

Virtual JTAG based Scan Chain to test Krypton CPLD

Abhinav Sridhar
Wadhwani Electronics Lab
IIT Bombay



Objectives

- Need for scan chain
- Basic understanding of JTAG and Scan Chain
- What is Virtual JTAG IP
- How Scan Chain works
- How to Run Scan Chain using virtual JTAG
- Some FAQs



Why Test on Hardware?

- Why to test on Hardware?
- We have written a testbench and thoroughly tested it with a tracefile
- Not just RTL but also Gate level simulations show zero errors in design
- Why do we need test our design on hardware again?



Why Test on Hardware?

- Why to test on Hardware?
- We have written a testbench and thoroughly tested it with a tracefile
- Not just RTL but also Gate level simulations show zero errors in design
- Why do we need test our design on hardware again?
- Gate Level files merely mimic the hardware
- Hardware may have some other timing differences, or a small faults
- The design may not be synthesizable (it still may work on software)
- All simulated designs may not work perfectly on board



Why Scan Chain?

- Designs tested so far using I/O pins:
 - 1 bit Full Adder, 4:1 MUX / 2:4 Decoder
- Can we test 4 bit Adder-Subtractor on board?
 - Solution: Add 1 more switch
- How to test 4 bit 4:1 multiplexers?
 - Add 10 switches for the 18 inputs
- What about bigger designs like an 16 bit adders?
- How many peripherals to add?



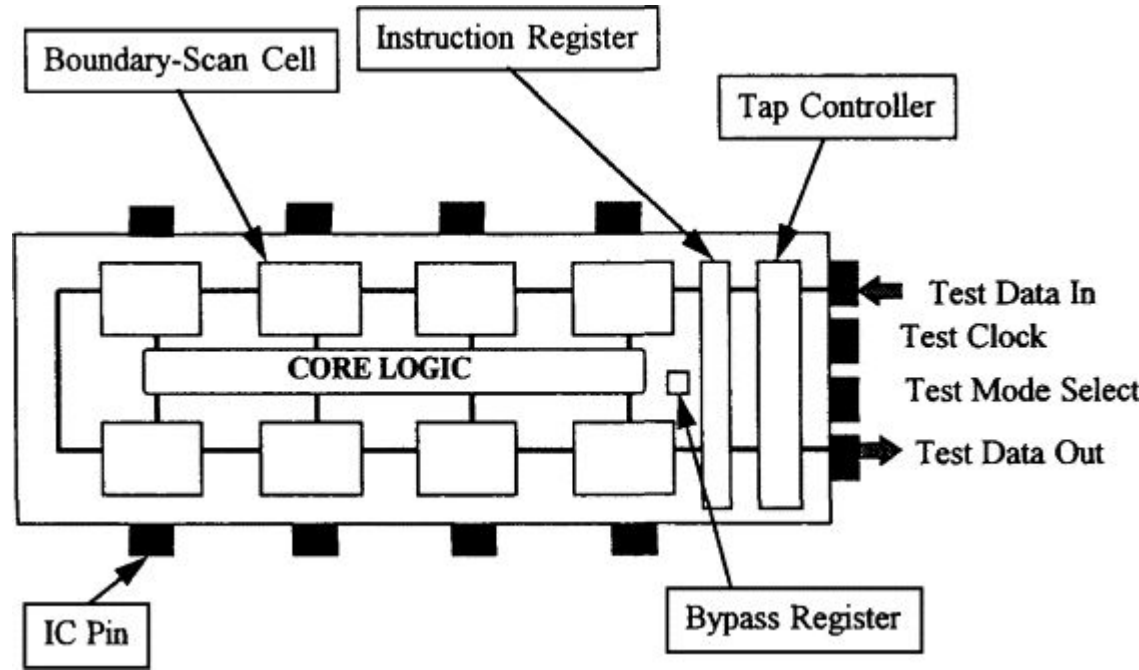
Why Scan Chain?

- Designs tested so far using I/O pins:
 - 1 bit Full Adder, 4:1 MUX / 2:4 Decoder
- Can we test 4 bit Adder-Subtractor on board?
 - Solution: Add 1 more switch
- How to test 4 bit 4:1 multiplexers?
 - Add 10 switches for the 18 inputs
- What about bigger designs like an 32 bit adders?
- How many peripherals to add?
- Also manually testing an input combination may take around 1 second
- For 4 bit 4:1 MUX, 2^{18} sec \Rightarrow 72+ hours
- What about a 32 bit adder?
- $2^{64} = 10^{12}$ YEARS
- How to test designs?
- Solution: Scan Chain



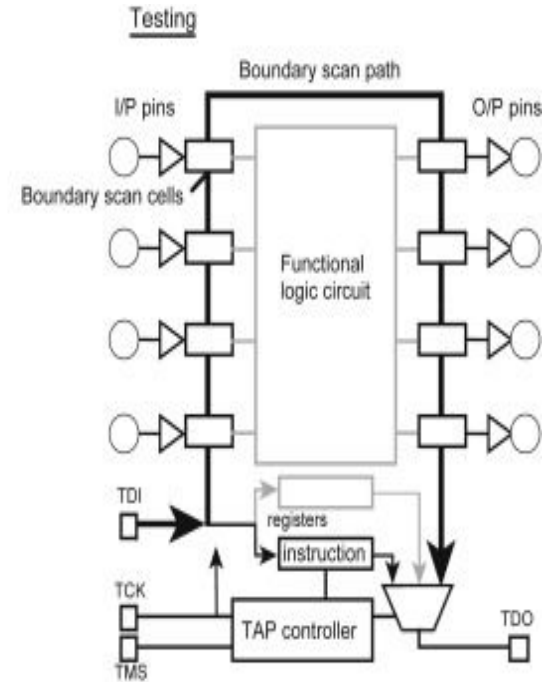
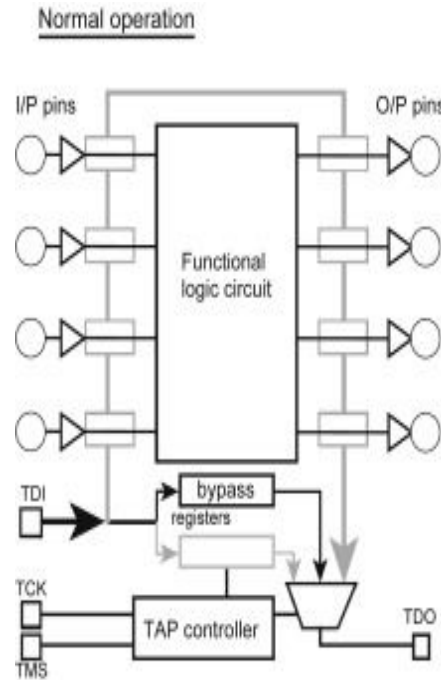
JTAG

- Joint Test Action Group
- Industry standard for verifying designs and testing PCBs
- Connects to an on-chip Test Access Port (TAP) controller
- TAP controller implements a protocol to access a set of registers or pins of a system
- Response is then stored into Flip-Flops and then compared with a golden response



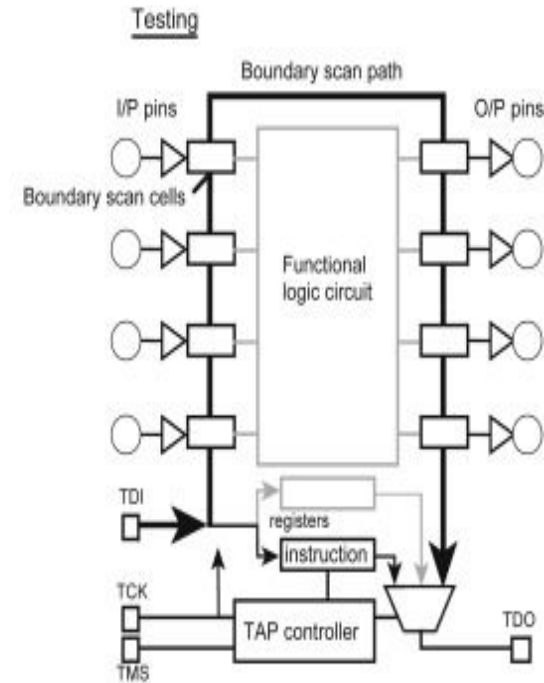
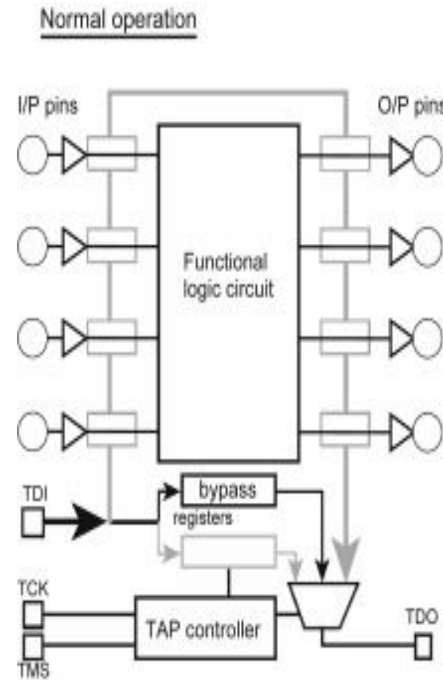
JTAG

- The testing is controlled with the help of the following pins
 - TDI
 - TDO
 - TMS
 - TCLK
 - TRST (optional)
- Normal operation: the test inputs are bypassed, circuit works as specified
- Testing mode: The boundary scan cells take the input set by the TAP controller, and output of each register or wire is stored in the flipflops



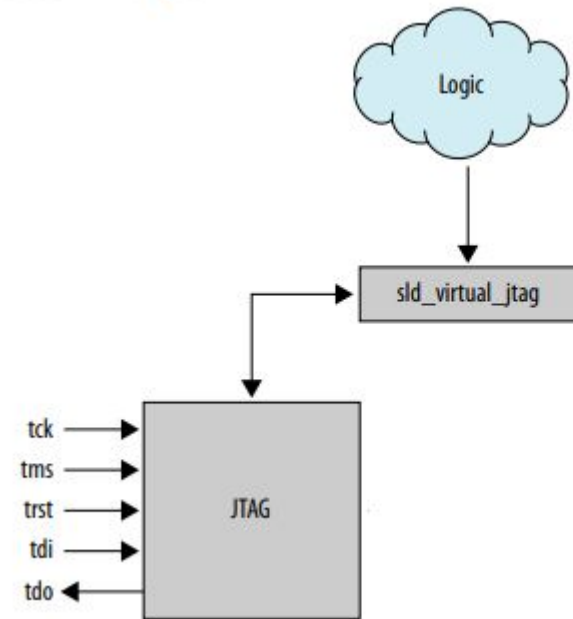
JTAG

- Used to compare it to golden response
- This is also called scan mode
- Flipflops connected in chained manner => SCAN CHAIN
- TDI: Test Data IN
- TDO: Test Data OUT
- TCK: Test Clock
- TMS: Test Mode Select
- TRST: Test RESET (Optional as per the protocol)



Virtual JTAG

- FPGAs already have the basic JTAG Hardware built-in
- To Control: Virtual JTAG Intel FPGA IP Core
- Intellectual Property: source code not visible
- Feature of the on-chip debugging tool suite
- Logic is implemented along with a system level debugging infrastructure for virtual JTAG
- Let us dive a little deeper



DISCLAIMER

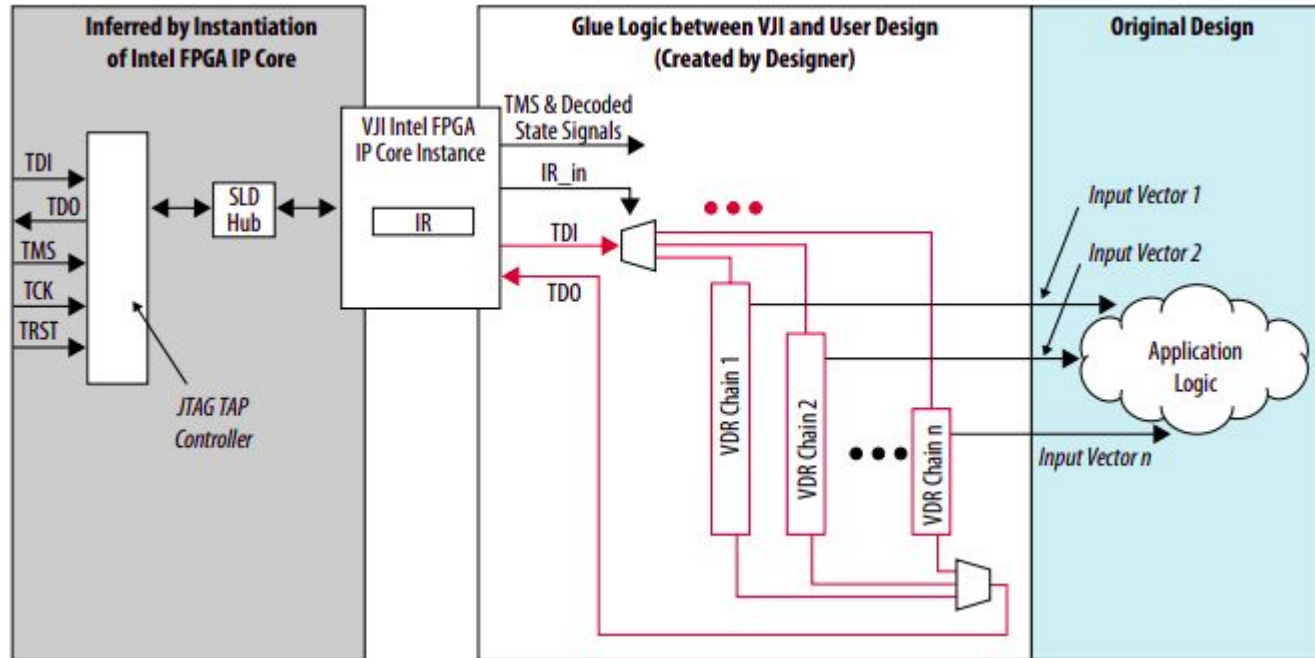
Objective of the course => Digital design using FPGA

Virtual JTAG based scan chain can be viewed as blackbox



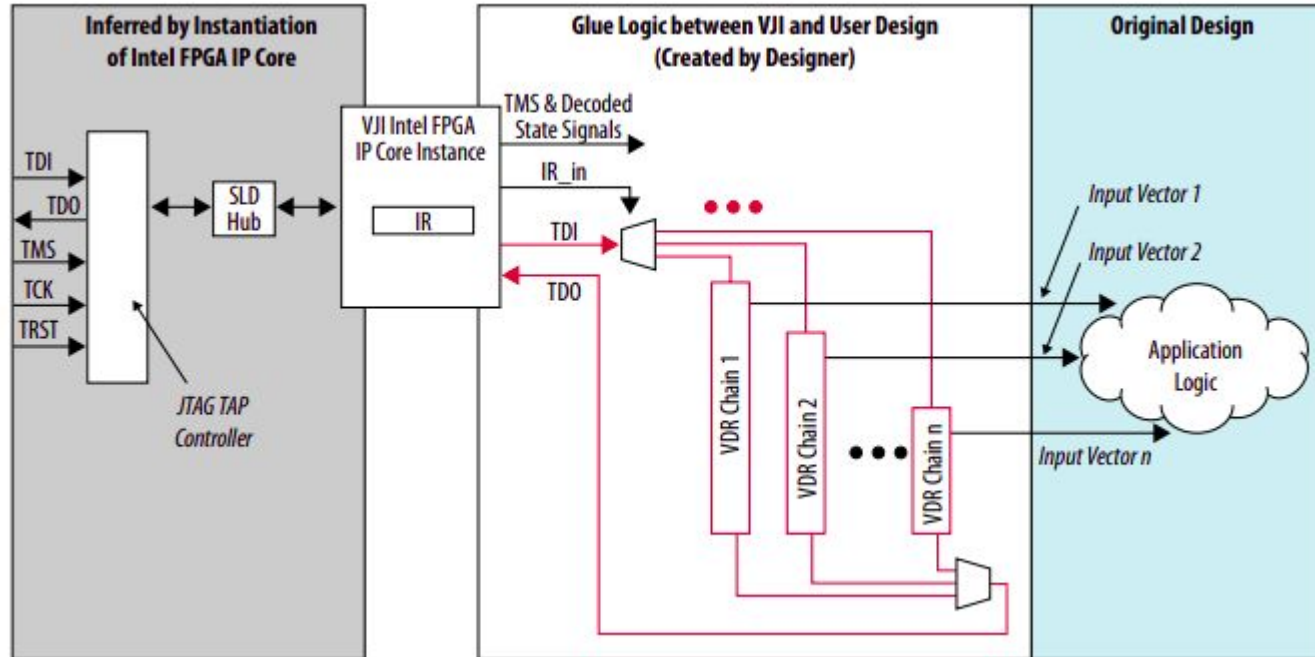
Design Flow

- The original design (wrapped in DUT) is created
- The IP core is instantiated and added to project



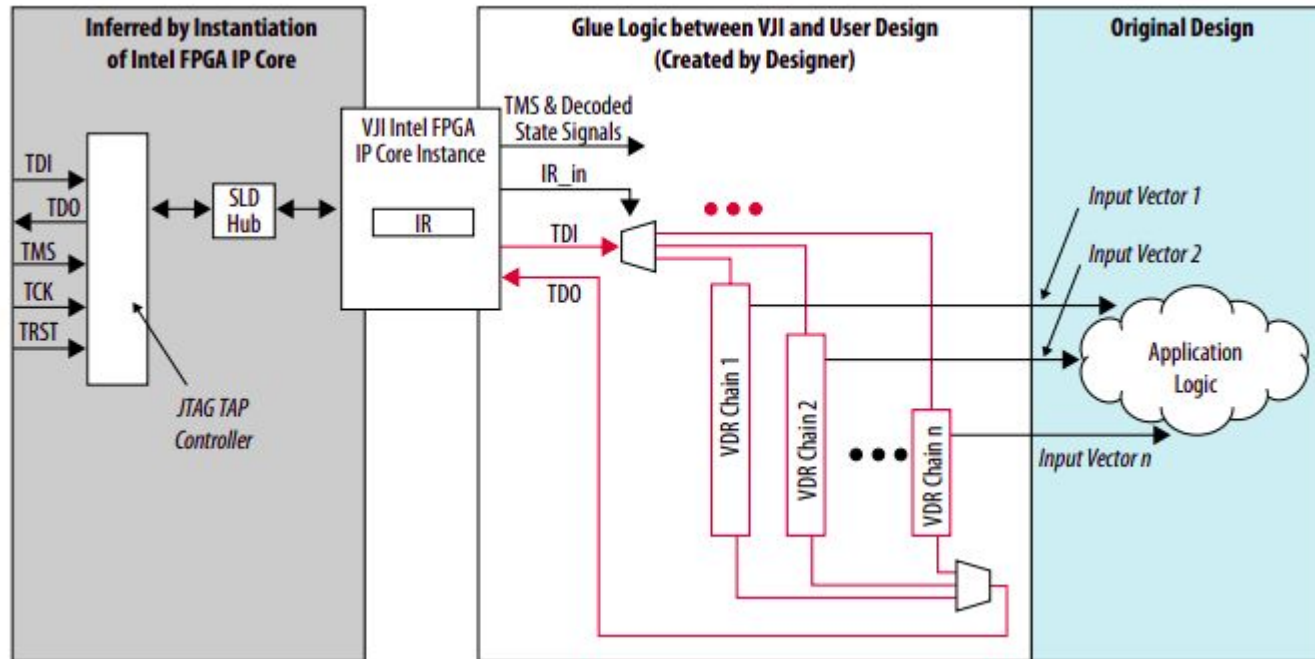
Design Flow

- The IP Core obtains control of the TAP controller
- How to decide inputs?: GLUE LOGIC



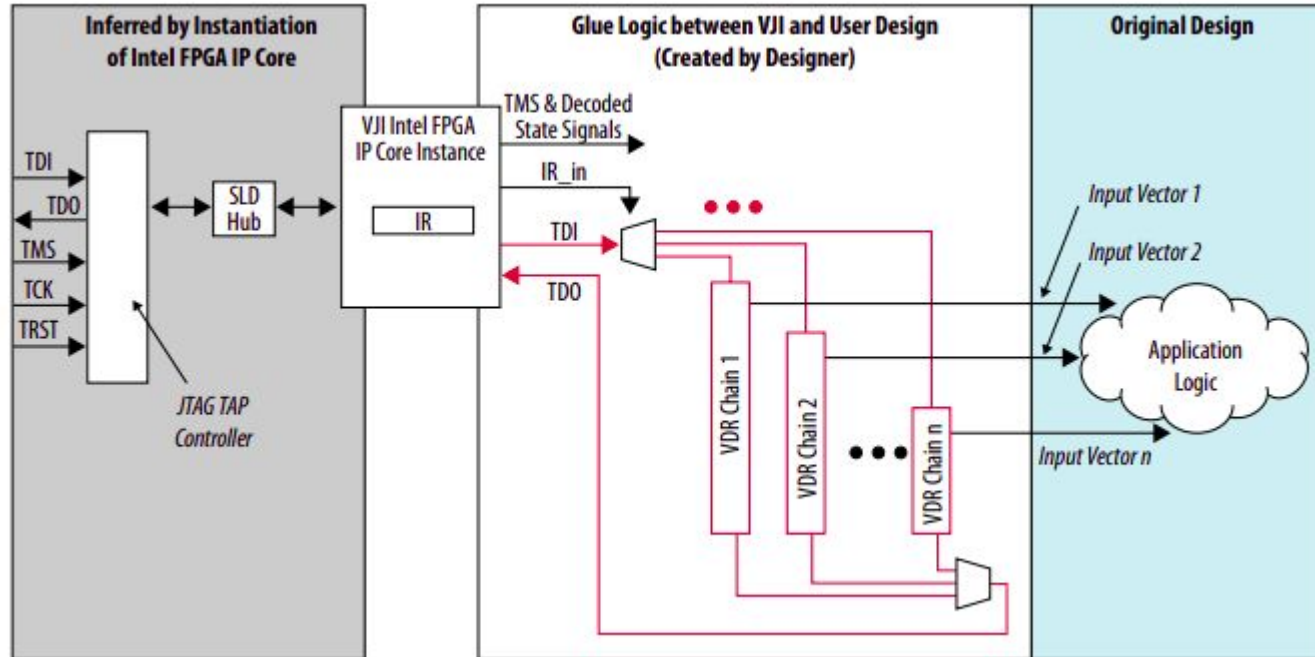
Design Flow

- Glue Logic: Contains Data Registers (holding data), Instruction Register (holding control bits)
- Acts as a pathway for these data and control inputs
- Implementation of Glue Logic: scan_vjtag.py



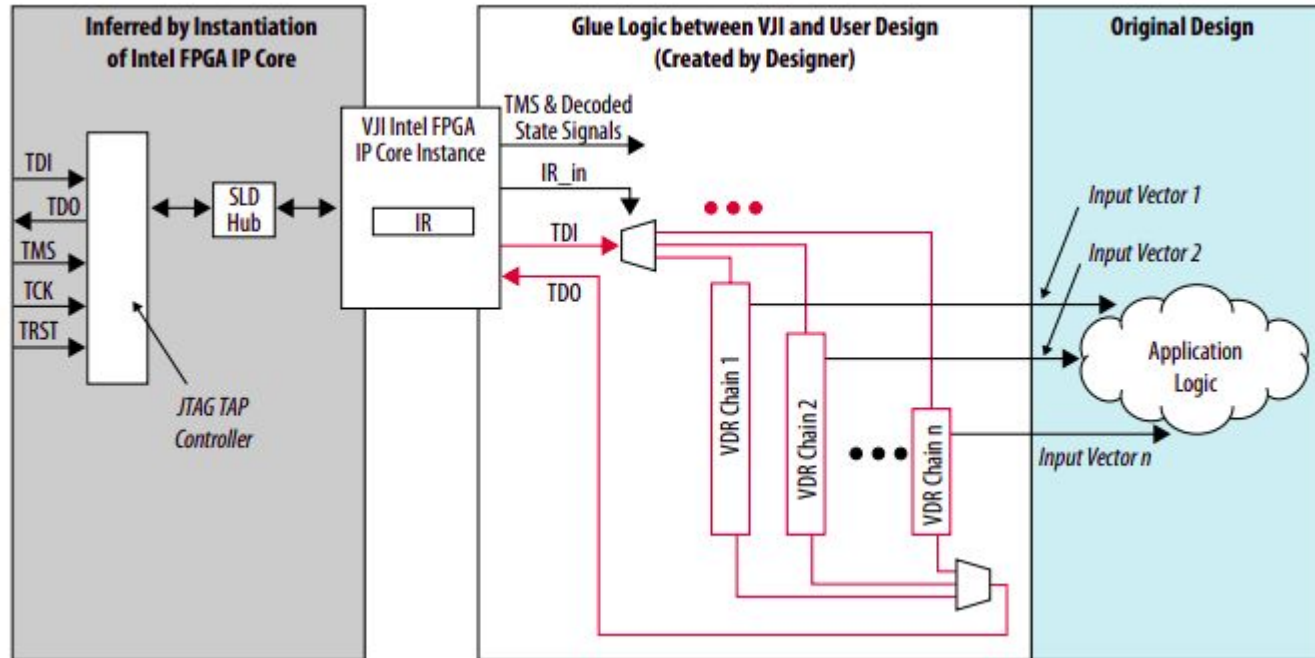
Design Flow

- Sends inputs to open communication with Krypton via the FTD2xx port
- Sets the modes for working of scan chain ([documentation](#))
- Takes inputs from TRACEFILE.txt



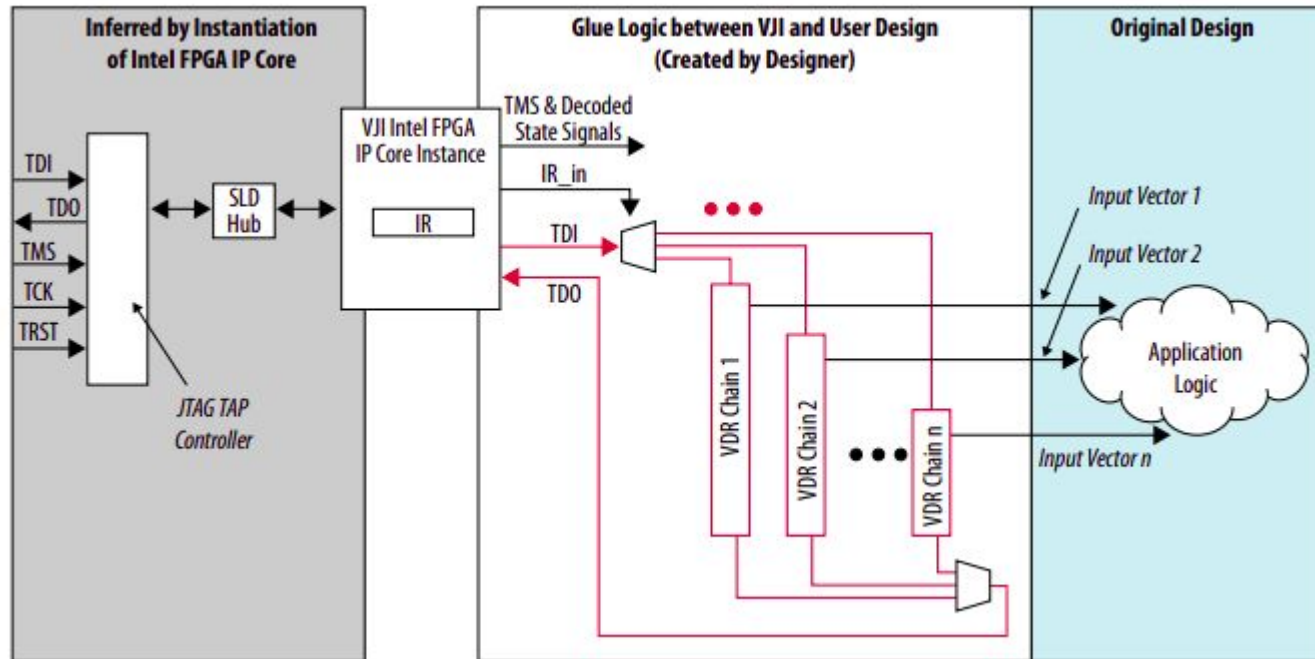
Design Flow

- Passes inputs to Data Registers, Generates Control Register values and passes it to instance
- Core Instance then switches to scan mode, gives inputs to all vectors
- Switches to normal mode, computes the logic on hardware
- Switches back to scan mode and gives back the outputs



Design Flow

- Python then compares received outputs to the golden response in TRACEFILE and updates
- Apart from the standard JTAG I/O pins, the states of the TAP controller are given as output
- Can be used to debug, need not be connected to any hardware unless needed



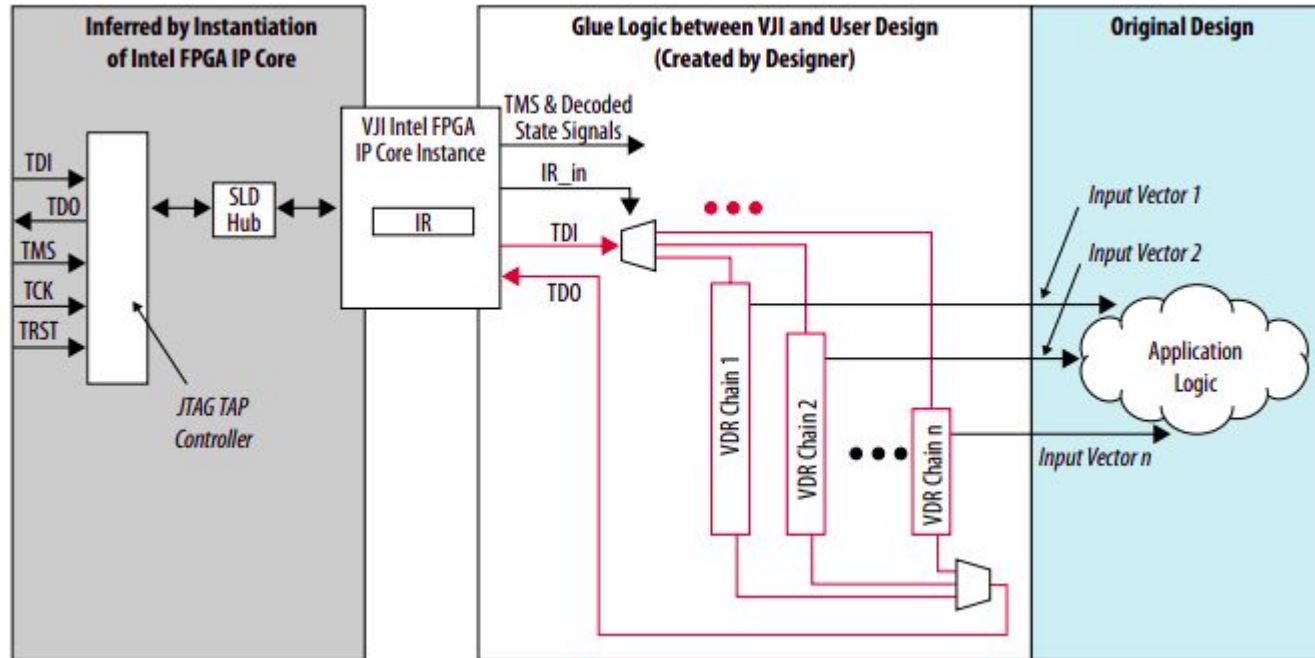
Design Flow

- Python then compares received outputs to the golden response in TRACEFILE and updates
- Apart from the standard JTAG I/O pins, the states of the TAP controller are given as output
- Can be used to debug, need not be connected to any hardware unless needed



Design Flow

- These concepts could sound like Latin and Greek at the moment
- As mentioned earlier, a blackbox type understanding of scan chain would suffice for EE214
- More details in EE705 (VLSI Design Lab)



Steps to Run Virtual JTAG

Design: Vidur Shah, Gaurav Kate

Documentation: Soumyajit Langal, Abhinav Sridhar



Install Python, PIP, Python Packages

- Install Python 3.7 or Python 3.8
- Install PIP
- Install the following packages using the commands as shown
 - `pip install bitstring`
 - `pip install ftd2xx`
 - `pip install pyusb`



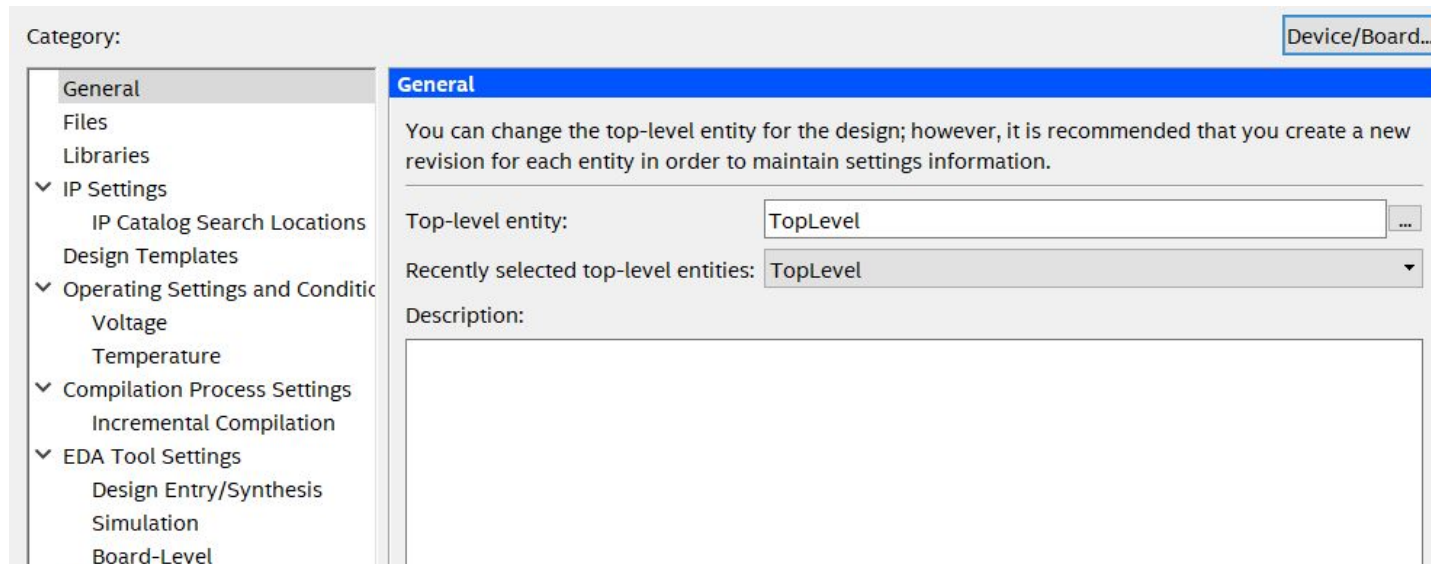
Setup Environment

- Necessary files
 - DUT wrapped design file and other dependent design files with working RTL and Gate level simulations
 - Tracefile
- Files Required for Scan Chain (Shared)
 - TopLevel.vhd
 - Folder v_jatg
 - scan_vjtag.exe file
 - scan_vjtag.py
- Copy these files and paste them in the project directory



Modifications to Quartus Project

- Add TopLevel.vhdl to the Quartus Project
- Add the number of inputs and outputs to the design in lines 20 and 21 respectively
- This change is the same as what was done for testbench
- Go to Assignments -> Settings -> General
- Set top level entity as TopLevel as shown



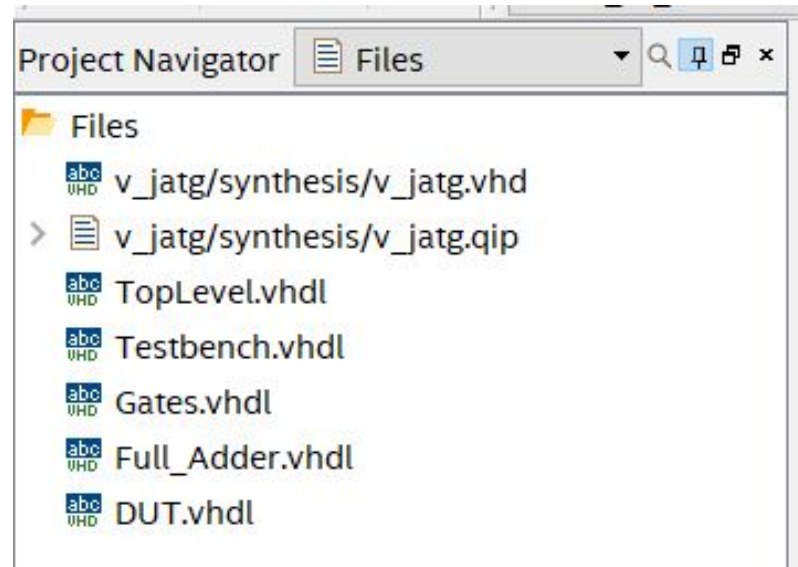
The screenshot shows the 'Settings' dialog box in Quartus, with the 'General' tab selected. The left sidebar lists various categories, with 'General' highlighted. The main area contains the following information:

- Category:** General
- Device/Board...** (button)
- General** (tab title)
- Description:** You can change the top-level entity for the design; however, it is recommended that you create a new revision for each entity in order to maintain settings information.
- Top-level entity:** TopLevel (text field with a browse button '...')
- Recently selected top-level entities:** TopLevel (dropdown menu)
- Description:** (empty text area)



Modifications to Quartus Project

- Add the following files in the path: v_jatg\synthesis
 - v_jatg.vhd
 - v_jtag.qip
- For a sample full adder project, these are the list of files that have to be present
- Note that the presence or absence of testbench in the project does not matter during scan chain



Quartus Instructions

- Do a full compilation of the design (note you need to repeat the task since the top level entity has changed)
- Generate SVF file again
- Dump the SVF file on the Krypton Board
- **TURN OFF THE KRYPTON BOARD AND TURN IT ON AGAIN USING THE POWER SWITCH GIVEN**
- Open Command Prompt and change path



Change Path using CD command

- Open Command Prompt and change path using cd command
- In case the project is on C drive, use the following
 - `cd ..\..\`
 - `cd <absolute path from C drive>`

```
C:\Users\abhin>cd ..\..\n\nC:\>cd intelFPGA_lite\18.1\
```

- This path can be copied from windows explorer
- In case you are in D/E/F drives, enter the following
 - D: or E: or F:
 - `cd <absolute path from D/E/F drive>`

```
C:\Users\abhin>D:\n\nD:\>cd IIT_Bombay\WEL\NPTEL\Scan_Chain_Test\Full_Adder_code_files
```



Run Scan Chain

- Run the following command to observe the output
 - `scan_vjtag.exe TRACEFILE.txt out.txt`

```
D:\IIT_Bombay\WEL\NPTEL\Scan_Chain_Test\Full_Adder_code_files>scan_vjtag.exe TRACEFILE.txt out.txt  
{'type': 6, 'id': 67330064, 'description': b'Dual RS232-HS A', 'serial': b'A'}
```

- `scan_vjtag.exe` is the name of the file to run
- `TRACEFILE.txt` needs to be a text file in the specified format. It is an input to the exe file
- In case you have renamed it, write the new file name in the command
- `out.txt` is a file that is generated by the code. There need not be an existing file with the name. In case there is, the program will overwrite the file without warnings
- Check `out.txt` to find whether your design has successfully passed all test cases or not



Frequently Encountered Errors

- In case you get the below error, turn off the Krypton board and turn it on again
- Then repeat the last command: `scan_vjtag.exe TRACEFILE.txt out.txt`

```
D:\IIT_Bombay\WEL\NPTEL\Scan_Chain_Test\Full_Adder_code_files>scan_vjtag.exe TRACEFILE.txt out.txt
Traceback (most recent call last):
  File "scan_vjtag.py", line 11, in <module>
    dev = ftd.open(0)
  File "ftd2xx\ftd2xx.py", line 100, in open
  File "ftd2xx\ftd2xx.py", line 44, in call_ft
ftd2xx.ftd2xx.DeviceError: DEVICE_NOT_OPENED
[18740] Failed to execute script scan_vjtag
```



Frequently Encountered Errors

- In case the .exe file is treated as a virus by your antivirus software, then give it permission by allowing the file in settings
- If the issue still persists, the scan_vjtag.py file is provided as backup
- Verify installation of the pip packages in slide 12 (enter installation command again and you will get something similar to the following if it is installed successfully)

```
D:\IIT_Bombay\WEL\NPTEL\Scan_Chain_Test\Full_Adder_code_files>pip install pyusb  
Requirement already satisfied: pyusb in c:\users\abhin\appdata\local\programs\python\python38-32\lib\site-packages (1.0.2)
```

- Repeat all steps until the command `scan_vjtag.exe TRACEFILE.txt out.txt`
- Enter the following command instead of that
 - `python scan_vjtag.py TRACEFILE.txt out.txt`

```
D:\IIT_Bombay\WEL\NPTEL\Scan_Chain_Test\Full_Adder_code_files>python scan_vjtag.py TRACEFILE.txt out.txt  
{'type': 6, 'id': 67330064, 'description': b'Dual RS232-HS A', 'serial': b'A'}
```



Frequently Encountered Errors

- In case Quartus throws an error unrelated to your design, then do the following
- Verify the port names in v_jtag.vhd with the component declared in TopLevel.vhd
- Before dumping the SVF file, open pin planner and map it as shown in image (also mentioned in comments of TopLevel.vhdl)
- Compile design again, and proceed from slide 16

Named: *		Edit: ✕ ✓		PIN_51	
All Pins	Node Name	Direction	Location	I/O Bank	
	out state_out[4]	Output	PIN_58	4	
	out state_out[3]	Output	PIN_57	4	
	out state_out[2]	Output	PIN_55	4	
	out state_out[1]	Output	PIN_53	4	
	out state_out[0]	Output	PIN_51	4	
<<new node>>					



THANK YOU

