Bit-reverse Gray code method:

$$(30)_2 = (1\ 1\ 1\ 1\ 0) \qquad \text{binary representation}$$

$$(30)_2 = (0\ 1\ 1\ 1\ 1) \qquad \text{bit-reverse}$$

$$(d_{30})_2 = (0\ 1\ 0\ 1\ 0) \qquad \text{bit-reverse inverse Gray code}$$

$$d_{30} = [0\ 1\ 0\ 1\ 0][16\ 8\ 4\ 2\ 1]^T = 10.$$

Conclusion

We have developed a simple method for determining the sequency ordering of any row in a given Hadamard matrix directly from the binary represenation of the order of that row with simple modification in the signs of the "ones." The whole sequency vector can be also determined by using a single recursive formula, in contrast to other methods which require more than one recursive formula. Both proposed methods are found to be much simpler than other known methods.

References

[1] R. Lackey and D. Meltzer, "A simplified definition of the Walsh functions," *IEEE Trans. Comput.*, vol. C-20, pp. 211-213, Feb. 1971.

[2] D. Cheng and J. Liu, "An algorithm for sequency ordering of Hadamard function," *IEEE Trans. Comput.*, vol. C-26, pp. 308-309, Mar. 1977.

[3] S. Srihari and M. Ohanesian, "An efficient algorithm for determining Hadamard sequency vector," *IEEE Trans. Comput.*, vol. C-28, pp. 243-244, Mar. 1979.

Definition and Design of Strongly Language Disjoint Checkers

INGRID JANSCH AND BERNARD COURTOIS

***Abstract*—This paper defines strongly language disjoint (SLD) checkers. SLD checkers are to sequential systems what strongly code disjoint checkers are to combinational systems. SLD checkers are the largest class of checkers with which a functional system may achieve the TSC goal. Self-checking sequential systems are first addressed, then formal definitions of SLD checkers are given. The next point is the design of SLD checkers based on regular combinational self-checking components.**

***Index Terms*—Checkers, concurrent error detection, self-checking design, sequentially self-checking systems, strongly language disjoint checkers, VLSI design.**

I. Introduction

Basic definitions and properties of self-checking systems have been given by Anderson in [1]. Self-testing (ST), fault secure (FS), totally self-checking (TSC), and code disjoint (CD) circuits have been defined. Next, strongly fault secure (SFS) and strongly code disjoint (SCD) circuits have been defined, respectively, by Smith and Metze in [2] and by Nicolaidis and Courtois in [3]. SFS and SCD circuits are the largest classes of circuits such that the totally self-checking goal (i.e., the first erroneous output is a noncodeword) may be achieved, when the system is a *combinational* one.

Sequential systems are addressed in this paper. The definitions of sequentially self-checking (SeSC) systems given by Viaud and David in [4] are used as a starting point and checkers of SeSC systems are defined: the strongly language disjoint (SLD) checkers. SLD checkers are the largest class of checkers such that, when associated with an SeSC circuit, the totally self-checking goal may be achieved [5]. Basically, SLD (sequential) checkers are needed when the output string of the SeSC circuit is defined by a sequential property. They keep the language disjoint (LD) property even if faults are present inside, similarly to (combinational) SCD checkers keeping the code disjoint property and to SFS circuits keeping the FS property even if faults are present inside (combinational systems). Two formal definitions will be given. One of them defines SLD checkers such that "simultaneous" faults ("simultaneous" will be formally defined) cannot be admitted in the functional system and in the checker. The other one will define VSLD checkers such that "simultaneous" faults may be admitted in the functional system and in the checker. Lastly, the design of SLD checkers will be addressed. Proposals are given based on regular structures.

II. Sequentially Self-Checking Circuits

The approach considered here is of a sequentially self-checking system where the inputs are sequences defined by a given input language and the outputs are sequences defined by a given output language. The checker similarly receives a given input language (the output language of the functional sequential system) and produces outputs belonging to an output language. These outputs are an indication of an error when they do not belong to the output language. The basic scheme used is shown in Fig. 1. The sequential machine is sequentially self-checking as defined by Viaud and David in [4].

As in [4], $M = (Q, X, Z, \delta, w, q_0)$ is assumed to be a sequential machine. Q is the set of internal states. X and Z are the input and output alphabets, respectively. δ and w are the state transition function and the output function, respectively, and q_0 is the initial state. We use the Mealy machine definition for our considerations, but those may be easily applied to the equivalent Moore machine. Under the fault f, the machine $M = (Q, X, Z, \delta, w, q_0)$ becomes $M^f = (Q^f, X, Z^f, \delta^f, w^f, q_0)$. Under normal operation, $\delta(i, q)$ is the state reached from $q \in Q$ when input sequence i is applied, and $w(i, q)$ is the obtained output sequence. If a fault f occurs in the state q of M, it is supposed that M^f reaches immediately the state q^f. Hence, $\delta^f(\lambda, q) = q^f$, $q \in Q$, $q^f \in Q^f$, and $w^f(i, q)$ is the obtained output sequence. Let $i = i1 \cdot i2$, where "$\cdot$" denotes the concatenation. The input sequences $i1$ and $i2$ are said to be a prefix and a suffix of i, respectively. $P(i)$ and $S(i)$ will denote the set of prefixes and the set of suffixes of i, respectively. Iq is the set of input sequences which may be applied from that state q in normal operation: $Iq = \{i2 : i1 \cdot i2\} \in I_M$ and $\delta(i1, q_0) = q$. $Iq^\infty \in Iq$ defines the input sequences of nonbounded length. The shortest prefix of $i2$, such that an output sequence not included in the output language of M is obtained in the presence of fault f, is denoted by i_{2m}. Then, we have $w(i1, q_0) \cdot w^f(i_{2m}, q) \notin O_M$.

The concepts of input and output languages and of sequentially self-checking (SeSC) circuits are those defined in [4]. Now we give some basic concepts concerning sequential machines, which had not been given in [4], before defining sequential checkers.

Definition D1: A circuit M is redundant with respect to a fault f, a state q, and an input sequence $i2 \in Iq$, iff $\forall i2 \in P(i2)$, $w^f(i2^+, q) = w(i2^+, q)$.

Remark R1: In a circuit redundant with respect to a fault f, a state q, and an input sequence $i2 \in Iq$, i_{2m} does not exist. Remind that i_{2m} is the shortest prefix such that $w(i1, q_0) \cdot w^f(i_{2m}, q) \notin O_M$.

For sequential circuits, the *TSC goal* is the property which ensures

Manuscript received July 31, 1985; revised March 11, 1986 and January 26, 1987. This paper was presented in part at the 15th IEEE International Symposium on Fault-Tolerant Computing, Ann Arbor, MI, 1985. This work was supported in part by CNPq and FINEP, Brazil.

I. Jansch is with the Pós-Graduação em Ciência da Computação, Universidade Federal do Rio Grande do Sul, 90001 Porto Alegre, Brazil.

B. Courtois is with the Laboratoire TIM3/IMAG, 38031 Grenoble Cedex, France.

IEEE Log Number 8716232.

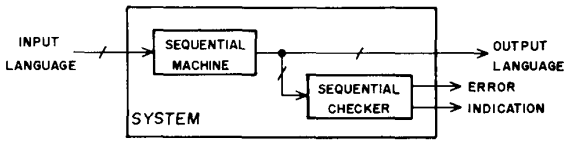

Fig. 1. Sequentially self-checking systems.

that the first erroneous output from the machine M results in an output sequence not included in the output language O_M. The hypothesis that may be used in order to achieve the TSC goal is expressed as follows:

Hypothesis H1: Between the occurrence of any two faults belonging to F, a sufficient time elapses such that, for any $q \in Q$, for any $i1$ such that $\delta(i1, q_0) = q$ and such that $i1 \cdot i2 \in I_M$, all i_{2m} with respect to each fault f or sequence of faults are applied, whenever they exist.

It may be shown [6][1] that any sequentially self-checking (SeSC) circuit, under Hypothesis H1, achieves the TSC goal.

Remark R2—(Remark on Hypothesis H1): The aim of Hypothesis H1 is to ensure the detection of the first fault or of a new one with respect to which the sequential circuit is not redundant before the occurrence of another fault. Another way to formulate Hypothesis H1 is to enforce the appearance of detection transition sequences (DTS), as defined by [7]. In this case, Hypothesis H1 may be formulated as: "Between the occurrence of any two faults belonging to F, a sufficient time elapses such that for any $q \in Q$, for any $i2 \in Iq$, for any $i1$ such that $i1 \cdot i2 \in I_M$, and $\delta(i1, q_0) = q$, at least one DTS with respect to each fault or sequence of faults is applied. Note that, when i_{2m} exists, $i2 \in Iq$, a transition sequence $q \cdot i_{2m}$ may be a DTS.

III. Strongly Language Disjoint Checkers

Strongly language disjoint (SLD) checkers defined in this paper will be different from the checkers used in the literature for the design of self-checking sequential machines. Strongly language disjoint checkers will check a *sequential* property, and consequently they will be defined as *sequential checkers* when *combinational* checkers have been used in the past for the design of self-checking sequential systems.

Several authors have studied the design of fail-safe or self-checking sequential systems (e.g., [8]–[12]) but in all these studies, the checker was a combinational circuit, checking in general the state's code. Others (like [13], [14], etc.) considered algebraic approaches or the use of reverse or parallel machines. In all these studies, the checker is combinational. On the contrary, SLD checkers are *sequential* checkers, to be used associated with a sequential machine, according to the general structure already represented in Fig. 1.

The first sequential checkers have been defined by Viaud and David [4] based on the following situation. Consider a functional part having a 2-bit output such that, in normal operation, this output is constituted by an alternation of even and odd number of zeros. This output language may be formally described as $O_M = ((00 + 11)(01 + 10)^*$. A fault-free sequence may be 00, 01, 00, 10, 11, 10, $\cdots$. For this defined output of a functional circuit, a checker is a sequential circuit. The language disjoint property had been introduced by [4], as a generalization of the code disjoint property.

Definition D2: A machine M is language disjoint if $\forall i \in I_M$, $w(i, q_0) \in O_M$ and $\forall i \notin I_M$, $w(i, q_0) \notin O_M$.

Definition D3: A circuit is a sequentially self-checking checker if it is both sequentially self-checking and language disjoint.

The problem with these definitions is that nothing is said concerning the language disjoint property when faults are present inside the checker. A similar problem was presented by combinational checkers, when they were defined to be code disjoint, but not strongly code disjoint.

[1] A complete version of this correspondence [6] can be obtained with one of the authors.

In the next definitions, we name, respectively, Ic and Oc the input and output languages of a checker, and $w_c(i', q')$ the output sequence obtained from $q' \in Q'$ on the checker. (The symbols are distinguished from those used for the functional block by a mark ($'$) or by an index c.) Notice that, although defining an output language to be obtained in the outputs of the checker, this output language may be degenerated in a string of output codewords (for example, a double-rail code). But in the following, a sequential property at the outputs is considered in order to keep general definitions.

Definition D4: A sequential checker C is redundant with respect to a fault f, and with respect to an input language Ic and with respect to an output language Oc if $\forall i' \in Ic$, $\forall i1' \in P(i')$, $w_c(i1', q_0') \cdot w_c^f(i2', q') \in Oc$, where $i' = i1' \cdot i2'$ and $q' = \delta'(i1', q_0')$, and $\forall i' \notin Ic$, $\forall i1' \in P(i')$, $w_c(i1', q_0') \cdot w_c^f(i2', q') \notin Oc$.

Definition D5: A sequential checker is strongly redundant with respect to the fault sequence $\langle f1, f2, \cdots, fn\rangle$ and with respect to an input language Ic and with respect to an output language Oc if it is redundant with respect to the n fault subsequences $\langle f1\rangle$, $\langle f1, f2\rangle$, $\langle f1, f2, f3\rangle$, $\cdots$, $\langle f1, f2, \cdots, fn\rangle$ and with respect to the input language Ic, and with respect to the output language Oc.

Now it is possible to give the definition of strongly language disjoint checkers (SLD checkers).

Definition D6: Before the occurrence of any fault, the checker C is language disjoint. For a fault sequence $\langle f1, f2, \cdots, fn\rangle$, a state q', and an input sequence $i2' \in Ic$, let k be the smallest integer for which $\delta'(i1', q_0') = q'$ and such that

$$i1' \cdot i2' \in Ic : w_c(i1', q_0') \cdot w_c^{\langle f1, \cdots fk\rangle}(i2', q') \notin Oc.$$

If there is no such k, let $k = n + 1$. Then C is strongly language disjoint (SLD) with respect to the fault sequence if

$$\forall i' \notin Ic, \forall m(1, 2, \cdots, k-1),$$

$$w_c(i1', q_0') \cdot w_c^{\langle f1, f2, \cdots, fm\rangle}(i', q') \notin Oc.$$

Definition D7: The checker C is strongly language disjoint (SLD) with respect to the fault set F if C is SLD with respect to all fault sequences whose members belong to F.

Checkers that are SLD with respect to class of faults are given by the following property.

Property PY1: If a checker is strongly language disjoint with respect to all sequences of faults fi belonging to a class Cf of fault hypotheses, it is strongly language disjoint with respect to the class Cf of fault hypotheses.

Hypothesis H2: Between the occurrence of any two faults affecting the functional block, a sufficient time elapses such that for any $q \in Q$, for any $i1$ such that $\delta(i1, q_0) = q$ and such that $i1 \cdot i2 \in I_M$, all i_{2m} with respect to each fault f or sequences of faults are applied to this block whenever they exist and no fault occurs in the checker. Between the occurrence of any two faults affecting the checker, a sufficient time elapses such that for any $q \in Q$, for any $i1$ such that $\delta'(i1', q_0') = q'$ and such that $i1' \cdot i2' \in Ic$, all i'_{2m} with respect to each fault f or sequence of faults are applied to the checker whenever they exist, and no fault can occur in the functional block.

Proposition P1: A system composed of an SeSC circuit and an SLD checker achieves the TSC goal under Hypothesis H2.

This proposition is demonstrated in [6].

Comments on SLD checkers may be derived from their definition.

1) Strongly language disjunction ensures the mapping of input sequences included (and, respectively, not included) in the input language to output sequences included (and, respectively, not included) in the output language, even if faults are present inside. The SLD property ensures also the detection of the first fault with respect to which the checker is not redundant with the application of a sequence defined by the input language.

2) The sequentially self-checking property is not verified, but the TSC quality of the system is achieved.

3) The sequentially fault secureness is not necessary for checkers, in general, as the output values will not be used subsequently.

Stronger definitions may be given for SLD checkers.

Definition D8: Before the occurrence of a fault, the checker C is language disjoint. For a fault sequence $\langle f1, f2, \cdots, fn\rangle$, a state q', and an input sequence $i2' \in Iq$, let k be the smallest integer for which $\delta'(i1', q_0') = q'$ and such that

$$i1' \cdot i2' \in Ic : w_c(i1', q_0') \cdot w_c^{\langle f1, f2, \cdots, fk\rangle}(i2', q') \notin Oc.$$

If there is no such k, let $k = n$. Then C is very strongly language disjoint (VSLD) with respect to the fault sequence if

$$\forall i' \notin Ic, \forall m \in \{1, 2, \cdots, k\},$$

$$w_c(i1', q_0') \cdot w_c^{\langle f1, f2, \cdots, fm\rangle}(i', q') \notin Oc.$$

Definition D9: The checker C is very strongly language disjoint (VSLD) with respect to the fault set F if C is VSLD (by Definition D8) with respect to all fault sequences whose members belong to F.

Property PY2: If (but not only if) a checker is VSLD, then it is SLD.

Definition D9 is more restrictive than D7. It (D9) ensures that, even if a fault with respect to which the circuit is not redundant occurs, the language disjoint property is kept (i.e., an output sequence $\in Oc$ will be necessarily generated by an input sequence Ic, while an input sequence $\notin Ic$ will always produce an output sequence $\notin Oc$). This property allows us to consider a less strong hypothesis than H2 with a sequential system in order to achieve the TSC goal which means that it is admitted the occurrence of faults in the functional block (resp., in the checker) preceding the detection of another one, occurred before, in the checker (resp., in the functional block). Hence, "simultaneous" faults in the checker and in the functional block are admitted.

The relationships among the checkers, obtained from the proposed definitions, are depicted in Fig. 2. In this scheme, it may be observed that sequentially self-checking (SeSC) checkers that are sequentially self-testing (SeST), are the most restrictive group. The next one is the group of SeSC checkers which are not SeST. The checkers defined by D9 may be SeST and/or SeFS or just very strongly language disjoint, even in the presence of faults with respect to which the circuit is not redundant. That is why they are a subgroup of SLD checkers described by D7, which is the largest class of checkers.

A. General Considerations

Strongly language disjoint checkers may be designed from traditional logical devices (i.e., using flip-flops and combinational logic or other approaches) or from regular structures, as PLA's and registers, for instance.

But, in a general manner, flip-flops seem not to be a good approach for the design of SLD checkers. Some types of flip-flops are inadequate, as input arrangements with "DON'T CARE" values are allowed; this behavior may hide some stuck-at faults or faults in the input language, for example. There are some difficulties also to ensure the propagation of certain faults to the outputs of flip-flops.

Another traditional approach used for the design of sequential circuits is that of asynchronous circuits, executed from simple logical gates and designed with basis in the fundamental mode (eliminating races). Such a design supposes that the primitive flow table is constructed with the corresponding flow graph and that the states are stable. The design of the complete circuit, from the transition maps, may lead to a good solution but it would be probably too fastidious, considering the designer's point-of-view because 1) a new design will correspond to each new input language (which depends on the SeSC circuit); 2) each design may require a great number of transistors and the analysis of the SLD properties becomes hard and time consuming. It is preferable to choose a solution where predesigned blocks and/or regular structures may be used.

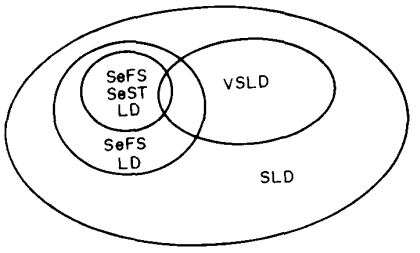

Fig. 2. The checkers in the sequential world.

B. Design of SLD Checkers Based on Regular Structures

SLD checkers cannot be assembled directly from basic cells, as it is the case of strongly code disjoint checkers [15]. But as they are sequential checkers, they may be decomposed internally in combinational and memorization blocks. Then, each one of these blocks may be designed using regular structures such as PLA's, registers, etc. According to the definitions, the outputs of the checker are included in an output language. This language may be particularized into a combinational double-rail code for the error indication. In the following, we will refer to such a particular system, having two double-rail coded outputs.

The inputs of the checker, which are defined by a language, come from the functional part. They are received by the internal combinational block, which executes the logic of the next state function—giving coded states, and the logic of the output function—giving the error indication. The memorization logic feeds back the combinational block and together with the general inputs constitutes the data for the succeeding next states function. Therefore, the complexity of the checker will depend on the sequential property of the input language to be checked, and not on the complexity of the functional system.

We consider a system based on the Mealy model, with a master–slave register and stable inputs (which come from the functional block). When the Moore model is considered, it will be explicitly noted.

The checker, being a finite state accepter, is language disjoint with respect to input language, and Hypothesis H2 may be reformulated. The LD accepter is such that is recognizes the language according to the fact that an input, belonging to an input sequence i, appears when the present state belongs to some set of states.

Let B be the set of codewords, where the codewords are each constitued by the concatenation of an input $x \in X$ (X is the set of input vectors) and a state $q' \in Q'$, such that a word is a codeword iff there is an input sequence $i' \in Ic$ corresponding to this word. The set of states being encoded, examples of noncodewords $y \notin B$ are

- one input $x_i \in X$ that cannot appear in any sequence $i' \in Ic$ when the state is s_j,
- any input $x_k \in X$ when the state $q' \notin Q'$.

Now Hypothesis H2 may be reformulated as follows.

Hypothesis H2': Between the occurrence of any two faults affecting the functional block, a sufficient time elapses such that, for any $q \in Q$, for any $i1$ such that $\delta(i1, q_0) = q$ and such that $i1 \cdot i2 \in I_M$, all i_{2m} with respect to each fault f or sequence of faults that may occur in the functional block are applied, whenever they exist, and no fault occurs in the deterministic finite state accepter (checker). Between the occurrence of any two faults affecting the deterministic finite state accepter, a sufficient time elapses such that all codewords B are applied to the checker (i.e., sequences $i' \in Ic$ applied to the checker are such that all codewords B are applied), and no fault occurs in the functional block.

In the next paragraphs, some propositions referring to internal properties of SLD checkers are given. A complete set of schemes including the respective demonstrations of the necessary internal properties are given in [6].

Proposition P2: For a partitioned design of an SLD checker according to Fig. 3 (Mealy model), the strongly language disjoint

property of the checker (as stated by Definition D6) will be ensured if it is composed of an SFS combinational logic for the next state function realization, an SCD combinational logic for the error indication output function (double-rail code), and an SFS/SCD* memorization logic, assuming Hypothesis H2′ and that any fault can affect only one block.

The SCD property of the output function refers to the input code B (concatenation of the external input with the state code) and to the output code Oc. The SCD and SFS properties of the memorization logic refer to the state code C. Both refer to the large (weak) definition of the combinational SCD checkers. The next state function block is SFS with respect to the output code C.

Proposition P2 is demonstrated in [6].

The star is associated with the denomination SFS/SCD* to mean the use of combinational properties to circuits where time variables are also involved. While load enable and clock signals are not valid, the circuit outputs do not change. When the circuit is enabled by load/clock, the outputs have to be the copy of the input values, or they have to be a noncodeword (then the SFS property is ensured). The noncode inputs have to produce the same noncode outputs, even if faults with respect to which the circuit is redundant have occurred; or if a different noncode output is produced, the concatenation of this one with the external input word has to result in a noncode input word (B) to the SCD output function circuit. Therefore, we will keep the star to assign the delay that will be needed between the input application and the output production (depending on clock period and signal propagation). It is assumed that the clock and load circuitry lines do not fail—or that they are checked separately.

If the Moore model is used similarly to the architecture depicted by Fig. 3, the next-state function block needs to be SCD (in addition to the SFS property).

Other architectures for SLD checkers may be used, if required properties are ensured. The combinational functions may be combined in one single block with SFS/SCD properties, which with an SFS* memorization logic and an internal SCD checker (that checks states code) will compound another proposition. This one and others (including the design of VSLD checkers) are all demonstrated in [6].

In practice, the difficulties for the design of these checkers, as well as the area increasing (if compared to the original circuits) depend on the properties of the internal blocks and on the considered fault hypotheses. The rules for the design of blocks with SFS and SCD properties have been studied by [16] and [17], respectively. In these references, Class I of fault hypotheses is considered, and a complete coverage for the faults included in this class may be achieved if the rules are respected. The complexity of the checker and its additional cost will depend basically on the output language of the functional block; therefore, it is impossible to estimate these parameters as a function of the system complexity.

IV. Conclusion

In this paper, the largest class of checkers is defined—the strongly language disjoint checkers. They check inputs that follow sequential properties and ensure the language disjoint property, even if faults with respect to which the circuit is redundant have already occurred. The SLD checkers, when associated with SeSC (sequentially self-checking) circuits, compose sequential systems that achieve the TSC goal under well-defined fault hypotheses.

The hypotheses related to fault detection suppose the application of a set of sequences between the occurrence of two faults. This set of detecting sequences allows the detection of any occurring fault (excluding the ones with respect to which the circuit is redundant), and independently from the state during which the fault has occurred. Before this detection, the circuit outputs are the same as they would be during normal operation.

Concerning practical applications, the most important conclusion is the possibility of using regular structures for the implementation of these SLD checkers, such as PLA's and register cells. This is interesting as CAD resources may be employed for the design of these checkers. Propositions suggest the use of internal combinational

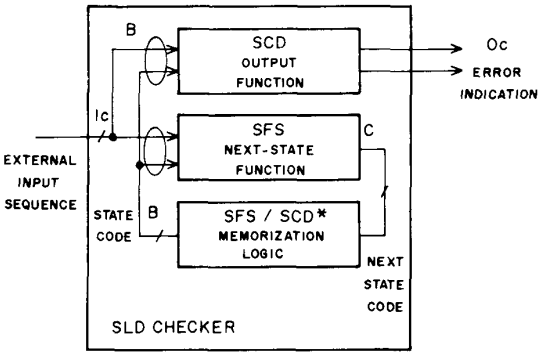

Fig. 3. Proposition for the internal structure of an SLD checker.

blocks with SFS, SCD, and SFS/SCD properties for the design of checkers defined by the large (weak) or the strong definitions, without loss of generality for the global sequential properties.

The last point to be noted is that to be really efficient, the design of SLD checkers, and then the design of SFS, SCD, etc., blocks should be based on realistic fault hypotheses. Such realistic fault hypotheses should include MOS s-on/s-open, shorts, etc., if an MOS technology is the target technology, such as NMOS or CMOS. Rules for the design of such NMOS [16] and CMOS circuits have been studied, and they are for the design of several projects, including large industrial ones.

References

[1] D. A. Anderson, "Design of self-checking digital networks using coding techniques," Univ. Illinois, Res. Rep. 527, Urbana, IL, Sept. 1971.

[2] J. E. Smith and G. Metze, "Strongly fault secure logic networks," *IEEE Trans. Comput.*, vol. C-27, pp. 491-499, June 1978.

[3] M. Nicolaidis, I. Jansch, and B. Courtois, "Strongly code disjoint checkers," in *Dig. 1984 Int. Symp. Fault-Tolerant Comput.*, June 1984.

[4] J. Viaud and R. David, "Sequentially self-checking circuits," in *Dig. 1980 Int. Symp. Fault-Tolerant Comput.*, Oct. 1980.

[5] I. Jansch and B. Courtois, "Strongly language disjoint checkers," in *Dig. 1985 Int. Symp. Fault-Tolerant Comput.*, June 1985.

[6] ——, "Strongly language disjoint checkers," TIM3/IMAG, Res. Rep. RR468, Grenoble, France, Oct. 1984.

[7] R. David and P. Thevenod-Fosse, "Detection transition sequences: Application to random testing of sequential circuits," in *Dig. 1979 Int. Symp. Fault-Tolerant Comput.*, June 1979.

[8] M. Diaz, "Design of totally self-checking and fail-safe machines," in *Dig. 1974 Int. Symp. Fault-Tolerant Comput.*, June 1974.

[9] F. Özguner, "Design of totally self-checking asynchronous and synchronous sequential machines," in *Dig. 1977 Int. Symp. Fault-Tolerant Comput.*, June 1977.

[10] M. Diaz, P. Azema, and J. Ayache, "Unified design of self-checking and fail-safe combinational circuits and sequential machines," *IEEE Trans. Comput.*, vol. C-28, pp. 276-281, Mar. 1979.

[11] Y. Tohma, Y. Ohyama, and R. Sakai, "Realization of fail-safe sequential machines by using a k-out-of-n code," *IEEE Trans. Comput.*, vol. C-20, pp. 1270-1275, Nov. 1971.

[12] H. Chuang and S. Das, "Design of fail-safe sequential machines using separable codes," *IEEE Trans. Comput.*, vol. C-27, pp. 249-252, Mar. 1978.

[13] T. Takaoka and T. Ibaraki, "Fail-safe realization of sequential machines," *Inform. Contr.*, vol. 22, pp. 31-55, Feb. 1973.

[14] J. Meyer and R. Sundstrom, "On-line diagnosis of unrestricted faults," *IEEE Trans. Comput.*, vol. C-24, pp. 468-475, May 1975.

[15] I. Jansch and B. Courtois, "Design of SCD checkers based on analytical fault hypotheses," in *Proc. Euro. Solid-State Circuits Conf. '84*, Sept. 1984.

[16] M. Nicolaidis and B. Courtois, "Layout rules for the design of self-checking circuits," in *Proc. VLSI 85 Conf.*, Aug. 1985.

[17] I. Jansch and B. Courtois, "SCD checkers, cellular checkers and multi-checker structures," TIM3/IMAG, Res. Rep. RR476, Grenoble, France, Nov. 1984.

A New Bit-Serial Systolic Multiplier Over $GF(2^m)$

B. B. ZHOU

Abstract—**A new bit-serial systolic array is developed to compute multiplications over $GF(2^m)$. In contrast to another systolic multiplier designed in [5], this new systolic algorithm allows the input elements to enter a linear systolic array in the same order and the system only requires one control signal.**

Index Terms—**Finite field, finite field multiplication, Massey-Omura multiplier, systolic array, VLSI.**

I. Introduction

Finite field arithmetic operations play a very important role in certain error-correcting codes. Multiplication in $GF(2^m)$ is one among those most complex and time-consuming operations. Recently, two VLSI architectures, systolic architecture [5] and the Massey and Omura algorithm-based pipeline architecture [4], were developed to compute multiplications in $GF(2^m)$. The Massey and Omura algorithm utilizes a normal basis $\{\alpha, \alpha^2, \alpha^4, \cdots, \alpha^{2^{m-1}}\}$ to represent elements in the field $GF(2^m)$, where α is a root of an irreducible polynomial of degree m over $GF(2)$. In this basis representation, the squaring of an element in $GF(2^m)$ is a simple cyclic shift of its binary digits followed by an Exclusive-OR operation. Based on this important property, multiplication for any one product digit requires the same logic circuitry as it does for any other product digit. The advantage of this design is that it takes less area than that required in systolic designs. However, it is much less expandable because irreducible polynomials for different m are not related, but the design only focuses on one particular m each time. Since systolic arrays have the advantages of regularity, modularity, and expandability [1], systolic designs are still worthy of consideration.

The systolic multiplier of [5], which uses a conventional basis, has two disadvantages. One is that it does not allow the two input elements to enter the system in the same order. If this order of the elements is essential, extra registers and control signals are required. The other is that it requires two control signals, which increases the chip area.

A new systolic algorithm for multiplications over $GF(2^m)$ will be discussed here. This algorithm allows the input elements to enter a linear systolic array in the same order and the system only requires one control signal.

II. Algorithm

Using a conventional basis representation, let

$$A=\sum_{k=0}^{m-1} a_k\alpha^k$$

and

$$B=\sum_{k=0}^{m-1} b_k\alpha^k$$

be two elements in $GF(2^m)$,

$$C=\sum_{k=0}^{m-1} c_k\alpha^k$$

Manuscript received December 16, 1985; revised January 23, 1987.

The author is with the Computer Sciences Laboratory, Research School of Physical Sciences, Australian National University, Canberra 2601, Australia.

IEEE Log Number 8716248.

be the product of A and B, and

$$F=\sum_{k=0}^{m} f_k\alpha^k \qquad (f_m=1)$$

be the irreducible polynomial.

The equation $F = 0$, or $\alpha_m = \sum_{k=0}^{m-1} f_k\alpha^k$ is used to reduce the product $C = AB$ to a polynomial of degree less than m [2].

C can be written as follows.

$$\begin{aligned} C&=AB\\ &=A\sum_{k=0}^{m-1} b_k\alpha^k\\ &=(\cdots((Ab_{m-1}\alpha+Ab_{m-2})\alpha+Ab_{m-3})\alpha+\cdots)\alpha_1+Ab_0 \end{aligned}$$

or

$$\begin{aligned} &C^{(0)}=Ab_{m-1},\\ &C^{(1)}=Ab_{m-1}\alpha+Ab_{m-2}=C^{(0)}\alpha+Ab_{m-2},\\ &\quad\vdots\\ &C^{(i)}=C^{(i-1)}\alpha+Ab_{m-1-i},\\ &\quad\vdots\\ &C=C^{(m-1)}=C^{(m-2)}\alpha+Ab_0. \end{aligned} \tag{1}$$

Using the fact that

$$\alpha^m=\sum_{k=0}^{m-1} f_k\alpha^k,$$

$C^{(i)}$ in (1) can be reduced to a polynomial of degree less than m.

$C^{(0)}$ in (1) can be rewritten as

$$C^{(0)}=Ab_{m-1}=\sum_{k=0}^{m-1} a_kb_{m-1}\alpha^k=\sum_{k=0}^{m-1} C_k^{(0)}\alpha^k \tag{2}$$

where $C_k^{(0)} = a_kb_{m-1}$.

Suppose

$$C^{(i-1)}=\sum_{k=0}^{m-1} C_k^{(i-1)}\alpha^k,$$

which has been reduced to a polynomial of degree less than m. Then

$$\begin{aligned} C^{(i)}&=C^{(i-1)}\alpha+Ab_{m-1-i}\\ &=\sum_{k=0}^{m-1} C_k^{(i-1)}\alpha^{k+1}+Ab_{m-1-i}\\ &=C_{m-1}^{(i-1)}\alpha^m+\sum_{k=0}^{m-2} C_k^{(i-1)}\alpha^{k+1}+Ab_{m-1-i}\\ &=\sum_{k=0}^{m-1} C_{m-1}^{(i-1)}f_k\alpha^k+\sum_{k=1}^{m-1} C_{k-1}^{(i-1)}\alpha^k+\sum_{k=0}^{m-1} a_kb_{m-1-i}\alpha^k\\ &=(C_{m-1}^{(i-1)}f_0+a_0b_{m-1-i})\\ &\quad+\sum_{k=1}^{m-1}(C_{m-1}^{(i-1)}f_k+a_kb_{m-1-i}+C_{k-1}^{(i-1)})\alpha^k\\ &=\sum_{k=0}^{m-1} C_k^{(i)}\alpha^k. \end{aligned} \tag{3}$$