Hindawi Publishing Corporation
VLSI Design
Volume 2012, Article ID 575389, 13 pages
doi:10.1155/2012/575389

Research Article

Redundant Logic Insertion and Latency Reduction in Self-Timed Adders

P. Balasubramanian,[1,2] D. A. Edwards,[3] and W. B. Toms[3]

[1] *Department of Electronics and Communication Engineering, Vel Tech Dr. RR and Dr. SR Technical University, Avadi, Tamil Nadu, Chennai 600 062, India*
[2] *Department of Electronics and Communication Engineering, S.A. Engineering College, Anna University, Thiruverkadu, Tamil Nadu, Chennai 600 077, India*
[3] *School of Computer Science, The University of Manchester, Oxford Road, Manchester M13 9PL, UK*

Correspondence should be addressed to P. Balasubramanian, spbalan04@gmail.com

Received 8 October 2011; Revised 24 January 2012; Accepted 17 March 2012

Academic Editor: Sungjoo Yoo

A novel concept of logic redundancy insertion is presented that facilitates significant latency reduction in self-timed adder circuits. The proposed concept is universal in the sense that it can be extended to a variety of self-timed design methods. Redundant logic can be incorporated to generate efficient self-timed realizations of iterative logic specifications. Based on the case study of a 32-bit self-timed carry-ripple adder, it has been found that redundant implementations minimize the data path latency by 21.1% at the expense of increases in area and power by 2.3% and 0.8% on average compared to their nonredundant counterparts. However, when considering further peephole logic optimizations, it has been observed in a specific scenario that the delay reduction could be as high as 31% while accompanied by only meager area and power penalties of 0.6% and 1.2%, respectively. Moreover, redundant logic adders pave the way for spacer propagation in constant time and garner actual case latency for addition of valid data.

1. Introduction

The 2009 International Technology Roadmap on Semiconductor (ITRS) design predicts that adaptive digital circuits will be increasingly necessary for the future as a consequence of increase in variability [1]. This is owing to a blurring of the boundary between catastrophic faults in circuits caused due to manufacturing defects and parametric faults resulting from device and interconnects variability. The ITRS roadmap [1] projects a growing requirement for asynchronous global signaling and emphasizes the need for a continuous development of asynchronous logic/circuit design tools. This is significant in the context of a key challenge faced in modern IC design, namely, distribution of a centralized clock signal throughout the chip with acceptably low skew whilst having to keep the power, congestion, and area costs of traditional repeater insertion in long global clock lines to a minimum. Indeed as variability increases, circuits can exhibit faulty behavior similar to that caused by catastrophic defects. The major sources of failures include (i) process variations—statistical variations of device parameters such as channel length, threshold voltage, and mobility, (ii) lifetime variations—variations causing shift in physical parameters over the operating life of a circuit, and (iii) intrinsic noise—noise sources (shot noise, thermal noise, and random noise) which are inherent to normal device operation that becomes dominant at small feature sizes. At a time when the issue of variability has become prominent and the reliability aspect tending to assume greater significance than quality of results in nanometer scale digital circuits, the self-timed design paradigm offers an attractive alternative to conventional synchronous design. In fact, self-timed logic circuits are inherently tolerant of process, temperature, and parameter uncertainties [2–6]. A recent work [7] by Chelcea et al. demonstrated the superior resiliency of asynchronous circuits vis-à-vis their synchronous counterparts in the presence of parametric variations (probabilistic device delays) for the case of a 32-bit Brent-Kung adder and a 16-bit multiplier.

Self-timed circuits also have better electromagnetic compatibility [8] and noise susceptibility attributes compared to synchronous designs [9], consume power only where and when active [10, 11], and feature excellent design reusability [12]. Moreover, self-timed circuits are self-checking [13, 14] and are latency insensitive thus being naturally elastic or adaptive.

Although the term "self-timed" has been used to refer to asynchronous circuits, it is important to note that self-timed circuits actually constitute a robust class of asynchronous circuits, namely, input/output mode circuits. In general, circuits corresponding to the input/output operating mode do not impose timing assumptions on when the environment should respond to the circuit. The robustness attribute in self-timed circuits usually results from employing a delay-insensitive (DI) code for data representation, communication, and processing, and a 4-phase (return-to-zero) handshake signaling convention is commonly adopted. Among the family of DI codes [15], the dual-rail (1-of-2) code is widely preferred owing to its simplicity and ease of logic implementation.

According to dual-rail data encoding, each data wire d is represented using two encoded data wires $d0$ (d^0) and $d1$ (d^1) as shown in Figure 1. A transition on the $d0$ wire indicates that a *zero* has been transmitted, while a transition on the $d1$ wire indicates that a *one* has been transmitted. Since the request signal is embedded within the data wires, a transition on either $d0$ or $d1$ informs the receiver about the validity of data. The condition of both $d0$ and $d1$ being a zero at the same time is referred to as the *spacer* or *empty data*. Both $d0$ and $d1$ are not allowed to transition simultaneously as it is illegal and invalid since the coding scheme utilized is *unordered* [16], where no codeword forms a subset of another codeword.

With reference to Figure 1, the 4-phase handshake protocol is explained as follows (the explanation remains valid for data representation using any DI data encoding scheme).

(i) The dual-rail data bus is initially in the spacer state. The sender transmits the codeword (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 assigned logic "high" state), which correspond to nonzero bits of the codeword.

(ii) After the receiver receives the codeword, it drives the *ackout* (*ackin*) wire "high" ("low").

(iii) The sender waits for the *ackin* to go "low" and then resets the data bus (i.e., the data bus is driven to the spacer state).

(iv) After an unbounded but finite (positive) amount of time, the receiver drives the *ackout* (*ackin*) wire "low" ("high"). A single transaction is now said to be complete, and the system is ready to proceed with the next transaction.

The timing diagram for the 4-phase asynchronous signaling protocol is shown in Figure 2, with the request (*req*) signal, which is actually embedded within the data wires, explicitly shown to describe the handshaking. The dual-rail code is the simplest member of the general family of delay-insensitive m-of-n codes [15], where m lines are asserted "high" out of a total of n physical lines to represent a codeword. The size (i.e., number of unique symbols) of a generic m-of-n code is given by the binomial coefficient n choose $m = n!/m!(n-m)!$. The dual-rail code is ideally suited for representing a single bit of binary information. To represent two bits of information, the dual-rail code can be concatenated as shown in Table 1 or can equivalently be represented through a 1-of-4 code.

The 1-of-4 encoded values of single-rail inputs given in Table 1 represent only one of many possible encodings, and an arbitrary choice is portrayed here. Two binary bits of information are represented by asserting only half of the physical lines as logic "high" in the 1-of-4 code in comparison with a dual-rail code, although both the coding schemes require the same number of physical lines. As a result, the 1-of-4 encoding scheme experiences only half the transitions of the dual-rail encoding convention. Thus the dynamic power dissipation of the former scheme is likely to be better than that of the latter due to reduced switching activity. This phenomenon was confirmed with the practical example of an ARM thumb instruction decoder [17]. However, considering the additional encoding and decoding circuitry required for realizing 1-of-4 encoded self-timed data paths in comparison with dual-rail encoded self-timed data paths [18], the power savings gained by the former might diminish.

Although higher order encoding schemes are available, apart from the dual-rail code that allows easier mapping between conventional binary functions, the other widely used DI code is the 1-of-4 code. This is owing to the reason that for self-timed data paths, encoding by sender and membership test and decoding by receiver are important aspects, and consequently the encoding and decoding complexity is dependent on the message space to be coded [19]. When the dual-rail code and 1-of-4 code are used to represent exactly one bit and two bits of binary information, respectively, they are said to be *complete* [14]. A code is said to be complete if and only if it contains all code words as implied by its definition. Even with one missing codeword, it would be labeled *incomplete*. A DI coding scheme, in general, is required to be unordered and complete.

Seitz classified a self-timed logic circuit into two robust categories on the basis of its indicating (acknowledging) genre as *strongly indicating* and *weakly indicating* [20]. It was also shown therein that a legal interconnection of strongly or weakly indicating logic circuits gives rise to a larger strong or weak-indication logic circuit.

(i) *Strong Indication.* In this case, the self-timed circuit waits for all of its inputs (valid/spacer) to arrive before it starts to produce all the outputs (valid/spacer). The sequencing constraints are given below:

(a) all the inputs become defined (valid)/undefined (spacer) before any output becomes defined/undefined; that is, any or all of the output(s) become defined/undefined only after all the inputs have become defined/undefined,

Table 1: Data representation via dual-rail and 1-of-4 encoding formats.

Single-rail inputs		Dual-rail encoded data		1-of-4 encoded data			
A	B	($A1$ $A0$)	($B1$ $B0$)	$E0$	$E1$	$E2$	$E3$
0	0	(0 1)	(0 1)	0	0	0	1
0	1	(0 1)	(1 0)	0	0	1	0
1	0	(1 0)	(0 1)	0	1	0	0
1	1	(1 0)	(1 0)	1	0	0	0

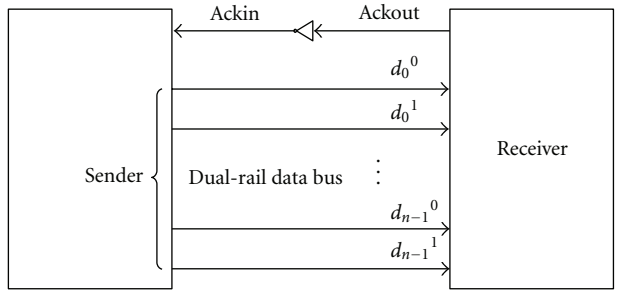

Figure 1: Delay-insensitive dual-rail data encoding and 4-phase handshaking.

(b) all the outputs become defined/undefined before any input becomes undefined/defined.

(ii) *Weak Indication.* According to this, the self-timed circuit is allowed to produce any of the outputs (valid/spacer) even with a subset of the inputs (valid/spacer). However, Seitz's weak timing specifications require that at least one output (valid/spacer) should not have been produced until after all the inputs (valid/spacer) have arrived. The sequencing constraints in this case are as follows.

(a) Some inputs become defined (undefined) before some outputs become defined (undefined); that is, some outputs could become defined (undefined) only after at least some inputs have become defined (undefined).

(b) All the inputs become defined (undefined) before all the outputs become defined (undefined); that is, all the outputs could become defined (undefined) only after all the inputs have become defined (undefined).

(c) All the outputs become defined (undefined) before any input becomes undefined (defined).

The signaling scheme for strong- and weak-indication timing regimes in terms of the input-output characteristics is illustrated graphically in Figure 3, which summarizes the sequencing constraints mentioned above. In general for iterative circuits, weakly indicating implementations are preferable compared to strongly indicating versions since the former's computation time is data dependent for valid data and may exhibit constant latency for spacer data, while the latter is always bound by worst-case latency for both valid data and spacers [21].

2. Redundant Logic Insertion

This section deals with an efficient method of reducing the critical path delay of self-timed adders by means of a novel concept called *redundant logic insertion*. In general, the concept can be extended to effect latency reduction in any iterative logic circuit that comprises a cascade of basic building blocks. Redundancy insertion, in general, implies inclusion of extra redundant logic into a non redundant implementation without modifying the original function that synthesizes the desired functionality to enable speeding up the propagation of certain signals, which are required to drive the subsequent stages of a circuit cascade.

Logic redundancy can be incorporated into a self-timed circuit implementation by careful duplication of similar logic, and this can lead to multiple acknowledgements, which might be useful in simplifying the timing assumptions. Additionally, this procedure could facilitate faster reset of logic during the return-to-zero phase with a constant latency. Logic redundancy achieved through input-incomplete gates basically introduces weak-indication property into the circuit as it relaxes the indication constraints of those outputs that are considered as candidates for optimization. (*Input-incomplete gates* need not have to wait for the arrival of all their inputs to produce the required output under all scenarios; examples include AND gates and OR gates. If any one of its inputs is assigned a 0(1), the output of the AND gate (OR gate) becomes a 0(1)). It can either be implicit or explicit in the circuit. The minor drawbacks of this approach are insignificant increases in area and power parameters. Since logic duplication is involved, switching activity would increase due to multiple acknowledgements, consequently pushing up the dynamic power and resulting in increased average power dissipation. However, the area and power overheads may be marginal depending upon the functionality and its initial nonredundant implementation,

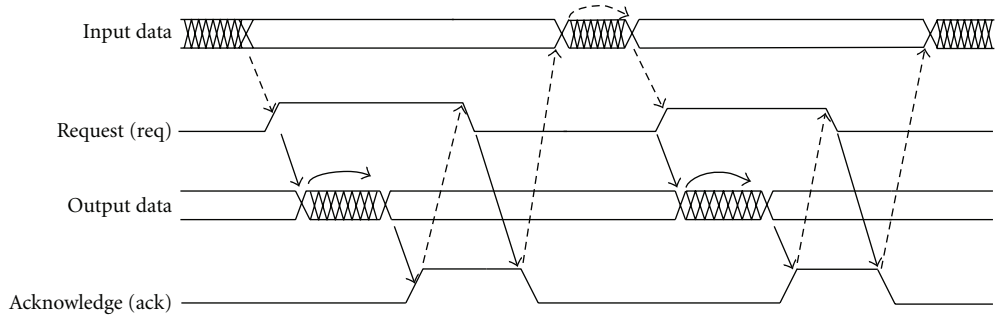

Figure 2: Timing diagram of a 4-phase handshake discipline.

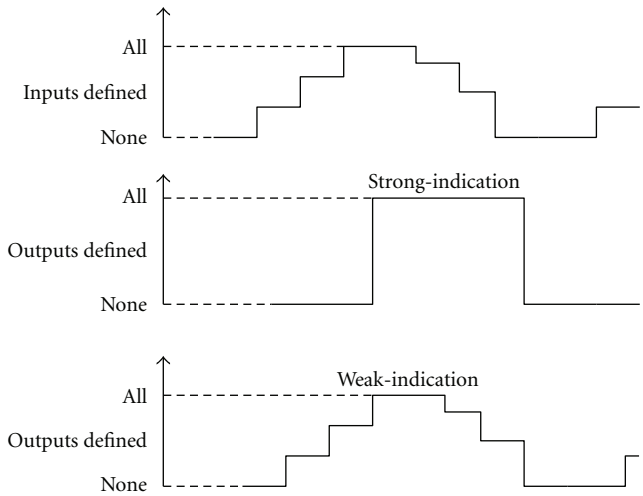

Figure 3: Depicting the input-output behavior of strong-and weak-indication circuits.

and eventually the degree of logic redundancy introduced. We will now consider some case studies to demonstrate the benefits of redundancy insertion on the basis of the self-timed ripple carry adder (RCA) architecture, where logic redundancy is targeted towards the carry output function since the carry is required to propagate between successive stages of the adder architecture.

2.1. Implicit Logic Redundancy. The basic equations corresponding to a dual-rail encoded full adder are given by (1)–(4). Here (*a0*, *a1*), (*b0*, *b1*), and (cin0, cin1) represent the dual-rail encoded augend, addend, and carry inputs of the adder, while (Sum*0*, Sum*1*) and (Cout*0*, Cout*1*) represent the dual-rail encoded sum and carry outputs of the adder, respectively:

$$\begin{aligned}
\text{Sum1} &= a0b0\text{cin1}+a0b1\text{cin0}+a1b0\text{cin0}+a1b1\text{cin1},\\
\text{Sum0} &= a0b0\text{cin0}+a0b1\text{cin1}+a1b0\text{cin1}+a1b1\text{cin0},\\
\text{Cout1} &= a0b1\text{cin1}+a1b0\text{cin1}+a1b1\text{cin0}+a1b1\text{cin1},\\
\text{Cout0} &= a0b0\text{cin0}+a0b0\text{cin1}+a0b1\text{cin0}+a1b0\text{cin0}.
\end{aligned} \tag{1}$$

The circuit shown in Figure 4 corresponds to our synthesized dual-rail encoded full adder, henceforth referred to as the SSSC_DRE adder (single sum, single carry dual-rail encoded adder). Three steps are involved in the synthesis process—(i) deriving the minimum orthogonal sum-of-products form of a given logic functionality [22], (ii) speed-independent decomposition of logic to facilitate realization using standard cells [23], and (iii) performing logic optimizations to pave the way for latency reduction. In the figures, the C-element is represented by the AND gate symbol with the marking *C* on its periphery. (The Muller C-element governs the rendezvous of input signals. It produces a 1(0) if all its inputs are 1(0); otherwise it retains its existing steady state. The C-element (also called C-gate) is classified as an *input-complete gate* as it waits for the arrival of all its input(s) to produce the desired output).

Firstly, it can be noticed that the responsibility of indication is confined to the sum outputs of the adder block, thereby freeing the carry signal from indication constraints which facilitates fast carry propagation. Even with the arrival of a subset of the inputs, the carry outputs could become defined/undefined, while the sum outputs would have to wait for the arrival of all the inputs to become defined/undefined. Thus the full adder satisfies Seitz's weak-indication timing constraints. This style of implementation is labeled as the *biased approach* [24], as there is no distribution of inputs indication between the primary outputs. In other words, the primary outputs are not collectively responsible for acknowledging the arrival of all the primary inputs and internal outputs. Our proposed synthesis solution corresponds to a direct synthesis strategy and differs from the method presented in [24] in that the latter generates a dual-rail asynchronous gate pair or a delay-insensitive minterm synthesis (DIMS) equivalent [25] of each synchronous logic gate. In fact, the process of generating a dual-rail asynchronous gate pair for a synchronous logic gate is based on the dual-rail combinational logic style [26, 27]. The asynchronous dual-rail gate pair equivalent or the DIMS equivalent of each synchronous logic gate is eventually realized using proprietary null convention logic (NCL) macros [28], which are constructed on the basis of threshold logic [29].

Secondly, the full adder block depicted in Figure 4 features implicit logic redundancy. The intermediate gate output functions "*int1*" and "*int2*" are found embedded within the logic producing the carry outputs Cout0 and Cout1, respectively, however, in their input-incomplete forms. The principal advantages of this full adder with

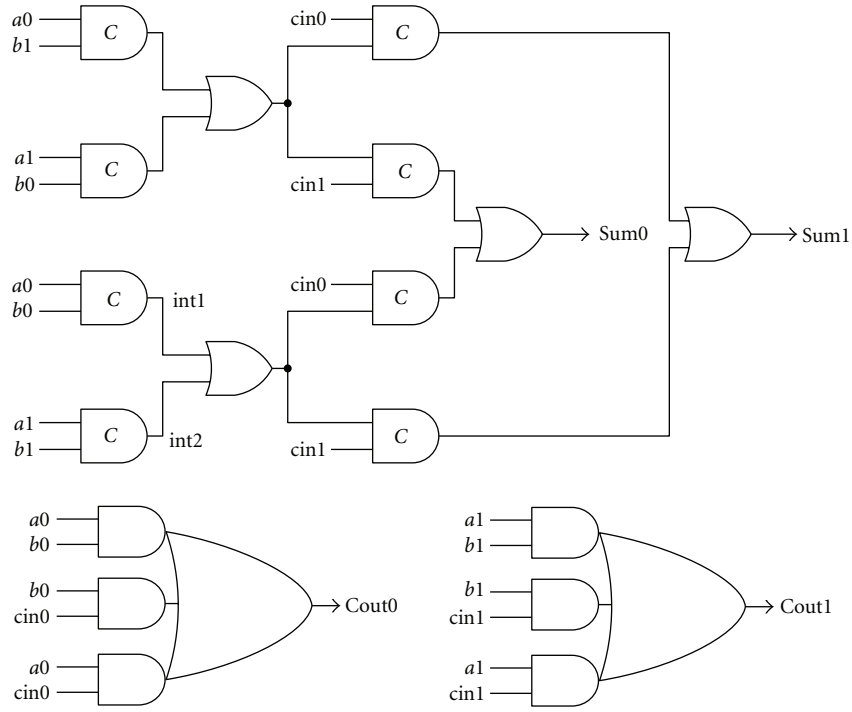

Figure 4: Proposed weak-indication full adder design (SSSC_DRE adder).

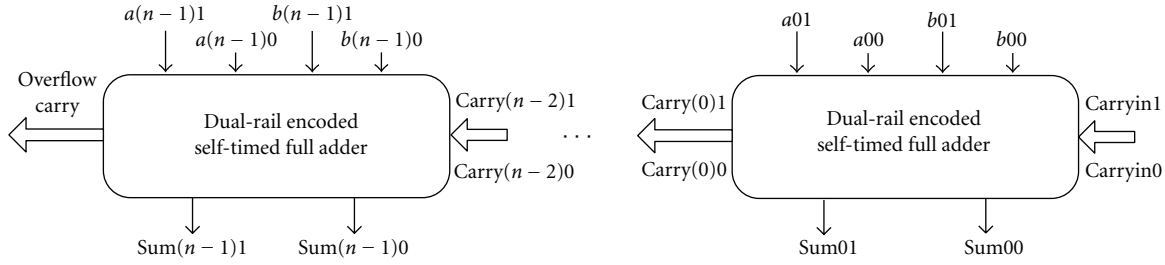

Figure 5: n-bit dual-rail encoded self-timed carry-ripple adder structure.

respect to the n-bit self-timed RCA architecture shown in Figure 5 are (i) fast carry output production and propagation when carry-generate ($a1 = b1 = 1$) and carry-kill ($a0 = b0 = 1$) conditions occur and (ii) reset of the entire adder circuitry with an approximate propagation delay of only two full adders during the return-to-zero phase regardless of the adder size. The latter advantage results from the fact that the intermediate dual-rail output carries of all the full adder modules connected in a cascade could be reset in parallel as the dual-rail encoded augend and addend inputs of every adder stage are reset. Subsequently, the dual-rail sum outputs of all the adder stages would be reset as their input carries assume a spacer state. This leads to a constant latency operation for application of spacer data, while data-dependent latency would manifest when valid data is applied. Indeed, this attribute becomes inherent in all the redundancy incorporated self-timed adders. The worst-case latency results when the carry-propagate mode is activated with respect to all the individual full adder stages; this happens for the case when $a1 = b0 = 1$ or $a0 = b1 = 1$. The SSSC_DRE adder has some similar properties as that of Martin's full adder [30], which is nevertheless a stand-alone full-custom transistor level realization.

2.2. Explicit Logic Redundancy. We now consider a variety of scenarios where logic redundancy is explicit in a circuit design. To this end, we analyze some adder circuits which employ a uniform DI data encoding protocol (dual-rail encoding) for both primary inputs and outputs, or a combination of DI codes (dual-rail and 1-of-4 codes) for primary inputs, but a single DI code (dual-rail code) for the primary outputs.

2.2.1. Single-Bit Adder Based on Hybrid Input Encoding. The term "hybrid input encoding" specifies a mix of at least two different DI data encoding schemes as adopted for the primary inputs. Considering the single-bit full adder block, the augend and addend input bits can be encoded using a 1-of-4 code, while the carry input, sum and carry outputs

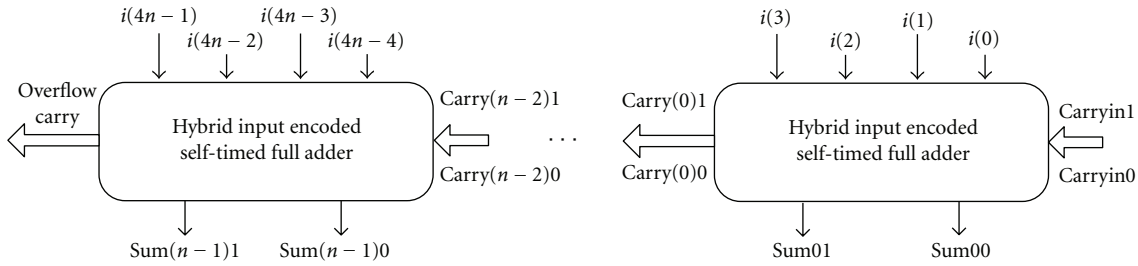

Figure 6: n-bit hybrid input encoded self-timed RCA configuration.

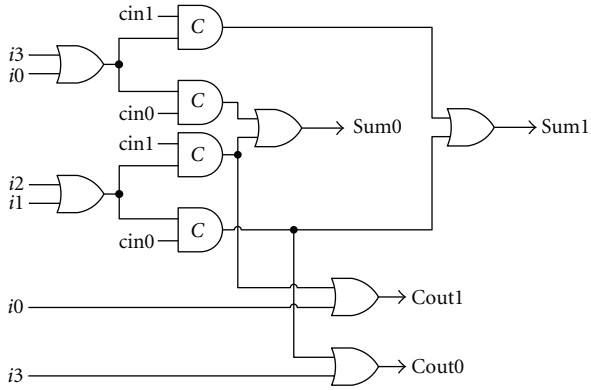

Figure 7: Hybrid input encoded full adder block (SSSC_HIE_NRL adder).

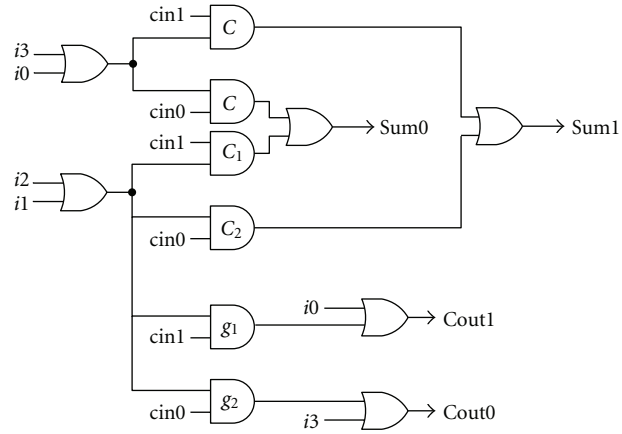

Figure 8: Hybrid input encoded full adder including redundancy (SSSC_HIE_RL adder).

can adopt the dual-rail code; that is, hybrid encoding of primary inputs and uniform encoding of primary outputs are resorted to. The structure of the n-bit hybrid input encoded self-timed RCA is depicted by Figure 6, which is similar to the topology shown in Figure 5 with the exception that the augend and addend single-rail inputs are now encoded using the 1-of-4 code.

The general expressions governing a full adder block utilizing hybrid input encoding for inputs and dual-rail encoding for outputs are given below. In the equations that follow, ($i0$, $i1$, $i2$, $i3$) represents the 1-of-4 encoded equivalent of the single-rail adder inputs (a, b), with a single-rail to 1-of-4 data representation scheme adopted as shown in Table 1:

$$\begin{aligned} \text{Sum1} &= i3\text{cin1} + i2\text{cin0} + i1\text{cin0} + i0\text{cin1}, \\ \text{Sum0} &= i3\text{cin0} + i2\text{cin1} + i1\text{cin1} + i0\text{cin0}, \\ \text{Cout1} &= i2\text{cin1} + i1\text{cin1} + i0\text{cin0} + i0\text{cin1}, \\ \text{Cout0} &= i3\text{cin0} + i3\text{cin1} + i2\text{cin0} + i1\text{cin0}. \end{aligned} \tag{2}$$

The full adder block that synthesizes equation (2) inclusive of carry output logic optimization is portrayed by Figure 7. Henceforth, this adder module shall be identified as the SSSC_HIE_NRL adder (single sum, single carry hybrid input encoded nonredundant logic adder). As the name implies, all the gates that constitute this adder are irredundant. It can be observed from Figure 7 that the sum outputs are entrusted with the responsibility of inputs indication, while the carry outputs could evaluate to the correct state whenever the carry-kill or carry-generate condition occurs without having to wait for the carry input. Thus the SSSC_HIE_NRL adder corresponds to the weak-indication timing model.

The synthesized hybrid input encoded full adder block that incorporates logic redundancy is shown in Figure 8.

Here, gates C_1 and C_2 denote 2-input C-elements, while gates g_1 and g_2 represent 2-input AND gates. It can be noticed in the figure that the functions realized by C_1 and C_2 are identical to that implemented by g_1 and g_2, respectively, for the case of upgoing transitions. Hence, redundancy is explicit in the present design, henceforth referred to as the SSSC_HIE_RL adder (single sum, single carry hybrid input encoded redundant logic adder). With respect to this adder design, logic redundancy is found to be beneficial in two ways. During the spacer phase, all the sum outputs could be reset in a parallel fashion, as the dual-rail carry output of the kth stage of a n-bit adder could be reset based on its 1-of-4 encoded augend and addend inputs, and the dual-rail sum output of the $(k + 1)$th adder stage would depend only on the dual-rail carry input of its preceding stage. There is also a benefit in terms of improving the computation speed during the valid data phase. This would be obvious by comparing the designs portrayed by Figures 7 and 8; it can be observed that the carry propagation delay is less in case of the SSSC_HIE_RL adder (AND2, OR2 gate delays) in

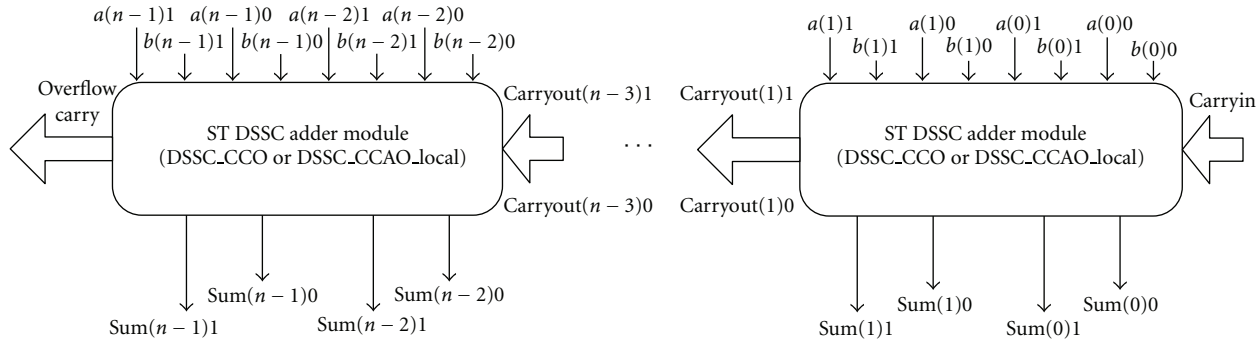

Figure 9: Dual-rail encoded n-bit RCA architecture comprising dual-bit adder blocks.

comparison with the SSSC_HIE_NRL adder (CE2, OR2 gate delays).

2.2.2. Dual-Bit Adder Utilizing Homogeneous Data Encoding. We now analyze the effect of introducing redundant logic in a self-timed dual-bit adder module that employs homogeneous data encoding for both its primary inputs and outputs. The homogeneous encoding procedure refers to a similar DI data encoding protocol as adopted for all the primary inputs and outputs of a function block—here dual-rail data encoding. The dual-bit adder block consists of dual-rail encoded versions of five single-rail inputs, namely, *a1*, *a0*, *b1*, *b0*, and cin, and three single-rail outputs Cout, Sum1 and Sum0, where (*a1*, *a0*) and (*b1*, *b0*) represent the addend and augend inputs and cin the carry input. The output Cout is the carry output or overflow bit of the addition process, and Sum1 and Sum0 signify the most significant and least significant sum output bits, respectively.

The reduced orthogonal sum-of-products forms corresponding to the encoded outputs of the dual-bit adder are given below, expressed in terms of their encoded inputs. In an orthogonal sum-of-products form, the logical conjunction of any pair of product terms yields a null:

$$\begin{aligned}
\text{Cout1} &= a10a00b11b01\text{cin1} + a11a00b10b01\text{cin1} \\
&\quad + a10a01b11b00\text{cin1} + a11a01b10b00cin1 \\
&\quad + a10a01b11b01 + a11a01b10b01 + a11b11, \\
\text{Cout0} &= a11a01b10b00\text{cin0} + a10a01b11b00\text{cin0} \\
&\quad + a11a00b10b01\text{cin0} + a10a00b11b01\text{cin0} \\
&\quad + a11a00b10b00 + a10a00b11b00 + a10b10, \\
\text{Sum11} &= a11a01b10b00\text{cin0} + a10a01b11b00\text{cin0} \\
&\quad + a11a00b10b01\text{cin0} + a10a00b11b01\text{cin0} \\
&\quad + a11a00b11b01\text{cin1} + a11a01b11b00\text{cin1} \\
&\quad + a10a00b10b01\text{cin1} + a10a01b10b00\text{cin1} \\
&\quad + a10a01b10b01 + a11a00b10b00 \\
&\quad + a10a00b11b00 + a11a01b11b01, \\
\text{Sum10} &= a11a01b10b00\text{cin1} + a10a01b11b00\text{cin1} \\
&\quad + a11a00b10b01\text{cin1} + a10a00b11b01\text{cin1} \\
&\quad + a10a01b10b00\text{cin0} + a10a00b10b01\text{cin0} \\
&\quad + a11a01b11b00\text{cin0} + a11a00b11b01\text{cin0} \\
&\quad + a11a00b11b00 + a11a01b10b01 \\
&\quad + a10a01b11b01 + a10a00b10b00, \\
\text{Sum01} &= a01b00\text{cin0} + a00b01\text{cin0} \\
&\quad + a00b00\text{cin1} + a01b01\text{cin1}, \\
\text{Sum00} &= a01b01\text{cin0} + a01b00\text{cin1} \\
&\quad + a00b01\text{cin1} + a00b00\text{cin0}.
\end{aligned} \tag{3}$$

The architecture of the n-bit self-timed carry-ripple adder structure that features dual-bit adder modules is shown in Figure 9. The synthesized dual-bit adder module is portrayed by Figure 10. It shall be referred to as the DSSC_DRE adder module (dual sum, single carry dual-rail encoded adder) in the subsequent discussions. Figure 10 depicts the redundant AND gates (shaded gates) inserted into the DSSC_DRE adder block. The nonredundant adder block would not feature the AND gates *rg1* and *rg2*, and so one of the inputs for the OR2 gates producing Cout1 and Cout0 would be the outputs of C-elements (C_1 and C_2), which are the nets labeled as gn2 and gn3, respectively. In fact, gn2 and gn3 would be isochronic forks in the nonredundant version. Isochronic forks are the weakest compromise to delay insensitivity [31], leading to quasidelay-insensitive (QDI) circuit implementations. According to the isochronicity assumption, if a transition on a wire fork is acknowledged, then the transition on the other wire forks is also said to be acknowledged. It was shown in [32] that QDI circuits which include isochronic fork assumptions can be realized in even nanometer scale dimensions. Indeed, QDI circuits are the practically implementable DI circuits, and they constitute the robust class of self-timed circuits. In the redundant dual-bit adder shown in Figure 10, the OR2 gates producing Cout1 and Cout0 have gn1 and gn4 feeding as

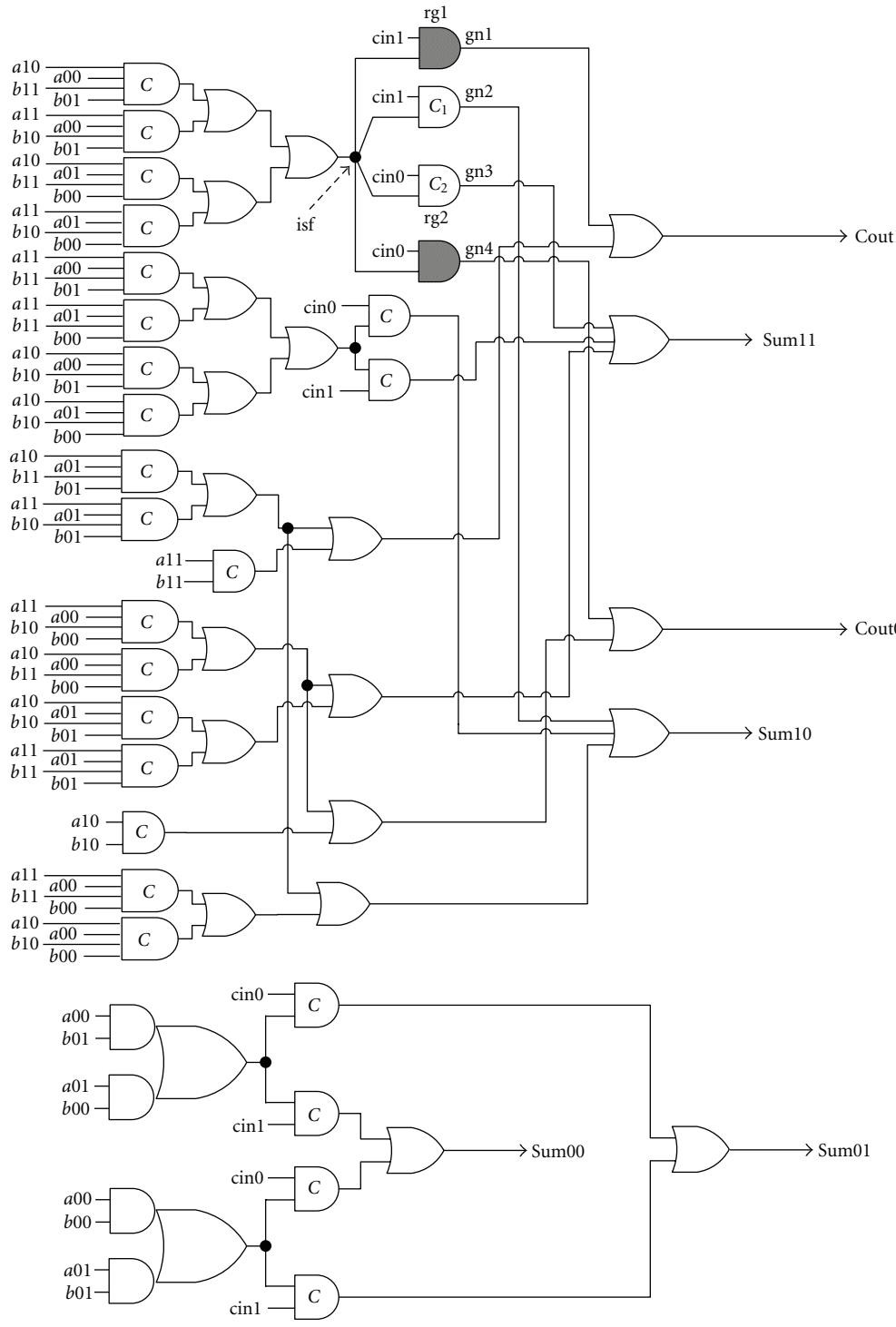

Figure 10: Redundant logic insertion in a homogeneously encoded dual-bit adder.

inputs, respectively. For the case of low-to-high transitions, the AND gates $rg1$ and $rg2$ are functionally equivalent to C-elements C_1 and C_2.

The gate output node labeled "*isf*" signifies an isochronic fork junction. Referring to Figure 10, it can be observed that an upgoing transition on the fork *isf* (*isf*↑) would be followed by either $gn2\uparrow$ or $gn3\uparrow$ in case of the nonredundant DSSC_DRE adder block and by ($gn1\uparrow$, $gn2\uparrow$) or ($gn3\uparrow$, $gn4\uparrow$) in case of the DSSC_DRE adder module that incorporates logic redundancy; this explains the possible

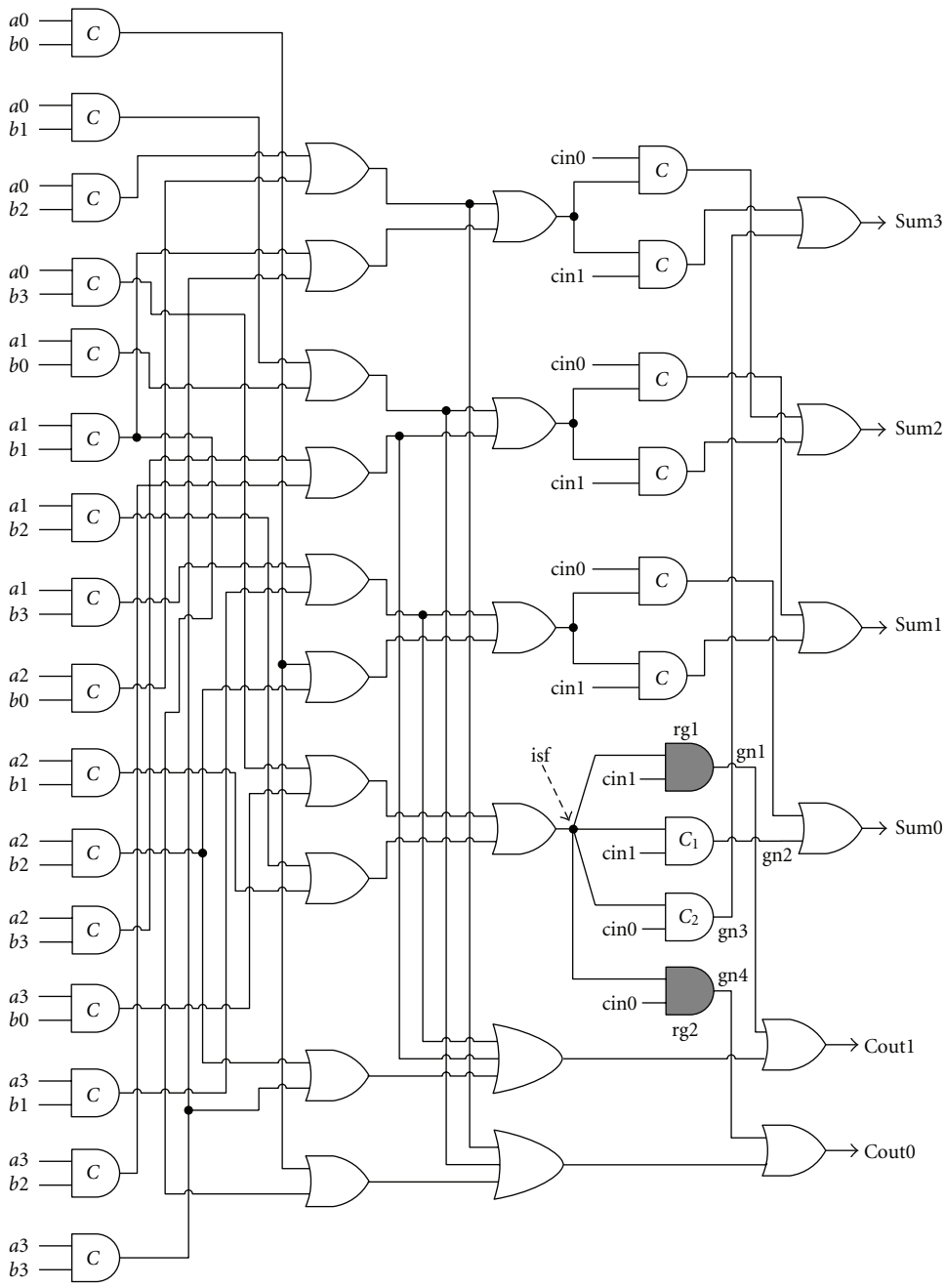

Figure 11: Weakly indicating heterogeneously encoded dual-bit adder module.

multiple acknowledgements. It can be observed that there is a possibility for fast or eager reset during the return-to-zero phase as a result of introducing logic redundancy into the adder. During the spacer phase, all the sum outputs could be reset in a parallel fashion, as the carry output of the previous dual-bit adder stage could be reset even by its corresponding augend and addend inputs without having to wait for an input carry from the preceding stage. The advantage of latency reduction gained by introduction of redundant logic is due to the lower data path delay encountered, as the critical path in every dual-bit adder stage contains input-incomplete gates instead of a mix of input-complete and input-incomplete gates as in the original nonredundant version.

2.2.3. Dual-Bit Adder Incorporating Heterogeneous Data Encoding. The heterogeneous encoding procedure implies a combination of at least two different DI codes (say, dual-rail

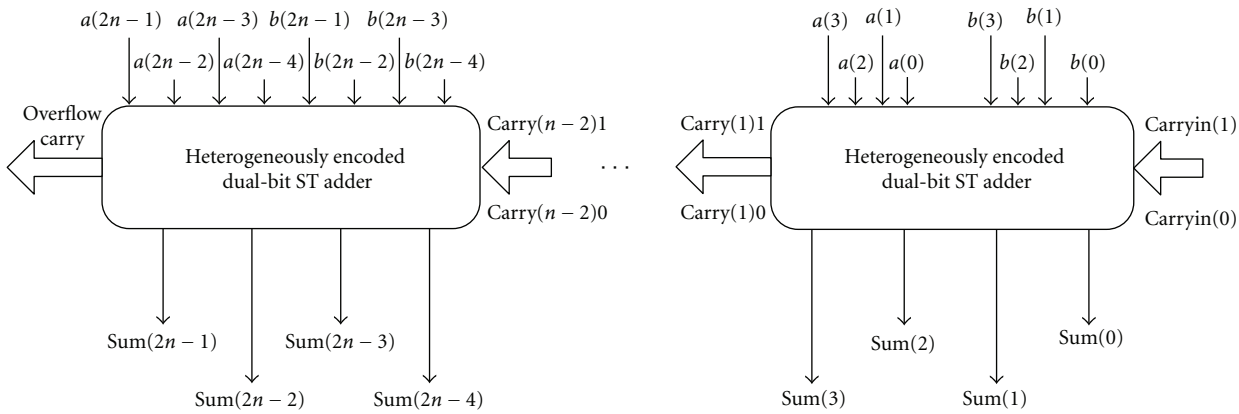

Figure 12: Heterogeneously encoded dual-bit adder based n-bit self-timed RCA.

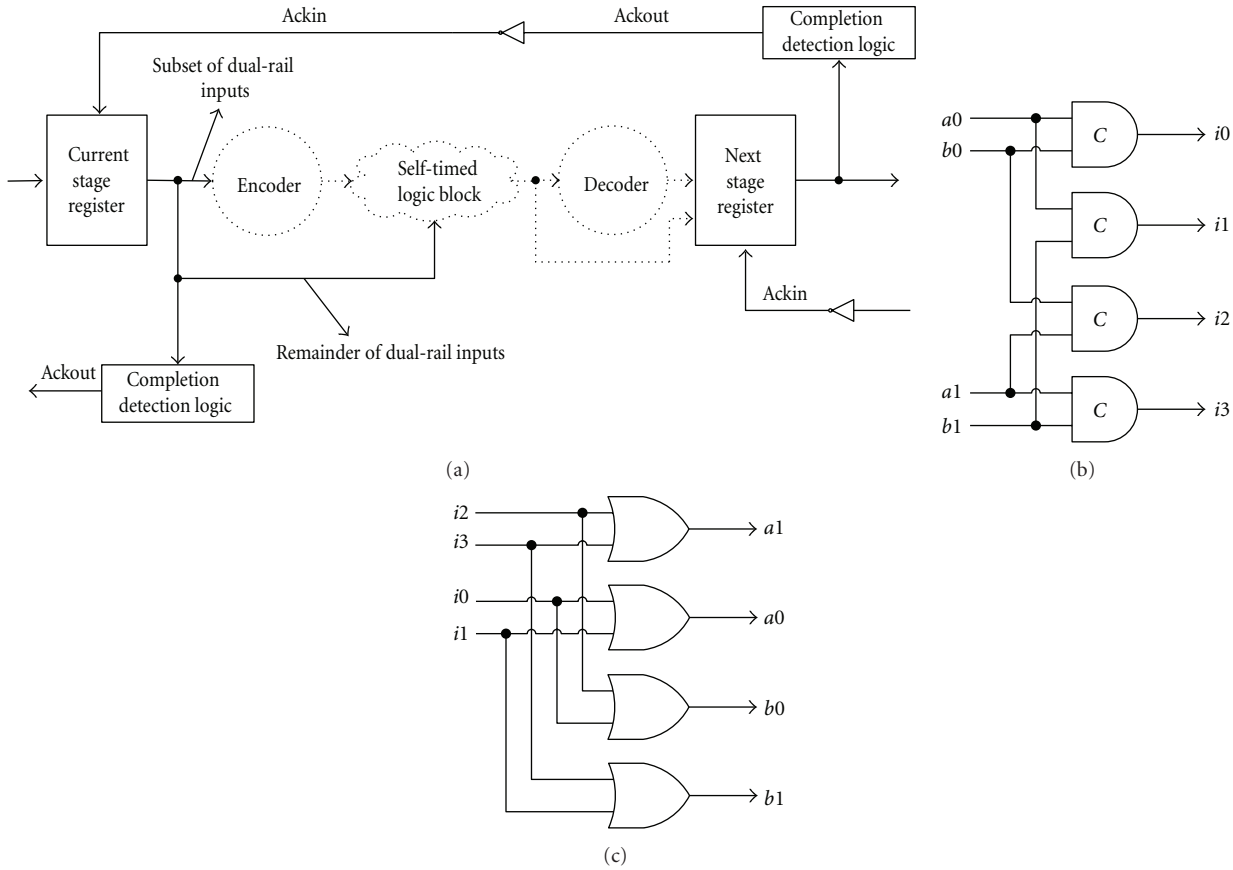

Figure 13: (a) Self-timed system handling heterogeneously encoded inputs and outputs, (b) dual-rail to 1-of-4 encoder, (c) 1-of-4 to dual-rail decoder.

and 1-of-4 codes), used to encode the primary inputs and outputs of a self-timed logic circuit. A dual-bit adder block based on heterogeneous DI data encoding can represent the augend, addend inputs, and sum outputs by a 1-of-4 code, while the input and output carry signals can be represented using the dual-rail code. Adopting such an encoding scheme, the minimized expressions for the function block outputs are given below. It is to be noted that the 1-of-4 code assignments for the augend, addend inputs, and the sum outputs are the reverse of the assignments given in Table 1:

$$\begin{aligned} \text{Cout1} = {} & a0b3\text{cin1} + a1b2\text{cin1} + a2b1\text{cin1} \\ & + a3b0cin1 + a1b3 + a2b2 + a3b1 \\ & + a2b3 + a3b2 + a3b3, \end{aligned} \tag{4}$$

$$\begin{aligned}\text{Cout0} = {} & a0b3\text{cin0} + a1b2\text{cin0} + a2b1\text{cin0} \\ & + a3b0\text{cin0} + a0b0 + a0b1 \\ & + a0b2 + a1b0 + a1b1 + a2b0,\end{aligned} \tag{5}$$

$$\begin{aligned}\text{Sum3} = {} & a0b3\text{cin0} + a1b2\text{cin0} + a2b1\text{cin0} \\ & + a3b0\text{cin0} + a0b2\text{cin1} + a1b1\text{cin1} \\ & + a2b0\text{cin1} + a3b3\text{cin1},\end{aligned} \tag{6}$$

$$\begin{aligned}\text{Sum2} = {} & a0b2\text{cin0} + a1b1\text{cin0} + a2b0\text{cin0} \\ & + a3b3\text{cin0} + a0b1\text{cin1} + a1b0\text{cin1} \\ & + a2b3\text{cin1} + a3b2\text{cin1},\end{aligned} \tag{7}$$

$$\begin{aligned}\text{Sum1} = {} & a0b1\text{cin0} + a1b0\text{cin0} + a2b3\text{cin0} \\ & + a3b2\text{cin0} + a0b0\text{cin1} + a1b3\text{cin1} \\ & + a2b2\text{cin1} + a3b1\text{cin1},\end{aligned} \tag{8}$$

$$\begin{aligned}\text{Sum0} = {} & a0b0\text{cin0} + a1b3\text{cin0} + a2b2\text{cin0} \\ & + a3b1\text{cin0} + a0b3\text{cin1} + a1b2\text{cin1} \\ & + a2b1\text{cin1} + a3b0\text{cin1}.\end{aligned} \tag{9}$$

The dual-bit adder module that synthesizes (4)–(9) is shown in Figure 11. Henceforth, this adder shall be referred to as the DSSC_HE adder (dual sum, single carry heterogeneously encoded adder). The DSSC_HE adder block satisfies the weak-indication timing constraints. The 1-of-4 encoded sum outputs assume responsibility for indicating the arrival of all the adder inputs, while the dual-rail encoded carry output can be relaxed with respect to ensuring input completeness. Logic redundancy, as introduced into the DSSC_HE adder module, is shown in the figure with the input-incomplete AND gates (shaded gates) marked as *rg1* and *rg2*. Similar notations have been used as that of Figure 10 so that the discussions of the previous section would hold well for this scenario too. As in the earlier case, the sum output(s) of the $(i+1)$th dual-bit adder stage could be reset based on the carry input from the ith dual-bit adder stage, and there does not arise any need for resetting of the entire carry chain during the return-to-zero phase.

The n-bit self-timed carry-ripple adder architecture that encompasses heterogeneously encoded dual-bit adder modules is shown in Figure 12. The self-timed system configuration that supports the RCA topology is depicted in Figure 13. A subset of the dual-rail inputs (augends and addends) is 1-of-4 encoded before being fed to the function block for data processing, while the remaining inputs (dual-rail encoded input carry) are fed as such. The non-dual-rail outputs produced by the logic block (sum outputs) are decoded before being passed onto the next stage, while the dual-rail outputs (output carry) are driven to the next stage. The encoding and decoding costs equate to 28 and 12 transistors per bit, respectively.

3. Simulation Mechanism and Results

To demonstrate the usefulness of the proposed concept of logic redundancy insertion, simulations have been performed by considering a 32-bit self-timed RCA architecture. In this context, a subset of well-known self-timed design methods [25, 33, 34] is considered in this work. Various 32-bit self-timed RCAs were built by considering different adder building blocks—32 single-bit adder blocks or 16 dual-bit adder modules. Before discussing the simulation results, the mechanism of estimating the design metrics is elucidated. The delay parameter refers to the maximum propagation delay (critical path delay) encountered in the data path, which is a sum of the latencies of the input register and that of the combinatorial adder logic. The delay metric was estimated using PrimeTime. To avoid the notion of a clock source, a virtual clock was used as a remote reference to constrain the input and output ports of the design. The area and power metrics correspond to the input registers, completion detection logic, and the 32-bit combinatorial adder. The delay and power metrics consider estimated parasitics in addition to the parameters associated with actual components (gates). The area metric gives a combined account of the area of all the logic cells. The total/average power dissipation is the sum of dynamic and static power components, where dynamic power is in turn composed of switching and internal power consumption values. NC-Sim has been used for functional simulation and also to obtain the switching activity files corresponding to gate-level simulations of Verilog descriptions of various 32-bit self-timed adders. Input data were supplied to the adders at a time interval of 15 ns through a random test bench which models the environment. The switching activity files obtained were subsequently used for power estimation using PrimeTime PX. The simulations targeted a PVT corner of the 130 nm bulk CMOS standard cell library whose recommended supply voltage is 1.32 V and the ambient junction temperature is -40°C. All the circuit inputs possess the driving strength of the minimum-sized inverter of the cell library, while the outputs are associated with a fanout-of-4 drive strength. Appropriate buffering for the input acknowledgement signal was provided where necessary to eliminate timing violations. Since identical registers and a similar completion detection circuit were used for all the 32-bit adders, the area and power metrics can be correlated with that of the function block, thus paving the way for a straightforward comparison between adders synthesized on the basis of different self-timed design methods. Strong/weak-indication adders corresponding to various self-timed design methods were constructed manually and were subsequently optimized for minimum latency by taking into account the physical constraints of the target cell library. (A 130 nm CMOS standard cell library was used. The maximum fan-in of AND gate and OR gate in this library is 4 and 3, respectively. The granularity of the C-element ranges from 2 to 4 inputs, and the gate level C-element models are given in [35]). The delay, area, and power metrics corresponding to the simulations of various nonredundant 32-bit self-timed RCAs are given in Table 2.

Table 2: Delay, area, and power of various nonredundant 32-bit self-timed RCAs.

Adder realization style	Delay (ns)	Area (μm^2)	Power (μW)
SSSC_HIE_NRL (weak)	8.0	6633 (78)	619.1
DIMS_DSSC_DRE (weak)* [25]	12.8	21833 (1202)	1025.9
Toms_DSSC_DRE (strong) [33]	9.4	10793 (512)	693.1
Toms_DSSC_HE (strong) [33]	9.0	12121 (479)	695.9
Folco et al._DSSC_DRE (weak) [34]	5.9	9417 (426)	740.4
DSSC_DRE (weak)	5.9	14921 (770)	871.9
DSSC_HE (weak)	5.8	10889 (402)	688.4

*The dual sum, single carry (DSSC) adder realization based on the DIMS method required careful speed-independent logic decomposition to decompose the high fan-in C-gates.

Table 3: Delay, area, and power metrics of various redundant 32-bit self-timed RCAs.

Adder realization style	Delay (ns)	Area (μm^2)	Power (μW)
SSSC_HIE_RL (weak)	5.9	6953 (88)	630.2
DIMS_DSSC_DRE (weak) [25]	10.5	22473 (1242)	1034.6
Folco et al._DSSC_DRE (weak) [34]	4.7	9577 (436)	743.8
DSSC_DRE (weak)	4.6	15081 (780)	875.3
DSSC_HE (weak)	4.6	11049 (412)	691.9

The nature of indication of the different adders is mentioned within brackets in the 1st column of the Table. The values specified within brackets in the 3rd column of the table signify the area of the respective individual single-bit/dual-bit self-timed adder block. The delay, area and power parameters of the different redundant logic incorporated 32-bit self-timed RCAs are given in Table 3. Introduction of logic redundancy into the dual-bit adder module synthesized on the basis of Toms and Edwards, approach [33] was not considered, since it would change the indication property of the original synthesis solution. Therefore, redundant versions of other adders were alone considered for comparison in Table 3. By comparing the results given in Tables 2 and 3, it is found that logic redundancy insertion has enabled a mean delay reduction of 21.1%, with associated area and power penalties to the tune of 2.3% and 0.8%, respectively. On average, the increase in size of an individual self-timed single-bit/dual-bit adder module is found to be 2.8% after incorporating redundant logic.

With reference to the DSSC_HE adder module shown in Figure 11, a further peephole optimization was carried out by merging the gate *rg1* and the OR gate producing Cout1 and *rg*2 and the OR gate producing Cout0 and replacing these combinations using complex gates (AO12 cells). Simulations were repeated for this case study, and the delay, area, and power values corresponding to the 32-bit RCA, comprising optimized redundant DSSC_HE adder blocks, are found to be 4 ns, 10953 μm^2, and 696.8 μW respectively. The optimized redundant DSSC_HE adder block occupies less area than the nonoptimized redundant DSSC_HE adder block by 1.5%. Hence, the 32-bit self-timed RCA comprising a cascade of optimized redundant DSSC_HE adder blocks exhibits reduced delay in comparison with the nonredundant DSSC_HE adder module based 32-bit self-timed RCA by 31%. However, in terms of area and average power, the latter features reduced figures to the tune of 0.6% and 1.2%, respectively, compared to the former.

4. Conclusions

A new concept of redundant logic insertion was described in this paper that can be used to minimize the data path delay of self-timed arithmetic circuits. It was shown that introduction of logic redundancy is feasible with respect to many self-timed design methods, especially for synthesizing iterative logic specifications. The advantages of logic redundancy insertion have been propounded on the basis of a 32-bit self-timed carry-ripple addition. It has been inferred from the simulation results that significant reduction in latency could be achieved at the expense of only marginal increases in area and power metrics. It was also discussed how logic redundancy paves the way for constant latency operation by permitting fast reset when applying spacer data, while actual case latency is encountered for addition of valid data.

Acknowledgments

This research was supported in part by the Engineering and Physical Sciences Research Council, UK, under Grant EP/D052238/1. The first author was additionally supported by a bursary from the School of Computer Science of the University of Manchester.

References

[1] "Semiconductor Industry Association's International Technology Roadmap for Semiconductors," Design Report, 2009, http://www.itrs.net/.

[2] A. J. Martin, S. M. Burns, T. K. Lee, D. Borkovic, and P. J. Hazewindus, "The first asynchronous microprocessor: the test results," *ACM SIGARCH Computer Architecture News*, vol. 17, no. 4, pp. 95–98, 1989.

[3] 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 (ISCAS '08)*, pp. 3190–3193, May 2008.

[4] 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, no. 2, pp. 401–410, 2010.

[5] M. Zamani and M. B. Tahoori, "A transient error tolerant self-timed asynchronous architecture," in *Proceedings of the 15th IEEE European Test Symposium (ETS'10)*, pp. 88–93, May 2010.

[6] J. Hamon and L. Fesquet, "Robust and programmable self-timed ring oscillators," in *Proceedings of the IEEE 9th International New Circuits and Systems Conference (NEWCAS '11)*, pp. 249–252, 2011.

[7] T. Chelcea, G. Venkataramani, and S. C. Goldstein, "Area optimizations for dual-rail circuits using relative-timing analysis," in *Proceedings of the 13th IEEE International Symposium on Asynchronous Circuits and Systems*, pp. 1–12, 2007.

[8] 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 Electro-Magnetic Compatibility of Integrated Circuits*, pp. 27–31, 2004.

[9] W. A. Lien, P. Day, C. Farnsworth et al., "Noise in self-timed and synchronous implementations of a DSP," in *Proceedings of the IEEE Radio and Wireless Conference*, pp. 75–78, 1998.

[10] L. S. Nielsen, C. Niessen, J. Sparso, and K. van Berkel, "Low-power operation using self-timed circuits and adaptive scaling of the supply voltage," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, no. 4, pp. 391–397, 1994.

[11] 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 (ASYNC '10)*, pp. 41–51, May 2010.

[12] C. H. Van Kees Berkel, M. B. Josephs, and S. M. Nowick, "Scanning the technology applications of asynchronous circuits," *Proceedings of the IEEE*, vol. 87, no. 2, pp. 223–233, 1999.

[13] I. David, R. Ginosar, and M. Yoeli, "Self-timed is self-checking," *Journal of Electronic Testing*, vol. 6, no. 2, pp. 219–228, 1995.

[14] S. J. Piestrak and T. Nanya, "Towards totally self-checking delay-insensitive systems," in *Proceedings of the 25th International Symposium on Fault-Tolerant Computing*, pp. 228–237, June 1995.

[15] T. Verhoeff, "Delay-insensitive codes—an overview," *Distributed Computing*, vol. 3, no. 1, pp. 1–8, 1988.

[16] B. Bose, "On unordered codes," *IEEE Transactions on Computers*, vol. 40, no. 2, pp. 125–131, 1991.

[17] D. W. Lloyd and J. D. Garside, "A practical comparison of asynchronous design styles," in *Proceedings of the 7th International Symposium on Asynchronous Circuits and Systems, (ASYNC '01)*, pp. 36–45, March 2010.

[18] W. J. Bainbridge, W. B. Toms, D. A. Edwards, and S. B. Furber, "Delay-insensitive, point-to-point interconnect using M-of-N codes," in *Proceedings of the 9th IEEE International Symposium on Asynchronous Circuits and Systems, (ASYNC '03)*, pp. 132–140, May 2003.

[19] V. Akella, N. H. Vaidya, and G. R. Redinbo, "Limitations of VLSI implementation of delay-sensitive codes," in *Proceedings of the 1996 26th International Symposium on Fault-Tolerant Computing*, pp. 208–217, June 1996.

[20] C. L. Seitz, "System timing," in *Introduction to VLSI Systems*, C. Mead and L. Conway, Eds., pp. 218–262, Addison-Wesley, Reading, Mass, USA, 1980.

[21] J. Sparso and S. B. Furber, Eds., *Principles of Asynchronous Circuit Design: A Systems Perspective*, Kluwer Academic Publishers, 2001.

[22] P. Balasubramanian and D.A. Edwards, "Self-timed realization of combinational logic," in *Proceedings of the 19th International Workshop on Logic and Synthesis*, pp. 55–62, 2010.

[23] 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 (DDECS '08)*, pp. 116–121, April 2008.

[24] 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 (ASYNC '08)*, pp. 95–104, April 2008.

[25] J. Sparsø and J. Staunstrup, "Delay-insensitive multi-ring structures," *Integration, The VLSI Journal*, vol. 15, no. 3, pp. 313–340, 1993.

[26] "Aperiodic circuits," in *Self-Timed Control of Concurrent Processes: The Design of Aperiodic Logical Circuits in Computers and Discrete Systems*, V. I. Varshavsky, Ed., chapter 4, pp. 77–85, Kluwer Academic Publishers, 1990.

[27] C. D. Nielsen, "Evaluation of function block designs," Tech. Rep. ID-TR: 1994-135, Department of Computer Science, Technical University of Denmark, 1994.

[28] K. M. Fant and G. E. Sobelman, "Null convention threshold gate," US Patent 5664211, 1997.

[29] P. M. LewisII and C. L. Coates, *Threshold Logic*, Wiley, New York, NY, USA, 1967.

[30] A. J. Martin, "Asynchronous datapaths and the design of an asynchronous adder," *Formal Methods in System Design*, vol. 1, no. 1, pp. 117–137, 1992.

[31] A.J. Martin, "The limitation to delay-insensitivity in asynchronous circuits," in *Proceedings of the 6th MIT Conference on Advanced Research in VLSI*, pp. 263–278, 1990.

[32] A. J. Martin and P. Prakash, "Asynchronous nano-electronics: preliminary investigation," in *Proceedings of the 14th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC '08)*, pp. 58–68, gbr, April 2008.

[33] W. B. Toms and D. A. Edwards, "Efficient synthesis of speed independent combinational logic circuits," in *Proceedings of the 10th Asia and South-Pacific Design Automation Conference*, pp. 1022–1026, 2005.

[34] 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 VLSI-SoC*, pp. 146–151, 2005.

[35] P. Balasubramanian, *Self-timed logic and the design of self-timed adders*, Ph.D. thesis, The University of Manchester, 2010.