LUT Structure for Delay: Cluster or Cascade?

Alan Mishchenko

Department of EECS, University of California, Berkeley

alanmi@eecs.berkeley.edu

Abstract

This paper compares two types of lookup table (LUT) structures useful for delay optimization in programmable devices. Realistic assumptions about the input-to-output delays in a LUT and the delays of routable and non-routable connections are used. The conclusion is that the cluster LUT structure results in better delays compared to the cascade structure. This LUT structure leads to mapped networks with a smaller ratio of near-critical outputs. Finally, it is easier to handle in the technology mapper.

1. Introduction

The drive to increase clock frequency of programmable devices motivates the following orthogonal improvements in technology: (1) designing new FPGA architectures that allow for faster implementation of digital circuits, and (2) improving methods for logic synthesis and technology mapping into the existing FPGA architectures.

It is known that the delay of the FPGA devices is dominated by the programmable interconnect. A typical ratio between the intrinsic delay of a LUT and a wire delay is 1:5. This motivates programmable fabrics that allow some connections to avoid the general-case routing.

Along this line of research, recently it was proposed to use *LUT structures* [11] to speed up programmable devices by introducing non-routable connections between adjacent LUTs. Because the corresponding wires are not routed through multiple switch-boxes and routing channels, the delay of such connection is much faster than an average delay of a wire in the programmable device.

Previous work [11] proposed several efficient algorithms for mapping into LUT structures, but experimental evaluation is limited to only two LUT structures: 44 and 444, composed of two and three 4-LUTs, respectively.

This work considers two generic type of LUT structures: *cluster* and *cascade*, shown in Figures 1 and 2, respectively. The structure composed of only two LUTs can be seen as both a cluster and a cascade, while larger structures can belong to only one of these categories.

The question considered in this paper is, assuming that a programmable architecture supports only one (and not both) structures, what structure leads to a better delay? Additional questions are: What structure has a better near-critical path ratio? What structure is easier to handle in a technology mapper fashioned along the lines of [9]?

The contribution of this paper is to show that the cluster is a preferable LUT structure, compared to the cascade.

The paper is organized as follows. Section 2 describes the background. Section 3 outlines the algorithm used to evaluate the LUT structures. Section 4 reports experimental results. Section 5 concludes and outlines future work.

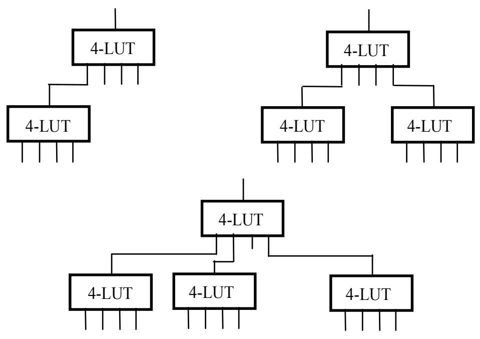

Figure 1: The **cluster** 4-LUT structures:
44 (top left), 444 (top right), 4444 (bottom center).

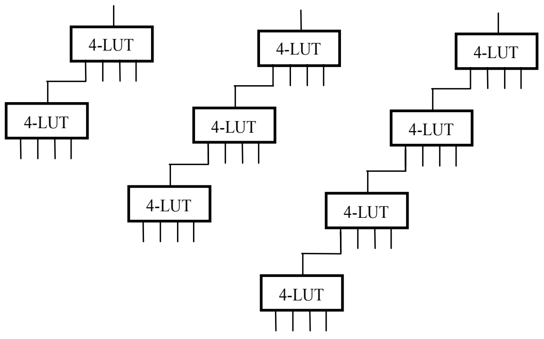

Figure 2: The **cascade** 4-LUT structures:
44 (left), 444 (center), 4444 (right).

2. Background

2.1 Boolean network

A *Boolean network* (or *netlist*, or *circuit*) is a directed acyclic graph (DAG) with nodes corresponding to logic gates and edges corresponding to wires connecting the gates. In this paper, we consider only combinational Boolean networks. Sequential networks are handled as combinational by cutting at the register boundary.

A combinational *And-Inverter Graph* (AIG) is a Boolean network composed of two-input ANDs and inverters. The *size* (*area*) of an AIG is the number of its nodes; the *depth* (*delay*) is the number of nodes on the longest path from the primary inputs (PIs) to the primary outputs (POs).

A *cut C* of a node *n* is a set of nodes, called *leaves,* such that each path from a PI to *n* passes through at least one leaf. Node *n* is called the *root* of cut *C*.

Additional information can be found in the following publications: delay-optimal technology mapping [4], cut-based mapping [5], AIGs [2], AIG-based synthesis [7][8], delay optimization and area recovery [9].

3. Algorithm

The current implementation of delay-optimal mapping into LUT structures assumes that the network is already mapped into LUTs. The algorithm follows the dynamic programming approach established by the traditional LUT mappers [4][5] and their recent variations [9][11].

Figure 3 shows the pseudo-code. The top level procedure mapDelayOriented() takes a mapped network and the set of parameters describing LUT structures used in the mapping in terms of delay, area, support sizes, etc. The procedure findCutMatchingLutStructureWithMinDelay() find the best cut and a matching LUT structure at a node. These cut and structures lead to the smallest delay among other cuts and the corresponding LUT structures at this node.

Finally, procedure getDelay() takes the cut and returns the delay of the LUT structure matched with this cut. When evaluating the delay, the delays of the previously mapped cut leafs are added to the input-to-output delays of the LUT structure. The maximum delay among these is computed and returned as the delay of the cut.

```
mapDelayOriented( network, parameters ) {
    for each network node n in topological order {
        cut = findCutMatchingLutStructureWithMinDelay( n );
        setDelay( n, getDelay(cut) );    setRepresentativeCut( n, cut );
    }
}
findCutMatchingLutStructureWithMinDelay( node, parameters ) {
    cut_best = NULL;
    for each cut c of node matching LUT structure s
        if ( cut_best == NULL or getDelay(cut_best) > getDelay(c) )
            cut_best = c;
    return cut_best;
}
getDelay( cut ) {
    delay_max = -∞;
    for each node m in cut
        delay_max = max(delay_max, getDelay(m) + getStructDelay(m));
    return delay_max;
}
```

Figure 3. Delay-oriented mapping into LUT structures.

It should be noted that this procedure assumes that the network is mapped into LUTs before it is mapped into LUT structures. A better way would be to map an AIG directly into LUT structures while bypassing mapping into LUTs. The use of structural choices [6][3] could greatly improve the quality of mapping in this case.

4. Experimental results

The proposed algorithm is implemented in ABC [1][2] as command *ifif*, applicable to the network of K-LUTs. It performs delay-oriented structural mapping (or packing) into LUT structures. The structure type and their delay parameters are configured on the command line.

The difference between *ifif*, the priority-cut-based technology mapper *if* [9] and its extension, *if –S*, to map into LUT structures [11], is that the former is applicable to a LUT mapped network while the latter two are applicable to an AIG. The result of applying both *ifif* and *if –S* is a network of LUT structures exhibiting delay improvements compared to the traditional LUT mapping.

The reason a new mapper had to be developed is because the LUT mapper *if* cannot map into LUT structures and its extension *if –S* can only map into a limited subset of LUT structures, which was not sufficient to perform the evaluation attempted in this paper.

Experiments were performed using a suite of public benchmarks used in the previous work [11].

4.1 Delay assumptions

The input-to-output delays of a LUT are assumed to be as follows: Di = d*(i+1), where i is input pin number and d is a constant equal to 0.1 (for 4-LUTs) or 0.05 (for 6-LUTs).

The wire delay is assumed to be constant and equal to 0.5 (for 4-LUTs) or 0.7 for (6-LUTs). The non-routable wire delay is assumed to be 0, that is, negligible compared to the routable wire delay.

These assumptions led to the following LUT libraries represented in ABC LUT library format:

```
# The area/delay of 4- LUTs:
# k   area    delay
1     1.00    0.60
2     1.00    0.60 0.70
3     1.00    0.60 0.70 0.80
4     1.00    0.60 0.70 0.80 0.90

# The area/delay of 6-LUTs:
# k   area    delay
1     1.00    0.75
2     1.00    0.75 0.80
3     1.00    0.75 0.80 0.85
4     1.00    0.75 0.80 0.85 0.90
5     1.00    0.75 0.80 0.85 0.90 0.95
6     1.00    0.75 0.80 0.85 0.90 0.95 1.00
```

In these libraries, the wire delay is added to the input-to-output delays, resulting in the time a signal needs to travel from the output of one LUT to that of another.

This delay model is approximate but it was successfully used for technology-independent synthesis in an industrial FPGA design flow [10].

4.2 Delay-oriented mapping into 4-LUTs

The results of delay-oriented mapping into different 4-LUT structures of the cluster/cascade type are reported in Table 1 and visualized in the diagram in Figure 4.

The conclusion is that the cluster dominates the cascade in all but two cases when they lead to identical structures (both 4-LUT mapping and 44 structure are the same in the case of cluster and cascade).

4.3 Near-critical path ratio for 4-LUTs

The near-critical paths are defined as those paths from the PIs to the POs whose delay is within 10% of the most critical. To compute the near-critical path ratio (NCPR), the number of near-critical paths in a design is divided by the total number of PI/PO paths and multiplied by 100%.

The NCPR statistics are for different 4-LUT structures of the cluster/cascade type are reported in Table 2 and visualized in Figure 5.

The conclusion is that the cluster is better for the 444 structure and the cascade is better for larger structures (4444 and 44444).

It would be interesting to explain the characteristic dip of the curve for the 444 structure, which can also be seen in the curve for LUT structures based on 6-LUTs.

4.4 Experimental results for 6-LUTs

Experiments similar to those described in Sections 4.2-4.3 were performed for 6-LUTs. The diagrams comparing the delay after LUT mapping and the NCPRs are shown in Figures 6 and 7, respectively.

4.5 Experiments on industrial benchmarks

Experiments similar to those described in Sections 4.2-4.4 were performed for several suites of industrial FPGA benchmarks. The results were close to those listed in Tables 1 and 2 for the academic benchmarks.

5. Conclusions

This paper compares two types of LUT structures, the cluster and the cascade, and their impact on the delay of circuits implemented in programmable devices.

The conclusion is that the cluster is more suitable for delay optimization, compared to the cascade.

Future work includes developing a dedicated technology mapper that starts with an AIG and produces mapping into a given family of LUT structures.

6. Acknowledgements

This work was partly supported by SRC contract 1875.001 and NSA grant, ”Enhanced equivalence checking in crypto-analytic applications”. The author is grateful to the industrial sponsors of BVSRC: Altera, Atrenta, Cadence, Calypto, IBM, Intel, Jasper, Microsemi, Real Intent, Synopsys, Tabula, and Verific, for their continued support. The author thanks industrial collaborators for helpful discussions related to this project.

7. REFERENCES

[1] Berkeley Logic Synthesis and Verification Group. *ABC: A System for Sequential Synthesis and Verification*. http://www-cad.eecs.berkeley.edu/~alanmi/abc

[2] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool", *Proc. CAV'10*, LNCS 6174, pp. 24-40.

[3] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, and T. Kam, “Reducing structural bias in technology mapping”, *ICCAD '05*.

[4] J. Cong and Y. Ding, “FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs”, *IEEE Trans. CAD*, vol. 13(1), Jan. 1994, pp. 1-12.

[5] R. J. Francis, J. Rose, and K. Chung, ”Chortle: A technology mapping program for lookup table-based field programmable gate arrays”, *Proc. DAC ’90*, pp. 613-619.

[6] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness, “Logic decomposition during technology mapping,” *IEEE Trans. CAD*, Vol. 16(8), Aug. 1997, pp. 813-833.

[7] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG rewriting: A fresh look at combinational logic synthesis", *Proc. DAC '06*, pp. 532-536.

[8] A. Mishchenko and R. K. Brayton, "Scalable logic synthesis using a simple circuit structure", *Proc. IWLS '06*, pp. 15-22.

[9] A. Mishchenko, S. Cho, S. Chatterjee, R. Brayton, “Combinational and sequential mapping with priority cuts”, *Proc. ICCAD ’07*, pp. 354-361.

[10] A. Mishchenko, N. Een, R. K. Brayton, S. Jang, M. Ciesielski, and T. Daniel, "Magic: An industrial-strength logic optimization, technology mapping, and formal verification tool". *Proc. IWLS'10*, pp. 124-127.

[11] S. Ray, A. Mishchenko, N. Een, R. Brayton, S. Jang, and C. Chen, "Mapping into LUT structures", *Proc. DATE'12*.

Table 1: Delay comparison for cluster vs. cascade 4-LUT structures.

Example	Cluster					Cascade				
	4-LUT	44	444	4444	44444	4-LUT	44	444	4444	44444
alu4	5.20	4.90	4.00	3.60	3.20	5.20	4.90	4.90	4.90	4.90
apex2	5.70	5.10	4.20	4.20	3.80	5.70	5.10	4.60	4.60	4.60
b14	12.90	9.70	9.20	8.70	7.90	12.90	9.70	9.20	9.20	8.70
b15	15.70	12.80	11.50	10.70	10.20	15.70	12.80	11.90	11.70	11.30
b17	21.50	17.10	15.40	14.30	13.40	21.50	17.10	15.70	15.10	14.60
b20	14.50	10.60	10.00	9.60	9.30	14.50	10.60	9.60	9.00	8.90
b21	14.70	11.30	10.80	9.70	9.40	14.70	11.30	10.20	9.70	9.60
b22	15.30	11.20	10.80	10.30	9.40	15.30	11.20	10.20	9.70	9.30
clma	8.70	7.60	6.60	6.20	5.80	8.70	7.60	7.40	7.40	7.40
des	4.50	4.00	3.50	3.40	3.00	4.50	4.00	4.00	4.00	4.00
elliptic	6.20	5.60	5.20	4.60	4.10	6.20	5.60	5.40	5.40	5.40
ex5p	4.50	4.00	3.50	3.40	3.00	4.50	4.00	4.00	4.00	4.00
frisc	12.80	9.60	8.70	8.10	7.80	12.80	9.60	8.70	8.20	8.20
i10	10.00	8.40	7.40	6.80	6.50	10.00	8.40	7.80	7.80	7.80
pdc	6.20	5.90	5.20	4.70	4.10	6.20	5.90	5.90	5.90	5.90
s38584	6.10	5.40	4.90	4.50	4.20	6.10	5.40	5.40	5.40	5.40
s5378	3.70	3.20	3.10	2.80	2.20	3.70	3.20	3.20	3.20	3.20
seq	4.60	4.20	4.00	3.40	3.10	4.60	4.20	4.20	4.20	4.20
spla	6.20	6.00	5.20	4.60	4.20	6.20	6.00	6.00	6.00	6.00
tseng	8.70	6.20	5.70	5.70	5.20	8.70	6.20	5.70	5.60	5.20
Geo	1.000	0.840	0.758	0.704	0.643	1.000	0.840	0.802	0.790	0.780

Table 2: Near-critical path ratio (NCPR) comparison for cluster vs. cascade 4-LUT structures.

Example	Cluster					Cascade				
	4-LUT	44	444	4444	44444	4-LUT	44	444	4444	44444
alu4	37.50	25.00	25.00	25.00	25.00	37.50	25.00	25.00	25.00	25.00
apex2	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
b14	22.41	10.37	4.01	2.34	3.34	22.41	10.37	3.68	4.01	21.40
b15	22.54	18.11	20.23	16.96	16.96	22.54	18.11	22.54	19.85	20.04
b17	7.74	1.92	2.31	2.31	2.51	7.74	1.92	3.24	3.70	7.41
b20	12.30	9.18	7.81	5.27	5.47	12.30	9.18	7.23	9.18	8.20
b21	14.26	4.30	1.17	1.17	0.39	14.26	4.30	1.76	8.01	9.77
b22	12.81	6.87	3.96	3.30	3.96	12.81	6.87	5.81	6.47	11.89
clma	4.35	3.48	3.48	3.48	3.48	4.35	3.48	3.48	4.35	3.48
des	46.53	26.12	23.67	23.67	23.67	46.53	26.12	26.12	15.92	27.35
elliptic	10.20	8.16	8.16	8.16	8.16	10.20	8.16	10.20	10.20	10.20
ex5p	49.21	49.21	49.21	49.21	49.21	49.21	49.21	49.21	11.11	49.21
frisc	10.78	11.98	8.38	10.78	8.38	10.78	11.98	11.98	11.98	10.78
i10	3.13	2.68	2.68	2.68	2.68	3.13	2.68	1.79	2.23	1.79
pdc	42.50	35.00	35.00	35.00	35.00	42.50	35.00	35.00	35.00	40.00
s38584	0.64	0.23	0.23	0.23	0.23	0.64	0.23	0.23	0.52	0.12
s5378	27.62	14.76	5.71	5.71	5.71	27.62	14.76	0.95	9.52	22.38
seq	40.00	28.57	22.86	22.86	22.86	40.00	28.57	5.71	11.43	37.14
spla	23.91	19.57	19.57	19.57	19.57	23.91	19.57	19.57	21.74	23.91
tseng	5.72	4.54	3.75	2.56	2.76	5.72	4.54	4.93	3.75	7.30
Geo	1.000	0.651	0.520	0.484	0.470	1.000	0.651	0.481	0.589	0.796

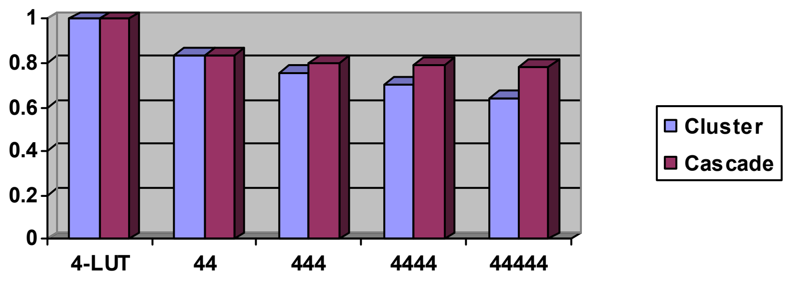

Figure 4: Delay comparison for cluster vs. cascade 4-LUT structures (Table 1).

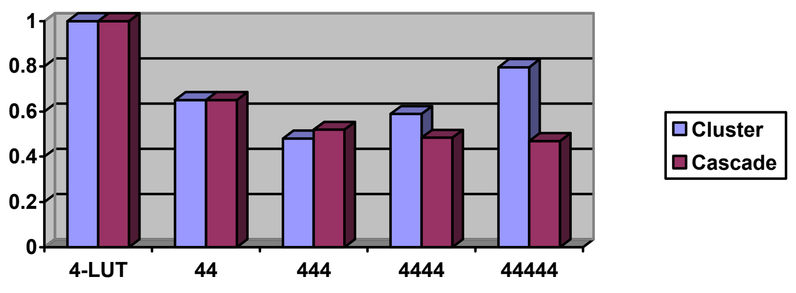

Figure 5: Near-critical path ratio (NCPR) comparison for cluster vs. cascade 4-LUT structures (Table 2).

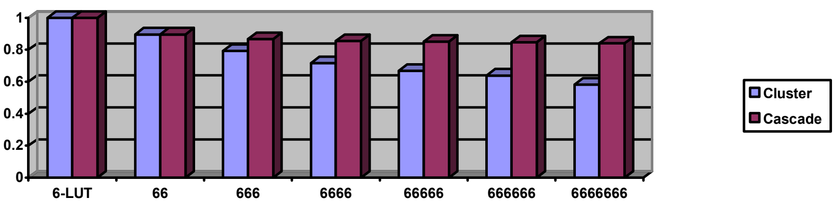

Figure 6: Delay comparison for cluster vs. cascade 6-LUT structures.

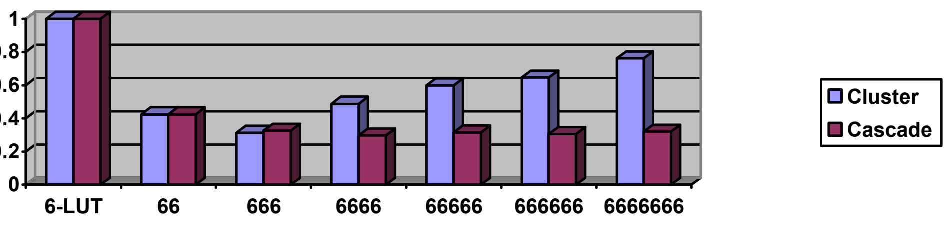

Figure 7: Near-critical path ratio (NCPR) comparison for cluster vs. cascade 6-LUT structures.