Low-Power Technology Mapping for Mixed-Swing Logic

Nicola Dragone*, Rob A. Rutenbar, L. Richard Carley, Roberto Zafalon**

Department of Electrical and Computer Engineering
Carnegie Mellon University, Pittsburgh, PA 15213, USA

* The author is also with PDF Solutions, Desenzano (BS), Italy ** STMicroelectronics, Agrate (MI), Italy

Abstract: *Mixed-swing logic employs multiple power supply rails and device threshold voltages and allows us to create richer cell libraries with a wider range of power/speed tradeoffs. However, mapping onto such a library with a conventional technology mapper will not exploit the full potential of a mixed-swing methodology. To remedy this, we have developed a new technology mapping tool that specifically targets mixed-swing logic. Our approach combines (1) efficient clustering and cluster-level delay budgeting for the uncommitted logic, with (2) an exhaustive search for the optimal cover that is rendered practical by the clustering process. Power savings up to 3X have been demonstrated with our mixed-swing solutions versus single power supply implementations.*

I. Introduction

Voltage scaling remains one of the most effective techniques for reducing power consumption because of the quadratic relation between the power supply voltage and the dynamic power consumption, which is generally the dominant contribution to the total power. Unfortunately, voltage scaling leads to reduced CMOS drive currents and increased gate delays. One option is to optimize the supply voltage and threshold voltages globally so as to achieve the minimum power given a required speed of operation [1], [2]. Another strategy is to "schedule" supply voltages to just meet the speed requirements of each segment of a functional block [3],[4]. However, a problem with these techniques is that they apply uniformly to all the logic of a particular functional block.

Mixed-swing circuit techniques focus on more fine-grain optimization at gate level. The key idea is to employ multiple power supply rails to exploit the delay slack that exists between the critical and noncritical timing paths. The goal is to implement only the timing critical portions of the logic using fast, higher-power, *high-swing* logic, while implementing the remainder of the design using slower, lower-power, *low-swing* logic. Low-swing gates dissipate less power internally, and also drive their output interconnect at a more power-efficient reduced voltage. Examples include the *clustered voltage scaling* (CVS) approach [5],[6], which uses standard CMOS gates operated at two voltages, along with level converter cells to connect high- and low-swing logic, and *QuadRail* [7], which introduces a 4-rail logic family with fast high-swing logic and power-efficient low-swing buffer/driver circuits in each individual gate.

In this paper we develop a technology mapping algorithm explicitly targeted at these mixed-swing approaches. The mapper minimizes area and power under an explicit timing constraint. By relaxing the timing (if allowed in the design) one can achieve a surprisingly smooth trade-off between power and speed. The algorithm combines Boolean matching techniques with an integrated timing/power budgeting algorithm. The rest of the paper is organized as follows. Section 2 reviews our circuit-level assumptions. Section 3 develops the technology mapping algorithm. Section 4 shows experimental results and Section 5 offers concluding remarks.

ISLPED'01, August 6-7, 2001, Huntington Beach, California, USA.

II. Mixed Swing Logic Circuits

An unpleasant reality in mixed-swing logic is the fact that not all gates may be able to drive each others' inputs. While a beneficial overdriving effect occurs when a high-swing gate is used to drive a low-swing gate, unacceptable leakage currents can result when a low-swing gate is used to drive a high-swing gate. Voltage level shifters have appeared as the standard solution to this problem, inserted on the signals between clusters, on incompatible source/sink gate types.

The solution we focus on makes use of two logic swings (four power rails) and two threshold voltages to avoid the need for added level shifters. As shown in Fig. 1, for a low-to-high swing interface, the driven gate is implemented using transistors with *higher* threshold voltage ($V_{T\text{-}high}$). This gives more room for voltage scaling without incurring in the penalty of high leakage currents. For this reason the difference between Vdd_1 and Vdd_2 (or equivalently Vss_1 and Vss_2) must be kept less than $V_{T\text{-}high}$.

III. Low Power Technology Mapping

A. Approach

Technology mapping consists of two major steps: *pattern matching* and *network covering*. Pattern matching determines whether a portion of the uncommitted logic network can be realized by a given cell in the library. Network covering selects those cells, among the set of all the potential matches, that minimize the cost function.

While pattern matching is independent from the cost function, gate selection depends on the specific objective of technology mapping. If the goal is area minimization, tree covering based on dynamic programming is very effective [8]. Unfortunately, this approach is not equally effective if delay is added into the figure of merit. The reason for this is that the cost of a match can no longer be computed without information, such as the load factor, determined by other matches in the tree. The problem of delay minimization is solved in [9] using piecewise linear functions as a representation for all the possible load values at each node. It is not always the case, though, that the fastest circuit is the best solution. We may be interested in a circuit that meets certain timing constraints and occupies minimum area. An area recovery procedure is presented in [9] as a subsequent phase to delay minimization. An alternative approach is described in [10] where the possible area-delay points are computed at each node of the tree. To keep the problem manageable, the area-delay space is discretized and an adaptive merging of the solutions is performed. At each node an area-delay curve is generated and all the solutions below this curve are dropped since they are slower or more area consuming than other solutions. All the approaches mentioned above are based on a simplified delay model that considers only an intrinsic component and a load dependent

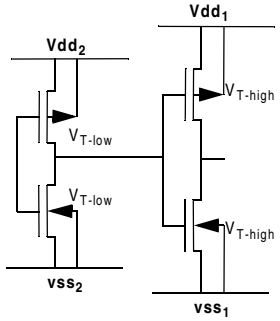


Fig. 1 A mixed-swing solution: dual swing + dual V_T style

component. The algorithms in this paper offer a different approach to network covering and consider the more general and accurate model for the gate delay which takes into account the input slope of the signals.

The delay of a gate is a function of the input transition time and output load. It can be expressed as follows:

$$T_D = t_{intr} + k_1 S + k_2 C_L \quad (1)$$

where t_{intr} is the intrinsic component of the delay, S is the input transition time, C_L is the capacitive load and k_1 and k_2 are two fitting parameters. The power dissipated by a digital gate is modeled as the contribution of three components:

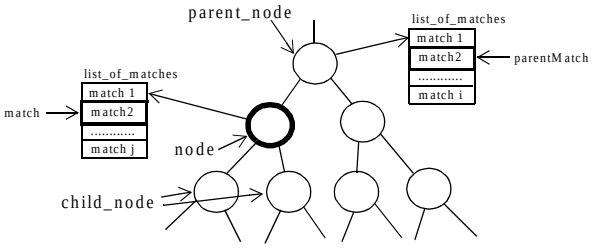
$$P_{total} = P_{load} + P_{internal} + P_{leakage} \quad (2)$$

where P_{load} is the net dynamic power due to charging and discharging of the output load, $P_{internal}$ is the internal gate dynamic power due to charging and discharging of the capacitances of internal nodes and short-circuit currents and $P_{leakage}$ is the static power due to leakage currents. $P_{internal}$ is again a function of the input transition time and output load. In [11] we showed an efficient library characterization methodology that allows us to drastically reduce the number of lookup tables while maintaining a good accuracy of delay and power models. The basic idea is to avoid the characterization of events with heterogeneous input levels and exploit a technique within the mapping engine that we call *slope rescaling* to derive missing information in our library delay or power models.

B. Network Covering

Given a network and a list of potential matches at each of its nodes, we want to identify the implementation of the circuit that minimizes the power consumption and satisfies the timing constraints. The extension of the problem to take into account area minimization as well is trivial. In the following discussion we assume that the network is represented as a tree of nodes. The pseudocode of the algorithm for minimum cost covering appears in Fig. 2. In the following discussion, for simplicity, we will assume that each matching gate covers exactly one node in the decomposed network. Because of the recursive nature of the algorithm the nodes are traversed from their output to their inputs (*preorder traversal*) and then from their inputs to their output (*postorder traversal*). We now analyze in detail these two phases.

Preorder traversal: In order to identify the optimal covering, the cost associated with all the matches at each node of the decomposed network must be computed. The cost of selecting a particular gate matching a node is given by the cost of the gate itself and the cost of the minimum cost cover of the subtrees rooted at its input pins. Two factors influence the cost of a match: the capacitive load at the output and the transition time at the input signals. During preorder traversals, the capacitive load of a gate matching a node is known if we propagate a pointer to the gate matching the parent node ('parentMatch' in Fig. 2). Conversely, the value of the input slopes is not known since the subtrees rooted at the input pins of the




```
minCost (node,parentMatch) {                                          1
    powerCost = ∞;                                                    2
    forall_matches (match,node) {                                     3
        required_delay = estimate_required_delay(match,parentMatch);  4
        forall_children (child_node,node) {                           5
            minCost(child_node, match);                               6
        }                                                             7
        [delay,power] = computeCost(match);                           8
        if (delay < required_delay & power < powerCost) {             9
            powerCost = power;                                        10
            match.selected;                                           11
        }                                                             12
    }                                                                 13
}                                                                     14
```

Fig. 2 Pseudocode of the algorithm for optimal covering

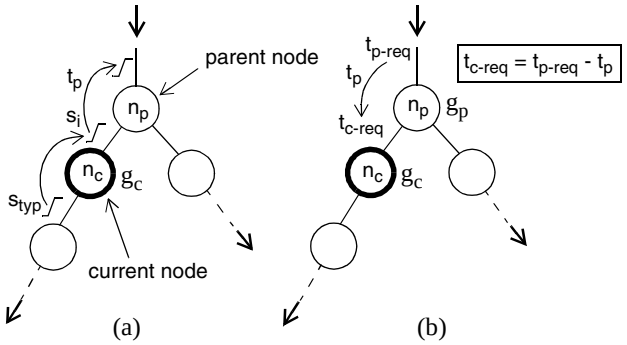


Fig. 2 Preorder traversal of a node.

match have not been visited at this point of the recursion. Hence, the cost of a gate can be computed only during the postorder traversal of the node. Nevertheless, during the preorder traversal of a node, a good *approximation* of the required time at its output can be computed. The approach used in estimating the required arrival time is shown in Fig. 2. The required arrival time $t_{c\text{-}req}$ at the output of the current node n_c is computed by subtracting the propagation delay t_p through the gate g_p matching the parent node n_p from the required arrival time $t_{p\text{-}req}$ at the output of the parent node (Fig. 2b). The required time at the primary output of the tree is the target delay. In order to derive the propagation delay t_p we need an estimate of the slope s_i of the signal at the input of the gate g_p. Although its value will be influenced by all the gates belonging to the fanin cone of the node n_p, we assume that only g_c, which is the gate matching n_c, eventually determines that value, decoupling the signal from the rest of the circuit. Under this assumption, we used a dummy signal with a typical slope (s_{typ}) at the input to g_c to derive s_i (Fig. 2a).

Postorder traversal: during the postorder traversal of a node the best match is selected. The cost of the gate matching the node can be computed because the subtrees rooted at its input pins have already been mapped. The load is also known since the pointer to the gate matching the parent node is available. The match that gives minimum power consumption while guaranteeing that the delay at its output is smaller than the required time is selected. The computation of the delay at the output of the node is shown in Fig. 4. Given the input slope s_i from the child node n_{ch} and the load associated to the match of the parent node n_p the propagation delay t_p and the output slope s_o can be easily derived. The arrival time at the output $t_{c\text{-}arr}$ is simply obtained by adding t_p to the arrival time $t_{ch\text{-}arr}$ at the input pin. The actual cost of a match is given by the value of power dissipation. It can be expressed as follows:

$$powerCost = P_{int} + P_{leak} + P_{load} + P_{int}^{(p)} + P_{leak}^{(p)} + \sum_{i=1}^{fanin} P_{subtree(i)} \quad (3)$$

where P_{int} is the internal power of the match, P_{leak} is the leakage power of the match, P_{load} is the dynamic power of the match, $P_{int}^{(p)}$ is the internal power of the gate matching the parent node, $P_{leak}^{(p)}$ is the leakage power of the gate matching the parent node and the last term is the power contribution of the subtrees rooted at the input pins of the match. $P_{int}^{(p)}$ and $P_{leak}^{(p)}$ must be included because the choice of the match for the current node influences the internal and leakage power of the gate matching the parent node. P_{int} and P_{leak} are computed using the actual voltage swing and slope of the signal from the child node

Our algorithm for optimal network covering works very similarly to the algorithm proposed in [12]. It can be considered as its generalization to handle a delay constrained problem and a more complex cost function that depends on matches different from the one the cost is being computed for. The problem is that the procedure is now much more computationally intensive since the recursions must always reach the leaves of the tree. In the following, we suggest a solution to this problem based on network DAG clustering.

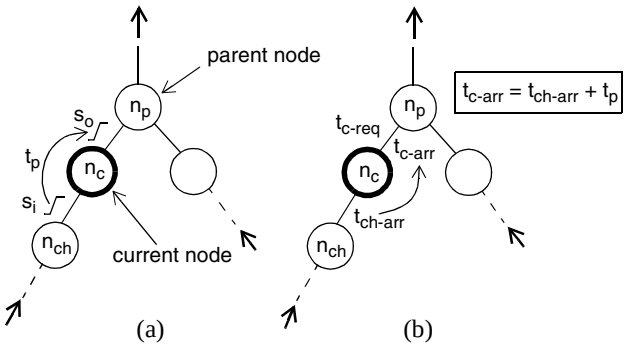


Fig. 4 Postorder traversal of a node.

C. DAG Clustering

DAG clustering is the technique we used to reduce the computational complexity of the algorithm for optimal network covering. This is done at the cost of some optimality in the solution. Nevertheless, if clustering and delay budgeting, described in the next section, are carefully designed, the quality of the final solution can still be excellent. The basic idea behind DAG clustering is to restrict the search space by computing the optimal solution on each cluster separately. The computational cost, instead of being exponential in the number of nodes of the DAG, is exponential in the number of nodes of the clusters and linear with the number of clusters. We chose to cluster the DAG in trees with a fixed maximum size, where the size of a tree is defined as the number of its nodes. The biggest problem with DAG clustering is that potentially good solutions can be eliminated since matches crossing the boundaries between two clusters are no longer valid. To limit this, a pre-mapping is performed to explore the DAG. Such a pre-mapping simply looks for the covering that utilizes the minimum number of gates. Clustering is then performed in such a way that none of the matches comprising this preliminary covering is made invalid. In practice, this means that we want to avoid cases where large cells that have the possibility to cover large portions of the DAG are eliminated. In fact, such cells are generally very effective in reducing delay, area and power for they allow us to get rid of many wires. The simple rules adopted in our DAG clustering are illustrated in Fig. 5 and are summarized here:

- Clustering considers only edges external to the matches comprising the preliminary covering
- Clusters must not exceed a maximum size
- Clustering is always performed if there is a multiple fanout

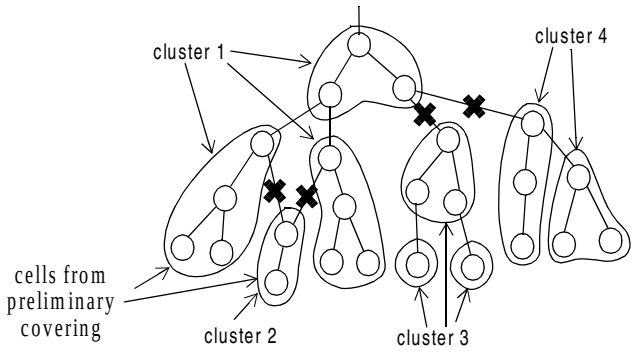


Fig. 5 Network clustering.

D. Delay Budgeting

To be able to use the optimal covering procedure we need to assign an adequate time interval to each cluster. The only information we can rely on is the target delay at the primary outputs and the technology independent description of the network. A cluster in the decomposed network is comprised of several nodes, one output and one or more inputs. Outputs and inputs of a cluster are generally internal edges of the DAG, not necessarily corresponding to primary inputs or primary outputs. Arrival times and required times at the internal edges of the DAG are unknown because the only constraint is on the target speed that the circuit must guarantee.

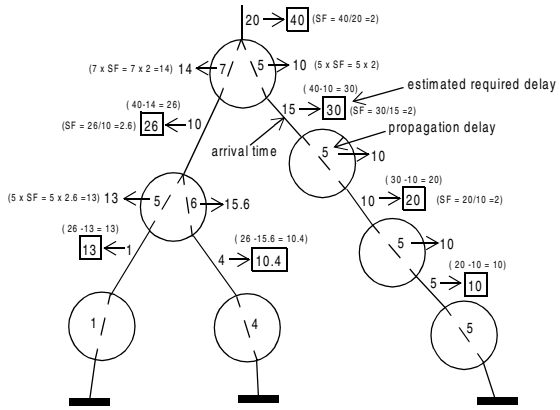
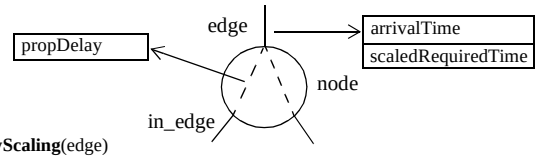


Fig. 6 Delay budgeting: static timing analysis and delay rescaling.

To use the procedure for optimal covering, we need a *delay budgeting* strategy for assigning a valid time interval to each cluster. Our implementation of delay budgeting incorporates two steps: the first step entails *static timing analysis* of the network; the second step entails *delay rescaling*. An example of delay budgeting appears in Fig. 6. In this example a simple DAG is considered. Static timing analysis is performed on the network, decomposed in inverters and nors, considering real gates from the technology library. The estimation of the capacitive contribution of the interconnects is done using the wireload models usually provided along with the technology library. During a postorder traversal of the DAG, the delays are computed and annotated on the edges. In Fig. 6 we are also showing the propagation delays associated with each timing arc. Delay rescaling is needed because the target delay for the application can be larger or smaller than the actual delay. How delay scaling is implemented is also illustrated in Fig. 6. Starting from the output, in a preorder traversal of the DAG, the *scale factor* (SF), representing the ratio between the estimated required time and the arrival time, is computed at each node. The scale factor is used to scale up or down the propagation delays that subsequently are used to compute the value for the required time at each edge.

The pseudocode for delay budgeting is presented in Fig. 7. The proposed implementation of delay budgeting guarantees the relaxation of the required arrival times in the branches of the DAG with positive scaled delay slack and the delay equalization of the paths in the DAG. These concepts are of crucial importance if our goal is power reduction and if we want that this is accomplished uniformly in the circuit.




```
delayScaling(edge)                                                    1
{                                                                     2
    node = target(edge);                                              3
    if (edge == primary output) {                                     4
        edge.scaledRequiredTime = targetDelay;                        5
    }                                                                 6
    scaleFactor = edge.scaledRequiredTime / edge.arrivalTime;         7
    forall_in_edges (in_edge , node) {                                8
        scaledPropDelay = in_edge.propDelay * scaleFactor;            9
        in_edge.scaledRequiredTime = edge.arrivalTime - scaledPropDelay; 10
        delayScaling(in_edge);                                        11
    }                                                                 12
}                                                                     13
                                                                      14
```

Fig. 7 Delay rescaling: pseudocode of the algorithm.

IV. Experimental Results

The algorithms presented in this paper have been implemented in a tool called *teCMUpper*.Three kinds of information are fed as input to the mapper: the technology libraries, the specifications on the target speed and the description of decomposed network. The technology independent description of the network is represented in terms of inverter and nor cells. The

database of the Dual-Swing/Dual-V_T libraries is loaded by the mapper and the signature of each cell is computed. This is achieved by building the equivalent BDD for each cell [13]. Once all the possible matches have been identified, a preliminary covering of the network aiming at the solution with a minimum number of gates is performed. The output of this phase is used during DAG clustering to prevent good matches from being eliminated. The time intervals assigned to each cluster are computed using our delay budgeting. Depending on the mapping effort set by the user, the maximum number of nodes in a cluster can vary. Bigger clusters allow for a more complete exploration of the search space, but at the cost of longer execution times. A final evaluation of the critical path delay and power consumption, based on the power and delay models introduced in Section II, is done before the netlist of the mapped circuit is written out in Verilog.

The tool described in the previous paragraph has been used to map circuits from the ISCAS85 set of combinatorial benchmarks. The target library included 50 combinational cells implemented in an industrial 0.18µm process. The high voltage swing is between 0V and 1.8V while the low voltage swing is defined between 0.35V ad 1.45V. The low threshold voltage is 0.5V and the high threshold voltage is 0.7V. The results obtained using *teCMUpper* on a subset of the circuits used as benchmarks are shown in Fig. 8. For each testbench we have four different plots: Delay slack vs. target delay, Power consumption vs. target delay, Power savings vs. target delay, High swing cell fraction vs. target delay.The results demonstrate *teCMUpper*'s ability to attain power savings by operating noncritical gates at lower swing. The power reduction factor is up to 3X, depending on the specific target delay. If the timing constraints are stringent, only few cells can be operated at low swing. The number of low-swing gates increases as the target delay is relaxed. It is worth mentioning that power savings are obtained against completely high-swing circuits mapped by *teCMUpper* and therefore already optimized in terms of power consumption.

Fig. 8 also shows the number of gates comprising the decomposed description of each circuits. The execution time varies, but is not always increasing with the size of the circuit. This is because of our clustering procedure. Circuits with a high number of multi-fanout cells generate a higher number of clusters that translates into faster execution time. The results in Fig. 8 have been obtained using a *medium mapping effort* that implies a maximum cluster size of 20 nodes. This value is probably excessive if bigger circuits were to be considered. Nevertheless, good quality can be achieved even reducing the maximum size of the cluster (*low mapping effort*). The only problem we had with *teCMUpper* is the unjustified negative delay slack in a few mixed-swing implementations. This was due to the approximation of the slope of the signals used at the input of the clusters. This can have great impact (proving the importance of a delay model that accounts for input slope as well) if the critical path is made of small clusters. A final pass for assuring timing enclosure could resolve this.

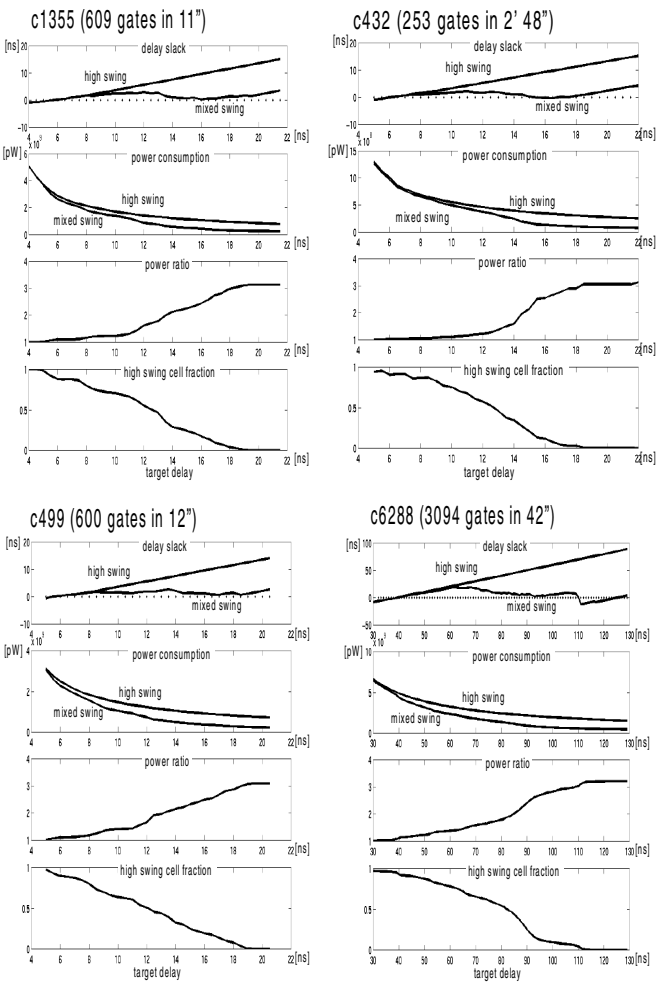


Fig. 8 Power-delay trade-offs for our mapped circuits

V. CONCLUSIONS

We have developed an effective mapping technique for libraries that employ mixed-swing voltages and multiple threshold voltages to reduce power consumption in digital circuits. Power savings is achieved by operating non critical gates at reduced swing and using higher threshold voltages to limit the leakage currents in high-swing gates driven by low-swing input signals. In our mapping strategy, we first look for the smallest covering of the network, then use clustering and delay/power budgeting to allow an aggressive power optimization of this cover under a delay constraint. Our strategy selects also the optimal voltage swing of the gates and the threshold voltage of their active devices. The proposed ideas have been implemented in a tool called *teCMUpper* and its ability to generate low-power mixed swing solutions has been proven on several benchmarks. Smooth trade-offs of delay for power (up to 3X, depending on the tightness of the timing constraint) have been demonstrated.

REFERENCES

[1] D. Liu and C. Svensson, "Trading Speed for Low Power by Choice of Supply and Threshold Voltages", *IEEE J. Solid-State Circuits*, Vol. 28, January 1993, pp. 10-17.

[2] D.J. Frank et al, "Supply and Threshold Voltage Optimization for Low Power Design", *Proc. IEEE/ACM ISLPED97*, August 1997, pp. 317-322.

[3] Raje, S. and Sarrafzadeh, M. "Variable voltage scheduling." *Int. Symp. on Low Power Desig*n, 1995, pp. 9-14.

[4] J. Chang and M. Pedram, "Energy Minimization Using Multiple Sup-ply Voltages," *IEEE Trans. on VLSI System*s, vol. 5, pp. 1–8, Dec. 1997.

[5] Usami, K. and Horowitz, M. "Clustered voltage scaling technique for low-power design." *Int. Symp. on Low Power Desig*n, 1995, pp. 3-8.

[6] M. Igarashi et al, "A Low-Power Design Method using Multiple Supply Voltages", *Proc. IISLPED*, Aug. 1997, pp. 36-41.

[7] L.R. Carley, "QuadRail: A Design Methodology for Ultra Low Power Integrated Circuits", *Proc. IEEE ISLPED94*, April 1994.

[8] E. Detjens et al, "Technology Mapping in MIS", *CAD87*, 1987, pp. 116-119.

[9] H. Touati, *Performance Oriented Technology Mapping*. PhD thesis, University of California, Berkeley, 1990.

[10] K. Chaudhary and M. Pedram, "A near optimal algorithm for technology mapping minimizing area under delay constraints", *DAC92*, June 1992, pp. 492-498.

[11] N. Dragone, *Low-Power Technology Mapping for Mixed-Swing Logic*, MS Thesis, Dept. of ECE, Carnegie Mellon University, 2000.

[12] K. Keutzer, "DAGON: technology binding and local optimization by DAG matching", *DAC87*, July 1987, pp. 341-347.

[13] F. Mailhot and G. De Micheli, "Algorithms for Technology Mapping Based on Binary Decision Diagrams and on Boolean Operations", *IEEE Trans. on CAD*, Vol. 12, May 1993, pp. 599-620.