Problem 1: Static Timing Analysis for Combinational Circuits

2005 ICCAD/SIGDA CADathlon

1. Introduction

A static timing analyzer is one of the most important timing verification tools in today's VLSI design flow. Compared to simulation based verification methods, it doesn't require a set of input vectors, which can be very difficult to generate. It also has superior runtime efficiency. Because of these advantages, nowadays virtually all VLSI designs have to be verified by a static timing analysis tool, quite often multiple times.

Modern static timing analyzers are complex. Most of them are based on critical path method (CPM). A more detailed description of the CPM can be found in the reference. In a nutshell, in CPM the combinational circuit is represented by a (directed) timing graph, in which each gate is presented by a vertex, while each interconnection is represented by an edge. Once the arrival times (AT) at each primary input (PI) are given, a graph traversal can be performed to propagate the AT to primary outputs (PO) via all internal vertexes, using only two mathematical operations: add and max. If the required times (RT) are specified at each PO, a reverse graph traversal can be performed on the same graph, using subtract and min operation to propagate the RT back to PI. At each vertex, the difference between RT and AT is the slack, which can be used to determine which gates should be optimized in order to improve the overall AT of the circuit.

2. Problem statement

2.1. Brief description

Given a combinational circuit and a pre−characterized cell library, write a CPM−based program to calculate the arrival time (AT) at all primary outputs (POs) and required time (RT) at all primary inputs (PIs). Also calculate all most critical paths for each PO. You may assume that the fanout of all gates is one.

2.2. More detailed description

The reference provides a clear explanation of the traversal method to propagate arrival times from PIs to POs. We do not consider false paths and path re−convergence.

The propagation of required times is a dual problem of AT propagation. Details are also available in the reference.

There may be multiple critical paths for each PO, whose total path delays equal the AT at the particular PO. A trivial example is a circuit that consists of only a NAND2 gate with zero arrival time at both inputs. It is easy to verify that for each signal edge, there are two critical paths, whose path delays equal the NAND2 gate delay for the corresponding signal edge. A traversal algorithm can be written to calculate all of them.

As will be explained in the following section, the calculation of timing information requires the combinational netlist and a pre−characterized cell library. A parser has been provided to load both the library and the netlist into memory. Some printing routines are also provided. You are required to implement the three traversal routines: void forward_traversal() to propagate ATs, void reverse_traversal() to calculate the RTs, and void traversal_critical_path() to calculate all critical paths for each PO. To facilitate the implementation, a vertex queue is provided and initialized prior to calling forward_traversal() and reverse_traversal(). The correct timing info (AT and RT) should be inserted into the TimingGraph *pG. The output of traversal_critical_path() is an array of stacks of edges, each stack includes PI and PO. For each stack, after a successful traversal, the top of the stack should be the PI on the critical path, followed by the first edge of the path, and so on. Since the critical path can be different for rising and falling edges, two traversals are needed. The critical path itself is specified as a sequence of edges specified in the stack. Please read the printing routines carefully to get a good understanding. Note that it is your responsibility to allocate the stacks and populate them. The main routine st_main() will only traverse them individually and delete them. The number of the critical paths should also be returned.

2.3. Input/output specification

The input to the program is the name of a file containing a netlist of a combinational circuit. The program should be invoked like this:

```
./st_main file
```

2.3.1. Input

The combinational netlists are also stored in text files. A simple parser is provided to parse the netlist and load the circuit into memory. For your reference, the syntax of the netlist and a sample can be found in timing_graph.h.

The program will also read a library containing timing information of the existing cells. We use a simple library that ignores the dependency of the transition delay to the input slew and driving capacitive load. Also a function has been written to load a library specified as text file into memory.

2.3.1.1. Example input

The following is a sample of the input netlist:

```
# example circuit
# inputs
PI a 0.0 0.0
PI b 0.0 0.0
PI c 0.0 0.0
PI d 0.0 0.0
PI e 0.0 0.0
PI f 0.0 0.0
PI g 0.0 0.0
PI h 0.0 0.0

# outputs
PO o 10.0 10.0

# gates
# out gate_type gate_name inputs
i NAND2_A ig a b
j NAND2_B jg c d
k NAND2_C kg e f
l NAND2_B lg g h
```

```
m NAND2_D mg j k
n NAND2_E ng i m
p NAND2_F pg m l
o NAND2_G og n p

# end
```

2.3.2. Output

Printing routines are provided to print the AT at all POs, the RT at all PIs and the critical paths for each PO. Please do not modify those routines.

2.3.2.1. Example output

The following is the output of the netlist shown above, with the correct implementation of the required functions:

```
AT PO: o -> 7.0/11.0
RT PI: a -> 3.0/5.0
RT PI: b -> 3.0/5.0
RT PI: c -> 4.0/-1.0
RT PI: d -> 4.0/-1.0
RT PI: e -> 5.0/0.0
RT PI: f -> 5.0/0.0
RT PI: g -> 3.0/3.0
RT PI: h -> 3.0/3.0
SLACK i: +5/+3
SLACK j: -1/+4
SLACK k: +0/+5
SLACK l: +3/+3
SLACK m: +4/-1
SLACK n: -1/+4
SLACK p: +0/+3
CRIT PATH o(R): g->l->p->o
CRIT PATH o(R): h->l->p->o
CRIT PATH o(F): c->j->m->n->o
CRIT PATH o(F): d->j->m->n->o
```

3. Additional information

The manuals of some STL classes are also provided.

4. References

- Timing analysis for combinational circuits, Chapter 5, from "Timing" by Sachin Sapatnekar, Kluwer Academic Publishers, 2004