ACiD’2002, Münich, Germany, January 28-29, 2002

Design Flow and CAD tools for asynchronous design of sequential library cells

Cédric Cuche (1), Christian Piguet (1, 2), Vojin G. Oklobdzija (3)

1. LAP-EPFL, Lausanne, Switzerland
2. CSEM, Neuchâtel, Switzerland
3. University of California Davis, CA 95616 U.S.A.

Abstract

The design of “small” Speed-Independent (SI) sequential library cells, such as Flip-Flops and master-slave latches, can be performed using STG-based (Signal Transition Graph) methodologies. Starting from a STG that is adapted to produce only negative logical equations, this method has been used for manual design of many basic cells. However, a more automatic approach could be beneficial. CAD tools and a design flow on PC Windows implementing this methodology will be presented, including existing tools such as Petrify, as well as new CAD tools that are capable of verifying if the produced cells are SI or not. The logical equations of the considered cell or the logical equations of the N-ch and P-ch networks are analysed to produce the flow table of the cell as well as an analysis of the possible critical races. In case of critical races, the considered cell is not SI. The design flow is not completely automatic, as logical equations produced by Petrify from a “negative” STG are not always in suitable forms.

The presented design flow has been used for the design of several different basic cells, including Flip-Flops and master-slave latches and other various “small” asynchronous Finite State Machines. The presented CAD tools are also interesting in case of critical races, i.e. in case of non-SI circuits. For some circuits, the transistor schematic is simpler and faster than the SI circuit implementing the same behaviour. The complete description of the race between two logical gates switching at the same time is available, for both cases in which the first or the other logic gate is faster than the other one. It is therefore possible to check the conditions on the gate delays for a correct behaviour.

A special non-SI master-slave latch circuit, i.e. containing critical races, will be presented as an example. This circuit has been patented, and has been analyzed and compared with nearly all the other master-slave latch types in very deep submicron technology. The results show that it is one the best master-slave latch structure if only pure and robust static logic is considered.

1. Introduction

The design of digital library cells has received much attention due to the very deep submicron technologies for which speed and power performances are mandatory, but also robustness. Asynchronous design methodologies that claim to be "Delay or Speed Independent (DI or SI)" could be more and more important if robustness is considered. Any library contains master-slave latches and many publications present new structures with improved performances. It is therefore interesting to propose asynchronous design methodologies as well as CAD tools capable of producing, in addition to finite state machines, such master-slave latches.

A design methodology for finite state machines, including master-slave latches, has been proposed several years ago. It is based on STG (Signal Transition Graph). A first method has been implemented by Petrify [1, 2] and produces SI circuits. Another method based on negative gates has been proposed for CMOS SI circuits without any CAD tool [3, 4, 5]. It is the goal of this paper to present a design flow based on this method but using also Petrify and some in-house CAD tools for a semi-automatic design. Two design examples, which are master-slave latches, will be presented as well as some problems in the Petrify tool discovered during the test phase.

2. Proposed Design Methodology

The proposed methodology contains different phases (Figure 1):

1) Any cell or finite state machine behaviour is specified *manually* through a STG [1, 2]. Not every STG specification is suitable for synthesis in a speed-independent asynchronous circuit. In order to be a SI specification, the STG must satisfy the following properties, i.e. boundness, complete state assignment, complete state coding, persistency and commutability. As shown in Figure 2, this can be done while using Petrify.

2) The STG is *manually* modified according to the method proposed in [3, 4, 5] to produce a "negative" STG that is implementable only with negative gates [6]. Negative gates are the basic building blocks in CMOS circuit design. Figure 2 in the middle shows a modified STG for a master-slave latch.

3) A STG implementable with only negative gates must also satisfy to properties mentioned in [3, 4, 5], i.e. to present alternate transitions, not to have input conflicts, to respect the cycle rule, to memorize successive changes of two inputs (at that time, we have no proof that these properties are sufficient to always generate a SI negative circuit!). The VerifySTG tool allows the designer to check if his STG is negative (Figure 2).

4) From this negative STG, the designer can generate *manually* the logical equations of the cell or he can use Petrify to generate the corresponding equations. All the logical equations are negative. One can also generate manually the logical equations of the N-ch and P-ch

networks of each negative gate. The latter is a MOS transistor level description. In Figure 1, one can see two different implementations, i.e. logical equations on the left and transistor N-ch and P-ch networks on the right.

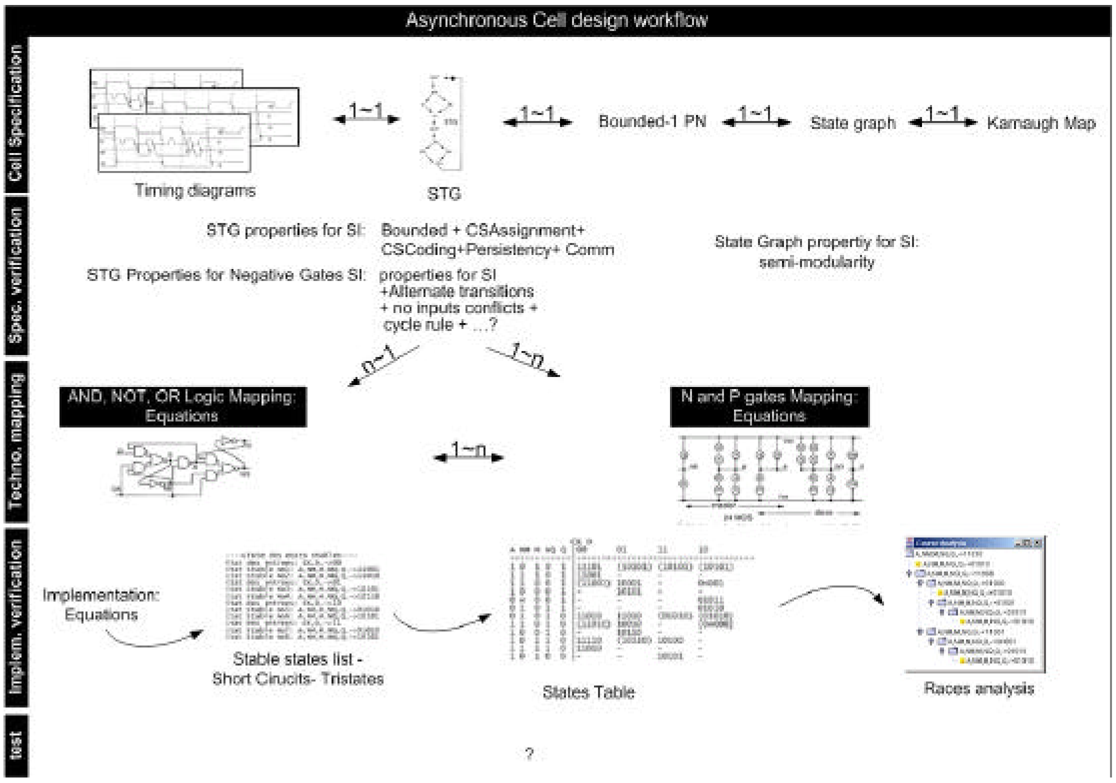

Asynchronous Cell design workflow
Cell Specification
Spec. verification
Techno. mapping
Implem. verification
test
Timing diagrams
1~1
STG
Bounded-1 PN
State graph
Karnaugh Map
STG properties for SI: Bounded + CSAssignment+ CSCoding+Persistency+ Comm
STG Properties for Negative Gates SI: properties for SI +Alternate transitions + no inputs conflicts + cycle rule + ...?
State Graph propertiy for SI: semi-modularity
n~1
1~n
AND, NOT, OR Logic Mapping: Equations
N and P gates Mapping: Equations
Implementation: Equations
Stable states list - Short Ciruits- Tristates
States Table
Races analysis
?

Figure 1. Complete Design Flow

5) The last step consists in verifying that the CMOS implementation is a SI implementation. An in-house tool (called Alcyon) generates the flow table [3] of the considered circuit, as well as the list of the stable states and the list of the races if any (Figure 1). A race occurs if two (or more) CMOS gates are switching at the same time after an input or another gate activation. The tool considers both situations in which the first gate is faster than the second one and vice-versa. If the two paths finally join the same stable state, the race is not critical and the circuit is SI (the circuit behaviour does not depend on the fact that the first gate is faster or not than the second one). If the two paths end up to two different stable states, the race is critical and the circuit in not SI (the circuit behaviour depends on the fact that the first gate is faster or not than the second one).

Figure 2 contains the additional steps (2 and 3) to a SI design flow in order to design SI CMOS cells.

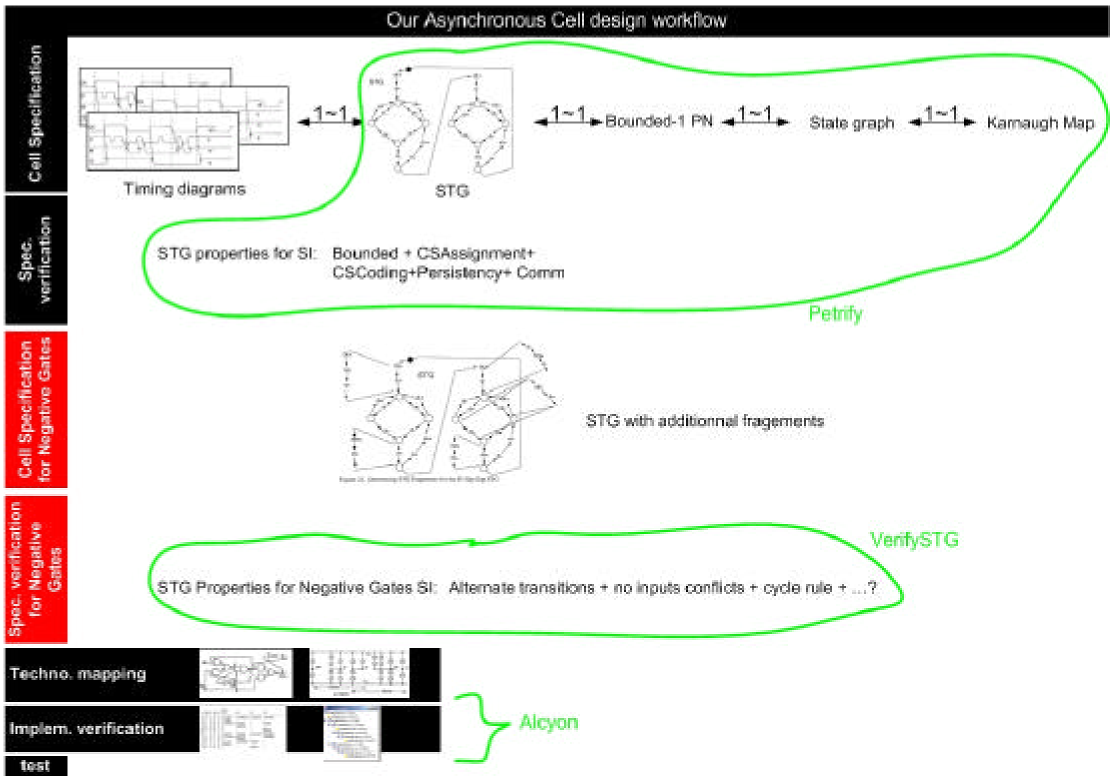

Our Asynchronous Cell design workflow
Cell Specification
Spec. verification
Cell Specification for Negative Gates
Spec. verification for Negative Gates
Timing diagrams
1~1
STG
Bounded-1 PN
State graph
Karnaugh Map
STG properties for SI: Bounded + CSAssignment+ CSCoding+Persistency+ Comm
Petrify
STG with additionnal fragements
VerifySTG
STG Properties for Negative Gates SI: Alternate transitions + no inputs conflicts + cycle rule + ...?
Techno. mapping
Implem. verification
test
Alcyon

Figure 2. STG design and verification

3. CAD Tools

The tools that have been designed are the following:

- The first one (VerifySTG) use Petrify to check SI properties of an STG and to generate logical equations. It also checks STG properties for a negative implementation of the cell (Figure 2).
- The second tool (Alcyon) is used to check that equations are in negative form. Then it performs a complete analysis of the cell and extracts all stable state, shortcuts between Vdd and Vss, tri-states and floating states in the cell implementation. It prints a flow table for all reachable states (stable or transient) and gives the list of all the races if any. The designer can then look at the execution of any race in order to check if the race is critical or not (Figure 1).

All these tools run on Window2k at least. They are written in Java2. Petrify properly runs on this platform as it is used in VerifySTG. Alcyon logic uses parts of VHDL IEEE1164 bit library translated in Java. Alcyon supports up to 65'000 variables per logic equation.

4. Example 1: SI master-slave latch

Figure 3 shows the STG of a master-slave latch [7] (called D-Flip-Flop in [4, 5]). This STG can be verified while using VerifySTG and the Petrify coding format shown in Figure 4.

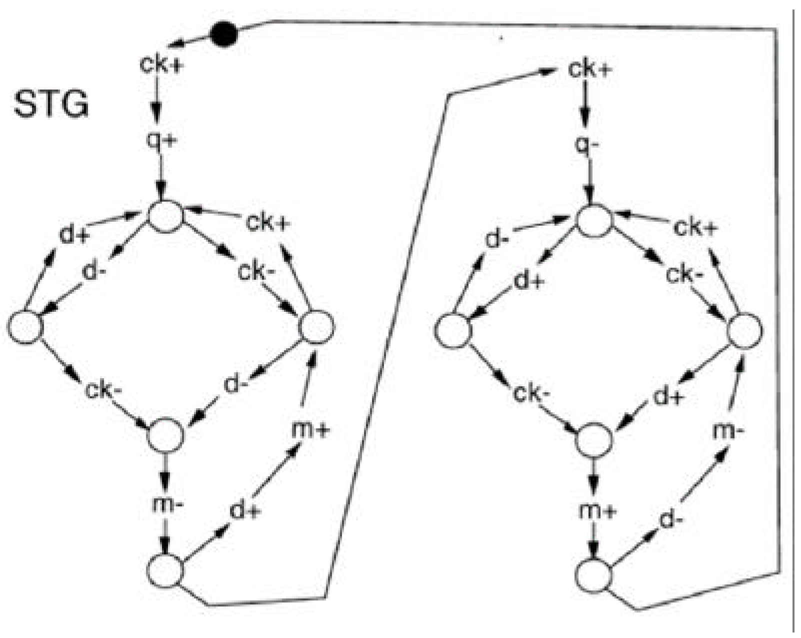


Figure 3. STG of a master-slave latch

```
#D-Flip-Flop (p.4 from St-Petersbourg 98, ref. [5])
.model D-Flip-Flop
.inputs ck d
.outputs q
.internal m
.dummy dum
#--Transition graph definition
.graph
ck+/4 q+
q+ p0
p0 d-/1 ck-/1
d-/1 p1
p1 d+/1 ck-/2
d+/1 p0
ck-/1 p2
p2 ck+/1 d-/2
ck+/1 p0
ck-/2 p3
d-/2 p3
p3 m-/1
m-/1 p4
p4 d+/2 ck+/2
d+/2 m+/1
m+/1 p2
ck+/2 q-
q- p5
p5 d+/3 ck-/3
d+/3 p6
ck-/3 p7
p6 d-/3 ck-/4
d-/3 p5
p7 ck+/3 d+/4
ck+/3 p5
ck-/4 p8
d+/4 p8
p8 m+/2
```

```
m+/2 p9
p9 d-/4 dum
d-/4 m-/2
m-/2 p7
dum p10
p10 ck+/4
.marking{p10}
.end
```

Figure 4. Input file for Petrify of the STG shown in Figure 3.

Every signal transition that appears more than once in the STG is followed by an attribute "/number" in order to reference them uniquely. Arcs are defined with two signal names on a line. The arc origin is the first signal name. A place (circle in the STG) definition is a line with more than two signal names. First name is the place name.

Petrify checks that this STG has the mandatory CSC property for SI implementation. Resulting equations are:

```
INORDER = ck d q m;
OUTORDER = [q] [m];
[q] = ck m + q ck';
[m] = ck' d + ck m;
```

These logical equations are not negative and this structure is not SI [9] due to the clock inverter delay. Figure 5 is an output file from VerifySTG. The rules to get a negative STG are not satisfied. ATe mean "alternate transition rule violation". These faulty STG fragments have to be modified according rules defined in [4, 5]. ICe mean "input conflit rule violation". Adequate transformations are also described in [4, 5]. Note that this STG has no "cycle rule violation". After rework, as shown in Figures 6 and 7, a negative STG is eventually manually defined.

```
---Check for alternate transition rule---
ATe: t:d+/2->t:m+/1
ATe: t:d-/4->t:m-/2
ATe: t:ck+/4->t:q+
ATe: t:d-/2->t:m-/1
ATe: t:ck-/2->t:m-/1
ATe: t:ck-/2->t:m-/1
ATe: t:d-/2->t:m-/1
ATe: t:d-/2->t:m-/1
ATe: t:ck-/2->t:m-/1
ATe: t:d-/2->t:m-/1
ATe: t:d-/2->t:m-/1
ATe: t:d+/4->t:m+/2
ATe: t:d+/4->t:m+/2
ATe: t:d+/4->t:m+/2
ATe: t:d+/4->t:m+/2
ATe: t:d+/4->t:m+/2

---Check for input conflicts rule---
ICe: t:ck-/4,t:d+/4 -> t:m+/2
ICe: t:d+/4,t:ck-/4 -> t:m+/2
```

```
---Check for cycle rule---
rule 3.6 ok
****************************
Please rework your STG
****************************
```

Figure 5. Output file form VerifySTG

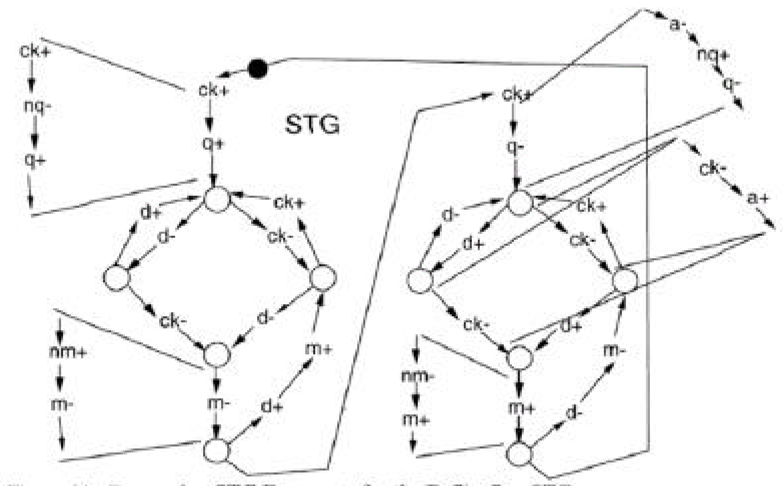


Figure 21. Decreasing STG Fragments for the D-flip-flop STG

Figure 6. Negative STG fragment transformation for a master-slave latch

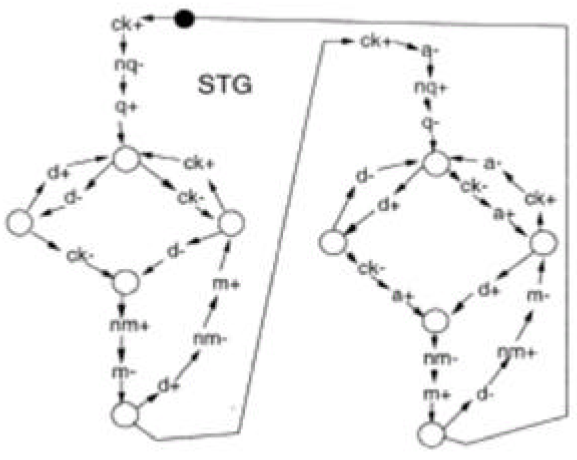


Figure 7. Negative (or decreasing) STG for a master-slave latch

From the negative STG of Figure 7, the designer has two ways to generate the corresponding implementations:

- to use Petrify for the logical equations generation
- to manually design the logical equations (gate level or P-ch/N-ch networks) from the STG or the corresponding flow table

The logical equations provided by Petrify are shown in Figure 8. Hopefully, as the STG of Figure 7 is negative, the logical equations provided by Petrify are all negative (all variables are complemented). They have to be manually slightly modified according to Figure 9 to be used as an input file for the verification tool named Alcyon. Alcyon is capable of generating the corresponding flow table and of checking races.

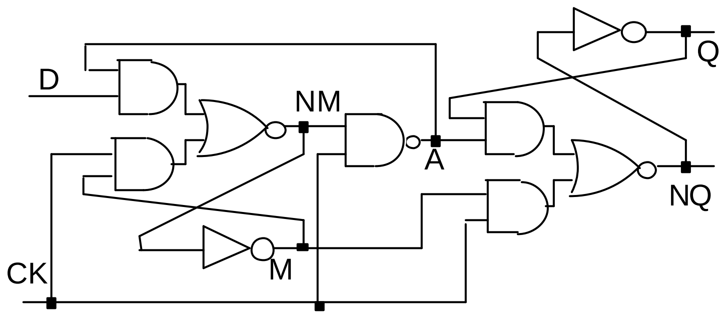
```
# EQN file for model datas/FlipFlops/CSC-D-flip-flop.stg
# Generated by 4.2 (compiled Fri Nov 30 16:02:46  2001)
# Outputs between brackets "[out]" indicate a feedback to input "out"
# Estimated area = 12.00
INORDER = ck d q nq nm;
OUTORDER = [q] [nq] [nm];
[q] = nq';
m = nm';
[nq] = nq (m' + ck') + a';
[nm] = d' (nm + ck') + a';
a = nm' + ck';
# Set/reset pins: set(nq)
```

Figure 8. Logical Equations generated by Petrify

```
##D-Flip-flop
#
# format normal (sortie de petrify: CSC-D-flip-flop.eqn)
##
A  = NM'+CK';
NM = (D'*(M'+CK'))+A';
M  = NM';
NQ = (Q'*(M'+CK'))+A';
Q  = NQ';
```

Figure 9. Input file for Alcyon

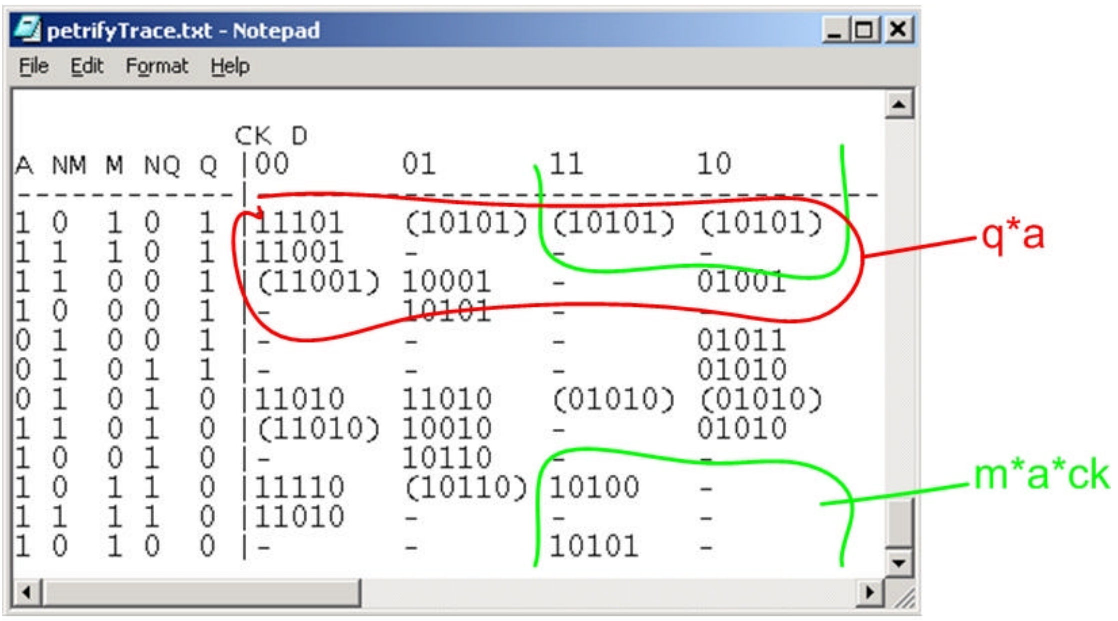
However, the logical equations provided by Petrify are not the minimal logical equations. Figure 10 shows a minimal set of logical equations manually generated from the negative STG of Figure 7. Both sets of equations implement the same behaviour and result in the same flow table (Figure 11). However, as shown in Figure 11, Petrify considers one Karnaugh block as “m*a*ck” while it can be coded as “m*ck”.



```
##D-Flip-flop
#
##
A  = (NM*CK)';
```

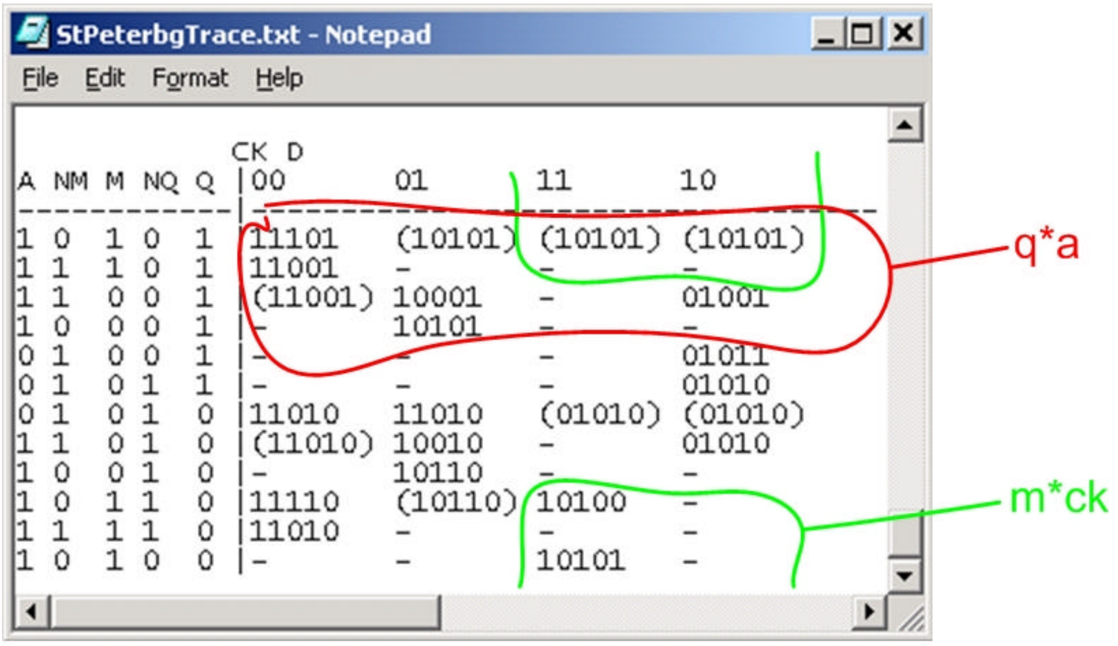
```
NM = ((CK*M)+(D*A))';
M  = NM';
NQ = ((CK*M)+(A*Q))';
Q  = NQ';
```

Figure 10. Manually generated logical equation



```
                 CK  D
A  NM  M  NQ  Q  |00        01        11        10
------------------------------------------------------
1  0   1  0   1  |11101    (10101)   (10101)   (10101)
1  1   1  0   1  |11001     -         -         -
1  1   0  0   1  |(11001)   10001     -         01001
1  0   0  0   1  |-         10101     -         -
0  1   0  0   1  |-         -         -         01011
0  1   0  1   1  |-         -         -         01010
0  1   0  1   0  |11010     11010    (01010)   (01010)
1  1   0  1   0  |(11010)   10010     -         01010
1  0   0  1   0  |-         10110     -         -
1  0   1  1   0  |11110    (10110)    10100     -
1  1   1  1   0  |11010     -         -         -
1  0   1  0   0  |-         -         10101     -
```

NQ = q*a + m*a*ck



```
                 CK  D
A  NM  M  NQ  Q  |00        01        11        10
------------------------------------------------------
1  0   1  0   1  |11101    (10101)   (10101)   (10101)
1  1   1  0   1  |11001     -         -         -
1  1   0  0   1  |(11001)   10001     -         01001
1  0   0  0   1  |-         10101     -         -
0  1   0  0   1  |-         -         -         01011
0  1   0  1   1  |-         -         -         01010
0  1   0  1   0  |11010     11010    (01010)   (01010)
1  1   0  1   0  |(11010)   10010     -         01010
1  0   0  1   0  |-         10110     -         -
1  0   1  1   0  |11110    (10110)    10100     -
1  1   1  1   0  |11010     -         -         -
1  0   1  0   0  |-         -         10101     -
```

NQ = q*a + m*ck

Figure 11. Petrify and Manual Simplifications

The logical equations can also be defined as different logical equations for both N-ch and P-ch networks. Sometimes, the N-ch and P-ch networks, for optimisation purposes, can be chosen as non-strictly dual networks. So Alcyon can analyse such situations.

The output if the Alcyon tool for the equations of Fig. 9 is the following:

- a list of shortcuts and tristate if any (not given to limit size of this paper)
- a list of all stable states (Figure 12)
- the flow table (Figure 13).
- race description.

```
equations:A=((NM)')+((CK)');      NM=(((D)')*(((M)')+((CK)')))+((A)');
M=(NM)'; NQ=(((Q)')*(((M)')+((CK)')))+((A)'); Q=(NQ)';


----Liste des etats stables----
Etat des entrees: CK,D,->00
Etat Stable No1: A,NM,M,NQ,Q,->11001
Etat Stable No2: A,NM,M,NQ,Q,->11010
Etat des entrees: CK,D,->01
Etat Stable No3: A,NM,M,NQ,Q,->10101
Etat Stable No4: A,NM,M,NQ,Q,->10110
Etat des entrees: CK,D,->10
Etat Stable No5: A,NM,M,NQ,Q,->01010
Etat Stable No6: A,NM,M,NQ,Q,->10101
Etat des entrees: CK,D,->11
Etat Stable No7: A,NM,M,NQ,Q,->01010
Etat Stable No8: A,NM,M,NQ,Q,->10101
```

Figure 12. A list of the stable states

```
                CK D
A NM M NQ Q |00      01      11      10
------------|----------------------------------
1 0  1 0  1 |11101   (10101) (10101) (10101)
1 1  1 0  1 |11001   -       -       -
1 1  0 0  1 |(11001) 10001   -       01001
1 0  0 0  1 |-       10101   -       -
0 1  0 0  1 |-       -       -       01011
0 1  0 1  1 |-       -       -       01010
0 1  0 1  0 |11010   11010   (01010) (01010)
1 1  0 1  0 |(11010) 10010   -       01010
1 0  0 1  0 |-       10110   -       -
1 0  1 1  0 |11110   (10110) 10100   -
1 1  1 1  0 |11010   -       -       -
1 0  1 0  0 |-       -       10101   -
```

Figure 13. Flow Table of the master-slave latch

By choosing the stable state number 6 in Figure 12, one will end up with the flow table of Figure 13. The first row of the flow table contains the state number 6, thus facilitating the analysis. States in parenthesis are stable states. There is no indication of race in this flow table (given within brackets […]). It results that the master-slave latch implemented through these equations is speed-independent (SI).

5. Example 2: a non-SI master-slave latch

The second case study aims at showing how races are analysed by Alcyon. The considered cell is also a master-slave latch derived from the logic structure in Figure 10. A careful analysis shows that some gate and transistors could be removed while keeping the same behaviour but introducing critical races. The inverter M in the master (Figure 10) could be suppressed and the NQ gate controlled by A. This master-slave latch has been patented [10]. Figure 14 shows the logical equations as well as the corresponding gate implementation.

```
##D-Flip-flop
#
# Patent-case3
##
A  = (B*CK)';
B  = ((D*A)+(CK*A))';
NQ = ((Q*A)+(CK*A))';
Q  = NQ';
```

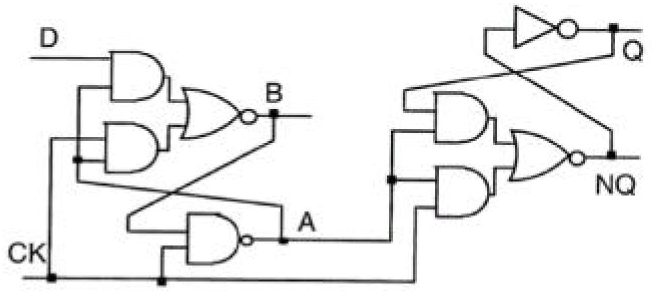


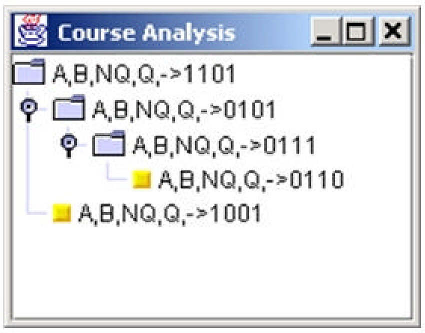
Figure 14. Logical equations and structure of another master-slave latch

```
          CK D
A B NQ Q |00     01     11     10
---------|-----------------------------
1 0 0  1 |1101   (1001) (1001) (1001)
1 1 0  1 |(1101) 1001   -      [0001]
1 1 1  0 |(1110) 1010   -      [0000]
1 0 1  0 |1110   (1010) 1000   -
1 0 0  0 |-      -      1001   -
0 1 1  0 |1110   1110   (0110) (0110)
```

Figure 15. Corresponding Flow Table (with 2 critical races)

Figure 15 shows the corresponding flow table generated by Alcyon. Stable states are between parentheses. Races are indicated between brackets. There are two races in the column CK*D=10 and rows 1101 and 1110. Variables that race against each other can be determined by comparing present state (first column A B NQ Q) to the next states between brackets. For instance, from state 1101 (second row) to [0001], the two first variables A and B are switching simultaneously. The race is between A and B (Figure 16). For such a race, Alcyon will provide a tree of all execution paths, considering both the cases for which A is faster than B and the second case for which B is faster than A. Figure 16 shows the race as provided by Alcyon (left part of the Figure). The race is critical as two different stable states are reached depending on the gate delays: if B is faster than A, the stable state is 1001 and if A is faster than B, the stable state is 0110 (behaviour in Figure 15 should be equivalent to the non critical one in Figure 13). The flow table (Figure 15) shows that the correct stable state has to 0110. So the condition on the gate delays for a correct behaviour is that A MUST be faster than B.

Figure 17 shows the second race, which is also critical. Fortunately, the conditions on the delays for a correct behaviour is the same than previously, i.e. A MUST be faster than B and NQ. Looking at the structure in Figure 14, the variable A can be implemented by a fast NAND gate while the two other gates B and NQ are more complex and consequently naturally slower than A. However, the proposed master-slave latch is a NOT a Speed-Independent (SI) circuit.



```
A,B,NQ,Q,->1101
  A,B,NQ,Q,->0101
    A,B,NQ,Q,->0111
      A,B,NQ,Q,->0110
  A,B,NQ,Q,->1001
```

Root = state with race

Leafs = final race stable state

Nodes = unstable states (either with race or transitory)

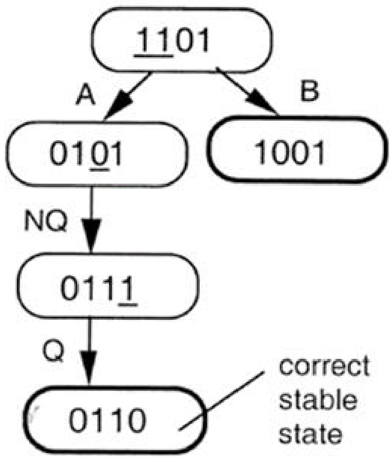
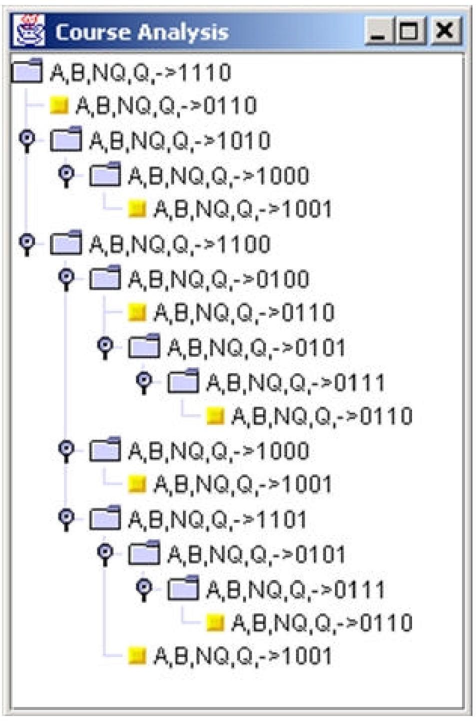


Figure 16. First Critical Race

6. Performances

The master-slave latch described in Figure 14 can be implemented with only 20 MOS transistors for a static structure. As University California Davis has proposed a general framework to compare many different master-slave latches in the same technology [7, 8], the structure of Figure 14 has been compared to the other known structures. Figure 18 shows the results assuming an activity of 50% [7]. Definitions of delay, power and Energy-Delay Product (EDP) as well as references to these various structures are given in [7].



Course Analysis

```
A,B,NQ,Q,->1110
  A,B,NQ,Q,->0110
  A,B,NQ,Q,->1010
    A,B,NQ,Q,->1000
      A,B,NQ,Q,->1001
  A,B,NQ,Q,->1100
    A,B,NQ,Q,->0100
      A,B,NQ,Q,->0110
      A,B,NQ,Q,->0101
        A,B,NQ,Q,->0111
          A,B,NQ,Q,->0110
    A,B,NQ,Q,->1000
      A,B,NQ,Q,->1001
    A,B,NQ,Q,->1101
      A,B,NQ,Q,->0101
        A,B,NQ,Q,->0111
          A,B,NQ,Q,->0110
      A,B,NQ,Q,->1001
```

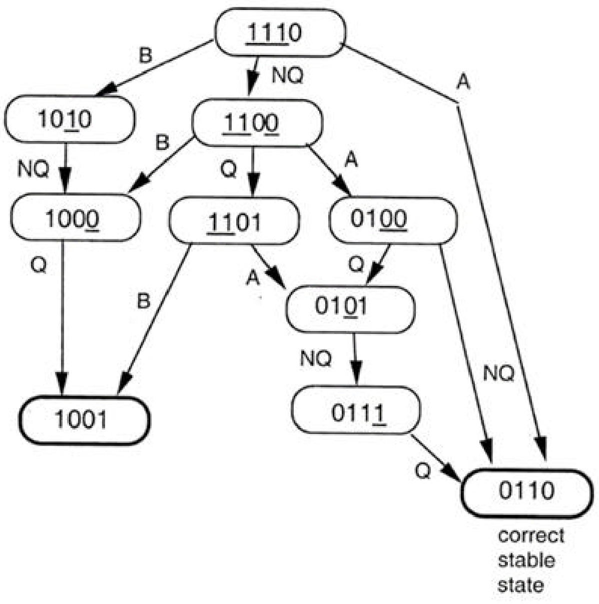


Figure 17. Second Critical Race

The names used are the following [7, 8]:
1) CCFF - Conditional Capture FF
2) CPFF - Conditional Precharge FF
3) imCCFF - improved CCFF
4) HLFF - HLFF (Partovi): hybrid-latch flip-flop, semi-dynamic, single-ended
5) SDFF - SDFF (F. Klass): semi-dynamic flip-flop, single-ended
6) CMOS - C2MOS: single-ended, static
7) PowPC - MS Latch used in PowerPC (Gerosa): single-ended, static
8) This work: single ended, static
9) TSPC - Svenson
10) TGCPFF - Transmission Gate CPFF
11) SSTC – Svenson : master-slave latch, differential
12) DSTC – Svenson : master-slave latch, differential
13) SAFF - Alpha 21264: sense-amplifier flip-flop, differential
14) SABFF - Improved second stage SAFF (patent US 6,232,810)
15) SLFF
16) DE CCFF - Dual Edge CCFF

From this list, if single-ended and static structures are considered (No 6 to 8), the presented master-slave latch can be considered as an interesting structure. Finally, only the sense-amplifier flip-flops present a significant better EDP.

act.=50 %	Delay [ps]	Pi [uW]	Pclk [uW]	Pd [uW]	Ptot [uW]	EDP [fJ@500MHz]
CCFF	246.8	127.3	18.5	7.7	153.4	37.9
CPFF	202.3	117.0	24.4	4.9	146.3	29.6
imCCF	257.4	110.8	10.2	0.7	121.7	31.3
HLFF	187.9	161.3	18.0	4.4	183.8	34.5
SDFF	164	256.0	50.4	4.4	311.4	51.4
C^2MOS	354	110.8	27.5	2.8	141.1	49.9
PowPC	300	80.0	32.1	11.1	123.2	36.9
This	280	87.7	28.4	4.5	120.7	33.5
TSPC	254	97.8	30.1	2.8	130.7	31.7
TGCPF	292	110.5	8.7	9.3	128.5	37.5
SSTC	501	79.8	5.7	1.2	86.7	43.4
DSTC	508	82.5	5.7	1.2	89.4	45.4
SAFF	274	87.0	5.2	1.3	93.5	25.7
SABFF	169	100.8	5.8	1.3	107.9	18.2
SLFF	155	95.9	6.6	1.3	104.0	16.1
DE	169	112.5	17.0	2.6	132.1	22.3

Figure 18. Comparison from University California Davis

7. Conclusion

This paper has presented the tools that are used for the design and analysis of master-slave latches based on STG and negative gates. Although the goal is to design robust speed-independent circuits, often non-SI circuits outperform the previous ones. Such a non-SI master-slave latch has been presented with a comparison with other structures.

8. References

[1] L. Lavagno et al. "Algorithms for Synthesis and Testing of Asynchronous Circuits", Kluwer Academic Publishers, 1993.
[2] J. Cortadella et al. "Petrify: A Tool for Manipulating Concurrent Specifications and Synthesis of Asynchronous Controllers", EICE Trans. Inf & Syst. Vol. E80-D, No 3, March 1997, pp. 315-325.
[3] C. Piguet, "Logic Synthesis of Race-Free Asynchronous CMOS Circuits" IEEE JSSC-26, No 3, March 1991, pp. 271-380.
[4] C. Piguet, J. Zahnd, "Signal-Transition Graphs-based Design of Speed-Independent Basic Cells", PATMOS'98, October 6-8, 1998, Copenhagen, Denmark.

[5] C. Piguet, J. Zahnd, "STG-Based Synthesis of Speed-Independent CMOS Cells", Workshop on Exploitation of STG-based Design Technology, St. Petersburg, Russia, July 6-7, 1998.
[6] C. Piguet, “Logic Design with Negative Gates”, invited paper, 1st Int. Workshop on Multi-Architecture Low Power Design MALOPD, Moscow, Russia, Sept. 13-14, 1999.
[7] V. Stojanovic, Vojin G. Oklobdzija, “Comparative Analysis of Master-Slave Latches and Flip-Flops for High-Performance and Low-Power Systems”, JSSC Vol. 34, No 4, April 1999.
[8] N. Nedovic, M. Aleksic, V. G. Oklobdzija, “Conditional Techniques for Low-Power Consumption Flip-Flops”, IEEE ICECS’2001, Malta, September 3-5, 2001, pp. 803-806.
[9] C. Piguet, "Supplementary condition for STG-designed speed-independent circuits", Electronics Letters, 2nd April 1998, Vol. 34, No 7, pp. 620-622.
[10] C. Piguet et al. « Memory Element of the Master-slave latch Type, constructed by CMOS Technology”, US Patent 5’748’522, May 5, 1998