

An Area-Efficient Timing Closure Technique for FPGAs using Shannon’s Expansion

Deshanand P. Singh

Dept. of Electrical and Computer Engineering
University of Toronto
singhd@eecg.toronto.edu

Stephen D. Brown

Dept. of Electrical and Computer Engineering
University of Toronto
brown@eecg.toronto.edu

Abstract—This paper presents a technique to optimize the speed performance of circuits implemented in FPGAs. After synthesis, technology mapping and placement are complete, we apply Shannon’s expansion to the most critical sections of the circuit. This approach allows us to precompute the values of functions that depend on late-arriving critical signals and use a multiplexer to quickly select the appropriate value when the signal arrives. Any new logic elements created by this technique are incrementally placed in a minimally disruptive fashion to ensure convergence between the circuit optimization and the netlist placement. Experimental results show that this technique can improve the performance of circuits by 11% on average, and up to 30% in some cases.

I. INTRODUCTION

FPGA designers are increasingly facing the dilemma that their designs are dominated by the connection delays routed along the programmable interconnect. Traditional solutions to this problem involve improvements to automated placement and routing algorithms; however, we can now go one step further by restructuring the circuit itself to better cope with the interconnect delays.

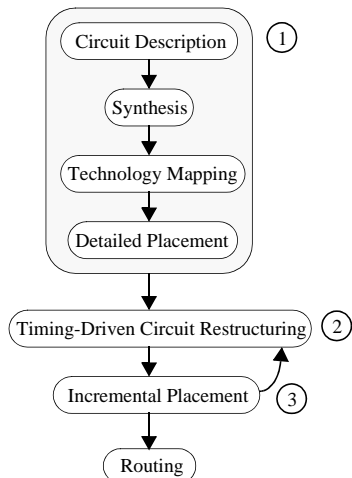


Fig. 1. Three step approach.

This alternative, and perhaps orthogonal, strategy involves tightly coupling timing-driven circuit optimizations with the placement step of the CAD flow. Within the placement phase there is still the freedom to add new elements to the netlist

of logic elements, and the routing delays can be accurately approximated for many architectures. In this manner, critical portions of the circuit can be restructured to account for the routing delays. We know of no reported work that can simultaneously optimize a general circuit and produce a legal placement that respects the many constraints that exist in modern FPGA architectures. Thus we have adopted a three-step approach to the coupling of placement with netlist optimizations as shown in Figure 1. Concepts like this three-step flow have been explored for ASIC implementations in academia [4] [6] [11] and industry [2] [5] [7].

The first step executes the conventional FPGA CAD flow of $HDL \rightarrow synthesis \rightarrow techmapping \rightarrow placement$. In the second step, routing delays for every connection are estimated by calculating their fastest possible route. Any timing-driven netlist optimization technique can then be applied to perturb the circuit to reduce the critical path(s). The estimation and optimization techniques are collectively referred to as *layout-driven* optimizations since the layout of the logic elements directly affects how the circuit is perturbed. In this paper, we will focus on the use of Shannon’s expansion as our netlist optimization technique, although other optimizations such as *sequential retiming* have been used in this flow [10]. Every additional logic element introduced in the circuit is given a preferred placement location. These preferred locations ignore even the most fundamental architectural constraints of the programmable device under consideration but are chosen strictly on the basis of improving timing. These preferred locations could form an illegal placement, but we rely on a subsequent fixup stage to remove the illegalities. This concept relaxes the problem of simultaneously optimizing a circuit and producing a legal placement. Our netlist optimization techniques need only restructure the netlist and produce an approximately legal “preferred” placement. Our experimental results will show that the use of these preferred locations is key to producing a quality result.

The final step of our three-step flow occurs after the preferred locations have been generated. The job of the incremental placement (ICP) engine is to perturb the preferred locations as little as possible to ensure that the final placement respects all FPGA architectural constraints. These constraints include features such as the limited number of logic elements/inputs/secondary signals per clustered logic block. A

detailed description of the incremental placement algorithm used in this study is presented in [9]. For the purposes of this paper, ICP can be abstracted as an algorithm that takes in a possibly illegal placement described by the preferred locations, and legalizes the solution while disrupting the length of the critical path and total wirelength as little as possible. Ideally, the netlist contains a number of non-critical logic elements that can be moved from their preferred locations to resolve illegalities, while truly critical elements stay at their preferred locations for delay and area reasons.

The remainder of this paper is organized as follows. First we describe Shannon’s decomposition theorem and show how it can be applied to arbitrary critical portions of logic. We then motivate the reasons that Shannon’s expansion should be applied post placement. Next we examine methods to select the groups of logic where Shannon’s Decomposition can be effectively applied. Finally, we detail an iterative application of this decomposition technique, its integration with the incremental placement algorithm and present results on a number of large benchmark circuits.

II. SHANNON’S THEOREM

Consider an n -input function $f(x_0, x_1, \dots, x_i, \dots, x_n)$. Shannon’s Decomposition Theorem [8] allows us to express the function as

$$f(x_0, \dots, x_i, \dots, x_n) = \bar{x}_i f(x_0, \dots, 0, \dots, x_n) + x_i f(x_0, \dots, 1, \dots, x_n) \quad (1)$$

This means that we can precompute the function values for $x_i = 0$ and $x_i = 1$ and then select the appropriate value depending on the value of x_i .

This theorem has powerful implications for restructuring critical portions of combinational logic after placement. Figure 2 shows an example of the realization of Shannon’s Theorem. This example shows a simple logic function, with

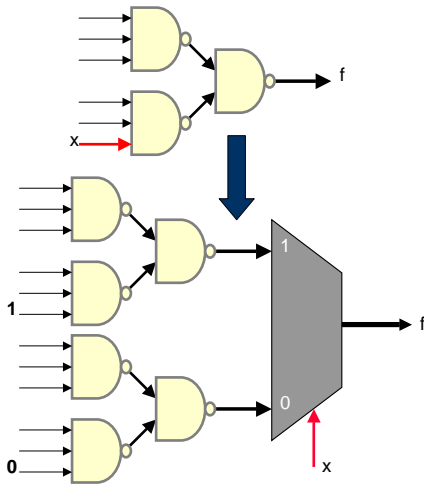


Fig. 2. Physical realization.

a single late arriving input x . Most likely, the signals used to generate x incur significant routing delays or x itself may

be located far from its fanout logic. Shannon’s decomposition allows the tradeoff of the number of levels of logic that must be experienced in paths that are influenced by the signal x for extra area. In this example instead of going through 2 levels of logic, the paths emanating from x now only experience one level of logic delay. Hence, we can compensate for long routing delay components in certain paths by reducing the amount of logic delay on these paths. Notice that by reducing the depth of the signals emanating from x , many other paths may have been increased by an additional logic level. This is the tradeoff that we have to consider when applying Shannon’s theorem.

III. POST-PLACEMENT OPTIMIZATION

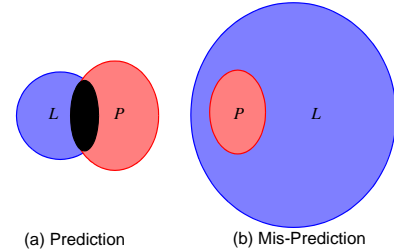


Fig. 3. Prediction and Mis-Prediction rates.

The application of Shannon’s Expansion is performed after placement so that the circuit can cope with routing delays. One of the natural questions concerning this techniques is: how necessary is it that they be applied as a post-placement step rather than as part of LUT-level synthesis? Our implementation of Shannon’s Expansion relies heavily on finding critical sections of logic. Here we show that it is extremely difficult to predict the post-placement critical connections from a LUT-level netlist. In this experiment, we refer to a connection as being *critical* if its criticality is greater than 0.9. Two different statistics are considered. These are depicted graphically in Figure 3(a). The first is the *prediction rate*. This is the ratio of connections that are critical in both the LUT-level netlist (highlighted in blue in Figure 3) and the post-placement netlist (highlighted in red) divided by the number of critical connections in the post-placement netlist. The critical connections in the LUT-level netlist are computed by only considering the logic delay and assuming 0 delay routing connections. If the prediction rate is low, then the circuit restructuring techniques may be applied to the wrong paths and not help the true critical paths that exist post-placement. This statistic attempts to quantify how much of the timing critical region of the circuit can be predicted before the place and route process. Mathematically, this can be expressed in the following manner:

- Let L represent the critical connections in the LUT-level netlist.
- Let P represent the critical connections in the post-placement netlist.

$$PredictionRate = |P \cap L|/|P| \quad (2)$$

We often find that the prediction rate approaches 100% for some circuits, but this is because of the situation shown in Figure 3(b). In this case the LUT-level netlist predicts too many connections as being critical. These mis-predictions are troublesome because it means that we will apply our optimization techniques to far too many regions of logic that end up being non-critical after placement, and perhaps wasting area to improve portions of logic that aren't really critical. In addition, mis-prediction also hurts our algorithm because it essentially moves delays from critical to non-critical paths. Since many non-critical paths are considered critical at the LUT-level, this tradeoff may not be possible. The mis-prediction rate can be expressed as follows:

$$MisPredictionRate = |\bar{P} \cap L|/|P| \quad (3)$$

TABLE I
PREDICTION RATES.

Circuit	#LUT	Predict%	Mis-Predict%
bigkey-mcnc	1707	100%	3675%
dsip-mcnc	1370	100%	858%
diffeq-mcnc	1497	100%	59.3%
elliptic-mcnc	3604	0%	15.2%
frisc-mcnc	3556	57.8%	7.32%
s38417-mcnc	6406	94.2%	5703%
tseng-mcnc	1047	97.3%	27.2%
hc11-oc	3877	94.3%	15%
des-fip	15509	0%	329%
sisc8	1434	66.1%	0.41%

Table I shows these statistics for a cross-section of benchmark circuits. Generally the prediction rate is either extremely low, or the mis-prediction rate is extremely high. A circuit that exhibits either of these characteristics may prove challenging to apply restructuring techniques at the LUT-level. It is these results that motivate the optimization of critical paths after placement has been completed.

IV. DEFINITIONS

The algorithms described in subsequent sections rely heavily on being able to manipulate critical regions of logic. This section presents definitions that will allow us to easily express how critical regions of logic can be manipulated.

First the most critical nodes and connections in the netlist must be established. These critical elements can be represented by an ϵ -graph which contains elements that are within ϵ of being critical. The value of ϵ is typically a small number that determines the threshold of what logic is considered critical. If ϵ is too large then Shannon's decomposition will be applied to many nodes that don't affect the path. The net effect is simply a waste of area. If ϵ is too small then decompositions are applied to the critical path, but are not applied to any of the near-critical paths.

Recall that a combinational circuit, or the combinational portion of a sequential circuit, can be represented by a directed acyclic graph $G(V, E)$. Each vertex in the graph corresponds to a combinational element in the corresponding circuit. The

edges represent connections between the combinational elements. The ϵ -graph $G_\epsilon(V_\epsilon, E_\epsilon)$ is a subgraph of the combinational graph $G(V, E)$. The edge set E_ϵ is defined as the set of edges that are within ϵ of being critical

$$E_\epsilon = \{\forall e \in E \mid crit(e) \geq 1 - \epsilon\} \quad (4)$$

The critical vertices V_ϵ are the set of all vertices that have any adjacent edge which is within ϵ of being critical.

$$V_\epsilon = \{\forall v \in V \mid (\exists e_{uv} \text{ st. } crit(e) \geq 1 - \epsilon)\} \cup \{\forall v \in V \mid (\exists e_{vu} \text{ st. } crit(e) \geq 1 - \epsilon)\} \quad (5)$$

By construction, $E_\epsilon \subseteq E$ and $V_\epsilon \subseteq V$.

V. APPLYING SHANNON'S THEOREM

Consider a critical late-arriving signal x that we wish to transform such that signals affected by x incur less delay due to logic elements. Assume that $x \in V_\epsilon$ since we are applying the decomposition techniques to critical signals. The first step is shown in Figure 4. We find all critical vertices

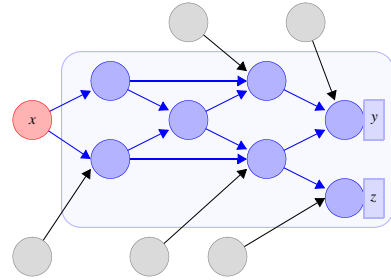


Fig. 4. Finding the Transitive Fanouts.

that are affected by the signal x . These vertices can be found by examining the transitive fanouts of x in the subgraph G_ϵ . A vertex v is a transitive fanout of x if there exists a path from $x \rightarrow v$ in the subgraph G_ϵ . This operation is denoted $v \in TF_\epsilon(x)$. In Figure 4, the transitive fanouts of x in G_ϵ are encapsulated in the rounded rectangle. The signal x affects two critical signals $y(\dots, x, \dots)$ and $z(\dots, x, \dots)$. Shannon's decomposition can now be applied to these two functions.

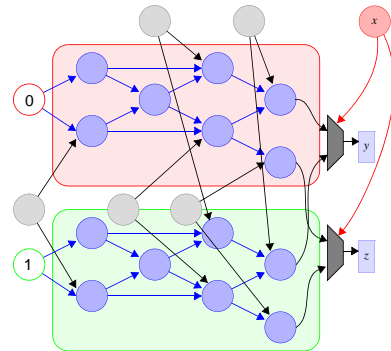


Fig. 5. Duplication of the Transitive Fanouts.

Following the identification of the transitive fanout vertices, the next step is to create two duplicate copies for each of these

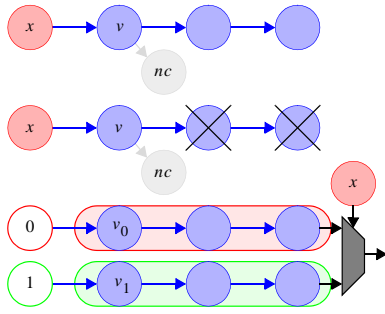


Fig. 6. Finding the vertices to delete

vertices. One set of these vertices will be used to evaluate y and z for $x = 0$, while the second set will evaluate these functions for $x = 1$. Consider a vertex $v \in TF_\epsilon(x)$. A duplicated version is needed to evaluate for $x = 0$; it is denoted v_0 . Similarly to evaluate for $x = 1$, the duplicated vertex is denoted v_1 . This duplication operation is shown in Figure 5. New edges must also be added to wire the appropriate combinational cells together. For each vertex $v \in TF_\epsilon(x)$ the following procedure is used to create new edges. For every edge $e_{uv} \in FANIN(v)$:

- if $u \in TF_\epsilon(x)$, then create two new edges from $u_0 \rightarrow v_0$ and $u_1 \rightarrow v_1$. This wires together the intermediate signals used to compute y and z for $x = 0$ and $x = 1$.
- if $u \notin TF_\epsilon(x)$, then create two new edges from $u \rightarrow v_0$ and $u \rightarrow v_1$. This procedure is used to wire the input signals into the logic that computes $y(\dots, 0, \dots)$, $y(\dots, 1, \dots)$, $z(\dots, 0, \dots)$ and $z(\dots, 1, \dots)$

The next step in the application of Shannon's Decomposition is to add multiplexer logic to select from the cofactors. The multiplexer is controlled by the critical signal x on which the decomposition was based.

The final step is to remove any vertices that have become unused. Recall that each vertex $v \in TF_\epsilon(x)$ is duplicated into two copies v_0 and v_1 . Obviously v_0 and v_1 are used in place of v so the natural question is: can v be deleted? In Figure 5, the non-duplicated logic is no longer needed because the duplicated versions serve to produce the functions y and z . However, if any of these vertices $v \in TF_\epsilon(x)$ was used as an input to another function $nc(\dots, v, \dots)$ such that $nc \notin TF_\epsilon(x)$ then the vertex v cannot be deleted. This situation is depicted in Figure 6. It occurs because we apply Shannon's Decomposition to the ϵ -graph which is a subgraph of the entire combinational circuit. Hence non-critical fanouts of v still require the signal produced by v . This could easily be accomplished by keeping the original vertex v or by using $\bar{x}v_0 + xv_1$. Keeping the original vertex is preferable as it does not affect the number of levels of logic element delay sent to the non-critical fanouts. Thus a vertex $v \in TF_\epsilon(x)$ is deleted if and only if $TF(v) \subseteq TF_\epsilon(v)$.

VI. SELECTING LOGIC TO EXPAND

Although it would be possible to apply the techniques described in the previous section to any critical vertex in

G_ϵ , a method is needed to try to minimize the amount of duplication produced and to ensure that the decomposition techniques actually provide timing improvement. These goals are achieved with a cost function that evaluates the suitability for applying the decomposition and duplication techniques to groups of logic.

The first part of the cost function attempts to quantify the number of critical or near critical paths that a particular vertex affects. This quantity is denoted with a label named $cpcount$ for each vertex. These identifiers are all initially set to zero $\forall v \in V_\epsilon, cpcount(v) = 0$.

Figure 7 graphically depicts the algorithm used. Each path highlighted in red represents a single critical path traced backward from a sink vertex. The $cpcount$ value of a vertex, v , is incremented whenever the traceback goes through v . In this example there are two nodes that affect 4 critical sinks. If decomposition and duplication techniques are applied with respect to these nodes, then it may be possible to quickly reduce a number of paths by applying the techniques in the previous section. This algorithm is clearly heuristic in nature

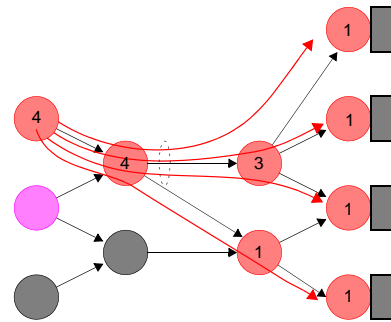


Fig. 7. Finding the Most Critical Nodes.

as there may be several different near critical paths that affect a sink vertex, instead of the single path that is traced backwards.

After a cost has been applied to each vertex, it is still necessary to check if the decomposition can actually provide any gain before applying it. Consider the situation shown in Figure 8. We wish to perform the decomposition and duplication techniques with respect to the signal x . While this technique is beneficial for signals emanating from x , we have not considered the side effects of this operation. Consider the fanins labeled i_0, i_1, i_2, i_3 in the figure. After the application of decomposition and duplication, signals downstream from these connections experience an extra level of logic delay due to the multiplexer logic added at the sink nodes. This extra level of logic delay is acceptable as long as the slack on the connections $i_0 \dots i_3$ is greater than the amount of delay introduced by the multiplexer logic and the routing delay needed to wire the logic to the mux. Thus, these paths must be considered before applying our techniques.

More formally, the side effects can be checked in the following manner. Let side-input set I represent the fanin edges of $TF_\epsilon(x)$ whose source vertex is not an intermediate

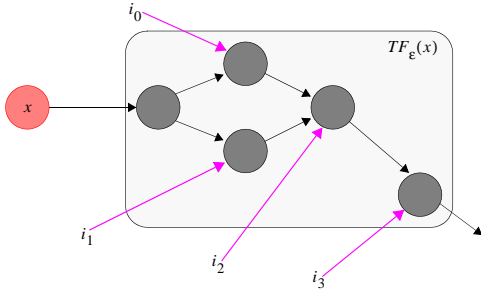


Fig. 8. Checking Critical Fanins.

variable in $TF_\epsilon(x)$.

$$I = \{\forall e_{uv} \in FANIN(TF_\epsilon(x)) \mid u \notin TF_\epsilon(x)\} \quad (6)$$

Simply stated, I contains all external input edges to $TF_\epsilon(x)$. The decomposition and duplication technique can be applied as long as the following condition holds:

$$\forall i \in I, \text{slack}(i) \geq \text{mux delay} + \text{routing delay to mux} \quad (7)$$

VII. CONTROLLING DUPLICATION

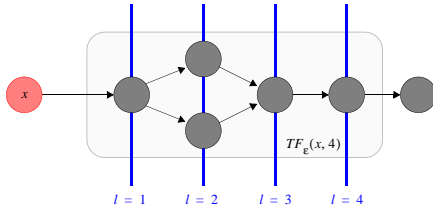


Fig. 9. Controlling Vertex Duplication.

The procedures described so far duplicate each vertex in $TF_\epsilon(x)$. However, this set may be arbitrarily large. The set-size is controlled by redefining the critical transitive fanout set. Each vertex in $v \in TF_\epsilon(x)$ is associated with a label $l(v)$ that is set to the maximum number of logic levels between x and v . The set $TF_\epsilon(x, D)$ represents all vertices v where there exists a path from $x \rightarrow v$ and $l(v) \leq D$. The value of D controls the tradeoff between the amount of duplication allowed and the number of levels of logic delay removed from critical paths. Note that the labels $l(v)$ can be computed using a breadth-first search from vertex x .

All of the procedures described previously that use $TF_\epsilon(x)$ can easily be replaced by $TF_\epsilon(x, D)$. Experiments have shown that this strategy is quite effective as values of $D = 2$ or $D = 3$ effectively optimize the critical-path delay while adding a small number of extra logic elements. The reduction in the number of logic elements has the added benefit of making the job of the incremental placer easier as it has to resolve fewer architectural constraints.

VIII. INTEGRATION WITH ICP

Each of the fundamental operations described previously must also be tightly integrated with placement. In this way, we can accurately make decisions about critical regions of logic

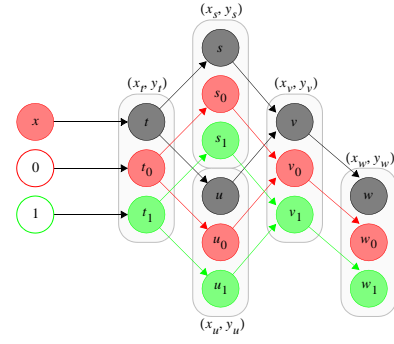


Fig. 10. Setting the preferred locations.

```

procedure optimize(  $G(V, E)$  )
begin
  do
    VerticesAdded = 0;
  do
     $G_\epsilon = \text{TimingAnalyzer}(G)$ 
     $\forall v \in \text{sinks}(V_\epsilon)$ , call traceback(  $v$  );
    select the “best” node  $x$  with the largest cpcount( $x$ )
      and large side-input slack using sliding window
    if  $x = \emptyset$  then exit-loop;
    apply decomposition and duplication w.r.t.  $x$ ;
    VerticesAdded += VerticesDuplicated -
      VerticesDeleted;
    if VerticesAdded > AvailableLUTs then exit-loop;
  end loop
  if VerticesAdded = 0 then exit-loop;
  call IncrementalPlacement( );
end loop
end procedure

```

Fig. 11. Iterative Application.

that can be optimized by the decomposition and duplication techniques. Figure 10 shows an example of how the preferred locations are derived. The vertices s, t, u, v, w are duplicated into two sets to precompute for $x = 0$ and $x = 1$. The preferred location of a given vertex v_0 is simply set to the current location of v . Similarly the preferred location of v_1 is also set to the current location of v . Recall that FPGA logic blocks typically consist of clusters of several logic elements for each physical cluster location. Hence it is possible for vertices v, v_0, v_1 to exist at the same physical location. Hopefully ICP can shift non-critical logic to other locations if there is not enough space.

IX. ITERATIVE APPLICATION

Figure 12 shows a graphical representation of how vertices are chosen for the application of Shannon’s Decomposition. The “best” nodes are chosen using a linear cost function given

TABLE II
DECOMPOSITION AND DUPLICATION RESULTS.

Circuit	Size(#LUTs)	T_P (ns),%Speedup Shannon+ICP	T_P (ns),%Speedup Shannon+Replace	Area Increase%
bigkey-mcnc	1707	7.68 (+7.8%)	8.24 (+0.5%)	0.35%
dsip-mcnc	1370	8.17 (0%)	8.17 (0%)	0%
diffeq-mcnc	1497	17.9 (0%)	17.9 (0%)	0%
elliptic-mcnc	3604	18.5 (+24%)	22.2 (+3.1%)	9.41%
frisc-mcnc	3556	29.4 (0%)	29.4 (+0%)	0%
s38417-mcnc	6406	18.3 (+8.5%)	19.9 (-0.2%)	0.64%
tseng-mcnc	1047	17.0 (+3%)	16.3 (+7.2%)	1.43%
hc11-oc	3877	34.9 (+13%)	36.6 (+8%)	0.52%
des-fip	15509	13.0 (+34.7%)	17.7 (-1%)	0.17%
sisc8	1434	16.9 (+12.9%)	17.5 (+9.1%)	18.41%
clma-mcnc	8383	27.7 (+12.6%)	30.7 (+1.5%)	1.06%
ex1010-mcnc	4598	21.9 (+10.1%)	25.4 (-5.3%)	0.33%
spla-mcnc	3690	18.7 (+6.9%)	21.9 (-8.7%)	0.24%
pd-mcnc	4575	16.9 (+12.9%)	22.9 (-0.1%)	0.07%
Average		+10.8%	+1.2%	+2.33%

by:

$$cost(x) = k_{cp} * cpcount(x) + \min_{i \in I} slack(i) \quad (8)$$

It simply states that we search for nodes with high *cpcount* along with large side-input slacks, so that the algorithm does not end up thrashing by fixing critical paths and then creating new ones. Searching for these nodes is very much like moving a small sliding window across the netlist that encompasses nodes and their transitive fanouts.

Figure 11 shows how circuit performance can be optimized through the iterative application of the techniques described earlier. Notice that the routine contains two main loops. The inner loop controls the iterative application of Shannon’s expansion. It is continued until the chip is full or there is no longer any benefit from applying Shannon’s theorem.

The next step is a call to the incremental placement routine that attempts to map the preferred locations into legal physical locations on the target device. If the incremental placement routine succeeds, then the decomposition and duplication loop is attempted again because the new placement may perturb the timing characteristics of the circuit sufficiently so that decomposition may again be beneficial.

Briefly, our implementation also includes area recovery steps such as constant propagation and incremental LUT mapping to collapse the multiplexer and logic that feeds it into a single lookup table.

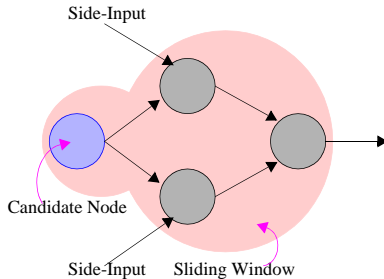


Fig. 12. Sliding Window search.

X. RESULTS

Our experiments involved using FlowMap [3] to map circuits to a netlist of 4-input lookup tables. The timing-driven VPR [1] tool was used to place and route the design. Each circuit was mapped into a device such that the chip was not more than 90% utilized. The routing architecture was also chosen such that there are 20% more tracks than the minimum needed to route the circuit. This ensures that the device is not so congested that delays cannot be predicted from netlist placement.

Table II shows the results of the application of the techniques described in this paper after placement. The third column shows the speedup vs. the result obtained through the conventional CAD flow. As shown, the average speedup is 10.8%. If the incremental placement is replaced with a complete re-placement, the speedup is reduced to only 1.2%. In this case, there are no preferred locations. An optimized netlist is simply passed to the VPR and another complete placement and routing step is executed. This result is not unexpected, because there is no reason to expect that the optimized netlist will correspond to the new placement. Most practical annealing algorithms produce very different results even with small differences in their initial solution. For both of these experiments, the area penalty was 2.33%.

XI. SUMMARY

This paper has described a physical netlist optimization based on Shannon’s Decomposition Theorem. This technique in concert with incremental placement algorithms described previously produce average speedups of 11% and as much as 30%. As the complete re-placement flow shows, even with complete knowledge of criticalities, the flow of netlist optimization followed by complete placement cannot compete with our incremental approach. In addition, by focusing only on the timing critical logic elements, our average area penalty is extremely small.

REFERENCES

- [1] V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.
- [2] Cadence. *Cadence Physically Knowledgeable Synthesis*. http://www.cadence.com/datasheets/dat_pdf/pks101801.pdf.
- [3] J. Cong and Y. Ding. FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *IEEE Transactions on CAD*, Jan 1994.
- [4] Y. Jiang, A. Krstic, K. Cheng, M. Marek-Sadowska. Post-Layout Logic Restructuring for Performance Optimization *DAC*, 1997.
- [5] L. Kannan, P. Suaris and H. Fang. A Methodology and Algorithms for Post-Placement Delay Optimization. *DAC*, 1994.
- [6] Y. Lian and Y. Lin. Layout-based Logic Decomposition for Timing Optimization. *ASPDAC*, 1999.
- [7] Mentor Graphics. *Physical Implementation with: TeraPlace, TeraOptimize, TeraCTS*. http://www.mentor.com/teraplace/teraplace_ds.pdf.
- [8] C. Shannon. The Synthesis of Two-Terminal Switching Circuits. *Bell Systems Technical Journal*, 28(1), 1949.
- [9] D. Singh and S. Brown. Incremental Placement for Layout-Driven Optimizations on FPGAs. *ICCAD 2002*.
- [10] D. Singh and S. Brown. Integrated Retiming and Placement for FPGAs. *FPGA 2002*.
- [11] G. Stenz, B. Riess, B. Rohfleisch and F. Johannes. Timing Driven Placement in Interaction with Netlist Transformations. *ISPD*, 1997.