Research Article

A General Design Methodology for Synchronous Early-Completion-Prediction Adders in Nano-CMOS DSP Architectures

Mauro Olivieri and Antonio Mastrandrea

*Department of Information Engineering, Electronics and Telecommunications, Sapienza University of Rome,
Via Eudossiana 18, 00184 Rome, Italy*

Correspondence should be addressed to Mauro Olivieri; olivieri@diet.uniroma1.it

Received 12 September 2012; Revised 2 December 2012; Accepted 5 December 2012

Academic Editor: Meng-Hsueh Chiang

Copyright © 2013 M. Olivieri and A. Mastrandrea. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Synchronous early-completion-prediction adders (ECPAs) are used for high clock rate and high-precision DSP datapaths, as they allow a dominant amount of single-cycle operations even if the worst-case carry propagation delay is longer than the clock period. Previous works have also demonstrated ECPA advantages for average leakage reduction and NBTI effects reduction in nanoscale CMOS technologies. This paper illustrates a general systematic methodology to design ECPA units, targeting nanoscale CMOS technologies, which is not available in the current literature yet. The method is fully compatible with standard VLSI macrocell design tools and standard adder structures and includes automatic definition of critical test patterns for postlayout verification. A design example is included, reporting speed and power data superior to previous works.

1. Introduction

Fast integer adders are an essential component of most DSP datapaths. Synchronous early-completion-prediction adders (ECPAs) [1], also known as variable-latency adders [2], have been introduced for high clock rate and high-precision datapaths, as they allow single-cycle operations even if the clock period is shorter than the worst-case carry propagation delay. Thanks to the data dependency of actual carry chain propagation, the occurrence of multicycle operations can be maintained statistically rare, thus allowing an overall speed improvement. The industrial effectiveness of the idea was first proven by the design of a full-custom ECPA unit for a DSP datapath at Toshiba Labs [1]. The logic foundation of that adder is shown in [3]. An extension to multiply unit design has been shown in [4]. The works in [2] and [5] have recently pointed out the potentials of variable-latency adder units in nano-CMOS addition units, for reducing average leakage power consumption and improving robustness to NBTI faults occurring in nano-scale technologies.

An ECPA consists of a conventional adder plus a completion-prediction logic unit (Figure 1). The prediction unit estimates the actual critical path length in the adder depending on the operand values and hence the cycle count of the operation for the target cycle time. This approach differs from asynchronous completion detection units [6–8], as it is based on a totally synchronous scheme. From the design point of view, the logic specification of the prediction function depends on the target cycle time and on the estimation of the variable completion time of the adder, in order to define the cycle count output. Moreover, the speed of the prediction unit is critical, since the prediction must always be completed in a single cycle in order to be effective.

No general design methodology for ECPA VLSI cores has been proposed yet. In [3], Lee and Asada analyzed the design problem on the basis of 2-input-gate unit delay within a ripple carry adder structure. In [1], Kondo et al. address the full-custom design case of a fast carry-select structure. In [9], Nowick et al. deal with the design of speculative-completion adders, similar in principle to ECPA but again

addressing asynchronous design. In [2], a 64-bit carry-select design case is presented, where the prediction logic synthesis *a priori* assumes that single-cycle latency occurs when the carry propagation chain is shorter than 32 bits.

We present a detailed, general method for the design of completion-prediction logic in full-custom adder macrocells in nano-scale CMOS technologies, targeting carry-select and hybrid carry-select/carry-lookahead addition schemes. Notably, the prediction logic unit insertion does not modify the adder logic in any way. The proposed method supports the prediction of generally any cycle count latency value and not only single-cycle and two-cycle latency.

The paper illustrates the reference architecture template, the adder propagation delay model upon which the design method is built, the design procedure detailed description, the general validation of the approach, and a specific example with performance results referring to 32 ns CMOS technology and reporting speed and power data.

2. Architecture Template

We start from the well-known carry-select addition scheme [10–12], illustrated in Figure 2. The logic components indicated with “setup” produce the *propagate* and *generate* bits out of the operand bits. The components indicated with “adder block” produce potential carry and sum values to be selected by the chain of 2:1 multiplexers in the lower part of the picture. The proposed methods targets two standard addition schemes:

- (i) a conventional, low area carry-select with ripple adder circuitry in its adder blocks (in the following, CSA);
- (ii) a hybrid scheme of carry-select with carry-lookahead circuitry in its adder blocks (in the following, CLA/CSA).

In both schemes, we refer to the operand size as N bits and the block size as m bits.

A design option is the introduction of internal pipeline stages in the adder. As an ECPA unit has the goal of statistically exploit single-cycle operation latency, the introduction of pipelined operations may result in relatively low effectiveness at the expense of a pipeline register delay overhead. Its effectiveness was studied by instruction level simulations of SPECint benchmarks in [8]. For a 2-cycle worst-case latency ECPA sustaining 1 GHz clock rate, only 6% of the additions resulted to take benefit from the internal pipeline. Here we illustrate the design of nonpipelined ECPA specifically addressing single-cycle latency maximization; specific applications may benefit from the introduction of a pipeline in the architecture.

The clocking strategy underlying the proposed high-speed ECPA design is a two-phase symmetric clock with dynamic logic and transparent latches [11, 12]. Figure 3 illustrates the architecture and timing of the operations. The set-up logic and the adder block logic are implemented

FIGURE 1: General scheme of an ECPA unit.

in Domino style, precharged on the low clock half-cycle. The selection chain multiplexers are implemented in static logic, while the prediction logic is implemented in Domino style, precharged in the high clock half-cycle. The input register of the whole adder is split into two latches, one between the adder block and the selection chain and one after the selection chain. The whole adder structure is developed adopting a standard VLSI macrocell design tool chain.

In case of single cycle operation, the adder operates as a normal single-cycle Domino circuit. In case of multicycle operation, the input of the adder blocks should not change for the time needed to complete the selection chain and the sum generation. This is normally accomplished by input registers of the arithmetic unit in the datapath architecture. In general, depending on the operand values and on the clock cycle time, an addition can take $1 + X$ cycles, where X ranges from 0 (single cycle operation) to some units (usually not more than 3 for practical interest design cases).

3. Timing Analysis and Model of the Variable Completion Time

In the analyzed adder schemes, the addition operation proceeds as follows: after the propagate and generate bit vectors have been set up, any block having at least one propagate bit at “0” produces a valid anticipated output carry bit independent from its input carry bit. Then, the output carry bits of the remaining blocks are evaluated by means of the carry selection chain. The longest chain of unknown carry bits determines the time duration of the latter operation [7, 8, 14, 15]. When all the carries are ready, the adder performs the sum selection to produce the result.

Referring to Figure 3, the delays of the components involved in the above operations are the set-up logic delay, the adder block delay (including a latch delay), the multiplexer delay, and the latch delay, namely, δ_{SETUP} , δ_{BLOCK} , δ_{MUX} , and δ_{LATCH} . Finally, let us refer to the length of the longest chain of consecutive unknown carries as $\max L$, an integer ranging from 0 to $N/m - 1$, and to the prediction unit propagation delay as δ_{PRED} . Given a clock cycle time T_{CK} , the timing conditions for correctly performing an addition in $1 + X$ clock cycles are as follows.

FIGURE 2: Generic scheme of carry-select addition.

FIGURE 3: ECPA architecture template and timing diagram of the adder, for a single-cycle and a two-cycle operations.

FIGURE 4: Generic structure and timing diagram of the prediction logic for a single-cycle and a two-cycle operation.

FIGURE 5: Critical circuit path in the prediction logic implementation.

Precharge phase conditions:

$$\begin{aligned} \delta_{\text{SETUP}}|_{\text{precharge}} &< \frac{T_{CK}}{2}, \\ \delta_{\text{BLOCK}}|_{\text{precharge}} &< \frac{T_{CK}}{2}, \\ \delta_{\text{PRED}}|_{\text{precharge}} &< \frac{T_{CK}}{2}. \end{aligned} \quad (1)$$

FIGURE 6: State diagram of a synchronous FSM generating the wait signal.

Evaluation phase conditions:

$$\delta_{\text{PRED}} < \frac{T_{CK}}{2}, \quad (2)$$

$$\delta_{\text{SETUP}} + \delta_{\text{BLOCK}} < \frac{T_{CK}}{2}, \quad (3)$$

$$\begin{aligned} \delta_{\text{SETUP}} + \delta_{\text{BLOCK}} + \delta_{\text{MUX}} + \delta_{\text{MUX}} \cdot \max L + \delta_{\text{MUX}} + \delta_{\text{LATCH}} \\ < (1 + X) \cdot T_{CK}. \end{aligned} \quad (4)$$

In (4), the term $\delta_{\text{BLOCK}} + \delta_{\text{MUX}}$ is the time needed to produce the anticipated carries [7, 15], the term $\delta_{\text{MUX}} \cdot \max L$ is the delay of the longest carry selection chain, and the last δ_{MUX} is the delay of the 2:1 final sum selection after all the carries are ready (minor conservative approximation contained in (4)—mainly in the time for producing the anticipated carries and in the individual role of the least/most significant blocks—slightly overestimates the addition time).

Equation (1) must be verified after the circuit implementation but do usually not represent a problem as the evaluation pull-down, performed sequentially, is normally slower than the pull-up precharge, performed in parallel [16]. Equations (2) and (3) are necessary conditions for the correct implementation of the ECPA and constitute a circuit design constraint. Equation (4) constitutes the functional specification of the prediction logic, as the smallest integer X satisfying it defines the predicted cycle count $1 + X$. The worst-case cycle count occurs if $\max L = N/m - 1$.

FIGURE 7: Circuit implementation of the prediction logic example.

We characterized the propagation delay of the CMOS circuits in the adder structure by means of NGSPICE simulation [17], referring to a 32 nm CMOS technology. Transistor sizing was optimized for critical path speed according to the logical effort method [16]. Table 1 summarizes the resulting component delays for the two adder schemes. In order for the reported characterization to be used in different technologies, the delay values are normalized with respect to the reference time unit FO4, given by the delay of an inverter driving four identical inverters, which is a characteristic datum for a given technology. Note that δ_{LATCH} results to be always 1.7 FO4 delay units, as we use a fixed latch cell with fixed load.

4. Synthesis Method of the Completion Prediction Unit

According to (4), the prediction circuit must evaluate $\max L$, convert this information into the binary cycle count X of the addition, and/or activate a synchronous completion signal at the X th cycle after the addition has started. Figure 4 shows the generic architecture of the prediction logic, composed of three stages.

(1) The first stage computes the dependency bit P_i of each block (except for block 0); that is

$$P_i = \overline{cL_i} \wedge cH_i, \quad i = 1, \dots, \frac{N}{m} - 1, \quad (5)$$

TABLE 1: Delay estimates for different adder schemes, assuming N bits operands and m bits blocks. "Total delay" refers to the worst-case completion time.

N	m	4	5	6	7	8	9	10	11	12
CSA	Total delay	31.7	29.3	28.1	27.4	27.2	28.0	29.1	29.4	30.8
	δ_{SETUP}	3.7	3.6	3.5	3.4	3.4	3.3	3.2	3.2	3.2
	δ_{BLOCK}	9.2	11.0	12.8	14.7	16.5	18.2	19.8	21.6	23.4
	δ_{MUX}	2.7	2.6	2.6	2.5	2.4	2.4	2.4	2.4	2.4
	Total delay	57.8	50.6	46.1	43.6	41.2	41.2	40.2	39.8	40.5
	δ_{SETUP}	4.0	4.0	3.9	3.8	3.7	3.6	3.6	3.5	3.4
64	δ_{BLOCK}	10.1	12.4	14.5	16.4	18.5	21.1	21.9	23.8	25.4
	δ_{MUX}	3.0	2.9	2.8	2.7	2.7	2.6	2.6	2.6	2.5
	Total delay	29.3	26.0	23.9	22.6	21.8	22.0	22.5	22.8	23.8
CLA/CSA	δ_{SETUP}	4.2	4.1	4.1	4.2	4.3	4.2	4.0	4.2	4.1
	δ_{BLOCK}	4.2	5.0	5.9	7.0	8.3	9.5	10.9	12.6	14.3
	δ_{MUX}	3.0	3.0	3.0	3.0	3.1	3.0	2.9	3.0	3.0
	Total delay	56.7	49.0	44.1	41.0	38.2	37.5	36.7	36.4	36.7
	δ_{SETUP}	4.4	4.5	4.6	4.6	4.8	4.7	4.8	5.0	4.8
	δ_{BLOCK}	4.4	5.2	6.2	7.3	8.6	9.9	11.4	13.1	14.8
64	δ_{MUX}	3.2	3.2	3.3	3.2	3.4	3.4	3.4	3.5	3.4

FIGURE 8: Layout of the adder macrocell design example.

where cL_i and cH_i are the potential carry values produced in block i by the two adders with input carry = 0 and input carry = 1, respectively. The output carry of a block depends on the preceding block if and only if P_i is “1,” provided that cL_i and cH_i have reached a stable state [7].

(2) The second stage evaluates $\max L$ and encode its value by $N/m - 1$ signals named L_k , such that $L_k = “1”$ means $\max L \geq k$.

To synthesize L_1 , we have to find out any block having P_i active; for L_2 we have to find out any two consecutive blocks i and $i + 1$ having both P_i and P_{i+1} active; and so on. As a general expression, we have

$$L_k = (P_1 \wedge \dots \wedge P_k) \vee (P_2 \wedge \dots \wedge P_{k+1}) \vee \dots \vee (P_{N/m-k} \wedge \dots \wedge P_{N/m-1}). \quad (6)$$

Referring to a target cycle time T_{CK} , from (4) in conjunction with the delay values in Table 1, we can evaluate the cycle count X for all the possible values of $\max L$ and hence for all the combinations of the logic variables L_k . We can build up a prediction table for each adder type and size, defining the correct values of X for a set of values of the target cycle time T_{CK} . Table 2 shows an example of two prediction tables, for the CSA and the CLA/CSA, respectively. The shaded part covers those T_{CK} values that do not match (3). Once we choose the column corresponding to the target cycle, we can look at the prediction table as a truth table with logic signals L_k as input variables and the cycle count X (e.g., its binary expression) as output variable. The adjacent rows having the same output cycle count correspond to two logic minterms differing only in one input variable L_k , occurring as direct and complemented. Such adjacent rows can, therefore, collapse into a single row by Boolean reduction, resulting in a final truth table dedicated to the target adder and target clock period, where only a subset of *effective* L_k variables appears, drastically reducing the hardware overhead in both the second and third stage of the prediction logic. Section 6 presents an example of the truth table generation.

(3) The third stage converts the logic signals L_k into a binary expression of X . The binary digits x_j of the number X can be explicitly specified in the truth table and synthesized as a (very simple) combinational function of the L_k variables. If the datapath architecture does not require a binary expression of the cycle count X but rather a synchronous wait signal to flag the completion of the operation, the third stage can be a (very simple) synchronous state machine, directly driven by the logic signals L_k , activating a synchronous completion signal X cycles after the current one. An example of both solutions is shown in Section 6.

The fast full-custom circuit implementation of the first two logic stages relies on dynamic Domino circuits precharged on the low clock half-cycle, according to the two-phase clocking strategy sketched in Figure 2. The third stage, thanks to its inherently low complexity, can be implemented as static logic without compromising speed. The schematic of the critical path of the prediction logic is sketched in Figure 5: the first and second stages, implementing (5) and (6), consist of single-stage Domino circuits. All of the signals coming out of the Domino gates are precharged low during the low clock phase. The external logic is supposed to sample the output of the prediction unit on the clock falling edge.

5. Design Validation Procedure

The validation of the prediction logic design must address two issues: verifying the predicted cycle count correctness and verifying that the prediction unit evaluates the cycle count X within the second half-cycle of the current clock cycle.

To verify the correct prediction in a postlayout SPICE simulation, we can automatically generate the critical test patterns corresponding to the boundaries between different cycle count predictions shown by the rows of the truth table. In each row, only one L_k variable occurs with explicit value “0”; the corresponding critical addition operands are all the input patterns that set a string of $k-1$ consecutive dependency bits P_i high. In formulas, such critical operand values A and B can be defined as follows, for each truth table row, in which $L_k = 1$:

$$A = 2^N - 1, \quad B = \sum_{i=0}^{N/m-1} 2^{i \cdot m} - \sum_{i=j}^{j+k-1} 2^{i \cdot m}, \quad (7)$$

$$\forall j = 0, \dots, \frac{N}{m} - k.$$

A special test case is the prediction of the worst-case cycle count, which corresponds to operands $A = 2^N - 1$ and $B = 1$.

To test the prediction unit critical path delay in a post-layout SPICE-level simulation, the adder input to be used is the same as for the worst-case cycle count prediction; that is, $A = 2^N - 1$, and $B = 1$.

6. Example of ECPA Unit Design

6.1. Summary of the Design Flow. Designing an ECPA with the proposed approach is an eight-step process. The first

TABLE 2: Prediction table for the cycle count in two different ECPA schemes with $N = 32$, $m = 4$.

L_k variables										Cycle time in FO4 units													
L_1	L_2	L_3	L_4	L_5	L_6	L_7	16	16.5	17	17.5	18	18.5	19	19.5	20	20.5	21	21.5	22	22.5	23	23.5	24
0	—	—	—	—	—	—	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1
1	0	—	—	—	—	—	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1
—	1	0	—	—	—	—	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
—	—	1	0	—	—	—	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
—	—	—	1	0	—	—	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
—	—	—	—	1	0	—	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
—	—	—	—	—	1	0	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2
—	—	—	—	—	—	1	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2
—	—	—	—	—	—	—	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2
0	—	—	—	—	—	—	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	—	—	—	—	—	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1
—	1	0	—	—	—	—	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1
—	—	1	0	—	—	—	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
—	—	—	1	0	—	—	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
—	—	—	—	1	0	—	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
—	—	—	—	—	1	0	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
—	—	—	—	—	—	1	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2
—	—	—	—	—	—	—	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2

CLA/CSA

TABLE 3: Truth table for the prediction unit of a 32-bit, 4-bit clock, CLA/CSA ECPA for a cycle time equivalent to 25.5 FO4 delay units.

L_1	L_6	Predicted X	Binary output X
0	—	0	0 0
1	0	1	0 1
—	1	2	1 0

TABLE 4: Statistical performance data (average speedup with respect to fixed latency implementation).

Design	Random operands	Real operands
[1]	n.a.	1.79
[9]	1.19	1.01
[13]	n.a.	1.78
This work	2.13	2.43

two are performed off-line and their result can be reused in different ECPA designs.

Offline Steps

- (1) delay table compilation through SPICE characterization,
- (2) prediction table compilation through delay tables, (3), and (4).

Actual Design Steps

- (3) adder architecture configuration and selection of the prediction table,
- (4) column selection in the prediction table and truth table generation,
- (5) generation of the test addition set through (7) and truth table,
- (6) synthesis and circuit design of the prediction logic three stages (from (5), (6), and Truth Table),
- (7) adder circuit design through standard full-custom VLSI design tools,
- (8) postlayout design validation through SPICE simulation of the prediction logic critical path and of the test additions set produced at step (5).

If postlayout simulation reports a prediction fault (i.e., predicted cycle count lower than real latency), one may permanently update the prediction table accordingly and repeat the synthesis process. All of the steps can be implemented in a software ECPA macrocell generator.

6.2. Design Example. We show the case of a prediction unit designed for a 32-bit operand, 4-bit block size CLA/CSA ECPA, considering a 32 nm metal-gate high-K CMOS process characterized by an FO4 propagation delay of 8.7 ps. We target a clock frequency of 6 GHz, that is, 166.4 ps cycle time, equivalent to 18.5 FO4 delay units. Such clock speed is extremely high for the reference CMOS process.

From Table 2, selecting the cycle time column labeled with number 25.5, we have 3 possible values for the cycle

count, that is, 1, 2, and 3. Hence the prediction table will collapse into a 3 row truth table, shown by Table 3. Consequently, the logic functions to be implemented are

$$\begin{aligned}
 P_i &= \overline{cL_i} \wedge cH_i, \quad i = 1, \dots, 7, \\
 L_1 &= (P_1 \wedge P_2 \wedge P_3 \wedge P_4 \wedge P_5 \wedge P_6 \wedge P_7), \\
 L_6 &= (P_1 \wedge P_2 \wedge P_3 \wedge P_4 \wedge P_5 \wedge P_6) \\
 &\quad \wedge (P_2 \wedge P_3 \wedge P_4 \wedge P_5 \wedge P_6 \wedge P_7).
 \end{aligned} \tag{8}$$

The logic synthesis of the cycle count X obtained from Table 3 is the 2-bit expression $x_1 = L_6$, $x_0 = \overline{L_6} \wedge L_1$. As an alternative, the synthesis of the wait signal can be obtained by the state machine specified in Figure 6.

Figure 7 shows the transistor level design of the whole prediction circuitry resulting from applying the synthesis procedure. The transistor sizing in the prediction unit as well as in the adder is optimized according to the logical effort method [16].

The prediction logic transistor count is 181, including the static logic for encoding the binary digits of number X and the state machine to produce the *wait* signal (in practice they are mutually exclusive solutions).

The critical test cases resulting from (7) in conjunction with Table 3 are additions with the following values of operands A and B .

- (i) Test pattern for predicted latency of 1 cycle ($X = 0$):

$$\begin{aligned}
 A &= 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111; \\
 B &= 0001\ 0001\ 0001\ 0001\ 0001\ 0001\ 0001\ 0001;
 \end{aligned}$$

- (ii) Test patterns for predicted latency of 2 cycle ($X = 1$):

$$\begin{aligned}
 A &= 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111; \\
 B &= 0001\ 0001\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000; \\
 A &= 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111; \\
 B &= 0001\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001; \\
 A &= 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111; \\
 B &= 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001\ 0001;
 \end{aligned}$$

- (iii) Test pattern for predicted latency of 3 cycle ($X = 2$, worst case):

$$\begin{aligned}
 A &= 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111; \\
 B &= 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001.
 \end{aligned}$$

The total hardware complexity of this ECPA unit is 1705 transistors, resulting from the prediction circuitry (181 transistors) plus the hybrid CLA/CSA adder (1524 transistors). Figure 8 shows the layout of the adder macrocell, total size being $6.6\ \mu\text{m} \times 5.8\ \mu\text{m}$.

6.3. Speed Performance Results. We tested the critical paths of the resulting circuits by NGSPICE simulation, verifying that all of the tests give correct prediction. An additional verification was performed at the logic level on the full

TABLE 5: Power saving over fixed-latency implementation having the same operations/second throughput.

Design	V_{DD} value for wanted throughput	Dynamic power saving	Leakage power saving	Total power saving
[2]	0.77 V	36%	77%	44%
This work	0.60 V	78%	89%	81%

architecture of the adder, by means of a nano-CMOS dedicated delay model [18], confirming the positive results of the test. Finally, NGSPICE BSIM4 simulation of the critical path of the prediction unit confirms that the evaluation of the predicted cycle time is always within one clock cycle. Figure 9 shows the NGSPICE output of the latter simulated test, referring to the worst-case cycle count; that is, $X = 2$, at 6 GHz clock frequency. The x_1 and x_0 bits correctly take the “10” value on the falling clock edge after the presentation of the addition operands, and the wait signal correctly remains high for two consecutive falling clock edges. The critical path of the prediction logic goes from the cH_1 signal (potential carry bit of carry-select block coming out of slave latch in Figure 3) to the wait signal and results to have a slack time of over 30% of the clock period.

The statistical speed performance of the proposed example is shown in Table 4, compared with previous early-completion-prediction designs for which performance data are available in the literature. The works in [1] refer to a variable-latency 32-bit carry-select ALU, in [13] refers to variable-latency 32-bit Brent-Kung adder implementation, while the work in [9] refers to an asynchronous early-completion 32-bit Brent-Kung adder. In Table 4 the speed-up value refers to the average improvement with respect to a fixed-latency synchronous implementation of the same adder design. In the proposed approach, a fixed-latency implementation is simply obtained by eliminating the completion prediction logic, with no modification of the conventional adder structure. Performance results are reported referring to random uniformly distributed input operands and to real operands obtained by execution traces of SPECint benchmark suite, except for [9] for which the published performance refers to Dhrystone and Espresso benchmarks. The reported performance values are directly obtained from the results claimed in [1, 9, 13]. In all cases, the proposed adder outperforms the speedup attained by previously published works.

6.4. Power Saving and NBTI Mitigation Estimation. The average power consumption of the designed unit simulated at 1 V power supply with random inputs at 6 GHz is 0.148 mW, subdivided in 0.127 mW dynamic and 0.021 mW leakage power consumption.

The statistical speed advantage over a fixed-latency implementation can be effectively used for reducing power consumption at the same operations/second throughput, by means of V_{DD} reduction and clock period relaxation [2]. In the proposed design example, the supply voltage to obtain the same average throughput as a fixed latency implementation

is $V_{DD} = 0.6$ V with relaxed clock period of 330 ps. Table 5 shows the resulting energy saving with respect to the fixed latency design. The normalized results are compared with the energy saving attained by the variable-latency carry-select design in 70 nm CMOS described in [2], for which power saving data are available in the literature and the supply voltage compatible with the same throughput as the fixed latency implementation is reported 0.77 V.

According to the characterization of NBTI reported in [19], the shift in the PMOS threshold voltage V_{th} caused by NBTI is strongly dependent on the V_{DD} value. As a result, the same mechanism applied for power consumption reduction can be applied to NBTI mitigation. In the proposed design example, the statistical performance advantage allows a 40% reduction of V_{DD} (from 1.0 V to 0.6 V) which results in 35% reduction in V_{th} shift in one year of circuit operation, according to [19]. As a supplementary countermeasure, like any variable-latency unit, the proposed design can be equipped with the insertion of guard band violation sensors [2] in order to detect the NBTI effect on completion time and adjust predicted latency, as shown in the variable-latency adder presented in [2].

7. Conclusions

A general methodology has been presented, for synthesizing the prediction logic of early-completion-predicting adders (ECPA), also known as variable-latency (VL) adders, which have been proposed for reducing leakage and NBTI failures in high-speed DSP datapath to be realized in nano-CMOS technology.

While previous works present specific design cases, a general method for prediction logic synthesis is not available in the literature. The proposed method utilize the well-known high-speed carry-select and hybrid carry-select/carry-lookahead as reference addition schemes, and the prediction logic does not affect the adder logic design in any way. The design method is implemented through a standard VLSI custom macrocell design tool chain. Finally, the methodology includes an automatic way to generate critical test patterns for the ECPA postlayout validation.

The resulting ECPA circuit complexity is competitive with conventional high-speed adders, as the hardware overhead is only 10% of the adder logic. A design case in 32 nm CMOS technology, simulated at postlayout SPICE BSIM4 level, results in sustaining a 6 GHz clock frequency with correct cycle time predictions. Results on statistical speed performance advantage, power consumption reduction, and NBTI mitigation have been obtained with respect to a fixed latency implementation of the same adder architecture.

FIGURE 9: SPICE simulation of the prediction logic example.

References

- [1] Y. Kondo, N. Ikumi, K. Ueno, J. Mori, and M. Hirano, "Early-completion-detecting ALU for a 1 GHz 64 b datapath," in *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC '97)*, pp. 418–419, February 1997.
- [2] Y. Chen, H. Li, C. K. Koh et al., "Variable-latency adder (VL-adder) designs for low power and NBTI tolerance," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 11, pp. 1621–1624, 2010.
- [3] J. Lee and K. Asada, "A synchronous completion prediction adder (SCPA)," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E80-A, no. 3, pp. 606–609, 1997.
- [4] M. Olivieri, "Design of synchronous and asynchronous variable-latency pipelined multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 2, pp. 365–376, 2001.
- [5] Y. Chen, H. Li, J. Li, and C. K. Koh, "Variable-latency adder (VL-adder): new arithmetic circuit design practice to overcome

- NBTI,” in *International Symposium on Low Power Electronics and Design (ISLPED '07)*, pp. 195–200, Portland, Ore, USA, August 2007.
- [6] A. De Gloria and M. Olivieri, “Statistical Carry Lookahead adders,” *IEEE Transactions on Computers*, vol. 45, no. 3, pp. 340–347, 1996.
 - [7] A. D. Gloria, “Completion-detecting carry select addition,” *IEE Proceedings: Computers and Digital Techniques*, vol. 147, no. 2, pp. 93–100, 2000.
 - [8] D. J. Kinniment, “An evaluation of asynchronous addition,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 4, no. 1, pp. 137–140, 1996.
 - [9] S. M. Nowick, K. Y. Yun, P. A. Beerel, and A. E. Dooply, “Speculative completion for the design of high-performance asynchronous dynamic adders,” in *Proceedings of the 3rd International Symposium on Advanced Research Asynchronous Circuits and Systems*, pp. 210–223, Eindhoven, The Netherlands, April 1997.
 - [10] O. J. Bedrij, “Carry-select adder,” *IRE Transactions on Electronic Computers*, vol. 11, pp. 340–346, 1962.
 - [11] J. M. Rabaey, *Digital Integrated Circuits: A Design Perspective*, Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.
 - [12] N. H. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Addison Wesley, Reading, Mass, USA, 2nd edition, 1994.
 - [13] D. Koes, T. Chelcea, C. Onyeama, and S. Goldstein, “Adding faster with application specific early termination,” Technical Report No. CMU-CS-05-101, Carnegie Mellon University, May 2005, <http://www.cs.cmu.edu/~seth/papers/koes-tr05.html>.
 - [14] B. E. Briley, “Some new results on average worst case carry,” *IEEE Transactions on Computers*, vol. 22, no. 5, pp. 459–463, 1973.
 - [15] G. W. Reitwiesner, “The determination of carry propagation length for binary addition,” *IRE Transactions on Electronic Computers*, vol. 9, no. 1, pp. 35–38, 1960.
 - [16] I. Sutherland, B. Sproull, and D. Harris, *Logical Effort: Designing Fast CMOS Circuits*, Morgan-Kaufmann, San Francisco, Calif, USA, 1999.
 - [17] F. Lannutti, P. Nenzi, and M. Olivieri, “KLU sparse direct linear solver implementation into NGSPICE,” in *Proceedings of the 19th International Conference on Mixed Design of Integrated Circuits and Systems (MIXDES '12)*, p. 69, 2012.
 - [18] A. Mastrandrea, M. Olivieri, and F. Menichelli, “A delay model allowing nano-CMOS standard cells statistical simulation at the logic level,” in *Proceedings of the 7th IEEE Conference on Ph.D. Research in Microelectronics and Electronics (PRIME '11)*, pp. 217–220, Trento, Italy, July 2011.
 - [19] R. Vattikonda, W. Wang, and Y. Cao, “Modeling and minimization of PMOS NBTI effect for robust nanometer design,” in *Proceedings of the 43rd Annual Design Automation Conference (DAC '06)*, pp. 1047–1052, ACM, San Francisco, Calif, USA, 2006.