Technology Mapping for SOI Domino Logic Incorporating Solutions for the Parasitic Bipolar Effect

Shrirang K. Karandikar and Sachin S. Sapatnekar *Fellow, IEEE*
Department of Electrical and Computer Engineering,
University of Minnesota.
email: {srirang, sachin}@ece.umn.edu

***Abstract*— We present a technology mapping algorithm for implementing a random logic gate network in domino logic. The target technology of implementation is Silicon on Insulator (SOI). SOI devices exhibit an effect known as Parasitic Bipolar Effect (PBE), which can lead to incorrect logic values in the circuit. Our algorithm solves the technology mapping problem by permitting several transformations during the mapping process in order to avoid PBE, such as transistor reordering, altering the way that transistors are organized into gates, and adding pmos discharge transistors. We minimize the total cost of implementation, which includes discharge transistors required for correct functioning. Our algorithm generates solutions that reduce the number of discharge transistors required by 53%, and reduces the size of the final solution by 6.3% on average. We compare our results with a modification of a current technology mapping algorithm for bulk CMOS domino logic that reduces the cost of the final solution, and find that our algorithm outperforms this method.**

I. Introduction

As the scaling of bulk CMOS proceeds along the roadmap, interest in Silicon on Insulator (SOI) as an alternative technology has increased. In addition, manufacturing processes have matured enough to allow large circuit implementations in SOI at acceptable defect levels. However, current algorithms used for implementing circuits in bulk CMOS are inadequate for SOI. The best approaches and traditional design techniques from bulk CMOS could be disastrous if applied to SOI. An example is the use of precharge transistors in bulk CMOS, to offset the charge sharing effect. As we will show in section III-B, if precharge transistors are used in SOI, we will possibly obtain circuits that do not function correctly. Current EDA techniques too do not adequately address the needs of SOI design, and there is a requirement for new algorithms and tools targeted towards SOI designs, since simple modifications to existing algorithms by adding post-processing steps leads to solutions that are sub-optimal, as shown in later sections. This paper address the libraryless technology mapping problem in the context of SOI. We present an algorithm that maps an arbitrary two-input logic gate network to domino logic in a manner that eliminates the "Parasitic Bipolar Effect" (PBE) [1], [2] by applying transformations such as reordering transistor stacks in the gate, altering the structure of the gates to reduce their susceptibility to PBE, and inserting pmos pre-discharge transistors at appropriate points in the circuit. We take an approach of using metrics for area and delay that will generally be acceptable for any SOI implementation. The corresponding loss of detail is compensated by the reduction in complexity of the algorithm. The mapping step can be followed by a post-processing step that is specific to a given set of SOI technology parameters, possibly including transistor sizing, which our work does not address. During the mapping, the algorithm minimizes the cost of the implementation: for example, for an area objective, it would minimize the total number of transistors, including pre-discharge transistors. The techniques that we use to control PBE operate by ensuring that the body voltage of the SOI device never becomes very high, so that PBE is never triggered. This yields an added side benefit of reducing the timing hysteresis exhibited by SOI circuits due to variations in the body voltage. In narrowing the range of permissible voltages for the body (to reduce the PBE), we make the timing behavior of the circuit more predictable.

This paper is organized as follows. We briefly introduce SOI and domino logic, and present problems typical to SOI implementations and solutions to these, with emphasis on overcoming PBE. We then present our algorithm, which performs the technology mapping specifically taking PBE into consideration. We also present a simple alteration to the technology mapping algorithm currently used for

This research was supported in part by the NSF under award CCR-0098117 and by the SRC under contract 99 TJ-692

mapping into bulk CMOS domino logic that reduces the number of discharge transistors required. The efficacy of our approach, and extensions to it are shown in the results section, wherein we present cost functions that minimize the area and delay of the solution. We conclude with directions for future work.

II. Background

A. Silicon-on-Insulator

SOI has long been used in a variety of fields, such as radiation-hardened and high-voltage applications [5], [6]. SOI circuits have attractive properties as compared to bulk CMOS, such as reduced source- and drain-to-substrate capacitances, no body effect in series stacks of transistors and suitability for reduced V_{dd} operation for given performance [7], [8]. In addition, due to reduced capacitances, SOI devices consume less power [9], [10]. Moreover, since transistors are isolated from each other by an insulator, they require smaller area. In spite of being smaller, faster and consuming less power than bulk CMOS, SOI has not found widespread use in the VLSI community until recently. However, recent advances in manufacturing processes coupled with a realization of the limitations of bulk CMOS technology have led to a renewed interest in SOI. Increased understanding of how SOI devices behave, and possible solutions to their quirks has lead to a wider acceptance of SOI in the VLSI community. Recently, SOI has been used in a number of high end microprocessor designs, e.g. IBM Power PC [11], [12], HP-PA 8700 [13], and others [14], [15], as well as other high performance logic circuits [16], [17] and [18].

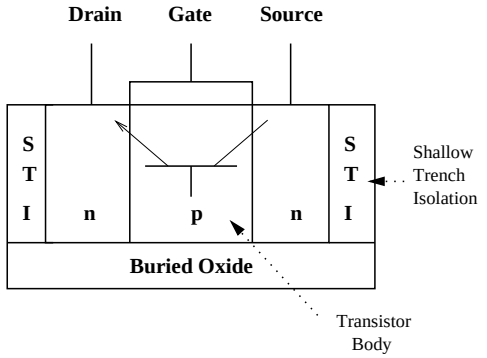


Fig. 1. SOI Transistor Fabrication, and Presence of the Parasitic Bipolar Transistor

The manufacturing process of SOI is very similar to that of bulk CMOS. One of the processes used for SOI fabrication is SIMOX [6]. The preliminary step is to implant a layer of silicon dioxide beneath the surface of the silicon wafer. This is the "Insulator" in Silicon-on-Insulator. Transistors are created by masking and doping exposed regions on the layer of silicon above the silicon dioxide. Once transistors are fabricated in this manner, they are isolated from other devices by another layer of silicon dioxide, called Shallow Trench Isolation (STI). This structure is shown in figure 1. Due to this structure, the bodies of individual transistors are electrically isolated from the rest of the circuit, unlike bulk CMOS circuits where the body is identical to the substrate or well, which is connected to a supply node. Hence, the body potential in SOI is free to seek its own level [19], and is determined to a large extent by the voltage levels at the source and drain of the transistor, due to leakage currents. Changes in the gate voltage also affect the body potential due to capacitive coupling. Thus, if the gate is held low and drain and source are at a logic high for an extended period of time, charge accumulates in the body due to leakage current and impact ionization [8]. This causes the body potential to increase. This variance in body voltage is the main source of problems associated with SOI. The change in body voltage of a device results in different switching speeds at different time instants. Also, switching speeds across a circuit can vary due to different devices having different body voltages. Another problem that a high body voltage can cause is called the Parasitic Bipolar Effect (PBE), and is described in more detail in section III.

B. Domino Logic

Domino logic [20] is a favored approach for implementing timing critical circuits due to their high performance. The basic structure of a domino gate is as shown in figure 2(a). During precharge, the dynamic node is charged to a high logic value, and the output of the gate is set to logic zero. During the evaluate phase the n-clock transistor switches on, and depending on the inputs to the pull down network, the dynamic node is either discharged or retains its charge. If the dynamic node switches, the output of the gate goes to a logic high value. OR logic functionality is obtained in domino by connecting n-transistors in parallel in the pull down network. Similarly, AND functionality can be obtained by connecting n-transistors in series. More complicated logic operations are obtained by combining these basic operations. An example circuit, shown in figure 2(a), implements the logic function $(A + B + C) * D$.

Note that the n-clock transistor shown in the figure is only required for domino gates connected to primary inputs. The outputs of domino gates change only during the evaluate phase of the clock cycle, during the precharge phase they are held low. Hence, any n-transistors driven by these outputs will be off during precharge. For a domino

gate, this means that if its inputs are coming from another domino gate, the pull down network will not switch on during precharge, and hence there is no need for an n-clock transistor. Such a configuration is sometimes referred to as footless domino.

III. Parasitic Bipolar Effect in SOI

A. Issues with SOI Implementations

The advantages of SOI listed in the previous section come at a cost. Prominent among these are the hysteretic V_t variation [21], in which the behavior of a transistor varies according to its previous switching history. Another serious problem associated with SOI devices is the PBE, described in more detail in the following sections.

B. Parasitic Bipolar Effect

PBE occurs in certain circuit topologies and switching patterns, such as stack OR-AND structures. The topology typically involves an off transistor situated high in the stack, with the source and drain voltages in the *high* state. Over a period, this causes the body voltage to be *high*. When the source is subsequently pulled down, either by the clocked evaluation transistor in dynamic circuits or by an input signal, a large forward body bias is developed across the body-source junction, causing bipolar current to flow through the lateral parasitic bipolar transistor (shown in figure 1). The parasitic bipolar current and the FET current (caused by noise and aggravated by the low Vt) result in a loss of charge on the dynamic node.

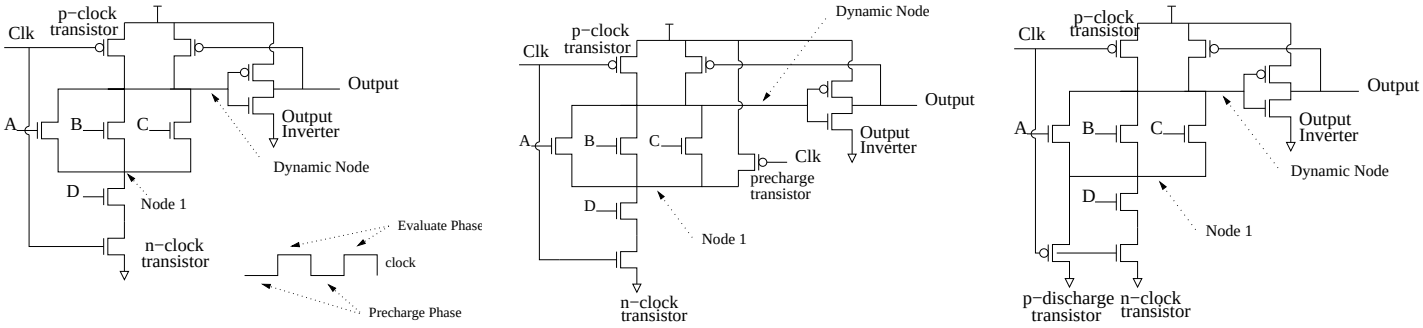
This can be illustrated by an example from [1]. For the circuit shown in figure 2(a), consider a steady state condition with inputs A = 1, B = 0, C = 0 and D = 0. Transistor A is on, and the other n-transistors in the pulldown logic network are off. Hence, as the dynamic node charges to a high value during precharge, node 1 is charged to a potential of $V_{dd} - V_{threshold}$. Recall that transistors B and C are off at this point. Under this set of conditions, the bodies of transistors A, B and C charge to a *high* value over a sufficiently large period of time. Now if signal A switches *low*, the potential at node 1 remains at its high logic value since transistor D is off. Moreover, the switching event on A sets the body voltage for device A to be low (due to strong capacitive coupling), but leaves the body voltages of B and C to be high. In the evaluate phase, if D is switched on (with A, B and C off), node 1 is suddenly pulled down. This causes the parasitic bipolar transistor to switch on, since the base and collector of the parasitic transistor are high while the emitter has been pulled low, and a large current can flow through transistors B and C. If this current is large enough, it can pull the voltage at the dynamic node to a level small enough to switch the output of the gate to a *high* value. Thus, even though the output node should have evaluated to *low*, it ends up as a *high*. In this manner, the PBE can result in a wrong evaluation if not accounted for in an SOI implementation. This value will eventually be brought to its correct value by the keeper, but this is liable to take time and may cause erroneous circuit behavior temporarily, or even permanently if any state bits are altered in the interim.

It is interesting to note that this is a typical configuration in bulk CMOS implementations that requires the use of precharge transistors, as shown in figure II-B. In bulk CMOS, charge sharing is a significant problem, and precharge transistors are used to ameliorate its effects at the cost of a slight performance penalty. If any of the transistors A, B or C are on during evaluate, with transistor D off, charge on the dynamic node may be distributed to Node 1, and the potential on the dynamic node may drop low enough for the output inverter to switch erroneously. In figure II-B, a precharge transistor controlled by the clock connects node 1 to V_{DD}. This transistor charges the intermediate node 1 to a high value during precharge, and redistribution of charge does not occur. SOI circuits, however, exhibit much lower drain and source capacitances, and charge sharing is not as problematic as the parasitic bipolar effect. Using precharge transistors rather than predischarge transistors as in the example above, will almost guarantee breakdown of correct circuit operation even without the sequence of transistor switchings described above.

C. Solutions to the PBE

There are a several solutions for handling the PBE, and we will enumerate these as follows:

1) The keeper pmos device can be sized up to provide some resistance to the PBE, but such a choice comes at the expense of a performance penalty due to the increased capacitance that it presents at the dynamic node and particularly at the output node.
2) Body contacts connected to the ground lines in the case of nmos, or V_dd in the case of pmos transistors can be added selectively to some devices in the circuit, but this results in an increased area and input capacitance, and is a choice that is generally avoided by SOI circuit designers [5].
3) If the cost is acceptable, parallel stacks can be broken up by transistor replication. For example, $(A + B + C) * D$ can be re-implemented as $A * D + B * D + C * D$ (D is replicated three times in this example). If this implementation is connected to ground, there are no paths for transistor bodies to charge high, since parallel stacks have been



(a) An Example of a Domino Gate

(b) Precharge Transistor Used to Avoid Charge Sharing in Bulk CMOS

(c) Avoiding the Parasitic Bipolar Effect using a Predischarge Transistor

Fig. 2. Domino Gate Structure, and Modifications Applied in Bulk CMOS and SOI Technologies

eliminated. A drawback of this approach is the cost requirement of duplicating logic for each finger of a potentially wide parallel stack.

4) The stack of transistors in a gate may be reordered to reduce its susceptibility to the PBE. For the gate in figure 2(a), if the parallel stack of transistors A, B and C is moved to the bottom of the gate, so that the sources of all three transistors are connected to ground, it will not be possible to excite the PBE. This approach exploits the reduced charge sharing effect and reduced delay dependency on stack ordering in SOI technology.
5) The above procedure works only if there is only one parallel stack per gate. If this condition is not met, it may be possible to remap the Boolean logic to the gates to ensure that each gate contains no more than one parallel stack, which can then be reordered within the gate to connect it to ground.
6) Intermediate nodes in a stack may be predischarged in every clock cycle. In figure II-B, a clock-driven p-discharge transistor has been added to the circuit. Such a transistor can be used to connect intermediate points in the circuit (such as node 1) to ground. Thus, during every precharge cycle these intermediate nodes are discharged, and the bodies of transistors in the pulldown network are not permitted to charge to a high voltage level. The drawback of using p-discharge transistors is the additional load on the clock network.
7) Complex domino structures with the output inverters replaced by static NAND or NOR gates may be used to break up large parallel logic trees [7].

One approach to performing these optimizations is to start with the original design in bulk silicon, analyze it to identify potential sources of the PBE, and apply the above transformations to eliminate them. The main criticism of such an approach is that the solutions obtained are local in nature. For example, while a particular mapping may be optimal for bulk CMOS, it becomes non-optimal if it requires a large number of p-discharge transistors. A better approach would be to perform the mapping from logic gates to the transistor level, *keeping the requirement of p-discharge transistors in mind*. In section V we propose an algorithm that performs such a mapping.

In this work, we avoid the first three transformations of sizing the keeper, adding body contacts and splitting parallel stacks using duplication, since they can cause significant cost increases, and instead, focus on the rest. We perform our procedure at the time of synthesis, prior to circuit sizing, and note that the transformation that sizes the keeper is more appropriately applied after or during the transistor sizing step. In applying the remaining transformations, we will penalize the addition of clock-connected transistors and additional transistors required due to gate reorganization, since they represent a cost-increasing transformation. Reordering changes delay, but since diffusion capacitances are relatively low, we ignore them as a first order approximation.

IV. Technology Mapping for Domino Logic

Synthesis of domino circuits is more complicated than that of static circuits. The added complexity is due to the monotonic nature of domino logic which forces it to implement only non-inverting functions. Therefore, domino logic can only be mapped to a network of non-inverting functions, where needed logic inversions must be performed at either primary inputs and/or primary outputs. Any random logic network can be transformed into a

network of non-inverting functions by finding a unate network representation[1].

Generating a unate network from a binate random logic network may require logic duplication since both positive and negative signal phases may be needed. An algorithm for finding the minimum logic duplication necessary when transforming a binate random logic network into an inverter free unate network has been developed in [22]. Binate-to-unate network conversion will at most double the amount of original logic (with typical overheads being much smaller) and will not increase the number of logic levels.

However, in order to avoid the complexity of [22], we use a simple bubble pushing algorithm to generate the unate network. In our implementation we simply attempt to push inverters as far back as possible (i.e., towards the primary inputs), by applying DeMorgan's laws where necessary. If inverters cannot be pushed through a gate, e.g., when both positive and negative phases of a signal are required, logic duplication is necessary. After a unate network representation has been created, the network can then be technology mapped to domino gates. Note that starting from an initial decomposed network consisting of 2-input AND-OR gates and inverters, the unate network thus obtained will only consist of 2-input AND-OR gates, since all inverters have been removed in the unating process.

Technology mapping has traditionally used library based methods. In [23], the authors presented a library free algorithm for technology mapping. Library free approaches have the advantage of searching a large solution space, while library based methods are restricted by the size of the libraries. Since nmos pulldown networks for domino gates can be larger than their static counterparts, any precharacterized library can only explore a fraction of the exponential number of possible gate functionalities. A parameterized library overcomes this limitation although it is restricted by the use of more approximate delay models. Parameterized libraries have been used successfully to design industrial circuits, e.g., in [24].

The algorithm of [23] uses a dynamic programming based approach. A set of tuples[2] of $\{W, H, C\}$ (width, height and cost corresponding to a pull down network configuration) are associated with each logic gate of the network. The cost here may be the number of transistors, the number of logic levels, or the delay. The values of maximum gate width and height determine the number of tuples associated with each gate. The input network of 2-input AND-OR nodes is traversed from primary inputs to primary outputs, and sub-solutions for each node for all possible configurations of $\{W, H\}$ are calculated based on the sub-solutions of its inputs. Note that, depending on the inputs, a gate may not have all combinations of $\{W, H\}$ and in practice, only a fraction of $W_{max} \times H_{max}$ tuples are associated with each gate. When calculating the sub-solution of a node, all permissible configurations of the input nodes are enumerated, and the best ones are selected. Once all valid tuples for a node have been calculated, the $\{1,\ 1\}$ tuple is constructed by selecting the best (lowest cost) sub-solution for that logic gate, and converting this partial structure into a domino gate by adding the clock transistors, the output inverter and a keeper transistor. Thus, the cost of a $\{1,\ 1\}$ configuration is the lowest cost among all other configurations plus 5. The basic operations for combining input tuples to form the tuples of the current node are AND and OR. These operations are as follows. An AND operation requires a series connection of inputs. Hence, the $\{W_1, H_1\}$ and $\{W_2, H_2\}$ solutions of the input nodes are combined to form the $\{\max(W_1, W_2), H_1 + H_2\}$ solution. Similarly, $\{W_1, H_1\}$ and $\{W_2, H_2\}$ solutions of the inputs can be combined as $\{W_1 + W_2, \max(H_1, H_2)\}$ for an OR node. A more detailed explanation of combining inputs of an AND and OR node (alongwith our enhancements) is presented in the next section. The algorithm is described above illustrated briefly in listing 1. For further details, the interested reader is referred to [23].

This algorithm guarantees optimal-cost solutions. Note that the best sub-solution of a input node may not necessarily end up as part of the final solution. Thus, local optimal solutions are avoided if they are not globally optimal. Finally, at the primary outputs, the best solution in terms of the cost function is selected. This specifies a domino circuit that implements the input network logic with minimum cost. The cost function in the above algorithm has been taken to be the total number of transistors in the implementation, but this may also be modified to minimize the maximum number of levels from primary inputs to primary outputs, in order to reduce the maximum input-to-output delay of a domino implementation.

This algorithm is easily illustrated with the help of an example. Consider the circuit in figure 3, and assume that the maximum number of transistors allowed in series and in parallel are 4. This simple circuit consists of 2 AND nodes and 1 OR node. The AND nodes are driven by the primary inputs, which have only one possible tuple associated with them : $\{1,\ 1,\ 1\}$. These can be combined

[1] A *unate* network is one where all signals transitions occur in one direction only, either high-to-low or low-to-high. For domino logic to function correctly, all inputs to a domino gate can only make a single low-to-high transition during the evaluate cycle. Hence only unate functions can be mapped to domino logic.

[2] An *n-tuple* is simply a set of n ordered elements. In [23], 3-tuples are used as explained in the text, in our work we associate 6-tuples with intermediate solutions as explained in the following sections.

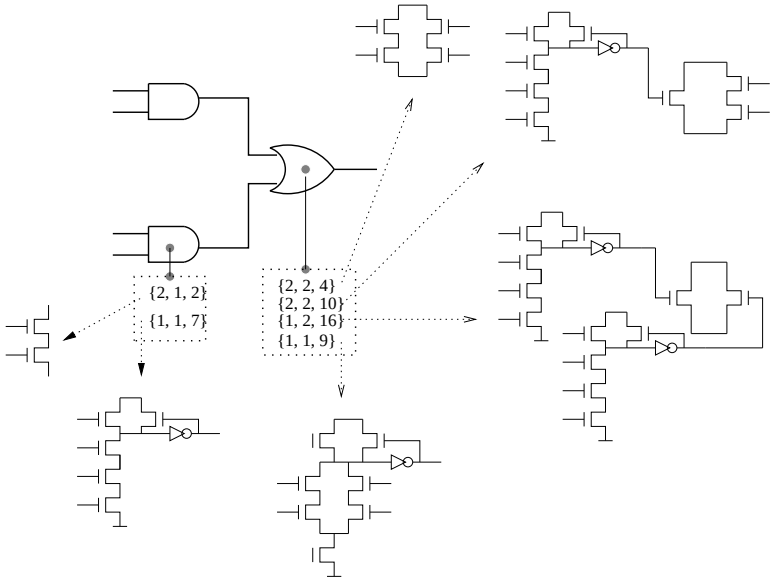


Fig. 3. Technology Mapping for Domino Logic

Algorithm 1 Technology Mapping for Domino Circuits

{Process each node in topological order from inputs to outputs}
for each node n whose inputs have been processed **do**
  for each $\{W, H\}$ configuration of the inputs **do**
    if n is OR **then**
      $\{W_{new}, H_{new}\} = \{W_1 + W_2, max(H_1, H_2)\}$
    end if
    if n is AND **then**
      $\{W_{new}, H_{new}\} = \{max(W_1, W_2), H_1 + H_2\}$
    end if
    if $\{W_{new}, H_{new}\}$ is a valid configuration **then**
      $cost_{new} = cost_1 + cost_2$
    end if
    if $cost_{new}$ is better than the original cost **then**
      update $cost$ for configuration $\{W_{new}, H_{new}\}$
    end if
  end for
  {1, 1} = convert configuration with lowest cost into a gate
end for

in an AND operation to form the tuple {2, 1, 2}, for which the transistor structure is as shown. Since there is only one tuple for this gate, it is used to construct the tuple corresponding to $W = 1, H = 1$, $\{1, 1, 7\}$. The two solutions for each of the AND nodes can be combined in 4 possible ways, but due to symmetry we have only three unique combinations - {1, 2, 16}, {2, 1, 10} (repeated twice) and {2, 2, 4}. Note that when a *gate* from an input node is used (corresponding to the {1, 1} solution), an extra transistor is needed in the next level. For the OR node, the {2, 2} solution is clearly the best, and it is used to form the corresponding {1, 1} solution, with a cost of 9.

We use this basic approach in our algorithm with modifications to the cost function calculation in order to properly account for the PBE.

V. An Algorithm for SOI Mapping

We follow the basic algorithmic framework of [23], presented in brief in section IV. As before, each node in the input network is associated with a set of tuples corresponding to one $\{W, H\}$ solution of the subtree rooted at the current node. W and H represent the width and height of the pull down network of the domino gate; the maximum values are user-specified.

Our objective in this work is to reduce the number of discharge transistors required to avoid PBE. An area objective (in terms of number of transistors) follows logically from this as the cost function to minimize. Hence we choose our initial cost associated with each tuple to be the number of transistors required to implement the logic correctly, as well as to avoid PBE. A delay objective can also be used as the cost function; in this case the actual cost function is a combination of delay and number of discharge transistors used.

In addition to the cost associated with each tuple, we also store p_{dis}, the number of *potential discharge*

transistors required by the configuration, and par_b, which tracks whether or not a given tuple has a parallel branch at the bottom of its structure. The potential discharge transistor count is used to guide tuple combination and gate formation. Depending on the actual operation performed (i.e., AND or OR), p_{dis} is converted to actual discharge transistors, else its value is propagated to the next level. Since OR is the only operation that introduces parallel stacks, par_b is set to true in an OR operation and is propagated in an AND operation depending on the combination of input tuples (this will be explained in greater detail shortly). As mentioned in the previous section, the solutions of the input gates are combined to form the solutions for the current node. In case of multiple solutions being available, the lowest cost solution is selected. Ties for the lowest cost solutions are resolved by the p_{dis} values.

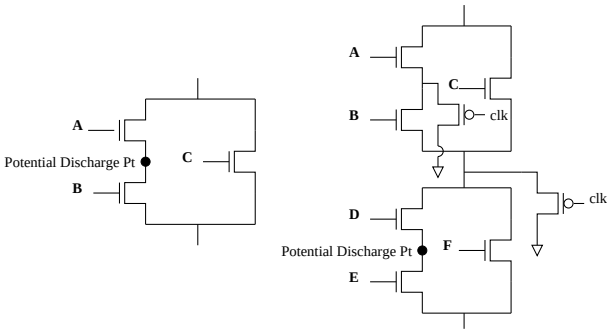


Fig. 4. Potential Discharge Points and p-discharge Transistors

We now explain the concepts of p_{dis} and par_b, which are central to our algorithm. The parameter p_{dis} is used to account for the discharge transistors that will have to be added to eliminate the PBE. From the explanation of section III-B, we see that the PBE can be excited only in the presence of one or more parallel stacks. This provides a path for the bottom of the stack to get charged to a high value (the top of the stack is charged via a path from the precharge transistor). Additionally, at least one transistor is required beneath a parallel stack to excite the PBE; when this transistor switches on, the common node for the stack will be pulled low, possibly resulting in the PBE. Hence, the bottom of a parallel stack is one potential discharge point. The parameter par_b keeps track of whether a given intermediate structure has a parallel branch at the bottom or not. In the final solution, if this point is connected to ground, no discharge transistors are required. On the other hand if it is not connected to ground, all intermediate points as specified by p_{dis} will have to be discharged. Hence, in an OR operation, we set par_b to true to account for the presence of a parallel stack. For an AND operation, it is set to the value of the tuple being placed at the bottom of the stack. In addition, we conditionally increment p_{dis} by one for an AND operation, since the intermediate point in a series stack may have to be discharged. In figure 4(a), the series connection of $A * B$ has introduced an intermediate discharge point. If $A*B$ were converted to a domino gate, or combined with other transistors in series, there would be no need to discharge this point. However, if it is connected in parallel with another configuration (as shown in the figure), this point becomes a potential discharge point for the OR tuple too - which will have to be discharged if the OR configuration is not connected directly to ground. Intermediate points in OR structures have to be discharged because of the following possible scenario. When A = 0, B = C = 1, there is a path form the top of the stack to the drain of transistor A. The source and drain of transistor A can now potentially go *high*, causing the body voltage of A to increase and thus leading to PBE.

Now consider a more complex case. Let us assume that two structures of the form shown in figure 4(a) have to be ANDed together - $A * B + C$ and $D * E + F$. Each of them has 1 potential discharge point, at the junction of A and B, and D and E. The AND operation will introduce one more potential discharge point. However, when these two parallel stacks are connected in series, the structure on the top will never be connected to ground. Hence, its potential discharge points always have to be discharged by the addition of p-discharge transistors. In addition, the intermediate point introduced by the AND operation also has to be discharged. This is shown in figure 4(b). To sum up, for an AND operation we need to perform the following computation -

$$p_{dis} = p_{dis^{bottom}};$$
$$\text{discharge transistors} = p_{dis^{top}} + 1;$$
$$cost = cost_{bottom} + cost_{top} + \text{ discharge transistors };$$
$$par_b = par_{b^{bottom}};$$

Note here that the $cost$ of a particular tuple includes not only the cost of implementing the logic, but also the discharge transistors required for avoiding PBE. Thus, when we select a lowest cost solution from various available solutions, we obtain an implementation that minimizes the cost while simultaneously avoiding PBE. The cost may be area, measured in terms of the total number of transistors required to correctly implement the required functionality, or the delay, in terms of the number of levels traversed by an input signal.

This leads to another interesting optimization that is used in our algorithm. Since our aim is to minimize the cost of the implementation as well as the total number of discharge transistors used, we can use the information implicit in p_{dis} and par_b to determine which input tuple

Algorithm 2 Algorithm for Mapping SOI Circuits

for each node n whose inputs have been processed **do**
  if n is OR **then**
    `combine_or(inputs)` ;
  end if
  if n is AND **then**
    `combine_and(inputs)` ;
  end if
  if multiple tuples obtained for the same W, H **then**
    Select tuple with lowest cost
    if costs are equal **then**
      Select tuple with lowest p_{dis}
    end if
  end if
  `create_domino_gate`
end for

`combine_or`
$W = W_{input_1} + W_{input_2}$;
$H = \max(H_{input_1}, H_{input_2})$;
$cost = cost_{input_1} + cost_{input_2}$;
$p_{dis} = p_{dis^{input_1}} + p_{dis^{input_2}}$;
$par_b = true$;

`combine_and`
if $par_{b^{input_1}}$ && $par_{b^{input_2}}$ **then**
  $top = \min(p_{dis^{input_1}}, p_{dis^{input_2}})$;
  $bottom = \max(p_{dis^{input_1}}, p_{dis^{input_2}})$;
else
  top = input with $(par_b == false)$;
end if
$W = \max(W_{top}, W_{bottom})$;
$H = H_{top} + H_{bottom}$;
$total\ dis\ trans. = p_{dis^{top}} + 1$;
$cost = cost_{top} + cost_{bottom} + total\ dis\ trans$;
$p_{dis} = p_{dis^{bottom}}$;
$par_b = par_{b^{bottom}}$;

`create_domino_gate`
Select tuple with lowest cost
Add p-clock transistor, output inverter
and feedback transistor
if tuple has primary inputs **then**
  Add n-clock transistor
end if

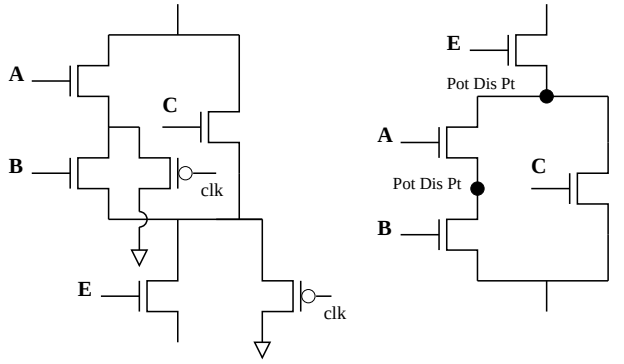


Fig. 5. Switching Transistor Stacks, with Potential Discharge Points Highlighted

is on the top in the series connection and which is on the bottom. If only one input has a parallel branch, we place this at the bottom, in the assumption that it could potentially be connected to ground. However, if both inputs have $par_b == true$, i.e., both inputs have parallel branches, the tuple order is determined by p_{dis}. We select the tuple with the larger p_{dis} to be at the bottom of the stack since this introduces fewer discharge transistors (ofcourse, all of these calculations are made under the optimistic assumption that the bottom of this stack could potentially be connected directly to ground. If this does not happen, the ordering of parallel stacks in series is irrelevant). Consider the circuit shown in figure 5, wherein $A * B + C$ is to be ANDed with E. If the structure on the left is used (with E at the bottom), we have to add two discharge transistors. However, if the circuit on the right is used (with E on the top, and the parallel stack on the bottom), we have two potential discharge points, but no immediate discharge transistors. If this structure is then connected to ground, the potential discharge points will not have to be discharged, as explained previously. Note that this switching of stacks can also be done for ciruits mapped for regular bulk CMOS. As we will show in the results section, using this technique as a stand-alone optimization is not as effective as our algorithm.

For an OR operation, we only need to add the p_{dis} values of the input tuples, and set the par_b parameter to $true$:

$$p_{dis} = p_{dis^{input_1}} + p_{dis^{input_2}};$$
$$cost = cost_{input_1} + cost_{input_2};$$
$$par_b = true;$$

Note that though the p_{dis} seems to function in an identical manner, for OR and AND structures, their interpretation is quite different. In both cases, p_{dis} refers to the number of points that must potentially be discharged.

TABLE I

COMPARISON OF DOMINO_MAP AND REARRANGE_STACKS_MAP

Circuit	Domino_Map			RS_Map			Reduction in T_{disch}		Total Reduction	
	T_{logic}	T_{disch}	T_{total}	T_{logic}	T_{disch}	T_{total}	ΔT_{disch}	%	ΔT_{total}	%
cm150	73	19	92	73	15	88	4	21.05	4	4.35
mux	73	21	94	73	18	91	3	14.29	3	3.19
z4ml	127	16	143	127	12	139	4	25.00	4	2.80
cordic	199	38	237	202	23	225	15	39.47	12	5.06
frg1	244	78	322	239	43	282	35	44.87	40	12.42
b9	365	87	452	367	57	424	30	34.48	28	6.19
apex7	663	124	787	662	106	768	18	14.52	19	2.41
c432	655	167	822	675	128	803	39	23.35	19	2.31
c880	1163	198	1361	1182	153	1335	45	22.73	26	1.91
t481	1448	232	1680	1458	193	1651	39	16.81	29	1.73
c1355	1856	130	1986	1856	86	1942	44	33.85	44	2.22
apex6	1889	319	2208	1896	275	2171	44	13.79	37	1.68
c1908	1924	208	2132	1924	171	2095	37	17.79	37	1.74
k2	2425	345	2770	2441	278	2719	67	19.42	51	1.84
c2670	2467	422	2889	2481	341	2822	81	19.19	67	2.32
c5315	5498	830	6328	5510	603	6113	227	27.35	215	3.40
c7552	8088	1082	9170	8138	760	8898	322	29.76	272	2.97
des	9069	1416	10485	9097	929	10026	487	34.39	459	4.38
Average								25.41		3.44

TABLE II

COMPARISON OF DOMINO_MAP AND SOI_DOMINO_MAP

Circuit	Domino_Map			SOI_Domino_Map			Reduction in T_{disch}		Total Reduction	
	T_{logic}	T_{disch}	T_{total}	T_{logic}	T_{disch}	T_{total}	ΔT_{disch}	%	ΔT_{total}	%
cm150	73	19	92	73	15	88	4	21.05	4	4.35
mux	73	21	94	73	15	88	6	28.57	6	6.38
z4ml	127	16	143	127	12	139	4	25.00	4	2.80
cordic	199	38	237	206	18	224	20	52.63	13	5.49
frg1	244	78	322	245	20	265	58	74.36	57	17.70
f51m	297	71	368	309	31	340	40	56.34	28	7.61
count	333	71	404	365	22	387	49	69.01	17	4.21
b9	365	87	452	367	29	396	58	66.67	56	12.39
9symml	424	107	531	440	39	479	68	63.55	52	9.79
apex7	663	124	787	667	59	726	65	52.42	61	7.75
c432	655	167	822	706	99	805	68	40.72	17	2.07
c880	1163	198	1361	1223	81	1304	117	59.09	57	4.19
t481	1448	232	1680	1495	54	1549	178	76.72	131	7.80
c1355	1856	130	1986	1856	46	1902	84	64.62	84	4.23
apex6	1889	319	2208	1928	183	2111	136	42.63	97	4.39
c1908	1924	208	2132	1949	109	2058	99	47.60	74	3.47
k2	2446	348	2794	2527	114	2641	234	67.24	153	5.48
c2670	2467	422	2889	2498	244	2742	178	42.18	147	5.09
c5315	5498	830	6328	5510	474	5984	356	42.89	344	5.44
c7552	8088	1082	9170	8164	637	8801	445	41.13	369	4.02
des	9069	1416	10485	9122	581	9703	835	58.97	782	7.46
Average								53.00		6.29

However, in case of an AND, these points will have to be discharged only in case of an OR operation, for OR they will have to be discharged only if the stack is not directly connected to ground.

The algorithm is presented in listing 2. Each node is processed in topological order, from primary inputs to primary outputs. This ensures that the inputs of the current node being processed have been processed previously, and the corresponding sub-solutions for the inputs are available. We then combine the inputs of the node being processed in functions `combine_or` or `combine_and`, depending on the functionality of the node. These functions carry out the calculations presented previously. In addition, `combine_and` also determines the order of its inputs in the series stack as a function of the input values of par_b and p_{dis}. For multiple solutions for a given $\{W, H\}$ pair, we select the tuple with the lowest cost (which includes the number of discharge transistors). Ties on cost are split according to the value of p_{dis}.

A final comment on the algorithm is that we need to maintain two costs for each tuple. The first specifies the optimal cost if the partial structure is connected to ground, and the second if it is not. At the time of gate formation, the appropriate value is used in determining the optimal cost. For convinience, these details are omitted in the pseudocode in listing 2.

This algorithm is an example of a dynamic programming approach to solving an optimization problem. Each node stores all possible solutions, with associated costs, as defined by the cost function. Locally optimal solutions need not be part of a globally optimal solution, however by enumerating all possible solutions at each node we are guaranteed an optimal solution at the output. This assertion holds for all cost function that are monotonic increasing as we proceed from inputs to outputs. The cost functions that we address in this work are area and delay metrics, which also include the number of discharge transistors have this property, and hence the final solution obtained is optimal.

VI. Results

The algorithm presented in section V has been implemented in C++ and has been tested on ISCAS benchmark circuits. In all cases, we chose the maximum width and height of the pull down network of a domino gate to be 5 and 8 respectively. Such a large value for a pull down network is valid for SOI due to the reduced source and drain capacitances. Since SOI has lower source/drain capacitances than bulk CMOS, charge sharing is not a major problem and can be handled by the weak pullup. By adding at most one p-discharge transistor at each node, we minimize its detrimental effect on charge sharing. While circuit performance does degrade slightly when compared with having a precharge transistor, this is a minor cost to pay to avoid circuit malfunction[3]. For comparison purposes, we have implemented a bulk CMOS mapping algorithm that maps circuits without regard to potential discharge points. This algorithm is referred to as `Domino_Map`. p-discharge transistors are added in a post-processing step. We compare this solution with different approaches to mapping circuits for SOI, including our algorithm with area and delay cost functions.

A. *Rearrange_Stacks_Map*

The first three columns next to each circuit name in table I show the cost associated with the solution obtained from `Domino_Map`, specifically listing the total number of domino transistors (T_{logic}), the number of pmos discharge transistors added (T_{disch}) and the sum of these two, which is the total number of transistors(T_{total}). We then ran our algorithm without regard to potential discharge points as in `Domino_Map`, but added a post-processing step that rearranges series stacks (generated by AND operations) so as to move parallel sections with a large number of potential discharge points closer to ground. The reasons for doing this have been discussed in the previous section. The solution obtained is listed in the columns under `RS_Map`. We found an average reduction of 25.4% in the number p-discharge transistors, and a 3.44% reduction in the total number of transistors. As can be seen, simply re-ordering transistor stacks leads to some decrease in the number of discharge transistors.

B. *SOI_Domino_Map*

The results of applying algorithm `SOI_Domino_Map` of listing 2 to the benchmark circuits are presented in table II. Comparing the results obtained from `Domino_Map` and `SOI_Domino_Map`, it is clear that though the number of domino logic transistors required in SOI may be more, this increase is more than compensated by the fewer number of p-discharge transistors required, thus saving on the total number of transistors used. The average reduction in the number of discharge transistors is 53%. The last two columns list the reduction in the *total* number of transistors required for the implementation. We obtain an average reduction of 6.29%, even though the number of logic transistors (without p-discharge) has increased.

Thus, while a simple reordering of series stacks does result in some cost benefit, it is still only half the reduction of our algorithm.

[3]Another option to avoiding the detrimental affect of the p-discharge transistors is to remap the logic, avoiding PBE-inducing structures. However, this will result in a larger number of domino gate levels, leading to an even larger delay.

TABLE III

COMPARISON OF THE NUMBER OF TRANSISTORS UNDER DIFFERENT WEIGHTS OF p_{dis}

Circuit	k = 1					k = 5					%Improv
	T_{logic}	T_{disch}	T_{total}	$\#G$	T_{clock}	T_{logic}	T_{disch}	T_{total}	$\#G$	T_{clock}	
cm150	73	15	88	3	21	73	15	88	3	21	0.00
mux	73	15	88	3	21	73	15	88	3	21	0.00
z4ml	134	13	147	9	39	134	13	147	9	39	0.00
cordic	222	19	241	14	52	217	19	236	13	51	1.92
frg1	283	20	303	19	58	277	21	298	18	57	1.72
count	374	22	396	28	77	374	22	396	28	77	0.00
b9	367	29	396	29	87	373	26	399	30	86	0.11
c8	331	42	373	26	94	325	42	367	25	92	2.12
f51m	405	42	4 47	27	104	391	38	429	26	98	5.76
9symml	571	57	628	34	132	482	36	518	33	106	19.69
apex7	739	67	806	54	175	733	67	800	53	173	1.14
x1	825	63	888	65	193	816	60	876	64	188	2.59
c432	799	93	892	52	197	804	89	893	53	194	1.52
i6	1155	67	1222	67	201	1155	67	1222	67	201	0.00
c1908	992	117	1109	77	259	957	111	1068	78	254	1.93
t481	1916	77	1993	132	325	1927	70	1997	135	316	2.77
c499	2016	46	2062	130	440	2016	46	2062	130	440	0.00
c1355	2016	46	2062	130	440	2016	46	2062	130	440	0.00
dalu	2073	182	2255	158	446	2065	177	2242	158	441	1.12
k2	3127	109	3236	195	481	3142	107	3249	195	475	1.24
apex6	2418	206	2624	158	520	2516	185	2701	160	504	3.07
rot	2520	290	2810	174	627	2449	262	2711	172	595	5.10
c2670	2608	247	2855	162	642	2614	244	2858	163	641	0.15
C5315	5755	535	6290	433	1501	5754	515	6269	439	1491	0.66
c3540	6659	634	7293	427	1501	6377	552	6929	412	1393	7.93
des	9818	600	10418	594	1581	9390	493	9883	586	1453	8.09
c7552	7519	584	8103	582	1853	7376	508	7884	580	1759	5.07
Average											3.82

C. Penalizing Clock Connected Transistors

Realizing the effects of loading on the clock network, we then applied algorithm `SOI_Domino_Map` to the same circuits, assigning a cost for the clock-driven transistors that is k times the cost of a regular transistor, where k is a user specified value. The clock connected transistors include p-clock and n-clock transistors in the domino gates along with the p-discharge transistors. On the one hand, the effect of including the cost of the p-clock and n-clock transistors of the gate is to make gate formation operation more expensive (the cost of the $\{1, 1\}$ solution for each tuple makes it less likely to be selected), and the algorithm prefers to include as many transistors in each pull down network as possible. Incrementing the cost of the p-discharge transistors, on the other hand, pushes the algorithm towards forming domino gates early, so as to avoid the overhead of the p-discharge transistors. As the results show, our algorithm chooses a result balanced between these extremes, and as the cost of clock driven transistors is increased, the solutions reduce the number of gates *and* p-discharge transistors, along with an increase in the total number of transistors required for the implementation[4]. The columns labelled T_{clock} is the number of transistors connected to the clock network. This figure is obtained by adding the number of p-discharge transistors to the clock transistors in the domino gates. The last column shows the percentage reduction in the number of clock-driven transistors, on average we reduce this figure by 3.82%. An interesting observation is that the number of clock connected transistors does not change significantly as k is varied. Also note that not much improvement is obtained for circuits that have a relatively small number of

[4]The figures in table III represent the number of transistors, not their weighted cost

clock connected transistors. This is to be expected, since there is less freedom to choose between different solutions. Larger circuits, on the other hand, provide the algorithm with a greater number of options, and we obtain more improvement for these.

D. Depth Optimization

We now address the minimization of the *delay* of an implementation. As an approximation of the delay, we set the cost function to be the depth, i.e., the maximum number of levels of domino gates that a signal passes through from primary inputs to primary outputs. A more accurate delay model would require characterization of every possible pulldown network, and this is not feasable for a libraryless approach. As before, `Domino_Map`, reduces the number of levels required for an implementation, and discharge transistors are added as a post-processing step. In `SOI_Domino_Map`, the number of discharge transistors needed is included as a part of the cost. The results of running each of these algorithms are as presented in table IV. The second column next to each circuit name shows the maximum number of 2-input AND/OR gates in the original network that a signal passes through, from primary inputs to primary outputs. The columns under `Domino_Map` list the number of transistors required for implementing the logic functionality, the number of discharge transistors added in the post-processing phase, the total cost of the solution, and the number of levels in the solution. The corresponding columns under `SOI_Domino_Map` are similiarly calculated using the new cost function, so that the discharge transistors are included during the mapping phase. Reducing the number of levels in an implementation drives a solution towards favouring complex gates. As in the previous section, trying to reduce the number of discharge transistors drives the solution in the opposite direction, i.e., towards smaller gates. As can be seen from the results, algorithm `SOI_Domino_Map` reduces the number of levels for a few circuits, and increases them for others, in comparison with `Domino_Map`. The key result, though, is that the *sum* of number of levels and discharge transistors is reduced. We obtain an average reduction of 49.76% in discharge transistors required, and a 6.36% reduction in the number of levels.

VII. Conclusion and Future Work

We have presented an algorithm that maps gates in a logic network to a domino implementation suitable for use in SOI circuits. As the results in section VI show, the lowest cost solution for domino mapping in bulk silicon technology is not an effective solution in the context of SOI. Our algorithm minimizes a specified cost function, which includes the discharge transistors required. This cost function can be as area cost or a delay cost. We also show how we can apply the algorithm by skewing the cost of clock transistors in order to reduce the load on the clock network. A similar approach can be used to derive a solution with as few gates as possible, by increasing the relative cost of gate formation. In fact, this technique can be applied to mapping for bulk CMOS too, in order to reduce the load on the clock network.

A further improvement is from the observation that the n-clock transistors are required only for gates that have primary inputs. Gates whose inputs come from other domino gates do not need these transistors, since these inputs are always low during precharge, and footless domino may be used instead. Hence, the pull down network will never be switched on during precharge. The big advantage of using this scheme for SOI is that even if a parallel stack has potential discharge points, if these are connected to ground it is not necessary to actually discharge them.

Our mapping algorithm assumes the worst case scenario, in which particular structures susceptible to breakdown have to be discharged correctly. However, breakdown will only occur for a particular sequence of input logic values. We have not taken this into account in our algorithm, and incorporating this information could lead to better solutions. In fact, this could be used for solving the hysteresis effect [21] in SOI too. Once a transistor netlist has been obtained from the logic level description of the circuit as presented in this paper, a followup technology-specific optimization step can be used to obtain further delay improvements.

References

[1] P.-F. Lu, C.-T. Chuang, J. Ji, L. F. Wagner, C.-M. Hsieh, J. B. Kuang, L. L.-C. Hsu, J. Mario M. Pelella, S.-F. S. Chu, and C. J. Anderson, "Floating-Body Effects in Partially Depleted SOI CMOS Circuits," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 8, pp. 1241–1253, Aug. 1997.

[2] C. T. Chuang, "Design Considerations of SOI Digital CMOS VLSI," in *Proceedings of the Proceedings of the 1998 IEEE International SOI Conference*, 1998, pp. 5–8.

[3] S. K. Karandikar and S. S. Sapatnekar, "Technology Mapping for SOI Domino Logic Incorporating Solutions for the Parasitic Bipolar Effect," in *Proceedings of the IEEE/ACM Design Automation Conference*, 2001, pp. 377–382.

[4] D. Allen, D. Behrends, and B. Stanisic, "Converting a 64b PowerPC Processor from CMOS Bulk to SOI Technology," in *Proceedings of the IEEE/ACM Design Automation Conference*, 1999, pp. 892–896.

[5] K. Bernstein and N. J. Rohrer, *SOI Circuit Design Concepts*. Boston, MA: Kluwer Academic Publishers, 2000.

[6] J. B. Kuo and K.-W. Su, *CMOS VLSI Engineering Silicon-on-Insulator (SOI)*. Boston, MA: Kluwer Academic Publishers, 1998.

[7] C. T. Chuang and R. Puri, "SOI Digital CMOS VLSI - a Design Perspective," in *Proceedings of the IEEE/ACM Design Automation Conference*, 1999, pp. 709–714.

TABLE IV

DEPTH AND DISCHARGE TRANSISTOR OPTIMIZATION

Circuit	L	Domino_Map				SOI_Domino_Map				Reduction in T_{disch}		Reduction in L	
		T_{logic}	T_{disch}	T_{total}	L	T_{logic}	T_{disch}	T_{total}	L	ΔT_{disch}	%	ΔL	%
z4ml	16	182	22	204	7	176	12	188	6	10	45.45	1	14.29
cm150	10	268	35	303	9	193	20	213	7	15	42.86	2	22.22
mux	10	268	35	303	9	193	19	212	7	16	45.71	2	22.22
cordic	12	373	40	413	9	310	19	329	8	21	52.50	1	11.11
f51m	30	534	75	609	25	598	49	647	20	26	34.67	5	20.00
c8	11	591	80	671	6	564	44	608	6	36	45.00	0	0.00
frg1	14	607	102	709	12	503	52	555	11	50	49.02	1	8.33
b9	10	659	106	765	9	537	47	584	6	59	55.66	3	33.33
count	21	741	76	817	7	672	56	728	9	20	26.32	-2	-28.57
c432	34	981	125	1106	26	1229	107	1336	25	18	14.40	1	3.85
apex7	17	974	139	1113	11	1111	82	1193	7	57	41.01	4	36.36
9symml	21	1038	174	1212	14	800	70	870	12	104	59.77	2	14.29
c1908	32	1292	251	1543	16	1625	167	1792	14	84	33.47	2	12.50
x1	12	1490	233	1723	9	1364	106	1470	8	127	54.51	1	11.11
i6	6	2109	237	2346	4	2143	133	2276	4	104	43.88	0	0.00
c1355	20	2640	244	2884	7	2456	44	2500	7	200	81.97	0	0.00
t481	23	2794	196	2990	17	3301	97	3398	16	99	50.51	1	5.88
rot	27	2768	514	3282	11	3259	320	3579	14	194	37.74	-3	-27.27
apex6	21	3816	584	4400	15	4222	315	4537	12	269	46.06	3	20.00
k2	21	4181	324	4505	13	3847	143	3990	12	181	55.86	1	7.69
c2670	31	4052	521	4573	16	4207	281	4488	14	240	46.07	2	12.50
dalu	23	3795	786	4581	10	2747	249	2996	12	537	68.32	-2	-20.00
c3540	42	7675	1341	9016	19	9021	601	9622	20	740	55.18	-1	-5.26
c5315	36	8216	1074	9290	17	9409	493	9902	17	581	54.10	0	0.00
c7552	42	10374	1172	11546	29	10747	501	11248	22	671	57.25	7	24.14
des	26	14068	2653	16721	14	21313	944	22257	14	1709	64.42	0	0.00
Average											49.76		6.36

[8] D. A. Antoniadis, "SOI CMOS as a Mainstream Low-Power Technology: A Critical Assessment," in *Proceedings of the Proceedings of the IEEE/ACM International Symposium on Low Power Electronics and Design*, 1997, pp. 295–300.

[9] Y.-C. Tseng, S. C. Chin, and J. C. S. Woo, "The Impact of SOI MOSFETs on Low Power Digital Circuits," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 1997, pp. 243–246.

[10] W. Jin, P. C. H. Chan, and M. Chan, "On the Power Dissipation in Dynamic Threshold Silicon-on-Insulator CMOS Inverter," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 1997, pp. 247–250.

[11] D. H. Allen, A. G. Aipperspach, D. T. Cox, N. V. Phan, and S. N. Storino, "A 0.2um 1.8V SOI 550MHz 64b PowerPC Microprocessor with Copper Interconnects," in *Proceedings of the IEEE International Solid-State Circuits Conference*, Feb. 1999, pp. 438–439.

[12] T. C. Buchholtz, A. G. Aipperspach, D. T. Cox, N. V. Phan, S. N. Storino, J. D. Strom, and R. R. Williams, "A 660MHz 64b SOI Processor with Cu Interconnects," in *Proceedings of the IEEE International Solid-State Circuits Conference*, 2000, pp. 88–89.

[13] Hewlett-Packard. (2000) PA-RISC 8x00 Family of Microprocessors With Focus on PA-8700. [Online]. Available: http://www.cpus.hp.com/techreports/PA-8700wp.pdf

[14] Y. W. Kim, S. B. Park, Y. G. Ko, K. I. Kim, I. K. Kim, K. J. Bae, K. W. Lee, J. O. Yu, U. Chung, and K. P. Suh, "A 0.25um 600MHz 1.5V SOI 64b ALPHA Microprocessor," in *Proceedings of the IEEE International Solid-State Circuits Conference*, 1999, pp. 432–433.

[15] M. Canada, C. Akrout, D. Cawthron, J. Corr, S. Geissler, R. Houle, P. Kartschoke, D. Kramer, P. McCormick, N. Rohrer, G. Salem, and L. Warriner, "A 580MHz RISC Microprocessor in SOI," in *Proceedings of the IEEE International Solid-State Circuits Conference*, 1999, pp. 430–431.

[16] D. Stasieak, J. Tran, F. Mounes-Toussi, and S. Storino, "A 2nd generation 440ps SOI 64b Adder," in *Proceedings of the IEEE International Solid-State Circuits Conference*, 2000, pp. 288–289.

[17] J. M. Hill and J. Lachman, "A 900MHz 2.25MB Cache with On-Chip CPU - Now in Cu SOI," in *Proceedings of the IEEE International Solid-State Circuits Conference*, 2001, pp. 176–177.

[18] S. Mathew, R. Krishnamurthy, M. Anders, R. Rios, K. Mistry, and K. Soumyanath, "Sub-500ps 64b ALUs in 0.18um SOI/Bulk CMOS: Design and Scaling Trends," in *Proceedings of the IEEE International Solid-State Circuits Conference*, 2001, pp. 318–319.

[19] A. Wei, M. J. Sherony, and D. A. Antoniadis, "Effect of Floating-Body Charge on SOI MOSFET Design," *IEEE Transactions on Electron Devices*, vol. 45, no. 2, pp. 430–438, Feb. 1998.

[20] R. H. Krambeck, C. M. Lee, and H.-F. S. Law, "High Speed Com-

pact Circuits with CMOS," *IEEE Journal of Solid-State Circuits*, vol. 17, no. 3, pp. 614–619, June 1982.

[21] R. Puri and C. T. Chuang, "Histeresis Effect in Pass-Transistor Based Partially-Depleted SOI CMOS Circuits," in *Proceedings of the Proceedings of the 1998 IEEE International SOI Conference*, 1998, pp. 103–104.

[22] R. Puri, A. Bjorksten, and T. E. Rosser, "Logic Optimization by Output Phase Assignment in Dynamic Logic Synthesis," in *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, 1996, pp. 2–6.

[23] M. Zhao and S. S. Sapatnekar, "Technology Mapping for Domino Logic," in *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, 1998, pp. 248–251.

[24] J. L. Burns and J. A. Feldman, "C5M - A Control-Logic Layout Synthesis System for High-Performance Microprocessors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 1, pp. 14–23, Jan. 1998.

Shrirang K. Karandikar received the B.E. degree from the University of Pune in 1994, and the M. S. degree in Clarkson University in 1996. He worked with Intel's Logic and Validation Technology group from 1997 to 1999. He is currently pursuing his Ph.D. at the Department of Electrical and Computer Engineering at the University of Minnesota.

Sachin S. Sapatnekar received the B.Tech. degree from the Indian Institute of Technology, Bombay in 1987, the M.S. degree from Syracuse University in 1989, and the Ph.D. degree from the University of Illinois at Urbana-Champaign in 1992. From 1992 to 1997, he was an assistant professor in the Department of Electrical and Computer Engineering at Iowa State University. He is currently a Professor in the Department of Electrical and Computer Engineering at the University of Minnesota.

He has coauthored two books, "Timing Analysis and Optimization of Sequential Circuits," and "Design Automation for Timing-Driven Layout Synthesis," and is a co-editor of a volume, "Layout Optimizations in VLSI Designs," all published by Kluwer. He has been an Associate Editor for the IEEE Transactions on VLSI Systems, the IEEE Transactions on CAD, and the IEEE Transactions on Circuits and Systems II, has served on the Technical Program Committee for various conferences, as Technical Program and General Chair for the Tau workshop and the International Symposium on Physical Design. He is currently a Distinguished Visitor for the IEEE Computer Society and a Distinguished Lecturer for the IEEE Circuits and Systems Society. He is a recipient of the NSF Career Award and best paper awards at DAC 1997, ICCD 1998, and DAC 2001.