UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

Department of Computer & Communication Engineering

TIMING ANALYSIS OF INTEGRATED CIRCUITS

Master Thesis :

Alexandros Mittas Lazaridis

July 2012

Volos - Greece

DEDICATION

To my parents and friends

Hope for better days.

ACKNOWLEDGMENTS

First I would like to thank Dr. George Stamoulis for advising me for the last 4 years. I have learned many things from him and consider myself fortunate to have been one of his students.

I would also like to thank Dr. Nestoras Eymorfopoulos and Dr. Ioannis Moudanos. Without their patience and crucial support this thesis would not have been completed.

Finally I am really grateful to my roommates in E5 room of Glavani Steet whose help was really appreciated. Konstantis, Giorgos, Babis, Tasos, Sofia, and Alexia I am really obliged. Forgive me if having forgotten to mention anyone.

Chapter 1. Introduction

1.1 Goal of this thesis

Given a digital Integrated Circuit described in a hardware description language we will analyze its timing behavior. This is why we have implemented an EDA tool for timing analysis of Integrated Circuits. We are going to present the factors that affect timing values of logic gates such as delays or transition times.

The circuit may consist of both combinational and sequential elements. Out tool has the ability to analyze circuits consisting of NAND, NOR, AND, OR, NOT, MUX, OAI, AOI, DFF, DFFR, and many other gates.

We are going to describe the methodology for creating a timing analysis tool which is capable of finding each path's delay in a circuit and analyze the timing behavior of various circuits given some timing constraints. The last may guide in a violation of the circuit's clock frequency which means that the hardware description of the circuit is incorrect, as it violates the constraints dictated by the clock.

1.2 Moore's Law

Moore's Law is a rule of thumb in the history of computing hardware whereby the number of transistors that can be placed inexpensively on an integrated circuit doubles approximately every two years. This trend has continued for more than half a century. 2005 sources expected it to continue until at least 2015 or 2020. However, the 2010 update to the International Technology Roadmap for Semiconductors has growth slowing at the end of 2013, after which time transistor counts and densities are to double only every 3 years.

The capabilities of many digital electronic devices are strongly linked to Moore's law: processing speed, memory capacity, sensors and even the number and size of pixels in digital cameras. All of these are improving at (roughly) exponential rates as well. This exponential improvement has dramatically enhanced the impact of digital electronics in nearly every segment of the world economy.

.

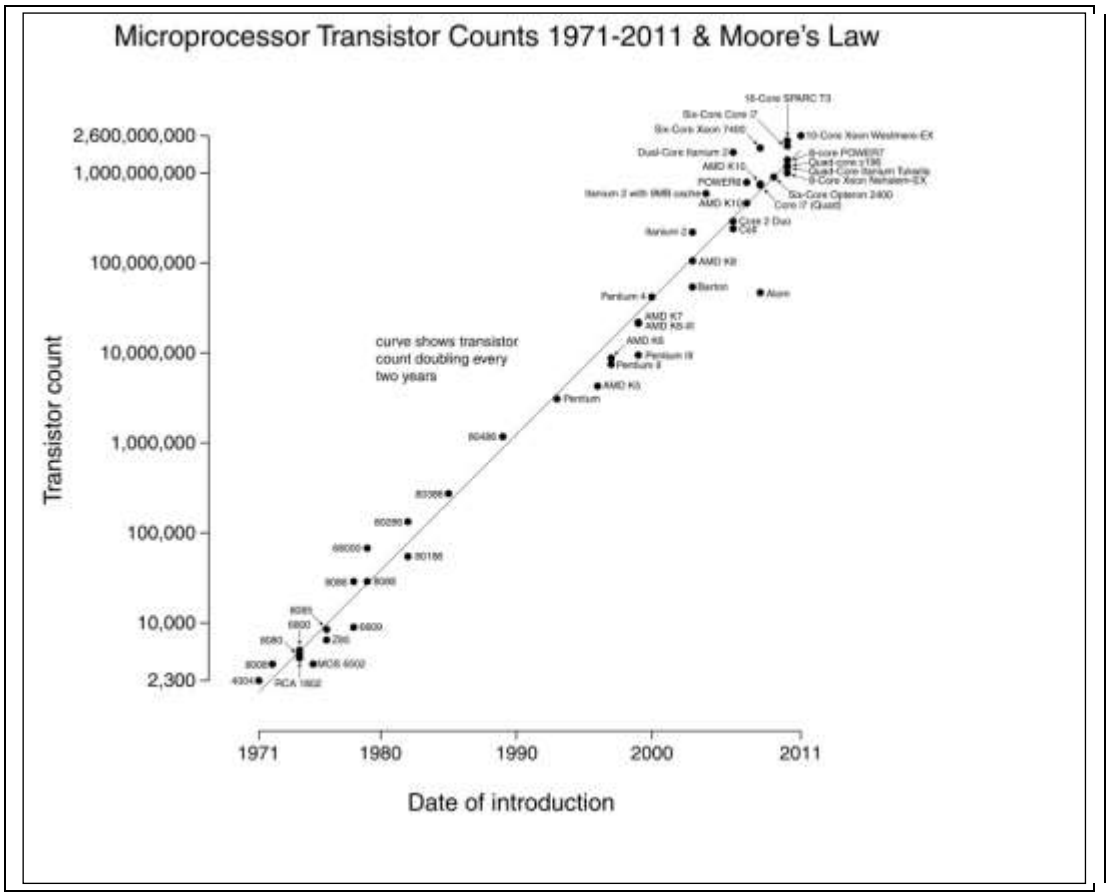

Figure 1.1 Moore's Law

Consequently, circuits consist of even more transistors. Even more transistors mean larger area and distance for a signal to be propagated. Each Integrated Circuits' Developer has to worry about the constraints that these technological achievements induce. The current master thesis is closely connected with the previous developments as our goal is to determine the timing values of a digital circuit.

Chapter 2. TIMING ANALYSIS

2.1 What is Timing Analysis

Static Timing Analysis (STA) is a method of computing the expected timing of a digital circuit without requiring simulation. High-performance integrated circuits have traditionally been characterized by the clock frequency at which they operate. Gauging the ability of a circuit to operate at the specified speed requires an ability to measure, during the design process, its delay at numerous steps. Moreover, delay calculation must be incorporated into the inner loop of timing optimizers at various phases of design, such as logic synthesis, layout (placement and routing), and in in-place optimizations performed late in the design cycle. While such timing measurements can theoretically be performed using a rigorous circuit simulation, such an approach is liable to be too slow to be practical. Static timing analysis plays a vital role in facilitating the fast and reasonably accurate measurement of circuit timing. The speedup appears due to the use of simplified delay models, and on account of the fact that its ability to consider the effects of logical interactions between signals is limited. Nevertheless, it has become a mainstay of design over the last few decades.

2.2 Types of Timing Analysis

There are 3 types of timing analysis widely-used to verify the behavior of an Integrated Circuit:

1. **Static Timing Analysis**
2. **Dynamic Timing Analysis**
3. **Statistical Timing Analysis**

- Problems with both the first two approaches have resulted in the formation of a new tool category **hybrid timing verification**. It selectively combine both static and dynamic timing in an attempt to create the best of both worlds.

2.3 Static Timing Analysis

Static Timing Analysis (also referred as STA) is one of the many techniques available to verify the timing of a digital design. An alternate approach used to verify the timing is the timing simulation which can verify the functionality as well as the timing of the design. The term timing analysis is used to refer to either of these two methods - static timing analysis, or the timing simulation. Thus, timing analysis simply refers to the analysis of the design for timing issues.

The STA is static since the analysis of the design is carried out statically and does not depend upon the data values being applied at the input pins. This is in contrast to simulation based timing analysis where a stimulus is applied on input signals, resulting behavior is observed and verified, then time is advanced with new input stimulus applied, and the new behavior is observed and verified and so on.

Given a design along with a set of input clock definitions and the definition of the external environment of the design, the purpose of static timing analysis is to validate if the design can operate at the rated speed. That is ,the design can operate safely at the specified frequency of the clocks with-out any timing violations. Figure 2-1 shows the basic functionality of static timing analysis. The DUA is the design under analysis. Some examples of timing checks are setup and hold checks. A setup check ensures that the data can arrive at a flip-flop within the given clock period. A hold check ensures that the data is held for at least a minimum time so that there is no unexpected pass-through of data through a flip-flop: that is, it ensures that a flip-flop captures the intended data correctly. These checks ensure that the proper data is ready and available for capture and latched in for the new state.

The more important aspect of static timing analysis is that the entire design is analyzed once and the required timing checks are performed for all possible paths and scenarios of the design. Thus, STA is a complete and exhaustive method for verifying the timing of a design.

The design under analysis is typically specified using a hardware description language such as VHDL or Verilog HDL.

2.4 Static Vs Dynamic Timing Analysis

- **Dynamic timing** analysis uses simulation vectors to verify that the circuit computes accurate results from a given input without any timing violations. The problem is that the simulations vector not can guarantee 100% coverage. The goal for the dynamic analysis is to get a 100% coverage. Dynamic timing simulation is still preferred for non-synchronous logic style. As a rule, however, only dynamic timing verification tools support glitch detection and race conditions, since these are inherently dynamic events.

- **Static timing** analysis on the other hand check all path in the circuit even the false paths. False paths are paths that are not possible or interesting in actual operation of the circuit. Therefore you can say that static analysis starts above 100% and works towards 100% by detecting and excluding the false paths. Static tools have made major advancements in recent years, in fact all synthesis tools use static timing analysis internally. Something good about this approach is that almost all tools using it supports multi-cycle paths, in which a path delay constraint exceeds a single clock period. Everything isn't just good, many static timing tools have problems with feedback loops.

2.5 Why Static Timing Analysis

Static timing analysis is a complete and exhaustive verification of all timing checks of a design. Other timing analysis methods such as simulation can only verify the portions of the design that get exercised by stimulus. Verification through timing simulation is only as exhaustive as the test vectors used. To simulate and verify all timing conditions of a design with 10-100million gates is very slow and the timing cannot be verified completely .Thus, it is very difficult to do exhaustive verification through simulation. Static timing analysis on the other hand provides a faster and simpler way of checking and analyzing all the timing paths in a design for any timing violations. Given the complexity of present day ASICs, which may contain 10 to 100 million gates, the static timing analysis has become a necessity to exhaustively verify the timing of a design.

Chapter 3. METHODOLOGY

3.1 In general

3.1.1 Problem Description

In order to achieve the goal of this master thesis we need a digital circuit's design in hardware description language. This file would be one of the two crucial inputs to our application for getting a timing report for that circuit.

We have used the following two crucial inputs to our application:

1. Some integrated circuits written in **Hardware Description Language (VHDL).** Of course our work can give results for any VHDL circuit or be expanded so as to perform the same operations for any other **hardware description language** (e.g. **Verilog**). The circuits that have been used are **ISCAS Benchmark Circuits '89** and b circuits which consist form simple to large-scale circuits consisting of tens of thousands of components.

2. a **Standard Cell Library** is a collection of low-level logic-functions such as AND, OR, INVERT, flip-flops, latches, and buffers. These cells are realized as fixed-height, variable-width full-custom cells. The key aspect with these libraries is that they are of a fixed height, which enables them to be placed in rows, easing the process of automated digital layout. The cells are typically optimized full-custom layouts, which minimize delays and area.

In first step these two types of input files have been read and their data have been stored in appropriate data structures for later usage.

After having fully represented in our systems memory the interconnection of the circuit, we divide it in levels and subsequently perform computational operations to this data such as computation of capacitances, delays, transitions e.t.c.

3.1.2 Programming Language and Environment

Before expand in detail the process followed we have to mention that our EDA-tool is written on C programming language. This is not at all a random choice as the previously mentioned programming language can boasts for a variety of advantages that other languages do not.

For example a C written application is “light” program, with no extremely high storing demands that can be compiled and executed quickly. This is one of the most important reasons for why many industrial applications even whole operating systems are being programmed in C. What is more C shows high portability and the code can be executed in any machine. There is only need of two simple commands for compilation and execution in console’s prompt. Last but not least even if a high-level programming language C is quite close to a CPU’s language (assembly language) as it makes available a significant tool to the programmer, immediate memory usage. In C memory allocation and deallocation is a hardcore bit by bit operation and the programmer can access memory bytes by using pointers on them

Our code has been written in ubuntu UNIX environment and has been compiled and executed with the following two commands.

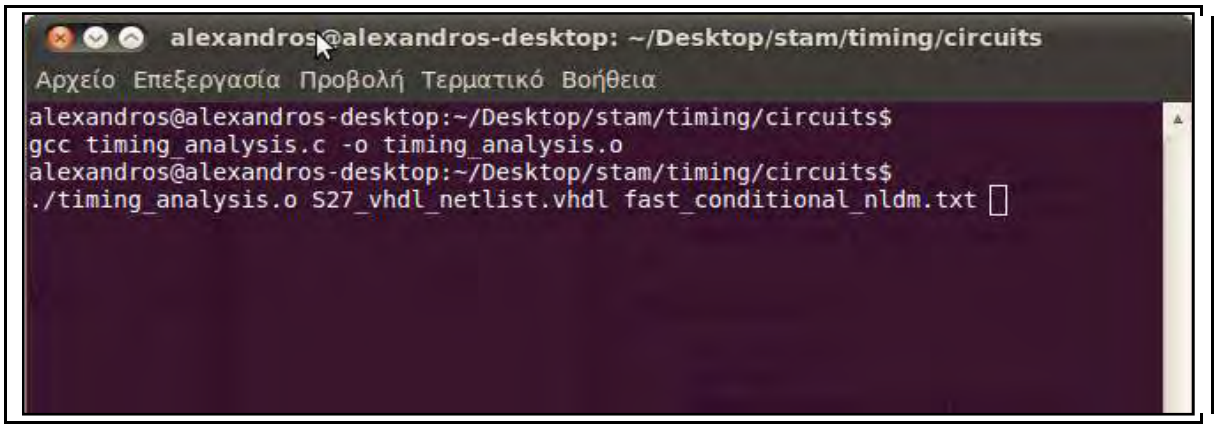

Figure 3.1 Compilation and execution of our code

Where timing_analysis.c is our EDA tool’s source code and timing_analysis.o the relevant executable file. The two input files coming next on the execution command are

1. S27_vhdl_netlist.vhdl: a netlist of a circuit
2. fast_conditional_nldm.txt: a Standard Cell Library

3.2 Parsing input Files

3.2.1 Parsing an Integrated Circuit

A digital circuit consists of **primary inputs**, **primary outputs**, **logic-gates** (**components**) and **nets** connecting the gates (**signals**). The following VHDL commands

```
port( S1, S2, S3, S4, S6, S8, S10 : in std_logic;
      S7, S11, S5, S9_out         : out std_logic);
```

Figure 3.2 inputs and outputs of a digital circuit described in VHDL

declares that primary inputs and primary outputs of this circuit are:

S1, S2, S3, S4, S6, S8, S10

S7, S11, S5, S9_out

respectively

whereas the following command declares the nets names that connect the components

```
signal net285, net286, net292, net305, net291, net282,
       net281, n6, n7, n8, n9, n10, n11, n12, n13 : std_logic;
```

Figure 3.3 nets of a digital circuit described in VHDL

Finally when reading a *port map* command

```
U13 : NAND2_X4 port map( A1 => S6, A2 => S2, ZN => net282);
```

Figure 3.4 example of *port map( )*

The input and output connections the name and the type of the component are being declared. This gate for example is U13 and is of type NAND2_X4. Also on pins A1, A2 of the gate the primary inputs S6, S2 are being inserted respectively and has an output on net net282 which will be used as input on another component.

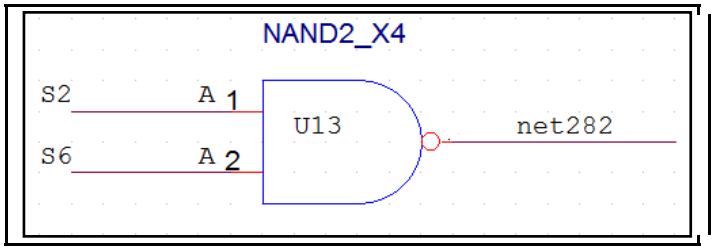

Figure 3.5 schematic representation of a NAND-gate VHDL description

We defined appropriate data structures in order to store in the memory the aforementioned information.

3.2.2 Parsing Standard Cell Library

NanGate 45nm Open Cell Library

For the purposes of the current master thesis we have used the open-source library **NanGate 45nm Open Cell Library**. This library is appropriate for testing of integrated circuits.

Nangate has developed and donated this library to Si2 for open use. The library is intended to aid university research programs and organizations such as Si2 in developing flows, developing circuits and exercising new algorithms. In its first release the Open Cell Library contains 38 different functions ranging from buffers to scan flip-flops with set and reset. All the different cell functions come in multiple drive strength variants end up with more than 100 different cells in the library.

The library was generated using Nangate's Library Creatorâ„¢ and the 45nm FreePDK Base Kit from North Carolina State University (NCSU) and characterization was done using the Predictive Technology Model (PTM) from Arizona State University (ASU).

The library is enhanced over time based on user suggestions and requests.

This Open Cell Library contains the following views:

- Liberty (.lib) formatted libraries with CCS Timing, ECSM Timing and NLDM/NLPM data (fast, slow and typical corners)
- Geometric library in Library Exchange Format (LEF)
- Simulation libraries in Verilog and Spice (pre and post parasitic extracted netlists)
- Cell layouts in GDSII
- Schematics
- Library databook in HTML/XML format
- OpenAccess database containing layouts and netlists

On the following we can see some library data related to the input pin A1 of a 2-input NAND gate.

Inputs data

```
cell (AND2_X1) {
        pin (A1) {

                direction               : input;
                capacitance             : 0.000543;
                fall_capacitance        : 0.000540;
                rise_capacitance        : 0.000547;
                fall_capacitance_range          (0.000486, 0.000601);
                rise_capacitance_range          (0.000469, 0.000609);
                max_transition          : 0.600000;
        }

        pin (A2) {
                ......................
        }

        pin (ZN) {
                ......................
        }
}
```

Figure 3.6 contents of **Standard Cell Library** of a 2-input NAND gate

We can see some information the input pin A1 of a 2-input NAND gate such as its capacitance and its max transition time.

Outputs data

```
pin (ZN) {
        direction               : output;
        max_capacitance         : 0.025600;
        max_transition          : 0.600000;
        function                : "(A1 & A2)";

        timing () {

                related_pin         : "A1";
                timing_sense        : positive_unate;

                cell_fall(Timing_data_X1) {
                        values ("0.023009,0.025401,0.029763,0.037776,0.053083,0.083403,0.144080", \
                                "0.028061,0.030452,0.034814,0.042824,0.058114,0.088432,0.149083", \
                                "0.034724,0.037276,0.041841,0.050021,0.065334,0.095600,0.156223", \
                                "0.044981,0.047800,0.052758,0.061396,0.076998,0.107269,0.167779", \
                                "0.061197,0.064453,0.070166,0.079888,0.096625,0.127442,0.187951", \
                                "0.086875,0.090849,0.097847,0.109399,0.128647,0.162130,0.224500", \
                                "0.130507,0.135411,0.143863,0.158202,0.181444,0.220061,0.288321");
                }
                cell_rise(Timing_data_X1) {
                        ....................................
                }
                fall_transition(Timing_data_X1) {
                        ....................................
                }
                rise_transition(Timing_data_X1) {
                        ....................................
                }
        }
}
```

Figure 3.7 contents of **Standard Cell Library** of a 2-input NAND gate's output

We can see values like the pin's capacitance or the logic function that the component implements as well as the cell fall matrix its timing sense and the input related to that data.

```
lu_table_template (Timing_data_X1) {
        variable_1 : input_net_transition;
        variable_2 : total_output_net_capacitance;
        index_1 ("0.007500,0.018750,0.037500,0.075000,0.150000,0.300000,0.600000")
        index_2 ("0.000400,0.000800,0.001600,0.003200,0.006400,0.012800,0.025600")
 }
```

Figure 3.8 look_up_table for interpolation in the previous matrices

All the aforementioned information is being stored in appropriately defined data structures just like in the previous sector.

3.3 COMPUTATIONS

3.3.1 LEVELIZATION OF THE CIRCUIT

The circuits gates can be categorized in levels according to the following simple observation

Each gate of level N has as inputs, output nets of gates of levels no larger than N-1.

Consequently a gate which has only primary inputs as input nets will by default be a level 0 gate. Whereas a gate which has as inputs the output of the previously mentioned gate and a primary input would be a level 2 gate and so on. According to this, each gate's level should be the maximum of all its inputs levels plus 1.

Resuming for a gate k of i inputs it is

$$\mathbf{level(k) = max\ (\ level(\ input_1\ ),\ level(\ input_2\ ),\ \dots\ ,\ level(\ input_i\ )\ ) + 1}$$

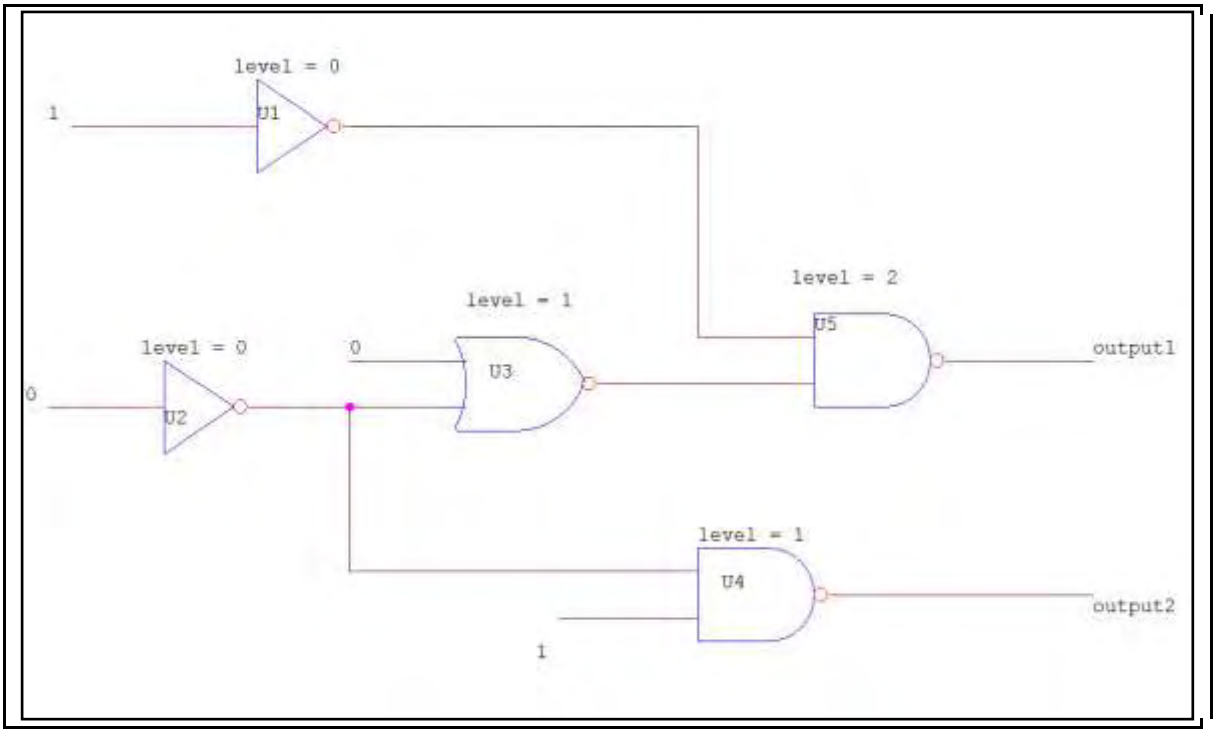

Figure 3.9 levels of gates of an integrated circuit

Violating the previously mentioned rule a flip flop's output net behaves as a primary input.

If consider a gate of two inputs, where the first input is a level one gate's output, whereas the second input is a flip flops output, then the gate's level would be

$$\mathbf{max\ (\ 1\ ,\ 0\ )\ +\ 1\ =\ 2}$$

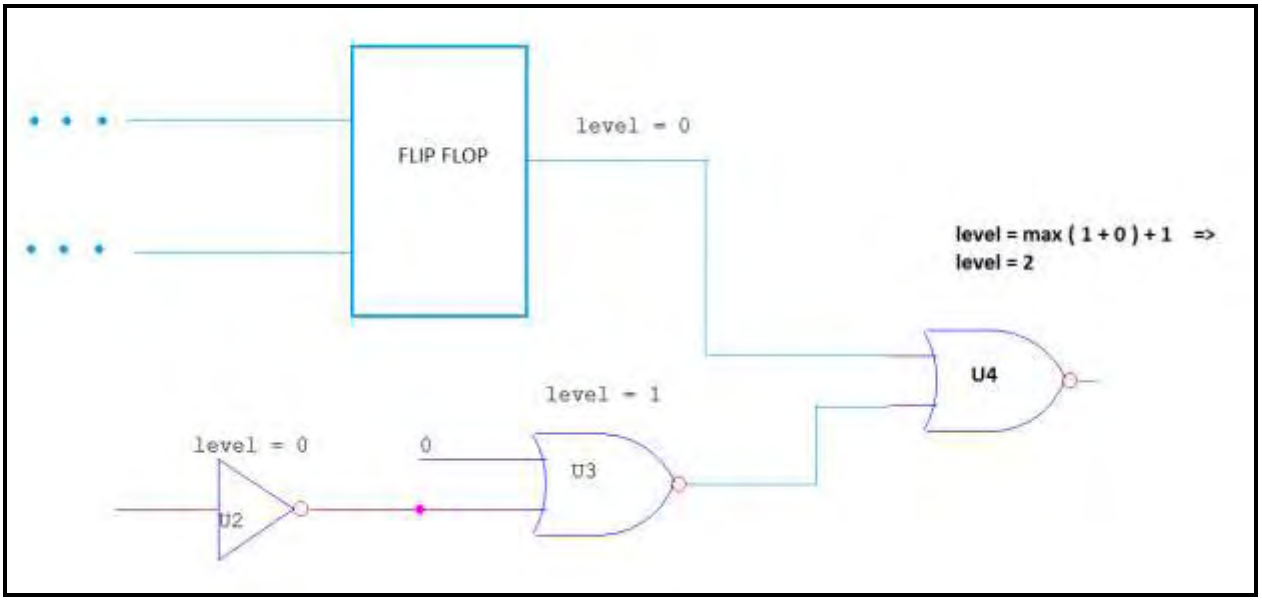

Figure 3.10 levels in sequential circuits

3.3.2 Output capacitance

The output capacitance of a gate equals to the sum of the capacitances of the input pins in the gates that this output is an input, plus the parasitic capacitance of the output net.

k : gate to compute it's output capacitance

$\mathbf{i_1}$ **, … ,** $\mathbf{i_n}$: gates which have as input the output net of k

$\mathbf{C_{out}}$: output capacitance

$\mathbf{C_{in}}$: input capacitance of a specific pin in a specific gate

$\mathbf{C_P}$: parasitic capacitance of the wire

Thus,

$$\mathbf{C_{out}(k) = C_{in} (i_1) + \dots + C_{in} (i_n) + Cp}$$

On **Figure 3.11** output capacitance of gate U1 is being calculated. At this moment to point out that the parasitic capacitance of the wire has not been added yet. Moreover to mention that for gates with multiple outputs such as a FULL/HALF ADDERS or FLIP FLOPS each output probably has different values of output capacitances, always with respect to the interconnections that these output nets form.

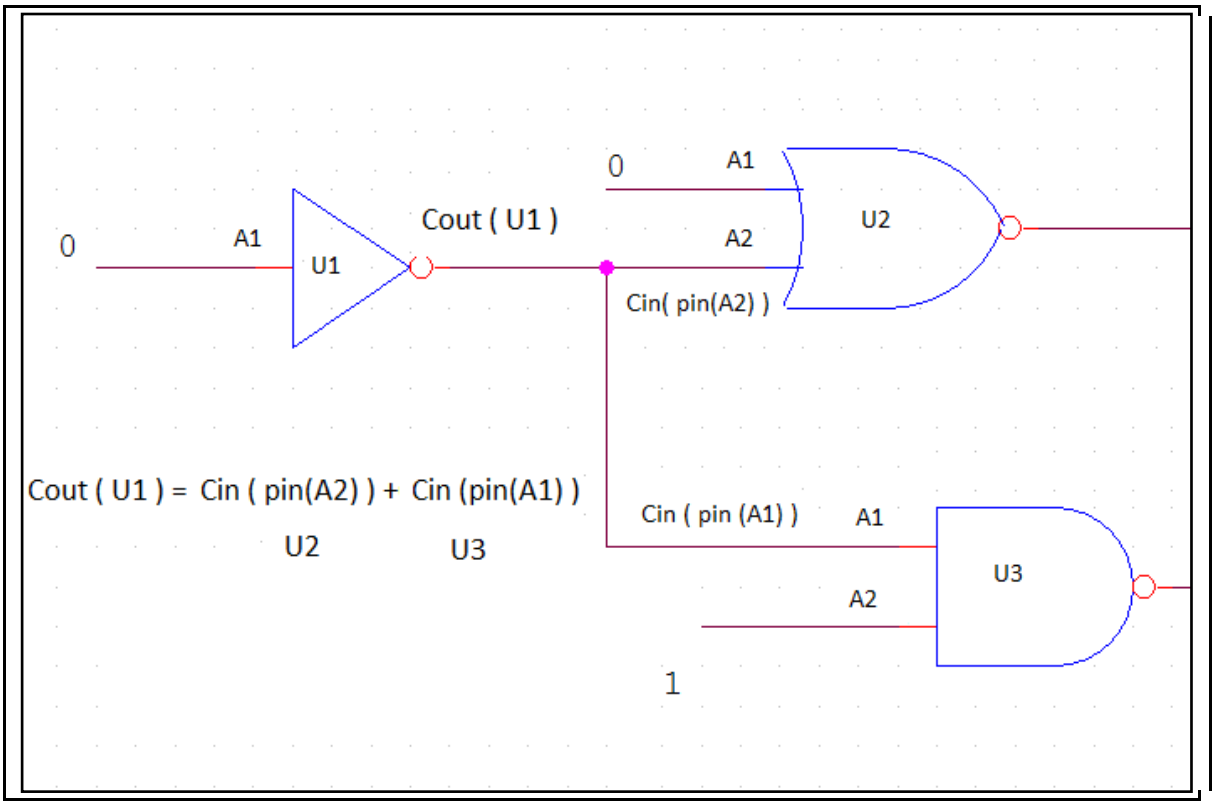

figure 3.11 computation of output capacitance of a gate (parasitic capacitance not included)

Parasitic Capacitance

Wireload models can be used to estimate capacitance, resistance and the area overhead due to interconnect. It is used to estimate the length of a net based upon the number of its fanouts. Given the following wireload model we can compute an approximation of the length of each component. For the not explicitly listed fanout values we can compute the interconnect length using linear extrapolation.

```
wire_load ("wlm_conservative"
  resistance : 5.0;
  capacitance : 1.1;
  area : 0.05;
  slope : 0.5;
  fanout_length (1, 2.6);
  fanout_length (2, 2.9);
  fanout_length (3, 3.2);
  fanout_length (4, 3.6);
  fanout_length (5, 4.1);
}
```

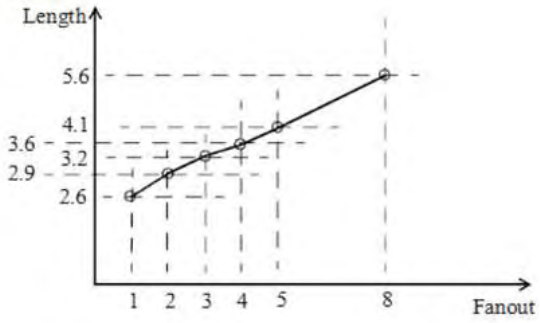

Figure 4-8 *Fanout vs wire length.*

```
Length = 4.1 + (8 - 5) * 0.5 = 5.6 units
Capacitance = Length * cap_coeff (1.1) = 6.16 units
Resistance = Length * res_coeff (5.0) = 28.0 units
Area overhead due to interconnect = Length * area_coeff (0.05)
  = 0.28 area units
```

Figure 3.12 wire load model and calculation of wire's length, capacitance, resistance and area

3.3.3 Values calculation

We now have to compute the delays and transitions of each component.

*Timing sense of a component's input pin (**positive**, **negative**, **non unate**)*

Each input pin of a gate can be **positive unate**, **negative unate** or **non unite**, that is how the output changes for different types of transitions on input.

The timing arc is **positive unate**, if a rising transition on an input causes the ouput to rise (or not to change). Whats more a falling transition on an input causes the output to fall (or not to change. For example an **AND** gate.

	AND 2	
input	input	output
A1	A2	Z
0	0	0
0	1	0
1	0	0
1	1	1

If an input from 1 turns 0, the output either turns 0 either remains the same

If an input from 0 turns 1, the output either turns 1 either remains the same

Table 3.1 True Table of 2 a input **AND** and its **timing sense** explanation

Conversely in a **negative unate** timing arc, a rising transition on an input causes the ouput to fall (or not to change). What iss more a falling transition on an input causes the output to rise (or not to change. For example a N**AND** gate

	NAND 2	
input	input	output
A1	A2	Z
0	0	1
0	1	1
1	0	1
1	1	0

If an input from 1 turns 0, the output either turns 1 either remains the same

If an input from 0 turns 1, the output either turns 0 either remains the same

Table 3.2 True Table of 2 a input N**AND** and its **timing sense** explanation

In a **non unate** timing arc cannot be determined only from the direction of change of an input, but also depends upon the state of the other inputs. For example a **XOR** gate.

Interpolation for delay-transition computation

Given the input transition time and the output capacitance of a gate we will calculate the gate's delays and transition times. Consider that input transition time in an input pin of gate equals to 0.22ns and output capacitance at an output pin equals 0.49pF. In order to compute for example the fall transition time in that timing arc we simply have to interpolate between the values 0.1937, 0.7280, 0.2327, 0.7676 and get the requested value. This procedure will be further explained in a following section.

fall_transition (delay_template_3X3){				
index_1("0.1, 0.3, 0.7"); /*input transition */				
index_2("0.16, 0.35, 1.43"); /* output capacitance */				
values (/*	0.16	0.35	1.43	*/
/* 0.1 */	"0.0817,	**0.1937,**	**0.7280",**	
/* 0.3 */	"0.1018,	**0.2327,**	**0.7676",**	
/*0.7 */	"0.1334,	0.2937,	0.8452");	

Table 3.3 matrix to interpolate the indexed values and get fall transition time

Compuation of delays-transtions (input and output)

For each of the matrices **cell_fall**, **cell_rise**, **fall_transition**, **rise_transition** we perform this interpolation operation and calculate the corresponding values.

- If a timing arc of a gate has **positive** timing sense,
    - then the **output fall transition** of the previous gate linked in that pin is the interpolated value for obtaining the **cell fall** and the **fall transition** values for that input pin.
    - Similarly the **output rise transition** of the previous gate linked in that pin is the interpolated value for obtaining the **cell rise** and the **rise transition** values for that input pin.

- If a timing arc of a gate has **negative** timing sense,
    - then the **output fall transition** of the previous gate linked in that pin is the interpolated value for obtaining the **cell rise** and the **rise transition** values for that input pin.
    - Similarly the **output rise transition** of the previous gate linked in that pin is the interpolated value for obtaining the **cell fall** and the **fall transition** values for that input pin.

Values computed	timing sense	Output transition interpolated
cell_fall / fall transition	positive	*fall_transition*
cell_rise / rise transition	positive	*rise_transition*
cell_fall / fall transition	negative	*rise_transition*
cell_rise / rise transition	negative	*fall_transition*

Table 3.4 computation of gate delays and transitions in dependence of timing sense of the pin and the output transition of the previous in the path gate

There are two types of static timing analysis that can be performed while computing gates' delays.

- max value analysis
- min value analysis

After having computed all **cell_fall**, **cell_rise**, **fall_transition**, **rise_transition** values for all gate's timing arcs we have to find the max/min fall and rise transition. These would be the output transitions (falling and rising) of a gate, which would be used as input transition in the gate which is connected to that output.

We have admit that input transition equals to 0 for all level 0 gates and to the previously level's output transition time for each non zero level gate.

3.3.4 Interpolation for values calculation

We will explain how the previous values are being calculated with usage of interpolation. Consider that we want to calculate **fall_transition** where input transition time in an input pin of gate equals to x_0 = 0.15ns and output capacitance at an output pin equals y_0 = 1.16pF.

x_1, x_2 are the input transition times from index1 for which it is $x_1 < x_0 < x_2$

y_1, y_2 are the output capacitances from index2 for which it is $y_1 < y_0 < y_2$.

We search and find for the fall_transition values in that range T_{11}, T_{12}, T_{21}, T_{22}

fall_transition (delay_template_3X3){
index_1("0.1, 0.3, 0.7"); /*input transition */
index_2("0.16, 0.35, 1.43"); /* output capacitance*/

values(/*	0.16	0.35	1.43 */
/* 0.1 */	"0.0817,	0.1937,	0.7280",
/* 0.3 */	"0.1018,	0.2327,	0.7676",
/*0.7 */	"0.1334,	0.2937,	0.8452");

Table 3.5 matrix for interpolation. The interpolated values are in light blue background

Now we calculate T_{00} for the specific x_0, y_0 as follows:

$$T_{00} = x_{20} * y_{20} * T_{11} + x_{20} * y_{01} * T_{12} + x_{01} * y_{20} * T_{21} + x_{01} * y_{01} * T_{22}$$

where

$x_{01} = (x_0-x_1) / (x_2-x_1)$

$x_{20} = (x_2-x_0) / (x_2-x_1)$

$y_{01} = (y_0-y_1) / (y_2-y_1)$

$y_{20} = (y_2-y_0) / (y_2-y_1)$

from the previous operations we can found that fall transition $\mathbf{T_{00} = 0.8516}$.

In case x_0 or y_0 or both is/are out of the borders specified by the indices values, then interpolation is being performed by the closest to that value elements. Of course this operation can be omitted in case that our data are matching exactly to the values of the indices. In such an occasion the requested value can be simply extracted by picking an element from the 2D matrix, without interpolation being needed, although this has been extremely rare as shown from our experiments.

3.3.5 Critical Path

Having followed the previously mentioned algorithm we computed all the required data in order to find the critical path and its latency. This path is the slowest of all possible paths in a circuit and consequently its importance is crucial for the verification of the timing requirements of the circuits design. If for example from the Register to Register paths the critical has delay larger than the clock's frequency then the circuit's implementation was not successful and it has to be recreated so meet the requested requirements.

So we have to compute the total delay up to a gate for each gate. For a max static timing analysis type total delay is the max sum of:

1) the delay of the cell
2) the total delay of a previous level gate whose output is input on that cell

so for each gate we have to compute #timings arcs sums and find the worst. Of course with respect to the timing sense of each timing arc in similar way that is described in **Table 3.5**

timing sense of timing arc	Cell delay	Total delay of previous gate
positive	cell rise / cell_fall	*Total delay rise / Total delay fall*
negative	cell_fall / cell rise	*Total delay rise / Total delay fall*

Table 3.6 computation of total delay up to a gate related to specific timing arc

To sum up consider a gate *g* with *k* timing arcs and $T_{up\_to\_pin\_i}$ to be the total delay until the input pin *i* of gate *g* whereas d_i is the cell delay of input *i*

then total delay of gate *g* is

$$T_g = \max (T_{up\_to\_pin\_0} + d_0 , \dots , T_{up\_to\_pin\_k} + d_k)$$

always with respect to timing sense of pins, so as to get rising and falling total delays.

It is obvious that total delay of a level 0 gate would be the max cell delay $T_g = \max (d_0 , \dots , d_k)$ as all its input are primary inputs of the circuit. The computation of the total delays is being performed from level 0 gates to higher level gates. First we trivially compute level 0 gates total delays, secondly level 1 gates total delays and so on.

After having computed the total delay of its gate we have to search for ones whose outputs are primary outputs for the whole circuit and find the gate with the highest value of total delay. Then we go backwards to the circuit to a lower level gate linked with the last whose total delay is the highest of all possible accessed gates. Then we go backwards again if necessary with the same criterion until we finally reach to a gate that its total delay is in dependence with some primary input and not with a lower level gate's output. The critical path has been found.

To note here that in circuits with sequential logic the previous process terminates when reach a sequential element. So the first component on the critical path would be for example a flip flop. This process should be used for critical path's track to all types of paths explained in a following section.

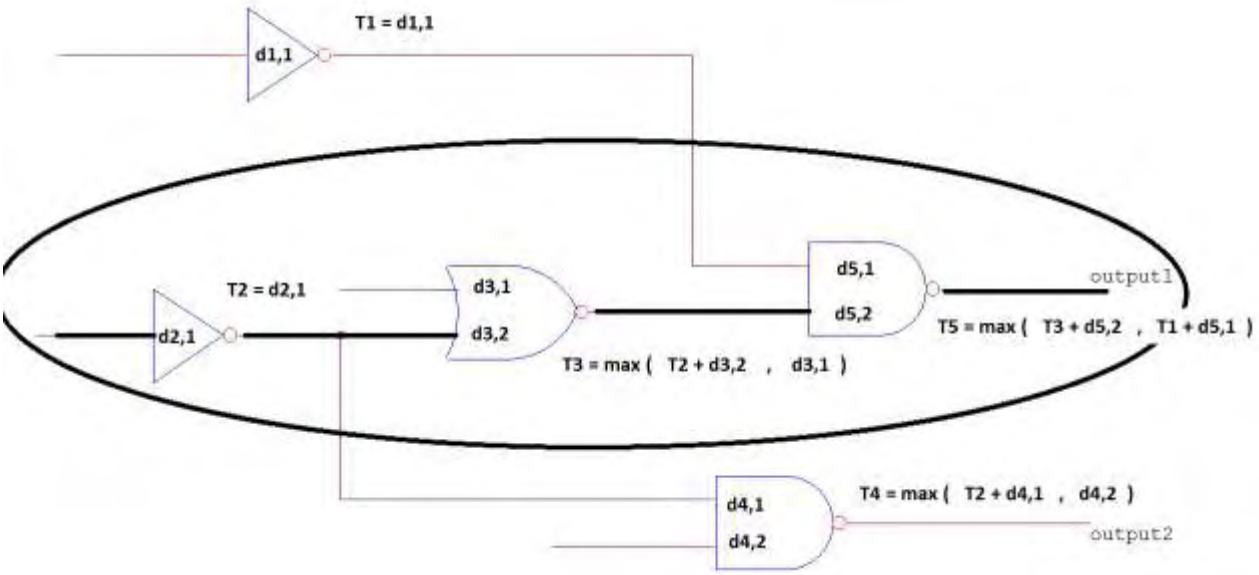

Figure 3.13 Critical Path

the minimum version of the algorithm to find the critical path is symmetrical.

3.3.6 Categorize different types of paths

On next step we have to distinguish all types of paths between inputs, outputs and registers. At first step we observe 4 different types of paths between sequential elements.

- from Primary Input to Register
- from Primary Input to Primary Output
- from Register to Primary Output
- from Register to Register

Consequently we recursively traverse the graph twice backwards.

- From all Primary Outputs. So we found the paths
    - from Primary Input to Primary Output
    - from Register to Primary Output
- From all Registers. So we found the paths
    - from Primary Input to Register
    - from Register to Register

after having determine the previous paths, and the total delay of each path up to a Register, we can check for **setup** and **hold** time violations with respect to these constraints.

Definitions:

- **Setup time** is the minimum amount of time the data signal should be held steady **before** the clock event so that the data are reliably sampled by the clock. This applies to synchronous circuits such as the flip-flop.

- **Hold time** is the minimum amount of time the data signal should be held steady **after** the clock event so that the data are reliably sampled. This also applies to synchronous circuits such as the flip-flop.

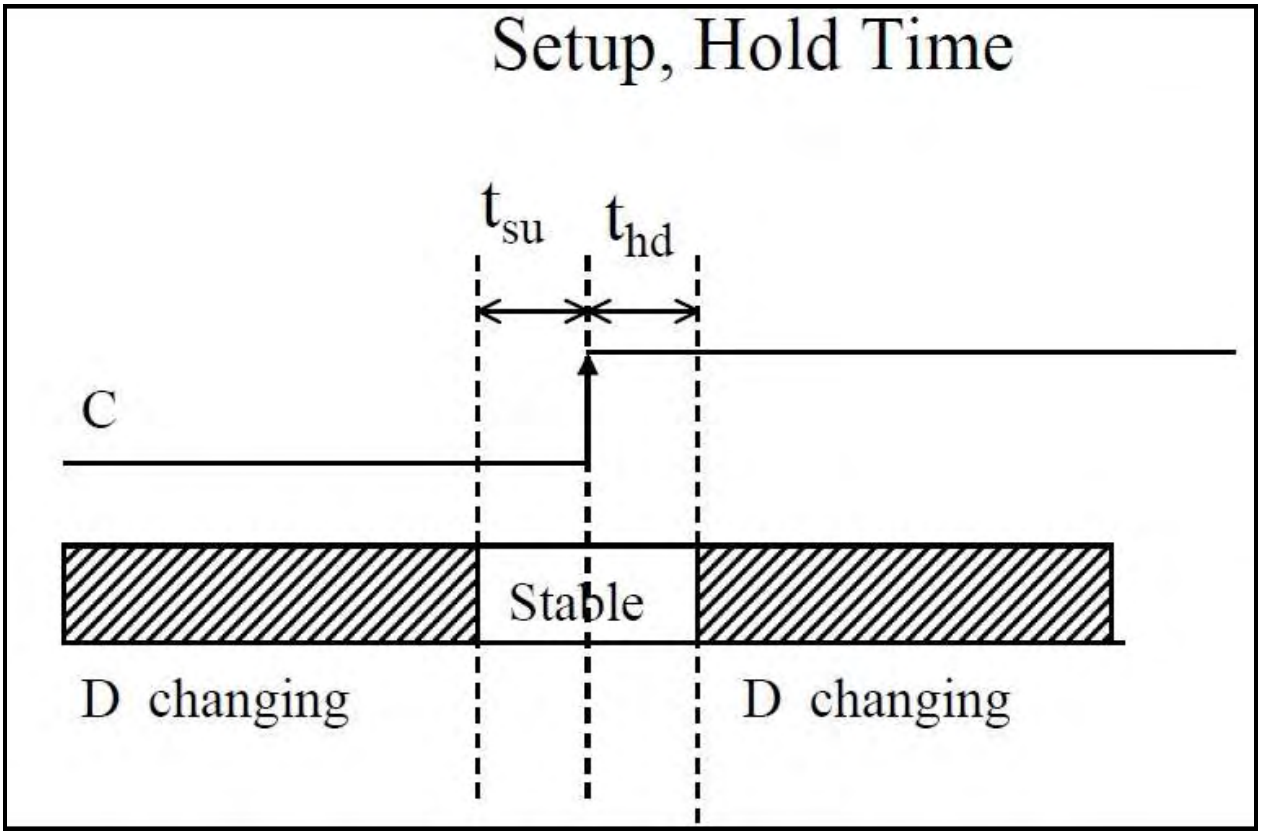

Figure 3.14 setup and hold values

These checks for violations in mathematical scope are equivalent to the following equations:

For the setup requirement it should be:

Trequire >= Tarrival

1. Register to Register
    - Tarrival = Tclk1 + TDFF1(clk->Q) + Tpath

- Trequire = Tclk2 - TDFF2(setup)
- Tslack = Trequire – Tarrival

2. Primary Input to Register
    - Tarrival = TPI(delay) + Tpath
    - Trequire = Tclk1 – TDFF1(setup)
    - Tslack = Trequire - Tarrival

3. Register to Primary Output
    - Tarrival = Tclk1 + TDFF1(clk->Q) + Tpath
    - Trequire = Tclk1 - TPO(output delay)
    - Tslack = Trequire - Tarrival

4. Primary Input to Primary Output
    - Tarrival = TPI(delay) + Tpath
    - Trequire = Tcycle – TPO(output delay)
    - Tslack = Trequire - Tarrival

To meet the hold time requirement it should be:

- Trequire <= Tarrival

1. Register to Register

    - Tarrival = Tclk1 + TDFF1(clk->Q) + Tpath
    - Trequire = Tclk2 - TDFF2(hold)
    - Tslack = Tarrival – Trequire

2. Primary Input to Register

    - Tarrival = TPI(delay) + Tpath
    - Trequire = Tclk - TDFF(hold)
    - Tslack = Tarrival - Trequire

3. Register to Primary Output

    - Tarrival = Tclk + TDFF(clk->Q) + Tpath
    - Trequire = - TPO(output delay)
    - Tslack = Tarrival - Trequire

4. Primary Input to Primary Output

    - Tarrival = TPI(delay) + Tpath
    - Trequire = - TPO(output delay)
    - Tslack = Tarrival - Trequire

3.3.7 Acceleration of the execution

Parsing vhdl file acceleration

Assuming that all signals are connected to simple linked list, while reading the netlist each input pin of each component has to be checked with each possible signal, so as to determine whether the last is input to that component or not. Consequently for a circuit of 1,000,000 signals and 1,000,000 components, each of whom, consists in middle case of about 3 inputs, we have to perform 5*105 * 3*106 = 15*1011 string comparison operations. This demands extremely high CPU usage and amount of time for the operations to be performed!

As a result we have to implement a more dynamic data structure to store the signals' information than a simply linked list. So we implement a multi-dimensional hash-table (referred as k-tree in data structure bibliography).

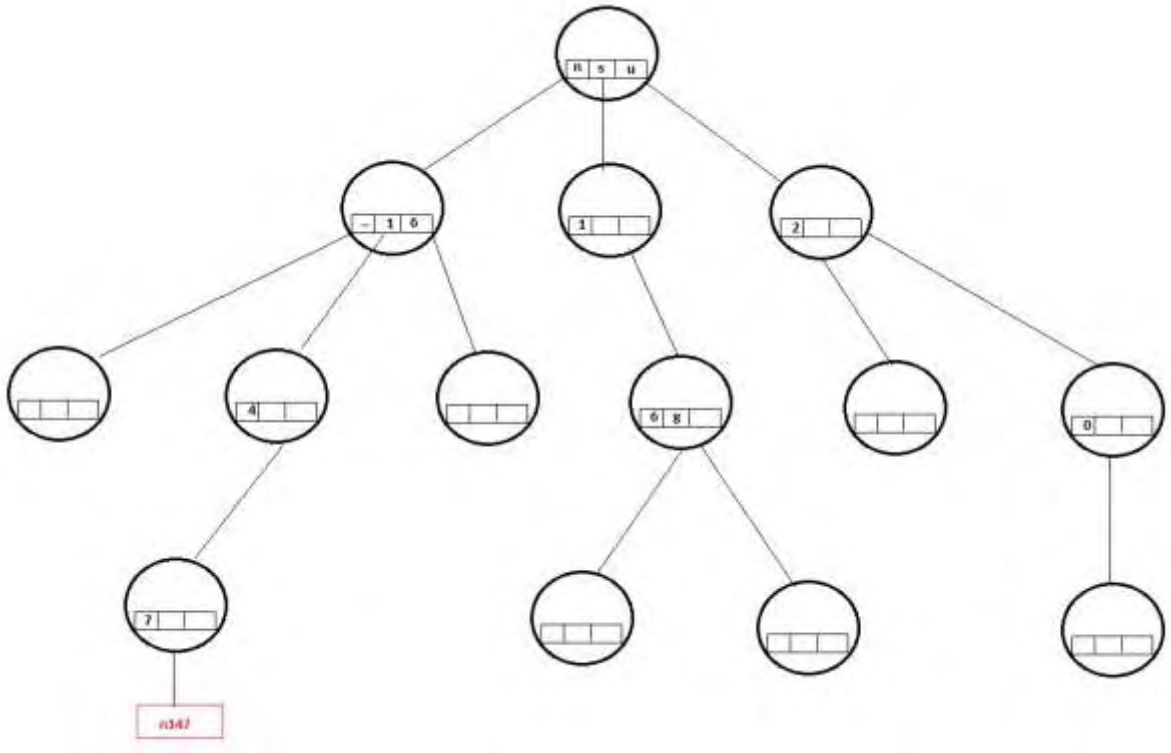

Figure 3.15 K-Tree example

Now a signal's id could be found in O(strlen(signal)) memory accesses whereas previously the complexity was in the average case O (signals_cnt / 2). However we have not added the cost of the linear search of the hashing characters of each node until we find the correct character, so as to move on a lower level of our tree hierarchy and so on. However if we consider that all possible signals id characters created by a Synthesis tool are [0-9, _ , n] or a few some more, in a average case each node has about 6 hash characters. So the new complexity is O(strlen(signal) * (1 + 6/2)) memory accesses, yet an extremely worth-implementing data structure.

Dynamic linked lists of components

While parsing the netlist we allocate memory and store linking information for each component.

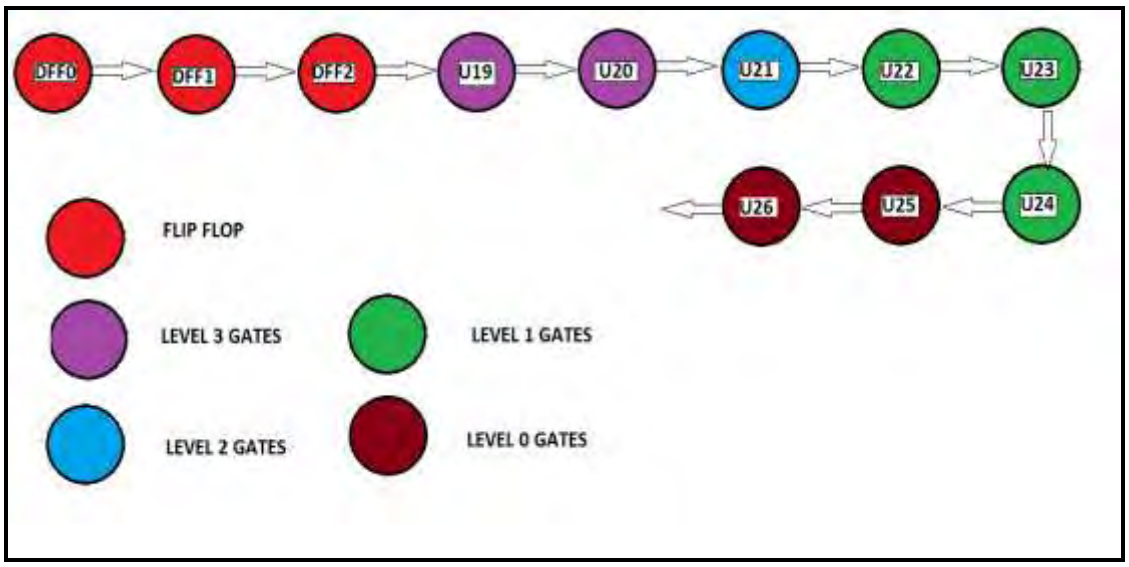

Figure 3.16 simply linked list of components

Consider a simply linked list where each component would be stored in the same order that there are read from the netlist. In many functions that we have implemented there is need to search all the components and firstly do some operations with the level 0 gates values',

secondly search the list again and do the same operations with level 1 gates values' and so on. For example computation of delays, transitions, and output capacitances of the gates' needs such an approach.

There is obviously a need to do some more dynamically linked data structure so as to access only the needed components in each level. Consequently during the levelization function we dynamically create #levels simply linked lists so as to categorize our components properly.

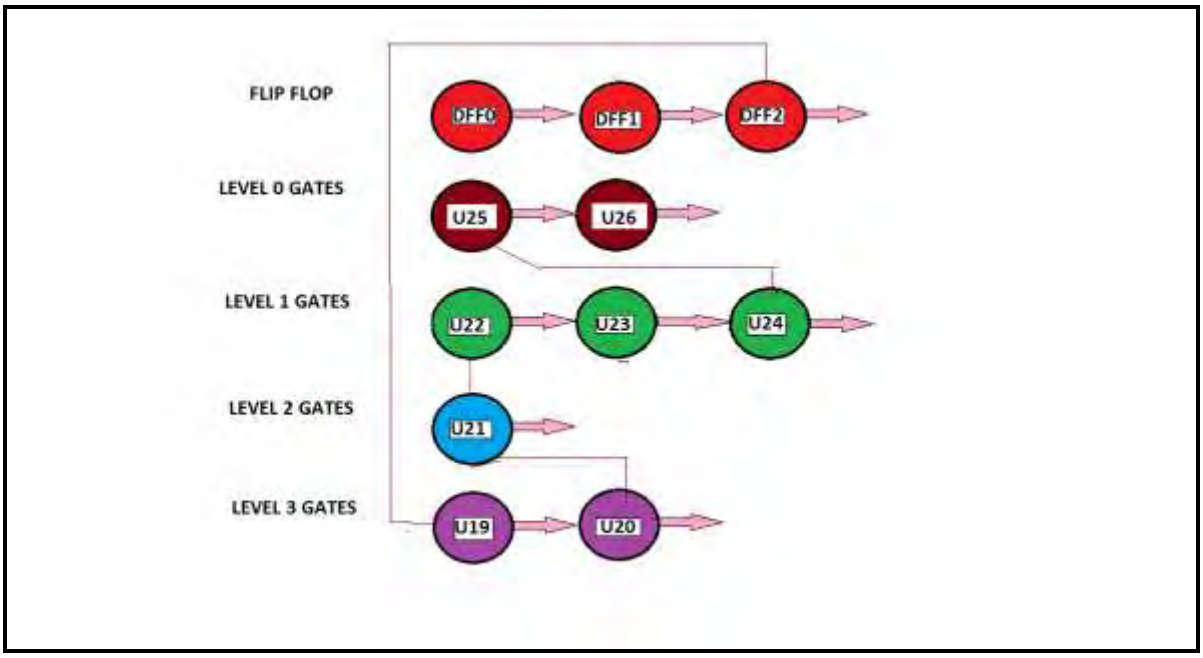

Figure 3.17 dynamic linked lists of components

3.3.8 Synthesis of ISCAS Benchmark Circuits ’89 and B-Circuits

For the purposes of the current master thesis we had to design some testing VHDL files in order to validate the correctness of our EDA tool. For synthesis of these circuits we used Synopsis Design Vision. In the following lines we present the methodology to create the aforementioned circuits.

1. We have to setup the synthesis environment. Declare the **link** and **target library**.

2. **Analysis** and **Elaboration** of the design

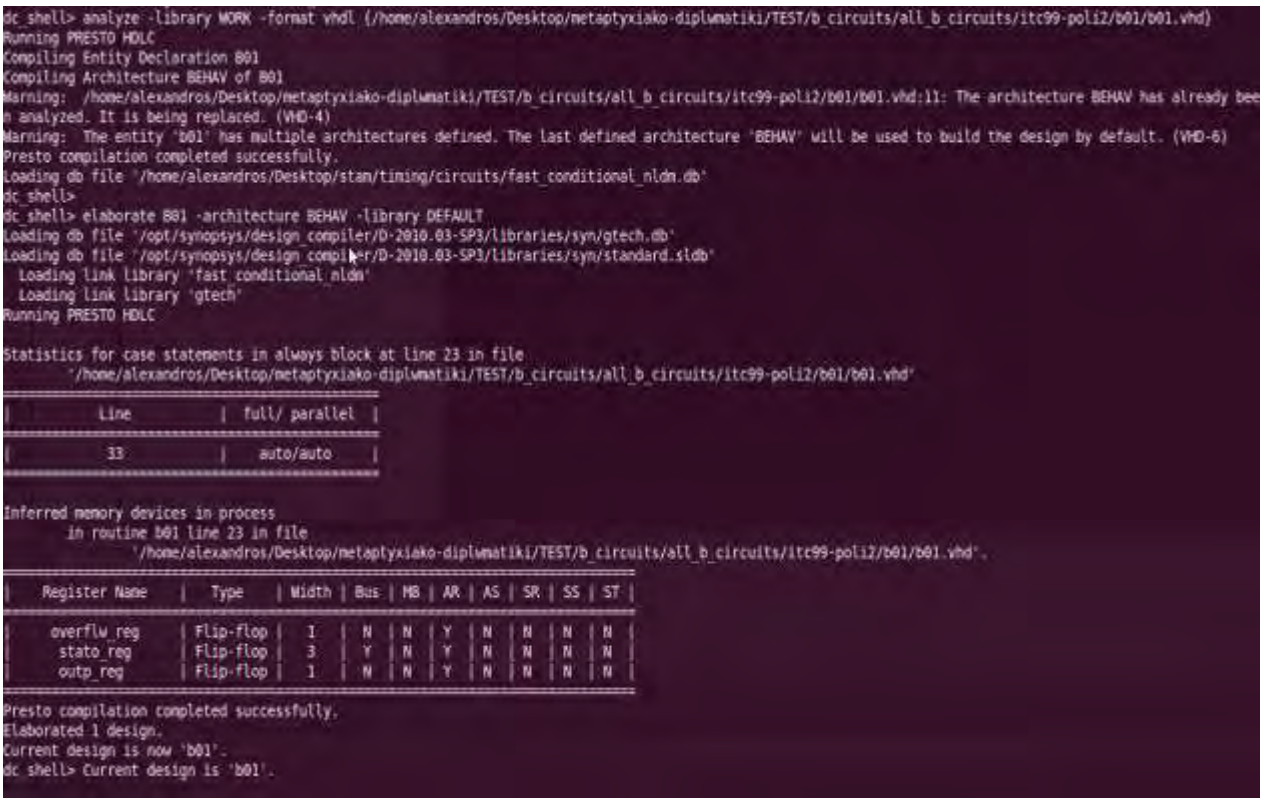

Figure 3.18 analysis and elaboration results

3. **Compile**

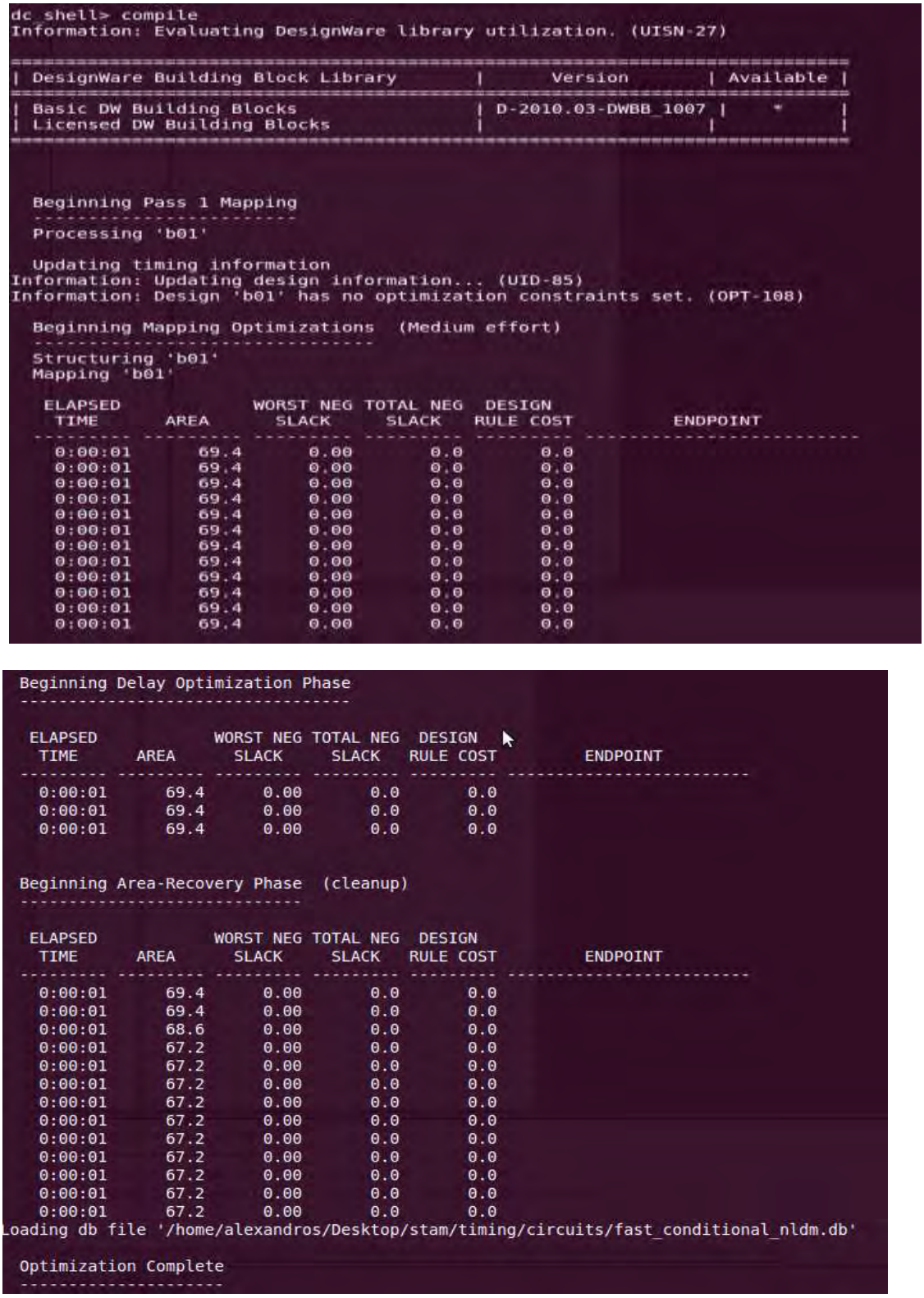

Figure 3.19 compile results

4. **Ungroup**

   Our tool is not implemented to analyze hierarchies of circuits. One circuit's description which consists of thousands of components will probably be synthesized as a set of subcomponents appropriately connected which on may contain subcomponents on their turn etc. As our tool was implemented to analyze only flat circuits we have to completely remove this hierarchy from the created netlist.

5. **Ungroup_bus**

   After synthesis they may be have defined some non-primitive VHDL types (e.g. some 64 bit vector). Our tool has the ability to use only primitive VHDL types such as std_logic. Consequently we have to analyze each non-primitive VHDL type to a set of separately declared bits.

6. After having done all the previous we have to save our synthesized circuit in **VHDL format**.

The circuit with the largest number of components synthesized was S38584 which consists of about 8,000 components. In order to really test our tool's high performance computing capabilities we had to design some really large-scale circuit so we used also the b benchmarks.

3.3.9 Parallel Static Timing Analysis

The execution times we have achieved can be considered quite satisfactory (paragraph **4.4**). Although in high performance computing the need to analyze a circuit which consists of millions of gates is not rare. As a result a parallel execution of our algorithm would have extremely high interest. A first attempt in parallel execution of our method has been implemented.

We tried to solve this problem using a parallel hybridic model. In detail we modeified our code for execution in a cluster of PCs with multicore abilities on each PC's processor. The parallel programming libraries that have been used were **Message Passing Interface** for distributed communication and **POSIX threads** for multicore execution.

Assuming 10 PCs properly linked in a local network and a **QUAD core CPU** on each PC we separate the whole problem in 10*4 = 40 smaller problems. For example in the delays' computation procedure each core would have to analyze and compute the delays of only the 1/40 of the total delays. The code to be executed by each core would still be the same but with much fewer computations being demanded. Of course this had to be done always with respect to the levels of the gates, moving from a lower to a higher level. Also an appropriate synchronization and produced data fetching from each generated process has to be implemented.

Unfortunately there were to crucial factors that tougher our research:

1. We did not have a circuit consisting of millions of gates appropriate for our research. Although we created a circuit of approximately 400 000 components by starting from the largest circuit we had, which was about 50 000 components. We cloned this circuit and attached this clone next to the original by making the original's Primary Outputs, inputs for the cloned circuit's components. We repeated that procedure three times and created a circuit of about 400 000 components.

2. We did not have a cluster of PCs to test our code's performance. So we executed are code in an environment that simulated a virtual cluster of PCs.

Because of many parallel procedures that had to be implemented our whole parallel algorithm is still under construction but many positive summaries have been made. In conclusion the Parallel Static Timing Analysis is an area worth researching and will definitely be probed in the foreseeable future.

Chapter 4. Results Presentation

4.1 Critical Path

We are going to demonstrate our experiments' results for a specific VHDL circuit. The presentation will be based on the simplest circuit we have analyzed for practical reasons (e.g. limited duration and area of the present lecture). Of course the same methodology has been applied on the really large scale circuits' analysis where similar results have been taken.

We quote the VHDL description of s27.vhdl circuit from the **ISCAS Benchmark Circuits '89** and a layout designed by our experiments' verification tool.

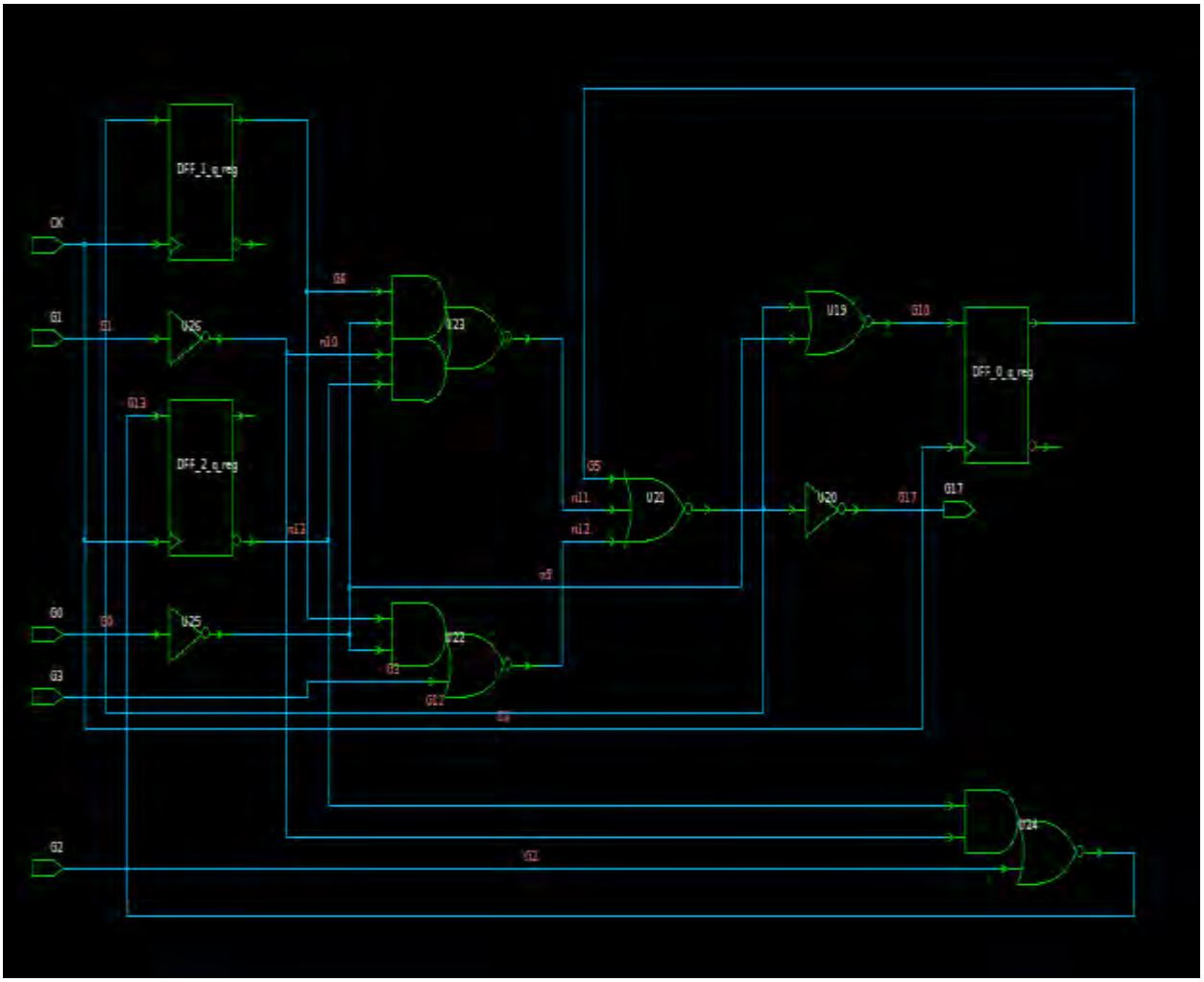

Figure 4.1 layout of s27.vhdl

```
library IEEE;

use IEEE.std_logic_1164.all;

entity s27 is

   port( CK, G0, G1, G2, G3 : in std_logic;  G17 : out std_logic);

end s27;

architecture SYN_verilog of s27 is

   component INV_X1
      port( A : in std_logic;  ZN : out std_logic);
   end component;

   component AOI21_X1
      port( B1, B2, A : in std_logic;  ZN : out std_logic);
   end component;

   component AOI22_X1
      port( A1, A2, B1, B2 : in std_logic;  ZN : out std_logic);
   end component;

   component NOR3_X1
      port( A1, A2, A3 : in std_logic;  ZN : out std_logic);
   end component;

   component NOR2_X1
      port( A1, A2 : in std_logic;  ZN : out std_logic);
   end component;

   component DFF_X1
      port( D, CK : in std_logic;  Q, QN : out std_logic);
   end component;

   signal G5, G10, G6, G11, G13, n9, n10, n11, n12, n13, n_1000, n_1001, n_1002
      : std_logic;

begin

   DFF_0_q_reg : DFF_X1 port map( D => G10, CK => CK, Q => G5, QN => n_1000);
   DFF_2_q_reg : DFF_X1 port map( D => G13, CK => CK, Q => n_1001, QN => n13);
   DFF_1_q_reg : DFF_X1 port map( D => G11, CK => CK, Q => G6, QN => n_1002);
   U19 : NOR2_X1 port map( A1 => G11, A2 => n9, ZN => G10);
   U20 : INV_X1 port map( A => G11, ZN => G17);
   U21 : NOR3_X1 port map( A1 => G5, A2 => n11, A3 => n12, ZN => G11);
   U22 : AOI21_X1 port map( B1 => G6, B2 => n9, A => G3, ZN => n12);
   U23 : AOI22_X1 port map( A1 => G6, A2 => n9, B1 => n10, B2 => n13, ZN => n11
                         );
   U24 : AOI21_X1 port map( B1 => n13, B2 => n10, A => G2, ZN => G13);
   U25 : INV_X1 port map( A => G0, ZN => n9);
   U26 : INV_X1 port map( A => G1, ZN => n10);

end SYN_verilog;
```

Figure 4.2 hardware language implementation of s27.vhdl

After having implemented appropriate data structures on the memory we determine the level of each component. The relevant data are being presented on **Table 4.1**.

component	type	level
DFF_0_q_reg	DFF_X1	4
DFF_2_q_reg	DFF_X1	2
DFF_1_q_reg	DFF_X1	3
U19	NOR2_X1	3
U20	INV_X1	3
U21	NOR3_X1	2
U22	AOI21_X1	1
U23	AOI22_X1	1
U24	AOI21_X1	1
U25	INV_X1	0
U26	INV_X1	0

Table 4.1 levelization of s27.vhdl

Afterwards, we computed the output capacitances of the components. It is worthy to mention here that the following results are based on a Nangate's fast version library. On the following table we can see the relevant results.

component	type	1st output's capacitance	2nd output's capacitance
DFF_0_q_reg	DFF_X1	0.001014	0.000000
DFF_2_q_reg	DFF_X1	0.000000	0.001909
DFF_1_q_reg	DFF_X1	0.001906	0.000000
U19	NOR2_X1	0.001202	0.000000
U20	INV_X1	0.000310	0.000000
U21	NOR3_X1	0.003102	0.000000
U22	AOI21_X1	0.000929	0.000000
U23	AOI22_X1	0.000977	0.000000
U24	AOI21_X1	0.001202	0.000000
U25	INV_X1	0.002845	0.000000
U26	INV_X1	0.001952	0.000000

Table 4.2 output capacitances of s27.vhdl components

On next step we calculate for each cell of each component their falling and rising delays as well as falling and rising transitions. **Table 4.3.1** demonstrates the maximum delay analysis type whereas **table 4.3.2** the minimum delay analysis type.

component	type	related pin	cell fall	cell rise	fall transition	rise transition	timing sense
DFF_0_q_reg	DFF_X1	CK	0.0981925	0.0472844	0.0109709	0.0105332	x
		CK	0.0363526	0.0635519	0.0071294	0.0084965	x
DFF_2_q_reg	DFF_X1	CK	0.0912411	0.0402463	0.0071453	0.0051523	x
		CK	0.0496623	0.0796773	0.0142703	0.0195819	x
DFF_1_q_reg	DFF_X1	CK	0.1035835	0.0530488	0.0140351	0.0159084	x
		CK	0.0363526	0.0635519	0.0071294	0.0084965	x
U19	NOR2_X1	A1	0.0197662	0.0292298	0.0166388	0.0193131	n
		A2	0.0209801	0.0291132	0.0132844	0.0180793	n
U20	INV_X1	A	0.0088710	0.0160919	0.0121508	0.0098294	n
U21	NOR3_X1	A1	0.0260024	0.0450677	0.0179624	0.0404980	n
		A2	0.0356030	0.0563398	0.0223852	0.0405963	n
		A3	0.0368945	0.0594153	0.0240135	0.0405539	n
U22	AOI21_X1	A	0.0179404	0.0236752	0.0131191	0.0161987	n
		A	0.0163588	0.0289999	0.0124682	0.0199436	n
		A	0.0166646	0.0344360	0.0153348	0.0233212	n
		B1	0.0183118	0.0267470	0.0120070	0.0199535	n
		B2	0.0201712	0.0314294	0.0118803	0.0232510	n
U23	AOI22_X1	A1	0.0184230	0.0224161	0.0121384	0.0153040	n
		A1	0.0184796	0.0264451	0.0121196	0.0202802	n
		A1	0.0187671	0.0317194	0.0148644	0.0236541	n
		A2	0.0202669	0.0257133	0.0120295	0.0175160	n
		A2	0.0203545	0.0310876	0.0120076	0.0236583	n
		A2	0.0206374	0.0364773	0.0147930	0.0270908	n
		B1	0.0258263	0.0294055	0.0163698	0.0170389	n
		B1	0.0238348	0.0348100	0.0158638	0.0206236	n
		B1	0.0241982	0.0403001	0.0191247	0.0239174	n
		B2	0.0278802	0.0352747	0.0164077	0.0195990	n

		B2	0.0258080	0.0417190	0.0158571	0.0239373	n
		B2	0.0261861	0.0472338	0.0191294	0.0272431	n
U24	AOI21_X1	A	0.0193294	0.0255154	0.0143604	0.0179035	n
		A	0.0177868	0.0313783	0.0137430	0.0221122	n
		A	0.0181043	0.0368408	0.0165939	0.0255235	n
		B1	0.0207898	0.0291562	0.0135649	0.0221099	n
		B2	0.0207668	0.0320357	0.0123593	0.0254984	n
U25	INV_X1	A	0.0172384	0.0235929	0.0134164	0.0212874	n
U26	INV_X1	A	0.0131764	0.0175763	0.0091140	0.0154247	n

Table 4.3.1 calculation of delays and transitions for each cell of the circuit (maximum values)

component	type	related pin	cell fall	cell rise	fall transition	rise transtiotion	timing sense
DFF_0_q_reg	DFF_X1	CK	0.0981925	0.0472844	0.0109709	0.0105332	x
		CK	0.0363526	0.0635519	0.0071294	0.0084965	x
DFF_2_q_reg	DFF_X1	CK	0.0912411	0.0402463	0.0071453	0.0051523	x
		CK	0.0496623	0.0796773	0.0142703	0.0195819	x
DFF_1_q_reg	DFF_X1	CK	0.1035835	0.0530488	0.0140351	0.0159084	x
		CK	0.0363526	0.0635519	0.0071294	0.0084965	x
U19	NOR2_X1	A1	0.0197639	0.0267758	0.0166169	0.0183248	n
		A2	0.0209801	0.0291132	0.0132844	0.0180793	n
U20	INV_X1	A	0.0088762	0.0143781	0.0121323	0.0086820	n
U21	NOR3_X1	A1	0.0260024	0.0450677	0.0179624	0.0404980	n
		A2	0.0316100	0.0544126	0.0208344	0.0405312	n
		A3	0.0344430	0.0585595	0.0234106	0.0405600	n
U22	AOI21_X1	A	0.0179404	0.0236752	0.0131191	0.0161987	n
		A	0.0163588	0.0289999	0.0124682	0.0199436	n
		A	0.0166646	0.0344360	0.0153348	0.0233212	n

		B1	0.0183118	0.0267470	0.0120070	0.0199535	n
		B2	0.0201712	0.0314294	0.0118803	0.0232510	n
U23	AOI22_X1	A1	0.0184230	0.0224161	0.0121384	0.0153040	n
		A1	0.0184796	0.0264451	0.0121196	0.0202802	n
		A1	0.0187671	0.0317194	0.0148644	0.0236541	n
		A2	0.0202669	0.0257133	0.0120295	0.0175160	n
		A2	0.0203545	0.0310876	0.0120076	0.0236583	n
		A2	0.0206374	0.0364773	0.0147930	0.0270908	n
		B1	0.0258263	0.0294055	0.0163698	0.0170389	n
		B1	0.0238348	0.0348100	0.0158638	0.0206236	n
		B1	0.0241982	0.0403001	0.0191247	0.0239174	n
		B2	0.0278802	0.0352747	0.0164077	0.0195990	n
		B2	0.0258080	0.0417190	0.0158571	0.0239373	n
		B2	0.0261861	0.0472338	0.0191294	0.0272431	n
U24	AOI21_X1	A	0.0193294	0.0255154	0.0143604	0.0179035	n
		A	0.0177868	0.0313783	0.0137430	0.0221122	n
		A	0.0181043	0.0368408	0.0165939	0.0255235	n
		B1	0.0207898	0.0291562	0.0135649	0.0221099	n
		B2	0.0207668	0.0320357	0.0123593	0.0254984	n
U25	INV_X1	A	0.0172384	0.0235929	0.0134164	0.0212874	n
U26	INV_X1	A	0.0131764	0.0175763	0.0091140	0.0154247	n

Table 4.3.2 calculation of delays and transitions for each cell of the circuit (minimum values)

Subsequently we have all the required data to compute each components' total delay. **Table 4.4.1** demonstrates the maximum total delay from a path ending to the named component whereas **Table 4.4.2** shows the equivalent minimum total delay.

component	type	total delay			
		1 st output		2nd output	
		falling	rising	falling	rising
DFF_0_q_reg	DFF_X1	0.098193	0.063552	0.036353	0.063552
DFF_2_q_reg	DFF_X1	0.091241	0.079677	0.049662	0.079677
DFF_1_q_reg	DFF_X1	0.103584	0.063552	0.036353	0.063552
U19	NOR2_X1	0.183664	0.200136	0.000000	0.000000
U20	INV_X1	0.172768	0.186998	0.000000	0.000000
U21	NOR3_X1	0.170906	0.163897	0.000000	0.000000
U22	AOI21_X1	0.081864	0.130331	0.000000	0.000000
U23	AOI22_X1	0.107557	0.135303	0.000000	0.000000
U24	AOI21_X1	0.100467	0.078819	0.000000	0.000000
U25	INV_X1	0.017238	0.023593	0.000000	0.000000
U26	INV_X1	0.013176	0.017576	0.000000	0.000000

Table 4.4.1 calculation of total delays for each component of the circuit (maximum values)

component	type	total delay			
		1st output		2nd output	
		falling	rising	falling	rising
DFF_0_q_reg	DFF_X1	0.036353	0.047284	0.036353	0.063552
DFF_2_q_reg	DFF_X1	0.049662	0.040246	0.049662	0.079677
DFF_1_q_reg	DFF_X1	0.036353	0.053049	0.036353	0.063552
U19	NOR2_X1	0.044573	0.046352	0.000000	0.000000
U20	INV_X1	0.083795	0.072496	0.000000	0.000000
U21	NOR3_X1	0.058118	0.074918	0.000000	0.000000
U22	AOI21_X1	0.016359	0.023675	0.000000	0.000000
U23	AOI22_X1	0.041411	0.042582	0.000000	0.000000
U24	AOI21_X1	0.017787	0.025515	0.000000	0.000000
U25	INV_X1	0.017238	0.023593	0.000000	0.000000
U26	INV_X1	0.013176	0.017576	0.000000	0.000000

Table 4.4.2 calculation of total delays for each component of the circuit (minimum values)

Finally we could execute our critical path find algorithm and compute the maximum and the minimum version of the critical path of the circuit. **Tables 4.5.1** and **4.5.2** demonstrate the previous respectively.

gate	type	increment	total_delay	gate_level	polar
U20	INV_X1	0.01609190	0.18699779	3	r
U21	NOR3_X1	0.03560297	0.17090589	2	f
U23	AOI22_X1	0.03171939	0.13530292	1	r
DFF_1_q_reg	DFF_X1	0.10358353	0.10358353	0	f

Table 4.5.1 maximum critical path

gate	type	increment	total_delay	gate_level	polar
U20	INV_X1	0.01437810	0.07249625	3	r
U21	NOR3_X1	0.03444300	0.05811815	2	f
U22	AOI21_X1	0.02367515	0.02367515	1	r

Table 4.5.2 minimum critical path

On the following we quite a schematic repsrentation of the critical paths.

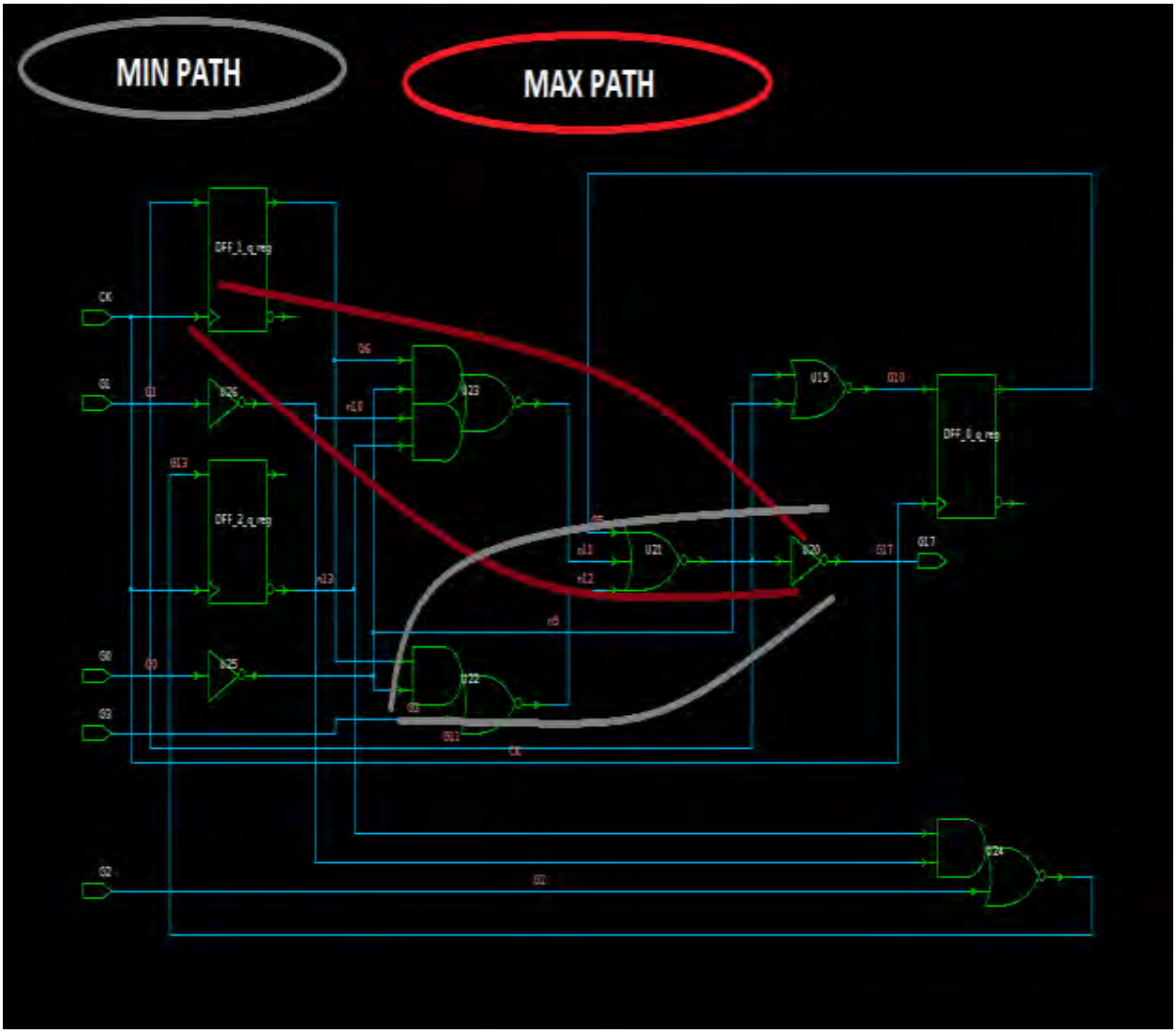

Figure 4.3 demonstration of the critical paths on the layout of the circuit s27

4.2 Path Categories

Previously we have described the methodology for analyzing the different types of paths on a circuit and check for violations. This section demonstrates the results taken by our tool's analysis on the s27 circuit. The paths are being presented in 4 sections with respect to section 3.3.6 for each delay analysis type (max or min). We have assumed a clock of frequency 0.2ns and it is because of the clock's frequency violation the behavior of the last 3 paths on register to register path family.

1. **Primary Input to Primary Output Paths**

G0			
INV_X1	U25	r	0.0235929
AOI21_X1	U22	f	0.0201712
NOR3_X1	U21	r	0.0594153
INV_X1	U20	f	0.0088710
G0			
INV_X1	U25	r	0.0235929
AOI22_X1	U23	f	0.0206374
NOR3_X1	U21	r	0.0563398
INV_X1	U20	f	0.0088710
G1			
INV_X1	U26	r	0.0175763
AOI22_X1	U23	f	0.0258263
NOR3_X1	U21	r	0.0563398
INV_X1	U20	f	0.0088710
G0			
INV_X1	U25	f	0.0172384
AOI22_X1	U23	r	0.0364773
NOR3_X1	U21	f	0.0356030
INV_X1	U20	r	0.0160919

G1			
INV_X1	U26	f	0.0131764
AOI22_X1	U23	r	0.0403001
NOR3_X1	U21	f	0.0356030
INV_X1	U20	r	0.0160919
G0			
INV_X1	U25	f	0.0172384
AOI21_X1	U22	r	0.0314294
NOR3_X1	U21	f	0.0368945
INV_X1	U20	r	0.0160919
G3			
AOI21_X1	U22	r	0.0344360
NOR3_X1	U21	f	0.0368945
INV_X1	U20	r	0.0160919
G3			
AOI21_X1	U22	f	0.0179404
NOR3_X1	U21	r	0.0594153
INV_X1	U20	f	0.0088710

Table 4.6.1 Primary Input to Primary Output Paths in s27

2. Register to Primary Output Paths

CLK			
DFF_X1	DFF_1_q_reg	f	0.1035835
AOI22_X1	U23	r	0.0317194
NOR3_X1	U21	f	0.0356030
INV_X1	U20	r	0.0160919
CLK			
DFF_X1	DFF_1_q_reg	f	0.1035835
AOI21_X1	U22	r	0.0267470
NOR3_X1	U21	f	0.0368945
INV_X1	U20	r	0.0160919
CLK			
DFF_X1	DFF_2_q_reg	r	0.0796773
AOI22_X1	U23	f	0.0278802
NOR3_X1	U21	r	0.0563398
INV_X1	U20	f	0.0088710
CLK			
DFF_X1	DFF_0_q_reg	f	0.0981925
NOR3_X1	U21	r	0.0450677
INV_X1	U20	f	0.0088710

CLK			
DFF_X1	DFF_2_q_reg	f	0.0496623
AOI22_X1	U23	r	0.0472338
NOR3_X1	U21	f	0.0356030
INV_X1	U20	r	0.0160919
CLK			
DFF_X1	DFF_1_q_reg	r	0.0530488
AOI21_X1	U22	f	0.0183118
NOR3_X1	U21	r	0.0594153
INV_X1	U20	f	0.0088710
CLK			
DFF_X1	DFF_1_q_reg	r	0.0530488
AOI22_X1	U23	f	0.0187671
NOR3_X1	U21	r	0.0563398
INV_X1	U20	f	0.0088710
CLK			
DFF_X1	DFF_0_q_reg	r	0.0472844
NOR3_X1	U21	f	0.0260024
INV_X1	U20	r	0.0160919

Table 4.6.2 Register to Primary Output Paths in s27

3. Primary Input to Register Paths

G0			
INV_X1	U25	r	0.0235929
AOI21_X1	U22	f	0.0201712
NOR3_X1	U21	r	0.0594153
NOR2_X1	U19	f	0.0197662
DFF_X1	DFF_0_q_reg	f	0.0000000
G0			
INV_X1	U25	r	0.0235929
AOI22_X1	U23	f	0.0206374
NOR3_X1	U21	r	0.0563398
NOR2_X1	U19	f	0.0197662
DFF_X1	DFF_0_q_reg	f	0.0000000
G1			
INV_X1	U26	r	0.0175763
AOI22_X1	U23	f	0.0258263
NOR3_X1	U21	r	0.0563398
NOR2_X1	U19	f	0.0197662
DFF_X1	DFF_0_q_reg	f	0.0000000
G0			
INV_X1	U25	f	0.0172384
AOI22_X1	U23	r	0.0364773
NOR3_X1	U21	f	0.0356030
NOR2_X1	U19	r	0.0292298
DFF_X1	DFF_0_q_reg	r	0.0000000
G1			
INV_X1	U26	f	0.0131764
AOI22_X1	U23	r	0.0403001
NOR3_X1	U21	f	0.0356030
NOR2_X1	U19	r	0.0292298
DFF_X1	DFF_0_q_reg	r	0.0000000
G0			
INV_X1	U25	f	0.0172384
AOI21_X1	U22	r	0.0314294
NOR3_X1	U21	f	0.0368945
NOR2_X1	U19	r	0.0292298
DFF_X1	DFF_0_q_reg	r	0.0000000
G0			
INV_X1	U25	r	0.0235929
AOI21_X1	U22	f	0.0201712
NOR3_X1	U21	r	0.0594153
DFF_X1	DFF_1_q_reg	r	0.0000000

G0			
INV_X1	U25	r	0.0235929
AOI22_X1	U23	f	0.0206374
NOR3_X1	U21	r	0.0563398
DFF_X1	DFF_1_q_reg	r	0.0000000
G3			
AOI21_X1	U22	r	0.0344360
NOR3_X1	U21	f	0.0368945
NOR2_X1	U19	r	0.0292298
DFF_X1	DFF_0_q_reg	r	0.0000000
G1			
INV_X1	U26	r	0.0175763
AOI22_X1	U23	f	0.0258263
NOR3_X1	U21	r	0.0563398
DFF_X1	DFF_1_q_reg	r	0.0000000
G3			
AOI21_X1	U22	f	0.0179404
NOR3_X1	U21	r	0.0594153
NOR2_X1	U19	f	0.0197662
DFF_X1	DFF_0_q_reg	f	0.0000000
G0			
INV_X1	U25	f	0.0172384
AOI22_X1	U23	r	0.0364773
NOR3_X1	U21	f	0.0356030
DFF_X1	DFF_1_q_reg	f	0.0000000
G1			
INV_X1	U26	f	0.0131764
AOI22_X1	U23	r	0.0403001
NOR3_X1	U21	f	0.0356030
DFF_X1	DFF_1_q_reg	f	0.0000000
G0			
INV_X1	U25	f	0.0172384
AOI21_X1	U22	r	0.0314294
NOR3_X1	U21	f	0.0368945
DFF_X1	DFF_1_q_reg	f	0.0000000

G3			
AOI21_X1	U22	f	0.0179404
NOR3_X1	U21	r	0.0594153
DFF_X1	DFF_1_q_reg	r	0.0000000
G3			
AOI21_X1	U22	r	0.0344360
NOR3_X1	U21	f	0.0368945
DFF_X1	DFF_1_q_reg	f	0.0000000
G0			
INV_X1	U25	f	0.0172384
NOR2_X1	U19	r	0.0291132
DFF_X1	DFF_0_q_reg	r	0.0000000
G1			
INV_X1	U26	f	0.0131764
AOI21_X1	U24	r	0.0320357
DFF_X1	DFF_2_q_reg	r	0.0000000
G0			
INV_X1	U25	r	0.0235929
NOR2_X1	U19	f	0.0209801
DFF_X1	DFF_0_q_reg	f	0.0000000
G1			
INV_X1	U26	r	0.0175763
AOI21_X1	U24	f	0.0207668
DFF_X1	DFF_2_q_reg	f	0.0000000
G2			
AOI21_X1	U24	r	0.0368408
DFF_X1	DFF_2_q_reg	r	0.0000000
G2			
AOI21_X1	U24	f	0.0193294
DFF_X1	DFF_2_q_reg	f	0.0000000

Table 4.6.3 Primary Input to Register Paths in s27

4. Register to Register Paths

CLK			
DFF_X1	DFF_0_q_reg	r	0.0472844
NOR3_X1	U21	f	0.0260024
DFF_X1	DFF_1_q_reg	f	0.0000000
clock CK (rise edge)			0.2000000
DFF_1_q_reg / CK			0.2000000
library setup time			-0.0216833
data required time			0.1783167
data arrival time			0.0732868
slack (MET)			0.1050299
CLK			
DFF_X1	DFF_2_q_reg	f	0.0496623
AOI21_X1	U24	r	0.0291562
DFF_X1	DFF_2_q_reg	r	0.0000000
clock CK (rise edge)			0.2000000
DFF_2_q_reg / CK			0.2000000
library setup time			-0.0293465
data required time			0.1706535
data arrival time			0.0788185
slack (MET)			0.0918350
CLK			
DFF_X1	DFF_2_q_reg	r	0.0796773
AOI21_X1	U24	f	0.0207898
DFF_X1	DFF_2_q_reg	f	0.0000000
clock CK (rise edge)			0.2000000
DFF_2_q_reg / CK			0.2000000
library setup time			-0.0194163
data required time			0.1805837
data arrival time			0.1004671

slack (MET)			0.0801166
CLK			
DFF_X1	DFF_0_q_reg	r	0.0472844
NOR3_X1	U21	f	0.0260024
NOR2_X1	U19	r	0.0292298
DFF_X1	DFF_0_q_reg	r	0.0000000
clock CK (rise edge)			0.2000000
DFF_0_q_reg / CK			0.2000000
library setup time			-0.0281015
data required time			0.1718985
data arrival time			0.1025166
slack (MET)			0.0693819
CLK			
DFF_X1	DFF_2_q_reg	f	0.0496623
AOI22_X1	U23	r	0.0472338
NOR3_X1	U21	f	0.0356030
DFF_X1	DFF_1_q_reg	f	0.0000000
clock CK (rise edge)			0.2000000
DFF_1_q_reg / CK			0.2000000
library setup time			-0.0216833
data required time			0.1783167
data arrival time			0.1324991
slack (MET)			0.0458176
CLK			
DFF_X1	DFF_1_q_reg	r	0.0530488
AOI22_X1	U23	f	0.0187671
NOR3_X1	U21	r	0.0563398
DFF_X1	DFF_1_q_reg	r	0.0000000

clock CK (rise edge)			0.2000000
DFF_1_q_reg / CK			0.2000000
library setup time			-0.0322240
data required time			0.1677760
data arrival time			0.1281557
slack (MET)			0.0396203
CLK			
DFF_X1	DFF_1_q_reg	r	0.0530488
AOI21_X1	U22	f	0.0183118
NOR3_X1	U21	r	0.0594153
DFF_X1	DFF_1_q_reg	r	0.0000000
clock CK (rise edge)			0.2000000
DFF_1_q_reg / CK			0.2000000
library setup time			-0.0322240
data required time			0.1677760
data arrival time			0.1307760
slack (MET)			0.0370000
CLK			
DFF_X1	DFF_1_q_reg	r	0.0530488
AOI22_X1	U23	f	0.0187671
NOR3_X1	U21	r	0.0563398
NOR2_X1	U19	f	0.0197662
DFF_X1	DFF_0_q_reg	f	0.0000000
clock CK (rise edge)			0.2000000
DFF_0_q_reg / CK			0.2000000
library setup time			-0.0194289
data required time			0.1805711
data arrival time			0.1479219

slack (MET)			0.0326491
CLK			
DFF_X1	DFF_1_q_reg	r	0.0530488
AOI21_X1	U22	f	0.0183118
NOR3_X1	U21	r	0.0594153
NOR2_X1	U19	f	0.0197662
DFF_X1	DFF_0_q_reg	f	0.0000000
clock CK (rise edge)			0.2000000
DFF_0_q_reg / CK			0.2000000
library setup time			-0.0194289
data required time			0.1805711
data arrival time			0.1505422
slack (MET)			0.0300288
CLK			
DFF_X1	DFF_0_q_reg	f	0.0981925
NOR3_X1	U21	r	0.0450677
DFF_X1	DFF_1_q_reg	r	0.0000000
clock CK (rise edge)			0.2000000
DFF_1_q_reg / CK			0.2000000
library setup time			-0.0322240
data required time			0.1677760
data arrival time			0.1432602
slack (MET)			0.0245157
CLK			
DFF_X1	DFF_0_q_reg	f	0.0981925
NOR3_X1	U21	r	0.0450677
NOR2_X1	U19	f	0.0197662
DFF_X1	DFF_0_q_reg	f	0.0000000

clock CK (rise edge)			0.2000000
DFF_0_q_reg / CK			0.2000000
library setup time			-0.0194289
data required time			0.1805711
data arrival time			0.1630265
slack (MET)			0.0175446
CLK			
DFF_X1	DFF_1_q_reg	f	0.1035835
AOI21_X1	U22	r	0.0267470
NOR3_X1	U21	f	0.0368945
DFF_X1	DFF_1_q_reg	f	0.0000000
clock CK (rise edge)			0.2000000
DFF_1_q_reg / CK			0.2000000
library setup time			-0.0216833
data required time			0.1783167
data arrival time			0.1672250
slack (MET)			0.0110917
CLK			
DFF_X1	DFF_2_q_reg	f	0.0496623
AOI22_X1	U23	r	0.0472338
NOR3_X1	U21	f	0.0356030
NOR2_X1	U19	r	0.0292298
DFF_X1	DFF_0_q_reg	r	0.0000000
clock CK (rise edge)			0.2000000
DFF_0_q_reg / CK			0.2000000
library setup time			-0.0281015
data required time			0.1718985
data arrival time			0.1617289
slack (MET)			0.0101696

CLK			
DFF_X1	DFF_1_q_reg	f	0.1035835
AOI22_X1	U23	r	0.0317194
NOR3_X1	U21	f	0.0356030
DFF_X1	DFF_1_q_reg	f	0.0000000
clock CK (rise edge)			0.2000000
DFF_1_q_reg / CK			0.2000000
library setup time			-0.0216833
data required time			0.1783167
data arrival time			0.1709059
slack (MET)			0.0074108
CLK			
DFF_X1	DFF_2_q_reg	r	0.0796773
AOI22_X1	U23	f	0.0278802
NOR3_X1	U21	r	0.0563398
DFF_X1	DFF_1_q_reg	r	0.0000000
clock CK (rise edge)			0.2000000
DFF_1_q_reg / CK			0.2000000
library setup time			-0.0322240
data required time			0.1677760
data arrival time			0.1638973
slack (MET)			0.0038787
CLK			
DFF_X1	DFF_2_q_reg	r	0.0796773
AOI22_X1	U23	f	0.0278802
NOR3_X1	U21	r	0.0563398
NOR2_X1	U19	f	0.0197662
DFF_X1	DFF_0_q_reg	f	0.0000000

clock CK (rise edge)			0.2000000
DFF_0_q_reg / CK			0.2000000
library setup time			-0.0194289
data required time			0.1805711
data arrival time			0.1836635
slack (VIOLATED)			-0.0030924
CLK			
DFF_X1	DFF_1_q_reg	f	0.1035835
AOI21_X1	U22	r	0.0267470
NOR3_X1	U21	f	0.0368945
NOR2_X1	U19	r	0.0292298
DFF_X1	DFF_0_q_reg	r	0.0000000
clock CK (rise edge)			0.2000000
DFF_0_q_reg / CK			0.2000000
library setup time			-0.0281015
data required time			0.1718985
data arrival time			0.1964548
slack (VIOLATED)			-0.0245563
CLK			
DFF_X1	DFF_1_q_reg	f	0.1035835
AOI22_X1	U23	r	0.0317194
NOR3_X1	U21	f	0.0356030
NOR2_X1	U19	r	0.0292298
DFF_X1	DFF_0_q_reg	r	0.0000000
clock CK (rise edge)			0.2000000
DFF_0_q_reg / CK			0.2000000
library setup time			-0.0281015
data required time			0.1718985
data arrival time			0.2001357
slack (VIOLATED)			-0.0282372

Table 4.6.4 Register to Register Paths in s27

4.3 Aggregate Results of Test Circuits

We are going to present the results of the total experiments we came across on the ISCAS circuits' family. **Table 4.7.1** demonstrates the maximum and the minimum critical path's delay that our tool has calculated. We can also observe the equivalent values that the industrial EDA tool that we have use for the result's verification has computed. Finally a comparison between the two different sets of results has been made and an accuracy rate of out tool's result is being presented.

Circuit	#Inputs	#Outputs	#Gates	MITTAS TOOL			INDUSTRIAL TOOL		accuracy rate	
				max path	min path	CPU runtime (sec)	max path	min path	max path	min path
s27	5	1	11	0.186998	0.072496	0.210	0.187019	0.072516	99.99	99.97
s298	4	6	70	0.106506	0.036353	0.202	0.107284	0.049016	99.27	74.17
s349	12	11	81	0.238822	0.064014	0.204	0.240188	0.064679	99.43	98.97
s382	6	6	104	0.111801	0.037188	0.201	0.111807	0.044187	99.99	84.16
s386	10	7	72	0.350483	0.008365	0.202	0.350525	0.008373	99.99	99.90
s400	6	6	106	0.108168	0.036353	0.199	0.108175	0.042442	99.99	85.65
s420	21	1	83	0.774004	0.022863	0.207	0.774125	0.022877	99.98	99.94
s444	6	6	103	0.108168	0.036353	0.202	0.108175	0.042442	99.99	85.65
s510	22	7	134	0.481774	0.084612	0.203	0.479222	0.143609	99.47	58.92
s526	6	6	122	0.106017	0.036353	0.200	0.106795	0.048971	99.27	74.23
s641	38	24	112	0.748394	0.000000	0.201	0.751182	0.000000	99.63	100.00
s713	38	23	112	0.748394	0.000000	0.222	0.751182	0.000000	99.63	100.00
s820	21	19	162	0.460932	0.008365	0.206	0.460982	0.008374	99.99	99.89
s832	21	19	163	0.498944	0.008365	0.202	0.498995	0.008374	99.99	99.89
s1196	17	14	298	0.584170	0.010211	0.211	0.584228	0.010221	99.99	99.90
s1238	17	14	297	0.759229	0.012890	0.211	0.759345	0.012900	99.98	99.92
s1423	20	5	366	1.706443	0.017294	0.210	1.697314	0.017929	99.47	96.46
s1488	11	19	320	0.742423	0.016782	0.211	0.742448	0.016793	100.00	99.93
s5378	39	49	816	0.560624	0.000000	0.232	0.563911	0.000003	99.42	100.00
s15850	81	150	2075	1.060990	0.000000	0.295	1.061151	0.000002	99.98	100.00
s35932	38	320	5857	0.302718	0.036353	0.458	0.300790	0.029995	99.36	82.51
s38584	45	304	7793	0.939377	0.000000	0.609	0.934391	0.000002	99.47	100.00

Table 4.7 ISCAS Benchmark Circuits '89 Timing Analysis results

By the previous table we can estimate that our EDA tool's **average accuracy** is **99.74%** in calculation of maximum critical path and **92.73%** in the equivalent minimum procedure.

4.4 Execution times

4.4.1 Execution times

We will quote the execution times for the two previously mentioned circuits' families. To note here that our tool's implementation, and the execution of the code for gathering the results has been exclusively taken place in an **Intel Core 2 6400** system equipped with **1GB RAM.** Obviously this in not an obsolete system but it is for sure that cannot be characterized high-tech. Consequently our EDA tool can boast of even better execution times in an up to date system.

Circuit	#Inputs	#Outputs	#Gates	CPU runtime (sec)
s27	5	1	11	0.210
s298	4	6	70	0.202
s349	12	11	81	0.204
s382	6	6	104	0.201
s386	10	7	72	0.202
s400	6	6	106	0.199
s420	21	1	83	0.207
s444	6	6	103	0.202
s510	22	7	134	0.203
s526	6	6	122	0.200
s641	38	24	112	0.201
s713	38	23	112	0.222
s820	21	19	162	0.206
s832	21	19	163	0.202
s1196	17	14	298	0.211
s1238	17	14	297	0.211
s1423	20	5	366	0.210
s1488	11	19	320	0.211
s5378	39	49	816	0.232
s15850	81	150	2075	0.295
s35932	38	320	5857	0.458
s38584	45	304	7793	0.609

Table 4.8 execution times of ISCAS Benchmark Circuits '89

Circuit	#Inputs	#Outputs	#Gates	CPU runtime (sec)
b01	4	2	30	0.201
b02	3	1	16	0.196
b03	6	4	78	0.200
b05	3	36	320	0.211
b14	34	54	3824	0.437
b18	38	23	24452	2.327
b19	47	30	46964	3.762

Table 4.6.2 execution times of b circuits

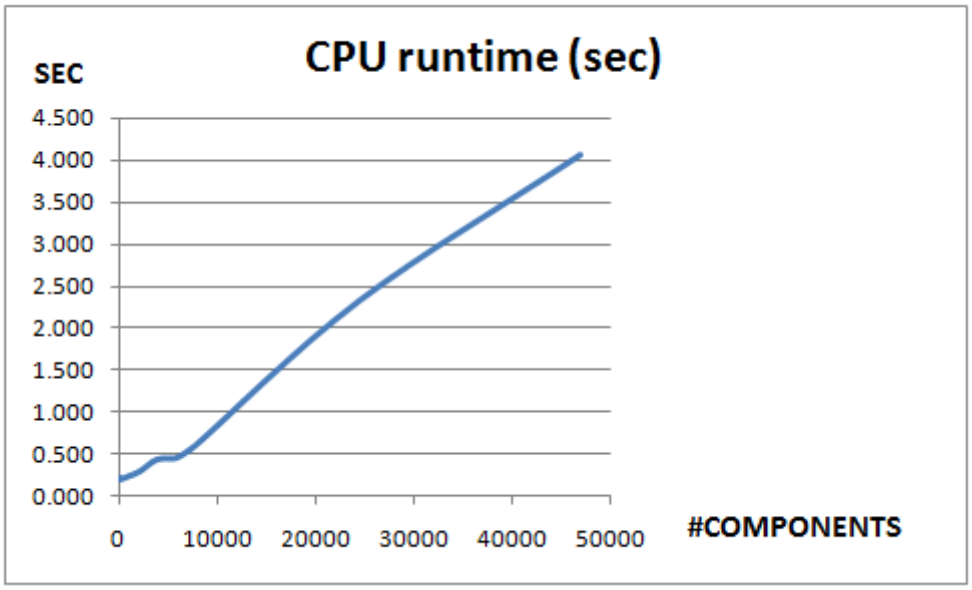

Figure 4.4 execution times of timings analysis with respect to the number of gates

For the execution time information we regularly compile our code and append on the beginning of the execution command the **LINUX *time*** command.

4.4.2 Code Profile

In order to understand our code's weaknesses we profiled our code with Intel VTune Performance Amplifier. We have used the aforementioned profiler several times during our tool's implementation for spotting bottlenecks in our code. We wrote a different version of the same algorithm and reprofiling the new code and so on. On this section we will present the results of b19.vhdl hotspots analysis as it is the most demanding netlist we had to test.

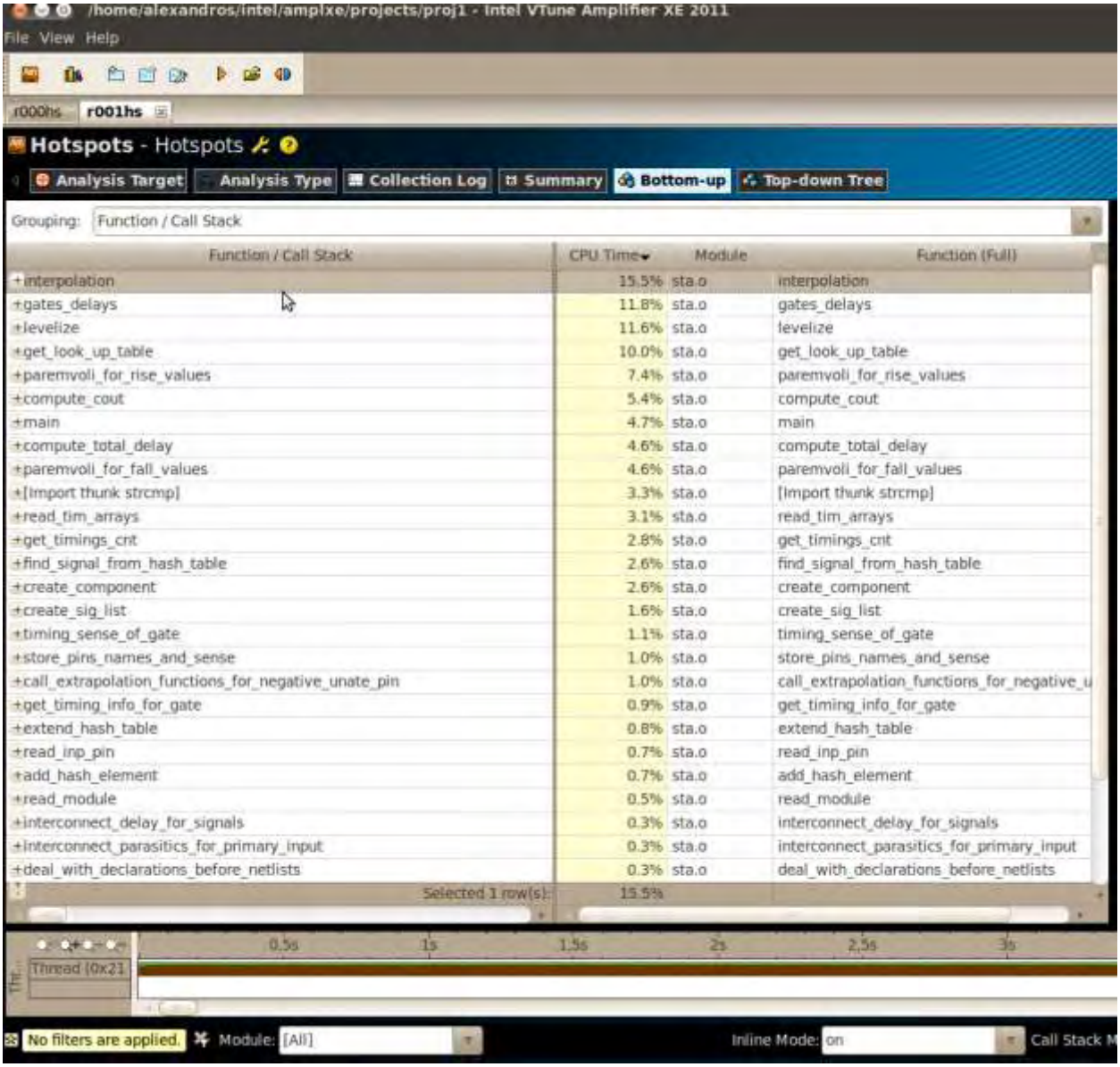

Figure 4.5 Hotspots Analysis of the code with Intel VTune Performance Analyzer

We can see that the execution time of our application is finely balanced on its functions. The interpolation, the delay calculation and the levelization functions consume a considerable fraction of the CPU although these functions' effects were not detrimental to our application's performance. To sum up we can conclude that the CPU has been used quite wisely and the occupation of the CPU by these procedures is not unjustified. What is more to note here that library parsing functions, although reading a quite large text file and storing its data on the memory they had a little, almost inconsiderable, effect on the tool's performance. This is quite expected, if we consider the amount of calculations that have been made by the other processes.

The only demand of the code profile is to compile our application with **flag** ***–ggdb.***

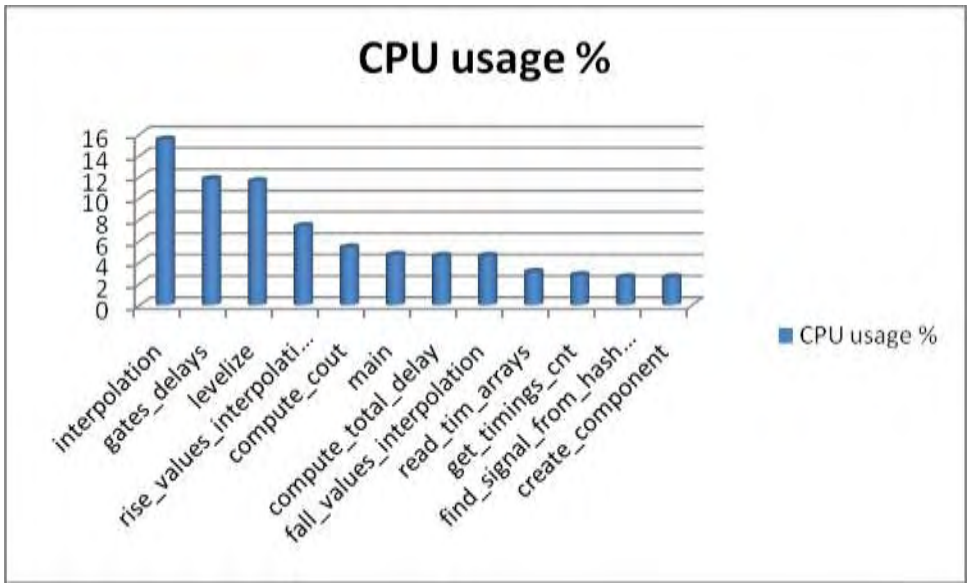

Figure 4.6 CPU usage by the tool's Procedures

Chapter 5. Conclusion

5.1 Summary

The software that has been developed is complete effort on the field of Timing Analysis of Integrated Circuits. This EDA tool has the potential to analyze each digital integrated circuit's structural description and verify its timing behavior. After the analysis conclusions can be made and circuits' validation methods can be continued properly by examining other aspects of verification in Integrated Circuits.

5.2 Future Directions - Optimizations

This project can be combined with similar EDA tools used for verification purposes of Integrated Circuits, as timing is not an absolute factor to verify a circuit's behavior. Consequently the current project can be combined with

- Power and Noise analysis tools
- Place and Root tools
- Clock Domain Crossing Verification tools
- Temperature analysis or local voltage variations

The current project can also be expanded so as to perform

- Statistical Static Timing analysis. The aforementioned method replaces the normal deterministic timing of gates and interconnects with probability distributions, and gives a distribution of possible circuit outcomes rather than a single outcome.
- Parallel Execution so as to accelerate the execution times in VLSI.
- Graphics interface for schematic representation of the circuit with respect to its placement using a library such as OpenGL.

Bibliography

- **http://en.wikipedia.org/wiki/Static_timing_analysis**
- **http://en.wikipedia.org/wiki/Statistical_static_timing_analysis**
- http://en.wikipedia.org/wiki/Electronic_design_automation
- Static Timing Analysis for Nanometer Designs. A Practical Approach (J. Bhasker, Rakesh Chadha)
- CMOS VLSI Design (Neil Waste, David Harris)
- http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html#SEC1
- http://linux.about.com/library/cmd/blcmdl1_time.htm
- http://en.wikipedia.org/wiki/Static_timing_analysis
- http://wiki.answers.com/Q/Difference_between_the_dynamic_timing_analysis_and_static_timing_analysis
- http://asic-soc.blogspot.com/2008/08/dynamic-vs-static-timing-analysis.html
- http://asic-soc.blogspot.com/2009/06/timing-paths.html
- http://www.nangate.com/
- http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html
- http://en.wikipedia.org/wiki/Time_%28Unix%29
- http://wiki.answers.com/Q/What_is_the_advantages_of_c_language
- http://en.kioskea.net/forum/affich-21243-advantages-and-disadvantages-of-c-language