Scan chain based test setup for DE0-Nano based systems

Advisor : Prof. Madhav P. Desai

Titto Thomas
Wadhwani Electronics Laboratory
EE Dept. IIT Bombay

April 5, 2015

1 Introduction

Scan chain is a technique used for testing the hardware systems. The objective is to make testing easier by providing a simple way to set the inputs for the system and observe their 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 gives the basic idea on how the scan chain interacts with DUT.

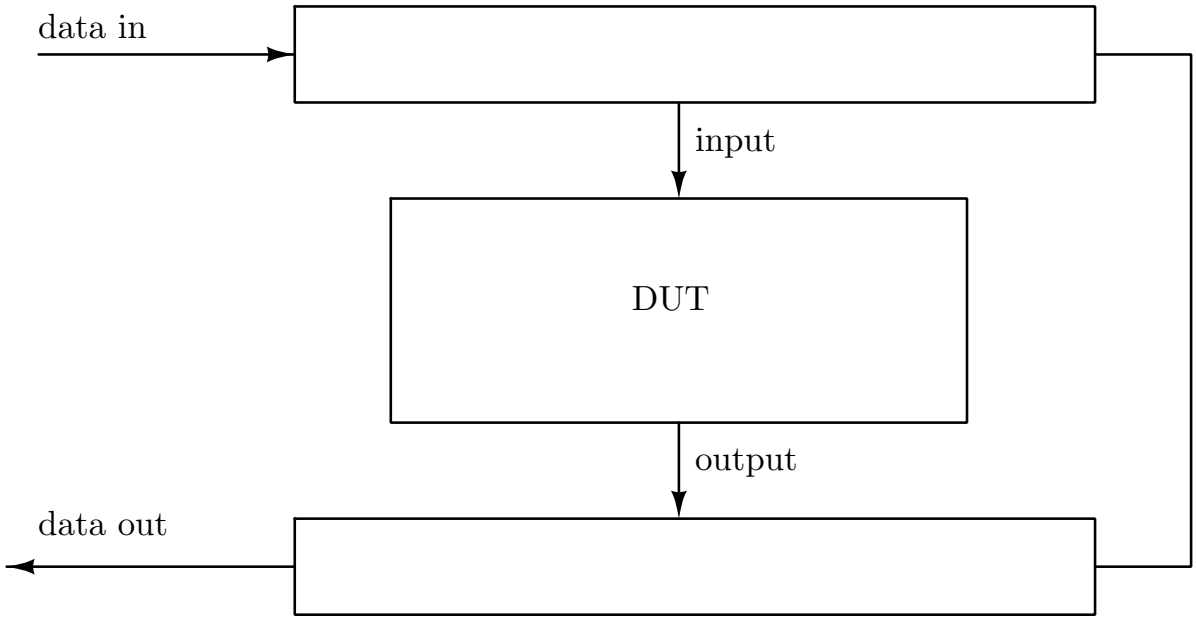

Figure 1: Basic scan chain

Joint Test Action Group (JTAG)[1] is a standard for testing hardware using scan chains. A simplified version of standard JTAG is proposed here for testing designs on the DE0-Nano board. The setup supports automated testing of the system implemented on DE0-Nano, i.e. send input combinations to the hardware and read back their outputs.

2 Test Setup

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 in a text file. Second one is a microcontroller(ATxMega128) that converts the commands (obtained from PC through USB) into a set of signals for the DE0-Nano board. The final one is the DE0-Nano board containing both the DUT and the proposed scan chain.

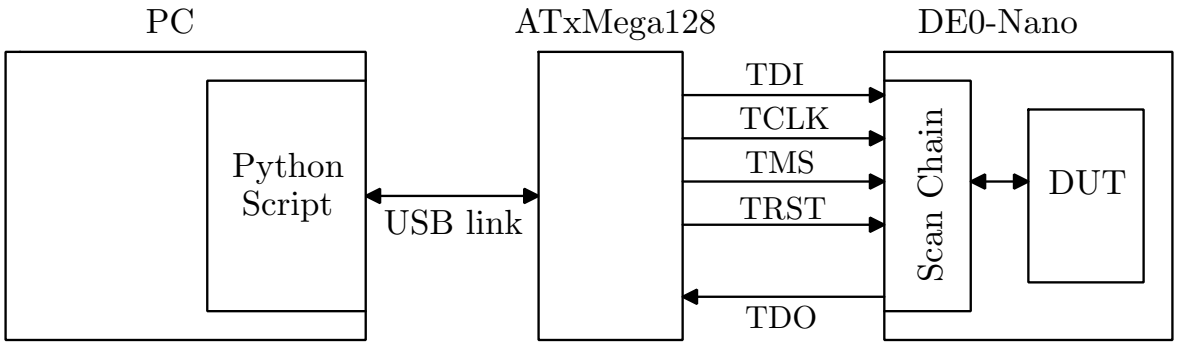

Figure 2: Block Diagram

The user can provide input combinations and their expected outputs in an input text file. The python script will parse the file, extract those commands and pass them over to ATxMega128. 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 DE0-Nano board. 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.

3 Adding scan chain to your exisitng VHDL design

The user will have a particular system (described in VHDL) that needs to be tested. This DUT should first be tested in gate level simulation and verified to be working, similar to the previous experiments.

Now for testing this DUT, the user has to write a top level entity (shown as `mySystem` in Figure 3) which contains 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` should have 5 interface signals (1 bit each) `TDI`, `TMS`, `TCLK`, `TRST` and `TDO`.

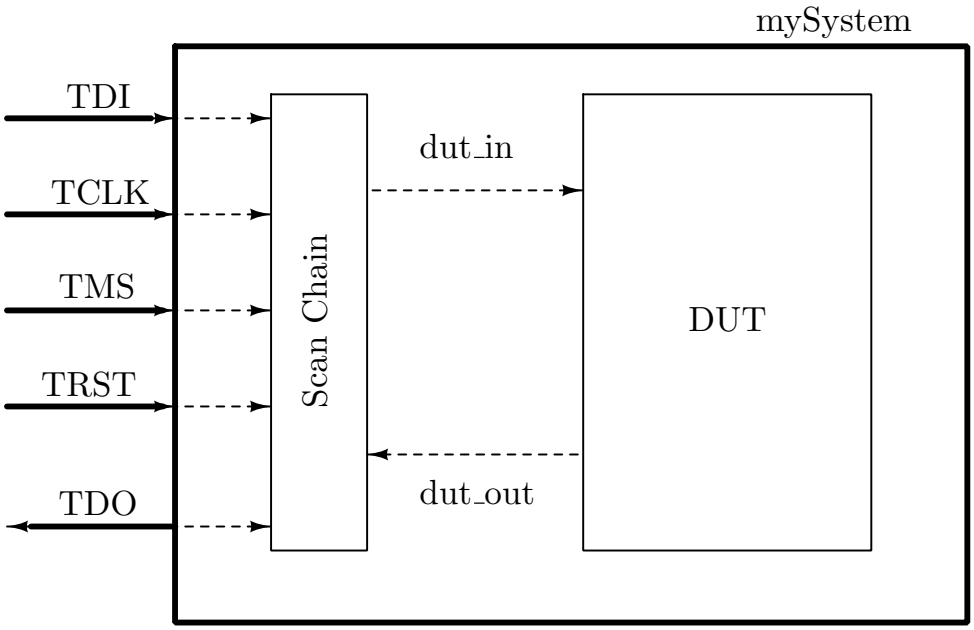

Figure 3: Scan chain added with DUT

All the interface signals of `mySystem` should be directly connected to the scan chain module as shown in Figure 3.

The scan chain module 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. The entity of the `Scan_Chain` module is given below, which could be used for instantiating it as a component in the top level entity.

```
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;
```

4 Reference Implementation

To understand more about using the scan chain module, here's an example implementation of a counter. The counter has two single bit inputs (`clock` and `reset`) and an 8 bit single output (`count`). Here, the `dut_in` will be 2 bits (one for each of the inputs) and `dut_in` will be same as `count`. The connections are clearly shown in Figure 4.

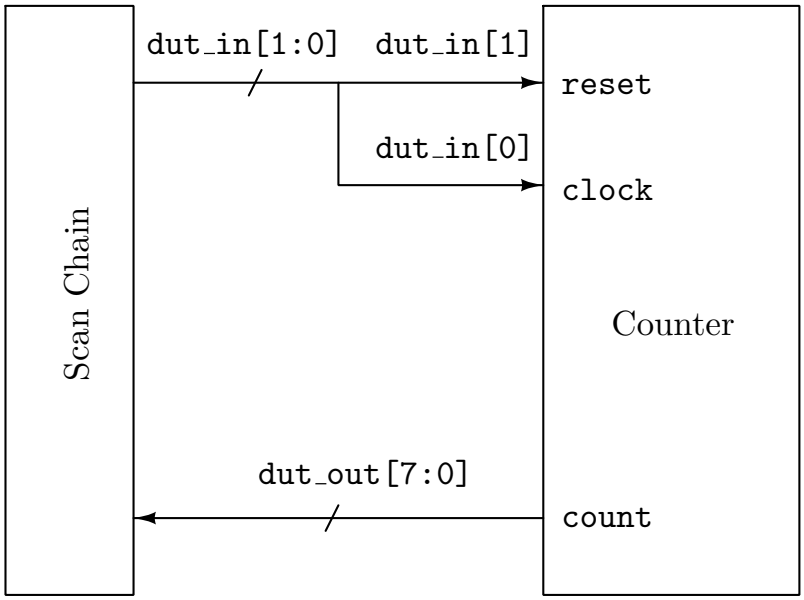

Figure 4: Connecting Counter to the scan chain

The top level VHDL description of this system is given in Appendix I.

5 Hardware connections

After loading the bundled Next step is to make physical connections between the host PC, microcontroller board and the user module(on DE0-Nano). The microcontroller board is PtX-128 (ATxMega128 based) developed in WEL lab, IITB. The following connections between the microcontroller board and the DE0-Nano need to be made.

- Connect `TRST` (DE0-Nano) to PD4 (PORTD.2)
- Connect `TDI` (DE0-Nano) to PD0 (PORTD.0)
- Connect `TMS` (DE0-Nano) to PD1 (PORTD.1)
- Connect `TCLK` (DE0-Nano) to PD5 (PORTD.3)
- Connect `TDO` (DE0-Nano) to PC0 (PORTC.0)

Also connect the PtX-128 board to the host machine using the USB cable provided.

6 Test vector 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)[3], usually used in JTAG boundary scan. Only two commands are required for the current implementation and they are given below.

- `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).

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

  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.

  Example : `RUNTEST 60 SEC`

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.

7 Running the Python script

Before running the python script, pyUSB[2] 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.

References

[1] Ieee standard for reduced-pin and enhanced-functionality test access port and boundary-scan architecture. *IEEE Std 1149.7-2009*, pages c1–985, Feb 2010.

[2] Wander Lairson Costa. Pyusb - usb access on python, 2015. [Online; accessed 5-March-2015].

[3] Neil Jacobson. *The in-system configuration handbook : a designer's guide to ISC*. Kluwer Academic Publishers, Boston, 2004.

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