Performance-Driven Circuit Partitioning for Prototyping by Using Multiple FPGA Chips

Chunghee Kim, Hyunchul Shin

Dept. of Electronics Eng.,
Hanyang University, Korea

Younguk Yu

Seodu Logic Inc.
Seoul, Korea

Abstract— A new performance-driven partitioning algorithm has been developed to implement a large circuit by using multiple FPGA chips. Partitioning for multiple FPGAs has several constraints to satisfy so that each partitioned subcircuit can be implemented in a FPGA chip. To obtain satisfactory results under the constraints, the partitioning is performed in two phases which are the initial partitioning for global optimization and the iterative partitioning improvements for constraint satisfaction. Experimental results using the MCNC benchmark examples show that our partition method produces better results than those of other recent approaches on the average and that performance-driven partitioning is effective in reducing critical time delays.

I. Introduction

FPGAs (Field Programmable Gate Arrays) are widely used since a circuit can be implemented simply by programming fabricated FPGAs to perform desired functions. Rapid prototyping by using FPGAs, therefore, can frequently reduce the cost and time required for design, implementation and verification of a circuit. A FPGA chip consists of configurable logic blocks (CLBs), input/output blocks (IOBs), and programmable interconnection networks. A CLB contains flipflops and lookup tables which can be configured to perform combinational or sequential logical functions [1].

The density of a FPGA chip is much less than that of other technologies such as gate array or standard cell. When the designed circuit is too large to be configured within a single FPGA chip, the circuit has to be partitioned into subcircuits so that each subcircuit can be configured within a FPGA chip. To incorporate the additional constraints, several FPGA partitioning algorithms has been published recently. [2, 3, 4, 5].

In [6], a bottom-up partitioning approach tailored to FPGA chips is described, in which functions are selected and placed on a chip, one by one. Since this method constructively places each function, the solution may not be close to the optimum one.

In [3], an iterative improvement algorithm for FPGA partitioning is reported. At each pass cells are moved between blocks and the best partition is chosen. The move is made to decrease the number of pins of source and/or destination subcircuits.

In [2], a large logic circuit is partitioned into a collection of subcircuits implementable with devices selected from a given library. Each chip in the library may have a different price, size, and terminal capacity. In [4], an improved partitioning method is described, in which the functional replication capability is added to the partitioning algorithm of [2]. The functional replication reduces the size of interconnect as well as the total chip cost.

In [5], circuit partitioning for logic emulation is studied. After local ratio-cut clustering to reduce the circuit complexity, a set covering partitioning approach is used to replace the widely adopted recursive partitioning. There are also algorithms to partition for PLA-based FPGAs [7].

In this paper, a new hierarchical performance-driven partitioning algorithm for multiple FPGA implementation is described. The algorithm consists of two phases. During the first phase, two-level hierarchical partitioning is performed to find a "good" initial solution. The constraints are not strictly enforced during the first phase. This is to allow more moves of circuit elements among subcircuits.

During the second phase, iterative optimization is performed, in which the cost function is dynamically adjusted for each pass, to find an near-optimal solution which satisfies all the constraints.

To illustrate the effectiveness of our partitioning algorithm, it is used to partition MCNC partition benchmark

examples and its results are compared with those of [2, 4, 5]. Since the benchmark circuits are already technology mapped into Xilinx 3000 device families, we can make exact comparisons without worrying about the performance of technology mapping tools. In most cases, and on the average, our method outperforms other approaches published in [2, 4, 5]. Furthermore, our method significantly outperforms others when the size of the given circuit is large.

In Section II, the FPGA partitioning problem is formally defined and the objective function is described. In Section III, the performance-driven hierarchical FPGA partitioning algorithm is described in detail. In Section IV, experimental results are shown. Finally, in Section V, conclusions are summarized.

II. FPGA Partitioning Problem

The input circuit is a netlist of circuit elements of a target FPGA chip. For example, Xilinx chips have only one type of circuit elements which are CLBs.

Let a given circuit have n nodes or CLBs and let V be the set of nodes and E be the set of nets where a net $e \in E$ connects two or more elements of V. Now the multiple FPGA partitioning problem can formally be stated as follows :

Problem : Given a hypergraph $H(V, E)$, a cost function f, and a set of l FPGA chips with size and pin count constraints (S_1, ..., S_l and P_1, ..., P_l, respectively), find a mapping $\Phi : V \rightarrow \{1, 2, ..., l\}$ such that

(i) $a_i \leq S_i$ for all i=1,2, ...,l, where $a_i = \{ v \mid \Phi(v) = i, v \in V \}$ (size of each node (CLB) is 1.)

(ii) $b_i \leq P_i$ for all i=1,2,..., l, where b_i is the number of nets connecting a node in subcircuit i and another node in subcircuit $j \neq i$.

(iii) the given cost function f is minimized.

The cost can be written as the weighted sum of the number of nets cut by partitioning and the delays on the timing critical paths. Specifically,

$$f = cut\_net + r \times delay\_cost$$

where r is the weighting factor.

Previous partitioning methods [2, 3, 4, 5, 6] did not consider delays during partitioning. However, one of the major drawbacks of FPGAs compared to custom chips is that the speed of FPGA chips is much slower due to programmability. Sometimes slow speed of FPGAs prohibits FPGAs from being used in real time applications. Therefore, time delays should be considered and excessive delays should be penalized during partitioning. In our approach, delays on critical paths are estimated and are considered during clustering and partitioning. The best way to minimize delay of a timing critical path is to put all the elements (nodes) on the path into a chip (subcircuit). Therefore, it is reasonable to merge several "closely" connected elements into a cluster and to consider the cluster as an object during the first phase of partitioning. The "closeness" between two connected elements (nodes) is represented by an edge weight.

For example, in Fig 1, let (a, b, c, d, e) be the most timing critical path, then the edges belong to this path have weight $w1$. If the second critical path is (f, c, d, e), then the edge (f, c) has weight $w2$, where $w1 > w2$. The weights are determined to normalize the criticality of paths as shown in Fig 2.

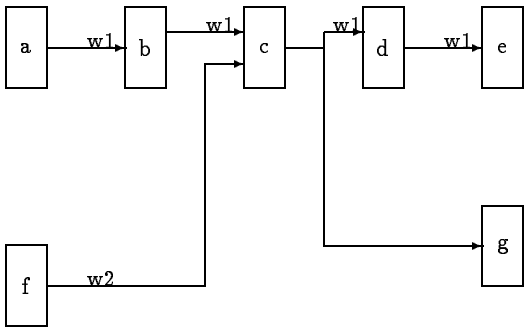

Fig. 1. An example of path weighting

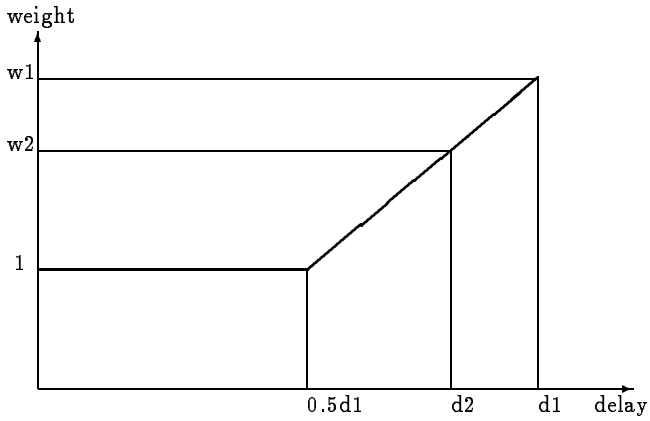

Fig. 2 : Net weight vs. path delay

III. Performance-Driven Hierarchical FPGA Partitioning

The new partitioning method attempts to minimize the cost while satisfying constraints on the number of I/O terminals and on the size of chips. Partitioning algorithm

consists of two phases. In first phase, clustering-based partitioning which is modified from [8] is performed. The objective function of the first phase is the weighted sum of the cut size and the maximum delay. Since we iteratively move nodes from a subset to another as in [8] to minimize the objective function, the result may be dependent on the initial partition. To remove this dependency and to obtain consistently "good" solution, we partition the clusters N1 times from the randomly generated initial partition and then choose the best result. The best result is flattened and optimized once more to finish the initial partitioning phase.

During the second phase, the partition result optimized during the first phase is improved to satisfy all constraints implied by the given FPGA chips. The overall algorithm of partitioning is given in Algorithm 1.

Algorithm 1 : Multi-Chip Partitioning

```
/** First phase **/
make_clusters();
Perform cluster partition N1 times;
flatten the best result obtained from the above;
partition CLBs by GCEP;
/** Second phase **/
while( iteration number is less than 3 or the partition
       is improved in previous 3 iterations ) {
  improve_current_partition_result();
}
```

3.1 Clustering-Based Initial Partitioning

Clustering of "closely connected" CLBs improves both of the efficiency and the performance of partitioning [8, 9], since closely connected CLBs should be assigned to a single FPGA chip to minimize the cut size and delay. The clustering is bottom-up. The closeness of two CLBs or clusters, C and D, is evaluated by the following formula:

$$closeness(C,D) = cnet(C,D)/MIN(net(C), net(D)) - \alpha \times (cl\_size(C,D)/avg\_size)$$

where the variable $cnet(C,D)$ is the sum of common net weights between C and D, and $net(C)$ is the sum of net weights in C. The variable $cl\_size(C,D)$ is the size of the new cluster constructed when C and D are merged. The variable avg_size is the average size of clusters. The first term represents the attractive force due to common nets between C and D, and the second term represents the repulsive force to encourage forming uniformly sized clusters.

For cluster and CLB partitioning, the gradual constraint enforcing partitioning (GCEP) method developed by the authors is adopted [8]. At the beginning, the size constraints are relaxed and the sizes of partitioned subsets may be severely unbalanced. This allows very flexible movements of nodes among the subsets. After each pass of optimization, the size constraints are tightened gradually so that the desired size constraints are enforced at the final pass. Thus the GCEP algorithm has the hill-climbing effect and searches a broader solution space.

Our present partitioning algorithm is improved from the one in [8] in the following respects : (1) The nodes are moved from a selected subset to other subsets, as long as size constraints are satisfied, in [8]. However, in the present algorithm, nodes are moved out of the selected subset to other subsets and then moved into the selected subset from others to allow more general bidirectional movements. (2) Each node movement is stored with its corresponding cost and the minimum cost partition is chosen among feasible ones after each iteration, in the present implementation. (3) Cut-net is minimized in [8], while the weighted sum of cut-net and critical delay is minimized in the present implementation.

The following is the partition algorithm used for both of cluster and CLB partitioning. A cell is a cluster of CLBs for cluster partition and is a CLB for CLB partition. K is the number of chips (or groups in the algorithm).

Algorithm 2 : GCEP Partitioning

```
initial_partition;
evaluate_initial_gain;
curr_group = 1; /* Initial current group */
bal = INIT_BAL; /* INIT_BAL = 0.5 */
dff_size = avg_size × bal;

/* Main optimization */
for( True ) {
  /* Move out as long as the size of curr_group
     is not less than (avg_size - dff_size) */
  move_out( curr_group, dff_size );
  /* Move in as long as the size of curr_group
     is not greater than (avg_size + dff_size) */
  move_in( curr_group, dff_size );
  /* Check move_history and find the optimal point.
     Then undo moves made after the point */
  unmove;
  curr_group ++;

  /* If one iteration is finished */
  if( curr_group == K+1 ) {
      curr_group = 1;
      bal = bal × REDUCTION_RATIO;
```

```
            /* REDUCTION_RATIO = 0.9 */
        if( bal ≤ FINAL_BAL )
          break; /* Partition completed */
        dff_size = avg_size × bal;
        if( dff_size < avg_size × FINAL_BAL )
            /* FINAL_BAL = 0.2 */
          dff_size = avg_size × FINAL_BAL;
    }
}
```

```
move_out( from_group, dff_size ) {
    /* from_group is curr_group */
    while( ( cell = find_cell_with_largest_gain such that
            | to_group | + | cell | ≤ avg_size+dff_size )
            is not empty ) {
        if( | from_group | - | cell | < avg_size-dff_size )
            return;
        move_cell;
        update_gain;
        store move_history with gain;
    }
}
```

```
move_in( to_group, dff_size ) {
    /* to_group is curr_group */
    while( ( cell = find_cell_with_largest_gain such that
            | from_group | - | cell | ≥ avg_size-dff_size )
            is not empty ) {
        if( | to_group | + | cell | > avg_size+dff_size )
            return;
        move_cell;
        update_gain;
        store move_history with gain;
    }
}
```

3.2 Partitioning Improvement Step

The partition improvement is performed based on the gain (cost reduction) of moving a CLB from a chip to another. The movements are made in the decreasing order of gain. Violations of the constraints on the numbers of CLBs and pins are reflected to the cost by adding penalty terms. When there are violations left at the end of an iteration, the penalty values for the violated constraints are increased. When the violated result is not improved during three consecutive iterations, the algorithm reports a failure and terminates. The cost function including the penalty terms can be written as

$$cost = cut\_net + r \times delay\_cost + \alpha \times CLB\_overflow + \beta \times Pin\_overflow$$

The first and second terms are explained in Section 3.1. The default value of r is 1 and the user may change it if desired. The values of α and β are dynamically adjusted. The third and fourth terms will become zero when the constraints on the number of CLBs and on the number of pins are satisfied.

The partition improvement algorithm is very similar with the GCEP partitioning algorithm given in Algorithm 2. The differences are in that (1) bal is fixed ($bal = FINAL\_BAL$), (2) α is increased by 1 if CLB_count is violated for a subcircuit(group) and β is increased by 1 if *pin_count* is violated, after execution of the main optimization 'for' loop, (3) CLBs are moved only if gain is positive in procedures, *move_in* and *move_out*, and (4) the main optimization loop terminates when violation or cost is not improved during three consecutive iterations, in the partition improvement procedure. The initial values of α and β are 1. Experimental results show that performance-driven partitioning can be completed after 2-4 iterations. Table 1 shows partition of the largest benchmark circuit, s38584, with 2904 CLBs and 292 I/Os. The circuit is partitioned into 22 FPGA chips. Each chip has 144 CLBs and 96 I/O pins. The total $CLB\_overflow$ is 54 and the total $Pin\_overflow$ is 1 for the initial solution. After the first improvement iteration, there are still violations, therefore $\alpha = 2$ and $\beta = 2$ are used for iteration 2. After iteration 3 ($\alpha = 3, \beta = 2$), a feasible partition has been obtained.

Table 1: Partition improvement for s38584

Iteration	α	β	CLB overflow	Pin overflow	cut_net
Initial	-	-	54	1	770
1	1	1	34	1	748
2	2	2	1	0	790
3	3	2	0	0	791

IV. Experimental Results

The new partitioning algorithm has been implemented by using "C" programming language on DEC3000 workstation running UNIX operating system.

Table 2: Xilinx Chip Types.

Chip Type	Device	Cost	#IOBs	#CLBs
1	XC3020	1.00	64	64
2	XC3030	1.36	80	100
3	XC3042	1.84	96	144
4	XC3064	3.15	110	224
5	XC3090	4.83	144	320

Table 3: Benchmark Examples.

Circuit	#CLBs	#IOBs	#NETs	#PINs
c3540	283	72	489	1645
c5315	377	301	699	2409
c6288	833	64	1472	3438
c7552	489	313	921	2924
s5378	381	86	628	2332
s9234	454	43	716	2671
s13207	915	154	1377	5309
s15850	843	102	1265	4977
s38584	2904	292	3884	17483

The MCNC benchmark circuits are technology-mapped for Xilinx 3000 series FPGAs. In Table 2, the types of Xilinx 3000 series chips are listed with the device names, the device costs, the number of IOBs and the number of CLBs. Benchmark circuits are shown in Table 3. Chip types and the benchmark examples listed in Table 2, 3 are also used in [2, 4].

4.1 Partitioning for a Minimal Cost

First, we compare our partition results with those of [2, 4] based on the total chip cost needed to implement each circuit. Since several types of chips were used in [2, 4] for a circuit, the number of devices used are listed in the column "Device Distribution" in Table 4. We partitioned the same circuits by using only one type of chips. After partitioning, if it is possible to substitute a chip by a cheaper one, then it is substituted. The initial chip type was determined experimentally for minimal cost. Our method has produced about 12% better results on the average than those of [2, 4]. Furthermore, our method significantly outperforms other methods when the size of the given circuit is large.

To compare with the results of [5], we used only type 5 (XC 3090). The minimum number of chips required to implement each circuit is shown in Table 5. "RFM" is the recursive Fiduccia-Mattheyses method [10] and its results

Table 5: Partitioning Results

Circuit	RFM #FPGAs	[5] #FPGAs	NEW #FPGAs
s15850	4	3	3
s13207	7	6	4
s38417	12	10	8
s38584	17	14	14
Total	40	33	29

are quoted from [5]. Again, our method (NEW) produces substantially better results.

4.2 Partitioning for Minimal Delay

Since there has not been a report on performance-driven FPGA partitioning, we can not make comparisons with other results. Therefore, we present our results with and without delay optimization. For delay optimization, the delay-cost is added to the cut-net cost and the total cost is minimized. For delay estimation, we assume that CLB-to-CLB delay is 4 ns and chip-to-chip delay is 20 ns. The results are summarized in Table 6. In the table, Delay lower bound shows the achievable minimum delay of the longest critical path, i.e., this is the delay when all CLBs are configured within a single FPGA chip. Then the maximum delays are shown when the devices in the NEW column of Table 4 are used, with and without performance optimization. These results show that delays can be reduced by using the performance-driven approach without increasing the chip cost. The last column shows the maximum delays when all the devices are replaced by type 5 (XC3090). Since larger chips are used, the delay can be significantly reduced.

V. Conclusions

The problem of partitioning a large circuit for multiple FPGA implementation is considered. In addition to finding a feasible partitioning, the objective function is chosen to reduce the total cost of devices used in the partitioning and the delay on the critical paths. Experimental results show that new method outperforms other approachs [2, 4, 5] by more than 10% and it can reduce the maximum delay significantly when performance-driven optimization is used.

Table 4: Partitioning Results

Circuit	[2]		[4]	NEW		
	Device Distribution	Total Cost	Total Cost	Device Distribution	Total cost	CPU (sec)
c3540	{0, 0, 3, 0, 0}	5.52	4.56	{0, 3, 0, 0, 0}	4.08	9
c5315	{2, 1, 2, 0, 0}	7.03	6.92	{0, 6, 0, 0, 0}	8.16	11
c6288	{0, 0, 4, 2, 0}	13.66	13.76	{0, 0, 6, 0, 0}	11.04	38
c7552	{0, 0, 4, 0, 0}	7.36	7.36	{0, 6, 0, 0, 0}	8.16	18
s5378	{0, 0, 1, 0, 1}	6.67	6.19	{0, 0, 0, 2, 0}	6.30	14
s9234	{0, 0, 0, 1, 1}	7.98	7.98	{0, 1, 3, 0, 0}	6.88	26
s13207	{3, 5, 4, 0, 0}	17.16	18.12	{1, 11, 0, 0, 0}	15.96	375
s15850	{0, 0, 2, 2, 1}	14.80	14.97	{0, 10, 0, 0, 0}	13.60	272
s38584	{0, 5, 15, 4, 1}	51.83	51.19	{0, 1, 21, 0, 0}	40.00	1308
Total		132.01	131.05		114.18	

Table 6: Delays on Critical Paths.

Circuit	Delay lower bound	Chip types in Table 4		For Chip type 5 (XC3090) perform. opt.
		Without perform. opt.	With perform. opt.	
c3540	132	172	172	168
c5315	88	168	128	108
c6288	396	620	532	456
c7552	84	164	136	116
s5378	72	132	84	84
s9234	56	124	92	84
s13207	72	212	180	144
s15850	72	244	216	136
s38584	68	204	160	132
Total (%)	1040 (100)	2040 (196)	1700 (163)	1428 (137)

References

[1] Xilinx, Inc., The Programmable Gate Array Data Book, Xilinx, San Jose, 1994.

[2] R. Kuznar, F. Brglez and K. Kozminski, "Cost Minimization of Partitioning into Multiple Device," Proc. of 30th Design Automation Conference, pp. 315-320, 1993.

[3] N.-S. Woo and Jaeseok Kim, "An Efficient Partitioning Circuits for Multiple-FPGA Implementation," Proc. of 30th Design Automation Conference, pp. 202-207, 1994.

[4] R. Kuznar, F. Brglez and B. Zajc, "Multi-way Netlist Partitioning into Heterogeneous FPGAs and Minimization of Total Device Cost and Interconnect," Proc. of 31th Design Automation Conference, pp. 238-243, 1994.

[5] N.-C. Chou, L.-T. Liu, C.-K. Cheng, W.-J. Dai, and R. Lindelof, "Circuit Partitioning for Huge Logic Emulation Systems," Proc. of 31th Design Automation Conference, pp. 244-249, 1994.

[6] W. O. McDermith, "A Bottom-Up Approach to FPGA Partitioning," in Proc. IEEE Custom Integrated Circuits Conference, pp. 5.4, 1992.

[7] Z. Hasan, D. Harrison and M. Ciesielski, "A Fast Partitioning Method for PLA-Based FPGAs," IEEE Design and Test of Computers, pp. 34-39, Dec., 1992.

[8] H. Shin and C. Kim, "A Simple Yet Effective Technique for Partitioning," IEEE Trans. on VLSI Systems, Vol. 1, No. 3, pp. 380-386, Sep. 1993.

[9] C. W. Yeh and C. K. Cheng, "A General Purpose Multiple way Partitioning Algorithm", Proc. 28th Design Automation Conference, pp. 421-426, 1991.

[10] C. M. Fiduccia and R. M. Mattheyses. "A Linear-Time Heuristic for Improving Network Partitions", Proc. 19th Design Automation Conference, pp. 175-181, 1982.