Department of Electronic & Telecommunication Engineering
University of Moratuwa

B.Sc. Eng. Semester 7

EN 4603 – Digital IC Design

Laboratory Experiment 2: DFT Insertion

Objectives:

a) To understand and follow the DFT insertion design flow to insert scan-test into a design
b) To familiarize with Cadence Genus for DFT insertion

Software Required: Cadence Genus - Ver. 18.10

In this practical, you will be using Cadence Genus to perform DFT insertion into the design you synthesized in Laboratory Experiment 1. As inputs to Genus, you will provide

1. Source Verilog files
2. Technology libraries provided by the fabrication plant (here, 45 nm educational GPDK given by cadence) : .lib, .lef, .tch
3. Timing constraints

and will obtain the Scan-test compatible netlist (Verilog files), timing constrains (.sdc file), scanDEF file and a set of files to be used as inputs to Cadence Modus for automatic test pattern generation (ATPG) as output. You will then analyze, compare and comment on the area and ports of the design at various stages of the Scan test insertion design flow.

Note: Unix commands are given as **`unix:`** and Genus commands are given as **`genus:`**

Lab report:

You need to submit a small report, answering the questions in the exercise with screenshots and explanations. The format of the report is not a concern. You need to demonstrate that you understand every step of the process. Ask the junior staff for ANY clarification.

Design for Testability (DFT)

In the ever-growing semiconductor industry, to meet the high-performance expectations of the consumers, designs tend to be smaller and smaller every year. As of May 2021, IBM has introduced a new 2nm technology! In addition, the ability to mass produce devices has allowed the industry to keep pace with the growing demand of the consumers. However, smaller die sizes increase the chances of errors occurring in ICs and such errors are highly undesirable. These errors may occur even after fabrication and packaging. Therefore, it is vital to be able to test each and every chip before they are shipped to the consumers. Modern microprocessors may contain more than 1000 pins and contain billions of transistors. It is impossible to check every possible combination of inputs and outputs, and still maintain a reasonable time-to-market. This is where DFT comes into play. Simply put, DFT is the process of adding additional logic into the design in order to improve the testability of the device and improve the testing efficiency. An IC designer should keep in mind that the goal in DFT is to cover a maximum number of faults with a minimum amount of additional logic, as the insertion of additional logic directly translates to additional cost per chip.

Although there are various DFT techniques in digital IC design, in this experiment, we will be focusing on inserting the **Scan Test** into an example design.

Scan Test

Scan test is a method of detecting manufacturing faults in silicon devices. It replaces the flip-flops in a design (may or may not be all) with scannable flip-flops and links the scannable flip-flops in a chain (termed “scan chain”) in such a way that it allows the designer to monitor the secondary inputs and the secondary outputs of the design.

As shown in the schematic in figure 1, a scannable flip-flop has two additional pins and a mux compared to a normal flip-flop. By setting the shift enable (a.k.a scan enable) pin to high or low, we can specify whether D or SD (scan input) passes to the internal flip-flop as the input. When scan input is connected to the input of the internal flip-flop, we call that the scannable flip-flop is in “scan-shift mode”. Otherwise, it is in the “functional mode”, which is the mode that the device functions under normal conditions.

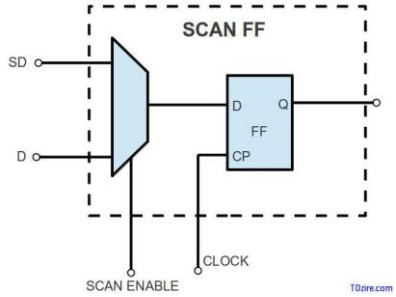

Figure 1: Schematic of a Scannable Flip-Flop

By connecting the scannable flip-flops in a chain as shown in figure 2 (below), in the scan-shift mode, an input test vector can be fed into the design from the starting flop and can be shifted through the scan-chain to set the inputs to the combinational part. Then, the flops are set back to the functional mode and given a clock pulse to capture the outputs from the combinational block for the given inputs. Afterwards, the flops are set back to the scan-shift mode to shift the output vector out through the scan output port. Given the design, there are techniques that we can determine these input test vectors and expected output vectors such that we can detect more than 99% of the faults in the design! (tools used to generate such test vectors are called Automatic Test Pattern Generator (ATPG) tools).

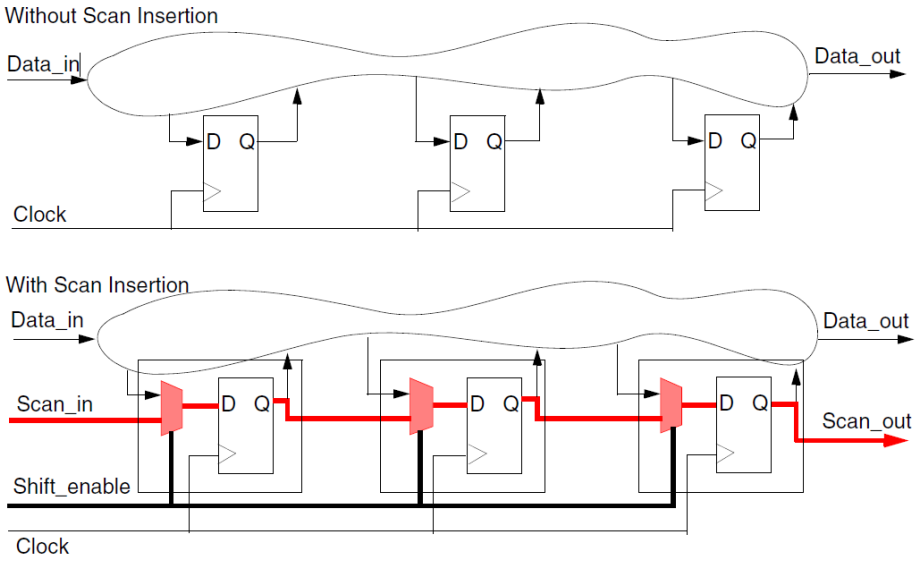

Figure 2: Before (above) and after (below) Scan Test Insertion

The general design flow for Scan Test insertion, which we will be following in this laboratory experiment, is shown in figure 3. The same design flow can be followed with minor modifications for inserting most other DFT techniques.

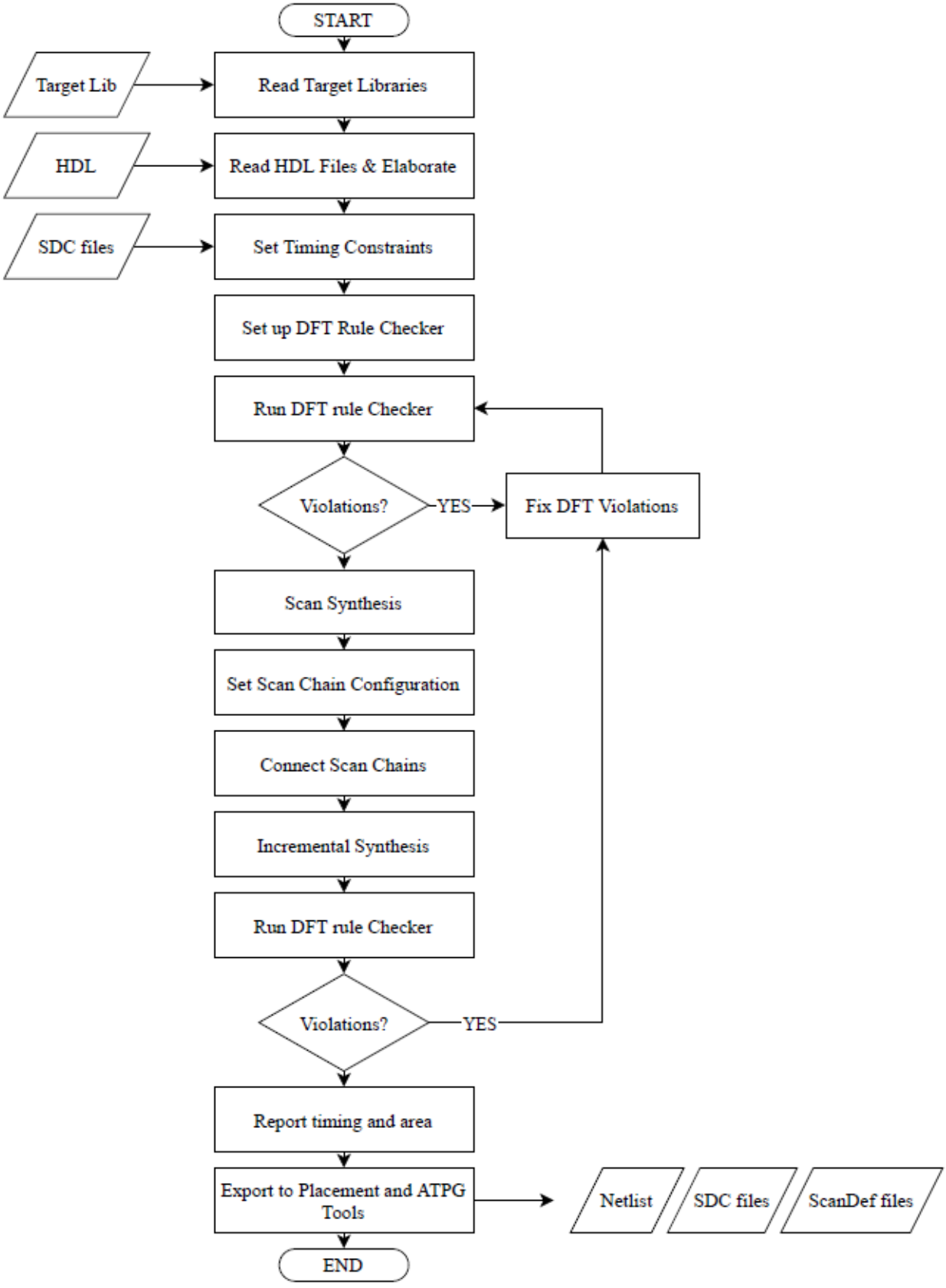

Figure 3: DFT insertion design flow – Scan Test Insertion

Step 1: Understand the UART Transceivers Reference Design

Figure 4 shows the example design that is being used in this Lab. It contains two UART Transceivers connected together as shown. All the codes are written in Verilog HDL.

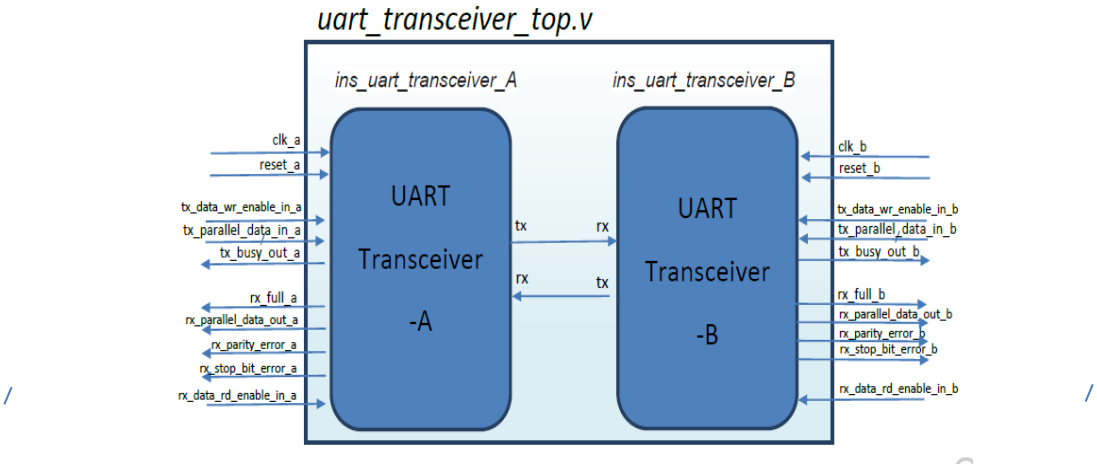

Figure 4: Top Module Schematic

The design has two asynchronous clock signals and two reset signals. The clock signal of **Transceiver A** is ***clk_a*** and Transceiver B is ***clk_b***. The Transceivers A and B are having two reset signals, ***reset_a*** and ***reset_b*** respectively. Each Transceiver module sends and receives data according to the RS232 communication standard. User interface for reception and transmission is parallel.

The transmitter parallel user interface having three ports named:

1. *tx_data_wr_enable_in_x,*
2. *tx_parallel_data_in_x,* and
3. *tx_busy_out_x.*

The receiver interface ports are:

1. *rx_full_x,*
2. *rx_parallel_data_out_x,*
3. *rx_parity_error_x,*
4. *rx_stop_bit_error_x. and*
5. *rx_data_rd_enable_in_x.*

For more information about the design, go through its Verilog HDL files.

Step 2: Setup the Project Directories

1. Launch Terminal: In the start menu (application launcher) search and open: *Konsole*

2. Check if you are in the right directory:

   `unix: pwd`

   You should get **`/home/student/`**

3. Navigate to your directory:
   ```
   unix: cd entc17_dicd/<index_no>
   ```

4. Copy the lab files & examine the folder structure
   ```
   unix: cp -r ../../dicd_labs/lab2_dft_insertion ./
   unix: cd lab2_dft_insertion
   unix: ls
   ```

   lab2_dft_insertion:

   |**--input/**
   | |**--rtl/** # Contains HDL source
   | |**--libs/** # Contains technology libraries (45nm GPDK)
   | |**--constraints.tcl** # Commands to set timing constraints
   |**--log/** # Stores generated log files
   |**--output/** # Output Netlist & Constraints (SDC)
   |**--report/** # Generated reports
   | |**--after_scan_synthesis/** # Generated reports after scan synthesis step
   | |**--after_scan_connect/** # Generated reports after scan connect step
   |**-- scripts/** # Scripts to be sourced directly
   |**--work/** # This is the working directory of the project

5. Change directory to **`work`**

Step 3: Using Genus for Scan Test Insertion

1. Start Genus (command line tool). Your terminal will change to the Genus command line.
   ```
   unix: genus
   ```

2. Since the purpose of setting the search path & including the technology libraries was discussed in the previous laboratory experiment, we shall execute these commands at once by sourcing the given **`../scripts/setup.tcl`** file.
   ```
   genus: source ../scripts/setup.tcl
   ```

   You may observe the contents of **`setup.tcl`** using vim. It contains the following commands:
   ```
   genus: set_db init_lib_search_path [list ../input/libs/gsclib045/lef
   ../input/libs/gsclib045/timing ../input/libs/gsclib045/qrc/qx]
   genus: set_db library {slow_vdd1v0_basicCells.lib
   fast_vdd1v0_basicCells.lib}
   genus: set_db lef_library {gsclib045_tech.lef gsclib045_macro.lef
   gsclib045_multibitsDFF.lef}
   genus: set_db qrc_tech_file gpdk045.tch
   ```

3. Read the RTL design files into Genus:
   ```
   genus: read_hdl [glob ../input/rtl/*.v]
   ```

4. Elaborate the design.
   ```
   genus: elaborate uart_top
   ```

5. Uniquify the top module.
   ```
   genus: uniquify uart_top
   ```

6. Set timing constraints in Genus.
   ```
   genus: source ../input/constraints.tcl
   ```

Now that we have set the target libraries, read the design files and set the design constrains, we may now proceed to the DFT insertion steps.

7. Set the DFT scan style in order to configure the DFT rule checker. This is due to the fact that DFT rules vary with different scan styles. Genus supports two scan styles – Muxed Scan Style and Clocked Level-Sensitive-Scan (LSS) Style. We will be using the Muxed Scan Style in this experiment.
   ```
   genus: set_db dft_scan_style muxed_scan
   ```

   Note: we use **`set_db`** command to set attributes in the common UI mode in Genus.

8. Set the prefix for names of additional modules/ports which will be automatically created during the DFT insertion process to accommodate test logic. We will set this prefix name as "**`dft_`**".
   ```
   genus: set_db dft_prefix dft_
   ```

9. Define shift_enable signal. Since the muxed-scan style uses shift_enable signals to switch scan flip-flops from functional mode to scan-shift mode, it is recommended that we define these signals before running DFT rule checker. The following command defines an active-high shift_enable pin with the name "**`SE`**" and creates a port for it.
   ```
   genus: define_shift_enable -name SE -active high -create_port SE
   ```

10. Run DFT rule checker.
    ```
    genus: check_dft_rules
    ```

    The DFT rule checker checks all flip-flops to determine if clock pins to the flip-flops can be controlled and if asynchronous set/reset pins (if available) to the flip-flops can be held to their non-controlling value during scan-shift mode. It is essential that there are no DFT violations at this step as any flip-flops with DFT violations will not be affected by DFT insertion commands which will be executed in the subsequent steps.

    If all steps have been followed correctly up until this point, we should not see any DFT violations.

11. Synthesize the design into the generic logic netlist and then map to the technology library. In this step, all the non-scannable flip-flops will be replaced by scannable flip-flops.

    Synthesize to generic logic with medium effort
    ```
    genus: set_db syn_generic_effort medium
    genus: syn_generic
    ```

Map to technology library and re-synthesize with medium effort

```
genus: set_db syn_map_effort medium
genus: syn_map
```

12. Write the scan synthesized netlist as *uart_top_1.v*

```
genus: write_hdl > ../output/uart_top_1.v
```

13. Generate the reports after scan synthesis. These reports will be necessary for comparisons later.

```
genus: report_area > ../report/after_scan_synthesis/area.log
genus: report_timing  -nworst 10 >
../report/after_scan_synthesis/timing.log
genus: report_port * > ../report/after_scan_synthesis/ports.log
genus: report_power > ../report/after_scan_synthesis/power.log
```

14. Set scan configuration. In this step, we will define the scan chains according to our requirements. Note that the design contains two transceivers which operate on two clock domains. Therefore, we need at least one scan chain per clock domain to cover the entire design. Execute the following commands to define two scan chains – one per clock domain.

```
genus: define_scan_chain -name top_chain_a -sdi scan_in_a -sdo
scan_out_a -non_shared_output -create_ports -domain clk_a

genus: define_scan_chain -name top_chain_b -sdi scan_in_b -sdo
scan_out_b -non_shared_output -create_ports -domain clk_b
```

15. Preview the scan chains. Here, we will be able to see a summary of the scan chains we defined in the previous step. It will show the name, scan input port, scan output port, clock domain, sensitive edge and the number of scannable flip-flops in each chain.

```
genus: connect_scan_chains -preview -auto_create_chains
```

16. Connect scan chains. This command will connect the scan inputs (SI) of the scannable flip-flops to respective output ports of other scannable flip-flops or the scan input port (if it is the first scannable flip-flop in the chain).

```
genus: connect_scan_chains -auto_create_chains
```

Note: This process is also referred to as “Scan Stitching”.

17. Perform incremental synthesis to generate the netlist of the scan connected design

```
genus: syn_opt -incr
```

18. Perform DFT rule check after scan connecting.

```
genus: check_dft_rules
```

19. Report scan setup and scan chain information. You may open and read their contents using vim.

```
genus: report_scan_setup > ../report/scan_setup.log
genus: report_scan_chains > ../report/scan_chains.log
```

20. Write the DFT (scan test) inserted netlist and constrains.

```
genus: write_hdl > ../output/uart_top_2.v
genus: write_sdc > ../output/uart_top_2.sdc
```

21. Write the scanDEF file. The scanDEF file describes the scan chain configuration (architecture) as the set of re-orderable scan chains present in the scan-mapped netlist. This file is necessary for the Place&Route tool to re-order scan chains.

```
genus: write_scandef > ../output/uart_top_2_scanDEF.scandef
```

22. Generate the reports after scan connect.

```
genus: report_area > ../report/ after_scan_connect/area.log
genus: report_timing  -nworst 10 > ../report/
after_scan_connect/timing.log
genus: report_port * > ../report/ after_scan_connect/ports.log
genus: report_power > ../report/ after_scan_connect/power.log
```

23. Write the scripts required for the ATPG tool. This command generates a script and the data files required to run an ATPG analysis using the Cadence Modus Test software (not covered in this experiment).

```
genus: write_dft_atpg -library
../input/libs/gsclib045/timing/slow_vdd1v0_basicCells.lib
```

24. You can observe the DFT inserted design in GUI too. Try to select the designs in hierarchy (left), right click, schematic view, in New. You can descend down the hierarchical blocks and observe how the scan chains are connected, how the I/O ports are muxed to switch between test mode and functional mode etc.

```
genus: gui_show
```

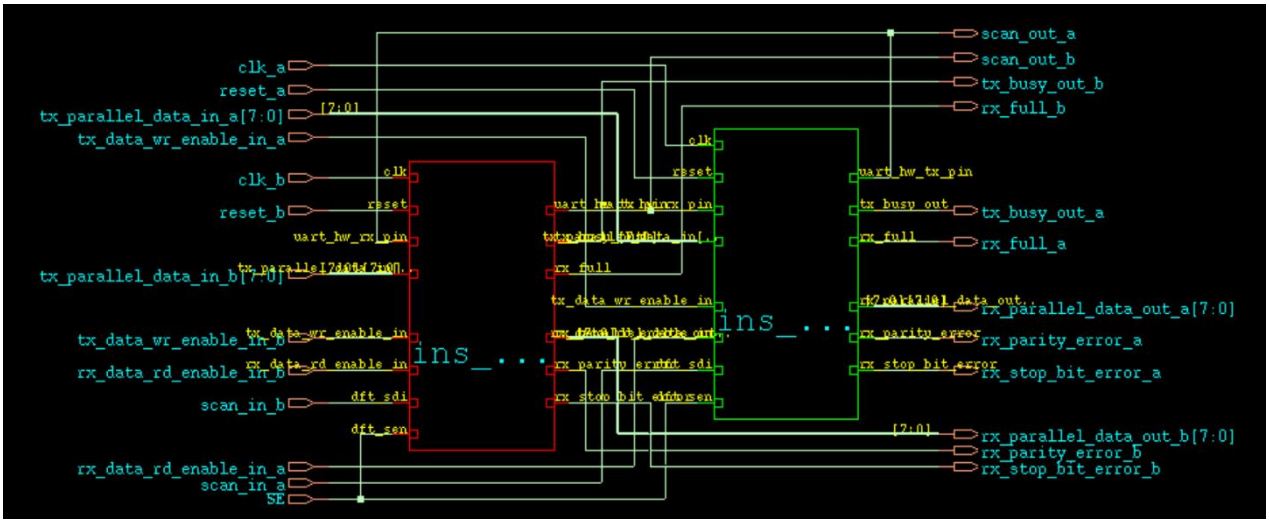

Figure 5: Schematic of the Top Module after Scan test insertion

Exercise

1. Comment on the area before (output of the laboratory experiment 1) and after (*uart_top_1.v*) scan synthesis.

2. Comment on the number of ports before (output of the laboratory experiment 1) and after (*uart_top_1.v*) scan synthesis.

3. Compare how the scan input (**`SI`**) of scan cells are connected before (*uart_top_1.v*) and after (*uart_top_2.v*) scan stitching. Show comparable code snippets to justify your answer.

4. Comment on the area before (*uart_top_1.v*) and after (*uart_top_2.v*) scan stitching.

5. Comment on the number of ports before (*uart_top_1.v*) and after (*uart_top_2.v*) scan stitching.

6. Reiterate through the steps, but do the necessary changes in the design flow to create two scan chains per transceiver along with dedicated scan I/O ports. A single port may be used for scan enable.

   Hint: find out more about **`define_scan_chain`** and **`connect_scan_chains`** commands

7. It can be seen that, if we choose to have dedicated scan I/O ports for each additional scan chain, then the number of ports in the design will increase with the number of scan chains. In a pad limited design, it is crucial that the IC designer uses an absolute minimum number of ports in the final design. Assume that your design is pad-limited and that you have 32 scan chains in the design – which will result in 32 **`SDI`** ports + 32 **`SDO`** ports + 1 **`scan_en`** port = 65 scan I/O ports. Suggest a method how we can reduce the number of total scan I/O ports in the design to less than 10. You are allowed to add additional logic/registers into the design.

   Explanation of the method is sufficient (no need to implement).