Scan chain based testing setup for CPLD / FPGA based systems

Advisor : Prof. Madhav P. Desai

Titto Thomas
Wadhwani Electronics Laboratory
EE Dept. IIT Bombay

March 24, 2015

Abstract

This system is designed to provide a testing setup for any CPLD / FPGA based system. The user will be able to load the input combinations, and apply them on the system. Also, the output could be taken out and compared against the expected value. The tester module has to be added along with the user's device (DUT) on the hardware, and the pins should be connected to the microcontroller. Then, just by giving the input file containing the commands the test will be carried out and the results will be displayed.

1 Introduction

Scan chain is a technique used in design for testing. The objective is to make testing easier by providing a simple way to set the inputs and observe the outputs. For simplicity we can assume them to be a serial in parallel out shift register at the input side and a parallel in serial out shift register at the output side of the Device Under Test (DUT).

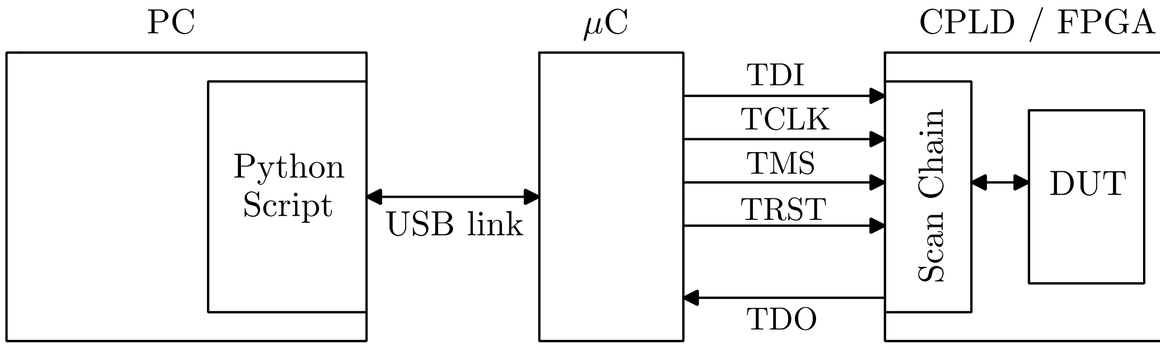

Figure 1: Block Diagram

Joint Test Action Group (JTAG) is a standard for testing hardware using scan chains. A simplified version of standard JTAG is proposed here for testing any CPLD / FPGA based system. The system is aimed to test the digital system design, i.e. send input combinations to the hardware and read back their outputs. The complete system has mainly three parts. The first one is a software running on the host PC that gets the command inputs from the user from a text file. Second one is a microcontroller that converts the commands (obtained from PC through USB) into a set of signals for the CPLD / FPGA board. The final one is the CPLD / FPGA board containing both the DUT and the proposed scan chain.

The user can provide input combinations and their expected outputs in an input text file. The python script will parse the file and extract the commands. 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 it's port pins. The hardware description of DUT and the tester hardware are bundled together and implemented on the FPGA / CPLD board. The tester hardware contains the scan chain and the controller (TAP controller) to control it. The interface signals are

- `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 for all the other communication lines.
- `TRST` : Pin to reset the TAP controller at any instant.
- `TDO` : The serial data output from the scan chain.

Further design details and specifications are explained in the following sections.

2 Adding scan chain to your VHDL design

The user have to write a top level entity (shown as `mySystem` in fig.2) which use the DUT as a component. Along with DUT it will also contain `Scan_Chain` module as a component, which contains the entire testing hardware. The top level entity will have only these two components communicating with each other. `mySystem` will have 5 interface signals (1 bit each) `TDI`, `TMS`, `TCLK`, `TRST` and `TDO`.

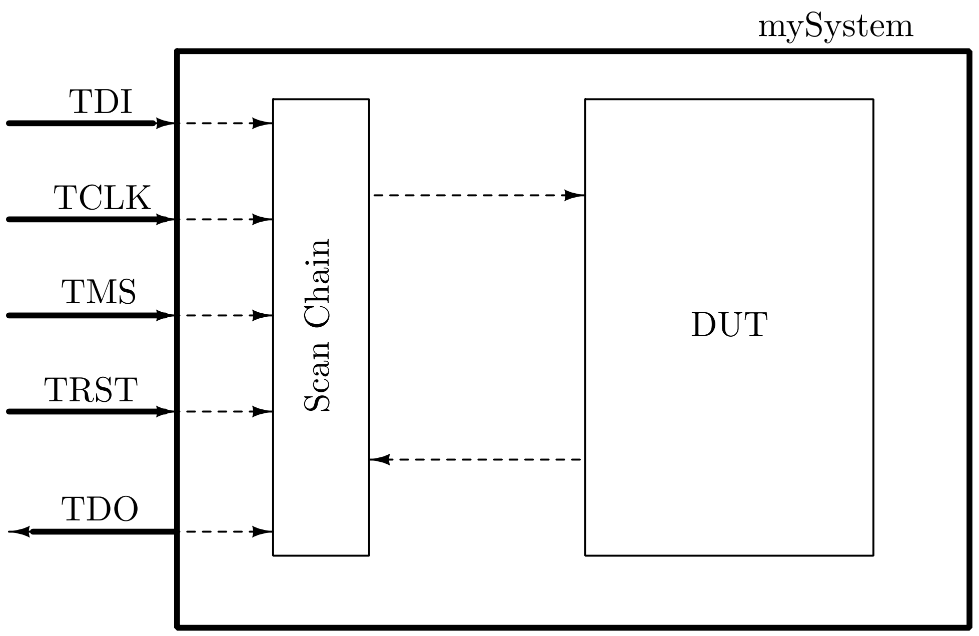

Figure 2: Scan chain added with DUT

All the interface signals of `mySystem` should be directly connected to the scan chain module as shown in fig.2.

The scan chain module have 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 (*TAP Controller*), one input scan register (*In Register*) and one output scan register (*Out Register*). The diagram showing the details of scan chain module is given in fig.3 and the entity of the scan chain module is given below.

```
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;
```

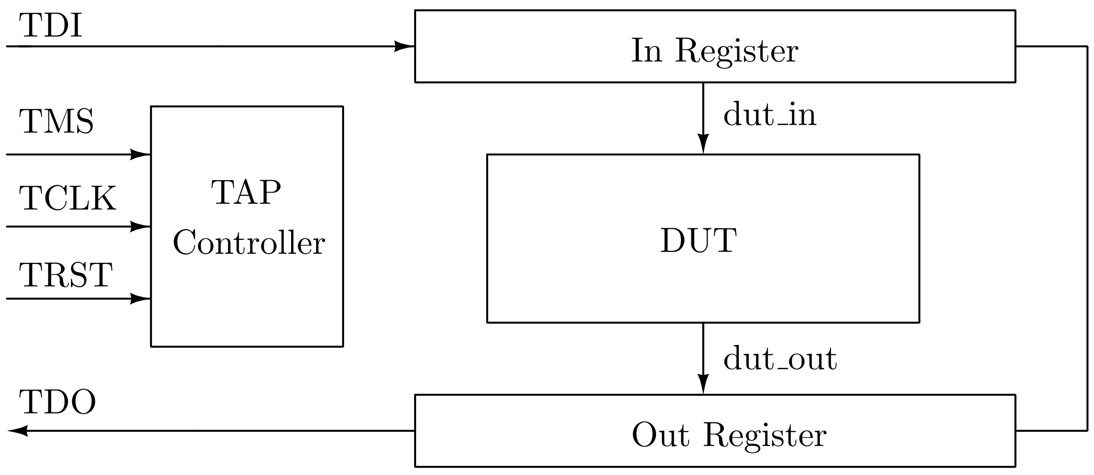

Figure 3: Scan chain module interfaced with DUT

An example VHDL file for `mySystem` is given in the Appendix I.

3 Hardware connections

Next step is to make physical connections between the host PC, microcontroller board and the user module(CPLD/FPGA). The microcontroller board is PtX-128 developed in WEL lab, IITB. The following connections between the microcontroller board and the FPGA / CPLD need to be made.

- Connect `TRST` (CPLD/FPGA) to PD2 (PORTD.2)

- Connect `TDI` (CPLD/FPGA) to PD2 (PORTD.2)
- Connect `TMS` (CPLD/FPGA) to PD2 (PORTD.2)
- Connect `TCLK` (CPLD/FPGA) to PD2 (PORTD.2)
- Connect `TDO` (CPLD/FPGA) to PD2 (PORTD.2)

Also connect the microcontroller board to the host machine using the USB cable provided.

4 Input file format

The commands are derived from the SVF format usually used in JTAG boundary scan. The inputs commands (a subset of standard SVF commands) are tabulated below. Only two commands are required for the current implementation.

- `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. Similarly, *input* & *output* contains the input combination to be applied and it's expected output combination respectively. *mask bits* are used to specify if any of the output bits are not important and could be taken as don't care (if one bit is 0 then the corresponding output bit is taken as don't care).

  Note : If the scanned output should not be compared, then all the *mask bits* should be kept as 0.

- `RUNTEST` $<$ *delay* $>$ `SEC` :

  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.

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

5 Running the Python script

Before running the python script, pyUSB library needs to be installed on the PC by the following steps.

- Download pyusb v1.0 from the site "*http://sourceforge.net/projects/pyusb/*"
- Follow the steps in the README document to install libusb and pyusb v1.0 on linux machine.

After the installation run the `scan.py` script with the following command.

```
$ sudo scan.py <input file> <output file>
```

Where the *input file* should contain all the commands to be executed, and *output file* should be an empty file (already created by the user) for storing the results back.

Appendix I

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity mySystem is
  port (
    clock_50 : in std_logic;
    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
    LED : out std_logic_vector(7 downto 0)   -- LED outputs for checking
  );
end mySystem;

architecture behave of mySystem is

component Scan_Chain is        -- Scan Chain as a component
  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 component;

component Counter is
  port (
    clock : in std_logic;   -- Clock signal
    reset : in std_logic;   -- Reset signal
    count : out std_logic_vector(7 downto 0)   -- Counter output
    );
end component;

signal dut_in : std_logic_vector(1 downto 0);
signal dut_out : std_logic_vector(7 downto 0);
signal reset, clock : std_logic;

begin  -- behave
```

```
 reset <= dut_in(1);  -- seperating the DUT input signals
 clock <= dut_in(0);

-- Connect the scan chain for testing
 SC : Scan_Chain generic map (2,8) port map ( TDI,TDO,TMS,TCLK,TRST,dut_in,dut_out );

 DUT: Counter port map (clock, reset, dut_out); -- Connect DUT

end behave;
```

Appendix II

SDR 2 TDI(2) 8 TDO(00) MASK(00)
RUNTEST 60 SEC
SDR 2 TDI(0) 8 TDO(00) MASK(00)
RUNTEST 60 SEC
SDR 2 TDI(1) 8 TDO(00) MASK(00)
RUNTEST 60 SEC
SDR 2 TDI(0) 8 TDO(00) MASK(00)
RUNTEST 60 SEC
SDR 2 TDI(1) 8 TDO(00) MASK(00)
RUNTEST 60 SEC
SDR 2 TDI(0) 8 TDO(00) MASK(00)
RUNTEST 60 SEC
SDR 2 TDI(1) 8 TDO(00) MASK(00)
RUNTEST 60 SEC
SDR 2 TDI(0) 8 TDO(00) MASK(00)
RUNTEST 60 SEC
SDR 2 TDI(1) 8 TDO(00) MASK(00)
RUNTEST 60 SEC
SDR 2 TDI(0) 8 TDO(00) MASK(FF)
RUNTEST 60 SEC