

Digital Design 2  
Dr. Mohammed Shalan  
Project 2 Report: Static Timing Analysis Tool  
Group: Ahmed A. Agiza (900121143)  
Mohammed R. Anany (900120267)

### **Outline:**

The aim of the project is to design a static timing analysis tool that receives a Verilog netlist, standard cell library, timing constraints, parse these inputs, perform static timing analysis on the circuit, generate timing report with highlighting of violation and perform basic fixes and optimization.

### **Technology Used:**

- Node.js framework for the back-end.
- Express.js application framework.

### **Design and Implementation:**

#### ***Models:***

The implementation of the system depended on the existence of three main models:

- Liberty Parser
- Netlist Parser
- Static Timing Analyzer

#### ***Liberty Parser:***

The liberty parser model parses standard cell liberty file, extracts all the cells and templates from the file while providing interpolation and extrapolation methods for the tables presented in the file.

#### ***Netlist Parser:***

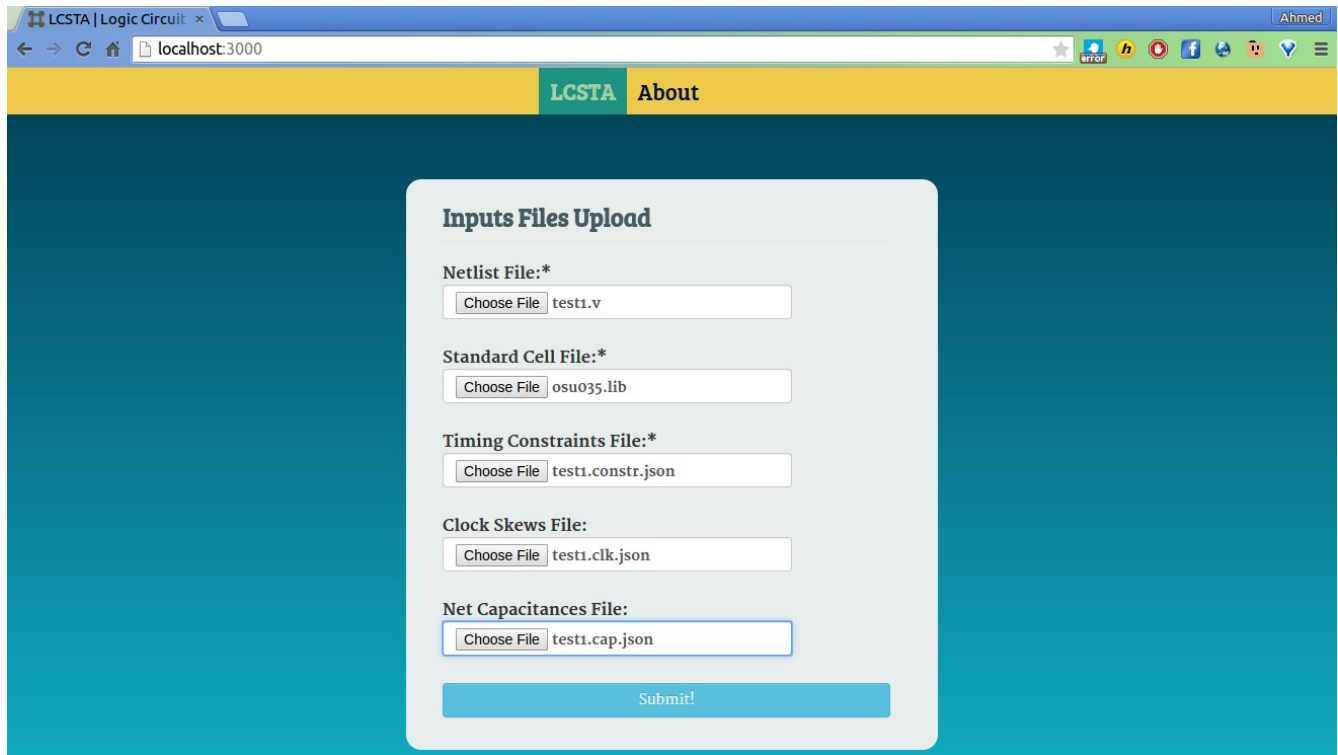
The netlist parser receives a Verilog netlist file content, parsed standard cell library, and constraints. It then parses the Verilog files and constructs Cell models for each gate with the required connections and helper methods, and adds all the available attributes from the standard cell library and the constraints.

#### ***Static Timing Analyzer:***

The static timing analyzer receives the parsed cells along with the remaining constraints. It then constructs the DAG, perform static timing analysis based on the data present in the cell, analyze critical path, calculate arrival time, required time, and slack, and finally report back the analyzed data while highlighting setup and hold violation. After the analysis the model should do basic optimization and violation fixes.

### Application Interface:

Upon opening the application the user is prompted to upload the desired netlist file, standard cell file, timing constraints file, clock skews file, and net capacitances file.



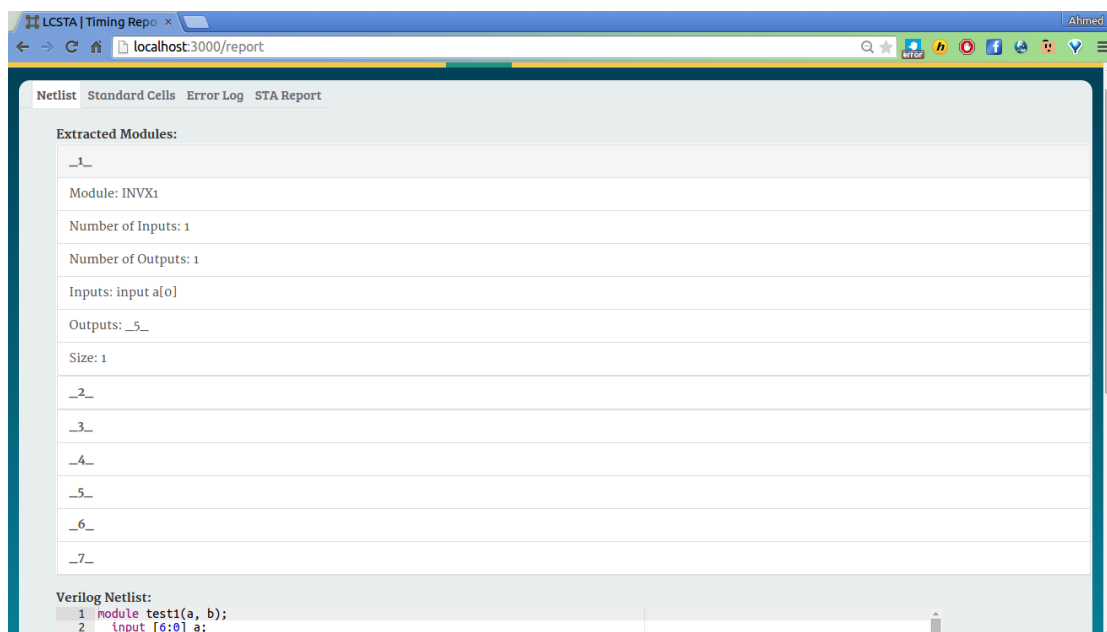
The screenshot shows a web browser window with the address bar at 'localhost:3000'. The page has a yellow header with 'LCSTA' and 'About' links. The main content area is a light blue box titled 'Inputs Files Upload'. It contains five file upload sections, each with a 'Choose File' button and a text input field:

- Netlist File:\* test1.v
- Standard Cell File:\* osu035.lib
- Timing Constraints File:\* test1.constr.json
- Clock Skews File: test1.clk.json
- Net Capacitances File: test1.cap.json

A blue 'Submit!' button is at the bottom of the form.

After uploading and processing the file(s), the user is then directed to another page where the interface is divided into 4 tabs: The Verilog modules explorer, standard cell library explorer, error log and timing report.

*The Verilog explorer* shows the modules, their sizes, their connection and the raw Verilog code.



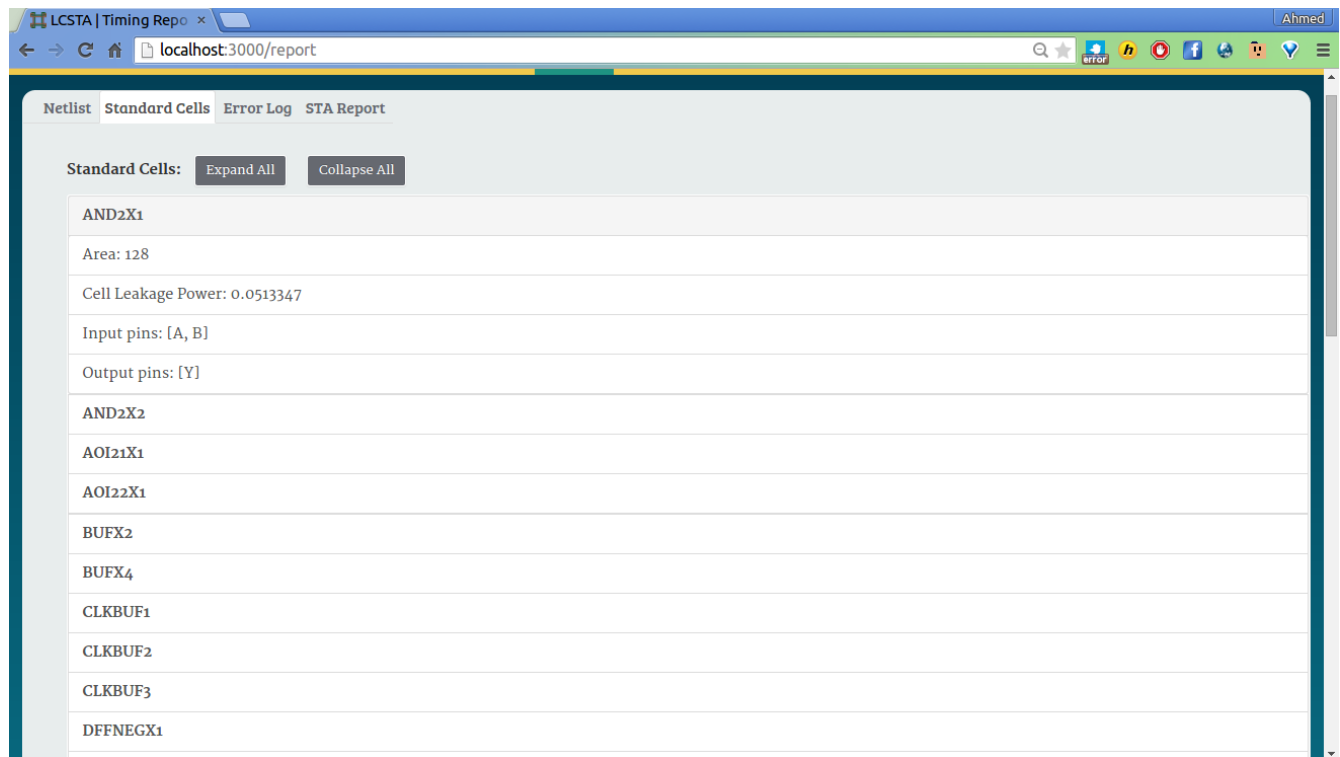
The screenshot shows the 'Timing Report' tab selected in the application. The interface has a dark blue header with 'LCSTA | Timing Report' and a user profile 'Ahmed'. Below the header is a tab bar with 'Netlist', 'Standard Cells', 'Error Log', and 'STA Report'. The main content area is divided into two sections:

- Extracted Modules:** A table listing modules and their properties.
- Verilog Netlist:** A code editor showing the raw Verilog code.

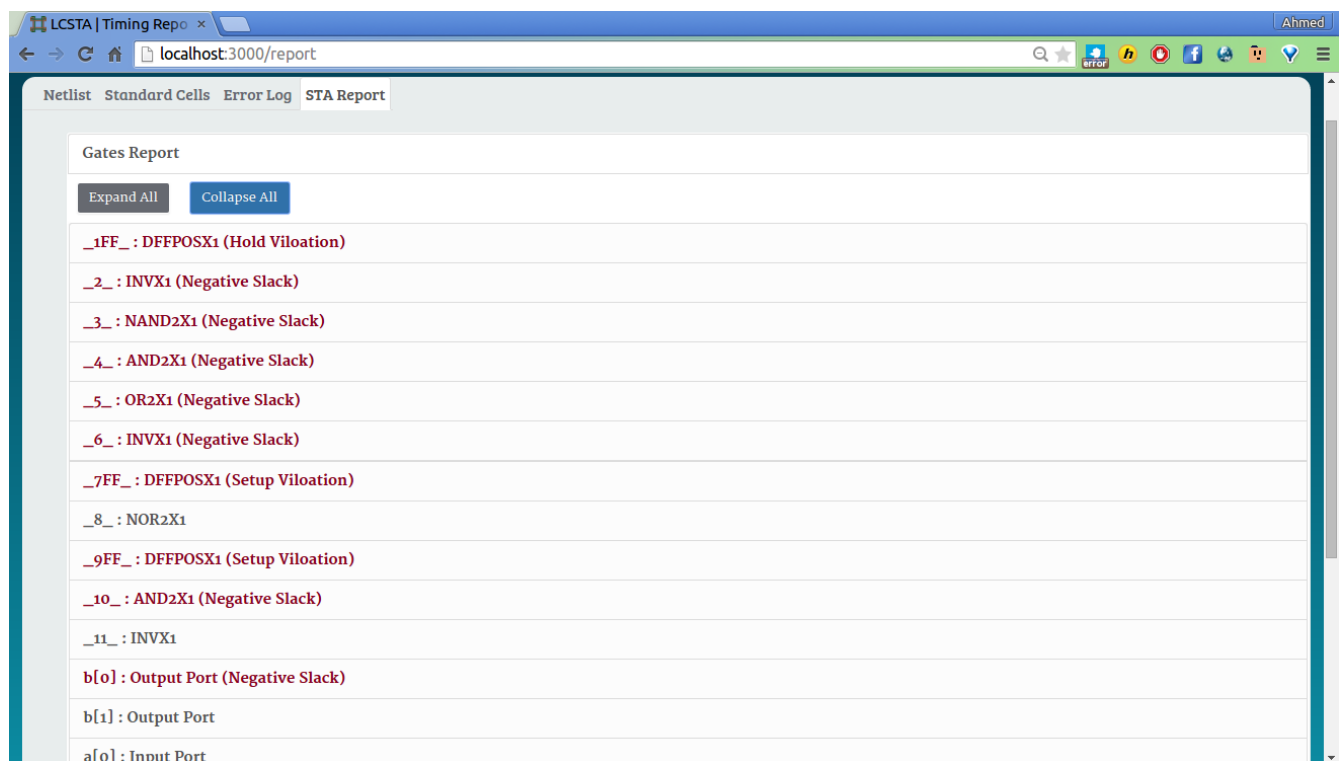
Module Name	Number of Inputs	Number of Outputs	Inputs	Outputs	Size
INVX1	1	1	input a[0]	output _5_	1
_1_					
_2_					
_3_					
_4_					
_5_					
_6_					
_7_					

```
1 module test1(a, b);
2   input [6:0] a;
```

The standard cell explorer shows the parsed standard cell library with some cell attributes.



Finally, the timing report shows collapsible section for each gate with the calculated data for the cell while highlight the cells that have timing violation.



<b>_9FF_ : DFFPOSX1 (Setup Vilation)</b>	
Minimum Input Slew Rate:	0.09
Maximum Input Slew Rate:	0.1
Minimum Output Slew Rate:	0.16647686063849773
Maximum Output Slew Rate:	0.1905648419287409
Minimum Capacitance Load:	0.1611078
Maximum Capacitance Load:	0.1611936
Minimum Cell Delay:	0.33619309199578823
Maximum Cell Delay:	0.34704575742724875
Minimum Arrival Time:	0.7101138398274185
Maximum Arrival Time:	2.2356859375816267
Required Arrival Time:	2.04
Setup Slack:	-0.19568593758162667
Hold Slack:	0.5923524695931451
Arrival Time Start:	0.04
Required Time Start:	-0.38421702501908284
Flip-flop Slack as start node:	-0.4242170250190828
Minimum Setun Time:	0.27450524280666154

### Constraints Files Format:

For constraints files, JSON formats were used for portability and flexibility, where for the net capacitances each wire name was mapped to its capacitance, while in the skews file each D Flip-flop was mapped to its clock skew. As for the constraints file, it contains 5 objects, an essential clock object, input slew object, input delay object, output capacitance object, and output delay.

### Testing Strategy:

For testing, we used an incremental approach. Each part was first tested alone. The parser was run separately from the rest of the application using netlist files that we wrote and the standard cell library. The static timing analyzed was tested with hard-coded data that were tested incrementally with an attempt to introduce edge cases periodically to detect potential bugs. Eventually, all components were integrated, tested together, fine-tuned, and reported through the interface.

### Test Cases:

6 test cases were developed with the aim of testing various pars of the system with following attributes:

- *Test case 1:* purely combinational (no flip-flops) without timing violations to test the parser and the static timing analyzer as an initial test.
- *Test case 2:* a larger test case that incorporates flip-flops and contains timing violations.
- *Test case 3:* a small test to test the edge case of a flip-flop feeding back to a previous flip-flops.
- *Test case 4:* another edge case for a flip-flop feeding back to itself.
- *Test case 5:* similar to test case 2 for extra verification.
- *Test case 6:* a comprehensive test cases with many edge cases, such as self feedback, a net feeding into multiple gates, and net feeding multiple times to the same gate.