Scan chain based test setup for the Krypton CPLD board

Rahul C P

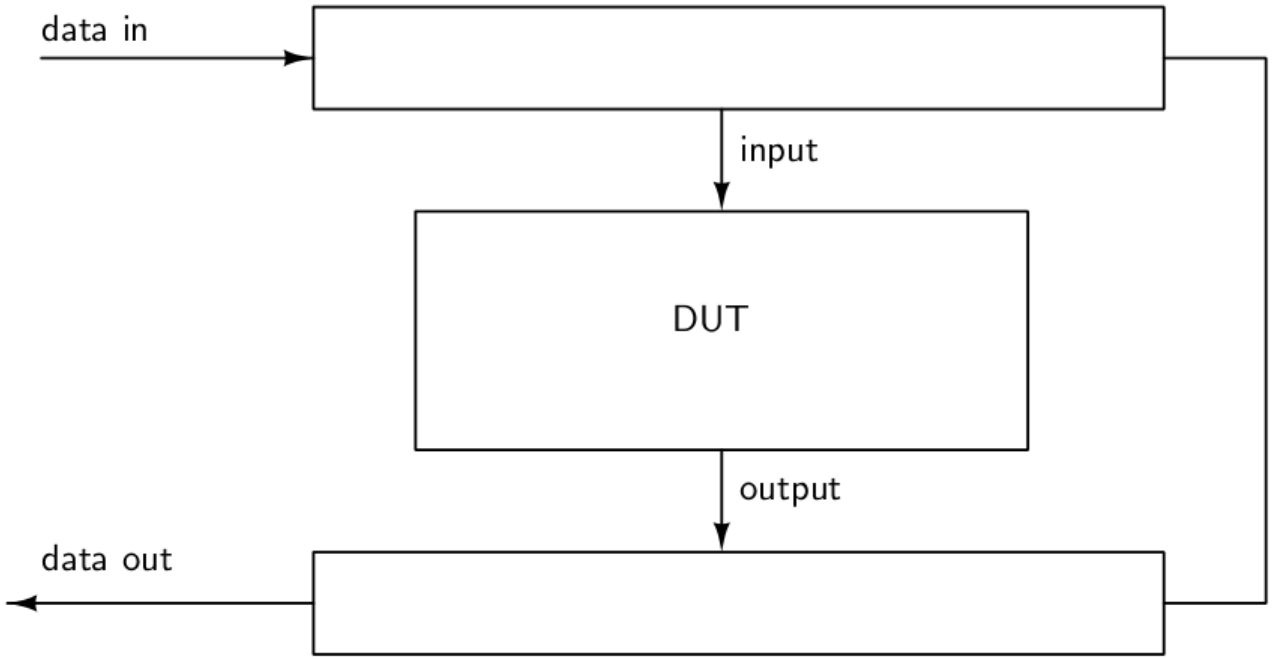
Wadhawani Electronics Lab (WEL)
Department of EE
IIT Bombay

Courtesy:Varun Warrier, Vineesh V S, Titto Thomas

Objectives

- Familiarize yourself with the Scan Chain environment
- Identify connections used for the Scan Chain system
- Create an input file
- Program the CPLD with a design
- Verify the design using the input file and the scan chain environment
- Analyse the output file
- References

Scan Chain



- Scan chain is a technique used for testing the hardware systems.
- a simple way to set the inputs for the system and observe their outputs.
- It consists of two shift registers and their control signals.

Main blocks

- Has three main parts.
  - Python Script : Gets the command inputs from the user in a text file.
  - Microcontroller(TM4C123GH6PM) : Convert these commands into a set of signals for the Krypton board.
  - Krypton board : Contains both the DUT and the scan chain.
- The PC communicates to the microcontroller through a USB link , with a predefined standard data transfer scheme.
- The microcontroller will translate the commands, and generate corresponding signals through its port pins.

Block Diagram

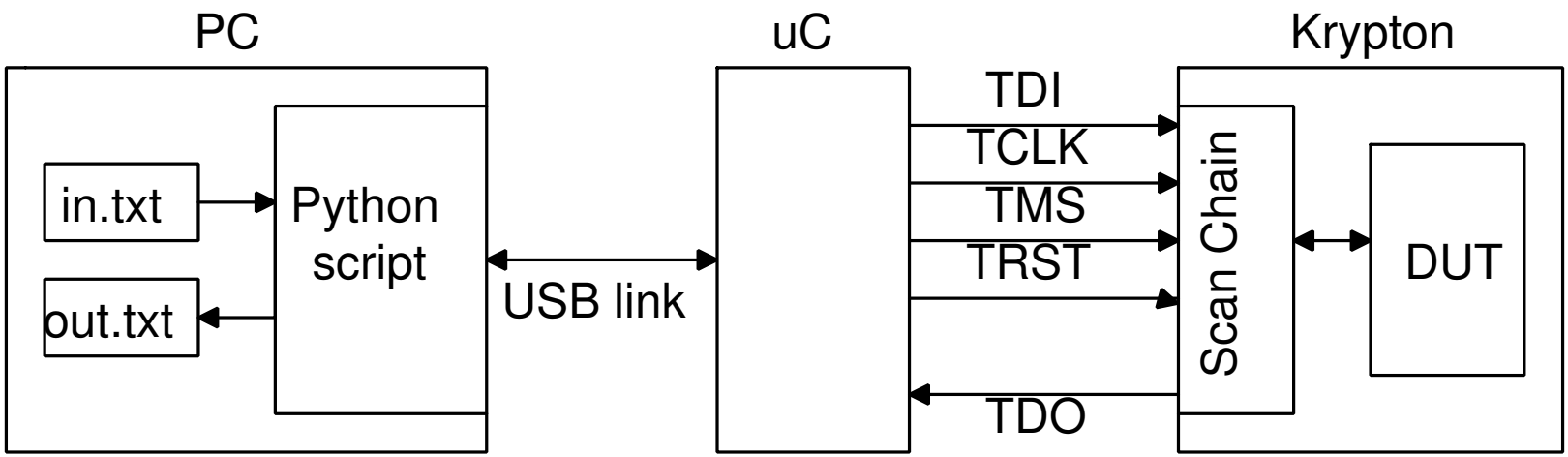


Figure: Tester Architecture

Interface Signals

- The tester hardware contains the scan chain and the controller (TAP controller) that provides necessary control signals to it.
- The interface signals of this top level system would be
  - TDI : The serial test data input to be loaded in the scan chain.
  - TMS : The commands for the TAP (Test Access Port) controller are passed serially through this pin.
  - TCLK : The clock reference forin design for testing all the other communication lines.
  - TRST : Pin to reset the TAP controller at any instant.
  - TDO : The serial data output from the scan chain.

Adding Scan Chain

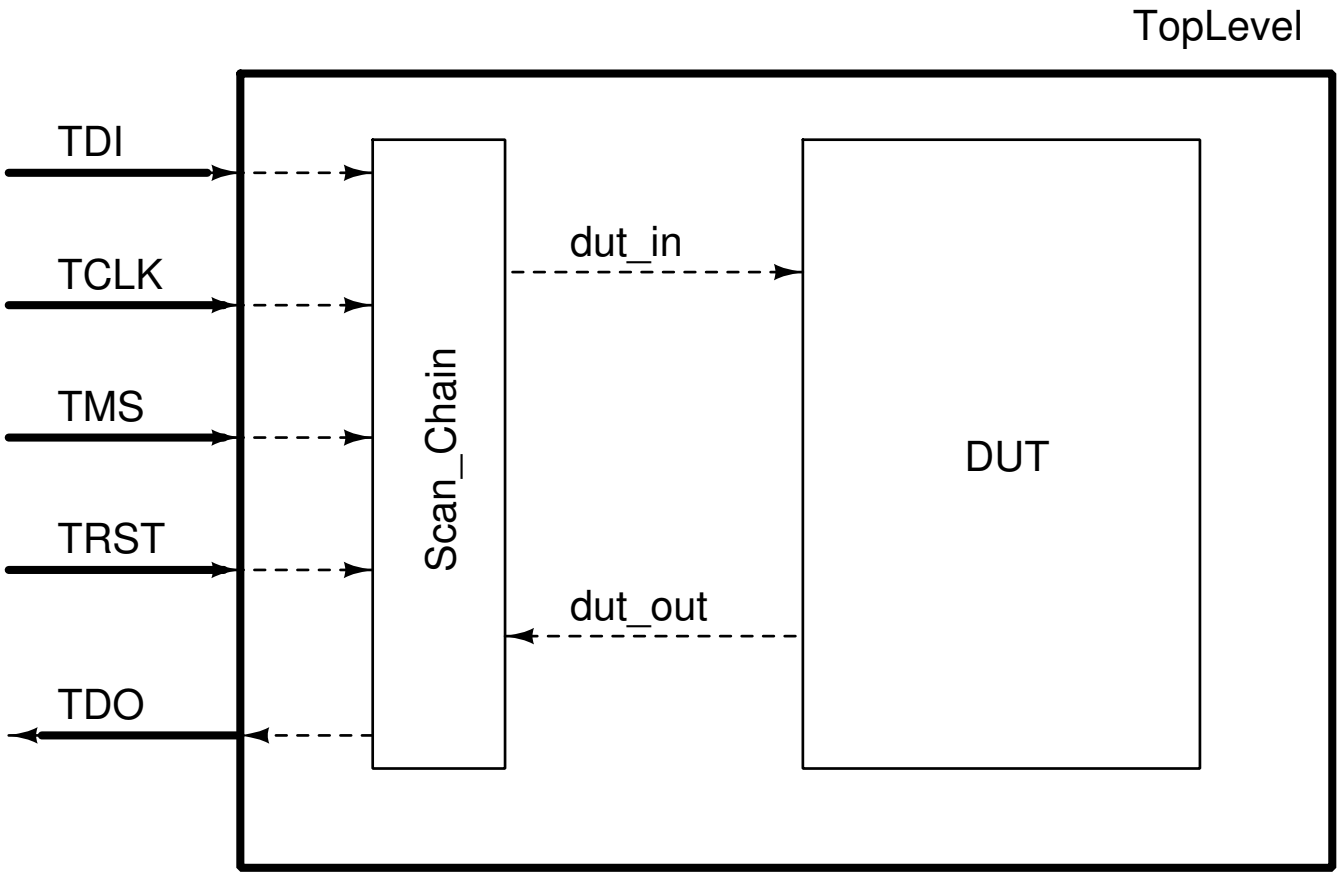


Figure: Top Level entity to be implemented into the Krypton CPLD

Adding Scan Chain(contd.)

- DUT should first be tested in gate level simulation and verified to be working.
- User has to write a top level entity (shown as TopLevel) which contains the DUT and Scan Chain module as component
- TopLevel should have 5 interface signals, TDI, TMS, TCLK, TRST and TDO.

Scan Chain Specifications

```
entity Scan_Chain is
generic (
in_pins : integer; -- Number of input pins
out_pins : integer -- Number of output pins
);
port (
TDI : in std_logic; -- Test Data In
TDO : out std_logic; -- Test Data Out
TMS : in std_logic; -- TAP controller signal
TCLK : in std_logic; -- Test clock
TRST : in std_logic; -- Test reset
dut_in : out std_logic_vector(in_pins-1 downto 0)
-- Input for the DUT
dut_out : in std_logic_vector(out_pins-1 downto 0
-- Output from the DUT
);
end Scan_Chain;
```

Scan Chain specifications (Contd.)

- Scan chain has two configurable parameters (in pins and out pins) indicating the number of input and output bits to the DUT, which can be generic mapped.
- It also has one output (dut in) and one input (dut out) that should be connected to the DUT
- Internally it contains an FSM (that implements the TAP Controller), one input scan register and one output scan register.

Overall Steps for Implementation

- Make your circuits design and connecting it with the DUT wrap,do the gate level simulation. Add the scan chain related designs files.
- Do the pin planning ,compile the code and flash the program into the Krypton
- Make the hardware connections between Tiva and Krypton as well as the PC.
- Generate the input file for testing from the trace file
- Run the python script for scan chain testing.

Reference Implementation : Full adder

- DUT should first be tested in gate level simulation and verified to be working.
- Now user have to add the three files provided: TopLevel.vhdl , scan_chain.vhdl and scan_reg.vhdl in your project and assign the top level entity to "TopLevel".
- Change the *number_of_inputs* and *number_of_outputs* values in the TopLevel.vhdl.
- Compile the new design .

Pin Planning and Programming CPLD

- After successful compilation of your design,do the pin assignment as per following table :

Node Name	Location
TDO	PIN_3
TDI	PIN_5
TMS	PIN_7
TRST	PIN_21
TCLK	PIN_23

Table: Pin Planning

- Compile the design again and create the *.svf* file.
- Flash the *.svf* file into CPLD using *UrJTAG*.

Hardware connections

- Next step is to make physical connections between the host PC, microcontroller board and the Krypton board.
- The microcontroller board is Tiva-C (TM4C123GXL based) from Texas Instruments. Connect it to the PC. (Tiva-C has 2 ports for USB connection. Make sure you use the **Device Port**)
- The following connections between the microcontroller board and the Krypton need to be made

CPLD Pin	Tiva-C Pin	Function
Header 0: 3	J1:PB5	TDO
Header 0: 5	J1:PB0	TDI
Header 0: 7	J1:PB1	TMS
Header 0: 13	J1:PB4	TRST
Header 0: 15	J1:PA5	TCLK
Header 0: 25	J2:GND	GND

Table: Hardware Connections

Hardware connections

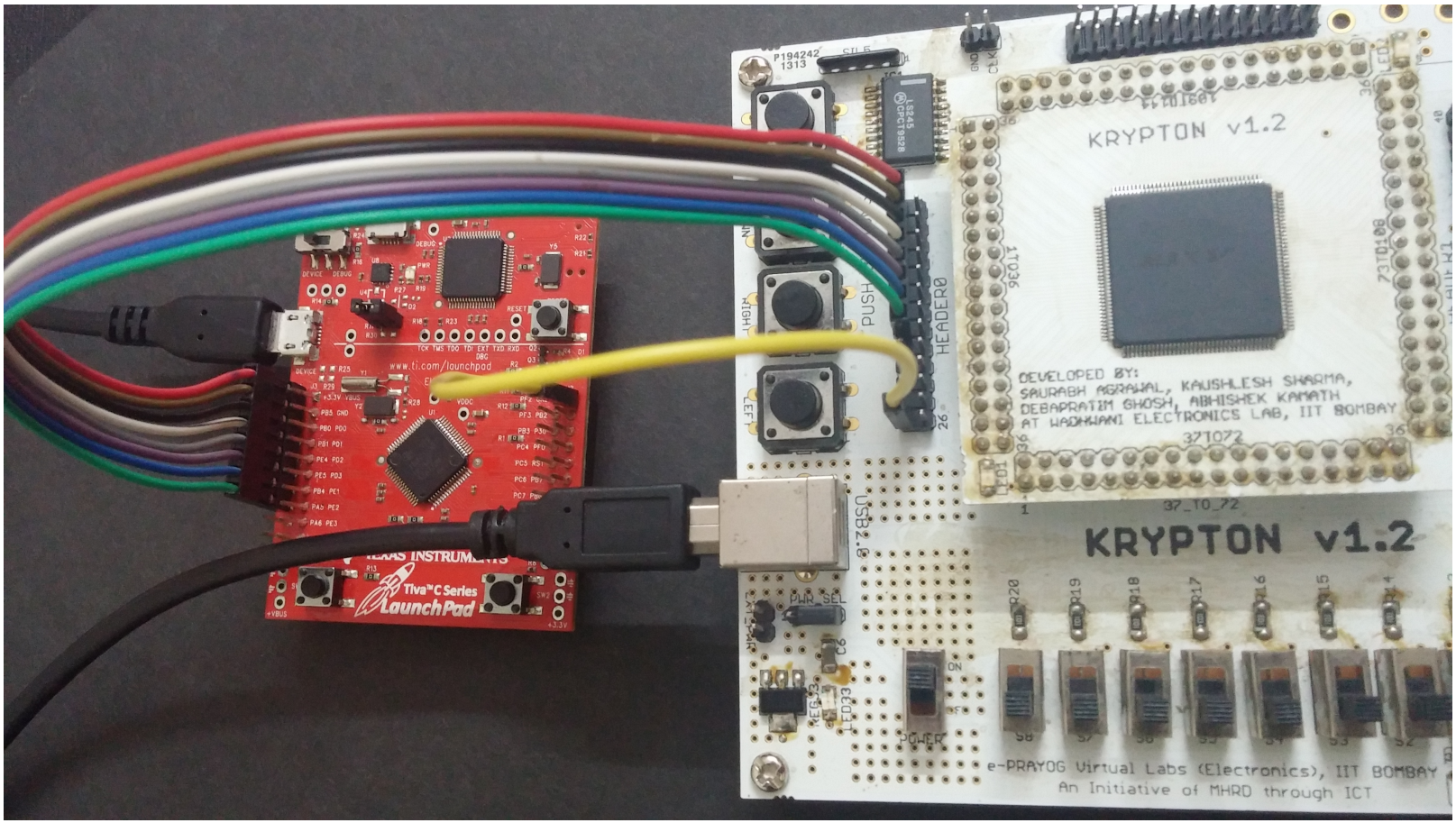


Figure: Hardware Connection between CPLD and Tiva-C using 8-PIN Female Connector

Input file format

- For testing the hardware, the user has to provide input combinations, their expected results and the time duration of execution.
- They should be written as commands in a text file and passed to the python script for test execution.
- These commands are derived from the Serial Vector Format (SVF), usually used in JTAG boundary scan.
- Only two commands are required for the current implementation.

SDR

SDR <*inpins*> TDI(<*input*>) <*outpins*> TDO(<*output*>) MASK(<*maskbits*>)

- This Serial Data Register instruction is for carrying out a data scan in process.
- in pins & out pins contains the number of input and output bits respectively. (**decimal** format)
- input & output contains the input combination to be applied and its expected output combination respectively.(**hex** notation)
- mask bits are used to specify if any of the output bits are not important and could be taken as dont care (**hex** notation)

```
Example :  SDR 2 TDI(0) 8 TDO(00) MASK(FF)
```

SDR(Contd.)

Note :

- If the scanned output should not be compared, then all the mask bits should be kept as 0.
- The mask bits used in our Scan Chain flow should be written nibblewise and the values can be "F"(i.e.0b1111) or "0"(i.e. 0b0000).
- `Example : SDR 3 TDI(2) 2 TDO(2) MASK(F)` –*if we want to compare the two output bits*

RUNTEST

RUNTEST <delay >MSEC

- As the previous instruction loads the input and samples the output, this instruction is used to apply the input combination to the DUT and wait for delay seconds.

Example : RUNTEST 60 MSEC

- The input, output and mask bits are to be written as hexadecimel numbers (uppercase for alphabets).
- An example input file is given in the supporting document.

Conversion from Trace File to Scan Chain Trace file

- As explained earlier we should have proper format for the file to run with TIVA hardware.
- We will convert the generated tracefile to proper format using python script.
- Open the terminal and type
- **python scan_input.py TRACEFILE.txt in.txt #input #output**
- After successful compilation in.txt will be generated which can be use further to test our design with scan chain.

Running the Python script

- First install pyUSB library on the PC by the following steps.
  - open terminal
  - type **sudo apt-get install python-pip**
  - type **sudo pip install pyusb**
- Now run the scan.py script with the following command.

sudo python scan.py <input_file> <output_file> tiva

Where <input_file > contains all the commands to be executed, and <output_file > should be an empty file for storing the results.

- To debug errors after complete test run, you can analyse the <output_file>generated which contain the list of "Expected Output", "Received Output" and the Remark, ie Success/Failure status of each input cases.

Thank you