THE AMERICAN UNIVERSITY IN CAIRO
الجامعة الأمريكية بالقاهرة

DIGITAL PROJECT II:
STATIC TIMING ANALYSIS

DELIVERED TO:
DR. SHALAN

MEMBERS:
AHMED WAEL
AYA HARB
MAHMOUD EL-KODARY

Project Overview:

This project is intended to analyze a circuit to determine if such circuit with certain constraints and properties is subjected to timing violations. This is done through multiple phases:

1. Parsing required files for analysis:
    - Verilog netlist.
    - Liberty file.
    - Constraints file (ClockPeriod, delays of inputs and outputs).
    - Skew File for the FlipFlops (clock skew and slew rate of each FlipFlop clock).
    - Capacitance File (wire capacitance).
2. Constructing the DAG.
3. Traversing graph using different algorithms to calculate Arrival time and required time.

Process & Assumptions:

Parsing Netlist:

First the verilog netlist is parsed to produce maps for gates and wires. Wires have one source but multiple destinations and while parsing the wires are connected to it's relative gate to create a graph. The methodology followed here is that each gate is considered a node and every destination connecting a node is an edge. Flying wires (wires that do not have a source or a destination) can cause the program not to function properly. Before Parsing the netlist a gate called "root" is created and added to the gate map to signify the starting point of the graph. During the process of parsing the gate level netlist, once a flipflop is encountered it is translated into two flipflops, one for capturing

and the other is for releasing; this is done to avoid cycles in a graph. Also inputs, outputs, and wires are considered both a gate and an edge in the graph.

Parsing other Required Files:

Capacitance file: This contains the list of wires corresponding and their capacitances. To avoid extra complexities edges of the same wire (same source but different destination) have same capacitance.

File format : wireName #float

Skew File: Has the clock skew and slew entering each flipflop. FlipFlop parameters are edited during parsing.

File format : flipFlopName #skew_float #slew_float

Constraint File: Contains the ClockPeriod used and input slew rate entering each input gate. Relevant gates are edited during this process.

File format : ClockPeriod #float

inputName #inslew_float

Liberty file: The properties of gates are extracted from this file.

Construction/Traversing of DAG:

During parsing a DAG is constructed but not sorted yet. Topological sort algorithm is implemented to be able calculate output slew rates of the gates level by level. Since the output slew rate of one gate is the input slew rate of another gate, therefore to ensure that each gate has an input slew rate, gates must be traversed level by level.

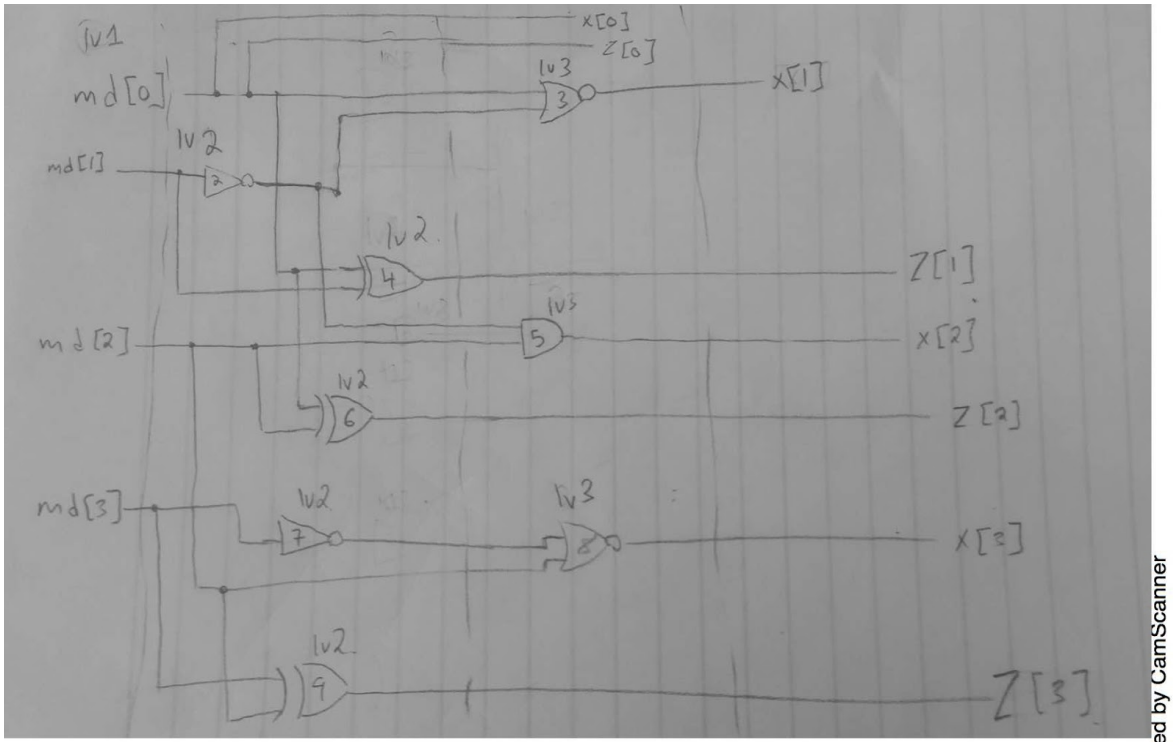
Output slew rate is added upon the edge connected to gate, while keeping in mind that an edge is different from wire, as an edge has only one destination.

Depth first algorithm is implemented to traverse through a path and get the path delay (gates delays are calculated using lookup tables extracted from the liberty file) and report it back to consol. This DFS algorithm will report back all the timing paths in the circuit in all their types (Input-Reg, Reg-Reg, Input-Output, Reg-Output). For each timing path, the arrival and required times of the circuit are calculated to show if there is a setup violation (a negative difference between required and arrival means a setup violation).

Test Cases:

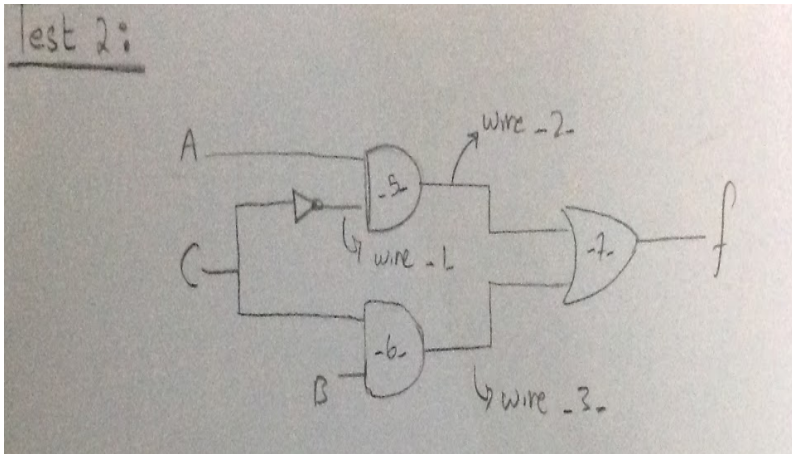
Six test cases were designed (Assign Statements are Not supported):

1. Booth file: A very simple test case with few gates and wires. No FlipFlops

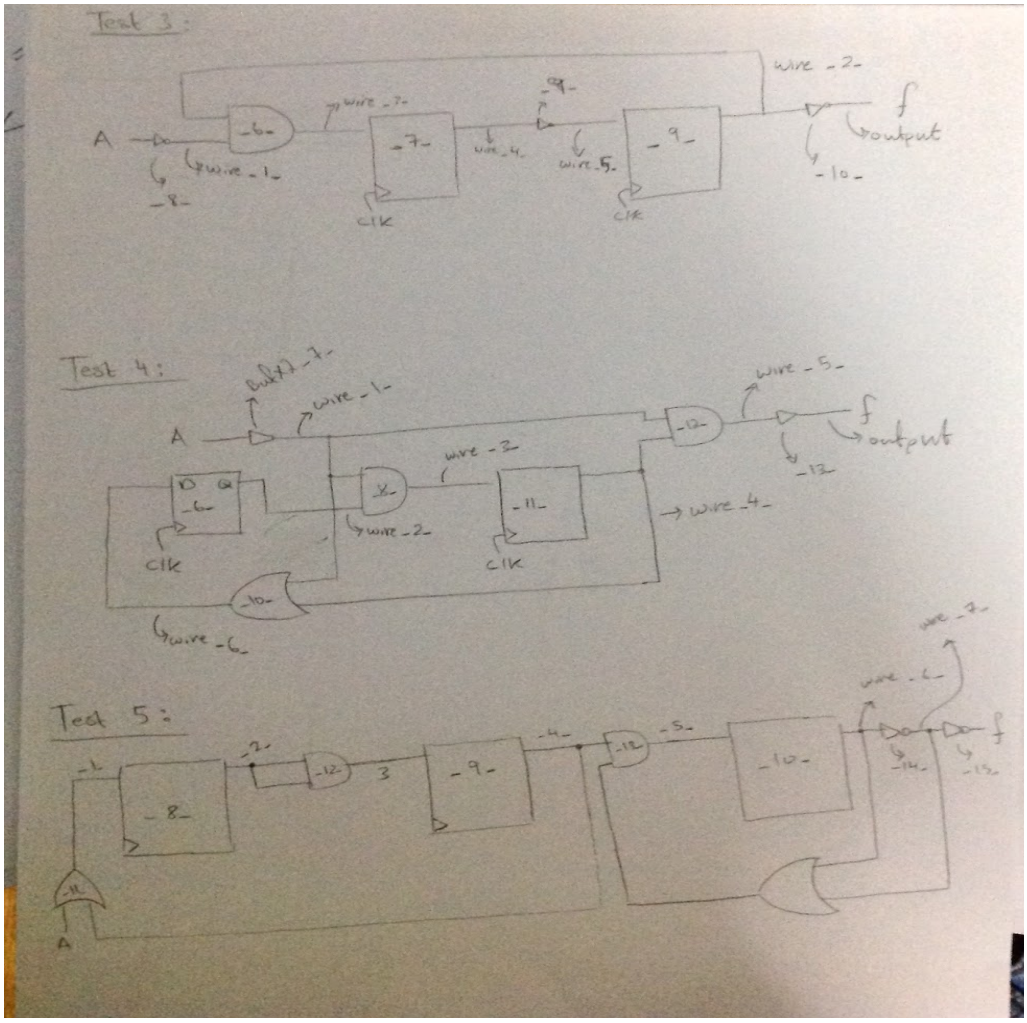


ed by CamScanner

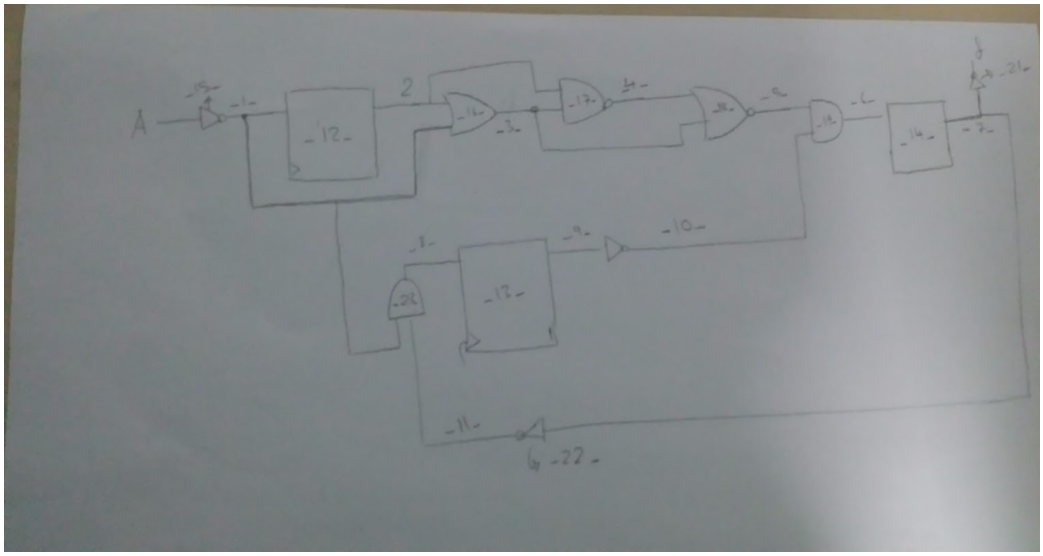
2. Test 2: Simple test case no flipflop



3. Test 3-6: FlipFlops with combinational



4. Test Case 6:



5. Special Test: 6502_alu.g.v The function for printing the timing paths, stripped off the abilities of STA (due to inability to write corresponding Constraints, Skew, and Capacitance files), was able to print all the timing paths for this enormous netlist file.

Results:

1. Paths are printed to console correctly on all test cases

Difficulties Faced:

Python was new to us to and project one was buggy due to such reason. And since project two depends on project one, the team has decided to re-code everything from step **zero** in C++. Team members used different libraries which was not available in some compilers, such as QRegEXP built in Qt. So code had to be moved and new inconsistencies developed which caused a bit of a headache, but still not a major problem.