Minimum-Area Sequential Budgeting for FPGA

Chao-Yang Yeh and Malgorzata Marek-Sadowska

Department of Electrical and Computer Engineering,
University of California, Santa Barbara, CA 93106, USA

Abstract

The constraint-based approach to timing-driven placement requires delay budgeting to define the delay upper bounds for nets. While most of the previous delay-budgeting works have been focused on optimizing combinational circuits, the work in [9] introduces sequential budgeting, which combines budgeting and retiming to optimize sequential circuits better. However, the formulation in [9] does not consider flip-flop (FF) minimization, which is important in practical applications. Here, we propose a new sequential budgeting algorithm, C-SBGT, that not only controls the FF count, but also can be solved more efficiently compared to [9]. Our formulation has fewer constraints than [9] and the procedure to realize retiming is also simpler. Our experiments show that our new min-area sequential budgeting algorithm produces a good trade-off between the area and budgeting optimization goals, as well as improving the timing of previous sequential budgeting method by 12%.

1. INTRODUCTION

Placement, an important step in VLSI design, affects greatly the speed and area of circuits. Timing-driven placement is a technique to improve a circuit's speed. Several approaches for timing-driven placement have been proposed, one of which involves net budgeting [5][2][9]. For an expected clock period defined by a user, budgeting converts the path timing constraints into timing (or length) upper bounds for nets. Those upper bounds are then used to guide placement and routing. The net-lengths or delay upper bounds constitute *delay budget*.

Retiming is a procedure of relocating flip-flops (FFs) across the combinational blocks. It can be used to maximize the speed, or to minimize the chip area by reducing the total count of FFs or latches. Retiming was first proposed by Leiserson & Saxe [3]. It can be viewed as a procedure of assigning integer values to combinational blocks in a circuit. We can move FFs according to the values assigned to those blocks. The assignment of integer values implies that the smallest granularity of FF which can move is a single FF. Continuous version of retiming (C-retiming) has been described in [6]. As with conventional retiming [3], c-retiming also assigns values to the combinational blocks. The values can, however, be real numbers as opposed to integers in conventional retiming. To put it differently, c-retiming introduces *fractional FFs*. The c-retiming algorithm is very fast compared to [3], because the effort required to generate constraints is much less and the number of constraints is greatly reduced. However, for the min-area optimization goal, c-retiming can not guarantee that the total FF number is the minimum.

ICCAD'03, November 11-13, 2003, San Jose, California, USA.

Most of the previous works on delay budgeting have been focused on optimizing combinational circuits [5][2]. For sequential circuits, it is usually assumed that FFs are at fixed positions, and they are treated like primary inputs (PIs) and outputs (POs). Sequential budgeting was first proposed in [9], where the authors combine budgeting with retiming and demonstrate that results can be improved compared to a case with fixed FFs. They solve the optimization problem first and then realize retiming by moving FFs iteratively. However in [9], they were unable to control the FF population during retiming. In practice, in FPGA and ASIC designs, reducing the number of FFs usually also improves the circuit speed and reduces the power consumption. In ASICs, reducing the number of FFs also reduces the chip area. Speed, area, and power are very important from a practical standpoint. In this paper, we propose a new formulation of the min-area sequential budgeting. We apply the idea of c-retiming and combine it with sequential budgeting for FF reduction. This is one of the main contributions of our paper. Our new sequential budgeting formulation is more efficient than that proposed in previous work [9], because we reduce the number of constraints. The new procedure of realizing retiming is also simpler than that proposed in [9]. We no longer relocate FFs iteratively, but determine their positions directly. The retiming procedure also considers interconnect delays and allows for interconnect pipelining to further improve budgeting. In this paper we assume linear dependency between interconnect delay and its length.

We demonstrate the effectiveness of our algorithm in an FPGA placement flow which uses a budgeting-aware placer. The results show that our new min-area sequential budgeting formulation produces a good trade-off between the area and budgeting optimization. We improve the circuit speeds by 12% compared to the previous sequential budgeting method [9].

This paper is organized as follows. In section 2 we explain the previous sequential budgeting algorithm. In section 3 we give a brief summary of c-retiming. Then, we show an interesting relationship between the simplified sequential budgeting formulation and c-retiming. At the same time, we propose the min-area sequential budgeting. In section 4, we explain the weighting function we use in sequential budgeting and in our experiments. Section 5 shows the experimental results. Conclusions are given in section 6.

2. PREVIOUS WORK ON SEQUENTIAL BUDGETING

In [9], we have extended the conventional delay budgeting into sequential budgeting by incorporating retiming. Our approach allows us to optimize the entire circuit instead of optimizing

each combinational block individually. Incorporating retiming, the budgeting formulation has a larger solution space to explore and has more chances to obtain desired results. That formulation, T-SBGT, has two kinds of constraints. The first are the clock period constraints, which ensure that timing is satisfied; the second are budgeting constraints, which set the limits for budgeting optimization.

For the original circuit, we first create a constraint graph G. The edges in the graph represent the circuit source-sink relations, and the nodes correspond to gates. In the formulation, x_i represents the latest fan-in arrival time at a node i; D_i is a delay of a node i; and e_{ij} is an edge from the node i to j. E is a set of all the edges in G. We also create a set PS of those PI/FF-PO/FF pairs which are connected by a combinational path. If there is a combinational path from a primary input (PI) or FF i to a primary output (PO) or FF j, we include P_{ij} into PS. Υ_{ij} is a delay of the edge e_{ij}. $max(P_{ij})$ represents the maximum combinational path delay between the nodes i and j. L_{ij} is the budget lower bound on the edge e_{ij}. A user can assign the lower bound based on the results of a placement run, or based on delay predictions. $Lmax(P_{ij})$ denotes the longest path delay from i to j using L_{ij} as the net delays, rather than Υ_{ij}. Below we state this formulation:

Timing-aware Sequential Budgeting Formulation (T-SBGT) [9]: Given the clock period P, a concave function C_e and a timing constraint graph $G(V,E)$:

maximize:

$$\sum_{e_{ij}\in E}\begin{bmatrix} C_e(x_j - x_i - D_i) & if(i \notin FF) \\ C_e(x_j - x_i + P - D_i) & if(i \in FF) \end{bmatrix} \quad \text{(EQ 1)}$$

subject to:

$$x_i - x_j \le P - (D_i + max(P_{ij})),\ \forall p_{ij} \in PS \text{ if } i \in FF \quad \text{(EQ 2)}$$

$$x_i - x_j \le -(D_i + max(P_{ij})),\ \forall p_{ij} \in PS \text{ if } i \notin FF \quad \text{(EQ 3)}$$

$$x_i - x_j \le P - (D_i + L_{ij}),\ \forall e_{ij} \in E \text{ if } i \in FF \quad \text{(EQ 4)}$$

$$x_i - x_j \le -(D_i + L_{ij}),\ \forall e_{ij} \in E \text{ if } i \notin FF \quad \text{(EQ 5)}$$

$$Lmax(P_{ij}) \le max(P_{ij}),\ \forall p_{ij} \in PS \quad \text{(EQ 6)}$$

$$x_k \le P,\ k \in PO;\ x_k = 0,\ k \in PI \quad \text{(EQ 7)}$$

In (EQ1), $(x_j - x_i - D_i)$ and $(x_j - x_i + P - D_i)$ represent the budgets for e_{ij}. The concave function C_e is used to guide the allocation of delay budgets. (EQ2) and (EQ3) are the clock period constraints. The path constraints in (EQ2) and (EQ3) say that the budget assigned to a path from a PI or FF, i, to an FF or PO, j, cannot be bigger than the delay of i and the longest path delay from i to j. (EQ4) and (EQ5) are the budgeting constraints. (EQ7) states the constraints for primary inputs and primary outputs. We call them peripheral constraints. (EQ4) and (EQ5) are the budgeting constraints. Since we do not constrain the values of L_{ij} and allow those lower bounds to be bigger than the original edge delays, we need (EQ6) to make sure the longest path of edge-budget lower bounds is still smaller than the real longest path delay. Otherwise, the timing constraints in (EQ2) and (EQ3) would be violated.

The algorithm in [9] has two steps. (1) Optimize the budgeting formulation (T-SBGT) and obtain the input arrival times for the FFs. Transform the input arrival times of FFs to clock skews. (2) Use the skew-retiming equivalence relation [7] to realize the clock skew assigned to FFs.

We simplify the T-SBGT formulation by removing (EQ2), (EQ3) and (EQ6). This does not mean that we do not take edge delay or path timing constraints into account. Instead, by assigning the budget lower bound L_{ij} to Υ_{ij}, we make sure that the budget for this edge is bigger than its delay. Besides, we can enforce the path timing constraints implicitly by assigning appropriate values to L_{ij}. In brief, we use L_{ij} to replace path timing constraints in (EQ2) and (EQ3).

Now we need only constraints (EQ4), (EQ5) and (EQ7). We can further simplify constraints by not representing FFs as nodes in the graph. We can represent FFs using a weight $w(e)$ for each edge e. Originally if there are k FFs linked in a chain, there will be k edges in the graph, but now there will be only one edge with weight $w(e)$ equal to k. We use D_{ff} to represent the delay for an FF. We call the simplified formulation S-SBGT and state it below:

Simplified Sequential Budgeting formulation (S-SBGT):

Maximize:

$$\sum_{e(u,v)\in E} C_e[\, w(e)\cdot(P - D_{ff}) + x_v - x_u - D_v] \quad \text{(EQ 8)}$$

subject to:

$$x_u - x_v \le w(e)\cdot(P - D_{ff}) - D_v - L_{uv},\ \forall e(u,v) \in E \quad \text{(EQ 9)}$$

$$x_k \le P,\ k \in PO;\ x_k = 0,\ k \in PI \quad \text{(EQ 10)}$$

This simplified formulation still does not consider min-area budgeting. But, after introducing c-retiming, we will show an interesting relationship between S-SBGT and c-retiming. Then, we will explain how we consider the area minimization in the new formulation.

3. MIN-AREA SEQUENTIAL BUDGETING

In this section we will explain c-retiming and transform it into a new form, the arrival_time-based c-retiming (ACRet). We will show that ACRet and S-SBGT have similar constraints, and we can combine their objectives to obtain the min-area sequential budgeting formulation. We will also show that the new formulation can consider interconnect pipelining. In order to distinguish the Leiserson & Saxe's [3] retiming formulation from the c-retiming, we call the classical formulation in [3] d-retiming. There are two kinds of constraints in d-retiming. The first are the legitimacy constraints, which guarantee that after retiming the number of FFs on every edge is non-negative. The second kind are the clock period constraints, which guarantee that there exist more than one FF on every path with delay larger than the clock period. D-retiming assigns only integers to combinational blocks and so may be viewed as discrete *retiming*.

3.1 C-retiming

C-retiming was proposed by Pan [6]. The idea is to allow fractional FFs during retiming, changing retiming from a discrete to a continuous process. This formulation is suitable for sequential circuit synthesis with retiming, and the algorithm is very fast compared to d-retiming. C-retiming is performed in two steps. (1) Do retiming optimization using fractional FFs. (2) Transform the c-retimed circuit into a discretely retimed circuit. A drawback of the c-retiming is that it does not guarantee that the final result is of the same quality as that determined from the d-retiming.

The constraint graph used in c-retiming is similar to the graph G used in sequential budgeting. Edges represent the circuit's

source-sink relations. Nodes represent gates and combinational blocks. We no longer represent FFs as nodes. Instead, a variable $w(e)$ is assigned to each edge representing the number of FFs on it. As in T-SBGT, D_v represents delay of a node v, and P denotes the expected clock period.

Suppose $s(u)$ denotes the amount of fractional FFs retimed from the fanout of a gate u to its fanin. $w_s(e) = w(e) + s(v) - s(u)$ equals the number of fractional FFs on the edge $e(u,v)$ after c-retiming. C-retiming minimizes the count of fractional FFs subject to a given clock period P using the following linear programming formulation:

Continuous-retiming formulation (C-retiming):

minimize:

$$\sum_{e(u,v)\in E} w_s(e) = \sum_{e(u,v)\in E} w(e) + s(v) - s(u) \quad \text{(EQ 11)}$$

subject to:

$$s(u) - s(v) \le w(e) - \left(\frac{D_v}{P}\right),\ e_{(u,v)} \in E \quad \text{(EQ 12)}$$

$$s(j) = 0, \forall j \in PI;\ s(k) \le 1,\ \forall k \in PO; \quad \text{(EQ 13)}$$

(EQ12) states that the amount of fractional FFs assigned to an edge multiplied by P minus the sink gate delay is the delay budget allocated to this edge. The purpose of (EQ12) is to enforce non-negative edge weight after c-retiming. These are the legitimizing constraints. Assume that the delay from a node u to v is Q_{uv}. Let Z be a path from u to v in the circuit, then $w_s(Z) \ge (Q_{uv}/P)$, where $w_s(Z) = \sum_{e\in Z} w_s(e)$. If $Q_{uv} \ge P$, then $w_s(Z) \ge 1$. So (EQ12) also ensures that the clock period constraints in c-retiming are satisfied. To verify that the assigned clock period P is feasible, after solving the problem, we need to make sure (EQ13) is satisfied.

The number of constraints in c-retiming is equal to the number of edges in the timing graph. Those constraints can be derived by enumerating edges in the graph. This process is much simpler than that in d-retiming, where to generate clock period constraints we have to list all paths with delays larger than the clock period. Even with the pruning method as shown in [4], c-retiming is still more likely to be simpler.

After solving the above optimization problem, in the second step of the c-retiming, we have a value $s(v)$ for each gate v. We calculate $r(v)$, the *lag* in d-retiming using (EQ14). It represents the amount of FFs moved from its fanout to its fanin. Expression in (EQ14) guides the movement of FFs.

$$r(v) = \begin{cases} 0 & v \text{ is a PI or PO} \\ \lceil s(v) \rceil - 1 & \text{otherwise} \end{cases} \quad \text{(EQ 14)}$$

Note that although the authors of [6] prove that their formulation can find the minimum clock period, using c-retiming for min-area retiming purpose is just a heuristic. They cannot establish a formal relation between c-retiming and the Leiserson&Saxe's [3] min-area formulation. Although it is a heuristic, the procedure is very fast. In some applications, c-retiming is satisfactory when it is relatively unimportant whether we have the minimum FF number (e.g. in FPGA physical synthesis).

3.2 Relationship between S-SBGT and C-retiming

We can transform C-retiming formulation to an arrival_time-based c-retiming formulation (ACRet) using the variable transformation function $x_u = P \cdot s(u)$. In this new formulation, we also take interconnect delay and FF delay into account and represent them using L_{ij} and D_{ff}. We rewrite the c-retiming formulation using arrival_time-based variables in the following.

Arrival_time-based c-retiming formulation (ACRet):

$$\text{minimize: } \sum_{e(u,v)\in E} [w(e) \cdot P + x_v - x_u] \quad \text{(EQ 15)}$$

subject to:

$$x_u - x_v \le w(e) \cdot (P - D_{ff}) - D_v - \Upsilon_{uv},\ \forall e(u,v) \in E \quad \text{(EQ 16)}$$

$$x_k \le P,\ \forall k \in PO;\ x_k = 0,\ \forall k \in PI; \quad \text{(EQ 17)}$$

Comparing the constraints of S-SBGT with ACRet, we can see that they are the same, except we assign L_{ij} as Υ_{uv} in (EQ16). These variables represent latest fan-in arrival times. But, the objective functions are different. In Section 3.3, we will combine the objective functions of both formulations and share their constraints.

The *lag* function for ACRet is expressed by (EQ18):

$$r(v) = \begin{cases} 0 & v \text{ is a PI or P(} \\ \left\lceil \frac{x_v}{P} \right\rceil - 1 & \text{otherwise} \end{cases} \quad \text{(EQ 18)}$$

Note that now ACRet considers edge delays and allows for interconnect pipelining. After retiming, for each gate u, we will transform x_u again and obtain the new fanin arrival time, $G(u)$, such that $0 < G(u) \le P$. For each gate in the retimed circuit, we compute the new fanin arrival time using the function $G(u) = \lfloor x_u - (r(u) \cdot P) \rfloor$. Note that $r(u)$ is one minus the ceiling of $(x_u)/P$, so we guarantee that $0 < G(u) \le P$ for every gate u. $G(u)$ represents the expected fanin arrival time for all nodes $u \in V$ after retiming. After retiming we have $G(u)$ and $r(u)$ for each gate u, and $w(e) + r(v) - r(u)$ is the number of FFs on each edge $e(u,v)$. Now we can assign FFs to locations such that the fanin arrival times $G(u)$ assigned for all gates $u \in V$ are satisfied, and we can implement interconnect pipelining.

3.3 Optimizing the budgeting

Having shown the relation between ACRet and S-SBGT formulations, we can now combine the area and budgeting objectives. The objective of area minimization is stated in (EQ15) and the objective for budgeting is expressed by (EQ8). The combined objective function now is expressed by (EQ19).

maximize:

$$\text{(Delay budgeting goal)} - \beta \cdot \text{(Min-area goal)} \quad \text{(EQ 19)}$$

β is the area_weight. If β is larger, the optimization tries harder to reduce the FF count. The formulation using (EQ19) as the objective function and having constraints as in ACRet, will be called the continuous sequential budgeting formulation, ***C-SBGT***. Note that in our implementation for C-SBGT, we also consider maximum-fanout FF sharing using the model in [3].

We select the concave function C_e in (EQ8) using (EQ20):

$$C_e(e_{bgt}) = \alpha_e \cdot \log(e_{bgt}) \quad \text{(EQ 20)}$$

α_e is a weight of the edge e. e_{bgt} means the budgeting for that edge. In the following section, we will explain how we determine those weights. The purpose of using logarithmic function is that we want to spread the budgets out evenly among all edges. If an edge is assigned a larger weight, the budget allocated to it should be larger. The formulation can be solved efficiently using piece-wise linear approximation as shown in [8]. The dual of this formulation is a min-cost flow problem.

One advantage of using the C-SBGT compared to T-SBGT [9], is that we can control the number of FFs. Also, the number of constraints in the new formulation is smaller, because we do not require path constraints (EQ2) and (EQ3). Additionally, we can realize the movement of FFs without iterations. In T-SBGT, we move FFs iteratively across the gates or interconnects to reduce the skew on each FF to be as close to 0 as possible. Sometimes this procedure takes a large number of steps. In C-SBGT the new position of FFs can be readily computed from the x and G variables of the source and sink of each edge e, and from $w(e)$ as it was shown in section 3.2.

4. USING SEQUENTIAL BUDGETING TO DERIVE NET CRITICALITY

We can also use the new sequential budgeting formulation to derive net criticality. Net criticality is used to further improve the speed of a circuit. We derive net criticality using C-SBGT and setting β to 0 in the objective function (EQ19). In this way we consider only the budgeting optimization. As explained before, our objective function tries to spread out the budgets evenly for all edges. Since the delay budgets represent the delay upper bounds for the edges, if a budget of one edge is larger, this edge should be less critical than others. We can use the budgeting result to derive the criticality of the edges. Suppose that max_budget represents the maximum delay budget assigned for all edges. We can use the criticality function given in (EQ21).

$$Crit(e)^{-1} = \frac{budget(e)}{1.5 \times \text{max\_budget}} \quad \text{(EQ 21)}$$

In order to improve the timing result after placement, we assign smaller budgets for those nets with higher criticality. We use $Crit(e)^{-1}$ as the α_e in (EQ20) for weighting, and our new C_e is expressed in (EQ22).

$$C_e(e_{bgt}) = Crit(e)^{-1} \cdot \log(e_{bgt}) \quad \text{(EQ 22)}$$

5. EXPERIMENTAL RESULT FOR MIN-AREA BUDGETING

We perform our experiments for island architecture FPGAs, in which each table look-up (TLB) block is associated with one FF. We use a budgeting-aware placer with cost function penalizing edges whose delays are larger than their budgets. In our experiments, we use MCNC benchmark circuits [1]. We route the circuits with larger channel width than required, so that the results will be controlled by placement. For timing calculations, we assume 0.13 um technology. In our implementation of C-SBGT, we also consider maximum-fanout FF sharing. We demonstrate two experimental results. The first is to show that using min-area budgeting we can achieve a trade-off between the area and budgeting. The second experiment shows that the new budgeting formulation achieves better performance than the previously proposed sequential budgeting algorithm [9] and that the net criticality weighting described in section 4 is effective. Experimental setups are explained in each subsection.

5.1 Trade-off between FF number and budgeting

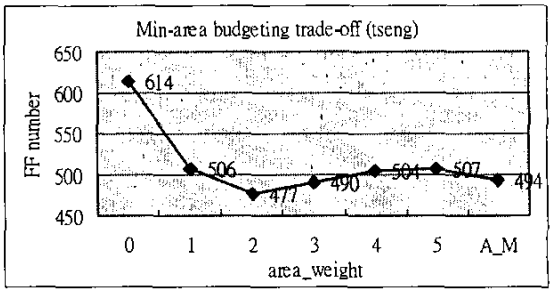

Figure 1. Trade-off between budgeting and FF number

Figure 1 shows trade-off between the FF count and budgeting on the benchmark circuit *tseng*. We have run the min-area sequential budgeting algorithm (C-SBGT) with different β to observe its effect on the FF count. Increasing the area_weight, β, in (EQ19) should result in fewer FFs. When the area_weight is 0 we only optimize budgeting. A_M on the x-axis corresponds to the case when we only optimize the area. We observe that the FFs count reaches the minimum for the area_weight of 2. Then, the FF count is in the range of 490~507. The reason that this graph is not a monotone decreasing function is that in the second step of min-area budgeting, the algorithm transforming a continuously retimed circuit into a discretely retimed one does not guarantee optimality. However, the result in this figure demonstrates that the heuristic is reasonable.

5.2 Comparing the min-area sequential budgeting with the original sequential budgeting [9]

Table 1. Experimental results

	TSB	Using Criticality						NC
		aw 0		aw 1		aw 2		aw 1
	P(ns)	P(ns)	FF#	P(ns)	FF#	P(ns)	FF#	P(ns)
tseng	13.0	15.3	682	11.1	501	11.8	496	14.3
diffeq	17.3	17.5	834	11.3	482	12.4	424	15.8
dsip	6.60	6.25	602	6.67	896	6.26	656	6.17
bigkey	7.17	6.75	603	7.55	899	7.24	869	7.36
elliptic	21.0	23.3	2066	14.9	1483	17.3	1234	20.0
s298	21.3	25.5	86	21.6	43	23.0	56	21.2
	1.00	1.06	1.00	0.88	0.92	0.91	0.84	0.99

The previous work T-SBGT [9] does not allow for the area-delay trade-off. Our experiments in Table 1 demonstrate that

C-SBGT achieves better results than T-SBGT. The column TSB lists the results using (T-SBGT) [9]. Since T-SBGT does not control the FF count, it is possible that its count increases a lot. We modified the procedure in [9] by post-processing the circuits to share FFs maximally at the fan-out edges of every node. By doing so, we reduce the FF count. The columns aw0, aw1 and aw2 under the label *Using Criticality* show timing results and *FF* count after min-area budgeting for area_weight, β, set to 0, 1 and 2 in (EQ19) and using the criticality computed as explained in section 4. To compute criticality we run the initial budgeting first with β set to 0. Then we run the second round budgeting with β equal to the value we want. The column aw1 under NC shows the result when we apply area_weight 1 and set the criticality of every edge to the same value.

The results show that the timing is best for area_weight 1 and *Using Criticality*. The timing of aw1 and aw2 are all better than those obtained from TSB. The improvement is 12% and 9%, respectively. As the area_weight becomes larger, the FF number decreases,.showing that the min-area goal is effective. When aw is 1, the FF number decreases by 8% compared to aw0. When aw is 2, the FF number decreases by 16%. The timing with net criticality is better than without it. The NC column shows that when we do not use sequential net criticality and set aw to 1, the timing improvement is only 1%. On the other hand, when using criticality (see column six) the improvement is 12%.

Note that although we use delay budgeting, the resulting clock periods for placed circuits could still be different, because delay budgets only constrain the net-delay upper bounds. The real net delay could be smaller than the budget assigned. Besides, since we use a simpler timing model, after routing, the placer will perform a more sophisticated timing calculation. This also leads to differences between the assigned budgeting-based clock period with the final clock period. Also note that the results for the benchmark *tseng* are different here from those in section 5.1. This is because we use a different placement.

6. CONCLUSION

In this paper, we have considered min-area budgeting using the concept of c-retiming. Our new formulation not only can optimize the budgeting, but also minimize the area. We analyze the relationship between the previously proposed timing-aware sequential budgeting [9] and c-retiming. We derive the *Arrival_time-based c-retiming (ACRet)* considering net delay and FF delay. Our experimental results show that we can achieve a good trade-off between budgeting and area minimization, and we can also improve the previous sequential budgeting results by 12%.

Acknowledgement. This work was supported by the California MICRO grant through Xilinx. The authors gratefully acknowledge Intel's equipment grant.

7. REFERENCE

[1] Vaughn Betz, www.eecg.toronto.edu/~vaughn/challenge/challenge.html

[2] D. Knol, G. Tellez and M. Sarrafzadeh, "A Delay Budgeting Algorithm Ensuring Maximum Flexibility in Placement", *IEEE Transactions on Computer Aided Design*, vol 16, no 11, pp 1332-1341, 1997.

[3] C. E. Leiserson and J. B. Saxe, "Optimizing Synchronous Systems", *In Journal of VLSI and Computer Systems*, pp. 41-67, 1983.

[4] N. Maheshwari and S. S. Sapatnekar, "An improved algorithm for minimum-area retiming", *Design Automation Conf.*, 1997.

[5] 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, pp 860-874, 1989.

[6] P.Pan, "Continuous Retiming: Algorithms and Applications", *International Conf. on Computer Design*, 1997.

[7] S. S. Sapatnekar, R. B. Deokar, "Utilizing the Retiming-Skew Equivalence in a Practical Algorithm for Retiming Large Circuits", *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol 15, no 10, pp 1237-1248, Oct, 1996.

[8] A. Tabbara, B. Tabbara, R. K. Brayton, A. R. Newton, "Integration of retiming with architectural floor planning", *Integration the VLSI journal*, 29, pp 25-43, 2000

[9] C.-Y. Yeh and M. Marek-Sadowska, "Delay Budgeting in Sequential Circuit with Application On FPGA Placement", *Design Automation Conf.*, 2003.