Fast Timing Closure by Interconnect Criticality Driven Delay Relaxation

Love Singhal and Elaheh Bozorgzadeh
Donald Bren School of Information and Computer Sciences
University of California, Irvine, California 92697-3425
Email: lsinghal,eli@ics.uci.edu

***Abstract*— Due to decreasing transistor sizes and increasing clock frequency, interconnect delay is a dominant factor in achieving timing closure in deep sub-micron designs. Techniques like wire pipelining and retiming can manage delay of timing critical wires. The latency of the system, however, limits the total pipelining in the design. New techniques are, thus, needed at synthesis stage to consider the effect of critical wires in the design. In this work, we propose a novel intuitive algorithm, Critical Edge Reduction (CER) algorithm, which produces a maximal delay budgeting solution under fixed latency while minimizing the number of critical wires. We also present an in-depth analysis of trade-off between maximum budgeting and critical edge minimization. We implemented our design flow using a set of MediaBench data paths on Xilinx VirtexE FPGA devices. Using our algorithm, the Xilinx Place and Route tool achieved timing closure, on average, 2.8 times faster than using maximum budgeting. The resulting average clock period using CER algorithm outperforms the one using maximum budgeting by 6%.**

I. Introduction

Due to increasing design complexity and clock frequency, timing closure cannot be achieved easily in a design flow and design time can get unacceptably long. In today's high performance chip design, wire delay has a significant impact on timing closure of the design. The delay of a long wire can be comparable to the delay of logic gates. This increases the complexity of the design making the task of routing at physical layout stage harder. In order to reduce the complexity and design time, interconnect planning must be integrated in high level synthesis. Our goal in this work is to reduce the number of timing critical interconnects at the datapath level in order to reduce the complexity of physical layout stage and leverage that burden on early stages of design flow.

In order to meet the timing constraints, designers use resource pipelining by delay budgeting techniques [1]. The extra clock cycle latency (or *budget*) assigned to the operations is used for resource pipelining. *Retiming* allows the movement of pipeline registers from resources to timing critical wires or other resources. Hence, wire pipelining can be equivalently handled by delay budgeting on resources coupled with retiming. In addition to these pipeline registers, we can add additional pipeline registers to edges that have a slack of few clock cycles, as the extra registers will not affect latency constraints but can reduce the wire delay of those edges. If, in a design, the number of edges with slack is increased, then registers can be added on more number of wires. We noted that a well placed register on a wire can reduce the wire delay by 40%.

Figure 1 shows two different delay assignments on a data flow graph. The number inside a node represents the respective latency of each node. The latency constraint is 20 clock cycles. Delay assignment in both graphs is maximal, i.e., no more additional latency can be allowed at any node. However, the Solution B has fewer number of critical edges compared to Solution A. The two edges that have slack in Solution B (shown with shaded arrows) can each be assigned extra registers. These registers can allow the placement tools to place the two parallel critical paths far from each other without affecting the clock frequency and clock latency. However, in solution A, during placement, all the modules have to be placed close to each other to meet the strict timing conditions and as a result, design becomes more complicated and may not meet the timing constraint.

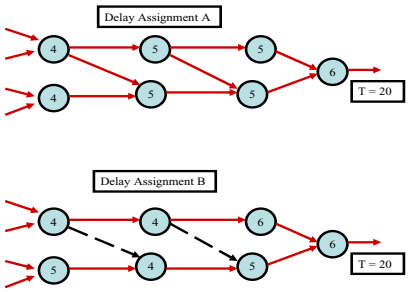

Fig. 1. Critical edge minimization during delay assignment.

Our goal is, therefore, to find more potential locations for adding registers on wires. Solution B is better than solution A for effective wire pipelining. None of the existing delay relaxation techniques consider the criticality of edges during delay assignment on the nodes. Our goal is to generate such delay assignments that have more number of non-critical edges along with fully pipelined resources under fixed latency constraints.

In this paper, we propose a polynomial time algorithm named CER which minimizes the number of critical edges in a design at datapath level. We use our algorithm along with resource and wire pipelining, and retiming on a set of MediaBench applications. Then we implement our designs on FPGA devices. In FPGAs, the wire delay is quite significant due to presence of routing switches in connecting wires. Our experimental results show that increasing number of non-critical edges and adding registers to non-critical edges in a design results in 2.8 times speedup in place and route runtime, on average.

The rest of the paper is organized as follows: Section II gives an overview of related work in delay budgeting and wire pipelining. Some preliminaries and the formulation of critical edge minimization problem are described in Section III. In Section IV, our heuristic algorithm, Critical Edge Reduction, is presented. In section V, the experimental flow of applying our method in wire pipelining at the data-path level is presented. Section VI concludes our paper.

II. Related Work

Designers use delay relaxation techniques to add pipeline registers in the design. The maximum delay budgeting focuses on maximum slowdown of components under the given latency constraints. In [2], polynomial algorithms are proposed. Researchers in [3] proposed a delay budgeting on sequential circuits. Recently, researchers in [1], have proposed a polynomial optimal algorithm for maximum delay budgeting using network flow technique. In all existing work, either

producing optimal solution or not, the output is a design in which the delay of none of the components can be further increased unless latency constraint is violated. As a result, the design has more critical edges (edges with zero slack). Critical edges have tighter timing constraints in later stages of design flow. Our work, therefore, targets to minimize the number of critical edges in a design to further reduce the design complexity while keeping maximal budgets on the nodes.

Wire pipelining has been recently studied extensively [4]–[7]. Although in [8], it is argued that wire pipelining can increase the control logic of the system, pipelining the global long wires which dominate the clock frequency can significantly enhance the clock frequency of the design [6], [7]. In [5], [6], wire pipelining is applied while considering placement and availability of pipeline registers. In [5], retiming is applied to distribute the pipeline registers. Though retiming can redistribute the registers along a path, the total number of registers along each path or cycle is a constant. This makes the scope of improvement limited. In our approach, we aim at increasing the number of non-critical edges which can be used as potential locations to inject pipeline registers.

III. Interconnect Criticality Driven Delay Assignment

A. Preliminaries

The operations of a data path are represented as nodes of a directed acyclic graph (DAG) and interconnects are represented by edges of that graph. The difference between required time and arrival time of an edge is called slack of that edge. Critical edges have zero slack and non-critical edges have positive slack. The extra delay assigned to a node is termed as delay budget on that node. In this paper, we use delay budget and budget interchangeably.

If after assigning budgets to the nodes, no node can get any more budget, then the budget assignment is called *maximal delay budget assignment*. The slack of each node is zero in a maximal budgeting. However, slack of all the edges is not necessarily zero. The necessary and sufficient condition for maximal budgeting is that each node (except PIs and POs) has at least one critical incoming edge and has at least one critical outgoing edge. A maximal budgeting solution can be obtained in polynomial time by applying the existing algorithms [1], [2], [9].

B. Problem Formulation

The problem of criticality-driven delay budgeting is to obtain a Maximal Budgeting solution with Minimum Critical Edges (henceforth referred to as MB-MCE). Equations 1 to 8 show a $0-1$ mixed integer linear programming (MILP) formulation of the problem. Variable x_{ij} is a binary variable which is 1 if edge e_{ij} is critical and 0 otherwise. Variables b_i, a_i, and d_i represents budget, arrival time, and delay on node v_i. Variable s_{ij} represents slack on edge e_{ij}. Variable T is latency constraint and variable M is some large constant. Equation 1 is the objective function. Equation 2 specifies latency constraints. In a maximal budgeting solution, each node must have at least one incoming critical edge and one outgoing critical edge. These constraints are explicitly specified in equations 6 and 7. Other equations relate arrival times, required times and slack with the binary variables x_{ij}.

$$Min \sum_{e_{ij} \in E} x_{ij} \tag{1}$$

$$a_i \leq T \qquad \forall v_i \in PO \tag{2}$$

$$a_j = a_i + b_j + d_j + s_{ij} \tag{3}$$

$$s_{ij} \leq M \times (1 - x_{ij}) \quad M : Large\ constant \tag{4}$$

$$-s_{ij} < M \times (x_{ij}) \tag{5}$$

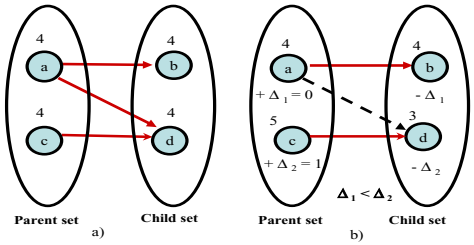

Fig. 2. a) A Critical Bipartite Sub-graph. b) Delay Re-assignment on a Critical Bipartite Sub-graph. Numbers on node represent their delay budgets.

$$\sum_{\forall j, e_{ij} \in E} x_{ij} \geq 1 \qquad \forall v_i \in V \tag{6}$$

$$\sum_{\forall i, e_{ij} \in E} x_{ij} \geq 1 \qquad \forall v_j \in V \tag{7}$$

$$x_{ij} \in \{0, 1\} \qquad a_i, s_{ij}, b_i \geq 0 \tag{8}$$

Though our problem considers maximal budgeting on the nodes, it does not restrict wire pipelining of edges. The budget from nodes can be transferred to edges using retiming. However, since in maximal budgeting, each node has at least one incoming and one outgoing critical edge, retiming can not add registers to any non-critical edge of the graph unless retiming removes budget from the nodes at either of its ends. Hence, this problem tries to find additional locations for adding registers that retiming could not cover. The time complexity of this problem for general DAG is an open problem and it is currently under investigation. To the best of our knowledge, no polynomial time optimal algorithm is known for the problem. We have developed a heuristic algorithm, which we call Critical Edge Reduction (CER) algorithm, to solve the MB-MCE problem. Although it is not guaranteed to produce an optimal solution, the CER algorithm is intuitive and, as our experiments show, can be quite close to optimum.

IV. CER: Critical Edge Reduction Algorithm

The CER algorithm starts from a maximal budgeting solution on a given directed acyclic graph. It then extracts all critical bipartite subgraphs (explained in Section IV-A) from the given graph, and reduces the critical edges in those bipartite subgraphs. A systematic way of reducing the critical edges in critical bipartite subgraphs is through the use of composite nodes, modeled in Section IV-B. In Sections IV-C and IV-D, we propose the novel techniques of doing this task systematically and efficiently. Finally, in Section IV-E, we give the complete algorithm of reducing critical edges to non-critical edges on a DAG. Our algorithm does not make any existing non-critical edge critical and hence, converges fast and generates a solution close to optimal solution.

A. Budget Re-Assignment on Critical Bipartite Graph

From a given DAG, we extract critical bipartite subgraphs. In a maximal budgeting, a critical bipartite subgraph is an induced bipartite subgraph such that all the edges in this subgraph are critical and the subgraph includes all the critical edges incoming or outgoing to the nodes in the subgraph. The bipartite graph has two independent sets of nodes: One set with only outgoing edges (parent set) and the other set with only incoming edges (child set). The node with only outgoing edges is referred to as a parent node. The node with only incoming edges is referred to as a child node. Figures 2 a) and 3 a) shows examples of such a subgraph.

The shifting of budgets from parent nodes to child nodes or vice versa is termed as *budget reassignment*. When there is an unequal

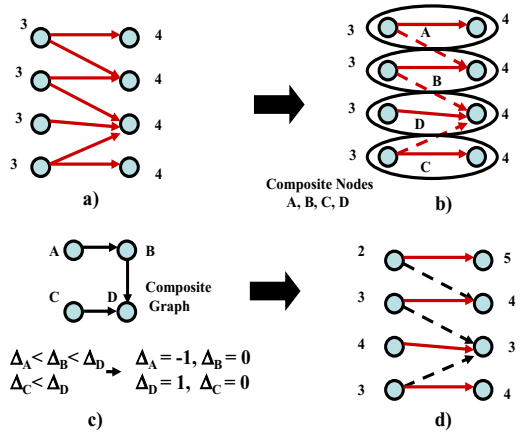

Fig. 3. Critical Edge Minimization. a) A Critical Bipartite Subgraph b) Decomposed to four critical bipartite subgraphs (composite nodes). c) Composite Graph. d) Delay budget re-assignment to convert composite edges to non-critical edges.

exchange of budget between parent node and child node and latency constraint is satisfied, the edge connecting the two nodes will get a positive slack and will become non critical. Figure 2 b) shows that when the budget on nodes a and b are exchanged by Δ_1 and budget nodes on c and d exchanged by Δ_2, and if Δ_1 is less than Δ_2, then the edge between nodes a and d becomes non-critical. The new budgeting is still a maximal budgeting as all the nodes are connected with at least one critical edge. In the new budgeting, nodes a and b, and nodes c and d form two different critical bipartite subgraphs.

Hence, if we can reduce in this way most number of critical edges to non-critical in various critical bipartite subgraphs of the graph, we could solve the MB-MCE problem.

B. Composite Nodes and Composite Edges

In the previous section, we explained how a critical bipartite subgraph can be decomposed to smaller critical bipartite subgraphs. Each smaller critical bipartite subgraph is termed *composite node*. An edge connecting the two composite nodes, exposed edges between the decomposed subgraphs, is termed as *composite edge*. The graph consisting of composite nodes and composite edges is called *composite graph*. The composite edges are potential edges to become non-critical after delay budget exchange.

Figure 3 shows construction of composite graph and shows composite nodes in the graph.

Before proceeding further into the details of reassigning budgets on composite graph, we would like to highlight important properties of the composite nodes. A composite node has the following properties:

- It should be a valid independent critical bipartite graph.
- It should have at least one parent node and one child node.
- Every node in the critical bipartite subgraph should belong to one and only one composite node.
- The budget exchange between parent nodes and child nodes in a composite node is equal. This budget exchange inside a composite node is henceforth referred to as just Δ for the sake of brevity. The positive value of Δ refers to an increase in budget of parent nodes and decrease in budget of child nodes in a composite node by the same amount.This ensures the condition of maximal budgeting after budget reassignment.

The next section gives an algorithm for creating composite graph from a bipartite graph with an objective to maximize the number of composite edges.

```
function construct_composite_graph (Graph G)
        c = Null;
        CG = new CompositeGraph ();
        while (G is not empty)
                v = find_min_degree_node (G);
                if (deg (v) == 0)
                        c.add (v);
                else
                        c = new CompositeNode (v);
                        c.add (get_all_neighbors (v, G));
                        CG.add (c);
                remove_nodes_from_graph (c, G);
        return CG;
```

Fig. 4. Pseudo-code for constructing composite graph

C. Composite Graph Construction

The idea is to keep least number of edges inside the composite nodes so that most of the edges that are outside (composite edges) could become non-critical.

Lemma 1: If there is a directed edge from composite nodes A to B and Δ in composite node A is less than Δ in composite node B, it is a sufficient case to transform the edge AB to a non-critical edge.

A composite edge, therefore, imposes a less-than constraint on values of Δ at the two end nodes of the edge. This constraint should be satisfied for every composite edge in the graph. In order to have a valid ordering on the values of Δ, the composite graph, therefore, should not have any *cycle*. Hence, the problem of constructing composite graph is to group the parent and child nodes in such a way that the number of external edges (composite edges) is maximized and the resulting composite graph is a directed acyclic graph. Figure 3 shows the example of a critical bipartite graph with the corresponding Δ values.

We propose a greedy algorithm to generate composite graph from a given critical bipartite subgraph. Figure 4 shows the pseudo code of the algorithm. Since the goal is to expose as many edges outside the composite nodes, the algorithm finds a node with minimum edge degree as a seed for composite node and adds the neighbors around the seed.

Lemma 2: Any new composite node created in each iteration of the algorithm will have either outgoing edges to the remaining graph (the graph that remains to be clustered) or incoming edges from the remaining graph; and can not have both incoming or outgoing edges.

From Lemma 2, the following theorem holds true.

Theorem 1: The composite graph created by the algorithm in Figure 4 is acyclic.

D. Assigning Δ on composite graph

After the composite graph is constructed, the next step is to assign Δ values to the composite nodes such that Lemma 1 is satisfied for every composite edge. Such an assignment will ensure that all composite edges become non-critical.

The budget of parent nodes in a composite node is increased by value of Δ and budget of child nodes is decreased by value of Δ. If budget of any node is increased by Δ, the slack of non-critical outgoing edges from that node is decreased by Δ. So, budget on any node can not be increased by amount equal to or more than the minimum slack of its non-critical outgoing edges. Also, if budget of any node is decreased by Δ, absolute value of Δ can not exceed the current budget on that node. These constraints impose an upper bound and a lower bound on the value of Δ for each composite node.

Due to these upper and lower bounds on Δ for each composite node, a perfect assignment where Lemma 1 is satisfied for every

```
function assign_delta (CompositeGraph G)
    value = large_number;
    while (G is not empty)
        S = find_all_output_nodes (G);
        value =
        find_max_delta_less_than_cur_value
            (value, S);
        assign_delta_to_nodes (value, S);
        remove_nodes (S, G);
```

Fig. 5. Pseudo code for Δ assignment.

```
function CER_impl (Graph G)
    find_maximal_budgeting (G);
    L = sort_nodes_by_arrival_time (G);
    while (L is not empty)
        S = find_nodes_with_min_arr_time (L);
        P = find_critical_bipartite_subgraph (S, G);
        CG = construct_composite_graph (P);
        assign_delta (CG);
        update_slack_on_edges (G);
        remove_nodes (S, L);
```

Fig. 6. Pseudo code for CER algorithm

edge may not be possible. Some of the critical edges will, therefore, remain critical. The problem of finding a Δ assignment in which total sum of the weights of composite edges that become non-critical is maximized is an NP Hard problem. We can reduce Longest Path [10] problem on planar graphs to the above mentioned problem. The long proof is, however, out of scope of this paper. The pseudo code of a heuristic algorithm to solve Δ assignment problem is given in Figure 5. The algorithm is similar to reverse breadth first search.

Assignment of Δ is a generic and important phase in the overall critical edge reduction (CER) algorithm. The function *find_max_delta_less_than_cur_value* in Figure 5 can be modified to be more circuit aware. The delta allocation stage can use circuit awareness to assign positive or negative values to delta on each node. For example, in a design, clock speed of a module is proportional to the amount of budget on it as that budget determines total stages of pipeline. So, if a node has the lowest budget (and thus lowest clock speed), then the delta allocation stage can make sure that the node gets a positive budget so that its clock speed increases. This function essentially gives designer a control over assigning budgets to special nodes.

E. Critical Edge Minimization on a DAG - CER Algorithm

The composite graph construction and then, its subsequent Δ assignment can reduce the critical edges on a critical bipartite subgraph. In order to apply the two algorithms on a general DAG, the given graph should be decomposed into a set of critical bipartite subgraphs. The following Lemma holds true.

Lemma 3: A directed acyclic graph with maximal budgeting can be decomposed into a set of critical bipartite subgraph. Each node will belong to exactly two critical bipartite subgraphs adjacent to each other. In one subgraph, the node serves as parent node while in other set, it serves as a child node.

In order to extract the critical bipartite subgraphs, the CER algorithm picks up the nodes with minimum arrival time as parent nodes of the critical bipartite subgraph. The pseudo code of the CER algorithm is given in Figure 6.

The above algorithm traverses the graph from primary inputs to primary outputs. In each iteration, arrival times of only parent nodes are modified and so, we do not need to modify the arrival times of any other node. Also, the Δ assignment stage is guaranteed to not make any non-critical edge critical. This is because it considers the slack of all the non-critical edges while defining the upper and lower bounds on Δ values of composite nodes. A single execution of the CER algorithm on a graph transfers budgets from primary outputs to primary inputs. It is found that more critical edges can be reduced if the algorithm is iteratively executed many times on an input graph. This is due to rapidly changing topology of critical bipartite graphs in the graph in each iteration. The algorithm is guaranteed to converge as it does not increase the number of critical edges in any iteration. For all our experiments, the algorithm converged in less than 10 iterations and further iterations did not affect circuit topologies. The CER algorithm is quite efficient and is, therefore, a better alternative to inefficient ILP based solution.

Lemma 4: The time complexity of CER algorithm is $O(VE)$.

V. Experiments

In this work, we apply delay budgeting at data path level and show optimizations that can be done by adding pipeline registers both on resources and wires, and improving non-critical paths on FPGAs. First, in the next subsection, we show the effect of wire delay in FPGAs.

A. Wire Delay in FPGAs

In FPGAs, the wire delay is a significant component of total delay. This is due to the presence of routing switches in connecting wires. Since wires in FPGAs are prefabricated, nets are connected together through programmable switches which are basically transistors connecting the wires.

In this section, we show the effect of wire delay in FPGA. We implemented our design on Xilinx VirtexE device. A very simple data flow graph with two connected multipliers is used, with one multiplier connected to primary inputs and other to primary output. The two multipliers are placed at the two diagonally opposite corners of the FPGA. Table I shows the maximum delay of wires connecting the two multipliers. The maximum combinatorial delay of multipliers was found to be 10.66 ns. Table I shows that the wire delay is significant compared to delay of the multiplier and as the area of chip increases, wire delay across the chip increases.

Devices	$xcv200e$	$xcv600e$	$xcv1000e$
Wire Delay (in ns)	6.764	7.782	10.021
Devices	$xcv1600e$	$xcv2000e$	$xcv3200e$
Wire Delay (in ns)	11.170	12.691	16.173

TABLE I
MAXIMUM WIRE DELAY BETWEEN TWO MULTIPLIERS IN VARIOUS XILINX VIRTEXE DEVICES.

We, then, inserted a register in the edge connecting the two multipliers. It was found that the place and route tool automatically places the register in the middle of the chip. The maximum wire delay is then reduced to approximately 60% of the delay shown in table I. If the registers are added in between the two components, the two components can be placed away from each other, resulting in design simplicity and smaller wire delays. However, registers can not be added indiscriminately as the latency of system is fixed. They can be only added to non-critical edges. Hence, if we minimize the number of critical edges and add registers to non-critical edges, the design will converge faster and will meet timing more constraints easily.

TABLE II
BENCHMARK DESCRIPTIONS.

Benchmark	Description	No. of Nodes	Latency
inv1	Invert matrix 1 from Mesa	101	9
inv2	Invert matrix 2 from Mesa	215	16
inv3	Invert matrix 3 from Mesa	351	15
jpg1	IDCT application 1 from JPEG	57	16
jpg2	IDCT application 2 from JPEG	200	13
mat1	Matmul application 1 from Mesa	72	10
mat2	Matmul application 2 from Mesa	109	14
rot1	Rotate matrix 1 from Mesa	28	6

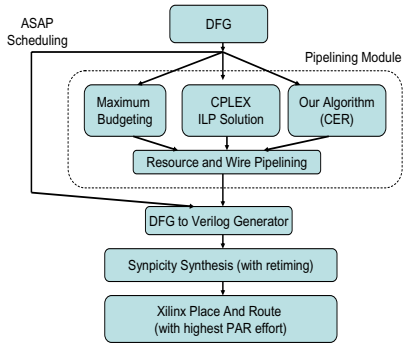

Fig. 7. Experimental Design Flow

B. Experimental Setup

We implemented our design flow on a set of 8 dataflow graphs extracted from MediaBench test suite [11] using SUIF compiler [12] and machine SUIF [13]. The circuits were implemented on Xilinx VirtexE FPGA devices. The overall design flow of experiments is given in Figure 7. The characteristics of the data flow graphs are mentioned in Table II. Experiments were conducted on Intel Pentium 4 3.2 GHz CPU with hyperthreading and 1 GB RAM, running Windows XP.

To compare the results of our algorithm, we implemented three different techniques - ASAP scheduling, maximum budgeting, and MB-MCE ILP formulation. In ASAP scheduling, each component was executed fastest without any extra pipeline register. In maximum budgeting, total budget on nodes was maximized. In order to have a fair comparison with our algorithm, we added registers on non-critical wires of maximum budgeting as well. We wanted to show the effect of our budgeting technique compared to maximum budgeting rather than the effect of adding registers on wires only. The MB-MCE ILP, given in Section III-B, was run on CPLEX ILP solver. We added one more constraint in the ILP that the total budget does not drop by more than 10% of the maximum budget. We believe that this is the best case as it gives the design both resource pipelining and wire pipelining advantages. We, then, assigned resource registers and wire registers found through the above two techniques, and our algorithm. The verilog files of the data flow graphs were then created. The verilog files were then implemented using Synplicity Synplify Pro 7.7.1 and Xilinx 6.3 PAR tool with tight timing constraints, retiming and highest speed effort. This was done as we wanted to get the best clock periods of various implementations and see how fast the designs could meet those constraints. The data paths were 16-bit wide. We added at most 4 registers in modules that have lot of combinatorial delay like multiplier.

TABLE IV
AVERAGE NUMBER OF CRITICAL EDGES AND TOTAL DELAY BUDGET

Critical Edges			Average Total Budget		
Max	ILP	Alg	Max	ILP	Alg
142.1	131.9	132.6	135.8	123.3	124.4

TABLE V
AVERAGE CLOCK PERIOD (IN NS) FOR DIFFERENT TECHNIQUES.

Clock period (in ns)			
ASAP	Max	ILP	Our Alg
12.14	9.36	8.84	8.79

C. Experimental Results

1) Critical edges and Total Budget: First, we show the number of critical edges reduced by the MB-MCE ILP (henceforth, referred to just ILP) and our algorithm. Table IV shows the average number of critical edges and average total budget in maximum budgeting, ILP solution, and our CER algorithm. As seen from the table, our algorithm is able to successfully reduce many critical edges and gives results close to optimum ILP solution.

2) Effect on clock period: In order to have fair comparisons of place and route time of all the designs, we ran each benchmark with a very strict timing constraint and at the highest PAR effort level. Along with place and route time, it is interesting to see the performance (clock speed) of each technique in such conditions. Table V shows the average clock period of various techniques. On average, our algorithm gives the best clock frequencies of all other techniques. While average clock period of maximum budgeting with registers is 6.4% higher than CER algorithm, average clock period of ASAP scheduling is 36.5% higher than clock period of CER algorithm. Since ASAP implementation is not pipelined, it has the highest clock delay. The average clock period of ILP solution is very close to our algorithm and confirms that a critical edge minimization (ILP) with budget more than 90% of maximum budget is indeed a good solution of the design.

3) Effect on Place and Route Time: Table III presents the place and route time of the benchmarks using various techniques. We compared only the place and route times of the design flow as this stage takes the longest time and all other steps take almost same amount of time. Hence, place and route time can be referred to as the run time of whole design flow.

As shown in Table III, the place and route of our algorithm is 2.8 times faster than place and route time of maximum budgeting, on average. This is significant considering that the designs by our algorithm have lower average clock period as well. The performance of ILP and our algorithm is almost same. The two techniques represent two different solutions to same problem of minimizing critical interconnects. The ILP solution is, however, not a polynomial time solution.

One interesting observation is that for ILP, though the circuit *inv1* had same number of critical edges and almost same budget as CER algorithm, the PAR runtime was quite higher than our algorithm. We observed that the ILP solver, being a mathematical tool, only tries to maximize or minimize the budgets and/or critical edges and gives

TABLE III

PLACE-AND-ROUTE RUNTIME (SEC) RESULTED FROM DIFFERENT TECHNIQUES. RATIO COLUMN REPRESENTS THE RATIO OF PAR TIME OF THE TECHNIQUE OVER PAR TIME OF CER ALGORITHM.

Benchmark	Place and Route Runtime (sec)						
	Our Algorithm (CER Alg)	ASAP		Max-Budgeting + wire-pipelining		Minimum Critical Edge ILP	
	Time	Time	Ratio	Time	Ratio	Time	Ratio
inv1	190	708	3.7	725	**3.8**	559	2.9
inv2	271	2335	8.6	861	3.2	335	1.2
inv3	368	1467	4	1119	3	539	1.5
jpg1	52	267	5.13	28	0.5	46	0.9
jpg2	273	519	1.9	472	1.2	231	0.9
mat1	229	928	4.1	578	2.5	100	0.4
mat2	124	77	0.62	547	**4.4**	203	1.6
rot1	58	279	4.8	56	1	28	0.5
Average	195.6	822.5	**4.2**	548.3	**2.8**	255.1	**1.3**

irregular budget distribution. In case of circuit *inv1*, along a path while some multipliers got budget of 4, some multipliers got budget of 2; whereas in case of CER algorithm, all those multipliers got a uniform budget of 3. The non-uniformity of ILP, in turn, affects the clock period and makes meeting of timing constraints very hard for the tool. The CER algorithm, however, allows transfer of budgets from the paths that have higher budgets to locally present *hot-spots* in the design.

For the benchmarks, *jpg1* and *rot1*, the PAR time of maximum budgeting is better than that of CER algorithm. In these small circuits (refer to Table II for the sizes), the budget of maximum budgeting is higher than that of CER algorithm and this budget decrease on multipliers in case of CER algorithm impacted the total clock frequency and hence, total runtime in meeting timing constraints. We observed that compared to large circuits, in small circuits, the effect of budgets on modules is more and the effect of wire delay is less. This is because wire delay becomes more significant compared to module delay in larger circuits than in smaller circuits due to higher overall area of larger circuits. Hence, resource pipelining is more advantageous than wire pipelining in small circuits.

We observed that, in general, modules in designs using our algorithm were placed more uniformly compared to designs using maximum budgeting. The congestion in designs using maximum budgeting was higher than designs using our algorithm. This is because modules that have less wire pipelining have to be placed close to each other to avoid routing overhead.

Overall, the results show that increasing the potential non-critical edges can improve the design quality. By minimizing critical edges and adding registers on non-critical edges, both clock period and design time can come down. The area occupied through our algorithm was also less than maximum budgeting as area due to additional registers for resource and wire pipelining is proportional to the total budget in a design and as shown in Table IV, the average total budget using our algorithm was less than that of maximum budgeting.

VI. CONCLUSIONS

We introduce a novel critical edge reduction CER algorithm in this paper. By performing critical path planning at early stage of design, we are able to reduce the complexity for place and route tools. None of the existing delay budgeting techniques consider increasing the non-critical edges to improve timing closure. Using our algorithm, the Xilinx Place and Route tool is capable of achieving timing closure faster than by using maximum budgeting.

REFERENCES

[1] S. Ghiasi, E. Bozorgzadeh, S. Choudhary, and M. Sarrafzadeh, "Unified theory of timing budget management," in *Proc. IEEE International Conference on Computer-Aided Design*, San Jose, California, Nov. 2004.

[2] R. Nair, C. L. Berman, P. S. Hauge, and E. J. Yoffa, "Generation of performance constraints for layout," *IEEE Trans. Computer-Aided Design*, vol. 8, no. 8, pp. 860–874, Aug. 1989.

[3] C.-Y. Yeh and M. Marek-Sadowska, "Delay budgeting in sequential circuit with application on fpga placement," in *Proc. IEEE Design Automation Conference*, 2003, pp. 202–207.

[4] A. Sharma, K. Compton, C. Ebeling, and S. Hauck, "Exploration of pipelined fpga interconnect structures," in *Proc. IEEE International Symposium on Field Programmable Gate Arrays*, 2004.

[5] C. Lin and H. Zhou, "Retiming for wire pipelining in system-on-chip," in *Proc. IEEE ternational Conference on Computer-Aided Design*, San Jose, California, Nov. 2003.

[6] J. Cong, Y. Fan, G. Han, X. Yang, and Z. Zhang, "Architectural synthesis integrated with global placement for multi-cycle communication," in *Proc. IEEE ternational Conference on Computer-Aided Design*, San Jose, California, Nov. 2003, pp. 536–543.

[7] J. Cong, Y. Fan, and Z. Zhang, "Architecture-level synthesis for automatic interconnect pipelining," in *Proc. IEEE Design Automation Conference*, June 2004, pp. 602–607.

[8] L. P. Carloni and A. L.Sangiovanni-Vincentelli, "On-chip communication design: roadblocks and avenues," in *Proc. IEEE International Conference on Hardware-software Codesign and System Synthesis*, 2003, pp. 75–76.

[9] E. Bozorgzadeh, S. Ghiasi, A. Takahashi, and M. Sarrafzadeh, "Optimal integer delay budgeting on directed acyclic graphs," in *Proc. IEEE Design Automation Conference*, 2003, pp. 920–925.

[10] M. Garey and D. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*. Bell Laboratories, Murray Hill: Freeman & Co., N.J., USA, 1978.

[11] C. Lee, M. Potkonjak, and W. Mangione-Smith, "Mediabench: A tool for evaluating and synthesizing multimedia and communications systems," in *Proc. IEEE International Symposium on Microarchitecture*, 1997.

[12] M. Hall, J. Anderson, S. Amarasinghe, B. Murphy, L. Shih-Wei, E. Bugnion, and M. Lam, "Maximizing multiprocessor performance with the suif compiler," *IEEE Trans. Comput.*, vol. 29, no. 12, pp. 84–89, Dec. 1996.

[13] M. Smith and G. Holloway, "An introduction to machine suif and its portable libraries for analysis and optimization," Division of Engineering and Applied Sciences, Harvard University, Tech. Rep., 2002.