50. F. Najm, R. Burch, P. Yang, I. Hajj. “Probabilistic simulation for reliability analysis of CMOS VLSI circuits.” *IEEE Transactions on CAD*, 1990, volume 9, pages 439-450.
51. M. Pedram, B. T. Preas. “Interconnection length estimation for optimized standard cell layouts.” Proceedings of the *IEEE International Conference on Computer Aided Design*, pages 390-393, November 1989.
52. S. Rajgopal and G. Mehta. “Experiences with simulation-based schematic level current estimation.” Proceedings of the *International Workshop on Low Power Design*, pages 9–14, April 1994.
53. B. Rohfleisch, B. Wurth and K. Antreich. “Logic clause analysis for delay optimization.” in Proceedings of the *ACM/IEEE Design Automation Conference*, pages 668-672, June 1995.
54. B. Rohleisch, A. Kolbl and B. Wurth. “Reducing power dissipation after technology mapping by structural transformations.” Proceedings of the *ACM/IEEE Design Automation Conference*, June 1996.
55. K. Roy, S. C.Prasad. “Circuit activity based logic synthesis for low power reliable operations.” *IEEE Transactions on VLSI Systems*. 1(4):503-513, December 1993.
56. R. Rudell. “Logic Synthesis for VLSI Design.” Ph.D. thesis, University of California, Berkeley, 1989.
57. H. Savoj. “Don’t Cares in Multi-Level Network Optimization.” PhD thesis, University of California, Berkeley, 1992.
58. M. Schulz and E. Auth. “Advanced Automatic test pattern generation and redundancy identification techniques.” Proceedings of the *Fault Tolerant Computing Symposium*, pages 30-34, June 1988.
59. E. Sentovich, K. Singh, C. Moon, H. Savoj, R. Brayton and A. Sangiovanni-Vintentelli. “Sequential circuit design using synthesis and optimization.” Proceedings of the *International Conference on Computer Design*, pp328-333, October 1992.
60. A. A. Shen, A. Ghosh, S. Devadas, and K. Keutzer. “On average power dissipation and random pattern testability of CMOS combinational logic networks.” Proceedings of the *IEEE International Conference on Computer Aided Design*, November 1992.
61. A. A. Shen, A. Ghosh, S. Devadas, and K. Keutzer. “On average power dissipation and random pattern testability of CMOS combinational logic networks,” Proceedings of the *IEEE International Conference on Computer Aided Design*, November 1992.
62. D. Stirzaker. “Elementary Probability.” Cambridge University Press, 1994.
63. V. Tiwari, S. Malik, P. Ashar. “Guarded Evaluation: Pushing Power management to Logic Synthesis/Design.” Proceeding of the *Symposium on Low Power Design*, pages 221-226, April 1995.
64. V. Tiwari, P. Ashar, and S. Malik. " Technology mapping for low power. " *Proceedings of the 30th Design Automation Conference*, pages 74-79, June 1993.
65. C-Y. Tsui, M. Pedram, C-H. Chen, and A. M. Despain. “Low power state assignment targeting two- and multi-level logic implementations.” Proceedings of the *IEEE International Conference on Computer Aided Design*, pages 82–87, November 1994.
66. C-Y. Tsui, M. Pedram, and A. M. Despain. “Efficient estimation of dynamic power dissipation under a real delay model.” Proceedings of the *IEEE International Conference on Computer Aided Design*, pages 224–228, November 1993.
67. C-Y. Tsui, M. Pedram and A. M. Despain. “Power efficient technology decomposition and mapping under an extended power consumption model.” *IEEE Transactions on Computer Aided Design*, Vol.~13, No.~9 (1994), pages 1110--1122.
68. C-Y. Tsui. “Power Analysis and Optimization for CMOS Circuits.” Ph.D. thesis, University of Southern California, 1994.
69. A. Tyagi. “Hercules: A power analyzer of MOS VLSI circuits” Proceedings of the *IEEE International Conference on Computer Aided Design*, pages 530–533, November 1987.
70. T. Villa and A. Sangiovanni-Vincentelli. “NOVA: State assignment of finite state machines for optimal two-level logic implementations.” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9: 905-924, September 1990.
71. S. B. K. Vrudhula, H.Y. Xie. “Techniques for CMOS Power Estimation and Logic Synthesis for Low Power.” Proceedings of the *International Workshop on Low Power Design*, pages 21-26, 1994.
72. j. A. Waicukauski, E.B. Eichelberger, D.O. Forlenza, E. Lindbloom and T. McCarthy, “Fault simulation for structures VLSI.” *VLSI Systems Design*, pages 20-32, 1985.
73. A. Wang. “Algorithms for Multi-Level Logic Optimizations.” Ph.D. Thesis, UC Berkeley, 1989.

26. G. D. Hachtel, R. M. Jacoby and P. H. Moceyunas. “On computing and approximating the Observability Don’t Care Set.” Proceedings of *MCNC Workshop on Logic Synthesis*, 1989.
27. G. D. Hachtel and F. Somenzi. *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publishers, 1996.
28. J. Hong, R. G. Cain, and D. L. Ostapko. “MINI: A heuristic approach for logic minimization.” *IBM journal of Research and Development*, volume 18, pages 443–458, September 1974.
29. S. Iman and M. Pedram. “An approach for multi-level logic optimization targeting low power.” *IEEE Transactions on Computer Aided Design,* August 1996.
30. S. Iman and M. Pedram. “Multi-level network optimization for low power," Proceedings of the *IEEE International Conference on Computer Aided Design*, November 1994.
31. S. Iman and M. Pedram. “Logic extraction and factorization for low power.” Proceedings of the *ACM/IEEE Design Automation Conference*, June 1995.
32. S. Iman and M. Pedram. “Two-Level Logic Minimization for Low Power.” Proceedings of the *IEEE International Conference on Computer Aided Design*, November 1995.
33. S. Iman. *Logic Synthesis for Low Power VLSI Design*. Ph.D. thesis, University of Southern California, 1996.
34. K. Keutzer. " DAGON: Technology mapping and local optimization. " *Proceedings of the 24th Design Automation Conference*, pages 341–347, June 1987.
35. S. M. King. “Accurate simulation of power dissipation in VLSI circuits.” *IEEE Journal of Solid State Circuits*, 21(5):889–891, Oct. 1986.
36. T. Kirkland, M.R. Mercer. “A topological search algorithm for ATPG.” Proceedings of the *ACM/IEEE Design Automation Conference*, pages 502-508, June 1987.
37. W. Kunz and D. K. Pradhan. “Recursive learning: An attractive alternative to the decision tree for test generation digital circuits.” Proceedings of the *International Test Conference*, pp816-825, October 1992.
38. P.E. Landman and J. Rabaey. “Power estimation for high level synthesis.” Proceedings of the *European Conference on Design Automation*, pages 361-366, February 1993.
39. B. Lin and A. R. Newton. “Synthesis of multiple-level logic from symbolic high-level description languages.” Proceedings of *IFIP International Conference on Very Large Scale Integration*, pages 187-196, August 1989.
40. B. Lin and H. De Man. " Low-power driven technology mapping under timing constraints " *Proceedings of the International Conference on Computer Design,* pages 421-427, October 1993.
41. R. Lisanke, editor. “FSM benchmark suite,” Microelectronics Center of North Carolina, Research Triangle Park. North Carolina, 1987.
42. D. Liu and C. Svensson. “Trading speed for low power by choice of supply and threshold voltages.” *IEEE Journal of Solid State Circuits*, 28(1):10–17, January 1993.
43. C. K. Leonard, A.R. Newton. “An Estimation Technique to Guide Low Power Resynthesis Algorithms.” Proceedings of the *International Symposium on Low Power Design*, pages 227-232, April 1995.
44. D. Liu and C. Svensson. “Trading speed for low power by choice of supply and threshold voltages.” *IEEE Journal of Solid State Circuits*, 28(1):10–17, January 1993.
45. A. A. Malik, R. K. Brayton, A. R. Newton, and A. L. Sangiovanni-Vincentelli. “A modified approach to two-level logic minimization.” Proceedings of the *IEEE International Conference on Computer Aided Desig*n, Nov. 1988.
46. R. Marculescu, D. Marculescu, M. Pedram. “Logic level power estimation considering spatio-temporal correlations.” Proceedings of the *IEEE International Conference on Computer Aided Design*, pages 294-299, 1994.
47. D. Marculescu, R. Marculescu, and M. Pedram. “Information theoretic measures for energy consumption at register transfer level.” Proceedings of the *International Symposium on Low Power Design*, pages 81-86, April 1995.
48. P. McGeer and R. K. Brayton. “Consistency and Observability Invariance in Multi-Level Logic Synthesis.” Proceedings of the *IEEE International Conference on Computer Aided Design*, 1989.
49. S. Muroga, Y. Kambayashi, H.C. Lai, and J.N. Culliney. “The Transduction Method - Design of Logic Networks Based on Permissible Functions.” *IEEE Transactions on Computers*, October 1989.

actions on Computer-Aided Design of Integrated Circuits and Systems, volume 7, pages 723–740, June 1988.

4. L. Benini, M. Favalli, and B. Ricco. “Analysis of hazard contribution to power dissipation in CMOS ICs.” Proceedings of the *International Workshop on Low Power Design*, pages 27–32, April 1994.
5. R. K. Brayton, G. D. Hachtel, A. Sangiovanni-Vincentelli. “Multi-Level Logic Synthesis.” Proceedings of the *IEEE Tranactions*, 78(2):264-300, February 1990.
6. R. Brayton, G.D. Hachtel, C. McMullen and A. Sangiovanni-Vincentelli. “Logic Minimization Algorithms for VLSI Synthesis.” *Kluwer Academic Publishers*, Boston, 1984.
7. R. Brayton, C. McMullen. “The decomposition and factorization of boolean expressions.” Proceedings of the *International Symposium on Circuits and Systems*, pages 49-54, 1982.
8. R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, A. Wang. “Multi-level Logic Optimization and the Rectangular Covering Problem” Proceedings of the I*EEE International Conference on Computer-Aided Design*, 1987.
9. R. Bryant. “Graph-based algorithms for Boolean function manipulation.” *IEEE Transactions on Computers*, volume C-35, pages 677–691, August 1986.
10. J. B. Burr. “Stanford ultra low power CMOS.” Proceedings of *Hot Chips Symposium V*, pages 583–588, June 1993.
11. R. Burch, F. N. Najm, P. Yang, and T. Trick. “A Monte Carlo approach for power estimation.” *IEEE Transactions on VLSI Systems*, 1(1):63–71, March 1993.
12. E. Cerny. “An Approach to Unified Methodology of Combinational Switching Circuits.” *IEEE Transactions on Computers*, 27(8), 1977.
13. S. Chakravarty. “On the complexity of using BDDs for the synthesis and analysis of boolean circuits.” Proceedings of the *27th Annual Allerton Conference on Communication, Control and Computing*, pages 730–739, 1989.
14. A. Chandrakasan, M. Potkonjak, J. Rabaey, and R. W. Brodersen. “HYPER-LP: A system for power minimization using architectural transformations.” Proceedings of the *IEEE International Conference on Computer Aided Design*, pages 300--303, November 1992.
15. A. P. Chandrakasan, S. S. Scheng, and R. W. Broderson. “Low power CMOS digital design.” *IEEE Journal of Solid State Circuits*, 27(4):473–483, April 1992.
16. S. C. Chang, K. T. Cheng, N. S. Woo and M. Marek-Sadowska, “Layout driven logic synthesis for FPGAs.” Proceedings of the *ACM/IEEE Design Automation Conference*, pages 308-313, 1994.
17. K.T. Cheng and L. A. Entrena, “Multi-level logic optimization by redundancy addition and removal.” *European Conference on Design Automation*, pages 373-377, 1993.
18. O. Coudert, C. Berthet, and J.C. Madre. “Verification of Sequential Machines based on Symbolic Execution.” Proceedings of the *Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, France, 1989.
19. M. Damiani, G. De Micheli. “Observability Don’t Care Sets and Boolean Relations.” Proceedings of the *IEEE International Conference on Computer Aided Design*, pages 502-505, November 1990.
20. S. Devadas, K. Keutzer and J. White. “Estimation of Power Dissipation in CMOS Combinational Circuits using Boolean Manipulations.” *IEEE Transactions on Computer Aided Design*, vol 11, no 3, March 1992.
21. C-S. Ding and M. Pedram. “Tagged probabilistic simulation provides accurate and efficient power estimates at the gate level.” Proceedings of the *Symposium on Low Power Electronics*, September 1995.
22. L. A. Entrena and K.T. Cheng. “Sequential Logic Optimization by redundancy addition and removal.” Proceedings of the *IEEE International Conference on Computer-Aided Design*, pages 310-315, NOV 1993.
23. H. Goldstein. “Controllability/observability of digital circuits.” *IEEE Transactions on Circuits and Systems*, 26(9):685–693, September 1979.
24. A. Ghosh, S. Devadas, K. Keutzer, and J. White. “Estimation of average switching activity in combinational and sequential circuits.” Proceedings of the *ACM/IEEE Design Automation Conference*, pages 253–259, June 1992.
25. C. Halatsis and N. Gaitanis. “Irredundant normal forms and minimal dependence sets of a boolean function.” *IEEE Transaction on Computers*, pages 1064–1068, November 1978.

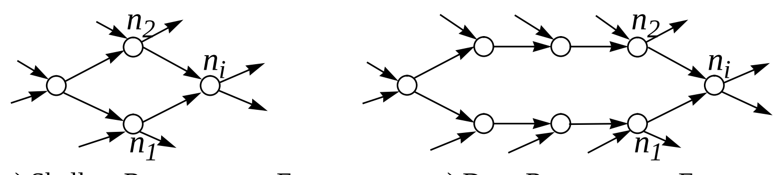

Figure 21. Reconvergent fanout regions.

The definition for deep and shallow regions is implementation dependent. However experimental results show that in a network that is decomposed into 2-input gates, taking into account Reconvergant regions that span over 5 levels capture a significant part of the spatial correlation at the input of nodes in the network. In POSE an option is available for using semi-local BDDs for computing node signal probabilities.

As the network is being optimized, the network structure is modified and therefore the definition of semi-local BDDs for each node will change. Therefore, if an optimization step changes the network structure drastically, the node switching activity values should be recomputed so that the new activity values reflect the new structure of the network.

It should also be noted that for small networks, using global BDDs provides a more efficient and exact technique for computing the signal probabilities. This is mainly due to the overhead of computing the semi-local BDDs for each node. For larger size networks, however, the use of semi-local BDDs will result in significant speed up during power estimation.

7 Acknowledgements

The authors would like to thank Chi-Ying Tsui for helpful discussions. The research was supported in part by DARPA under contract no. F33615-95-C1627, SRC under contract no. 94-DJ-559, and an NSF NYI award.

8 References

1. “SIS: A system for sequential circuit synthesis,” Report M92/41, UC Berkeley, 1992.
2. M. Alidina, J. Monteiro, S. Devadas, A. Ghosh and M.Papaefthymiou. “Pre-computation-based Sequential Logic Optimization for Low Power.” *IEEE Transactions on VLSI Design,* volume 2, pages 426-436, DecemProceedingber 1994.
3. K. Bartlett, R. K. Brayton, G. D. Hachtel, R. M. Jacoby, C. R. Morrison, R. L. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang. “Multi-level logic minimization using implicit don’t cares.” *IEEE Trans-*

This correlation may be present even if primary inputs are spatially uncorrelated. Under these conditions, the spatial correlation at the immediate fanins of a node *n* is due to the Reconvergant fanout regions in the transitive fanin cone of *n*. This means that an exact calculation of the signal probability for an internal node *n* requires that this signal probability be computed using the global function of *n*. BDDs have provided a more feasible approach for representing the global function of nodes in a Boolean network. Also [13] presents an efficient procedure for computing the signal probability of a function from its BDD representation. Therefore BDD based techniques are a good candidate for computing the signal probability of nodes in a network.

Representing the global function of nodes in some circuits may however become too expensive even when BDDs are used to represent the global functions. Therefore it is necessary to provide a mechanism for making speed-accuracy trade-offs when computing signal probabilities. In the following we describe and justify our technique for speeding up the procedure for computing these signal probability values.

6.3.1 *Computing Signal Probabilities using Semi-local BDDs*

Local BDD for a node *n* is defined as a BDD where immediate fanins of node *n* are used in building the BDD. The semi-local BDD for a node *n* is defined as the BDD where the nodes used as the BDD variables create a cut in the transitive fanin cone of *n*. Note that the global and local BDD for a node are special cases of the semi-local BDDs for that node.

The main reason for using global BDDs when computing the signal probability for a node *n* is to take into account the spatial correlation at the immediate fanins of the node. This correlation is due to reconvergant fanout regions in the transitive fanin cone of node *n*. Figure 21 shows a shallow and a deep Reconvergant fanout region in the fanin cone of node n. A shallow Reconvergant region spans over less number of levels than a deep reconvergant region. It can be stated that a shallow reconvergant region, in general, results in more spatially correlated fanins to a node. At the same time, a deep reconvergant region will in general result in a lower spatial correlation at the immediate fanins of node *n*. This means that it is possible to capture most of the spatial correlation at immediate inputs of a node n by using the semi-local BDD for that node. The semi-local BDD will take into account the spatial correlation due to shallow reconvergant region while ignoring the spatial correlation due to deep reconvergant fanout regions. This means that using semi-local BDDs allows us to account for most of the spatial correlation at the immediate fanins of a node without building the global BDD for that node. The use of semi-local BDDs for signal probability calculation is discussed in detail in [21].

This means that a power optimization procedure requires knowledge of the internal capacitances (diffusion capacitances) of all gates in the library to be able to compute the self-loading capacitance and internal power of the gate.

A complete description of substrate capacitances is obtained by analyzing the transistor circuit of each library cell. Figure 20 shows a 3-input NOR gate with 3 NMOS transistor in parallel with one end connected to ground and the other end connected to gate output. The PMOS section is made of 3 PMOS transistors in serial connection with one end connected to V_{cc} and the other end connected to the gate output. There are three diffusion capacitances C_0, C_1 and C_2 in this gate. Each of these substrate capacitances have different charging and discharging functions. To charge C_1 for example, it is required that *tr1* be conducting so that a circuit path between V_{dd} and the capacitance is established. To discharge the capacitance, both *tr2* and *tr3* have to be on and at least one of the transistors in the NMOS network has to be on so that a circuit path from the capacitance to ground is established. In summary, to completely specify the diffusion capacitances, we not only need the capacitance value, but also the charging and discharging function associated with each capacitance. The charging and discharging function can be found by analyzing the switch network of the library cell. The self-loading capacitance for a gate is also computed using the diffusion capacitance information.

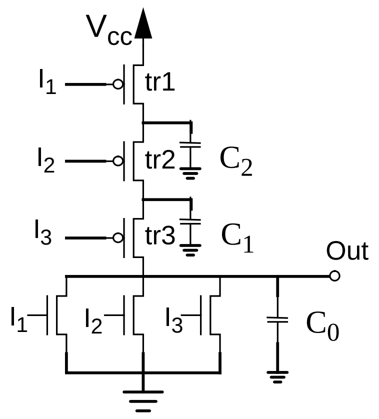

Figure 20. Diffusion capacitances for a 3 input NOR gate.

6.3 POWER ESTIMATION

The immediate fanins of an internal node are in general spatially correlated[1].

1.Spatial correlation between nodes x and y in a network means that the values of x and y in the same clock cycle are dependent.

6.2 DESIGN SPECIFICATION

In the past, logic synthesis has concentrated on minimizing the area of a circuit while meeting the timing constraints. The design specification for logic synthesis therefore consisted mainly of providing the functional description of the circuit, the timing constraints and the area/delay characteristics of the target library. As logic synthesis environments are extended to take into account power consumption, the conventional design specification techniques prove to be inadequate. In this section we discuss information that is necessary for effective power estimation and optimization at the logic synthesis stage.

The power consumption of a CMOS circuit is a function of the expected number of times the logic values in the circuit change values. This means that unlike conventional logic synthesis where the circuit performance (delay) is only affected by the *physical* characteristics of the surrounding logic (input drives and output loads), the transient behavior of the circuit has a significant impact on the power consumption of the circuit. In order to exactly capture the transient behavior of a circuit, it is necessary to provide a sequence of bit vectors applied at the primary inputs of the network. This sequence of bit-vectors may in turn be used by *statistical power estimation techniques* to compute an estimate of the expected switching rate of gates in a network thus providing the necessary information for power estimation. Recently, probabilistic techniques have provided an efficient approach for capturing this transient behavior of circuits. By providing the expected behavior of the circuit at the primary inputs in terms of probability values, *probabilistic power estimation techniques* [66] [21] [46] provide an estimate of expected switching rates of the internal gates in a circuit.

Libraries used during logic synthesis only provide information on area and delay of each gate in the library. More information is however required to accurately measure the power consumption of a gate in a technology mapped network. The power consumption of a gate consists of the power consumed at the output of the gate and the power internal to the gate.

The power at the output of a gate *g* is a function of the load seen at the output of this gate. This load is a combination of the input loads for output gates and also the *self-loading capacitance* for the gate itself. The self loading capacitance for a gate is defined as the load driven by the gate when the gate output is left open and is due to the source/drain diffusion capacitances of the gate. Experimental results show that self-loading capacitances contribute up to 20% of the total power consumption in CMOS circuits. Ignoring these capacitances will no doubt affect the accuracy of power estimation and the optimality of power optimization. The *internal power consumption* of a gate is computed by measuring the power required to charge and discharge the internal capacitances of this gate.

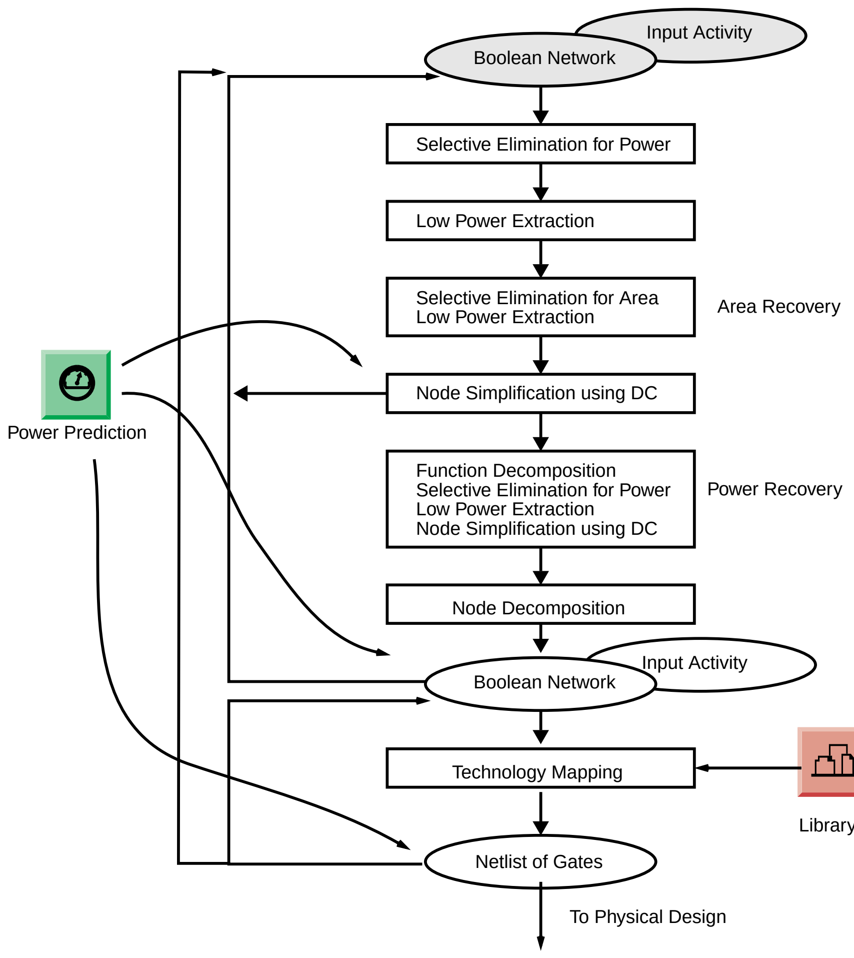

Figure 19. Pose synthesis methodology.

for power optimization. Therefore the “power recovery” stage is included to use this flexibility.

Note that at some points during the optimization, the quality of the results are checked using the power estimation utilities. This is to check how much reduction in power has been obtained after each iteration. The synthesis process is terminated if no further improvement can be obtained or the resulting power estimate is within the design specifications. The next section presents a detailed discussion on the design specification requirement and power estimation.

ation, a design has to meet the target power consumption. This means that power estimation procedures are necessary to check design compliance with the given specifications. At the same time, power estimation is necessary to check the quality of the synthesis steps. Power estimation is also a crucial part of interactive optimization techniques.

Synthesis procedures are the basic steps used to incrementally change the circuit structure while optimizing the target cost function. The combination of these steps guided by the power estimation procedures are used to develop a power optimization methodology.

Figure 19 presents the power optimization methodology used in POSE. The highlighted boxes specify the information that has to be provided by the designer. The input to POSE is a Boolean network specification describing the circuit to be designed. The set of power information required by design specifications is also provided at this time. A Boolean network is a directed graph where each node represents either a Boolean function or a latch element. The Boolean network is then optimized using a set of logic synthesis operations.

Low power optimization algorithms provided in POSE consist of four categories of optimization techniques: 1)Low power algebraic restructuring techniques 2)Low power node simplification using power compatible don't cares, 3)Low power technology mapping and 4)Post mapping structural optimization techniques. Techniques in categories 1 and 2 are used during the logic synthesis process while the technology mapping is used to map the optimized networks to gates in the target technology. Post mapping structural optimization is applied to the network after it is mapped to the target technology.

It should be noted that power optimization techniques are developed to make area-power trade-offs. This is mainly done by reducing the power density[1] of the network so that even with the larger area, the total power consumption is smaller than the area optimized network. In fact if a power optimization technique reduces power by reducing area, then it should be considered as an area optimization technique. In the methodology given in figure 19, the initial logic restructuring have been performed for minimum power. This means that network area may not have maximally decreased during this operation. Therefore the methodology includes a number of operations for recovering some area. This is called "area recovery". A "power recovery" stage is also included before the technology mapping process. The reason for this is that decomposition for minimum power does not maximally decompose all node functions. In other words it leaves some area redundancy in the network that may still be taken advantage of

1.Power density is defined as the total power divided by the area of the network.

bilities of these design steps. The more significant problem faced by the lack of a unified framework for low power design is that designers are forced to use low power techniques as isolated procedures. This has been a major obstacle in developing a methodology for effective and efficient power specification, estimation and optimization. This lack of a methodology in turn results in a limited understanding of the applicability of existing techniques which contributes to holding back the state of the art technology. At the same time, many optimization techniques are only applicable and relevant when applied in conjunction with other optimization approaches. This means that without a unified framework, many new techniques will not be discovered.

In order to address this problem, a complete system for designing low power digital circuit needs to be developed. The overall design flow consists of a process where the initial design specification is optimized using the techniques provided at the behavioral, RT, logic and physical levels. The inputs to each level is the set of power relevant information which includes the necessary library information. The results of each stage is also checked using power prediction tools provided in the optimization environment.

This section presents a methodology for designing low power digital circuits at the logic levels and discusses issues behind design specification for power and power estimation during the optimization process.

6.1 THE FLOW

A low power design methodology can only be developed when a number of key components are made available to the designer. These components fall into the following categories. *Design Specification*, *Design Verification* and *Synthesis Procedures*. Complete design specification is necessary in order to provide the synthesis environment with maximum information necessary for the optimization process. For example, when designing a circuit for minimum area, the design specification for this synthesis process should include a measure of the area for the gates in the target library. Similarly, a synthesis environment for low power should include all the power related information that is necessary during the synthesis and validation procedure. An important issue to also consider is that design specification for power may not be the same at different levels of abstraction and therefore a set of consistent specification standards need to be available at different levels.

Design validation is also an important part of any design methodology. The final design has to comply with the design specifications. For conventional logic synthesis, this validation is in the form of checking functional correctness and checking that design meets the area and speed requirements. For power consider-

Experimental results show that considering all functions of pairs of wires will in general provide reasonable amount of flexibility for finding substitute wires. In order to obtain a good balance between flexibility and efficiency it is decided to only look at all functions of pairs of wires in the circuit. This means that for every pair of wires x and y, all possible functions of these two variables (which include x, y and their complement and excluding constants 0 and 1) are also checked as candidate substitutions.

The problem with the approach presented so far is that it is very expensive to perform functional equivalency as proposed in section 5.2. This is because operations on global BDDs are very expensive for large circuits and therefore it is not possible to check a large number of candidate wire as a possible substitute wire. Functional simulation can be used to greatly speed-up the functional equivalence check as described in [33]. Experimental results show that the number of global BDD compare operations is significantly reduced for a small number of vectors simulated at the input. For example for 256 vectors simulated at the circuit inputs, the number of required checks using global BDDs is reduced by two orders of magnitude.

6 Power Optimization Methodology

In the past, the main objective for circuit designers has been to design fast and compact digital systems. In response to this demand, design automation tools have been developed to help the designers in automatic synthesis of digital circuits. In addition to synthesis tools, other automated programs have also been developed for simulation and verification of these synthesized circuits. By taking advantage of these tools and combining their features, designers have been able to develop and apply novel design methodologies enabling them to design faster and more complex systems while speeding up the design process. These design automation tools have been used extensively in the industry and are an integral part of any design cycle.

Even though considerable effort has been made in creating new techniques for power estimation and optimization, a unified framework for designing low power digital systems has not yet been developed. The void created by the absence of such a tool has presented designers with serious problems. Optimization algorithms that target low power circuits use the frameworks designed for synthesizing minimum area and delay circuits. This means that critical information needed for power estimation and optimization is not available when low power techniques are applied. In most cases, minimal information is made available to the power related procedures which in turn results in reducing the capa-

Once the ODC is calculated, this condition is easily checked for by using global BDDs. Once a possible candidate substitution is identified, the value for this substitution is computed using equation 22 and compared to other possible substitutions.

A number of issues need to be considered while searching for candidate substitutions. These issues include the techniques for computing ODCs, increasing the search space for finding candidate substitutions and computational complexity of the optimization process.

Using ODC computed for low power, it is guaranteed that changes in the function of a node will not result in increasing the power consumption in the transitive fanout of the node. This means that it is not necessary to analyze the effect of changes in the switching activity of transitive fanout nodes. Using ODCs computed for low power, it is also guaranteed that any wire substitution will not increase the power consumption in the transitive fanout nodes. Therefore it is possible to only consider the effect of power changes due to regions A and B in figure 18. It should however be noted that even though all possible substitutions will never result in an increase in the power consumption of nodes in the transitive fanout nodes, some substitutions may reduce the power in the transitive fanout nodes more than other substitutions. Therefore it is worthwhile to also consider this component of power reduction in selecting one substitution over another one. At the same time it should be noted that using ODCs computed for low power will in general reduce the number of possible substitutions which may potentially lead to a large decrease in power by removing some gates from the network which fall into region A in figure 18. Therefore if a decision is made to also take into account the change in power in the transitive fanout of a node, then ODCs can be computed without regard for power consumption. However if it is too expensive to keep track of this change in the transitive fanout power, then ODCs should be calculated for low power.

ODC computation procedures should also be extended to be used with input substitution. ODC computation techniques discussed previously can directly be used for output substitution since the ODC computed for the gate that is driving the target wire gives the ODC conditions for the target wire.

As pointed out earlier, only considering the existing wires in the circuit as candidate substitute wires will not provide enough flexibility in performing structural optimizations. In general a substitute wire can be generated by combining any number of existing wires in the circuit using any function of these wires. This will however generate a large number of candidate wires to be checked as substitute wires. Given a target wire w and m other wires that may be considered as substitute wires, there are 2^{2^m} possible functions to check for substitute wires. This number is very large and cannot in practice be considered.

$$Value = \sum_{w \in A} C_w \cdot E_w - \sum_{w \in B} C_w \cdot E_w + \sum_{w \in C} C_w \cdot \Delta E_w \qquad (22)$$

where regions A, B and C are shown in figure 18 and correspond to the three sources of change in power. E_w corresponds to the switching activity of wire w and C_w gives the load seen by wire w. Change in switching activity in the last part of this equation corresponds to the change in the switching activity of wires in the transitive fanout of gate g_1 as it is replaced with the output of gate g_2. Note that input substitution is a special case of output substitution and the value for an input substitution can be computed using the same equation except that region A will be an empty region. The reason is that for an input substitution, only one of the fanout edges of a gate is replaced with another wire and therefore the gate is required to drive the other fanout edges therefore no gate can be removed from the circuit after an input substitution (unless the gate has only one fanout in which case input substitution and output substitution will be equivalent operations). At the same time, for a single wire substitution (without needing an inverter) region B will be an empty region since no new gates will be added to the circuit after the substitution.

5.2 FUNCTION SUBSTITUTIONS

In a Boolean network, in order to replace a wire a with another wire b, it is necessary that after replacing wire a with wire b, the behavior of the circuit at the outputs is not changed. As pointed out before, the problem of structural optimization is simplified by only looking at wire substitutions that replace the target wire at the sink. This means that the function of the sink gate for the target wire need not be modified. In this context, the problem of identifying a replacement wire for a target wire is formulated as follows: Given a target wire w_1 and a wire w_2, determine if w_2 can replace w_1.

In the absence of observability don't care conditions, this substitution is only possible if the global function of w_2 is the same as the global function of w_1. In this case, wire w_1 can be replaced with wire w_2. Even though this condition is easy to check for using BDDs, in general the number of such wires is very small. The conditions for replacing wires can be relaxed by taking into account the observability don't care conditions for wire w_1. Given F_1, the global function and ODC, the observability don't care conditions for wire w_1 and F_2, the global function of wire w_2, then wire w_1 can be replaced with wire w_2 if and only if:

$$F_1 - ODC \subseteq F_2 \subseteq F_1 + ODC \qquad (23)$$

ing existing wires does not provide enough flexibility in optimizing the cost of the network. In order to provide more wires that may be considered as replacement wires, complement of existing wires are also considered. In addition, a new set of wires are generated by combining pairs of existing wires in the network. This operation is generalized as follows: For every pair of wires x and y, all possible functions of these two variables (which include x, y and their complement and excluding constant 0 and 1) are also considered as candidate substitutions.

5.1.1 *Cost Functions*

The goal of structural optimization is to reduce the switched capacitance of a Boolean network after it is mapped to the target technology. Since the network is already mapped to the target technology, it is possible to find an accurate cost for each possible wire replacement operation. In this section, a cost model is presented for taking into account all factors affecting power consumption when a wire is replaced with another wire in the network.

Replacing a wire in a netlist of gates will result in changing the total switched capacitance of the circuit. Figure 18 shows an example where output of gate g_1 is substituted with wire *d* which is the output of a new gate formed by forming the AND of wires *b* and *c*. Three sources of change in power can be identified in this example: 1)power reduction due to removing gate g_1 and all nodes in its exclusive transitive fanin cone, 2)power increase due to inserting gate g_2 in the network which results in increased load on its inputs and 3)the power change due to the change in the switching activity values in the transitive fanout cone of gate g_2. Note that this example shows the change in power for an output substitution. The value of a wire substitution gives the reduction in the power consumption of the network when the substitution is performed and is computed using equation 22.

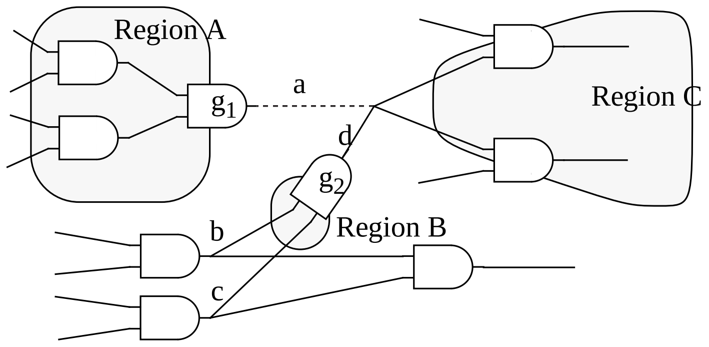

Figure 18. Change in power consumption after a substitution.

able stuck-at fault. Therefore, each new wire is checked to be a redundant wire before the target wire is removed.

The approach used in [54] is based on logic clause analysis [53]. A clause is defined as sum of variables and a valid clause describes dependencies among its variables. A clause is said to be valid if and only if it evaluates to 1 for all assignments to the primary inputs of the circuits. Global implications introduced in [58] correspond to valid global clauses which cannot be derived from a single gate's formula. A valid global clause describes global signal dependencies and therefore can be used to optimize the cost that is being minimized. In [54], bit parallel fault simulation (BPFS) [72] and ATPG techniques have been used to compute a set of valid clauses which correspond to single wire or a combination of wires that can be used to replace a target wire. This information is then used to minimize the delay in the network.

Previous techniques for structural optimization have been based on test pattern generation and fault simulation techniques.In the next section, a technique is presented for performing post-mapping structural optimization by using the observability don't care conditions which are derived from the network structure.

5.1 CANDIDATE WIRES

Redundancy addition followed by redundancy removal is the underlying technique in structural network optimization. In general it is possible to add redundancy in one part of the network followed by removing redundancies from a different part of the network. The approach presented in [16] makes use of sets of mandatory assignments to find the redundancy that needs to be added to a network in order to remove a target wire. The time complexity of this operation however, proves to be very high. A different approach for removing a wire w is to only consider the set of wires that can be used for source-replacing wire w. This technique is equivalent to redundancy addition/removal where the added redundancy is limited to the same gate that the target wire is used in. Using this restriction, the problem is formulated as follows:

Problem: *Given a Boolean network mapped to the target technology, a target wire w (or a target gate g), find all possible wires that can be used to input substitute wire w (or output substitute gate g).*

The goal in structural optimization is to take advantage of the redundancy present in the current implementation of the network being optimized. Therefore the first step in performing structural optimization is to consider substitutions that involve only a single wire that is already present in the network. An extension of this technique is to also consider wires that are combinations of other existing wires in the network. Experimental results however, show that consider-

Given a Boolean network, it is often possible to replace a wire with an existing wire or a combination of other wires in the network. Consider the example in figure 17. As can be seen in this example, function *f* can be expressed both in terms of inputs *a* and *d* and also inputs *e* and *d*. This flexibility in expressing the function of *f* can be exploited to optimize different costs in the network. For example, if path *a*-*e*-*f* in configuration A is on the critical path, then delay can be reduced by replacing wire *e* with wire *a* and therefore reducing the circuit delay. If the switching activity of wire *a* is less than the switching activity of wire *e*, then power can also be reduced by replacing wire *e* with wire *a*.

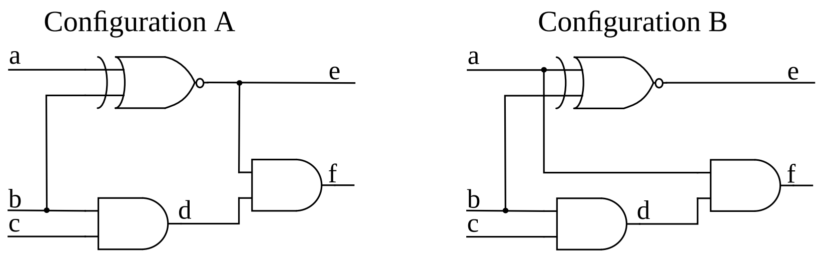

Figure 17. Replacing a wire with another wire in the network..

The main task in performing post-mapping structural optimization is to find the set of possible wire replacements for a given wire in the network. Once the set of candidate replacement wires has been identified, the best replacement can be selected by assigning a value to each candidate wire which gives the contribution of this replacement to the cost of the network being minimized.

A number of approaches have been presented in the past for performing structural optimization. The approach presented in [16] uses redundancy addition/removal techniques to find alternative wires for a given wire in the network. Based on this approach, the alternative wire, when added to the network, makes the target wire redundant. A wire is redundant if and only if the corresponding stuck-at fault is un-testable. The concept of mandatory assignments introduced in [22] is used in [16] to find alternative wires for a target wire. When testing a wire for a stuck-at fault, the set of mandatory assignments are the value assignments required for a test vector to exist for this stuck-at fault. The set of mandatory assignments are computed in [16] by using implication [36] [37] and recursive learning [58] techniques. If the set of mandatory assignments for a wire cannot be satisfied, then the corresponding stuck-at fault is redundant. Therefore a target wire is made redundant by adding wires to the network that will make the set of mandatory assignments for this wire inconsistent. It should be guaranteed that adding the new wire will not change the behavior of the circuit. This can be guaranteed by checking that adding the new connection forms an un-testnals in the networks. In doing so, it is possible to optimize different costs in the network. For example delay optimization is performed by replacing a signal in a network with an equivalent signal while this replacement will result in reducing the critical path in the network. In [17] a method is presented where a Boolean network is optimized by first inserting redundancies in the network and then removing other sets of redundancies which lead to a lower total cost of the network. In [16] a method is presented that attempts to identify alternative wires and alternative functions for a target wire that cannot be routed due to the limited routing resources in an FPGA. Alternative wires that can be routed through less congested areas substitute the un-routable wires without changing the circuit's functionality. Again, redundancy addition and removal techniques have been used to find alternative wires in the circuit. In both these methods, automatic test pattern generation (ATPG) techniques have been used to check for functional correctness of the circuit when checking for candidate alternative wires.

The techniques presented in [16], [17] and [53] have been extended in [54] to minimize the power consumption of Boolean network. The techniques for structural optimization have been applied after the Boolean network is mapped to the target technology. The reason for this is that cost functions can more accurately be calculated for a netlist of gates since all technology dependent information such as area, delay and load values are readily available from the netlist of gates. In this section, a technique for minimizing the power consumption of netlist of gates based on structural optimization techniques is presented. The technique presented in this section is based on an incremental approach where observability don't cares in the network are first calculated and then possible substitutions are explored and taken advantage of when power consumption can be reduced by making the substitution.

The following notation will be used in this section to describe the post-mapping optimization techniques. A *wire* in a netlist of gates corresponds to an edge in the graph that represents the Boolean network corresponding to the netlist of gates. Source of a wire corresponds to the gate that drives the wire and sink of a wire corresponds to the gate that the wire is driving. Replacing a wire w_1 with a wire w_2, in the most general case, refers to adding a new wire w_2 to the network where this addition allows us to remove wire w_1. Source-replacing a wire w_1 with a wire w_2 refers to replacing the sink of wire w_1 with the source for wire w_2. Note that source-replacing wire w_1 with a wire w_2 is a special case of wire replacement where the sink for wire w_2 is constrained to be the sink for wire w_1. Source-replacing is also defined as input-substitution. Output substitution of gate *g* with wire *w* refers to source-replacing all fanout wires of g with wire *w*. A target wire is defined as wire that is considered for removal. An alternative wire is wire that will be used to replace the target wire.

The approach presented in [65] consists of two steps. In the first step, power-delay curves (that capture power consumption versus arrival time trade-off) at all nodes in the network are computed. In the second step, the mapping solution is generated based on the computed power-delay curves and the required times at the primary outputs. For a NAND-decomposed tree, subject to load calculation errors, this two step approach finds the minimum area mapping satisfying any delay constraint if such a solution exists. Compared to a technology mapper that minimizes the circuit delay, this procedure leads to an average of 18% reduction in power consumption at the expense of 16% increase in area without any degradation in performance.

Generally speaking, the power-delay mapper reduces the number of high switching activity nets at the expense of increasing the number of low switching activity nets. In addition, it reduces the average load on the nets. By taking these two steps, this mapper minimizes the total weighted switching activity and hence the total power consumption in the circuit.

Under a real delay model, the dynamic programming based tree mapping algorithm does not guarantee to find an optimum solution even for a tree. The dynamic programming approach was adopted based on the assumption that the current best solution is derived from the best solutions stored at the fanin nodes of the matching gate. This is true for power estimation under a zero delay model, but not for that under a real delay model.

The extension to a real delay model is also considered in [66]. Every point on the power-delay curve of a given node uniquely defines a mapped subnetwork from the circuit inputs up to the node. Again, the idea is to annotate each such point with the probability waveform for the node in the corresponding mapped subnetwork. Using this information, the total power cost (due to steady-state transitions and hazards) of a candidate match can be calculated from the annotated power-delay curves at the inputs of the gate and the power-delay characteristics of the gate itself.

5 Post-Mapping Power Optimization

Traditionally, logic synthesis has been divided into two stages: technology independent optimization and technology mapping. Power optimization techniques during technology independent phase have been presented in previous sections. Recently, structural optimization techniques have been introduced as a new optimization step. In a multi-level network, it is in general possible to replace a signal (a node input or output) with an existing signal in the network. At the same time it is also possible to replace a signal with a combination of two or more sig-

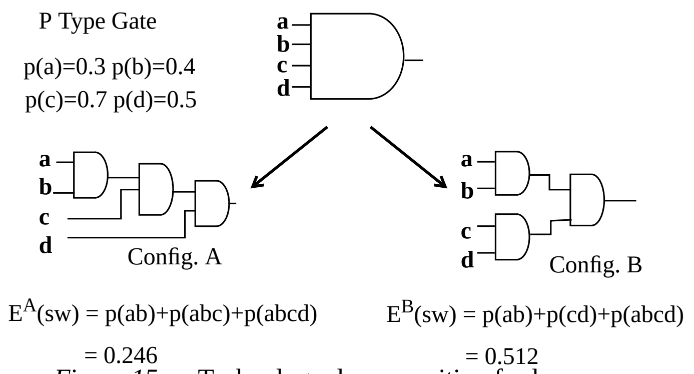

Figure 15. Technology decomposition for low power.

similar to Huffman's algorithm for constructing a binary tree with minimum average weighted path length) is optimal for dynamic CMOS circuits and produces very good results for static CMOS circuits. An example is shown in Figure 15 where the input signal with the highest switching activity (that is, signal *d*) is injected last in the decomposition tree in configuration A, thus yielding lower power dissipation for this configuration.

In general, the low power technology decomposition procedure reduces the total switching activity in the networks by 5% over the conventional balanced tree decomposition method.

4.2 CELL BINDING

The problem of minimizing the average power consumption during technology mapping is addressed in [66], [64] and [40]. The general principle is to hide nodes with high switching activity inside the gates where they drive smaller load capacitances (see Figure 16).

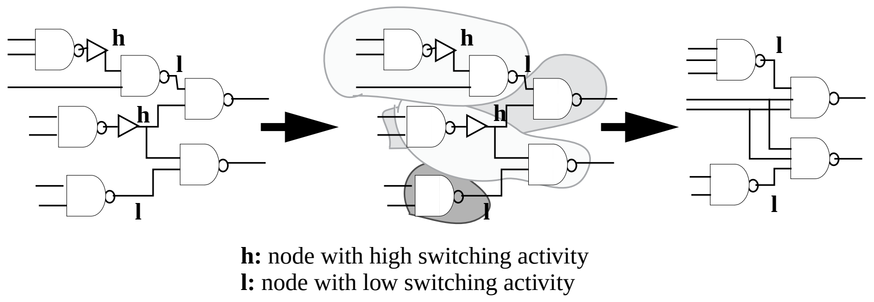
h
l
h
l
h
l
h
l
l
l
h: node with high switching activity
l: node with low switching activity

Figure 16. Technology mapping for low power.

mized.

While finding an irredundant cover, the cover $C(F)$ that minimizes the following cost function is generated:

$$\sum_{I \in C} PwrCost(I) + \sum_{dc \in DC} PwrCost(dc) \tag{21}$$

For the reduction step, the prime implicant is reduced such that after reduction, the cover $C(F)$ has minimum power cost given by equation 21.

4 Low Power Technology Mapping

Technology mapping is the problem of binding a set of logic equations (or a boolean network) to the gates in some target cell library. A successful and efficient solution to the minimum area mapping problem was suggested in [34] and implemented in programs such as DAGON and MIS. The idea is to reduce technology mapping to DAG covering and to approximate DAG covering by a sequence of tree coverings which can be performed optimally using dynamic programming. In this approach, the technology mapping process is divided into two phases where in the first phase, the Boolean network and library components are decomposed into basic gates and in the second phase, the Boolean network (subject graph) is covered using the library gates (pattern graphs) using a tree covering technique. In the following sections the application of these techniques for low power synthesis are discussed.

4.1 TECHNOLOGY DECOMPOSITION

Technology decomposition is the problem of converting a set of Boolean equations (or a Boolean network) to another set (or another network) consisting of only two-input NAND and inverter gates. It is difficult to come up with a NAND decomposed network which will lead to a minimum power implementation after technology mapping since gate loading and mapping information are unknown at this stage. Nevertheless, it has been observed that a decomposition scheme which minimizes the sum of the switching activities at the internal nodes of the network, is a good starting point for power-efficient technology mapping.

Given the switching activity value at each input of a complex node, a procedure for AND decomposition of the node is described in [66] which minimizes the total switching activity in the resulting two-input AND tree under a zero-delay model. The principle is to inject high switching activity inputs into the decomposition tree as late as possible. The decomposition procedure (which is by one of the prime implicants already in the partial cover, then the corresponding don't care point is in $DC_{PartialCover}$.

For incompletely specified multiple output functions that target pseudo NMOS PLAs, it is possible to obtain a lower power implementation by removing don't cares from product terms. A *pseudo prime implicant* is defined as follows:

Definition 8 *Given a Boolean function f and its don't care set, a pseudo-prime implicant is defined as an implicant p such that if it is possible to expand p in any direction, then the new points in the on-set of p will only include don't care points.*

Note that for a function with no don't care, the set of pseudo-prime implicants are identically the same as the set of prime implicants of the function. Also note that each pseudo-prime implicant P_p corresponds to a prime implicant P of the function where P_p and P contain the same on-set points of the function. The incompletely specified multiple output pseudo-NMOS problem is solved by using the minimum covering problem to select the set of pseudo-prime implicants which result in minimum total power cost. Once a minimum power solution is obtained, each pseudo-prime implicant is expanded in the cover after checking to make sure this expansion does not increase the total power. Note that this approach does not guarantee exact optimal solution. Another approach for solving the problem for incompletely specified multiple-output pseudo-NMOS PLA is to solve the exact covering problem using the prime implicants of the function and then reduce each prime implicant in the final cover if this reduction results in a decrease in total power. Special cases may also be considered during this step. For example theorem 2 states that if a don't care set is only included in one output function, then that don't care set may freely be included in the on-set.

For a heuristic solution, Espresso can be used with the following modification. Let $C(F)$ be a cover of the function F. If every implicant in $C(F)$ is used in the final implementation, the power cost is given by:

$$\sum_{I \in C(F)} PwrCost(I) \tag{19}$$

If an implicant is expanded to a prime implicant PI which covers implicants in $C_1 \subset C(F)$, then including PI in the cover will reduce the power cost by:

$$\sum_{I \in C_1} PwrCost(I) - PwrCost(PI) - \sum_{dc \in DC} PwrCost(dc) \tag{20}$$

where DC is the set of additional don't cares introduced in the output when PI is included in the cover. During the expansion step, each implicant is expanded to prime such that the reduction in power cost specified in equation 20 is maxiif the prime implicants are derived without enumerating minterms, there exist functions with an exponential number of prime implicants as a function of the number of implicants in a minimum cover. Also the minimum-cover problem itself is NP-complete.

Rudell [56] proposed an exact algorithm to solve the two-level minimization problem based on the motivation that the problems faced in reality do not have the worst case behavior. Also an exact solution can be used to indicate the quality of the heuristic methods. Heuristic methods such as ESPRESSO[6] were proposed to solve problems with large number of inputs and product terms. ESPRESSO uses an iterative improvement to achieve confidence in the optimality of the final result. Instead of generating all prime implicants and finding a minimum cost cover, implicants are expanded to prime implicants and then an irredundant cover is generated from the expanded prime implicants. This approach eliminates the basic problem of explicit generation of prime implicants. To avoid local minima, after the expansion and removal of covered implicants, the remaining implicants are maximally reduced while still maintaining a cover. Then the expansion process is repeated and the entire procedure is iterated until no improvement is obtained.

3.3.2 *Power Optimization*

Only prime implicants need to be considered for completely specified single output functions and incompletely specified multiple-output functions which target dynamic CMOS and pseudo-NMOS implementations. In these cases, standard Quine-McCluskey procedure is used to find the minimum power solution. In building the prime implicant table, each column representing a prime implicant is tagged with the power cost given by equation 11 or 14.

For an exact solution, the algorithm described in Espresso-Exact [56] is modified by using appropriate weight functions during the branch-and-bound process that is used to find a minimum cost cover. Instead of the number of prime implicants in the partial solution, the sum of the power costs of the prime implicants in the partial solution is used as the current cost. In case of incompletely specified single output function, power cost incurred at the output functions which is caused by including don’t cares in the cover has to be added. Therefore the power cost of a partial cover is given by:

$$\sum_{p \in PartialCover} PwrCost(p) + \sum_{d \in DC_{PartialCover}} V_{dd} \cdot I_{dc} \cdot sp_d^1 \tag{18}$$

where $DC_{partialCover}$ is the set of don’t cares which are included in the partial cover. $DC_{PartialCover}$ can be found by augmenting the prime implicant table by adding a row for each element in the *DC_set* of the function. If a row is covered

$$\frac{V_{dd}^2 \cdot f}{2} \cdot \left(\sum_{i=1}^{k} c_i \cdot E_{x_i} + c_p \cdot sp_p^0 \right) \tag{17}$$

Let P_1 be a product term that is contained in P, then PwrCost(P) $\leq$ PwrCost(P_1).

This means that only prime implicants need to be considered in search for the minimum power solution in PLA optimization.

3.3 TWO LEVEL FUNCTION MINIMIZATION

Traditionally two level logic minimization for PLA targets for minimum area. PLA is a regular structure and its area is proportional to (number of inputs + number of outputs) $\times$ (number of product terms). Therefore the problem of minimizing the PLA area is equivalent to the problem of finding the minimum number of product terms to implement the Boolean function. For power minimization, the objective is to minimize the power consumption at both the AND plane and the OR plane. In the following sections, the relationship between optimizing area and optimizing power for PLA minimization is shown and it is described how algorithms for area minimization are modified to minimize power consumption.

3.3.1 *Area Optimization*

Since the area of a PLA is proportional to the sum of inputs and outputs multiplied by the number of AND terms (or product terms), minimizing area is equivalent to minimizing the number of product terms. The PLA area is relatively independent of the implementation technology. Therefore the minimization algorithms that target minimum area work for different types of technologies. It is easy to see that if the cost of a product term is not larger than any product term which it contains, then a minimum cost solution exists for the two level minimization problem which consists only of prime implicants. For PLA area minimization, the cost of a product term is equal to the area for adding a row to the PLA and is constant for every product term, therefore the above condition is satisfied and only prime implicants need to be considered during the minimization process. The classic solution of finding a minimum cover for a function is the Quine-McCluskey algorithm which consists of the following steps:

1-Generate all of the prime implicants.

2-Form the prime implicant table.

3-Derive a minimum cover of this table.

For generating all prime implicants, some algorithms resort to explicit enumeration of the minterms which in worst case is exponential in complexity. Even sidering the power at the input, including don't cares at worst will not increase the power. Considering the reduction in power on the inputs, including don't cares will always result in reduction in power for a single output function.

The statement of theorem 2 does not in general hold for incompletely specified multiple output functions. This is because the power cost of a product term in the AND plane of a pseudo-NMOS PLA is independent of the load at its output. Consider a product term *p* and OR-terms (o_1,..., o_n) that *p* fans out to. Removing don't care points from *p* will result in an increase in the power cost of *p*. Note however that this increase is independent of the number of fanouts. Now removing don't care from *p* will in turn result in a decrease in the power cost of OR-plane terms (o_1,..., o_n) if the don't cares are true don't cares for more than one OR-terms and at the same time are not covered by other product terms. In general it is possible for the reduction in power at the OR plane to be larger than the increase in power of *p* and therefore reducing the overall power cost. The approach for dealing with multiple output functions is discussed in section 3.3.2.

The following theorem shows that power cost of a product term in a single output Boolean function is no more than any product term which it contains.

Theorem 3 *Let power cost (PwrCost) of a product term P for a single output Boolean function be given as:*

$$V_{dd} \cdot I_{dc} \cdot sp_p^0 \tag{16}$$

Let P_1 be a product term that is contained in P, then PwrCost(P) $\leq$ PwrCost(P_1).

3.2.2 *Dynamic PLA*

It is assumed that NOR gates in the AND and OR planes drive the same capacitive loading. In this case, the effect of using don't care in the logic cover will be the same as the case for pseudo-NMOS PLAs. Also note that for dynamic PLA implementations, this result also holds for multiple output functions. This is easily proved by noting that AND plane power for dynamic CMOS PLAs is a function of its load. This load is proportional to the number of OR-terms that this product terms fans out to. Removing don't cares from a product term *p* will result in even more increase in power as the number of fanouts of a p increases. This is in contrast with pseudo-NMOS circuits where the power in the AND plane is independent of its load. Equation 14 is used for computing the power cost of a product term for dynamic PLA. The final term in this equation ($2 * c_{clock}$) is a constant term and is dropped in the optimization.

Theorem 4 *Let Power cost (PwrCost) of a product term P be given as:*

minimization. The cases for pseudo-NMOS and dynamic PLAs are discussed separately.

3.2.1 *Pseudo-NMOS PLA*

Effect of using don't cares on the power cost of a pseudo-NMOS PLA is examined first. The first level NOR gate in a pseudo-NMOS PLA will draw direct current when the AND term evaluates to 0. Similarly the second level NOR gate will draw direct current when the gate output is 0 (i.e., the function output evaluates to 1). Including don't cares in the cover will reduce the probability of the AND term evaluating to 0 and hence reduces power consumption at the output of the AND term. At the same time, it will increase the probability of the output evaluating to 1 which increases the power consumption.

The example in Figure 13 shows how don't cares may be used to make power trade-offs at the output of the AND plane and OR plane. Assuming each variable has 0.5 signal probability and $V_{dd}I_{dc(AND)}=1$, the *OutPwr* cost of implicant P_1 (given by equation 11) is 0.75 while that of implicant P_{11} is 0.875. Therefore *OutPwr* cost is reduced when P_1 is used instead of P_{11}. A similar statement holds for P_2 and P_{21}. However, if P_1 and P_2 are used instead of P_{11} and P_{12}, the *OutPwr* cost at the output of the OR plane will be increased from 0.25 to 0.375.

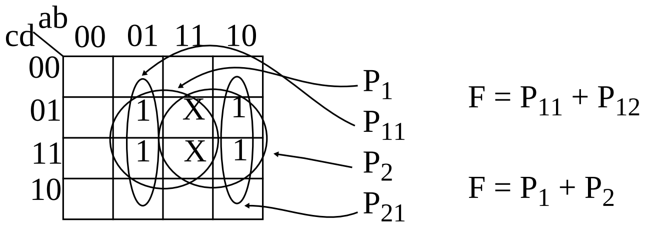

Figure 14. Including Don't Cares in the cover.

PLA is a regular structure and the transistor sizes for the OR plane and the AND plane are identical. Therefore $I_{dc(AND)}$ and $I_{dc(OR)}$ are the same. Based on this observation, the following theorem shows that including don't cares in the cover of a single output function, in the worst case, will keep the power of the PLA unchanged, while in most cases, it will reduce the power consumption.

Theorem 2 *Including don't cares in the logic cover of a single output Boolean function will not increase the power cost of the final implementation of a pseudo-NMOS PLA.*

Note also that when don't cares are included in a product term, in general the number of inputs to the product term is also reduced. This will in turn reduce the power due to the input literals. The preceding analysis shows that without conwhere c_p is the load seen at the output of the product term and is estimated as the number of gates in the OR-plane that p fans out to.

The total power cost due to a product term p is obtained by taking into account the power consumption on the inputs and clocks and is given by:

$$Pwr(p) = \frac{V_{dd}^2 \cdot f}{2} \cdot \left(\sum_{i=1}^{k} c_i \cdot E_i + c_p \cdot sp_p^0 + c_{clock} \cdot 2 \right) \tag{14}$$

where c_{clock} is the load capacitance of the pre-charge and evaluate transistors that the clock drives.

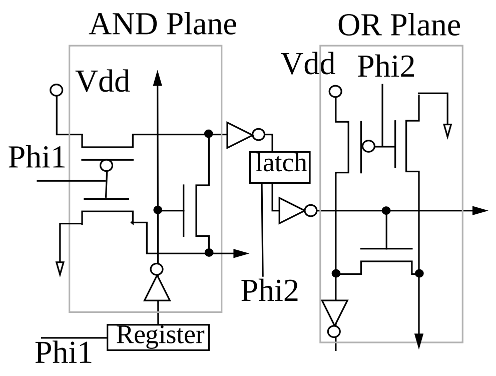

Figure 13. Dynamic PLA.

The power due to the OR plane of the PLA is measured using equations 11 and 13 presented for the product terms since both the input and output planes of the PLA are implemented using NOR gates.

3.2 PRIME IMPLICANTS

The two level logic minimization problem for low power is equivalent to finding a cover *C(F)* such that the following objective function is minimized:

$$\sum_{p \in C(F)} Pwr(p) + \sum_{q \in O(F)} OutPwr(q) \tag{15}$$

where *O(F)* is the set of gates in the OR plane.

For area minimization, it has been shown that it is sufficient to consider the set of prime implicants of a function based on the assumption that the area cost of a prime implicant is always lower than or equal to that of any implicant it contains. In the following, it is examined whether this assumption holds for power

The primary source of power consumption for a pseudo-NMOS NOR gate is the static power dissipation (Figure 12). When a pseudo-NMOS NOR gate evaluates to zero, both the PMOS and NMOS parts of the gate are conducting and there exists a direct current path. The charging and discharging energy is small compared with that dissipated by the direct current when the frequency of operation is not extremely large. Furthermore, the direct current I_{dc} is relatively constant irrespective of the number of NMOS transistors that are on. It is assumed here that static power dominates in pseudo-NMOS PLAs. The power cost at the output of a product term p is given by:

$$OutPwr(p) = V_{dd} \cdot I_{dc} \cdot sp_p^0 \tag{11}$$

where $sp^0{}_p$ is the probability that the product term p evaluates to 0. The total power cost of a product term which accounts for power consumption at its inputs and output is then given by:

$$Pwr(p) = \frac{V_{dd}^2 \cdot f}{2} \cdot \sum_{i=1}^{k} c_i \cdot E_i + V_{dd} \cdot I_{dc} \cdot sp_p^0 \tag{12}$$

where c_i gives the load due to product term p on input i .

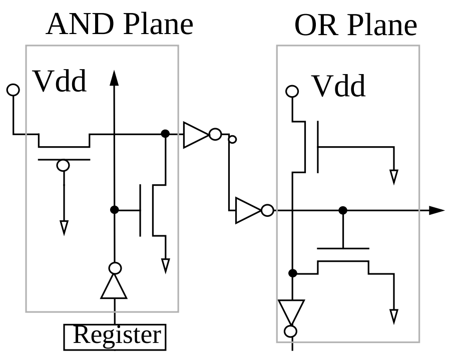

Figure 12. Pseudo-NMOS PLA.

In a dynamic PLA circuits (Figure 13), dynamic power consumption is the major source of power dissipation. The output of the product term is pre-charged to 1 and switches when it evaluates to 0. Therefore the power consumption at the output of a product term p is given by:

$$OutPwr(p) = \frac{V_{dd}^2 \cdot f}{2} \cdot c_p \cdot sp_p^0 \tag{13}$$

inputs to generate the product terms required by the defining logic functions. The other is the OR plane which combines the product terms to generate the output functions. A convenient measure of a PLAs size is the triplet (i, p, o) where i is the number of inputs, p is the number of product terms and o is the number of outputs. The number of potential transistors in the AND and OR planes is given by the expression *(2i + o) p*. Increasing *i* or *o* adds to the width of the AND plane or the OR plan, respectively. Increasing *p* adds to the height of both the AND and OR planes. A relative measure of the PLA size is then given by the calculation *(2i+o)p*. Since for a given set of Boolean functions, the number of inputs and outputs is constant, therefore the problem of optimizing a PLA for area is equivalent to minimizing the number of product terms necessary to implement the Boolean functions.

This section, addresses the problem of minimizing power consumption in two-level logic circuits implemented as PLAs. The loading information in PLAs are known for a specific technology/implementation and hence the switched capacitance can be minimized directly. In particular, it is shown how logic minimization for area is modified to obtain a minimum power solution.

3.1 POWER COST FUNCTION

High speed PLAs are built by transforming the SOP representation of a two level logic to the NOR-NOR structure with inverting inputs and outputs and implementing it with two NOR arrays (Figure 11). Two common types of implementing NOR arrays are pseudo-NMOS NOR gates and dynamic CMOS NOR gate.

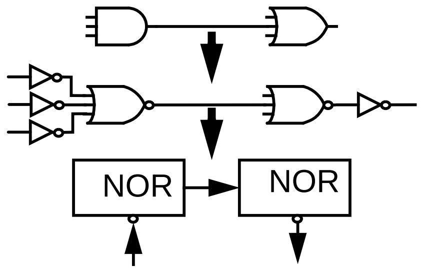

Figure 11. NOR-NOR PLA.

It is assumed here that the functions are implemented using a PLA that is driven using static CMOS drivers. This means that the cost function for an implicant in the final cover of the function assumes that the AND gate is implemented using either pseudo-NMOS or dynamic technology while the input power consumption follows the power cost model for static CMOS structures.

rectness, any power optimization and area optimization algorithm will produce the same results. In general it is possible to obtain a lower power solution for a trivial or non-cyclic function if this function is a power cyclic function.

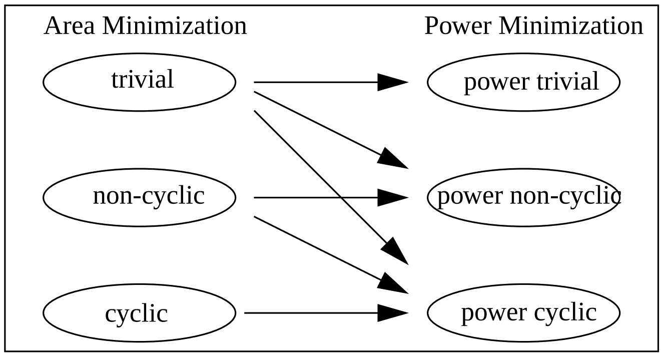

Figure 10. Function characterization for low power and low area optimizations.

An important observation is that power trivial and power non-cyclic functions all have the same minimum solutions for area and power. This means that for such functions a minimum area solution will also provide the minimum power solution. Therefore only power cyclic functions need to be optimized for power. If the power cyclic function is also a cyclic function, then the freedom to choose different subsets of the partially redundant primes of the function can potentially lead to power reductions. This means that it is possible to obtain power savings in a cyclic function even if no non-prime PPIs are included in the covering problem.

3 Power Optimization in PLAs

Minimizing the number of product terms and literals of a logic function is independent of the technology used to implement the function. This means that the same optimized implementation may be used for static and dynamic logic circuits. The cost functions used for measuring power however, are different in static and dynamic circuits [20]. This means that different strategies must be developed for different circuit types. The goal of this section is to provide function minimization techniques for reducing the power consumption in PLAs.

One important method of implementing logic functions in a regular, structured way is to use a Programmable Logic Array (PLA). Normally a PLA can realize several output functions concurrently. The PLA structure can be realized in either CMOS or NMOS technology. In either case, a PLA consists of two major subsections or planes. One is the AND plane, which requires double-rail

Solving the exact two-level function minimization problem for area consists of first generating a set of prime implicants that will be considered as candidate cubes of the function and then selecting a subset of these primes which result in minimum cost of the cover. The set of primes of a function can be partitioned into three classes. *Essential primes*, *partially redundant primes* and *totally redundant primes* [6]. Essential primes are defined as primes that cover at least one minterm that is covered by only one prime implicant. Totally redundant primes are then defined as the set of primes which are covered by the essential primes of the function. The remaining primes of the function are then defined as partially redundant primes. In order to study the complexity of the two-level minimization procedure, a function can be classified as *trivial*, *non-cyclic* or *cyclic* [56]. For a trivial function all primes of the function are essential therefore the two-level representation of the function includes all prime implicants of the function. A non-cyclic function does not have any partially redundant primes. Consequently the two level representation of the function again consists only of the essential primes of the function since all totally redundant primes can be dropped by including the set of all essential primes. The two-level representation of a cyclic function includes all essential primes and a subset of the partially redundant primes of the function. Note that for trivial and non-cyclic functions the minimum solution is obtained by first generating the set of all primes and then partitioning the primes into essential, partially redundant and totally redundant. It is not necessary to solve a minimum covering problem for trivial and non-cyclic functions. Cyclic functions do however require that a minimum covering problem be solved to minimize the cost function under consideration.

The classification of prime implicants and functions based on the make-up of their prime implicants can also be applied to functions when power is being minimized. *Essential PPIs*, *Totally Redundant PPIs* and *Partially Redundant PPIs* are defined similar to the area optimization case. Essential *PPIs* are *PPIs* that cover a minterm that is covered by only one *PPI*. Totally redundant *PPIs* are *PPIs* that are covered by essential *PPIs*. The remaining *PPIs* are defined as partially redundant *PPIs*. *Power trivial* functions are defined as functions where all *PPIs* are essential. *Power non-cyclic* functions are defined as functions that have no partially redundant PPI and *power cyclic* functions are functions which have partially redundant *PPIs*.

Figure 10 shows the relationship between function classifications for area and power. As shown in the figure, a trivial function can be power trivial, power non-cyclic or power cyclic. However a cyclic function can only be a power cyclic function. This relationship between function classification is due to non-prime implicants which are included in the set of *PPIs* of the function. Note that if a trivial function is also power trivial then in order to guarantee functional corwill then generate set *PP* of sets PP_n where each PP_n will contain *PPIs* with *n* literals.

```
1:  function Generate_PPI(F)
2:  F is a Boolean function with input set $V = (v_1,...v_N)$
3:   begin
4:       P = generateAllPrimes (F)
5:       PP = initializePP(P)
6:       for ( $i = 1 ; i < N ; i++$ ) do
7:           foreach ( implicant q with at least one predecessor in $PP_i$ ) do
8:               $q_j$ = findMinPowPredecessor($PP_i$)
9:               l = literalLoweredInQ(q, $q_j$)
10:              if ( $p(l) < (p(q_j) - p^2(q_j) )/(1+ p^2(q_j) )$ ) then
11:                  $PP_{i+1} = PP_{i+1} \cup q$
12:              endif
13:          endfor
14:      endfor
15:      return PP
16:  end
```

Figure 9. Computing the set of all PPIs.

Procedure *initializePP* is used to initialize *PP* by placing prime implicants with *n* literals in PP_n. The main loop in the procedure updates PP_{i+1} by adding all implicants that have a predecessor cube in PP_i and also satisfy the conditions for a PPI. In this loop *findMinPowPredecessor* returns the predecessor cube of *q* which has the smallest power cost. Procedure *literalLoweredInQ* will return the literal which was lowered in q_j to obtain *q*. Set *PP* returned by the procedure contains the set of all *PPIs* of the function.

Once the set of all *PPIs* of the function are generated, a minimal covering problem [56] will be used to select a set of *PPIs* which cover the function and have minimum power cost. The complexity of the minimum covering problem, grows exponentially as a function of the number of implicants which are being considered. This means that even though it might be possible to find an exact minimum area solution for a function, finding an exact minimum power solution might not be possible due to the increase in the number of implicants being considered. Assuming a uniform distribution for the signal probability of inputs to the function, it is shown in [33] that the upper bound on the E(PPI), expected number of PPIs introduced due to each prime implicant q (with any number of literals) of a function f with N inputs, is given by:

$$N \cdot \frac{2}{3} \cdot \left(2 - \left(\frac{2}{3}\right)^{N-1}\right). \tag{10}$$

Example 2:

Assume $p(a) = 0.9$, $p(b) = p(c) = 0.5$ and the following two-level implementations for function f:

$$F_1 = a.b + b.c$$

$$F_2 = a.b + a.b.c$$

Also assume $C_{OR} = C_{AND} = C_{IN} = 1$, Then:

$$Pwr(F_1) = 1.525\ V_{dd}^2\ f$$

$$Pwr(F_2) = 1.45\ V_{dd}^2\ f$$

■

This example shows that implementation F_2 provides a better power solution in spite of including a non-prime implicant. Even though the implementation for a non-prime implicant requires more literals and more transistors, overall, less power is consumed. This is because in static CMOS circuits a significant part of power is consumed when gate outputs change values. In this example, for implementation F_2, the reduction in power at the output of the second cube more than offsets the increase in the power due to including literal *a* therefore reducing the overall power consumption. It can be concluded from this example that area-power trade-offs can be made by including non-prime implicants in the set of candidate implicants when the covering problem is being set up. This observation motivates the definition for *Power Prime Implicants* of a function.

Definition 6 *An implicant q_i is a Power Prime Implicant (PPI) if:*

$$\forall q_j: \qquad q_i \subset q_j \Leftrightarrow Pwr(q_i) < Pwr(q_j)$$

By definition, all prime implicants (PI) of a function are also power prime implicants (PPI). In what follows the set of all PPIs of the function will also include the set of all prime implicants of the function. Non-prime PPIs are the set of PPIs which are not prime implicants.

Definition 7 *Predecessors cubes of an implicant q_i with n literals are defined as all implicants with n-1 literals obtained by raising a literal in q_i. The Successors cubes of q_i are defined as all implicants with n+1 literals obtained by lowering a literal in q_i.*

A simple approach for generating all PPIs of a function *f* with *N* inputs is to generate all n-cubes of a function for $n=1,...,N$ and compare the power for each n-cube *q* with that of all other PPIs containing *q*. This approach however quickly becomes intractable as the number of such cubes grows exponentially. An efficient algorithm for generating the set of all PPIs of a function [33] is presented in figure 9. In this procedure, efficiency is achieved by using the set of *PPIs* with *n* literals for generating and checking candidate *PPIs* with *n+1* literals. The input to this procedure is the set of all prime implicants of the function. The procedure measure the quality of heuristic approaches. Second, an exact procedure can be used to study the effectiveness of the optimization on the cost function which is being optimized. For example during logic synthesis, area and power are closely related. This means that a low area solution tends to provide a low power solution as well. An exact method in this case provides insight into the maximum power savings that can be achieved as compared to a minimum area solution.

The problem of exact two level logic minimization for minimum area is stated as follows:

Problem: *Given a Boolean function f with input set $V = (v_1,...v_N)$, find a two-level implementation of the function f such that the number of product terms and then the number of literals in the sum-of-products form is minimum.*

This problem is solved by first generating the set of all prime implicants of the function[6]. The minimum area solution is then obtained by solving an exact minimum covering formulation of the problem where a subset of the primes are selected to cover the function.

The problem of exact two-level function minimization for low power is stated as follows:

Problem: *Given a Boolean function f with input set $V = (v_1,...v_N)$ and signal probability p(i) for each input, find a two-level implementation of the function f such that the power as given in equation 9 is minimum.*

The main difficulty in generating an exact minimum power solution is that compared to a minimum area solution which only requires prime implicants, more implicants need to be considered while solving the covering problem. In the next section Power Prime Implicants (PPIs) are introduced and discussed.

In what follows, it is assumed that the value for $C_{AND} = C_{IN}$. This is a valid assumption since the functions being optimized are internal nodes of a Boolean network. Also the following notation will be used.

Given q_i, an implicant of function f, $q_i^{l_1, \ldots, l_k}$ represents the implicant generated by lowering literals $l_1,....l_k$ in implicant q_i. An *n-cube* is defined as an implicant with n literals.

2.2.3.B. *Power Prime Implicants*

It can easily be shown that only prime implicants need to be considered while minimizing a function for minimum area. This can be proved by noting that any solution that contains a non-prime implicant *q*, will be improved by making *q* prime and therefore reducing the area of the solution. This argument cannot however be applied when implicants are being considered for a minimum power solution. The following example illustrates this observation:

```
1:   function node_power_optimize(f, dc)
2:   f is the function and dc is the don't care of node;
3:   begin
4:       $f_{esp}$ = espresso(f , dc).
5:       V = find_k_minimal_switching_activity_var_sup(f, dc).
6:       foreach $v \in V$ do
7:           $f_{tmp}$ = node_fanin_optimize(f, dc, v)
8:           if( power_cost($f_{tmp}$) < power_cost($f_{esp}$) ) then
9:               $f_{esp} = f_{tmp}$
10:          endif
11:      endfor
12:      return $f_{new}$.
13:  end
```

Figure 7. Node optimization for power.

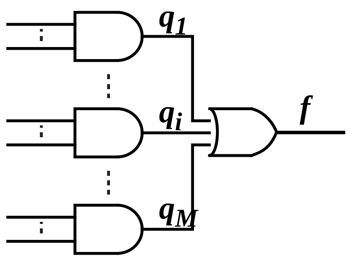

Figure 8. SOP form implementation of function *f*.

$$Pwr = \frac{V_{dd}^2 \cdot f}{2} \cdot C_{OR} \cdot E(f) + \sum_{q_i \in Q} Pwr(q_i) \tag{9}$$

where C_{OR} is the load seen by the output of the OR gate, C_{AND} is the load seen by the output of the AND gate and C_{IN} is the load seen by the inputs. *E(i)* gives the switching activity of gate *i*. Note that the power consumption of any implementation of the circuit (NOR-NOR, NAND-NAND, etc.) can be estimated using equation 9 since a function *f* and its complement will exhibit the same switching activity.

2.2.3.A. *Minimization Algorithms*

Exact methods cannot be applied to many large functions. Therefore an exact optimization procedure cannot effectively be used in real circuits. However an exact procedure is needed for two reasons. First an exact solution can be used to

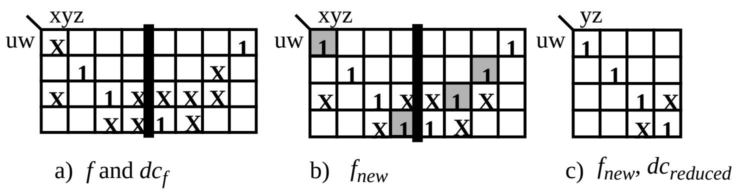
a) f and dc_f b) f_{new} c) f_{new}, $dc_{reduced}$

Figure 5. Using the reduced don't care.

```
1: function node_fanin_optimize(f, dc, v)
2: f is the function and dc is the don't care of node, v is a variable support;
3: begin
4:     p= cube representing eliminated variables
5:     $f_{reduced}$ = function_elim_variables(f, p)
6:     $dc_{reduced} = C_p(dc)$
7:     $f_{new}$ = espresso($f_{reduced}$, $dc_{reduced}$)
8:     return $f_{new}$
9: end
```

Figure 6. Node minimization using variable support *v*.

sible, the power estimate for the node implementation of the *k* lowest cost variable supports are compared where *k* is a user defined parameter. The node power cost function in the factored form also takes into account the output load and switching activity. Hence by selecting the lowest power cost implementation, a solution which minimizes all factors contributing to the network power consumption is selected. Given a node function *f* and its don't care *dc*, the procedure in figure 7 is used to select the lowest power cost implementation of the function.

2.2.3 *Exact Two-level Optimization in CMOS Circuits*

Given a Boolean function *f* with product terms $Q=(q_1,...q_M)$ and input set $V = (v_1,...v_N)$. The following model is used to estimate the power consumption of the two-level implementation of function *f*.

The power consumption due to each product term q_i is given by:

$$Pwr(q_i) = \frac{V_{dd}^2 \cdot f}{2} \cdot \left((C_{AND} \cdot E(q_i)) + \sum_{l_j \in lit(q_{i_i})} (C_{IN} \cdot E(l_j)) \right) \quad (8)$$

The power consumption of the function is then estimated by:

It is also shown in *[45]* that the reduced off-set of a cube is a unate function and therefore has a unique minimal representation in the SOP form. A procedure is presented in *[45]* for computing the reduced off-set of each cube in the function where reduced off-sets are computed without computing the complete off-set of the function. In [29] a technique is presented for computing the set of all minimal literal supports using the reduced offsets of the function.

Once the set of minimal variable supports for a function is computed, a decision has to be made as to which set of variables to use for implementing the function. A simple cost function for minimizing power consumption is to count the number of variables in the variable support. The drawback with this cost function is that it does not consider the switching activity of fanin variables that constitute the support variables. A better cost function is to choose a variable support where the sum of the switching activity for all the variables in the support is minimum. This procedure is referred to as the "*minimal switching activity support*" procedure. Once the new variable support for a node is determined, the new function of the node is computed by dropping variables not in the support.

When a variable support is selected for the function, a part of the don't care is assigned to eliminate the variables not in the selected support of the node. This operation results in a new function f_{new}. However a subset of the don't care can still be used to minimize the cover of f_{new}. This subset of the don't care is called reduced don't care $dc_{reduced}$.

Theorem 1 *Given a cube v representing the variables removed from the on-set of a function f and dc, don't care for function f,* $dc_{reduced}$ *the reduced don't care for f, as given below, is the maximal set of don't care that can be used to optimize f without including variables from v in f.*

$$dc_{reduced} = dc_v \cdot dc_{\bar{v}}$$

($\bar{v}$ is the bit-wise complement of *v).*

Figure 5.a shows the on-set and don't care for a function *f*. Figure 5.b shows the don't care assignment which is used to eliminate variable *x* from the support to obtain f_{new} and figure 5.c shows the reduced don't care for function *f* after variable *x* is eliminated from the k-map. Using reduced don't care for this node, one product term is removed from the on-set of the function *f*. Given a function *f*, its don't care set *dc* and a variable support *v*, the procedure in figure 6 is used to optimize the power consumption of the function.

The given procedure will provide a low area implementation which has the lowest sum of switching activities on the immediate fanins of the node. It is however possible for a variable support with a higher switching activity support cost to have a smaller factored form and hence have a lower power. In order to select a variable support which also reduces the node's power estimate as much as pos-

This section first presents a more efficient method for computing the set of minimal literal and variable supports of the nodes in the network. The set of minimal supports gives the flexibility in implementing the node function using different sets of variables. Techniques for selecting a node support which will potentially lead to maximal reduction in the node input power are then discussed.

Given an incompletely specified function *ff*, it is often possible to implement *ff* using different sets of literals. For example let $F = a.b$ and $D_F = a \oplus b$. This function can be simplified to $F = a$ or $F = b$. Then set $\{\{a\}, \{b\}\}$ is the set of all minimal literal supports for node *F*. The problem of finding the minimal literal support of a function is stated as follows.

Problem: *Given* $C^1=\{C^1{}_0, C^1{}_1,...., C^1{}_g\}$*, the cover of the on-set and* $C^0=\{C^0{}_0, C^0{}_1,...., C^0{}_h\}$*, the cover of the off-set of a function* $F(x_1, x_2,.... x_n) \in R^n$*, find the set of all minimal literal supports of the function F.*

Once the set of all minimal literal supports of a function has been computed, a literal support is selected to implement the function. This problem is stated as follows:

Problem: *Given* $C^1=\{C^1{}_0, C^1{}_1,...., C^1{}_g\}$*, the cover of the on-set for a function* $F(x_1, x_2,.... x_n) \in R^n$*, and a set of minimal literal support given as a set of literals* $MLS_F=\{lit_1, lit_2,..., lit_k\}$*, find the minimal irredundant form of the function F.*

The solution to this problem is obtained by raising all literals in on-set of F which are not a member of set $\mathrm{MLS_F}$ [25]. The method described in [25] for computing the set of all minimal literal supports requires that a cover of the on-set and off-set of the function be computed. The off-set has to be computed by complementing the union of on-set and don’t care of the function. This operation is in general computation expensive and the resulting off-set might have an exponential size. An example of this function is the Achilles Heel function which has *n* terms in the cover of on-set and 3^n terms in the cover of off-set. Therefore it is desirable to compute the set of all minimal literal supports without computing the off-set of the function. In this section a method is presented for computing the set of all minimal literal supports of a function without computing the off-set by using the ideas behind reduced off-sets.

Reduced off-sets are introduced by observing that some minterms of the on-set or don’t care cannot be used to expand a cube of the on-set. Assume $p=a.b$ and the complete off-set is $a \oplus b$. Then the reduced off-set of *p* is $(a + b)$ which is all that is needed to expand *p*.

Definition 5 *[45] Given a cube p of a function f,* R_p*, the reduced off-set of function f with respect to cube p is obtained by including all minterms of the on-set that cannot be used to expand p, in the off-set of the function.*

high switching activity. The second drawback is that this operation may prove to be expensive when the number of nodes in the transitive fanin cones of *f* and *m* are large. An alternative approach for including *SDC* of nodes that do not share the same immediate support as *f* is proposed here by observing that given a Boolean network with *n* primary inputs and *m* internal nodes, the range of B^n, space of primary inputs onto B^{n+m}, the space of all nodes in the network, gives the set of all satisfiable conditions in the network.

Assume an internal node *f* with fanins $\{i_1,..., i_l\}$ *is* being optimized while considering possible substitution of nodes $\{n_1,..., n_k\}$ into the function of *f*. The complement of the range of the space of primary inputs by function $F=\{i_1,..., i_l, n_1,..., n_k\}$ gives all unsatisfiable conditions in the space of function *F* which is used as the *SDC* set while optimizing node *f*. Using this technique, it is no longer necessary to include the *SDC* due to nodes which are not good candidates for substitution into the function of node *f*.

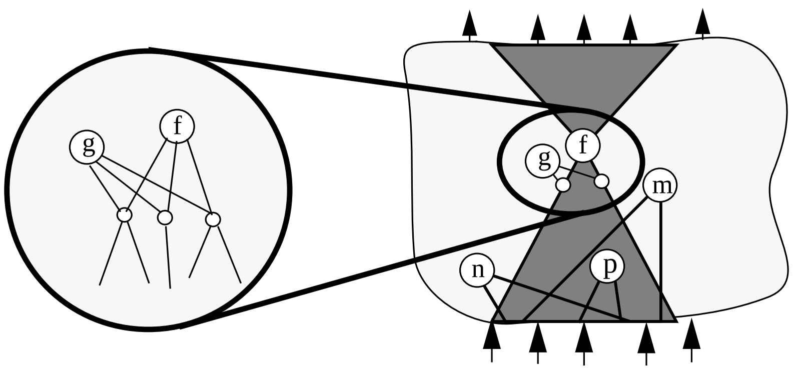

Figure 4. Candidate nodes for SDC computation.

The procedures presented in this section compute a set of local don’t cares for the function that is being optimized. This local don’t care guarantees that global power is not degraded while the node is optimized and also allows for expressing the node function using a new variable set which may potentially result in a lower power consumption. In the next section techniques using minimal literal and variable supports are presented to optimize the local function of a node for low power.

2.2.2 *Node Optimization Using Minimal Variable Supports*

The goal of node optimization is to minimize the power contribution of the node to the overall power consumption of the network. This requires that the combination of the node power at the input and output as well the estimate for the internal power of the node be minimized. This section presents a method for minimizing the power of the node by reducing load on high activity inputs of the network.

immediate support is a subset of the immediate support of the node being optimized is considered.

It has been shown [48] that using *SDC* due to any node in the network for simplifying the function of a node *f* does not result in changing the global function of *f* or any other node in the network. This means that using *SDC* does not change the signal probability or switching activity of any of the nodes in the network. Therefore *SDC* may freely be used to optimize the function of nodes without concern that switching activities may increase. Successful use of *SDC* may however result in using a new variable *v* in the function of the node being optimized. This results in a load increase at the output of node n_v generating variable *v.* If the switching activity of the new variable is high, then increasing load on this variable may result in an unexpected increase in the power consumption of the network. It is therefore important to take into account the switching activity of nodes that are being considered for possible substitution in the function of node that is being optimized.

The approach presented in [57] for using *SDC* consists of identifying a set of nodes that with high probability may be substituted in node *f* being optimized. These nodes are identified as all nodes whose immediate support is a subset of the immediate support of node f. The *SDC* due to each of these candidate nodes is then included in the don’t care of *f.* For power optimization, the following approach is used for finding the set of candidate nodes whose *SDC* will be included in the don’t care of a node *f*. Figure 4 shows the transitive fanin and fanout nodes of node *f* in a Boolean network. *SDC* of nodes in the transitive fanout of node *f* cannot be included in the don’t care of *f*. All other nodes in the network may be substituted into *f* if their primary input support is a subset of the primary input support for node *f*. Nodes *g*, *m*, *n* and *p* show such candidate nodes. In order to select a set of candidate nodes, first all nodes that are not in the transitive fanout of *f* and whose primary input support is a subset of the primary input support of *f* are identified. Among these nodes, nodes whose switching activity is below a user defined threshold value is selected.

The SDC due to a node *g* (see figure 4) whose immediate support is a subset of the immediate support for node *f* is easily included as $g \oplus F(g)$. In order to include the *SDC* due to nodes that do not share the same immediate support as *f* (node *m* in figure 4), it is necessary to include the *SDC* for all nodes that are in the transitive fanin cone of *f* and transitive fanin cone of *m*. This is necessary since successful substitution of *m* into *f* requires that the unsatisfiable conditions relating the values at the output of node *m* and all immediate fanins of *f* be known. The first problem with this approach is that including the *SDC* for all these nodes will result in also considering them for possible substitution into *f* while *f* is being optimized and this may not be desirable if these nodes have a *will not affect the global function of node f.*

Lemma 2 *Using a minterm v_i in don't care regions $R_{g,f}(1,1)$ and $R_{g,f}(0,0)$ while optimizing node g will result in bringing this minterm from the off-set to the on-set of function f.*

Lemma 3 *Using a minterm v_i in don't care regions $R_{g,f}(0,1)$ and $R_{g,f}(1,0)$ while optimizing node g will result in bringing this minterm from the on-set to the off-set of function f.*

For each region in ODC_g, the change in the function of *f* as minterm v_i in region *i* is included in or excluded from the on-set of *g* is well defined. This means that while optimizing node *g*, the effect of changes in global function of node *f* is exactly known using the information on the don't care regions for node *g*. Therefore the effect of changing the function of *g* on the signal probability and therefore the switching activity of node *f* can exactly be measured.

Don't care regions are then used in [29] to define *Propagated Power Relevant Observability Don't Care* conditions for a node *f*. The *Propagated Power Relevant Observability Don't Care* for node f is defined as a subset of the observability don't care conditions for f that is used to compute the observability don't care conditions for node g while guaranteeing that any changes in the function of g does not increase the switching activity of node f. *Propagated Power Relevant Observability Don't Cares* are then used to compute *Power Relevant Observability Don't Cares* for node g. *Power Relevant Observability Don't Care* is defined as the observability don't care conditions for g that guarantees any changes in the function of g does not increase the switching activity of its fanout nodes. A complete algorithm for optimizing a Boolean network for power using power relevant observability don't cares is presented in [29].

In a Boolean network, some combinations for the values of the internal nodes are not possible no matter what input vector is applied at the primary inputs of the circuit. If a network has *n* primary input nodes and *m* internal nodes then satisfiability don't care conditions (*SDC*) for a network contain all impossible combinations in the space of B^{n+m}. The contribution of each node in the network to *SDC* of network is given in definition 2. In this sense *SDC* due to a node *g* is defined as $g \oplus F(g)$ where *g* is the variable at the output of node *g* and *F(g)* is the function of node *g* in terms of its immediate fanins. Satisfiability don't cares are usually used to substitute a new variable into a function if this substitution results in a lower cost implementation.

While optimizing a function *f* in the network, a subset of *SDC* for nodes that with high probability may be substituted into *f* are usually used. In [57] a method for selecting a subset of *SDC* is presented where only *SDC* of nodes whose plete *ODC*, the *ODC* at other nodes in the network will potentially change. Therefore the *ODC* for each node has to be recalculated after each optimization step. At the same time, the size of the complete *ODC* can become extremely large. Therefore subsets of *ODC* have to be used instead of the complete *ODC* of each node. *RESTRICT* was the first *ODC* filter introduced in [28]. This filter removed any cube in the *ODC* of a node y_i which had a literal corresponding to a node in its transitive fanout. Although this filter and a number of other filters made the *ODC*s smaller, *ODC*s still had to be recomputed after each node simplification. Compatible set of permissible functions (*CSPF*), introduced in [49] allows for simultaneous optimization of all nodes in a network. In [57] the concept of *CSPFs* which is only defined for NOR gates, is extended to complex nodes of a general multi-level network and is called Compatible *ODC*s (*CODCs*). *CODCs* are used to simultaneously minimize the function of each node in the network. Even though by using *CODCs* some of the information contained in each of the full *ODCs* is lost, the time complexity of using the full *ODC* makes it impossible to use them for any practical size problem.

The compatible don't care computed here is freely used while minimizing the function of nodes in a Boolean network guaranteeing that the global function of circuit outputs will only change within their specified external don't care set. The change in the global function of transitive fanouts of node *n* as *n* is optimized, is not of concern when area is being minimized since the change in the function of each node will be within the observability don't care calculated for that node. However as mentioned before, this observation does not hold for power minimization. For example if by modifying the function of an internal node, the signal probability of a fanout node is changed from 0.1 to 0.2 a 78% increase in the power consumption of the fanout node can be expected.

In order to study the effect of using observability don't cares on the switching activity of other nodes in the network, the notion of don't care regions for a node f and its fanin node g is introduced in [29].

Definition 4 *Given a node g and its fanout nodes $F=\{f_1,...f_k\}$, the don't care regions of g with respect to F are denoted as $R_{g,F}(\alpha,\beta)$. This don't care region specifies all global conditions where g evaluates to α ($\alpha=\{0,1\}$). The second entry is a k bit vector where each bit takes values from the set $\beta=\{0,1,-\}$. Bit i of this vector specifies whether for points in this region f_i evaluates to the same value as g ($\beta_i=0$), an opposite value than g ($\beta_i=1$) or whether f_i is independent of g ($\beta_i=-$).*

The following lemmas give properties of don't care regions which are used to study the effect of changing the global function of node *g* on the global function of its fanout node *f*.

Lemma 1 *Using the minterms in $R_{g,f}(1,-)$ and $R_{g,f}(0,-)$ while optimizing node g*

Definition 1 *The external don't care set for each output z_i of the network is all input combinations that either do not occur or the value of z_i for that input combination is not important.*

Definition 2 *If y_i is the variable at an intermediate node and f_i its logic function, then $y_i = f_i$. Therefore, $y_i \neq f_i$.represents conditions that are not satisfiable. The expression $SDC = \sum(y_i \oplus f_i)$ for all nodes in the network is called the Satisfiability Don't Care set (SDC).*

Definition 3 *The observability don't cares (ODCs) at each intermediate node y_0 of a multi-level network are conditions under which y_0 can be either 1 or 0 while the functions generated at each primary output remain unchanged. If $z = (z_1, \ldots, z_l)$ gives the set of circuit outputs, then the complete ODC at node y_0 is:*

$$ODC_0 = \langle m \in B^n | z_{y_0}(m) \equiv z_{\bar{y}_0}(m) \rangle = \prod_{i=1}^{l} \overline{\frac{\partial z_i}{\partial y_0}} \quad (6)$$

The complete don't care set for a node *n* is found by first computing the *ODC* as a function of the primary inputs of the circuit. The external don't care which is also expressed in terms of the primary inputs is then added to *ODC*. Image projection techniques [18] are then used to find the *ODC* plus *EDC* of the node in terms of the immediate fanins of the node. Finally a subset of *SDC* for nodes which can be substituted into *n* with high probability, is added to this local don't care. In general computing the *ODC* for a node is the most complex part of this computation.

In [19] a method is described for computing the complete set of observability conditions for each node in the network where the *ODC* at each node is computed as a function of the *ODC* for its fanout edges. In this procedure the *ODC* of the node with respect to each primary output is computed separately. A different technique for computing the complete don't care set is presented in [57] which takes advantage of observability relations [12] at the primary outputs of the network. The given algorithm computes the complete *ODC* for each node in a multi-level combinational network. For tree networks, the following equation is used to compute the maximal set of *ODC*s at the output of a node *g*:

$$ODC_g = ODC_f + ODC_g^f \quad (7)$$

where $ODC_g^f = \overline{\frac{\partial f}{\partial g}}$.

The complete *ODC* cannot however be used in synthesis for any real size problem. This is because once the function of a node is minimized using its com-

ESPRESSO [6] presents a heuristic approach where novel techniques are used to efficiently produce good area solutions while ESPRESSO EXACT [56] presents an exact method for solving the minimum area solution. In [19] a method is presented for computing the complete set of observability conditions for each node in the network where the observability don't care (*ODC*) at each node is computed as a function of the *ODC* for its fanout edges.

This section addresses the problem of Boolean function minimization for low power using a don't care set. The problem of power optimization during logic synthesis has been addressed in a number of recent publications. The don't care set computed for area optimization [57] is used in [60] to optimize the local function of nodes for power. This work does not however take into account the effect of changes in the function of internal nodes on the power consumption of other nodes in the network. The idea of power relevant don't cares was first introduced in [29] where an efficient technique for computing power relevant don't cares was presented. The computed power relevant don't care guarantees that any changes in the local function of a node does not result in increasing the switching activity of other nodes in the network beyond their value *when these nodes were optimized*. The notion of minimal variable supports is used in [29] to optimize the local function of nodes for power.

The analysis on power relevant don't cares [29] is used in [43] to compute a re-synthesis potential for nodes in a technology mapped network. This re-synthesis potential represents the estimated effect of a change in the local function of a node on the power consumption of its transitive fanout nodes. The method presented in [43] also takes into account changes in power consumption due to variations in hazardous transitions in the network after an internal node is re-synthesized. Under a simplifying assumption (that is, assuming a signal probability of 0.5 for all primary inputs) an efficient technique is presented for computing this re-synthesis potential.

2.2.1 *Computing Don't Cares*

In logic synthesis, the concept of don't cares is used to represent the available flexibility in implementing Boolean functions. Don't care conditions for a function specify part of the Boolean space where the function can evaluate to one or zero. Three sources of don't cares are external don't cares (*EDC*), observability don't cares (*ODC*) and satisfiability don't care (*SDC*). It has been shown that if a node is minimized using all three types of don't cares, then all connections to and inside the node are irredundant. This means that *EDC*, *ODC* and *SDC* provide a complete set of don't care during optimization [5].

$$\left(\sum_{n_j \in fanins(n_i)} E(n_j) \cdot LF(n_j, n_i)\right) \cdot \left(\sum_{n_k \in fanouts(n_i)} LF(n_i, n_k) - 1\right) - \left(E(n_i) \cdot \sum_{n_k \in fanouts(n_i)} LF(n_i, n_k)\right) \tag{5}$$

The first term in this equation gives an estimate of the power added to the network by duplicating the node function. The second term accounts for the power at the output of the node which is removed from the network.

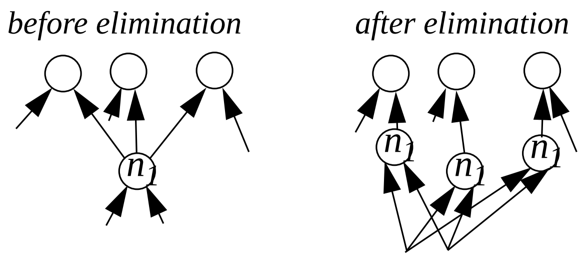

Figure 3. Node elimination in the network.

2.2 BOOLEAN OPTIMIZATION TECHNIQUES

Logic synthesis algorithms use the flexibility provided by don't care conditions to more effectively manipulate a Boolean network. Current techniques for computing and using these don't care conditions allow for correct functional operation of the network by guaranteeing that as each function in the network is being optimized using its don't care conditions, other functions in the network are changed only within their don't care sets.

Power consumption in a Boolean network is proportional to the switching activity of the nodes in the network. Switching activity of a node is a function of the signal probability of the function and therefore dependent on the global function of the node. This means that as a node is being optimized using its don't care set, the switching activity of the node function as well as the switching activity of other nodes in the network is also being changed. This clearly shows the need for new methods to analyze the effect of using don't care sets on the switching activity of nodes in the network.

Function minimization using a don't care set is an important part of logic synthesis for minimum area. The problem of don't care computations and Boolean function minimization has been addressed by many researchers in the past.

After the substitution, only the input plane for F_1 is changed in the network and the power at the input of F_1 is increased from 0.54 to 0.68 even though literal count is decreased.

■

Once it has been determined that a node n_j can be substituted into a node n_i, the power cost function of n_i for both SOP implementations of n_i before and after substituting n_j is computed. Substitution is performed if the power cost of n_i is reduced after substitution.

2.1.4 *Selective Collapse*

Selective collapse is the process of selectively eliminating nodes in a network in order to reduce the area cost of the network. Selective collapse is performed on an initial Boolean network to provide a better starting point for the extraction procedures. This is done since the initial network might have factors identified which are not good candidates for extraction. During area optimization, two value functions are used to decide if a node should be eliminated by collapsing it into its fanout nodes. One is the sum-of-products value of the node. This value is the reduction in the number of literals in the sum-of-products form of the network if the node is collapsed into all its fanout nodes.

The second value function is the factored form value. This value gives the reduction in the number of literals in the factored form of the network if the node is collapsed into its fanout nodes. The area value in the factored form of a node is defined as follows[56]:

$$\left(lit(n_i) \cdot \sum_{n_k \in \ fanouts(n_i)} LF(n_i, n_k) \right) - \left(\sum_{n_k \in \ fanouts(n_i)} LF(n_i, n_k) \right) - lit(n_i) \quad (4)$$

where *lit(n_i)* is the number of literals in the factored form of the node n_i and *LF(n_i, n_k)* denotes the number of times variable n_i appears in the factored form of node n_k. The first term in this equation accounts for the duplication of the literals in the factored expression of n_i once it is collapsed into its fanouts. The second term is the decrease in the number of literals by removing the literals corresponding to n_i from its fanout nodes. The last term account for the removal of n_i from the network where *lit(n_i)* literals are removed from the network.

The power value for a node is presented in [31] and is defined as follows:

transformed into $F=Q.D+R$ where D is the divisor, Q is the quotient and R is the remainder of dividing F by D. The procedure guarantees that the resulting factorized form of F is maximally factorized. The procedure *DIVISOR* passed to the function is used to find a candidate divisor for the function. By changing this procedure, trade-offs in terms of speed and quality of results can be made. Quotient Q is computed by performing weak division on F and D.

The problem of factorization for low power is stated as follows: *Given a sum-of-products expression for function F, find a maximally factored expression where the weighted sum of the literals by their switching activity is minimized.*

In [31], a DIVISOR procedure is presented that guarantees that at each step of the factorization process the resulting factorized form has minimum sum of literals weighted by their switching activities.

2.1.3 *Substitution*

Substitution of a function G into F is the process of re-expressing F in terms of variable G and the original inputs of the function *F*[73]. Substitution can be performed using algebraic or Boolean division algorithms. Even though Boolean division in general yields better results, algebraic division is a much faster heuristic which produces comparable results. If algebraic division is used for substituting variable G into function F and the quotient of the result is not 0, then it can be concluded that G was a sub-expression of function F. In fact if a non-trivial expression G (a function other than buffers or wires) can be substituted in function F using an algebraic division procedure (i.e. weak division), then the area value as defined in equation 2 is always greater than zero. The important consequence of this is that if a function G can be substituted into a function F, then it is guaranteed that this substitution will result in a decrease in the number of literals of the network in the SOP form.

Substitution is also used to minimize the network power consumption. However substitution does not always guarantee a reduction in the power cost of the network. This means that an operation which decreases the number of literals in the network can potentially increase the power cost of the network in the SOP form.

Example 1:

Assume $F_1 = a\ b\ c$ and $F_2 = a\ b$

also $p(a) = p(b) = p(c) = 0.9, p(F_2) = 0.5$

Note that $p(F_2) \neq p(a)*p(b)$. The cause for this is spatial dependence between inputs a and b[46].

F_1 can be expressed in terms of F_2 as follows:

$$F_1 = F_2\ c$$

that is inserted in the network. The last term accounts for the power at the output of the new cubes inserted in the SOP representation of function *f*.

$$\left((R-1)\cdot\sum_{(i=1)}^{M} E(v_i)\cdot LS(v_i, D)\right)+\left((P-1)\cdot\sum_{(i=1)}^{R}\sum_{(j=1)}^{M} E(v_j)\cdot LS(v_j, q_i)\right)$$
$$+\left(\sum_{(i=1)}^{P}\sum_{(j=1)}^{R} E(d_i q_j)\right)-(R\cdot E(D)) \qquad (3)$$
$$-\left(\sum_{(i=1)}^{P} E(d_i)\right)-\left(\sum_{(j=1)}^{R} E(Dq_j)\right)$$

where *LS(v, q)* denotes the number of times variable *v* appears in the expression for *q*. The procedure for performing low power kernel extraction for a Boolean network first computes the set of all kernels for all the nodes in the network using the procedure described in [56]. The sub-expression that has maximum power value is then extracted

Power tracks well with the area of a network. Therefore it is desired that power optimization techniques do not degrade the area quality of the solution while optimizing the network for power. It is also shown in [31] that extractions performed using equation 3 will not increase the literal count in the network. This is an important property since if the area of the circuit is increased a lot during power extraction, then the power due to the increased area will more than offset the power gain and no power gain will be obtained. Similar techniques are also presented in [31] for performing common cube extraction.

2.1.2 *Factorization*

Factorization is the process of deriving a factored form from a sum-of-products form of a function. For example if *F=a.b+a.c+b.c* then one possible factorization of *F* is *a.(b+c)+b.c*. In most logic synthesis systems Boolean functions are internally stored in the sum-of-products form. The area of a Boolean network however is more accurately estimated by the number of literals in the factored form of the network. This means that an efficient factoring algorithm is needed in order to guide the optimization problems. Exact methods for computing the best factorized form of a function have been presented in the past. However these algorithms are too complex to be used since the factorization algorithms are used to guide the optimization procedure and have to be performed numerous times.

A heuristic factoring algorithm is described in [73]. In this algorithm a recursive procedure is used to find a factored form of a given sum-of-products representation. At each step of the recursion the function *F* passed to the procedure is network. During algebraic operations, the value of a divisor is measured in terms of the reduction in the number of literals that is obtained by using that divisor.

2.1.1 *Common Sub-expression Extraction*

Algebraic extraction uses the concepts of common kernel and cube extraction to introduce nodes into a network in order to optimize the area cost of a network. These techniques use the SOP cost of nodes to minimize the total number of literals in the SOP form of the network. Algebraic extraction techniques utilizing kernels and cubes are used extensively and result in significant reduction in the area of the mapped network [56].

The problem of power optimization during extraction is stated as follows:

Problem: *Given a boolean network, find a set of nodes to introduce in the network such that the power cost of the network in the SOP form is minimized.*

Kernel and cube extractions can be used to minimize the network power consumption. In the following each method is presented separately [55] [31]. In the following, we summarize the work presented in [31].

Consider a multiple-output Boolean function $F=\{f_1,..,f_L\}$ with cubes $(c_1,...,c_N)$ and input set $(v_1,...v_M)$. Let $D=(d_1 +.. + d_P)$ represent a kernel of function f used as a divisor. Also assume $Q=\{q_1,..q_R\}$ is the set of co-kernels for kernel D in functions $\{f_1, ..,f_L\}$. The area value for extracting D is given by equation 2 [56]. In this equation the first term accounts for literals saved by not repeating the kernel, the second term accounts for literals saved by not repeating the co-kernels and the last term accounts for the number of literals introduced by extracting kernel D.

$$\left\{(R-1)\cdot\sum_{(i=1)}^{P} lit(d_i)\right\}+\left\{(P-1)\cdot\sum_{(i=1)}^{R} lit(q_i)\right\}-R \qquad (2)$$

where $lit(d)$ denotes the number of literals of cube d.

The power (saving) value for extracting kernel D is defined in equation 3. In this equation, the first term accounts for the reduction in the load on the inputs of the kernel. The second term accounts for the reduction in the load on the inputs to the co-kernels of the given kernel. The third term accounts for the cubes of the function that are removed from the original SOP representation of the function. Note that in this representation $d_i.q_j$ corresponds to a cube of the original SOP representation of the function. The forth term corresponds to power consumption at the output of the new node which is inserted into the network. The fifth term corresponds to the power consumption at the output of the cubes of the new node mized. Boolean optimization techniques use don't care sets derived from the network structure and/or specified by the user to compute a local don't care for each node in the network and then optimize the local function of each node using this local don't care. The following sections discuss how each of these techniques are used to optimize power consumption in a Boolean network.

2.1 ALGEBRAIC LOGIC RESTRUCTURING

Logic extraction applied to a set of Boolean functions (e.g. represented as a multi-level Boolean network) is the process of identifying and creating some new functions and variables, and re-expressing the original functions in terms of the original as well as the new variables. The optimization problem associated with the extraction is to find a set of new functions such that the resulting Boolean network is optimal (e.g. area, power dissipation). Factorization is the process of deriving a factored form from a sum-of-products form. The associated optimization problem is to find a factored form with the minimum cost (e.g. number of literals, switched capacitance) [8]. The complexity of finding the factorized form and decomposition of nodes is very high for any real sized network therefore approximation need to be made in order to make the problem tractable. A powerful approach introduced in [7] is to assume that logic functions in the network are algebraic expressions. Using this approach only common sub-expressions instead of sub-functions are considered for extraction and factorization. The approach is to first make all functions prime and irredundant and then look for sets of cubes which divide two or more expressions in a formal, algebraic way.

By restricting computations to algebraic manipulations, fast algorithms for division and other similar operations are obtained. These operations include: 1) algebraic extraction algorithms which provide fast methods for identifying logic sharing among different nodes, 2) substitution algorithms which help in reducing the area by taking advantage of functions that have already been implemented, 3) algebraic factorization algorithms which provide fast techniques for estimating the area of the network by computing the number of literals in the factored form of the network and 4) decomposition of nodes into smaller nodes by taking advantage of logic sharing within the same node. The application of these procedures during technology independent phase of logic synthesis has proved to be quite effective. The formulation of Rectangle Covering Problem introduced in [8] provides an efficient technique for generating the set of kernels, kernel intersections and common cubes [7] which represent good candidate divisors of algebraic expressions.

The traditional cost for optimization using algebraic techniques has been network area. This in general has been represented as the number of literals in the

A number of models have been proposed to more accurately estimate the contribution of nodes in the technology mapped network to the load values in the technology mapped network. *Load in the Factored Form* (LF) and *Load in the Sum-of-Products from* (LS) are two such models presented in [33].

Meanwhile, interconnect plays a role in determining the total chip area, delay and power dissipation, and hence, must be accounted for as early as possible during the design process. The interconnect capacitance estimation is however a difficult task even after technology mapping due to lack of detailed place and route information. Approximate estimates can be obtained by using information derived from a companion placement solution or by using stochastic/procedural interconnect models [51].

1.1.2 *Switching Activity Values*

Switching activity at a node gives the number of times in a clock cycle that the value at the output of the node changes its value. Hazards and glitches contribute to the switching activity of a node therefore the value of the switching activity at a node is dependent on the delay model being used. Power estimation techniques could account for steady-state transitions and/or hazards and glitches. Sometimes, the first component of power consumption is referred to as the f*unctional activity* while the latter is referred as the *spurious activity.* It is shown in [4] that an average of 10-40% of the total power is dissipated in glitching. Estimating power due to spurious activity at the technology independent phase of logic synthesis is however difficult and imprecise as it requires accurate delay information which is not available since the circuit is not yet mapped to gates in a cell library. Therefore, logic optimization techniques tend to consider only the functional activity of nodes.

2 Technology Independent Power Optimization

Given the functional description of a combinational design, the goal of technology independent logic optimization is to produce an area, delay or power optimal multi-level description of the circuit. This multi-level description or a Boolean network consists of single output nodes and edges where each node represents a Boolean function of the variables associated with its incoming edges. The optimal circuit produced during the technology independent optimization, is then used as the input to the technology mapping process. Technology independent optimization techniques include algebraic logic restructuring techniques and Boolean optimization techniques. Algebraic restructuring consist mainly of identifying common sub-expressions in the node functions and introducing new nodes or removing existing nodes such that the cost under consideration is mini-logic synthesis while Section 3 considers power optimization in PLAs (two-level logic). Sections 4 and 5 address technology mapping and post-mapping optimization for low power. The chapter is concluded with the description of the methodology (scripts) for power optimization in Section 6.

1.1 SOURCES OF POWER DISSIPATION

Power dissipation in CMOS circuits is caused by three sources: 1) the leakage current, 2) the short-circuit (rush-through) current, and 3) the capacitive current. The short-circuit and subthreshold currents in CMOS circuits can be made small with proper circuit and device design techniques. The dominant source of power dissipation is thus the charging and discharging of the capacitances associated with node n_i and is given by:

$$Pwr_i = \frac{1}{2} \cdot V_{dd}^2 \cdot f \cdot C_i \cdot E_i \tag{1}$$

where V_{dd} is the supply voltage, f is the clock frequency, C_i is the capacitance seen by the node and E_i (referred to as the *switching activity*) is the average number of transitions at the output of the node in a clock cycle. The product of C_i and E_i is referred to as the *switched capacitance*. At the logic level, it is assumed that V_{dd} and f are fixed, thus, the total switched capacitance of the circuit is minimized.

An estimate for the total power consumption of a network is obtained by summing equation 1 over all the nodes in the technology mapped network. Leakage and sub-threshold currents are small compared to charging and discharging currents and are ignored in this model. Also short circuit current can be modeled as an equivalent capacitance which is added to C_i. During logic synthesis, all architectural and technology related issues for the network being optimized have been decided. This means that the values for V_{dd} and f are fixed during logic synthesis. Therefore the main issues in computing the power estimate are in computing the load and switching activity value at the output of the node. These issues are discussed in the next section.

1.1.1 *Load Values*

Power dissipation is dependent on the physical capacitances seen by individual gates in the circuit. Accurate values for this load can be obtained for a mapped network by using the logic and delay information from the target library. Estimating this capacitance at the technology independent phase of logic synthesis is however difficult and imprecise as it requires estimation of the load capacitances from structures which are not yet mapped to gates in a cell library.

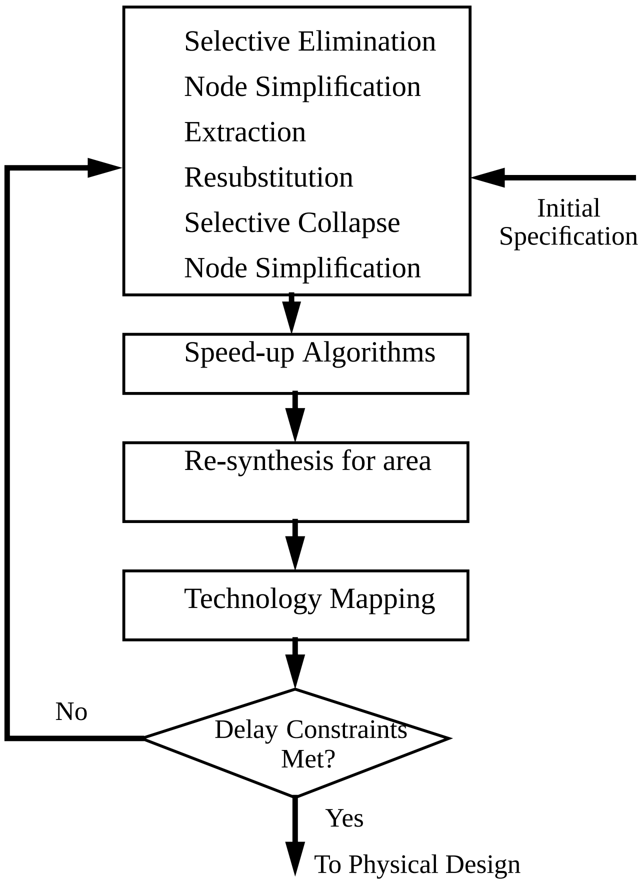

Figure 2. Logic synthesis flow.

that these techniques are a good starting point for developing a power optimization paradigm.

This chapter discusses the procedures and the methodology necessary to perform logic synthesis on a Boolean network such that after technology mapping the power consumption of the network is minimized. It assumes that the readers are familiar with the basic logic synthesis flow and operations. (A recent book by Hachtel and Somenzi [27] is an excellent reference on the subject.) This chapter is not meant to provide a comprehensive survey of related work, rather to summarize the basic techniques in a form suitable for researchers and workers in the field.

In the remainder of this section, power estimation issues that need to be considered during the power optimization process are discussed. Section 2 describes power optimization techniques for technology independent phase of multi-level already in the network to implement the function of another node in the network. Combinational speed-up algorithms work by identifying a set of nodes which when removed reduce the delay of the network. The operation then proceeds to eliminate these nodes from the network by collapsing them into their fanout nodes. Local optimization techniques include two-level node simplification using the local don't care set of the network which is derived from the network structure and also the don't care specified by the user. These techniques work on finding a minimal representation of the function of the node. Factorization and decomposition techniques operate by identifying logic sharing within the same function which when extracted, will result in a smaller representation of the function. Technology mapping is used to implement optimized Boolean network using the gates in the target technology such as standard cells and FPGAs.

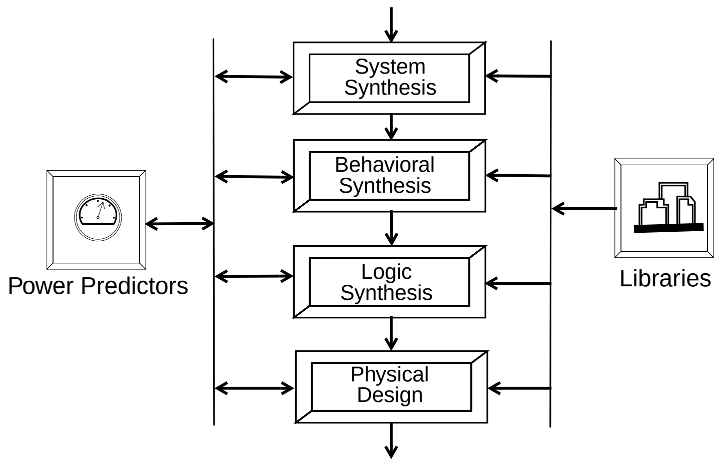

Figure 1. ASIC design flow.

Figure 2 shows the typical flow for logic synthesis targeting minimum area. The approach here is to first optimize the network for area and then use speed-up techniques to reduce the network delay. The network is then mapped to the target technology using a technology mapping algorithm. If the delay constraints are not met, then the network is re-synthesized to achieve the desired delay requirements.

Power consumption of a Boolean network is in general proportional to the area of the circuit. This means that even though it is possible to make area-power trade-offs, still the general trend is that as area is reduced, power consumption in the circuit is also reduced, that is, any paradigm for minimizing the power consumption of the network should also attempt to achieve an area solution that is comparable to the solution provided by an area optimization technique. Current techniques for minimizing circuit area have proved quite effective. This means

COMBINATIONAL CIRCUIT OPTIMIZATION

SASAN IMAN and MASSOUD PEDRAM
University of Southern California
Department of Electrical Engineering - Systems
Los Angeles, CA 90089
{iman,massoud}@zugros.usc.edu

1 Introduction

Low power VLSI design can be achieved at various levels of the design abstraction from algorithmic and system levels down to layout and circuit levels (see Figure 1). Power optimization techniques at the system, architectural (behavioral) design level, register-transfer (RT) level, physical design level, and the circuit level have been addressed by researchers in the past. Logic synthesis however, is an important part of the design cycle for a digital system. This means that in order to minimize power effectively, power has to be considered during logic synthesis and optimization.

Logic synthesis provides the automatic synthesis of gate-level netlists, minimizing some objective function subject to various constraints. The goal, in general is to obtain a minimum area circuit subject to a given delay requirement. Example inputs to a logic synthesis system include two-level logic representation, multi-level Boolean networks, finite state machines and technology mapped circuits. Depending on the input specification, the target implementation, the objective function, and the delay models used, different techniques are applied to transform and optimize the original RTL description. These techniques include the global area minimization strategy, combinational speed-up techniques based on local restructuring, local optimizations, and technology mapping for area or speed optimization.

The basic idea behind extraction techniques is to look for expressions that are observed many times in the nodes of the network and extract such expressions. The extracted expression is implemented only once and the output of that node replaces the expression in any other node in the network where the expression appears. Re-substitution is the process of using the function of a node