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

Srirang K. Karandikar
Department of ECE,
University of Minnesota.
srirang@ece.umn.edu

Sachin S. Sapatnekar
Department of ECE,
University of Minnesota.
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 the PBE, such as transistor reordering, altering the way transistors are organized into gates, and adding pmos discharge transistors. We minimize the total cost of implementation, which includes the discharge transistors required for correct functioning. Our algorithm generates solutions that reduce the number of discharge transistors needed by 44.23%, and reduces the size of the final solution by 11.66% on average.

1. INTRODUCTION

As the scaling of bulk CMOS proceeds along the roadmap, interest in SOI as an alternative technology has increased. Manufacturing processes have recently 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. If these are used in SOI, we would obtain circuits that would not function correctly. Current EDA techniques too do not adequately address the needs of SOI design. There is a requirement for new algorithms and tools targeted towards SOI designs. Simply modifying existing algorithms and adding post-processing steps leads to solutions that are sub-optimal. This paper address the technology mapping problem in the context of SOI. We present an algorithm that maps an arbitrary 2-input logic gate network to domino logic eliminating the "Parasitic Bipolar Effect" (PBE) 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. In doing so, the algorithm minimizes the cost of this implementation: for example, for an area objective, it would minimize the total number of transistors, including pre-discharge transistors.

This paper is organized as follows. We briefly introduce SOI and domino logic, and present problems typical to SOI implementations, with emphasis on overcoming the PBE. We then present an algorithm that performs the technology mapping taking the PBE into consideration, and show the effectiveness of this algorithm in the results section. We conclude with directions for future work.

DAC 2001, June 18-22, 2001, Las Vegas, Nevada, USA.

2. BACKGROUND

2.1 Silicon-on-Insulator

SOI has long been used in a variety of fields, such as radiation-hardened and high-voltage applications [13], [14]. Circuits implemented in SOI have attractive properties as compared to bulk CMOS, such as reduced source- and drain-substrate capacitances, no body effect in series stacks of transistors and suitability for reduced V_{dd} operation for given performance [4], [9]. Due to reduced capacitances, SOI devices consume less power. 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. This has been due to the rapid progress and scaling of bulk CMOS technology. However, recent advances in manufacturing processes 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. A number of high end microprocessor designs have recently been implemented in SOI, e.g. HP-PA 8700 [11], IBM Power PC [1], [3], [2], and others [5], [6].

The manufacturing process of SOI is very similar to that of bulk CMOS. The preliminary step in SOI fabrication 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). 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, 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 [9]. 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), described in more detail in section 3.

2.2 Domino Logic

Domino logic is a favoured approach for implementing timing critical circuits due to their high performance. The basic structure of a domino gate is as shown in figure 1(without the p-discharge transistor. 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 the n-transistors in series. More complicated logic operations are obtained by combining these basic operations. The circuit shown in figure 1 implements the logic function $(A + B + C) * D$.

3. PARASITIC BIPOLAR EFFECT IN SOI

3.1 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 [8], in which the behaviour of a transistor varies according to its previous switching history. Another serious problem associated with SOI devices is the Parasitic Bipolar Effect (PBE) [12], [7], which is described in more detail in the following sections.

3.2 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 charge high. When the source is subsequently pulled down, either by the clocked evaluation transistor in dynamic circuits or by an input signal, a large overdrive is developed across the body-source junction, causing bipolar current to flow through the lateral parasitic bipolar transistor (the emmitter and collector of the parasitic transistor correspond to the drain and source of the FET, while the base corresponds to the body). 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 [12]. For the circuit shown in figure 1(without the p-discharge transistor), 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 high voltage level. 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. Now if 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 between the body and gate of A), but leaves the body voltages of B and C 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 in devices B and C to switch on - 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 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 low, it ends up as a high. In this manner, the PBE can result in a wrong evaluation if not properly 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 behaviour in the interim.

3.3 Solutions to the PBE

There are a several solutions for handling the PBE, such as sizing up the keeper pmos device, adding body contacts to some devices in the circuit, breaking parallel stacks by transistor replication, reodering parallel stacks to reduce susceptibility to PBE, and predischarging intermediate nodes.

Stacks of transistors in a gate may be reordered to reduce its susceptibility to PBE. For the gate in Figure 1, 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 PBE. This approach exploits the reduced charge sharing effect and reduced delay dependency on stack ordering in SOI technology. However, this works only if there is one parallel stack per gate. If this condition is not met, it may be possible to remap Boolean logic 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.

Intermediate nodes in a stack may be predischarged in every clock cycle. In figure 1, 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.

One approach to performing optimizations such as those mentioned above is to start with the original design in bulk silicon, analyze it to identify potential sources of PBE, and

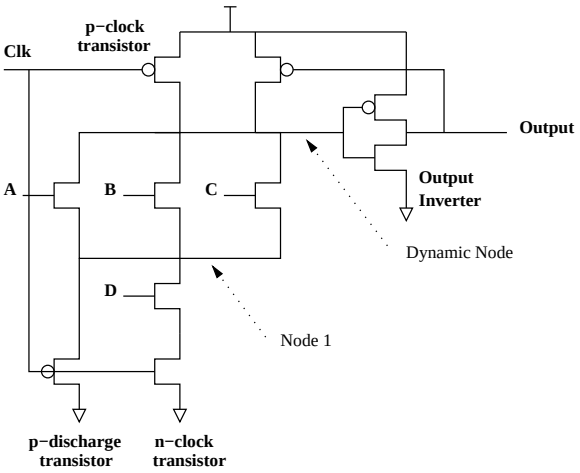

Figure 1: Predischarge Trans for Resolving PBE

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 5, we propose an algorithm that performs such a mapping.

In this work, we avoid the transformations of sizing the keeper, adding body contacts and splitting parallel stacks using duplication, since they can cause significant cost increases [13], 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 penalise the addition of clock-connected transistors and additional transistors required due to gate reorganization, since they represent a cost-increasing transformation.

4. 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.

We use a simple bubble pushing algorithm to generate the unate network. 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 the 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.

We now briefly present a library free technology mapping algorithm for domino logic initially presented in [10]. A set of tuples of $\{W, H, C\}$ (width, height and cost) are associated with each logic gate of the input network. The cost here may be the number of transistors, the number of logic levels, or 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 gates is traversed from primary inputs to primary outputs, and sub-solutions for each gates 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 $W \times H$ tuples are associated with each gate. Once all valid tuples for a gate 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 curren gate are AND and OR. These operations are as follows. An AND operation requires a series connection of inputs. Hence, the $\{W_l, H_l\}$ and $\{W_r, H_r\}$ solutions of the inputs are combined to form the $\{\max(W_l, W_r), H_l + H_r\}$ solution. Similarly, $\{W_l, H_l\}$ and $\{W_r, H_r\}$ solutions of the inputs can be combined as $\{W_l + W_r, \max(H_l, H_r)\}$.

This algorithm guarantees optimal-cost solutions. The best sub-solution of an 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 with minimum cost.

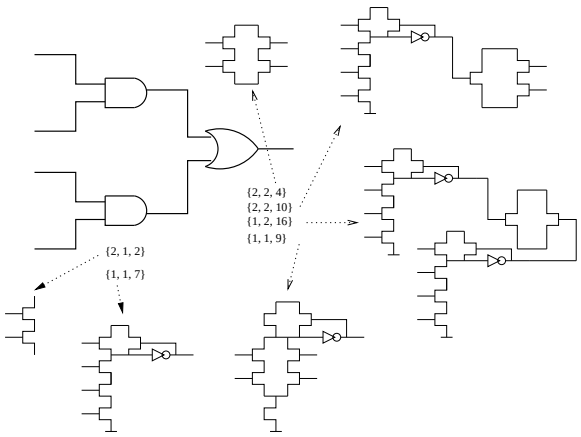

Figure 2: Technology Mapping for Domino Logic

This algorithm is easily illustrated with the help of an example. Consider the circuit in figure 2, and assume that the maximum number of transistors allowed in series and in parallel are 4. This simple circuit consists of 2 AND gates and 1 OR gate. The AND gates are driven by the primary inputs, which have only one possible tuple associated with them :

{1, 1, 1}. These can be combined in an AND operation to form the tuple {2, 1, 2}. The transistor structure associated with this tuple 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 gates 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 a input node is used (corresponding to the {1, 1} solution), an extra transistor is needed in the next gate. For the OR gate, the {2, 2} solution is clearly the best, and it is used to form the corresponding {1, 1} solution, with a cost of 9.

5. AN ALGORITHM FOR SOI MAPPING

We follow the basic algorithmic framework of [10], presented in brief in the previous section with modifications to the cost function calculation in order to properly account for PBE. As before, each gate 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. 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. As mentioned in the previous section, the solutions of the input gates are combined to form the solutions for the current gate. 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.

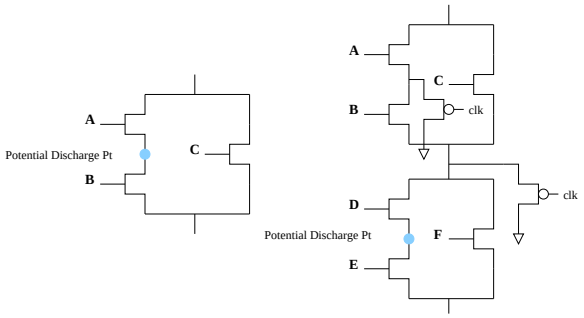

Figure 3: Potential Discharge Points

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 PBE. From the explanation of section 3.2, we see that 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 at the bottom of 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 3(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 logic gates 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.

Now consider a more complex case. Let us assume that two structures of the form shown in figure 3(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 will always need 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 3(b). Hence, for an AND operation we perform the following computation:

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

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 is on the top in a 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 (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).

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

$$p_{dis} = p_{dis^{left}} + p_{dis^{right}};$$
$$cost = cost_{left} + cost_{right};$$
$$par_b = true;$$

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 discharged. 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.

Algorithm SOI_Domino_Map

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

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

combine_and
  If $par_{b^{left}}$ && $par_{b^{right}}$
    $top = \min(p_{dis^{left}}, p_{dis^{right}})$;
    $bottom = \max(p_{dis^{left}}, p_{dis^{right}})$;
  else $top = input\ with(par_b == false)$;
  $W = \max(W_{top}, W_{bottom})$;
  $H = H_{top} + H_{bottom}$;
  $total\ dis.\ trans. = p_{dis^{top}} + 1$;
  $cost = cost_{top} + cost_{bottom} + total\ dis.\ trans.$;
  $p_{dis} = p_{dis^{bottom}}$;
  $par_b = par_{b^{bottom}}$;

Figure 4: Algorithm for Mapping SOI Circuits

The algorithm is listed in figure 4. We process each node in topological order, from primary inputs to primary outputs. This ensures that the inputs of the 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.

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.

6. RESULTS

The algorithm presented in section 5 has been implemented in C++ and results on ISCAS benchmark circuits are shown in table 1. In all cases, we chose the maximum width and height of a domino gate to be 5 and 8 respectively. Such a large value for a domino gate is valid for SOI due to reduced source and drain capacitances.

We first obtained a regular domino solution for bulk CMOS, and added p-discharge transistors in a post-processing step. The first three columns next to each circuit name in table 1 show the cost associated with this solution, specifically listing the total number of domino transistors (#tran), the number of pmos predischarge transistors added (#p-dis) and the sum of these two, which is the total number of transistors. The results of applying algorithm SOI_Domino_Map of figure 4 to the circuits are presented in the next three columns. Comparing these two sets of results, 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. As can be seen in the columns labeled #p-dis Reduction, there is a large decrease in the number of p-discharge transistors used, 44.23% on average. The last two columns list the reduction in the *total* number of transistors required for the implementation. We obtain an average reduction of 11.66%, even though the number of domino transistors (without p-discharge) has increased.

We also ran the algorithm without regard to potential discharge points as before, 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, and then add the necessary p-discharge transistors. The reasons for doing this have been discussed in the previous section. We found an average reduction of 22.5% in the number p-discharge transistors. Thus, a simple reordering of series stacks only delivers half the potential reduction as compared to our algorithm.

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 (this includes the clock transistors along with the p-discharge transistors). The motivation for doing this is to reduce the load on the clock network. Though we may end up with a solution that requires a larger number of total transistors, this could be an acceptable tradeoff since we reduce the number of transistors connected to the clock network. 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 domino gate as possible. Incrementing the cost of the p-discharge transistors, on the other hand, pushes the algorithm towards forming gates early, so as to avoid the overhead of the p-discharge transistors. As the results show, our algorithm chooses a path 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. The columns labelled $\#tran_{clock}$ is the number of transistors connected to the clock network. This figure is obtained by adding the number of p-discharge transistors to the n- and p-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 only 5.11%. An interesting observation is that the number of clock connected transistors does not change significantly as k is varied.

7. 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 6 show, the lowest cost solution for domino mapping in bulk silicon technology is not necessarily good when the circuit is to be implemented in SOI. 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.

Table 1: Comparison of Domino Mapping and SOI Domino Mapping

Circuit	Domino_Map			SOI_Domino_Map			#p-dis Reduction		#total Reduction	
	#tran	#p-dis	total	#tran	#p-dis	total	Δ p-dis	%	Δ total	%
mux	91	19	110	72	12	84	7	36.84	26	23.63
cordic	205	11	216	196	10	206	1	9.09	10	4.62
f51m	798	103	901	706	35	741	68	66.02	160	17.75
b9	432	61	493	378	28	406	33	54.09	87	17.63
frg1	278	39	317	233	20	253	19	48.71	64	20.19
c8	321	26	347	295	21	316	5	19.23	31	8.93
9symml	481	68	549	415	39	454	29	42.64	95	17.30
c432	1475	232	1707	1290	145	1435	87	37.50	272	15.93
apex7	776	57	833	708	22	730	35	61.40	103	12.36
x1	803	76	879	721	53	774	23	30.26	105	11.94
t481	1562	67	1629	1507	31	1538	36	53.73	91	5.58
rot	2360	273	2633	2093	155	2248	118	43.22	385	14.62
apex6	2753	237	2990	2518	98	2616	139	58.64	374	12.5
c2670	3252	331	3583	3276	196	3472	135	40.78	111	3.09
k2	3452	81	3533	3355	24	3379	57	70.37	154	4.35
dalu	2146	168	2314	1980	74	2054	94	55.95	260	11.23
c3540	7841	715	8556	7218	474	7692	241	32.13	864	10.09
c5315	5917	503	6420	5804	242	6046	261	51.88	374	5.82
c7552	16548	1333	17881	15766	703	16469	630	47.26	1412	7.89
des	10110	710	10820	9446	533	9979	177	24.93	841	7.72
Average								44.23		11.66

Table 2: Comparison of the Number of Transistors Under Different Weights of p_{dis}

Circuit	$k=1$					$k=5$					%Improv
	$\#tran$	$\#p-dis$	$total$	$\#G$	$\#tran_{clock}$	$\#tran$	$\#p-dis$	$total$	$\#G$	$\#tran_{clock}$	
z4ml	195	11	206	9	29	259	4	263	11	26	10.34
b9	378	28	406	30	88	422	8	430	35	78	11.36
c8	295	21	316	27	75	307	19	326	26	71	5.33
c432	1290	145	1485	53	251	1720	104	1824	58	220	12.35
x1	721	53	774	56	165	798	31	829	66	163	1.21
c880	1117	62	1179	90	242	1136	61	1197	87	235	2.89
i6	835	0	835	74	148	835	0	835	73	146	1.35
t481	1507	31	1538	128	287	1653	22	1675	125	272	5.23
rot	2093	155	2248	177	509	2398	120	2518	186	492	3.34
apex6	2518	98	2616	179	456	3112	75	3187	186	447	1.97
c2670	3276	196	3472	191	578	3910	160	4070	186	532	7.96
c5315	5804	242	6046	442	1126	6353	220	6573	439	1098	2.48
des	9446	533	9979	711	1955	10543	418	10961	763	1944	0.56

8. REFERENCES

[1] D. H. Allen et al, "A 0.2um 1.8V SOI 550MHz 64b PowerPC Microprocessor with Copper Interconnects", in *IEEE Int. Solid-State Circuits Conf.*, Feb. 1999.

[2] T. C. Buchholtz et al, "A 660MHz 64b SOI Processor with Cu Interconnects", in *IEEE Int. Solid-State Circuits Conf.*, 2000.

[3] D. Allen, D. Behrends and B. Stanisic, "Converting a 64b PowerPC Processor from CMOS Bulk to SOI Technology", in *Proc. DAC*, 1999.

[4] C. T. Chuang and R. Puri, "SOI Digital CMOS VLSI - A Design Perspective", in *Proc. DAC*, 1999.

[5] Y. W. Kim et al, "A 0.25um 600MHz 1.5V SOI 64b ALPHA Microprocessor", in *IEEE Int. Solid-State Circuits Conf.*, 1999.

[6] M. Canada et al, "A 580MHz RISC Microprocessor in SOI", in *IEEE Int. Solid-State Circuits Conf.*, 1999.

[7] C. T. Chuang, "Design Considerations of SOI Digital CMOS VLSI", in *Proc. IEEE International SOI Conf.*, 1998.

[8] R. Puri and C. T. Chuang, "Histeresis Effect in Pass-Transistor Based Partially-Depleted SOI CMOS Circuits", in *Proc. IEEE Int. SOI Conf.*, 1998.

[9] D. A. Antoniadis, "SOI CMOS as a Mainstream Low-Power Technology: A Critical Assessment", in *Proc. ISLPED*, 1997.

[10] M. Zhao and S. S. Sapatnekar, "Technology Mapping for Domino Logic", in *Proc. ICCAD*, 1998.

[11] "PA-RISC 8x00 Family of Microprocessors With Focus on PA-8700", Hewlett-Packard White Paper, in *http://www.cpus.hp.com/techreports/PA-8700wp.pdf*

[12] P.-F. Lu et al, "Floating-Body Effects in Partially Depleted SOI CMOS Circuits", in *IEEE J. Solid-State Circuits*, vol. 32, Aug. 1997.

[13] K. Bernstein and N. J. Rohrer, *SOI Circuit Design Concepts*, Kluwer Academic Publishers, 2000.

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