To Calculate the Maximum Possible Frequency for the Given Circuit Using Static Timing Analysis

Siddharth Dalia and Aayush Desai
Submitted to: Prof. Jyoti Maheshwari
Institute of Technology, Nirma University

***Abstract*—This paper presents an in-depth investigation of static timing analysis (STA) applied to digital circuits for determining the maximum operating frequency. We explore key timing parameters such as propagation delay, setup and hold times, and critical path determination. A Python-based approach is used to automate the STA process by analyzing digital netlists. The methodology includes recursive path tracing and delay computation to identify bottlenecks. Simulation results confirm the theoretical findings, with our test circuit achieving a maximum frequency of 0.0435 Hz.**

I. Introduction

In modern digital circuit design, timing verification is critical to ensuring both functionality and performance. Static Timing Analysis (STA) is an essential tool that evaluates the timing characteristics of a circuit without requiring dynamic simulation. By analyzing signal propagation through various paths, designers can determine whether the circuit meets timing constraints and identify the maximum clock frequency at which it can operate reliably.

II. Theoretical Background

A. Timing Parameters in Digital Circuits

The performance of a digital circuit is constrained by several key parameters:

- **Propagation Delay (t_{pd}):** The time taken for a signal to propagate from the input to the output of a logic gate.
- **Setup Time (t_{setup}):** The minimum time before the clock edge that data must be stable at an input.
- **Hold Time (t_{hold}):** The minimum time after the clock edge that data must remain stable.
- **Critical Path:** The longest combinational delay path from input to output.

B. Static Timing Analysis (STA)

STA evaluates circuit timing using mathematical analysis rather than simulation. It involves:

1) **Netlist Parsing:** Extracting circuit components and their connectivity.
2) **Recursive Path Analysis:** Tracing paths from primary inputs to outputs.
3) **Delay Accumulation:** Summing individual delays along each path.
4) **Critical Path Determination:** Identifying the path with the longest accumulated delay.

The maximum clock frequency is computed as:

$$f_{\text{max}} = \frac{1}{t_{\text{critical}}} \tag{1}$$

where t_{critical} is the total delay of the critical path.

III. Methodology

A. Flowchart of the STA Process

Below is a flowchart summarizing the steps in our static timing analysis procedure. It outlines how the netlist is parsed, paths are traced, delays are accumulated, and final reports are generated.

B. Algorithm Outline

The following pseudocode highlights our approach:

Read netlist CSV file
Initialize delay dictionary and path list
Identify primary outputs from netlist
foreach *primary output* **do**
  Trace fanin connections recursively
  Accumulate path delays
  Store valid paths
end
Calculate frequencies from path delays using $f_{max} =$ frac1t$_{delay}$
Identify the critical path with the lowest frequency
Generate output report

Algorithm 1: Critical Path Analysis Algorithm

IV. Implementation and Results

A. Python Code

Below is the complete Python code for our static timing analysis. It reads a netlist from a CSV file, recursively traces circuit paths, accumulates delays, and computes the maximum frequency.

```
import pandas as pd
import numpy as np

input_csv = r"C:\Users\aayus\Downloads\netlist (1).
    csv"
output_csv = r"C:\Users\aayus\Downloads\
    traced_paths_output.csv"

df = pd.read_csv(input_csv)

net_delay = {int(row["net No."]): row["delay"] for _
    , row in df.iterrows()}
```

```
paths = []

def to_int(x):
    if isinstance(x, np.generic):
        x = x.item()
    if isinstance(x, float):
        return int(x)
    return x

def trace_net(net, current_path, primary_num):
    net = to_int(net)
    row = df[df["net No."] == net]
    if row.empty:
        return
    row = row.iloc[0]
    current_path.append(net)
    net_type = row["type"]
    if net_type == "inpt":
        paths.append([to_int(primary_num)] +
    current_path)
        return
    elif net_type == "from":
        next_net = row["fanout"]
        trace_net(next_net, current_path,
    primary_num)
    elif net_type in ["nand", "nor", "and", "or", "
    xor", "xnor"]:
        fanin1 = row["fanin1"]
        fanin2 = row["fanin2"]
        trace_net(fanin1, current_path.copy(),
    primary_num)
        trace_net(fanin2, current_path.copy(),
    primary_num)
    elif net_type == "not":
        fanin1 = row["fanin1"]
        trace_net(fanin1, current_path.copy(),
    primary_num)
    else:
        return

for idx, row in df.iterrows():
    if row["fanout"] == 0:
        primary_number = row["net No."]
        trace_net(row["fanin1"], [], primary_number)
        trace_net(row["fanin2"], [], primary_number)

output_rows = []
frequencies = []

print("Traced paths with total delays and
    frequencies:")
for path in paths:
    total_delay = 0
    for net in path:
        total_delay += net_delay.get(net, 0)
    frequency = 1 / total_delay if total_delay != 0
    else 0
    frequencies.append(frequency)
    print(f"{path} - [{total_delay}] - [{frequency
    :.4f}]")
    output_rows.append({
        "Path": path,
        "Total Delay": total_delay,
        "Frequency": round(frequency, 4)
    })

if frequencies:
    non_zero_frequencies = [f for f in frequencies
    if f > 0]
    if non_zero_frequencies:
        lowest_frequency = min(non_zero_frequencies)
        summary_text = f"maximum possible frequency
    for this circuit is: {lowest_frequency:.4f} Hz"
    else:
        summary_text = "No non-zero frequency
    computed."
else:
    summary_text = "No valid paths were traced."

print("\\n" + summary_text)

output_df = pd.DataFrame(output_rows)
summary_row = pd.DataFrame({
    "Path": [summary_text],
    "Total Delay": [""],
    "Frequency": [""]
})
output_df = pd.concat([output_df, summary_row],
    ignore_index=True)
output_df.to_csv(output_csv, index=False)

print(f"\\nOutput saved to {output_csv}")
```

Listing 1: Complete Python Implementation

B. Analysis Results

For the provided test circuit, the analysis produced the following results:

TABLE I: Critical Path Summary

Total Paths Analyzed	11
Critical Path Delay	23 units
Maximum Frequency	0.0435 Hz
Critical Path	[23, 21, 16, 14, 11, 4, 6]

V. Discussion

Our results confirm that static timing analysis is a reliable method for determining the highest possible operating frequency of a digital circuit. By tracing all possible paths and accumulating their delays, we identify the bottleneck that ultimately limits performance. Designers can then optimize this critical path through gate restructuring or technology improvements to achieve higher speeds.

Conclusion

This paper has demonstrated how to calculate the maximum operating frequency of a digital circuit using static timing analysis. The combination of theoretical insights and practical Python implementation provides a clear roadmap for identifying and analyzing critical paths. The methodology is applicable to a wide range of digital designs, offering a valuable tool for both academic study and industrial development.

Appendices

Input Netlist Example (netlist (1).csv)

```
net No.,con,type,fanout,fanin,fanin1,fanin2,delay
1,gat,inpt,1,0,,,2
10,gat,nand,1,2,1,8,5
22,gat,nand,0,2,10,20,5
23,gat,nand,0,2,21,19,5
```

Listing 2: netlist (1).csv Snippet

Output Results Example (traced$_p aths_o utput.csv$)

```
Path,Total Delay,Frequency
"[22, 10, 1]",12,0.0833
"[22, 20, 16, 14, 11, 4, 6]",23,0.0435
"maximum possible frequency for this circuit is:
    0.0435 Hz",,
```

Listing 3: traced$_p aths_o utput.csv Snippet$

REFERENCES

[1] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.

[2] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed., Addison-Wesley, 2010.

[3] D. Chinnery and K. Keutzer, *Closing the Gap Between ASIC & Custom: Tools and Techniques for High-Performance ASIC Design*, Kluwer, 2002.

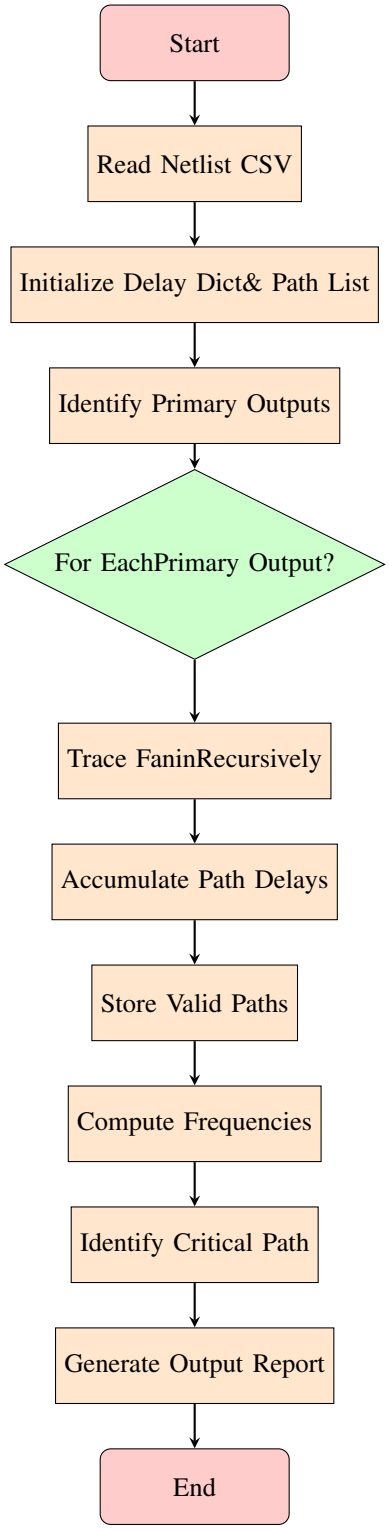

Fig. 1: Flowchart of the Static Timing Analysis Process