VLSI Design, 2002 Vol. 14 (2), pp. 155–169

Taylor & Francis
Taylor & Francis Group

Testability Synthesis for Jumping Carry Adders

CHIEN-IN HENRY CHEN[a,*] and MAHESH WAGH[b]

[a]*Department of Electrical Engineering, Wright State University, Dayton, OH 45435, USA;* [b]*LSI Logic, Milpitas, CA 95035, USA*

(Received 27 June 1999; In final form 4 February 2000)

Synthesis for testability ensures that the synthesized circuit is testable by exploring the fundamental relationship between don't care and redundancy. With the exploration of the relationship, redundancy removal can be applied to improve the testability, reduce the area and improve the speed of a synthesized circuit. The test generation problems have been adequately solved, therefore an innovative testability synthesis strategy is necessary for achieving the maximum fault coverage and area reduction for maximum speed. This paper presents a testability synthesis methodology applicable to a top–down design method based on the identification and removal of redundant faults. Emphasis has been placed on the testability synthesis of a high-speed binary jumping carry adder. A synthesized 32-bit testable adder implemented by a 1.2 μm CMOS technology performs addition in 4.09 ns. Comparing with the original synthesized circuit, redundancy removal yields a 100% testable design with a 15% improvement in speed and a 25% reduction in area.

Keywords: Binary adders; Ripple carry adders; Carry select adders; Testability synthesis; Redundant logic; Redundant faults

INTRODUCTION

A synthesis process involves converting a register-transfer level (RTL) description to a gate-level netlist consisting of interconnected gate-level primitive and macro cells which have been optimized for performance and area. In synthesis, the optimization and mapping processes need to meet design constraints which are typically classified as area, timing, and testability.

Synthesis for testability approaches can be classified into two categories: synthesis approaches that impose constraints on logic optimization such that the resulting circuit is restricted to the fully testable subset of the overall design space, or approaches that exploit the fundamental relationships between don't cares and redundancy in combinational and sequential circuits [1,2]. The latter approach has the advantage that the addition of extra don't care conditions during the logic optimization step can improve the area and performance characteristics of a design as well as its testability. Logic minimization techniques augmented with satisfiability and observability don't-care sets can ensure primality and irredundancy for a Boolean network and guarantee 100% single stuck-at fault testability [3]. While don't-care exploitation and minimization is useful for area minimization [4], the don't-care minimization procedure that makes a Boolean network prime and irredundant has found not practical in use [3]. This is because complete don't-care sets are typically very large and are difficult to be generated and used during minimization.

Currently, the most popular method for obtaining prime and irredundant Boolean networks or fully single stuck-at fault testable circuits is to use test generation algorithms to iteratively identify and remove single stuck-at fault redundancies in combinational logic circuits. Extensive work in test generation for single stuck-at fault has resulted in the development of efficient methods for redundancy removal.

Redundancy is the main link between test and logic optimization. If there are untestable stuck-at faults, there is likely to be redundant logic. The reason is that if a stuck-at fault does not have any test (the fault is untestable), the output responses of the faulty circuit (with this untestable fault) will be identical to the responses of the fault-free circuit. Thus, the faulty circuit with an untestable stuck-at fault is indeed a valid implementation of the fault-free circuit. Therefore, when test generation identifies a stuck-at-1 (stuck-at-0) fault as untestable, you can simplify the

*Corresponding author.

ISSN 1065-514X print/ISSN 1563-5171 online © 2002 Taylor & Francis Ltd
DOI: 10.1080/10655140290010079

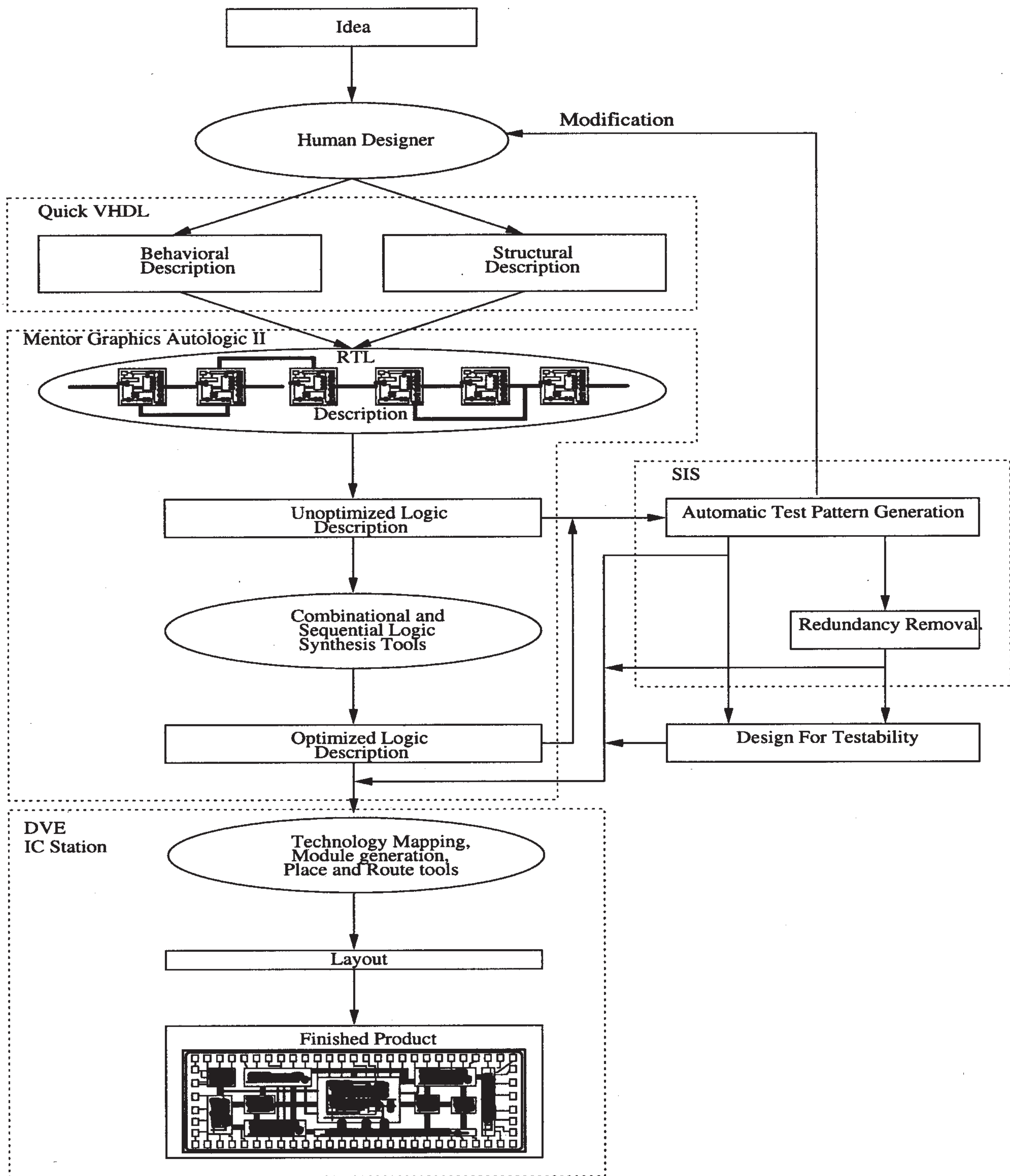


FIGURE 1 Synthesis for testability pipeline.

circuit by setting the faulty net to logic 1(0), thus effectively removing the faulty net from the circuit. This operation, called redundancy removal, also removes all the logic driving the faulty net if the net has no fanout. Because this method only removes logic from the circuits, the circuit is smaller when the process ends; the topological delay of the longest paths will be shorter than or at the most equal to that of the original circuit. The power dissipation of the optimized circuit will also be lower.

Removing a redundant fault can change the status of other faults. Those that were redundant may now not be redundant and vice versa. Although these changes complicate redundancy removal, they also pave the way for optimization in redundancy removal. Even for a circuit with no redundancies, designers can add redundancies to

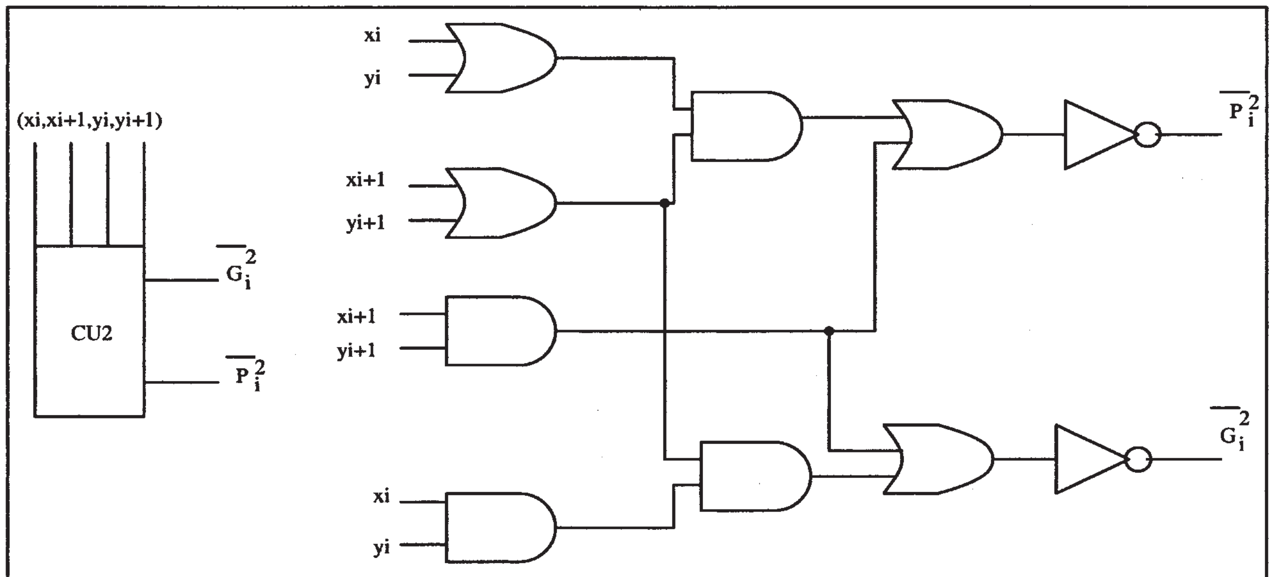


FIGURE 2 2-bit carry unit "CU2".

create new redundancies elsewhere in the circuit. Thereafter, an optimized circuit can be obtained by removing the newly created redundancies [5].

Figure 1 describes the testability synthesis pipeline which employs test generation for identification and removal of redundancy. To ensure that all logic was testable, we started checking it at the RTL. We designed the chip hierarchically, with different modules that we can assemble to create a full chip. For each module of the design, we included all testable modules that could result in changes to the design, to avoid costly surprises later. The testability synthesis pipeline eliminated time-consuming iterations caused by conventional design flows in which test engineers returning to the design engineers with requests to address testability issues. In this synthesis pipeline the Mentor graphics toolset is used for RTL synthesis. The *SIS* toolset [6] is used for logic optimization. A conversion handler software named *ABC* [7] was developed for conversion between the synthesized logic and the structural format used in *SIS* for the purpose of testability synthesis.

The testability synthesis pipeline employs efficient algorithms for test pattern generation and redundancy removal. First, fault collapsing is performed across simple gates; both fault equivalence and fault dominance algorithms are used to minimize the total number of faults to be considered. Random test generation is done using parallel fault simulation. After the random patterns have been simulated, the algorithm performs a deterministic search to find tests for the remaining faults. A set of equations is written to express the difference between the good and the faulty circuits and Boolean satisfiability is used to find a satisfying assignment for these equations. While the test generation for stuck-at faults is NP-complete, the implementation has been able to produce 100% stuck-at-fault coverage. By properly orienting the search for redundancy, redundancy removal iteratively removes all redundant faults.

32-BIT JUMPING CARRY BINARY ADDER

A new class of jumping carry adder was designed in which the carry generation circuitry was modified to generate all the carry bits. The principle of a conditional carry generation was applied so that a configuration similar to a spanning tree adder could be obtained for high-speed additions.

Carry Generation

Let $\oplus$ denote the exclusive-OR operation, $+$ denote the logic OR operation, and the juxtaposition of two variables denote the logic AND operation. In a two-operand addition, let the two operands be $X = \{x_{n-1}, \ldots, x_1, x_0\}$ and $Y = \{y_{n-1}, \ldots, y_1, y_0\}$ and let the sum be $S = \{s_{n-1}, \ldots, s_1, s_0\}$. The addition process at each bit position then obeys the rule $s_i = x_i \oplus y_i \oplus c_i$, where c_0 is the carry input to an n-bit adder. The carry generation at each bit position can be described as $c_{i+1} = \bar{c}_i x_i y_i + c_i(x_i + y_i)$. This form of representation was intentionally used to derive carry generation based on a 2:1 multiplexer Boolean function. Alternatively, the carry generation can be expressed as: $c_{i+1} = \bar{c}_i G_i^1 + c_i P_i^1$ where $G_i = G_i^1 = x_i y_i$ and $P_i = P_i^1 = x_i + y_i$. These definitions are also used in carry lookahead adders. Note that some carry lookahead adders and especially Manchester carry chain and spanning tree adders use $P_i = x_i \oplus y_i$ instead. The difference is that $G_i = P_i$ when $x_i = y_i = 1$. If $P_i = x_i \oplus y_i$, G_i and P_i will not

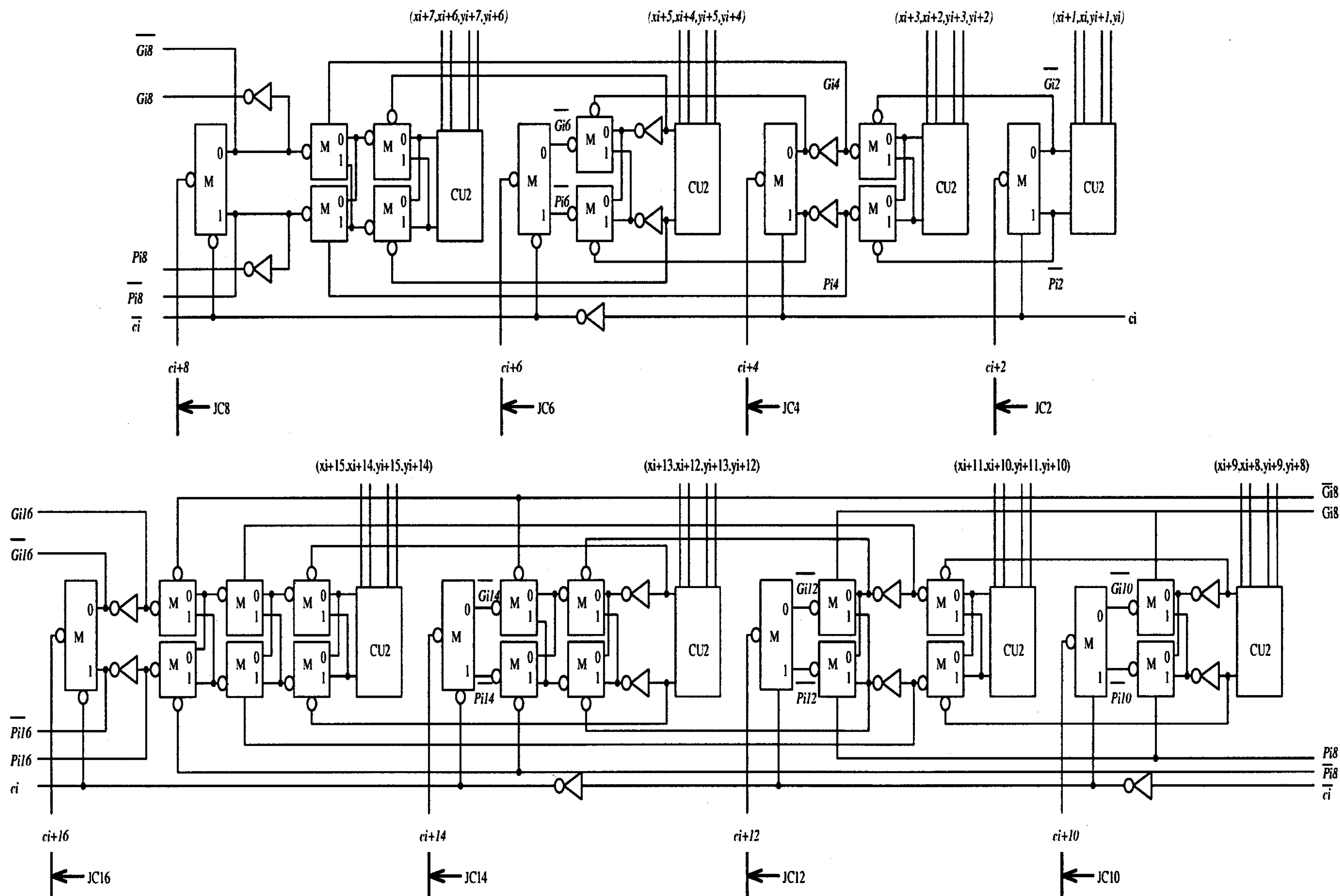


FIGURE 3 Schematic of JC2 to JC16 units.

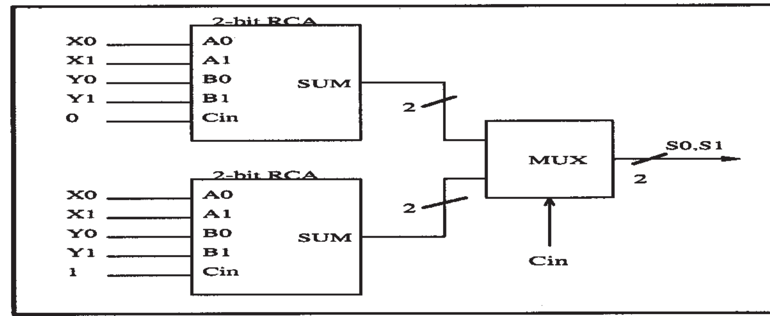


FIGURE 4 Schematic of 2-bit CSA.

be 1 at the same time. Next, we find

$$\begin{aligned} c_{i+2} &= \bar{c}_{i+1}G^1_{i+1} + c_{i+1}P^1_{i+1} \\ &= (\bar{c}_iG^1_i + c_iP^1_i)G^1_{i+1} + (\bar{c}_iG^1_i + c_iP^1_i)P^1_{i+1} \\ &= \bar{c}_i(\bar{G}^1_iG^1_{i+1} + G^1_iP^1_{i+1}) + c_i(\bar{P}^1_iG^1_{i+1} + P^1_iP^1_{i+1}). \end{aligned}$$

In other words, we may define

$$G^2_i = \bar{G}^1_iG^1_{i+1} + G^1_iP^1_{i+1} \quad \text{and}\, P^2_i = \bar{P}^1_iG^1_{i+1} + P^1_iP^1_{i+1}$$

such that

$$c_{i+2} = \bar{c}_iG^2_i + c_iP^2_i.$$

Adder Construction

The original conditional carry generation used multiplexers in a cascade configuration. A modification of the conditional carry generation uses a more regular or monolithic structure of multiplexers. This design is similar to the first one where multiplexers are cascaded. However, the 2-bit carry generation function was used and thus, only half of the carry bits were generated. Redundant carry generation circuits are used to generate c_2,c_6,c_{14},c_{22}, and c_{32} independently for a 32-bit adder. Carry-select adders are then used to generate sum bits accordingly. Moreover, the adder has to be used in the 8-bit and 16-bit carry select adders (CSAs) such that the sum bits can be generated at a speed comparable to that of the carry bits. There are only three basic building blocks in the proposed adder design; 2:1 multiplexer, CU2 and ripple carry adder.

Carry lookahead functions are derived two bits at a time. It is shown that a more efficient design is achieved this way. Similarly, a direct generation of G^2_i and P^2_i can be derived easily as follows: $G^2_i = x_iy_i(x_{i+1} + y_{i+1}) + x_{i+1}y_{i+1}$ and $P^2_i = (x_i + y_i)(x_{i+1} + y_{i+1}) + x_{i+1}y_{i+1}$. This direct implementation of G^2_i and P^2_i will be used as the basic building block of the carry generation circuit. Figure 2 shows the gate level realization of $\bar{G}^2_i$ and $\bar{P}^2_i$. A 2-bit carry unit (CU2) consists of both functions.

The sum of the adder is done by ripple carry adder blocks. These blocks are the main part in the CSAs for the sum generation. The sum generation can be derived easily as $s_i = x_i \oplus y_i \oplus c_i$. The schematic of a 16-bit jumping carry (JC) units, JC16, is shown in Fig. 3. Other jumping carry units for a word length less than 16 can also be seen in this figure. When the carry circuit is realized exactly as shown in Fig. 3 where all eight carry bits are generated, it is essentially a higher radix version of the carry portion of a conditional-sum adder.

The CSA units are built using the ripple carry adders (RCAs) and the multiplexers. The adders are pre-charged by either a low or high voltage carry-in. The select signal on the multiplexer is the carry out from a previous cell, which controls which sum should be chosen. Figure 4 shows a 2-bit CSA that will generate sum_0 and sum_1. The schematics for the 4, 8 and 16 bit CSA units will be shown later in this section. The JC units are used as building blocks in these units to generate the carry.

For an optimal speed design based on the timing characteristics of the available building blocks, it was decided that each JC unit would generate only one carry bit. Figure 5 shows the optimal speed design of the 32-bit adder [8]. The bold lines show the critical path in this design. As shown in Fig. 5, a hierarchical structure is used to ensure that the sum bits are available when the carry bit is generated. As indicated in Fig. 5, the worst case delay time occurs at S_{22}–S_{29}. This delay time is measured between the insertion of input operands, represented by X_0, and the arrival of the worst case sum bit, represented by S_{29}.

Testability Analysis

The 32-bit adder designed above was not designed to ensure a 100% testable design. Each and every individual module of the design as well as the entire design is subjected to testability analysis which is shown in Table I. The 32-bit fast carry adder design has 714 redundant faults where JC2, JC4, JC8, CSA2, CSA4, CSA8 and CSA16 have 1, 3, 7, 34, 16, 35 and 492 redundant faults, respectively. The fault coverage of 32-bit fast carry adder is 81.17% where JC2, JC4, JC8, CSA2, CSA4, CSA8 and CSA16 has the fault coverage of 96.7, 95.7, 95.36, 69.36, 92.19, 92.44 and 72.33%, respectively. The next section deals with the synthesis for testability of the 32-bit adder in order to ensure 100% testability.

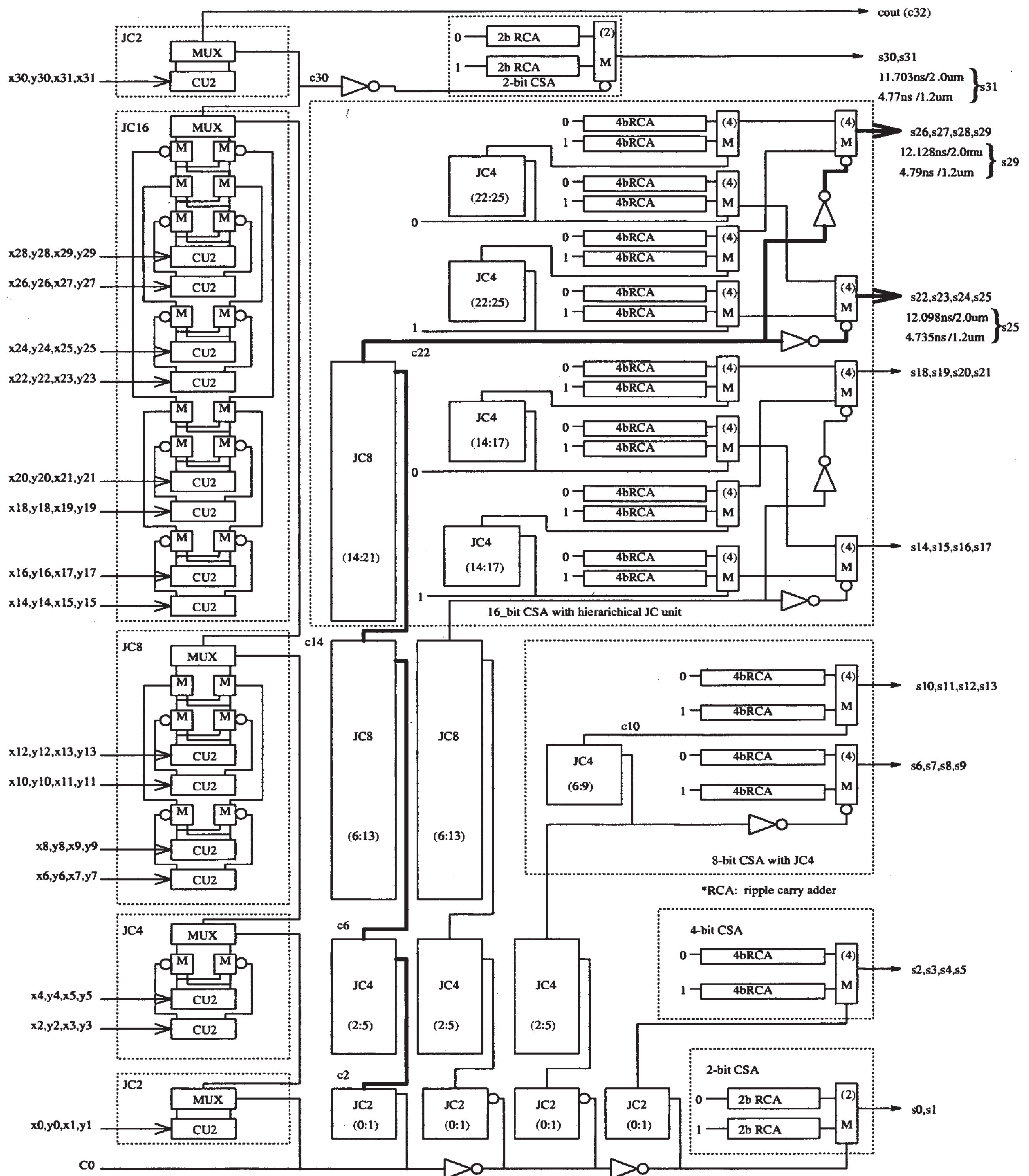


FIGURE 5 The optimal speed a 32-bit adder structure.

TESTABILITY SYNTHESIS OF 32-BIT JUMPING CARRY ADDER

The automatic test pattern generation (ATPG) for the 32-bit jumping carry adder resulted in a total of 714 redundant faults. These redundant faults are due to the redundant logic present in the design. In this section, testability synthesis for identification and removal of redundant faults for achieving maximum fault coverage and area reduction for maximum speed is illustrated.

Jumping Carry (JC) Unit

The jumping carry unit JC2 as described consists of a CU2 unit cascaded with a multiplexer. The synthesis for testability process detected one redundant fault. The fault is

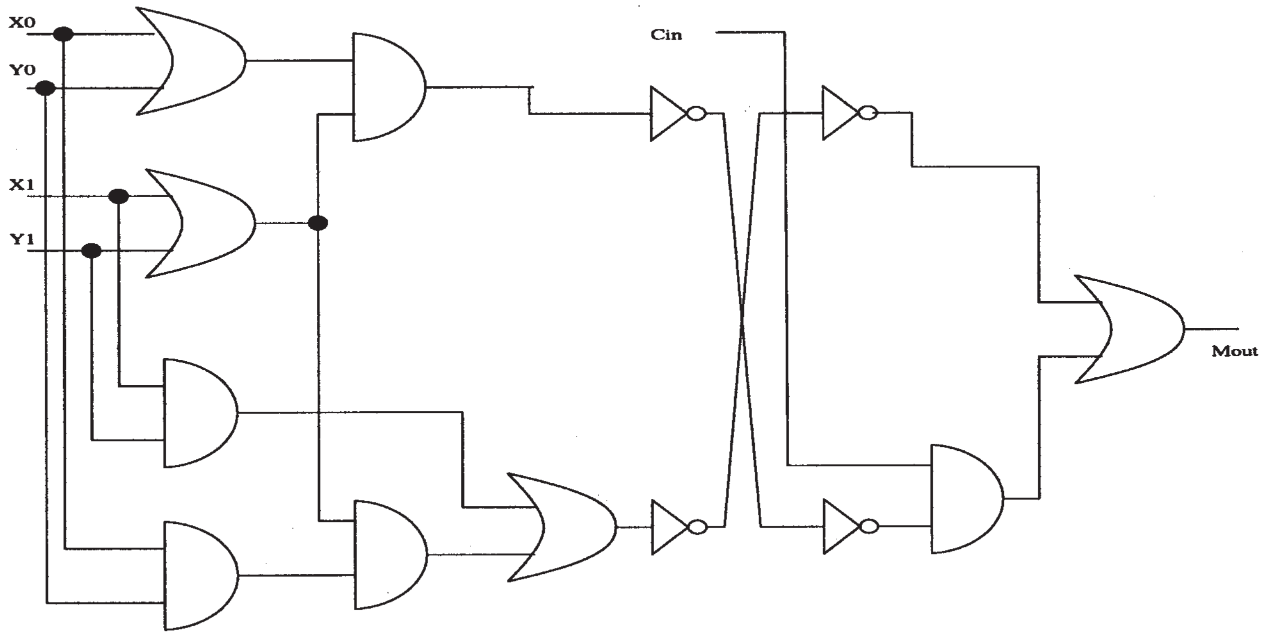


FIGURE 6 Testable JC2 unit.

TABLE I Testability analysis of a 32-bit adder

Design	Total faults	Detectable faults	Redundant faults	Fault coverage	Tests
JC2	31	30	1	96.7%	12
JC4	71	68	3	95.7%	21
JC8	151	144	7	95.36%	45
CSA2	111	77	34	69.36%	14
CSA4	205	189	16	92.19%	23
CSA8	463	428	35	92.44%	33
CSA16	1845	1353	492	73.33%	93
Adder32	3793	3079	714	81.17%	365

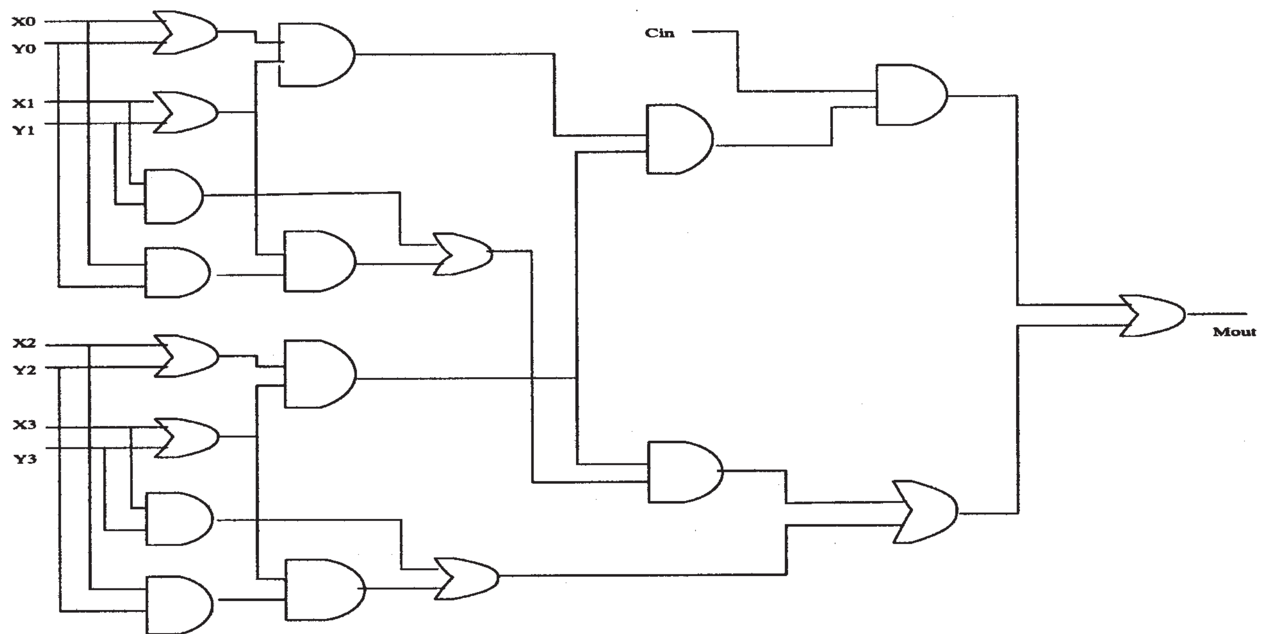


FIGURE 7 JC4 unit after redundancy removal.

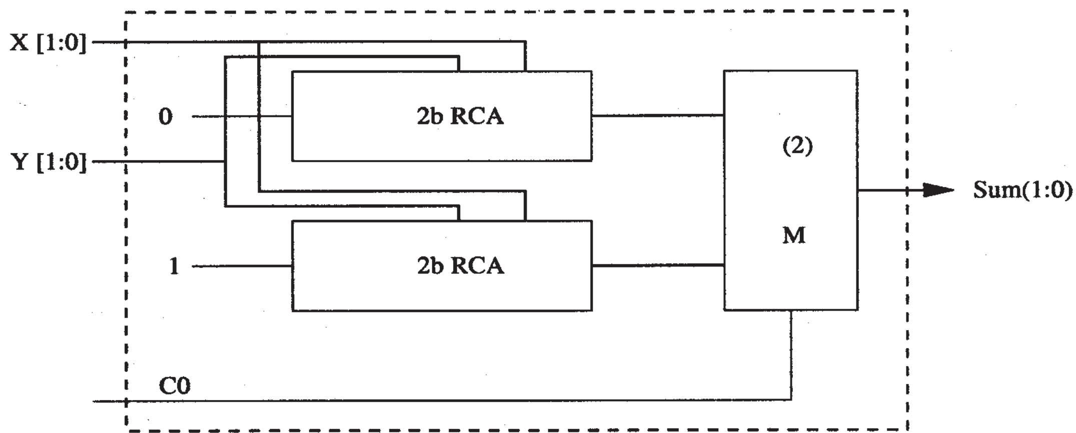


FIGURE 8 2-bit carry select adder (CSA).

identified and removed by using *SIS*. Figure 6 shows the testable circuit. Similarly, redundant faults were removed from JC4, JC8, JC16 units. Redundant logic was present in most of the cases in the multiplexer and in some of the cases at the fanout branches (OR gates). Figure 7 describes the circuit after redundancy removal. The CU2 unit and the multiplexer units by themselves were optimized and had no redundant logic present. It is observed that the CU2 unit produces two outputs whereas the multiplexer has only one output. This affects the controllability and observability of the nodes drastically. The reason for this is that when these circuits are analyzed individually they have a better controllability features because there are two primary outputs and the goal is to propagate the faulty value at the faulty site to any one of the two primary outputs. The multiplexer on the other hand has only one output and, thus the controllability of the nodes is now difficult since the faulty value has to be propagated to the only primary output. The use of a multiplexer introduces more don't care conditions in the design which need to be taken care of. It is also important to discuss the interdependency of the circuits. It has been observed that circuits that previously appeared to be optimized and were irredundant appear to carry redundant logic when cascaded.

Carry Select Adders

A major portion of redundant logic was identified as present in the carry select adders. Table II lists the redundant faults for the carry-select adders. Figure 8

TABLE II ATPG analysis of CSA modules

Design	Total faults	Detectable faults	Redundant faults	Fault coverage	Tests
CSA2	111	77	34	69.36%	14
CSA4	205	189	16	92.19%	23
CSA8	463	428	35	92.44%	33
CSA16	1845	1353	492	73.33%	93
Adder32	3793	3079	714	81.17%	365

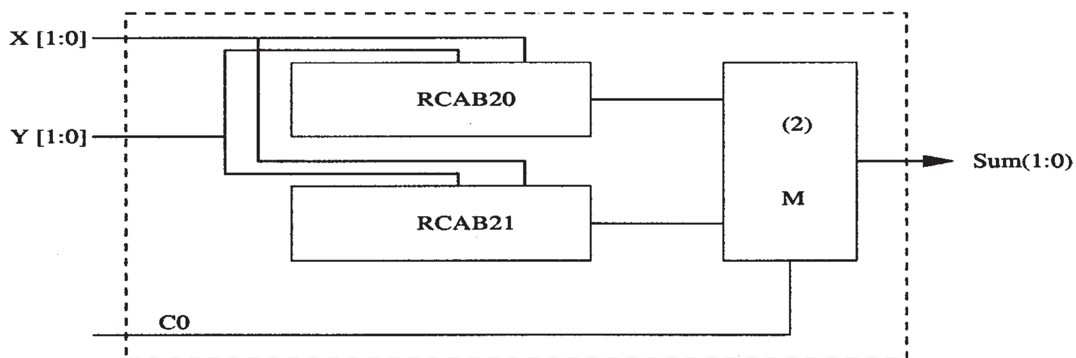


FIGURE 9 2-bit CSA with embedded constant logic.

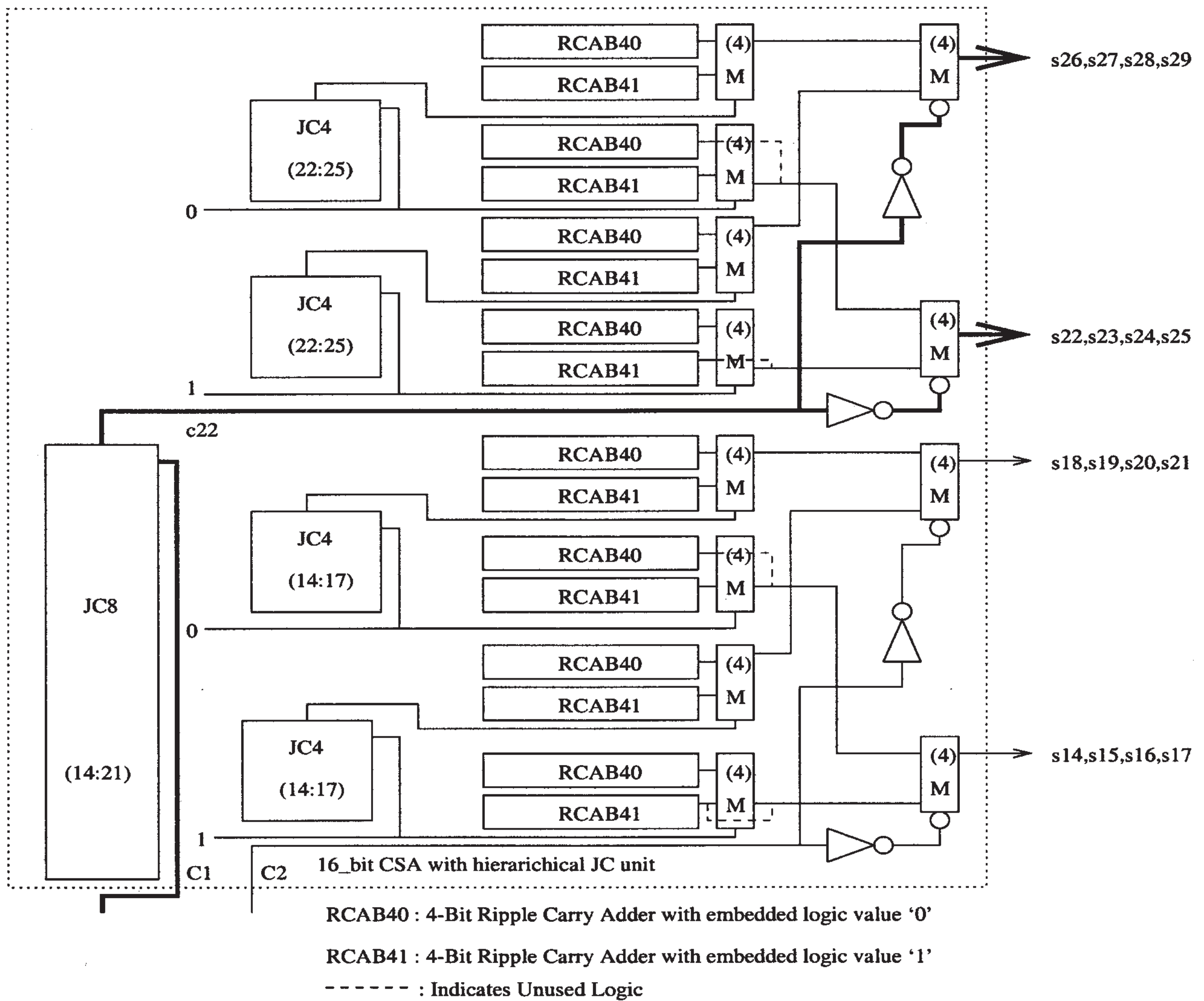


FIGURE 10 16-bit CSA schematic.

shows the schematic for the 2-bit carry select adder. It consists of the two ripple carry adder modules feeding a multiplexer. The carry-in input of one of the adders is set to logic "1" and that of the other is set to logic "0". The select signal of the multiplexer is driven by the carry-in bit, which selects either of the sum bits generated by the adders. If the carry-in is "0", then the sum bits generated by the adder whose carry-in input was set to "0" are selected; otherwise, the sum bits generated by the other adder are selected. The main cause of the faults in the case of carry-select adders is at the carry-in node of the ripple carry adders where it is set to either a logic "1" or "0" value. Consider a case when the carry-in bit is set to a logic value "1", a certain number of faults cannot be detected by test patterns that need the carry-in bit to be "0". Similarly, in the case where a carry-in of the ripple carry-adder is set to logic "0", test patterns requiring the carry-in bit to be set to value "1" can never be generated. Since several faults are interdependent, the resultant analysis yields undetected faults. These faults can be classified as redundant faults.

The remedy for such faults is to implement or hard code the constant logic in the circuit. The sum expression for an adder is described as $SUM = A \oplus B \oplus carry\text{-}in$. The modified description of the sum expression that hard codes the constant value is $SUM = A \oplus B$, when the carry-in is set to "0" and is $SUM = A \oplus \bar{B}$, when the carry-in is set to "1". Figure 9 shows the embedded carry-select adder schematic. The constant logic in the 4-bit and 8-bit carry select adder was embedded similarly.

Synthesis of a 16-bit Testable Carry Select Adder

Redundant logics were removed from the jumping carry units and the carry-select adders were modified by embedding the constant logics as described above. The ATPG process for the 32-bit fast adder resulted in 2604 total faults, 2600 detectable faults, and 4 redundant faults. The circuit was investigated again module by module. The faults were finally linked to the 16-bit carry-select adder with the hierarchical JC unit, as shown in Fig. 5. Figure 10

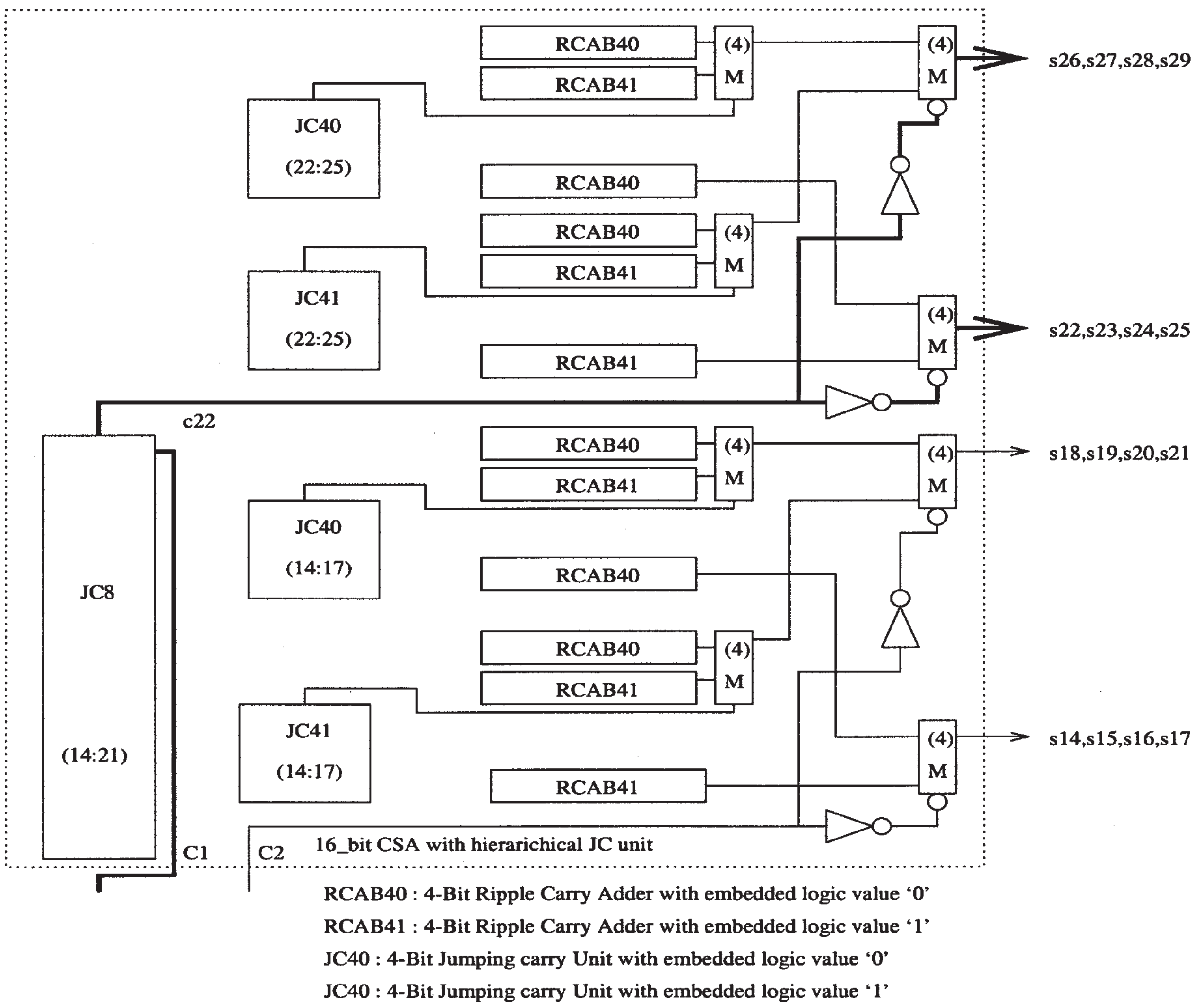


FIGURE 11 16-Bit CSA with redundancy removal.

shows the 16-bit carry-select adder module (embedded logic ripple carry adders). Note that two of the inputs C1 and C2 of this module are generated from the same circuit and hence, would always have the same values, i.e C1 = C2. Thus, test patterns that would require C1 and C2 to be different can never be generated in this configuration. The circuit can be divided into two 8-bit carry select adder sections and there is no direct dependency between these sections and therefore, the above mentioned problem does not arise. It is advisable not to have the two inputs of interdependent circuits generated from identical modules. Functionally, they yield the same result but in a testability analysis, these two inputs would be treated separately as the focus is on the module and some test patterns would require these inputs to be different. Furthermore, there is unused logic present in the circuit that needs to be removed. There are two multiplexers with the select line driven by a constant value "1" and other two multiplexers with the select line driven by a constant value "0". This results in only one of the inputs being selected all the time. Figure 11 shows the circuit with the unused logic being removed. The overall area is reduced as two ripple carry adders and two 2:1 4-bit multiplexers are removed. However, the circuit still has some constant logic that needs to be embedded. Figure 11 shows the circuits with embedded JC4 units. Figure 12 shows the schematic of the embedded JC4 units. Figure 13 denotes a simplified version of the 16-bit CSA. The decoder circuit is used instead of the multiplexers. It is observed that the output is either x_1 or x_2.

TABLE III Truth table

y_1	y_2	z_1	z_2	IN	OUT
0	0	x_1	x_1	0	x_1
0	1	x_1	x_2	0	x_1
1	0	x_2	x_1	0	x_2
1	1	x_2	x_2	0	x_2
0	0	x_1	x_1	1	x_1
0	1	x_1	x_2	1	x_2
1	0	x_2	x_1	1	x_1
1	1	x_2	x_2	1	x_2

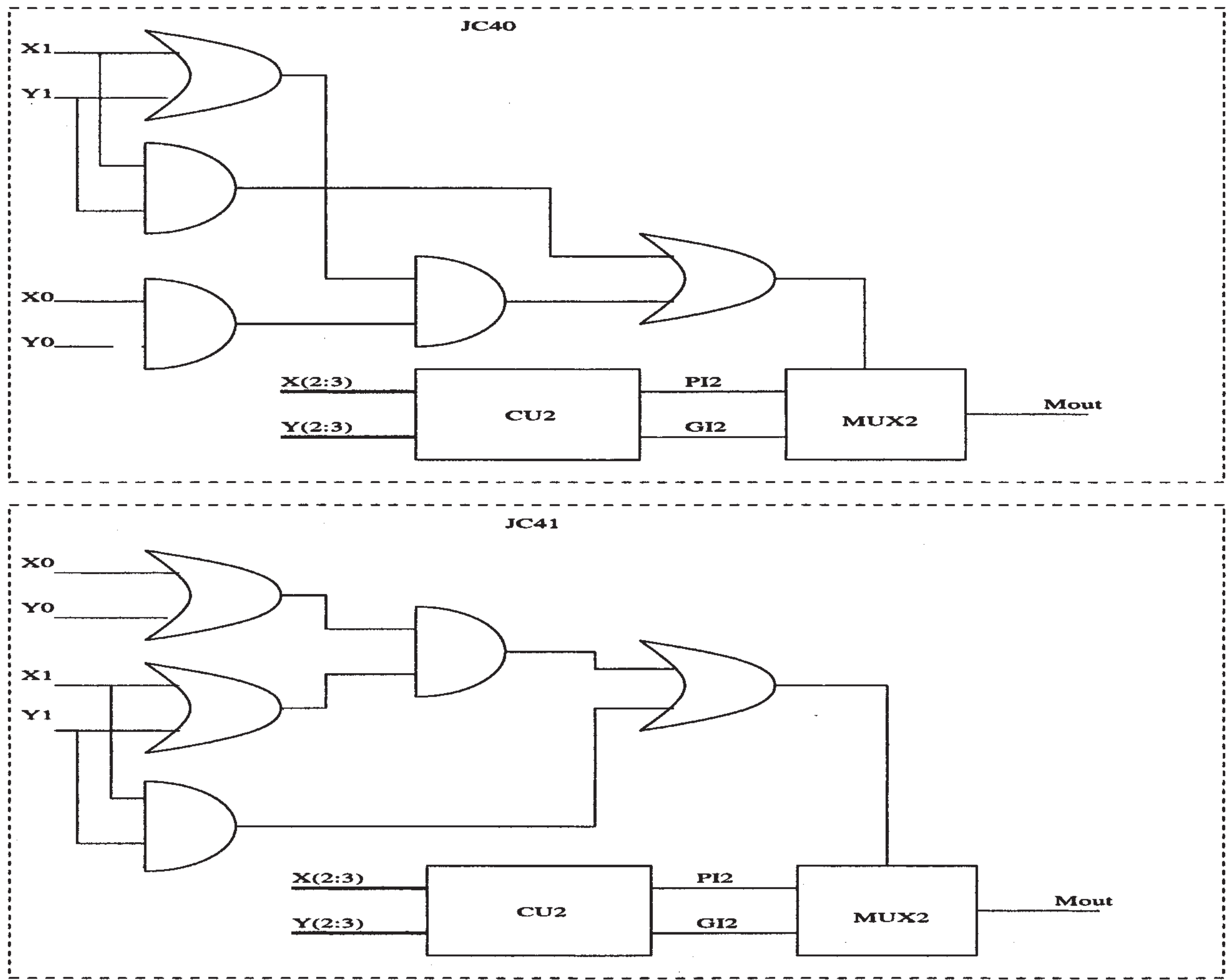


FIGURE 12 JC4 unit with embedded constant logic.

The outputs generated by the original ripple carry adders propagate through two levels of multiplexers; thus, there is an increase in the delay. The functionality of the circuit is not altered. The truth table for the 8-bit CSA section is listed in Table III. The truth table can be further simplified which removes intermediate logic and is listed in Table IV , where y_1 and y_2 are the outputs of the embedded logic "0" and "1" jumping carry units; x_1 and x_2 are the outputs of the 4-bit embedded logic "0" and "1" 4-bit ripple carry adders; z_1 and z_2 are the outputs of the decoder that are fed to the 2:1 4-bit multiplexer. The select line is driven by the signal IN which is the carry-in for the module. Signal OUT represents the 4-bit output of the multiplexer. The signal OUT gets the value of either x_1 or x_2 depending on the status of y_1, y_2 and IN. Thus, signals y_1, y_2 and IN can be decoded to generate the select signal. Figure 14 depicts the schematic of the decoder circuit. The modified 16-bit carry save adder is shown in Fig. 15 . The total number of faults in the modified 16-bit carry select adder are 1275 and all of these faults can be detected.

TABLE IV Modified truth table of decoder circuit

y_1	y_2	IN	OUT	Select
0	0	0	x_1	0
0	1	0	x_1	0
0	0	1	x_1	0
1	0	1	x_1	0
1	0	0	x_2	1
1	1	0	x_2	1
0	1	1	x_2	1
1	1	1	x_2	1

Table V lists the number of test patterns and the total number of detected faults. The entire design is now 100% testable for all possible stuck-at-faults.

TESTABILITY SYNTHESIS RESULTS

The synthesis for testability process ensured that the adder design was 100% testable. This was achieved by removing all the redundant logic in the circuit. The testable design was then subjected to synthesis for

16_bit CSA with hierarichical JC unit

RCAB40 : 4-Bit Ripple Carry Adder with embedded logic value '0'

RCAB41 : 4-Bit Ripple Carry Adder with embedded logic value '1'

JC40 : 4-Bit Jumping carry Unit with embedded logic value '0'

JC40 : 4-Bit Jumping carry Unit with embedded logic value '1'

FIGURE 13 16-bit simplified CSA schematic.

layout in 1.2 μm CMOS technology library. The critical path is shown in the Fig. 16. The worst case timing occurs at $S_{22}-S_{29}$. This delay time is measured between the insertion of input operands and arrival of the true output at the respective worst case sum bit. Table VI compares the area and timing of the untestable and the testable design.

Significant improvement in area and timing is observed from the data obtained from synthesis and simulation results. A 15% improvement in speed is observed, as worst

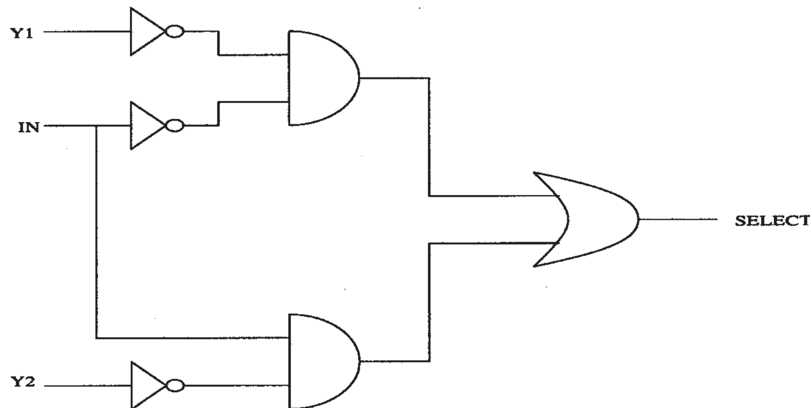


FIGURE 14 Decoder schematic.

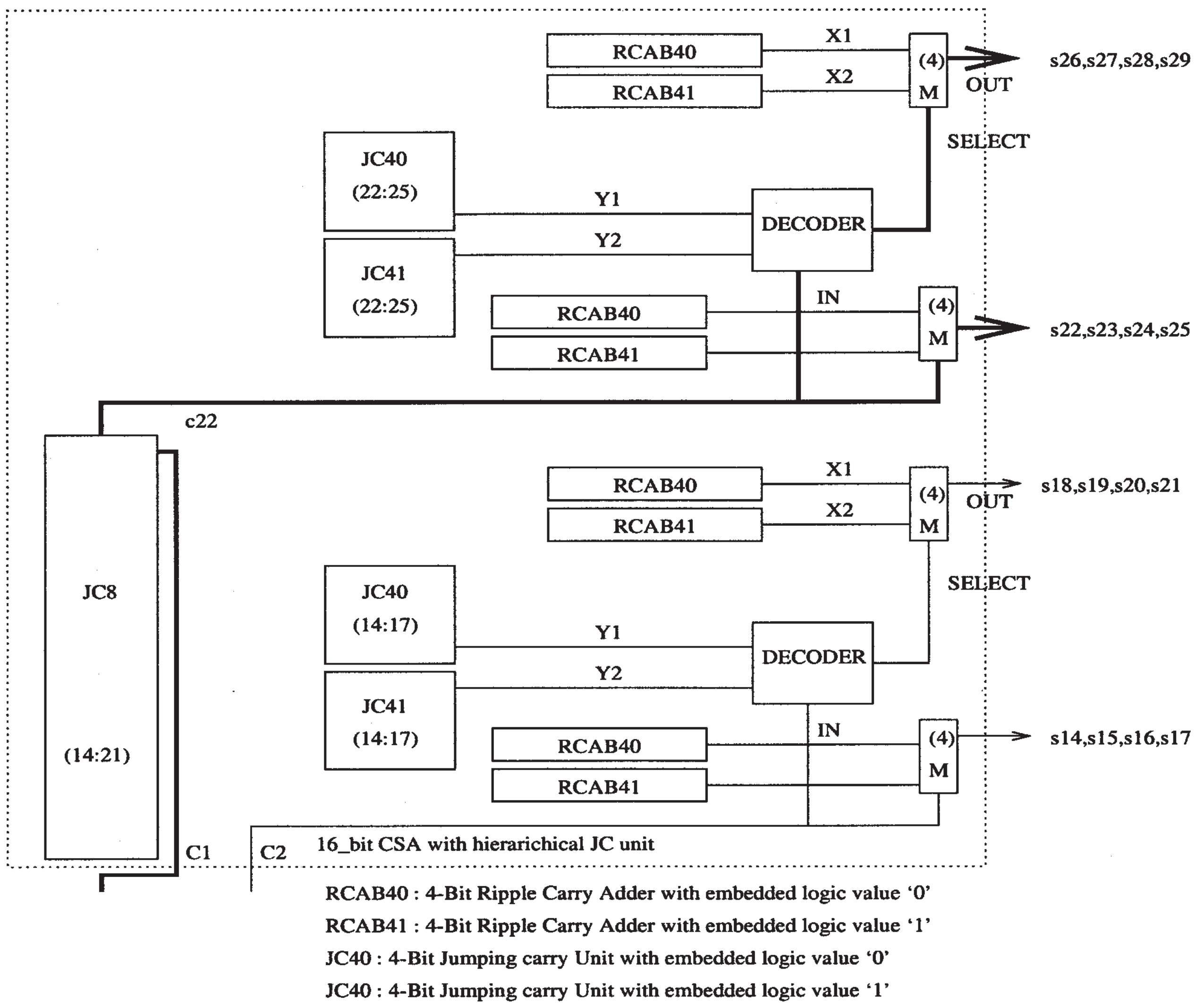


FIGURE 15 A 16-bit testable CSA.

case propagation delay time, for the original adder was 4.79 ns and the worst case propagation delay for the testable design is 4.09 ns. A 25% reduction in overall area is observed as improvement in area and timing are inherent features of synthesis for testability process. Removal of redundant logic results in removal of some elements in the critical path, reduction in levels and significant reduction in area.

The original 32-bit adder design had 3793 total possible faults, 3079 detectable faults and 714 redundant faults. The number of test patterns required to detect the testable faults were 365 and the fault coverage of the design was 81.17%. The testable 32-bit adder design has a total of 2072 possible faults. All these faults are detected by applying 179 test patterns; the fault coverage is 100%. In addition, all subcircuits JC2,

TABLE V ATPG results for 32-bit testable adder design

Module	Total faults	Detectable faults	Redundant faults	Fault coverage	Tests
JC2	25	25	0	100%	15
JC4	50	50	0	100%	19
JC8	103	103	0	100%	34
JC16	206	206	0	100%	61
CSA2	70	70	0	100%	13
CSA4	186	186	0	100%	23
CSA8	406	406	0	100%	37
CSA16	1275	1275	0	100%	92
Adder32	2072	2072	0	100%	179

FIGURE 16 A 32-bit testable adder structure.

TABLE VI Area and propagation delay results

Properties	Original design	Testable design
Area	3868 × 5719	3352 × 3404
t_p: s-bus25(22)	4.717 ns	3.73 ns
t_p: s-bus29(26)	4.748 ns	3.948 ns
t_p: s-bus29(29)	4.79 ns	4.024 ns
t_p: s-bus29(28)	4.75 ns	4.09 ns
t_p: s-bus31(31)	4.77 ns	3.56 ns
t_p: C_{32}	4.38 ns	3.674 ns
Max-t_p	4.79 ns	4.09 ns

JC4, JC8, CSA2, CSA4, CSA8 and CSA16 are testable with 100% fault coverage.

SUMMARY

Testability constraint plays a vital role in the synthesis process. In this paper, synthesis for a testable 32-bit fast binary adder with a conditional carry generation was presented. A comparison of ATPG, area and timing results of the original design and the testability synthesis

design has shown the effectiveness and efficiency of the proposed techniques. Testable 64-bit or 128-bit binary adders with conditional carry generation can be designed by making use of the generated testable modules, thus giving the present testable design an excellent scalability feature.

Acknowledgements

This work was supported in part by the US Air Force under contract F33615-93-C-1226.

References

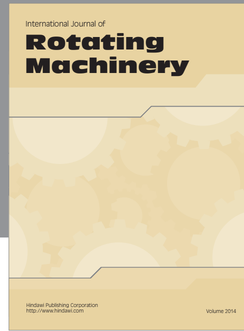
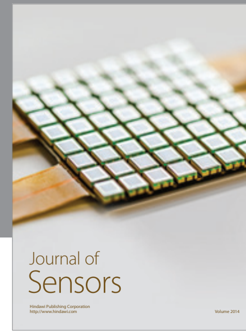
[1] Brand, D. (1983) "Redundancy and don't cares in logic synthesis", *IEEE Trans. Comput.* **31**(10), 947–952.
[2] Ghosh, A., Devadas, S. and Newton, R. (1992) Sequential Logic Testing and Verification (Kluwer Academic Publishers, Dordrecht).
[3] Bartlett, K., Brayton, R.K., Hachtel, G.D., Jacoby, R.M., Morrison, C.R., Rudell, R.L., Sangiovanni-Vincentelli, A. and Wang, A.R. (1988) "Multilevel logic minimization using implicit don't-cares", *IEEE Trans. Comput.-Aided Des.* **7**(6), 723–740.
[4] Bostick, D., Hachtel, G.D., Jacoby, R., Lightner, M.R., Moceyunas, P., Morrison, C.R. and Ravenscroft, D. (1987) "The boulder optimal logic design system", *Proc. Int. Conf. Comput.-Aided Des.* **November**, 62–65.
[5] Entrena, L.A. and Cheng, K.-T. (1995) "Combinational and sequential logic optimization by redundancy addition and removal", *IEEE Trans. Comput.-Aided Des.* **July**, 909–916.
[6] Sentovich, E.M., *et al.* "SIS: a system for sequential circuit synthesis", Memorandum No: UCB/ERL M92/41, U. C. Berkeley, May 1992.
[7] Pham, D., Chen, C.-I.H., (1998). "ABC: a VHDL/BLIF conversion software handler", Technical Report, Wright State University.
[8] Lo, J.C. and Fast, A. (1997) "Binary adder with conditional carry generation", *IEEE Trans. Comput.* **46**(2), 248–253.

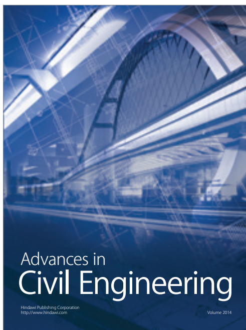
Authors' Biographies

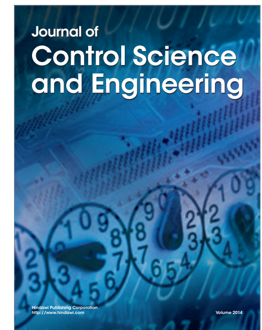
Chien-In Henry Chen received his B.S. degree from the National Taiwan University, Taiwan, in 1981, his M.S. degree from the University of Iowa, Iowa City, in 1986, and his PhD degree from the University of Minnesota, Minneapolis, in 1989, all in electrical engineering. Since joining Wright State University in 1989, he has worked primarily in computer-aided design, simulation and testing of very large scale integrated (VLSI) circuits, where he is currently a Professor. He has written over 60 publications in professional journal and conference proceedings and is a technical reviewer for various journals and conferences. He was a technical committee member of 1995 IEEE International ASIC Conference and Exhibit and was a plenary speaker in the 6th VLSI Design/CAD Symposium. He is a technical committee member of 2000 IEEE International ASIC/SOC Conference.

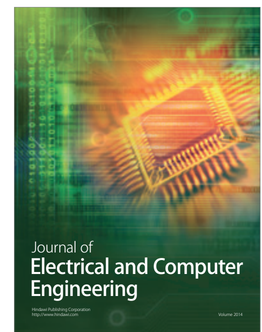
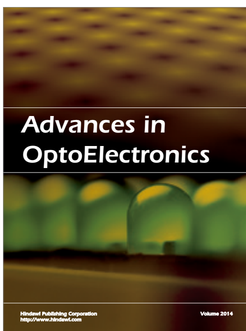
Mahesh Wagh received his B.E. degree in instrumentation from Bombay University, India, 1995 and his M.S. degree in electrical engineering from Wright State University, Dayton, Ohio, 1998. He was with Intel Corporation, Portland, Oregon and is now with LSI Logic, Milpitas, California. His research interests include VLSI circuit design, design for testability, and testability synthesis.

Journal of
Engineering

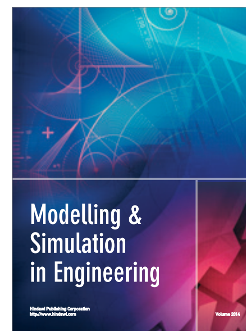
The Scientific
World Journal

International Journal of
Rotating
Machinery

Journal of
Sensors

International Journal of
Distributed
Sensor Networks

Advances in
Civil Engineering

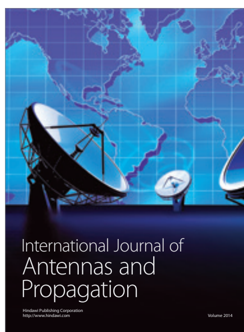
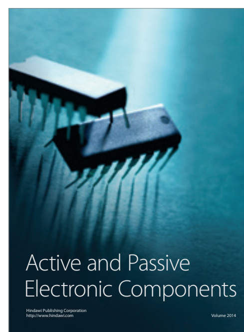
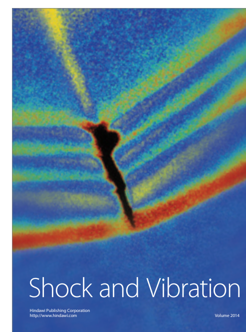
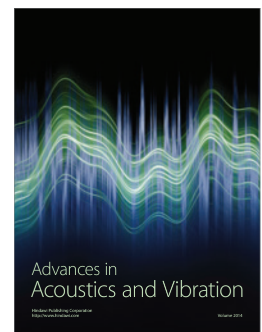
Hindawi
Submit your manuscripts at
http://www.hindawi.com

Journal of
Control Science
and Engineering

Journal of
Robotics

Journal of
Electrical and Computer
Engineering

Advances in
OptoElectronics

VLSI Design

International Journal of
Navigation and
Observation

Modelling &
Simulation
in Engineering

International Journal of
Aerospace
Engineering

International Journal of
Chemical Engineering

International Journal of
Antennas and
Propagation

Active and Passive
Electronic Components

Shock and Vibration

Advances in
Acoustics and Vibration