

Automated Extraction of Physical Hierarchies for Performance Improvement on Programmable Logic Devices

Deshanand P. Singh
Altera Corporation
dsingh@altera.com

Terry P. Borer
Altera Corporation
tborer@altera.com

Stephen D. Brown
Altera Corporation
sbrown@altera.com

Abstract—Circuits implemented in FPGAs have delays that are dominated by its programmable interconnect. This interconnect provides the ability to implement arbitrary connections. However, it contains both highly capacitive and resistive elements. The delay encountered by any connection depends strongly on the number of interconnect elements used to route the connection. These delays are only completely known after the place and route phase of the CAD flow. We propose to produce timing-directed placement constraints for designs using a concept referred to as *critically connect components* (CCCs). These placement constraints are created after an initial unconstrained run of the QuartusII place and route tool. In this manner, the placement tool can create constraints that guide subsequent passes away from any mistakes made on the initial pass. Experiments conducted on Altera’s APEX20K family using this methodology along with LogicLock placement constraints indicate an average performance gain of approximately 6% on a varied suite of industrial circuits.

I. INTRODUCTION

Automated placement algorithms for complex PLDs perform the time-consuming task of manually mapping logic elements to physical locations on their target device. However, even state of the art automated algorithms are incapable of producing a solution that is comparable to a user defined manual placement. High performance designs are often floorplanned in such a way that critical regions of logic are grouped close together so that interconnect delays can be reduced as much as possible for timing critical paths. In this paper, we present an algorithm that tries to emulate the decision making process of a typical designer to constrain and floorplan a design to achieve better performance. There are two key components to this algorithm. The first involves the generation of timing-directed placement grouping constraints (PGC) and the second is a placement tool that can respect these constraints.

Although this paper concentrates on the PGC algorithm, we review the capabilities of the LogicLock [2] placement tool available since the introduction of Altera’s QuartusIIv1.1. This is the tool that allows the placement and routing algorithms to respect the constraints generated by PGC. LogicLock provides the capability to instruct the automated placement algorithm to keep certain groups of logic together. Logical groupings are accomplished with a construct known as a *LogicLock Region*. Each region contains a group of logic elements as well as a bounding box that defines the physical constraint of the

logical grouping. The bounding box itself has several attributes associated with it. For example, the size of the bounding box need not be specified by the user as the box may be *automatically-sized* by the LogicLock tool. The origin of the bounding box is also not required as the box may *float*. In this case the designer requests that their logical elements should be confined to a box, however the position of the box should be determined by the tool. This ability is useful because the designer does not need to spend the time to position these boxes. Together with the automatic-sizing attribute, the most basic constraint is the assignment of a group of logic elements to an automatically-sized floating bounding box.

The LogicLock placement constraints can also be applied in a hierarchical fashion. Regions of logic may be freely inserted into other parent regions to create a hierarchy of arbitrary depth. The flexibility of this system allows designer to create

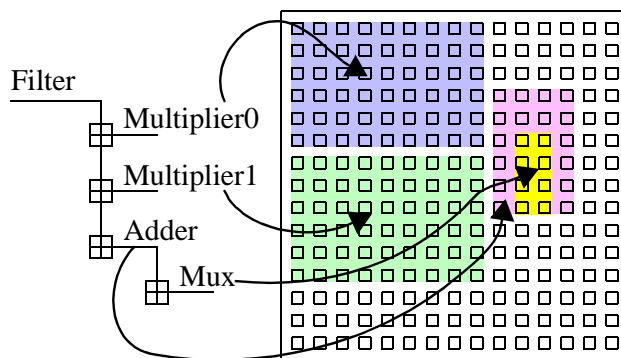


Fig. 1. Logical to physical mapping.

a mapping from their logical hierarchy to a physical hierarchy, as shown in Figure 1. Although a 1-to-1 mapping is shown, there need not be any correlation between the functional and physical hierarchies. The PGC algorithm takes full advantage of this type of flexibility.

We’ve seen that using these LogicLock placement constraints can lead to significant performance improvements. Industrial designs often comprised of a series of smaller subcomponents with a significant amount of custom logic that links these components together. A natural first experiment was to have designers assign these subcomponents to auto-

matically sized floating LogicLock regions simply by looking at the names in the logical hierarchy. In this way, these subcomponents are naturally kept together during placement. Figure I shows the results of this designer driven approach. We

TABLE I
 F_{max} SPEEDUP OVER FLAT EQUIVALENT

Name	LEs	Regions	Speedup (%)
cct1	2061	11	15.53%
cct2	10460	32	22.22%
cct3	4999	17	4.34%
cct4	10630	11	4.20%
cct5	13405	22	11.44%
cct6	14149	23	1.59%
cct7	20532	4	-0.18%
cct8	1583	5	-1.53%
cct9	8300	9	11.59%
cct10	793	2	7.54%
cct11	10914	16	-1.30%
cct12	5455	6	29.39%
cct13	361	1	5.39%
cct14	36836	2	1.61%
cct15	614	5	17.24%
cct16	32725	141	58.85%
cct17	2504	20	24.73%
cct18	4425	25	10.04%
cct19	5994	6	4.93%
cct20	5357	33	24.32%
cct21	5828	34	6.68%
cct22	4720	14	5.38%
cct23	3640	6	8.91%
cct24	13405	56	8.21%
cct25	7147	20	7.15%
			11.66%

find an average performance improvement of approximately 12%. Note that in many cases only a small portion of the design was constrained to LogicLock regions because we could only extract portions of logic that were explicitly marked as a subcomponent of the hierarchy via the naming convention. However, these were usually timing critical regions of logic such as multipliers. O

Our hope is that we can automatically assign portions of logic to regions to replicate these results. To ensure generality, we adopt an approach that does not use the naming scheme produced by the designer since many designs may be flattened or even encrypted before entry to the place and route engine.

The rest of this paper is organized as follows: First we provide some background information on Altera’s APEX family of devices. It is these devices that we target for our PGC experiments. Next we provide a detailed description of the PGC algorithm and present results for a suite of industrial circuits. Finally, we present conclusions and future work.

II. APEX ARCHITECTURE

Figure 2 gives a simplified depiction of Altera’s APEX architecture [1]. The APEX chip has a hierarchical structure that exploits the wiring locality properties of target circuits. The lowest level of the hierarchy is the basic logic-element (LE) which consists of a 4-input lookup table with a configurable flip-flop. Groups of 10 LEs are clustered together to form

a logic-array-block (LAB). Each logic element in a LAB can communicate with other logic elements via the *local* interconnect. These local lines are the fastest method of communication for general purpose circuitry¹.

The specific details of the higher level APEX hierarchy structure vary from chip-to-chip. In the remainder of this section, we describe the APEX 20K400 part. Groups of 16 LABs and 1 Embedded System Block (ESB) form a MegaLab. ESBs are used to implement memory circuitry such as RAMs, ROMs, CAMs, etc. Each LAB in the MegaLab can communicate with its adjacent neighbors via its local interconnect lines. Otherwise, the communication is sent via the MegaLab interconnect, which consists of a set of continuous metal lines that span the width of the MegaLab. LEs within any LAB can directly drive these MegaLab lines. The signal then traverses the metal line to a local input line associated with the destination LAB, and then to any dependent LEs. Signals sent via the MegaLab interconnect have the fastest propagation time, other than intra-LAB and nearest-neighbor communication.

In the APEX 20K400, groups of 13 MegaLabs form an Octant. Each MegaLab in the Octant can communicate with other MegaLabs via a series of continuous vertical interconnect lines (V-lines). Signals communicating across an Octant start from a source LE that directly drives a V-line, traverses to the target MegaLab, switches on to the MegaLab interconnect and makes its way to the destination LEs as described previously.

Two Octants are stacked vertically to form a Column. Communication between Octants is made possible by a buffered switch that connects the V-lines between two Octants together.

Groups of two Columns form a Half. Continuous horizontal interconnect lines (H-lines) run across the width of the two Columns. Signals that traverse across the Half start at the source LE which can directly drive the H-line. From the H-line, the signal can drive the an appropriate V-line in the target Octant and traverse its way to the destination LEs using the intra-Octant communication scheme described above.

Finally, the two halves are grouped side by side. Signals may traverse between halves because the adjacent H-lines in each half are connected with a buffered switch.

Each level of hierarchy described has delay characteristics which are progressively slower as we move from LABs → MegaLabs → Octants → Columns → Halves. Table II shows the relative point-to-point delays experienced at different points in the APEX hierarchy structure [3]. For example, the signal traversing across a Column may experience more than twice the delay of a signal traversing across the MegaLab interconnect.

The step function behavior of the worst-case delays for different hierarchies motivates the need for controlled placement groupings. Critical sections of logic such as multipliers have a greater chance of meeting timing constraints if the logic can be grouped together into a MegaLab sized region, rather than

¹The APEX architecture has high-speed direct connections (carry and cascade chains) for arithmetic operations.

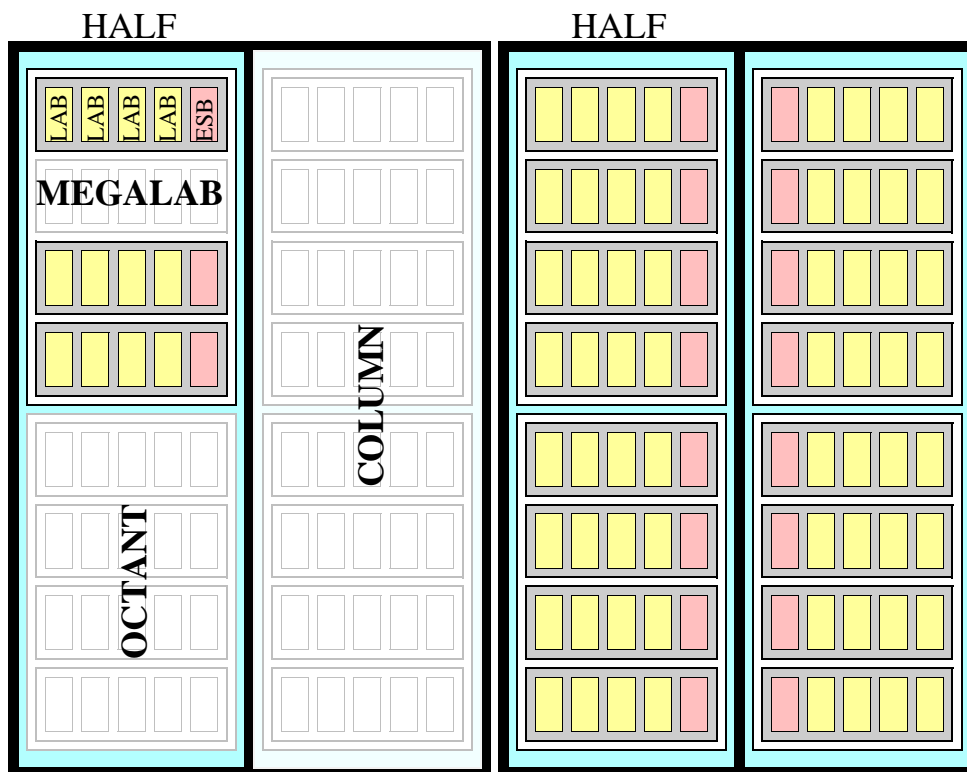


Fig. 2. Simplified view of the APEX architecture.

TABLE II
RELATIVE DELAYS FOR EACH HIERARCHY LEVEL.

Region	Relative Delay
LAB	1
MegaLab	3
Octant	5
Column	7
Half	9
Chip	11

depending on the flat placement engine to naturally find this grouping.

III. PLACEMENT GROUPING CONSTRAINT GENERATION (PGC)

A. Overview

As mentioned previously, manual assignment of portions of the circuit to regions is an effective method for increasing maximum operating frequency of the circuit. In this section, we describe the totally automated approach for PGC which requires two-passes as shown in Figure 4. The first pass is a flat compile without any hierarchical grouping constraints. After this initial pass, we find the portions of the circuit that the placer initially had difficulty with. These critical sections are then constrained to automatically-sized regions and the circuit is compiled again with the hierarchical placement algorithms described previously. Hopefully, constraining these critical portions of the netlist will lead to better speed performance.

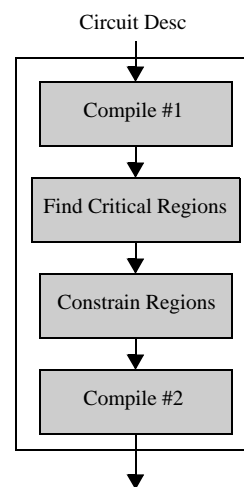


Fig. 4. Two Pass flow.

The algorithm for finding critical sections of logic involves isolating the *critically connected components* in the netlist. A critically connected component is defined as follows: Annotate the circuit netlist post-placement and routing delay information, remove all connections that have a slack below some threshold, CC_T , and find all connected components in the resulting netlist. The goal of this optimization is to try to find a natural grouping of the critical and near-critical paths that exist in the design. A graphical depiction of the critically connected components for a circuit in our benchmark



Fig. 3. Critical connected components.

suite is shown in Figure 3. Each square with the same color represents a group of logic cells that belong to the same critically connected component. Clearly, some of the critically connected components are spread across the chip in tightly connected clusters. One of the reasons for this phenomena is that to bring these cluster close together, the entire cluster must be moved. This is impossible with traditional placement schemes that rely on swaps or moves of single cells at one time.

One problem with clustering critical paths into floating regions is shown in Figure 3. The critically connected component algorithm tends to produce regions with a large number of side-inputs. This creates a large amount of localized congestion because a large number of wires must be routed into a relatively small area. To avoid the congestion problems, a post processing step is run after the identification of critically connected components. This step "pulls" in a number of highly attracted logic elements into each critically connected component. For a given critically connected component CMP_i and a logic element LE_j , the attraction is given by:

$$Att(CMP_i, LE_j) = |ExNets(CMP_i) \cap Nets(LE_j)| \quad (1)$$

The quantity $ExNets(CMP_i)$ represents the set of all nets that connect a LE outside CMP_i to a LE within CMP_i , while $Nets(LE_j)$ represents all nets connected to LE_j .

Localized congestion also results from the fact that highly connected sections of the circuit are placed in a confined region. This region of the chip may physically not have enough routing resources for all the connections within the region. We account for this situation by modifying the LogicLock hierarchical placement algorithms to create a "soft" region. These regions have all the same properties as floating regions described previously, except that the logic elements are

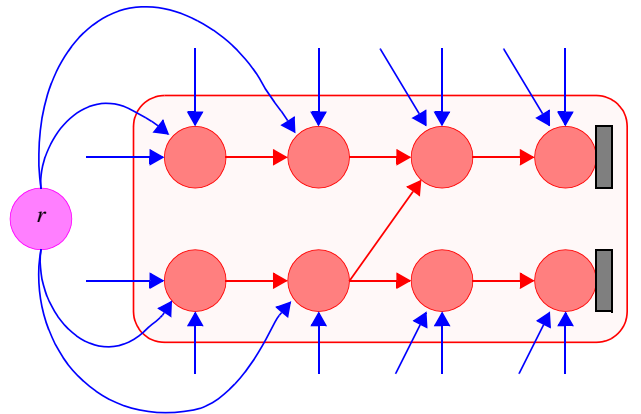


Fig. 5. Routing demand of critical regions.

strongly encouraged to stay within their region; however, they may move outside if timing or routing congestion could be improved. This allows any non-critical logic elements within the region to move outside the region to resolve congestion while maintaining timing.

B. Detailed Description

Pseudocode for the PGC algorithm is presented in Figure 6. First every LE is examined to see if it is critical. An LE is critical if it has a fanin or fanout whose slack is less than a threshold CC_T . All critical LEs will eventually be assigned to an automatically sized floating LogicLock region, while most non-critical LEs will not. Next, while there are still unconstrained critical LEs, one is selected as a *seed*. From this seed LE, we find the entire critically connected component that involves this seed LE. This critically connected component involves all LEs that are reachable from the seed LE via critical connections. Reachability is even allowed through registered nodes, although our subsequent experimental results will show that this decision was perhaps sub-optimal. As described previously, a post processing step is run to incorporate any LEs that are tightly connected to the current set so that we are not require to route a large number of wires into a small region. This post-processing step may have brought us in contact with another critically connected component so we rerun the loop to check for this very situation.

```

Mark all critical LEs
while (there exists an unconstrained critical LE)
  C = ∅
  Select a critical LEs as a seed
  C = seed
  do
    C' = findCriticallyConnectedLEsFrom(C)
    C = C ∪ C'
    R = R ∪ findAttractedLEs(R)
  while C' ≠ ∅
  Assign the elements of R to an new LL-region
end while

```

Fig. 6. PGC Algorithm

C. Adding Attracted LEs

The post-processing step of adding attracted LEs is a greedy loop that adds and new LE, l , that maximizes the value of $Att(C, l)$. Simply stated, it looks for an LE that has many nets in common with the critical with the connected component C . However, this implementation would end up eventually grabbing the entire circuit because all LEs will eventually have some attraction to the component, assuming that the circuit does not consist of disjoint entities.

First, we enforce a limit on the number of tightly attracted LEs that can be added to a critically connected component.

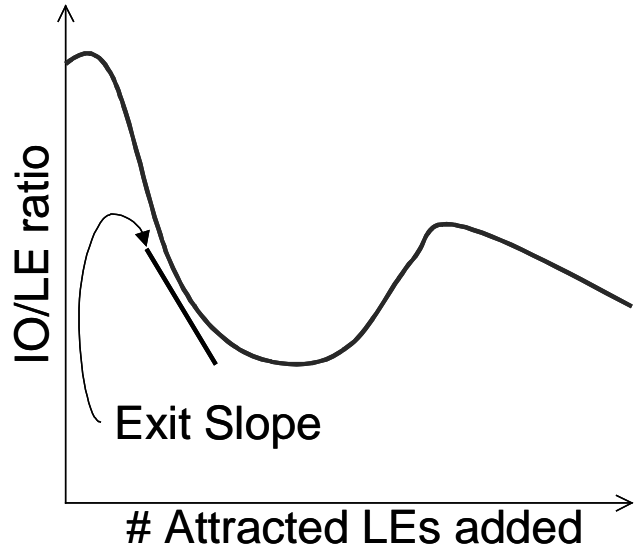


Fig. 7. IO to LE ratio within region.

This constraint is parameterized the growth factor, GF . This quantity controls how much an critically connected component is allowed to be expanded. For example, $GF = 2$ would indicate that we would allow the component to double in size.

Secondly, we would like to capture just enough LEs to reduce the I/O requirements to and from the region. Thus we monitor the I/O to logic element ratio for the set C as we add more and more attracted LEs. A typical example is shown in Figure 7. After some time, it becomes apparent that there is little point to adding extra LEs as the IO to LE ratio is reduced at a far slower rate. Thus, we monitor the slope of the ratio and exit at a point where we feel that there is no gain to be obtained by adding more tightly connected LEs. This criterion is controlled by a parameter known as the IO to LE exit slope, E_{slope} .

IV. EXPERIMENTAL RESULTS

Our experiments used a set of 29 industrial circuits. These circuits were gathered from real world FPGA applications, and are varied in their function. Each of these circuits was mapped into a netlist of LUTs using Synplicity's Synplify software. The QuartusII 2.1 software was then used to place, partition, re-place and route the design. Each design was compiled into the smallest APEX 20KE device that would hold it.

Table III plots the speedups for the 29 industrial circuits. Flat (no LogicLock region extraction) Fmax, PGC extraction Fmax, % improvement, and number of LEs in any LogicLock region. PGC with region boundaries is also shown. Each result is an arithmetic average of three seeds.

It is important to note that this experimental setup represents the state-of-the-art CAD flow. Since the circuits are already well placed and routed, the opportunities for improvement are reduced. This must be taken into account when interpreting the gains of this algorithm.

It was found experimentally that some partitions grew too large, or that some partitions expanded through registers to

TABLE III
RESULTS

Circuit	#LEs	Flat F_{max}	PGC	%Imp	#LEs part	PGC w/bnd	%Imp	#LEs part
ind1	6513	122.91	113.58	-7.6%	1099.33	118.73	-3.4%	972.33
ind2	5807	38.35	41.87	9.16%	4767.67	40.73	6.20%	4542.33
ind3	5163	28.64	28.30	-1.2%	4877.00	26.67	-6.89%	4842.33
ind4	5828	42.29	42.91	1.47%	2145.67	45.45	7.46%	1866.00
ind5	4514	82.58	97.08	17.55%	1549.33	95.61	15.77%	1437.67
ind6	4178	83.02	82.99	-0.04%	1337.33	83.55	0.65%	1187.67
ind7	7496	60.33	60.37	0.07%	1308.67	65.38	8.37%	1302.67
ind8	10868	30.16	31.31	3.82%	2741.33	31.22	3.54%	2449.00
ind9	12243	70.75	71.64	1.26%	2484.33	72.82	2.93%	2330.33
ind10	13490	82.01	93.17	13.60%	690.00	93.51	14.02%	154.00
ind11	14149	18.98	19.99	5.30%	4538.33	20.26	6.76%	4719.00
ind12	4744	103.77	103.56	-0.21%	1542.33	106.51	2.64%	904.00
ind13	5593	41.63	44.93	7.94%	1846.00	44.20	6.19%	1560.67
ind14	33443	29.65	30.99	4.50%	6026.33	30.36	2.38%	6916.50
ind15	11304	57.49	53.97	-6.13%	1294.00	55.23	-3.93%	1127.67
ind16	4338	76.63	80.46	5.00%	230.67	80.99	5.69%	213.67
ind17	518	71.10	77.89	9.55%	277.33	77.40	8.86%	269.00
ind18	7548	58.84	50.95	-13.41%	1755.67	60.47	2.77%	1721.50
ind19	12765	56.74	54.81	-3.41%	4418.67	56.72	-0.04%	3875.33
ind20	9027	63.33	66.90	5.63%	1089.67	67.04	5.85%	968.00
ind21	20000	35.62	38.83	9.02%	4064.67	38.95	9.36%	4590.00
ind22	2546	138.63	158.13	14.06%	754.00	156.61	12.96%	616.33
ind23	5155	28.86	29.75	3.11%	368.33	31.03	7.53%	357.67
ind24	3428	71.96	70.61	-1.88%	966.00	71.28	-0.94%	934.00
ind25	13776	40.00	40.55	1.38%	3033.00	44.33	10.83%	2990.33
ind26	7147	136.12	150.30	10.41%	188.33	158.66	16.56%	246.33
ind27	2673	90.82	89.86	-1.05%	708.00	89.04	-1.95%	629.67
ind28	2504	41.56	44.92	8.09%	1409.33	45.42	9.30%	1124.67
ind29	8554	32.05	32.60	1.72%	2432.00	33.16	3.47%	2477.00
Geo Mean	6517.75	55.80	57.56	3.15%	1459.29	58.66	5.12%	1295.83
Arith Mean	8459.03	63.27	65.63	3.73%	2067.01	66.94	5.81%	1976.75

encapsulate parts of the circuit that were far from critical. To compensate for this, registers were used as soft boundaries for regions. This means that a region would only expand through a register if that register had both inputs and outputs that were critical. As shown, PGC with registered boundaries yields a 5.81% speedup, compared to 3.73% speedup from normal PGC variant. It was also found that this technique reduce the size of some partitions, and more importantly, increased the average number of LogicLock constraints per design.

Table IV shows the values uses for the constants that govern the performance of PGC.

TABLE IV
EXPERIMENTALLY DETERMINED VALUES OF PARAMETERS

Parameter Name	Value
CC_T	0.225 * longest path
GF	1.3
E_{slope}	9.5

V. CONCLUSIONS

We have shown that physical extraction can be an effective technique to optimize FPGAs. This technique holds promise because detailed knowledge of the designs is not required. Also, effective hand partitions of designs often usually involve good code level partitioning. This is so that design hierarchy corresponds to physical hierarchy.

A. Convergence

This method described in this paper are iterative in nature. Thus it is natural to whether this algorithm will *converge* meaning will repeated applications of the same technique continue to increase results. This was found not to be true; repeated application of this algorithm in a loop was found to degrade results slightly.

REFERENCES

- [1] Altera. *Altera 2000 Databook*. Available from: <http://www.altera.com/html/literature/lds.html>.
- [2] Constrained FPGA Placement Algorithms for Timing Optimization Field Programmable Logic and Applications, submitted, 2003.
- [3] M. Hutton, K.Adibsamii and A. Leaver. *Timing-Driven Placement for Hierarchical Programmable Devices*. Ninth International Symposium on Field-Programmable Gate Arrays, 2001.