Design of a Low Latency Asynchronous Adder Using Early Completion Detection

Article *in* Journal of Engineering Science and Technology · December 2014

CITATIONS
0

READS
162

4 authors, including:

Kok Keong Lai
Taylor's University
23 PUBLICATIONS **15** CITATIONS
SEE PROFILE

Edwin Chin Yau Chung
Taylor's University
23 PUBLICATIONS **21** CITATIONS
SEE PROFILE

Shih-Lien Lu
Taiwan Semiconductor Manufacturing
147 PUBLICATIONS **2,557** CITATIONS
SEE PROFILE

Journal of Engineering Science and Technology
Vol. 9, No. 6 (2014) 755 - 772

DESIGN OF A LOW LATENCY ASYNCHRONOUS ADDER USING EARLY COMPLETION DETECTION

KOK KEONG LAI[1,*], EDWIN C.Y. CHUNG[1], SHIH-LIEN L. LU[2], STEVEN F. QUIGLEY[3]

[1] School of Engineering, Taylor's University, 1 Jalan Taylor's, 47500, Subang Jaya, Selangor, Malaysia
[2] Intel Corporation, Hillsboro, OR 97124, USA
[3] School of Electronic, Electrical and Computer Engineering, University of Birmingham, Edgbaston, Birmingham, B15 2TT, UK
*Corresponding Author: KokKeong.Lai@sd.taylors.edu.my

Abstract

A new method for designing completion detection for asynchronous adders is introduced. The new completion detection is based on the property of a carry-merge tree for parallel-prefix adders where a generate bit at one level will have the same value as that in the previous level if there is no carry into the sequence of bits. This method has the advantages of a bundled data approach, allowing the use of single-rail completion detection design methodology, yet it allows the detection of early completion with very minimal gate count overhead. An alternative to "speculative completion," this method has approximately 10% improvement in performance at the costs of a 4% increase in area and a negligible increase in power consumption for Hybrid Skalansky Carry-Select and self-timed Kogge-Stone parallel prefix adders.

Keywords: Completion detection, parallel prefix adder.

1. Introduction

Self-timed asynchronous circuits have potential advantages over their synchronous counterparts in terms of speed, power, greater modularity [2], and scalability towards nanotechnology. Without a global clock, self-timed asynchronous circuits rely on the use of a request signal to initiate an operation and an acknowledge signal to indicate the completion of the operations. A request signal generally indicates that the data at the input of a processing unit are valid and ready whilst the acknowledge signal produced by the processing unit is used

to indicate that the data at the output of the processing unit are valid and ready. Requiring the processing unit to know when its output is valid is a conundrum not found in synchronous circuits.

To date, bundled data delay and dual-rail completion detection are the two most commonly used mechanism for completion detection [1-3]. A bundled data design uses the worst-case delay model and it is designed to exceed the longest path through the subsystem. The main advantage of this approach is that the standard synchronous single-rail design and implementation methodology remains applicable and the process to design and implement these circuits is well understood. The disadvantage is that completion is fixed to the worst-case computation, regardless of actual data inputs. A true completion detection method is expected to detect when a computation actually completes. Here, completion detection is typically implemented using dual-rail, where each bit is mapped to a pair of signals and encodes with it both the value and validity of the data. The advantage of this approach is that the self-timed circuits will operate at average rates giving better performance. The disadvantage of this approach is that in addition to the increase gate count associated with the complementary logic it inevitably adds additional gate delays between the completion and the detection logic. Furthermore, the additional logic for the complementary signals for dual-rail increases switching activities and inevitably results in higher power dissipation. For an arithmetic datapath design, such as an adder, both bundled data and dual-rail completion detection are not the optimum solution in term of power and performance. The low-cost bundled data approach does not take advantage of the circuit average operation delay and dual-rail, on the other hand, though completion detection is embedded in the dual-rail mechanism, it is too costly, in term of power and gate count. The completion comparator logic for dual-rail also adds additional delay to the arithmetic datapath.

In this paper, an alternative method for completion detection of asynchronous datapath components is proposed. This method is a derivative of the speculative completion [1, 2] and its modifications [3] proposed by Nowick and Koes et al. Similar to these methods, the proposed method uses the familiar single-rail bundled-data synchronous datapath approach coupled with a number of matched delay models. One represents the worst-case model delay, and the rests representing other possible delay paths through the computation element allowing a more accurate allocation of delay the computation element needs to complete its operation. The main difference in this approach is the basis upon which completion is determined.

2. Background

In this section, background information pertaining to parallel-prefix adders and the method of speculative completion detection are described.

2.1. Black and grey cells of radix-2 parallel prefix adder

Parallel-prefix adders are a family of adders derived from the commonly known carry-look-ahead adder (CLA). Parallel-prefix adders are best suited for operation involving wider word lengths and are frequently organized in a tree network to

reduce signal latency to $O(\log_2 n)$ where n represent the word length in number of bits [4]. In the following discussion, the black and the grey cell are used to represent the propagate and the generate logic commonly used to construct the various associative carry merge tree topologies in parallel prefix adders respectively. The logic of the black cell and the grey cell are as shown in Fig. 1.

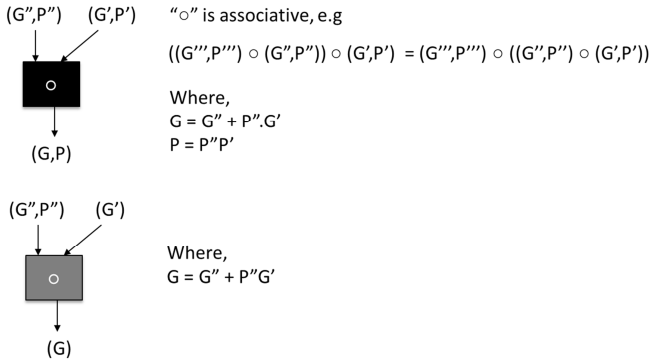

Fig. 1. Parallel Prefix Black and Grey Cells.

The speed and power consumption of these adders depends on many factors such as process technology, circuit family, circuit techniques and topology of the adders. In our experiment, one of our goals is to apply the early completion detection method on a few known fast adders without introducing too much overhead in power, logic gate count and delay.

2.2. Kogge-Stone and Skalansky adders

Based on the comparison study of VLSI adders by Ramanathan et al. [4], the Kogge-Stone (KS) adder and the Skalansky (SK) adder have the lowest logic depth as shown in Table 1.

Table 1. Characteristics of 32-bit Parallel Prefix Adders.

Adder Architecture	Number of Computation Nodes		Logic Depth
	Black Cell	Gray Cell	
Brent-Kung	26	31	8
Han-Carlson	33	31	6
Ladner-Fischer	33	31	6
Kogge-Stone	98	31	5
Sklansky	33	31	5

The architecture of a KS adder is as depicted in Fig. 2. It is considered as one of the fastest parallel prefix adders though it uses a lot more cells and wires in comparison to other adders. A 32-bit KS adder will have 5 levels of logic depth in its carry merge network from bit 16 onwards. In addition to completion detection, the goal of the early completion detection adder is to reduce one logic depth for sum[31:16] and to avoid adding an additional full mux delay to the carry chain due to the early completion detection selection.

The SK adder has the same carry logic depth in its carry merge as the KS adder and it uses fewer logic gates in term of implementation as shown in Fig. 3. The main disadvantage of the Skalansky adder is the presence of a number of high fanout nets in this design. The even bits of the SK adder have less logic depth vertically in the carry merge tree. Likewise, the goal of the early completion detection adder is to save on the logic depth for sum[31:16] and to avoid adding an additional full mux delay to the critical path along the carry chain of these adders due to the early completion detection selection.

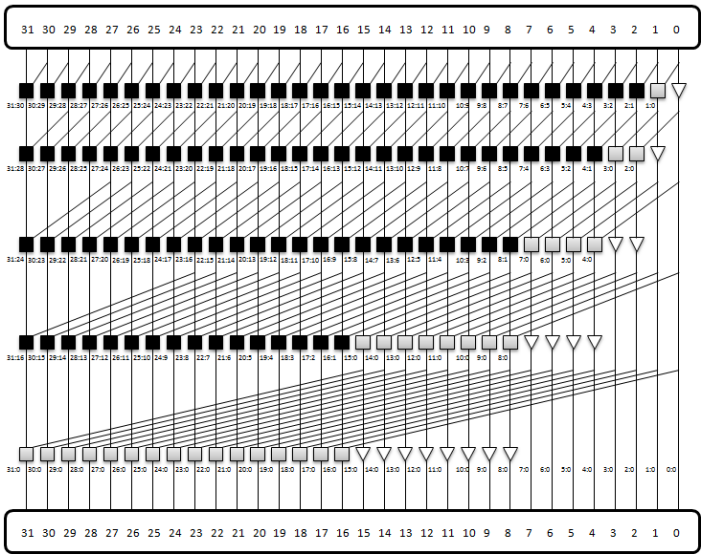

Fig. 2. 32 bits KS adder [5].

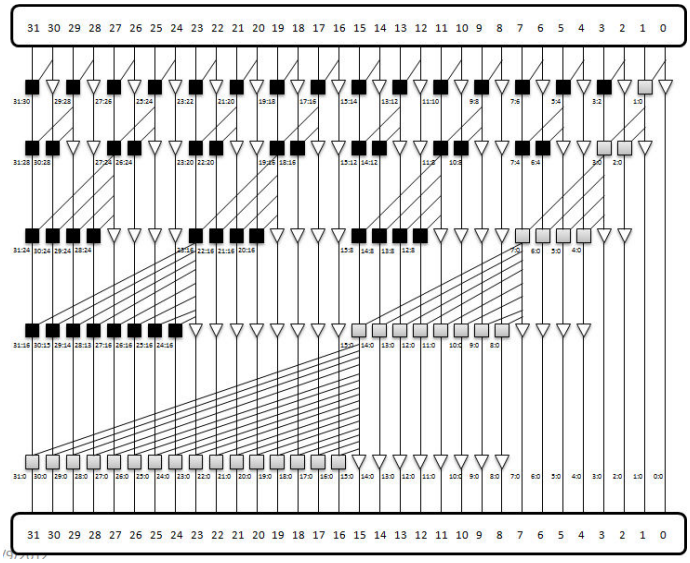

Fig. 3. 32 bits SK adder [5].

Both KS and SK adders are used in this study. Since both adders have different topology in the carry-merge logic, the completion detection carry chain qualifier logics will differ.

2.3. Speculative completion

The basic architecture of the speculative completion Brent-Kung adder was introduced by Nowick [1] and it has two key features. The first being a series of model delays that mimics the delay paths through the Brent-Kung adder. One for the worst-case scenario and the other for a speculative delay as depicted in Fig. 4. The second is the abort detection network associated with each speculative delay as shown in Fig. 5. These networks determine if the corresponding early completion signal propagating through their corresponding delay chain must be aborted due to the presence of operands that require a much longer computation time. Operating in parallel with the datapath the abort network must have a shorter computation delay than its corresponding delay for it to be able to abort the speculative completion signal in time. Nowick [1] states that the design of these abort detection networks must meet the following criteria for this method to work properly.

1) An abort must be asserted whenever the condition for late completion arises
2) Abort should not be asserted for most cases where early completion is expected
3) The network should be small and fast.

Condition (1) is a safety requirement and is satisfied by using a conservative approximation to detect the condition for late completion. Conditions (2) and (3) are optimality requirements on hit rate and logic realization, respectively.

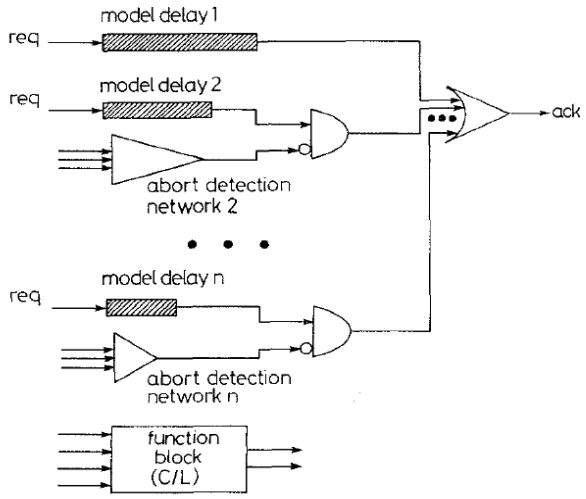

Fig. 4. General Architecture of Speculative Completion [1].

Based on the carry look ahead tree of a Brent-Kung adder, the generate signal for bit i at level n, $G_i^n = G_i^{n-1} + P_i^{n-1}G_{i-2^{n-1}}^{n-1}$ and $G_i^n = G_i^{n-1}$ if $P_i^{n-1} = 0$. An implication of this is that the result for bit i is known and equal to its value at the

intermediate level $n-1$ if the property of the segment of bits computing P_i^{n-1} is not propagative. The presence of 8 consecutive level-0 propagate, or what Nowick refers to as an 8-p run, would mean that the value of the most significant bit of these 8 consecutive bits at level 3 of the Brent-Kung adder may change and will need to be computed in the remain levels to arrive at the final value. Nowick uses this condition of 8-p run to indicate late completion and its logic is $p_0p_1p_2p_3p_4p_5p_6p_7 + p_1p_2p_3p_4p_5p_6p_7p_8 + \cdots + p_{24}p_{25}p_{26}p_{27}p_{28}p_{29}p_{30}p_{31}$.

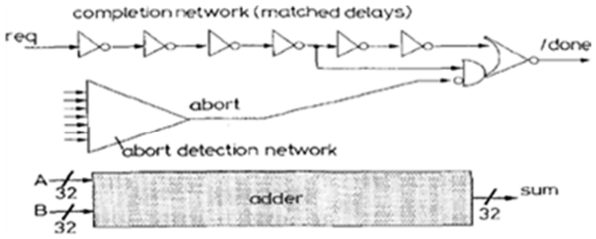

Fig. 5. Abort Network [1].

For a 32 bit adder, the implementation of the detection logic for all possible 8-p runs using conventional CMOS NAND and NOR gates is prohibitively expensive and exceedingly slow. It is also not applicable for faster parallel prefix adder such as the Skalansky or the Kogge-Stone adder. To overcome this limitation, Nowick proposed late completion detection logic with reduced literals as follows.

- 3-literal products:
  $p_7p_8p_9 + p_{13}p_{14}p_{15} + p_{19}p_{20}p_{21} + p_{25}p_{26}p_{27}$
- 4-literal products:
  $p_6p_7p_8p_9 + p_{11}p_{12}p_{13}p_{14} + p_{16}p_{17}p_{18}p_{19} + p_{21}p_{22}p_{23}p_{24} + p_{26}p_{27}p_{28}p_{29}$
- 5-literal products:
  $p_5p_6p_7p_8p_9 + p_9p_{10}p_{11}p_{12}p_{13} + p_{13}p_{14}p_{15}p_{16}p_{17} + p_{17}p_{18}p_{19}p_{20}p_{21} + p_{21}p_{22}p_{23}p_{24}p_{25} + p_{25}p_{26}p_{27}p_{28}p_{29}$

To understand the rationale behind this simplification, notice that the 3-literal term, $p_7p_8p_9$, will return a 1 if any one of the following 8 consecutive level-0 propagates is valid. Note however, it will also return a 1 if only $p_7p_8p_9$ is active while none of the following 8 consecutive level-0 propagate condition is valid.

- $p_2p_3p_4p_5p_6p_7p_8p_9$
- $p_3p_4p_5p_6p_7p_8p_9p_{10}$
- $p_4p_5p_6p_7p_8p_9p_{10}p_{11}$
- $p_5p_6p_7p_8p_9p_{10}p_{11}p_{12}$
- $p_6p_7p_8p_9p_{10}p_{11}p_{12}p_{13}$
- $p_7p_8p_9p_{10}p_{11}p_{12}p_{13}p_{14}$

Naturally, a 5-literal product late-detection logic is more accurate than its 3-literal counterpart and at the expense of complexity and speed. And to improve the accuracy of these simplified detection networks, Nowick further proposes augmenting them using kill terms which are less complex to implement than propagate terms. The augmented 3-literal simplified late-detection network would now be $\overline{k_6}p_7p_8p_9 + \overline{k_{12}}p_{13}p_{14}p_{15} + \overline{k_{18}}p_{19}p_{20}p_{21} + \overline{k_{24}}p_{25}p_{26}p_{27}$. Here kill term, k_i, denote that both the i^{th} input bits are 0. Finally, the i^{th} sum bit, s_i, is then computed as $S_i = p_i^0 \oplus (\overline{late}\,.\,G_{i-1}^3 + late.G_{i-1}^5)$.

To further improve on Nowick's speculative Brent-Kung adder, Koes et al. [3] proposed an alternative to the way the final sum bits are computed. This however require the generate signals at level-3 and level-5 to reset at the start of a computation. Koes et al. [3] also proposed the use of customised application specific late completion detection logic with improve accuracy of late completion detection in cases where applicable.

3. Early Completion Detection

The key disadvantage of the speculative completion detection method described above is not only that the late completion detection logic is speculative but that it also add additional delay and high logic depth to the adder implementation. In this paper an alternative to late completion detection is proposed. There is no speculation or approximation in this approach and as such will not require an abort network for each delay nor will it incorrectly identify an early completion as being late.

Principle of early completion

When adding 2 numbers, it is the propagation of the carry signal through the carry merge tree that determines the speed of a particular adder configuration. Accordingly, any means to determine the value of a carry signal based only on the properties of the operands would allow the value of these carry signals to be set ahead of time. It would also be possible to know the delay required for the remaining logic to compute the final results.

Consider the carry out of the i^{th} bit when adding 2 numbers. This carry out is related to the level-0 propagate, generate and zero signals of the i^{th} and the $(i-1)^{th}$ bit. These bits are denoted in this paper as p_i^0, g_i^0, z_i^0, p_{i-1}^0, g_{i-1}^0 and z_{i-1}^0 respectively where $p_i^0 = a_i \oplus b_i$, $g_i^0 = a_i \cdot b_i$ and $z_i^0 = \overline{a_i + b_i}$. The relationship of these signals to c_i (the carry out of bit i) is as detailed in Table 2 and is captured in the following expression. Note that it is possible to determine the value of the carry in most of the scenario but one.

$$c_i = \begin{cases} 0 & when\ p_i^0.z_{i-1}^0 + z_i^0 \\ 1 & when\ p_i^0.g_{i-1}^0 + g_i^0 \\ g_{i-2} & when\ p_i^0.p_{i-1}^0 \end{cases}$$

Table 2. Early Completion Detection Table.

	p_{i-1}^0	g_{i-1}^0	z_{i-1}^0	
p_i^0	g_{i-2}	1	0	
g_i^0	1	1	1	c_i
z_i^0	0	0	0	

The generate and the propagate signals for a group of bits spanning from bit i to bit j in a parallel-prefix adder at any level, denoted here as $g_{i:j}$ and $p_{i:j}$ respectively, are each derived using the expressions $g_{i:j} = g_{i:k} + p_{i:k+1} \cdot g_{k:j}$ and $p_{i:j} = p_{i:k+1} \cdot p_{k:j}$ with $i < k < j$. Note that if $z_k = 1$ then $g_{i:j} = g_{i:k+1}$. In other words, any carry from bits *k-1:j* will not propagate through bit *k*.

The early-completion function where $c_i = 0$ when $p_i^0 . z_{i-1}^0 + z_i^0$ will be used to select a different model delay and also to select the output of the adder. Based on the simple 2 input operands combination, the probability of hitting the early completion condition is tabulated in Table 3.

Table 3. Early Completion Probability.

a_i	b_i	a_{i-1}	b_{i-1}	$p_i^0 . z_{i-1}^0$	z_i^0
0	0	–	–		0.25
0	1	0	0	0.125	
1	0	0	0		

The total probability of hitting the early completion condition for any arbitrary bit i is 0.375 for early detection method. To increase the probability of the early completion detection, more bits can be used to detect the early termination with different delay models by adding additional detection logics with the patterns of $p_i p_{i-1} \cdots p_{j+1} z_j$.

The design of the early completion detection network is similar to previous works in terms of integration. This early completion detection network, as shown in Fig. 6, has two key components. First, a worst-case delay model and early completion detection delay model are used. Second, the detection logic result is used as carry qualifier in the carry merge logic of the adder.

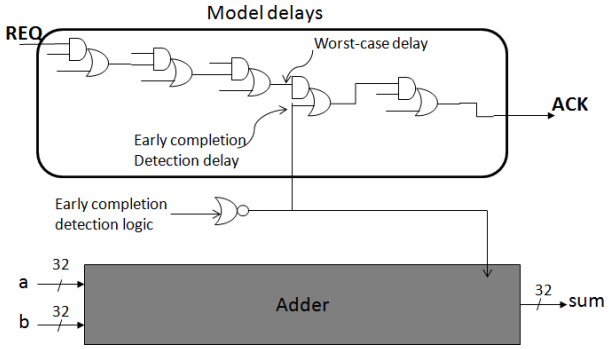

Fig. 6. Early Completion Detection Network.

4. Hybrid Carry-Select Early Completion Detection Adder (HCSECDA)

The goals of the early completion detection adder are to (i) simplify the delay model selection, (ii) faster delay in producing the SUM, (iii) minimal overhead in implementing the early completion detection logic, (iv) accurately detect completion.

For the purpose of practicality, the following criteria for the early completion detection adder were established. (i) The early completion detection logic delay, t_{ELDL} combined with the sum of product of the various delay models, t_{SDM} has to be smaller than the normal delay model, t_{NDM}. In other words, $t_{ELDL} + t_{SDM} < t_{NDM}$. (ii) The early completion detection method is general and applicable for all parallel prefix adders. (iii) The additional logic gates introduced must be minimal.

Thus far, we have implemented the early completion detection method for both KS and SK adders and will use the SK adder implementation as the example to illustrate the HCSECDA implementation. This HCSECDA implementation was implemented using the Synopsys GTECH cells instantiation as shown in Fig. 7. Note that only the logic for sum[8] is shown in Fig. 7. The delay is measured based on GTECH cell default simulation delay. For simplicity, all the cells are implemented using positive logic gates.

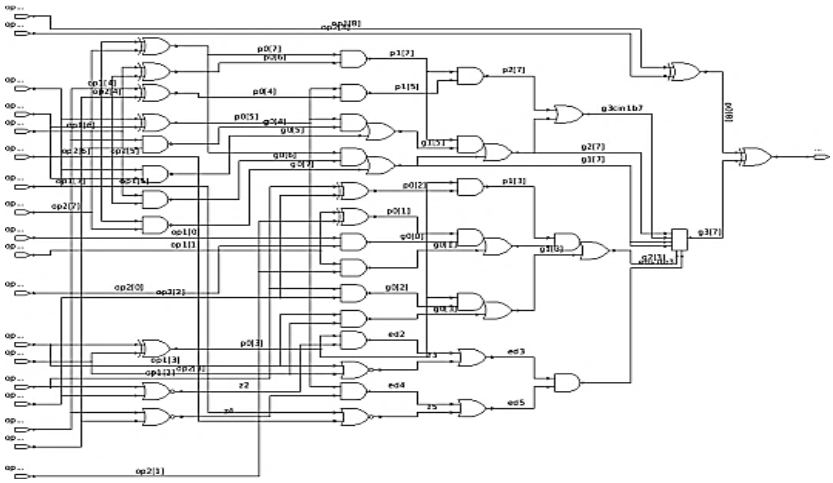

Fig. 7. Schematic for bit 8 of the HCSECDA Sum Output.

4.1. Carry chain implementation

The longest delay in the 32 bits Skalansky adder carry chain is 5 logic depth stages in the carry-merge tree starting from bit 16 as shown in Fig. 3. The last level of the Skalansky adder produces $g^5_{31:0} = g^4_{31:16} + p^4_{31:16} \cdot g^4_{15:0}$. When *EarlyCompletionDection*, $ECD^0_{15} = p^0_{15} \cdot z^0_{14} + z^0_{15} = 1$ then $g^5_{31:0} = g^4_{31:16}$.

In this early completion scenario, the sum[31:16] at stage 5 is the same as that at stage 4. Using the stage 4 output under this condition gives the effect of eliminating one logic depth stage from the implementation. Applying the same

principle, note that $g^4_{15:0} = (g^2_{15:12} + p^2_{15:12} \cdot g^2_{11:8}) + p^3_{15:8} \cdot g^3_{7:0}$. For the case of $ECD^0_{15} = 0$, 2 further logic depth can be eliminated from $g^4_{15:0}$ if and $ECD^0_7 = 1$.

Also, in order to avoid adding a full mux delay on the carry-merge tree to select the early completion detection path, the carry select method mux is combined with the early completion detection mux as shown in Fig. 8 where the logic for g^4_{15} is as follows. The hybrid carry select and early completion detection mux structure is replicated in sum[31:16].

$$g^4_{15} = \begin{cases} g^2_{15} & when\ ECD^0_7 . ECD^0_{11} = 1 \\ g^3_{15} & when\ g^3_7 = 0 \\ g^3_{15} + p^3_{15} & when\ g^3_7 = 1 \end{cases}$$

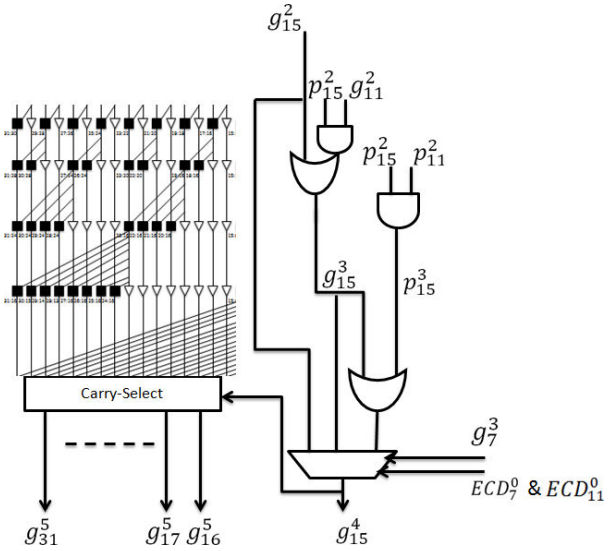

Fig. 8. Carry Chain Network.

4.2. Delay models and REQ/ACK control

By combining the usage of *ECD* from different bits will produce different completion timing of the adder. Different early completion delay models will be used with different *ECD* usage. In order not to introduce a big slow OR logic to sum all the delay models, the delay model is constructed using the 2 input AND-OR gate as shown in Fig. 9. The delay model structure is to match the carry-merge logic of the adder. The ECD results from different bits will be merged into the single AND-OR gate delay model to produce different completion delay.

In HCSECDA, four delay models are used to generate the ACK. Each delay model has to be qualified with the REQ to prevent the ACK from being falsely triggered. In order to control the hazards, D flip-flops are used to ensure all the operands and all the sum arrived and released at the same time respectively. The flip-flops are controlled by the REQ and ACK as shown in Fig. 10.

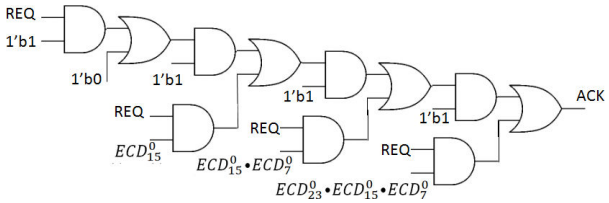

Fig. 9. HCSECDA Delay Models.

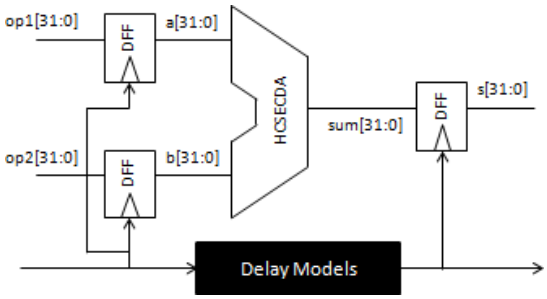

Fig. 10. HCSECDA Integration.

The HCSECDA uses the four-phase bundled-data protocol, only the request line, REQ going high signifies the validity of data and a single transition of the acknowledge signal, ACK identifies that the data has been consumed by the receiver and can be safely altered by the sender as shown in Fig. 11 [7].

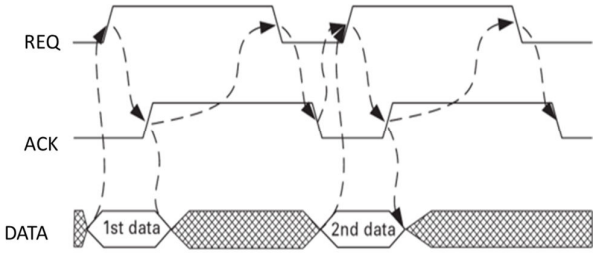

Fig. 11. Four-phase Bundled-data Early Protocol.

5. Simulation Results

Simulation of both SK and KS adders were carried out using Synopsys Verilog Compiler Simulator, VCS using Synopsys GTECH cells and default delay with a timescale of 100ps. Both adders were simulated with 2 billion patterns for the adders' speed benchmark. The HCSECDA SK adder has a better performance improvement due to the reason that more aggressive early completion detection

logic was implemented in the SK adder. The aggressive early completion detection network was implemented by taking advantage of the high fanout topology of SK adder. Overall, as shown in Fig. 12, the HCSECDA gains 8% to 10% speed improvement in simulation time.

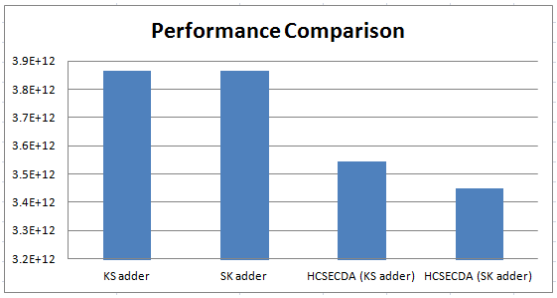

Fig. 12. Computation Delay Performance Comparisons.

As for area and power comparison, the adders are synthesized using the Silterra 0.13um with ARM Sage-X regular Vt standard cells library. As shown in Fig. 13, HCSECDA is about 4% larger for both adders. The cell area number is based on the library liberty's value with the 2 input NAND gate size of 15.27. As shown in Fig. 14, the difference in power is very insignificant and there is an interesting observation that the HCSECDA for the SK adder has a slightly lower power consumption due to a decrease in cell power.

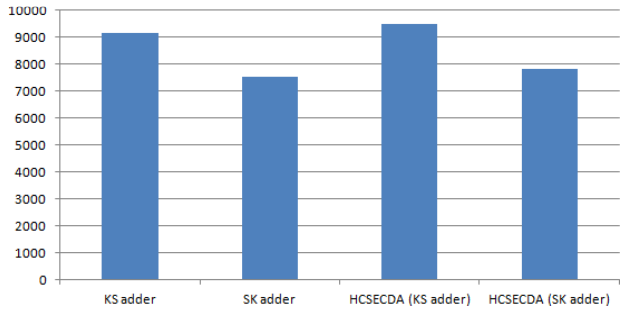

Fig. 13. Adders Area Comparison.

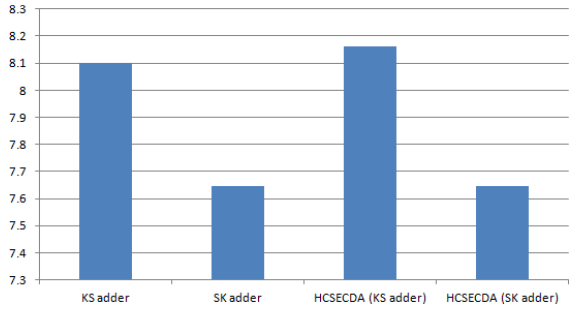

Fig. 14. Adders Power Comparison (mW).

6. Conclusion

There are not many researches were done in the past to have an optimum way to implement the completion detection for the asynchronous datapaths especially for adders. Although there are many studies were done how to design the faster parallel prefix adder in the synchronous domain but the new adder design improvement will not be more than 1 stage of inversion delay which will be lesser 10%. This paper presents a novel optimum method for improving the speed of adders by using a uniform early completion detection method in the asynchronous domain. It combines the principle of speculative completion [1], early termination [3] and the property that $g_{i:j} = g_{i:k+1}$ if z_k = 1. This has been found to have minimal overhead in term of power and area while producing about 10% performance improvement in comparison to one of the fastest parallel prefix synchronous adder. The new completion detection concepts that are presented in this paper can be applied to most of the asynchronous design datapaths such as multipliers and dividers. This paper creates a new concept on how to design an optimum completion detection circuits for asynchronous datapaths design.

References

1. Nowick, S.M. (1996). Design a low-latency asynchronous adder using speculative completion. *IEE Proceeding – Computers and Digital Techniques*, 143(5), 301-307.
2. Nowick, S.M.; Yun, K.Y.; Beerel, P.A.; and Dooply, A.E. (1997). Speculative completion for the design of high performance asynchronous dynamic adders. *Proceedings of Third International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 210-223.
3. Koes, D.; Chelcea , T.; Onyeama, C.; and Goldstein, S.C. (2005). *Adding faster with application specific early termination. Computer Science Department, Carnegie Mellon University*, Paper 765.
4. Ramanathan, P. and Vanathi, P.T. (2009). Hybrid prefix adder architecture for minimizing the power delay product. *International Journal Electrical and Computer Engineering*, 4(9), 613-617.
5. Koren, I. (2002). *Computer arithmetic algorithm*. MA:A K Peters, 93-138.
6. Beerel, P.A.; Ozdag, R.O. and Ferretti, M. (2010). *A Designer's guide to asynchronous VLSI*. New York: Cambridge University Press.
7. Tsai, H.-Y.; Cheng, W.-M.; Chang, Y.T.; Chen, C.-J.; and F.-C. (2011). Self-timed dual-rail processor core implementation for microcontrollers. *Proceedings of the 2011 International Conference on Electronic Devices, System and Applications* (*ICEDSA*)**,** 39-44.