# Self-timed multiplier based on canonical signed-digit recoding

G.A.Ruiz and M.A.Manzano

**Abstract:** A data-dependent self-timed multiplier structure in dynamic logic and DCVS logic based on canonical signed-digit (CSD) recoding is presented. This coding increases the number of null partial products up to 33%, compared with the 25% of the traditional modified Booth recoding. The carry–save structure is a data-dependent parallel array, which uses this characteristic to reduce the number of addition operations, and thus increase the speed of the multiplier by 20% compared with other classical implementations. Thus, the adders of a null partial product become pass cells postponing the addition operation to the next stage. The layouts of a 16 × 16-bit and a 32 × 32-bit signed CSD multiplier have been devised. These present a rectangular-shaped structure and regular layout suitable for implementation in VLSI. Simulation results highlight that these CSD multipliers have a similar throughput to other pipeline asynchronous multipliers, but with a significant reduction of latency. The delay computation time is less than for published synchronous multipliers. However, the cost in terms of area and power is high.

## 1 Introduction

Asynchronous digital logic appears to be a valid alternative to its synchronous counterparts in VLSI digital systems, since it is free of clock skew problems (it does not use a global clock) [1] and has a substantial advantage in power and noise-sensitive applications [2]. Asynchronous circuits are designed for average-case, rather than worst-case, performance at the maximum speed, which allows for the intrinsic hardware delays, variations in fabrication process and operating conditions. However, the design of these circuits is more difficult because of hazard and race problems, and, in many cases, they lack the required supporting design tools [3].

The 'speed-independent' self-timed circuits are a 'purist' asynchronous approach, which uses dual-rail data recoding and completion detection logic to determine when an operation has finished. This approach enables a handshaking protocol to be established between computation blocks, independent of both circuit and interconnection delays. The dynamic differential cascode voltage switch (DCVS) logic has complementary outputs [4], which provide a general way of generating completion signals. It also has an inherent self-testing property to implement fault-tolerance circuits at low cost [5]. These completion signals can produce a performance improvement when there is a data-dependent variation in computation time. For example, Reitwiesner [6] and Hendrickson [7] demonstrated that, in an asynchronous $n$-bit ripple–carry adder, the average longest carry can be approximated by $\log_2(5n/4)$.

© IEE, 2001

*IEE Proceedings* online no. 20010524

*DOI*: 10.1049/ip-cds:20010524

Paper first received 13th March 2000 and in revised form 9th April 2001

The authors are with the Departamento de Electronica y Computadores, Facultad de Ciencias, Avda. de Los Castros s/n, 39005-Santander, Spain

The self-timed array multiplier with Booth recoding in DCVS logic presented by Meng [1] is an example of a speed-independent design. Its main drawback is that the computational delay of its carry–save structure is practically independent of input data patterns since there is no carry propagation. This drawback is partly solved using the concept of data-dependent computation time in carry–save arrays introduced in by Kearny and Berman [8] and applied to basic bundled asynchronous parallel array multipliers and iterative multipliers. However, most proposed designs of asynchronous multipliers are based on pipeline and micropipeline structures, which achieve a high throughput in a low area, but with a significative latency [9–17].

In this paper, we present an efficient structure of a speed-independent self-timed multiplier based on a data dependent carry–save array with CSD recoding. The CSD recoding [18] generates an average of 33% of null partial products, compared with 25% of the traditional modified Booth recoding of synchronous multipliers. When a partial product is null, the adders are turned into simple pass cells, postponing the addition operation to the next stage. In this way, the number of arithmetic operations in the carry–save structure is reduced, increasing its speed and lowering its power. The characteristics of CSD recoding and Booth recoding are analysed and compared. The self-timed signed CSD multiplier structure in mainly DCVS logic is described. The simulation results of 16 × 16-bit and 32 × 32-bit multipliers are presented and compared with other implementations.

## 2 Booth recoding against CSD recoding

The radix-4 modified Booth algorithm [19] has been widely used in high-speed multiplier structures. In an $m$-bit by $n$-bit multiplication ($A \times B$), the multiplier $B$ represented in two's complement is converted into a radix-4 redundant signed digit (SD) representation $D_J = \{\pm 2, \pm 1, 0\}$, with the

result that

$$B = -2^{n-1}B_{n-1} + \sum_{j=0}^{n-2} 2^j B_j$$
$$= \sum_{J=0}^{(n/2)-1} (B_{2J-1} + B_{2J} - 2B_{2J+1})4^J = \sum_{J=0}^{(n/2)-1} D_J 4^J \quad (1)$$

with $B_{-1} = 0$. This recoding reduces the design of an $n$-bit ($n$ is even) synchronous multiplier exactly to $n/2$ partial products, which can be easily calculated by shifting and by add/subtract operations. The CSD algorithm (whose rules for radix-4 are summarised in Table 1) simplifies to a minimum the number of add/subtract operations, finding the minimal non-redundant SD representation in terms of the number of non-zero digits. Table 2 shows the average percentage of digit values for different lengths resulting from the Booth and CSD recoding scheme. It can be observed that the probability that the digit $D_J$ is 0 is always higher in the CSD recoding than in the Booth recoding. $D_J = 0$ means that no operation needs to be performed on the partial product ($J$); it is enough to transmit data between the previous partial product ($J - 1$) and the subsequent one ($J + 1$).

**Table 1: Recoding scheme of radix-4 CSD algorithm and 5-out-of-1 coding signals {$N_J$, $X_J$, $2X_J$, $Y_J$, $2Y_J$}**

| $B_{j+1}$ | $B_j$ | $B_{j-1}$ | $CI$ | $D_J$ | $CO$ | $N_J$ | $X_J$ | $2X_J$ | $Y_J$ | $2Y_J$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 2 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | –1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | –2 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | –1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 2 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | –1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | –2 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | –1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

**Table 2: Average percentage of digit values in radix-4 SD representation using CSD and Booth recoding for different numbers of bits**

| $D_J$ | 16-bit | | 32-bit | | 64-bit | | ∞-bit | |
|---|---|---|---|---|---|---|---|---|
| | CSD | Booth | CSD | Booth | CSD | Booth | CSD | Booth |
| 0 | 32.6 | 25.0 | 32.7 | 25.0 | 33.1 | 25.0 | 33.3 | 25.0 |
| +1 | 18.0 | 25.0 | 17.4 | 25.0 | 17.0 | 25.0 | 16.6 | 25.0 |
| +2 | 15.7 | 10.9 | 16.3 | 11.7 | 16.5 | 12.1 | 16.6 | 12.5 |
| –1 | 18.0 | 25.0 | 17.3 | 25.0 | 17.0 | 25.0 | 16.6 | 25.0 |
| –2 | 15.8 | 14.1 | 16.4 | 13.3 | 16.5 | 12.9 | 16.6 | 12.5 |

In a self-timed multiplier, this characteristic can be used to increase the speed of the multiplier by reducing the number of partial products with arithmetic operations. The complexity of this multiplier is slightly greater than other standard self-timed multipliers based on Booth recoding for two reasons: (i) the CSD recoder is slightly more complex than the Booth recoder in order to propagate the carry, and (ii) the partial products must include the non-arithmetic operation or the null operation ($D_J = 0$). However, electrical simulations at transistor level comparing the two multipliers show that the CSD recoding self-timed multipliers are 20% faster than the self-timed Booth multipliers with a small area cost.

The CSD recoder has been carried out on a 5-out-of-1 code $\{N_J, X_J, 2X_J, Y_J, 2Y_J\}$ representing each of the values of the SD representation {0, +1, +2, –1, –2}. These signals (defined in Table 1) are given by

$$N_J = \overline{B}_j \overline{B}_{j-1} \overline{CI} + B_j B_{j-1} CI$$
$$X_J = \overline{B}_j (B_{j-1} \oplus CI)$$
$$2X_J = \overline{B}_{j+1} (B_j \overline{B}_{j-1} \overline{CI} + \overline{B}_j B_{j-1} CI)$$
$$Y_J = B_j (B_{j-1} \oplus CI)$$
$$2Y_J = B_{j+1} (B_j \overline{B}_{j-1} \overline{CI} + \overline{B}_j B_{j-1} CI) \quad (2)$$

Figs. 1*a* and *b* show the implementation of eqn. 2 in dynamic domino logic, including output buffers for the driving load capacity requirement. These gates use compact transistor sharing structures to reduce the number of transistors, and they have weak PMOS transistors (labelled *) fed back from the outputs to solve the charge sharing and current leakage problems.

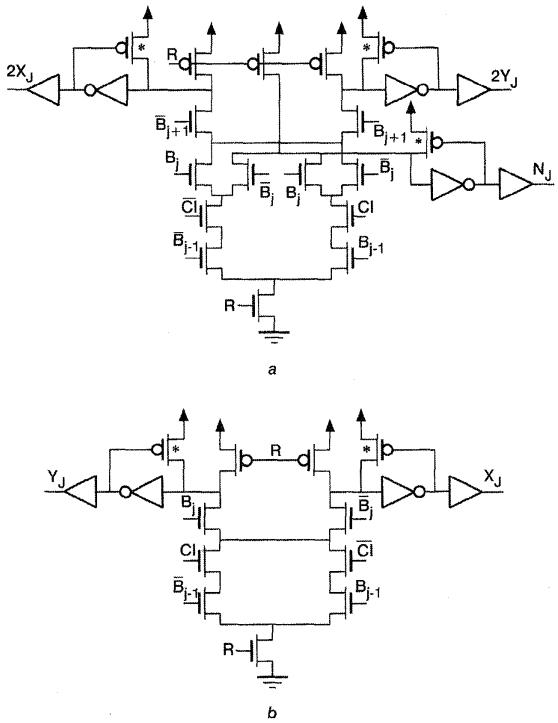


**Fig. 1** *Implementation of CSD recoding signals in domino logic*
*a* Generation of $N_J$, $2X_J$, $2Y_J$
*b* Generation of $Y_J$ and $X_J$

The carry-out of the CSD recoder and its complement are defined as

$$CO = B_{j+1} B_{j-1} CI + B_j (CI + B_{j-1} + B_{j+1})$$
$$\overline{CO} = \overline{B}_{j+1} \overline{B}_{j-1} \overline{CI} + \overline{B}_j (\overline{CI} + \overline{B}_{j-1} + \overline{B}_{j+1}) \quad (3)$$

Fig. 2 shows the implementation in DCVS logic of eqn. 3.

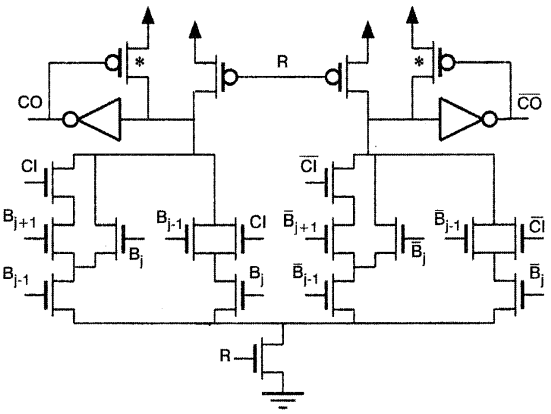


**Fig. 2** *Carry generation of CSD recoder in DCVS logic*

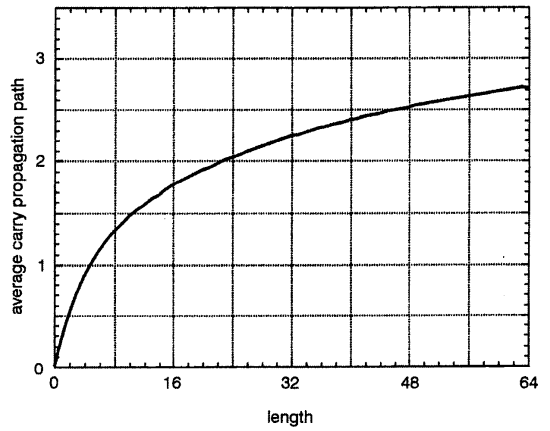


**Fig. 3** *Average size of longest carry propagation path for different lengths of CSD decoder*

In the CSD recoder, similar to an asynchronous adder, the average carry propagation path is lower than the worst-case path. Fig. 3 shows the average size of the longest carry propagation path for different lengths, assuming random data distribution. These experimental results show that the average length of the longest carry sequence is bounded by $\sim 0.5\log_2 n$. This means that the average carry propagation is far lower than the worst-case propagation, which leads to the minimisation of the average computation time of the CSD recoder.

## 3 Self-timed signed CSD multiplier

The architecture of an (8 × 8)-bit self-timed signed CSD multiplier is shown in Fig. 4; this architecture can be easily extended to an $n \times m$-bit multiplier. This is composed of an array ($n/2$) of CSD recoders described above, an array ($n/2$) of partial products made up mainly of half/full adders and selector circuits, and a final $2n$-bit ripple-carry adder, with a completion circuit to generate the *Done* signal. A 'high' *Done* indicates that the multiplication has finished. The completion circuit is based on a tree-like structure of dynamic Or-And-Invert gates described elsewhere [20].

The cell PFA represents one bit in a partial product and is made up of two components: a selector (PP) and a full adder (FA). This cell also works in non-arithmetic operation mode when the control signal is $N_J = 1$; in this mode, it simply transmits data between the previous and the subsequent partial products. The PP selector carries out the selection of the bit $A_i$, $A_{i-1}$, $\bar{A}_i$ or $\bar{A}_{i-1}$ of the partial product according to the control signals, in order to perform the operation of $A$, $2A$, $-A$ y $-2A$. The logic equations for the output PP and PN signals of this selector are given by

$$PP_{J,i} = 2Y_J\bar{A}_{i-1} + Y_J\bar{A}_i + 2X_JA_{i-1} + X_JA_i$$
$$PN_{J,i} = 2X_J\bar{A}_{i-1} + X_J\bar{A}_i + 2Y_JA_{i-1} + Y_JA_i \quad (4)$$

Note that $PP_{J,i} = \overline{PN}_{J,i}$ when $N_J = 0$. Otherwise, when $N_J = 1$, these signals are inactive ($PP_{J,i} = PN_{J,i} = 0$) since $X_J = 2X_J = Y_J = 2Y_J = 0$. Fig. 5 shows the implementation in dynamic logic of eqn. 4. This implementation is not strictly DCVS logic since precharging voltages are maintained when $N_J = 1$ without switching the outputs (no evaluation phase is made). In this operational mode, the power

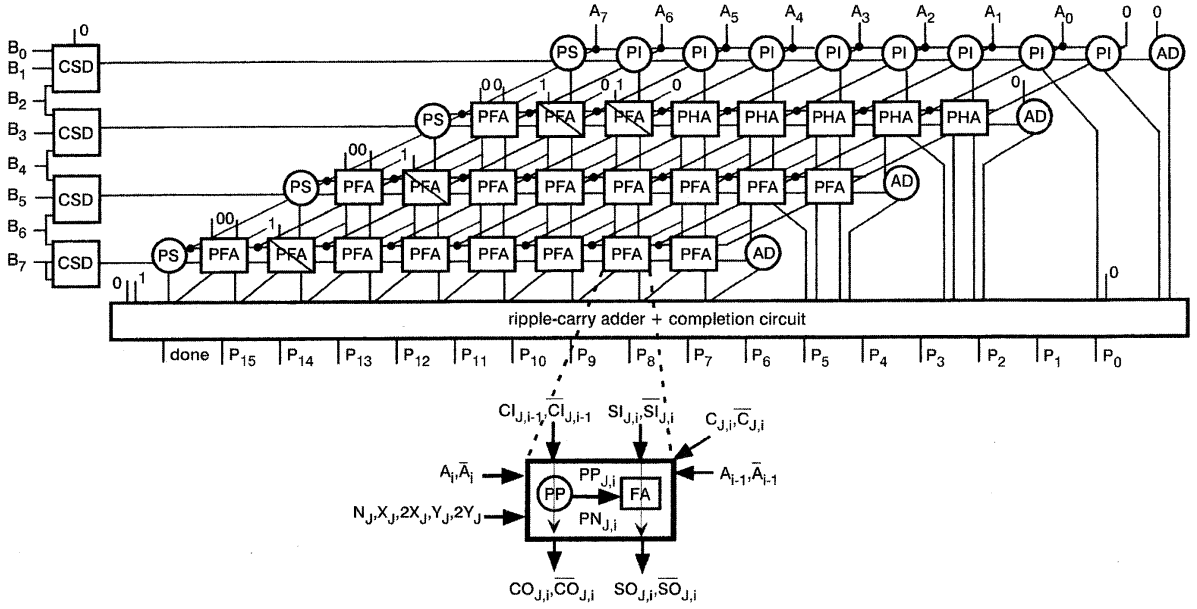


**Fig. 4** *Architecture of (8 × 8)-bit self-timed signed CSD multiplier*

consumption of the circuit in Fig. 6 is reduced, as there is no discharge of either the dynamic nodes or of the outputs.

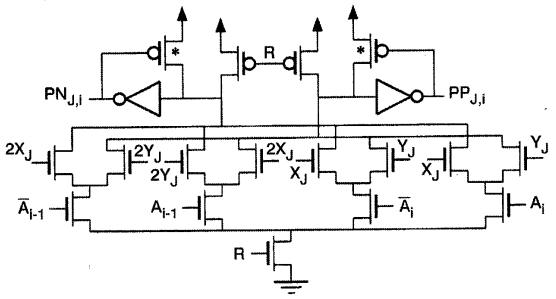


**Fig. 5** *PP cell in dynamic logic*

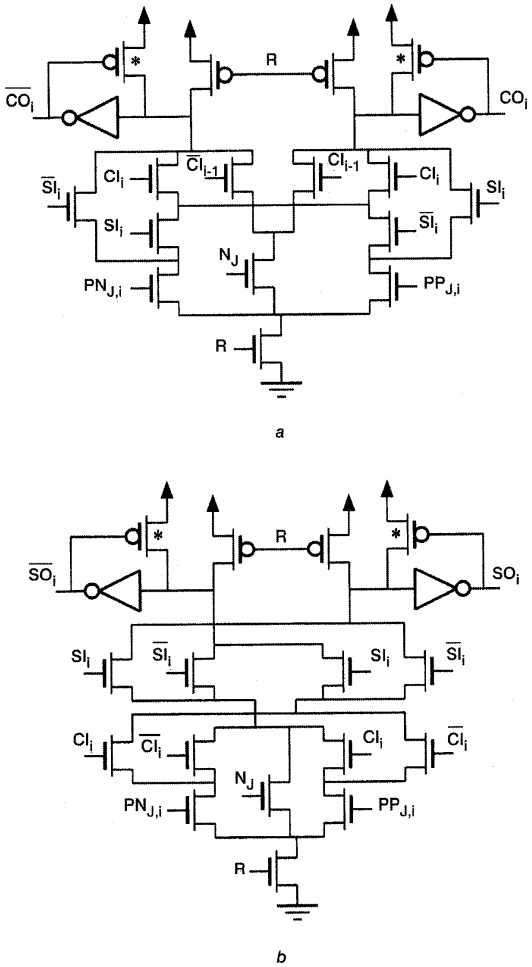


**Fig. 6** *FA cell in DCVS logic*
*a* Sum gate
*b* Carry gate

The FA adds the bit of selector PP to the output of the previous partial product. The Boolean expression for the FA is

$$SO_{J,i} = PP_{J,i}\left(\overline{CI_{J,i} \oplus SI_{J,i}}\right) + PN_{J,i}\left(CI_{J,i} \oplus SI_{J,i}\right) + N_J SI_{J,i}$$

$$\overline{SO}_{J,i} = PP_{J,i}\left(CI_{J,i} \oplus SI_{J,i}\right) + PN_{J,i}\left(\overline{CI_{J,i} \oplus SI_{J,i}}\right) + N_J \overline{SI}_{J,i}$$

$$CO_{J,i} = PP_{J,i}SI_{J,i} + \left(PN_{J,i}SI_{J,i} + PP_{J,i}\overline{SI}_{J,i}\right)CI_{J,i} + N_J CI_{J,i-1}$$

$$\overline{CO}_{J,i} = PN_{J,i}\overline{SI}_{J,i} + \left(PN_{J,i}SI_{J,i} + PP_{J,i}\overline{SI}_{J,i}\right)\overline{CI}_{J,i} + N_J \overline{CI}_{J,i-1} \quad (5)$$

Fig. 6 shows the sum gate and carry gate in DCVS logic of FA. This FA only has four more transistors than the implementation proposed by Meng ([1], p. 99). Its purpose is to transform the FA into a pass circuit when $N_J = 1$: $SI_{J,i}$, $\overline{SI}_{J,i}$ is transmitted directly to the output $SO_{J,i}$, $\overline{SO}_{J,i}$ and $CI_{J,i-1}$, $\overline{CI}_{J,i-1}$, which is the input carry of the right-hand FA, at output $CO_{J,i}$, $\overline{CO}_{J,i}$.

In the partial product $J = 1$ the PHA cell is used, since only the addition of two bits is necessary. This cell is made up of the PP selector and a half-adder (HA), whose Boolean expression is given by

$$\begin{aligned} SO_{J,i} &= PN_{J,i}SI_{J,i} + PP_{J,i}\overline{SI}_{J,i} + N_J SI_{J,i} \\ \overline{SO}_{J,i} &= PP_{J,i}SI_{J,i} + PN_{J,i}\overline{SI}_{J,i} + N_J \overline{SI}_{J,i} \\ CO_{J,i} &= PP_{J,i}SI_{J,i} \\ \overline{CO}_{J,i} &= \overline{SI}_{J,i} + PN_{J,i} + N_J \end{aligned} \quad (6)$$

The multiplier also uses two cells; the PI selector and the AD cell. PI, has a function similar to the PP selector and performs the selection of the bit 0, $A_i$, $A_{i-1}$, $\overline{A}_i$, $\overline{A}_{i-1}$ of the first partial product ($J = 0$). The logical expression for the PI is obtained from eqn. 4 resulting in

$$\begin{aligned} PI_i &= 2Y_0\overline{A}_{i-1} + Y_0\overline{A}_i + 2X_0A_{i-1} + X_0A_i \\ \overline{PI}_i &= 2X_0\overline{A}_{i-1} + X_0\overline{A}_i + 2Y_0A_{i-1} + Y_0A_i + N_0 \end{aligned} \quad (7)$$

The AD cell performs the 'add 1' operation to LSB required to take two's complement of a partial product. The Boolean equation of this cell is

$$\begin{aligned} AD_J &= Y_J + 2Y_J + N_J CI \\ \overline{AD}_J &= X_J + 2X_J + N_J \overline{CI} \end{aligned} \quad (8)$$

The sign extension has been resolved using the 'sign generate' method described by Annaratone [21]. This method applied to an $n \times m$-bit multiplier allows the sign bit (Sgn) to be defined as

$$\mathrm{Sgn} = 2^m + \sum_{J=0}^{n/2-1} 2^{2J+m+1} + \sum_{J=0}^{n/2-1} \overline{S}_J 2^{2J+m} \quad (9)$$

where $S_J$ is the sign bit for each partial product. The implementation of each of the three terms of eqn. 9 are carried out in cells PFA marked with a diagonal line and in cell PS whose logical expressions are

$$\begin{aligned} PS_J &= N_J + \overline{A}_{m-1}(X_J + 2X_J) + A_{m-1}(X_J + 2X_J) \\ \overline{PS}_J &= A_{m-1}(X_J + 2X_J) + \overline{A}_{m-1}(X_J + 2X_J) \end{aligned} \quad (10)$$

## 4 Realisation and circuit simulation

A 16 × 16-bit and a 32 × 32-bit self-timed signed CSD multiplier were designed in a standard 0.6 μm n-well CMOS

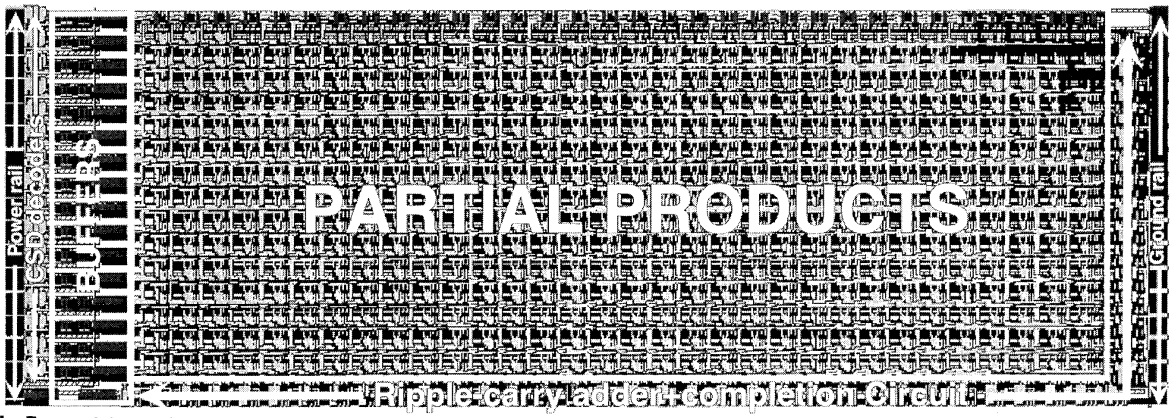


**Fig. 7** *Mask layout of 16 × 16-bit CSD multiplier*

double-metal process technology (Austria Mikro Systeme International AG Technology). Fig. 7 shows only the mask layout, with the different blocks for the 16 × 16-bit multiplier designed using the Design Framework II tool from CADENCE; the chip has not been processed. The final size of this multiplier is 2.43mm × 0.83mm (2mm$^2$) and the total number of transistors is 10931, ~1000 of which correspond to fed-back PMOS transistors. The 32 × 32-bit multiplier has an area of 4.47mm × 1.57mm (7mm$^2$), with 36897 transistors (3500 are fed-back PMOS transistors).

The total area of both circuits is (except for some small variations) divided as follows: partial products (68%), CSD recoder (3.5%), final adder and completion circuit (17%), buffers (8%) and power rail (3.5%). The electrical simulation was made with HSPICE at level 49, 25°C and $V_{DD}$ = 5V and 3.3V, using typical device parameters and very precise extracted capacitances.

Our multiplier has an iterative parallel-array structure, with rectangular-shaped and regular layout suitable for implementation in VLSI. The problem associated with DCVS logic (and, a general problem inherent in all differential logic) is the large area required for routing, resulting from the need to duplicate the interconnection lines. For example, the carry–save multiplier described by Meng [1] dedicates 50% of its area to routing, owing mainly to its highly irregular interconnections. The BCL adder presented previously [22] (even though it has great regularity) uses 35% of routing area due to the high number of interconnecting cells. In the proposed multiplier, the iterative parallel-array structure substantially reduces this area to 24%, resulting in a less area-consuming implementation.

The completion signal *Done* indicates when the multiplication process has finished. The multiplication time is measured from when signal *R* goes to high until *Done* goes to high. This time includes the delay in the distributed control buffers. Table 3 shows the simulation results for a different power supply. The addition of average computation time ($t_{ave}$) and precharging time ($t_{pre}$) is equivalent to an effective throughput rate of 159MHz at $V_{DD}$ = 5V (108MHz at 3.3V) for the 16 × 16-bit multiplier and 104MHz (72MHz) for the 32 × 32-bit multiplier. The speed of these multipliers is a result of the reduction in the evaluation time of the FA cell when it works in non-operation mode ($N_J$ = 1). This time varies for the sum output of 0.464ns ($N_J$ = 0) at 0.309ns ($N_J$ = 1) for $V_{DD}$ = 5V, 0.664ns at 0.450ns for $V_{DD}$ = 3.3V, achieving a reduction of 35%. Similar results are obtained for the carry output with 0.469ns ($N_J$ = 0) at 0.302ns ($N_J$ = 1) for $V_{DD}$ = 5V and 0.676ns at 0.454ns for $V_{DD}$ = 3.3V.

**Table 3: Simulation results of 16 × 16-bit and 32 × 32-bit CSD multipliers**

| | $t_{ave}$ | | $t_{pre}$ | | $t_{wc}$ | |
|---|---|---|---|---|---|---|
| | 3.3V | 5V | 3.3V | 5V | 3.3V | 5V |
| 16×16-bit | 7.95ns | 5.34ns | 1.25ns | 0.95ns | 16.3ns | 10.6ns |
| 32×32-bit | 12.46ns | 8.58ns | 1.36ns | 1.03ns | 18.5ns | 12.4ns |

$t_{ave}$: average computation time, $t_{pre}$: precharging time, $t_{wc}$: worst-case delay time

Table 3 reflects that the average computation time is substantially lower than the worst-case delay time resulting from the multiplication of –1 × 1. This difference is far more significant for the 16 × 16-bit multiplier than for the 32 × 32-bit multiplier. The computation time probability provides good information on how the average function delays of these multipliers may govern overall throughput rates, resulting in higher performance. Fig. 8 shows the approximate distribution curves of completion signal *Done* at $V_{DD}$ = 3.3V, whose maximum values represent the average multiplication time; distribution curves obtained for $V_{DD}$ = 5V have a similar form. These results clearly show that these multipliers operate with an average computation time that is 61% less for the 16 × 16-bit and 33% less for the 32 × 32-bit with respect to the worst-case delay time.

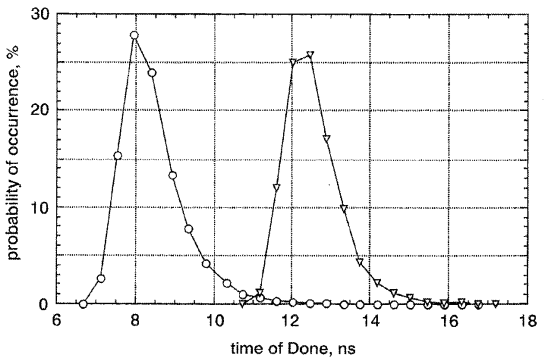


**Fig. 8** *Distribution of Done signal for $V_{DD}$ = 3.3V*
○ 16 × 16-bit
▽ 32 × 32-bit

There is a wide variety of asynchronous approach and design styles proposed in the design of asynchronous multipliers, according to speed, area and power consumption. Most of these implementations are based on pipeline and micropipeline structures as a result of a trade-off between throughput, latency and chip area.

**Table 4: Performance comparison of some asynchronous and synchronous multipliers**

| Asynchronous | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Ref. | Size | Techn. | Area | Trans. | $V_{DD}$ | Power | Thro. | Lat. |
| [9] | 16 × 16 | 1.0 CMOS | 2.59 | | 5 | 2.19 | 156 | 64 |
| [1] | 16 × 16 | 1.6 CMOS | 7 | | 5 | | 23 | — |
| [12] | 16 × 16 | 1.2 CMOS | 3.03 | | 5 | | 180 | 25 |
| [11] | 4 × 4 | 1.0 CMOS | 4.6 (i.p.) | 4213 | 5 | 2.36 | 195 | 29.2 |
| | | | | | 4 | 1.14 | 162.5 | 35 |
| [16] | 24 × 24 | 1.0 GaAs | 25 | | 5 | 46.7 | 76 | 24 |
| [10] | 4 × 4 | 1.2 CMOS | 5.3 (i.p.) | 6800 | 5 | | 120 | 33.3 |
| [14] | 8 × 8 | 0.6 CMOS | 1 (i.p.) | 6428 | 5 | 0.43 | 250 | 16.4 |
| [13] | 16 × 16 | 0.6 CMOS | 0.18 | | 5 | | 92 | 15 |
| This work | 16 × 16 | 0.6 CMOS | 2.026 | 10931 | 5 | 13.7 | 159 | 5.3 |
| | | | | | 3.3 | 4.7 | 109 | 7.9 |
| | 32 × 32 | 0.6 CMOS | 7.045 | 36897 | 5 | 48 | 104 | 8.6 |
| | | | | | 3.3 | 16.5 | 72 | 12.5 |
| Synchronous | | | | | | | | |
| Ref. | Size | Techn. | Area | Trans. | $V_{DD}$ | Power | Delay | |
| [23] | 32 × 32 | 0.8 CMOS | 7.26 | 27704 | 5 | 27.7 | 15 | |
| [24] | 54 × 54 | 0.8 CMOS | 12.9 | 82500 | 5 | 21.87 | 13 | |
| [25] | 54 × 54 | 0.5 CMOS | 12.5 | 81600 | 3.3 | 8.7 | 10 | |

Ref.: Reference; Size: Length of multiplier operands; Techn.: Technology;
Area: Chip area (in $mm^2$, i.p.: including pads); Trans.: Number of transistors; $V_{DD}$: power supply (in V);
Power: Average consumption power supply (in mW/MHz); Thro.: Throughput (in MHz);
Lat.: Latency (in ns); Delay: Delay time (in ns)

Table 4 shows the characteristics of some asynchronous multiplier designs and of the proposed CSD multipliers. A fully four-stage pipeline multiplier with Manchester chain adders combined with carry–save adder architecture in latched differential pass transistor logic (LDPL) logic has been presented elsewhere [9]. Meng [1] implemented a speed-independent multiplier based on Booth decoding, with carry–save architecture in DCVS logic. A five-stage pipeline multiplier with carry–save adder architecture using progressive evaluation technique (bundled data) is described by Burford *et al.* [12]. Acosta *et al.* [11] present another pipeline multiplier with an array-based architecture in switched-output differential structure (SODS) logic. Chandramouli *et al.* [16] implemented in GaAs a micropipeline parallel multiplier with a partial array of array architecture, using bundled data-path and Booth decoding in direct-coupled FET logic (DCFL). The wave pipelining technique has been applied to a fully asynchronous design of a carry–save multiplier in balanced CMOS gates [10]. Chiang and Lao [14] describe a ten-stage pipeline multiplier with Booth decoding and carry–save architecture in complementary pass-gate logic (CPL). Kearny and Bergmann [13] present a data-dependent multiplier with iterative data path in DCVS logic. Table 4 shows clearly that the 16 × 16-bit and 32 × 32-bit CSD multiplier presents a high throughput (without latency), far higher even than some pipeline structures with similar characteristics [13], but at the expense of area and power.

Table 4 also shows some representative static synchronous multipliers based on the Wallace tree, with a technology similar to that used in CSD multipliers. It was highlighted [9, 11] that synchronous multipliers are always better than pipeline asynchronous multipliers because of the cost in area and delay overhead of local control circuitry. However, the 32 × 32-bit CSD multiplier has a $t_{ave}$ of 8.58ns, which is a smaller delay time than the 15ns of the multiplier of the same size of Schuman *et al.* [23], but has a higher cost in area and power. This comparison is relative because the CSD multiplier is dynamic and the synchronous multiplier static, and the used technology is 0.6μm and 0.8μm, respectively.

## 5 Conclusions

We have presented a self-timed multiplier structure in dynamic and DCVS logic, with a data-dependent carry–save structure more efficient than the carry–save structure presented by Meng [1] whose computation time of its carry–save structure is practically independent of input data patterns. The multiplier uses a CSD recoding, which generates the highest value of null partial products (33%). A null partial product means that the adders become pass cells. The cost of including this non-arithmetic operation is four transistors per adder out of a total of 40 transistors, achieving a reduction in delay time of 35% in this operational mode.

The layout of a 16 × 16-bit and 32 × 32-bit signed CSD multiplier presents a rectangular-shaped structure and regular layout suitable for implementation in VLSI. These multipliers have a throughput comparable with other pipeline asynchronous multipliers but with a significant reduction in latency. The average computation time is 61% less for the 16 × 16-bit and 33% less for the 32 × 32-bit with respect to the worst-case delay time; this average time is apparently less than for published synchronous multipliers. However, the cost in area and power is high. The selection of an asynchronous multiplier implementation should be made as a result of a trade off between throughput, latency and chip area.

Finally, the CSD multiplier is a fault-secure circuit because of the characteristics of DCVS logic and the 5-out-of-1 code used on recoding. This permits concurrent error

detection for all single faults and a large percentage of multiple faults, highly suitable for applications with high-reliability requirements and high-integrity data-processing.

## 6 References


1 MENG, T.H.: 'Synchronization design for digital systems' (Kluwer Academic Publishers, Boston, 1991)
2 PAVER, N.C., DAY, P., FANSWORTH, C., JACKSON, D.L., LIEN, W.A., and LIU, J.: 'A low-power, low noise, configurable self-timed DSP'. Cogency Technology, October 1997, (available via http://www.cogency.com)
3 HAUCK, S.: 'Asynchronous design methologies: An overview', *Proc. IEEE,* 1995, **83**, (1), pp. 69–93
4 CHU, K.M., and PULFREY, D.L.: 'Design procedures for differential cascode voltage switch circuits', *IEEE J. Solid-State Circuits,* 1986, **SC-21**, (6), pp. 1082–1087
5 KANOPOULUS, N., PANTZARTZIS, D., and BARTRAM, F.R.: 'Design of self-checking circuits using DCVS logic: a case study', *IEEE Trans. Comput.,* 1992, **41**, (7), pp. 891–896
6 REITWIESNER, G.W.: 'The determination of carry propagation length for binary addition', *IRE Trans. Electron. Comput.,* 1960, **9**, pp. 35–38
7 HENDRICKSON, H.C.: 'Fast high-accuracy binary parallel addition', *IRE Trans. Electron. Comput.,* 1960, **9**, pp. 465–469
8 KEARNY, D.A., and BERMANN, N.W.: 'Bundled data asynchronous multipliers with data dependent computation times'. Proceedings of 3rd international symposium on Advanced Research in Asynchronous Circuits and Systems, IEEE Computer Society, 1997, pp. 186–197
9 SALOMON, O., and KLAR, H.: 'Self-timed fully pipelined multipliers'. Proceedings of IFIP WG 10.5 working conference on Asynchronous Design Methodologies, Manchester, UK, 1993, pp. 45–55
10 SCHUMANN, T., KLAR, H., and HOROWITZ, M.: 'A fully asynchronous, high throughput multiplier using wave-pipeline for DSP custom applications'. Mikroelektronik für die Informationstechnik, Chemnitz, 18–19 March 1996, pp. 283–286 (TG-Fachbericht, (138), VDE Verlag, Vorträge der ITG-Fachtagung)
11 ACOSTA, A.J., JIMENEZ, R., BARRIGA, A., VALENCIA, M., and HUERTAS, J.L.: 'Design and characterization of a CMOS VLSI self-timed multiplier architecture based on a bit-level pipelined-array structure', *IEE Proc, Circuits Devices Syst.,* 1998, **145**, (4), pp. 247–253
12 BURFORD, R.G., FAN, X., and BERGMANN, N.W.: 'An 180 MHz 16-bit multiplier using asynchronous logic design techniques'. Proceedings of IEEE 1994 Custom Integrated Circuits confereuce, 1994, pp. 215–218
13 KEARNY, D.A., and BERMANN, N.W.: 'VLSI design of an asynchronous multiplier with data dependent processing times'. Proceedings of 14th Australian Microelectronics Conference, IEEE Society, 1997, pp. 282–287
14 CHAING, J.S., and LIAO, J.Y.: 'A novel asynchronous control unit and the application to a pipelined multipler'. Proceedings of international symposium on Circuits and Systems (ISCAS), June 1998, Vol. 2, pp. 169–172
15 SPARSO, J., NIELSEN, C.D., NIELSEN, L.S., and STAUNSTRUP, J.: 'Design of self-timed multipliers: A comparation'. Proceedings of IFIP WG 10.5 Working Conference on Asynchronous Design Methodologies, Manchester, UK, 1993, pp. 165–180
16 CHANDRAMOULI, V., BRUNVAND, E., and SMITH, K.F.: 'Self-timed design in GaAs–case study of a high-speed, parallel multiplier', *IEEE Trans. VLSI Syst.,* 1996, **4**, (1), pp. 146–149
17 PANG, T.C.J., CHOY, C.S., CHAN, C.F., and CHAM, W.K.: 'Self-timed Booth's multiplier'. Proceedings of 2nd International Conference on ASIC, Shangai, China, October 1996, pp. 21–24
18 REITWIESNER, G.W.: 'Binary arithmetic', *Advances in Computers,* 1960, **1**, pp. 231–308
19 McSORLEY, O.L.: 'High Speed Arithmetic in Binary Computers', *Proceedings of the IRE,* 1961, **49**, (1), pp. 67–91
20 JOHNSON, D., and AKELLA, D.V.: 'Design and analysis of asynchronous adders', *IEE Proc., Comput. Digit. Tech.,* 1998, **145**, (1), pp. 1–8
21 ANNARATONE, M.: 'Digital CMOS Circuit Design' (Kluwer Academic Publisers, Boston, 1986)
22 RUIZ, G.A.: 'Evaluation of three 32-bit CMOS Adders in DCVS logic for self-timed circuits', *IEEE J. Solid-State Circuits,* 1998, **33**, (4), pp. 604–613
23 SCHUMANN, T., KLAR, H., and HOROWITZ, M.: 'A fully asynchronous, high-throughput multiplier using wave-pipelining for DSP custom applications'. Mikroelektronik für die Informationstechnik, ITG-Fachtagung, Chemnitz, Germany, 1996, pp. 283–286
24 NAGAMATSU, M., TANAKA, S., MORI, J., HIRANO, K., NOGUCHI, T., and HATANAKA, K.: 'A 15 ns 32 × 32-b CMOS multiplier with an improved parallel structure', *IEEE J. Solid-State Circuits,* 1990, **25**, (2), pp. 494–497
25 GOTO, G., SATO, T., NAKAJIMA, M., and SUKEMURA, T.: 'A 54 × 54-b regularly structured tree multiplier', *IEEE J. Solid-State Circuits,* 1992, **27**, (9), pp. 1229–1235
26 MORI, J., NAGAMATSU, M., HIRANO, M., TANAKA, S., NODA, M., TOYOSHIMA, Y., HASHIMOTO, K., HAYASHIDA, H., and MAEGUCHI, K.: 'A 10 ns 54 × 54-b parallel structured full array multiplier with 0.5-μm CMOS technology', *IEEE J. Solid-State Circuits,* 1991, **26**, (4), pp. 600–605