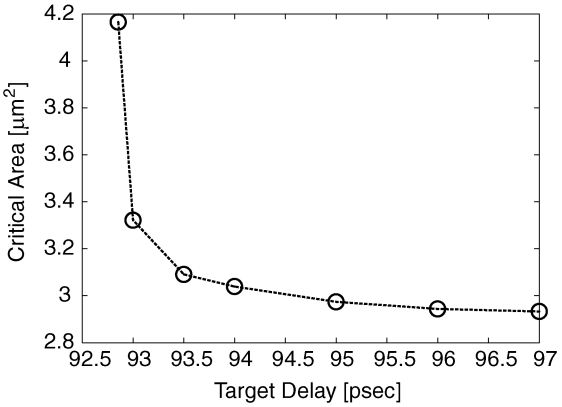

Fig. 8. Tradeoff curve of delay versus CA in the case of the multistage cell OR3_2.

cells are prepared as a yield enhanced library which is essential to realize yield-aware VLSI design flows.

VI. Conclusion

This paper proposed a yield optimization method for standard cells by CA minimization under timing constraints. The proposed method performs a decompaction of the original layout under given timing constraints using LP. We developed a novel linear delay model which approximates the difference from the original cell delay and used this model to formulate the timing constraints as LP. Experimental results showed that the developed delay model is accurate enough to constrain the delay during decompaction. The proposed method can pick up the yield and performance variants of a cell layout from the cell delay versus CA tradeoff curve and can provide a yield enhanced library. The proposed method is the essential technique to realize the yield-aware VLSI design methodologies.

The techniques described in this paper have been proven to be effective for CA minimization. The extension of this timing-aware decompaction framework to the redundant contact insertion has been described in [12]. Not only the functional yield, but also the parametric yield is significantly important in the recent technologies. To take these effects into account, the impact of the decompaction on the OPC results is analyzed in [8] and our future work includes the gate layout pattern regularity enhancement to reduce the systematic variation of the gate critical dimensions.

References

[1] L. Capodieci, P. Gupta, A. B. Kahng, D. Sylvester, and J. Yang, "Toward a methodology for manufacturability-driven design rule exploration," in *Proc. ACM/IEEE 41st Design Autom. Conf.*, 2004, pp. 311–316.

[2] P. Gupta and F.-L. Heng, "Toward a systematic-variation aware timing methodology," in *Proc. ACM/IEEE 41st Design Autom. Conf.*, 2004, pp. 321–326.

[3] J. Mitra, P. Yu, and D. Z. Pan, "RADAR: RET-aware detailed routing using fast lithography simulations," in *Proc. ACM/IEEE 42nd Design Autom. Conf.*, 2005, pp. 369–372.

[4] A. Nardi and A. L. Sangiovanni-Vincentelli, "Synthesis for manufacturability: A sanity check," in *Proc. IEEE/ACM Design, Autom. Test Eur.*, 2004, pp. 796–801.

[5] C. Guardiani, N. Dragone, and P. McNamara, "Proactive design for manufacturability (DFM) for nanometer SoC designs," in *Proc. IEEE Custom Integr. Circuits Conf.*, 2004, pp. 15.1.1–15.1.8.

[6] C. Bamji and E. Malavasi, "Enhanced network flow algorithm for yield optimization," in *Proc. ACM/IEEE 33rd Design Autom. Conf.*, 1996, pp. 746–751.

[7] Y. Bourai and C.-J. R. Shi, "Layout compaction for yield optimization via critical area minimization," in *Proc. IEEE/ACM Design, Autom. Test Eur.*, 2000, pp. 122–125.

[8] T. Iizuka, M. Ikeda, and K. Asada, "OPC-friendly de-compaction with timing constraints for standard cell layouts," in *Proc. IEEE Int. Symp. Quality Electron. Design*, 2006, pp. 776–781.

[9] ILOG, Inc., Mountain View, CA, "ILOG CPLEX 9.1 user's manual," 2005.

[10] Mentor Graphics, Corp., Wilsonville, OR, "Calibre xL user's manual," 2005.

[11] Synopsys, Inc., Mountain View, CA, "HSPICE simulation and analysis user guide," 2005.

[12] T. Iizuka, M. Ikeda, and K. Asada, "Timing-driven redundant contact insertion for standard cell yield enhancement," in *Proc. IEEE Int. Conf. Electron., Circuits Syst.*, 2006, pp. 704–707.

Registers for Phase Difference Based Logic

D. Shang, A. Yakovlev, A. Koelmans, D. Sokolov, and A. Bystrov

***Abstract*—A logic design style known as phase difference-based logic (PDBL) has several benefits with respect to security and testing. An existing design method for PDBL circuits has so far been lacking an important component, a register. In this paper, we present the design of a speed independent PDBL register and a timed PDBL register, which can be used in asynchronous or synchronous circuits. Comparisons are presented in terms of speed, size, and power consumption.**

I. Introduction

Phase difference-based logic (PDBL) was proposed in [7]. It is an extension to the combined use of *dual-rail encoding* and the *return-to-zero protocol*. Dual-rail encoding [2] uses two rails to present a single bit. It has two valid signal combinations, $\{01\}$ and $\{10\}$, which encode the value 0 and 1, respectively, and are called *codewords*. Dual-rail encoding is widely used in self-timed circuits [10], [11], where the return-to-zero switching protocol helps to avoid switching hazards. The protocol allows only transitions from the *all-zeros* state $\{00\}$, which is a *noncodeword*, to a codeword and back again, which has the effect that the switching is monotonic [14]. The all-zeros state is used to indicate the absence of data and is called a *spacer*. Circuits that use a spacer-based protocol can, therefore, detect the presence or absence of data without the use of a clock signal (or a separate request signal), making it attractive to designers of asynchronous circuits. An existing example of spacer-based logic is *null convention logic* (NCL) [9].

PDBL uses *two* spacers (*all-zeros* and *all-ones*) instead of the single all-zeros spacer, and the switching protocol repeats the following sequence: *all zeroes spacer* → *codeword* → *all-ones spacer* → *codeword*. Because the all-zeros and all-ones spacers are used alternatively, this kind of spacer arrangement is called the *alternating spacer protocol* [3]. It is illustrated in Fig. 1.

PDBL has the following advantages [7].

1) Its switching activity is independent of processed data, because all logic gates switch regardless of the actual data values during

Manuscript received July 27, 2005; revised November 17, 2005. This work was supported by the University of Newcastle upon Tyne under the EPSRC STELLA and SCREEN Projects.

The authors are with the School of Electrical, Electronic, and Computer Engineering, University of Newcastle upon Tyne, NE1 TRU, U.K. (e-mail: albert.koelmans@ncl.ac.uk)

Digital Object Identifier 10.1109/TVLSI.2007.898772

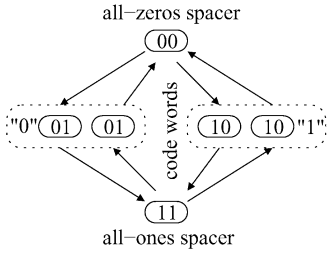

Fig. 1. Alternating spacer protocol.

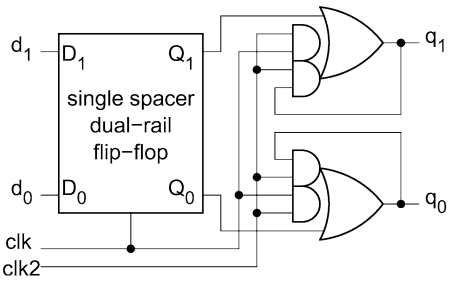

Fig. 2. Dual-rail alternating spacer register.

operation. This increases security [3], because it makes it very difficult to deduce the internal behavior of the circuit from observing its switching activity patterns.

2) Its Idd, measured over one cycle, characterizes the absence or presence of a fault. This is clearly useful during online testing [8].
3) Its nodes have a stable and predictable periodicity when switching. This is good for refreshing (cf. decoherence problem in quantum circuits [13]).

A basic method to implement PDBL circuits was introduced in [7] and improved in [3]. A block diagram of the register is shown in Fig. 2, which combines a dual-rail single spacer register and a converter between a single spacer and an alternating spacer. This register is not a "proper" PDBL circuit because its operation depends on a single spacer register. To implement the alternating spacer protocol, a converter between a single spacer and an alternating spacer is used as shown on the right-hand side of Fig. 2.

The aim of this paper is to present the design of a "real" PDBL register in which all logic gates switch independently of the data value. The remainder of this paper is organized as follows. Section II provides two different designs of a PDBL register. Section III compares it with other designs in terms of power, size, and speed. Section IV presents the conclusions.

II. PDBL Register Design

A. Speed Independent Register

We designed a speed independent (SI) PDBL register according to the block diagram shown in Fig. 3, following ideas from [10] and [11]. It consists of three latches connected in such way that a data value is never overwritten and that two codewords are never stored simultaneously. The register operation (at a high level) is captured above each latch, denoting the events on its output. The order of the events is shown by numbers above the arrows. Initially, *latch_2* is reset to a codeword CWa, and *latch_1* and *latch_3* are reset to the all-zeros spacer. When the reset is released, the codeword *CWa* is copied from *latch_2* to *latch_3*. Then the spacer is copied from *latch_1* to *latch_2*. Then the codeword *CWb* from the combinational logic output is stored in *latch_1*, which allows the all-ones spacer onto the output of *latch_3*, and so on. A pair of inverters with their outputs crossed between *latch_1* and *latch_2* provides the alternation of the spacer, but this does not affect the codewords. The inverters between the acknowledgements of *latch_2* and requests of *latch_1* serve the same purpose: spacer alternation.

An individual dual-rail latch is shown in Fig. 4. It was designed using the Petrify tool[1] with subsequent manual optimization. The latch has two request inputs: *reqCW* and *reqSP*. A transition on *reqCW* means that the codeword on the input $\langle d\_1, d\_0 \rangle$ is ready to be latched. The latching is acknowledged by a transition on the *ackCW* output (with the same polarity as *reqCW*). A positive transition of the *reqSP* input requests the storage of the all-ones spacer, and its negative transition denotes the storage of the all-zeros spacer. The latching of a spacer is acknowledged by a transition on *ackSP* output (with the same polarity as *reqSP*).

B. Timed Register

The SI solution requires a large number of gates. Recently, relative timing [5] has been introduced in self-timed circuit designs to reduce area and improve performance. To implement this kind of solution, the crucial issue is the implementation of the alternating spacers. The spacers are naturally propagated from one latch to the next. Our preferred solution is to force a register into a spacer state before any data is written in, rather than allowing the spacers to propagate. A similar idea is also used in [1] for secure latch design. As this register is a master-slave latch, this mechanism works as follows. When writing new data, the master is forced into a spacer first, and then the data is latched into the master. After that the slave is forced into a (alternating) spacer state, which is followed by the withdrawal of the *req* signal. Finally, the data latched at the master is propagated to the slave.

The circuit implementation, shown in Fig. 5, consists of a master-slave latch, a toggle, and a controller. The master-slave latch is the main part, and is used to propagate and store data. The toggle generates either all-zeros or all-ones spacers for the master and slave latches. If the value of the toggle is 1, the spacer at the master is the all-zeros spacer, otherwise the all-ones spacer. For the slave it is the other way round, in order to balance the power consumption. The controller is used to generate the spacers for the master and slave, and to propagate and store the data inside the register.

In Fig. 5, 0 and 1, 2, and 3 stand for logic low, logic high, an inverter, and an "AND-OR" logic gate (i.e., a complex gate consisting of one AND gate and one OR gate), respectively. Initially, the latch stores a codeword; because of this, the *mFFen* signal is cleared through the XNOR gate ($mFFen = 0$, the inputs of the XNOR gate are 01 or 10) which blocks new inputs; when new data is available, the *req* signal is asserted ($req+$). The signal is propagated through gate 2 (inverter) and the following "NOR" gate, and then the *enMsp* signal is generated ($enMsp = 1$) because the output of gate 3 is low and the $req+$ makes the output of gate 2 low. In this case, if the toggle is 1, $\{a, b\}$ equals $\{1, 1\}$, which is generated through two "AND-NOR" gates, and then the master is forced into the all-zeros spacer; otherwise, the all-ones spacer is generated. After the master is forced into the spacer state, the *mFFen* is set ($mFFen = 1$) because both inputs of the XNOR gate are $\{00\}$ or $\{11\}$, and that forces the output of gate 3 to remain high. New data cannot be latched until *enMsp* is withdrawn, and that results in $\{a, b\} = \{0, 1\}$. After the new data is latched into the master, the *mFFen* is cleared again, and the *enSsp* signal is generated ($enSsp = 1$) through a "NOR" gate. The slave is subsequently forced into a spacer using the same mechanism as used at the master. The choice of all-zeros or all-ones spacer is decided by the value of the toggle: if the toggle is

[1][Online]. Available: http://www.lsi.upc.es/~jordic/petrify

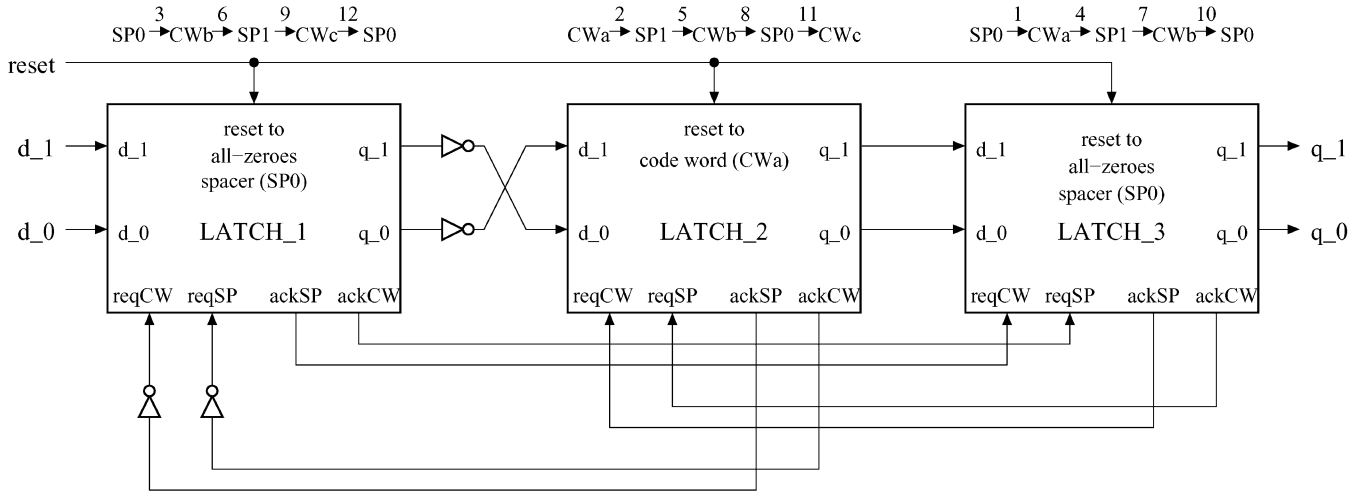

Fig. 3. SI PDBL register.

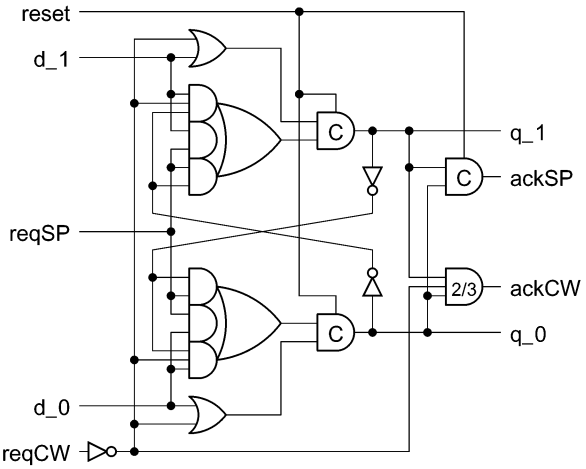

Fig. 4. One stage of SI PDBL register.

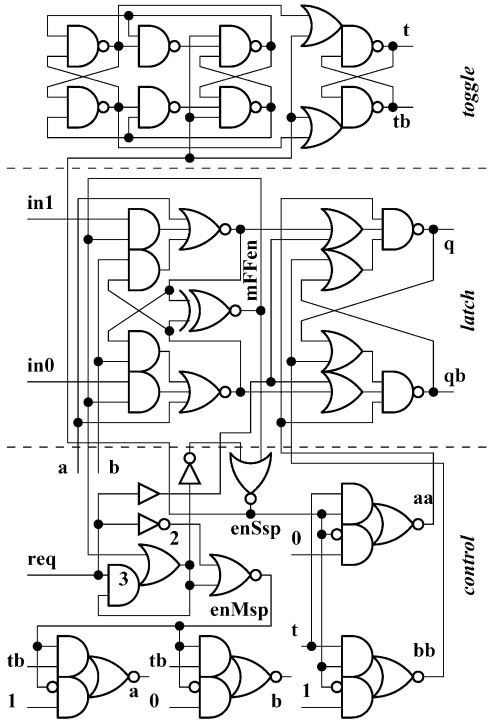

Fig. 5. Circuit diagram of the PDBL register.

$1, \{aa, bb\}$ is set to $\{0, 0\}$, and the all-ones spacer is generated; otherwise, the all-zeros spacer is generated. After the slave is forced into a spacer state, the spacer is fed back to the inputs, the input data is returned to a spacer and *req* is withdrawn. After *req*-, the data is propagated and stored in the slave. This happens only after $\{aa, bb\}$ is changed to $\{1, 0\}$ when *enSsp*- occurs.

Normally, the master is in a stable state, during which the control variables a, b, and *mFFen* have the values 0, 1, and 0, respectively. When data is being written, if $\mathrm{toggle} = 1$, the control variables are set to 1XX, and the master is in the all-zeros spacer state; if $\mathrm{toggle} = 0$, the three control variables are 0, 0, and 0, and the master is in the all-ones spacer state. After that, the master goes to the "enable inputs" state due to the control variables changing to 0, 1, and 1. Finally, the master returns to the stable state where it is waiting for new data. A number of timing assumptions are used.

Timing Assumption 1: In order to keep the enMsp signal low when req-, gate 2 (inverter) should be faster than gate 3 ("AND-OR" gate).

1) $\Delta\,\mathrm{inverter} \prec \Delta 3\mathrm{input}$ "AND–OR" (AO21)

This assumption is reasonable because the inverter is smaller and faster than the "AND-OR" gate.

The toggle (shown at the top of Fig. 5), is used only for generating alternating spacers, so a dual-rail T-latch is sufficient here. The data propagation in the main part register and the toggle update proceed sequentially, which may reduce the performance of the circuit. An alternative is to design the toggle as a master-slave latch, in which a dual-rail T-latch is the master. The spacer propagation at the slave end of the main part register and the update of the master of the toggle now execute concurrently, and the update of the slave of the toggle now executes concurrently with the data propagation at the slave end of the main part register. Because the toggle has no completion detection logic (to save hardware), we must guarantee that the toggle settles down before the input data has settled down at the register, so we must make sure that the slave of the toggle is fast compared to the slave of the main part register. This is implemented as follows: after $enSsp+$, the master of the toggle is updated. The update is executed concurrently with the spacer propagation through the slave of the main part register. After $enSsp-$, the data held by the master of the toggle is propagated to the slave of the toggle. This happens concurrently with the data propagation from the master to the slave of the register.

Two more timing assumptions are used here.

Timing Assumption 2: Before $enSsp-$, the master of the toggle should have been updated.

TABLE I
COMPARISON OF RESULTS

	SYNC Env.	ASYNC Env.	Shared Controller	req(clk) to req(clk)	Size
Clocked REG	yes	no	no	2.0n	1167
Timed REG	yes	yes	yes	7.5n	1948 (1.67 times)
SI REG	yes	yes	no	18.0n	4740 (3.2 times)

TABLE II
MEASUREMENT RESULTS (UNIT: PICOJOULES/3.3)

	STD LAT	Single LAT	Plana's REG	Clocked REG	Timed REG
write 0	0.30	1.07	1.13	2.35	3.58
write 1	0.48	1.09	1.16	2.34	3.56
difference	0.18	0.02	0.03	0.01	0.02

2) $\Delta\{3*2\,\text{input "NAND"}\,(NA2)+2*3\,\text{input "NAND"}\,(NA3)\} \prec \Delta\{4\,\text{input "AND–NOR"}\,(AN22)+5\,\text{input "OR–NAND"}\,(ON221)+3\,\text{input "AND–OR"}\,(AO21)+2\,\text{input "NOR"}\,(NO2)\}$.

Timing assumption 3: After $enSsp-$, the slave of the toggle is faster than the slave of the main part latch.

3) $\Delta\{2 * 3\,\text{input "OR–NAND"}\,(ON21)\} \prec \Delta\{2 * 5\,\text{input "OR–NAND"}\,(ON221)\}$.

These two timing assumptions are used to improve the performance of the register. Timing assumption 3 is used to guarantee that before the data is ready, the toggle should be ready for the next cycle. Compared to the slave of the main part latch, the design of the slave of the toggle is simpler, so it is reasonable to assume that it is faster. Timing assumption 2 is used to guarantee that before the input data is propagated to the slave, the master of the toggle has already been updated because of timing assumption 3. Again, analysis of the circuit shows that this is a reasonable assumption.

The SI PDBL register and timed PDBL register were implemented in the AMS 0.35-μm CMOS technology using the Cadence toolkit. The analogue simulation results show that the registers work as expected. The waveforms of the simulation for the timed PDBL register.[2]

III. COMPARISON OF RESULTS

The performance of the register shown in Fig. 5 (called "Timed REG") was measured in terms of speed and size. The SI register shown in Fig. 3 ("SI REG") and the register shown in Fig. 2 ("Clocked REG") were also measured. A 1-bit clocked register, 1-bit timed register and 1-bit SI register were all implemented in CMOS technology (AMS 0.35-μm CMOS TECH_CSI). The comparison of the results is shown in Table I.

According to these results, Clocked REG is the fastest and smallest. Although it is robust in terms of delay variation, SI REG is the slowest and biggest. Timed REG is clearly a compromise with respect to speed and size. Clocked REG can only be used in synchronous systems, while Timed REG and SI REG can be used in both synchronous and asynchronous systems. In synchronous systems, the *req* signal should be replaced by the global clock signal, and the registers then work in Fundamental Mode (i.e., worst case performance). Timed REG has a potential advantage: by building a centralized controller it is possible to save area compared to both Clocked REG and SI REG. For example, if a group of parallel registers is used, a shared controller and toggle built only based on the slowest bit must be constructed. Clocked REG is the fastest, but does not include completion detection logic. This means that the data should be settled in each clock cycle, otherwise the system may fail. In contrast to Clocked REG, Timed REG and SI REG both contain completion detection logic.

[2][Online]. Available: http://www.staff.ncl.ac.uk/i.g.clark/async/tech-reports/NCL-EECE-MSD-TR-2005-107.pdf

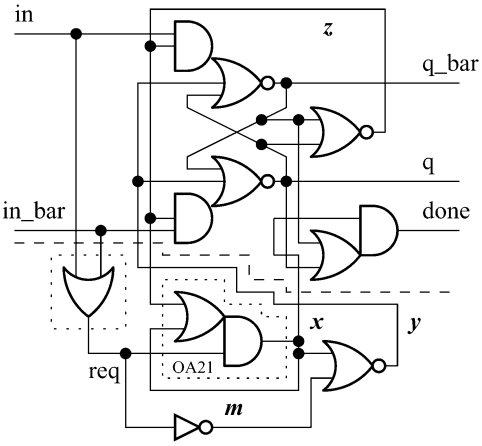

Fig. 6. Schematic of the return-to-zeros spacer latch.

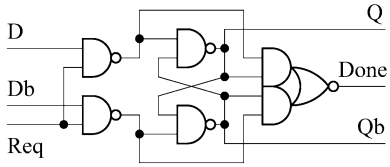

Fig. 7. Schematic of the standard dual-rail latch.

The clocked register has the property that its switching activity is independent from the processed data, which may help reduce leakage of information through measurement of the energy consumption. We, therefore, measured the difference in power consumption when the registers process different data streams. We would like to emphasize that we are not claiming that we can totally prevent DPA—how the dual rail architecture can help resisting attacks requires more investigation, for example by studying forms of attacks which integrate power in time. This paper does not cover such issues.

We compared the 1-bit timed register, the dual-rail with single spacer latch (Single LAT) shown in Fig. 6 and published in [4], the clocked register (Clocked REG) shown in Fig. 2 and the secure latch designed by Plana *et al.* [1]. To illustrate the difference when processing different data (logic zero and logic one), the standard dual-rail latch (STD LAT) shown in Fig. 7 and published in [12] is used as a reference in the experiment. The results are shown in Table II.

We find that when writing different data, the standard latch (STD LAT) consumes a significantly different amount of energy; the others show quite a small difference in terms of power consumption. This is because they use return-to-spacers in their implementations. Standard

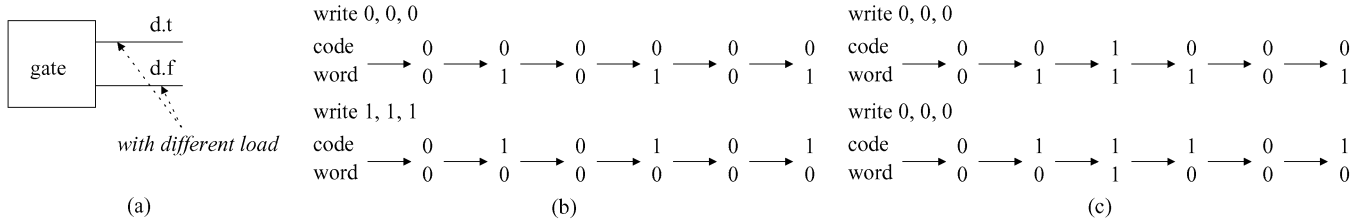

Fig. 8. Comparison between single spacer and alternating spacer.

latches (as well as Plana *et al.*'s latch) do not have the property of firing all nodes per cycle, regardless of the data processed, so the small differences on every cycle will accumulate over time for a sequence of bits processed over many cycles.

We also find that all latches (registers) with return-to-spacers show similar small differences in terms of power consumption when different data is written, regardless of whether the single spacer protocol or the alternating spacer protocol is used. Here, we assume that the load of the latch (register) is balanced. We discuss the unbalanced load case in the following.

IV. Conclusion

The design of an SI PDBL register and a timed PDBL register were presented. The analogue simulation results show that the registers work as expected. Comparisons are presented in terms of speed, size, power consumption, and power balancing. These show that PDBL registers may contribute to resistance to attacks that are based on energy consumption. This is because in PDBL circuits, switching activity is totally independent of processed data due to the following reasons.

1) Dual-rail encoding has a symmetry between logic high and logic low actions that reveals little through its power consumption and electromagnetic emissions [6], [1].
2) In addition, if there is still some remaining unbalance, the alternating spacer protocol helps to avoid the exploitation of the accumulation of the imbalance over time caused by uneven loads and parasitic parameters in dual-rail circuits.

The first point is explained in [6]. For the second point, a typical example is shown in Fig. 8. Fig. 8 shows a dual-rail register. With the data sequence 000, in the return-to-zero spacer protocol the corresponding signal transition trace will be $\mathrm{d.f} + (\mathrm{data}\,0), \mathrm{d.f} - (\mathrm{spacer}), \mathrm{d.f} + (\mathrm{data}\,0), \mathrm{d.f} - (\mathrm{spacer}), \mathrm{d.f} + (\mathrm{data}\,0), \mathrm{d.f} - (\mathrm{spacer})$. In this case, the $\mathrm{d.t}$ wire does not switch at all. With the data sequence 111, the corresponding signal transition trace switches at the $\mathrm{d.t}$ wire only, and the $\mathrm{d.f}$ wire does not switch at all. If the two rails have different loads due to, for example, fabrication faults, the resulting power consumption will vary. However, in PDBL circuits, the data sequence 000 produces the signal transition trace $\mathrm{d.f} + (\mathrm{data}\,0), \mathrm{d.t} + (\mathrm{spacer}), \mathrm{d.t} - (\mathrm{data}\,0), \mathrm{d.f} - (\mathrm{spacer}), \mathrm{d.f} + (\mathrm{data}\,0), \mathrm{d.f} - (\mathrm{spacer})$. In this case, both the $\mathrm{d.t}$ and $\mathrm{d.f}$ wires switch. The same happens when the data sequence 111 occurs. Thus, even with unbalanced loads, the circuit will consume the same amount of power (but at different moments in time). Because of the fact that in PDBL circuits the two rails of each signal are always switched, this should also help to test the circuit and to perform data refreshment in dynamic storage systems. We do not consider current leakage in this paper. Other possible areas of using the switching invariance property include testing based on monitoring switching activity, and systems that require highly periodic refreshing.

Acknowledgment

The authors would like to thank Dr. F. Burns and Dr. F. Xia for useful discussions. They would also like to thank the anonymous referees for helpful comments on this paper's manuscript.

References

[1] L. A. Plana, P. A. Riocreux, W. J. Bainbridge, A. Bardsley, S. Temple, J. D. Garside, and Z. C. Yu, "SPA—A secure amulet core for smartcard applications," *Microprocess. Microsyst.*, vol. 27, pp. 431–446, 2003.
[2] J. Sparso and S. Furber, *Principles of Asynchronous Circuit Design: A System Perspective*. Norwell, MA: Kluwer, 2001.
[3] D. Sokolov, J. Murphy, A. Bystrov, and A. Yakovlev, "Design and analysis of dual-rail circuits for security applications," *IEEE Trans. Comput.*, vol. 54, no. 4, pp. 449–460, Apr. 2005.
[4] D. Shang, F. Burns, A. Bystrov, A. Koelmans, D. Sokolov, and A. Yakovlev, "A low and balanced power implementation of the AES security mechanism using self-timed circuits," in *Proc. PATMOS, LNCS 3254*, 2004, pp. 471–480.
[5] K. Stevens, R. Ginosar, and S. Rotem, "Relative Timing," in *Proc. ASYNC*, 1999, pp. 208–218.
[6] Z. Yu, S. Furber, and L. Plana, "An investigation into the security of self-timed circuits," in *Proc. ASYNC*, 2003, pp. 206–215.
[7] A. Yakovlev, A. Bystrov, D. Sokolov, V. Varshavsky, and V. Marakhovsky, "Phase-difference based logic: Principle and applications," in *Proc. ACiD-WG*, 2004, pp. 160–172.
[8] C. D. Alessandro, D. Shang, A. Bystrov, and A. Yakovlev, "PSK Signalling on NoC Buses," in *PATMOS, LNCS 3728*, 2005, pp. 286–296.
[9] A. Kondratyev and K. Lwin, "Design of asynchronous circuits using synchronous CAD tools," *IEEE Comput.*, vol. 19, no. 4, pp. 107–117, Jul./Aug. 2002.
[10] V. Varshavsky, M. Kishinevsky, A. Kondratyev, and A. Taubin, *Self-Timed Control of Concurrent Process*. Norwell, MA: Kluwer, 1990.
[11] I. David, R. Ginosar, and M. Yoeli, "An efficient implementation of boolean functions as self-timed circuits," *IEEE Trans. Comput.*, vol. 41, no. 1, pp. 2–11, Jan. 1992.
[12] A. Bystrov, D. Shang, F. Xia, and A. Yakovlev, "Self-timed and speed independent latch circuits," in *Proc. 6th U.K. Asynchronous Forum, Univ. Manchester*, 1999, pp. 1–11.
[13] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge, U.K.: Cambridge Univ. Press, 2000.
[14] T. Thorp, G. Yee, and C. Sechen, "Design of monotonic circuits," in *Proc. ICCD*, 1999, pp. 569–572.