Technology Mapping with Choices, Priority Cuts, and Placement-Aware Heuristics

Alan Mishchenko

UC Berkeley

Overview

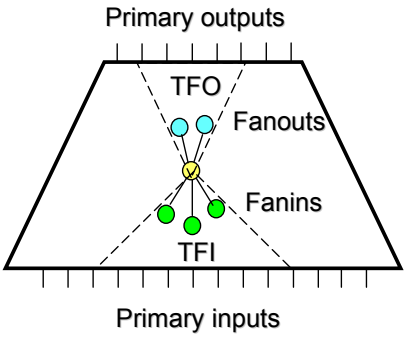
(1) Introduction
(2) Technology mapping
(3) Priority cuts
(4) Structural choices
(5) Tuning mapping for placement
(6) Other applications

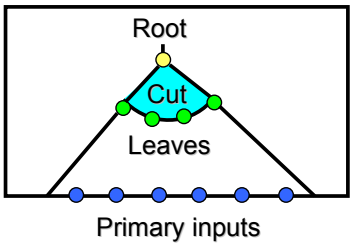
(1) Introduction

- Terminology
- And-Inverter Graphs
- Technology mapping in a nutshell

Terminology

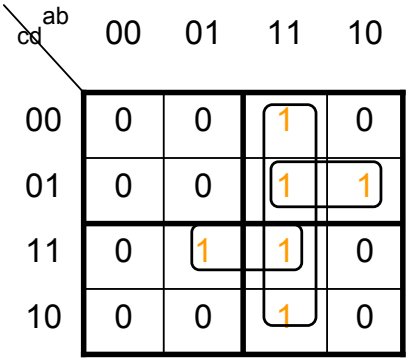
- Logic network
  - Primary inputs/outputs (PIs/POs)
  - Logic nodes
  - Fanins/fanouts
  - Transitive fanin/fanout cone (TFI/TFO)
- Structural cut of a node
  - Cut is a boundary in the network separating the node from the PIs
  - Boundary nodes are the leaves
  - The node is the root
  - K-feasible cut has K or less leaves
  - Function of the cut is function of the root in terms of the leaves



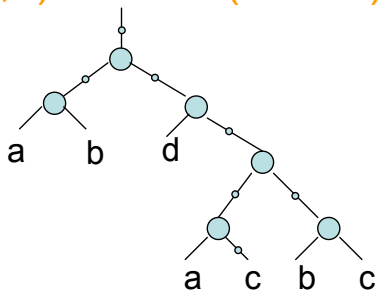


AIG Definition and Examples

AIG is a Boolean network composed of two-input ANDs and inverters.

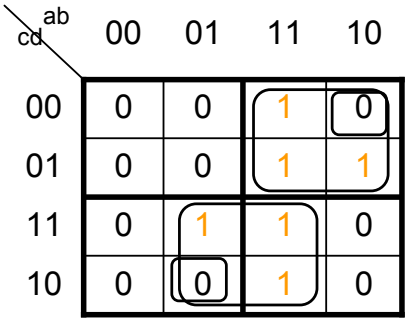


F(a,b,c,d) = ab + d(ac'+bc)

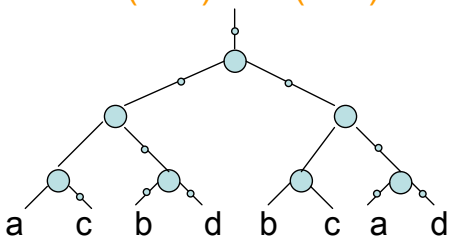


6 nodes

4 levels



F(a,b,c,d) = ac'(b'd')' + c(a'd')' = ac'(b+d) + bc(a+d)

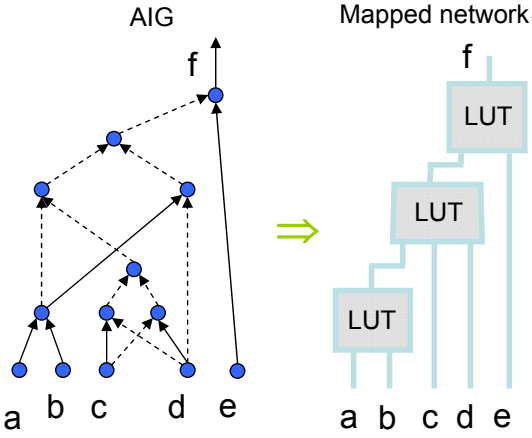


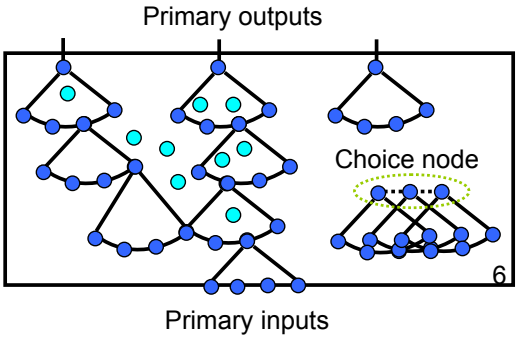
7 nodes

3 levels

Mapping in a Nutshell

- AIGs reprsent logic functions
  - A good subject graph for mapping
- Technology mapping expresses logic functions to be implemented
  - Uses a description of a technology
- Technology
  - Primitives with delay, area, etc
- Structural mapping
  - Computes a cover of AIG using primitives of the technology
- Cut-based structural mapping
  - Computes cuts for each AIG node
  - Associates each cut with a primitive
  - Selects a cover with a minimum cost
- Structural bias
  - Good mapping cannot be found because of the poor AIG structure
- Overcoming structural bias
  - Need to map over a number of AIG structures (leads to choice nodes)





(2) Technology Mapping

- Traditional LUT mapping
  - Delay-optimal mapping
  - Area recovery
- Drawbacks of the traditional mapping
  - Excessive memory and runtime
  - Structural bias
- Ways to mitigate the drawbacks
  - Priority cuts
  - Structural choices

Traditional LUT Mapping Algorithm

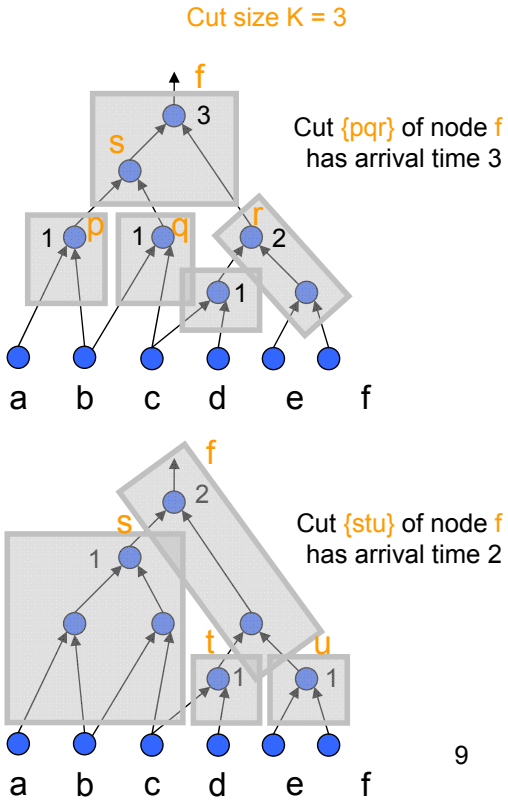
Input: And-Inverter Graph

1. Compute *K*-feasible cuts for each node
2. Compute best arrival time at each node
   - In topological order (from PI to PO)
   - Compute the depth of all cuts and choose the best one
3. Perform area recovery
   - Using area flow
   - Using exact local area
4. Chose the best cover
   - In reverse topological order (from PO to PI)

Output: Mapped Netlist

Delay-Optimal Mapping

- Input:
  - AIG and K-cuts computed for all nodes
- Algorithm:
  - For all nodes in a topological order
    - Compute arrival time of each cut using fanin arrival times
    - Select one cut with min arrival time
    - Set the arrival time of the node to be the arrival time of this cut
- Output:
  - Delay-optimal mapping for all nodes

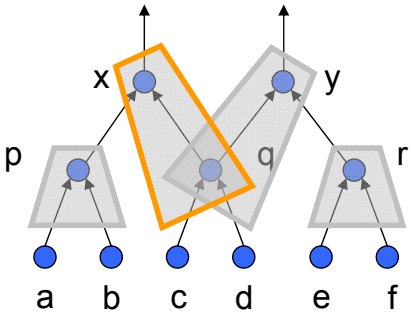


Area Recovery During Mapping

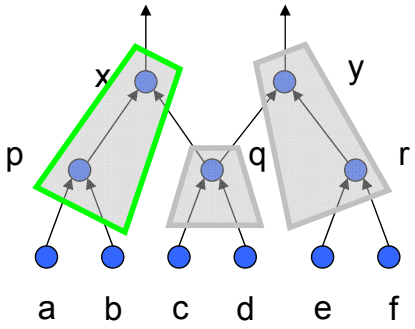
- Delay-optimal mapping is performed first
  - Best match is assigned at each node
  - Some nodes are used in the mapping; others are not used
- Arrival and required times are computed for all AIG nodes
  - Required time for all used nodes is determined
  - If a node is not used, its required time is set to +infinity
- Slack is a difference between required time and arrival time
- If a node has positive slack, its current best match can be updated to reduce the total area of mapping
  - This process is called area recovery
- Exact area recovery is exponential in the circuit size
  - A number of area recovery heuristics can be used
- Heuristic area recovery is iterative
  - Typically involved 3-5 iterations
- Next, we discuss cost functions used during area recovery
  - They are used to decide what is the best match at each node

How to Measure Area?

Suppose we use the naïve definition:
Area (cut) = 1 + [Σ area (fanin)]
(assuming that each LUT has one unit of area)



Area of **cut {pcd}**
= 1 + [1 + 0 + 0]
= 2



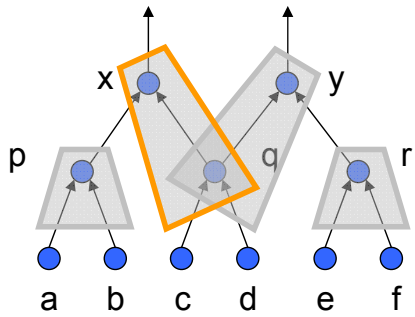
Area of **cut {abq}**
= 1 + [0 + 0 + 1]
= 2

Naïve definition says both cuts are equally good in area

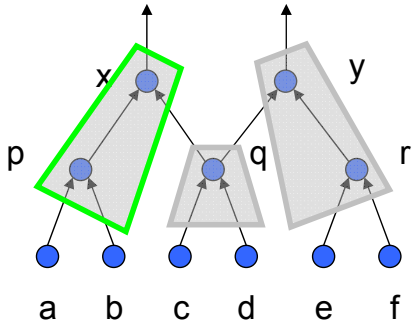
Naïve definition ignores sharing due to multiple fanouts

Area-flow

area-flow (cut) = 1 + [Σ (area-flow (fanin) / fanout_num(fanin))]



Area-flow of **cut {pcd}**
= 1 + [1 + 0 + 0]
= 2



Area-flow of **cut {abq}**
= 1 + [0/1 + 0/1 + ½]
= 1.5

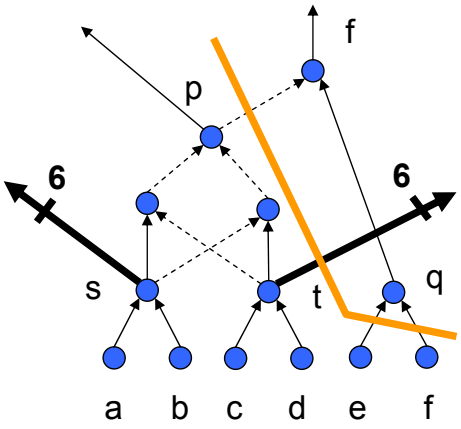
Area-flow recognizes that **cut {abq}** is better

Area-flow "correctly" accounts for sharing

(Cong '99, Manohara-rajah '04)

Exact Local Area

Exact-local-area (cut) = 1 + [Σ exact-local-area (fanin with no other fanout)]

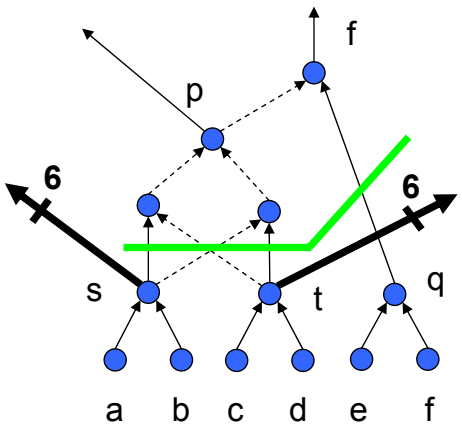


Cut {pef}

Area flow = 1+ [(.25+.25+3)/2] = 2.75

Exact area = 1 + 0 (p is used elsewhere)

Exact area will choose this cut.



Cut {stq}

Area flow = 1+ [.25+.25 +1] = 2.5

Exact area = 1 + 1 = 2 (due to q)

Area flow will choose this cut.

Area Recovery Summary

- Area recovery heuristics
  - Area-flow (global view)
    - Chooses cuts with better logic sharing
  - Exact local area (local view)
    - Minimizes the number of LUTs by looking one node at a time
- The results of area recovery depends on
  - The order of processing nodes
  - The order of applying two passes
  - The number of iterations
  - Implementation details
- This scheme works for the constant-delay model
  - Any change off the critical path does not affect critical path

Drawbacks of Traditional Mapping

- Excessive memory and runtime requirements
  - Exhaustive cut enumeration leads to many cuts (especially when K ≥ 6)
- Structural bias
  - The structure of the object graph does not allow good mapping to be found

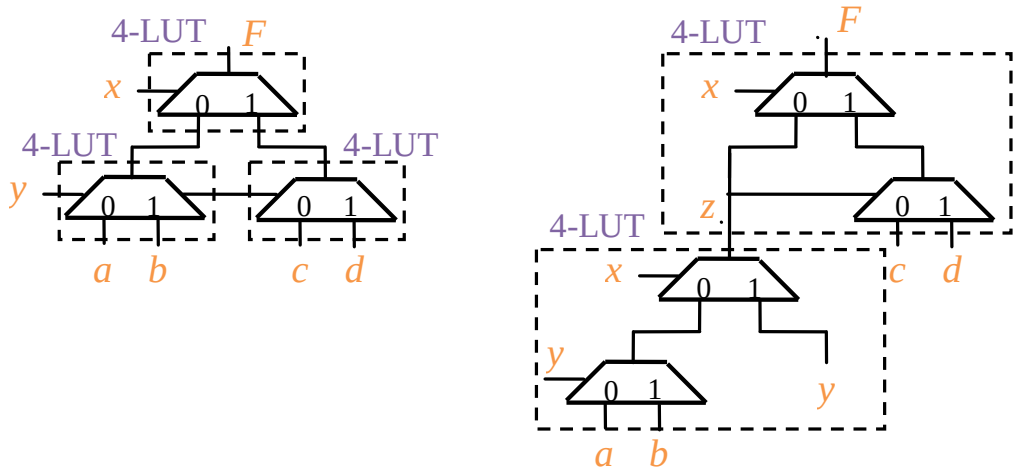
Excessive Memory and Runtime

- For large designs, there may be too many *K*-feasible cuts
  - 1M node AIG has ~50M 6-cuts
  - Requires ~2GB of storage memory and takes ~30 sec to compute
- Past ways of tackling the problem
  - Detect and remove dominated cuts
    - Does not help much
  - Perform cut pruning (store N cuts/node)
    - Throws away useful cuts even if N = 1000
  - Store only cuts on the frontier
    - Reduces memory but increases runtime

k	Average number of cuts per node
4	6
5	25
6	50
7	120
8	250

Structural Bias

- Consider mapping 4:1 MUX into 4-LUTs
  - The naïve approach results in 3 LUTs
  - After logic structuring, mapping with 2 LUTs can be found



Ways to Mitigate the Drawbacks

- Excessive memory and runtime requirements
  - Compute only a small number of “useful” cuts
    - Leads to mapping with priority cuts
- Structural bias
  - Perform mapping over multiple circuit structures
    - Leads to mapping with structural choices

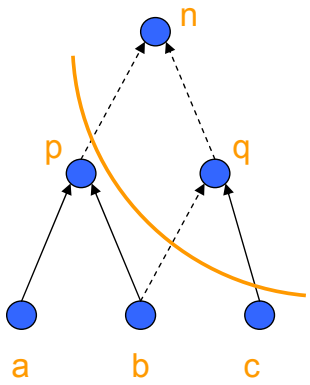
(3) Priority Cuts

- Structural cuts
- Exhaustive cut enumeration
- Prioritizing cuts
- Implementation tricks

Structural Cuts in AIG

A cut of a node *n* is a set of nodes in transitive fanin such that every path from the node to PIs is blocked by nodes in the cut.

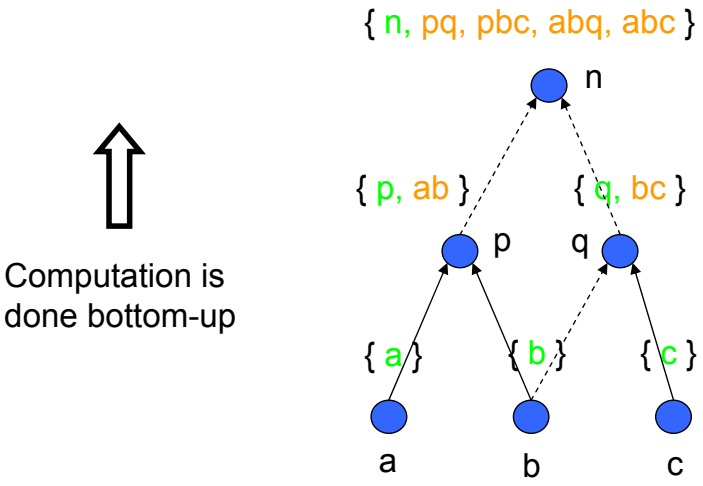
A k-feasible cut has no more than k leaves.



The set {pbc} is a 3-feasible cut of node n. (It is also a 4-feasible cut.)

k-feasible cuts are important in LUT mapping because the logic between root n and the cut leaves {pbc} can be replaced by a *k*-LUT.

Exhaustive Cut Enumeration

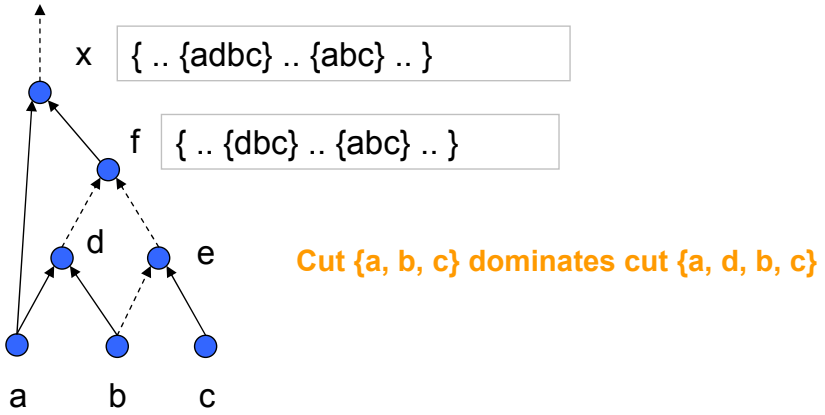


The set of cuts of a node is a 'cross product' of the sets of cuts of its children.
Any cut that is of size greater than *k* is discarded.

(P. Pan et al, FPGA '98; J. Cong et al, FPGA '99)

Cut Filtering

Bottom-up cut computation in the presence of re-convergence might produce *dominated* cuts



- The "good" cut {abc} is present (so not a quality issue)
- But the "bad" cut {adbc} may be propagated further (so a run-time issue)
- It is important to discard dominated cuts ***quickly***

Signature-Based Cut Filtering

Problem: Given two cuts, how to quickly determine whether one can be a subset of another.

Solution: *Signature* of a cut is a 32-bit integer defined as:

$$sig(c) = \sum_{n \in c} 2^{ID(n) \bmod 32}$$

(Σ means bit-wise OR)

where ID(n) is the integer id of node *n*

Observation: If cut c_1 dominates cut c_2, then

$sig(c_1)$ OR $sig(c_2) = sig(c_2)$

Signature checking is a quick test for the most common case when a cut does not dominate another. Only if this check fails, an actual comparison is performed.

Example

- Let the node IDs be a = 1, b = 2, c = 3, d= 4
- Let c_1 = {a, b, c} and c_2 = {a, d, b, c}
- $sig(c_1) = 2^1$ OR 2^2 OR 2^3
  = 0001 OR 0010 OR 0100
  = 0111
- $sig(c_2) = 2^1$ OR 2^4 OR 2^2 OR 2^3
  = 0001 OR 1000 OR 0010 OR 0100
  = 1111
- As $sig(c_1)$ OR $sig(c_2) \neq sig(c_1)$, c_2 does **not** dominate c_1
- But $sig(c_1)$ OR $sig(c_2) = sig(c_2)$, so c_1 **may** dominate c_2

Experiment with K-Cut Computation

		K = 4		*K* = 5		*K* = 6		*K* = 7		*K* = 8		
Name	AIG	C/N	T, s	C/N	T, s	C/N	T, s	C/N	T, s	C/N	T, s	L/N, %
alu4	2642	6.7	0.00	12.3	0.01	23.1	0.04	45.5	0.18	94.7	1.02	0.00
apex2	2940	7.2	0.01	14.2	0.02	29.2	0.07	62.6	0.32	139.7	1.90	0.00
apex4	2017	8.5	0.00	19.5	0.03	47.0	0.10	116.3	0.62	293.5	4.49	0.10
bigkey	3080	6.6	0.01	12.1	0.02	24.2	0.05	50.1	0.20	99.7	0.84	0.00
clma	11869	8.1	0.04	18.2	0.11	44.4	0.51	114.9	3.01	306.3	20.99	1.64
des	3020	8.0	0.01	17.0	0.03	38.7	0.12	92.0	0.69	218.0	4.80	4.37
diffeq	2566	6.5	0.01	12.3	0.01	26.6	0.07	65.0	0.50	155.9	2.80	3.66
dsip	2521	6.2	0.01	10.7	0.01	20.7	0.03	42.0	0.10	86.7	0.44	0.00
elliptic	5502	6.4	0.01	10.6	0.03	18.5	0.07	36.9	0.33	83.4	2.12	0.20
ex1010	7652	9.2	0.02	23.3	0.11	61.8	0.61	165.8	4.01	438.2	30.43	1.99
ex5p	1719	9.4	0.01	24.1	0.02	66.2	0.17	188.2	1.30	514.8	10.50	14.14
frisc	5905	7.1	0.01	14.4	0.04	32.3	0.16	79.8	0.88	209.0	6.30	1.24
misex3	2441	7.7	0.01	15.7	0.02	33.3	0.08	73.7	0.38	170.7	2.48	0.00
pdc	7527	9.4	0.03	24.8	0.12	67.4	0.68	183.7	4.41	489.4	31.71	4.40
s298	2514	7.9	0.00	17.5	0.02	44.0	0.13	121.9	0.94	346.5	7.10	7.56
s38417	12867	6.6	0.03	13.5	0.10	32.0	0.46	83.1	3.24	225.9	23.72	3.38
s38584	11074	6.1	0.03	11.4	0.06	22.4	0.20	46.7	0.98	101.5	5.81	0.86
seq	2761	7.5	0.00	15.2	0.02	31.7	0.08	68.6	0.37	153.3	2.25	0.04
spla	6556	9.6	0.03	25.8	0.11	73.9	0.69	215.5	4.98	561.4	31.14	13.83
tseng	1920	6.5	0.01	11.8	0.01	23.5	0.04	50.6	0.21	112.7	1.32	1.35
Average	**4954**	**7.56**	**0.01**	**16.22**	**0.05**	**38.05**	**0.22**	**95.15**	**1.38**	**240.0**	**9.61**	**2.94**

C/N is the number of cuts per node; T is time in seconds; L/N is the ratio of nodes with the number of cuts exceeding the limit (N=1000); for K < 8, the number of cuts did not exceed 1000

Computing Priority Cuts

- Consider nodes in a topological order
  - At each node, merge two sets of fanin cuts (each containing C cuts) resulting in (C+1) * (C+1) + 1 cuts
  - Sort these cuts using a given cost function, select C best cuts, and use them for computing priority cuts of the fanouts
  - Select one best cut, and use it to map the node

- Sorting criteria

Mapping pass	Primary metric	Tie-breaker 1	Tie-breaker 2
depth	depth	cut size	area flow
area flow	area flow	fanin refs	depth
exact area	exact area	fanin refs	depth

The tie-breaking criterion denoted “fanin refs” means “prefer cuts with larger average fanin reference counters”.

Priority Cuts: A Bag of Tricks

- Compute and use priority cuts (a subset of all cuts)
- Dynamically update the cuts in each mapping pass
- Use different sorting criteria in each mapping pass
- Include the best cut from the previous pass into the set of candidate cuts of the current pass
- Consider several depth-oriented mappings to get a good starting point for area recovery
- Use complementary heuristics for area recovery
- Perform cut expansion as part of area recovery
- Use efficient memory management

Priority-Cut-Based Mapping

Input: And-Inverter Graph

1. ~~Compute *K*-feasible cuts for each node~~
2. Compute arrival time at each node
   - In topological order (from PI to PO)
   - ~~Compute the depth of all cuts and choose the best one~~
   - Compute at most C good cuts and choose the best one
3. Perform area recovery
   - Using area flow
   - Using exact local area
   - In each iteration, re-compute at most C good cuts and choose the best one
4. Chose the best cover
   - In reverse topological order (from PO to PI)

Output: Mapped Netlist

Complexity Analysis

- The worst-case complexity of traditional mapping
  - FlowMap $O(Kmn)$ (J. Cong et al, TCAD '94)
  - CutMap $O(2Kmn^{\lfloor K \rfloor})$ (J. Cong et al, FPGA '95)
  - DAOmap $O(Kn^{\lfloor K \rfloor})$ (J. Cong et al, ICCAD'04)
- Mapping with priority cuts
  - $O(KC^2n)$

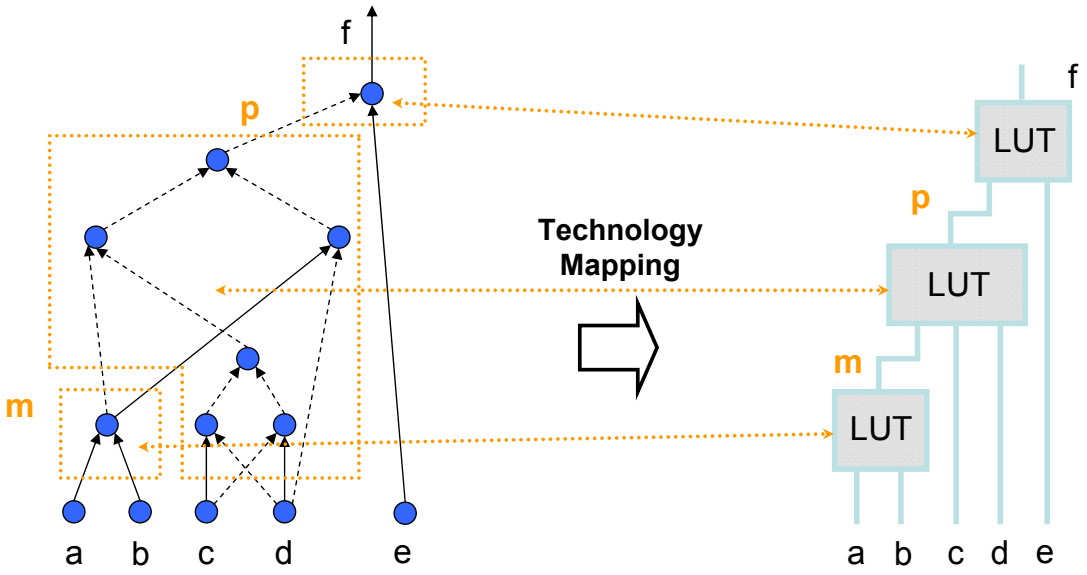
K is max cut size
C is max number of cuts
n is number of nodes
m is number of edges

(4) Structural Choices

- Structural bias
- Ways to overcome structural bias
  - Need some form of (re)synthesis to get multiple circuit structures
    - Computing and using several synthesis snapshots
    - Running several scripts and combining the resulting networks
    - Performing Boolean decomposition during mapping
- Multiple circuit structures = structural choices
- Questions:
  - How to efficiently detect and store structural choices?
  - How to perform technology mapping with structural choices?

Structural Bias

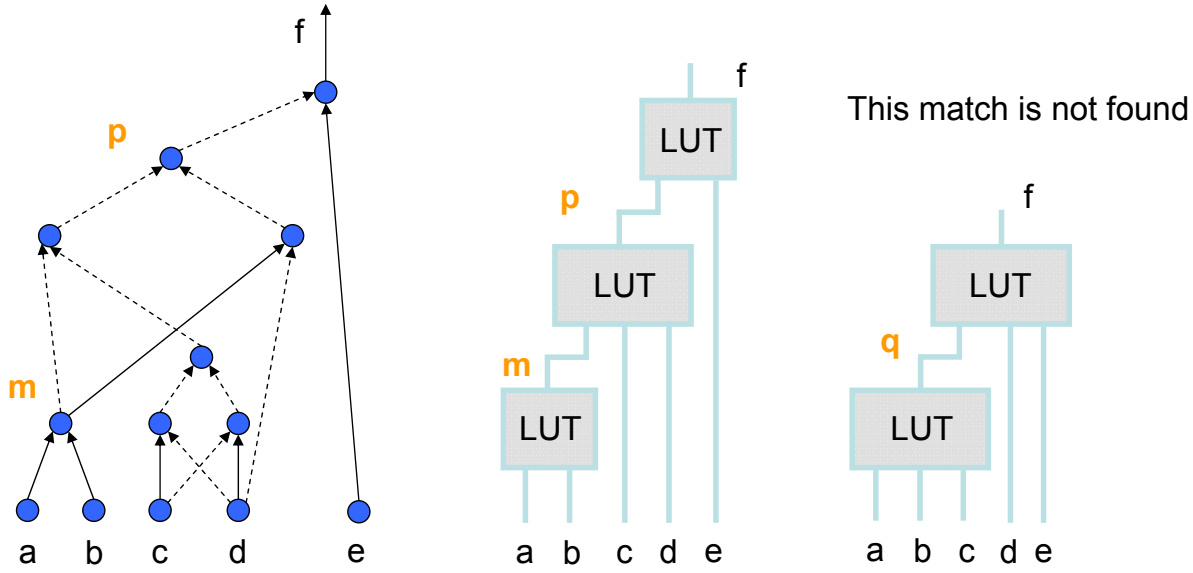
The mapped netlist very closely resembles the subject graph



Every input of every LUT in the mapped netlist must be present in the subject graph - otherwise technology mapping will not find the match

Example of Structural Bias

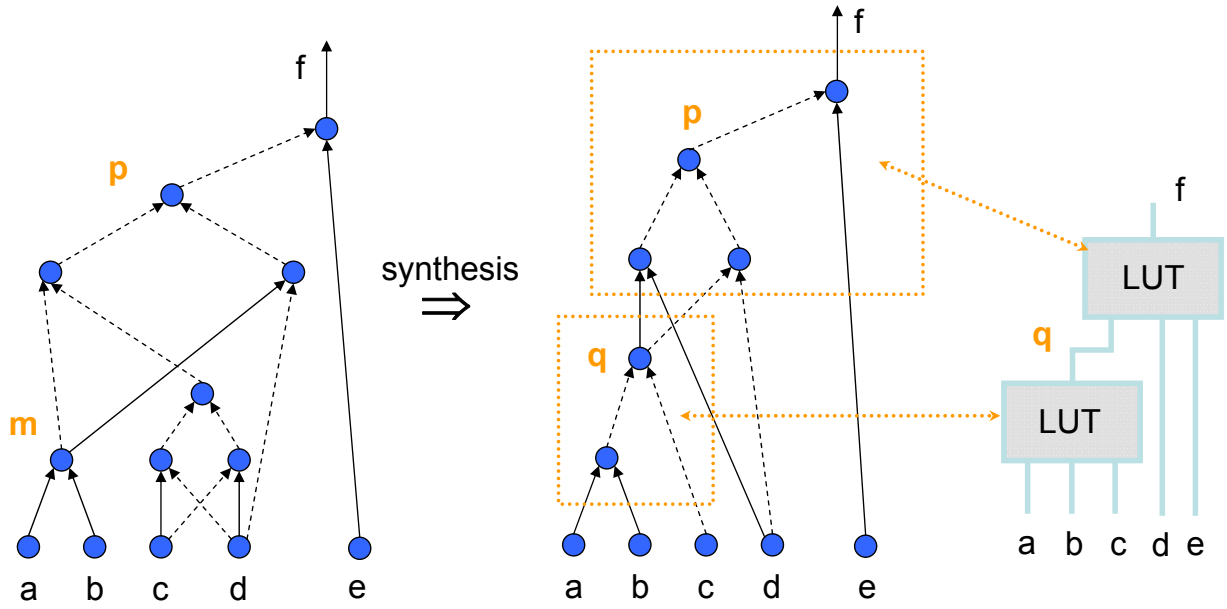
A better match may not be found



Since the point q is ***not*** present in the subject graph, the match on the right is ***not*** found

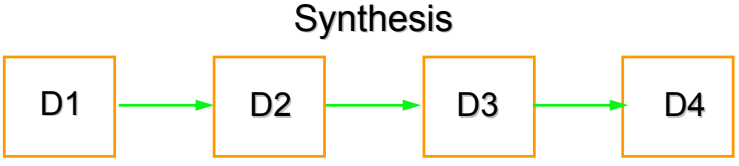
Example of Structural Bias

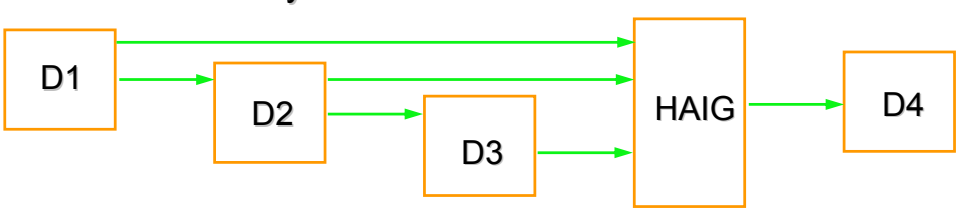
The better match can be found with a different subject graph



Synthesis for Structural Choices

- Traditional synthesis produces one “optimized” network
- Synthesis with choices produces several networks
  - These can be different snapshot of the same synthesis flow
  - These can be results of synthesizing the design with different options
    - For example, area-oriented and delay-oriented scripts





Mapping with Structural Choices

- Two questions have to be answered
  - How to store multiple circuit structures?
  - How to perform mapping with multiple circuit structures?
- Both questions can be solved due to the following:
  - The subject graph is an AIG
    - Structural hashing quickly merges isomorphic circuit structures
  - There are powerful equivalence checking methods
    - They can be used to prove equivalence
  - Cut computation can be extended to work with structural choices
    - The modification is straight-forward

Detecting Choices

Given two Boolean networks, create a network with choices

Network 1

x = (a + b)c

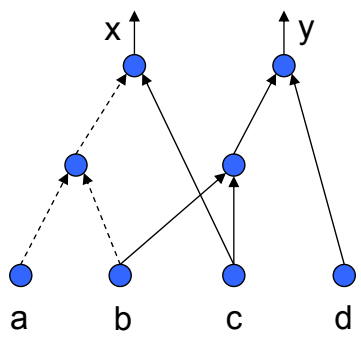
y = bcd

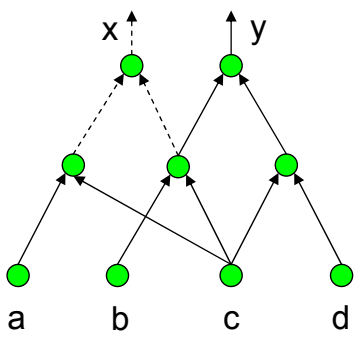
Network 2

x = ac + bc

y = bcd

Step 1: Make And-Inverter decomposition of networks





Detecting Choices

Step 2: Use combinational equivalence to detect functionally equivalent nodes up to complementation (A. Kuehlmann, TCAD'02)

- Random simulation to detect possibly equivalent nodes
- SAT-based decision procedure to prove equivalence

Network 1

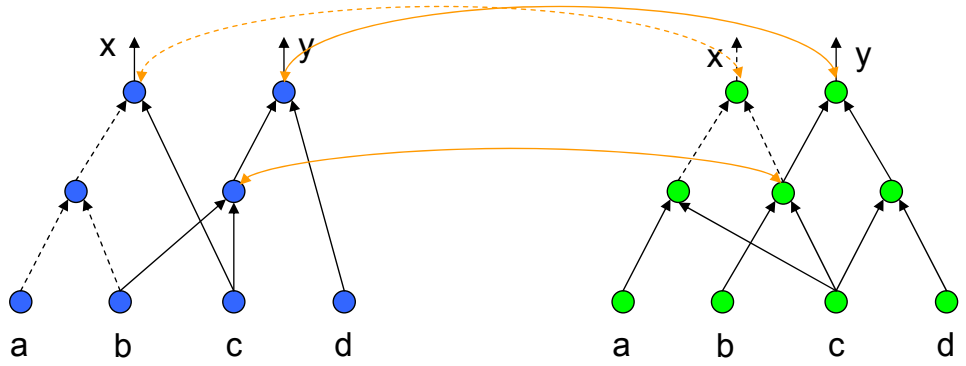
x = (a + b)c

y = bcd

Network 2

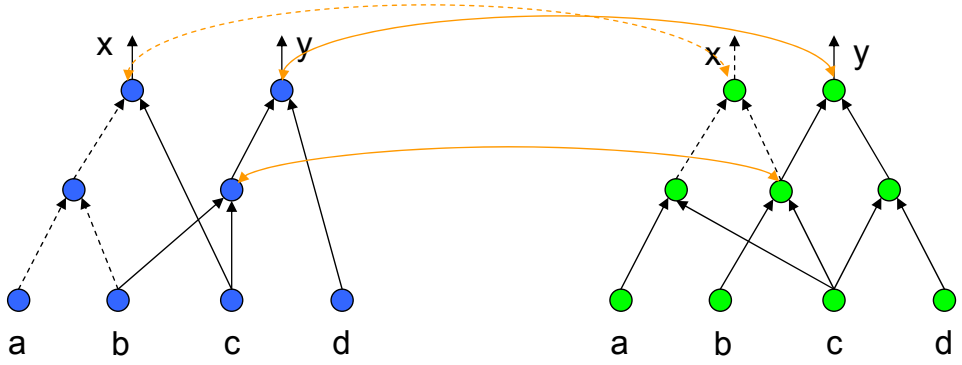
x = ac + bc

y = bcd

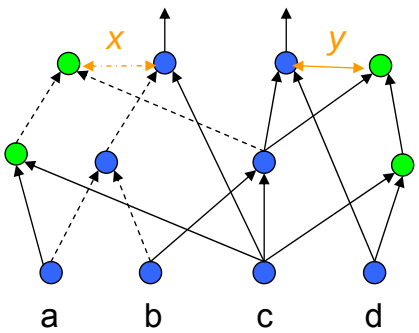


Detecting Choices

Step 3: Merge equivalent nodes with choice edges

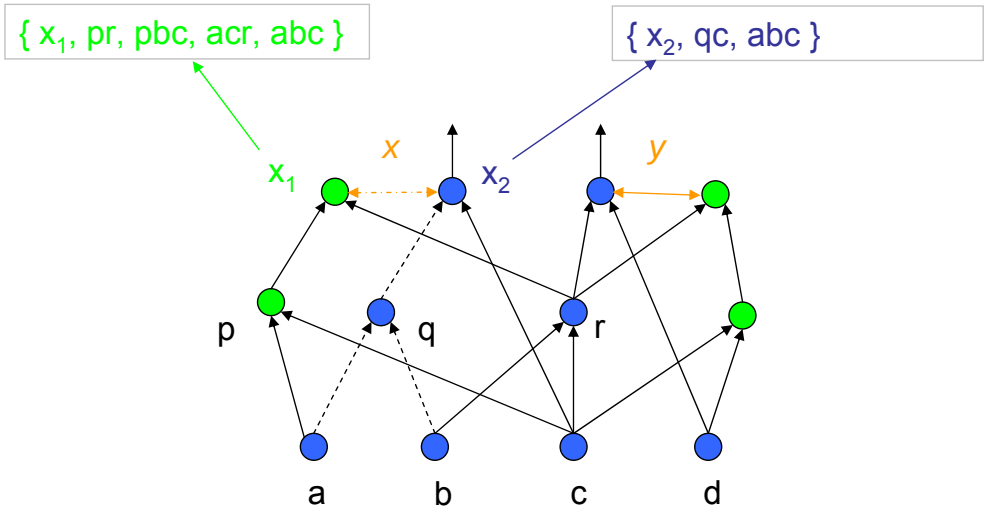


x now represents a *class of nodes* that are functionally equivalent up to complementation



Cut Computation with Choices

Cuts are now computed for *equivalence classes* of nodes



Cuts (x) = Cuts (x_1) ∪ Cuts(x_2)

= { x_1, pr, pbc, acr, abc, x_2, qc }

Mapping Algorithm with Choices

Only Step 1 has to be changed

Input: And-Inverter Graph with choices

1. Compute *K*-feasible cuts with choices
2. Compute best arrival time at each node
   - In topological order (from PI to PO)
   - Compute the depth of all cuts and choose the best one
3. Perform area recovery
   - Using area flow
   - Using exact local area
4. Chose the best cover
   - In reverse topological order (from PO to PI)

Output: Mapped Netlist

(5) Tuning Mapping for Placement

- Placement-aware cost function for priority-cut computation
  - The total number of edges in a mapped network
- Advantages
  - Correlates with the total wire-length after placement
  - Easy to take into account during area recovery
- Treat “edges” as “area” resulting in
  - Edge flow (similar to area flow)
  - Exact local edges (similar to exact local area)
- WireMap
  - New placement-aware mapping algorithm

Modified Cut Prioritization Heuristics in WireMap

- Consider nodes in a topological order
  - At each node, merge two sets of fanin cuts (each containing C cuts) getting (C+1) * (C+1) + 1 cuts
  - Sort these cuts using a given cost function, select C best cuts, and use them for computing priority cuts of the fanouts
  - Select one best cut, and use it to map the node
- Sorting criteria

Mapping pass	Primary metric	Tie-breaker 1	Tie-breaker 2
Depth	depth	cut size	area flow
area/edge flow	area flow	edge flow	depth
exact area/edge	exact area	exact edge	depth

WireMap Algorithm

Input: And-Inverter Graph

1. Compute *K*-feasible cuts for each node
2. Compute best arrival time at each node
   - In topological order (from PI to PO)
   - Compute the depth of all cuts and choose the best one
3. Perform area recovery
   - Using area flow and edge flow
   - Using exact local area and exact local edge
4. Chose the best cover
   - In reverse topological order (from PO to PI)

Output: Mapped Netlist

Experimental Results

- Experimental comparison
  - WireMap vs. the same mapper w/o edge heuristics
- WireMap leads to the average edge reduction
  - 9.3% (while maintaining depth and LUT count)
- Place-and-route after WireMap leads to
  - 8.5% reduction in the total wire length
  - 6.0% reduction in minimum channel width
  - 2.3% reduction in critical path delay
- Changes in the LUT size distribution
  - The ratio of 5- and 6-LUTs in a typical design is reduced
  - The ratio of 2-, 3-, and 4-LUTs is increased
- Changes after LUT merging
  - 9.4% reduction in dual-output LUTs

(6) Other Applications of Priority-Cut-Based Mapping

- Sequential mapping (mapping + retiming)
- Speeding up SAT solving
- Cut sweeping
- Delay-oriented resynthesis for sequential circuits

Sequential Mapping

- That is, combinational mapping and retiming combined
  - Minimizes clock period in the combined solution space
  - Previous work:
    - Pan et al, FPGA'98
    - Cong et al, TCAD'98
- Our contribution: dividing sequential mapping into steps
  - Finding the best clock period via sequential arrival time computation (Pan et al, FPGA'98)
  - Running combinational mapping with the resulting arrival/required times of the register outputs/inputs
  - Performing final retiming to bring the circuit to the best clock period computed in Step 1

Sequential Mapping (continued)

- Advantages
  - Uses priority cuts (L=1) for computing sequential arrival times
    - very fast
  - Reuses efficient area recovery available in combinational mapping
    - almost no degradation in LUT count and register count
  - Greatly simplifies implementation
    - due to not computing sequential cuts (cuts crossing register boundary)
- Quality of results
  - Leads to ~15% better quality compared to comb. mapping + retiming
    - due to searching the combined search space
  - Achieves almost the same (-1%) clock period as the general sequential mapping with sequential cuts
    - due to using transparent register boundary without sequential cuts

Speeding Up SAT Solving

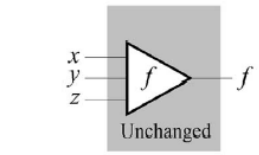
- Perform technology mapping into K-LUTs for area
  - Define area as the number of CNF clauses needed to represent the Boolean function of the cut
  - Run several iterations of area recovery
- Reduces the number of CNF clauses by ~50%
  - Compared to a good circuit-to-CNF translation (M. Velev)
- Improves SAT solver runtime by 3-10x
  - Experimental results are in the SAT'07 paper

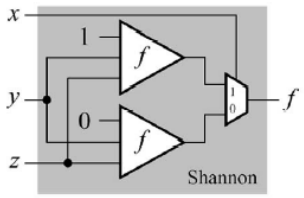
Cut Sweeping

- Reduce the circuit by detecting and merging shallow equivalences (proposed by Niklas Een)
  - By “shallow” equivalences, we mean equivalent points, A and B, for which there exists a K-cut C (K < 16) such that $F_A(C) = F_B(C)$
  - A subset of “good” K-cuts can be computed
  - The cost function is the average fanout count of cut leaves
    - The more fanouts, the more likely the cut is common for two nodes
- Cut sweeping quickly reduces the circuit
  - Typically ~50% gain of SAT sweeping (fraiging)
- Cut sweeping is much faster than SAT sweeping
  - Typically 10-100x, for large designs
- Can be used as a fast preprocessing to (or a low-cost substitute for) SAT sweeping

Sequential Resynthesis for Delay

- Restructure logic along the tightest sequential loops to reduce delay after retiming (Soviani/Edwards, TCAD’07)
  - Similar to sequential mapping
  - Computes seq. arrival times for the circuit
  - Uses the current logic structure, as well as logic structure, transformed using Shannon expansion w.r.t. the latest variables
  - Accepts transforms leading to delay reduction
  - In the end, retimes to the best clock period
- The improvement is 7-60% in delay with 1-12% area degradation (ISCAS circuits)
- This algorithm could benefit from the use of priority cuts





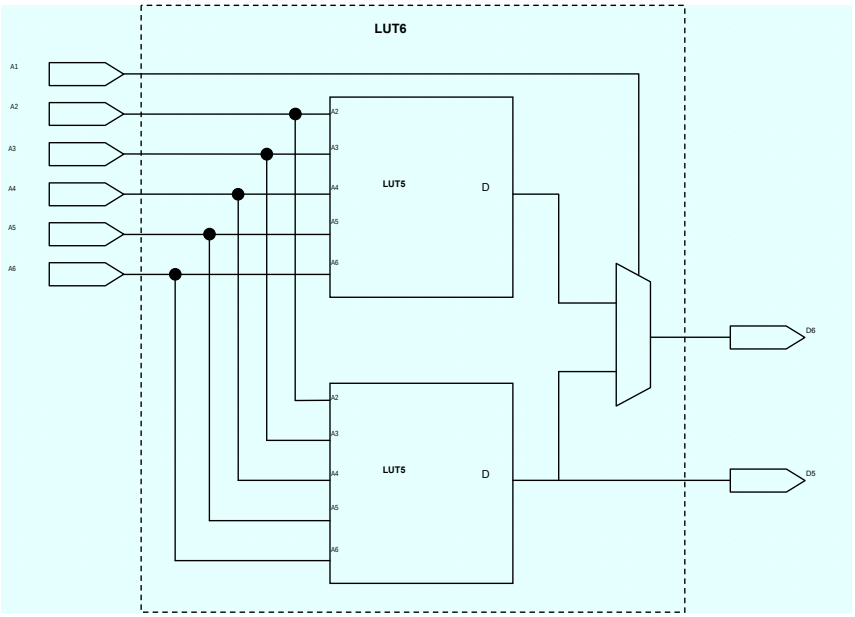
Summary

- Reviewed traditional and novel LUT mapping
- Presented the current mapping solution
  - Starts with an optimized AIG (with choices)
  - Performs exhaustive (or priority) cut computation
  - Performs heuristic area recovery
  - Uses placement-aware heuristics
- Experimental results are promising
- Future work
  - Area- and delay-oriented resynthesis for mapped networks
  - Using delay information from preliminary placement

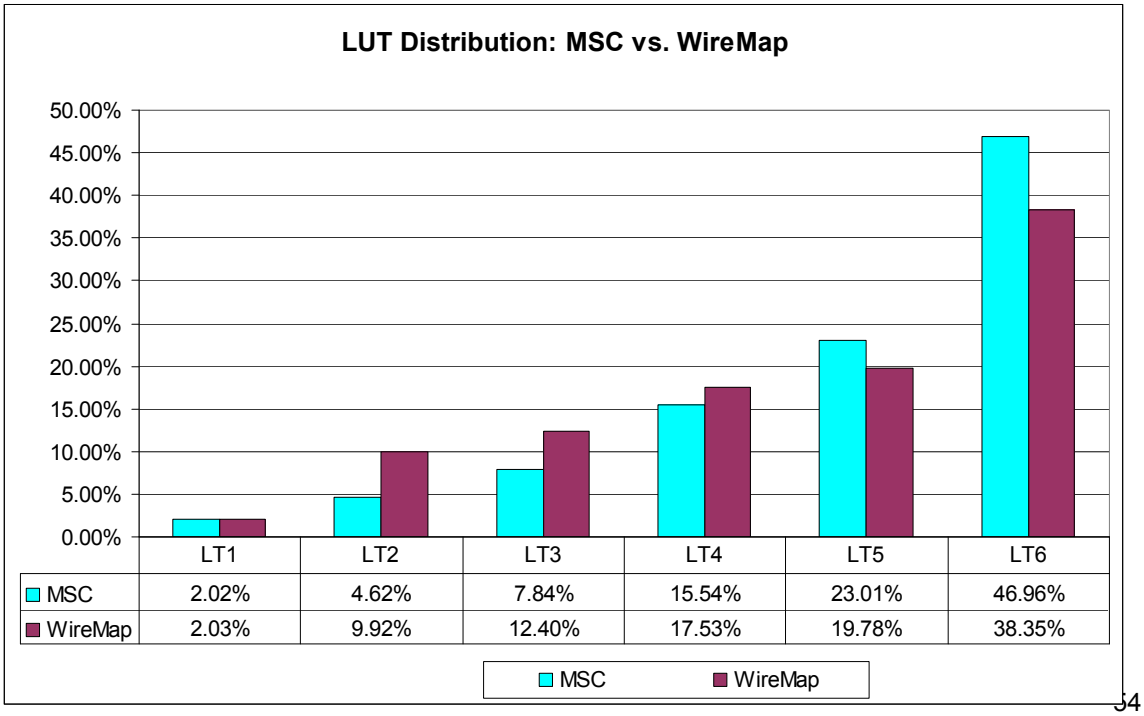
Backup Slides on WireMap

- Virtex-5 dual-output LUT
- Comparison of LUT distribution
- Comparison of area flow and edge flow mapping (K = 6)
- Wirelength, channel width, and critical path delay comparison

Virtex-5 Dual-Output LUT



Comparison of LUT Distribution



	LT1	LT2	LT3	LT4	LT5	LT6
MSC	2.02%	4.62%	7.84%	15.54%	23.01%	46.96%
WireMap	2.03%	9.92%	12.40%	17.53%	19.78%	38.35%

Comparison of Area Flow and Edge Flow Mapping (K = 6)

	Baseline					Mapping with Structural Choices					WireMap				
	luts	lev	edg	t,s	clb	luts	lev	edg	t,s	clb	luts	lev	edg	t,s	clb
alu4	807	6	3927	0.6	652	742	5	3520	6.79	585	742	5	3298	7.25	550
apex2	983	6	4664	0.75	778	807	6	3850	10.73	654	805	6	3574	11.11	603
b14	1214	13	5620	1.94	976	1162	13	5578	61.36	935	1163	13	5014	53.7	823
b15	2169	15	11073	2.25	1856	2103	15	10485	61.35	1804	2056	14	9499	51.21	1626
b17	6507	21	33151	6.73	5625	6480	18	32906	191.55	5602	6419	18	29552	169.62	5090
b20	2490	15	11953	3.95	2024	2380	14	11768	138.27	1980	2312	14	10582	118.02	1760
b21	2569	15	12418	4.15	2098	2391	14	11807	135.8	1995	2399	14	10781	116.26	1815
b22	3742	15	18027	5.89	3074	3613	14	17910	187.25	3053	3618	14	16426	180.92	2787
clma	3310	10	15576	2.72	2585	2392	9	11520	44.55	1952	2478	9	10846	42.61	1833
des	681	5	3541	0.94	624	502	4	2643	14.39	473	498	3	2192	15.48	370
ex5p	624	5	3019	0.6	497	562	4	2716	10.46	450	578	4	2666	10.49	437
elliptic	1800	10	8777	1	1662	1859	10	9173	17.95	1682	1807	9	8362	18.86	1569
frisc	1750	14	8610	1.35	1621	1798	12	8753	28.02	1668	1690	12	7662	24.37	1524
i10	629	9	2863	0.59	470	603	7	2765	9.81	465	574	8	2375	8.75	404
pdc	2305	7	11307	2.4	1923	2012	6	10061	77.7	1695	1891	6	8795	73.69	1476
s38584	2740	6	11574	1.63	1996	2667	6	11219	17.32	1948	2648	6	10580	17.54	1849
s5378	392	4	1553	0.3	253	357	4	1469	2.66	248	359	4	1346	2.69	226
seq	933	5	4521	0.67	750	737	5	3577	11.69	599	732	5	3276	11.21	551
spla	1862	6	9062	2.06	1538	1588	6	8065	52.58	1350	1515	6	7013	49.14	1198
tseng	657	7	2546	0.46	455	645	7	2488	5.18	452	645	6	2343	5.41	423
Geomean	1480	8.65	6988	2.05	1194	1346	7.97	6420	54.27	1102	1329	7.78	5824	49.42	998
Ratio	1	1	1	1	1	0.909	0.921	0.919	26.473	0.923	0.898	0.899	0.833	24.107	0.836
Ratio						1	1	1	1	1	0.987	0.976	0.907	0.911	0.906

Wirelength, Channel Width, and Critical Path Delay Comparison

	Baseline			MSC			WireMap		
	twl	mcw	cpd	twl	mcw	cpd	twl	mcw	cpd
alu4	15896	15	83.87	13594	13	78.94	14014	15	78.26
apex2	20995	16	88.45	17004	14	90.27	16197	14	90.97
b14	18331	11	148.02	18768	13	129.97	16265	10	149.57
b15	38895	15	180.51	36037	14	203.55	33401	14	195.91
b17	117551	16	249.05	120451	15	222.67	113153	15	225.29
b20	38672	11	152.00	39000	12	155.19	33885	12	139.94
b21	38684	11	143.18	39093	12	170.93	33791	11	135.72
b22	61069	13	157.05	63852	14	157.88	56914	13	150.88
clma	70021	18	167.45	48469	15	131.35	47018	15	136.31
des	19571	8	91.91	16944	10	101.23	13222	7	129.25
elliptic	28546	13	150.32	29670	14	181.29	24611	12	133.04
ex5p	12314	14	87.90	10346	13	73.26	11039	13	75.15
frisc	33763	15	159.11	35412	14	154.96	30398	14	134.79
i10	16383	9	211.59	17186	8	155.60	15103	8	162.49
pdc	64130	21	162.38	52978	21	148.93	47431	19	154.44
s38584	28083	11	72.55	26760	10	68.97	24723	9	71.15
s5378	4685	8	40.31	4358	10	49.26	4261	9	40.91
seq	20151	16	85.97	15640	16	86.58	15005	15	87.30
spla	44885	18	151.14	37925	19	130.99	34542	18	137.23
tseng	5718	7	49.91	5610	7	48.91	5365	7	47.47
Geomean	26524	12.76	119.55	24440	12.79	116.45	22364	12.03	113.81
Ratio	1.00	1.00	1.00	0.921	1.002	0.974	0.834	0.943	0.952
Ratio				1.00	1.00	1.00	0.915	0.94	0.977

twl = total wire length, mcw = minimum channel width required to route in VPR, cpd = critical path delay with min channel width across the three implementations