Counting Two-State Transition-Tour Sequences

Nirmal R. Saxena[1] & *Edward J. McCluskey*
Center for Reliable Computing, ERL 460
Department of Electrical Engineering, Stanford University,
Stanford, CA 94305

Abstract

This paper develops a closed-form formula, *f*(*k*), to count the number of transition-tour sequences of length *k* for bistable machines. It is shown that the function *f*(*k*) is related to Fibonacci numbers. Some applications of the results in this paper are in the areas of: testable sequential machine designs, random testing of register data paths, and qualification tests for random pattern generators.

Index Terms: Transition-Tours, Sequential Machine Testing, Fibonacci Numbers, Checking Experiments, and Testable Synthesis.

1.0 Introduction

A *transition-tour sequence* is a binary sequence that includes all four transitions between adjacent binary bits. For example, 01100 is a transition-tour sequence because it has all four transitions $0 \rightarrow 1$, $1 \rightarrow 0$, $1 \rightarrow 1$ and $0 \rightarrow 0$ between adjacent bits. The definition of transition-tour is consistent with the general definition of transition tours defined for finite state machines in [1-5]. We want to find a function, *f*(*k*), that counts the number of distinct length *k* transition-tour sequences.

An application of transition-tour sequences in testing registers (for data path designs) and memory elements for finite state machines has been shown in [6]. Transition-tour sequences relate to checking experiments [7]; and checking experiments can be used for an exhaustive test of sequential machines. Registers and memory elements in data path designs [6] generally model the behavior of *D* flip-flops or *D* latches [8]. In [6] transition-tour sequences were called checking sequences. The change in terminology was motivated by two reasons. First, the use of transition-tour sequence is consistent with earlier definitions in references [1-5]. Second, the term checking sequence is likely to be considered as a synonym for checking experiments. Although it is argued in [6] that checking sequences for D-latches (without considering clock signals) are checking experiments; the consideration of clock signal as an explicit input to the D-latch (as shown in [9]) results in checking experiments that are different from checking sequences (as defined in [6]). However, it is important to point out that transition-tour sequence is an important part of the checking experiment because it includes all state-transitions.

1. Also Engineering Manager, MIPS Technology, Silicon Graphics Inc., Mountain View, CA.

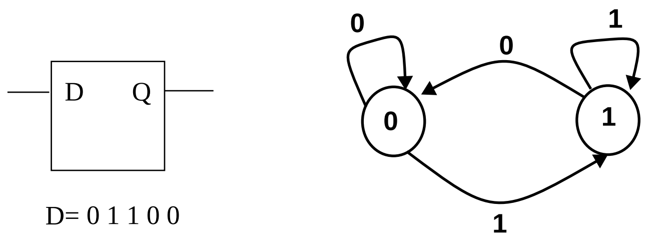

Figure 1. D Flip-Flop/Latch Transition-Tour Sequence

The sequence 01100 is a shortest transition-tour sequence that takes the *D* latch or flip-flop (Figure 1) from an initial state (0) through all four transitions. The other three shortest transition-tour sequences are: 10011, 00110, 11001.

The organization of the rest of this paper is as follows. Sections 2 and 3 develop a closed-form formula for *f*(*k*). Some applications of *f*(*k*) are discussed in Section 4. Section 5 concludes the paper.

2.0 Development of *f*(*k*) using Auxiliary Function *g*(*k*)

Direct derivation of *f*(*k*) appears to be difficult. The authors feel that an auxiliary function is necessary to obtain a closed-form formula for *f*(*k*). It is useful to define an auxiliary function, *g*(*k*); *g*(*k*) is the number of length *k* binary sequences that have at least one $0 \rightarrow 0$ transition. In other words *g*(*k*) counts all length *k* sequences that have a “00” subsequence. This section derives a closed-form formula for *g*(*k*). In Section 3, a simple closed-form relationship is derived between *f*(*k*) and *g*(*k*).

Lemma 1: Function *g*(*k*), for $k \geq 2$, satisfies the following recurrence relation:

$$g(k) = 2^{k-2} + g(k-1) + g(k-2)$$

Proof: Length *k* ($k \geq 2$) sequences can be partitioned into the following three disjoint classes:

1. Binary sequences that begin with subsequence “00” followed by length *k*-2 sequence. All of these sequences have subsequence “00” and therefore form a subset of sequences enumerated by *g*(*k*). There are 2^{k-2} sequences in this class.
2. Binary sequences that begin with bit 1. In these sequences, *g*(*k*-1) sequences will have a subsequence “00”.
3. Binary sequences that begin with subsequence “01”. In these sequences, *g*(*k*-2) sequences will have a subsequence “00”.

Classes 1, 2, and 3 completely specify *g*(*k*). Therefore we have the recurrence

$$g(k) = 2^{k-2} + g(k-1) + g(k-2) \quad (1)$$

Lemma 2: Function $g(k)$ is related to Fibonacci numbers, $F(k)$, by the following relation: $g(k) = 2^k - F(k+2)$

Proof: Consider the following generating function of $g(k)$: $G(X) = \sum_{k \geq 2} g(k) X^k$. Using equation (1) we can re-express the generating function as

$$G(X) = \sum_{k \geq 2} 2^{k-2} X^k + \sum_{k \geq 2} g(k-1) X^k + \sum_{k \geq 2} g(k-2) X^k$$

Manipulating the indices in the summations of $g(k\text{-}1)X^k$ and $g(k\text{-}2)X^k$ and using the closed-form solution for the geometric power series summation with terms $2^{k-2}X^k$, we have

$$G(X) = \frac{X^2}{(1-2X)} + XG(X) + X^2G(X) + g(1)X + g(1)X^3 + g(0)X^2 \quad (2)$$

By definition $g(2)=1$, because there is only one length-two sequence (00) that has a “00” subsequence. Also, $g(1)=0$ because it is impossible for a length-one sequence to have “00” subsequence. Using equation (1) for $k=2$, we can show that $g(2)=1+g(1)+g(0)$ and therefore $g(0)=0$. Equation (2) can now be simplified to get a closed-form solution for $G(X)$ as

$$G(X) = \frac{X^2}{(1-2X)(1-X-X^2)} = \frac{1}{(1-2X)} - \frac{X+1}{1-X-X^2} \quad (3)$$

The first term, $1/(1\text{-}2X)$, in equation (3) has a power series expansion having coefficients as powers of 2; and the second term, $(X+1)/(1\text{-}X\text{-}X^2)$, in equation (3) includes the generating function for the Fibonacci numbers [10]. From this generating function, $g(k)$ is the coefficient of X^k in the power series expansion of equation (3). Therefore we have,

$$g(k) = 2^k - F(k) - F(k+1) = 2^k - F(k+2) \quad (4)$$

TABLE 1. Enumeration of Some Values of $g(k)$

k	2	3	4	5	6	7	8	9
$F(k+2)$	3	5	8	13	21	34	55	89
$g(k)$	1	3	8	19	43	94	201	423

Example 1: For k=5, there are 19 (Table 1) length 5 sequences that have a “00” subsequence. These sequences are: 00000, 00001, 00010, 00011, 00100, 00101, 00110, 00111, 01000, 01001, 01100, 10000, 10001, 10010, 10011, 10100, 11000, 11001, and 11100.

Next, we use $g(k)$ to derive a closed-form formula for $f(k)$.

3.0 Closed-Form Formula for *f*(*k*)

The following theorem derives a formula for $f(k)$.

Theorem 1: The number of transition-tour sequences, $f(k)$, of length k (≥3) is given by the relation:

$$f(k) = 2^k - 2k + 8 - 2F(k+2)$$

Proof: The transition properties of the length k binary sequence can be characterized by four Boolean variables where each Boolean variable specifies the truth value of a particular transition. The following table lists the notation for these four Boolean variables:

TABLE 2. Boolean Variable Characterization of Sequences

Variable	Sequence Property	Negated Variable	Sequence Property
<00>	Sequence has “00” subsequence	<00>’	Sequence has no “00” subsequence
<01>	Sequence has “01” subsequence	<01>’	Sequence has no “01” subsequence
<10>	Sequence has “10” subsequence	<10>’	Sequence has no “10” subsequence
<11>	Sequence has “11” subsequence	<11>’	Sequence has no “11” subsequence

For a given sequence, the truth value of these four Boolean variables can be independently tested. Figure 2. shows the four variable *Karnaugh map* [7,8,11] partition of the sequences. There are sixteen cells in the Karnaugh map (for compactness in notation we will call it *K*-map) shown in Figure 2. These cells are labeled 0 through 15 and each cell corresponds to one of the 16 possible states of the Boolean variables defined in Table 2. For example, the cell labeled 4 corresponds to the set of binary sequences that have $1 \rightarrow 0$, $0 \rightarrow 1$, and $1 \rightarrow 1$ transitions but not the $0 \rightarrow 0$ transition. The defining Boolean term for this cell 4 from the *K*-map is <11> & <10> & <01> & <00>’, where “&” is the Boolean “AND” operation. The expressions inside each cell represent the number of sequences that satisfy the transition properties defined by the cell.

We will use this *K*-map based characterization to derive a closed-form formula for *f*(*k*). Our interest is to count the sequences specified by cell 5 in the *K*-map. This counting is done indirectly by counting sequences specified by other cells. For example, in Lemma 1 we have already counted the number of sequences *g*(*k*) that correspond to the union of cells 1, 2, 5, 6, 9, 10, 13, 14 (Region defined by the Boolean variable <00>).

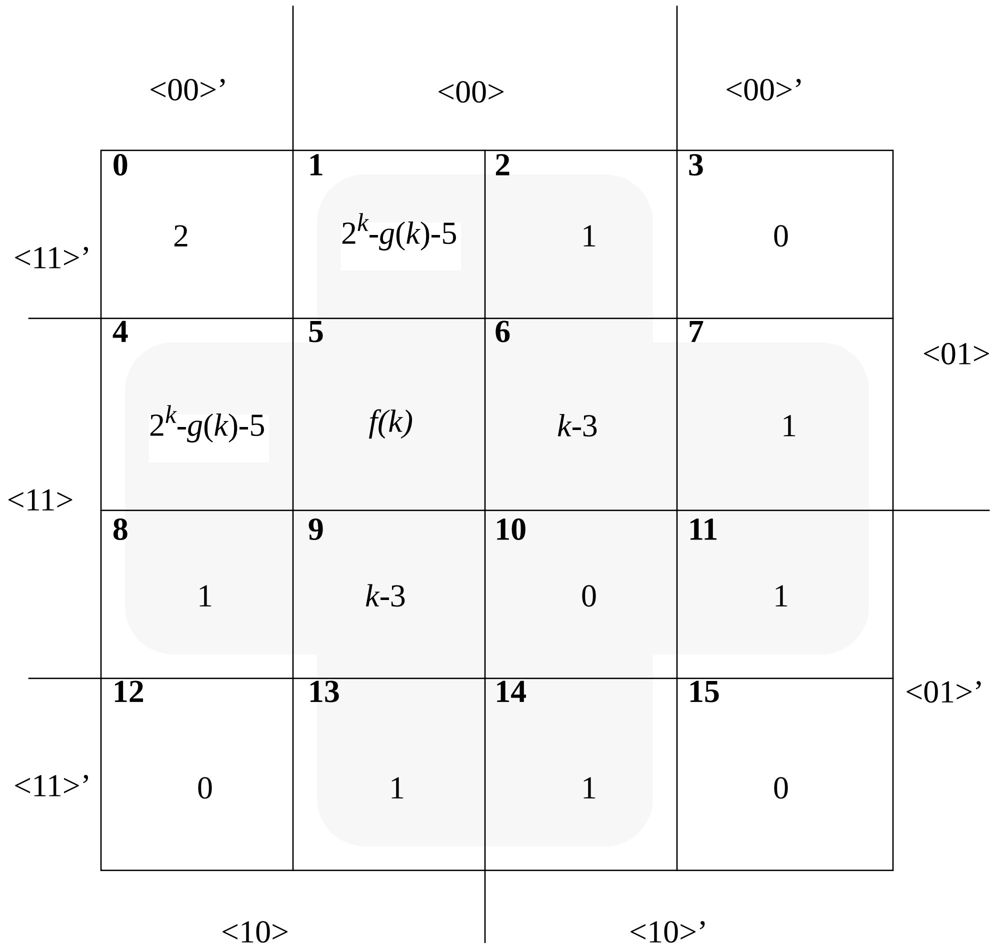

Figure 2. Karnaugh-Map Partition of Binary Sequences

Now we will begin to count the number of sequences in some of the cells in the K-map.

Cell 15: This cell corresponds all those sequences of length greater than or equal to 3 that do not have any of the sequence transitions (<11>' & <10>' & <01>' & <00>'). Any sequence of length greater than or equal to 3 will have at least one transition. Therefore, there are no sequences in this category.

Cell 10: This cell characterizes all those sequences that have $1 \rightarrow 1$ and $0 \rightarrow 0$ transitions but no $1 \rightarrow 0$ and no $0 \rightarrow 1$ transitions (i.e., the term <11> & <10>' & <01>' & <00>). This again is impossible because if a sequence has both $1 \rightarrow 1$ and $0 \rightarrow 0$ transitions then there must be at least one $1 \rightarrow 0$ or $0 \rightarrow 1$ transition. Therefore, the number of sequences in this cell is zero.

Cell 14: These are the sequences that have only $0 \rightarrow 0$ transition and no other transition. The only sequence in this category would be the all-zero sequence. Therefore, the number of sequences in this cell is one.

Cell 11: These are the sequences that have only $1 \rightarrow 1$ transition and no other transition. The only sequence in this category would be the all-one sequence. Therefore, the number of sequences in this cell is one.

Cell 13: These are the sequences that have only $0 \rightarrow 0$ and $1 \rightarrow 0$ transitions and no other transition. The only sequence in this category is a sequence that begins with 1 and is followed by (k-1) 0's.

Cell 7: These are the sequences that have only $1 \rightarrow 1$ and $0 \rightarrow 1$ transitions and no other transition. The only sequence in this category is a sequence that begins with 0 and is followed by (k-1) 1's.

Cell 2: These are the sequences that have only $0 \rightarrow 0$ and $0 \rightarrow 1$ transitions and no other transition. The only sequence in this category is a sequence that ends with 1 and is preceded by (k-1) 0's.

Cell 8: These are the sequences that have only $1 \rightarrow 1$ and $1 \rightarrow 0$ transitions and no other transition. The only sequence in this category is a sequence that ends with 0 and is preceded by (k-1) 1's.

Cell 0: These are the sequences that have only $1 \rightarrow 0$ and $0 \rightarrow 1$ transitions and no other transition. For sequence lengths greater than or equal to 3 there are only two sequences of alternating 0's and 1's. The sequences are 101010... and 010101... . Therefore this cell has two sequences.

Cell 12: These are the sequences that have only $1 \rightarrow 0$ and no other transition. This is again an impossible case for sequences of length greater than or equal to 3 (however, there is one sequence of length 2). Therefore there are no sequences in this category for $k \geq 3$.

Cell 3: These are the sequences that have only $0 \rightarrow 1$ and no other transition. This is again an impossible case for sequences of length greater than or equal to 3 (however, there is one sequence of length 2). Therefore there are no sequences in this category for $k \geq 3$.

Cell 6: There are the sequences that have $1 \rightarrow 1$, $0 \rightarrow 0$, and $0 \rightarrow 1$ transition but no $1 \rightarrow 0$ transition. These sequences must have only two runs; that is, a run of zeros followed by a run of ones. Further, the run-length of zeros must be at least two and the run-length of ones must be at least two. The sequences in this category are those sequences that begin with at least two consecutive 0's and are followed by at least two consecutive 1's. There are k-3 such sequences. These sequences can be indexed as follows: consider i 0's ($i \geq 2$) followed by k-i 1's ((k-i) $\geq$ 2). Therefore, i ranges from 2 to k-2 and hence k-3 sequences.

Cell 9: There are the sequences that have $1 \rightarrow 1$, $0 \rightarrow 0$, and $1 \rightarrow 0$ transition but not $0 \rightarrow 1$ transition. These sequences must have only two runs; that is, a run of ones followed by a run of zeros. Further, the run-length of ones must be at least two and the run-length of zeros must be at least two. The sequences in this category are those sequences that begin with at least two consecutive 1's and are followed by at least two consecutive 0's. There are k-3 such sequences. These sequences can be indexed as follows: consider i 1's ($i \geq 2$) followed by k-i 0's ((k-i) $\geq$ 2). Therefore, i ranges from 2 to k-2 and hence k-3 sequences.

Next, we enumerate the number of sequences in the remaining cells 1, 4, and 5.

Cell 1: The union of cells 0,1, 2, 3, 12, 13, 14, and 15 represents all those sequences that do not have $1 \rightarrow 1$ transition. Clearly, the number of sequences in this union is all possible sequences (2^k) minus the number of sequences that have $1 \rightarrow 1$ transition (There are $g(k)$ such sequences; this follows from the symmetry between 0's and 1's). The only unknown in this union is the number of sequences in cell 1. Therefore, the number of sequences in cell 1 = 2^k-$g(k)$-(the number of sequences in the union of cells 0,2,3,12,13,14, and 15) = 2^k-$g(k)$-5.

Cell 4: By symmetry between 0's and 1's, the number of sequences in cell 4 is also equal to 2^k-$g(k)$-5.

Cell 5: The number of sequences in cell 5 can be considered by the number of sequences in the union of cells 1, 2, 5, 6, 9, 10, 13, and 14. This union is precisely the enumeration of all the sequences that have $0 \rightarrow 0$ transition. This is $g(k)$. The only unknown in this union is the number of sequences, $f(k)$, in cell 5. Therefore, the number of sequences $f(k)$ is equal to $g(k)$ minus the number of sequences in the union of cells 1, 2, 6, 9, 10, 13, and 14. Using the known values, we have $f(k) = 2g(k)-2k+8-2^k$.

Substituting the expression for $g(k)$ from Lemma 1, we have

$$f(k) = 2^k - 2k + 8 - 2F(k+2) \qquad \text{Q.E.D.}$$

The following table lists values of $f(k)$ for some k.

Example 2: For k=6, there are 18 (Table 3) transition-tour sequences. These are: 011001, 011000, 110010, 110011, 110001, 001100, 001101, 001110, 100111, 100110, 101100, 010011, 111001, 000110, 110100, 001011, 011100, 100011

TABLE 3. Enumeration of $f(k)$ for some values of k

k	3	4	5	6	7	8	9
$F(k+2)$	5	8	13	21	34	55	89
$f(k)$	0	0	4	18	54	138	324

4.0 Applications

This section presents some applications of the closed-form results in this paper. These applications are in the areas of:

- Random pattern testability measures for register faults in datapath elements
- State assignment and synthesis of finite-state machines for testability of memory elements. Here testability applies to the detection of internal faults in the memory elements (mostly implemented using latches).
- Qualification tests to determine if multiple single-bit binary sequences are statistically independent or uncorrelated.

4.1 Datapath Register Random Pattern Testability

If L random test patterns are applied to an n-bit datapath register then the probability that a transition-tour sequence is applied to all of the n-bits in the register is $(f(L))^n/2^{nL}$. Let us denote this as (n,L) *transition-tour probability.* This transition-tour probability relates to the testability of faults (internal faults in the latches that implement register elements) in the n-bit register for the applied pattern length L. Figure 3 shows the plot of transition-tour probability as a function of L, for n=32, 64, and 128. As the register width increases the random pattern test length also has to increase in order to achieve a given transition-tour probability. For example, in order to get greater than 97% transition-tour probability, a 32-bit register would require a random pattern test length of at least 37; however, a 128-bit register would require at least a test length of 44.

In actual practice, the patterns applied to registers are not truly random. This either results in higher or lower than expected transition-tour probability. The subsequent sections show empirical evidence that linear feedback based patterns and some restricted cases of linear congruential patterns have higher transition-tour probability than that expected from a pure random pattern.

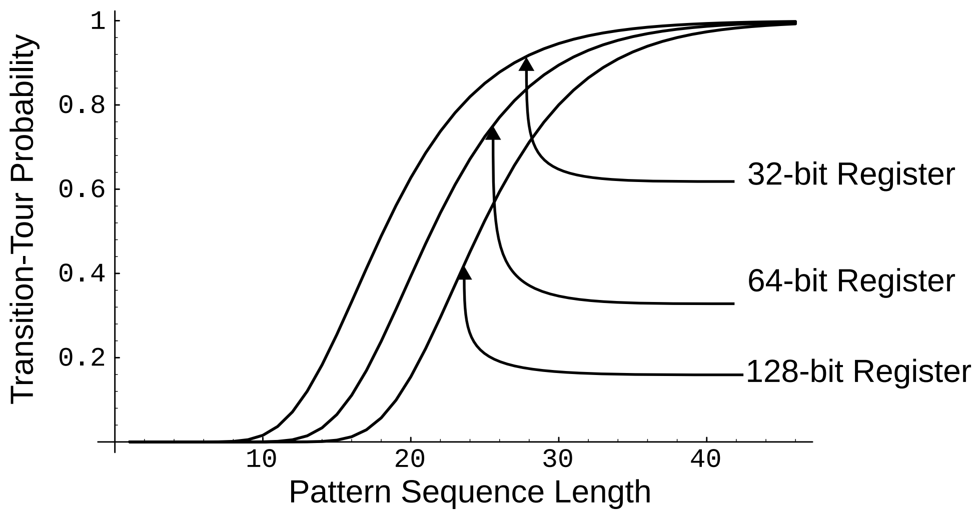

Figure 3. Probability that all Register Elements are Covered by a Transition-Tour Sequence

4.2 Testable Control Unit Synthesis for Memory Element Faults

The control units in digital systems are generally implemented using finite state machines. The inputs to the combinational and the memory elements in the control units are usually constrained by architectural and implementation features. For example, the inputs to the memory elements in the finite-state machines are determined by the state machine encoding and the state transitions. In contrast, the inputs to memory elements (*D* latches or flip-flops) in datapath registers are generally less constrained and are more controllable.

Typically, the number of states in finite state machines implementing simple control units is small. Assuming that state assignment is done in an arbitrary manner (i.e., assignment done for other optimizations like area and delay but not for transition-tour sequence coverage) then the transition-tour probability could be very low. For instance, assume a finite state machine with 16 states (encoded by 4-bits, n=4) and with a state transition sequence involving 5 (L=5) states then the (4,5) transition-tour sequence coverage probability (assuming random state assignment) would be (4/32)^4 = 0.000244. Computations based on function $f(k)$ suggest that the state machine encoding done not taking into account the transition-tour coverage could result in low testability of memory elements.

Figure 4 from [6] illustrates an encoding for a finite state machine implementing the control function of a load/store unit. Fault simulation of memory element faults [6] using normal operational sequences for this unit fail to detect some memory element stuck-at faults. The modified finite state machine (Figure 5), with an encoding that guarantees transition-tour coverage for all of the memory elements, detected all of the memory element stuck-at faults using normal load/store operational sequences.

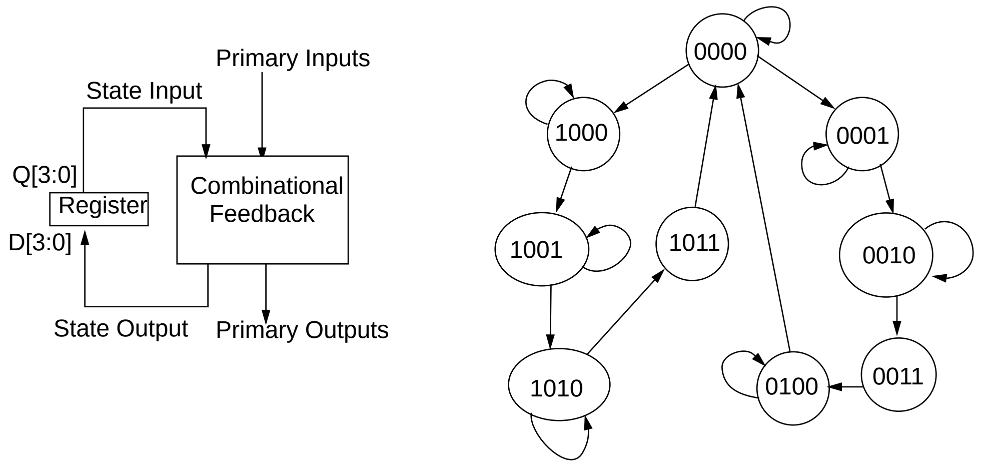

Figure 4. State Transition Diagram for a Load/Store

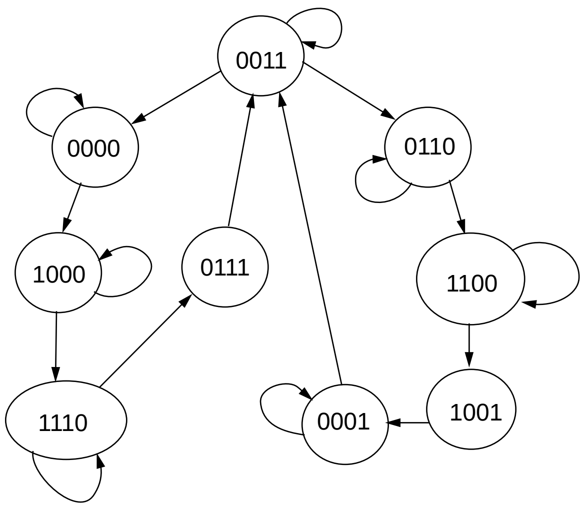

Figure 5. Modified State Encoding with Embedded Transition-

The modified encoding method suggested in the foregoing paragraph illustrates the application of the results in this paper. The actual synthesis algorithms for transition-tour state assignments are not the subject of this paper and will be presented in a separate report. More work needs to be done in this synthesis area; both, in terms of obtaining transition-tour state assignments for arbitrary state machines and quantifying the impact of these assignments on silicon implementation area and circuit delays.

4.3 Qualification Tests for Random Bit-Pattern Generators

The function $f(k)$ can also be used to test whether multi-bit pattern sequence generators are statistically independent. This can be done as follows:

- The theoretical (n,L) transition-tour probability is calculated using the function $f(k)$.
- Using the multi-bit pattern sequence generators, the (n,L) transition-tour probability can be experimentally estimated for various values of n and L. The experimental estimation is done as follows: An experimental trial consists of generating a length L n-bit pattern. Each of the n individual (decomposing n-bits) length L single-bit sequences are examined to see if they include all transition tours. If all of the n individual sequences include all transition tours individually then this event is counted. The trail is repeated with different multi-bit patterns for a given n and L. The fraction of events counted estimates the (n,L) transition-tour probability. This experiment is repeated for various values of n and L.
- By comparing experimental estimates with the theoretical estimates from $f(k)$, we can show if the multi-bit pattern sequence departs from a statistically independent multi-bit pattern sequence.

Figure 6 compares the transition-tour probability of a single bit *linear feedback shift register* (LFSR) sequence [12] (for a modular LFSR [8] with polynomial $x^{32}+x^4+x^3+x^2+1$) with the analytical probability derived using $f(k)$ for a pure random sequence. The close match between the experimental and analytical probability values suggests that single bit sequences from LFSRs are fairly random. This is not surprising because single-bit shift register sequences demonstrate randomness properties [12].

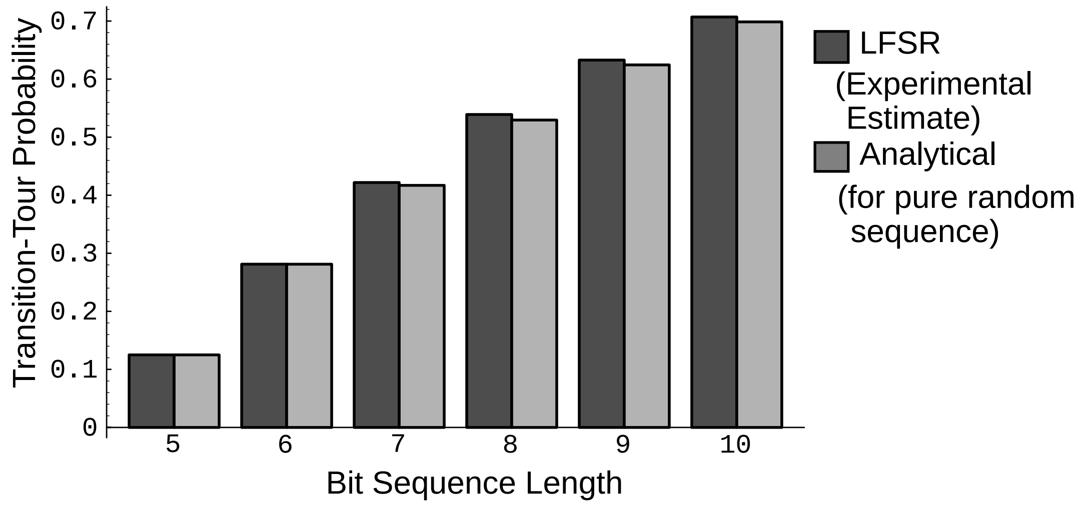

Figure 6. Experimental Estimate of Transition-Tour Sequence Probability for an

This is generally is not true for other pseudorandom generators like *linear congruential* (LC) generators [13]. For example, if the bit sequence is derived from the least significant bit of the word generated by the LC generator then the sequence will be alternating 0's and 1's. This is hardly random.

Figure 7 clearly shows the departure of LFSR and LC generators from pure random pattern generators. The plots in Figure 7 are for 16-bit pattern sequences. The LFSR (with polynomial $x^{32}+x^4+x^3+x^2+1$) and the LC generator were both 32-bits in precision. The LC function generator was based on function, *rand*(), used in UNIX operating systems. The 16-bits from the LFSR were obtained from the least significant 16 bits and the 16-bits from the LC generator were derived from the most significant bits (avoiding less random least significant bits) from the 32-bit precision words. The plots of are for (16,*L*) transition-tour probability.

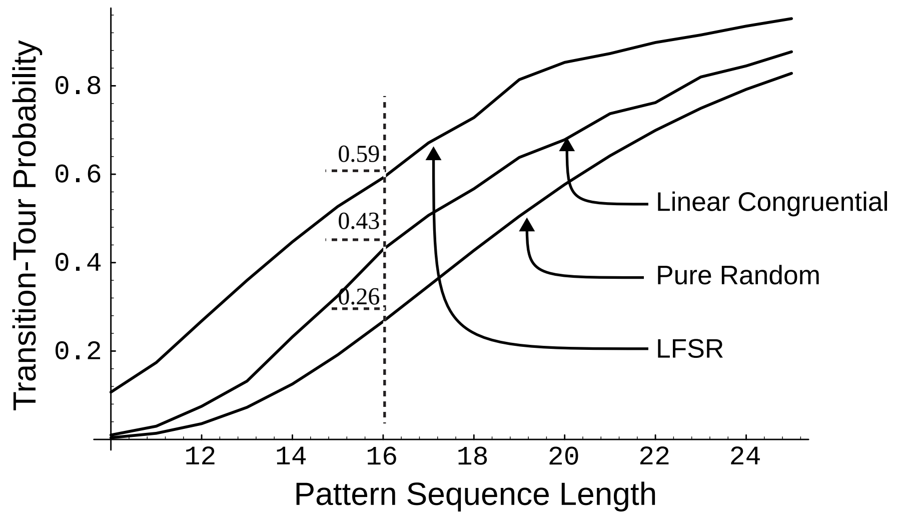

Figure 7. Experimental Estimates of Transition-Tour Probability for All bits in a Pattern Sequence

For L=16 and n=16, the transition-tour probability for pure random 16-bit pattern is around 0.26. The corresponding estimates of 0.43 and 0.59 for LC and LFSR patterns respectively were based on a sample size of 1000. The departure from pure random sequence value, 0.26, cannot be a chance variation because the standard deviation due to estimation error (based on sample size of 1000) cannot be more than ±0.014 from the expected value 0.26. Fortunately, this departure is on the higher side. This is not surprising for the shift register patterns because any decimated LFSR sequence has runs of 00, 01, 10, and 11 [12]. However, for LC sequences the explanation for higher transition-tour probability is not obvious.

5.0 Conclusions

This paper has developed a closed-form formula for the number of transition-tour sequences. For testing memory elements, normally one would require simple tests of reading and writing 0's and 1's respectively. However, internal faults in memory elements such as latches [9] do require more than mere reading and writing of 1's and 0's for fault detection. Transition-tour sequences exhaustively test for all memory element state transitions and detect internal faults not detected by simple write/read 0/1 tests. In this respect, applications in the areas of testability measures for datapath registers, testable state machine synthesis, and randomness qualification tests for multi-bit pattern sequence generators have been shown. Results on transition-tour probability in this paper suggest that LFSRs are a more desirable choice than other random pattern generators for built-in test of embedded RAMs. The K-map based technique to count $f(k)$ can also count other functions. For example, if we are interested in finding the number of sequences of length k that have $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions then considering the cells defined by <10> & <01> we can show that there are 2^k-2 k such sequences.

Acknowledgment:

The authors would like to thank Siyad Ma for bringing the transition-tour terminology and reference [5] to our attention.

References

[1] Ural, H., “Formal Methods for Test Sequence Generation,” Computer Communications, vol. 15, no. 5, p. 311-325, June 1992.

[2] Aho, A.V., A.T. Dahbura, & D. Lee, “An Optimization Technique for Protocol Conformance Test Generation Based UIO sequences and Rural Chinese Postman Tours,” IEEE Transactions on Communications, vol. 39, no. 11, p. 1604-1615, Nov. 1991.

[3] Boyd, S., “The Synchronization Problem in Protocol testing and its Complexity,” Information Processing Letters, vol. 48, no. 3, p. 131-136, Nov. 1991.

[4] Shen, Y.N., F. Lombardi, & D. Sciuto, “Evaluation and Improvement of Fault Coverage for Verification and Validation of Protocols,” Proceedings of the Second IEEE Symposium on Parallel and Distributed Computing, p. 200-207, 1990.

[5] Naito S., & M. Tsunoyama, “Fault Detection for Sequential Machine by Transition-Tours,” Proceedings FTCS-11, p. 238-243, June 1981.

[6] Saxena, N.R, R. Tangirala & A. Srivastava, “Algorithmic Synthesis of High Level Tests for Data Path Designs,” *Proceedings of 23rd International Symposium on Fault-Tolerant Computing*, Toulouse, France, p. 360-369, June 1993.

[7] Kohavi, Zvi, *Switching and Finite Automata Theory*, Second Edition, Tata McGraw-Hill Publishing Co., New Delhi, 1978.

[8] McCluskey, E.J., *Logic Design Principles with Emphasis on Testable Semicustom Circuits*, Prentice-Hall, Englewood Cliffs, NJ, 1986.

[9] Makar S.R, & E.J. McCluskey, “Checking Experiments to Test Latches,” Proceedings 13th IEEE VLSI Test Symposium, pp. 196-201, Apr. 1995.

[10] Graham, R.L., D.E. Knuth, & O. Patashnik, *Concrete Mathematics A Foundation for Computer Science*, Addison-Wesley Publishing Company, 1989.

[11] Karnaugh, M., “The Map Method for Synthesis of Combinational Logic Circuits,” *Trans. AIEE.* pt. I, vol. 72, no. 9, p. 593-599, 1953.

[12] Golomb, S. W., *Shift Register Sequences*, Aegean Park Press, Laguna Hills, Calif. 1982.

[13] Knuth, D.E., *Seminumerical Algorithms*, Addison-Wesley Publishing Company, Inc., Reading, Mass., 1969.