FACTA UNIVERSITATIS
Series: **Electronics and Energetics** Vol. 28, N^o 4, December 2015, pp. 657 - 671
DOI: 10.2298/FUEE1504657B

A LATENCY OPTIMIZED BIASED IMPLEMENTATION STYLE WEAK-INDICATION SELF-TIMED FULL ADDER

Padmanabhan Balasubramanian

School of Computer Engineering, Nanyang Technological University, Singapore

Abstract. *This article presents a biased implementation style weak-indication self-timed full adder design that is latency optimized. The proposed full adder is constructed using the delay-insensitive dual-rail code and adheres to the 4-phase handshaking. Performance comparisons of the proposed full adder vis-à-vis other strong and weak-indication full adders are done on the basis of a 32-bit self-timed ripple carry adder architecture, with the full adders and ripple carry adders realized using a 32/28nm CMOS process. The results show that the proposed full adder leads to reduction in latency by 63.3% against the best of the strong-indication full adders whilst reporting decrease in area by 10.6% and featuring comparable power dissipation. On the other hand, when compared with the existing optimized weak-indication full adder, the proposed full adder is found to minimize the latency by 25.1% whilst causing an increase in area by just 1.6%, however, with no associated power penalty.*

Key words: *self-timed design, full adder, RCA, indication, standard cells, CMOS*

1. INTRODUCTION

Self-timed design, which constitutes a robust flavor of asynchronous design, is considered to be a viable alternative and/or a necessary supplement to mainstream synchronous design by the Semiconductor Industry Association [1] due to several reliability and variability issues, which have become prominent in the nanoscale electronics regime. Random dopant fluctuations, sub-wavelength lithography, high heat flux, electro-migration, hot carrier effects, negative bias temperature instability, stress-induced variation, electrostatic discharge, process-induced defects, and metrology and other manufacturing defects [2] are complicated issues which have become more pronounced in the nanoelectronics are compared to the microelectronics era and are indeed difficult to deal with. To circumvent these issues, various material-level, device-level, process-level, circuit-level and system-level solutions have been developed and further developments are also underway [1].

Received April 06, 2015; received in revised form June 08, 2015
Corresponding author: Padmanabhan Balasubramanian
School of Computer Engineering, Nanyang Technological University, 50 Nanyang Avenue, Singapore 639798
(e-mail: balasubramanian@ntu.edu.sg)

At the circuit/system-level, self-timed design has been drawing sustained interest from the research community over the past decades due to several inherent advantages such as low noise [3] and almost nil electro-magnetic interference (EMI) [4], greater modularity [5], ability to cope with process, temperature, and parametric variations with ease thus being inherently adaptive [6] [7], consuming power only when and where active [5] [8], and being self-checking [9]. Low noise and EMI compatibility imply that self-timed circuits are innately resistant to side channel attacks [10] [11] and are therefore preferable for secure banking and financial and other sensitive applications. Modularity, also known as design reusability, and the capacity to tolerate process, temperature and parametric uncertainties imply that self-timed circuits are well positioned to deal with statistical timing analysis and reliability issues whilst delivering an average case performance. Due to the consumption of power only on-demand, depending on when and where required, self-timed circuits/systems form a natural choice for ultra low power VLSI designs where complimentary design strategies such as multiple supplies, multiple thresholds, and dynamic voltage and/or frequency scaling may be deployed to leverage the maximum benefits from a self-timed design. Being self-checking, self-timed circuits/systems conform to the design-for-testability paradigm although complexities may be involved in the testability aspect of certain asynchronous elements, for example the C-element which incorporates feedback; nevertheless some feasible approaches are reported [12] [13].

Asynchronous designs are primarily classified as bundled-mode and input-output mode, and here, we only consider the input-output mode, which is the robust among the two as it employs unbounded delay models for components (gates) and/or interconnect [14]. Asynchronous circuits/systems corresponding to input-output mode are commonly referred to as *self-timed circuits/systems* [15]. The fundamental architecture of a self-timed system is shown in Figure 1, which has a centrally located *function block*. The function block in a self-timed system is equivalent to the combinational logic of a synchronous system, with the exceptions of being realized using delay-insensitive codes, and being bestowed with the responsibility of not only having to produce the correct outputs subject to the applied inputs but also should signal the completion of internal data computation. Thus the function block forms the heart of a self-timed system that performs data processing. In this article, the term ‘function block’ may refer to an arithmetic element, say the full adder, or a sub-system, for example, a ripple carry adder (RCA). The self-timed system, portrayed in Figure 1, utilizes delay-insensitive codes (here, dual-rail code) for data representation, communication and processing, and the 4-phase return-to-zero (RTZ) handshaking. The dual-rail code is the simplest member of the generic family of delay-insensitive *m*-of-*n* codes [16], where *m* wires are asserted ‘high’ (i.e. binary 1) out of a total of *n* wires to represent binary data. In a dual-rail code, a data wire D is encoded using two wires viz. D0 and D1, where D = 1 is represented by D1 = 1 and D0 = 0, and D = 0 is represented by D0 = 1 and D1 = 0. When D1 and D0 signify a binary value of 0 or 1 according to the assignments mentioned, it is called ‘valid data’. The state of both D0 and D1 being equal to 0 is referred to as the ‘spacer’. It may be noted that both D0 and D1 cannot simultaneously transition to 1 as it is illegal and invalid since the coding scheme adopted is unordered [17], where no codeword should form a subset of another codeword.

Referring to Figure 1, the 4-phase handshake protocol is explained as follows[1]. The dual-rail data bus that feeds the current stage register is initially in the spacer state, and the acknowledge input (ACKIN) for the current stage register is high (binary 1), since the acknowledge output (ACKOUT) provided by the next stage register is low (binary 0). The current stage register now transmits a codeword (i.e. valid data). This results in low to high transitions on the bus wires (i.e. any one of the rails of all the dual-rail signals is asserted as binary 1) feeding the function block. After the next stage register receives a codeword, subsequent to data processing in the function block, it drives the ACKOUT wire to binary 1, and the ACKIN wire assumes binary 0. The current stage register waits for the ACKIN signal to become 0 and then resets the data bus, i.e. the data bus feeding the function block is driven to the spacer state. After an unbounded but finite and positive amount of time taken for resetting the function block and the passage of spacer data to the following register stage, the next stage register drives the ACKOUT (ACKIN) to 0 (1). A data transaction is now said to be complete, and the system is ready to proceed with the next transaction. The application of data in the self-timed system depicted by Figure 1 follows the sequence: *valid data* – *spacer* – *valid data* – *spacer*, and so forth.

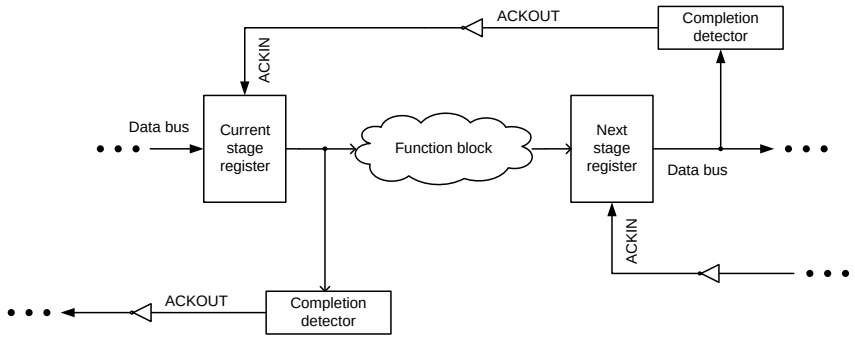

Fig. 1 Standard self-timed system architecture employing delay-insensitive data encoding and 4-phase handshaking

2. Function Blocks – Classification and Timing Behavior

Self-timed function blocks are classified as strongly indicating and weakly indicating [18] depending on the manner in which they indicate (i.e. acknowledge) the arrival of the primary inputs. The differences between the properties of strong and weak-indication function blocks are explained using the illustrative timing diagram shown as Figure 2.

2.1. Strong-indication function block

A strongly indicating function block [19] waits for all the valid/spacer primary inputs to arrive and then starts to compute and produce the desired valid/spacer primary outputs. The strong input-output conditions are given as follows:

[1] This explanation remains valid for any delay-insensitive data encoding scheme.

- All the primary inputs must attain the valid/spacer state before any primary output attains the valid/spacer state
- All the primary outputs must have attained the valid/spacer state before any primary input attains the spacer/valid state

2.2. Weak-indication function block

A weakly indicating function block [20] is capable of producing valid/spacer primary outputs subsequent to the arrival of just a subset of the valid/spacer primary inputs. However, the production of at least one valid/spacer primary output is delayed until all the valid/spacer primary inputs have arrived. Reference [21] discusses two kinds of weak-indication function block implementations: i) distributed implementation [22], where the task of indicating the primary inputs is shared between the primary outputs, and ii) biased implementation [18] [23], where the responsibility of primary inputs indication is just delegated to a single primary output. The weak input-output conditions are given below:

- Some valid/spacer primary outputs are produced subsequent to the arrival of a subset of the valid/spacer primary inputs
- All the valid/spacer primary inputs should have arrived before all the respective valid/spacer primary outputs are produced
- All the valid/spacer primary outputs should have been produced before any subsequent spacer/valid primary input(s) arrive

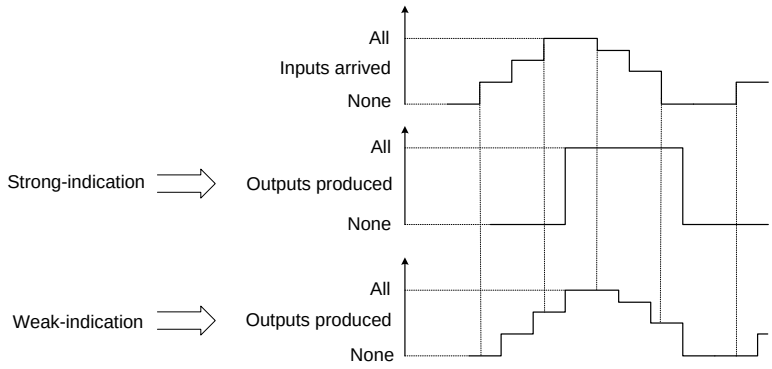

Fig. 2 Input-output behavior of strong and weak-indication function blocks

3. WEAK-INDICATION: BASIC, DISTRIBUTED, AND BIASED IMPLEMENTATIONS

Three types of weakly indicating self-timed full adder implementations viz. basic, distributed, and biased implementations are discussed in this section.

3.1. Basic implementation

The weakly indicating DIMS full adder [24], shown in Figure 3, is an example of the basic weak-indication implementation style. The circuit consists of two levels, with C-elements realizing the product terms in the first level and the OR gates summing up the

product terms in the second level. Only one product term will be activated to produce the sum or carry outputs thus satisfying the monotonic cover constraint [14], which requires that the product terms comprising a Boolean function should be mutually disjoint [25] [26]. The C-element[2] is highlighted by the circle with the marking 'C' on its periphery. A1, A0, B1, B0, CIN1 and CIN0 represent the dual-rail primary inputs, while SUM1, SUM0, COUT1 and COUT0 represent the dual-rail primary outputs. A1 and A0 represent the augend, and B1 and B0 represent the addend inputs. The logic equations governing the full adder are,

$$SUM1 = A0B0CIN1 + A0B1CIN0 + A1B0CIN0 + A1B1CIN1 \quad (1)$$

$$SUM0 = A0B0CIN0 + A0B1CIN1 + A1B0CIN1 + A1B1CIN0 \quad (2)$$

$$COUT1 = A0B1CIN1 + A1B0CIN1 + A1B1 \quad (3)$$

$$COUT0 = A0B1CIN0 + A1B0CIN0 + A0B0 \quad (4)$$

From (1) to (4), it is evident that the sum outputs depend upon all the primary inputs, while the carry outputs may not. When the carry-generation occurs (i.e. A1 = B1 = 1), COUT1 would output binary 1 irrespective of the value of the carry input CIN1 or CIN0. On the other hand, when the carry-kill condition occurs (i.e. A0 = B0 = 1), COUT0 would output binary 1 regardless of the value of the incoming carry signal CIN1 or CIN0. Thus for both valid data and spacer, while the sum outputs have to wait for the arrival of all the primary inputs, the carry outputs may not. However, since the product terms are realized using C-elements for both the sum and carry outputs, the carry-output logic provides an additional acknowledgement for some/all of the primary inputs besides the sum outputs.

When the carry-propagate condition (i.e. A1 = B0 = 1 or A0 = B1 = 1) occurs, the arrival of the augend, addend and carry inputs are indicated by both the sum and carry outputs, i.e., the responsibility of indicating the primary full adder inputs is not shared between the sum and carry outputs, neither is the responsibility confined to a single primary output (i.e. the sum or carry output), but rather, multiple acknowledgments tend to manifest for both valid data and spacers. As a result, if the full adder shown in Figure 3 is cascaded to form an *n*-bit RCA and if the carry signal propagates through a maximum *m* out of *n* full adder stages in the RCA, the forward latency and the reverse latency would be specified by O(*m*). The forward latency signifies the maximum propagation delay incurred in processing the valid data inputs, and the reverse latency denotes the maximum propagation delay encountered for the passage of spacer data inputs (i.e. the time taken for the reset of the self-timed circuit/system) [14]. Hence, the cycle time, which is the time taken for a single data transaction, and computed as the sum of forward and reverse latencies, would be specified by O(2*m*).

[2] The Muller C-element/C-gate basically governs the rendezvous of the input signals. Hence the C-element is also referred to as an 'input-complete element'. It outputs a 1/0 only if all its inputs are 1s/0s respectively. It retains the existing steady-state in case the inputs are different.

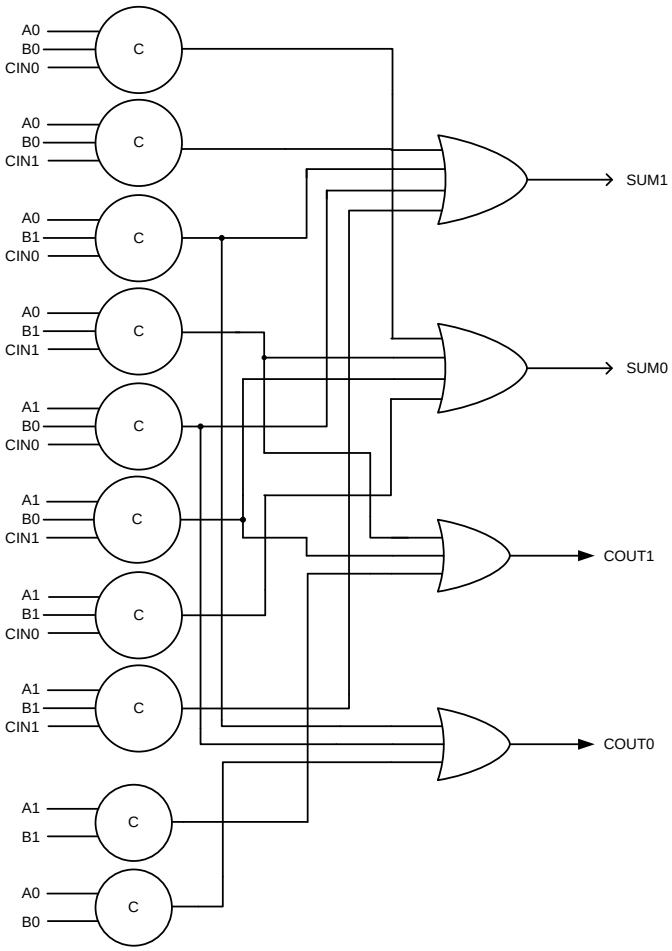

Fig. 3 DIMS weak-indication full adder

3.2. Distributed Implementation

Martin's full adder [22], which forms an example for the distributed weak-indication implementation, is shown in Figure 4. Equations (5) to (8) are synthesized by Martin's full adder using a full-custom static CMOS design involving 42 transistors. In fact, (5) to (8) represent the factorized forms of (1) to (4). The nMOS network realizes the full adder functionality and is activated when the valid data inputs are supplied, while the pMOS network is activated during the application of spacers and resets the full adder during the RTZ phase.

$$SUM1 = (A0B0 + A1B1)\,CIN1 + (A0B1 + A1B0)\,CIN0 \quad (5)$$

$$SUM0 = (A0B0 + A1B1)\,CIN0 + (A0B1 + A1B0)\,CIN1 \quad (6)$$

$$COUT1 = (A0B1 + A1B0)\,CIN1 + A1B1 \quad (7)$$

$$COUT0 = (A0B1 + A1B0)\,CIN0 + A0B0 \quad (8)$$

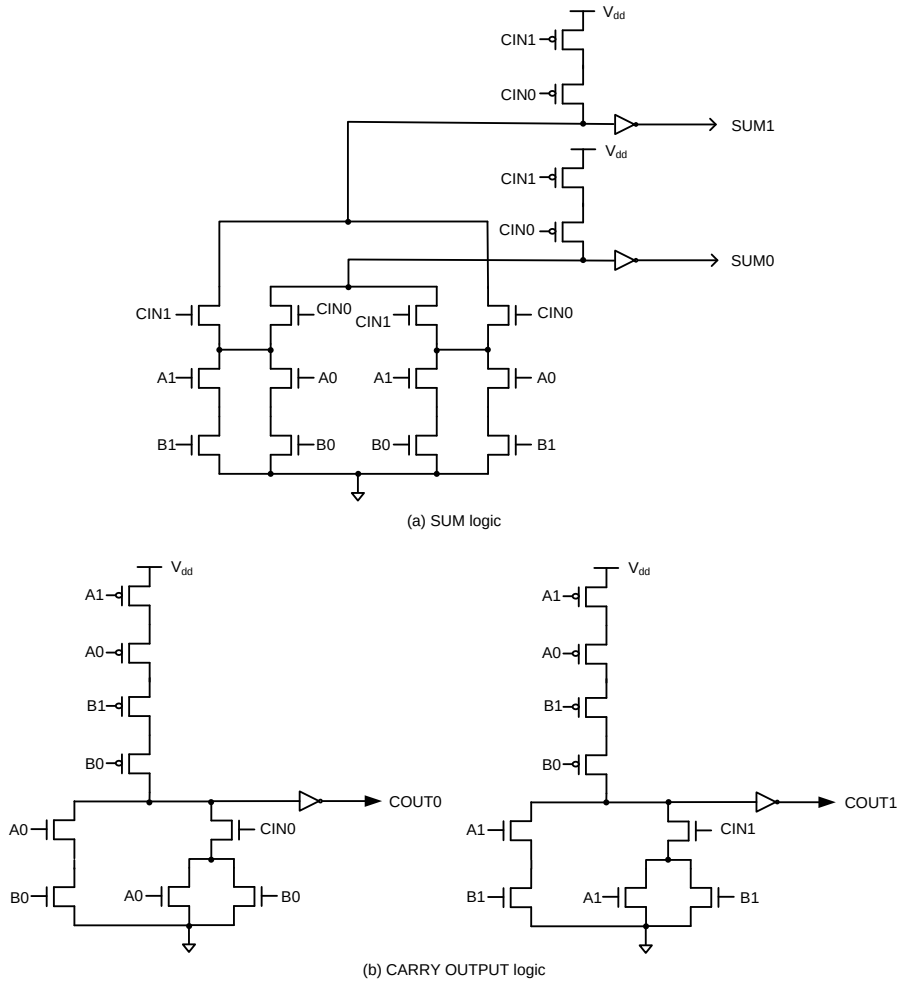

Fig. 4 Martin's weak-indication full adder

When valid data inputs are supplied, the operation of the Martin's full adder is identical to the weakly indicating DIMS full adder discussed previously. Therefore the forward latency of an *n*-bit self-timed RCA employing the Martin's full adder would be specified by O(*m*), where *m* signifies the maximum length of the carry chain activated in the RCA. However when spacer data are applied, the carry output is reset through the spacer states of augend and addend inputs, while the sum output is reset subsequent to the arrival of the carry input as well. Therefore, the responsibility of indicating the primary inputs is distributed between the primary outputs (viz. sum and carry outputs) in the Martin's full adder, especially when spacer data are applied during the RTZ phase.

Since the carry signal alone may be required to propagate through the full adders in an *n*-bit RCA, subject to the propagate mode becoming active, the Martin's full adder paves

the way for a fast and simultaneous reset of all the intermediate carry outputs in the RCA by involving just one full adder delay, soon after the corresponding augend and addend inputs of the full adders have become spacers regardless of the carry input. Once the carry output of a k^{th} stage full adder attains a spacer which in turn serves as the carry input for the $(k + 1)^{th}$ stage full adder, the sum output of the $(k + 1)^{th}$ stage full adder in the RCA would also become a spacer which entails another full adder delay. Therefore the reverse latency of the *n*-bit self-timed RCA employing a cascade of Martin's full adders is not data-dependent unlike the previous case but is just a constant, which is approximately equal to two full adder delays. As a result, the cycle time of the *n*-bit RCA incorporating a cascade of Martin's full adders is specified by $O(m + 2)$.

3.3. Biased implementation

Seitz's weakly indicating full adder design [18], shown in Figure 5, constitutes a good example for the biased implementation style, and synthesizes (1) to (4). Note that the full adder shown in Figure 5 is similar in many aspects to that portrayed by Figure 3 with a few exceptions: i) the product terms are realized using AND gates in Figure 5 instead of the C-elements in Figure 3, and ii) the intermediate sum outputs viz. *intsum*1 and *intsum*0 are combined with the output of the 6-input OR gate (*org*) that logically sums up the dual-rail input signals to produce the primary sum outputs viz. SUM1 and SUM0. The carry output logic of Seitz's weak-indication full adder may utilize the carry-generate or the carry-kill condition similar to that of the DIMS weak-indication full adder or the Martin's full adder. To explain the biased approach prevalent in the design of the Seitz's weak-indication full adder, let us two consider two example scenarios.

When valid data inputs are applied to the full adder shown in Figure 5, and assuming that the carry propagates, one of the 3-input AND gates present in the first-level of the full adder would transition to 1, which will cause a similar transition on *intsum*1 or *intsum*0. Even with a single dual-rail primary input transitioning to 1, *org* would transition to 1. However with isochronic fork assumptions [27] imposed on the primary inputs, it is implied that the low to high transitions on the primary inputs of the AND gate are simultaneously accompanied by similar transitions on the inputs of the 6-input OR gate. The isochronic fork implies that when a transition arrives on one branch of a node and is acknowledged, the transitions on all other branches of the same node are also assumed to have arrived at the same time and hence they are considered to be acknowledged. Subsequently, the low to high transition on *intsum*1/*intsum*0 is combined with the transition on *org*, resulting in the production of a low to high transition on the primary sum output viz. SUM1/SUM0. Notice that the low to high transition on the output of an AND gate would also cause a similar transition on the carry output viz. COUT1/COUT0. In a subsequent RTZ phase, the AND gate which experienced a low to high transition earlier would now output the spacer and this can happen even with anyone of its inputs assuming the spacer state, which leads to the production of a spacer output on COUT1/COUT0, either of which was asserted high previously.

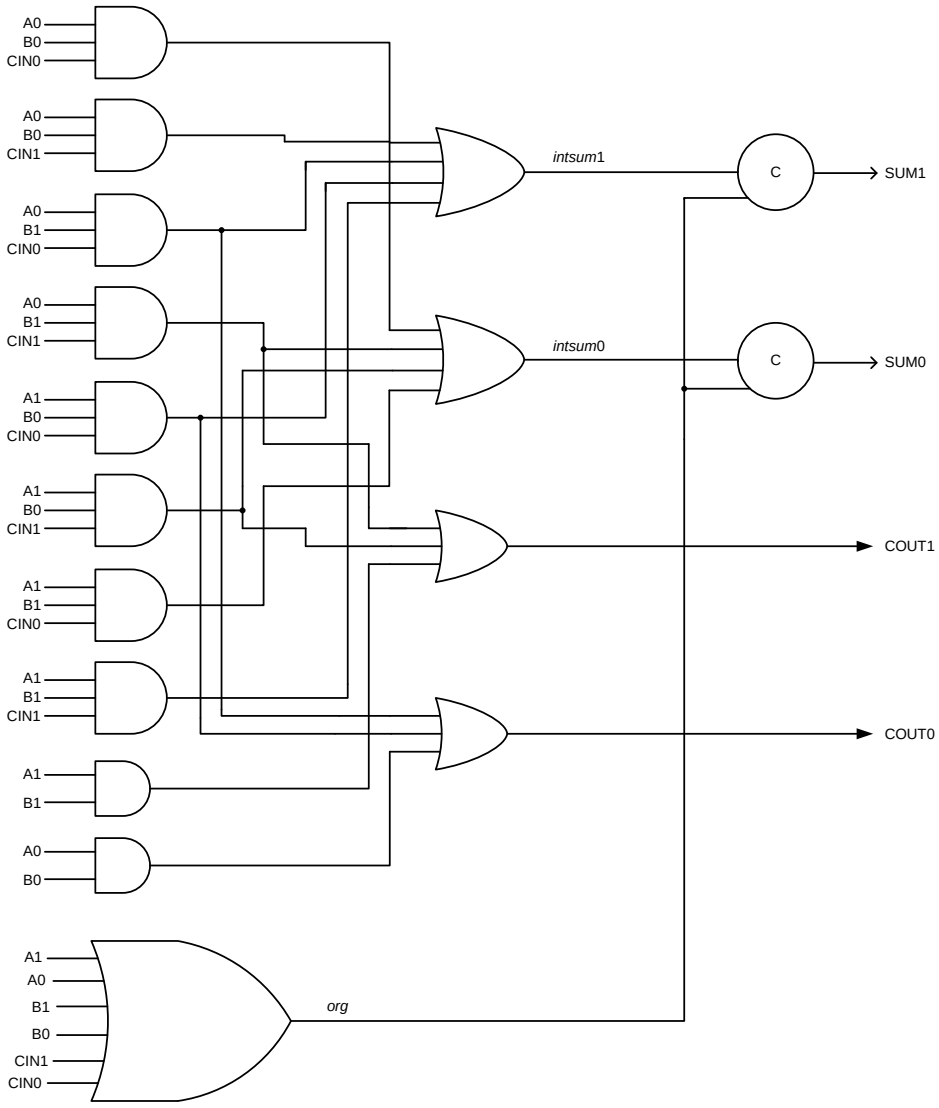

Fig. 5 Seitz's weak-indication full adder

Let us now consider that the carry-generate mode is active. Under this consideration, when valid data inputs are supplied to the full adder shown in Figure 5, a 3-input AND gate and a 2-input AND gate (which implements 'A1B1') would transition to 1. With *org* also transitioning to 1, either SUM1 or SUM0 would experience a low to high transition, and COUT1 would also experience a low to high transition. Notice here that COUT1 acknowledges the arrival of only the augend and addend inputs, i.e. A1 and B1, and not the carry input CIN1/CIN0. However, *intsum*1/*intsum*0 and subsequently SUM1/SUM0 indicates the arrival of the augend and addend inputs as well as the carry input. In the

following RTZ phase, even with either A1 or B1 becoming a spacer, COUT1/COUT0, which transitioned to 1 earlier, would now assume the spacer state. Thus COUT1/COUT0 may not specifically acknowledge the RTZ of A1 and/or B1, nor is the RTZ of the carry input acknowledged. However, *org* that indicates the RTZ of those dual-rail primary inputs which experienced a low to high transition previously, when coupled with *intsum*1/*intsum*0 results in the RTZ of SUM1/SUM0 respectively. Hence the primary sum output is found to assume the entire responsibility of duly indicating the RTZ of all the primary inputs. This deliberation would equally apply for the carry-kill condition.

From the preceding discussions, it may be understood that in the case of Seitz's weak-indication full adder, the sum output assumes the responsibility of indicating the complete arrival of all the primary inputs subsequent to the application of valid or spacer data inputs and that the carry output is freed from indication constraints; thus there is a bias towards the carry output. Nevertheless, this tends to benefit by paving the way for fast carry propagation between the full adder stages in an *n*-bit RCA.

Also, note that the 6-input OR gate shown as part of Figure 5 cannot be decomposed arbitrarily due to the gate orphan problem [21] [29] that would arise for the application of valid data. The gate orphan implies an unacknowledged transition at a gate output.

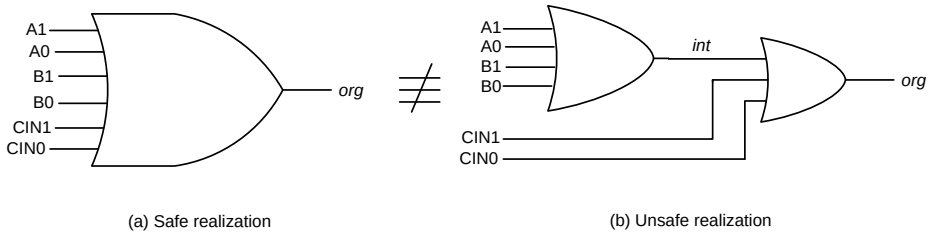

Fig. 6 Naïve decomposition of the 6-input OR gate, potentially causing a gate orphan

To explain the gate orphan problem associated with a naïve logic decomposition, consider Figure 6, wherein the 6-input OR gate is decomposed into a 4-input OR gate and a 3-input OR gate. Supposing during a valid data phase CIN1 transitions to 1, output *org* will transition to 1 in the case of both the realizations shown in Figure 6 without waiting for the low to high transitions to occur on the remainder of the dual-rail inputs viz. A1/A0 and B1/B0. Subsequently, if A1/A0 and B1/B0 also experience transitions, they will not be acknowledged by the OR gate in Figure 6a but they do not give rise to wire orphans since they are considered to be acknowledged by the AND gate(s) present in the first level of Figure 5 through the isochronic fork assumption. Let us revisit the similar scenario of CIN1 transitioning to 1 before A1/A0 and B1/B0 experience low to high transitions with reference to Figure 6b. It can be seen that after CIN1 experiences a low to high transition, the output *org* will also experience a low to high transition irrespective of any transition occurring on the intermediate output *int*. Subsequently if A1/A0 and B1/B0 also transition to 1, the internal output *int* will experience a transition to 1. However, the low to high transition on *int* will not be acknowledged by the output *org*, and the unacknowledged transition on the internal gate output (*int*) is referred to as a gate orphan, which may get eliminated only through sophisticated timing assumptions. Gate orphans are problematic and tend to affect the robustness of a self-timed circuit/system. Therefore, self-timed implementations should be devoid of gate orphans in order to be robust.

It can be inferred from [28] that a strong-indication *n*-bit RCA constructed using strongly indicating full adder blocks has fixed forward and reverse latencies of O(*n*), and hence exhibits the worst-case cycle time of O(2*n*). On the other hand, the weak-indication *n*-bit RCA composed using basic weak-indication full adders has similar forward and reverse latencies of O(*m*), and hence features a cycle time of O(2*m*), where *m* denotes the maximum length of carry propagation in the *n*-bit RCA. The weak-indication *n*-bit RCA constructed using the distributed or biased implementation style weak-indication full adders have forward and reverse latencies of O(*m*) and O(2), and hence the least cycle time of O(*m* + 2) [28]. Given these, weak-indication realizations are preferable than their strong-indication counterparts for self-timed design of arithmetic circuits.

4. Proposed Biased Implementation Style Weak-Indication Full Adder

The proposed full adder that corresponds to the biased implementation style of weak-indication is depicted in Figure 7, synthesizing (5) to (8) using 4 simple gates viz. 2-input OR gates and 10 complex gates. Among the 10 complex gates, 8 of them are 2-input Muller C-elements, where a 2-input C-element is realized using an AO222 cell with feedback, and the remaining are AO21 gates. Assuming X and Y are the inputs and Z is the output of a 2-input C-element, Z = XY + (X + Y) Z. Presuming that A and B are the inputs given to the AND logic part of an AO21 gate and with C as its other input, the output of the AO21 gate, say D, is given by D = AB + C. When AB and/or C are equal to 1, D equates to 1; hence the AO21 gate is said to be input-incomplete. In general, with the exception of the C-gate, all other logic gates tend to exhibit input-incomplete behavior.

It can be seen in Figure 7 that the product terms corresponding to the sum logic are realized using input-complete C-elements. Hence the sum output would indicate the complete arrival of the entire primary inputs viz. augend, addend and carry inputs for both valid and spacer data. On the other hand, the carry output logic is realized using a mix of input-complete C-elements and input-incomplete complex gates. Since the sum output fully indicates the arrival of all the primary inputs during the valid and spacer data phases, the carry output at the best provides multiple acknowledgments for the arrival of valid or spacer data on the augend and addend inputs and/or the carry input. Hence the proposed full adder features a bias toward the carry output in terms of relaxing its indication constraints, and the advantages associated with such an implementation in terms of less forward and reverse latencies and cycle time have been articulated earlier. To elaborate on this, let us consider the following:

- *Carry-propagate mode*: Once the primary inputs assume valid data states, internal outputs *intm*1 or *intm*2 and *intm*3, shown in Figure 7, would transition to 1. Depending on whether CIN1 or CIN0 experiences a low to high signal transition, a similar transition is reflected on the corresponding primary output, COUT1 or COUT0. When spacer data are applied subsequently, even with *intm*1 or *intm*2 and *intm*3 becoming a spacer, the carry output which transitioned to 1 earlier would now be reset regardless of the carry input becoming a spacer
- *Carry-generate or Carry-kill mode*: When valid data are supplied through the primary inputs, an intermediate output *intm*4 or *intm*5, highlighted in Figure 7, makes a low to high signal transition, which is followed by a similar transition on

COUT1/COUT0 respectively. This could occur regardless of a transition on the carry input. Subsequently, when spacer data are applied on the primary inputs, *intm*4 or *intm*5 which transitioned to 1 earlier would now assume the spacer state, which is acknowledged by the respective carry output. This could also happen irrespective of the carry input becoming a spacer

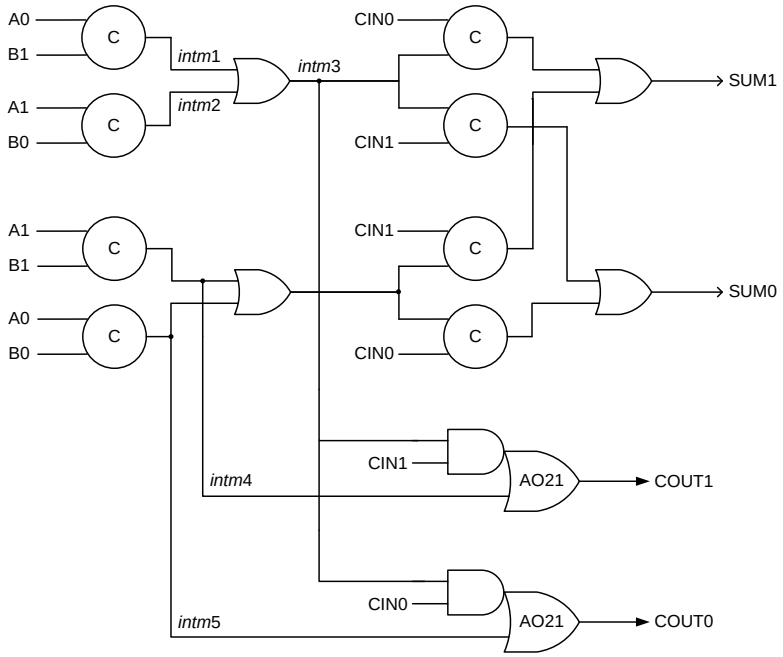

Fig. 7 Proposed weak-indication full adder

5. Simulation Results and Discussion

A number of 32-bit self-timed RCAs were constructed in a semi-custom design fashion at the gate-level by utilizing the various strong and weak-indication full adders separately. The structural integrity of the different gate-level self-timed full adders was preserved during the physical realization (technology mapping) to pave the way for a legitimate comparison, and they were implemented using the elements of a 32/28nm CMOS cell library [30]. The 2-input C-element was alone designed manually using the AO222 cell with feedback and was made available to realize the self-timed designs, and the 3-input C-elements were decomposed safely into 2-input C-elements using the method of [29]. The self-timed RCAs comprise the function block, the input registers, and the completion detection circuit. The input registers and the completion detector part of the various RCAs are identical, and only the function blocks differ. Hence the differences between the simulation results of the various RCAs can be attributed to the differences between their constituent full adders. More than 1000 random input vectors were supplied to the RCAs at time intervals of 20ns through test benches in order to capture the

switching activities. The .vcd files generated were subsequently used for power estimation using Synopsys tools. Since the EDA tool estimates just critical path timing, only the worst-case forward latency was evaluated. Appropriate wire loads were included automatically whilst performing the simulations. As part of the advanced timing analysis, a virtual clock was used just to constrain the input and output ports of the RCAs, and it did not consume any power. The power, (forward) latency, and area results obtained for the various 32-bit RCAs are shown in Table 1. The indication type of each full adder is highlighted in the 1st column of Table 1. The area of the RCAs and the respective full adders are given before and after the semicolon in the 4th column. The gates present in the critical path of the different RCAs are mentioned in the 5th column. The simulation results correspond to a typical case specification (1.05V, 25^0 C) of the 32/28nm CMOS process [30]. The primary sum and carry outputs of the RCAs possess fanout-of-4 drive strength.

Table 1 Power, latency and area parameters of different 32-bit self-timed RCAs incorporating distinct full adders

Full adder and its indication type	Power (μW)	Latency (ns)	RCA; Full adder area (μm^2)	Critical path elements
Singh [31] – Strong	2190	14.61	2529; 54.64	2 CE2, 2 OR3
DIMS [24] – Strong	2181	9.26	2504.60; 53.88	CE2, OR4
DIMS [24] [14] – Weak	2177	8.24	2423.27; 51.34	CE2, OR3
Toms [32] – Strong	2172	9.04	2293.14; 47.27	CE2, 2 OR2
Folco *et al*. [33] – Weak	2171	7.00	2016.63; 38.63	CE2, OR2
SSSC [23] – Weak	2174	4.43	2097.96; 41.17	AO222
Toms & Edwards [34] – Weak	2192	9.66	2642.85; 58.20	AND2, CE2, OR3
Proposed – Weak	2171	3.32	2049.16; 39.65	AO21

CE2: 2-input C-element; AND2: 2-input AND gate; OR2/3/4: 2/3/4-input OR gate; AO222 and AO21 are complex gates

The reason for the differences in the latency figures of the various RCAs is due to the different logical operators found in their critical paths, as mentioned in Table 1. It can be seen in Table 1 that the 32-bit RCA incorporating the proposed full adder features the least latency of 3.32ns among its counterparts – thanks to the AO21 cell used for implementing the carry output logic of the proposed full adder. With respect to power dissipation, the 32-bit RCA featuring the Folco *et al*.'s full adder is comparable with that incorporating the proposed full adder since both these dissipate a similar average power of 2171μW. It can be seen that the total power dissipation does not vary much across the different RCAs, although the variations in area are quite significant. This is because self-timed designs have a unique signal propagation path for each input pattern unlike synchronous designs as they adhere to the monotonic cover constraint [14]. In terms of area, the 32-bit RCA incorporating Folco *et al*.'s full adder occupies the least area while the 32-bit RCA constructed using the proposed full adder occupies more Silicon by just 1.6%. However, the latter enables considerably less latency by 52.6% compared to the former. Also, note that this latency reduction is achieved not at the expense of any extra power dissipation for the latter compared to the former. The SSSC full adder which is a gate-level design features a carry output logic that is similar to the carry output logic of

Martin's full adder which is a transistor-level design. Hence the SSSC full adder was considered as a substitute for the Martin's full adder in implementing the RCA as both these have similar latencies and cycle time metrics. The weak-indication SSSC full adder corresponds to the biased implementation style similar to that of the proposed full adder, and the proposed full adder leads to reduced latency by 25.1% than the SSSC full adder for a 32-bit RCA implementation with no penalty in terms of power or area parameters.

6. CONCLUSION

A new full adder design that corresponds to the biased implementation style of weak-indication was presented. For an *n*-bit RCA realized using the proposed full adder, the forward and reverse latencies and cycle time are specified by O(*m*), O(2), and O(*m* + 2) respectively, where *m* denotes the maximum number of full adder stages in the *n*-bit RCA through which the carry propagates. Example 32-bit self-timed RCA implementations incorporating different strong and weak-indication full adders were analyzed. Against the best of the strong-indication full adders, the proposed full adder reports respective reductions in latency and area by 63.3% and 10.6% whilst dissipating similar power. On the other hand, in comparison with the existing optimized weak-indication full adder, the proposed full adder achieves reduced latency by 25.1% with no power penalty albeit at a small area expense of 1.6%. Overall, from a combined power-latency-area perspective, the proposed full adder is found to yield optimum quality-of-results.

REFERENCES

[1] Semiconductor Industry Association's ITRS report. Available: http://www.itrs.net
[2] S. Kundu and A. Sreedhar, *Nanoscale CMOS VLSI Circuits: Design for Manufacturability*, McGraw-Hill, USA, 2010.
[3] N.C. Paver, P. Day, C. Farnsworth, D.L. Jackson, W.A. Lien and J. Liu, "A low-power, low noise, configurable self-timed DSP", In Proceedings of the 4th International Symposium on Advanced Research in Asynchronous Circuits and Systems, 1998, pp. 32-42.
[4] G.F. Bouesse, G. Sicard, A. Baixas and M. Renaudin, "Quasi delay insensitive asynchronous circuits for low EMI", In Proceedings of the 4th International Workshop on Electromagnetic Compatibility of Integrated Circuits, 2004, pp. 27-31.
[5] C.H. Van Kees Berkel, M.B. Josephs and S.M. Nowick, "Scanning the technology applications of asynchronous circuits", In Proceedings of the *IEEE*, vol. 87, pp. 223-233, 1999.
[6] K.J. Kulikowski, V. Venkataraman, Z. Wang, A. Taubin and M. Karpovsky, "Asynchronous balanced gates tolerant to interconnect variability", In Proceedings of the IEEE International Symposium on Circuits and Systems, 2008, pp. 3190-3193.
[7] I.J. Chang, S.P. Park and K. Roy, "Exploring asynchronous design techniques for process-tolerant and energy-efficient subthreshold operation", *IEEE Journal of Solid-State Circuits*, vol. 45, pp. 401-410, 2010.
[8] O.C. Akgun, J. Rodrigues and J. Sparsø, "Minimum-energy sub-threshold self-timed circuits: design methodology and a case study", In Proceedings of the 16th IEEE International Symposium on Asynchronous Circuits and Systems, 2010, pp. 41-51.
[9] I. David, R. Ginosar and M. Yoeli, "Self-timed is self-checking", *Journal of Electronic Testing: Theory and Applications*, vol. 6, pp. 219-228, 1995.
[10] Z.C. Yu, S.B. Furber and L.A. Plana, "An investigation into the security of self-timed circuits", In Proceedings of the 9th International Symposium on Asynchronous Circuits and Systems, 2003, pp. 206-215.
[11] D. Sokolov, J. Murphy, A. Bystrov and A. Yakovlev, "Design and analysis of dual-rail circuits for security applications", *IEEE Transactions on Computers*, vol. 54, pp. 449-460, 2005.

[12] D. Koppad and A. Efthymiou, "BIST for strongly-indicating asynchronous circuits", In Proceedings of the 17th IFIP International Conference on Very Large Scale Integration, 2009, pp. 215-218.
[13] A. Efthymiou, "Initialization-based test pattern generation for asynchronous circuits", *IEEE Transactions on VLSI Systems*, vol. 18, pp. 591-601, 2010.
[14] J. Sparsø and S. Furber (Editors), *Principles of Asynchronous Circuit Design: A Systems Perspective*, Kluwer Academic Publishers, Netherlands, 2001.
[15] Balasubramanian Padmanabhan, "Self-timed logic and the design of self-timed adders", *PhD thesis*, School of Computer Science, The University of Manchester, UK, 2010.
[16] T. Verhoeff, "Delay-insensitive codes – an overview", *Distributed Computing*, vol. 3, pp. 1-8, 1998.
[17] B. Bose, "On unordered codes", *IEEE Transactions on Computers*, vol. 40, pp. 1-8, 1988.
[18] C.L. Seitz, "System Timing", *Introduction to VLSI Systems*, C. Mead and L. Conway (Editors), pp. 218-262, Addison-Wesley, Reading, Massachusetts, USA, 1980.
[19] P. Balasubramanian and D.A. Edwards, "Efficient realization of strongly indicating function blocks", In Proceedings of the IEEE Computer Society Annual Symposium on VLSI, 2008, pp. 429-432.
[20] P. Balasubramanian and D.A. Edwards, "A new design technique for weakly indicating function blocks", In Proceedings of the 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems, 2008, pp. 116-121.
[21] C. Jeong and S.M. Nowick, "Block-level relaxation for timing-robust asynchronous circuits based on eager evaluation", In Proceedings of the 14th IEEE International Symposium on Asynchronous Circuits and Systems, pp. 95-104, 2008.
[22] A.J. Martin, "Asynchronous datapaths and the design of an asynchronous adder", *Formal Methods in System Design*, vol. 1, pp. 117-137, 1992.
[23] P. Balasubramanian and D.A. Edwards, "A delay efficient robust self-timed full adder", In Proceedings of the IEEE 3rd International Design and Test Workshop, 2008, pp. 129-134.
[24] J. Sparsø and J. Staunstrup, "Delay-insensitive multi-ring structures", *Integration, the VLSI Journal*, vol. 15, pp. 313-340, 1993.
[25] P. Balasubramanian and D.A. Edwards, "Self-timed realization of combinational logic", In Proceedings of the 19th International Workshop on Logic and Synthesis, 2010, pp. 55-62.
[26] P. Balasubramanian, R. Arisaka and H.R. Arabnia, "RB_DSOP: a rule based disjoint sum of products synthesis method", In Proceedings of the 12th International Conference on Computer Design, 2012, pp. 39-43.
[27] A.J. Martin, "The limitation to delay-insensitivity in asynchronous circuits", In Proceedings of the 6th MIT Conference on Advanced Research in VLSI, 1990, pp. 263-278.
[28] P. Balasubramanian and N.E. Mastorakis, "Timing analysis of quasi-delay-insensitive ripple carry adders – a mathematical study", In Proceedings of the 3rd European Conference of Circuits Technology and Devices, 2012, pp. 233-240.
[29] P. Balasubramanian and N.E. Mastorakis, "QDI decomposed DIMS method featuring homogeneous/ heterogeneous data encoding", In Proceedings of the International Conference on Computers, Digital Communications and Computing, 2011, pp. 93-101.
[30] Synopsys Digital Standard Cell Library SAED_EDK32/28_CORE Databook, Revision 1.0.0, 2012.
[31] N.P. Singh, "A design methodology for self-timed systems", *M.Sc. thesis*, MIT Laboratory for Computer Science Technical Report TR-258, 1981.
[32] W.B. Toms, "Synthesis of quasi-delay-insensitive datapath circuits", *PhD thesis*, School of Computer Science, The University of Manchester, UK, 2006.
[33] B. Folco, V. Bregier, L. Fesquet and M. Renaudin, "Technology mapping for area optimized quasi delay insensitive circuits", In Proceedings of the IFIP International Conference on Very Large Scale Integration, 2005, pp. 146-151.
[34] W.B. Toms and D.A. Edwards, "A complete synthesis method for block-level relaxation in self-timed datapaths", In Proceedings of the 10th International Conference on Application of Concurrency to System Design, 2010, pp. 24-34.