Peter the Great St. Petersburg Polytechnic University
Institute of Computer Science and Cybersecurity
Graduate School of Computer Technologies and Information Systems

Lecture: ModelSim GUI Essentials

Subject: Automation of discrete device design (in English)

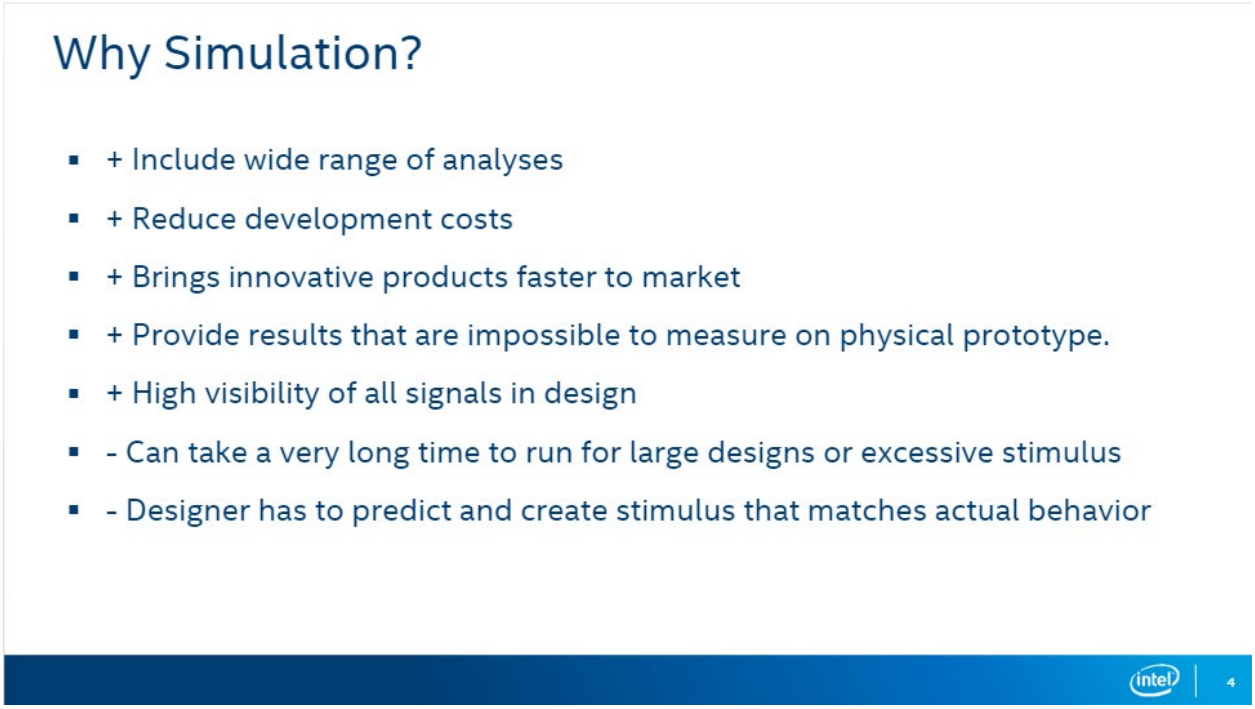
Completed by student of group 5130901/10101 _____ Nepomnyashchiy M.T.
(signature)

Lecturer _____ Antonov A.P.
(signature)

Saint Petersburg
2023

3. Simulation with ModelSim

3.1. Why Simulation?



There're a lot of reasons for using simulators for debugging. As opposed to trial-and-error, a smart simulation allows targeted implementation of design choices in various stages of cycle.

Benefits of simulation:

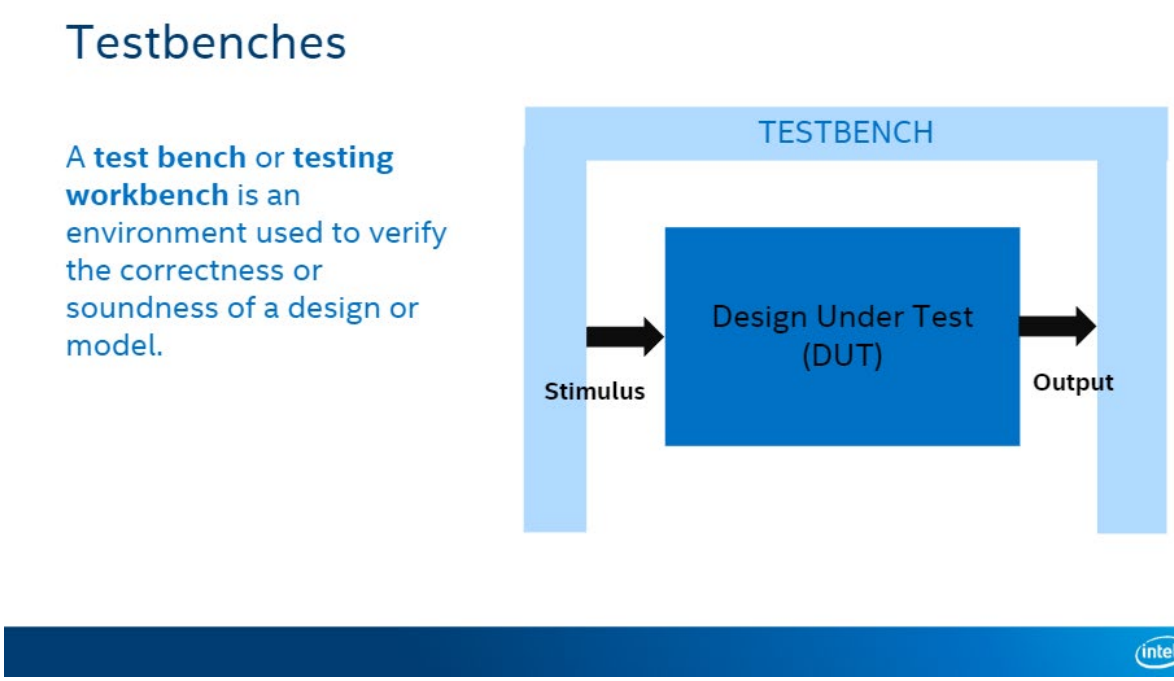
- **Wider range of analyses**: You can simulate many different scenarios and conditions that would be difficult or even impossible to create in the real world. This can help you to identify potential problems early in the design process.
- **Reduced development costs**: Simulation can be much cheaper than physical prototyping, especially for complex products.
- **Faster time to market**: You can iterate on your designs more quickly using simulation, which can help you to get your products to market faster.
- **Impossible results**: You can get results from simulations that you could never get from physical prototype, such as how a product would perform in extreme conditions.
- **High visibility of signals**: You can see all of the signals in your design at once during a simulation, which can help you to identify and fix problems.

Drawbacks of simulation:

- **Time-consuming**: Simulations can take long time to run, especially for large or complex designs.
- **Simulation prediction**: The designer needs to be able to predict and create realistic stimulus that matches the actual behavior of the product.

So, simulation is a valuable tool that can be used to save time and money during the development process. However, it is important to be aware of the limitations of simulation before using it.

3.2. Testbenches



A testbench or testing module is an **RTL** module or entity that resides one level higher in the hierarchy than the top-level design. It is used to verify the correctness or soundness of a design or a model.

A testbench can be written in the **Verilog** (.v file), **System Verilog** (.sv file) or **VHDL** languages. The DUT, or Device Under Test, is the main part of a design that is being tested. It's the top-level design module, which means it's the highest level of organization in the design hierarchy.

Universal Verification Methodology (**UVM**) is a methodology used to thoroughly test a logic function. It's a popular approach for ensuring that the function performs as expected.

UVM provides a standardized way to create testbenches, which are essentially simulated environments used to apply inputs to the logic function and check its outputs. By using UVM, you can create more reusable and efficient testbenches, which can save time and effort during the design process.

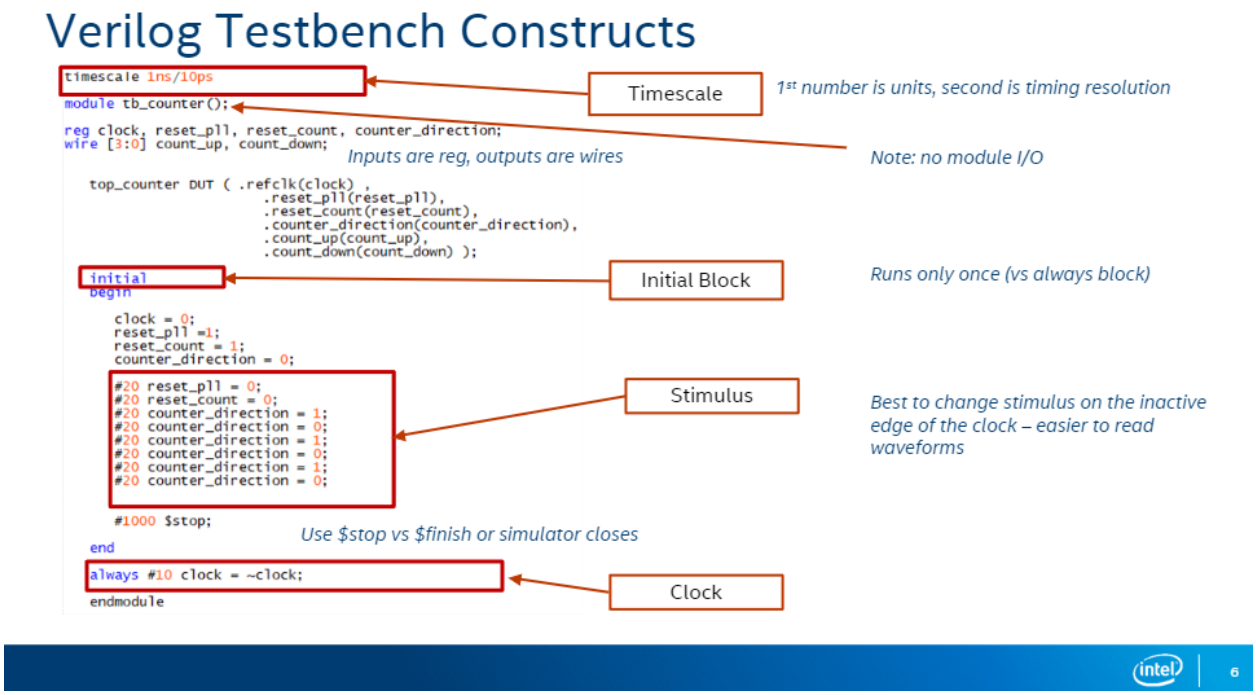
Testbenches are like testing grounds for designs (think machines under construction). They send inputs (like signals) and check outputs to ensure they match expectations.

However, a critical feature of advanced testbenches is self-checking:

- The testbench automatically compares the DUT's output with expected values.
- If deviations occur, the testbench flags them, alerting you to potential issues in your circuit.

This self-checking capability becomes even more valuable during regression testing. With a self-checking testbench, each added module undergoes automatic verification, ensuring its proper operation within the larger design.

3.3. Verilog Testbench Constructs

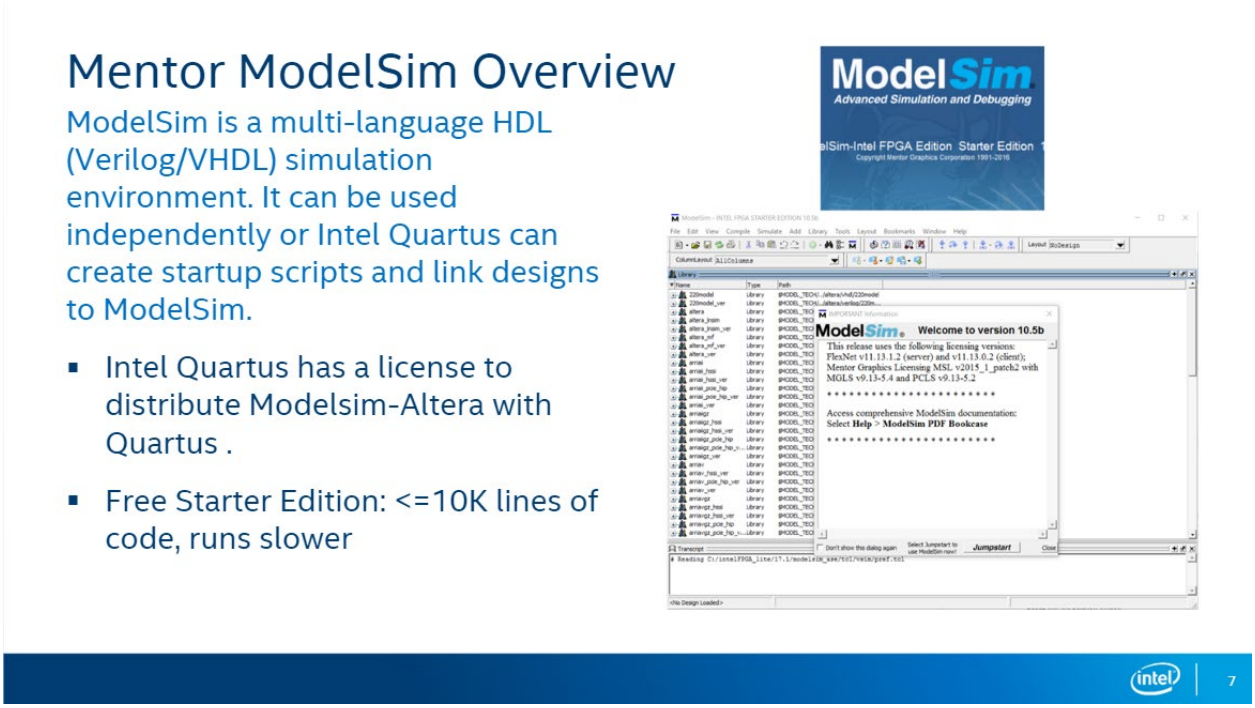


On the picture above we can see a Verilog module called `tb_counter` that serves as a testbench for another module called `top_counter`. The testbench module includes the following components:

- **Timescale:** The line `timescale 1ns/10ps` sets the timescale for the simulation. This means that there are 10ps in 1ns.
- **Inputs:** The testbench module has four input signals: `clock`, `reset_pll`, `reset_count`, and `counter_direction`. These signals are used to control the behavior of the `top_counter` module.
- **Outputs:** The testbench module has two output signals: `count_up` and `count_down`. These signals are used to monitor the outputs of the `top_counter` module.
- **Initial Block:** The initial block is used to initialize the testbench module. In this case, the initial block sets the initial values of the input signals and then starts the simulation.
- **Always Block:** The always block is used to generate the clock signal for the simulation. The always block also changes the values of the other input signals at specific points in time.
- **Stimulus:** The stimulus code is used to provide input signals to the `top_counter` module. In this case, the stimulus code changes the values of the input signals at specific points in time.
- **$stop:** The `$stop` system task is used to stop the simulation after a certain amount of time.

The testbench module is designed to test the functionality of the `top_counter` module. The testbench module provides a set of input signals to the `top_counter` module and then monitors the outputs of the `top_counter` module to see if they are correct.

3.4. Mentor ModelSim Overview



Quartus Prime Lite and ModelSim Integration:

- Quartus Prime Lite includes ModelSim for FPGA design simulation.
- ModelSim supports multiple languages (Verilog, VHDL).

Free Version Limitations:

- Free version simulates up to 10,000 lines of code.
- This includes "IP" libraries, not just your written code.
- Line count can add up quickly due to hidden libraries.

Linking Underlying Libraries:

- Linking IP libraries (memories, PLLs, interfaces) is crucial.
- Need to know which libraries to compile and link.

Nativelink Feature:

- Quartus offers the Nativelink feature.
- Creates a simulation setup script for ModelSim.
- Automatically references necessary design files.

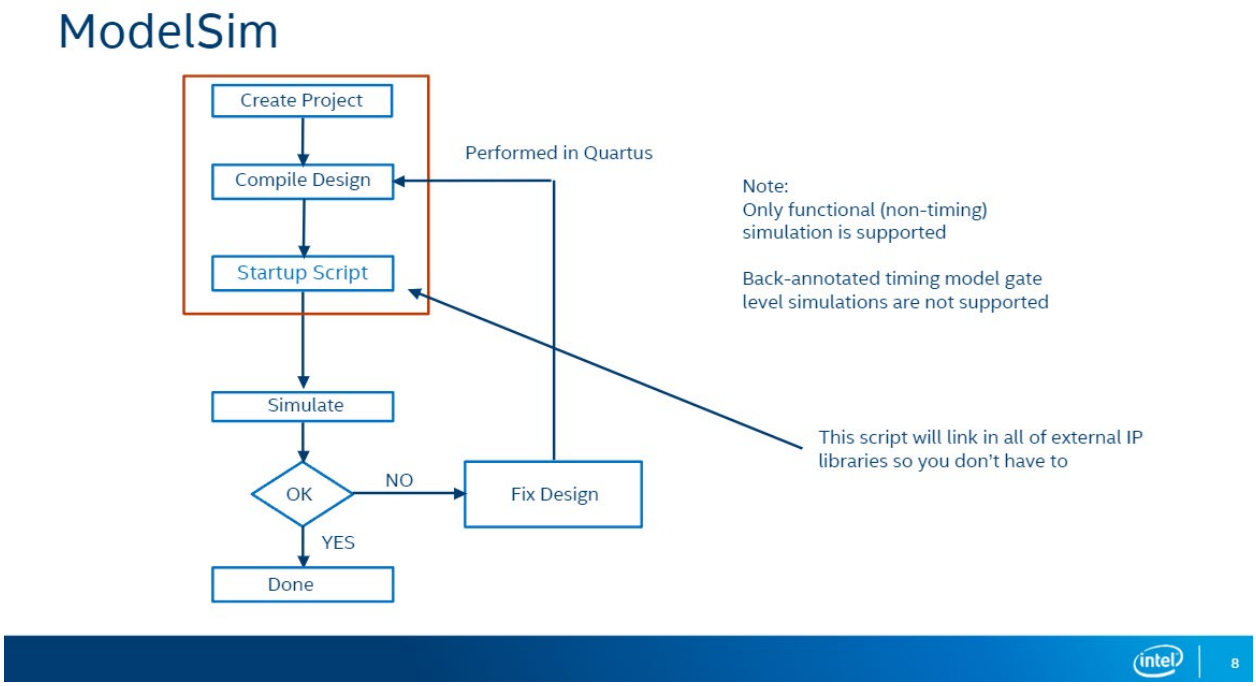
RTL vs. Gate Level Simulation:

- Simulator can run on RTL or gate level netlist.
- Gate level simulation is much slower (10-100 times slower).
- Use RTL for functional correctness, not timing analysis.

Timing Analysis:

- Use a static timing analyzer for accurate timing analysis.
- RTL simulation does not account for lookup table delays.

3.5. ModelSim

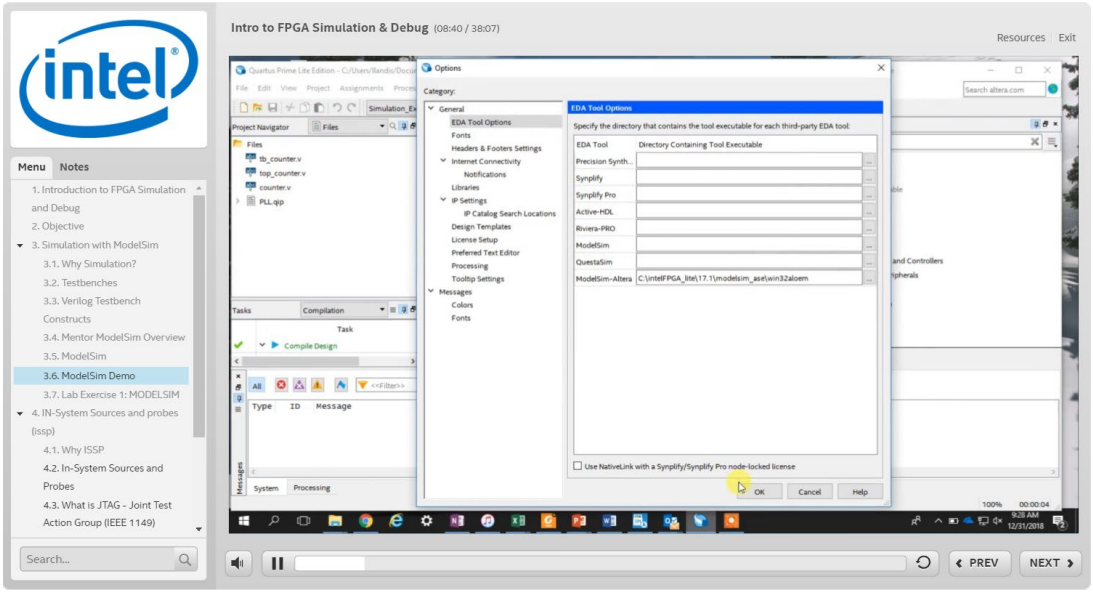


To create a simulation model using ModelSim in the Quartus Prime softwarte we need to follow some basic steps:

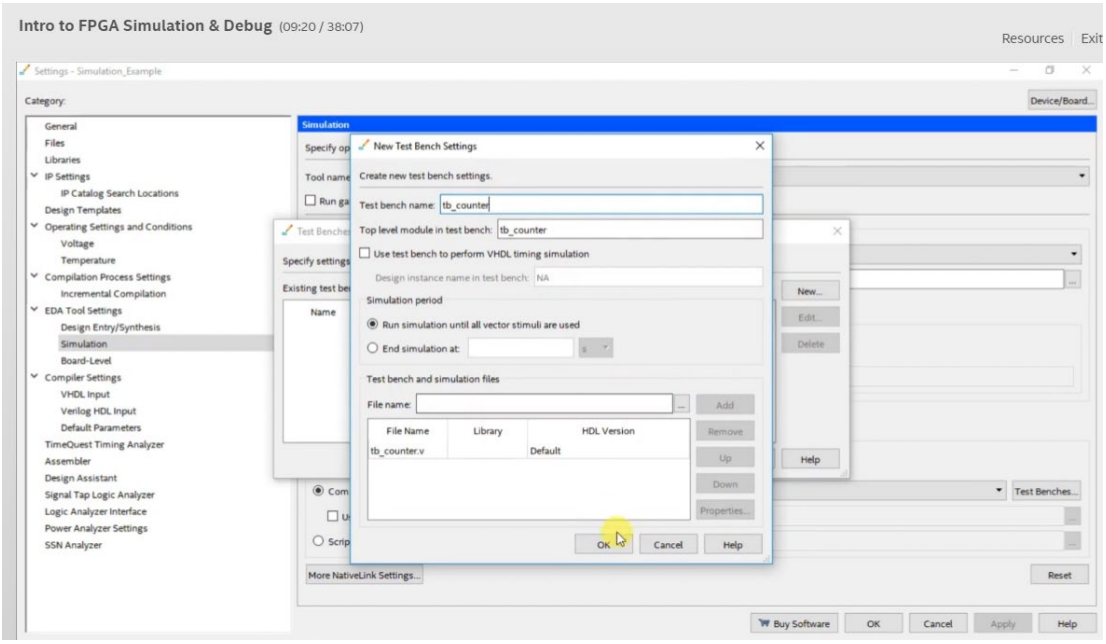
1. **Create Project**: This step involves creating a new project in Quartus Prime.
2. **Compile Design**: This step involves compiling the design code that you have written.
3. **Startup Script**: This step involves creating a simulation startup script. This script will tell ModelSim which files to simulate and how to run the simulation.
4. **Simulate**: This step involves running the simulation.
5. **Fix Design (Optional)**: If the simulation fails, you will need to fix the errors in your design code and then go back to step 2.
6. **Done**: If the simulation is successful, then you are finished.

3.6. Working with ModelSim using QP software

1. The first step is to show QP using EDA Tool Optoins where ModelSim tools are installed:



2. The next step is to select the testbrench in EDA Tool Settings → Simulation:



3. Then you should go to Tools → Run Simulation Tool → RTL Simulation which will launch ModelSim and direct you to its' window

4. Next steps which lecturer shows on the ModelSim Demo contain information which we took in lab 7 last semester (how to coop with the ModelSim cmd and how to use wave tools)

4. IN-System Sources and probes (ISSP)

4.1. Why ISSP

Why ISSP

+ Quickly set signals to constants: pins or internal nodes

+ Easily monitor signals – non-triggered continuous display

+ Works on actual hardware

- Not triggered – might miss activity

The slide highlights the Interactive Signal Probe (ISSP) tool within ModelSim, and how it can enhance your design workflow. ISSP enables you to:

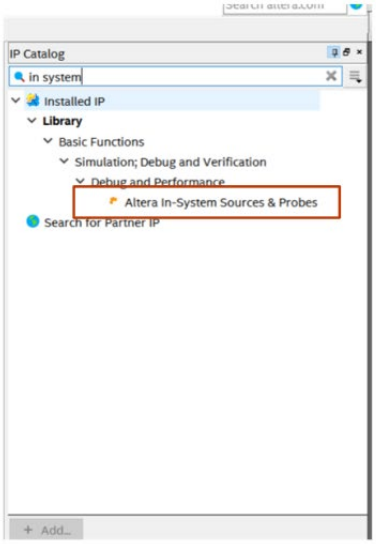
- Efficiently assign values to signals: With ISSP, you can swiftly set signals to constant values, such as 0 or 1, or establish connections between them and other signals or pins within your design. This proves to be particularly useful when examining and testing various segments of your circuit.
- Effortlessly monitor signal activity: The continuous, non-triggered display offered by ISSP eliminates the need for manual trigger configuration or waiting for specific events to occur. This streamlines the process of visualizing signal behavior and facilitates debugging within your circuit.
- Hardware integration: Notably, ISSP extends its functionality beyond simulations, allowing you to work with actual hardware. This empowers you to validate your circuit's behavior on real-world FPGAs or other devices, providing a more accurate assessment compared to pure simulation.

4.2. In-System Sources and Probes

In-System Sources and Probes

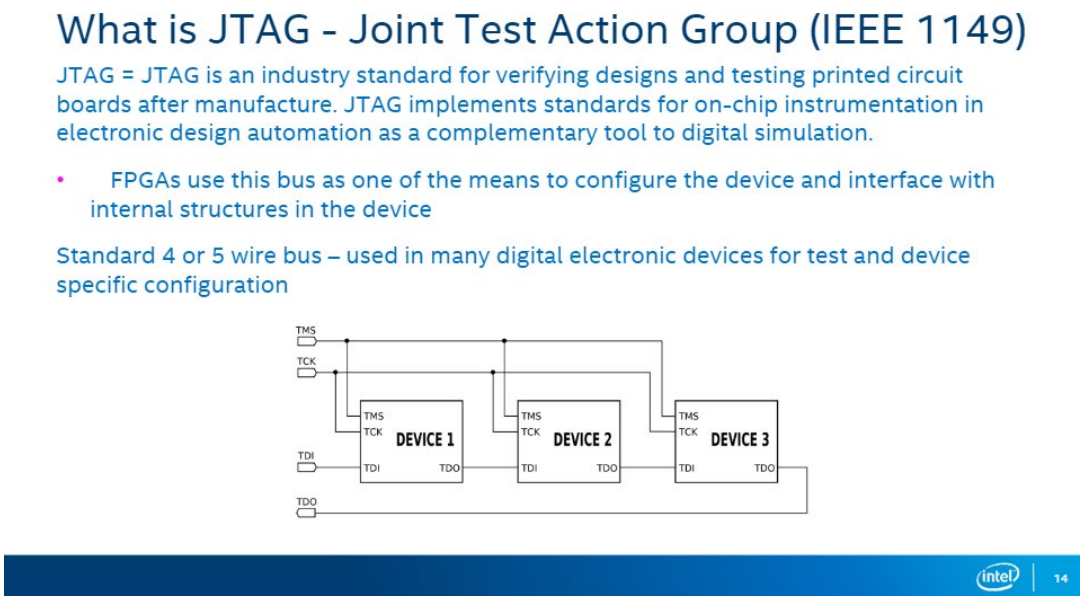
ISSP allows an easy way to drive and sample signals in hardware and provides a dynamic debugging environment.

- ISSP Editor consists of a probe function and interface to control the instances during run time.
- It is operated over JTAG.
- Each instance can drive and toggle values up to 512 signals.
- Can create up to 128 instances of ISSP using IP Catalog



The In-System Sources and Probes Editor in the Quartus software allows you to easily drive and sample signals directly on your hardware, which can be useful for debugging and testing your design. It provides a dynamic debugging environment, which means that you can make changes to your design while it is running and see the results immediately.

4.3. What is JTAG – Join Test Action Group (IEEE 1149)



JTAG (Joint Test Action Group) is an industry standard for verifying designs and testing printed circuit boards after manufacture. It implements standards for on-chip instrumentation in electronic design automation as a complementary tool to digital simulation.

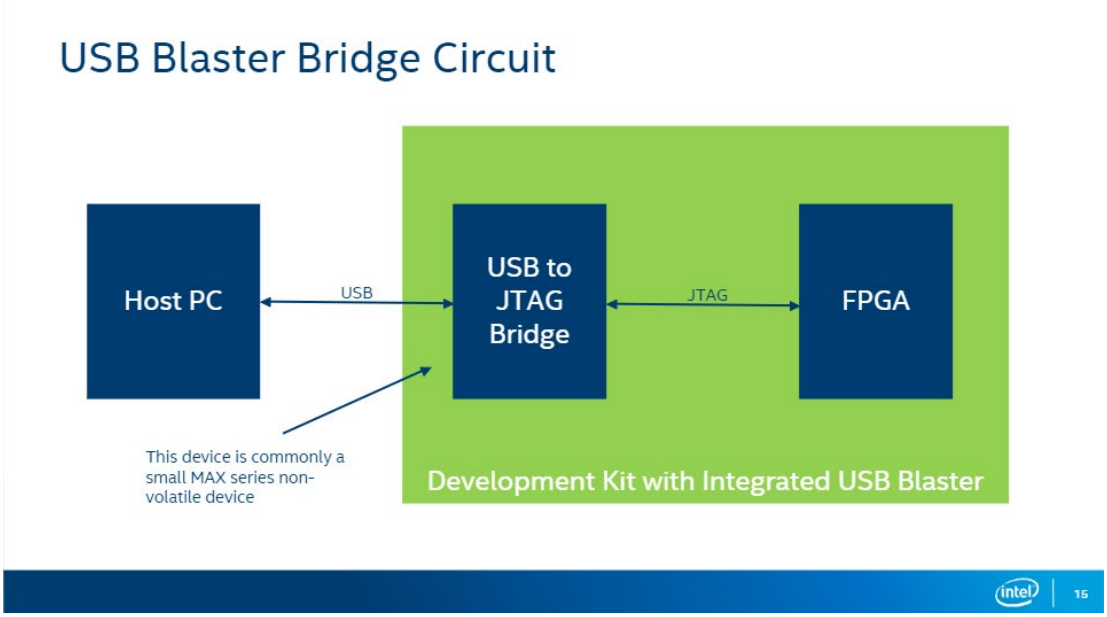
The slide also includes a diagram of a JTAG bus, which is a 4 or 5 wire bus used in many digital electronic devices for test and device-specific configuration. The four or five wires are:

- Test Data In (TDI)
- Test Data Out (TDO)
- Test Clock (TCK)
- Test Mode Select (TMS)
- Test Reset (TRST) (optional)

The TDI line is used to send data into the JTAG chain, while the TDO line is used to send data out. The TCK line is used to clock the data in and out of the JTAG chain, and the TMS line is used to select the test mode. The TRST line is used to reset the JTAG chain.

FPGAs (Field-Programmable Gate Arrays) use the JTAG bus as one of the means to configure the device and interface with internal structures in the device.

4.4. USB Blaster Bridge Circuit



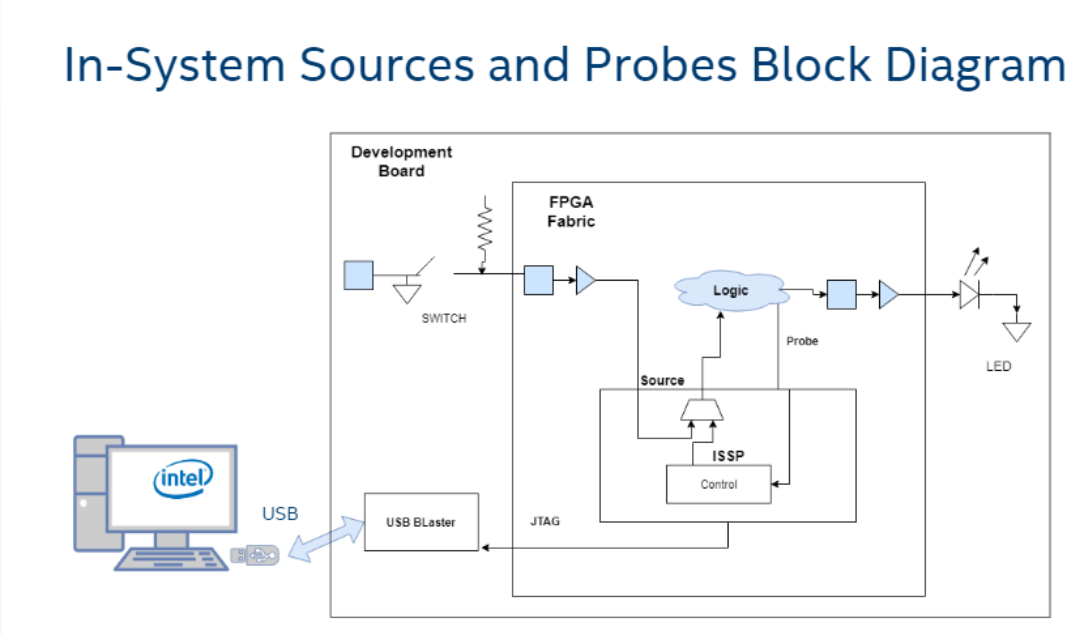
The slide above shows a diagram of a USB Blaster Bridge circuit, which is a device that connects a host PC to a USB to JTAG bridge. The USB to JTAG bridge is a small MAX-series non-volatile device that is typically integrated into a development kit. The JTAG port is used to program and debug FPGAs.

Explanation of the working process:

- **Host PC**: This is the computer that you will use to program and debug the FPGA.
- **USB**: This is the Universal Serial Bus, which is a type of serial bus that is used to connect devices to a computer.
- **JTAG**: This is the Joint Test Action Group, which is a standardized interface that is used to program and debug devices such as FPGAs.
- **FPGA**: This is a Field-Programmable Gate Array, which is a type of integrated circuit that can be programmed after it has been manufactured.

* The USB Blaster Bridge Circuit is a device that connects a host PC to an FPGA development board. It is used to program and debug the FPGA. The USB Blaster Bridge Circuit converts USB signals from the host PC to JTAG signals that the FPGA can understand.

4.5. In-System Sources and Probes Block Diagram

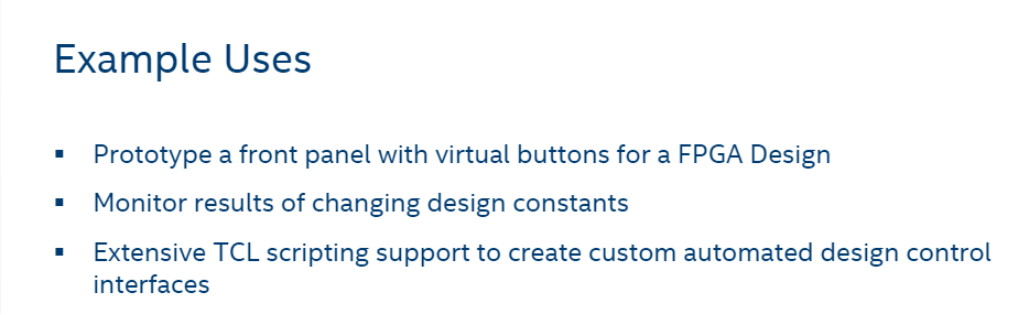


The block diagram includes the following components:

- **Development Board:** This is the board that contains the FPGA and other components that are needed to run the design.
- **FPGA Fabric:** This is the programmable logic fabric of the FPGA.
- **Logic:** This is the design that is being implemented on the FPGA.
- **SWITCH:** This is a multiplexer that selects which signal is routed to the probe.
- **Probe:** This is a circuit that monitors the value of a signal.
- **Source:** This is a circuit that provides test stimuli to the design.
- **LED:** This is an optional output that can be used to indicate the value of a signal.
- **ISSP:** This is the In-System Serial Programmer, which is used to program the FPGA.
- **Control:** This is the interface that is used to control the in-system sources and probes.
- **USB:** This is the connection that is used to connect the development board to the computer.
- **USB Blaster:** This is a device that is used to program the FPGA and to communicate with the in-system sources and probes.
- **JTAG:** This is a standard interface that is used to debug and test FPGAs.

The In-System Sources and Probes block diagram is a useful tool for understanding how to debug and test designs on FPGAs. By using in-system sources and probes, you can quickly and easily identify and fix problems in your design.

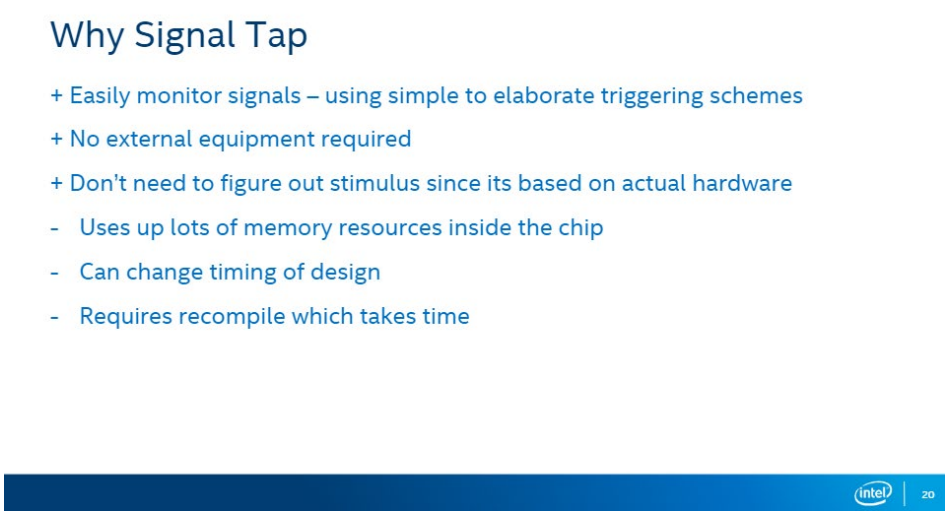
4.6. Example Uses



We can use ISSP to monitor results of changing design constants as well as for extensive tcl scripting support to create custom automated design control interfaces. Keep in mind that recompiling an FPGA after changing a single constant can take from a few minutes to several hours. Adding ISSP can make these changes quick and easy with no recompilation needed.

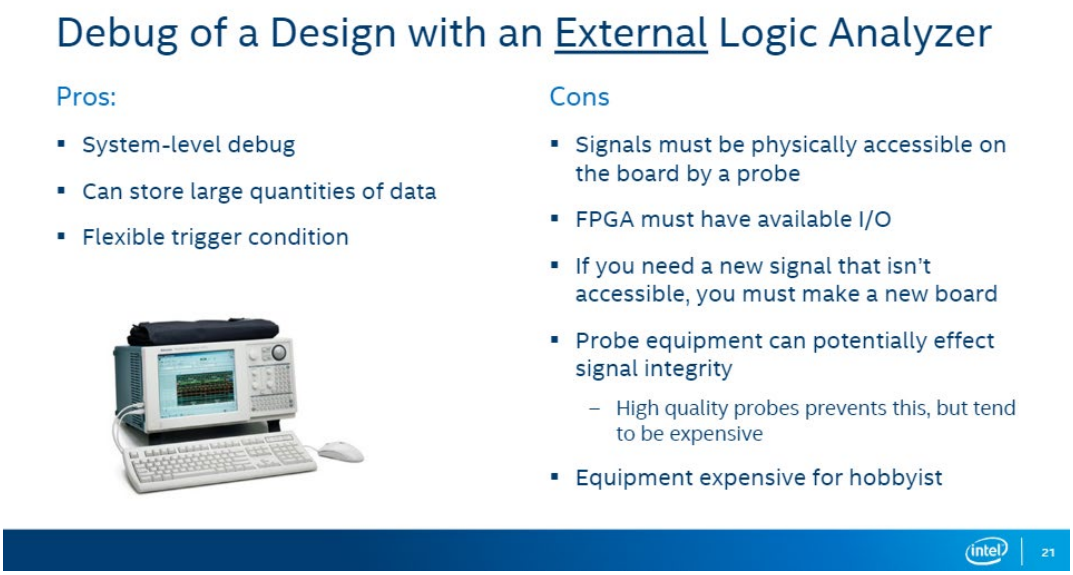
5. Signal Tap

5.1. Why Signal Tap



Signal Tap offers a streamlined approach to hardware debugging by enabling direct signal monitoring on the chip itself, eliminating the need for external equipment, additional pins, or manual stimulus definition. This method often yields significant time and resource savings compared to traditional debugging techniques. However, it's crucial to acknowledge the potential trade-offs. Signal Tap can consume considerable chip memory and may introduce modifications to the design's timing characteristics. Therefore, careful consideration is necessary to ensure its suitability for each specific application.

5.2. Debug of a Design with an external Logic Analyzer



Pros:

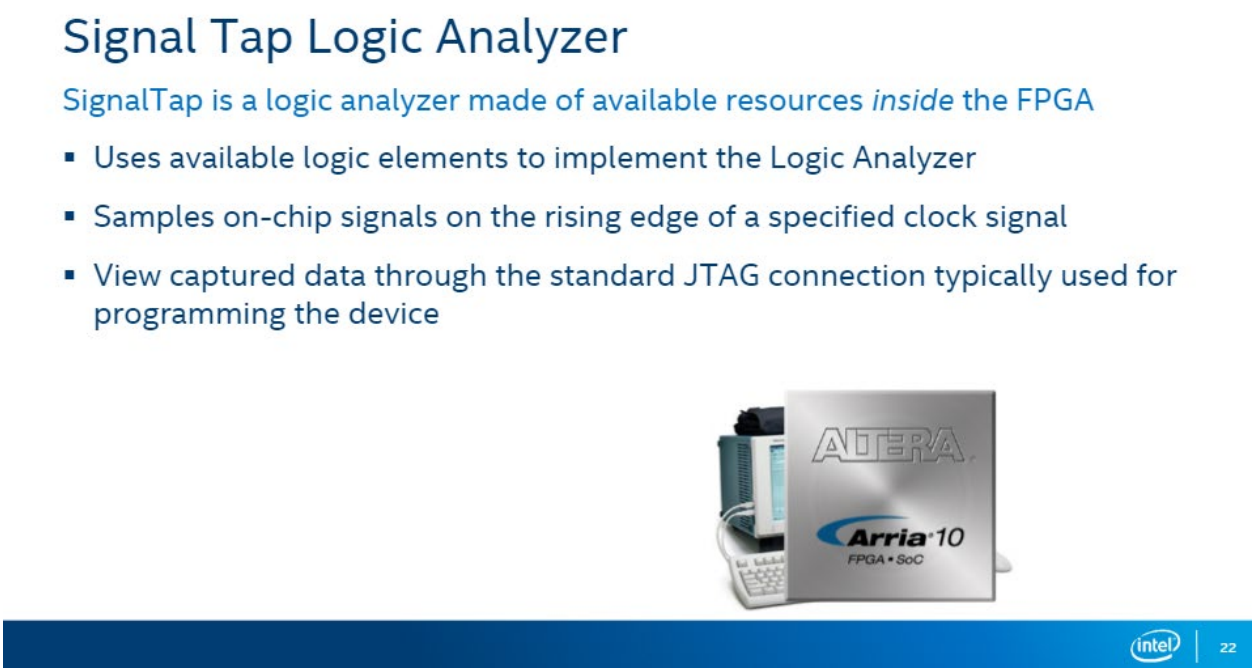
- **System-level debug**: You can examine the interactions between different parts of your design. This contrasts with using simulation, which often focuses on individual components or modules.
- **Large data storage**: Logic analyzers can store large quantities of data. This is important because debugging often involves capturing signals over long periods of time to identify infrequent issues.
- **Flexible trigger conditions**: You can specify exactly what events should cause the analyzer to start capturing data, which can help you isolate specific problems.

Cons:

- **Physical accessibility**: The signals you want to analyze must be physically accessible on the board by a probe. This can be challenging for complex designs where some signals are buried within the chip or difficult to reach.
- **FPGA I/O availability**: The FPGA you are using must have available I/O pins to accommodate the logic analyzer probes. This can be a limitation if your design is already using all the available I/O.
- **Board redesign**: If you need to debug a signal that is not easily accessible, you may need to redesign your board to make it reachable. This can be time-consuming and expensive.
- **Signal integrity**: Probe equipment can potentially affect the integrity of the signals you are trying to measure. This is because the probes add capacitance to the circuit, which can change the signal timing. Using high-quality probes can help mitigate this issue, but they tend to be more expensive.
- **Cost**: Logic analyzers can be expensive, especially for hobbyists.

Overall, an external logic analyzer is a powerful tool that can be helpful for debugging complex designs, it is important to be aware of the limitations and costs involved before using it.

5.3. Signal Tap Logic Analyzer



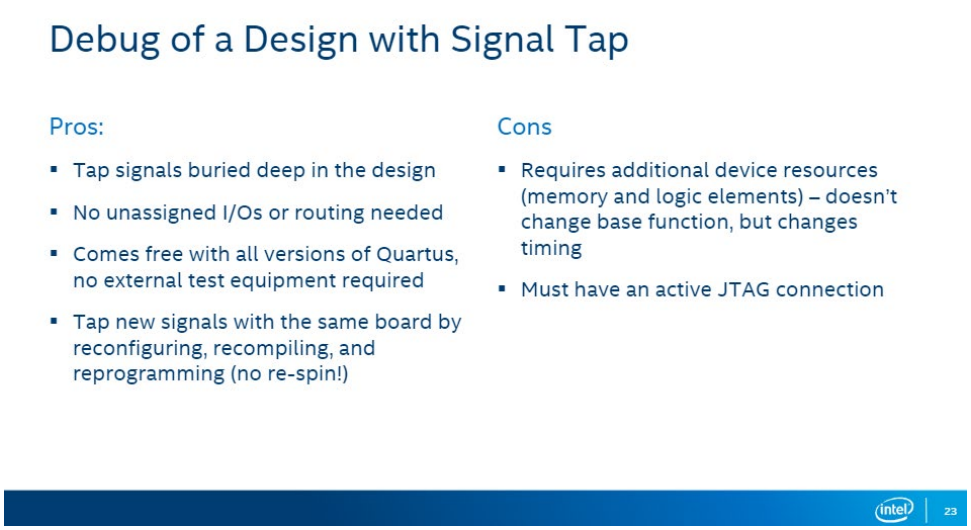
The SignalTap Logic Analyzer is a logic analyzer that uses available resources inside the FPGA, such as the lookup tables, memories, and routing fabric. This eliminates the need for external hardware, making it a convenient and cost-effective solution for embedded system debugging.

- **Built-in tool:** Unlike traditional logic analyzers which are external hardware, SignalTap utilizes resources within the FPGA itself, eliminating the need for extra equipment and connections.
- **Efficient resource utilization:** SignalTap leverages existing logic elements inside the FPGA to perform its analysis, making it an efficient and cost-effective solution.
- **Synchronized sampling:** It captures data samples based on the rising edge of a specified clock signal, ensuring consistency and accuracy in measurements.
- **Easy data access:** You can view the captured data through the standard JTAG connection, which is commonly used for programming the FPGA, simplifying the process of retrieving and analyzing the information.

*Additional information:

- **Potentially lower data capacity**: Since it relies on internal FPGA resources, its data capture depth might be limited compared to external tools with dedicated memory.
- **Possible performance impact**: Extensive use of SignalTap could consume resources and impact the performance of your main design implemented on the FPGA.

5.4. Debug of a Design with Signal Tab



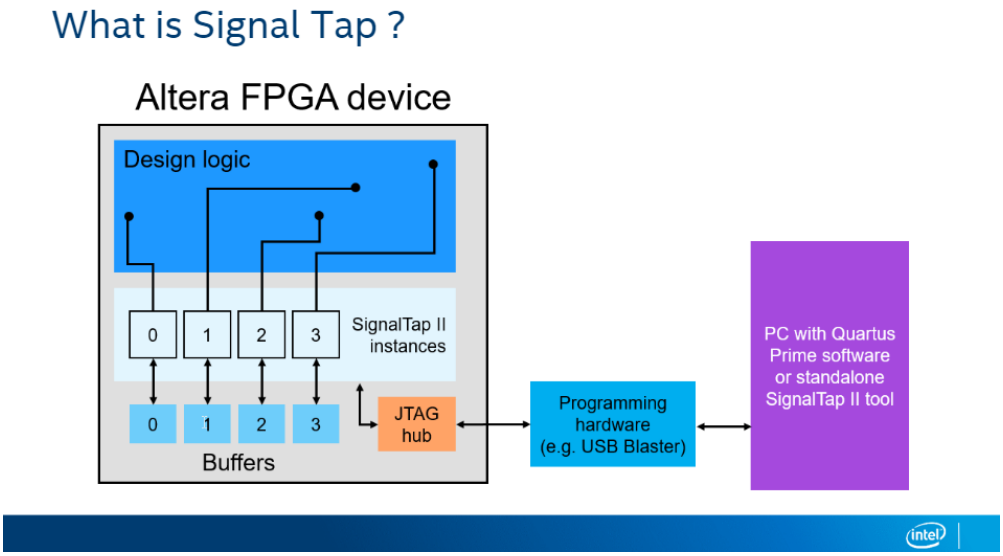
Pros:

- **No unassembled loops or routing needed**: Signal taps can be inserted into a design without modifying the routing or logic of the design. This can save time and effort, and it can also help to avoid introducing errors into the design.
- **Comes free with all versions of Quartus, no external test equipment required**: Signal taps are a built-in feature of the Quartus Prime software, so you don't need to purchase any additional hardware or software to use them.
- **Tap new signals with the same board by reconfiguring, recompiling, and reprogramming (no re-spin!)**: You can easily add or remove signal taps to your design without having to re-fabricate the board. This can be helpful if you need to debug different parts of your design or if you want to make changes to your design after it has been fabricated.

Cons:

- **Requires additional device resources (memory and logic elements)**: Signal taps take up memory and logic resources on the FPGA. This can be a problem if your design is already resource constrained.
- **Must have an active JTAG connection**: You need a JTAG connection to your FPGA to use signal taps. This may not be possible if your FPGA is embedded in a system that does not have a JTAG interface.

5.5. What is Signal Type?



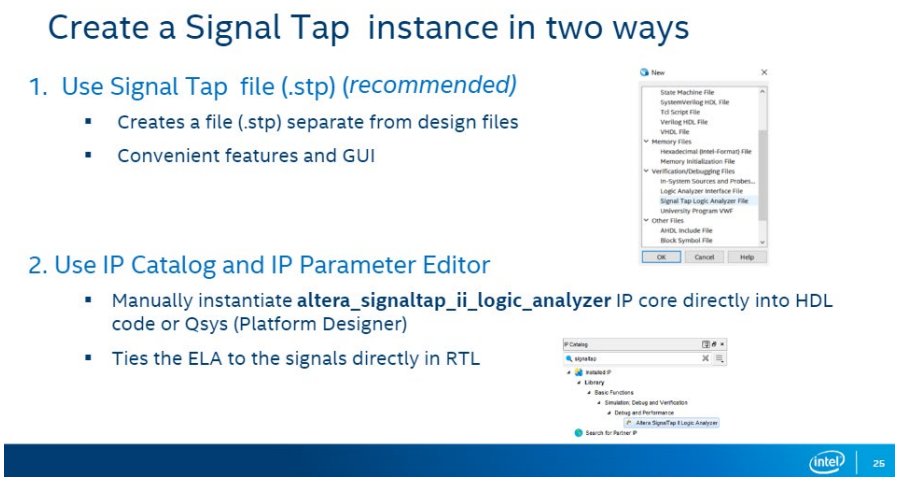
SignalTap II is a powerful embedded logic analyzer (ELA) tool included in the Intel Quartus® Prime software suite. It allows you to debug your designs directly on the target hardware, without the need for external test equipment. SignalTap II can monitor internal signals, trigger on specific events, and capture signal values over time. This makes it an ideal tool for debugging complex FPGA designs.

Here is how it works:

1. SignalTap II is integrated into the FPGA design.
2. The design logic is implemented on the FPGA.
3. SignalTap II monitors the internal signals of the design logic.
4. When a trigger condition is met, SignalTap II captures the values of the specified signals and stores them in memory.
5. The captured data can be analyzed using the Quartus Prime software or the standalone SignalTap II tool.

SignalTap II works by integrating into your FPGA design. It uses logic resources on the FPGA to monitor internal signals and trigger on user-defined events. When a trigger occurs, SignalTap II captures the values of the specified signals and stores them in memory. You can then use the SignalTap II software to analyze the captured data to identify and debug problems in your design.

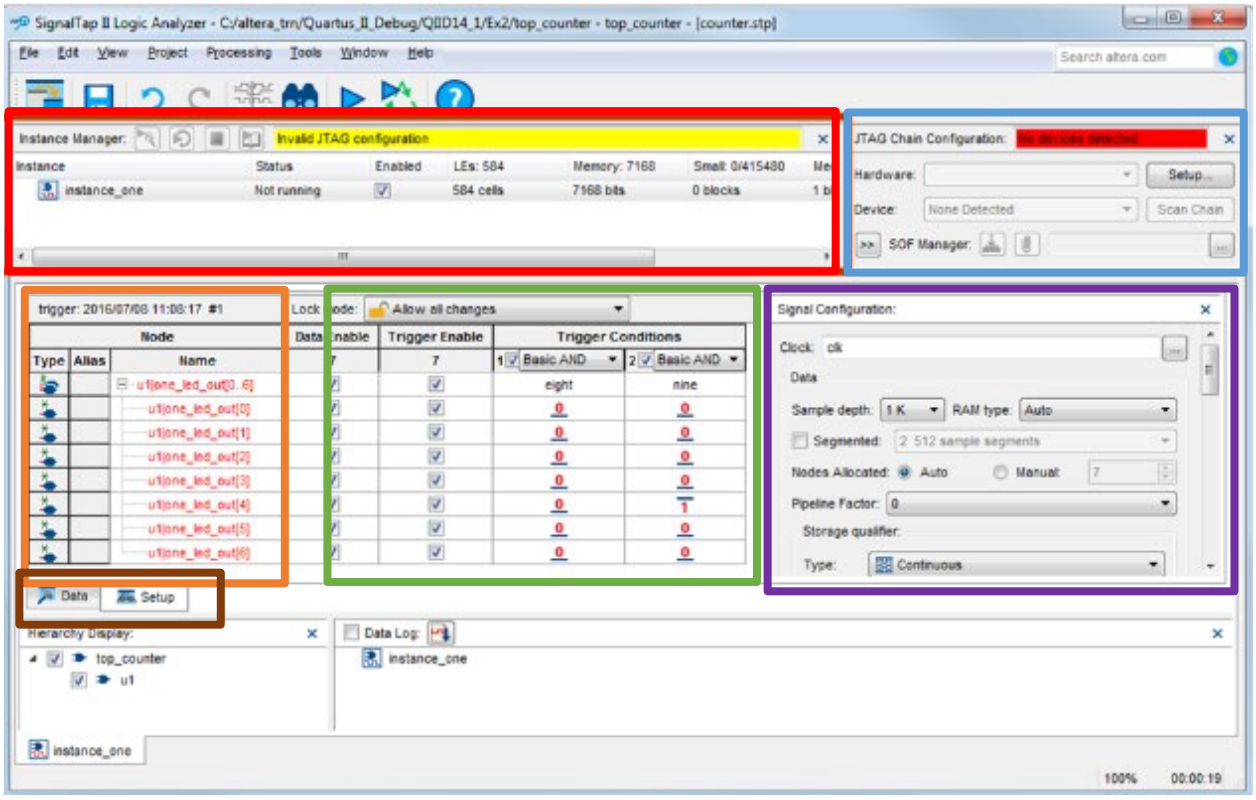
5.6. Create a Signal Tap instance in two ways



There are two ways to create a Signal Tap:

Method	Pros	Cons
Signal Tap file (.stp)	Easy to use, good for collaboration	Not as flexible as manual instantiation
IP Catalog and IP Parameter Editor	More flexible	More complex to set up

5.8. Signal Tap Logic Analyzer Window



On the picture below it is shown how Signal Tap Logic Analyzer Window looks like. There are several parts (windows) there:

1) Instance Manager

- Identifies which instance is being edited in the GUI
- Enable/Disable instance quickly.
- Gives status and resource utilization (LEs and memory)

2) JTAG Chain Configuration

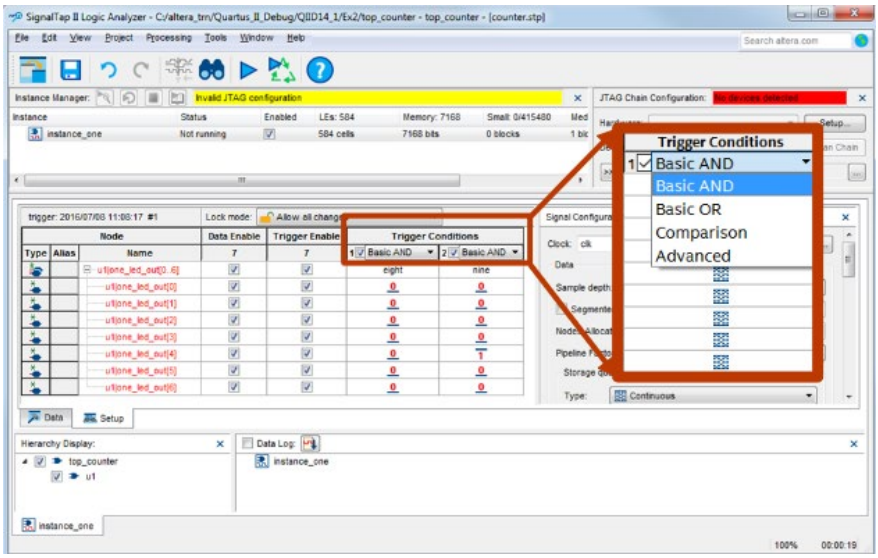
- Built in “Programer”
- Scans JTAG chain and identifies available devices.

3) Nodes List

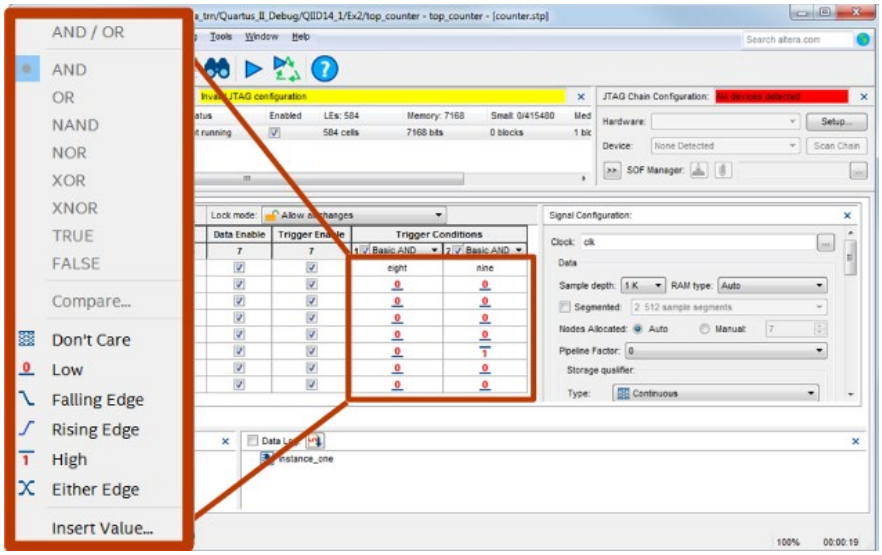
- Use the Node Finder to add signals to be tapped.
- Automatically groups busses together and create custom groups

4) Trigger Conditions and Qualifiers

- Data Enable: Saves signal data (disable to save memory)
- Trigger Enable: Signal is part of the trigger condition (disable to save less)



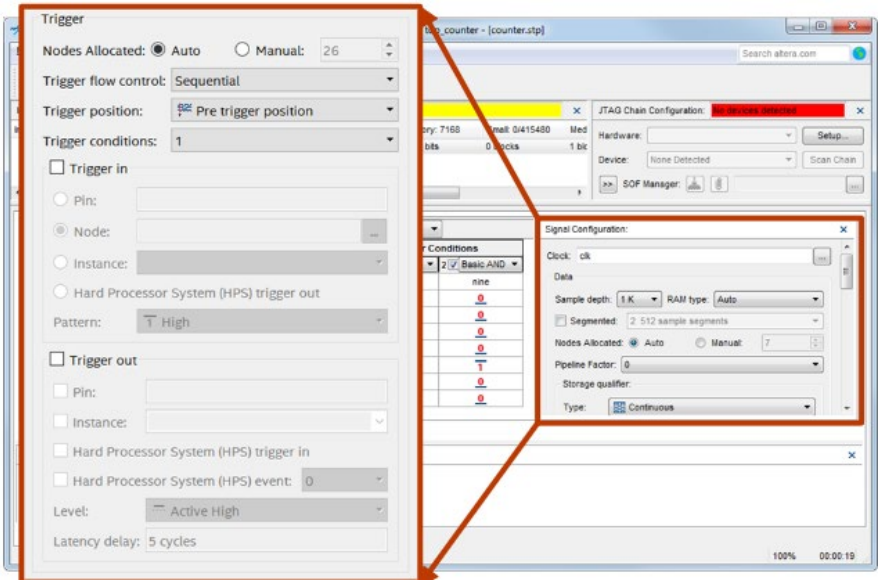
- Add up to 10 trigger conditions.
- Choose how every node is compared.



- Add up to 10 trigger conditions.
- Choose how every node is compared.
- Choose what action triggers a specific node.

5) Signal Configuration

- Select which clock runs the instance.
- Sample Depth: how much data from each signal is stored.



- Advanced trigger control
- Select the number of triggers conditions.
- Trigger In/Out options.

6) Data/Setup Window

- Setup allows configuration of nodes and trigger conditions (for making edits)
- Data shows the acquired signal information (for viewing results)

Conclusion

In conclusion, this presentation has provided a comprehensive exploration of the ModelSim GUI's essential tools, empowering you to navigate the intricacies of design verification and debugging. We commenced by examining the foundational aspects of simulation, elucidating its potential for testing designs under diverse scenarios. Following that, we embarked on a journey into the domain of In-System Sources and Probes (ISSP), unveiling their remarkable ability to seamlessly interact with live hardware, forging a vital link between simulation and real-world functionality. Finally, we delved into the potent capabilities of SignalTap, underscoring its embedded logic analysis prowess for meticulously dissecting internal signals and pinpointing design issues with utmost precision.

By adeptly wielding these ModelSim GUI essentials, you are equipped to streamline your design workflow, elevate the rigor of your verification processes, and bring your creations to life with unwavering confidence. It is crucial to remember that simulation extends beyond mere bug identification; it represents a proactive endeavor to guarantee design quality, performance, and, ultimately, the resounding success of your projects.