Logic Restructuring for Delay Balancing in Wave-Pipelined Circuits: an Integer Programming Approach

Srivastav Sethupathy, Nohpill Park
Computer Science Department
OSU
Stillwater, OK, USA
npark@cs.okstate.edu

Marcin Paprzycki (contact author)
Computer Science Institute
SWPS
Warsaw, Poland
marcin.paprzycki@swps.edu.pl

Abstract

In this paper we apply integer programming (IP) based techniques to the problem of delay balancing in wave-pipelined circuits. The proposed approach considers delays, as well as fan-in and fan-out associated with every node in the circuit. After a weighted graph representation of the circuit is formed a node collapsing procedure is used to preprocess (reduce the size of) the system and obtain the final formulation of the IP problem, which is solved by using a branch and bound heuristic to obtain a minimum delay in the circuit. We also compare the proposed technique with application – to the same problem – of a linear programming solver.

1. Introduction

Wave-pipelining is one of widely used methods for designing high throughput VLSI circuits and its goal is to maximize utilization of the combinational logic without use of intermediate latches and registers [4]. Two common assumptions made in wave-pipelining are: (1) the total length of the logic path in a circuit is "long enough" to cause delay variations, and (2) the total data dispersion is small i.e. locality of data or logic signals in a circuit is high [13]. These assumptions make it possible for the system to have several *waves* (logic signals) that propagate simultaneously (locality) across the circuitry (logic path is long), thus allowing for a higher throughput. Some of basic concepts involved in wave-pipelined systems are [8]:

- *clock skew* – represents arrival of a signal at different times at the clock inputs of different flip-flops due to the propagation delay denoted as t_{prop}
- *speedup* – signifies improvement in the circuit performance resulting from the fact that ***N*** data waves propagate simultaneously through the circuit (note that in standard pipelines, the speedup is associated with number of pipeline stages).
- *timing constraints* – denote constrains imposed on the circuit by the timing factors and are represented by:

$$T_{clk} > T_{diff} + \textit{clocking overhead}$$

where T_{diff} is the difference between the maximum and minimum delays ($T_{diff} = D_{max} - D_{min}$), while the clocking overhead is $T_{setup} + T_{hold} + (2 * \textit{clock skew})$, where T_{setup} is the setup time and T_{hold} is the hold time.

When data is fed into the wave-pipelined circuit, the main goal of system design is to effectively pipeline the flow of data waves without the use of intermediate stoppages such as latches and registers to achieve the maximum throughput (rate of pipelining) [14]. The key idea here is to allow multiple waves of data to propagate through the logic circuit, which is designed so that data waves do not overlap. In this context, the two most desirable features of wave-pipelined systems are [16]:

- most logic paths have the same depth,
- fan-in/fan-out of the elements are the same.

To achieve this goal, logic restructuring for delay balancing is applied.

It should be noted, that due to the following factors, the design of wave-pipelined systems is quite complex and challenging:

- it depends strongly on nature of the circuit, i.e. the logic family to which a given circuit belong to,
- it is vulnerable to the *process, volume and temperature* (PVT) variations [13] – in other words, changes in the operating environment have a direct effect on the design of the system as they affect the delay in wave-pipelined circuits, and
- both maximum delay D_{max} and minimum delay D_{min} have to be considered in designing wave-pipelined circuits – logic waves can be neither too fast nor too slow (while in conventional pipeline circuits, only the maximum delay has to be considered).

The aim of our work is to investigate how integer programming based techniques can be applied to logic restructuring in wave-pipelined circuits. We proceed as follows. In the next section we briefly summarized related

work. We follow by a description of integer programming techniques arising in the context of logic restructuring and the branch and bound heuristics. In Section 5 we present results of our experimental work and follow with description of possible future research directions.

2. Related Work

In design of wave-pipelined systems, one of fundamental approaches used in logic restructuring is *node collapsing* [13]. This process results in balanced circuit configuration and helps further conditioning of the circuit to confirm to the minimum timing requirements [5, 6]. Furthermore, it can be applied as a pre-processing stage of solving the logic restructuring problem.

Thus far, the problem of logic restructuring in wave-pipelined circuits has been solved mostly by utilizing linear programming (LP) techniques [2, 3]. Here, the ultimate goal was to minimize the difference in path delays of the circuit (to achieve maximum rate pipelining with minimal number of circuit nodes [4]). LP techniques used to address problem of delay balancing in have been applied at different levels [7, 17, 18]:

- By Berkelaar et al at the *transistor level [24]*.
- By Agrawal et al at the *gate level [25]*.

Both these techniques are essentially the same and try to minimize the overall power dissipation of the circuit by reducing the gate load capacitance**.**

Finally, it was suggested that integer programming (IP) can be applied to logic restructuring in wave-pipelined circuits. Such suggestion has been made in [1], which an overview of how IP can be applied to various resource allocation problems. However, beyond a suggestion made, there was not an instance of IP being actually applied to logic restructuring.

3. Integer Programming Approach

Basic steps involved in solving the problem of logic restructuring in wave-pipelined circuits are:

- initial formulation of the IP problem,
- optimizing the circuit using the *node collapsing*
- solving the IP problem using branch and cut algorithm**,**
- quantitative analysis of the results,

where the second step, while highly desirable, can be omitted if only a pure IP problem is to be solved. The role of node collapsing is to reduce the total number of nodes (simplify the circuit design) and improve its overall balance).

An integer programming problem is an optimization problem of the form:

$$Min \text{ (or } Max) \; cx$$

where, $x \in X$; $X = Z^n \cap \{x \in R^n \mid Ax \leq b\}$, Z^n denotes the set of all integers, R^n represents the set of all real n-vectors [10]. Finally, $Ax \leq b$ is referred to as the formulation of the *Set X*. An important factor involved in prictivcal use of integer programming is a correct – from the perspective of the problem to be solved – formulation of the particular IP problem (as an integer programming problem can have several formulations). In the case of the logic restructuring we apply a formulation based on weighted graph representation of the circuit [1]. Here, the weighted graph representation has the form $G(d, f_{in}, f_{out})$, where, d is the delay (weight of the path), f_{in} is the fan-in of the node and f_{out} represents fan-out of the node. In Figure 1 we depict a simple one-bit adder (top panel) and its weighted graph representation. Here, each node has fan-in value 2 and fan-out value 1.

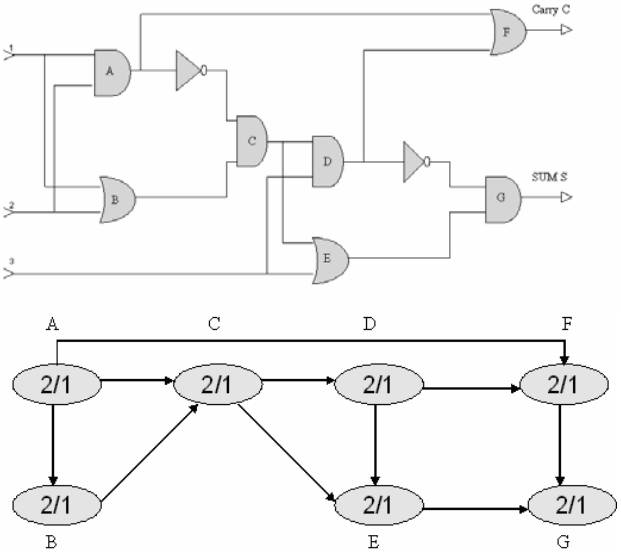

Figure 1 Sample one-bit adder and its weighted-graph representation; first digit – fan-in, second digit – fan-out.

The value in each of the nodes of the weighted-graph is used as input to the branch and bound algorithm (which is used to solve the IP problem). An optimal solution to the logic restructuring (and thus the IP) problem is a sequence of nodes that give the minimum delay in the circuit. The heuristic technique used here, in one run through the circuit, explores every possible permutation of nodes, i.e. every possible path in the graph before arriving at a solution.

The initial step of the solution process, before the IP problem is solved is node collapsing. Node collapsing is one of the most widely used circuit optimization tools in practice [5, 6, 9, 13]. As a result, the total number of logic gates is reduced and a balanced circuit configuration is

step #	P – current node	Delay d_{ij}	$f(u)$ value	number of nodes – n	nodes compared
1.	A	0	0	7	-
2.	B	1	1	6	{A,B}
3.	C	2	2	5	{A,B,C}
4.	D	3	5[AC,CD]	4	{A,B,C,D}
5.	E	4	5[AC,CE]	3	{A,B,C,D,E}
6.	F	0 (no edge)	5[AC,CE]	2	{A,B,C,D,E,F}
7.	G	0 (no edge)	5[AC,CE]	1	all nodes.

Table 1. Operation and results of the branch and bound algorithm.

obtained. The circuit after the node collapsing procedure becomes the actual input into the IP solver.

Identifying various constraints involved in the integer programming problem forms an important part of the problem-solving process. The typical constraints involved in this IP problem are:

- *node constraints* – set up a specific domain for each circuit parameter, and
- *assignment constraints* – represent the nature of the inputs of every gate.

These constraints define the domain of parameters associated with every node in the circuit and may also be used to specify lower and upper bounds on selected values of each circuit parameter. This latter situation results in reduced number of constraints in the case if integer programming compared with that in the case of linear programming based approach. The objective function of the IP takes into account each of these constraints. The *lower bound* is given by $f_{in} \geq 1$ i.e. minimum of one input, while the *upper bound* is given by the maximum number of inputs at any node in the circuit $\leq f_{in}$.

After circuit constraints are defined, the next step is the definition of the objective function. This function defines mathematically the basic objective of the integer programming problem. It is given by the equation:

$$Min \sum_{i,j} d_{ij} x_{ij}$$

where d_{ij} denotes the arc-costs (delay) between nodes i and j, while the $x_{ij} = 1$, if path (i, j) exists in the graph, and 0 otherwise (this is an assignment constraint).

Problem prepared in the above described way can be solved, and one of the best approaches to the solution of an IP problem (especially in the case of data representation based on a weighted graph) is a branch and bound approach described in the next section.

4. The Branch and Bound Algorithm

The optimal solution to the integer programming problem is a sequence (or its permutation) of nodes that results in the minimum delay for the circuit. Note that an IP problem, in general, is NP-hard – as it is a subset of the 3-SAT Circuit Satisfiability problem [12]. When an integer programming formulation of the logic restructuring problem is to be solved, we can apply the branch and bound algorithm as a solution heuristic. Let us denote by n the number of nodes, $f(u)$ function that gives the best possible permutation of the circuit and p pointer to the current node in the subsequence. The proposed branch and bound approach consists of the following steps:

1. Increment the pointer p by one
2. $f(p) = f(p) + 1$ – update the best possible permutation based on the current node
3. for any node $k = 2\ldots (p - 1)$, if $f(p) = f(k)$ then proceed to step 2 – this will ensure that the same node has not been selected more than once
4. for any two nodes i, j, if $d_i > d_j$ then include node j in the formulation for $f(p)$ – here d_i refers to the delay associated with node i in the circuit.
5. if $p = n - 1$, then proceed to step 6, otherwise go to step 2
6. compute value of $f(p)$ for the entire sequence of nodes – this will give an optimal solution to the problem.

Not that the bounding condition of the algorithm is given by $f_l < f(u)$ where $f_l = \sum^{p}_{k=2} \sum^{k-1}_{l=1} \{d_{kl} f(k)\ f(l)\}$and d_{kl} denotes the delay between any two nodes k and l in the circuit.

In Table 1 we present a sample run of thus described algorithm applied to the 7 node 1 bit adder represented in Figure 1. As follows form the algorithm, the branch and bound process stops after 7 steps, when all nodes of the adder have been visited. In the case of one-bit adder represented in Figure 1, and solution of which is illustrated in Table 1, the optimal $f(u) = 5$ (see column $f(u)$ value).

5. Experimental results

Let us now illustrate the performance of the integer programming approach to the problem of logic restructuring in wave-pipelined circuits by a number of

examples. The first experiment uses standard CMOS circuit logic gates and the results representing values of parameters D_{min} and D_{max} for increasing total number of nodes and edges in the circuit are depicted in Table 2 and Figure 2 (data represented in Figure 2 originates from columns D_{min} and D_{max} in Table 2).

Case #	D_{min}	D_{max}	# of edges (in graph)	# of nodes
1	3	8	10	7
2	4	12	12	8
3	6	14	15	10
4	8	18	21	25
5	12	21	32	40

Table 2. Effects of circuit complexity on the circuit parameters D_{max} and D_{min}; times in nano seconds.

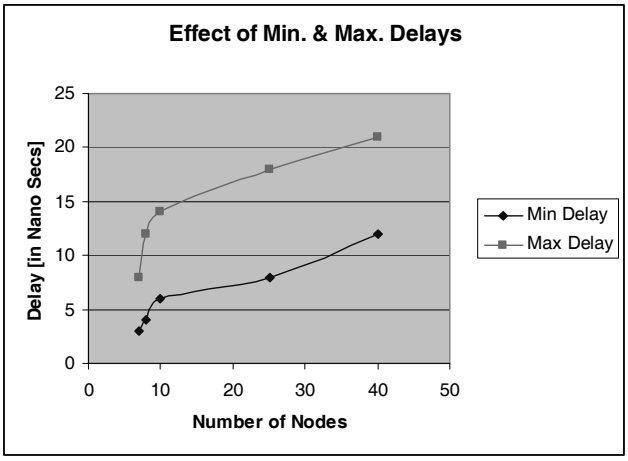

Figure 2. D_{min} and D_{max} as a function of the number of nodes in the circuit.

A few observations can be made. (1) As it is easy to see, plots of D_{max} and D_{min} are not linear when the number of nodes increases initially (7, 8, 10 nodes). It is only when the total number of nodes is larger than 10, when the further growth of both D_{min} and D_{max} becomes practically linear. (2) Furthermore, when the number of nodes is increasing initially from 7 to 10, the increase in the value of D_{max} is more substantial than the corresponding increase in the value of D_{min}. For larger number of nodes, both D_{min} and D_{max} grow at a similar rate. (3) Finally, in the case of the 1-bit adder circuit (7 nodes, 10 edges $\rightarrow$ row 1 in Table 2), the value of D_{max} - D_{min} = 5, which is exactly the optimal value of $f(u)$ obtained when the integer programming approach was applied to it (see Table 1).

In the next series of experiments we have compared the performance of the linear programming and integer programming based approaches to logic restructuring. Before we proceed further, a few observations about interactions between the problem we are solving and the two approaches to its solution (note that these points are not specific to the IP and LP as such, but originate from the problem we are solving).

S.No	Linear Programming [LP] Approach	Integer Programming [IP] Approach
1.	Domain of the path constraints may be non-linear.	Domain of every constraint is strictly linear.
2.	Objective Function: to minimize the power dissipation P_{diss} of circuit.	Objective Function: to minimize the circuit delay D.
3.	Weighted-graph representation not possible	Weighted-graph representation used in the heuristic technique.
4.	Difficult to model certain circuits due to non-linearity.	A wide variety of circuits can be easily modeled.

Table 3. Linear vs. integer programming based approaches.

Nature of the Constraints: Some of the constraints involved in the linear programming problem formulation may be nonlinear in nature. For instance, the expression for the propagation delay is given by [10]:

$$T_{prop} = \Delta + (C_{in} * S_k)$$

where, Δ denotes an internal time delay of the node, C_{in} is the input capacitance of the gate and S_k is a constant factor that relates the value of Δ to the input capacitance. This makes it difficult for some circuits to be modeled as a linear programming problems. On the other hand, an IP formulation of the problem has only linear constraints to consider.

Objective Function: Since the input capacitance is related to the power dissipated by the gate the LP problem aims to minimize the power requirements [10]. The IP problem, on the other hand, takes into account the fan-in (f_{in}) and fan-out (f_{out}) of the nodes, thus minimizing the delay requirements.

Circuit Complexity: As the number of nodes in the circuit increase, the non-linear problem solvers in LP techniques become more complex in nature. This increases the optimization time [11].

	D_{min}	D_{max}	# of steps (LP)	# of steps (IP)	D_{max} - D_{min}
1.	3	8	10	7	5
2.	4	12	12	8	8
3.	6	14	15	9	8
4.	12	21	23	14	9
5.	8	18	18	11	10
6.	10	22	25	16	12
7.	13	27	28	19	14
8.	15	31	31	23	16
9.	18	38	33	27	20

Table 4. Comparing LP and IP based approaches.

In the IP approach, the heuristic-technique applied is more efficient as well as simpler and thus faster to implement due to the weighted-graph representation. LP problem cannot use the weighted-graph representation due to the non-linearity of certain constraints.

These considerations have been summarized in Table 3 to give a complete picture of differences between the linear programming and integer programming based approaches as applied to the logic restructuring problem in the case of wave pipelining circuits.

We have applied linear programming and integer programming based techniques to nine different circuits and compared number of steps necessary to obtain the solution. The results are summarized in Table 4.

As we can see, the non-linear nature of some of the constraints involved in the LP makes it difficult for it to solve larger circuits using the linear programming approach. In the case of the integer programming approach, this problem is taken care of by restricting the domain of the constraints to be linear in nature. This is achieved through assignment constraints and the subsequent weighted-graph representation.

Furthermore, the number of iterations required to solve the problem using LP is always larger than that in the IP approach and this difference can be attributed to the nonlinearities materializing in the LP approach. This in turn increases the total time needed for circuit optimization. Obviously, here we are solving a set of problems that are characterized by a very small number of nodes, but as the number of nodes increases, this effect will become more and more significant.

Finally, the additional cost in the LP approach is incurred due to the nonlinear nature of the node constraints. Since the number of iterations required is proportional to the optimization time, the time required for delay balancing also increases. The heuristic technique used i.e. the branch and bound algorithm along with the lesser number of constraints approach accounts for the lesser number of iterations in the IP approach.

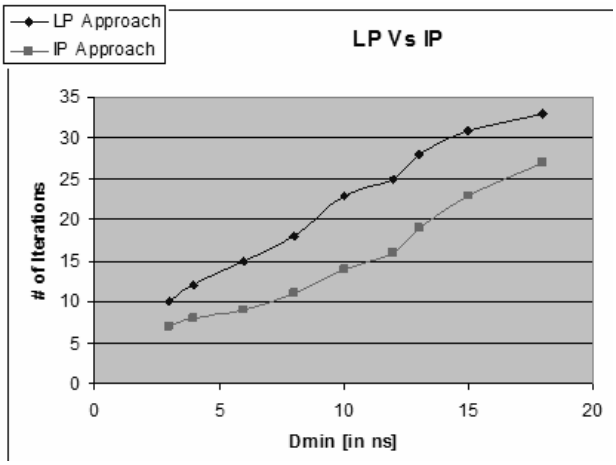

Figure 3. D_{min} as a function of the number of iterations.

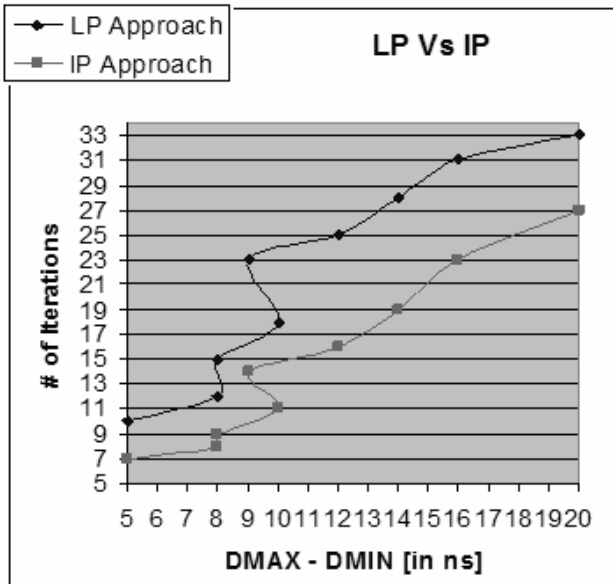

Figure 4. Number of iterations as the function of the difference D_{min} - $D_{max.}$

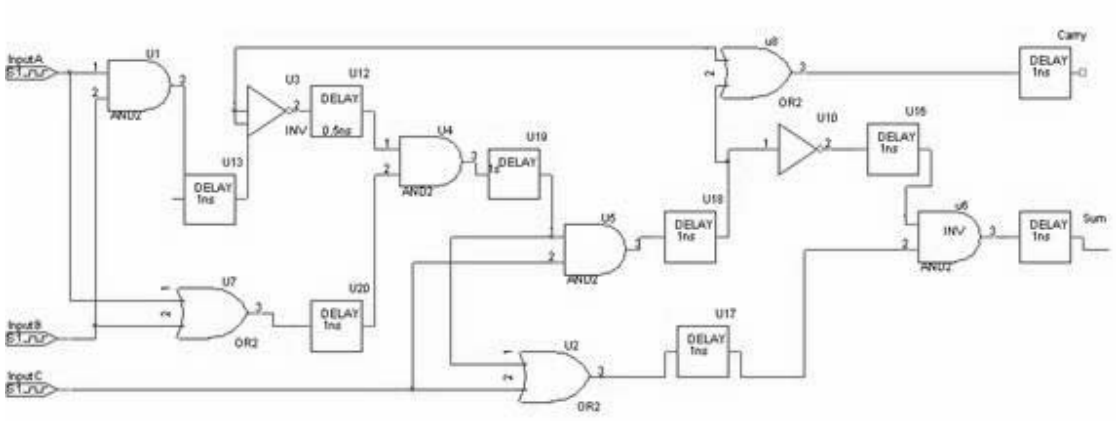

Figure 5. Simulation circuit for a 1-bit adder

	Circuit	Output Pins	Minimum Delay (without node collapsing)	Minimum Delay (with node collapsing)
1.	One-Bit Adder	Sum and Carry	7	2.5
2.	Four-Bit Adder	Sum and First Carry	8	2.5
3.	Four-Bit Adder	Sum, First Carry and Second Carry	10	2.55
4.	Four-Bit Adder	All pins included.	11	2.6
5.	Eight-Bit Adder	All pins up to and including the fourth carry.	12.5	2.6
6.	Eight-bit Adder	All pins up to and including the fifth carry.	14	2.61
7.	Eight-Bit Adder	All pins up to and including the sixth carry.	15	2.62
8.	Eight-Bit Adder	All pins included.	16	2.62

Table 5. Minimum delay Dmin values for the adder circuit.

To illustrate these points further, in Figure 3 we depict the relationship between the # of iterations and the value of $D_{min.}$. Note that the increase of the value of $D_{min.}$ indicates that the difference D_{min} - $D_{max.}$ is reduced for a certain $D_{min.}$ value with a minimal variation. Therefore, executing larger # of iterations results in a narrower gap between $D_{max.}$and $D_{min..}$ This allows one to realize balancing of delay between the shortest and longest paths, which is one of the important characteristics of the wave-pipelined circuits.

Finally in Figure 4 we compare the performance of linear programming and integer programming based approaches in terms of number of iterations as the function of the difference between the values of D_{max} and D_{min}. In other words, we want to see how many iterations of either approach are necessary to achieve the same reduction in the value of D_{min} - $D_{max.}$.The plot clearly shows the lesser number of iterations in IP for various values of the delay difference, when compared to that in LP.

In the last series of experiments we have gone beyond the single bit adder and applied the proposed techniques to a number of found and 8 bit adders. Each of the subsequent adder circuits uses the design of the 1-bit adder, depicted in Figure 5, as the basis. For each adder we have applied the complete logic restructuring process as described in Section 2. Then we have applied the Pspice simulator to evaluate the resulting circuits obtained in two ways, with and without node collapsing. For the node collapsing we have implemented the procedure dscribed in [5, 6, 9]. Table 5 shows the minimum delay D_{min} values (in *ns*) for the two cases.

It can be easily seen that the difference between the minimum delay values gradually increases with an increase in the complexity of the circuit. This gives an idea about the effectiveness of the branch and bound heuristic and the node collapsing procedures. Furthermore, the D_{min} value after node collapsing does not change by a significant amount. This is due to the fact that the ripples carry characteristics in the adder circuits result in the saturation of the minimum delay value in the circuit.

6. Concluding remarks

In this paper we have addressed the problem of logic restructuring in wave-pipelined circuits and showed that it can be effectively solved using an integer programming based optimization. The success of optimization depends on how well the circuit is first conditioned using the node collapsing procedure is carried out followed by the subsequent circuit optimization. The results of our experiments showed that the number of nodes in the optimized circuit was reduced by about a factor of 1.5 in comparison to the original circuit, which without loss of generality will result in a reduced power consumption that is critical in asynchronous circuits. The branch and bound heuristic used to solve the delay balancing in wave pipelined circuits was found to be more effective in case of the IP approach than the conventional LP approach. The difference between the minimum and maximum delays in the circuit depended on the circuit configuration as well as the degree of the node collapsing procedure.

The following are some of the topics where additional research could be carried out in logic restructuring for delay balancing. (1) Analysis of the additional cost – the additional overhead involved in the IP approach that needs to be analyzed in greater detail would be the cost incurred in the weighted-graph representation of the given circuit i.e. the additional cost incurred in pre-conditioning the circuit before applying the heuristic procedure. (2) Trade-off between the reduction rates – The reduction rates between the delay difference ($D_{max} - D_{min}$) and the number of iterations for both the LP and IP approaches needs to be considered. This will give a better focus on how efficient the heuristic search has been applied in the IP approach. A trade-off analysis needs to be done on whether it is the delay difference or the number of iterations which is the deciding factor.

References

[1] J.F. Bard, *Integer Programming*, Edition 2, Chapter 3, John Wiley & Sons, New York (1988).

[2] J.Fishburn, "Clock Skew Optimization", *IEEE Transactions on Computers*, Vol. 39, Issue 7, July 1990, Page(s) 945-951.

[3] D.Roy and M.Ciesielski, "Clock period minimization with multiple wave propagation", Proceedings of the 28th Design Automation Conference on, July 1991.

[4] Burleson et al., "Wave-Pipelining: A tutorial and research Survey", *IEEE Transactions on VLSI Systems*, Vol. 6, Issue 3, Sep 1998.

[5] T.S. Kim et al., "Logic restructuring for wave-pipelined circuits", Proceedings of the IEEE International workshop on Logic Synthesis, Sep 1993.

[6] Cao et al., "Non-Crossing Ordered BDD for Physical Synthesis of Regular Circuit Structure", Proceedings of the ICCD, July 2003.

[7] Bertsimas D, *Introduction to Linear Optimization*, Chapter 3, Athena Scientific, New York (1997).

[8] Morris M., *Digital Design*, Edition 3, Chapters 5 & 6, Aug (2001).

[9] http://jason.sdsu.edu/minc/13.pdf.

[10] Chris J. Meyers. *Asynchronous Circuit Design*, Chapter 5, Wiley (2001).

[11] Brian et al., *Surfing in Wave-Pipelined Circuits*, John Wiley & Sons, New York (1999). Page(s) 35-40.

[12] Cormen, et. al. *Introduction to Algorithms*, McGraw Hill (1990).

[13] Woo Kim, Yong Kim, "Automating Wave-Pipelined Circuit Design", *IEEE Design & Test of Computers*, Vol. 20, Nov 2003.

[14] Vijay Sundararaian et al, "Synthesis of Low Power CMOS VLSI Circuits Using Dual Supply Voltages", Proceedings of the 36th ACM/IEEE conference on Design automation, Pages: 72 - 75 , 1999

[15] Manfred Padberg, Minendra P. Rijal, *Location, Scheduling, Design & Integer Programming*, Wiley, July 1996.

[16] Michael J. Brusco, "Optimal Solution Methods for the Minimum-backtracking row layout problem", *IIE Transactions*, Vol. 36, No.2, Feb. 2004, Page(s) 181-189.

[17] Michel Berkelaar and Koen van Eijk, "Efficient and Effective Redundancy Removal for Million-Gate Circuits", DATE 2002

[18] A. E. Dunlop, V. D. Agrawal, D. N. Deutsch, M. F. Jukl, P. Kozak, and M. Wiesel, "Chip layout optimization using critical path weighting," Proc. of ACM/IEEE 21st Des. Auto. Conf., (Albuquerque, N.M.), pp. 133-136, June 25-27, 1984.