# New CMOS VLSI Linear Self-Timed Architectures

A.J. Acosta, M. Bellido, M. Valencia, A. Barriga, R. Jiménez and J.L. Huertas

Dpto. de Diseño de Circuitos Analógicos
Centro Nacional de Microelectrónica/ Universidad de Sevilla
Edificio CICA, Avda. Reina Mercedes s/n, 41012-Sevilla, SPAIN
e-mail: acojim@cnm.us.es

## Abstract

*The implementation of digital signal processor circuits via self-timed techniques is currently a valid alternative to solve some problems encountered in synchronous VLSI circuits. However, a main difference between synchronous and asynchronous circuits is the hardware resources needed to implement asynchronous circuits. This communication presents four less-costly alternatives to a previously reported linear self-timed architecture, and their application in the design of FIFO memories. Furthermore, the integration and characterization in the laboratory of prototypes of these FIFOs are presented.*

## 1: Introduction

The timing problem is currently becoming more important in the design of CMOS VLSI digital systems, due to increased operation speed and complexity. The inherent problems in synchronous circuits, such as clock skew, synchronization failures, and speed limitation due to the slowest unit, and considering current technologies and the case of high-speed or very large integrated circuits, make the asynchronous approach a valid alternative to the synchronous one [10]. Furthermore, asynchronous implementations are recently proving very suitable for low-power DSP applications [12]. With these encouraging prospects, the design of digital systems without a global clock become very interesting. Among several asynchronous techniques, the self-timed one is very promising [8][5][10]. This strategy does not need the distribution of a global clock and presents the potential advantages of higher average speed -since each module operates at the maximum possible speed and hence, the fastest ones need not wait for the slowest ones- and very low power consumption -since each module operates only when there is data to process.

The work presented in this communication is based on the self-timed technique presented in [9], especially suited for CMOS VLSI design [5]. The purpose of this communication is dual. On one hand, a specific methodology is used to present the design of pipeline interface circuits, modifying the data-storage scheme and the handshaking protocol of the scheme presented in [9]. On the other, these proposals and methodology are applied to the design of a linear FIFO memory used in DSP applications.

The communication is organized as follows: in Section 2, the basic linear architecture is discussed; Section 3 presents the proposed modifications for data-storage and handshaking protocol; in Section 4, the new architectures are applied to the design of FIFO memories; Section 5 includes some experimental results obtained from laboratory measurements of integrated prototypes. Finally, the architectures are compared and the most important conclusions are presented.

## 2: Basic Linear Architecture

The basic linear architecture proposed in [9][10] is shown in Fig. 1.a, where the interface and computation block are separated, as well as data buses and protocol signals. Computation blocks (Fig. 1.c) are built with differential circuits [3][4], generating both the output (out) and its complement (outbar), and a complete signal indicating that the logic operation is finished. Two operating phases are distinguished: the precharge phase which generates not-valid-data (PD:11), and the evaluation phase which generates the valid-data (VD:10 or 01). The protocol used is the so-called "full-handshaking", that operates according to the Signal Transition Graph shown in Fig. 1.d[1] [9]. One implementation of this interface circuit is shown in

1. For every STG in this communication, notation + y - after a signal name refers to the rise and the fall of that signal, respectively. The "o" indicates the token-marking in the STG.

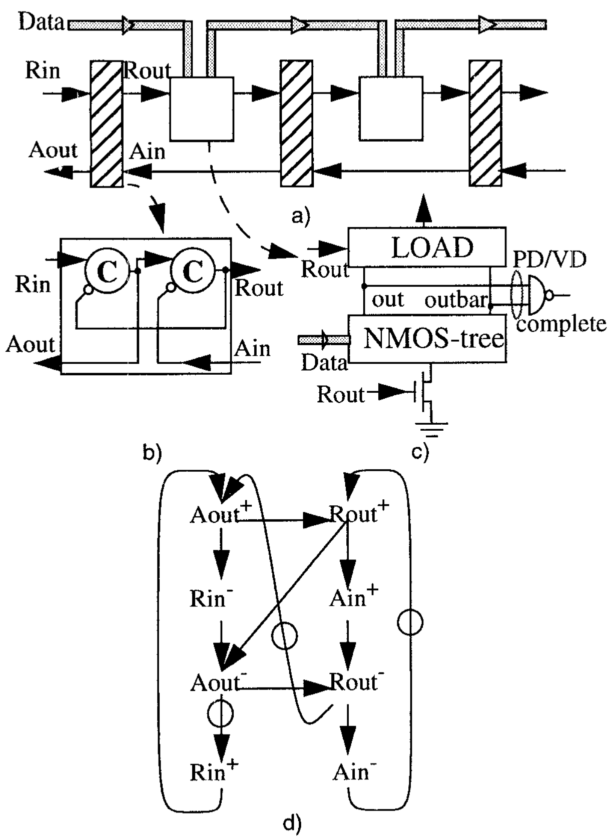


**Figure 1. a) Basic Linear Architecture used in [9][10]. b) Interface Circuit. c) Processing Element. d) STG.**

Fig. 1.b, where the memory elements used are the so-called "C(Muller)-elements". With this handshaking protocol, the request made at this stage is stored in any one of the C-elements forming the interface block, and two consecutive stages can process data simultaneously.

However, there is a situation in protocol signals for this architecture that may provoke an error [9][10]. This may occur when the current stage is still processing - Rout is still high and Ain is still low- and the previous computation block has just entered its precharge phase (Rin becomes low). When the previous block enters its precharge phase, its output data changes from the valid data stored in the previous evaluation phase (VD:10 or 01) to the precharge data (PD:11). This undesired change (10 -> 11 or 01 -> 11) may cause a failure while the current block still needs correct data for its computation. The solution proposed in [9][10] is to place a data-register in the interface circuit, as shown in Fig. 2a. To ensure that the writing operation in the flip-flops finishes and hence, to show speed-independent behavior in global operation [7], a specific mechanism to generate a complete signal -"comp_reg" in Fig. 2a- had to be considered. This complete signal is fed into the second C-element and is also used as an acknowledge signal Aout for the previous stage. This scheme introduces two new signals in the protocol ("load" and "comp-reg"). Fig. 2b shows the STG for the modified protocol.

However, to our knowledge, there is a problem associated with the implementation of the data-register and the generation of complete signal "comp-reg". Particularly, for every mechanism used in such generation, we must ensure the fulfilment of some timing restrictions, concerning the data-storage in the flip-flops composing the data-register. For instance, consider the case in Fig. 2a showing the data-register scheme presented in [10], and that corresponding to the generation of the "comp-reg" signal. This signal is obtained by comparing the input (reg_in[j]) and output (reg_out[j]) values of bit "j" with a two-input AND gate, while the "load" signal is active. In addition, an n+1-input AND gate is needed for the final comparison, being n the number of bits. A first consideration is that

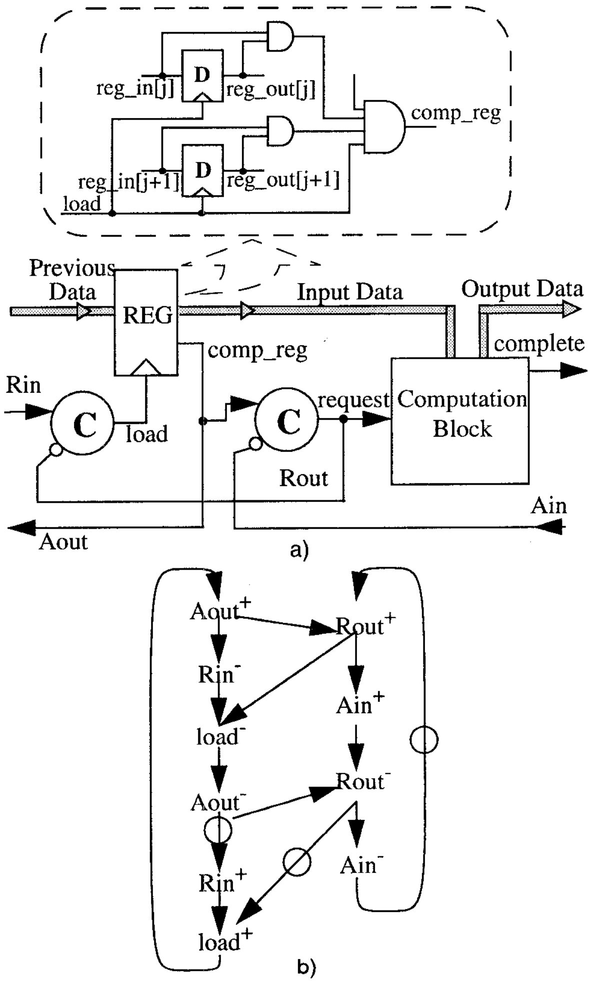


**Figure 2. a) Connection of data register, interface circuit and computation block in [9][10]. b) STG.**

these comparisons should be performed with (N)XOR gates, instead of AND gates. In any case, if one of the memory elements in the data-register enters its metastable state, incorrect data may be stored and provoke an error in the data-path. Furthermore, this error may disable generation of the complete signal, and thus reaching a deadlock situation [2]. To ensure a correct behavior, we must impose a timing restriction referring to possible violations of setup time which may occur when data is being written in the data-register. In the reference case, the implementation must verify the restriction for setup time presented in (1).

$$t_{setup} < t_{gate} + t_C \qquad (1)$$

Eq. (1) is violated if, prior to the time that valid data should be stable, the active edge of "load" signal ($t_{setup}$) occurs, not-valid data is stored, coming from the precharge data of the computation block of the previous stage. In other words, eq. (1) is not fulfilled if $t_{setup}$ is larger than the time elapsed from the generation of valid data in the previous stage to the rising edge of Aout signal. This time corresponds to the generation of complete signal in the previous stage ($t_{gate}$), plus $t_C$ (the propagation delay of the first C-element of the interface circuit, that also generates Aout).

An estimate for integrated blocks in a 1.5 µm CMOS technology, obtains for the right-hand member of (1) a value of 1.6 ns ($t_C$=1.1 ns, $t_{gate}$=0.5 ns). This value for $t_{setup}$ is not very restrictive in integrated flip-flops, but it should be considered in their design. The cost of this structure for each stage of the pipeline is 24n+4 transistors, where n is the width of the data-path. The addition of protocol circuitry (two 8-transistor C-elements) makes total cost of the stage 24n + 12 transistors, not including the part corresponding to the computation block.

With these considerations, the reference structure is not globally speed-independent; thus design techniques must be used to ensure that the restrictions are met. Based on this requirement, a group of simplifications may be established which reduce the hardware in self-timed architectures.

## 3: Proposed Linear Architectures

Our proposals [2] are based on the fact that if the storage time of all the bits in the flip-flops is ensured to be approximately the same within the VLSI environment, the "complete" signal for data storage in the latches is not necessary as an integrated part of the protocol. This implies that it should meet some time restrictions concerning the data storage time in the flip-flops, as in the reference case. This will reflect in considerable savings in hardware resources. Four proposals follow that modify both the data storage scheme and the interface circuitry.

### 3.1: PSCA (Protocol and Storage Controlled by Acknowledge)

The first proposal consists of making the load signal of the flip-flops and the acknowledge signal (Aout) be coincident. Thus, the reference circuit becomes that shown in Fig. 3a. Since the protocol circuit is not modified, the STG describing this circuit's behavior coincides with the original (Fig. 1d). The data storage mode is the following: once the preceding module has performed a request ($Rin^+$), when the actual module is ready to receive the data ($Rout^-$), Aout is activated ($Aout^+$). The effect of this rise is double: first, it orders the previous module to start precharging and second, it allows writing in the flip-flops. Data storage should be done while Rin is high, since Rin falls as consequence of the previous module going into precharge (PD, "11") and data becomes not valid (VD, "10" or"01"). Since data storage occurs once the precharge order is given and thus the valid data disappears, time intervals should be considered which ensure correct operation. This infers that the time for data storage in the flip-flops should be less than the precharge time of the previous module plus the propagation time in the C-element, which is the time elapsed between the rise of Rin and the rise of Aout. This is shown in the time diagram of Fig. 3b. Analytically, this time restriction can be expressed as (2):

$$t_{hold} < t_C + t_{precharge} \qquad (2)$$

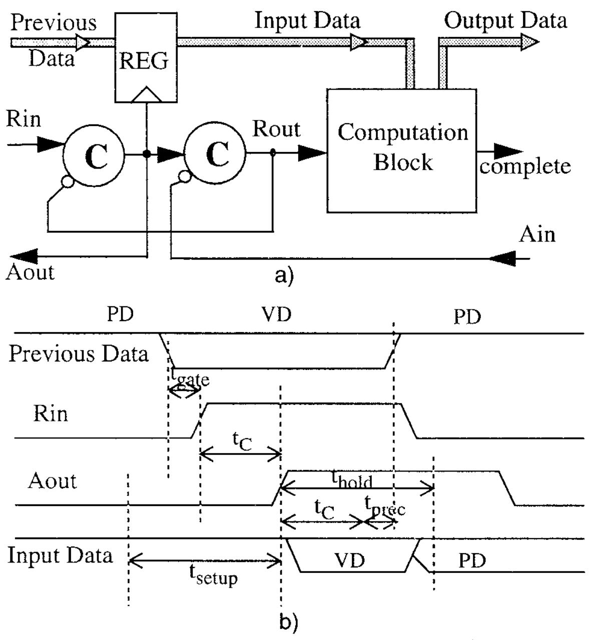


**Figure 3. a) Proposed architecture PSCA.**
**b) Timing Diagram in the case of time violations.**

As in the reference case, the same type of restriction must be imposed on violations in the setup time of the edge-triggered flip-flops. Thus, so that such a violation does not occur, the previous input data in the flip-flops must be stored an stable a $t_{setup}$ time before the active edge of the clock (Aout) arrives. For this, $t_{setup}$ must be less than the signal generation time at the end of the previous stage ($t_{gate}$), plus $t_C$ (the propagation time in the first C-element of the pipeline block which generates Aout). Analytically, this restriction is given in eq. (3), which is the same as that expressed for the structure of [9][10].

$$t_{setup} < t_{gate} + t_C \qquad (3)$$

Regarding time parameters, the most interesting is the throughput, or time needed between consecutive input data for the system to function correctly, which gives an idea of the maximum operation speed of these systems. The maximum throughput is given as the time between two consecutive rises of the "complete" signal in one of the modules. In our case, for a stage (i) of a linear matrix, assuming that the next stage (i+1) is idle (most favorable case), this amount is given by (4):

$$throughput_{max-PSCA} = \frac{1}{4t_C + t_{prech} + t_{eval}} \qquad (4)$$

The advantage of this scheme respecting to that of [9][10] is that PSCA need not generate a specific protocol signal to control the register, rather the signal Aout is sufficient, also used as acknowledge signal in the previous stage. This reflects in increased throughput with respect to the reference scheme. Regarding the required hardware resources, the number of transistors will be 16n+16, where n is the number of bits or data-path width.

## 3.2: PLCAR (Protocol and Latching Controlled by Acknowledge and Request)

This proposal and the following aim to substitute the edge-triggered flip-flops with simpler and smaller latches, consequently requiring less transistors, but preserving the operation speed high. Fig. 4a shows the circuit for this proposal. The logical implementation of the pipeline interface is also composed of two C-elements interconnected in the same manner as in the reference scheme [9][10]. The modification lies in the data-storage system. Since neither STG nor handshaking protocol has been modified from the PSCA scheme, the hardware can be 100% concurrently used. The proposed operation philosophy is summarized in the following.

From a global point of view, the interface circuit can be considered as an element storing the request of one cell to another. In this case, there are two states: the module in question is occupied or idle. If the module is occupied, the

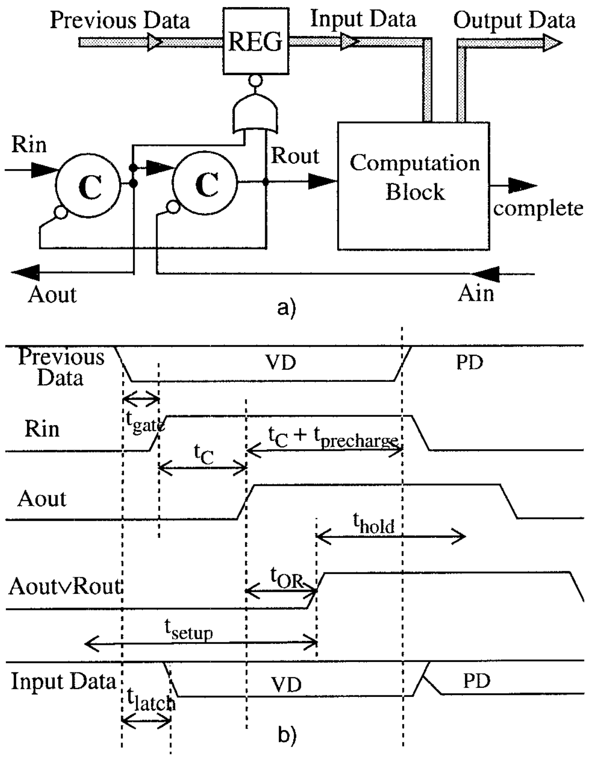


**Figure 4. a) Proposed Architecture PLCAR. b) Timing Diagram in the case of time violations.**

protocol signal Ain is high, allowing the request to be stored in the first C-element (Aout$^+$). If the module is idle, Ain is low, and the request is stored in the second C-element (Rout$^+$), making the module compute. In either case, the fact that Rout or Aout are high indicates a request in the module.

The question of how to store data is easily answered if we consider that while the "complete" signal of the previous stage (Rin for our module) is high, the input data is valid (VD), and when Rin is low, the data corresponding to precharge of the previous module is undesired (PD). Thus, data storage should occur prior to storing the request in the pipeline interface, and writing in the latches should be inhibited during the time the request is stored to avoid changing data while the module computes the correct data.

A simple implementation that allows this operation mode is shown in Fig. 4a, where the OR operation of the Aout and Rout signals acts as the "load" signal in the latch (active in low). Thus, writing is inhibited in the latch when Aout$^+$ $\cup$ Rout$^+$.

Obviously, as in the previous cases, if the "complete" signal is not generated after writing in the latch, proper operation must be ensured by time restrictions, concerning the relations between the data path and the protocol signals. For the circuit to operate well, input data must be stored before writing is inhibited in the latch. This is the same as stating that the storage time in the latch ($t_{hold}$) plus the propagation time of the OR gate, must be less than the propagation time of one (or two) C-elements, accordingly if Aout or Rout stores the request, plus the precharge time of the previous module. In the most restrictive case, we may consider analytically the restriction (5):

$$t_{OR} + t_{hold} < t_C + t_{precharge} \tag{5}$$

This is shown in the time diagram of Fig. 4b. Once the previous block finishes evaluation, its "complete" Rin signal rises, with an equivalent gate delay ($t_{gate}$ in Fig. 4b). The rise of Rin leads to the rise of Aout, and this forces the output of gate OR to rise after a time $t_{OR}$, thus inhibiting storage of the register REG. On the contrary, the rise in Aout also provokes the rise in Rout, if the next module is idle, which is our case. Aout rises after the propagation time of a C-element ($t_C$ in Fig. 4b). Due to protocol, when Aout rises, the previous module enters precharge with the delay time of a C-element plus the precharge time of the previous module and thus, the previous data is no longer valid. Until Rout v Aout (Rout low) rise, previous data is stored in the register and the input data "follows" the previous data, with the only delay of propagation time through the latch ($t_{latch}$ in Fig. 4b). Since in the rise of Aout v Rout, the input data stored in the register contains the valid result of the previous stage, writing in the register is not necessary, and the stored data is valid for the time needed by the block to compute.

The other time restriction to verify is the setup time, or time that the input data must remain stable prior to deactivation of the latch's load signal. In the proposed structure, this restriction is (6):

$$t_{setup} < t_C + t_{gate} + t_{OR} \tag{6}$$

To meet this restriction $t_{setup}$ must be less than the time to generate the complete signal of the previous stage ($t_{gate}$), plus $t_C$ (propagation time of the first C-element of the pipeline block, which generates Aout), plus $t_{OR}$ (propagation time of the OR gate that generates the latches' load signal). Incompliance of this restriction is shown in Fig. 4b.

As to hardware requirements, this structure requires two 2-input C-elements plus an OR gate. This makes a total of 22 transistors. To this, we must add 6 transistors corresponding to the latch. Since one latch is needed to store each bit of information, the cost will be 6n for the case of n bits. The total cost will be 6n+22 transistors for the transmission of n bits.

Referring to time performance, the maximum throughput in the PLCAR case is the quantity appearing in (4), since the path of protocol signals is no different than the previous case. However, we will experimentally verify performance variation caused by a different capacitive load in the output nodes of the C-elements.

The main difference of this scheme in view of that of [9][10] is the use of level-sensitive logic instead of edge-sensitive logic, with its intrinsic advantages regarding hardware requirements and simplicity. In addition, the time behavior, like the PCSA case, is better than that of the reference architecture [9][10].

## 3.3: PLCR (Protocol and Latching Controlled by Request)

The new structure aims to allow writing in the latches according to the value of the protocol signal Rout. This is the same as assuming the request made by the previous module be stored in precisely the second memory element of the pipeline interface.

With this scheme, we intend that the input data remains stable until the next computation block receive the actual data. Thus, Rout values define the data writing and reading phases in the register. When Rout is high, the undesired change in the input signal (01 -> 11 or 10 -> 11) may overwrite actual data. This is avoided if the storage signal is the request signal of Rout, and if the latches are active at a low level. However, this operation mode requires modifying the handshaking protocol.

The STG allowing this operation mode is shown in Fig. 5a. It is basically the original STG, the difference being the transition $Rout^+$ -> $Rin^-$ instead of the original $Rout^+$ -> $Aout^-$. This introduces a time penalty since signal concurrence is decreased. That is, the simultaneous rising and falling of Rout and Rin, respectively, are prohibited, once Aout has risen. In other words, Rin cannot be disabled ($Rin^-$), precharging the previous block, until Rout has risen (the block is processing valid data and consequently the latch should have the correct data stored). However, the new condition imposed ($Rout^+$ -> $Rin^-$) establishes restrictions on an input signal (Rin), which de-

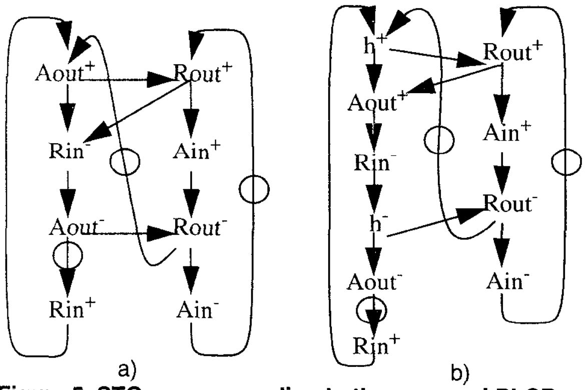


**Figure 5. STGs corresponding to the proposal PLCR.**

pend on the previous, rather than the actual, block. To avoid this, a new intermediate protocol signal (h) is introduced in the STG that does not suppose a substantial modification in the global behavior and eases the synthesis procedure. Thus, the definite STG is shown in Fig. 5b. Since there is no loop in this STG that contains the transitions ($Rout^+$->$Ain^+$ and $h^-$->$Rin^+$) where successive stages are computing , this protocol enables concurrent use of all the hardware, as it happens in the previous cases.

The STG of Fig. 5b served as input for SIS [11], to check that it verifies the properties of correctness (i.e. the STG was live, safe, free-choice and with the complete state coding property) and can thus be synthesized hazard-free. From this STG, SIS generated the following next state equations (7):

$$\begin{aligned} Rout\_next &= Ain'\,h\_ + Ain'\,Rout\_ + h\_\,Rout\_ \\ h\_next &= h\_\,Rin + Rin\,Rout\_' \\ Aout &= h\_Rout\_ \end{aligned} \tag{7}$$

Implementation at a logic level of these equations gave us the circuit of Fig. 6a. As in previous cases, the circuit presents two memory elements. The first is an OR-AND gate (oa12) with a feedback loop connecting the output with one of the OR gate input. The second (a two-input C-element) generates the request (Rout), which, upon performing the AND operation with the output of the first (h), generates the acknowledge (Aout), whose complement serves as Ain input for the preceding module.

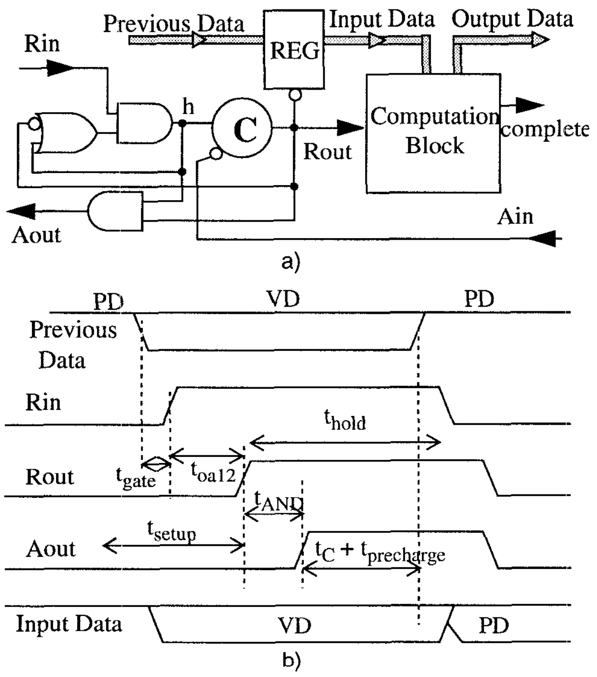


**Figure 6. a) Proposed Architecture PLCR.**
**b) Timing Diagram in the case of time violations.**

As in previous cases, we must apply certain time restrictions to ensure correct operation. Concretely, the data storage time in the latches should be less than the time it takes for Rout to rise, to transmit through the AND gate, causing Aout ($t_{AND}$) to rise, plus the time elapsed since Aout rises until the data shows the precharge value ($t_C$ + $t_{precharge}$, Fig. 6b). This restriction is shown in eq. (8).

$$t_{hold} < t_C + t_{precharge} + t_{AND} \tag{8}$$

Once again, time restrictions must be set for the setup time. This restriction appears in eq. (9):

$$t_{setup} < t_C + t_{oa12} + t_{gate} \tag{9}$$

To meet this restriction $t_{setup}$ must be less than the time to generate the "complete" signal of the previous stage ($t_{gate}$ in Fig. 6b), plus $t_{oa12}$ (the propagation time for the first flip-flop in the pipeline stage) plus $t_C$ (propagation time for the second C-element of the pipeline block, which generates Rout). Incompliance of this restriction is shown in Fig. 6b.

To implement the first flip-flop of the circuit in Fig. 6a, we will take advantage of the abilities of CMOS technology to perform composed boolean functions. Thus, this implementation is realizable with 8 transistors, and the output and its complement are available. Consequently, excluding the part corresponding to the latch, the interface circuit consists of 20 transistors, which added to the 6 needed to implement each latch, makes a total of 6n+20 for a data-path of n bits.

Referring to time performance, the throughput will be less than in previous cases due to the elimination of concurrent transitions in the protocol. This occurs because the acknowledge signal, Aout, will only activate ($Aout^+$) when the auxiliary signal (h) and the request signal (Rout) are active, that is, when $h^+ \cap Rout^+$ is fulfilled. Thus, precharge is delayed in the previous module by, at least, the propagation time of a C-element. In this case, for a cell of a linear array, assuming that the next stage is idle (most favorable case), the throughput will be given by eq. (10):

$$thr_{PLCR} = \frac{1}{4t_C + 2t_{oa12} + 2t_{gate} + t_{prech} + t_{eval}} \tag{10}$$

The biggest advantage of this scheme with respect to the reference scheme lies in the hardware gain, although it is achieved at the cost of reducing the time efficiency.

## 3.4: PLCRAI (Protocol and Latching Controlled by Request via Acknowledge In)

As in the PLCR architecture, this proposal intends to allow writing data in the latches depending on the value of the request signal Rout. However, the operation philosophy is very distinct, since activation of the acknowledge signal (Aout$^+$) that allows precharging the previous module, will only be possible once the acknowledge signal (Ain$^-$) of the following module is disabled. To enable this type of operation, we have slightly modified the handshaking protocol in such a way that after deactivation of Ain (Ain$^-$), Aout$^+$ and Rout$^+$ will rise concurrently. This can be described with the STG of Fig. 7a.

Analyzing this STG, we observe that to activate Rout (Rout$^+$) the acknowledge signal (Aout$^+$) must first be activated, at the same time ordering the previous module (Rin$^-$) to precharge. Consequently, the previous module (Rin$^-$) may precharge an instant before the inhibition of valid data storage in the latches (Rout$^+$). Thus, time restrictions must be imposed on the latches, as in all the previously proposed cases, including the reference.

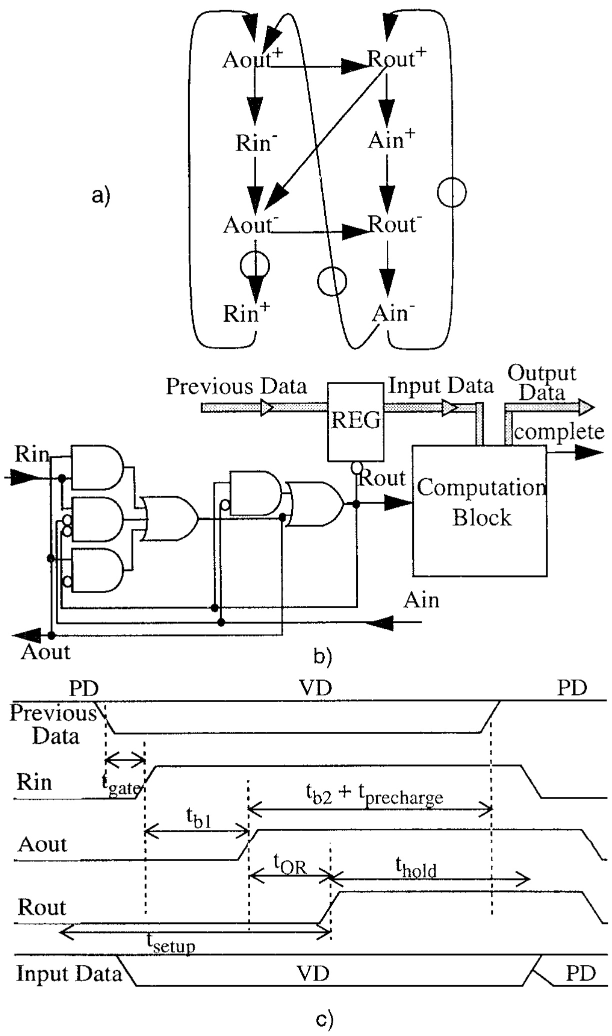


**Figure 7. a) STG for PLCRAI. b) Proposed Architecture. c) Timing Diagram in the case of time violations.**

The STG of Fig. 7a served as input for SIS [11], providing the following next state equations (11):

$$Rout\_next = Ain'\,Rout + Aout$$
$$Aout\_next = Ain'\,Rin\,Rout' + Aout\,Rin + Aout\,Rout' \qquad (11)$$

Once again, in this STG there is no loop that covers the two transitions corresponding to consecutive modules computed (Rout$^+$->Ain$^+$ and Aout$^-$->Rin$^+$), and thus there is total hardware benefit.

Implementation at the gate level of the pipeline interface that verifies (11) is shown in Fig. 7b. As seen, the circuit is composed of two interlaced flip-flops. The most interesting aspect is the fact that the primary input Ain serves as input for both flip-flops, in such a way that deactivation of Ain (Ain$^-$) affects both Rout and Aout output signals. However, activation of Aout (Aout$^+$), provokes activation of Rout (Rout$^-$) after the propagation delay of the OR gate of the second flip-flop (Fig. 7c). Thus, eq. (12) must be met:

$$t_{hold} + t_{or} < t_{precharge} + t_{b2} \qquad (12)$$

The other time restriction to meet is the setup time of the latches. In PLCRAI, this restriction is (13):

$$t_{setup} < t_{b1} + t_{b2} + t_{gate} \qquad (13)$$

To meet this restriction $t_{setup}$ must be less than the time to generate the "complete" signal of the previous stage ($t_{gate}$, in Fig. 7c), plus $t_{b1}$ (propagation time of the first memory element of the pipeline block) plus $t_{b2}$ (propagation time of the second memory element of the pipeline block, which generates the charge signal of the latches). Incompliance of this restriction is shown in Fig. 7c.

When implementing the circuit of Fig. 7b, we take advantage again of the capabilities of CMOS technology to perform composed boolean functions. Thus, implementation of the first flip-flop is realizable with 9 transistors, and the second with 8, having the output and its complement in both cases. Thus, excluding the part corresponding to the latch, the interface circuit consists of 17 transistors, which added to the 6 needed to implement each latch, makes a total of 6n +17 for a structure with n bits.

Regarding time restrictions, the maximum throughput is given by eq. (14):

$$thr_{max-PLCRAI} = \frac{1}{3t_{b1} + 2t_{b2} + t_{prech} + t_{eval}} \qquad (14)$$

As in the PLCR case, we obtain hardware gain, but based on degrading timing perfomance.

As summary of the proposed architecture, a group of parameters to check the distinct proposals is included in Table 1. All the architectures proposed improve the architecture referenced, in terms of hardware resources. The time restrictions to impose in each case have been measured through electrical simulation, and show values that are not very limiting in a VLSI environment. Concerning throughput, both PSCA and PLCAR architectures improve the referenced one.

| | Transistor count | $t_{hold}$ (ns) | $t_{setup}$ (ns) | (Maximum Throughput)$^{-1}$ |
|---|---|---|---|---|
| [9][10] | 24n+12 | - | 1.60 | $4t_C + t_p + t_e + 2t_{gate}$ |
| PSCA | 16n+16 | 1.41 | 1.67 | $4t_C + t_p + t_e$ |
| PLCAR | 6n+22 | 1.24 | 1.84 | $4t_C + t_p + t_e$ |
| PLCR | 6n+20 | 2.31 | 2.49 | $4t_C+2t_{oa12}+2t_{gate}+t_p+t_e$ |
| PLCRAI | 6n+17 | 0.30 | 2.43 | $3t_{b1} + 2t_{b2} + t_p + t_e$ |

**Table 1. Characteristics of the architectures; n: bit number; $t_p$: time of precharge; $t_e$: time of evaluation.**

## 4: Application to designing a FIFO memory

In this section, the pipeline architectures described in Section 3 are applied to the design of linear FIFO memories of 1 bit width. This type of circuit was chosen for its simplicity, its wide use in DSP architectures and its suitability to check protocols and interface circuitry. Implementation of computation elements, in this case realizing the logic function of buffer-inverter, uses the differential structure SODS, presented in [1][2]. This proposal, shown at a transistor level in Fig. 8, has almost half the area, higher operation speed, and less power consumption than other differential structures (for example, DCVS- Domino [4] and ECDL [6]), making it especially suited to improve all aspects of self-timed architectures.

Fig. 9 shows circuits at a transistor level of the proposed architectures. Pseudostatic implementation was used for both the C-elements as well as the D-type edge-triggered flip-flops. Those proposals requiring latches used a bidirectional latch, that is, the input "in" passes to the output "out", and vice versa. It has operated correctly in our case, since the data only goes in the direction in->out, being the inverse path of very high impedance. Another requirement for correct operation is that the in-

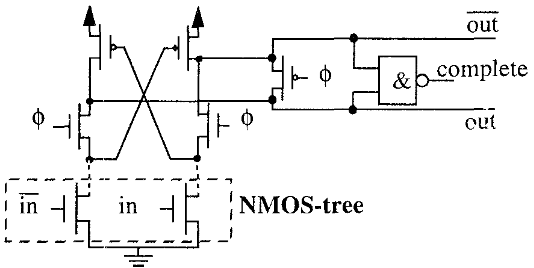


**Figure 8. Implementation of computation blocks with SODS [1][2].**

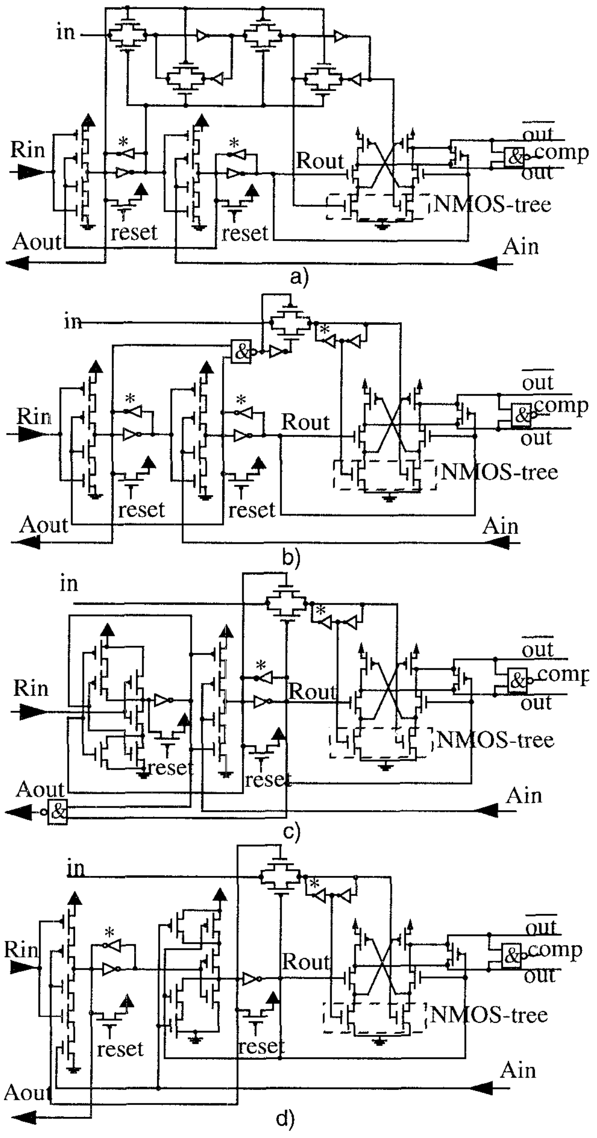


**Figure 9. One-bit FIFO cell: a) PSCA, b) PLCAR, c) PLCR and d) PLCRAI. Inverters marked * are "weak".**

verter closing the loop must be "weak". This is due to the fact that since the data storage loop never opens, the input value should be stored whenever the switch opens the writing phase, regardless of the value stored in the loop. For the implementation of other memory elements we have taken advantage of the facility that CMOS technology to implement all or-and type functions, including a feedback loop. Reset logic is performed by two P transistors, setting to "0" the memory elements of the pipeline interface, indicating that all the stages are in idle state.

To measure the time performance of each cell making up the circuits, a specific methodology has been conceived, consisting of multiplexing the output and connect-

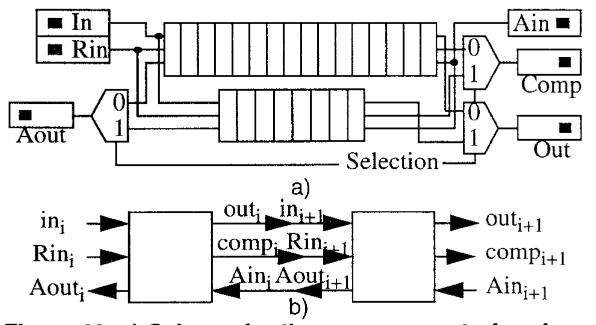


**Figure 10. a) Scheme for the measurement of perform ances of basic cell. b) Interconnection detail.**

ing the input to the 8- and 16-bit FIFOs of each proposal. This scheme, presented in Fig. 10, allows determining the propagation time of a cell (tp) if we measure the quantity (15), where tp(16) and tp(8) are the times measured in the terminals of the integrated circuit (including pads and multiplexers) for the 16- and 8-bit FIFOs, respectively. This eliminates the contributions of the pads and output multiplexers, since they contribute in the same way to tp(16) and tp(8).

$$tp = \frac{tp(16) - tp(8)}{8} \tag{15}$$

The total number of transistors per cell, including 9 of the differential logic, isshown in Table 2. Both 8- and 16-bit FIFOs, for each architecture, were integrated in a double metal standard 1.5 μm CMOS technology. The area of each cell, shown in blocks, and of each FIFO are presented in Table 2. Note that the least costly proposals are PLCRAI and PLCR. A microphotograph of the FIFOs is shown in Fig. 11.

| | PSCA | PLCAR | PLCR | PLCRAI |
|---|---|---|---|---|
| Computation block area | 34 x 94 | 34 x 94 | 34 x 94 | 34 x 94 |
| Memory elements area | 42 x 83 | 37 x 36 | 37 x 36 | 37 x 36 |
| Pipeline area | 50 x 82 | 50 x 122 | 50 x 120 | 50 x 84 |
| Cells size | 50 x 280 | 50 x 261 | 50 x 251 | 50 x 230 |
| Global area | 14000 | 13050 | 12550 | 11500 |
| Transistor-count | 45 | 41 | 39 | 36 |
| 8-bit FIFO area | 505 x 328 | 505 x 314 | 505 x 308 | 505 x 300 |
| 16-bit FIFO area | 1010x328 | 1010x314 | 1010x308 | 1010x300 |

**Table 2. Characteristics of 1-bit FIFO cell including the proposed architectures. Dimensions are in μm and area in μm$^2$.**

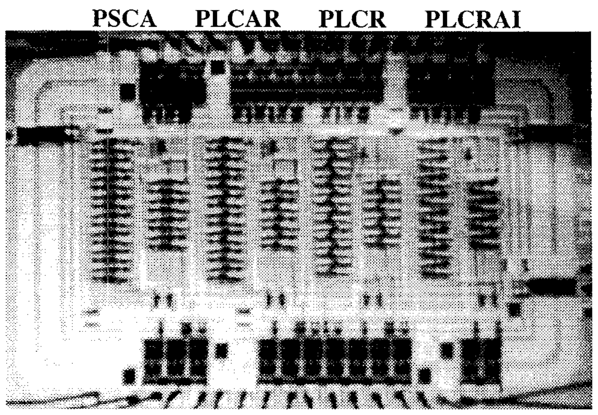


**Figure 11. Microphotograph of the chip.**

## 5: Experimental Results

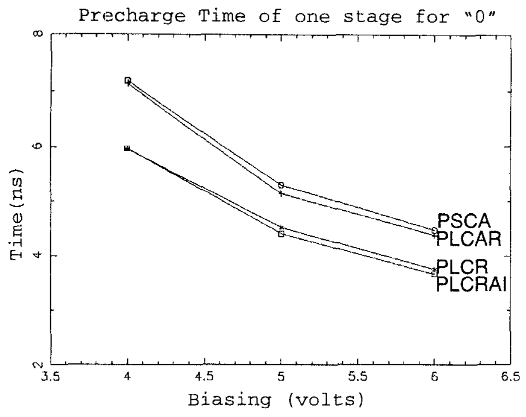
The designs realized were completely characterized through HSPICE simulations and laboratory measurements of integrated prototypes, showing a good behavior in a wide range of operating conditions.

As a result of the timing characterization, Table 3 shows a group of time parameters measured from experimental data. The most significant data is the throughput and the latency of an 8-bit FIFO. This table also shows the data calculated for each cell from eq. (15): precharge and evaluation times for each data (0 and 1). As to power consumption, measurements made in the frequency range 0 to 50 MHz and Vdd = 5 volts, obtained the data included in the last row of Table 3. Power values are around 3 mW per cell, at a frequency of 50 MHz.

| Parameter | PSCA | PLCAR | PLCR | PLCRAI | |
|---|---|---|---|---|---|
| throughput (Mbits/s) | 162.6-79.4 | 152.7-75.7 | 96.1-56.5 | 96.1-74.1 | FIFO |
| latency 8 bits (ns) | 42.5-49.6 | 41.0-50.6 | 38.1-46.9 | 37.4-46.5 | |
| $t_{eval.}$-1 (ns) | 5.12 | 4.88 | 4.70 | 4.57 | 1 CELL |
| $t_{eval.}$-0 (ns) | 4.89 | 5.37 | 5.22 | 5.08 | |
| $t_{prech.}$-1 (ns) | 5.29 | 5.13 | 4.51 | 4.40 | |
| $t_{prech.}$-0 (ns) | 5.55 | 5.12 | 4.36 | 4.24 | |
| Power cons. (μW/MHz) | 55.4 | 62.9 | 58.5 | 49.8 | |

**Table 3. Summary of experimental results.**

Lastly, Fig. 12 shows a representation of the variation with the bias voltage of two time parameters in each cell. Concretely it shows the evaluation and precharge time of a cell if a "1" is introduced in each FIFO.



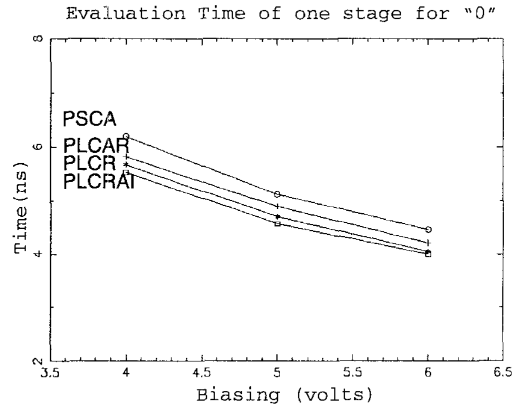


**Figure 12. Experimental timing data for the FIFOs.**

## 6: Structure Comparison and Conclusions

This paper has shown the need to establish certain time restrictions in a reknown reference architecture [9][10]. To this end, four new linear self-timed cells have been presented, all aiming to reduce the hardware of the reference architecture. The first (PSCA) is a small, but important, modification of the cell proposed in [9] and [10]. The second (PLCAR) is the result of using latches to store data instead of edge-triggered flip-flops, and leads to a slight modification of the data storage scheme. The third and fourth proposals (PLCR and PLCRAI) also use latches to store data, storing the request in the second element of the pipeline interface memory. In both cases, STG is modified to increase operation safety. For comparison and application of the structures, two FIFO memories of 8 and 16 bits have been designed and integrated, incorporating these architectures and the high-performance differential structure SODS. A comparative study of the structures presented led to the following conclusions:

Regarding time parameters, the structure with the greatest throughput is PSCA, even moreso than the reference structure. This is followed by PLCAR, PLCRAI, and PLCR. Latency values are very similar, with variations of 10%, being PLCRAI the fastest. Regarding time restrictions, the least limiting are those of PLCR architecture.

Power experimental measurements show similar results for all structures (within 20%). The PLCRAI structure presents the lowest power consumption.

Regarding hardware requirements, a 1-bit cell for the proposed architectures improves the original architecture between 11% and 36%. This improvement increases as we increase the width of the data-path, since the proposed schemes eliminates part of the hardware related to store data in the data-path. Thus, for a 32-bits wide FIFO, the gain goes from 32% (PSCA) up to 75% (PLCRAI).

## 7: References

[1] ACOSTA, A.J., VALENCIA, M., BARRIGA, A., BELLIDO, M.J. and HUERTAS, J.L.: "SODS: A New CMOS Differential-type Structure", Proc. of European Solid-State Circuits Conference, pp. 148-151. Sept. 1994.

[2] ACOSTA, A.J.: "Circuitos Integrados CMOS Autotemporizados", PhD Thesis, University of Seville, February, 1995.

[3] ERDELYI, C.K., GRIFFIN, W.R. and KILMOYER, R.D.: "Cascode Voltage Switch Logic Design", VLSI Design, Vol. 5, No. 10, pp. 78-86. Oct. 1984.

[4] HELLER, L.G., GRIFFIN, W.R., DAVIS, J.W. and THOMA, N.G.: "Cascode Voltage Switch Logic: A Differential CMOS Logic Family". Proc. of the IEEE International Solid-State Circuits Conference, pp. 16-17. 1984.

[5] JACOBS, M.J. and BRODERSEN, R.W.: "A Fully Asynchronous Digital Signal Processor Using Self-Timed Circuits", IEEE Journal of Solid State Circuits, Vol. 25, No. 6, pp. 1526-1537. December 1990.

[6] LU, S.L.: "Implementation of Iterative Networks with CMOS Differential Logic". IEEE Journal of Solid-State Circuits, Vol. 23, No. 4, pp. 1013-1017. August 1988.

[7] MARTIN, A.J,: "Compiling Communicating Processes into Delay-Insensitive VLSI Circuits", Distributed Computing, Vol. 1, No. 4, pp. 226-234, 1986.

[8] MEAD, C.A. and CONWAY, L.: "Introduction to VLSI Systems", Addison-Wesley, 1980.

[9] MENG, T.H.Y., BRODERSEN, R.W. and MESSERSCHMITT, D.G.: "Automatic Synthesis of Asynchronous Circuits from High-Level Specifications", IEEE Transactions on Computer-Aided Design, Vol. 8, No. 11, pp. 1185-1205. Nov. 1989.

[10] MENG, T.H.Y.: "Synchronization Design for Digital Systems", Kluwer Academic Pubs. 1991.

[11] "SIS: A System for Sequential Circuit Synthesis", Mem. No. UCB/ERL M92/41. Univ. of California, Berkeley. 1992.

[12] VAN BERKEL, K., BURGESS, R., KESSELS, J., RONCKEN, M., SCHALIJ, F. and PEETERS, A.: "Asynchronous Circuits for Low Power: A DCC Error Corrector", IEEE Design and Test of Computers, Vol. 11, No. 2, pp. 22-32. Summer 1994.