Exact lower bound for the number of switches in series to implement a combinational logic cell

F.R.Schneider[1] felipers@inf.ufrgs.br

R.P.Ribas[1] rpribas@inf.ufrgs.br

S.S.Sapatnekar[2] sachin@ece.umn.edu

A.I.Reis[1,2] andreis@inf.ufrgs.br

[1]*Instituto de Informática - UFRGS*
CEP 91501-970 - Caixa Postal 15064
Porto Alegre – RS - Brasil
+55 51 3316 6810

[2]*Department of Electrical and Computer Engineering*
200 Union Street SE, University of Minnesota
Minneapolis, MN 55455.
+1 (612) 625-0025

ABSTRACT

This paper addresses the question of how many serial switches are necessary to implement a given logic function as a switch network. This issue is important because it affects directly the resistance that will be charging/discharging output loads, thus affecting cell and circuit performance. We derive exact lower bounds to easily evaluate the number of serial switches needed and demonstrate that Complementary Series/Parallel (CSP) and Pass Transistor Logic (PTL) topologies exceed the lower bounds for many practical examples. We also propose a design methodology that will produce cells with minimum number of transistors in series and evaluate the benefits obtained in circuit delay.

1. INTRODUCTION

The number of serial switches inside a cell is related to the maximum speed this cell may attain in CMOS technologies. Regardless of the transistor topology used to implement a switching function, there is a strong correlation between the length of the longest transistor chain and the delay, since the switches along this path are likely to correspond to the charge or discharge path that corresponds to the worst case delay scenario. As an example, let us consider the implementation of a pass transistor logic (PTL) module. The approach in [1] utilized this notion in its algorithm for performance-driven PTL synthesis. The method presented there exploits two separate effects. First, it aims to reduce the number of serially connected gates by applying functional decomposition. Second, it reduces the number of serial transistors (switches) inside the gates by encoding decompositions with a one-hot code and deriving cell level PTL networks partially from a BDD (Binary Decision Diagram) and partially from a one-hot multiplexer. The results reported there show significant performance gains, proving the importance of the number of serial transistors (switches) as a parameter to the quality of cell networks, especially when performance is the design goal.

Synthesis techniques for PTL circuits have been closely related to BDD representation of logic functions, for reasons such as elimination of sneak paths and the availability of efficient algorithms for the construction of BDDs [1]. Indeed, the approaches in [1-8] are based on BDDs. Therefore, when discussing PTL networks, we will concentrate our discussion on PTL networks derived from BDDs. Despite the gains demonstrated by [1], we will demonstrate that even the introductory example used there, the circuit *c3* (carry-out for the first 3 bits of an adder), shown in Fig. 1.a, may be synthesized with a significantly smaller number of serial transistors than originally presented. Fig. 1.a shows a PTL gate for *c3* with (inverting) buffers inserted. This way, the bottom part, below the first stage of buffers, has four transistors in series (counting the transistors inside the buffer that generates signal B2), while the top part of the cell has three transistors in series (counting the transistors inside the buffers). Consequently, the PTL realization in Fig. 1.a may then be viewed as two independent gates connected in cascade. However, we noticed that the *c3* function may be synthesized as a single cell where the pull-down network has at most four serial transistors and the pull-up chain has at most three. This implementation is shown in Fig. 1.b, and it is 60% faster than the fastest decomposed version presented in [1]. Moreover, this is not a problem that is unique to the cell generation method in [1]. Indeed, this seems to be true for approaches based on PTL, as we also found examples of non-optimized pull-up and/or pull-down paths in several other papers based on PTL logic [2, 8]. Notice that the circuit in Fig. 1.b does not use Complementary Series/Parallel (CSP) logic [9-12]. The use of CSP topology would produce a network with a transistor chain longer than the PTL version of Fig. 1.a. These observations lead us to formulate the following questions.

1) What is the minimum length for the pull-up and pull-down chains when designing a switch network for a given logic function?

2) Is it possible to synthesize a network for the circuit in Fig. 1.b with shorter pull-up and pull-down path lengths?

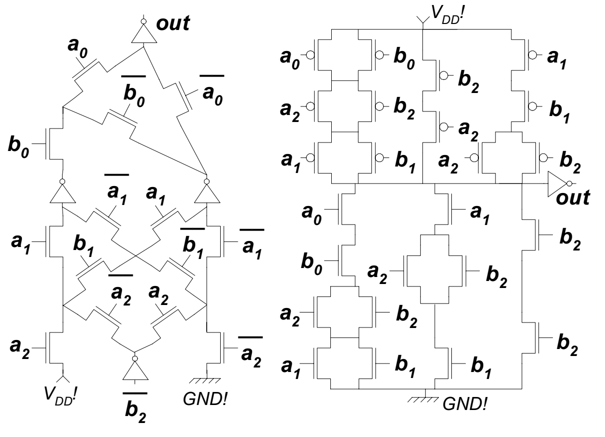
2. SWITCHES AND LOGIC CELLS

A switch controls the connection between two different points. The discussion in this paper will be restricted to two different kinds of switches, as described in the following. An **active-*0* switch** will connect two nodes if the control variable is equal to *0*; the switch will not connect these points when the control signal is equal to *1*. Similarly, an **active-*1* switch** will short-circuit two nodes if the control variable is equal to *1* and it will be an open circuit if the variable is *0*. PMOS transistors are active-*0* switches and NMOS transistors are active-*1* switches.

The main switch topologies used to design transistor networks for logic cells are Pass Transistor Logic (PTL) [1-8] and Complementary Series/Parallel (CSP) CMOS Logic [9-12]. PTL topology is composed of a single non-disjoint **pull-up/down plane**, while CSP topology has two disjoint switch planes: one **pull-up plane** and one **pull-down plane**. Independently of the topology, the output of the cell is connected to V_{DD} or *GND* through a path composed of serially connected switches that are active (connected) under a given input assignment. A **pull-up path** connects the output of the cell to the V_{DD} (logic-*1*) reference, through a set of serially connected switches. A **pull-down path** connects the output of the cell to the *GND* (logic-*0*) reference, through a set of serially connected switches. A pull-up path is associated with an on-set implicant cube, while a pull-down path is associated with an off-set implicant cube.

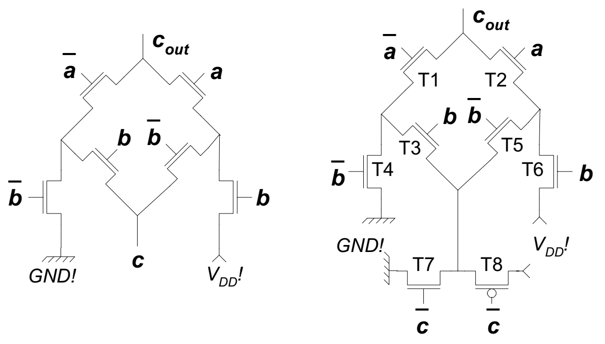
Table 1: Pull-up and pull-down paths Fig. 2.b.

Type	Transistors	Cube
Pull-up	T1-T3-T8	$\bar{a} \cdot b \cdot c$
	T2-T5-T8	$a \cdot \bar{b} \cdot c$
	T2-T6	$a \cdot b$
Pull-down	T1-T4	$\bar{a} \cdot \bar{b}$
	T1-T3-T7	$\bar{a} \cdot b \cdot \bar{c}$
	T2-T5-T7	$a \cdot \bar{b} \cdot \bar{c}$



(a) PTL implementation of *c3* (b) CMOS Network of *c3*

Figure 1: Implementations for function *c3*.



(a) drain inputs (b) strong signals through inverters

Figure 2: Two distinct PTL cells.

Example 1: Consider the PTL cells shown in Fig. 2.a and 2.b, for the carry-out function of a full adder. The PTL cell in Fig. 2.a has the input *c* connected to transistor drains. However, inside an integrated circuit these signals will always be available through another cell that will generate them. In the best case, these signals will be generated through an inverter. Thus the following discussion will consider that drain signals are available through inverters. Fig. 2.b shows the PTL cell with the transistors corresponding to the inverters added to the transistor network. The existing pull-up and pull-down paths for circuit in Fig. 2.b are shown in Table 1. The on-set (off-set) equation is given by the sum of all on-set (off-set) implicants corresponding to pull-up (pull-down) paths, as given by equations *1* and *2* below. Notice that these equations are correct, but they are not prime covers, as prime covers for the on-set and off-set of this particular function would have cubes composed of at most two literals.

$$on - set = a \cdot \bar{b} \cdot c + \bar{a} \cdot b \cdot c + a \cdot b \quad (1)$$

$$off - set = \bar{a} \cdot b \cdot \bar{c} + \bar{a} \cdot \bar{b} + a \cdot \bar{b} \cdot \bar{c} \quad (2)$$

3. EXACT LOWER BOUND FOR SERIAL SWITCHES

The lower bound we propose is based on the number of literals of the smallest cube in a prime irredundant cover. The problem with this is that if a function might have distinct prime and irredundant covers with a different number of literals in the smallest cube, then the lower bound would not be univocally defined. In the following we present a proof to ensure that this condition will never happen as the size of the smallest cube in distinct prime irredundant covers of a logic function is univocally defined.

Definition 1: A cube with *m* literals will have **cube size** 2^{n-m} in Boolean space B^n. By definition, the **smallest cube** is the cube with larger number of literals.

Theorem 1: The number *m* of literals in the smallest cube does not change for distinct prime implicant covers of the same logic function.

Proof (by contradiction): Consider two distinct prime irredundant covers *C1* and *C2* of the same function *f* such that the smallest cubes in *C1* and *C2* have different sizes. Suppose that cover *C1* has smallest cube(s) composed of *m* literals. Suppose also that the smallest(s) cube(s) in cover *C2* are composed of *m-i* literals, such that $0<i<m$. To turn on the any prime irredundant cube in *C1*, it is necessary to assign *m* variables to logic-*0* or logic-*1* as appropriate. The reason for this is because the smallest cube in *C1* is not redundant and has *m* literals. However, assigning at most *m-i* variables is sufficient to turn on any prime implicant cube in *C2*. Thus, the analysis of the complete set of all variable assignments containing *m-i* or fewer variables in B^n is sufficient to decide if function represented by *C2* evaluates to logic-*1*. However, to decide if function represented by *C1* evaluates to logic-*1*, it is necessary to assign *m* variables in the worst case, due to the irredundant prime implicant(s) with *m* literals. Therefore, the functions given by covers *C1* and *C2* are not the same logic function, leading to a contradiction of our initial hypothesis that two distinct prime implicant covers of the same function could have smallest cubes with different sizes. QED.

Corollary: Theorem 1 is valid for two prime irredundant covers. Non-prime covers may have smaller non-prime cubes with more than *m* literals, where *m* is the size of the smallest cube in any prime irredundant cover. However, no cover may have only cubes greater (with a smaller number of literals) than the smallest cube in any prime irredundant cover.

Theorem 2: Given a function *f*, it is not possible to have a cell whose longest pull-up path has fewer switches than *m*, the number of literals in the smallest cube of any prime irredundant cover *C1* for the on-set of function *f*.

Proof (by contradiction): Recall that the smallest cube has the greater number of literals (Definition 1). By theorem 1, all the prime irredundant covers of function *f* will have at least one cube with *m* literals, where *m* is the size of the smaller cube in any prime cover of *f*. Non-prime covers of *f* may have smaller cubes with more than *m* literals. However, covers where all the cubes have less than *m* literals are not possible, due to theorem 1 and its corollary. Consider a function *f* defined in B^n, such that a prime irredundant cover *C1* of *f* has *m* literals. Suppose now that function *f* has a generic switch realization where the i^{th} pull-up path has size p_i, such that $p_i < m$. As described in detail through example 1 of section 2, each of these pull-up paths would be associated with an on-set implicant cube with p_i literals. As a consequence, every pull-up path with size $p_i < m$ will produce a cube with p_i literals where $p_i < m$. The cover obtained from the network this way will have only cubes with less than *m* literals, as $p_i<m$, for every path. According to theorem 1, this contradicts the initial hypothesis that the smallest cube in the prime implicant cover *C1* has *m* literals, as all the cubes in the realization would have a smaller number of literals $p_i < m$ and a greater size. QED.

Theorem 3: Given a function *f*, it is possible to produce a cell where the longest pull-up path has *m* switches in series, where *m* is the number of literals in the smallest cube in a prime irredundant cover *C1* for the on-set of function *f*.

Proof (by construction): It is possible to construct a pull-up plane for function *f* given a prime irredundant cover P_i of the function, where each cube $P_i=\prod l_i$ is a product of literals associated to the variables in the domain of the function. Every prime cube P_i contributes to the pull-up plane with an independent pull-up path. The paths for each cube P_i are independent as they are parallel paths among each other. Each of the independent paths is composed of serially connected switches between the logic-*1* reference (VDD) and the output of the cell. The path for a given cube in the cover contains one serially connected switch for each literal l_i in the cube, as described by example 1 in section 2. As the smallest cube has the greater number of literals, it will determine the size of the longest path. Thus this implementation will have by construction the longest pull-up path with a size correspondent to the number *m* of literals in the smallest cube, as stated by the theorem. Furthermore, as the path for each cube is independent, this solution has no sneak paths. QED.

Theorem 4: The exact lower bound in the number of serial switches in the longest pull-up path of a logic function *f* is given by *m*, the number of literals in the smallest cube in any prime irredundant cover *C1* for the on-set of function *f*.

Proof: Immediate corollary of Theorems 2 and 3, and univocally defined as consequence of Theorem 1. QED.

Theorem 5: Given a function *f*, it is not possible to have a cell whose longest pull-down path has fewer switches than *m*, the number of literals in the smallest cube of any prime irredundant cover *C1* for the off-set of function *f*.

Proof: The proof is similar to that of Theorem 2.

Theorem 6: Given a function *f*, it is possible to produce a cell where the longest pull-down path has *m* switches in series, where *m* is the number of literals in the smallest cube in a prime irredundant cover *C1* for the off-set of function *f*.

Proof: The proof is similar to that of Theorem 3.

Theorem 7: The exact lower bound in the number of serial switches in the longest pull-down path of a logic function *f* is given by *m*, the number of literals in the smallest cube in any prime irredundant cover *C1* for the off-set of function *f*.

Proof: Immediate corollary of Theorems 5 and 6, and univocally defined as consequence of Theorem 1.QED.

4. APPLICATIONS AND CONSEQUENCES

This section presents consequences of the lower bounds introduced in previous section. For a better understanding of the lower bounds, an illustrative example is presented.

Example 2: As an example, consider a function *f* given by equation $f = a \cdot b + b \cdot c + a \cdot c \cdot d$. The minimum covers for the on-set and the off-set of this function are:

$$on\text{-}set = a \cdot b + b \cdot c + a \cdot c \cdot d \quad (3)$$

$$off-set = \overline{a} \cdot \overline{b} + \overline{a} \cdot \overline{c} + \overline{b} \cdot \overline{d} + \overline{b} \cdot \overline{c} \quad (4)$$

The smallest cube in the on-set is $a \cdot c \cdot d$, so the lower bound for the number of serial transistors in the pull-up network is three. The cubes in the off-set are all the same size, and the lower bound for the number of serial switches in the pull-down network is two. This way the cell corresponding to the function *f* is a 3-2 cell, when mapped with the constructive method proposed here.

Example 3: Recall the function in example 1, the carry-out of a full adder. Prime irredundant covers for the on-set and off set are given by the following equations.

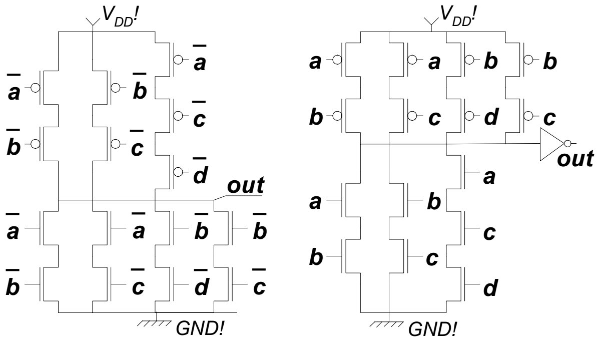
$$on\text{-}set = a \cdot b + a \cdot c + b \cdot c \quad (5)$$

$$off-set = \overline{a} \cdot \overline{b} + \overline{a} \cdot \overline{c} + \overline{b} \cdot \overline{c} \quad (6)$$

It is easy to see that the lower bounds for the carry-out in a full adder are two switches for both the pull-up and pull-down planes. This is consistent with the classic 2-2 cell for carry-out generation presented in the cover of the Weste-Eshraghian book [13].

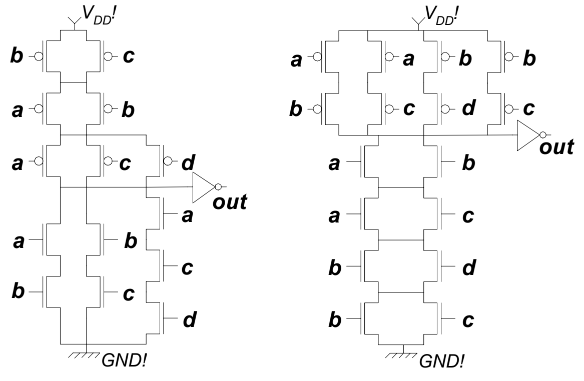
4.1 NCSP topology

We will refer to the constructive method presented in this paper as **Non-Complementary Series/Parallel (NCSP)** topology. The constructive method was detailed in the proof of theorem 3. As an example, consider the resulting cell for the function *f* in example 2. The NCSP cell has the pull-up derived from the on-set equation (hence the longest pull-up path has 3 switches) and the pull-down derived from the off-set equation (thus the longest pull-down path has 2 switches), as it may be observed in Fig. 3.a. As the on-set and the off-set may be interchanged by inverting the function and adding and inverter at the output, it is always possible to use the smaller constraint in the pull-up network. This is desirable because PMOS transistors are more resistive than NMOS ones. The NCSP network considering the inverted version of function *f* is shown in Fig. 3.b.



(a) for direct *f* (b) for inverted *f*

Figure 3: NCSP cells respecting the lower bounds for function *f* from example 2.



(a) from on-set equation (b) from off-set equation

Figure 4: CSP cells not respecting the lower bounds for function *f* from example 2.

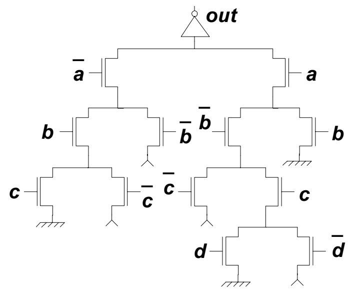


Figure 5: PTL cell not respecting the lower bounds for function *f* from example 2.

4.2 Comparison with a CSP topology

A **Complementary Series/Parallel** (CSP) [9-12] derived only from the on-set equation would result in a 3-3 cell, as shown in Fig. 4.a. Similarly, if a CSP [9-12] cell were derived exclusively from the off-set equation, the result would be the 2-4 cell illustrated in Fig. 4.b. Thus, the use of the lower bound will produce a cell that has smaller pull-up and pull-down networks than CSP [9-12]. Based on example 2 above, it is possible to see that the NCSP topology we propose may be used to reduce the length of pull-up and pull-down chains when implementing cell level networks.

4.3 Comparing to PTL topology

The Pass Transistor Logic (PTL) [1-8] realization of the cell from example 2 would be a 4-4 cell, independently of the BDD variable order used to generate the PTL network [14]. One possible PTL network is shown in Fig. 5. This way, the use of the lower bound will produce a cell that has smaller pull-up and pull-down networks than PTL [1-8]. Based on example 2, it is possible to see that the NCSP we propose may be used to reduce the length of pull-up and pull-down chains when implementing cell level networks. Indeed, we found examples of circuits not respecting the lower bound in [1-7].

4.4 Lower bound and general PTL styles

Notice that the lower bounds we defined here apply to general PTL styles, due to our definition of pull-up and pull-down paths. As a pitfall counterexample, consider a prime irredundant cover, $a{\cdot}b$. The theorem states that lower bound in the number of serial switches is two, but one could argue that it is possible to synthesize a PTL circuit with only one serial switch: an NMOS transistor with gate input *b* and the drain terminals directly connected to variable *a*. However, our definition of pull-up and pull-down paths starts at a power supply node, VDD or GND. Therefore, this counterexample is not valid, because variable *a* is not a power supply node.

4.5 Evaluating the lower bound

At first, it may seem too time-intensive to calculate the lower bound for candidate functions, as it is necessary to calculate two prime irredundant SOPs. However, as we only evaluate the pull-up and pull-down chain inside cells, the evaluation process is not a critical step because it is easy to obtain prime irredundant SOPs when the number of variables is small.

5. RESULTS

Experiments were run on a set of well-known benchmark circuits available from many sources. Each circuit was preprocessed with SIS [15] and mapped with 4-input Look-up Tables (LUTs). A script for LUT based FPGAs suggested in the documentation of SIS was used. As a result each circuit is composed by a set of sub-functions having at most four inputs. From these preprocessed networks, four different transistor networks were derived, as described in the following. We have used SIS, but the lower bound and the NCSP topology are compatible with the decomposition methods presented in [1-6]. What is important in the data presented is the relative performance among the four topologies, as the gains we obtain are at the cell level, and they are preserved independently of the specific decomposition method used, when comparing NCSP to more common PTL [1-8] or CSP [9-12] approaches.

CSP. Each logic function in the preprocessed circuit is mapped into a transistor network as a single cell using the Complementary Series/Parallel (CSP) topology [9-12]. Notice that this procedure could generate some infeasible cells: an EXOR4 would have eight transistors in series, for one of the cell planes. As a consequence, single cells with more than four serial transistors in series are decomposed into a set of cells with at most four transistors in series, so that the mapping with 4-input LUTs may be translated to a CSP circuit.

NPTL. Each logic sub-function in the preprocessed circuit is mapped into a transistor network using the PTL topology [1-8] composed only of NMOS transistors, as used in [1-6]. Notice that all the cells are feasible, as they will have at most 4 transistors in series. For instance, an EXOR4 would have four transistors in series, the maximum height of a 4-variable BDD [14]. However, some functions will be exceeding the lower bound. For instance a NAND4 would have four serial transistors for charging and discharging the output. We have used PTL networks derived from BDDs, as most of the recent publications on PTL synthesis derive PTL networks from BDDs [1-8].

CPTL. Each logic sub-function in the preprocessed circuit is mapped into a transistor network using the PTL topology composed of a pair of PMOS/NMOS transistors, as used in [7]. Notice that all the cells are feasible. For instance, an EXOR4 would have four transistors in series. However, some functions will be exceeding the lower bound. For instance a NAND4 would have four serial transistors for charging and discharging the output.

NCSP. Each logic sub-function in the preprocessed circuit is mapped into a transistor network using either the CSP or with the non-complementary series/parallel (NCSP) topology proposed in this paper to minimize the number of transistor in series. NCSP is preferred; CSP will only be used when it respects the lower bound and decreases the total transistor count (for area reasons). Notice that with this configuration all the cells will respect the lower bound. This is not achieved by using CSP or PTL alone, or even by using a mix of both [16-20], as demonstrated for the cell in example 2. Observe that NCSP uses disjoint pull-up (PMOS) and pull-down (NMOS) planes, which further increases speed when compared to NPTL.

The main goal of the theory exposed in this paper was to reduce the number of switches in series in the pull-up and pull-down networks. Table 2 illustrates the reduction in the sum of the pull-up and pull-down chains. As expected, the pull-up and pull-down chains obtained through NCSP are the shortest.

The number of transistors necessary to implement each circuit is presented in Table 3 for mapping with 4-input LUTs. Notice that this number of transistors is shown only for a relative area comparison. The final number of transistors and the area depends on the quality of the decomposition. Small differences between CSP and NCSP in transistor count can be seen in Table 3. The implementations using NPTL and CPTL will result in smaller transistor counts, as they were obtained from optimized BDDs, while CSP and NCSP were obtained from non-optimized SOPs.

Table 2: Sum of pull-up/pull-down switches in the critical path: LUT4.

	CSP		**NPTL**		**CPTL**		**NCSP**	
Circuit	PU	PD	PU	PD	PU	PD	PU	PD
C432	28	60	68	68	68	68	28	60
C1355	20	33	28	28	28	28	20	26
C499	23	36	31	31	31	31	23	29
C1908	33	59	48	48	48	48	33	43
C880	27	44	48	48	48	48	27	36
C2670	31	48	56	56	56	56	30	46
C3540	45	72	67	67	67	67	44	64
C7552	29	39	36	36	36	36	29	31
C5315	28	49	45	45	45	45	28	42
C6288	96	127	119	119	119	119	96	102
Total	362	570	550	550	550	550	360	482

Table 3: # of transistors for benchmark mapping: LUT4.

Circuit	**CSP**	**NPTL**	**CPTL**	**NCSP**
C432	726	754	958	726
C1355	2910	1402	2031	2910
C499	2800	1370	1962	2800
C1908	2296	1452	1980	2298
C880	1814	1436	1891	1814
C2670	3414	2718	3465	3421
C3540	5218	5010	6610	5219
C7552	10384	7502	10125	10455
C5315	8284	7286	9444	8292
C6288	14912	8882	12478	15250
Total	52786	37846	50990	53213

Table 4: # Delay of the circuits simulated with SPICE: LUT4.

Circuit	**CSP**	**NPTL**	**CPTL**	**NCSP**
C432	5012ps	7746ps	7441ps	5012ps
C880	5289ps	6008ps	5448ps	4951ps
C2670	5269ps	6934ps	6310ps	4978ps
C3540	8513ps	9285ps	9149ps	8056ps
Total	25539ps	31713ps	29930ps	24453ps

Table 4 shows the delay of the circuits obtained through SPICE simulations. The delay is in ps (picoseconds), for the TSMC 0.25μm technology. It is possible to see that the delays correlate to the sum of the sizes of pull-up and pull-down networks in the longest path. The results in Table 4 are even more attractive for NCSP as it is composed of disjoint planes for pull-up and pull down. Therefore PMOS may be used for pull-up and NMOS for pull-down, and NCSP will be less susceptible to noise margin problems. This represents a significant reliability advantage for NCSP.

6. CONCLUDING REMARKS

The approach recently presented in [21] to increase circuit performance shows examples that use complex cells with reduced pull-up and pull-down paths, but these parameters (minimum pull-up and pull-down chains) are not explicitly cited in the text. The approach in [21] clearly states that the cell transistor network generation is an important point in their performance oriented design flow. However, they make no explicit mention to the pull-up and pull-down length of the cells as important parameters. We believe this work to be the first to explicitly and unequivocally point out the importance of these parameters.

The use independently optimized logic for pull-up and pull-down planes was also used in [22]. However, the goal in that work was to produce hazard-free complex gates and the length of pull-up and pull-down chains was not discussed as a design parameter. It is important to notice that the work in [22] does not point to older references that use independently optimized logic for pull-up and pull-down planes.

We have derived an exact lower bound for the number of switches needed to implement a logic function at the switch level. We have shown that the most used transistor topologies, pass transistor logic (PTL) [1-8] and complementary series/parallel (CSP) [9-12] will not respect the presented lower bound for some logic functions. This was demonstrated through the sum of the lengths of pull-up and pull-down longest chains for each cell in the critical path for circuits mapped with 3-input and 4-input logic functions. The impact of this overhead in the number of serial transistors in the delay of mapped circuits was evaluated through SPICE simulations, and compared to the delay of NCSP networks proposed here. The reduction of the length of pull-up and pull-down translates into better timing for the NCSP approach. We believe that the lower bounds presented here will have a significant impact in the design of high performance integrated circuits using methods like [1, 21], through the guidance for the choice of better cell topologies. Note that the NCSP topology and the lower bounds proposed here are valid and useful independently of the decomposition used [1-6], and could possibly increase the speed of many circuits by simply exchanging the switch level cell networks in the critical paths with optimized versions. We are currently working on methods for automatic synthesis of area efficient switch networks that respect the lower bound.

7. ACKNOWLEDGEMENTS

This work was partially supported by CAPES Brazilian funding agency.

8. REFERENCES

[1] R.S. Shelar and S. Sapatnekar. Recursive bipartitioning of BDDs for performance driven synthesis of pass transistor logic circuits. ICCAD 2001. Pages: 449 – 452.

[2] P. Buch, A. Narayan, A.R. Newton and A. Sangiovanni-Vincentelli. Logic synthesis for large pass transistor circuits. ICCAD 1997. Pages: 663 – 670.

[3] C. Scholl and B. Becker. On the generation of multiplexer circuits for pass transistor logic. DATE 2000. Pages: 372 – 378.

[4] P. Lindgren, M. Kerttu, M. Thornton and R. Drechsler. Low power optimization technique for BDD mapped circuits. ASP-DAC 2001. Pages: 615 – 621.

[5] R.S. Shelar and S.S. Sapatnekar. An efficient algorithm for low power pass transistor logic synthesis. ASP-DAC 2002. Pages: 87 – 92.

[6] S-F. Hsiao, J-S. Yeh and D-Y. Chen. High-performance multiplexer-based logic synthesis using pass-transistor logic. ISCAS 2000. Pages: 325-328, Vol.2.

[7] M. Avci and T. Yildirim. General design method for complementary pass transistor logic circuits. Electronics Letters, Vol.: 39, Number: 1, 9 Jan. 2003. Pages: 46 – 48.

[8] H. Zhou and A. Aziz. Buffer minimization in pass transistor logic. IEEE Transactions on CAD, Volume: 20, Number: 5, May 2001. Pages: 693 – 697.

[9] M.R.C.M. Berkelaar and J.A.G. Jess. Technology mapping for standard-cell generators. ICCAD-1988, Pages: 470 – 473.

[10] A.I. Reis, M. Robert, D. Auvergne and R. Reis. Associating CMOS transistors with BDD arcs for technology mapping. Electronics Letters, v. 31, n. 14, p. 1118-1120, 1995.

[11] A. I. Reis, R. Reis, D. Auvergne and M. Robert. Library free technology mapping. In: Reis, R.; Claesen, L. (Org.). VLSI: Integrated Circuits on Silicon. London, 1997, p. 303-314.

[12] S. Gavrilov, A. Glebov, S. Pullela, S.C. Moore, A. Dharchoudhury, R. Panda, G. Vijayan and D.T. Blaauw. Library-less synthesis for static CMOS combinational logic circuits. ICCAD 1997. Pages: 658 – 662.

[13] N. H. E. Weste and K. Eshraghian, Principles of CMOS VLSI Design: A Systems Perspective, Addison Wesley, Boston, MA, USA, 1999.

[14] S. Nagayama and T. Sasao, On the minimization of longest path length for decision diagrams, IWLS04, Pages: 28-35.

[15] E.M. Sentovich et al. SIS: A System for Sequential Circuit Synthesis. University of California at Berkeley, Memorandum No. UCB/ERL M92/41, May 1992.

[16] S. Yamashita, K. Yano, Y. Sasaki, Y. Akita, H. Chikata, K. Rikino and K. Seki. Pass-transistor/CMOS collaborated logic: the best of both worlds. Symposium on VLSI Circuits, 1997. Pages: 31 – 32.

[17] Y.Jiang, S. Sapatnekar and C. Bamji. Technology mapping for high-performance static CMOS and pass transistor logic designs. IEEE Transactions on VLSI, Volume: 9, Number: 5, Oct. 2001. Pages: 577 – 589.

[18] G.R. Cho and T. Chen. Mixed. PTL/static logic synthesis using genetic algorithms for low-power applications. ISQED 2002. Pp: 458 – 463.

[19] G. R.Cho and T. Chen. Synthesis of single/dual-rail mixed PTL/static logic for low-power applications. IEEE Transactions on CAD, Volume: 23, Number: 2, Feb. 2004. Pages: 229–242.

[20] K. Yip and D. Al-Khalili. Multilevel logic synthesis using hybrid pass logic and CMOS topologies. IEE Proceedings, Circuits, Devices and Systems. Volume: 150, Number: 5, Oct. 2003. Pages: 445-452.

[21] R.Roy, D.Bhattacharya and V.Boppana. Transistor-level optimization of digital designs with flex cells. IEEE Computer, Feb. 05, pp. 53-61.

[22] P.Kudva, G.Gopalakrishnan, H.Jacobson and S.M.Nowick. Synthesis of hazard-free customized CMOS complex-gate networks under multiple-input changes. DAC 1996. Pages: 77 – 82.