BCH-based Compactors with Data Compression for Test Responses

Taweesak Reungpeerakul[1], **Xiaoshu Qian**[2], and **Samiha Mourad**[3], Non-members

ABSTRACT

This paper presents a novel approach to compacting a test response for a multiple scan chains design. The compactor design is based on an extended $(n+1, k)$ BCH code, where k is the number of information bits and $n+1$ is the number of bits in the block. It can detect any odd number of single-bit errors or up to $2t$ single-bit errors, where t is a positive integer, $n-k \leq mt$, and $n+1 = 2^m$. Also we use a controllable mask to handle any number of unknown logic values (Xs) on test responses. We show how extra control data can be reduced by proposed compression technique. Compared to augmenting previous space compaction techniques with additional circuitry to mask any number of Xs, our approach can detect more single-bit errors with minimum number of compactor outputs. This leads to fewer tester channels, shorter test application time, and smaller test data volumes regardless of the Circuit Under Test (CUT) and fault models.

Keywords: BCH, Space Compactor, Test Response, Controllable Mask, Galois Field

1. INTRODUCTION

In the scan-based design for testability (DFT) of very large scale integration (VLSI) circuits, the test responses of the circuits are observed after applying test patterns from the tester. All observed test responses are matched with fault-free test responses in order to pass the test. As the complexity and number of scan cells on a System-On-Chip (SOC) are rapidly increasing, using a single scan chain is becoming impractical in term of test time. However, the use of multiple scan chains has been limited by the number of tester channels. Achieving good test quality for a complex SOC, with the limited test access imposed by the finite number of tester input/output pins, poses a big challenge. This begs for a technique that deals with a large number of scan chains while reducing the connections to the tester pins. This concept is illustrated in Fig. 1.

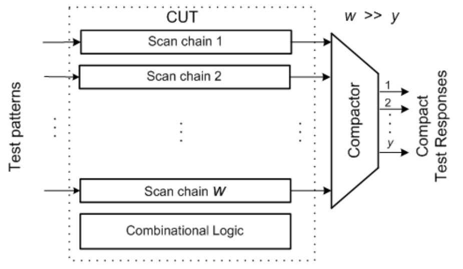


Fig.1: *Use of Compactors for Test Response*

Many techniques have been proposed to compact test responses. They can be classified into three categories:

1) Time compactors: These have an infinite impulse response. The compaction is performed by a linear finite state machine such as the multiple input signature register (MISR). Time compactors can achieve high compaction ratios. However, all values in the CUT's test responses must be known [1-3].

2) Space compactors: They are combinational circuits constructed from Exclusive-OR (XOR) networks that compact w scan chains of the CUT into the compactor's y outputs [4-6], where $y \ll w$.

3) Finite memory compactors (FMCs): These have a finite impulse response. The compactors are comprised of XOR networks and memory elements [7-8]. The organization of the XOR networks is based on the X-compact approach [5].The memory elements are used to hold and shift out the compacted test responses.

The focus of this paper falls under the Space Compactor category. One of the main challenges when using a space compactor is that the error values can propagate from the compactor's inputs to its outputs. The X-compact technique uses parity check matrices to detect dual and any odd number of single-bit errors. It cannot guarantee the detection of an even number of errors greater than two. Another method, i-compact [4], uses a checking matrix based on the Hamming code, which is extended to the SEC-DEC code, in order to detect 1) up to d_{min} -1 single-bit errors, or 2) up to e errors in the presence of up to x unknowns, where $e + x < d_{min}$ and d_{min} is Hamming distance.

Another challenge involved in processing test re-

Manuscript received on September 15, 2008 ; revised on , .

[1]The author is with Department of Computer Engineering, Prince of Songkla University, Thailand, E-mail: rtaweesak@coe.psu.ac.th

[2] The author is with Intel Corporation, SC12-512, 2200 Mission College Blvd., Santa Clara, CA 95054 E-mail: xiaoshu.qian@intel.com

[1,3] The authors are with Santa Clara University, 500 El Camino Real, Santa Clara, CA 95053 E-mail: treungpeerakul, smourad@scu.edu

sponses is the presence of many unknown logic values (Xs) which can affect the results of the compactors:

- If there are Xs in a scan chain that is fed into a time compactor, especially one built with a MISR, the entire signature will be corrupted.
- Space compactors are limited to how well they are able to handle Xs. For example, X-compactor guarantees detection of only one X on each scan-out cycle when one or two scan chains produce errors [5]. An i-compact guarantees the detection of up to d_{min}-1 Xs when there is no error in the same scan-out cycle.
- In the case of the FMC, because the XOR network is implemented using an X-compact approach, only a few numbers of Xs can be handled and hence some of known values in the test responses may become unobservable.

In order to handle test responses with a wide range of Xs, a compaction scheme has to be combined with a masking mechanism to prevent the Xs from entering the compactor. Several approaches have been proposed to this method, such as using masks to block out the Xs [9-15], using the X tolerant compactor circuitry [7-8], synthesizing linear selector to suppress unknown states [16], or reading the MISR out before an X arrives and masking selected unload values to prevent X reaching a compactor, called OPMISR technique [1]. For the i-compact, if the number of Xs increases, the compaction ratio will decrease significantly. In this paper, we not only propose an alternative solution to mask any number of Xs on each scan-out cycle without reducing compaction ratio, but also present an efficient compression to significantly reduce control data overhead.

We propose a class of BCH-based compactors with controllable masks that guarantee the detection of any odd number and up to $2t$ even number of single-bit errors. Here t is a design parameter determined by the BCH code [17] that is used. It can also handle any number of Xs in the test responses. The advantages of our techniques include ability to detect more single-bit errors, capability to handle any number of Xs independent of fault models, minimum number of compactor outputs, independence of the CUT for a given number of scan chains, and reduction of the test application time and the test data volume.

The rest of this paper is organized as follows Section 2 describes the implementation of BCH-based compactors. In the Section 3, the architecture of a controllable mask circuit is presented. Section 4 describes the compression used to encode the control data of the mask. In Section 5, results are presented. Conclusions are given in Section 6.

2. BCH-BASED COMPACTORS

We propose a BCH-based compactor that is guaranteed to detect both an odd number of errors and even number of errors up to $2t$, where t is a positive integer. Our work is based on the following theorem.

Theorem The compaction matrix based on the extended BCH code guarantees the detection of up to d_{min} - 1 single-bit errors and any odd number of single-bit errors, where d_{min} is the minimum Hamming distance of the extended BCH code.

Let H be the parity check matrix of the extended BCH code. If a test vector i containing up to d_{min}-1 single-bit errors, is compared to the reference response vector i_0, the binary sum of i and i_0, $v = i + i_0$, has a weight less than d_{min}. Therefore, v is not a codeword because any non-zero codeword must have weight d_{min} or larger. Hence, $vH^T \neq 0$ or $(i{+}i_0)H^T \neq 0$. The latter expression can be written in the form $iH^T \neq i_0H^T$. This shows that the compactor can detect up to d_{min}-1 single-bit errors.

Similarly, if a test vector i contains an odd number of single-bit errors, then the weight of $v = i{+}i_0$ is odd. Since the last row of H contains all 1s and the last entry of vH^T is the sum of an odd number of 1s, at least the last entry of vH^T is non-zero. Again, we have $vH^T \neq 0$, or $iH^T \neq i_0H^T$. This shows that the compactor can detect up to any odd number of single-bit errors.

In the following steps, we detail a procedure for generating a compact matrix based on the generator of a BCH code. Examples will be given to illustrate the procedure.

1. Select a generator polynomial of order n such that $n \geq w$ -1, where w is the number of scan chains, and that the compactor can detect the largest even number of expected errors ($2t$). The generator polynomials for BCH codes are available in many coding books, for example [17].
2. Construct the Galois field $GF(2^m)$, where $n = 2^m$ - 1, $m \geq 3$.
3. Form the parity check matrix H of the BCH code. This parity check matrix, when used as a compact matrix, enables us to detect up to $2t$ single-bit errors.
4. Extend the parity check matrix H to detect any odd number of single-bit errors.
5. Perform elementary row operations on matrix H to give each column an odd parity. This is achieved by changing only the values of the last row of the parity check matrix. The purpose of this step is to minimize the number of XOR gates in the compactors. Note that the elementary row operations are not affected by whether vH^T is zero or not. The resulting matrix specifies the connections of the XOR networks for BCH-based compactor.

Example I: $n = 15$, $k = 11$, and $t = 1$

We use the (15, 11) BCH code and extend it to a (16, 11) code in order to illustrate the procedure

Table 1:*Galois Field $GF(2^4)$ with $p(\alpha)=\alpha^4+\alpha+1=0$*

	α^3	α^2	α	1	Comments
1	0	0	0	1	1
2	0	0	1	0	α
3	0	1	0	0	α^2
4	1	0	0	0	α^3
5	0	0	1	1	$\alpha^4 = \alpha + 1$
6	0	1	1	0	$\alpha^5 = \alpha(\alpha^4) = \alpha^2+\alpha$
7	1	1	0	0	$\alpha^6 = \alpha(\alpha^5) = \alpha^3+\alpha^2$
8	1	0	1	1	$\alpha^7 = \alpha(\alpha^6) = \alpha^3+\alpha+1$
9	0	1	0	1	$\alpha^8 = \alpha(\alpha^7) = \alpha^2+1$
10	1	0	1	0	$\alpha^9 = \alpha(\alpha^8) = \alpha^3+\alpha$
11	0	1	1	1	$\alpha^{10} = \alpha(\alpha^9) = \alpha^2+\alpha+1$
12	1	1	1	0	$\alpha^{11} = \alpha(\alpha^{10}) = \alpha^3+\alpha^2+\alpha$
13	1	1	1	1	$\alpha^{12} = \alpha(\alpha^{11}) = \alpha^3+\alpha^2+\alpha+1$
14	1	1	0	1	$\alpha^{13} = \alpha(\alpha^{12}) = \alpha^3+\alpha^2+1$
15	1	0	0	1	$\alpha^{14} = \alpha(\alpha^{13}) = \alpha^3+1$

outlined above. The generator polynomial for the (15, 11) BCH code is $g(X) = X^4 + X + 1$. The elements of the Galois field $GF(2^4)$ are shown in Table 1.

By rotating the second column of Table 1, we form the matrix, *H*. We then add 0s on the last column and 1s on the last row to *H* in order to obtain the parity matrix, H_{ext}, for the extended (16,11) BCH code.

$$H = \begin{pmatrix} 1&0&0&0&1&0&0&1&1&0&1&0&1&1&1 \\ 0&1&0&0&1&1&0&1&0&1&1&1&1&0&0 \\ 0&0&1&0&0&1&1&0&1&0&1&1&1&1&0 \\ 0&0&0&1&0&0&1&1&0&1&0&1&1&1&1 \end{pmatrix}_{4\times15}$$

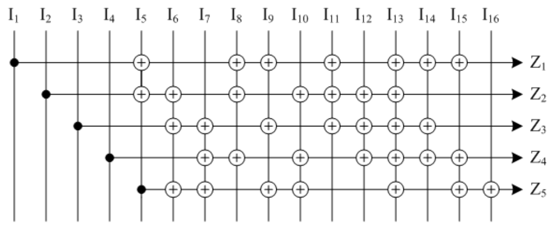
$$H_{ext} = \begin{pmatrix} 1&0&0&0&1&0&0&1&1&0&1&0&1&1&1&0 \\ 0&1&0&0&1&1&0&1&0&1&1&1&1&0&0&0 \\ 0&0&1&0&0&1&1&0&1&0&1&1&1&1&0&0 \\ 0&0&0&1&0&0&1&1&0&1&0&1&1&1&1&0 \\ 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \end{pmatrix}_{5\times16}$$

The final compaction matrix, H_{comp}, is obtained by applying some elementary row operations to H_{ext}. The operations involve adding appropriate rows to the last row in order to eliminate any 1s in the last row so that each column contains an odd number of 1s.

$$H_{comp} = \begin{pmatrix} 1&0&0&0&1&0&0&1&1&0&1&0&1&1&1&0 \\ 0&1&0&0&1&1&0&1&0&1&1&1&1&0&0&0 \\ 0&0&1&0&0&1&1&0&1&0&1&1&1&1&0&0 \\ 0&0&0&1&0&0&1&1&0&1&0&1&1&1&1&0 \\ 0&0&0&0&1&1&1&0&1&1&0&0&1&0&1&1 \end{pmatrix}_{5\times16}$$

The circuit in Fig. 2 specifies a compactor based on this compaction matrix. There is an XOR-connection from input I_i to output Z_j if and only if the value of row *j* and column *i* in the matrix is 1. For example, the value of row 1 and column 5 in the matrix is 1, so there is an XOR-connection from inputs I_5 to output Z_1. After connecting all the related inputs, we have $Z_1 = I_1 \oplus I_5 \oplus I_8 \oplus I_9 \oplus I_{11} \oplus I_{13} \oplus I_{14} \oplus I_{15}$.

The compactor shown in Fig. 2 can support up to sixteen inputs from scan chains, I_1, I_2, …, I_{16}, and has five outputs, Z_1, Z_2,…, Z_5. More generally,



I_1 I_2 I_3 I_4 I_5 I_6 I_7 I_8 I_9 I_{10} I_{11} I_{12} I_{13} I_{14} I_{15} I_{16}
Z_1
Z_2
Z_3
Z_4
Z_5

Fig.2: *The XOR network of (16,11) BCH-based compactor*

the relationship between the number of inputs and outputs, in the case of $t = 1$, is calculated by the following equation:

$$w_{max} = \sum_{y-i}^{y} C(y, i)$$

where $C(y, i) = \frac{y!}{(y-i)! * i!}$ and *i* is odd number.

For the above example, the number of compactor outputs is 5. The maximum number of scan chains is C(5,1)+C(5,3)+C(5,5) = 16. The compactor guarantees the detection of two and any odd number of single-bit errors.

Example II: $n = 15$, $k = 7$, and $t = 2$

As VLSI circuits become larger and more complex, detection of more single-bit errors is necessary. The BCH-based compactors extend error detection capability to any odd number of single-bit errors and up to $2t$ number of single-bit errors for any integer *t* provided $mt + 1$ compactor outputs are needed in the worst case, where *m* is related to the maximum number *n* of compactor inputs by $n + 1 = 2^m$. For example, by choosing $t = 2$, we can obtain a compactor that detects any odd number of single-bit errors and up to 4 single-bit errors. The compactor allows up to 16 ($=2^m$) inputs and uses 9 outputs ($= mt + 1$). We now repeat the procedure outlined in Section 2:

- Select generator polynomial $g(X) = X^8 + X^7 + X^6 + X^4 + 1$, based on $t = 2$ and $n = 15$.
- Generate all elements of $GF(2^4)$ from *g(X)*.
- Arrange bit patterns of all $GF(2^4)$ elements column by column. Next, add an all-0 last column and an all-1 last row in order to form a parity check matrix for the (16, 7) BCH extended code.
- Applying elementary row operations to the parity check matrix above to form H_{comp}.

$$H_{comp} = \begin{pmatrix} 1&0&0&0&0&0&0&0&1&1&0&1&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&1&1&0&1&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0&1&1&0&1&0&0 \\ 0&0&0&1&0&0&0&0&0&0&0&1&1&0&1&0 \\ 0&0&0&0&1&0&0&0&1&1&0&1&1&1&0&0 \\ 0&0&0&0&0&1&0&0&0&1&1&0&1&1&1&0 \\ 0&0&0&0&0&0&1&0&1&1&1&0&0&1&1&0 \\ 0&0&0&0&0&0&0&1&1&0&1&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&1&0&0&1&1&1&1&1 \end{pmatrix}_{9\times16}$$

Figure 3 shows the XOR network related to the parity check matrix H_{comp}.

The compactor based on H_{comp} supports up to sixteen inputs from scan chains and requires nine outputs. More generally, Table 2 shows the numbers of compactor outputs for the range of w scan chains in Column 1. The compactor guarantees the detection of two, four, and any odd number of single-bit errors. There is no presence of Xs because all Xs have to be forced to known values by controllable masks described in the next section.

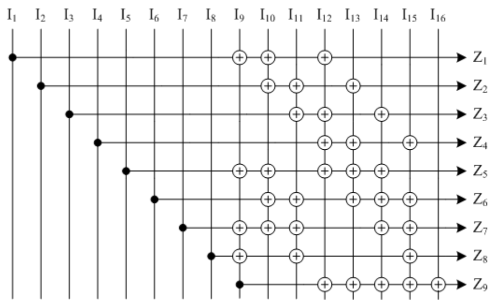

I_1 I_2 I_3 I_4 I_5 I_6 I_7 I_8 I_9 I_{10} I_{11} I_{12} I_{13} I_{14} I_{15} I_{16}
Z_1
Z_2
Z_3
Z_4
Z_5
Z_6
Z_7
Z_8
Z_9

Fig.3: *The XOR network of (16,7) BCH-based compactor*

Table 2: *Range of Observable Scan Chains of Compactors for n=15, k=7, and t=2*

Scan Chains (w)	Compactor Outputs (y)
9-16	9
17-32	11
33-64	13
65-128	15
129-256	17
257-512	19
513-1,024	21

3. CONTROLLABLE MASKS CIRCUIT

The output data from the scan chains represents the results of the tests and consists of 0s, 1s, or Xs. Frequently, the source of an X cannot be identified until after the part has been manufactured. Causes of Xs include un-initialized and uncontrollable bistables, bus contention, floating buses, multiple clock domains, and inaccurate simulation models [9]. In order to mask any number of Xs, an additional unit can be inserted between the scan chains and the compactor. We propose the controllable mask circuit shown in Fig. 4. The mask data (M), and the control data (C) are used to assure the outputs of the circuit feeding the BCH-based compactor (I_1, I_2, ..., I_w) are all X-free. The mask and control data are used to force all Xs in test responses to 1s. We illustrate by the example in Table 3. The first column gives the scan chain numbers. The second column shows the test responses from each scan chain. The control and mask data are listed in the last two columns of the table, respectively. The control data of scan chain 1, 2, and 4 are 1s because there is at least one X in the corresponded scan chains. There are no Xs in scan chain 3 or 5 so the control data is 0s. The mask data consists of both 0s and 1s. The mask data has a 1 in the position corresponding to any X, otherwise it will be a 0.

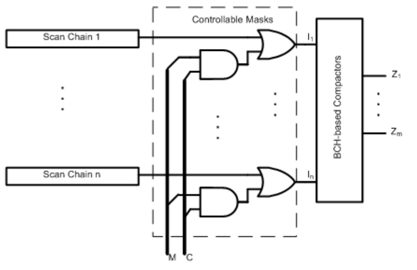


Fig.4: *The Architecture of Controllable Masks*

In the actual test response, the number of Xs is usually less than 1 %. This means that there are many 0s generated in the control and mask data that have to be counted as overhead. Therefore compressing both data is a natural solution and will be covered in the next section.

The control and mask data can be supplied in many different ways including from an embedded memory by a BIST controller, from an ATE through a serial-to-parallel unit, etc. The advantages of controllable masks are: 1) the ability to handle any number of Xs; 2) independence from a fault model; and 3) no modification on test patterns.

Table 3: *An Example of Mask Data Generation and the Corresponding Control Data*

Scan Chain	Test Responses	Control Data	Mask Data
1	0X01X11X	1	01001001
2	XX1100XX	1	11000011
3	11100101	0	-
4	X11111XX	1	10000011
5	00011101	0	-

4. COMPRESSION OF CONTROLLABLE MASK

In order to minimize the control and mask data required by the circuit in Fig. 4, the combined compression technique based on a uni-phase run-length code and binary coding briefly presented in [18]. Our compression technique will detail in this section.

As mentioned in Section 3, we need to generate the control and mask data in order to mask any number of Xs. We now investigate the distribution of 0s run-length on both parts. We illustrate the distribution of 0s run-length on ISCAS s13207 benchmark in Fig. 5.

Figure 5 a). shows the distribution of control data. The frequency of runs on 0s is very high for the length less than 6. We then will propose the appropriate method by modifying Kay coding [19] in next paragraph. Figure 5 b). shows the distribution of mask data. Most of the data is distributed in the range of 0s runs. We simply use the binary coding of run-length of 0s to encode mask data. The efficiency of the proposed technique will be compared with other methods in the Section 5.

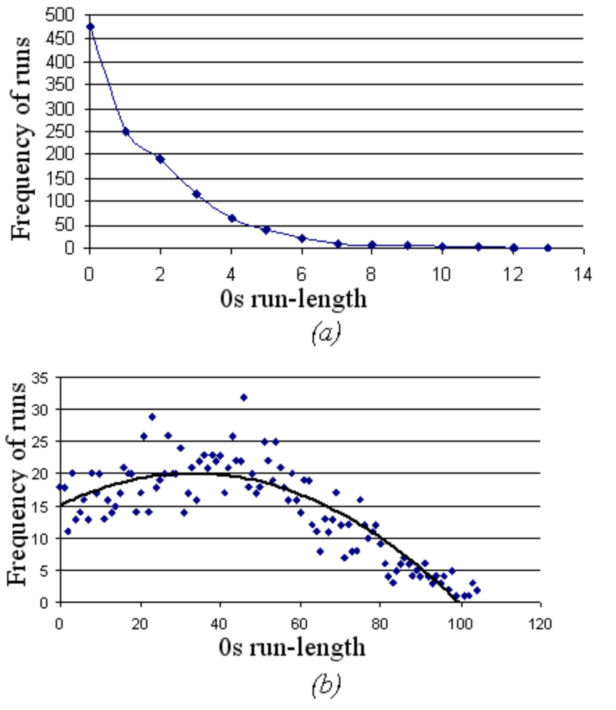


Fig.5: *The Distribution of 0s Run-length for S13207 a) Control Data b) Mask data*

The compression method of the control data is modified the Kay coding [19]. Table 4 shows an example of proposed coding for group size $s = 16$. The first column lists run-length of 0s followed by a 1. If no 0 precede the 1, the run-length is zero. The prefix, tail, and codeword are showed in the second, third, and forth, respectively. The encoding of a control data is illustrated in Fig. 6.

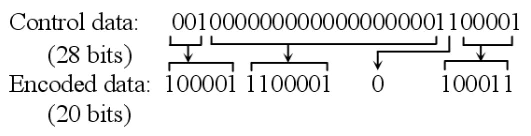


Fig.6: *Control Data Encoding Procedure*

Figure 7 shows a block diagram of the decoder for decompressing the encoded control data. It is implemented by finite-state-machine (FSM) and 2 counters. Counter 1 is a $log_2 j$ - bit counter where j is the maximum number of 0s run-length. Counter 2 is a j-bit counter. Input signals of the decoder are *data_in* and *clk*. The *data_in* is the encoded data input. Output signals of the decoder are *rdy*, *data_out*, and *valid*. The *rdy* is used to indicate that the decoder is ready to receive next data on *data_in*. The *data_out* is the decoded data output and the *valid* indicates that the output data is available on *data_out*. The signal inc_1 (inc_2) is used to increase the counter 1 (counter2) and rch_1 (rch_2) indicates that the counter 1 (counter 2) has finished counting. The signals *ld* and *one_in* are used to load and shift of the encoded data into the counter 2, respectively.

Table 4: *An Example of Proposed Coding*

Run-length	Prefix	Tail	Codeword
0	0	-	0
1	10	0000	100000
2		0001	100001
3		0010	100010
...			
16		1111	101111
17-32	110	bbbb	110bbbb
33-48	1110	bbbb	1110bbbb
...	...	...	...

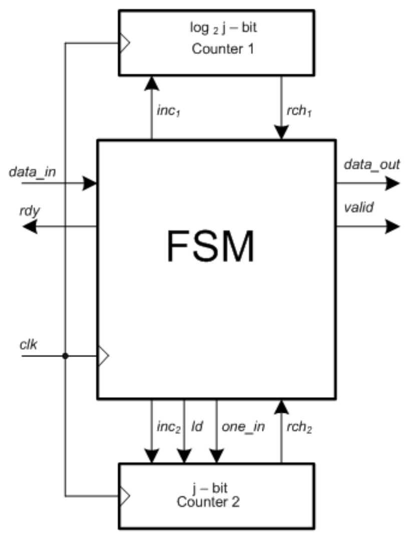


Fig.7: *Block Diagram of the Decoder*

The state diagram for the decoders FSM is shown in Fig. 8. State S_0 is the idle state. Once the *rdy* is set to high, then it becomes state S_1 for starting to decode the beginning to each codeword. State S_2 is used to decode all zero run-length bits as long as the data of *data_in* is 0. The logic 1 of *data_in* will change the current state from S_1 or S_2 to state S_3 which is the beginning of the new codeword other than zero run-length codeword. States S_3 - S_6 are used to decode the prefix part of the codeword by having 0 of *data_in* to identify the end of prefix and entering state S_6. States S_7 - S_9 are used to decode the tail part of the codeword.

The second part of compression simply uses binary coding of run-length of 0s. The width of codeword is computed by upper bound, U, of maximum run-length of mask data. Since each group of data consists of at least the 1, the maximum run-length is related to maximum scan length, l_{max}. The upper bound is given by 2$*$ l_{max} -2. The width of the codeword is then given by $\lceil log_2(2l_{max}\text{-}2)\rceil$. The encoding of mask data is illustrated Fig. 9 with l_{max} = 16. The width of the codeword is given by $\lceil log_2(32\text{-}2)\rceil$= 5.

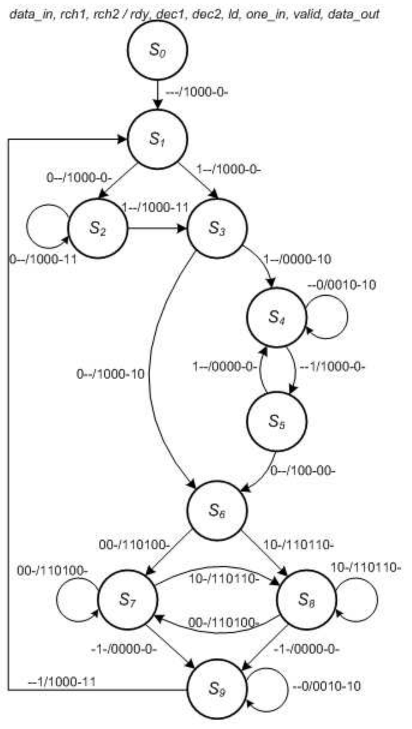


Fig.8: *State Diagram of the FSM to Decompress the Encoded Control Data*

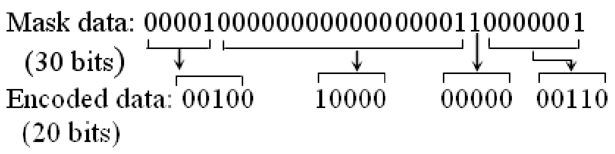

Mask data: 000010000000000000000110000001
(30 bits)
Encoded data: 00100 10000 00000 00110
(20 bits)

Fig.9: *Mask Data Encoding Procedure*

A block diagram of the decoder to decompress the encoded mask data is the same as the control data, as shown in Fig. 7. The state diagram for the decoder of the mask data is shown in Fig. 10. State S_0 is the idle state. Once the *rdy* signal is set to high, then FSM moves to state S_1 that initiates the decoding of each codeword. The codeword has a fixed length, without the prefix and tail, and is decoded in states S_1-S_4.

The proposed compression technique is more effective than Kay [19], FDR [20], and Golomb [21] tech-

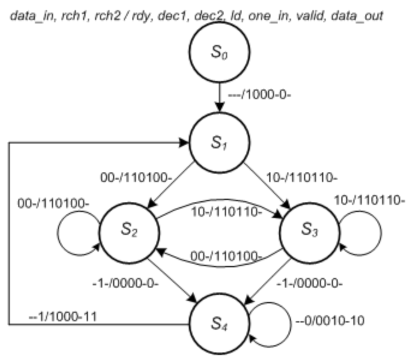


Fig.10: *State Diagram of the FSM to Decompress the Encoded Mask Data*

niques as shown in the next section. The optimality of the coding technique is demonstrated in the Appendix to the paper.

5. EXPERIMENTAL RESULTS

a) Compactor Input/Output Pins Relationship

Using the constructive approach detailed in Section 2 and the data from [5] and [9], we constructed Table 5. The first column gives the number of compactor outputs and the next two columns show the number of scan chains that can be handled by an X-compact with a new LFSR [9] and by the proposed method, respectively. Both methods are guaranteed to detect two and any odd numbers of errors while handling any number of unknowns (Xs), but the proposed compactor can process about twice as many observable scan chains as [9] with the same number of output pins. The fourth column of Table 5 shows the percentage improvement of proposed approach over the one in [9], as defined by:

$$\left\{\frac{\text{max.chains of our approach-max.chains of [9]}}{\text{max.chains of [9]}}\right\}\times 100$$

The maximum number of observable scan chains in the second column and the third column are plotted versus the compactor outputs in Fig. 11. It can be easily seen that the proposed approach can support a larger number of observable scan chains for all values in the first column of the table. It is clear that for the same number of compactor outputs, the proposed approach can handle more scan chains than [9].

In the next experiment, we used our BCH-based compactor to replace the X-compact in the Block Compactor [7] and applied to the controllable mask to replace any Xs in the test responses from the scan chains. The maximum number of scan chains is computed for a given number of compactor outputs in both cases. The results are shown in Table 6. In both cases, the Block Compactor 1) is guaranteed to

Table 5: *Range of Observable Scan Chains*

Compactor Outputs	Scan Chains		Improvements
	X-compact with new LFSR[9]	Our Approach	
5	5-10	9-16	60
6	11-20	17-32	60
7	21-35	33-64	82.86
8	36-56	65-128	128.57
9	57-126	129-256	103.17
10	127-252	257-512	103.17
11	253-462	513-1,024	121.65
12	463-792	1,025-2,048	158.59
13	793-1,716	2,049-4,096	138.69
14	1,717-3,432	4,097-8,192	138.69

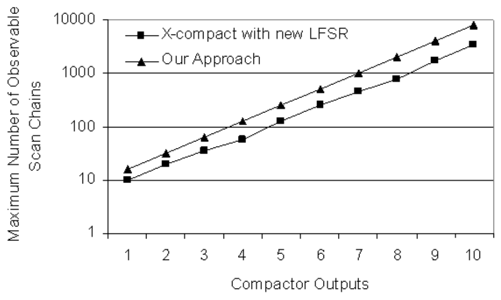


Fig.11: *The Maximum Numbers of Observable Scan Chains Obtained by 2 Approaches*

detect two and any odd number of errors; 2) can handle any number of Xs; and 3) has 4 scan cycles. It is clear that in all cases, the proposed approach has many more observable scan chains. Thus, the proposed approach can be applied to the finite memory compactors in order to improve a large number of observable scan chains and the overall efficiency.

b) Test Area Overhead

To estimate the area overhead of the proposed technique, we used a commercial tool to compile and optimized several benchmark circuits from the ISCAS89 suites with the results listed in Table 7. In this table, the first column lists the cores used in the experiment. The next two columns give the numbers of flip-flops and of scan chains, respectively. The next three columns give the size of the core, of the controllable mask circuit, and of the BCH-based compactor. The last two columns give the percentage of area overhead represented by the controllable mask

Table 6: *Maximum Numbers of Observable Scan Chains on Block Compactor Implementation*

Compactor Outputs	Scan Chains	
	X-compact with new LFSR(W=5)	Our Approach
4	1,092	8,192
6	10,626	2,097,152
8	50,344	536,870,912

Table 7: *Area Overhead*

Core	Numbers of		Numbers of Gates (2-input NAND)			Overhead	
	FF	Chain	Core	Ctrl. mask	Comp-actor	Ctrl. mask	Comp-actor
S9234	211	8	3,391	24	30	0.71	0.88
S13207	638	12	7,977	36	57	0.45	0.71
S15850	534	10	7,710	30	39	0.39	0.51
S35932	1728	34	24,476	102	225	0.42	0.92
S38417	1636	30	23,009	90	207	0.39	0.90
S38584	1426	28	23,353	84	180	0.36	0.77

circuits and the BCH-based compactors. The area overhead values are all less than 1 percent and hence we consider them negligible.

c) Compression Effectiveness

The control and mask data are encoded by the proposed compression and compared to Kay, FDR, and Golomb techniques. The reduction of test data as obtained by the different techniques is computed using the compression effectiveness, C.E., defined by:

$$\frac{controlandmaskdata - Encodeddata}{controlandmaskdata} \times 100$$

From the definition of C.E., the techniques are evaluated by¡/font¿ comparing the size of the encoded data to the size of the raw data. The encoding technique with the highest C.E. will be the most effective. All benchmark circuits in the first column of Table 8 were examined in this study for different percentages of X values, 0.01-0.95 %, in the test responses. We show the example of the result for the S13207 benchmark, as shown in Fig. 12. In the figure, the proposed codings C.E. is higher than Golombs and significantly greater than FDRs and Kays for percentages in range of 0.10-0.95 %. This example is representative of all benchmarks.

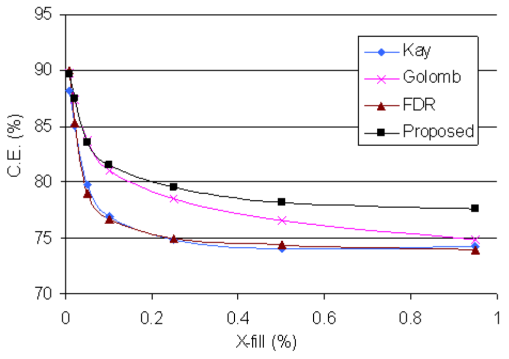


Fig.12: *C.E. of Four Encoding Techniques on S13207.*

6. CONCLUSIONS

In this paper, we addressed the two main problems usually encountered with test response compaction: error detection and unknown response bit masking. We propose BCH-based compactors. First, the check matrix of an (n, k) BCH code generated from the Galois field is extended to the (n+1, k) code. A controllable mask circuit is placed at the output of the scan chains in order to prevent unknown values to affect the operation of the compactor. The control and mask data are compressed to reduce the test data overhead. The BCH-based compactors with controllable masks are guaranteed not only detecting up to $2t$ single-bit errors (where $2t \leq d_{min}$-1; d_{min} is Hamming distance and t is any positive integer), but also any odd number of single-bit errors. We have demonstrated the feasibility and effectiveness of our approach. For a given number of outputs, the compactor can handle more scan chains than [9]. This leads to fewer tester channels, smaller test data volumes, and shorter test application times. Moreover, the control and mask data can be effectively compressed by the proposed compression technique as compared to using the Golomb, Kay, and FDR codes.

APPENDIX

[Optimality of Integer Codes] The purpose of this appendix is to heuristically demonstrate the optimality of some encoding schemes in Section 4 and thus to provide some guidelines for choosing encoding schemes to match statistical distributions of the data. The main reference used in this appendix is [22].

Suppose that we want to compress an integer random variable X. The average number of bits required is lower bounded by

$$H(X){=}E\,\lg\left(\frac{1}{p(X)}\right){=}\sum_n Pr(X=n)\log_2\left(\frac{1}{Pr(X=n)}\right).$$

Thus, if the number of bits required for encoding an integer n is given by $l(n)=\log_2(\frac{1}{Pr(X=n)})$, the minimal average length is achieved since in this case

$$\overline{l(X)}=El(X)=\sum_n Pr(X=n)l(X=n)=H(X).$$

Binary code

When X is of uniform distribution in [0, 1, , 2^m-1], binary encoding achieves the minimal average length, since in this case

$$Pr(X=n)=\frac{1}{2^m}$$

$$l(n)=\log_2(\frac{1}{Pr(X=n)})=\log_2(2^m)=m$$

which is exactly the number of bits required by the binary code to encode a number in the range of [0, 1, , 2^m-1].

Table 8: *Encoded Data and C.E.*

		Encoded data (bits)				Compression Effectiveness (%)			
Benchmark	X (%)	Kay	FDR	Golomb	Proposed	Kay	FDR	Golomb	Proposed
S9234	0.01	159	96	144	150	90.14	94.04	91.07	90.70
	0.02	221	178	187	197	87.11	89.62	89.10	88.51
	0.05	373	368	310	323	81.46	81.71	84.59	83.95
	0.10	652	654	521	547	74.30	74.22	79.46	78.44
	0.25	1369	1366	1127	1150	65.52	65.59	71.61	71.10
	0.50	2618	2532	2186	2186	59.83	61.15	66.46	66.46
	0.95	4402	4146	3861	3784	56.71	59.23	62.03	62.79
S13207	0.01	455	384	393	398	88.19	90.03	89.80	89.67
	0.02	699	686	588	585	85.08	85.36	87.45	87.52
	0.05	1436	1492	1155	1170	79.80	79.02	83.76	83.54
	0.10	2633	1660	2153	2110	76.90	76.66	81.11	81.48
	0.25	5886	5850	5018	4779	74.79	74.95	78.51	79.53
	0.50	10389	1023	9361	8738	74.02	74.42	76.60	78.15
	0.95	17017	0	1659	1477	74.22	73.87	74.87	77.62
			1725 2	0	5				
S15850	0.01	202	162	172	176	87.44	89.93	89.30	89.05
	0.02	314	304	258	264	84.09	84.60	86.93	86.63
	0.05	589	608	476	487	79.59	78.93	83.51	83.13
	0.10	1077	1090	880	879	76.66	76.38	80.93	80.09
	0.25	2415	2380	2027	1983	74.12	74.49	78.27	78.75
	0.50	4313	4214	3871	3672	73.96	74.56	76.63	77.83
	0.95	7088	7236	6943	6172	74.55	74.01	75.07	77.99
S35932	0.01	124	108	106	110	87.42	89.05	89.25	88.84
	0.02	189	184	157	162	84.12	84.54	86.81	86.39
	0.05	367	376	300	301	79.57	79.06	83.30	83.24
	0.10	663	690	541	534	76.46	75.50	80.79	81.04
	0.25	1487	1462	1232	1231	73.03	73.48	77.65	77.67
	0.50	2625	2602	2385	2234	73.81	74.04	76.20	77.71
	0.95	4213	4234	4057	3743	72.83	72.69	73.84	75.86
S38417	0.01	690	580	599	605	88.2	90.08	89.76	89.65
	0.02	1101	1042	908	915	84.81	85.63	87.48	87.38
	0.05	2217	2248	1795	1792	80.03	79.75	83.83	83.85
	0.10	4006	4068	3263	3204	77.01	76.65	81.27	81.61
	0.25	9050	8822	3263	7416	74.49	75.13	78.56	79.09
	0.50	16206	15878	14595	13501	74.47	74.99	77.01	78.73
	0.95	26257	26890	25865	22706	74.92	74.32	75.30	78.31
S38584	0.01	630	522	500	551	88.35	90.03	90.75	89.81
	0.02	964	928	802	815	85.10	85.66	87.61	87.41
	0.05	1979	2004	1586	1632	79.66	79.40	83.70	83.23
	0.10	3555	3592	2874	2898	76.27	76.03	80.82	80.66
	0.25	8025	7894	6721	6589	73.57	74.00	77.86	78.30
	0.50	14310	1400	1281	1223	73.20	73.76	75.99	77.09
	0.95	23247	8	7	0	73.49	73.03	74.07	76.52
			2365 6	2274 3	2059 1				

Unary code

When the Probability Mass Function (PMF) of X is given by

$$Pr(X=n)=\frac{1}{2^{n+1}},\quad n\geq 0$$

we have

$$l(n)=\log_2(\frac{1}{Pr(X=n)})=\log_2(2^{n+1})=n+1.$$

This is exactly the number of bits required by unary code to encode a non-negative integer (n number of 1s followed by a 0). Thus, unary code is optimal for encoding integer random variable with the above PMF.

Golomb code

When the probability mass function of X is given by

$$Pr(X = n) = p^n(1 - p)$$

we have

$$l(n) = \log_2(\frac{1}{Pr(X = n)}) = n\log_2(\frac{1}{p}) + \log_2(\frac{1}{1-p}).$$

On the other hand, the number of bits required by Golomb code to encode an integer n is given by

$$\lfloor \frac{n}{M} \rfloor + 1 + \log_2 M.$$

Equating this to *l(n)*, we see that for the code to be optimal, the coefficient of n must be approximately equal. That is,

$$\frac{1}{M} \approx \log_2 \frac{1}{p}$$

or

$$M \approx -\frac{1}{\log_2 p}.$$

In the literature [20], it is known that Golomb code is optimal when

$$M = \lceil -\frac{1}{\log_2 p} \rceil.$$

The diagram in Fig.13 shows the relationship between M and p. From the diagram, we see that when $p \leq^1 \backslash_2$, unary code is optimal. When $p > \frac{1}{2}$, Golomb code becomes optimal. Since n in our case is the number of '0' bits before a '1' bit occurs, we can interpret p as the probability that a bit is '0' in the original binary sequence (assumed to be independent for this interpretation). Thus, the above results show that when '0' occurs less frequently than '1' ($p \leq \frac{1}{2}$), unary code is optimal. When '1' occurs less frequently than '0' ($p > \frac{1}{2}$), Golomb code becomes optimal. The more infrequently that bit '1' occurs, the larger M we should pick.

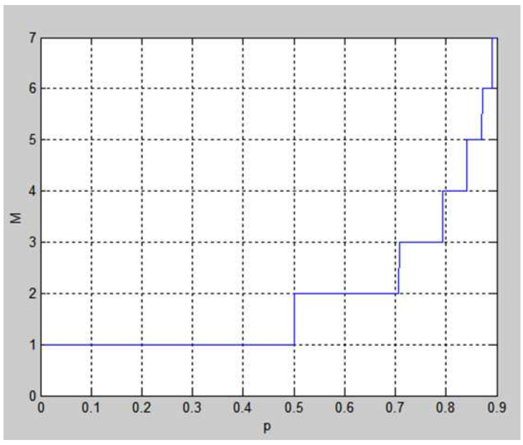


Fig.13: *The relationship between M and p*

N	*l(n)*	
0	2	covering n in [0, 1]
1		
2	4	covering n in [2, 2+4-1]
3		
4		
5		
6	6	covering n in [2+4, 2+4+8-1]
7		
8		
9		
10		
11		
12		
13		
14	8	covering n in [2+4+8, 2+4+8+16-1]
...		
29		
n in [30, 61]	10	cover n in [2 + 4 + 8 + 16, 2 + 4 + 8 + 16 + 32 -1]

$$2^j - 3 < N \leq 2^{j+1} - 3$$
$$2^j < n + 3 \leq 2^{j+1}$$
$$j < \log_2(n+3) \leq j + 1$$
$$j - 1 < \log_2(n+3) - 1 \leq j$$

Thus, we have

$$j = \lceil \log_2(n+3) - 1 \rceil = \lceil \log_2(n+3) \rceil - 1$$
$$l(n) = 2j = 2(\lceil \log_2(n+3) \rceil - 1)$$

Using the sufficient condition for the optimal code

$$l(n) = \lg(\frac{1}{Pr(X = n)})$$

we have

FDR code

Let

n = run length

$l(n)$ = codeword length for integer n.

To relate n in the first column of the table below with *l(n)* in the second column, we note

$$\log_2 \frac{1}{P_n} = l(n) = 2([\log_2(n+3)] - 1)$$

where we use p_n to denote $\Pr(X = n)$

$$\frac{1}{p_n} = 2^{2(\lceil \log_2(n+3) \rceil - 1)}$$

$$p_n = \frac{1}{2^{2\left\lceil \log_2(\frac{n+3}{2}) \right\rceil}},$$

which is the PMF that makes FDR optimal.

To better understand the above equation, we note $\log_2 \frac{n+3}{2} \leq \lceil \log_2 \frac{n+3}{2} \rceil < \log_2 \frac{n+3}{2} + 1 = \log_2(n+3)$

$$\frac{n+3}{2} \leq 2^{\left\lceil \log_2 \frac{n+3}{2} \right\rceil} < n+3$$

$$\frac{(n+3)^2}{4} \leq 2^{2\left\lceil \log_2 \frac{n+3}{2} \right\rceil} \leq (n+2)^2$$

$$\frac{1}{(n+2)^2} \leq p_n = \frac{1}{2^{2\left\lceil \log_2 \frac{n+3}{2} \right\rceil}} \leq \frac{4}{(n+3)^2}.$$

Thus, as n increases, p_n decreases quadratically with $n \left(i.e..p_n \propto \frac{1}{n^2} \right)$. Thus, if large n occurs more frequently, FDR is better than Golomb code, which is optimal for data PMF that decreases exponentially with n (making large n occurring less frequently).

References

[1] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, A. Ferko, B. Keller, D. Scott, B. Koenemann, T. Onodera, "Extending OP-MISR beyond 10X Scan Test Efficiency," *Design and Test Computers*, pp. 65-73, Sep-Oct 2002.

[2] S. Mitra, S. S. Lumetta, M. Mitzenmacher, N. Patel, "X-Tolerant Test Response Compaction," *Design and Test Computers*, pp. 566-574, Nov-Dec 2005.

[3] P. Wohl, J. A. Waicukauski, T. W. Williams, "Design of Compactors for Signature-analyzers in Built-in Self-test," *ITC*, pp. 54-63, 2001.

[4] J. H. Patel, S. S. Lumetta, S. M. Reddy, "Application of Saluja-Karpovsky Compactors to Test Responses with Many Unknowns," *Proc. of VTS*, pp. 107-112, 2003.

[5] S. Mitra, K. S. Kim, "X-Compact: An Efficient Response Compaction Technique," *Trans. on Computer-Aided Design*, vol. 23, no. 3, pp. 421-432. 2004.

[6] B. Pouya, N. A. Touba, "Synthesis of Zero-aliasing Element-tree Space Compactors," *Proc. of VTS*, pp. 70-77, 1998.

[7] C. Wang, S. M. Reddy, I. Pomeranz, J. Rajski, J. Tyszer, "On Compacting Test Response Data Containing Unknown Values," *Proc. of ICCAD*, pp.855-862, 2003.

[8] J. Rajski, J. Tyszer, C. Wang, S. M. Reddy, "Convolutional Compaction of Test Responses," *ITC*, pp. 745-754, 2003.

[9] E. H. Volkerink, S. Mitra, "Response Compaction with any Number of Unknowns Using a new LFSR Architecture," *DAC*, pp. 177-122, 2005.

[10] V. Chickermane, B. Foutz, B. Keller, "Channel Masking Synthesis for Efficient On-Chip Test Compression," *ITC*, pp. 452-461, 2004.

[11] Y. Tang, H-J Wunderlich, H. Vranken, F. Hapke, M. Wittke, P. Engelke, I. Polian, B. Becker, "X-Making During Logic BIST and Its Impact on Defect Coverage," *ITC*, 2004.

[12] M. Naruse, I. Pomeranz, S. M. Reddy, S. Kundu, "On-Chip Compression of Output Responses with Unknown Values Using LFSR Reseeding," *ITC*, pp.1060-1068, 2003.

[13] I. Pomeranz, S. Kundu, S. M. Reddy, "On Output Response Compression in the Presence of Unknown Output Values," *Proc. of DAC*, pp. 255-258, 2002.

[14] H. Tang, C. Wang, J. Rajski, S. M. Reddy, J. TYszer, and I. Pomeranz, "On Efficient X-handling Using a Selective Compaction Scheme to Achieve High Test Response Compaction Ratios," *Proc. of VLSI Design*, 2005.

[15] I. Pomeranz, S. Kundu, S. M. Reddy, "Making of Unknown Output Values during Output Response Compression by Using Comparison Units," *Trans. on Computers*, Vol. 53, No. 1, pp. 83-89, 2004.

[16] G. Mrugalski, J. Rajski, J. Tyszer, "Test Response Compactor with Programmable Selector," *DAC*, pp. 1089-1094, 2006.

[17] S. Lin, D. J. Costello, Jr., *Error Control Coding*, Pearson Education, Inc., 2004.

[18] T. Reungpeerakul, X. Qian, and S. Mourad, "BCH-based Compactors with Data Compression," *Proc. of ECTI-CON*, pp. 685-688, 2008.

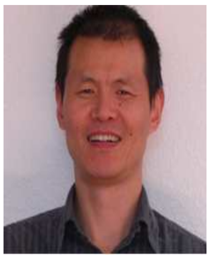
[19] D. Kay, and S. Mourad, "Compression Technique for Interactive BIST Application," *Proc. of VLSI Test Symposium*, pp. 9-14, 2001.

[20] A. Chandra, and K. Chakrabarty, "Test Data Compression and Test Resource Partitioning for System-on-a-Chip Using Frequency-Directed Run-Length (FDR) Codes," *Trans. on Computers*, vol. 52, no. 8, pp. 1076-1088, 2003.

[21] A. Chandra, and K. Chakrabarty, "System-on-a-Chip Test-Data Compression and Decompression Architectures Based on Golomb Codes," *Trans. on Computer-Aided Design*, vol.20, no. 3, pp. 355-368, 2001.

[22] K. Sayood , *Introduction to Data Compression*, Academic Press, 2nd ed., 2000.

Taweesak Reungpeerakul is currently working as an Assistant Professor at the Department of Computer Engineering, Prince of Songkla University. He received the Bachelor of Engineering degree from Prince of Songkla University and the Master of Engineering degree from Chulalongkorn University, Thailand. He is receiving the Ph.D. degree from Santa Clara University in June 2009. His research interest is in the field of Design for Testability and Built-In Self- Test.



Xiaoshu Qian was awarded Ph.D. in Electrical Engineering, M.S. in Computer Science and M.S. in Mathematics, all from University of Rhode Island in US and B.S. in Physics from Zhejiang University in China. Xiaoshu Qian has been with Intel Corporation for the past 11 years, working mainly on multimedia ICs, cable modem ICs, WiFi algorithms and WiMax ICs. His role ranges from logical designer, algorithm designer, system engineer and engineering manager. Prior to joining Intel, he worked at a Silicon Valley start-up and Texas Instrument. He has published numerous technical papers and is the main author of five U.S. patents with several pending. In addition, he has taught a dozen graduate courses in the area of digital communication and digital signal processing at Santa Clara University and several courses in the University of Rhode Island, International Technology University and Northwestern Technology University.

Samiha Mourad is the William and Janice Terry professor in Electrical Engineering Department at SCU. In addition she served as the Chair of the Department and the Associate Dean of the School. Immediately before joining SCU, she worked at Stanford University as the Associate Director of the Center of Reliable Computers. Prior to this she was the chair of the Division of Science and Mathematics at Fordham University, New York City. Professor Mourad is an IEEE fellow and a member of ACM and Sigma Xi.