Low Power Self-Timed Carry Lookahead Adders

P. Balasubramanian, D. Dhivyaa, J. P. Jayakirthika, P. Kaviyarasi
Department of Electronics and Communication Engineering
S. A. Engineering College, Chennai 600 077, India
dr.bala@saec.ac.in

K. Prasad
Department of Electrical and Electronic Engineering
Auckland University of Technology
Auckland 1142, New Zealand
krishnamachar.prasad@aut.ac.nz

***Abstract*—Cell based self-timed synthesis of recursive carry lookahead adders (RCLA) utilizing generate, propagate and kill functions is described in this paper, and are compared with the recently proposed designs of self-timed section-carry based carry lookahead (SCBCLA) adders. From the simulation results corresponding to a 130nm CMOS process, it is found that with 2-bit CLA, the RCLA adder dissipates 20.2% less power than the SCBCLA adder. With 4-bit CLA, the RCLA adder reports power reduction by 16.5% than the SCBCLA adder. Further, for addition widths ranging from 32 to 64-bits, RCLA adders consume 19% less average power compared to SCBCLA adders.**

I. INTRODUCTION

With successful nanometer scale IC designs getting rolled out from semiconductor fabs in huge volumes every year, and with the continuous venture into deeper nanoscale device geometries, the semiconductor industry is contemplating several options to push the limits of conventional digital IC design in terms of devices, dielectric materials, interconnects, foundry processes, fabrication methods, testing techniques, and manufacturing and packaging technologies. Given the aggressive technological trend fuelled by an ever-increasing market demand for mobile and portable electronic products, the Semiconductor Industry Association's 2011 International Technology Roadmap on Semiconductors (ITRS) report [1] has identified 'design for reliability' as one of the long-term grand challenges. Indeed, taking cognizance of decreasing feature sizes and associated increases in variability of devices, the ITRS report [1] mentions that the issue of 'reliability' could assume comparable significance with quality-of-results in the nanometer regime. In this context, the self-timed design paradigm is pegged to be a strong contender and a viable alternative to mainstream synchronous design style for implementing digital logic functionality such as arithmetic and logic units, circuits used in telecommunications, defense and security applications, and subsystems deployed in a wide range of industrial and consumer electronics. The primary motivation for adopting the self-timed design style arises from the fact that self-timed circuits consume power only when and where active, absorb process, temperature and parametric variations with ease, feature greater modularity, and inherently possess good noise and EMI tolerance capabilities [5].

II. SELF-TIMED CIRCUITS - BACKGROUND

Self-timed circuits actually pertain to a robust class of asynchronous design styles viz. input/output mode circuits, which do not exhibit any timing correlations with the external environment. Basically, there are three classes of self-timed circuits: delay-insensitive, quasi-delay-insensitive, and speed-independent. Delay-insensitive circuits employ an unbounded timing model for both components (gates) and wires, while quasi-delay-insensitive circuits offer a relaxation by permitting 'isochronic forks' – the weakest compromise to delay-insensitivity. The isochronic fork assumption implies that if a transition at one end of a wire fork is acknowledged, the transition(s) at the remaining fork end(s) are also assumed to be acknowledged – reference [2] demonstrates the validity of the isochronicity assumption at nanoscale dimensions. The speed-independent class prescribes unbounded delay model for components and a zero delay model for wires. Since wire delays can be accounted for in the component delays, speed-independent circuits tend to be synonymous with quasi-delay-insensitive circuits. The arithmetic circuits discussed in this work are designed in robust asynchronous style.

Self-timed combinational circuits also called 'function blocks', are generally implemented in conformance with Seitz's strong or weak indication timing constraints [3]. Strong-indication requires the function block to wait for the arrival of all the inputs before producing the requisite outputs. Weak-indication implies the function block is free to produce all but one primary output until the last primary input has arrived, i.e. with the exception of one output, all other outputs can be produced based on a subset of inputs. Early output logic [9] further relaxes strong and weak-indication constraints by permitting early set or reset, i.e., all the outputs may be produced (data/spacer) based on even a subset of the inputs (data/spacer). However, early propagative circuits have to be robust in order to qualify as being quasi-delay-insensitive [10]. As a norm, function blocks apart from having to produce desired output(s) for the given input(s) have the additional responsibility of indicating (acknowledging) completion of computation at all the internal nodes, and no dangling inputs should be present. Function blocks, in general, are endowed with the ability to indicate completion of data processing by

reason of adopting delay-insensitive codes and by employing a 4-phase handshaking mechanism [5]. Reference [4] discusses various delay-insensitive *m*-of-*n* codes, where *m* wires out of a total of *n* wires are asserted high to represent distinct single-rail data – among these, the dual-rail or 1-of-2 code is widely preferred owing to its simplicity and ease of mapping with binary data [5]. For example, a data wire d is encoded using two data wires d^1 and d^0, where $d = 1$ is represented by $d^1 = 1$ and $d^0 = 0$, and $d = 0$ is specified by $d^1 = 0$ and $d^0 = 1$. Both these combinations imply valid data, while $d^1 = d^0 = 0$ is referred to as the spacer data. As per 4-phase handshaking, application of input data should follow the alternate spacer protocol (*data-spacer-data*) that guarantees robustness [5].

III. Self-Timed Carry Lookahead Adders

The CLA adder is a fast carry-propagate adder which reduces carry propagation time to a logarithmic order by predicting future carries based on a priori knowledge of the input signals, instead of ascertaining carries on a stage-by-stage basis unlike the ripple carry adder (RCA). Reference [6] presents the ultimate design of a weakly indicating full adder which is cascaded in order to realize a delay-insensitive RCA; nevertheless the full adder is a custom transistor level design. Reference [7] deals with the design of a self-timed CLA adder from a circuit-level perspective – this article discusses about the design of a delay-insensitive CLA adder with and without extra speed-up circuitry. However, these self-timed CLA modules and adders have been designed manually and are not borne out of a synthesis scheme. On the contrary, our focus is on semi-custom design (direct synthesis) of self-timed CLA adders which are physically implemented using standard cells, with an eye on timing optimization. In this context, reference [8] deals with the design of self-timed CLA adders based on the notion of section-carry, where intra-section carries are allowed to ripple within an adder group, while inter-section carries are generated via lookahead.

In this work, two self-timed CLA adder structures are considered based on generate, propagate and kill signals viz. 'Regular' and 'Mixed'. 'Regular CLA adders' imply a routine cascade of CLA blocks of similar and/or varying sizes, while 'Mixed CLA adders' mean a hybrid architecture incorporating both CLA modules and RCA sections, with CLA logic being predominant – the use of a RCA in the least significant stages helps to improve the timing. In this paper, both 'regular' and 'mixed' versions of self-timed SCBCLA and RCLA adders are considered to perform a comparative analysis. The former are weakly indicating while the latter are early propagative.

A. Self-Timed RCLA Adder

With (a_i^1, a_i^0), (b_i^1, b_i^0), (C_{i-1}^1, C_{i-1}^0) representing the augend, addend and input carries of a dual-rail encoded i^{th} adder stage, and with (Sum_i^1, Sum_i^0), (C_i^1, C_i^0) representing the encoded sum and output carries, the governing equations are,

$$Sum_i^1 = P_i^0 C_{i-1}^1 + P_i^1 C_{i-1}^0 \quad (1)$$

$$Sum_i^0 = P_i^0 C_{i-1}^0 + P_i^1 C_{i-1}^1 \quad (2)$$

$$C_i^1 = G_i + P_i^1 C_{i-1}^1 \quad (3)$$

$$C_i^0 = K_i + P_i^1 C_{i-1}^0 \quad (4)$$

Where $P_i^1 = a_i^0 b_i^1 + a_i^1 b_i^0$, $P_i^0 = a_i^0 b_i^0 + a_i^1 b_i^1$; $G_i = a_i^1 b_i^1$; and $K_i = a_i^0 b_i^0$ denote the respective dual-rail propagate, generate and kill functions.

Based on (1) – (4), the generic structure of a *m*-bit self-timed CLA adder incorporating generate, propagate and kill signals is envisaged as shown in Figure 1(a). Complex gates (AO22 cells) are used to produce propagate signals, while 2-input AND gates (AND2 cells) are used to obtain generate and kill signals. To realize the sum outputs, the products are implemented using C-elements[1] to ensure proper indication of signal events. The RCLA module and adder both correspond to 'early output logic' [9] [10], i.e., the dual-rail carry and sum outputs may be reset in an eager fashion (eager reset) based on a subset of the inputs becoming spacers, while early set does not occur. Early output logic could generally help in reducing area, delay and/or power metrics [9] [10] [12]. The completion detection circuit [5] [10], present before the combinational adder logic, is assigned the responsibility of indicating the arrival of all the primary inputs, and hence isochronic fork assumptions are made with respect to all the primary inputs. This ensures that the adder's sum and carry outputs are always produced in a robust fashion i.e. devoid of circuit orphans.

Equations (1) – (4) are orthogonal in the sense that the conjunction of their respective product terms results in a 'null' – this condition satisfies the monotonic cover constraint [5], which is an important criterion characteristic of robust function blocks that facilitates unambiguous indication of signal transitions. The carry output equations are implicitly recursive and unwinding the recursion leads to the following expressions for a 3-bit dual-rail encoded CLA module.

$$C_0^1 = G_0 + P_0^1 C_{-1}^1 \quad (5)$$

$$C_0^0 = K_0 + P_0^1 C_{-1}^0 \quad (6)$$

$$C_1^1 = G_1 + P_1^1 G_0 + P_1^1 P_0^1 C_{-1}^1 \quad (7)$$

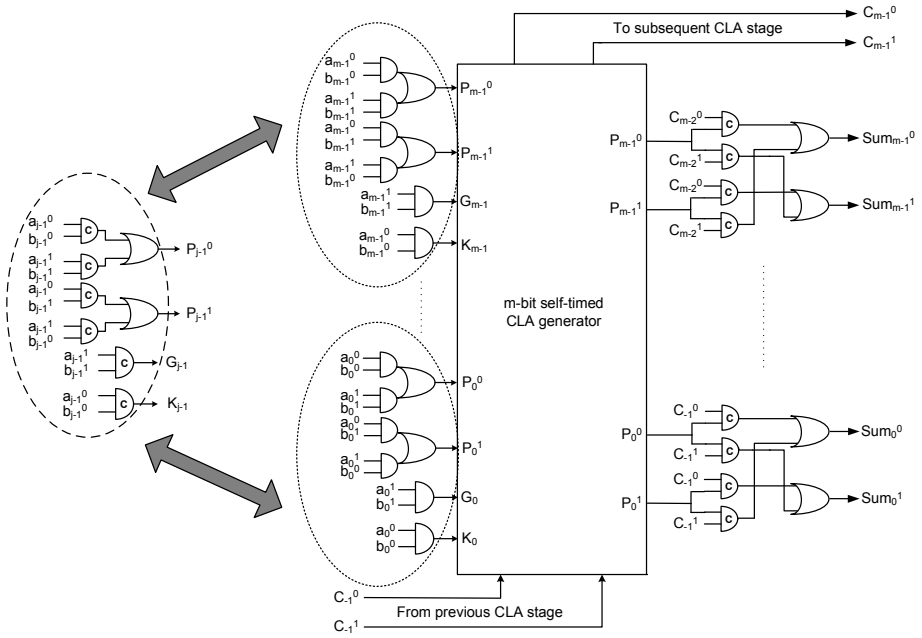
$$C_1^0 = K_1 + P_1^1 K_0 + P_1^1 P_0^1 C_{-1}^0 \quad (8)$$

$$C_2^1 = G_2 + P_2^1 G_1 + P_2^1 P_1^1 G_0 + P_2^1 P_1^1 P_0^1 C_{-1}^1 \quad (9)$$

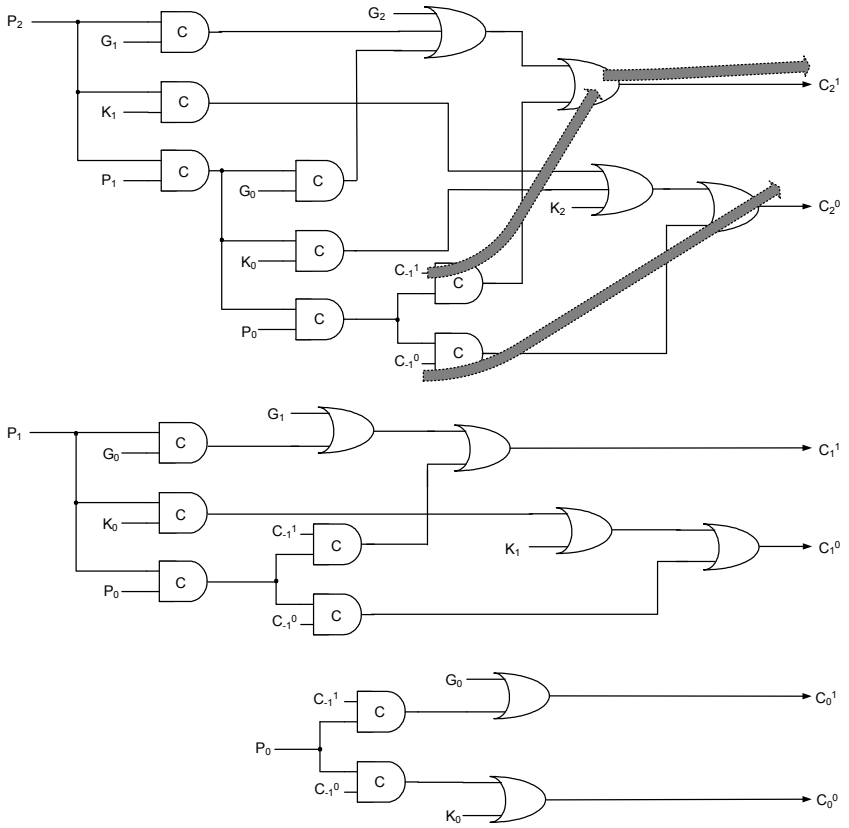
$$C_2^0 = K_2 + P_2^1 K_1 + P_2^1 P_1^1 K_0 + P_2^1 P_1^1 P_0^1 C_{-1}^0 \quad (10)$$

Equations (5) – (10), if synthesized directly, would require C-elements with a maximum fan-in of 4, and the fan-in requirement would eventually increase with increase in the size of lookahead. Hence, logic decomposition is necessary and a careful gate-orphan-free decomposition scheme [11] has been adopted to synthesize the 3-bit CLA module as shown in Figure 1(b). Any unacknowledged gate output transition is construed to be a gate orphan, which is to be avoided for proper circuit operation. Similarly, an unacknowledged wire transition called as wire orphan should also be eliminated. The avid reader is directed to [10] [12] for an explanation of gate and wire orphans; the details are omitted here for brevity. The critical (carry) propagation path in the CLA modules is represented by either of the curved arrows in Figure 1(b).

[1] The C-element produces a 1(0) if all its inputs are 1(0); otherwise it retains its existing state. The C-element inherently has memory and hence is used both as function element as well as register. A C-element with inputs *a* and *b*, and output *y* implements the function: $y = ab + ay + by$. The C-element is symbolized through a AND gate with the marking 'C' in the Figures.



(a) Self-timed CLA adder architecture employing dual-rail generate, propagate and kill signals
Note: If the actual generate, propagate and kill logic of the m-bit CLA block is replaced by the circuit shown on the left-hand-side containing C-elements and OR gates, the RCLA module and adder would be weakly indicating. Otherwise, as such, they correspond to early output logic. The bi-directional arrows signify logical equivalence.



(b) An example 3-bit self-timed CLA generator module synthesized using standard cells

Figure 1. Recursive self-timed CLA adder architecture involving dual-rail propagate, generate and kill functions, and a 3-bit decomposed RCLA module synthesized using standard cells, with the critical path(s) highlighted using curved arrow(s)

B. Simulation Method and Results

The simulation method is explained first before presenting the results. Delay signifies the maximum propagation delay of the datapath. Area represents the combined area of all the standard cells. Power signifies the total of dynamic and static components, where dynamic power is in turn the sum of switching and internal power consumption. Delay, area and power figures were estimated using Synopsys tool. The delay and power parameters evaluated consider estimated parasitics apart from the measured values of actual gates, which are listed in the 130nm CMOS library file. Random input data were supplied to the adders at nominal time intervals of 15ns through a test bench that models the environment.

TABLE 1. DELAY, AREA AND POWER DISSIPATION VALUES OF VARIOUS 32-BIT SELF-TIMED CLA ADDERS

Self-timed CLA adder type	Delay (ns)	Area (μm^2)	Power (μW)
Previous work			
DIMS (in Type 2 SCBCLA) (2-bit CLAs & RCA)	14.8	24273	1208.8
Toms *et al*. (in Type 2 SCBCLA) (2-bit CLAs & RCA)	10.1	13479	906.1
Type 2 SCBCLA (2-bit CLAs & RCA)	5.3	**8887**	**749.7**
Type 2 SCBCLA (4-bit CLAs & RCA)	4.0	9385	757.3
Mixed SCBCLA (3-bit, 4-bit CLAs & RCA)	**3.8**	9601	765.5
This work			
Regular RCLA (2-bit CLAs)	5.4	7177	**598.5**
Mixed RCLA (2-bit CLAs & RCA)	5.2	**7171**	603.2
Mixed RCLA (3-bit & 2-bit CLAs)	4.4	7807	611.3
Mixed RCLA (3-bit CLAs & RCA)	4.3	7801	616.1
Regular RCLA (4-bit CLAs)	4.2	8553	632.6
Mixed RCLA (4-bit & 2-bit CLAs)	4.2	8381	627.8
Mixed RCLA (4-bit, 2-bit CLAs & RCA)	**3.9**	8375	632.6
Mixed RCLA (4-bit CLAs & RCA)	3.9	8369	637.6

* Sizes: 2-bit RCLA – 286μm^2; 3-bit RCLA – 492μm^2; 4-bit RCLA – 744μm^2

TABLE 2. COMPARING DELAY, AREA AND POWER PARAMETERS OF 32, 48 AND 64-BIT MIXED SCBCLA AND MIXED RCLA ADDERS

Self-timed CLA adder type	Delay (ns)	Area (μm^2)	Power (μW)
Earlier work (32, 48 and 64-bit SCBCLA adders)			
Mixed SCBCLA (3-bit, 4-bit CLAs & RCA)	**3.8**	9601	765.5
Mixed SCBCLA (3-bit, 4-bit CLAs & RCA)	**4.8**	14667	1150.5
Mixed SCBCLA (3-bit, 4-bit CLAs & RCA)	**6.6**	19721	1697.1
This work (32, 48 and 64-bit RCLA adders)			
Mixed RCLA (4-bit CLAs & RCA)	3.9	**8375**	**632.6**
Mixed RCLA (4-bit CLAs & RCA)	5.2	**12641**	**941.5**
Mixed RCLA (4-bit CLAs & RCA)	7.3	**16895**	**1350**

CLA adders corresponding to different self-timed design methods were constructed and were subsequently optimized for minimum latency taking into account library constraints. Since identical registers and a similar completion detection circuit was used for all the CLA adders, the variations in their delay, area and power values can be directly attributed to physical differences in the function blocks synthesized, thus paving the way for a factual comparison. Table 1 shows the delay, area and total power dissipation metrics of various 32-bit self-timed CLA adders based on the DIMS technique [13], Toms & Edwards' method [14], SCBCLA [8], and RCLA adder design approaches. Table 2 gives the design parameters corresponding to larger size CLA adders constructed using Mixed SCBCLA and Mixed RCLA adder design methods. The optimum design metrics are highlighted in 'bold-face' in Tables 1 and 2 for a quick perusal.

IV. RESULTS DISCUSSION AND CONCLUSION

Referring to Table 1, among the various CLA architectures discussed in [8], only three SCBCLA adders are mentioned here which represent the low power, speed-power optimized and high speed cases respectively. From Table 1, it may be noted that from power and area perspectives, the RCLA adders outperform the SCBCLA adders, while from the delay viewpoint the latter are preferable. Table 2 confirms this with the Mixed RCLA adders achieving a power reduction of 19% and occupying less area by 13.8% compared to the Mixed SCBCLA adders, while the latter minimizes propagation delay by 7.3% compared to the former. Moreover, considering dual-rail encoded self-timed carry-ripple adders with sizes ranging from 32 to 64-bits, it is found that RCLA adders feature less delay and less area by 34.4% and 16% over RCA, while RCAs consume just 3.9% lesser power than RCLA adders.

ACKNOWLEDGMENT

The authors record their thanks to Dr. Wei Song of the School of Computer Science at the University of Manchester, UK for his help with the simulations.

REFERENCES

[1] SIA's ITRS Design Report 2011. Available: http://www.itrs.net

[2] A.J. Martin, P. Prakash, "Asynchronous nano-electronics: preliminary investigation," *Proc. IEEE ASYNC*, 2008, pp. 58-68.

[3] C. L. Seitz, *Introduction to VLSI Systems*, eds. C. Mead and L. Conway Addison-Wesley, MA, USA, 1980, pp. 218-262.

[4] T. Verhoeff, "Delay-insensitive codes: an overview," *Dist. Computing*, vol. 3, 1988, pp. 1-8.

[5] J. Sparsø, S.B. Furber (eds.), *Principles of Asynchronous Circuit Design – A Systems Perspective*, Kluwer Academic, 2001.

[6] A.J. Martin, "Asynchronous datapaths and the design of an asynchronous adder," *Formal Methods in System Design*, vol. 1, 1992, pp. 117-137.

[7] F.-C. Cheng, S.H. Unger, M. Theobald, "Self-timed carry-lookahead adders," *IEEE Trans. on Computers*, vol. 49, no. 7, 2000, pp. 659-672.

[8] P. Balasubramanian *et al*., "Robust asynchronous carry lookahead adders," *Proc. Intl. Conf. on Computer Design*, 2011, pp. 119-124.

[9] C.F. Brej, *Early Output Logic and Anti-Tokens*, PhD thesis, School of Computer Science, The University of Manchester, 2006.

[10] P. Balasubramanian, "A robust asynchronous early output full adder," *WSEAS Trans. on Circuits and Systems*, vol. 10, 2011, pp. 221-230.

[11] P. Balasubramanian, D.A. Edwards "A new design technique for weakly indicating function blocks," *Proc. DDECS*, 2008, pp. 116-121.

[12] C. Jeong, S.M. Nowick, "Optimization of robust asynchronous circuits by local input completeness relaxation," *Proc. ASP-DAC*, 2007, pp. 622-627.

[13] J. Sparsø, J. Staunstrup, "Delay-insensitive multi-ring structures," *Integration, the VLSI Journal*, vol. 15, 1993, pp. 313-340.

[14] W.B. Toms, D.A. Edwards, "Efficient synthesis of speed-independent combinational logic circuits," *Proc. ASP-DAC*, 2005, pp. 1022-1026.