From cube 1 in Fig. 1*b*, *a*'{1,11,14,15}* *b*{7,12,14,15}**d*{7,12, 14,15}, we can obtain test vectors that sensitise a path {*d*,7,12,14,15}. Since a set of test vectors {(*a*=0, *b*=1, *c*=1, *d*=0, *e*=0, *f*=0, *g*=0), (*a*=0, *b*=1, *c*=1, *d*=1, *e*=1, *f*=0, *g*=0)} only sensitises the path {*d*,7,12,14,15}, they can detect the path delay faults along it.

For cube 3, *b*{7,12,14,15)**c*'{2,6,11,14,15)**d*{6,11,14,15}**d*{7, 12,14,15}, we can obtain test vectors that sensitise a path {*d*,6,11,14,15} or a path {*d*,7,12,14,15}. However, these test vectors cannot detect the path delay faults along the path {*d*,6,11,14,15} or the path {*d*,7,12,14,15} because they sensitise more than one path. The path {*d*,6,11,14,15} and the path {*d*,7,12,14,15) are untestable paths.

For cube 4, *b*{8,12,14,15}**c*'{2,6,11,14,15}**d*{6,11,14,15}**e*{6, 11,14,15}, we cannot achieve test vectors that detect path delay faults along the path {*d*,6,11,14,15}, because any test vectors that sensitise the path {*d*,6,11,14,15} also sensitise another path {*d*,7,12,14,15} of cube 3. The paths {*d*,6,11,14,15} and {*d*,7,12,14,15} are untestable.

ENF analysis for reconvergent paths: From the analysis of the untestable paths in the multi-level circuits, it is observed that reconvergent paths are sensitised simultaneously.

In the untestable paths with different parity, the output of a reconvergent gate has a constant value: if it is an AND gate output is always 0; if it is an OR gate output is always 1. The redundancy exists in the output of such a reconvergent gate, and the redundancy can be detected easily without ENF analysis [1]. However, in the untestable paths with the same parity, no redundancy exists in the output of their reconvergent gates. These paths can be detected only with ENF analysis.

Table 1: Comparison in run-time and memory usage

Benchmark circuits	ins/outs	lits	det. paths /untest. paths	'PO-ENF' [3]		'Re-ENF' (proposed)	
				CPU time, s	Mem. usage, KB	CPU time, s	Mem. usage, KB
5xpl	7/10	293	704/782	14.9	44.4	9.5	23.6
9sym	9/1	516	1,047/1,047	25.2	127.7	13.1	42.6
conl	7/2	23	2/2	2.2	1.4	0.5	0.5
duke2	22/29	1,775	2,592/2,592	48.6	990.0	32.2	289.2
rd73	7/3	843	848/854	12.3	29.7	8.2	23.1
rd53	5/3	140	45/45	5.3	3.4	4.3	2.0
C432	36/7	372	1,778/1,778	42.1	344.7	31.3	266.6
C499	41/32	616	2,475/2,520	53.3	1,234.3	22.1	176.7
C880	60/26	729	5,636/5,746	62.4	1,679.6	44.7	919.5
C1355	41/43	1,064	12,292/12,647	149.8	70,492.0	78.2	33,608.2
C1906	33/25	1,497	26,323/26,541	250.9	140,242.4	100.8	87,951.8
C2670	233/140	2,075	149,181/150,082	1,843.5	457,562.1	1,243.6	304,863.4
C3540	50/22	2,936	142,082/147,235	962.4	219,640.8	649.5	107,049.0
C3515	178/123	4,386	304,470/312,598	2,256.1	512,021.0	1,048.6	292,925.6
				Total			
				5,729.0	1,404,413.5	3,286.6	828,141.8
				Δ (%)			
				—	—	−42.6	−41.0

ins/outs: number of primary inputs/outputs
lits : number of literals in sum-of-product form
det. paths/untest. paths: detected untestable paths/untestable paths
'PO-ENF': untestable path detection by ENFs for primary outputs
'Re-ENF': untestable paths detection by ENFs for reconvergent paths

As described above, in order to detect untestable paths efficiently in a circuit, it is necessary to construct the ENF for the reconvergent paths with the same parity and to remove the redundancy. If more than one reconvergent path with the same parity exists, we construct a logic cone for each reconvergent path. In this process, ENFs are constructed for the logic cones that are not contained in the others. This containment relationship among logic cones can be identified through the fan-in/fan-out relationship of reconvergent gates for the logic cones. Via the containment relationship, we can minimise the number of the duplicated cubes produced by constructing other ENFs.

The ENF expressions for a primary output f1 and for a reconvergent gate 14 are shown in Fig. 1*b* and *c*, respectively. While the ENF expression for the primary output consists of six SOP terms; the ENF expression for reconvergent path consists of only four SOP terms. The proposed algorithm is described as follows:

Step 1: Identify the reconvergent paths with the same parity.
Step 2: If more than one reconvergent path with the same parity exists, construct a logic cone for each reconvergent path. Otherwise, go to Step 4.
Step 3: If the logic cone is contained in other logic cones, neglect it.
Step 4: Construct the ENF for each logic cone.
Step 5: Find and remove untestable paths by analysing the ENF expression for each path.

Experimental results: In Table 1, we show the experimental results on MCNC/ISCAS benchmark circuits. The results of the proposed algorithm (tagged 'Re-ENF') are compared to the previous algorithms that construct the ENF for all the primary outputs (tagged 'PO-ENF') in terms of the CPU time and memory usage. Each circuit is optimised by a standard script (~cad/lib/misII/lib/script) for the experiments. By constructing ENFs for reconvergent paths, CPU time and memory usage are decreased by 42.6 and 41.0%, respectively.

Conclusions: We have described an efficient algorithm for detecting untestable paths in multi-level circuits. By constructing ENF only for reconvergent paths, the proposed algorithm detects and removes untestable paths efficiently.

Acknowledgments: This work was supported by Ministry of Education through Inter-university Semiconductor Research Center (ISRC 94-E-2030) in Seoul National University.

12 February 1996
Electronics Letters Online No: 19960499

Hun Heo and Sun Young Hwang (*Department of Electronic Engineering, Sogang University, C.P.O. Box 1142, Seoul, 100-611, Korea*)

References

1 SEQUIN, C.: 'Advanced research in VLSI' (MIT Press, 1991), pp. 35–54
2 DEVADAS, S., and KEUTZER, K.: 'Synthesis of robust delay-fault-testable circuits: Theory', *IEEE Trans. Comput. Aid. Design.*, 1992, **11**, (1), pp. 87–101
3 ARMSTRONG, D.: 'On finding a nearly minimal set of fault detection tests for combinational logic nets', *IEEE Trans.*, 1966, **COM-15**, (2), pp. 66–73

Self-timed Manchester chain carry propagate adder

J. Escribá and J.A. Carrasco

Indexing terms: Asynchronous circuits, Adders

The authors present a self-timed adder that uses two Manchester chains to propagate carries in a two-rail code. With the inclusion of buffers in the chains, the adder meets the timing conditions typical of an asynchronous design based in the 'bundled-data, bounded-delay' model and is significantly faster than self-timed adders with restoring logic and similar complexity.

Introduction: Fully delay-insensitive asynchronous digital systems tend to be complex. Simpler designs can be achieved using the 'bundled-data, bounded-delay' model [1, 2] illustrated in Fig. 1*a*. Data is processed using combinational self-timed modules with non-coded data inputs and outputs and control signals I and D, which, in a 4-phase handshaking protocol, go through alternated initialisation and evaluation phases as shown in Fig. 1*b*. Data latching is performed between transitions D^- and I^+, guaranteeing that the self-timed modules will have stable data inputs during the evaluation phase and allowing the overlapping of alternating initialisation-evaluation phases in adjacent self-timed modules.

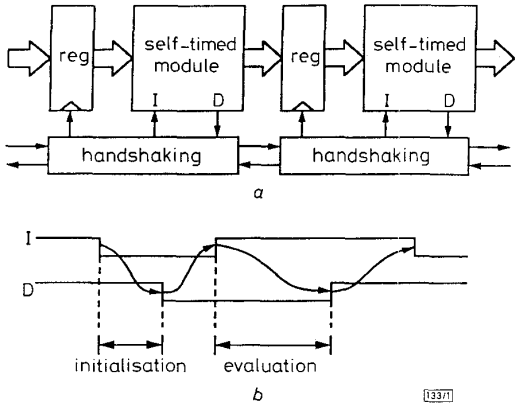

Fig. 1 *Architecture of self-timed system based on 'bundled-data, bounded-delay' model, and module operation of four-phase handshaking protocol*

a Architecture of self-timed system
b Module operation

Self-timed carry propagate adders are attractive because they combine low complexity and good mean delay if the longest carry propagation chain is short in average. For uniform operand distributions it is known that the length of the longest carry propagation chain grows logarithmically with the operand length n and is upper bounded by $\log_2 n - 1/2$ [3]. A self-timed carry propagate adder implemented with DCVSL logic has been described in [2]. A self-timed ALU designed with domino logic appears in [4]. In this Letter we describe a self-timed adder based on the 'bundled-data, bounded delay' model which uses two Manchester chains to compute two-rail coded carry signals. Buffers are inserted in the chains to guarantee timing conditions and reduce delays for long carry propagation chains.

Description and qualitative analysis of the adder: The adder has the iterative structure shown in Fig. 2*a*. The operands are $A_{n-1} \ldots A_0$ and $B_{n-1} \ldots B_0$ (from the most to the least significant bit), and the carry input is C_0. The sum is $S_{n-1} \ldots S_0$ and the carry output is C_n. PGK modules designed using precharged logic (Fig. 2*c*) generate propagation ($P_i = A_i \oplus B_i$), generation ($G_i = A_iB_i$), kill ($K_i = A'_iB'_i$) and negative propagation ($Pbar_i = P'_i$) signals. Two precharged Manchester chains realised with n transistors, faster than p transistors, generate positive logic carry signals C_i, $1 \le i \le n$ and negative logic carry signals $Cbar_i$, $1 \le i \le n$. Signal $Cbar_0$ is obtained from C_0 using a static inverter. The sum bits are obtained using static XOR gates with two-rail inputs (see Fig. 2*b*). Standard static NAND gates generate the stage completion signals D_i, $0 \le i \le n - 1$, which are combined in an n-input C module to generate signal D. Following [5], the C module is designed with a two-input C module fed by NAND and NOR gates with inputs D_i, $0 \le i \le n - 1$ implemented with Wallace NAND/NOR trees.

For a correct operation the data outputs of the adder have to reach their values before transition D^+. Also, module initialisation must finish before transition D^-. We show next that these timing constraints are satisfied if: i) the delay of the XOR gate is smaller than the delay of the completion detection circuitry (NAND gates and n-input C module), and ii) the initialisation time of the PGK modules (delay between I^- and P_i^-, G_i^-, K_i^-) is smaller than the delay of the completion detection circuitry. We analyse the four phases of the handshaking protocol, assuming that at transition D^- the precharge of the PGK modules and Manchester chains is completed, and verify that these conditions are met at the end of the initialisation phase.

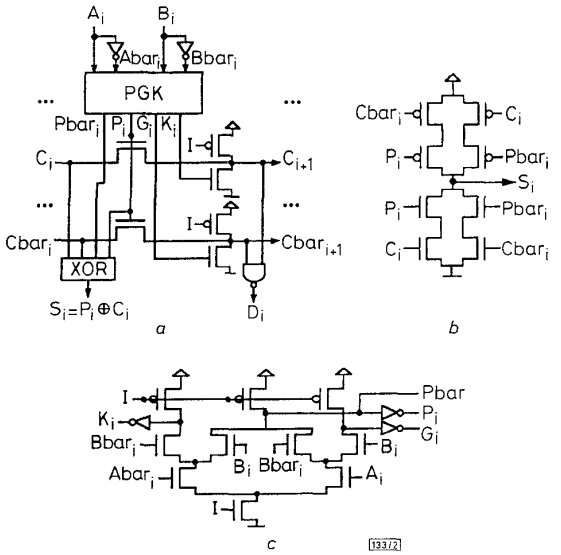

Fig. 2 *Iterative structure of adder, design of XOR gates, and design of PGK modules*

a Adder
b XOR gate
c PGK module

D^- to I^+ phase: Being $I = 0$, the PGK modules stay precharged, independently of the transitions which may occur in the inputs to the modules (data latching occurs in this phase), and no signal transition will be propagated through the adder.

I^+ to D^+ phase (evaluation): The data inputs of the adder will be stable during this phase. I^+ starts the evaluation of the PGK modules and each of them will raise one of the 1-set exclusive signals P_i, G_i, K_i. As these signals are raised, the Manchester chains will be evaluated, falling for each bit i, $1 \le i \le n$ one of the signals C_i, $Cbar_i$. Each of these transitions will originate a D_i^+ transition. Owing to the C module logic, D will raise after the propagation of all the carry bits with a delay equal to the delay of the completion detection circuitry. It has to be shown that D^+ occurs after the latest output transition (C_n^- or latest transition in S_i, $0 \le i \le n - 1$). Since D_{n-1}^+ occurs after C_n^-, D^+ will occur after C_n^-. The latest transition in S_i will not occur before the latest of $Pbar_i^-$, P_i^+, C_i^- and $Cbar_i^-$ plus an XOR gate delay. But, since P_i^+ occurs after $Pbar_i^-$, and P_i^+ implies either C_{i+1}^- or $Cbar_{i+1}^-$, the latest transition in the set S_i, $0 \le i \le n - 1$ will occur no later than the latest transition in the set C_i^-, $Cbar_i^-$, $1 \le i \le n$ plus the delay of an XOR gate, and timing condition (i) ensures that the latest transition in the set S_i, $0 \le i \le n - 1$ will occur no later than D^+.

D^+ to I^- phase: Data inputs do not change and no signal transition will be generated.

Phase I^- to D^- (initialisation): Transition I^- will initiate the precharge of the PGK modules, and the signals $Pbar_i$ at 0 will go to 1 and the signals P_i, G_i and K_i at 1 will go to 0. The Manchester chains will also be precharged. Note that, eventually, all signals P_i, G_i and K_i will be 0, isolating all internal nodes of the chain from ground. There will be transitions D_i^- and a transition D^- with a delay from the end of the precharging of the Manchester chains equal to the delay of the completion detection logic. However, there is no guarantee at the logic level that all $Pbar_i$ will be 1 and all P_i, G_i, K_i will be 0 at the time of transition D^-, since, depending on the strength of the p transistors that precharge the Manchester chains, the precharges could be completed without the signals P_i, G_i, K_i going to 0. This is only guaranteed by timing condition (ii). ■

Quantitative analysis and comparison: We made SPICE simulations for a 32 bit, 1.5 μm CMOS technology implementation of the adder. Using inputs $A = 0$, $B = \max\{0, 2^L - 1\}$, $C_0 = 0, 1$, $0 \le L \le n$, which produce carry propagation chains of length L we verified

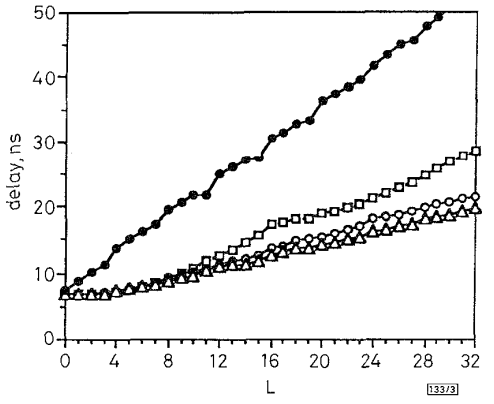

Fig. 3 *Delays for Manchester adder with buffers and DCVSL adder*
—●— DCVSL
—□— $M = 16$
—○— $M = 8$
—△— $M = 4$

that timing condition (ii) was met with the usual 2/3 overlap rule. Timing condition (i) was however not met with that rule owing to the low slopes that the signals C_i and $Cbar_i$ can have for long carry propagation chains and the resulting large delay of the XOR gate. The condition was met after inserting buffers (2 inverters) in the middle of the Manchester chains. Buffer insertion may also have a beneficial impact on the delay of the adder. We analysed the delay characteristics of our adder when buffers are inserted every $M = 2$, 4, 8, and 16 bits. We also analysed the delay of a 32 bit variation of the DCVSL adder described in [2] with n cut transistors to avoid charge transfer problems. Fig. 3 compares the delays for the input vectors producing carry propagation chains of length L and $C_0 = 0$ (the delays for $C_0 = 1$ are similar and exhibit the same qualitative behaviour). We do not include the delays for $M = 2$, which are very similar to the delays for $M = 8$. Our adder with $M = 4$ has a smaller delay for any L and will be faster than any of the other adders for any input distribution. This is relevant, since recent workload characterisations [4] have shown that realistic operand distributions are often far from the uniform distribution. For uniform input distribution the mean delays were 12.59 ± 0.40 ns for the DCVSL adder, and 8.42 ± 0.14 ns, 8.26 ± 0.12 ns, 8.29 ± 0.14 ns, and 8.39 ± 0.16 ns for our adder with, respectively, $M = 2$, $M = 4$, $M = 8$, and $M = 16$; with a 95% confidence interval in all cases. Without including completion detection logic and the precharge distribution trees, which are similar for both adders, and the buffers and inverter generating $Cbar_0$ of the Manchester adder, the Manchester adder has 34 transistors per bit, while the DCVSL adder has 36 transistors per bit. Thus both adders have similar complexity. The delay overhead owing to completion detection was 32% for our adder.

23 January 1996
Electronics Letters Online No: 19960512

J. Escribá and J.A. Carrasco (*Departament d'Enginyería Electrònica, Universitat Politècnica de Catalunya, Diagonal 647, plta. 9, 08028 Barcelona, Spain*)

References

1 MENG, T., BRODERSEN, R.W., and MESSERSCHMITT, D.G.: 'Automatic synthesis of asynchronous circuits from high level specifications', *IEEE Trans. Comput. Aided Design*, 1989, **CAD–8**, (11), pp. 1185–1205

2 JACOBS, G.M., and BRODERSEN, R.W.: 'A fully asynchronous digital signal processor using self-timed circuits', *IEEE J. Solid-State Circuits*, 1990, **SSC–25**, (6), pp. 1526–1537

3 BRILEY, B.E.: 'Some new results on average worst-case carry', *IEEE Trans. Comput.*, 1973, **22**, (5), pp. 459–463

4 GARSIDE, J.D.: 'A CMOS VLSI implementation of an asynchronous ALU'. Proc. IFIP WG10.5 Working Conf. Asynchronous Design Methodologies, Manchester, UK, March 1993, pp. 181–192

5 WUU, T.-Y., and VRUDHULA, S.B.K.: 'A design of a fast and area efficient multi-input Muller C-element', *IEEE Trans. VLSI Syst.*, 1993, **1**, (2), pp. 215–219

Asynchronous VLSI architecture for adaptive echo cancellation

R.P. Mackey, J.J. Rodríguez, J.D. Carothers and S.B.K. Vrudhula

Indexing terms: VLSI, Adaptive systems, Echo suppression

A single chip, 128 coefficient, asynchronous echo canceller is presented. Cancellation is performed by an FIR filter whose coefficients are adapted using the power-of-two modified LMS algorithm. The pipelined circuit updates all coefficients and generates the filtered output every cycle while allowing a sampling rate >206.5kHz.

Introduction: In a communication system echo occurs when the received signal is coupled onto the desired output signal, this is owing to transmission line mismatches and reflections off objects. The frequency response of the echo path is time-varying and environment dependent, thus requiring adaptive filtering for echo cancellation: e.g. consider the two-way communication system shown in Fig. 1. The primary signals are the received data $r(n)$, desired transmit data $t(n)$, distorted transmit data $d(n)$, echo $y(n)$, echo estimate $\hat{y}(n)$, and output/error $e(n)$. The echo $y(n)$ can only be estimated; therefore, $e(n) = t(n) + \epsilon(n)$, where $\epsilon(n)$ is the uncancelled error. Echo cancellation is performed by adapting the filter to minimise $\epsilon(n)$. This process requires both echo subtraction and echo signature reestimation. Implementations normally perform these tasks using separate hardware units when working with a large number of coefficients.

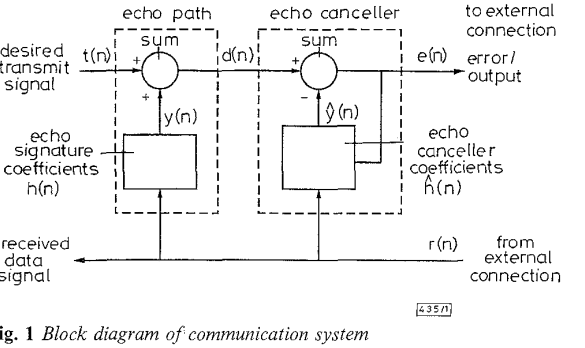

Fig. 1 *Block diagram of communication system*

The LMS algorithm is a gradient descent algorithm for estimating the echo signature [6]. The original LMS algorithm suffers from either long processing delay or high hardware requirements — hence, the need for separate processors in most implementations. In the implementation presented here, the order of computation in the LMS algorithm was modified in order to achieve an efficient, single-processor implementation.

Synchronous against asynchronous design: In synchronous design, a clock is used to control the circuit, and its speed is governed by the longest path in any combinational logic block (CLB). Most systems today process data in a pipeline which is broken into stages consisting of register-CLB pairs. As CLB delays decrease, clock frequencies can increase. However, owing to design issues such as clock distribution and skew, the maximum possible clock frequencies are not being achieved in practice.

Asynchronous design eliminates clock skew and distribution problems by using a local handshaking protocol instead of a clock. The normal flow of an asynchronous system includes (i) presentation of valid data, (ii) request line activation, and (iii) acknowledge line activation when data has been latched. Sutherland has shown that synchronously pipelined circuits can also be asynchronously pipelined. Furthermore, the resulting asynchronously pipelined circuit can have lower latency [5].

Implementation: Conventional, LMS echo cancellers are governed by