Automatic Partitioning for Improved Placement and Routing in Complex Programmable Logic Devices

Valavan Manohararajah, Terry Borer, Stephen D. Brown, and Zvonko Vranesic

Altera Corporation
Toronto Technology Center
151 Bloor St. West, Suite 200
Toronto, ON
CANADA M4Y 1R5
vmanohar@altera.com, tborer@altera.com,
sbrown@altera.com, zvonko@altera.com

Abstract. This work explores the effect of adding a new partitioning step into the traditional complex programmable logic device (CPLD) CAD flow. A novel algorithm based on Rent's rule and simulated annealing partitions a design before it enters the place and route stage in CPLD CAD. The resulting partitions are then placed using an enhanced placement tool. Experiments conducted on Altera'a APEX20K chips indicate that a partitioned placement can provide an average performance gain of 7% over flat placements.

1 Introduction

An incremental design methodology called LogicLock [1] was introduced in version 1.1 of Altera's Quartus II software. LogicLock offers users an alternative to the traditional "flat" place and route steps in CPLD CAD. In traditional CPLD CAD, the design being compiled is represented as one large netlist as it enters the place and route steps. This allows a global view that is beneficial for some circuits. However, for many large circuits it is difficult for a placement tool to ensure that tightly connected components in the design are not separated by large distances. With the LogicLock feature, users can create *partitions* (referred to as *regions* in Quartus II terminology) of logic which are kept together during the place and route stages. A partition may contain both logic and any smaller child partitions. Feedback from LogicLock users has indicated that the ability to keep components together during placement and routing helped increase the overall speed of several designs. This work considers a natural extension to the LogicLock feature. When the user has not specified a set of partitions, a partitioner within Quartus II is run to determine a set of good partitions for the design. To operate without any user assistance, the partitioner automatically determines both the number of partitions as well as the partitioning itself. The partitioning procedure may not be beneficial for all circuits therefore we also consider the problem of determining when the automatically created partitions are to be used.

2 Previous Work

An excellent survey of previous work in netlist partitioning was presented by Alpert and Kahng [4]. In their work, partitioning approaches are classified into four categories: move-based approaches, geometric representations, combinatorial formulations and clustering approaches. Our work uses a move-based approach with some elements of the clustering approach. During a single iteration of the proposed algorithm, a move-based approach is used within a simulated annealing framework to generate a set of partitions. Bigger partitions are created out of smaller partitions discovered during earlier passes of the algorithm in a process that is similar to many bottom-up clustering approaches [7] [8].

Roy and Sechen [5] explored the use of a simulated annealing approach within a timing-driven FPGA partitioner. In their work, the netlist is clustered to create a smaller problem for the partitioning tool. In another simulated annealing approach described by Sun and Sechen[6], the circuit to be placed is clustered in a preprocessing step and the clustering information is used during placement.

Our work uses a cost function based on Rent's rule [16]. A clustering algorithm based on Rent's rule was previously described by Ng et al [9]. Hagen et al. [10] described a spectra-based ratio cut partitioning scheme that creates partitions with the lowest observed Rent's parameter. Recently, Rent's rule has found several uses in interconnection prediction and congestion estimation during placement [11] [12] [13] [14].

3 The APEX Architecture

In this work, Altera's APEX chips were the target of the partitioning experiments. A simplified internal view of an APEX chip is presented in Figure 1. At the highest level, the chip is organized into four quadrants. Each quadrant contains an array of *MEGALABs* arranged into set of columns and rows. A *MEGALAB* contains a number of *LABs* and a single *ESB*. Each LAB contains a set of *LEs*. An LE is the basic logic element in the APEX architecture. It consists of a four-input LUT (4-LUT), a programmable register, carry and cascade logic. The ESB can be used to realize memory functions in the design. For a detailed description of the internal structure and interconnect arrangement in the APEX chips see [2].

4 The CAD flow

The CAD flow used in our work is illustrated in Figure 2. Once technology mapping is complete an optional partitioning step is executed to discover a set of partitions for the design. Following partitioning, a partition aware clustering

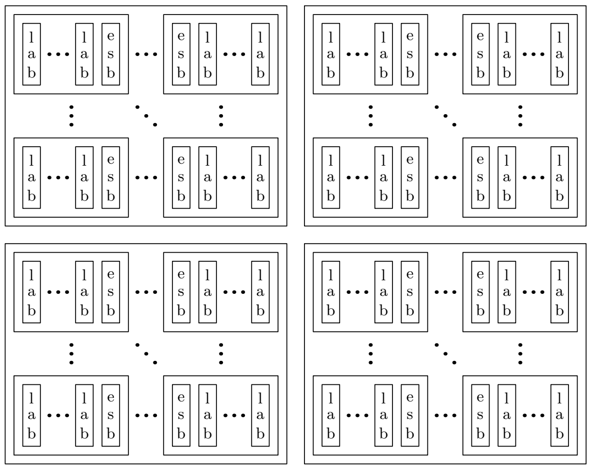

Fig. 1. The APEX architecture.

step is performed. The partition aware clusterer decomposes the logic within a partition into LABs. It also ensures that logic from one partition is not mixed with logic from another partition. The partition aware placement step is much more complicated than the normal placement step found in the traditional flow. First, each partition in the design is assigned a rectangular shape large enough to hold its contents. After shape assignment, the new placement step has to determine a position for each of the rectangular shapes in the design as well a position for each of its contents. A detailed description of the placement tool is beyond the scope of this paper. A description can be found in [3]. The final step, routing, requires no changes and is identical to the one used in an unpartitioned CAD flow.

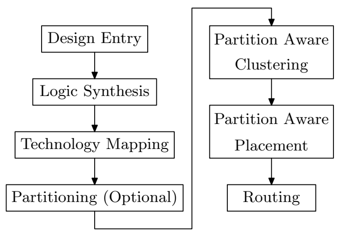

Fig. 2. A CAD flow that incorporates a partitioning step.

5 Partitioning

The input to the partitioning step is a netlist that consists of memory elements, LEs and IOs. The partitioning step produces a set of partitions for the netlist as output. A partition may contain both LEs and smaller partitions. Memory elements and IOs do not participate in the partitioning process because their placement is quite restricted. Both memory elements and IOs can only be placed at select locations on the chip whereas the possibilities for the placement of LEs is numerous. The placement tool is left the task of discovering the best spots for both memory elements and IOs.

5.1 Overview

An overview of the partitioning algorithm is presented in Figure 3. First, a timing analysis routine is executed to determine a weight for each wire in the design. Interconnect delays are required in order to perform a timing analysis but they only become available after placement and routing have completed. An experiment was performed with 25 small to medium sized industrial circuits to determine if a net's delay could be related to its fanout. Each circuit was placed and the delay of the fastest path (using general interconnect) between each source-sink pair was obtained. This data allows for some rough estimation of wire delays. A rough estimate for the delay (in picoseconds) of a source-sink connection on a net with fanout of f is given by $(1141 \log f + 733)$. Now that a model for interconnect delay is available, timing analysis can be performed. Timing analysis determines a slack s_{uv} [15] for each source-sink pair (u, v). We define the criticality, c_{uv}, of each source-sink pair (u, v) as

$$c_{uv} = 1 - \frac{s_{uv}}{\max_{\forall ij} s_{ij}} \qquad (1)$$

The criticality provides an indication of the relative importance of each source-sink connection and is used as the weight when the cost function is computed.

The main partitioning loop consists of block formation and partitioning steps. Block formation behaves differently depending on the value of h. During the first call to block formation ($h = 0$), blocks will be created out of the LEs in the design. Most blocks will contain a single LE. However, if there is logic in the design that uses the specialized carry or cascade chain routing, then all LEs that are a part of the carry or cascade chain will become a part of a single block. During the subsequent calls to block formation ($h > 0$), blocks are created out of the partitions discovered during the previous pass. Each partition discovered during the previous call to PARTITION is transformed into a block. In this manner, larger partitions can be built out of smaller ones. The partitioning step, PARTITION, uses a simulated annealing based optimization algorithm to divide the blocks into partitions. The main loop terminates whenever new partitions are not being discovered or when the number of levels in the hierarchy discovered so far exceeds the prespecified limit, *hlimit*. In the experiments, *hlimit* was set to 4.

```
1: TimingAnalysis()
2: $h \leftarrow 0$
3: do
4:     BlockFormation($h$)
5:     Partition()
6:     $h \leftarrow h + 1$
7: while NewPartitionsFormed() = true
           and $h < hlimit$
```

Fig. 3. Overview of the partitioning process.

5.2 Cost Function

Rent's Rule The partitioning cost function is based on Rent's rule [16]. The rule relates the number of blocks in a partition, B, to the number of external connections, P, emanating from the partition. Rent's rule is given by

$$P = T_b B^r \tag{2}$$

where T_b denotes the average number of interconnections for a block in the partition and r is Rent's exponent. Rent's exponent, r, is a value in the range $[0, 1]$. A value close to 1 indicates that most of the connections in the partition are external and a value near 0 indicates that almost all connections are internal.

Weighted Rent's Rule Rent's rule as stated in Equation 2 treats all connections equally. However, to account for the fact that some connections may be more critical than the others, each connection is weighted using the criticality value obtained through timing analysis (Section 5.1). The terms in Equation 2 were revised slightly to account for weighted connections. The number of external connections, P, now denotes the total weight of all external connections. Similarly, the average number of interconnections for a block in the partition, T_b, now denotes the average weight of interconnections for a block in the partition.

Cost of a Partitioning Solution A solution to the partitioning problem consists of a set of partitions each containing a number of blocks. The cost of the partitioning is given by

$$C = \frac{\sum B_i r_i}{\sum B_i} \tag{3}$$

where B_i is the number of blocks in region i and r_i is Rent's exponent for region i. Like Rent's exponent, the cost, C, is also a value in the range $[0, 1]$. A value closer to 0 is indicative of a good partitioning and a value closer to 1 is indicative of a bad partitioning.

Effect of Partition Size Consider two extreme solutions to the partitioning problem. In one case, all blocks may be placed in a single partition. In the other case, every partition contains a single block. Both cases cannot improve the quality of the placement that is obtained. A lower and upper limit on partition size was established to prevent the creation of partitions that are either too small or too large. If some solution to the partitioning problem has partitions that are smaller than the lower limit or bigger than the upper limit, the cost, C, is pulled towards 1.

5.3 Optimization

The main partitioning procedure, referred to as PARTITION in Figure 3, starts with a random partitioning solution and iteratively improves it using simulated annealing [17].

The starting temperature for the anneal, is obtained using a method similar to that described in [18] [19]. A set of m moves is randomly generated. Each move is then evaluated and the change in score is observed. The initial temperature is computed to be 20 times the standard deviation of the set of scoring changes.

At each temperature in the anneal, m moves are generated and evaluated. The value of m is equal to the total number of external connections of the blocks participating in the partitioning process. During the first pass of partitioning, most blocks will contain a single LE. A four-input LUT is the primary component within an LE therefore, on average, each block is expected to have four input connections and one output connection. If n is the number of LEs in the circuit then m is approximately equal to $5n$ during the first pass of partitioning. During the subsequent passes, blocks are formed from the partitions discovered during the preceding pass. A block that is formed in this manner will contain highly "localized" connections. The value of m will be greatly reduced because very few connections will be external to the newly formed blocks.

The move generation routine used by the simulated annealer generates two types of moves: *directed* and *random*. To generate a directed move, a block with at least one connection crossing a partition boundary is picked at random. The directed move that is generated moves the block into a partition containing one of its endpoints. A random move may be further classified into two types: *empty* and *non-empty*. An empty random move picks a block at random and creates a new partition to contain the block. A non-empty random move picks a block at random and moves it into a randomly selected partition that already contains a number of blocks. The move generation routine generates directed moves with a probability of 0.75 and random moves with a probability of 0.25. During random move generation, empty moves are generated with a probability of 0.25 and non-empty moves are generated with a probability of 0.75.

Once all moves at a particular temperature have been generated and evaluated, the temperature is reduced for the next iteration in the anneal. The new temperature, t_{new}, is given by

$$t_{new} = t_{old} \cdot \gamma \cdot \left(\beta + (1-\beta)e^{-(\alpha-0.45)^2/4}\right) \tag{4}$$

Here, t_{old} is the current temperature and α is the accept ratio observed during the iteration. The accept ratio is defined to be the ratio of the number of moves accepted to the total number of moves tried. The multiplier, γ, controls the basic rate at which the temperature decreases. The fraction, β, controls whether the full multiplier or a scaled down version of the multiplier is used in determining the next temperature value. A Gaussian function which depends on the accept ratio, α, generates the scale factor used to reduce the multiplier. The function reaches its maximum value of 1 when the accept ratio is 0.45. Previous research has indicated the benefit of keeping the temperature hovering around an accept ratio of 0.45 [20]. Note that the full multiplier is used when the accept ratio is close to 0.45 or when β is close to 1. A value for β is computed by observing the improvement in score obtained during the pass that just completed. If there was a large improvement in score and if the accept ratio is low enough to ensure that the improvement cannot be attributed to random behavior, β is given a value close to 1. This allows the search to spend more time at those temperatures that seem to produce large improvements in the score.

Simulated annealing stops when the exit criterion is met. The exit criterion used here is similar to that described in [18]. The exit criterion is true if the current temperature is lower than the exit temperature, t_{exit}. A value for t_{exit} is computed based on the current cost, C, and the total number of external connections of all blocks in the design, n_{ext}:

$$t_{exit} = \epsilon \frac{C}{n_{ext}} \tag{5}$$

Here, ϵ is a small constant which was set to 0.05 during the experiments. Any move performed during the anneal is likely to affect several external connections. If the temperature drops below a fraction of the average cost of an external connection, it is unlikely that any move which increases the cost will be accepted and the annealing process can be terminated.

6 Experimental Results

In the first experiment, the performance of the partitioned flow was compared to the flat flow on 20 industrial circuits (Table 1). The partitioned flow produces an average speedup of 6.81% over the flat flow. Note that the partitioning algorithm produces a wide variety of partitions — circuit "ccta16" was partitioned into 6 partitions while "ccta14" was partitioned into 118 partitions.

In the second experiment, the performance of the automatic partitioning algorithm was compared to the performance of user created partitions (Table 2). The user partitions were created by Altera's field application engineers and design engineers working on the LogicLock feature. The user partitioned flow obtained a speedup of 9.3% over the flat flow whereas the automatically partitioned flow obtained a speedup of 7.82%. Apart from the results observed for circuits "cctb2" and "cctb3", the automatic partitioning is competitive with the user created partitions. In fact, for circuits "cctb1", "cctb4", "cctb5", "cctb7", "cctb9", "cctb11" and "cctb12", the automatically generated partitions outperformed the user created partitions.

Combining the results observed for both experiments, the automatically generated partitioning improved the overall speed of the 32 circuits by 7.04%.

Compared to the flat flow, the automatically partitioned flow increases the overall compile time by 25.26% for the 32 circuits. However, most of this time is spent within the partition aware placer rather than in the partitioner. The partitioning procedure itself consumes 7.67% of the total compile time. Most of the increase in compile time is due to the increased cost of performing partition moves (the normal placer deals with LAB moves only) in the placer. Given a circuit of size n, the complexity of the partitioning procedure is $O(n)$ (see Section 5.3) whereas the complexity of the placer is $O(n^{\frac{4}{3}})$. As circuits grow larger, the time spent within the partitioner will be a smaller fraction of the overall compile time.

Circuit		*Flat*	*Partitioned*					
Name	*Size (LEs)*	*Speed (Mhz)*	*Speed (Mhz)*	*Part-itions*	*Speedup (%)*	*C*	*R*	*U*
ccta1	9027	60.93	60.38	26	-0.90	0.36	3.27	1
ccta2	5357	126.58	153.73	33	21.45	0.31	12.44	1
ccta3	5828	39.02	43.08	34	10.40	0.54	5.38	1
ccta4	11304	52.87	52.89	23	0.04	0.51	2.86	1
ccta5	11273	38.02	46.43	16	22.12	0.52	4.39	1
ccta6	5593	41.47	43.45	43	4.77	0.55	4.45	1
ccta7	5455	99.03	109.06	30	10.13	0.09	2.80	1
ccta8	10824	42.57	43.83	46	2.96	0.32	17.79	1
ccta9	5380	57.51	56.36	15	-2.00	0.37	8.90	1
ccta10	7147	157.16	177.97	20	13.24	0.42	7.30	1
ccta11	6145	45.54	41.72	84	-8.39	0.76	1.48	0
ccta12	5086	95.19	110.14	10	15.71	0.41	27.22	1
ccta13	6789	30.16	29.47	28	-2.29	0.63	1.00	0
ccta14	9565	25.49	26.18	118	2.71	0.36	1.25	1
ccta15	4720	105.94	124.19	14	17.23	0.38	2.13	1
ccta16	3640	70.88	70.92	6	0.06	0.22	18.61	1
ccta17	12064	43.61	41.12	21	-5.71	0.58	4.86	0
ccta18	8940	12.43	12.46	16	0.24	0.67	2.98	0
ccta19	3403	71.57	78.29	16	9.39	0.37	9.76	1
ccta20	3378	43.14	51.84	35	20.17	0.34	16.09	1
Average					6.57			

Table 1. Comparing flat compilations with partitioned compilations.

7 Auto On/Off

There are instances where automatic partitioning does not improve the performance of the circuit. For example, in Table 1, when automatic partitioning is used, the performance of “ccta11” drops by 8.39% and the performance of “ccta17” drops by 5.71%. Similarly, in Table 2, the performance of “cctb2” drops by 12.84% with automatic partitioning. A natural question to ask is if situations like this can be prevented by using some statistics generated by the partitioner. First, we define the *crossing ratio*, R, as

$$R = \frac{c_{AvgAll}}{c_{AvgCross}} \tag{6}$$

Circuit		*Flat*	*User Partitioned*			*Auto Partitioned*					
Name	*Size (LEs)*	*Speed (Mhz)*	*Speed (Mhz)*	*Part-itions*	*Speedup (%)*	*Speed (Mhz)*	*Part-itions*	*Speedup (%)*	*C*	*R*	*U*
cctb1	4514	85.59	93.61	5	9.37	104.25	29	21.80	0.42	3.31	1
cctb2	10460	58.87	72.63	32	23.37	51.31	16	-12.84	0.61	2.33	0
cctb3	4999	26.76	30.47	17	13.86	26.97	79	0.78	0.48	4.82	1
cctb4	10630	98.65	109.84	11	11.34	113.07	2	14.62	0.31	16.50	1
cctb5	13405	92.82	94.28	22	1.57	96.99	64	4.49	0.31	17.29	1
cctb6	14149	20.85	21.38	23	2.54	20.60	13	-1.20	0.47	8.61	1
cctb7	20532	29.18	32.14	4	10.14	33.57	12	15.04	0.48	6.34	1
cctb8	8300	72.21	80.28	9	11.18	77.92	24	7.91	0.39	2.51	1
cctb9	10914	44.26	46.69	16	5.49	47.40	44	7.09	0.40	5.42	1
cctb10	4385	129.25	134.32	11	3.92	127.06	11	-1.69	0.09	3.60	1
cctb11	4936	79.81	87.02	16	9.03	101.33	136	26.96	0.40	1.68	1
cctb12	4425	107.97	118.57	25	9.82	119.75	4	10.91	0.17	13.10	1
Average					9.30			7.82			

Table 2. Comparing flat compilations, user partitioned compilations and automatically partitioned compilations.

Here, c_{AvgAll} is the average criticality of all connections in the circuit and $c_{AvgCross}$ is the average criticality of all connections that cross a partition boundary. Intuitively, a good set of partitions would have a low cost (see Section 5.2) and a high crossing ratio. Now, consider the following rule

$$\mathit{Useful} = \begin{cases} 1 \text{ if } C < 0.56 \text{ or } R > 5 \\ 0 \text{ otherwise} \end{cases} \tag{7}$$

The rightmost three columns of Tables 1 and 2 summarize the C, R and *Useful* values for each circuit. This simple rule could help eliminate the use of the automatically created partitions on circuits “ccta11”, “ccta13”, “ccta17” and “cctb2”. However, it does not eliminate circuit “ccta9” which has a low cost and a high crossing ratio. More experimentation with a wide variety of circuits is necessary to determine whether rules such as this could be used to determine when the automatically created partitioning is useful.

8 Conclusions

This work introduced a new partitioning algorithm that can be used to improve the quality of CPLD placements. The algorithm used simulated annealing to optimize a cost function based on Rent’s rule. An average performance improvement of 7% was observed for a set of 32 industrial circuits. The automatically created partitions were found to be competitive with partitions created by experienced users. This work also considered the possibility of using some of the statistics available to the automatic partitioner to determine if the automatically generated partitions would help or hurt a circuit’s performance.

References

1. Altera. “LogicLock Methodology White Paper”. Available at: http://www.altera.com/literature/wp/wp_logiclock.pdf.

2. Altera. *Altera 2000 Databook*. Available at: http://www.altera.com/html/literature/lds.html.
3. D. P. Singh, T. P. Borer and S. D. Brown. "Constrained FPGA Placement Algorithms for Timing Optimization". *ACM Intl. Conf. FPGAs*, submitted, 2003.
4. C. J. Alpert and A. B. Kahng. "Recent Directions in Netlist Partitioning: A Survey". *Integration: The VLSI Journal*, 19:1–81, 1995.
5. K. Roy and C. Sechen. "A Timing-Driven n-way Chip and Multi-Chip Partitioner". In *Proc. IEEE/ACM Intl. Conf. Computer-Aided Design*, pages 240–247, 1993.
6. W. Sun and C. Sechen. "Efficient and Effective Placement for Very Large Circuits". In *Proc. IEEE/ACM Intl. Conf. Computer-Aided Design*, pages 170–177, 1993.
7. D. M. Schuler and E. G. Ulrich. "Clustering and Linear Placement". In *Proc. IEEE/ACM Design Automation Conf.*, pages 50–56, 1972.
8. H. Shin and C. Kim. "A Simple Yet Effective Technique for Partitioning". *IEEE Trans. VLSI Systems*, 1(3): 380–386, September 1993.
9. T.-K. Ng, J. Oldfield and V. Pitchumani. "Improvements of a Mincut Partition Algorithm". In *Proc. IEEE/ACM Intl. Conf. Computer-Aided Design*, pages 470–473, 1987.
10. L. Hagen, A. B. Kahng, F. J. Kurdahi and C. Ramachandran. "On the Intrinsic Rent Parameter and Spectra-Based Partitioning Methodologies". *IEEE Trans. Computer-Aided Design*, 13(1):27–37, 1994.
11. A. Singh, G. Parthasarathy and M. Marek-Sadowska. "Interconnect Resource-Aware Placement for Hierarchical FPGAs". In *Proc. IEEE/ACM Intl. Conf. Computer-Aided Design*, pages 132–136, 2001.
12. J. Dambre, P. Verplaetse, D. Stroobandt and J. Van Campenhout. "On Rent's Rule for Rectangular Regions". In *Proc. IEEE/ACM Intl. Workshop on System-Level Interconnect Prediction*, pages 49–56, 2001.
13. X. Yang, R. Kastner and M. Sarrafzadeh. "Congestion Estimation During Top-Down Placement". In *Proc. Intl. Symp. on Physical Design*, pages 164–169, 2001.
14. D. Stroobandt. "A Priori System-Level Interconnect Prediction: Rent's Rule and Wire Length Distribution Models". In *Proc. IEEE/ACM Intl. Workshop on System-Level Interconnect Prediction*, pages 3–21, 2001.
15. R. B. Hitchcock, G. L. Smith and D. D. Cheng. "Timing Analysis of Computer Hardware". *IBM Journal of Research and Development*, 26(1):100–105, January 1982.
16. B. Landman and R. Russo. "On a Pin Versus Block Relationship for Partitions of Logic Graphs". *IEEE Transactions on Computers*, c-20:1469–1479, 1971.
17. S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi. "Optimization by Simulated Annealing". *Science*, 220:671–680, 1983.
18. V. Betz, J. Rose and A. Marquardt. "Architecture and CAD for Deep-Submicron FPGAs". Kluwer Academic Publishers, 1999.
19. M. Huang, F. Romeo and A. Sangiovanni-Vincentelli. "An Efficient General Cooling Schedule for Simulated Annealing". In *Proc. IEEE/ACM Intl. Conf. Computer-Aided Design*, pages 381–384, 1986.
20. J. Lam and J.-M. Delosme. "Performance of a New Annealing Schedule". In *Proc. IEEE/ACM Intl. Design Automation Conf.*, pages 306–311, 1988.