An Efficient Signature Computation Method

KEWAL K. SALUJA
University of Wisconsin, Madison
CHIN-FOO SEE
Hewlett-Packard Singapore

This article presents a signature generation algorithm for LFSR-based compactors used in fault simulation of built-in self-test digital circuits. The algorithm uses small- to medium-size lookup tables to generate signatures for internal as well as external exclusive-OR LFSRs of any length. The basic concept can be extended to general linear compactors. The authors also present algorithms that convert signatures from one form of LFSR to the other.

HARDWARE OVERHEAD, impact on system performance, and fault coverage are key variables a designer must consider in deciding to use BIST (built-in self-test) in a design. Once the designer completes a design with BIST capability, its hardware overhead and system performance become known deterministically. But the designer must rely on probabilistic results about the aliasing aspect of fault coverage. (A vast body of literature deals with the aliasing problem.) For high fault coverage, that approach is unsatisfactory. Indeed, the only way to obtain a deterministic answer about fault coverage is to simulate the circuit and the BIST environment for each fault in the circuit. While simulating BIST circuits, designers often use pseudorandom pattern generators and compactors such as linear-feedback shift registers (LFSRs). The simulation is costly because it entails the simulation of the generator, the faulty circuit, and the compactor for large numbers of test vectors, without fault dropping.

Advances in fault simulation techniques have reduced simulation time considerably, but research has done little to improve the efficiency of compaction, or signature computation, algorithms.[1-3] Recently, Lambidonis, Ivanov, and Agarwal[1] proposed a table-driven compaction algorithm applicable to single-input linear compactors, which is similar to the algorithm presented in this article.

An earlier version of this article was presented at the Fifth International Conference on VLSI Design, Bangalore, India, January 1992.

We propose a signature computation algorithm based on methods proposed earlier.[2,4,5] We restrict our discussion and application of the algorithm to single-input LFSRs, although the basic concepts described here can be applied to arbitrary linear compactors with multiple inputs.

Preliminaries and notation

Figure 1 shows a popular realization of an LFSR. It is an internal-exclusive-OR (internal-XOR) implementation and a true serial divisor. In the figure each g_i is either a 1 (closed connection) or a 0 (open connection). We express the LFSR as a polynomial, $g(x)$, which characterizes the connections. Similarly, we express the input stream as a polynomial, $I(x)$, and the contents of the LFSR as a polynomial, $R_I(x)$. We perform division by taking $I(x)$ as dividend and $g(x)$ as divisor, with $R_I(x)$ the remainder. After we fully process $I(x)$, the final value of $R_I(x)$, called the internal signature, is the check bits for a cyclic redundancy check (CRC) code.[6]

Figure 2 shows an alternative LFSR realization, an external-XOR implementation. We use $I(x)$ to express the input stream, $g(x)$ to express the LFSR with external-XOR, and $R_E(x)$ to express the

0740-7475/92/1200-0022$03.00 © 1992 IEEE

contents of the LFSR. We call the final $R_E(x)$ the external signature.

We now explain how to construct lookup tables and use them to compute the internal signature. Consider the situation shown in Figure 3a. Let us assume that the contents of the LFSR are S^*, and an 8-bit input to the LFSR is i^*. We wish to compute the contents of the LFSR after i^* has been shifted into the LFSR. Using the principle of superposition (linearity), we partition the problem into two parts, as shown in Figure 3b. Clearly, for an internal-XOR LFSR of length 8 or more bits, with an internal state of 0 and an input i^*, the state of the LFSR after eight shifts will be i^*, as shown in Figure 3c. On the other hand, the state S^{**} shown in Figure 3c is the state of the LFSR after we start from state S^* and shift eight zeros into it. We must determine this state.

We can obtain S^{**} from S^*, using precomputed lookup tables. To explain the process, we assume that the LFSR is 16 bits long and we use the pictorial representation developed in Figure 3. Figure 4 shows the complete process. In Figure 4a, $S_1{}^*$ and $S_2{}^*$ are the two 8-bit components of S^*, and we assume the input to the LFSR is 0. Again using the principle of superposition in linear systems, we partition the problem into two parts (Figure 4b). Figure 4c shows the two-part state of the internal-XOR LFSR after eight shifts. It is now evident that we need to compute S, the state of the LFSR after eight shifts, provided the initial content of the LFSR is $S_1{}^*$.

Knowing the polynomial or the feedback structure of the LFSR, we can construct a table off line for every possible value of $S_1{}^*$. If $S_1{}^*$ is 8 bits, the table will have $2^8 = 256$ entries, and each entry for our example will be 16 bits wide. Finally, by XORing S, $S_2{}^*$, and i^*, we compute the state S^{**}. Thus, the process of finding S^{**} from S^* and i^* involves one table lookup and two XOR operations. Further, if we look carefully at Figure 3c and Figure 4c, we see that $S_2{}^*$ and i^* need not be XORed. Instead, they can be concatenated, a step we can integrate into the

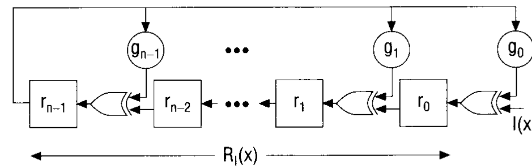


Figure 1. *Internal-XOR LFSR.*

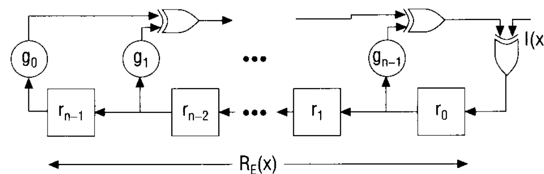


Figure 2. *External-XOR LFSR.*

formation of the table.[5]

Before proceeding to the next section, we point out that for a given $I(x)$ and $g(x)$, the values of $R_I(x)$ and $R_E(x)$ are different, but there is a one-to-one mapping between internal and external signatures.[6] Later in the article we present a technique to convert an internal signature to an external signature and vice versa. The conversion technique takes an amount of time proportional to the LFSR's length and independent of the input data stream's length. Moreover, our scheme works for any polynomial (not just primitive polynomials) characterizing the LFSR.

Signature computation

Our signature computation algorithm is based on the explanation given in the preceding section. (Sarwate presents an alternative explanation.[5]) Figure 5 (page 24)presents a pictorial representation of the algorithm; each $A(i)$ is a byte-long input to the LFSR, and the input stream consists of m bytes.

The algorithm consists of two parts. The first part constructs the lookup tables by simulating the LFSR or by computing for all possible values of $S_1{}^*$ as explained earlier. Next, the algorithm processes the input stream and computes the check bits, using the lookup

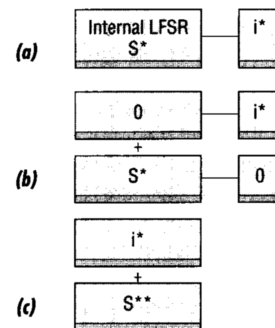


Figure 3. *Processing an input stream by an internal-XOR LFSR: internal state* ***(a)****; partitions* ***(b)****; states after processing* ***(c)****.*

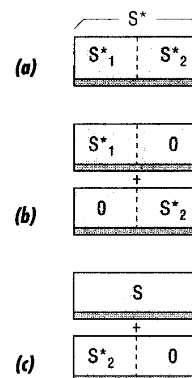


Figure 4. *Determining the state of an internal-XOR LFSR: initial state* ***(a)****; partition state* ***(b)****; state after eight shifts* ***(c)****.*

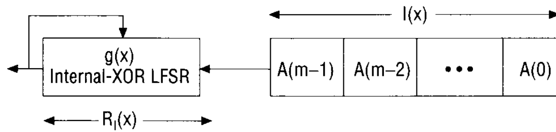


Figure 5. General representation of signature computation algorithm.

tables. It is this part of the algorithm that is of interest to us in this section. The algorithm variations for computing 8-, 16-, 24-, and 32-bit signatures are shown in the box on page 25. They are in pseudo C language. In each case, the size of the lookup tables is 2^8 because each $A(i)$ is 1 byte (8 bits) long.

Algorithm 1 first initializes the 8-bit register C0 to zero. Then, it takes m cycles to process m bytes of input. The byte $A(m-1)$, the most significant byte, is processed first. T, a temporary variable, is used as an index into the lookup table f0. When all the input bytes have been processed, the remainder (the check bits or the signature) is left in C0.

In the algorithms for computing 16-, 24-, and 32-bit signatures, we use the variables C0, C1, C2, and C3 to form the signature. Each of these Ci variables is 8 bits long. Similarly, f0, f1, ..., represent lookup tables. (We give a method of extending these algorithms to nonmultiples of 8-bit signatures in the earlier version of this article.[2])

Algorithm 2 is Sarwate's algorithm for computation of 16-bit signatures.[5] The algorithm produces the signature for an input stream by appending a stream of zeros. Furthermore, it modifies the input stream while generating the signature. To produce the correct signature for a given input stream, we must modify the algorithm or follow it with some postprocessing. Algorithm 3 is a modified version that produces the correct signature and does not modify the input stream.

Algorithms 4 and 5, for 24- and 32-bit LFSRs, are similar to the 8-bit and 16-bit algorithms, except that they require more lookup tables and more operations per cycle.

Signature conversion

If internal-XOR and external-XOR LFSRs are characterized by the same polynomial, $g(x)$, a one-to-one relationship exists between $R_I(x)$ and $R_E(x)$.[6] That means that conversion from one form of signature to the other is always possible and the result is always unique. Conversion may be required for any of several reasons:

- If we know either the internal or the external signature, we can obtain the other form of signature by conversion, instead of reprocessing all the input stream to derive the other form. One of the most time-consuming steps in reprocessing the input stream is reading the input file, which we avoid in the conversion process.
- We may wish to initiate signature generation as soon as we choose the LFSR polynomial but before we decide on the actual implementation of the LFSR (internal or external).
- Sometimes we need to know both signatures before choosing one form over the other—for example, when the storage or hardware realization of one signature is superior to the other in the BIST environment. In such a case we can simulate either the internal or the external signature and then perform the conversion. This reduces the processing time to approximately half the time of simulating both forms of LFSRs.
- Simulating one form may be faster than simulating the other form. In that case we can compute the signature with the faster form, and convert it to the other form.

In the following explanation of the conversion processes, we have not included formal proofs as they are unduly long, but they are easy to deduce by logical reasoning.

Internal to external. If we define $I(x)$, $g(x)$, and $R_I(x)$ as in Figure 1, where $I(x)$ is the input polynomial, $g(x)$ is the division polynomial, and $R_I(x)$ is the internal-XOR LFSR signature, then

$$I(x) = Q(x) \cdot g(x) + R_I(x)$$

where $Q(x)$ is the quotient polynomial. Similarly, if we process the same input through the external-XOR LFSR with the same $I(x)$ and $g(x)$, and $R_E(x)$ is the external-XOR LFSR signature, there also exists a relation

$$I(x) = Q(x) \cdot g(x) + f(R_E(x))$$

where $R_I(x) = f(R_E(X))$, and the function f is one-to-one. Note that the quotient polynomial is the same for both forms, but the signatures ($R_I(x)$ and $R_E(x)$) may be different.

To convert $R_I(x)$ to $R_E(x)$, we first ini-

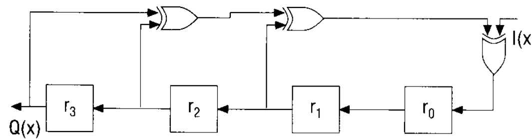


Figure 6. External-XOR LFSR with $g(x) = x^4 + x^2 + x + 1$.

tialize the external-XOR LFSR to an all-zero state. Then we input $R_I(x)$ into the external-XOR LFSR. After n cycles (where n is the length of the signature), the external signature, $R_E(x)$, remains in the LFSR.

We demonstrate this process through an example. Let us consider $g(x) = x^4 + x^2 + x + 1$ (a nonprimitive polynomial). Figure 6 gives the external-XOR LFSR for this polynomial. For $R_I(x) = 1101$, we can verify that $R_E(x)$ is 1111. This is a unique value of $R_E(x)$ for $R_I(x) = 1101$. Table 1 shows all possible 4-bit signatures $R_I(x)$ and $R_E(x)$ derived through this procedure and the mapping between them for the polynomial $g(x)$ in this example.

External to internal. In the preceding subsection, we demonstrated that with $R_I(x)$ as input to the external-XOR LFSR, we can obtain $R_E(x)$ in the LFSR. To derive the internal signature, $R_I(x)$, from the external signature, $R_E(x)$, we load the external-XOR LFSR with $R_E(x)$, and compute "backward" till the contents of the register become zero. At this stage the value of $R_I(x)$ appears as $I(x)$.

Table 1. Signatures of $g(x) = x^4 + x^2 + x + 1$.

Internal signature $r_3\, r_2\, r_1\, r_0$	External signature $r_3\, r_2\, r_1\, r_0$
0000	0000
0001	0001
0010	0010
0011	0011
0100	0101
0101	0100
0110	0111
0111	0110
1000	1011
1001	1010
1010	1001
1011	1000
1100	1110
1101	1111
1110	1100
1111	1101

Signature computation algorithms

Algorithm 1

```
                                  /* 8-bit LFSR */
C0 = 0;                           /* initialize LFSR to zeros */
/* process m bytes of input, one byte per iteration */
for i := m - 1 to 0 do
{
   T = C0;                        /* T is the lookup index */
   C0 = f0[T] XOR A[i];           /* f0 is the only table */
}
```

Algorithm 2

```
                                  /* 16-bit LFSR */
C0 = C1 = 0;                      /* initialize LFSR to zeros */
/* process m bytes of input, one byte per iteration */
for i := m - 1 to 0 do
{
   T = C1 XOR A[i];               /* T is the lookup index */
   C1 = C0 XOR f1[T];             /* process bit 15-8 of LFSR */
   C0 = f0[T];                    /* process bit 7-0 of LFSR */
}
```

Algorithm 3

```
                                  /* 16-bit LFSR */
/* Two tables, f0 and f1, needed */
C0 = C1 = 0;                      /* initialize LFSR to zeros */
for i := m - 1 to 0 do
{
   T = C1;                        /* T is the index of tables */
   C1 = C0 XOR f1[T];             /* process upper 8 bits of LFSR */
   C0 = f0[T] XOR A[i];           /* process lower 8 bits of LFSR */
}
```

Algorithm 4

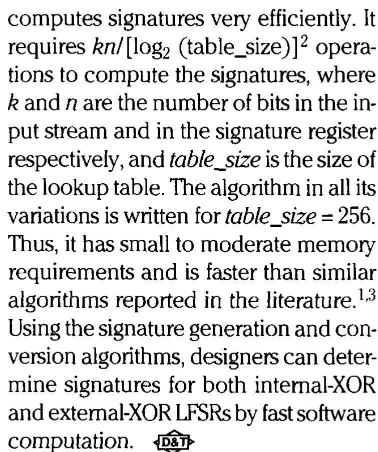
```
                                  /* 24-bit LFSR */
/* Three tables, f0, f1, and f2, needed */
C0 = C1 = C2 = 0;                 /* initialize LFSR to zeros */
for i := m - 1 to 0 do
{
   T = C2;                        /* T is the index of tables */
   C2 = C1 XOR f2[T];             /* process bit 23-16 of LFSR */
   C1 = C0 XOR f1[T];             /* process bit 15-8 of LFSR */
   C0 = f0[T] XOR A[i];           /* process bit 7-0 of LFSR */
}
```

Algorithm 5

```
                                  /* 32-bit LFSR */
/* Four tables needed: f0, f1, f2, and f3 */
C0 = C1 = C2 = C3 = 0;            /* initialize LFSR to zeros */
for i := m - 1 to 0 do
{
   T = C3;                        /* T is the index of tables */
   C3 = C2 XOR f3[T];             /* process bit 31-24 of LFSR */
   C2 = C1 XOR f2[T];             /* process bit 23-16 of LFSR */
   C1 = C0 XOR f1[T];             /* process bit 15-8 of LFSR */
   C0 = f0[T] XOR A[i];           /* process bit 7-0 of LFSR */
}
```

Note that during the backward computation, $Q(x)$ must be set to zero. Also, the first output bit at $I(x)$ is the least significant bit of $R_I(x)$.

Again, we demonstrate the process through an example. Let $R_E(x) = 1011$ and $g(x) = x^4 + x^2 + x + 1$. Using the steps for computing $R_I(x)$ shown in Table 2, we find that $R_I(x) = 1000$. This value is the same as shown in Table 1 for $R_E(x) = 1011$.



THE ALGORITHM WE HAVE PRESENTED computes signatures very efficiently. It requires $kn/[\log_2 (\text{table_size})]^2$ operations to compute the signatures, where k and n are the number of bits in the input stream and in the signature register respectively, and *table_size* is the size of the lookup table. The algorithm in all its variations is written for *table_size* = 256. Thus, it has small to moderate memory requirements and is faster than similar algorithms reported in the literature.[1,3] Using the signature generation and conversion algorithms, designers can determine signatures for both internal-XOR and external-XOR LFSRs by fast software computation.

Table 2. Example computation of internal signature from external signature.

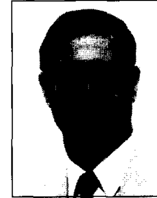
Q(x)	r_3	r_2	r_1	r_0	I(x)	
0000	1	0	1	1	—	/* set 1011 in register */
000	0	1	0	1	0	/* process for four clock cycles */
00	0	0	1	0	00	
0	0	0	0	1	000	
—	0	0	0	0	1000	

Acknowledgment

The work described in this article was supported in part by National Science Foundation grant MIP-9111886.

References

1. D. Lambidonis, A. Ivanov, and V.K. Agarwal, "Fast Signature Computation for Linear Compactors," *Proc. Int'l Test Conf.*, IEEE Computer Society Press, Los Alamitos, Calif., 1991, pp. 808-817.
2. C.F. See and K.K. Saluja, "An Efficient Method for Computation of Signatures," *Proc. Fifth Int'l Conf. VLSI Design*, IEEE CS Press, 1992, pp. 245-250.
3. S.B. Tan et al., "A Fast Signature Simulation Tool for Built-In Self-Testing Circuits," *Proc. 24th Design Automation Conf.*, IEEE CS Press, 1987, pp. 17-25.
4. G. Griffiths and G.C. Stones, "The Tea-Leaf Reader Algorithm: An Efficient Implementation of CRC-16 and CRC-32," *Comm. ACM*, Vol. 30, No. 7, July 1987, pp. 617-620.
5. D.V. Sarwate, "Computation of Cyclic Redundancy Checks via Table Lookup," *Comm. ACM*, Vol. 31, No. 8, Aug. 1988, pp. 1008-1013.
6. W.W. Peterson and E.J. Weldon, Jr., *Error-Correcting Codes*, 2nd ed., MIT Press, Cambridge, Mass., 1972.



Kewal K. Saluja is a professor in the Department of Electrical and Computer Engineering at the University of Wisconsin-Madison, where he teaches logic design, computer architecture, microprocessor-based systems, and VLSI design and testing. Previously, he worked at the University of Newcastle, Australia. He has also held visiting and consulting positions at the University of Southern California, the University of Iowa, and Hiroshima University. His research interests include design for testability, fault-tolerant computing, VLSI design, and computer architecture. He is an associate editor of the *Journal of Electronic Testing: Theory and Applications*. Saluja received the BE from the University of Roorkee, India, and the MS and the PhD in electrical and computer engineering from the University of Iowa.

Chin-Foo See is a production engineer at Hewlett-Packard Singapore. He is responsible for the HP 75LX Palmtop Computer production, quality assurance, testing, line automation, and cost reduction programs. He received the BSEE from the University of Southwestern Louisiana, Lafayette, and the MSEE in electrical and computer engineering from the University of Wisconsin-Madison, where he participated in the research described in this article. His areas of interest include VLSI design, testing, and computer architecture.

Address correspondence about this article to Kewal K. Saluja, Dept. of Electrical and Computer Engineering, University of Wisconsin, Madison, WI 53706; e-mail: saluja@engr.wisc.edu.