[14] *XILINX 2000/3000 Development System Reference Guide: The XNFMAP Program.* 2100 Logic Drive, San Jose, CA 95124, 1991.

[15] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: A Multiple-Level Logic Optimization System," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. CAD-6, pp. 1062–1081, Nov. 1987.

[16] K. Karplus, "Xmap: a technology mapper for table-lookup field programmable gate arrays," in *ACM IEEE* 28^{th} *Design Automation Conference Proceedings*, (San Franciso, California), pp. 240–243, June 1991.

[17] *EXEMPLAR: XNFOPT.* 2550 Ninth Street, Suite 102, Berkeley, CA 94710, 1990.

References

[1] K. A. El-Ayat, A. El Gamal, R. Guo, J. Chang, R. K. Mak, F. Chiu, E. Z. Hamdy, J. McCollum, and A. Mohsen, "A CMOS electrically configurable gate array," *IEEE Journal of Solid-State Circuits*, vol. 24, pp. 752–762, June 1989.

[2] XILINX*: The Programmable Gate Array Data Book.* 2100 Logic Drive, San Jose, CA 95124, 1991.

[3] R. Murgai, Y. Nishizaki, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Logic synthesis for programmable gate arrays," in *ACM IEEE* 27^{th} *Design Automation Conference Proceedings*, (Orlando, Florida), pp. 620–625, June 1990.

[4] R. Murgai, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Improved logic synthesis algorithms for table look up architectures," in *IEEE International Conference on Computer-Aided Design ICCAD-91*, (Santa Clara, California), pp. 564–567, November 1991.

[5] N.-S. Woo, "A heuristic method for FPGA technology mapping based on the edge visibility," in *ACM IEEE* 28^{th} *Design Automation Conference Proceedings*, (San Franciso, California), pp. 248–251, June 1991.

[6] R. J. Franics, J. Rose, and K. Chung, "Chortle: A technology mapping program for lookup table-based field programmable gate arrays," in *ACM IEEE* 27^{th} *Design Automation Conference Proceedings*, (Orlando, Florida), pp. 613–619, June 1990.

[7] R. J. Franics, J. Rose, and Z. Vranesic, "Chortle-crf: Fast technology mapping for lookup table-based FPGAs," in *ACM IEEE* 28^{th} *Design Automation Conference Proceedings*, (San Franciso, California), pp. 227–233, June 1991.

[8] F. Dresig, O. Rettig, and U. G. Baitinger, "Logic synthesis for universal logic cells," in *Proceedings of International Workshop on Field Programmable Logic and Applications*, (Oxford, England), 4–6 September 1991.

[9] D. Filo, J. Yang, F. Mailhot, and G. Micheli, "Technology Mapping for a Two-Output RAM-based Field-Programmable Gate Arrays," in *European Design Automation Conference*, pp. 534–538, February 1991.

[10] J. Rose and S. Brown, "Flexibility of interconnection structures for field-programmable gate arrays," *IEEE Journal of Solid-State Circuits*, vol. 26, pp. 277–282, Mar. 1991.

[11] *XILINX Application Note: Fundamentals of Placement and Routing.* 2100 Logic Drive, San Jose, CA 95124, 1990.

[12] A. El Gamal, "Two-Dimensional Stochastic Model for Interconnections in Master Slice Integrated Circuits," *IEEE Transactions on Circuits and Systems*, vol. 28, pp. 127–138, Feb. 1981.

[13] R. J. Franics, J. Rose, and Z. Vranesic, "Technology mapping for lookup table-based fpgas for performance," in *IEEE International Conference on Computer-Aided Design ICCAD-91*, (Santa Clara, California), pp. 568–571, November 1991.

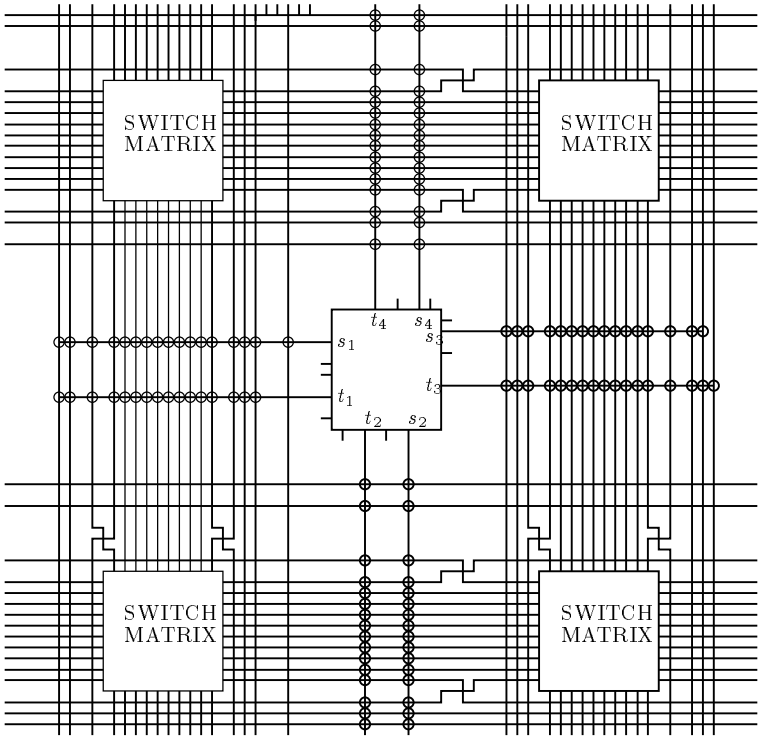


Figure 5: The 4000 series routing.

Because of its simplicity and generality, `Rmap` can be easily extended to other LUT-based architectures by adjusting the cost function to reflect the characteristics of the FPGA architecture. For example, in the XC3000 series the flip-flop outputs of a cell can be internally routed to the cell's LUT. The cost function can be easily adjusted to reward pairings which exploit this internal routing resource.

In the XC4000, the cells can again accommodate two 4-input LUTs (plus a third LUT which can combine the outputs of the two 4-input LUTs with another input). The two 4-input LUTs can have completely disjoint inputs in the 4000 series. However as shown in Figure 5, routing resources are used most efficiently when these two tables share inputs. This is because when the input (t_i) of one table is obtained from one of the adjacent wire segments, the other table can connect the input s_i to the same segment (except in one case) without consuming any additional routing resources.

Acknowledgments

The first and second authors were supported by the National Science Foundation Presidential Young Investigator Grant No. MIP-8896276. The third author was supported in part by the National Science Foundation Research Initiation Award Grant No. MIP-9111607. The authors wish to thank Xilinx Inc. for donating their XACT XC3000/2000 Sparc Design Implementation Systems in support of this research.

It appears from the pins per net and pins per cell ratios of these four difficult-to-route designs and their `apr` results, that routability is enhanced when these ratios are reduced and balanced.

Circuit	Time	#Nodes		#Potential	#Potential	#CLBs
	Sparc 1+	Before	After	Blocks	Pairs	
5xp1	5.87s	111	113	582	21746	20
9sym	13.02s	152	160	722	30742	46
9symml	17.48s	156	169	790	40298	46
C499	651.08s	455	455	10126	592823	71
C880	65.37s	357	372	2445	127073	82
alu2	96.62s	306	321	2383	157713	91
alu4	606.00s	564	597	5965	635497	167
apex1	259.08s	509	619	3073	234268	241
apex5	425.87s	743	783	2718	352780	206
apex6	432.02s	745	753	2153	292114	176
apex7	17.80s	252	256	830	47175	45
apex7b	17.78s	252	256	830	47175	45
b9	2.03s	144	144	301	7781	25
duke2	34.58s	269	278	1286	62107	85
e64	1.33s	226	226	187	2863	53
misex1	0.55s	51	54	182	2149	9
misex2	0.45s	88	94	129	1583	20
misex3	86.13s	344	358	1757	110715	115
rd73	6.25s	97	100	788	19478	18
rd84	25.37s	164	174	1345	58318	40
rot	178.03s	695	701	2246	216893	137
vg2	1.03s	79	97	245	4363	21
z4ml	0.68s	41	41	280	3862	4

Table 6: Time and memory requirements for `Rmap`.

Table 6 gives the time and memory requirements for `Rmap` on the benchmarks. The number of nodes in the network are given before and after the node-splitting routine. The number of potential blocks appears reasonable except for *C499*. Typically, `Rmap` is 5 times slower than `Chortle-crf`. However, it should be noted that these times are small when compared to the time required to place and route even the small mapped designs.

Conclusions

We have presented a new algorithm for technology mapping of combinational circuits for LookUp Table-based Field-Programmable Gate Arrays. This new algorithm treats covering, pairing, and replication under the same framework rather than applying local optimizations. The mapper, `Rmap`, is very general and has the flexibility of trading routability with compactness of a design. It is able to produce routable mappings on benchmarks for which `Chortle-cfr` and `xnfmap` do not. The routability issue in LUT-based technology mapping is discussed and three different mappers are evaluated with respect to routability.

`Rmap` produces a bit more routable design, in terms of average numbers of unrouted pins and nets, than `xnfmap` and `chortle`. For the benchmark circuit *duke2*, `xnfmap` outperforms the others; and finally for *misex3*, again `Rmap` produces a bit closer to routable design than the other mappers.

The results in Table 4 raise the question of whether the routings can be completed by simply adjusting the parameters c_I and c_O properly? This is indeed the case as illustrated in Table 5, which shows that the number of unrouted pins (using `apr`) in these designs can actually be brought down to zeros by appropriate values of c_I and c_O. A routed mapping for the circuit *alu4* was obtained with $c_I = 2.1$ and $c_O = 1.0$, despite the discouraging initial result of 85 unrouted pins obtained with $c_I = c_O = 0$.

Circuit	`Chortle-crf`				`Rmap` $c_I = c_O = 0$				`Rmap` $c_I = c_O = 1$				`Xnfmap 3.13`			
	average		minimum		average		minimum		average		minimum		average		minimum	
	pins	nets	pins	nets	pins	nets	pins	nets	pins	nets	pins	nets	pins	nets	pins	nets
alu2*	12.1	7.4	6	6	15.3	10.8	5	5	3.1	2.9	0	0	10.9	8.1	4	3
duke2*	27.9	15.0	16	11	18.5	12.5	6	5	5.1	4.1	2	2	3.3	2.9	1	1
misex3*	26.6	17.1	7	6	27.0	19.7	18	13	8.3	6.1	1	1	14.7	9.5	5	5
alu4*			29	21			85	62			36	19			52	30

Table 4: Number of unrouted pins and nets after 10 `apr` runs (except for *alu4*); *alu2* is routed on a 3042PG132, *duke2* is routed on a 3042PQ100, *misex3* is routed on a 3064PG132, *alu4* is routed on a 3090PC84 only once.

`Rmap`	$c_I = c_O = 0$					$c_I = c_O = 1$					$c_I = 0$ $c_O = 2$					$c_I = 1.5$ $c_O = 1$				
Circuit	# CLB	P/C	P/N	Unrouted pins	nets	# CLB	P/C	P/N	Unrouted pins	nets	# CLB	P/C	P/N	Unrouted pins	nets	# CLB	P/C	P/N	Unrouted pins	nets
alu2	91	5.66	3.54	5	5	110	4.70	3.68	0	0	109	4.79	3.86	0	0	114	4.68	3.56	0	0
duke2	85	4.44	3.47	6	5	105	3.92	3.62	2	2	113	3.87	3.80	5	3	118	3.68	3.42	0	0
misex3	115	5.41	3.69	18	13	131	4.79	3.77	1	1	147	4.58	4.17	0	0	146	4.48	3.57	0	0
alu4	167	5.81	3.52	85	62	194	4.92	3.59	36	19	212	4.66	3.80	38	31	213	4.65	3.49	22	12

Table 5: CLBs, Pin-to-cell (P/C), Pin-to-net ratios (P/N), and number of unrouted pins and nets after ten `apr` runs; *alu2* is routed on a 3042PG132, *duke2* is routed on a 3042PQ100, *misex3* is routed on a 3064PG132, *alu4* is routed on a 3090PC84 only once.

We made several observations from Table 5:

1. Positive c_I and c_O values deliver less compact but more routable mappings.
2. Increasing c_I or c_O can generate more routable mappings.
3. Continuing to increase the values of c_I or c_O increases utilization of the devices, but will not impair the routabilities of the mappings.

communication.

In the total number of CLBs summed over the 19 implementable benchmarks `Chortle-crf` had 12 fewer CLBs than `Rmap` with constants $c_I = c_O = 0$, 108 fewer CLBs than `Xnfmap`. However in general, although `Xnfmap` produces mapped designs with more CLBs, they are rather balanced in terms of pins per cell and pins per net ratios. We also tried `Xnfmap -s` to attempt to squeeze more functions into CLBs, but this didn't improve the CLB counts.

	`Chortle-crf`		`Rmap` $c_I = c_O = 0$		`Rmap` $c_I = c_O = 1$		`Xnfmap 3.13`	
Circuit	Pins/Cell	Pins/Net	Pins/Cell	Pins/Net	Pins/Cell	Pins/Net	Pins/Cell	Pins/Net
5xp1	4.00	4.43	3.78	3.89	3.48	3.86	3.83	3.65
9sym	5.21	4.82	5.55	3.53	4.80	3.70	5.03	3.66
9symml	5.21	4.64	5.54	3.56	5.03	3.57	4.97	3.55
alu2*	5.34	4.58	5.66	3.54	4.70	3.68	5.08	3.60
alu4*	5.51	4.64	5.81	3.52	4.92	3.59	5.15	3.40
b9	2.56	2.85	2.50	2.62	2.29	2.60	2.31	2.63
duke2*	4.27	4.27	4.44	3.47	3.92	3.62	3.90	3.46
misex1	3.33	3.75	3.28	3.04	3.04	2.93	3.23	3.34
misex2	2.84	3.51	2.63	2.81	2.55	2.80	2.57	2.78
misex3*	5.22	4.56	5.41	3.69	4.79	3.77	4.82	3.59
rd73	4.53	4.14	4.39	3.24	4.10	3.32	4.32	3.58
rd84	5.0	4.59	5.08	3.47	4.73	3.56	4.76	3.55
vg2	2.81	3.17	3.11	2.58	2.73	2.54	2.69	2.80
z4ml	1.86	2.00	2.27	2.27	2.19	2.33	1.86	2.00

Table 3: Pins per cell (local congestion), and pins per net (global congestion) of different mappers

We ran `Rmap` with two sets of parameters to demonstrate the flexibility of the mapper, as illustrated in Table 3. With the parameters of the cost function (equation 1) set to $c_I = c_O = 0$, `Rmap` produces comparatively fewer CLB counts and performs very well in reducing global congestion. However, it also has the poorest local congestion grades. With the weights of the cost function set to $c_I = c_O = 1$, `Rmap` produces comparatively more CLB counts than with $c_I = c_O = 0$, however it scores well in both reducing global and local congestions. `Chortle`'s performance is average on CLB counts and poor on global congestion.

To validate the use of these ratios as routability criteria, we conducted another set of experiments. The unrouted circuits are the most interesting ones in this context. In some sense, these circuits indicate the routability of the mappers. Again, routability is defined to be the *probability* of automatic completion of a design. It is the "probability" because there are degrees of randomness in the simulated annealing-based placement algorithm.

We ran `apr` ten times and measured the average and the minimum number of unrouted pins and nets to assess the routability of the mappers. The minimum number of unrouted pins and nets indicates how close a design is from being routed. A partially routed design with one or two unrouted nets or pins is considered to be acceptable. Often the routing can be completed with additional routing iterations or human intervention. The results are tabulated in Table 4. We demonstrate that by simply setting the parameters of the cost function to $c_I = c_O = 1$ in `Rmap`, we could trade (and control) routability for compactness of a design. For the benchmark circuit *alu2*,

Circuit	Chortle-crf			Rmap $c_I = c_O = 0$			Rmap $c_I = c_O = 1$			Xnfmap 3.13		
	#CLB	#Pin	#Net	#CLB	#Pin	#Net	#CLB	#Pin	#Net	#CLB	#Pin	#Net
5xp1	24	164	37	20	140	36	21	139	36	24	157	43
9sym	51	318	66	46	311	88	54	307	83	54	322	88
9symml	47	297	64	46	310	87	51	307	86	55	323	91
alu2*	92	577	126	91	606	171	110	592	161	99	584	162
alu4*	154	970	209	167	1099	312	194	1063	296	180	1030	303
apex1+	227	1505	351	241	1624	445	290	1619	432	246	3182	865
apex4+	456	2832	567	491	3248	887	526	3103	863	526	6589	1810
apex5+	202	1395	354	206	1558	486	233	1526	458	202	3185	958
apex6+	165	1219	347	176	1337	408	205	1325	403	171	2939	926
apex7	45	368	114	45	379	130	57	380	129	51	366	120
apex7b	45	368	114	45	379	130	57	380	129	51	366	120
e64	55	462	146	53	450	151	53	406	159	63	383	160
C499	50	379	115	71	524	181	79	509	176	70	399	121
C880	76	522	146	82	620	209	103	610	199	76	530	178
b9	26	225	79	25	220	84	29	211	88	28	208	79
duke2*	85	581	136	85	604	174	105	612	169	89	546	158
misex1	12	90	24	9	77	26	11	79	27	15	97	29
misex2	25	193	55	20	166	59	23	168	60	22	167	60
misex3*	117	757	166	115	774	210	131	761	202	128	752	209
rd73	22	145	35	18	123	38	20	123	37	24	147	41
rd84	44	280	61	40	264	76	43	260	73	50	295	83
rot+	148	1072	324	137	1139	378	169	1127	370	146	2491	788
vg2	19	146	46	21	168	65	22	150	59	19	140	50
z4ml	3	26	13	4	34	15	5	35	15	3	26	13

Table 2: Numbers of CLBs, pins and nets generated by three different mappers; all designs were routed without difficulty except those marked with $*$'s (see Table 4) and those marked with +'s which are not implementable, e.g., *apex6* has 234 I/O pins

All mappers use identical package types for placement and routing, the package types are chosen to maintain approximately 80% cell utilization. We used the following package types in our experiments: XC3030, XC3042, XC3064, and XC3090(only for *alu4*).

The MCNC benchmark circuits are minimized with `misII2.0` using the standard script once. The minimized circuits are then mapped by three different mappers: `Chortle-crf`, `Rmap`, and `Xnfmap`. All the mapped designs are converted to `.map` format, and then translated to `.lca` format by `map2lca` which uses a mincut approach to obtain an initial placement. The final step is the automatic placement and routing by `apr`.

Small designs were placed and routed with the default settings without difficulty. The *unrouted* designs are marked with asterisks in Tables 2 and 3.

Note that the numbers of CLBs produced by `chortle-crf` differ from the ones reported in [7] for two reasons. The first reason is that the authors of `chortle-crf` used the `misII2.0` standard script *plus* an additional simplification step in `misII` to minimize the circuits before mapping them. The second reason is that they revised the published results after the publication through private

Device	XC3020	XC3030	XC3042	XC3064	XC3090
Total CLBs	64	100	144	224	320
80% × Total CLBS	51	80	115	180	256

Table 1: Package types and CLBs available

2. Decompose by internal common dependency sets.

For each child c_i of node x we compute $g(c_i)$ which is the number of children of x which have at least one child in common with c_i. Note that $g(c_i)$ will be at least 1 since c_i is included in the count. We pick the child c_i with the largest $g(c_i)$ breaking ties arbitrarily. If $1 < g(c_i) < f$ then we create a node, new, whose children are exactly those counted in $g(c_i)$ (the children of x having one child in common with c_i, including c_i). These new children of new are removed as children of x and new becomes a child of x. If $g(c_i) = f$ then we create a new node, new, with children c_1 and c_2, remove c_1 and c_2 as children of x and add new as a child of x. If $g(c_i) = 1$ nothing is changed.

3. Decompose by building balanced feasible subtrees.

The set of children of x, $C(x)$ is partitioned arbitrarily into three sets of roughly equal size (differing by at most one), $C(x) = S_1 \cup S_2 \cup S_3$. Three new nodes, new_1, new_2, and new_3 are created with S_1, S_2, S_3 as their sets of children, respectively. These three new nodes become the children of x: $C(x) = \{new_1, new_2, new_3\}$.

The first technique is applied to all nodes with four or more children until the fanout is three or less, or it fails. The technique is also applied to any new nodes created as a result of the splitting of their parents. If there are still nodes of four or more children at this point, then the second technique is applied in the same manner. Finally, any remaining nodes with four or more children are divided into balanced trees of degree three using the last technique.

Creating all blocks can be memory intensive. However creating only feasible blocks for a small value of k is manageable. In Section 5 the number of feasible blocks and the number of potential pairings are given in Table 6. This method is more general than previous approaches because it considers replication, covering, and cell packing within one framework, allowing these operations to interact. The heuristics in `Rmap` are the node-splitting and the selection of pairs to commit. Experimentation with different pair selection criteria is simple and convenient in this framework. Time permitting, even an exhaustive backtracking approach to pair selection could be performed to obtain the optimal mapping (for a fixed node-splitting and logic network). We shall present some experimental results in the next section.

5 Experimental results

Due to their size and scope, the experiments are conducted using only three different mappers, and an impartial subset of the standard MCNC combinational circuit benchmarks. The selected benchmark circuits have numbers of CLBs and I/O pins feasible for realization with the XC3000 packages. Part of the experiments involved using the industrial placement and routing tool `apr` with the default settings.

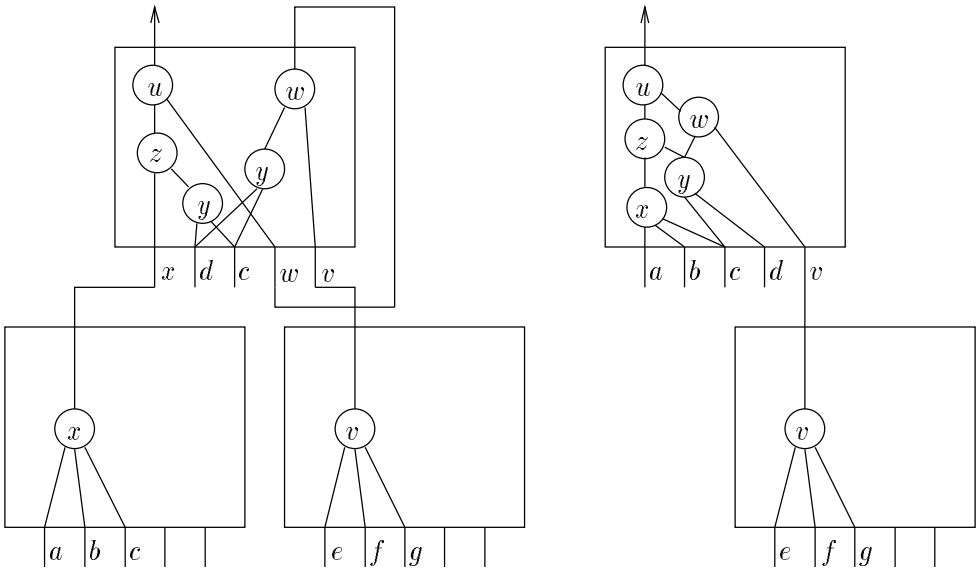


Figure 4: Mappings for the network in Fig. 1 obtained for $c_I = c_O = 0$ (left) and $c_I = c_O = 1$ (right).

Which nodes of the paired blocks to remove is determined by a traversal originating at the block outputs which decrements the outdegree of each node visited, and visits a node's children only when the outdegree of that node reaches zero. All nodes in the block whose outdegrees reach zero are removed from the network. What is left behind is a network where each of the blocks have been collapsed to single nodes and any sub-networks of the blocks which are not dominated by the block outputs have been replicated. Figure 3 shows the resulting network after committing a pair containing the block with nodes w and y, and the empty block. Node w becomes a primary input, w is removed while y remains, and nodes x and v become primary outputs.

Figure 4 contains the two cell packings of the network in Figure 1 obtained for two sets of parameters c_I and c_O in the cost function.

The four steps presented above will produce a covering and cell packing if the initial network can be covered by feasible blocks. If the dependency set of every block of a node x has size greater than k, then x cannot be included in any feasible block. In `Rmap`, node-splitting is a pre-processing step which attempts to increase the chances of finding pairable blocks. We use three techniques to split a node x with $f > 3$ children. Here, $C(x) = \{c_1, \ldots, c_f\}$, denotes the set of children of node x.

1. Decompose by external common dependency sets.

If there is a proper subset, S, of children of a node x, $S \subset C(x)$ which is exactly the set of children of a node y ($S = C(y)$), then a node, new, with child set S is created and x's new children become $\{new\} \cup (C(x) - S)$. If there is more than one choice for S, one of maximum size is selected arbitrarily.

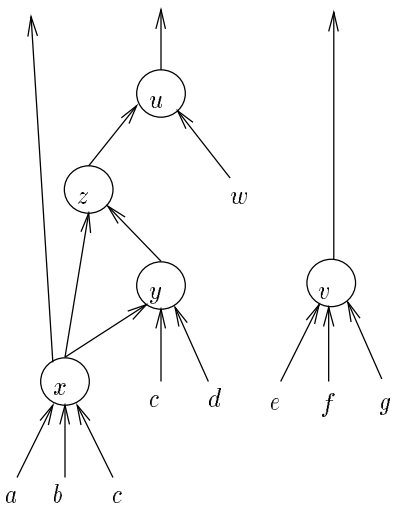


Figure 3: Resulting network after committing the block with nodes w and y.

is chosen. The number of input pins is the size of the union of the dependency sets of the two blocks and the number of output pins is 2 unless one of the blocks is the empty block (in other words, it is 1 if only one block is packed into the cell). The input parameters c_I and c_O are varied to affect the routing characteristics of the mapping. Their effects are described in the next section. Since there is quite a bit of overlap among our blocks, the choice of pairing to commit will affect the remaining available pairings, and so ties must be resolved carefully. Ties (which are the norm rather than the exception) are broken in the following manner:

(a) The pairing whose block outputs appear the least frequently among the tied potential pairs,

(b) if there are still ties and if one particular block output, say x, appears in every one of the remaining tied pairs, then we select for x the partner whose next best alternative is the worst in terms of the cost function.

4. Commit Pairings

After selecting the pairing we modify the network and remove any invalid blocks (and their pairings) by

(a) making the block output(s) of the pairing into primary inputs (removing all their in-edges),

(b) making all nodes with edges into the pairing, into primary outputs,

(c) removing all nodes in the blocks which now have no influence on the remaining primary outputs (have no directed path to any primary output), and

(d) eliminating any blocks containing any removed nodes and/or the block outputs of the pairing.

```
begin_Rmap(η)
    η ← Node_Split(η)
    Create_All_Blocks(η)
    Create_All_Potential_pairs(η)
    while there are uncovered nodes in η
        p ← Select_pairing(η)
        Add p to mapping
        η ← Commit_pairing(η, p)
    endwhile
end_Rmap
```

The four main steps in Rmap's covering and packing algorithm are described below. Node_Split() is discussed after the covering and packing are described.

1. Create all blocks.

   At each node x we find the dependency sets for all possible feasible blocks for which x is the block output. The unique block corresponding to each set is also recorded. These sets (and their blocks) are obtained for x from the dependency sets of x's children in a bottom-up traversal of the network. Blocks of x's children are combined to form blocks for x and any of these new blocks which contain a node in its own dependency set are discarded. This is the case for node u in Figure 2 when combining the blocks for its two children with dependency sets $\{x, c, d\}$ and $\{y, v\}$, respectively. The resulting block would depend on y and contain y. We discard this block rather than repair it, because the repaired block will always be generated from another pair of blocks. In this case, combining blocks for the dependency sets $\{x, c, d\}$ and $\{x, c, d, v\}$ yields the block for dependency set $\{x, c, d, v\}$ with nodes y, z, and w.

2. Create All Potential Pairings

   From their lists of dependency sets, it is possible to determine whether two nodes x_1 and x_2 have blocks b_1 and b_2 which can be paired. A *potential pairing* is created for each pair of blocks, b_1 and b_2 of x_1 and x_2 if,

   (a) b_1 and b_2 can be packed together in a single cell, and

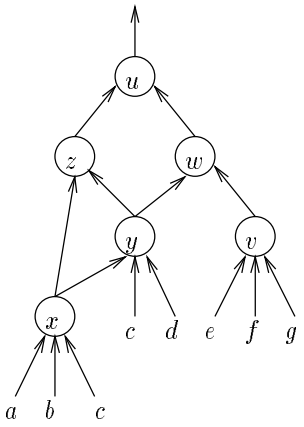
   (b) $x_1 \notin b_2$ and $x_2 \notin b_1$.

   Each block can also be paired with the empty block which provides the mechanism for packing a single block into a cell. The number of edges covered by a pairing is defined to be the sum of the edges covered by the individual blocks; edges common to both blocks are counted twice. (The empty block has zero edges.)

3. Select Pairings

   From the list of all potential pairings, Rmap uses a greedy method to choose a pair of blocks to pack into one cell. A pairing which maximizes the following cost function:

$$edges - c_I \cdot (\#\text{ of input pins}) - c_O \cdot (\#\text{ of output pins}) \tag{1}$$



$x : \{a,b,c\}$
$v : \{e,f,g\}$
$y : \{x,c,d\},\{a,b,c,d\}$
$z : \{x,y\},\{a,b,c,y\},\{a,b,c,d\},\{x,c,d\}$
$w : \{y,v\},\{y,e,f,g\},\{x,c,d,v\},\{v,a,b,c,d\}$
$u : \{z,w\},\{z,y,v\},\{z,y,e,f,g\},\{z,x,c,d,v\},\{x,y,v\},\{x,y,w\},\{x,y,e,f,g\}$
$\{x,c,d,w\},\{x,c,d,v\},\{a,b,c,y,w\},\{a,b,c,y,v\},\{a,b,c,d,w\},\{a,b,c,d,v\}$

Figure 2: A network and the dependency sets of its feasible blocks.

increases the number of alternatives. Similarly, collapsing is also taken care of by considering all blocks with both the parent and child (assuming the collapse would not remove any variable from the fanin). Mostly, however, `Rmap` differs from these approaches in that pairs of blocks and/or single blocks are selected directly from among all potential blocks so that covering and packing are considered simultaneously. Although pairing potential is considered by **Hydra** in its decomposition and covering phases, cell packing is only performed once the covering has been fixed. In constrast to **mispga** and other mappers which attempt to minimize the size of the covering, `Rmap` uses the number of edges covered by a block (edges within and entering the block) to greedily construct a covering which covers as many edges as possible. Finally, because it generates all possible blocks (LUTs) before making any decisions, `Rmap` is able to look ahead one or two steps to determine the effect that selecting a particular pairing will have on the prospects for other nodes' blocks.

4 Mapping for Routability: `Rmap`

The approach to mapping taken in `Rmap` is a greedy one with respect to covering the *edges* in the network as quickly as possible. All feasible blocks are considered and the largest ones which can be paired together are selected. Feasible blocks are generated quickly by keeping track of dependency sets. Recall that the dependency set of a block b in the network is the set of nodes external to b with edges into nodes of b. A node x cannot be the block output of two distinct blocks with the same dependency set. This would require the existence of a node in one block, b_1, not in the other block, b_2, with an edge into some node of b_2. Such a node would by definition be in the dependency set of b_2 and hence also b_1's. Since a node in a block is not in the dependency set of that block, this cannot occur.

We assume that technology independent logic minimization has been performed on the network and that the nodes of the network have been decomposed so that the only Boolean functions represented in the network are ANDs and ORs, possibly with negated inputs: an AND/OR network.

The algorithm used by `Rmap` is given below. It takes as input a logic network, η.

phase is accomplished by `xl_partition` and `xl_cover`. The former uses a greedy method to collapse a child into its parent if the resulting node would form a feasible block. The command `xl_cover` implements both an exact exponential algorithm and heuristics to find a cover with the minimum number of blocks. Candidate blocks for `xl_cover` are determined using a maxflow technique. The cell packing is performed by `xl_merge`[3] which finds a maximum cardinality matching between blocks which can be paired.

Several enhancements to **mispga** are presented in [4]. The decomposition phase now includes a cube-packing algorithm similar to the bin-packing node-splitting in **Chortle-crf** (described above) and co-factoring. Collapsing a child into its parent is now permitted (even if the resulting node is infeasible) as long as the cost is reduced. Finally, a covering technique geared towards minimizing the number of cells rather than blocks is proposed (but not implemented).

Hydra [9] uses techniques similar to those of `mis-pga`, but with stronger emphasis on exploiting two-output CLBs. Simple disjoint decomposition is first applied to network to extract a child (sub-function) with a disjoint input set from its parent. Pairs of nodes with common inputs are identified and decomposed together in an attempt to produce children which can be paired together later. The second decomposition phase is an AND/OR decomposition which groups children whose combined input set is 5 or less in one subtree. The covering (referred to as local elimination in this case) consists of examining nodes from the bottom up and collapsing them into their parent if the result will be a node with 5 or less inputs. If the node has multiple parents, the collapse is performed only when the parents will have 4 or less inputs. Once the blocks are determined, a cost function based on the number of shared inputs and total inputs is used to pair the blocks in a greedy manner.

Xmap [16] converts the Boolean network into an `If-then-else` dag [16]. Simple, efficient heuristics are used to designate some of the nodes in network as block outputs in a manner which guarantees that each designated node has a feasible block whose fanin consists only of other designated nodes. The covering is generated in a second phase by examining each designated node and selecting either the feasible block found for it during the first phase or the block consisting of the "closest designated descendants," which ever block has smaller fanin. A greedy method then pairs up the blocks to form the cells.

Aloe-CLB [8] uses functional decomposition rather than structural decomposition techniques. Specifically, multiplexor based decomposition and Ashenhurst decomposition are used to obtain a Boolean network in which each node forms a feasible block. Blocks are then paired using a greedy method based on the number of shared inputs.

Xnfopt [17] We classify the industrial tool `xnfopt` in this category as well since it performs logic minimization.

All of these tools (with the exception of `hydra` and possibly `xnfmap`) have the common trait of considering cell packing separately from generating the covering. In `Rmap`, we perform covering and cell packing *simultaneously* allowing replication and covering by blocks to be considered along with cell packing. We first perform AND/OR decomposition to reduce the fanin of each node to 3. Because `Rmap` considers all possible blocks in generating its covering, decomposition only

[3]This command relies on a separate integer programming package.

LUT Covering: This phase can be viewed as a graph theoretic problem in which a covering of the network with blocks is constructed. The final set of feasible blocks must satisfy constraints given in Definition 6.

Cell Packing: To increase cell utilization, the cells (CLBs) are designed so that their LUT can be sub-divided into separate LUTs. Cell packing consists of combining the blocks generated by the covering phase into groups which can be accommodated in a single cell. For example, the XC3000 family accommodates one table of 5 inputs or two tables of 4 inputs each (if they have at most 5 distinct inputs between them).

Below we have classified LUT-based mappers according to whether or not they perform logic minimization (restructuring) of the network before generating its covering. We first examine tools which can be characterized as being strictly technology mappers. These tools solve the covering, splitting and packing problems on AND/OR networks. Note that these tools generally assume that some form of technology independent minimization has been performed on the network.

Chortle [6] divides a network into fanout-free trees and uses a dynamic programming approach to obtain the optimal number of blocks covering a tree. This approach produces the minimum number of blocks in fanout-free trees. `Chortle-crf`[7] has additional enhancements to support node-splitting, exploiting reconvergent fanout, and replicating nodes. A bin packing algorithm is used to group a node's children into subtrees (bins) which can be covered by the fewest number of blocks. Reconvergent fanout is exploited by attempting to group children with the same inputs together. For nodes with multiple fanouts, both the unreplicated and replicated cases are considered. `Chortle-d` [13] uses the same dynamic programming approach but with the goal of minimizing the depth of the resulting network; minimizing the number of cells is a secondary optimization.

Vismap [5]. Here the covering problem is formulated as the problem of deciding whether or not each edge will be inside a block (whether the origin of the edge will be a block output). Given a set of choices, one for each edge, it is straightforward to determine whether these choices correspond to a covering, and if so, to determine the number of blocks in the covering. Because of the exponential nature of this potentially exhaustive search, heuristics are used to efficiently generate and control the number of combinations considered. Cell packing is performed only after the blocks have been determined.

Xnfmap [14] is the Xilinx mapper. From our observations, it performs only minor local rearrangements of the network and so we have classified it in this category. (This classification is based solely on our observations since no public documentation is available on its implementation.)

The second class of "mappers" are logic synthesis tools for LUT based FPGAs. They incorporate technology independent as well as technology dependent logic minimization techniques in the covering, splitting and/or packing phases.

Mis-pga [3] enhances the technology independent minimization techniques of `misII` [15] by providing new LUT technology dependent logic optimization commands such as `xl_k_decomp` which performs a limited Roth-Karp decomposition, and `xl_split` which uses kernel extraction and AND/OR decomposition to reduce the number of fanins to each node. The covering

Definition 3 The *dependency set* of a block is the set of nodes outside the block which have edges directed into the block.

Definition 4 The *fanin* of a block is the size of its dependency set.

Definition 5 A *feasible block* is one whose fanin is at or below the maximum lookup-table width (e.g., $k = 5$ in the XC 3000 series).

A block corresponds to a sub-function of the network. Figure 1 shows a network and one of its blocks, the one formed by $\{w, y\}$. This block has dependency set $\{x, c, d, v\}$, fanin 4, and block output w. A block can be implemented with a lookup-table of k inputs if its fanin is at most k. For example in Figure 1, the block $\{w, y, v\}$ has dependency set $\{x, c, d, e, f, g\}$ and is not feasible for $k = 5$, while the larger block $\{u, z, w, y, x\}$ with dependency set $\{a, b, c, d, v\}$ is feasible. Nodes other than the designated block output may have edges in the network to nodes outside the block. These nodes will be replicated in other blocks.

The output of the mapper consists of a covering of the network by blocks and an assignment of the blocks to cells.

Definition 6 A *covering* of the network is a collection of blocks which satisfy the following constraints:

1. each primary output is the block output of some block
2. no primary input is in any block, and
3. each non-primary input node is the block output of some block, or it belongs to every block containing one or more of its parents.

The first constraint defines the boundary of the covering, the second enforces implementation of the outputs of the network and the last constraint ensures that the inputs to each node are implemented either inside the same block, as a block output or as primary inputs. The assignment of blocks to cells is constrained according to the particular resources the cell possesses for accommodating multiple LUTs. We shall assume that the cells can accommodate one or two LUTs as in XC3000 series.

The major activities of logic synthesis for LUT based FPGAs are:

Logic Minimization: The Boolean network can first be altered using technology independent as well as technology dependent logic minimization. Technology independent multi-level logic minimization tools are used to reduce the literal count, while technology dependent minimization techniques take into account the number of inputs of the LUTs in simplifying and rearranging the network.

Node Decomposition/Splitting: If a node is not the block output of any feasible block, then it is not contained in any feasible block. Such nodes must be split into two or more nodes which implement the same logical function as the original node. Node-splitting (or decomposition) can be part of the logic optimization and/or the covering phase (below) where the effects of the various alternatives can be better evaluated. In an AND/OR network, the children of a a node can always be regrouped arbitrarily into subtrees.

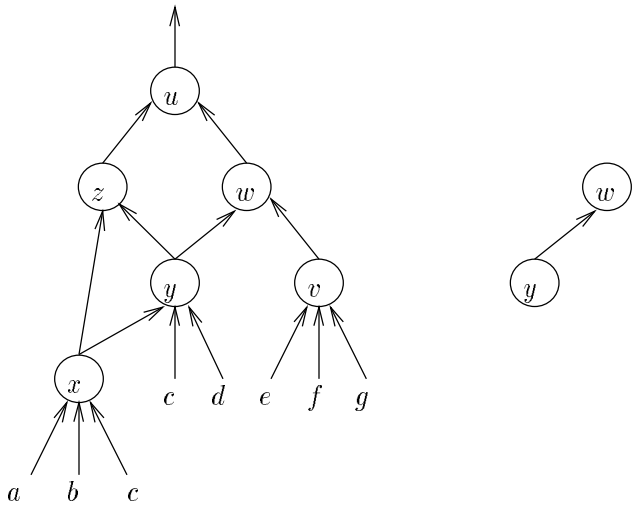


Figure 1: A network and one of its blocks.

a secondary optimization. The goal was to further reduce the number of cells and the byproduct was the reduction of the pins-to-net ratio. In this paper we present a mapper, `Rmap`, which can balance the amount of logic packed into the LUTs and the CLBs against both the pin-to-nets and pins-to-cell ratios. Given different parameters, `Rmap` can balance the various goals given above. This provides the designer with the flexibility of trading number of CLBs for routability.

3 Previous work and definitions

The input of the LUT technology mapping problem is an acyclic directed Boolean network whose sinks are the primary outputs of the circuit and whose roots are the primary inputs. Each node of the graph has a Boolean function associated with it. If there is an edge from node x to node y in the network, x is a child of y while y is a parent of x. [1] Note that since the network is not assumed to be a tree, a node can have more than one parent. An AND/OR network is a Boolean network in which all node functions are either logical ANDs or ORs with complemented or uncomplemented inputs. An arbitrary Boolean network can be converted to an AND/OR network by replacing non-complying nodes with subgraphs implementing an equivalent sum-of-products form for the original node's function.

Definition 1 A *block*[2] of the network is a set of nodes which induces a subgraph of the network having only one sink (a node with no fanout).

Definition 2 The single sink of the sub-network induced by the block is called its *block output*.

[1] We find the terminology parent/child more descriptive than successor/predecessor.

[2] In our terminology a block corresponds to a lookup table, while a cell is a CLB (Configurable Logic Block). A block is equivalent to the term *supernode* introduced in [3,4].

The probability of automatic completion of a particular design.

Using a stochastic two-dimensional gate array model, El Gamal [12] estimates the expected number of tracks required in each channel at about half the product of the average number of pins per cell and the average wire length. The XC3000 series violates this model in not providing full switch matrix connectivity as well as by providing long lines (heterogeneous routing resources). However, the model does support the following two observations:

- Routing larger designs and packages will be more difficult since the average wire length is expected to increase while the routing resources (channel width) remains constant.
- The pins-to-cell ratio will affect the routability since the average number of pins per cell varies while the channel width remains constant.

Based on practical experience and supported by El Gamal's results, a number of related factors which determine the routability of a mapped design can be identified:

Ratio of cells used to total cells available: This ratio is a measure of the density (sparsity) of a design. This ratio depends of the package type used.

Ratio of pins to cells: A high pins-to-cells ratio indicates that a design has a large number of high fanin cells. Both IOBs (Input/Output Blocks) and CLBs are included in the accounting of the number of cells. Similarly, both input and output pins are counted. This ratio is a measure of the amount of traffic in and out around a cell. This ratio is independent of the package type used. It is a characteristic of the mapper.

Ratio of pins to nets: A high pins-to-nets ratio indicates that a design has many high fanout nets. The vendor considers values in excess of 3.6 to be high [11]. This ratio is a measure of the amount of traffic on the chip. This is more of a *global* measure, whereas the pins to cells ratio is a *local* measure. This ratio is independent of the package type used.

These observations lead to the following goals for a mapper:

1. Generate a mapping which does not exceed the number of cells available on the XC3000 LCA.
2. Minimize the fanout of nets by creating LUTs which cover multiple occurrences of the same input.
3. Minimize the fanout of nets by pairing two LUTs which share three inputs since those three nets will each consume only one pin for both LUTs.
4. Minimize and/or distribute the pins evenly among the CLBs.

The first three goals are compatible, however the fourth is in conflict both with the goal of minimizing the number of cells, and with the goal of maximizing LUT pairing (two output pins will be used rather than one). Making the logic units as large as possible is consistent with attempting to minimize the number of cells utilized. However, it conflicts with the reduction in pins-to-nets that occurs because of the packing of multiple LUTs in a single cell. Previous approaches to the mapping problem (with the exception of **Hydra** [9]) have considered the packing of LUTs into cells only as

shall see in Section 5 the manner in which a design is *mapped* can determine whether or not the design can be *automatically routed* and if not, how close to completion the automatic routing results will be. The routing resources in FPGA architectures must be balanced against cell resources. Tools which use the routing resources effectively admit architectures which devote more area to cells.

Typically, only 80% of the cells can be allocated in a mapping that is routable [2]. Minimizing the number of cells in the mapping is important since the number of cells available is a hard constraint. However, mappers which concentrate only on minimzing the number of cells may require an even lower threshold to ensure routability. Minimizing the number of cells without regard to routability is futile if the mapped design cannot be routed. It is quite possible that with a slightly less compact mapping, a larger design can still be accommodated and be routed.

In this paper, we present a technology mapper for LUT-based cell arrays, `Rmap`, which balances cell utilization with the goal of producing routable mappings. In Section 2 we examine the architecture of a LUT-based FPGA and routability criteria. Section 3 we define the mapping problem and review previous approaches. The mapping algorithm used in `Rmap` is presented in Section 4 and we compare `Rmap`'s results with two other mappers' in Section 5. Experiments validating the routability criteria are also provided in Section 5.

2 LUT-based LCAs and Routability Criteria

We focus on the Logic Cell Array (LCA) architecture of the XC3000 series [2], however these techniques can be generalized to similar Look-Up Table(LUT)-based architectures. The architecture of the XC3000 LCA series consists of an array of configurable logic blocks (CLBs) separated by routing channels. Each CLB provides a 32-bit LUT and two D flip-flops. The LUT can either be configured as a single 5-input LUT or as two 4-input LUTs. However since only 5 signals can be routed, the two LUTs must have at most 5 inputs among them in the latter case. Input/Output Blocks (IOBs) for tristate-able outputs and buffered, latched input/output connections to each user pins are along the perimeter of the LCA. The routing resources on the LCA consist of

1. direct lines which connect each cell with its four immediate neighbors in the array,
2. general-purpose interconnect which consist of vertical and horizontal segments intersecting at switch matrices, and
3. long lines which run horizontally and vertically the entire length of the array bypassing the switch matrices.

The switch matrices do not offer full connectivity. Rose and Brown [10] have shown that limited switch matrix flexibility is preferable in balancing area between routing and cell resources. Some of the horizontal long lines are provided for tristate busses, and can also be used for high-fanout nets or even local routing.

The XC3000 series consists of various packages with arrays ranging in size from 64 cells to 320 cells with 34 to 144 user programmable pads. The direct line and general interconnect resources do not vary over the range of packages, however the larger LCAs have additional long lines with the added possibility of splitting some long lines in half.

Once a design has been mapped, it must be placed and routed. The vendor defines routability [11, pg. 34] of a mapped design as:

Routability-Driven Technology Mapping for LookUp Table-Based FPGAs

Martine Schlag, Jackson Kong and Pak K. Chan

February 7, 1992

Abstract

A new algorithm for technology mapping of LookUp Table-based Field-Programmable Gate Arrays (FPGAs) is presented. It has the capability of producing slightly more *compact* designs (using less cells (CLBs)) than some existing mappers. More significantly, it has the flexibility of trading routability with compactness of a design. Research in this area has focussed on minimizing the number of cells. However, minimizing the number of cells without regard to routability is ineffective. Since placement and routing is really the most time-consuming part of the FPGA design process, producing a routable design with a slightly larger number of cells is preferable than producing a design using fewer cells which is difficult to route, or in the worst case unroutable. We have implemented our algorithm in the `Rmap` program, and studied routability of two other mappers with respect to `Rmap` in this paper. In general `Rmap` produces mappings with better routability charactersitics, and more significantly `Rmap` produces routable mappings when other mappers do not.

1 Introduction

Field-Programmable Gate Arrays (FPGAs) are integrated circuits consisting of arrays of gates that can be configured – and reconfigured – by the system designer through software, rather than by chip manufacturer in the fabrication line. With realization times measured in hours, systems incorporating up to thousands of gates on a single FPGA can be designed, programmed and evaluated within a few weeks. The advent of FPGA technology facilitates a mechanism for rapid prototyping. There are multiplexor-based [1] and LookUp Table-based (or RAM-based) FPGAs. This paper focusses on LookUp Table-based (LUT) FGPAs implemented as Logic Cell Arrays (LCAs) [2].

The success of FPGA technologies is supported by a set of CAD tools to aid the design process. The two final steps in the implementation of a prototype on an FPGA are:

1. its mapping to a network of logic cells (technology mapping), and
2. the assignment of the network cells to physical cells on the LCA and the configuration of routing structures to interconnect them as in the network (placement and routing).

The figure of merit for logic synthesis and mapping techniques for LUTs in the literature has been the number of logic cells in the network (Configurable Logic Blocks) [3,4,5,6,7,8,9]. However, as we