Workshop on Exploitation of STG-based Design Technology
St. Petersburg, July 6-7, 1998

STG-Based Synthesis of Speed-Independent CMOS Cells

C. Piguet *, J. Zahnd **

* CSEM, Neuchâtel, Switzerland, piguet@csemne.ch
** EPFL, Lausanne, Switzerland, zahnd@di.epfl.ch

Abstract

This paper presents a method for designing speed-independent sequential cells from signal transition graphs (STG). The initial STG is assumed to satisfy the complete state coding property (CSC). It is modified by introducing auxiliary signals to satisfy the rule of alternate up and down transitions for CMOS circuits, as well as other rules derived from the properties of negative gates. Two design examples show that this method minimizes the number of transistors and the switching activity, and is thus well suited for the design of basic low-power library cells.

1. Introduction

Speed-Independent circuits are hazard-free under the unbounded gate delay model [1]. Their logical behaviour is thus insensitive to the variations of gate delays that may result from temperature and process variations. This is a crucial property for designing low-power library cells in submicron technologies. The correct behaviour of a delay-sensitive circuit can only be guaranteed by sufficiently sizing some transistors in order to enforce the desired outcome of critical races between internal variables. This is a difficult design problem, as transistor oversizing can conflict with such goals as minimizing the time delay of critical paths or the power consumption.

Most methods proposed in the literature for the logic synthesis of speed-independent circuits are unsatisfactory when it comes to real circuit implementation, as they do not completely free the designer from the responsibility of controlling some critical delays. In this sense, it may be said that they often fail to give really speed-independent circuits. The reason of this apparent contradiction between claim and result is that the property of speed independence is relative to a particular *model* of a circuit. In most methods, speed independence is only considered with respect to the binary coded state graph, or equivalently, the flow table of the circuit. But this property may be lost when logical equations and a corresponding logic diagram are derived from the flow table. It can be shown [4] that this is the case, in particular, with previously published methods based on signal transition graphs [1, 2, 3].

The method we propose also starts from a signal transition graph (STG). The key operation is to modify this STG by adding new internal variables in such a way that it gives rise to excitation functions for secondary variables which are all monotone decreasing boolean functions, also called negative functions. Each one can thus be implemented with a single CMOS gate, whereby no additional delay element is introduced. The speed independence of the signal transition graph is thus conserved when implementing the circuit. A similar method based on T-nets has been proposed in [14, 15], but seems to be unknown to integrated circuits designers. Additional rules are proposed to generate efficient race-free implementations built with negative logical functions only. A similar method has also been proposed in [6, 7, 8, 13] starting from a flow table. However it will be seen that the method we present here, starting from a STG, is more practical and easier.

2. Signal transition graphs

The concept of a signal transition graph (STG) will be informally presented here by means of an example. We refer to the literature [2] for a formal definition. Figure 1 shows a typical timing diagram of a master-slave D-flip-flop. Both master and slave output signals, Q and M, are represented in the diagram and in the corresponding STG (Fig. 2). The timing diagram of Figure 1 does not represent the full behavior of a D-flip-flop. Various other possible sequences are not represented, such as several CK pulses while D=1 or D=0, or D pulses (with implied M pulses) while CK=0. The flow table (Fig. 2) corresponding to this signal transition graph contains many question marks for which the behavior is not specified.

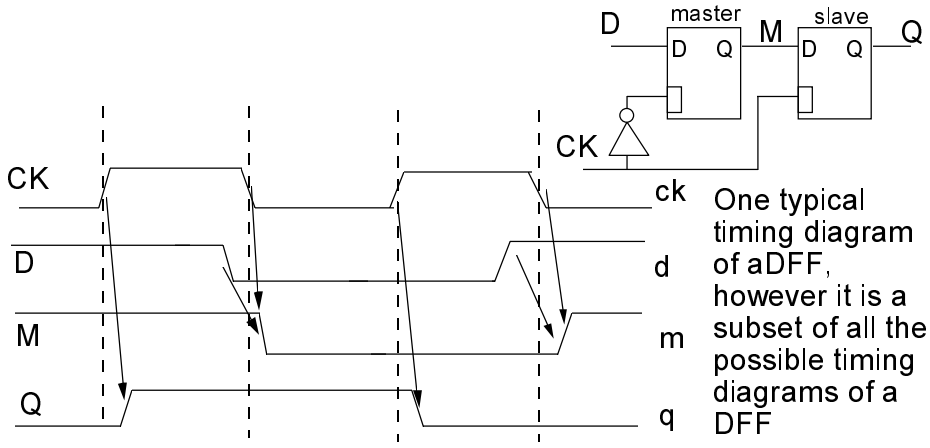


Figure 1. Partial Timing Diagram of a D-flip-flop

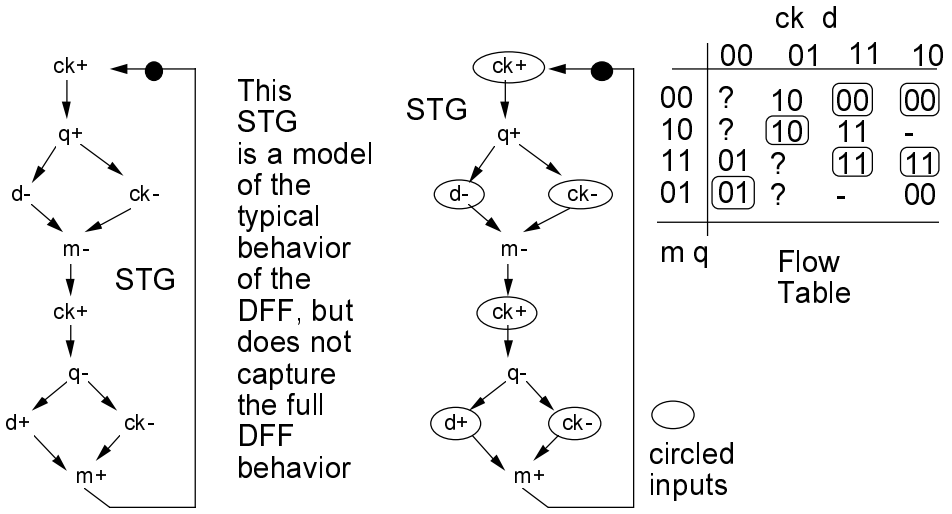


Figure 2. Signal Transition Graph (STG) of a D-flip-flop

Figure 3 shows a more complete timing diagram of a D-flip-flop, however without CK pulses while D=1 or D=0. Figure 4 shows the signal transition graph representing the full behavior of a D-flip-flop [16], including D pulses as well as CK pulses. They are represented as short loops going back to starting points. The flow table (Fig. 4) corresponding to this signal transition graph is completely specified.

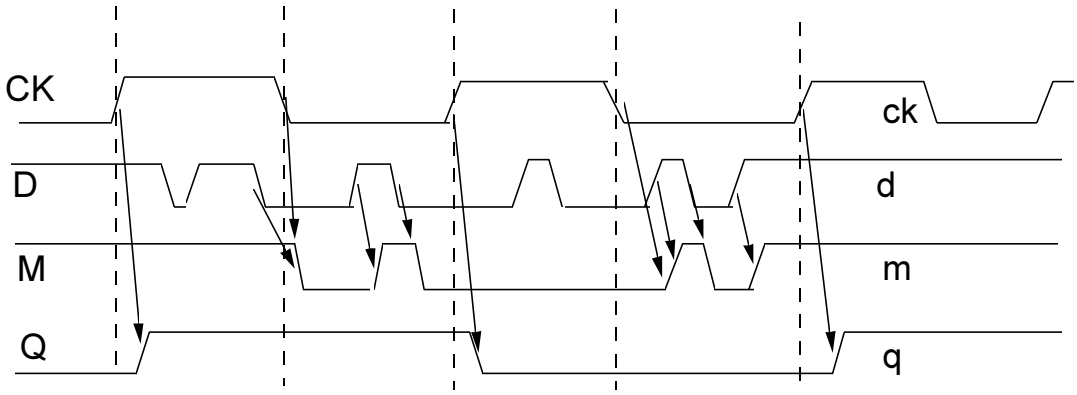


Figure 3. Timing Diagram of a D-flip-flop

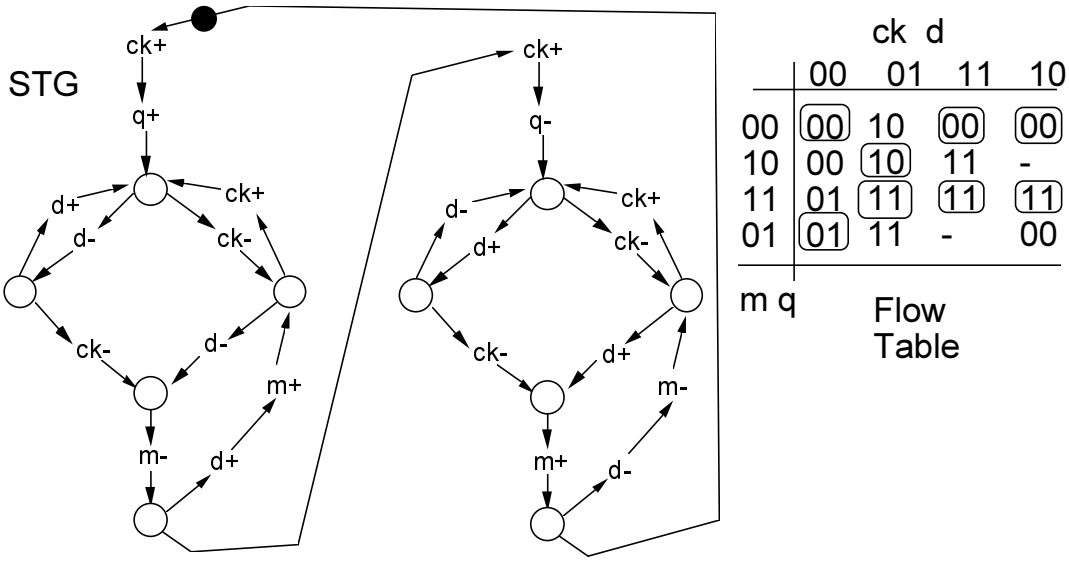


Figure 4. Signal Transition Graph of the full behavior of a D-flip-flop

Each signal change (or signal transition) of Figure 1, such as the first change of CK from 0 to 1, is represented in the STG of Figure 2, where it is associated with a node (or vertex) of the graph. A positive transition (0-->1) is noted with a "+" sign (e.g., ck+), and a negative transition (1-->0) with a "-" sign. The ordering of signal transitions given by the timing diagram is represented in the STG by the arrows (or directed edges) of the graph. Arrows of a STG can be *marked* with a heavy dot, like the upward arrow from m+ to ck+ in Figure 2, and such a marking of one or more arrows is a *state* of the STG, also called a *marking*. A STG is always given with an *initial marking*. A transition (node) is said to be *enabled* in a given state if all its incoming arrows are marked. A change of state is performed by *firing* (or *executing*) an enabled transition. This has the effect of unmarking all the incoming arrows of the transition and marking all the outgoing ones. For example, by executing ck+ and then q+, the initial marking of Figure 2 is transformed into a marking in which the two transitions d- and ck- are enabled. These may then be executed in any order, but both must be executed to obtain a marking that enables transition m-. Thus, the sequences ck+, q+, d-, ck-, m- and ck+, q+, ck-, d-, m- are two transition sequences that can be executed from the initial marking. Only one of them is represented in the timing diagram (Fig. 1). The set of all transition sequences that can be executed in a STG from its initial marking characterizes the *behavior* of the STG. In most cases it is an infinite set. Input signal transitions will be circled (Fig. 2) to help distinguish them from other signal transitions, which are all called output signal transitions.

Not any graph with nodes carrying signal transitions, together with an initial marking of arrows, is a valid STG. In a *valid* STG, it must be possible to assign each state (reachable marking) a combination of signal values in such a way that for any state change due to the firing of a transition, the signal change associated with the transition exactly corresponds to the difference between the two combinations of signal values that the two states are assigned. This is called a *consistent state assignment* [5].

Moreover, in order to be functionally implementable, a STG has to satisfy the *complete state coding* (CSC) requirement [2, 5], i.e., the set of non-input signal transitions that are enabled in a state *s* must be completely determined by the binary code of *s* in a consistent state assignment. This condition is satisfied in particular if the STG admits a *one-to-one* consistent state assignment, where distinct states are assigned distinct combinations of signal values. This is obviously the case for the STG in Figure 2, due to the fact that both master and slave output signals have been used in this behavior description. STGs admitting a one-to-one consistent state assignment are the starting point of our method. Any valid STG can be put in this form by introducing auxiliary signals in the behavior description [5]. We also assume the property of *semi-modularity* [2, 5], which amounts to say that there are no critical races in the corresponding state graph.

The signal transition graph of Figure 4 can be synthesized with the Petrify tool [17]. Figure 5 shows the obtained logical equations that correspond to latches [16]. Therefore, the resulting D-flip-flop structure is the well-known master-slave or C^{2}MOS D-flip-flop (Fig. 6). As shown in [4], such a flip-flop is not speed-independent due the clock inverter delay.

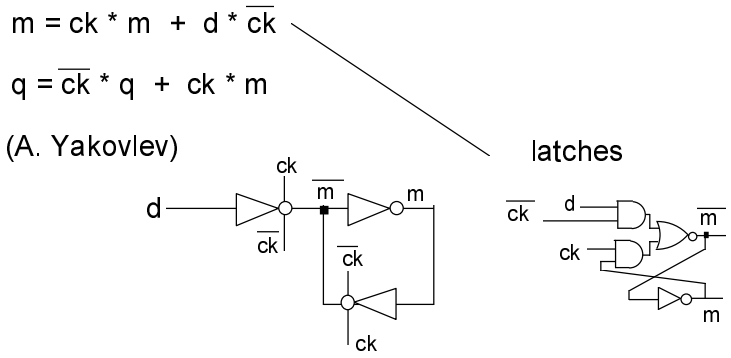


Figure 5. Logical Equations from Petrify

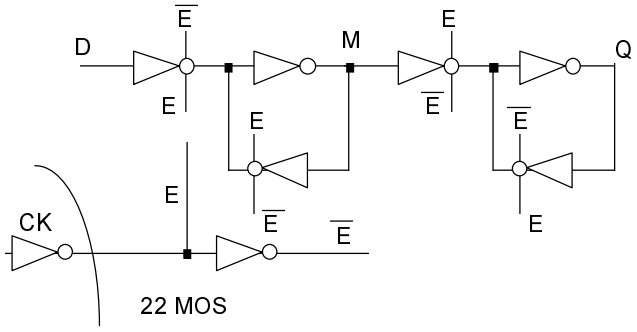


Figure 6. C^{2}MOS D-flip-flop

3. STG-based CMOS circuit design

3.1 CMOS gates

A realistic model of a CMOS gate [6, 7] is that of an ideal delay-free combinational circuit followed by an inversion operator, with which an inertial delay is associated. The delay-free combinational circuit contains no inverter, i.e., it is a positive Boolean function [6, 7]. Thus, the function implemented by a CMOS gate is the negation of a positive Boolean function, which is a negative or decreasing Boolean function. For any two input vectors Xi, Xj to such a function f, Xi ² Xj implies f(Xi) ³ f(Xj). For instance, a positive AND gate is realized in CMOS technology by a negative NAND gate followed by an inverter.

3.2 The rule of alternate transitions

In a circuit made of interconnected CMOS gates (CMOS circuit), the propagation of a signal change from one input through the circuit consists of a sequence of signal transitions with alternating signs. Each 1->0 signal transition is followed by a 0-> 1 transition (of another signal) and vice-versa. This results from the fact that CMOS gates are negative gates. An example is given in Figure 7. It could be a STG fragment (see 3.3) starting from an input variable change (ck+), followed by alternating transitions of output variables (a-, then b+), after which the next stable state is reached.

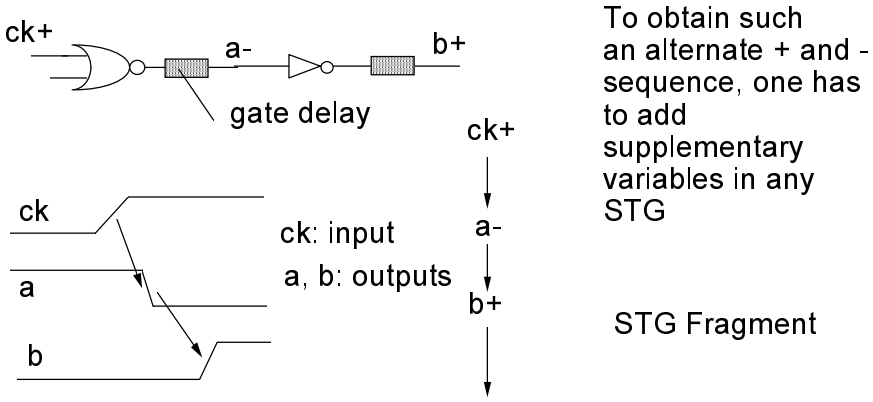


Figure 7. Alternating up and down transitions

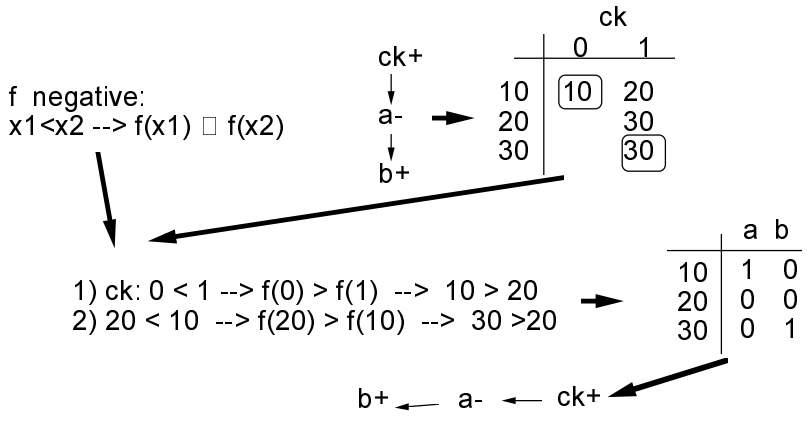
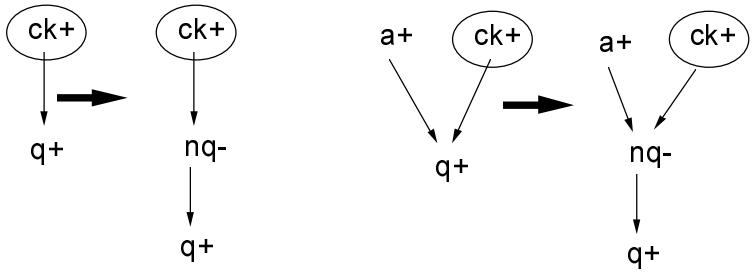


Figure 8. Alternating up and down transitions in a flow table

Figure 8 shows how this rule can be easily proven while using flow tables [6, 13]. The STG fragment can be represented by a flow table fragment and the property of negative functions can be applied to the first row of this flow table, resulting in state 10 > state 20, then to the second column. resulting in state 30 > state 20. To satisfy to these order relations, the state binary coding implies alternating up and down transitions.

3.3 STG fragments

With each output signal transition t of a STG, we associate the corresponding STG *fragment* that contains *t* and also contains the preceding transitions of other variables (input or output) that imply *t*. Figure 9 shows two STG fragments for the output transition q+ (the first one being implied by the input transition ck+, and the second one by the input ck+ and the output a+).



STG fragment with alternate up and down transitions

Input signal transitions need not obey the rule of alternating signs, as they are not implied by other signal transitions.

Figure 9. STG Fragments

3.4 The rule of alternate transitions for STG fragments

If S is a fragment of a STG for a CMOS circuit, then any sequence of transitions occurring within S has alternating signs. This is an obvious consequence of rule 3.2.

Input signal transitions need not obey the rule of alternating signs, as they are not implied by other signal transitions. This is why the rule only applies to STG fragments.

3.5 Input conflicts

When an output signal transition is implied by two input signal transitions with opposite signs (Fig. 10), we speak of an *input conflict.* Such a conflict has to be solved by introducing an additional internal signal (a+ in Fig. 10).

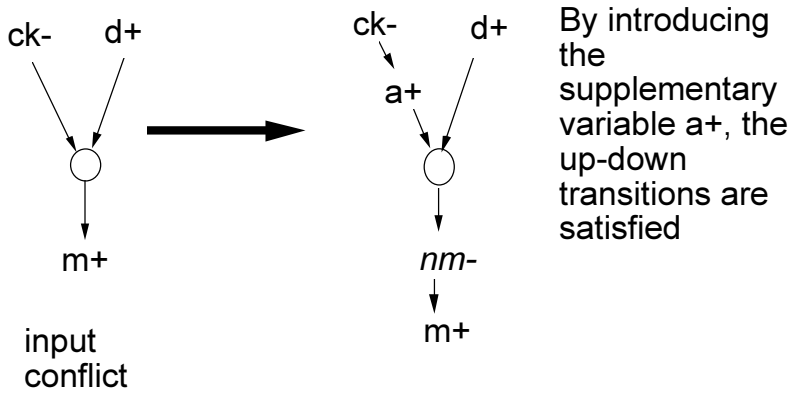


Figure 10. Input Conflict

As shown in Figure 11, an input conflict implies a contradiction if properties of negative functions are applied to a flow table presenting an input conflict. Both ck- and d+ result in the change from state 10 to state 20. Taking into account the order relations between input vectors, one has the following result that state 10 > state 20 > state 10. So such an input conflict cannot be implemented as such by only negative gates.

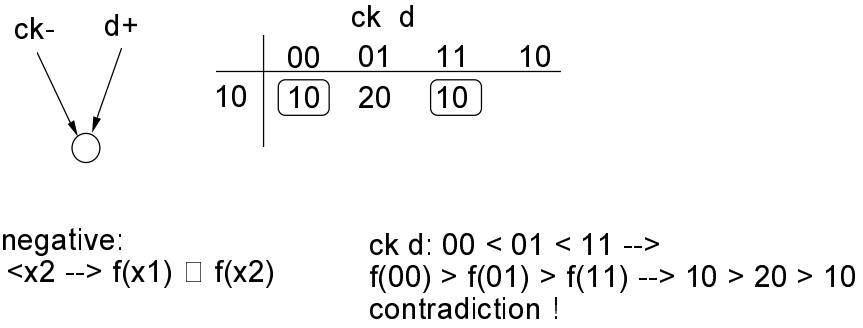


Figure 11. Input Conflict in a flow table

As shown in Figure 12, generally an input conflict is solved by the use in the circuit of both true and complemented inputs. If not(ck) is used, the resulting D-flip-flop structure is the well-known master-slave structure with a clock inverter that result in a non speed-independent circuit [4]. If not(d) is used, another well-known structure without clock inverter is obtained (Fig. 12). The differential D-flip-flop presented in [18, 19] is derived from this static structure. It is said to be clock slope insensitive, which is not the case of single-phase or TSPC static D-flip-flops.

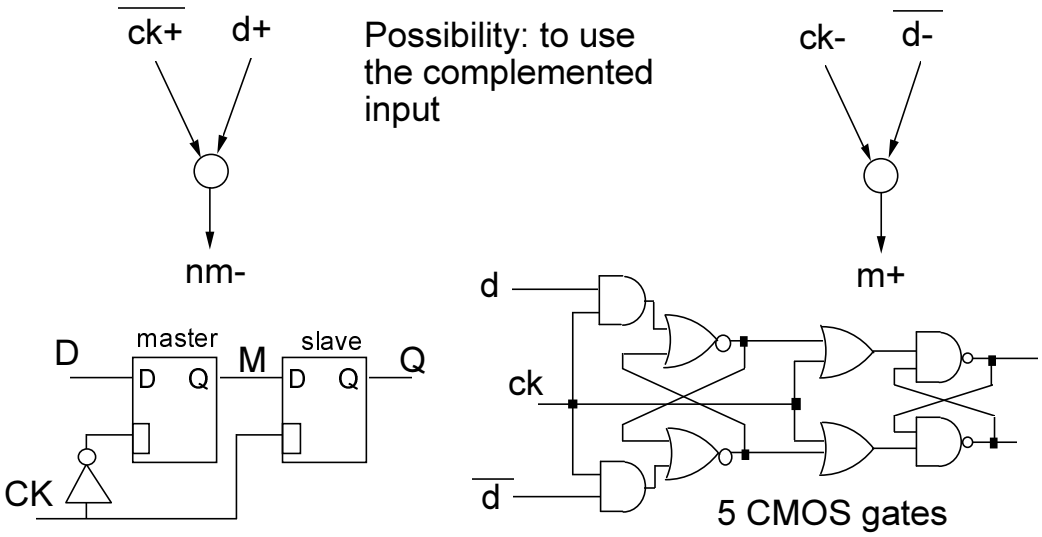


Figure 12. Input Conflict solved by true and complemented input

Figure 13 shows a possible modification of the input conflict by allowing m+ to switch after ck if d stays at "0" [16]. It is a modification of the original specifications for which m+ can switch only if both events ck- and d+ occur. As shown in Figure 13, when d=0, the output master m is switching with ck, and such a structure requires that the master output is not used (it has to be only an internal variable of the flip-flop). Synthesized, the resulting D-flip-flop is the only existing 4 negative gates structure (also presented in [13]), but it contains many transistors and slow branches with 3 transistors in series.

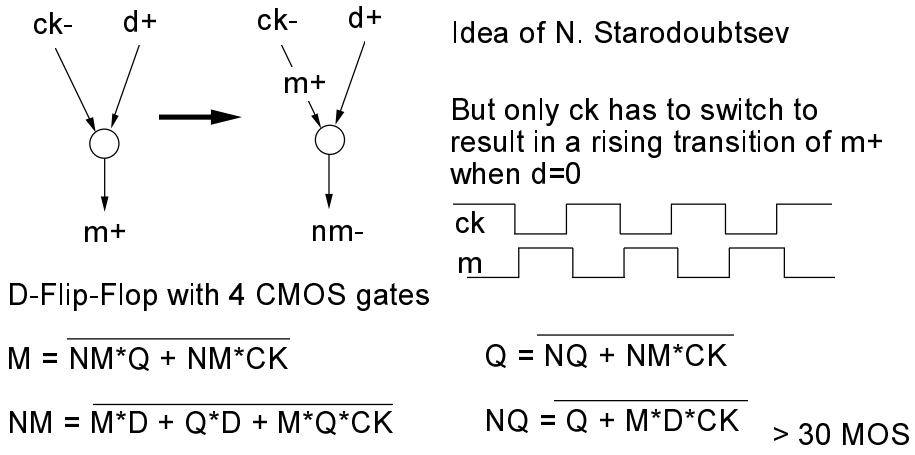


Figure 13. D-flip-flop with 4 CMOS gates

3.6 The cycle rule

In a complete cycle of a valid STG, any variable x has to switch an even number of times with alternate up and down transitions (x+ and x-).

3.7 Successive transitions of an input variable

Let t1, t2, t3 be a sequence of three consecutive transitions in the STG of a CMOS circuit. If t1 and t3 are transitions of the same input variable x and t2 is an transition of an output variable y implied by t1, then there is a transition t4 that follows t3 in the graph and reverses the signal change of t2. For example, if t1, t2, t3 is the sequence ck+, q-, ck-, then a transition q+ necessarily follows ck- in the graph. Indeed, if q is the output of a CMOS gate with input ck, and if an input change ck+ implies the output change q-, then the input change ck- implies q+. An example is given in Figure 14. If such symmetrical implications of two opposite transitions of the same input variable are not desired, then at least two different output variables must change between these two transitions. An additional variable has to be introduced to this effect (Fig. 14, nq+).

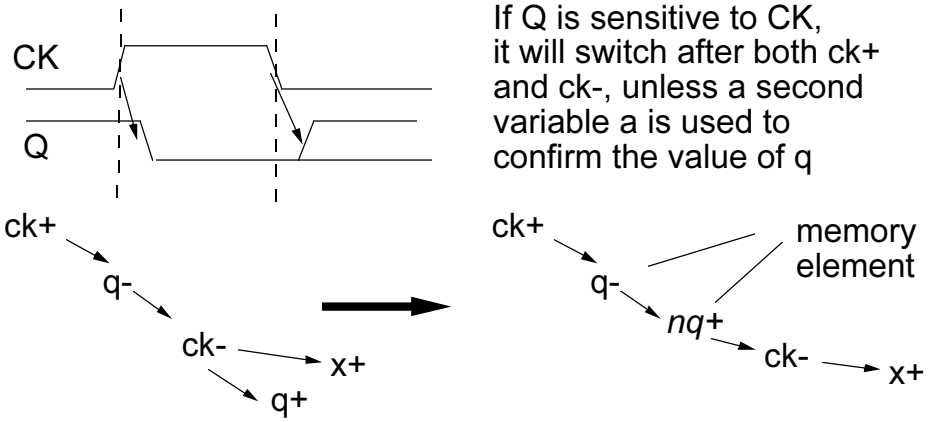


Figure 14. Transitions of the same variable

Figure 15 shows that if only one variable switches between two successive events of the same input, there is a contradiction in the flow table. Starting from state 10, after ck+ the state 20 is reached with q-. Then state 30 is reached with ck-. Applying the property of negative gates, one has state 10 > state 20 in the first row of the flow table, that implies state 30 > 10 in the first column, that also implies the contradictory order relation state 10 > state 30 in the first column. So two internal variables have to switch to have no order relation between stable states in the same column (for instance, states 10 and 30).

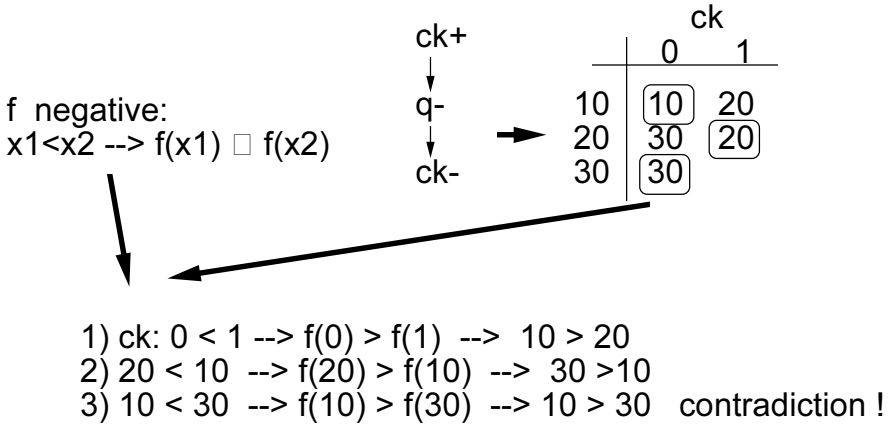


Figure 15. Transitions of the same variable in a flow table

3.8 Using memory elements

An assignment of a cycle in a given STG can be performed by means of memory elements, made of two CMOS gates a and an, with a+ and an- for a first fragment of the cycle and a- and an+ for another fragment. If a switches first in both fragments, the memory element is an "sm" (set-memory) element [6, 7]. If a switches first in one fragment and an first in the other, the memory element is an "sr" element (Figure 16).

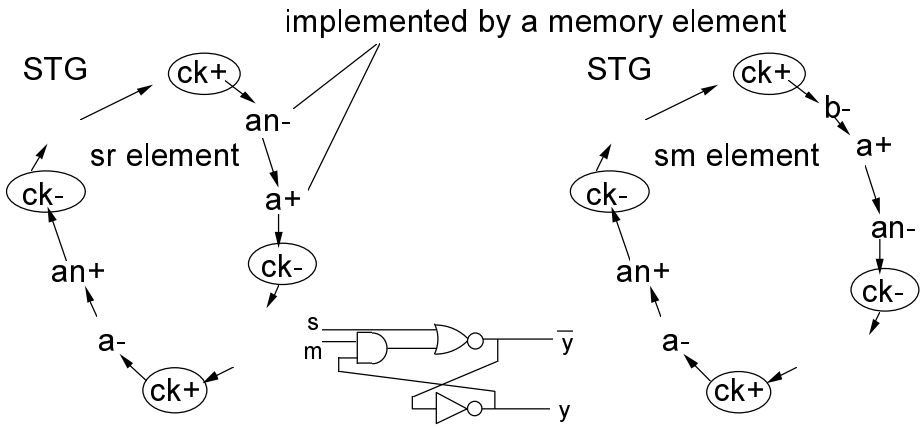


Figure 16. Memory Elements

3.9 Input variable complementation

The complementation of some input variables, i. e., the inversion of all their transitions in a STG (x+ becomes x- and vice-versa), can result in a STG with less input conflicts (3.5).

4. Design examples

Our first example is the design of a D-flip-flop starting from the timing diagram and STG of Figures 3 and 4. As explained in Section 3, this STG has to be modified to obtain alternate up and down transitions. Figure 17 shows the fragments (3.3) of this STG. Each fragment is modified by introducing additional internal variables to satisfy the rule of alternate transitions 3.3.

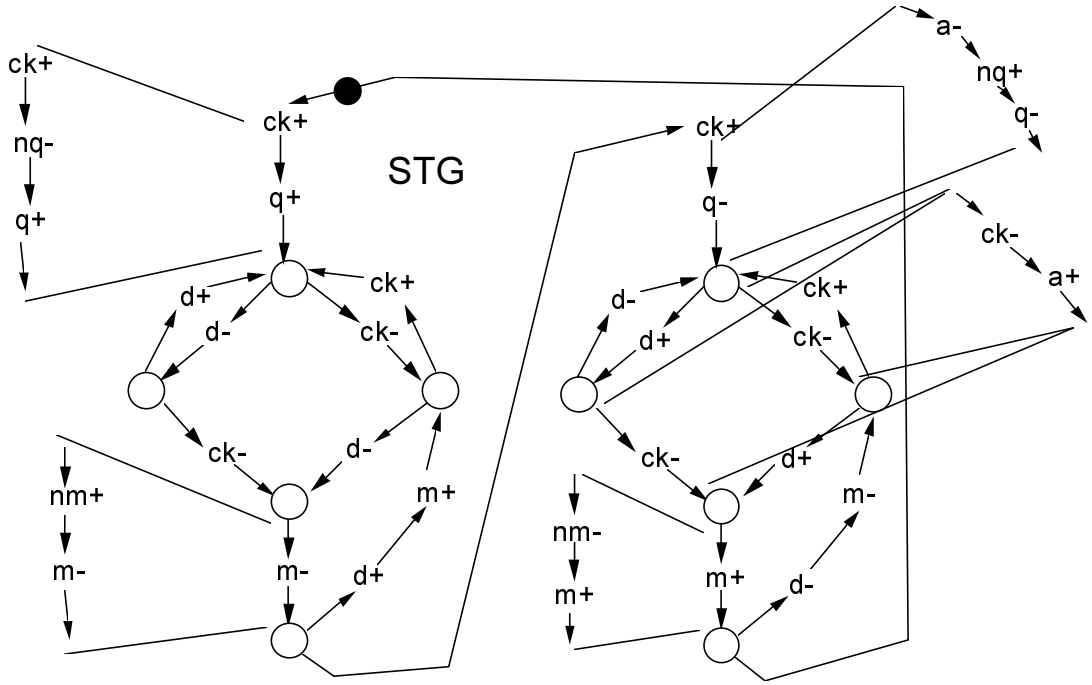


Figure 17. Decreasing STG Fragments for the D-flip-flop STG

An input conflict occurs in the fragment starting with input transitions d+ and ck-. A straightforward solution is to introduce an internal variable a+ between ck- and nm- (Figure 17 and Figure 10). Both (q, nq) and (m, nm) can be implemented with sm memory elements.

If only nm and nq as well as a+ are introduced, the resulting STG violates the cycle rule 3.6 as the transition a- is not specified. A new transitions a- has to be inserted in the cycle. This is done in Figure 17. As a+ is implied by ck- in one fragment, it simplifies the logic if a- is implied by ck+ in another one.

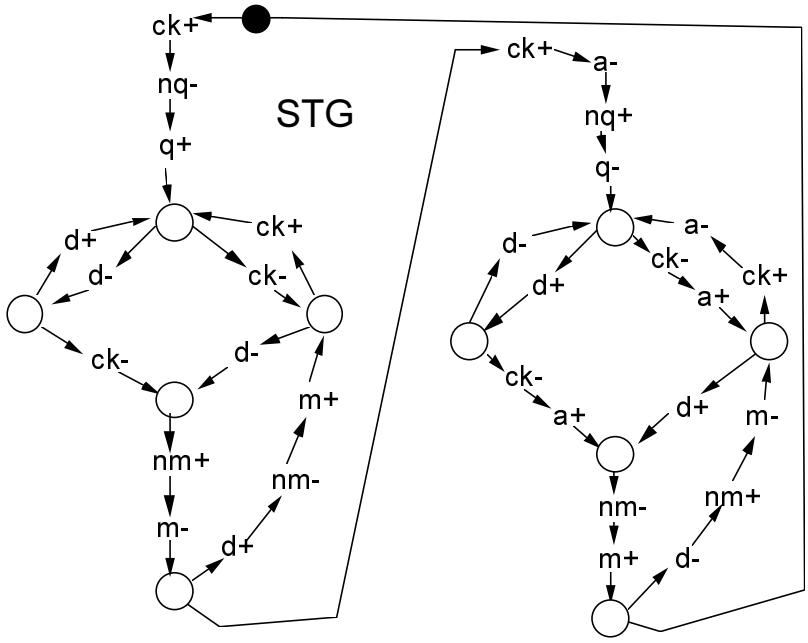


Figure 18. Decreasing STG of the D-flip-flop

Figure 18 shows the final STG which can be implemented with only negative gates. It represents the state graph together with the 5-variable one-to-one consistent state assignment. Stable states are circled and variables that switch next are marked with an asterisk. The corresponding flow table and the negative logic equations derived therefrom are given in Figure 20.

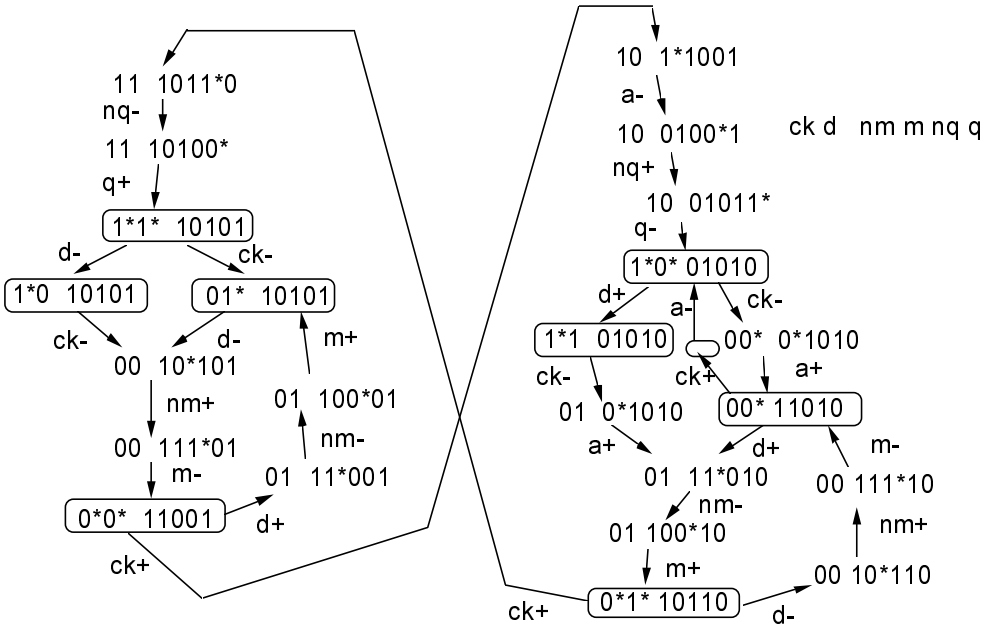


Figure 19. State Graph (SG) of the D-flip-flop

Y	Present States	CK D 00	01	11	10
1	10101	11101	(10101)	(10101)	(10101)
11	11101	11001	-	-	-
2	11001	(11001)	10001	-	01001
12	10001	-	10101	-	-
13	01001	-	-	-	01011
31	01011	-	-	-	01010
3	01010	11010	11010	(01010)	(01010)
21	11010	(11010)	10010	-	01010
14	10010	-	10110	-	-
4	10110	11110	(10110)	10100	-
15	11110	11010	-	-	-
16	10100	-	-	10101	-
	A B NB NQ Q				

nm

m

$$\overline{nm} = ck*m + d*a$$

$$\overline{nq} = ck*m + a*q$$

$$a = \overline{ck*nm}$$

$$m = \overline{nm}$$

$$q = \overline{nq}$$

Figure 20. Flow Table and Logical Equations of the D-flip-flop

This D flip-flop is clocked only by the uncomplemented variable CK, and not by both CK and CKbar (TSPC feature [10]). Figure 21 shows a branch-based [9] transistor implementation and an equivalent logic diagram with 5 gates. Compared with a C^2MOS gate implementation (Fig. 6), this single phase D-flip-flop presents a reduction of switched clock capacitance by a factor of 1.7, as only six transistors are driven by the clock signal. This circuit has been used in a low-power standard cell library [9]. Its power consumption is lower by a factor of 2 to 4 than that of flip-flops from commercial libraries. A phase-frequency detector (PFD) for both PLL of the StrongArm [8] and Alpha microprocessors [11] using a similar D-flip-flop with Reset and D=1 has also been designed.

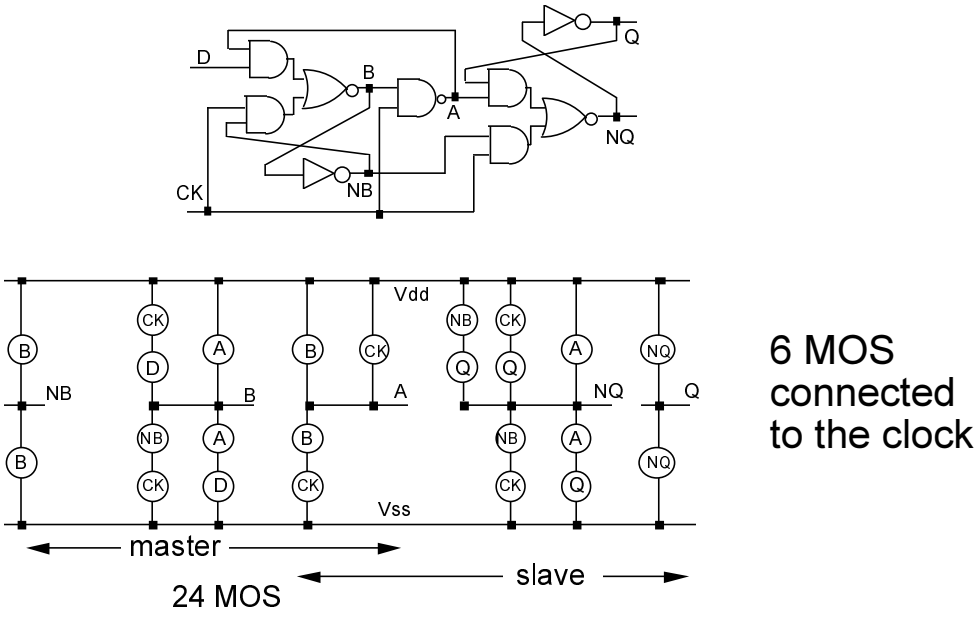


Figure 21. Logical and branch-based structure of the D-flip-flop

Our second example is a half-handshake circuit [12], for which the corresponding STG is given in Figure 22. The STG contains an input conflict that can be solved by complementing the input Ain (method 3.9). Figure 22 shows the modified STG with alternate up and down transitions obtained by introducing internal variables Ab and Rb. The internal variable Ab- is used instead of Aout+ to solve the input conflict, which is possible as Ab is the complement of Aout. According to 3.8, the variables (Ab, Aout) and (Rb, Rout) can be implemented by sm memory elements. The corresponding circuit based on set-memory elements and its flow table are shown in Figure 23. It has less transistors and is faster than the circuit build with C-elements, with 8 gates and 42 transistors [12], that has been analyzed in [7] (non race-free, but without critical race).

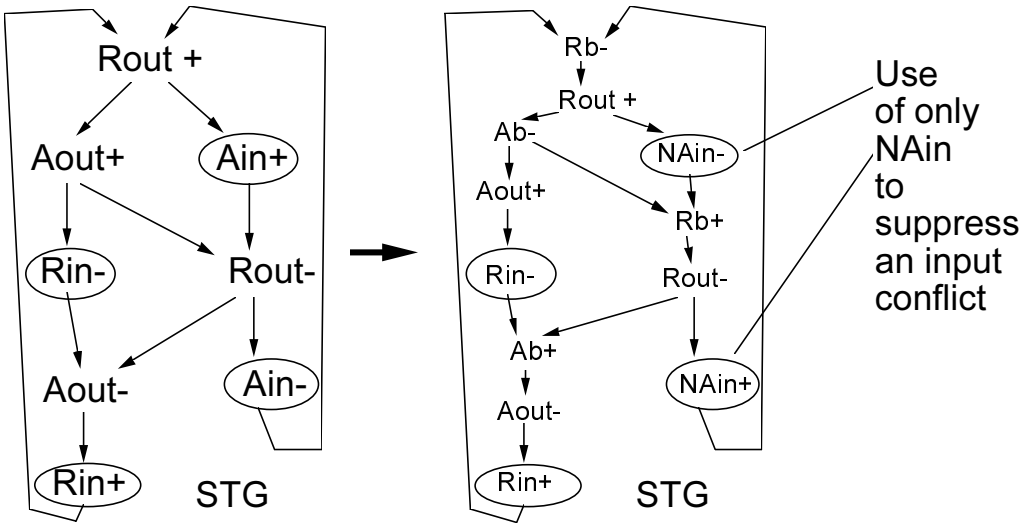


Figure 22. STG of the Half-Handshake Circuit

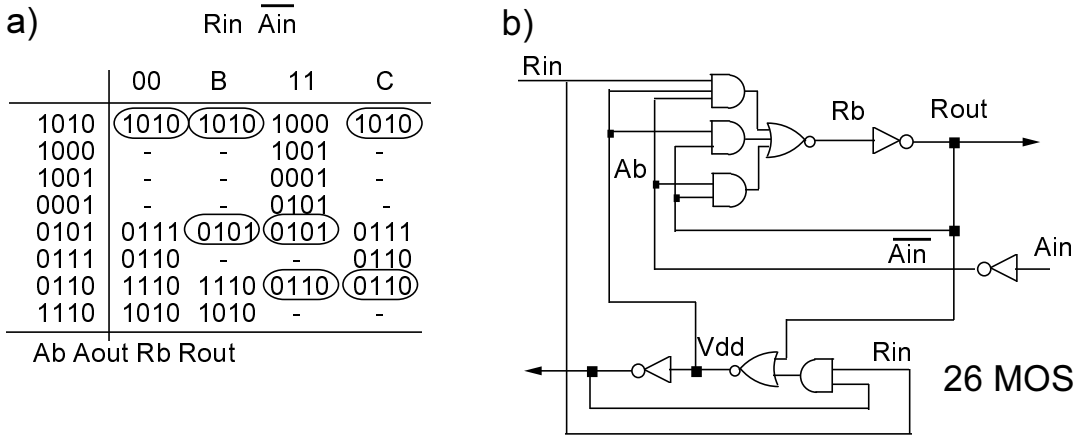


Figure 23. Half-Handshake Circuit.

5. How Many D-flip-flop Structures with 5 negative gates ?

A systematic procedure can be applied to generate all the D-flip-flop structures based on 5 negative gates. These structures have already been synthesized starting from flow tables [13]. However, in this section, the proposed procedure starts from the STG of Figure 24. This STG contains all the necessary transitions labelled a, b, c, ... k, i, which are necessary to guarantee that it is implementable by negative gates, except for Rule 3.6, as there are 5 rising transitions and 4 falling transitions. The method is based on finding which transitions can be selected to form 5 pairs of transitions (x+ and y-) for which each transition appears only in one pair.

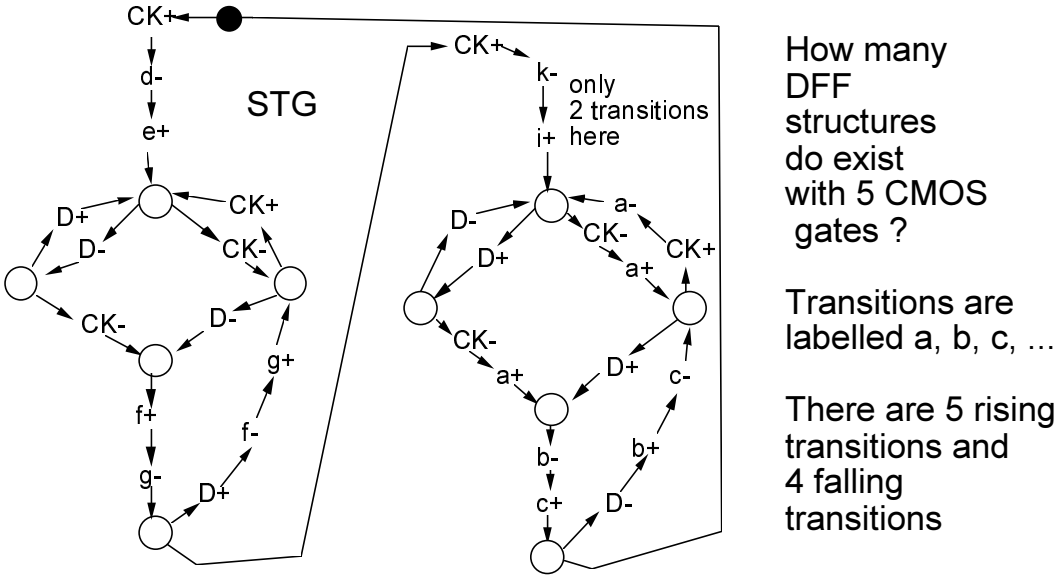


Figure 24. D-flip-flop STG with non assigned transitions.

The first thing to do is to locate the 10th transition (which is a falling j- transition) in the STG. It can be located in 3 different places (after a rising transition), according to the STG of Figure 25.

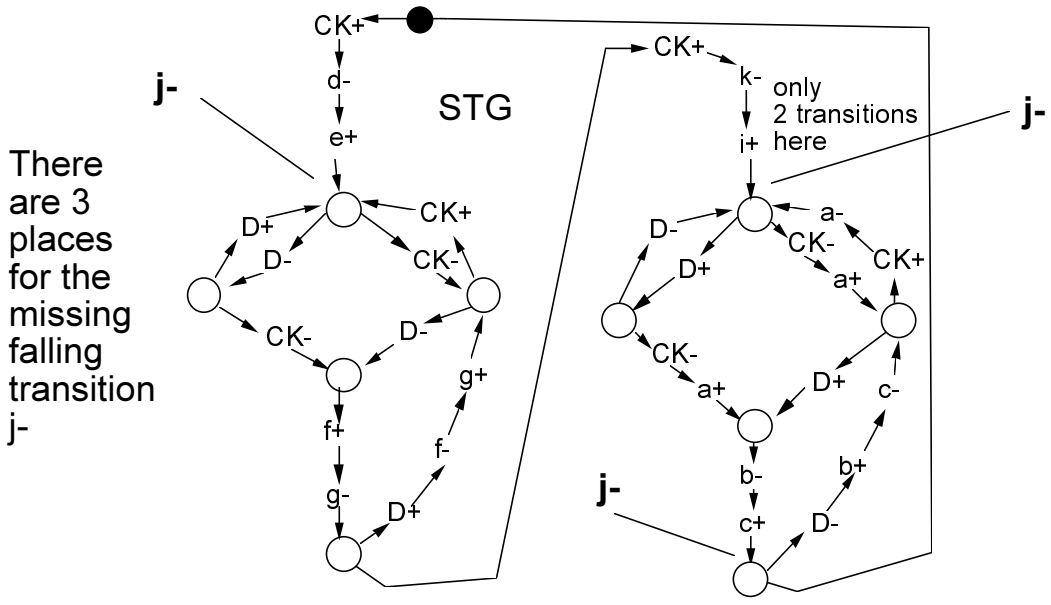


Figure 25. Location of the missing j- transition.

The next step is to search for all the transitions pairs that satisfy to the properties of negative functions (assuming, for instance, that the j- transition is located after the transition i+). While using Rule 3.7, the transition a+ can be associated to d-, g- or k- (Fig. 27) as there are always two other transitions in between. The transition b- can be associated to f+ or i+, but cannot be associated to e+. There are two other transitions (c+, e-) in between in the main loop, but if one consider the feedback loop D-, b+ and c-, there are only the transition d- (after CK+) between the b+ transition in the feedback loop and e+ in the main loop. According to Figure 26, this association is not valid for an implementation with negative functions (proof in [13] while using flow tables). If e+=b+ in Figure 26, one state in the feedback loop starting with D- has an order relation with the state reached with the e+ transition. This order relation results in the fact that the STG cannot be implemented with only negative gates.

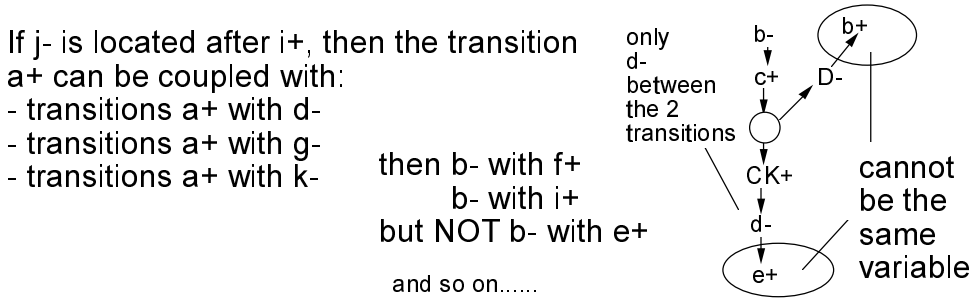


Figure 26. Supplementary rule

Figure 27 shows all the pairs of transitions as well as the D-flip-flop structures that have been found in [13] also with another location of the j- transition. All these structures contain more transistors than the structure presented above (Fig. 21).

To solve the input conflict, it is also possible to add a supplementary variable after the input D+, as shown in the STG of Figure 28.

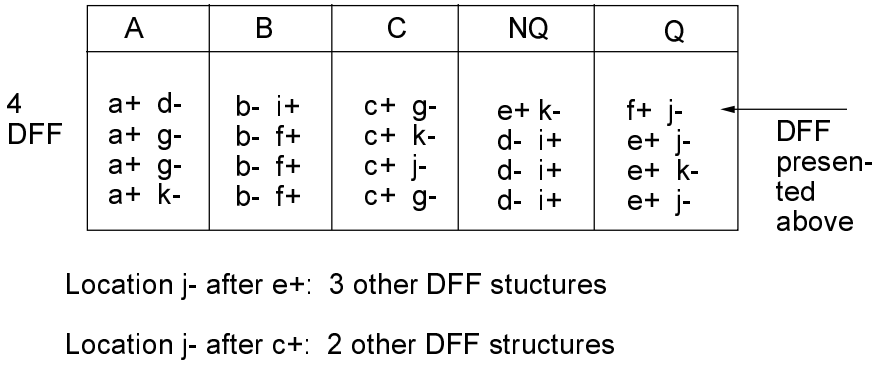


Figure 27. Pairs of transitions to generate the 5 negative gates.

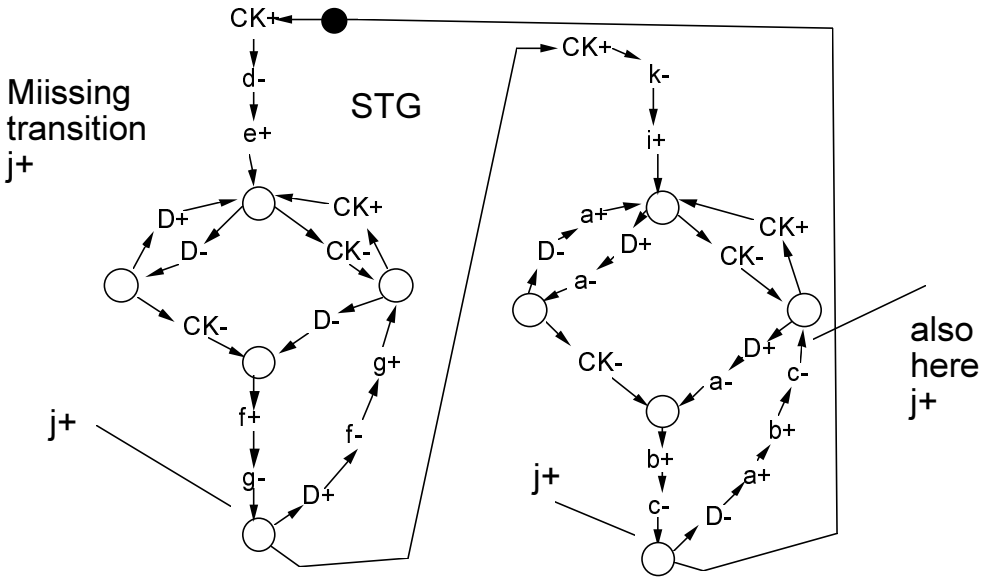


Figure 28. D-flip-flop STG with a- after D+

The same procedure can be applied to find the resulting D-flip-flop structures. As shown in Figure 29, only two structures have been found, the most interesting being a structure with 8 MOS connected to the clock, but in which no CMOS gate is switching with CK if D=0 or D=1 (for the D-flip-flop of Figure 21, the NAND gate A is switching with the clock if D=0).

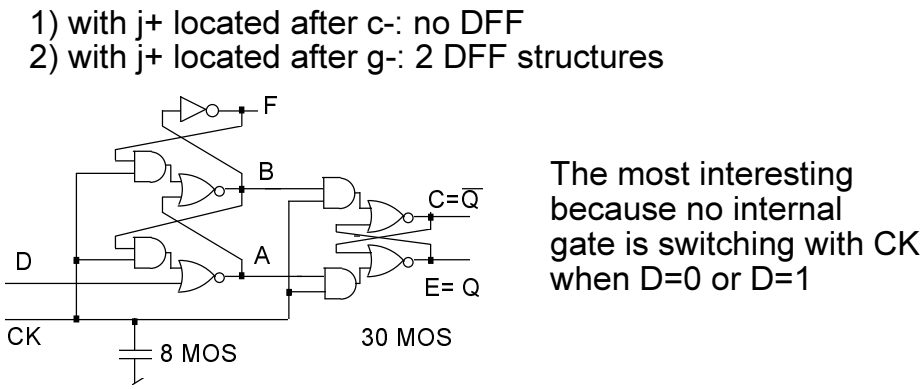


Figure 29. Another D-flip-flop structure

The total number of D-flip-flop structures with 5 negative gates is 11, plus one structure with 4 negative gates.

6. STG-based Synthesis of Latches

Figure 30 shows the STG of a Latch. Such a latch also presents an input conflict in the right part of the STG.

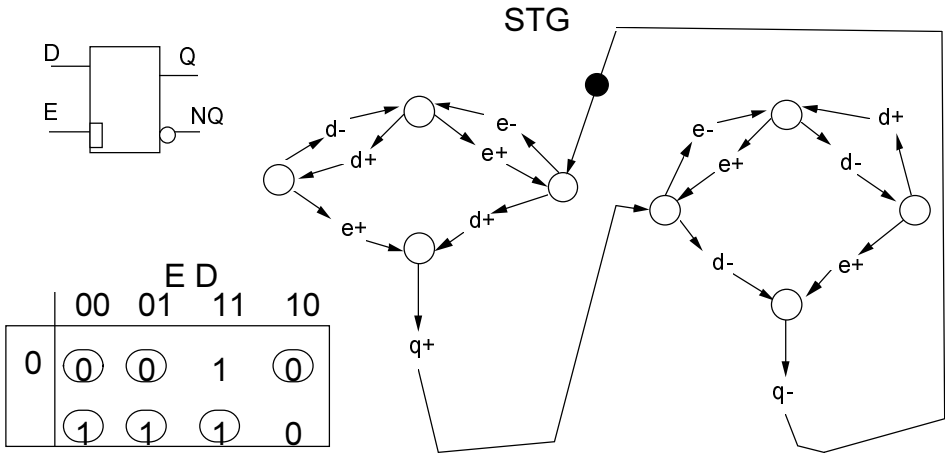


Figure 30. STG of a Latch

Figure 31 shows the way it is generally solved by using the complemented clock Ebar (the right part of the STG contains Ebar). It is possible to have nq- followed by q+ in the left part of the STG and nq+ followed by q- in the right part of the STG, so one generates a sm memory element. The resulting structure is the well-known C^2MOS latch that contains a non-critical race.

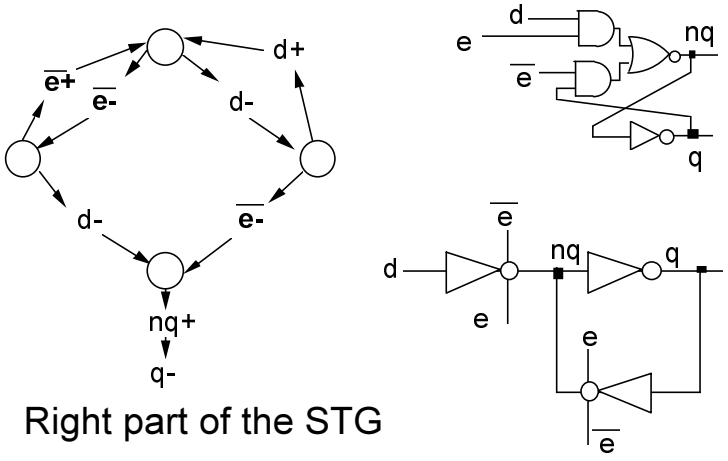


Figure 31. Latch with E and Ebar

Obviously, there is another way to solve the mentioned input conflict, i.e. to use both true and complemented D input. Figure 32 shows also the right part of the STG with a complemented D input. In this case, the transition sequence is q- followed by nq+, and one has a sr memory element. The resulting structure is also the well-known latch shown in Figure 32. It contains also a race, but it is not critical.

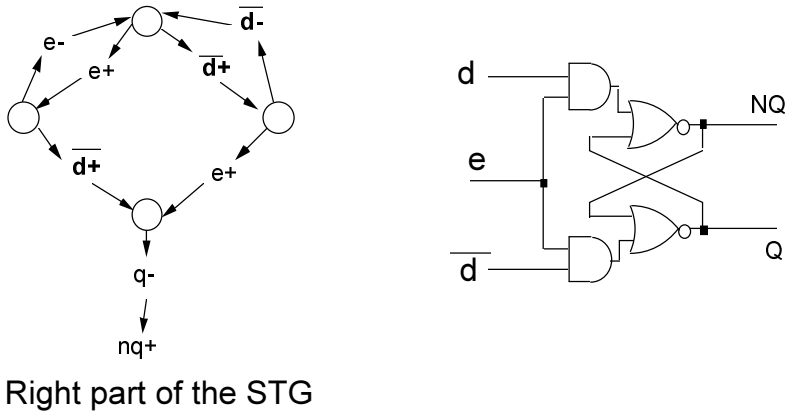


Figure 32. Latch with D and Dbar

Figure 33 shows a decreasing STG of the latch with the introduction of a supplementary variable A used to solve the input conflict in the right part of the STG. One can see that Q and NQ are a sm memory element.

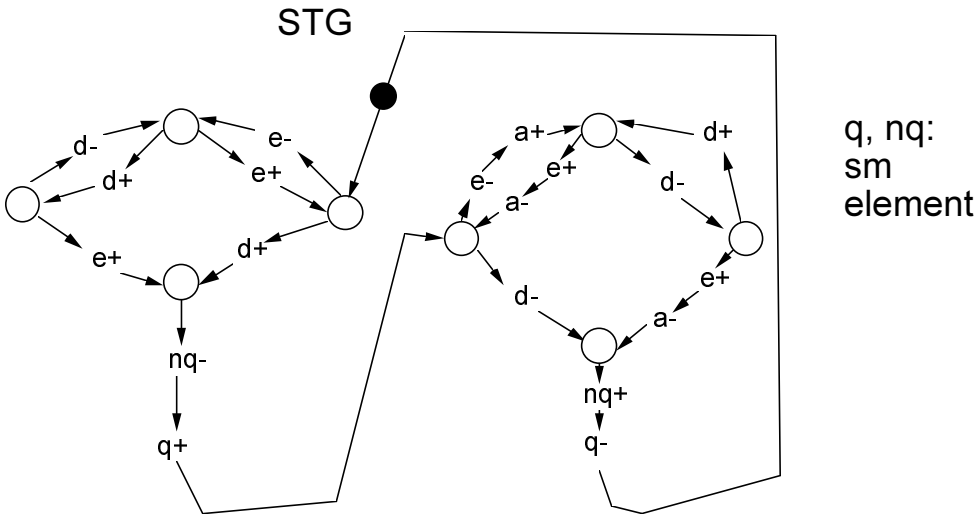


Figure 33. Decreasing STG of a Latch with a supplementary variable A

The resulting structure is shown in Figure 34. One can see the sm memory element for NQ and +Q as well as the supplementary gate A. It can be implemented with 13 MOS transistors and has been patented.

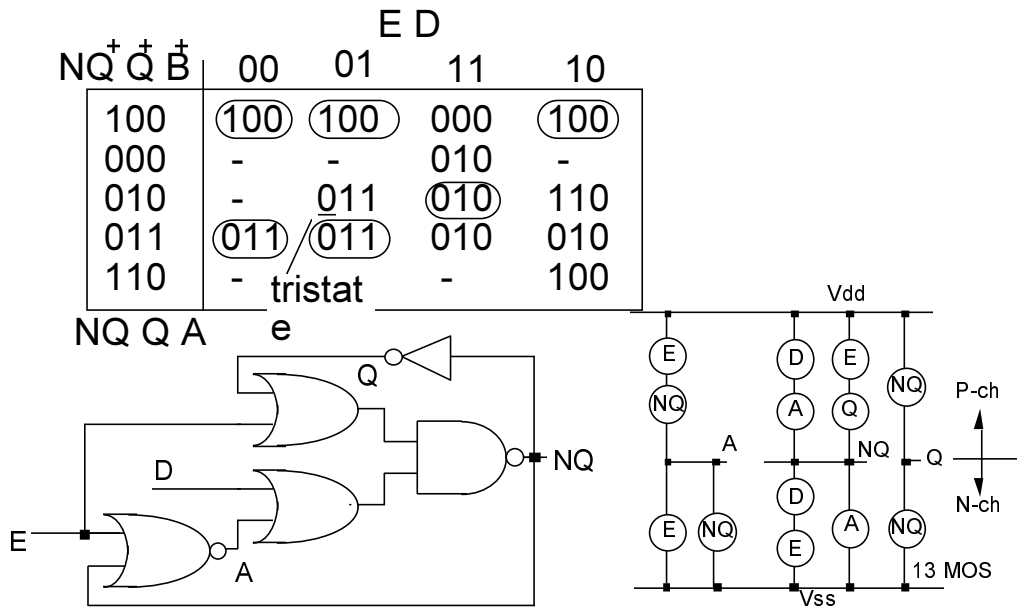


Figure 34. Structure of a Latch

7. Speed-independent/low-power cells for cell libraries

The implementations of D-flip-flops, frequency dividers [6,7] and latches that result from our method contain no input or clock inverter, contrary to most current implementations (C2MOS, for instance). The use of a clock inverter (Figure 6) gives rise to a well-known critical race [4]: if the clock inverter has a large delay, both the master and the slave of the divider (or flip-flop) can be simultaneously transparent, resulting in a fatal failure (the circuit is not speed-independent). However, generally the inverter delay is shorter than the internal gate delays.

Experience shows that it could be dangerous to design basic cells which are not speed-independent, i.e., in which inverter delays are considered as negligible:

- the input and clock inverters can be loaded by very large capacitances if they are connected to many gates of the circuit. Furthermore, the routing capacitance can also be very large if there are many long and short connections. This problem is more and more important in deep submicron technologies.

- if the circuit (state machine) is realized with a standard cell library, the input inverters are also standard cells and will be placed and routed like any other cell. It cannot be predicted what are their delays mainly due to the routing. Standard cell designs have to be critical race-free.

- for very low supply voltage, the delay variations due to the voltage threshold voltages could be very large [9]. It can be shown that if the clock inverter delay at high voltage is generally smaller than the master latch delay, it could be the contrary at very low voltage (less than 1 Volt) [9].

- for the cell design itself, i.e. people designing for instance flip-flops for cell libraries, it is much more difficult to design cells containing a critical race. Obviously, one has to control the race, so one has, for instance, to keep an internal delay quite long. It could be at the price of a slower cell if this internal delay is on the critical path of the cell. Or if a fast cell is mandatory, it means that the input inverter must be faster, requiring very large transistors that increase the power consumption.

- transitions between stable states of the flow table can be performed by 2 internal variables (memory elements). It is not the case if the circuit contains a clock inverter which has to switch before the two variables of the memory element, resulting in three transitions in series and a slower circuit.

- implementations without clock inverter are also better for power. If a clock inverter is needed, the circuit consumes more power as the clock inverter always switches between stable states. The clock inverter presents a 100% activity compared to 50% for the internal gates.

Conclusion

The proposed method ensures race-free operation and speed-independence at the implementation level while taking into account all the delays including input inverter delays. It is crucial for the design of low-power and fast library cells in submicron technologies for which it is very difficult to perform a transistor sizing if the cell presents a critical race that has to be controlled. Furthermore, the resulting circuits contain less transistors than those produced by other methods [1, 2, 3, 5]. Switching activity is also reduced, as the switching between two stable states is performed by a minimal number of variables without clock inverter. The proposed method is therefore the key for the design of low-power library cells.

References

[1] A. Kondratyev et al. "Technology mapping for Speed-Independent Circuits: Decomposition and Resynthesis", ASYNC'97, Eindhoven, The Netherlands, pp. 240-253.

[2] L. Lavagno et al. "Algorithms for Synthesis and Testing of Asynchronous Circuits", Kluwer Academic Publishers, 1993.

[3] A. Kondratyev et al. "Basic Gate Implementation of Speed-Independent Circuits", 31st Design Automation Conf. DAC, 1994, pp. 56-62.

[4] C. Piguet, "Supplementary Condition for STG-designed Speed-Independent Circuits", 2nd April 1998, Vol. 34, No 7, pp. 620-622.

[5] P. Vanbekbergen et al. "A Generalized State Assignment Theory for Transformations on Signal Transition Graphs", Proc. International Conf. on Computer-Aided-Design ICCAD, pp. 112-117, 1992.

[6] C. Piguet, "Logic Synthesis of Race-Free Asynchronous CMOS Circuits" IEEE JSSC-26, No 3, March 1991, pp. 271-380.

[7] C. Piguet, "Synthesis of Asynchronous CMOS Circuits with Negative Gates", Journal of Solid-State Devices and Circuits, Vol. 5, No 2, Brazilian Microelectronics Society, July 1997, pp. 12-20.

[8] C. Piguet, V. von Kaenel, "Logic Synthesis of a PLL Phase Frequency Detector", IEE Proceedings Computers and Digital Techniques, Vol. 144, No 6, November 1997, pp. 381-385.

[9] C. Piguet et al. "Low-Power Low-Voltage Digital CMOS Cell Design", PATMOS'94, Oct. 17-19, Barcelona, Spain, pp. 132-139.

[10] J. Yuan, C. Svensson, "High-Speed CMOS Circuit Technique", JSSC SC-24, pp. 62-70, 1989.

[11] V. von Kaenel et al. "A 600 MHz CMOS PLL Microprocessor Clock Generator with a 1.2 GHz VCO", ISSCC'98, San Francisco, February 5-7, 1998, Conf. 25.1

[12] T. H.-Y Meng et al. "Automatic Synthesis of Asynchronous Circuits from High-Level Specifications", IEEE Trans. on CAD Vol. 9, No 11, November 1989, pp. 1185-1205.

[13] C. Piguet, "Synthèse de Systèmes Logiques Asynchrones à l'aide des Propriétés des Fonctions Logiques Décroissantes", Ph. D. Thesis, No 395, Ecole Polytechnique Fédérale de Lausanne, Switzerland, 1981.

[14] N. Starodoubtsev, "Autonomous Antitone Sequential Circuits, I. Definitions and Interpretation, II, Cyclograms and their Properties", Izv. AN SSSR Tekhn. kibernet. (Engineering Cybernetics), No 4, 1981 and No 5, 1981

[15] N. Starodoubtsev, "Asynchronous Processes and Antitonic Control Circuits, I. description Language, II. Basic Properties, III. Realization", Soviet Journal of Computer & System Science, Vol. 23, pt 2, pp. 112-119, 1985 and Vol. 23, pt 6, 1985, and Vol. 24, pt 2, 1986.

[16] A. Yakovlev, N. Starodoubtsev, private communications.

[17] J. Cortadella et al. "Petrify: A Tool for Manipulating Concurrent Specifications and Synthesis of Asynchronous Controllers", IEICE Trans. Inf & Syst. Vol. E80-D, No 3, March 1997, pp. 315-325.

[18] M. Afghazi, "A Robust Single Phase Clocking for Low Power, High-Speed VLSI Applications", IEEE JSSC Vol. 31, No2, February 1996, pp. 247-254.

[19] J. Yuan, C. Svensson, "New Single Clock CMOS Latches and Flip-Flops with Improved Speed and Power Savings", IEEE JSSC, Vol. 32. No 1, January 1997, pp. 62-69.