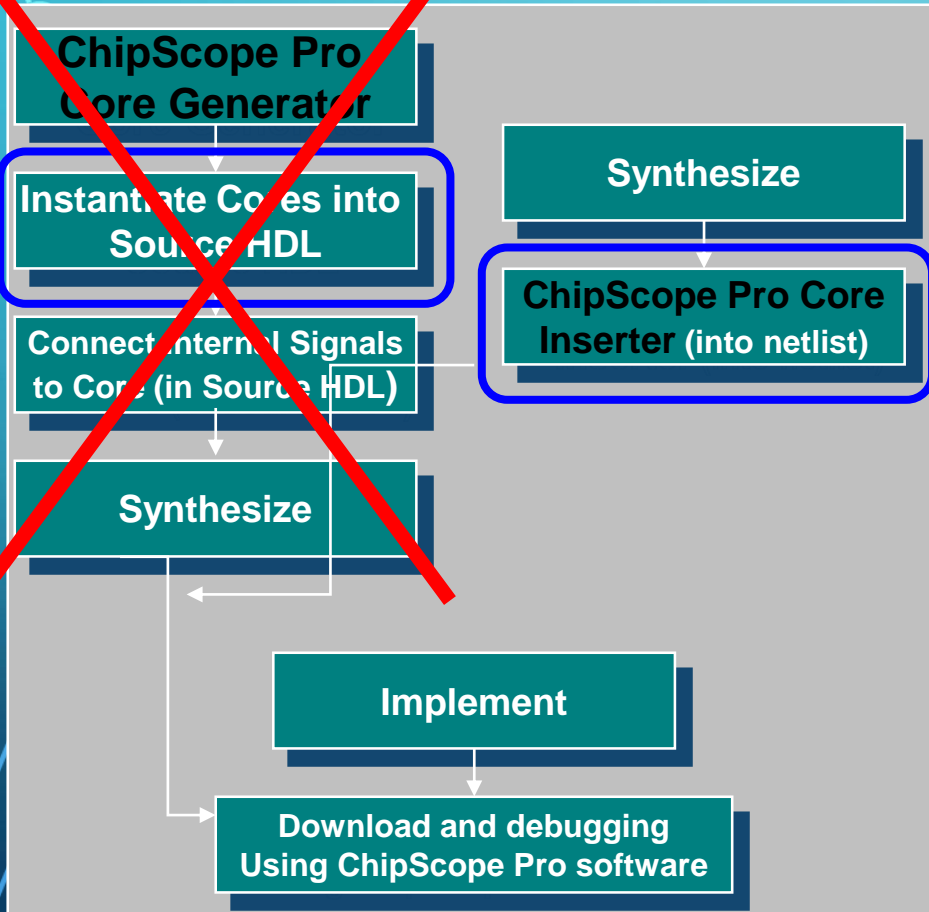
➤ Core Insert

- Click Project
- Select ChipScope Definition and Connection File
- and Connect



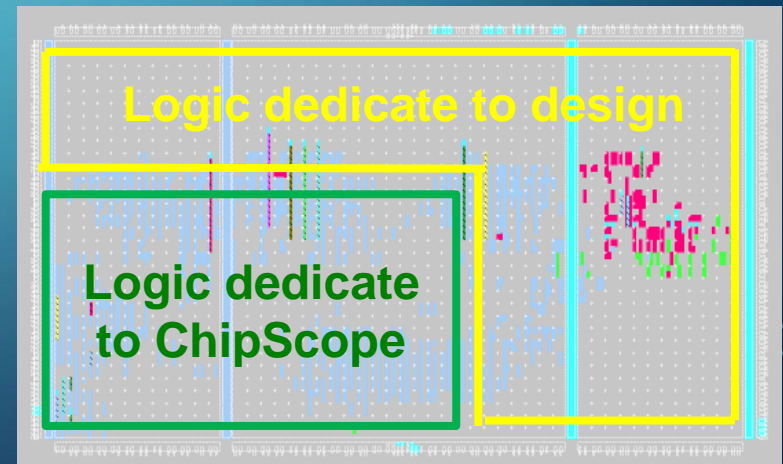
USING CHIPSCOPE PRO SOFTWARE

- Two way to insert the ChipScope: into **HDL source** or with **Core Insert**



A source VHDL for the design

A different source for the
ChipScope



THE ICON CORE

- **ICON (Integrated Control) core:** This core controls up to 15 capture cores
 - The ICON core interfaces between the JTAG interface and the capture cores
- **Capture cores:** Customizable cores for creating triggers and data storage
 - Customizable number, width, and storage of trigger ports
 - **ILA (Integrated Logic Analyzer) core:** Capture core for HDL designs
 - **IBERT core:** High speed monitoring
 - ~~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

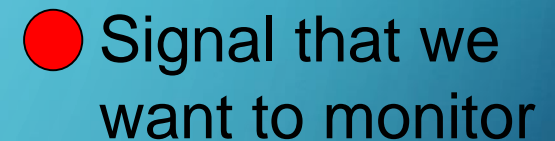
A decorative graphic on the left side of the slide, consisting of a network of white lines and circles on a blue gradient background, resembling a circuit board or a neural network.

ILA CORE

THE ILA CORE

- Integrated Logic Analyzer (ILA) cores can be added with either the CORE Generator or Core Inserter tools or PlanAhead tool
- A design can contain up to 15 ILA cores
- Maximum speed of the ILA core varies according to device family and selected features
 - Turning on more “features” generally slows down the performance of the core and causes it to **consume additional fabric resources**

- Considering the baby FTK project we select the point that we want to monitor the data



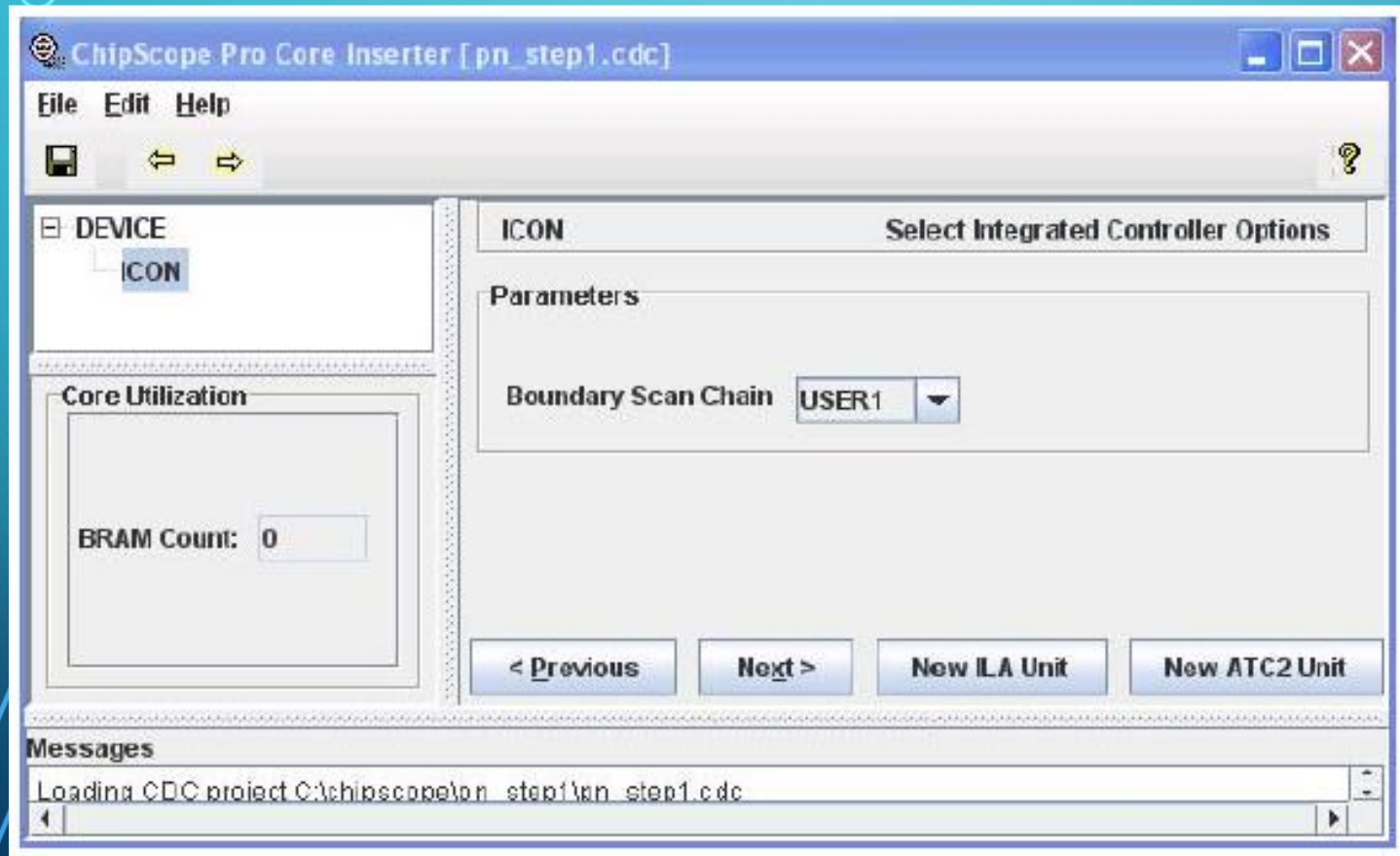
HIT_A_IN

HIT_B_IN

STATE_OF_FSM

ILA CORE

➤ The wizard after the inserting chipscope core



ILA CORE: WIZARD

- The wizard to define the option of the ILA core

The screenshot displays the 'Select Integrated Logic Analyzer Options' wizard for the ILA core. The interface is divided into three main sections: a left-hand pane for device and core selection, and a main right-hand pane for configuration.

Left-hand pane:

- DEVICE:** A tree view showing 'ICON' and 'U0: ILA'.
- Core Utilization:** A section showing 'BRAM Cou...' with a value of '1'.

Main right-hand pane:

- Tabbed Interface:** Three tabs are visible: 'Trigger Parameters' (selected), 'Capture Parameters', and 'Net Connections'.
- Trigger Parameters Section:**
 - Trigger Input and Match Unit Settings:**
 - Number of Input Trigger Ports:** 6
 - Number of Match Units Used:** 6
 - TRIG0:**
 - Trigger Width:** 2
 - # Match Units:** 1
 - Counter Width:** Disabled
 - Match Type:** Basic w/wedges
 - Bit Values:** 0, 1, X, R, F, B
 - Functions:** =, <>
 - TRIG1:** (Similar settings to TRIG0)
 - Trigger Condition Settings:**
 - ☒ **Enable Trigger Sequencer**
 - Max Number of Sequencer Levels:** 16
 - Storage Qualification Condition Settings:**
 - ☒ **Enable Storage Qualification**
- Navigation:** Buttons for '< Previous', 'Next >', and 'Remove Unit'.

ILA CORE: WIZARD

➤ The trigger parameters and capture parameters

The screenshot displays the ILA Core Wizard interface. On the left, a tree view shows the device hierarchy: **DEVICE** > **ICON** > **U0: ILA**. The main panel is titled **ILA** and has three tabs: **Trigger Parameters** (selected), **Capture Parameters**, and **Net Connection**.

Trigger Parameters Tab:

- Trigger Input and Match Unit Settings:**
 - Number of Input Trigger Ports:** 6
 - Number of Match Units Used:** 6
- TRIG0:**
 - Trigger Width:** 2
 - # Match Units:** 1
 - Counter Width:** Disabled
 - Match Type:** Basic wedges
 - Bit Values:** 0, 1, X, R, F, B
 - Functions:** =, <>
- TRIG1:**
 - Trigger Width:** 2
 - # Match Units:** 1
 - Counter Width:** Disabled
 - Match Type:** Basic wedges
 - Bit Values:** 0, 1, X, R, F, B
 - Functions:** =, <>

Trigger Condition Settings:

- ☒ **Enable Trigger Sequencer**
- Max Number of Sequencer Levels:** 16

Storage Qualification Condition Settings:

- ☒ **Enable Storage Qualification**

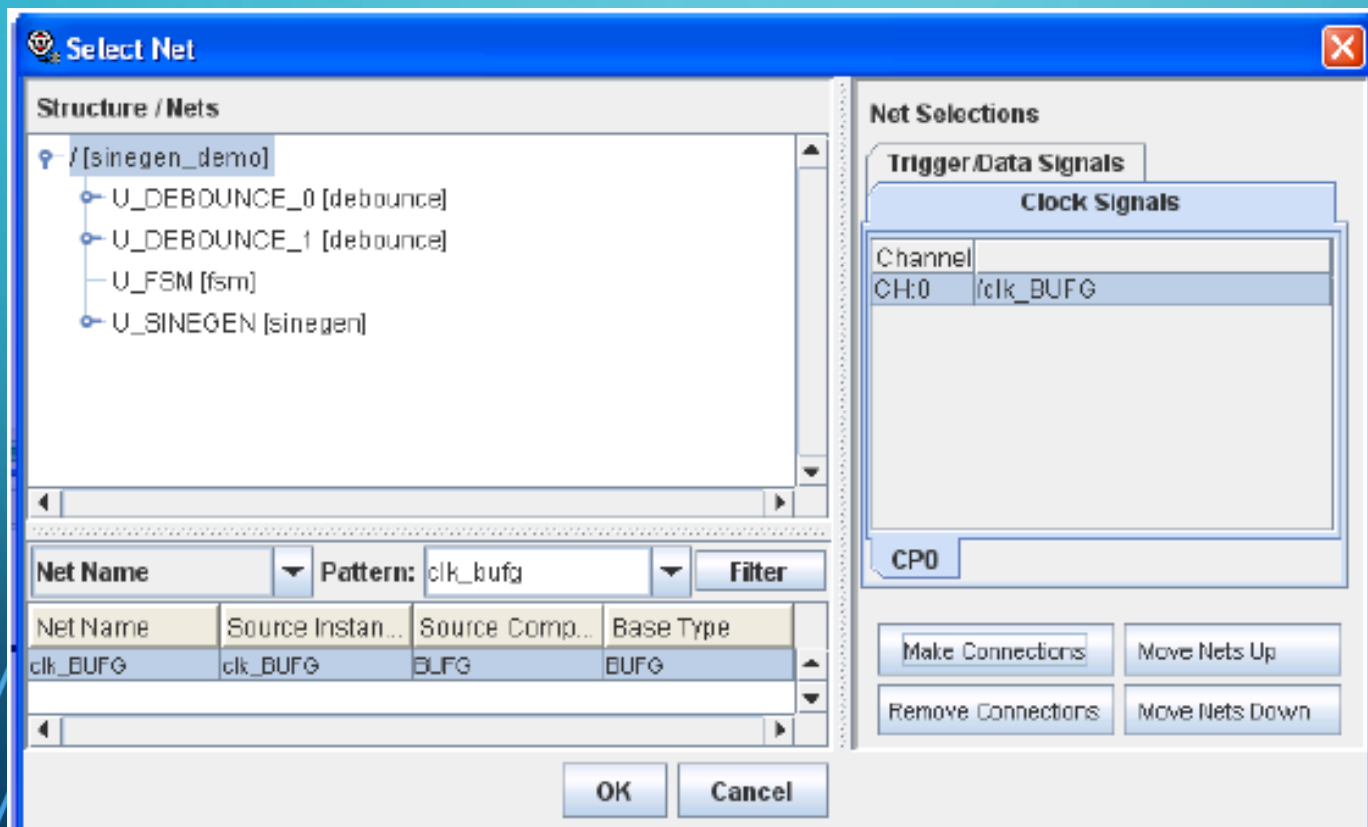
At the bottom, there are buttons for **< Previous**, **Next >**, and **Remove Unit**.

Annotations:

- An orange box labeled "Width of the trigger" points to the **Trigger Width** field for TRIG0.
- An orange box labeled "Different signal selection for the trigger" points to the **Match Type** dropdown for TRIG0.
- An orange box labeled "Matching option for the trigger" points to the **Bit Values** and **Functions** fields for TRIG0.

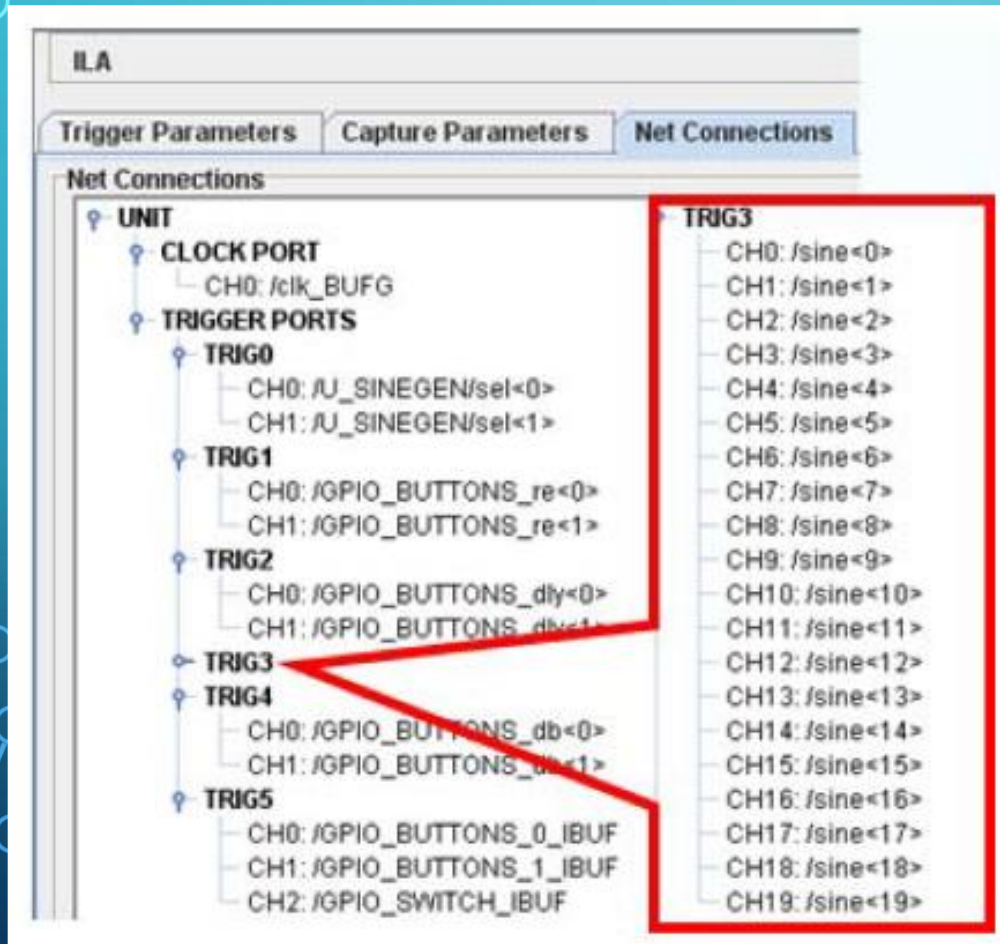
ILA CORE: WIZARD

- The net selection wizard: the clock selection to sample the data

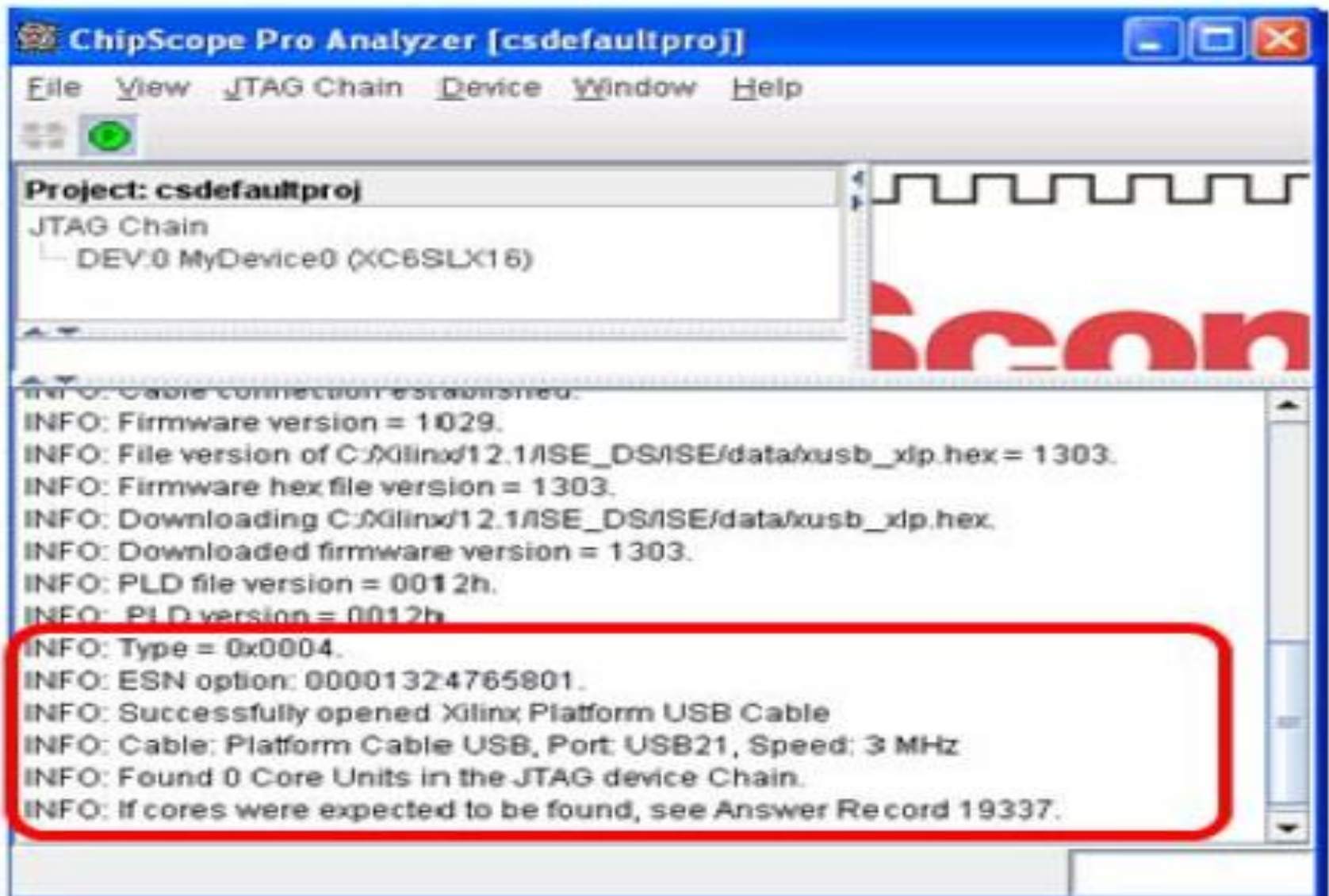


ILA CORE: WIZARD

➤ The net selection wizard: the data signals selection

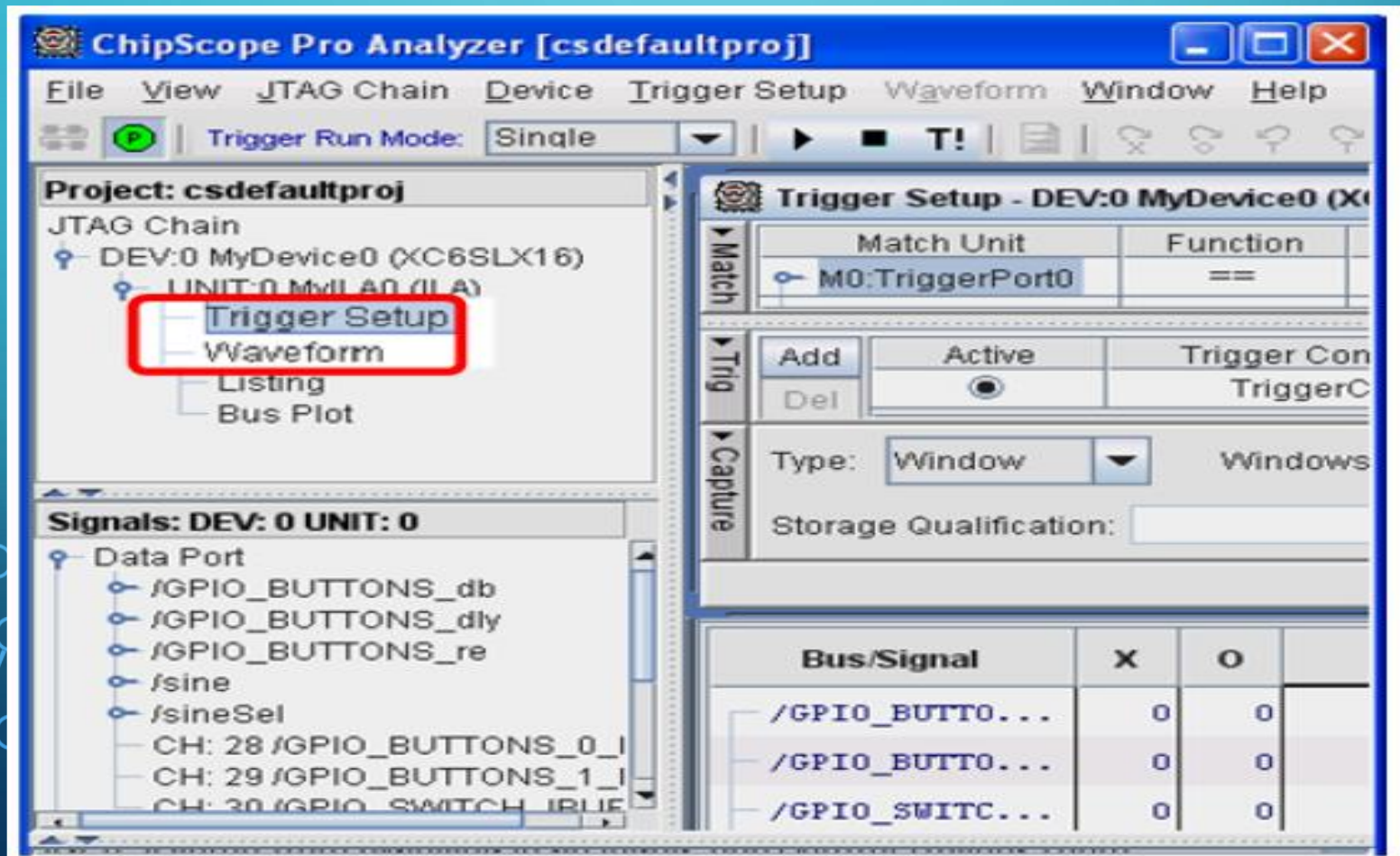


CHIPSCOPE: ANALYZER



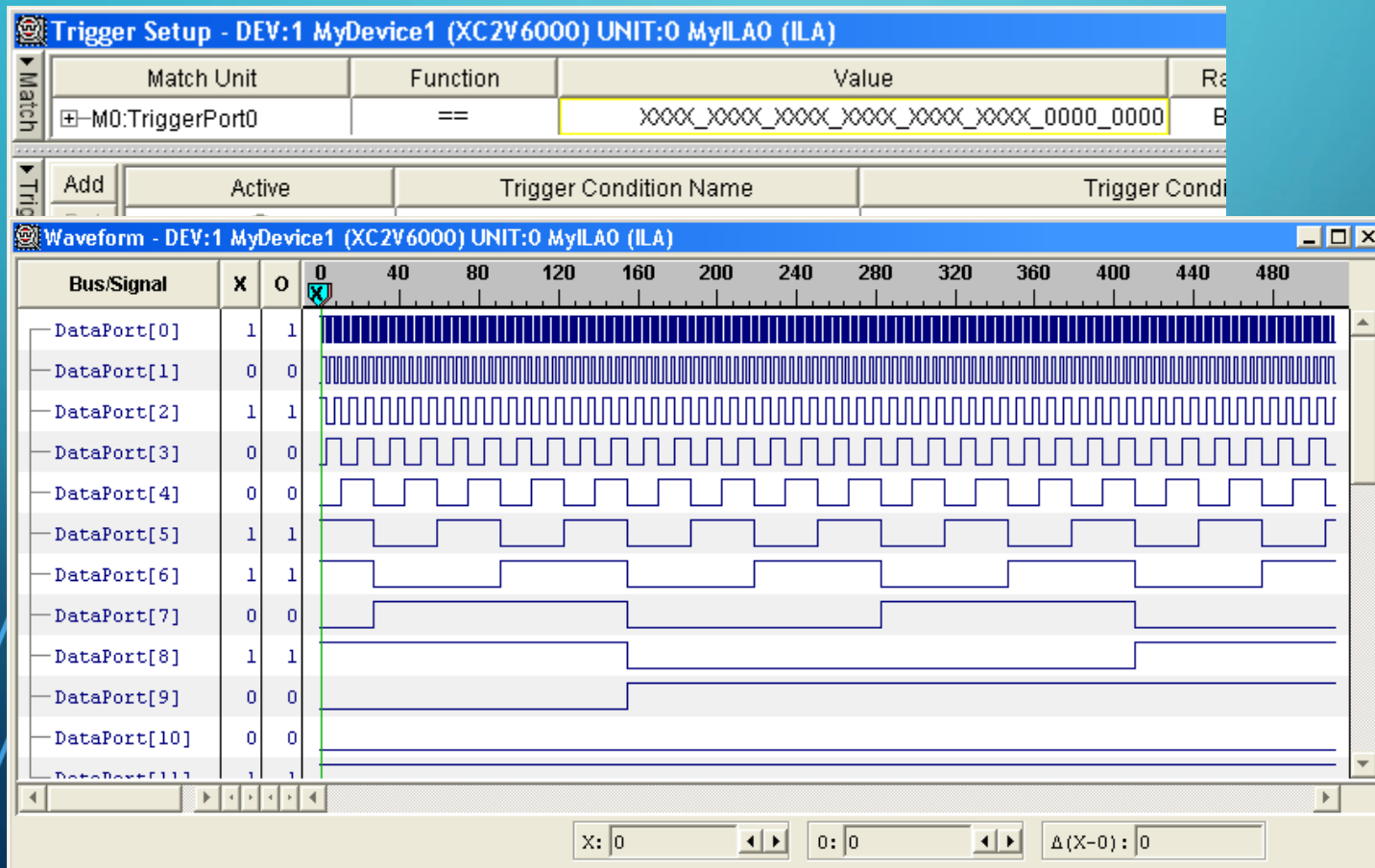
CHIPSCOPE: ANALYZER

- The Chipscope software: selection of the ILA unit: trigger and waveform



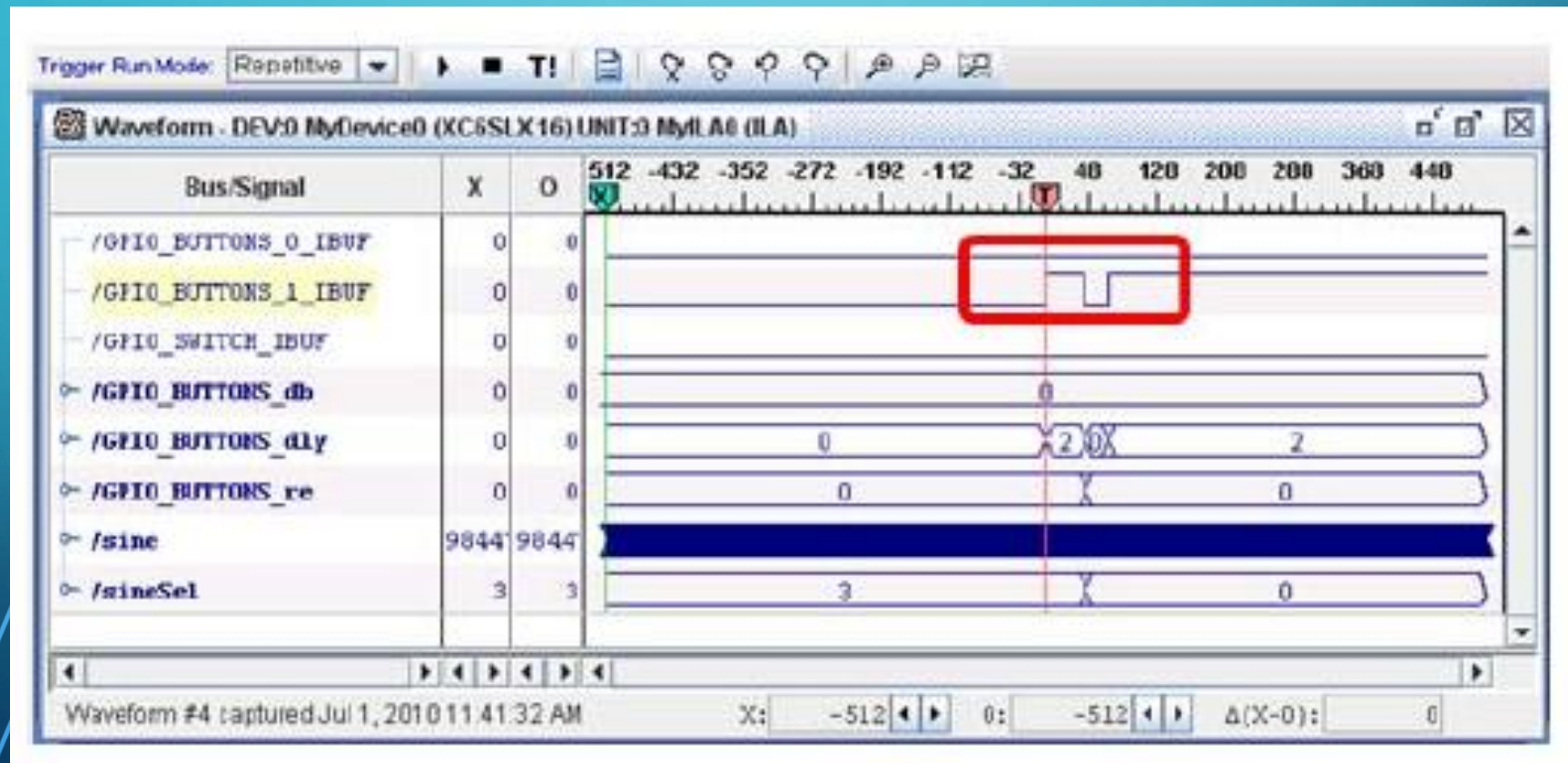
CHIPSCOPE: ANALYZER

- The Chipscope software: selection of the **trigger sequence** and monitor of the **data selected**



CHIPSCOPE: ANALYZER

- The Chipscope software: selection of the **trigger sequence** and monitor of the **data selected**



A decorative graphic on the left side of the slide, consisting of a network of white lines and circles on a blue gradient background, resembling a circuit board or a neural network.

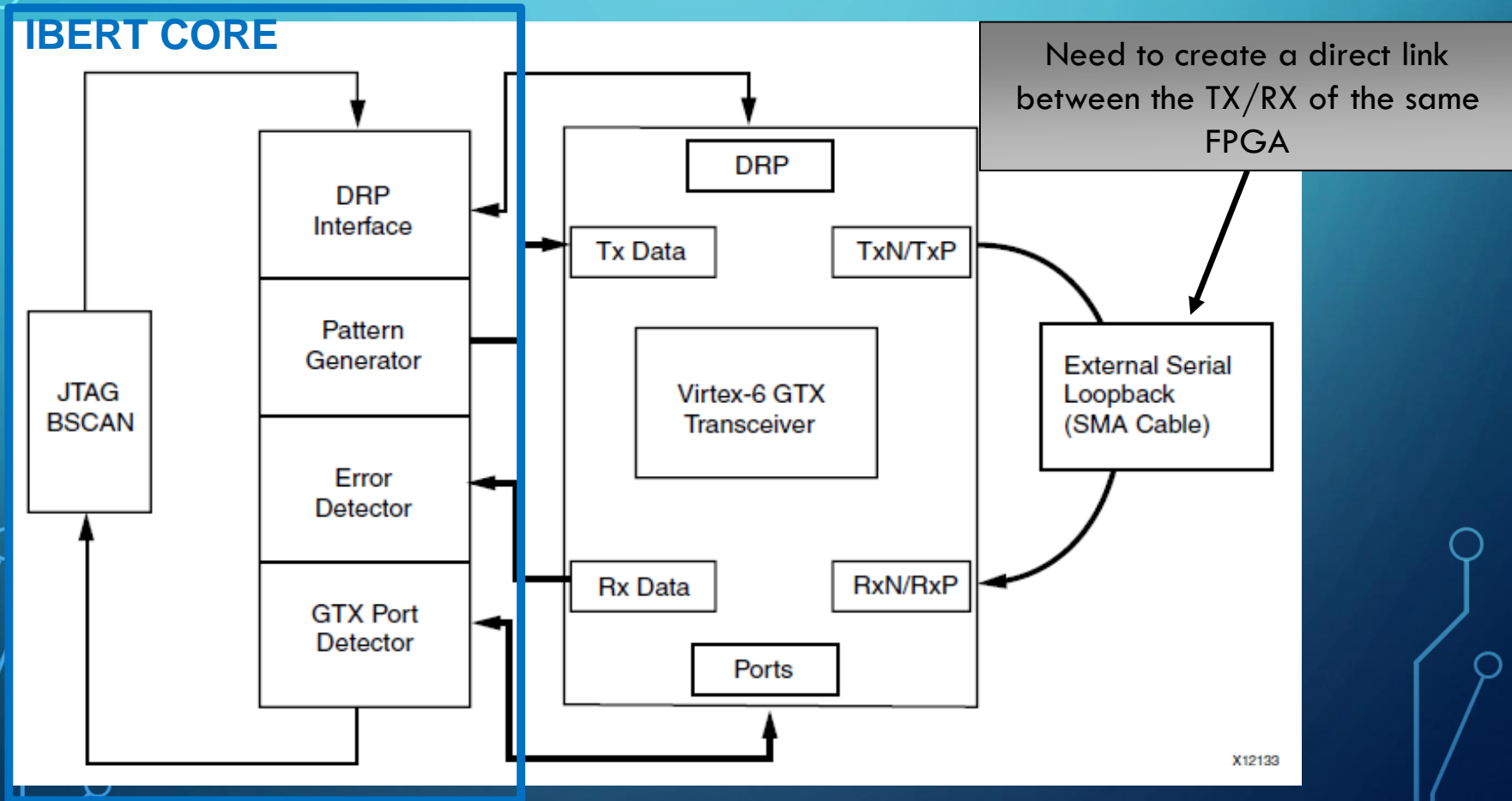
IBERT CORE

THE IBERT CORE

- The IBERT core is used to evaluate and monitor the functionality of transceivers for a variety of Xilinx devices, (for example the Spartan®-6 GTP transceiver, Virtex®-6 GTX transceivers) .
- The design includes pattern generators and checkers implemented in FPGA logic, as well as access to the ports and dynamic reconfiguration port (DRP) attributes of the GTX transceivers.
- The IBERT core is a self-contained design. When generated, it runs through the entire implementation flow, including bitstream generation.

IBERT DESIGN FLOW (2/1)

- Description of interface between the IBERT core and the transceiver

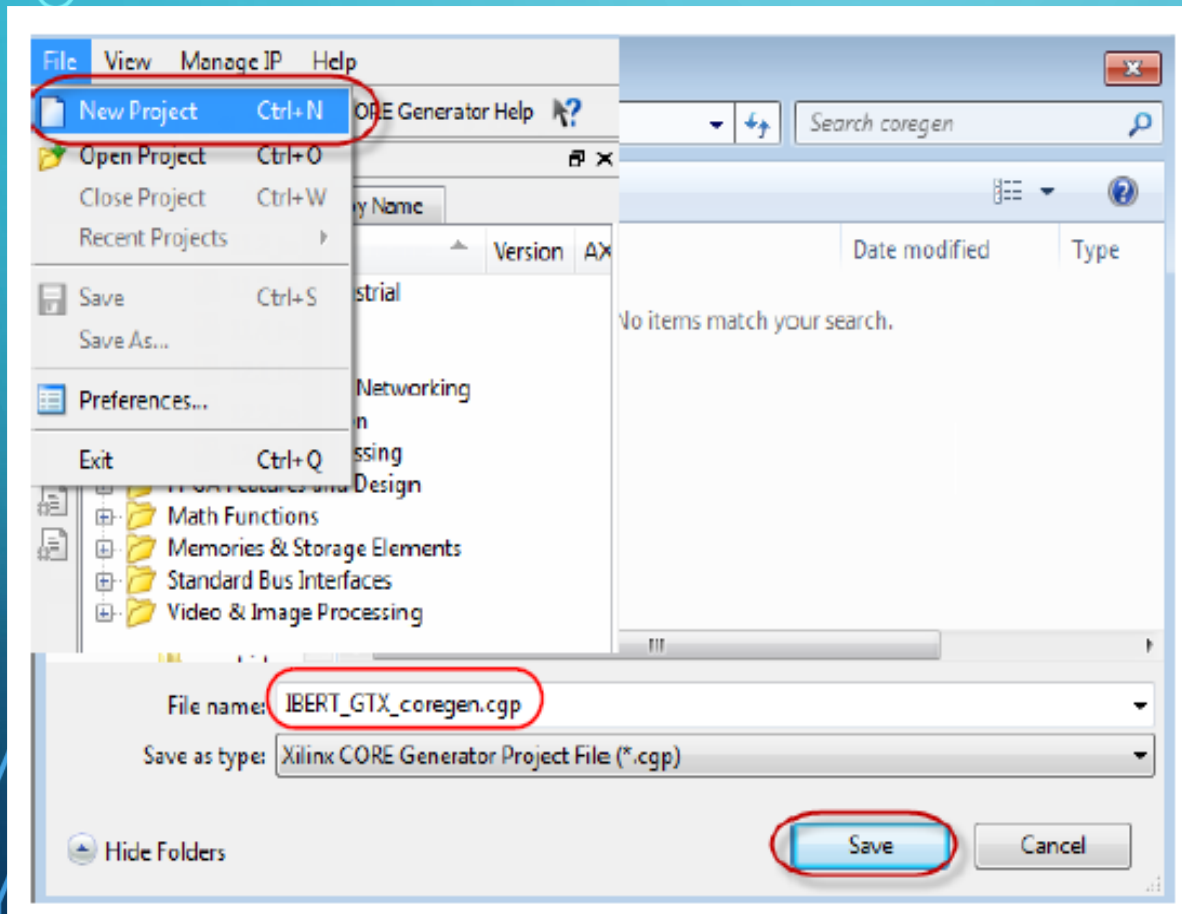


IBERT DESIGN FLOW (2/2)

- **DRP Interface and GTX Port Registers:** IBERT provides you with the flexibility to change GTX transceiver ports and attributes. Dynamic reconfiguration port (DRP) logic is included, which allows the runtime software to monitor and change any attribute in any of the GTX transceivers included in the IBERT core.
- **Pattern Generator:** Each GTX transceiver enabled in the IBERT design has both a pattern generator and a pattern checker. The pattern generator sends data out through the transmitter.
- **Error Detector:** Each GTX transceiver enabled in the IBERT design has both a pattern generator and a pattern checker. The pattern checker takes the data coming in through the receiver and checks it against an internally generated pattern.

GENERATING AN IBERT DESIGN

➤ Create a new project with only the **IBERT** core

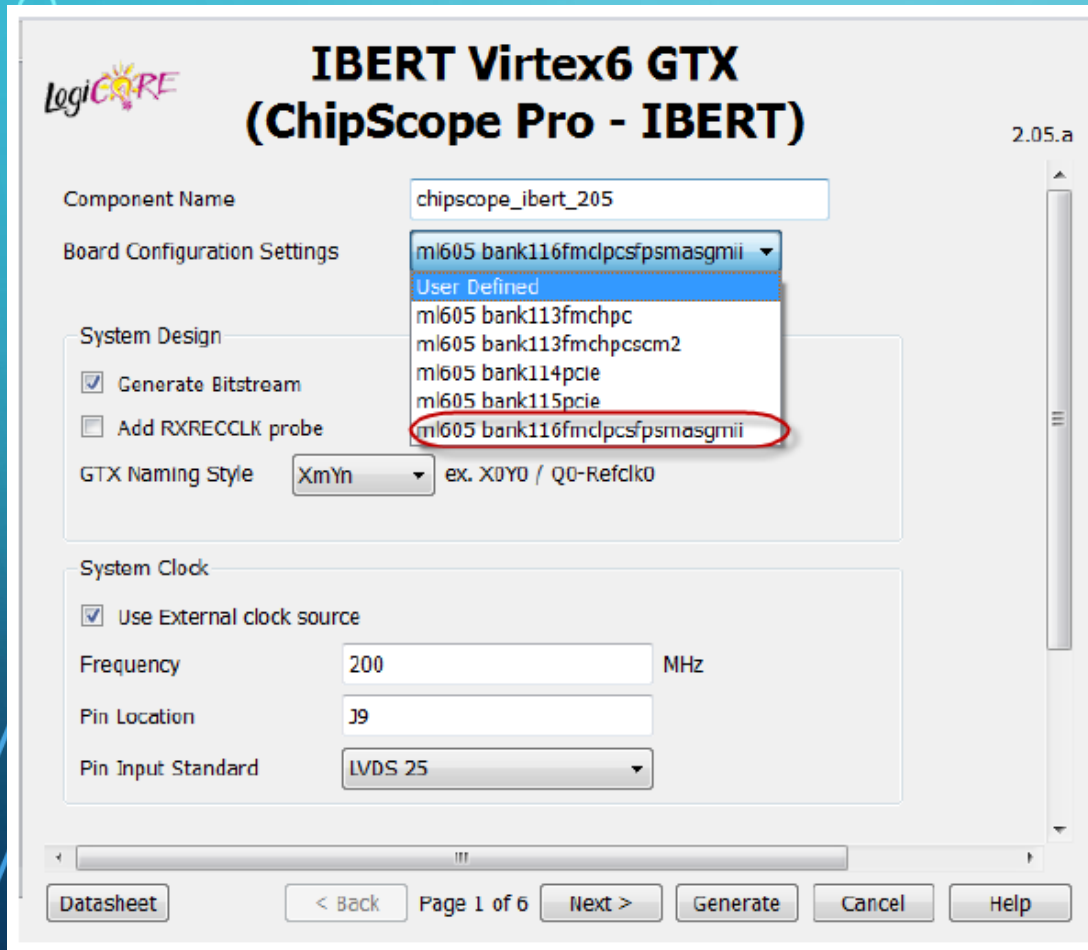


The core create a standalone project.

A bit programming file is generated with the purpose to test only the quality of link

GENERATING AN IBERT DESIGN

- Define the parameter of the IBERT core



LogiCORE
IBERT Virtex6 GTX
(ChipScope Pro - IBERT) 2.05.a

Component Name: chipscope_ibert_205

Board Configuration Settings: **mi605 bank116fmcpsfpmasgmii** (selected), User Defined, mi605 bank113fmchpc, mi605 bank113fmchpcscm2, mi605 bank114pcie, mi605 bank115pcie, mi605 bank116fmcpsfpmasgmii (circled in red)

System Design:

- ☒ Generate Bitstream
- ☐ Add RXRECCLK probe

GTX Naming Style: XmYn ex. X0Y0 / Q0-Refclk0

System Clock:

- ☒ Use External clock source

Frequency: 200 MHz

Pin Location: J9

Pin Input Standard: LVDS 25

Buttons: Datasheet, < Back, Page 1 of 6, Next >, Generate, Cancel, Help

- With the core generator interface we can select a series of parameter of the link, for example the speed of the link and the clock reference.

GENERATING AN IBERT DESIGN

➤ The graphic interface of the IBERT

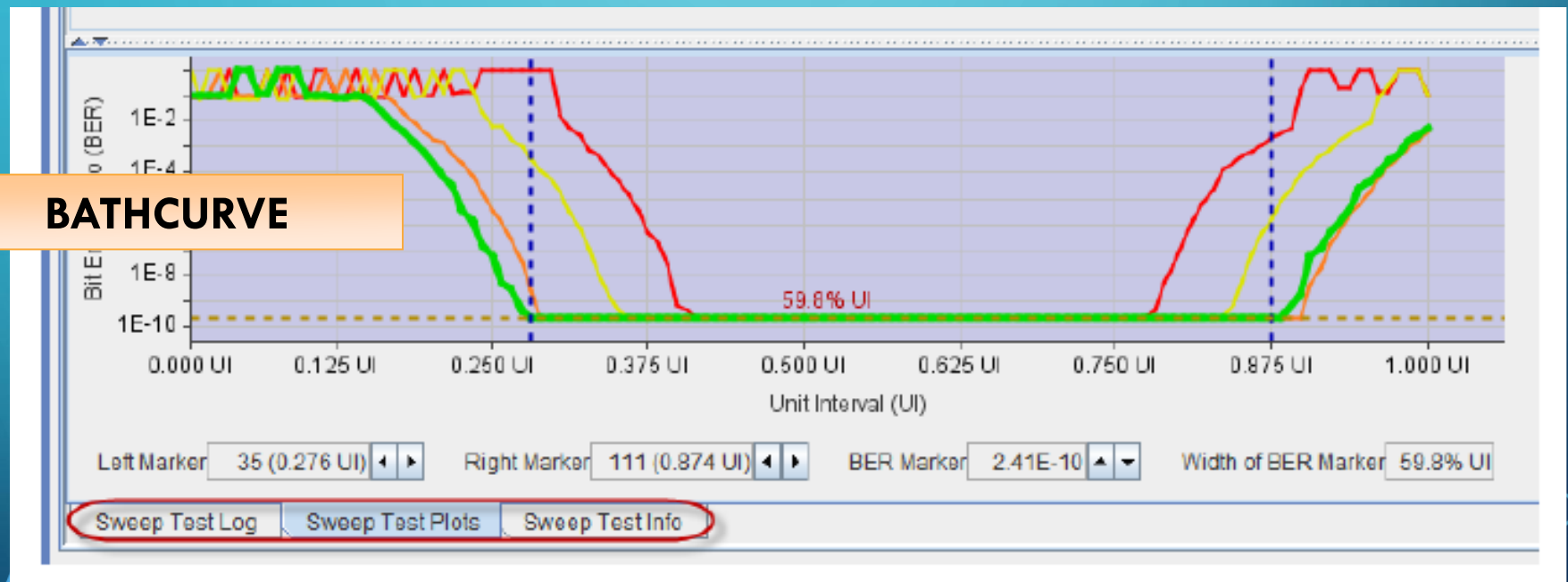
The screenshot displays the IBERT Console window for a MyDevice1 (XC6VLX240T) unit. The 'MGT/BERT Settings' tab is active, showing a table of settings for four GTX ports: GTX_X0Y16, GTX_X0Y17, GTX_X0Y18, and GTX_X0Y19. The 'Loopback Mode' row is highlighted with a red circle, and its dropdown menu is open, showing options: None, Near-End PCS, Near-End PMA, Far-End PMA, and Far-End PCS. Other settings include MGT Alias, Tile Location, MGT Link Status, MGT Edit Line Rate, TX PLL Status, RX PLL Status, Channel Reset, TX Polarity Invert, TX Error Inject, TX Diff Output Swing, TX Pre-Emphasis, TX Post-Emphasis, RX Polarity Invert, and RX AC Coupling Enable.

	GTX_X0Y16	GTX_X0Y17	GTX_X0Y18	GTX_X0Y19
MGT Settings				
- MGT Alias	GTX0_116	GTX1_116	GTX2_116	GTX3_116
- Tile Location	GTX_X0Y16	GTX_X0Y17	GTX_X0Y18	GTX_X0Y19
- MGT Link Status	No Link	No Link	5.0 Gbps	No Link
- MGT Edit Line Rate	5.0 Gbps	5.0 Gbps	5.0 Gbps	1.25 Gbps
- TX PLL Status	LOCKED	LOCKED	LOCKED	LOCKED
- RX PLL Status	LOCKED	LOCKED	LOCKED	LOCKED
- Loopback Mode	None	None	None	None
- Channel Reset	Reset	Reset	None	Reset
- TX Polarity Invert	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
- TX Error Inject	Inject	Inject	Inject	Inject
- TX Diff Output Swing	590 mV (0110)	590 mV (0110)	590 mV (0110)	590 mV (0110)
- TX Pre-Emphasis	0.000 dB (0000)	0.000 dB (0000)	0.000 dB (0000)	0.000 dB (0000)
- TX Post-Emphasis	0.000 dB (00000)	0.000 dB (00000)	0.000 dB (00000)	0.000 dB (00000)
- RX Polarity Invert	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
- RX AC Coupling Enable	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

GENERATING AN IBERT DESIGN

- In order to characterize the link the Bit Error Rate (BER) is plotted (called “bath” curve)

BATHCURVE



GENERATING AN IBERT DESIGN

- In order to characterize the link the Bit Error Rate (BER) is plotted (called “bath” curve)



An abstract graphic on the left side of the slide, consisting of a network of light blue lines and small circles, resembling a circuit board or a neural network, set against a dark blue background.

EXERCISE IN THE LABORATORY

EXERCIZE IN THE LABORATORY

➤ The FTK system is composed of 2 different boards + 1 chip

VME CRATE



AMBFTK



LAMBFTK

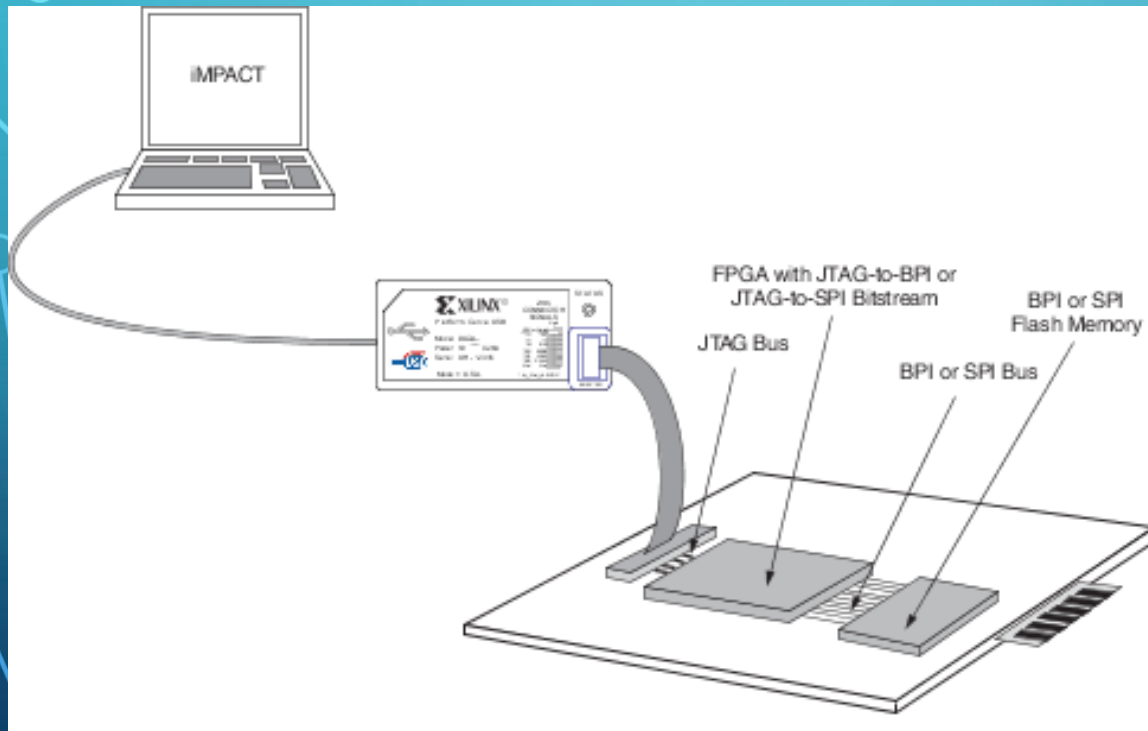


AMCHIP



EXERCISE IN THE LABORATORY

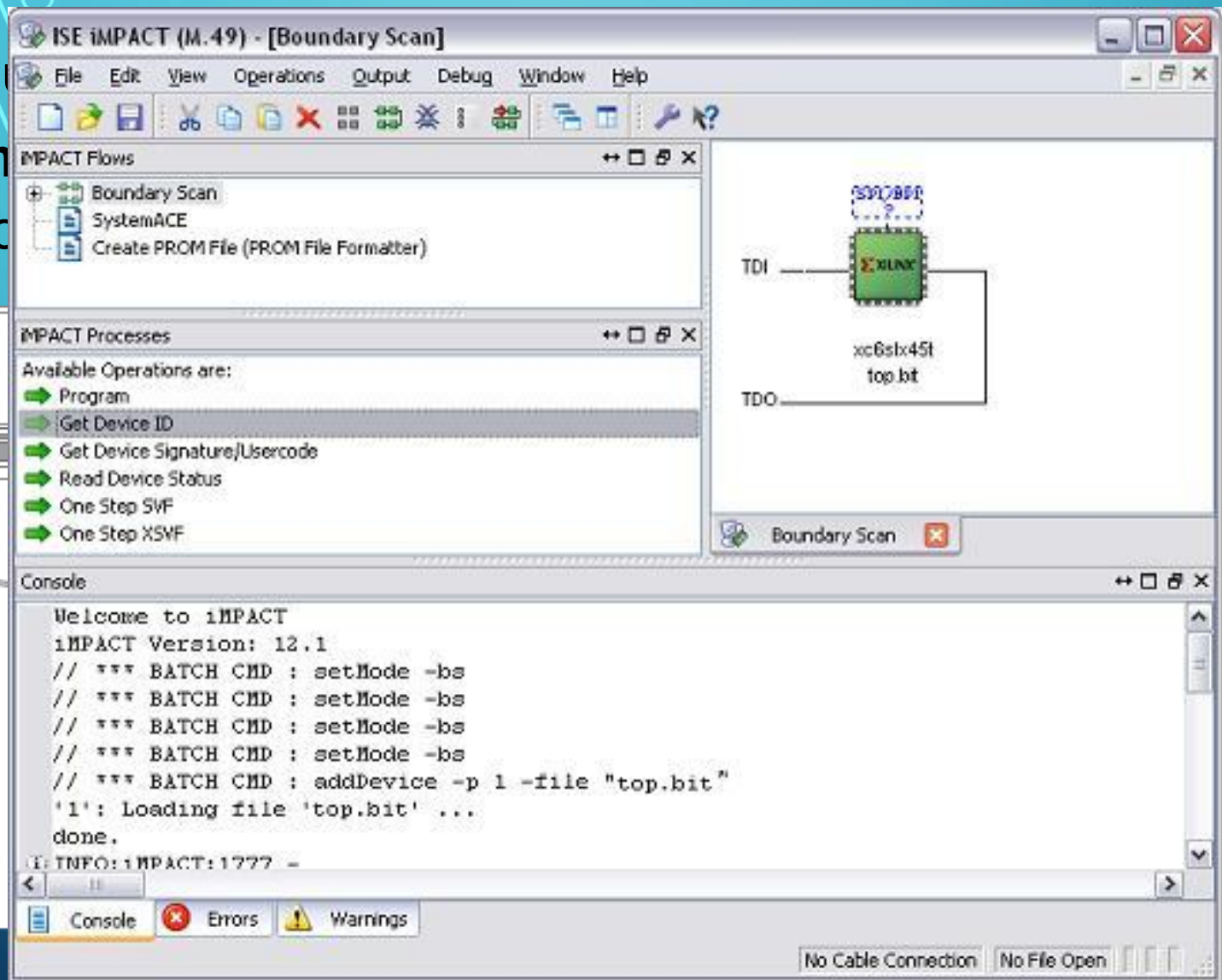
- Do you remember the 6° STEP?
- The **Impact software** help us to load the program bit file into your FPGA



EXERCISE IN THE LABORATORY

➤ Do you

➤ The In
into yo



EXERCISE IN THE LABORATORY

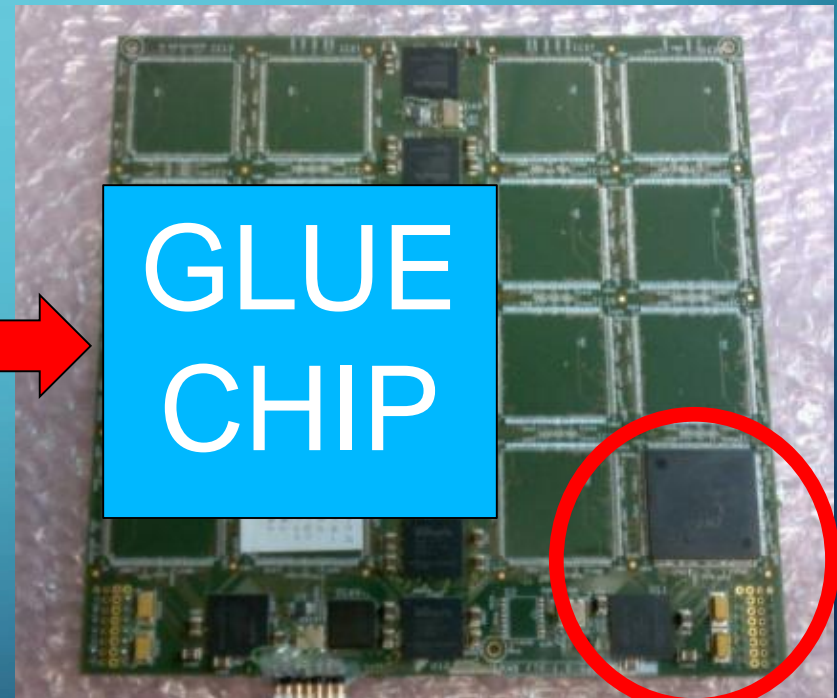
- We test with ChipScope the data chain of the output of the Amchip to LAMB and to AMB



Output
parallel data



GLUE
CHIP



Input Stimulus

EXERCISE IN THE LABORATORY

- We test with ChipScope the data chain of the output of the Amchip to LAMB and to AMB

