International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2014): 5.611

Implementation of Error Detection Network in High- Speed Variable Latency Speculative Han-Carlson Adder

C. Dhanalakshmi[1], C. Manjula[2]

[1, 2]Adhiyamaan College of Engineering, Hosur

Abstract: ***Variable latency adders have been recently proposed in literature. In variable latency adder unwanted interconnections also reduced compared with kogge-stone topology. Kogge-Stone adder consists of large number of black cells and many wire tracks. A variable latency adder employs speculation: the exact arithmetic function is replaced with an approximated one that is faster and gives the correct result most of the time, but not always. In order to detecting the error, error detection network also used. The approximated adder is augmented with an error detection network that asserts an error signal when speculation fails. Speculative variable latency adders to reduce average delay compared to traditional architectures. This paper proposes a novel variable latency speculative adder based on Han-Carlson parallel- prefix topology which proposes the error detection network that reduces error probability compared to previous approaches. Several variable latency speculative adders, for various operand lengths, using both Han-Carlson and Kogge-Stone topology, have been synthesized using Xilinx 14.3. Obtained results show that proposed variable latency Han-Carlson adder used in high-speed application.***

Keywords: Addition, digital arithmetic, parallel-prefix adders, speculative adders, speculative functional units, variable latency adders

1. Introduction

VLSI binary adders are critically important elements in processor chips, they are used in floating-point arithmetic units, ALUs, memory addresses program counter update and magnitude comparator [1, 2]. Adders are extensively used as a part of the filter such as DSP lattice filter [3]. Ripple carry adder is the first and most fundamental adder that is capable of performing binary number addition. Since its latency is proportional to the length of its input operands, it is not very useful. To speed up the addition, carry look ahead adder is introduced. Parallel prefix adders provide a good results as compared to the conventional adders. The adders with the large complex gates will be too slow for VLSI, so the design is modularized by breaking it into trees of smaller and faster adders which are more readily implemented. For large adders the delay of passing the carry through the look-ahead stages becomes dominated and therefore tree adders or parallel prefix adders are used. High speed adders depend on the previous carry to generate the present sum. In integer addition any decrease in delay will directly relate to an increase in throughput. In nanometer range, it is very important to develop addition algorithm that provide high performance while reducing power. Parallel prefix adders are suitable for VLSI implementation since they rely on the use of simple cells and maintain regular connection between them. We can define each prefix.

Structures in terms of logic levels, fanout and wiring tracks. Zero or more inverters are added to each prefix cell output to minimize the delay based on this model, buffers are individually sized to minimize the delay, buffers are used to minimize the fanout and loading on gates since high fanout causes poor performance. we design an extremely fast unreliable adder that produces correct results for the vast majority of input combinations. For brevity, we will call this adder an *Almost Correct Adder (ACA).*

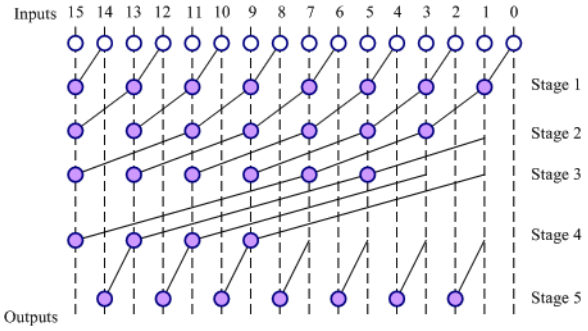


Figure 1: Graph representation of 16-bit Hybrid Han-Carlson Adder

2. Preliminaries

This section introduces the unreliable ACA. Throughout the paper, binary integers are denoted by uppercase letters, e.g., *A,B,X* etc.; the *ith* least significant bit of an integer *X* is denoted by *xi*−1. In order to add two *n*-bit integers *A* and *B*, one can define generate, propagate and kill signals at each bit position as follows:

$gi = aibi,$

$pi = ai \oplus bi,$ and

$ki = ai + bi.$

Using these signals the carry output *ci* at each bit position *i* is generated and is used to compute the sum bits. The recurrence for *ci* is shown below.

$$ci = \begin{cases} 0 \; if \; ki = 1, \\ 1 \; if \; gi = 1, \\ ci-1 \; otherwise \; (pi = 1). \end{cases}$$

$$si = ai \oplus bi \oplus ci-1.$$

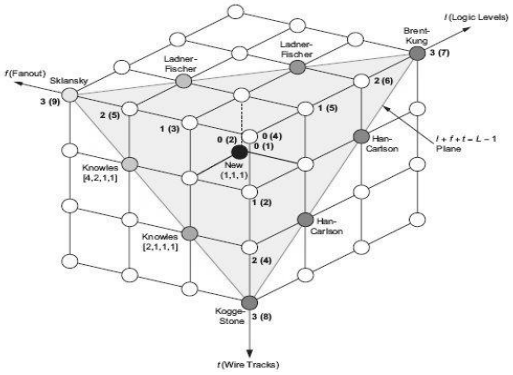
Note that the carry bit ci depends on the carry bit ci−1 only if the propagate signal pi is true, otherwise ci can be determined locally based on the values of gi and ki. Similarly, ci−1 depends on ci−2 only if pi−1 is true. This means ci depends on ci−2 only if both pi and pi−1 are true. In general ci will depend on ci−k only if every propagate signal between bit position i and i−k+1 (inclusively) is true. If an oracle provides us with the longest sequence of propagate signals in advance, then an extremely fast adder could be constructed. For example, we want to add two 20-bit integers. Since the longest sequence of propagate signals is 4, the carry output ci at bit position i will be independent of ci−5. Hence, si can be computed only using the input bits of 6 preceding bit positions starting from ith bit position. In other words, we can form several 6 bit adders, each computing the carry-in and sum bit for a particular bit position. The delay of this particular 20-bit adder will be virtually the same as that of a 6-bit adder. Ideally, one would like to know the longest sequence of propagate signals in the input addenda. There are, in fact, no bounds on the length of the longest propagate sequence. In extreme cases such as for integers A = 11. . . 1, B = 00. . . 0 the length of the longest propagate sequence is the same as the bit width of A and B. However, in the next section, we show that on average, the length of the longest propagate sequence is approximately log n, where n is the bit width of the integers.

3. Previous Work

The different types of parallel prefix adders available are Kogge-Stone adder, Brent-kung adder, Sklansky adder, Han-Carlson adder, Knowles adder and Ladner-Fischer adder. These adders offer a tradeoffs among the number of stages of logic, the number of logic gates, fanout and amount of wiring between stages. Kogge-Stone adder, Brent-kung adder and Sklansky adder are the fundamental adders. Brent-Kung uses minimal number of computation nodes which yields in reduced area but structure has maximum depth which yields slight increase in latency. Slansky reduces the delay at the expense of increased fanout. Kogge-Stone achieves high speed and low fanout but produces complex circuitry with more numbers of wiring tracks[5].

The Knowles trees are family of network between Kogge-Stone and Sklansky with increased fan-out. Ladner Fischer introduced a network between Sklansky and Brent-Kung which provides tradeoffs between logic levels and fanout. T. Han and D.A. Carlson presented a hybrid construction of a parallel prefix adder using two designs the Kogge-Stone construction having the best feature of higher speed and the Brent-kung construction with best feature of low area requirement. A modified Han-Carlson adder uses fewer number of prefix operations by adjusting the number of stages amongst Kogge-Stone and Brent-kung adder and thus reduces the area required by the adder circuitry. Fig 2.below shows a 3-dimentional taxonomy of tree adders [6]. There are three axes representing the fan-out, wiring tracks and logic levels and each tree is indicated by three integers (l, f, t) in the range [0, L-1]. The tree adders lie on the plane l + f + t = L-1, where L= $\log_2 N$ and indicates the number of bits. Brent-Kung, Kogge-Stone and Sklansky represent the vertices of the cube (3, 0, 0),(0, 0, 3) and (0, 3, 0) respectively. Han-Carlson, Ladner-Fischer and Knowles lie along the diagonals. Where N indicates the number of bits the variables l, f, and t are integers in the range [0, L− 1] indicating:

1) Logic Levels: L+ l
2) Fan-out: 2^f+1
3) Wiring Tracks:



2^t

Figure 2: Taxonomy of prefix networks

4. Proposed Work

Design the Speculative Han-Carlson Adder. It differs from other adder in that it can be used for large word sizes. The proposed design reduces the number of prefix operation by using more number of Brent-Kung stages and lesser number of Kogge-Stone stages. This also reduces the complexity, silicon area and power consumption significantly.

Variable latency speculative prefix adders can be subdivided in five stages: pre-processing, speculative prefix-processing, post-processing, error detection and error correction. The error correction stage is off the critical path, as it has two clock cycles to obtain the exact sum when speculation fails.

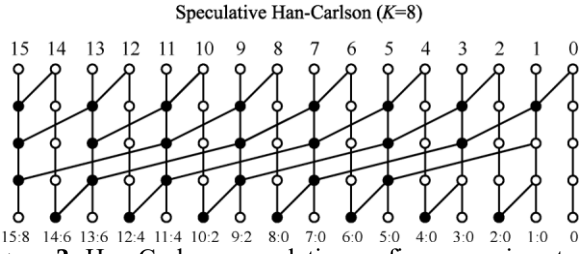


Figure 3: Han-Carlson speculative prefix-processing stage.

Consider the *n*-bit addition of two numbers: $A = an-1$, $an-2$, $a0$ and $B = bn-1$, $bn-2$, $b0$ resulting in the sum, $S = sn-1$, $sn-2$, $s0$ and a carry, Cout. The first stage in CLA computes the bit generate and bit propagate as follows:

$$gi = ai \cdot bi$$
$$pi = ai + bi, \quad (1)$$

Where *gi* is the bit generates and *pi* is the bit propagate. The schematic of *gi* and *pi* using CMOS and transmission gates design style. These are then utilized to compute the final sum and Carry bits, in the last stage as follows:

$$si = pi \oplus ci,$$
$$Ci+1 = gi + pi \cdot ci, \quad (2)$$

Where ·, + and ⊕represent AND,OR, and XOR operations. It is seen that the first and last stages are intrinsically fast because they involve only simple operations on signals local to each bit position. However, intermediate stages embody the long-distance propagation of carries, as a result of which the performance of the adder hinges on this part [10]. These intermediate stages calculate group generate and group propagate to avoid waiting for a ripple which, in turn, reduces the delay. These group generate and propagates are given by

$$Pi{:}\,j = Pi{:}k \cdot Pk-1{:}\,j\,,$$
$$Gi{:}\,j = Gi{:}k + Gk-1{:}\,j \cdot Pi{:}k. \quad (3)$$

There are many ways to develop these intermediate stages, the most common being parallel prefix. Many parallel prefix networks have been described in the literature, especially in the context of addition. In this paper, we have used the Kogge-Stone implementation, Hans-Carlson, Sklansky, Brent-Kung implementation of CLA, and Kogge- Stone implementation of Ling adder. PG logic in all adders is generally represented in the form of cells. These diagrams known as cell diagrams will be used to compare a variety of adder architectures in the following sections. Here two cells are used for implementation of all the adders: grey cell and the black cell.

Han-Carlson Adder: The *Han-Carlson* trees are a family of networks between Kogge-Stone and Brent-Kung. The logic performs Kogge-Stone on the odd numbered bits and then uses one more stage to ripple into the even positions.

Kogge-Stone Adders: The main difference between Kogge-Stone adders and other adders is its high performance. It calculates carries corresponding to every bit with the help of group generate and group propagates. In this adder the logic levels are given by log2*N*, and fan-out is 2.

$$di = ai \oplus bi. \quad (4)$$

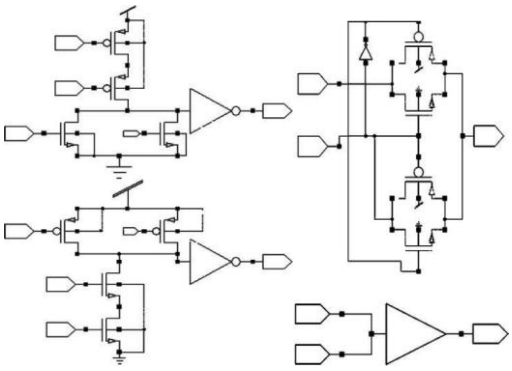


Figure 4: Block generate and propagate (Ling carry) using CMOS and transmission gate.

Now, instead of utilizing traditional carries, a new of carry, known as Ling carries, is produced where the *i*th Ling carry in [11] is defined to be

$$ci = Hi \cdot pi, \quad (5)$$

Where

$$Hi = ci + ci-1. \quad (6)$$

In this way, each Hi can be in turn represented by

$$Hi = gi + gi-1 + pi-1 \cdot gi-2 + \cdots + pi-1 \cdot pi-2 \cdot pi-3 \cdot \ldots p1g0. \quad (7)$$

We can see from (5) that Ling carries can be calculated much faster than Boolean carry. Consider the case of $c4$ and $H4$

$$c4 = g4 + p4 \cdot g3 + p4 \cdot p3 \cdot g2 + p4 \cdot p3 \cdot p2 \cdot g1 + p4 \cdot p3 \cdot p2 \cdot p1 \cdot g0, \quad (8)$$
$$H4 = g4 + g3 + p3 \cdot g2 + p3 \cdot p2 \cdot g1 + p3 \cdot p2 \cdot p1 \,.\, g0.$$

If we assume that all input gates have only two inputs, we can see that calculation of $c4$ requires 5 logic levels, whereas that for $H4$ requires only four. Although the computation of carry is simplified, calculation of the sum bits using Ling carries is much more complicated. The sum bit, when calculated by using traditional carry, is given to be

$$si = di \oplus ci-1. \quad (9)$$

Substituting (5) into (9), we get that

$$si = di \oplus pi-1 \cdot Hi-1 \quad (10)$$

However, according to [12] the computation of the bits si can be transformed as follows:

$$si = Hi-1 \cdot di + Hi-1(di \oplus pi-1) \quad (11)$$

Equation (11) can be implemented using a multiplexer with $Hi-1$ as the select line, which selects either di or $(di \oplus pi-1)$. No extra delay is added by Ling carries to compute the sum since the delay generated by the XOR gate is almost equal to that generated by the multiplexer and that the time taken to compute the inputs to the multiplexer is lesser than that taken to compute the Ling carry.. Here, for *n*-bit addition, Ling carry Hi and $Hi+1$ is given by

$$Hi = (G*i, P*i-1).(G*i-2, P*i-3).\square\cdots\square(G*0, P*-1),$$
$$Hi+1 = (G*i+1, P*i).(G*i-1, P*i-2).\cdots.(G*1, P*0), \quad (12)$$

Where

$$G*I = gi + gi-1,$$
$$P*I = pi \cdot pi-1, \quad (13)$$

$$H3 = g3 + g2 + p2 \cdot g1 + p2 \cdot p1 \cdot g0, \quad (14)$$
$$H4 = g4 + g3 + p3 \cdot g2 + p3 \cdot p2 \cdot g1 + p3 \cdot p2 \cdot p1 \cdot g0.$$

$$H3 = G*3 + P*2 \cdot G*1,$$
$$H4 = G*4 + P*3 \cdot G*2 + P*3 \cdot P*1 \cdot G*0. \quad (15)$$

This can be further reduced by using (13) to

This can be then further reduced by using the "." operator to

$$H3 = (G*3, P*2).(G*1, P*0),$$
$$H4 = (G*4, P*3).(G*2, P*1).(G*0, P*-1). \quad (16)$$

This allows the parallel prefix computation of Ling adders using a separate tree [9] for even and odd indexed positions. Using this methodology, we implemented a 16-bit adder using the Kogge-Stone tree and then utilized that block to develop 32 and 64-bit adders. The gates and blocks used for this implementation were then modified using transmission gates. Cells other than gray and black cell that are used as components in Ling adder.

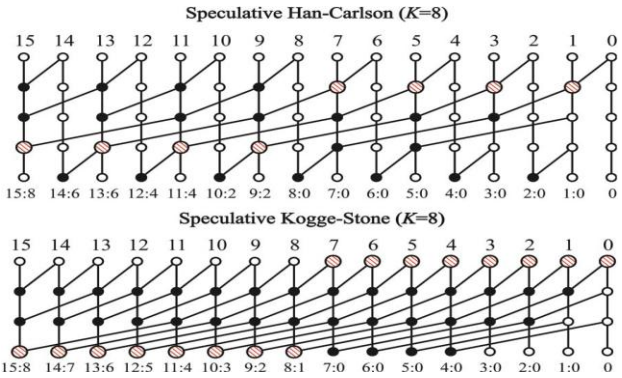


Figure 5: The nodes of the prefix-processing stage, whose outputs are needed to compute the error signal, are named "checking nodes" and are highlighted as big hatched dots.

5. Adders Characterization

5.1 Error Detection

This section presents a circuit that flags an error if the sum computed by the ACA is incorrect. This only occurs when there is a chain of more than k propagates in the addenda. To check for the presence of an error, we must consider all chains of length $k + 1$, and check if any of them contain solely propagates. The expression for error signal is stated as follows:

$$ER = \sum_{i=0}^{n-k-1} pipi+1 \ldots . pi+k.$$

The critical path delay to compute error signal has complexity O (log k + log ($n - k$)). Since $k = O(\log n)$, the error signal complexity can be reduced to O (log n). The critical path delay of error detection has the same complexity as that of the critical path delay of a traditional adder; however, the error detector only requires simple gates, such as AND, OR, etc.,. Experimentally, we report that the delay of the error detection signal is approximately two-thirds of the delay of a traditional adder.

5.2 Error Recovery

Once an error has been detected, one could simply employ a traditional correct adder to produce the sum. Instead, we have developed a novel error recovery technique that uses a computation inside the ACA to reduce both the critical path delay and hardware area. The matrix product $MiMi-1$, $Mi-k+1$ computes the propagate and generate signals for the block between bit position i and $i - k + 1$. Thus, the ACA computes the propagate and generate signals for each k-bit block. If we divide the input integers into n/k blocks of k-bits, the values of propagate and generate for each block can be taken from the ACA. An n/k-bit carry look-ahead adder then takes these values and computes the carry for each of the blocks. Meanwhile, we compute the propagate and generate signals for each bit in a block.

In Han-Carlson the critical checking cells are in the second last level of the graph and are also available after three black cells delay. As it can be observed, in Kogge-Stone some of the checking cells are at the last level of the graph; their output signals are available after three black cells delay.

Table 1: Delay and power of Han-Carlson Speculative adder

	Han- Carlson Adder
Delay	9.810 ns
Power	0.016W (or) 160mW

6. Variable Latency Speculative Adder

We observe that for 16 bits, the delay of the ACA and error detection mechanisms are approximately equal and, individually, both are significantly less than the delay of a traditional adder. Based on this observation, we have designed the circuit shown in Fig. 6 whose clock period is slightly greater than the critical path delay of the error detection circuit. After one cycle, the circuit produces the result of the ACA and a bit indicating whether or not an error has been detected. If there is no error, then the circuit provides $SUM *$ as its output and will also set the *VALID* bit to 1. Since *STALL* is the complement of valid, the circuit will be ready to accept a new set of input addenda. If an error occurs, the valid bit will be set to 0 and the stall bit will be set to 1. After two cycles, the corrected sum value will be available and the valid bit will be set to 1. At this point, the circuit is ready to accept new inputs. We call this type of adder a *Variable Latency Speculative Adder (VLSA)*. . Since the ACA produces a correct sum in more than 99.99% of all cases, the average latency will be 1.0001 cycles. The effective latency of the circuit is almost half of the latency of the fastest traditional adder.

7. Synthesis Results

Verilog descriptions of the proposed variable latency speculative adders, and of their non-speculative counterpart. It is not easy to compare performances (in terms of power, speed, and area) of different designs, since they strongly depend on timing constraint used during synthesis.

7.1 The Optimal K Choice

Comparison between variable latency adder and the non-speculative Han-Carlson topology reveal that variable latency adders allow to reduce the minimum achievable delay. For instance, in the 64 bit case, the minimum achievable delay is

about 280 ps for the non-speculative adder and reduces up to 225 ps in the variable latency architecture.

7.2 Comparison with Kogge-Stone Variable Latency Speculative Adder

Fig. 7 shows the comparison between proposed speculative adder and Kogge-Stone one. Also in this case, we report the performance of non-speculative adders, in order to identify the region where the speculative approach is effective. As an example, focusing on 64-bit adders, for lower than 350 ps, the proposed Han-Carlson speculative adder is the best choice in terms of silicon area and power consumption. Moreover, it allows to reduce the minimum achievable to 225 ps, with a 18% improvement respect to Kogge-Stone non-speculative adder and a 11% improvement respect to Kogge-Stone speculative adder. For , proposed speculative adders offer 45% area reduction and 35% power saving compared to Kogge-Stone non-speculative adder.

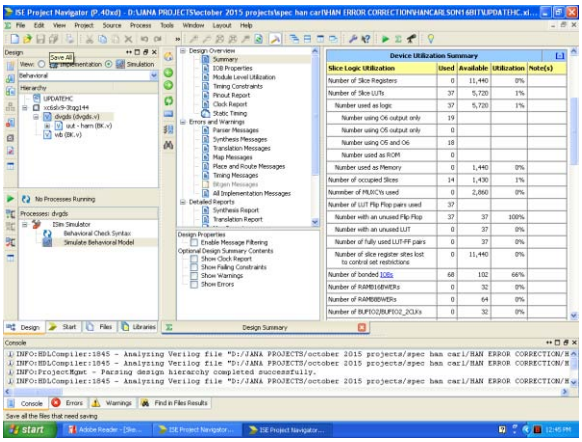


Figure 7(a)

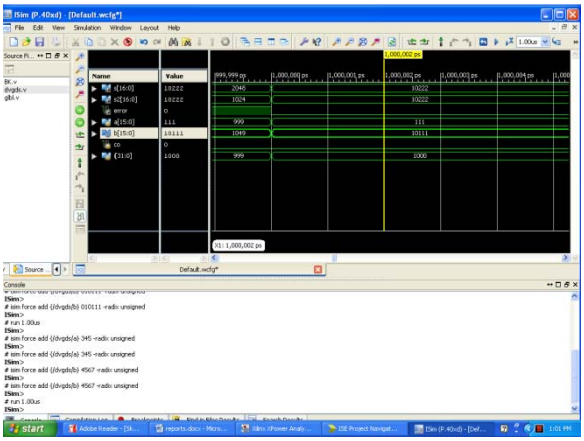


Figure 7(b)

Fig. 7(a). Area report of Han-Carlson Adder, 7(b). error correction report of Han- Carlson Adder

8. Conclusion

In this paper a novel variable latency Han-Carlson parallel prefix Speculative adder for high-speed application is proposed. An accurate error detection network is implemented to reduce the error probability compared with kogge stone adder. Variable latency adder performance mainly depends on prefix-processing stage. Speculative latency adder reduces the number of black cells. Compared with traditional, non-speculative, adders, our analysis demonstrates that variable latency Han-Carlson adders show sensible improvements when the highest speed is required; otherwise the burden imposed by error detection and error correction stages overwhelms any advantage. Additional work is required to extend the speculative approach to other parallel-prefix architectures, such as Brent-Kung, Ladner-Fisher, and Knowles.

References

[1] I. Koren, Computer Arithmetic Algorithms. Natick, MA, USA: A K Peters, 2002.
[2] R. Zimmermann, “Binary adder architectures for cell-based VLSI and their synthesis,” Ph.D. thesis, Swiss Federal Institute of Technology, (ETH) Zurich, Zurich, Switzerland, 1998, Hartung-Gorre Verlag.
[3] R. P. Brent and H. T. Kung, “A regular layout for parallel adders,” IEEE Trans. Comput., vol. C-31, no. 3, pp. 260–264, Mar. 1982.
[4] P. M. Kogge and H. S. Stone, “A parallel algorithm for the efficient solution of a general class of recurrence equations,” IEEE Trans. Comput., vol. C-22, no. 8, pp. 786–793, Aug. 1973.
[5] J. Sklansky, “Conditional-sum addition logic,” IRE Trans. Electron. Comput., vol. EC-9, pp. 226–231, Jun. 1960.
[6] T. Han and D. A. Carlson, “Fast area-efficient VLSI adders,” in Proc. IEEE 8th Symp. Comput. Arith. (ARITH), May 18–21, 1987, pp. 49–56.
[7] K. M. Butler, J. Saxena, A. Jain, T. Fryars, J. Lewis, and G. Hetherington, “Minimizing power consumption in scan testing: Pattern generation and DFT techniques,” in *Proc. Int. Test Conf.*, 2004, pp. 355–364.
[8] V. R. Devanathan, C. P. Ravikumar, and V. Kamakoti, “On power profiling and pattern generation for power-safe scan tests,” in *Proc. Des.,Autom., Test Eur. Conf.*, 2007, pp. 1–6.
[9] C.-W. Tzeng and S.-Y. Huang, “QC-fill: Quick-and-cool X-filling for multicasting-based scan test,” *IEEE Trans. Comput., Aided Des.*, vol. 28,no. 11, pp. 1756–1766, Nov. 2009.
[10] M.-F. Wu, H.-C. Pan, T.-H. Wang, J.-L. Huang, K.-H. Tsai, and W.-T. Cheng, “Improved weight assignment for logic switching activity during at-speed test pattern generation,” in *Proc. 15th Asia South Pacific Des. Autom. Conf.*, 2010, pp. 493–498.