Post-Placement Functional Decomposition for FPGAs

Valavan Manohararajah, Deshanand P. Singh,
Stephen D. Brown and Zvonko G. Vranesic
Altera Toronto Technology Center
151 Bloor Street West, Suite 200
Toronto, Ontario, CANADA
vmanohar|dsingh|sbrown|zvranesi@altera.com

ABSTRACT

This work explores the effect of adding a simple functional decomposition step to the traditional field programmable gate array (FPGA) CAD flow. Once placement has completed, alternative decompositions of the logic on the critical path are examined for potential delay improvements. The placed circuit is then modified to use the best decompositions found. Any placement illegalities introduced by the new decompositions are resolved by an incremental placement step. Experiments conducted on Altera's Stratix chips indicate that this functional decomposition technique can provide a performance improvement of 7.6% on average, and up to 26.3% on a set of industrial designs.

1. INTRODUCTION

Recent research effort in *physical synthesis* has strived to eliminate the artificial separation that exists between the various steps in CAD. Most of the existing work is applicable to ASIC CAD flows [2, 4, 5, 6, 7, 8]. However, a few of the more recent efforts explore FPGA CAD flows [3, 9, 10]. Some have concentrated on making the synthesis step more aware of what happens during placement and routing [2, 3], while others have explored the use of synthesis type algorithms during placement and routing [4, 5, 6, 7, 8, 9, 10]. Our work falls into the latter category. It considers the effect of a simple functional decomposition algorithm that is used after placement.

Most of the delays in an FPGA circuit are due to the programmable routing network [1]. These delays will not be known for certain until the routing step completes. It would be advantageous to perform local optimizations once routing has completed and accurate routing delays are available. However, making changes to the circuit during the routing step is extremely complicated. Here we choose to perform local optimizations at the placement step, which is sufficiently close to the routing step that reasonably accurate delays are known. Furthermore, small changes to the circuit can still be made without much difficulty.

2. THE STRATIX ARCHITECTURE

Altera's Stratix chips were used as the target device for the experiments. At the highest level, the chip is made up of *LABs*, memory elements and DSP blocks that are connected to a programmable routing network. Each LAB contains a set of *LEs* which are the basic logic elements in the Stratix architecture. An LE consists of a four-input lookup table (4-LUT) and a programmable register. The construction of arithmetic circuits is facilitated by the presence of *carry chain* circuitry that links adjacent LEs together. The LEs also contain *cascade chain* circuitry that can be used to link registers in adjacent LEs.

We do not consider memory elements and DSP blocks any further, as our work applies only to LABs. A detailed description of the internal structure of the Stratix chips can be found in [11].

3. A FRAMEWORK FOR POST-PLACEMENT OPTIMIZATION

Figure 1 illustrates the CAD flow used in our work. In the first step, design entry, the design is described in terms of a hardware description language such as VHDL or Verilog. Logic synthesis optimizes the circuit obtained from design entry. During logic synthesis the netlist is represented in terms of a generic gate library. The technology mapping stage converts the netlist so that it uses logic elements available in the target FPGA architecture. In order to reduce the size of the problem the placer has to deal with, a clustering step is used to decompose the technology mapped circuit into a set of clusters. In the Stratix architecture, the clustering step creates a set of LABs. Following clustering, placement determines a position for each cluster in the circuit.

Once placement is completed, various local optimization techniques are used to improve the circuit's critical path. Functional decomposition is one of the many local optimization techniques that can be used during this step. The local optimization techniques may make changes that result in an invalid placement. For example, a functional decomposition algorithm may create new wires that violate the constraint on the number of wires entering a LAB. A logic duplication algorithm may create new LEs which would then require placement. Incremental placement is used to integrate the

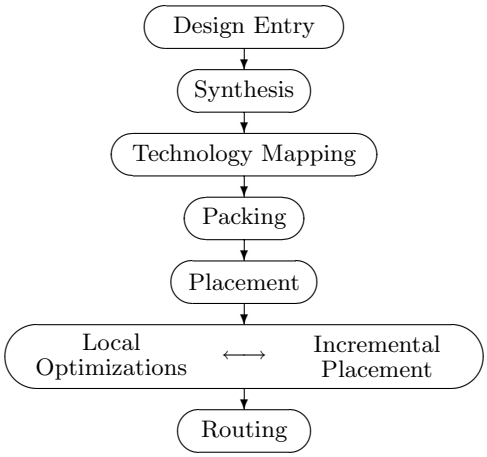

Figure 1: Post-placement optimizations in the CAD flow.

changes made by the local optimization techniques into the existing placement. It uses a greedy algorithm whose goal is to integrate the changes while making a minimum of change to the existing placement. A detailed description of the incremental placement step can be found in [12].

The final step in the CAD flow, routing, determines the wires that will be used to connect the elements that make up the circuit.

4. FUNCTIONAL DECOMPOSITION

4.1 Overview

An overview of the functional decomposition algorithm is presented in Figure 2. Timing analysis is the first major task performed by the algorithm. This analysis is only approximate as actual wire delays are not available until the routing step has completed. The analysis computes wire delay assuming that the best possible routes will be used for each connection in the circuit. Timing analysis establishes both the length of the critical path as well as the *slack* [16] for each sink. Slack is defined to be the amount of delay that can be added to a sink before it becomes critical. A slack of 0 indicates that the sink is on the critical path.

The algorithm performs a number of decomposition iterations until a user specified maximum is reached. Each iteration attempts to find alternative decompositions of the logic around the *near-critical* sinks returned by CriticalSinks. A near-critical sink is one that has a slack below some threshold. In the experiments, the threshold was set to a multiple of the average delay through an LE. For each near-critical sink, s, BestDecomp is called to determine the best possible decomposition. The function examines both the LE that drives the sink as well as the LE that is driven by the sink. Alternative decompositions exist if both LEs are being used as 4-LUTs. Figure 3 illustrates the situation. The function considers every possible decomposition of the 7-input function $f(i_1, i_2, i_3, i_4, i_5, i_6, i_7)$ formed by the two 4-LUTs, A and B, connected by the near critical sink s. The best decomposition, as determined by a cost function (see Section 4.4), is then returned.

```
1:  initt, bestt ← TimingAnalysis()
2:  initc, bestc ← circuit
3:  for i ← 1 upto MaxIterations
4:      D ← ∅
5:      for s ∈ CriticalSinks()
6:          D ← D ∪ BestDecomp(s)
7:      end for
8:      for d ∈ Sort(D)
9:          DoDecomp(d)
10:     end for
11:     t ← TimingAnalysis()
12:     if t ≤ bestt then
13:         bestt ← t, bestc ← circuit
14:     end if
15: end for
16: circuit ← bestc
17: IncrementalPlacement()
18: if TimingAnalysis() > initt then
19:     circuit ← initc
20: end if
```

Figure 2: An overview of the functional decomposition algorithm.

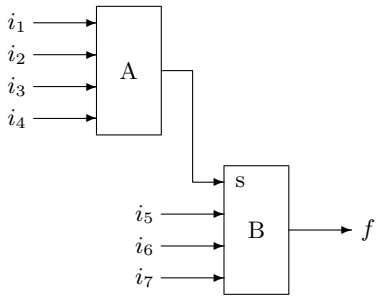

Figure 3: Two cascaded LEs.

	$i_1i_2i_3$							
$i_5i_6i_7$	000	001	010	011	100	101	110	111
000	1	1	1	1	1	1	1	1
001	0	1	1	0	1	1	1	1
010	0	1	1	0	1	1	1	1
011	1	1	1	1	1	1	1	1
100	0	0	0	0	0	0	0	0
101	1	0	0	1	0	0	0	0
110	1	0	0	1	0	0	0	0
111	0	0	0	0	0	0	0	0

Figure 4: A 6-variable decomposition chart.

$i_1i_2i_3$							
000	001	010	011	100	101	110	111
0	1	1	0	1	1	1	1

Figure 5: A function for LE A.

Tests for decomposability and the generation of new functions for A and B are handled by standard textbook methods [13, 14]. Figure 4 illustrates a 6-variable decomposition chart. The variables feeding LE A (*bound set*) are arranged along the columns and the variables feeding LE B (*free set*) are arranged along the rows. A decomposition exists as long as there are no more than two distinct column patterns in the chart (*column multiplicity* of two). Functions for LE A and LE B can be derived from the chart as well. A function that selects between the two distinct columns is implemented in A and a function that outputs the required column is implemented in B. Encoding the first column type as 0 and the second column type as 1, a function for LE A is obtained. An example of this function for the chart of Figure 4 is denoted f_A and is illustrated in Figure 5. The function for LE B, presented in Figure 6, uses f_A to select between the two distinct columns in Figure 4.

Once all near-critical sinks have been examined, the set of decompositions, D, is ordered from best to worst. This ordering is necessary because the decompositions present in D are not mutually exclusive — some decompositions interfere with others. For example, consider a set D containing two decompositions, one involving LE A and LE B and one involving LE B and LE C. The B-C decomposition can potentially interfere with the A-B decomposition if the A-B decomposition is no longer valid when the B-C decomposition is applied. The ordering ensures that when interference does occur, a higher ranked decomposition takes precedence over a lower ranked one.

	f_A	
$i_5i_6i_7$	0	1
000	1	1
001	0	1
010	0	1
011	1	1
100	0	0
101	1	0
110	1	0
111	0	0

Figure 6: A function for LE B.

The decompositions in set D are applied in sorted order. As the decompositions are applied in DoDecomp, the LUTs involved in the decomposition are marked so that other decompositions which involve the same LUTs will be skipped. Again, this is done to prevent problems where one decomposition interferes with another.

Once the best decompositions have been applied to the circuit, a call to timing analysis determines the effect of the changes on the critical path. This timing analysis will involve additional approximations as the changes made to the circuit have not yet been integrated into a legal placement. The analysis assumes that the modified LEs remain in their original positions. This assumption does not introduce a significant error because the changes made by the decomposition algorithm usually involve wire swaps between LEs, which are almost always legal.

The best circuit seen during the decomposition iterations, *bestc*, will be made the default circuit before calling IncrementalPlacement to integrate the changes. A final call to timing analysis (line 18) determines if the final placed circuit is an improvement over the initial circuit. The modified circuit is retained only if there is an improvement.

4.2 Special Cases

There are a variety of situations that warrant special case handling during decomposition. Some of these situations arise due to the structure of the Stratix LE. Consider the situation depicted in Figure 3. When LE A is used in arithmetic mode, alternative decompositions do not exist because the LUT inputs are used to compute a hidden carry function that is not connected to the routing network. Any changes to the inputs of A will modify the hidden carry function. A similar problem exists when the Stratix LE's programmable register is used. If the output of A is registered, then usable decompositions do not exist because any change to A's functionality will result in a wrong value being registered.

Another situation that requires special handling occurs when the output of A in Figure 3 fans out to more than one sink. Once again the functionality of A cannot be changed without changing the functionality seen at all sinks. However, if we duplicate A and attach every sink except s to the duplicate, the decomposition involving A can still be performed. Although this duplication allows the decomposition to occur, it may create area problems in the LABs containing A and B. If the LABs containing A and B are nearly full, the cost function (see Section 4.4) computes the slack improvements for the decomposition assuming that the version of LE A feeding LE B will be moved to a neighboring LAB.

4.3 Nondisjoint Decomposition

The decomposition function, BestDecomp, assumes that the inputs to A and B in Figure 3 are unique. However, there are cases where the best implementation is achieved using nondisjoint decomposition. Figure 7 gives a cost-efficient circuit for the 4-to-1 multiplexer. Inputs s_0 and s_1 select

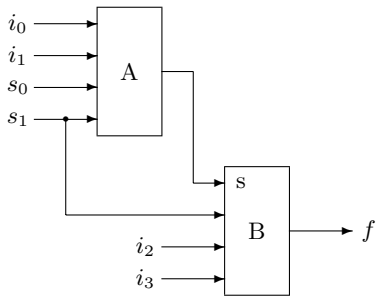

Figure 7: A 4-to-1 multiplexer using two LEs.

one of the four inputs i_0, i_1, i_2 and i_3 to appear at the output. LUT A implements the function,

$$f_A = \overline{s}_1(\overline{s}_0 i_0 + s_0 i_1) + s_1 s_0$$

and LUT B implements the function,

$$f_B = \overline{s}_1 f_A + s_1(\overline{f}_A i_2 + f_A i_3).$$

An alternative decomposition does not exist if the inputs to A and B are assumed to be unique. However, the knowledge that input s_1 is shared allows a non-disjoint decomposition such as

$$\begin{aligned} f_A &= s_1(\overline{s}_0 i_2 + s_0 i_3) + \overline{s}_1 s_0 \\ f_B &= s_1 f_A + \overline{s}_1(\overline{f}_A i_0 + f_A i_1). \end{aligned}$$

The decomposition function, BestDecomp, discovers these nondisjoint decompositions by using a preprocessing step which detects input sharing and marks off the 7-input cubes that are don't cares when input sharing is considered. Returning to our 4-to-1 multiplexer example, the decomposition function will consider alternative decompositions of the 7-input function $f(i_0, i_1, s_0, s_1, s_1', i_2, i_3)$. Signals s_1 and s_1' are different versions of the same signal — one connects to LE A and the other connects to LE B. The preprocessing step will will mark those cubes where s_1 and s_1' take on differing values. The presence of don't cares in the 7-input function allows the discovery of additional decompositions. Often it is possible to assign values to the don't cares in such a way that the column multiplicity requirement in the decomposition chart is met.

4.4 Cost Function

Alternative decompositions found by BestDecomp will move some of the inputs from LE A to LE B and vice versa. Each of these moves will change the slack of the input being moved. An accurate LE delay model and an approximate routing network delay model (which assumes best case routing) are used to estimate the change in slack for each input that was moved. The cost function is based on the changes to the slack of the inputs being moved and is modeled after the VPR cost function [15].

The slack of an input i before and after decomposition is denoted t_i and t_i', respectively. The criticality of an input depends on the maximum slack in the circuit, t_{max}, and the initial slack as follows

$$r_i = 1 - \frac{t_i}{t_{max}}$$

A criticality value of 1 is associated with inputs on the critical path. Using the criticality and the change in slack, the gain of an input i as a result of decomposition is given by

$$g_i = r_i^e(t_i' - t_i)$$

where e is a parameter that determines the relative importance of criticality to the cost calculations.

In addition to considering the effect of the decomposition on individual inputs, the overall effect of the decomposition on the two LEs A and B is also considered. The minimum slack at the inputs of A and B before and after decomposition is denoted $t_{A,B}$ and $t'_{A,B}$, respectively. The criticality and gain are defined similarly:

$$\begin{aligned} r_{A,B} &= 1 - \frac{t_{A,B}}{t_{max}} \\ g_{A,B} &= r_{A,B}^e(t'_{A,B} - t_{A,B}) \end{aligned}$$

The cost of a decomposition is defined as a weighted sum of the overall effect on the LEs and the individual effect on each input

$$c = \lambda g_{A,B} + (1 - \lambda)\sum_i g_i$$

where λ is a parameter that determines the relative importance of the two components. Decompositions with higher values of c are preferred and any decomposition with a negative value of c is rejected.

The cost function is not particularly sensitive to the values selected for e and λ. Values for e between 2 and 16 and values for λ between 0.25 and 0.95 were found to work well.

5. EXPERIMENTAL RESULTS

In our experiments, the first three steps of the FPGA CAD flow, design entry, synthesis and technology mapping, were performed by a leading third-party synthesis tool capable of targeting Altera devices and the last four steps, including functional decomposition and incremental placement, were performed by a modified version of Quartus II v2.2 [17] [1].

Table 1 presents the performance of the functional decomposition technique on 22 industrial circuits containing between 5000 and 10000 LEs. These results were obtained with the maximum number of decomposition iterations set to 8. This value was selected as each decomposition iteration increases compile time and there were very few circuits that benefited from a larger number of iterations. The functional decomposition and incremental placement steps increase the compile time of Quartus II by approximately 30%. The third column (*Speedup*) indicates the improvement in the final (post-routing) critical path as a result of applying the functional decomposition technique. An average performance improvement of 7.6% was obtained. Although the technique rejects bad decompositions there are circuits whose performance is

[1]The functional decomposition technique described in this paper is part of Quartus II's physical synthesis feature. In addition to the technique described here, the physical synthesis feature includes several other techniques that target both combinational and sequential logic. Refer to [18] for a description of how the physical synthesis feature can be used to speed up a design.

Circuit	*Size* (LEs)	*Speedup* (%)
cct1	5162	4.1
cct2	5308	14.0
cct3	5342	5.0
cct4	5677	5.3
cct5	5775	8.5
cct6	5872	6.5
cct7	5947	4.2
cct8	6036	7.9
cct9	6201	4.9
cct10	6404	26.3
cct11	6482	13.3
cct12	7267	-1.4
cct13	7269	9.1
cct14	7277	0.0
cct15	7494	9.5
cct16	7685	4.2
cct17	8107	17.7
cct18	8278	8.0
cct19	8346	-2.6
cct20	8768	-2.0
cct21	8886	12.4
cct22	9768	11.6
Average		7.6

Table 1: Performance of the post-placement functional decomposition.

degraded as a result of applying the technique (cct12, cct19 and cct20). These are a result of the approximation errors in the timing analysis performed by the technique. Decompositions that involve duplication will increase the size of the circuit. On average, these duplications increase a circuit's size by 0.3%.

6. CONCLUSION

We described a method for post-placement optimization based on functional decomposition. The technique is iterative in nature and attempts to find alternative decompositions of the logic around near-critical sinks. This technique, in concert with incremental placement, produces an average speedup of 7.6% while incurring an area penalty of 0.3%.

7. REFERENCES

[1] M. Sheng and J. Rose. Mixing Buffers and Pass Transistors in FPGA Routing Architectures. In *Proceedings of the ACM Int. Symposium on FPGAs*, Monterey, CA, Feb. 2001, pp. 75–84.

[2] M. Pedram and N. Bhat. Layout Driven Logic Restructuring/Decomposition. In *Proceedings of the Int. Conf. on Computer-Aided Design*, San Jose, CA, Nov. 1991, pp. 134–137.

[3] J. Y. Lin, A. Jagannathan and J. Cong. Placement-Driven Technology Mapping for LUT-Based FPGAs. In *Proceedings of the ACM Int. Syposium on FPGAs*, Monterey, CA, Feb. 2003, pp. 121–126.

[4] Y. Jiang, A. Krstic, K. Cheng and M. Marek-Sadowska. Post-Layout Logic Restructuring for Performance Optimization. In *Proceedings of the Design Automation Conference*, Anaheim, CA, June, 1997, pp. 662–665.

[5] Y. Lian and Y. Lin. Layout-based Logic Decomposition for Timing Optimization. In *Proceedings of the Asia Pacific Design Automation Conference*, Hong Kong, Hong Kong, Jan. 1999.

[6] G. Stenz, B. Riess, B. Rohfleisch and F. Johannes. Timing Driven Placement in Interaction with Netlist Transformations. In *International Symposium on Physical Design*, Napa Valley, CA, 1997, pp. 36–41.

[7] T. Tien, H. Su and Y. Tsay. Integrating Logic Retiming and Register Placement. In *Proceedings of the Int. Conf. on Computer-Aided Design*, San Jose, CA, 1998, pp. 136–139.

[8] L. Kannan, P. Suaris and H. Fang. A Methodology and Algorithms for Post-Placement Delay Optimization. In *Proceedings of the Design Automation Conference*, San Diego, CA, June 1994, pp. 327–332.

[9] D. Singh and S. Brown. Integrated Retiming and Placement for Field Programmable Gate Arrays. In *Proceedings of the ACM Int. Syposium on FPGAs*, Monterey, CA, Feb. 2002, pp. 67–76.

[10] K. Schabas and S. D. Brown. Using Logic Duplication to Improve Performance in FPGAs. In *Proceedings of the ACM Int. Syposium on FPGAs*, Monterey, CA, Feb. 2003, pp. 136–142.

[11] Altera. *Stratix Device Handbook*. Vol. 1, v2.0, July 2003.

[12] D. P. Singh and S. D. Brown. Incremental Placement for Layout-Driven Optimizations on FPGAs. In *Proceedings of the Int. Conf. on Computer-Aided Design*, San Jose, CA, 2002, pp. 752–759.

[13] R. Ashenhurst. The Decomposition of Switching Functions. In *Int. Symposium on Theory of Switching Functions*, 1959, pp. 74–116.

[14] H. Curtis. A generalized tree circuit. *Journal of the ACM*, 1961, 8:484–496.

[15] A. Marquardt, V. Betz and J. Rose. Timing-Driven Placement for FPGAs. In *Proceedings of the ACM Int. Syposium on FPGAs*, Monterey, CA, Feb. 2000, pp. 203–213.

[16] R. Hitchcock, G. Smith and D. Cheng. Timing Analysis of Computer-Hardware. In *IBM Journal of Research and Development*, Jan. 1983, pp. 100–105.

[17] Altera. *Quartus II Version 2.2 Release Notes*. v1.0, Dec. 2002.

[18] Altera. *Quartus II Development Software Handbook v4.0 (Three-Volume Set)*. v1.0, Feb. 2004.